MSYS2 + vs code => C++ DSP Dev

Objective

Install an environment for building native programs that use the GCC compilers, such `g++` for C++ programs and the `gdb` debugger. Secondly, you install a popular yet simple code editor, for writing, debugging, and running your code. For GCC there are multiple options to choose from, but for Windows one such environment is *MSYS2* (https://www.msys2.org/) with Mingw-w64 (https://www.mingw-w64.org). A cross-platform IDE that works well under macOS, Linux, and clearly Windows, is Microsoft's *vs code* configured for code development Mingw-w64 (https://code.visualstudio.com/docs/cpp/config-mingw).

With all of the above in place you have a comfortable environment to develop and benchmark DSP code using C++. The code executable can be fed binary file test vectors from Jupyter Lab notebooks and in turn output test vectors from the C++ code can be further analyzed and display results in the same Jupyter Lab notebook. Note Julia with the Pluto notebook can serve the same purpose.

From the Web Link for MSYS2 Read:

MSYS2

Software Distribution and Building Platform for Windows

MSYS2 is a collection of tools and libraries providing you with an easy-to-use environment for building, installing and running native Windows software.

It consists of a command line terminal called mintty, bash, version control systems like git and subversion, tools like tar and awk and even build systems like autotools, all based on a modified version of Cygwin. Despite some of these central parts being based on Cygwin, the main focus of MSYS2 is to provide a build environment for native Windows software and the Cygwin-using parts are kept at a minimum. MSYS2 provides up-to-date native builds for GCC, mingw-w64, CPython, CMake, Meson, OpenSSL, FFmpeg, Rust, Ruby, just to name a few.

To provide easy installation of packages and a way to keep them updated it features a package management system called Pacman, which should be familiar to Arch Linux users. It brings many powerful features such as dependency resolution and simple complete system upgrades, as well as straight-forward and reproducible package building. Our package repository contains more than 2000 pre-built packages ready to install.

For more details see 'What is MSYS2?' which also compares MSYS2 to other software distributions and development environments like Cygwin, WSL, Chocolatey, Scoop, ... and 'Who Is Using MSYS2?' to see which projects are using MSYS2 and what for.

- From the Web Link for Mingw-w64 Read:

Overview

Mingw-w64 is an advancement of the original mingw.org project, created to support the GCC compiler on Windows systems. It has forked it in 2007 in order to provide support for 64 bits and new APIs. It has since then gained widespread use and distribution.

The development and community are very active and welcoming with new contributors every month and simple installers.

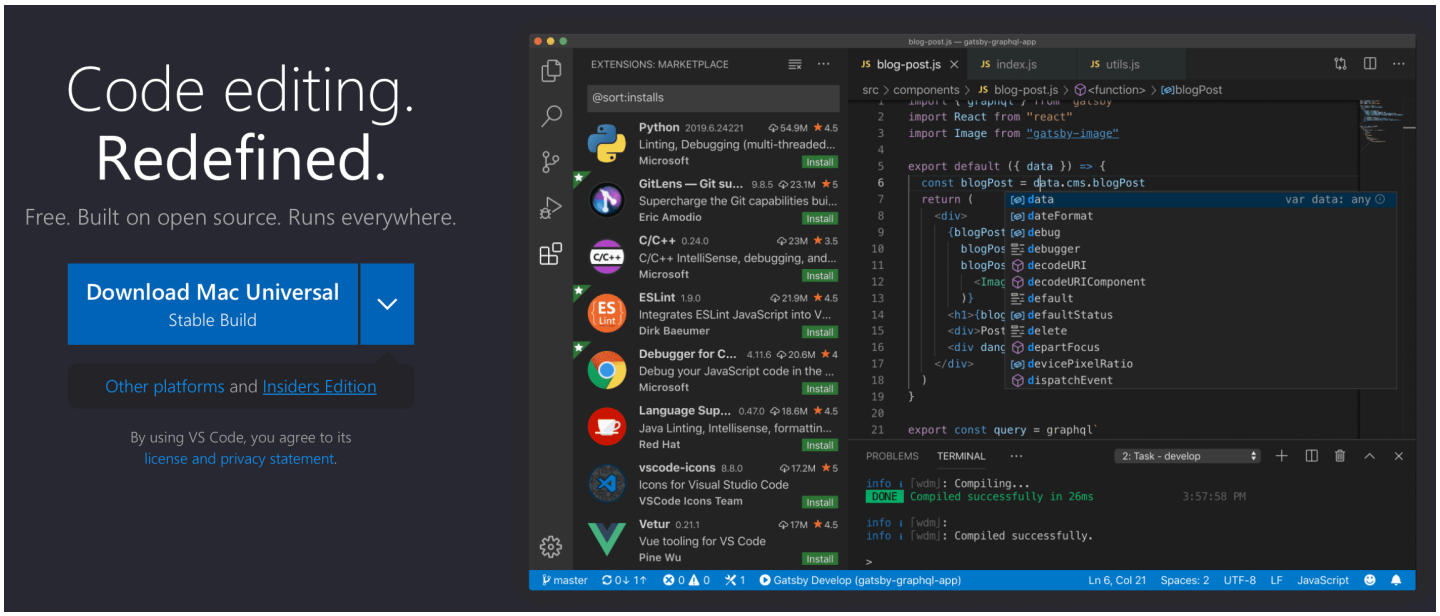
Headers, Libraries and Runtime

- More than a million lines of headers are provided, not counting generated ones, and regularly expanded to track new Windows APIs.
- Everything needed for linking and running your code on Windows.
- Winpthreads, a pthreads library for C++11 threading support and simple integration with existing project.
- Winstorecompat, a work-in-progress convenience library that eases conformance with the Windows Store.
- Better-conforming and faster math support compared to VisualStudio's.

Tools

- gendef: generate Visual Studio .def files from .dll files.
- genidl: generate .idl files from .dll files.
- widl: compile .idl files.

- From the Web Link for vs code Read:



Installing MSYS2 and Mingw-w64 under MSYS2

The steps to take here can be found in the MSYS2 web link. In particular this page provides nine total steps, with last two being comment steps, to get MSYS2 and Mingw-w64 setup on your system.

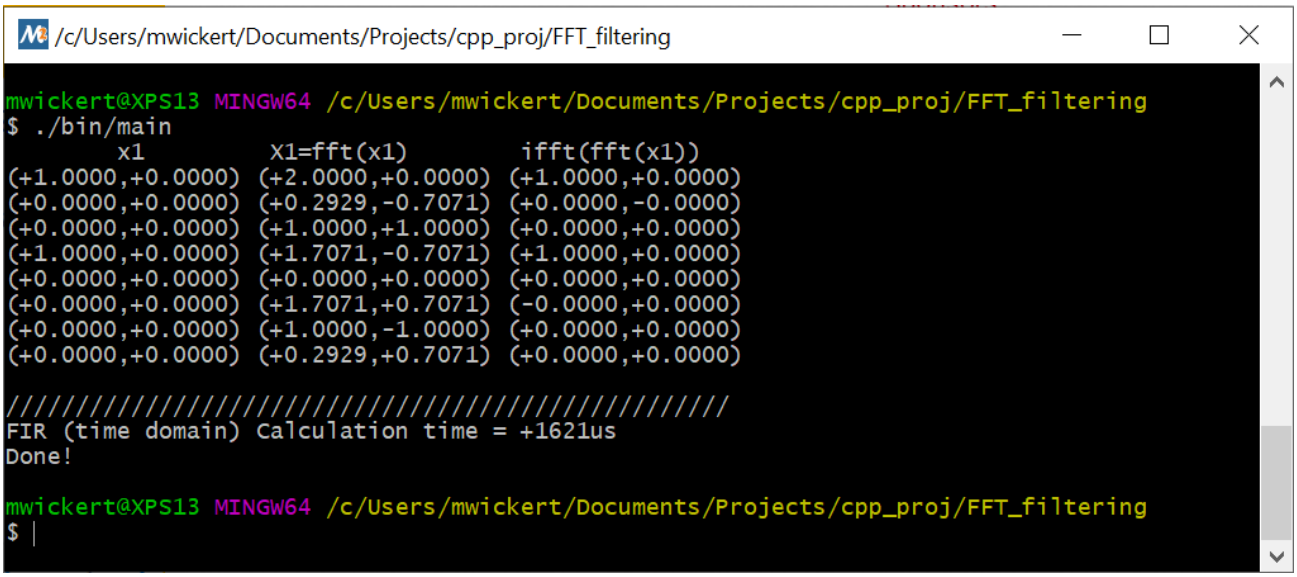
- Steps 1–7 bring you from download to full installation for our purposes. In Step 4 be sure to tick the check box to insure that the shell/terminal for MSYS2 will launch when completing the step.
  - In Step 5 you update the *package database* and the *base packages* using `$ pacman -Syu` in the "MSYS2 MSYS" shell:

  - In Step 6 you again use "MSYS2 MSYS", which can be launched from the maroon icon in the start menu,

  - This time you update the base packages using `$ pacman -Su`. Note all maintenance with MSYS2 is done using this shell
  - Finally in Step 7 you again using the "MSYS2 MSYS" shell to install GCC using via the

command line `$ pacman -S --needed base-devel mingw-w64-x86_64-toolchain`. New Icons now appear on the *Start* menu under `MSYS2 64bit`

- The blue `MSYS2 MinGW 64-bit` icon launches a shell/terminal that you can build and run GCC code. This shell and a text editor, perhaps notepad++ (with Language set to C++) would be a minimalist install for Problem 2, e.g. below I run the sample code after navigating to the folder where my `FFT_filtering` project is located; `make` can be run from this shell as well; my preference is to press on to install `vs code`



Installing `vs code` and Integrating Powershell

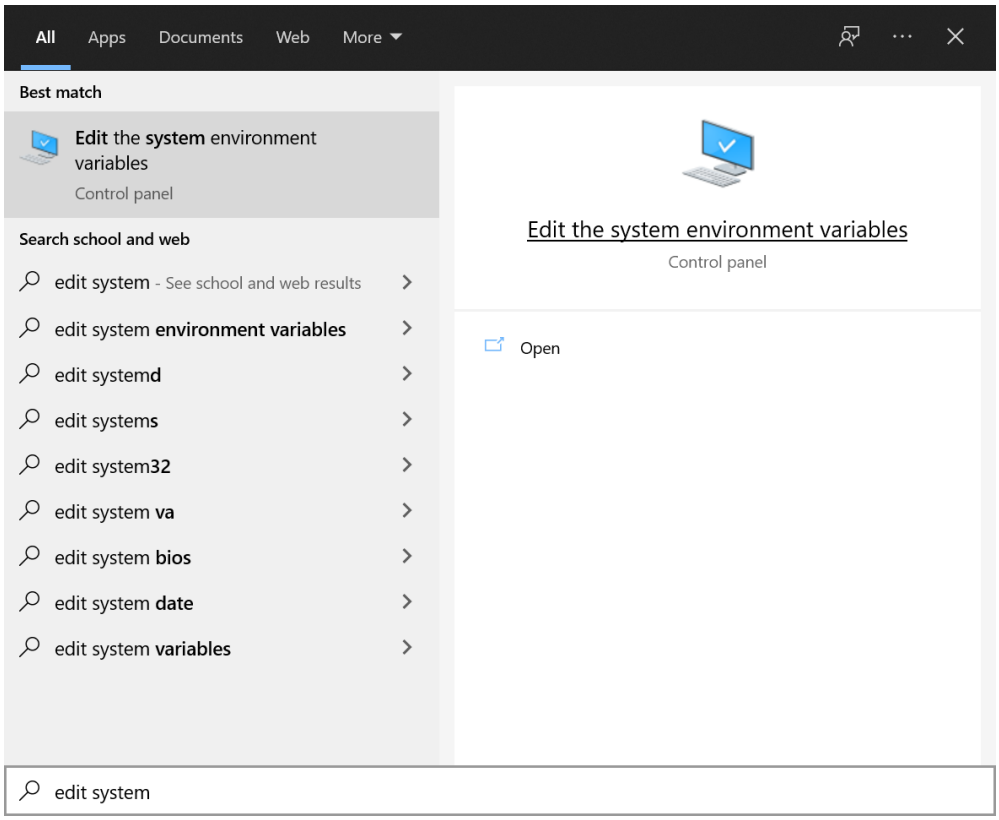
To make you environment more comfortable install vs code using the four steps *Prerequisites* in `vs code with mingw`. There is no need to go through the steps under *Create Hello World* unless choose. Under Step 4 there are five inner steps which involve setting *Environment Variables*. Ultimately you will be adding these paths:

```
C:\msys64\mingw64\bin
C:\msys64\usr\bin
```

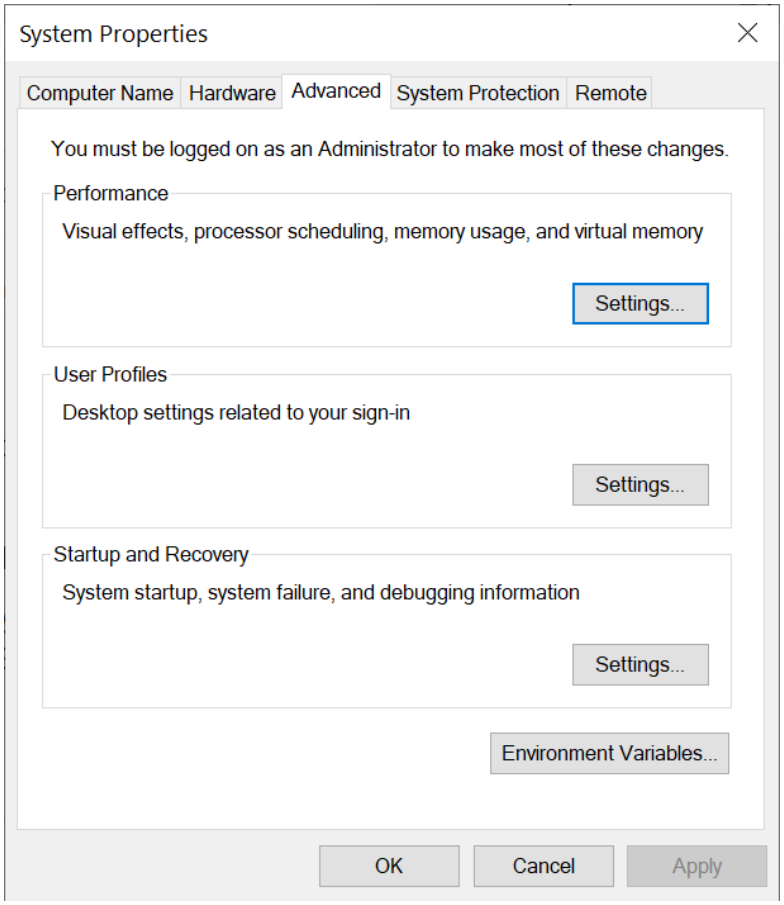
Setting Environment Variables from Scratch

If you are new to setting environment variables in Windows, I am providing here some screenshots to guide you through the process:

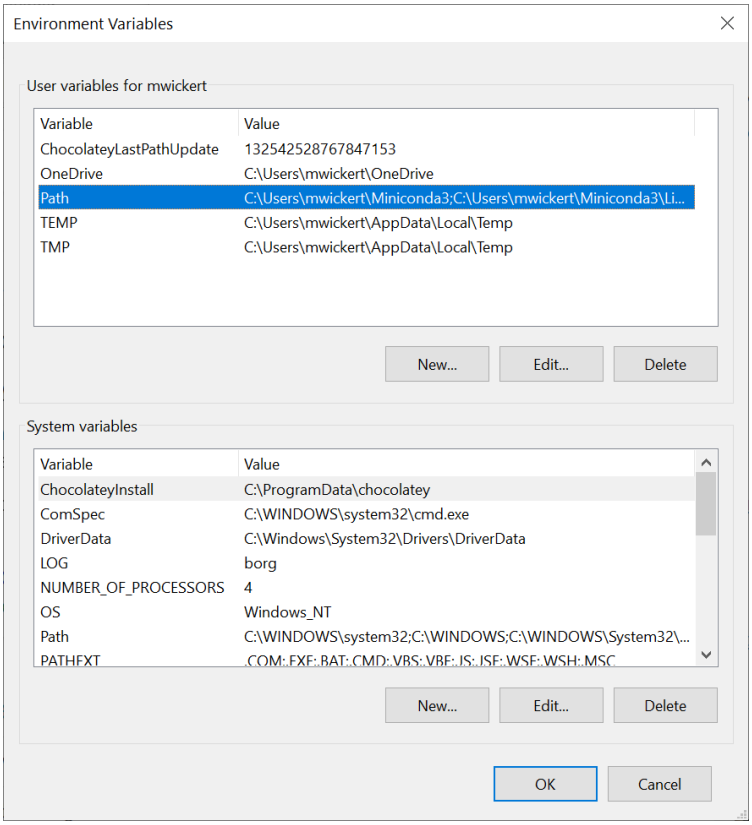
1. In the search field of the lower left task bar type something like `edit the system environment variables`; this should give the option seen at the top of the screenshot below:



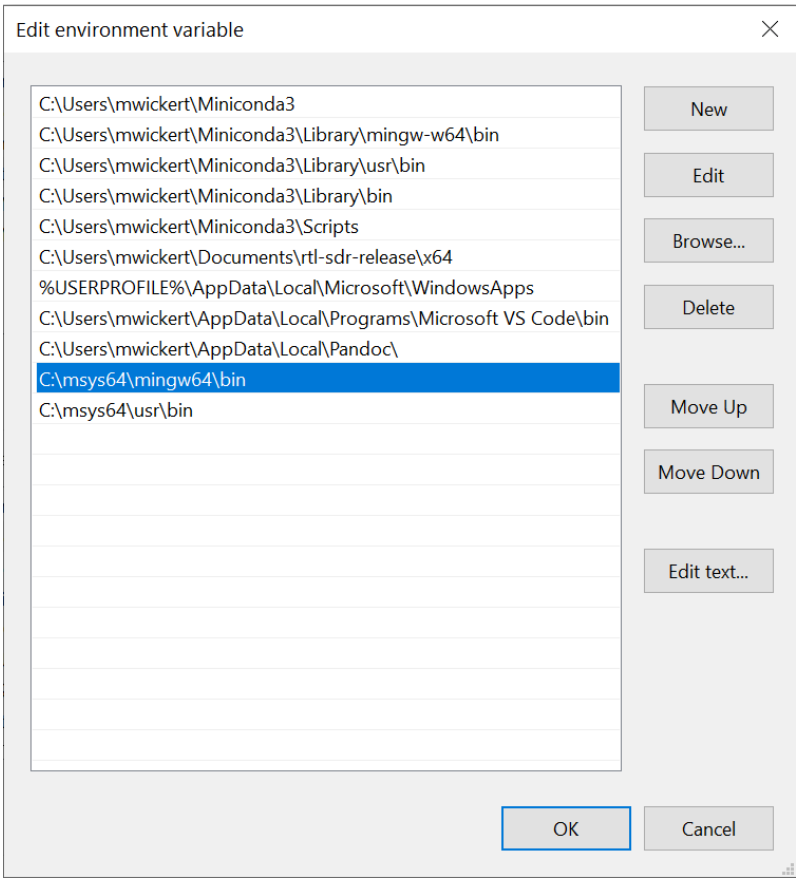
2. Choose the top option and you will be brought to:



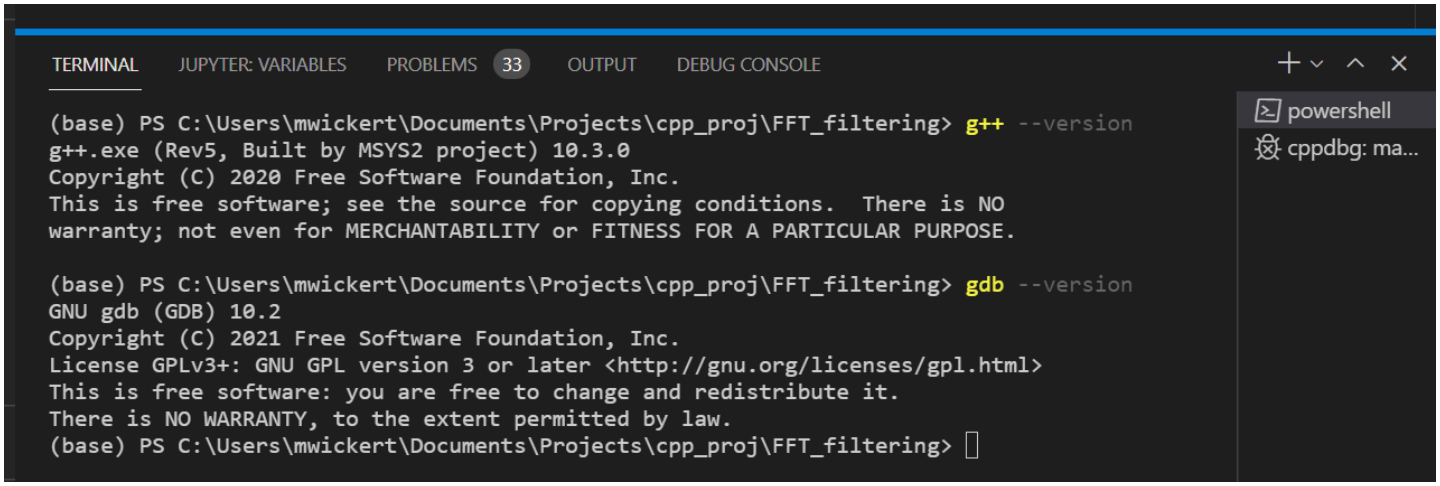
3. Next click the `Environment Variables` button in the lower left and you will be brought to



4. In the upper field choose `Path` (this is selected in the above screenshot), then click button `Edit...` just below, as in edit paths:



5. At this point you add the `New` paths shown at the bottom of the list (this assumes you installed MSYS2 in the default folder `C:\msys64`); OK out of all of the environment variable related open windows
6. Now you can move on to test that the paths work in `Powershell`, which is also described in the link `vs code with mingw` under the subsection *Check you MinGW Installation*:



Building (compile/link), Run, and Debugging

The final two steps are the build process and configuring the `vs code` debugger.

Building and Running Code

Since the project contains a `Makefile` the building step is easy. Inside `vs code` you can start a Powershell session using the Terminal app. This is shown in the above screenshot. Note the Terminal app normally sits at the bottom right of the `vs code` window. It may be hidden, but can be brought up using the key sequence `ctrl-back tic` (the upper left keyboard tic that shares the ~ key).

To build you project make sure the terminal is in the top level folder of the project, where `Makefile` resides, then type `make` at the terminal command prompt. The code fence below is a copy taken from the Powershell terminal in `vs code`. Here I ran `make` followed by code execution using `./bin/main`:

```
(base) PS C:\Users\mwickert\Documents\Projects\cpp_proj\FFT_filtering> make

g++ -std=c++17 -ggdb -Iinclude src/fft.cpp src/fir_float32.cpp
src/IQfile_IO.cpp src/main.cpp -o bin/main
(base) PS C:\Users\mwickert\Documents\Projects\cpp_proj\FFT_filtering>
./bin/main
        x1         X1=fft(x1)        ifft(fft(x1))
(+1.0000,+0.0000) (+2.0000,+0.0000) (+1.0000,+0.0000)
(+0.0000,+0.0000) (+0.2929,-0.7071) (+0.0000,-0.0000)
(+0.0000,+0.0000) (+1.0000,+1.0000) (+0.0000,+0.0000)
(+1.0000,+0.0000) (+1.7071,-0.7071) (+1.0000,+0.0000)
(+0.0000,+0.0000) (+0.0000,+0.0000) (+0.0000,+0.0000)
(+0.0000,+0.0000) (+1.7071,+0.7071) (-0.0000,+0.0000)
(+0.0000,+0.0000) (+1.0000,-1.0000) (+0.0000,+0.0000)
(+0.0000,+0.0000) (+0.2929,+0.7071) (+0.0000,+0.0000)

////////////////////////////////////////////////
FIR (time domain) Calculation time = +1660us
Done!
(base) PS C:\Users\mwickert\Documents\Projects\cpp_proj\FFT_filtering>
```

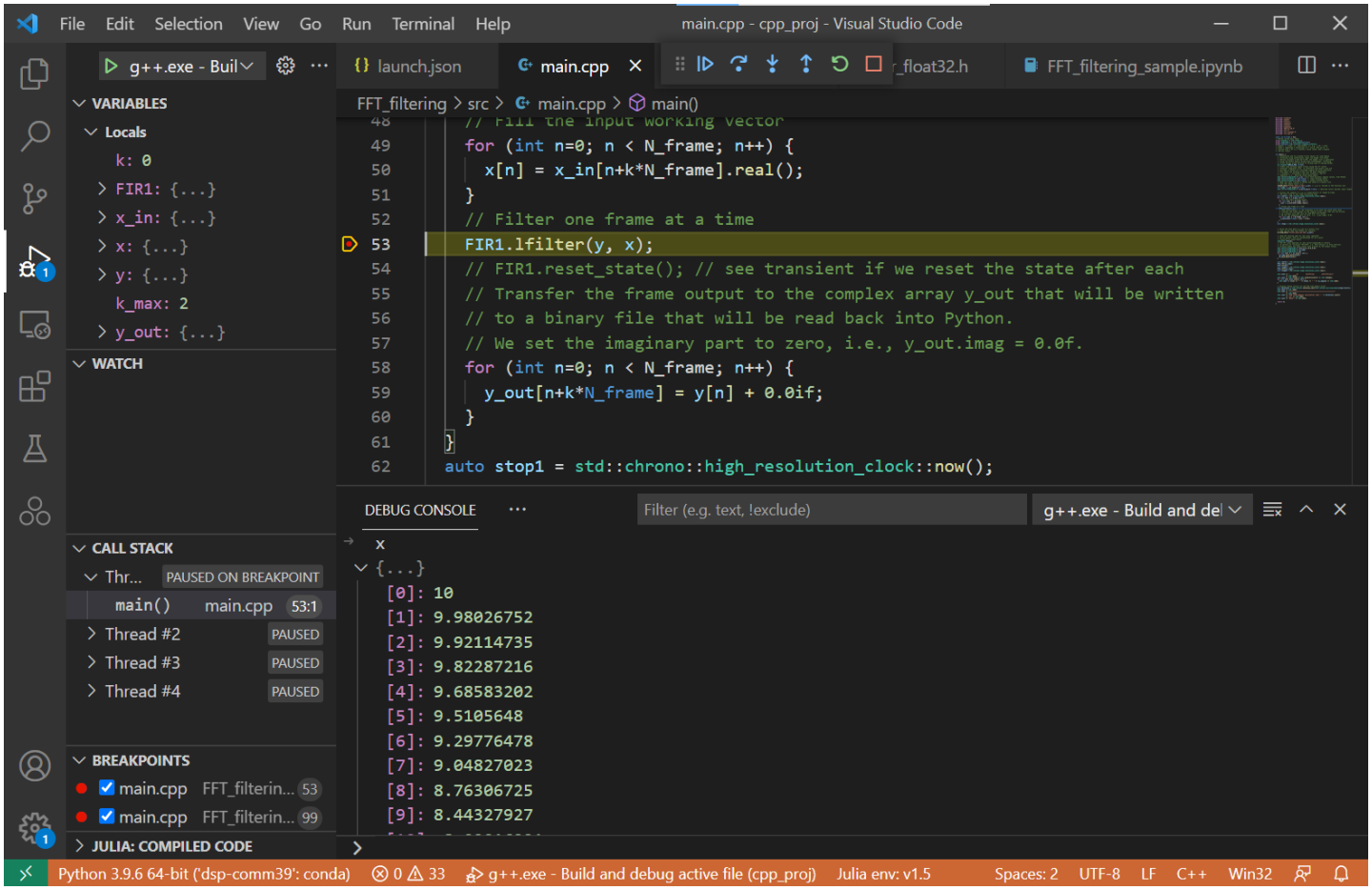
Configure the .vs.code` Debugger

The below is a `launch.json` file that sets up the C++ debugger on my XPS13 tablet. I will demo the remaining details of this in class.

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
```

```
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: Current File",
      "type": "python",
      "request": "launch",
      "program": "${file}",
      "console": "integratedTerminal"
    },
    {
      "name": "g++.exe - Build and debug active file",
      "type": "cppdbg",
      "request": "launch",
      "program": "${workspaceFolder}\\FFT_filtering\\bin\\main.exe",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}\\FFT_filtering",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "miDebuggerPath": "C:\\msys64\\mingw64\\bin\\gdb.exe",
      "setupCommands": [
        {
          "description": "Enable pretty-printing for gdb",
          "text": "-enable-pretty-printing",
          "ignoreFailures": true
        }
      ],
    }
  ]
}
```

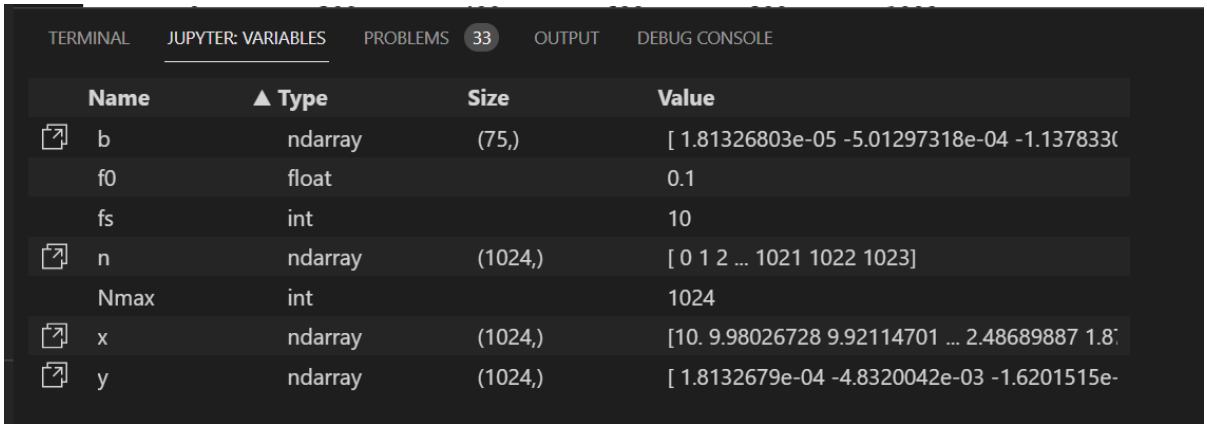
When the `launch.json` file is in place and configured properly appears as shown below when a break point is set on `FIR1.lfilter()`:



- The *DEBUG CONSOLE* lets you view the contents of variables and arrays and hovering over a variable in the code window lets you see the variable immediately

Options?

- Set up more `vs code` extensions:
  - Python and Python tutorial
  - Jupyter notebooks in `vs code`. The Jupyter tab adjacent to Terminal that display in the lower pane shows the below for the `FFT_filter_sample.ipynb` when loaded in `vs code` using the notebook extension:



- Git for Windows: `vs code` has Git version control build in, but you need to install the Git app itself
    - Once you have Git set up a good extension to add to `vs code` is Git History. To see it inside `vs code` click on the Extensions button on the far left side of vs code window and enter `Git History` in the search field.
- More...?