This write-up details the step by step process to configure OpenCL on Visual Studio and start working with it quickly. It also provides a recent working example to test out our configuration. It, however, does not serve as a OpenCL beginner tutorial. For that, I suggest to look at this [excellent article on OpenCL at Dr. Dobb](http://www.drdobbs.com/parallel/a-gentle-introduction-to-opencl/231002854).

Getting Started

OpenCL has a lot of articles, tutorials, blog posts and StackOverflow questions providing a lot of information for beginners to get started. However, many of these tutorials are outdated, don’t fully resolve all issues or miss out some specifics. It took me two days of fiddling to finally have a working OpenCL tutorial in Visual Studio. Hence, this blog post is to help other people save their two days in configuration, aggregated from a lot of sources and only providing those details which work.

System configuration

I have a Intel Core i3 processor with NVIDIA GeForce 710M and 4 GB RAM running on Windows 10 64-bit. I also have Visual Studio 2012 where I will be configuring OpenCL SDK. Same guide can be used for AMD GPUs too with some variations like location of SDK folders.

1. Getting required drives and SDK

Two things are needed here. First, OpenCL runtime for your graphics card. It can be achieved by simply [updating NVIDIA’s graphics card](https://www.nvidia.in/Download/index.aspx). Secondly, OpenCL SDK is needed for compiling OpenCL code. NVIDIA has hidden them under its CUDA toolkit. So, [install CUDA toolkit](https://developer.nvidia.com/cuda-toolkit) and you will get OpenCL SDK too.

2. Setting up Visual Studio

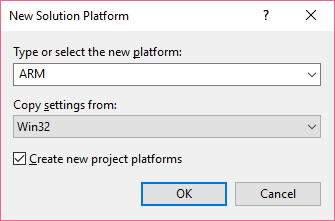
I have Visual Studio 2012, hence, the configurations are based on that only. Create a new Visual Studio C++ application (any template). Under src, create a new C file by name of *main.c*. Similarly, create a kernel file by the name of *kernel.cl*. *main.c*will contain the host code. *kernel.cl*will contain the kernel to be executed.

3. Project configurations

OpenCL is suggested to run on 64-bit configurations. However, the created solution would be in 32–bit only. To fix this, right click on the project > Choose Properties in the context menu. A *Property Pages*window will open.

Click on Configuration Manager on right. In the Configuration Manager window, select *<New..>*from Active solution platform dropdown menu. In the New Solution Platform window, choose x64 as new platform and copy settings option as Win32. This will make the project targeted for 64-bit build.

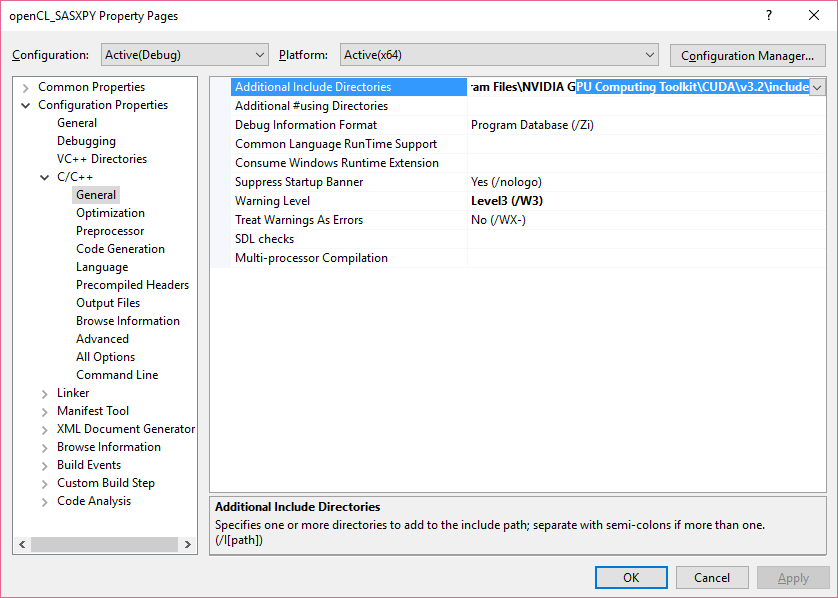
Image for post



Select x64 as new platform for 64-bit builds

For OpenCL config, go to C/C++ > General page. For *Additional Include Directories*, point to include folder inside your CUDA toolkit installation folder.

Image for post

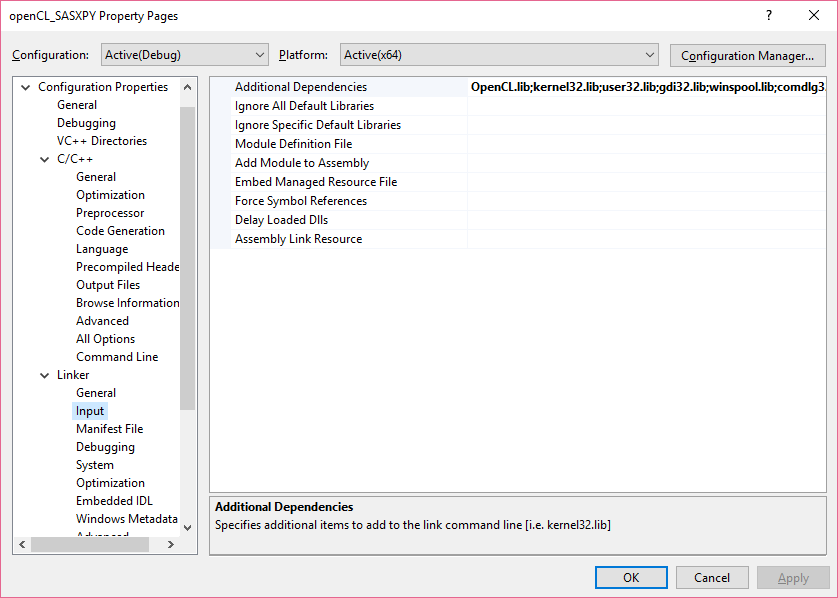


Property Page

Other tutorials on the internet suggest than instead of full path, one can also provide environmental variable *$(CUDA\_INC\_PATH)* here( It is automatically created when the toolkit is installed). I, however, found that on doing, Visual Studio’s IntelliSense and auto-complete features will fail to recognise OpenCL code, and mark whole codebase with errors(even though compilation would be successful). Hence, it is best to provide direct path to the include folder instead as done above.

Next in *Property Pages*, go to Linker > Input and add **OpenCL.lib** to Additional dependencies.

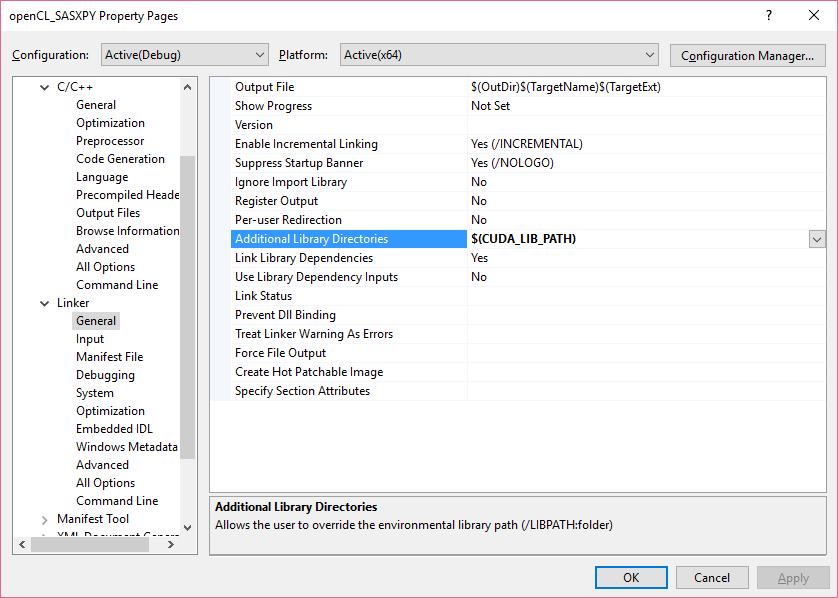
Image for post



Add OpenCL.lib to Additional dependencies

Lastly, in Linker > General option page, add environmental variable *$(CUDA\_LIB\_PATH)* for Additional Library Directories. This variable contains the path to the directory containing **OpenCL.dll**(It is also automatically created with CUDA toolkit installation).

Image for post



Setting path to OpenCL.dll

With this, we have completed configurations. Now, to code.

4. Adding code to the project

Since, this is not a OpenCL programming tutorial, hence, I would suggest you to copy main.c code from [here](https://github.com/pratikone/OpenCL-learning-VS2012/blob/master/main.cpp) and kernel.cl code from [here](https://github.com/pratikone/OpenCL-learning-VS2012/blob/master/vector_add_kernel.cl). Since, the original main.c picks up code from *vector\_add\_kernel.cl*but we have*kernel.cl*, hence change the filename in main.c

30: fp = fopen("vector\_add\_kernel.cl", "r"); replace it with30: fp = fopen("kernel.cl", "r");

5. Running the program

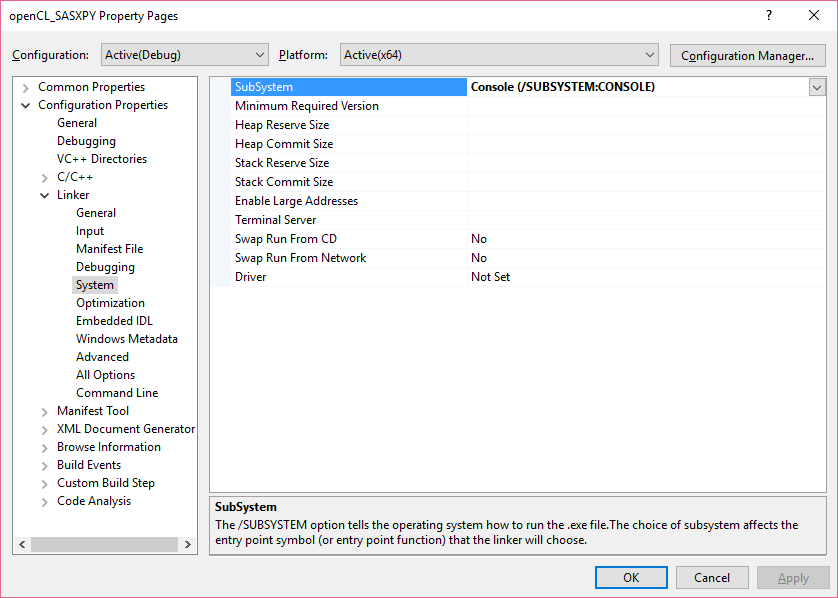
Press F5 to compile and run the program. If no compilation error, it will create a Debug build of the program and launch it.

*A quick reminder here that since it is VS2012, it only supports C89 formats. Hence, if your code is incompatible with C89, it will throw error. It has been fixed from VS2013 though. Many examples on the internet simply fail to compile because of this reason. Hence, this example has been selected for showcasing here because it is C89 compliant.*

A command prompt will open up quickly, print some messages and exit quickly.***If you want to prevent the command prompt window from closing down you can do the following.***

Open the Property pages of the project again. Goto Linker > System page. Change the SubSystem property to *Console (/SUBSYSTEM:CONSOLE)*.

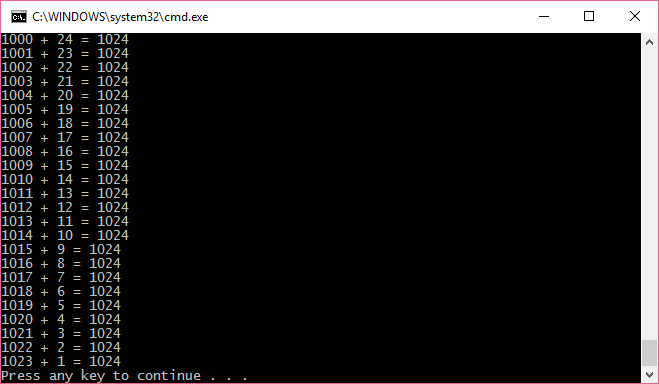
Image for post



Config to prevent command prompt from closing down automatically

Now, try to launch the program again as a *Release* build by pressing Ctrl + F5. The program will launch in command prompt but will not exit automatically and will wait for a key press to exit.

Image for post



Command Prompt waits for key press before exiting

6. Few thoughts over the code

This code has been ported from [here](http://www.eriksmistad.no/getting-started-with-opencl-and-gpu-computing/) with changes incorporated from [Dr. Doob’s article’s code](http://www.drdobbs.com/parallel/a-gentle-introduction-to-opencl/231002854?pgno=3) . The original code was not detecting my NVIDIA platform. Hence, I changed the API to fetch platform information differently.

*// Get platform and device information*  
 cl\_platform\_id platform\_id = NULL;  
 cl\_device\_id device\_id = NULL;   
 cl\_uint ret\_num\_devices;  
 cl\_uint ret\_num\_platforms;  
 cl\_int ret = clGetPlatformIDs(1, &platform\_id, &ret\_num\_platforms);  
 ret = clGetDeviceIDs( platform\_id, CL\_DEVICE\_TYPE\_GPU, 1,   
 &device\_id, &ret\_num\_devices);

This did not work and failed to find NVIDIA plaform. Setting CL\_DEVICE\_TYPE\_GPU to CL\_DEVICE\_TYPE\_DEFAULT or CL\_DEVICE\_TYPE\_ALL made the program choose Intel CPU which got stuck at building kernel, for reasons unknown.

Hence, I changed the code to the following which is more general form of finding OpenCL-compatible plaforms.

// Get platform and device informationcl\_device\_id device\_id = NULL;cl\_uint ret\_num\_devices;cl\_uint ret\_num\_platforms;cl\_int ret = clGetPlatformIDs(0, NULL, &ret\_num\_platforms);cl\_platform\_id \*platforms = NULL;platforms = (cl\_platform\_id\*)malloc(ret\_num\_platforms\*sizeof(cl\_platform\_id));ret = clGetPlatformIDs(ret\_num\_platforms, platforms, NULL);printf("ret at %d is %d\n", \_\_LINE\_\_, ret);ret = clGetDeviceIDs( platforms[1], CL\_DEVICE\_TYPE\_ALL, 1,&device\_id, &ret\_num\_devices);

*clGetPlatformIDs(0, NULL, &ret\_num\_platforms);* returns the number of platforms. This information is used to allocate memeory for *cl\_platform\_id \*platforms*. A second call to *clGetPlatformIDs(ret\_num\_platforms, platforms, NULL);* populates platforms with data.

One can print the values of platforms but I didn’t . platforms[0] is Intel CPU and platforms[1] is NVIDIA GPU. Hence, I have selected it for *clGetDeviceIDs*call.

Conclusion

If you have made it this far, then congratulations. I have tried to provide a simple configuration guide for setting up OpenCL on Visual Studio. Though the example uses NVIDIA GPUs, same instructions should apply to AMD GPUs as well. OpenCL documentation and support is really fragmented and it can be daunting task to figure out its nuances. I hope this post can help a person from hair pulling kind of experience I went through , while trying to learn OpenCL. Comments for improvement are always welcome.

*Some really good resources for OpenCL whose ideas this post uses :*

[*Streamcomputing blog*](https://streamcomputing.eu/blog/2015-03-16/how-to-install-opencl-on-windows/)*: Probably the best resource for OpenCL related stuff on the internet*

[*Dr Dobb Gentle introduction to OpenCL*](http://www.drdobbs.com/parallel/a-gentle-introduction-to-opencl/231002854)*: Best beginner’s guide to OpenCL concepts*

[*Erik Smistad ‘s OpenCL hello world program*](http://www.eriksmistad.no/getting-started-with-opencl-and-gpu-computing/)*: The base example used in this post*

[*Introduction to OpenCL slides*](http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/06-intro_to_opencl.pdf)*: Most concise way to learn OpenCL concepts quickly*