Arria 10 External Memory Interface
Design Guidelines

Quartus II Software v13.1 Arria 10 Edition

Arria 10 design guidelines are preliminary and subject to change
Contents

- Introduction
- Software requirements
- Generating interface IP and example design project
- Generating example design files
- Generating simulation design example files
- Simulation guidelines
- Timing closure guidelines
- Fitter guidelines
- DDR4 / DDR3 interface pin guidelines
Introduction

- Altera EMIF IPs have an optional example design to demonstrate a complete interface solution

- This design can be used by customers for initial interface validation

- Arria 10 example design improvements
  - Faster generation
  - Automatic pin assignments.
    - Script pin_assignments.tcl not created nor needed
Software Requirements

- Quartus II software version 13.1  Arria 10 Edition
Generating External Memory Interface IP and Example Design Project
### New Features

**ARRIA 10 External Memory Interface (EMIF) IP**

All memory protocol generated through a single IP
- Select your protocol in the Arria 10 External Memory Interface Megawizard GUI

Fast generation mechanism
- Faster IP and example design generation

Automatic pin assignments
- I/O standard and pin termination assignments are created during generation
- No `pin_assignments.tcl` file

Synthesis and Simulation file-sets are identical

Ability to create memory configuration preset which can be used in different designs

Select PLL reference clock frequency and FPGA termination settings directly in the Megawizard GUI
The following steps and slides demonstrate how to create the memory interface IP and the example design project.

1. Open Quartus and launch MegaWizard Plug-in Manager from the Tools menu.
2. Select ‘Create a new megafunction variation’ and click Next
3. Select ‘Arria 10 External Memory Interfaces v13.1’ IP under Interfaces->External Memory
4. Select VHDL or Verilog HDL
5. Enter the IP variation name for Memory IP used for the <variation_name>_example_design directory along with the IP files in your workspace
6. Click Next to configure memory IP
7. Select the memory protocol from the drop down list
8. Set the desired interface frequency
9. Configure the Memory IP by selecting appropriate settings available under different tabs on this page.

Note:
- Predefined configurations are available for various memory devices
- Pick the desired memory device preset from the list and click on Apply to populate all the fields with the vendor specified settings
- A custom preset can also be created by clicking on ‘New’ and then entering the configuration data
New Options in the MegaWizard

- Select the PLL reference clock frequency from the drop down menu.
- The allowed PLL ref. clock values are calculated based on the Interface frequency.
- On board oscillator frequency must be one of these values for the memory interface to function properly.

Select the FPGA on-chip termination settings directly in the GUI.
Creating the Example Design

10. After configuring the IP, click on Finish
11. A window will pop-up asking to create an example design
    Ensure the ‘Generate Example Design’ option is selected and click on Generate
After IP generation is complete, `<variation_name>_example_design` directory will be created in your project directory.

In this example, the variation name is `ddr3` and the script files needed to create an example design are available in `ddr3_example_design`.

Contains necessary TCL files required to create a synthesizable example design.
Two TCL Scripts Created

- `<variation_name>_example_design` contains two TCL scripts:
  
  I.  `make_qii_design.tcl`

  The `make_qii_design.tcl` generates a synthesizable design example along with a Quartus project, ready for compilation.

  II.  `make_sim_design.tcl`

  The `make_sim_design.tcl` generates a simulation design example along with tool-specific scripts to compile and elaborate the necessary files.
To generate synthesizable design example, run the `make_qii_design.tcl` script in Nios II command shell or from a command line:

12. Open Nios II command shell and browse to the `<variation_name>_example_design` directory
Or change directory to `<variation_name>_example_design` directory

13. Run the `make_qii_design.tcl` script by executing the following command:
```
quartus_sh -t make_qii_design.tcl
```

Optionally can run the `make_qii_design.tcl` script for a specific device
```
quartus_sh -t make_qii_design.tcl 10AX115R3F40I2SGES
```
This script runs for a few seconds and produces a qii directory containing a project called ed_synth.qpf

Open and compile this project with the Quartus II software v13.1 Arria 10 Edition

- QSYS file is also generated
- Open this file in QSYS to add remove or modify IPs in the example design
Generating Simulation Files for Example Design Project

Overview
Details in simulation guidelines section
Generating the Simulation Design Example Files

- To generate a simulation design example, run the following script in Nios II command shell or from the command line.
  Options of VERILOG or VHDL

```
quartus_sh -t make_sim_design.tcl VERILOG
```

- The simulation design example is made of a driver connected to the generated IP (device under test or DUT) and to the memory model.
  - Driver generates random traffic and internally checks the legality of the outgoing data.
Simulation Example Design Script Output

- Script creates a sim directory containing one subdirectory for each supported simulation tool
  - Each subdirectory contains the specific scripts to run simulation with the corresponding tool

```
Command: ls -F sim
aldec/  cadence/  ed_sim.qsys  ed_sim.v  mentor/  submodules/  synopsys/
```

```
Command: ls -F sim/mentor
msim_setup.tcl
```

© 2013 Altera Corporation—Public
Arria 10 Simulation Guidelines are preliminary and subject to change
Users will be able to choose between two simulation models

- **Skip Calibration**
  - Fastest simulation
  - Loads the settings calculated from memory configuration and enters user mode

- **Full Calibration (not supported in Quartus II 13.1)**
  - Performs all stages of memory calibration: calibration phases, delay sweeps, and centering of all data bits

<table>
<thead>
<tr>
<th>Skip Calibration mode</th>
<th>Full Calibration mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-level simulation focusing on user logic</td>
<td>Memory interface simulation focusing on calibration</td>
</tr>
<tr>
<td>Details of calibration are not captured</td>
<td>Details of calibration are captured (i.e. stages)</td>
</tr>
<tr>
<td>Enables users to store and retrieve data</td>
<td>Includes leveling, per-bit deskew, etc.</td>
</tr>
<tr>
<td>Efficiency accurate</td>
<td></td>
</tr>
<tr>
<td>No board skews are taken into account</td>
<td></td>
</tr>
</tbody>
</table>
Simulation: Supported & Not Supported

<table>
<thead>
<tr>
<th>Supported</th>
<th>Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Verification</td>
<td>Timing Verification</td>
</tr>
<tr>
<td>Skip Calibration (default)</td>
<td>Nativelink</td>
</tr>
<tr>
<td></td>
<td>Memory Vendor Models</td>
</tr>
<tr>
<td></td>
<td>Full Calibration*</td>
</tr>
<tr>
<td></td>
<td>Post-Fit Simulation*</td>
</tr>
<tr>
<td></td>
<td>Multi-Rank*</td>
</tr>
<tr>
<td></td>
<td>Multiple-CS memory interface*</td>
</tr>
<tr>
<td></td>
<td>Memory frequency &lt; 400Mhz*</td>
</tr>
<tr>
<td></td>
<td>RDIMM &amp; LRDIMM configurations*</td>
</tr>
</tbody>
</table>

*Available in a future Quartus II version

Note: Validating the timing of your design requires using Altera’s TimeQuest Timing Analyzer
Simulation

To simulate your design you will need the following components

<table>
<thead>
<tr>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera Supported Simulator</td>
</tr>
<tr>
<td>Design using Altera External Memory Interfaces IP</td>
</tr>
<tr>
<td>An Example Driver (Altera or User)</td>
</tr>
<tr>
<td>Testbench (Altera or User)</td>
</tr>
<tr>
<td>Altera’s Memory Simulation Model (We do not support simulation with memory vendor models)</td>
</tr>
</tbody>
</table>
Information About Simulation Filesets & Directory

- Core simulation filesets are identical to core synthesis filesets
  - Addresses simulation v synthesis fileset concerns of the past
  - Ensure users are simulating the blocks they are synthesizing
    
  ```
  <dut>/*
  <dut>_sim/altera_emif_arch_nf/*
  ```

- Any changes made in the synthesis directory should also be made in the simulation directory to reflect similar IP behavior

- Fewer files to compile for simulation compared to the External Memory Interfaces IP of the past (UniPHY, ALTMEMPHY)
  - 64 files for the example design
    - 7 are unique for each interface
    - Users can modify the BIST in the example design if needed
Important Assumptions Made by Simulation

- **Altera library simulation atoms assume the following:**
  - RTL simulation assumes an ideal layout including:
    - Interfaces are unaware of each other
      - Interface assumes it is the only interface in the column
      - Interface believes it has its own IOAUX and Hard Nios
    - Interface is at the bottom of the column, nearest to the physical IOAUX block location
      - Fitter may actually place an interface at the top of the column if left unconstrained but there are no drawbacks between an interface at the top of the column and an interface at the bottom of the column
  - One PLL per interface
    - At Post-Fit, it is possible for interfaces to share the same bank PLL
    - PLL reset only occur during power-up
    - Issue a recalibration request per-EMIF interface in place of a PLL reset

*EMIF = External Memory Interfaces*
There may be a discrepancy between the simulated latency versus the Post-Fit latency

Do not rely on the simulated interface latency
Introduction of AFI Clock Cycle penalty:
- For wide, multi-bank interfaces and/or ultra-low latency interfaces
- Fitter can detect this penalty and will issue a warning accordingly
- Only an issue when the requested write latency is less than the latency accrued by the farthest away bank
RTL Simulation v Post-Fit Implementation

- **RTL Simulation:** NIOS initialization and calibration code executes in parallel for all interfaces
  - Interfaces might assert ‘cal_done’ (calibration done) simultaneously in simulation
  - Do not rely on this behavior shown in simulation

- **Post-Fit Implementation:** NIOS initialization and calibration code executes sequentially
  - Order of calibration is determined by fitter operations
  - Calibration is complete when all interfaces in a column assert 'cal_done''
  - You must sample all cal_done signals in a column to determine when calibration is complete
Step 1: Generate the Design

In addition to generating the IP component with the settings you defined, you can also generate an example design. Generating an example design can significantly reduce processing times.
Simulating the example design

**Generation Output**

<table>
<thead>
<tr>
<th>File/Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dut/</td>
<td>Actual IP for your project</td>
</tr>
<tr>
<td>dut_example_design/</td>
<td>Example Design Subfolder</td>
</tr>
<tr>
<td>dut_sim/</td>
<td>Simulation Fileset (no driver)</td>
</tr>
<tr>
<td>dut.cmp</td>
<td>Component Declaration File (text file containing port definitions that can be used in VHDL Design Files)</td>
</tr>
<tr>
<td>dut.qip</td>
<td>Quartus IP File (contains paths for all files needed for the IP core)</td>
</tr>
<tr>
<td>dut.spd</td>
<td>Simulation Package Descriptor File (lists the required simulation files for the IP core or Qsys system)</td>
</tr>
<tr>
<td>dut.sip</td>
<td>Simulation IP File (contains information assignments that specify IP simulation source files)</td>
</tr>
<tr>
<td>dut.ppf</td>
<td>Pin Planner File (XML file that stores the port and node assignments for use with the Pin Planner)</td>
</tr>
<tr>
<td>dut.v</td>
<td>Variation File of the IP core (contains the IP settings used to generate the IP core)</td>
</tr>
</tbody>
</table>

**Core files in each subdirectory are identical**
- `dut_example_design/*`: Contains driver and memory model
- `dut_sim/*` and `dut/*`: Identical
**Example Design Output Files**

- **In dut_example_design**

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ed_sim.qsys</td>
<td>Qsys file capturing the simulation example design</td>
</tr>
<tr>
<td>ed_synth.qsys</td>
<td>Qsys file capturing the example design for synthesis</td>
</tr>
<tr>
<td>make_qii_design.tcl</td>
<td>Script to generate the example design project for synthesis</td>
</tr>
<tr>
<td>make_sim_design.tcl</td>
<td>Script to generate the example design for simulation</td>
</tr>
<tr>
<td>params.tcl</td>
<td>Support file for the generation scripts</td>
</tr>
<tr>
<td>params.txt</td>
<td>XML file created that holds user chosen IP settings</td>
</tr>
<tr>
<td>readme.txt</td>
<td>Instructions for user</td>
</tr>
</tbody>
</table>
Generating the Example Design Simulation Files

- Run "quartus_sh -t make_sim_design.tcl VERILOG" to generate the simulation files for the example design in verilog.
- Run "quartus_sh -t make_sim_design.tcl VHDL" to generate the simulation files for the example design in VHDL.
You can create your own .do file in order to view signals in Modelsim’s waveform viewer

Example: run.do

```tcl
if {file exists msim_setup.tcl} {
    source msim_setup.tcl
    dev_com
    # the "clab_debug" macro avoids optimizations which preserves signals so that they may be added to the wave viewer
    clab_debug
    add wave "ed_sim/*"
    run -all
} else {
    error "The msim_setup.tcl script does not exist. Please generate the example design RTL and simulation scripts. See ../../README.txt for help."
}
```

Execute ‘do run.do’ in the Modelsim console to run the simulation with signal waveforms

To only see a successful simulation result shown in the console messages

- Execute ‘source msim_setup.tcl’ in the Modelsim console
- Execute ‘ld_debug’ after msim_setup.tcl is loaded
- Execute ‘run -all’ after ld_debug finishes
To store the entire log of the simulation data and results

- Edit *msim_setup.tcl* & add `-l ed_sim.log` to the *vsim* line

```tcl
# Elaborate top level design
alias elab {
    echo "\[exec\] elab"
    eval vsim -t ps $ELAB_OPTIONS -L work -L work_lib -L dut_ctrl_amm_avalon_slave_0_translator -L avl_tg_;
}

# Elaborate the top level design with *novopt* option
alias elab_debug {
    echo "\[exec\] elab debug"
}

# Compile all the design files and elaborate the top level design
alias ld "
    dev_com
    com
    elab
"
```
Simulating the example design

**Supported simulators:**

<table>
<thead>
<tr>
<th>Supported</th>
<th>Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentor Graphics Modelsim</td>
<td>Aldec Riviera-PRO*</td>
</tr>
<tr>
<td>Synopsys VCS and VCS-MX</td>
<td>Cadence NCSIM*</td>
</tr>
</tbody>
</table>

*Available in a future Quartus II version*
Arria 10 EMIF Timing Closure Guidelines

Arria 10 timing closure guidelines are preliminary and subject to change
Arria 10 EMIF Timing paths

- **FPGA Core**
  - C2C
  - C2P
  - P2C
  - P2P
- **Periphery**
- **IO including read, write, write levelling, etc**
- **Memory**

© 2013 Altera Corporation—Public
Timing Closure Guidelines

- **Timing closure in any core transfers include**
  - From last set of registers in core to first set of registers to periphery (C2P)
  - From last set of registers in periphery to first of registers in core (P2C)
  - Note that C2P/P2C paths are cut and not analyzed in the current Quartus II release
  - Core Timing analysis will not include user logic timing nor user logic timing to/from EMIF block

- **Arria 10 timing analysis will not show any periphery-to-periphery timing in TimeQuest**

- **Timing closure in any of the IO transfers**
  - Not dependent on Quartus II compile
  - Dependent on customer memory, FPGA speed grade parameters and channel effects

- **For accurate timing analysis, simulate correct board parameters and channel effects**
  - Board skews must be simulated using board tool (not estimated or calculated via trace length)
  - Channel effects (ISI and crosstalk) can only be determined by a board simulator
  - Include simulated Board skew and Channel effects in Megawizard GUI during IP generation
  - Refer to Board guidelines for more details on board skew and channel effects

- **ReportDDR” will run automatically as part of signoff timing analysis**
  - User has to check “ReportDDR” as part of signoff timing analysis to make sure EMIF has closed timing
Users can see IO margins without compiling EMIF design
- Early IO timing will look like a spreadsheet type analysis shown in a TimeQuest panel
- Provides breakdown in margin loss between receiver/transmitter/channel

Flow
- Generate EMIF IP with configuration of interest including memory and board parameters
- Create Quartus II project files (QPF, QSF) with selected Arria 10 device part
- Run TimeQuest with `<name>_report_io_timing.tcl` that get’s generated as part of the IP
  - Details in the next slide

---

**Ideal Window**

**Channel Effects**

**Memory (Receiver) Effects**

**FPGA (Transmitter) Effects**

**Final Margin**

---

**Table:**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Margin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal Timing Window</td>
<td>0.469</td>
</tr>
<tr>
<td>51</td>
<td>0.063</td>
</tr>
<tr>
<td>55</td>
<td>0.023</td>
</tr>
<tr>
<td>slew rate derating</td>
<td>0.040</td>
</tr>
<tr>
<td>tDQSQ effect</td>
<td>0.075</td>
</tr>
<tr>
<td>tQH effect</td>
<td>0.112</td>
</tr>
<tr>
<td>memory calibration</td>
<td>-0.075</td>
</tr>
<tr>
<td>Jitter Effects</td>
<td>0.072</td>
</tr>
<tr>
<td>Duty Cycle Distortion</td>
<td>0.031</td>
</tr>
<tr>
<td>setup/hold time</td>
<td>0.016</td>
</tr>
<tr>
<td>EOL</td>
<td>0.025</td>
</tr>
<tr>
<td>Calibration uncertainty</td>
<td>0.027</td>
</tr>
<tr>
<td>skew effect</td>
<td>0.009</td>
</tr>
<tr>
<td>Final Read Margin</td>
<td>0.051</td>
</tr>
</tbody>
</table>
Running TimeQuest for Early IO Estimates

1. Start TimeQuest
2. Open Project
3. Pick “Script → Run TCL script”
4. Pick `<name>_report_io_timing.tcl` file
5. TimeQuest prints out summary and creates a “ReportDDR” panel
   - Same level of detail for IOs
   - Produces a warning mentioning that core timing is not included
   - Similar type of analysis available for all IO transfers
     (Read capture, DQS gating, A/C and Write Leveling)

**Note:** To generate early IO timing reports, run `report_IO_timing.tcl` before running any Quartus II compilation
Early IO Estimates – other Execution Methods

- Instead of “Script → Run TCL script” just type
  - “source <name>_report_io_timing.tcl” in the TCL console
  - OR

- At the command prompt type
  - Quartus_stata-t <core_name>_report_io_timing.tcl <project_name>

Note: Positive margins in Early IO timing estimate does not guarantee signoff timing analysis
Arria 10 Fitter Guidelines

Arria 10 fitter guidelines are preliminary and subject to change
Fitter Behaviors

- Multi-bank Interface: Multiple, contiguous banks that make up one interface

- Introduction of AFI Clock Cycle penalty:
  - For wide, multi-bank interfaces and/or ultra-low latency interfaces
  - Fitter can detect this penalty and will issue a warning accordingly
  - Only an issue when the requested write latency is less than the latency accrued by the farthest away bank in a multi-bank interface
Clocking

- Since multi-bank interfaces will use multiple PLLs and in turn multiple PHY clock trees, a reference clock tree will be used to route a common reference clock signal to all PLLs
  - Not all pins can drive the PLL reference clock tree
  - Quartus II software restricts PLL reference clock frequencies depending on the memory frequency
    - Use the Arria 10 External Memory Interfaces IP Megawizard GUI to determine the valid PLL reference clock frequencies

- **Fitter merges PLLs when a bank is shared by different interfaces**

- **Fitter duplicates PLL for multi-bank interfaces**
Example of the reference clock tree driving multiple PLLs which are, in turn, driving multiple PHY clock trees

- Jitter is lowered with this balanced structure
Sharing Resources

- In Arria 10, the following resources can be shared and in some cases are forced to be shared:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O bank</td>
<td>Ability to fit more interfaces in a single column</td>
</tr>
<tr>
<td>Hard Nios II</td>
<td>Cannot rely on one cal_done signal as representative of all interfaces passing the calibration stage</td>
</tr>
<tr>
<td>Core Clock Network</td>
<td>Shared PLL reference network. Users should place interfaces in consecutive banks</td>
</tr>
<tr>
<td>PLL reference clock pins*</td>
<td>Shared PLL reference clock and network trees</td>
</tr>
<tr>
<td>OCT block and RZQ pin*</td>
<td>None</td>
</tr>
<tr>
<td>Address/Command pins*</td>
<td>Shared for Ping Pong PHY</td>
</tr>
</tbody>
</table>

*More in Pin Guidelines

- Certain resources are forced to be shared
  - IOAUX & Hard Nios II CPU for all interfaces in a column
  - A bank shared by two interfaces

- PLL/DLL do not need to be shared as each bank has one
Sharing an I/O Bank

- Fitter can place interfaces in a shared bank if the interfaces share the same:
  - Protocol
  - Rate
  - Phase
  - Frequency

- Users can fit even more interfaces in a column

- Interfaces cannot share the same controller nor sequencer

- Fitter will not allow users to have a lane shared by two interfaces
  - One DQS-in tree can only talk to one controller

- Unused pins can be used by the customer as a GPIO
  - Must be the same voltage standard
Example: 2 x16 interfaces sharing a bank

Bank N+1
Controller 1
Addr/Cmd 1
Addr/Cmd 1
Addr/Cmd 1

Bank N
Controller 2
Data 2
Data 2

Bank N-1
Controller 2
Addr/Cmd 2
Addr/Cmd 2
Addr/Cmd 2

Unused
(Free for GPIO, but not LVDS)
Fixed Address / Command Pin out
Data path
Unused
(Free for GPIO, but not LVDS)
Data path
Data path
Fixed Address / Command Pin out
Sharing Hard Nios / IOAUX

- Interfaces placed within the same column by the fitter will share the same IOAUX and Hard Nios II
- The Hard Nios II calibrates each interface sequentially
  - You must sample all cal_done signals in a column to determine when calibration is complete
- RTL simulation behaves as if every interface has its own Hard Nios II
  - More on this in the Simulation Design Guidelines
The Arria 10 External Memory Interfaces IP will contain one Hard Nios II and IOAUX per interface but fitter will merge them all into a single instance.

You must use the same IOAUX clock and reset for all interfaces in the same column or Fitter will generate an error.
Sharing Core Clock Networks

- Fitter can use one core clock domain to synchronously access all interfaces in a column
- Users can share core clock networks by the master & slave setting in the IP generation GUI
  - Connect core_clks_master_out from the master to all slave’s core_clks_slave_in
  - Must use same column, PLL reference clock, rate, and frequency
  - Interfaces in different columns cannot use this feature
  - Place interfaces in consecutive banks as the PLL reference clock are forced to be shared when choosing to share core clock networks
**Fitter Relationship to Pin Assignments**

**Pin assignments**
- Fitter can reallocate banks based on user pin assignments
- Fitter can rotate pins within a lane based on user pin assignments but cannot move pins across lanes away from their DQS group
- Users can constrain a DQS pin to a lane, and the Fitter will place all DQ signals in their respective DQS group in the same lane
Arria10 Interface Pin Guidelines

Arria 10 Pin Guidelines are preliminary and subject to change
Overview

- **Pin Guidelines**
  - Guidelines
  - Rules for constraining pins
  - Determine IO bank requirements for DDR3, DDR4
  - Interface placement
  - Find pin names for A/C and data pins
  - Example for constraining DDR3 x8 and x72
  - Alternate methods for constraining interface pin assignments

- **Pin guidelines for sharing multiple interfaces**
  - Constraints for sharing multiple interfaces
  - Step by step guidelines
Pin Guidelines

1. Determine number of banks based on interface width

2. Pick CK0 pin based on desired interface location in the FPGA

3. Constrain CK0 pin to selected pin name or A/C bank

4. Constrain one DQS pin for each DQS group either to pin name or A/C bank

5. Constrain PLL reference clock pin and RZQ pin to pin names
Rules for constraining pins

- **A/C**
  - All A/C pins should be in a single BANK
  - A/C and data pins cannot share a lane (12 IOs)
  - But unused A/C pins in a lane can be used by GPIOs
  - A/C pins must follow predefined locations within a BANK
  - A/C and data pins can share a bank

- **DQ pins**
  - DQ signals from two different DQS groups cannot be constrained to same IO_12_LANE

- **DQS pins**
  - Related DQ pins must be in the same IO_12_LANE(s)
  - A read data group must be assigned based on DQS in grouping in pin table
Rules for constraining pins (contd)

- **PLL Reference Clock pins**
  - For a given interface speed, there is a restriction on values of possible PLL ref clock frequencies
  - **Must use Arria 10 Interface v13.1 MegaWizard to determine possible PLL reference clock frequencies for onboard crystal oscillator**
    - Altera recommends using the default PLL reference clock frequency from MegaWizard
  - Crystal clock frequency is generally lowest clock frequency of memory interface divided by integer N
    - Where N= 1,2,3,4,5

- **Find detailed step by step guidelines for constraining pins in next few slides**
PDF text content:

**DDR3: Determine Number of Banks Required**

- Find number of banks required based on whether there is any IO bank sharing or not
  - Pin count for DDR3 8/16/32-bit is based on 1CS
  - When using multiple CS for DDR3 8/16/32-bit,
    - Add 5 more pins for 2CS to total IO count
    - Add 15 more pins for 4CS to total IO count
  - Calculate the number of banks required with multiple CS pins
    - No. of IO Banks = (Total No. of IOs for 1CS + Additional IOs for multiple CS) / 48

<table>
<thead>
<tr>
<th>Interface width and memory configuration</th>
<th>Number of IOs</th>
<th>IO Banks (non-sharing)</th>
<th>IO Banks sharing with other Interfaces (1/4 Granularity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit w/o ECC 1CS</td>
<td>43</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16-bit w/o ECC 1CS</td>
<td>55</td>
<td>2</td>
<td>1.25</td>
</tr>
<tr>
<td>16-bit with ECC 1CS</td>
<td>67</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>32-bit w/o ECC 1CS</td>
<td>79</td>
<td>2</td>
<td>1.75</td>
</tr>
<tr>
<td>32-bit with ECC 1CS</td>
<td>91</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>72-bit UDIMM 1-Rank</td>
<td>139</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>72-bit UDIMM 2-Rank</td>
<td>144</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>72-bit UDIMM 4-Rank</td>
<td>154</td>
<td>4</td>
<td>3.25</td>
</tr>
<tr>
<td>72-bit SO-DIMM 1-Rank</td>
<td>127</td>
<td>3</td>
<td>2.75</td>
</tr>
<tr>
<td>72-bit SO-DIMM 2-Rank</td>
<td>132</td>
<td>3</td>
<td>2.75</td>
</tr>
<tr>
<td>72-bit SO-DIMM 4-Rank</td>
<td>142</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Find number of banks required based on whether there is any IO bank sharing or not

- Pin count for DDR3 8/16/32-bit is based on 1CS
- When using multiple CS for DDR3 8/16/32-bit,
  - Add 5 more pins for 2CS to total IO count
  - Add 15 more pins for 4CS to total IO count
- Calculate the number of banks required with multiple CS pins
  - No. of IO Banks = (Total No. of IOs for 1CS + Additional IOs for multiple CS) / 48

<table>
<thead>
<tr>
<th>Interface width and memory configuration</th>
<th>Number of IOs</th>
<th>IO Banks (non-sharing)</th>
<th>IO Banks sharing with other Interfaces (With 1/4 Granularity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit w/o ECC 1CS</td>
<td>49</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16-bit w/o ECC 1CS</td>
<td>61</td>
<td>2</td>
<td>1.25</td>
</tr>
<tr>
<td>16-bit with ECC 1CS</td>
<td>73</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>32-bit w/o ECC 1CS</td>
<td>85</td>
<td>2</td>
<td>1.75</td>
</tr>
<tr>
<td>32-bit with ECC 1CS</td>
<td>97</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>72-bit UDIMM 1-Rank</td>
<td>142</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>72-bit UDIMM 2-Rank</td>
<td>147</td>
<td>4</td>
<td>3.25</td>
</tr>
<tr>
<td>72-bit UDIMM 4-Rank</td>
<td>157</td>
<td>4</td>
<td>3.25</td>
</tr>
<tr>
<td>72-bit UDIMM 1-Rank</td>
<td>145</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>72-bit UDIMM 2-Rank</td>
<td>150</td>
<td>4</td>
<td>3.25</td>
</tr>
<tr>
<td>72-bit UDIMM 4-Rank</td>
<td>160</td>
<td>4</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Select consecutive banks out of 8 banks in a column and select middle bank for Address/command (A/C) pins (Must)
- In case of even number of banks, pick any one of the middle two

A/C pins can take 3 or 4 IO lanes depending on memory topology and protocol

When A/C requires only 3 IO lanes, only bottom 3 lanes (A/C 0, 1, 2) must be used
DDR3: Find pin number for CK0 pin

Find pin number for CK0 pin based on the A/C lanes selected

- Examples:
  - Pin 8 corresponds to CK0 pin
  - Pin 24 corresponds to PLL_clockin[0]
  - Pin 26 corresponds to RZQ

- The pin number for A/C pins is also generated in “project”_readme.txt in “Project”/submodules/ folder

- Altera recommends using the “project”_readme.txt file to find the pin numbers for A/C pins

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Component, UDIMM, SO-DIMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>CK3#</td>
</tr>
<tr>
<td>44</td>
<td>CK3</td>
</tr>
<tr>
<td>43</td>
<td>CK2#</td>
</tr>
<tr>
<td>42</td>
<td>CK2</td>
</tr>
<tr>
<td>41</td>
<td>CKE[3]</td>
</tr>
<tr>
<td>40</td>
<td>CKE[2]</td>
</tr>
<tr>
<td>39</td>
<td>ODT3</td>
</tr>
<tr>
<td>38</td>
<td>ODT2</td>
</tr>
<tr>
<td>37</td>
<td>CS[3]</td>
</tr>
<tr>
<td>36</td>
<td>CS[2]</td>
</tr>
<tr>
<td>35</td>
<td>BA[2]</td>
</tr>
<tr>
<td>34</td>
<td>BA[1]</td>
</tr>
<tr>
<td>33</td>
<td>BA[0]</td>
</tr>
<tr>
<td>32</td>
<td>CAS#</td>
</tr>
<tr>
<td>31</td>
<td>RAS#</td>
</tr>
<tr>
<td>30</td>
<td>A[15]</td>
</tr>
<tr>
<td>29</td>
<td>A[14]</td>
</tr>
<tr>
<td>28</td>
<td>A[13]</td>
</tr>
<tr>
<td>27</td>
<td>A[12]</td>
</tr>
<tr>
<td>26</td>
<td>RZQ</td>
</tr>
<tr>
<td>25</td>
<td>PLL_clockin[1]</td>
</tr>
<tr>
<td>24</td>
<td>PLL_clockin[0]</td>
</tr>
<tr>
<td>22</td>
<td>A[10]</td>
</tr>
<tr>
<td>21</td>
<td>A[9]</td>
</tr>
<tr>
<td>20</td>
<td>A[8]</td>
</tr>
<tr>
<td>19</td>
<td>A[7]</td>
</tr>
<tr>
<td>17</td>
<td>A[5]</td>
</tr>
<tr>
<td>15</td>
<td>A[3]</td>
</tr>
<tr>
<td>14</td>
<td>A[2]</td>
</tr>
<tr>
<td>13</td>
<td>A[1]</td>
</tr>
<tr>
<td>12</td>
<td>A[0]</td>
</tr>
<tr>
<td>11</td>
<td>CK1#</td>
</tr>
<tr>
<td>10</td>
<td>CK1</td>
</tr>
<tr>
<td>9</td>
<td>CK0#</td>
</tr>
<tr>
<td>8</td>
<td>CK0</td>
</tr>
<tr>
<td>7</td>
<td>CKE[1]</td>
</tr>
<tr>
<td>6</td>
<td>CKE[0]</td>
</tr>
<tr>
<td>5</td>
<td>ODT1</td>
</tr>
<tr>
<td>4</td>
<td>ODT0</td>
</tr>
<tr>
<td>3</td>
<td>CS[1]</td>
</tr>
<tr>
<td>2</td>
<td>CS[0]</td>
</tr>
<tr>
<td>1</td>
<td>RESET#</td>
</tr>
<tr>
<td>0</td>
<td>WE#</td>
</tr>
</tbody>
</table>
Find pin number for CK0 pin based on the A/C lanes selected

- Examples:
  - Pin 8 corresponds to CK0 pin
  - Pin 24 corresponds to PLL_clockin[0]
  - Pin 26 corresponds to RZQ

- The Pin number for A/C pins is also generated in “project”_readme.txt in “Project”/submodules/ folder

- Altera recommends using the “project”_readme.txt file to find the pin numbers for A/C pins

<table>
<thead>
<tr>
<th>Pin No.</th>
<th>Component</th>
<th>UDIMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>CK1#</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>CK1</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>CK3#</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>CK3</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>CK2#</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>CK2#</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>CKE[3]</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>CKE[2]</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>ODT3</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>ODT2</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>CS[3]</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Alert_n</td>
<td>CS[2]</td>
</tr>
<tr>
<td>35</td>
<td>BG[0]</td>
<td>BG[0]</td>
</tr>
<tr>
<td>34</td>
<td>BA[1]</td>
<td>BA[1]</td>
</tr>
<tr>
<td>33</td>
<td>BA[0]</td>
<td>BA[0]</td>
</tr>
<tr>
<td>26</td>
<td>RZQ</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>PLL_clockin[0]</td>
<td>PLL_clockin[0]</td>
</tr>
<tr>
<td>12</td>
<td>A[0]</td>
<td>A[0]</td>
</tr>
<tr>
<td>11</td>
<td>PAR_in</td>
<td>PAR_in</td>
</tr>
<tr>
<td>10</td>
<td>C2</td>
<td>CS[1]</td>
</tr>
<tr>
<td>9</td>
<td>CK0#</td>
<td>CK0#</td>
</tr>
<tr>
<td>8</td>
<td>CK0</td>
<td>CK0</td>
</tr>
<tr>
<td>7</td>
<td>C1</td>
<td>CKE[1]</td>
</tr>
<tr>
<td>6</td>
<td>CKE[0]</td>
<td>CKE[0]</td>
</tr>
<tr>
<td>5</td>
<td>C0</td>
<td>ODT1</td>
</tr>
<tr>
<td>4</td>
<td>ODT0</td>
<td>ODT0</td>
</tr>
<tr>
<td>3</td>
<td>ACT_n</td>
<td>ACT_n</td>
</tr>
<tr>
<td>2</td>
<td>CS[0]</td>
<td>CS[0]</td>
</tr>
<tr>
<td>1</td>
<td>RESET#</td>
<td>RESET#</td>
</tr>
<tr>
<td>0</td>
<td>BG[1]</td>
<td>BG[1]</td>
</tr>
</tbody>
</table>
Find Pin Name for CK0 and Constrain the Pin

- Based on A/C bank selected, identify column index and bank index for the A/C pins
  - Column Index ranges from 0-1 and Bank Index from 0-7

- Find pin name for A/C pin from pin table and constrain pin to pin name in QSF
  - Example: Pin name for CK0 in Column0 Bank 6 is L23
  - `set_location_assignment PIN_L23 –to CK0`

OR

- Find Bank name for A/C pin and constrain pin to selected IO48 bank in QSF
  - Example: Selected IO bank for CK0 in Column0 Bank 6 is 2K
  - `set_location_assignment IOBANK_2K –to CK0`
  - Effectively locks all A/C signals

- Use Arria10 Pin table to find Pin index, Column index and Bank Index

<table>
<thead>
<tr>
<th>I/O Bank</th>
<th>2K</th>
<th>2K</th>
<th>2K</th>
<th>2K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic I/O Input &amp; Output Power</td>
<td>Primary Function</td>
<td>Cell Name</td>
<td>DQS for X8</td>
<td>DQS for X8</td>
</tr>
<tr>
<td>2K</td>
<td>P10x0Y6</td>
<td>io_48_lvds_tied</td>
<td>DQ0_2K_1</td>
<td>DQ0_2K_11</td>
</tr>
<tr>
<td>2K</td>
<td>P9x0Y6</td>
<td>io_48_lvds_tied</td>
<td>DQ0_2K_2</td>
<td>DQ0_2K_12</td>
</tr>
<tr>
<td>2K</td>
<td>P8x0Y6</td>
<td>io_48_lvds_tied</td>
<td>DQ0_2K_3</td>
<td>DQ0_2K_13</td>
</tr>
<tr>
<td>2K</td>
<td>P7x0Y6</td>
<td>io_48_lvds_tied</td>
<td>DQ0_2K_4</td>
<td>DQ0_2K_14</td>
</tr>
<tr>
<td>2K</td>
<td>P6x0Y6</td>
<td>io_48_lvds_tied</td>
<td>DQ0_2K_5</td>
<td>DQ0_2K_15</td>
</tr>
<tr>
<td>2K</td>
<td>P5x0Y6</td>
<td>io_48_lvds_tied</td>
<td>DQ0_2K_6</td>
<td>DQ0_2K_16</td>
</tr>
</tbody>
</table>

- P = Pin index (0-47)
- X = Column index
- Y = Bank index
Constrain DQS and PLL Reference Clock Pins

- Constrain PLL reference clock pin and RZQ pin to the pin names

- Constrain one DQS pin for each DQS group either to the pin names or selected IO48 banks

- Follow same method as CK0 for finding pin names for DQS and PLL reference clock pins

- Constraining CK0 pin, one DQS pin per group and PLL reference clock effectively locks the entire interface
  - Good compromise between full-automatic and manual placement
  - Requires minimal effort
  - Altera recommends this method for constraining pin assignments
Example for constraining DDR3 x8

- Requires 1 Bank
  - 3 lanes for A/C pins and 1 lane for DQ and DQS pins
- Picked IO lanes 0,1,2 for A/C Pins
- Constrain pin CK0 to pin8 of Bank0 (P8X0Y0)
  - set_location_assignment PIN_AG16 –to CK0
- Constrain pin DQS0 to Bank0 (2A)
  - set_location_assignment IOBANK_2A –to DQS0
- Constrain PLL_refclk to pin24 to Bank0 (P24X0Y0)
  - set_location_assignment PIN_AM15 –to PLL_clockin
- Constrain rzqpin to pin24 to Bank0 (P26X0Y0)
  - set_location_assignment PIN_AK18 –to rzqpin
Example for constraining DDR3 x72

- **DDR3 x72 w/ Hard Controller**
  - Requires 3 banks
    - 3 lanes for A/C pins
    - 9 lanes for data

- **Constraining DDR3 x72**
  - Constrain pin CK0 to pin8 of Bank1 (P8X0Y1)
  - Constrain PLL refclk to pin24 of Bank1 (P24X0Y1)
  - Constrain rzqpin to pin26 of Bank1 (P24X0Y1)
  - Constrain DQS groups
    - DQS0 to Bank2 (2G)
    - DQS1 to Bank2 (2G)
    - DQS2 to Bank2 (2G)
    - DQS3 to Bank2 (2G)
    - DQS4 to Bank1 (2F)
    - DQS5 to Bank0 (2A)
    - DQS6 to Bank0 (2A)
    - DQS7 to Bank0 (2A)
    - DQS8 to Bank0 (2A)
Alternate Methods: Constraining Interface Pin assignments

1. Let the fitter assign all Interface signals (A/C, DQS, DQ pins) automatically
   a) Run the design through the fitter without any constraints
   b) Save the post-fit netlist, or back-annotate the pin assignments
      - Requires least effort but longer compilation time
      - This method works well for small designs (one interface per column)
      - Must not use this method for large designs with multiple IPs (Interfaces, HSSI, GPIOs, LVDS etc.)

2. Manually constrain all Interface signals (A/C, DQS, DQ pins) to pin locations
   a) Plan the Interface placement in a column (i.e., which IO48 banks to use)
   b) Use pin table to find legal position for each Interface pin
   c) Use QSF assignments to lock down the pins
      - Fast periphery placement
      - Can be a lengthy/tedious process (especially with multiple IPs)
Overview

- **Pin Guidelines**
  - Step by Step Guidelines
  - Rules for constraining A/C, DQ, DQS, and CLK Pins
  - Determine IO Bank Requirements for DDR3, DDR4
  - Interface placement
  - Find Pin Names for A/C and data pins
  - Constraining Interface Pin assignments

- **Pin Guidelines for sharing Multiple Interfaces**
  - Constraints for sharing multiple interfaces
  - Step by step guidelines
Constraints for Sharing Multiple Interfaces

- While sharing bank across multiple Interfaces the following criteria should be followed
  - Must use identical clocks (rate, frequency, PLL ref clock)
  - Same protocol
  - Same voltage settings (VCCIO, VREF)

- While sharing PLL Reference clock pin between interfaces the banks must be consecutive

- Interfaces using same IO standard can share OCT and RZQ Pin

- A bank cannot be used as A/C bank for two or more interfaces
  - Reason: hard controller and sequencer cannot be shared

- A lane cannot be shared
  - Reason: only one DQSin tree per lane; a lane can only talk to one controller
Pin Guidelines for Sharing Multiple Interfaces (Steps)

1. Determine total number of interfaces required

2. Determine No. of banks based on interface width and No. of interfaces

3. Ensure that interfaces meet the criteria for sharing interfaces

4. Plan Interface placement in column(s)
   a. Select middle bank for sharing DQ pins between two interfaces

5. For each interface, constrain CK pin to selected A/C bank or pin name

6. Constrain PLL reference clock to the A/C bank for one of the interfaces only

7. For each interface, constrain one DQS pin in each DQS group to a pin name or Bank.
Arria 10 board design guidelines are preliminary and subject to change
Guidelines

- Following guidelines are covered in the subsequent slides and they apply to both DDR3 and DDR4
  - Generic guidelines
  - Length
  - DQ to DQS delay
  - Address command (should include the package delay?)
  - Delay within the group
  - DQS to CK guideline

- Length matching guidelines are recommendations and they should not be considered as hard guidelines

- Customer must perform necessary board level simulation to make sure there are no signal integrity, ISI and crosstalk related issues

- Customers must also enter accurate information in the ‘Board Timing’ tab of the memory MegaCore and compile the design to ensure there are no timing violations
Trace impedance plays an important role in the signal integrity

- Users must perform board level simulation to determine the best characteristic impedance for their PCB
  - For example, it is possible that for multi rank systems 40 ohm would yield better result than a traditional 50 ohm characteristic impedance

- To minimize PCB layer propagation variance, Altera recommend that you route signals from the same net group on the same layer
  - Use 45° angles (*not* 90° corners)
  - Disallow critical signals across split planes
  - Route over appropriate VCC and GND planes
  - Keep signal routing layers close to GND and power planes
  - Avoid routing memory signals closer than 0.025 inch (0.635 mm) to memory clocks
Maximum Lengths

- **For DIMM**
  - From FPGA to DIMM connector max allowed trace length is 4.5 inches.
  - Maximum DIMM to DIMM distance is 0.425 inches

- **For Discrete components**
  - 7 inches maximum for address/command signal
  - 5 inches maximum for DQ/DQS/DM
- Match the (package + board) trace delays up to 20 ps of skew for DQ/DQS/DM signals within a DQS group.

- Details on how to do package de-skew is available in EMIF HB vol2 chapter 4.
All the address, command and control signals should match up to +/- 20 ps compare to the mem_clk trace

- For example if the mem_clk trace delay is 500 ps then the allowed range for any address/command/control signal is 480 ps to 520 ps

- For discrete components; make sure above recommendation is met for each component in the fly-by chain

- For DIMMs: For single or multiple DIMM configuration make sure this guideline is met at each DIMM connector
- \( x = y +/\ 20 \text{ ps} \)
- \( x + x_1 = y + y_1 +/\ 20 \text{ ps} \)
- \( x + x_1 + x_2 = y + y_1 + y_2 +/\ 20 \text{ ps} \)
- \( X + x_1 + x_2 + x_3 = y + y_1 + y_2 + y_3 +/\ 20 \text{ ps} \)
DQS-CLK guideline

- The timing between the DQS and clock signals on each device calibrates dynamically to meet tDQSS. To make sure the skew is not too large for the leveling circuit’s capability
  1. Propagation delay of clock signal must not be shorter than propagation delay of DQS signal at every device: \((CK_i - DQSi) > 0; 0 < i < \text{number of components} - 1\)
  2. Total skew of CLK and DQS signal between groups is less than one clock cycle: 
     \((CK_i + DQSi)_{\max} - (CK_i + DQSi)_{\min} < 1 \times tCK\)

- If you are using a DIMM topology, your delay and skew must take into consideration values for the actual DIMM.
Simulation Guideline

1. Using board sim tool such as Hyperlynx, set up a trace models which includes IBIS/Hspice buffer models for FPGA and memory, DIMM connector model (if applicable) and accurate board stack up

2. Use this setup to extract accurate trace delay and ISI information

3. Export memory interface layout into the board sim tool to run the board level simulation

4. Use PDN analysis tools to simulate the power supply noise
Users must enter accurate information about various delays and the skew in the MegaWizard.

**Timing analysis**
- DDR timing analysis scripts take board skews into account when generating timing analysis report.
- Inaccurate board skew parameters will result in inaccurate timing analysis of the memory interface.

**Delay Chain Settings**
- Physical delays are applied to delay chains to compensate for the skew mismatch between various signals.
- Board skew parameters affect the initial value applied to the delay chains.

Altera recommends that you simulate your interface in Hyperlynx (or similar tool) to acquire trace delays.
- You can use the [Board Skew Parameters Tool](#) available on Altera website to calculate the parameter once you have acquired the trace delays.
As the operating frequencies are pushing beyond 1 GHz, it is becoming increasingly important that user enters accurate slew rates and ISI/crosstalk information.

Customers should perform board simulation on the external memory interfaces and acquire all the necessary slew rates and ISI/crosstalk related information, and enter that information into the MegaWizard.

- Do not use the default values.

Accurate information about slew rates and ISI/crosstalk will result in accurate timing analysis of the interface.

Refer to EMIF HB Vol2 Chapter 9 for further information about ‘Board Timing’ parameters.
Thank You