PSpice Device and System Modeling in C/ C++ and SystemC

Product Version 17.2-2016
April 2016

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Product PSpice contains technology licensed from, and copyrighted by: Apache Software Foundation, and is © 2000-2005, Apache Software Foundation. All rights reserved. Sun Microsystems and © 1994-2007, Sun Microsystems, Inc. Free Software Foundation and © 1989, 1991, Free Software Foundation, Inc. Regents of the University of California, Sun Microsystems, Inc., Scriptics Corporation and © 2001, Regents of the University of California. Daniel Stenberg and © 1996 - 2006, Daniel Stenberg. UMFPACK and © 2005, Timothy A. Davis, University of Florida, (davis@cise.ulf.edu). Ken Martin, Will Schroeder, Bill Lorensen and © 1993-2002, Ken Martin, Will Schroeder, Bill Lorensen. Massachusetts Institute of Technology and © 2003, the Board of Trustees of Massachusetts Institute of Technology. ADMS[GNU Lesser General Public License], and © 2015, ADMS. All rights reserved.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

1

Introduction to PSpice System Design

The PSpice System Design with C and SystemC tutorial provides detailed instructions to build and compile various PSpice Device Modeling Interface (DMI) compatible C and SystemC models using Microsoft® Visual Studio Community 2013. It also provides step-by-step instructions to simulate these models in PSpice A/D.

This tutorial covers the following topics:

- Setting up the Environment for PSpice DMI Models
- Generating and Simulating a PSpice DMI model for Digital Power Supply Simulation
- Generating and Simulating a PSpice DMI Model for Analog Behavioral Circuit
- Generating and Simulating a Verilog-A file based PSpice DMI Model
- Generating and Simulating a PSpice DMI Model for State Model Simulation
- Generating and Simulating a SystemC based PSpice DMI Model

Audience

This tutorial is designed for first-time users of PSpice DMI models in PSpice simulation. If you want to use PSpice DMI models in PSpice simulation, compile and build the models using Microsoft Visual Studio Community 2013.

Prerequisities

Before you start to run the tutorial, ensure that the following software are installed on your system:

- Microsoft Visual Studio Community 2013
- Cadence® OrCAD® Capture 17.2-2016 or onwards
- Cadence® PSpice® A/D 17.2-2016 or onwards

- Mathworks® Matlab 2015b or onwards(64-bit)

It is assumed that you are familiar with Microsoft Visual Studio Community 2013, Cadence OrCAD Capture, Cadence PSpice A/D, and Mathworks Matlab. The scope of this document does not include explaining the interfaces, commands, or various methodologies of these software. This document contains detailed instructions around building and compiling PSpice DMI models.

Note: For more information on OrCAD Capture, PSpice A/D, and C APIs, refer to *OrCAD Capture User Guide*, *PSpice Reference Guide*, *PSpice Device Modeling Interface API Reference*, and *PSpice User Guide*.

2

Setting up the Environment for PSpice DMI Models

This chapter explains the setup procedure for C and SystemC models in PSpice.

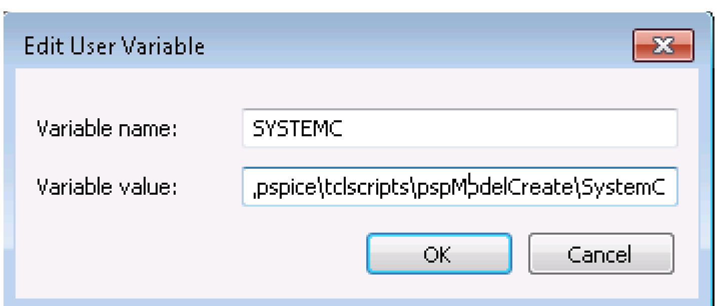
Do the following steps to create the environment variables that you need to get started to create a Visual Studio Project:

1. Unzip the `PSpiceSystems.zip` file in your system.

   Once unzipped, you can see the following sub-folders inside the `PSpiceSystems` folder: `DigitalPowerSupply`, `NoiseFilter`, `StateModel`, `VerilogA`, and `SystemC`.

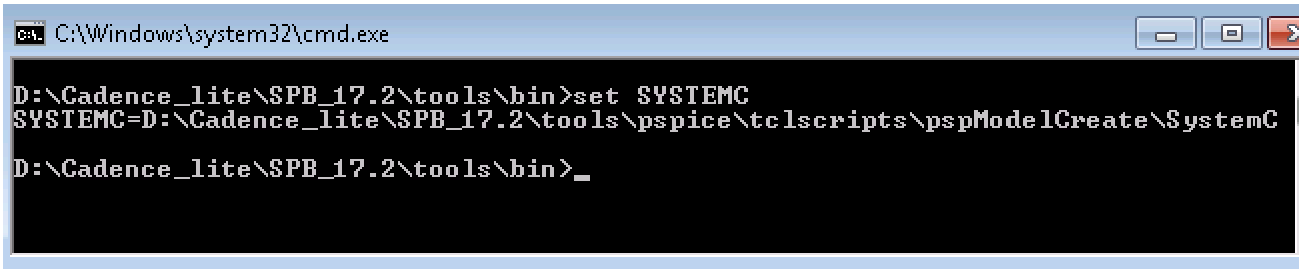
2. Create a new environment variable, *SYSTEMC*, and set the SystemC installation path as its value.

   By default, the systemC is installed with the Cadence installation at `<installation path>\tools\pspice\tclscripts\pspModelCreate\SystemC`.



3. Open the Windows command prompt and verify the SystemC path using the `set` command.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.



©1991-2015 Cadence Design Systems, Inc. All rights reserved.

3

Generating and Simulating a PSpice DMI model for Digital Power Supply Simulation

This module covers an example of a Digital Power Supply with models using multiple level of abstractions.

In this module, you will:

- Generate a template code for PSpice DMI model
- Use the PSpice DMI model for the Digital PWM Control block
- Simulate the PSpice DMI model with respect to Digital Power Supply circuit

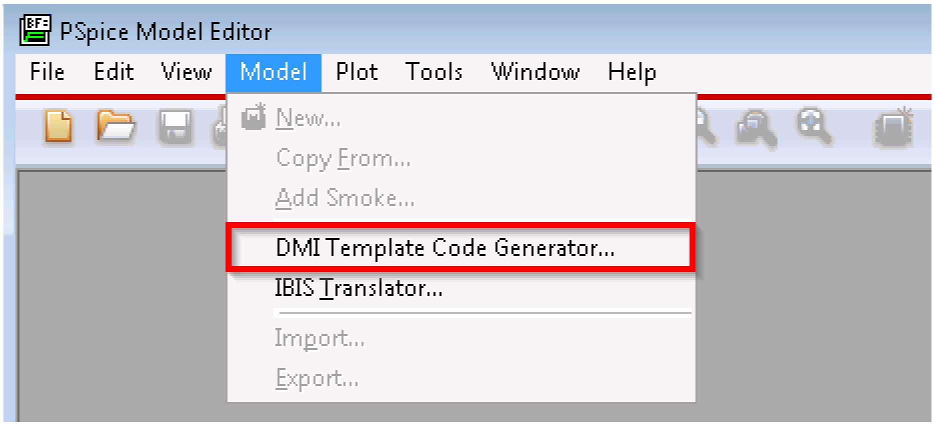
Do the following steps to generate a template code for a PSpice DMI model:

1. Select *Start Menu – All Programs – Cadence 17.2-2016 – Product Utilities – PSpice Utilities – Model Editor* to launch Model Editor.

   If prompted, choose a license that includes PSpice A/D; for example, *OrCAD PSpice Designer Plus*.

2. Select *Model – DMI Template Code Generator* in Model Editor.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.



3. Enter the following data in the DMI Template Code Generator window to generate a Digital C/C++ based PSpice DMI model:

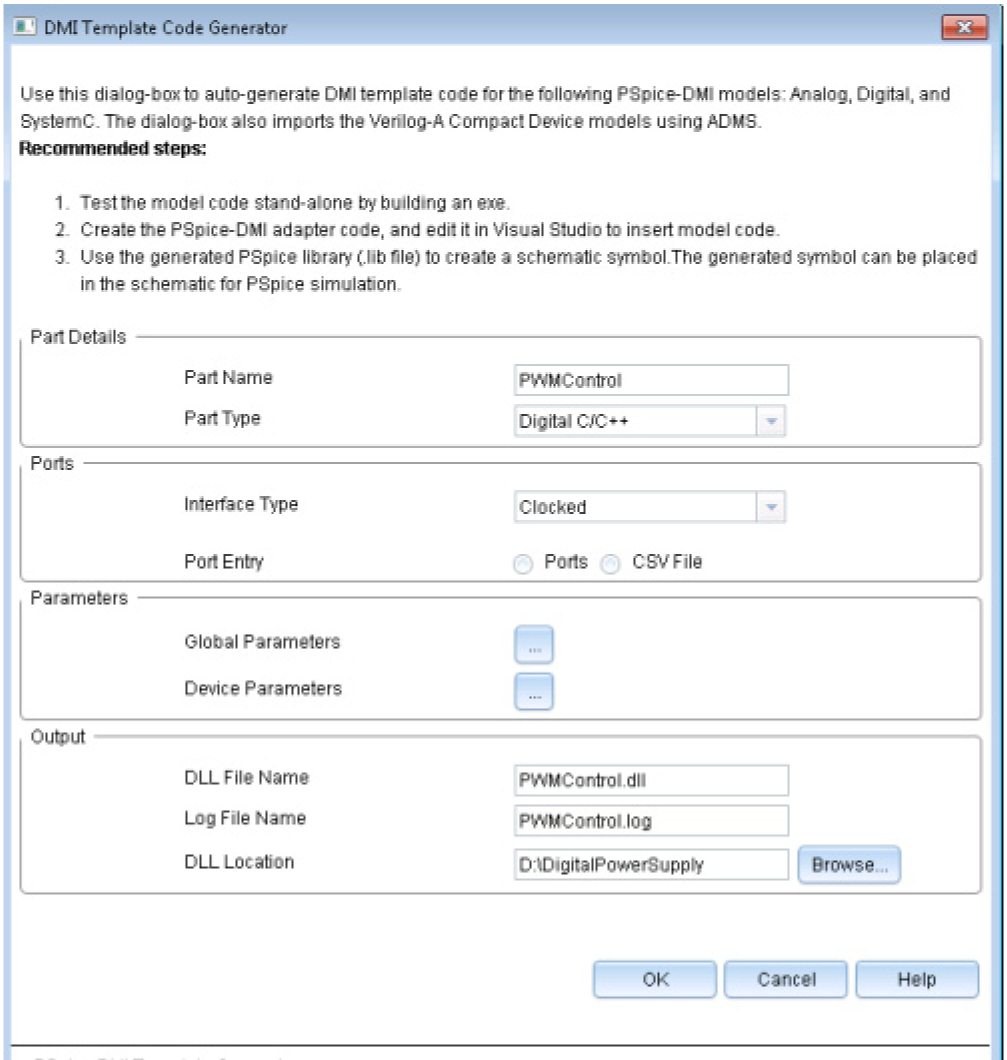
Part Name: `PWMControl`

Part Type: `Digital C/C++`

Interface Type: `Clocked`

DLL Location: `DigitalPowerSupply` folder

©1991-2015 Cadence Design Systems, Inc. All rights reserved.



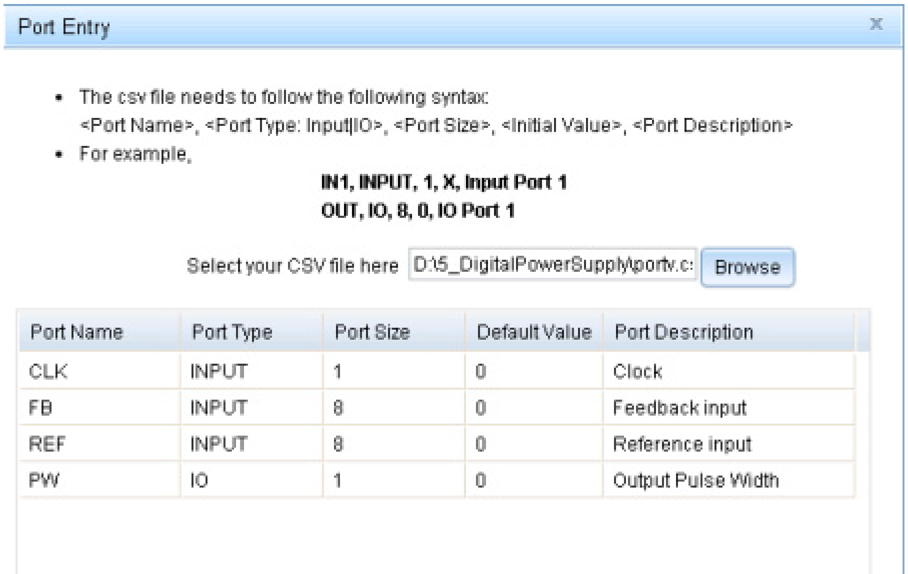
4. Select the CSV File checkbox in the Port Entry field.

A Port Entry window is displayed.

5. Browse the `portsv.csv` file from `DigitalPowerSupply` folder.

The ports are automatically read from the CSV file.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.



6. Review the port entry list in the Port Entry window and click `OK`.
7. Click the Global Parameters click box.

   A Global Parameters window is displayed.
8. Enter the following details in the Global Parameters window for `PER` and `D` parameters and click `OK`:

   Enter number of parameters: 2

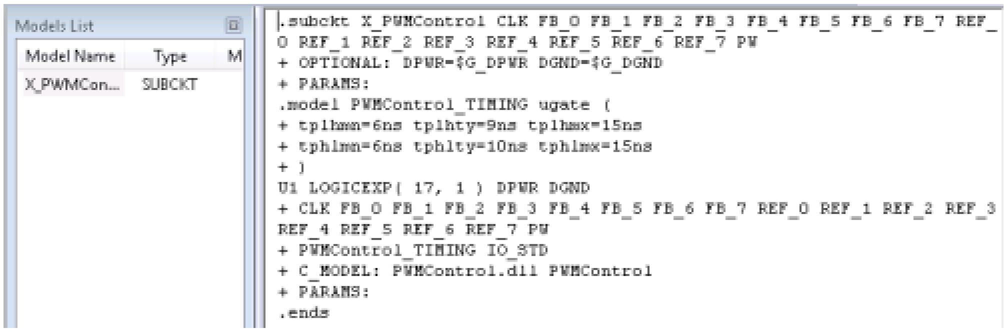
`Parameter Name`	Parameter Type	Default Value	Parameter Description
`PER`	double	0	Period
`D`	double	0	Duty Cycle

9. Click `OK` on the DMI Template Code Generator window.

   A log file is displayed. A .lib file is successfully created at the specified DLL location and opened in Model Editor.
10. Click on the library name in the Model List window of the Model Editor to see the library infromation.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

In the following screenshot, you can see that the library points to the DLL that is created for the model. You will complete the template model code that was generated on creation of the .dll and .lib file and regenerate the .dll file.



11. Launch Visual Studio Community 2013 in your machine.

12. Click *Open Project* in the Visual Studio's Start Page and browse to the `DLL location` for the Visual Studio Project.

In this case, the Visual Studio project is `PWMControl.vcxproj`.

13. Modify the default configuration in Configuration Manager (*Build – Configuration Manager*) to 64-bit platform using the following steps:

a. In the `Active Solution Platform` drop-down list, select the `<New...>` option to open the `New Solution Platform` window.

b. In the `Type or select the new platform` drop-down list, select 64-bit platform and close the window.

14. Build the project using *Build – Build Solution* in the Visual Studio to verify if there are no build issues.

15. Expand PWMControl project in Solution Explorer and open the `PWMControl_user.cpp` file to edit using the following steps:

a. Search the following text in the .cpp file: `psppspPWMControl::evaluate(`

b. Once you find `psppspPWMControl::evaluate(`, search for `// LOGIC TO BE IMPLEMENTED BY USER.`

You will add the model logic code here.

c. Add the following code after `PW=pVectorStates[17].getLevel();` inside the if loop:

```
pspBits2Int(FB, FBInt, 8);
pspBits2Int(REF, REFInt, 8);
```


©1991-2015 Cadence Design Systems, Inc. All rights reserved.

```
if (REFInt > FBInt && mD < 0.98) {
    mD += 0.001;
}
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;
}
```

d. Modify `fp_SetState(mRef, j, &lState, NULL);` to `fp_SetState(mRef, j-17, &lState, NULL);`

e. Save the file.

16. As the code require some extra variables, add the following text just before the last closing brace in the `pspPWMControl.h` file:

```
unsigned int FBInt, REFInt;
int mCurrentCLKCount;
bool mPWStatus;
```

17. Save the `pspPWMControl.h` file.

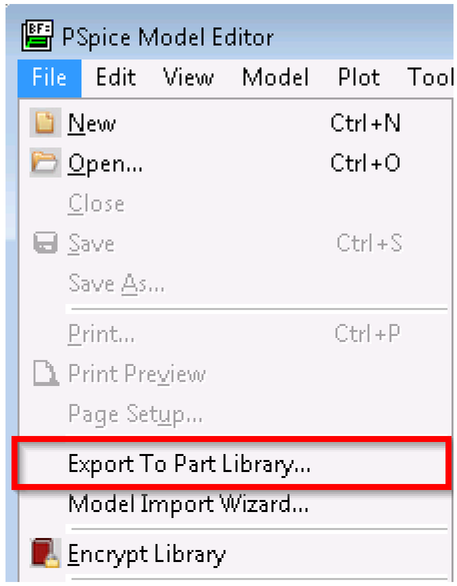
18. Rebuild the Visual Studio project using *Build – Build Solution*.

The model DLL file is built with the required model evalution code.

Note: When you rebuild your solution, ensure that the Configuration is *Release*, not Debug.

19. Once the PSpice library is generated, export the PSpice library to the Capture library using *Export to Part Library* in Model Editor.

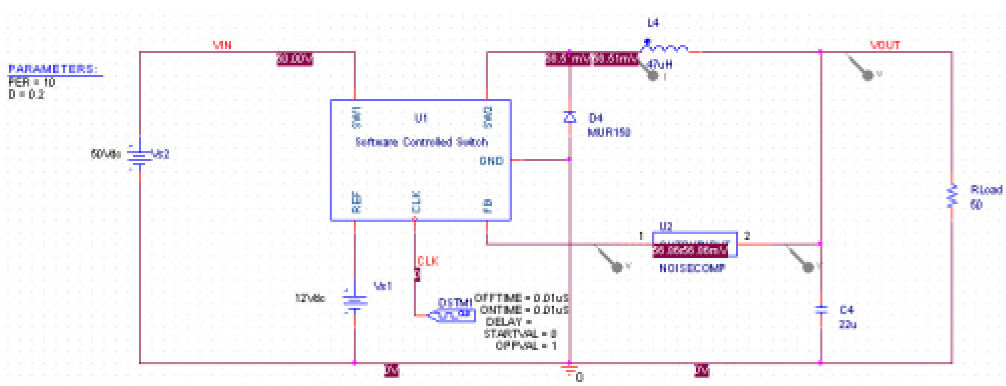
©1991-2015 Cadence Design Systems, Inc. All rights reserved.



20. Open the *DC-DC.dsn* file, present in the `DigitalPowerSupply` folder, in OrCAD Capture.

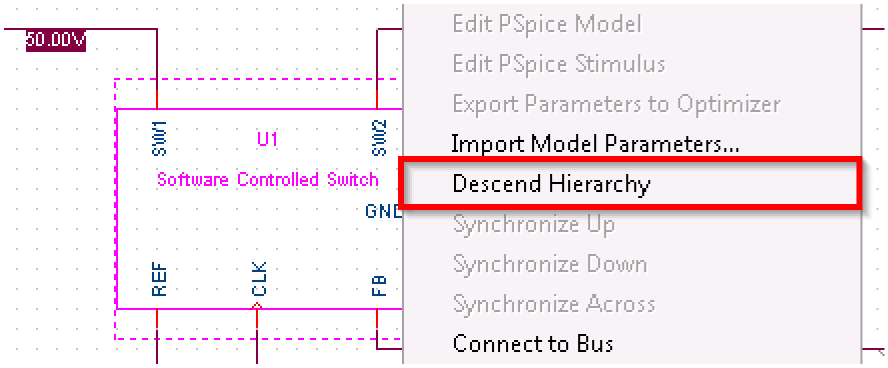
21. Right-click and select *Make Root* to make the BuckConverter-SW-Control schematic as root.

22. Open the `BuckConverter-SW-Control` schematic page.



23. Descend on the *Software Controlled Switch*, that is, *U1*, to see an Software-Controlled PWM Block implementation.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.



24. Activate the *BuckConverter-SW-Control-tran* simulation profile from the project Manager.

Note: For your convienience, the design already has the PWMControl block added. If you want to add your own PWMControl block, ensure that the your part's block shape and pin locations are same as the already added one for minimum modification.

25. Simulate the project and view the output in PSpice as shown in the following screenshot.

Note: Ensure that the PWMControl PSpice library(.lib) is added in the Simulation profile as configured files.

You can note that the Capture design simulated with the Digitally Clocked C/C++ PWMControl part successfully just like any other Capture part.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

4

Generating and Simulating a PSpice DMI Model for Analog Behavioral Circuit

This module describe steps to generate an analog behavioral model based PSpice DMI model. In this module, a MATLAB averaging filter is taken as an example.

In this module, you will:

- Generate a template code for PSpice DMI model
- Use the PSpice DMI model as an averaging filter
- Simulate the PSpice DMI model in a Capture design

Do the following steps to generate the PSpice DMI model and simulate the model in a Capture project:

1. Launch Model Editor.

   If prompted, choose a license that includes PSpice A/D; for example, *OrCAD PSpice Designer Plus*.

2. Select *Model – DMI Template Code Generator*.

3. Enter the following data in the DMI Template Code Generator window to generate an Analog based PSpice DMI model:

   Part Name: `NoiseFilter`

   Part Type: `Analog`

   Model Type: `Function-Dependent Voltage Source`

   DLL Location: `NoiseFilter` folder

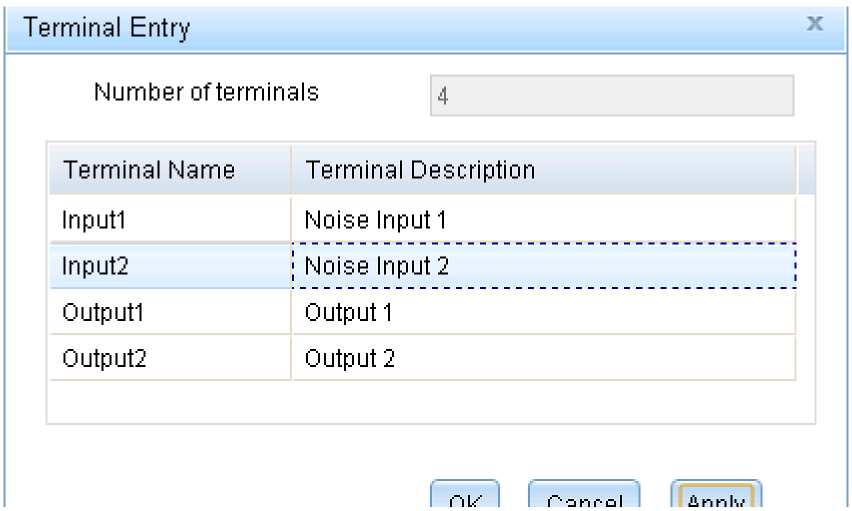
4. Click the *Terminal Entry* box in the Terminals field.

   A Terminal Entry window will be displayed.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

5. By default, the Terminal Entry window has 4 terminals, that are, 2 input terminals and 2 outer terminals.

6. Change the terminal description of both the input terminals to *Noisy Input 1* and *Noise Input 2* instead of Control Input 1 and Control Input 2 and click *OK*.

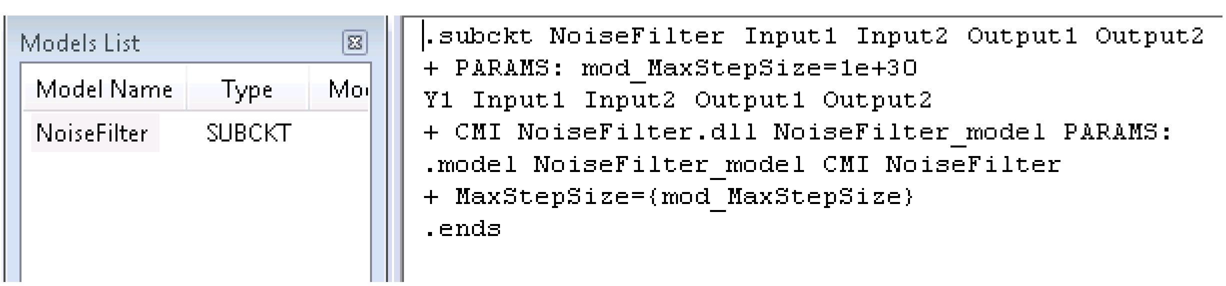


7. Click `OK` on the DMI Template Code Generator window.

   A log file is displayed. A .lib file is successfully created at the specified DLL location and opened in Model Editor.

8. Click on the library name in the Model List window of the Model Editor to see the library infromation.

   In the following screenshot, you can see that the library points to the DLL that is created for the model. You will complete the template model code that was generated on creation of the .dll and .lib file and regenerate the .dll file.



9. Launch Visual Studio Community 2013 in your machine.

10. Click *Open Project* in the Visual Studio's Start Page and browse to the `DLL location` for the Visual Studio Project.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

In this case, the Visual Studio project is `NoiseFilter.vcxproj`.

11. Modify the default configuration in Configuration Manager (*Build – Configuration Manager*) to 64-bit platform using the following steps:

   a. In the `Active Solution Platform` drop-down list, select the `<New...>` option to open the `New Solution Platform` window.

   b. In the `Type or select the new platform` drop-down list, select 64-bit platform and close the window.

12. Build the project using *Build – Build Solution* in the Visual Studio to verify if there are no build issues.

13. Expand NoiseFilter project in Solution Explorer and open the `NoiseFilter_user.cpp` file to edit using the following steps:

   a. Add the following code after `#include "pspNoiseFilter.h"`:

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

   The `averaging_filter.h` file is an MATLAB generated header file that contains the averaging filter function.

   b. Add the following code after `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);
////
```

   This code updates the state vector with respect to the latest input value and calls the averaging_filter function for gain computation.

   c. Save the `NoiseFilter_user.cpp` file.

14. In Visual Studio, right-click on the NoiseFilter in the Solution Explorer and select *Add – Existing Item* to add the MATLAB generated `averaging_filter.c` file to the project

The `averaging_filter.c` is located in the `averaging_filter` folder.

15. Rebuild the Visual Studio project using *Build – Build Solution*.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

The model DLL file is built with the required model evalution code.

Note: When you rebuild your solution, ensure that the Configuration is *Release*, not Debug.

16. Once the PSpice library is generated, export the PSpice library to the Capture library using *Export to Part Library* in Model Editor.

17. Open the *DC-DC.dsn* file, present in the `NoiseFilter` folder, in OrCAD Capture.

18. Right-click and select *Make Root* to make the BuckConverter-SW-Control schematic as root.

19. Open the BuckConverter-SW-Control schematic page.

20. Select Instance `U2`, that is, NOISECOMP.

21. Right-click `U2` and select *Edit Properties* to view the implementation defined as NOISECOMP.

This is added to add noise to the input voltage. The following implementation of NOISECOMP illustrates a random noise being added to the input voltage:

```
.subckt noisecomp OUTPUT input
E_RND OUTPUT 0 VALUE={V(INPUT)+0.3*RND}
R1 input 0 100K
.ends
```

22. Descend on the *Software Controlled Switch*, that is, *U1*, to see an Software-Controlled PWM Block implementation.

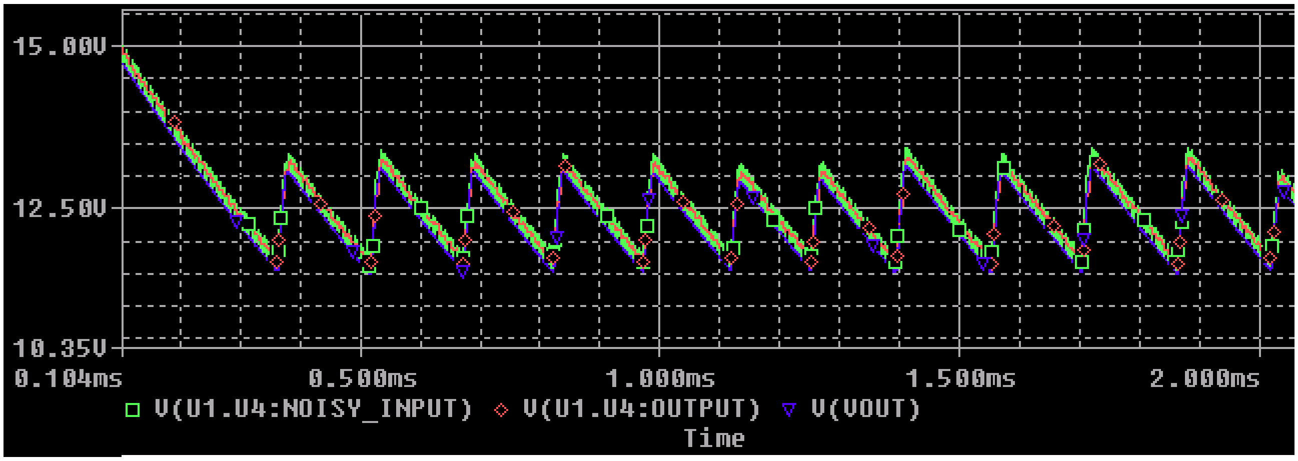
23. Activate the *BuckConverter-SW-Control-tran* simulation profile from the project Manager.

Note: For your convienience, the design already has the *averaging_filter* block added as noisefilter. If you want to add your own *noisefilter* block, ensure that the your part's block shape and pin locations are same as the already added one for minimum modification.

If you have added your own noisefilter block, ensure that the `pin 2` of Input and `pin 4` of Output of the block are connected to `GND`.

24. Simulate the project and view the output in PSpice as shown in the following screenshot.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.



Note: Ensure that the NoiseFilter PSpice library(.lib) is added in the Simulation profile as configured files.

You can note that the Capture design simulated with the Analog NoiseFilter part successfully just like any other Capture part.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

5

Generating and Simulating a PSpice DMI Model for State Model Simulation

This module covers an example of an automotive state model being simulated in PSpice as a PSpice DMI model. It uses an implementation of an automotive power window control module. The control logic is based on a state model which is referred from a MATLAB reference design.

In this module, you will:

- Generate a template code for PSpice DMI model using the DMI Template Code Generator window
- Use the PSpice DMI model for the power window module circuit
- Simulate the PSpice DMI model with respect to power window module circuit

Do the following steps to generate a template code for a PSpice DMI model:

1. Launch Model Editor.

   If prompted, choose a license that includes PSpice A/D; for example, *OrCAD PSpice Designer Plus*.

2. Select *Model – DMI Template Code Generator* to open DMI Template Code Generator window.

3. Enter the following data in the DMI Template Code Generator window to generate a Digital C/C++ based Combinatorial PSpice DMI model:

   Part Name: `StateMachine`

   Part Type: `Digital C/C++`

   Interface Type: `Combinatorial`

   DLL Location: *PSpiceSystems/StateModel/Code*

4. Click on the Ports radio button to enter Input and IO ports for the model.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

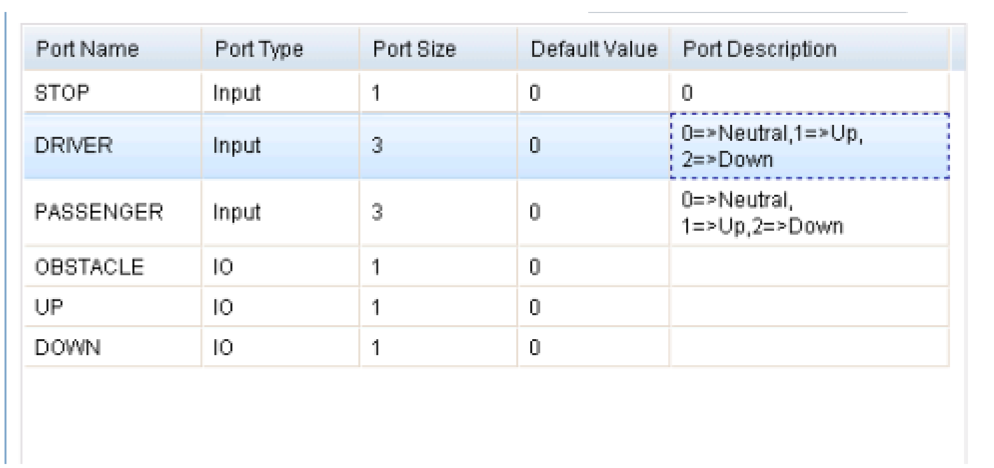
A Port Entry window is displayed.

5. Enter the following information in the Port Entry window:

   Enter number of input ports: 4

   Enter number of IO ports:2

Port Name	Port Type	Port Size	Default Value	Port Description
`STOP`	Input	1	0	
`DRIVER`	Input	3	0	0=>Neutral, 1=>Up, 2=>Down
`PASSENGER`	Input	3	0	0=>Neutral, 1=>Up, 2=>Down
`OBSTACLE`	IO	1	0	
`UP`	IO	1	0	
`DOWN`	IO	1	0	



Port Name	Port Type	Port Size	Default Value	Port Description
STOP	Input	1	0	0
DRIVER	Input	3	0	0=>Neutral,1=>Up, 2=>Down
PASSENGER	Input	3	0	0=>Neutral, 1=>Up,2=>Down
OBSTACLE	IO	1	0	
UP	IO	1	0	
DOWN	IO	1	0	

6. Click *OK* on the Port Entry window.

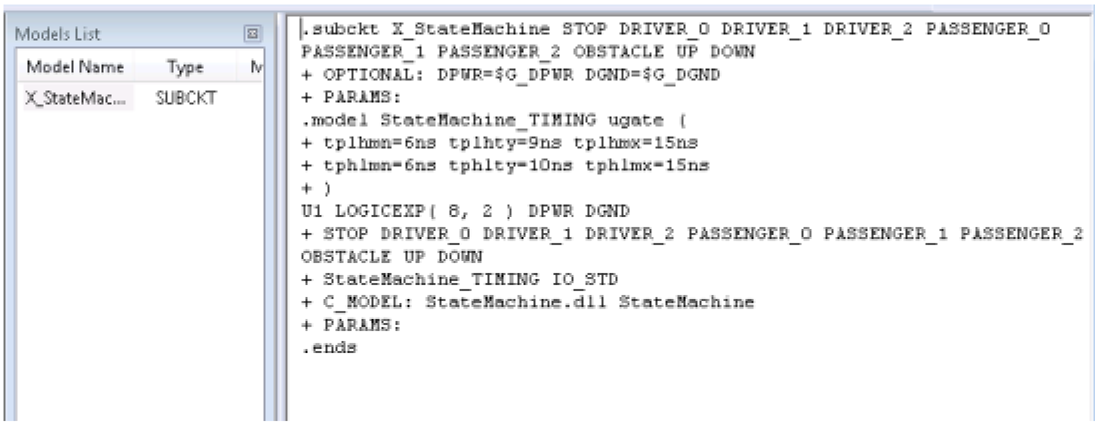
7. Click *OK* on the DMI Template Code Generator window.

   A log file is displayed. A .lib file is successfully created at the specified DLL location and opened in Model Editor.

8. Click on the library name in the Model List window of the Model Editor to see the library infromation.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

In the following screenshot, you can see that the library points to the DLL that is created for the model. You will complete the template model adaptor code that was generated on creation of the .dll and .lib file and regenerate the .dll file.



9. Launch Visual Studio Community 2013 in your machine.

10. Click *Open Project* in the Visual Studio's Start Page and browse to the `DLL location` for the Visual Studio Project.

    In this case, the Visual Studio project is `StateMachine.vcxproj`.

11. Modify the default configuration in Configuration Manager to 64-bit platform as described in Step 13 of Chapter 3.

12. Build the project using *Build – Build Solution* in the Visual Studio to verify if there are no build issues.

13. Expand StateMachine project in Solution Explorer and open the `StateMachine_user.cpp` file to edit it using the following steps:

    a. Add the following code after `#include "pspStateMachine.h"`:

    ```
    #include "../FSM.cpp"
    ```

    The `FSM.cpp` file contains the implementation of State Machine Model.

    b. Add the following code after // LOGIC TO BE IMPLEMENTED BY USER:

    ```
    int driverVect[3];
    driverVect[0] = (int)DRIVER[0];
    driverVect[1] = (int)DRIVER[1];
    driverVect[2] = (int)DRIVER[2];
    int passengerVect[3];
    passengerVect[0] = (int)PASSENGER[0];
    passengerVect[1] = (int)PASSENGER[1];
    ```


©1991-2015 Cadence Design Systems, Inc. All rights reserved.

```
passengerVect[2] = (int)PASSENGER[2];
int obstacleInt;
if((int)OBSTACLE == 1)
    obstacleInt = 1;
else
    obstacleInt = 0;
int stopInt;
if((int)STOP == 1)
    stopInt = 1;
else
    stopInt = 0;
setState(stopInt, obstacleInt, driverVect, passengerVect, &currentState,
&nextState, &prevState, &windowMovementOutput, timer);
if(windowMovementOutput.moveUp == 1)
    UP = 1;
else
    UP = 0;
if(windowMovementOutput.moveDown == 1)
    DOWN = 1;
else
    DOWN = 0;
```

The setState function is the promary function that updates the current state of the State Machine and output signals of the Power Window Control module with respect to input signals and the last state of the State Machine.

c. Edit `fp_SetState(mRef, j, &lState, NULL);` to `fp_SetState(mRef, j-8, &lState, NULL);` for the two different instances.

14. Add the following code in the `pspStateMachine.h` file after `#include "pspiceDigApiDefs.h"`:

```
#include "../FSM.h"
```

15. Add the following code in the `pspStateMachine.h` file after `double mPrevTicks;`:

```
// add required class variables here
int timer;
states currentState;
states prevState;
states nextState;
struct window_movement windowMovementOutput;
```

16. Rebuild the Visual Studio project using *Build – Build Solution*.

Ensure that the *Release* configuration is selected.

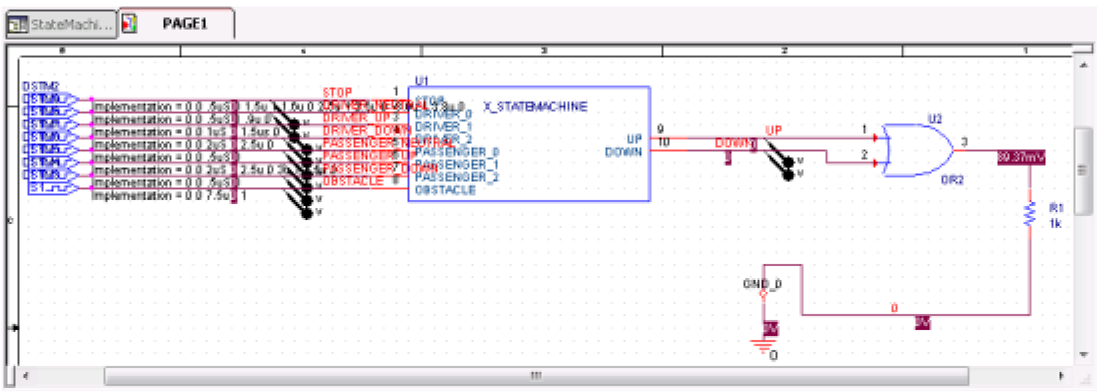
©1991-2015 Cadence Design Systems, Inc. All rights reserved.

17. Once the PSpice library is generated, export the PSpice library to the Capture library using *Export to Part Library* in Model Editor.

18. Open the *StateMachine.dsn* file, present in the `StateModel` folder, in OrCAD Capture.

19. Right-click and select *Make Root* to make the Schematic1 schematic as root.

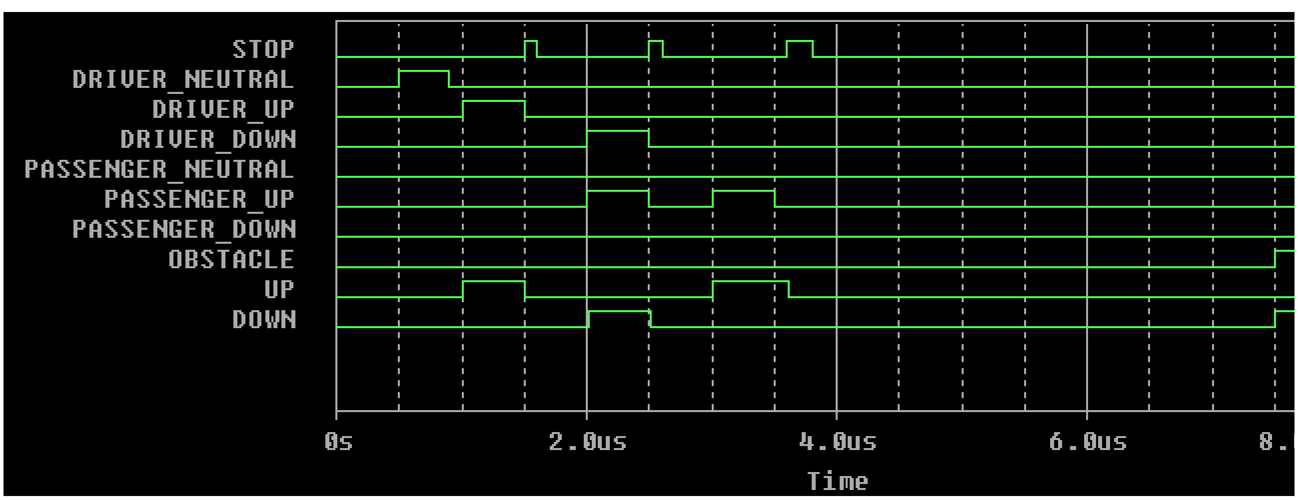
20. Open the Page1 schematic page.



21. Activate the *Schematic1-tran* simulation profile from Project Manager.

Note: For your convienience, the design already has the *statemachine* block added. If you want to add your own *statemachine* block, ensure that the your part's block shape and pin locations are same as the already added one for minimum modification.

22. Simulate the project and view the output in PSpice as shown in the following screenshot.



Note: Ensure that the StateMachine PSpice library(.lib) is added in the Simulation profile as configured files.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

You can note that the Capture design simulated with the Digital C/C++ Combinatorial part successfully just like any other Capture part.

The State Transition chart is provided in a .csv file to verify if the state model transition is correct.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

6

Generating and Simulating a Verilog-A file based PSpice DMI Model

This module illustrates importing of a Verilog-A file and translating the file to a PSpice DMI model. The DMI Template Code Generator feature supports Verilog-A file import using the ADMS parser.

In this module, you will:

- Import the Verilog-A file using Model Editor and convert it to a PSpice DMI model
- Simulate the PSpice DMI model and compare the DMI model's results with the regular capacitor simulation results

Do the following steps to generate a PSpice DMI model from a Verilog-A file:

1. Launch Model Editor.

   If prompted, choose a license that includes PSpice A/D; for example, *OrCAD PSpice Designer Plus*.

2. Select *Model – DMI Template Code Generator*.

   You can verify the path to the nom.lib file from: *Simulation Settings window - Configuration Files tab - Library category*.

3. Enter the following data in the DMI Template Code Generator window to generate a VerilogA-ADMS based PSpice DMI model:

   Part Name: `cap`

   Part Type: `VerilogA-ADMS`

   Verilog-A File: `<Path to cap.va>`

   XML Folder: `<Installation Path>\tools\pspice\api\adms\xmls`

   DLL Location: `VerilogA folder`

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

The `cap.va` fle is a verilog-A model for a capacitor that 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
```

4. Click *OK* on the DMI Template Code Generator window.

   The PSpice DMI model(.lib) is auto-generated from the verilog-A file, and a log file is generated.

5. If you get any build error during PSpice DMI model generation, debug the model behaviour using a visual studio project file (.vcxproj) in Visual Studio Community 2013.

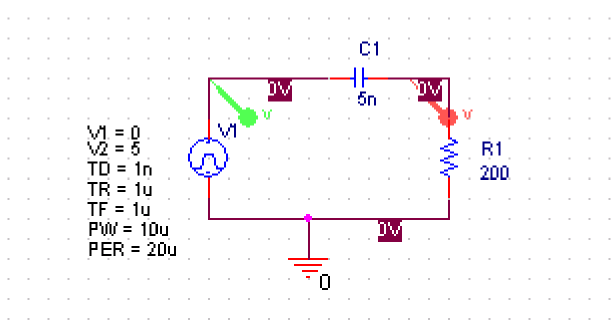
   The visual studio project file gets generated during the PSpice DMI model generation process.

6. Once the PSpice library is successfully generated, export the PSpice library to the Capture library using *Export to Part Library* in Model Editor.

7. Open the *Design1.dsn* file, present in the `VerilogA` folder, in OrCAD Capture.

   The *Design1*.dsn file has two schematics - *cap* and *capDMI*.

8. Open the Page1 schematic page of the cap schematic.

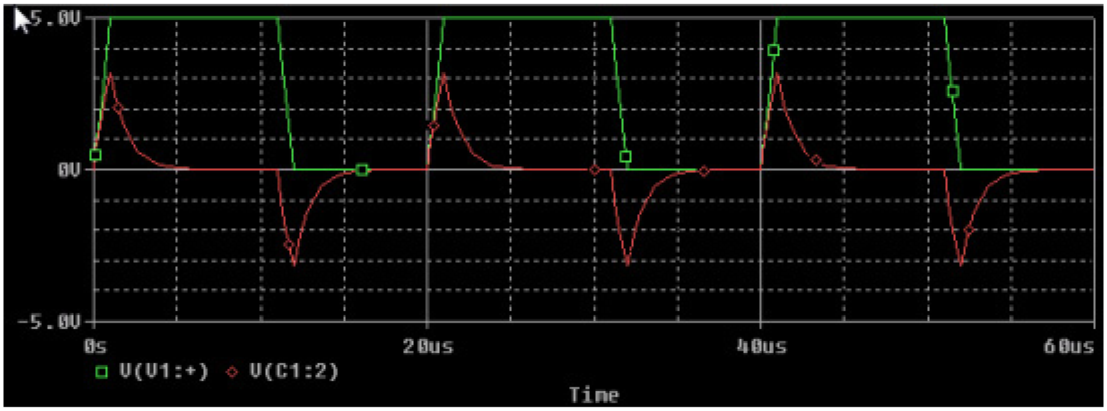


©1991-2015 Cadence Design Systems, Inc. All rights reserved.

9. If not already activated, activate the *cap-tran* simulation profile from Project Manager.

   Note: For your convienience, the design already has a *capacitor* added.

10. Simulate the project and view the output in PSpice as shown in the following screenshot.

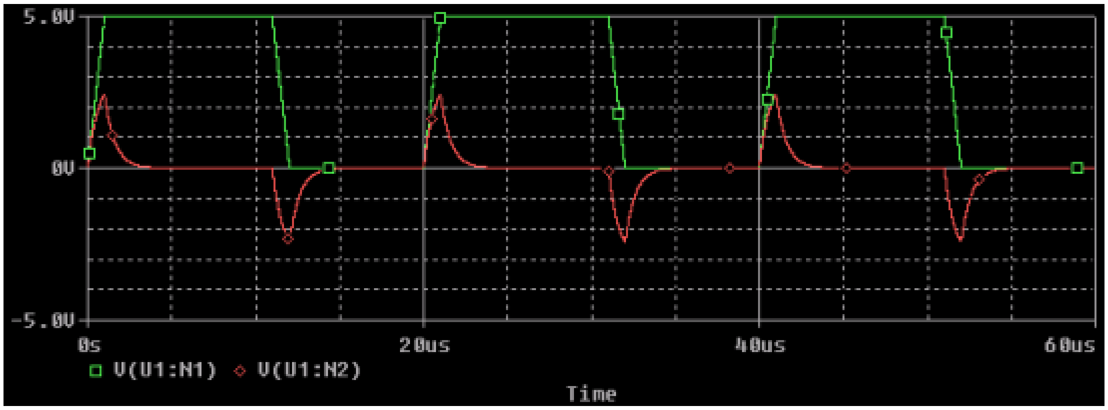


11. Change the simulation profile to *capDMI-tran*.

    Note: For your convienience, the page1 of the capDMI schematic has the *DMICAP* block added for capacitor. If you want to add your own *DMICAP* block, ensure that the your part's block shape and pin locations are same as the already added one for minimum modification.

    Note: Ensure that the *capDMI-tran* Simulation profile has *cap.lib* as configured library.

12. Run Simulation and view the output in PSpice as shown in the following screenshot.



The PSpice DMI model uses an equation C1 + 2*C2 to calculate value of equivalent capacitance.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

7

Generating and Simulating a SystemC based PSpice DMI Model

This module covers a simple example of generating and simulating a SystemC based PSpice DMI Model.

In thi module, you will:

- Write a Finite Impluse Response (FIR) filter model in SystemC
- Generate a PSpice DMI Template Code for the SystemC based PSpice DMI model using Model Editor
- Integrate the SystemC model with the DMI Template Code

Do the following steps to generate and simulate a SystemC based PSpice DMI model:

1. Launch Model Editor.

   If prompted, choose a license that includes PSpice A/D; for example, *OrCAD PSpice Designer Plus*..

2. Select *Model – DMI Template Code Generator*.

3. Enter the following data in the DMI Template Code Generator window to generate a Digital C/C++ based Combinatorial PSpice DMI model:

   Part Name: `FIR`

   Part Type: `SystemC`

   Interface Type: `Clocked`

   DLL Location: *PSpiceSystems/SystemC*

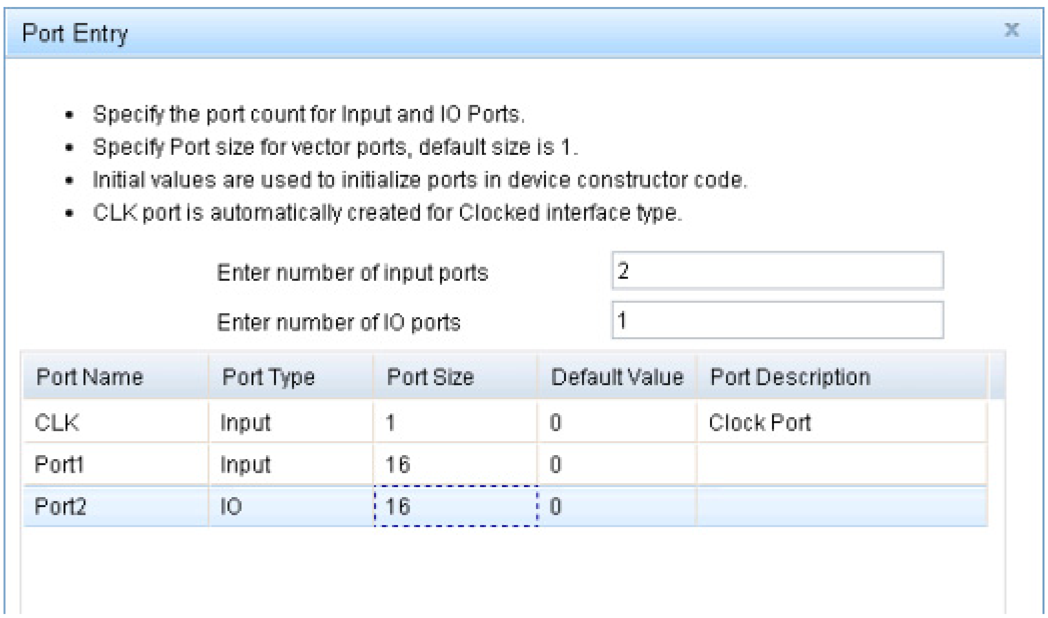
4. Click on the Ports radio button in the DMI Template Code Generator window to enter the following data:

   Enter number of input ports: 2

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

Enter number of IO ports: 1

Port Name	**Port Type**	**Port Size**	**Default Value**	**Port Description**
`CLK`	Input	1	0	Clock Port
`input`	Input	16	0	
`output`	IO	16	0	

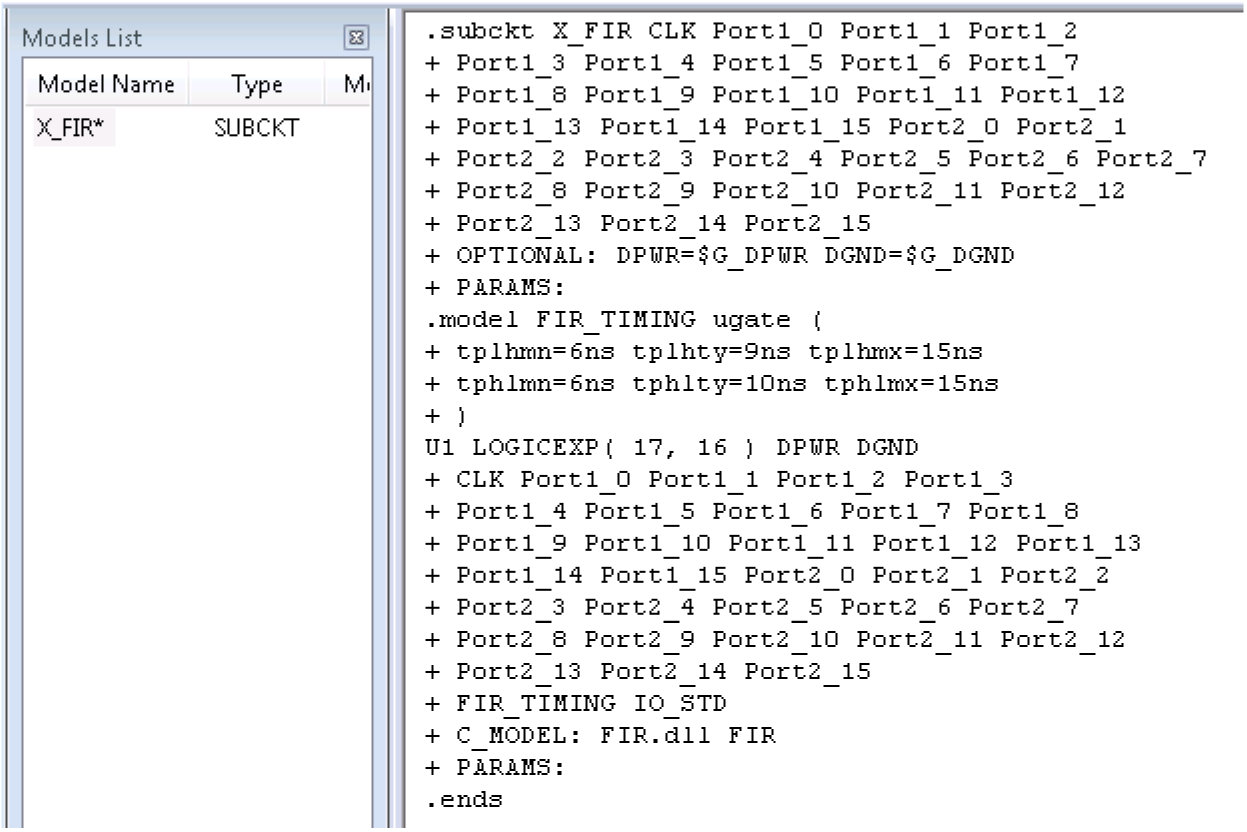


5. Click *OK* on the Port Entry window.
6. Click *OK* on the DMI Template Code Generator window.

   The DMI template code for the SysytemC model is generated and the log file is displayed in the text editor. The PSpice library (.lib) is also generated successfully.

   You can note that the generated library has pointer to a .dll file, that is, in this case `FIR.dll`. Now in some of the next steps you will add a model code to the generated adaptor code.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.



7. Launch Visual Studio Community 2013 in your machine.

8. Click *Open Project* in the Visual Studio's Start Page and browse to the `DLL location` for the Visual Studio Project.

   In this case, the Visual Studio project is `FIR.vcxproj`.

9. Modify the default configuration in Configuration Manager to 64-bit platform as described in Step 13 of Chapter 3.

10. Build the project using *Build – Build Solution* in the Visual Studio to verify if there are no build issues.

11. Expand FIR project in Solution Explorer and open the `SysCFIR.cpp` file to edit it using the following steps:

    a. Search for `SysCFIR::entry` function in `SysCFIR.cpp` and uncomment the following code inside the function. This code implements an FIR filter using SystemC.

```
void SysCFIR::entry() {
    // const sc_uint<8> coef[5] = { 18, 77, 107, 77, 18 };
```


©1991-2015 Cadence Design Systems, Inc. All rights reserved.

```
    // sc_int<16> taps[5];
    // //reset code
    // output.write(0);
    // //reset internal variables
    // //reset outputs
    // wait();
    // while (true) {
        //  //read inputs
        //  for (int i = 4; i > 0; i--) {
                //      taps[i] = taps[i - 1];
            //  }
        //  taps[0] = input.read();
        //  //algorithm
        //  sc_int<16> value;
        //  for (int i = 0; i < 5; i++) {
            //      value += coef[i] * taps[i];
        //  }
        //  //write outputs
        //  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"<< hex <<
value << "]" << endl;
        //  wait();
    // }
}
```

b. Add the following line after `m_SysCFIR->CLK(sysCsig_CLK);` in the `pspSysCFIR::pspSysCFIR(const char* pInstName, void*pRef)` function of the `pspSysCFIR.cpp` file:

```
m_SysCFIR->reset(sysCsig_reset);
```

12. Rebuild the Visual Studio project using *Build – Build Solution*.

Ensure that the *Release* configuration is selected.

13. Once the PSpice library is generated, export the PSpice library to the Capture library using *Export to Part Library* in Model Editor.

14. Open the *Design1.dsn* file, present in the `SystemC` folder, in OrCAD Capture.

15. If required, right-click and select *Make Root* to make the Schematic1 schematic as root.

16. Open the Page1 schematic page.

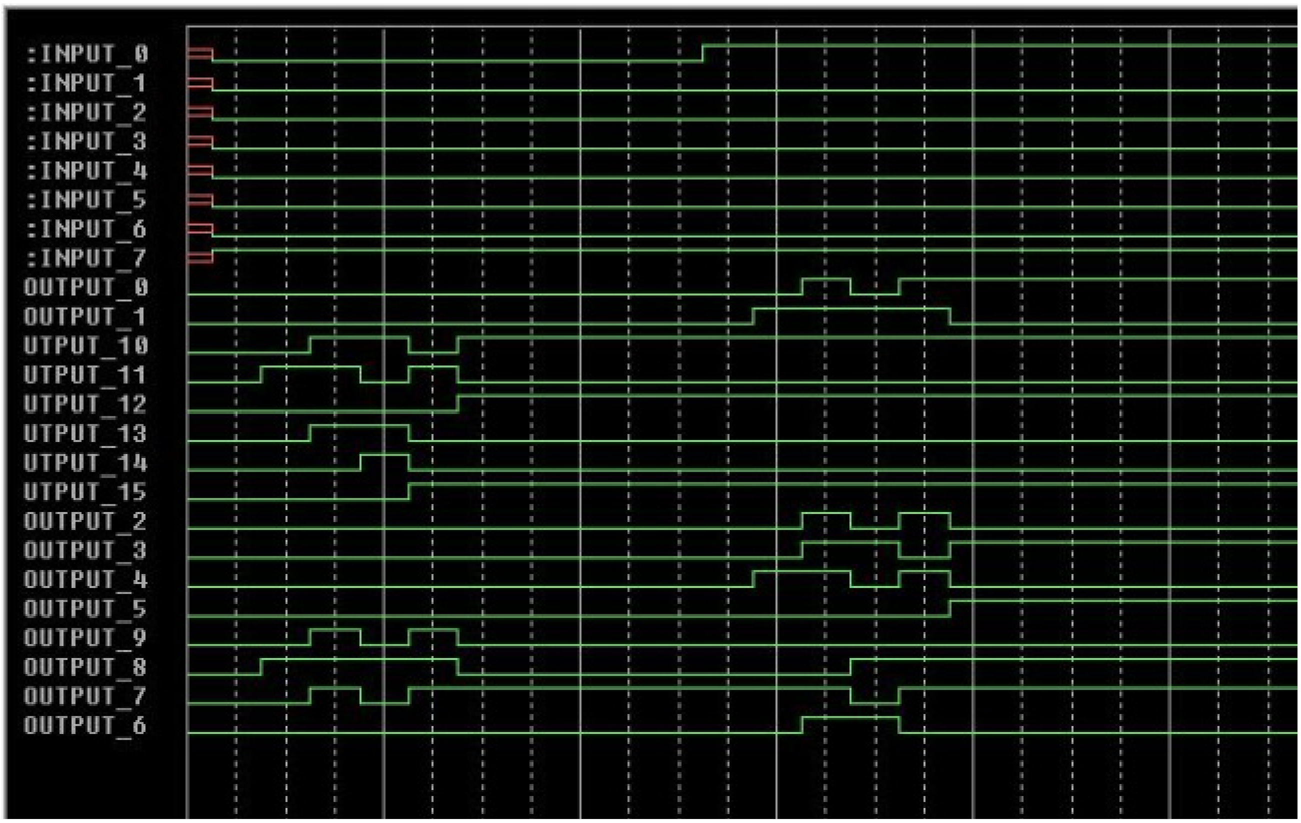
©1991-2015 Cadence Design Systems, Inc. All rights reserved.

17. Activate the *Schematic1-tran* simulation profile from Project Manager.

Note: For your convienience, the design already has the *FIR* block added. If you want to add your own *FIR* block, ensure that the generated part's block shape and pin locations are same as the already added one for minimum modification.

Note: In the Capture design, if you have used the generated SystemC based PSpice DMI model instead of the default model. Modify `+ C_MODEL: FIR.dll FIR` to `+ C_MODEL: FIR.dll SysCFIR` in the generated PSpice library file (.lib).

18. Simulate the project and view the output in PSpice as shown in the following screenshot:



Note: Ensure that the FIR PSpice library(.lib) is added in the Simulation profile as configured files.

You can note that the Capture design simulated with the SystemC part successfully just like any other Capture part.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.

©1991-2015 Cadence Design Systems, Inc. All rights reserved.