Convey Wolverine Architecture Reference

September 18, 2014

Version 1.3

900-000043-000

Trademarks

Revisions

Version	Description
1.0	February 10, 2014. Initial Release
1.1	April 18, 2014. Misc. Corrections
1.2	June 22, 2014. Updates supporting driver
1.3	September 18, 2014. Minor updates

Table of Contents

1 Overview

1.1 Introduction

This manual describes the Convey Wolverine Computer Architecture. The architecture integrates two types of processor architecture in one system: the Intel 64 architecture implemented by an Intel microprocessor and a reconfigurable architecture implemented as a coprocessor designed and implemented by Convey.

Specialized personalities may be developed by Convey or its partners and customers that address specific high value applications.

2 Wolverine System Organization

2.1 System Architecture

Wolverine accelerators support the Convey Hybrid-Core Architecture, which tightly integrates coprocessor resources in a Linux-based x86 server. The accelerators support virtual to physical addressing, allowing the coprocessor and its onboard memory to be mapped in a process' virtual address space. The coprocessor has full access to memory residing on the host platform and the host processors have full access to memory on the coprocessor.

The coprocessor is dynamically reconfigurable and can be reloaded with accelerated applications called personalities. Personalities are hardware implementations of performance-critical regions of code. Personalities provide acceleration via multiple mechanisms such as.

- Pipelining
- Multithreading
- Replication

The Wolverine coprocessor interfaces to the PCIe Express. It shares the virtual address space of the host through Globally Shared Virtual Memory.

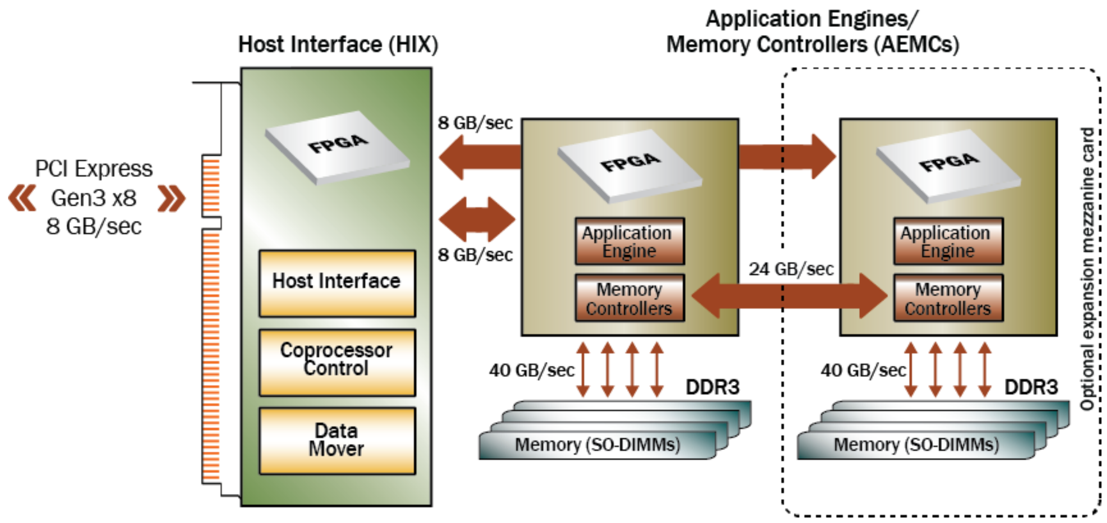

Figure 1: Convey Wolverine System Diagram

2.2 Coprocessor

The coprocessor has three major sets of functional components, referred to as the Application Host Interface (HIX), the Application Engines (AEs) and optional Memory Controllers (MCs).

2.2.1 Host Interface

The Host Interface (HIX) implements the PCIe interface to the host system, handles data movement between the coprocessor than host, and controls coprocessor execution. It processes data requests from the host processor, routing requests for addresses in coprocessor memory to the MCs. Coprocessor dispatches are passed to the AEs for execution. HIX logic is loaded at system boot time and is part of the fixed coprocessor infrastructure.

The HIX implements a PCIe 3.0, x8 electrical connection via an x16 physical slot. At boot time the coprocessor requests Memory Mapped IO High (MMIOH) to map the onboard coprocessor memory into the host's physical address space. This allows host processors to access coprocessor memory directly. Similarly, the HIX routes requests from processing elements in the AEMC to the host memory as required.

In addition to ordinary loads and stores, a data mover is incorporated into the HIX to initialize coprocessor memory or perform block copies between the host and coprocessor. The data mover can transfer data at up to 37GB/sec.

2.2.2 Application Engine

The Application Engine (AE) is dynamically loaded with the personalities that deliver accelerated application performance. Convey provided logic blocks implement the dispatch interface to the HIX, address translation, memory crossbar and interface to the coprocessor memory. These are incorporated with application specific kernels to implement a loadable personality.

2.2.3 Memory Subsystem

Wolverine coprocessors can be configured with local on board memory provided by four error correcting SO-DIMMs or as “memory free” accelerators with no local storage. Possible on board memory configurations are 16GB, 32GB or 64GB. Each SO-DIMM is is connected to the AEMC via its own 133MHz DDR3 channel, supporting an aggregate bandwidth of 40GB/sec.

For coprocessors with memory, the coprocessor memory system implements a distributed crossbar over a network of point-to-point links. Each Application Engine (AE) is connected to each Memory Controller (MC). An additional set of links connects each MC to the Host Interface.

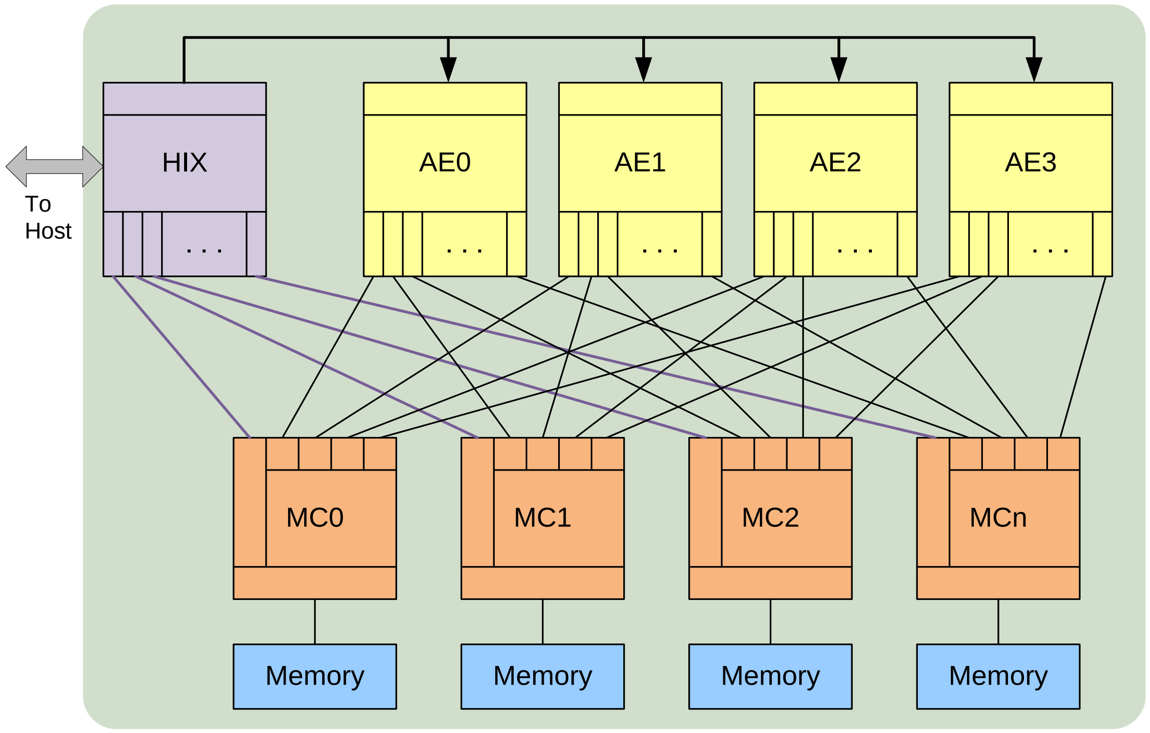

Figure 2: Convey Wolverine Diagram

Memory Controllers support memory channels, between the application engines and the memory subsystem. The MCs translate virtual to physical addresses and route references to host memory to the host. The Memory Controllers provide a path to coprocessor memory. Coprocessor memory is optimized for transfers of 64 byte accesses and provides suboptimal performance for transfers of less than 64 bytes. The coprocessor therefore not only has a much higher peak bandwidth than is available to commodity processors, but also delivers a much higher percentage of that peak for non-sequential accesses.

2.3 Coprocessor Programming Model

Processes that use the coprocessor contain Intel 64 host processor and coprocessor instructions and transfer control between the two. For performance reasons, it may be desirable to place data in the memory region associated with the host processor or coprocessor that uses it most. A data mover engine is built into the coprocessor and used for page zeroing, and data movement between host and coprocessor. Data mover functions are accessible via library calls.

Dispatching work to the coprocessor requires the following steps:

- Reserve a coprocessor
    - Reserves a coprocessor for a process
- Attach a process to the coprocessor
    - Loads required personalities into the coprocessor.
- Dispatch an application to the coprocessor.

    - Starts execution on the coprocessor

- Get dispatch complete status

The host processor can continue executing asynchronously, or wait for completion of the coprocessor dispatch. In either case the host processor must check dispatch status complete, before the coprocessor can be dispatched again.

Shared library routines perform the above functions.

The coprocessor is a dedicated resource: it executes one dispatch at a time, but a single process can use multiple coprocessors.

3 Data Types

The set of data types handled by the coprocessor were chosen to reflect the data types provided by high level programming languages that are used by Server Class Computing applications (C, C++ and FORTRAN).

The sections below talk about three sets of data types: fundamental, numeric and native. The fundamental data types represent the sizes of operands that the coprocessor is able to read and write from memory. The numeric data types are the set of numeric data representations that the coprocessor recognizes. The native data types are the subset of numeric data types that the coprocessor instruction set is able to manipulate with arithmetic and logical operations.

3.1 Fundamental Data Types

The fundamental data types represent the size of operands that the coprocessor reads and writes from memory. The fundamental data types are listed in Table 1 with the required memory alignment.

Fundamental Data Type	*Data Size (in bytes)*	*Alignment Requirement (in bytes)*
Byte	1	Any
Word	2	2
Double Word	4	4
Dual Double Word	8	4
Quad Word	8	8
64 Byte	64 (Read) 8 – 64 (Write)	64 (Read) 8 within 64 byte boundary (Write)

Table 1 - Fundamental Data Types

Note that all fundamental data types must be aligned on a natural boundary for that data type with the following exceptions

- Dual Double Words are able to be aligned on any 4-byte boundary.
- 64 Byte Writes are aligned on any 8-byte boundary, within the 64 byte boundary

Accesses to memory on unaligned boundaries cause a trap to the operating system.

Figure 3 shows how the fundamental data types can be located in memory.

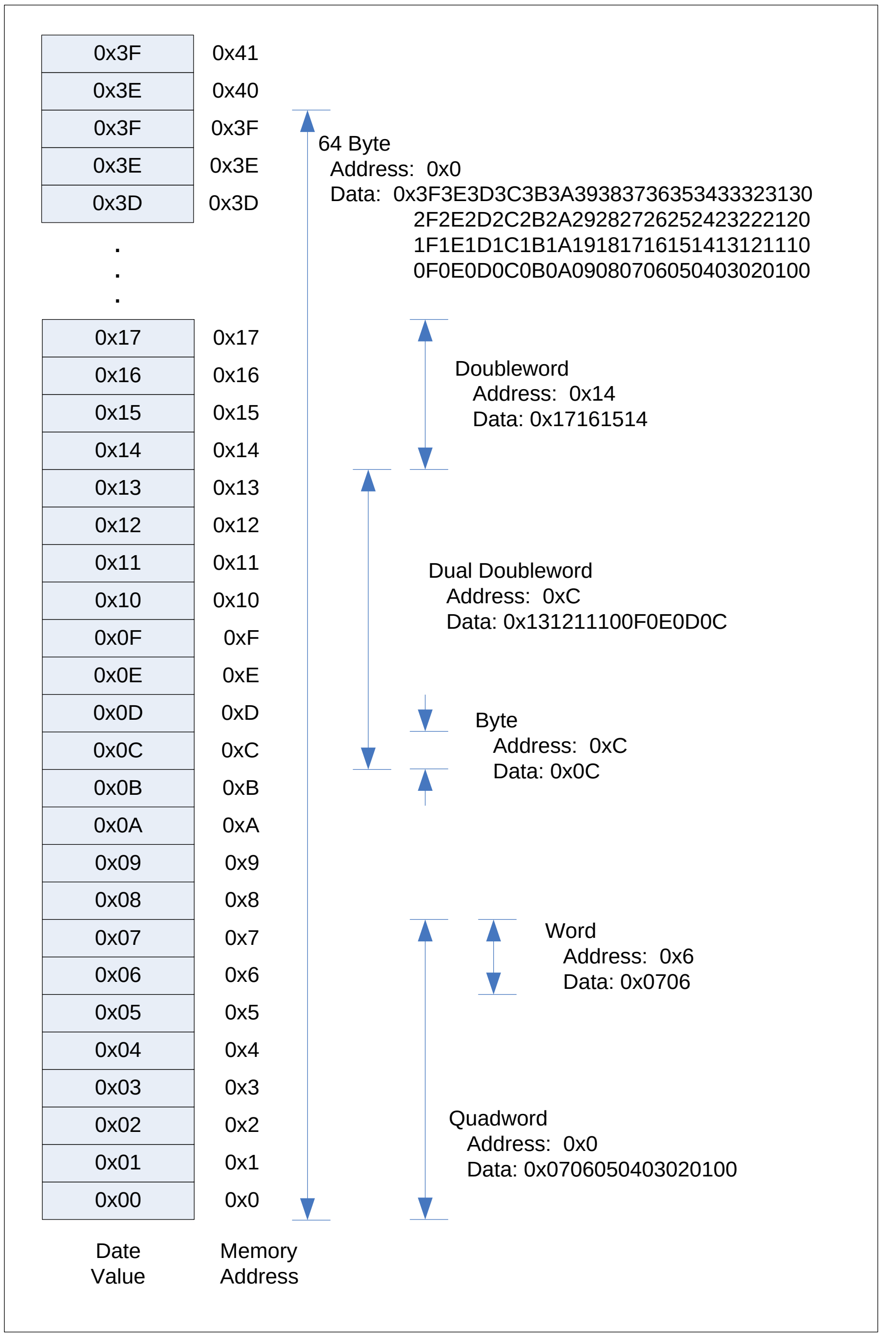

Figure 3 - Fundamental Data Types in Memory

3.2 Native Data Types

Data is accessed from memory in the fundamental memory units (1, 2, 4 or 8 bytes). Once written to a coprocessor internal register, all operations are performed on the native data types. The native data types include 64-bit integers, single precision, double precision, complex single precision and complex double precision.

The coprocessor loads integers from memory as 1, 2, 4 and 8 byte quantities. The data loaded from memory is either zero or sign extended to 64-bits as it is being written to an internal register. All integer operations are performed on the native 64-bit integer operand size. All arithmetic operations are checked for overflow/underflow for 64-bit values. Store operations store the native 64-bit values to the fundamental memory sizes of 1, 2, 4 or 8 bytes. Underflow/overflow is checked as the 64-bit values are read from the internal registers and converted to the smaller fundamental memory sizes.

3.3 Pointer Data Types

All pointers are 64-bits and are required to be in the IA32e 64-bit form. IA32e 64-bit form implies that virtual addresses are 64-bits, with bits 48-63 being the same value as bit 47. Since a pointer is an unsigned integer value, then with IA32e 64-bit addressing, half of the valid virtual address space is at the upper end of the 64-bit address range, and half is at the lower end of the 64-bit address range. Figure 4 illustrates the valid address range assuming the host processor supports a 48-bit virtual address. Future versions of processors that support the IA32e 64-bit architecture may support virtual addresses wider than 48-bits.

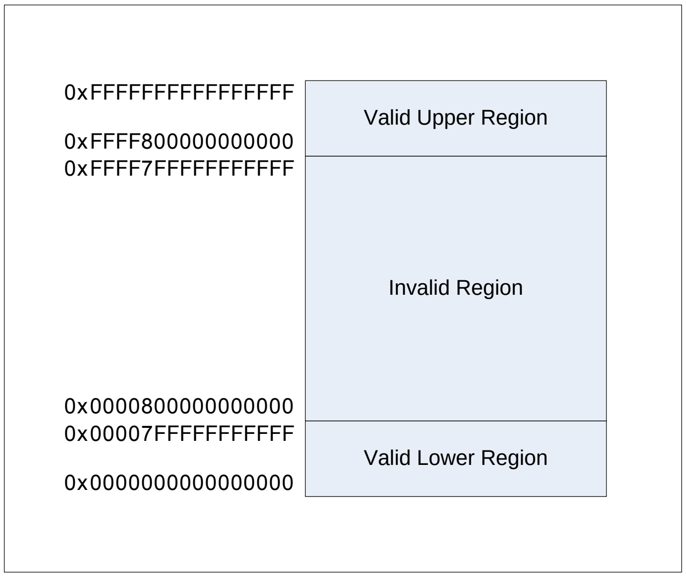

Figure 4 - Valid Virtual Address Regions

4 Addressing and Access Control

4.1 Introduction

The x86 host processor's architecture defines many address translation modes, privilege levels, and legacy compatibility mechanisms. The Convey coprocessor defines a single address translation mode (64-bit), minimal access protection checks, and no support for legacy compatibility with previous generations of x86 processors. All x86 applications that require legacy compatibility execute entirely on the x86 processor.

All operating system calls are performed on the x86 processor. As such, the coprocessor does not support changing execution privilege levels. A user application is only able to dispatch a user privilege level routine. The operating system can dispatch a supervisor level routine (I.e. context save or restore). Neither a user level nor a privileged level routine can change the privilege level during execution. As a result, checking mechanisms to validate legal privilege level changes are not required.

4.2 Bit and Byte Ordering

The Convey Coprocessor architecture uses the little endian byte ordering model. This model has bits numbered from right to left, starting with the right most, least significant bit being bit zero. Similarly, bytes are numbered with the least significant byte of a data element being byte 0.

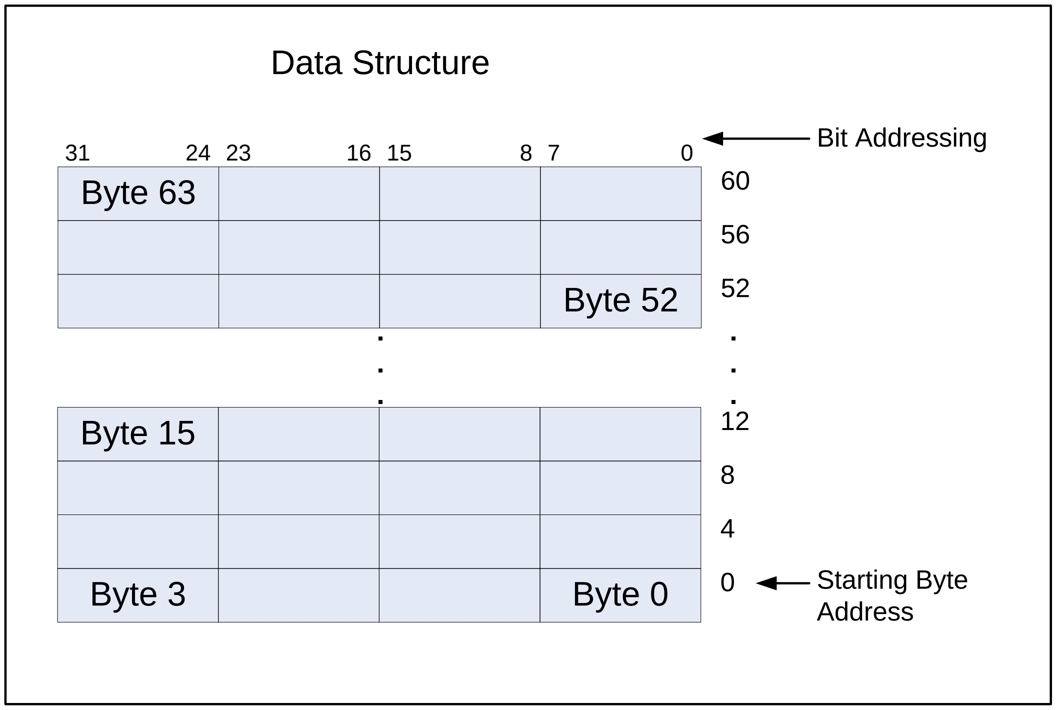

Figure 5 - Data Structure Bit and Byte Ordering

4.3 Pointers and Address Specification

All Convey coprocessor virtual addresses are 64-bits in width and must follow the Intel canonical form (see Section 3.3 Pointer Data Types).

4.4 Address Resolution and the TLB

The Convey coprocessor handles page faults to resolve address translations when the needed TLB entry is not present. The coprocessor includes a hardware based state machine that is able to walk memory based translation tables to obtain the needed TLB entry. The host processor and coprocessor share the same memory based address translation tables. The host processor manages and uses the translation tables, whereas the coprocessor only uses the translation tables to resolve pages faults.

Upon the occurrence of a page fault, the coprocessor first walks the physical memory based address translation tables. If the needed TLB entry is found then the coprocessor TLB is updated and coprocessor execution resumes. If the needed TLB entry is not found, then the coprocessor sends an interrupt to the host processor. The host processor must then handle the page fault by either determining that the page fault address is illegal for the coprocessor application (and terminating the application), or by adding the needed TLB entry to the memory based translation table (and telling the coprocessor to search the memory based translation tables again).

4.4.1 Supported Address Translation Mode

The x86 host processor supports multiple address translation modes and multiple page sizes. The coprocessor supports only one address translation mode with multiple page sizes. The supported address translation mode is referred to as 64-bit mode.

Applications that use the coprocessor must use 64-bit address translation mode. All other applications can run on the host processor without coprocessor support using any of the operating system supported address translation modes.

64-bit mode is defined such that segmentation registers are not used.

64-bit mode supports two pages sizes, 4K and 2M bytes. The Convey coprocessor supports all power-of-two pages sizes between 4K and 2M byte and 64M byte pages. The support of very large page sizes allows the coprocessor TLB to cover all physical memory.

4.4.2 Page Size

The following table shows the supported page sizes, the number of base address bits, and the number of virtual address bits used to construct the 40-bit physical address.

Page Size	*Page Base Address bits*	*Page Offset Address bits*
4K	39-12	11-0
8K	39-13	12-0
16K	39-14	13-0
32K	39-15	14-0
64K	39-16	15-0
128K	39-17	16-0
256K	39-18	17-0

Page Size	*Page Base Address bits*	*Page Offset Address bits*
512K	39-19	18-0
1M	39-20	19-0
2M	39-21	20-0
64M	39-26	25-0