Vulkan Synchronization2 Validation

Jeremy Gebben, LunarG
March 2021

Introduction

The newly released VK_KHR_synchronization2 extension brings extensive improvements to Vulkan queue submission, events, and pipeline barriers resulting in significant API usability enhancements for developers.

Synchronization2 highlights include:

- Data for semaphores and command buffers is passed in arrays of structures, rather than in separate arrays spread across multiple structures, to streamline queue submissions.
- Barrier pipeline stage masks are now stored in the barrier structure rather than passed as separate parameters to vkCmdPipelineBarrier() to simplify resource state tracking.
- VkPipelineStageFlags2KHR and VkAccessFlags2KHR are expanded to 64-bits to allow for future extensibility with new extensions.
- vkCmdSetEvent2KHR() requires pipeline barriers, enhancing driver efficiency by scheduling work at event 'set' time, rather than the 'wait' for barrier information to become available.
- New image layout types now 'do the right thing' for both color and depth/stencil images. Also, image layout transitions do not happen if a barrier's oldLayout and newLayout field are equal, even if the layout provided doesn't match the current layout of the image.

More details on this significant extension have been added to the Vulkan Guide.

The Vulkan SDK version 1.2.170.0 adds immediate support for Synchronization2, and here we highlight how developers can leverage the SDK to immediately use this new functionality, even before the extension ships in their Vulkan drivers.

Upgrading to Synchronization2

The Synchronization2 extension has been designed to be largely backward compatible with the previous Synchronization extension, enabling new Synchronization2 functions to be adopted incrementally into an existing application. For example, Synchronization2 queue submissions can be used with the original pipeline barriers or vice versa.

The one exception is that setting events and matching waiting for events must use calls from the same version of the extension, i.e., SetEvent must be used with WaitEvents, and SetEvent2 must be used with WaitEvents2.

There are code examples in the Vulkan Guide that show how to convert code to use the new VK_KHR_synchronization2 extension, together with Synchronization2 example code in the Vulkan Wiki and new Synchronization2 samples in the Vulkan-Samples repository/.

Synchronization2 Layer in Vulkan SDK

In order to promote rapid adoption of the new Synchronization2 extension and provide a smooth transition for developers, a new software layer is now available with the Khronos Vulkan SDK which efficiently implements Synchronization2 over the original synchronization APIs. This enables applications to use the new extension even if target devices do not have Synchronization2 support in their native drivers.

This new layer, called VK_LAYER_KHRONOS_synchronization2, is designed to be a transparent helper by intercepting calls to vkGetPhysicalDeviceFeatures2() and reporting support for the extension if the underlying device says it doesn't support it. Then if vkCreateDevice() is called with Synchronization2 enabled and the device doesn't implement it, the layer will intercept the new Synchronization2 functions, translating new extension calls into equivalent calls to original functions. The layer also intercepts many of the original synchronization functions to translate VK_PIPELINE_STAGE_NONE_KHR and the new image layouts (VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL_KHR and VK_IMAGE_LAYOUT_READ_ONLY_OPTIMAL_KHR) into values that an older driver implementation would support.

Synchronization2 Improvements

There are extensive improvements to queue submission, events, and pipeline barriers, which are highlighted below.

Queue Submission

vkQueueSubmit2KHR() simplifies queue submission by letting you create a single structure for all the data needed for each command buffer and semaphore, rather than having to manage multiple arrays of related values. This is best exemplified by the VkSemaphoreSubmitInfoKHR structure, which bundles together the semaphore handle, stage mask, timestamp value, and device group. Previously, these fields were spread across four different arrays in three different structures. Going forward, if an extension adds additional semaphore-related data to queue submission, the data could be added as an extension to VkSemaphoreSubmitInfoKHR, rather than having to add another array of data at the top level VkSubmitInfo2KHR.

Events

Events are substantially updated in this extension. vkCmdSetEvent2KHR() now requires you to provide all of the barriers that are associated with the event. Previously this data wasn't required until vkCmdWaitEvents() was called, which was problematic for drivers. Without the barrier information, it was hard for drivers to start sending work for the event to the GPU when vkCmdSetEvent() was called. Now that the barrier information is available, drivers should be able to provide a better event implementation. There's also a new flag, VK_EVENT_DEVICE_ONLY_BIT_KHR in VkEventCreateInfo, which you can set if you do not need to call vkSetEvent(), vkResetEvent() or vkGetEventStatus() with the event you are creating. This hint gives drivers the opportunity to further optimize their event implementation by not worrying about synchronizing the GPU with these host operations.

Pipeline Barriers

From a developer's perspective, it now makes sense to think of vkCmdSetEvent2KHR() as the first half of a vkCmdPipelineBarrier2KHR() call, and vkCmdWaitEvents2KHR() as the second half. This is further reinforced by all of these functions taking VkDependencyInfoKHR as a parameter containing all of the barrier information. This structure replaces the seven parameters required by vkCmdPipelineBarrier() and usually allows for more concise code. Additionally, pipeline stage masks are now part of each barrier's structure. This makes it possible to track all of the usage state for a resource directly in the barrier structure without worrying about constructing a global pipeline stage mask for all of your barriers.

There is an important change in this extension to pipeline stages and access masks. The original VkPipelineStageFlags and VkAccessFlags fields are 32-bits wide, and there are no longer any free bits for new extensions to use. The new types, VkPipelineStageFlags2KHR and VkAcessFlags2KHR are 64-bits wide, to allow more stages and access fields to be defined. All existing bit definitions from the original fields have the same definitions in the new fields, so it is easy to convert from old to new. Several existing pipeline stages and access masks have been expanded using bits in the 'upper' parts of the new fields. But using these new bits is optional, as the original bits are still supported. However, at some point, a new extension will most likely require an 'upper' bit that is only available in the new fields. Then you will be required to use Synchronization2 to be able to use all of the functionality in the new extension.

Additionally, 64-bit enumeration types are not available in all C/C++ compilers, so the code for the new fields uses 'static const' values instead of an enum. As a result of this, there are no equivalent types to VkPipelineStageFlagBits and VkAccessFlagBits. Some code, including API functions such as vkCmdWriteTimestamp(), used the 'Bits' type to indicate that the caller could only pass in a single bit value, rather than a mask of multiple bits. For Synchronization2, these calls need to be converted to take the "Flags" type and enforce the "only 1-bit" limitation via Valid Usage or the appropriate coding convent for your own code, as was done for vkCmdWriteTimestamp2KHR(). This part of the extension has often caused problems for code generators, so if you are generating code you may need to spend some time on this.

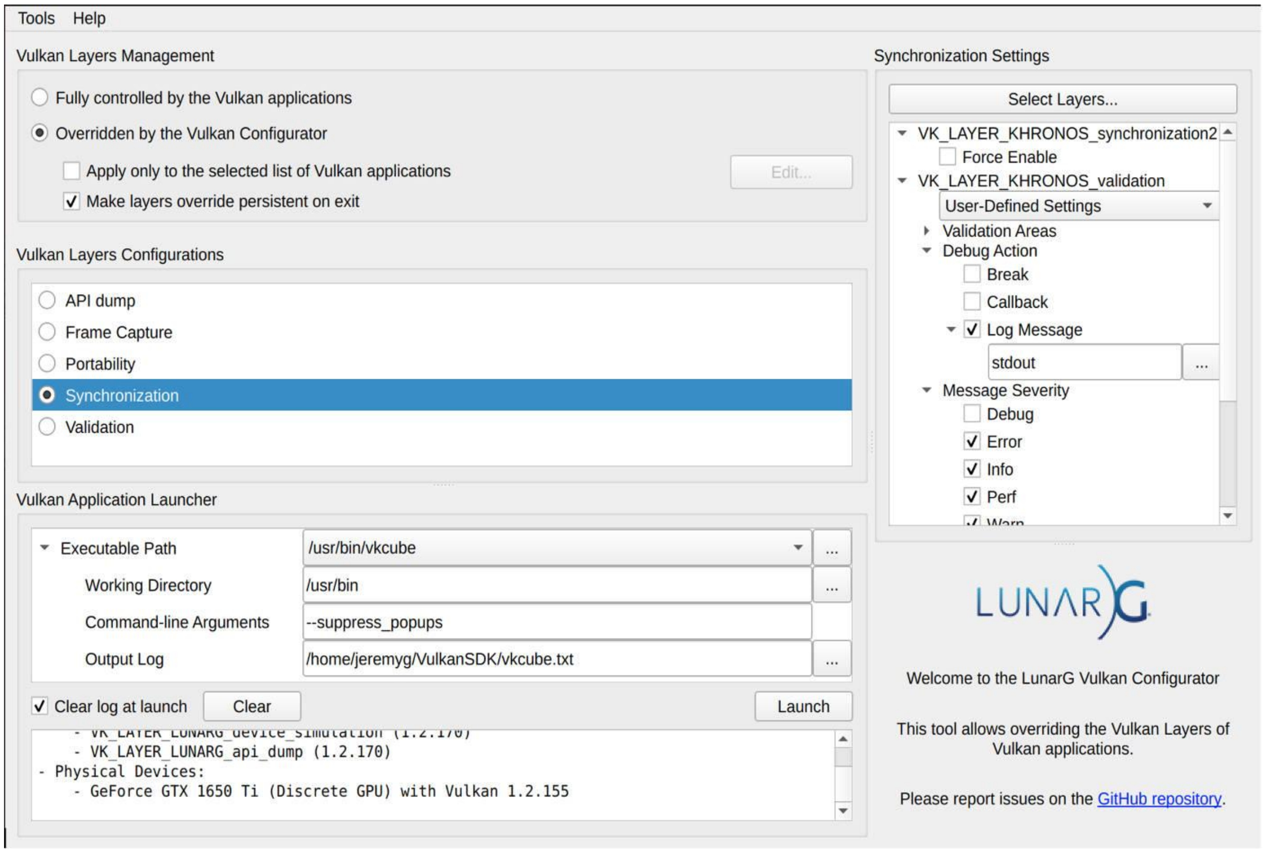
Additional Information

For more detailed and complete information about this extension, please see:

- The extension description in the Vulkan specification.
- The updated Vulkan Synchronization Examples wiki.
- Code conversion examples in the Vulkan Guide
- The new Synchronization2 sample in the Vulkan-Samples repository

Getting Started with Synchronization2 in the Vulkan SDK

The Vulkan SDK includes functionality to help you get started with this new extension. First of all, the Vulkan Configurator (vkconfig) has a new "Synchronization" configuration, which enables Synchronization Validation, and VK_LAYER_KHRONOS_synchronization2.



Vulkan Layers Management

- ( ) Fully controlled by the Vulkan applications
- (●) Overridden by the Vulkan Configurator
  - [ ] Apply only to the selected list of Vulkan applications
  - [✓] Make layers override persistent on exit

Vulkan Layers Configurations

- ( ) API dump
- ( ) Frame Capture
- ( ) Portability
- (●) Synchronization
- ( ) Validation

Vulkan Application Launcher

Executable Path	/usr/bin/vkcube
Working Directory	/usr/bin
Command-line Arguments	--suppress_popups
Output Log	/home/jeremyg/VulkanSDK/vkcube.txt

[✓] Clear log at launch

```
    - VK_LAYER_LUNARG_device_simulation (1.2.170)
    - VK_LAYER_LUNARG_api_dump (1.2.170)
- Physical Devices:
    - GeForce GTX 1650 Ti (Discrete GPU) with Vulkan 1.2.155
```

Synchronization Settings

- VK_LAYER_KHRONOS_synchronization2
  - [ ] Force Enable
- VK_LAYER_KHRONOS_validation
  - User-Defined Settings
  - Validation Areas
  - Debug Action
    - [ ] Break
    - [ ] Callback
    - [✓] Log Message
      - stdout
  - Message Severity
    - [ ] Debug
    - [✓] Error
    - [✓] Info
    - [✓] Perf
    - [✓] Warn

Welcome to the LunarG Vulkan Configurator

This tool allows overriding the Vulkan Layers of Vulkan applications.

Please report issues on the GitHub repository.

Enabling the Synchronization2 layer in the Vulkan Configurator

To always use the layer's Synchronization2 implementation, the "Force Enable" setting causes it to be active even when the underlying device also implements the extension. For more information, please see the Synchronization2 documentation for this layer in the SDK and the source code in the Vulkan-ExtensionLayer repo.

Improved Synchronization Validation

Synchronization Validation was first released in 2020 and has been substantially evolved to support Synchronization2. Even when the Synchronization2 extension isn't in use, Validation uses the new pipeline stages and access masks, as well as tracking pipeline stages on a per barrier basis, enabling improved error messages. For example, because Synchronization2 has separate pipeline stages for the different types of transfer operations, Synchronization Validation can report a hazard from a vkCmdCopyBuffer() call with SYNC_COPY_TRANSFER_READ in the access info, rather than SYNC_TRANSFER_TRANSFER_READ.

Core Validation also adds support for Synchronization2 and takes advantage of the extension's backward compatibility.

Where to Get Help

Synchronization2 brings big changes to Vulkan to improve developer productivity and application performance. Updates to the Vulkan SDK and vkconfig will improve API usability for developers. The extensive improvements to queue submission, events, and pipeline barriers will allow for more efficient driver implementations and provide for future extensibility. The implementation available in the new Synchronization2 layer enables developers to immediately start development using this significant new extension.

If you have questions or find problems with the SDK, you should report issues on https://vulkan.lunarg.com/. To offer suggestions for future releases, provide comments, or report issues about the Synchronization2 layer or Synchronization Validation, please contact the development team via the GitHub repositories using the links shown below:

- Synchronization2 Layer: https://github.com/KhronosGroup/Vulkan-ExtensionLayer
- Vulkan Validation Layers: https://github.com/KhronosGroup/Vulkan-ValidationLayers
- Synchronization2 Layer Example: https://github.com/KhronosGroup/Vulkan-Samples/tree/master/samples/extensions/synchronization_2

Acknowledgments:

Tobias Hector, AMD
John Zulauf, LunarG
Spencer Fricke, Samsung
Baldur Karlsson, RenderDoc