Tutorial: OpenCL™ Interoperability with Video Acceleration API on Linux* OS

OpenCL™ Code Builder
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>2</td>
</tr>
<tr>
<td>Legal Information</td>
<td>3</td>
</tr>
<tr>
<td>Introduction</td>
<td>4</td>
</tr>
<tr>
<td>Key Takeaways</td>
<td>4</td>
</tr>
<tr>
<td>Check VA Sharing Capabilities and Get OpenCL™ Functions</td>
<td>4</td>
</tr>
<tr>
<td>Creating VA Sharing OpenCL Context</td>
<td>5</td>
</tr>
<tr>
<td>Creating VA Sharing OpenCL Image</td>
<td>7</td>
</tr>
<tr>
<td>Using VA Sharing OpenCL Image</td>
<td>8</td>
</tr>
<tr>
<td>References</td>
<td>10</td>
</tr>
</tbody>
</table>
Legal Information

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to:

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Intel, Intel logo, Intel Core, VTune, Xeon are trademarks of Intel Corporation in the U.S. and other countries.

* Other names and brands may be claimed as the property of others.

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission from Khronos.

Microsoft product screen shot(s) reprinted with permission from Microsoft Corporation.

Copyright © 2015 Intel Corporation. All rights reserved.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
**Introduction**

Video Acceleration API (VA API) is an open source cross-platform API. VA API leverages acceleration of video encoding and decoding when hardware support is available. When a particular accelerator is a GPU, video surfaces used by VA API are also located on the GPU side. In turn, OpenCL™ software technology enables general processing of this data on the GPU through OpenCL API. To share the data between VA API and OpenCL API efficiently, Intel provides the `cl_intel_va_api_media_sharing` OpenCL extension. The goal of this extension is to provide interoperability between OpenCL and VA API, by sharing VA API surfaces with Intel’s implementation of the OpenCL standard without extra copy operations. The extension uses mechanisms and provides surface sharing capabilities for VA API that are similar to the ones covered by the OpenCL 1.2 Media Surface Sharing Khronos* extensions for DirectX* 9/DirectX 10/DirectX 11. To get access to the extension, install the [Media Server Studio](https://software.intel.com/en-us/mss).

**Key Takeaways**

The `cl_intel_va_api_media_sharing` extension enables sharing VA surface data with the OpenCL runtime using images acquired directly from VA surface planes. As a result of such sharing, OpenCL kernels become able to process VA surfaces without extra coping to OpenCL images. If the VA API Media Sharing extension support is available (see next section for details), you need to make the following modifications to leverage the efficient sharing flow in the OpenCL code (all the steps are detailed in the rest of document):

1. Get pointers to the following extension functions:
   a. `clGetDeviceIDsFromVA_APIMediaAdapterINTEL`
   b. `clCreateFromVA_APIMediaSurfaceINTEL`
   c. `clEnqueueAcquireVA_APIMediaSurfacesINTEL`
   d. `clEnqueueReleaseVA_APIMediaSurfacesINTEL`
2. Instead of `clGetDeviceIDs`, use the `clGetDeviceIDsFromVA_APIMediaAdapterINTEL` function to get devices corresponding to the VA adapter.
3. Pass VA adapter handle into `clCreateContext` through the `properties` argument.
4. Acquire the shared VA surfaces prior to OpenCL kernel execution and release the surfaces after the kernel completes the execution.

**Check VA Sharing Capabilities and Get OpenCL™ Functions**

Assuming that that the OpenCL platform is already selected (see "Platform/Device Capabilities Viewer" sample for an example of how to select an OpenCL platform), this tutorial also assumes that the OpenCL platform supports the VA Media Sharing extension. First step to check that the extension is supported - is to use the `clGetPlatformInfo` function.

```c
size_t len = 0; // length for extension string

// get size for string with extension names
clGetPlatformInfo(
    clPlatform, // the OpenCL platform to check
    CL_PLATFORM_EXTENSIONS, // the property name
    0, NULL,
    &len); // buffer length required for extension names

// allocate buffer to store extensions names
char* str = (char*)malloc(len);

//get string with extension names
clGetPlatformInfo(
```
Creating VA Sharing OpenCL Context

clPlatform,  // the OpenCL platform to check
CL_PLATFORM_EXTENSIONS,  // the platform property name
len,  // allocated buffer length
str,  // allocated buffer to store extension names
NULL);

The str is a char buffer that is filled by a null-terminated string that contains a list of space-separated extension names. Below is a sample content of str:

"cl_intel_accelerator cl_intel_advanced_motion_estimation cl_intel_motion_estimation cl_intel_subgroups cl_intel_va_api_media_sharing cl_khr_3d_image_writes cl_khr_byte_addressable_store cl_khr_global_int32_base_atomics cl_khr_global_int32_extended_atomics cl_khr_icd cl_khr_local_int32_base_atomics cl_khr_local_int32_extended_atomics cl_khr_spir".

If the extensions string has cl_intel_va_api_media_sharing entry, then the platform supports the extension and you can use this platform for media sharing.

Next step is to get media sharing function pointers. The following functions are required to use the media sharing:

- clGetDeviceIDsFromVA_APIMediaAdapterINTEL enables you to get OpenCL devices corresponding to the VA adapter.
- clCreateFromVA_APIMediaSurfaceINTEL creates an OpenCL 2D image object from a VA media surface. The new image object shares the same data as the media surface and is valid as long as the corresponding VA surface exists.
- clEnqueueAcquireVA_APIMediaSurfacesINTEL enables you to acquire OpenCL memory objects so that consecutive OpenCL commands, queued to a command-queue, can use the acquired memory objects.
- clEnqueueReleaseVA_APIMediaSurfacesINTEL releases the acquired OpenCL memory objects, returning ownership to VA API.

The pointer to each function can be obtained by the clGetExtensionFunctionAddressForPlatform OpenCL call. The functions are described in the extension specification and declared in CL/va_ext.h header provided with the Intel® Media Server Studio. The example below illustrates how to get pointer for the clGetDeviceIDsFromVA_APIMediaAdapterINTEL function only

```c
#include <CL/va_ext.h>

// define variables for the extension function pointers
clGetDeviceIDsFromVA_APIMediaAdapterINTEL_fn
clGetDeviceIDsFromVA_APIMediaAdapterINTEL = NULL;

//get pointer to the function and store it in the variable
clGetDeviceIDsFromVA_APIMediaAdapterINTEL = (clGetDeviceIDsFromVA_APIMediaAdapterINTEL_fn)clGetExtensionFunctionAddressForPlatform(
    clPlatform,  // the chosen platform
    "clGetDeviceIDsFromVA_APIMediaAdapterINTEL");  // the name of function

//check that returned pointer is not NULL
if(NULL == clGetDeviceIDsFromVA_APIMediaAdapterINTEL)
{
    printf("Can't get pointer to clGetDeviceIDsFromVA_APIMediaAdapterINTEL");
    return EXIT_FAILURE;
}
```

For convenience purposes consider keeping the pointer name equal to the function name.

You can query the rest of the function pointers similarly.

Creating VA Sharing OpenCL Context

Once all media sharing functions are available through pointers, you can create the OpenCL context using the device that corresponds to the VA adapter. To enumerate such devices, use the clGetDeviceIDsFromVA_APIMediaAdapterINTEL function. You can use this function similarly to an ordinary OpenCL clGetDeviceIDs routine except the fact that a VA adapter and additional VA-
related flags are used as input parameters. The code below uses
"CL_PREFERRED_DEVICES_FOR_VA_API_INTEL" flag to get a list of devices that is preferred for media
sharing.

```c
cl_int err = CL_SUCCESS; //error code
cl_device_id* pDevices = NULL; //buffer for device ids
cl_uint nDevices = 0; //number of devices

// get number of devices corresponding to VA adapter
err = clGetDeviceIDsFromVA_APIMediaAdapterINTEL(
    clPlatform,    // the selected OpenCL platform
    CL_VA_API_DISPLAY_INTEL,   // media adapter type
    vaDisplay, // media adapter handle
    CL_PREFERRED_DEVICES_FOR_VA_API_INTEL, // set of OpenCL devices
    NULL,NULL,
    &nDevices); // pointer to variable to store number of devices

if(CL_SUCCESS != err || 0 == nDevices)
{
    printf("Can’t get OpenCL device for media sharing");
    return EXIT_FAILURE;
}

// allocate buffer for device ids
pDevices = (cl_device_id*)malloc(sizeof(cl_device_id) * nDevices);

// get device ids corresponding to VA adapter
err = clGetDeviceIDsFromVA_APIMediaAdapterINTEL(
    clPlatform,    // the chosen platform
    CL_VA_API_DISPLAY_INTEL,   // media adapter type
    vaDisplay, // media adapter
    CL_PREFERRED_DEVICES_FOR_VA_API_INTEL, // set of opencl devices
    nDevices,    // size of output pDevices buffer
    pDevices, // pointer to output buffer
    NULL);

//check error here …
```

After calling the function, the `pDevices` buffer contains OpenCL device IDs that correspond to a given
`vaDisplay` adapter. If there is one or more devices corresponding to VA adapter, then you can create
the OpenCL context for media sharing. Create the context in a common way via the `clCreateContext`
function, but pass an additional property (for example, `CL_CONTEXT_VA_API_DISPLAY_INTEL`). The
example below creates context for all devices returned by
`clGetDeviceIDsFromVA_APIMediaAdapterINTEL`.

```c
cl_int err; // error code
cl_context clContext; // context object

// the properties list that is used to create context
cl_context_properties props[] = {
    // VA adapter
    CL_CONTEXT_VA_API_DISPLAY_INTEL, (cl_context_properties) vaDisplay,
    // declare that user does not give guarantee that any VA work is finished
    // before clEnqueueAcquireVA_APIMediaSurfacesINTEL will be called
    CL_CONTEXT_INTEROP_USER_SYNC, CL_FALSE,
    0);

//create context
clContext = clCreateContext(
    props, // properties with VA adapter
    nDevices, // number of devices used for context creation
    pDevices, // buffer with device IDs
    NULL, NULL, // callback function is not used here
    &err);   // returned error
```
// check error

The properties props is a list of pairs of property names and their corresponding values. The property list in this example has two components:

- A VA adapter vaDisplay that you should pass into clCreateContext to create a media sharing-enabled context.
- The CL_CONTEXT_INTEROP_USER_SYNC parameter. If this parameter is CL_FALSE then clEnqueueAcquireVA_APIMediaSurfacesINTEL guarantees that all VA calls finish before clEnqueueAcquireVA_APIMediaSurfacesINTEL finishes. If the parameter is CL_TRUE then you have to make sure that all VA calls finish before clEnqueueAcquireVA_APIMediaSurfacesINTEL.

Creating VA Sharing OpenCL Image

At this stage you can create cl_mem (image) object from VA surface via clCreateFromVA_APIMediaSurfaceINTEL function for the shared OpenCL context.

This function accepts an ID of the VA surface that is used as a base for the corresponding OpenCL image. The code below shows how to create four images for Y and UV planes for input and output VA_FOURCC_NV12 VA surfaces. NV12 surface has two planes. The first one is a full-sized (width,height) one-channel Y plane with an intensity component, the second one is half-sized (width/2,height/2) two-channel UV plane with color components.

// create input 2D images to share Y and UV planes for VA input surface
clImageInY = clCreateFromVA_APIMediaSurfaceINTEL(
    clContext,   // context created for media sharing
    CL_MEM_READ_ONLY,  // create 2D image for reading (by OpenCL kernels)
    vaSurfaceIn,  // input VA surface for sharing
    0,   // Create 2D image for Y plane for NV12 format
    &err);   // error code
// check error here...

clImageInUV = clCreateFromVA_APIMediaSurfaceINTEL(
    clContext,   // context created for media sharing
    CL_MEM_READ_ONLY,  // create 2D image for reading (by OpenCL kernels)
    vaSurfaceIn,  // input VA surface for sharing
    1,   // Create 2D image for UV plane for NV12 format
    &err);   // error code
// check err here

// create output 2D images to share Y and UV planes for VA output surface
clImageOutY = clCreateFromVA_APIMediaSurfaceINTEL(
    clContext,   // context created for media sharing
    CL_MEM_WRITE_ONLY, // create 2D image for writing (by OpenCL kernels)
    vaSurfaceOut,  // output VA surface for sharing
    0,   // Create 2D image for Y plane for NV12 format
    &err);   // error code
// check error here...

clImageOutUV = clCreateFromVA_APIMediaSurfaceINTEL(
    clContext,   // context created for media sharing
    CL_MEM_WRITE_ONLY, // create 2D image for writing (by OpenCL kernels)
    vaSurfaceOut,  // output VA surface for sharing
    1,   // Create 2D image for UV plane for NV12 format
    &err);   // error code
// check err here

Here vaSurfaceIn and vaSurfaceOut are input and output NV12 VA surfaces to be shared as OpenCL 2D image objects. The clImageInY, clImageInUV and clImageOutY, clImageOutUV are new OpenCL image objects that are based on the vaSurfaceIn and vaSurfaceOut.

1. clImageInY shares Y plane of input VA surface
2. clImageInUV shares UV plane of input VA surface
3. clImageOutY shares Y plane of output VA surface

4. clImageOutUV shares UV plane of output VA surface
4. **clImageOutUV** shares UV plane of output VA surface

The **CL_MEM_READ_ONLY** flag is used for input images as only read operations are made for these images in the OpenCL kernels. To create output images for writing use the **CL_MEM_WRITE_ONLY** flag. To create image for both reading and writing use the **CL_MEM_READ_WRITE** flag.

### Using VA Sharing OpenCL Image

Upon creating OpenCL memory objects from VA surfaces, you can use any regular OpenCL kernel that operates on images to process the VA surfaces. Note that NV12 surface has Y plane with (width, height) sizes and UV half-sized plane (width/2, height/2). So UV plane has one color pair for four neighboring pixels. Consequently, an OpenCL kernel that processes at least a 2x2 pixel block is more suitable for NV12 format and the created images.

For example, consider the following kernel:

```c
#define SCALE_COLOR 0.75f
#define SCALE_INTENSITY 1.5f
const sampler_t smp = CLK_FILTER_NEAREST;

__kernel void ProcessYUV(
    __read_only image2d_t srcY, //input Y plane
    __read_only image2d_t srcUV, //input UV plane
    __write_only image2d_t dstY, //output Y plane
    __write_only image2d_t dstUV) //output UV plane
{
    const int x = get_global_id(0);
    const int y = get_global_id(1);
    // single coordinate for single UV value
    const int2 coordUV = (int2)(x,y);
    // four coordinates for four Y values
    const int2 coordY[4] = {
        (int2)(2*x,2*y),
        (int2)(2*x+1,2*y),
        (int2)(2*x,2*y+1),
        (int2)(2*x+1,2*y+1)
    };

    float4 UV, Y[4];
    // read UV plane component
    UV = read_imagef(srcUV, smp, coordUV);
    // read 4 intensity components from input Y plane
```
```c
Y[0] = read_imagef(srcY, smp, coordY[0]); // read Y plane 1 component
Y[1] = read_imagef(srcY, smp, coordY[1]); // read Y plane 2 component
Y[2] = read_imagef(srcY, smp, coordY[2]); // read Y plane 3 component
Y[3] = read_imagef(srcY, smp, coordY[3]); // read Y plane 4 component

// process single color component
UV.xy = (UV.xy-0.5f)*SCALE_COLOR+0.5f;

// process four intensity components
Y[0].x = (Y[0].x-0.5f)*SCALE_INTENSITY+0.5f;
Y[1].x = (Y[1].x-0.5f)*SCALE_INTENSITY+0.5f;
Y[2].x = (Y[2].x-0.5f)*SCALE_INTENSITY+0.5f;
Y[3].x = (Y[3].x-0.5f)*SCALE_INTENSITY+0.5f;

// write new color component into output UV plane
write_imagef(dstUV, coordUV, UV);

// write 4 new intensity components into output Y plane
write_imagef(dstY, coordY[0], Y[0]);
write_imagef(dstY, coordY[1], Y[1]);
write_imagef(dstY, coordY[2], Y[2]);
write_imagef(dstY, coordY[3], Y[3]);
}
```

This kernel makes simple color and intensity corrections. The result of such processing is shown in the following pictures. The left one is an input image and the right one is an output image.

Before enqueueing this kernel with the regular `clEnqueueNDRangeKernel` function, call the `clEnqueueAcquireVA_APIMediaSurfacesINTEL` function to schedule acquiring of the VA surfaces. Then you can submit the kernel into the queue for execution. You can submit several kernels that process the acquired objects. Once you submit all the necessary kernels, call `clEnqueueReleaseVA_APIMediaSurfacesINTEL` to release the acquired VA surfaces. The code example below shows how to submit the kernel for execution to process OpenCL images that share VA surfaces.

```c
// the processed shared VA surfaces has to be acquired
// before submit OpenCL kernel to process corresponding images.
// place all image objects into single buffer to acquire all of them
cl_mem images[4] = {
clImageInY, // input image with Y plane of VA surface
clImageInUV, // input image with UV plane of VA surface
clImageOutY, // output image with Y plane of VA surface
clImageOutUV // output image with UV plane of VA surface
};

// no need for explicit sync with VA before AcquireVA
// as the context was created with CL_CONTEXT_INTEROP_USER_SYNC==CL_FALSE

clEnqueueAcquireVA_APIMediaSurfacesINTEL(
```
clQueue,     // queue for media sharing device
4,          // number of images in "images" buffer
images,     // buffer with images to be processed
0,  NULL, NULL); // no any events

// set arguments for kernel that
// will process acquired VA surface through images
clSetKernelArg(0,sizeof(cl_mem),&clImageInY);
clSetKernelArg(1,sizeof(cl_mem),&clImageInUV);
clSetKernelArg(2,sizeof(cl_mem),&clImageOutY);
clSetKernelArg(3,sizeof(cl_mem),&clImageOutUV);

// Enqueue kernel
// setup global size to execute image processing kernel
// NV12 has Y plane with (width,heigh) size
// and half sized UV plane. It is convenient to process
// 2x2 pixel block (with the same chroma) by one workitem.
// So the used global size is with/2 by height/2
size_t GS[2] = {width/2,height/2};
clEnqueueNDRangeKernel(
    clQueue,     // queue for media sharing device
    clKernel,    // OpenCL kernel
    2,          // GS has 2 elements
    NULL,       // no offset
    GS,         // global sizes for processing
    NULL,       // use default local size
    0, NULL, NULL); //no events

// release images that were acquired for OpenCL kernel processing
clEnqueueReleaseVA_APIMediaSurfacesINTEL(
    clQueue,     // queue for media sharing device
    4,          // number of images in "images" buffer
    images,     // buffer with processed images
    0,  NULL, NULL); // no any events

References
1. Video Acceleration API
2. OpenCL™ Platform/Device Capabilities Viewer Sample
3. cl_intel_dx9_media_sharing OpenCL™ platform extension
4. Intel® Media Server Studio 2015
5. cl_intel_va_api_media_sharing_extension OpenCL™ platform extension