

Simple DMA Decompression Sample

*\* This sample is compatible with the June 2017 Xbox One XDK*

# Description



# This sample demonstrates how to use the hardware LZ compression and decompression functionality in a couple different scenarios.

# The Xbox One GPU allows direct access to DMA engines that implement the DEFLATE algorithm (<http://www.rfc-base.org/rfc-1951.html>) as well as Jpeg Decode. These can be used to offload tasks that would otherwise be CPU intensive.

# Using the sample

## This sample uses the following controls:

|  |  |
| --- | --- |
| Action | Gamepad |
| Cycle through loaded textures | DPad Up/Down |
| Toggle hardware/software compression | A button |
| Toggle hardware/software decompression | B button |
| Exit | View Button |

# Implementation notes

## Scenario 1: Runtime Decode/Encode

When the sample loads, it will pick up any DDS texture files that exist in the Media\Textures\ folder of the sample. You can add additional files by modifying the MediaList.txt file and re-deploying.

Once loaded, you can change between compression with the Dma Hardware (LZ77 with a 1KB window) or using reference compression libraries Zlib or Zopfli. When using the latter software compression options, compression is limited to the most optimal options supported by the decompression engine, using a 4KB window. Switching between these options allows easy comparison of compression options available. Also note that that larger files are fragmented into chunks slightly under 4MB, to maintain compatibility with the hardware.

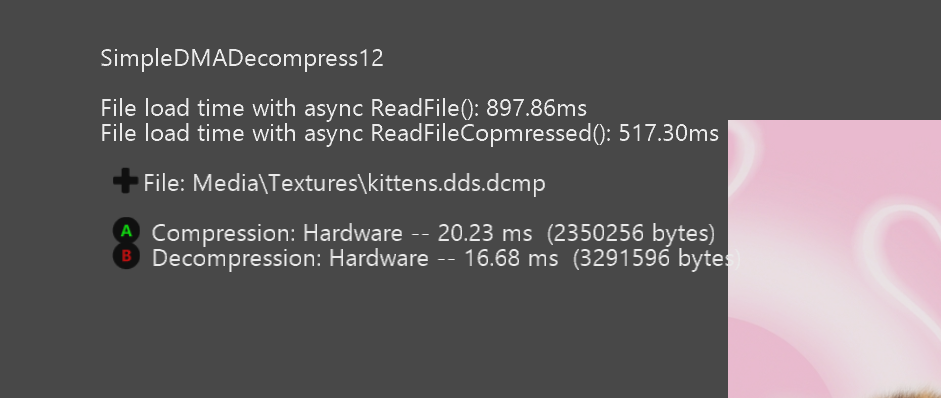
## Scenario 2: Streaming DMA Decompression

This scenario is like what most titles do, decompressing content that’s being streamed in from disk that was compressed as part of the content build pipeline. In this variant, all the decompression work is offloaded to the dedicated hardware instead of costing CPU cycles.

To accomplish this, the StreamingDmaCompressionLib project defines a simple file format consisting of compressed blocks that allow asynchronous offloading to the DMA hardware to increase effective I/O throughput and decrease the package size of the title. For ease of use, the XboxDmaCompression tool (PC-only) allows entire files to be compressed in the most optimal fashion that the hardware supports. Alternatively, the library can be used directly in order to compress at a stream level so as to embed compressed content into a larger composite file.

The resulting file (or embedded compressed stream) is read using ReadFileCompressed(), which is essentially API-compatible with the Win32 ReadFile() api. It’s intended for overlapped asynchronous I/O, and supports buffered and unbuffered reads, with the latter improving performance.

To demonstrate the throughput characteristics of this system, the default MediaList.txt replicates 20 compressed and 20 uncompressed copies of the included texture, and in the initialization\load path loads one batch at a time. Total uncompressed bytes in each batch are 59.6MB, and the effective throughputs of uncompressed load (66MB/s) and compressed load (115MB/s) are substantially different. Combined with the DmaCompressionTool, this is an easy way to determine if the DMA decompression hardware makes sense for your title. The library is fully asynchronous, and thus scales similarly to the underlying Windows overlapped\async I/O, improving with the number of requests in flight.



When using the ReadFileCompressed() api, the primary difference is that the size of the decompressed buffer is not known at the the time the API is invoked. Rather than requiring a title to know this information beforehand, since this would require some other blocking read, the API supports passing in custom allocator, or the default allocator can be used, and the allocation is done once the block containing the file header is loaded.

Other aspects of the overlapped I/O involving ReadFileCompressed are identical to ReadFile, with a couple exceptions. ReadFileCompressed can handle non-sector-aligned reads on unbuffered files, handling the pre\post buffering internally. ReadFileCompressed does, however, require that reads be 4 byte aligned, to ensure that both the dma move from the storage controller, and the dma move through the decompressor can be accomplished without any memory being copied by the CPU. If you use the StreamingDmaCompressionLib to compress a stream within a larger composite file, you must ensure that the first byte of the compressed stream is 4-byte aligned within the file.

# Known issues

* This sample does not properly implement suspend/resume.

# Update history

The original version of this sample was written in 2012. It was rewritten in September 2018.

# Privacy Statement

When compiling and running a sample, the file name of the sample executable will be sent to Microsoft to help track sample usage. To opt-out of this data collection, you can remove the block of code in Main.cpp labeled “Sample Usage Telemetry”.

For more information about Microsoft’s privacy policies in general, see the [Microsoft Privacy Statement](https://privacy.microsoft.com/en-us/privacystatement/).