Creating Heterogeneous Memory Systems in Intel® FPGA SDK for OpenCL Custom Platforms

Subscribe

Send Feedback

AN-789 | 2016.12.13

Latest document on the web: PDF | HTML

Contents

1. Creating Heterogeneous Memory Systems in Intel FPGA SDK for OpenCL Custom Platforms

The implementation of heterogeneous memory in a Custom Platform allows for more external memory interface (EMIF) bandwidth as well as larger and faster memory accesses. The combination of heterogenous memory access with an optimized OpenCL™[1] kernel can result in significant performance improvements for your OpenCL system.

This application note provides guidance on creating heterogeneous memory systems in a Custom Platform for use with the Intel® FPGA SDK for OpenCL[2]. Intel assumes that you are an experienced FPGA designer who is developing Custom Platforms that contains heterogeneous memory systems.

Prior to creating the heterogeneous memory systems, familiarize yourself with the Intel FPGA SDK for OpenCL documents specified below.

Related Information

- Intel FPGA SDK for OpenCL Programming Guide
- Intel FPGA SDK for OpenCL Best Practices Guide
- Intel FPGA SDK for OpenCL Arria 10 GX FPGA Development Kit Reference Platform Porting Guide

1.1. Verifying the Functionality of the FPGA Board and the EMIF Interfaces

Verify each memory interface independently and then instantiate your Custom Platform using global memory.

1. Verify each memory interface using hardware designs that can test the speed and stability of each interface.
2. Instantiate your Custom Platform using global memory.

   For example, if you have three DDR interfaces, one of them must be mapped as heterogeneous memory. In this case, verify the functionality of the OpenCL stack with each DDR interface independently.

[1] OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission of the Khronos Group™.

[2] The Intel FPGA SDK for OpenCL is based on a published Khronos Specification, and has passed the Khronos Conformance Testing Process. Current conformance status can be found at www.khronos.org/conformance.

Alternatively, if you have two DDR interfaces and one quad data rate (QDR) interface, verify the functionality of the OpenCL stack of the two DDR interfaces and the QDR interface independently.

Intel recommends that you use PCI Express®- (PCIe®-) or EMIF-exclusive designs to test your memory interfaces. After you verify that each memory interface is functional and that your OpenCL design works with a subset of the memory interfaces, proceed to create a fully functional heterogeneous memory system.

1.2. Modifying the board_spec.xml File

Modify the `board_spec.xml` file to specify the types of heterogeneous memory systems that are available to the OpenCL kernels.

During kernel compilation, the Intel FPGA SDK for OpenCL Offline Compiler assigns kernel arguments to a memory based on the buffer location argument that you specify.

1. Browse to the `board_spec.xml` file in the hardware directory of your Custom Platform.
2. Open the `board_spec.xml` file in a text editor and modify the XML accordingly.

   For example, if your hardware system has two DDR memories as default global memory and two QDR banks that you model as heterogeneous memory, modify the memory sections of the `board_spec.xml` file to resemble the following:

```
<!-- DDR3-1600 -->
<global_mem name="DDR" max_bandwidth="25600" interleaved_bytes="1024"
            config_addr="0x018">
  <interface name="board" port="kernel_mem0" type="slave" width="512"
             maxburst="16" address="0x00000000" size="0x100000000"
             latency="240"/>
  <interface name="board" port="kernel_mem1" type="slave" width="512"
             maxburst="16" address="0x100000000" size="0x100000000"
             latency="240"/>
</global_mem>

<!-- QDRII -->
<global_mem name="QDR" max_bandwidth="17600" interleaved_bytes="8"
            config_addr="0x100">
  <interface name="board" type="slave" width="64" maxburst="1"
             address="0x200000000" size="0x1000000" latency="1
             addpipe="1">
    <port name="kernel_qdr0_r" direction="r"/>
    <port name="kernel_qdr0_w" direction="w"/>
  </interface>
  <interface name="board" type="slave" width="64" maxburst="1"
             address="0x201000000" size="0x1000000" latency="150"
             addpipe="1">
    <port name="kernel_qdr1_r" direction="r"/>
    <port name="kernel_qdr1_w" direction="w"/>
  </interface>
</global_mem>
```

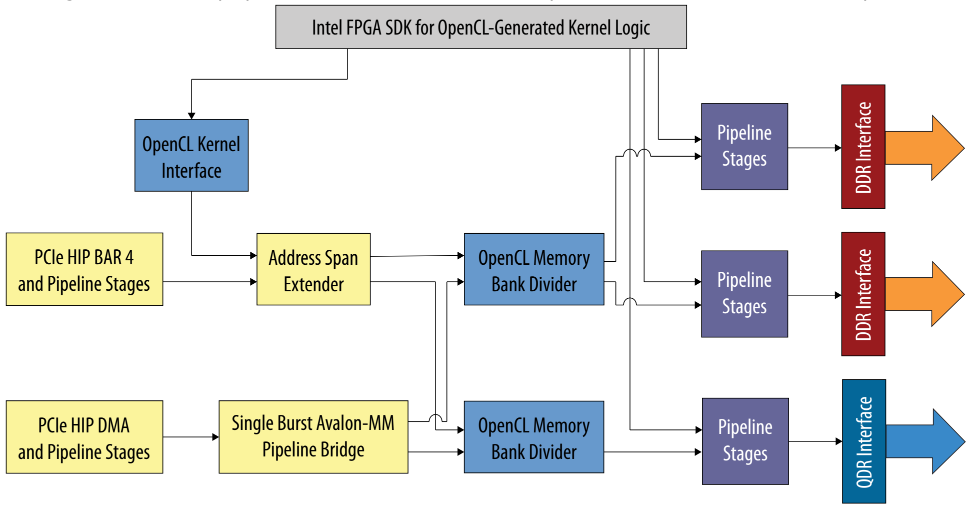
1.3. Setting Up Multiple Memory Dividers in Qsys

Currently, the OpenCL Memory Bank Divider in the the Qsys design does not support non-power-of-2 number of memory banks, which is not a limitation for typical configurations. However, there are scenarios where non-power-of-2 number of memory interfaces are necessary. To accommodate non-power-of-2 number of memory interfaces, use multiple OpenCL Memory Bank Dividers to create heterogeneous memory systems with non-power-of-2 number of memory banks.

You must create multiple OpenCL Memory Bank Dividers when you have a true heterogeneous memory system. Consider a system with one DDR memory interface and one QDR memory interface. Because the two banks have different memory topologies, you cannot combine them under a single global memory.

Figure 1. Block Diagram of a Three-Bank Heterogeneous Memory System

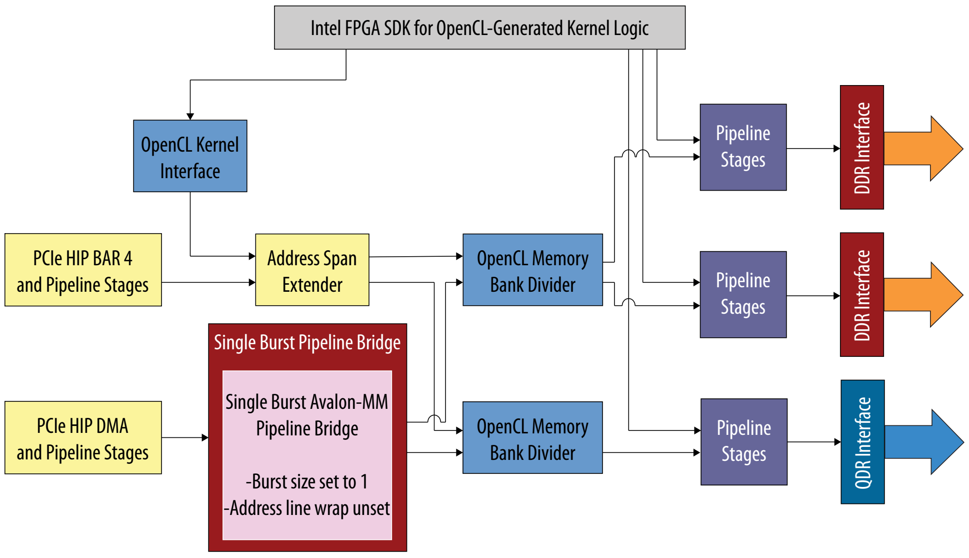
This heterogeneous memory system contains two DDR memory interfaces and one QDR memory interface.


Intel FPGA SDK for OpenCL-Generated Kernel Logic
OpenCL Kernel Interface
PCIe HIP BAR 4 and Pipeline Stages
Address Span Extender
OpenCL Memory Bank Divider
PCIe HIP DMA and Pipeline Stages
Single Burst Avalon-MM Pipeline Bridge
OpenCL Memory Bank Divider
Pipeline Stages
Pipeline Stages
Pipeline Stages
DDR Interface
DDR Interface
QDR Interface

If you are using version 16.0, 16.0.1, or 16.0.2 of the Quartus® Prime software and the Altera SDK for OpenCL, the OpenCL Memory Bank Divider incorrectly handles memory bursts across address boundaries. To work around this known issue, add a pipeline bridge with a burst size of 1 and connect its Avalon® Memory-Mapped (Avalon-MM) master to the OpenCL Memory Bank Divider's slave port.

Note: This known issue is fixed in the Quartus Prime software and the Intel FPGA SDK for OpenCL version 16.1.

Figure 2. Block Diagram of a Three-Bank Heterogeneous Memory System with a Pipeline Bridge


Intel FPGA SDK for OpenCL-Generated Kernel Logic
OpenCL Kernel Interface
PCIe HIP BAR 4 and Pipeline Stages
Address Span Extender
OpenCL Memory Bank Divider
Single Burst Pipeline Bridge
Single Burst Avalon-MM Pipeline Bridge
-Burst size set to 1
-Address line wrap unset
PCIe HIP DMA and Pipeline Stages
OpenCL Memory Bank Divider
Pipeline Stages
Pipeline Stages
Pipeline Stages
DDR Interface
DDR Interface
QDR Interface

1.4. Modifying the Boardtest Program and the Host Code for Your Heterogeneous Memory Solution

Use the `boardtest.cl` kernel that comes with the Intel FPGA SDK for OpenCL Custom Platform Toolkit to test the functionality and performance of your Custom Platform.

The `boardtest` program is an OpenCL kernel that allows you to test host-to-device bandwidth, memory bandwidth, and general functionality of your Custom Platform.

1. Browse to the `<`*`path to SDK installation`*`>/board/custom_platform_toolkit/tests/boardtest` directory.
2. Open the `boardtest.cl` file in a text editor and assign a buffer location to each global memory argument.

   For example:

   ```
   __kernel void
   mem_stream (__global__attribute__((buffer_location("DDR")))  uint *src,
               __global __attribute__((buffer_location("QDR"))) uint *dst,
               uint arg, uint arg2)
   ```

   Here, `uint *src` is assigned to DDR memory, and `uint *dst` is assigned to QDR memory. The `board_spec.xml` file specifies the characteristics of both memory systems.
3. To leverage your heterogeneous memory solution in your OpenCL system, modify your host code by adding the CL_MEM_HETEROGENEOUS_ALTERA flag to your `clCreateBuffer` call.

   For example:

   ```
   ddatain = clCreateBuffer(context,
                     CL_MEM_READ_WRITE | memflags |
   CL_MEM_HETEROGENEOUS_ALTERA,
   ```

```
                    sizeof(unsigned) * vectorSize,
                    NULL,
                    &status);
```

Intel strongly recommends that you set the buffer location as a kernel argument before writing the buffer. When using a single global memory, you can write the buffers either before or after assigning them to a kernel argument. In heterogeneous memory systems, the host sets the buffer location before writting the buffer. In other words, the host will call the `clSetKernelArgument` function before calling the `clEnqueueWriteBuffer` function.

In your host code, invoke the `clCreateBuffer`, `clSetKernelArg`, and `clEnqueueWriteBuffer` calls in the following order:

```
ddatain = clCreateBuffer(context,
                    CL_MEM_READ_WRITE | memflags |
CL_MEM_HETEROGENEOUS_ALTERA,
                    sizeof(unsigned) * vectorSize, NULL, &status);

…

status = clSetKernelArg(kernel[k], 0, sizeof(cl_mem), (void*)&ddatain);

…

status = clEnqueueWriteBuffer(queue, ddatain, CL_FALSE, 0,
                     sizeof(unsigned) * vectorSize,hdatain, 0, NULL, NULL);
```

The *ALTERAOCLSDKROOT*`/board/custom_platform_toolkit/tests/boardtest/host/memspeed.cpp` file presents a similar order of these function calls.

4. After you modify the `boardtest.cl` file and the host code, compile the host and kernel code and verify their functionality.

   When compiling your kernel code, you must disable burst-interleaving of all memory systems by including the `--no-interleaving` <*global_memory_type*> option in the aoc command.

Related Information

Disabling Burst-Interleaving of Global Memory (--no-interleaving <global_memory_type>)

1.5. Verifying the Functionality of Your Heterogeneous Memory System

To ensure that the heterogeneous memory system functions properly, unset the `CL_CONTEXT_COMPILER_MODE_ALTERA` flag in your host code.

In OpenCL systems with homogeneous memory, you have to option to set the `CL_CONTEXT_COMPILER_MODE_ALTERA=3` flag in your host code to disable the reading of the `.aocx` file and the reprogramming of the FPGA. Setting the `CL_CONTEXT_COMPILER_MODE_ALTERA=3` flag is useful when instantiating your board to verify the functionality of your Custom Platform without designing the floorplan and specifying the LogicLock™ regions.

With heterogeneous memory systems, the runtime environment must read the buffer locations of each buffer, described in the `.aocx` file, to verify the memory systems' functionality. However, you might want to verify the functionality of your Custom Platform without implementing the final features of the board design, such as designing the floorplan and specifying the LogicLock regions.

1. Verify that the `CL_CONTEXT_COMPILER_MODE_ALTERA` flag is unset in your host code.
2. Browse to the `board/<board name>/source/host/mmd` directory of your Custom Platform.
3. Open the `acl_pcie_device.cpp` memory-mapped device (MMD) file in a text editor.
4. Modify the `reprogram` function in the `acl_pcie_device.cpp` file by adding a `return 0;` line, as shown below:

```
int ACL_PCIE_DEVICE::reprogram(void *data, size_t data_size)
{
   return 0;

   // assume failure
   int reprogram_failed = 1;

   // assume no rbf or hash in fpga.bin
   int rbf_or_hash_not_provided = 1;

   // assume base and import revision hashes do not match
   int hash_mismatch = 1;
   ...
}
```

5. Recompile the `acl_pcie_device.cpp` file.
6. Verify that the `CL_CONTEXT_COMPILER_MODE_ALTERA` flag remains unset.

Attention: After you add `return 0;` to the reprogram function and recompile the MMD file, the runtime environment will read the `.aocx` file and assign the buffer locations but will not reprogram the FPGA. You must manually match the FPGA image with the `.aocx` file. To reverse this behavior, remove `return 0;` from the reprogram function and recompile the MMD file.

1.6. Document Revision History

Date	Version	Changes
December 2016	2016.12.13	Initial release.