



# NL5 DLL

## User's Manual

Ver. 2.74

03/17/2021

[nl5.sidelinesoft.com](http://nl5.sidelinesoft.com)



## VERSION

This version of User's Manual is current for NL5 DLL ver. 2.74 (03/17/2021).  
The latest version of User's Manual can be found at [nl5.sidelinesoft.com](http://nl5.sidelinesoft.com).

## LIMITED LIABILITY

NL5 DLL, together with all accompanying materials, is provided on a “as is” basis, without warranty of any kind. The author makes no warranty, either expressed, implied, or stationary, including but not limited to any implied warranties of merchantability or fitness for any purpose. In no event will the author be liable to anyone for direct, incidental or consequential damages or losses arising from use or inability to use NL5 DLL.

## COPYRIGHTS

© 2021, A.Smirnov, Sidelinesoft LLC. The software and User's Manual are copyrighted. No portion of this Manual can be translated or reproduced for commercial purpose without the express written permission from the copyright holder.

---

**Microsoft**, **Windows**, and **Microsoft Visual C++** are registered trademarks of Microsoft Corporation. **MATLAB** is a registered trademark of The MathWorks, Inc. **PYTHON** is a registered trademark of the Python Software Foundation. **Borland C++ Builder** is a registered trademark of Borland Corporation. **Verilog** is a registered trademark of Cadence Design Systems. **Xilinx** and **Vivado** are registered trademarks of Xilinx.

# Table of Contents

<b>I. Introduction .....</b>	<b>4</b>
What is NL5 DLL .....	5
Version .....	5
Files .....	5
License .....	6
<b>II. Using DLL.....</b>	<b>7</b>
<b>Functions.....</b>	<b>8</b>
Function parameters.....	8
Function result .....	8
Handles .....	9
<b>Using DLL.....</b>	<b>10</b>
Error message.....	10
DLL information .....	10
Schematic .....	11
Parameters .....	12
Traces .....	14
Co-simulation.....	15
Inputs/Outputs .....	16
Simulation .....	17
Simulation data .....	21
<b>Using DLL with MATLAB .....</b>	<b>24</b>
<b>Using DLL with SystemVerilog .....</b>	<b>26</b>
Files .....	26
Using DLL .....	26
Using DLL with C-code “wrapper” .....	27
<b>Running co-simulation demo with Xilinx Vivado.....</b>	<b>28</b>
Creating demo project.....	28
Creating library file.....	32
Configuring and running demo.....	33
Demo circuit.....	38
<b>III. DLL Functions.....</b>	<b>39</b>
NL5_GetError.....	40
NL5_GetInfo .....	41
NL5_GetLicense.....	42
NL5_Open .....	43
NL5_Close.....	44
NL5_Save .....	45
NL5_SaveAs.....	46
NL5_GetValue.....	47
NL5_SetValue.....	48
NL5_GetText .....	49

NL5_SetText .....	50
NL5_GetParam .....	51
NL5_GetParamValue .....	52
NL5_SetParamValue .....	53
NL5_GetParamText .....	54
NL5_SetParamText .....	55
NL5_GetTrace .....	56
NL5_AddVTrace .....	57
NL5_AddITrace .....	58
NL5_AddPTrace .....	59
NL5_AddVarTrace .....	60
NL5_AddFuncTrace .....	61
NL5_DeleteTrace .....	62
NL5_GetInput .....	63
NL5_SetInputValue .....	64
NL5_GetOutput .....	65
NL5_GetOutputValue .....	66
NL5_SetStep .....	67
NL5_SetTimeout .....	68
NL5_GetSimulationTime .....	69
NL5_Start .....	70
NL5_Simulate .....	71
NL5_SimulateInterval .....	72
NL5_SimulateStep .....	73
NL5_SaveIC .....	74
NL5_GetDataSize .....	75
NL5_GetDataAt .....	76
NL5_GetLastData .....	77
NL5_GetData .....	78
NL5_DeleteOldData .....	79

<b>IV. Attachments .....</b>	<b>80</b>
END USER LICENSE AGREEMENT .....	81

# I. Introduction



## What is NL5 DLL

NL5 DLL is a standard Windows dynamic-linked library (DLL) included into NL5 Circuit Simulator package. It performs transient simulation of circuits created by NL5 Circuit Simulator, provides raw simulation data, allows modification of circuit parameters, adding data traces, and some other operations through DLL API functions. NL5 DLL can be used as an analog simulator which is started and controlled from other applications and tools (MATLAB, Python, custom C/C++ code) , and as an analog co-simulation tool working with digital simulation tools, for example SystemVerilog simulators (through DPI interface).

NL5 DLL user is supposed to be familiar with NL5 Circuit Simulator principle and operation. Please refer to NL5 Circuit Simulation Manual and Help for information.

For most of circuits, DLL simulation results will be identical to NL5 simulation results. However, due to specifics of compilers used for DLL (Microsoft Visual C++) and NL5 (Borland Builder), there could be very slight variations, specifically at switching points. Typically, the difference in resulting waveforms should be within one simulation step, which should not be critical for overall circuit behavior. That difference will be minimized as much as possible in the future versions of NL5 and DLL.

Please use public resources or specific documentation for general information about dynamic-linked libraries, SystemVerilog, and digital simulation tools.

## Version

Current released Version and Revision of NL5 DLL is always the same as Version and Revision of NL5 Circuit Simulator. This guarantees full compatibility in terms of components, models, features, and performance. However, there is nothing wrong in using different Versions/Revisions of DLL and NL5.

Current Build of DLL can be different from NL5, due to possible DLL and NL5 specific fixes and modifications.

NL5 DLL is distributed as part of NL5 complete package, as well as separate, DLL only compressed file. The package, all files, and documentation can be found at [nl5.sidelinesoft.com](http://nl5.sidelinesoft.com).

## Files

NL5 DLL is a standard Windows dynamic-linked library (DLL) created using Microsoft Visual C++. The following files are distributed to customers:

- nl5\_dll\_32.dll
- nl5\_dll\_32.lib
- nl5\_dll\_64.dll
- nl5\_dll\_64.lib
- nl5\_dll.h
- nl5\_dll.pdf - this Manual
- MATLAB/ - demo files for MATLAB
- SystemVerilog/ - supporting files for SystemVerilog
- SystemVerilog/Vivado/ - supporting files for co-simulation with Vivado

## License

NL5 DLL does not require a license. Instead, it can simulate only files created by NL5 Simulator running with the License which has **DLL option** enabled. For demonstration purposes, small files (containing not more than 10 components) still can be simulated without DLL option.

DLL option can be selected while ordering the License. If you own NL5 License already, and want to include DLL option into your License, please contact us at [nl5.sidelinesoft.com](http://nl5.sidelinesoft.com), or send email to [nl5@sidelinesoft.com](mailto:nl5@sidelinesoft.com).

## II. Using DLL





# Functions

## Function parameters

The following parameter types are used in DLL functions:

`int` - 32-bit integer  
`double` - 8-byte floating point  
`char*` - pointer to null-terminated ASCII (1-byte) character string (character array)

Some functions return `double` values through pointers to `double` variable (`double*`) provided as a parameter of the function.

## Function result

Most of DLL functions return integer value: function result. If function result is negative, it is an error code. Only error code -1 is currently used, however more error codes may be added in the future. It is not recommended to continue DLL execution if error code was received, since it may result in DLL crash.

If error code is returned, text description of the error can be obtained by `NL5_GetError` function:

```
if(NL5_GetValue(ncir, "R1.R", *value) < 0)
{
    printf("%s", NL5_GetError());
}
```

In case of successful execution, some functions return 0, and some functions return non-negative integer value, with the meaning depending on the function. For example, `NL5_Open` returns integer value: circuit handle, `NL5_GetText` returns number of characters placed into the character array, etc.:

```
int ncir = NL5_Open("rc.nl5");
if(ncir < 0)
{
    printf("%s", NL5_GetError());
}
```

Functions `NL5_GetInfo` and `NL5_GetError` return pointer to null-terminated ASCII character string:

```
char* str = NL5_GetInfo();  
printf("%s", str);
```

The content of that string is valid only until execution of the next DLL function: then it will be changed. If the text requested by calling those functions is needed for the future use, it is user's responsibility to copy it to safe location.

## Handles

**Handle** is an index of the object in the internal DLL objects list. Handle is non-negative integer value. Some functions return handle as a function result. The handle referring to a specific object can be used as a parameter for other functions, related to that object. Handles are used for circuits, component parameters, inputs/outputs, and traces.

For example, function result of function `NL5_Open` is circuit handle. Once received, the handle can be used as an `ncir` parameter for many other functions, such as `NL5_Simulate`, `NL5_GetValue`, `NL5_GetParam`, `NL5_GetTrace`, etc.:

```
int ncir = NL5_Open("rc.nl5");  
if(ncir < 0)  
{  
    printf("%s", NL5_GetError());  
}  
  
double r;  
if(NL5_GetValue(ncir, "R1.R", &r) < 0)  
{  
    printf("%s", NL5_GetError());  
}
```

# Using DLL

## Error message

A general function which may be called after calling practically any other function is `NL5_GetError`. It returns text description of the error which might occur while executing previous function, or "OK" if execution was successful:

```
if(NL5_GetValue(ncir, "R1.R", *value) < 0)
{
    printf("%s", NL5_GetError());
}
```

## DLL information

A function you might want to call at DLL startup is `NL5_GetInfo`. It returns information about DLL: version and date. After calling `NL5_GetInfo`, call `NL5_GetError` to see DLL information:

```
char* str = NL5_GetInfo();
printf("%s", str);
```

This information is useful for troubleshooting, so please provide it when submitting bug reports or other requests:

## Schematic

To perform simulation, a schematic should be loaded into the DLL from a schematic “\*.nl5” file. Once loaded, the schematic is stored in the DLL memory, and can be used for simulation. During simulation, the circuit component parameters can be modified by DLL, and simulated data will be saved as a traces. A modified schematic with simulation data can be saved back into the schematic file.

To load schematic into DLL use `NL5_Open` function. If file name does not have a path, DLL will look for a file in the same directory where DLL is located. The function returns non-negative circuit handle `ncir`, which will be used in other DLL functions to identify the circuit:

```
int ncir = NL5_Open("rc.nl5");
if(ncir < 0)
{
    printf("%s", NL5_GetError());
}
```

If schematic file could not be loaded for any reason, a negative error code is returned. Also, an error occurs if requested file consists of too many components (currently 10), and is not DLL-enabled. Call `NL5_GetError` function to get text description of the error.

You can load several circuits at once by calling `NL5_Open`: a unique circuit handle will be returned for each circuit. If circuit is not needed anymore, it can be closed by `NL5_Close` function, however closing the circuit is not required.

The circuit can be saved back to the same schematic file by calling `NL5_Save`, or to a new file by calling `NL5_SaveAs` functions:

```
int ncir = NL5_Open("rc.nl5");
NL5_SetValue(ncir, "R1.R", 123.456);
NL5_SaveAs(ncir, "rc_new.nl5");
NL5_Close(ncir);
```

Use these functions to save schematic back to the file if any modification of component parameters were made by DLL, IC (Initial Conditions) were saved, or if you want to save schematic with obtained simulation data.

To save schematic with transient data, make sure the “Save with transient data” option is set in the schematic file. To set the option, open schematic file in NL5, go to File/Properties/Save, select “Save with transient data” checkbox, and save schematic into the file.

## Parameters

DLL functions can access and modify component parameters. Parameters can be modified before simulation is started, as well as between DLL simulation calls. This is similar to pausing NL5 simulation, changing the parameter, and continuing the simulation.

Please be aware that changing the parameter between DLL simulation calls will result in recalculating the system matrix, and switching to a new linear range of simulation. If parameters are being changed often, it may affect simulation speed. For better performance while co-simulation, DLL **input/output** functions can be used: those functions will modify the value of voltage/current sources of the schematic in a “continuous manner”, keeping the simulation in the same linear range.

To specify parameter name in parameter-related function, use component parameter name in the format <component>.<parameter> ("R1.R", "V1.V"). See NL5 Circuit Simulator Manual for details (User Interface/Data format/Names).

There are two methods to access component parameters:

1. Direct;
2. Through parameter handle.

**Direct** method is an easiest one, however not optimal in terms of performance. To get component parameter value, use `NL5_GetValue` function. It returns value into the variable of `double` type. The pointer to that variable is passed to the function as a parameter:

```
double value;
NL5_GetValue(ncir, "R1.R", &value);
```

See Reference for explanation on working with different parameter types.

To set parameter value, use function `NL5_SetValue`:

```
NL5_SetValue(ncir, "R1.R", 123.456);
```

To get/set parameter value represented as a text, use `NL5_GetText` and `NL5_SetText` functions. These functions are applicable to practically all parameter types, including numerical. If numerical parameter is defined as a formula, those functions will get/set text of the formula:

```
char str[100];

NL5_SetText(ncir, "V1.Slope", "Linear");

NL5_GetText(ncir, "V1.Slope", str, 100);
// returns str = "Linear"

NL5_GetText(ncir, "R1.R", str, 100);
// returns str = "1.23e-3"

NL5_SetText(ncir, "R2.R", "=R1.R*2");

NL5_GetText(ncir, "R2.R", str, 100);
// returns str = "=R1.R*2"
```

Accessing parameters **through parameter handle** would be a better option if parameter is being accessed at least several times. Using that method improves performance by parsing parameter name and searching for required component and parameter only once while obtaining parameter handle.

Use `NL5_GetParam` function to obtain the parameter handle first:

```
int nparam = NL5_GetParam(ncir, "R1.R");
if(nparam < 0)
{
    printf("%s", NL5_GetError());
}
```

Then use the parameter handle in functions `NL5_GetParamValue`, `NL5_SetParamValue`, `NL5_GetParamText`, and `NL5_SetParamText`:

```
SetParamValue(ncir, nparam, 1.0);
. . .
double r;
GetParamValue(ncir, nparam, &r);
```

## Traces

DLL will store simulation data for all traces specified in the schematic file. The data can be accessed through the trace handle, obtained by `NL5_GetTrace` function:

```
int ntrace = NL5_GetTrace(ncir, "V(R1)");
if(ntrace < 0)
{
    printf("%s", NL5_GetError());
}
```

A new trace can be added using functions `NL5_AddVTrace`, `NL5_AddITrace`, `NL5_AddPTrace`, `NL5_AddVarTrace`, and `NL5_AddFuncTrace`. These functions return trace handle. In the following example, a trace with voltage across resistor R1 is added:

```
int ntrace = NL5_AddVTrace(ncir, "R1");
if(ntrace < 0)
{
    printf("%s", NL5_GetError());
}
```

To minimize memory consumption, or accelerate simulation, any trace can be deleted by `NL5_DeleteTrace` function:

```
NL5_DeleteTrace(ncir, ntrace);
```

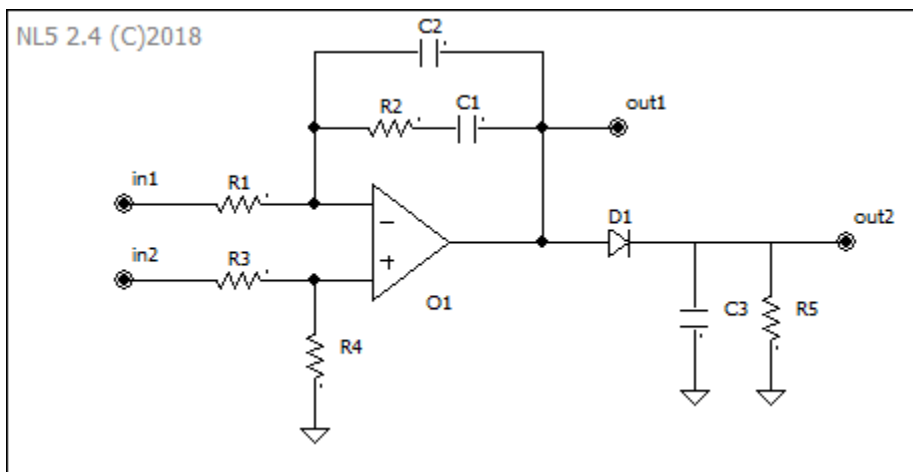
## Co-simulation

NL5 DLL can be used for co-simulation with other tools, such as system-modeling, behavioral modeling tools, or digital simulators. DLL will provide fast and reliable simulation of analog part of the system. To provide better performance of co-simulation, the following system structure is suggested.

The analog circuit has constant voltage or current sources (Labels, Voltage or Current source components) specified as **inputs**. The voltage or current value of those inputs are modified by the other tool before calling DLL simulation.

Also, the analog circuit has Labels specified as **outputs**. When DLL simulation is completed for requested interval, the voltages at specified outputs are transferred to the other tool as a result of analog simulation.

Here is an example of an analog part of the system, with two inputs ("in1", "in2") and two outputs ("out1", "out2"):



Please note that input signals are modified in a “continuous” manner, keeping the simulation in the same linear range, thus providing fast simulation. However, any component parameters can be modified using parameter-based functions (for example `NL5_SetValue`) as well: this will result in recalculating the system matrix, and switching to a new linear range of simulation.

If state of switch component needs to be modified, use voltage-controlled switch controlled by the input voltage source.

Please note that DLL won't store all simulated data at specified outputs: only last simulated data at the output is being stored until the next simulation call. However, DLL will still store data of all traces, specified in the circuit file, or added by calling DLL function. When the circuit is saved back into schematic file, the simulated data of those traces will be saved too, if “Save with transient data” option is set in the schematic file. To set the option, open schematic file in NL5, go to File/Properties/Save, select “Save with transient data” checkbox, and save schematic into the file.

Use **inputs/outputs** DLL functions to specify inputs and outputs for co-simulation.



## Inputs/Outputs

Inputs/outputs can be accessed through the input/output handle.

**Inputs.** Call `NL5_GetInput` function to specify the input. 3 types of components can serve as an input:

- Label component, model “V” (constant voltage source)
- Voltage source component (V), model “V” (constant voltage source)
- Current source component (I), model “I” (constant current source)

Provide the label/component name as a parameter of the function. Please note that once component is selected as an input, its model will be automatically set to the model “V” (constant voltage source), or model “I” constant current source). The function returns non-negative integer value: input handle:

```
int nin = NL5_GetInput(ncir, "in1");
if(nin < 0)
{
    printf("%s", NL5_GetError());
}
```

Use the handle and a desired source value to set input voltage/current by `NL5_SetInputValue` function:

```
int nin = NL5_GetInput(ncir, "in1");
. . .
NL5_SetInputValue(ncir, nin, 10.0);
```

**Outputs.** Call `NL5_GetOutput` function to specify the output. Only one component type can serve as an output: the Label. Provide the label name as a parameter of the function. Please note that once the label is selected as an output, its model will be automatically be set to the model “Label”. The function returns non-negative integer value: output handle:

```
int nout = NL5_GetOutput(ncir, "out1");
if(nout < 0)
{
    printf("%s", NL5_GetError());
}
```

Use the handle and a pointer to the double variable to obtain output voltage by `NL5_GetOutputValue` function:

```
int nout = NL5_GetOutput(ncir, "out1");
. . .
double v;
NL5_GetOutputValue(ncir, nout, &v);
```

Typically, `NL5_SetInputValue` functions should be called for each specified input before calling DLL simulation function, and `NL5_GetOutputValue` functions should be called for each specified output after simulation function returns. However, those functions can be called any time. Also, only functions for inputs/outputs that changed, or are needed for another tool, can be called.

## Simulation

Simulation is performed with **simulation step** defined in the schematic file (see NL5 transient settings: Transient/Settings/Calculation step). If needed, the step can be modified any time by `NL5_SetStep` function:

```
double step = 1.0e-6;
NL5_SetStep(ncir, step);
```

To prevent DLL from being “stuck” due to erroneous code of C-code component (infinite while/do/for loop), or inability to resolve states of piece-wise linear components, a **simulation time-out** can be set up using function `NL5_SetTimeout`:

```
int time_out = 3;
NL5_GetTimeout (ncir, time_out);
```

If simulation time of one transient step exceeds the time-out value (in seconds), the simulation will stop with error message. Time-out equal to zero disables time-out detection.

DLL keeps track on current **simulation time** in the internal `simulation_time` variable. When simulation function is called, simulation is continued for requested interval starting from current `simulation_time`. Current `simulation_time` value can be obtained by `NL5_GetSimulationTime` function:

```
double current_time;
NL5_GetSimulationTime(ncir, &current_time);
```

To **start simulation**, call `NL5_Start` function. It resets `simulation_time` to 0, initializes circuit components, erases existing simulation data, and calculates initial state of the circuit according to specified Initial Conditions. This function should be called first to start simulation from  $t=0$ , prior to calling any simulation functions. When `NL5_Start` returns, the simulation data consists of circuit state at  $t=0$ . The simulation data at  $t=0$  can be obtained by data-related functions described later.

However, calling `NL5_Start` is not required. It will be executed automatically if any of simulation functions is called, and simulation has not been performed yet.

After simulation is started, there are three methods of performing simulation:

1. Simulate;
2. Simulate interval;
3. Simulate step.

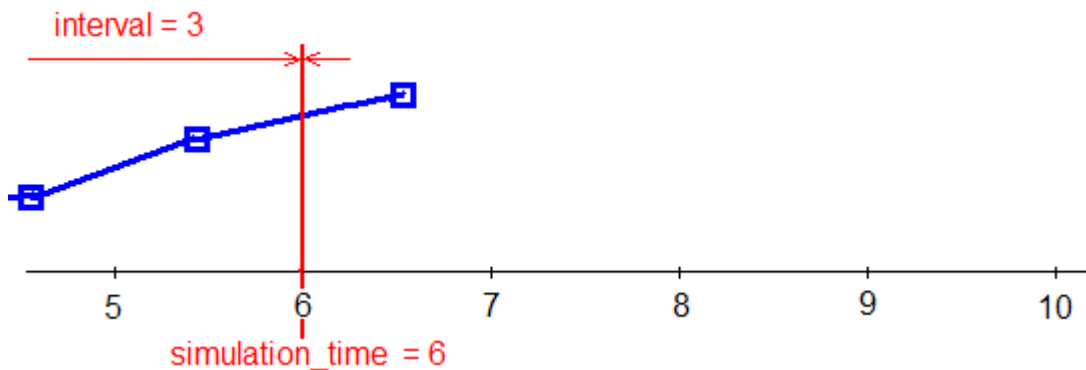
You can use just one method during all simulation, or any combination in any order.

**Simulate** method is performed by `NL5_Simulate` function, and it runs simulation **at least** for requested interval. First, current `simulation_time` variable is incremented by requested interval. Then simulation is performed until the time of calculated data is equal or greater than new `simulation_time`. The function does not change simulation step in order to stop exactly at the end of requested interval (new `simulation_time`), so the time of the last calculated data may exceed new `simulation_time`. When next simulation function is called, simulation will be continued with simulation step equal to the last simulation step.

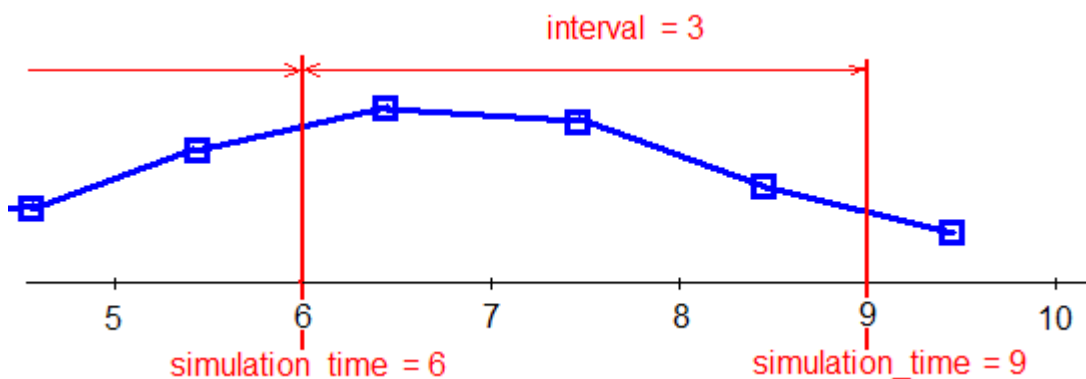
Here is an example of two consecutive calls of `NL5_Simulate` function. The first call was made at  $t = 3\text{s}$  (not shown on the graph), with `interval = 3\text{s}`:

```
NL5_Simulate(ncir, 3.0);
```

Simulation stopped when the time of the data point exceeded new `simulation_time = 6\text{s}`. The time of the last calculated data point =  $6.5\text{s}$ , the last simulation step =  $1\text{s}$ .



When `NL5_Simulate` function with the same  $3\text{s}$  interval is called again, simulation continues with the same simulation step =  $1\text{s}$ , and stops at `simulation_time = 9\text{s}`, with last calculated data point at  $t = 9.5\text{s}$ :



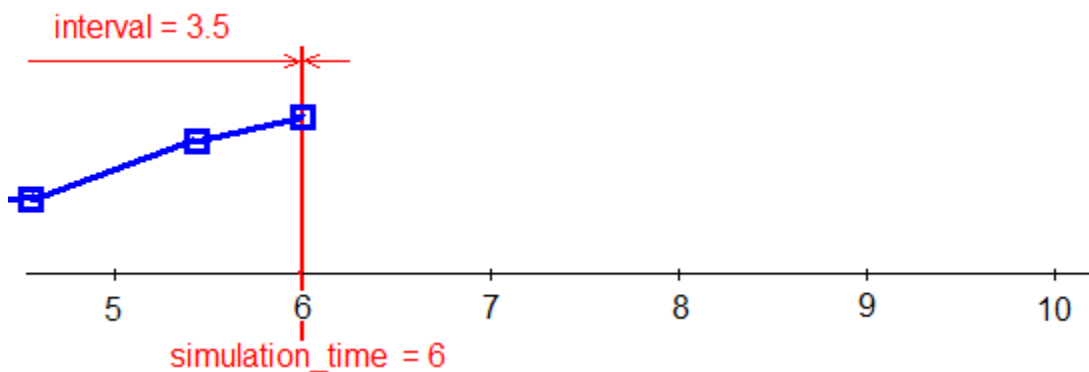
Using `NL5_Simulate` function provides the best simulation performance. It won't decrease simulation step at the end of current linear range, so that there is no need to restore the step back as simulation continues. Thus, the simulation will be performed in a fastest manner, regardless of simulation interruptions.

**Simulate interval** method is performed by `NL5_SimulateInterval` function, and it runs simulation **exactly** for requested interval. Unlike `NL5_Simulate`, it will decrease simulation step if needed to stop exactly at the end of the requested interval. When next simulation function is called, simulation step will be restored, and a new linear range will be started.

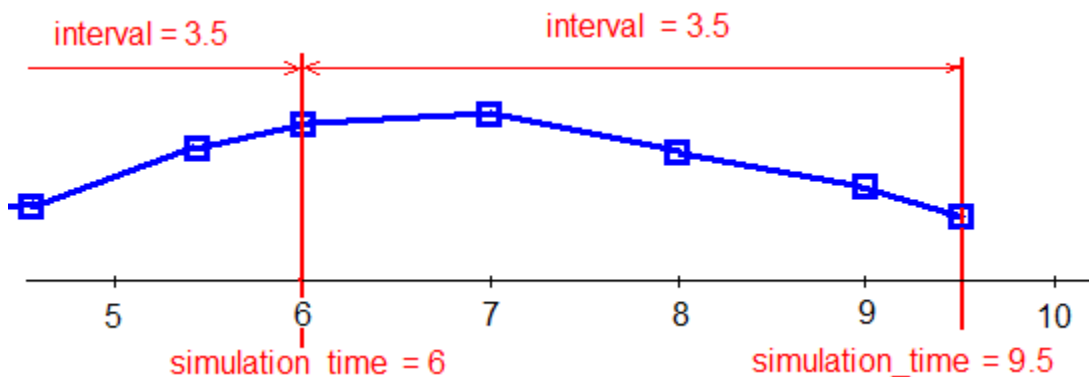
Here is an example of two consecutive calls of `NL5_SimulateInterval` function. The first call was made at  $t = 2.5\text{s}$  (not shown on the graph), with interval = 3.5s:

```
NL5_SimulateInterval(ncir, 3.5);
```

Simulation was performed with constant simulation step = 1s. Simulation stopped exactly at `simulation_time = 6s`, as requested. In order to do that, the last simulation step was decreased from 1s down to 0.5s:



When `NL5_SimulateInterval` is called again with requested interval = 3.5s, simulation step is restored back to 1s, and simulation continues:



In this call, simulation step was also decreased at the end of the interval from 1s down to 0.5s, in order to stop exactly at 9.5s.

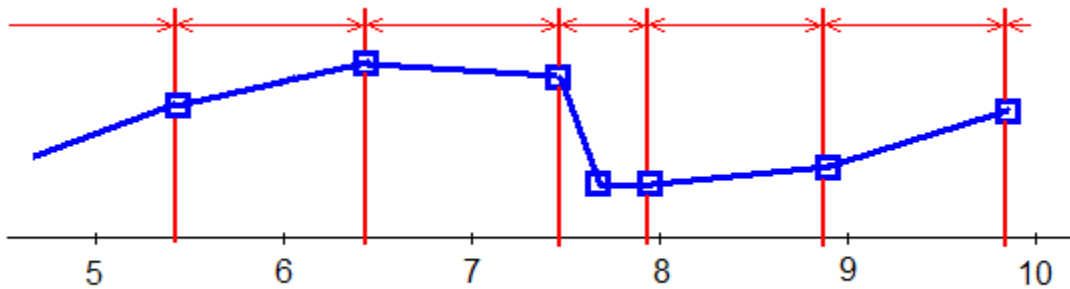
Due to possible change of simulation step even within the linear range, using of `NL5_SimulateInterval` may result in extremely slow simulation (especially if requested interval is small, and comparable with simulation step). Use this function only if it is really needed for your task.

**Simulate step** method is performed by `NL5_SimulateStep` function, and it executes just **one simulation step**. At the end, `simulation_time` is incremented by that simulation step, so that `simulation_time` is always equal to the time of last calculated data point.

Please note that in some situations (the very beginning of simulation, switching, change of linear range, etc.) more than one simulation step should be executed without interruption. In this case, `NL5_SimulateStep` will execute more than one simulation step.

Here is an example of simulation using `NL5_SimulateStep` function:

```
NL5_SimulateStep(ncir);
```



Most of the calls were in the same linear range of the circuit, so that only one step was executed. However, when switching occurs at  $t = 7.5\text{s}$ , two small uninterrupted steps were executed.

`NL5_SimulateStep` function can be used if DLL performs co-simulation with another simulation tool, when it should continuously provide state of analog circuit with minimal possible time interval.

One more function related to simulation is `NL5_SaveIC`. Calling this function is similar to executing command `Transient/Save IC` in the NL5 Circuit Simulator. Current Initial Conditions are saved into components in the DLL memory. Use `NL5_Save` or `NL5_SaveAs` to save components with new Initial Conditions into the schematic file.

## Simulation data

NL5 saves all simulated data points into DLL memory. To obtain data of a specific trace, first obtain trace handle by calling `NL5_GetTrace` function:

```
int ntrace = NL5_GetTrace(ncir, "V(R1)");
if(ntrace < 0)
{
    printf("%s", NL5_GetError());
}
```

There are three ways to retrieve the data of the trace:

1. Read interpolated data;
2. Read data of a specific data point;
3. Read last data.

To read **interpolated data** at specific time, use `NL5_GetData` function with the time as a parameter, and pointer to double for amplitude of the data point:

```
double data;
NL5_GetData(ncir, ntrace, 1.234, &data);
```

Please be aware that interpolated data are calculated using linear interpolation, and may not accurately represent actual signals of the circuit between calculated data points.

To read the data of a **specific data point**, use `NL5_GetDataAt` function with index of the data point. Provide pointers to double variables for time and amplitude of the data point:

```
double t, data;
int index = 123;
NL5_GetDataAt(ncir, ntrace, index, &t, &data);
```

Data point index is zero-based: index of the first data point is 0, index of the last data point is equal to number of data points minus 1. Use `NL5_GetDataSize` function to obtain number of data points available for the trace:

```
int ndata = NL5_GetDataSize(ncir, ntrace);
if(ntrace < 0)
{
    printf("%s", NL5_GetError());
}
```

Please note that the number of data points can be different for different traces due to data compression.

To read **last data**, use `NL5_GetLastData` function with pointers to double variables for time and amplitude of the data point:

```
double t, data;  
NL5_GetLastData(ncir, ntrace, &t, &data);
```

This function returns the data of last calculated data point.

As mentioned before, `NL5_Start` function erases all existing simulation data. Then, during simulation, all data points are being stored into DLL memory. There is a special algorithm in place to reduce the memory required for the data which are not changing (constant voltage/current supplies, output of digital components, etc.). However, if simulation is performed with small simulation step, the total available memory of the DLL can be easily exceeded.

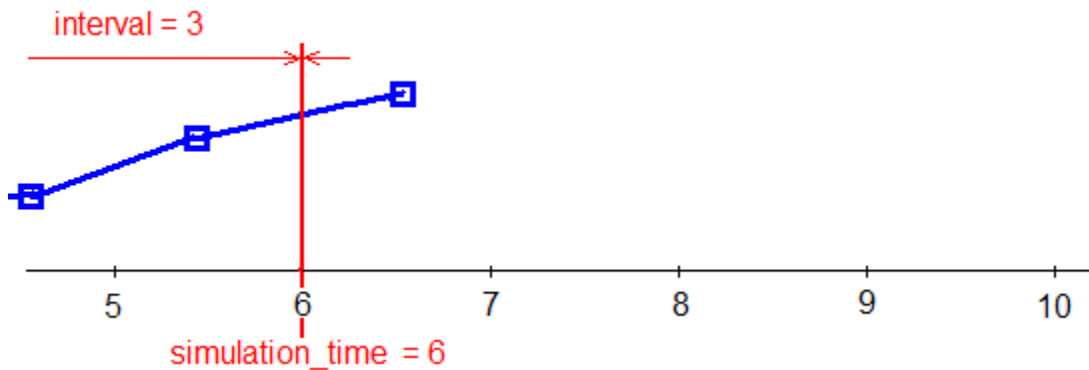
If large amount of simulated data is expected, it is recommended to upload simulated data to your application after from time to time, and delete that data from DLL memory by calling

`NL5_DeleteOldData` function:

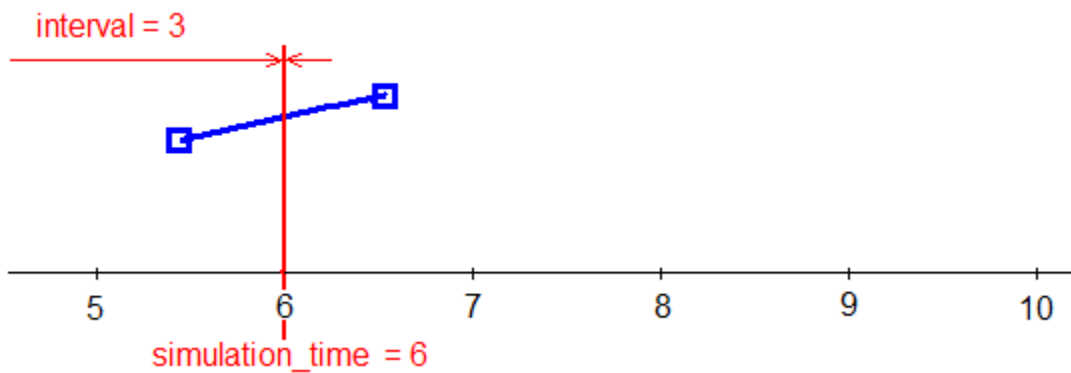
```
NL5_DeleteOldData(ncir);
```

In fact, that function does not erase all the data: it always leaves the very last calculated data point, or two data points, in order to be able to obtain interpolated data in the new interval.

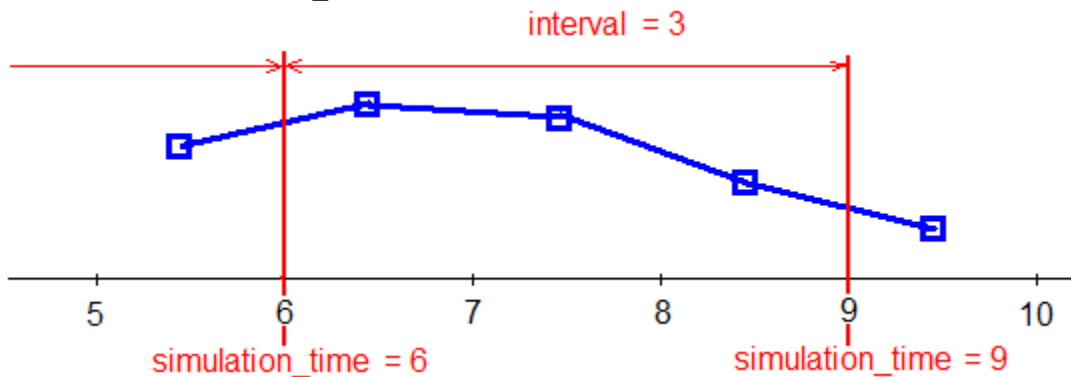
In the following example, simulation stopped at `simulation_time = 6s`, with last simulation step = 1s, and last calculated data point at  $t = 6.5s$ :



When `NL5_DeleteOldData` function is called, it will erase old data, except last two points:



After the next call of `NL5_Simulate`, the stored data would be:



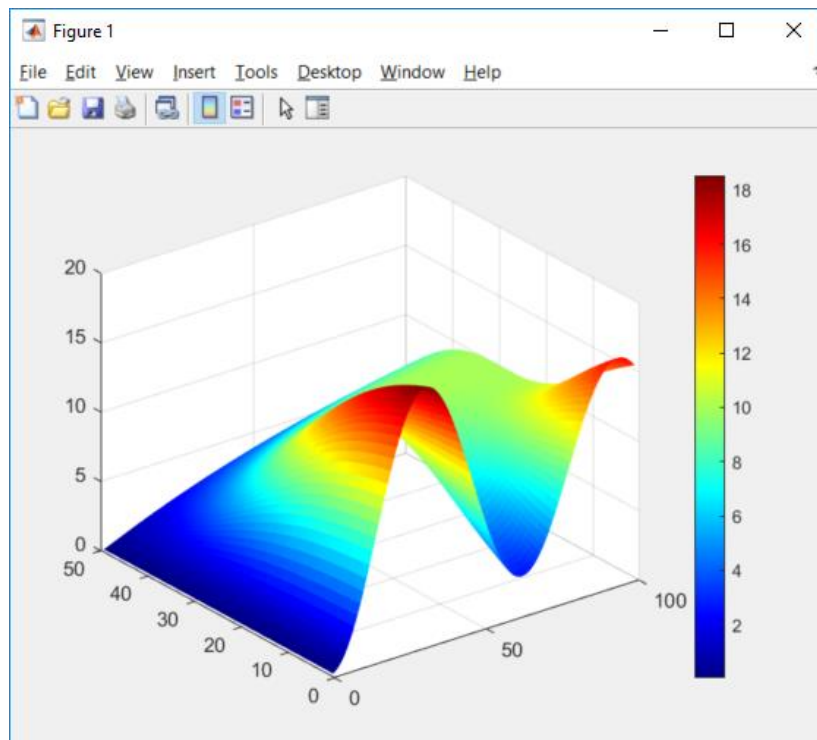
Now, all the data of the new calculated interval between  $t = 6s$  and  $t = 9s$  can be retrieved using interpolation.



## Using DLL with MATLAB

When using NL5 DLL with MATLAB, please use header file `n15_dll.h` located in the MATLAB folder of NL5 DLL download package. Due to the way MATLAB is handling Windows DLLs, all `extern "C"` declarations must be removed from the header file.

A simple example of the MATLAB code `d11_example.m` can be found in the MATLAB folder of NL5 DLL download package. It opens schematic file `d11_example.n15`, changes value of `R1` is specified range, runs transient for each `R1` value, reads transient data of trace `V(out)`, and displays results as a 2-D surface:



Please note that you may need to change path of DLL and header file in function `loadlibrary`:

```
loadlibrary('Your_Path\\n15_dll_64.dll', 'Your_Path\\n15_dll.h');
```

and path of the schematic file in the function `calllib` which calls DLL function `NL5_Open`:

```
is = calllib('n15_dll_64', 'NL5_Open', 'Your_Path\\d11_example.n15');
```

Here is the code:

```
clear;
clc;
close all;
R=logspace(-1,1,50);

% load library
loadlibrary('nl5_dll_64.dll', 'nl5_dll.h');

% open schematic
is = calllib('nl5_dll_64', 'NL5_Open', 'dll_example.nl5');
calllib('nl5_dll_64', 'NL5_GetError');

% get trace handle
it = calllib('nl5_dll_64', 'NL5_GetTrace', is, 'V(out)');

% create pointers to data
pd = libpointer('doublePtr', 0.0);

for k=1:50

    % set R1 value
    calllib('nl5_dll_64', 'NL5_SetValue', is, 'R1', R(k));

    % simulate for 10 s
    calllib('nl5_dll_64', 'NL5_Start', is);
    calllib('nl5_dll_64', 'NL5_Simulate', is, 10.0);

    % read data
    for i=1:100
        t = i*0.1;
        calllib('nl5_dll_64', 'NL5_GetData', is, it, t, pd);
        Z(k,i)=pd.value;
    end
end

% close document
calllib('nl5_dll_64', 'NL5_Close', is);
calllib('nl5_dll_64', 'NL5_GetError');

% unload library
unloadlibrary 'nl5_dll_64';

[X,Y] = meshgrid(1:100,1:50);
surf(X,Y,Z);
shading flat;
colormap jet;
colorbar;
ylim([0 50]);
```

## Using DLL with SystemVerilog

NL5 DLL can be used for co-simulation with SystemVerilog digital simulators, where DLL functions are being called through DPI – Direct Programming Interface.

### Files

The following files can be used for interfacing DLL with SystemVerilog DPI:

- `n15_dll_32.lib` : library file for 32-bit OS
- `n15_dll_64.lib` : library file for 64-bit OS
- `n15_dll_32.dll` : DLL file for 32-bit OS
- `n15_dll_64.dll` : DLL file for 64-bit OS
- `n15_sv.svh` : header file for SystemVerilog code
- `n15_sv.c` : “wrapper” C-code
- `svdpi.h` : header file for “wrapper” C-code

### Using DLL

To use DLL with SystemVerilog code, link the project with appropriate DLL library file (32 or 64 bit), and place appropriate NL5 DLL file into the directory where it can be accessed. Also, include `n15_sv.svh` header file into Verilog code. This file contains prototypes of DLL functions.

Refer to the documentation of your SystemVerilog simulation tool for details on creating the project, and using DPI.

## Using DLL with C-code “wrapper”

If DLL library file cannot be linked to the SystemVerilog project for any reason, NL5 DLL can be accessed using provided “wrapper” C-code `nl5_sv.c`. Compile and link that code to the SystemVerilog project. Please note that different tools may require their own specific header file `svdpi.h`. Refer to the documentation of your SystemVerilog simulation tool for details on creating the project and using DPI.

Include `nl5_sv.svh` header file into SystemVerilog code: this file contains prototypes of DLL functions.

Place appropriate DLL file (32 or 64 bit) into the directory where it can be accessed. Before calling any DLL functions first time, DLL should be loaded into memory by calling `NL5_OpenDLL` function with appropriate `dll` file name as a parameter. The function returns 0 if successful, or negative error code if failed. The following error codes are currently used:

```
int result = NL5_OpenDLL("nl5_dll_32.dll");
if(result == -1)
{
    // DLL not found. Handle the error here
    . . .
}
else if(result == -2)
{
    // Some DLL functions not found. Handle the error here
    . . .
}
else if(result == -3)
{
    // DLL already loaded. Handle the error here
    . . .
}
else
{
    // OK
}
```

Once DLL is successfully loaded, all DLL functions can be called.

# Running co-simulation demo with Xilinx Vivado

## Creating demo project

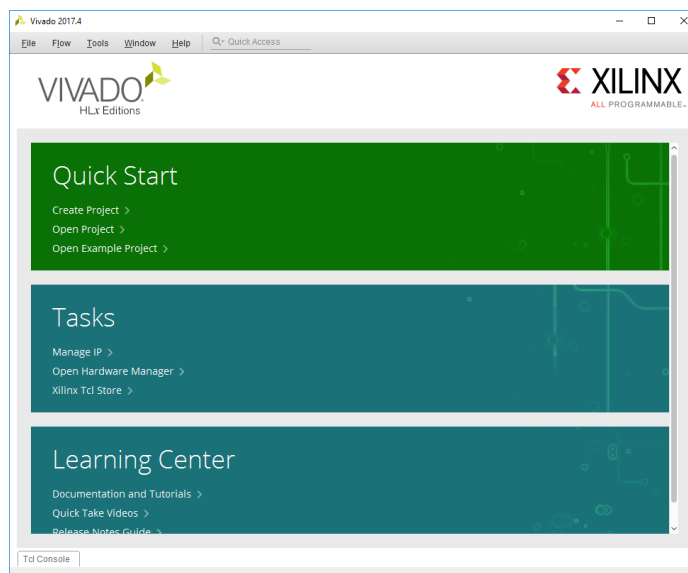
There are many ways of creating and configuring Vivado project. Please refer to Vivado Manual, or use public on-line tutorials on Vivado for more information.

For this instructions, Vivado HLx Edition, v2017.4 (64-bit) was used. For 32-bit version, the following modifications should be made:

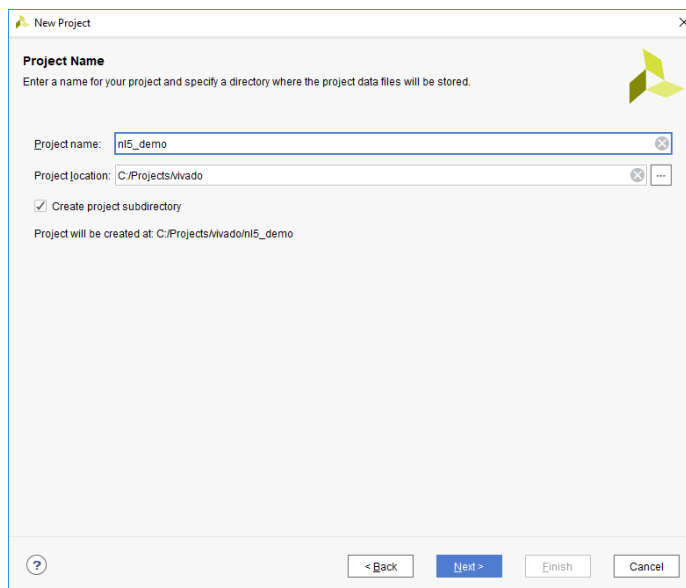
1. Instead of `n15_dll_64.dll`, use `n15_dll_32.dll`. It can be found in the NL5 DLL installation package.
2. In the `n15_demo.sv` source file, change the line

```
ret = NL5_OpenDLL("n15_dll_64.dll");  
to  
ret = NL5_OpenDLL("n15_dll_32.dll");
```

To create a new project, open Vivado:

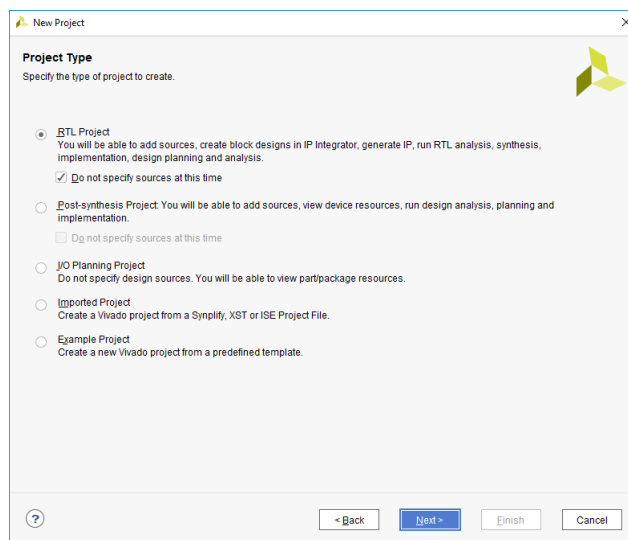


Select “Quick Start” / “Create Project”, Click “Next”



The "New Project" dialog box is shown. It has a title bar with a yellow Vivado logo and a close button. The main area is titled "Project Name" and contains the instruction "Enter a name for your project and specify a directory where the project data files will be stored." Below this, there are two input fields: "Project name:" with the text "nl5\_demo" and "Project location:" with the text "C:/Projects/Nvado". To the right of the location field is a browse button. Below these fields is a checkbox labeled "Create project subdirectory" which is checked. Underneath the checkbox, it says "Project will be created at: C:/Projects/Nvado/nl5\_demo". At the bottom of the dialog are four buttons: a help button (question mark), "< Back", "Next >" (highlighted in blue), "Finish", and "Cancel".

Project name: enter project name (“nl5\_demo”), click “Next”:

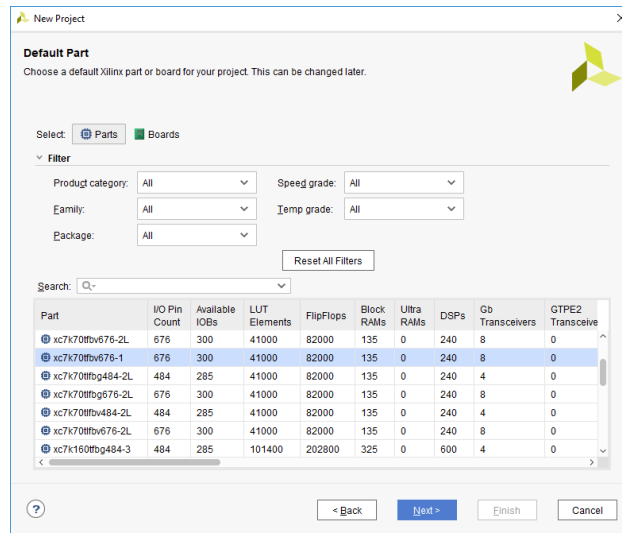


The "New Project" dialog box is shown, now on the "Project Type" step. The title bar is the same. The main area is titled "Project Type" and contains the instruction "Specify the type of project to create." Below this are five radio button options, each with a description and a checkbox for "Do not specify sources at this time":

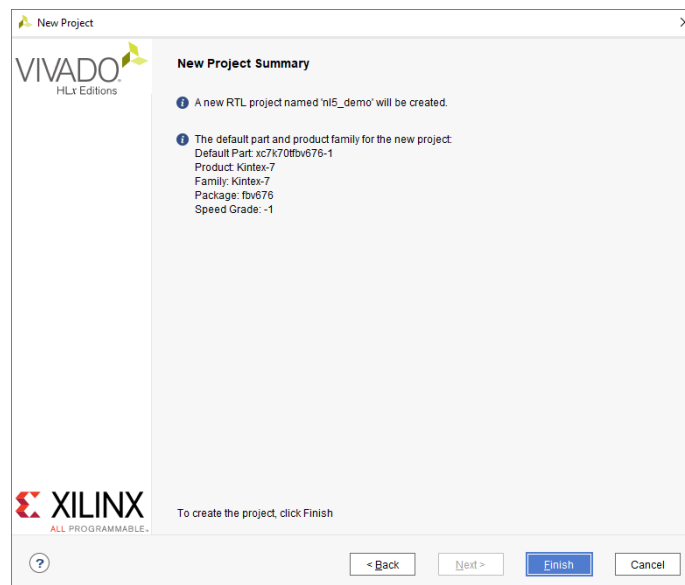
- RTL Project**: You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis. (Checked)
- Post-synthesis Project**: You will be able to add sources, view device resources, run design analysis, planning and implementation. (Not checked)
- I/O Planning Project**: Do not specify design sources. You will be able to view part/package resources. (Not checked)
- Imported Project**: Create a Vivado project from a Synplify, XIST or ISE Project File. (Not checked)
- Example Project**: Create a new Vivado project from a predefined template. (Not checked)

At the bottom of the dialog are four buttons: a help button (question mark), "< Back", "Next >" (highlighted in blue), "Finish", and "Cancel".

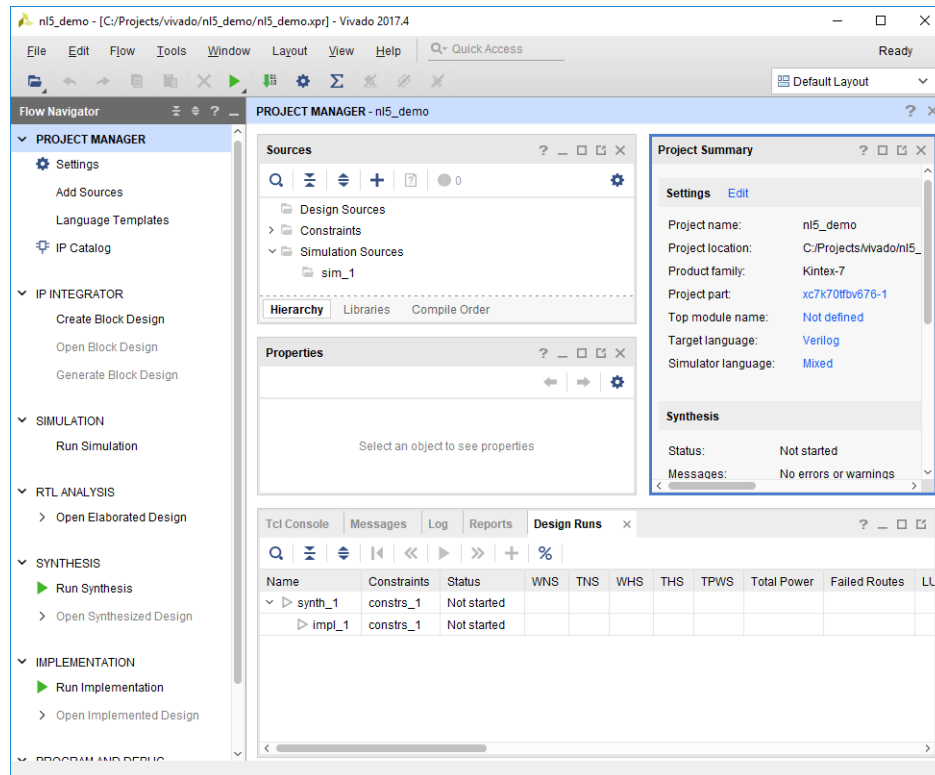
Project type: click “Next”:



Default part: please note that list of parts will depend on your installation. Select Xilinx part or board, click “Next”:



Click “Finish”



The project has been created, project directory is:

C:\Projects\vivado\nl5\_demo



## Creating library file

To create library file `dpi.a`, copy the following files from `SystemVerilog` directory of the NL5 DLL installation package to Vivado temporary directory

```
C:\Users\<UserName>\AppData\Roaming\Xilinx\Vivado
```

```
nl5_sv.c
```

```
svdpi.h
```

In the Vivado Tcl Console command line, type:

```
xsc nl5_sv.c
```

For running NL5 DLL demo, copy new `dpi.a` file from

```
C:\Users\<UserName>\AppData\Roaming\Xilinx\Vivado
```

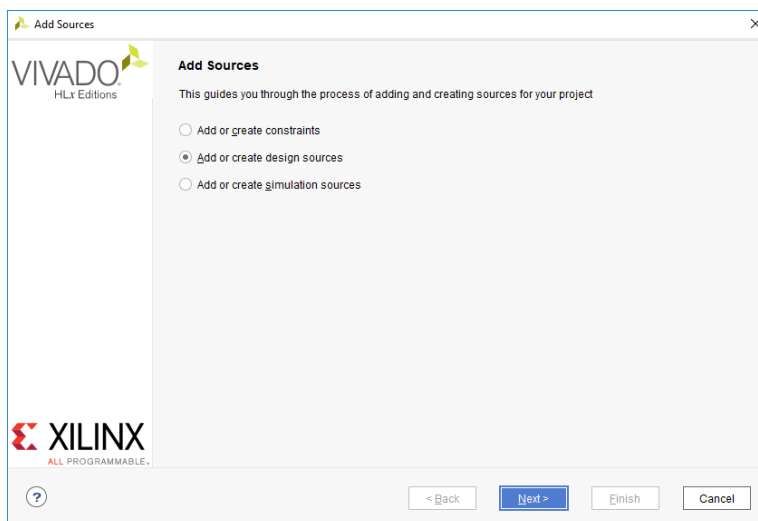
to `C:\Projects\vivado\nl5_demo\nl5_demo.sim\sim_1\behav\xsim`  
as described in the next section.

## Configuring and running demo

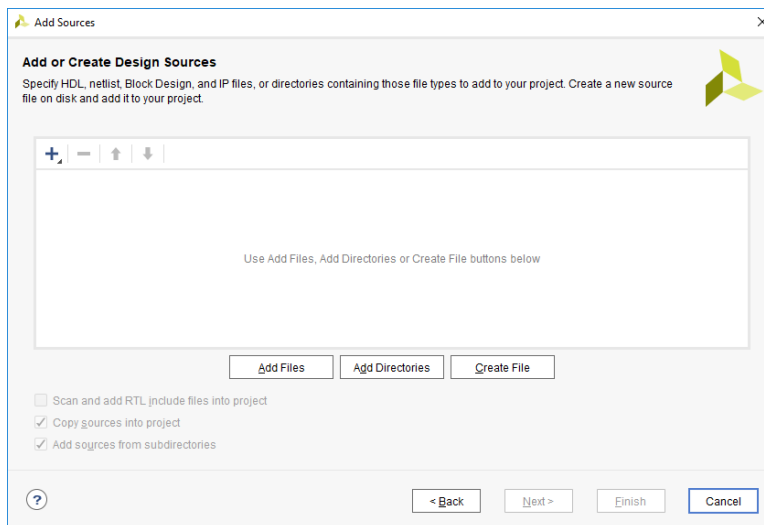
In the NL5 DLL installation package, go to `SystemVerilog\Vivado\src` directory, and copy the following files into project directory `C:\Projects\vivado\nl5_demo`

```
nl5_demo.sv
nl5_sv.svh
```

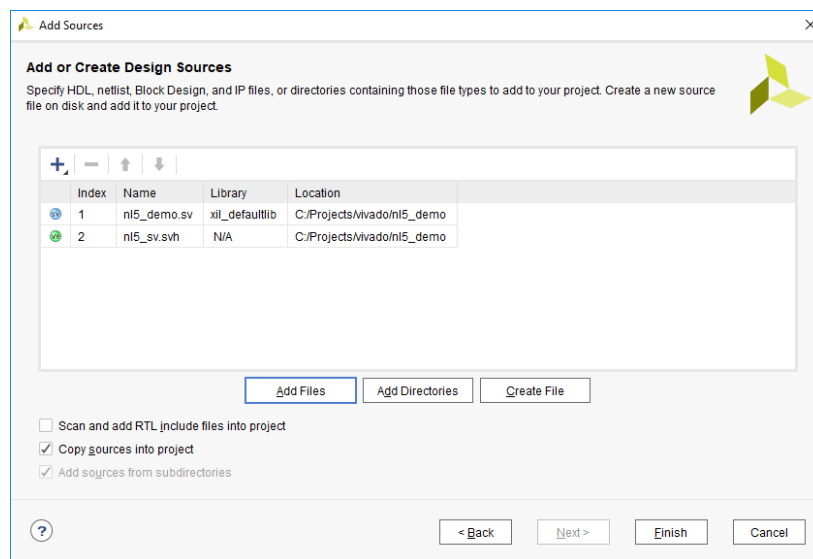
Select “Project manager” / “Add Sources”:



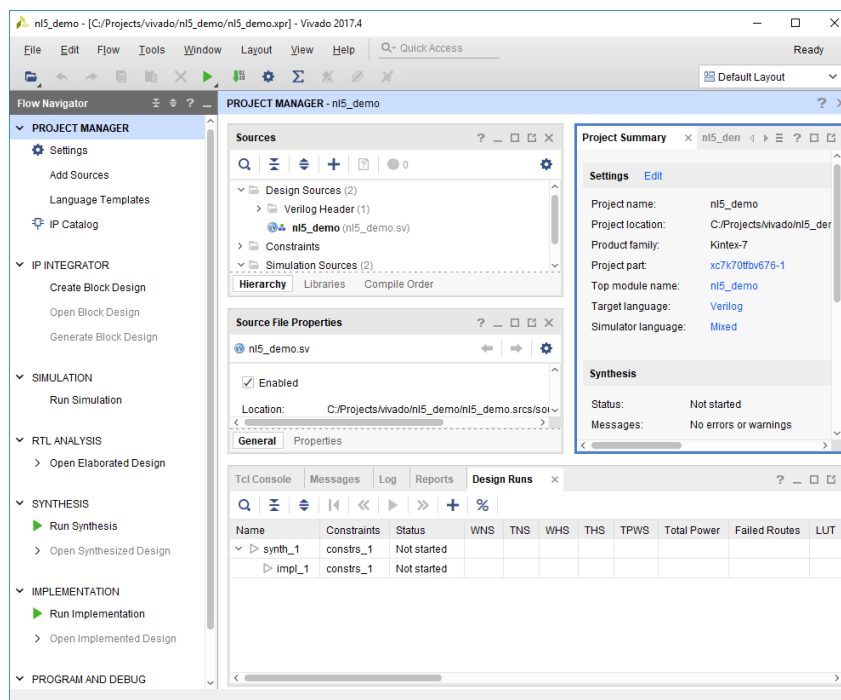
Add Sources: select “Add or create design source”, click “Next”:



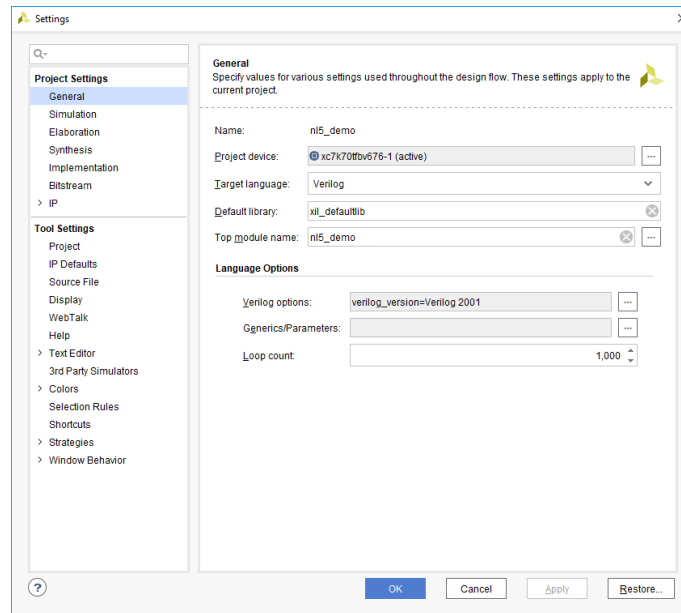
Add or Create Design Sources: click Add Files, select `C:\Projects\vivado\nl5_demo` directory, select `nl5_demo.sv` and `nl5_sv.svh` files (using Ctrl key), click “OK”:



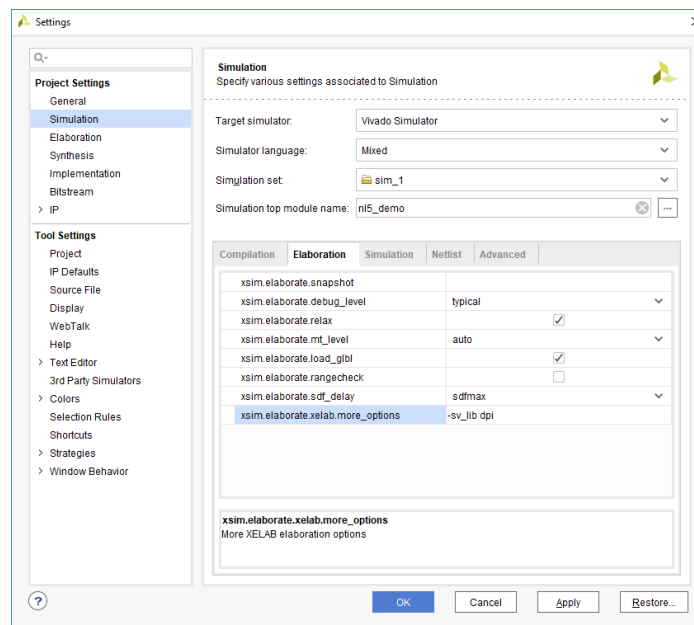
Click “Finish”:



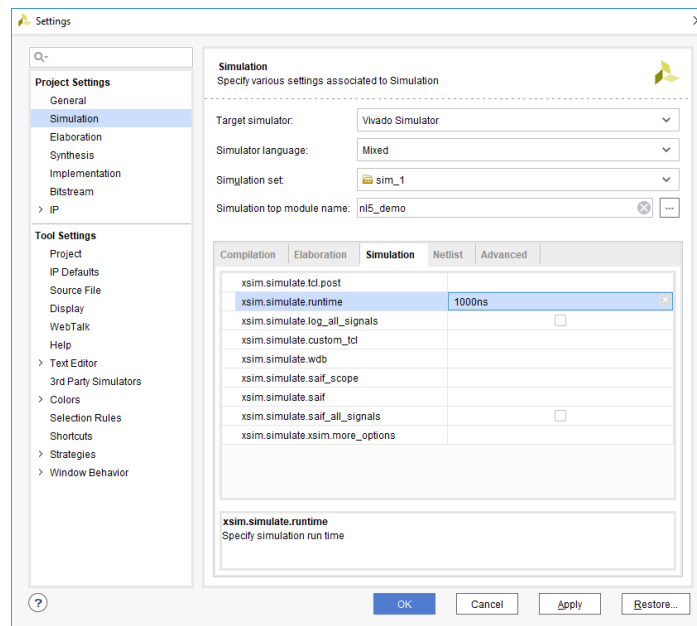
Select “Project manager” / “Settings”:



Select “Project Settings” / “Simulation”, “Elaboration” tab, enter:  
`xsim.elaborate.xelab.more_options = -sv_lib dpi`

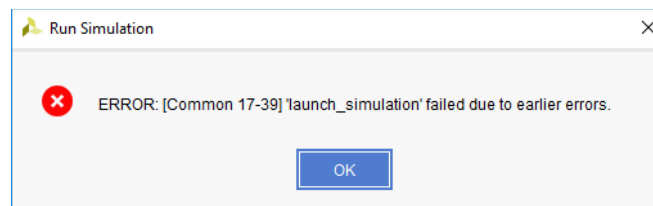


Select “Simulation” tab, enter:  
`xsim.simulation.runtime = 1000ns`



Click “OK”

Select “Project Manager” / “Simulation” / “Run Simulation” / Run Behavioral Simulation”. An error message will pop up:



Click “OK” two times. This step is required in order to force Vivado to create simulation directory, and then copy required nl5 demo files into that directory.

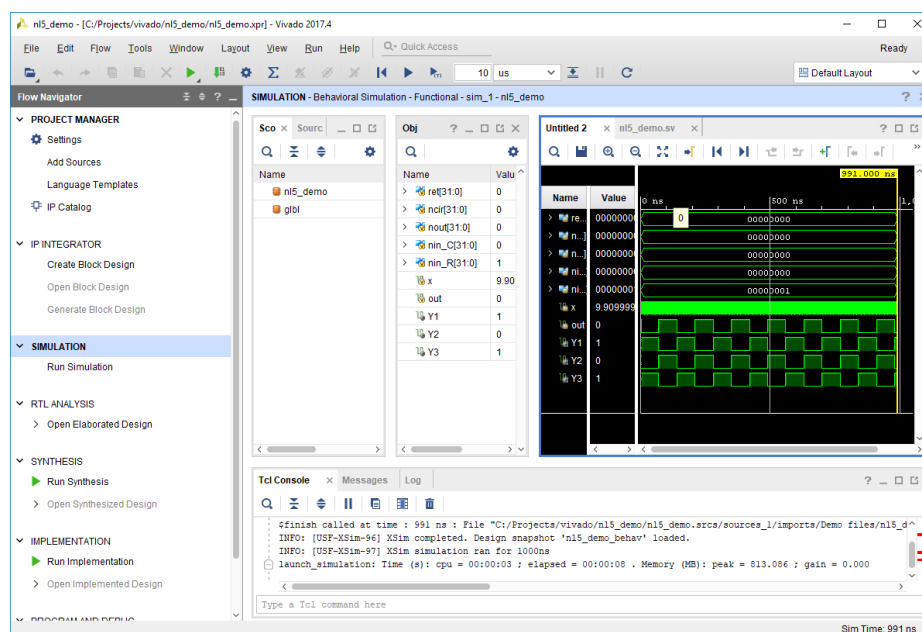
In the NL5 DLL installation package, go to `SystemVerilog\Vivado\sim` directory, and copy the following files into simulation directory

`C:\Projects\vivado\nl5_demo\nl5_demo.sim\sim_1\behave\xsim`

```
n15_dll_64.dll
rc.n15
```

Also, copy library file `dpi.a`, as described in “Creating library file” section.

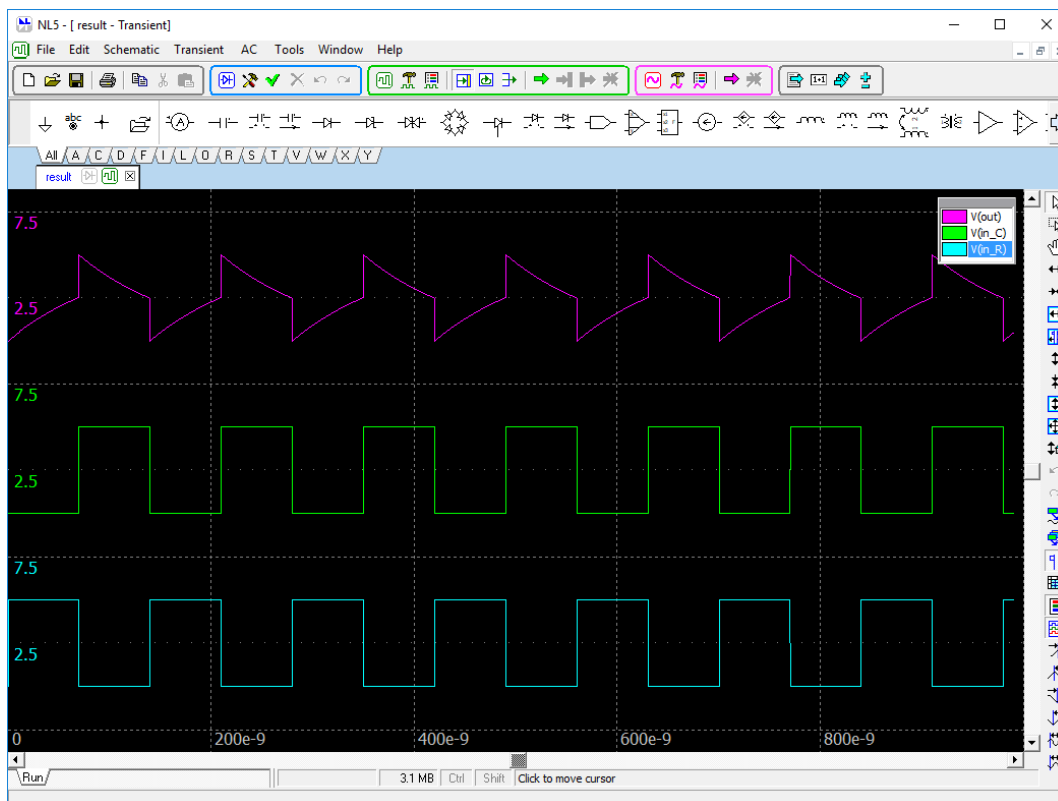
Select “Project Manager” / “Simulation” / “Run Simulation” / Run Behavioral Simulation”.  
After successful simulation, the results will be shown in the Waveform Window:



To see analog waveforms of the simulation, start NL5 Circuit Simulator, open nl5 file with simulation results

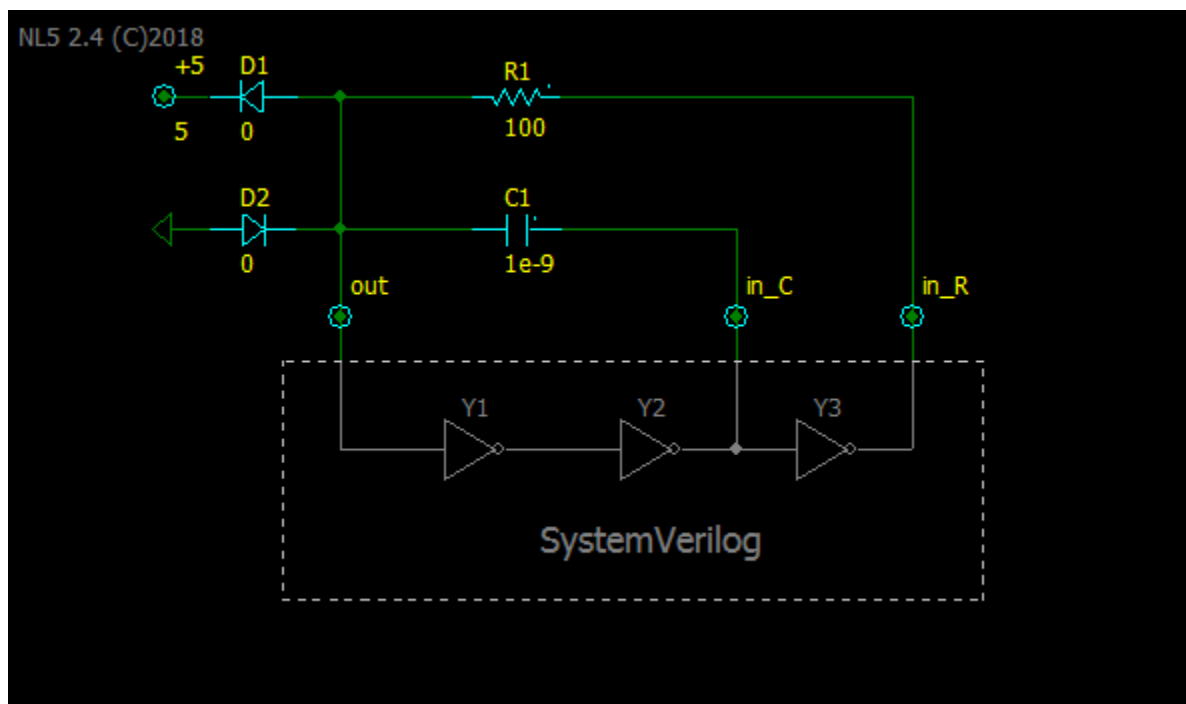
C:\Projects\vivado\nl5\_demo\nl5\_demo.sim\sim\_1\behav\xsim\result.nl5

and open transient window:



## Demo circuit

A simple oscillator circuit with 3 inverters is used as a demo:



Digital part (Y1, Y2, Y3) of the circuit is disabled, since it will be simulated by SystemVerilog. Labels “out”, “in\_C”, and “in\_R” are used for passing signals between analog and digital parts.

When SystemVerilog simulation is completed, the schematic is saved into the file `result.nl5` along with transient results. Start NL5 Circuit Simulator, and open `result.nl5` to see analog waveforms in details.

To run simulation with NL5 Circuit Simulator, enable digital part of the schematic, and run transient. To enable/disable schematic, select part of the schematic, right-click on the selection, select “Enable” or “Disable” from context menu.

## III. DLL Functions





## NL5\_GetError

### Prototype:

```
char* NL5_GetError()
```

### Parameters:

No parameters

### Returns:

Pointer to null-terminated ASCII character string

### Description:

Returns text description of last execution error. If no error, returns "OK".

The content of the string is valid only until execution of the next DLL function. If the text is needed for the future use, it is user's responsibility to copy it to safe location.

## NL5\_GetInfo

### Prototype:

```
char* NL5_GetInfo()
```

### Parameters:

No parameters

### Returns:

Pointer to null-terminated ASCII character string

### Description

Returns information about DLL, such as version and date.

The content of the string is valid only until execution of the next DLL function. If the text is needed for the future use, it is user's responsibility to copy it to safe location.

## NL5\_GetLicense

### Prototype:

```
int NL5_GetLicense()
```

### Parameters:

No parameters

### Returns:

0 : license valid

### Description

The function is obsolete, and will be removed from the future versions of NL5 DLL.

## NL5\_Open

### Prototype:

```
int NL5_Open(char* name)
```

### Parameters:

*char\** name – pointer to null-terminated ASCII character string with NL5 schematic file name

### Returns:

>=0 : circuit handle  
-1 : error

### Description

Opens NL5 schematic file “name”.

Returns non-negative circuit handle, or -1 if file not found, cannot be open for any reason, or file and is not DLL-enabled and contains too many components.

Circuit handle can be used as input parameter *ncir* for other DLL functions.

If file name does not have path specified, DLL will search for the file in the same directory where NL5 DLL is located.

## NL5\_Close

### Prototype:

```
int NL5_Close(int ncir)
```

### Parameters:

`int ncir` - circuit handle

### Returns:

0 : OK  
-1 : error

### Description

Close schematic with handle `ncir`. Schematic information will be removed from DLL, handle `ncir` cannot be used anymore.

## NL5\_Save

### Prototype:

```
int NL5_Save(int ncir)
```

### Parameters:

```
int ncir - circuit handle
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Save schematic with handle `ncir` into the same file.

Use this function to save schematic back to NL5 schematic file. You might want to save the schematic if any modification of component parameters were made, IC (Initial Conditions) were saved, or if you want to save schematic with transient data (simulation data traces).

To save schematic with transient data, make sure the "Save with transient data" option is set in the schematic file. To set the option, open schematic file in NL5, go to File/Properties/Save, select "Save with transient data" checkbox, and save schematic into the file.

## NL5\_SaveAs

### Prototype:

```
int NL5_Save(int ncir, char* name)
```

### Parameters:

```
int    ncir    - circuit handle
char*  name    - pointer to null-terminated ASCII character string with NL5 schematic file name
```

### Returns:

```
0   : OK
-1  : error
```

### Description

Save schematic with handle `ncir` into a new schematic file.

Use this function to save schematic into a new NL5 schematic file. You might want to save the schematic if any modification of component parameters were made, IC (Initial Conditions) were saved, or if you want to save schematic with transient data (simulation data traces).

To save schematic with transient data, make sure the "Save with transient data" option is set in the schematic file. To set the option, open schematic file in NL5, go to File/Properties/Save, select "Save with transient data" checkbox, and save schematic into the file.

## NL5\_GetValue

### Prototype:

```
int NL5_GetValue(int ncir, char* name, double* v)
```

### Parameters:

```
int      ncir  - circuit handle
char*    name  - pointer to null-terminated ASCII character string with parameter name
double*  v     - pointer to value variable
```

### Returns:

```
0   : OK
-1  : error
```

### Description

Returns `double` value of component parameter.

`name` is component parameter name in the format <component>.<parameter> ("R1.R", "V1.V").  
See NL5 Circuit Simulator Manual for details (User Interface/Data format/Names).

Returns -1 if parameter not found, or parameter type is not supported.

Depending on parameter type, the following value is returned:

- formula : number in `double` format
- Initial Condition : number in `double` format if not blank, not supported if blank
- "On/Off" : 1 for "On", 0 for "Off"
- "High/Low" : 1 for "High", 0 for "Low"
- "Yes/No" : 1 for "Yes", 0 for "No"
- text list : parameter number in the list (zero based)

Other parameter types are not supported.



## NL5\_SetValue

### Prototype:

```
int NL5_SetValue(int ncir, char* name, double v)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with parameter name
<i>double</i>	<i>v</i>	- parameter value

### Returns:

0	: OK
-1	: error

### Description

Sets value of parameter to *v*.

*name* is component parameter name in the format <component>.<parameter> ("R1.R", "V1.V").  
See NL5 Circuit Simulator Manual for details (User Interface/Data format/Names).

Returns -1 if parameter not found, or parameter type is not supported.

Depending on parameter type, number *v* is interpreted as follows:

- formula : number in double format
- Initial Condition : number in double format
- "On/Off" : 1 for "On", 0 for "Off"
- "High/Low" : 1 for "High", 0 for "Low"
- "Yes/No" : 1 for "Yes", 0 for "No"
- text list : parameter number in the list (zero based)

Other parameter types are not supported.

## NL5\_GetText

### Prototype:

```
int NL5_GetText(int ncir, char* name, char* text, int length)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with parameter name
<i>char*</i>	<i>text</i>	- pointer to null-terminated ASCII character string with parameter text
<i>int</i>	<i>length</i>	- max number of characters allowed to return into <i>text</i> , including trailing null

### Returns:

>=0 : number of characters returned into *text*, including trailing null.  
-1 : error

### Description

Returns *text* (parameter value in text format) of component parameter into character string *text*.

*name* is component parameter name in the format <component>.<parameter> ("R1.R", "V1.V").  
See NL5 Circuit Simulator Manual for details (User Interface/Data format/Names).

Size of character string *text* should be not less than *length*.

Returns -1 if parameter not found, or parameter type is not supported.

Practically all parameter types are supported. The text returned is the same as displayed in the components window of NL5 Circuit Simulator.

If parameter is defined as a formula, text of the formula will be returned.

## NL5\_SetText

### Prototype:

```
int NL5_SetText(int ncir, char* name, char* text)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with parameter name
<i>char*</i>	<i>text</i>	- pointer to null-terminated ASCII character string with parameter <i>text</i>

### Returns:

0 : OK  
-1 : error

### Description

Sets text of component parameter *name* to *text*.

*name* is component parameter name in the format <component>.<parameter> ("R1.R", "V1.V").  
See NL5 Circuit Simulator Manual for details (User Interface/Data format/Names).

Returns -1 if parameter not found, or parameter type is not supported.

Practically all parameter types are supported. The text provided is expected to be the same as displayed in the components window of NL5 Circuit Simulator.

To enter a formula for parameter of "formula" type, provide text of the formula started with equal sign '='.

## NL5\_GetParam

### Prototype:

```
int NL5_GetParam(int ncir, char* name)
```

### Parameters:

```
int      ncir  - circuit handle
char*    name  - pointer to null-terminated ASCII character string with parameter name
```

### Returns:

```
>=0 : parameter handle
-1   : error
```

### Description

`name` is component parameter name in the format <component>.<parameter> ("R1.R", "V1.V").  
See NL5 Circuit Simulator Manual for details (User Interface/Data format/Names).

Returns non-negative handle of component parameter, or -1 if parameter not found.

## NL5\_GetParamValue

### Prototype:

```
int NL5_GetParamValue(int ncir, int npar, double* v)
```

### Parameters:

```
int      ncir  - circuit handle
int      npar  - parameter handle
double* v     - pointer to the variable
```

### Returns:

```
0   : OK
-1  : error
```

### Description

Returns `double` value of parameter with handle `npar` into variable `v`. Parameter handle `npar` should be obtained by function `NL5_GetParam`.

Returns -1 if parameter handle `npar` is not valid, or parameter type is not supported.

Depending on parameter type, the following value is returned:

- formula : number in `double` format
- Initial Condition : number in `double` format if not blank, not supported if blank
- "On/Off" : 1 for "On", 0 for "Off"
- "High/Low" : 1 for "High", 0 for "Low"
- "Yes/No" : 1 for "Yes", 0 for "No"
- text list : parameter number in the list (zero based)

Other parameter types are not supported.

## NL5\_SetParamValue

### Prototype:

```
int NL5_SetParamValue(int ncir, int npar, double v)
```

### Parameters:

```
int      ncir  - circuit handle
int      npar  - parameter handle
double   v     - parameter value
```

### Returns:

```
0   : OK
-1  : error
```

### Description

Sets value of parameter with handle `npar` to `v`. Parameter handle `npar` should be obtained by function `NL5_GetParam`.

Returns -1 if parameter handle `npar` is not valid, or parameter type is not supported.

Depending on parameter type, number `v` is interpreted as follows:

- formula : number in double format
- Initial Condition : number in double format
- "On/Off" : 1 for "On", 0 for "Off"
- "High/Low" : 1 for "High", 0 for "Low"
- "Yes/No" : 1 for "Yes", 0 for "No"
- text list : parameter number in the list (zero based)

Other parameter types are not supported.

## NL5\_GetParamText

### Prototype:

```
int NL5_GetParamText(int ncir, int npar, char* text, int length)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>int</i>	<i>npar</i>	- parameter handle
<i>char*</i>	<i>text</i>	- pointer to null-terminated ASCII character string with parameter <i>text</i>
<i>int</i>	<i>length</i>	- max number of characters allowed to return into <i>text</i> , including trailing null

### Returns:

*>=0* : number of characters returned into *text*, including trailing null.  
*-1* : error

### Description

Copies text (parameter value in text format) of component parameter with handle *npar* into character string *text*.

Parameter handle *npar* should be obtained by function `NL5_GetParam`.

Size of character string *text* should be not less than *length*.

Returns -1 if parameter handle *npar* is not valid, or parameter type is not supported.

Practically all parameter types are supported. The text returned is the same as displayed in the components window of NL5 Circuit Simulator.

If parameter is defined as a formula, text of the formula will be returned.

## NL5\_SetParamText

### Prototype:

```
int NL5_SetParamText(int ncir, int npar, char* text)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>int</i>	<i>npar</i>	- parameter handle
<i>char*</i>	<i>text</i>	- pointer to null-terminated ASCII character string with parameter <i>text</i>

### Returns:

0	: OK
-1	: error

### Description

Sets text of component parameter with handle *npar* to *text*. Parameter handle *npar* should be obtained by function `NL5_GetParam`.

Returns -1 if parameter handle *npar* is not valid, or parameter type is not supported.

Practically all parameter types are supported. The text provided is expected to be the same as displayed in the components window of NL5 Circuit Simulator.

To enter a formula for parameter of “formula” type, provide text of the formula started with equal sign ‘=’.



## NL5\_GetTrace

### Prototype:

```
int NL5_GetTrace(int ncir, char* name)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with trace name

### Returns:

$\geq 0$	: trace handle
-1	: error

### Description

*name* is the trace name in the format used by NL5 Circuit Simulator. See NL5 Circuit Simulator Manual for details (User Interface/Data format/Names/Trace).

Returns non-negative trace handle, or -1 if trace *name* not found.

## NL5\_AddVTrace

### Prototype:

```
int NL5_AddVTrace(int ncir, char* name)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with component <i>name</i>

### Returns:

$\geq 0$	: trace handle
-1	: error

### Description

Creates voltage trace for component *name*.

Returns non-negative trace handle, or -1 if component *name* not found, or voltage trace is not supported by the component.

## NL5\_AddITrace

### Prototype:

```
int NL5_AddITrace(int ncir, char* name)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with component <i>name</i>

### Returns:

$\geq 0$	: trace handle
-1	: error

### Description

Creates current trace for component *name*.

Returns non-negative trace handle, or -1 if component *name* not found, or current trace is not supported by the component.

## NL5\_AddPTrace

### Prototype:

```
int NL5_AddPTrace(int ncir, char* name)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with component <i>name</i>

### Returns:

$\geq 0$	: trace handle
-1	: error

### Description

Creates power trace for component *name*.

Returns non-negative trace handle, or -1 if component *name* not found, or power trace is not supported by the component.

## NL5\_AddVarTrace

### Prototype:

```
int NL5_AddVarTrace(int ncir, char* name)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with schematic variable <i>name</i>

### Returns:

$\geq 0$	: trace handle
-1	: error

### Description

Creates trace for schematic variable *name*.

Returns non-negative trace handle, or -1 if variable *name* not found.

## NL5\_AddFuncTrace

### Prototype:

```
int NL5_AddFuncTrace(int ncir, char* text)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>text</i>	- pointer to null-terminated ASCII character string with function text

### Returns:

$\geq 0$	: trace handle
-1	: error

### Description

Creates trace of function *text*. See NL5 Circuit Simulator Manual for details on function trace (Transient Analysis/Transient Data/Traces/Function trace).

Returns non-negative trace handle, or -1 if error occurred.

## NL5\_DeleteTrace

### Prototype:

```
int NL5_DeleteTrace(int ncir, int ntrace)
```

### Parameters:

```
int  ncir  - circuit handle  
int  ntrace - trace handle
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Deletes traces with trace handle `ntrace`.

## NL5\_GetInput

### Prototype:

```
int NL5_GetInput(int ncir, char* name)
```

### Parameters:

```
int      ncir  - circuit handle
char*    name  - pointer to null-terminated ASCII character string with component name
```

### Returns:

```
>=0 : input handle
-1   : error
```

### Description

`name` is component name.

The following component types are supported:

- Label
- Voltage source
- Current source

Returns non-negative input handle or -1 if component not found, or is not supported as an input. The model of the component will be automatically changed to "V" (constant voltage source) or "I" (constant current source).



## NL5\_SetInputValue

### Prototype:

```
int NL5_SetInputValue(int ncir, int nin, double v)
```

### Parameters:

<i>int</i>	ncir	- circuit handle
<i>int</i>	nin	- input handle
<i>double</i>	v	- parameter value

### Returns:

0	: OK
-1	: error

### Description

Sets voltage or current of the input with handle `npar` to `v`. Input handle `nin` should be obtained by function `NL5_GetInput`.

Returns -1 if input handle `nin` is not valid.

## NL5\_GetOutput

### Prototype:

```
int NL5_GetOutput(int ncir, char* name)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>char*</i>	<i>name</i>	- pointer to null-terminated ASCII character string with component name

### Returns:

$\geq 0$	: input handle
-1	: error

### Description

*name* is component name. Only Label component type is supported.

Returns non-negative output handle or -1 if component not found, or is not supported as an output. The model of the label will be automatically changed to 'Label'.

## NL5\_GetOutputValue

### Prototype:

```
int NL5_GetOutputValue(int ncir, int nout, double* v)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>int</i>	<i>nout</i>	- output handle
<i>double*</i>	<i>v</i>	- pointer to the variable

### Returns:

0	: OK
-1	: error

### Description

Sets `double` value of voltage of output with handle `nout` into variable `v`. Output handle `nout` should be obtained by function `NL5_GetOutput`.

Returns -1 if output handle `nout` is not valid.

## NL5\_SetStep

### Prototype:

```
int NL5_SetStep(int ncir, double step)
```

### Parameters:

```
int      ncir  - circuit handle  
double step  - calculation step
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Sets maximum calculation step size. If this function was not called, an original calculation step from schematic file will be used (Transient/Settings/"Calculation step").

## NL5\_SetTimeout

### Prototype:

```
int NL5_SetTimeout(int ncir, int t)
```

### Parameters:

```
int    ncir  - circuit handle  
int    t     - time-out, seconds
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Sets maximum time allowed for calculating one simulation step. If this function was not called, a default time-out value is used (0). If time-out is equal to zero, time-out detection is disabled. If time-out occurred due to unresolved switching iterations, the error message will indicate a component which started switching process. Time-out may also occur due to infinite while/do/for loops of C-code.

## NL5\_GetSimulationTime

### Prototype:

```
int NL5_GetSimulationTime(int ncir, double* t)
```

### Parameters:

```
int      ncir  - circuit handle  
double* t      - pointer to time variable
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Sets `t` to the current value of internal `simulation_time` variable.

## NL5\_Start

### Prototype:

```
int NL5_Start(int ncir)
```

### Parameters:

```
int ncir - circuit handle
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Start simulation.

The function resets internal `simulation_time` variable to 0, initializes circuit components, erases existing simulation data, and calculates initial state of the circuit according to specified Initial Conditions. When function returns, the simulation data consists of circuit state at  $t=0$ .

The function should be called first to start simulation from  $t=0$ , prior to calling any simulation functions. However, calling `NL5_Start` is not required. It will be executed automatically if any of simulation functions is called, and simulation has not been performed yet.

The function may return error code if not-DLL enabled schematic contains too many components after loading subcircuits.

## NL5\_Simulate

### Prototype:

```
int NL5_Simulate(int ncir, double interval)
```

### Parameters:

```
int    ncir      - circuit handle  
double interval  - time interval to simulate, in seconds
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Performs transient simulation at least for requested `interval`, starting from current `simulation_time`.

The function increments current `simulation_time` variable by requested `interval`. Then it performs simulation until the time of calculated data is equal or greater than new `simulation_time`.

The function does not decrease simulation step in order to stop exactly at the end of requested `interval` (new `simulation_time`). ), so the time of the last calculated data may exceed new `simulation_time`.

When next simulation function is called, simulation will be continued with simulation step equal to the last simulation step.

The function may return error code if not-DLL enabled schematic contains too many components after loading subcircuits.



## NL5\_SimulateInterval

### Prototype:

```
int NL5_SimulateInterval(int ncir, double interval)
```

### Parameters:

```
int    ncir      - circuit handle  
double interval  - time interval to simulate, in seconds
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Performs transient simulation exactly for requested `interval`, starting from current `simulation_time`.

The function increments current `simulation_time` variable by requested `interval`. Then it performs simulation until the time of calculated data is equal to the new `simulation_time`.

The function may decrease simulation step in order to stop exactly at the end of requested `interval`.

When next simulation function is called, simulation step will be restored, and a new “linear range” will be started.

The function may return error code if not-DLL enabled schematic contains too many components after loading subcircuits.

## NL5\_SimulateStep

### Prototype:

```
int NL5_SimulateStep(int ncir)
```

### Parameters:

`int ncir` - circuit handle

### Returns:

0 : OK  
-1 : error

### Description

Performs one step of transient simulation.

In some situations (the very beginning of simulation, switching, change of linear range, etc.) more than one simulation step should be executed without interruption. In this case, the function will execute more than one simulation step.

When the function returns, `simulation_time` variable is set to the time of last calculated data.

The function may return error code if not-DLL enabled schematic contains too many components after loading subcircuits.

## NL5\_SaveIC

### Prototype:

```
int NL5_SaveIC(int ncir)
```

### Parameters:

```
int ncir - circuit handle
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Saves current component states into components' Initial Conditions.

The function does not save schematic into schematic file.

## NL5\_GetDataSize

### Prototype:

```
int NL5_GetDataSize(int ncir, int ntrace)
```

### Parameters:

```
int    ncir    - circuit handle  
int    ntrace  - trace handle
```

### Returns:

```
>=0 : data size (number of data points)  
-1  : error
```

### Description

Returns non-negative number of data points of the trace with trace handle `ntrace` or -1 if error occurred. Please note that NL5 Ver.2 algorithm performs automatic data compression, not storing identical data points. As a result, data size of different traces obtained during simulation could be different.

## NL5\_GetDataAt

### Prototype:

```
int NL5_GetDataAt(int ncir, int ntrace, int n, double* t, double* data)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>int</i>	<i>ntrace</i>	- trace handle
<i>int</i>	<i>n</i>	- data point index
<i>double*</i>	<i>t</i>	- pointer to time variable
<i>double*</i>	<i>data</i>	- pointer to value variable

### Returns:

0	: OK
-1	: error

### Description

Returns time and data of data point with index *n*. Data index is zero-based.

Returns -1 if index is less than zero, or greater or equal to data size.

Please note that NL5 Ver.2 algorithm performs automatic data compression, not storing identical data points. As a result, data size of different traces obtained during simulation could be different, and data points with the same index could be taken at different time.

## NL5\_GetLastData

### Prototype:

```
int NL5_GetLastData(int ncir, int ntrace, double* t, double* data)
```

### Parameters:

<i>int</i>	ncir	- circuit handle
<i>int</i>	ntrace	- trace handle
<i>double*</i>	t	- pointer to time variable
<i>double*</i>	data	- pointer to data variable

### Returns:

0	: OK
-1	: error

### Description

Sets `t` and `data` to the time and data value of the last data point.

Returns -1 if there is no data.

## NL5\_GetData

### Prototype:

```
int NL5_GetData(int ncir, int ntrace, double t, double* data)
```

### Parameters:

<i>int</i>	<i>ncir</i>	- circuit handle
<i>int</i>	<i>ntrace</i>	- trace handle
<i>double</i>	<i>t</i>	- time
<i>double*</i>	<i>data</i>	- pointer to data variable

### Returns:

0	: OK
-1	: error

### Description

Sets *data* to the data value at time *t*. The data is calculated as linear interpolation between two data points, with time below and above requested time.

Returns -1 if *t* is less than time of first data point, or greater than the time of last data point.

## NL5\_DeleteOldData

### Prototype:

```
int NL5_DeleteOldData(int ncir)
```

### Parameters:

```
int ncir - circuit handle
```

### Returns:

```
0   : OK  
-1  : error
```

### Description

Deletes transient data with time less than current `simulation_time`. If `simulation_time` falls between two data points, the last data point prior to `simulation_time` remains in the data. This allows obtaining interpolated data at `simulation_time` after simulation continues.



## IV. Attachments



## END USER LICENSE AGREEMENT

This End-User License Agreement ("EULA", "Agreement") is a legal agreement between you ("you", either an individual or a single entity) and Sidelinesoft, LLC ("Sidelinesoft") for the NL5 Circuit Simulator and NL5 DLL software ("the Software", "the Software Product"), NL5 License ("the Software License"), and accompanying documentation.

### Ownership

The Software, any accompanying documentation, and all intellectual property rights therein are owned by Sidelinesoft. The Software is licensed, not sold. The Software is protected by copyright laws and treaties, as well as laws and treaties related to other forms of intellectual property. The Licensee's license to download, use, copy, or change the Software Product is subject to these rights and to all the terms and conditions of this Agreement.

### Acceptance

YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT BY DOWNLOADING THE SOFTWARE PRODUCT OR BY INSTALLING, USING, OR COPYING THE SOFTWARE PRODUCT. YOU MUST AGREE TO ALL OF THE TERMS OF THIS AGREEMENT BEFORE YOU WILL BE ALLOWED TO DOWNLOAD THE SOFTWARE PRODUCT. IF YOU DO NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, YOU MUST NOT INSTALL, USE, OR COPY THE SOFTWARE PRODUCT.

### License Grant

Sidelinesoft grants you a right to download, install, and use unlimited copies of the Software Product. Without a Software License, the Software operates as a Demo version, with limited number of components in the schematic, and possibly some functional and performance limitations. Several types of Full-Function Software Licenses can be obtained at Software Product website ([nl5.sidelinesoft.com](http://nl5.sidelinesoft.com)). Terms and conditions of each type of Full-Function Software License are available at the website and are subject to change without notice.

### Restrictions on Reverse Engineering, Decompilation, and Disassembly.

You may not decompile, reverse-engineer, disassemble, or otherwise attempt to derive the source code for the Software Product.

### Restrictions on Alteration

You may not modify the Software Product or create any derivative work of the Software Product or its accompanying documentation without obtaining permission of Sidelinesoft. Derivative works include but are not limited to translations. You may not alter any files or libraries in any portion of the Software Product.

### Consent to Use of Data

Sidelinesoft may ask for your permission to collect and use technical information gathered as part of the product support services provided to you, if any, related to the Software. Sidelinesoft may use this information solely to improve the Software or to provide customized services to you and will not disclose this information in a form that personally identifies you.

### Disclaimer of Warranties and Limitation of Liability

UNLESS OTHERWISE EXPLICITLY AGREED TO IN WRITING BY SIDELINESOFT, SIDELINESOFT MAKES NO OTHER WARRANTIES, EXPRESS OR IMPLIED, IN FACT OR IN LAW, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF

MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OTHER THAN AS SET FORTH IN THIS AGREEMENT.

Sidelinesoft makes no warranty that the Software Product will meet your requirements or operate under your specific conditions of use. Sidelinesoft makes no warranty that operation of the Software Product will be secure, error free, or free from interruption. YOU MUST DETERMINE WHETHER THE SOFTWARE PRODUCT SUFFICIENTLY MEETS YOUR REQUIREMENTS FOR SECURITY AND UNINTERRUPTABILITY. YOU BEAR SOLE RESPONSIBILITY AND ALL LIABILITY FOR ANY LOSS INCURRED DUE TO FAILURE OF THE SOFTWARE PRODUCT TO MEET YOUR REQUIREMENTS. UNDER NO CIRCUMSTANCES SHALL SIDELINESOFT BE LIABLE TO YOU OR ANY OTHER PARTY FOR INDIRECT, CONSEQUENTIAL, SPECIAL, INCIDENTAL, PUNITIVE, OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOST REVENUES OR PROFITS OR LOSS OF BUSINESS) RESULTING FROM THIS AGREEMENT, OR FROM THE PERFORMANCE, INSTALLATION, USE OR INABILITY TO USE THE SOFTWARE PRODUCT, WHETHER DUE TO A BREACH OF CONTRACT, BREACH OF WARRANTY, OR THE NEGLIGENCE OF SIDELINESOFT OR ANY OTHER PARTY, EVEN IF SIDELINESOFT IS ADVISED BEFOREHAND OF THE POSSIBILITY OF SUCH DAMAGES. TO THE EXTENT THAT THE APPLICABLE JURISDICTION LIMITS SIDELINESOFT'S ABILITY TO DISCLAIM ANY IMPLIED WARRANTIES, THIS DISCLAIMER SHALL BE EFFECTIVE TO THE MAXIMUM EXTENT PERMITTED.

#### **Limitation of Remedies and Damages**

Your remedy for a breach of this Agreement or of any warranty included in this Agreement is the correction or replacement of the Software Product. Selection of whether to correct or replace shall be solely at the discretion of Sidelinesoft. Any claim must be made within the applicable warranty period. All warranties cover only defects arising under normal use and do not include malfunctions or failure resulting from misuse, abuse, neglect, alteration, improper installation, or a virus. All limited warranties on the Software Product are granted only to you and are non-transferable. You agree to indemnify and hold Sidelinesoft harmless from all claims, judgments, liabilities, expenses, or costs arising from your breach of this Agreement and/or acts or omissions.

#### **Severability**

If any provision of this Agreement shall be held to be invalid or unenforceable, the remainder of this Agreement shall remain in full force and effect. To the extent any express or implied restrictions are not permitted by applicable laws, these express or implied restrictions shall remain in force and effect to the maximum extent permitted by such applicable laws.

#### **Termination**

This Agreement is effective until terminated. Without prejudice to any other rights, Sidelinesoft may terminate this Agreement if you fail to comply with the terms and conditions of this Agreement. In such event, you must destroy all copies of the Software License.

#### **Governing Law, Dispute Resolution**

This Agreement is governed by the laws of the State of Colorado, U.S.A., without regard to its choice of law principles to the contrary.

#### **Contact Information.**

Any inquiries regarding this Agreement or the Software may be addressed to Sidelinesoft at the Software Product website ([nl5.sidelinesoft.com](http://nl5.sidelinesoft.com)).

*The end*