OpenCL™ 1.1 for 66AK2H

Product Version 0.8.1 Readme

Publication Date: February 27, 2014

* Product is based on a published Khronos Specification, and is expected to pass the Khronos Conformance Testing Process. Current conformance status can be found at www.khronos.org/conformance.

Contents

1 Product Description **2**

2 OpenCL Documentation **2**

3 TI OpenCL Extensions **2**

4 Limitations of this Beta OpenCL Implementation **5**

5 Notes **6**

6 Examples **7**
6.1 Building and Running the Examples 7
6.2 Platform Example 7
6.3 Simple Example 7
6.4 Mandelbrot/Mandelbrot_native Examples 8
6.5 Ccode Example 8
6.6 Matmpy Example 8
6.7 Offline Example 9
6.8 Offline_embed Example 9
6.9 Vecadd_openmp Example 9
6.10 Vecadd_openmp_t Example 9
6.11 Openmpbench_c_v3 Example 9
6.12 Vecadd Example 10
6.13 Vecadd_mpax Example 10
6.14 Vecadd_mpax_openmp Example 10
6.15 Dsplib_fft Example 10
6.16 Ooo and Ooo_map Examples 10
6.17 Null Example 11
6.18 Edmamgr Example 11

7 Environment Variables **12**

8 Installation and Dependencies **13**
8.1 CMEM module 13
8.2 MPM 13
8.3 MPM-TRANSPORT 14
8.4 Required Environment Variables 14
8.5 Additional package dependencies 14

1 Product Description

This product is an OpenCL 1.1 implementation. The OpenCL specification defines a platform model with a host and compute devices. For this implementation the host is a 4-core ARM Cortex-A15 running Linux. There are two compute devices:

1. The collection of 8 Texas Instruments' C66x DSP cores is exposed as one virtual accelerator compute device, and

2. The collection of 4 ARM Cortex-A15 cores is exposed as a virtual CPU device. (note: The CPU device currently only supports native kernels)

This OpenCL implementation is typically installed as part of the TI MCSDK-HPC product and environment variables and dependencies are handled automatically as part of the MCSDK-HPC installation. An installation section is still documented here for your knowledge and troubleshooting. See section 8 Installation and Dependencies as needed.

2 OpenCL Documentation

The OpenCL 1.1 specification and the 1.1 C++ bindings specification from Khronos are included in the `$(TI_OCL_INSTALL)/doc` sub-directory of this installation.

Additional OpenCL resources can be found on the web. Some useful links are provided below.

- The OpenCL 1.1 on-line manual pages can be found at: `http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml`

- The Khronos OpenCL resources page also has links to good OpenCL reference material: `http://www.khronos.org/opencl/resources`

3 TI OpenCL Extensions

This OpenCL implementation from Texas Instruments has been extended with a set of features beyond the OpenCL 1.1 specification. These features were added in order to better support the excution of code on the C66 DSP, to enable existing DSP libraries, and to better map to the 66AK2H hardware platform.

1. This OpenCL implementation supports the ability to call standard C code from OpenCL C kernels. This includes calling functions from existing C66 DSP libraries, such as the dsplib or mathlib. For examples of this capability please refer to the ccode example in section 6.5 for an example of calling a C function you define, or the dsplib_fft example in section 6.15 for an

example calling a function in a library.

2. Additionally the called standard C code can contain OpenMP pragmas. When using this feature the OpenCL C kernel containing the call to an OpenMP enabled C function must be submitted as a task (not an NDRangeKernel) and it must be submitted to an in-order OpenCL command queue (i.e not defined with the `CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE` flag). In this scenario, OpenCL will dispatch the kernel to one compute unit of the DSP accelerator and the OpenMP runtime will manage distribution of tasks across the compute units. Please see examples vecadd_openmp in section 6.9, vecadd_openmp_t in section 6.10 or openmpbench_C_v3 in section 6.11

3. A printf capability has been added for kernels running on the DSP. OpenCL C kernels or standard C code called from an OpenCL C kernel can call printf. The string resulting from the printf will be transmitted to the host ARM and displayed using a printf on the ARM side. This feature can be used to assist in debug of your OpenCL kernels. Note that there is a performance penalty in using printf from the DSPs, so it is not a feature that should be used when evaluating DSP performance. This feature is not the OpenCL 1.2 printf which contains additional formatting codes for printing vector types.

4. A TI extension for allocating buffers out of MSMC memory has been added. Other than the location on the DSP where the buffer will reside, this MSMC defined buffer will act as a standard global buffer in all other ways. Example:

   ```
   Buffer bufMsmc(context, CL_MEM_READ_ONLY|CL_MEM_USE_MSMC_TI, size);
   Buffer bufDdr (context, CL_MEM_READ_ONLY, size); }
   ```

   The matmpy example in section 6.6 illustrates the use of MSMC buffers. The platform example in section 6.2 will query the DSP device and report the amount of MSMC memory available for OpenCL use.

   MSMC stands for Multicore Shared Memory Controller and conaints is an on-chip memory shared across all ARM and DSP cores on the chip.

5. The OpenCL C compiler for the C66 DSP supports the C66x standard C compiler set of intrinsic functions, with the exception of those intrinsics that accept or result in a 40 bit value. Please refer to the C6000 Compiler User's Guide for a list of these intrinsic functions.

6. Additionally these non standard OpenCL C built-in functions are supported:

   ```
   uint32_t __core_num     (void);
   uint32_t __clock        (void);
   uint64_t __clock64      (void);
   void     __cycle_delay  (uint64_t cyclesToDelay);
   void     __mfence       (void);
   ```

   `__core_num` returns [0-7] depending on which DSP core executes the function.

`__clock` return a 32-bit time-stamp value, subtracting two values returned by `__clock` gives the number of elapsed DSP cycles between the two. This equates to the C66 device's TSCL register.

`__clock64` return a 64-bit time-stamp value similar to `__clock` but with more granularity to avoid potential overflow of a 32 bit counter. This equates to the C66 device's TSCH:TSCL register pair.

`__cycle_delay` takes a specified number of cycles to delay and will busy loop for that many cycles (approximately) before returning.

`__mfence` is a memory fence for the C66x dsp. Under typical OpenCL use, this will not be needed. However, when incorporating EDMA usage into OpenCL C kernels, it may be needed.

Note: In standard C for C66 a uint32_t is an unsigned int and a uint64_t is an unsigned long long. In OpenCL C for C66 a uint32_t is an unsigned int and a uint64_t is an unsigned long.

7. This OpenCL implementation also supports direct access to the 66AK2H EDMA system from the DSP and OpenCL C kernels. A wide range of EDMA constructs are supported. These include 1D to 1D, 1D to 2D, 2D to 1D, and chained transfers. The edmamgr example in section 6.18 shows an example usage of this feature.

8. An extended memory feature is also supported in this implementation of OpenCL. The C66 DSP is a 32-bit architecture and has a limit of 2GB of DDR that it can access at any given time. The 66AK2H platforms can support up to 8GB of DDR3. To enable usage of DDRs greater than 2GB, this OpenCL implementation can use a hardware mapping feature to move windows over the 8GB DDR into the 32-bit DSP address space. Movement of these windows will occur at kernel start boundaries so two sequential kernels dispatched to the DSP device may actually operate on different 2GB areas within the 8GB DDR. The windows are not moved within a kernel. As a result of this feature, large buffers may be created and subsequently populated on the ARM side. However, a dispatched kernel may not access the entire buffer in one dispatch. Any given OpenCL Kernel will be limited to a total of 2GB of DDR access. The example vecadd_mpax in section 6.13 illustrates a process of defining a large buffer and then defining sub-buffers within the larger buffer and dispatching multiple OpenCL kernels to the DSP on these sub-buffers, cumulatively resulting in the entire large buffer being processed.

4 Limitations of this Beta OpenCL Implementation

This is a Beta version of this product. It is not a complete OpenCL implementation and has not successfully completed the Khronos conformance tests yet. Specifically, the following features are missing:

- This OpenCL implementation is not designed to allow multiple OpenCL enabled Linux processes to concurrently execute. Attempting to execute two OpenCL applications concurrently can leave the DSPs in an unknown state and may require a reboot. This will be resolved in a future OpenCL release.
- Dispatching OpenCL C kernels to the CPU device with clEnqueueTask or clEnqueueNDRangeKernel is not currently supported. The CPU is still exposed in the platform because you can submit native code to the CPU using clEnqueueNativeKernel. This is illustrated in the ooo and ooo_map examples 6.16 in this installation. All dispatch models are supported for the DSP (accelerator) device.
- The OpenCL API functions are complete for the DSP device, with the exception of argument passing to kernels. The implementation today will support at least 19 arguments and as many as 29 arguments to kernels. The limit is variable based on the size of the arguments. Additionally, structures and vector types are not supported as arguments yet. Buffers of vector types are supported.
- Not all of the OpenCL C built-in functions are available yet. Missing are:
  - the convert_type built-ins where saturation or explicit rounding modes are specified from section 6.2.3 in the OpenCL 1.1 spec. The convert_type built-ins without saturation of explicit rounding are supported.
  - the atomic built-ins from section 6.11.11 in the OpenCL 1.1 spec
  - the shuffle and shuffle2 built-ins from section 6.11.12 in the OpenCL 1.1 spec.
  - the following math built-ins from section 6.11.2 in the OpenCL 1.1 spec: cbrt, copysign, erfc, erf, expm1, fdim, fma, fmax, fmin, fract, frexp, hypot, ilogb, ldexp, lgamma, lgamma_r, log1b, logb, maxmag, minmag, modf, nan, pown, powr, remainder, remquo, rint, rootn, round, sincos, tgamma, trunc
- The half (16bit) floating point format is not supported.
- Support for images and samplers is optional for non GPU devices per the OpenCL 1.1 spec and they are not supported by the DSP device in this implementation.
- The device info query for frequency on the CPU device is not accurate. It currently returns a fixed value of 1.0 Ghz.

- The OpenCL C built-in functions are not yet optimized for the C66 dsp device.
- The C66x DSP compiler used to compile OpenCL C kernels is not yet optimized for wide vector types. The resultant code can be inefficient and the compile time can be long on some code samples using 8 or 16 wide vectors. It is recommended that vector widths over 4 (2 for doubles and longs) be avoided in this release.

5 Notes

On-line compilations of OpenCL C code are cached on the system. If you run an OpenCL application that on-line compiles some OpenCL C code, the resultant binaries are cached on the system and the next time you run the OpenCL application, the compilation step is skipped and the cached binaries are used. The caching only uses the OpenCL C code and the compile options as a hash, so an example where the OpenCL C code is calling a C function in a linked object file or library and the object file or library is modified will result in an execution of the OpenCL C linked against the older version of the object. In this case you will need to clear the OpenCL C compile cache, which can be accomplished with the command

```
rm -f /tmp/opencl*.
```

Additionally you can disable this caching behavior by setting the `TI_OCL_CACHE_KERNELS_OFF` enviroment variable. See the Enviroment Variable section 7 for more details.

6 Examples

There are several OpenCL examples shipped with the product. They are located in the 'examples' directory within the OpenCL package.

The examples can be cross-compiled in an X86 development environment, or compiled native on the ARM A15, depending on the availability of native g++ or cross-compiled arm-linux-gnueabihf-g++ tool sets.

The example makefiles are setup to cross-compile by default and assume an ARM cross-compile environment has been installed. If the cross compiler is not installed, execute the following command to install it:

```
sudo apt-get install g++-4.6-arm-linux-gnueabihf
```

The environment variables defined in the Installation section /refinstallation are required for correct operation of all examples below.

6.1 Building and Running the Examples

From your X86 system: All the examples can be built at one time by invoking 'make' from the 'examples' directory.

Individual examples can be built by navigating to the desired directory and also issuing `make [cross]`.

Once the example is built and the resulting executable is available on the Hawking EVM file system it can be invoked from within a Hawking EVM xterm or console window.

6.2 Platform Example

The platform example uses the OpenCL C++ bindings to discover key platform and device information from the OpenCL implementation and print it to the screen.

6.3 Simple Example

This example simply illustrates the minimum steps needed to dispatch a kernel to a DSP device and read a buffer of data back.

6.4 Mandelbrot/Mandelbrot_native Examples

The Mandelbrot example is an OpenCL demo that uses OpenCL to generate the pixels of a Mandelbrot set image. This example also uses the C++ OpenCL binding. The OpenCL kernels are repeatedly called generating images that are zoomed in from the previous image. This repeats until the zoom factor reaches 1E15.

This example illustrates several key OpenCL features:

- OpenCL queues tied to potentially multiple DSP devices and a dispatch structure that allows the DSPs to cooperatively generate pixel data,
- The event wait feature of OpenCL,
- The division of one time setup of OpenCL to the repetitive en-queuing of kernels, and
- The ease in which kernels can be shifted from one device type to another.

The 'mandelbrot_native' example is non-OpenCL native implementation (no dispatch to the DSPs) that can be used for comparison purposes. It uses OpenMP for dispatch to each ARM core.

Note: The display of the resulting Mandelbrot images is disabled when this example is cross compiled. It relies on libsdl capability which is not currently supported on the default Linux file-system included in the MCSDK.

6.5 Ccode Example

This example illustrates the TI extension to OpenCL that allows OpenCL C code to call standard C code that has been compiled off-line into an object file or static library. This mechanism can be used to allow optimized C or C callable assembly routines to be called from OpenCL C code. It can also be used to essentially dispatch a standard C function, by wrapping it with an OpenCL C wrapper. Calling C++ routines from OpenCL C is not yet supported. You should also ensure that the standard C function and the call tree resulting from the standard C function do not allocate device memory, change the cache structure, or use any resources already being used by the OpenCL runtime.

6.6 Matmpy Example

This example performs a 1K x 1K matrix multiply using both OpenCL and a native ARM OpenMP implementation (GCC libgomp). The output is the execution time for each approach (OpenCL dispatch to the DSP vs. OpenMP dispatching to the 4 ARM A15s).

This example illustrates the use of local buffers, gloal buffers mapped to MSMC using the TI MSMC entension and async_workgroup_strided_copy for loading a global buffer into the local buffer. The async_workgroup_strided_copy built-in function currently just performs a memcpy and therefore the data movement and compute do not overlap, but the structure of the code is illustrative of how this will work in a later release where the async_workgroup_strided_copy will use the device's EDMA capability to offload the data movement from the DSP device.

6.7 Offline Example

This example performs a vector addition by pre-compiling an OpenCL kernel into a device executable file. The OpenCL program reads the file containing the pre-compiled kernel in and uses it directly. If you use offline compilation to generate a .out file containing the OpenCL C program and you subsequently move the executable, you will either need to move he .out as well or the OpenCL application will need to specificy a non relative path to the .out file.

6.8 Offline_embed Example

Similar to the offline example, but instead of compiling the kernel into a binary file, an embeddable header file is created which can be compiled directly into the host application.

6.9 Vecadd_openmp Example

This is an OpenCL + OpenMP example. OpenCL program is running on the host, managing data transfers, and dispatching an OpenCL wrapper kernel to the device. The OpenCL wrapper kernel will use the ccode mode (see ccode example) to call the C function that has been compiled with OpenMP options (–omp). To facilitate OpenMP mode, the OpenCL wrapper kernel needs to be dispatched as an OpenCL Task to an In-Order OpenCL Queue.

6.10 Vecadd_openmp_t Example

This is another OpenCL + OpenMP example, similar to vecadd_openmp. The main difference w.r.t vecadd_openmp is that this example uses OpenMP tasks within the OpenMP parallel region to distribute computation across the DSP cores.

6.11 Openmpbench_c_v3 Example

This OpenCL + OpenMP example is derived from EPCC OpenMP microbenchmarks, v3.0. The syncbench test was modified to dispatch using OpenCL.

6.12 Vecadd Example

The same functionality as the vecadd_openmp example, but expressed fully as an OpenCL application without OpenMP. Included for comparison purposes.

6.13 Vecadd_mpax Example

The same functionality as the vecadd example, but with extended buffers. The example iteratively traverses smaller chunks (sub-buffers) of large buffers. During each iteration, the smaller chunks are mapped/unmapped for read/write. The sub-buffers are then passed to the kernels for processing. This example could also be converted to use a pipelined scheme where different iterations of CPU computation and device computation are overlapped.

NOTE: The size of the buffers in the example (determined by the variable 'NumElements') is dependent on the available CMEM block size. Currently this example is configured to use buffers sizes for memory configurations that can support 1.5 GB total buffer size. The example can be modified to use more (or less) based on the platform memory configuration.

6.14 Vecadd_mpax_openmp Example

Similar to vecadd_mpax example, but used OpenMP to perform the parallelization and the computation. This example also illustrates that printf() could be used in OpenMP C code for debugging purpose.

6.15 Dsplib_fft Example

An example to compute FFT's using a routine from the dsplib library. This illustrates Calling a standard C library function from an OpenCL kernel.

6.16 Ooo and Ooo_map Examples

This Application illustrates several features of OpenCL.

- Using a combination of In-Order and Out-Of-Order queues
- Using native kernels on the CPU
- Using events to manage dependencies among the tasks to be executed. A JPEG in this directory illustrates the dependence graph being enforced in the application using OpenCL

events.

The Ooo_Map version additionally illustrates the use of OpenCL map and unmap operations for accessing shared memory between a host and a device. The Map/Unmap protocol can be used instead of read/write protocol on shared memory platforms.

6.17 Null Example

This application is intended to report the time overhead that OpenCL requires to submit and dispatch a kernel. A null(empty) kernel is created and dispatched so that the OpenCL profiling times queried from the OpenCL events reflects only the OpenCL overhead necessary to submit and execute the kernel on the device. This overhead is for the roundtrip for a single kernel dispath. In practice, when multiple tasks are being enqueued, this overhead is pipelined with execution and can approach zero.

6.18 Edmamgr Example

This application illustrates how to use the edmamgr api to asynchronously move data around the DSP memory hierarchy from OpenCL C kernels. The edmamgr.h header file in this directory enumerates the APIs available from the edmamgr package

7 Environment Variables

TI_OCL_INSTALL	Must be set to the top level directory, where your OpenCL installations resides.
TI_OCL_CGT_INSTALL	Must be set to the top level directory, where your C66 DSP compiler tools installation resides.
TI_OCL_KEEP_FILES	When OpenCL C kernels are compiled for DSPs, they are compiled to a binary .out file in the /tmp sub-directory. They are then subsequently available for download to the DSPs for running. The process of compiling generates several intermediate files for each source file. OpenCL typically removes these temporary files. However, it can sometimes be useful to inspect these files. This environment variable can be set to instruct the runtime to leave the temporary files in /tmp. This can be useful to inspect the assembly file associated with the out file, to see how well your code was optimized.
TI_OCL_DEBUG_KERNEL	The TI IDE and debugger, Code Composer Studio (CCS) is not required for running OpenCL applications on the 66AK2H, but if you do have CCS installed and and emulator connected, you can set this environment variable to enable assembly statement level debug of your kernel. When set, this environment variable will instruct the OpenCL runtime to pause before dispatch of a kernel. While paused the runtime will display data to the user indicating that a kernel dispatch is pending. It will instruct the user to connect to the board through an emulator and will display the appropriate break-point address to use for the start of the kernel code. Debug capability has not been a focus for this beta release and will definitely improve in later releases. Setting up the emulator and CCS is outside the scope of this Readme. If you do have those products, consult the documentation specific to those products.
TI_OCL_CACHE_KERNELS_OFF	This prevents the caching of OpenCL C Kernel compiles. This can be useful if your have OpenCL C kernels that call standard C from object code and you are actively modifying the standard C code. The standard C code is not seen by the caching algorithm and if you are actively modifying the C code, you may not pick up your modifications if a cached version of the OpenCL C kernel is being used.

8 Installation and Dependencies

This version of OpenCL is dependent on other packages for proper operation. If OpenCL was installed as part of the TI MCSDK-HPC product, it is likely that these dependencies are resolved already. However, they are listed here for your knowledge and as a first step in any troubleshooting.

8.1 CMEM module

This module allows for contiguous memory allocation of physical memory on the device.

After executing the Linux command `lsmod` you should see a cmemk module listed. If it is not listed, then it will need to be installed for proper OpenCL operation. Use the Linux modprobe command to install the cmem module. The actual parameters used in the command will differ depending on your DDR3 memory space partition. The following command and parameter set will likely get you started, however.

```
sudo modprobe cmemk phys_start=0x823000000 phys_end=0x880000000
     pools=1x1560281088 phys_start_1=0x0c040000 phys_end_1=0x0c500000
     allowOverlap=1
```

Additionally the /dev/cmem device should have appropriate permissions. Execute the command `ls -l /dev/cmem` and ensure that read and write permission for user, group and other are set. If not execute the command

```
sudo chmod 666 /dev/cmem
```

It is also important that a minimum of 1.5GB be reserved from Linux for proper cmem and OpenCL operation. Setting the uboot environment variable mem_reserve to at least 1536M will ensure this requirement.

8.2 MPM

The MPM package contains a client and server that allows the DSPs on the device to be loaded and run from Linux user space. To ensure that MPM is active, execute the Linux command `ps -ef | grep mpm` and your should see an instance of mpmsrv running. If your do not see that, then you can start the service by running:

```
/usr/sbin/mpmsrv.
```

If /usr/sbin/mpmsrv does not exist on your file-system refer to the MCSDK-HPC setup for installation.

8.3 MPM-TRANSPORT

The MPM-TRANSPORT package allows the A15 to read/write the shared memory from Linux user space. For proper operation, the devices `/dev/dsp*` need read/write permission for the user. To ensure that the permissions are correct, execute the Linux command `ls -l /dev/dsp*` and ensure that read/write permissions are set for user, group and other. If the permissions are not correct, you can set them with the command:

```
sudo chmod 666 /dev/dsp*
```

8.4 Required Environment Variables

The OpenCL package depends on a few environment variables for proper execution.

Note: Use EXPORT (bash) or SETENV (csh) to set these as environment variables.

TI_OCL_INSTALL	the location of the OpenCL package relative to the ARM A15
TI_OCL_CGT_INSTALL	the location of the ARM hosted C66 DSP compiler tools
LD_LIBRARY_PATH	`$TI_OCL_INSTALL/lib`
PATH	`$TI_OCL_INSTALL/bin/arm;$TI_OCL_CGT_INSTALL/bin`

Additionally if you are cross compiling OpenCL applications for the ARM target from an x86 Ubuntu machine, on that machine you will also need:

TI_OCL_INSTALL	location of the OpenCL package relative to the X86
TI_OCL_CGT_INSTALL	location of the x86 hosted C66 DSP compiler tools
PATH	`$TI_OCL_INSTALL/bin/x86;$TI_OCL_CGT_INSTALL/bin`
TARGET_ROOTDIR	location of Hawking EVM file system mount point on the x86 system

For platforms were native builds are possible, the cross compiling setup is not required, but is still supported for faster compilation.

The script hawking_env.sh is included in the examples directory of this installation. It illustrates the definition of these environment variables for both the ARM runtime side and a potential X86 cross compiling side.

8.5 Additional package dependencies

There are a few other packages that OpenCL depends on. These can be installed using apt-get on the machine on which you will compile (either cross or native) the OpenCL application.

```
sudo apt-get install binutils-dev mesa-common-dev libboost1.46-dev
                     libsqlite3-dev libffi6 zlib1g
```

For platforms that support Simple DirectMedia Layer the package libsdl1.2-dev can also be installed in order to see the mandelbrot display for the mandelbrot example. See section 6.4 for information on the mandelbrot example.

```
sudo apt-get install libsdl1.2-dev
```

Additionally, this OpenCL implementation requires the TI C6000 Compiler product for proper execution. IT can be obtained from `https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/hpcc6xcgt.htm`.