Tiago Oliveira Weber

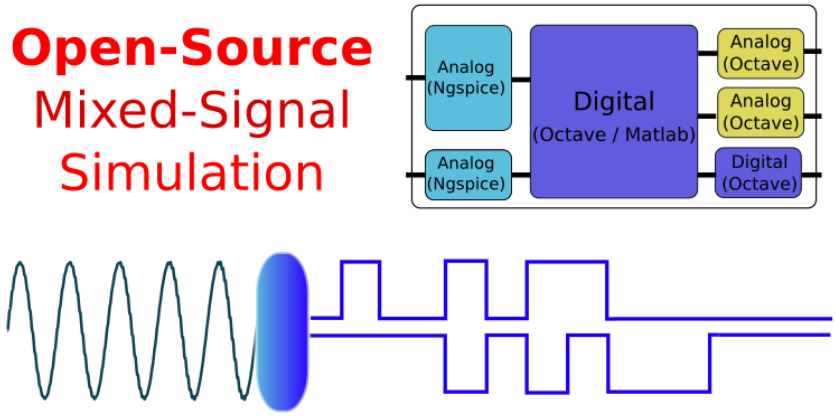
Tiago

April 8, 2017

Back to home

Open-Source Mixed-Signal Simulation: How to Describe Blocks in Octave/Matlab and Interact them with Ngspice

by Tiago Oliveira Weber *<2016-08-04 Thu>*



Mixed-signal simulation is the type of simulation in which analog and digital signals coexist in the same circuit. There are many types of systems that can be best analyzed through mixed-signal simulation. They are paramount for circuit simulation including Analog-Digital and Digital-Analog Converters (ADCs and DACs). For instance, to simulate how a microcontroller

interact with peripherals through an ADC, one could benefit from this type of simulation.

In the seek for a open-source solution for mixed-signal simulation, I developed an octave/matlab package that allows Octave to control the simulation through ngspice shared api.

How I Learned to Stop Worrying and Love Octave and Ngspice for Circuit Simulation

We usually simulate analog circuits through SPICE circuit simulators. On the other hand, we usually use a Hardware Description Language (HDL) such as Verilog or VHDL using specific simulators for the digital section. If simulation time were not an issue, we could consider all signals as being analog and simulate the whole circuit as a SPICE netlist. However, due to the level of abstraction that digital circuits allow us to operate in, this would be an enormous waste of time.

Sometimes we are not interested in modelling a block with a high-level language, but only interested in analyzing the output of the circuit. Usually Matlab or Octave are used to perform **post-simulation analysis**. In this approach, the circuit simulator acts independently and its output (waveforms and measurements) are used as input to a script in Matlab or Octave.

However, we may want to model a block (e.g.: counter, ADC, DAC, sensor, receiver, transceiver, ...) in a more abstract way. On those cases, we can use **Verilog-A** to describe the model and use a commercial tool (such as spectre or hspice) to perform the simulation. **QUCS** and **Ngspice** are open-source tools that provide a way to add Verilog-A models to the simulation, but not without some pain as you go through the compilation procedure of each block. Even using commercial tools the debugging of what went wrong on a Verilog-A model is not easy.

For simple tests and small projects, I rather use a **well-known high-level language**, such as **Matlab** or **Octave**. Easy prototyping is one the reasons that made Matlab one of the greatest tools for scientists. The codes are easy to write, to understand and to debug.

Therefore, in the seek of a simple solution that could simulate mixed-signal blocks in matlab or octave and ngspice, I propose an octave package to:

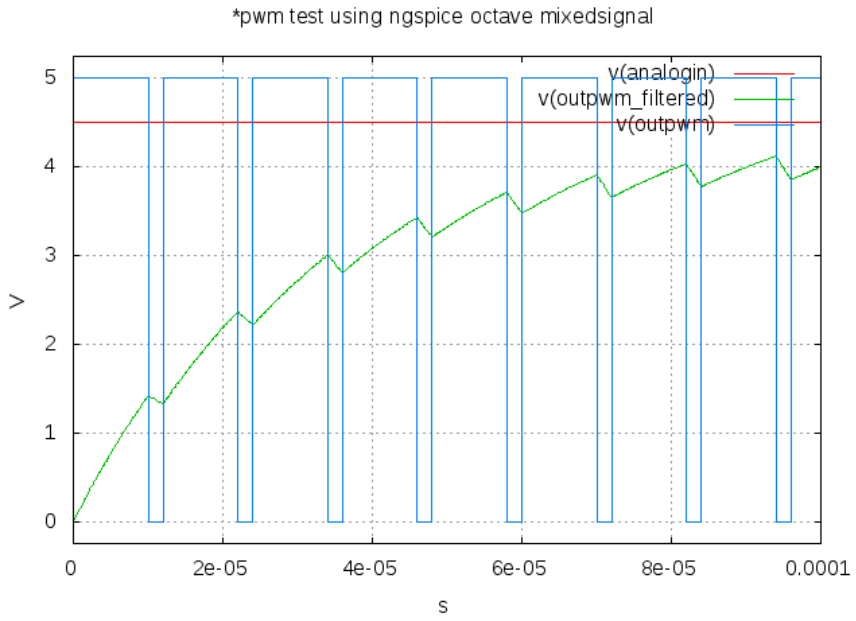
- allow the use of Matlab or Octave to describe the digital blocks;
- allow the simulation of mixed-signal circuits using open-source tools;

Of course, one using octave or matlab for modelling digital blocks needs to keep in mind this is not a synthesys-oriented approach. It is useful for behavioral modelling only.

```
*PWM Test using Ngspice Octave Mixedsignal
.param clk_freq = 1e6
.param clk_per = '1/clk_freq'
.param cap_value = 1e-6
V1 vdd  0 5
Xpwm analogin outpwm clk pwm
Rout outpwm outpwm_filtered '(clk_per*10*3)/cap_value'
Cout outpwm_filtered 0 'cap_value'
Rdummy outpwm_filtered 0 10meg
Vclock clk 0 PULSE(0 5 0 10n 10n 'clk_per/2' 'clk_per')
*Vin analogin 0 SIN(2.5 1 1k)
Vin analogin 0 PULSE(2.5 4.5 0 10n 10n '400*clk_per/2' '400*clk_per')
.subckt pwm in1 out1 clock edge=1 clockth=2.5 octave=1 vhigh=5
.ends

.tran 1e-6 0.1e-3
.print tran v(outpwm_filtered) v(analogin)
.end

.control
run
set gnuplot_terminal=png
gnuplot $file v(analogin) v(outpwm_filtered) v(outpwm)
.endc
```




```
function [pinvalue] = pwm(pinvalue,currenttime,vhigh,isfirst);
% temporarily I will pass arguments directly, this will be in a structure later

    max_cycles = 10;
    analog_in = pinvalue(1);
    global counter
    if (isfirst)
counter = 0;
    elseif (counter > max_cycles)
counter = 0;
    else
counter = counter + 1;
    end
    idealdutycycle = analog_in/vhigh;
    high_cycles = round(max_cycles*idealdutycycle+0.25);
    if (counter <= high_cycles)
digital_out = vhigh;
    else
digital_out = 0;
    end
    pinvalue(2) = digital_out;
```

```
endfunction
```