# MAX II Device Handbook

101 Innovation Drive
San Jose, CA 95134
www.altera.com

MII5V1-3.3

Copyright © 2009 Altera Corporation. All rights reserved. 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.

I.S. EN ISO 9001

# Contents

**Chapter Revision Dates** . . . . . xi

**About this Handbook** . . . . . About–xiii
How to Contact Altera . . . . . 1–xiii
Typographic Conventions . . . . . 1–xiii

**Section I. MAX II Device Family Data Sheet**
Revision History . . . . . I–1

**Chapter 1. Introduction**
Introduction . . . . . 1–1
Features . . . . . 1–1
Referenced Documents . . . . . 1–4
Document Revision History . . . . . 1–4

**Chapter 2. MAX II Architecture**
Introduction . . . . . 2–1
Functional Description . . . . . 2–1
Logic Array Blocks . . . . . 2–4
LAB Interconnects . . . . . 2–4
LAB Control Signals . . . . . 2–5
Logic Elements . . . . . 2–6
LUT Chain and Register Chain . . . . . 2–8
addnsub Signal . . . . . 2–8
LE Operating Modes . . . . . 2–8
Normal Mode . . . . . 2–9
Dynamic Arithmetic Mode . . . . . 2–9
Carry-Select Chain . . . . . 2–10
Clear and Preset Logic Control . . . . . 2–12
MultiTrack Interconnect . . . . . 2–12
Global Signals . . . . . 2–16
User Flash Memory Block . . . . . 2–18
UFM Storage . . . . . 2–19
Internal Oscillator . . . . . 2–20
Program, Erase, and Busy Signals . . . . . 2–20
Auto-Increment Addressing . . . . . 2–20
Serial Interface . . . . . 2–20
UFM Block to Logic Array Interface . . . . . 2–20
MultiVolt Core . . . . . 2–22

I/O Structure . . . . . 2–23
Fast I/O Connection . . . . . 2–23
I/O Blocks . . . . . 2–24
I/O Standards and Banks . . . . . 2–26
PCI Compliance . . . . . 2–28
Schmitt Trigger . . . . . 2–29
Output Enable Signals . . . . . 2–29
Programmable Drive Strength . . . . . 2–29
Slew-Rate Control . . . . . 2–30
Open-Drain Output . . . . . 2–30
Programmable Ground Pins . . . . . 2–30
Bus Hold . . . . . 2–31
Programmable Pull-Up Resistor . . . . . 2–31
Programmable Input Delay . . . . . 2–31
MultiVolt I/O Interface . . . . . 2–31
Referenced Documents . . . . . 2–32
Document Revision History . . . . . 2–33

**Chapter 3. JTAG and In-System Programmability**
Introduction . . . . . 3–1
IEEE Std. 1149.1 (JTAG) Boundary-Scan Support . . . . . 3–1
JTAG Block . . . . . 3–3
Parallel Flash Loader . . . . . 3–3
In System Programmability . . . . . 3–4
IEEE 1532 Support . . . . . 3–5
Jam Standard Test and Programming Language (STAPL) . . . . . 3–5
Programming Sequence . . . . . 3–5
UFM Programming . . . . . 3–6
In-System Programming Clamp . . . . . 3–6
Real-Time ISP . . . . . 3–7
Design Security . . . . . 3–7
Programming with External Hardware . . . . . 3–7
Referenced Documents . . . . . 3–7
Document Revision History . . . . . 3–8

**Chapter 4. Hot Socketing and Power-On Reset in MAX II Devices**
Introduction . . . . . 4–1
MAX II Hot-Socketing Specifications . . . . . 4–1
Devices Can Be Driven before Power-Up . . . . . 4–1
I/O Pins Remain Tri-Stated during Power-Up . . . . . 4–2
Signal Pins Do Not Drive the $V_{CCIO}$ or $V_{CCINT}$ Power Supplies . . . . . 4–2
AC and DC Specifications . . . . . 4–2
Hot Socketing Feature Implementation in MAX II Devices . . . . . 4–2
Power-On Reset Circuitry . . . . . 4–5
Power-Up Characteristics . . . . . 4–6
Referenced Documents . . . . . 4–8
Document Revision History . . . . . 4–8

**Chapter 5. DC and Switching Characteristics**
Introduction . . . . . 5–1

Operating Conditions . . . . . 5–1
Absolute Maximum Ratings . . . . . 5–1
Recommended Operating Conditions . . . . . 5–2
Programming/Erasure Specifications . . . . . 5–3
DC Electrical Characteristics . . . . . 5–3
Output Drive Characteristics . . . . . 5–5
I/O Standard Specifications . . . . . 5–5
Bus Hold Specifications . . . . . 5–7
Power-Up Timing . . . . . 5–8
Power Consumption . . . . . 5–8
Timing Model and Specifications . . . . . 5–8
Preliminary and Final Timing . . . . . 5–9
Performance . . . . . 5–10
Internal Timing Parameters . . . . . 5–11
External Timing Parameters . . . . . 5–18
External Timing I/O Delay Adders . . . . . 5–21
Maximum Input and Output Clock Rates . . . . . 5–24
JTAG Timing Specifications . . . . . 5–25
Referenced Documents . . . . . 5–26
Document Revision History . . . . . 5–27

**Chapter 6. Reference and Ordering Information**
Software . . . . . 6–1
Device Pin-Outs . . . . . 6–1
Ordering Information . . . . . 6–1
Referenced Documents . . . . . 6–2
Document Revision History . . . . . 6–2

**Section II. PCB Layout Guidelines**
Revision History . . . . . II–1

**Chapter 7. Package Information**
Introduction . . . . . 7–1
Board Decoupling Guidelines . . . . . 7–1
Device and Package Cross Reference . . . . . 7–1
Thermal Resistance . . . . . 7–2
Package Outlines . . . . . 7–3
68-Pin Micro FineLine Ball-Grid Array (MBGA) – Wire Bond . . . . . 7–3
100-Pin Plastic Thin Quad Flat Pack (TQFP) . . . . . 7–5
100-Pin Micro FineLine Ball-Grid Array (MBGA) . . . . . 7–7
100-Pin FineLine Ball-Grid Array (FBGA) . . . . . 7–8
144-Pin Plastic Thin Quad Flat Pack (TQFP) . . . . . 7–10
144-Pin Micro FineLine Ball-Grid Array (MBGA) – Wire Bond . . . . . 7–12
256-Pin Micro FineLine Ball-Grid Array (MBGA) . . . . . 7–13
256-Pin FineLine Ball-Grid Array (FBGA) . . . . . 7–15
324-Pin FineLine Ball-Grid Array (FBGA) . . . . . 7–16
Document Revision History . . . . . 7–18

**Chapter 8. Using MAX II Devices in Multi-Voltage Systems**
Introduction . . . . . 8–1
I/O Standards . . . . . 8–2
MultiVolt Core and I/O Operation . . . . . 8–3
5.0-V Device Compatibility . . . . . 8–3

Recommended Operating Condition for 5.0-V Compatibility . . . . . 8–7
Hot Socketing . . . . . 8–8
Power-Up Sequencing . . . . . 8–8
Power-On Reset . . . . . 8–8
Conclusion . . . . . 8–8
Referenced Documents . . . . . 8–9
Document Revision History . . . . . 8–9

**Section III. User Flash Memory**
Revision History . . . . . III–1

**Chapter 9. Using User Flash Memory in MAX II Devices**
Introduction . . . . . 9–1
UFM Array Description . . . . . 9–1
Memory Organization Map . . . . . 9–1
Using and Accessing UFM Storage . . . . . 9–2
UFM Functional Description . . . . . 9–3
UFM Address Register . . . . . 9–5
UFM Data Register . . . . . 9–5
UFM Program/Erase Control Block . . . . . 9–6
Oscillator . . . . . 9–6
Instantiating the Oscillator without the UFM . . . . . 9–7
UFM Operating Modes . . . . . 9–9
Read/Stream Read . . . . . 9–9
Program . . . . . 9–10
Erase . . . . . 9–11
Programming and Reading the UFM with JTAG . . . . . 9–12
Jam Files . . . . . 9–12
Jam Players . . . . . 9–12
Software Support for UFM Block . . . . . 9–13
Inter-Integrated Circuit . . . . . 9–14
I²C Protocol . . . . . 9–14
Device Addressing . . . . . 9–16
Byte Write Operation . . . . . 9–16
Page Write Operation . . . . . 9–17
Write Protection . . . . . 9–17
Erase Operation . . . . . 9–18
Read Operation . . . . . 9–20
ALTUFM I²C Interface Timing Specification . . . . . 9–22
Instantiating the I2C Interface Using the Quartus II altufm Megafunction . . . . . 9–22
Serial Peripheral Interface . . . . . 9–24
Opcodes . . . . . 9–25
ALTUFM SPI Timing Specification . . . . . 9–34
Instantiating SPI Using Quartus II altufm Megafunction . . . . . 9–35
Parallel Interface . . . . . 9–36
ALTUFM Parallel Interface Timing Specification . . . . . 9–37
Instantiating Parallel Interface Using Quartus II altufm Megafunction . . . . . 9–38
None (Altera Serial Interface) . . . . . 9–39
Instantiating None Using Quartus II altufm Megafunction . . . . . 9–39
Creating Memory Content File . . . . . 9–40
Memory Initialization for the altufm_parallel Megafunction . . . . . 9–43
Memory Initialization for the altufm_spi Megafunction . . . . . 9–43
Memory Initialization for the altufm_i2c Megafunction . . . . . 9–44

Simulation Parameters . . . . . 9–46
Conclusion . . . . . 9–46
Referenced Documents . . . . . 9–47
Document Revision History . . . . . 9–47

**Chapter 10. Replacing Serial EEPROMs with MAX II User Flash Memory**
Introduction . . . . . 10–1
Design Considerations . . . . . 10–1
List of Vendors and Devices . . . . . 10–2
Conclusion . . . . . 10–11
Referenced Documents . . . . . 10–11
Document Revision History . . . . . 10–12

**Section IV. In-System Programmability**
Revision History . . . . . IV–1

**Chapter 11. In-System Programmability Guidelines for MAX II Devices**
Introduction . . . . . 11–1
General ISP Guidelines . . . . . 11–1
Operating Conditions . . . . . 11–1
ISP Voltage . . . . . 11–1
Input Voltages . . . . . 11–2
UFM Operations During In-System Programming . . . . . 11–2
Interrupting In-System Programming . . . . . 11–3
MultiVolt Devices and Power-Up Sequences . . . . . 11–3
$V_{CCIO}$ Powered before $V_{CCINT}$ . . . . . 11–3
I/O Pins Tri-Stated during In-System Programming . . . . . 11–3
Pull-Up and Pull-Down of JTAG Pins During In-System Programming . . . . . 11–4
IEEE Std. 1149.1 Signals . . . . . 11–4
TCK Signal . . . . . 11–4
Programming via a Download Cable . . . . . 11–5
Disabling IEEE Std. 1149.1 Circuitry . . . . . 11–5
Working with Different Voltage Levels . . . . . 11–6
Sequential versus Concurrent Programming . . . . . 11–6
Sequential Programming . . . . . 11–6
Concurrent Programming . . . . . 11–6
ISP Troubleshooting Guidelines . . . . . 11–7
Invalid ID and Unrecognized Device Messages . . . . . 11–7
Download Cable Connected Incorrectly . . . . . 11–7
TDO Is Not Connected . . . . . 11–8
Incomplete JTAG Chain . . . . . 11–8
Noisy TCK Signal . . . . . 11–8
Jam Player Ported Incorrectly . . . . . 11–8
Troubleshooting Tips . . . . . 11–8
Verify the JTAG Chain Continuity . . . . . 11–8
Check the $V_{CC}$ Level of the Board During In-System Programming . . . . . 11–9
Power-Up Problems . . . . . 11–9
Random Signals on JTAG Pins . . . . . 11–9
Software Issues . . . . . 11–9
ISP via Embedded Processors . . . . . 11–9
Processor and Memory Requirements . . . . . 11–9
Porting the Jam Player . . . . . 11–10
ISP via In-Circuit Testers . . . . . 11–10

Conclusion . . . . . 11–10
Referenced Documents . . . . . 11–10
Document Revision History . . . . . 11–11

**Chapter 12. Real-Time ISP and ISP Clamp for MAX II Devices**
Introduction . . . . . 12–1
Real-Time ISP . . . . . 12–1
How Real-Time ISP Works . . . . . 12–1
Real-Time ISP with the Quartus II Software . . . . . 12–2
Real-Time ISP with Jam and JBC Players . . . . . 12–4
ISP Clamp . . . . . 12–4
How ISP Clamp Works . . . . . 12–5
Using ISP Clamp in the Quartus II Software . . . . . 12–5
Using the IPS File . . . . . 12–6
Defining the Pin States in Assignment Editor . . . . . 12–9
ISP Clamp with Jam/JBC Files . . . . . 12–10
Conclusion . . . . . 12–10
Referenced Documents . . . . . 12–10
Document Revision History . . . . . 12–11

**Chapter 13. IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices**
Introduction . . . . . 13–1
IEEE Std. 1149.1 BST Architecture . . . . . 13–2
IEEE Std. 1149.1 Boundary-Scan Register . . . . . 13–3
Boundary-Scan Cells of a MAX II Device I/O Pin . . . . . 13–4
JTAG Pins and Power Pins . . . . . 13–5
IEEE Std. 1149.1 BST Operation Control . . . . . 13–6
SAMPLE/PRELOAD Instruction Mode . . . . . 13–9
EXTEST Instruction Mode . . . . . 13–11
BYPASS Instruction Mode . . . . . 13–13
IDCODE Instruction Mode . . . . . 13–14
USERCODE Instruction Mode . . . . . 13–14
CLAMP Instruction Mode . . . . . 13–14
HIGHZ Instruction Mode . . . . . 13–14
I/O Voltage Support in JTAG Chain . . . . . 13–15
BST for Programmed Devices . . . . . 13–15
Disabling IEEE Std. 1149.1 BST Circuitry . . . . . 13–16
Guidelines for IEEE Std. 1149.1 Boundary-Scan Testing . . . . . 13–16
Boundary-Scan Description Language (BSDL) Support . . . . . 13–17
Conclusion . . . . . 13–17
Referenced Documents . . . . . 13–17
Document Revision History . . . . . 13–18

**Chapter 14. Using Jam STAPL for ISP via an Embedded Processor**
Introduction . . . . . 14–1
Embedded Systems . . . . . 14–1
Connecting the JTAG Chain to the Embedded Processor . . . . . 14–1
Example Interface PLD Design . . . . . 14–2
Board Layout . . . . . 14–3
TCK Signal Trace Protection and Integrity . . . . . 14–4
Pull-Down Resistors on TCK . . . . . 14–4
JTAG Signal Traces . . . . . 14–4
External Resistors . . . . . 14–4

Software Development . . . . . 14–4
Jam Files (.jam and .jbc) . . . . . 14–5
ASCII Text Files (.jam) . . . . . 14–5
Jam Byte-Code Files (.jbc) . . . . . 14–5
Generating Jam Files . . . . . 14–5
Using Jam Files with the MAX II User Flash Memory Block . . . . . 14–7
Jam Players . . . . . 14–7
Jam Player Compatibility . . . . . 14–7
The Jam STAPL Byte-Code Player . . . . . 14–7
Porting the Jam STAPL Byte-Code Player . . . . . 14–8
Jam STAPL Byte-Code Player Memory Usage . . . . . 14–11
Updating Devices Using Jam . . . . . 14–14
MAX II Jam/JBC Actions and Procedure Commands . . . . . 14–15
Running the Jam STAPL Byte-Code Player . . . . . 14–17
Conclusion . . . . . 14–18
Referenced Documents . . . . . 14–18
Document Revision History . . . . . 14–19

**Chapter 15. Using the Agilent 3070 Tester for In-System Programming**
Introduction . . . . . 15–1
New PLD Product for Agilent 3070 . . . . . 15–1
Device Support . . . . . 15–1
Agilent 3070 Development Flow without the PLD ISP Software . . . . . 15–2
Step 1: Create a PCB and Test Fixture . . . . . 15–3
Creating the PCB . . . . . 15–3
Creating the Fixture . . . . . 15–3
Step 2: Create a Serial Vector Format File . . . . . 15–4
Step 3: Convert SVF Files to PCF Files . . . . . 15–5
Step 4: Create Executable Tests from Files . . . . . 15–5
Create the Library for the Target Device or Scan Chain . . . . . 15–5
Run the Test Consultant . . . . . 15–6
Create Digital Tests . . . . . 15–6
Create the Wirelist Information for the Tests . . . . . 15–6
Modify the Test Plan . . . . . 15–6
Step 5: Compile the Executable Tests . . . . . 15–7
Step 6: Debug the Test . . . . . 15–7
Development Flow for Agilent 3070 with PLD ISP Software . . . . . 15–8
Programming Times . . . . . 15–10
Guidelines . . . . . 15–10
Conclusion . . . . . 15–11
Referenced Documents . . . . . 15–11
Document Revision History . . . . . 15–11

**Section V. Design Considerations**
Revision History . . . . . V–1

**Chapter 16. Understanding Timing in MAX II Devices**
Introduction . . . . . 16–1
External Timing Parameters . . . . . 16–1
Internal Timing Parameters . . . . . 16–2
Internal Timing Parameters for MAX II UFM . . . . . 16–3
Timing Models . . . . . 16–4

Calculating Timing Delays . . . . . 16–5
Setup and Hold Time from an I/O Data and Clock Input . . . . . 16–6
Programmable Input Delay . . . . . 16–7
Timing Model versus Quartus II Timing Analyzer . . . . . 16–7
Conclusion . . . . . 16–8
Referenced Documents . . . . . 16–8
Document Revision History . . . . . 16–8

## Chapter 17. Understanding and Evaluating Power in MAX II Devices

Introduction . . . . . 17–1
Power in MAX II Devices . . . . . 17–1
MAX II Power Estimation Using the PowerPlay Early Power Estimator . . . . . 17–3
PowerPlay Early Power Estimator Inputs . . . . . 17–3
Input Parameters . . . . . 17–4
Clock Section . . . . . 17–5
Logic Section . . . . . 17–6
UFM Section . . . . . 17–8
I/O Section . . . . . 17–8
Other Input Information . . . . . 17–11
Set Toggle % . . . . . 17–11
Reset . . . . . 17–12
Importing the Quartus II Early Power Estimator File . . . . . 17–12
Power Estimation Summary . . . . . 17–13
Power . . . . . 17–13
Thermal Analysis . . . . . 17–14
Power Supply Current . . . . . 17–15
Power Saving Techniques . . . . . 17–15
Conclusion . . . . . 17–16
Referenced Documents . . . . . 17–16
Document Revision History . . . . . 17–17

# Chapter Revision Dates

The chapters in this book, *MAX II Device Handbook*, were revised on the following dates. Where chapters or groups of chapters are available separately, part numbers are listed.

Chapter 1 Introduction
Revised: *August 2009*
Part Number: *MII51001-1.9*

Chapter 2 MAX II Architecture
Revised: *October 2008*
Part Number: *MII51002-2.2*

Chapter 3 JTAG and In-System Programmability
Revised: *October 2008*
Part Number: *MII51003-1.6*

Chapter 4 Hot Socketing and Power-On Reset in MAX II Devices
Revised: *October 2008*
Part Number: *MII51004-2.1*

Chapter 5 DC and Switching Characteristics
Revised: *August 2009*
Part Number: *MII51005-2.5*

Chapter 6 Reference and Ordering Information
Revised: *August 2009*
Part Number: *MII51006-1.6*

Chapter 7 Package Information
Revised: *October 2008*
Part Number: *MII51007-2.1*

Chapter 8 Using MAX II Devices in Multi-Voltage Systems
Revised: *October 2008*
Part Number: *MII51009-1.7*

Chapter 9 Using User Flash Memory in MAX II Devices
Revised: *October 2008*
Part Number: *MII51010-1.8*

Chapter 10 Replacing Serial EEPROMs with MAX II User Flash Memory
Revised: *October 2008*
Part Number: *MII51012-1.5*

Chapter 11 In-System Programmability Guidelines for MAX II Devices
Revised: *October 2008*
Part Number: *MII51013-1.7*

Chapter 12 Real-Time ISP and ISP Clamp for MAX II Devices

Revised: *October 2008*
Part Number: *MII51019-1.6*

Chapter 13 IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices
Revised: *October 2008*
Part Number: *MII51014-1.7*

Chapter 14 Using Jam STAPL for ISP via an Embedded Processor
Revised: *October 2008*
Part Number: *MII51015-1.8*

Chapter 15 Using the Agilent 3070 Tester for In-System Programming
Revised: *October 2008*
Part Number: *MII51016-1.5*

Chapter 16 Understanding Timing in MAX II Devices
Revised: *October 2008*
Part Number: *MII51017-2.1*

Chapter 17 Understanding and Evaluating Power in MAX II Devices
Revised: *October 2008*
Part Number: *MII51018-2.1*

This handbook provides comprehensive information about the Altera® MAX® II family of devices.

## How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

| Contact *(Note 1)* | Contact Method | Address |
|---|---|---|
| Technical support | Website | www.altera.com/support |
| Technical training | Website | www.altera.com/training |
| | Email | custrain@altera.com |
| Altera literature services | Email | literature@altera.com |
| Non-technical support (General) | Email | nacomp@altera.com |
| (Software Licensing) | Email | authorization@altera.com |

**Note:**

(1) You can also contact your local Altera sales office or sales representative.

## Typographic Conventions

This document uses the typographic conventions shown in the following table.

| Visual Cue | Meaning |
|---|---|
| **Bold Type with Initial Capital Letters** | Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box. |
| **bold type** | External timing parameters, directory names, project names, disk drive names, file names, file name extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\qdesigns** directory, **d:** drive, **chiptrip.gdf** file. |
| *Italic Type with Initial Capital Letters* | Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.* |
| *Italic type* | Internal timing parameters and variables are shown in italic type. Examples: $t_{PIA}$, $n + 1$.<br>Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file. |
| Initial Capital Letters | Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu. |
| "Subheading Title" | References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: "Typographic Conventions." |

| Visual Cue | Meaning |
|---|---|
| `Courier type` | Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix `n`, e.g., `resetn`.<br><br>Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier. |
| 1., 2., 3., and<br>a., b., c., etc. | Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure. |
| ■ ■ | Bullets are used in a list of items when the sequence of the items is not important. |
| ✓ | The checkmark indicates a procedure that consists of one step only. |
| ☞ | The hand points to information that requires special attention. |
| CAUTION | A caution calls attention to a condition or possible situation that can damage or destroy the product or the user's work. |
| WARNING | A warning calls attention to a condition or possible situation that can cause injury to the user. |
| ↵ | The angled arrow indicates you should press the Enter key. |
| 👣 | The feet direct you to more information on a particular topic. |

# Section I. MAX II Device Family Data Sheet

This section provides designers with the data sheet specifications for MAX® II devices. The chapters contain feature definitions of the internal architecture, Joint Test Action Group (JTAG) and in-system programmability (ISP) information, DC operating conditions, AC timing parameters, and ordering information for MAX II devices.

This section includes the following chapters:

- Chapter 1, Introduction
- Chapter 2, MAX II Architecture
- Chapter 3, JTAG and In-System Programmability
- Chapter 4, Hot Socketing and Power-On Reset in MAX II Devices
- Chapter 5, DC and Switching Characteristics
- Chapter 6, Reference and Ordering Information

## Revision History

Refer to each chapter for its own specific revision history. For information about when each chapter was updated, refer to the Chapter Revision Dates section, which appears in the complete handbook.

# 1. Introduction

MII51001-1.9

## Introduction

The MAX® II family of instant-on, non-volatile CPLDs is based on a 0.18-µm, 6-layer-metal-flash process, with densities from 240 to 2,210 logic elements (LEs) (128 to 2,210 equivalent macrocells) and non-volatile storage of 8 Kbits. MAX II devices offer high I/O counts, fast performance, and reliable fitting versus other CPLD architectures. Featuring MultiVolt core, a user flash memory (UFM) block, and enhanced in-system programmability (ISP), MAX II devices are designed to reduce cost and power while providing programmable solutions for applications such as bus bridging, I/O expansion, power-on reset (POR) and sequencing control, and device configuration control.

## Features

The MAX II CPLD has the following features:

- Low-cost, low-power CPLD
- Instant-on, non-volatile architecture
- Standby current as low as 25 µA
- Provides fast propagation delay and clock-to-output times
- Provides four global clocks with two clocks available per logic array block (LAB)
- UFM block up to 8 Kbits for non-volatile storage
- MultiVolt core enabling external supply voltages to the device of either 3.3 V/2.5 V or 1.8 V
- MultiVolt I/O interface supporting 3.3-V, 2.5-V, 1.8-V, and 1.5-V logic levels
- Bus-friendly architecture including programmable slew rate, drive strength, bus-hold, and programmable pull-up resistors
- Schmitt triggers enabling noise tolerant inputs (programmable per pin)
- I/Os are fully compliant with the Peripheral Component Interconnect Special Interest Group (PCI SIG) PCI Local Bus Specification, Revision 2.2 for 3.3-V operation at 66 MHz
- Supports hot-socketing
- Built-in Joint Test Action Group (JTAG) boundary-scan test (BST) circuitry compliant with IEEE Std. 1149.1-1990
- ISP circuitry compliant with IEEE Std. 1532

Table 1–1 shows the MAX II family features.

**Table 1–1.** MAX II Family Features

| Feature | EPM240<br>EPM240G | EPM570<br>EPM570G | EPM1270<br>EPM1270G | EPM2210<br>EPM2210G | EPM240Z | EPM570Z |
|---|---|---|---|---|---|---|
| LEs | 240 | 570 | 1,270 | 2,210 | 240 | 570 |
| Typical Equivalent Macrocells | 192 | 440 | 980 | 1,700 | 192 | 440 |
| Equivalent Macrocell Range | 128 to 240 | 240 to 570 | 570 to 1,270 | 1,270 to 2,210 | 128 to 240 | 240 to 570 |
| UFM Size (bits) | 8,192 | 8,192 | 8,192 | 8,192 | 8,192 | 8,192 |
| Maximum User I/O pins | 80 | 160 | 212 | 272 | 80 | 160 |
| $t_{PD1}$ (ns) *(1)* | 4.7 | 5.4 | 6.2 | 7.0 | 7.5 | 9.0 |
| $f_{CNT}$ (MHz) *(2)* | 304 | 304 | 304 | 304 | 152 | 152 |
| $t_{SU}$ (ns) | 1.7 | 1.2 | 1.2 | 1.2 | 2.3 | 2.2 |
| $t_{CO}$ (ns) | 4.3 | 4.5 | 4.6 | 4.6 | 6.5 | 6.7 |

**Notes to Table 1–1:**

(1) $t_{PD1}$ represents a pin-to-pin delay for the worst case I/O placement with a full diagonal path across the device and combinational logic implemented in a single LUT and LAB that is adjacent to the output pin.

(2) The maximum frequency is limited by the I/O standard on the clock input pin. The 16-bit counter critical delay will run faster than this number.

For more information about equivalent macrocells, refer to the *MAX II Logic Element to Macrocell Conversion Methodology* white paper.

MAX II and MAX IIG devices are available in three speed grades: –3, –4, and –5, with –3 being the fastest. Similarly, MAX IIZ devices are available in three speed grades: –6, –7, and –8, with –6 being the fastest. These speed grades represent the overall relative performance, not any specific timing parameter. For propagation delay timing numbers within each speed grade and density, refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*.

Table 1–2 shows MAX II device speed-grade offerings.

**Table 1–2.** MAX II Speed Grades

| Device | Speed Grade | | | | | |
|---|---|---|---|---|---|---|
| | –3 | –4 | –5 | –6 | –7 | –8 |
| EPM240<br>EPM240G | ✓ | ✓ | ✓ | — | — | — |
| EPM570<br>EPM570G | ✓ | ✓ | ✓ | — | — | — |
| EPM1270<br>EPM1270G | ✓ | ✓ | ✓ | — | — | — |
| EPM2210<br>EPM2210G | ✓ | ✓ | ✓ | — | — | — |
| EPM240Z | — | — | — | ✓ | ✓ | ✓ |
| EPM570Z | — | — | — | ✓ | ✓ | ✓ |

MAX II devices are available in space-saving FineLine BGA, Micro FineLine BGA, and thin quad flat pack (TQFP) packages (refer to Table 1–3 and Table 1–4). MAX II devices support vertical migration within the same package (for example, you can migrate between the EPM570, EPM1270, and EPM2210 devices in the 256-pin FineLine BGA package). Vertical migration means that you can migrate to devices whose dedicated pins and JTAG pins are the same and power pins are subsets or supersets for a given package across device densities. The largest density in any package has the highest number of power pins; you must lay out for the largest planned density in a package to provide the necessary power pins for migration. For I/O pin migration across densities, cross reference the available I/O pins using the device pin-outs for all planned densities of a given package type to identify which I/O pins can be migrated. The Quartus® II software can automatically cross-reference and place all pins for you when given a device migration list.

**Table 1–3.** MAX II Packages and User I/O Pins

| Device | 68-Pin Micro FineLine BGA *(1)* | 100-Pin Micro FineLine BGA *(1)* | 100-Pin FineLine BGA | 100-Pin TQFP | 144-Pin TQFP | 144-Pin Micro FineLine BGA *(1)* | 256-Pin Micro FineLine BGA *(1)* | 256-Pin FineLine BGA | 324-Pin FineLine BGA |
|---|---|---|---|---|---|---|---|---|---|
| EPM240<br>EPM240G | — | 80 | 80 | 80 | — | — | — | — | — |
| EPM570<br>EPM570G | — | 76 | 76 | 76 | 116 | — | 160 | 160 | — |
| EPM1270<br>EPM1270G | — | — | — | — | 116 | — | 212 | 212 | — |
| EPM2210<br>EPM2210G | — | — | — | — | — | — | — | 204 | 272 |
| EPM240Z | 54 | 80 | — | — | — | — | — | — | — |
| EPM570Z | — | 76 | — | — | — | 116 | 160 | — | — |

**Note to Table 1–3:**

(1) Packages available in lead-free versions only.

**Table 1–4.** MAX II TQFP, FineLine BGA, and Micro FineLine BGA Package Sizes

| Package | 68-Pin Micro FineLine BGA | 100-Pin Micro FineLine BGA | 100-Pin FineLine BGA | 100-Pin TQFP | 144-Pin TQFP | 144-Pin Micro FineLine BGA | 256-Pin Micro FineLine BGA | 256-Pin FineLine BGA | 324-Pin FineLine BGA |
|---|---|---|---|---|---|---|---|---|---|
| Pitch (mm) | 0.5 | 0.5 | 1 | 0.5 | 0.5 | 0.5 | 0.5 | 1 | 1 |
| Area (mm2) | 25 | 36 | 121 | 256 | 484 | 49 | 121 | 289 | 361 |
| Length × width (mm × mm) | 5 × 5 | 6 × 6 | 11 × 11 | 16 × 16 | 22 × 22 | 7 × 7 | 11 × 11 | 17 × 17 | 19 × 19 |

MAX II devices have an internal linear voltage regulator which supports external supply voltages of 3.3 V or 2.5 V, regulating the supply down to the internal operating voltage of 1.8 V. MAX IIG and MAX IIZ devices only accept 1.8 V as the external supply voltage. MAX IIZ devices are pin-compatible with MAX IIG devices in the 100-pin Micro FineLine BGA and 256-pin Micro FineLine BGA packages. Except for external supply voltage requirements, MAX II and MAX II G devices have identical pin-outs and timing specifications. Table 1–5 shows the external supply voltages supported by the MAX II family.

**Table 1–5.** MAX II External Supply Voltages

| Devices | EPM240<br>EPM570<br>EPM1270<br>EPM2210 | EPM240G<br>EPM570G<br>EPM1270G<br>EPM2210G<br>EPM240Z<br>EPM570Z *(1)* |
|---|---|---|
| MultiVolt core external supply voltage ($V_{CCINT}$) *(2)* | 3.3 V, 2.5 V | 1.8 V |
| MultiVolt I/O interface voltage levels ($V_{CCIO}$) | 1.5 V, 1.8 V, 2.5 V, 3.3 V | 1.5 V, 1.8 V, 2.5 V, 3.3 V |

**Notes to Table 1–5:**

(1) MAX IIG and MAX IIZ devices only accept 1.8 V on their VCCINT pins. The 1.8-V $V_{CCINT}$ external supply powers the device core directly.

(2) MAX II devices operate internally at 1.8 V.

# Referenced Documents

This chapter references the following documents:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *MAX II Logic Element to Macrocell Conversion Methodology* white paper

# Document Revision History

Table 1–6 shows the revision history for this chapter.

**Table 1–6.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| August 2009, version 1.9 | ■ Updated Table 1–2. | Added information for speed grade –8 |
| October 2008, version 1.8 | ■ Updated "Introduction" section.<br>■ Updated new Document Format. | — |
| December 2007, version1.7 | ■ Updated Table 1–1 through Table 1–5.<br>■ Added "Referenced Documents" section. | Updated document with MAX IIZ information. |
| December 2006, version 1.6 | ■ Added document revision history. | — |
| August 2006, version 1.5 | ■ Minor update to features list. | — |
| July 2006, version 1.4 | ■ Minor updates to tables. | — |

**Table 1–6.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| June 2005, version 1.3 | ■ Updated timing numbers in Table 1-1. | — |
| December 2004, version 1.2 | ■ Updated timing numbers in Table 1-1. | — |
| June 2004, version 1.1 | ■ Updated timing numbers in Table 1-1. | — |

# 2. MAX II Architecture

MII51002-2.2

## Introduction

This chapter describes the architecture of the MAX II device and contains the following sections:


- "Functional Description" on page 2–1
- "Logic Array Blocks" on page 2–4
- "Logic Elements" on page 2–6
- "MultiTrack Interconnect" on page 2–12
- "Global Signals" on page 2–16
- "User Flash Memory Block" on page 2–18
- "MultiVolt Core" on page 2–22
- "I/O Structure" on page 2–23


## Functional Description

MAX® II devices contain a two-dimensional row- and column-based architecture to implement custom logic. Row and column interconnects provide signal interconnects between the logic array blocks (LABs).

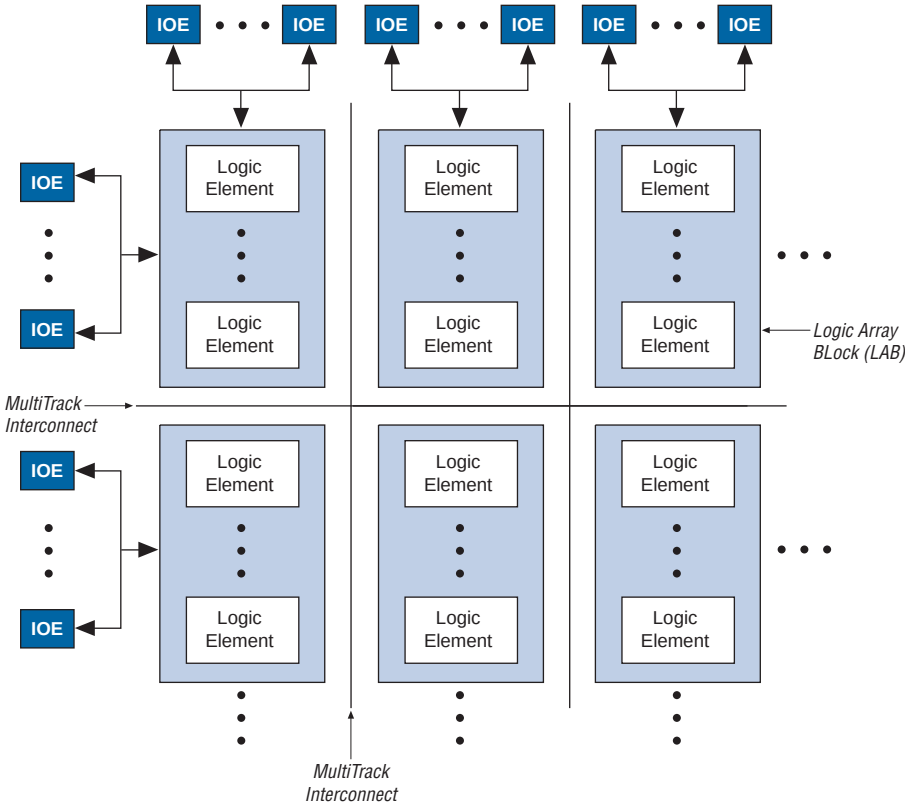
The logic array consists of LABs, with 10 logic elements (LEs) in each LAB. An LE is a small unit of logic providing efficient implementation of user logic functions. LABs are grouped into rows and columns across the device. The MultiTrack interconnect provides fast granular timing delays between LABs. The fast routing between LEs provides minimum timing delay for added levels of logic versus globally routed interconnect structures.

The MAX II device I/O pins are fed by I/O elements (IOE) located at the ends of LAB rows and columns around the periphery of the device. Each IOE contains a bidirectional I/O buffer with several advanced features. I/O pins support Schmitt trigger inputs and various single-ended standards, such as 66-MHz, 32-bit PCI, and LVTTL.

MAX II devices provide a global clock network. The global clock network consists of four global clock lines that drive throughout the entire device, providing clocks for all resources within the device. The global clock lines can also be used for control signals such as clear, preset, or output enable.

Figure 2–1 shows a functional block diagram of the MAX II device.

**Figure 2–1.** MAX II Device Block Diagram




Each MAX II device contains a flash memory block within its floorplan. On the EPM240 device, this block is located on the left side of the device. On the EPM570, EPM1270, and EPM2210 devices, the flash memory block is located on the bottom-left area of the device. The majority of this flash memory storage is partitioned as the dedicated configuration flash memory (CFM) block. The CFM block provides the non-volatile storage for all of the SRAM configuration information. The CFM automatically downloads and configures the logic and I/O at power-up, providing instant-on operation.

For more information about configuration upon power-up, refer to the *Hot Socketing and Power-On Reset in MAX II Devices* chapter in the *MAX II Device Handbook*.

A portion of the flash memory within the MAX II device is partitioned into a small block for user data. This user flash memory (UFM) block provides 8,192 bits of general-purpose user storage. The UFM provides programmable port connections to the logic array for reading and writing. There are three LAB rows adjacent to this block, with column numbers varying by device.

Table 2–1 shows the number of LAB rows and columns in each device, as well as the number of LAB rows and columns adjacent to the flash memory area in the EPM570, EPM1270, and EPM2210 devices. The long LAB rows are full LAB rows that extend from one side of row I/O blocks to the other. The short LAB rows are adjacent to the UFM block; their length is shown as width in LAB columns.

**Table 2–1.** MAX II Device Resources

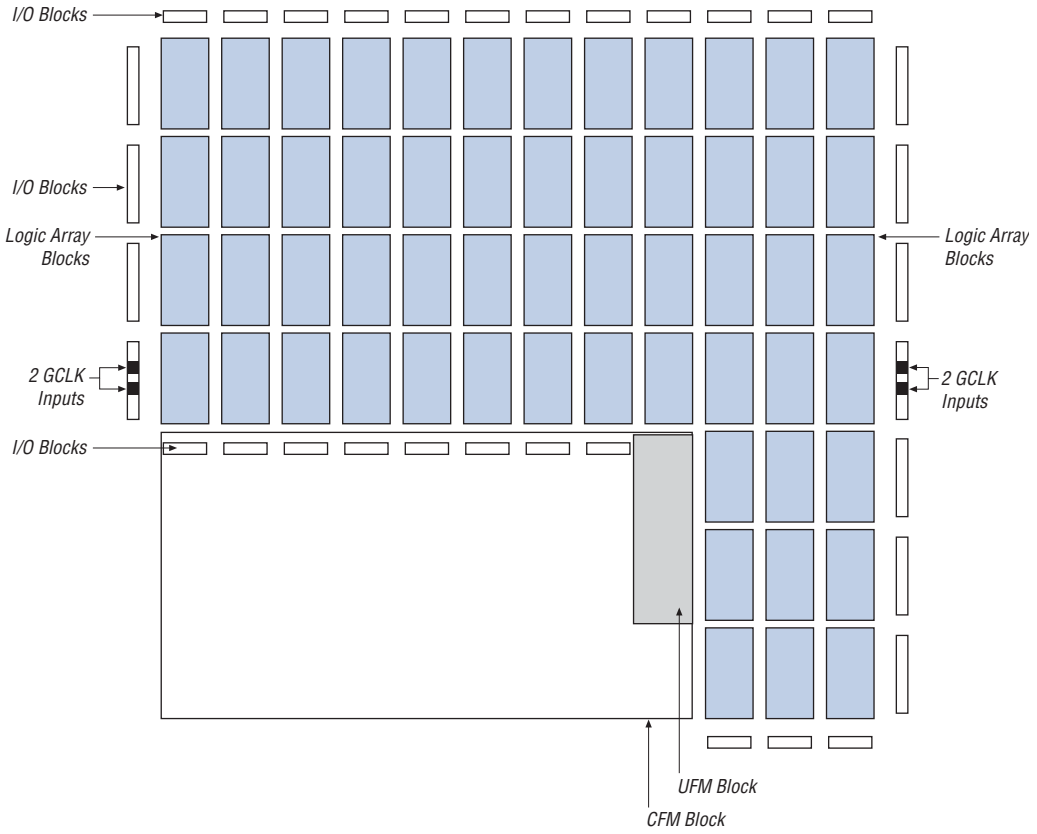
| Devices | UFM Blocks | LAB Columns | LAB Rows | | Total LABs |
|---|---|---|---|---|---|
| | | | Long LAB Rows | Short LAB Rows (Width) *(1)* | |
| EPM240 | 1 | 6 | 4 | — | 24 |
| EPM570 | 1 | 12 | 4 | 3 (3) | 57 |
| EPM1270 | 1 | 16 | 7 | 3 (5) | 127 |
| EPM2210 | 1 | 20 | 10 | 3 (7) | 221 |

**Note to Table 2–1:**

(1) The width is the number of LAB columns in length.

Figure 2–2 shows a floorplan of a MAX II device.

**Figure 2–2.** MAX II Device Floorplan *(Note 1)*



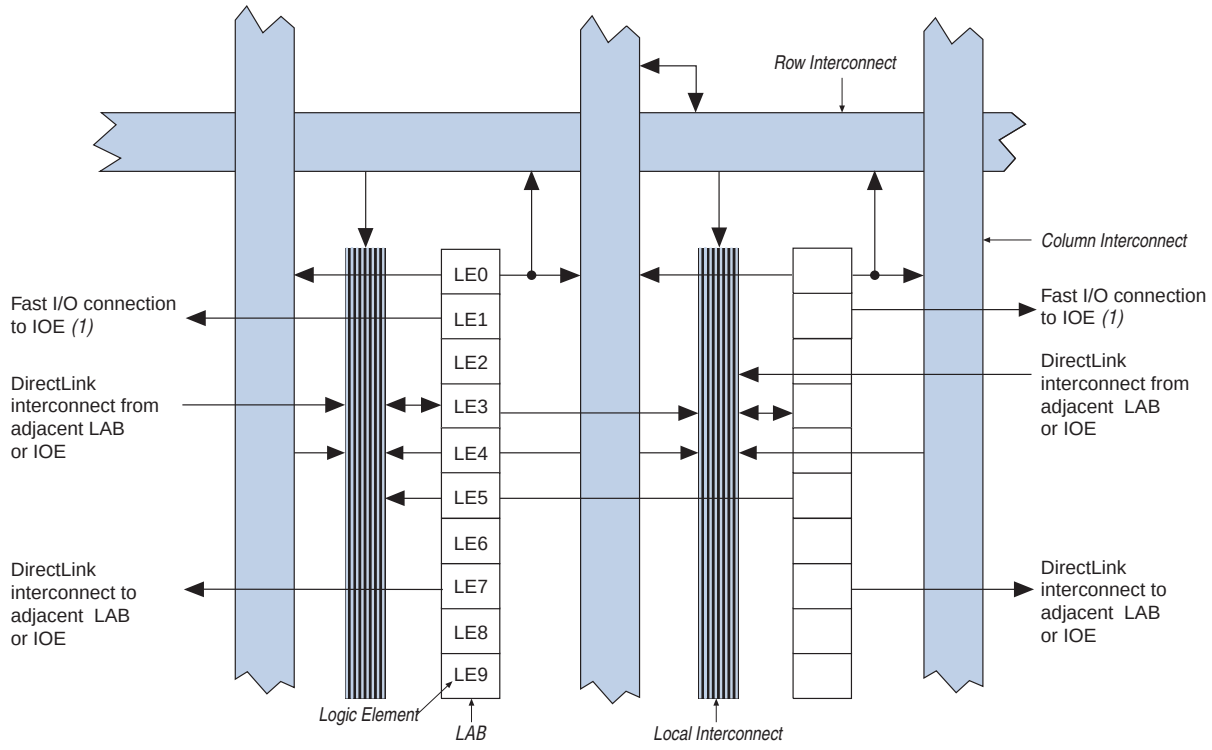

**Note to Figure 2–2:**

(1) The device shown is an EPM570 device. EPM1270 and EPM2210 devices have a similar floorplan with more LABs. For EPM240 devices, the CFM and UFM blocks are located on the left side of the device.

# Logic Array Blocks

Each LAB consists of 10 LEs, LE carry chains, LAB control signals, a local interconnect, a look-up table (LUT) chain, and register chain connection lines. There are 26 possible unique inputs into an LAB, with an additional 10 local feedback input lines fed by LE outputs in the same LAB. The local interconnect transfers signals between LEs in the same LAB. LUT chain connections transfer the output of one LE's LUT to the adjacent LE for fast sequential LUT connections within the same LAB. Register chain connections transfer the output of one LE's register to the adjacent LE's register within an LAB. The Quartus® II software places associated logic within an LAB or adjacent LABs, allowing the use of local, LUT chain, and register chain connections for performance and area efficiency. Figure 2–3 shows the MAX II LAB.

**Figure 2–3.** MAX II LAB Structure



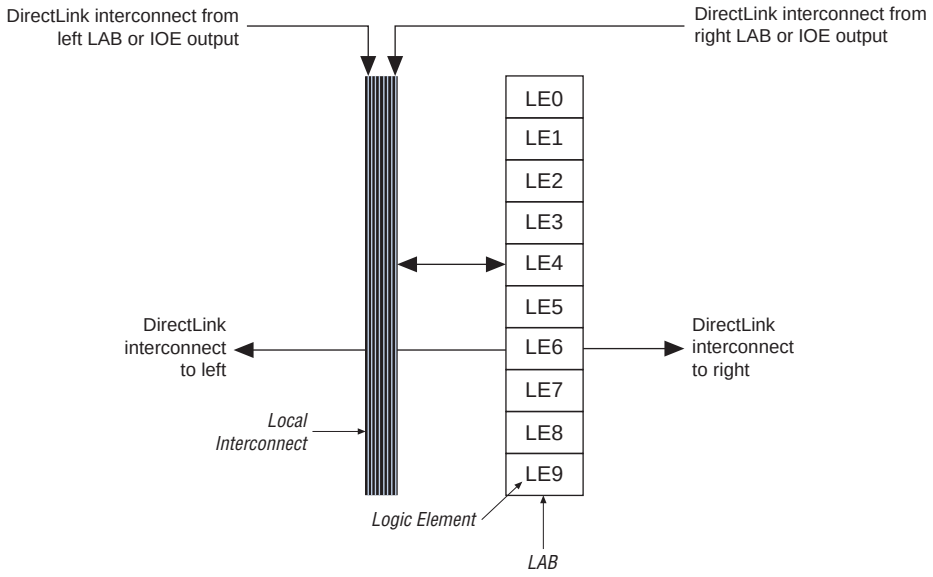

**Note to Figure 2–3:**

(1) Only from LABs adjacent to IOEs.

## LAB Interconnects

The LAB local interconnect can drive LEs within the same LAB. The LAB local interconnect is driven by column and row interconnects and LE outputs within the same LAB. Neighboring LABs, from the left and right, can also drive an LAB's local interconnect through the DirectLink connection. The DirectLink connection feature minimizes the use of row and column interconnects, providing higher performance and flexibility. Each LE can drive 30 other LEs through fast local and DirectLink interconnects. Figure 2–4 shows the DirectLink connection.

**Figure 2–4.** DirectLink Connection




## LAB Control Signals

Each LAB contains dedicated logic for driving control signals to its LEs. The control signals include two clocks, two clock enables, two asynchronous clears, a synchronous clear, an asynchronous preset/load, a synchronous load, and add/subtract control signals, providing a maximum of 10 control signals at a time. Although synchronous load and clear signals are generally used when implementing counters, they can also be used with other functions.

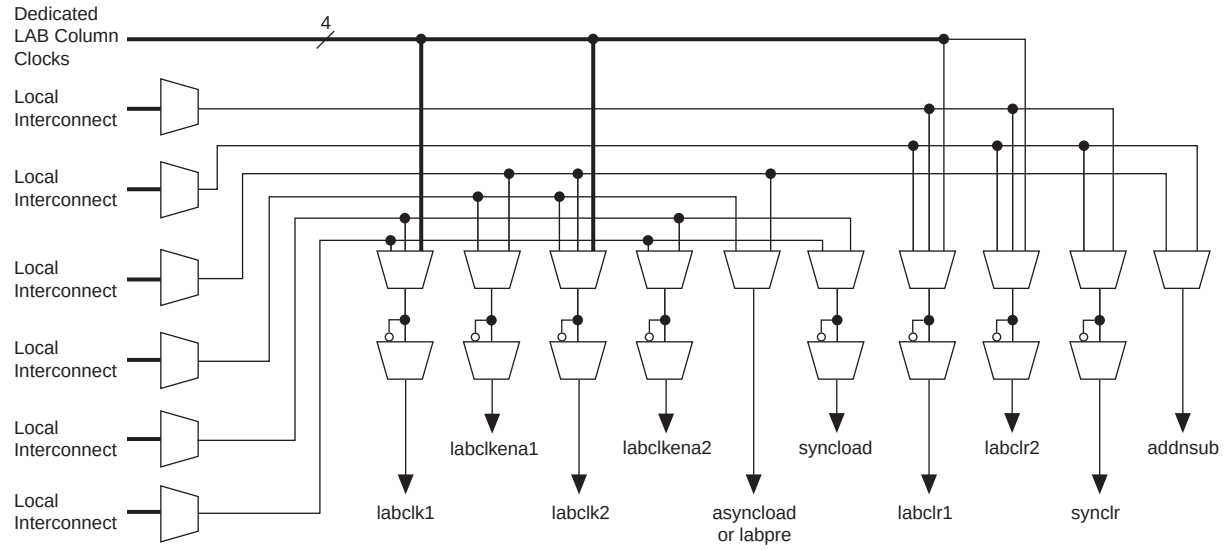
Each LAB can use two clocks and two clock enable signals. Each LAB's clock and clock enable signals are linked. For example, any LE in a particular LAB using the `labclk1` signal also uses `labclkena1`. If the LAB uses both the rising and falling edges of a clock, it also uses both LAB-wide clock signals. Deasserting the clock enable signal turns off the LAB-wide clock.

Each LAB can use two asynchronous clear signals and an asynchronous load/preset signal. By default, the Quartus II software uses a `NOT` gate push-back technique to achieve preset. If you disable the `NOT` gate push-back option or assign a given register to power-up high using the Quartus II software, the preset is then achieved using the asynchronous load signal with asynchronous load data input tied high.

With the LAB-wide addnsub control signal, a single LE can implement a one-bit adder and subtractor. This saves LE resources and improves performance for logic functions such as correlators and signed multipliers that alternate between addition and subtraction depending on data.

The LAB column clocks [3..0], driven by the global clock network, and LAB local interconnect generate the LAB-wide control signals. The MultiTrack interconnect structure drives the LAB local interconnect for non-global control signal generation. The MultiTrack interconnect's inherent low skew allows clock and control signal distribution in addition to data. Figure 2–5 shows the LAB control signal generation circuit.

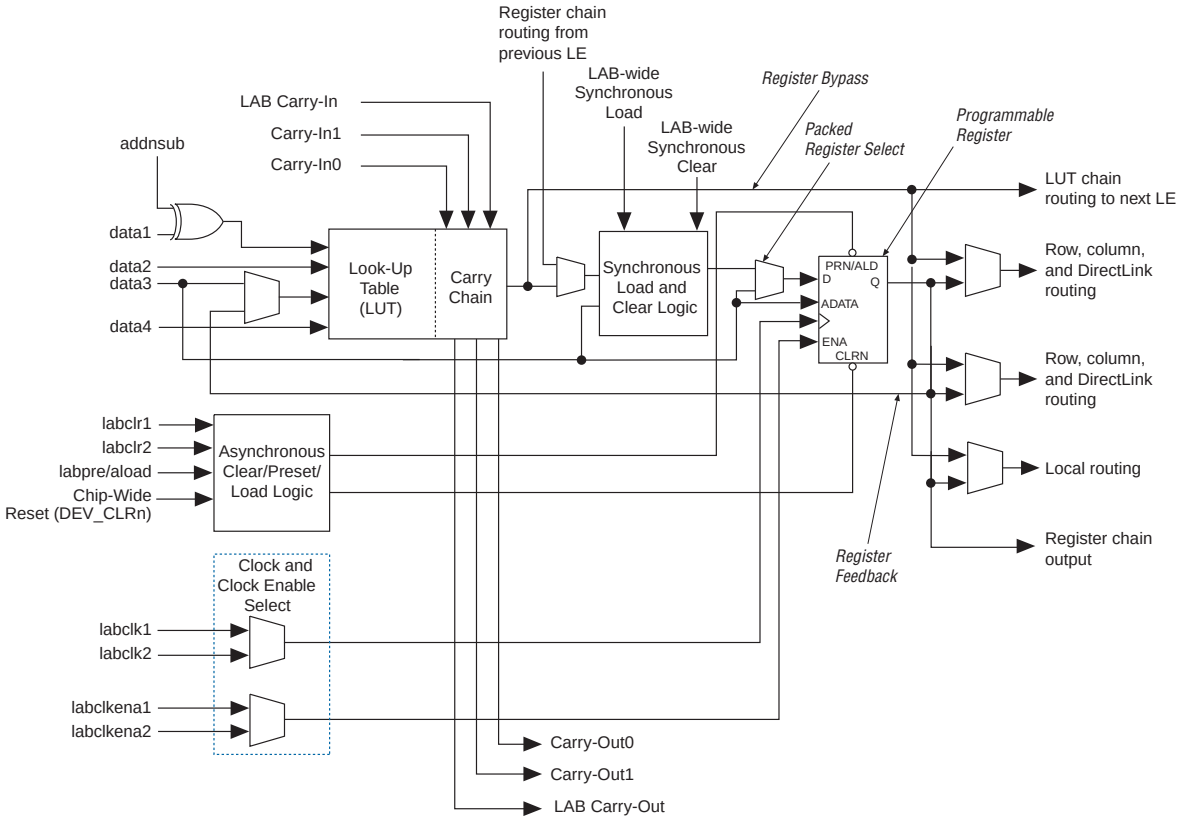
**Figure 2–5.** LAB-Wide Control Signals




# Logic Elements

The smallest unit of logic in the MAX II architecture, the LE, is compact and provides advanced features with efficient logic utilization. Each LE contains a four-input LUT, which is a function generator that can implement any function of four variables. In addition, each LE contains a programmable register and carry chain with carry-select capability. A single LE also supports dynamic single-bit addition or subtraction mode selectable by an LAB-wide control signal. Each LE drives all types of interconnects: local, row, column, LUT chain, register chain, and DirectLink interconnects. See Figure 2–6.

**Figure 2–6.** MAX II LE




Each LE's programmable register can be configured for D, T, JK, or SR operation. Each register has data, true asynchronous load data, clock, clock enable, clear, and asynchronous load/preset inputs. Global signals, general-purpose I/O pins, or any LE can drive the register's clock and clear control signals. Either general-purpose I/O pins or LEs can drive the clock enable, preset, asynchronous load, and asynchronous data. The asynchronous load data input comes from the data3 input of the LE. For combinational functions, the LUT output bypasses the register and drives directly to the LE outputs.

Each LE has three outputs that drive the local, row, and column routing resources. The LUT or register output can drive these three outputs independently. Two LE outputs drive column or row and DirectLink routing connections and one drives local interconnect resources. This allows the LUT to drive one output while the register drives another output. This register packing feature improves device utilization because the device can use the register and the LUT for unrelated functions. Another special packing mode allows the register output to feed back into the LUT of the same LE so that the register is packed with its own fan-out LUT. This provides another mechanism for improved fitting. The LE can also drive out registered and unregistered versions of the LUT output.

## LUT Chain and Register Chain

In addition to the three general routing outputs, the LEs within an LAB have LUT chain and register chain outputs. LUT chain connections allow LUTs within the same LAB to cascade together for wide input functions. Register chain outputs allow registers within the same LAB to cascade together. The register chain output allows an LAB to use LUTs for a single combinational function and the registers to be used for an unrelated shift register implementation. These resources speed up connections between LABs while saving local interconnect resources. Refer to "MultiTrack Interconnect" on page 2–12 for more information about LUT chain and register chain connections.

## addnsub Signal

The LE's dynamic adder/subtractor feature saves logic resources by using one set of LEs to implement both an adder and a subtractor. This feature is controlled by the LAB-wide control signal addnsub. The addnsub signal sets the LAB to perform either A + B or A – B. The LUT computes addition; subtraction is computed by adding the two's complement of the intended subtractor. The LAB-wide signal converts to two's complement by inverting the B bits within the LAB and setting carry-in to 1, which adds one to the least significant bit (LSB). The LSB of an adder/subtractor must be placed in the first LE of the LAB, where the LAB-wide addnsub signal automatically sets the carry-in to 1. The Quartus II Compiler automatically places and uses the adder/subtractor feature when using adder/subtractor parameterized functions.

## LE Operating Modes

The MAX II LE can operate in one of the following modes:

- "Normal Mode"
- "Dynamic Arithmetic Mode"

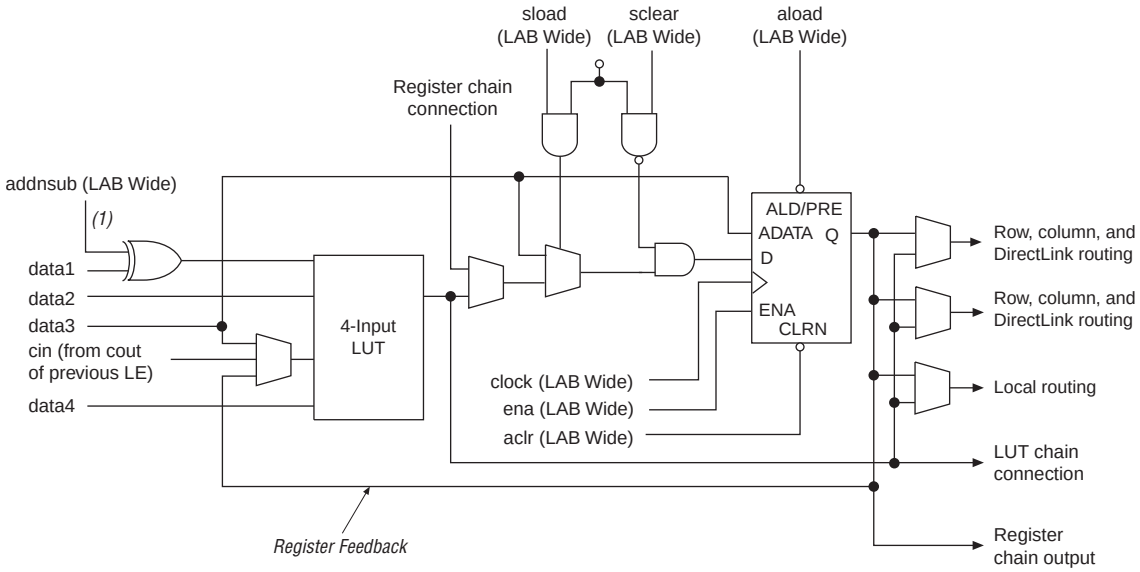
Each mode uses LE resources differently. In each mode, eight available inputs to the LE, the four data inputs from the LAB local interconnect, `carry-in0` and `carry-in1` from the previous LE, the LAB carry-in from the previous carry-chain LAB, and the register chain connection are directed to different destinations to implement the desired logic function. LAB-wide signals provide clock, asynchronous clear, asynchronous preset/load, synchronous clear, synchronous load, and clock enable control for the register. These LAB-wide signals are available in all LE modes. The addnsub control signal is allowed in arithmetic mode.

The Quartus II software, in conjunction with parameterized functions such as library of parameterized modules (LPM) functions, automatically chooses the appropriate mode for common functions such as counters, adders, subtractors, and arithmetic functions.

## Normal Mode

The normal mode is suitable for general logic applications and combinational functions. In normal mode, four data inputs from the LAB local interconnect are inputs to a four-input LUT (see Figure 2–7). The Quartus II Compiler automatically selects the carry-in or the data3 signal as one of the inputs to the LUT. Each LE can use LUT chain connections to drive its combinational output directly to the next LE in the LAB. Asynchronous load data for the register comes from the data3 input of the LE. LEs in normal mode support packed registers.

**Figure 2–7.** LE in Normal Mode




**Note to Figure 2–7:**

(1) This signal is only allowed in normal mode if the LE is at the end of an adder/subtractor chain.

## Dynamic Arithmetic Mode

The dynamic arithmetic mode is ideal for implementing adders, counters, accumulators, wide parity functions, and comparators. An LE in dynamic arithmetic mode uses four 2-input LUTs configurable as a dynamic adder/subtractor. The first two 2-input LUTs compute two summations based on a possible carry-in of 1 or 0; the other two LUTs generate carry outputs for the two chains of the carry-select circuitry. As shown in Figure 2–8, the LAB carry-in signal selects either the `carry-in0` or carry-in1 chain. The selected chain's logic level in turn determines which parallel sum is generated as a combinational or registered output. For example, when implementing an adder, the sum output is the selection of two possible calculated sums:

```
data1 + data2 + carry in0
```

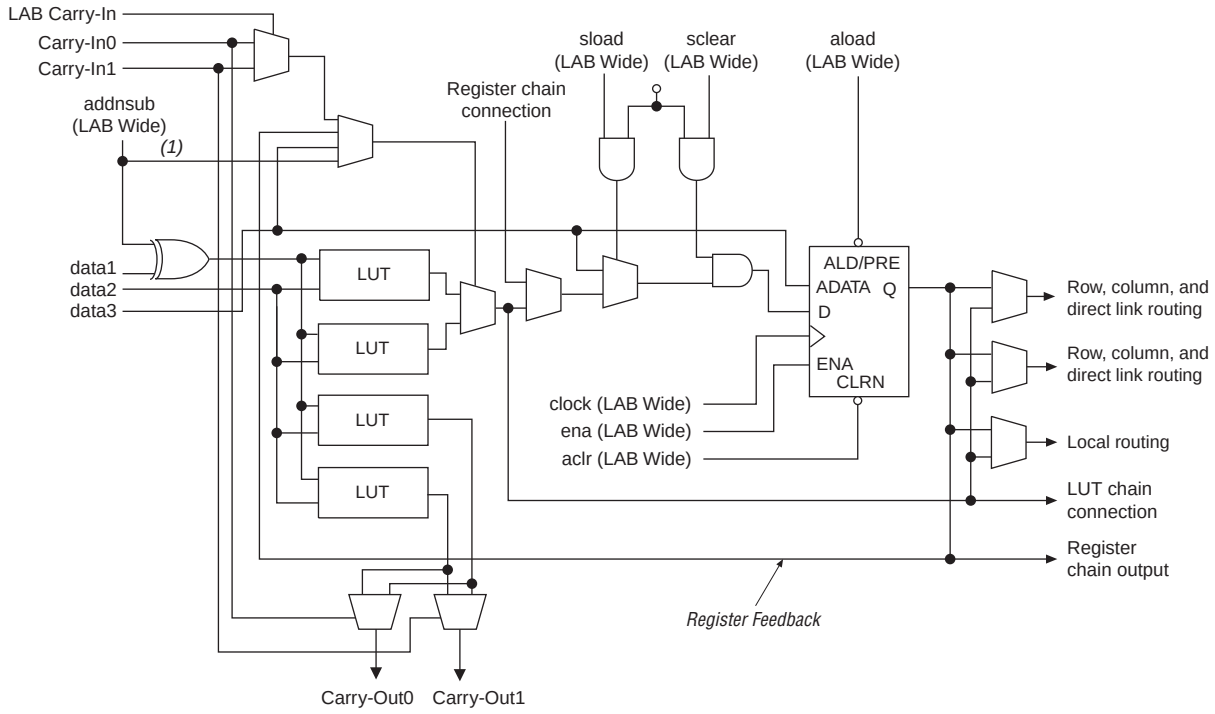
or

```
data1 + data2 + carry-in1
```

The other two LUTs use the data1 and data2 signals to generate two possible carry-out signals: one for a carry of 1 and the other for a carry of 0. The `carry-in0` signal acts as the carry-select for the `carry-out0` output and `carry-in1` acts as the carry-select for the `carry-out1` output. LEs in arithmetic mode can drive out registered and unregistered versions of the LUT output.

The dynamic arithmetic mode also offers clock enable, counter enable, synchronous up/down control, synchronous clear, synchronous load, and dynamic adder/subtractor options. The LAB local interconnect data inputs generate the counter enable and synchronous up/down control signals. The synchronous clear and synchronous load options are LAB-wide signals that affect all registers in the LAB. The Quartus II software automatically places any registers that are not used by the counter into other LABs. The addnsub LAB-wide signal controls whether the LE acts as an adder or subtractor.

**Figure 2–8.** LE in Dynamic Arithmetic Mode




**Note to Figure 2–8:**

(1) The addnsub signal is tied to the carry input for the first LE of a carry chain only.

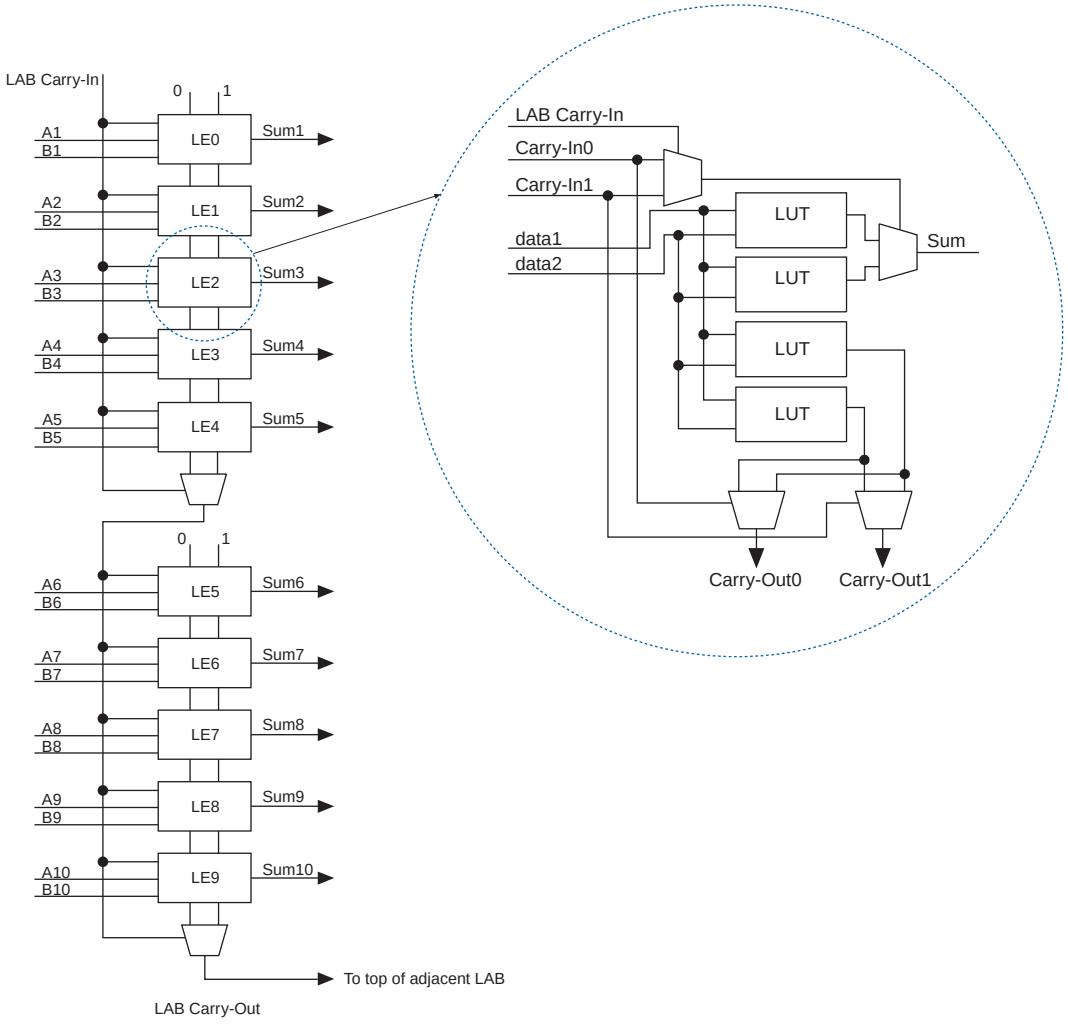
## Carry-Select Chain

The carry-select chain provides a very fast carry-select function between LEs in dynamic arithmetic mode. The carry-select chain uses the redundant carry calculation to increase the speed of carry functions. The LE is configured to calculate outputs for a possible carry-in of 0 and carry-in of 1 in parallel. The `carry-in0` and `carry-in1` signals from a lower-order bit feed forward into the higher-order bit via the parallel carry chain and feed into both the LUT and the next portion of the carry chain. Carry-select chains can begin in any LE within an LAB.

The speed advantage of the carry-select chain is in the parallel precomputation of carry chains. Since the LAB carry-in selects the precomputed carry chain, not every LE is in the critical path. Only the propagation delays between LAB carry-in generation (LE 5 and LE 10) are now part of the critical path. This feature allows the MAX II architecture to implement high-speed counters, adders, multipliers, parity functions, and comparators of arbitrary width.

Figure 2–9 shows the carry-select circuitry in an LAB for a 10-bit full adder. One portion of the LUT generates the sum of two bits using the input signals and the appropriate carry-in bit; the sum is routed to the output of the LE. The register can be bypassed for simple adders or used for accumulator functions. Another portion of the LUT generates carry-out bits. An LAB-wide carry-in bit selects which chain is used for the addition of given inputs. The carry-in signal for each chain, `carry-in0` or `carry-in1`, selects the carry-out to carry forward to the carry-in signal of the next-higher-order bit. The final carry-out signal is routed to an LE, where it is fed to local, row, or column interconnects.

**Figure 2–9.** Carry-Select Chain



The Quartus II software automatically creates carry chain logic during design processing, or you can create it manually during design entry. Parameterized functions such as LPM functions automatically take advantage of carry chains for the appropriate functions. The Quartus II software creates carry chains longer than 10 LEs by linking adjacent LABs within the same row together automatically. A carry chain can extend horizontally up to one full LAB row, but does not extend between LAB rows.

### Clear and Preset Logic Control

LAB-wide signals control the logic for the register's clear and preset signals. The LE directly supports an asynchronous clear and preset function. The register preset is achieved through the asynchronous load of a logic high. MAX II devices support simultaneous preset/asynchronous load and clear signals. An asynchronous clear signal takes precedence if both signals are asserted simultaneously. Each LAB supports up to two clears and one preset signal.

In addition to the clear and preset ports, MAX II devices provide a chip-wide reset pin (DEV_CLRn) that resets all registers in the device. An option set before compilation in the Quartus II software controls this pin. This chip-wide reset overrides all other control signals and uses its own dedicated routing resources (that is, it does not use any of the four global resources). Driving this signal low before or during power-up prevents user mode from releasing clears within the design. This allows you to control when clear is released on a device that has just been powered-up. If not set for its chip-wide reset function, the DEV_CLRn pin is a regular I/O pin.

By default, all registers in MAX II devices are set to power-up low. However, this power-up state can be set to high on individual registers during design entry using the Quartus II software.

## MultiTrack Interconnect

In the MAX II architecture, connections between LEs, the UFM, and device I/O pins are provided by the MultiTrack interconnect structure. The MultiTrack interconnect consists of continuous, performance-optimized routing lines used for inter- and intra-design block connectivity. The Quartus II Compiler automatically places critical design paths on faster interconnects to improve design performance.

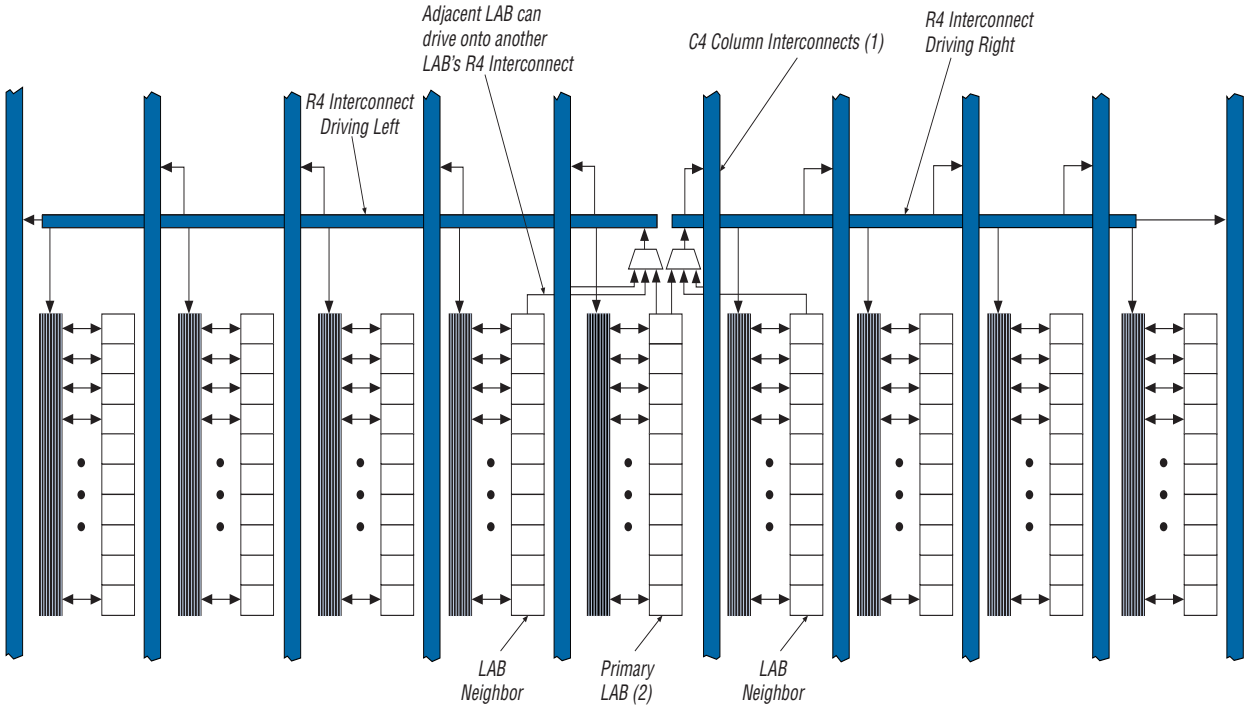
The MultiTrack interconnect consists of row and column interconnects that span fixed distances. A routing structure with fixed length resources for all devices allows predictable and short delays between logic levels instead of large delays associated with global or long routing lines. Dedicated row interconnects route signals to and from LABs within the same row. These row resources include:

- DirectLink interconnects between LABs
- R4 interconnects traversing four LABs to the right or left

The DirectLink interconnect allows an LAB to drive into the local interconnect of its left and right neighbors. The DirectLink interconnect provides fast communication between adjacent LABs and/or blocks without using row interconnect resources.

The R4 interconnects span four LABs and are used for fast row connections in a four-LAB region. Every LAB has its own set of R4 interconnects to drive either left or right. Figure 2–10 shows R4 interconnect connections from an LAB. R4 interconnects can drive and be driven by row IOEs. For LAB interfacing, a primary LAB or horizontal LAB neighbor can drive a given R4 interconnect. For R4 interconnects that drive to the right, the primary LAB and right neighbor can drive on to the interconnect. For R4 interconnects that drive to the left, the primary LAB and its left neighbor can drive on to the interconnect. R4 interconnects can drive other R4 interconnects to extend the range of LABs they can drive. R4 interconnects can also drive C4 interconnects for connections from one row to another.

**Figure 2–10.** R4 Interconnect Connections




**Notes to Figure 2–10:**

(1) C4 interconnects can drive R4 interconnects.

(2) This pattern is repeated for every LAB in the LAB row.

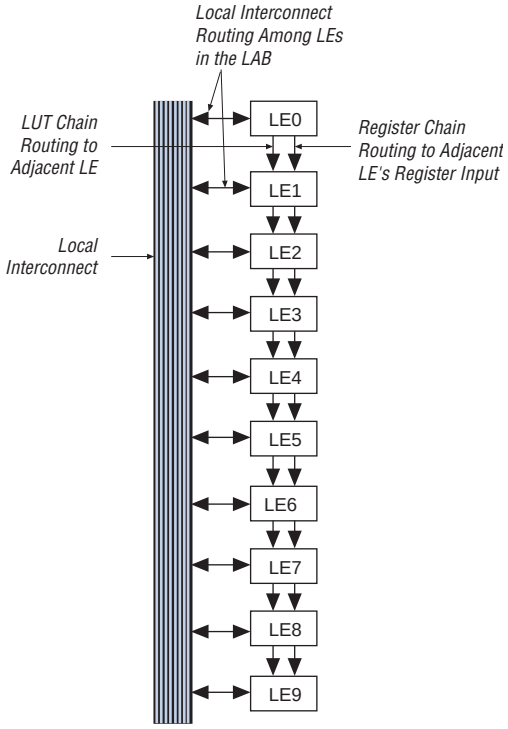
The column interconnect operates similarly to the row interconnect. Each column of LABs is served by a dedicated column interconnect, which vertically routes signals to and from LABs and row and column IOEs. These column resources include:

- LUT chain interconnects within an LAB
- Register chain interconnects within an LAB
- C4 interconnects traversing a distance of four LABs in an up and down direction

MAX II devices include an enhanced interconnect structure within LABs for routing LE output to LE input connections faster using LUT chain connections and register chain connections. The LUT chain connection allows the combinational output of an LE to directly drive the fast input of the LE right below it, bypassing the local interconnect. These resources can be used as a high-speed connection for wide fan-in

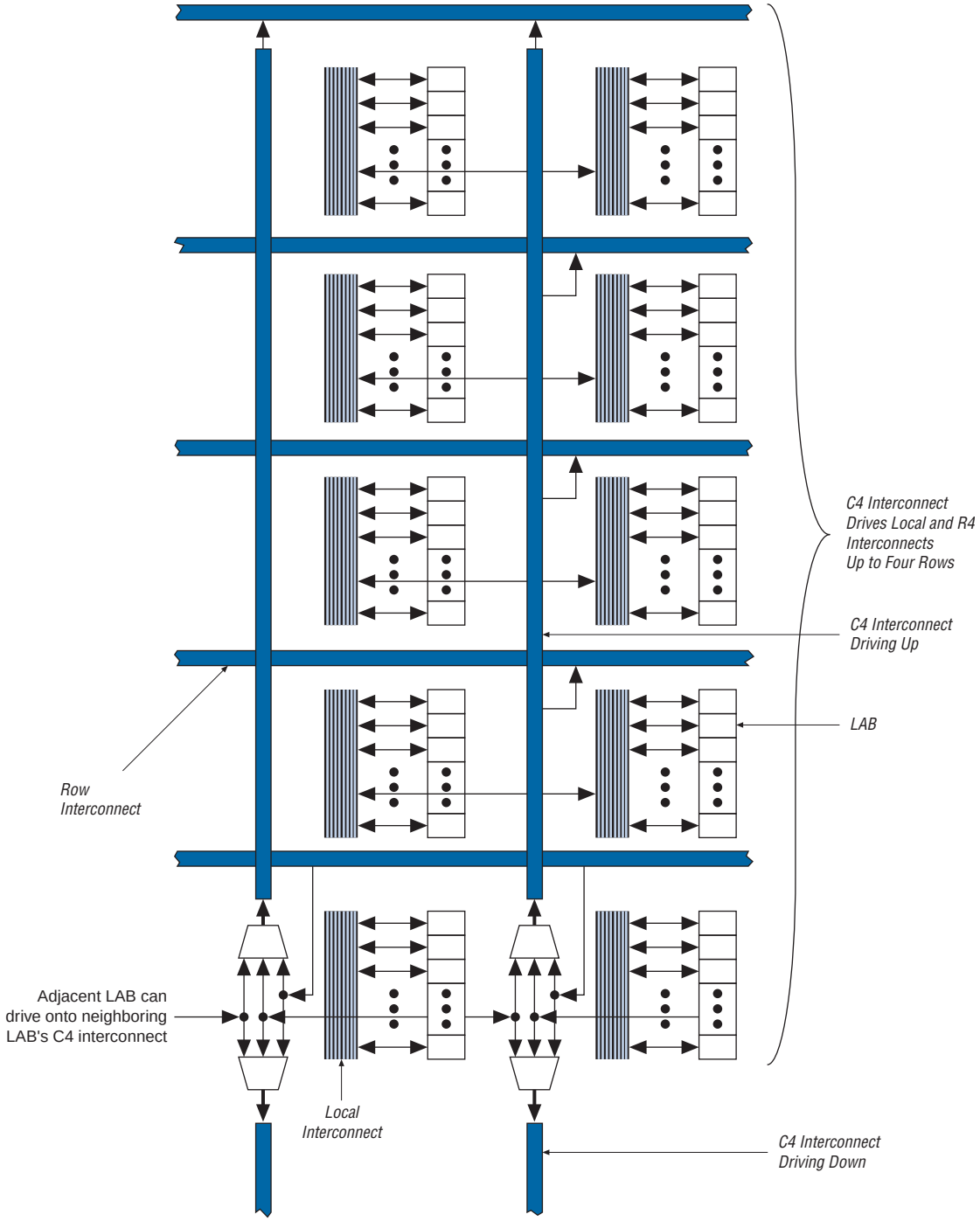
functions from LE 1 to LE 10 in the same LAB. The register chain connection allows the register output of one LE to connect directly to the register input of the next LE in the LAB for fast shift registers. The Quartus II Compiler automatically takes advantage of these resources to improve utilization and performance. Figure 2–11 shows the LUT chain and register chain interconnects.

**Figure 2–11.** LUT Chain and Register Chain Interconnects




The C4 interconnects span four LABs up or down from a source LAB. Every LAB has its own set of C4 interconnects to drive either up or down. Figure 2–12 shows the C4 interconnect connections from an LAB in a column. The C4 interconnects can drive and be driven by column and row IOEs. For LAB interconnection, a primary LAB or its vertical LAB neighbor can drive a given C4 interconnect. C4 interconnects can drive each other to extend their range as well as drive row interconnects for column-to-column connections.

**Figure 2–12.** C4 Interconnect Connections *(Note 1)*




**Note to Figure 2–12:**

(1) Each C4 interconnect can drive either up or down four rows.

The UFM block communicates with the logic array similar to LAB-to-LAB interfaces. The UFM block connects to row and column interconnects and has local interconnect regions driven by row and column interconnects. This block also has DirectLink interconnects for fast connections to and from a neighboring LAB. For more information about the UFM interface to the logic array, see "User Flash Memory Block" on page 2–18.

Table 2–2 shows the MAX II device routing scheme.

**Table 2–2.** MAX II Device Routing Scheme

| Source | Destination | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | **LUT Chain** | **Register Chain** | **Local *(1)*** | **DirectLink *(1)*** | **R4 *(1)*** | **C4 *(1)*** | **LE** | **UFM Block** | **Column IOE** | **Row IOE** | **Fast I/O *(1)*** |
| LUT Chain | — | — | — | — | — | — | ✓ | — | — | — | — |
| Register Chain | — | — | — | — | — | — | ✓ | — | — | — | — |
| Local Interconnect | — | — | — | — | — | — | ✓ | ✓ | ✓ | ✓ | — |
| DirectLink Interconnect | — | — | ✓ | — | — | — | — | — | — | — | — |
| R4 Interconnect | — | — | ✓ | — | ✓ | ✓ | — | — | — | — | — |
| C4 Interconnect | — | — | ✓ | — | ✓ | ✓ | — | — | — | — | — |
| LE | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | — | — | ✓ | ✓ | ✓ |
| UFM Block | — | — | ✓ | ✓ | ✓ | ✓ | — | — | — | — | — |
| Column IOE | — | — | — | — | — | ✓ | — | — | — | — | — |
| Row IOE | — | — | — | ✓ | ✓ | ✓ | — | — | — | — | — |

**Note to Table 2–2:**

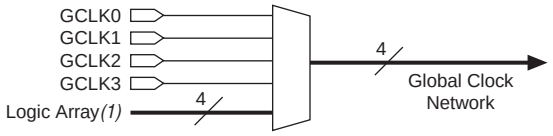
(1) These categories are interconnects.

# Global Signals

Each MAX II device has four dual-purpose dedicated clock pins (`GCLK[3..0]`, two pins on the left side and two pins on the right side) that drive the global clock network for clocking, as shown in Figure 2–13. These four pins can also be used as general-purpose I/O if they are not used to drive the global clock network.

The four global clock lines in the global clock network drive throughout the entire device. The global clock network can provide clocks for all resources within the device including LEs, LAB local interconnect, IOEs, and the UFM block. The global clock lines can also be used for global control signals, such as clock enables, synchronous or asynchronous clears, presets, output enables, or protocol control signals such as `TRDY` and `IRDY` for PCI. Internal logic can drive the global clock network for internally-generated global clocks and control signals. Figure 2–13 shows the various sources that drive the global clock network.

**Figure 2–13.** Global Clock Generation

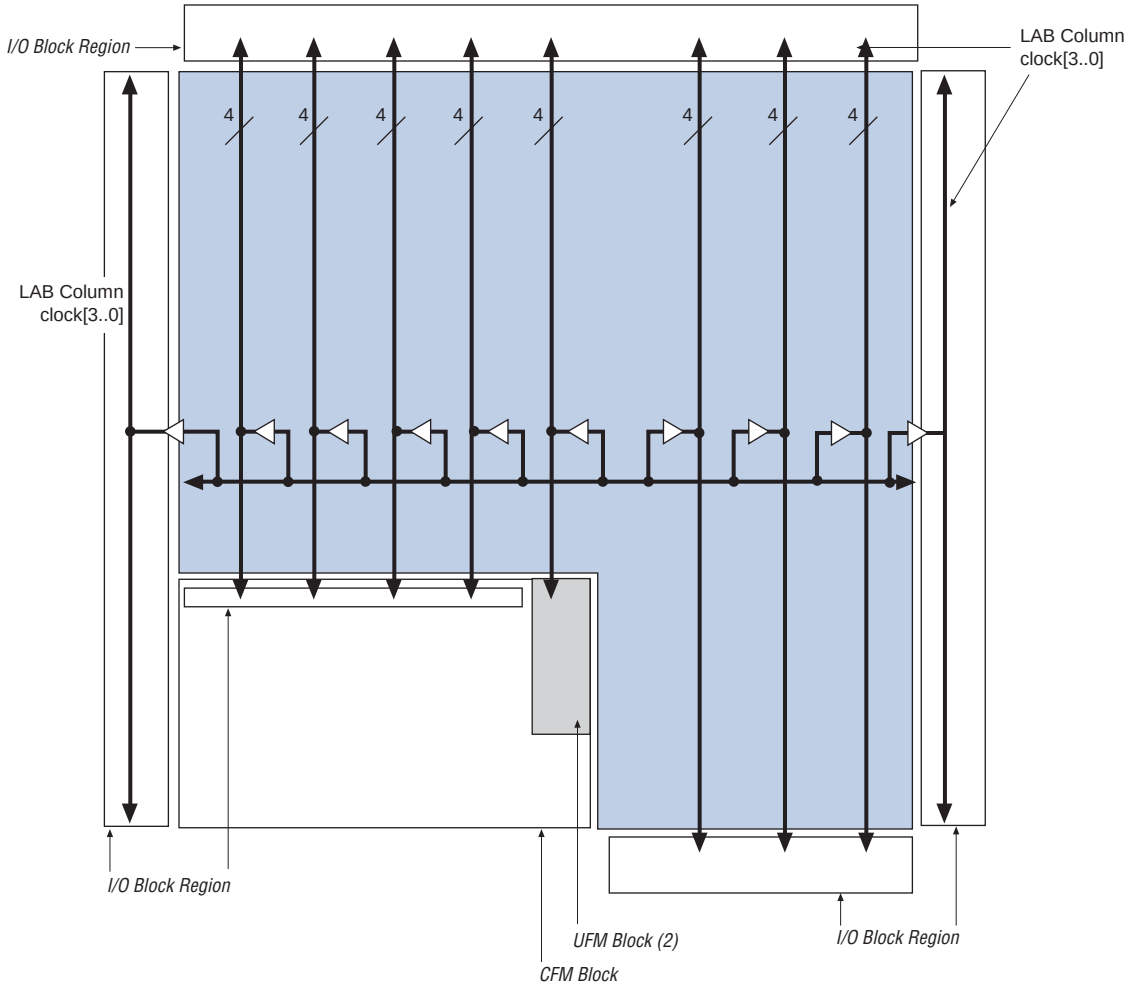



**Note to Figure 2–13:**

(1) Any I/O pin can use a MultiTrack interconnect to route as a logic array-generated global clock signal.

The global clock network drives to individual LAB column signals, LAB column clocks [3..0], that span an entire LAB column from the top to the bottom of the device. Unused global clocks or control signals in a LAB column are turned off at the LAB column clock buffers shown in Figure 2–14. The LAB column clocks [3..0] are multiplexed down to two LAB clock signals and one LAB clear signal. Other control signal types route from the global clock network into the LAB local interconnect. See "LAB Control Signals" on page 2–5 for more information.

**Figure 2–14.** Global Clock Network *(Note 1)*




**Notes to Figure 2–14:**

(1) LAB column clocks in I/O block regions provide high fan-out output enable signals.

(2) LAB column clocks drive to the UFM block.

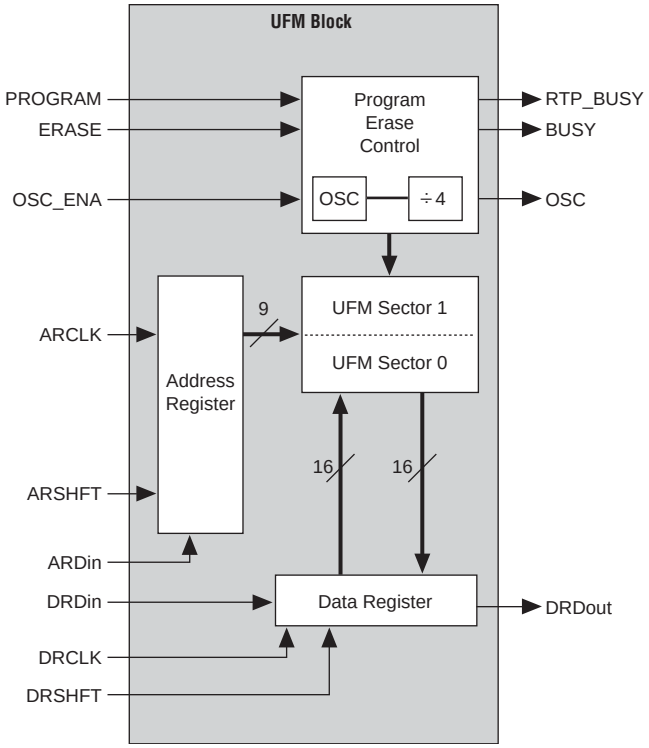
# User Flash Memory Block

MAX II devices feature a single UFM block, which can be used like a serial EEPROM for storing non-volatile information up to 8,192 bits. The UFM block connects to the logic array through the MultiTrack interconnect, allowing any LE to interface to the UFM block. Figure 2–15 shows the UFM block and interface signals. The logic array is used to create customer interface or protocol logic to interface the UFM block data outside of the device. The UFM block offers the following features:

- Non-volatile storage up to 16-bit wide and 8,192 total bits
- Two sectors for partitioned sector erase
- Built-in internal oscillator that optionally drives logic array
- Program, erase, and busy signals

- Auto-increment addressing
- Serial interface to logic array with programmable interface

**Figure 2–15.** UFM Block and Interface Signals




## UFM Storage

Each device stores up to 8,192 bits of data in the UFM block. Table 2–3 shows the data size, sector, and address sizes for the UFM block.

**Table 2–3.** UFM Array Size

| Device | Total Bits | Sectors | Address Bits | Data Width |
|---|---|---|---|---|
| EPM240<br>EPM570<br>EPM1270<br>EPM2210 | 8,192 | 2<br>(4,096 bits/sector) | 9 | 16 |

There are 512 locations with 9-bit addressing ranging from `000h` to `1FFh`. Sector 0 address space is `000h` to `0FFh` and Sector 1 address space is from `100h` to `1FFh`. The data width is up to 16 bits of data. The Quartus II software automatically creates logic to accommodate smaller read or program data widths. Erasure of the UFM involves individual sector erasing (that is, one erase of sector 0 and one erase of sector 1 is required to erase the entire UFM block). Since sector erase is required before a program or write, having two sectors enables a sector size of data to be left untouched while the other sector is erased and programmed with new data.

## Internal Oscillator

As shown in Figure 2–15, the dedicated circuitry within the UFM block contains an oscillator. The dedicated circuitry uses this internally for its read and program operations. This oscillator's divide by 4 output can drive out of the UFM block as a logic interface clock source or for general-purpose logic clocking. The typical OSC output signal frequency ranges from 3.3 to 5.5 MHz, and its exact frequency of operation is not programmable.

## Program, Erase, and Busy Signals

The UFM block's dedicated circuitry automatically generates the necessary internal program and erase algorithm once the PROGRAM or ERASE input signals have been asserted. The PROGRAM or ERASE signal must be asserted until the busy signal deasserts, indicating the UFM internal program or erase operation has completed. The UFM block also supports JTAG as the interface for programming and/or reading.

For more information about programming and erasing the UFM block, refer to the *Using User Flash Memory in MAX II Devices* chapter in the *MAX II Device Handbook*.

## Auto-Increment Addressing

The UFM block supports standard read or stream read operations. The stream read is supported with an auto-increment address feature. Deasserting the ARSHIFT signal while clocking the ARCLK signal increments the address register value to read consecutive locations from the UFM array.

## Serial Interface

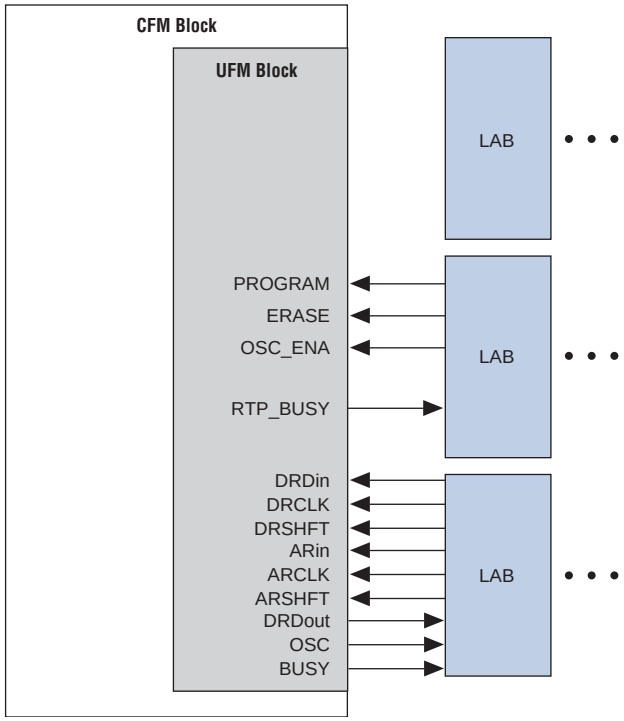
The UFM block supports a serial interface with serial address and data signals. The internal shift registers within the UFM block for address and data are 9 bits and 16 bits wide, respectively. The Quartus II software automatically generates interface logic in LEs for a parallel address and data interface to the UFM block. Other standard protocol interfaces such as SPI are also automatically generated in LE logic by the Quartus II software.

For more information about the UFM interface signals and the Quartus II LE-based alternate interfaces, refer to the *Using User Flash Memory in MAX II Devices* chapter in the *MAX II Device Handbook*.

## UFM Block to Logic Array Interface

The UFM block is a small partition of the flash memory that contains the CFM block, as shown in Figure 2–1 and Figure 2–2. The UFM block for the EPM240 device is located on the left side of the device adjacent to the left most LAB column. The UFM block for the EPM570, EPM1270, and EPM2210 devices is located at the bottom left of the device. The UFM input and output signals interface to all types of interconnects (R4 interconnect, C4 interconnect, and DirectLink interconnect to/from adjacent LAB rows). The UFM signals can also be driven from global clocks, GCLK[3..0]. The interface region for the EPM240 device is shown in Figure 2–16. The interface regions for EPM570, EPM1270, and EPM2210 devices are shown in Figure 2–17.

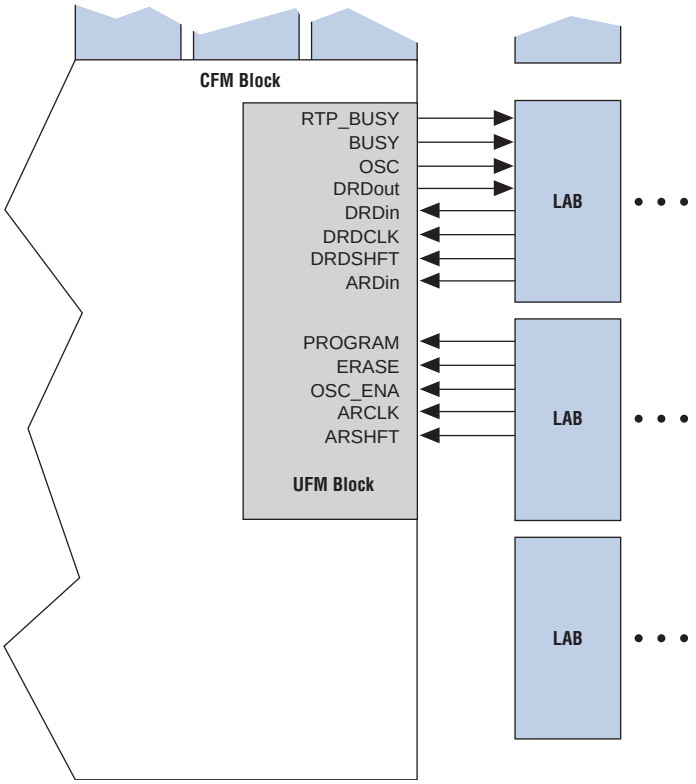
**Figure 2–16.** EPM240 UFM Block LAB Row Interface *(Note 1)*




**Note to Figure 2–16:**

(1) The UFM block inputs and outputs can drive to/from all types of interconnects, not only DirectLink interconnects from adjacent row LABs.

**Figure 2–17.** EPM570, EPM1270, and EPM2210 UFM Block LAB Row Interface

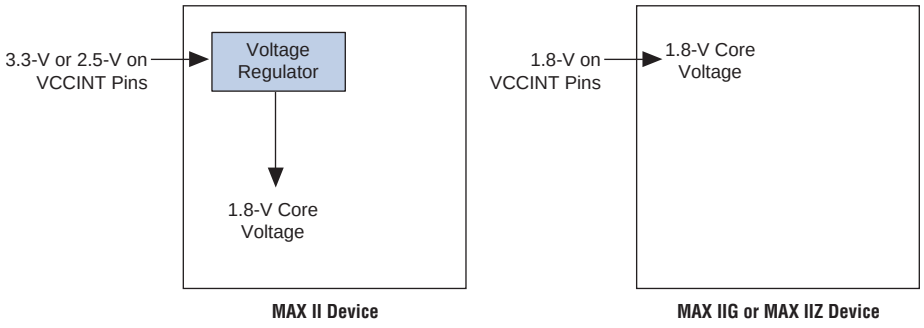



# MultiVolt Core

The MAX II architecture supports the MultiVolt core feature, which allows MAX II devices to support multiple $V_{CC}$ levels on the $V_{CCINT}$ supply. An internal linear voltage regulator provides the necessary 1.8-V internal voltage supply to the device. The voltage regulator supports 3.3-V or 2.5-V supplies on its inputs to supply the 1.8-V internal voltage to the device, as shown in Figure 2–18. The voltage regulator is not guaranteed for voltages that are between the maximum recommended 2.5-V operating voltage and the minimum recommended 3.3-V operating voltage.

The MAX IIG and MAX IIZ devices use external 1.8-V supply. The 1.8-V $V_{CC}$ external supply powers the device core directly.

**Figure 2–18.** MultiVolt Core Feature in MAX II Devices



# I/O Structure

IOEs support many features, including:

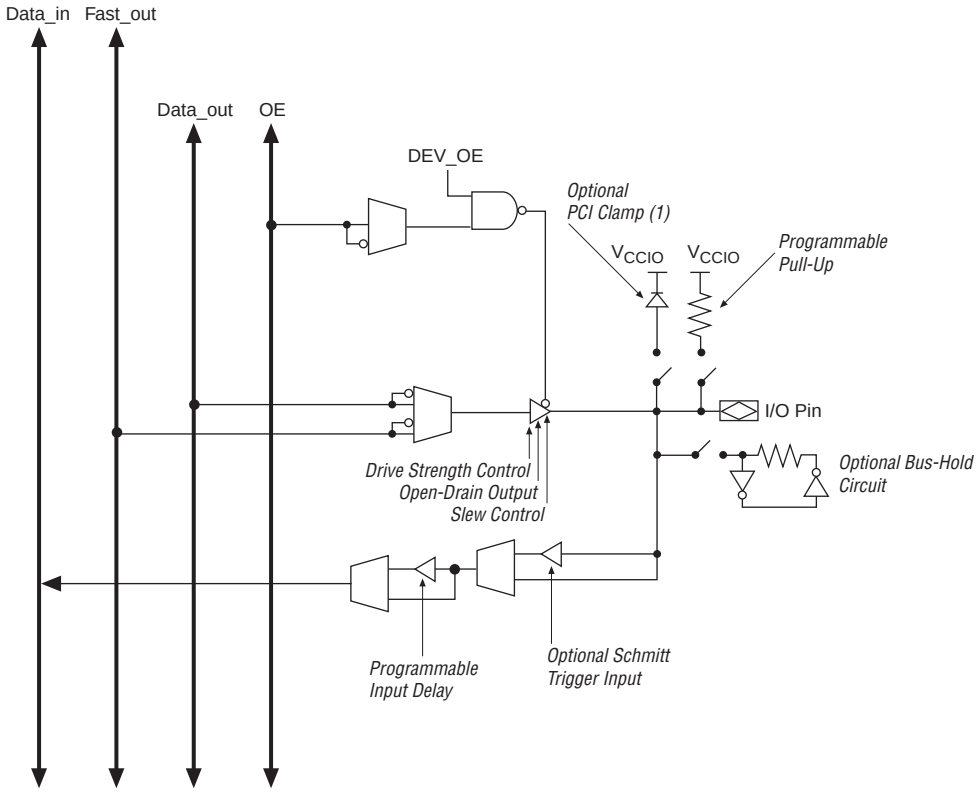
- LVTTL and LVCMOS I/O standards
- 3.3-V, 32-bit, 66-MHz PCI compliance
- Joint Test Action Group (JTAG) boundary-scan test (BST) support
- Programmable drive strength control
- Weak pull-up resistors during power-up and in system programming
- Slew-rate control
- Tri-state buffers with individual output enable control
- Bus-hold circuitry
- Programmable pull-up resistors in user mode
- Unique output enable per pin
- Open-drain outputs
- Schmitt trigger inputs
- Fast I/O connection
- Programmable input delay

MAX II device IOEs contain a bidirectional I/O buffer. Figure 2–19 shows the MAX II IOE structure. Registers from adjacent LABs can drive to or be driven from the IOE's bidirectional I/O buffers. The Quartus II software automatically attempts to place registers in the adjacent LAB with fast I/O connection to achieve the fastest possible clock-to-output and registered output enable timing. For input registers, the Quartus II software automatically routes the register to guarantee zero hold time. You can set timing assignments in the Quartus II software to achieve desired I/O timing.

## Fast I/O Connection

A dedicated fast I/O connection from the adjacent LAB to the IOEs within an I/O block provides faster output delays for clock-to-output and $t_{PD}$ propagation delays. This connection exists for data output signals, not output enable signals or input signals. Figure 2–20, Figure 2–21, and Figure 2–22 illustrate the fast I/O connection.

**Figure 2–19.** MAX II IOE Structure




**Note to Figure 2–19:**

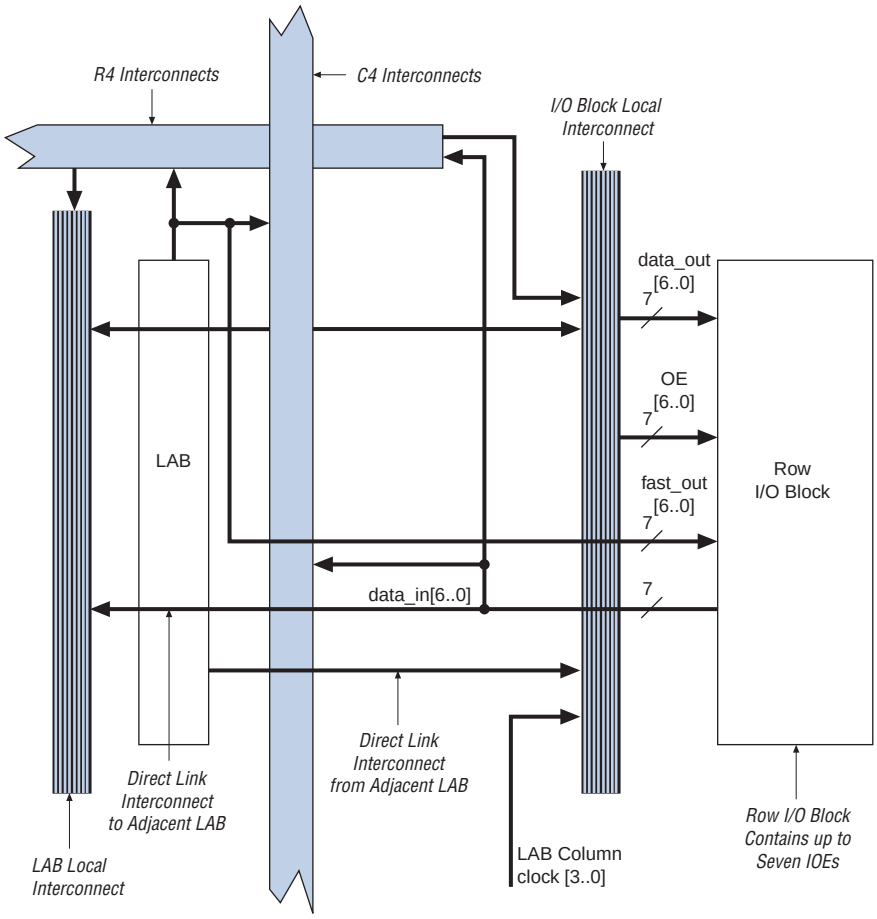
(1) Available in EPM1270 and EPM2210 devices only.

## I/O Blocks

The IOEs are located in I/O blocks around the periphery of the MAX II device. There are up to seven IOEs per row I/O block (5 maximum in the EPM240 device) and up to four IOEs per column I/O block. Each column or row I/O block interfaces with its adjacent LAB and MultiTrack interconnect to distribute signals throughout the device. The row I/O blocks drive row, column, or DirectLink interconnects. The column I/O blocks drive column interconnects.

Figure 2–20 shows how a row I/O block connects to the logic array.

**Figure 2–20.** Row I/O Block Connection to the Interconnect *(Note 1)*

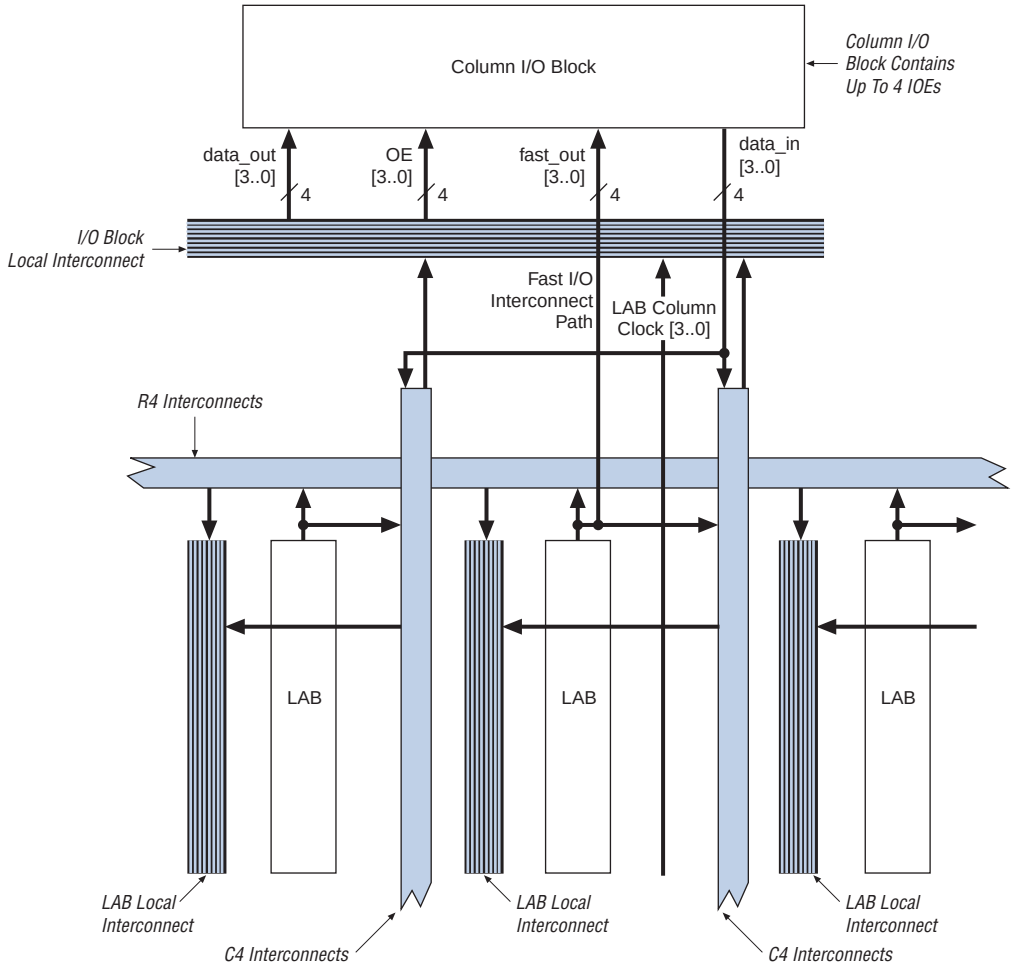



**Note to Figure 2–20:**

(1) Each of the seven IOEs in the row I/O block can have one `data_out` or `fast_out` output, one `OE` output, and one `data_in` input.

Figure 2–21 shows how a column I/O block connects to the logic array.

**Figure 2–21.** Column I/O Block Connection to the Interconnect *(Note 1)*




**Note to Figure 2–21:**

(1) Each of the four IOEs in the column I/O block can have one `data_out` or `fast_out` output, one `OE` output, and one `data_in` input.

## I/O Standards and Banks

MAX II device IOEs support the following I/O standards:

- 3.3-V LVTTL/LVCMOS
- 2.5-V LVTTL/LVCMOS
- 1.8-V LVTTL/LVCMOS
- 1.5-V LVCMOS
- 3.3-V PCI

Table 2–4 describes the I/O standards supported by MAX II devices.

**Table 2–4.** MAX II I/O Standards

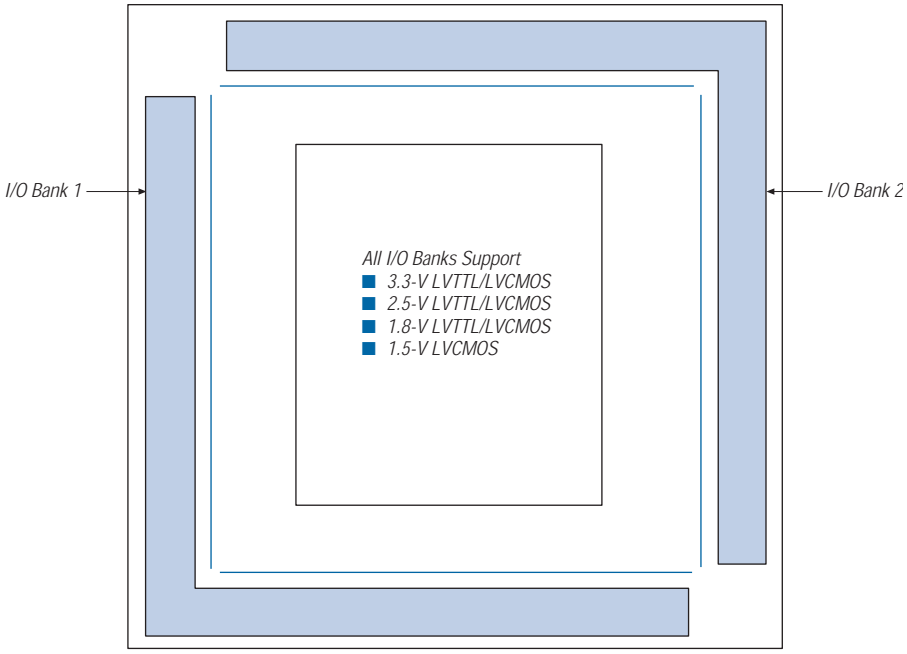
| I/O Standard | Type | Output Supply Voltage (VCCIO) (V) |
|---|---|---|
| 3.3-V LVTTL/LVCMOS | Single-ended | 3.3 |
| 2.5-V LVTTL/LVCMOS | Single-ended | 2.5 |
| 1.8-V LVTTL/LVCMOS | Single-ended | 1.8 |
| 1.5-V LVCMOS | Single-ended | 1.5 |
| 3.3-V PCI *(1)* | Single-ended | 3.3 |

**Note to Table 2–4:**

(1) The 3.3-V PCI compliant I/O is supported in Bank 3 of the EPM1270 and EPM2210 devices.

The EPM240 and EPM570 devices support two I/O banks, as shown in Figure 2–22. Each of these banks support all the LVTTL and LVCMOS standards shown in Table 2–4. PCI compliant I/O is not supported in these devices and banks.

**Figure 2–22.** MAX II I/O Banks for EPM240 and EPM570 *(Note 1)*, *(2)*



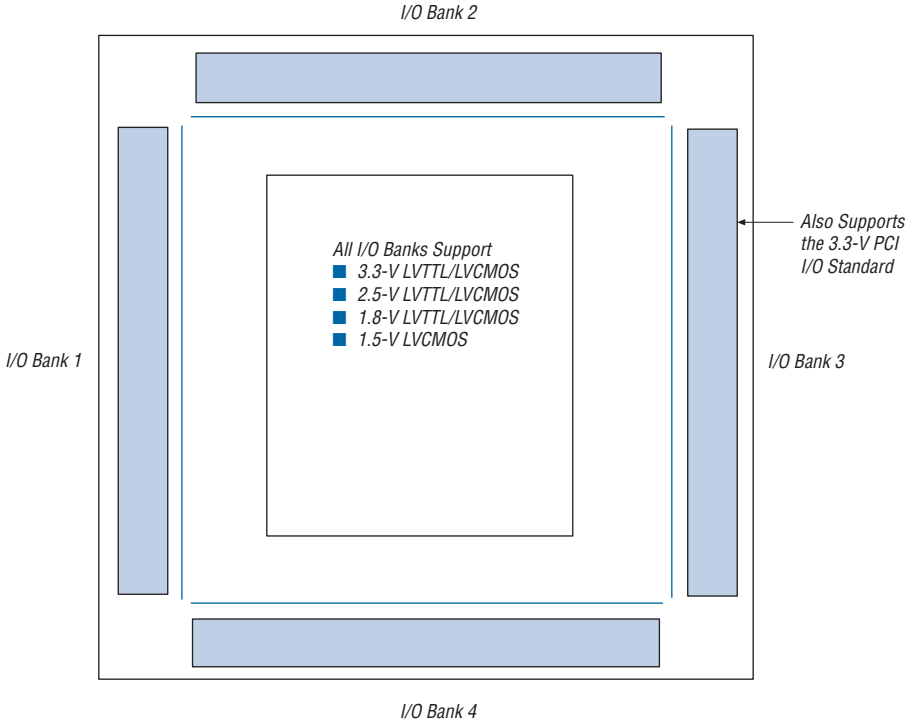

**Notes to Figure 2–22:**

(1) Figure 2–22 is a top view of the silicon die.

(2) Figure 2–22 is a graphical representation only. Refer to the pin list and the Quartus II software for exact pin locations.

The EPM1270 and EPM2210 devices support four I/O banks, as shown in Figure 2–23. Each of these banks support all of the LVTTL and LVCMOS standards shown in Table 2–4. PCI compliant I/O is supported in Bank 3. Bank 3 supports the PCI clamping diode on inputs and PCI drive compliance on outputs. You must use Bank 3 for designs requiring PCI compliant I/O pins. The Quartus II software automatically places I/O pins in this bank if assigned with the PCI I/O standard.

**Figure 2–23.** MAX II I/O Banks for EPM1270 and EPM2210 *(Note 1), (2)*




**Notes to Figure 2–23:**

(1) Figure 2–23 is a top view of the silicon die.

(2) Figure 2–23 is a graphical representation only. Refer to the pin list and the Quartus II software for exact pin locations.

Each I/O bank has dedicated $V_{CCIO}$ pins that determine the voltage standard support in that bank. A single device can support 1.5-V, 1.8-V, 2.5-V, and 3.3-V interfaces; each individual bank can support a different standard. Each I/O bank can support multiple standards with the same $V_{CCIO}$ for input and output pins. For example, when $V_{CCIO}$ is 3.3 V, Bank 3 can support LVTTL, LVCMOS, and 3.3-V PCI. $V_{CCIO}$ powers both the input and output buffers in MAX II devices.

The JTAG pins for MAX II devices are dedicated pins that cannot be used as regular I/O pins. The pins `TMS`, `TDI`, `TDO`, and `TCK` support all the I/O standards shown in Table 2–4 on page 2–27 except for PCI. These pins reside in Bank 1 for all MAX II devices and their I/O standard support is controlled by the $V_{CCIO}$ setting for Bank 1.

## PCI Compliance

The MAX II EPM1270 and EPM2210 devices are compliant with PCI applications as well as all 3.3-V electrical specifications in the *PCI Local Bus Specification Revision 2.2*. These devices are also large enough to support PCI intellectual property (IP) cores. Table 2–5 shows the MAX II device speed grades that meet the PCI timing specifications.

**Table 2–5.** MAX II Devices and Speed Grades that Support 3.3-V PCI Electrical Specifications and Meet PCI Timing

| Device | 33-MHz PCI | 66-MHz PCI |
|---|---|---|
| EPM1270 | All Speed Grades | –3 Speed Grade |
| EPM2210 | All Speed Grades | –3 Speed Grade |

## Schmitt Trigger

The input buffer for each MAX II device I/O pin has an optional Schmitt trigger setting for the 3.3-V and 2.5-V standards. The Schmitt trigger allows input buffers to respond to slow input edge rates with a fast output edge rate. Most importantly, Schmitt triggers provide hysteresis on the input buffer, preventing slow-rising noisy input signals from ringing or oscillating on the input signal driven into the logic array. This provides system noise tolerance on MAX II inputs, but adds a small, nominal input delay.

The JTAG input pins (TMS, TCK, and TDI) have Schmitt trigger buffers that are always enabled.

The TCK input is susceptible to high pulse glitches when the input signal fall time is greater than 200 ns for all I/O standards.

## Output Enable Signals

Each MAX II IOE output buffer supports output enable signals for tri-state control. The output enable signal can originate from the GCLK[3..0] global signals or from the MultiTrack interconnect. The MultiTrack interconnect routes output enable signals and allows for a unique output enable for each output or bidirectional pin.

MAX II devices also provide a chip-wide output enable pin (DEV_OE) to control the output enable for every output pin in the design. An option set before compilation in the Quartus II software controls this pin. This chip-wide output enable uses its own routing resources and does not use any of the four global resources. If this option is turned on, all outputs on the chip operate normally when DEV_OE is asserted. When the pin is deasserted, all outputs are tri-stated. If this option is turned off, the DEV_OE pin is disabled when the device operates in user mode and is available as a user I/O pin.

## Programmable Drive Strength

The output buffer for each MAX II device I/O pin has two levels of programmable drive strength control for each of the LVTTL and LVCMOS I/O standards. Programmable drive strength provides system noise reduction control for high performance I/O designs. Although a separate slew-rate control feature exists, using the lower drive strength setting provides signal slew-rate control to reduce system noise and signal overshoot without the large delay adder associated with the slew-rate control feature. Table 2–6 shows the possible settings for the I/O standards with drive strength control. The Quartus II software uses the maximum current strength as the default setting. The PCI I/O standard is always set at 20 mA with no alternate setting.

**Table 2–6.** Programmable Drive Strength *(Note 1)*

| I/O Standard | IOH/IOL Current Strength Setting (mA) |
|---|---|
| 3.3-V LVTTL | 16 |
| | 8 |
| 3.3-V LVCMOS | 8 |
| | 4 |
| 2.5-V LVTTL/LVCMOS | 14 |
| | 7 |
| 1.8-V LVTTL/LVCMOS | 6 |
| | 3 |
| 1.5-V LVCMOS | 4 |
| | 2 |

**Note to Table 2–6:**

(1) The $I_{OH}$ current strength numbers shown are for a condition of a $V_{OUT} = V_{OH}$ minimum, where the $V_{OH}$ minimum is specified by the I/O standard. The $I_{OL}$ current strength numbers shown are for a condition of a $V_{OUT} = V_{OL}$ maximum, where the $V_{OL}$ maximum is specified by the I/O standard. For 2.5-V LVTTL/LVCMOS, the $I_{OH}$ condition is $V_{OUT}$ = 1.7 V and the $I_{OL}$ condition is $V_{OUT}$ = 0.7 V.

## Slew-Rate Control

The output buffer for each MAX II device I/O pin has a programmable output slew-rate control that can be configured for low noise or high-speed performance. A faster slew rate provides high-speed transitions for high-performance systems. However, these fast transitions may introduce noise transients into the system. A slow slew rate reduces system noise, but adds a nominal output delay to rising and falling edges. The lower the voltage standard (for example, 1.8-V LVTTL) the larger the output delay when slow slew is enabled. Each I/O pin has an individual slew-rate control, allowing the designer to specify the slew rate on a pin-by-pin basis. The slew-rate control affects both the rising and falling edges.

## Open-Drain Output

MAX II devices provide an optional open-drain (equivalent to open-collector) output for each I/O pin. This open-drain output enables the device to provide system-level control signals (for example, interrupt and write enable signals) that can be asserted by any of several devices. This output can also provide an additional wired-OR plane.

## Programmable Ground Pins

Each unused I/O pin on MAX II devices can be used as an additional ground pin. This programmable ground feature does not require the use of the associated LEs in the device. In the Quartus II software, unused pins can be set as programmable GND on a global default basis or they can be individually assigned. Unused pins also have the option of being set as tri-stated input pins.

## Bus Hold

Each MAX II device I/O pin provides an optional bus-hold feature. The bus-hold circuitry can hold the signal on an I/O pin at its last-driven state. Since the bus-hold feature holds the last-driven state of the pin until the next input signal is present, an external pull-up or pull-down resistor is not necessary to hold a signal level when the bus is tri-stated.

The bus-hold circuitry also pulls undriven pins away from the input threshold voltage where noise can cause unintended high-frequency switching. The designer can select this feature individually for each I/O pin. The bus-hold output will drive no higher than $V_{CCIO}$ to prevent overdriving signals. If the bus-hold feature is enabled, the device cannot use the programmable pull-up option.

The bus-hold circuitry uses a resistor to pull the signal level to the last driven state. The *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* gives the specific sustaining current for each $V_{CCIO}$ voltage level driven through this resistor and overdrive current used to identify the next-driven input level.

The bus-hold circuitry is only active after the device has fully initialized. The bus-hold circuit captures the value on the pin present at the moment user mode is entered.

## Programmable Pull-Up Resistor

Each MAX II device I/O pin provides an optional programmable pull-up resistor during user mode. If the designer enables this feature for an I/O pin, the pull-up resistor holds the output to the $V_{CCIO}$ level of the output pin's bank.

The programmable pull-up resistor feature should not be used at the same time as the bus-hold feature on a given I/O pin.

## Programmable Input Delay

The MAX II IOE includes a programmable input delay that is activated to ensure zero hold times. A path where a pin directly drives a register, with minimal routing between the two, may require the delay to ensure zero hold time. However, a path where a pin drives a register through long routing or through combinational logic may not require the delay to achieve a zero hold time. The Quartus II software uses this delay to ensure zero hold times when needed.

## MultiVolt I/O Interface

The MAX II architecture supports the MultiVolt I/O interface feature, which allows MAX II devices in all packages to interface with systems of different supply voltages. The devices have one set of VCC pins for internal operation ($V_{CCINT}$), and up to four sets for input buffers and I/O output driver buffers ($V_{CCIO}$), depending on the number of I/O banks available in the devices where each set of VCC pins powers one I/O bank. The EPM240 and EPM570 devices have two I/O banks respectively while the EPM1270 and EPM2210 devices have four I/O banks respectively.

Connect VCCIO pins to either a 1.5-V, 1.8 V, 2.5-V, or 3.3-V power supply, depending on the output requirements. The output levels are compatible with systems of the same voltage as the power supply (that is, when VCCIO pins are connected to a 1.5-V power supply, the output levels are compatible with 1.5-V systems). When VCCIO pins are connected to a 3.3-V power supply, the output high is 3.3 V and is compatible with 3.3-V or 5.0-V systems. Table 2–7 summarizes MAX II MultiVolt I/O support.

**Table 2–7.** MAX II MultiVolt I/O Support *(Note 1)*

| | Input Signal | | | | | Output Signal | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **VCCIO (V)** | **1.5 V** | **1.8 V** | **2.5 V** | **3.3 V** | **5.0 V** | **1.5 V** | **1.8 V** | **2.5 V** | **3.3 V** | **5.0 V** |
| **1.5** | ✓ | ✓ | ✓ | ✓ | — | ✓ | — | — | — | — |
| **1.8** | ✓ | ✓ | ✓ | ✓ | — | ✓ *(2)* | ✓ | — | — | — |
| **2.5** | — | — | ✓ | ✓ | — | ✓ *(3)* | ✓ *(3)* | ✓ | — | — |
| **3.3** | — | — | ✓ *(4)* | ✓ | ✓ *(5)* | ✓ *(6)* | ✓ *(6)* | ✓ *(6)* | ✓ | ✓ *(7)* |

**Notes to Table 2–7:**

(1) To drive inputs higher than $V_{CCIO}$ but less than 4.0 V including the overshoot, disable the I/O clamp diode. However, to drive 5.0-V inputs to the device, enable the I/O clamp diode to prevent $V_I$ from rising above 4.0 V.

(2) When $V_{CCIO}$ = 1.8 V, a MAX II device can drive a 1.5-V device with 1.8-V tolerant inputs.

(3) When $V_{CCIO}$ = 2.5 V, a MAX II device can drive a 1.5-V or 1.8-V device with 2.5-V tolerant inputs.

(4) When $V_{CCIO}$ = 3.3 V and a 2.5-V input signal feeds an input pin, the VCCIO supply current will be slightly larger than expected.

(5) MAX II devices can be 5.0-V tolerant with the use of an external resistor and the internal I/O clamp diode on the EPM1270 and EPM2210 devices.

(6) When $V_{CCIO}$ = 3.3 V, a MAX II device can drive a 1.5-V, 1.8-V, or 2.5-V device with 3.3-V tolerant inputs.

(7) When $V_{CCIO}$ = 3.3 V, a MAX II device can drive a device with 5.0-V TTL inputs but not 5.0-V CMOS inputs. In the case of 5.0-V CMOS, open-drain setting with internal I/O clamp diode (available only on EPM1270 and EPM2210 devices) and external resistor is required.

For information about output pin source and sink current guidelines, refer to the *AN 428: MAX II CPLD Design Guidelines*.

# Referenced Documents

This chapter referenced the following documents:

- *AN 428: MAX II CPLD Design Guidelines*
- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *Hot Socketing and Power-On Reset in MAX II Devices* chapter in the *MAX II Device Handbook*
- *Using User Flash Memory in MAX II Devices* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 2–8 shows the revision history for this chapter.

**Table 2–8.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 2.2 | ■ Updated Table 2–4 and Table 2–6.<br>■ Updated "I/O Standards and Banks" section.<br>■ Updated New Document Format. | — |
| March 2008, version 2.1 | ■ Updated "Schmitt Trigger" section. | — |
| December 2007, version 2.0 | ■ Updated "Clear and Preset Logic Control" section.<br>■ Updated "MultiVolt Core" section.<br>■ Updated "MultiVolt I/O Interface" section.<br>■ Updated Table 2–7.<br>■ Added "Referenced Documents" section. | Updated document with MAX IIZ information. |
| December 2006, version 1.7 | ■ Minor update in "Internal Oscillator" section. Added document revision history. | — |
| August 2006, version 1.6 | ■ Updated functional description and I/O structure sections. | — |
| July 2006, vervion 1.5 | ■ Minor content and table updates. | — |
| February 2006, version 1.4 | ■ Updated "LAB Control Signals" section.<br>■ Updated "Clear and Preset Logic Control" section.<br>■ Updated "Internal Oscillator" section.<br>■ Updated Table 2–5. | — |
| August 2005, version 1.3 | ■ Removed Note 2 from Table 2-7. | — |
| December 2004, version 1.2 | ■ Added a paragraph to page 2-15. | — |
| June 2004, version 1.1 | ■ Added CFM acronym. Corrected Figure 2-19. | — |

# 3. JTAG and In-System Programmability

MII51003-1.6

## Introduction

This chapter discusses how to use the IEEE Standard 1149.1 Boundary-Scan Test (BST) circuitry in MAX II devices and includes the following sections:


- "IEEE Std. 1149.1 (JTAG) Boundary-Scan Support" on page 3–1
- "In System Programmability" on page 3–4


## IEEE Std. 1149.1 (JTAG) Boundary-Scan Support

All MAX® II devices provide Joint Test Action Group (JTAG) boundary-scan test (BST) circuitry that complies with the IEEE Std. 1149.1-2001 specification. JTAG boundary-scan testing can only be performed at any time after $V_{CCINT}$ and all $V_{CCIO}$ banks have been fully powered and a $t_{CONFIG}$ amount of time has passed. MAX II devices can also use the JTAG port for in-system programming together with either the Quartus® II software or hardware using Programming Object Files (**.pof**), JamTM Standard Test and Programming Language (STAPL) Files (**.jam**), or Jam Byte-Code Files (**.jbc**).

The JTAG pins support 1.5-V, 1.8-V, 2.5-V, or 3.3-V I/O standards. The supported voltage level and standard are determined by the $V_{CCIO}$ of the bank where it resides. The dedicated JTAG pins reside in Bank 1 of all MAX II devices.

MAX II devices support the JTAG instructions shown in Table 3–1.

**Table 3–1.** MAX II JTAG Instructions (Part 1 of 2)

| JTAG Instruction | Instruction Code | Description |
|---|---|---|
| SAMPLE/PRELOAD | 00 0000 0101 | Allows a snapshot of signals at the device pins to be captured and examined during normal device operation, and permits an initial data pattern to be output at the device pins. |
| EXTEST *(1)* | 00 0000 1111 | Allows the external circuitry and board-level interconnects to be tested by forcing a test pattern at the output pins and capturing test results at the input pins. |
| BYPASS | 11 1111 1111 | Places the 1-bit bypass register between the TDI and TDO pins, which allows the BST data to pass synchronously through selected devices to adjacent devices during normal device operation. |
| USERCODE | 00 0000 0111 | Selects the 32-bit USERCODE register and places it between the TDI and TDO pins, allowing the USERCODE to be serially shifted out of TDO. This register defaults to all 1's if not specified in the Quartus II software. |
| IDCODE | 00 0000 0110 | Selects the IDCODE register and places it between TDI and TDO, allowing the IDCODE to be serially shifted out of TDO. |
| HIGHZ *(1)* | 00 0000 1011 | Places the 1-bit bypass register between the TDI and TDO pins, which allows the boundary scan test data to pass synchronously through selected devices to adjacent devices during normal device operation, while tri-stating all of the I/O pins. |

**Table 3–1.** MAX II JTAG Instructions (Part 2 of 2)

| JTAG Instruction | Instruction Code | Description |
|---|---|---|
| CLAMP *(1)* | 00 0000 1010 | Places the 1-bit bypass register between the TDI and TDO pins, which allows the boundary scan test data to pass synchronously through selected devices to adjacent devices during normal device operation, while holding I/O pins to a state defined by the data in the boundary-scan register. |
| USER0 | 00 0000 1100 | This instruction allows you to define the scan chain between TDI and TDO in the MAX II logic array. This instruction is also used for custom logic and JTAG interfaces. |
| USER1 | 00 0000 1110 | This instruction allows you to define the scan chain between TDI and TDO in the MAX II logic array. This instruction is also used for custom logic and JTAG interfaces. |
| IEEE 1532 instructions | *(2)* | IEEE 1532 ISC instructions used when programming a MAX II device via the JTAG port. |

**Notes to Table 3–1:**

(1) HIGHZ, CLAMP, and EXTEST instructions do not disable weak pull-up resistors or bus hold features.

(2) These instructions are shown in the 1532 BSDL files, which will be posted on the Altera® website at www.altera.com when they are available.

WARNING

Unsupported JTAG instructions should not be issued to the MAX II device as this may put the device into an unknown state, requiring a power cycle to recover device operation.

The MAX II device instruction register length is 10 bits and the USERCODE register length is 32 bits. Table 3–2 and Table 3–3 show the boundary-scan register length and device IDCODE information for MAX II devices.

**Table 3–2.** MAX II Boundary-Scan Register Length

| Device | Boundary-Scan Register Length |
|---|---|
| EPM240 | 240 |
| EPM570 | 480 |
| EPM1270 | 636 |
| EPM2210 | 816 |

**Table 3–3.** 32-Bit MAX II Device IDCODE (Part 1 of 2)

| Device | Binary IDCODE (32 Bits) *(1)* | | | | HEX IDCODE |
|---|---|---|---|---|---|
| | Version (4 Bits) | Part Number | Manufacturer Identity (11 Bits) | LSB (1 Bit) *(2)* | |
| EPM240<br>EPM240G | 0000 | 0010 0000 1010 0001 | 000 0110 1110 | 1 | 0x020A10DD |
| EPM570<br>EPM570G | 0000 | 0010 0000 1010 0010 | 000 0110 1110 | 1 | 0x020A20DD |
| EPM1270<br>EPM1270G | 0000 | 0010 0000 1010 0011 | 000 0110 1110 | 1 | 0x020A30DD |
| EPM2210<br>EPM2210G | 0000 | 0010 0000 1010 0100 | 000 0110 1110 | 1 | 0x020A40DD |

**Table 3–3.** 32-Bit MAX II Device IDCODE (Part 2 of 2)

| Device | Binary IDCODE (32 Bits) *(1)* | | | | HEX IDCODE |
|---|---|---|---|---|---|
| | Version (4 Bits) | Part Number | Manufacturer Identity (11 Bits) | LSB (1 Bit) *(2)* | |
| EPM240Z | 0000 | 0010 0000 1010 0101 | 000 0110 1110 | 1 | 0x020A50DD |
| EPM570Z | 0000 | 0010 0000 1010 0110 | 000 0110 1110 | 1 | 0x020A60DD |

**Notes to Table 3–2:**

(1) The most significant bit (MSB) is on the left.

(2) The IDCODE's least significant bit (LSB) is always 1.

For JTAG AC characteristics, refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*.

For more information about JTAG BST, refer to the *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*.

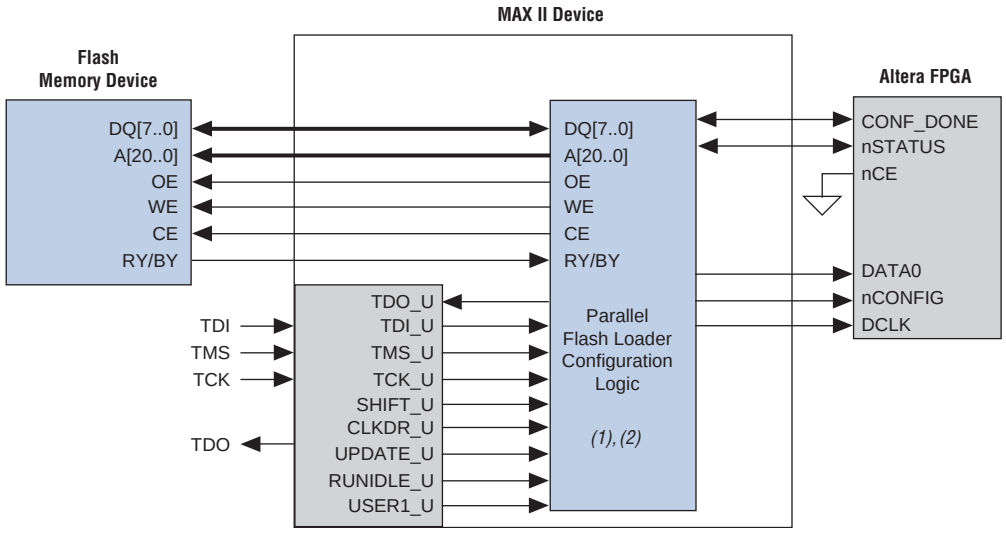
## JTAG Block

The MAX II JTAG block feature allows you to access the JTAG TAP and state signals when either the USER0 or USER1 instruction is issued to the JTAG TAP. The USER0 and USER1 instructions bring the JTAG boundary-scan chain (TDI) through the user logic instead of the MAX II device's boundary-scan cells. Each USER instruction allows for one unique user-defined JTAG chain into the logic array.

### Parallel Flash Loader

The JTAG block ability to interface JTAG to non-JTAG devices is ideal for general-purpose flash memory devices (such as Intel- or Fujitsu-based devices) that require programming during in-circuit test. The flash memory devices can be used for FPGA configuration or be part of system memory. In many cases, the MAX II device is already connected to these devices as the configuration control logic between the FPGA and the flash device. Unlike ISP-capable CPLD devices, bulk flash devices do not have JTAG TAP pins or connections. For small flash devices, it is common to use the serial JTAG scan chain of a connected device to program the non-JTAG flash device. This is slow and inefficient in most cases and impractical for large parallel flash devices. Using the MAX II device's JTAG block as a parallel flash loader, with the Quartus II software, to program and verify flash contents provides a fast and cost-effective means of in-circuit programming during test. Figure 3–1 shows MAX II being used as a parallel flash loader.

**Figure 3–1.** MAX II Parallel Flash Loader




**Notes to Figure 3–1:**

(1) This block is implemented in LEs.

(2) This function is supported in the Quartus II software.

# In System Programmability

MAX II devices can be programmed in-system via the industry standard 4-pin IEEE Std. 1149.1 (JTAG) interface. In-system programmability (ISP) offers quick, efficient iterations during design development and debugging cycles. The logic, circuitry, and interconnects in the MAX II architecture are configured with flash-based SRAM configuration elements. These SRAM elements require configuration data to be loaded each time the device is powered. The process of loading the SRAM data is called configuration. The on-chip configuration flash memory (CFM) block stores the SRAM element's configuration data. The CFM block stores the design's configuration pattern in a reprogrammable flash array. During ISP, the MAX II JTAG and ISP circuitry programs the design pattern into the CFM block's non-volatile flash array.

The MAX II JTAG and ISP controller internally generate the high programming voltages required to program the CFM cells, allowing in-system programming with any of the recommended operating external voltage supplies (that is, 3.3 V/2.5 V or 1.8 V for the MAX IIG and MAX IIZ devices). ISP can be performed anytime after $V_{CCINT}$ and all $V_{CCIO}$ banks have been fully powered and the device has completed the configuration power-up time. By default, during in-system programming, the I/O pins are tri-stated and weakly pulled-up to $V_{CCIO}$ to eliminate board conflicts. The in-system programming clamp and real-time ISP feature allow user control of I/O state or behavior during ISP.

For more information, refer to "In-System Programming Clamp" on page 3–6 and "Real-Time ISP" on page 3–7.

These devices also offer an `ISP_DONE` bit that provides safe operation when in-system programming is interrupted. This `ISP_DONE` bit, which is the last bit programmed, prevents all I/O pins from driving until the bit is programmed.

## IEEE 1532 Support

The JTAG circuitry and ISP instruction set in MAX II devices is compliant to the IEEE 1532-2002 programming specification. This provides industry-standard hardware and software for in-system programming among multiple vendor programmable logic devices (PLDs) in a JTAG chain.

The MAX II 1532 BSDL files will be released on the Altera website when available.

## Jam Standard Test and Programming Language (STAPL)

The Jam STAPL JEDEC standard, JESD71, can be used to program MAX II devices with in-circuit testers, PCs, or embedded processors. The Jam byte code is also supported for MAX II devices. These software programming protocols provide a compact embedded solution for programming MAX II devices.

For more information, refer to the *Using Jam STAPL for ISP via an Embedded Processor* chapter in the *MAX II Device Handbook*.

## Programming Sequence

During in-system programming, 1532 instructions, addresses, and data are shifted into the MAX II device through the `TDI` input pin. Data is shifted out through the `TDO` output pin and compared against the expected data. Programming a pattern into the device requires the following six ISP steps. A stand-alone verification of a programmed pattern involves only stages 1, 2, 5, and 6. These steps are automatically executed by third-party programmers, the Quartus II software, or the Jam STAPL and Jam Byte-Code Players.

1. *Enter ISP*—The enter ISP stage ensures that the I/O pins transition smoothly from user mode to ISP mode.
2. *Check ID*—Before any program or verify process, the silicon ID is checked. The time required to read this silicon ID is relatively small compared to the overall programming time.
3. *Sector Erase*—Erasing the device in-system involves shifting in the instruction to erase the device and applying an erase pulse(s). The erase pulse is automatically generated internally by waiting in the run/test/idle state for the specified erase pulse time of 500 ms for the CFM block and 500 ms for each sector of the UFM block.
4. *Program*—Programming the device in-system involves shifting in the address, data, and program instruction and generating the program pulse to program the flash cells. The program pulse is automatically generated internally by waiting in the run/test/idle state for the specified program pulse time of 75 µs. This process is repeated for each address in the CFM and UFM blocks.
5. *Verify*—Verifying a MAX II device in-system involves shifting in addresses, applying the verify instruction to generate the read pulse, and shifting out the data for comparison. This process is repeated for each CFM and UFM address.
6. *Exit ISP*—An exit ISP stage ensures that the I/O pins transition smoothly from ISP mode to user mode.

Table 3–4 shows the programming times for MAX II devices using in-circuit testers to execute the algorithm vectors in hardware. Software-based programming tools used with download cables are slightly slower because of data processing and transfer limitations.

**Table 3–4.** MAX II Device Family Programming Times

| Description | EPM240<br>EPM240G<br>EPM240Z | EPM570<br>EPM570G<br>EPM570Z | EPM1270<br>EPM1270G | EPM2210<br>EPM2210G | Unit |
|---|---|---|---|---|---|
| Erase + Program (1 MHz) | 1.72 | 2.16 | 2.90 | 3.92 | sec |
| Erase + Program (10 MHz) | 1.65 | 1.99 | 2.58 | 3.40 | sec |
| Verify (1 MHz) | 0.09 | 0.17 | 0.30 | 0.49 | sec |
| Verify (10 MHz) | 0.01 | 0.02 | 0.03 | 0.05 | sec |
| Complete Program Cycle (1 MHz) | 1.81 | 2.33 | 3.20 | 4.41 | sec |
| Complete Program Cycle (10 MHz) | 1.66 | 2.01 | 2.61 | 3.45 | sec |

## UFM Programming

The Quartus II software, with the use of POF, Jam, or JBC files, supports programming of the user flash memory (UFM) block independent of the logic array design pattern stored in the CFM block. This allows updating or reading UFM contents through ISP without altering the current logic array design, or vice versa. By default, these programming files and methods will program the entire flash memory contents, which includes the CFM block and UFM contents. The stand-alone embedded Jam STAPL player and Jam Byte-Code Player provides action commands for programming or reading the entire flash memory (UFM and CFM together) or each independently.

For more information, refer to the *Using Jam STAPL for ISP via an Embedded Processor* chapter in the *MAX II Device Handbook*.

## In-System Programming Clamp

By default, the IEEE 1532 instruction used for entering ISP automatically tri-states all I/O pins with weak pull-up resistors for the duration of the ISP sequence. However, some systems may require certain pins on MAX II devices to maintain a specific DC logic level during an in-field update. For these systems, an optional in-system programming clamp instruction exists in MAX II circuitry to control I/O behavior during the ISP sequence. The in-system programming clamp instruction enables the device to sample and sustain the value on an output pin (an input pin would remain tri-stated if sampled) or to explicitly set a logic high, logic low, or tri-state value on any pin. Setting these options is controlled on an individual pin basis using the Quartus II software.

For more information, refer to the *Real-Time ISP and ISP Clamp for MAX II Devices* chapter in the *MAX II Device Handbook*.

## Real-Time ISP

For systems that require more than DC logic level control of I/O pins, the real-time ISP feature allows you to update the CFM block with a new design image while the current design continues to operate in the SRAM logic array and I/O pins. A new programming file is updated into the MAX II device without halting the original design's operation, saving down-time costs for remote or field upgrades. The updated CFM block configures the new design into the SRAM upon the next power cycle. It is also possible to execute an immediate configuration of the SRAM without a power cycle by using a specific sequence of ISP commands. The configuration of SRAM without a power cycle takes a specific amount of time ($t_{CONFIG}$). During this time, the I/O pins are tri-stated and weakly pulled-up to $V_{CCIO}$.

## Design Security

All MAX II devices contain a programmable security bit that controls access to the data programmed into the CFM block. When this bit is programmed, design programming information, stored in the CFM block, cannot be copied or retrieved. This feature provides a high level of design security because programmed data within flash memory cells is invisible. The security bit that controls this function, as well as all other programmed data, is reset only when the device is erased. The SRAM is also invisible and cannot be accessed regardless of the security bit setting. The UFM block data is not protected by the security bit and is accessible through JTAG or logic array connections.

## Programming with External Hardware

MAX II devices can be programmed by downloading the information via in-circuit testers, embedded processors, the Altera® ByteblasterMV™, MasterBlaster™, ByteBlaster™ II, and USB-Blaster cables.

BP Microsystems, System General, and other programming hardware manufacturers provide programming support for Altera devices. Check their websites for device support information.

# Referenced Documents

This chapter references the following documents:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*
- *Real-Time ISP and ISP Clamp for MAX II Devices* chapter in the *MAX II Device Handbook*
- *Using Jam STAPL for ISP via an Embedded Processor* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 3–5 shows the revision history for this chapter.

**Table 3–5.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.6 | ■ Updated New Document Format. | — |
| December 2007, version 1.5 | ■ Added warning note after Table 3–1.<br>■ Updated Table 3–3 and Table 3–4.<br>■ Added "Referenced Documents" section. | — |
| December 2006, version 1.4 | ■ Added document revision history. | — |
| June 2005, version 1.3 | ■ Added text and Table 3-4. | — |
| June 2005, version 1.3 | ■ Updated text on pages 3-5 to 3-8. | — |
| June 2004, version 1.1 | ■ Corrected Figure 3-1. Added CFM acronym. | — |

# 4. Hot Socketing and Power-On Reset in MAX II Devices

MII51004-2.1

## Introduction

MAX® II devices offer hot socketing, also known as hot plug-in or hot swap, and power sequencing support. Designers can insert or remove a MAX II board in a system during operation without undesirable effects to the system bus. The hot socketing feature removes some of the difficulties designers face when using components on printed circuit boards (PCBs) that contain a mixture of 3.3-, 2.5-, 1.8-, and 1.5-V devices.

The MAX II device hot socketing feature provides:

- Board or device insertion and removal
- Support for any power-up sequence
- Non-intrusive I/O buffers to system buses during hot insertion

This chapter contains the following sections:


- "MAX II Hot-Socketing Specifications" on page 4–1
- "Power-On Reset Circuitry" on page 4–5


## MAX II Hot-Socketing Specifications

MAX II devices offer all three of the features required for the hot-socketing capability listed above without any external components or special design requirements. The following are hot-socketing specifications:

- The device can be driven before and during power-up or power-down without any damage to the device itself.
- I/O pins remain tri-stated during power-up. The device does not drive out before or during power-up, thereby affecting other buses in operation.
- Signal pins do not drive the $V_{CCIO}$ or $V_{CCINT}$ power supplies. External input signals to device I/O pins do not power the device $V_{CCIO}$ or $V_{CCINT}$ power supplies via internal paths. This is true if the $V_{CCINT}$ and the $V_{CCIO}$ supplies are held at GND.

Altera uses GND as reference for the hot-socketing and I/O buffers circuitry designs. You must connect the GND between boards before connecting the $V_{CCINT}$ and the $V_{CCIO}$ power supplies to ensure device reliability and compliance to the hot-socketing specifications.

### Devices Can Be Driven before Power-Up

Signals can be driven into the MAX II device I/O pins and GCLK[3..0] pins before or during power-up or power-down without damaging the device. MAX II devices support any power-up or power-down sequence ($V_{CCIO1}$, $V_{CCIO2}$, $V_{CCIO3}$, $V_{CCIO4}$, $V_{CCINT}$), simplifying the system-level design.

### I/O Pins Remain Tri-Stated during Power-Up

A device that does not support hot-socketing may interrupt system operation or cause contention by driving out before or during power-up. In a hot socketing situation, the MAX II device's output buffers are turned off during system power-up. MAX II devices do not drive out until the device attains proper operating conditions and is fully configured. Refer to "Power-On Reset Circuitry" on page 4–5 for information about turn-on voltages.

### Signal Pins Do Not Drive the $V_{CCIO}$ or $V_{CCINT}$ Power Supplies

MAX II devices do not have a current path from I/O pins or GCLK[3..0] pins to the $V_{CCIO}$ or $V_{CCINT}$ pins before or during power-up. A MAX II device may be inserted into (or removed from) a system board that was powered up without damaging or interfering with system-board operation. When hot socketing, MAX II devices may have a minimal effect on the signal integrity of the backplane.

### AC and DC Specifications

You can power up or power down the $V_{CCIO}$ and $V_{CCINT}$ pins in any sequence. During hot socketing, the I/O pin capacitance is less than 8 pF. MAX II devices meet the following hot socketing specifications:

- The hot socketing DC specification is: $| I_{IOPIN} | < 300$ µA.
- The hot socketing AC specification is: $| I_{IOPIN} | < 8$ mA for 10 ns or less.

MAX II devices are immune to latch-up when hot socketing. If the TCK JTAG input pin is driven high during hot socketing, the current on that pin might exceed the specifications above.

$I_{IOPIN}$ is the current at any user I/O pin on the device. The AC specification applies when the device is being powered up or powered down. This specification takes into account the pin capacitance but not board trace and external loading capacitance. Additional capacitance for trace, connector, and loading must be taken into consideration separately. The peak current duration due to power-up transients is 10 ns or less.

The DC specification applies when all $V_{CC}$ supplies to the device are stable in the powered-up or powered-down conditions.

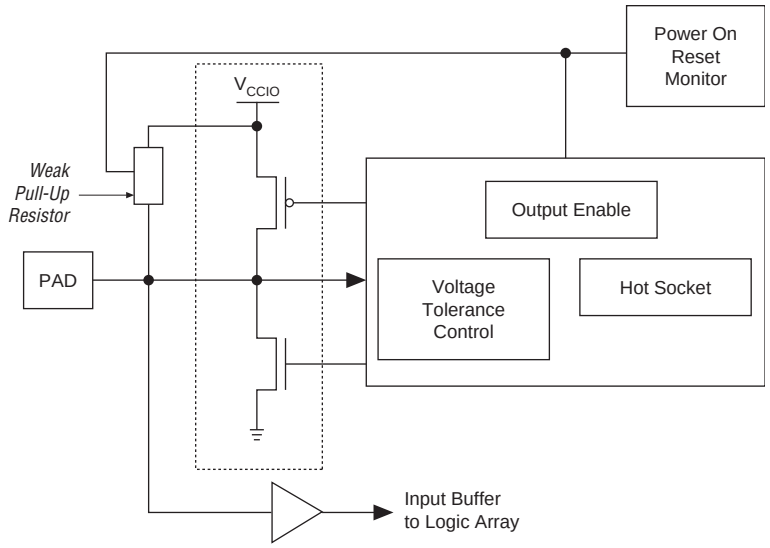
## Hot Socketing Feature Implementation in MAX II Devices

The hot socketing feature turns off (tri-states) the output buffer during the power-up event (either $V_{CCINT}$ or $V_{CCIO}$ supplies) or power-down event. The hot-socket circuit generates an internal HOTSCKT signal when either $V_{CCINT}$ or $V_{CCIO}$ is below the threshold voltage during power-up or power-down. The HOTSCKT signal cuts off the output buffer to make sure that no DC current (except for weak pull-up leaking) leaks through the pin. When $V_{CC}$ ramps up very slowly during power-up, $V_{CC}$ may still be relatively low even after the power-on reset (POR) signal is released and device configuration is complete.

Make sure that the $V_{CCINT}$ is within the recommended operating range even though SRAM download has completed.

Each I/O and clock pin has the circuitry shown in Figure 4–1.

**Figure 4–1.** Hot Socketing Circuit Block Diagram for MAX II Devices

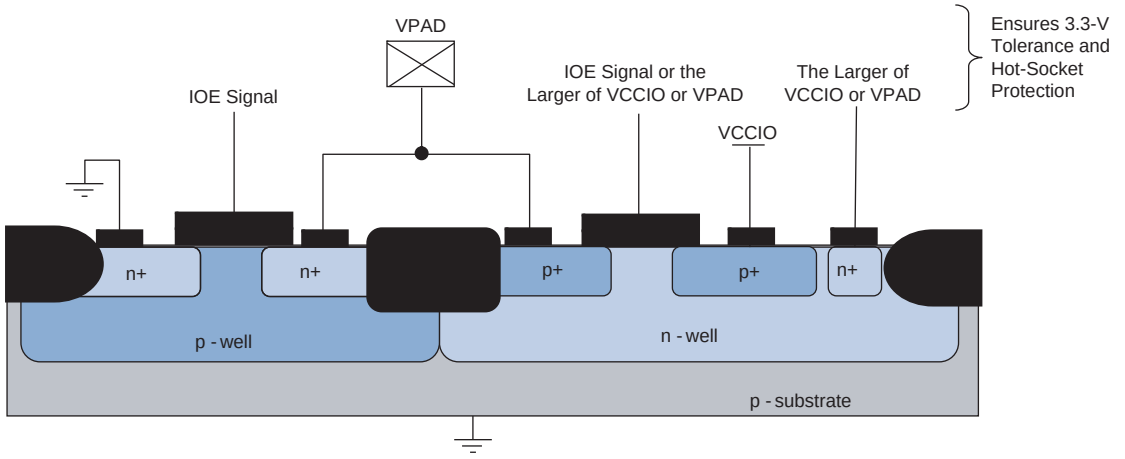



The POR circuit monitors $V_{CCINT}$ and $V_{CCIO}$ voltage levels and keeps I/O pins tri-stated until the device has completed its flash memory configuration of the SRAM logic. The weak pull-up resistor (R) from the I/O pin to $V_{CCIO}$ is enabled during download to keep the I/O pins from floating. The 3.3-V tolerance control circuit permits the I/O pins to be driven by 3.3 V before $V_{CCIO}$ and/or $V_{CCINT}$ are powered, and it prevents the I/O pins from driving out when the device is not fully powered or operational. The hot socket circuit prevents I/O pins from internally powering $V_{CCIO}$ and $V_{CCINT}$ when driven by external signals before the device is powered.

For information about 5.0-V tolerance, refer to the *Using MAX II Devices in Multi-Voltage Systems* chapter in the *MAX II Device Handbook*.

Figure 4–2 shows a transistor-level cross section of the MAX II device I/O buffers. This design ensures that the output buffers do not drive when $V_{CCIO}$ is powered before $V_{CCINT}$ or if the I/O pad voltage is higher than $V_{CCIO}$. This also applies for sudden voltage spikes during hot insertion. The $V_{PAD}$ leakage current charges the 3.3-V tolerant circuit capacitance.

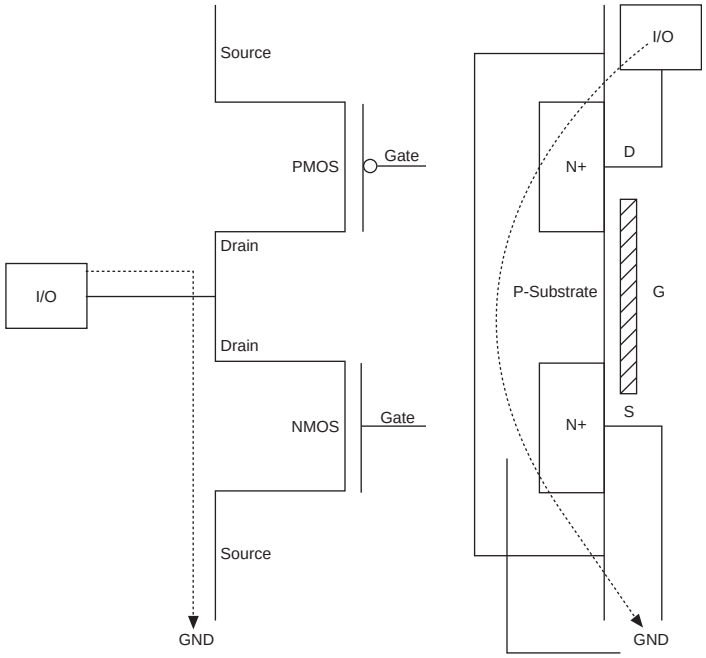
**Figure 4–2.** Transistor-Level Diagram of MAX II Device I/O Buffers




The CMOS output drivers in the I/O pins intrinsically provide electrostatic discharge (ESD) protection. There are two cases to consider for ESD voltage strikes: positive voltage zap and negative voltage zap.

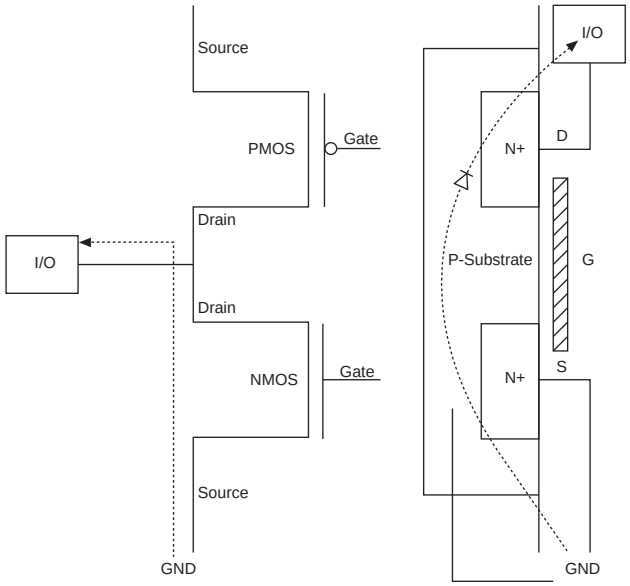
A positive ESD voltage zap occurs when a positive voltage is present on an I/O pin due to an ESD charge event. This can cause the N+ (Drain)/ P-Substrate junction of the N-channel drain to break down and the N+ (Drain)/P-Substrate/N+ (Source) intrinsic bipolar transistor turn on to discharge ESD current from I/O pin to GND. The dashed line (see Figure 4–3) shows the ESD current discharge path during a positive ESD zap.

**Figure 4–3.** ESD Protection During Positive Voltage Zap



When the I/O pin receives a negative ESD zap at the pin that is less than –0.7 V (0.7 V is the voltage drop across a diode), the intrinsic P-Substrate/N+ drain diode is forward biased. Therefore, the discharge ESD current path is from GND to the I/O pin, as shown in Figure 4–4.

**Figure 4–4.** ESD Protection During Negative Voltage Zap




# Power-On Reset Circuitry

MAX II devices have POR circuits to monitor $V_{CCINT}$ and $V_{CCIO}$ voltage levels during power-up. The POR circuit monitors these voltages, triggering download from the non-volatile configuration flash memory (CFM) block to the SRAM logic, maintaining tri-state of the I/O pins (with weak pull-up resistors enabled) before and during this process. When the MAX II device enters user mode, the POR circuit releases the I/O pins to user functionality. The POR circuit of the MAX II (except MAX IIZ) device continues to monitor the $V_{CCINT}$ voltage level to detect a brown-out condition. The POR circuit of the MAX IIZ device does not monitor the $V_{CCINT}$ voltage level after the device enters into user mode. More details are provided in the following sub-sections.

## Power-Up Characteristics

When power is applied to a MAX II device, the POR circuit monitors $V_{CCINT}$ and begins SRAM download at an approximate voltage of 1.7 V or 1.55 V for MAX IIG and MAX IIZ devices. From this voltage reference, SRAM download and entry into user mode takes 200 to 450 µs maximum, depending on device density. This period of time is specified as $t_{CONFIG}$ in the power-up timing section of the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*.

Entry into user mode is gated by whether all $V_{CCIO}$ banks are powered with sufficient operating voltage. If $V_{CCIN}$T and $V_{CCIO}$ are powered simultaneously, the device enters user mode within the $t_{CONFIG}$ specifications. If $V_{CCIO}$ is powered more than $t_{CONFIG}$ after $V_{CCINT}$, the device does not enter user mode until 2 µs after all $V_{CCIO}$ banks are powered.

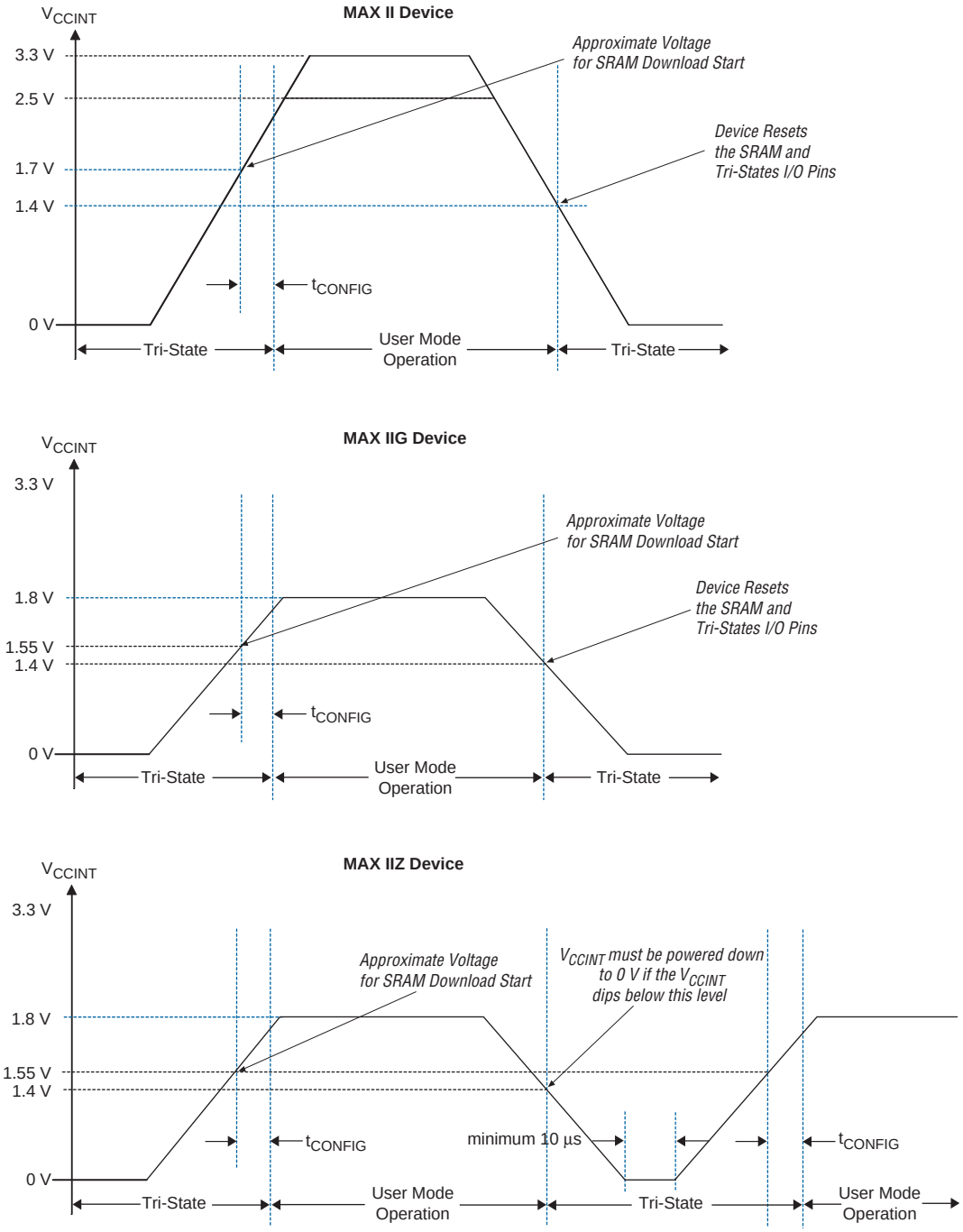
For MAX II and MAX IIG devices, when in user mode, the POR circuitry continues to monitor the $V_{CCINT}$ (but not $V_{CCIO}$) voltage level to detect a brown-out condition. If there is a $V_{CCINT}$ voltage sag at or below 1.4 V during user mode, the POR circuit resets the SRAM and tri-states the I/O pins. Once $V_{CCINT}$ rises back to approximately 1.7 V (or 1.55 V for MAX IIG devices), the SRAM download restarts and the device begins to operate after $t_{CONFIG}$ time has passed.

For MAX IIZ devices, the POR circuitry does not monitor the $V_{CCINT}$ and $V_{CCIO}$ voltage levels after the device enters user mode. If there is a $V_{CCINT}$ voltage sag below 1.4 V during user mode, the functionality of the device will not be guaranteed and you must power down the $V_{CCINT}$ to 0 V for a minimum of 10 µs before powering the $V_{CCINT}$ and $V_{CCIO}$ up again. Once $V_{CCINT}$ rises from 0 V back to approximately 1.55 V, the SRAM download restarts and the device begins to operate after $t_{CONFIG}$ time has passed.

Figure 4–5 shows the voltages for POR of MAX II, MAX IIG, and MAX IIZ devices during power-up into user mode and from user mode to power-down or brown-out.

All $V_{CCINT}$ and $V_{CCIO}$ pins of all banks must be powered on MAX II devices before entering user mode.

**Figure 4–5.** Power-Up Characteristics for MAX II, MAX IIG, and MAX IIZ Devices *(Note 1), (2)*




**Notes to Figure 4–5:**

(1) Time scale is relative.

(2) Figure 4–5 assumes all $V_{CCIO}$ banks power up simultaneously with the $V_{CCINT}$ profile shown. If not, $t_{CONFIG}$ stretches out until all $V_{CCIO}$ banks are powered.

After SRAM configuration, all registers in the device are cleared and released into user function before I/O tri-states are released. To release clears after tri-states are released, use the DEV_CLRn pin option. To hold the tri-states beyond the power-up configuration time, use the DEV_OE pin option.

# Referenced Documents

This chapter refereces the following documents:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *Using MAX II Devices in Multi-Voltage Systems* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 4–1 shows the revision history for this chapter.

**Table 4–1.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version2.1 | ■ Updated "MAX II Hot-Socketing Specifications" and "Power-On Reset Circuitry" sections.<br>■ Updated New Document Format. | — |
| December 2007, version 2.0 | ■ Updated "Hot Socketing Feature Implementation in MAX II Devices" section.<br>■ Updated "Power-On Reset Circuitry" section.<br>■ Updated Figure 4–5.<br>■ Added "Referenced Documents" section. | Updated document with MAX IIZ information. |
| December 2006, version 1.5 | ■ Added document revision history. | — |
| February 2006, version 1.4 | ■ Updated "MAX II Hot-Socketing Specifications" section.<br>■ Updated "AC and DC Specifications" section.<br>■ Updated "Power-On Reset Circuitry" section. | — |
| June 2005, version 1.3 | ■ Updated AC and DC specifications on page 4-2. | — |
| December 2004, version 1.2 | ■ Added content to Power-Up Characteristics section.<br>■ Updated Figure 4-5. | — |
| June 2004, version 1.1 | ■ Corrected Figure 4-2. | — |

# 5. DC and Switching Characteristics

MII51005-2.5

## Introduction

System designers must consider the recommended DC and switching conditions discussed in this chapter to maintain the highest possible performance and reliability of the MAX® II devices. This chapter contains the following sections:


- "Operating Conditions" on page 5–1
- "Power Consumption" on page 5–8
- "Timing Model and Specifications" on page 5–8


## Operating Conditions

Table 5–1 through Table 5–12 provide information about absolute maximum ratings, recommended operating conditions, DC electrical characteristics, and other specifications for MAX II devices.

### Absolute Maximum Ratings

Table 5–1 shows the absolute maximum ratings for the MAX II device family.

**Table 5–1.** MAX II Device Absolute Maximum Ratings *(Note 1)*, *(2)*

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{CCINT}$ | Internal supply voltage *(3)* | With respect to ground | –0.5 | 4.6 | V |
| $V_{CCIO}$ | I/O supply voltage | — | –0.5 | 4.6 | V |
| $V_I$ | DC input voltage | — | –0.5 | 4.6 | V |
| $I_{OUT}$ | DC output current, per pin *(4)* | — | –25 | 25 | mA |
| $T_{STG}$ | Storage temperature | No bias | –65 | 150 | °C |
| $T_{AMB}$ | Ambient temperature | Under bias *(5)* | –65 | 135 | °C |
| $T_J$ | Junction temperature | TQFP and BGA packages under bias | — | 135 | °C |

**Notes to Table 5–1:**

(1) Refer to the *Operating Requirements for Altera Devices Data Sheet*.

(2) Conditions beyond those listed in Table 5–1 may cause permanent damage to a device. Additionally, device operation at the absolute maximum ratings for extended periods of time may have adverse affects on the device.

(3) Maximum $V_{CCINT}$ for MAX II devices is 4.6 V. For MAX IIG and MAX IIZ devices, it is 2.4 V.

(4) Refer to *AN 286: Implementing LED Drivers in MAX & MAX II Devices* for more information about the maximum source and sink current for MAX II devices.

(5) Refer to Table 5–2 for information about "under bias" conditions.

## Recommended Operating Conditions

Table 5–2 shows the MAX II device family recommended operating conditions.

**Table 5–2.** MAX II Device Recommended Operating Conditions

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{CCINT}$ *(1)* | 3.3-V supply voltage for internal logic and ISP | MAX II devices | 3.00 | 3.60 | V |
| | 2.5-V supply voltage for internal logic and ISP | MAX II devices | 2.375 | 2.625 | V |
| | 1.8-V supply voltage for internal logic and ISP | MAX IIG and MAX IIZ devices | 1.71 | 1.89 | V |
| $V_{CCIO}$ *(1)* | Supply voltage for I/O buffers, 3.3-V operation | — | 3.00 | 3.60 | V |
| | Supply voltage for I/O buffers, 2.5-V operation | — | 2.375 | 2.625 | V |
| | Supply voltage for I/O buffers, 1.8-V operation | — | 1.71 | 1.89 | V |
| | Supply voltage for I/O buffers, 1.5-V operation | — | 1.425 | 1.575 | V |
| $V_I$ | Input voltage | *(2)*, *(3)*, *(4)* | –0.5 | 4.0 | V |
| $V_O$ | Output voltage | — | 0 | $V_{CCIO}$ | V |
| $T_J$ | Operating junction temperature | Commercial range | 0 | 85 | °C |
| | | Industrial range | –40 | 100 | °C |
| | | Extended range *(5)* | –40 | 125 | °C |

**Notes to Table 5–2:**

(1) MAX II device in-system programming and/or user flash memory (UFM) programming via JTAG or logic array is not guaranteed outside the recommended operating conditions (for example, if brown-out occurs in the system during a potential write/program sequence to the UFM, users are recommended to read back UFM contents and verify against the intended write data).

(2) Minimum DC input is –0.5 V. During transitions, the inputs may undershoot to –2.0 V for input currents less than 100 mA and periods shorter than 20 ns.

(3) During transitions, the inputs may overshoot to the voltages shown in the following table based upon input duty cycle. The DC case is equivalent to 100% duty cycle. For more information about 5.0-V tolerance, refer to the *Using MAX II Devices in Multi-Voltage Systems* chapter in the *MAX II Device Handbook*.

| $V_{IN}$ | Max. Duty Cycle |
|---|---|
| 4.0 V | 100% (DC) |
| 4.1 | 90% |
| 4.2 | 50% |
| 4.3 | 30% |
| 4.4 | 17% |
| 4.5 | 10% |

(4) All pins, including clock, I/O, and JTAG pins, may be driven before $V_{CCINT}$ and $V_{CCIO}$ are powered.

(5) For the extended temperature range of 100 to 125° C, MAX II UFM programming (erase/write) is only supported via the JTAG interface. UFM programming via the logic array interface is not guaranteed in this range.

## Programming/Erasure Specifications

Table 5–3 shows the MAX II device family programming/erasure specifications.

**Table 5–3.** MAX II Device Programming/Erasure Specifications

| Parameter | Minimum | Typical | Maximum | Unit |
|---|---|---|---|---|
| Erase and reprogram cycles | — | — | 100 *(1)* | Cycles |

**Note to Table 5–3:**

(1) This specification applies to the UFM and configuration flash memory (CFM) blocks.

## DC Electrical Characteristics

Table 5–4 shows the MAX II device family DC electrical characteristics.

**Table 5–4.** MAX II Device DC Electrical Characteristics *(Note 1)* (Part 1 of 2)

| Symbol | Parameter | Conditions | Minimum | Typical | Maximum | Unit |
|---|---|---|---|---|---|---|
| $I_I$ | Input pin leakage current | $V_I = V_{CCIO}$ max to 0 V *(2)* | –10 | — | 10 | µA |
| $I_{OZ}$ | Tri-stated I/O pin leakage current | $V_O = V_{CCIO}$ max to 0 V *(2)* | –10 | — | 10 | µA |
| $I_{CCSTANDBY}$ | $V_{CCINT}$ supply current (standby) *(3)* | MAX II devices | — | 12 | — | mA |
| | | MAX IIG devices | — | 2 | — | mA |
| | | EPM240Z (Commercial grade) *(4)* | — | 25 | 90 | µA |
| | | EPM240Z (Industrial grade) *(5)* | — | 25 | 139 | µA |
| | | EPM570Z (Commercial grade) *(4)* | — | 27 | 96 | µA |
| | | EPM570Z (Industrial grade) *(5)* | — | 27 | 152 | µA |
| $V_{SCHMITT}$ *(6)* | Hysteresis for Schmitt trigger input *(7)* | $V_{CCIO}$ = 3.3 V | — | 400 | — | mV |
| | | $V_{CCIO}$ = 2.5 V | — | 190 | — | mV |
| $I_{CCPOWERUP}$ | $V_{CCINT}$ supply current during power-up *(8)* | MAX II devices | — | 55 | — | mA |
| | | MAX IIG and MAX IIZ devices | — | 40 | — | mA |
| $R_{PULLUP}$ | Value of I/O pin pull-up resistor during user mode and in-system programming | $V_{CCIO}$ = 3.3 V *(9)* | 5 | — | 25 | kΩ |
| | | $V_{CCIO}$ = 2.5 V *(9)* | 10 | — | 40 | kΩ |
| | | $V_{CCIO}$ = 1.8 V *(9)* | 25 | — | 60 | kΩ |
| | | $V_{CCIO}$ = 1.5 V *(9)* | 45 | — | 95 | kΩ |

**Table 5–4.** MAX II Device DC Electrical Characteristics *(Note 1)* (Part 2 of 2)

| Symbol | Parameter | Conditions | Minimum | Typical | Maximum | Unit |
|---|---|---|---|---|---|---|
| $I_{PULLUP}$ | I/O pin pull-up resistor current when I/O is unprogrammed | — | — | — | 300 | µA |
| $C_{IO}$ | Input capacitance for user I/O pin | — | — | — | 8 | pF |
| $C_{GCLK}$ | Input capacitance for dual-purpose GCLK/user I/O pin | — | — | — | 8 | pF |

**Notes to Table 5–4:**

(1) Typical values are for $T_A$ = 25°C, $V_{CCINT}$ = 3.3 or 2.5 V, and $V_{CCIO}$ = 1.5 V, 1.8 V, 2.5 V, or 3.3 V.

(2) This value is specified for normal device operation. The value may vary during power-up. This applies for all $V_{CCIO}$ settings (3.3, 2.5, 1.8, and 1.5 V).

(3) $V_I$ = ground, no load, no toggling inputs.

(4) Commercial temperature ranges from 0°C to 85°C with maximum current at 85°C.

(5) Industrial temperature ranges from –40°C to 100°C with maximum current at 100°C.

(6) This value applies to commercial and industrial range devices. For extended temperature range devices, the $V_{SCHMITT}$ typical value is 300 mV for $V_{CCIO}$ = 3.3 V and 120 mV for $V_{CCIO}$ = 2.5 V.

(7) The TCK input is susceptible to high pulse glitches when the input signal fall time is greater than 200 ns for all I/O standards.

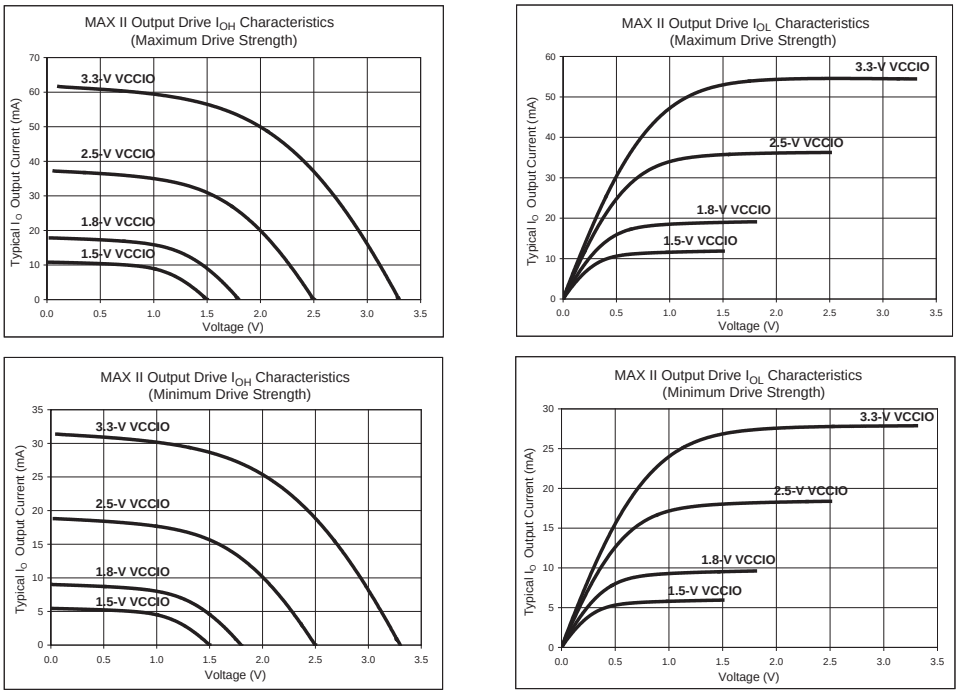
(8) This is a peak current value with a maximum duration of $t_{CONFIG}$ time.

(9) Pin pull-up resistance values will lower if an external source drives the pin higher than $V_{CCIO}$.

## Output Drive Characteristics

Figure 5–1 shows the typical drive strength characteristics of MAX II devices.

**Figure 5–1.** Output Drive Characteristics of MAX II Devices




**Note to Figure 5–1:**

(1) The DC output current per pin is subject to the absolute maximum rating of Table 5–1.

## I/O Standard Specifications

Table 5–5 through Table 5–10 show the MAX II device family I/O standard specifications.

**Table 5–5.** 3.3-V LVTTL Specifications

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{CCIO}$ | I/O supply voltage | — | 3.0 | 3.6 | V |
| $V_{IH}$ | High-level input voltage | — | 1.7 | 4.0 | V |
| $V_{IL}$ | Low-level input voltage | — | –0.5 | 0.8 | V |
| $V_{OH}$ | High-level output voltage | IOH = –4 mA *(1)* | 2.4 | — | V |
| $V_{OL}$ | Low-level output voltage | IOL = 4 mA *(1)* | — | 0.45 | V |

**Table 5–6.** 3.3-V LVCMOS Specifications (Part 1 of 2)

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{CCIO}$ | I/O supply voltage | — | 3.0 | 3.6 | V |
| $V_{IH}$ | High-level input voltage | — | 1.7 | 4.0 | V |
| $V_{IL}$ | Low-level input voltage | — | –0.5 | 0.8 | V |

**Table 5–6.** 3.3-V LVCMOS Specifications (Part 2 of 2)

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{OH}$ | High-level output voltage | $V_{CCIO}$ = 3.0, IOH = –0.1 mA *(1)* | $V_{CCIO}$ – 0.2 | — | V |
| $V_{OL}$ | Low-level output voltage | $V_{CCIO}$ = 3.0, IOL = 0.1 mA *(1)* | — | 0.2 | V |

**Table 5–7.** 2.5-V I/O Specifications

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{CCIO}$ | I/O supply voltage | — | 2.375 | 2.625 | V |
| $V_{IH}$ | High-level input voltage | — | 1.7 | 4.0 | V |
| $V_{IL}$ | Low-level input voltage | — | –0.5 | 0.7 | V |
| $V_{OH}$ | High-level output voltage | IOH = –0.1 mA *(1)* | 2.1 | — | V |
| | | IOH = –1 mA *(1)* | 2.0 | — | V |
| | | IOH = –2 mA *(1)* | 1.7 | — | V |
| $V_{OL}$ | Low-level output voltage | IOL = 0.1 mA *(1)* | — | 0.2 | V |
| | | IOL = 1 mA *(1)* | — | 0.4 | V |
| | | IOL = 2 mA *(1)* | — | 0.7 | V |

**Table 5–8.** 1.8-V I/O Specifications

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{CCIO}$ | I/O supply voltage | — | 1.71 | 1.89 | V |
| $V_{IH}$ | High-level input voltage | — | 0.65 × $V_{CCIO}$ | 2.25 *(2)* | V |
| $V_{IL}$ | Low-level input voltage | — | –0.3 | 0.35 × $V_{CCIO}$ | V |
| $V_{OH}$ | High-level output voltage | IOH = –2 mA *(1)* | $V_{CCIO}$ – 0.45 | — | V |
| $V_{OL}$ | Low-level output voltage | IOL = 2 mA *(1)* | — | 0.45 | V |

**Table 5–9.** 1.5-V I/O Specifications

| Symbol | Parameter | Conditions | Minimum | Maximum | Unit |
|---|---|---|---|---|---|
| $V_{CCIO}$ | I/O supply voltage | — | 1.425 | 1.575 | V |
| $V_{IH}$ | High-level input voltage | — | 0.65 × $V_{CCIO}$ | $V_{CCIO}$ + 0.3 *(2)* | V |
| $V_{IL}$ | Low-level input voltage | — | –0.3 | 0.35 × $V_{CCIO}$ | V |
| $V_{OH}$ | High-level output voltage | IOH = –2 mA *(1)* | 0.75 × $V_{CCIO}$ | — | V |
| $V_{OL}$ | Low-level output voltage | IOL = 2 mA *(1)* | — | 0.25 × $V_{CCIO}$ | V |

**Notes to Table 5–5 through Table 5–9:**

(1) This specification is supported across all the programmable drive strength settings available for this I/O standard, as shown in the *MAX II Architecture* chapter (*I/O Structure section*) in the *MAX II Device Handbook*.

(2) This maximum $V_{IH}$ reflects the JEDEC specification. The MAX II input buffer can tolerate a $V_{IH}$ maximum of 4.0, as specified by the $V_I$ parameter in Table 5–2.

**Table 5–10.** 3.3-V PCI Specifications *(Note 1)*

| Symbol | Parameter | Conditions | Minimum | Typical | Maximum | Unit |
|---|---|---|---|---|---|---|
| $V_{CCIO}$ | I/O supply voltage | — | 3.0 | 3.3 | 3.6 | V |
| $V_{IH}$ | High-level input voltage | — | 0.5 × $V_{CCIO}$ | — | $V_{CCIO}$ + 0.5 | V |
| $V_{IL}$ | Low-level input voltage | — | –0.5 | — | 0.3 × $V_{CCIO}$ | V |
| $V_{OH}$ | High-level output voltage | IOH = –500 µA | 0.9 × $V_{CCIO}$ | — | — | V |
| $V_{OL}$ | Low-level output voltage | IOL = 1.5 mA | — | — | 0.1 × $V_{CCIO}$ | V |

**Note to Table 5–10:**

(1) 3.3-V PCI I/O standard is only supported in Bank 3 of the EPM1270 and EPM2210 devices.

## Bus Hold Specifications

Table 5–11 shows the MAX II device family bus hold specifications.

**Table 5–11.** Bus Hold Specifications

| Parameter | Conditions | $V_{CCIO}$ Level | | | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 1.5 V | | 1.8 V | | 2.5 V | | 3.3 V | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | |
| Low sustaining current | $V_{IN}$ > $V_{IL}$ (maximum) | 20 | — | 30 | — | 50 | — | 70 | — | µA |
| High sustaining current | $V_{IN}$ < $V_{IH}$ (minimum) | –20 | — | –30 | — | –50 | — | –70 | — | µA |
| Low overdrive current | 0 V < $V_{IN}$ < $V_{CCIO}$ | — | 160 | — | 200 | — | 300 | — | 500 | µA |
| High overdrive current | 0 V < $V_{IN}$ < $V_{CCIO}$ | — | –160 | — | –200 | — | –300 | — | –500 | µA |

## Power-Up Timing

Table 5–12 shows the power-up timing characteristics for MAX II devices.

**Table 5–12.** MAX II Power-Up Timing

| Symbol | Parameter | Device | Min | Typ | Max | Unit |
|---|---|---|---|---|---|---|
| $t_{CONFIG}$ *(1)* | The amount of time from when minimum $V_{CCINT}$ is reached until the device enters user mode *(2)* | EPM240 | — | — | 200 | µs |
| | | EPM570 | — | — | 300 | µs |
| | | EPM1270 | — | — | 300 | µs |
| | | EPM2210 | — | — | 450 | µs |

**Notes to Table 5–12:**

(1) Table 5–12 values apply to commercial and industrial range devices. For extended temperature range devices, the $t_{CONFIG}$ maximum values are as follows:

| Device | Maximum |
|---|---|
| EPM240 | 300 µs |
| EPM570 | 400 µs |
| EPM1270 | 400 µs |
| EPM2210 | 500 µs |

(2) For more information about POR trigger voltage, refer to the *Hot Socketing and Power-On Reset in MAX II Devices* chapter in the *MAX II Device Handbook*.

# Power Consumption

Designers can use the Altera® PowerPlay Early Power Estimator and PowerPlay Power Analyzer to estimate the device power.

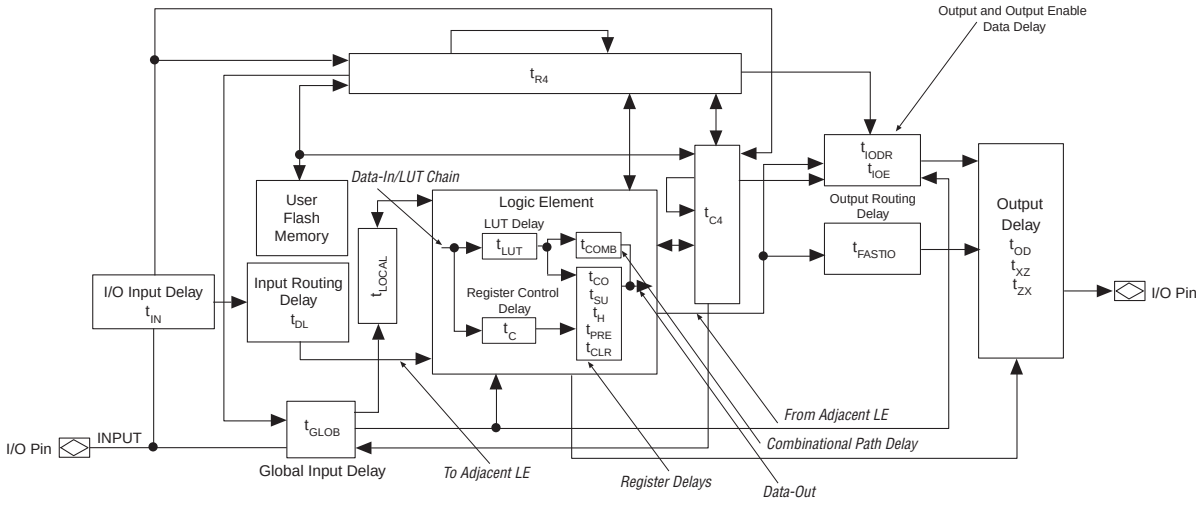
For more information about these power analysis tools, refer to the *Understanding and Evaluating Power in MAX II Devices* chapter in the *MAX II Device Handbook* and the *PowerPlay Power Analysis* chapter in volume 3 of the *Quartus II Handbook*.

# Timing Model and Specifications

MAX II devices timing can be analyzed with the Altera Quartus® II software, a variety of popular industry-standard EDA simulators and timing analyzers, or with the timing model shown in Figure 5–2.

MAX II devices have predictable internal delays that enable the designer to determine the worst-case timing of any design. The software provides timing simulation, point-to-point delay prediction, and detailed timing analysis for device-wide performance evaluation.

**Figure 5–2.** MAX II Device Timing Model




The timing characteristics of any signal path can be derived from the timing model and parameters of a particular device. External timing parameters, which represent pin-to-pin timing delays, can be calculated as the sum of internal parameters.

Refer to the *Understanding Timing in MAX II Devices* chapter in the *MAX II Device Handbook* for more information.

This section describes and specifies the performance, internal, external, and UFM timing specifications. All specifications are representative of the worst-case supply voltage and junction temperature conditions.

## Preliminary and Final Timing

Timing models can have either preliminary or final status. The Quartus II software issues an informational message during the design compilation if the timing models are preliminary. Table 5–13 shows the status of the MAX II device timing models.

Preliminary status means the timing model is subject to change. Initially, timing numbers are created using simulation results, process data, and other known parameters. These tests are used to make the preliminary numbers as close to the actual timing parameters as possible.

Final timing numbers are based on actual device operation and testing. These numbers reflect the actual performance of the device under the worst-case voltage and junction temperature conditions.

**Table 5–13.** MAX II Device Timing Model Status (Part 1 of 2)

| Device | Preliminary | Final |
|---|---|---|
| EPM240 | — | ✓ |
| EPM240Z *(1)* | — | ✓ |
| EPM570 | — | ✓ |
| EPM570Z *(1)* | — | ✓ |

**Table 5–13.** MAX II Device Timing Model Status (Part 2 of 2)

| Device | Preliminary | Final |
|---|---|---|
| EPM1270 | — | ✓ |
| EPM2210 | — | ✓ |

**Note to Table 5–13:**

(1) The MAX IIZ device timing models are only available in the Quartus II software version 8.0 and later.

## Performance

Table 5–14 shows the MAX II device performance for some common designs. All performance values were obtained with the Quartus II software compilation of megafunctions. Performance values for –3, –4, and –5 speed grades are based on an EPM1270 device target, while –6, –7, and –8 speed grades are based on an EPM570Z device target.

**Table 5–14.** MAX II Device Performance

| Resource Used | Design Size and Function | Resources Used | | | Performance | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | MAX II / MAX IIG | | | MAX IIZ | | | |
| | | Mode | LEs | UFM Blocks | –3 Speed Grade | –4 Speed Grade | –5 Speed Grade | –6 Speed Grade | –7 Speed Grade | –8 Speed Grade | |
| LE | 16-bit counter *(1)* | — | 16 | 0 | 304.0 | 247.5 | 201.1 | 184.1 | 123.5 | 118.3 | MHz |
| | 64-bit counter *(1)* | — | 64 | 0 | 201.5 | 154.8 | 125.8 | 83.2 | 83.2 | 80.5 | MHz |
| | 16-to-1 multiplexer | — | 11 | 0 | 6.0 | 8.0 | 9.3 | 17.4 | 17.3 | 20.4 | ns |
| | 32-to-1 multiplexer | — | 24 | 0 | 7.1 | 9.0 | 11.4 | 12.5 | 22.8 | 25.3 | ns |
| | 16-bit XOR function | — | 5 | 0 | 5.1 | 6.6 | 8.2 | 9.0 | 15.0 | 16.1 | ns |
| | 16-bit decoder with single address line | — | 5 | 0 | 5.2 | 6.6 | 8.2 | 9.2 | 15.0 | 16.1 | ns |
| UFM | 512 × 16 | None | 3 | 1 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | MHz |
| | 512 × 16 | SPI *(2)* | 37 | 1 | 8.0 | 8.0 | 8.0 | 9.7 | 9.7 | 9.7 | MHz |
| | 512 × 8 | Parallel *(3)* | 73 | 1 | *(4)* | *(4)* | *(4)* | *(4)* | *(4)* | *(4)* | MHz |
| | 512 × 16 | $I^2C$ *(3)* | 142 | 1 | 100 *(5)* | 100 *(5)* | 100 *(5)* | 100 *(5)* | 100 *(5)* | 100 *(5)* | kHz |

**Notes to Table 5–14:**

(1) This design is a binary loadable up counter.
(2) This design is configured for read-only operation in Extended mode. Read and write ability increases the number of LEs used.
(3) This design is configured for read-only operation. Read and write ability increases the number of LEs used.
(4) This design is asynchronous.
(5) The $I^2C$ megafunction is verified in hardware up to 100-kHz serial clock line (SCL) rate.

## Internal Timing Parameters

Internal timing parameters are specified on a speed grade basis independent of device density. Table 5–15 through Table 5–22 describe the MAX II device internal timing microparameters for logic elements (LEs), input/output elements (IOEs), UFM blocks, and MultiTrack interconnects. The timing values for –3, –4, and –5 speed grades shown in Table 5–15 through Table 5–22 are based on an EPM1270 device target, while –6, –7, and –8 speed grade values are based on an EPM570Z device target.

For more explanations and descriptions about each internal timing microparameters symbol, refer to the *Understanding Timing in MAX II Devices* chapter in the *MAX II Device Handbook*.

**Table 5–15.** LE Internal Timing Microparameters

| Symbol | Parameter | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{LUT}$ | LE combinational LUT delay | — | 571 | — | 742 | — | 914 | — | 1,215 | — | 2,247 | — | 2,247 | ps |
| $t_{COMB}$ | Combinational path delay | — | 147 | — | 192 | — | 236 | — | 243 | — | 305 | — | 309 | ps |
| $t_{CLR}$ | LE register clear delay | 238 | — | 309 | — | 381 | — | 401 | — | 541 | — | 545 | — | ps |
| $t_{PRE}$ | LE register preset delay | 238 | — | 309 | — | 381 | — | 401 | — | 541 | — | 545 | — | ps |
| $t_{SU}$ | LE register setup time before clock | 208 | — | 271 | — | 333 | — | 260 | — | 319 | — | 321 | — | ps |
| $t_{H}$ | LE register hold time after clock | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | ps |
| $t_{CO}$ | LE register clock-to-output delay | — | 235 | — | 305 | — | 376 | — | 380 | — | 489 | — | 494 | ps |
| $t_{CLKHL}$ | Minimum clock high or low time | 166 | — | 216 | — | 266 | — | 253 | — | 335 | — | 339 | — | ps |
| $t_{C}$ | Register control delay | — | 857 | — | 1,114 | — | 1,372 | — | 1,356 | — | 1,722 | — | 1,741 | ps |

**Table 5–16.** IOE Internal Timing Microparameters

| Symbol | Parameter | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{FASTIO}$ | Data output delay from adjacent LE to I/O block | — | 159 | — | 207 | — | 254 | — | 170 | — | 348 | — | 428 | ps |
| $t_{IN}$ | I/O input pad and buffer delay | — | 708 | — | 920 | — | 1,132 | — | 907 | — | 970 | — | 986 | ps |
| $t_{GLOB}$ *(1)* | I/O input pad and buffer delay used as global signal pin | — | 1,519 | — | 1,974 | — | 2,430 | — | 2,261 | — | 2,670 | — | 3,322 | ps |
| $t_{IOE}$ | Internally generated output enable delay | — | 354 | — | 374 | — | 460 | — | 530 | — | 966 | — | 1,410 | ps |
| $t_{DL}$ | Input routing delay | — | 224 | — | 291 | — | 358 | — | 318 | — | 410 | — | 509 | ps |
| $t_{OD}$ *(2)* | Output delay buffer and pad delay | — | 1,064 | — | 1,383 | — | 1,702 | — | 1,319 | — | 1,526 | — | 1,543 | ps |
| $t_{XZ}$ *(3)* | Output buffer disable delay | — | 756 | — | 982 | — | 1,209 | — | 1,045 | — | 1,264 | — | 1,276 | ps |
| $t_{ZX}$ *(4)* | Output buffer enable delay | — | 1,003 | — | 1,303 | — | 1,604 | — | 1,160 | — | 1,325 | — | 1,353 | ps |

**Notes to Table 5–16:**

(1) Delay numbers for $t_{GLOB}$ differ for each device density and speed grade. The delay numbers for $t_{GLOB}$, shown in Table 5–16, are based on an EPM240 device target.

(2) Refer to Table 5–32 and 5–24 for delay adders associated with different I/O standards, drive strengths, and slew rates.

(3) Refer to Table 5–19 and 5–14 for $t_{XZ}$ delay adders associated with different I/O standards, drive strengths, and slew rates.

(4) Refer to Table 5–17 and 5–13 for $t_{ZX}$ delay adders associated with different I/O standards, drive strengths, and slew rates.

Table 5–17 through Table 5–20 show the adder delays for $t_{ZX}$ and $t_{XZ}$ microparameters when using an I/O standard other than 3.3-V LVTTL with 16 mA drive strength.

**Table 5–17.** $t_{ZX}$ IOE Microparameter Adders for Fast Slew Rate (Part 1 of 2)

| Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | 16 mA | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | 8 mA | — | 28 | — | 37 | — | 45 | — | 72 | — | 71 | — | 74 | ps |
| 3.3-V LVCMOS | 8 mA | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | 4 mA | — | 28 | — | 37 | — | 45 | — | 72 | — | 71 | — | 74 | ps |
| 2.5-V LVTTL / LVCMOS | 14 mA | — | 14 | — | 19 | — | 23 | — | 75 | — | 87 | — | 90 | ps |
| | 7 mA | — | 314 | — | 409 | — | 503 | — | 162 | — | 174 | — | 177 | ps |
| 1.8-V LVTTL / LVCMOS | 6 mA | — | 450 | — | 585 | — | 720 | — | 279 | — | 289 | — | 291 | ps |
| | 3 mA | — | 1,443 | — | 1,876 | — | 2,309 | — | 499 | — | 508 | — | 512 | ps |

**Table 5–17.** $t_{ZX}$ IOE Microparameter Adders for Fast Slew Rate (Part 2 of 2)

| Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 1.5-V LVCMOS | 4 mA | — | 1,118 | — | 1,454 | — | 1,789 | — | 580 | — | 588 | — | 588 | ps |
| | 2 mA | — | 2,410 | — | 3,133 | — | 3,856 | — | 915 | — | 923 | — | 923 | ps |
| 3.3-V PCI | 20 mA | — | 19 | — | 25 | — | 31 | — | 72 | — | 71 | — | 74 | ps |

**Table 5–18.** $t_{ZX}$ IOE Microparameter Adders for Slow Slew Rate

| Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | 16 mA | — | 6,350 | — | 6,050 | — | 5,749 | — | 5,951 | — | 5,952 | — | 6,063 | ps |
| | 8 mA | — | 9,383 | — | 9,083 | — | 8,782 | — | 6,534 | — | 6,533 | — | 6,662 | ps |
| 3.3-V LVCMOS | 8 mA | — | 6,350 | — | 6,050 | — | 5,749 | — | 5,951 | — | 5,952 | — | 6,063 | ps |
| | 4 mA | — | 9,383 | — | 9,083 | — | 8,782 | — | 6,534 | — | 6,533 | — | 6,662 | ps |
| 2.5-V LVTTL / LVCMOS | 14 mA | — | 10,412 | — | 10,112 | — | 9,811 | — | 9,110 | — | 9,105 | — | 9,237 | ps |
| | 7 mA | — | 13,613 | — | 13,313 | — | 13,012 | — | 9,830 | — | 9,835 | — | 9,977 | ps |
| 3.3-V PCI | 20 mA | — | –75 | — | –97 | — | –120 | — | 6,534 | — | 6,533 | — | 6,662 | ps |

**Table 5–19.** $t_{XZ}$ IOE Microparameter Adders for Fast Slew Rate

| Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | 16 mA | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | 8 mA | — | –56 | — | –72 | — | –89 | — | –69 | — | –69 | — | –69 | ps |
| 3.3-V LVCMOS | 8 mA | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | 4 mA | — | –56 | — | –72 | — | –89 | — | –69 | — | –69 | — | –69 | ps |
| 2.5-V LVTTL / LVCMOS | 14 mA | — | –3 | — | –4 | — | –5 | — | –7 | — | –11 | — | –11 | ps |
| | 7 mA | — | –47 | — | –61 | — | –75 | — | –66 | — | –70 | — | –70 | ps |
| 1.8-V LVTTL / LVCMOS | 6 mA | — | 119 | — | 155 | — | 191 | — | 45 | — | 34 | — | 37 | ps |
| | 3 mA | — | 207 | — | 269 | — | 331 | — | 34 | — | 22 | — | 25 | ps |
| 1.5-V LVCMOS | 4 mA | — | 606 | — | 788 | — | 970 | — | 166 | — | 154 | — | 155 | ps |
| | 2 mA | — | 673 | — | 875 | — | 1,077 | — | 190 | — | 177 | — | 179 | ps |
| 3.3-V PCI | 20 mA | — | 71 | — | 93 | — | 114 | — | –69 | — | –69 | — | –69 | ps |

**Table 5–20.** $t_{XZ}$ IOE Microparameter Adders for Slow Slew Rate

| Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | 16 mA | — | 206 | — | –20 | — | –247 | — | 1,433 | — | 1,446 | — | 1,454 | ps |
| | 8 mA | — | 891 | — | 665 | — | 438 | — | 1,332 | — | 1,345 | — | 1,348 | ps |
| 3.3-V LVCMOS | 8 mA | — | 206 | — | –20 | — | –247 | — | 1,433 | — | 1,446 | — | 1,454 | ps |
| | 4 mA | — | 891 | — | 665 | — | 438 | — | 1,332 | — | 1,345 | — | 1,348 | ps |
| 2.5-V LVTTL / LVCMOS | 14 mA | — | 222 | — | –4 | — | –231 | — | 213 | — | 208 | — | 213 | ps |
| | 7 mA | — | 943 | — | 717 | — | 490 | — | 166 | — | 161 | — | 166 | ps |
| 3.3-V PCI | 20 mA | — | 161 | — | 210 | — | 258 | — | 1,332 | — | 1,345 | — | 1,348 | ps |

The default slew rate setting for MAX II devices in the Quartus II design software is "fast".

**Table 5–21.** UFM Block Internal Timing Microparameters (Part 1 of 3)

| Symbol | Parameter | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{ACLK}$ | Address register clock period | 100 | — | 100 | — | 100 | — | 100 | — | 100 | — | 100 | — | ns |
| $t_{ASU}$ | Address register shift signal setup to address register clock | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{AH}$ | Address register shift signal hold to address register clock | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{ADS}$ | Address register data in setup to address register clock | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{ADH}$ | Address register data in hold from address register clock | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{DCLK}$ | Data register clock period | 100 | — | 100 | — | 100 | — | 100 | — | 100 | — | 100 | — | ns |
| $t_{DSS}$ | Data register shift signal setup to data register clock | 60 | — | 60 | — | 60 | — | 60 | — | 60 | — | 60 | — | ns |
| $t_{DSH}$ | Data register shift signal hold from data register clock | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |

**Table 5–21.** UFM Block Internal Timing Microparameters (Part 2 of 3)

| Symbol | Parameter | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{DDS}$ | Data register data in setup to data register clock | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{DDH}$ | Data register data in hold from data register clock | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{DP}$ | Program signal to data clock hold time | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | ns |
| $t_{PB}$ | Maximum delay between program rising edge to UFM busy signal rising edge | — | 960 | — | 960 | — | 960 | — | 960 | — | 960 | — | 960 | ns |
| $t_{BP}$ | Minimum delay allowed from UFM busy signal going low to program signal going low | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{PPMX}$ | Maximum length of busy pulse during a program | — | 100 | — | 100 | — | 100 | — | 100 | — | 100 | — | 100 | µs |
| $t_{AE}$ | Minimum erase signal to address clock hold time | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | ns |
| $t_{EB}$ | Maximum delay between the erase rising edge to the UFM busy signal rising edge | — | 960 | — | 960 | — | 960 | — | 960 | — | 960 | — | 960 | ns |
| $t_{BE}$ | Minimum delay allowed from the UFM busy signal going low to erase signal going low | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | 20 | — | ns |
| $t_{EPMX}$ | Maximum length of busy pulse during an erase | — | 500 | — | 500 | — | 500 | — | 500 | — | 500 | — | 500 | ms |
| $t_{DCO}$ | Delay from data register clock to data register output | — | 5 | — | 5 | — | 5 | — | 5 | — | 5 | — | 5 | ns |

**Table 5–21.** UFM Block Internal Timing Microparameters (Part 3 of 3)

| Symbol | Parameter | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{OE}$ | Delay from data register clock to data register output | 180 | — | 180 | — | 180 | — | 180 | — | 180 | — | 180 | — | ns |
| $t_{RA}$ | Maximum read access time | — | 65 | — | 65 | — | 65 | — | 65 | — | 65 | — | 65 | ns |
| $t_{OSCS}$ | Maximum delay between the OSC_ENA rising edge to the erase/program signal rising edge | 250 | — | 250 | — | 250 | — | 250 | — | 250 | — | 250 | — | ns |
| $t_{OSCH}$ | Minimum delay allowed from the erase/program signal going low to OSC_ENA signal going low | 250 | — | 250 | — | 250 | — | 250 | — | 250 | — | 250 | — | ns |

Figure 5–3 through Figure 5–5 show the read, program, and erase waveforms for UFM block timing parameters shown in Table 5–21.

**Figure 5–3.** UFM Read Waveforms

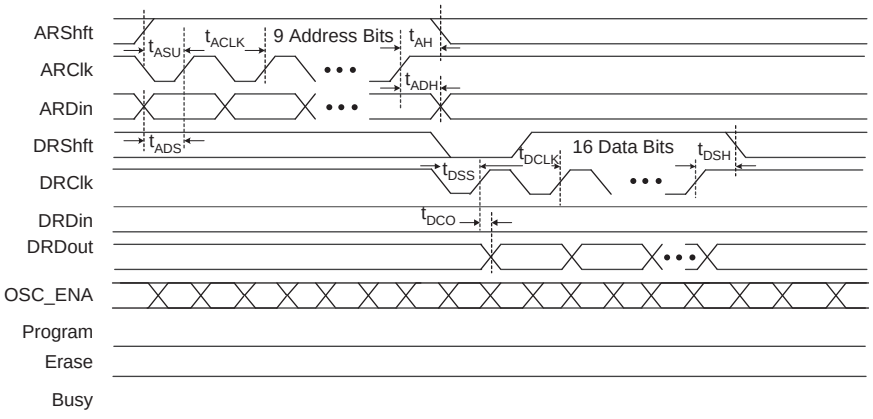


**Figure 5–4.** UFM Program Waveforms

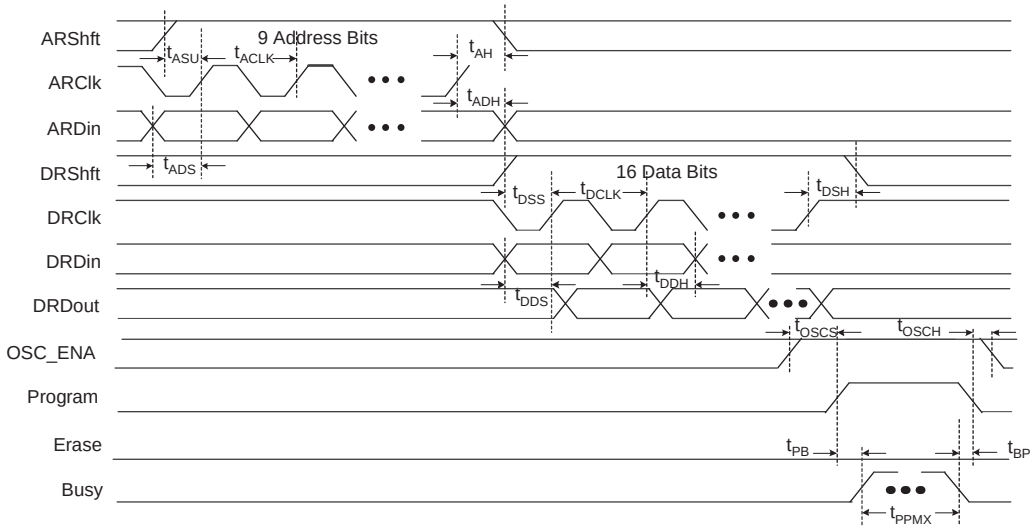



**Figure 5–5.** UFM Erase Waveform

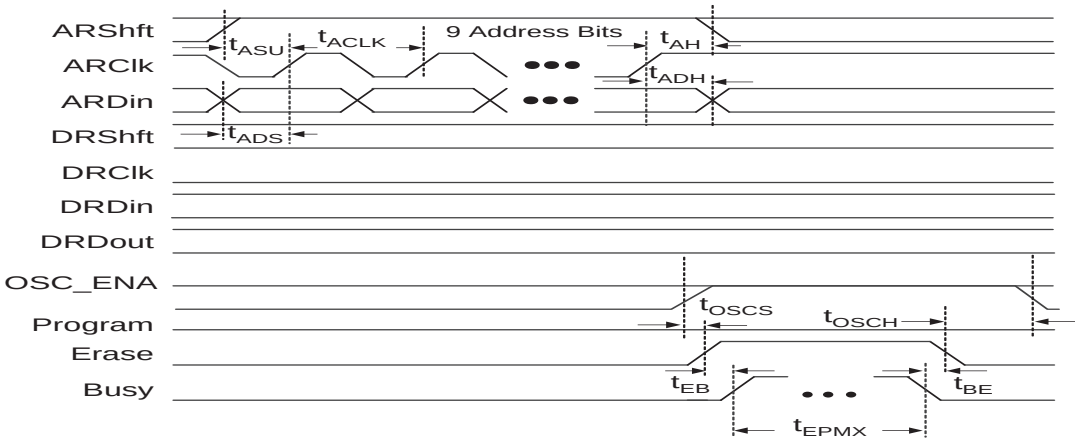



**Table 5–22.** Routing Delay Internal Timing Microparameters

| Routing | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{C4}$ | — | 429 | — | 556 | — | 687 | — | *(1)* | — | *(1)* | — | *(1)* | ps |
| $t_{R4}$ | — | 326 | — | 423 | — | 521 | — | *(1)* | — | *(1)* | — | *(1)* | ps |
| $t_{LOCAL}$ | — | 330 | — | 429 | — | 529 | — | *(1)* | — | *(1)* | — | *(1)* | ps |

**Note to Table 5–22:**

(1) The numbers will only be available in a later revision.

## External Timing Parameters

External timing parameters are specified by device density and speed grade. All external I/O timing parameters shown are for the 3.3-V LVTTL I/O standard with the maximum drive strength and fast slew rate. For external I/O timing using standards other than LVTTL or for different drive strengths, use the I/O standard input and output delay adders in Table 5–27 through Table 5–31.

For more information about each external timing parameters symbol, refer to the *Understanding Timing in MAX II Devices* chapter in the *MAX II Device Handbook*.

Table 5–23 shows the external I/O timing parameters for EPM240 devices.

**Table 5–23.** EPM240 Global Clock External I/O Timing Parameters (Part 1 of 2)

| | | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| Symbol | Parameter | Condition | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Unit |
| $t_{PD1}$ | Worst case pin-to-pin delay through 1 look-up table (LUT) | 10 pF | — | 4.7 | — | 6.1 | — | 7.5 | — | 7.9 | — | 12.0 | — | 14.0 | ns |
| $t_{PD2}$ | Best case pin-to-pin delay through 1 LUT | 10 pF | — | 3.7 | — | 4.8 | — | 5.9 | — | 5.8 | — | 7.8 | — | 8.5 | ns |
| $t_{SU}$ | Global clock setup time | — | 1.7 | — | 2.2 | — | 2.7 | — | 2.4 | — | 4.1 | — | 4.6 | — | ns |
| $t_{H}$ | Global clock hold time | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | ns |
| $t_{CO}$ | Global clock to output delay | 10 pF | 2.0 | 4.3 | 2.0 | 5.6 | 2.0 | 6.9 | 2.0 | 6.6 | 2.0 | 8.1 | 2.0 | 8.6 | ns |
| $t_{CH}$ | Global clock high time | — | 166 | — | 216 | — | 266 | — | 253 | — | 335 | — | 339 | — | ps |
| $t_{CL}$ | Global clock low time | — | 166 | — | 216 | — | 266 | — | 253 | — | 335 | — | 339 | — | ps |
| $t_{CNT}$ | Minimum global clock period for 16-bit counter | — | 3.3 | — | 4.0 | — | 5.0 | — | 5.4 | — | 8.1 | — | 8.4 | — | ns |

**Table 5–23.** EPM240 Global Clock External I/O Timing Parameters (Part 2 of 2)

| Symbol | Parameter | Condition | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $f_{CNT}$ | Maximum global clock frequency for 16-bit counter | — | — | 304.0 *(1)* | — | 247.5 | — | 201.1 | — | 184.1 | — | 123.5 | — | 118.3 | MHz |

**Note to Table 5–23:**

(1) The maximum frequency is limited by the I/O standard on the clock input pin. The 16-bit counter critical delay performs faster than this global clock input pin maximum frequency.

Table 5–24 shows the external I/O timing parameters for EPM570 devices.

**Table 5–24.** EPM570 Global Clock External I/O Timing Parameters (Part 1 of 2)

| Symbol | Parameter | Condition | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $t_{PD1}$ | Worst case pin-to-pin delay through 1 look-up table (LUT) | 10 pF | — | 5.4 | — | 7.0 | — | 8.7 | — | 9.5 | — | 15.1 | — | 17.7 | ns |
| $t_{PD2}$ | Best case pin-to-pin delay through 1 LUT | 10 pF | — | 3.7 | — | 4.8 | — | 5.9 | — | 5.7 | — | 7.7 | — | 8.5 | ns |
| $t_{SU}$ | Global clock setup time | — | 1.2 | — | 1.5 | — | 1.9 | — | 2.2 | — | 3.9 | — | 4.4 | — | ns |
| $t_{H}$ | Global clock hold time | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | ns |
| $t_{CO}$ | Global clock to output delay | 10 pF | 2.0 | 4.5 | 2.0 | 5.8 | 2.0 | 7.1 | 2.0 | 6.7 | 2.0 | 8.2 | 2.0 | 8.7 | ns |
| $t_{CH}$ | Global clock high time | — | 166 | — | 216 | — | 266 | — | 253 | — | 335 | — | 339 | — | ps |
| $t_{CL}$ | Global clock low time | — | 166 | — | 216 | — | 266 | — | 253 | — | 335 | — | 339 | — | ps |
| $t_{CNT}$ | Minimum global clock period for 16-bit counter | — | 3.3 | — | 4.0 | — | 5.0 | — | 5.4 | — | 8.1 | — | 8.4 | — | ns |

**Table 5–24.** EPM570 Global Clock External I/O Timing Parameters (Part 2 of 2)

| Symbol | Parameter | Condition | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| $f_{CNT}$ | Maximum global clock frequency for 16-bit counter | — | — | 304.0 *(1)* | — | 247.5 | — | 201.1 | — | 184.1 | — | 123.5 | — | 118.3 | MHz |

**Note to Table 5–24:**

(1) The maximum frequency is limited by the I/O standard on the clock input pin. The 16-bit counter critical delay performs faster than this global clock input pin maximum frequency.

Table 5–25 shows the external I/O timing parameters for EPM1270 devices.

**Table 5–25.** EPM1270 Global Clock External I/O Timing Parameters

| Symbol | Parameter | Condition | MAX II / MAX IIG | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | |
| | | | Min | Max | Min | Max | Min | Max | |
| $t_{PD1}$ | Worst case pin-to-pin delay through 1 look-up table (LUT) | 10 pF | — | 6.2 | — | 8.1 | — | 10.0 | ns |
| $t_{PD2}$ | Best case pin-to-pin delay through 1 LUT | 10 pF | — | 3.7 | — | 4.8 | — | 5.9 | ns |
| $t_{SU}$ | Global clock setup time | — | 1.2 | — | 1.5 | — | 1.9 | — | ns |
| $t_{H}$ | Global clock hold time | — | 0 | — | 0 | — | 0 | — | ns |
| $t_{CO}$ | Global clock to output delay | 10 pF | 2.0 | 4.6 | 2.0 | 5.9 | 2.0 | 7.3 | ns |
| $t_{CH}$ | Global clock high time | — | 166 | — | 216 | — | 266 | — | ps |
| $t_{CL}$ | Global clock low time | — | 166 | — | 216 | — | 266 | — | ps |
| $t_{CNT}$ | Minimum global clock period for 16-bit counter | — | 3.3 | — | 4.0 | — | 5.0 | — | ns |
| $f_{CNT}$ | Maximum global clock frequency for 16-bit counter | — | — | 304.0 *(1)* | — | 247.5 | — | 201.1 | MHz |

**Note to Table 5–25:**

(1) The maximum frequency is limited by the I/O standard on the clock input pin. The 16-bit counter critical delay performs faster than this global clock input pin maximum frequency.

Table 5–26 shows the external I/O timing parameters for EPM2210 devices.

**Table 5–26.** EPM2210 Global Clock External I/O Timing Parameters

| Symbol | Parameter | Condition | MAX II / MAX IIG | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|
| | | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | |
| | | | Min | Max | Min | Max | Min | Max | |
| $t_{PD1}$ | Worst case pin-to-pin delay through 1 look-up table (LUT) | 10 pF | — | 7.0 | — | 9.1 | — | 11.2 | ns |
| $t_{PD2}$ | Best case pin-to-pin delay through 1 LUT | 10 pF | — | 3.7 | — | 4.8 | — | 5.9 | ns |
| $t_{SU}$ | Global clock setup time | — | 1.2 | — | 1.5 | — | 1.9 | — | ns |
| $t_{H}$ | Global clock hold time | — | 0 | — | 0 | — | 0 | — | ns |
| $t_{CO}$ | Global clock to output delay | 10 pF | 2.0 | 4.6 | 2.0 | 6.0 | 2.0 | 7.4 | ns |
| $t_{CH}$ | Global clock high time | — | 166 | — | 216 | — | 266 | — | ps |
| $t_{CL}$ | Global clock low time | — | 166 | — | 216 | — | 266 | — | ps |
| $t_{CNT}$ | Minimum global clock period for 16-bit counter | — | 3.3 | — | 4.0 | — | 5.0 | — | ns |
| $f_{CNT}$ | Maximum global clock frequency for 16-bit counter | — | — | 304.0 *(1)* | — | 247.5 | — | 201.1 | MHz |

**Note to Table 5–26:**

(1) The maximum frequency is limited by the I/O standard on the clock input pin. The 16-bit counter critical delay performs faster than this global clock input pin maximum frequency.

## External Timing I/O Delay Adders

The I/O delay timing parameters for I/O standard input and output adders, and input delays are specified by speed grade independent of device density.

Table 5–27 through Table 5–31 show the adder delays associated with I/O pins for all packages. The delay numbers for –3, –4, and –5 speed grades shown in Table 5–27 through Table 5–33 are based on an EPM1270 device target, while –6, –7, and –8 speed grade values are based on an EPM570Z device target. If an I/O standard other than 3.3-V LVTTL is selected, add the input delay adder to the external $t_{SU}$ timing parameters shown in Table 5–23 through Table 5–26. If an I/O standard other than 3.3-V LVTTL with 16 mA drive strength and fast slew rate is selected, add the output delay adder to the external $t_{CO}$ and $t_{PD}$ shown in Table 5–23 through Table 5–26.

**Table 5–27.** External Timing Input Delay Adders (Part 1 of 2)

| I/O Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | Without Schmitt Trigger | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | With Schmitt Trigger | — | 334 | — | 434 | — | 535 | — | 387 | — | 434 | — | 442 | ps |

**Table 5–27.** External Timing Input Delay Adders (Part 2 of 2)

| I/O Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVCMOS | Without Schmitt Trigger | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | With Schmitt Trigger | — | 334 | — | 434 | — | 535 | — | 387 | — | 434 | — | 442 | ps |
| 2.5-V LVTTL / LVCMOS | Without Schmitt Trigger | — | 23 | — | 30 | — | 37 | — | 42 | — | 43 | — | 43 | ps |
| | With Schmitt Trigger | — | 339 | — | 441 | — | 543 | — | 429 | — | 476 | — | 483 | ps |
| 1.8-V LVTTL / LVCMOS | Without Schmitt Trigger | — | 291 | — | 378 | — | 466 | — | 378 | — | 373 | — | 373 | ps |
| 1.5-V LVCMOS | Without Schmitt Trigger | — | 681 | — | 885 | — | 1,090 | — | 681 | — | 622 | — | 658 | ps |
| 3.3-V PCI | Without Schmitt Trigger | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |

**Table 5–28.** External Timing Input Delay $t_{GLOB}$ Adders for GCLK Pins

| I/O Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | Without Schmitt Trigger | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | With Schmitt Trigger | — | 308 | — | 400 | — | 493 | — | 387 | — | 434 | — | 442 | ps |
| 3.3-V LVCMOS | Without Schmitt Trigger | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | With Schmitt Trigger | — | 308 | — | 400 | — | 493 | — | 387 | — | 434 | — | 442 | ps |
| 2.5-V LVTTL / LVCMOS | Without Schmitt Trigger | — | 21 | — | 27 | — | 33 | — | 42 | — | 43 | — | 43 | ps |
| | With Schmitt Trigger | — | 423 | — | 550 | — | 677 | — | 429 | — | 476 | — | 483 | ps |
| 1.8-V LVTTL / LVCMOS | Without Schmitt Trigger | — | 353 | — | 459 | — | 565 | — | 378 | — | 373 | — | 373 | ps |
| 1.5-V LVCMOS | Without Schmitt Trigger | — | 855 | — | 1,111 | — | 1,368 | — | 681 | — | 622 | — | 658 | ps |
| 3.3-V PCI | Without Schmitt Trigger | — | 6 | — | 7 | — | 9 | — | 0 | — | 0 | — | 0 | ps |

**Table 5–29.** External Timing Output Delay and $t_{OD}$ Adders for Fast Slew Rate

| I/O Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | 16 mA | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | 8 mA | — | 65 | — | 84 | — | 104 | — | –6 | — | –2 | — | –3 | ps |
| 3.3-V LVCMOS | 8 mA | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | — | 0 | ps |
| | 4 mA | — | 65 | — | 84 | — | 104 | — | –6 | — | –2 | — | –3 | ps |
| 2.5-V LVTTL / LVCMOS | 14 mA | — | 122 | — | 158 | — | 195 | — | –63 | — | –71 | — | –88 | ps |
| | 7 mA | — | 193 | — | 251 | — | 309 | — | 10 | — | –1 | — | 1 | ps |
| 1.8-V LVTTL / LVCMOS | 6 mA | — | 568 | — | 738 | — | 909 | — | 128 | — | 118 | — | 118 | ps |
| | 3 mA | — | 654 | — | 850 | — | 1,046 | — | 352 | — | 327 | — | 332 | ps |
| 1.5-V LVCMOS | 4 mA | — | 1,059 | — | 1,376 | — | 1,694 | — | 421 | — | 400 | — | 400 | ps |
| | 2 mA | — | 1,167 | — | 1,517 | — | 1,867 | — | 757 | — | 743 | — | 743 | ps |
| 3.3-V PCI | 20 mA | — | 3 | — | 4 | — | 5 | — | –6 | — | –2 | — | –3 | ps |

**Table 5–30.** External Timing Output Delay and $t_{OD}$ Adders for Slow Slew Rate

| I/O Standard | | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| 3.3-V LVTTL | 16 mA | — | 7,064 | — | 6,745 | — | 6,426 | — | 5,966 | — | 5,992 | — | 6,118 | ps |
| | 8 mA | — | 7,946 | — | 7,627 | — | 7,308 | — | 6,541 | — | 6,570 | — | 6,720 | ps |
| 3.3-V LVCMOS | 8 mA | — | 7,064 | — | 6,745 | — | 6,426 | — | 5,966 | — | 5,992 | — | 6,118 | ps |
| | 4 mA | — | 7,946 | — | 7,627 | — | 7,308 | — | 6,541 | — | 6,570 | — | 6,720 | ps |
| 2.5-V LVTTL / LVCMOS | 14 mA | — | 10,434 | — | 10,115 | — | 9,796 | — | 9,141 | — | 9,154 | — | 9,297 | ps |
| | 7 mA | — | 11,548 | — | 11,229 | — | 10,910 | — | 9,861 | — | 9,874 | — | 10,037 | ps |
| 1.8-V LVTTL / LVCMOS | 6 mA | — | 22,927 | — | 22,608 | — | 22,289 | — | 21,811 | — | 21,854 | — | 21,857 | ps |
| | 3 mA | — | 24,731 | — | 24,412 | — | 24,093 | — | 23,081 | — | 23,034 | — | 23,107 | ps |
| 1.5-V LVCMOS | 4 mA | — | 38,723 | — | 38,404 | — | 38,085 | — | 39,121 | — | 39,124 | — | 39,124 | ps |
| | 2 mA | — | 41,330 | — | 41,011 | — | 40,692 | — | 40,631 | — | 40,634 | — | 40,634 | ps |
| 3.3-V PCI | 20 mA | — | 261 | — | 339 | — | 418 | — | 6,644 | — | 6,627 | — | 6,914 | ps |

**Table 5–31.** MAX II IOE Programmable Delays

| Parameter | MAX II / MAX IIG | | | | | | MAX IIZ | | | | | | Unit |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | –3 Speed Grade | | –4 Speed Grade | | –5 Speed Grade | | –6 Speed Grade | | –7 Speed Grade | | –8 Speed Grade | | |
| | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | Min | Max | |
| Input Delay from Pin to Internal Cells = 1 | — | 1,225 | — | 1,592 | — | 1,960 | — | 1,858 | — | 2,171 | — | 2,214 | ps |
| Input Delay from Pin to Internal Cells = 0 | — | 89 | — | 115 | — | 142 | — | 569 | — | 609 | — | 616 | ps |

## Maximum Input and Output Clock Rates

Table 5–32 and Table 5–33 show the maximum input and output clock rates for standard I/O pins in MAX II devices.

**Table 5–32.** MAX II Maximum Input Clock Rate for I/O

| I/O Standard | | MAX II / MAX IIG | | | MAX IIZ | | | Unit |
|---|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | –4 Speed Grade | –5 Speed Grade | –6 Speed Grade | –7 Speed Grade | –8 Speed Grade | |
| 3.3-V LVTTL | Without Schmitt Trigger | 304 | 304 | 304 | 304 | 304 | 304 | MHz |
| | With Schmitt Trigger | 250 | 250 | 250 | 250 | 250 | 250 | MHz |
| 3.3-V LVCMOS | Without Schmitt Trigger | 304 | 304 | 304 | 304 | 304 | 304 | MHz |
| | With Schmitt Trigger | 250 | 250 | 250 | 250 | 250 | 250 | MHz |
| 2.5-V LVTTL | Without Schmitt Trigger | 220 | 220 | 220 | 220 | 220 | 220 | MHz |
| | With Schmitt Trigger | 188 | 188 | 188 | 188 | 188 | 188 | MHz |
| 2.5-V LVCMOS | Without Schmitt Trigger | 220 | 220 | 220 | 220 | 220 | 220 | MHz |
| | With Schmitt Trigger | 188 | 188 | 188 | 188 | 188 | 188 | MHz |
| 1.8-V LVTTL | Without Schmitt Trigger | 200 | 200 | 200 | 200 | 200 | 200 | MHz |
| 1.8-V LVCMOS | Without Schmitt Trigger | 200 | 200 | 200 | 200 | 200 | 200 | MHz |
| 1.5-V LVCMOS | Without Schmitt Trigger | 150 | 150 | 150 | 150 | 150 | 150 | MHz |
| 3.3-V PCI | Without Schmitt Trigger | 304 | 304 | 304 | 304 | 304 | 304 | MHz |

**Table 5–33.** MAX II Maximum Output Clock Rate for I/O

| I/O Standard | | MAX II / MAX IIG | | | MAX IIZ | | |
|---|---|---|---|---|---|---|---|
| | | –3 Speed Grade | –4 Speed Grade | –5 Speed Grade | –6 Speed Grade | –7 Speed Grade | –8 Speed Grade |
| 3.3-V LVTTL | 304 | 304 | 304 | 304 | 304 | 304 | MHz |
| 3.3-V LVCMOS | 304 | 304 | 304 | 304 | 304 | 304 | MHz |
| 2.5-V LVTTL | 220 | 220 | 220 | 220 | 220 | 220 | MHz |
| 2.5-V LVCMOS | 220 | 220 | 220 | 220 | 220 | 220 | MHz |
| 1.8-V LVTTL | 200 | 200 | 200 | 200 | 200 | 200 | MHz |
| 1.8-V LVCMOS | 200 | 200 | 200 | 200 | 200 | 200 | MHz |
| 1.5-V LVCMOS | 150 | 150 | 150 | 150 | 150 | 150 | MHz |
| 3.3-V PCI | 304 | 304 | 304 | 304 | 304 | 304 | MHz |

## JTAG Timing Specifications

Figure 5–6 shows the timing waveforms for the JTAG signals.

**Figure 5–6.** MAX II JTAG Timing Waveforms

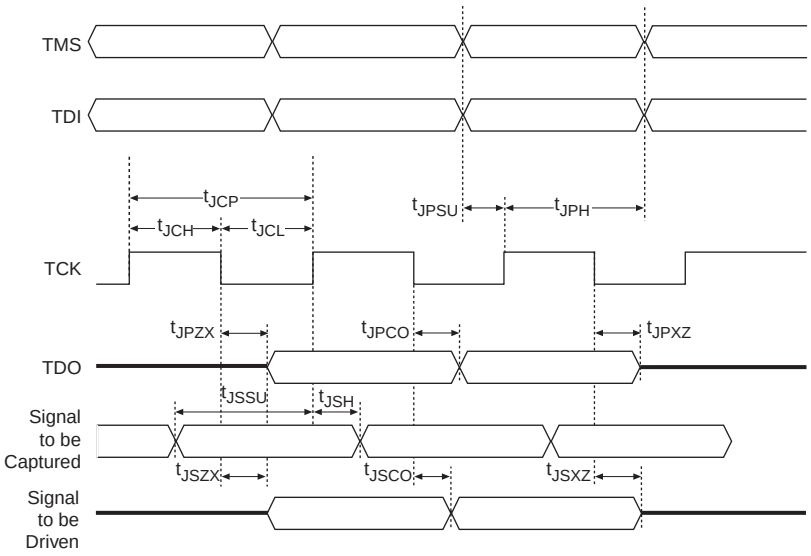



Table 5–34 shows the JTAG Timing parameters and values for MAX II devices.

**Table 5–34.** MAX II JTAG Timing Parameters (Part 1 of 2)

| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| $t_{JCP}$ *(1)* | TCK clock period for $V_{CCIO1}$ = 3.3 V | 55.5 | — | ns |
| | TCK clock period for $V_{CCIO1}$ = 2.5 V | 62.5 | — | ns |
| | TCK clock period for $V_{CCIO1}$ = 1.8 V | 100 | — | ns |
| | TCK clock period for $V_{CCIO1}$ = 1.5 V | 143 | — | ns |
| $t_{JCH}$ | TCK clock high time | 20 | — | ns |
| $t_{JCL}$ | TCK clock low time | 20 | — | ns |

**Table 5–34.** MAX II JTAG Timing Parameters (Part 2 of 2)

| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| $t_{JPSU}$ | JTAG port setup time *(2)* | 8 | — | ns |
| $t_{JPH}$ | JTAG port hold time | 10 | — | ns |
| $t_{JPCO}$ | JTAG port clock to output *(2)* | — | 15 | ns |
| $t_{JPZX}$ | JTAG port high impedance to valid output *(2)* | — | 15 | ns |
| $t_{JPXZ}$ | JTAG port valid output to high impedance *(2)* | — | 15 | ns |
| $t_{JSSU}$ | Capture register setup time | 8 | — | ns |
| $t_{JSH}$ | Capture register hold time | 10 | — | ns |
| $t_{JSCO}$ | Update register clock to output | — | 25 | ns |
| $t_{JSZX}$ | Update register high impedance to valid output | — | 25 | ns |
| $t_{JSXZ}$ | Update register valid output to high impedance | — | 25 | ns |

**Notes to Table 5–34:**

(1) Minimum clock period specified for 10 pF load on the TDO pin. Larger loads on TDO will degrade the maximum TCK frequency.

(2) This specification is shown for 3.3-V LVTTL/LVCMOS and 2.5-V LVTTL/LVCMOS operation of the JTAG pins. For 1.8-V LVTTL/LVCMOS and 1.5-V LVCMOS, the $t_{JPSU}$ minimum is 6 ns and $t_{JPCO}$, $t_{JPZX}$, and $t_{JPXZ}$ are maximum values at 35 ns.

# Referenced Documents

This chapter references the following documents:

- *I/O Structure* section in the *MAX II Architecture* chapter in the *MAX II Device Handbook*
- *Hot Socketing and Power-On Reset in MAX II Devices* chapter in the *MAX II Device Handbook*
- *Operating Requirements for Altera Devices Data Sheet*
- *PowerPlay Power Analysis* chapter in volume 3 of the *Quartus II Handbook*
- *Understanding and Evaluating Power in MAX II Devices* chapter in the *MAX II Device Handbook*
- *Understanding Timing in MAX II Devices* chapter in the *MAX II Device Handbook*
- *Using MAX II Devices in Multi-Voltage Systems* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 5–35 shows the revision history for this chapter.

**Table 5–35.** Document Revision History (Part 1 of 2)

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| August 2009, version 2.5 | ■ Added Table 5–28, Table 5–29, and Table 5–30.<br>■ Updated Table 5–2, Table 5–4, Table 5–14, Table 5–15, Table 5–16, Table 5–17, Table 5–18, Table 5–19, Table 5–20, Table 5–21, Table 5–22, Table 5–23, Table 5–24, Table 5–27, Table 5–31, Table 5–32, and Table 5–33. | Added information for speed grade –8 |
| November 2008, version 2.4 | ■ Updated Table 5–2.<br>■ Updated "Internal Timing Parameters" section. | — |
| October 2008, version 2.3 | ■ Updated New Document Format.<br>■ Updated Figure 5–1. | — |
| July 2008, version 2.2 | ■ Updated Table 5–14 , Table 5–23 , and Table 5–24. | — |
| March 2008, version 2.1 | ■ Added (Note 5) to Table 5–4. | — |
| December 2007, version 2.0 | ■ Updated (Note 3) and (4) to Table 5–1.<br>■ Updated Table 5–2 and added (Note 5).<br>■ Updated ICCSTANDBY and ICCPOWERUP information and added IPULLUP information in Table 5–4.<br>■ Added (Note 1) to Table 5–10.<br>■ Updated Figure 5–2.<br>■ Added (Note 1) to Table 5–13.<br>■ Updated Table 5–13 through Table 5–24, and Table 5–27 through Table 5–30.<br>■ Added tCOMB information to Table 5–15.<br>■ Updated Figure 5–6.<br>■ Added "Referenced Documents" section. | Updated document with MAX IIZ information. |
| December 2006, version 1.8 | ■ Added note to Table 5–1.<br>■ Added document revision history. | — |
| July 2006, version 1.7 | ■ Minor content and table updates. | — |
| February 2006, version 1.6 | ■ Updated "External Timing I/O Delay Adders" section.<br>■ Updated Table 5–29.<br>■ Updated Table 5–30. | — |
| November 2005, version 1.5 | ■ Updated Tables 5-2, 5-4, and 5-12. | — |
| August 2005, version 1.4 | ■ Updated Figure 5-1.<br>■ Updated Tables 5-13, 5-16, and 5-26.<br>■ Removed Note 1 from Table 5-12. | — |

**Table 5–35.** Document Revision History (Part 2 of 2)

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| June 2005, version 1.3 | ■ Updated the $R_{PULLUP}$ parameter in Table 5-4.<br>■ Added Note 2 to Tables 5-8 and 5-9.<br>■ Updated Table 5-13.<br>■ Added "Output Drive Characteristics" section.<br>■ Added $I^2C$ mode and Notes 5 and 6 to Table 5-14.<br>■ Updated timing values to Tables 5-14 through 5-33. | — |
| December 2004, version 1.2 | ■ Updated timing Tables 5-2, 5-4, 5-12, and Tables 15-14 through 5-34.<br>■ Table 5-31 is new. | — |
| June 2004, version 1.1 | ■ Updated timing Tables 5-15 through 5-32. | — |

# 6. Reference and Ordering Information

MII51006-1.6

## Software

MAX® II devices are supported by the Altera® Quartus® II design software with new, optional MAX+PLUS® II look and feel, which provides HDL and schematic design entry, compilation and logic synthesis, full simulation and advanced timing analysis, and device programming. Refer to the Design Software Selector Guide for more details about the Quartus II software features.

The Quartus II software supports the Windows XP/2000/NT, Sun Solaris, Linux Red Hat v8.0, and HP-UX operating systems. It also supports seamless integration with industry-leading EDA tools through the NativeLink interface.

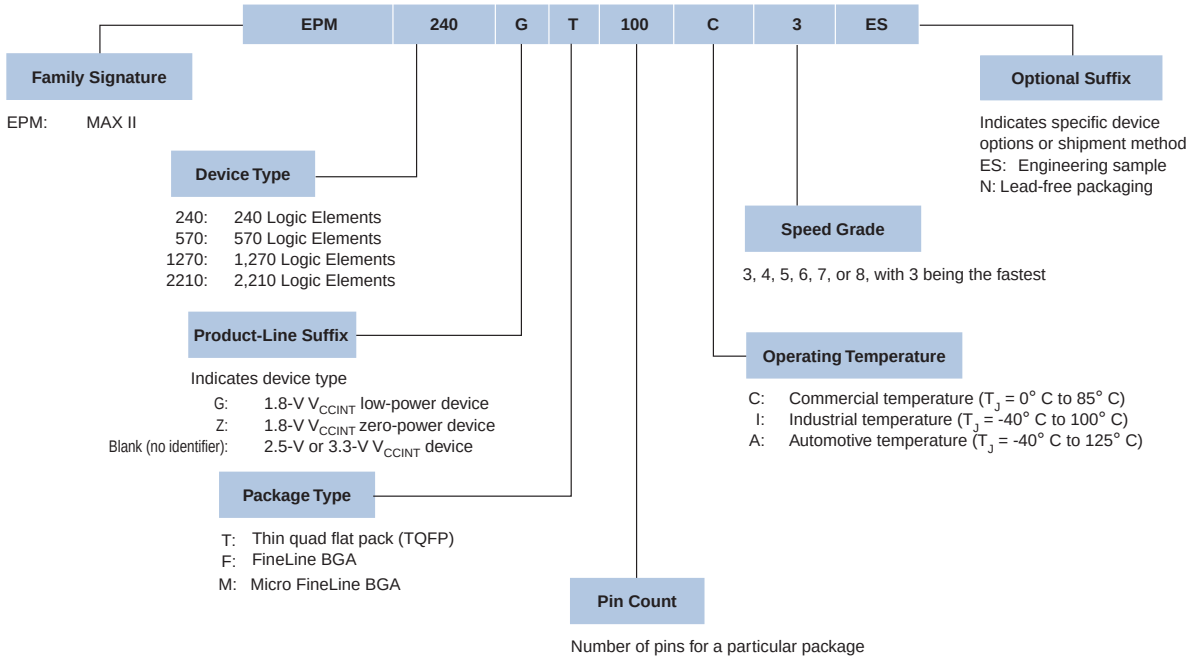
## Device Pin-Outs

Printed device pin-outs for MAX II devices are available on the Altera website (www.altera.com).

## Ordering Information

Figure 6–1 describes the ordering codes for MAX II devices. For more information about a specific package, refer to the *Package Information* chapter in the *MAX II Device Handbook*.

**Figure 6–1.** MAX II Device Packaging Ordering Information



# Referenced Documents

This chapter references the following document:

- *Package Information* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 6–1 shows the revision history for this chapter.

**Table 6–1.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| August 2009, version 1.6 | ■ Updated Figure 6–1. | Added information for speed grade –8 |
| October 2008, version 1.5 | ■ Updated New Document Format. | — |
| December 2007, version 1.4 | ■ Added "Referenced Documents" section.<br>■ Updated Figure 6–1. | Updated document with MAX IIZ information. |
| December 2006, version 1.3 | ■ Added document revision history. | — |
| October 2006, version 1.2 | ■ Updated Figure 6-1. | — |
| June 2005, version 1.1 | ■ Removed Dual Marking section. | — |

# Section II. PCB Layout Guidelines

This section provides information for board layout designers to successfully layout their boards for MAX® II devices. It contains the required printed circuit board (PCB) layout guidelines, device pin tables, and package specifications.

This section includes the following chapters:

- Chapter 7, Package Information
- Chapter 8, Using MAX II Devices in Multi-Voltage Systems

## Revision History

Refer to each chapter for its own specific revision history. For information about when each chapter was updated, refer to the Chapter Revision Dates section, which appears in the complete handbook.

# 7. Package Information

MII51007-2.1

## Introduction

This chapter provides package information for Altera's MAX® II devices, and includes these sections:


- "Board Decoupling Guidelines" on page 7–1
- "Device and Package Cross Reference" on page 7–1
- "Thermal Resistance" on page 7–2
- "Package Outlines" on page 7–3


In this chapter, packages are listed in order of ascending pin count. See Figure 7–1 through 7–17.

## Board Decoupling Guidelines

Decoupling requirements are based on the amount of logic used in the device and the output switching requirements. As the number of I/O pins and the capacitive load on the pins increase, more decoupling capacitance is required. As many as possible 0.1-mF power-supply decoupling capacitors should be connected to the VCC and GND pins or the VCC and GND planes. These capacitors should be located as close as possible to the MAX II device. Each VCCINT/GNDINT and VCCIO/GNDIO pair should be decoupled with a 0.1-mF capacitor. When using high-density packages, such as ball-grid array (BGA) packages, it may not be possible to use one decoupling capacitor per VCC/GND pair. In this case, you should use as many decoupling capacitors as possible. For less dense designs, a reduction in the number of capacitors may be acceptable. Decoupling capacitors should have a good frequency response, such as monolithic-ceramic capacitors.

## Device and Package Cross Reference

Table 7–1 shows which Altera® MAX II devices are available in thin quad flat pack (TQFP), FineLine BGA (FBGA), and Micro Fineline BGA (MBGA) packages.

**Table 7–1.** MAX II Devices in TQFP, FineLine BGA, and Micro FineLine BGA Packages (Part 1 of 2)

| Device | Package | Pin |
|---|---|---|
| EPM240Z | MBGA *(1)* | 68 |
| EPM240<br>EPM240G | FBGA *(1)* | 100 |
| EPM240<br>EPM240G<br>EPM240Z | MBGA *(1)* | 100 |
| EPM240<br>EPM240G | TQFP | 100 |

**Table 7–1.** MAX II Devices in TQFP, FineLine BGA, and Micro FineLine BGA Packages (Part 2 of 2)

| Device | Package | Pin |
|---|---|---|
| EPM570<br>EPM570G | FBGA *(1)* | 100 |
| EPM570<br>EPM570G<br>EPM570Z | MBGA *(1)* | 100 |
| EPM570<br>EPM570G | TQFP | 100 |
| EPM570Z | MBGA *(1)* | 144 |
| EPM570<br>EPM570G | TQFP | 144 |
| EPM570<br>EPM570G | FBGA | 256 |
| EPM570<br>EPM570G<br>EPM570Z | MBGA *(1)* | 256 |
| EPM1270<br>EPM1270G | TQFP | 144 |
| | FBGA | 256 |
| | MBGA *(1)* | 256 |
| EPM2210 | FBGA | 256 |
| EPM2210G | FBGA | 324 |

**Note to Table 7–1:**

(1) Packages available in lead-free versions only.

# Thermal Resistance

Table 7–2 provides $\theta_{JA}$ (junction-to-ambient thermal resistance) and $\theta_{JC}$ (junction-to-case thermal resistance) values for Altera MAX II devices.

**Table 7–2.** Thermal Resistance of MAX II Devices (Part 1 of 2)

| Device | Pin Count | Package | $\theta_{JC}$ (°C/W) | $\theta_{JA}$ (°C/W) Still Air | $\theta_{JA}$ (°C/W) 100 ft./min. | $\theta_{JA}$ (°C/W) 200 ft./min. | $\theta_{JA}$ (°C/W) 400 ft./min. |
|---|---|---|---|---|---|---|---|
| EPM240Z | 68 | MBGA | 35.5 | 68.7 | 63.0 | 60.9 | 59.2 |
| EPM240<br>EPM240G | 100 | FBGA | 20.8 | 51.2 | 45.2 | 43.2 | 41.5 |
| EPM240<br>EPM240G<br>EPM240Z | 100 | MBGA | 32.1 | 53.8 | 47.7 | 45.7 | 44.0 |
| EPM240<br>EPM240G | 100 | TQFP | 12.0 | 39.5 | 37.5 | 35.5 | 31.6 |
| EPM570<br>EPM570G | 100 | FBGA | 14.8 | 42.8 | 36.8 | 34.9 | 33.3 |

**Table 7–2.** Thermal Resistance of MAX II Devices (Part 2 of 2)

| Device | Pin Count | Package | $\theta_{JC}$ (°C/W) | $\theta_{JA}$ (°C/W) Still Air | $\theta_{JA}$ (°C/W) 100 ft./min. | $\theta_{JA}$ (°C/W) 200 ft./min. | $\theta_{JA}$ (°C/W) 400 ft./min. |
|---|---|---|---|---|---|---|---|
| EPM570<br>EPM570G<br>EPM570Z | 100 | MBGA | 25.0 | 46.5 | 40.4 | 38.4 | 36.8 |
| EPM570<br>EPM570G | 100 | TQFP | 11.2 | 38.7 | 36.6 | 34.6 | 30.8 |
| EPM570Z | 144 | MBGA | 20.2 | 51.8 | 45.1 | 43.2 | 41.5 |
| EPM570<br>EPM570G | 144 | TQFP | 10.5 | 32.1 | 30.3 | 28.7 | 26.1 |
| EPM570<br>EPM570G | 256 | FBGA | 13.0 | 37.4 | 33.1 | 30.5 | 28.4 |
| EPM570<br>EPM570G<br>EPM570Z | 256 | MBGA | 12.9 | 39.5 | 33.6 | 31.6 | 30.1 |
| EPM1270 | 144 | TQFP | 10.5 | 31.4 | 29.7 | 28.2 | 25.8 |
| EPM1270G | 256 | FBGA | 10.4 | 33.5 | 29.3 | 26.8 | 24.7 |
| | 256 | MBGA | 10.6 | 36.1 | 30.2 | 28.3 | 26.8 |
| EPM2210 | 256 | FBGA | 8.7 | 30.2 | 26.1 | 23.6 | 21.7 |
| EPM2210G | 324 | FBGA | 8.2 | 29.8 | 25.7 | 23.3 | 21.3 |

# Package Outlines

The package outlines on the following pages are listed in order of ascending pin count. Altera package outlines meet the requirements of JEDEC Publication No. 95.

## 68-Pin Micro FineLine Ball-Grid Array (MBGA) – Wire Bond

- All dimensions and tolerances conform to ASME Y14.5M – 1994
- Controlling dimension is in millimeters
- Pin A1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

*Package Information (Part 1 of 2)*

| Description | Specification |
|---|---|
| Ordering Code Reference | M |
| Package Acronym | MBGA |
| Substrate Material | BT |
| Solder Ball Composition | Pb-free: Sn:3Ag:0.5Cu (Typ.) |
| JEDEC Outline Reference | MO-195 Variation: AB |

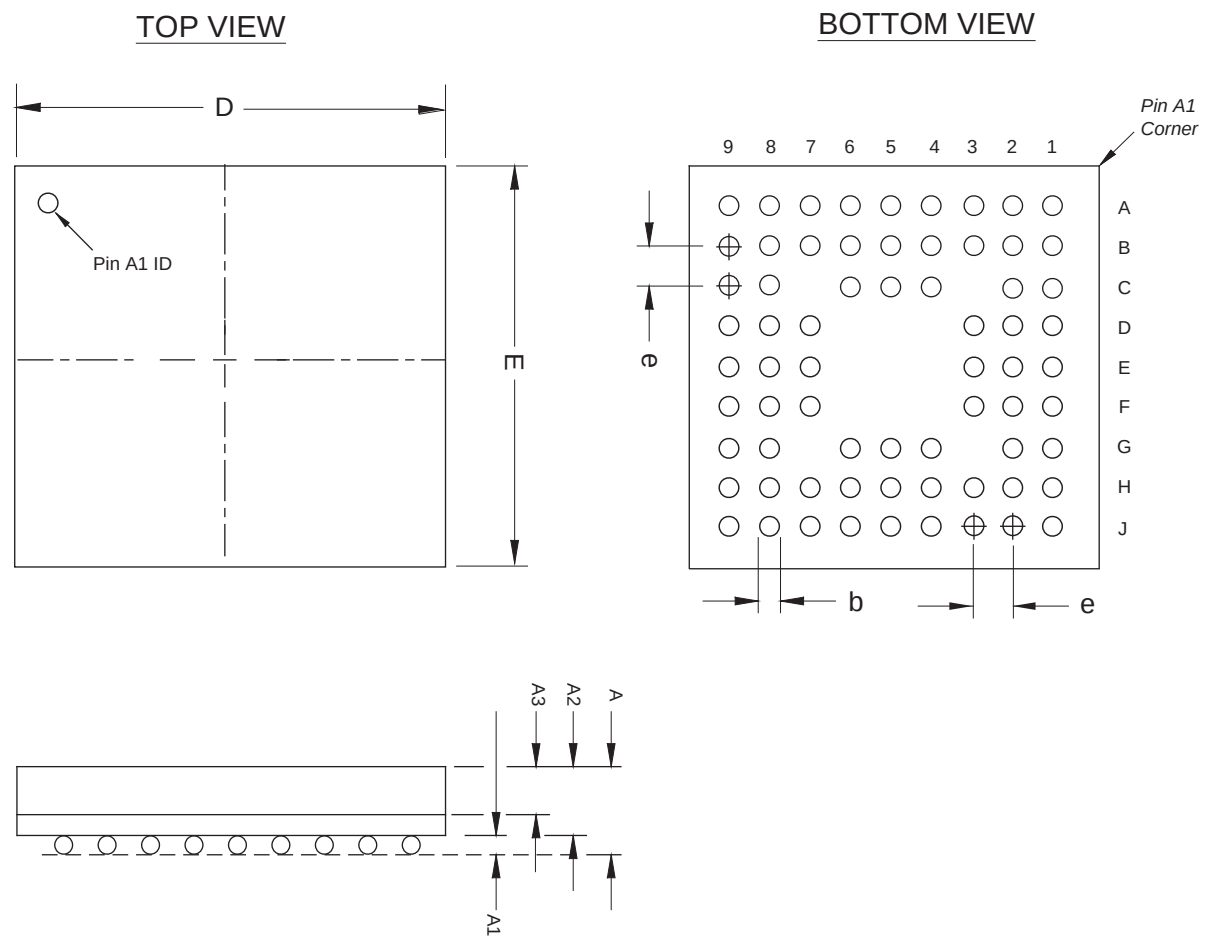
*Package Outline Dimension Table (Part 1 of 2)*

| Symbol | Millimeters | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | — | 1.20 |
| A1 | 0.15 | — | — |
| A2 | — | — | 1.00 |
| A3 | 0.60 REF | | |

| Package Information (Part 2 of 2) | |
|---|---|
| Maximum Lead Coplanarity | 0.003 inches (0.08 mm) |
| Weight | 0.1 g |
| Moisture Sensitivity Level | Printed on moisture barrier bag |

| Package Outline Dimension Table (Part 2 of 2) | | | |
|---|---|---|---|
| D | 5.00 BSC | | |
| E | 5.00 BSC | | |
| b | 0.25 | 0.30 | 0.35 |
| e | 0.50 BSC | | |

**Figure 7–1.** 68-Pin Micro FineLine BGA Package Outline



## 100-Pin Plastic Thin Quad Flat Pack (TQFP)

- All dimensions and tolerances conform to ANSI Y14.5M – 1994
- Controlling dimension is in millimeters
- Pin 1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

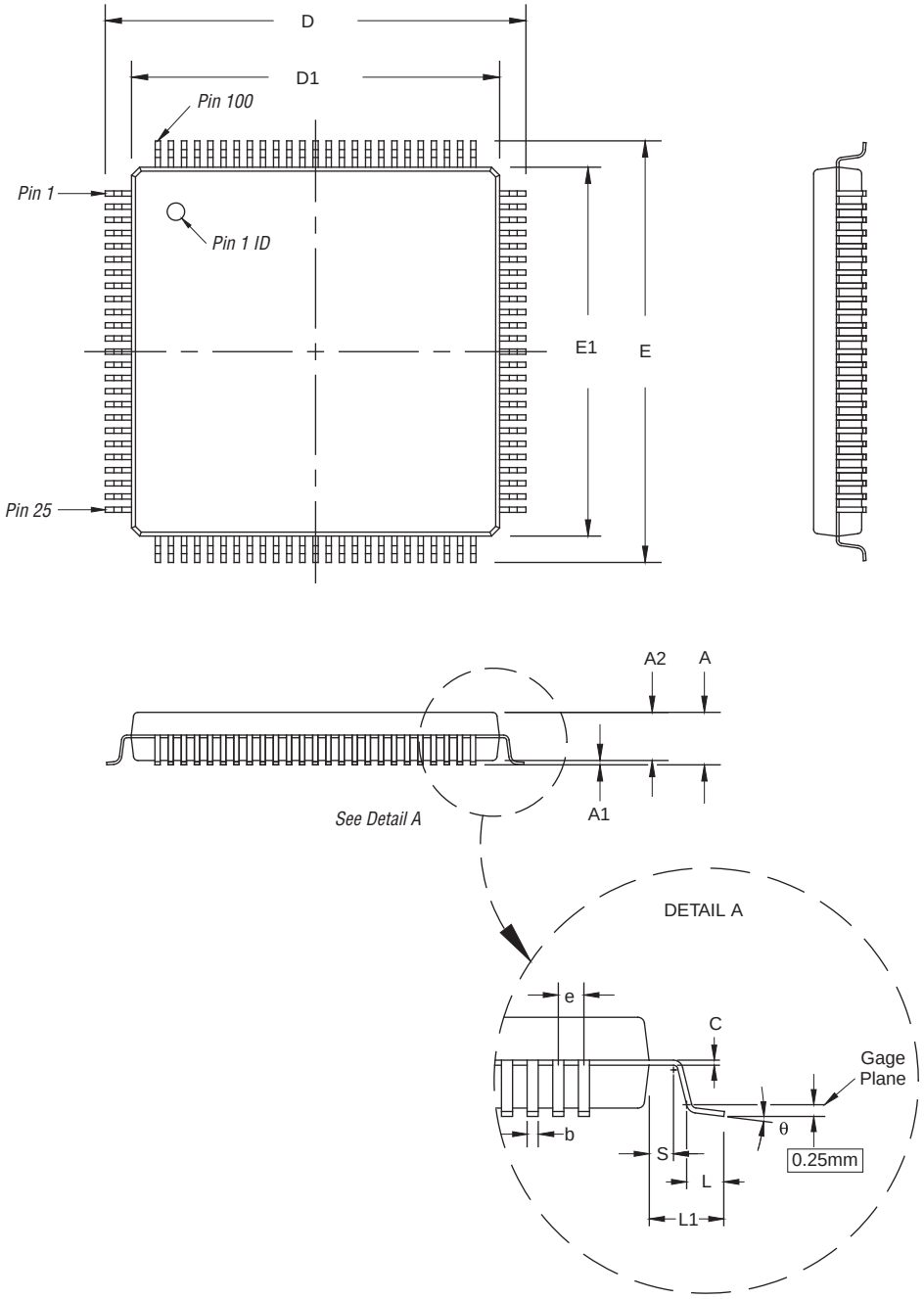
***Package Information***

| Description | Specification |
|---|---|
| Ordering Code Reference | T |
| Package Acronym | TQFP |
| Leadframe Material | Copper |
| Lead Finish (Plating) | Regular: 85Sn:15Pb (Typ.)<br>Pb-free: Matte Sn |
| JEDEC Outline Reference | MS-026 Variation: AED |
| Maximum Lead Coplanarity | 0.003 inches (0.08mm) |
| Weight | 0.6 g |
| Moisture Sensitivity Level | Printed on moisture barrier bag |

***Package Outline Dimension Table***

| Symbol | Millimeters | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | — | 1.20 |
| A1 | 0.05 | — | 0.15 |
| A2 | 0.95 | 1.00 | 1.05 |
| D | | 16.00 BSC | |
| D1 | | 14.00 BSC | |
| E | | 16.00 BSC | |
| E1 | | 14.00 BSC | |
| L | 0.45 | 0.60 | 0.75 |
| L1 | | 1.00 REF | |
| S | 0.20 | — | — |
| b | 0.17 | 0.22 | 0.27 |
| c | 0.09 | — | 0.20 |
| e | | 0.50 BSC | |
| θ | 0° | 3.5° | 7° |

**Figure 7–2.** 100-Pin TQFP Package Outline



## 100-Pin Micro FineLine Ball-Grid Array (MBGA)

- All dimensions and tolerances conform to ASME Y14.5 – 1994.
- Controlling dimension is in millimeters.
- Pin A1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

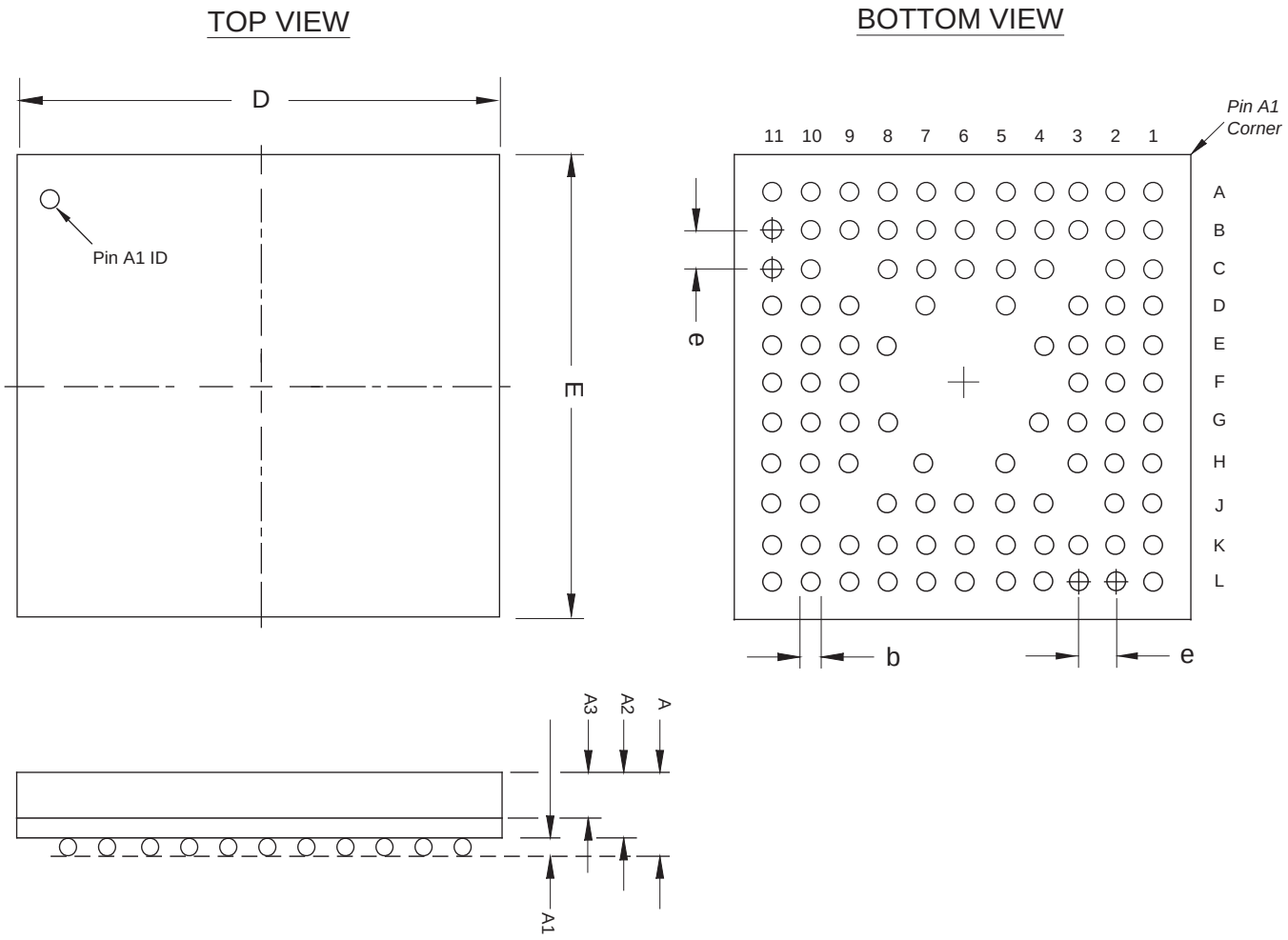
***Package Information***

| Description | Specification |
|---|---|
| Ordering Code Reference | M |
| Package Acronym | MBGA |
| Substrate Material | BT |
| Solder Ball Composition | Pb-free: Sn:3Ag:0.5Cu (Typ.) |
| JEDEC Outline Reference | MO-195 Variation: AC |
| Maximum Lead Coplanarity | 0.003 inches (0.08 mm) |
| Weight | 0.1 g |
| Moisture Sensitivity Level | Printed on moisture barrier bag |

***Package Outline Dimension Table***

| Symbol | Millimeters | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | — | 1.20 |
| A1 | 0.15 | — | — |
| A2 | — | — | 1.00 |
| A3 | 0.60 REF | | |
| D | 6.00 BSC | | |
| E | 6.00 BSC | | |
| b | 0.25 | 0.30 | 0.35 |
| e | 0.50 BSC | | |

**Figure 7–3.** 100-Pin Micro FineLine BGA Package Outline




## 100-Pin FineLine Ball-Grid Array (FBGA)

- All dimensions and tolerances conform to ASME Y14.5 – 1994
- Controlling dimension is in millimeters
- Pin A1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

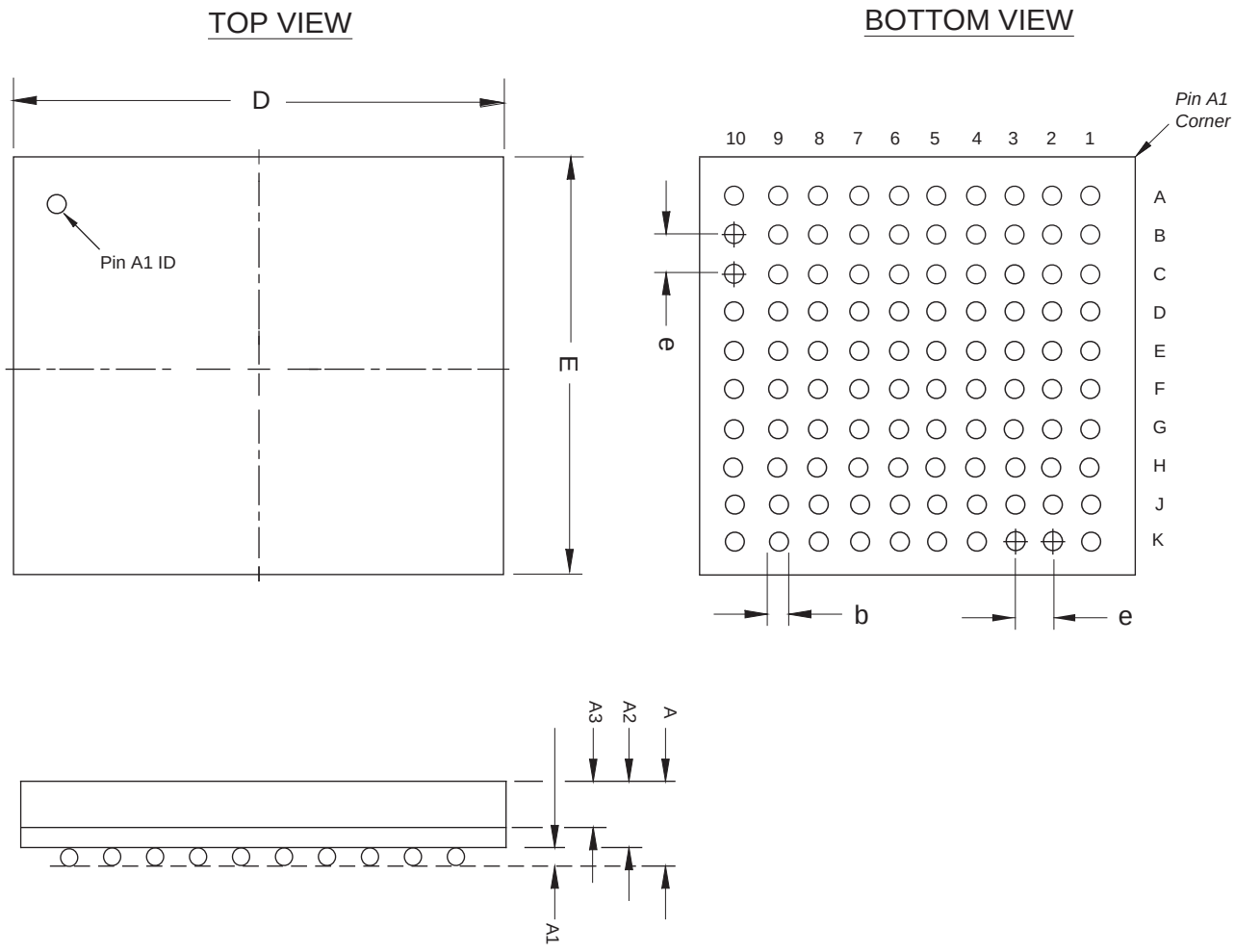
| *Package Information* | |
|---|---|
| **Description** | **Specification** |
| Ordering Code Reference | F |
| Package Acronym | FBGA |
| Substrate Material | BT |

| *Package Outline Dimension Table* | | | |
|---|---|---|---|
| **Symbol** | **Millimeters** | | |
| | **Min.** | **Nom.** | **Max.** |
| A | — | — | 1.55 |
| A1 | 0.25 | — | — |

| *Package Information* | |
|---|---|
| Solder Ball Composition | Regular: 63Sn:37Pb (Typ.)<br>Pb-free: Sn:3Ag:0.5Cu (Typ.) |
| JEDEC Outline Reference | MO-192 Variation: DAC-1 |
| Maximum Lead Coplanarity | 0.008 inches (0.20 mm) |
| Weight | 0.6 g |
| Moisture Sensitivity Level | Printed on moisture barrier bag |

| *Package Outline Dimension Table* | | | |
|---|---|---|---|
| A2 | 1.05 REF | | |
| A3 | — | — | 0.80 |
| D | 11.00 BSC | | |
| E | 11.00 BSC | | |
| b | 0.45 | 0.50 | 0.55 |
| e | 1.00 BSC | | |

**Figure 7–4.** 100-Pin FineLine BGA Package Outline



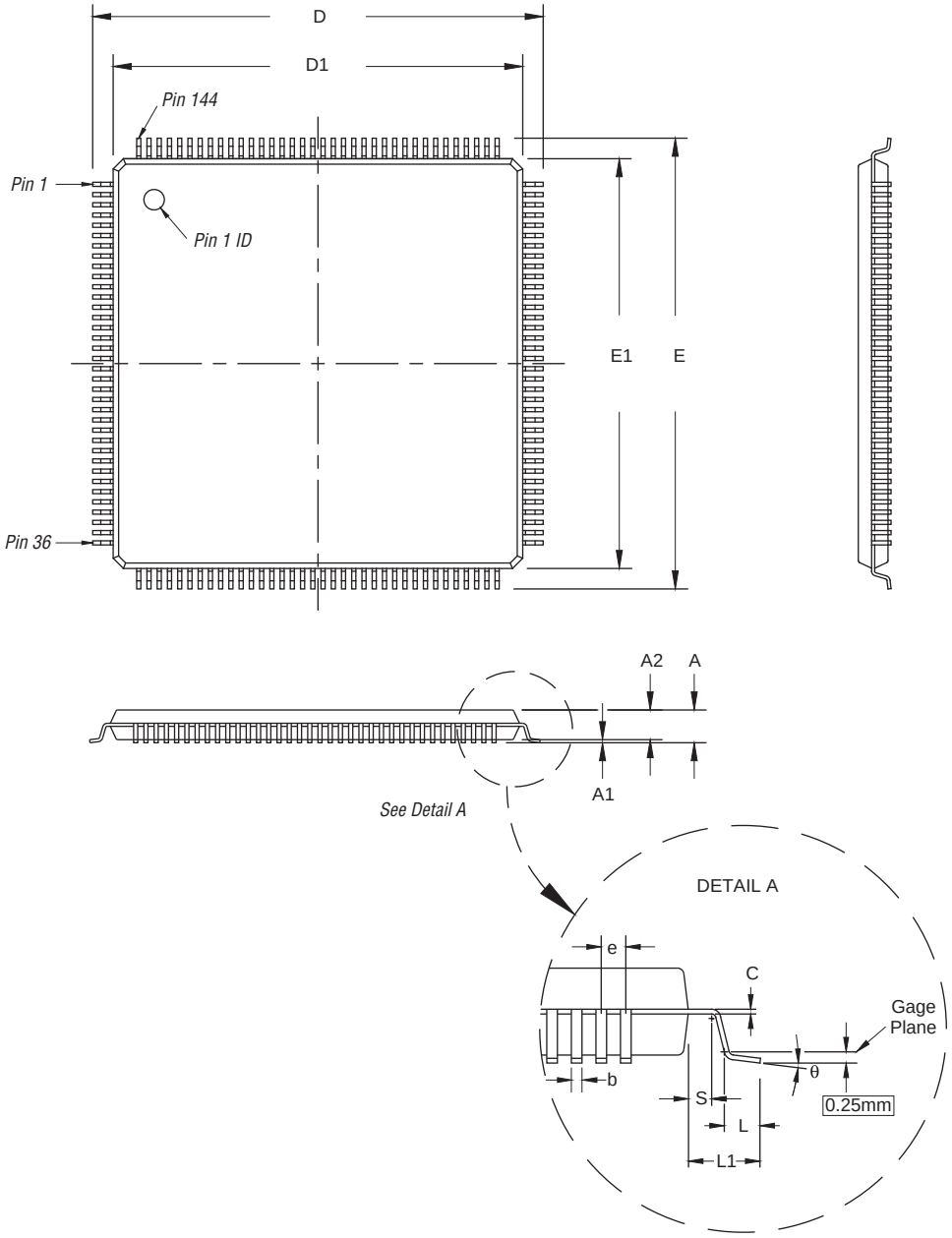
## 144-Pin Plastic Thin Quad Flat Pack (TQFP)

- All dimensions and tolerances conform to ANSI Y14.5M – 1994
- Controlling dimension is in millimeters
- Pin 1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

| *Package Information* | |
|---|---|
| **Description** | **Specification** |
| Ordering Code Reference | T |
| Package Acronym | TQFP |
| Leadframe Material | Copper |
| Lead Finish (Plating) | Regular: 85Sn:15Pb (Typ.)<br>Pb-free: Matte Sn |
| JEDEC Outline Reference | MS-026 Variation: BFB |
| Maximum Lead Coplanarity | 0.003 inches (0.08 mm) |
| Weight | 1.1 g |
| Moisture Sensitivity Level | Printed on moisture barrier bag |

| *Package Outline Figure Reference* | | | |
|---|---|---|---|
| **Symbol** | **Millimeters** | | |
| | **Min.** | **Nom.** | **Max.** |
| A | — | — | 1.60 |
| A1 | 0.05 | — | 0.15 |
| A2 | 1.35 | 1.40 | 1.45 |
| D | | 22.00 BSC | |
| D1 | | 20.00 BSC | |
| E | | 22.00 BSC | |
| E1 | | 20.00 BSC | |
| L | 0.45 | 0.60 | 0.75 |
| L1 | | 1.00 REF | |
| S | 0.20 | — | — |
| b | 0.17 | 0.22 | 0.27 |
| c | 0.09 | — | 0.20 |
| e | | 0.50 BSC | |
| θ | 0° | 3.5° | 7° |

**Figure 7–5.** 144-Pin TQFP Package Outline



## 144-Pin Micro FineLine Ball-Grid Array (MBGA) – Wire Bond

- All dimensions and tolerances conform to ASME Y14.5M – 1994.
- Controlling dimension is in millimeters.
- Pin A1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

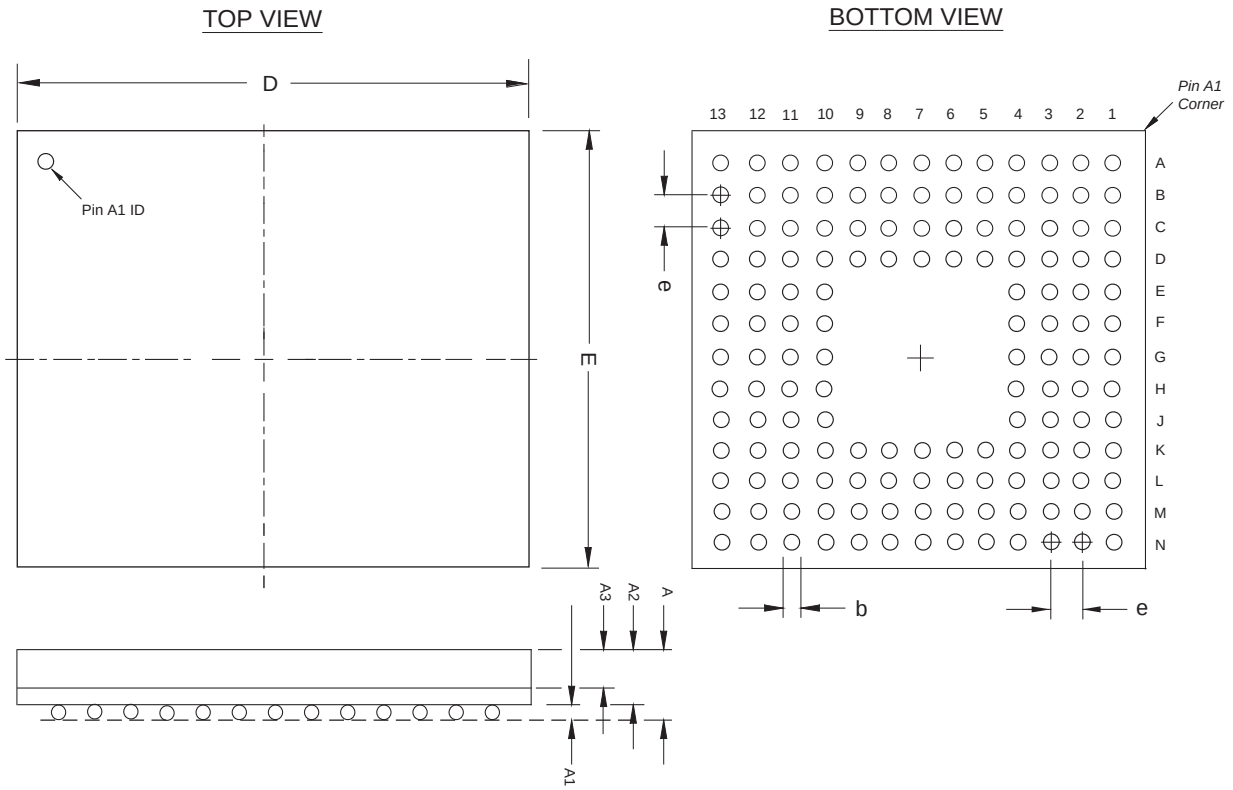
***Package Information***

| Description | Specification |
|---|---|
| Ordering Code Reference | M |
| Package Acronym | MBGA |
| Substrate Material | BT |
| Solder Ball Composition | Pb-free: Sn:3Ag:0.5Cu (Typ.) |
| JEDEC Outline Reference | MO-195 Variation: AD |
| Maximum Lead Coplanarity | 0.003 inches (0.08 mm) |
| Weight | 0.1 g |
| Moisture Sensitivity Level | Printed on moisture barrier bag |

***Package Outline Dimension Table***

| Symbol | Millimeters | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | — | 1.20 |
| A1 | 0.15 | — | — |
| A2 | — | — | 1.00 |
| A3 | | 0.60 REF | |
| D | | 7.00 BSC | |
| E | | 7.00 BSC | |
| b | 0.25 | 0.30 | 0.35 |
| e | | 0.50 BSC | |

**Figure 7–6.** 144-Pin Micro FineLine BGA Package Outline




## 256-Pin Micro FineLine Ball-Grid Array (MBGA)

- All dimensions and tolerances conform to ASME Y14.5 – 1994
- Controlling dimension is in millimeters
- Pin A1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

*Package Information (Part 1 of 2)*

| Description | Specification |
|---|---|
| Ordering Code Reference | M |
| Package Acronym | MBGA |
| Substrate Material | BT |
| Solder Ball Composition | Pb-free: Sn:3Ag:0.5Cu (Typ.) |
| JEDEC Outline Reference | MO-192 Variation: BH |
| Maximum Lead Coplanarity | 0.003 inches (0.08 mm) |
| Weight | 0.3 g |

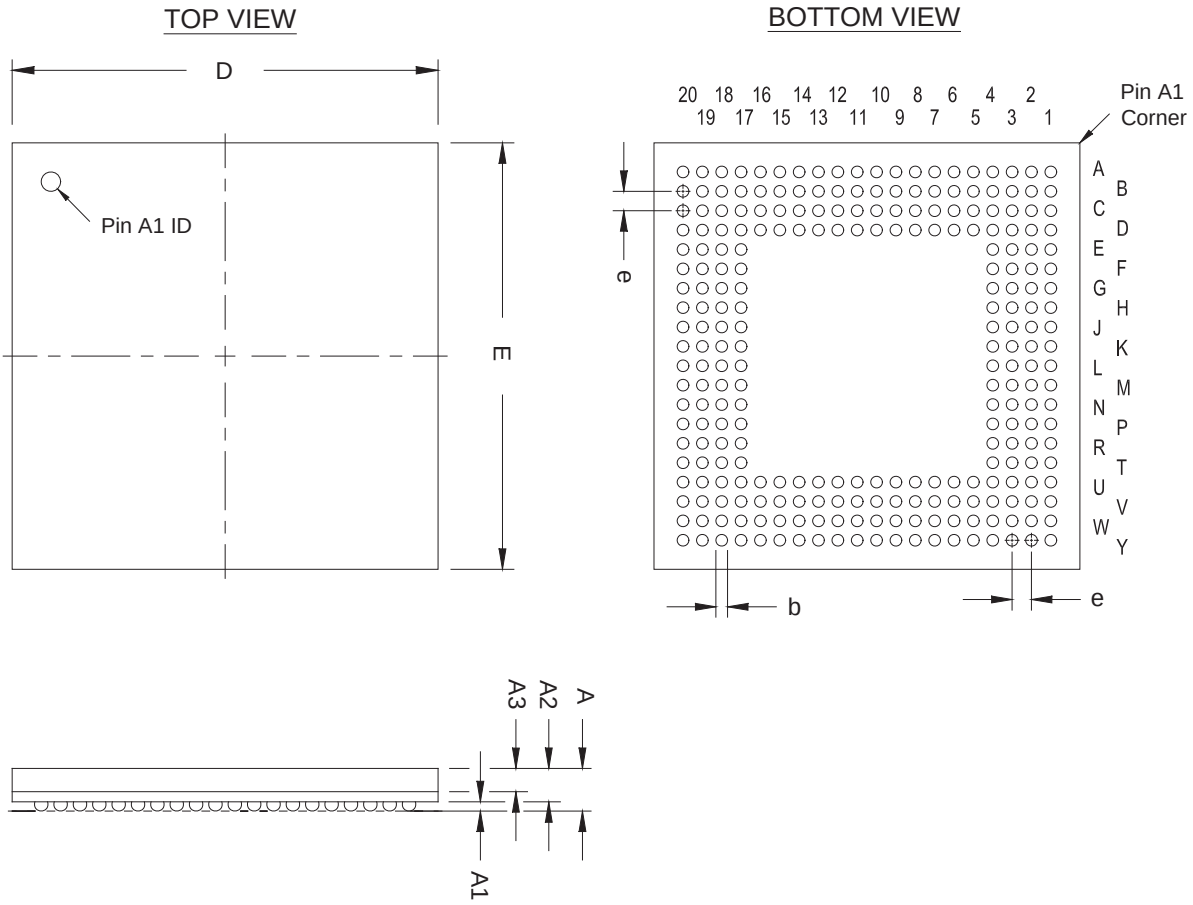
*Package Outline Dimension Table (Part 1 of 2)*

| Symbol | Millimeters | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | — | 1.20 |
| A1 | 0.15 | — | — |
| A2 | — | — | 1.00 |
| A3 | 0.60 REF | | |
| D | 11.00 BSC | | |
| E | 11.00 BSC | | |

| Package Information (Part 2 of 2) | |
|---|---|
| Moisture Sensitivity Level | Printed on moisture barrier bag |

| Package Outline Dimension Table (Part 2 of 2) | | | |
|---|---|---|---|
| b | 0.25 | 0.30 | 0.35 |
| e | 0.50 BSC | | |

**Figure 7–7.** 256-Pin Micro FineLine BGA Package Outline



## 256-Pin FineLine Ball-Grid Array (FBGA)

- All dimensions and tolerances conform to ANSI Y14.5M – 1994
- Controlling dimension is in millimeters
- Pin A1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

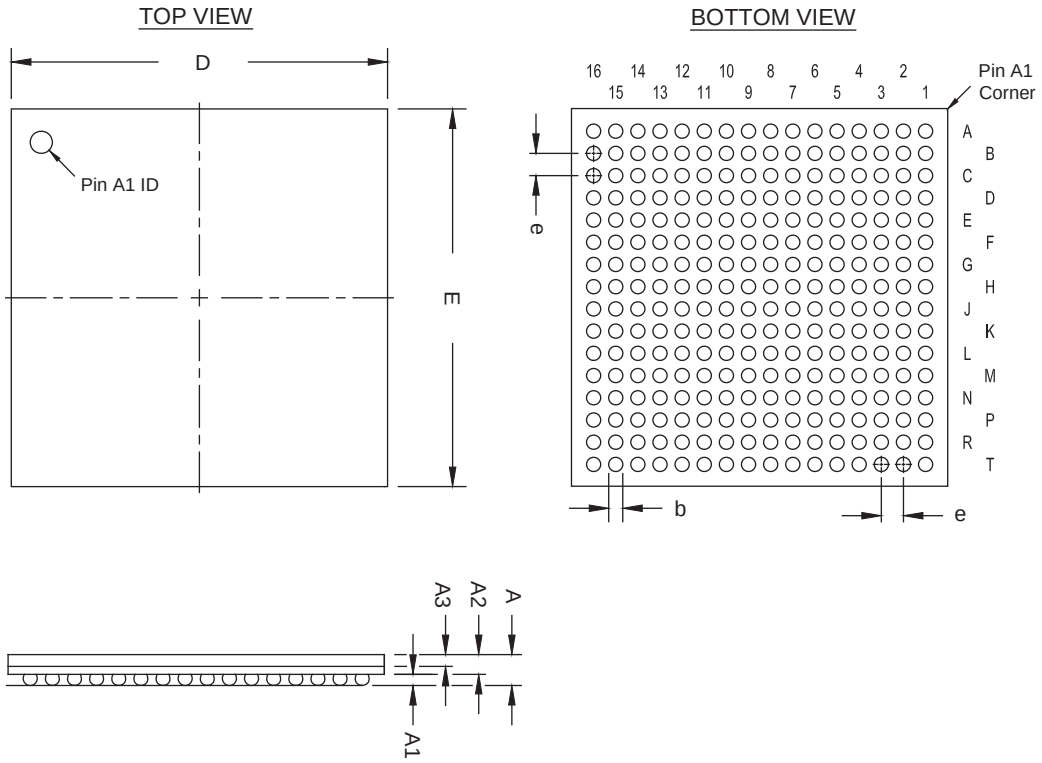
***Package Information***

| Description | Specification |
|---|---|
| Ordering Code Reference | F |
| Package Acronym | FBGA |
| Substrate Material | BT |
| Solder Ball Composition | Regular: 63Sn:37Pb (Typ.)<br>Pb-free: Sn:3Ag:0.5Cu (Typ.) |
| JEDEC Outline Reference | MS-034 Variation: AAF-1 |
| Maximum Lead Coplanarity | 0.008 inches (0.20 mm) |
| Weight | 1.5 g |
| Moisture Sensitivity Level | Printed on moisture barrier bag |

***Package Outline Dimension Table***

| | Millimeters | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | — | 2.20 |
| A1 | 0.30 | — | — |
| A2 | — | — | 1.80 |
| A3 | | 0.70 REF | |
| D | | 17.00 BSC | |
| E | | 17.00 BSC | |
| b | 0.50 | 0.60 | 0.70 |
| e | | 1.00 BSC | |

**Figure 7–8.** 256-Pin FineLine BGA Package Outline




## 324-Pin FineLine Ball-Grid Array (FBGA)

- All dimensions and tolerances conform to ANSI Y14.5M – 1994
- Controlling dimension is in millimeters
- Pin A1 may be indicated by an ID dot, or a special feature, in its proximity on package surface

*Package Information (Part 1 of 2)*

| Description | Specification |
|---|---|
| Ordering Code Reference | F |
| Package Acronym | FBGA |
| Substrate Material | BT |
| Solder Ball Composition | Regular: 63Sn:37Pb (Typ.)<br>Pb-free: Sn:3Ag:0.5Cu (Typ.) |
| JEDEC Outline Reference | MS-034 Variation: AAG-1 |
| Maximum Lead Coplanarity | 0.008 inches (0.20 mm) |
| Weight | 1.6 g |

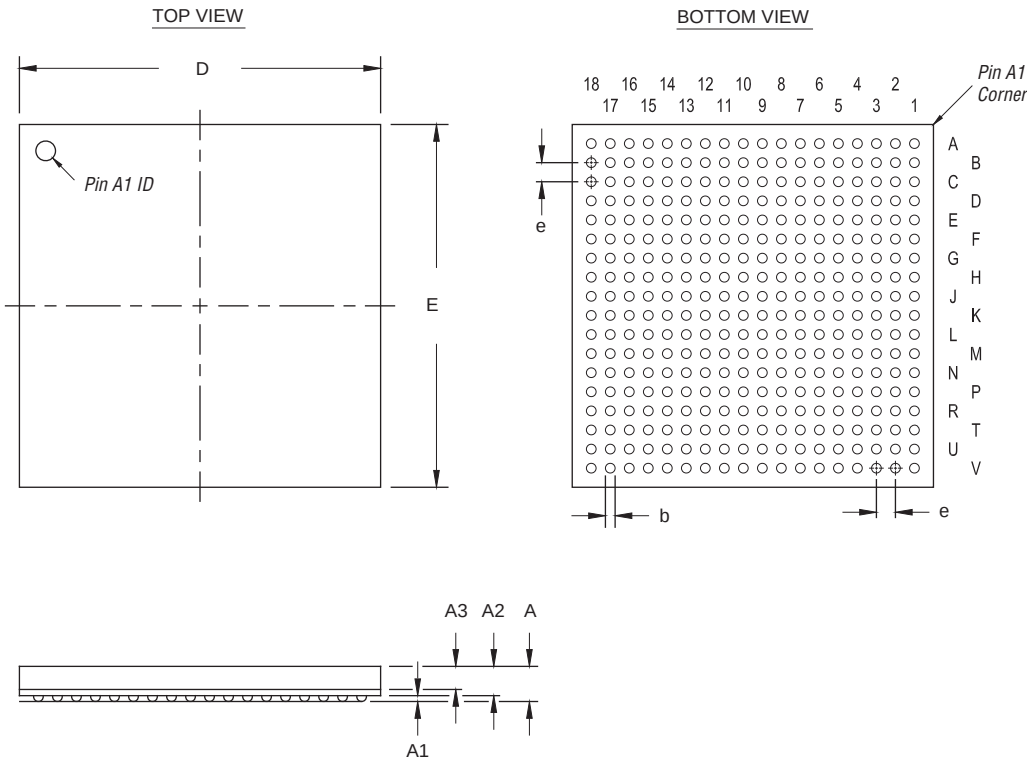
*Package Outline Dimension Table (Part 1 of 2)*

| Symbol | Millimeters | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | — | 2.20 |
| A1 | 0.30 | — | — |
| A2 | — | — | 1.80 |
| A3 | 0.70 REF | | |
| D | 19.00 BSC | | |
| E | 19.00 BSC | | |
| b | 0.50 | 0.60 | 0.70 |

| *Package Information (Part 2 of 2)* | |
|---|---|
| Moisture Sensitivity Level | Printed on moisture barrier bag |

| *Package Outline Dimension Table (Part 2 of 2)* | |
|---|---|
| e | 1.00 BSC |

**Figure 7–9.** 324-Pin FineLine BGA Package Outline



# Document Revision History

Table 7–3 shows the revision history for this chapter.

**Table 7–3.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 2.1 | ■ Updated New Document Format. | — |
| December 2007, version 2.0 | ■ Updated Table 7–1 and Table 7–2.<br>■ Added "68-Pin Micro FineLine Ball-Grid Array (MBGA) – Wire Bond" and "144-Pin Micro FineLine Ball-Grid Array (MBGA) – Wire Bond" sections.<br>■ Replaced Figure 7–9 with correct diagram. | ■ Updated document with MAX IIZ information.<br>■ Added information about 68-Pin Micro FineLine Ball-Grid Array and 144-Pin Micro FineLine Ball-Grid Array. |
| December 2006, version 1.4 | ■ Added document revision history. | — |
| July 2006, version 1.3 | ■ Updated packaging information. | — |
| August 2005, version 1.2 | ■ Updated the 100-pin plastic thin quad flat pack (TQFP) information. | — |
| December 2004, version 1.1 | ■ Updated Board Decoupling Guidelines section (changed the 0.2 value to 0.1.) | — |

# 8. Using MAX II Devices in Multi-Voltage Systems

MII51009-1.7

## Introduction

Technological advancements in deep submicron processes have lowered the supply voltage levels of semiconductor devices, creating a design environment where devices on a system board may potentially use many different supply voltages such as 5.0, 3.3, 2.5, 1.8, and 1.5 V, which can ultimately lead to voltage conflicts.

To accommodate interfacing with a variety of devices on system boards, MAX® II devices have MultiVolt I/O interfaces that allow devices in a mixed-voltage design environment to communicate directly with MAX II devices. The MultiVolt interface separates the power supply voltage ($V_{CCINT}$) from the output voltage ($V_{CCIO}$), enabling MAX II devices to interface with other devices using a different voltage level on the same printed circuit board (PCB).

Additionally, the MAX II device family supports the MultiVolt core feature. For 1.8-V operation, use the MAX IIG or MAX IIZ devices. The 1.8-V input directly powers the core of the devices. For 2.5-V or 3.3-V operation, use the MAX II devices. MAX II devices that support 2.5-V and 3.3-V operation have an internal voltage regulator that regulates at 1.8 V.

This chapter discusses several features that allow you to implement Altera® devices in multiple-voltage systems without damaging the device or the system, including:

- Hot Socketing—Insert or remove MAX II devices to and from a powered-up system without affecting the device or system operation
- Power-Up Sequence Flexibility—MAX II devices can accommodate any possible power-up sequence
- Power-On Reset—MAX II devices maintain a reset state until voltage is within operating range

This chapter contains the following sections:


- "I/O Standards" on page 8–2
- "MultiVolt Core and I/O Operation" on page 8–3
- "5.0-V Device Compatibility" on page 8–3
- "Recommended Operating Condition for 5.0-V Compatibility" on page 8–7
- "Hot Socketing" on page 8–8
- "Power-Up Sequencing" on page 8–8
- "Power-On Reset" on page 8–8

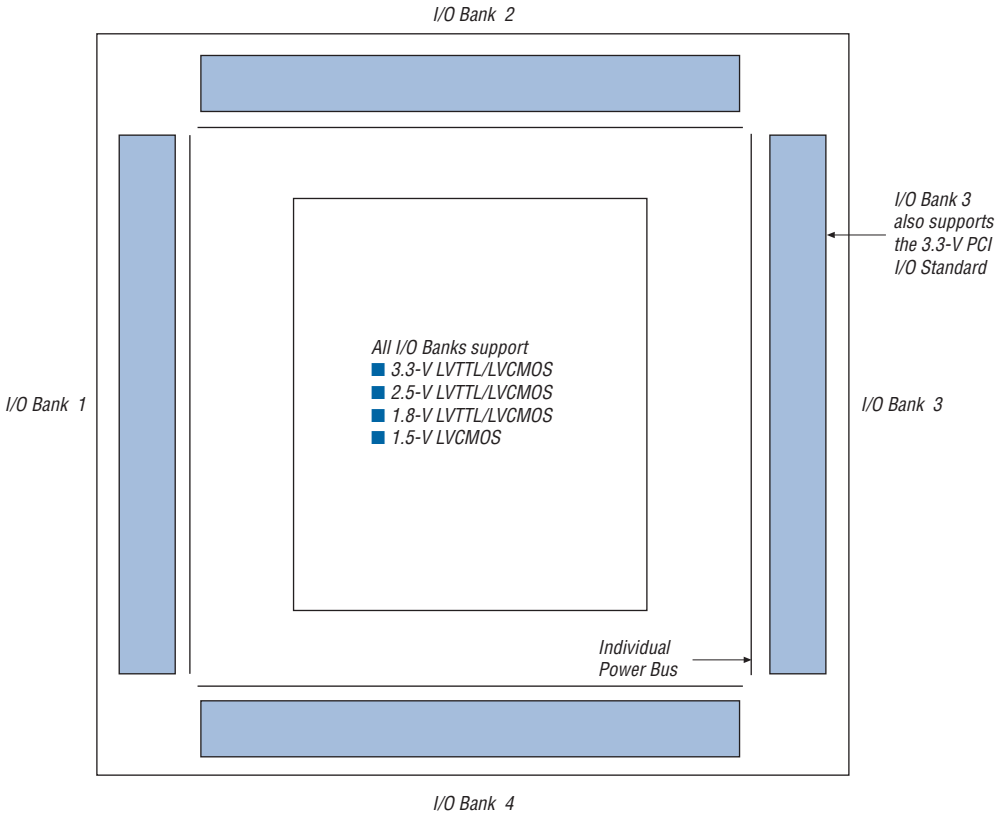
# I/O Standards

The I/O buffer of MAX II devices is programmable and supports a wide range of I/O voltage standards. Each I/O bank in a MAX II device can be programmed to comply with a different I/O standard. All I/O banks can be configured with the following standards:

- 3.3-V LVTTL/LVCMOS
- 2.5-V LVTTL/LVCMOS
- 1.8-V LVTTL/LVCMOS
- 1.5-V LVCMOS

The Schmitt trigger input option is supported by the 3.3-V and 2.5-V I/O standards. The I/O Bank 3 also includes 3.3-V PCI I/O standard interface capability on the EPM1270 and EPM2210 devices. See Figure 8–1.

**Figure 8–1.** I/O Standards Supported by MAX II Device *(Note 1)*, *(2)*, *(3)*, *(4)*, *(5)*



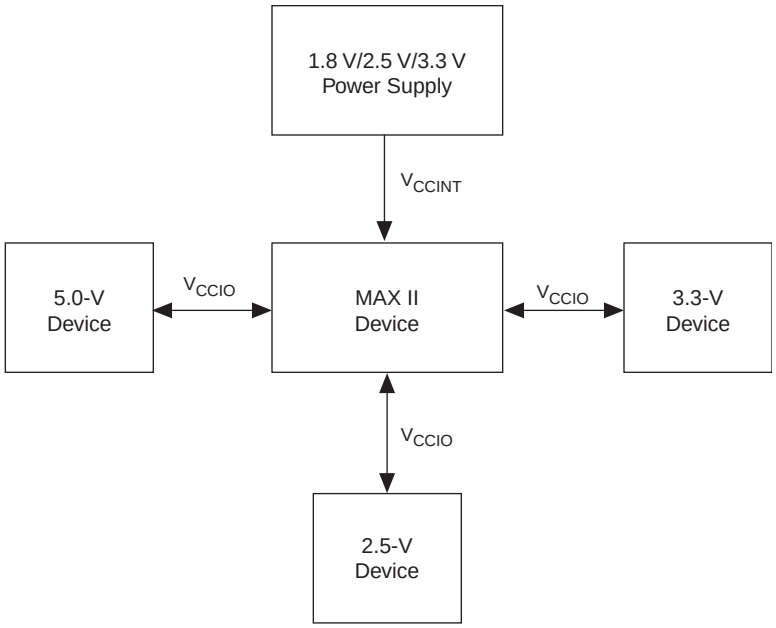

**Notes to Figure 8–1:**

(1) Figure 8–1 is a top view of the silicon die.
(2) Figure 8–1 is a graphical representation only. Refer to the pin list and the Quartus® II software for exact pin locations.
(3) EPM240 and EPM570 devices only have two I/O banks.
(4) The 3.3-V PCI I/O standard is only supported in EPM1270 and EPM2210 devices.
(5) The Schmitt trigger input option for 3.3-V and 2.5-V I/O standards is supported for all I/O pins.

# MultiVolt Core and I/O Operation

MAX II devices include MultiVolt core I/O operation capability, allowing the core and I/O blocks of the device to be powered-up with separate supply voltages. The VCCINT pins supply power to the device core and the VCCIO pins supply power to the device I/O buffers. The VCCINT pins can be powered-up with 1.8 V for MAX IIG and MAX IIZ devices or 2.5/3.3 V for MAX II devices. All the VCCIO pins for a given I/O bank that have MultiVolt capability should be supplied from the same voltage level (for example, 5.0, 3.3, 2.5, 1.8, or 1.5 V). See Figure 8–2.

**Figure 8–2.** Implementing a Multiple-Voltage System with a MAX II Device *(Note 1), (2), (3), (4)*




**Notes to Figure 8–2:**

(1) For MAX IIG and MAX IIZ devices, VCCINT pins will only accept a 1.8-V power supply.

(2) For MAX II devices, VCCINT pins will only accept a 2.5-V or 3.3-V power supply.

(3) MAX II devices can drive a 5.0-V TTL input when $V_{CCIO}$ = 3.3 V. To drive a 5.0-V CMOS, an open-drain setting with internal I/O clamp diode and external resistor are required.

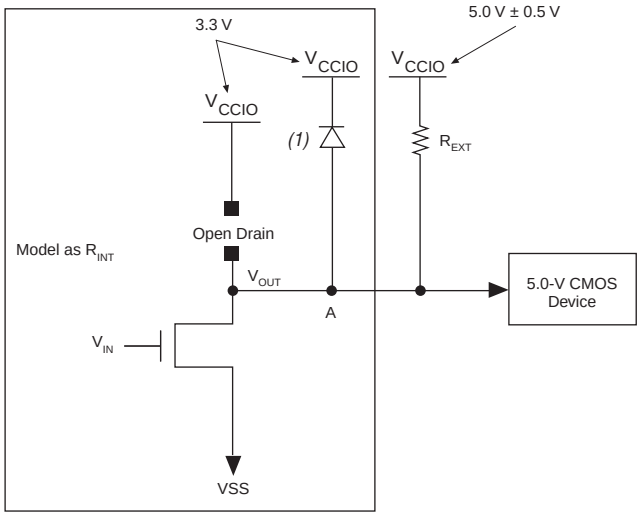
(4) MAX II devices can be 5.0-V tolerant with the use of an external resistor and the internal I/O clamp diode on EPM1270 and EPM2210 devices.

# 5.0-V Device Compatibility

A MAX II device can drive a 5.0-V TTL device by connecting the $V_{CCIO}$ pins of the MAX II device to 3.3 V. This is possible because the output high voltage ($V_{OH}$) of a 3.3-V interface meets the minimum high-level voltage of 2.4 V of a 5.0-V TTL device.

A MAX II device may not correctly interoperate with a 5.0-V CMOS device if the output of the MAX II device is connected directly to the input of the 5.0-V CMOS device. If the MAX II device's $V_{OUT}$ is greater than $V_{CCIO}$, the PMOS pull-up transistor still conducts if the pin is driving high, preventing an external pull-up resistor from pulling the signal to 5.0 V. To make MAX II device outputs compatible with 5.0-V CMOS devices, configure the output pins as open-drain pins with the I/O clamp diode enabled, and use an external pull-up resistor. See Figure 8–3.

**Figure 8–3.** MAX II Device Compatibility with 5.0-V CMOS Devices




**Note to Figure 8–3:**

(1) This diode is only active after power-up. MAX II devices require an external diode if driven by 5.0 V before power-up.

The open-drain pin never drives high, only low or tri-state. When the open-drain pin is active, it drives low. When the open-drain pin is inactive, the pin is tri-stated and the trace pulls up to 5.0 V by the external resistor. The purpose of enabling the I/O clamp diode is to protect the MAX II device's I/O pins. The 3.3-V $V_{CCIO}$ supplied to the I/O clamp diodes causes the voltage at point A to clamp at 4.0 V, which meets the MAX II device's reliability limits when the trace voltage exceeds 4.0 V. The device operates successfully because a 5.0-V input is within its input specification.

The I/O clamp diode is only supported in the EPM1270 and EPM2210 devices' I/O Bank 3. An external protection diode is needed for other I/O banks in EPM1270 and EPM2210 devices and all I/O pins in EPM240 and EPM570 devices.

The pull-up resistor value should be small enough for sufficient signal rise time, but large enough so that it does not violate the $I_{OL}$ (output low) specification of MAX II devices.

The maximum MAX II device $I_{OL}$ depends on the programmable drive strength of the I/O output. Table 8–1 shows the programmable drive strength settings that are available for the 3.3-V LVTTL/LVCMOS I/O standard for MAX II devices. The Quartus II software uses the maximum current strength as the default setting. The PCI I/O standard is always set at 20 mA with no alternate setting.

**Table 8–1.** 3.3-V LVTTL/LVCMOS Programmable Drive Strength

| I/O Standard | $I_{OH}/I_{OL}$ Current Strength Setting (mA) |
|---|---|
| 3.3-V LVTTL | 16 |
| | 8 |
| 3.3-V LVCMOS | 8 |
| | 4 |

To compute the required value of $R_{EXT}$, first calculate the model of the open-drain transistors on the MAX II device. This output resistor ($R_{EXT}$) can be modeled by dividing $V_{OL}$ by $I_{OL}$ ($R_{EXT} = V_{OL}/I_{OL}$). Table 8–2 shows the maximum $V_{OL}$ for the 3.3-V LVTTL/LVCMOS I/O standard for MAX II devices.

For more information about I/O standard specifications, refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*.

**Table 8–2.** 3.3-V LVTTL/LVCMOS Maximum $V_{OL}$

| I/O Standard | Voltage (V) |
|---|---|
| 3.3-V LVTTL | 0.45 |
| 3.3-V LVCMOS | 0.20 |

Select $R_{EXT}$ so that the MAX II device's $I_{OL}$ specification is not violated. You can compute the required pull-up resistor value of $R_{EXT}$ by using the equation: $R_{EXT} = (V_{CC}/I_{OL}) – R_{INT}$. For example, if an I/O pin is configured as a 3.3-V LVTTL with a 16 mA drive strength, given that the maximum power supply ($V_{CC}$) is 5.5 V, the value of $R_{EXT}$ can be calculated as follows:

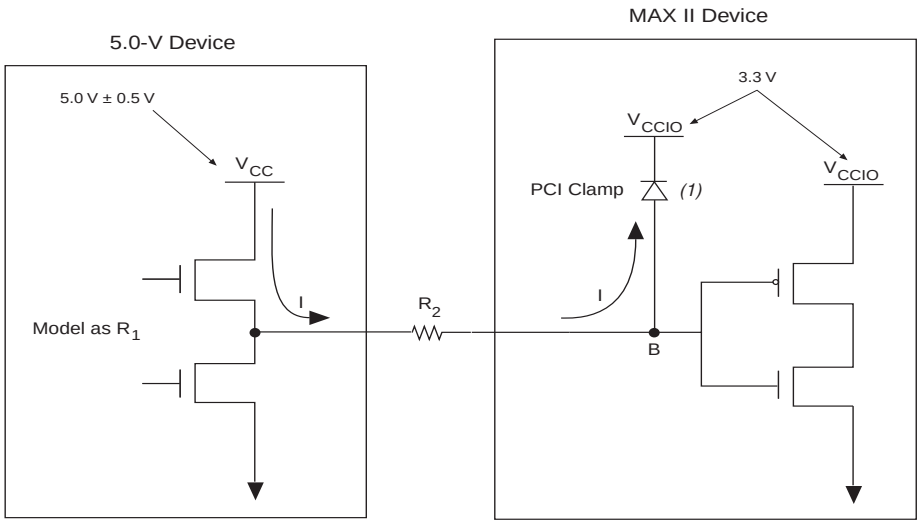
**Equation 8–1.**

$$R_{EXT} = \frac{(5.5\,V - 0.45\,V)}{16\,mA} = 315.6\,\Omega$$

This resistor value computation assumes worst-case conditions. You can adjust the $R_{EXT}$ value according to the device configuration drive strength. Additionally, if your system does not see a wide variation in voltage-supply levels, you can adjust these calculations accordingly.

Because MAX II devices are 3.3-V, 32-bit, 66-MHz PCI compliant, the input circuitry accepts a maximum high-level input voltage ($V_{IH}$) of 4.0 V. To drive a MAX II device with a 5.0-V device, you must connect a resistor ($R_2$) between the MAX II device and the 5.0-V device. See Figure 8–4.

**Figure 8–4.** Driving a MAX II PCI-Compliant Device with a 5.0-V Device




**Note to Figure 8–4:**

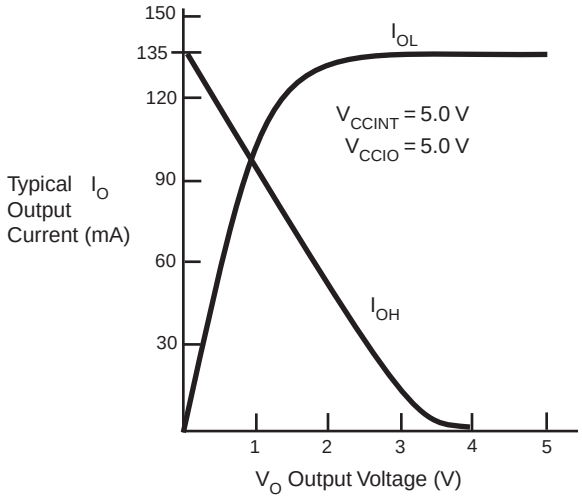
(1) This diode is only active after power-up. MAX II devices require an external diode if driven by 5.0 V before power-up.

If $V_{CCIO}$ for MAX II devices is 3.3 V and the I/O clamp diode is enabled, the voltage at point B in Figure 8–4 is 4.0 V, which meets the MAX II devices reliability limits when the trace voltage exceeds 4.0 V. To limit large current draw from the 5.0-V device, $R_2$ should be small enough for a fast signal rise time and large enough so that it does not violate the high-level output current ($I_{OH}$) specifications of the devices driving the trace.

To compute the required value of $R_2$, first calculate the model of the pull-up transistors on the 5.0-V device. This output resistor ($R_1$) can be modeled by dividing the 5.0-V device supply voltage ($V_{CC}$) by the $I_{OH}$: $R_1 = V_{CC}/I_{OH}$

Figure 8–5 shows an example of typical output drive characteristics of a 5.0-V device.

**Figure 8–5.** Output Drive Characteristics of a 5.0-v Device



As shown above, $R_1$ = 5.0 V/135 mA.

The values usually shown in data sheets reflect typical operating conditions. Subtract 20% from the data sheet value for guard band. This subtraction applied to the above example gives $R_1$ a value of 30.

Select $R_2$ so that the MAX II device's $I_{OH}$ specification is not violated. For example, if the above device has a maximum $I_{OH}$ of 8 mA, given the I/O clamp diode, $V_{IN} = V_{CCIO}$ + 0.7 V = 3.7 V. Given that the maximum supply load of a 5.0-V device ($V_{CC}$) is 5.5 V, the value of $R_2$ can be calculated as follows:

**Equation 8–2.**

$$R_2 = \frac{(5.5\ V - 3.7\ V) - (8\ mA \times 30\ \Omega)}{8\ mA} = 194\ \Omega$$

This analysis assumes worst-case conditions. If your system does not see a wide variation in voltage-supply levels, you can adjust these calculations accordingly.

Because 5.0-V device tolerance in MAX II devices requires use of the I/O clamp, and this clamp is activated only after power-up, 5.0-V signals may not be driven into the device until it is configured. The I/O clamp diode is only supported in the EPM1270 and EPM2210 devices' I/O Bank 3. An external protection diode is needed for other I/O banks for EPM1270 and EPM2210 devices and all I/O pins in EPM240 and EPM570 devices.

# Recommended Operating Condition for 5.0-V Compatibility

As mentioned earlier, a 5.0-V tolerance can be supported with the I/O clamp diode enabled with external series/pull-up resistance. To guarantee long term reliability of the device's I/O buffer, there are restrictions on the signal duty cycle that drive the MAX II I/O, which is based on the maximum clamp current. Table 8–3 shows the maximum signal duty cycle for 3.3-V $V_{CCIO}$ given a PCI clamp current-handling capability.

**Table 8–3.** Maximum Signal Duty Cycle

| $V_{IN}$ (V) *(1)* | $I_{CH}$ (mA) *(2)* | Max Duty Cycle (%) |
|---|---|---|
| 4.0 | 5.00 | 100 |
| 4.1 | 11.67 | 90 |
| 4.2 | 18.33 | 50 |
| 4.3 | 25.00 | 30 |
| 4.4 | 31.67 | 17 |
| 4.5 | 38.33 | 10 |
| 4.6 | 45.00 | 5 |

**Notes to Table 8–3:**

(1) $V_{IN}$ is the voltage at the package pin.

(2) The $I_{CH}$ is calculated with a 3.3-V $V_{CCIO}$. A higher $V_{CCIO}$ value will have a lower $I_{CH}$ value with the same $V_{IN}$.

For signals with duty cycle greater than 30% on MAX II input pins, Altera recommends a $V_{CCIO}$ voltage of 3.0 V to guarantee long-term I/O reliability. For signals with duty cycle less than 30%, the $V_{CCIO}$ voltage can be 3.3 V.

## Hot Socketing

For information about hot socketing, refer to the *Hot Socketing and Power-On Reset in MAX II Devices* chapter in the *MAX II Device Handbook*.

## Power-Up Sequencing

MAX II devices are designed to operate in multiple-voltage environments where it may be difficult to control power sequencing. Therefore, MAX II devices are designed to tolerate any possible power-up sequence. Either $V_{CCINT}$ or $V_{CCIO}$ can initially supply power to the device, and 3.3-V, 2.5-V, 1.8-V, or 1.5-V input signals can drive the devices without special precautions before $V_{CCINT}$ or $V_{CCIO}$ is applied. MAX II devices can operate with a $V_{CCIO}$ voltage level that is higher than the $V_{CCINT}$ level.

When $V_{CCIO}$ and $V_{CCINT}$ are supplied from different power sources to a MAX II device, a delay between $V_{CCIO}$ and $V_{CCINT}$ may occur. Normal operation does not occur until both power supplies are in their recommended operating range. When $V_{CCINT}$ is powered-up, the IEEE Std. 1149.1 Joint Test Action Group (JTAG) circuitry is active. If the `TMS` and `TCK` are connected to $V_{CCIO}$ and $V_{CCIO}$ is not powered-up, the JTAG signals are left floating. Thus, any transition on `TCK` can cause the state machine to transition to an unknown JTAG state, leading to incorrect operation when $V_{CCIO}$ is finally powered-up. To disable the JTAG state during the power-up sequence, `TCK` should be pulled low to ensure that an inadvertent rising edge does not occur on `TCK`.

## Power-On Reset

For information about Power-On Reset (POR), refer to the *Hot Socketing and Power-On Reset in MAX II Devices* chapter in the *MAX II Device Handbook*.

## Conclusion

MAX II devices have MultiVolt I/O support, allowing 1.5-V, 1.8-V, 2.5-V, and 3.3-V devices to interface directly with MAX II devices without causing voltage conflicts. In addition, MAX II devices can interface with 5.0-V devices by slightly modifying the external hardware interface and enabling I/O clamp diodes via the Quartus II software. This MultiVolt capability also enables the device core to run at its core voltage, $V_{CCINT}$, while maintaining I/O pin compatibility with other devices. Altera has taken further steps to make system design easier by designing devices that allow $V_{CCINT}$ and $V_{CCIO}$ to power-up in any sequence and by incorporating support for hot socketing.

# Referenced Documents

This chapter references the following documents:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *Hot Socketing and Power-On Reset in MAX II Devices* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 8–4 shows the revision history for this chapter.

**Table 8–4.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.7 | ■ Updated Figure 8–2.<br>■ Updated "5.0-V Device Compatibility" and "Conclusion"sections.<br>■ Updated New Document Format. | — |
| December 2007, version 1.6 | ■ Updated "Introduction" section.<br>■ "MultiVolt Core and I/O Operation" section.<br>■ Updated (Note 1) to Figure 8–2.<br>■ Added "Referenced Documents" section. | Updated document with MAX IIZ information. |
| December 2006, version 1.5 | ■ Added document revision history. | — |
| August 2006, version 1.4 | ■ Updated "5.0-V Device Compatibility" section. | — |
| February 2006, version 1.3 | ■ Updated Figure 8–3. | — |
| January 2005, version 1.2 | ■ Previously published as Chapter 9. No changes to content. | — |
| December 2004, version 1.1 | ■ Corrected typographical errors in Note 3 of Figure 8–2. | — |

# Section III. User Flash Memory

This section provides information on the user flash memory (UFM) block in MAX® II devices.

This section includes the following chapters:

- Chapter 9, Using User Flash Memory in MAX II Devices
- Chapter 10, Replacing Serial EEPROMs with MAX II User Flash Memory

## Revision History

Refer to each chapter for its own specific revision history. For information on when each chapter was updated, refer to the Chapter Revision Dates section, which appears in the complete handbook.

# 9. Using User Flash Memory in MAX II Devices

MII51010-1.8

## Introduction

MAX® II devices feature a user flash memory (UFM) block that can be used similar to a serial EEPROM for storing non-volatile information up to 8 Kbits. The UFM provides an ideal storage solution supporting any possible protocol for interfacing (SPI, parallel, and other protocols) through bridging logic designed into the MAX II logic array.

This chapter provides guidelines for UFM applications by describing the features and functionality of the MAX II UFM block and the Quartus® II altufm megafunction.

This chapter contains the following sections:


- "UFM Array Description" on page 9–1
- "UFM Functional Description" on page 9–3
- "UFM Operating Modes" on page 9–9
- "Programming and Reading the UFM with JTAG" on page 9–12
- "Software Support for UFM Block" on page 9–13
- "Creating Memory Content File" on page 9–40
- "Simulation Parameters" on page 9–46


## UFM Array Description

Each UFM array is organized as two separate sectors with 4,096 bits per sector. Each sector can be erased independently. Table 9–1 shows the dimension of the UFM array.

**Table 9–1.** UFM Array Size

| Device | Total Bits | Sectors | Address Bits | Data Width |
|---|---|---|---|---|
| EPM240<br>EPM570<br>EPM1270<br>EPM2210 | 8,192 | 2 (4,096 bits per sector) | 9 | 16 |

### Memory Organization Map

Table 9–2 shows the memory organization for the MAX II UFM block. There are 512 locations with 9 bits addressing a range of 000h to 1FFh. Each location stores 16-bit wide data. The most significant bit (MSB) of the address register indicates the sector in operation.

**Table 9–2.** Memory Organization

| Sector | Address Range | |
|---|---|---|
| 1 | 100h | 1FFh |
| 0 | 000h | 0FFh |

## Using and Accessing UFM Storage

You can use the UFM to store data of different memory sizes and data widths. Even though the UFM storage width is 16 bits, you can implement different data widths or a serial interface with the altufm megafunction. Table 9–3 shows the different data widths available for the three types of interfaces supported in the Quartus II software.

**Table 9–3.** Data Widths for Logic Array Interfaces

| Logic Array Interface | Data Width (Bits) | Interface Type |
|---|---|---|
| $I^2C$ | 8 | Serial |
| SPI | 8 or 16 | Serial |
| Parallel | Options of 3 to 16 | Parallel |
| None | 16 | Serial |

For more details about the logic array interface options in the altufm megafunction, refer to "Software Support for UFM Block" on page 9–13.

The UFM block is accessible through the logic array interface as well as the JTAG interface. However, the UFM logic array interface does not have access to the CFM block.

# UFM Functional Description

Figure 9–1 is the block diagram of the MAX II UFM block and the interface signals.

**Figure 9–1.** UFM Block and Interface Signals

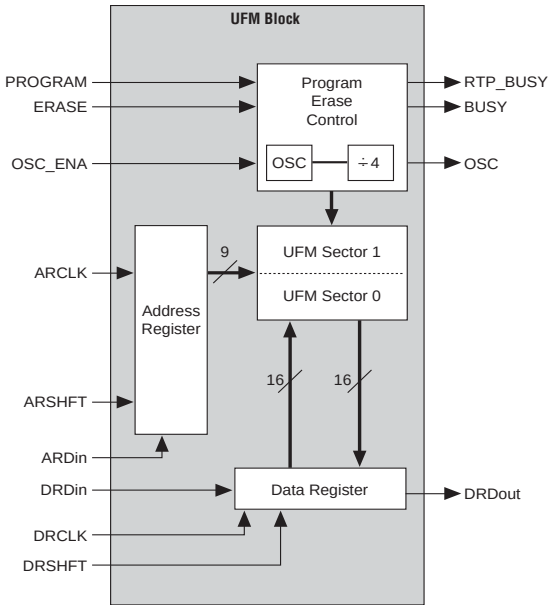



Table 9–4 summarizes the MAX II UFM block input and output interface signals.

**Table 9–4.** UFM Interface Signals (Part 1 of 2)

| Port Name | Port Type | Description |
|---|---|---|
| DRDin | Input | Serial input to the data register. It is used to enter a data word when writing to the UFM. The data register is 16 bits wide and data is shifted serially from the least significant bit (LSB) to the MSB with each DRCLK. This port is required for writing, but unused if the UFM is in read-only mode. |
| DRCLK | Input | Clock input that controls the data register. It is required and takes control when data is shifted from DRDin to DRDout or loaded in parallel from the flash memory. The maximum frequency for DRCLK is 10 MHz. |
| DRSHFT | Input | Signal that determines whether to shift the data register or load it on a DRCLK edge. A high value shifts the data from DRDin into the LSB of the data register and from the MSB of the data register out to DRDout. A low value loads the value of the current address in the flash memory to the data register. |
| ARDin | Input | Serial input to the address register. It is used to enter the address of a memory location to read, program, or erase. The address register is 9 bits wide for the UFM size (8,192 bits). |

**Table 9–4.** UFM Interface Signals (Part 2 of 2)

| Port Name | Port Type | Description |
|---|---|---|
| ARCLK | Input | Clock input that controls the address register. It is required when shifting the address data from ARDin into the address register or during the increment stage. The maximum frequency for ARCLK is 10 MHz. |
| ARSHFT | Input | Signal that determines whether to shift the address register or increment it on an ARCLK edge. A high value shifts the data from ARDin serially into the address register. A low value increments the current address by 1. The address register rolls over to 0 when the address space is at the maximum. |
| PROGRAM | Input | Signal that initiates a program sequence. On the rising edge, the data in the data register is written to the address pointed to by the address register. The BUSY signal asserts until the program sequence is completed. |
| ERASE | Input | Signal that initiates an erase sequence. On a rising edge, the memory sector indicated by the MSB of the address register will be erased. The BUSY signal asserts until the erase sequence is completed. |
| OSC_ENA | Input | This signal turns on the internal oscillator in the UFM block, and is optional but required when the OSC output is used. If OSC_ENA is driven high, the internal oscillator is enabled and the OSC output will toggle. If OSC_ENA is driven low, the internal oscillator is disabled and the OSC output drives constant low. |
| DRDout | Output | Serial output of the data register. Each time the DRCLK signal is applied, a new value is available. The DRDout data depends on the DRSHFT signal. When the DRSHFT signal is high, DRDout value is the new value that is shifted into the MSB of the data register. If the DRSHFT is low, DRDout would contain the MSB of the memory location read into the data register. |
| BUSY | Output | Signal that indicates when the memory is BUSY performing a PROGRAM or ERASE instruction. When it is high, the address and data register should not be clocked. The new PROGRAM or ERASE instruction will not be executed until the BUSY signal is deasserted. |
| OSC | Output | Output of the internal oscillator. It can be used to generate a clock to control user logic with the UFM. It requires an OSC enable input to produce an output. |
| RTP_BUSY | Output | This output signal is optional and only needed if the real-time ISP feature is used. The signal is asserted high during real-time ISP and stays in the RUN_STATE for 500 ms before initiating real-time ISP to allow for the final read/erase/write operation. No read, write, erase, or address and data shift operations are allowed to be issued once the RTP_BUSY signal goes high. The data and address registers do not retain the contents of the last read or write operation for the UFM block during real-time ISP. |

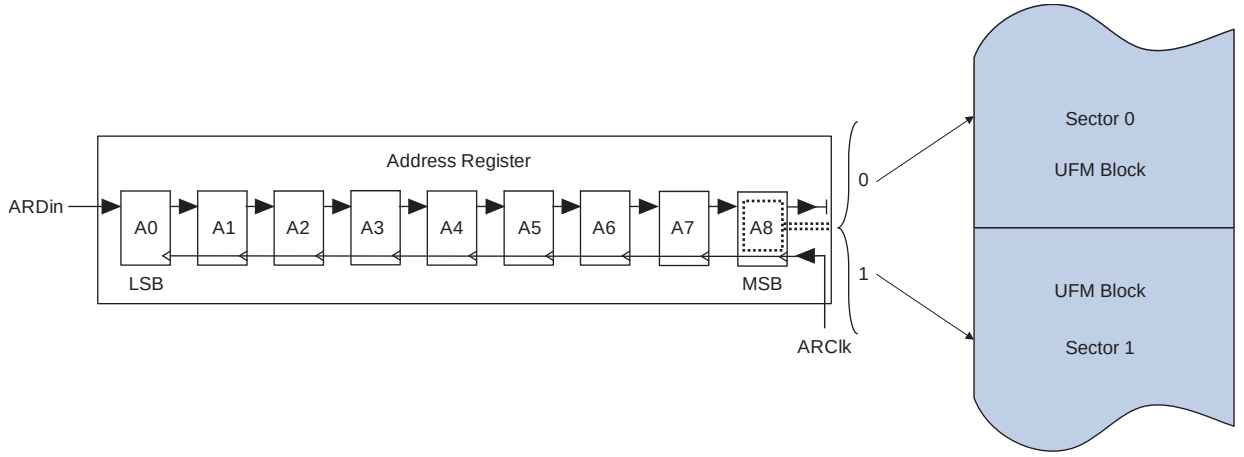
To see the interaction between the UFM block and the logic array of MAX II devices, refer to the *MAX II Architecture* chapter in the *MAX II Device Handbook* (Figure 2–16 for EPM240 devices and Figure 2–17 for EPM570, EPM1270, and EPM2210 devices).

## UFM Address Register

The MAX II UFM block is organized as a 512 × 16 memory. Since the UFM block is organized into two separate sectors, the MSB of the address indicates the sector that will be in action; 0 is for sector 0 (UFM0) while 1 is for sector 1 (UFM1). An ERASE instruction erases the content of the specific sector that is indicated by the MSB of the address register. Figure 9–2 shows the selection of the UFM sector in action using the MSB of the address register.

Refer to "Erase" on page 9–11 for more information about ERASE mode.

**Figure 9–2.** Selection of the UFM Sector Using the MSB of the Address Register

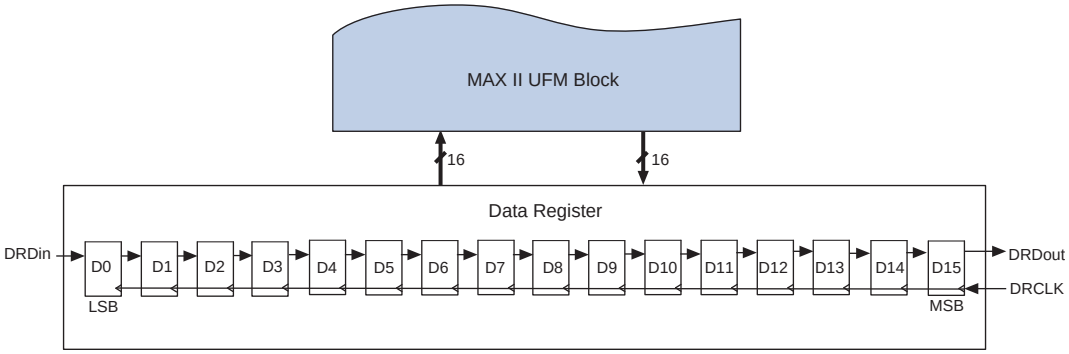



Three control signals exist for the address register: ARSHFT, ARCLK, and ARDin. ARSHFT is used as both a shift-enable control signal and an auto-increment signal. If the ARSHFT signal is high, a rising edge on ARCLK will load address data serially from the ARDin port and move data serially through the register. A clock edge with the ARSHFT signal low increments the address register by 1. This implements an auto-increment of the address to allow data streaming. When a program, read, or an erase sequence is executing, the address that is in the address register becomes the active UFM location.

## UFM Data Register

The UFM data register is 16 bits wide with four control signals: DRSHFT, DRCLK, DRDin, and DRDout. DRSHFT distinguishes between clock edges that move data serially from DRDin or to DRDout and clock edges that latch parallel data from the UFM sectors. If the DRSHFT signal is high, a clock edge moves data serially through the registers from DRDin to DRDout. If the DRSHFT signal is low, a clock edge captures data from the UFM sector pointed by the address register in parallel. The MSB is the first bit that will be seen at DRDout. The data register DRSHFT signal will also be used to enable the UFM for reading data. When the DRSHFT signal is low, the UFM latches data into the data register. Figure 9–3 shows the UFM data register.

**Figure 9–3.** UFM Data Register




## UFM Program/Erase Control Block

The UFM program/erase control block is used to generate all the control signals necessary to program and erase the UFM block independently. This reduces the number of LEs necessary to implement a UFM controller in the logic array. It also guarantees correct timing of the control signals to the UFM. A rising edge on either PROGRAM or ERASE causes this control signal block to activate and begin sequencing through the program or erase cycle. At this point, for a program instruction, whatever data is in the data register will be written to the address pointed to by the address register.

Only sector erase is supported by the UFM. Once an ERASE command is executed, this control block will erase the sector whose address is stored in the address register. When the PROGRAM or ERASE command first activates the program/erase control block, the BUSY signal will be driven high to indicate an operation in progress in the UFM. Once the program or erase algorithm is completed, the BUSY signal will be forced low.

## Oscillator

OSC_ENA, one of the input signals in the UFM block, is used to enable the oscillator signal to output through the OSC output port. You can use this OSC output port to connect with the interface logic in the logic array. It can be routed through the logic array and fed back as an input clock for the address register (ARCLK) and the data register (DRCLK). The output frequency of the OSC port is one-fourth that of the oscillator frequency. As a result, the frequency range of the OSC port is 3.3 to 5.5 MHz. The maximum clock frequency accepted by ARCLK and DRCLK is 10 MHz and the duty cycle accepted by the DRCLK and ARCLK input ports is approximately 45% to 50%.

When the OSC_ENA input signal is asserted, the oscillator is enabled and the output is routed to the logic array through the OSC output. When the OSC_ENA is set low, the OSC output drives constant low. The routing delay from the OSC port of the UFM block to OSC output pin depends on placement. You can analyze this delay using the Quartus II timing analyzer.

The undivided internal oscillator, which is not accessible, operates in a frequency range from 13.33 to 22.22 MHz. The internal oscillator is enabled during power-up, in-system programming, and real-time ISP. At all other times, the oscillator is not running unless the UFM is instantiated in the design and the OSC_ENA port is asserted. To see how specific operating modes of ALTUFM handle OSC_ENA and the oscillator, refer to "Software Support for UFM Block" on page 9–13. For user generated logic interfacing to the UFM, the oscillator must be enabled during PROGRAM or ERASE operations, but not during READ operations. OSC_ENA can be tied low if you are not issuing any PROGRAM or ERASE commands.

During real-time ISP operation, the internal oscillator automatically enables and outputs through the OSC output port (if this port is instantiated) even though the OSC_ENA signal is tied low. You can use the RTP_BUSY signal to detect the beginning and ending of the real-time ISP operation for gated control of this self-enabled OSC output condition.

The internal oscillator is not enabled all the time. The internal oscillator for the program/erase operation is only activated when the flash memory block is being programmed or erased. During the READ operation, the internal oscillator is activated whenever the flash memory block is reading data.

### Instantiating the Oscillator without the UFM

You can use the IO/MAX II oscillator megafunction selection in the MegaWizard® Plug-In Manager to instantiate the UFM oscillator if you intend to use this signal without using the UFM memory block. Figure 9–4 shows the altufm_osc megafunction instantiation in the Quartus II software.

**Figure 9–4.** The Quartus II altufm_osc Megafunction

This megafunction is in the I/O folder on page 2a of the MegaWizard® Plug-In Manager, as shown in Figure 9–5. You can start the MegaWizard Plug-In Manager on the Tools menu.

**Figure 9–5.** Selecting the altufm_osc Megafunction in the MegaWizard Plug-In Manager

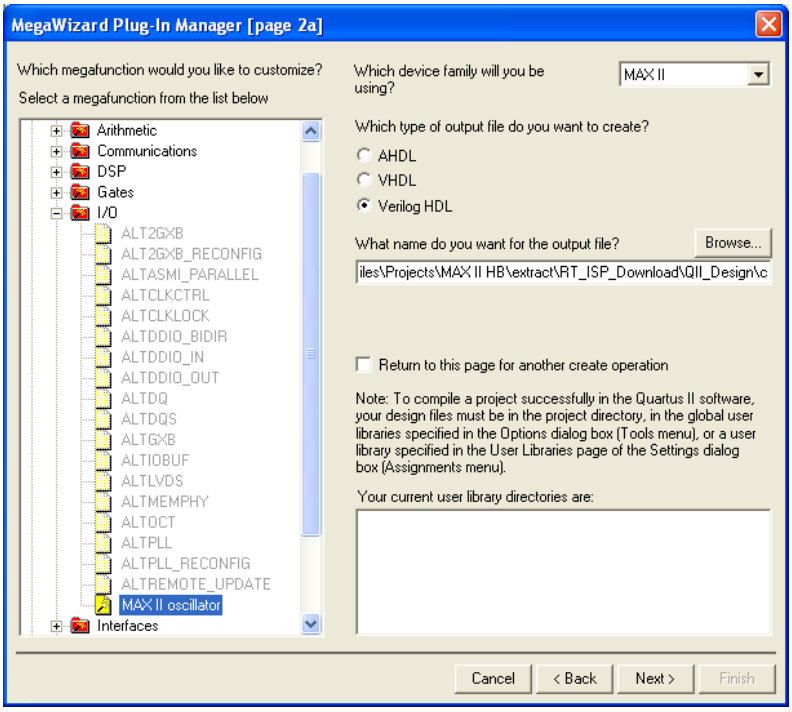
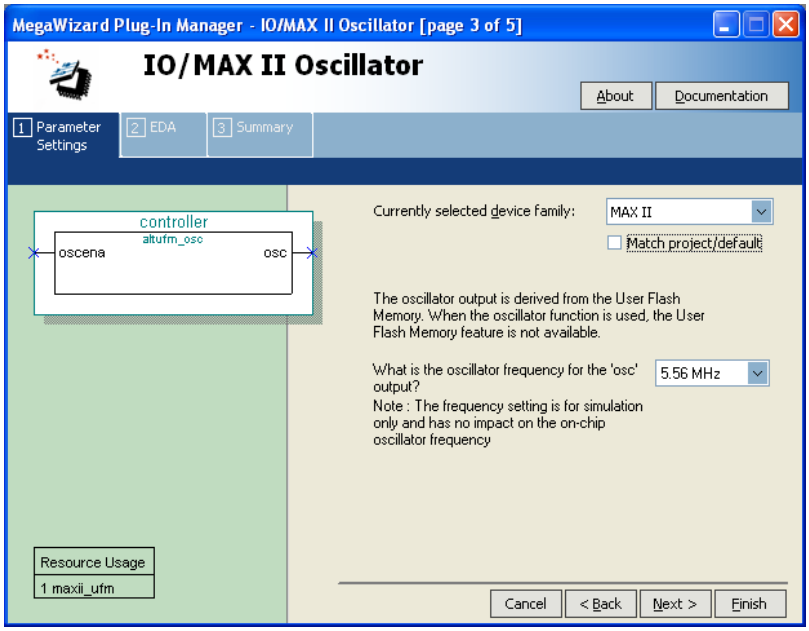



Figure 9–6 shows page 3 of the IO/MAX II oscillator megafunction. You have an option to choose to simulate the OSC output port at its maximum or minimum frequency during the design simulation. The frequency chosen is only used as a timing parameter simulation and does not affect the real MAX II device OSC output frequency.

**Figure 9–6.** Page 3 of the OSC Megafunction MegaWizard Plug-In Manager



# UFM Operating Modes

There are three different modes for the UFM block:

- Read/Stream Read
- Program (Write)
- Erase

During program, address and data can be loaded concurrently. You can manipulate the UFM interface controls as necessary to implement the specific protocol provided the UFM timing specifications are met. Figure 9–7 through Figure 9–10 show the control waveforms for accessing UFM in three different modes. For PROGRAM mode (Figure 9–9) and ERASE mode (Figure 9–10), the PROGRAM and ERASE signals are not obligated to assert immediately after loading the address and data. They can be asserted anytime after the address register and data register have been loaded. Do not assert the READ, PROGRAM, and ERASE signals or shift data and address into the UFM after entering the real-time ISP mode. You can use the RTP_BUSY signal to detect the beginning and end of real-time ISP operation and generate control logic to stop all UFM port operations. This user-generated control logic is only necessary for the altufm_none megafunction, which provides no auto-generated logic. The other interfaces for the altufm megafunction (altufm_parallel, altufm_spi, altufm_i2c) contain control logic to automatically monitor the RTP_BUSY signal and will cease operations to the UFM when a real-time ISP operation is in progress.

You can program the UFM and CFM blocks independently without overwriting the other block which is not programmed. The Quartus II programmer provides the options to program the UFM and CFM blocks individually or together (the entire MAX II Device).

Refer to the *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook* for guidelines about using ISP and real-time ISP while utilizing the UFM block within your design.

Refer to the *MAX II Architecture* chapter in the *MAX II Device Handbook* for a complete description of the device architecture, and for the specific values of the timing parameters listed in this chapter.

## Read/Stream Read

The three control signals, PROGRAM, ERASE, and BUSY are not required during read or stream read operation. To perform a read operation, the address register has to be loaded with the reference address where the data is or is going to be located in the UFM. The address register can be stopped from incrementing or shifting addresses from ARDin by stopping the ARCLK clock pulse. DRSHFT must be asserted low at the next rising edge of DRCLK to load the data from the UFM to the data register. To shift the bits from the register, 16 clock pulses have to be provided to read 16-bit wide data. You can use DRCLK to control the read time or disable the data register by discontinuing the DRCLK clock pulse. Figure 9–7 shows the UFM control waveforms during read mode.

The UFM block can also perform stream read operation, reading continuously from the UFM using the address increment feature. Stream read mode is started by loading the base address into the address register. DRSHFT must then be asserted low at the first rising edge of DRCLK to load data into the data register from the address pointed to by the address register. DRSHFT will then assert high to shift out the 16-bit wide data with the MSB out first. Figure 9–8 shows the UFM control waveforms during stream read mode.

**Figure 9–7.** UFM Read Waveforms

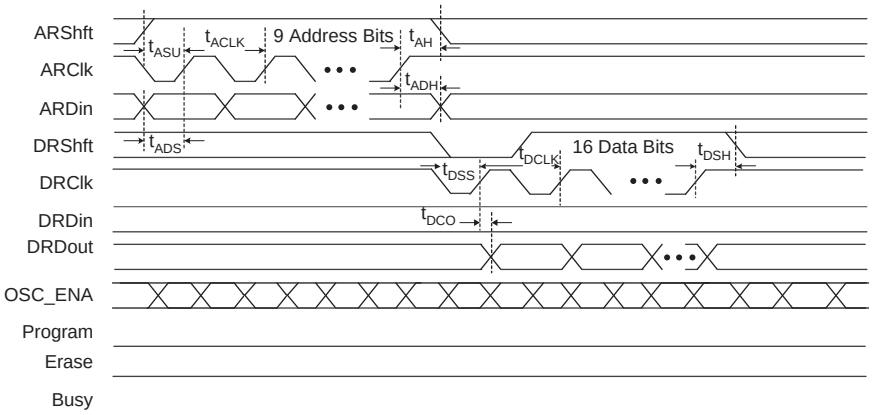
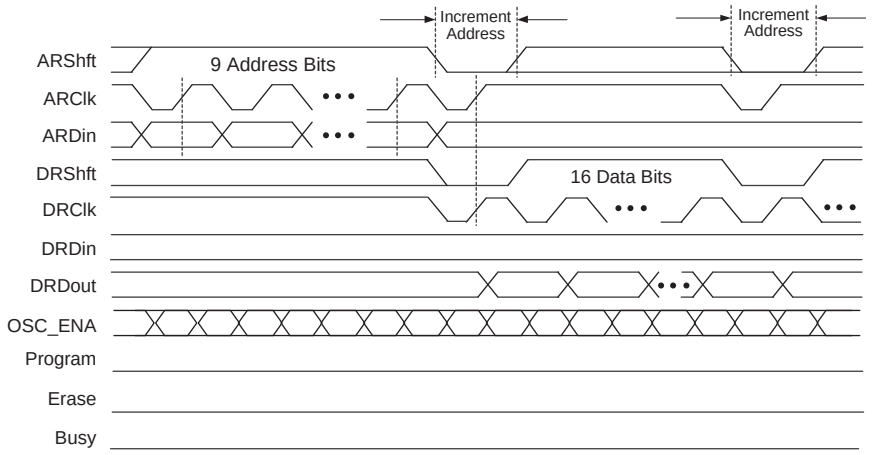



**Figure 9–8.** UFM Stream Read Waveforms

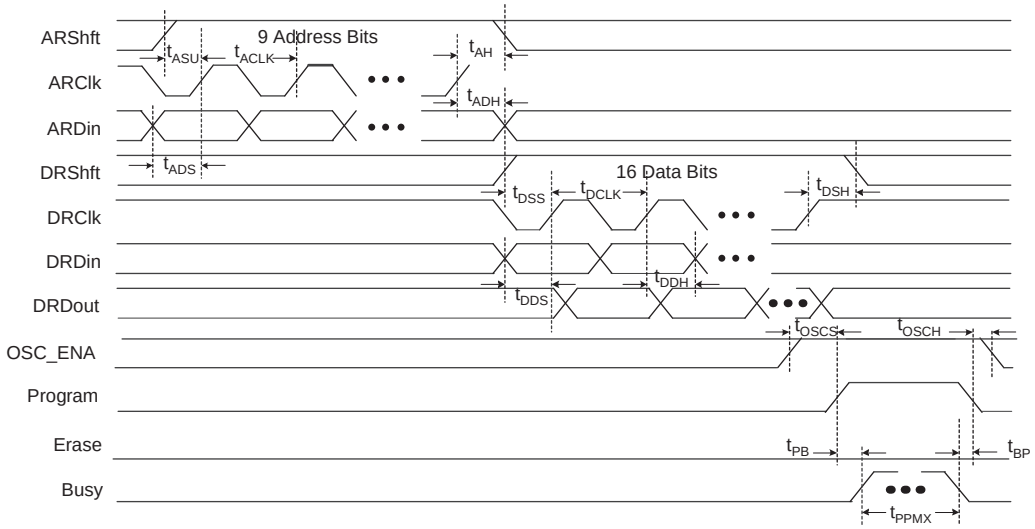



## Program

To program or write to the UFM, you must first perform a sequence to load the reference address into the address register. DRSHFT must then be asserted high to load the data serially into the data register starting with the MSB. Loading an address into the address register and loading data into the data register can be done concurrently. After the 16 bits of data have been successfully shifted into the data register, the PROGRAM signal must be asserted high to start writing to the UFM. On the rising edge, the data currently in the data register is written to the location currently in the address register. The BUSY signal is asserted until the program sequence is completed. The

data and address register should not be modified until the BUSY signal is de-asserted, or the flash content will be corrupted. The PROGRAM signal is ignored if the BUSY signal is asserted. When the PROGRAM signal is applied at exactly the same time as the ERASE signal, the behavior is undefined and the contents of flash is corrupted. Figure 9–9 shows the UFM waveforms during program mode.

**Figure 9–9.** UFM Program Waveforms

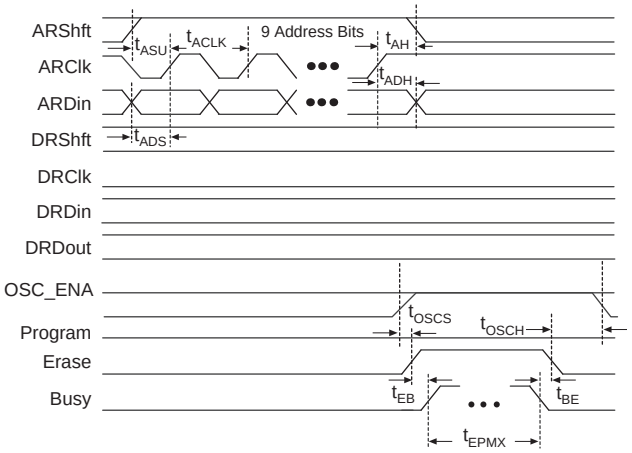



## Erase

The ERASE signal initiates an erase sequence to erase one sector of the UFM. The data register is not needed to perform an erase sequence. To indicate the sector of the UFM to be erased, the MSB of the address register should be loaded with 0 to erase the UFM sector 0, or 1 to erase the UFM sector 1 (Figure 9–2 on page 9–5). On a rising edge of the ERASE signal, the memory sector indicated by the MSB of the address register will be erased. The BUSY signal is asserted until the erase sequence is completed. The address register should not be modified until the BUSY signal is de-asserted to prevent the content of the flash from being corrupted. This ERASE signal will be ignored when the BUSY signal is asserted. Figure 9–10 illustrates the UFM waveforms during erase mode.

When the UFM sector is erased, it has 16-bit locations all filled with FFFF. Each UFM storage bit can be programmed no more than once between erase sequences. You can write to any word up to two times as long as the second programming attempt at that location only adds 0s. 1s are mask bits for your input word that cannot overwrite 0s in the flash array. New 1s in the location can only be achieved by an erase. Therefore, it is possible for you to perform byte writes since the UFM array is 16 bits for each location.

**Figure 9–10.** UFM Erase Waveforms




# Programming and Reading the UFM with JTAG

In Altera MAX II devices, you can write or read data to/from the UFM using the IEEE Std. 1149.1 JTAG interface. You can use a PC or UNIX workstation, the Quartus II Programmer, and the ByteBlaster™ MV or ByteBlaster™ II parallel port download cable to download Programmer Object File (**.pof**), Jam™ Standard Test and Programming Language (STAPL) Files (**.jam**), or Jam Byte-Code Files (**.jbc**) from the Quartus II software targeting the MAX II device UFM block.

The POF, Jam File, or JBC File can be generated using the Quartus II software.

## Jam Files

Both Jam STAPL and JBC files support programming for the UFM block.

## Jam Players

Jam Players read the descriptive information in Jam files and translate them into data that programs the target device. Jam Players do not program a particular device architecture or vendor; they only read and understand the syntax defined by the Jam file specification. In-field changes are confined to the Jam file, not the Jam Player. As a result, you do not need to modify the Jam Player source code for each in-field upgrade.

There are two types of Jam Players to accommodate the two types of Jam files: an ASCII Jam STAPL Player and a Jam STAPL Byte-Code Player. Both ASCII Jam STAPL Player and Jam STAPL Byte-Code Player are coded in the C programming language for 16-bit and 32-bit processors.

For guidelines on UFM operation during ISP, refer to the *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*.

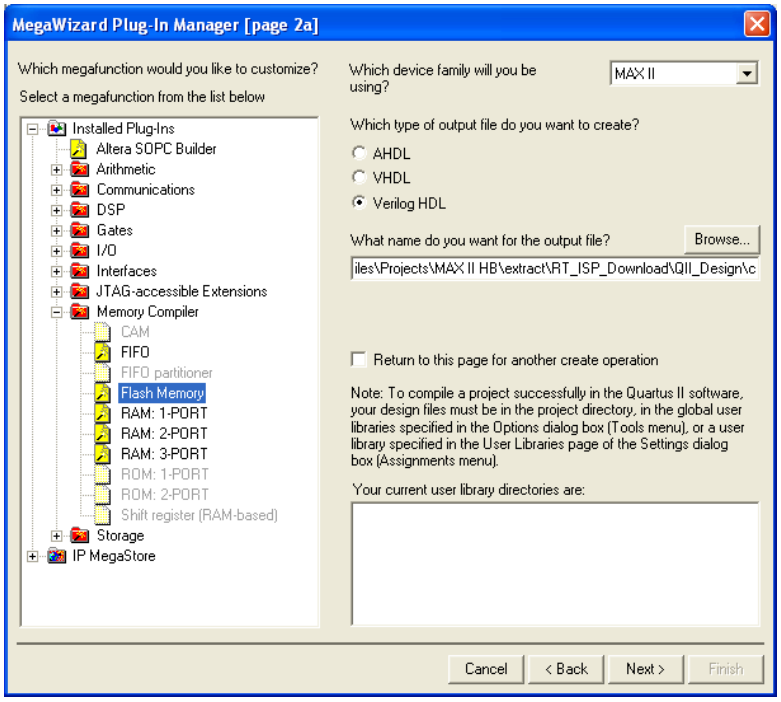
# Software Support for UFM Block

The Altera Quartus II software includes sophisticated tools that fully utilize the advantages of UFM block in MAX II device, while maintaining simple, easy-to-use procedures that accelerate the design process. The following section describes how the altufm megafunction supports a simple design methodology for instantiating standard interface protocols for the UFM block, such as:

- I²C
- SPI
- Parallel
- None (Altera Serial Interface)

This section includes the megafunction symbol, the input and output ports, a description of the MegaWizard Plug-In Manager options, and example MegaWizard screen shots. Refer to Quartus II Help for the altufm megafunction AHDL functional prototypes (applicable to Verilog HDL), VHDL component declaration, and parameter descriptions. Figure 9–11 shows altufm megafunction selection (Flash Memory) in the MegaWizard Plug-In Manager. This megafunction is in the **memory compiler** directory on page 2a of the MegaWizard Plug-In Manager. You can start the MegaWizard Plug-In Manager on the Tools menu.

**Figure 9–11.** altufm Megafunction Selection in the MegaWizard Plug-In Manager




The altufm MegaWizard Plug-In Manager has separate pages that apply to the MAX II UFM block. During compilation, the Quartus II Compiler verifies the altufm parameters selected against the available logic array interface options, and any specific assignments.

## Inter-Integrated Circuit

Inter-Integrated Circuit (I²C) is a bidirectional two-wire interface protocol, requiring only two bus lines; a serial data/address line (SDA), and a serial clock line (SCL). Each device connected to the I²C bus is software addressable by a unique address. The I²C bus is a multi-master bus where more than one integrated circuit (IC) capable of initiating a data transfer can be connected to it, which allows masters to function as transmitters or receivers.

The altufm_i2c megafunction features a serial, 8-bit bidirectional data transfer up to 100 Kbits per second. With the altufm_i2c megafunction, the MAX II UFM and logic can be configured as a slave device for the I²C bus. The altufm megafunction's I²C interface is designed to function similar to I²C serial EEPROMs.

The Quartus II software supports three different memory sizes:

- (128 × 8) 1 Kbits
- (256 × 8) 2 Kbits
- (512 × 8) 4 Kbits
- (1,024 × 8) 8 Kbits

### I²C Protocol

The following defines the characteristics of the I²C bus protocol:

- Only two bus lines are required: SDA and SCL. Both SDA and SCL are bidirectional lines which remain high when the bus is free.
- Data transfer can be initiated only when the bus is free.
- The data on the SDA line must be stable during the high period of the clock. The high or low state of the data line can only change when the clock signal on the SCL line is low.
- Any transition on the SDA line while the SCL is high is one such unique case which indicates a start or stop condition.

Table 9–5 summarizes the altufm_i2c megafunction input and output interface signals.

**Table 9–5.** altufm_i2c Interface Signals

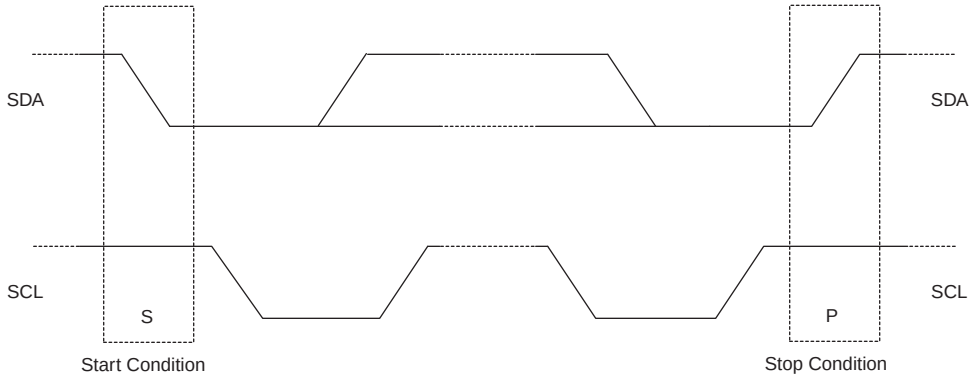
| Pin | Description | Function |
|---|---|---|
| SDA | Serial Data/Address Line | The bidirectional SDA port is used to transmit and receive serial data from the UFM. The output stage of the SDA port is configured as an open drain pin to perform the wired-AND function. |
| SCL | Serial Clock Line | The bidirectional SCL port is used to synchronize the serial data transfer to and from the UFM. The output stage of the SCL port is configured as an open drain pin to perform a wired-AND function. |
| WP | Write Protect | Optional active high signal that disables the erase and write function for read/write mode. The altufm_i2c megafunction gives you an option to protect the entire UFM memory or only the upper half of memory. |
| $A_2$, $A_1$, $A_0$ | Slave Address Input | These inputs set the UFM slave address. The $A_6$, $A_5$, $A_4$, $A_3$ slave address bits are programmable, set internally to 1010 by default. |

### START and STOP Condition

The master always generates start (S) and stop (P) conditions. After the start condition, the bus is considered busy. Only a stop (P) condition frees the bus. The bus stays busy if the repeated start (Sr) condition is executed instead of a stop condition. In this occurrence, the start (S) and repeated start (Sr) conditions are functionally identical.

A high-to-low transition on the SDA line while the SCL is high indicates a start condition. A low-to-high transition on the SDA line while the SCL is high indicates a stop condition. Figure 9–12 shows the start and stop conditions.

**Figure 9–12.** Start and Stop Conditions

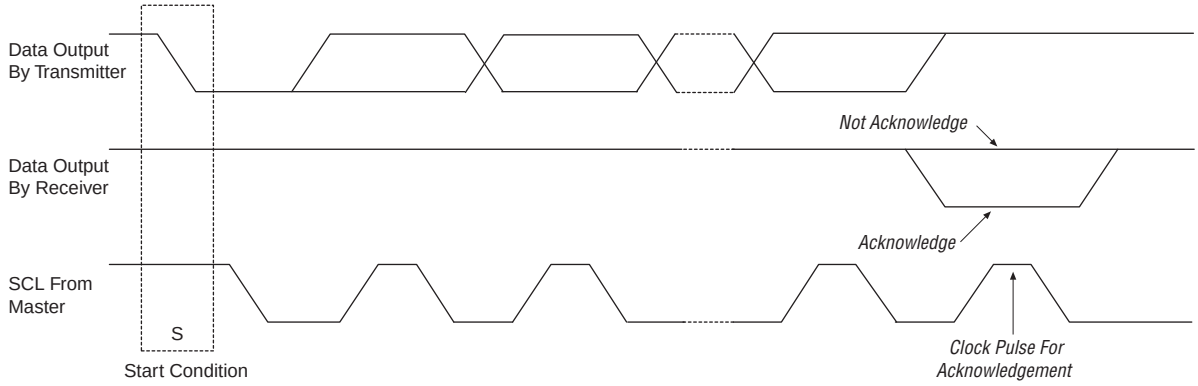



### Acknowledge

Acknowledged data transfer is a requirement of I²C. The master must generate a clock pulse to signify the acknowledge bit. The transmitter releases the SDA line (high) during the acknowledge clock pulse.

The receiver (slave) must pull the SDA line low during the acknowledge clock pulse so that SDA remains a stable low during the clock high period, indicating positive acknowledgement from the receiver. If the receiver pulls the SDA line high during the acknowledge clock pulse, the receiver sends a not-acknowledge condition indicating that it is unable to process the last byte of data. If the receiver is busy (for example, executing an internally-timed erase or write operation), it will not acknowledge any new data transfer. Figure 9–13 shows the acknowledge condition on the I²C bus.

**Figure 9–13.** Acknowledge on the I²C Bus

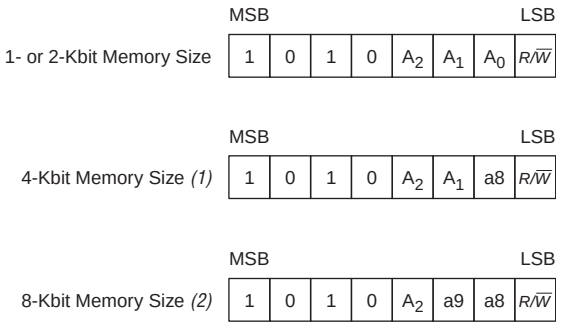


## Device Addressing

After the start condition, the master sends the address of the particular slave device it is requesting. The four most significant bits (MSBs) of the 8-bit slave address are usually fixed while the next three significant bits ($A_2$, $A_1$, $A_0$) are device address bits and define which device the master is accessing. The last bit of the slave address specifies whether a read or write operation is to be performed. When this bit is set to 1, a read operation is selected. When this bit is set to 0, a write operation is selected.

The four MSBs of the slave address ($A_6$, $A_5$, $A_4$, $A_3$) are programmable and can be defined on page 3 of the altufm MegaWizard Plug-In Manager. The default value for these four MSBs is `1010`. The next three significant bits are defined using the three $A_2$, $A_1$, $A_0$ input ports of the altufm_i2c megafunction. You can connect these ports to input pins in the design file and connect them to switches on the board. The other option is to connect them to $V_{CC}$ and GND primitives in the design file, which conserves pins. Figure 9–14 shows the slave address bits.

**Figure 9–14.** Slave Address Bits




**Notes to Figure 9–14:**

(1) For the 4-Kbit memory size, the $A_0$ location in the slave address becomes the MSB (a8) of the memory byte address.

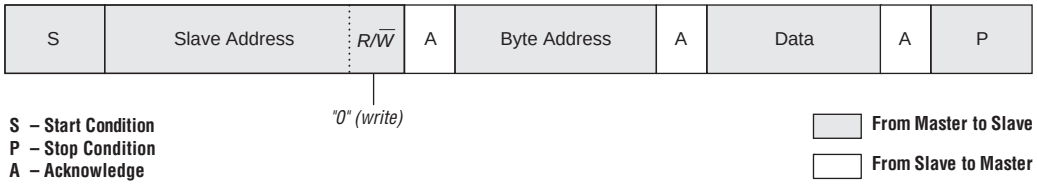
(2) For the 8-Kbit memory size, the $A_0$ location in the slave address becomes a8 of the memory byte address, while the $A_1$ location in the slave address becomes the MSB (a9) of the memory byte address.

After the master sends a start condition and the slave address byte, the altufm_i2c logic monitors the bus and responds with an acknowledge (on the SDA line) when its address matches the transmitted slave address. The altufm_i2c megafunction then performs a read or write operation to/from the UFM, depending on the state of the bit.

## Byte Write Operation

The master initiates a transfer by generating a start condition, then sending the correct slave address (with the R/$\overline{W}$ bit set to 0) to the slave. If the slave address matches, the altufm_i2c slave acknowledges on the ninth clock pulse. The master then transfers an 8-bit byte address to the UFM, which acknowledges the reception of the address. The master transfers the 8-bit data to be written to the UFM. Once the altufm_i2c logic acknowledges the reception of the 8-bit data, the master generates a stop condition. The internal write from the MAX II logic array to the UFM begins only after the master generates a stop condition. While the UFM internal write cycle is in progress, the altufm_i2c logic ignores any attempt made by the master to initiate a new transfer. Figure 9–15 shows the Byte Write sequence.

**Figure 9–15.** Byte Write Sequence




## Page Write Operation

Page write operation has a similar sequence as the byte write operation, except that a number of bytes of data are transmitted in sequence before the master issues a stop condition. The internal write from the MAX II logic array to the UFM begins only after the master generates a stop condition. While the UFM internal write cycle is in progress, the altufm_i2c logic ignores any attempt made by the master to initiate a new transfer. The altufm_i2c megafunction allows you to choose the page size of 8 bytes, 16 bytes, or 32 bytes for the page write operation, as shown in Figure 9–24 on page 9–24.

A write operation is only possible on an erased UFM block or word location. The UFM block differs from serial EEPROMs, requiring an erase operation prior to writing new data in the UFM block. A special erase sequence is required, as discussed in "Erase Operation" on page 9–18.

### Acknowledge Polling

The master can detect whether the internal write cycle is completed by polling for an acknowledgement from the slave. The master can resend the start condition together with the slave address as soon as the byte write sequence is finished. The slave does not acknowledge if the internal write cycle is still in progress. The master can repeat the acknowledge polling and can proceed with the next instruction after the slave acknowledges.

## Write Protection

The altufm_i2c megafunction includes an optional Write Protection (WP) port available on page 4 of the `altufm` MegaWizard Plug-In Manager (see Figure 9–24 on page 9–24). In the MegaWizard Plug-In Manager, you can choose the WP port to protect either the full or upper half memory.

When WP is set to 1, the upper half or the entire memory array (depending on the write protection level selected) is protected, and the write and erase operation is not allowed. In this case the altufm_i2c megafunction acknowledges the slave address and memory address. After the master transfers the first data byte, the altufm_i2c megafunction sends a not-acknowledge condition to the master to indicate that the instruction will not execute. When WP is set to `0`, the write and erase operations are allowed.

## Erase Operation

Commercial serial EEPROMs automatically erase each byte of memory before writing into that particular memory location during a write operation. However, the MAX II UFM block is flash based and only supports sector erase operations and not byte erase operations. When using read/write mode, a sector or full memory erase operation is required before writing new data into any location that previously contained data. The block cannot be erased when the altufm_i2c megafunction is in read-only mode.

Data can be initialized into memory for read/write and read-only modes by including a memory initialization file (**.mif**) or hexidecimal file (**.hex**) in the altufm MegaWizard Plug-In Manager. This data is automatically written into the UFM during device programming by the Quartus II software or third-party programming tool.

The altufm_i2c megafunction supports four different erase operation methods shown on page 4 of the altufm MegaWizard Plug-In Manager:

- Full Erase (Device Slave Address Triggered)
- Sector Erase (Byte Address Triggered)
- Sector Erase ($A_2$ Triggered)
- No Erase

These erase options only work as described if that particular option is selected in the MegaWizard Plug-In Manager before compiling the design files and programming the device. Only one option is possible for the altufm_i2c megafunction.

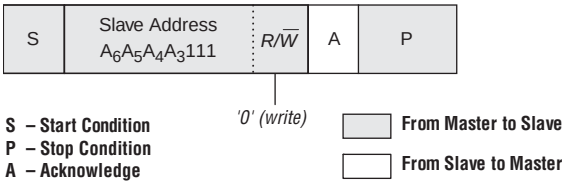
Erase options are discussed in more detail in the following sections.

### Full Erase (Device Slave Address Erase)

The full erase option uses the $A_2$, $A_1$, $A_0$ bits of the slave address to distinguish between an erase or read/write operation. This slave operation decoding occurs when the master transfers the slave address to the slave after generating the start condition. If the $A_2$, $A_1$, and $A_0$ slave address bits transmitted to the UFM slave equals 111 and the four remaining MSBs match the rest of the slave addresses, then the Full Erase operation is selected. If the $A_6$, $A_5$, $A_4$, $A_3$ $A_2$, $A_1$, and $A_0$ slave address bits transmitted to the UFM match its unique slave address setting, the read/write operation is selected and functions as expected. As a result, this erase option utilizes two slave addresses on the bus reserving $A_6$, $A_5$, $A_4$, $A_3$, 1, 1, 1 as the erase trigger. Both sectors of the UFM block will be erased when the Full Erase operation is executed. This operation requires acknowledge polling. The internal UFM erase function only begins after the master generates a stop condition. Figure 9–16 shows the full erase sequence triggered by using the slave address.

If the memory is write-protected (WP = 1), the slave does not acknowledge the erase trigger slave address ($A_6$, $A_5$, $A_4$, $A_3$, 1, 1, 1) sent by the master. The master should then send a stop condition to terminate the transfer. The full erase operation will not be executed.

**Figure 9–16.** Full Erase Sequence Triggered Using the Slave Address




### Sector Erase (Byte Address Triggered)

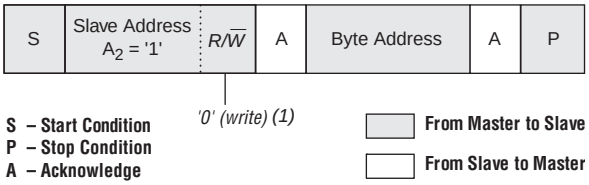
This sector erase operation is triggered by defining a 7- to 10-bit byte address for each sector depending on the memory size. The trigger address for each sector is entered on page 4 of the altufm MegaWizard Plug-In Manager, as shown in Figure 9–24 on page 9–24. When a write operation is executed targeting this special byte address location, the UFM sector that contains that byte address location is erased. This sector erase operation is automatically followed by a write of the intended write byte to that address. The default byte address location for UFM Sector 0 erase is address 0x00. The default byte address location for UFM Sector 1 erase is [(selected memory size)/2]. You can specify another byte location as the trigger-erase addresses for each sector.

This sector erase operation supports up to eight UFM blocks or serial EEPROMs on the I²C bus. This sector erase operation requires acknowledge polling.

### Sector Erase ($A_2$ Triggered)

This sector erase operation uses the received $A_2$ slave address bit to distinguish between an erase or read/write operation. This slave operation decoding occurs when the master transmits the slave address after generating the start condition. If the $A_2$ bit received by the UFM slave is 1, the sector erase operation is selected. If the $A_2$ bit received is 0, the read/write operation is selected. While this reserves the $A_2$ bit as an erase or read/write operation bit, the $A_0$ and $A_1$ bits still act as slave address bits to address the UFM. With this erase option, there can be up to four UFM slaves cascaded on the bus for 1-Kbit and 2-Kbit memory sizes. Only two UFM slaves can be cascaded on the bus for 4-Kbit memory size, since $A_0$ of the slave address becomes the ninth bit (MSB) of the byte address. After the slave acknowledges the slave address and its erase or read/write operation bit, the master can transfer any byte address within the sector that must be erased. The internal UFM sector erase operation only begins after the master generates a stop condition. Figure 9–17 shows the sector erase sequence using the $A_2$ bit of the slave address.

**Figure 9–17.** Sector Erase Sequence Indicated Using the $A_2$ Bit of the Slave Address




**Note to Figure 9–17:**

(1) $A_2$ = 0 indicates a read/write operation is executed in place of an erase. In this case, the R/W bit determines whether it is a read or write operation.

If the altufm_i2c megafunction is write-protected (WP=1), the slave does not acknowledge the byte address (which indicates the UFM sector to be erased) sent in by the master. The master should then send a stop condition to terminate the transfer and the sector erase operation will not be executed.

### No Erase

The no erase operation never erases the UFM contents. This method is recommended when UFM does not require constant re-writing after its initial write of data. For example, if the UFM data is to be initialized with data during manufacturing using I²C, you may not require writing to the UFM again. In that case, you should use the no erase option and save logic element (LE) resources from being used to create erase logic.

## Read Operation

The read operation is initiated in the same manner as the write operation except that the R/W bit must be set to 1. Three different read operations are supported:

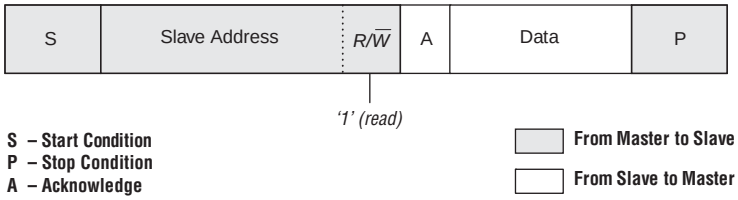
- Current Address Read (Single Byte)
- Random Address Read (Single byte)
- Sequential Read (Multi-Byte)

After each UFM data has been read and transferred to the master, the UFM address register is incremented for all single and multi-byte read operations.

### Current Address Read

This read operation targets the current byte location pointed to by the UFM address register. Figure 9–18 shows the current address read sequence.

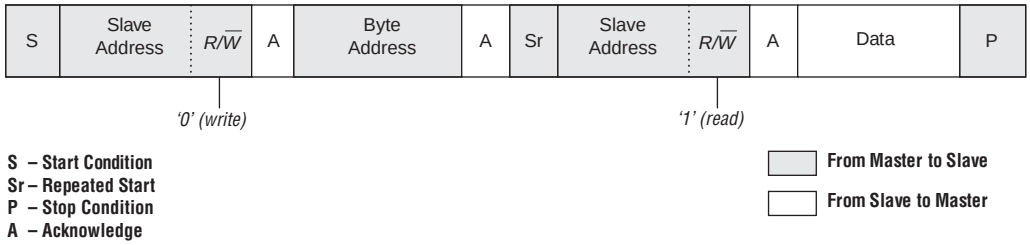
**Figure 9–18.** Current Address Read Sequence




### Random Address Read

Random address read operation allows the master to select any byte location for a read operation. The master first performs a "dummy" write operation by sending the start condition, slave address, and byte address of the location it wishes to read. After the altufm_i2c megafunction acknowledges the slave and byte address, the master generates a repeated start condition, the slave address, and the R/$\overline{W}$ bit is set to 1. The altufm_i2c megafunction then responds with acknowledge and sends the 8-bit data requested. The master then generates a stop condition. Figure 9–19 shows the random address read sequence.

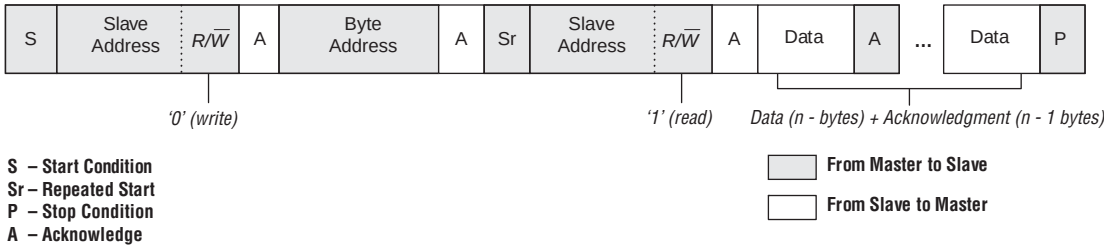
**Figure 9–19.** Random Address Read Sequence




### Sequential Read

Sequential read operation can be initiated by either the current address read operation or the random address read operation. Instead of sending a stop condition after the Slave has transmitted one byte of data to the master, the master acknowledges that byte and sends additional clock pulses (on SCL line) for the slave to transmit data bytes from consecutive byte addresses. The operation is terminated when the master generates a stop condition instead of responding with an acknowledge. Figure 9–20 shows the sequential read sequence.

**Figure 9–20.** Sequential Read Sequence



## ALTUFM I²C Interface Timing Specification

Figure 9–21 shows the timing waveform for the altufm_i2c megafunction read/write mode.

**Figure 9–21.** Timing Waveform for the altufm_i2c Megafunction

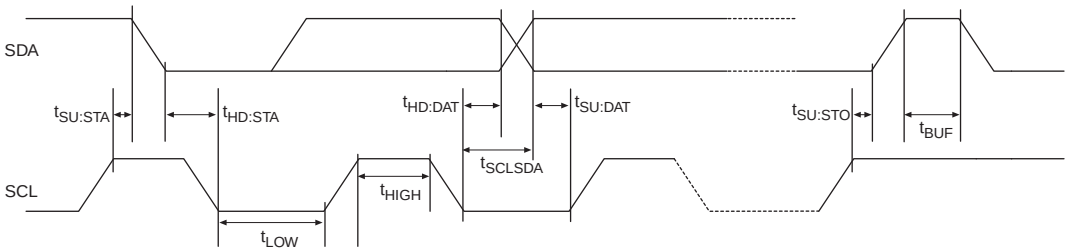



Table 9–6 through Table 9–8 list the timing specification needed for the altufm_i2c megafunction read/write mode.

**Table 9–6.** I2C Interface Timing Specification

| Symbol | Parameter | Min | Max | Unit |
|---|---|---|---|---|
| $F_{SCL}$ | SCL clock frequency | — | 100 | kHz |
| $t_{SCL:SDA}$ | SCL going low to SDA data out | — | 15 | ns |
| $t_{BUF}$ | Bus free time between a stop and start condition | 4.7 | — | µs |
| $t_{HD:STA}$ | (Repeated) start condition hold time | 4 | — | µs |
| $t_{SU:STA}$ | (Repeated) start condition setup time | 4.7 | — | µs |
| $t_{LOW}$ | SCL clock low period | 4.7 | — | µs |
| $t_{HIGH}$ | SCL clock high period | 4 | — | µs |
| $t_{HD:DAT}$ | SDA data in hold time | 0 | — | ns |
| $t_{SU:DAT}$ | SDA data in setup time | 20 | — | ns |
| $t_{SU:STO}$ | STOP condition setup time | 4 | — | ns |

**Table 9–7.** UFM Write Cycle Time

| Parameter | Min | Max | Unit |
|---|---|---|---|
| Write Cycle Time | — | 110 | µs |

**Table 9–8.** UFM Erase Cycle Time

| Parameter | Min | Max | Unit |
|---|---|---|---|
| Sector Erase Cycle Time | — | 501 | ms |
| Full Erase Cycle Time | — | 1,002 | ms |

## Instantiating the I2C Interface Using the Quartus II altufm Megafunction

Figure 9–22 shows the altufm megafunction symbol for a I²C interface instantiation in the Quartus II software.

**Figure 9–22.** altufm Megafunction Symbol For the I²C Interface Instantiation in the Quartus II Software

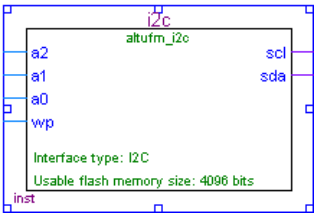
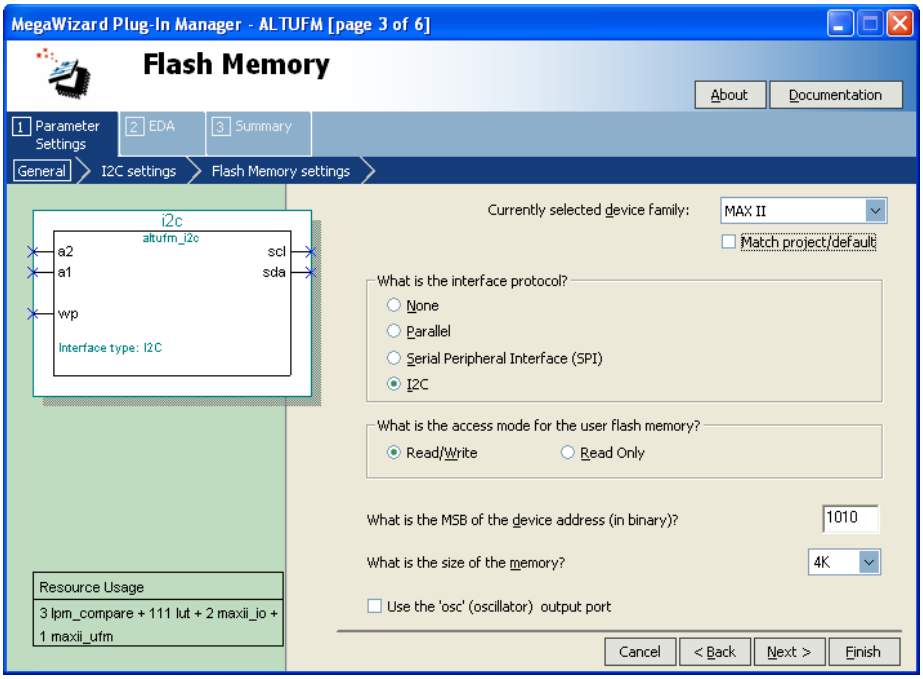



Figure 9–23 shows page 3 of the altufm MegaWizard Plug-In Manager when selecting I²C as the interface. On this page, you can choose whether to implement the read/write mode or read-only mode for the UFM. You also have an option to choose the memory size for the altufm_i2c megafunction as well as defining the four MSBs of the slave address (default 1010).

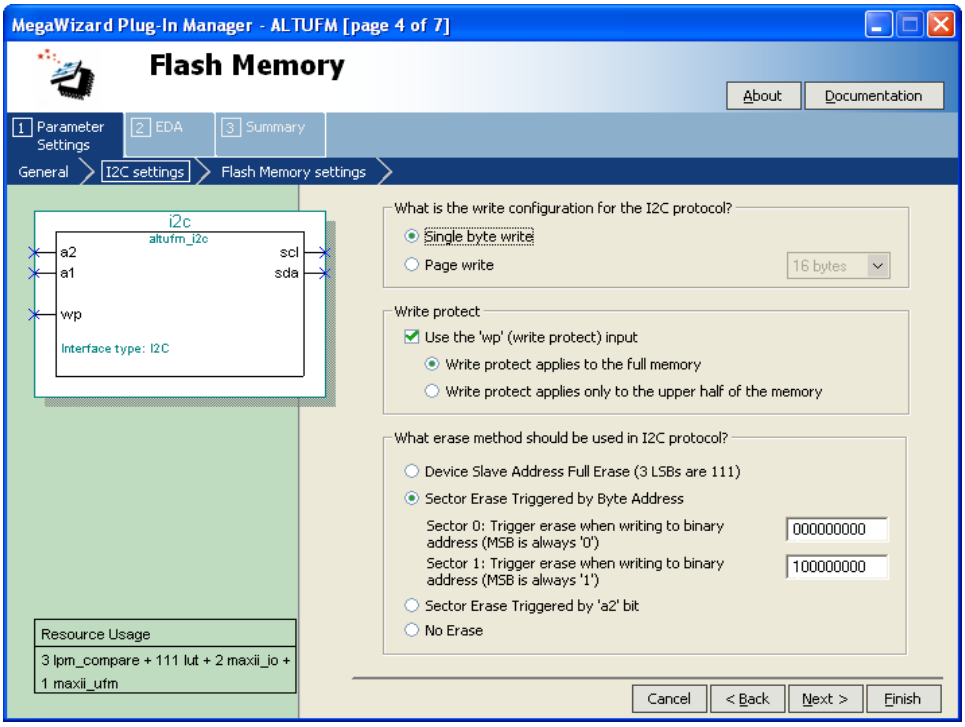
**Figure 9–23.** Page 3 of the altufm MegaWizard Plug-In Manager (I²C)




The UFM block's internal oscillator is always running when the altufm_i2c megafunction is instantiated for both read-only and read/write interfaces.

Figure 9–24 shows page 4 of the altufm MegaWizard Plug-In Manager. You can select the optional write protection and erase operation methods on this page.

**Figure 9–24.** Page 4 of the altufm MegaWizard Plug-In Manager ($I^2C$)




## Serial Peripheral Interface

Serial peripheral interface (SPI) is a four-pin serial communication subsystem included on the Motorola 6805 and 68HC11 series microcontrollers. It allows the microcontroller unit to communicate with peripheral devices, and is also capable of inter-processor communications in a multiple-master system.

The SPI bus consists of masters and slaves. The master device initiates and controls the data transfers and provides the clock signal for synchronization. The slave device responds to the data transfer request from the master device. The master device in an SPI bus initiates a service request with the slave devices responding to the service request.

With the altufm megafunction, the UFM and MAX II logic can be configured as a slave device for the SPI bus. The `OSC_ENA` is always asserted to enable the internal oscillator when the SPI megafunction is instantiated for both read only and read/write interfaces.

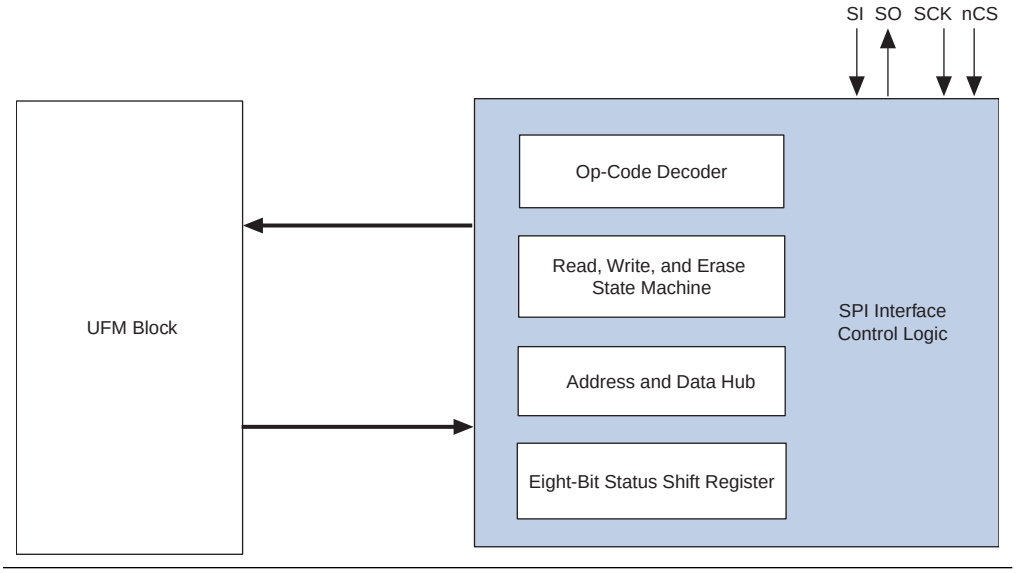
The Quartus II software supports both the Base mode (which uses 8-bit address and data) and the Extended mode (which uses 16-bit address and data). Base mode uses only UFM sector 0 (2,048 bits), whereas Extended mode uses both UFM sector 0 and sector 1 (8,192 bits). There are only four pins in SPI: `SI`, `SO`, `SCK`, and `nCS`. Table 9–9 describes the SPI pins and functions.

**Table 9–9.** SPI Interface Signals

| Pin | Description | Function |
|---|---|---|
| SI | Serial Data Input | Receive data serially. |
| SO | Serial Data Output | Transmit data serially. |
| SCK | Serial Data Clock | The clock signal produced from the master device to synchronize the data transfer. |
| nCS | Chip Select | Active low signal that enables the slave device to receive or transfer data from the master device. |

Data transmitted to the SI port of the slave device is sampled by the slave device at the positive SCK clock. Data transmits from the slave device through SO at the negative SCK clock edge. When nCS is asserted, it means the current device is being selected by the master device from the other end of the SPI bus for service. When nCS is not asserted, the SI and SCK ports should be blocked from receiving signals from the master device, and SO should be in High Impedance state to avoid causing contention on the shared SPI bus. All instructions, addresses, and data are transferred with the MSB first and start with high-to-low nCS transition. The circuit diagram is shown in Figure 9–25.

**Figure 9–25.** Circuit Diagram for SPI Interface Read or Write Operations




## Opcodes

The 8-bit instruction opcode is shown in Table 9–10. After nCS is pulled low, the indicated opcode must be provided. Otherwise, the interface assumes that the master device has internal logic errors and ignores the rest of the incoming signals. Once nCS is pulled back to high, the interface is back to normal. nCS should be pulled low again for a new service request.

**Table 9–10.** Instruction Set for SPI

| Name | Opcode | Operation |
|---|---|---|
| WREN | 00000110 | Enable Write to UFM |
| WRDI | 00000100 | Disable Write to UFM |
| RDSR | 00000101 | Read Status Register |
| WRSR | 00000001 | Write Status Register |
| READ | 00000011 | Read data from UFM |
| WRITE | 00000010 | Write data to UFM |
| SECTOR-ERASE | 00100000 | Sector erase |
| UFM-ERASE | 01100000 | Erase the entire UFM block (both sectors) |

The READ and WRITE opcodes are instructions for transmission, which means the data will be read from or written to the UFM.

WREN, WRDI, RDSR, and WRSR are instructions for the status register, where they do not have any direct interaction with UFM, but read or set the status register within the interface logic. The status register provides status on whether the UFM block is available for any READ or WRITE operation, whether the interface is WRITE enabled, and the state of the UFM WRITE protection. The status register format is shown in Table 9–11. For the read only implementation of ALTUFM SPI (Base or Extended mode), the status register does not exist, saving LE resources.

**Table 9–11.** Status Register Format

| Position | Status | Default at Power-Up | Description |
|---|---|---|---|
| Bit 7 | X | 0 | — |
| Bit 6 | X | 0 | — |
| Bit 5 | X | 0 | — |
| Bit 4 | X | 0 | — |
| Bit 3 | BP1 | 0 | Indicate the current level of block write protection *(1)* |
| Bit 2 | BP0 | 0 | Indicate the current level of block write protection *(1)* |
| Bit 1 | WEN | 0 | 1= SPI WRITE enabled state<br>0= SPI WRITE disabled state |
| Bit 0 | nRDY | 0 | 1 = Busy, UFM WRITE or ERASE cycle in progress<br>0 = No UFM WRITE or ERASE cycle in progress |

**Note to Table 9–11:**

(1) Refer to Table 9–12 and Table 9–13 for more information about status register bits BPI and BPO.

The following paragraphs describe the instructions for SPI.

### READ

READ is the instruction for data transmission, where the data is read from the UFM block. When data transfer is taking place, the MSB is always the first bit to be transmitted or received. The data output stream is continuous through all addresses until it is terminated by a low-to-high transition at the nCS port. The READ operation is always performed through the following sequence in SPI, as shown in Figure 9–26:

1. nCS is pulled low to indicate the start of transmission.
2. An 8-bit READ opcode (00000011) is received from the master device. (If internal programming is in progress, READ is ignored and not accepted).
3. A 16-bit address is received from the master device. The LSB of the address is received last. As the UFM block can take only nine bits of address maximum, the first seven address bits received are discarded.
4. Data is transmitted for as many words as needed by the slave device through SO for READ operation. When the end of the UFM storage array is reached, the address counter rolls over to the start of the UFM to continue the READ operation.
5. nCS is pulled back to high to indicate the end of transmission.

For SPI Base mode, the READ operation is always performed through the following sequence in SPI:

1. nCS is pulled low to indicate the start of transmission.
2. An 8-bit READ opcode (00000011) is received from the master device, followed by an 8-bit address. If internal programming is in progress, the READ operation is ignored and not accepted.
3. Data is transmitted for as many words as needed by the slave device through SO for READ operation. The internal address pointer automatically increments until the highest memory address is reached (address 255 only since the UFM sector 0 is used). The address counter will not roll over once address 255 is reached. The SO output is set to high-impedance (Z) once all the eight data bits from address 255 has been shifted out through the SO port.
4. nCS is pulled back to high to indicate the end of transmission.

**Figure 9–26.** READ Operation Sequence for Extended Mode

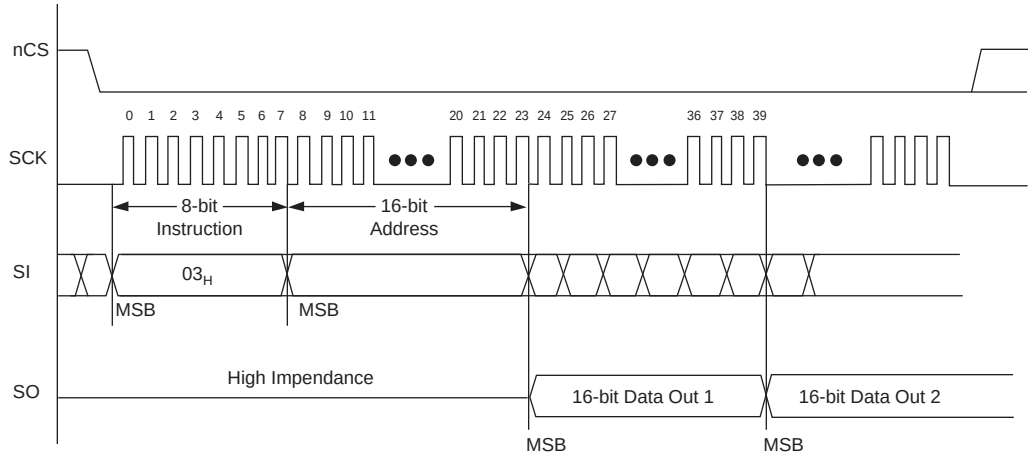
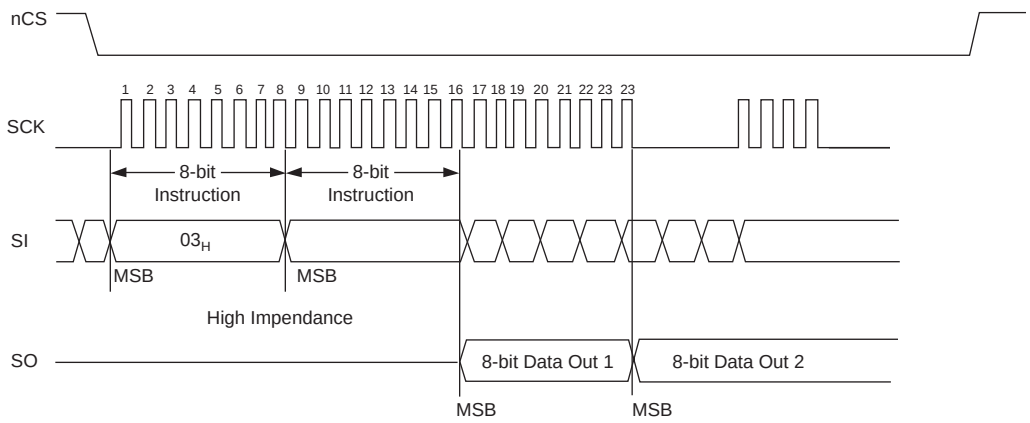


Figure 9–27 shows the READ operation sequence for Base mode.

**Figure 9–27.** READ Operation for Base Mode




### WRITE

WRITE is the instruction for data transmission, where the data is written to the UFM block. The targeted location in the UFM block that will be written must be in the erased state ($FFFF_H$) before initiating a WRITE operation. When data transfer is taking place, the MSB is always the first bit to be transmitted or received. nCS must be driven high before the instruction is executed internally. You may poll the nRDY bit in the software status register for the completion of the internal self-timed WRITE cycle. For SPI Extended mode, the WRITE operation is always done through the following sequence, as shown in Figure 9–28:

1. nCS is pulled low to indicate the start of transmission.
2. An 8-bit WRITE opcode (00000010) is received from the master device. If internal programming is in progress, the WRITE operation is ignored and not accepted.
3. A 16-bit address is received from the master device. The LSB of the address will be received last. As the UFM block can take only nine bits of address maximum, the first seven address bits received are discarded.
4. A check is carried out on the status register (see Table 9–11) to determine if the WRITE operation has been enabled, and the address is outside of the protected region; otherwise, Step 5 is bypassed.
5. One word (16 bits) of data is transmitted to the slave device through SI.
6. nCS is pulled back to high to indicate the end of transmission.

For SPI Base mode, the WRITE operation is always performed through the following sequence in SPI:

1. nCS is pulled low to indicate the start of transmission.
2. An 8-bit WRITE opcode (00000010) is received. If the internal programming is in progress, the WRITE operation is ignored and not accepted.
3. An 8-bit address is received. A check is carried out on the status register (see Table 9–11) to determine if the WRITE operation has been enabled, and the address is outside of the protected region; otherwise, Step 4 is skipped.

4. An 8-bit data is transmitted through SI.
5. nCS is pulled back to high to indicate the end of transmission.

**Figure 9–28.** WRITE Operation Sequence for Extended Mode

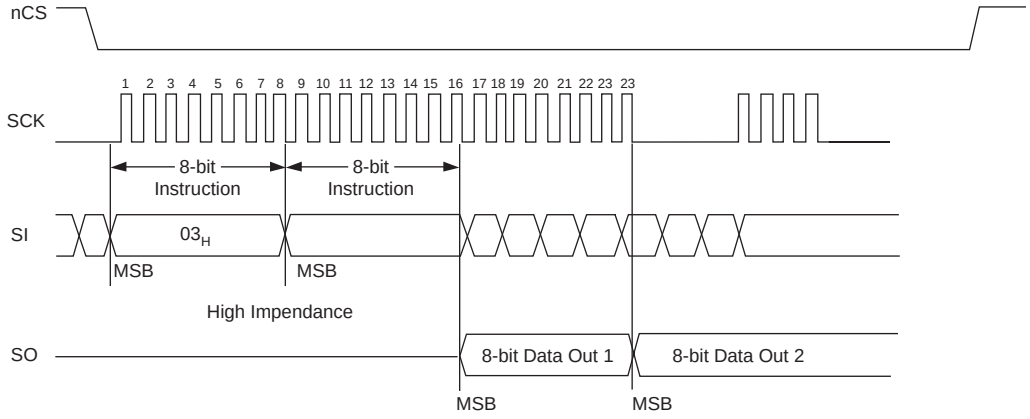
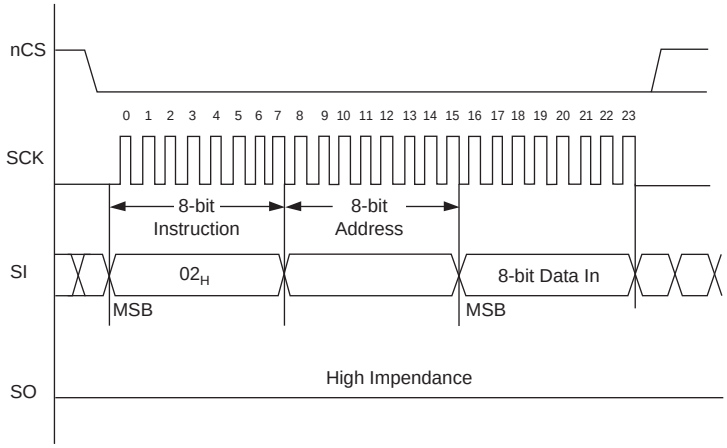



Figure 9–29 shows the WRITE operation sequence for Base mode.

**Figure 9–29.** WRITE Operation Sequence for Base Mode




### SECTOR-ERASE

SECTOR-ERASE is the instruction of erasing one sector of the UFM block. Each sector contains 256 words. WEN bit and the sector must not be protected for SE operation to be successful. nCS must be driven high before the instruction is executed internally. You may poll the nRDY bit in the software status register for the completion of the internal self-timed SECTOR-ERASE cycle. For SPI Extended mode, the SE operation is performed in the following sequence, as shown in Figure 9–30:

1. nCS is pulled low.
2. Opcode 00100000 is transmitted into the interface.
3. The 16-bit address is sent. The eighth bit (the first seven bits will be discarded) of the address indicates which sector is erased; a 0 means sector 0 (UFM0) is erased, and a 1 means sector 1 (UFM1) is erased.

4. nCS is pulled back to high.

For SPI Base mode, the SE instruction erases UFM sector 0. As there are no choices of UFM sectors to be erased, there is no address component to this instruction. The SE operation is always done through the following sequence in SPI Base mode:

1. nCS is pulled low.
2. Opcode `00100000` is transmitted into the interface.
3. nCS is pulled back to high.

**Figure 9–30.** SECTOR-ERASE Operation Sequence for Extended Mode

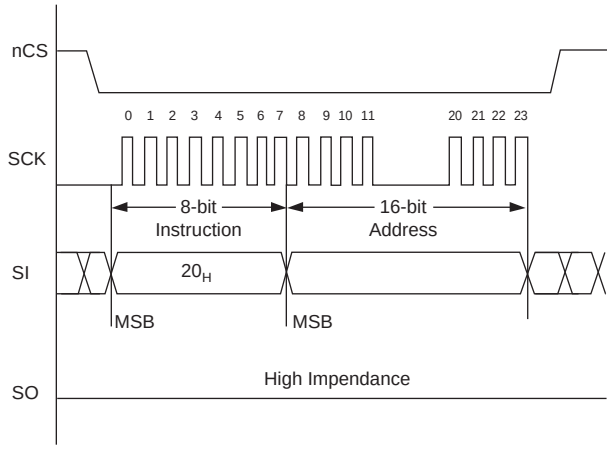
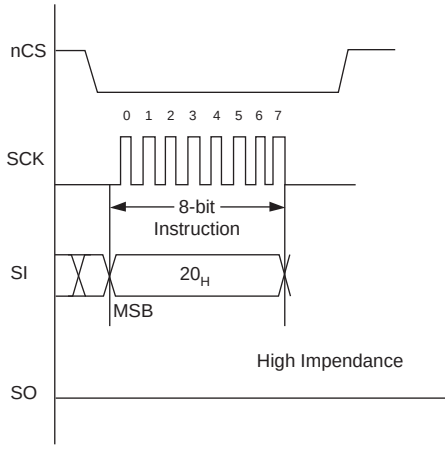



Figure 9–31 shows the SECTOR-ERASE operation sequence for Base mode.

**Figure 9–31.** Sector_ERASE Operation Sequence for Base Mode



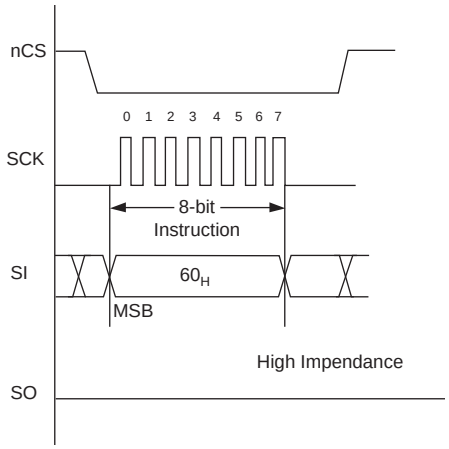
### UFM-ERASE

The UFM-ERASE (CE) instruction erases both UFM sector 0 and sector 1 for SPI Extended Mode. While for SPI Base mode, the CE instruction has the same functionality as the SECTOR-ERASE (SE) instruction, which erases UFM sector 0 only. WEN bit and the UFM sectors must not be protected for CE operation to be successful. nCS must be driven high before the instruction is executed internally. You may poll the nRDY bit in the software status register for the completion of the internal self-timed CE cycle. For both SPI Extended mode and Base mode, the UFM-ERASE operation is performed in the following sequence as shown in Figure 9–32:

1. nCS is pulled low.
2. Opcode 01100000 is transmitted into the interface.
3. nCS is pulled back to high.

Figure 9–32 shows the UFM-ERASE operation sequence.

**Figure 9–32.** UFM-ERASE Operation Sequence

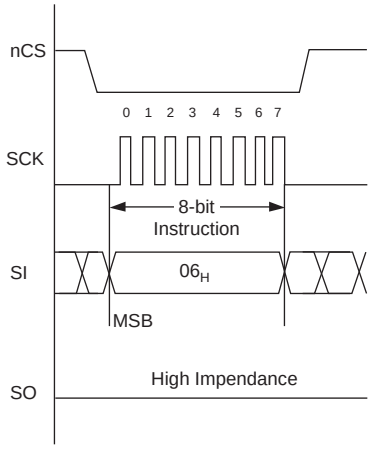



### WREN (Write Enable)

The interface is powered-up in the write disable state. Therefore, WEN in the status register (see Table 9–11) is 0 at power-up. Before any write is allowed to take place, WREN must be issued to set WEN in the status register to 1. If the interface is in read-only mode, WREN does not have any effect on WEN, since the status register does not exist. Once the WEN is set to 1, it can be reset by the WRDI instruction; the WRITE and SECTOR-ERASE instruction will not reset the WEN bit. WREN is issued through the following sequence, as shown in Figure 9–33:

1. nCS is pulled low.
2. Opcode 00000110 is transmitted into the interface to set WEN to 1 in the status register.
3. After the transmission of the eighth bit of WREN, the interface is in wait state (waiting for nCS to be pulled back to high). Any transmission after this is ignored.
4. nCS is pulled back to high.

**Figure 9–33.** WREN Operation Sequence

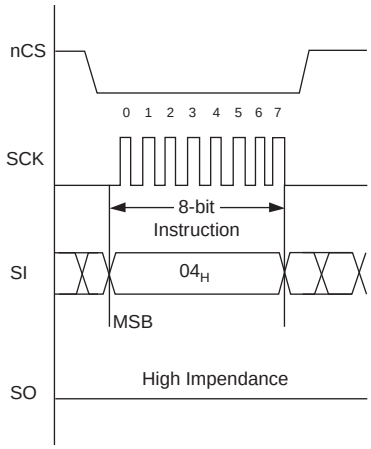



### WRDI (Write Disable)

After the UFM is programmed, WRDI can be issued to set WEN back to 0, disabling WRITE and preventing inadvertent writing to the UFM. WRDI is issued through the following sequence, as shown in Figure 9–34:

1. nCS is pulled low.
2. Opcode 00000100 is transmitted to set WEN to 0 in the status register.
3. After the transmission of the eighth bit of WRDI, the interface is in wait state (waiting for nCS to be pulled back to high). Any transmission after this is ignored.
4. nCS is pulled back to high.

**Figure 9–34.** WRDI Operation Sequence



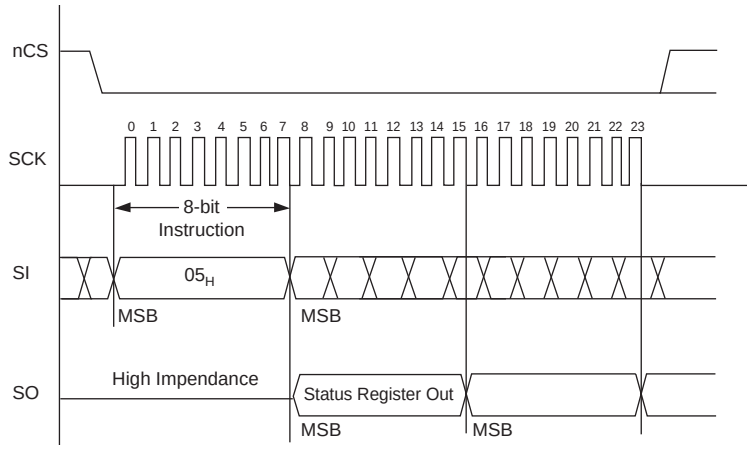

### RDSR (Read Status Register)

The content of the status register can be read by issuing RDSR. Once RDSR is received, the interface outputs the content of the status register through the SO port. Although the most significant four bits (Bit 7 to Bit 4) do not hold valuable information, all eight bits in the status register will output through the SO port. This allows future compatibility when Bit 7 to Bit 4 have new meaning in the status register. During the

internal program cycle in the UFM, RDSR is the only valid opcode recognized by the interface (therefore, the status register can be read at any time), and nRDY is the only valid status bit. Other status bits are frozen and remain unchanged until the internal program cycle is ended. RDSR is issued through the following sequence, as shown in Figure 9–35:

1. nCS is pulled low.
2. Opcode 00000101 is transmitted into the interface.
3. SI ignores incoming signals; SO output the content of the status register, Bit 7 first and Bit 0 last.
4. If nCS is kept low, repeat step 3.
5. nCS is pulled back to high to terminate the transmission.

**Figure 9–35.** RDSR Operation Sequence




### WRSR (Write Status Register)

The block protection bits (BP1 and BP0) are the status bits used to protect certain sections of the UFM from inadvertent write. The BP1 and BP0 status are updated by WRSR. During WRSR, only BP1 and BP0 in the status register can be written with valid information. The rest of the bits in the status register are ignored and not updated. When both BP1 and BP0 are 0, there is no protection for the UFM. When both BP1 and BP0 are 1, there is full protection for the UFM. BP0 and BP1 are set to 0 upon power-up. Table 9–12 describe more on the Block Write Protect Bits for Extended mode, while Table 9–13 describes more on the Block Write Protect Bits for Base mode. WRSR is issued through the following sequence, as shown in Figure 9–36:

1. nCS is pulled low.
2. Opcode 00000001 is transmitted into the interface.
3. An 8-bit status is transmitted into the interface to update BP1 and BP0 of the status register.
4. If nCS is pulled high too early (before all the eight bits in Step 2 or Step 3 are transmitted) or too late (the ninth bit or more is transmitted), WRSR is not executed.

5. nCS is pulled back to high to terminate the transmission.

**Figure 9–36.** WRSR Operation Sequence

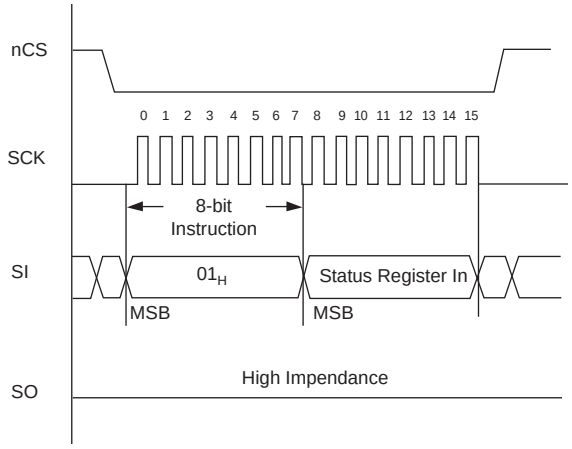



**Table 9–12.** Block Write Protect Bits for Extended Mode

| Level | Status Register Bits | | UFM Array Address Protected |
|---|---|---|---|
| | BP1 | BP0 | |
| 0 (No protection) | 0 | 0 | None |
| 3 (Full protection) | 1 | 1 | 000 to 1FF |

**Table 9–13.** Block Write Protect Bits for Base Mode

| Level | Status Register Bits | | UFM Array Address Protected |
|---|---|---|---|
| | BP1 | BP0 | |
| 0 (No protection) | 0 | 0 | None |
| 3 (Full protection) | 1 | 1 | 000 to 0FF |

## ALTUFM SPI Timing Specification

Figure 9–37 shows the timing specification needed for the SPI Extended mode (read/write). These nCS timing specifications do not apply to the SPI Extended read-only mode nor any of the SPI Base modes. However, for the SPI Extended mode (read only) and the SPI Base mode (both read only and read/write), the nCS signal and SCK are not allowed to toggle at the same time. Table 9–14 shows the timing parameters which only apply to the SPI Extended mode (read/write).

**Figure 9–37.** SPI Timing Waveform

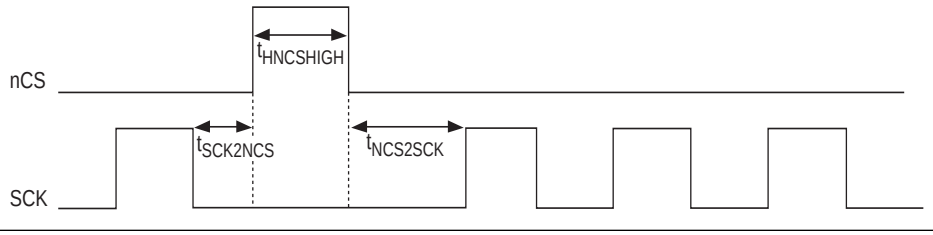


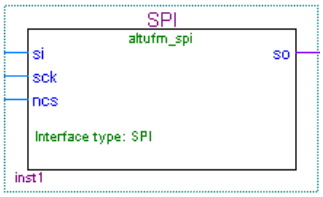
**Table 9–14.** SPI Timing Parameters for Extended Mode

| Symbol | Description | Minimum (ns) | Maximum (ns) |
|---|---|---|---|
| $t_{SCK2NCS}$ | The time required for the SCK signal falling edge to nCS signal rising edge | 50 | — |
| $t_{HNCSHIGH}$ | The time that the nCS signal must be held high | 600 | — |
| $t_{NCS2SCK}$ | The time required for the nCS signal falling edge to SCK signal rising edge | 750 | — |

## Instantiating SPI Using Quartus II altufm Megafunction

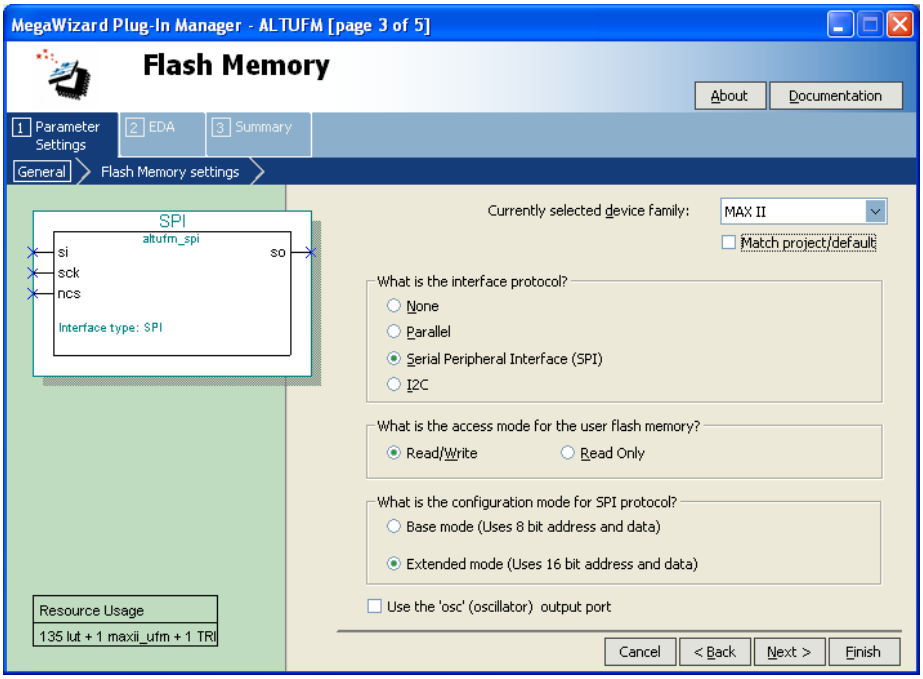
Figure 9–38 shows the altufm megafunction symbol for SPI instantiation in the Quartus II software.

**Figure 9–38.** altufm Megafunction Symbol for SPI Instantiation




You can select the desired logic array interface on page 3 of the altufm MegaWizard® Plug-In Manager. Figure 9–39 shows page 3 of the altufm MegaWizard Plug-In Manager, selecting SPI as the interface protocol. On this page, you can choose whether to implement the Read/Write or Read Only mode as the access mode for the UFM. You can also select the configuration mode (Base or Extended) for SPI on this page. You can specify the initial content of the UFM block in page 4 of the altufm MegaWizard Plug-In Manager as discussed in "Creating Memory Content File" on page 9–40.

**Figure 9–39.** Page 3 altufm MegaWizard Plug-In Manager (SPI)




The UFM block's internal oscillator is always running when the altufm_spi megafunction is instantiated for read/write interface. The UFM block's internal oscillator is disabled when the `altufm_spi` megafunction is instantiated for read only interface.

## Parallel Interface

This interface allows for parallel communication between the UFM block and outside logic. Once the READ request, WRITE request, or ERASE request is asserted (active low assertion), the outside logic or device (such as a microcontroller) are free to continue their operation while the data in the UFM is retrieved, written, or erased. During this time, the nBUSY signal is driven "low" to indicate that it is not available to respond to any further request. After the operation is complete, the nBUSY signal is brought back to "high" to indicate that it is now available to service a new request. If it was the Read request, the DATA_VALID is driven "high" to indicate that the data at the DO port is the valid data from the last read address.

Asserting READ, WRITE, and ERASE at the same time is not allowed. Multiple requests are ignored and nothing is read from, written to, or erased in the UFM block. There is no support for sequential read and page write in the parallel interface. For both the read only and the read/write modes of the parallel interface, OSC_ENA is always asserted, enabling the internal oscillator. Table 9–15 summarizes the parallel interface pins and functions.

**Table 9–15.** Parallel Interface Signals

| Pin | Description | Function |
|---|---|---|
| DI[15:0] | 16-bit data Input | Receive 16-bit data in parallel. You can select an optional width of 3 to 16 bits using the altufm megafunction. |
| DO[15:0] | 16-bit data Output | Transmit 16-bit data in parallel. You can select an optional width of 3 to 16 bits using the altufm megafunction. |
| ADDR[8:0] | Address Register | Operation sequence refers to the data that is pointed to by the address register. You can determine the address bus width using the altufm megafunction. |
| nREAD | READ Instruction Signal | Initiates a read sequence. |
| nWRITE | WRITE Instruction Signal | Initiates a write sequence. |
| nERASE | ERASE Instruction Signal | Initiates a SECTOR-ERASE sequence indicated by the MSB of the ADDR[] port. |
| nBUSY | BUSY Signal | Driven low to notify that it is not available to respond to any further request. |
| DATA_VALID | Data Valid | Driven high to indicate that the data at the DO port is the valid data from the last read address for read request. |

Even though the altufm megafunction allows you to select the address widths range from 3 bits to 9 bits, the UFM block always expects full 9 bits width for the address register. Therefore, the altufm megafunction will always pad the remaining LSB of the address register with '0's if the register width selected is less than 9 bits. The address register will point to sector 0 if the address received at the address register starts with a '0'. The address register will point to sector 1 if the address received starts with a '1'.

Even though you can select an optional data register width of 3 to 16 bits using the altufm megafunction, the UFM block always expects full 16 bits width for the data register. Reading from the data register always proceeds from MSB to LSB. The altufm megafunction always pads the remaining LSB of the data register with 1s if the user selects a data width of less than 16-bits.

### ALTUFM Parallel Interface Timing Specification

Figure 9–40 shows the timing specifications for the parallel interface. Table 9–16 parallel interface instruction signals. The nREAD, nWRITE, and nERASE signals are active low signals.

**Figure 9–40.** Parallel Interface Timing Waveform

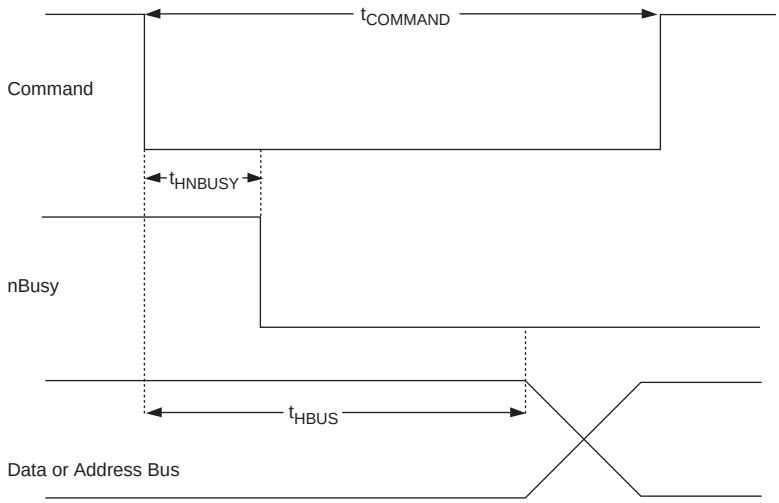



**Table 9–16.** Parallel Interface Timing Parameters

| Symbol | Description | Minimum (ns) | Maximum (ns) |
|---|---|---|---|
| $t_{COMMAND}$ | The time required for the command signal (nREAD/nWRITE/nERASE) to be asserted and held low to initiate a read/write/erase sequence | 600 | 3,000 |
| $t_{HNBUSY}$ | Maximum delay between command signal's falling edge to the nBUSY signal's falling edge | — | 300 |
| $t_{HBUS}$ | The time that data and/or address bus must be present at the data input and/or address register port after the command signal has been asserted low | 600 | — |

## Instantiating Parallel Interface Using Quartus II altufm Megafunction

Figure 9–41 shows the altufm megafunction symbol for a parallel interface instantiation in the Quartus II software.

**Figure 9–41.** altufm Megafunction Symbol for Parallel Interface Instantiation

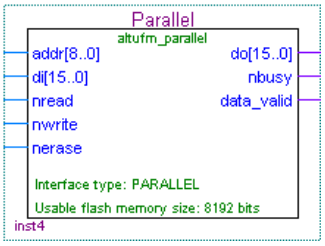
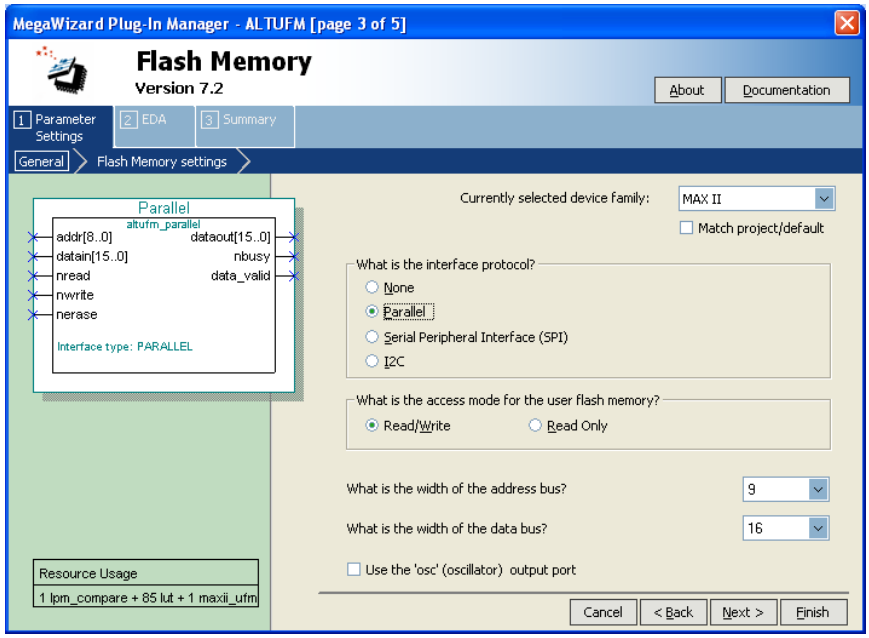



Figure 9–42 shows page 3 of the altufm MegaWizard Plug-In Manager, selecting the Parallel Interface as the interface. On this page, you can choose whether to implement the Read/Write mode or Read Only mode for the UFM. You also have an option to choose the width for address bus (up to 9 bits) and for the data bus (up to 16 bits). You can specify the initial content of the UFM block on page 4 of the altufm MegaWizard Plug-In Manager as discussed in "Creating Memory Content File" on page 9–40.

**Figure 9–42.** Page 3 altufm MegaWizard Plug-In Manager (Parallel)



The UFM block's internal oscillator is always running when the altufm_parallel magafunction is instantiated for read/write interface. The UFM block's internal oscillator is disabled when the altufm_parallel megafunction is instantiated for read only interface.

## None (Altera Serial Interface)

None means using the dedicated UFM serial interface. The built-in UFM interface uses 13 pins for the communication. The functional description of the 13 pins are described in Table 9–4 on page 9–3. You can produce your own interface design to communicate to/from the dedicated UFM interface and implement it in the logic array.

### Instantiating None Using Quartus II altufm Megafunction

Figure 9–43 shows the altufm megafunction symbol for None instantiation in the Quartus II software.

**Figure 9–43.** altufm Megafunction Symbol for None Instantiation

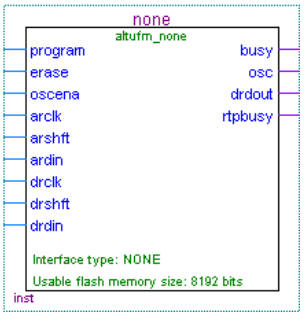
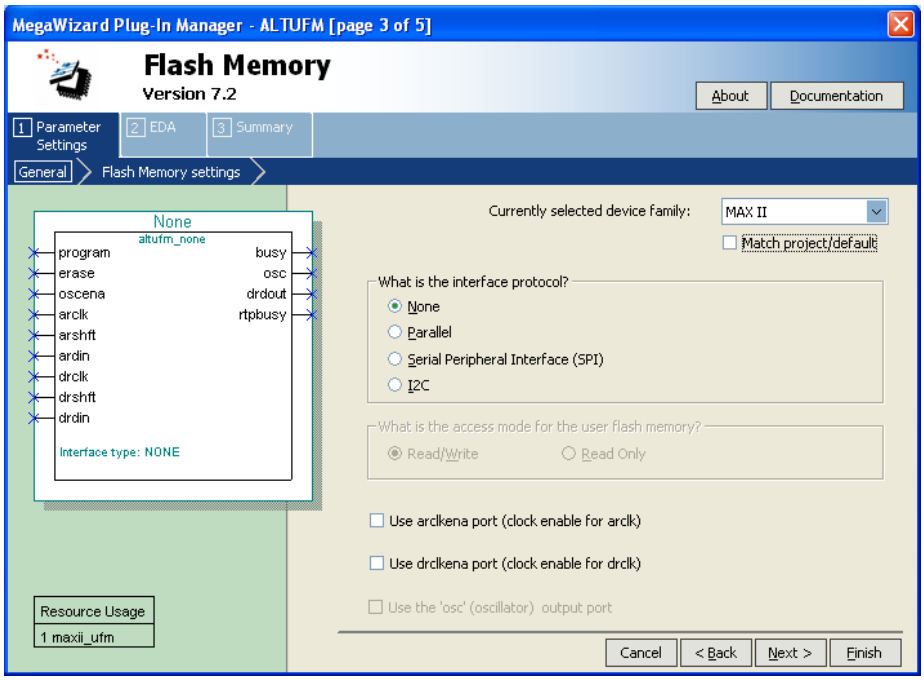


Figure 9–44 shows page 3 of the altufm MegaWizard Plug-In Manager, selecting **none** for the interface protocol. By selecting **none**, all the other options are grayed out or unavailable to you. However, you still can specify the initial content of the UFM block on page 4 of the altufm MegaWizard Plug-In Manager as discussed in "Creating Memory Content File" on page 9–40.

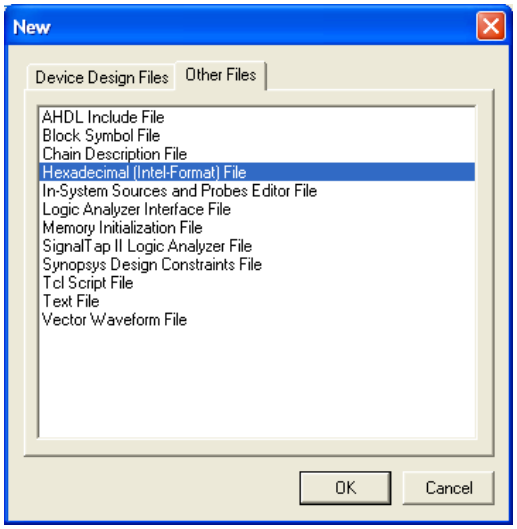
**Figure 9–44.** Page 3 altufm MegaWizard Plug-In Manager (None)




# Creating Memory Content File

You can initialize the content of the UFM through a memory content file. Quartus II software supports two types of initial memory content file format: Memory Initialization File (**.mif**) and Hexadecimal File (**.hex**). A new memory content file for the UFM block can be created by clicking **New** on the File menu. Select the HEX file or MIF in the **Other Files** tab (Figure 9–45).

**Figure 9–45.** Create New File Dialog Box




Immediately after clicking **OK**, a dialog box appears. In this dialog box, the **Number of words** represents the numbers of address lines while the **Word size** represents the data width. To create a memory content file for the altufm megafunction, enter 512 for the number of words and 16 for the word size, as shown in Figure 9–46.

**Figure 9–46.** Number of Words and Word Size Dialog Box

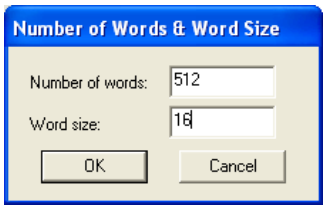


Figure 9–47 shows the memory content being written into a HEX file.

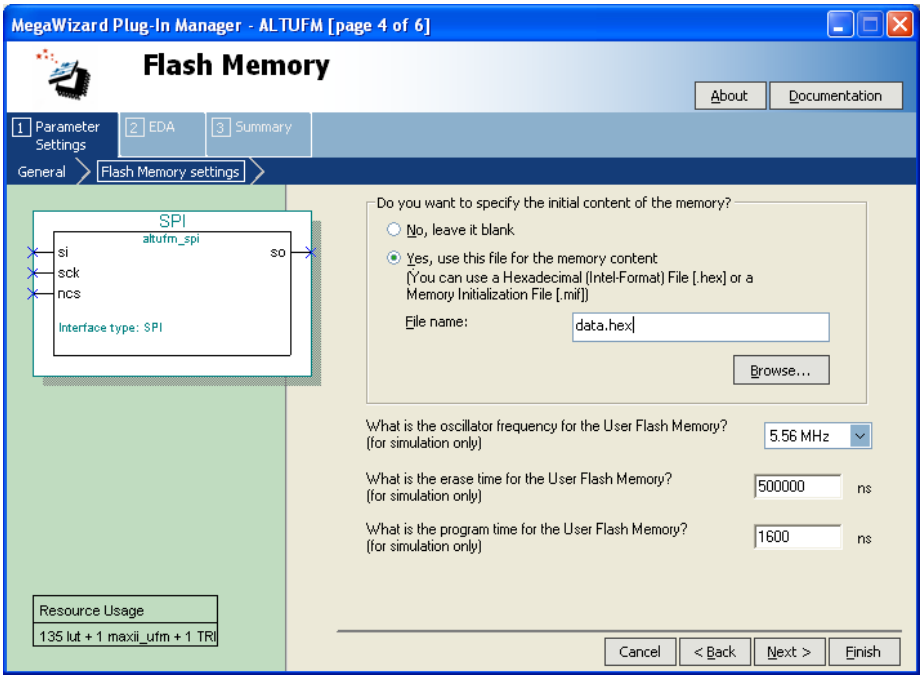
**Figure 9–47.** Hexadecimal (Intel-Format) File

data.hex

| Addr | +0 | +1 | +2 | +3 | +4 | +5 | +6 | +7 |
|---|---|---|---|---|---|---|---|---|
| 000 | 003C | 0044 | 0045 | 0053 | 0049 | 0047 | 004E | 0020 |
| 008 | 006E | 0061 | 006D | 0065 | 003D | 0020 | 0022 | 004C |
| 010 | 0045 | 0044 | 0020 | 0044 | 0072 | 0069 | 0076 | 0065 |
| 018 | 0072 | 0020 | 0046 | 0055 | 004E | 0043 | 0054 | 0049 |
| 020 | 004F | 004F | 0022 | 003E | 000D | 003C | 0043 | 004F |
| 028 | 004D | 0050 | 0041 | 004E | 0059 | 003E | 0020 | 0041 |
| 030 | 004C | 0054 | 0045 | 0052 | 0041 | 0020 | 0043 | 004F |
| 038 | 0052 | 0050 | 004F | 0052 | 0041 | 0054 | 0049 | 004F |
| 040 | 004E | 0020 | 0077 | 0077 | 0077 | 002E | 0061 | 006C |
| 048 | 7400 | 0065 | 0072 | 0061 | 002E | 0063 | 006F | 006D |
| 050 | 000D | 003C | 0044 | 0065 | 0076 | 0069 | 0063 | 0065 |
| 058 | 004E | 0061 | 0074 | 0061 | 003E | 000D | 003C | 0044 |
| 060 | 0065 | 0076 | 0069 | 0063 | 0065 | 004E | 0061 | 006D |
| 068 | 0065 | 003E | 0020 | 0045 | 0050 | 004D | 0031 | 0032 |
| 070 | 0037 | 0030 | 0054 | 0031 | 0034 | 0034 | 0043 | 0033 |
| 078 | 000D | 003C | 0050 | 0061 | 0072 | 0074 | 004E | 006F |
| 080 | 003E | 0020 | 0032 | 0030 | 0030 | 0033 | 002D | 004D |
| 088 | 0041 | 0058 | 0032 | 002D | 0031 | 0031 | 0033 | 000D |
| 090 | 003C | 004D | 0061 | 006E | 0075 | 0063 | 0044 | 0061 |
| 098 | 0074 | 0065 | 003E | 0020 | 0031 | 0031 | 002F | 0033 |
| 0a0 | 002F | 0032 | 0030 | 0030 | 0033 | 0020 | 0032 | 002E |
| 0a8 | 0031 | 0037 | 0050 | 004D | 003C | 0052 | 0065 | 0076 |
| 0b0 | 0069 | 0073 | 0069 | 006F | 006E | 003E | 0020 | 0061 |
| 0b8 | 0030 | 002E | 0039 | 0035 | 000D | 003C | 0043 | 0068 |
| 0c0 | 0065 | 0063 | 006B | 0053 | 0075 | 006D | 003E | 0020 |
| 0c8 | 0030 | 0030 | 0030 | 0030 | 0043 | 0038 | 0039 | 0039 |
| 0d0 | 000D | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0d8 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0e0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0e8 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0f0 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0f8 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 100 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 108 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 110 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |

This memory content file is then included using the altufm megafunction. On the Tools menu, click **MegaWizard Plug-In Manager**. The memory content file (**data.hex**) is included on page 5 of the altufm megafunction (Figure 9–48). Click **Yes**, and use this file for the memory content file. Click **Browse** to include the memory content file.

**Figure 9–48.** Page 4 of the altufm Megafunction




### Memory Initialization for the altufm_parallel Megafunction

For the parallel interface, if a HEX file is used to initialize the memory content for the altufm megafunction, you have to fully specify all 16 bits in each memory address, regardless of the data width selected. If your data width is less than 16 bits wide, your data must be placed in the MSBs of the data word and the remaining LSBs must be padded with 1's.

For an example, if `address_width = 3` and `data_width = 8` are selected for the altufm_parallel megafunction, the HEX file should contain eight addresses of data ($2^3$ addresses), each word containing 16 bits. If the initial content at the location `000` is intended to be `10101010`, you should specify `1010101011111111` for address `000` in the HEX file.

This specification applies only to HEX files used with the parallel interface. MIFs do not require you to fully specify 16 bits for each data word. However, both MIF and HEX files require you to specify all addresses of data according to the `address_width` selected in the megafunction.

### Memory Initialization for the altufm_spi Megafunction

The same 16-bit data padding mentioned for altufm_parallel is required for HEX files used with the SPI Base (8 bits) and Extended (16 bits) mode interface. In addition, for SPI Base and Extended mode, you must fully specify memory content for all 512 addresses (both sector 0 and sector 1) in the HEX file and MIF, even if sector 1 is not used. You can put valid data for SPI Base mode addresses 0 to 255 (sector 0), and initialize sector 1 to all ones.

## Memory Initialization for the altufm_i2c Megafunction

The MAX II UFM physical memory block contains a 16-bit wide and 512 deep (9-bit address) array. The altufm_i2c megafunction uses the following smaller array sizes:

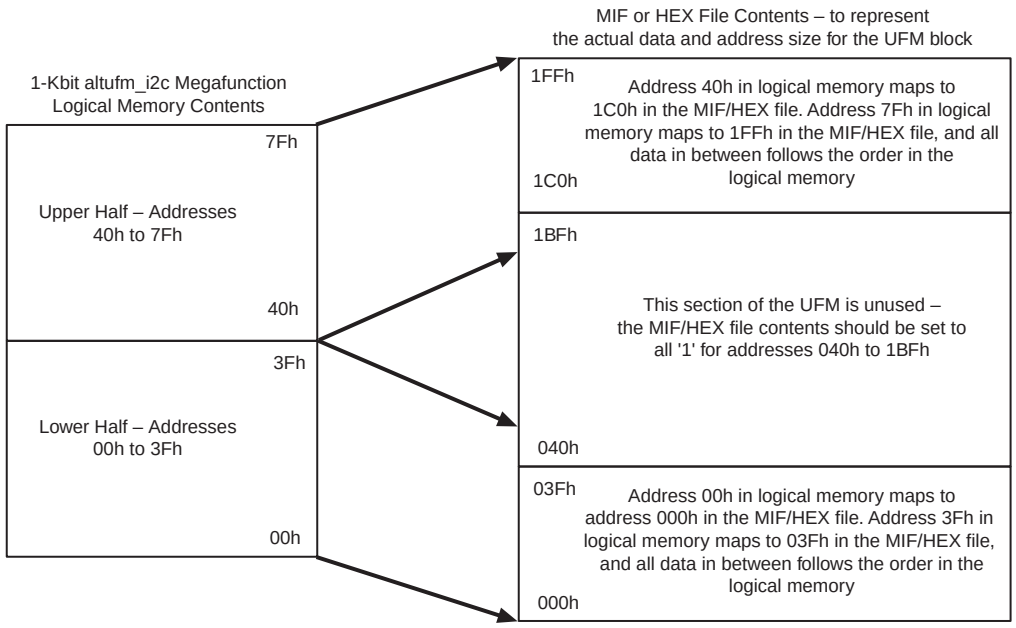
- An 8-bit wide and 128 deep (7-bit address) mapping for 1 Kbit memory size
- An 8-bit wide and 256 deep (8-bit address) mapping for 2 Kbits memory size
- An 8-bit wide and 512 deep (9-bit address) mapping for 4 Kbits memory size
- An 8-bit wide and 1,024 deep (10-bit address) mapping for 8 Kbits memory size

Altera recommends that you pad the MIF or HEX file for both address and data width to fill the physical memory map for the UFM block and ensure the MIF/HEX file represents a full 16-bit word size and a 9-bit address space.

### Memory Map for 1-Kbit Memory Initialization

Figure 9–49 shows the memory map initialization for the altufm_i2c megafunction of 1-Kbit memory size. The altufm_i2c megafunction byte address location of `00h` to `3Fh` is mapped to the UFM block address location of `000h` to `03Fh`. The altufm_i2c megafunction byte address location of `40h` to `7Fh` is mapped to the UFM block address location of `1C0h` to `1FFh`. Altera recommends that you pad the unused address locations of the UFM block with all ones.

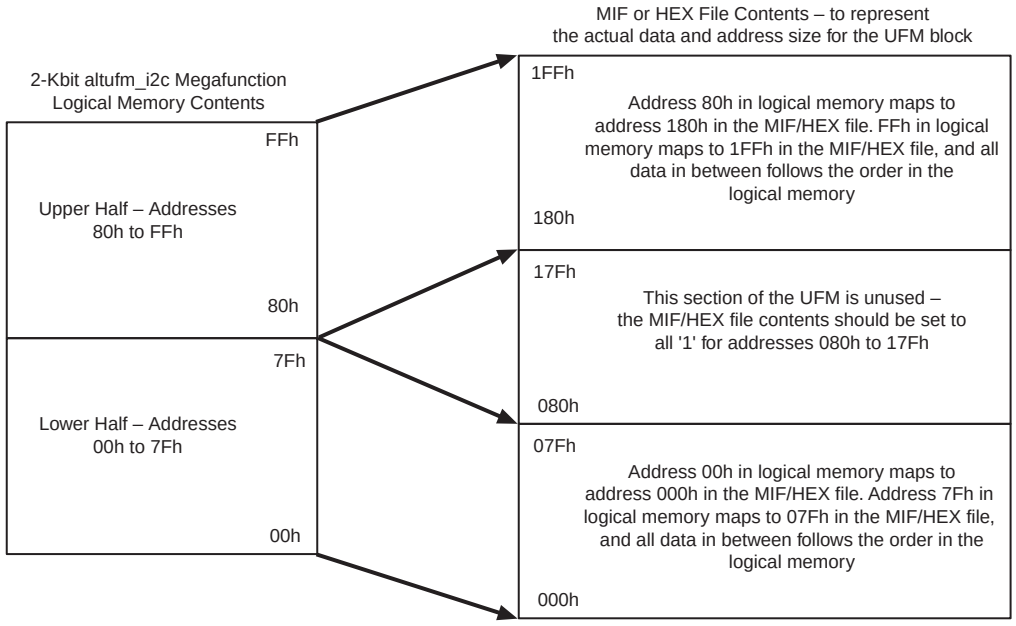
**Figure 9–49.** Memory Map for 1-Kbit Memory Initialization




### Memory Map for 2-Kbit Memory Initialization

Figure 9–50 shows the memory map initialization for the altufm_i2c megafunction of 2 Kbits of memory. The altufm_i2c megafunction byte address location of `00h` to `7Fh` is mapped to the UFM block address location of `000h` to `07Fh`. The altufm_i2c megafunction byte address location of `80h` to `FFh` is mapped to the UFM block address location of 180h to 1FFh. Altera recommends that you pad the unused address location of the UFM block with all ones.

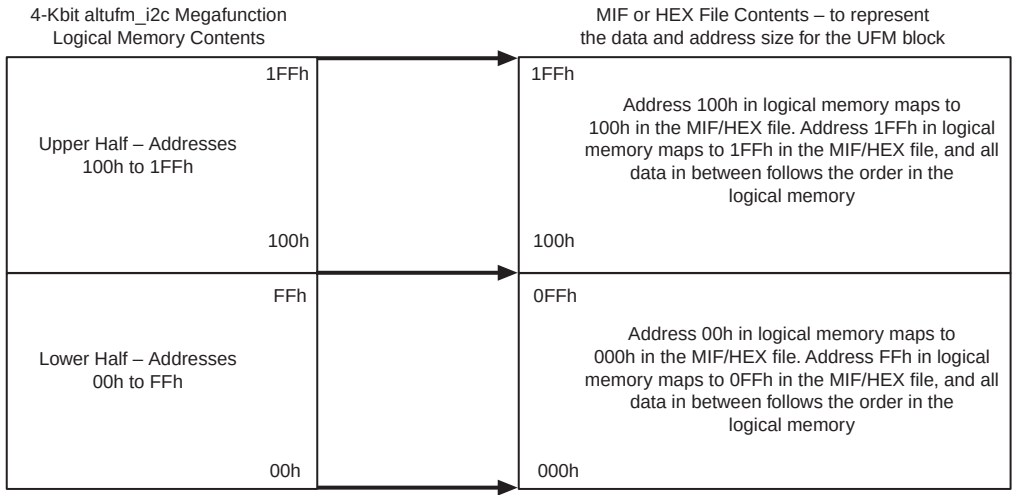
**Figure 9–50.** Memory Map for 2-Kbit Memory Initialization




### Memory Map for 4-Kbit Memory Initialization

Figure 9–49 shows the memory map initialization for the altufm_i2c megafunction of 4-Kbit memory. The altufm_i2c megafunction byte address location of `00h` to `FFh` is mapped to the UFM block address location of `000h` to `0FFh`. The altufm_i2c megafunction byte address location of `100h` to `1FFh` is mapped to the UFM block address location of `100h` to `1FFh`.

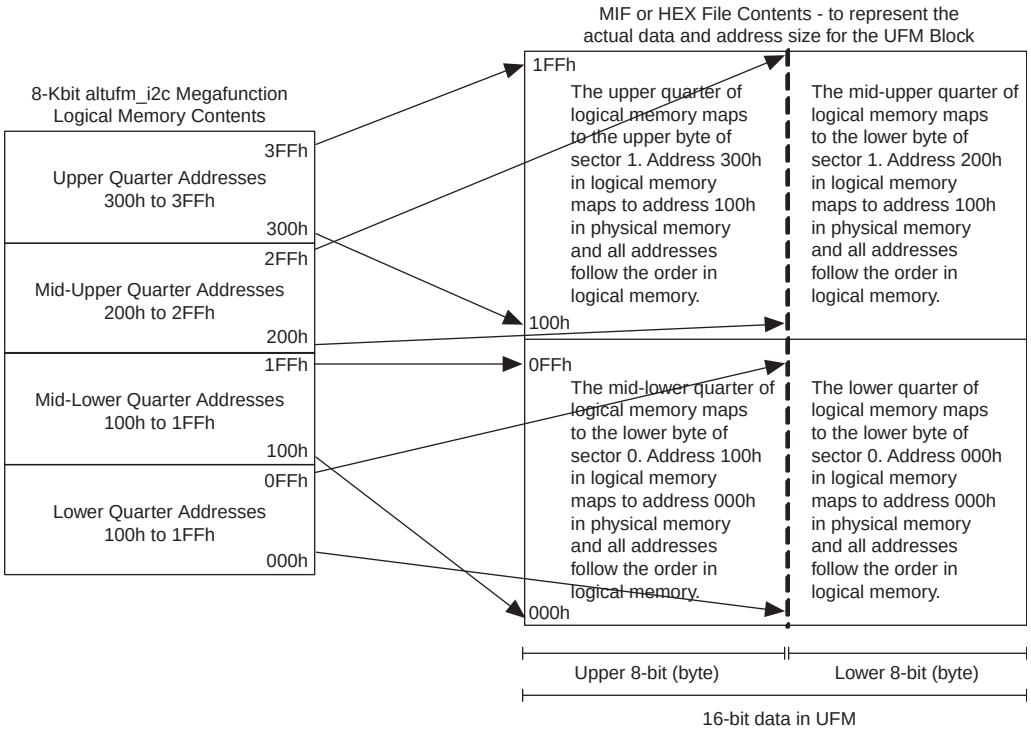
**Figure 9–51.** Memory Map for 4-Kbit Memory Initialization




### Memory Map for 8-Kbit Memory Initialization

Figure 9–52 shows the memory map initialization for the altufm_i2c megafunction of 8-Kbit memory. The altufm_i2c megafunction of
8-Kbit memory fully utilizes all the memory locations in the UFM block.

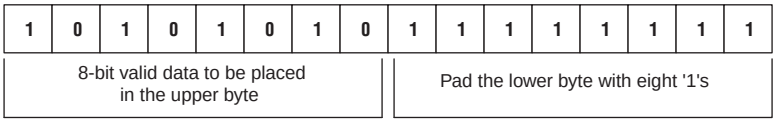
**Figure 9–52.** Memory Map for 8-Kbit Memory Initialization




### Padding Data into Memory Map

The altufm_i2c megafunction uses the upper 8 bits of the UFM 16-bit word; therefore, the 8 least significant bit (LSB) should be padded with 1, as shown in Figure 9–53.

**Figure 9–53.** Padding Data into Memory Map




# Simulation Parameters

Figure 9–48 on page 9–43 shows page 4 of the altufm megafunction where you can have an option to choose to simulate the OSC output port at the maximum or the minimum frequency during the design simulation. The frequency chosen is only used as the timing parameter for the Quartus II simulator and does not affect the real MAX II device OSC output frequency.

# Conclusion

The MAX II UFM block is a user-accessible, programmable non-volatile flash memory block that provides significant flexibility in its interfacing. MAX II devices fill the need for on-board non-volatile storage in any application, minimizing board space and reducing total system cost.

# Referenced Documents

This chapter references the following documents:

- *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*
- *MAX II Architecture* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 9–17 shows the revision history for this chapter.

**Table 9–17.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.8 | ■ Updated "Using and Accessing UFM Storage", "Oscillator", "UFM Operating Modes", "ALTUFM SPI Timing Specification", and "ALTUFM Parallel Interface Timing Specification" sections.<br>■ Updated New Document Format. | — |
| December 2007, version 1.7 | ■ Corrected Figure 9–3.<br>■ Added "Referenced Documents". | — |
| December 2006, version 1.6 | ■ Changed signal format in Table 9–4. Added Revision History section. | — |
| August 2005, version 1.5 | ■ Added I²C row to Table 9-3.<br>■ Added a new *Inter-Integrated Circuit* section.<br>■ Added a new *Memory Initialization for the altufm_i2c Megafunction* section<br>■ Updated Figure 9-39. | — |
| June 2005, version 1.4 | ■ Added the Instantiating the Oscillator without the UFM section.<br>■ Updated Figure 9-14. | — |
| January 2005, version 1.3 | ■ Previously published as Chapter 10. No changes to content. | — |

**Table 9–17.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| December 2004<br>v1.2 | ■ Updated text to RTP_BUSY in Table 9-4.<br>■ Updated text in the Oscillator section.<br>■ Updated text in the UFM Operating Modes section.<br>■ Updated text in the Serial Peripheral Interface section.<br>■ Added a row to Table 9-6.<br>■ Updated Table 9-7.<br>■ Updated text to the READ section.<br>■ Updated text to the WRITE section.<br>■ Updated text to the SECTOR-ERASE section.<br>■ Added a new UFM-ERASE section.<br>■ Updated text to the WRSR section.<br>■ Updated Table 9-8.<br>■ Added Table 9-9.<br>■ Added section ALTUFM SPI Timing Specification.<br>■ Added Figures 9-13, 9-15, 9-16, 9-21, and 9-24.<br>■ Added Table 9-10.<br>■ Added section ALTUFM Parallel Interface Timing Specification.<br>■ Added section Simulation Parameters.<br>■ Added Table 9-12 | — |
| June 2004<br>v1.1 | ■ Updated Figures 9-4 through 9-7. | — |

# 10. Replacing Serial EEPROMs with MAX II User Flash Memory

MII51012-1.5

## Introduction

Each MAX® II device has a user flash memory (UFM) block to store up to 8 Kbits of user data. You can use the UFM block to replace on-board flash and EEPROM memory devices which are used to store ASSP or processor configuration bits, or electronic ID information for a board during manufacturing. MAX II device logic capacity allows integration of system power-on reset (POR), interface bridging, and I/O expansion designs in addition to these serial flash capabilities.

This chapter provides a comprehensive listing of 2-Kbit, 4-Kbit, and 8-Kbit, non-volatile memory devices that could be potentially replaced by MAX II UFM devices. Table 10–1 shows the capacity for the UFM block for all MAX II devices.

**Table 10–1.** MAX II UFM Array Size

| Device | Total Bits | Sectors | Address Bits | Data Width |
|---|---|---|---|---|
| EPM240<br>EPM570<br>EPM1270<br>EPM2210 | 8,192 | 2 (4096 bits per sector) | 9 | 16 |

This chapter contains the following sections:


- "Design Considerations" on page 10–1
- "List of Vendors and Devices" on page 10–2


## Design Considerations

The MAX II UFM can be programmed, erased, and verified through the Joint Test Action Group (JTAG) port or through connections to/from the logic array in accordance with IEEE Std. 1532-2002. There are 13 interface signals to and from the UFM block and logic array which allow the logic array to read or write to the UFM during device user mode. A reference design or user logic can be used to interface the UFM to many standard interface protocols such as Serial Communication Interface (SCI), Serial Peripheral Interface (SPI), Inter-Integrated Circuit ($I^2C$), Microwire, or other proprietary protocols. Altera's Quartus® II altufm megafunction provides interface logic for a subset of these interfaces (parallel and SPI). Any interfaces not provided by the megafunction or design examples, require you to create user logic to bridge the UFM block to your desired interface protocol.

For more information about programming and erasing the UFM block and/or the altufm megafunction, refer to the *Using User Flash Memory in MAX II Devices* chapter in the *MAX II Device Handbook*.

The differences between the UFM block and serial EEPROMs that you should consider in your integration of serial EEPROM applications are the sector-based erase and erase/reprogram cycles. Serial EEPROMs support byte wide erase, which is automatically implemented during a byte write sequence. The UFM block supports byte writes, but does not support byte erase requiring a sector-based erase sequence prior to any programming or writing. If the data content of a specific byte location needs to be overwritten in the UFM, the entire sector that byte resides in must be erased unless that byte location was already erased (all 1s). For programming endurance, the UFM erase/reprogram cycles do not meet the $10^7$ and greater cycles seen in serial EEPROMs.

Refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* for the MAX II UFM block erase/programming endurance specification.

# List of Vendors and Devices

Table 10–2 through Table 10–10 list the vendors and their devices which can be replaced by the MAX II UFM block. The operating condition range for the UFM block and MAX II devices are within the range of the devices listed.

**Table 10–2.** Asahi Kasei Microsystems Co. Device Characteristics

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ (MHz) | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | 1-Wire | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | AK93C75AV | 8,192 | — | — | — | — | — | ✓ | — | 1.8 to 5.5 |
| EEPROM | AK93C75BH | 8,192 | — | — | — | — | — | ✓ | — | 1.8 to 5.5 |
| EEPROM | AK6480AF/M | 8,192 | ✓ | — | — | — | — | — | 1 | 1.8 to 5.5 |
| EEPROM | AK6480BH/L | 8,192 | ✓ | — | — | — | — | — | 1 | 1.8 to 5.5 |
| EEPROM | AK93C65AF/V | 4,096 | — | — | — | — | — | ✓ | — | 1.8 to 5.5 |
| EEPROM | AK93C65BH | 4,096 | — | — | — | — | — | ✓ | — | 1.8 to 5.5 |
| EEPROM | AK93C61AV | 4,096 | — | — | — | — | — | ✓ | — | 0.9 to 3.6 |
| EEPROM | AK6440AF/M | 4,096 | ✓ | — | — | — | — | — | 1 | 1.8 to 5.5 |
| EEPROM | AK6440BH/L | 4,096 | ✓ | — | — | — | — | — | 1 | 1.8 to 5.5 |
| EEPROM | AK6004AF | 4,096 | — | — | — | — | ✓ | — | — | 1.8 to 5.5 |
| EEPROM | AK93C55AF/V | 2,048 | — | — | — | — | — | ✓ | — | 1.8 to 5.5 |
| EEPROM | AK93C55BH | 2,048 | — | — | — | — | — | ✓ | — | 1.8 to 5.5 |
| EEPROM | AK93C51AV | 2,048 | — | — | — | — | — | ✓ | — | 0.9 to 3.6 |
| EEPROM | AK6420AF/M | 2,048 | ✓ | — | — | — | — | — | 1 | 1.8 to 5.5 |
| EEPROM | AK6420BH | 2,048 | ✓ | — | — | — | — | — | 1 | 1.8 to 5.5 |
| EEPROM | AK6003AV | 2,048 | — | — | — | — | ✓ | — | — | 1.8 to 5.5 |

**Note to Table 10–2:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–3.** Atmel Corporation Device Characteristics

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | AT25020 | 2,048 | — | ✓ | — | — | — | — | 3 MHz | 2.7 (2.7 ~ 5.5) |
| EEPROM | AT25040 | 4,096 | — | ✓ | — | — | — | — | 3 MHz | 2.7 (2.7 ~ 5.5) |
| EEPROM | AT25020A | 2,048 | — | ✓ | — | — | — | — | 20 MHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT25040A | 4,096 | — | ✓ | — | — | — | — | 20 MHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT25080 | 8,192 | — | ✓ | — | — | — | — | 3 MHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT25080A | 8,192 | — | ✓ | — | — | — | — | 20 MHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT24C02 | 2,048 | — | — | ✓ | — | — | — | 400 kHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT24C04 | 4,096 | — | — | ✓ | — | — | — | 400 kHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT24C08 | 8,192 | — | — | ✓ | — | — | — | 400 kHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT24C02A | 2,048 | — | — | ✓ | — | — | — | 400 kHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT24C04A | 4,096 | — | — | ✓ | — | — | — | 400 kHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT24C08A | 8,192 | — | — | ✓ | — | — | — | 400 kHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT34C02 | 2,048 | — | — | ✓ | — | — | — | 400 kHz | 2.7 (2.7 ~ 5.5)<br>1.8 (1.8 ~ 5.5) |
| EEPROM | AT93C56 | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 2.7 (2.7 ~5.5)<br>2.5 (2.5 ~ 5.5)<br>1.8 (1.8 ~5.5) |
| EEPROM | AT93C66 | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 2.7 (2.7 ~5.5)<br>2.5 (2.5 ~ 5.5)<br>1.8 (1.8 ~5.5) |

**Note to Table 10–3:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–4.** Catalyst Semiconductor, Inc. Device Characteristics (Part 1 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | CAT93C56 | 2,048 | — | — | — | — | — | ✓ | 1 MHz | 1.8 to 6.0 |
| EEPROM | CAT93C57 | 2,048 | — | — | — | — | — | ✓ | 1 MHz | 1.8 to 6.0 |
| EEPROM | CAT93C66 | 4,096 | — | — | — | — | — | ✓ | 1 MHz | 1.8 to 6.0 |
| EEPROM | CAT34WC02 | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 6.0 |
| EEPROM | CAT24WC03 | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 6.0 |

**Table 10–4.** Catalyst Semiconductor, Inc. Device Characteristics (Part 2 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) (1) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | CAT24WC05 | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 6.0 |
| EEPROM | CAT24WC02 | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 6.0 |
| EEPROM | CAT24WC04 | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 6.0 |
| EEPROM | CAT24WC08 | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 6.0 |
| EEPROM | CAT64LC20 | 2,048 | — | ✓ | — | — | — | — | 1 MHz | 2.5 to 6.0 |
| EEPROM | CAT64LC40 | 4,096 | — | ✓ | — | — | — | — | 1 MHz | 2.5 to 6.0 |
| EEPROM | CAT25C02 | 2,048 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |
| EEPROM | CAT25C03 | 2,048 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |
| EEPROM | CAT25C04 | 4,096 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |
| EEPROM | CAT25C05 | 4,096 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |
| EEPROM | CAT25C08 | 8,192 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |
| EEPROM | CAT25C09 | 8,192 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |
| EEPROM | CAT25020 | 2,048 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |
| EEPROM | CAT25040 | 4,096 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 6.0 |

**Note to Table 10–4:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–5.** Dallas Semiconductor – Maxim Integrated Products, Inc. Device Characteristics

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ (MHz) | Operating Voltage (V) (1) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | 1-Wire | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | DS2433 | 4,096 | — | ✓ | — | — | — | — | — | 2.8 to 6.0 |

**Note to Table 10–5:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–6.** Fairchild Semiconductor Device Characteristics (Part 1 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) (1) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | FM34W02UL | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.7 to 5.5 |
| EEPROM | FM93C56L | 2,048 | — | — | — | — | — | ✓ | 1 MHz | 2.7 to 5.5 |
| EEPROM | FM93C66L | 4,096 | — | — | — | — | — | ✓ | 1 MHz | 2.7 to 5.5 |
| EEPROM | FM93CS56L | 2,048 | — | — | — | — | — | ✓ | 1 MHz | 2.7 to 5.5 |
| EEPROM | FM93CS66L | 4,096 | — | — | — | — | — | ✓ | 1 MHz | 2.7 to 5.5 |
| EEPROM | FM24C08UL | 8,192 | — | — | ✓ | — | — | — | 400 kHz | 2.7 to 5.5 |
| EEPROM | FM24C09UL | 8,192 | — | — | ✓ | — | — | — | 400 kHz | 2.7 to 5.5 |

**Table 10–6.** Fairchild Semiconductor Device Characteristics (Part 2 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | NM24C02L | 2,048 | — | — | ✓ | — | — | — | 400 kHz | 2.7 to 5.5 |
| EEPROM | NM25C020L | 2,048 | — | ✓ | — | — | — | — | 2.1 MHz | 2.7 to 5.5 |
| EEPROM | NM25C040L | 4,096 | — | ✓ | — | — | — | — | 2.1 MHz | 2.7 to 5.5 |

**Note to Table 10–6:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–7.** Holtek Semiconductor Inc. Device Characteristics

| Type | Device | Size (Bits) | Interface | | | | | | Clock Rate (MHz) ($V_{CC}$ = 5.0 V) | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | 1-Wire | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | HT24LC02 | 2,048 | — | — | ✓ | — | — | — | 0.4 | 2.2 to 5.5 |
| EEPROM | HT24LC04 | 4,096 | — | — | ✓ | — | — | — | 0.4 | 2.4 to 5.5 |
| EEPROM | HT24LC08 | 8,192 | — | — | ✓ | — | — | — | 0.4 | 2.4 to 5.5 |
| EEPROM | HT93LC56 | 2,048 | — | — | — | ✓ | — | — | 1 | Read: 2.0 ~ 5.5 Write: 2.4 ~ 5.5 |
| EEPROM | HT93LC66 | 4,096 | — | — | — | ✓ | — | — | 1 | Read: 2.0 ~ 5.5 Write: 2.4 ~ 5.5 |

**Note to Table 10–7:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–8.** Microchip Technology Inc. Device Characteristics (Part 1 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | 24LCS62 | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 24LCS52 | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 24LC22A | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 24LC02B | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 24LC025 | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 24LC024 | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 24C02SC | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 24LCS22A | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | 24AA52 | 2,048 | — | — | — | — | ✓ | — | 100 kHz | 1.8 to 5.5 |
| EEPROM | 24AA02 | 2,048 | — | — | — | — | ✓ | — | 100 kHz | 1.8 to 5.5 |
| EEPROM | 24AA04 | 4,096 | — | — | — | — | ✓ | — | 400 kHz *(2)* | 1.8 to 5.5 |
| EEPROM | 24AA08 | 8,192 | — | — | — | — | ✓ | — | 400 kHz *(2)* | 1.8 to 5.5 |

**Table 10–8.** Microchip Technology Inc. Device Characteristics (Part 2 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) (1) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | 24LC04B | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | 24LC08B | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | 24LC09 (3) | 8,192 | — | — | Advanced Communication Riser (4) | — | — | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | 93LC66A | 4,096 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 6.0 |
| EEPROM | 93AA66 | 4,096 | — | — | — | — | — | ✓ | 2 MHz | 1.8 to 5.5 |
| EEPROM | 93LC66B | 4,096 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 6.0 |
| EEPROM | 93LC56A | 2,048 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 6.0 |
| EEPROM | 93AA56 | 2,048 | — | — | — | — | — | ✓ | 2 MHz | 1.8 to 5.5 |
| EEPROM | 93LC56B | 2,048 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 6.0 |
| EEPROM | 25LC080 | 8,192 | — | ✓ | — | — | — | — | 2 MHz | 2.5 to 5.5 |
| EEPROM | 25LC040 | 4,096 | — | ✓ | — | — | — | — | 2 MHz | 2.5 to 5.5 |
| EEPROM | 25AA080 | 8,192 | — | ✓ | — | — | — | — | 1 MHz | 1.8 to 5.5 |
| EEPROM | 25AA040 | 4,096 | — | ✓ | — | — | — | — | 1 MHz | 1.8 to 5.5 |

**Notes to Table 10–8:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

(2) 100 kHz for $V_{CC} < 2.5$ V.

(3) This device is designed to meet the proprietary protocol.

(4) Proprietary protocol by Microchip Technology Inc.

**Table 10–9.** Philips Semiconductors Device Characteristics

| Type | Device | Size (bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) (1) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | PCF8582C-2 | 2,048 | — | — | — | — | ✓ | — | 100 kHz | 2.5 to 6.0 |
| EEPROM | PCF8594C-2 | 4,096 | — | — | — | — | ✓ | — | 100 kHz | 2.5 to 6.0 |
| EEPROM | PCF8598C-2 | 8,192 | — | — | — | — | ✓ | — | 100 kHz | 2.5 to 6.0 |
| EEPROM | PCF85102C-2 | 2,048 | — | — | — | — | ✓ | — | 100 kHz | 2.5 to 6.0 |
| EEPROM | PCF85103C-2 | 2,048 | — | — | — | — | ✓ | — | 100 kHz | 2.5 to 6.0 |

**Note to Table 10–9:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–10.** Rohm Co., Ltd. Device Characteristics (Part 1 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | BR24L02-W | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L04-W | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L08-W | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L02F-W | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L04F-W | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L08F-W | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L02FJ-W | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L04FJ-W | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L08FJ-W | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L02FV-W | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L04FV-W | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L08FV-W | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L02FVM-W | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L04FVM-W | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR24L08FVM-W | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | BR93L56-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L56F-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66F-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L56RF-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66RF-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L56FJ-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66FJ-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L56RFJ-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66RFJ-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L56FV-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66FV-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L56RFV-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66RFV-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L56RFVM-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR93L66RFVM-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 1.8 to 5.5 |
| EEPROM | BR9020-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9040-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9080AF-W | 8,192 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9020F-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9040F-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9080ARFV-W | 8,192 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |

**Table 10–10.** Rohm Co., Ltd. Device Characteristics (Part 2 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | BR9020FV-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9040FV-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9080ARFVM-W | 8,192 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9020RFV-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9040RFV-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9020RFVM-W | 2,048 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |
| EEPROM | BR9040RFVM-W | 4,096 | — | — | — | ✓ | — | — | 2 MHz | 2.7 to 5.5 |

**Note to Table 10–10:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–11.** Seiko Instruments Inc. Device Characteristics (Part 1 of 3)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | 1-Wire | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | S-93C66B | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 5.5<br>Write: 2.7 ~ 5.5 |
| EEPROM | S-93C56B | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 2.0 ~ 5.5<br>Write: 2.4 ~5.5 |
| EEPROM | S-93C76A | 8,192 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 5.5<br>Write: 2.7 ~ 5.5 |
| EEPROM | S-93C66A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-93C56A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-29430A | 8,192 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 5.5<br>Write: 2.5 ~ 5.5 |
| EEPROM | S-29453A | 8,192 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 5.5<br>Write: 2.5 ~ 5.5 |
| EEPROM | S-29330A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29230A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29220A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29331A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29231A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29221A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29390A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |

**Table 10–11.** Seiko Instruments Inc. Device Characteristics (Part 2 of 3)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | 1-Wire | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | S-29290A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29391A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29291A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29394A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29294A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.5 ~ 6.5 |
| EEPROM | S-29355A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 V ~ 6.5 V<br>Write: 2.7 V ~ 6.5 V |
| EEPROM | S-29255A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | Read: 1.8 ~ 6.5<br>Write: 2.7 ~ 6.5 |
| EEPROM | S-29L330A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-29L220A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-29L331A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-29L221A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-29L394A | 4,096 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-29L294A | 2,048 | — | — | — | ✓ | — | — | 2.0 MHz | 1.8 to 5.5 |
| EEPROM | S-29U330A | 4,096 | — | — | — | ✓ | — | — | 500 kHz | Read: 0.9 ~ 3.6<br>Write:1.8 ~3.6 |
| EEPROM | S-29U220A | 2,048 | — | — | — | ✓ | — | — | 500 kHz | Read: 0.9 ~ 3.6<br>Write:1.8 ~3.6 |
| EEPROM | S-29U331A | 4,096 | — | — | — | ✓ | — | — | 500 kHz | Read: 0.9 ~ 3.6<br>Write:1.8 ~3.6 |
| EEPROM | S-29U221A | 2,048 | — | — | — | ✓ | — | — | 500 kHz | Read: 0.9 ~ 3.6<br>Write:1.8 ~3.6 |
| EEPROM | S-29U394A | 4,096 | — | — | — | ✓ | — | — | 500 kHz | Read: 0.9 ~ 3.6<br>Write:1.8 ~3.6 |
| EEPROM | S-29U294A | 2,048 | — | — | — | ✓ | — | — | 500 kHz | Read: 0.9 ~ 3.6<br>Write:1.8 ~3.6 |
| EEPROM | S-29Z330A | 4,096 | — | — | — | ✓ | — | — | 500 kHz | 0.9 to 3.6 |
| EEPROM | S-29ZX30A | 8,192 | — | — | — | ✓ | — | — | 500 kHz | 0.9 to 3.6 |
| EEPROM | S-24CS08A | 8,192 | — | — | ✓ | — | — | — | 400 kHz | Read: 1.8 to 5.5<br>Write: 2.7 to 5.5 |
| EEPROM | S-24CS04A | 4,096 | — | — | ✓ | — | — | — | 400 kHz | Read: 1.8 ~ 5.5<br>Write: 2.7 ~ 5.5 |
| EEPROM | S-24CS02A | 2,048 | — | — | ✓ | — | — | — | 400 kHz | Read: 1.8 ~ 5.5<br>Write: 2.7 ~ 5.5 |
| EEPROM | S-24C08A | 8,192 | — | — | ✓ | — | — | — | 400 kHz | Read: 1.8 ~ 5.5<br>Write: 2.7 ~ 5.5 |

**Table 10–11.** Seiko Instruments Inc. Device Characteristics (Part 3 of 3)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | 1-Wire | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | S-24C04A | 4,096 | — | — | ✓ | — | — | — | 100 kHz | Read: 1.8 ~ 5.5<br>Write: 2.5 ~ 5.5 |
| EEPROM | S-24C02A | 2,048 | — | — | ✓ | — | — | — | 100 kHz | Read: 1.8 ~ 5.5<br>Write: 2.5 ~ 5.5 |
| EEPROM | S-24C04B | 4,096 | — | — | ✓ | — | — | — | 400 KHz | 2.0 to 5.5 |
| EEPROM | S-24C02B | 2,048 | — | — | ✓ | — | — | — | 400 KHz | 2.0 to 5.5 |

**Note to Table 10–11:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–12.** STMicroelectronics Device Characteristics (Part 1 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | M24C04-W | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | M24C02-W | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | M24C08-W | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 2.5 to 5.5 |
| EEPROM | M24C04-L | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 2.2 to 5.5 |
| EEPROM | M24C02-L | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 2.2 to 5.5 |
| EEPROM | M24C08-L | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 2.2 to 5.5 |
| EEPROM | M24C04-R | 4,096 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | M24C02-R | 2,048 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | M24C08-R | 8,192 | — | — | — | — | ✓ | — | 400 kHz | 1.8 to 5.5 |
| EEPROM | ST24W04 | 4,096 | — | — | — | — | ✓ | — | 100 kHz | 3.0 to 5.5 |
| EEPROM | ST25W04 | 4,096 | — | — | — | — | ✓ | — | 100 kHz | 2.5 to 5.5 |
| EEPROM | ST24C04 | 4,096 | — | — | — | — | ✓ | — | 100 kHz | 3.0 to 5.5 |
| EEPROM | ST25C04 | 4,096 | — | — | — | — | ✓ | — | 100 kHz | 2.5 to 5.5 |
| EEPROM | M93C76-W | 8192 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 5.5 |
| EEPROM | M93C66-W | 4,096 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 5.5 |
| EEPROM | M93C56-W | 2,048 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 5.5 |
| EEPROM | M93C76-R | 8,192 | — | — | — | — | — | ✓ | 2 MHz | 1.8 to 5.5 |
| EEPROM | M93C66-R | 4,096 | — | — | — | — | — | ✓ | 2 MHz | 1.8 to 5.5 |
| EEPROM | M93C56-R | 2,048 | — | — | — | — | — | ✓ | 2 MHz | 1.8 to 5.5 |
| EEPROM | M93S66-W | 4,096 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 5.5 |
| EEPROM | M93S56-W | 2,048 | — | — | — | — | — | ✓ | 2 MHz | 2.5 to 5.5 |
| EEPROM | M93S66-R | 4,096 | — | — | — | — | — | ✓ | 2 MHz | 1.8 to 5.5 |
| EEPROM | M93S56-R | 2,048 | — | — | — | — | — | ✓ | 2 MHz | 1.8 to 5.5 |
| EEPROM | M95080-W | 8,192 | — | ✓ | — | — | — | — | 10 MHz | 2.5 to 5.5 |
| EEPROM | M95040-W | 4,096 | — | ✓ | — | — | — | — | 5 MHz | 2.5 to 5.5 |
| EEPROM | M95020-W | 2,048 | — | ✓ | — | — | — | — | 5 MHz | 2.5 to 5.5 |

**Table 10–12.** STMicroelectronics Device Characteristics (Part 2 of 2)

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | SPI | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | M95080-R | 8,192 | — | ✓ | — | — | — | — | 10 MHz | 1.8 to 5.5 |
| EEPROM | M95040-S | 4,096 | — | ✓ | — | — | — | — | 5 MHz | 1.8 to 3.6 |
| EEPROM | M95020-S | 2,048 | — | ✓ | — | — | — | — | 5 MHz | 1.8 to 3,6 |

**Note to Table 10–12:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

**Table 10–13.** Toshiba Corporation Device Characteristics

| Type | Device | Size (Bits) | Interface | | | | | | $f_{MAX}$ | Operating Voltage (V) *(1)* |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | SCI | 4-Wire | 2-Wire | 3-Wire | I²C | Microwire | | |
| EEPROM | TC9WMA2FK | 2,048 | — | ✓ | — | ✓ | — | — | 1 MHz | Read: 1.8 ~ 5.5<br>Write: 2.3 ~ 5.5 |
| EEPROM | TC9WMB2FK | 2,048 | — | — | — | — | ✓ | — | 400 kHz | Read: 1.8 ~ 5.5<br>Write: 2.3 ~ 5.5 |

**Note to Table 10–13:**

(1) The MAX II device supports two different $V_{CCINT}$ of operating voltage ranges, which are 2.375 to 2.625 V, and 3.0 to 3.6 V; the MAX IIG device supports the 1.71 to 1.89 V operating voltage range.

# Conclusion

MAX II devices can be used to incorporate logic and memory devices on a design board, eliminating chip-to-chip delays, minimizing board space, and reducing total system cost. Since you can program the UFM block to suit your needs, MAX II devices offer more interface flexibility than an off-the-shelf EEPROM device.

# Referenced Documents

This chapter references the following documents:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *Using User Flash Memory in MAX II Devices* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 10–14 shows the revision history for this chapter.

**Table 10–14.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.5 | ■ Updated New Document Format. | — |
| December 2007, version 1.4 | ■ Added "Referenced Documents" section. | — |
| December 2006, version 1.3 | ■ Added document revision history. | — |
| January 2005, version 1.2 | ■ Previously published as Chapter 11. No changes to content. | — |
| December 2004, version 1.1 | ■ Updated text to Design Considerations section. | — |

# Section IV. In-System Programmability

This section provides information and guidelines for in-system programmability (ISP) and Joint Test Action Group (JTAG) boundary scan testing (BST).

This section includes the following chapters:

- Chapter 11, In-System Programmability Guidelines for MAX II Devices
- Chapter 12, Real-Time ISP and ISP Clamp for MAX II Devices
- Chapter 13, IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices
- Chapter 14, Using Jam STAPL for ISP via an Embedded Processor
- Chapter 15, Using the Agilent 3070 Tester for In-System Programming

## Revision History

Refer to each chapter for its own specific revision history. For information about when each chapter was updated, refer to the Chapter Revision Dates section, which appears in the complete handbook.

# 11. In-System Programmability Guidelines for MAX II Devices

MII51013-1.7

## Introduction

As time-to-market pressure increases, design engineers require advanced system-level products to ensure problem-free development and manufacturing. Programmable logic devices (PLDs) with in-system programmability (ISP) can help accelerate development time, facilitate in-field upgrades, simplify the manufacturing flow, lower inventory costs, and improve printed circuit board (PCB) testing capabilities. Altera® ISP-capable MAX® II devices can be programmed and reprogrammed in-system via the IEEE Std. 1149.1 Joint Test Action Group (JTAG) interface. This interface allows MAX II devices to be programmed and the PCB to be functionally tested in a single manufacturing step, saving testing time and assembly costs. This chapter describes guidelines you should follow to design successfully with ISP, including:


- "General ISP Guidelines" on page 11–1
- "IEEE Std. 1149.1 Signals" on page 11–4
- "Sequential versus Concurrent Programming" on page 11–6
- "ISP Troubleshooting Guidelines" on page 11–7
- "ISP via Embedded Processors" on page 11–9
- "ISP via In-Circuit Testers" on page 11–10


## General ISP Guidelines

This section provides guidelines that help you design successfully for ISP-capable MAX II devices. These guidelines should be used regardless of your specific design implementation.

### Operating Conditions

Each MAX II device has several parametric ratings, or operating conditions, that are required for proper operation. Although MAX II devices can exceed these conditions when in user mode and still operate correctly, these conditions should not be exceeded during in-system programming. Violating any of the operating conditions during in-system programming can result in programming failures or incorrectly programmed devices. $V_{CCIO}$ of all I/O banks and $V_{CCINT}$ of the device must be fully powered up for ISP to function.

#### ISP Voltage

The $V_{CCINT}$ and $V_{CCIO}$ level specified in the device operating conditions table must be maintained on the $V_{CCINT}$ and $V_{CCIO}$ pins during in-system programming to ensure that the device's flash cells are programmed correctly. The $V_{CCINT}$ and $V_{CCIO}$ specification applies for both commercial- and industrial-temperature-grade devices.

### Input Voltages

The *MAX II Device Family Data Sheet* lists the MAX II device input voltage specification in the absolute maximum ratings and the recommended operating conditions tables. The input voltages in the absolute maximum rating table refers to the maximum voltage which the device can tolerate before risking permanent damage.

The recommended operating conditions table specify the voltage range for safe device operation. Make sure all pins that transition during in-system programming do not have a ground or $V_{CC}$ overshoot. Overshoot problems typically occur on free-running clocks or data buses that can toggle during in-system programming. All pins that have an overshoot greater than 1.0 V must have series termination.

For more information about the recommended operating conditions and the absolute maximum ratings for MAX II devices and termination, refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* and *AN 75: High-Speed Board Designs*, respectively.

## UFM Operations During In-System Programming

If your design allows you to access the MAX II UFM (write or erase), you must ensure that all the erase or write operations of the UFM are completed before starting any ISP session (including stand-alone verify, examine, setting security bit, and reading the contents of the UFM). You should never start an ISP session when any erase or write operation of the UFM is on going, as this may put the device in an unrecoverable state. However, this restriction does not apply to the read operation of the UFM.

If you cannot ensure that any erase or write operation of the UFM is complete before attempting an ISP operation to the MAX II device, then you should enable the real-time ISP feature. When used properly, this feature can help guard against any UFM/ISP operation contention. When real-time ISP is enabled, the programming algorithm used by the Quartus® II software or the Jam™ (**.jam**)/Jam Byte-Code (**.jbc**) files will wait 500 ms before it begins any operation. This is the same amount of time it takes for one UFM sector to be erased (that is, the real-time ISP programming algorithm waits for what may have been a previously started UFM erase sequence to complete).

However, if you are using a real-time ISP feature, no other UFM operations are allowed after that time (no address shifting, no data shifting, and no read, write, or erase operations). This can be controlled by monitoring the RTP_BUSY signal on the altufm_none megafunction. When real-time ISP is under way, the RTP_BUSY output signal on the UFM block goes high. You can monitor this signal and ensure that all UFM operations from the logic array cease until real-time ISP is complete. This user-generated control logic is only necessary for the altufm_none megafunction, which provides no auto-generated logic. The other interfaces for the altufm megafunction (altufm_parallel, altufm_spi, altufm_i2c) contain control logic that automatically monitors the RTP_BUSY signal and ceases operations to the UFM when a real-time ISP operation is under way.

## Interrupting In-System Programming

Altera does not recommend interrupting the programming process. However, the MAX II device has an ISP_DONE bit that will only be set at the very end of a successful program sequence. The I/O pins will only drive out if this bit is set. This prevents a partially programmed device from driving out and operating unpredictably.

## MultiVolt Devices and Power-Up Sequences

For the JTAG circuitry to operate correctly during in-system programming or boundary-scan testing, all devices in a JTAG chain must be in the same state. Therefore, in systems with multiple power supply voltages, the JTAG pins must be held in the test-logic-reset state until all devices in the chain are completely powered up. This procedure is particularly important because systems with multiple power supplies cannot power all voltage levels simultaneously.

MAX II devices have the MultiVolt feature and can use more than one power supply voltage: $V_{CCINT}$ and $V_{CCIO}$ for each I/O bank. $V_{CCINT}$ provides power to the JTAG circuitry; $V_{CCIO}$ provides power to input pins and output drivers for output pins, including TDO. Therefore, when using two power supply voltages, the JTAG circuitry must be held in the test-logic-reset state until both power supplies are turned on. If the JTAG pins are not held in the test-logic-reset state, in-system programming errors can occur.

### $V_{CCIO}$ Powered before $V_{CCINT}$

If $V_{CCIO}$ is powered up before $V_{CCINT}$, the JTAG circuitry is not active but TDO is tri-stated. Even though the JTAG circuitry is not active, if the next device in the JTAG chain is powered up with the same trace as $V_{CCIO}$, its JTAG circuitry must stay in the test-logic-reset state. Because all TMS and TCK signals are common, they must be disabled for all devices in the chain. Therefore, the JTAG pins must be disabled by pulling TCK low and TMS high.

## I/O Pins Tri-Stated during In-System Programming

By default, all device I/O pins are tri-stated during in-system programming. In addition, the MAX II device provides a weak pull-up resistor during ISP. The purpose of this weak pull-up resistor is to eliminate the need for external pull-up resistors on tri-stated I/O pins.

For pins that are used to drive signals and require a particular value during in-system programming (for example, output enable or chip enable signals), you can use the in-system programming clamp feature or the real-time ISP feature available for MAX II devices. These two features ensure that each I/O pin is clamped to a specific state during in-system programming.

For more information, refer to the *In-System Programming Clamp* and *Real-Time ISP* sections in the *JTAG and In-System Programmability* chapter in the *MAX II Device Handbook*.

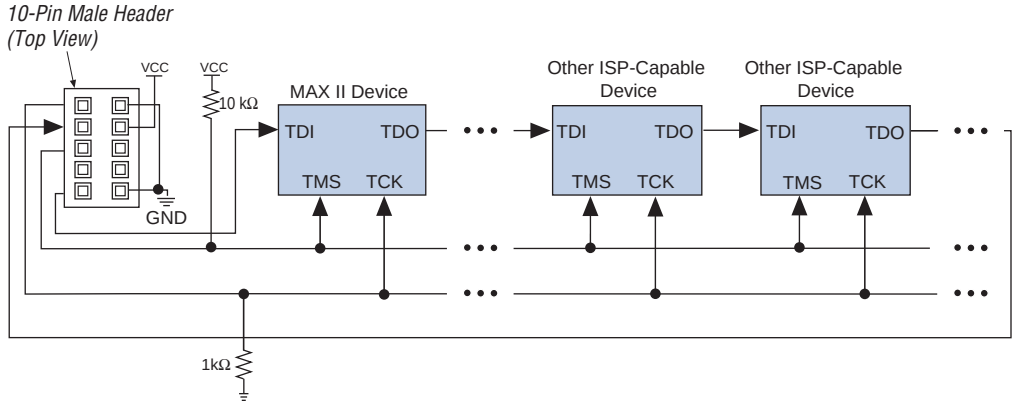
## Pull-Up and Pull-Down of JTAG Pins During In-System Programming

A MAX II device operating in in-system programming mode requires four pins: TDI, TDO, TMS, and TCK. The detailed description and function of each pin can be found in the *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*.

Three of the four JTAG pins have internal weak pull-up or pull-down resistors. The TDI and TMS pins have internal weak pull-up resistors while the TCK pin has an internal weak pull-down resistor. However, for device programming in a JTAG chain, there might be devices that do not have internal pull-up or pull-down resistors. Altera recommends to externally pull TMS high through 10-kΩ and TCK low through 1-kΩ resistors. Pulling-up the TDI signal externally for the MAX II device is optional.

Figure 11–1 shows the external pull-up and pull-down for TMS and TCK of the JTAG chain. The TDO pin does not have internal pull-up or pull-down resistors, and does not require external pull-up or pull-down resistors.

**Figure 11–1.** External Pull-Up and Pull-Down Resistors for TMS and TCK of a JTAG Chain




The TMS pin is pulled high so that the TAP controller will remain in the TEST_LOGIC/RESET state even if there is input from TCK. To prevent TCK from pulsing high, the TCK pin is pulled low during power-up. Pulling TCK high is not recommended because an increase in the power supply to the pull-up resistor causes the TCK to pulse high; thus, it is possible for the TAP controller to reach an unintended state.

# IEEE Std. 1149.1 Signals

This section provides guidelines for programming with the IEEE Std. 1149.1 (JTAG) interface.

## TCK Signal

Most in-system programming failures are caused by a noisy TCK signal. Noisy transitions on rising or falling edges can cause incorrect clocking of the IEEE Std. 1149.1 Test Access Port (TAP) controller. Incorrect clocking can cause the state machine to transition to an unknown state, leading to in-system programming failures.

Further, because the TCK signal must drive all IEEE Std. 1149.1 devices in the chain in parallel, the signal may have a high fan-out. Like any other high fan-out user-mode clock, you must manage a clock tree to maintain signal integrity. Typical errors that result from clock integrity problems are invalid ID messages, blank-check errors, or verification errors.

Altera recommends pulling the TCK signal low through the internal weak pull-down resistor or an external 1-kΩ resistor.

Fast TCK edges combined with board inductance can cause overshoot problems. When this combination occurs, you must either reduce inductance on the trace or reduce the switching rate by selecting a transistor-to-transistor logic (TTL) driver chip with a slower slew rate. Altera does not recommend using resistor and capacitor (RC) networks to slow down edge rates, because they can violate the device's input specifications. In most cases, using a driver chip prevents the edge rate from being too slow. Altera recommends using driver chips that do not glitch upon power-up.

## Programming via a Download Cable

You can program MAX II devices using a MasterBlaster™, ByteBlasterMV™, ByteBlaster™ II, or USB Blaster download cable. Using a PC or UNIX workstation with the Quartus II software programmer, Programmer Object File (**.pof**), JamTM Files (**.jam**), or Jam Byte-Code Files (**.jbc**) can be downloaded to the MAX II devices via the download cable.

If you are using the download cables and your JTAG chain contains three or more devices, Altera recommends adding a buffer to the chain. You should select a buffer with slow transitions to minimize noise, but be sure that the transition rates can still meet TCK performance requirements of your chain.

If you must extend the download cable, you can attach a standard PC parallel or USB port cable to the download cable. Do not extend the 10-pin header portion of the download cable; extending this portion of the cable can cause noise and in-system programming problems.

Different download cables will have different programming times. For more information about the MasterBlaster, ByteBlasterMV, ByteBlaster II, or USB Blaster download cable, refer to the *MasterBlaster Serial/USB Communications Cable User Guide*, *ByteBlasterMV Download Cable User Guide*, *ByteBlaster II Download Cable User Guide*, or *USB-Blaster Download Cable User Guide*.

## Disabling IEEE Std. 1149.1 Circuitry

By default, the JTAG circuitry in MAX II devices is always enabled because they have dedicated JTAG pins and circuitry. The JTAG circuitry must be enabled during ISP and boundary-scan testing, but disabled at all other times. If your design does not use ISP or boundary-scan test (BST) circuitry, Altera recommends disabling the IEEE Std. 1149.1 circuitry.

To disable the JTAG circuitry, Altera recommends pulling TMS high and TCK low. Pulling TCK low ensures that a rising edge does not occur on TCK during the power-up sequence. You can pull TCK high, but you must first pull TMS high. Pulling TMS high first ensures that the rising edge or edges on TCK do not cause the JTAG state machine to leave the test-logic-reset state.

For more information about disabling the IEEE 1149.1 circuitry, refer to the *Disabling IEEE Std. 1149.1 BST Circuitry* section of the *IEEE 1149.1 (JTAG) Boundary-Scan Testing* chapter in the *MAX II Device Handbook*.

### Working with Different Voltage Levels

When devices in a JTAG chain operate at different voltage levels, a device's output voltage specification must meet the subsequent device's input voltage specification. If the devices do not meet this criteria, you must add additional circuitry, such as a level-shifter, to adjust the voltage levels. For example, when a 5.0-V device drives a 2.5-V device, you must adjust the 5.0-V device's output voltage to meet the 2.5-V device's input voltage specification.

Because all devices in a JTAG chain are tied together, you must also ensure that the first device's `TDO` output meets the subsequent device's `TDI` input voltage specification to program a chain of devices successfully.

All MAX II devices include a MultiVolt I/O feature, which allows these devices to interface with systems that have different supply voltages. All MAX II devices can be set for 3.3-V, 2.5-V, 1.8-V, or 1.5-V I/O operation. The JTAG pins of MAX II devices support these voltage levels. Refer to the *MAX II Architecture* chapter in the *MAX II Device Handbook* for I/O standard compatibility for each $V_{CCIO}$ voltage. For example, $V_{CCIO1}$ at 3.3 V does not allow JTAG input pins to accept 1.8- or 1.5-V signals.

## Sequential versus Concurrent Programming

This section describes how to program multiple devices using sequential and concurrent programming. The JTAG chain setup for sequential and concurrent programming is similar, only the programming algorithms are different.

### Sequential Programming

Sequential programming is the process of programming multiple devices in a chain, one device at a time. After the first device in the chain is finished being programmed, the next device is programmed. This sequence continues until all specified devices in the JTAG chain are programmed. After a device is programmed, it will be in bypass mode to allow data to be passed to the subsequent devices in the chain. The devices in the chain do not go into user mode until all the devices are programmed.

### Concurrent Programming

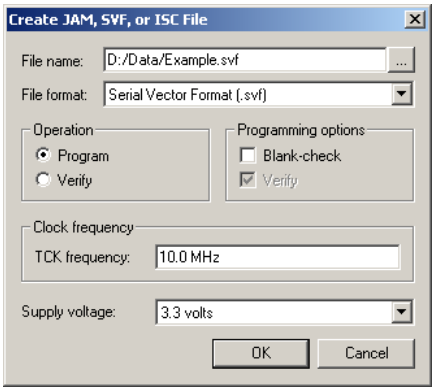
Concurrent programming is used to program devices from the same family (for example, the MAX II family) in parallel. The programming time is slightly longer than the time needed to program the largest device in the chain, resulting in considerably faster programming times than sequential programming (where programming time is equal to the sum of individual programming times for all devices). Higher clock rates for shifting data result in even greater time savings.

Concurrent programming of devices can be done using Serial Vector Format files (**.svf**), Jam files, or JBC files created from the Quartus II software. See Figure 11–2.

1. On the Tools menu, click **Programmer**.
2. Click **Add File** and select programming files for the respective devices.

3. On the File menu, point to **Create/Update** and click **Create JAM, SVF, or ISC File**.
4. Specify a file in the File format list.
5. Click **OK**.

**Figure 11–2.** Create JAM, SVF, or ISC File




# ISP Troubleshooting Guidelines

This section provides tips for troubleshooting ISP-related problems.

## Invalid ID and Unrecognized Device Messages

The first step during in-system programming is to check the device's silicon ID. If the silicon ID does not match, an `Invalid ID` or `Unrecognized Device` error is generated. Typical causes for this error are shown below:

- Download cable connected incorrectly
- `TDO` is not connected
- Incomplete JTAG chain
- Noisy `TCK` signal
- Jam Player ported incorrectly

### Download Cable Connected Incorrectly

You will receive an error if the download cable is connected incorrectly to the parallel or USB port or if it is not receiving power from your board.

For more information about installing the MasterBlaster, ByteBlasterMV, ByteBlaster, or USB Blaster download cable, refer to the *MasterBlaster Serial/USB Communications Cable User Guide*, *ByteBlasterMV Download Cable User Guide*, *ByteBlaster II Download Cable User Guide*, or *USB-Blaster Download Cable User Guide*.

### TDO Is Not Connected

You will receive an error if the TDO port of one device in the chain is not connected. During in-system programming, data must be shifted in and out of each device in the JTAG chain through the JTAG pins. Therefore, each device's TDO port must be connected to the subsequent device's TDI port, and the last device's TDO port must be connected to the download cable's TDO port.

### Incomplete JTAG Chain

You will receive an error if the JTAG chain is not complete. To check if an incomplete JTAG chain is causing the error, use an oscilloscope to monitor vectors coming out of each device in the chain. If each device's TDO port does not toggle during in-system programming, your JTAG chain is not complete.

### Noisy TCK Signal

Noise on the TCK signal is the most common reason for in-system programming errors. Noisy transitions on rising or falling edges can cause incorrect clocking of the IEEE Std. 1149.1 TAP controller, causing the state machine to be lost and in-system programming to fail. For more information about dealing with noisy TCK signals, refer to "TCK Signal" on page 11–4.

### Jam Player Ported Incorrectly

You will receive an error if the Jam Player was not ported correctly for your platform. To check if the Jam Player is causing the error, apply the IDCODE instruction to the target device using a Jam file. You can use a Jam file to load an IDCODE instruction and then shift out the IDCODE value. This test determines if the JTAG chain is set up correctly and if you can read and write to the JTAG chain properly.

You can download the **idcode.zip** file from the Altera website to obtain the **idcode.jam** file.

## Troubleshooting Tips

This section discusses some additional suggestions for troubleshooting ISP issues.

### Verify the JTAG Chain Continuity

For in-system programming to occur successfully, the number of devices physically in the JTAG chain must match the number reported in the Quartus II software. The following steps show one simple way to verify that the JTAG chain is connected properly.

1. Open the Programmer in the Quartus II software.
2. Click **Auto Detect** in the Programmer. The Quartus II software reports the number of devices found on the JTAG chain. If this fails, check the JTAG chain to make sure it is not broken.

### Check the $V_{cc}$ Level of the Board During In-System Programming

Using an oscilloscope, monitor the $V_{CCINT}$ signal on your JTAG chain and set the trigger to the minimum $V_{CC}$ level listed in the recommended operating conditions table of the appropriate device family data sheet. If a trigger occurs during in-system programming, the devices may need more current than is being supplied by the existing power supply. Try replacing the existing power supply with one that provides more current.

### Power-Up Problems

Excessive voltage or current on I/O pins during power-up can cause one of the devices in the JTAG chain to experience latch-up. Check if any of the devices are hot to the touch; hot devices have probably experienced latch-up and may have been damaged. In this situation, check all voltage sources to make sure that excessive voltage or current is not being fed into the device. Then, replace the affected device and try programming again.

### Random Signals on JTAG Pins

During normal operation, each device's TAP controller must be in the test-logic-reset state. To force the device back into this state, try pulling the `TMS` signal high and pulsing the `TCK` clock signal six times. If the device then powers-up successfully, you must add a higher pull-down resistor on the `TCK` signal.

### Software Issues

Failures during in-system programming may occasionally be related to the Quartus II software. Software-related issues are documented in the Find Answers section under the Support Center on the Altera website at www.altera.com. Search the database for information relating to software issues that interfere with in-system programming.

# ISP via Embedded Processors

This section provides guidelines for programming ISP-capable devices using the Jam Standard Test and Programming Language (STAPL) and an embedded processor.

## Processor and Memory Requirements

The Jam Byte-Code Player supports 8-bit and higher processors; the ASCII Jam Player supports 16-bit and higher processors. The Jam Player uses memory in a predictable manner, which simplifies in-field upgrades by confining updates to the Jam File. The Jam Player memory uses both ROM and dynamic memory (RAM). ROM is used to store the Jam Player binary and the Jam File; dynamic memory is used when the Jam Player is called.

For information about how to estimate the maximum amount of RAM and ROM required by the Jam Player, refer to the *Using Jam STAPL for ISP via an Embedded Processor* chapter in the *MAX II Device Handbook*.

### Porting the Jam Player

The Altera Jam Player (both Byte-Code and ASCII versions) works with a PC parallel port. To port the Jam Player to your processor, you only need to modify the **jamstub.c** or **jbistub.c** file (for the ASCII Jam Player or Jam Byte-Code Player, respectively). All other files should remain the same. If the Jam Player is ported incorrectly, an Unrecognized Device error is generated. The most common causes for this error are listed below:

- After porting the Jam Player, the `TDO` value may be read in reversed polarity. This problem may occur because the default I/O code in the Jam Player assumes the use of the PC parallel port.
- Although the `TMS` and `TDI` signals are clocked in on the rising edge of `TCK`, outputs do not change until the falling edge of `TCK`. This situation causes a half `TCK` clock cycle lag in reading out the values. If the `TDO` transition is expected on the rising edge, the data appears to be offset by one clock.
- Altera recommends using registers to synchronize the output transitions. In addition, some processor data ports use a register to synchronize the output signals. For example, reading and writing to the PC's parallel port is accomplished by reading and writing to registers. The use of these registers must be taken into consideration when reading and writing to the JTAG chain. Incorrect accounting of these registers can cause the values to either lead or lag the expected value.

## ISP via In-Circuit Testers

MAX II devices can also be in-system programmed via in-circuit testers. For more information about using Agilent's 3070 in-circuit tester to in-system program MAX II devices, refer to the *Using Jam STAPL for ISP via an Embedded Processor* chapter in the *MAX II Device Handbook*.

## Conclusion

The information provided in this document is based on development experiences and customer issues resolved by Altera. For more information about resolving in-system programming problems, contact Altera Applications.

## Referenced Documents

This chapter references the following documents:

- *AN 75: High-Speed Board Designs*
- *ByteBlasterMV Download Cable User Guide*
- *ByteBlaster II Download Cable User Guide*
- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*
- *JTAG and In-System Programmability* chapter in the *MAX II Device Handbook*
- *MasterBlaster Serial/USB Communications Cable User Guide*

- *USB-Blaster Download Cable User Guide*
- *Using Jam STAPL for ISP via an Embedded Processor* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 11–1 shows the revision history for this chapter.

**Table 11–1.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.7 | ■ Updated New Document Format. | — |
| December 2007, version 1.6 | ■ Updated "Pull-Up and Pull-Down of JTAG Pins During In-System Programming" section.<br>■ Added "Referenced Documents" section. | External pull-up for `TDI` is optional. |
| December 2006, version 1.5 | ■ Added document revision history. | — |
| August 2006, version 1.4 | ■ Corrected Figure 11–1. | — |
| January 2005, version 1.3 | ■ Previously published as Chapter 12. No changes to content. | — |
| December 2004, version 1.2 | ■ Added section User Flash Memory Operations During In-System Programming. | — |
| June 2004, version 1.1 | ■ Pull-up resistor values. Textual updates. | — |

# 12. Real-Time ISP and ISP Clamp for MAX II Devices

MII51019-1.6

## Introduction

During in-system programming, most CPLDs automatically tri-state their input/output (I/O) pins to prevent contention issues on a board. After successful programming, the device enters user mode and the new design begins to function. Apart from this normal programming mode, MAX® II devices also support real-time in-system programmability (ISP) and ISP Clamp programming modes, which allow control of I/O and device behavior during ISP. This chapter describes the following two features and how to use them in the Quartus® II software, as well as the Jam™ Standard Test and Programming Language (STAPL) and Jam STAPL Byte-Code Players:


- "Real-Time ISP" on page 12–1
- "ISP Clamp" on page 12–4

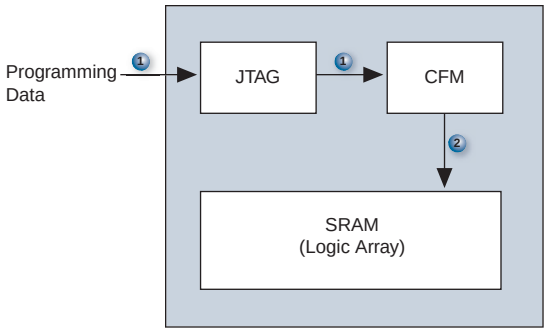

## Real-Time ISP

Real-time ISP allows you to program a MAX II device while the device is still in operation. The new design only replaces the existing design when there is a power cycle to the device (i.e., powering down and powering up again). This feature enables you to perform in-field updates to the MAX II device at any time without affecting the operation of the whole system.

### How Real-Time ISP Works

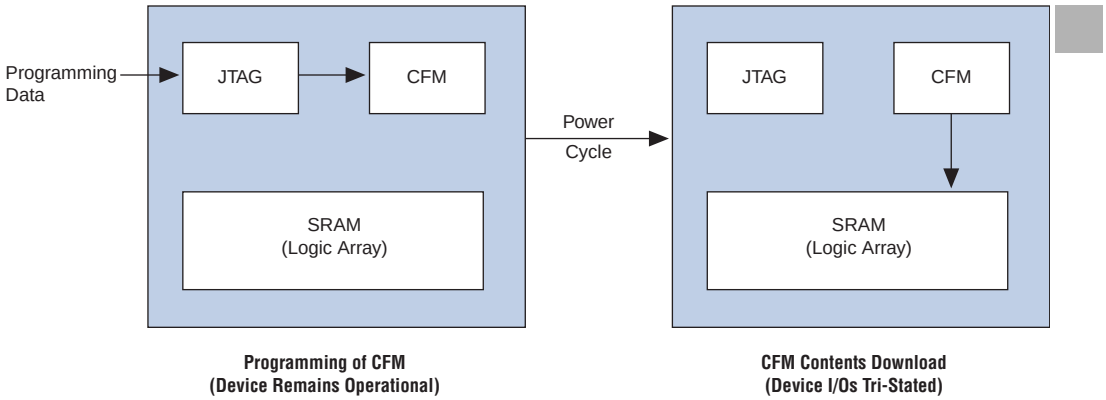
For normal ISP operation, downloading the new design data from the configuration flash memory (CFM) to the SRAM begins after the completion of CFM programming. During the process of CFM programming and subsequent downloading of CFM data to SRAM, I/O pins will remain tri-stated. After the CFM download to the SRAM, the device resets and enters user mode operation. Figure 12–1 shows the flow of normal programming.

**Figure 12–1.** MAX II Device with Normal ISP Operation



In real-time ISP mode, the user flash memory (UFM), programmable logic, and I/O pins remain operational while programming of the CFM is in progress. The contents of the CFM will not download into the SRAM after the successful programming of the CFM. Instead, the device waits for a power cycle to occur. The normal power-up sequence occurs (CFM downloads to SRAM at power-up) and the device enters user mode after $t_{CONFIG}$ time. Figure 12–2 shows the flow of real-time ISP.

**Figure 12–2.** Real-Time ISP Operation



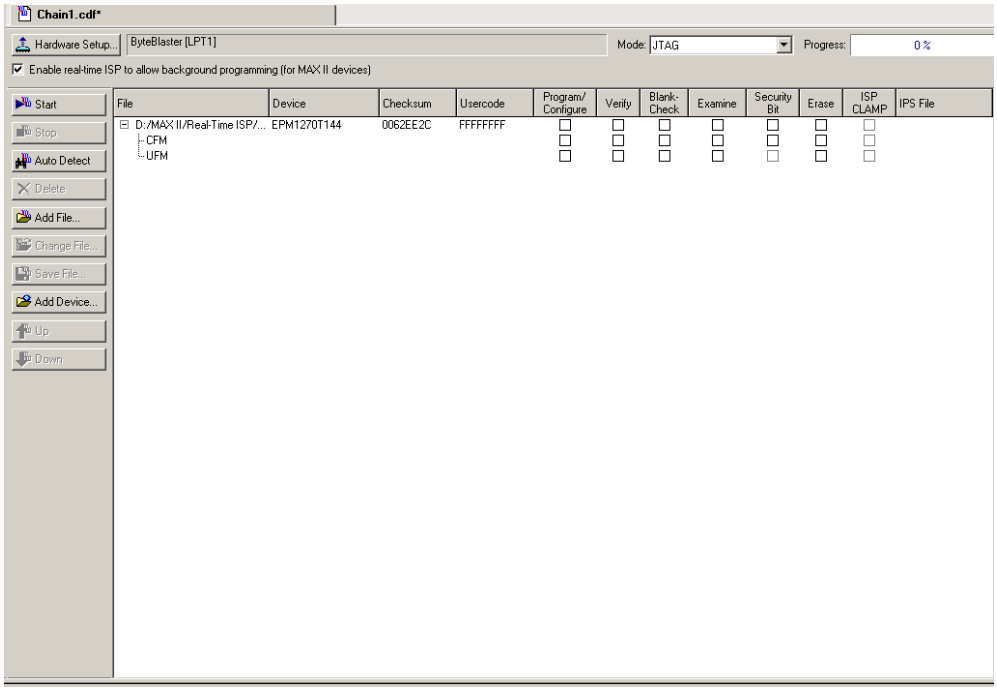

For the $t_{CONFIG}$ value for a specific MAX II device, refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*.

## Real-Time ISP with the Quartus II Software

The programming file formats generated by the Quartus II software that support these two features are the Programmer Object File (**.pof**) that is used with the Quartus II programmer, and the Jam File (**.jam)** and Jam Byte-Code File (**.jbc**) that are used with either the Quartus II programmer or other programming tools.

Ensure that you enable this feature before programming a MAX II device through the Quartus II programmer. You can enable the real-time ISP feature by selecting the **Enable real-time ISP to allow background programming (for MAX II devices)** option from the Quartus II programmer window. See Figure 12–3.

**Figure 12–3.** Real-Time ISP Option in the Quartus II Programmer Window

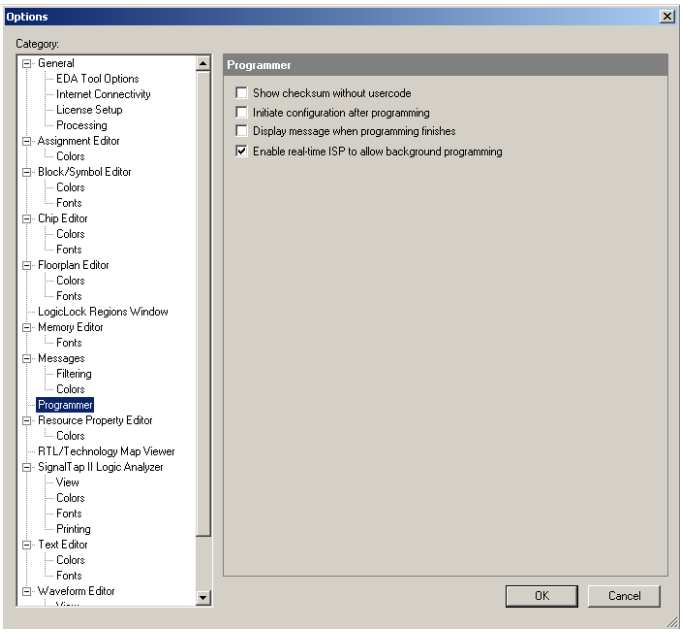



You can also enable the real-time ISP feature in the Quartus II software through the following steps:

1. On the Tools menu, click **Options**.
2. Under **Category**, select **Programmer**.
3. Turn on **Enable real-time ISP to allow background programming (for MAX II devices)** and click **OK**. The MAX II device will go into real-time ISP mode when the Quartus II programmer starts programming it with any one of the three types of programming files.

Figure 12–4 shows the Programmer options in the Options menu.

**Figure 12–4.** Programmer Options in the Options Menu




## Real-Time ISP with Jam and JBC Players

You can use the Jam or JBC file created from the POF to program a MAX II device in real-time ISP mode with the Jam or JBC Player.

For real-time ISP with the Jam File and Jam Player, type the following at the command-line prompt:

```
jp_23 -aprogram -ddo_real_time_isp=1 <file_name.jam>
```

For Real-Time ISP with the JBC File and JBC Player, type the following at the command-line prompt:

```
jbi_22 -aprogram -ddo_real_time_isp=1 <file_name.jbc>
```

The names of the executable files for the players are different, depending on the version of the players. Download the latest version of the Jam and JBC Player from the Altera® web site at www.altera.com.

# ISP Clamp

When a MAX II device enters normal ISP operation, all the I/O pins tri-state and are weakly pulled up to $V_{CCIO}$ with internal pull-up resistors. However, there are situations when the I/O pins of the device should not be tri-stated when the device is in ISP operation. For instance, in a running system, some signals (e.g., output enable or chip enable signals) might use some of the I/O pins and require those I/O pins to assume a high or low logic level, or even maintain their current state when the device is in ISP mode.

With the ISP clamp feature in MAX II devices, you can hold each I/O pin of a device to a specified static state when programming the device. You can set the state in the Quartus II software. After successfully programming the device in ISP clamp mode, those I/O pins will be released and function according to the new design.

This feature can be used to indicate when the device is being programmed and when the programming is done by setting a particular pin to a specific state (different from the state when the device is in user mode) when the device enters ISP clamp mode.

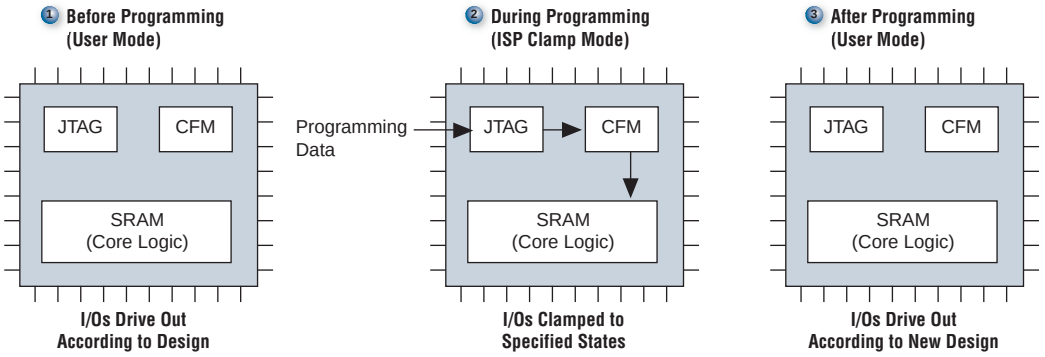
## How ISP Clamp Works

When the ISP clamp feature is used, you can set the I/O pins to tri-state (default), high, low, or even sample the existing state of a pin and hold the pin to that state when the device is in ISP clamp operation. The software determines the values to be scanned into the boundary-scan registers of each I/O pin, based on your settings. This will determine the state of the pins to be clamped to when the device programming is in progress. The weak I/O pull-up resistors are disabled during programming when the ISP clamp feature is used, even if the I/O is clamped to a tri-state value.

Before clamping the I/O pins, the SAMPLE/PRELOAD JTAG instruction is first executed to load the appropriate values to the boundary-scan registers. After loading the boundary-scan registers with the appropriate values, the EXTEST instruction is executed to clamp the I/O pins to the specific values loaded into the boundary-scan registers during SAMPLE/PRELOAD.

If you choose to sample the existing state of a pin and hold the pin to that state when the device enters ISP clamp mode, you must make sure that the signal is in steady state. You need a steady state signal because you cannot control the sample set-up time as it depends on the TCK frequency as well as the download cable and software. You might not capture the correct value when sampling a signal that toggles or is not static for long periods of time. Figure 12–5 shows the ISP clamp operation.

**Figure 12–5.** ISP Clamp Operation




## Using ISP Clamp in the Quartus II Software

You have to define the states of the I/O pins to use the ISP clamp feature. There are two ways to define the pin states in the Quartus II software. You can either:

- Use an I/O Pin State file (**.ips**), or
- Use the Assignment Editor to set the clamp states of the pins

## Using the IPS File

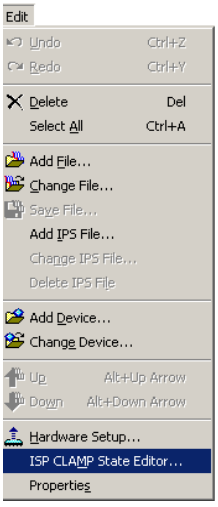
### Creating an IPS File

You can specify the clamp states of the pins when the device is in ISP clamp operation without configuring the settings in the Assignment Editor and recompiling the design. You must first create a new I/O pin state file (**.ips**) file and define the states of the pins in the file, or use an existing IPS file. The IPS file defines the states for all the pins of the device when the device is in ISP clamp operation. The file created is usable for programming the device with any designs, as long as it targets the same device and package. An IPS file must be used together with a POF file, which contains the programming data to program the device.

To create an IPS file, perform the following:

1. Click **Programmer** on the toolbar, or on the Tools menu, click **Programmer** to open the Quartus II Programmer window.
2. Click **Add File** in the programmer to add the programming file (POF, Jam, or JBC) into the programmer window.
3. Click on the programming file in the programmer (the entire row will be highlighted) and on the Edit menu, click **ISP CLAMP State Editor**. See Figure 12–6.

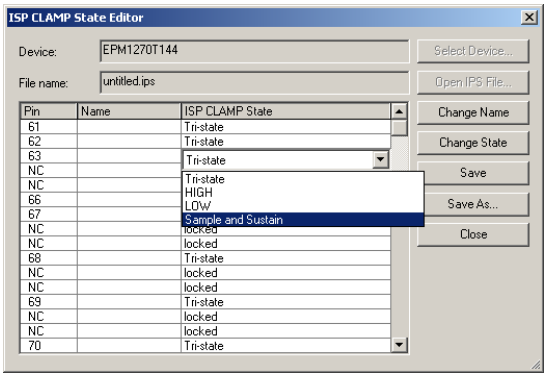
**Figure 12–6.** Edit Menu




4. Specify the states of the pins in your design in the ISP Clamp State Editor. There are four clamp state choices: tri-state, high, low, and sample and sustain. By default, all pins are set to tri-state.
5. Save the IPS file after making the modifications.

Figure 12–7 shows the ISP Clamp State Editor. On the File menu, you can also click **Create/Update** > **Create/Update IPS File** to open the ISP Clamp State Editor and create a new IPS file.

**Figure 12–7.** ISP Clamp State Editor



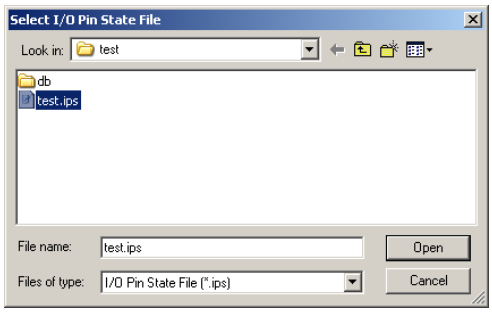

### Using the IPS File

In the Quartus II Programmer, you must specify the IPS file you want to use by performing the following steps:

1. Double-click on the cell under the IPS File column. The **Select I/O Pin State File** menu appears.
2. Choose the IPS file for your project and click **Open**.

You can also left-click on the programming file (this will highlight the entire row) and on the Edit menu, click **Add IPS File** to open the Select I/O Pin State File dialog box as shown in Figure 12–8.

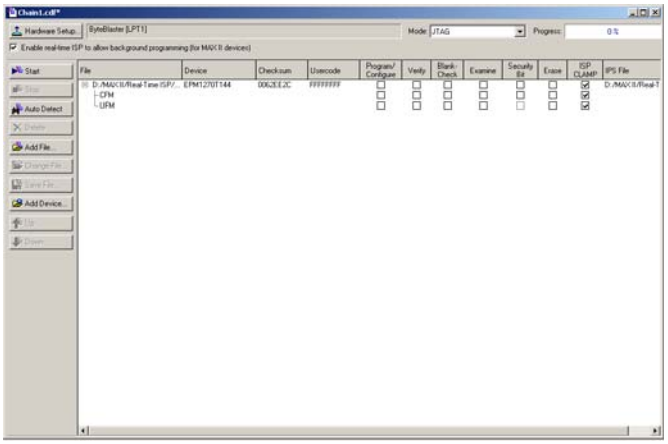
**Figure 12–8.** Select I/O Pin State File Menu




1. The IPS file you have selected will be listed in the Quartus II Programmer window, as shown in Figure 12–9.

Make sure the ISP CLAMP check box is checked before you start programming your device.

**Figure 12–9.** The Quartus II Programmer Window with the Specific IPS File



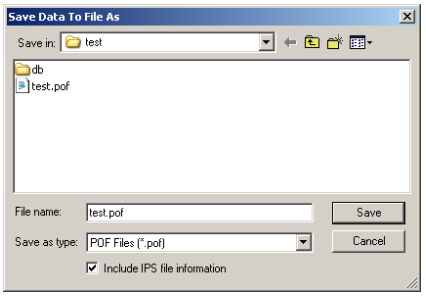
### Saving the IPS File Information to the Programming File

The pin state information in the IPS file can be saved into the POF to avoid requiring two files. You will only need the programming file to program a device in ISP clamp mode. This programming file is also used for creating the Jam and JBC files for the ISP clamp so that the Jam or JBC files will contain the pin state information. The following are the steps to save the pin state information from the IPS file to the programming files.

1. Add in the programming file in the programmer window.
2. Add in the IPS file to the programmer.
3. Click **Save File** in the programmer window or on the Edit menu, and the **Save Data To File** As dialog box will appear. See Figure 12–10.
4. Enter the file name, check the **Include IPS file information** box, and click **Save**.

The POF with saved IPS information only supports ISP clamp operation in the Quartus II software and not with third-party programming tools. For third-party tools, Jam or JBC files should be used if ISP clamp is required.

**Figure 12–10.** Save Data To File as Menu




When programming a device with the ISP Clamp box checked, the Quartus II Programmer will first look for the IPS file. If the IPS file is not found, only then it will look into the POF for the pin state information.

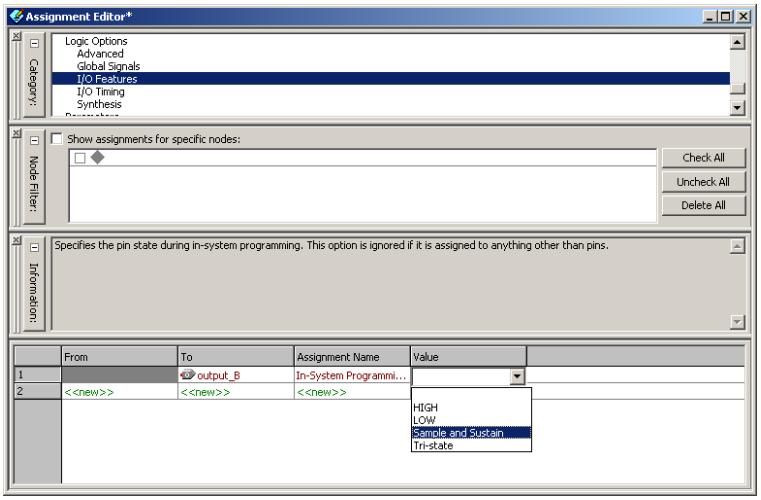
### Defining the Pin States in Assignment Editor

Another way to define the pin states is through the Assignment Editor. After you have defined the pin states in the Assignment Editor and compile the design, the programming file generated will have all the pin state information in it. The following are the assignment editor states:

1. Click **Start Analysis and Synthesis** on the toolbar.
2. On the Assignments menu, click **Assignment Editor**.
3. In the Assignment Editor, under **Category**, select **I/O Features**.
4. List down all the pins you wish to clamp when the device is in ISP clamp mode under the **To** column. You can use the Node Finder to help you select the pins.
5. Select **In-System Programming Clamp State** for all the pins under Assignment Name after you have listed down the pins you wish to set state values.
6. Define the states for each of the pins under Value. You can also choose to clamp the pins to high, low, tri-state, or sample and sustain the pin state. By default, the pins are tri-stated when the device enters ISP clamp mode.

Figure 12–11 shows how to define the states of the pins in the Assignment Editor.

**Figure 12–11.** Assignment Editor



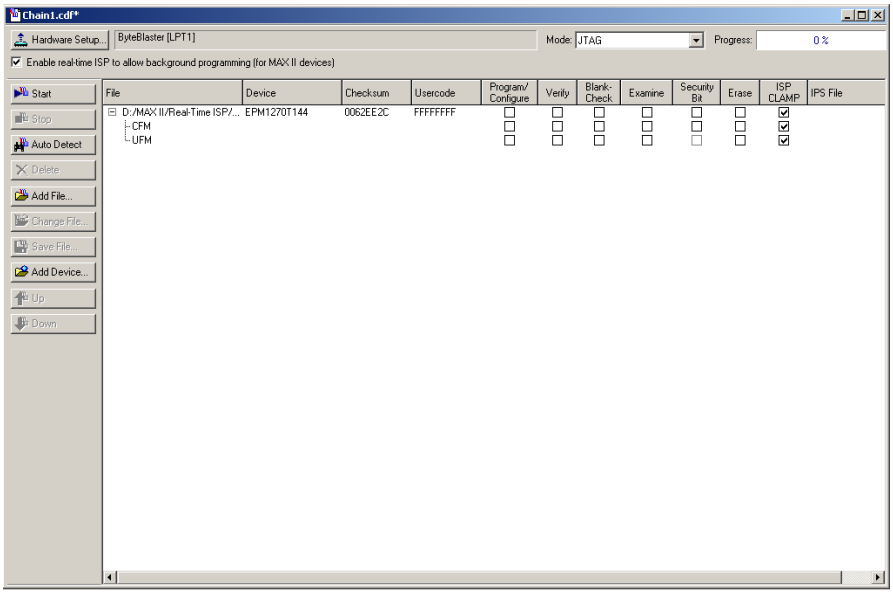

1. Save the assignments and recompile your design.

After you have recompiled the design, the ISP clamp state information will be stored in the POF. You can also view the settings in the Quartus II Settings File (**.qsf**).

#### Running ISP Clamp in the Quartus II Programmer

In the Quartus II Programmer window, make sure that the ISP Clamp check box is checked before programming the device. Do not add any IPS file in the programmer as the programmer will use the values specified in the IPS file instead of the values you set in the Assignment Editor (which is stored in the POF). Figure 12–12 shows the **Quartus II Programmer** window with ISP Clamp checkbox. Jam and JBC files created using the POF will have the pin state information in them.

**Figure 12–12.** Quartus II Programmer Window with ISP Clamp Checkbox




## ISP Clamp with Jam/JBC Files

The Jam or JBC files used for ISP clamp should contain all the pin state information and do not need any IPS file. Always use the POF file with pin state information to create the Jam or JBC files. The pin state information can be stored into the POF through the Assignment Editor or saving the pin state information to the POF as mentions earlier. The Jam or JBC files can be used with the Quartus II programmer, or with the Jam or JBC player, respectively.

# Conclusion

With the real-time ISP and ISP clamp features in MAX II devices, you can set the I/O pins of a device to certain states while programming the device. Through real-time ISP, you can program a MAX II device at any time without affecting the functionality of your system. The ISP clamp feature allows you to hold the I/O pins of a device to specific states when programming the device.

# Referenced Documents

This chapter references the following document:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 12–1 shows the revision history for this chapter.

**Table 12–1.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.6 | ■ Updated New Document Format. | — |
| December 2007, version 1.5 | ■ Added "Referenced Documents" section. | — |
| December 2006, version 1.4 | ■ Added document revision history. | — |
| February 2006, version 1.3 | ■ Updated the "Real-Time ISP with the Quartus II Software" section.<br>■ Added Figure 12–3.<br>■ Updated Figure 12–9 and 12–10. | — |
| June 2005, version 1.2 | ■ Updated the first paragraph of the How ISP Clamp Works section. | — |
| January 2005, version 1.1 | ■ Previously published as Chapter 13. No changes to content. | — |

# 13. IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices

MII51014-1.7

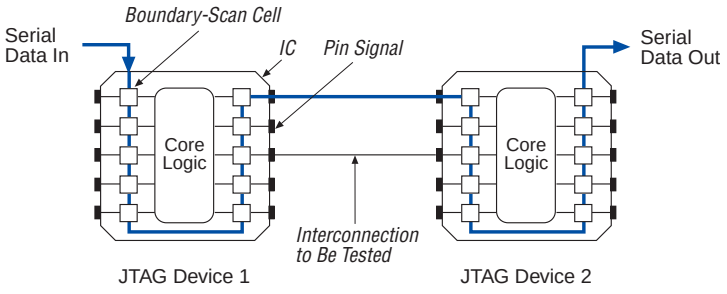
## Introduction

As printed circuit boards (PCBs) become more complex, the need for thorough testing becomes increasingly important. Advances in surface-mount packaging and PCB manufacturing have resulted in smaller boards, making traditional test methods (for example, external test probes and "bed-of-nails" test fixtures) harder to implement. As a result, cost savings from PCB space reductions are sometimes offset by cost increases in traditional testing methods.

In the 1980s, the Joint Test Action Group (JTAG) developed a specification for boundary-scan testing that was later standardized as the IEEE Std. 1149.1 specification. This boundary-scan test (BST) architecture offers the capability to efficiently test components on PCBs with tight lead spacing.

This BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. Boundary-scan cells in a device can force signals onto pins, or capture data from pin or core logic signals. Forced test data is serially shifted into the boundary-scan cells. Captured data is serially shifted out and externally compared to expected results. Figure 13–1 shows the concept of boundary-scan testing.

**Figure 13–1.** IEEE Std. 1149.1 Boundary-Scan Testing




This chapter discusses how to use the IEEE Std. 1149.1 BST circuitry in MAX® II devices. The topics are as follows:


- "IEEE Std. 1149.1 BST Architecture" on page 13–2
- "IEEE Std. 1149.1 Boundary-Scan Register" on page 13–3
- "IEEE Std. 1149.1 BST Operation Control" on page 13–6
- "I/O Voltage Support in JTAG Chain" on page 13–15
- "BST for Programmed Devices" on page 13–15
- "Disabling IEEE Std. 1149.1 BST Circuitry" on page 13–16
- "Guidelines for IEEE Std. 1149.1 Boundary-Scan Testing" on page 13–16
- "Boundary-Scan Description Language (BSDL) Support" on page 13–17

In addition to BST, you can use the IEEE Std. 1149.1 controller for in-system programming for MAX II devices. MAX II devices support IEEE 1532 programming, which utilizes the IEEE Std. 1149.1 Test Access Port (TAP) interface. However, this chapter only discusses the BST feature of the IEEE Std. 1149.1 circuitry.

# IEEE Std. 1149.1 BST Architecture

A MAX II device operating in IEEE Std. 1149.1 BST mode uses four required pins, TDI, TDO, TMS, and TCK. Table 13–1 summarizes the functions of each of these pins. MAX II devices do not have a TRST pin.

**Table 13–1.** EEE Std. 1149.1 Pin Descriptions

| Pin | Description | Function |
|---|---|---|
| TDI *(1)* | Test data input | Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK. |
| TDO | Test data output | Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data is not being shifted out of the device. |
| TMS *(1)* | Test mode select | Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur at the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK. |
| TCK *(2)* | Test clock input | The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge. |

**Notes to Table 13–1:**

(1) The TDI and TMS pins have internal weak pull-up resistors.

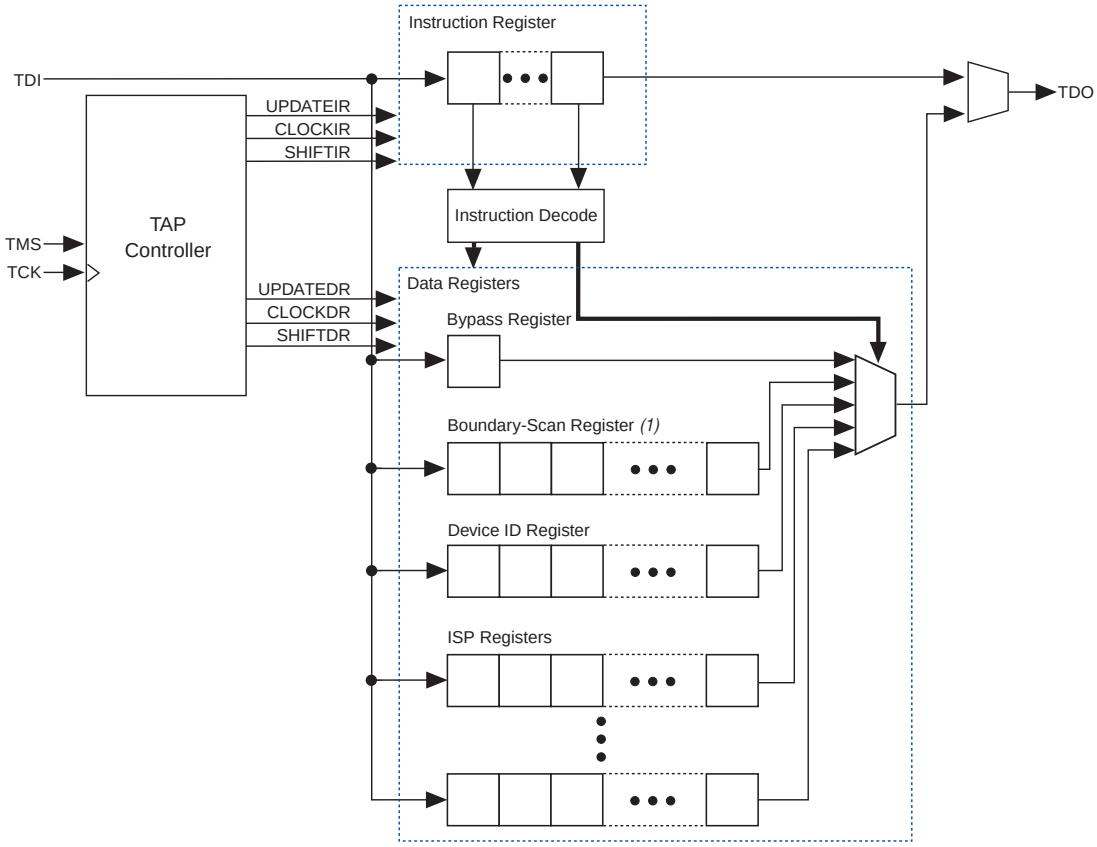
(2) The TCK pin has an internal weak pull-down resistor.

The IEEE Std. 1149.1 BST circuitry requires the following registers:

- The instruction register, which is used to determine the action to be performed and the data register to be accessed.
- The bypass register, which is a 1-bit-long data register used to provide a minimum-length serial path between TDI and TDO.
- The boundary-scan register that is a shift register composed of all the boundary-scan cells of the device.

Figure 13–2 shows a functional model of the IEEE Std. 1149.1 circuitry.

**Figure 13–2.** IEEE Std. 1149.1 Circuitry




**Note to Figure 13–2:**

(1) Refer to the *JTAG and In-System Programmability* chapter in the *MAX II Device Handbook* for the boundary-scan register length in MAX II devices.

IEEE Std. 1149.1 boundary-scan testing is controlled by a TAP controller, which is described in "IEEE Std. 1149.1 BST Operation Control" on page 13–6. The TMS and TCK pins operate the TAP controller, and the TDI and TDO pins provide the serial path for the data registers. The TDI pin also provides data to the instruction register, which then generates control logic for the data registers.

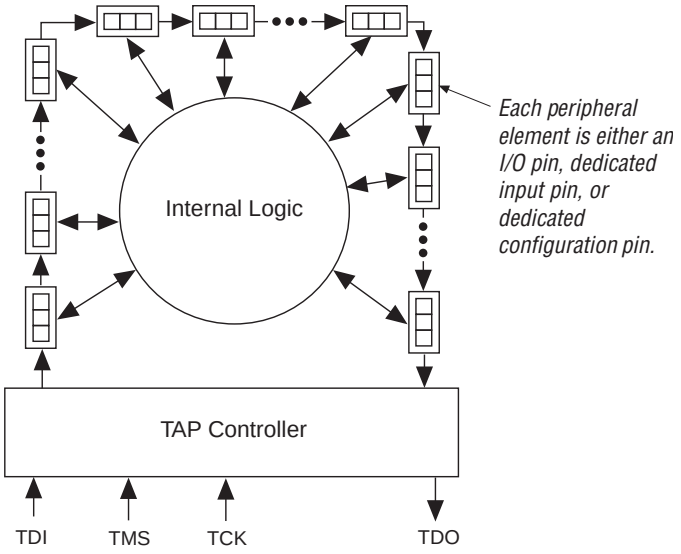
# IEEE Std. 1149.1 Boundary-Scan Register

The boundary-scan register is a large serial shift register that uses the TDI pin as an input and the TDO pin as an output. The boundary-scan register consists of 3-bit peripheral elements that are associated with I/O pins of the MAX II devices. You can use the boundary-scan register to test external pin connections or to capture internal data.

Refer to *the JTAG and In-System Programmability* chapter in the *MAX II Device Handbook* for the boundary-scan register length of MAX II devices.

Figure 13–3 shows how test data is serially shifted around the periphery of the IEEE Std. 1149.1 device.

**Figure 13–3.** Boundary-Scan Register




## Boundary-Scan Cells of a MAX II Device I/O Pin

Except for the four JTAG pins and power pins, all pins of a MAX II device (including clock pins) can be used as user I/O pins and have a boundary-scan cell (BSC). The 3-bit BSC consists of a set of capture registers and a set of update registers. The capture registers can connect to internal device data via the `OUTJ` and `OEJ` signals, while the update registers connect to external data through the `PIN_OUT` and `PIN_OE` signals. The global control signals for the IEEE Std. 1149.1 BST registers (for example, `SHIFT`, `CLOCK`, and `UPDATE`) are generated internally by the TAP controller; the `MODE` signal is generated by a decode of the instruction register. The data signal path for the boundary-scan register runs from the serial data in (SDI) signal to the serial data out (SDO) signal. The scan register begins at the `TDI` pin and ends at the `TDO` pin of the device.

Figure 13–4 shows the User I/O Boundary-Scan Cell of MAX II devices.

**Figure 13–4.** MAX II Device's User I/O BSC with IEEE Std. 1149.1 BST Circuitry

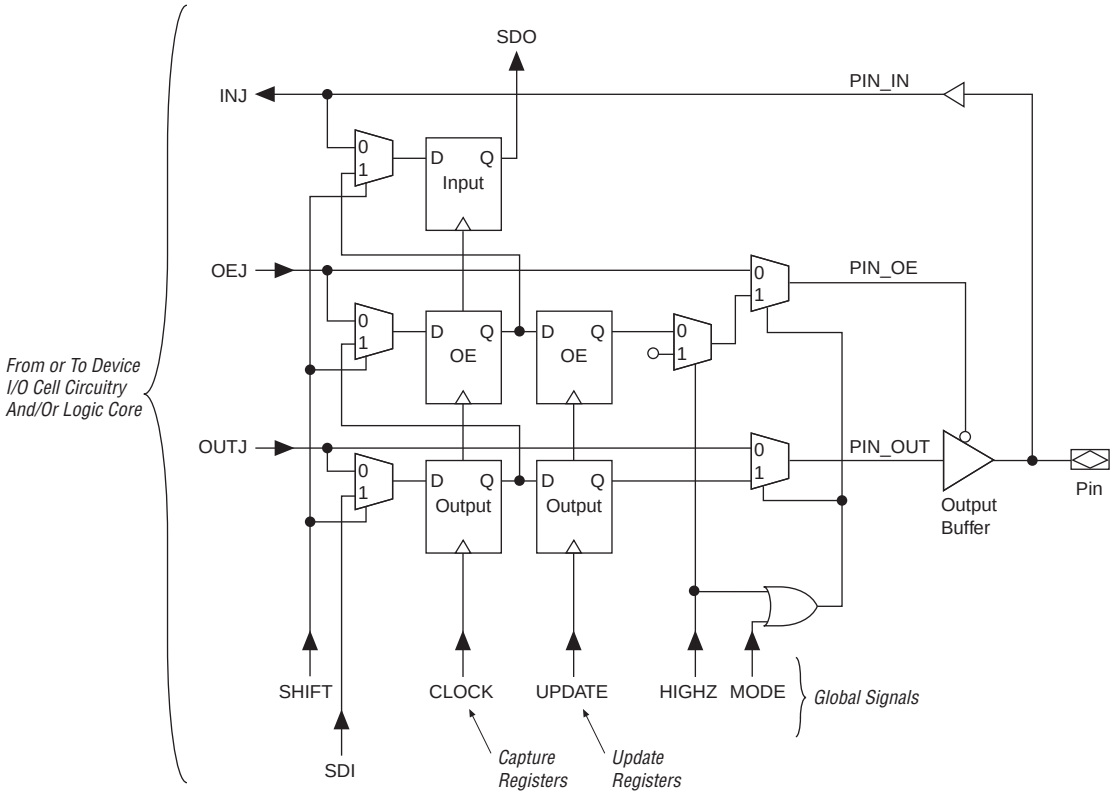



Table 13–2 describes the capture and update register capabilities of all boundary-scan cells within MAX II devices.

**Table 13–2.** MAX II Device's Boundary-Scan Cell Descriptions *(Note 1)*

| Pin Type | Captures | | | Drives | | | Notes |
|---|---|---|---|---|---|---|---|
| | Output Capture Register | OE Capture Register | Input Capture Register | Output Update Register | OE Update Register | Input Update Register | |
| User I/O | OUTJ | OEJ | PIN_IN | PIN_OUT | PIN_OE | — | Includes User Clocks |

**Note to Table 13–2:**

(1) TDI, TDO, TMS, and TCK pins, and all VCC and GND pin types do not have boundary-scan cells.

## JTAG Pins and Power Pins

MAX II devices do not have boundary-scan cells for the dedicated JTAG pins (TDI, TDO, TMS, and TCK) and power pins (VCCINT, VCCIO, GNDINT, and GNDIO).

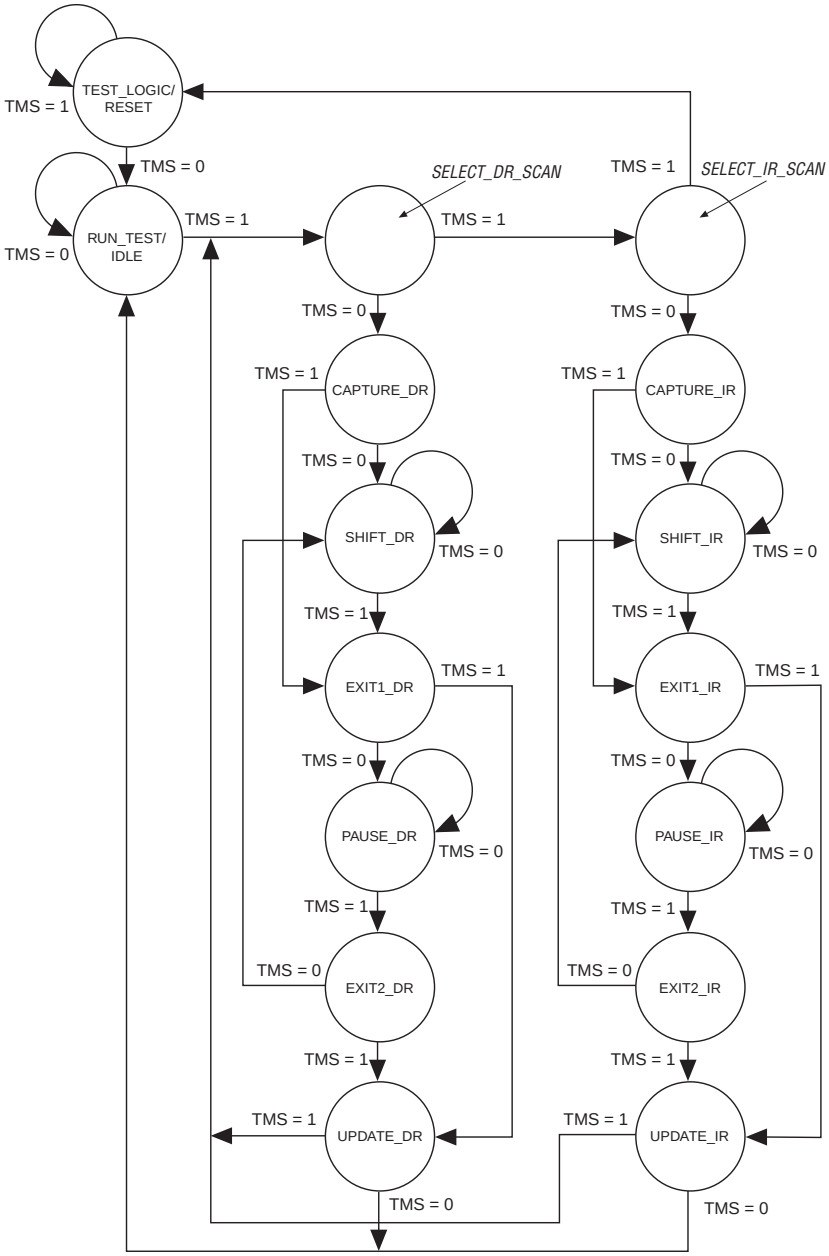
# IEEE Std. 1149.1 BST Operation Control

MAX II devices implement the following IEEE Std. 1149.1 BST instructions: SAMPLE/PRELOAD, EXTEST, BYPASS, IDCODE, USERCODE, CLAMP, and HIGHZ. The length of the BST instructions is 10 bits. These instructions are described in detail later in this chapter.

Refer to *the JTAG and In-System Programmability* chapter in the *MAX II Device Handbook* for a summary of the BST instructions and their instruction codes.

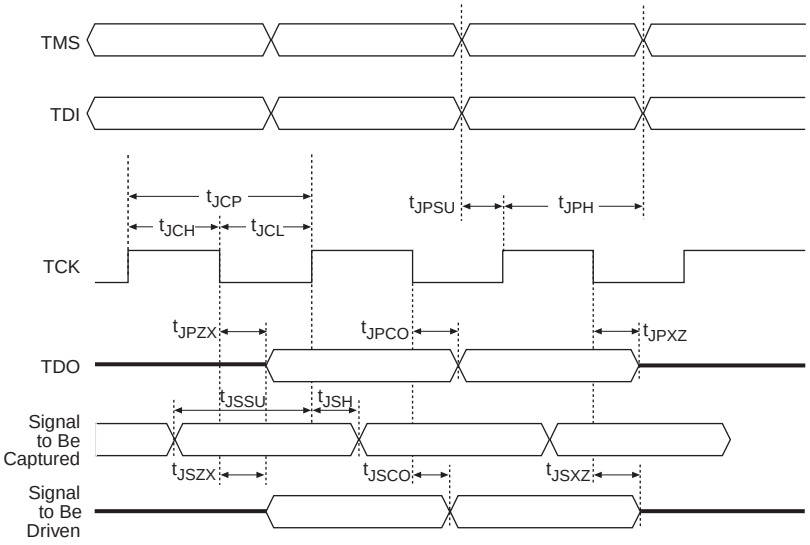
The IEEE Std. 1149.1 TAP controller, a 16-state state machine clocked on the rising edge of TCK, uses the TMS pin to control IEEE Std. 1149.1 operation in the device. Figure 13–5 shows the TAP controller state machine.

**Figure 13–5.** IEEE Std. 1149.1 TAP Controller State Machine




When the TAP controller is in the TEST_LOGIC/RESET state, the BST circuitry is disabled, the device is in normal operation, and the instruction register is initialized with IDCODE as the initial instruction. At device power-up, the TAP controller starts in this TEST_LOGIC/RESET state. In addition, the TAP controller may be forced to the TEST_LOGIC/RESET state by holding TMS high for five TCK clock cycles. Once in the TEST_LOGIC/RESET state, the TAP controller remains in this state as long as TMS continues to be held high while TCK is clocked. Figure 13–6 shows the timing requirements for the IEEE Std. 1149.1 signals.

**Figure 13–6.** IEEE Std. 1149.1 Timing Waveforms *(Note 1)*

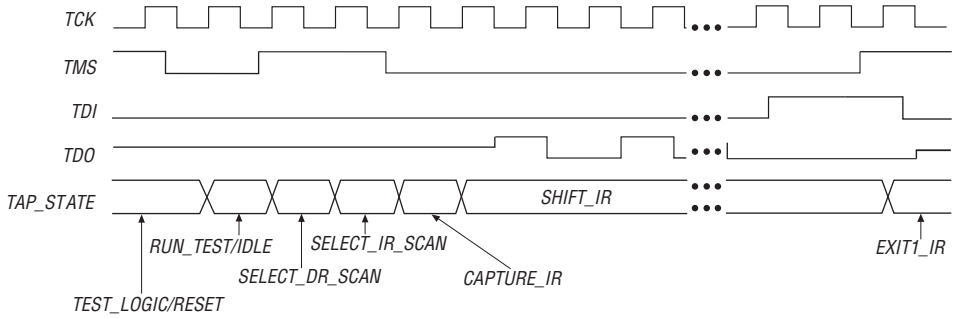



**Note to Figure 13–6:**

(1) For timing parameter values, refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*.

To start IEEE Std. 1149.1 operation, select an instruction mode by advancing the TAP controller to the shift instruction register (SHIFT_IR) state and shift in the appropriate instruction code on the TDI pin. The waveform diagram in Figure 13–7 represents the entry of the instruction code into the instruction register. It shows the values of TCK, TMS, TDI, and TDO and the states of the TAP controller. From the RESET state, TMS is clocked with the pattern 01100 to advance the TAP controller to SHIFT_IR.

**Figure 13–7.** Selecting the Instruction Mode




The TDO pin is tri-stated in all states except the SHIFT_IR and SHIFT_DR states. The TDO pin is activated at the first falling edge of TCK after entering either of the shift states and is tri-stated at the first falling edge of TCK after leaving either of the shift states.

When the SHIFT_IR state is activated, TDO is no longer tri-stated, and the initial state of the instruction register is shifted out on the falling edge of TCK. TDO continues to shift out the contents of the instruction register as long as the SHIFT_IR state is active. The TAP controller remains in the SHIFT_IR state as long as TMS remains low.

During the SHIFT_IR state, an instruction code is entered by shifting data on the TDI pin on the rising edge of TCK. The last bit of the opcode must be clocked at the same time that the next state, EXIT1_IR, is activated; EXIT1_IR is entered by clocking a logic high on TMS. Once in the EXIT1_IR state, TDO becomes tri-stated again. TDO is always tri-stated except in the SHIFT_IR and SHIFT_DR states. After an instruction code is entered correctly, the TAP controller advances to perform the serial shifting of test data in one of three modes—SAMPLE/PRELOAD, EXTEST, or BYPASS—that are described below.

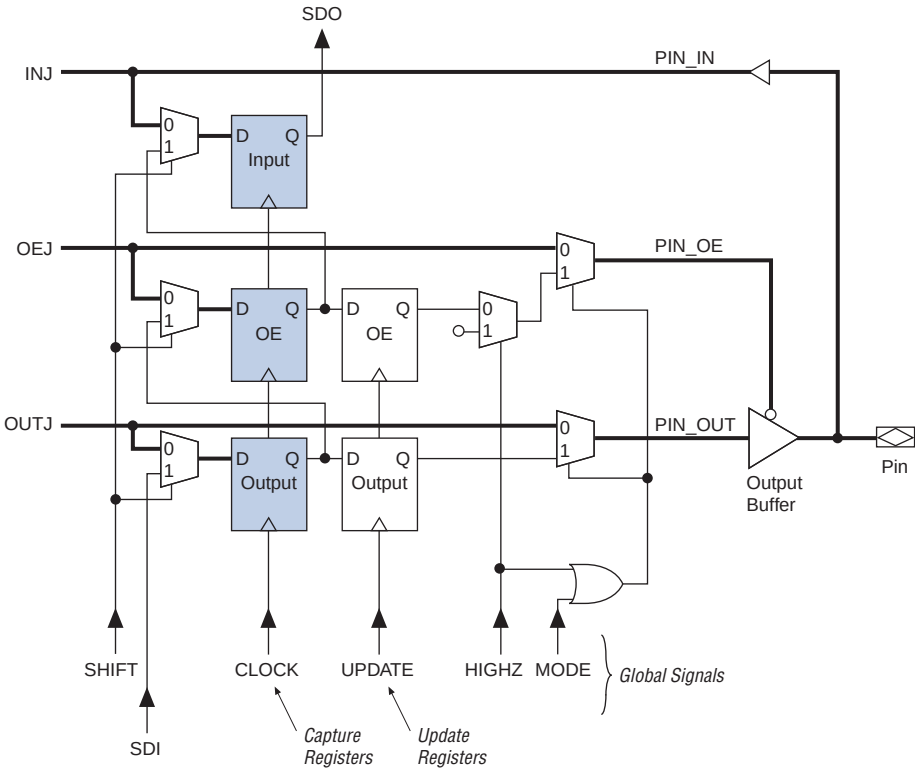
For MAX II devices, there are weak pull-up resistors for TDI and TMS, and pull-down resistors for TCK. However, in a JTAG chain, there might be some devices that do not have internal pull-up or pull-down resistors. In this case, Altera recommends pulling the TMS pin high (through an external 10-kΩ resistor), and pulling TCK low (through an external 1-kΩ resistor) during BST or in-system programmability (ISP) to prevent the TAP controller from going into an unintended state. Pulling-up the TDI signal externally for the MAX II device is optional.

For more information about the pull-up and pull-down resistors, refer to the *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*.

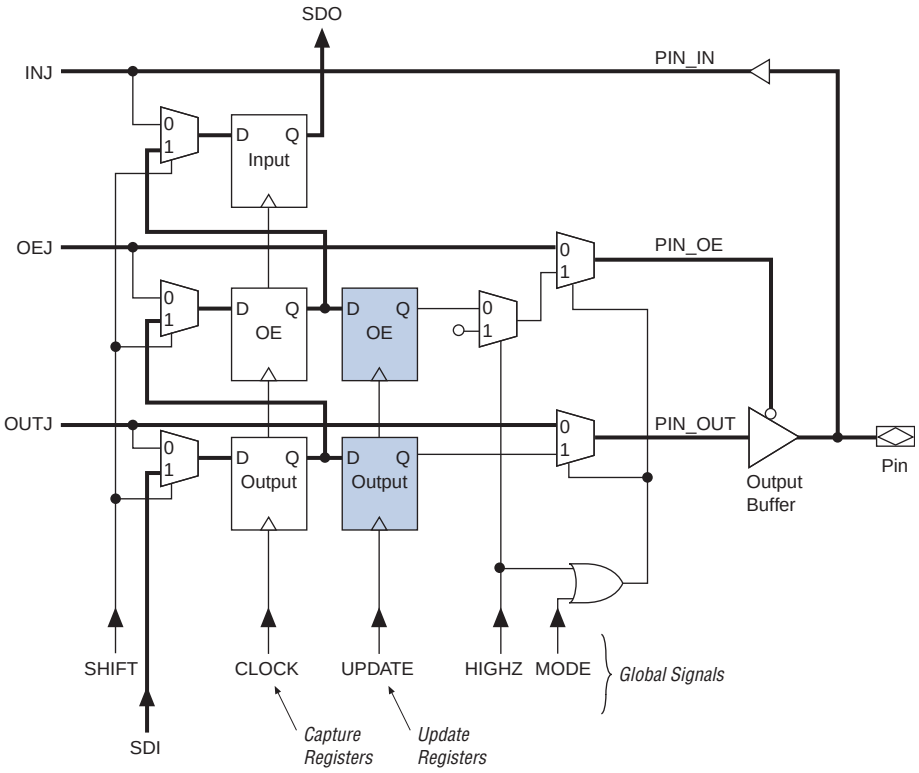
## SAMPLE/PRELOAD Instruction Mode

The SAMPLE/PRELOAD instruction mode allows you to take a snapshot of device data without interrupting normal device operation. However, this instruction mode is most often used to preload the test data into the update registers prior to loading the EXTEST instruction. Figure 13–8 shows the capture, shift, and update phases of the SAMPLE/PRELOAD mode.

**Figure 13–8.** IEEE Std. 1149.1 BST SAMPLE/PRELOAD Mode




**(Capture Phase)**




**(Shift and Update Phase)**

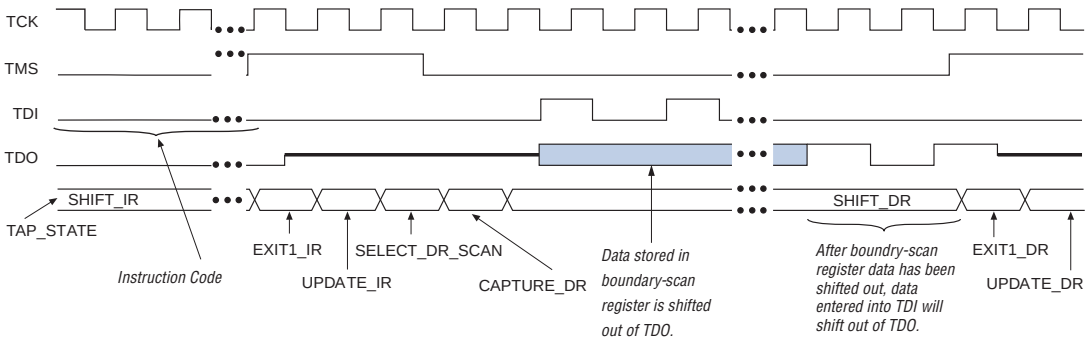
During the capture phase, multiplexers preceding the capture registers select the active device data signals; this data is then clocked into the capture registers. The multiplexers at the outputs of the update registers also select active device data to prevent functional interruptions to the device. During the shift phase, the boundary-scan shift register is formed by clocking data through capture registers around the device periphery and then out of the TDO pin. New test data can simultaneously be shifted into TDI and replace the contents of the capture registers. During the update phase, data in the capture registers is transferred to the update registers. This data can then be used in the EXTEST instruction mode.

Refer to "EXTEST Instruction Mode" on page 13–11 for more information.

Figure 13–9 shows the SAMPLE/PRELOAD waveforms. The SAMPLE/PRELOAD instruction code is shifted in through the TDI pin. The TAP controller advances to the CAPTURE_DR state and then to the SHIFT_DR state, where it remains if TMS is held low. The data shifted out of the TDO pin consists of the data that was present in the capture registers after the capture phase. New test data shifted into the TDI pin appears at the TDO pin after being clocked through the entire boundary-scan register. Figure 13–9 shows that the test data that shifted into TDI does not appear at the TDO pin until after the capture register data that is shifted out. If TMS is held high on two consecutive TCK clock cycles, the TAP controller advances to the UPDATE_DR state for the update phase.

If the device output enable feature is enabled but the DEV_OE pin is not asserted during boundary-scan testing, the OE boundary-scan registers of the boundary-scan cells capture data from the core of the device during SAMPLE/PRELOAD. These values are not high impedance, although the I/O pins are tri-stated.

**Figure 13–9.** SAMPLE/PRELOAD Shift Data Register Waveforms

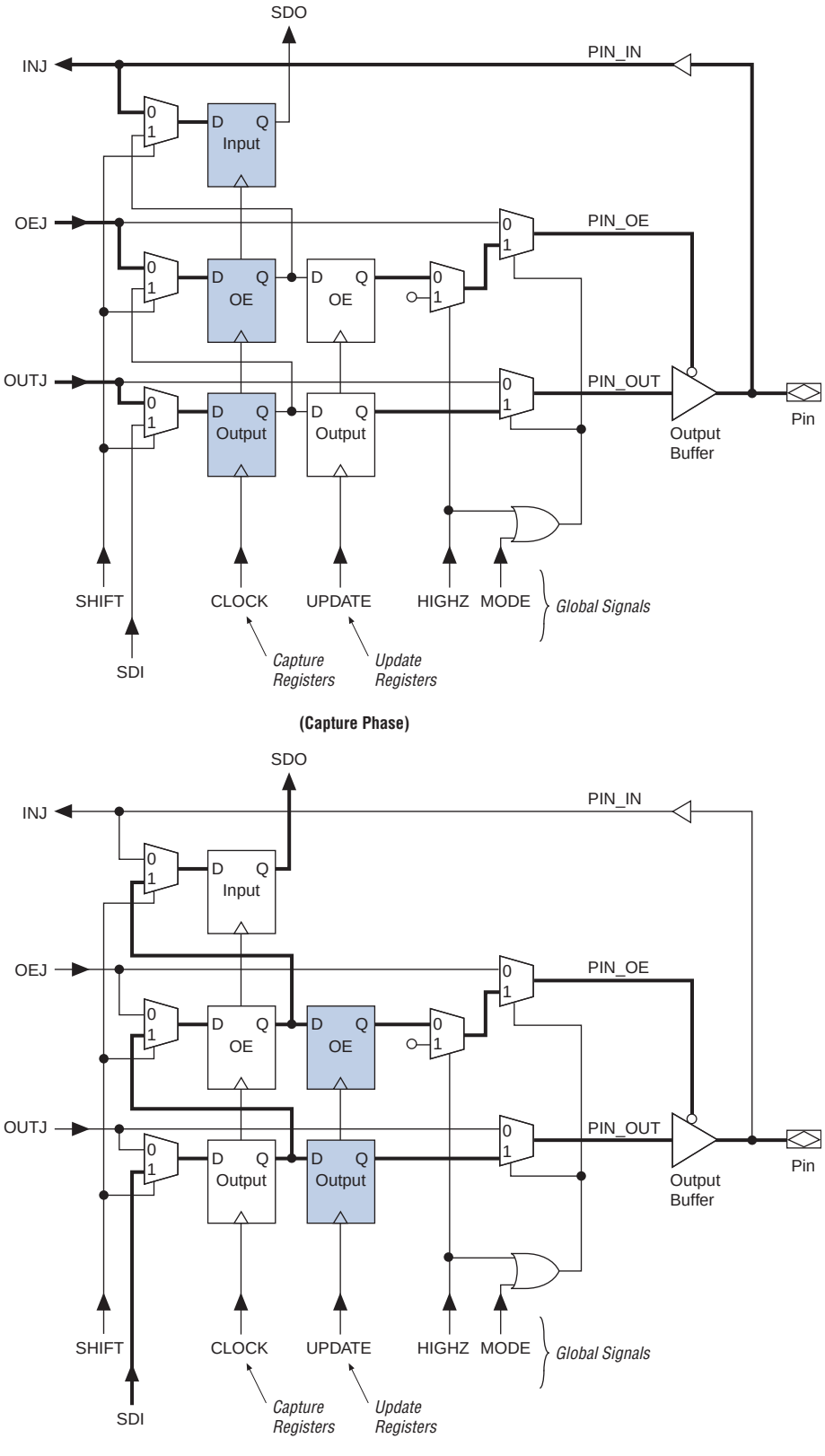



## EXTEST Instruction Mode

The EXTEST instruction mode is used primarily to check external pin connections between devices. Unlike the SAMPLE/PRELOAD mode, EXTEST allows test data to be forced onto the pin signals. By forcing known logic high and low levels on output pins, opens and shorts can be detected at pins of any device in the scan chain.

Figure 13–10 shows the capture, shift, and update phases of the EXTEST mode.

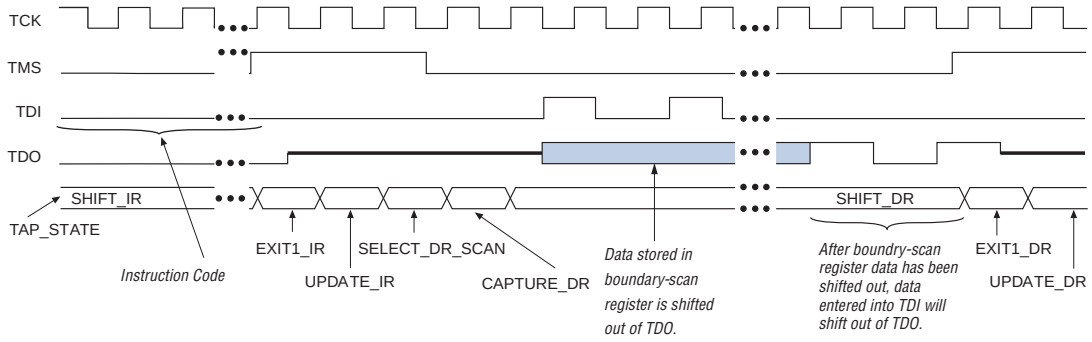
**Figure 13–10.** IEEE Std. 1149.1 BST EXTEST Mode



EXTEST selects data differently than SAMPLE/PRELOAD. EXTEST chooses data from the update registers as the source of the output and output enable signals. Once the EXTEST instruction code is entered, the multiplexers select the update register data; thus, data stored in these registers from a previous EXTEST or SAMPLE/PRELOAD test cycle can be forced onto the pin signals. In the capture phase, the results of this test data are stored in the capture registers and then shifted out of TDO during the shift phase. New test data can then be stored in the update registers during the update phase.

The waveform diagram in Figure 13–11 resembles the SAMPLE/PRELOAD waveform diagram, except that the instruction code for EXTEST is different. The data shifted out of TDO consists of the data that was present in the capture registers after the capture phase. New test data shifted into the TDI pin appears at the TDO pin after being clocked through the entire boundary-scan register.

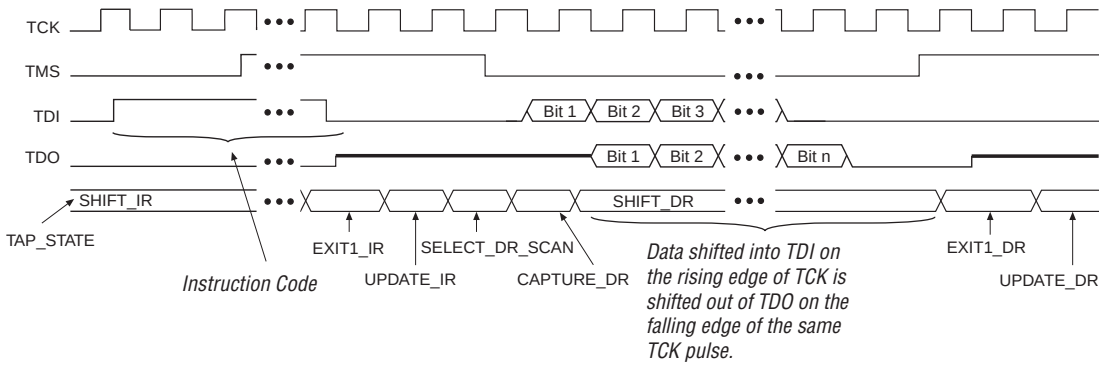
**Figure 13–11.** EXTEST Shift Data Register Waveforms




## BYPASS Instruction Mode

The BYPASS instruction mode is activated with an instruction code made up of only ones. The waveforms in Figure 13–12 show how scan data passes through a device once the TAP controller is in the SHIFT_DR state. In this state, data signals are clocked into the bypass register from TDI on the rising edge of TCK and out of TDO on the falling edge of the same clock pulse.

**Figure 13–12.** BYPASS Shift Data Register Waveforms



## IDCODE Instruction Mode

The IDCODE instruction mode is used to identify the devices in an IEEE Std. 1149.1 chain. When IDCODE is selected, the device identification register is loaded with the 32-bit vendor-defined identification code. The device ID register is connected between the TDI and TDO ports, and the device IDCODE is shifted out.

The IDCODE for MAX II devices are listed in the *JTAG and In-System Programmability* chapter in the *MAX II Device Handbook*.

## USERCODE Instruction Mode

The USERCODE instruction mode is used to examine the user electronic signature (UES) within the devices along an IEEE Std. 1149.1 chain. When this instruction is selected, the device identification register is connected between the TDI and TDO ports. The user-defined UES is shifted into the device ID register in parallel from the 32-bit USERCODE register. The UES is then shifted out through the device ID register. The USERCODE information is available to the user only after the device is configured successfully.

The non-volatile USERCODE data is written to the configuration flash memory (CFM) block and then written to the SRAM at power-up. The USERCODE instruction reads the data values from the SRAM. When using real-time ISP to update the CFM block and write new USERCODE data, executing the USERCODE instruction returns the current running design's USERCODE (stored in the SRAM), not the new USERCODE data. The new design's USERCODE, stored in the CFM, can only be read back correctly if a power cycle or forced SRAM download has transpired after the real-time ISP update.

In the Quartus II software, there is an **Auto Usercode** feature where you can choose to use the checksum value of a programming file as the JTAG user code. If selected, the checksum will be automatically loaded to the USERCODE register. On the Assignments menu, click **Device**. In the Device dialog box, click **Device and Pin Options** and click the General tab. Turn on **Auto Usercode**.

## CLAMP Instruction Mode

The CLAMP instruction mode is used to allow the state of the signals driven from the pins to be determined from the boundary-scan register while the bypass register is selected as the serial path between the TDI and TDO ports. The state of all signals driven from the output pins will be completely defined by the data held in the boundary-scan register. However, CLAMP will not override the I/O weak pull-up resistor or the I/O bus hold if you have any of them selected.

## HIGHZ Instruction Mode

The HIGHZ instruction mode is used to set all of the user I/O pins to an inactive drive state. These pins are tri-stated until a new JTAG instruction is executed. When this instruction is selected, the bypass register is connected between the TDI and TDO ports. HIGHZ will not override the I/O weak pull-up resistor or the I/O bus hold if you have any of them selected.

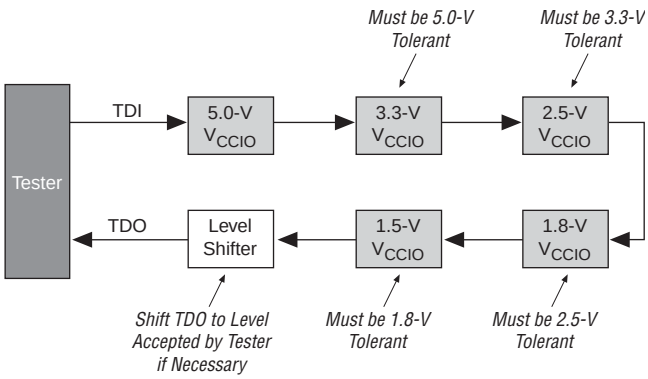
# I/O Voltage Support in JTAG Chain

There can be several different Altera or non-Altera devices in a JTAG chain. However, you should be cautious if the chain contains devices that have different $V_{CCIO}$ levels. The TDO pin of a device drives out at the voltage level according to the $V_{CCIO}$ of the device. For MAX II devices, the TDO pin will drive out at the voltage level according to the $V_{CCIO}$ of I/O Bank 1. The devices can interface with each other although they might have different $V_{CCIO}$ levels. For example, a device with 3.3-V $V_{CCIO}$ can drive to a device with 5.0-V $V_{CCIO}$ because 3.3 V meets the minimum VIH on TTL-level input for the 5.0-V $V_{CCIO}$ device. JTAG pins on MAX II devices can support 1.5-, 1.8-, 2.5-, or 3.3-V input levels, depending on the $V_{CCIO}$ voltage of I/O Bank 1.

Refer to the *MAX II Architecture* chapter in the *MAX II Device Handbook* for more information on MultiVoltTM I/O support.

You can interface the TDI and TDO lines of the JTAG pins of devices that have different $V_{CCIO}$ levels by inserting a level shifter between the devices. If possible, the JTAG chain should be built such that a device with a higher $V_{CCIO}$ level drives to a device with an equal or lower $V_{CCIO}$ level. By building the JTAG chain in this manner, a level shifter may be required only to shift the TDO level to a level acceptable to the JTAG tester. Figure 13–13 shows the JTAG chain of mixed voltages and how a level shifter is inserted in the chain.

**Figure 13–13.** JTAG Chain of Mixed Voltages




# BST for Programmed Devices

For a programmed device, the input buffers are turned off by default for I/O pins that are set as output only in the design file. You cannot sample on the programmed device output pins with the default BSDL file when the input buffers are turned off. You can set the Quartus II software to always enable the input buffers on a programmed device so it behaves the same as an unprogrammed device for boundary-scan testing, allowing sample function on output pins in the design. This aspect can cause slight increase in standby current as the unused input buffer is always on.

1. On the Assignments menu, click **Settings.**
2. Under **Category**, select **Assembler**.
3. Turn on **Always Enable Input Buffers**.

# Disabling IEEE Std. 1149.1 BST Circuitry

The IEEE Std. 1149.1 BST circuitry for MAX II devices is enabled upon device power-up. Because this circuitry may be used for BST or ISP, this circuitry must be enabled only if these features are used. This section describes how to disable the IEEE Std. 1149.1 circuitry to ensure that the circuitry is not inadvertently enabled when it is not needed.

Table 13–3 shows the pin connections necessary for disabling JTAG in MAX II devices that have dedicated IEEE Std. 1149.1 pins.

**Table 13–3.** Disabling IEEE Std. 1149.1 Circuitry

| JTAG Pins *(1)* | | | |
|---|---|---|---|
| TMS | TCK | TDI | TDO |
| VCC *(2)* | GND *(3)* | VCC *(2)* | Leave Open |

**Notes to Table 13–3:**

(1) There is no software option to disable JTAG in MAX II devices. The JTAG pins are dedicated.

(2) VCC refers to $V_{CCIO}$ of Bank 1.

(3) The TCK signal may also be tied high. If TCK is tied high, power-up conditions must ensure that TMS is pulled high before TCK. Pulling TCK low avoids this power-up condition.

# Guidelines for IEEE Std. 1149.1 Boundary-Scan Testing

Use the following guidelines when performing boundary-scan testing with IEEE Std. 1149.1 devices:

- If a pattern (for example, a 10-bit 1010101010 pattern) does not shift out of the instruction register via the TDO pin during the first clock cycle of the SHIFT_IR state, the proper TAP controller state has not been reached. To solve this problem, try one of the following procedures:
  - Verify that the TAP controller has reached the SHIFT_IR state correctly. To advance the TAP controller to the SHIFT_IR state, return to the RESET state and clock the code 01100 on the TMS pin.
  - Check the connections to the VCC, GND, and JTAG pins on the device.
- Perform a SAMPLE/PRELOAD test cycle prior to the first EXTEST test cycle to ensure that known data is present at the device pins when the EXTEST mode is entered. If the OEJ update register contains a 0, the data in the OUTJ update register will be driven out. The state must be known and correct to avoid contention with other devices in the system.
- Do not perform EXTEST and SAMPLE/PRELOAD tests during ISP. These instructions are supported before and after ISP but not during ISP.

If problems persist, contact Altera Applications.

# Boundary-Scan Description Language (BSDL) Support

The BSDL—a subset of VHDL—provides a syntax that allows you to describe the features of an IEEE Std. 1149.1 BST-capable device that can be tested. Test software development systems then use the BSDL files for test generation, analysis, failure diagnostics, and in-system programming.

For more information, or to receive BSDL files for IEEE Std. 1149.1-compliant MAX II devices, refer to the Altera website at www.altera.com.

# Conclusion

The IEEE Std. 1149.1 BST circuitry available in MAX II devices provides a cost-effective and efficient way to test systems that contain devices with tight lead spacing. Circuit boards with Altera and other IEEE Std. 1149.1-compliant devices can use the EXTEST, SAMPLE/PRELOAD, and BYPASSmodes to create serial patterns that internally test the pin connections between devices and check device operation.

Institute of Electrical and Electronics Engineers, Inc. IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE Std. 1149.1-2001). New York: Institute of Electrical and Electronics Engineers, Inc., 2001.

# Referenced Documents

This chapter references the following documents:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*
- *JTAG and In-System Programmability* chapter in the *MAX II Device Handbook*
- *MAX II Architecture* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 13–4 shows the revision history for this chapter.

**Table 13–4.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.7 | ■ Updated New Document Format. | — |
| December 2007, version 1.6 | ■ Removed Figure 13-14.<br>■ Updated Figure 13–6.<br>■ Added "Referenced Documents" section. | — |
| December 2006, version 1.5 | ■ Added document revision history. | — |
| August 2006, version 1.4 | ■ Updated IEEE Std. 1149.1 BST Operation<br>■ Control section. | — |
| July 2006, version 1.3 | ■ Updated "BST for Programmed Devices" section. | — |
| June 2005, version 1.2 | ■ Added a paragraph under the *USERCODE Instruction Mode* section.<br>■ Added a new section - *BST for Programmed Devices*. | — |
| January 2005, version 1.1 | ■ Previously published as Chapter 14. No changes to content. | — |
| March 2004, version 1.0 | ■ Initial Release. | — |

# 14. Using Jam STAPL for ISP via an Embedded Processor

MII51015-1.8

## Introduction

Advances in programmable logic devices (PLDs) have enabled the innovative in-system programmability (ISP) feature. The Jam™ Standard Test and Programming Language (STAPL), JEDEC standard JESD-71, is compatible with all current PLDs that offer ISP via Joint Test Action Group (JTAG), providing a software-level, vendor-independent standard for in-system programming and configuration. Designers who use Jam STAPL to implement ISP enhance the quality, flexibility, and life-cycle of their end products. Regardless of the number of PLDs that must be programmed or configured, Jam STAPL simplifies in-field upgrades and revolutionizes the programming of PLDs.

This chapter describes MAX® II device programming support using Jam STAPL in embedded systems.

This chapter contains the following sections:


- "Embedded Systems" on page 14–1
- "Software Development" on page 14–4
- "Updating Devices Using Jam" on page 14–14


## Embedded Systems

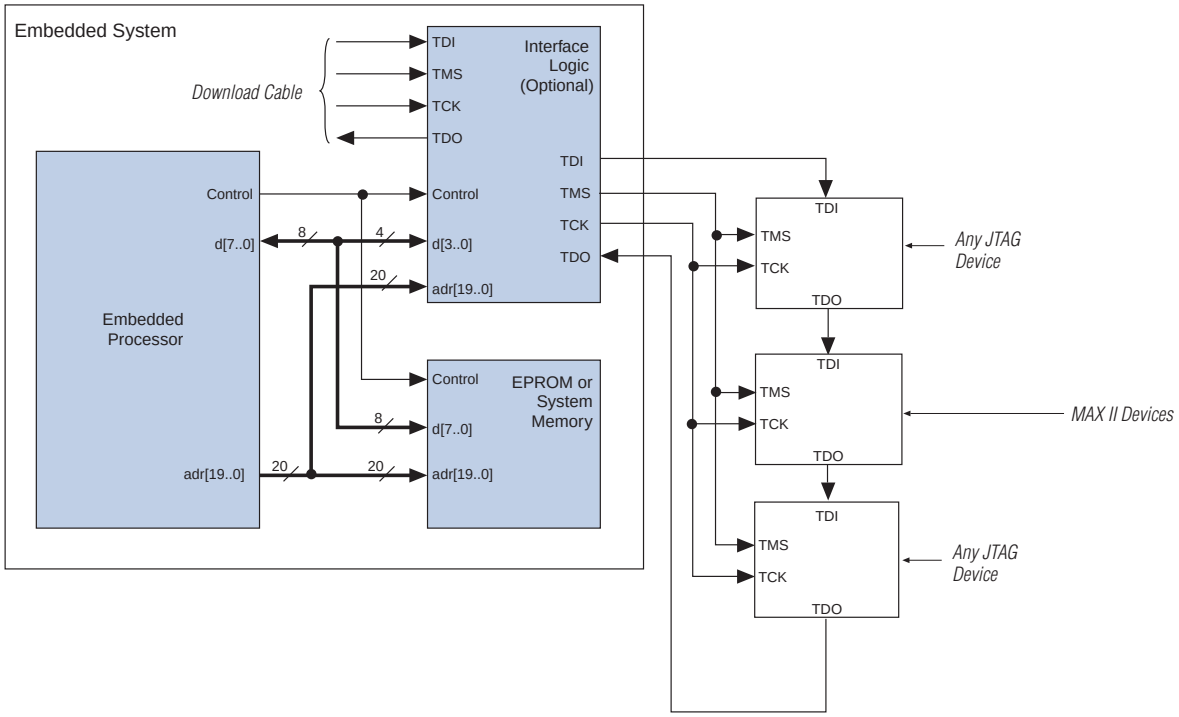
All embedded systems are made up of both hardware and software components. When designing an embedded system, the first step is to layout the printed circuit board (PCB). The second step is to develop the firmware that manages the board's functionality.

### Connecting the JTAG Chain to the Embedded Processor

There are two ways to connect the JTAG chain to the embedded processor. The most straightforward method is to connect the embedded processor directly to the JTAG chain. In this method, four of the processor pins are dedicated to the JTAG interface, thereby saving board space but reducing the number of available embedded processor pins.

Figure 14–1 illustrates the second method, which is to connect the JTAG chain to an existing bus via an interface PLD. In this method, the JTAG chain becomes an address on the existing bus. The processor then reads from or writes to the address representing the JTAG chain.

**Figure 14–1.** Embedded System Block Diagram




Both JTAG connection methods should include space for the MasterBlaster™, ByteBlaster™ II, or USB-Blaster™ header connection. The header is useful during prototyping because it allows designers to quickly verify or modify the PLD's contents. During production, the header can be removed to decrease cost.

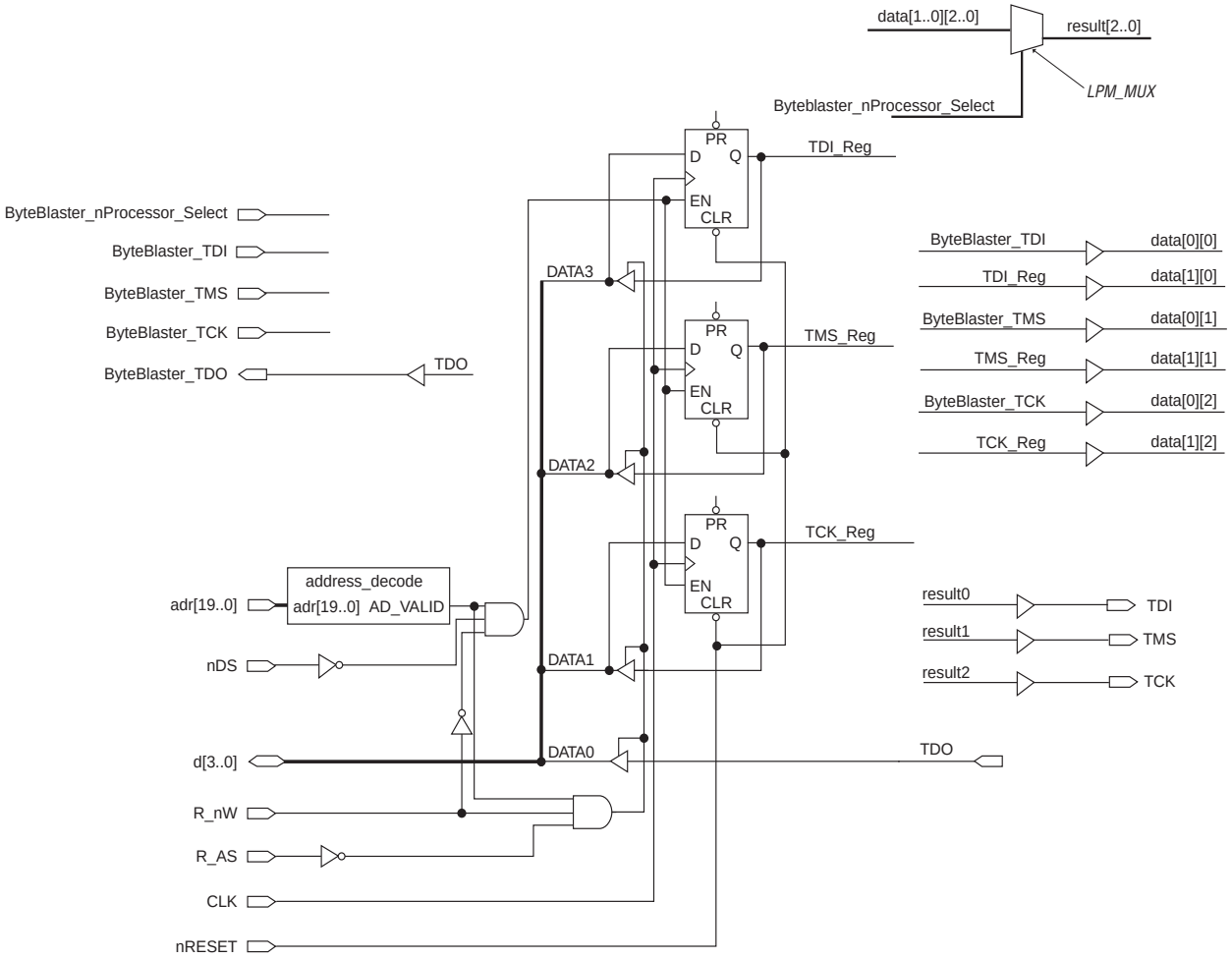
## Example Interface PLD Design

Figure 14–2 shows an example design schematic of an interface PLD. A different design can be implemented; however, important points exemplified in this design are:

- `TMS`, `TCK`, and `TDI` should be synchronous outputs
- Multiplexer logic should be included to allow board access for the MasterBlaster, ByteBlaster II, or USB-Blaster download cable

This design example is for reference only. All of the inputs except `data[3..0]` are optional and included only to show how an interface PLD can act as an address decoder on an embedded data bus.

**Figure 14–2.** Interface Logic Design Example




In Figure 14–2, the embedded processor asserts the JTAG chain's address, and the R_nW and R_AS signals can be set to tell the interface PLD when the processor wants to access the chain. A write involves connecting the data path data[3..0] to the JTAG outputs of the PLD via the three D registers that are clocked by the system clock (CLK). This clock can be the same clock that the processor uses. Likewise, a read involves enabling the tri-state buffers and letting the TDO signal flow back to the processor. The design also provides a hardware connection to read back the values in the TDI, TMS, and TCK registers. This optional feature is useful during the development phase, allowing software to check the valid states of the registers in the interface PLD. In addition, multiplexer logic is included to permit a download cable to program the device chain. This capability is useful during the prototype phase of development, when programming must be verified.

## Board Layout

The following elements are important when laying out a board that programs via the IEEE Std. 1149.1 JTAG chain:

- Treat the TCK signal trace as a clock tree

- Use a pull-down resistor on TCK
- Make the JTAG signal traces as short as possible
- Add external resistors to pull outputs to a defined logic level

### TCK Signal Trace Protection and Integrity

TCK is the clock for the entire JTAG chain of devices. These devices are edge-triggered on the TCK signal, so it is imperative that TCK is protected from high-frequency noise and has good signal integrity. Ensure that the signal meets the rise time ($t_R$) and fall time ($t_F$) parameters in the appropriate device family data sheet. The signal may also need termination to prevent overshoot, undershoot, or ringing. This step is often overlooked since this signal is software-generated and originates at a processor general-purpose I/O pin.

### Pull-Down Resistors on TCK

TCK should be held low via a pull-down resistor to keep the JTAG Test Access Port (TAP) in a known state at power-up. A missing pull-down resistor can cause a device to power-up in a JTAG BST state, which may cause conflicts on the board. A typical resistor value is 1 kΩ.

### JTAG Signal Traces

Short JTAG signal traces help eliminate noise and drive-strength issues. Special attention should be paid to the TCK and TMS pins. Because TCK and TMS are connected to every device in the JTAG chain, these traces will see higher loading than TDI or TDO. Depending on the length and loading of the JTAG chain, some additional buffering may be required to ensure that the signals propagate to and from the processor with integrity.

### External Resistors

You should add external resistors to output pins to pull outputs to a defined logic level during programming. Output pins will tri-state during programming. Also, on MAX® II devices, the pins will be pulled up by a weak internal resistor. Altera recommends that outputs driving sensitive input pins be tied to the appropriate level by an external resistor.

Each preceding board layout element may require further analysis, especially signal integrity. In some cases, you may need to analyze the loading and layout of the JTAG chain to determine whether to use discrete buffers or a termination technique.

For more information, refer to the *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*.

## Software Development

Altera's embedded programming uses the Jam file output from the Quartus® II software tool with the standardized Jam Player software. Designing these tools requires minimal developer intervention because Jam files contain all of the data for programming MAX II devices. The bulk of development time is spent porting the Jam Player to the host embedded processor.

For more information about porting the Jam Byte-Code Player, see "Porting the Jam STAPL Byte-Code Player" on page 14–8.

## Jam Files (.jam and .jbc)

Altera supports the following types of Jam files:

- ASCII text files (**.jam**)
- Jam Byte-Code files (**.jbc**)

### ASCII Text Files (.jam)

Altera supports two types of Jam files:

- JEDEC Jam STAPL format
- Jam version 1.1 (pre-JEDEC format)

The JEDEC Jam STAPL format uses the syntax specified by the JEDEC Standard JESD-71A specification. Altera recommends using JEDEC Jam STAPL files for all new projects. In most cases, Jam files are used in tester environments.

### Jam Byte-Code Files (.jbc)

JBC files are binary files that are compiled versions of Jam files. JBC files are compiled to a virtual processor architecture, where the ASCII Jam commands are mapped to byte-code instructions compatible with the virtual processor. There are two types of JBC files:

- Jam STAPL Byte-Code (compiled version of JEDEC Jam STAPL file)
- Jam Byte-Code (compiled version of Jam version 1.1 file)

Altera recommends using Jam STAPL Byte-Code files in embedded applications because they use minimal memory.

## Generating Jam Files

The Quartus II software can generate both Jam and JBC file types. In addition, Jam files can be compiled into JBC files via a stand-alone Jam Byte-Code compiler. The compiler produces a functionally equivalent JBC file.

Generating JBC files directly from the Quartus II software is simple. The software tool supports the programming and configuration of multiple devices from single or multiple JBC files. Figure 14–3 and Figure 14–4 show the dialog boxes that specify the device chain and JBC file generation in the Quartus II software.

**Figure 14–3.** Multi-Device JTAG Chain's Name and Sequence in Programmer Window in the Quartus II Software

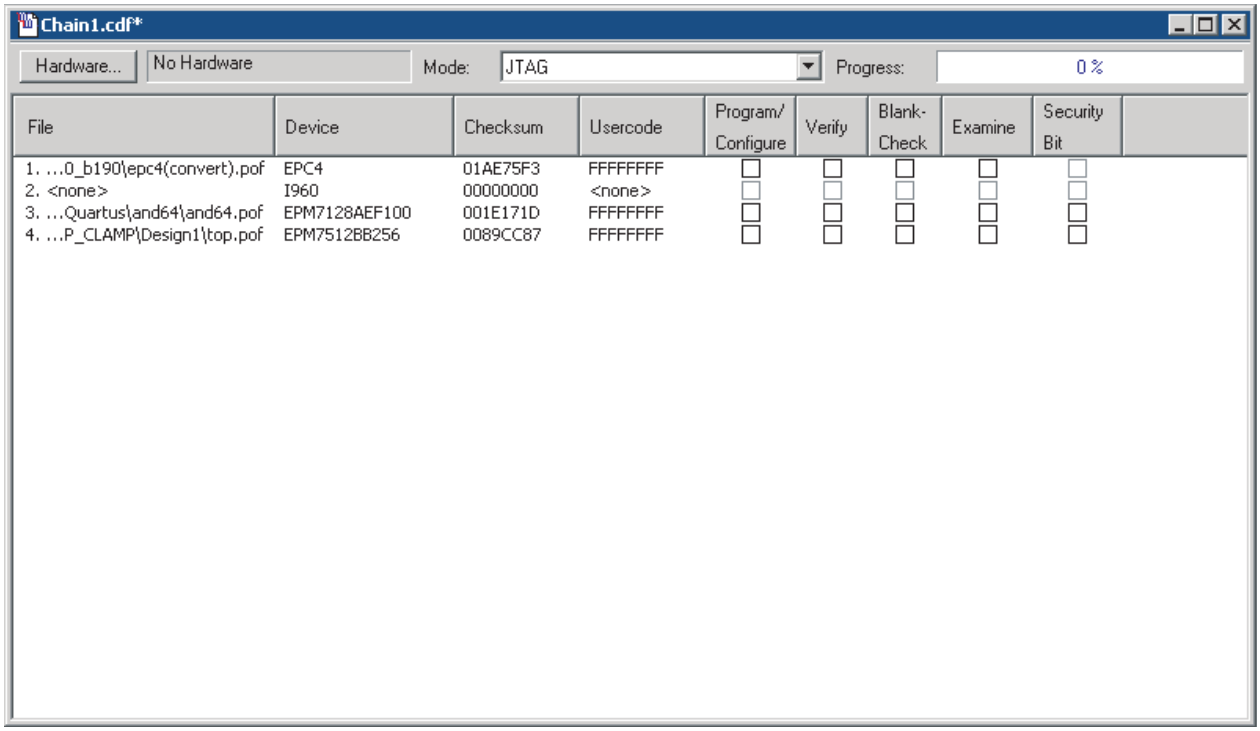
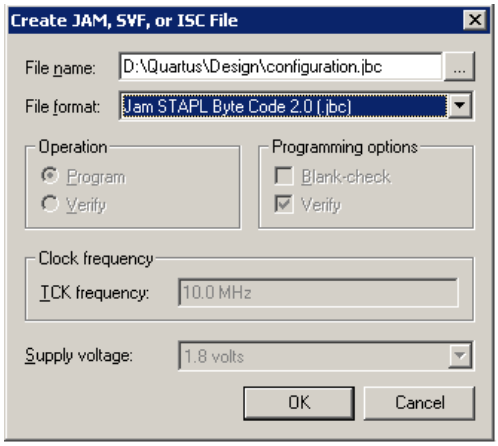



**Figure 14–4.** Generating a JBC File for a Multi-Device JTAG Chain in the Quartus II Software




The following steps explain how to generate JBC files using the Quartus II software.

1. On the Tools menu, click **Programmer**.
2. Click **Add File** and select programming files for the respective devices.
3. On the File menu, point to **Create/Update** and click **Create JAM, SVF, or ISC File**. See Figure 14–4.
4. Specify a Jam STAPL Byte-Code File in the File format list.
5. Click **OK**.

You can include both Altera and non-Altera JTAG-compliant devices in the JTAG chain. If you do not specify a programming file in the *Programming File Names* field, devices in the JTAG chain will be bypassed.

### Using Jam Files with the MAX II User Flash Memory Block

The Quartus II Programmer provides the option to individually target the entire device, logic array, or the user flash memory (UFM) block. As you can program the (UFM) section independently from the logic array, separate Jam STAPL and JBC options can be used in the command line to separately program UFM and configuration flash memory (CFM) blocks.

For more information, see "MAX II Jam/JBC Actions and Procedure Commands" on page 14–15.

## Jam Players

Jam Players read the descriptive information in Jam files and translate them into data that programs the target PLDs. Jam Players do not program a particular device architecture or vendor; they only read and understand the syntax defined by the Jam file specification. In-field changes are confined to the Jam file, not the Jam Player. As a result, you do not need to modify the Jam Player source code for each in-field upgrade.

There are two types of Jam Players to accommodate the two types of Jam files: an ASCII Jam STAPL Player and a Jam STAPL Byte-Code Player. The general concepts within this chapter apply to both player types; however, the following information focuses on the Jam STAPL Byte-Code Player.

Jam Players can be used to program or write the MAX II configuration flash memory block and the UFM block separately since Jam STAPL and JBC files can be generated targeting only to either one or both sectors of the MAX II UFM block.

### Jam Player Compatibility

The embedded Jam Player is able to read Jam files that conform to the standard JEDEC file format. The embedded Jam Player is compatible with legacy Jam files that use version 1.1 syntax. Both Players are backward-compatible; they can play version 1.1 files and Jam STAPL files.

For more information about Altera's support for version 1.1 syntax, refer to *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*.

### The Jam STAPL Byte-Code Player

The Jam STAPL Byte-Code Player is coded in the C programming language for 16-bit and 32-bit processors.

For more information about Altera's support for 8-bit processors, refer to *AN 111: Embedded Programming Using the 8051 & Jam Byte-Code*.

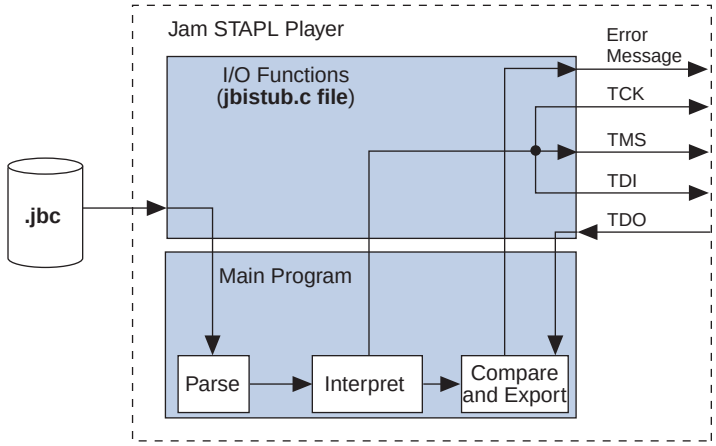
The 16-bit and 32-bit source code is divided into two categories:

- Platform-specific code that handles I/O functions and applies to specific hardware (**jbistub.c**)

- Generic code that performs the Player's internal functions (all other C files)

Figure 14–5 illustrates the organization of the source code files by function. Keeping the platform-specific code inside the **jbistub.c** file simplifies the process of porting the Jam STAPL Byte-Code Player to a particular processor.

**Figure 14–5.** Jam STAPL Byte-Code Player Source Code Structure




## Porting the Jam STAPL Byte-Code Player

The default configuration of the **jbistub.c** file includes code for DOS, 32-bit Windows, and UNIX so that the source code can be easily compiled and evaluated for the correct functionality and debugging of these pre-defined operating systems. For the embedded environment, this code is easily removed using a single preprocessor `#define` statement. In addition, porting the code involves making minor changes to specific parts of the code in the **jbistub.c** file.

To port the Jam Player, you need to customize several functions in the jbistub.c file, which are shown in Table 14–1.

**Table 14–1.** Functions Requiring Customization

| Function | Description |
|---|---|
| `jbi_jtag_io()` | Interface to the four IEEE 1149.1 JTAG signals, `TDI`, `TMS`, `TCK`, and `TDO` |
| `jbi_export()` | Passes information such as the User Electronic Signature (UES) back to the calling program |
| `jbi_delay()` | Implements the programming pulses or delays needed during execution |
| `jbi_vector_map()` | Processes signal-to-pin map for non-IEEE 1149.1 JTAG signals |
| `jbi_vector_io()` | Asserts non-IEEE 1149.1 JTAG signals as defined in the VECTOR MAP |

To ensure that you have customized all of the necessary code, follow these four steps:

1. Set preprocessor statements to exclude extraneous code.
2. Map JTAG signals to hardware pins.
3. Handle text messages from `jbi_export()`.
4. Customize delay calibration.

### Step 1: Set Preprocessor Statements to Exclude Extraneous Code

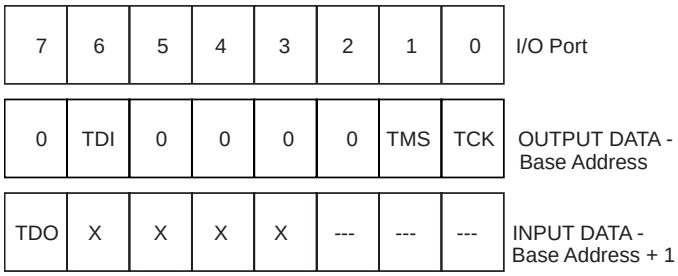
At the top of jbistub.c, change the default PORT parameter to EMBEDDED to eliminate all DOS, Windows, and UNIX source code and included libraries.

```
#define PORT EMBEDDED
```

### Step 2: Map JTAG Signals to Hardware Pins

The jbi_jtag_io() function contains the code that sends and receives the binary programming data. Each of the four JTAG signals should be re-mapped to the embedded processor's pins. By default, the source code writes to the PC's parallel port. The jbi_jtag_io() signal maps the JTAG pins to the PC parallel port registers shown in Figure 14–6.

**Figure 14–6.** Default PC Parallel Port Signal Mapping *(Note 1)*




**Note to Figure 14–6:**

(1) The PC parallel port hardware inverts the most significant bit, TDO.

The mapping is highlighted in the following jbi_jtag_io() source code:

```
int jbi_jtag_io(int tms, int tdi, int read_tdo)
{
    int data=0;
    int tdo=0;

    if (!jtag_hardware_initialized)
    {
        initialize_jtag_hardware();
        jtag_hardware_initialized=TRUE;
    }
    data = ((tdi?0x40:0)|(tms?0x2:0));       /*TDI,TMS*/
    write_byteblaster(0,data);
    if (read_tdo)
    {
        tdo=(read_byteblaster(1)&0x80)?0:1;  /*TDO*/
    }
    write_blaster(0,data|0x01);             /*TCK*/
    write_blaster(0,data);
    return (tdo);
}
```

In the previous code, the PC parallel port inverts the actual value of TDO. The jbi_jtag_io() source code inverts it again to retrieve the original data. The line which inverts the TDO value is as follows:

```
tdo=(read_byteblaster(1)&0x80)?0:1;
```

If the target processor does not invert TDO, the code should look like:

```
tdo=(read_byteblaster(1)&0x80)?1:0;
```

To map the signals to the correct addresses, use the left shift (<<) or right shift (>>) operators. For example, if TMS and TDI are at ports 2 and 3, respectively, the code would be as follows:

```
data=(((tdi?0x40:0)>>3)|((tms?0x02:0)<<1));
```

Apply the same process to TCK and TDO.

The read_byteblaster and write_byteblaster signals use the inp() and outp() functions from the **conio.h** library, respectively, to read and write to the port. If these functions are not available, equivalent functions should be substituted.

### Step 3: Handle Text Messages from jbi_export()

The jbi_export() function sends text messages to stdio, using the printf() function. The Jam STAPL Byte-Code Player uses the jbi_export() signal to pass information (for example, the device UES or USERCODE) to the operating system or software that calls the Player. The function passes text (in the form of a string) and numbers (in the form of a decimal integer).

For definitions of these terms, refer to *AN 39: IEEE 1149.1 (JTAG) Boundary-Scan Testing in Altera Devices*.

If there is no device available to stdout, the information can be redirected to a file or storage device, or passed as a variable back to the program that calls the Player.

### Step 4: Customize Delay Calibration

The calibrate_delay() function determines how many loops the host processor runs in a millisecond. This calibration is important because accurate delays are used in programming and configuration. By default, this number is hard-coded as 1,000 loops per millisecond and represented as the following assignment:

```
one_ms_delay = 1000
```

If this parameter is known, it should be adjusted accordingly. If it is not known, you can use code similar to that for Windows and DOS platforms. Code is included for these platforms that count the number of clock cycles that run in the time it takes to execute a single while loop. This code is sampled over multiple tests and averaged to produce an accurate result upon which the delay can be based. The advantage to this approach is that calibration can vary based on the speed of the host processor.

Once the Jam STAPL Byte-Code Player is ported and working, verify the timing and speed of the JTAG port at the target device. Timing parameters in MAX II devices should comply with the values given in the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*.

If the Jam STAPL Byte-Code Player does not operate within the timing specifications, the code should be optimized with the appropriate delays. Timing violations can occur if the processor is very powerful and can generate TCK at a rate faster than 18 MHz.

Other than the **jbistub.c file**, Altera strongly recommends keeping source code in other files unchanged from their default state. Altering the source code in these files will result in unpredictable Jam Player operation.

## Jam STAPL Byte-Code Player Memory Usage

The Jam STAPL Byte-Code Player uses memory in a predictable manner. This section documents how to estimate both ROM and RAM memory usage.

### Estimating ROM Usage

Use the following equation to estimate the maximum amount of ROM required to store the Jam Player and JBC file:

**Equation 14–1.**

$$ROM_{Size} = \text{JBC file size} + \text{Jam player size}$$

The JBC file size can be separated into two categories: the amount of memory required to store the programming data, and the space required for the programming algorithm. Use the following equation to estimate the JBC file size:

**Equation 14–2.**

$$\text{JBC file size} = Alg + \sum_{k=1}^{N} Data$$

**Notes to Equation 14–2:**

(1) $Alg$ =Space used by algorithm.
(2) $Data$ =Space used by compressed programming data.
(3) $k$ =Index representing device being targeted.
(4) $N$ =Number of target devices in the chain.

This equation provides a JBC file size estimate that may vary by ±10%, depending on device utilization. When device utilization is low, JBC file sizes tend to be smaller because the compression algorithm used to minimize file size is more likely to find repetitive data.

The equation also indicates that the algorithm size stays constant for a device family, but the programming data size grows slightly as more devices are targeted. For a given device family, the increase in JBC file size (due to the data component) is linear.

Table 14–2 shows algorithm file size constants when targeting a single MAX II device.

**Table 14–2.** Algorithm File Size Constants Targeting a Single Altera Device Family

| Device | Typical JBC File Algorithm Size (Kbytes) |
|---|---|
| MAX II | 24.3 |

Table 14–3 shows data size constants for MAX II devices that support the Jam language for ISP.

**Table 14–3.** Data Constants

| Device | Typical Jam STAPL Byte-Code Data Size (Kbytes) | |
|---|---|---|
| | Compressed | Uncompressed *(1)* |
| EPM240 | 12.4 *(2)* | 12.4 *(2)* |
| EPM570 | 11.4 | 19.6 |
| EPM1270 | 16.9 | 31.9 |
| EPM2210 | 24.7 | 49.3 |

**Notes to Table 14–3:**

(1) For more information about how to generate JBC files with uncompressed programming data, contact Altera Applications.

(2) There is a minimum limit of 64K bits for compressed arrays with the JBC compiler. Programming data arrays smaller than 64K bits (8K bytes) are not compressed. The EPM240 programming data array is below the limit, which means the JBC files are always uncompressed. The reason for this limit is that a memory buffer is needed for decompression, and for small embedded systems it is more efficient to use small uncompressed arrays directly rather than to uncompress the arrays.

After estimating the JBC file size, estimate the Jam Player size using the information in Table 14–4.

**Table 14–4.** Jam STAPL Byte-Code Player Binary Sizes

| Build | Description | Size (Kbytes) |
|---|---|---|
| 16-bit | Pentium/486 using the MasterBlaster or ByteBlasterMV download cables | 80 |
| 32-bit | Pentium/486 using the MasterBlaster or ByteBlasterMV download cables | 85 |

### Estimating Dynamic Memory Usage

Use the following equation to estimate the maximum amount of DRAM required by the Jam Player:

**Equation 14–3.**

$$\text{RAM Size} = \text{JBC File Size} + \sum_{k=1}^{N} Data \text{ (Uncompressed Data Size)}k$$

The JBC file size is determined by a single- or multi-device equation (see "Estimating ROM Usage" on page 14–11).

The amount of RAM used by the Jam Player is the size of the JBC file plus the sum of the data required for each device that is targeted. If the JBC file is generated using compressed data, some RAM is used by the Player to uncompress the data and temporarily store it. The uncompressed data sizes are provided in Table 14–3. If an uncompressed JBC file is used, use the following equation:

**Equation 14–4.**

$$\text{RAM Size} = \text{JBC File Size}$$

The memory requirements for the stack and heap are negligible, with respect to the total amount of memory used by the Jam STAPL Byte-Code Player. The maximum depth of the stack is set by the `JBI_STACK_SIZE` parameter in the **jbimain.c** file.

### Estimating Memory Example

The following example uses a 16-bit Motorola 68000 processor to program an EPM7128AE device and an EPM7064AE device in an IEEE Std. 1149.1 JTAG chain via a JBC file that uses compressed data. To determine memory usage, first determine the amount of ROM required and then estimate the RAM usage. Use the following steps to calculate the amount of DRAM required by the Jam Byte-Code Player:

1. Determine the JBC file size. Use the following multi-device equation to estimate the JBC file size. Because JBC files use compressed data, use the compressed data file size information, listed in Table 14–3, to determine Data size.where:

**Equation 14–5.**

$$\text{JBC File Size} = \text{Alg} + \sum_{k=1}^{N} Data$$

**Notes to** Equation 14–5:

(1) *Alg* =21 Kbytes.

(2) *Data* =EPM7064AE Data + EPM7128AE Data = 8 + 4 = 12 Kbytes.

Thus, the JBC file size equals 33 Kbytes.

2. Estimate the JBC Player size. This example uses a JBC Player size of 62 Kbytes because this 68000 is a 16-bit processor. Use the following equation to determine the amount of ROM needed:

**Equation 14–6.**

$$\text{ROM Size} = \text{JBC File Size} + \text{Jam Player Size}$$

$$\text{ROM Size} = 95 \text{ Kbytes}$$

3. Estimate the RAM usage with the following equation:

**Equation 14–7.**

$$\text{RAM Size} = 33 \text{ Kbytes} + \sum_{k=1}^{N} Data \ \text{(Uncompressed Data Size)k}$$

Because the JBC file uses compressed data, the uncompressed data size for each device must be summed to find the total amount of RAM used. The Uncompressed Data Size constants are as follows:

- EPM7064AE = 8 Kbytes
- EPM7128AE = 12 Kbytes

Calculate the total DRAM usage as follows:

**Equation 14–8.**

$$\text{RAM Size} = 33\ \text{Kbytes} + (8\ \text{Kbytes} + 12\ \text{Kbytes}) = 53\ \text{Kbytes}$$

In general, Jam Files use more RAM than ROM, which is desirable because RAM is cheaper and the overhead associated with easy upgrades becomes less of a factor as a larger number of devices are programmed. In most applications, easy upgrades outweigh the memory costs.

# Updating Devices Using Jam

Updating a device in the field means downloading a new JBC file and running the Jam STAPL Byte-Code Player with what in most cases is the "program" action.

The main entry point for execution of the Player is `jbi_execute()`. This routine passes specific information to the Player. When the Player finishes, it returns an exit code and detailed error information for any run-time errors. The interface is defined by the routine's prototype definition.

```
JBI_RETURN_TYPE jbi_execute
(
    PROGRAM_PTR program
    long program_size,
    char *workspace,
    long workspace_size,
    *action,
    char **init_list,
    long *error_line,
    init *exit_code
)
```

The code within main(), in **jbistub.c**, determines the variables that will be passed to `jbi_execute()`. In most cases, this code is not applicable to an embedded environment; therefore, this code can be removed and the `jbi_execute()` routine can be set up for the embedded environment. Table 14–5 describes each parameter.

**Table 14–5.** Parameters *(Note 1)* (Part 1 of 2)

| Parameter | Status | Description |
|---|---|---|
| `program` | Mandatory | A pointer to the JBC file. For most embedded systems, setting up this parameter is as easy as assigning an address to the pointer before calling `jbi_execute()`. |
| `program_size` | Mandatory | Amount of memory (in bytes) that the JBC file occupies. |
| `workspace` | Optional | A pointer to dynamic memory that can be used by the JBC Player to perform its necessary functions. The purpose of this parameter is to restrict Player memory usage to a pre-defined memory space. This memory should be allocated before calling `jbi_execute()`. If maximum dynamic memory usage is not a concern, set this parameter to null, which allows the Player to dynamically allocate the necessary memory to perform the specified action. |
| `workspace_size` | Optional | A scalar representing the amount of memory (in bytes) to which `workspace` points. |

**Table 14–5.** Parameters *(Note 1)* (Part 2 of 2)

| Parameter | Status | Description |
|---|---|---|
| action | Mandatory | A pointer to a string (text that directs the Player). Example actions are PROGRAM or VERIFY. In most cases, this parameter will be set to the string PROGRAM. The Player is not case-sensitive, so the text can be either uppercase or lowercase. The Player supports all actions defined in the *Jam Standard Test and Programming Language Specification*. See Table 14–6. Note that the string must be null terminated. |
| init_list | Optional | An array of pointers to strings. This parameter is used when applying Jam version 1.1 files. *(2)* |
| error_line | — | A pointer to a long integer. If an error is encountered during execution, the Player will record the line of the JBC file where the error occurred. |
| exit_code | — | A pointer to a long integer. Returns a code if there is an error that applies to the syntax or structure of the JBC file. If this kind of error is encountered, the supporting vendor should be contacted with a detailed description of the circumstances in which the exit code was encountered. |

**Notes to Table 14–5:**

(1) Mandatory parameters must be passed for the Player to run.

(2) For more information, refer to *AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor*.

## MAX II Jam/JBC Actions and Procedure Commands

The Jam/JBC supported action commands for MAX II devices are listed in Table 14–6, including their definitions. The optional procedures that you can execute with each action are listed along with their definitions in Table 14–7.

**Table 14–6.** MAX II Jam/JBC Actions (Part 1 of 2)

| Jam/JBC Action | Description | Optional Procedures (Off by Default) |
|---|---|---|
| PROGRAM | Programs the device. You can optionally program CFM and UFM separately. | DO_BYPASS_CFM<br>DO_BYPASS_UFM<br>DO_SECURE<br>DO_REAL_TIME_ISP<br>DO_READ_USERCODE |
| BLANKCHECK | Blank checks the entire device. You can optionally blank check CFM and UFM separately. | DO_BYPASS_CFM<br>DO_BYPASS_UFM<br>DO_REAL_TIME_ISP |
| VERIFY | Verifies the entire device against the programming data in the Jam file. You can optionally verify CFM and UFM separately. | DO_BYPASS_CFM<br>DO_BYPASS_UFM<br>DO_REAL_TIME_ISP<br>DO_READ_USERCODE |

**Table 14–6.** MAX II Jam/JBC Actions (Part 2 of 2)

| Jam/JBC Action | Description | Optional Procedures (Off by Default) |
|---|---|---|
| ERASE | Erases the programming content of the device. You can optionally erase CFM and UFM separately. | DO_BYPASS_CFM<br>DO_BYPASS_UFM<br>DO_REAL_TIME_ISP |
| READ_USERCODE | Returns the JTAG USERCODE register information from the device. READ_USERCODE can be set to a specific value in the programming file in the Quartus II software by using the Assignments menu -> Device -> Device and Pin options -> General tab, which has a USERCODE data entry. | — |

**Table 14–7.** MAX II Jam/JBC Optional Procedure Definitions

| Procedure | Description |
|---|---|
| DO_BYPASS_CFM | When set =1, DO_BYPASS_CFM bypasses the CFM and performs the specified action on the UFM only. When set =0, this option is ignored (default). |
| DO_BYPASS_UFM | When set =1, DO_BYPASS_UFM bypasses the UFM and performs the specified action on the CFM only. When set =0, this option is ignored (default). |
| DO_BLANKCHECK | When set =1, the device, CFM, or UFM is blank checked. When set =0, this option is ignored (default). |
| DO_SECURE | When set =1, the device's security bit is set. The security bit only affects the CFM data. The UFM cannot be protected. When set =0, this option is ignored (default). |
| DO_REAL_TIME_ISP | When set =1, the real-time ISP feature is enabled for the ISP action being executed. When set =0, the device uses normal ISP mode for any operations. |
| DO_READ_USERCODE | When set =1, the player returns the JTAG USERCODE register information from the device. |

Executing the Jam file from a command prompt requires that an action is specified using the -a option, as shown in the following example:

```
jam -aPROGRAM <filename>
```

This command programs the entire MAX II device with the Jam file specified in the filename.

You can execute the optional procedures with its associated actions by using the -d option, as shown in the following example:

```
jam -aPROGRAM -dDO_BYPASS_UFM=1
      -dDO_REAL_TIME_ISP=1 <filename>
```

This command programs the MAX II CFM block only with real-time ISP enabled (i.e., the device remains in user mode during the entire process).

The JBC player uses the same format except for the executable name.

The Player returns a status code of type `JBI_RETURN_TYPE` or integer. This value indicates whether the action was successful (returns "0"). `jbi_execute()` can return any one of the following exit codes in Table 14–8, as defined in the *Jam Standard Test and Programming Language Specification*.

**Table 14–8.** Exit Codes

| Exit Code | Description |
|---|---|
| 0 | Success |
| 1 | Checking chain failure |
| 2 | Reading IDCODE failure |
| 3 | Reading USERCODE failure |
| 4 | Reading UESCODE failure |
| 5 | Entering ISP failure |
| 6 | Unrecognized device ID |
| 7 | Device version is not supported |
| 8 | Erase failure |
| 9 | Blank check failure |
| 10 | Programming failure |
| 11 | Verify failure |
| 12 | Read failure |
| 13 | Calculating checksum failure |
| 14 | Setting security bit failure |
| 15 | Querying security bit failure |
| 16 | Exiting ISP failure |
| 17 | Performing system test failure |

## Running the Jam STAPL Byte-Code Player

Calling the Jam STAPL Byte-Code Player is like calling any other sub-routine. In this case, the sub-routine is given actions and a file name, and then it performs its function.

In some cases, in-field upgrades can be performed depending on whether the current device design is up-to-date. The JTAG USERCODE is often used as an electronic "stamp" that indicates the PLD design revision. If the USERCODE is set to an older value, the embedded firmware updates the device. The following pseudocode illustrates how the Jam Byte-Code Player could be called multiple times to update the target PLD:

```
result = jbi_execute(jbc_file_pointer, jbc_file_size, 0, 0,
"READ_USERCODE", 0, error_line, exit_code);
```

The Jam STAPL Byte-Code Player will now read the JTAG USERCODE and export it using the `jbi_export()` routine. The code can then branch based upon the result.

The following shows example code for the Jam Player.

```
switch (USERCODE)
{
    case "0001":   /*Rev 1 is old - update to new Rev*/
       result = jbi_execute (rev3_file, file_size_3, 0, 0, “PROGRAM”,
         0, error_line, exit_code);
    case "0002":   /*Rev 2 is old - update to new Rev*/
       result = jbi_excecute(rev3_file, file_size_3, 0, 0, "PROGRAM",
         0, error_line, exit_code);
    case "0003":
       ;           /*Do nothing - this is the current Rev*/
    default:       /*Issue warning and update to current Rev*/
       Warning - unexpected design revision;   /*Program device with
         newest rev anyway*/
       result = jbi_execute(rev3_file, file_size_3, 0, 0, "PROGRAM", 0,
         error_line, exit_code);
}
```

A switch statement can be used to determine which device needs to be updated and which design revision should be used. With Jam STAPL Byte-Code software support, PLD updates become as easy as adding a few lines of code.

## Conclusion

Using Jam STAPL provides an simple way to benefit from ISP. Jam meets all of the necessary embedded system requirements, such as small file sizes, ease of use, and platform independence. In-field upgrades are simplified by confining updates to the Jam STAPL Byte-Code file. Executing the Jam Player is straightforward, as is the calculation of resources that will be used. For the most recent updates and information, visit the Jam website at: www.altera.com/jamisp.

## Referenced Documents

This chapter references the following documents:

- AN 39: IEEE 1149.1 (JTAG) Boudary-Scan Testing in Altera Devices
- AN 111: Embedded Programming Using the 8051 & Jam Byte-Code
- AN 122: Using Jam STAPL for ISP & ICR via an Embedded Processor
- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*
- *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 14–9 shows the revision history for this chapter.

**Table 14–9.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.8 | ■ Updated New Document Format. | — |
| December 2007, version 1.7 | ■ Added "Referenced Documents" section. | — |
| December 2006, version 1.6 | ■ Added document revision history. | — |
| August 2006, version 1.5 | ■ Updated "Embedded Systems" section. | — |
| August 2005, version 1.4 | ■ Updated Tables 14-2 and 14-3. | — |
| June 2005, version 1.3 | ■ Removed Table 14-6 from v1.2.<br>■ Added a new section "MAX II Jam/JBC Actions and Procedure Commands". | — |
| January 2005, version 1.2 | ■ Previously published as Chapter 15. No changes to content. | — |
| December 2004, version 1.1 | ■ Changed document reference from AN 88 to AN 122. | — |

# 15. Using the Agilent 3070 Tester for In-System Programming

MII51016-1.5

## Introduction

In-system programming is a mainstream feature in programmable logic devices (PLDs), offering system designers and test engineers significant cost benefits by integrating PLD programming into board-level testing. These benefits include reduced inventory of pre-programmed devices, lower costs, fewer devices damaged by handling, and increased flexibility in engineering changes. Altera provides software and device support that integrates in-system programmability (ISP) into the existing test flows for the Agilent 3070 system. This chapter discusses how to use the Agilent 3070 test system to achieve faster programming times for Altera's MAX® II devices.

This chapter contains the following sections:


- "New PLD Product for Agilent 3070" on page 15–1
- "Device Support" on page 15–1
- "Agilent 3070 Development Flow without the PLD ISP Software" on page 15–2
- "Development Flow for Agilent 3070 with PLD ISP Software" on page 15–8
- "Programming Times" on page 15–10
- "Guidelines" on page 15–10


## New PLD Product for Agilent 3070

Agilent Technologies, the manufacturer of the Agilent 3070 tester, has introduced a new PLD ISP software product to help address the issues of programming PLDs. There are several advantages of using the new product that are discussed later in this chapter.

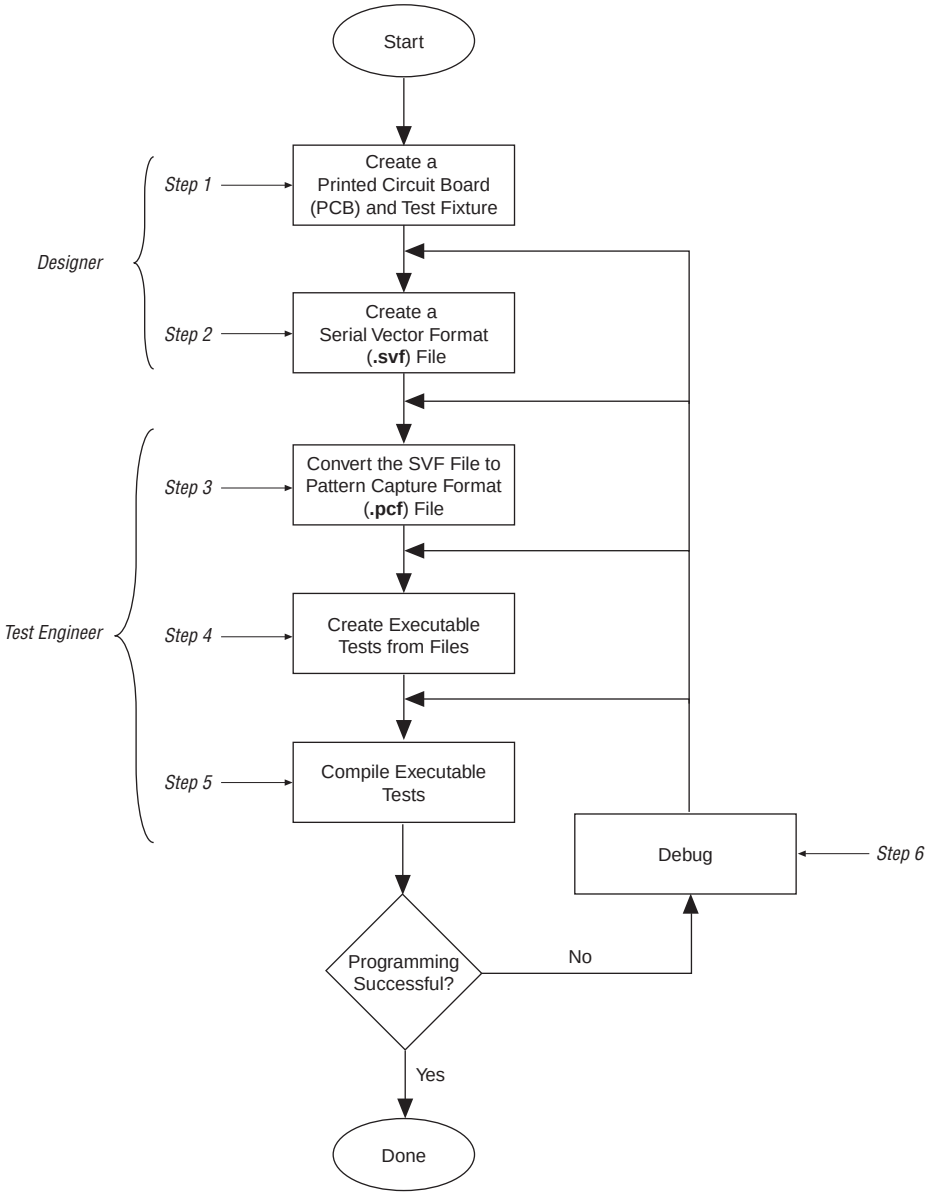
## Device Support

When programming MAX II devices together with devices from other families using the Agilent 3070 tester, ensure that all devices in the chain can be programmed using the tester.

# Agilent 3070 Development Flow without the PLD ISP Software

Programming devices with the Agilent 3070 tester (using a Serial Vector Format (**.svf**) File) without Agilent's PLD ISP software requires the following steps. Refer to Figure 15–1.

**Figure 15–1.** Agilent 3070 Development Flow for In-System Programming Using SVF File without PLD ISP



## Step 1: Create a PCB and Test Fixture

Before starting test development, the first step to successful in-system programming is the proper layout of the board and the subsequent creation of the test fixture.

### Creating the PCB

The following recommendations highlight important areas of PCB design issues:

- The TCK signal trace should be treated as carefully as a clock tree. TCK is the clock for the entire Joint Test Action Group (JTAG) chain of devices. These devices are edge-triggered on the TCK signal, so it is imperative that this signal be protected from high-frequency noise and have good signal integrity. Ensure that the signal meets the tR and tF parameters specified in the device data sheet.
- Add a pull-down resistor to TCK. The TCK signal should be held low through a pull-down resistor in-between PCF downloads. For more information about pattern capture format (PCF) downloads, refer to "Step 2: Create a Serial Vector Format File". You should hold TCK low because the Agilent 3070 drivers go into a "high-Z" state in-between tests and briefly drive low as the next PCF is applied. When the TCK line "floats", the programming data stream is corrupted and the device is not programmed correctly.
- Provide VCC and GND test access points for the nails of the test fixture. During operation, there should be enough access points to allow quiet PCB operation. Having too few access points results in a noisy system that can disrupt JTAG scans.
- Turn off on-board oscillators. During programming, on-board oscillators should have the ability to be electrically turned off to reduce system noise.
- Add external resistors to pull outputs to a defined logic level during programming.

Output pins are tri-stated during programming and are pulled up by a weak internal resistor. However, Altera recommends that signals requiring a pre-defined level be externally forced to the appropriate level using an external resistor.

For more information about board design for ISP, refer to the *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*.

### Creating the Fixture

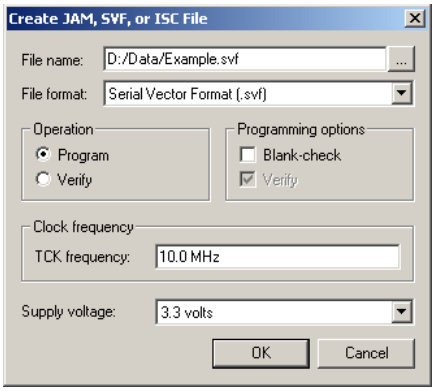
Providing a clean interface between the test fixture and the target board is essential for successful in-system programming. To provide a clean interface, use short wires in the test fixture to improve the TCK connection. Longer wires can introduce inductive noise into the system, which can disrupt programming. The wire connecting TCK should be no longer than 1 inch. Use the Agilent Fixture Consultant to manage the layout and creation of the test fixture (see the Agilent Board Test Family Manual).

## Step 2: Create a Serial Vector Format File

The Quartus II software generates SVF Files for programming one or more devices. When targeting multiple devices in the same MAX II CPLD family, the Quartus II software automatically generates one SVF File to program the devices concurrently. Therefore, the programming time for all of the devices approaches the programming time for the largest CPLD device in the IEEE Std. 1149.1 JTAG chain.

Figure 15–2 shows the **Create JAM, SVF, or ISC File** dialog box (File menu), which is used to generate the SVF File.

**Figure 15–2.** Create JAM, SVF, or ISC File Dialog Box




Before creating the SVF File, you must open the Programmer in the Quartus II and add the Programmer Object File (**.pof**) for all the devices in the chain into the programmer. Each POF corresponds to a targeted device, respectively.

In the **Create JAM, SVF, or ISC File** dialog box, the value in the TCK frequency box should match the frequency that TCK runs at during the test. If you enter a different frequency from the one used in actual testing, programming may fail or you may experience an excessively long programming time.

You can also select whether to perform a program or verify operation and optionally verify or blank-check the device by turning on programming options. Altera recommends generating SVF Files that include verify vectors, which ensure that programming failures are identified and a limited amount of additional programming time is used. You can generate the necessary SVF File based on the scan-chain topology of the board and the Altera devices to be programmed. Once the SVF File is generated, it can be given to test engineers for development.

If a device must be programmed independently, you can generate individual SVF Files for each Altera device in the chain. When creating the SVF File for a single device in the chain, specify the POF for the device and leave the rest of the devices set to <none>. This can be done by selecting **Add Device** in the Programmer. These devices are bypassed during programming. Repeat this process until all targeted devices have an SVF File.

## Step 3: Convert SVF Files to PCF Files

You must convert the SVF Files to PCF Files for use with the Agilent 3070 tester with the Altera **svf2pcf** conversion utility. The **svf2pcf** utility can create multiple PCF Files for one device chain; running the utility allows you to specify the number of vectors per file. The amount of memory used by the resulting files varies depending on the data. The Agilent 3070 digital compiler looks for repeating patterns of vectors and optimizes the directory and sequences RAM on the tester control card to apply the maximum number of vectors before re-loading the files. The number of vectors in a compiled PCF File ranges from 100,000 to over one million, depending on the size and density of the targeted devices.

You can download the **svf2pcf** conversion utility from the Agilent ISP Support website at www.altera.com.

## Step 4: Create Executable Tests from Files

Creating digital tests for programming a chain of devices with the Agilent 3070 tester requires the following steps:

1. Create the library for the target device or scan chain.
2. Run the Test Consultant.
3. Create digital tests.
4. Create the wirelist information for the tests.
5. Modify the test plan.

### Create the Library for the Target Device or Scan Chain

The initial program development for the board contains a setup-only node test library for the ISP boundary-scan chain interface. The test library ensures that Agilent 3070 tester resources are reserved in the test fixture for programming the targeted devices. If only one target device is on the board and it is not part of a boundary-scan chain (isolated), use a pin library; otherwise, use a node library. If using a pin library, you must describe every device pin. Do not include test vectors in a test library.

The following code example shows a setup-only node test library.

```
!Setup only test for the boundary scan chain
assign  TCK  to  nodes   "TCK"! Node name for the TCK pin
assign  TMS  to  nodes   "TMS"! Node name for the TMS pin
assign  TDI  to  nodes   "TDI"! Node name for the TDI pin
assign  TDO  to  nodes   "TDO"! Node name for the TDO pin
inputs  TCK, TMS, TDI
outputs TDO
pcf order is TCK, TMS, TDI, TDO! The order is defined by the program
that
    ! generates the PCF files.
```

Mark the `TCK` and `TMS` boundary-scan nodes as `CRITICAL` in the Board Consultant. This critical attribute minimizes the nodes' wire length in the test fixture.

### Run the Test Consultant

Run the Test Consultant to create all of the files for new board development. Once the Test Consultant finishes running with this
setup-only test library, it creates an executable test (without vectors) with the correct fixture wiring resource information. Use this file as a template to create the executable test's source code.

### Create Digital Tests

Create the digital tests, which are required to program the device(s), by copying the executable template to the desired program names. For example, if **svf2pcf** created four PCF Files, copy the template file to four executable tests (for example, `prog_a`, `prog_b`, `prog_c`, and `prog_d`) in the digital directory.

Add these test names to your **testorder** file and mark them permanent using the following syntax:

```
test digital "prog_a";  permanent
test digital "prog_b";  permanent
test digital "prog_c";  permanent
test digital "prog_d";  permanent
```

### Create the Wirelist Information for the Tests

Compile these executable tests to generate object files (see "Modify the Test Plan") for the setup only versions of the tests. Run **Module Pin Assignment** to create the necessary entries in the **wirelist** file.

Next, modify the executable tests so that they contain the vectors to program the target device. An `include` statement can be used in the executable test, or the vectors can be merged into the file. Use the following syntax for the `include` statement, which should be the last statement in the executable test.

```
include "pcf1"
```

Remember that the PCF File must reside in the digital directory and must be a digital file. To ensure that the digital file is in the correct directory, run the following command on the BT-Basic command line:

```
load  digital  "digital/pcf1"  |  re-save
```

You can also use the chtype command at a shell prompt to verify the location of the file:

```
chtype  -n6  digital/pcf1
```

Repeat this step for each PCF File.

### Modify the Test Plan

Add the test statements to the test plan using the following syntax:

```
test  "digital/prog_a" ! First program file
test  "digital/prog_b" ! Second program file
test  "digital/prog_c" ! Third program file
test  "digital/prog_d" ! Fourth program file
```

Keep the test execution in the same order in which the SVF File was split. For example, if the SVF File was split into four files (**pcf1**, **pcf2**, **pcf3**, and **pcf4**), the tests must be executed in the order that they split (execute `prog_a` followed by `prog_b` followed by `prog_c` followed by `prog_d`). If the order is not preserved, the device(s) will fail to program correctly.

## Step 5: Compile the Executable Tests

Altera recommends batch-driven compilation using either BT-Basic or a UNIX shell. See the following batch file code in BT-Basic (assuming four executable tests to program the target device and generation of debugging object code):

```
compile  "digital/prog_a" ;  debug
compile  "digital/prog_b" ;  debug
compile  "digital/prog_c" ;  debug
compile  "digital/prog_d" ;  debug
```

This file should be saved in the board directory to allow engineering changes to take place at a later date. See the corresponding shell script (–D option generates debugging information):

```
dcomp  -D  digital/prog_a
dcomp  -D  digital/prog_b
dcomp  -D  digital/prog_c
dcomp  -D  digital/prog_d
```

Compile times can be long, depending on the number of PCF vectors contained in the source files, the type of controller, and controller loading. Altera recommends using a batch file to automate the compilation of the ISP tests.

If a boundary-scan chain containing Altera devices is defined, only the Altera devices will be programmed when the PCF vectors have been applied to the JTAG interface.

## Step 6: Debug the Test

Once the executable tests have been created, the test system can be debugged. The applied vector set ensures that the device is programmed correctly by verifying the contents of the device. The programming algorithm uses the `TDO` pin to check the bitstream coming from the device. If any vector does not match the expected value, the test fails, indicating one of two things:

- The device ID does not match what is expected. This scenario is evident if the failure occurs at the beginning of the first test.
- Device programming failed.

Because many vectors are verified, it may not be practical to sift through each vector to determine the cause of the failure. Use the following troubleshooting guidelines if the device fails to program:

- Check the pull-down resistor in the test fixture. The design engineer may have placed pull-up resistors on the board for the `TCK` pin. If the pull-down resistor is too large, the `TCK` pin may be above the device's threshold for a logic low. Adjust the value of the resistor accordingly. See the appropriate device family data sheet for the specification on input logic levels.

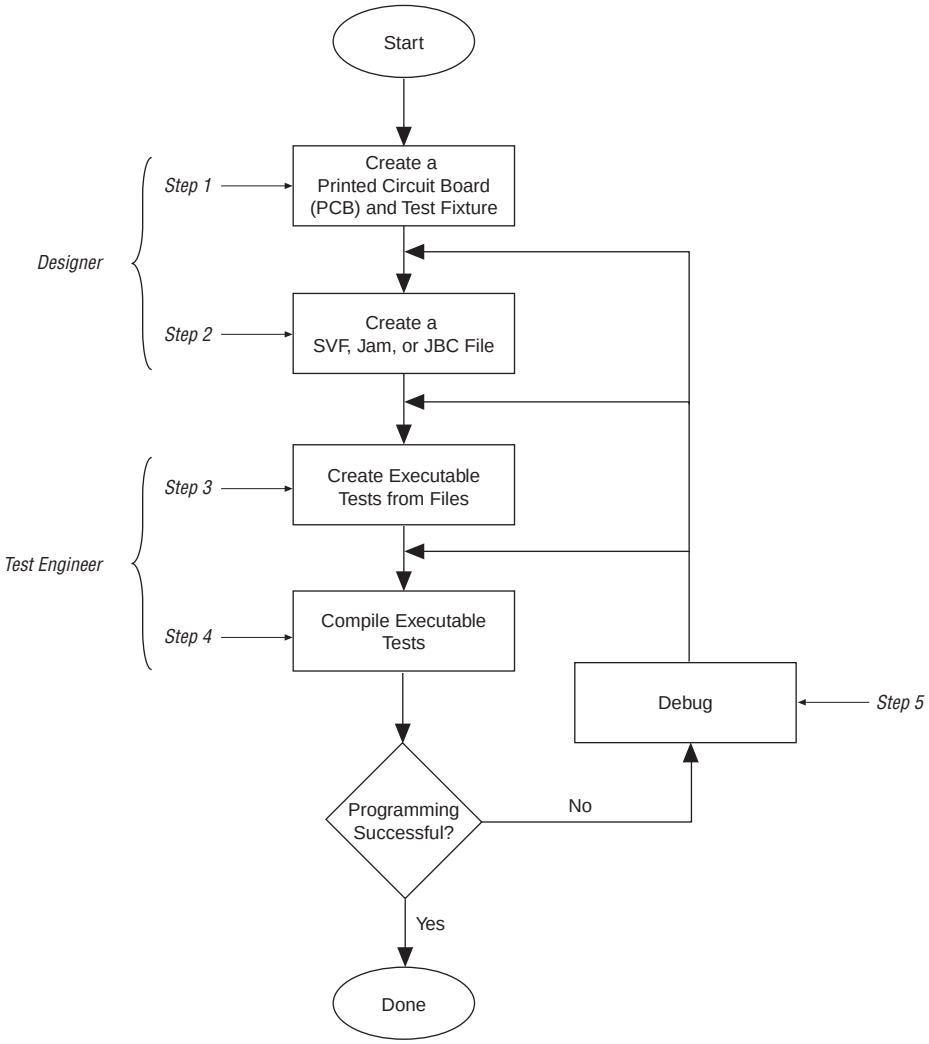
- If an overpower error on the TCK pin occurs, check the value of the resistors because they may be too low for the test system to back-drive for an extended period of time.
- Ensure that the test execution order is correct. If the tests are executed out of order, the programming information is incorrect. Also, if the same test is executed twice in a row, the target device will be out of sequence and will not receive the correct programming information.
- Ensure that the actual vectors match the expected values for the input pins (TCK, TMS, and TDI). If they are not the same, the tests may need to be recompiled.
- Ensure that the pcf order statement in the test matches the order of the PCF code generated in "Step 2: Create a Serial Vector Format File" on page 15–4. If they do not match, the order must be changed and the tests recompiled.
- If possible, verify that the device is programmed correctly by using the Quartus II software, the ByteBlaster™ II download cable, and the POF that was used to generate the SVF File. This action is not practical in a production situation, but is useful during test development and debugging.
- If you need to isolate an individual device, you can generate an individual SVF File for each targeted Altera device in the chain. The process of generating the SVF Files is explained in "Step 2: Create a Serial Vector Format File" on page 15–4. This process is useful when a verification error occurs and more than one Altera device is programmed in the chain.
- If you still have problems, look at the boundary-scan chain definition. Make sure that the number of bits for the instruction register are specified correctly for each device in the chain. If an incorrect number of bits have been defined for any device in the chain, the programming test will fail.

Once the test is running smoothly, the board is ready for production programming. Altera recommends saving the PCF Files and object code for back-up purposes. Use a compression program to minimize the size of the stored binaries and files.

## Development Flow for Agilent 3070 with PLD ISP Software

Programming devices with the Agilent 3070 tester and PLD ISP software is slightly different than the steps in Figure 15–1. Figure 15–3 shows the development flow using the Agilent 3070 tester with Agilent's optional PLD ISP software.

**Figure 15–3.** Agilent 3070 Development Flow for In-System Programming with Agilent's PLD ISP Software




Some advantages of using the Agilent PLD ISP software over the SVF2PCF flow for device programming are:

- The tester can support the programming of devices using SVF, Jam STAPL, or JBC file formats directly (that is, no conversion to PCF or VCL).
- The Agilent 3070 digital test to program a device is only one file.
- Pull-up and pull-down resistors are not required on the TCK and TMS lines in the fixture of the tester since the device programming executes entirely as one test.
- The size of the digital test source file as well as the compiled object file is much smaller than with the SVF2PCF solution.

- Execution time for larger CPLDs and configuration devices is faster as only a single digital test file is executed.

With Agilent's PLD ISP software, a Jam Byte-Code Player is implemented in the Control XTP card of the tester. This allows users to program devices using JBC files created directly from Quartus II. The tester also supports Jam or SVF files as it has a JBC compiler to compile these files for programming. The Jam Byte-Code Player is executed via the microcontroller on the Control XTP card and allows users to apply vectors algorithmically rather than executing a sequence of vectors. The Jam Byte-Code Player reads the programming and erase pulse width registers of the devices and uses those values in the programming and erase algorithms.

## Programming Times

Programming times on the Agilent 3070 are very consistent. The only variable is the TCK frequency, which affects programming times. The faster the clock, the less time is spent shifting data into the device. The programming time is a function of the TCK clock rate. MAX II devices support TCK clock rates up to 18 MHz.

## Guidelines

While using the Agilent 3070 tester for programming, use the following guidelines:

- Use caution if a pin library is used to describe the target device in a stand-alone boundary-scan chain. Altera does not recommend describing all of the ISP device's I/O pins as bidirectional. This practice uses a large number of hybrid card channels and potentially causes a fixture overflow error when developing the test.
- Do not include PCF vectors in the test library. Use a setup-only node library. Creating a test library with PCF vectors creates a large library object file and results in a much slower test development time. This delay occurs because the integrated program generator (IPG) looks at the entire vector set of the library object to determine if vectors need to be commented out due to conflicts. Library object compiles are different from executable compiles. Additionally, the IPG may fail due to the large library object file.
- To save time and disk space, generate SVF Files that include a verify in the programming operation. This process integrates verification vectors into one step, minimizing the amount of work in the test development process. This integrated verify accurately captures any programming errors; therefore, it is not necessary to add an additional stand-alone verify in the test sequence.
- While this document describes how to generate a test to apply vectors to the device for programming, a boundary-scan description language (BSDL) file is required to functionally test the device. If you need to perform a boundary-scan test or functional test, generate a BSDL file for the programmed state of the target device that contains the pin configuration information (for example, which pins are inputs, outputs, or bidirectional pins). Use the Agilent 3070 boundary-scan software to generate a test.

For more information about Altera's support for boundary-scan testing, refer to the *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*.

# Conclusion

Altera provides complete solutions for programming all MAX II devices using the Agilent 3070 test system. All MAX II devices can be programmed together with other ISP-capable devices. With software and device support, the opportunity for cutting costs and increasing manufacturing productivity is available to any Agilent 3070 user.

# Referenced Documents

This chapter references the following documents:

- *IEEE 1149.1 (JTAG) Boundary-Scan Testing for MAX II Devices* chapter in the *MAX II Device Handbook*
- *In-System Programmability Guidelines for MAX II Devices* chapter in the *MAX II Device Handbook*

# Document Revision History

Table 15–1 shows the revision history for this chapter.

**Table 15–1.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 1.5 | ■ Updated New Document Format. | — |
| December 2007, version 1.4 | ■ Added "Referenced Documents" section. | — |
| December 2006, version 1.3 | ■ Added document revision history. | — |
| June 2005, version 1.2 | ■ Text edit to the "Programming Times" section (25 MHz to 18 MHz). | — |
| January 2005, version 1.1 | ■ Previously published as Chapter 16. No changes to content. | — |

# Section V. Design Considerations

This section provides information for MAX® II design considerations.

This section includes the following chapters:

- Chapter 16, Understanding Timing in MAX II Devices
- Chapter 17, Understanding and Evaluating Power in MAX II Devices

## Revision History

Refer to each chapter for its own specific revision history. For information on when each chapter was updated, refer to the Chapter Revision Dates section, which appears in the complete handbook.

© October 2008 Altera Corporation

# 16. Understanding Timing in MAX II Devices

MII51017-2.1

## Introduction

Altera® devices provide predictable device performance that is consistent from simulation to application. Before programming a device, you can determine the worst-case timing delays for any design. You can approximate propagation delays with either the Quartus® II Timing Analyzer or the timing models given in this chapter and the timing parameters listed in individual device data sheets.

For the most precise timing results, you should use the Quartus II Timing Analyzer, which accounts for the effects of the secondary factors as mentioned later in this chapter.

This chapter defines external and internal timing parameters, and illustrates the timing models for the MAX® II device family.

Familiarity with device architecture and characteristics is assumed. Refer to specific device or device family data sheets in this handbook for a complete description of the architecture, and for the specific values of the timing parameters listed in this chapter.

This chapter contains the following sections:


- "External Timing Parameters" on page 16–1
- "Internal Timing Parameters" on page 16–2
- "Internal Timing Parameters for MAX II UFM" on page 16–3
- "Timing Models" on page 16–4
- "Calculating Timing Delays" on page 16–5
- "Programmable Input Delay" on page 16–7
- "Timing Model versus Quartus II Timing Analyzer" on page 16–7


## External Timing Parameters

External timing parameters represent actual pin-to-pin timing characteristics. Each external timing parameter consists of a combination of internal timing parameters. You can find the values of the external timing parameters in the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*. These external timing parameters are worst-case values, derived from extensive performance measurements and ensured by testing. All external timing parameters are shown in bold type. Table 16–1 defines external timing parameters for the MAX II family.

**Table 16–1.** External Timing Parameters

| Parameter | Description |
|---|---|
| $t_{PD1}$ | Pin-to-pin delay for the worst case I/O placement with full a diagonal path across the device with combinational logic implemented in a single look-up table (LUT) in a logic array block (LAB) adjacent to output pin. Fast I/O Connection is used from the adjacent logic element (LE) to the output pin. |
| $t_{PD2}$ | Pin-to-pin delay for the best case I/O placement with combinational logic (2-input AND gate) implemented in a single edge LE adjacent to the input pin. The longest pin path of the two inputs is shown. Fast I/O Connection is used from the adjacent LE to the output pin. |
| $t_{CLR}$ | Time to clear register delay. The time required for a low signal to appear at the external output, measured from the input transition. |
| $t_{SU}$ | Global clock setup time. The time that data must be present at the input pin before the global (synchronous) clock signal is asserted at the clock pin. |
| $t_{H}$ | Global clock hold time. The time that data must be present at the input pin after the global clock signal is asserted at the clock pin. |
| $t_{CO}$ | Global clock to output delay. The time required to obtain a valid output after the global clock is asserted at the clock pin. |
| $t_{CNT}$ | Minimum global clock period. The minimum period maintained by a globally clocked counter. |

# Internal Timing Parameters

Within a device, the timing delays contributed by individual architectural elements are called internal timing parameters, which cannot be measured explicitly. All internal parameters are shown in italic type. Table 16–2 defines the internal timing microparameters for the MAX II device family.

**Table 16–2.** Internal Timing Microparameters (Part 1 of 2)

| Parameter | Description |
|---|---|
| $t_{LUT}$ | LE combinational LUT delay for data-in to data-out. |
| $t_{COMB}$ | Combinational path delay. The delay from the time when a combinational logic signal from the LUT bypasses the LE register to the time it becomes available at the LE output. |
| $t_{CLR}$ | LE register clear delay. The delay from the assertion of the register's asynchronous clear input to the time the register output stabilizes at logical low. |
| $t_{PRE}$ | LE register preset delay. The delay from the assertion of the register's asynchronous preset input to the time the register output stabilizes at logical high. |
| $t_{SU}$ | LE register setup time before clock. The time required for a signal to be stable at the register's data and enable inputs before the register clock rising edge to ensure that the register correctly stores the input data. |
| $t_{H}$ | LE register hold time after clock. The time required for a signal to be stable at the register's data and enable inputs after the register clock's rising edge to ensure that the register correctly stores the input data. |
| $t_{CO}$ | LE register clock-to-output delay. The delay from the rising edge of the register's clock to the time the data appears at the register output. |
| $t_{C}$ | Register control delay. The time required for a signal to be routed to the clock, preset, or clear input of an LE register. |
| $t_{FASTIO}$ | Combinational output delay. $t_{FASTIO}$ is the time required for a combinational signal from the LE adjacent to the I/O block using the fast I/O connection. |
| $t_{IN}$ | I/O input pad and buffer delay. The $t_{IN}$ applies to I/O pins used as inputs. |
| $t_{GLOB}$ | $t_{GLOB}$ applies to GCLK pins when used for global signals. $t_{GLOB}$ is the delay required for a global signal to be routed from the GCLK pins to the LAB column clocks through the global clock network. |

**Table 16–2.** Internal Timing Microparameters (Part 2 of 2)

| Parameter | Description |
|---|---|
| $t_{IOE}$ | Internal generated output enable delay. The delay from an internally generated signal on the interconnect to the output enable of the tri-state buffer. |
| $t_{DL}$ | Input routing delay. The delay incurred from the row I/O pin used as input to the LE adjacent to it. |
| $t_{IODR}$ | Output data delay for the row interconnect. The delay incurred by signals routed from an interconnect to an I/O cell. |
| $t_{OD}$ | Output delay buffer and pad delay. Refer to *Timing Model and Specifications* section in the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* for delay adders associated with different I/O standards, drive strengths, and slew rates. |
| $t_{XZ}$ | Output buffer disable delay. The delay required for high impedance to appear at the output pin after the output buffer's enable control is disabled. Refer to *Timing Model and Specifications* section in the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* for delay adders associated with different I/O standards, drive strengths, and slew rates. |
| $t_{ZX}$ | Output buffer enable delay required for the output signal to appear at the output pin after the tri-state buffer's enable control is enabled. Refer to *Timing Model and Specifications* section in the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* for delay adders associated with different I/O standards, drive strengths, and slew rates. |
| $t_{C4}$ | Delay for a column interconnect with average loading. The $t_{C4}$ covers a distance of four LAB rows. |
| $t_{R4}$ | Delay for a row interconnect with average loading. The $t_{R4}$ covers a distance of four LAB columns. |
| $t_{LOCAL}$ | Local interconnect delay. |

# Internal Timing Parameters for MAX II UFM

Timing parameters for MAX II user flash memory (UFM) are the timing delays contributed by the UFM architectural elements, which cannot be measured explicitly. All timing parameters are shown in italic type. Table 16–3 defines the timing microparameters for MAX II UFM.

**Table 16–3.** Internal Timing Microparameters for MAX II UFM (Part 1 of 2)

| Parameter | Description |
|---|---|
| $t_{ASU}$ | Address register shift signal setup to address register clock. |
| $t_{AH}$ | Address register shift signal hold from address register clock. |
| $t_{ADS}$ | Address register data in setup to address register clock. |
| $t_{ADH}$ | Address register data in hold from address register clock. |
| $t_{DSS}$ | Data register shift signal setup to data register clock. |
| $t_{DSH}$ | Data register shift signal hold from data register clock. |
| $t_{DDS}$ | Data register data in setup to data register clock. |
| $t_{DDH}$ | Data register data in hold from data register clock. |
| $t_{DCO}$ | Delay incurred from the data register clock to data register output when shifting the data out. |
| $t_{DP}$ | PROGRAM signal to data clock hold time. |
| $t_{PB}$ | Maximum delay between PROGRAM rising edge to UFM BUSY signal rising edge. |
| $t_{BP}$ | Minimum delay allowed from UFM BUSY signal going low to PROGRAM signal going low. |
| $t_{PPMX}$ | Maximum length of busy pulse during a program. |
| $t_{AE}$ | Minimum ERASE signal to address clock hold time. |

**Table 16–3.** Internal Timing Microparameters for MAX II UFM (Part 2 of 2)

| Parameter | Description |
|---|---|
| $t_{EB}$ | Maximum delay between ERASE rising edge to UFM BUSY signal rising edge. |
| $t_{BE}$ | Minimum delay allowed from UFM BUSY signal going low to ERASE signal going low. |
| $t_{EPMX}$ | Maximum length of busy pulse during an erase. |
| $t_{RA}$ | Maximum read access time. The delay incurred between the DRSHFT signal going low to the first bit of data observed at the data register output. |
| $t_{OE}$ | Delay from OSC_ENA signal reaching UFM to rising clock of OSC leaving the UFM. |
| $t_{OSCS}$ | Maximum delay between the OSC_ENA rising edge to the ERASE/PROGRAM signal rising edge. |
| $t_{OSCH}$ | Minimum delay allowed from the ERASE/PROGRAM signal going low to the OSC_ENA signal going low. |

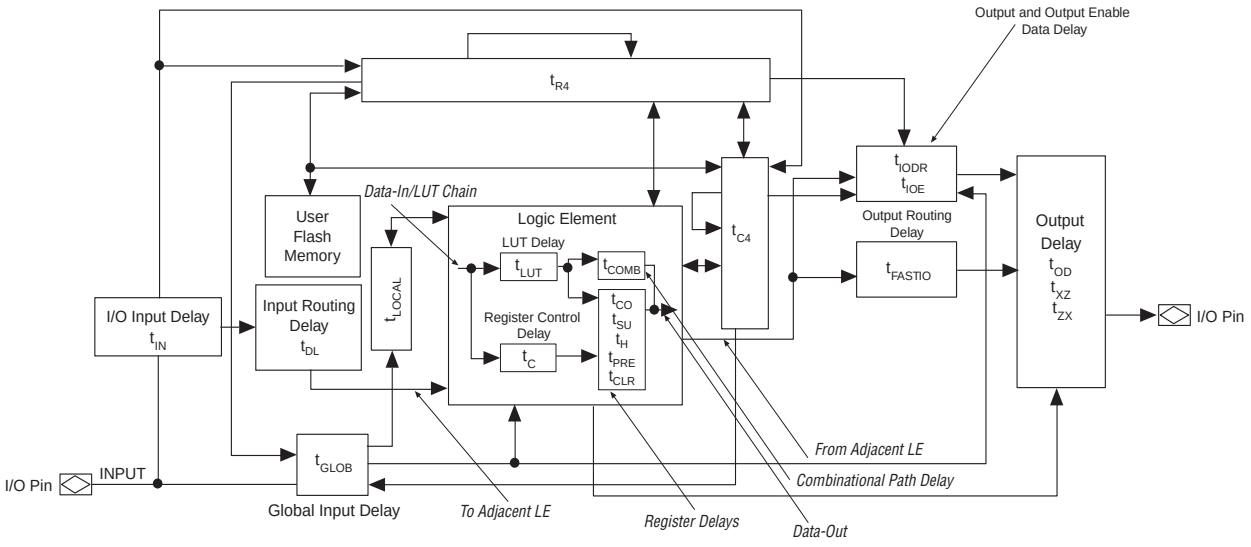
# Timing Models

Timing models are simplified block diagrams that illustrate the delays through Altera devices. Logic can be implemented on different paths. You can trace the actual paths used in your design by examining the equations listed in the Quartus II Report File (**.rpt**) for the project. You can then add up the appropriate internal timing parameters to estimate the delays through the device.

The MAX II architecture has a globally routed clock. The MultiTrack interconnect ensures predictable performance, accurate simulation, and accurate timing analysis across all MAX II device densities and speed grades.

Figure 16–1 shows the timing model for MAX II devices. The timing model is the preliminary version which is subject to change. The final version of the timing model will be released once available.

**Figure 16–1.** MAX II Device Timing Model

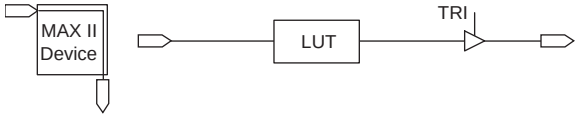


# Calculating Timing Delays

You can calculate approximate pin-to-pin timing delays for MAX II devices with the timing model shown in Figure 16–1 and by referring to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*. Each external timing parameter is calculated from a combination of internal timing parameters. Figure 16–2 through Figure 16–6 show the external timing parameters for the MAX II device family. To calculate the delay for a signal that follows a different path through the MAX II device, refer to the timing model to determine which internal timing parameters to add together.

For the most precise timing results, use the Quartus II Timing Analyzer, which accounts for the effects of secondary factors such as placement and fan-out.

**Figure 16–2.** External Timing Parameter ($t_{PD1}$) *Note (1)*




**Note to Figure 16–2:**

(1) $t_{PD1} = t_{IN} + N \times t_{R4}/4 + M \times t_{C4}/4 + t_{LUT} + t_{COMB} + t_{FASTIO} + (t_{OD} + \Delta t_{OD})$

Table 16–4 lists the numbers of LABs according to device density.

**Table 16–4.** Numbers of LABs According to Device Density

| Device Density | N LAB Rows | M LAB Columns |
|---|---|---|
| EPM240 | 4 | 6 |
| EPM570 | 7 | 12 |
| EPM1270 | 10 | 16 |
| EPM2210 | 13 | 20 |

$Dt_{OD}$ is the adder delay (see note to Figure 16–2) for the $t_{OD}$ microparameter when using an I/O standard other than 3.3-V LVTTL with 16 mA current strength.

Refer to the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* for adder delay values.

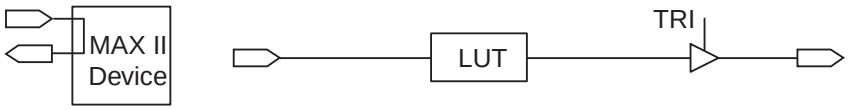
The following is an example:

$t_{PD1}$ for the EPM240 device using an I/O standard of 3.3-V LVTTL fast slew rate with a drive strength of 16 mA:

$t_{PD1} = t_{IN} + 4 \times t_{R4}/4 + 6 \times t_{C4}/4 + t_{LUT} + t_{COMB} + t_{FASTIO} + t_{OD}$……(a)

$t_{PD1}$ for the EPM240 device using an I/O standard of 2.5-V LVTTL fast slew rate with a drive strength of 7 mA: $t_{PD1}$ = (a) + ($Dt_{OD}$ of 2.5-V LVTTL fast slew 7 mA)

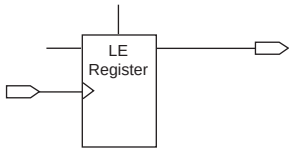
**Figure 16–3.** External Timing Parameter ($t_{PD2}$) *Note (1)*




**Note to Figure 16–3:**

(1) $t_{PD2} = t_{IN} + t_{DL} + t_{LUT} + t_{COMB} + t_{FASTIO} + (t_{OD} + \Delta t_{OD})$

**Figure 16–4.** External Timing Parameter ($t_{CO}$) *Note (1), (2)*

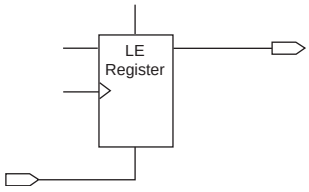



**Notes to Figure 16–4:**

(1) $t_{CO} = t_{GLOB} + t_C + t_{CO} + (N \times t_{R4}/4 + M \times t_{C4}/4) + (t_{IODC} \text{ or } t_{IODR}) + (t_{OD} + \Delta t_{OD})$

(2) The constants N and M are subject to change according to the position of the LAB in the entire device.

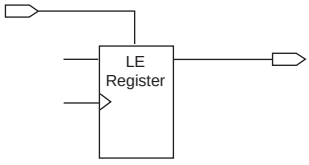
**Figure 16–5.** LE Register Clear and Preset Time ($t_{CLR}$) *Note (1)*




**Note to Figure 16–5:**

(1) $t_{CLR} = t_{GLOB} + t_C + t_{CLR} + (N \times t_{R4}/4 + M \times t_{C4}/4) + (t_{IODC} \text{ or } t_{IODR}) + (t_{OD} + \Delta t_{OD})$

**Figure 16–6.** LE Register Clear and Preset Time ($t_{PRE}$) *Note (1)*



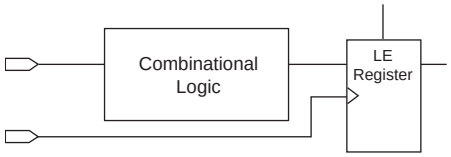

**Note to Figure 16–6:**

(1) $t_{PRE} = t_{GLOB} + t_{LOCAL} + t_C + t_{PRE} + (N \times t_{R4}/4 + M \times t_{C4}/4) + (t_{IODC} \text{ or } t_{IODR}) + (t_{OD} + \Delta t_{OD})$

## Setup and Hold Time from an I/O Data and Clock Input

The Quartus II software might insert additional routing delays from the input pin to the register input to ensure a zero hold time for the LE register. Altera recommends that you use the Quartus II Timing Analyzer to obtain the setup time and hold time. See Figure 16–7 and Figure 16–8.

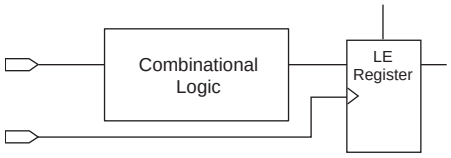
**Figure 16–7.** Setup and Hold Time ($t_{SU}$) *Note (1)*




**Note to Figure 16–7:**

(1) $t_{SU} = (t_{IN} + N \times t_{R4}/4 + M \times t_{C4}/4 + t_{LUT}) - (t_{GLOB} + t_C) + t_{SU}$

**Figure 16–8.** Setup and Hold Time ($t_H$) *Note (1)*




**Note to Figure 16–8:**

(1) $t_H = (t_{GLOB} + t_C) - (t_{IN} + N \times t_{R4}/4 + M \times t_{C4}/4 + t_{LUT}) + t_H$

For Figure 16–4 through Figure 16–8, the constants N and M are subject to change according to the position of LAB in the entire device for combinational logic implementation.

# Programmable Input Delay

The programmable input delay provides an option to add a delay to the input pin, guaranteeing a zero hold time. You can set this option in the Assignment Editor (Assignments menu) on a pin-by-pin basis. The following procedure shows how to turn on the input delay for the selected input pin in the Quartus II software:

1. Select input pin name in the design file.
2. Right-click and select **Locate** in the Assignment Editor.
3. Double-click the cell under Assignment Name and select **Input Delay from Pin to Internal Cells** in the pull-down list.
4. Double-click the **Value** cell to the right of the assignment name just made and enter 1.
5. On the File menu, click **Save**.

# Timing Model versus Quartus II Timing Analyzer

Hand calculations based on the timing model provide a good estimate of a design's performance. However, the Quartus II Timing Analyzer always provides the most accurate information on design performance because it takes into account secondary factors that influence the routing microparameters such as:

- Fan-out for each signal in the delay path
- Positions of other loads relative to the signal source and destination

- Distance between the signal source and destination
- Various interconnect lengths where some interconnects are truncated at the edge of the device

## Conclusion

The MAX II device architecture has predictable internal timing delays that can be estimated based on signal synthesis and placement. The Quartus II Timing Analyzer provides the most accurate timing information. However, you can use the timing model along with the timing parameters listed in the *DC and Switching Characteristics* chapter in the *MAX II Device Handbook* to estimate a design's performance before compilation. Both methods enable you to accurately predict your design's in-system timing performance.

## Referenced Documents

This chapter references the following document:

- *DC and Switching Characteristics* chapter in the *MAX II Device Handbook*

## Document Revision History

Table 16–5 shows the revision history for this chapter.

**Table 16–5.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 2.1 | ■ Updated New Document Format. | — |
| December 2007, version 2.0 | ■ Updated $t_{PD2}$ information in Table 16–1.<br>■ Added $t_{COMB}$ information in Table 16–2.<br>■ Updated Figure 16–1.<br>■ Updated *Note (1)* to Figure 16–2.<br>■ Updated "Calculating Timing Delays" section.<br>■ Added "Referenced Documents" section. | — |
| December 2006, version 1.4 | ■ Added document revision history. | — |
| January 2005, version 1.3 | ■ Previously published as Chapter 17. No changes to content. | — |
| December 2004, version 1.2 | ■ Added section Programmable Input Delay. | — |
| June 2004, version 1.1 | ■ Updated Table 16–1. Various parameter naming updates. | — |

# 17. Understanding and Evaluating Power in MAX II Devices

MII51018-2.1

## Introduction

Power consumption has become an important factor for CPLD applications with the increased use of CPLDs in low power designs. Overall low standby (static) and dynamic power is becoming increasingly important to reduce system power, and can be achieved with MAX® II devices which have low stand-by and dynamic power.

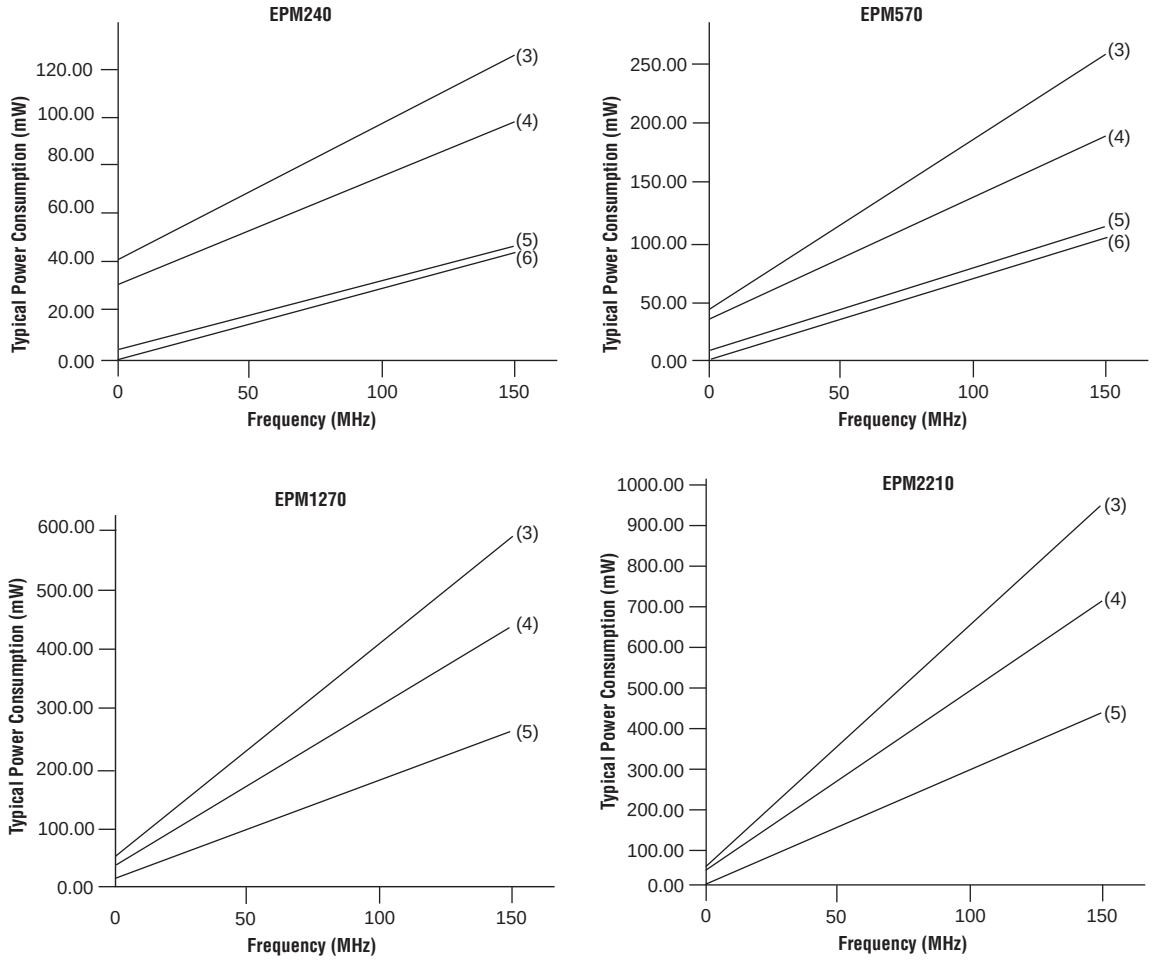
This chapter contains the following sections:


- "Power in MAX II Devices" on page 17–1
- "MAX II Power Estimation Using the PowerPlay Early Power Estimator" on page 17–3
- "PowerPlay Early Power Estimator Inputs" on page 17–3
- "Power Estimation Summary" on page 17–13
- "Power Saving Techniques" on page 17–15


## Power in MAX II Devices

Different from previous CPLD architectures, MAX II logic does not use sense amplifiers that require bias currents to amplify signal voltages within the device. Additionally, with the Quartus® II software, efficient implementation of most interconnects with local routing in MAX II devices significantly lowers the dynamic power. Figure 17–1 shows the typical power consumption versus frequency for MAX II devices. The power consumption (mWatts) provided is based on typical conditions using a pattern that fills a device with a 16-bit, loadable, enabled, up/down counter with no output load.

**Figure 17–1.** Power Consumption versus Frequency for MAX II Devices *(Note 1)*, *(2)*




**Notes to Figure 17–1:**

(1) Every device is fully utilized with 16-bit counters for power estimation.
(2) The MAX II and MAX IIG devices can operate up to 304 MHz.
(3) $V_{CCINT}$ = 3.3 V
(4) $V_{CCINT}$ = 2.5 V
(5) $V_{CCINT}$ = 1.8 V (MAX IIG)
(6) $V_{CCINT}$ = 1.8 V (MAX IIZ)

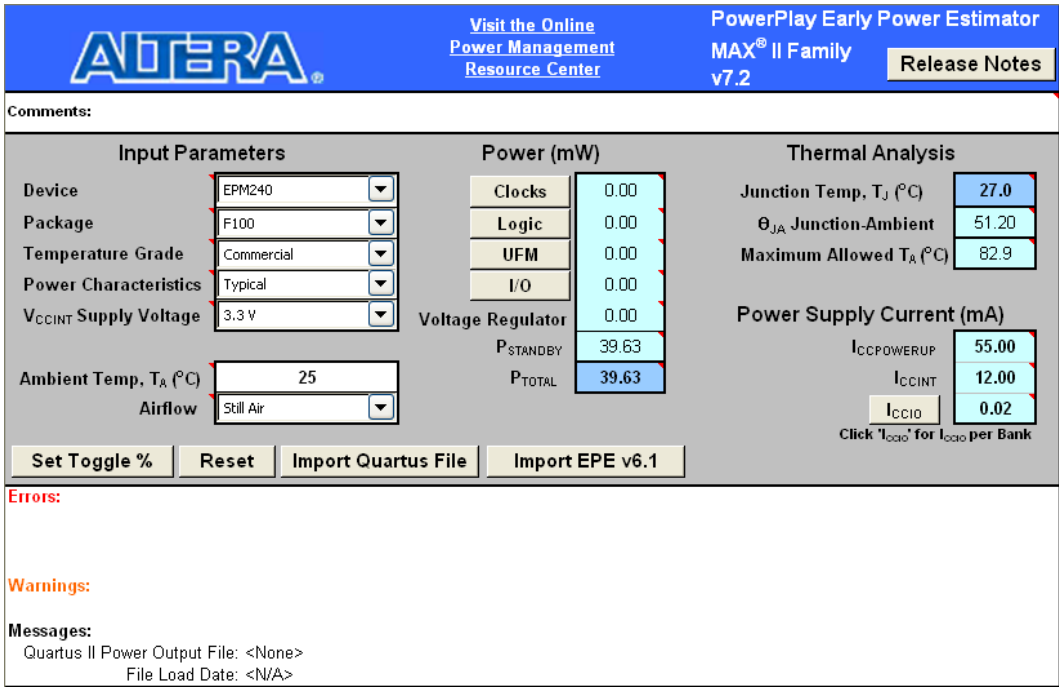
The power consumed in MAX II devices is dependent on the design. It is very important to complete a power evaluation early in the design process to ensure that the power dissipation by MAX II devices meets system requirements and specifications.

This chapter discusses how to evaluate and manage MAX II power using the MAX II PowerPlay Early Power Estimator spreadsheet, available at www.altera.com.

# MAX II Power Estimation Using the PowerPlay Early Power Estimator

The PowerPlay Early Power Estimator spreadsheet allows you to enter information into sections based on architectural features. The PowerPlay Early Power Estimator spreadsheet also provides a subtotal of power consumed by each architectural feature reported in each section in mWatts (mW). Figure 17–2 shows the overview of the MAX II PowerPlay Early Power Estimator summary worksheet.

**Figure 17–2.** MAX II PowerPlay Early Power Estimator




The power estimator results are based on estimated power data from device simulations and typical silicon measurements under nominal conditions. Results obtained should only be used as an estimation of power, not as a specification. The actual $I_{CC}$ must be verified during device operation, as this measurement is sensitive to the actual pattern in the device and the environmental operating conditions.

# PowerPlay Early Power Estimator Inputs

The following sections of the chapter explain what values you need to enter for the PowerPlay Early Power Estimator spreadsheet. The areas of entry in the PowerPlay Early Power Estimator spreadsheet include input parameters, clock, logic, UFM, and input/output (I/O) module.

## Input Parameters

Different MAX II devices consume different amounts of power for the same design. The larger the device, the more power it consumes because of a larger clock tree. In the Main section, you can enter the following parameters for the device and design:

- Device
- Package
- Temperature grade
- Power characteristics
- $V_{CCINT}$ supply
- Ambient temperature
- Airflow

Figure 17–3 shows the Input Parameter section in the PowerPlay Early Power Estimator spreadsheet.

**Figure 17–3.** Input Parameter Section

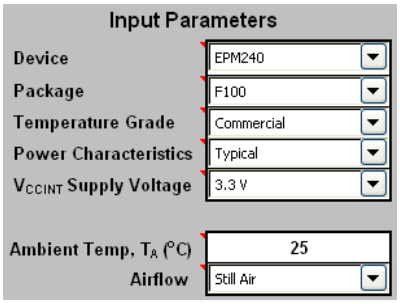



Table 17–1 describes the values that must be specified in the Input Parameter section of the PowerPlay Early Power Estimator spreadsheet.

**Table 17–1.** Input Parameter Section Information (Part 1 of 2)

| Input Parameter | Description |
|---|---|
| Device | Select your MAX II device. Larger devices have slightly higher clock dynamic power. MAX IIZ devices have the lowest $I_{CCINT}$ compared to the MAX II and MAX IIG devices because MAX IIZ devices have optimized circuitry to reduce $I_{CCINT}$. Compared to MAX II devices, MAX IIG devices use less power because they do not use the on-chip voltage regulator. |
| Package | Select the package that will be used. Larger packages provide a larger cooling surface and more contact points to the circuit board, leading to lower thermal resistance. Package selection does not affect power consumption. |
| Temperature Grade | Commercial devices have a maximum junction operating temperature of 85°C. Industrial devices offer 100°C operation while the MAX II automotive-grade devices can operate up to 125°C. This field affects the maximum junction temperature used in thermal calculations. |
| Power Characteristics | For MAX IIZ devices, you can select either typical or maximum power characteristics for the power estimation. The power characteristics are based on typical and theoretical worst-case silicon process. *Maximum* should be used for thermal design, while *Typical* gives you the estimation of the average use of the devices. |

**Table 17–1.** Input Parameter Section Information (Part 2 of 2)

| Input Parameter | Description |
|---|---|
| $V_{CCINT}$ Supply | The voltage of the $V_{CCINT}$ power supply. For MAX IIG and MAX IIZ devices, the supply voltage must be 1.8 V. For other devices, it can be either 2.5 V or 3.3 V. Devices with lower $V_{CCINT}$ have lower total standby power consumption. |
| Ambient Temperature | Enter the air temperature near the CPLD. This value can range from –40°C to 125°C, depending on the device temperature grade. This parameter is used to compute junction temperature based on power dissipation and thermal resistances through the top of the chip. |
| Airflow | Select an available ambient airflow in linear feet per minute (lfm) or meters per second (m/s). The options are still air, 100 lfm (0.5 m/s), 200 lfm (1.0 m/s), or 400 lfm (2.0 m/s). Increased airflow results in a lower junction-to-air thermal resistance, and thus lower junction temperature. |

## Clock Section

MAX II devices have four global clocks each. Each row in the Clock Domain subsection of the spreadsheet represents a clock network or a separate clock domain. You must enter the clock frequency ($f_{MAX}$) in MHz, the total fan-out for each clock network used, and the local clock enable percentage. Figure 17–4 shows the Clock section in the PowerPlay Early Power Estimator spreadsheet.

**Figure 17–4.** Clock Section

| Clock Domain | Clock Freq (MHz) | Total Fanout | Local Enable % | Total Power (mW) | User Comments |
|---|---|---|---|---|---|
| | 0.0 | 0 | 50% | 0.00 | |
| | 0.0 | 0 | 50% | 0.00 | |
| | 0.0 | 0 | 50% | 0.00 | |
| | 0.0 | 0 | 50% | 0.00 | |

Table 17–2 describes the parameters in the Clock section of the PowerPlay Early Power Estimator spreadsheet.

**Table 17–2.** Clock Section Information

| Column Heading | Description |
|---|---|
| Clock Domain | Enter a name for the clock network in this column (optional entry). |
| Clock Frequency (MHz) | Enter the frequency of the clock domain. The operating frequency for MAX II and MAX IIG is between 0 and 304 MHz. For MAX IIZ, the operating frequency is between 0 and 152 MHz. |
| Total Fanout | Enter the total number of logic element (LE) flipflops fed by this clock. The number of resources driven by every global clock is reported in the Fanout column of the Quartus II Compilation Report under Fitter > Resource Section > Global & Other Fast Signals > Fanout. |
| Local Enable % | Enter the average percentage of time that clock enable is high for destination flipflops. Local clock enables for flipflops in the LEs are promoted to logic array block (LAB)-wide signals. When a given flipflop is disabled, the LAB-wide clock is also disabled, cutting clock power in addition to power for downstream logic. This sheet models only the impact on clock tree power. |
| Total Power (mW) | Represents the total power dissipation due to clock distribution. |
| User Comments | Enter any comments (optional entry). |

## Logic Section

A design is a combination of several design modules operating at different frequencies and toggle rates. Each design module can have a different amount of logic. For the most accurate power estimation, partition the design into different design modules. You can partition your design by grouping modules by clock frequency, location, hierarchy, or entities. Figure 17–5 shows the logic section in the PowerPlay Early Power Estimator spreadsheet.

**Figure 17–5.** Logic Section

| | | | | Power (mW) | | | |
|---|---|---|---|---|---|---|---|
| Logic Module | Clock Freq (MHz) | # LEs | Toggle % | Routing | Block | Total | User Comments |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |
| | 0.0 | 0 | 12.5% | 0.00 | 0.00 | 0.00 | |

Each row in the Logic section represents a separate design module. Table 17–3 describes the parameters in the Logic section of the PowerPlay Early Power Estimator spreadsheet.

**Table 17–3.** Logic Section Information (Part 1 of 2)

| Column Heading | Description |
|---|---|
| Logic Module | Enter a name for each module of the design (optional entry). |
| Clock Frequency (MHz) | Enter a clock frequency (MHz). The operating frequency for MAX II and MAX IIG is between 0 and 304 MHz. For MAX IIZ, the operating frequency is between 0 and 152 MHz. A 100 MHz input clock with a 12.5% toggle means that each look-up table (LUT) or flipflop output toggles 12.5 million times per second (100 × 12.5%). |
| # LEs | Enter the number of LEs in this module. |
| Toggle % | Enter the average percentage of logic toggling on each clock cycle. The toggle percentage ranges from 0 to 100%. Typically, the toggle percentage is 12.5%, which is the toggle percentage of a 16-bit counter. To ensure you do not underestimate the toggle percentage, you can use a higher toggle percentage. Most logic toggles infrequently, and therefore toggle rates of <50% are more realistic.<br><br>For example, a TFF with its input tied to $V_{cc}$ has a toggle rate of 100% because its output is changing logic states on every clock cycle (see Figure 17–6). Figure 17–7 shows an example of a 4-bit counter. The first TFF with least significant bit (LSB) output cout0 has a toggle rate of 100% because the signal toggles on every clock cycle. The toggle rate for the second TFF with output cout1 is 50% since the signal only toggles on every two clock cycles. Consequently, the toggle rate for the third TFF with output cout2 and fourth TFF with output cout3 are 25% and 12.5%, respectively. Therefore, the average toggle percentage for this 4-bit counter is (100 + 50 + 25 + 12.5)/4 = 46.875%. |

**Table 17–3.** Logic Section Information (Part 2 of 2)

| Column Heading | Description |
|---|---|
| Routing | Represents the power dissipation due to estimated routing.<br><br>Routing power is highly dependent on placement and routing, which itself is a function of design complexity. The values shown are representative of routing power average based on experimentation on over 100 real-world designs.<br><br>Use the Quartus II PowerPlay Power Analyzer for detailed analysis based on the routing used in your design. |
| Block | Represents the power dissipation due to internal toggling of the LEs.<br><br>Logic block power is a function of the function implemented and relative toggle rates of the various inputs. The PowerPlay Early Power Estimator spreadsheet uses an estimate based on observed behavior across over 100 real-world designs.<br><br>Use the Quartus II PowerPlay Power Analyzer for an accurate analysis based on the exact synthesis of your design. |
| Total | Represents the total power dissipation. The total power dissipation is the sum of the routing and block power. |
| User Comments | Enter any comments (optional entry). |

**Figure 17–6.** T-Flipflop

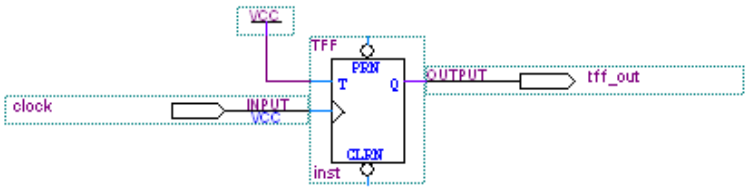
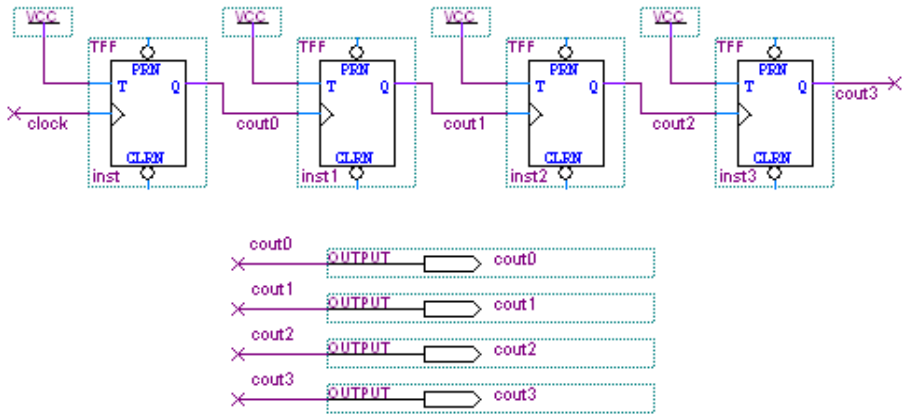



**Figure 17–7.** 4-Bit Counter



## UFM Section

When the design utilizes the UFM, the PowerPlay Early Power Estimator spreadsheet considers the time spent during read operations into the power estimation. Figure 17–8 shows the UFM section in the PowerPlay Early Power Estimator spreadsheet.

**Figure 17–8.** UFM Section

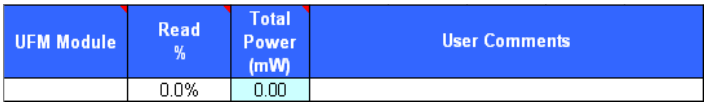



Table 17–4 describes the parameters in the UFM section of the PowerPlay Early Power Estimator spreadsheet.

**Table 17–4.** UFM Section Information

| Column Heading | Description |
|---|---|
| UFM Module | Enter a name for the UFM module in this column (optional entry). |
| Read % | Enter the percentage of time the UFM spends in Read mode. It takes 16 clock cycles to shift the serial data out after an internal UFM read so the read operation occurs less than 1/17 (or about 6%) of the time. The clock in this calculation is the UFM block's DRCLK signal. |
| Total Power (mW) | Total power dissipation due to reading from the UFM block (mW). Programming and erasing can only be performed a limited number of times over the life of the device so they do not contribute to average power. |
| User Comments | Enter any comments (optional entry). |

## I/O Section

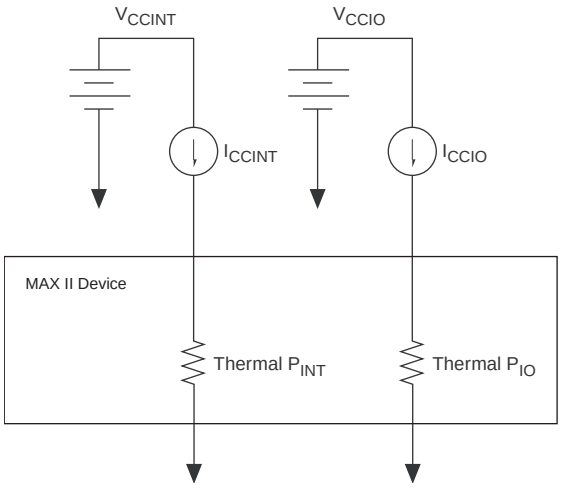
MAX II devices feature programmable I/O pins that support a wide range of industry I/O standards for increased design flexibility. The I/O section in the PowerPlay Early Power Estimator spreadsheet allows you to estimate the I/O pin power consumption based on the pin's I/O standards.

The total thermal power is the sum of the thermal power consumed by the device based on each power rail.

Thermal Power = Thermal $P_{INT}$ + Thermal $P_{IO}$

Figure 17–9 shows a graphical representation of the thermal power consumption.

**Figure 17–9.** Thermal Power Representation




The PowerPlay Early Power Estimator spreadsheet estimates the current for each I/O bank based on the $V_{CCIO}$ settings, if you specify the I/O bank for I/O pins in the I/O section. Figure 17–10 shows the I/O bank parameter settings.

**Figure 17–10.** I/O Bank Parameter Settings

| | $V_{CCIO}$ (V) | $I_{CCIO}$ (mA) |
|---|---|---|
| I/O Bank 1 | 1.5 | 0.01 |
| I/O Bank 2 | 1.5 | 0.01 |
| N/A | 1.5 | 0.00 |
| N/A | 1.5 | 0.00 |
| Unassigned | | 0.00 |

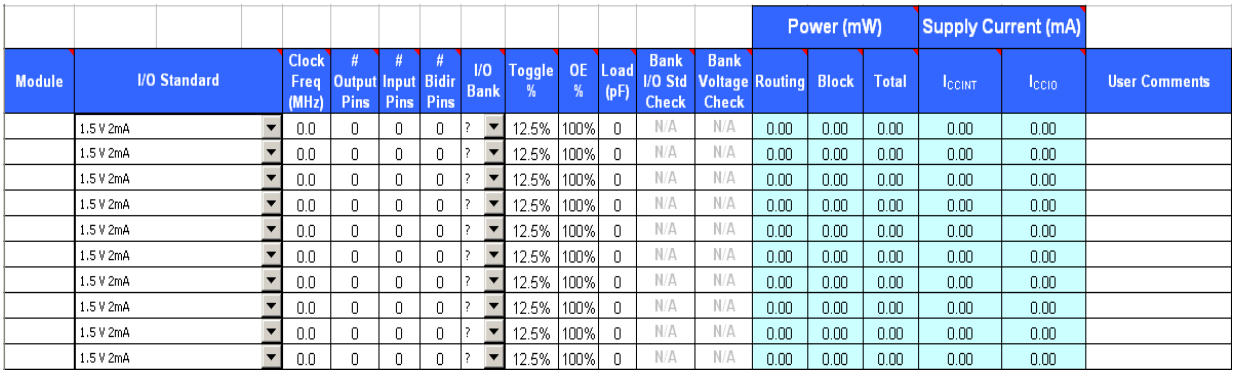
Table 17–5 describes the I/O bank parameters in the I/O section of the PowerPlay Early Power Estimator spreadsheet.

**Table 17–5.** I/O Bank Information

| Column Heading | Description |
|---|---|
| $V_{CCIO}$ | Select the $V_{CCIO}$ voltage for each bank. Used to cross-check selected I/O standards in I/O section for warning purposes. |
| $I_{CCIO}$ | Shows the total supply current due to the I/O pins in each I/O bank. |
| Unassigned | Represents the $I_{CCIO}$ of all I/O modules not assigned to an I/O bank. |

Each row in the I/O section represents a design module where the I/O pins have the same frequency, toggle percentage, average capacitive load, I/O standard, and I/O bank. Figure 17–11 shows the I/O section of the PowerPlay Early Power Estimator spreadsheet and Table 17–6 describes the I/O module parameters.

**Figure 17–11.** I/O Section



| | | | | | | | | | | | | Power (mW) | | | Supply Current (mA) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Module | I/O Standard | Clock Freq (MHz) | # Output Pins | # Input Pins | # Bidir Pins | I/O Bank | Toggle % | OE % | Load (pF) | Bank I/O Std Check | Bank Voltage Check | Routing | Block | Total | $I_{CCINT}$ | $I_{CCIO}$ | User Comments |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |
| | 1.5 V 2mA | 0.0 | 0 | 0 | 0 | ? | 12.5% | 100% | 0 | N/A | N/A | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | |

**Table 17–6.** I/O Section Information (Part 1 of 2)

| Column Heading | Description |
|---|---|
| Module | Enter a name for the module in this column (optional entry). |
| I/O Standard | Select the I/O standard for the input, output, or bidirectional pins in this module from the pull-down list. The calculated I/O power varies based on the I/O standard. |
| Clock Freq (MHz) | Enter the clock frequency (MHz). The operating frequency for MAX II and MAX IIG is between 0 and 304 MHz. For MAX IIZ, the operating frequency is between 0 and 152 MHz. A 100 MHz input clock with a 12.5% toggle means that each I/O pin toggles 12.5 million times per second (100 × 12.5%). |
| # Output Pins | Enter the number of output pins in this module. |
| # Input Pins | Enter the number of input pins in this module. |
| # Bidir Pins | Enter the number of bidirectional pins in this module.<br>An I/O pin configured as bidirectional but used only as an output consumes more power than one configured as an output-only, due to the toggling of the input buffer every time the output buffer toggles (they share a common pin). |
| I/O Bank | Select the I/O bank for the module. If you do not know which I/O bank the pins will be assigned to, leave the value as "?". Assigning the I/O module to a bank checks whether your I/O voltage assignments are compatible or not, allowing per-bank $I_{CCIO}$ reporting.<br>The PowerPlay Early Power Estimator spreadsheet does not take any I/O placement constraints into consideration except for I/O standard and bank match, and I/O voltage. |
| Toggle % | Enter the average percentage of output, bidirectional, and input pins toggling on each clock cycle. The toggle percentage ranges from 0 to 100% for output pins and can be up to 200% for input pins used as clocks because clocks toggle at twice the clock frequency.<br>Typically, the toggle percentage is 12.5%. To be more conservative, you can use a higher toggle percentage. |

**Table 17–6.** I/O Section Information (Part 2 of 2)

| Column Heading | Description |
|---|---|
| OE % | Enter the average percentage of time that:<br>■ The output I/O pins are enabled.<br>■ Bidirectional I/O pins are outputs and enabled.<br><br>During the remaining time:<br>■ Output I/O pins are tri-stated.<br>■ Bidirectional I/O pins are inputs.<br>This number must be a percentage between 0% and 100%. |
| Load (pF) | Enter the pin loading external to the chip (pF). This parameter only applies to output and bidirectional pins.<br>Pin and package capacitance is already included in the I/O model. Therefore, you only need to include off-chip capacitance in the Load parameter. |
| Bank I/O Std Check | Indicates whether the selected I/O standard is available on the selected I/O bank or not. Not all I/O banks can implement every I/O standard. |
| Bank Voltage Check | Indicates whether the selected I/O bank has a voltage compatible with the selected I/O standard or not. |
| Routing | Represents the power dissipation due to estimated routing.<br>Routing power is highly dependent on placement and routing, which itself is a function of design complexity. The values shown are representative of routing power based on experimentation on over 100 real-world designs.<br>Use the Quartus II PowerPlay Power Analyzer for detailed analysis based on the routing used in your design. |
| Block | Represents the power dissipation due to internal and load toggling of the I/O.<br>Use the Quartus II PowerPlay Power Analyzer for accurate analysis based on the exact I/O configuration of your design. |
| Total | Represents the total power dissipation. The total power dissipation is the sum of the routing and block power. |
| $I_{CCINT}$ | Represents the current drawn from the $I_{CCINT}$ rail. Powers internal digital circuitry and routing. |
| $I_{CCIO}$ | Represents the current drawn from this bank's $V_{CCIO}$ rail. |
| User Comment | Enter any comments (optional entry). |

## Other Input Information

There are three other buttons below the input parameters section: Set Toggle %, Reset, and Import Quartus File, as shown in Figure 17–12.

**Figure 17–12.** The Three Buttons

### Set Toggle %

Sets the toggle rate for the Logic Module and I/O Module.

### Reset

Clears all input values in the PowerPlay Early Power Estimator spreadsheet.

### Importing the Quartus II Early Power Estimator File

If you have created the user design, you can use the Quartus II software to generate the PowerPlay Early Power Estimator file and then import this file into the PowerPlay Early Power Estimator spreadsheet. This power estimation report file contains the device resource information and importing this file saves you time and effort otherwise spent manually entering information into the PowerPlay Early Power Estimator spreadsheet. You can manually change any of the values after importing the file.

To generate the PowerPlay Early Power Estimator file, first compile your design in the Quartus II software. After that, on the Project menu, click **Generate PowerPlay Early Power Estimator File**. The Quartus II software creates a PowerPlay Early Power Estimator file with the name <revision name>**_early_pwr.csv**.

For more information about generating the PowerPlay Early Power Estimator file in the Quartus II software, refer to the *PowerPlay Power Analysis* chapter in volume 3 of the *Quartus II Handbook*.

To import data into the PowerPlay Early Power Estimator spreadsheet, perform the following steps:

1. Click **Import Quartus File** in the PowerPlay Early Power Estimator spreadsheet.
2. Browse to a power estimation file generated from the Quartus II software. Click **OK**.

Clicking **OK** clears any user-entered values in the PowerPlay Early Power Estimator spreadsheet and populates the PowerPlay Early Power Estimator spreadsheet with device resource information from the specified power estimation file.

After importing a file, manually specify some of the input parameters in the main section. These input parameters include:

- $V_{CCINT}$ supply voltage
- Ambient temperature
- Airflow

The ambient temperature and airflow are used for thermal analysis only. Refer to the input parameters section for more information on these parameters.

The clock frequency values imported into PowerPlay Early Power Estimator Clock Domain, Logic, and I/O modules are the same as the $f_{MAX}$ values of the design. You can manually edit the clock frequency and the toggle percentage in the PowerPlay Early Power Estimator spreadsheet to suit your system requirements.

# Power Estimation Summary

The main worksheet of the PowerPlay Early Power Estimator spreadsheet summarizes the power and current estimates for the design. It displays the total power, thermal analysis, and power supply current information. The accuracy of the information depends on the information entered. The power consumed can also vary greatly depending on the toggle rates entered. The following sections provide a description of the results of the PowerPlay Early Power Estimator spreadsheet.

## Power

This section shows the power dissipated in the MAX II device. The total thermal power is shown in mWatts and is a sum of the thermal power of all the resources being used in the device. The total thermal power includes the typical power from standby and dynamic power. Figure 17–13 shows the Power section.

**Figure 17–13.** Power Section

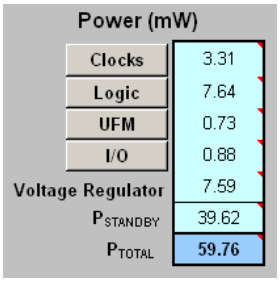



Table 17–7 describes the thermal power parameters in the PowerPlay Early Power Estimator spreadsheet.

**Table 17–7.** Power Information

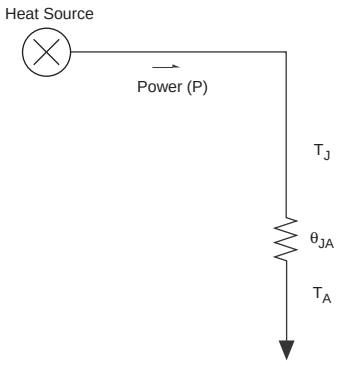
| Column Heading | Description |
|---|---|
| Clock | Represents the dynamic power consumed by clock networks. Click **Clocks** for details. |
| Logic | Represents the dynamic power consumed by LEs and associated routing. Click **Logic** for details. |
| UFM | Represents the dynamic power consumed by the **UFM** block. Click **UFM** for details. |
| I/O | Represents the dynamic power consumed by I/O pins and associated routing. Click **I/O** for details. |
| Voltage Regulator | Represents the dynamic power consumed by the on-chip voltage regulator for a device that supports 2.5-V/3.3-V $V_{CCINT}$. |
| $P_{STANDBY}$ | Represents the standby/static power consumed irrespective of clock frequency. The value includes static power consumed by the I/O banks and the voltage regulator.<br>$P_{STANDBY}$ is dependent on the selected device and the $V_{CCINT}$ supply voltage. |
| $P_{TOTAL}$ | Represents the total power consumed by the CPLD. Refer to "Power Supply Current" on page 17–15 for the current draw from the CPLD supply rails. |

## Thermal Analysis

In the Thermal Analysis part, the PowerPlay Early Power Estimator spreadsheet considers the device's ambient temperature and the airflow to determine the junction temperature ($T_J$) of the device in °C.

The device can be considered a heat source and the junction temperature is the temperature at the device. The thermal resistance of the path is referred to as the junction-to-ambient thermal resistance ($\theta_{JA}$). Figure 17–14 shows the thermal model for the PowerPlay Early Power Estimator spreadsheet.

**Figure 17–14.** Thermal Model for the PowerPlay Early Power Estimator




The PowerPlay Early Power Estimator spreadsheet determines the junction-to-ambient thermal resistance ($\theta_{JA}$) based on the device, package, and airflow selected in the main input parameters.

The PowerPlay Early Power Estimator spreadsheet calculates the total power based on the device properties which provide $\theta_{JA}$ and the ambient and junction temperature using the following equation:

**Equation 17–1.**

$$P = \frac{T_J - T_A}{\theta_{JA}}$$

Figure 17–15 shows the Thermal Analysis section and Table 17–8 describes the thermal analysis parameters in the PowerPlay Early Power Estimator spreadsheet.

**Figure 17–15.** Thermal Analysis Section

| Thermal Analysis | |
|---|---|
| Junction Temp, $T_J$ (°C) | 27.4 |
| $\theta_{JA}$ Junction-Ambient | 39.50 |
| Maximum Allowed $T_A$ (°C) | 82.6 |

**Table 17–8.** Thermal Analysis Information

| Column Heading | Description |
|---|---|
| Junction Temp, $T_J$ (°C) | Represents the estimated device junction temperature. |
| $\theta_{JA}$ Junction-Ambient | Represents the junction-to-ambient thermal resistance through the top of the device (°C/W). |
| Maximum Allowed $T_A$ (°C) | Represents a guideline for the maximum ambient temperature (°C) that the device can be subjected to without violating maximum junction temperature. |

## Power Supply Current

The power supply current provides the estimated current consumption for power supplies. The $I_{CCPOWERUP}$ is only applicable during power up when the configuration flash memory (CFM) block downloads to the SRAM. The $I_{CCINT}$ current is the supply current required from $V_{CCINT}$. The total $I_{CCIO}$ current is the supply current required from $V_{CCIO}$ for all I/O banks. For estimates of $I_{CCIO}$ based on I/O banks, refer to the "I/O Section" on page 17–8 of the PowerPlay Early Power Estimator spreadsheet. Figure 17–16 shows the Power Supply Current section.

**Figure 17–16.** Power Supply Current

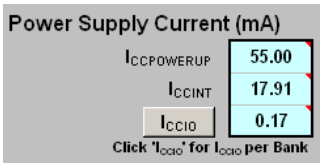



Table 17–9 describes the Power Supply Current parameters of the PowerPlay Early Power Estimator spreadsheet.

**Table 17–9.** Power Supply Current Information

| Column Heading | Description |
|---|---|
| $I_{CCPOWERUP}$ | Represents the maximum current drawn during power-up. |
| $I_{CCINT}$ | Represents the total current drawn from the $I_{CCINT}$ supply. |
| $I_{CCIO}$ | Represents the total current drawn from the $I_{CCIO}$ power rail(s). Refer to the "I/O Section" on page 17–8 for details about the current drawn from each I/O rail. |

# Power Saving Techniques

The following guidelines reduce power consumption for an application:

- Slow the operation in portions of the circuit. $I_{CC}$ is proportional to the frequency of operation. Slowing parts of a circuit lowers the $I_{CC}$ and therefore reduces the power. MAX II devices provide global or array clock source for all registers. Signals that do not require high-speed operation can use a slower array clock that reduces the system power consumption.
- Reduce the number of outputs. Standby and dynamic current are required to support all I/O pins on the device. Reducing the number of I/O pins can reduce current necessary for the device, and thereby reduce the power.

- Reduce the loading and/or external capacitance on the outputs. Excessive loading and capacitance of printed circuit board (PCB) traces and other ICs on the output pins significantly increases the power. Keeping excess load and external capacitance to a minimum on the outputs pins whenever possible will significantly reduce the current necessary for the device.
- Reduce the amount of circuitry in the device. Power depends on the amount of internal logic that switches at any given time. Reducing the amount of logic in a device reduces the current in the device and thus reduces the power.
- Modify the design to reduce power. Identify areas in the design that can be revised to reduce the power requirements. Common solutions include reducing the number of switching nodes and/or required logic, and removing redundant unnecessary signals.
- Modify I/O Locations. Grouping I/O pins from common logic blocks allows the Quartus II software to place the associated logic closer together. The more compact a logic block and I/O, the lower its dynamic power (especially true of low utilization designs with I/O spread around the device).
- Increase the performance requirements in the constraint file. Improving the performance that is beyond the need for operation reduces the power dissipation. The Quartus II software optimizes the design and places logic closer together, uses shorter routing and fewer logic levels, and lowers dynamic power and improves performance.

MAX II devices offer a power-down capability that conserves battery life for portable applications. For more information about the power-down capability in MAX II devices and an application design example, refer to *AN 422: Power Management in Portable Systems Using MAX II CPLDs*.

## Conclusion

This chapter discusses how to evaluate and manage MAX II power by using the MAX II PowerPlay Early Power Estimator spreadsheet. This power estimation tool estimates the power consumption for your design based on typical conditions. The MAX II board-level designer can exploit the power calculator before board design and layout. The MAX II PowerPlay Early Power Estimator spreadsheet is available on the Altera website at www.altera.com.

## Referenced Documents

This chapter references the following documents:

- AN 422: Power Management in Portable Systems Using MAX II CPLDs
- *PowerPlay Power Analysis* chapter in volume 3 of the *Quartus II Handbook*

# Document Revision History

Table 17–10 shows the revision history for this chapter.

**Table 17–10.** Document Revision History

| Date and Revision | Changes Made | Summary of Changes |
|---|---|---|
| October 2008, version 2.1 | ■ Updated New Document Format. | — |
| December 2007, version 2.0 | ■ Updated Figure 17–1, Figure 17–2, and Figure 17–3.<br>■ Updated Table 17–1 with information about power characteristics.<br>■ Updated Table 17–2, Table 17–3, and Table 17–6.<br>■ Added "Referenced Documents" section. | Updated document with MAX IIZ information. |
| December 2006, version 1.5 | ■ Added document revision history. | — |
| July 2006, version 1.4 | ■ Minor content update. | — |
| August 2005, version 1.3 | ■ Updated the entire *MAX II Power Estimation Using the PowerPlay Early Power Estimator* section. | — |
| January 2005, version 1.2 | ■ Previously published as Chapter 18. No changes to content. | — |
| December 2004, version 1.1 | ■ Added Excel Macro, General I/O AC Power, and General I/O DC Power sections.<br>■ Updated figures.<br>■ Updated Table 17-1. | — |