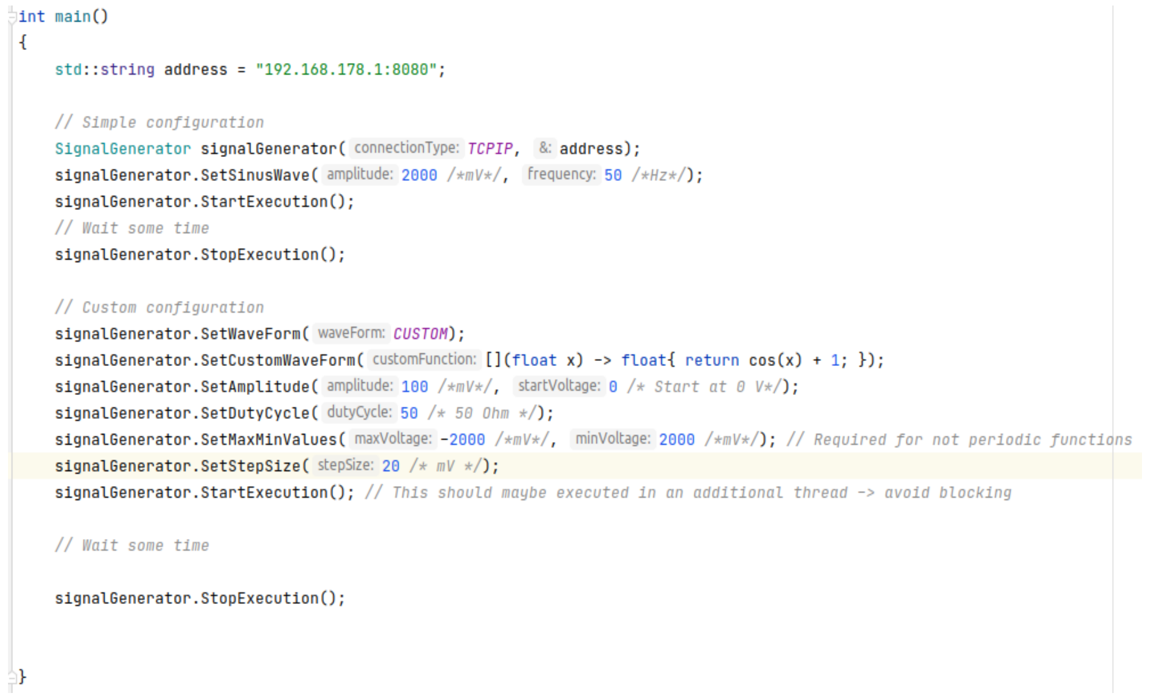
Signal Generator and Oscilloscope Lib

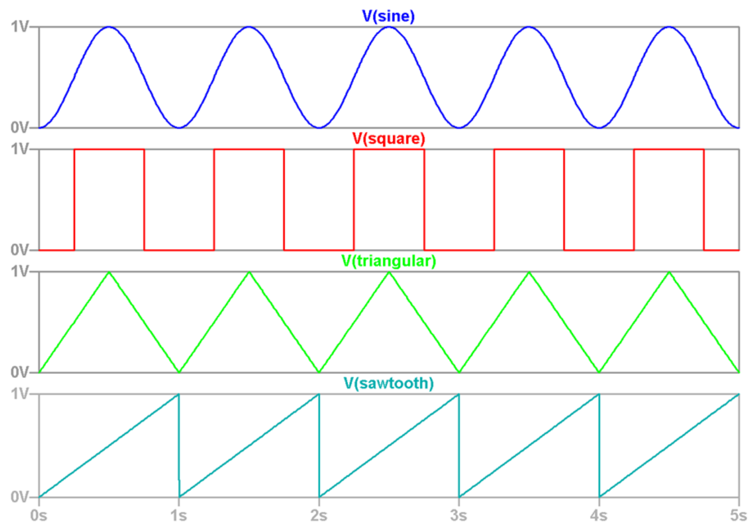
- Preconditions
  - Configururation of Signalgenerators, Oscilloscopes and DC power supplys programmatically
  - Support different Interfaces (Ethernet, USB)
  - Modular concept (simple integration of new devices)
  - Platform independent C++ library using the VISA communication protocol
  - Data serialization with different formats (e.g. Raw, ASCII)
  - Synchronous and asynchronous functions
  - Sufficient error handling

Signal Generator

Sample Configuration:



```
int main()
{
    std::string address = "192.168.178.1:8080";

    // Simple configuration
    SignalGenerator signalGenerator( connectionType: TCPIP,  &: address);
    signalGenerator.SetSinusWave( amplitude: 2000 /*mV*/,  frequency: 50 /*Hz*/);
    signalGenerator.StartExecution();
    // Wait some time
    signalGenerator.StopExecution();

    // Custom configuration
    signalGenerator.SetWaveForm( waveForm: CUSTOM);
    signalGenerator.SetCustomWaveForm( customFunction: [](float x) -> float{ return cos(x) + 1; });
    signalGenerator.SetAmplitude( amplitude: 100 /*mV*/,  startVoltage: 0 /* Start at 0 V*/);
    signalGenerator.SetDutyCycle( dutyCycle: 50 /* 50 Ohm */);
    signalGenerator.SetMaxMinValues( maxVoltage: -2000 /*mV*/,  minVoltage: 2000 /*mV*/); // Required for not periodic functions
    signalGenerator.SetStepSize( stepSize: 20 /* mV */);
    signalGenerator.StartExecution(); // This should maybe executed in an additional thread -> avoid blocking

    // Wait some time

    signalGenerator.StopExecution();


}
```



Oscilloscope

Sample Configuration:

```
void fetchCallbackFunction(float *retArray)
{
    // Do something with the return Array
}

int main()
{
    Oscilloscope oscilloscope(USB, "USBAddr");

    // Request sample point array
    float pointArray[100];
    oscilloscope.RequestPointArray(Channel1, pointArray, 100);

    // Trace output of channel 1 to file
    FileHandle fileHandle = oscilloscope.OpenTraceFile(ASCII, "./Logfile.txt");
    Channel channel1 = oscilloscope.GetChannel( channelID: Channel1);
    channel1.StartTrace( &: fileHandle);

    // Wait some time

    channel1.StopTrace( &: fileHandle);

    // Fetch periodically in the background after a certain time period
    FetchHandle fetchHandle = channel1.FetchPeriodically(fetchCallbackFunction(), 100, 2000 /* 2 seconds */);

    // Wait some time

    channel1.StopFetching(fetchHandle);

    // Make commputations between different channels
    FileHandle fileHandle = oscilloscope.OpenTraceFile(ASCII, "./Logfile.txt");
    Channel channel2 = oscilloscope.GetChannel( channelID: Channel2);
    channel1.SynchronizeChannels(channel2,  computation: [](float &v1, float &v2) -> float { return v1 + v2; }, fileHandle)


}
```