SWIPT Spice and Verilog Tutorial

Bram Veraverbeke, Jonah Van Assche, Roel Uytterhoeven

July 2020

1 Introduction

In this tutorial, the simulation part of the SWIPT assignment is discussed. We will use two types of simulators for SWIPT, a pure SPICE simulator called LTSpice and a mixed-signal simulator that can co-simulate SPICE and digital hardware called SimVision. For a more elaborate LTSpice tutorial, take a look at the tutorial called *SWIPT_LTSpiceTutorial*.
The tutorials below will cover SimVision (Tutorial 2-5) and how it is possible to create Spice netlists from LT-Spice and optionally use them in SimVision (Tutorial 1). LTSpice can be downloaded at `https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html`, SimVision is normally available for students at ESAT (login on your Linux account, or make one if you don't have one yet). Under *Tutorial Files*, you can find the necessary files for the tutorials. Good luck!

2 Analog Simulation Using LTSpice

LTSpice is a software tool that allows to draw circuit schematics and run analog simulations on your circuits. It supports the typical Spice-level simulation types, e.g. transient (time domain) and ac (frequency domain), and many others. In this project, you will need this tool to design, simulate and understand the analog parts of your power/data transfer circuit. General information and a getting started guide on LTSpice can be found at

Note that Spice simulations typically do not allow for complex digital control. For instance, simulating your frequency tracking algorithm would be extremely complex at this point. This is solved by using mixed-signal simulations where the digital control is written in Verilog and the analog circuit is represented by a Spice-netlist. The details of these simulations are discussed later on in this document, but the next section already explains how you can obtain a Spice-netlist from an LTSpice schematic.

2.1 Tutorial 1: LTSpice to Spice-netlist

This tutorial starts from the example circuit shown in Figure 1. Next all the steps to extract the netlist of this schematic are discussed.

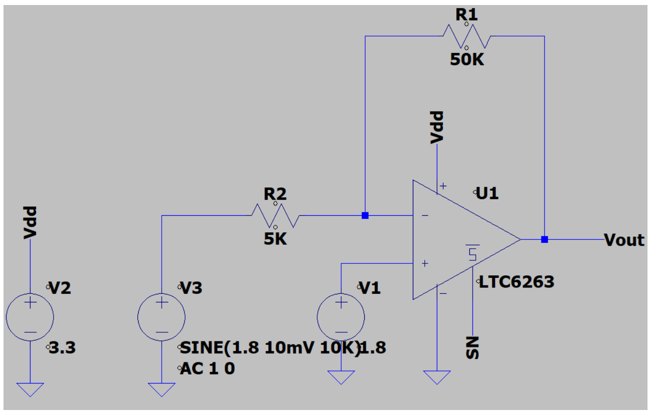
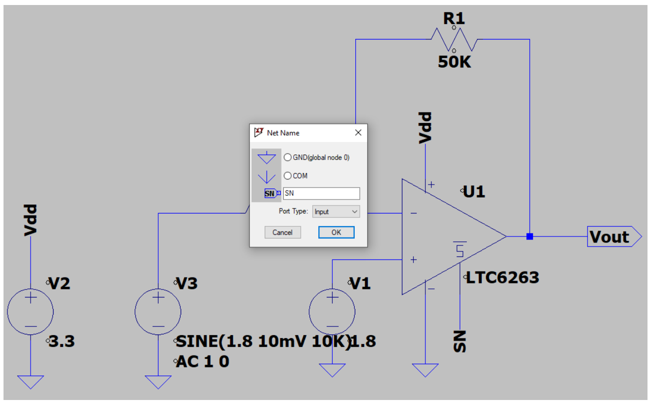


Figure 1: Example LTSpice design of an amplifier that has a shutdown option

1. Create ports for any signal that you want to access in the mixed-signal simulations as shown in Figure 2 by right clicking the netname and setting the port type. Pure digital signals normally have a "strong" direction and are thus either input or output. Analog signals are typically bi-directional, but should not be used as a driver for a digital signal (since once in the digital domain it will become a "1" or a "0").



SN

Port Type: Input

Figure 2: Adding ports to a schematic

2. Create a symbol for this schematic as shown in Figure 3.

3. The symbol should look like Figure 4. Save it.

4. Create a new schematic and add the symbol as a component as shown in Figure 5. If you cannot find your symbol name, double check that the "top directory" matches the save location of your symbol.

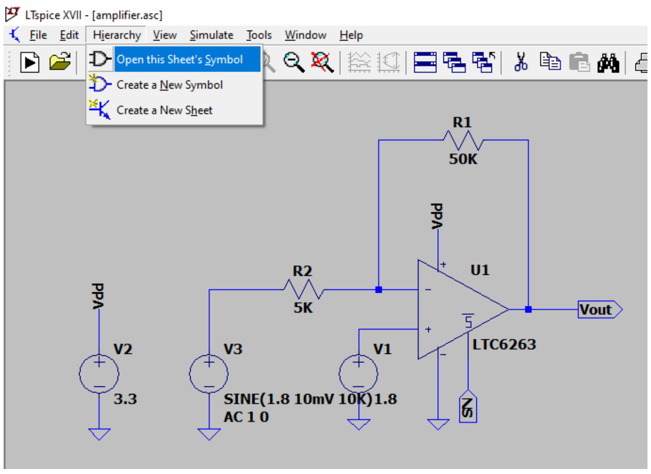


Figure 3: Creating a symbol from a schematic

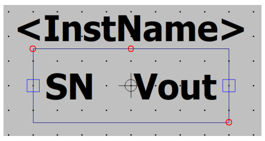
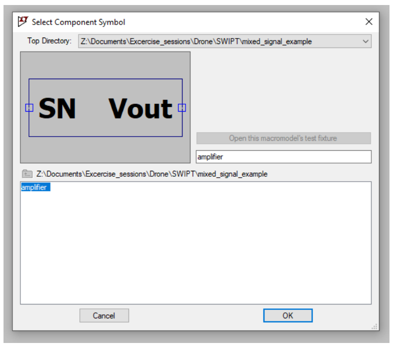
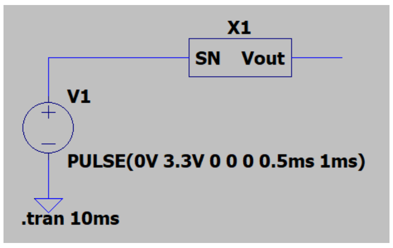


Figure 4: The resulting symbol


Select Component Symbol
Top Directory: Z:\Documents\Excercise_sessions\Drone\SWIPT\mixed_signal_example
SN Vout
Open this macromodel's test fixture
amplifier
Z:\Documents\Excercise_sessions\Drone\SWIPT\mixed_signal_example
amplifier
Cancel
OK

(a) Add the symbol



(b) The resulting netlist

Figure 5: Creating a netlist in which the amplifier's symbol is used

5. (OPTIONAL) Run the simulation to check that the symbol does indeed work and contains the correct circuit. Fig 6 shows the transient waveforms. The amplification is triggered by ”SN” being high.

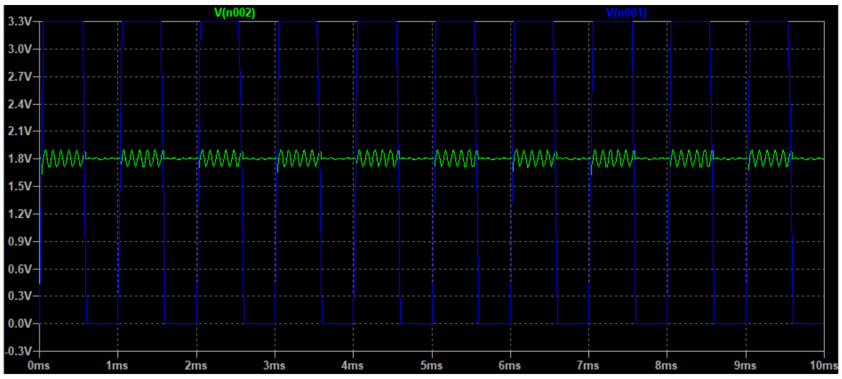


Figure 6: Transient waveforms of the amplifier circuit

6. View the Spice-netlist as shown in Figure 7 by right clicking on any open space in the document.

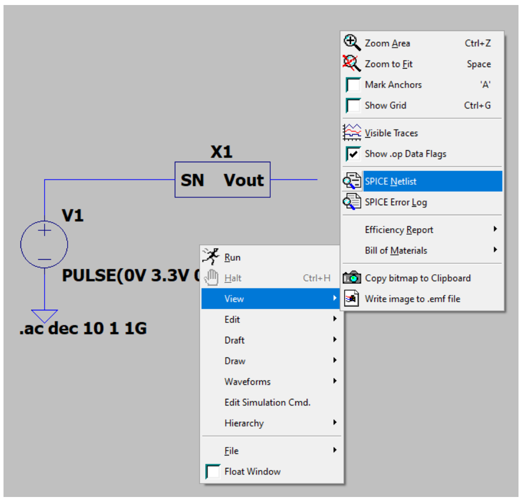
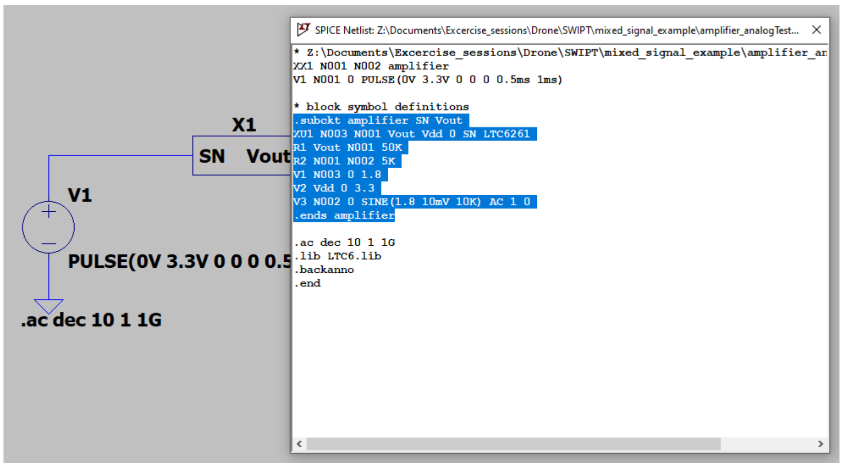


Figure 7: Open the netlist

7. Select and copy the SUBCKT of interest, the amplifier in this case, as shown in Figure 8. You can now use this SUBCKT in other netlists and/or in mixed signal simulations. Note that, if any ".lib" files or other source files are included in the shown netlist (here this would be "LTC6.lib"), you will need to ensure that these files can be found when using the SUBCKT outside LTSpice. A simple solution is to copy the included file to the run directory of your simulation. Alternatively, you can update the path of the file so it will be found correctly. The location of native LTSpice libraries can be seen when adding the component to the schematic.


X1
SN Vout
V1
PULSE(0V 3.3V 0 0 0 0.5
.ac dec 10 1 1G
SPICE Netlist: Z:\Documents\Excercise_sessions\Drone\SWIPT\mixed_signal_example\amplifier_analogTest...

```
* Z:\Documents\Excercise_sessions\Drone\SWIPT\mixed_signal_example\amplifier_an
XX1 N001 N002 amplifier
V1 N001 0 PULSE(0V 3.3V 0 0 0 0.5ms 1ms)

* block symbol definitions
.subckt amplifier SN Vout
XU1 N003 N001 Vout Vdd 0 SN LTC6261
R1 Vout N001 50K
R2 N001 N002 5K
V1 N003 0 1.8
V2 Vdd 0 3.3
V3 N002 0 SINE(1.8 10mV 10K) AC 1 0
.ends amplifier

.ac dec 10 1 1G
.lib LTC6.lib
.backanno
.end
```

Figure 8: The final netlist. Normally you only need the subckt and the included libraries

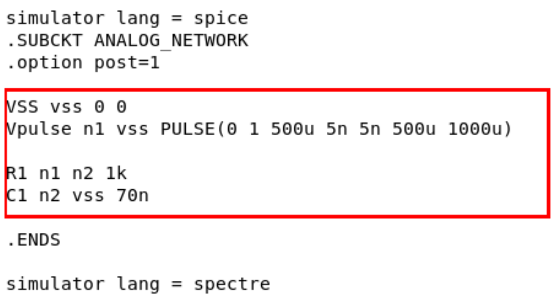
3 Analog Simulations Using Simvision

LTSpice is not suited for complex digital simulations. Simvision is a tool from Cadence that allows to do transient simulations from both analog circuits (represented by Spice-netlists) and digital control (represented by Verilog code). This section will learn you how to do analog simulations. Section 4 dives deeper into digital simulations, while Section 5 combines analog circuits and digital control into mixed-signal simulations.

3.1 Tutorial 2: RC-network

The purpose of this tutorial is to implement an RC-network in Spice and and simulate it using transient simulations.

1. Start the computer under CentOs (Linux).
2. Create a new directory (It it best not to use any spaces in the directory name). Copy the content of the *Analog_example_circuit_blank* directory into your new directory.
3. For the SWIPT-project we will use Spice files (with a ".spi" filename extension) to describe our analog circuits. Open *ANALOG_NETWORK.spi* , and add what is in the red box of Figure 9.



```
simulator lang = spice
.SUBCKT ANALOG_NETWORK
.option post=1

VSS vss 0 0
Vpulse n1 vss PULSE(0 1 500u 5n 5n 500u 1000u)

R1 n1 n2 1k
C1 n2 vss 70n

.ENDS

simulator lang = spectre
```

Figure 9: Adding an RC-network to the Spice file

If this is the first time you encounter Spice-netlists, it might be somewhat difficult to read. But don't worry, after a few weeks, you will see that it is not that difficult after all. A netlist is a description of how your electronic devices (resistors, inductors, capacitors, transistors,...) are interconnected. The drawback of netlists is that a wrong connection (e.g. a short circuit) is easily made. In order to avoid mistakes, it is important that you always draw the circuit on a piece of paper (or import the netlist from a visual circuit simulator like LTSpice). This will make it a lot easier to reason about your circuit.

Lets take a closer look to what we added to our *ANALOG_NETWORK.spi* file. The first line creates a dc voltage source of $0V$ between the ground and the *vss* node. Hence the *vss* node is simply connected to the ground. The second line creates a pulse source between nodes *n1* and *vss* with period of $1000\mu s$ and a duty cycle of 50%. The third line creates a resistor of $1k\Omega$ between nodes *n1* and *n2*. The last line creates a capacitor of 70nF between nodes *n2* and *vss*. A schematic of this circuit is shown in Figure 10. It it nothing more than a low-pass filter. More info on Spice-netlists can be found at `https://www.seas.upenn.edu/~jan/spice/spice.overview.html`, or use Google to find more details.

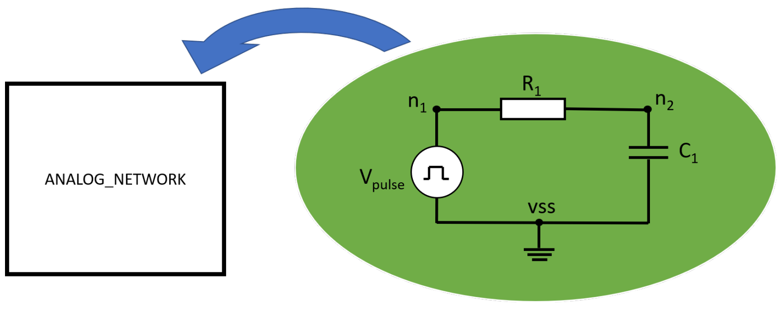


Figure 10: Schematic of circuit described in Figure 9

4. Open a terminal and navigate to the directory you created at the beginning of this tutorial. Enter the following command:

   ```
   > sh run.sh
   ```

5. The design browser window shown in Figure 11 will open. To start the simulation press the play-button in the upper left corner.

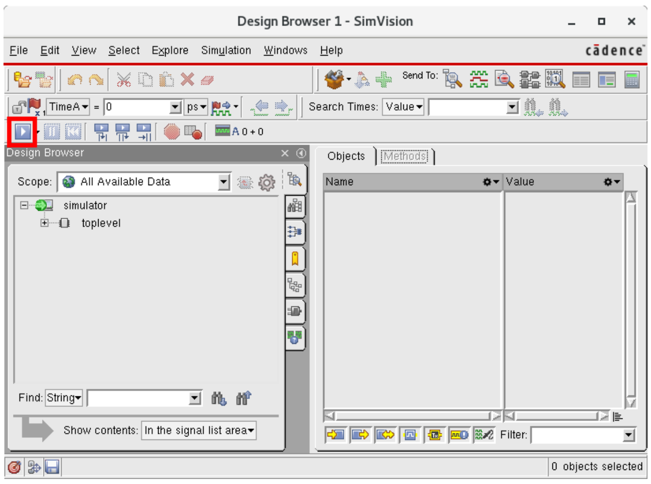


Figure 11: Design browser

6. Browse to *analog_input_inst* and select the signals at the right (*n1* and *n2*). Right click on the selected signals and choose "Send to Waveform Window" as shown in Figure 12.

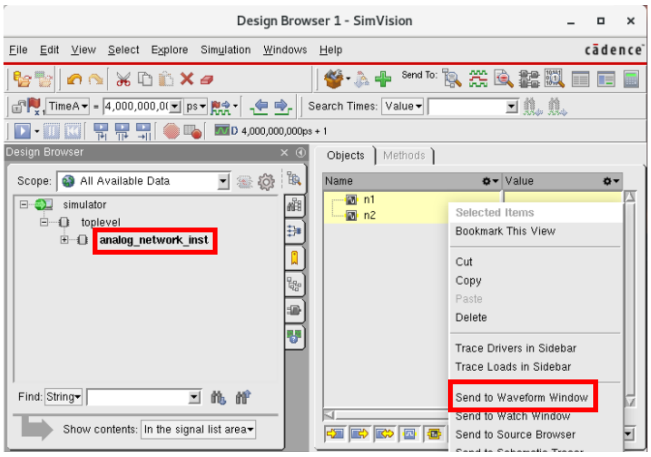


Figure 12: Send signals to waveform window

7. When the waveform window opens, zoom out so that you can see the waveforms better. It is possible that even now the waveforms are barely visible. You can solve this by clicking on the squares at the right, indicated by the blue boxes in Figure 13. The orange waveform at the top is the input of the RC-filter, the red waveform at the bottom is the output. It is clear that the output is the low passed version of the input.

8. If you have adapted the Spice code and want to rerun the simulation, click on "simulation" and then on "Reinvoke Simulator..." as shown in Figure 14. You will get a message. Click 'Yes'. Press the play-button to start the simulation again.

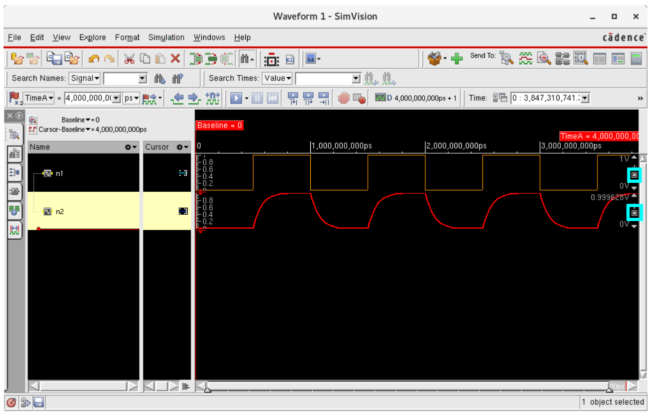


Figure 13: In- and output of the RC-network

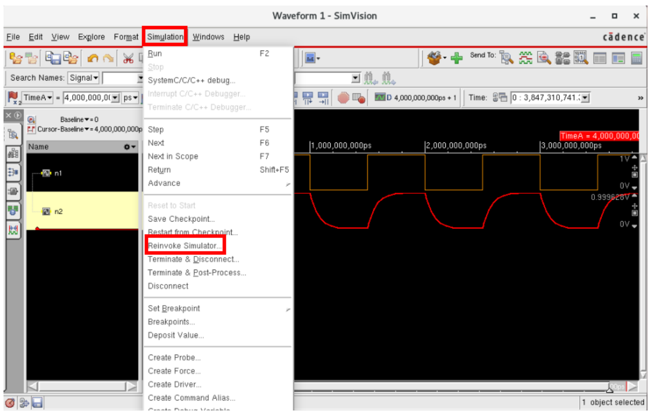


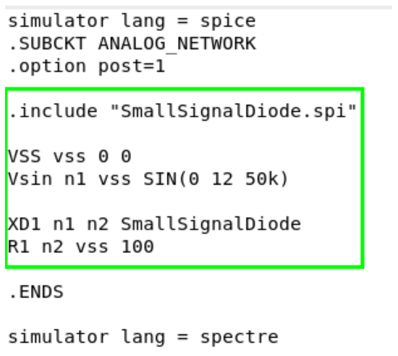
Figure 14: Reinvoke simulator

3.2 Tutorial 3: Subcircuits and parameters

The purpose of this tutorial is to introduce subcircuits and parameters as they are used in Spice. It also explains how to use the expression calculator of SimVision.

1. Start the computer under CentOs (Linux).

2. We will work in the same directory we created at the beginning of Tutorial 2.



```
simulator lang = spice
.SUBCKT ANALOG_NETWORK
.option post=1

.include "SmallSignalDiode.spi"

VSS vss 0 0
Vsin n1 vss SIN(0 12 50k)

XD1 n1 n2 SmallSignalDiode
R1 n2 vss 100

.ENDS

simulator lang = spectre
```

Figure 15: New content of *ANALOG_NETWORK.spi*

3. Open *ANALOG_NETWORK.spi* , and replace what you added in Tutorial 2 (Figure 9) with the lines in the green box in Figure 15. The first line includes the *SmallSignalDiode.spi*-file that contains the Spice model of a diode. This model is implemented as a subcircuit (more info on `https://www.seas.upenn.edu/~jan/spice/spice.overview.html#Subcircuits`).

The circuit implemented in this file consists of a diode and a resistor of 100Ω and a sine source with a DC-value of $0V$, an amplitude of $12V$ and a frequency of 50kHz as shown in Figure 16.

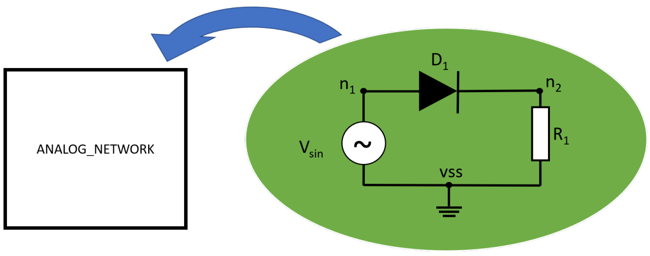


Figure 16: Schematic of the circuit described in Figure 15

4. Start the simulator as explained in steps 4-6 of Tutorial 2. If you did everything right you should see something like in Figure 17 (after zooming in). If you forgot to send a signal to the waveform window, you can still add it by clicking on the tab indicated by the green box in the upper left corner of Figure 17. Now you can browse the all the signals and plot them by clicking on them (single click, no double click).

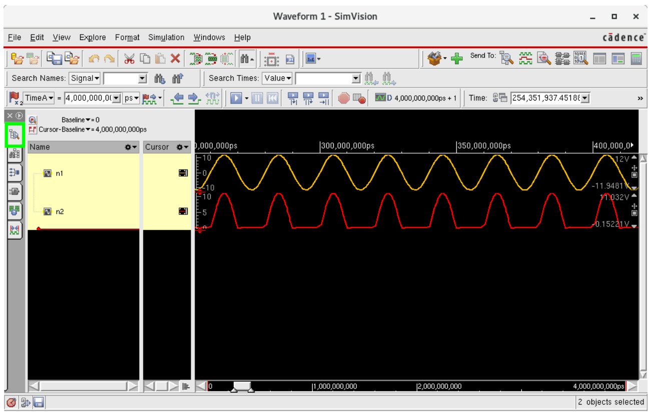


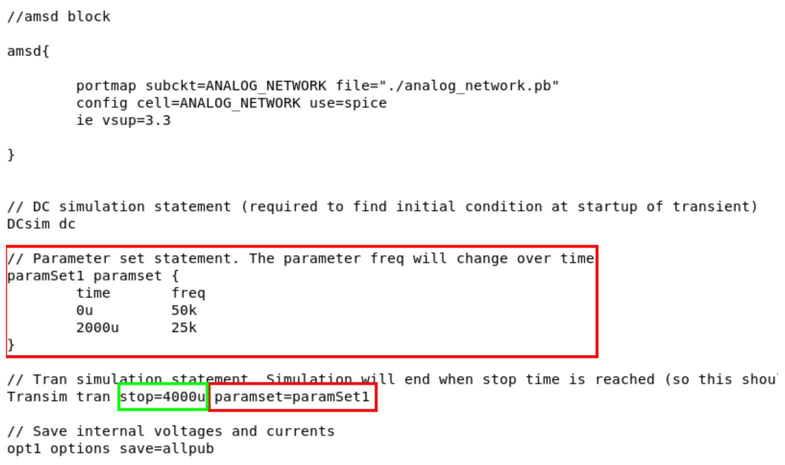
Figure 17: Anode (n1) and cathode (n2) voltage in waveform window

5. Now you will learn how to use parameters in Spice. Open *ANALOG_NETWORK.spi*, and add what is in the red boxes of Figure 18. Make sure that the *".param freq = 50k"* line is stated before the definition of the subcircuit.

```
simulator lang = spice

.param freq = 50k

.SUBCKT ANALOG_NETWORK
.option post=1

.include "SmallSignalDiode.spi"

VSS vss 0 0
Vsin n1 vss SIN(0 12 freq)

XD1 n1 n2 SmallSignalDiode
R1 n2 vss 100

.ENDS

simulator lang = spectre
```

Figure 18: Adding parameters to the Spice file

6. Open *amsSim.scs*. This file contains the top level analog testbench. You can find more details about the simulation file structure in Tutorial 5 (see Section 5). Add what is in the red boxes of Figure 19. The first 2000μs of the transient simulation the frequency of the sine we will be 50kHz. From then on the frequency will be 25kHz, until the simulation will stop at 4000μs. If you want to simulate for a longer time, you can adapt the value in the green box.



```
//amsd block

amsd{

        portmap subckt=ANALOG_NETWORK file="./analog_network.pb"
        config cell=ANALOG_NETWORK use=spice
        ie vsup=3.3

}


// DC simulation statement (required to find initial condition at startup of transient)
DCsim dc

// Parameter set statement. The parameter freq will change over time
paramSet1 paramset {
        time       freq
        0u         50k
        2000u      25k
}

// Tran simulation statement. Simulation will end when stop time is reached (so this shou
Transim tran stop=4000u paramset=paramSet1

// Save internal voltages and currents
opt1 options save=allpub
```

Figure 19: Adding parameters to the top level analog testbench

7. Reinvoke the simulator and restart the simulation as explained in step 8 of Tutorial 2. If you did everything right you should see something like in Figure 20.

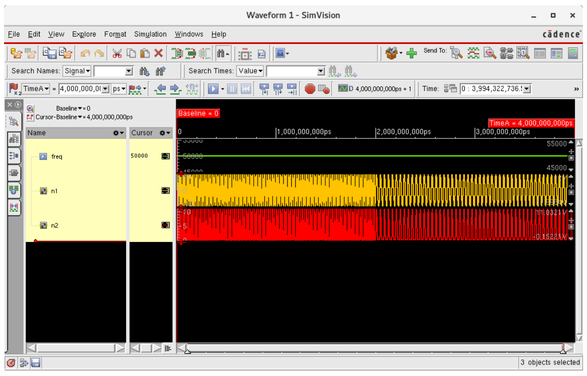


Figure 20: Anode (n1) and cathode (n2) voltage for a changing frequency

8. As shown in Figure 20 you can also plot the value of the parameters you use. But be careful with this. Even though this parameter changes in time (you can see that the frequency of *n1* and *n2* changes at t=2000μs), the plotted value of the parameter *freq* stays constant. If you want to plot the actual value of the *freq* parameter in function of time, you can apply a trick. Add the dummy voltage source in the red box of Figure 21. Reinvoke the simulator. Now you can plot the signal of the *param_freq* node (as shown in Figure 22). This is a representation of the value of the *freq* parameter in function of time.

```
simulator lang = spice

.param freq = 50k

.SUBCKT ANALOG_NETWORK
.option post=1

.include "SmallSignalDiode.spi"

VSS vss 0 0
Vsin n1 vss SIN(0 12 freq)

XD1 n1 n2 SmallSignalDiode
R1 n2 vss 100

Vfreq param_freq vss freq

.ENDS

simulator lang = spectre
```

Figure 21: Adding a dummy source

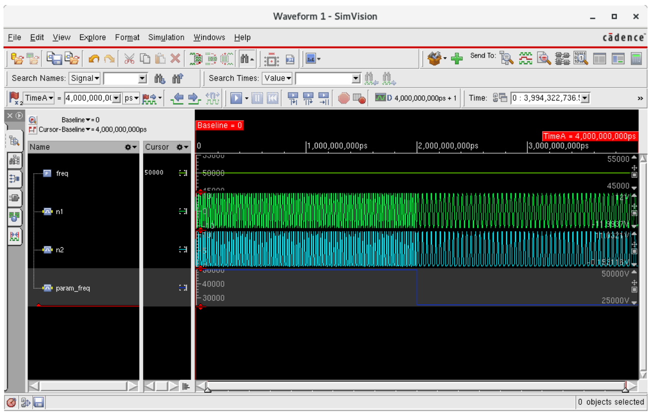


Figure 22: Use a dummy voltage source to see parameter change in time

9. The waveform window can show us the voltages on every node of our circuit. If, for example, you want to calculate the power dissipated in the resistor of 100 Ω, you can use the expression calculator. We will do this in two different ways. First we will use the voltage over the resistor. This is nothing more than the *n2* signal. Select the *n2* signal by clicking at the location of the red box in Figure 23. Then click on the icon of the calculator indicated by the orange box in the upper right corner of Figure 23.

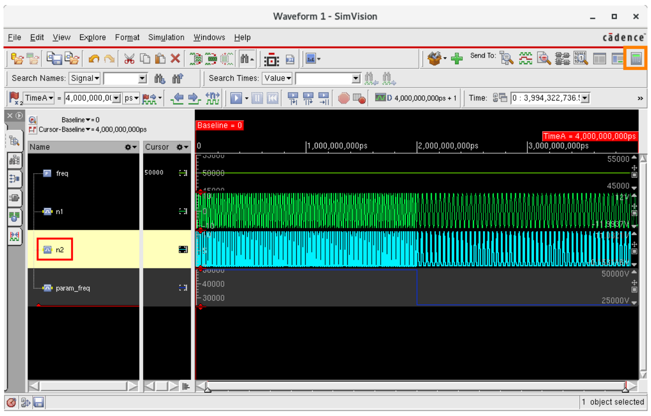
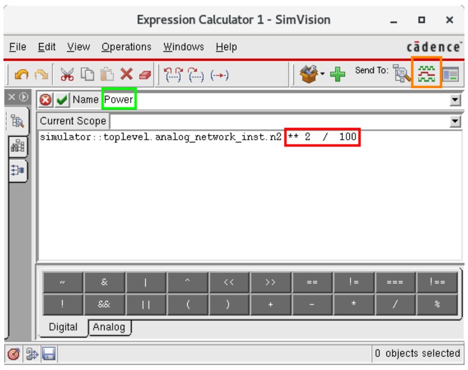


Figure 23: Send waveforms to calculator

10. The expression calculator should open. Now you can use the formula $p = \frac{v^2}{R}$ to calculate the dissipated power. Add what is in the red box of Figure 24. Give your newly calculated waveform a proper name (e.g. "Power" as indicated in the green box), and send it to the waveform window by pressing the tab indicated by the orange box in the upper right corner.


Expression Calculator 1 - SimVision
File Edit View Operations Windows Help
cādence
Send To:
Current Scope
Digital
Analog
0 objects selected

Name: Power

```
simulator::toplevel.analog_network_inst.n2 ** 2 / 100
```

Figure 24: Expression calculator

11. You could also use the current through the resistor to calculate the dissipated power ($p = i^2R$). Before you can do that, you have to add the current trough the sine voltage source (that is the same as the current through the resistor) to the waveform window. Click on the tab indicated by the green box of Figure 17. Add the current waveform to the waveform window by clicking on the signal indicated in the red box of Figure 25. Note that SimVision gives this current signals quite confusing names (like *p_$flow*). Luckily you can rename them in the waveform window by right clicking on them.

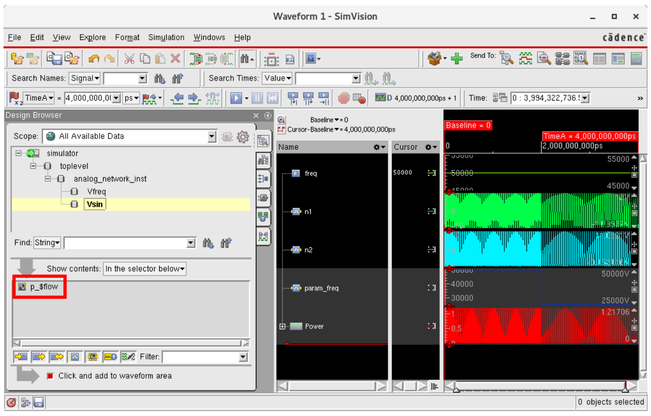


Figure 25: Send current to waveform window

12. Now use this current to calculate the power dissipated in the resistor. It is a good exercise to figure this out on your own. Of course you should get the same results as the power you calculated in step 10. You can use this expression calculator for a lot more than shown in this tutorial. You could for example try to calculate the efficiency as the average power dissipated in the resistor over the average power delivered by the source.

4 Digital Simulations Using SimVision

SimVision also allows you to simulate Verilog code. The tutorial in this section will mainly focus on how to organise your Verilog files, so that they can be simulated and how to use the waveform window for digital signals. Note that this is more a tutorial on using the simulator than on using Verilog. For more information on Verilog, you can check: `http://www.asic-world.com/verilog/veritut.html`

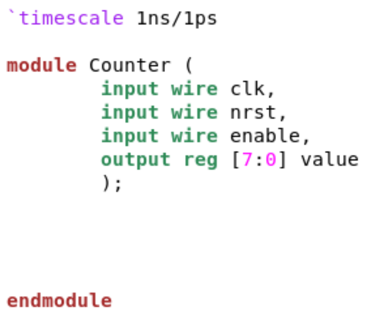
4.1 Tutorial 4: Simple Counter

The purpose of this tutorial is to create a simple 8-bit counter and simulate it using SimVision.

1. Start the computer under CentOs (Linux).

2. Create a new directory (It it best not to use any spaces in the directory name). Copy the content of the *Digital_example_circuit_blank* directory into a new directory.

3. Before we can simulate our counter, we first have to implement it in Verilog code. This is done in a Verilog file using a ".v" filename extension. We will implement our counter in *Counter.v*. Open that file in the text editor. You will find a module where the in- and outputs are already defined, but body is still empty as shown on Figure 26.



```
`timescale 1ns/1ps

module Counter (
        input wire clk,
        input wire nrst,
        input wire enable,
        output reg [7:0] value
        );



endmodule
```

Figure 26: empty counter module

4. Add the following code to the body of the module:

```
always @(posedge clk or negedge nrst) begin
    if (~nrst)
        value <= 0;
    else begin
        if (enable)
            value <= value + 1;
        else
            value <= value;
    end
end
```

   This always-block implements the counter. If the *enable* signal is high, the counter value is incremented at every rising clock edge. If the *enable* signal is low the counter just remembers its current value. If the active low reset signal *nrst* gets low, the counter value is set to zero.

5. Now the counter module can be added to the testbench. Open *Testbench.v* in the text editor and add what is in the red box of Figure 27.

6. Before we start the simulation, we should tell the simulator which Verilog files it should use. Open *verilogFiles.f* in the text editor. This file is empty. Now add the following lines:

```
  ./Counter.v
  ./Testbench.v
```

7. Open a terminal and navigate to the directory you created at the beginning of this tutorial. Enter the following command:

```verilog
`timescale 1ns/1ps

module toplevel ();

        reg clk = 1'b1;
        reg nrst = 1'b1;
        reg enable = 1'b0;

        wire [7:0] value;

        // Clk gen (fclk = 100MHz)
        always
        #5 clk = ~clk;

        // Reset and enable
        initial begin
                #5 nrst = 1'b0;
                #15 nrst = 1'b1;
                #25 enable = 1'b1;
        end

        // Counter instance
        Counter inst_counter (
                .clk    (clk),
                .nrst   (nrst),
                .enable (enable),
                .value  (value)
                );

endmodule
```

Figure 27: Testbench

```
> sh run.sh
```

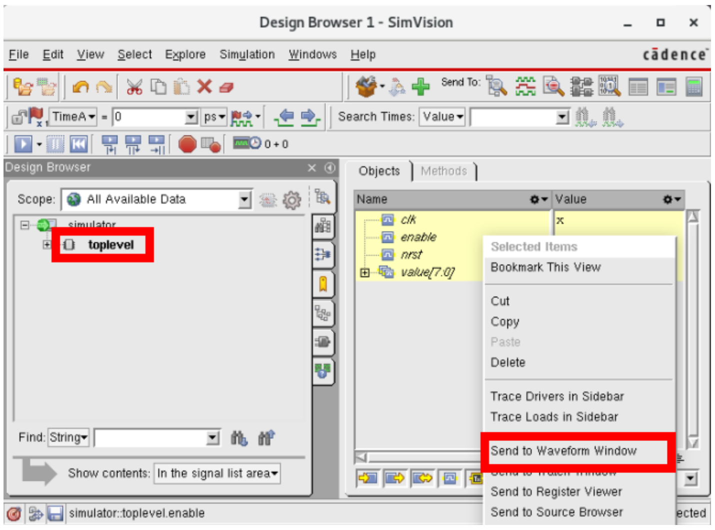


Figure 28: Design browser

8. The design browser window shown if Figure 28 will open. Click on *toplevel* and then select all the signals at the right (using the shift button). Right click on the selected signals and choose "Send to Waveform Window"

9. The waveform window shown if Figure 29 will open. To start the simulation press the play-button. The counter we are simulating, is a very simple circuit. Hence the simulation will be very fast. Press the pause-button next to the play-button to pause the simulation. Now we can take a closer look to the simulation results.

10. Zoom in on the results as shown in Figure 30. Are they as you would expect? SimVision expresses numbers by default as hexadecimal numbers. If you want them expressed in another base right click on the number in the red box in Figure 30. Now click on "Radix/Mnemonic" and select the base you want.

11. If you have adapted the Verilog code and want to rerun the simulation, click on "simulation" and then on "Reinvoke Simulator..." as shown in Figure 31. You will get a message. Click 'Yes'. Press the play-button

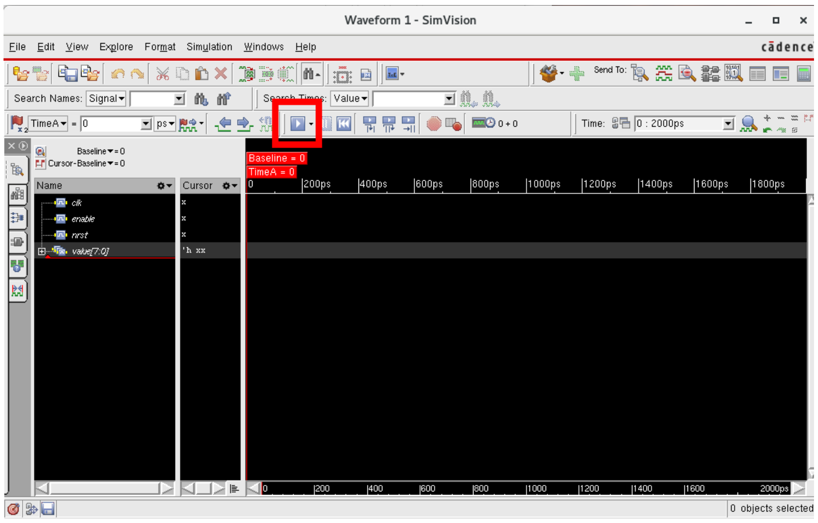


Figure 29: Waveform window

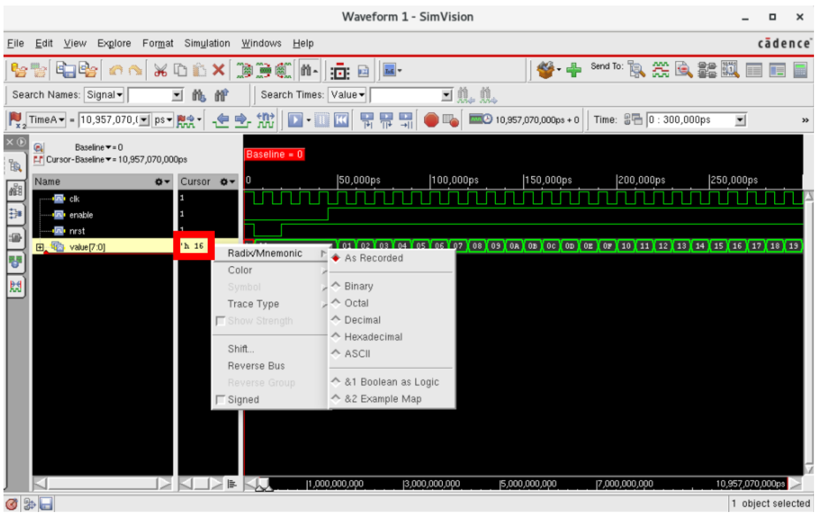


Figure 30: Waveform window: results

to start the simulation again.

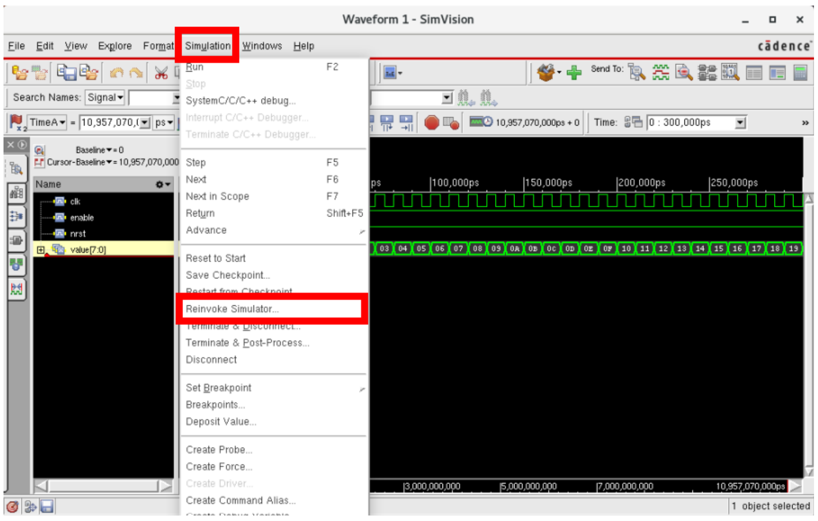


Figure 31: Waveform window: reinvoke simulator

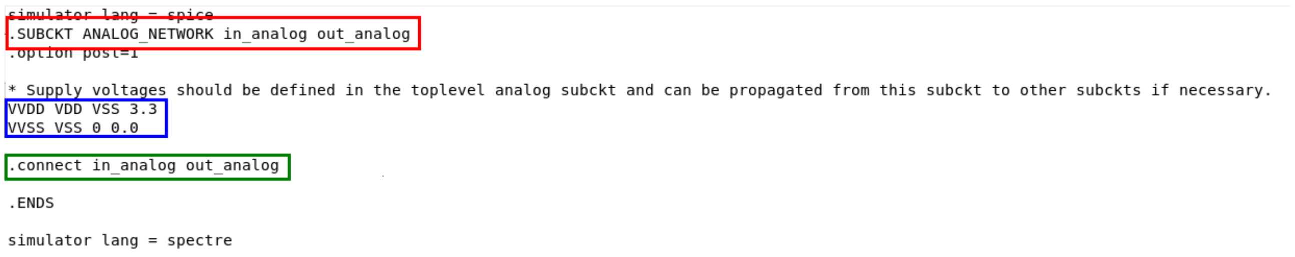
5 Mixed Signal Simulations Using SimVision

The tutorial in this section connects everything you have learnt in the previous tutorials. We will simulate a mixed signal design. The main focus lies on the simulation file structure: How do we connect the Verilog and Spice files so that they can be simulated together?

5.1 Tutorial 5: Add Analog Circuits to Simple Counter

The purpose of this tutorial is to add additional analog circuitry to the simple 8-bit counter.

1. Start the computer under CentOs (Linux).
2. We will work in the same directory we created at the beginning of Tutorial 4.
3. Analog circuits are not implemented in Verilog. For the SWIPT-project we will use Spice files (with a ".spi" filename extension) to describe our analog circuits. Open *ANALOG_NETWORK.spi* . You should see what is in Figure 32.



```
simulator lang = spice
.SUBCKT ANALOG_NETWORK in_analog out_analog
.option post=1

* Supply voltages should be defined in the toplevel analog subckt and can be propagated from this subckt to other subckts if necessary.
VVDD VDD VSS 3.3
VVSS VSS 0 0.0

.connect in_analog out_analog

.ENDS

simulator lang = spectre
```

Figure 32: ANALOG_NETWORK.spi

The line in the red box is the definition of the *ANALOG_NETWORK* subcircuit. It has two terminals *in_analog* and *out_analog*. Note that although these names suggest that the first terminal is an input and the last terminal is an output, Spice does not make this distinction. For Spice these terminals are just nodes in an electrical circuit, without any specific in or out direction. In the blue box are two DC voltage sources defining the VDD and VSS lines. The supply voltage of the Zybo is 3.3V . The line in the green box just connects *in_analog* and *out_analog*. A schematic of the subcircuit is shown in Figure 33. It is in fact nothing more than a wire connecting *in_analog* and *out_analog* and a DC voltage source that is not used in the rest of the circuit.

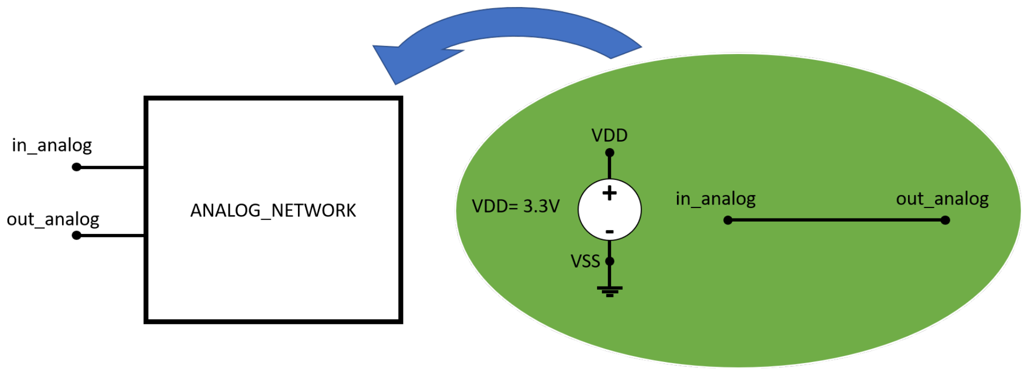


Figure 33: Schematic of ANALOG_NETWORK

4. How do we combine the digital Verilog files and the analog Spice files so that they can be simulated together? Figure 34 gives an overview of the file structure. This figure will be discussed in detail in the following steps.

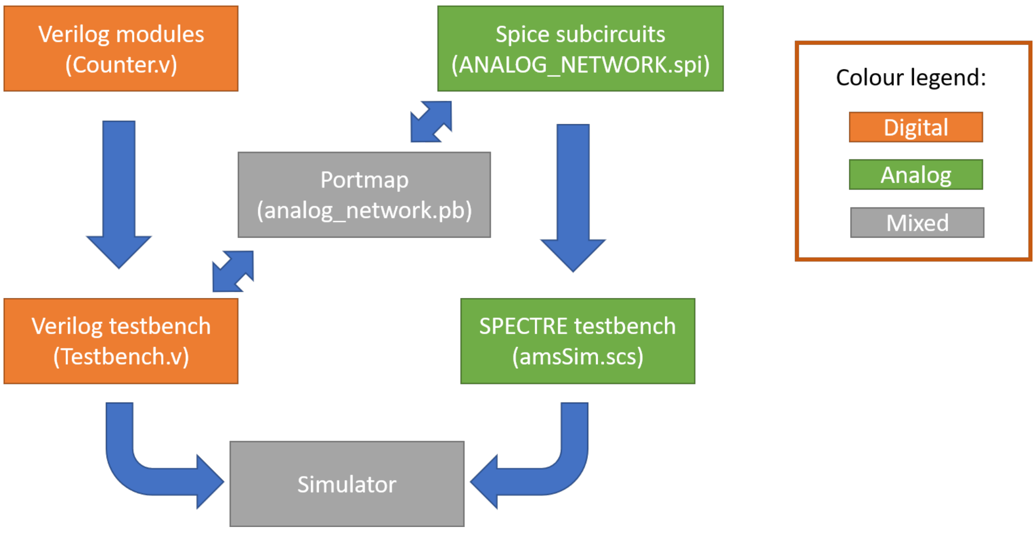
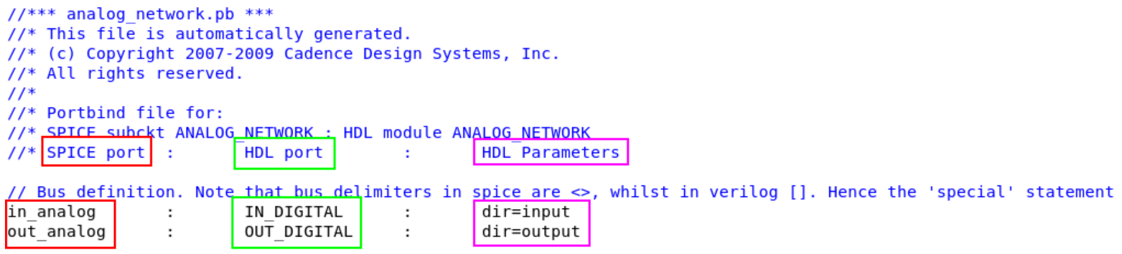


Figure 34: Simulation file structure

5. Open *Testbench.v* and add the following code to the toplevel module:

```
// Input of analog circuit
wire IN_DIGITAL;
// Output of analog circuit
wire OUT_DIGITAL;

// Input of the analog part is the MSB of the counter value
assign IN_DIGITAL = value[7];

// Analog circuit
ANALOG_NETWORK inst_ANALOG_NETWORK (
    .IN_DIGITAL (IN_DIGITAL),
    .OUT_DIGITAL (OUT_DIGITAL)
    );
```

6. Open *analog_network.pb* . This file contains the portmap, which defines the interface between the analog en digital circuitry. Portmaps are saved in files with a ".pb" filename extension. It connects the analog ports and the digital ports to each other. The content of *analog_network.pb* is shown in Figure 35. You don't have to change anything for now. But it is important to understand its structure, so that you can adapt it correctly when you change the ports of your design. In Figure 35, the analog ports are indicated in the red boxes, the digital ports in the green boxes and the direction (in- or output) is indicated in the purple boxes.



```
//*** analog_network.pb ***
//* This file is automatically generated.
//* (c) Copyright 2007-2009 Cadence Design Systems, Inc.
//* All rights reserved.
//*
//* Portbind file for:
//* SPICE subckt ANALOG_NETWORK : HDL module ANALOG_NETWORK
//* SPICE port  :       HDL port        :       HDL Parameters

// Bus definition. Note that bus delimiters in spice are <>, whilst in verilog []. Hence the 'special' statement
in_analog       :       IN_DIGITAL      :       dir=input
out_analog      :       OUT_DIGITAL     :       dir=output
```

Figure 35: Portmap

7. Open *amsSim.scs*. This is the top level analog testbench. Uncomment the amsd block in the red box of Figure 36. Be careful when you try to simulate your own circuits. You should adapt the file and subcircuit names in the blue boxes to your new Spice and portmap files. Now the simulation stops after 2000μs . You can adapt this by changing the stop=... in the green box.

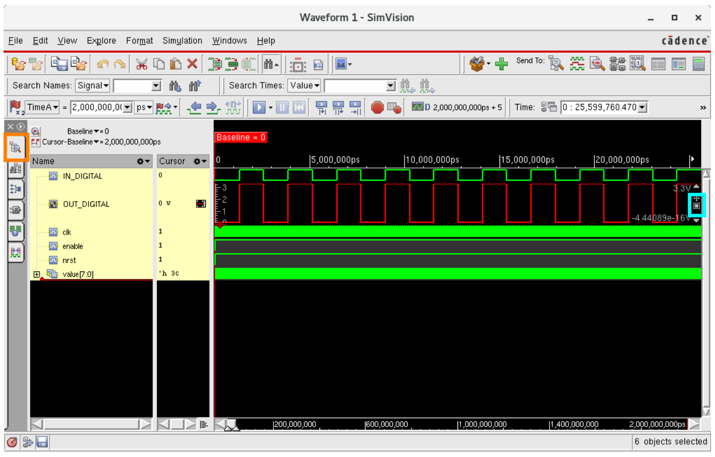
```
include "./ANALOG_NETWORK.spi" // analog subcircuit definitions

// If device models are used in the subcircuit, these have to be included as well of course.
//include "/users/micas/ruytterh/Documents/Research/Coolflux/Analog/toplevel.scs" section=top_tt

// Some options can also be set through a analog control file.
// Info here...
//include "/users/micas/ruytterh/Documents/Research/Coolflux/Implementation/95_CD/acf.scs" // analog
control file

//amsd block

//amsd{
//
//      portmap subckt=ANALOG_NETWORK file="./analog_network.pb"
//      config cell=ANALOG_NETWORK use=spice
//      ie vsup=3.3
//
//}

// DC simulation statement (required to find initial condition at startup of transient)
DCsim dc

// Tran simulation statement. Simulation will end when stop time is reached (so this should be longer
than the required time to run the entire digital simulation or not if you only want to simulate a small
part)
Transim tran stop=2000u

// Save internal voltages and currents
opt1 options save=allpub
```

Figure 36: Spectre testbench

8. Now you are ready to start the mixed signal (digital and analog) simulation. Start the simulator just like in Tutorial 4: Open the terminal, navigate to the directory you are currently working in and enter the following command:

```
> sh run.sh
```

9. Send the signals of the toplevel module to the waveform window en start the simulator just like you did in Tutorial 4 (Figure 28 and 29). Zoom in until you properly see the block wave of *IN_DIGITAL*. It is possible that the block wave of *OUT_DIGITAL* is barely visible now. You can solve this by clicking on the square at the right indicated by the blue box in Figure 37.



Name	Cursor
IN_DIGITAL	0
OUT_DIGITAL	0 V
clk	1
enable	1
nrst	1
value[7:0]	'h 3c

Figure 37: Waveform window mixed signal simulation

10. The *IN_DIGITAL* signal in Figure 37 is a digital signal. It can only have a the logical values used by Verilog ('0', '1', 'z' , 'x',...) while the *OUT_DIGITAL* signal is an analog signal that continuously varies between 0 and 3.3 V (even though it looks quite digital in this example). To plot the analog variant of *IN_DIGITAL*, click on the tab indicated by the orange box at the left shown in Figure 37. Now you can browse trough the different simulation results. Click on the *in_analog* signal as shown in Figure 38. This is the analog representation of *IN_DIGITAL* (remember that we connected these two signals in the *analog_network.pb* - file?).

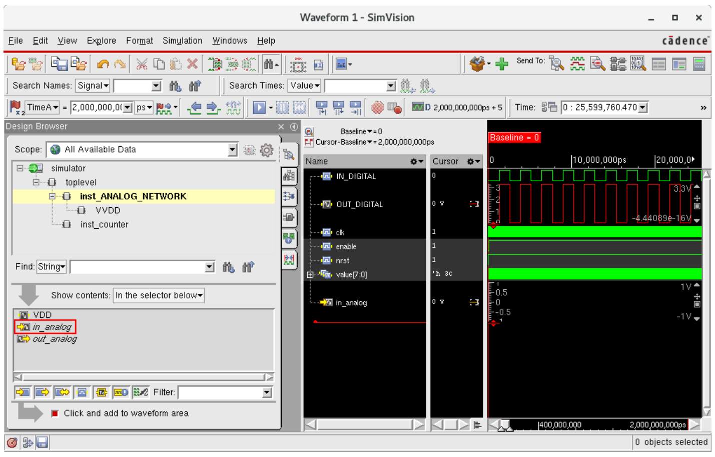


Figure 38: Browse trough the different waveforms

11. Reinvoke the simulator as you did in Tutorial 4 (Figure 31). And start the simulation again by pressing the play button. Now you can see the analog input signal. You will see that the output signal is exactly the same as the input signal. This is no surprise because our analog circuit was nothing more than a wire connecting in- and output. Now you are ready to implement your own (more complex) analog an digital circuitry.

Tip1:
In stead of putting your analog module directly into the testbench as shown in step 5 of Tutorial 5, you can also put it first in a seperate Verilog module. In a second step you can add this Verilog module to the testbench, just like you did with the counter.

Tip2:
Verilog-AMS is a variant of Verilog that can be used to model mixed signal circuits like ADCs and DACs. The syntax is very similar to normal Verilog. You can also define modules with in- and outputs, and put these modules in your testbench. If you want to do some 'processing' of outputs of the analog part, before these analog signals hit the digital part, you could use Verilog-AMS. Always save your Verilog-AMS files wit a ".ams" filename extension, and don't forget to add these files to the *verilogFiles.f*-file.