Title: Virtual Prototyping in PSpice

Product: PSpice

Summary: How to define C/C++, SystemC and Verilog-A components in PSpice. Extension to Hardware in the Loop using Arduino

Version up 17.2

Autor/Datum: Roberto Gandía / 22.07.2016

Table of Contents

1 Overview

Virtual prototyping is a method in the process of product development, which allows to validate a design before making a physical prototype. Since V17.2, PSpice offers the opportunity to simulate System Designs using different kind of abstractions thanks to the Device Modeling Interface. With GUI, users can define C/C++, SystemC, and Verilog-A components and simulate them in simulator.

1.1 Contents of this document

- How to use Device Modeling Interface.
- Setup for Visual Studio Community 2013.
- How to integrate C/C++, SystemC and Verilog-A models to be simulated in PSpice.
- Debug of C/C++, SystemC and VerilogA devices.
- Importation of MATLAB Blocks in PSpice.
- Hardware in the Loop using Arduino.

1.2 How to use this AN?

This document explains the steps for integrating C/C++, SystemC and Verilog-A models with PSpice Device Model Interface (DMI), so that they can be used for PSpice simulations.

This document is valid up Release 17.2. License required for:

a. PSpice DMI – Model **development** capability:

   - OrCAD PSpice Designer **OR**
   - OrCAD PSpice Designer Plus **OR**
   - Allegro PSpice Simulator

b. PSpice DMI – Model **Simulation** capability:

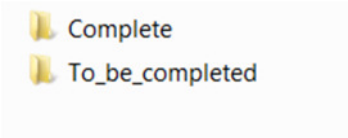
   - OrCAD PSpice Designer Plus **OR**
   - Allegro PSpice Simulator

Examples available:

- Digital Power Supply using C/C++ defined PWM.
- FIR Filter using SystemC.
- Capacitor behavior analysis defined with Verilog-A.
- Noise Filter using a MATLAB Block.
- Hardware in the Loop using Arduino.

The structure of the attached ZIP file is divided in 6 folders:

- Each folder at the same time is divided in two folders:



- You will work with the folder called “To_be_completed”.
- In each example the reference <directory> is used. It means the directory where you place this ZIP File.

NOTE: You can find more specific information clicking on Start → All Programs → Cadence Release 17.2-2016 → Documentation → Cadence Help. Search for the next documents:

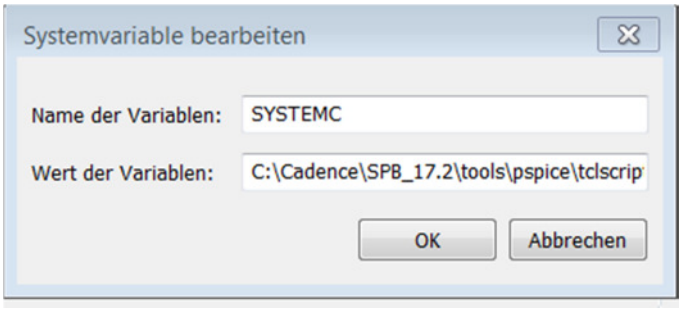
- pspDMIRef
    - Information about the functions exposed by PSpice Engine and by the DLL files.
- pspcref
    - Information about the meaning of the different parameters you can define for DMI components.

2 Device Modeling Interface Template Code Generator

2.1 Required Software and Setup

This module covers the information on the setup required for compilation of C/C++, SystemC and Verilog-A models in PSpice.

- SYSTEMC is an environment Variable you need to use if you want to define SystemC components. For that, setup SYSTEMC environment variable pointing to the SystemC installation path.



  %CDSROOT%\tools\pspice\tclscripts\pspModelCreate\SystemC

  e.g.

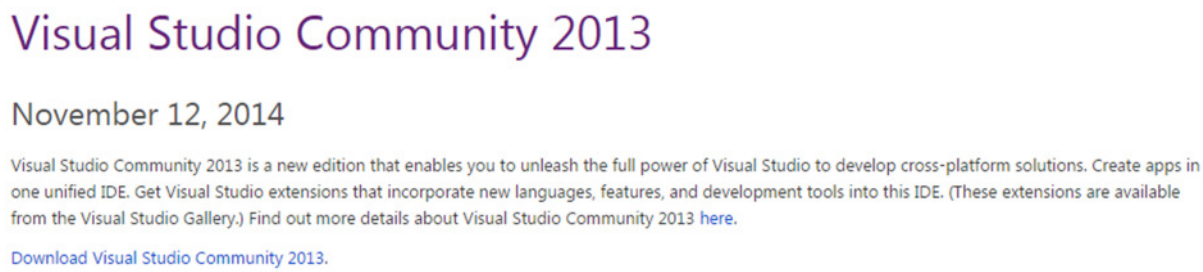
  C:\Cadence\SPB_17.2\tools\pspice\tclscripts\pspModelCreate\SystemC

- Visual Studio Community 2013

  It is a free software that you can download easily here:

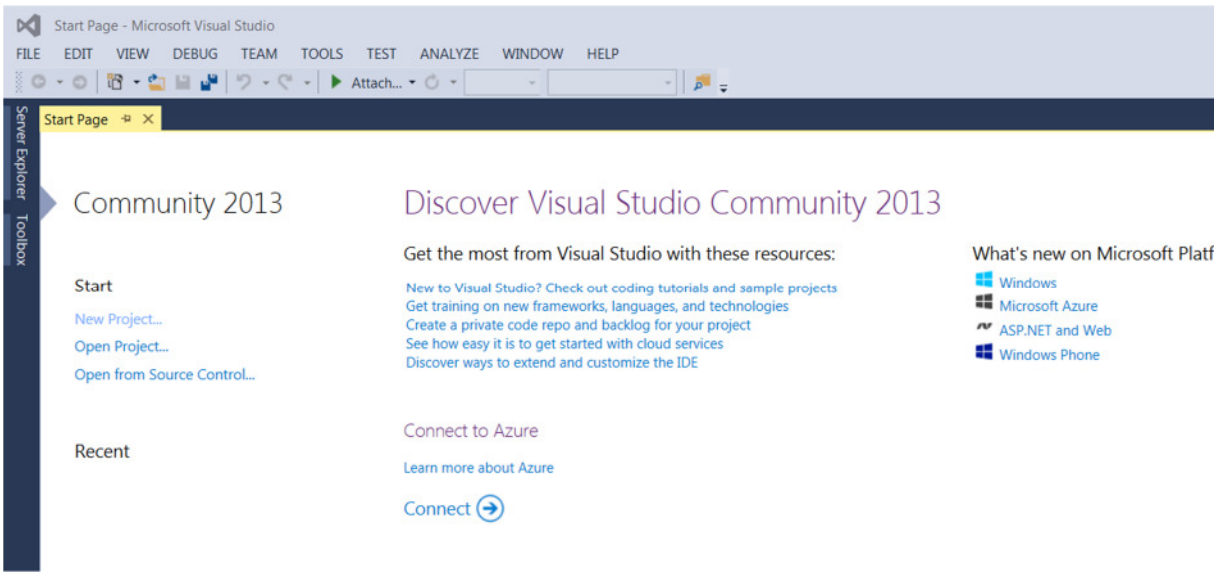
  https://www.visualstudio.com/en-us/news/vs2013-community-vs.aspx

  You will be forwarded to Visual Studio Community 2013 Website. Click on Download and install it.



  After the installation, open the software and verify it works.

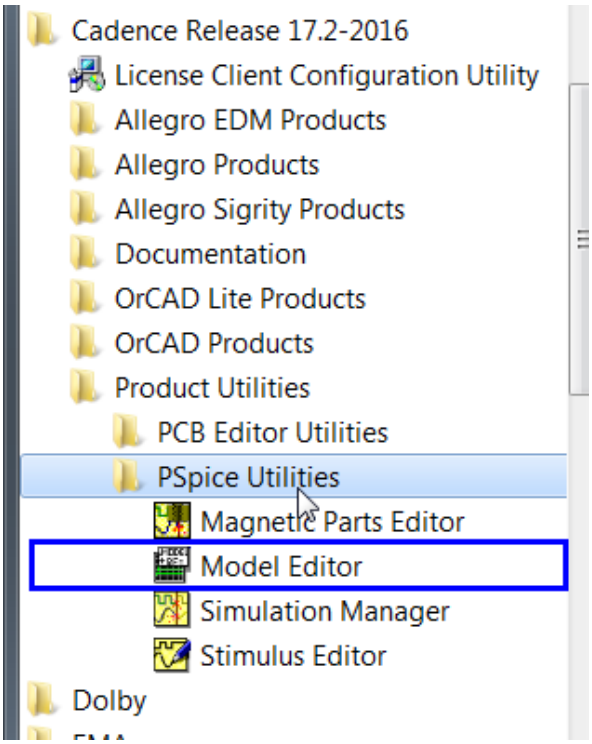
  NOTE: It works with VS Express as well, but the project will not be generated automatically.



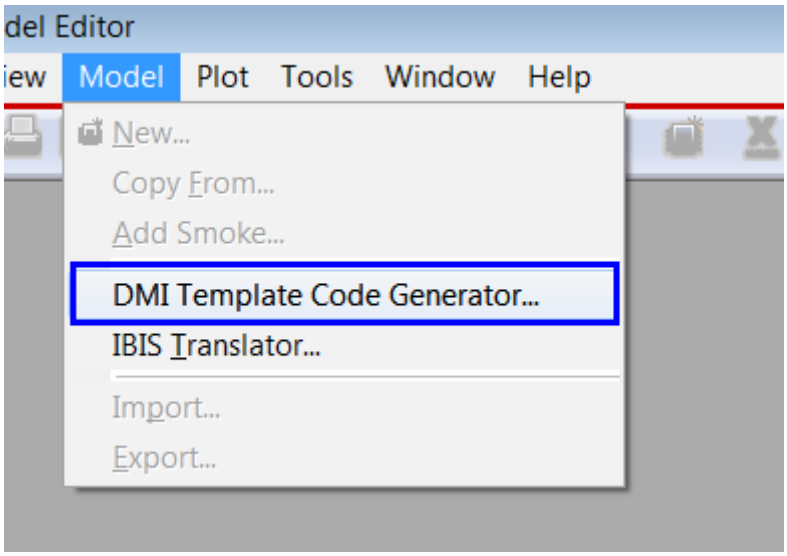
- Arduino IDE (Only in case you want to do Hardware in the Loop)
    - You can download the software from Arduino Website.
    - You also need the Hardware, specifically the Arduino Starter Kit.

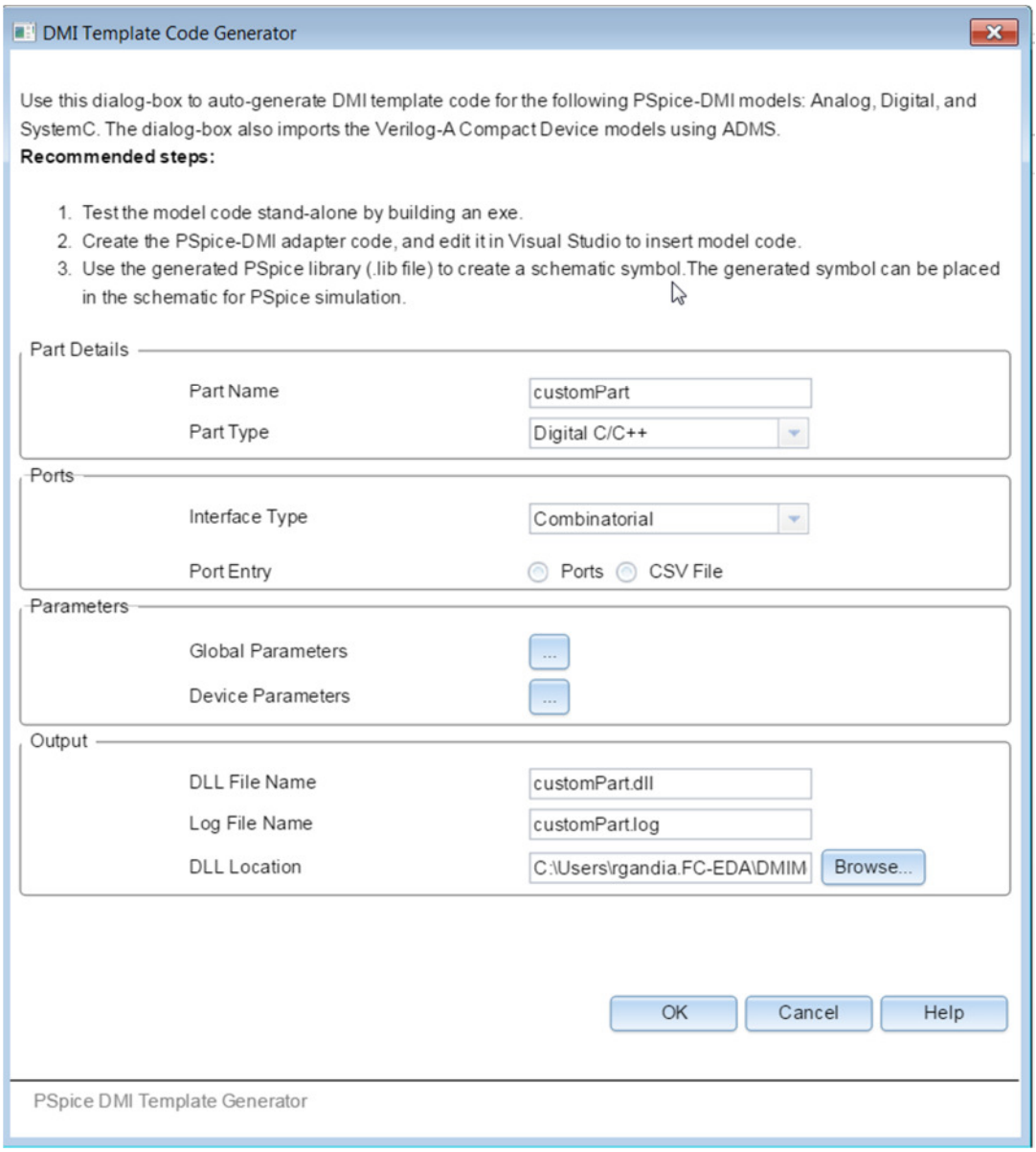
2.2 How to use Device Modeling Interface?

Device Modeling Interface Template Code Generator can be launched from Model Editor in PSpice Accessories.



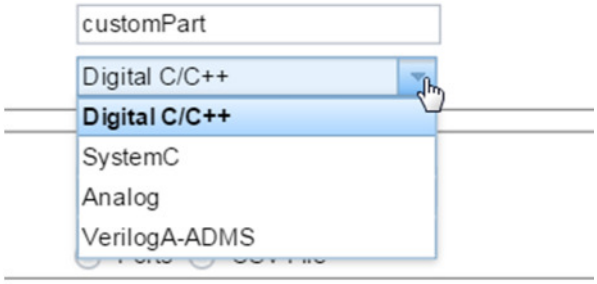
clicking on Model → DMI Template Code Generator.



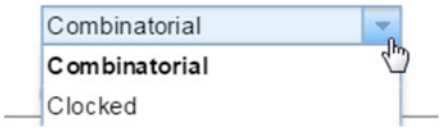


This interface is divided into 4 sections:

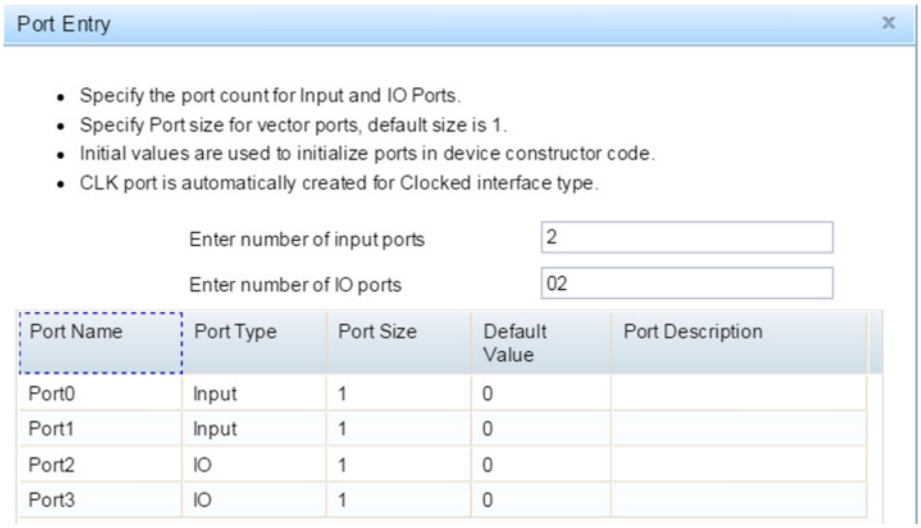
- Part Details:
  - Part Name: Name for the function you want to define. For example, Pulse Width Modulation.
  - Part Type: Define the type of component you want to describe.



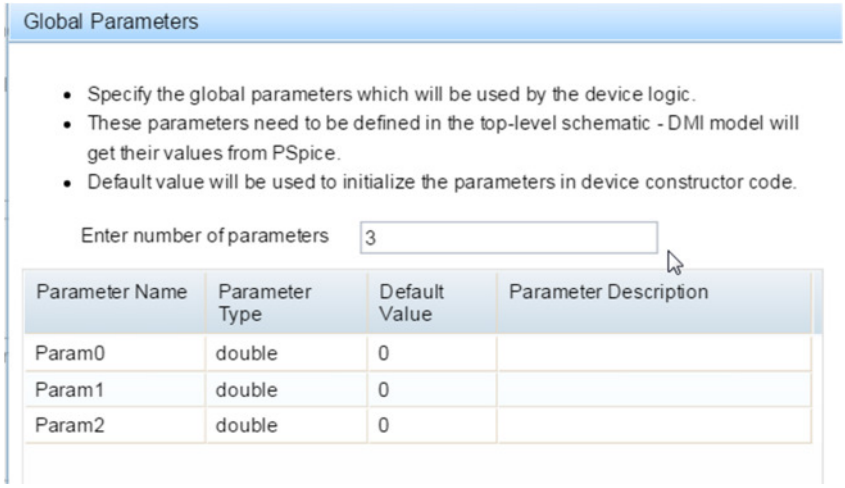
- Ports:
  - Interface Type: Classify your component as Clocked or as Combinatorial.



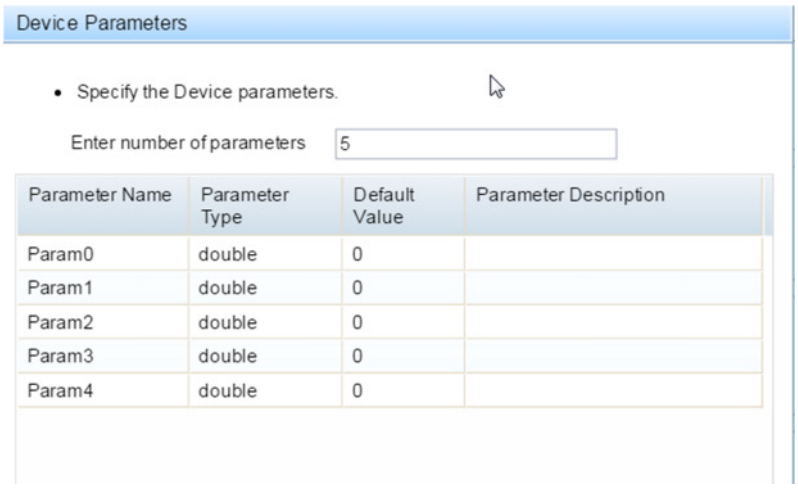
- Port Entry: Define the inputs and the outputs of your device through an excel file or writing down them directly.



- Parameters:
  - Global Parameters: Parameters that can be used by all the components which define them.



  - Device Parameters: Parameters used just for a particular device.



- Output: Define the name for your DLL, the log file where information of the generation of the DLL is going to be written and the directory where it is going to be saved.

3 Examples

3.1 Digital Power Supply using a C/C++ defined PWM

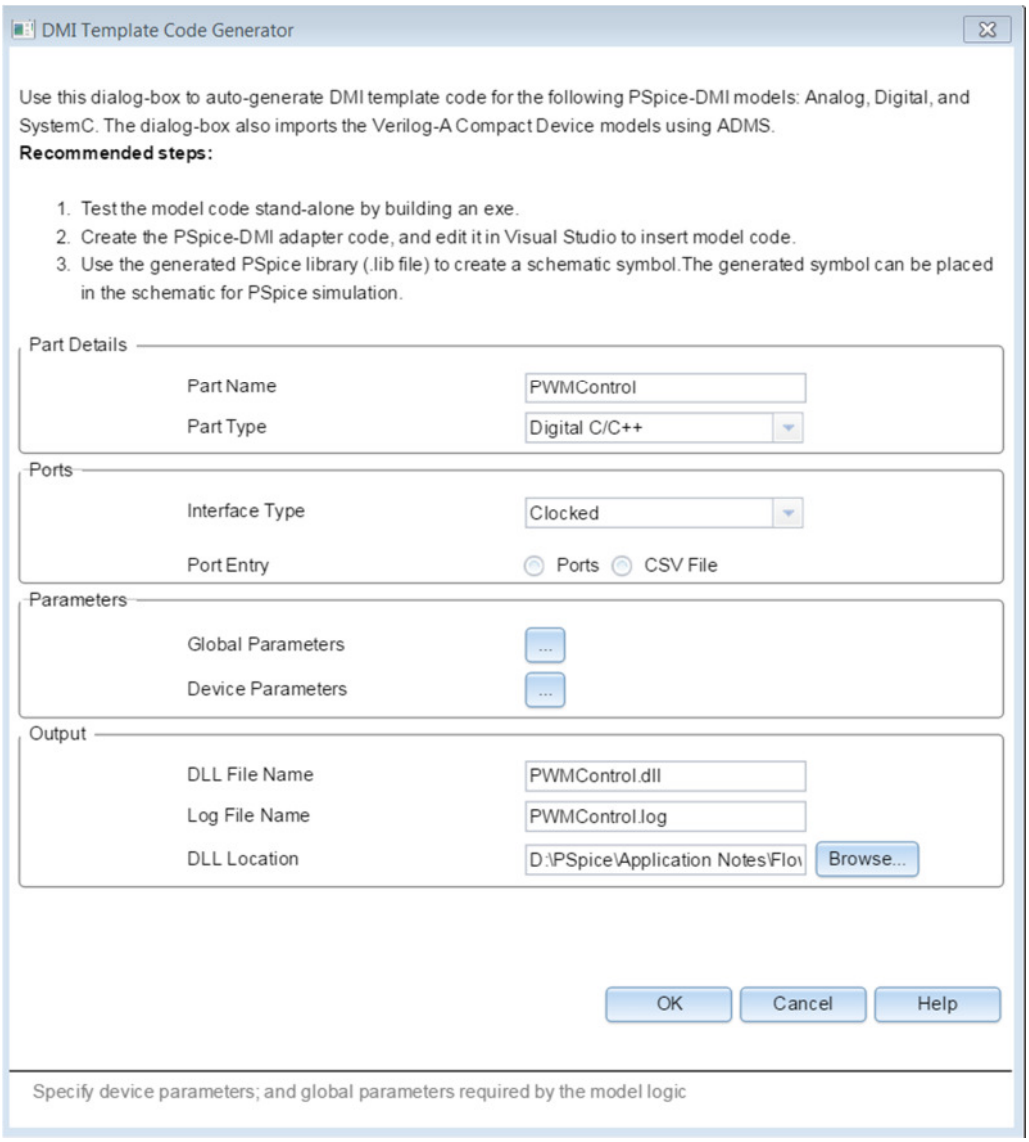
This module shows a Digital Power Supply design, which uses models with different levels of abstraction.

This example demonstrates:

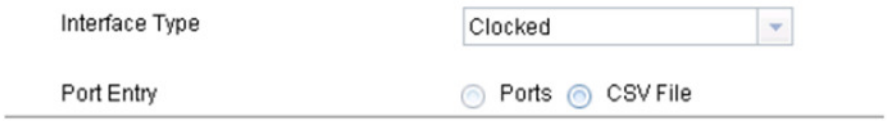
- The generation of template code for a DMI Model and the implementation of a Digital PWM Control block.
- Simulation of the DMI model in the context of a Digital Power Supply circuit.

Steps:

1. Launch Model Editor
2. Select Menu Item Model → DMI Template Code Generator
3. Enter the data as follows:



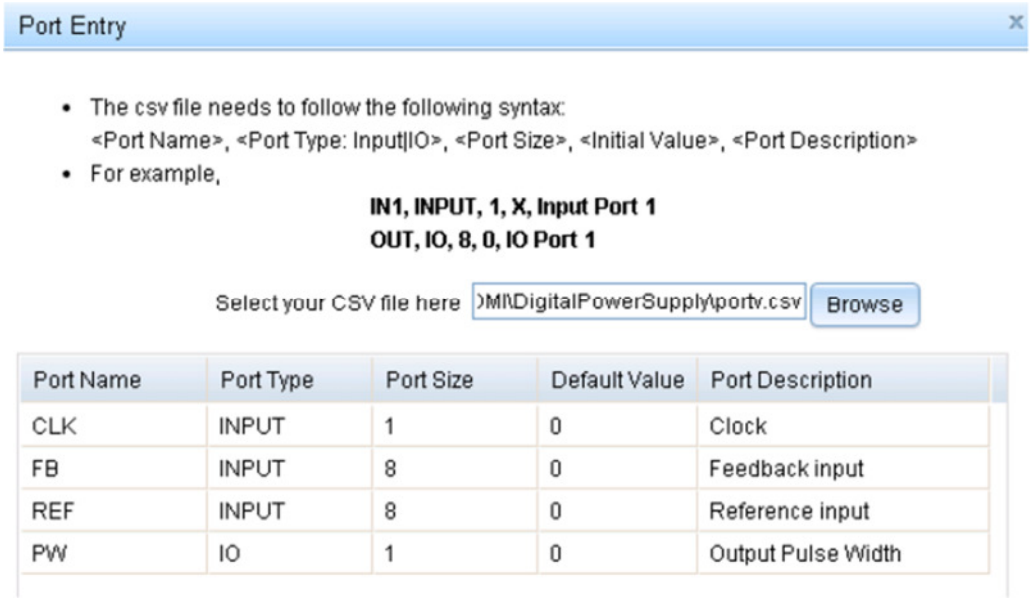
4. Click on CSV File to enter ports using a csv file



5. Browse to the csv file **portsv.csv**

   You will find this file in

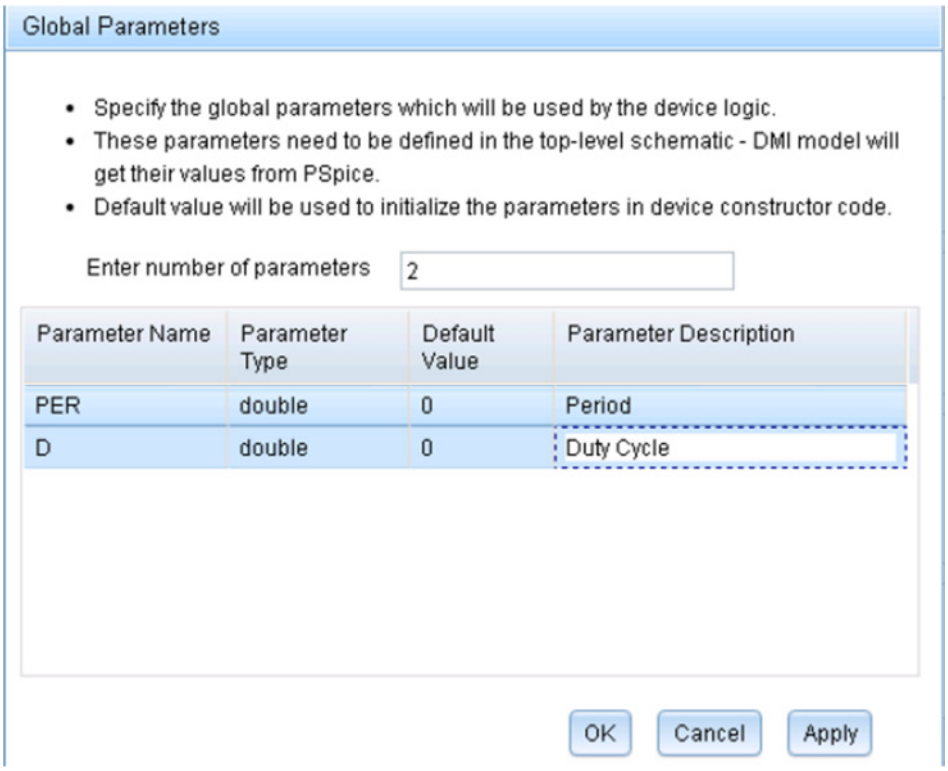
   <directory>\Circuits\Digital Power Supply\To_be_completed\Ports



Port Name	Port Type	Port Size	Default Value	Port Description
CLK	INPUT	1	0	Clock
FB	INPUT	8	0	Feedback input
REF	INPUT	8	0	Reference input
PW	IO	1	0	Output Pulse Width

   The ports are automatically read from the csv file and populated in the form as above.

6. Review the port list and click OK.
7. Enter Global parameters as shown below:



8. Click OK.
9. Before generating the whole files, browse to directory to locate them.

   <directory>\Circuits\Digital Power Supply\To_be_completed\DMI_Code

   NOTE: This file does not generate any DLL. The DLL will be generate it in Visual Studio when the code is compiled.

10. Click OK and generate the files:

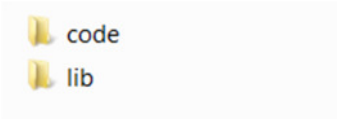
    NOTE: A message might appear indicating that it could not be possible to find the lib. This message can be clicked away and ignored.

```
tran.out | capDMI.net | cap.lib | tran.out | pspPWMControl.h | pspPWMControl.log*
Format  Column  Macro  Advanced  Window  Help
Reading json
Json read..Logging Model Creation...
reading Model Name and Type...
reading global and instance params for digital and analog..
declaring the global variables..
reading instance params..
declaring and defining instance params...
Copying common files ..
Creating Digital Model...
Getting port information...
Initializing input port specific variables...
A\capture\DESIGN1-PSpiceFiles\capDMI\tran\tran.cir
ific variables...
writing port specific code in cpp file...
Changes in evaluate function...
Digital/SystemC GUID creation...
read input file CommonTemplate/pspEngFunc_model.cpp
opened output file D:/customers/Denmark2016/DMI/DigitalPowerSupply/Code//pspEngFunc.cpp for writing
read input file CommonTemplate/pspEngFunc_model.h
opened output file D:/customers/Denmark2016/DMI/DigitalPowerSupply/Code//pspEngFunc.h for writing
read input file DigitalTemplate/pspDigitalModelName.cpp
opened output file D:/customers/Denmark2016/DMI/DigitalPowerSupply/Code//psppspPWMControl.cpp for writing
read input file DigitalTemplate/pspDigitalModelName.h
opened output file D:/customers/Denmark2016/DMI/DigitalPowerSupply/Code//psppspPWMControl.h for writing
read input file DigitalTemplate/digitalmodel_user.cpp
opened output file D:/customers/Denmark2016/DMI/DigitalPowerSupply/Code//pspPWMControl_user.cpp for writing
Populating vcproj with the source and header files
read input file DigitalTemplate/DigitalModelName.vcxproj
Lib created..
```

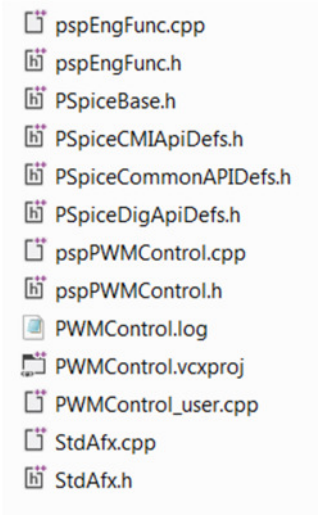
11. The new created files are located in a new folder called PWMControl. You can find this folder in

    <directory>\Circuits\Digital Power Supply\To_be_completed\DMI_Code

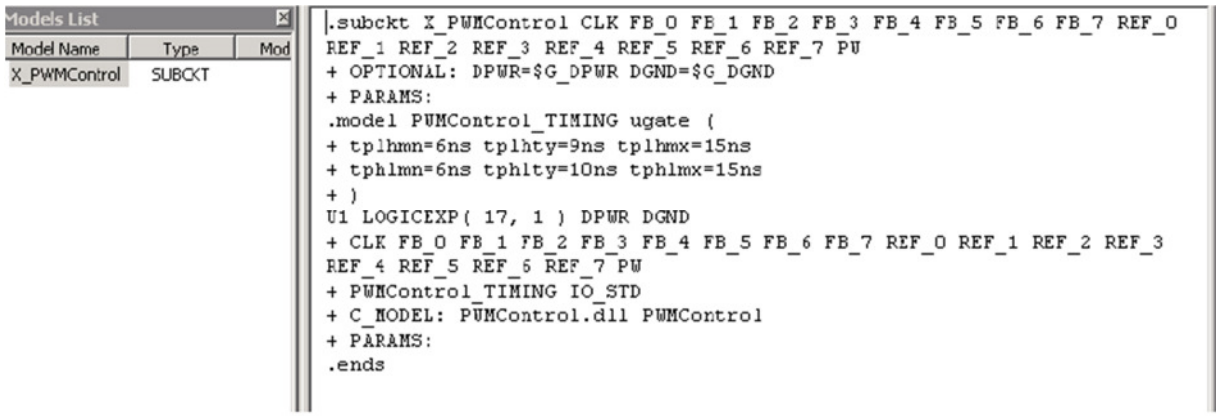
    In this folder, the created files are organized in two folders:



    In code you will find the Visual Studio project files. With them you will generate the DLL:



    In lib you will find the PSpice Model (PWMControl.lib).

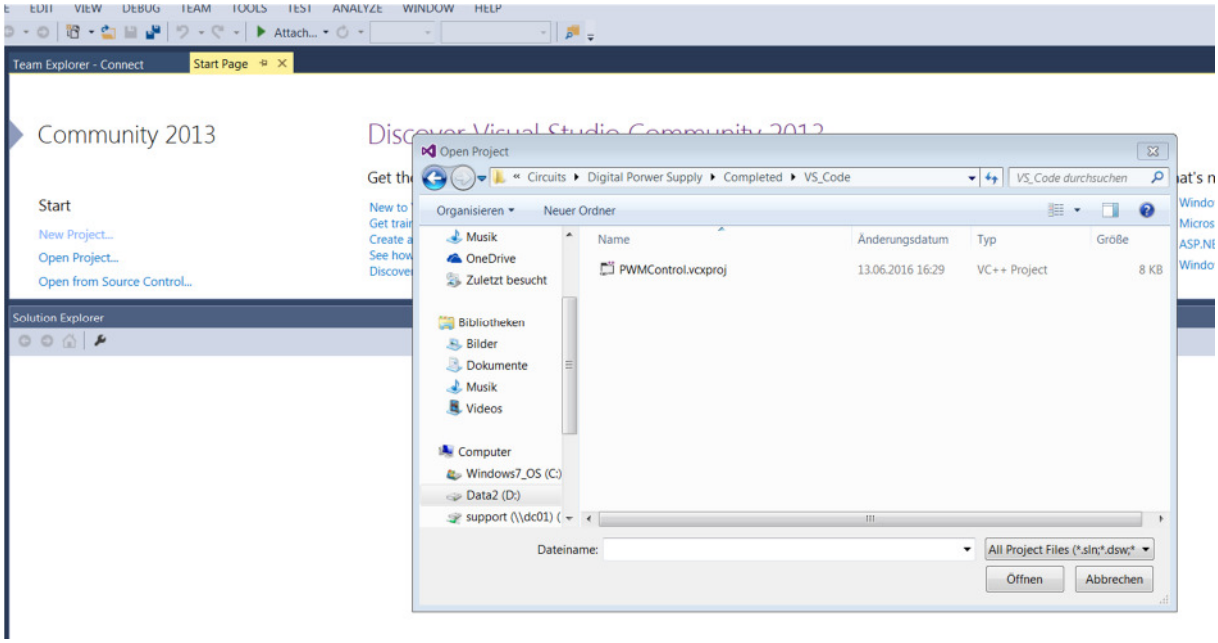


Note: The generated model already points to the PSpice-DMI dll PWMControl.dll, although it still has to be generated – For that, the next step is to describe the model code and generate the dll.

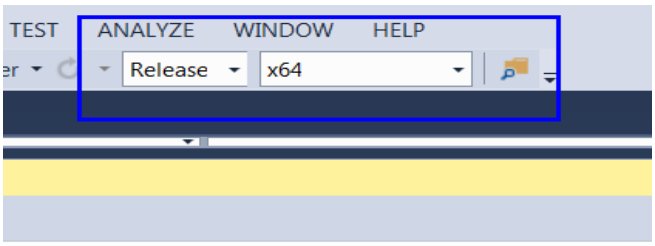
12. Launch Visual Studio Community 2013
13. Click File→ Open→ Project/Solution

    Search for PWMControl.vcxproj located in:

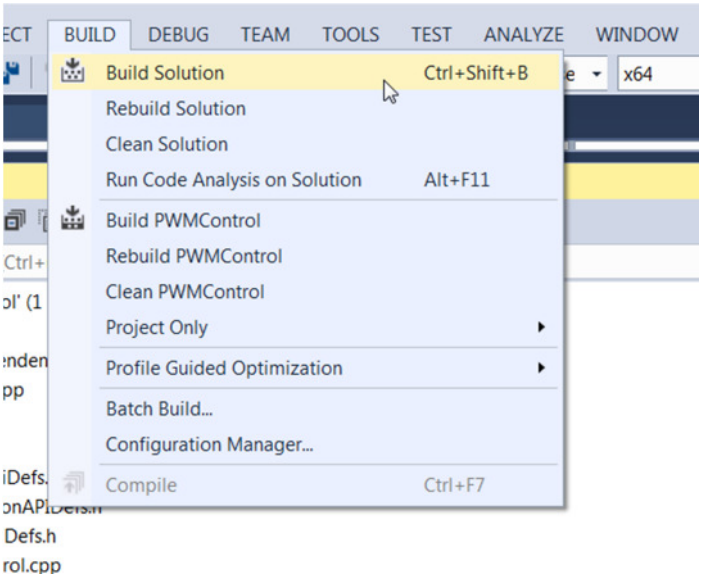
    <directory>\Circuits\DigitalPowerSupply\To_be_completed\DMI_Code\PWMControl\code

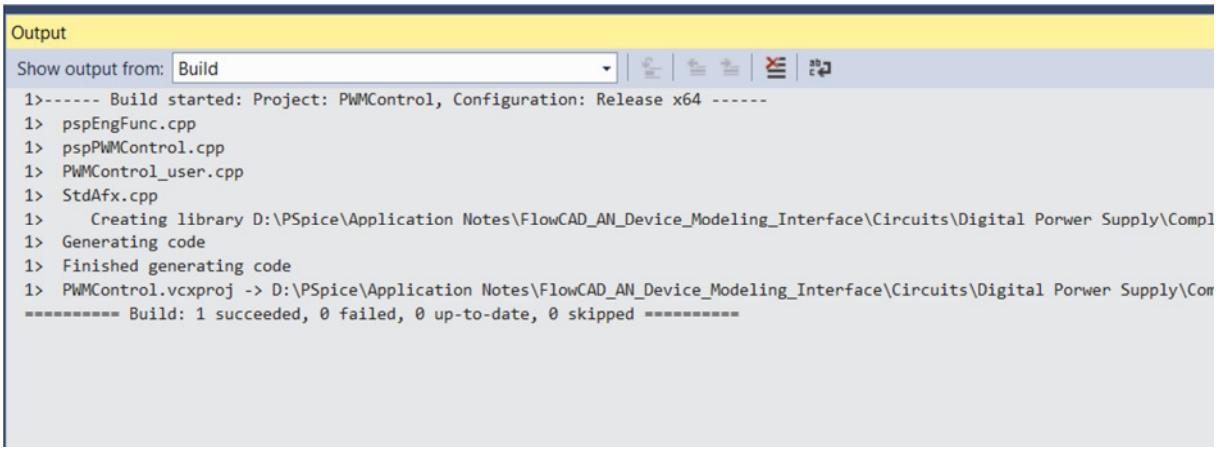


14. Change the default configuration on the top to Release x64



and Build the solution to verify that there are no build issues.





15. Edit the file PWMControl_user.cpp

16. Go to evaluate function by searching for the text “pspPWMControl::evaluate”. Now, search for “// LOGIC TO BE IMPLEMENTED BY USER”

    This is the place where you will add the model evaluation code.

17. Open the file **COPY.TXT** located in

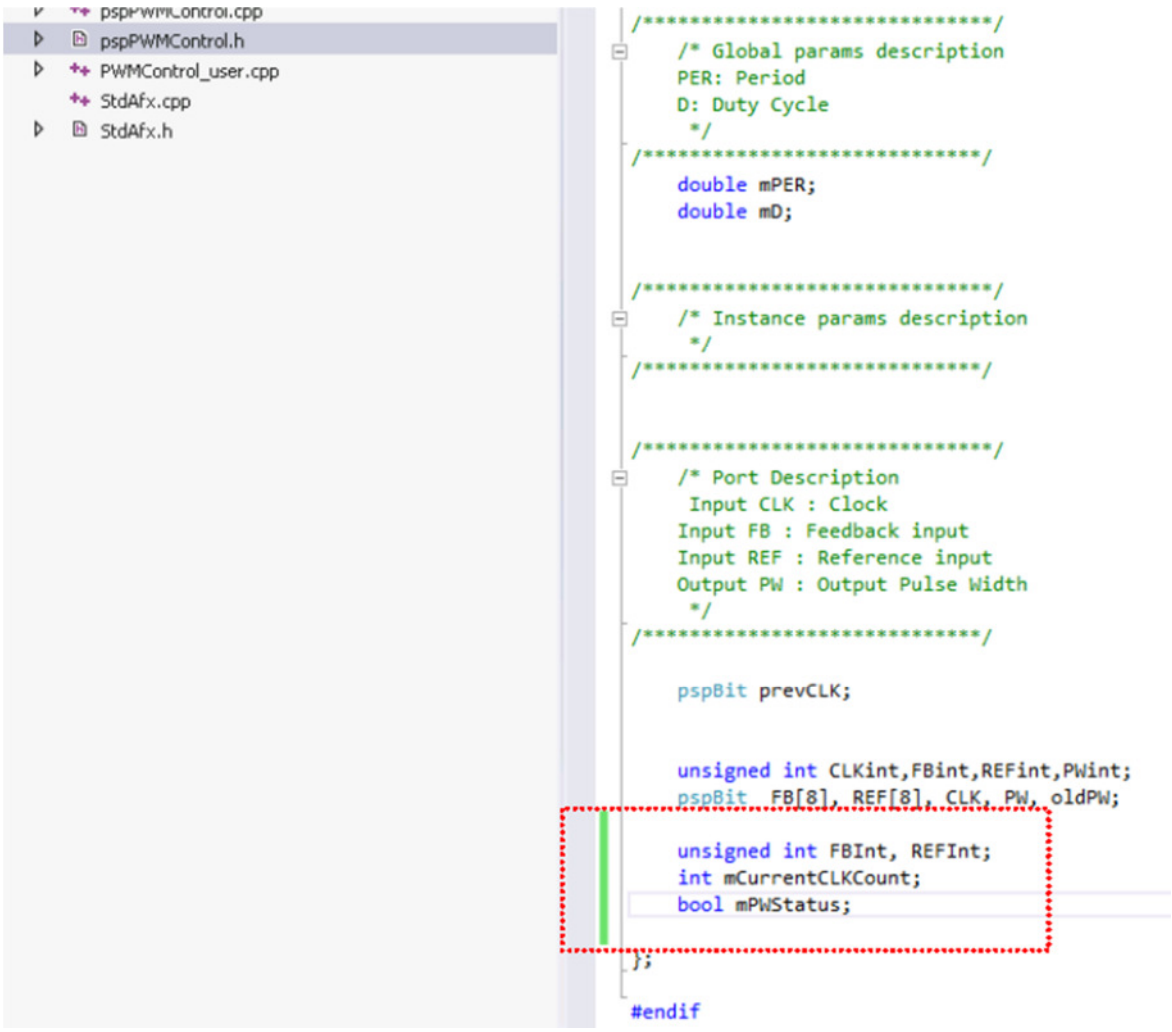
    <directory>\Circuits\DigitalPowerSupply\To_be_completed\DMI_Code

copy the code that there is inside and paste it just after //LOGIC TO BE IMPLEMENTED BY USER.

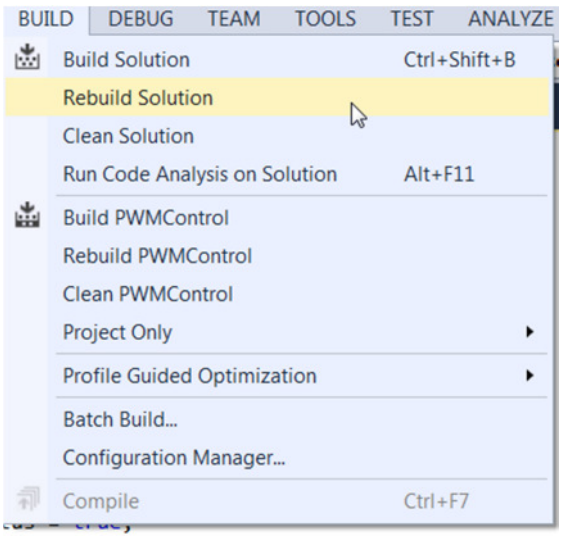
```
// LOGIC TO BE IMPLEMENTED BY USER
        oldPW = PW;
        pspBits2Int(FB, FBInt, 8);
        pspBits2Int(REF, REFInt, 8);

        if (REFInt > FBInt && mD < 0.98) {
            mD += 0.01;
        }
        else if (REFInt < FBInt && mD > 0.02) {
            mD -= 0.01;
            //fprintf(stderr, "Reducing DutyCycle\n");
        }

        if (mCurrentCLKCount <= 0) {
            mCurrentCLKCount = mPER;
        }

        if (mCurrentCLKCount > mD * mPER)
            mPWStatus = false;
        else
            mPWStatus = true;

        if (mPWStatus == true && (int)PW != 1){
            PW = pspBit::HI;
        }
        else if (mPWStatus == false && (int)PW != 0){
            PW = pspBit::LO;
        }

        mCurrentCLKCount--;


        fprintf(stderr, "FBInt=%d REFInt=%d mDutyCycle=%g PW=%d\n", FBInt, REFInt, mD, (int)PW);
```

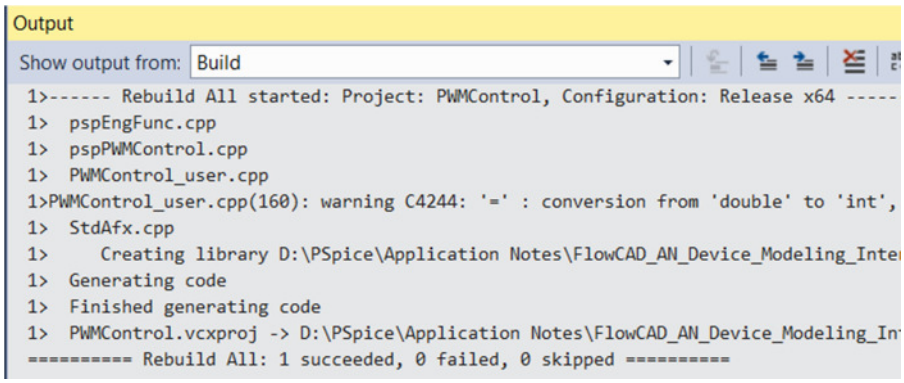
18. Save the project.

19. This code uses some extra variables which need to be declared – edit file pspPWMControl.h and add the following lines at the end before the last closing brace.



20. Build the solution again. The model dll is now built with the required model evaluation code.



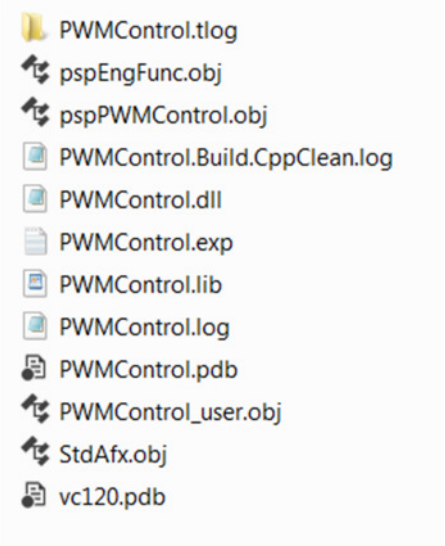


NOTE: Do not consider the warning.

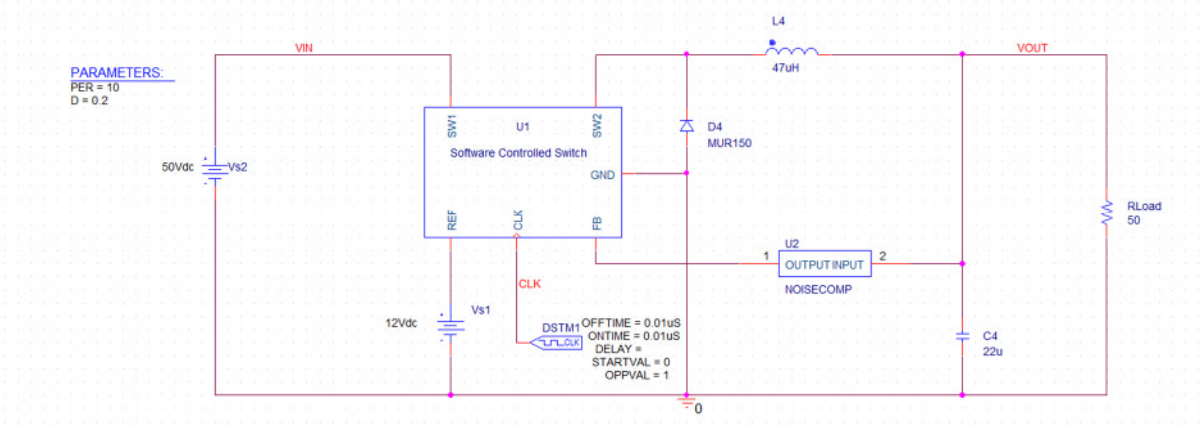
21. Go to

    <directory>\Circuits\DigitalPowerSupply\To_be_completed\DMI_Code\PWMControl\code\x64\Release

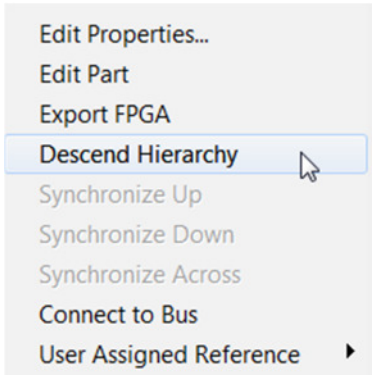
    and verify that the DLL and the PDB has been generated:



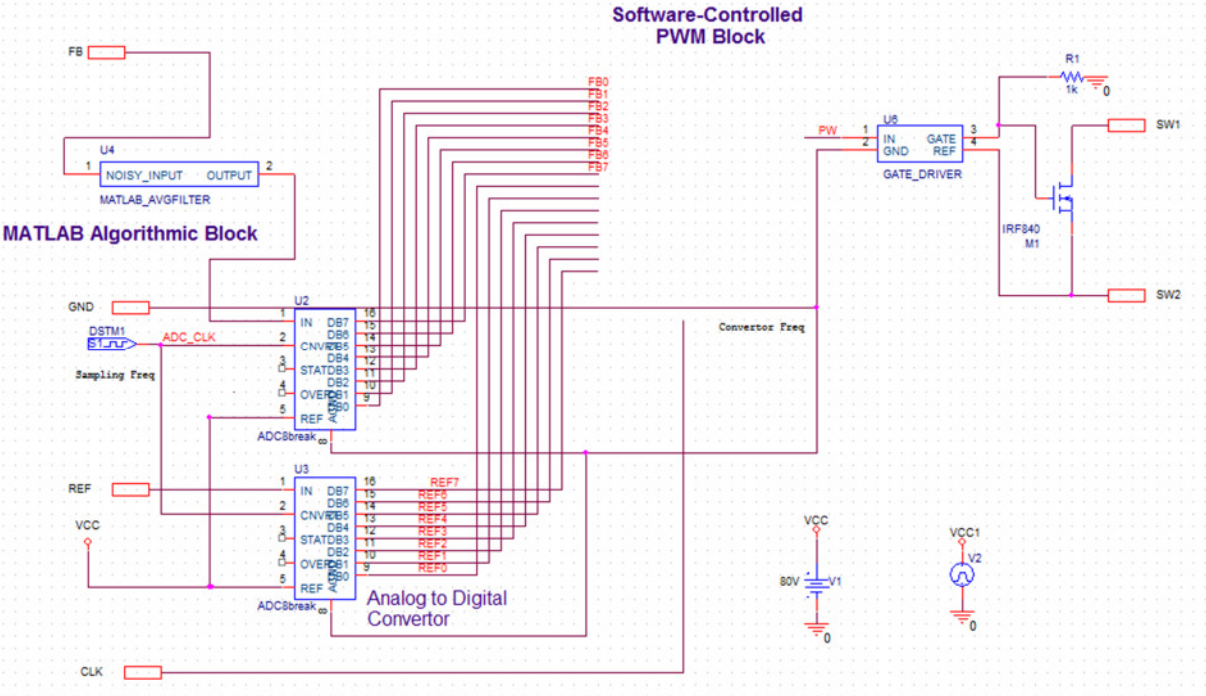
22. Open OrCAD Capture and click File→ Open→ Project.
23. Open the Capture project located in <directory>Circuits/Digital_Power_Suppy/To_be_completed/circuit/Power_Supply_incomplete.dsn:



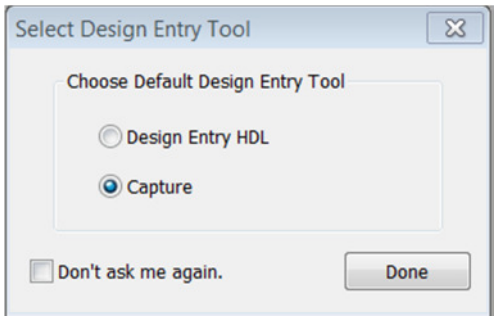
Click on the component Software Controlled Switch with RMB and click on Descend Hierarchy:



Now you have to create the Symbol for the PSpice component you have just described: **PWM Block**



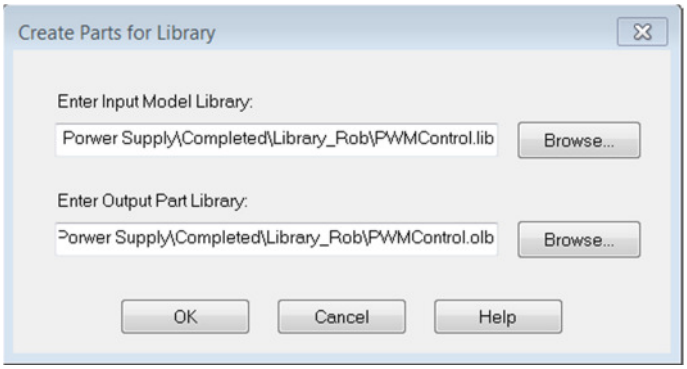
24. Open Model Editor and click on Done.



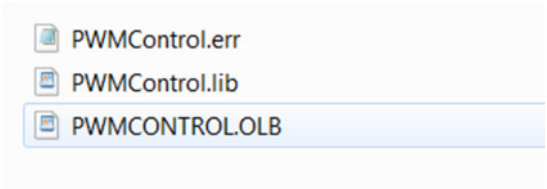
25. Click on File → Open and look for PWMControl.lib

    <directory>\Circuits\DigitalPowerSupply\To_be_completed\DMI_Code\PWMControl\lib

26. Click on File → Export to part library and click OK.

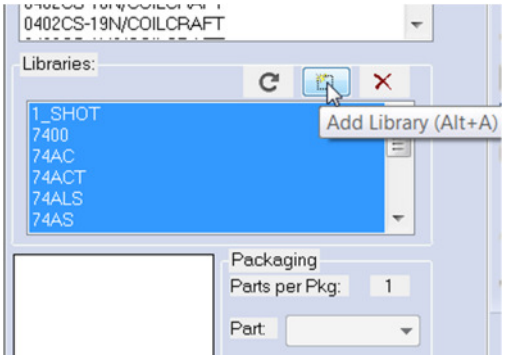


With this option you are creating the schematic part for this PSpice component automatically:

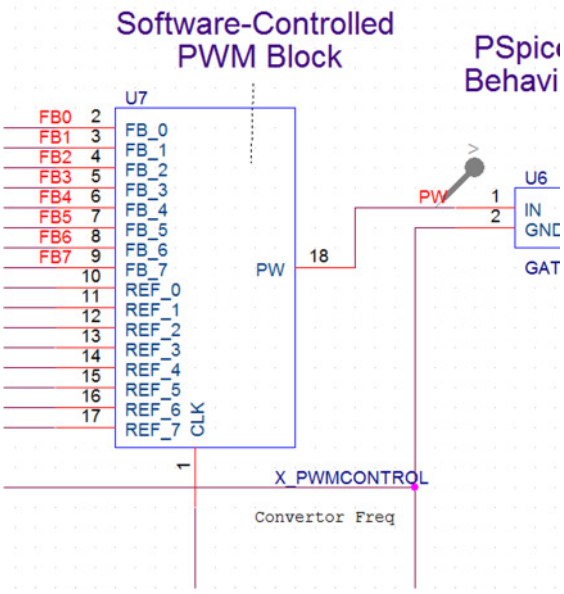


27. Come back to the Schematic and place the symbol you have just created, modifying this symbol as seen below or using the one from

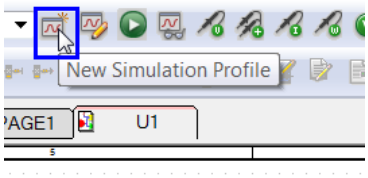
    <directory>\Circuits\Digital Power Supply\To_be_completed\Library\Capture

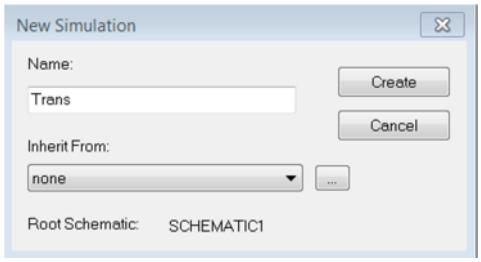


28. Place the symbol and connect it in the circuit:

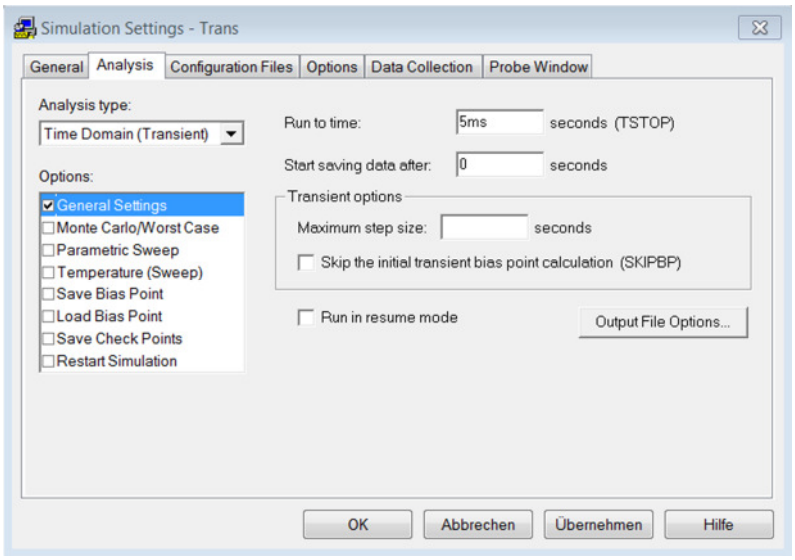


29. Define a Transient Analysis Simulation and call it Trans:





30. Define Run to Time = 5ms



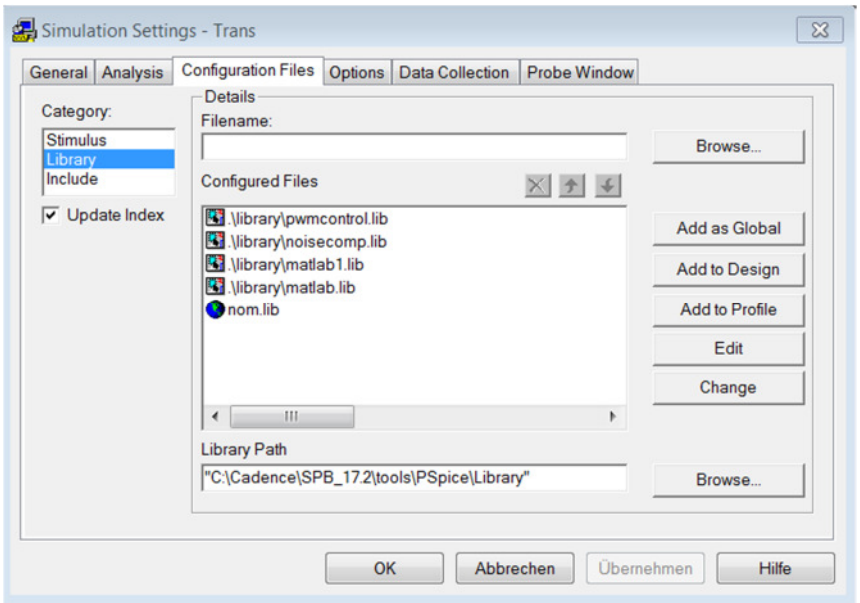
31. Click on Configuration Files. You have to import the PSpice model for the component you have just created and the already defined ones. Click on Browse, select one by one and click on Add to Design.

    You will find your library PWMControl.lib in:

    <directory>/Circuits/DigitalPowerSupply/To_be_completed/DMI_Code/PWMControl/lib

    You will find the other libraries in
    <directory>/Circuits/Digital Power Supply/To_be_completed/Library/Capture:

    a. Noisecomp.lib
    b. Matlab1.lib
    c. Matlab.lib

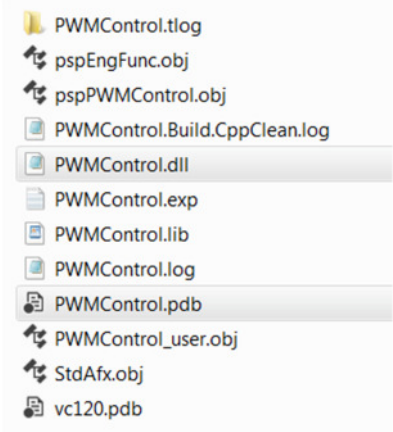


32. Before running the simulation, do **NOT** forget to associate all the .DLL's, which describe the behaviour of your components. For that, place the .DLL and the .PDB in the Simulation Profile Folder that you have just created:

    <directory>\Circuits\DigitalPowerSupply\To_be_Completed\circuit\Power_Supply_Uncomplete-PSpiceFiles\SCHEMATIC1\Trans

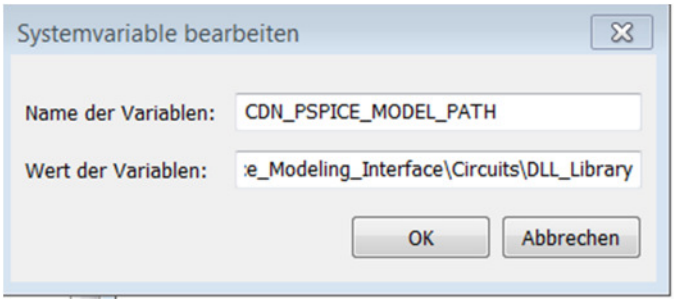
    NOTE:

    Your created .DLL is in:
    <directory>/Circuits/DigitalPowerSupply/To_be_completed/DMI_Code/PWMControl/code/x64/Release



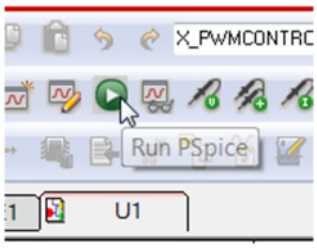
    The other one is in:
    <directory>/Circuits/DigitalPowerSupply/To_be_completed/Library/DLL

    NOTE: There is another option to store your created DLL's and PDB's using an Environment Variable. It allows you to use a central library for all your DLL's.

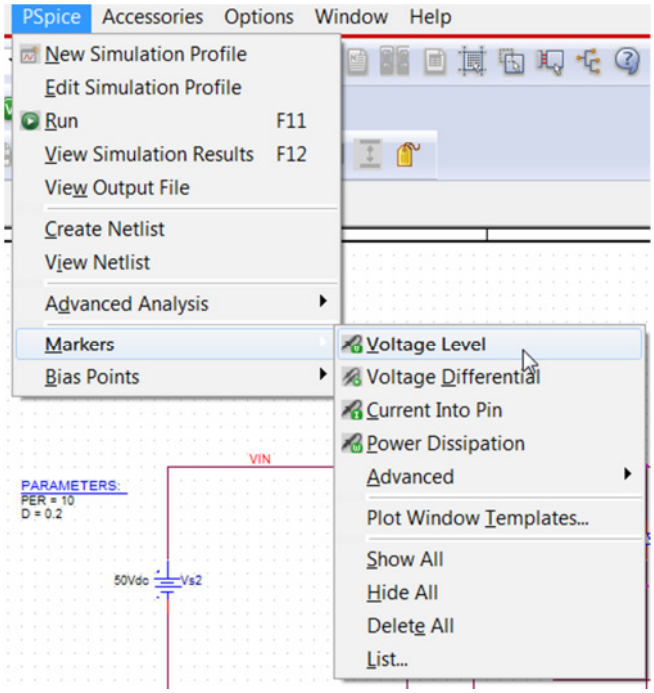


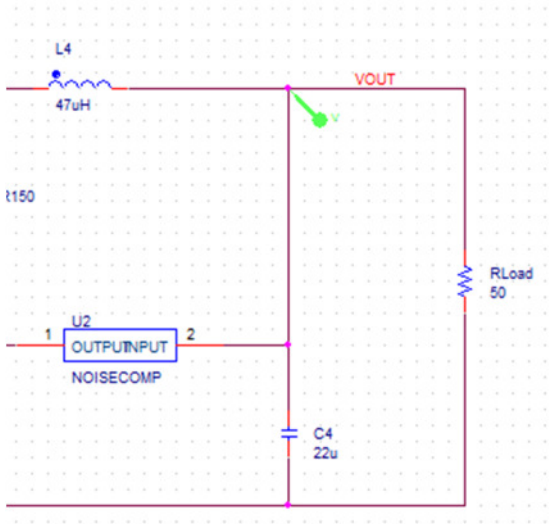
    In this Application Note however, where are placing them in the Simulation Profile Folder.

33. Click on Run:



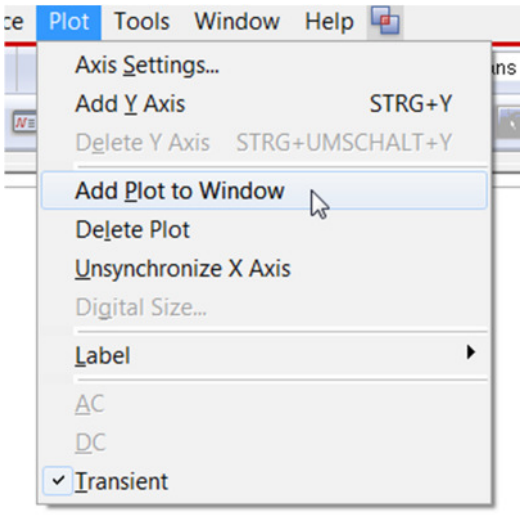
34. Analyze the results. Click on PSpice → Markers → Voltage Level and place it in the node called VOUT.



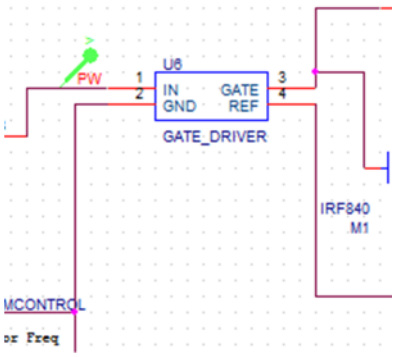


You will be able to see this results in the Probe Window:

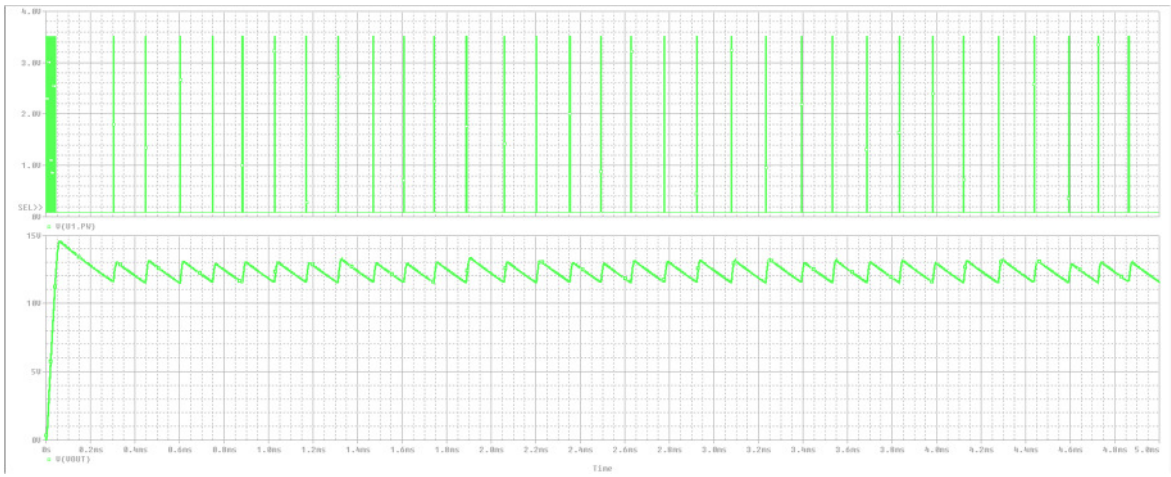
35. In Probe Window click on Plot → Add plot to Window.



36. Go to OrCAD Capture again, and place another Voltage Marker in the wire called PW:



37. Visualize the results in the Probe Window:



NOTE: In this case everything has worked perfectly, but most times it is necessary to debug for detecting error and mistakes in the source code description. More information in section 3.3.

3.2 FIR Filter using SystemC

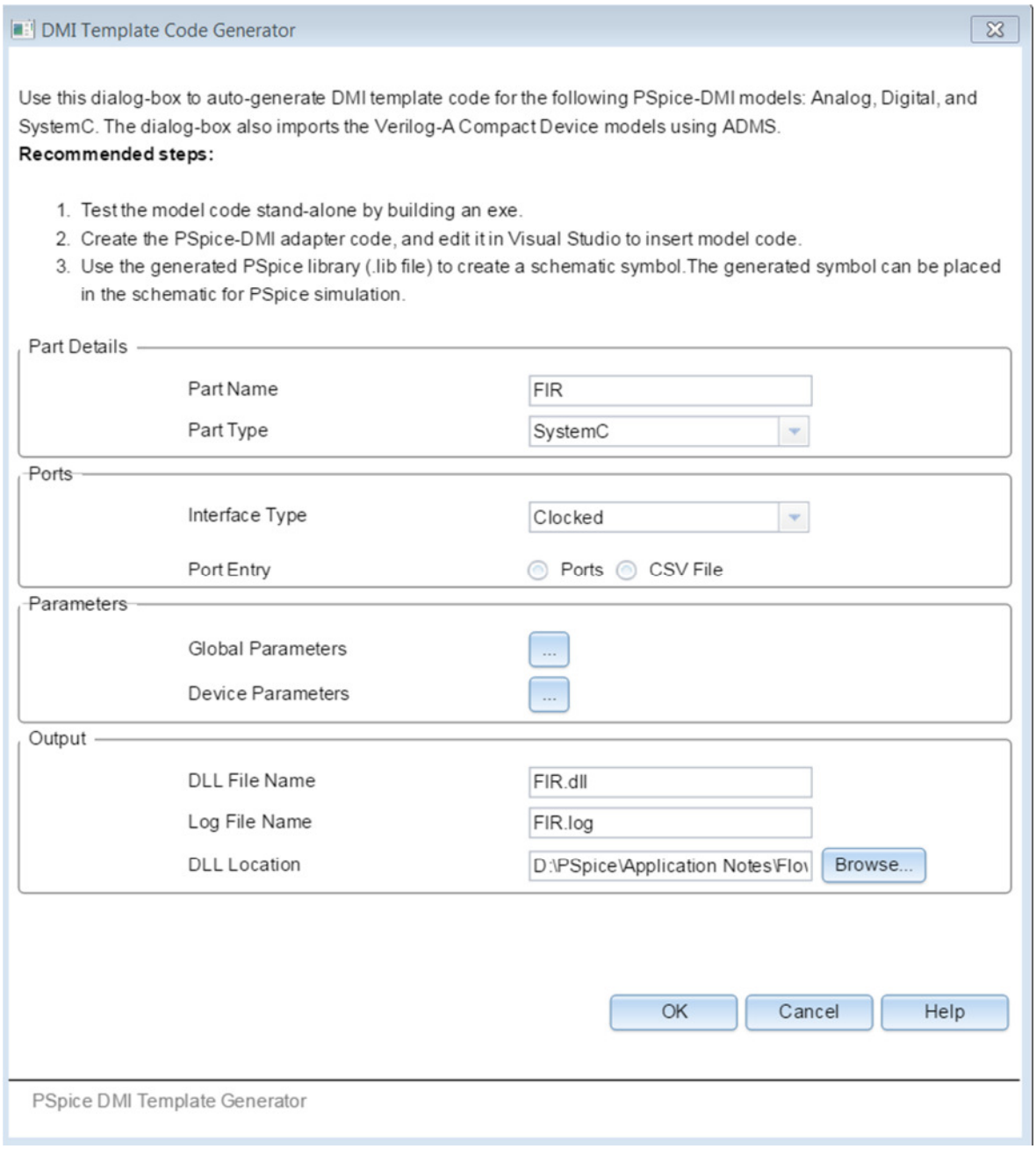
This module explains definition and simulation of a SystemC model using PSpice-DMI.

It demonstrates:

- An example of a FIR model written in SystemC.
- Generation of DMI Template code for SystemC models using Model Editor.

Steps:

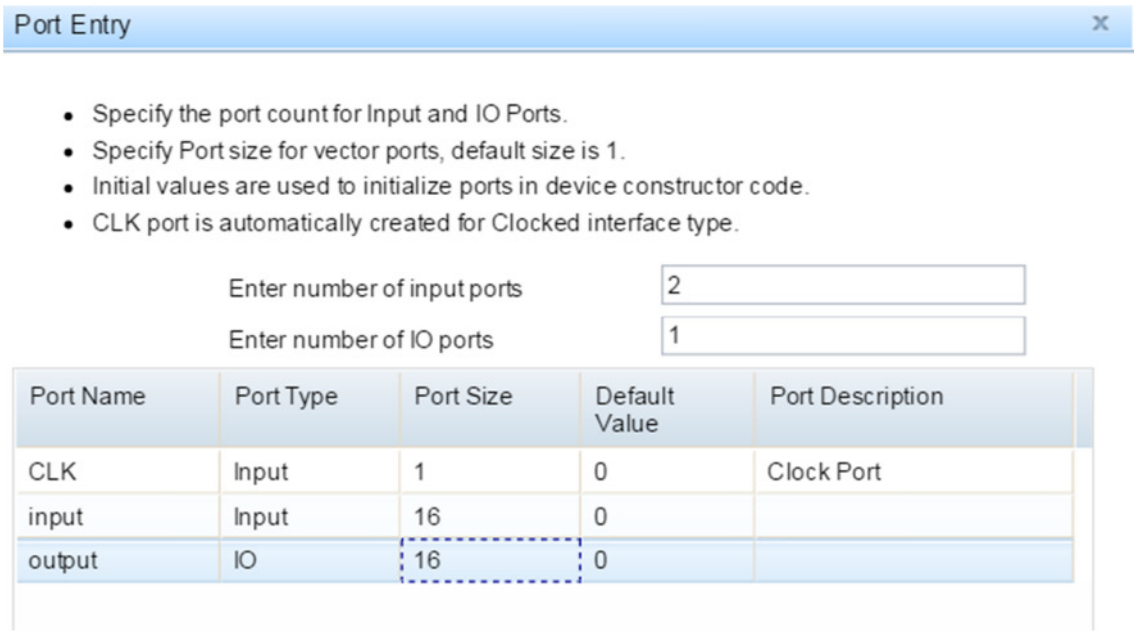
1. Launch Model Editor
2. Select Menu Item Tools/DMI Template Generator
3. Enter data as below:



DLL Location: <directory>\Circuits\FIR_Filter\To_be_completed\DMI_Code

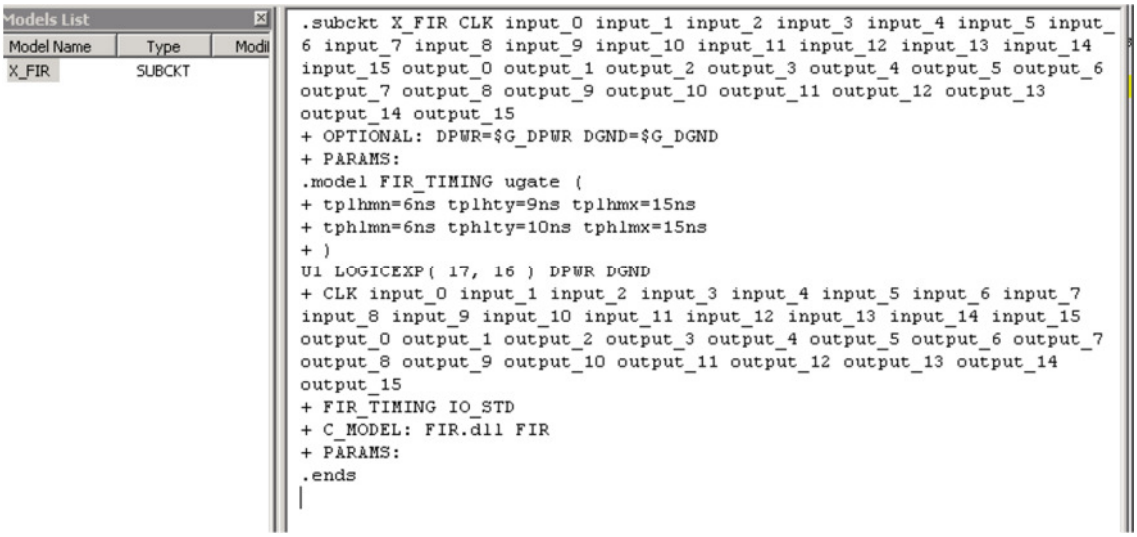
Do not click OK.

4. Click on Ports radio button to enter the following port data:



5. Click OK to close the Port Entry and click OK to generate the code.

6. The PSpice library for the generated model is created automatically. The vector ports are expanded to PSpice-supported scalar ports in the lib file:



This library points to the PSpice-DMI dll FIR.dll – The next step is to complete the model code and generate this dll.

7. Launch Visual Studio Community 2013 and click on Open Project.

   Select <directory>\Circuits\FIR_Filter\To_be_completed\DMI_Code\FIR\code\pspSysCFIR.vcxproj

8. Change the default configuration on the top to Release x64, and Build the solution to verify that there are no build issues.

   NOTE: There are many warnings, but you can omit them.

9. Edit SysCFIR.cpp

```
void SysCFIR::entry() {
//    const sc_uint<8> coef[5] = { 18, 77, 107, 77, 18 };
//    sc_int<16> taps[5];
//    output.write(0);
//    wait();
//    while (true) {
//       for (int i = 4; i > 0; i--) {
//           taps[i] = taps[i - 1];
//       }
//       taps[0] = input.read();
//       sc_int<16> value;
//       for (int i = 0; i < 5; i++) {
//           value += coef[i] * taps[i];
//       }
//       output.write(value);
//
//       FILE* fp = fopen("out.vcd", "a");
//       fprintf(fp, "\n%d %d %d", value);
//       fclose(fp);
//
//       cout << "Time[" << sc_time_stamp() << "]  Value[0x"<< he
//       wait();
//    }
}
```

Search for SysCFIR::entry function in SysCFIR.cpp and uncomment the sample code inside the function clicking on Edit → Advanced → Uncomment. This code implements an FIR filter using SystemC.

10. Edit pspSysCFIR.cpp:

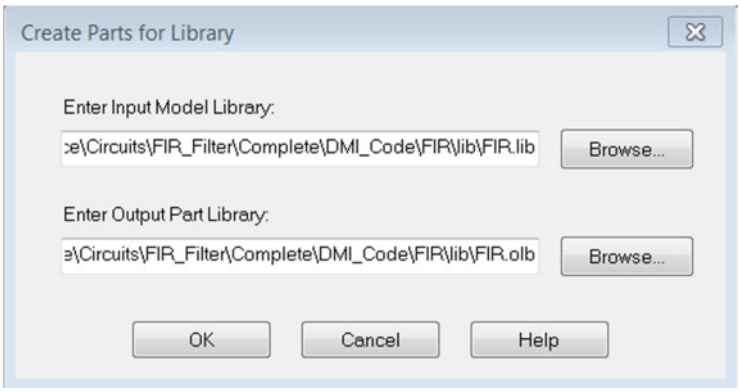
```
pspSysCFIR::pspSysCFIR(const char* pInstName, void*pRef){
    mRef = pRef;
    mInstName = pInstName;
    mPortCount = 33;
    mInputPortCount = 17;

    m_SysCFIR = new SysCFIR(pInstName);
m_SysCFIR->CLK(sysCsig_CLK);
m_SysCFIR->reset(sysCsig_reset);  <- Add it
m_SysCFIR->input(sysCsig_input);
m_SysCFIR->output(sysCsig_output);

}
```

11. Build the solution again to generate the PSpice-DMI dll.

12. Open Model Editor from PSpice Accessories, click File→ Open and load FIR.lib from

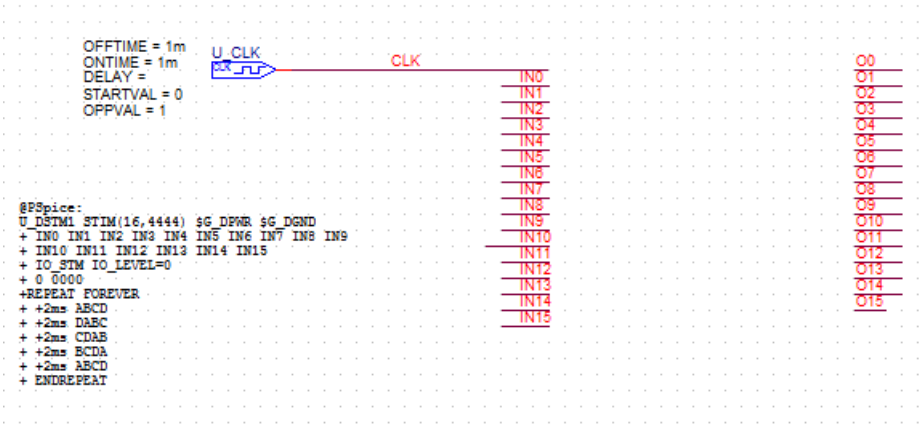
    <directory>\Circuits\FIR_Filter\To_be_completed\DMI_Code\FIR\lib

13. Click File → Export to Part Library and click OK.

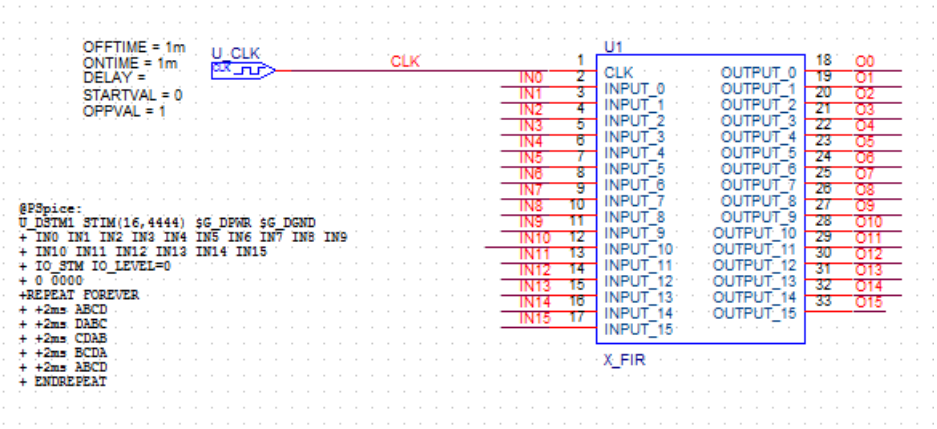


14. Copy FIR.lib and FIR.olb files and paste them in <directory>\Circuits\FIR_Filter\To_be_completed\Circuit\Library

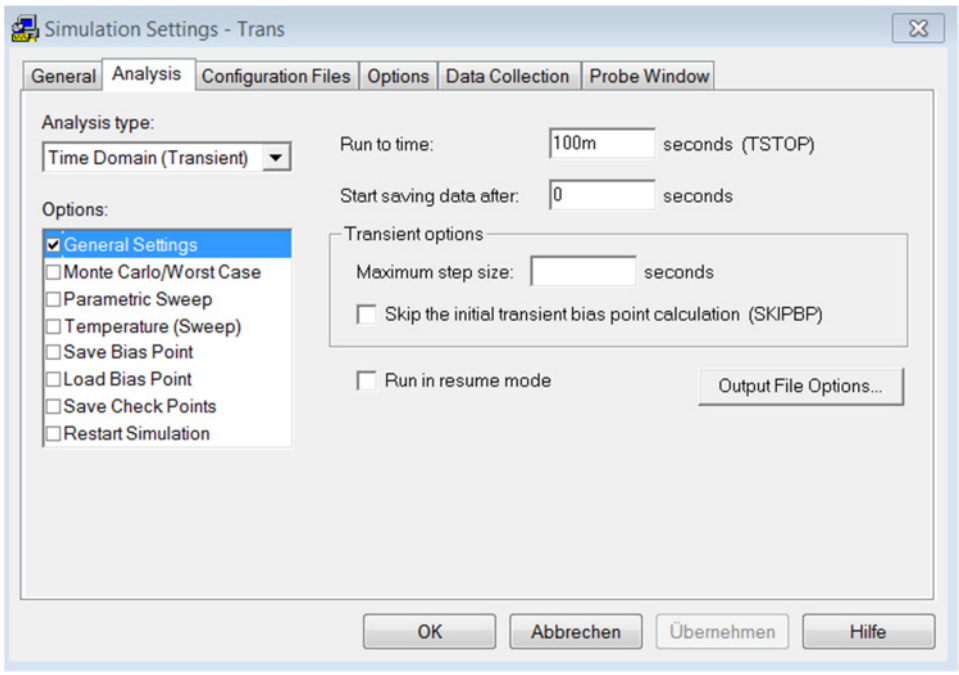
15. Launch OrCAD Capture and open the project located in <directory>\Circuits\FIR_Filter\To_be_Completed\Circuit



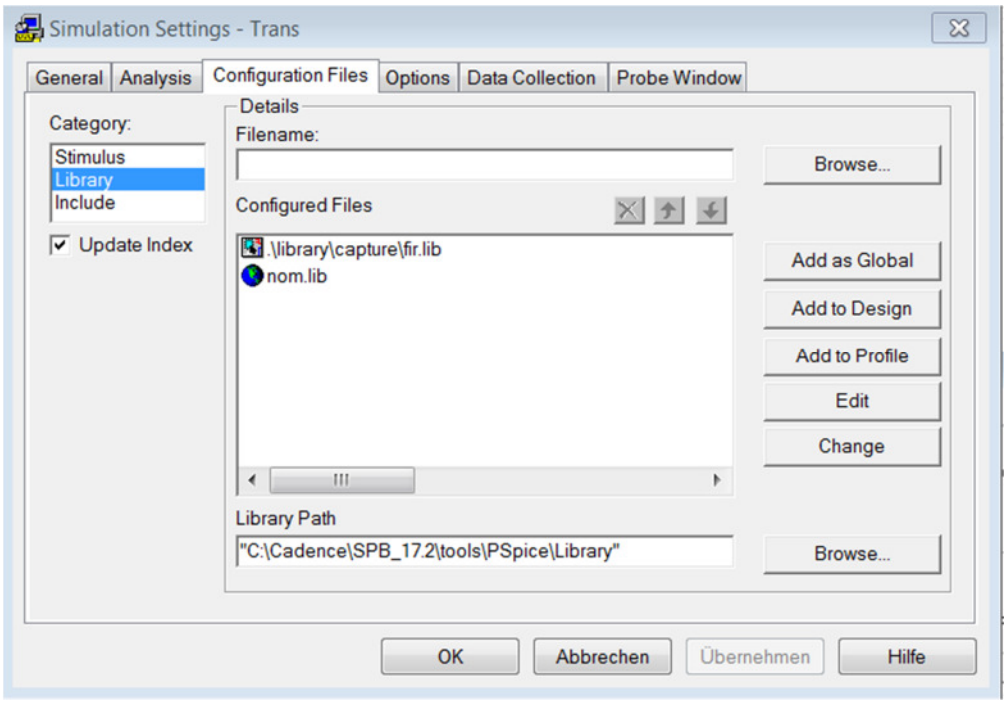
16. Click on Add Library, select FIR.olb and place the FIR component:



17. Create a new Simulation Profile called Trans.
18. Complete the new Simulation Settings with this values:



19. Click on Configuration Files and add to Design the library you have just created:



20. Copy the DLL from

    <directory>/Circuits/FIR_Filter/To_be_completed/DMI_Code/FIR/code/x64/Release

    And copy it in

    <directory>/Circuits/FIR_Filter/To_be_completed/Circuit/FIR_Filter-PSpiceFiles/SCHEMATIC1/Trans

21. Simulate.

You will get this error:

```
.subckt X_FIR CLK input_0 input_1 input_2 input_3 input_4 input_5 input_6 input_7 input_8 input_9 input_10 input_11 input_12 input_13
$
ERROR(ORPSIM-16366): Line too long. Limit is 132 characters.
+ OPTIONAL: DPWR=$G_DPWR DGND=$G_DGND
+ PARAMS:
.model FIR_TIMING ugate (
+ tplhmn=6ns tplhty=9ns tplhmx=15ns
+ tphlmn=6ns tphlty=10ns tphlmx=15ns
+ )
U1 LOGICEXP( 17, 16 ) DPWR DGND
+ CLK input_0 input_1 input_2 input_3 input_4 input_5 input_6 input_7 input_8 input_9 input_10 input_11 input_12 input_13 input_14 in
$
ERROR(ORPSIM-16366): Line too long. Limit is 132 characters.

Index has 1 entries from 1 file(s).
```

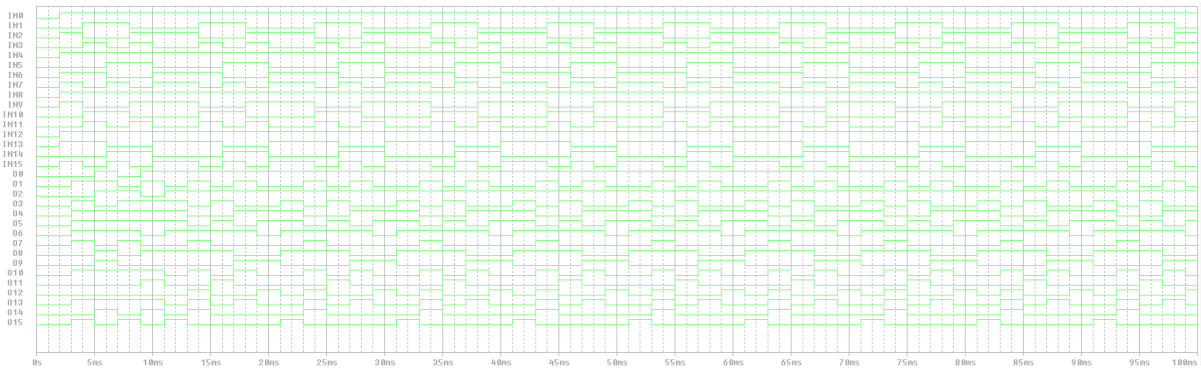
To solve it, RMB on the FIR component and click on Edit PSpice Model.

22. Make next modifications and do not forget to write down + at the beginning of the new line:

```
.subckt X_FIR CLK input_0 input_1 input_2 input_3 input_4 input_5 input_6 input_7 input_8 input_9 input_10
+input_11 input_12 input_13 input_14 input_15 output_0 output_1 output_2 output_3 output_4 output_5 output_6
+output_7 output_8 output_9 output_10 output_11 output_12 output_13 output_14 output_15
+ OPTIONAL: DPWR=$G_DPWR DGND=$G_DGND
+ PARAMS:
.model FIR_TIMING ugate (
+ tplhmn=6ns tplhty=9ns tplhmx=15ns
+ tphlmn=6ns tphlty=10ns tphlmx=15ns
+ )
U1 LOGICEXP( 17, 16 ) DPWR DGND
+ CLK input_0 input_1 input_2 input_3 input_4 input_5 input_6 input_7 input_8 input_9 input_10 input_11
+input_12 input_13 input_14 input_15 output_0 output_1 output_2 output_3 output_4 output_5 output_6 output_7
+output_8 output_9 output_10 output_11 output_12 output_13 output_14 output_15
+ FIR_TIMING IO_STD
+ C_MODEL: FIR.dll SysCFIR
+ PARAMS:
.ends
```

23. Close and click OK to save.
24. Simulate again and analyze the results (plot each input and each output).

NOTE: Notice that the input signal is coming from text format placed on the Schematic.



3.3 Noise Filter using a MATLAB Block

This module explains a simple example of Analog Behavioural Model imported into PSpice as a DMI Model. This module takes the example of a MATLAB averaging filter to demonstrate this.

The details of generating the code may be found at

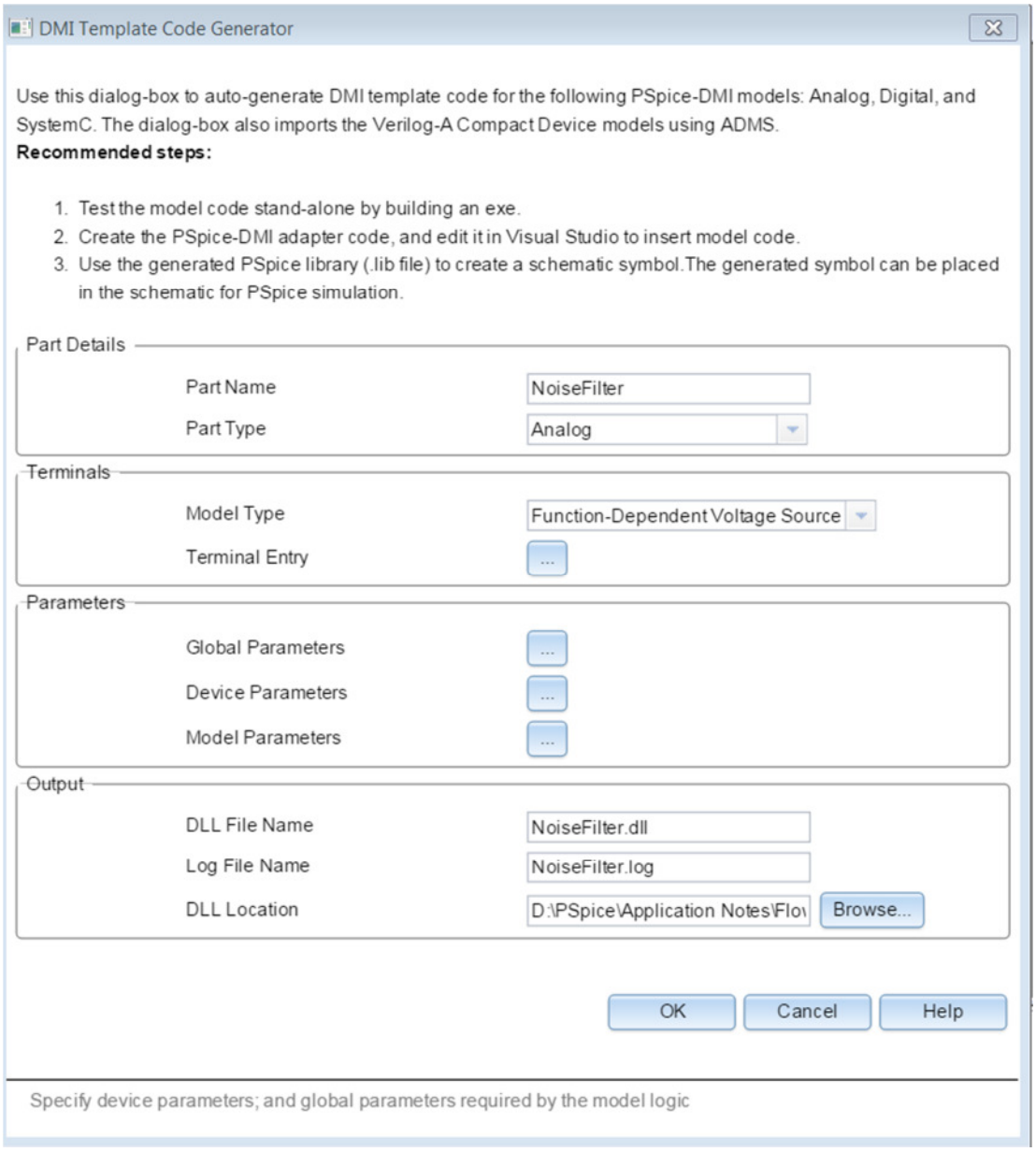
http://www.mathworks.com/help/coder/examples/averaging-filter.html?prodcode=ME&language=en

This Lab demonstrates:

- An example of MATLAB generated code imported to PSpice as a DMI model.
- Generation of a template code for an Analog behavioural model and its use in a PSpice simulation.

Steps:

1. Launch Model Editor
2. Select Menu Item Model → DMI Template Code Generator
3. Enter the data as follows:

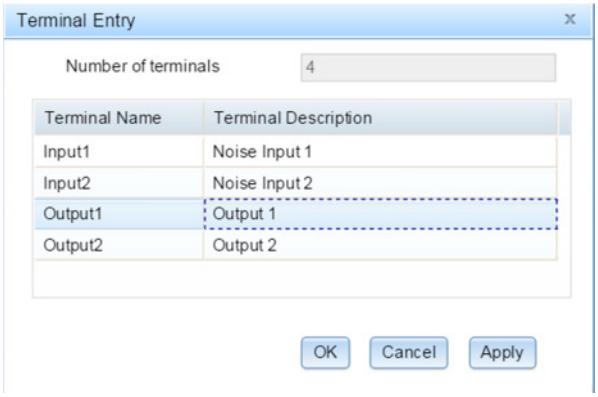


Click on Browse to save the code in

<directory>\Circuits\MATLAB_Block_Simulation\To_be_combpleted\DMI_Code

Do **NOT** click on OK.

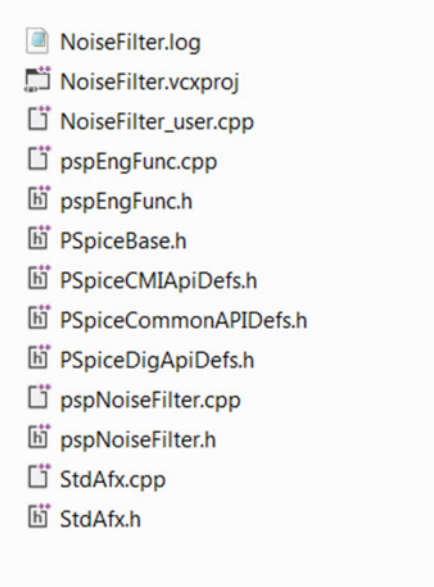
4. Click on Terminal Entry – for the selected predefined model type, the number of terminals is fixed at 4.



5. Click OK and generate the code.
6. Verify that the code and the lib folder are created correctly:

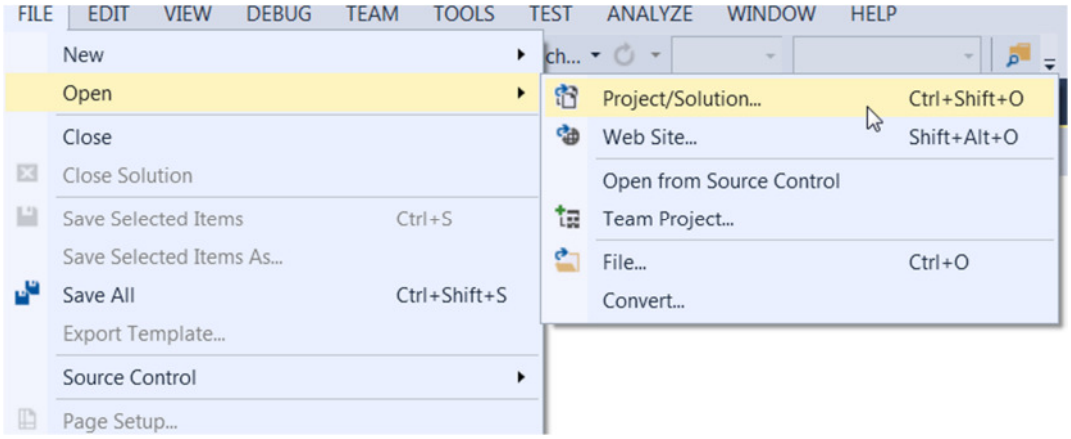
<directory>\Circuits\MATLAB_Block_Simulation\To_be_completed\DMI_Code\NoiseFilter\code

NOTE: the DLL will not be created until you compile the code in Visual Studio.



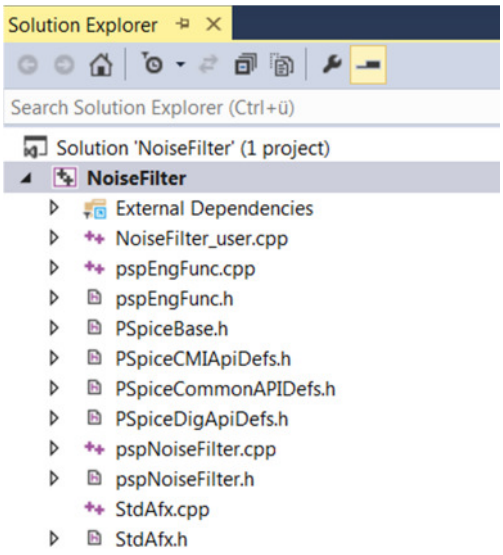
<directory>\Circuits\MATLAB_Block_Simulation\To_be_completed\DMI_Code\NoiseFilter\lib

7. Launch Visual Studio Community 2013 and open the file NoiseFilter.vcxproj

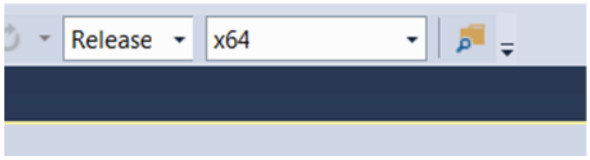


The project is located in

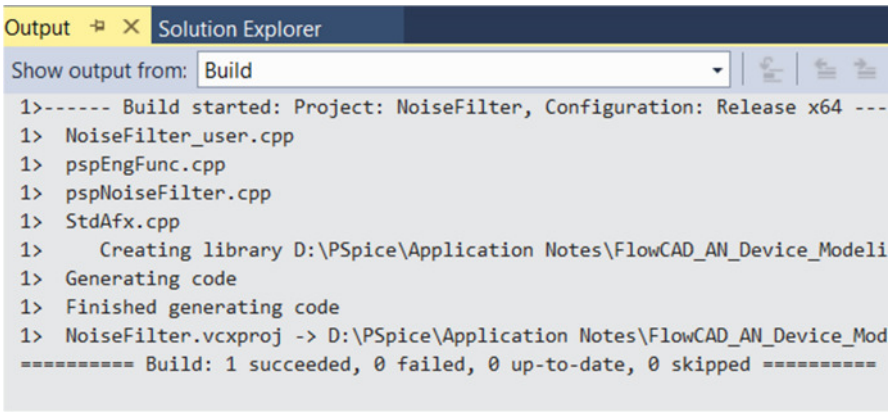
<directory>\Circuits\MATLAB_Block_Simulation\To_be_completed\DMI_Code\NoiseFilter\code



8. Change the configuration on the top as follows:



9. Select Menu Item Build/Build Solution and verify that the Build completes successfully.



10. The PSpice-DMI template code is ready. Now, the model behaviour for the exported averaging filter (using C-Coder in MATLAB) needs to be inserted in the code.

11. Open NoiseFilter_user.cpp for editing:

    Under the line:
    ```
    #include "pspNoiseFilter.h"
    ```

    Add the text:

    ```
    extern "C" {
    #include "../../averaging_filter/averaging_filter.h"
    }
    ```

    This includes the MATLAB generated header file so that its Averaging filter function can be accessed.

12. Edit the load function as follows:

    Under the line:

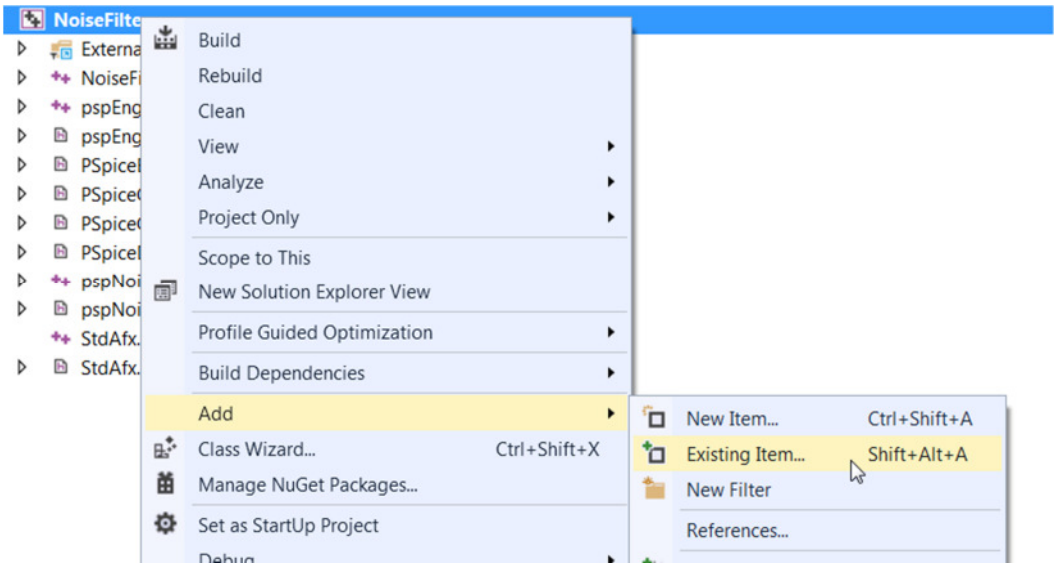
    ```
    double gain = 0.0;
    ```

    Add the text:

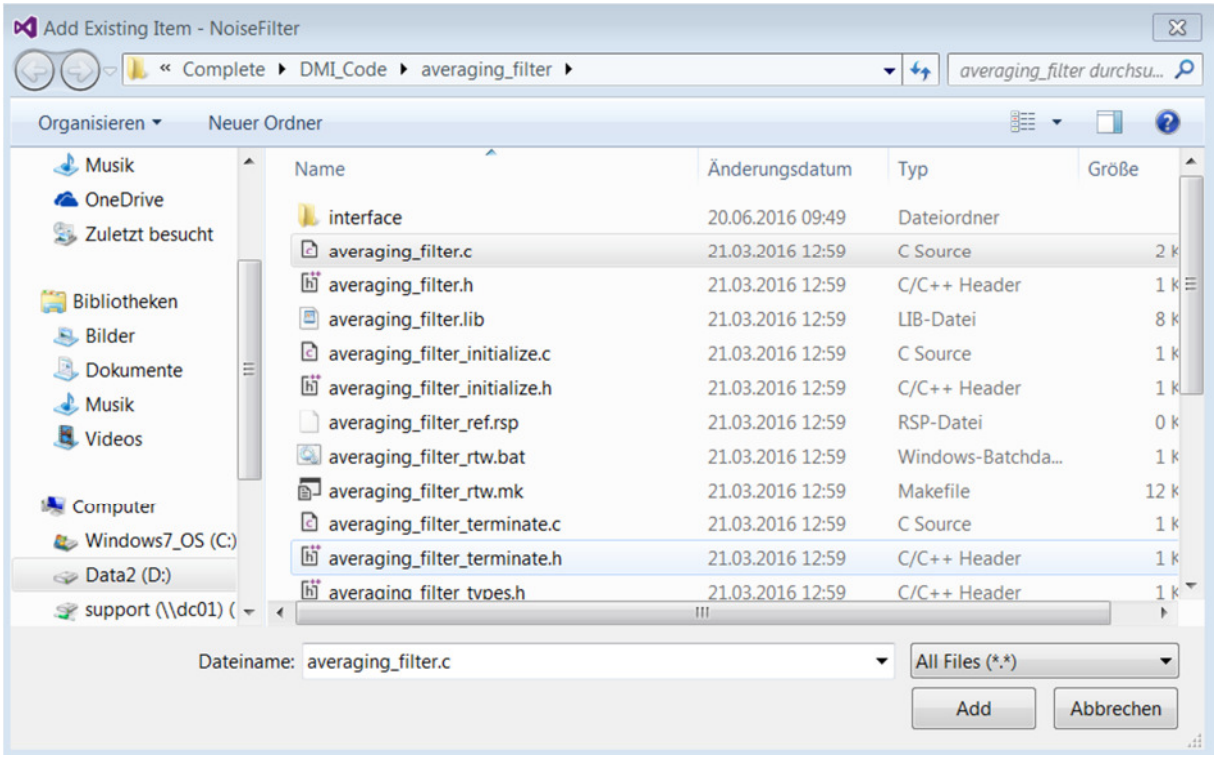
    ```
    // User Code
        if (pMode != MDTRAN) {
            for (int i = 0; i < 16 + MSTVCT; i++) {
                sv.x[i] = xVal;
            }
        }
        sv.y[0] = yVal = averaging_filter(xVal, sv.x);
     //
    ```

    This code will update the state vector with the latest input value, and call the MATLAB averaging_filter function to compute the gain.

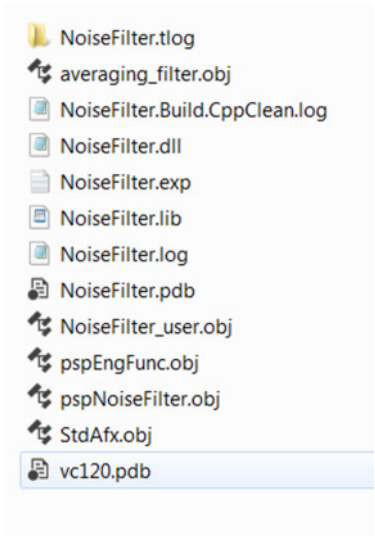
13. Finally, add the MATLAB averaging_filter file to the project. In the solution Explorer, RMB on NoiseFilter and select Menu Item Add – Existing Item



14. Browse to ../../averaging_filter and select the averaging_filter.c



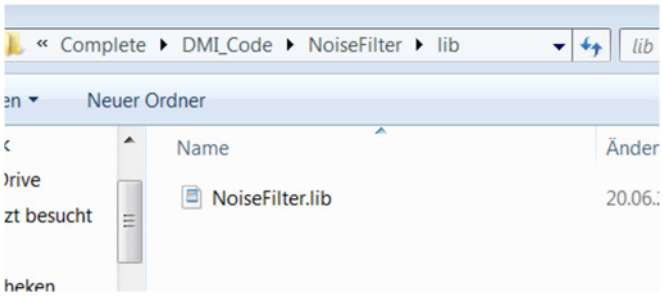
15. Build Solution from the Menu Item Build/Build Solution and verify it works.
16. Open the location <directory>Circuits\MATLAB_Block_Simulation\To_be_completed\DMI_Code\NoiseFilter\code\x64\Release and verify that the DLL and the PDB have been generated:



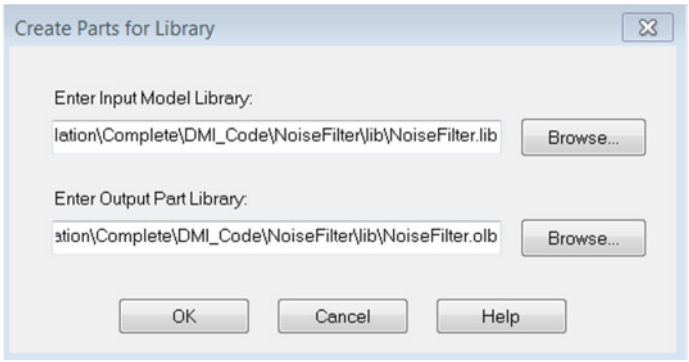
17. Launch Model Editor from PSpice Accessories.

18. Click Done

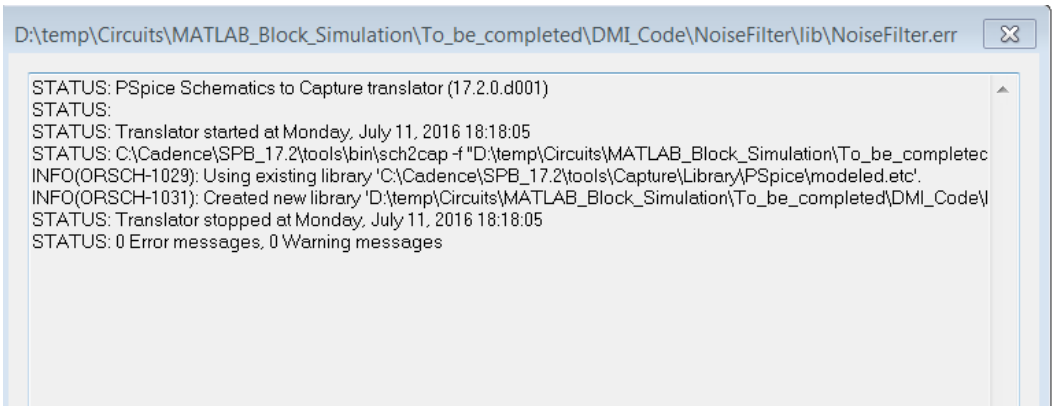
19. Click File→Open and select the library generated with DMI Template Code Generator. This is located in <directory>/circuits/MATLAB_Block_Simulation/To_be_completed/DMI_Code/NoiseFilter/lib



20. Select the component you have just loaded and click on File→Export to Part Library:



21. Click OK. A window indicating that everything worked properly should pop up:



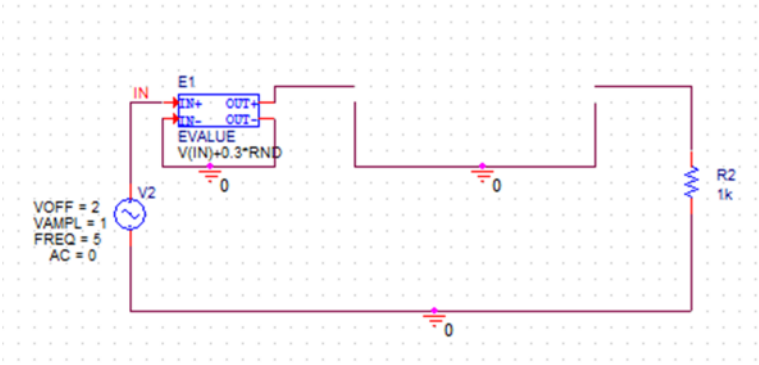
22. Copy NoiseFilter.lib and NoiseFilter.olb from

    <directory>/circuits/MATLAB_Block_Simulation/To_be_completed/DMI_Code/NoiseFilter/lib

    and paste them in

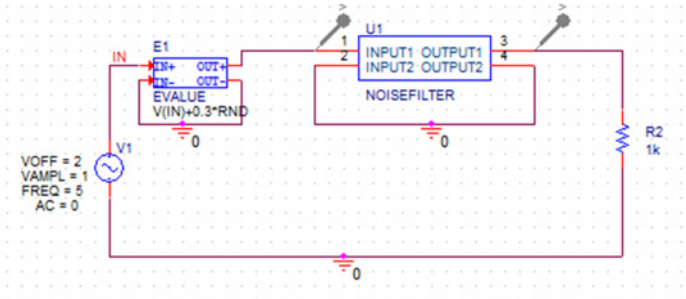
    <directory>/Circuits/MATLAB_Block_Simulation/To_be_completed/Circuit/Library/Capture

23. Launch OrCAD Capture and open the MatlabBlock design located in <directory>/Circuits/MATLAB_Block_Simulation/To_be_completed/Circuit

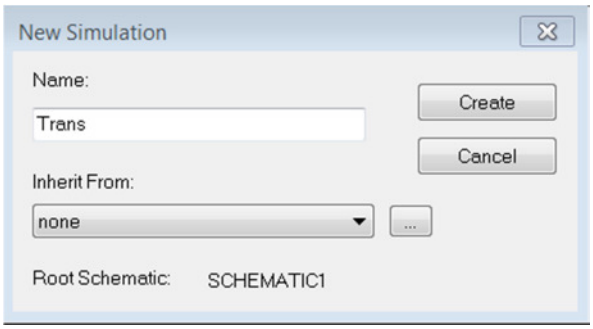


24. Click on Add Library, select the NoiseFilter.olb symbol and place it in the schematic:

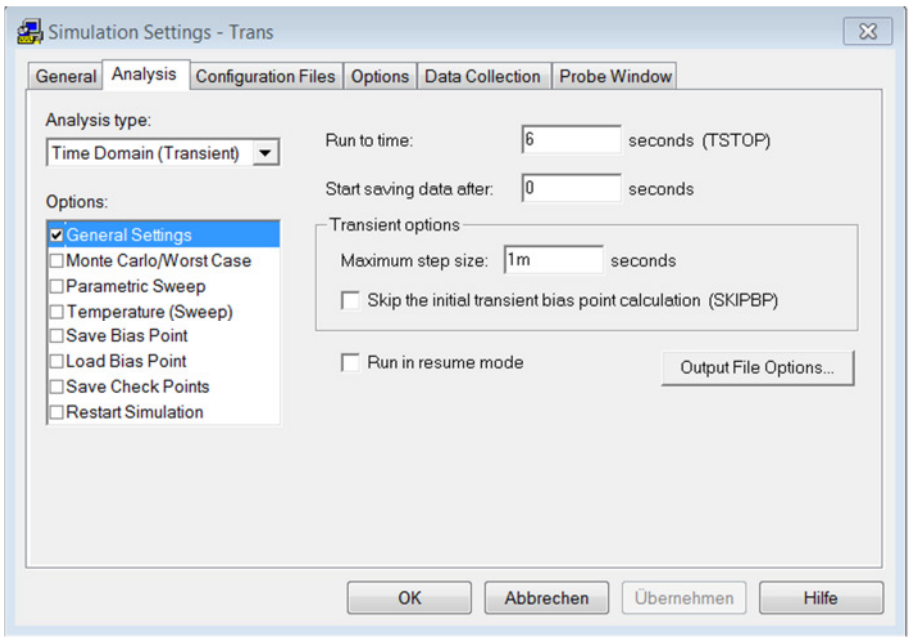
    <directory>/circuits/MATLAB_Block_Simulation/To_be_completed/Circuit/Library/Capture



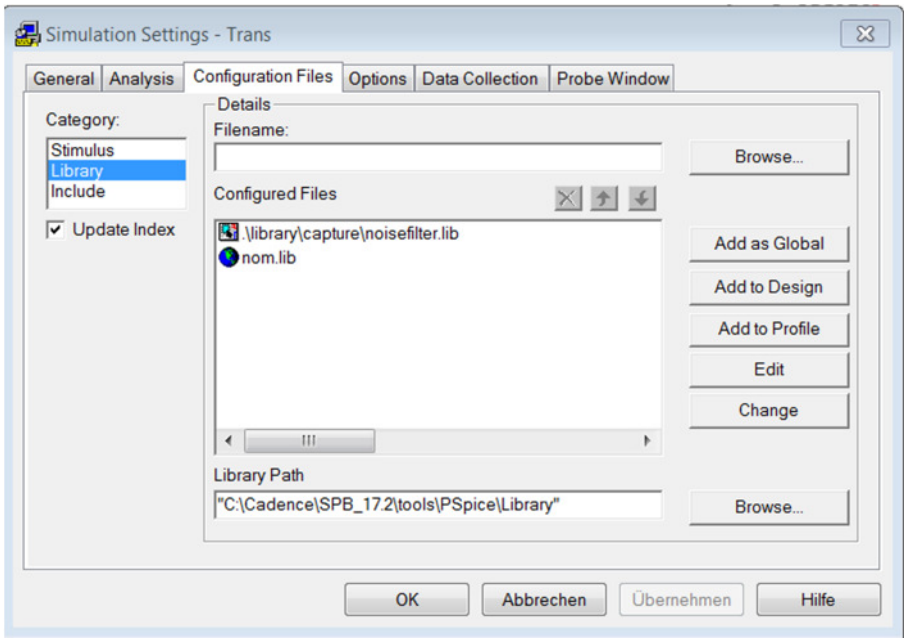
25. Create a new Simulation Profile and name it Trans.



26. Complete the values like in the image:



27. Without closing the Settings select Configurations Files and add to Design the NoiseFilter.lib:



28. Click OK.
29. Before simulating, place the created NoiseFilter.dll and its .pdb in the Simulation Profile Folder.

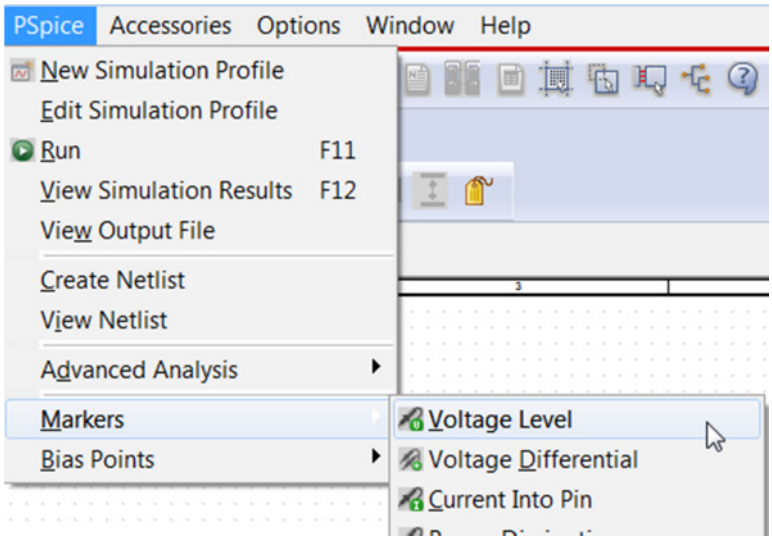
    DLL Location:

    <directory>/Circuits/MATLAB_Block_Simulation/To_be_completed/DMI_Code/NoiseFilter/code/x64/Release

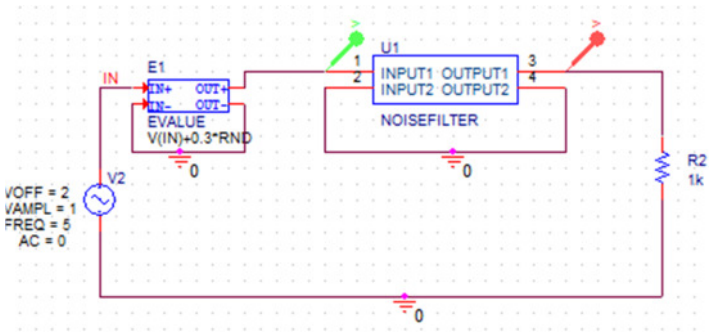
    Simulation Profile Location:

    <directory>/Circuits/MATLAB_Block_Simulation/To_be_completed/Circuit/MatlabBlock-PSpiceFiles/SCHEMATIC1/Trans

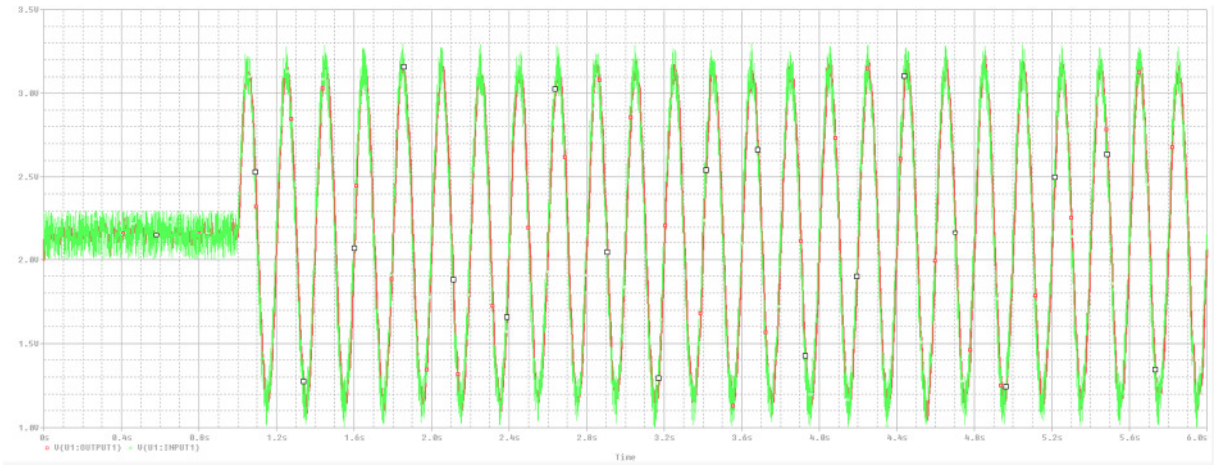
30. Simulate and verify it works:
31. Analyze the results. Click on PSpice → Markers → Voltage Level.



32. Place the markers like in the image:



33. Open Probe Window:



3.4 Capacitor behaviour analysis defined using VerilogA-ADMS

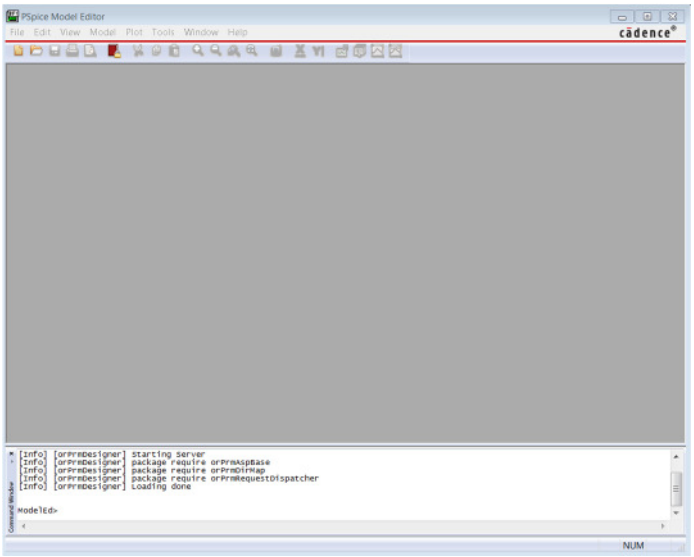
This module explains the import of a VerilogA file and its translation to a DMI model. VerilogA import in PSpice is supported using ADMS parser – this is primarily useful for importing VerilogA compact models.

This Lab will demonstrate:

- An example of a capacitor model written in VerilogA, using 2 parameters to specify the capacitance value.
- Import of a VerilogA file using Model Editor and translation into PSpice-DMI model.
- Sample simulation and comparison of the results with a regular capacitor simulation.

Steps:

1. Launch Model Editor from PSpice Accessories:



2. Select Model → DMI Template Code Generator.
3. Enter Part Name as cap

   NOTE: Part Name should match the module name specified in the VerilogA file.

4. In the DMI Template code Generator UI, select Part Type as VerilogA-ADMS

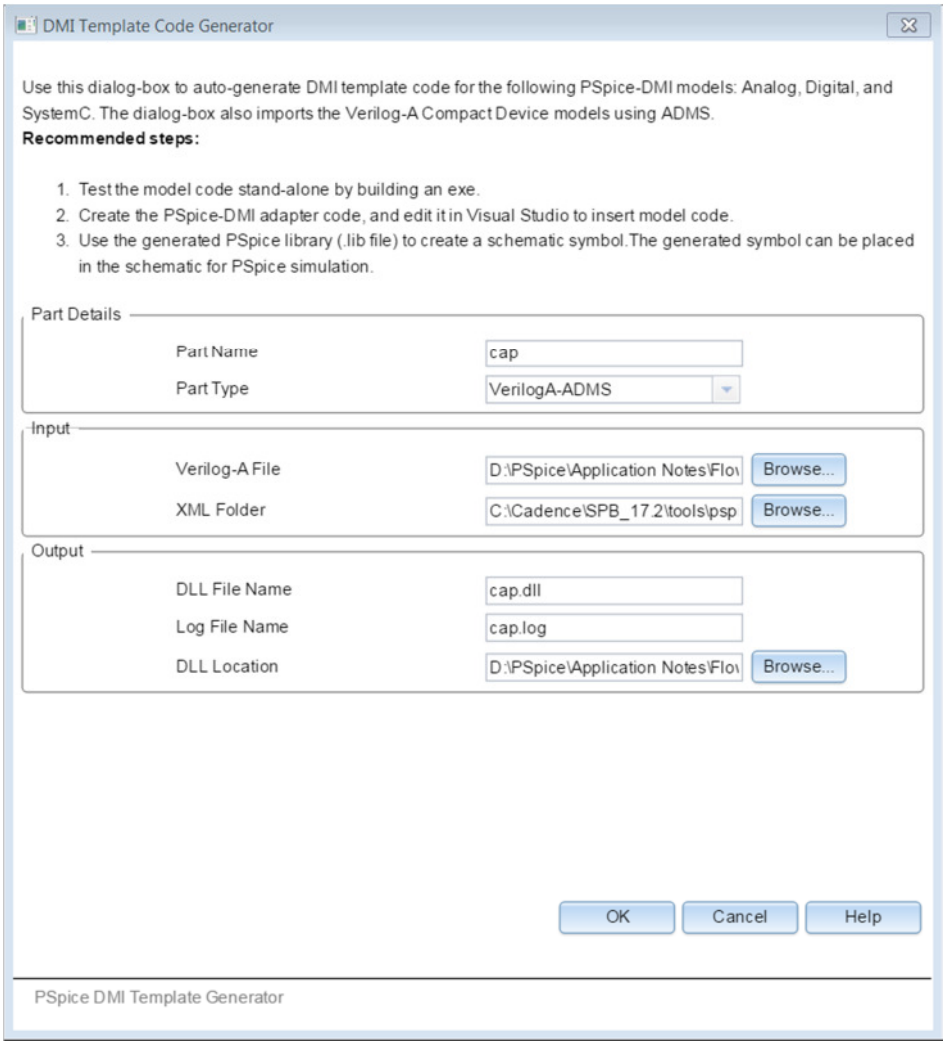
5. In Verilog-A File field, enter the path to the cap.va file located in the VerilogA_component folder:

   <directory>/Circuits/VerilogA_Capacitor/To_be_completed/VerilogA_Component

   NOTE: The XML Folder is automatically selected by the tool. You do not have to browse anything.

6. In the Output, select only the DLL Location to:

   <directory>/Circuits/VerilogA_Capacitor/To_be_completed/DMI_Code

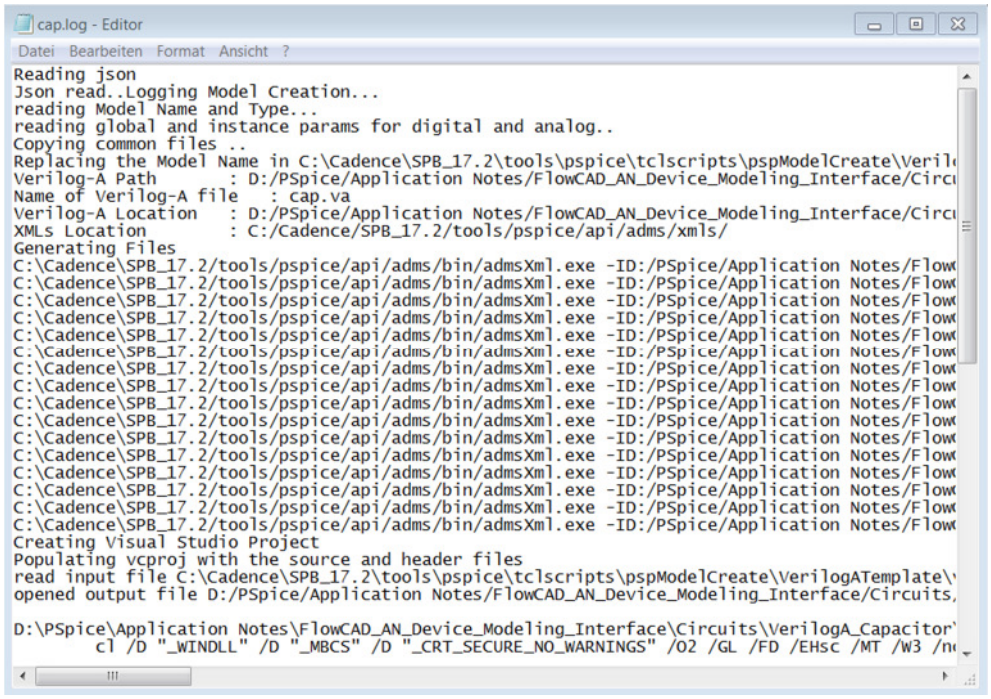


The cap.va is a VerilogA model for a capacitor which uses 2 parameters to define the capacitor values: C1 and C2.

```
`include "discipline.h"
module cap(p,n);
inout p,n;
electrical p,n;
parameter real c1=0 from [0:inf);
```

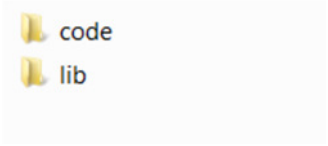
```
parameter real c2=0 from [0:inf);
analog
I(p,n) <+ ddt((c1+2*c2)*V(p,n));
endmodule
```

7. Click OK. The DMI model is auto-generated from the VerilogA file and a log file is generated listing the translation steps.

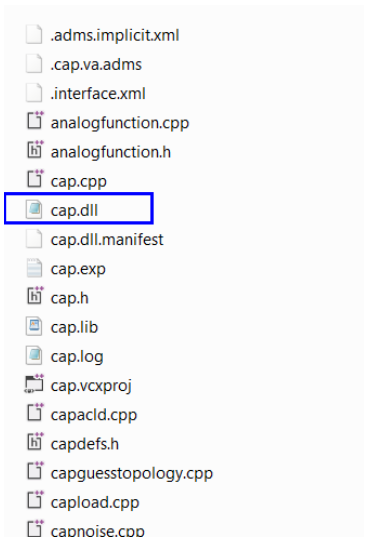


8. The generated code is automatically compiled using nmake. If there are no build errors, the dll and the lib files are generated, and can be directly used in a PSpice Simulation.

   <directory>/Circuits/VerilogA_Capacitor/To_be_completed/DMI_Code/cap/lib



- Dll file created in the folder code:



- Lib file created in the folder lib:

```
.subckt cap_sub p n PARAMS: c1=0 c2=0
Y1 p n CMI cap.dll cap_mod
.model cap_mod CMI cap c1={c1} c2={c2}
.ends
```

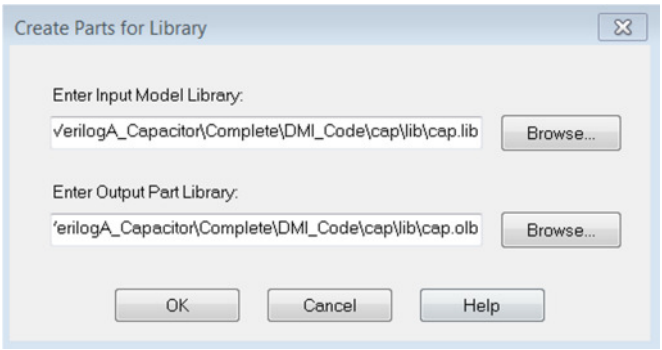
9. Copy the cap.lib and paste it in <directory>\circuits\VerilogA_Capacitor\To_be_completed\Circuit\Library\Capture in order to generate the symbol (olb) to be used in Capture.

10. Copy the DLL and paste it in <directory>\circuits\VerilogA_Capacitor\To_be_completed\Circuit\Library\DLL

11. Open Model Editor from PSpice Accessories
12. Click on File → Open and look for cap.lib in

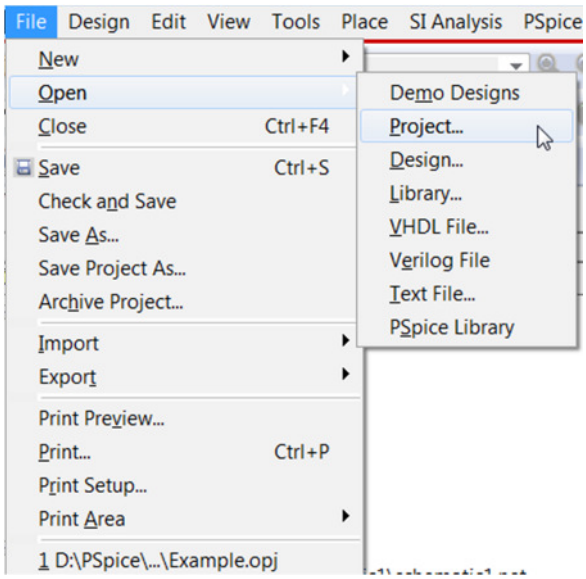
    <directory>\circuits\VerilogA_Capacitor\To_be_completed\Circuit\Library\Capture

13. Click on File → Export to part library

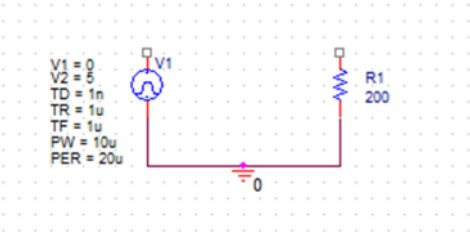


14. Now open in OrCAD Capture the example design where you are going to evaluate a capacitor from the default Cadence library and the capacitor you have just defined. Click on File→Open→Project and select the project located in

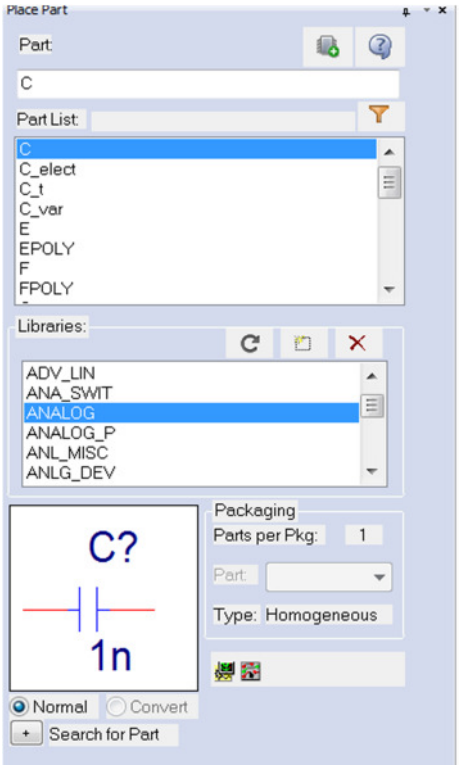
    <directory>\Circuits\VerilogA_Capacitor\To_be_completed\Circuit\Example.opj



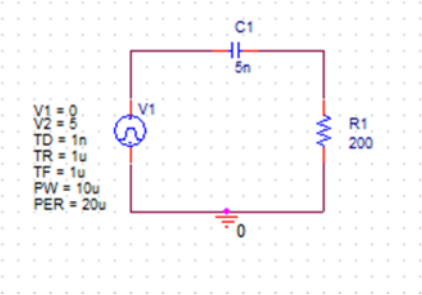
16. Make double click on Page1 from the schematic Cap:



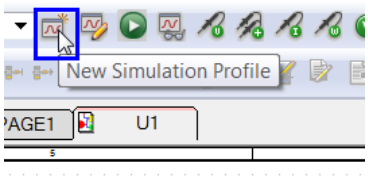
17. Click on Place Part and select C from the library analog.olb:



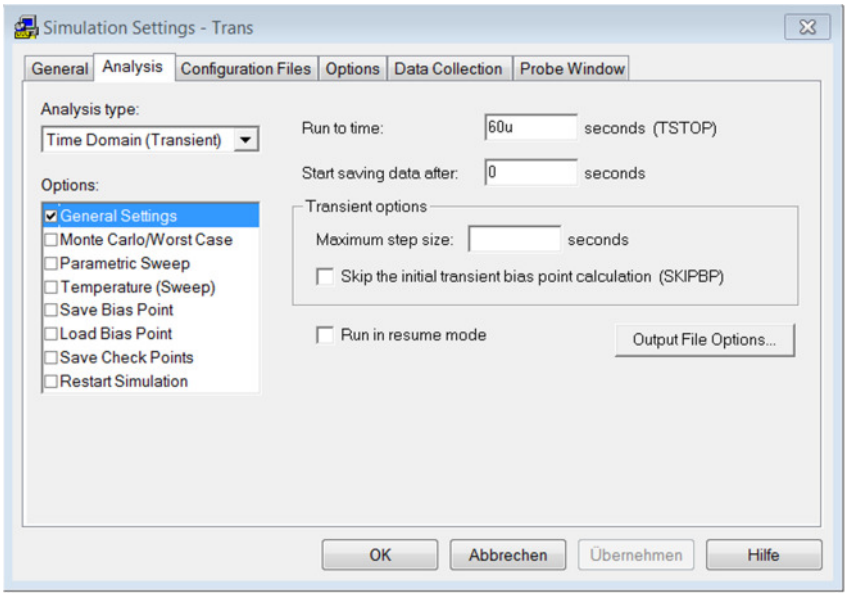
18. Place it in the schematic and change the value to 5n



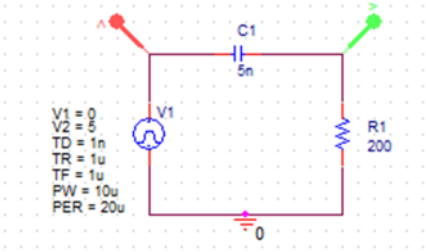
19. Create a new Simulation Profile and call it Trans:



25. Complete the Simulation profile with these values:

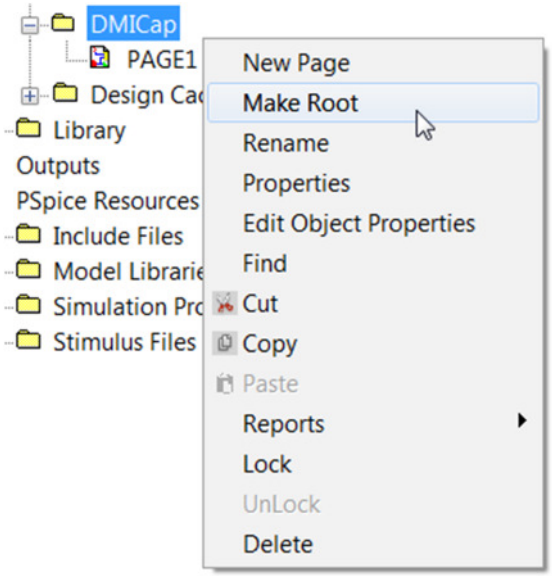


26. Click OK.
27. Run the simulation clicking on Play.
28. In OrCAD Capture click on PSpice → Markers → Voltage Level and place the markers as in the image:



29. Analyse the results:

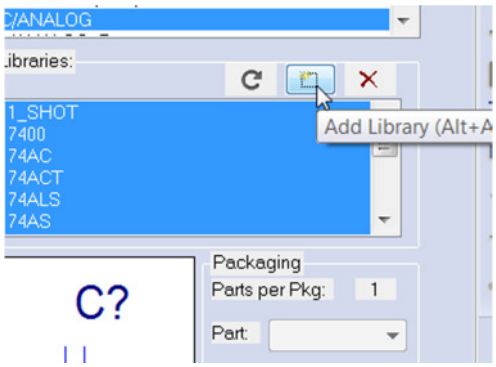
22. Come back to OrCAD Capture, select the DMICap Schematic and with RMB make it Root. Now you are going to simulate your Verilog-A Cap.



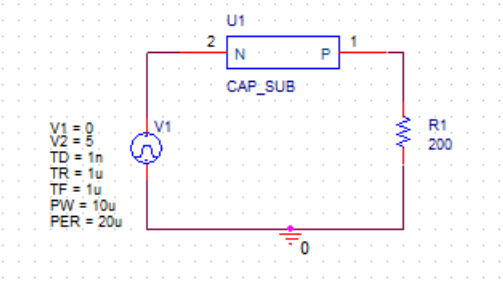
NOTE: If you have not saved, it will ask you automatically. Click on Save.

23. Click on Add Library and select the .OLB you have created

<directory>/Circuits/VerilogA_Capacitor/To_be_completed/Circuit/Library/Capture

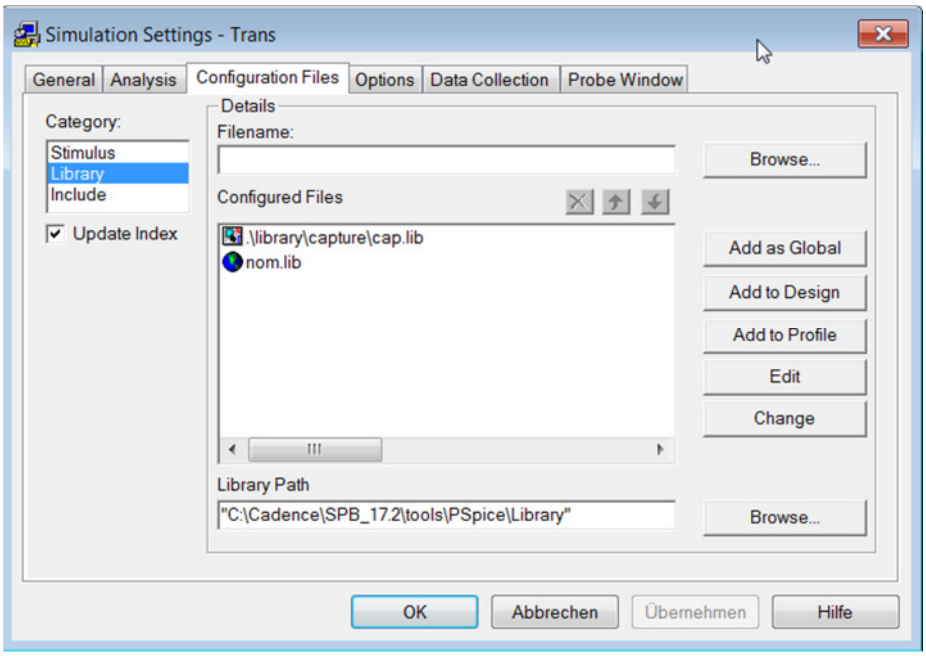


and place the capacitor changing the values of C1 and C2 to 1n. You will find these properties making double click on the DMICAP.



	A
	DMICap : PAGE1 : U1
BiasValue Power	0W
C1	1n
C2	1n
Color	Default
Designator	
Graphic	CAP_SUB.Normal
Designator	
Implementation	CAP_SUB
Implementation Path	
Implementation Type	PSpice Model
Location X-Coordinate	410
Location Y-Coordinate	140
Name	INS707
Part Reference	U1
PCB Footprint	
Power Pins Visible	
Primitive	DEFAULT
PSpiceTemplate	X^@REFDES %P %N @MO
Reference	U1
Source Library	C:\TEMPORAL\TEST\C
Source Package	CAP_SUB
Source Part	CAP_SUB.Normal
Value	CAP_SUB

24. Create a new Simulation profile called Trans, with the same values than before and click Configuration Files and Libraries:

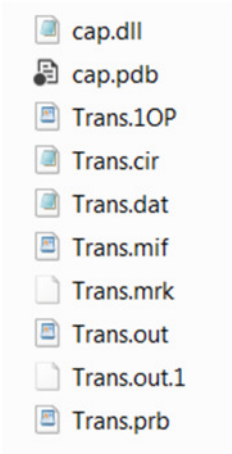


25. Add to Design the .lib generated you located in

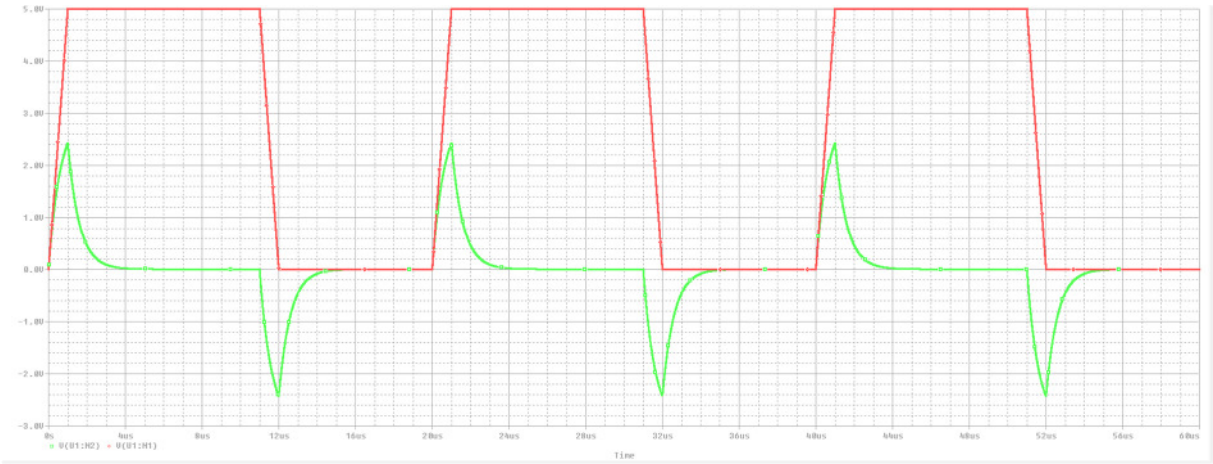
<directory>/Circuits/VerilogA_Capacitor/To_be_completed/Circuit/Library/Capture

26. Before simulating, you have to place the .DLL in the Simulation Profile Folder you have just created. Place it in:

<directory>\VerilogA_Capacitor\To_be_completed\Circuit\Example-PSpiceFiles\DMICap\Trans



27. Simulate and analyse the results placing the markers as done before:



3.5 Debugging

This section explains how the behaviour of DMI Models can be debugged on the circuit where it has been designed. There are two options to debug DMI models. To understand both of them, you will use the project that is located in

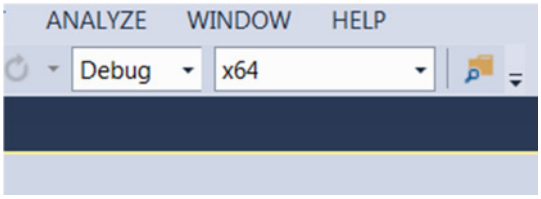
<directory>\Circuits\Debugging\Circuit\MatlabBlock.opj

Option 1: Setup path to psp_cmd.exe in Visual Studio

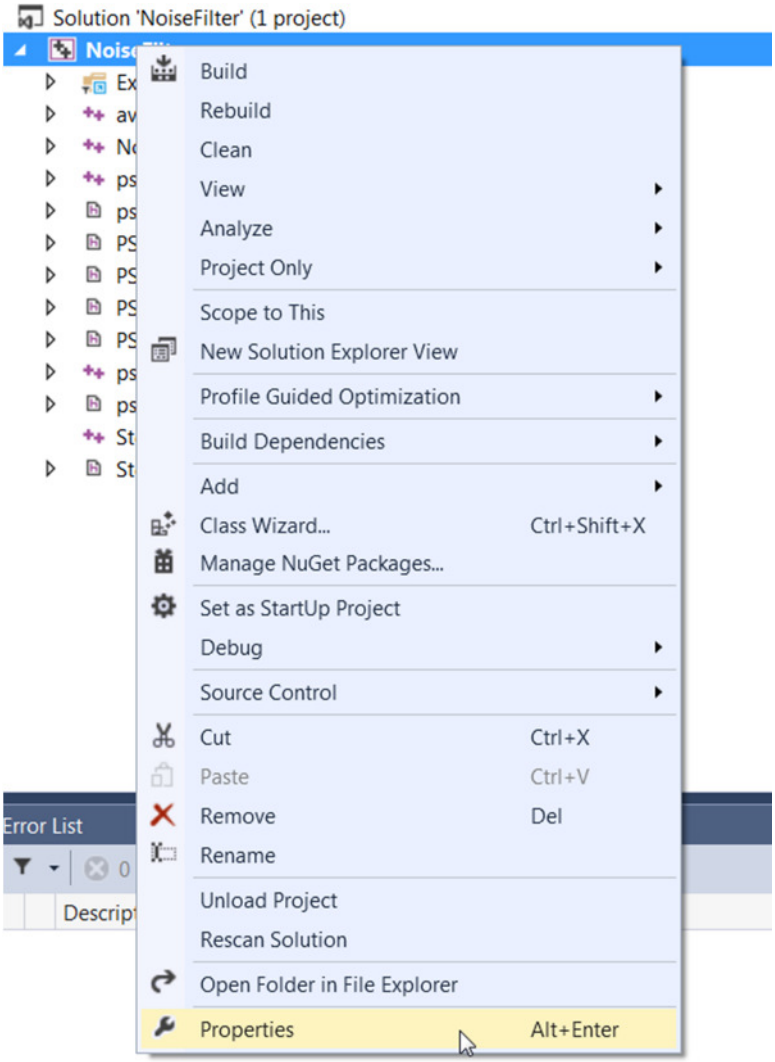
1. Open Visual Studio Community 2013
2. Click File→Open→Project/Solution and search for:

   <directory>\Circuits\Debugging\DMI_Code\NoiseFilter\code\noiseFilter.vcxproj

3. Ensure DEBUG profile is selected on the top:



4. Select the project with RMB and select properties:



5. A window pops up. Click on Debugging and setup three property values:

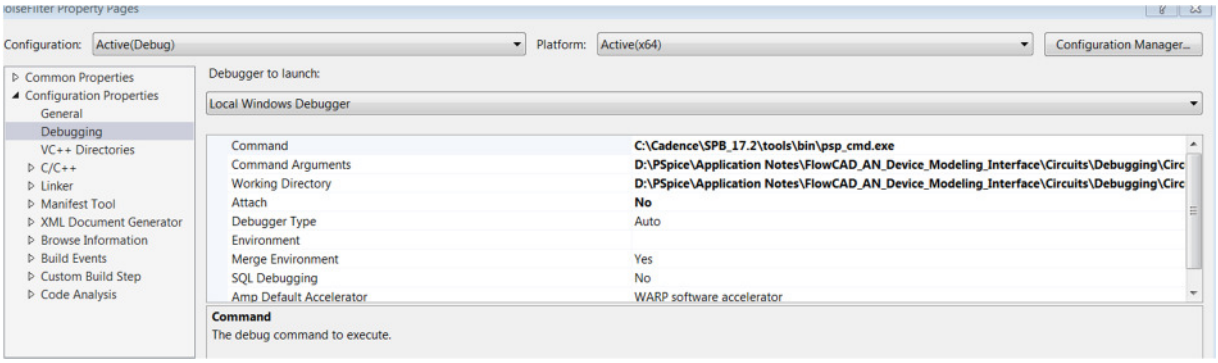
Command	Enter absolute path to psp_cmd.exe
Command Arguments	Enter absolute path to cir file which contains the C Model instance
Working Directory	Enter the absolute path to the directory which contains the cir file

For this example:

Command:
%CDSROOT%\tools\bin\psp_cmd.exe

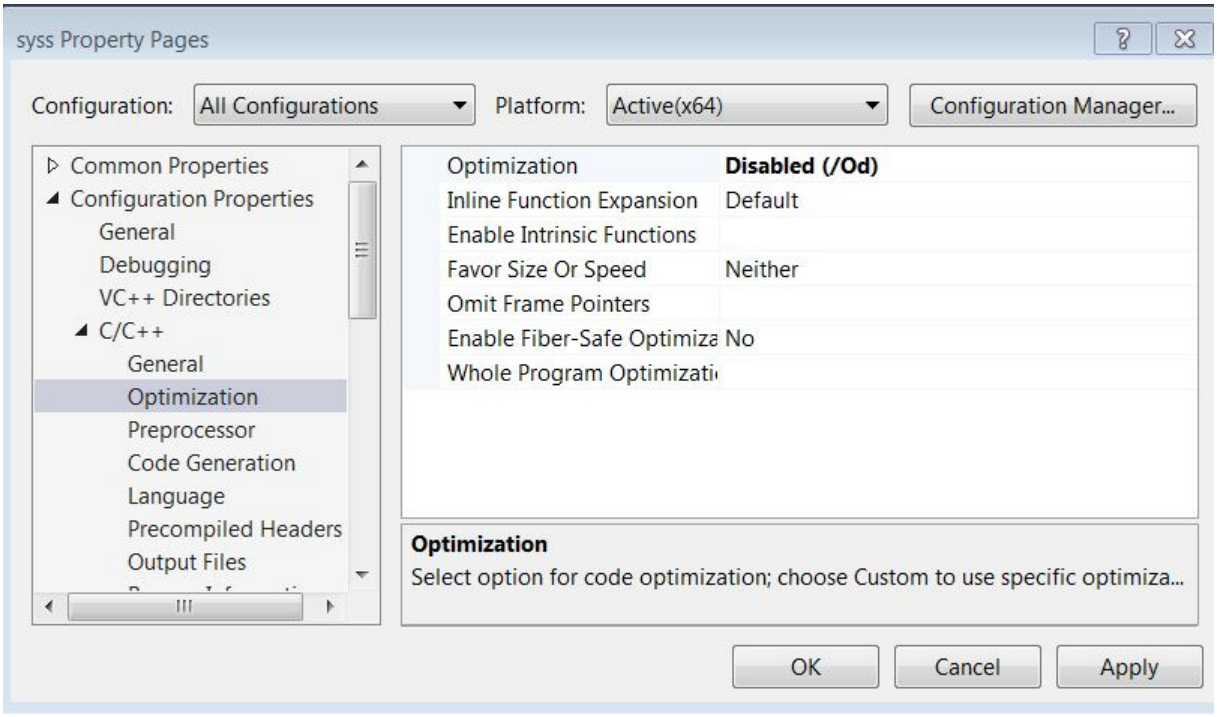
Command Arguments:
<directory>\Circuits\Debugging\Circuit\MatlabBlock-PSpiceFiles\SCHEMATIC1\Trans\Trans.cir

Working Directory:
<directory>\Circuits\Debugging\Circuit\MatlabBlock-PSpiceFiles\SCHEMATIC1\Trans

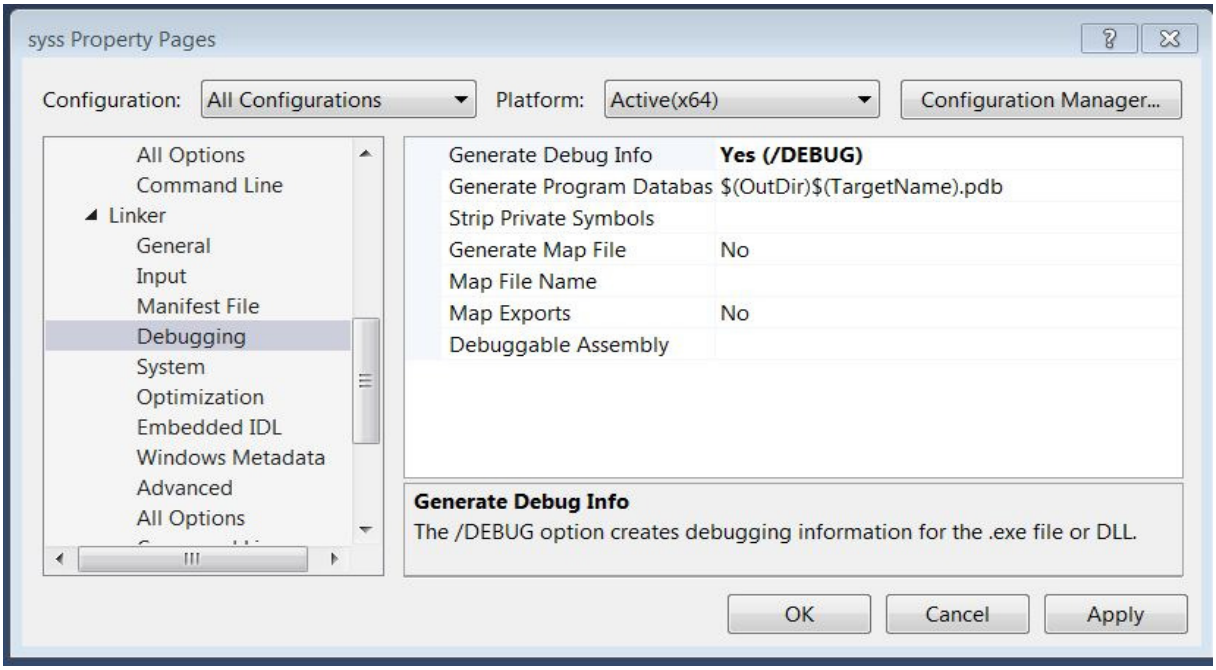


6. Verify that you have the same values in the next options:

   - Click on Optimization inside of C/C++

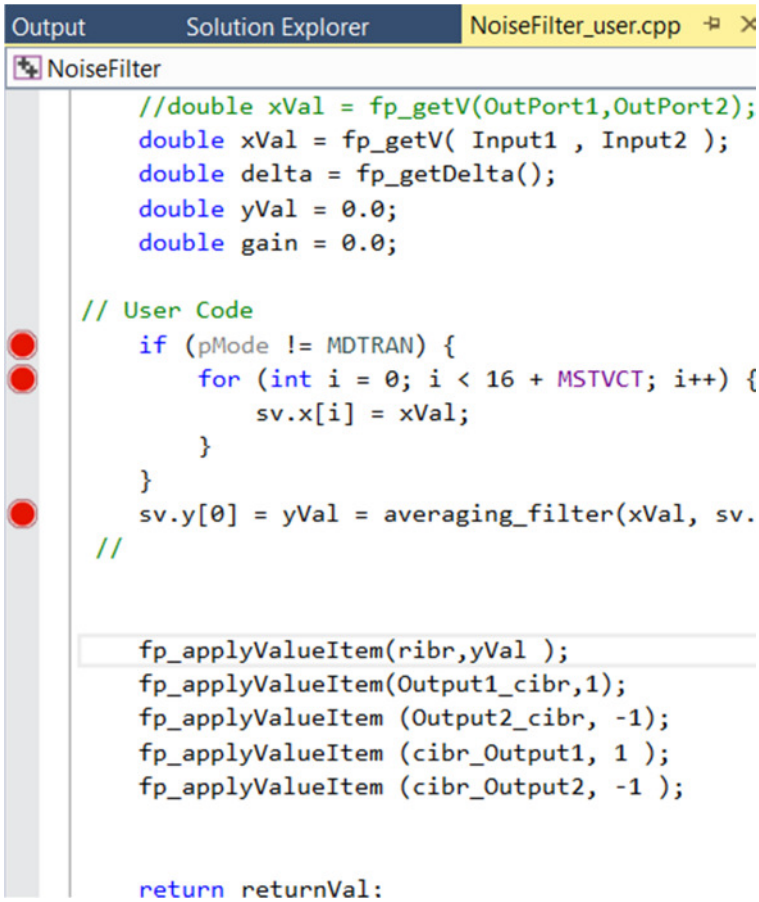


- Click on Debugging inside of Linker:

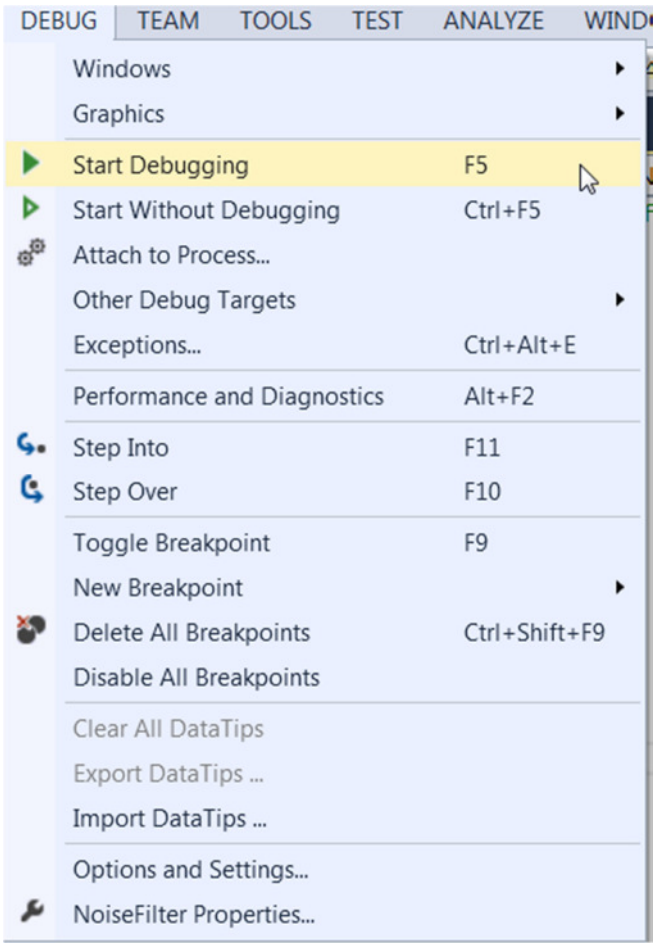


7. Click Apply and OK.

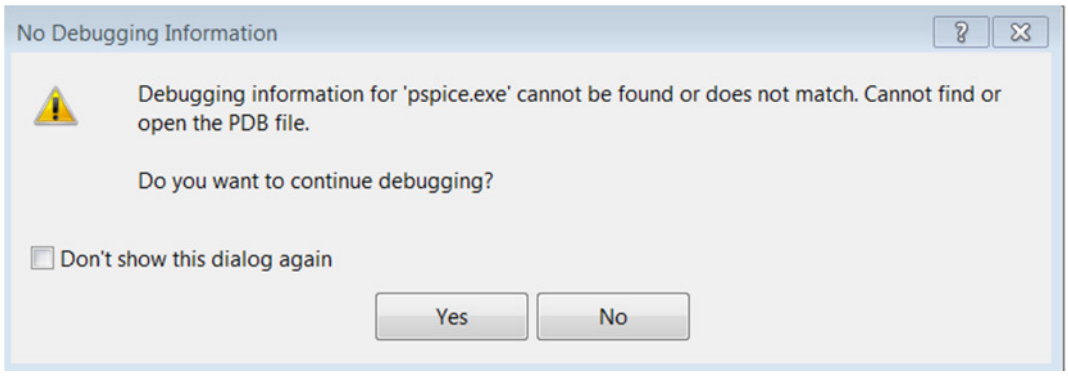
8. Open file NoiseFilter_user.cpp and add breakpoints clicking on the desired line:



9. Run the simulation in debug mode



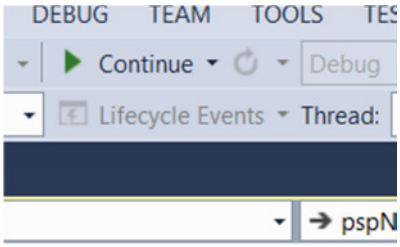
10. Choose Yes on the following message:



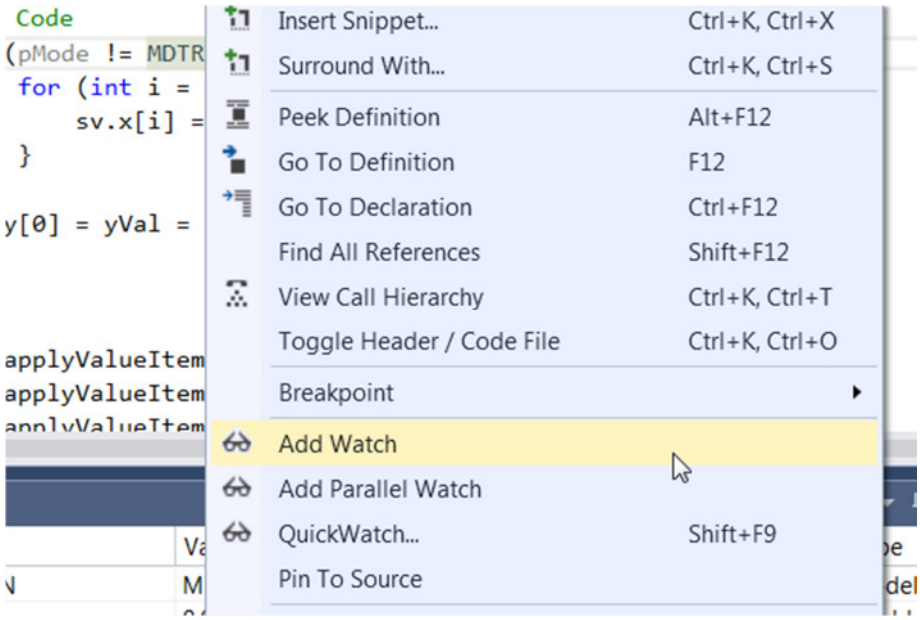
11. Observe that simulation stops at breakpoint:

```
    double yVal = 0.0;
    double gain = 0.0;

// User Code
    if (pMode != MDTRAN) {
        for (int i = 0; i < 16 + MSTVCT; i++) {
            sv.x[i] = xVal;
        }
    }
    sv.y[0] = yVal = averaging_filter(xVal, sv.x);
//
```

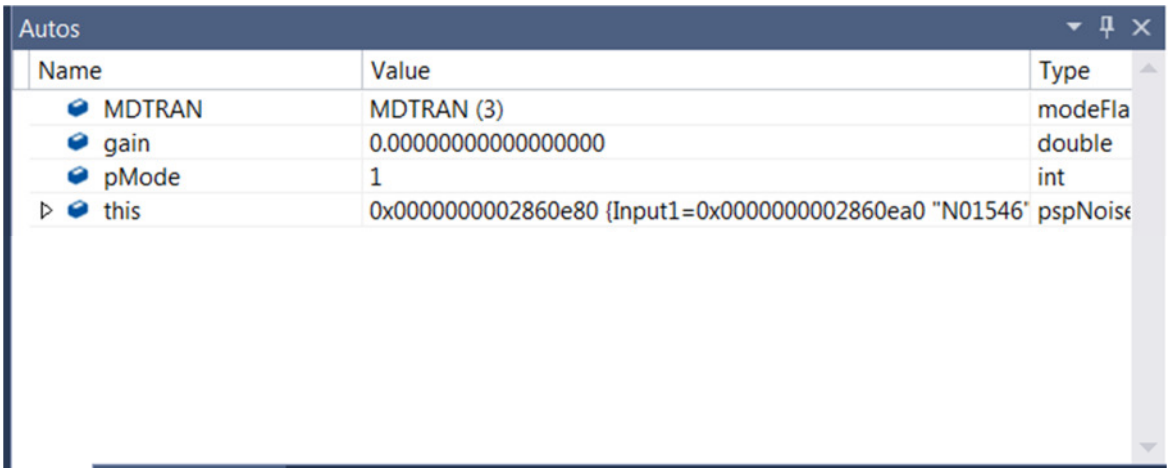
12. Click on Continue to jump from one breakpoint to another one.



13. Use Visual Studio Watch function to see any variable value



NOTE: If it is said that the project is out of date or you make many changes in the code, build the project again, copy .dll and .pdb and past them in the Simulation Profile Trans, so that you can debug properly.



Option 2: Attach Visual Studio to PSpice

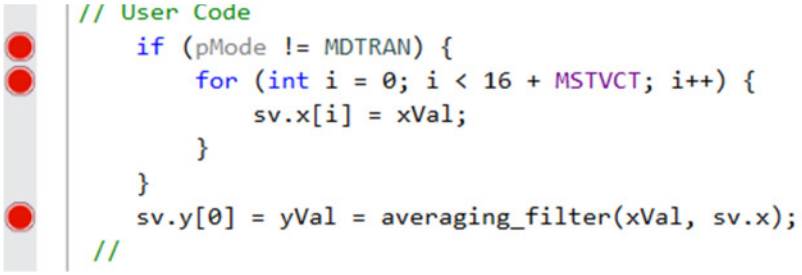
1. Open Visual Studio and load the previous project.

   <directory>\Circuits\Debugging\DMI_Code\NoiseFilter\code\NoiseFilter.vcxproj

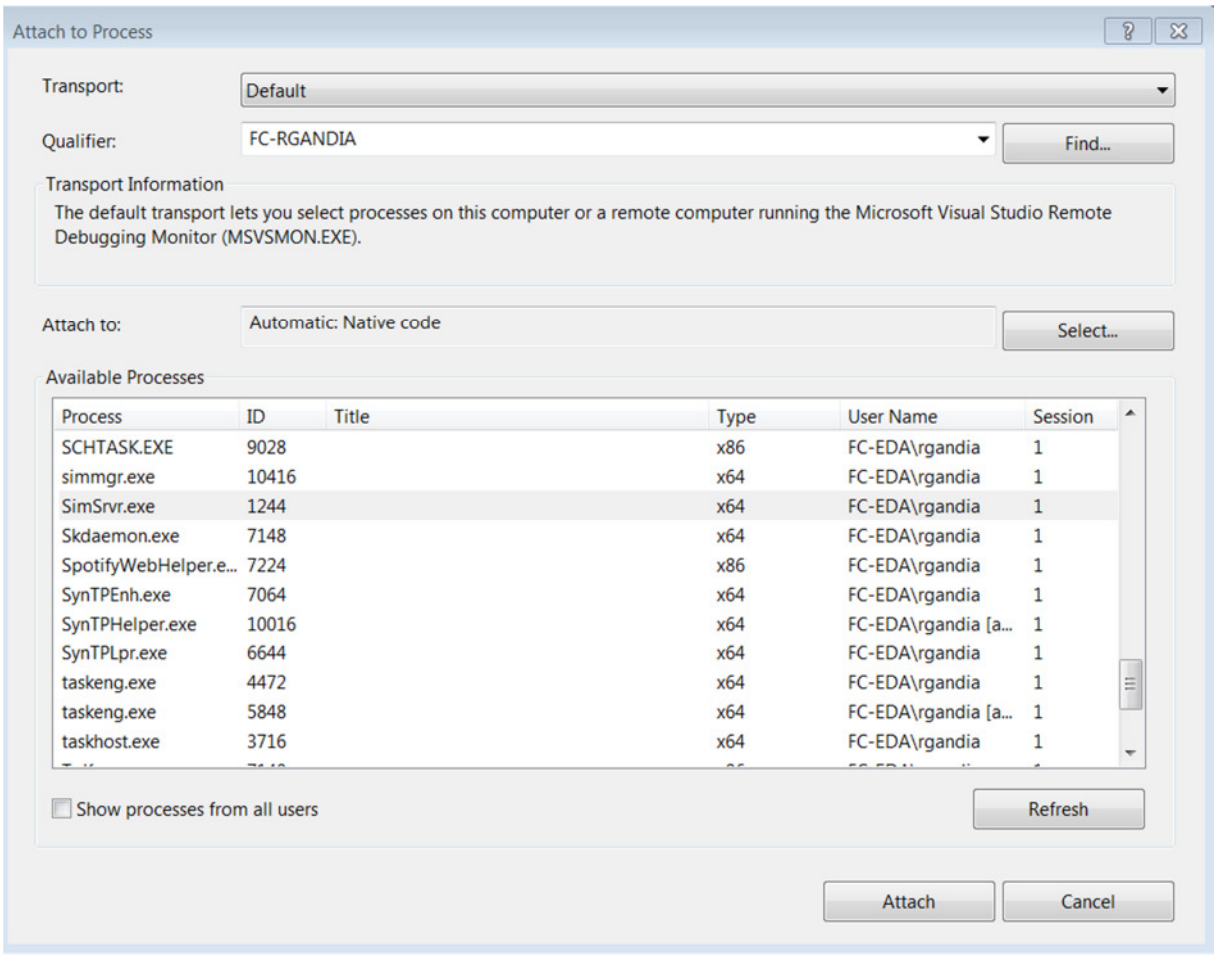
2. Open the project in OrCAD Capture

   <directory>\Circuits\Debugging\Circuit\MatlabBlock.opj

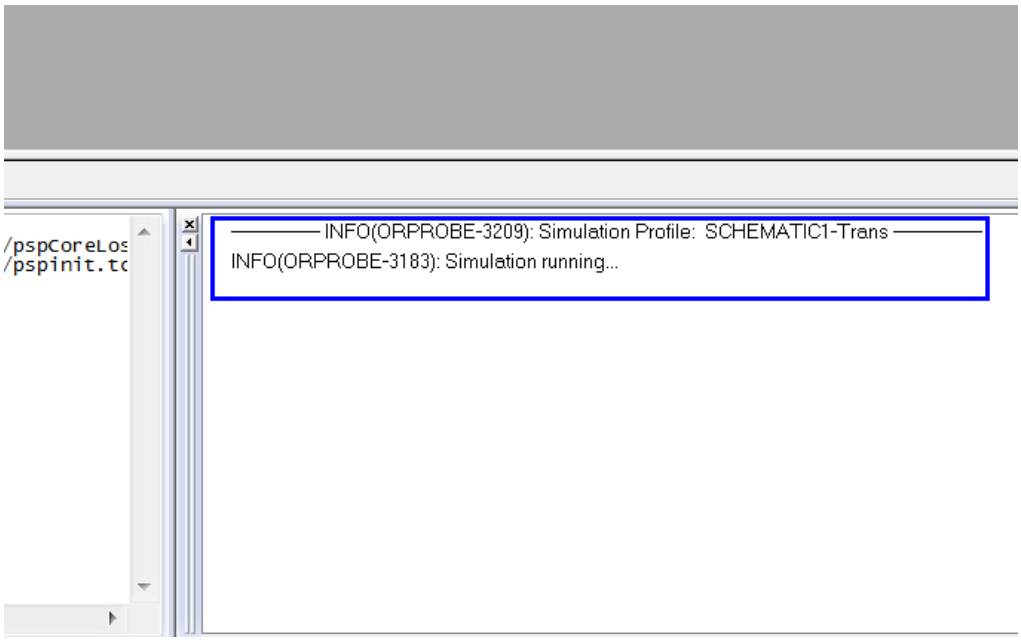
3. Open NoiseFilter_user.cpp and place your breakpoints in the code



4. In Visual Studio, click on Tools→Attached to process:

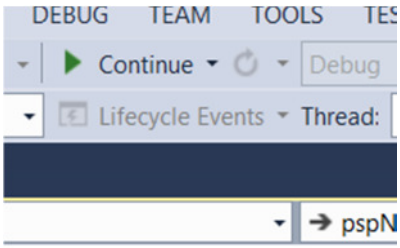


5. Select all the SimSrvr.exe available (clicking on Ctrl+LMB) and click on Attach.
6. Click on Run Simulation in PSpice
7. If everything works fine, PSpice keeps running and the pointer remains in VS:

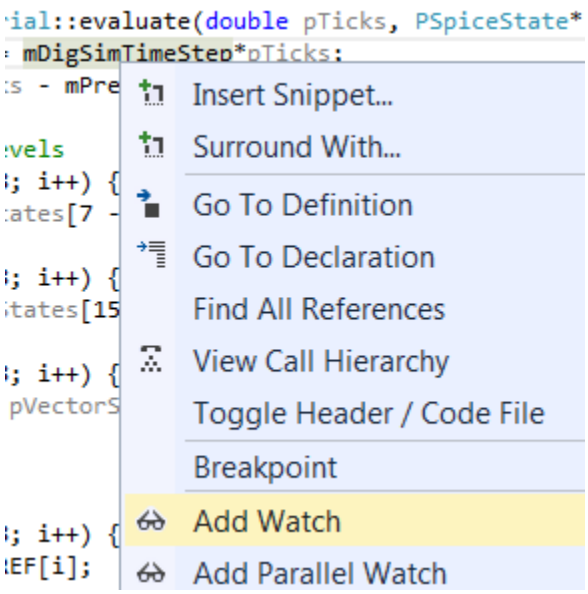


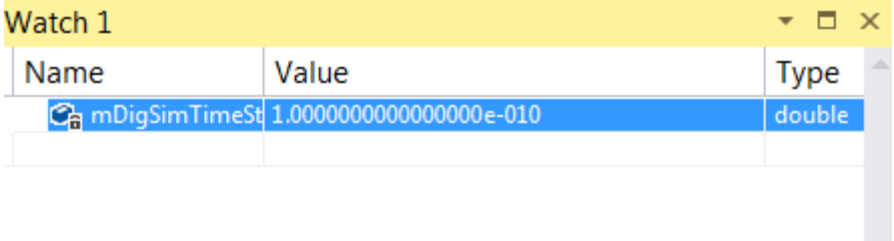
```
    double yVal = 0.0;
    double gain = 0.0;

// User Code
    if (pMode != MDTRAN) {
        for (int i = 0; i < 16 + MSTVCT; i++) {
            sv.x[i] = xVal;
        }
    }
    sv.y[0] = yVal = averaging_filter(xVal, sv.x);
//
```

8. Click on Continue to jump from breakpoint to breakpoint.



9. Use Visual Studio Watch function to see any variable value





3.6 Hardware in the Loop using Arduino

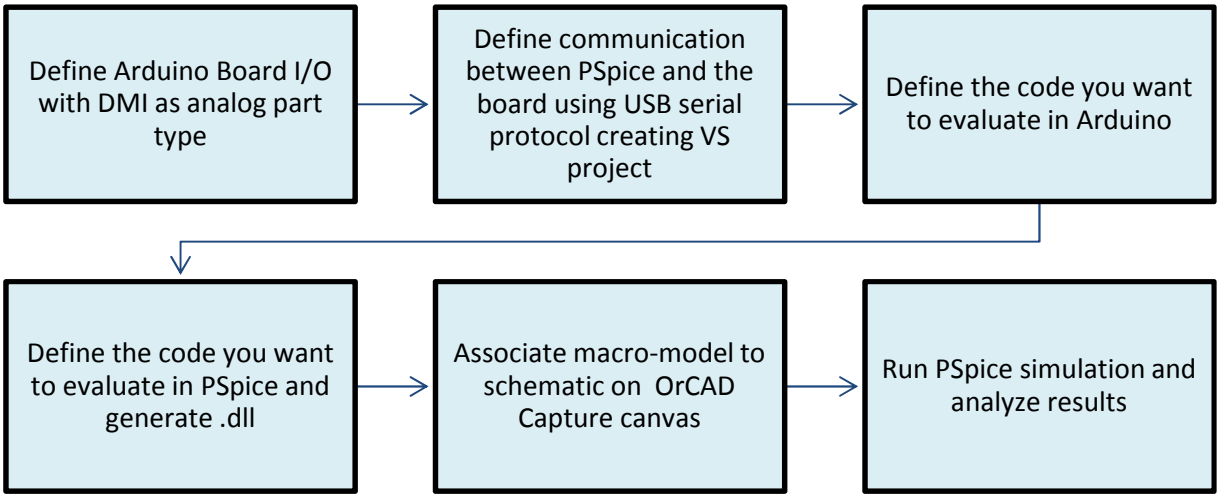
This module shows Hardware in the Loop using an Arduino Board where the communication between the physical board and PSpice is done using Serial USB protocol.

This board has been chosen because it is worldwide used for multiple applications, it is cheap and it allows to demonstrate this new PSpice feature. Of course another boards could be used.

This example demonstrates:

- The advantages of using Virtual Prototyping in PSpice, focusing it to Hardware in the Loop, where data flows from PSpice to the board and vice versa.
- How to define a component using DMI Template Code Generator in the context of Hardware in the Loop.

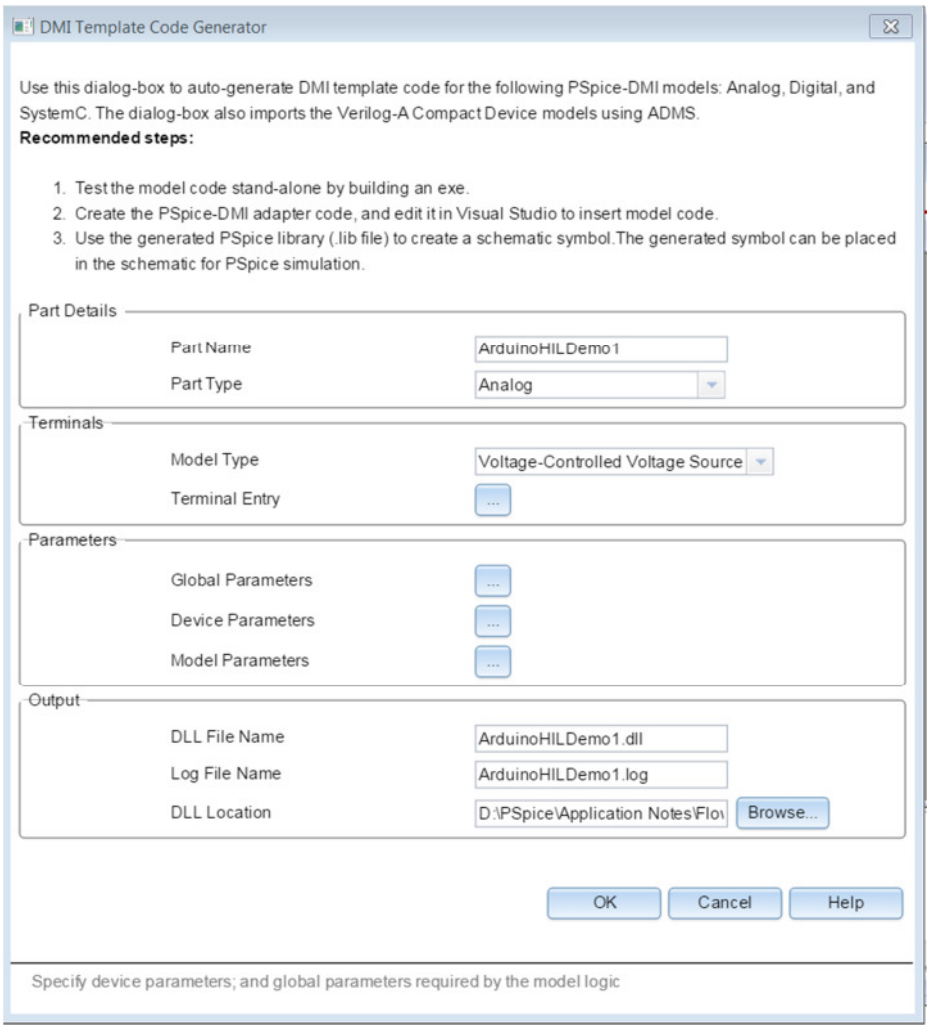
The steps to be followed to design such example are more or less the same than for Digital Power Supply, but some extra steps have to be considered. Let start with the next schema:



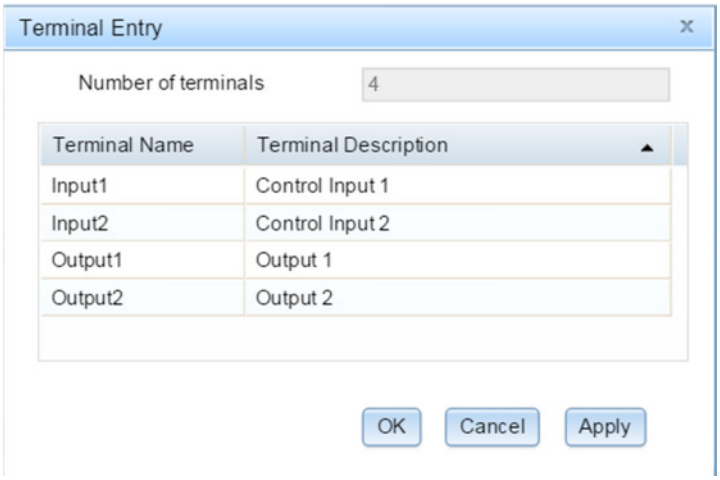
Steps:

1. Launch Model Editor
2. Select Menu Item Model → DMI Template Code Generator

   Enter the data as follows:



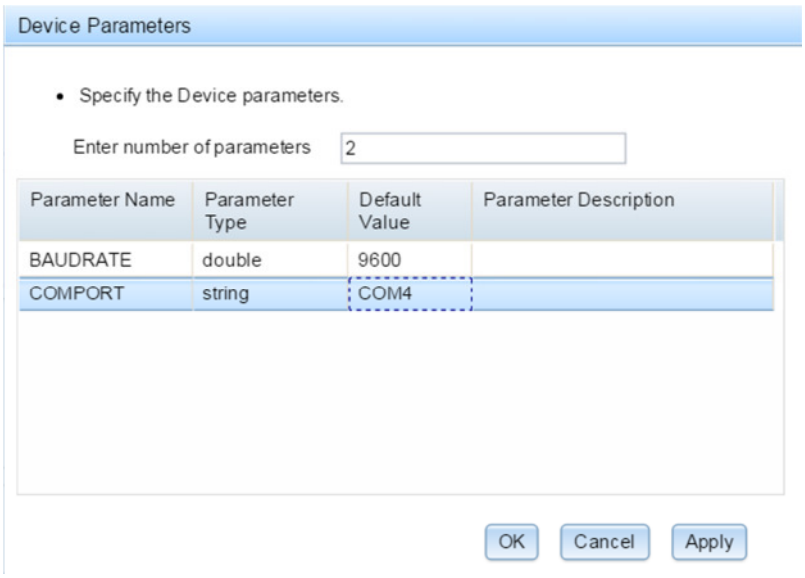
On Terminal Entry, this is what you have to see:



For the DLL Location select please the directory of this example:

<directory>\Circuits\Hardware_in_the_Loop\To_be_completed/DMI_Arduino

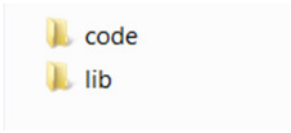
3. Click on Device Parameters and define two new parameters as in the image:



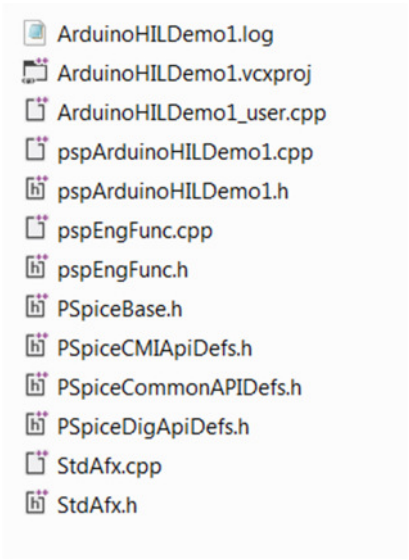
4. Click Apply and OK.

5. Click OK and generate all the files. Automatically a new folder called ArduinoHILDemo1 (Part Name) is included in

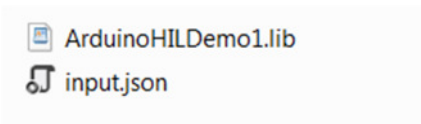
   <directory>\Circuits\Hardware_in_the_Loop\To_be_completed/DMI_Arduino with another two folders: code and lib.



In the code folder you will find the files to be used in Visual Studio to generate the DLL:



In Lib folder you will find the PSpice Model:



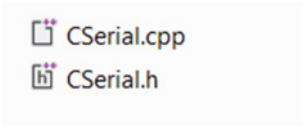
NOTE: The DLL will be generated when all the files are compiled in VS.

6. Include inside of the folder

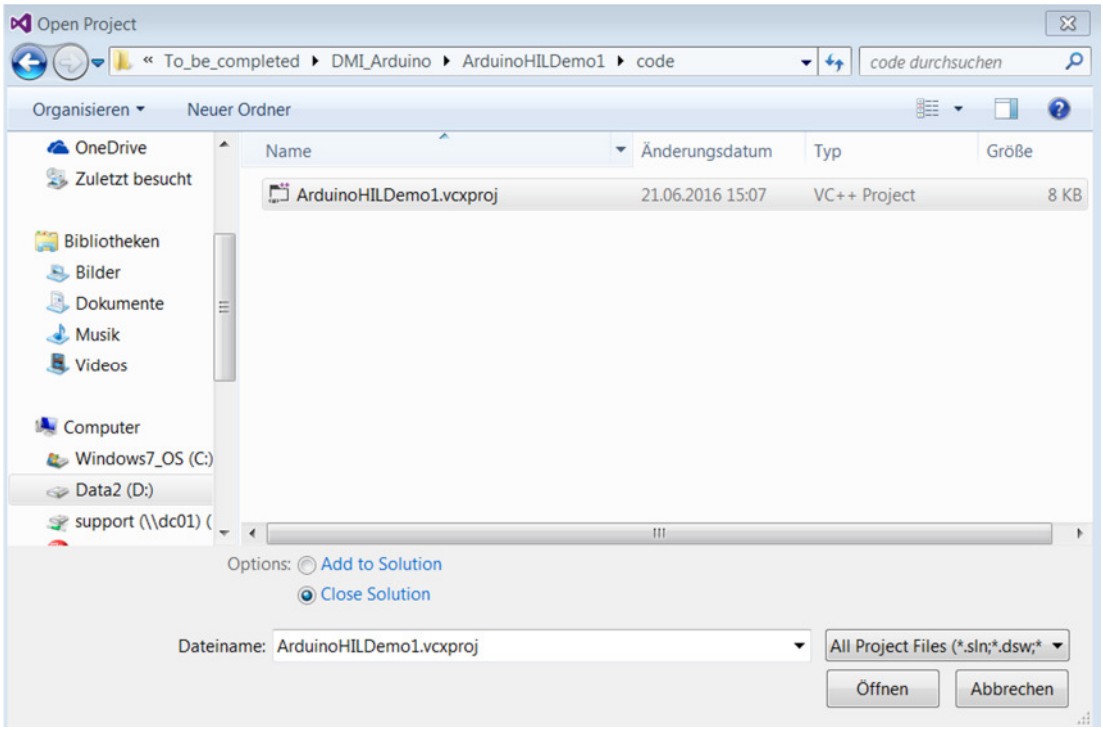
   <directory>Circuits\Hardware_in_the_Loop\To_be_completed\DMI_Arduino\ArduinoHILDemo1\code

   the files, which allow the USB Serial transmission of data. They are located in

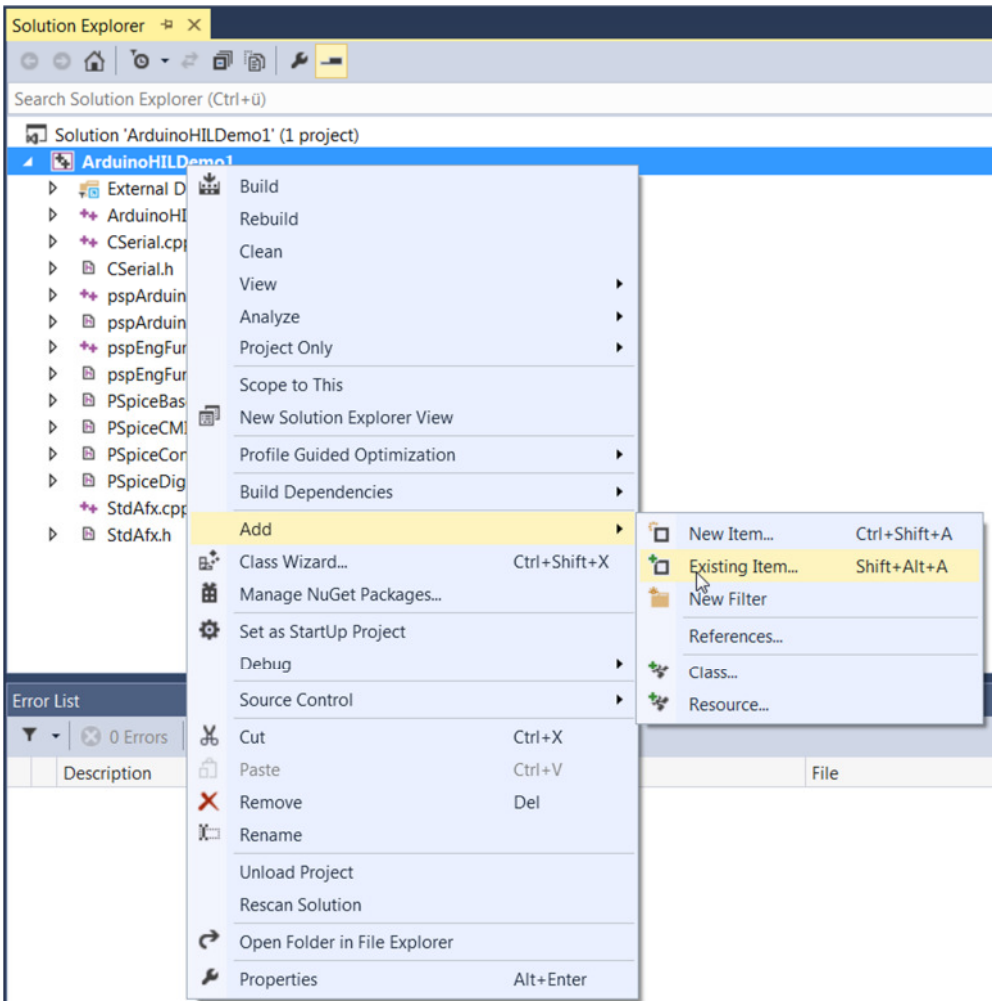
   <directory>Circuits/Hardware_in_the_Loop/To_be_completed/USB_Serial_Protocol



7. Open Visual Studio Community 2013 and click on File→Open→Project/Solution to load the project:



8. On the solution Explorer Tab select ArduinoHILDemo1, click RMB → Add → Existing Item and look for the USB Serial Protocol Files.



9. Select CSerial.cpp and CSerial.h and click Add.

10. Open next files and compare them with the completed ones that are located in <directory>/Completed/DMI_Arduino in order to analyse the code that was added. These files define the working of the model and internal configuration of the code so that the dll works properly using the interface. (Optional).

    - ArduinoHILDemo1_user.cpp
    - pspArduinoHILDemo1.cpp
    - pspArduinoHILDemo1.h

11. Now that you know the differences in terms of code among these files, go to the directory

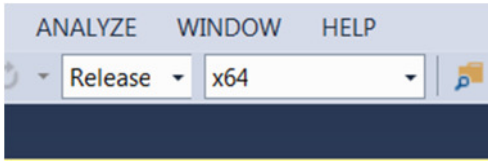
    <directory>\Circuits\Hardware_in_the_Loop\Completed\DMI_Arduino\ArduinoHILDemo1\code

    copy the files that you can see in point 10 and add them in

    <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\DMI_Arduino\ArduinoHILDemo1\code

    replacing those you have created previously.

12. Open Visual Studio again and select Release and x64 from the top of the window:



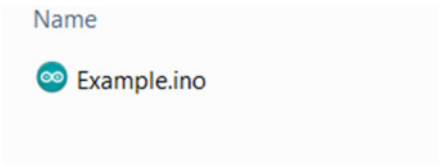
13. Click on Build→Build Solution and generate DLL:

```
1>------ Build started: Project: ArduinoHILDemo1, Configuration: Release x64 ------
1>  ArduinoHILDemo1_user.cpp
1>ArduinoHILDemo1_user.cpp(52): warning C4267: 'argument' : conversion from 'size_t' to 'int', possible loss of data
1>  CSerial.cpp
1>  pspArduinoHILDemo1.cpp
1>  pspEngFunc.cpp
1>  StdAfx.cpp
1>     Creating library D:\PSpice\Application Notes\FlowCAD_AN_Device_Modeling_Interface\Circuits\Hardware_in_the_Loo
1>  Generating code
1>  Finished generating code
1>  ArduinoHILDemo1.vcxproj -> D:\PSpice\Application Notes\FlowCAD_AN_Device_Modeling_Interface\Circuits\Hardware_in_
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

14. Install Arduino IDE in your computer downloading the software from the official Homepage:

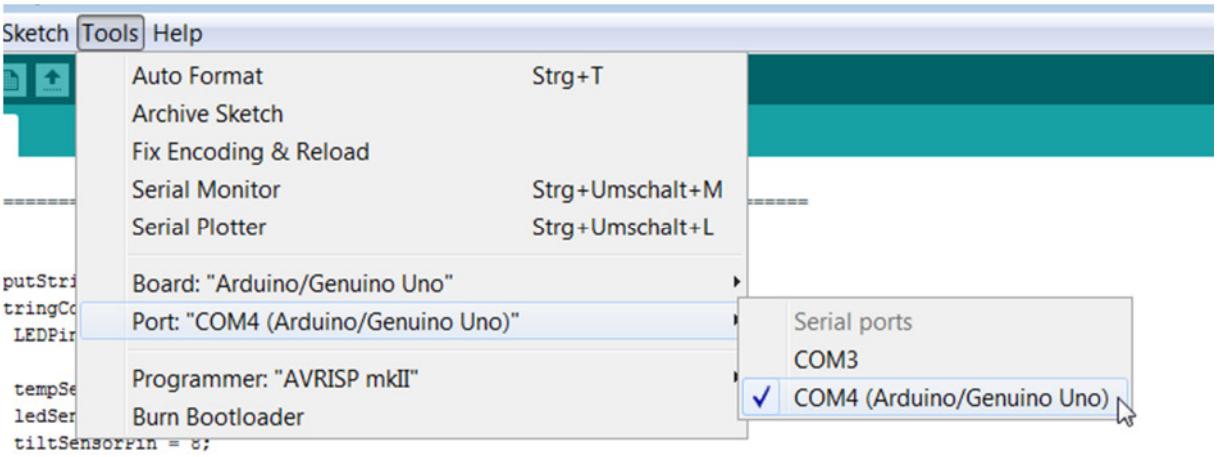
    https://www.arduino.cc/en/Main/Software

15. Open the Arduino Code located in <directory>Circuits\Hardware_in_the_Loop\To_be_completed\Arduino_Code\Example

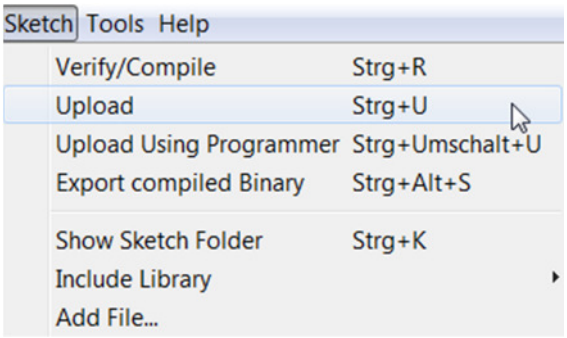


16. Connect the Arduino Board with the USB Cable.

17. Click on Tools and make sure that the Port, where the Arduino Board is connected is selected to COM4.

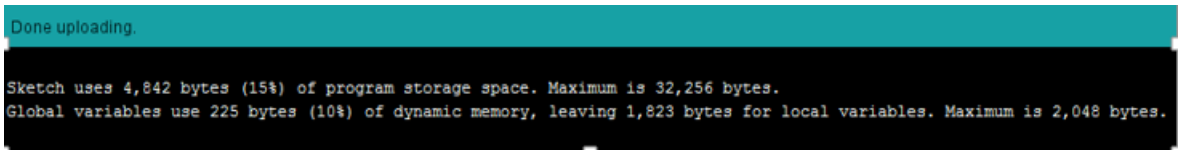


18. Upload the code in the Arduino Board clicking on Sketch → Upload:

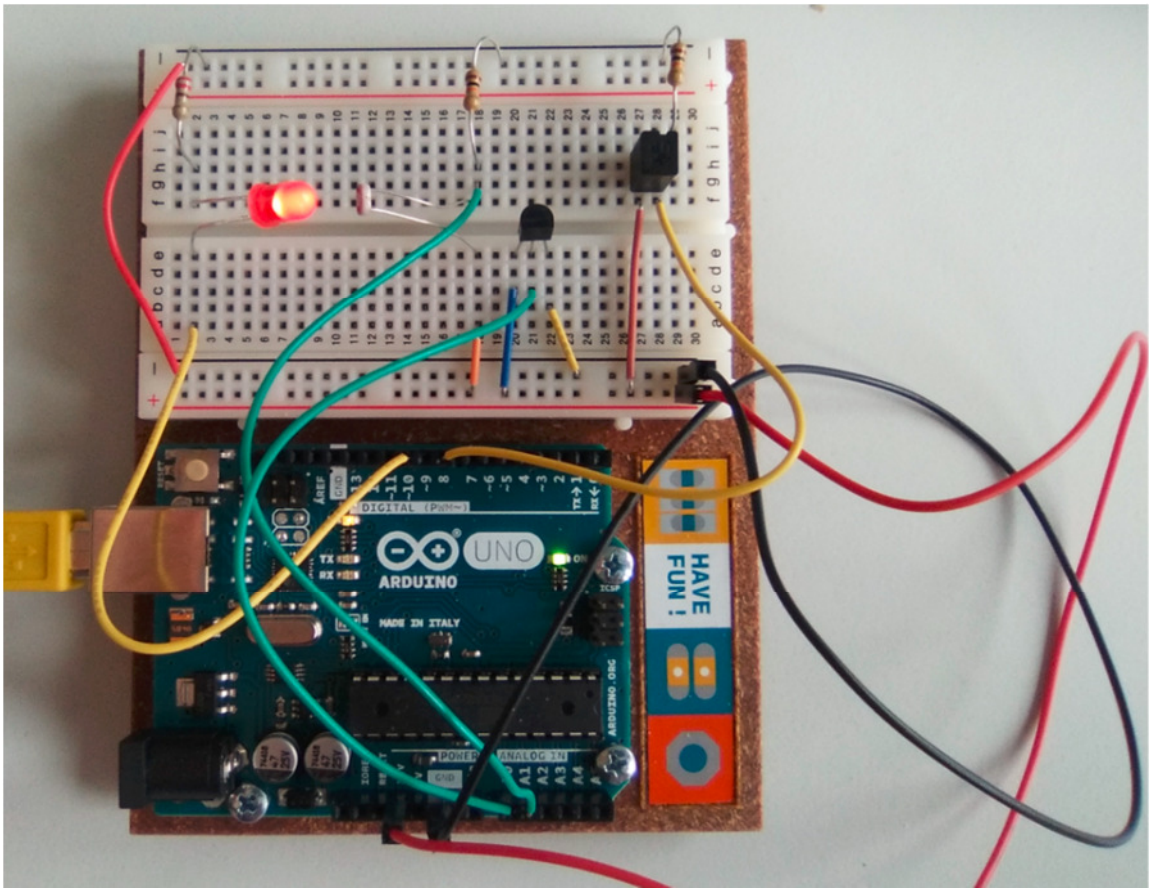


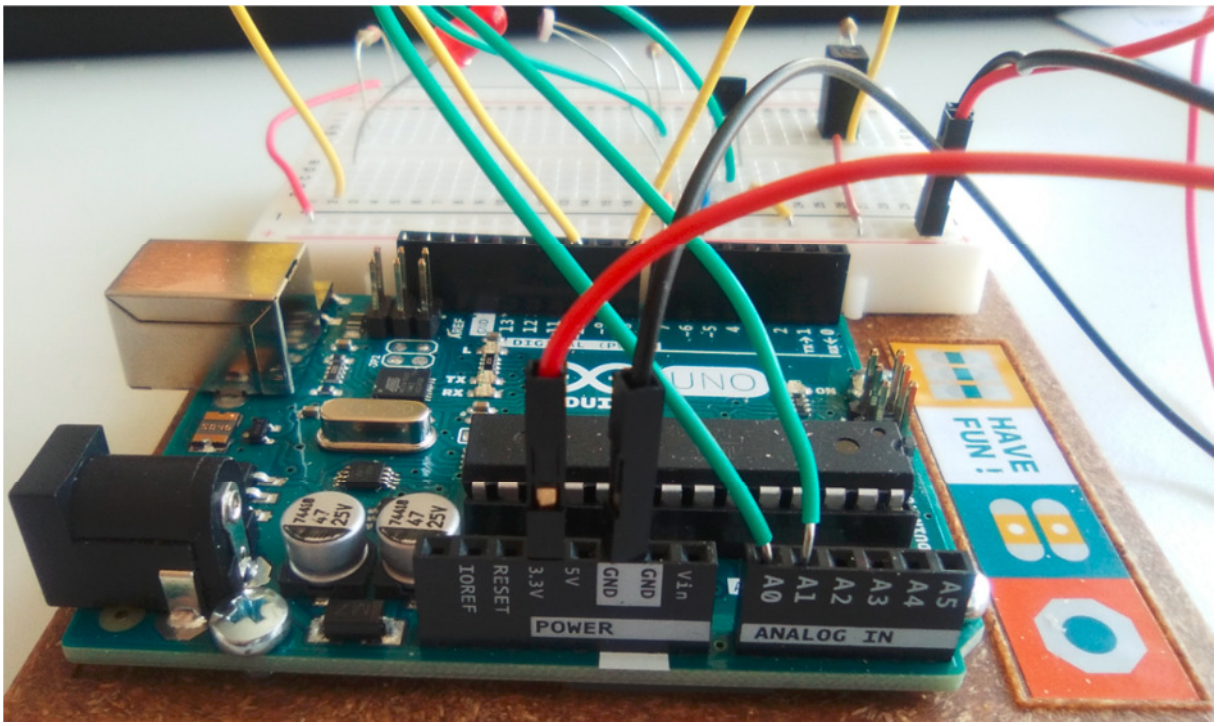
19. If everything works well, you will see the LED, the pin L and the pin ON turned on.

    In Arduino software something like that is shown:

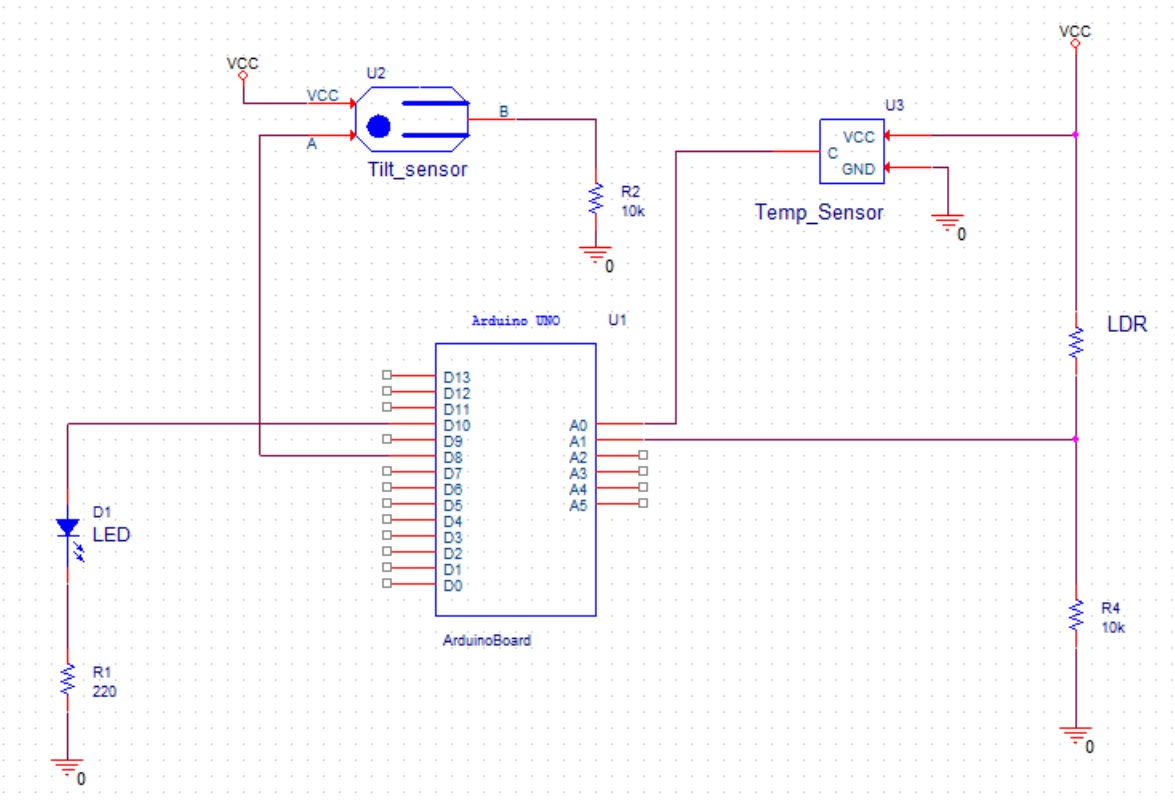


20. Design the Hardware to be simulated:



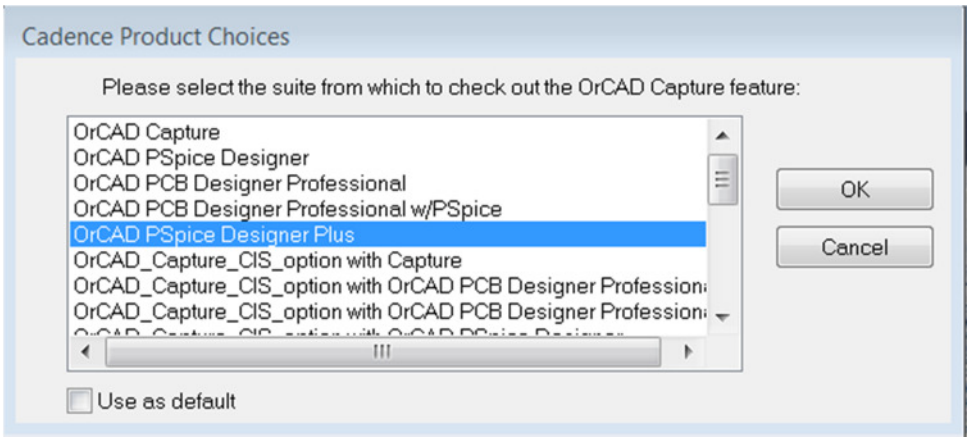


Hardware connection schematic:

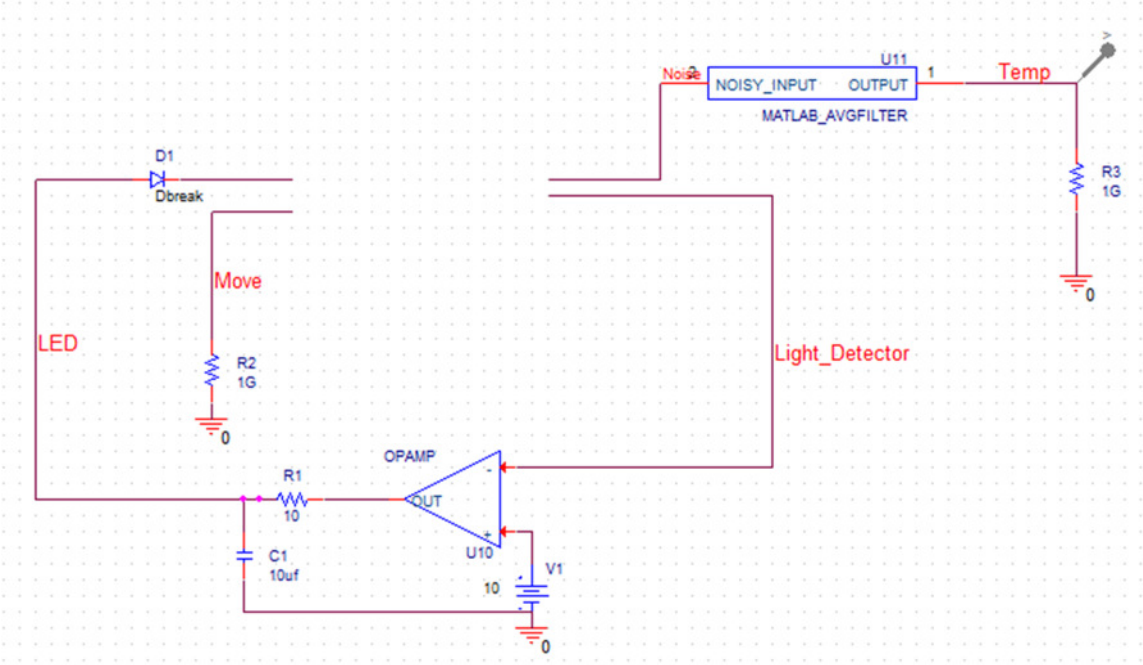


21. Open OrCAD Capture

22. Select Allegro Design Entry CIS or OrCAD PSpice Designer Plus:

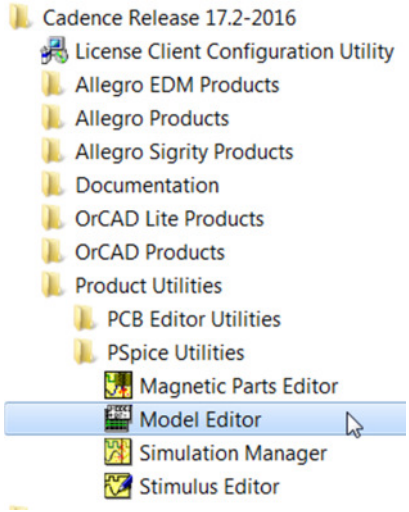


23. Open the project located in <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\Circuit\ArduinoHiL.opj

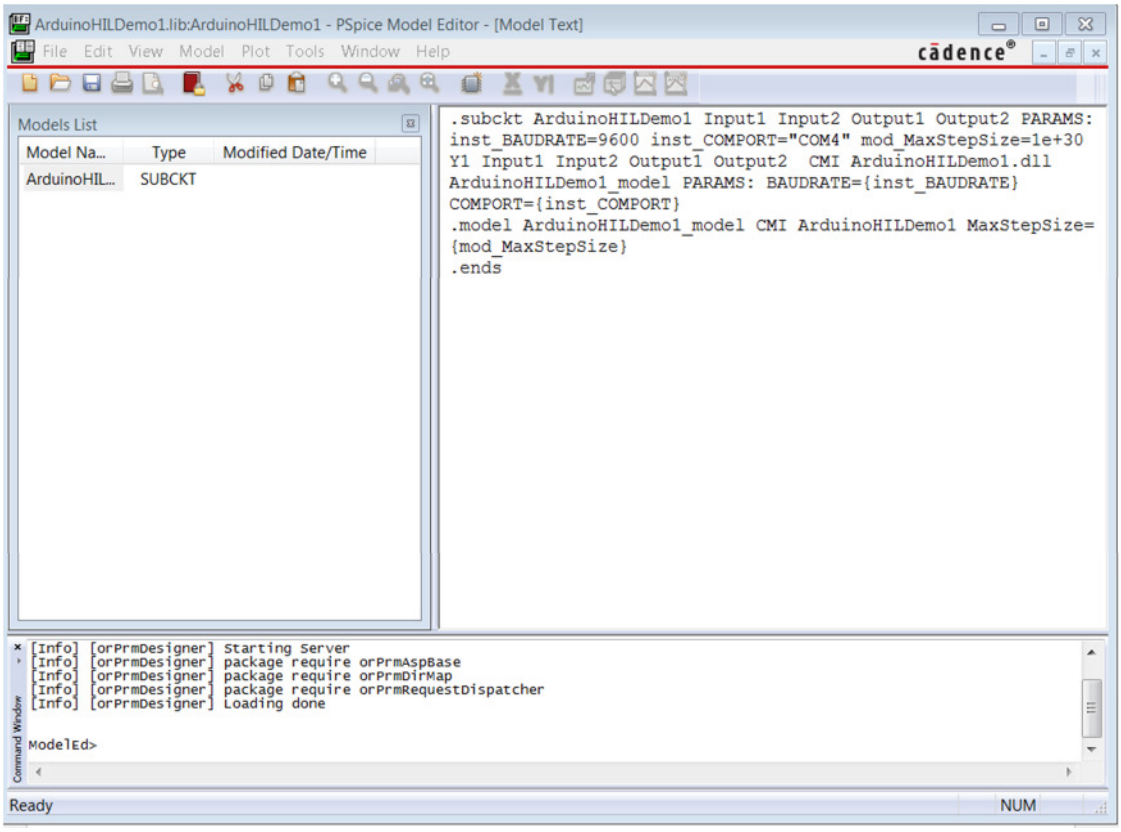


As you can see, everything is defined except the Arduino component.

24. Open Model Editor from PSpice Utilities:



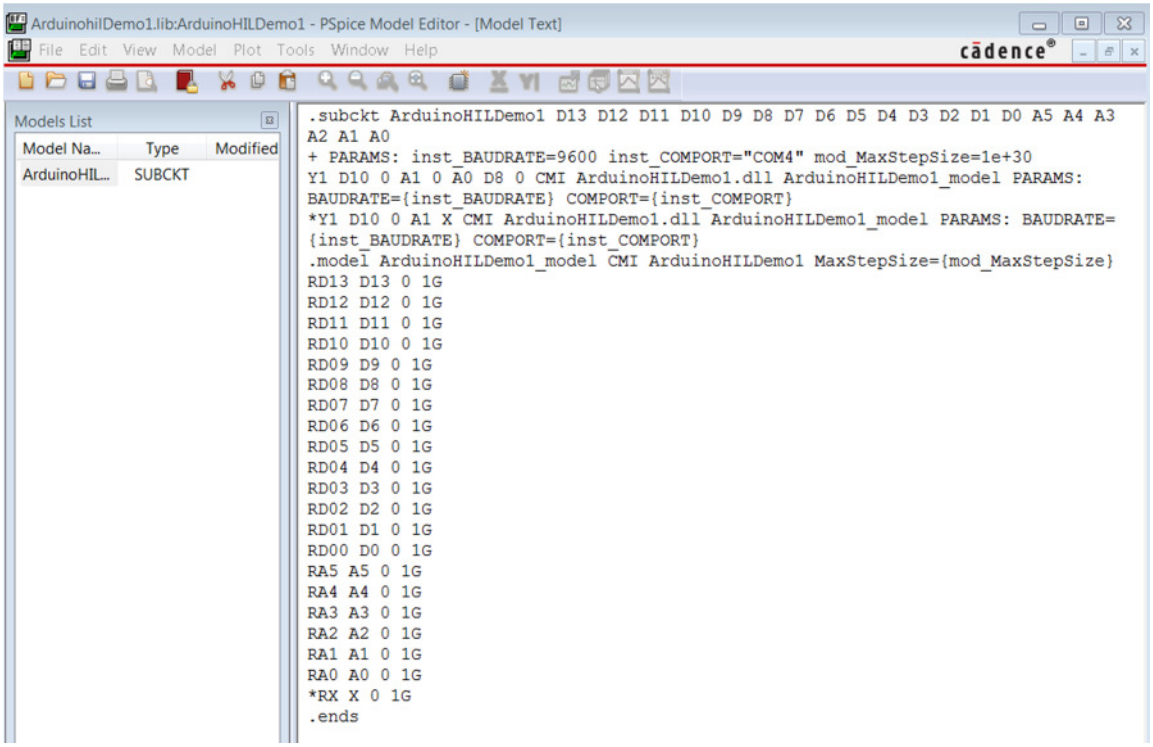
25. Click on File → Open and load the library that was created automatically when you defined the Arduino component using DMI Template Code Generator. It should be placed in <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\DMI_Arduino\ArduinoHILDemo1\Lib



26. Modify the description introducing the information that is available in the file **"Change_Me"** located in

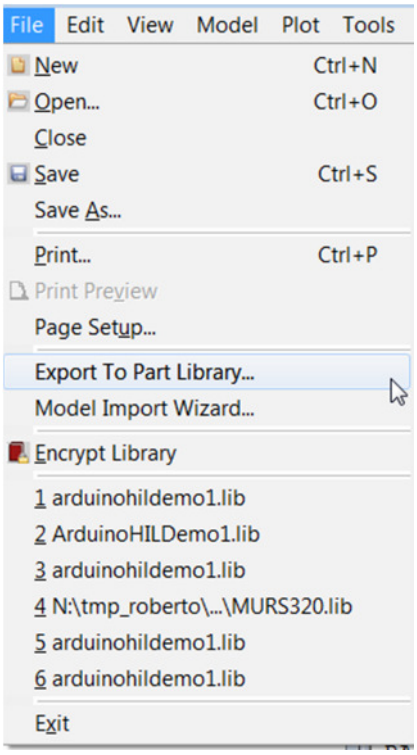
    <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\DMI_Arduino

Copy and paste the whole data in your PSpice Model

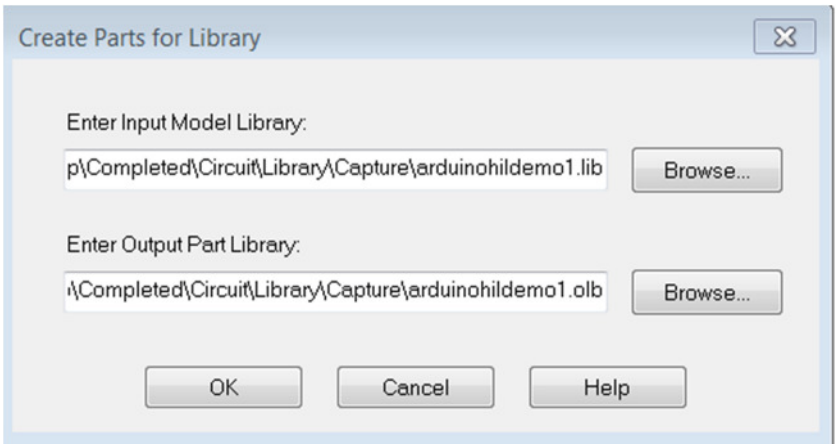


27. Save.

28. With Model Editor opened, click on File → Export to Part Library to generate the symbol (.olb), which is being placed in the schematic:



29. Click on OK

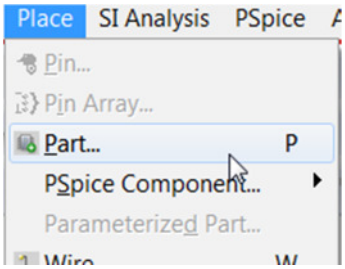


30. Now that your Symbol has been created, open OrCAD Capture and place it. But do **NOT** take the symbol you have just created, but the symbol located in

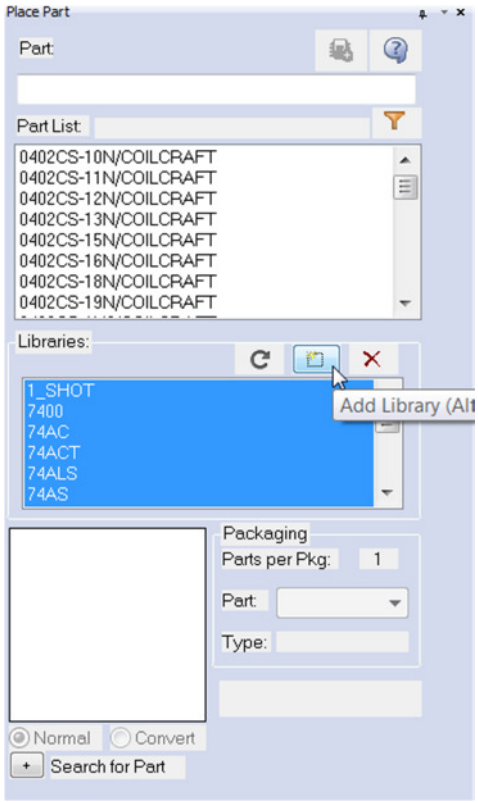
    <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\Circuit\Library\Capture

    as I have reorganized the pins position.

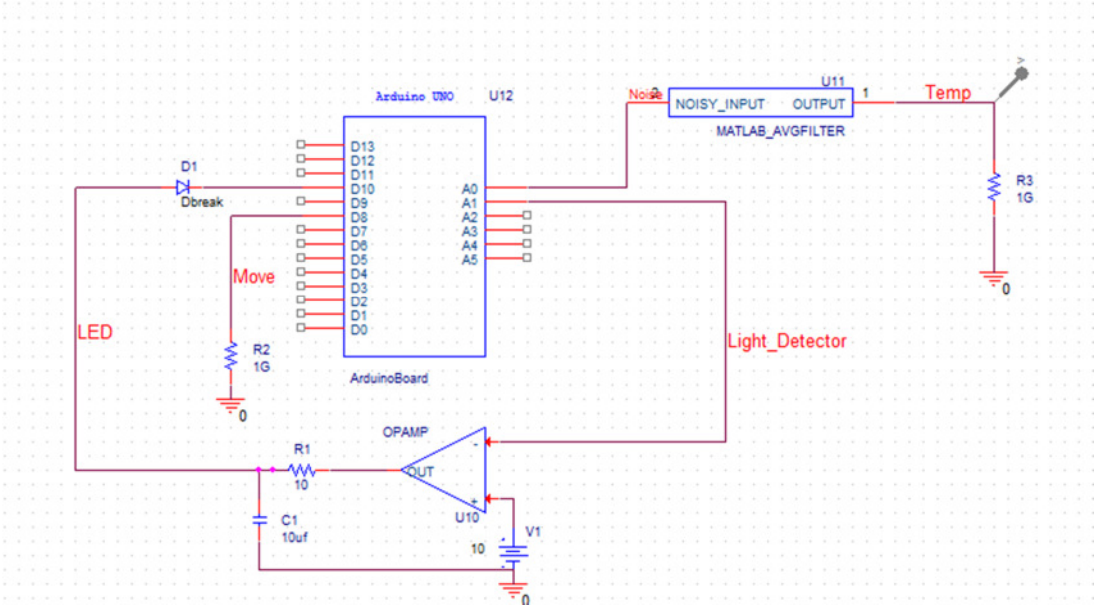
    a. Click on Place → Part



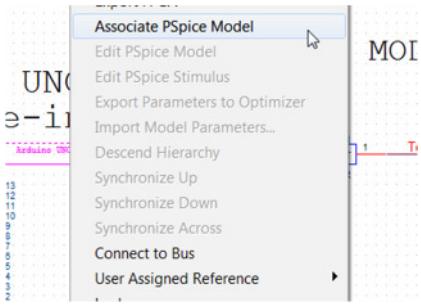
    b. Click on Add Library



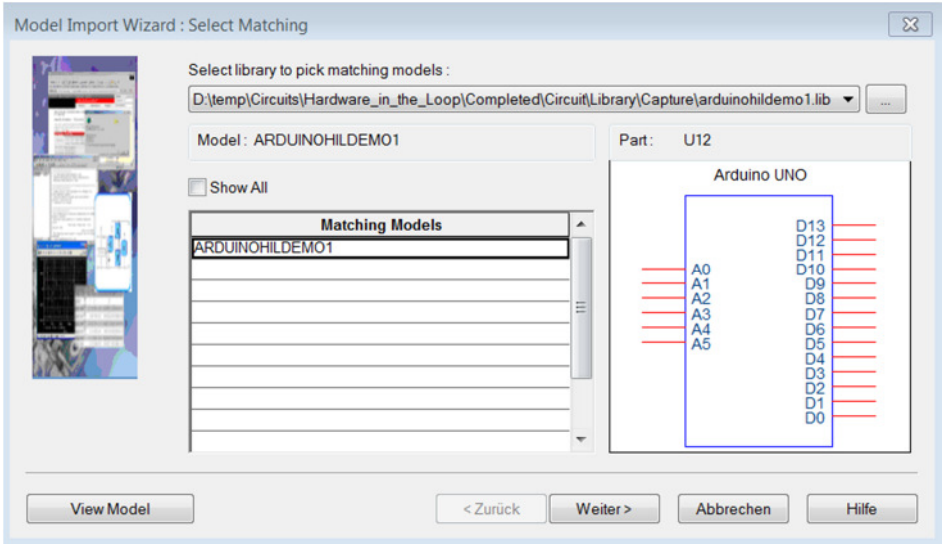
c. Select Arduino.OLB and place it making double click on the component. When it is placed, click on H to mirror horizontally and connect it with the pins:



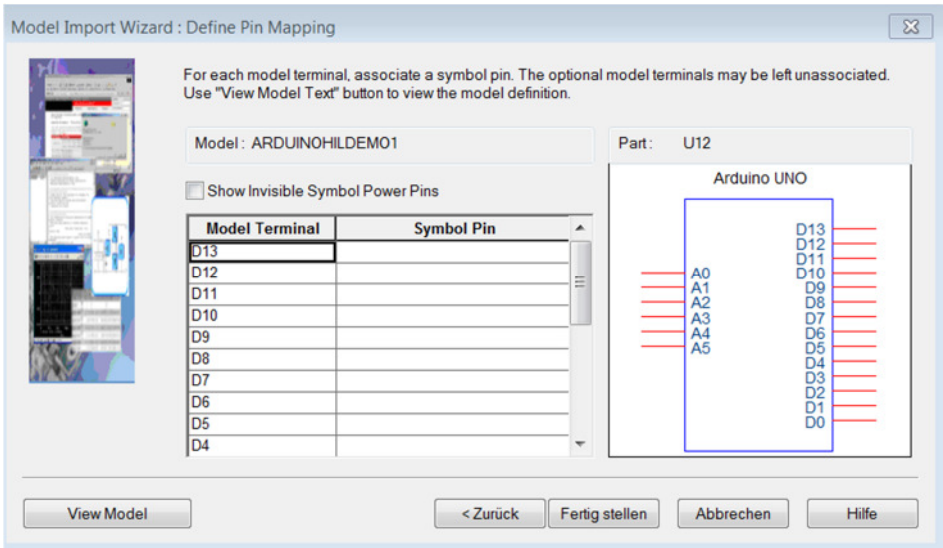
d. As you have placed an Arduino Symbol without an associated PSpice Model, select the Symbol you have just placed and click RMB:



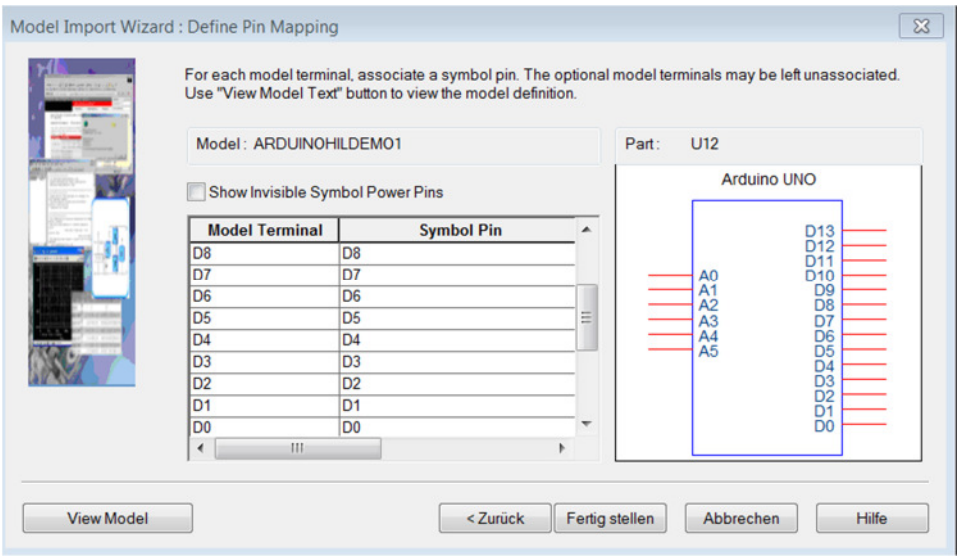
e. Click Yes and a windows pops up:



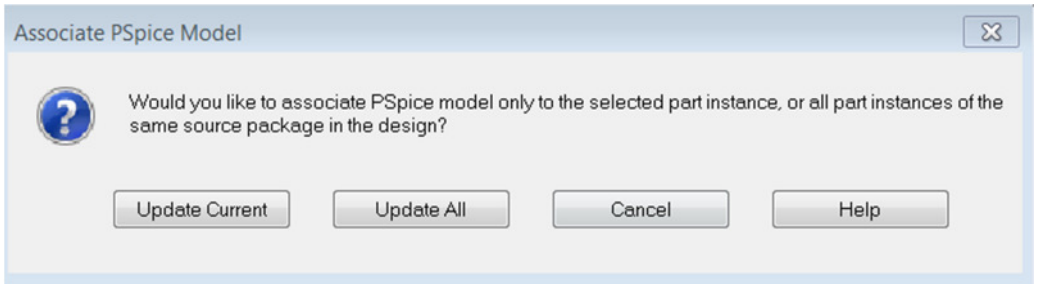
f. Click Next:



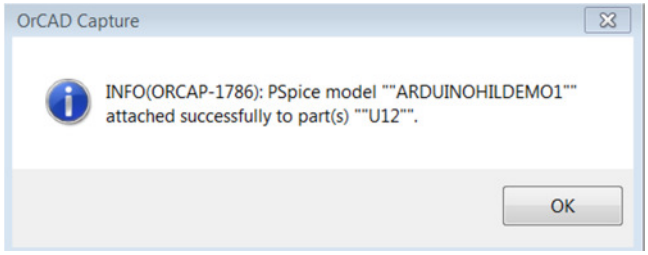
g. Associate each Model Terminal yith the corresponding Symbol PIN:



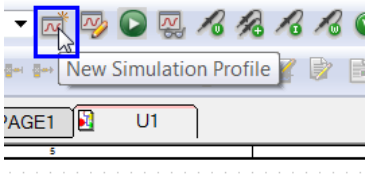
h. Click on Fertig and next Windows pops up:

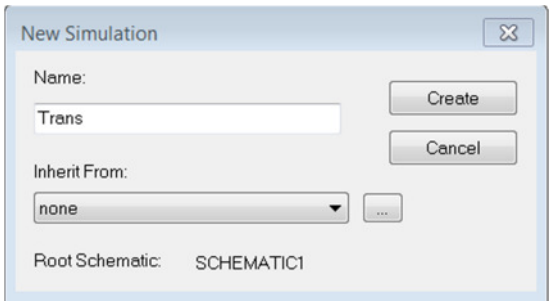


i. Click on Update All and OK

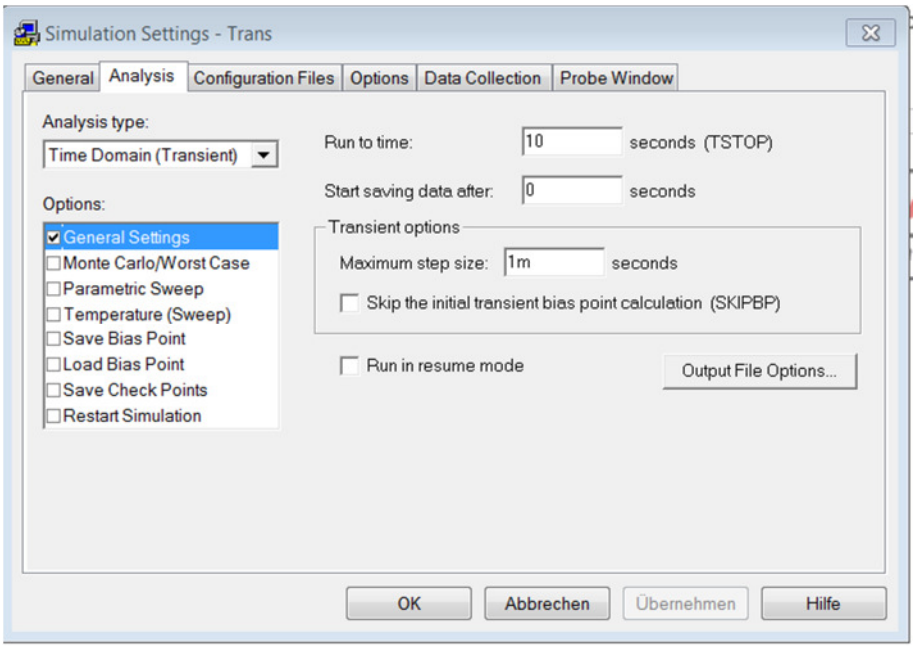


31. Now you are ready to define the Simulation Profile. Click on New Simulation with the name Trans and click on Create:

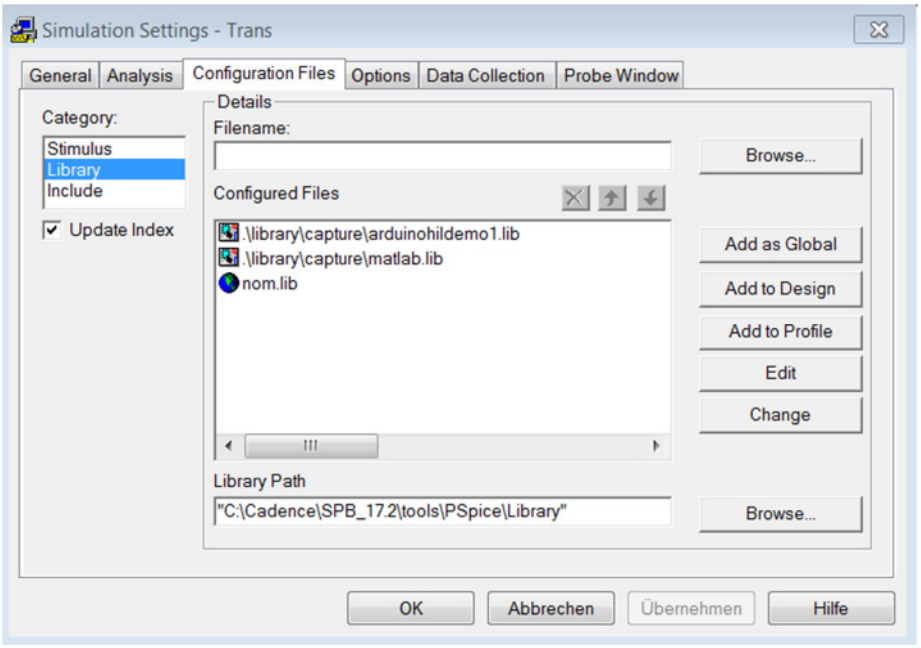




32. Define a Time Domain Simulation and fill in the options with the next values:



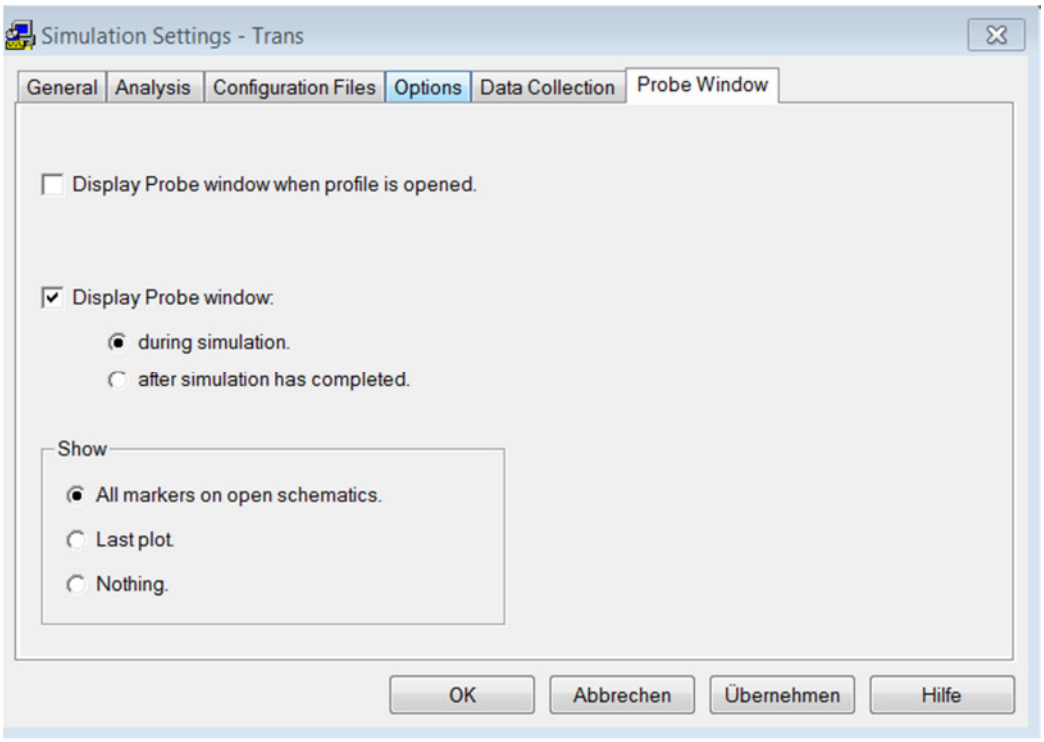
33. Click on Configuration Files, select Library and add the PSpice Model for the filter and the Arduino Board:



Your library for the Arduino Board is in <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\DMI_Arduino\ArduinoHILDemo1\lib

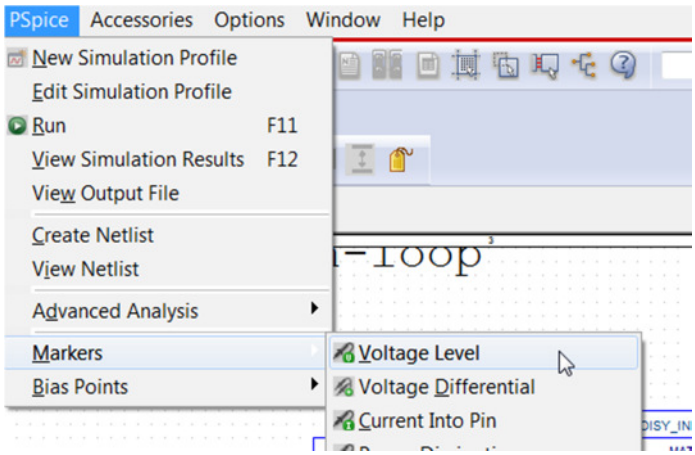
The library for the Matlab Filter is in <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\Circuit\Library\Capture

34. Click on Probe Window and select the next configuration:

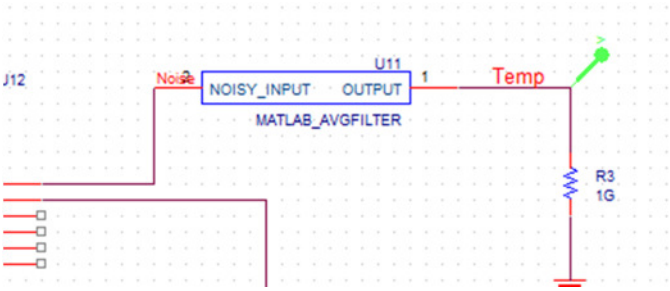


35. Click on OK.

36. In the Schematic click on PSpice → Markers → Voltage Level



and place this marker as in the image:



37. Before you can simulate Hardware in the Loop, you have to place the DLL's in the PSpice Simulation Settings Folder you have just created. Copy your DLL located in

    <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\DMI_Arduino\ArduinoHILDemo1\code\x64\Release\ArduinoHILDemo1.dll

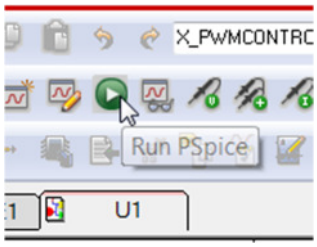
    and past it in

    <directory>Circuits\Hardware_in_the_Loop\To_be_completed\Circuit\ArduinoHiL-PSpiceFiles\SCHEMATIC1\Trans

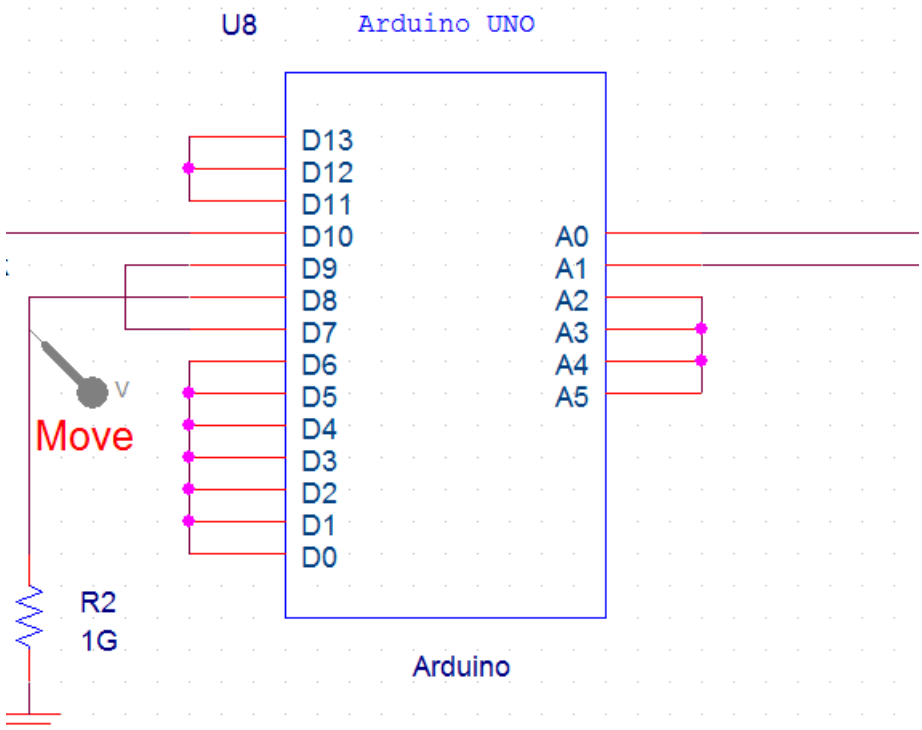
38. Do the same with the DLL for the Matlab Filter located in

    <directory>\Circuits\Hardware_in_the_Loop\To_be_completed\Circuit\Library\DLL

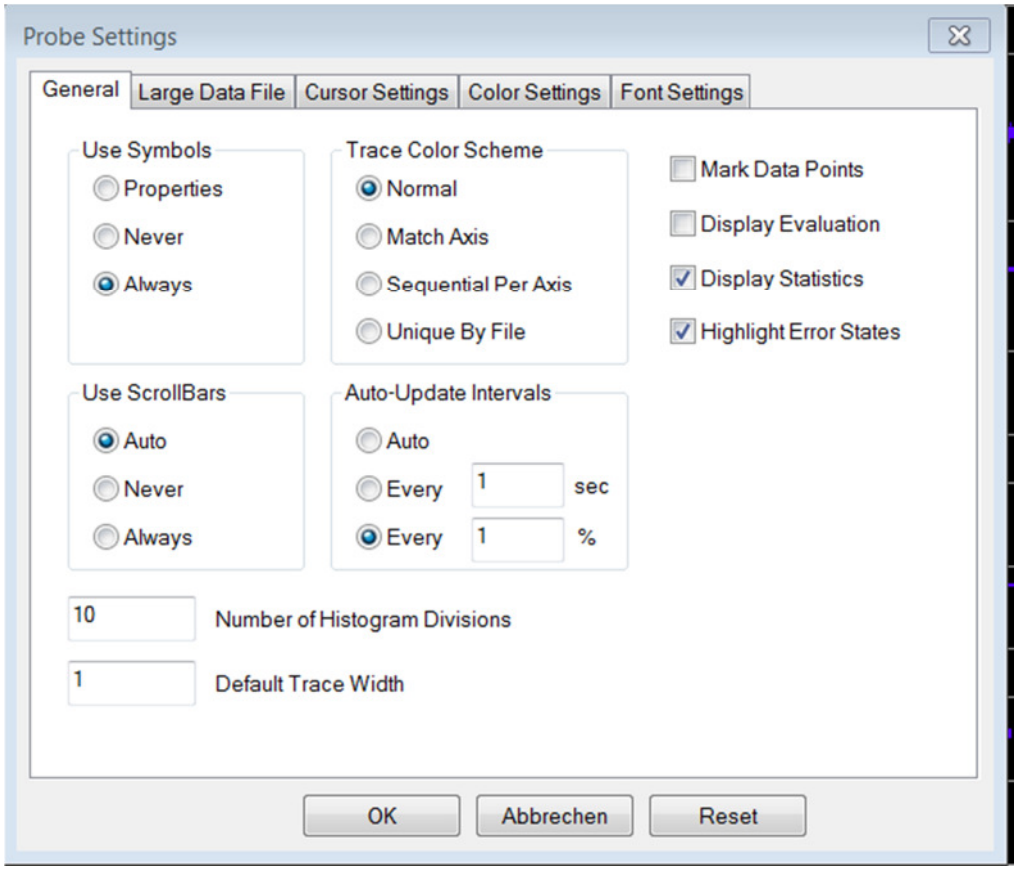
39. Now connect the hardware to the computer (if it is not already connected) and click on Run Simulation:



    NOTE: If you get a Netlisting Error, you have to connect all the unconnected pins together.



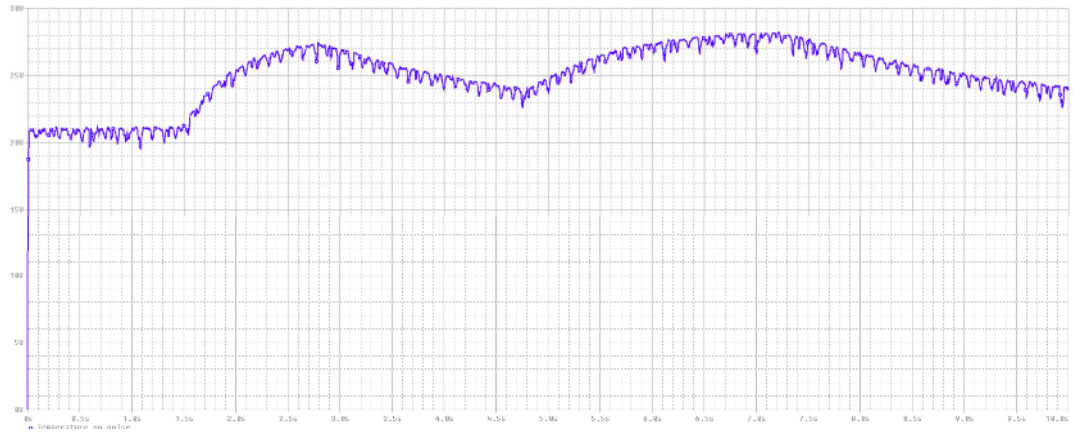
40. When the Probe Window opens, click on Tools → Options and select Auto-Update Intervals as in the image.



41. Test the simulation HiL moving the board, incrementing the sensor temperature or varying the amount of received light:

NOTE: Move the voltage Marker you placed to the different nodes to visualize the different results.

- Temperature Sensor:



- Light Sensor:

- Tilt Sensor: