XMODEL Primitives

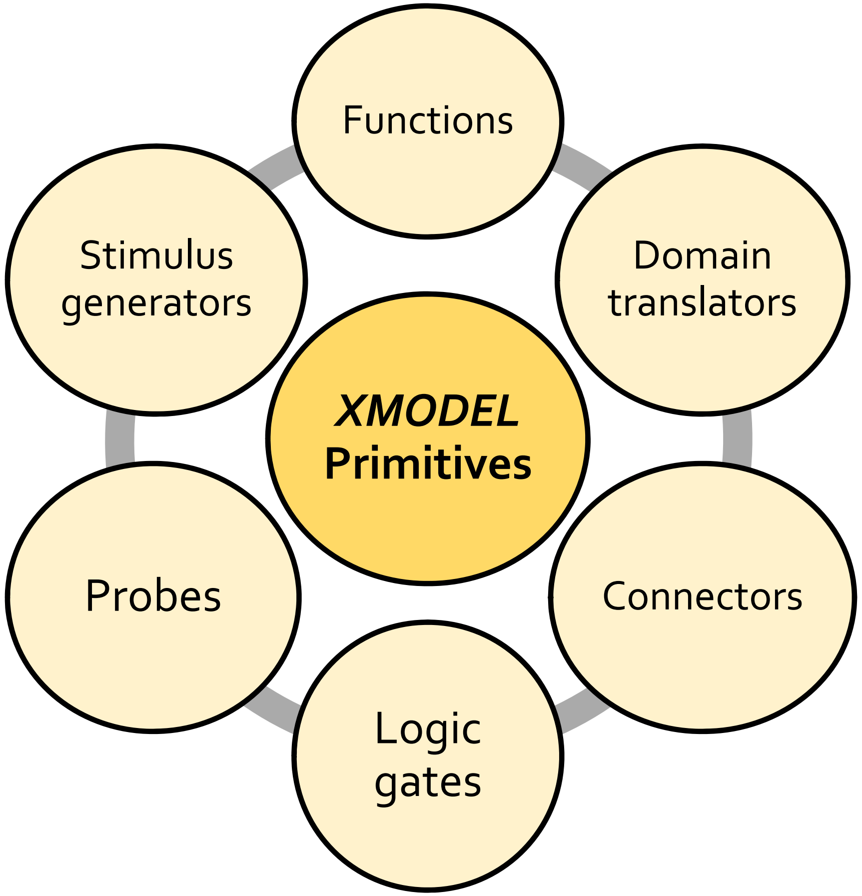
Scientific Analog, Inc.

July 2017

Overview

- *XMODEL* provides an extensive library of primitives that help you compose analog models and testbenches

- This lecture will cover their essentials



XMODEL Primitives List

- **Functions**
  - add, multiply, deriv, integ, integ_mod, filter, select, limit, power, pwl_func, poly_func, transition, sample, compare, dac, adc, …
- **Circuits**
  - resistor, capacitor, inductor, switch, diode, nmosfet, pmosfet, vsource, isource, vprobe, iprobe, vcvs, vccs, ccvs, cccs, …
- **Logic gates**
  - buf_xbit, inv_xbit, nand_xbit, nor_xbit, and_xbit, or_xbit, xor_xbit, xnor_xbit, mux_xbit, dff_xbit, …
- **Domain translators**
  - clk_to_freq, clk_to_phase, clk_to_period, clk_to_duty, clk_to_delay, freq_to_clk, phase_to_clk, period_to_clk, …

XMODEL Primitives List (2)

- **Stimulus generators**
  - dc_gen, noise_gen, step_gen, exp_gen, sin_gen, pwl_gen, clk_gen, pulse_gen, pat_gen, prbs_gen, …
- **Probes**
  - probe_xbit, probe_xreal, probe_bit, probe_real, probe_freq, probe_phase, probe_period, probe_duty, probe_delay, dump, …
- **Connectors**
  - xbit_to_bit, bit_to_xbit, xreal_to_real, real_to_xreal, xreal_to_xbit, xreal_to_bit, real_to_xbit, real_to_bit, xbit_to_xreal, bit_to_xreal, xbit_to_real, bit_to_real, …

Accessing On-line Documentation

- Use '-h' command for on-line help:

```
$ xmodel -h
…
list of help topics:
        function       Functions
        gate           Logic gates
        circuit        Circuit elements
        stim           Stimulus generators
        meas           Probes
        vdt            Domain translators
        connect        Connectors
```

- Offline documentation is located at **$XMODEL_HOME/doc/XMODEL_Reference_Manual.pdf**

Accessing On-line Documentation (2)

- Use '-h TOPIC' to get a list of primitives of each category:

```
$ xmodel -h stim
==============================
TOPIC stim
==============================
The XMODEL stimulus generator primitives provide means to
generate various stimulus waveforms both in analog and
digital format.

list of stimulus generator primitives:
       clk_gen        A digital clock generator.
       dc_gen         Analog DC generator
       exp_gen        Analog exponential signal generator
       noise_gen      Noise generator
       …
```

Accessing On-line Documentation (3)

- Use '`-h PRIMITIVE`' to get the documentation on each primitive:

```
$ xmodel -h sin_gen
==============================
PRIMITIVE sin_gen
==============================
Analog sinusoid generator

The 'sin_gen' primitive generates a sinusoidal signal that
can optionally be exponentially decaying or frequency/
amplitude-modulated.

The generated stimulus waveform V(t) is defined as follows:
    for t < delay:
        V(t) = offset + amp*AM_offset*sin(init_phase)
…
```

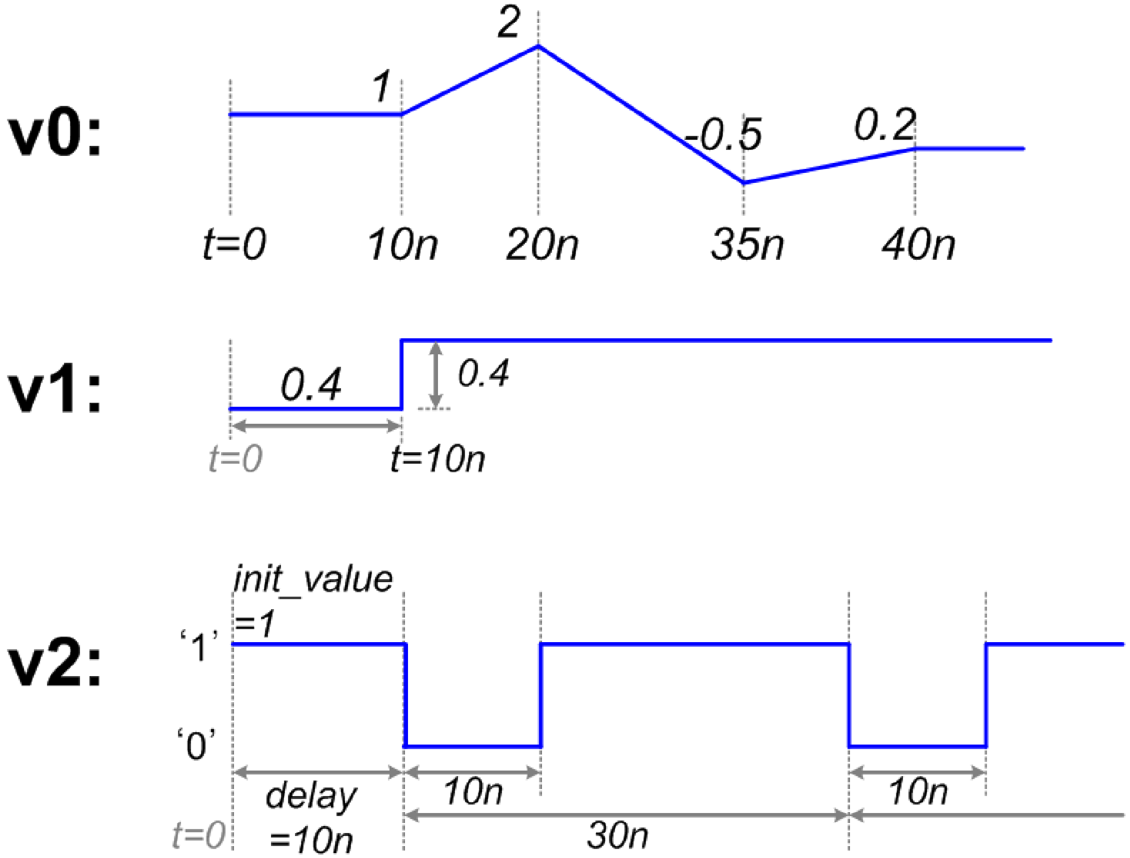
Stimulus and Probe Primitives

- Both stimulus and probe primitives are useful when composing testbenches
  - Stimulus primitives generate input stimuli
  - And probe primitives record results

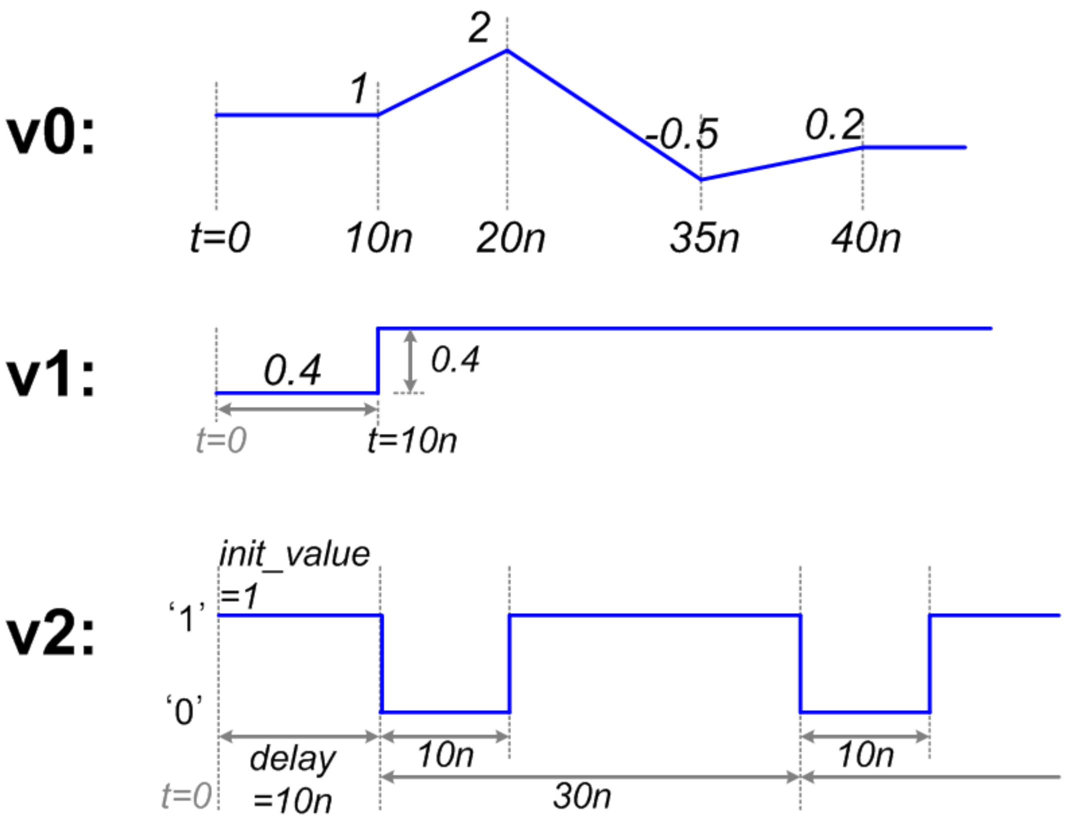
- The available stimulus primitives are:
  - Analog output (xreal): dc_gen, exp_gen, noise_gen, pwl_gen, sin_gen, step_gen
  - Digital output (xbit): clk_gen, pat_gen, prbs_gen, pulse_gen

Exercise #1: Generate Waveforms

- Complete a testbench in **prims/tb_stim_ex/tb_stim.sv** that generates the following 3 waveforms



Exercise #1: Hints



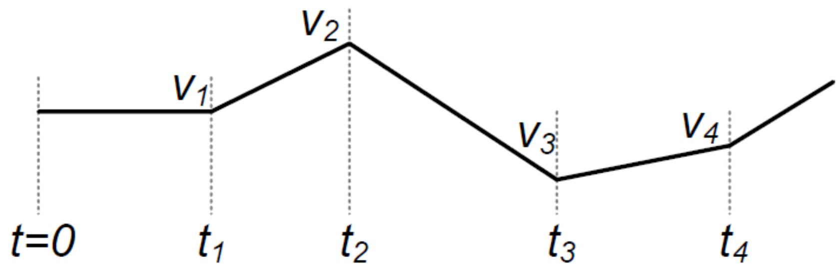
Use:

- ***pwl_gen*** primitive (piecewise linear)
- ***step_gen*** primitive
- ***pulse_gen*** primitive

- Refer to the online/offline documentations for the full details for each primitive

pwl_gen Primitive

```
pwl_gen #(.data('{t1, v1, t2, v2, ...})) my_gen(signal);
```



- I/O description

Name	I/O	Type	Description
out	output	xreal	signal output

- Parameters

Name	Type	Default	Description
data	real array	'{0.0, 0.0, 1.0e-9, 1.0}	PWL data series (time, value pairs)
period	real	-1.0	A repetition period in seonds

probe_{xreal,xbit,real,bit} Primitives

- Record the waveform of the corresponding-typed signal into a file in a JEZ or FSDB format
- I/O description:

Name	I/O	Type	Description
in	input		Signal to save

- Parameters:

Name	Type	Default	Description
filename	string	"xmodel.jez"	Output filename
start	real	0.0	Absolute time to start the recording
stop	real	-1.0	Absolute time to stop the recording
abstol	real	1e-4	Absolute tolerance
reltol	real	1e-2	Relative tolerance
format	string	"jezbinary"	Format version

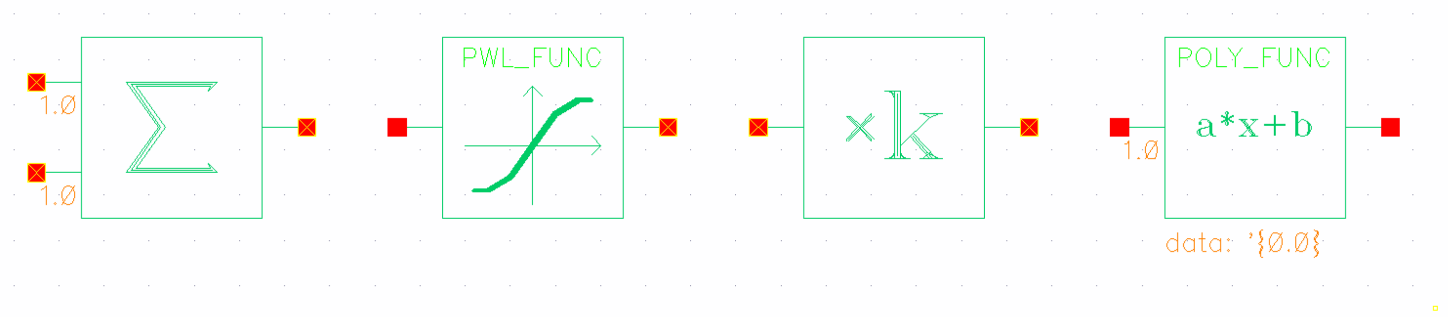
Exercise #1: Answer

- Located in **prims/tb_stim_ex/answer/tb_stim.sv**

```
pwl_gen    #(.data('{10e-9,1,20e-9,2,35e-9,-0.5,40e-9,0.2}))
           v0_gen(pwl_signal);

step_gen   #(.init_value(0.4),.change(0.4),.delay(10e-9))
           v1_gen(step_signal);

pulse_gen #(.init_value(1),.delay(10e-9),.width(10e-9),
             .period(30e-9))
           v2_gen(pulse_signal);
```

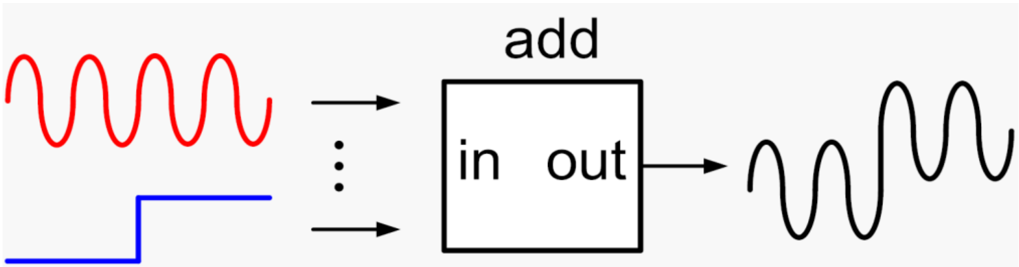
Math Function Primitives

- Math Functions primitives perform mathematical or logical operations on analog signals
  - e.g. add, scale, multiply, deriv, integ, integ_mod, poly_func, pwl_func, limit, power, select



add Primitive

- Computes a weighted sum of multiple xreal-typed signals



- I/O description:

Name	I/O	Type	Description
out	output	xreal	output signal
in	input	xreal array	input signal array

- Parameters:

Name	Type	Default	Description
num_in	integer	2	size of input array
scale	real array	`{1.0,1.0}	weighting factors

add Primitive (2)

- Usage examples:
  - out1 = a – b +2*c

```
xreal   a, b, c, out1;
add     #(.num_in(3), .scale('{1.0, -1.0, 2.0}))
        I1 (.in({a, b, c}), .out(out));
```

  - out2 = (in[0]+in[1]+in[2]+in[3]) / 4.0;

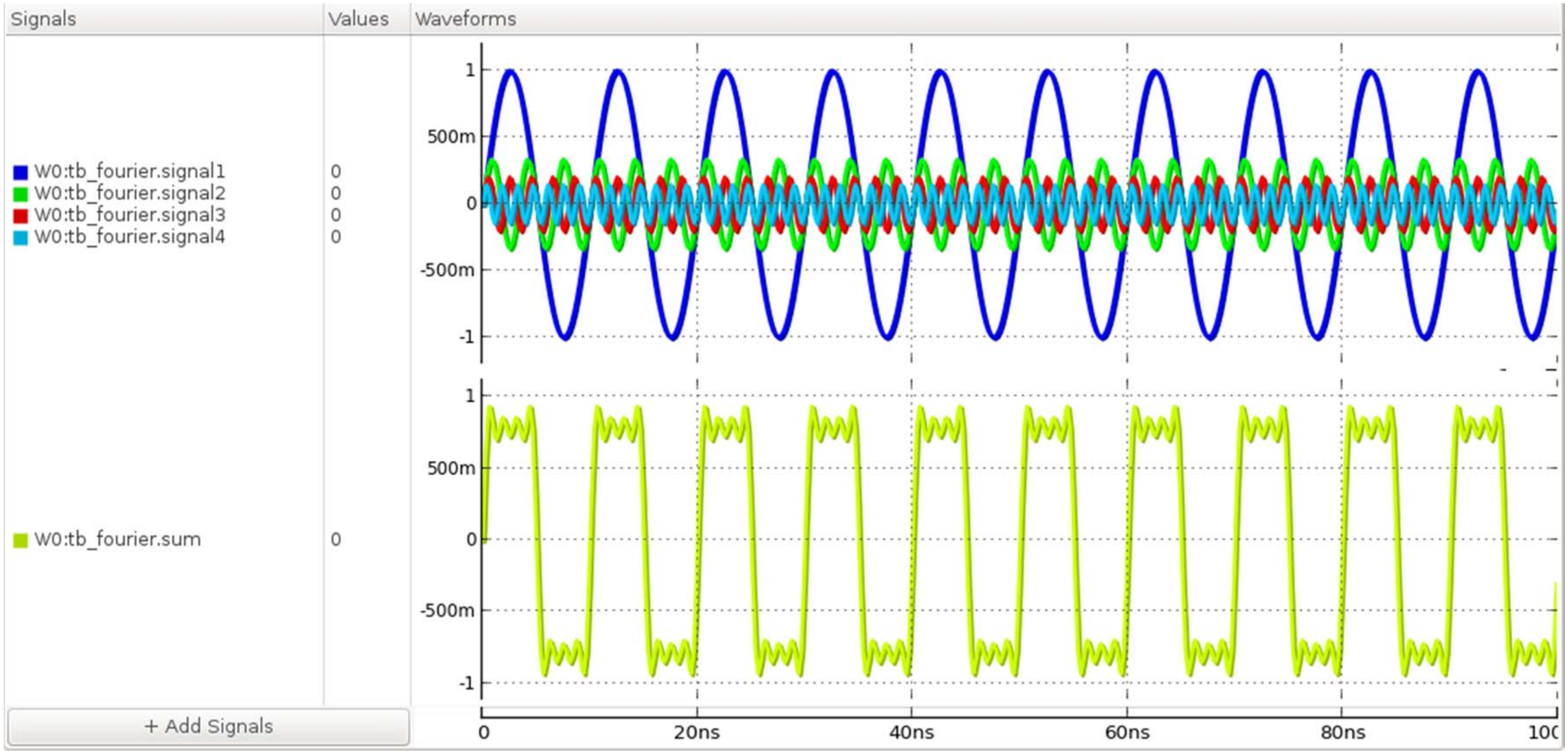
```
xreal   out2;
xreal   [3:0] in;
add     #(.num_in(4), .scale('{0.25, 0.25, 0.25, 0.25}))
        I2 (.in(in), .out(out));
```

Exercise #2

- Compose a testbench that generates a 100-MHz square-wave signal by adding the following 4 harmonic sinusoidal signals:
  - Signal #1: 100MHz, 1-V amplitude sinusoid
  - Signal #2: 300MHz, 1/3-V amplitude sinusoid
  - Signal #3: 500MHz, 1/5-V amplitude sinusoid
  - Signal #4: 700MHz, 1/7-V amplitude sinusoid
  - Start with the skeleton **prims/tb_fourier/tb_fourier.sv**

Answer #2

- Solution located in **prims/tb_fourier/answer/ tb_fourier.sv**
- Simulation waveform (check out the event markers!):



XMODEL Waveform Recording

- By inserting the following lines in the testbench, you can record signal waveforms without having to individually place the *probe* primitives

```
initial begin
        $xmodel_dumpfile();
        $xmodel_dumpvars();
end
```

$xmodel_dumpfile()

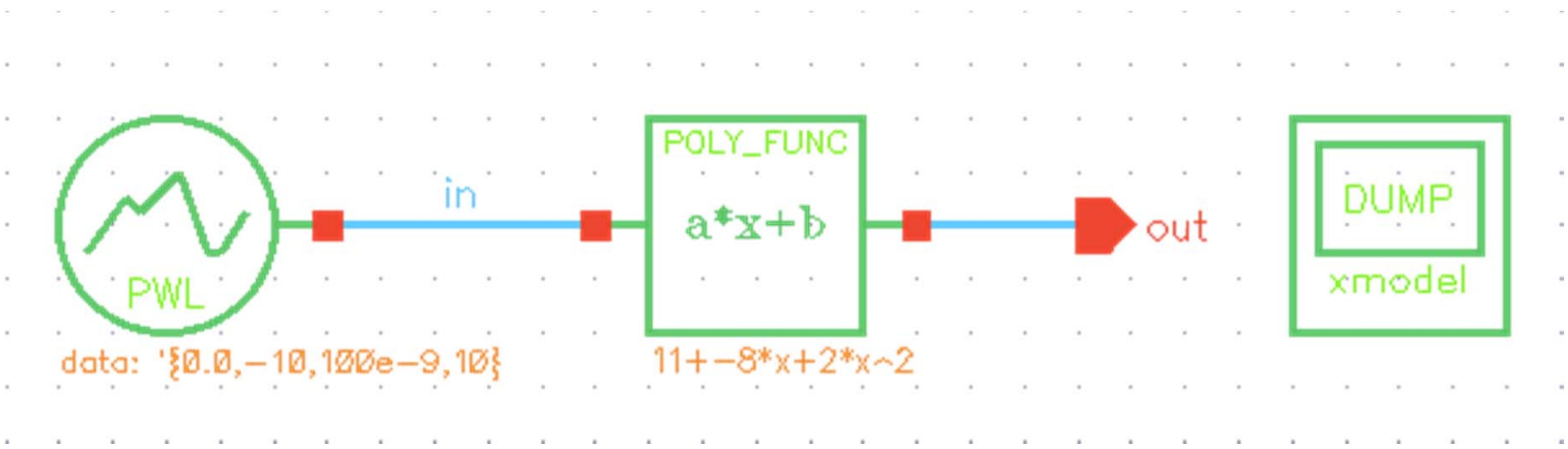
- Defines the name and format of the dump file
- **Usage: $xmodel_dumpfile(*filename*, [*version*])**
  - *filename* : name of the dump file; its extension defines the file format (e.g. ".jez" for JEZ and ".fsdb" for FSDB format)
  - *version* : file format version; currently used only for JEZ format files (e.g. "jezbinary" for binary and "jezascii" for ASCII format)
  - [...] denotes optional arguments
- **Examples**
  - **$xmodel_dumpfile("xmodel.jez", "jezascii");**
  - **$xmodel_dumpfile("xmodel.fsdb");**

$xmodel_dumpvars()

- Defines the variables to be monitored and dumped
- Usage: **$xmodel_dumpvars([*option spec*]*, [*module or variable*]*)**
- Examples:
  - **$xmodel_dumpvars();**
    : dumps all the variables in the current scope and below
  - **$xmodel_dumpvars("level=1", module1);**
    : dumps only the variables in module1
  - **$xmodel_dumpvars("start=10e-9:stop=200e-9", var1, var2, var3);**
    : dumps var1, var2, var3 from 10ns to 200ns

Exercise #3

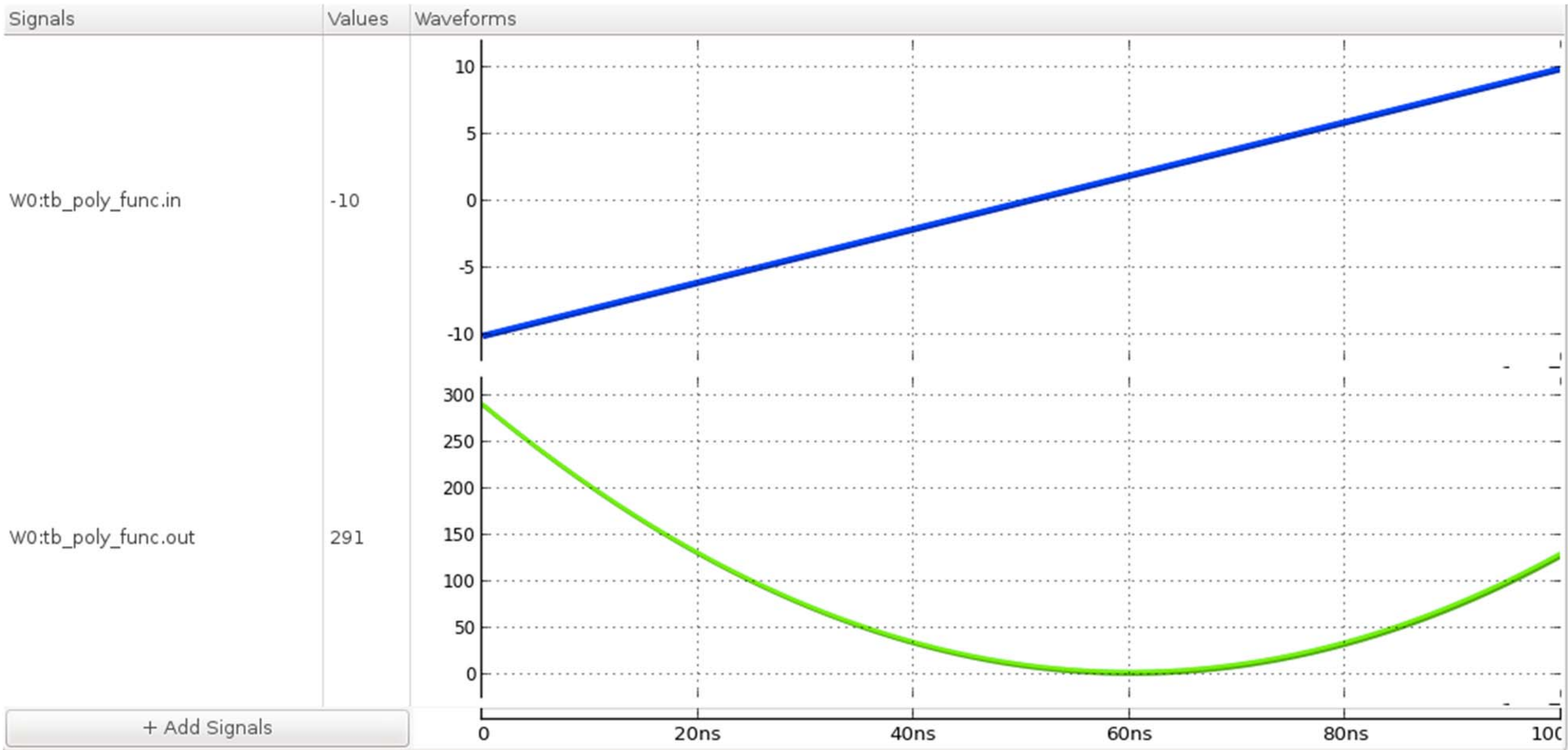
- What is the expected result of the following testbench: **prims/tb_poly_func**?



poly_func Primitive

- Computes the polynomial expression of one or more input signals
- The **data** array parameter defines the coefficients
- Example 1: when **num_in** = 1
  - out = data[0] + data[1]*in + data[2]*in*in + ...
  - Note: data[0] is the first element of the data array
- Example 2: when **num_in** =2
  - out = data[0] + data[1]*in[1] + data[2]*in[0]
    + data[3]*in[1]*in[1] + data[4]*in[1]*in[0]
    + data[5]*in[0]*in[0] + data[6]*in[1]*in[1]*in[1]
    + data[7]*in[1]*in[1]*in[0] + ...

Answer #3: Simulation Waveform

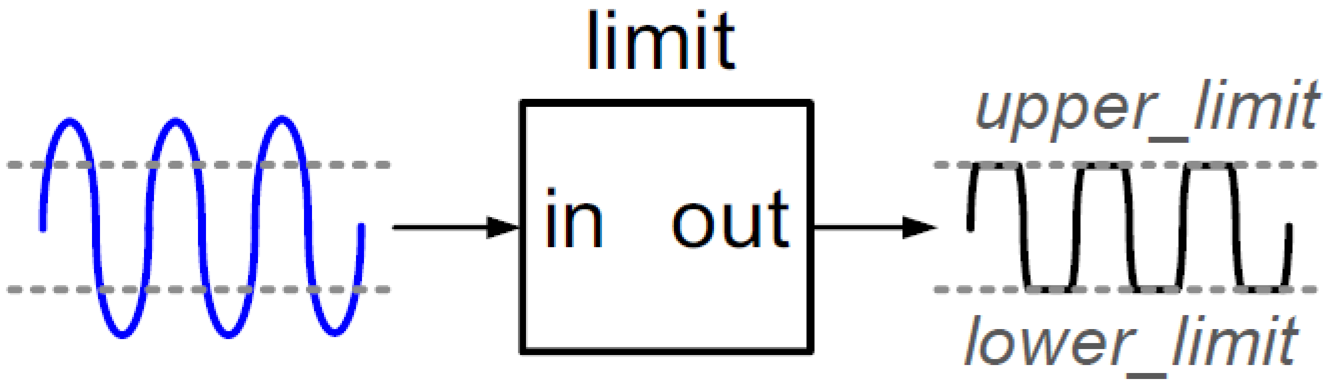


Exercise #4

- Generate a 5-V amplitude sinusoid and clip its level at 3V maximum and -2V minimum
- Use **prims/tb_limit/tb_limit.sv** as a skeleton

limit Primitive

- Limit the input signal to a specified range
  - [*lower_limit*, *upper_limit*]



- Parameters:

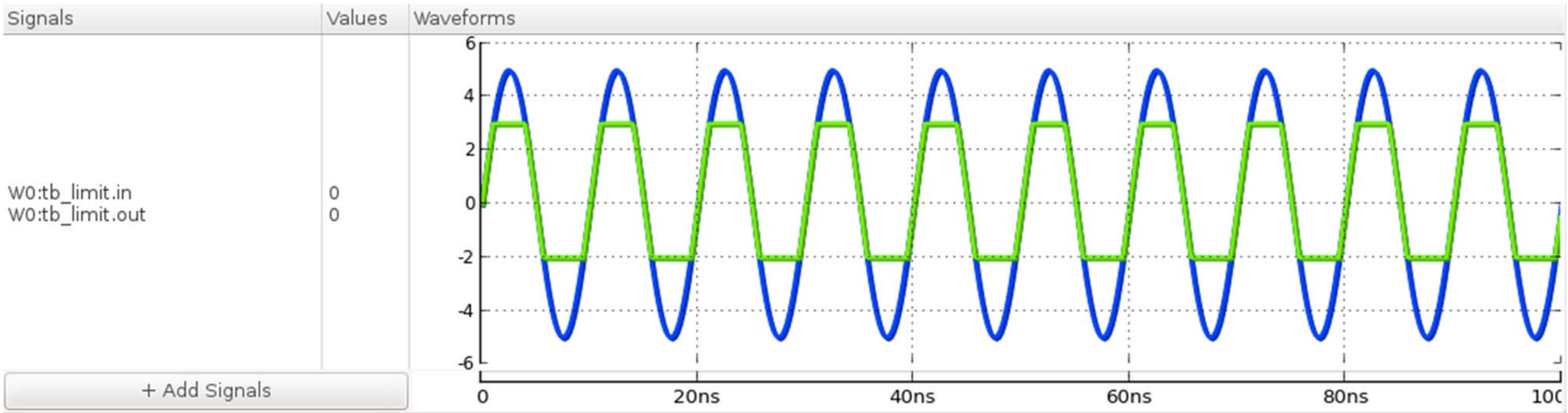
Name	Type	Default	Description
lower_limit	real	0	lower bound
upper_limit	real	1.0	upper bound

Answer #4

- Located in **prims/tb_limit/answer/tb_limit.sv**

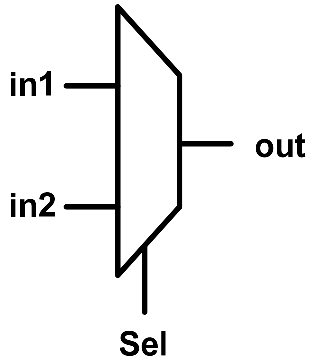
```
xreal       in_signal, out_signal;

sin_gen     #(.freq(100e6), .amp(5.0))
            inst_sin (in_signal);
limit       #(.upper_limit(3), .lower_limit(-2))
            inst_limit (.out(out_signal), .in(in_signal));
```

- Simulated waveforms:



Other Math Primitives

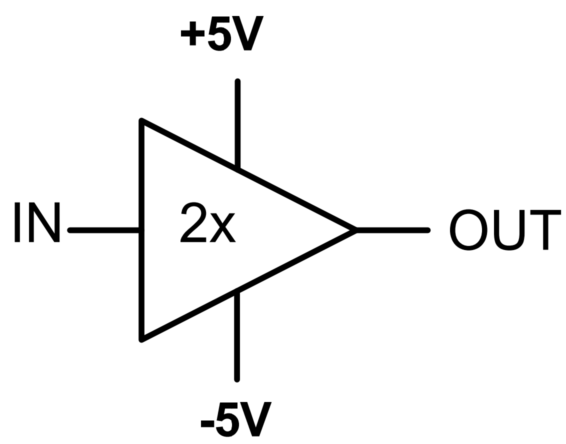
- ***scale***: scale a signal by a constant factor
  - $y = cx$
- ***power***: raise to an m-th power
  - $y = x^m$
- ***multiply***: multiply analog signals
  - $y = x_1 x_2 x_3 \ldots$
- ***select***: select one among multiple analog inputs
  - An analog multiplexer



Exercises with Function Primitives

- Prob #1. Compute $y = (2x+3)^2$
  - Complete the skeleton in **prims/prob1/prob1.sv**

- Prob #2. model an amplifier with a gain of 2 and its output limited to [-5, +5] range
  - Complete the skeleton in **prims/prob2/prob2.sv**



Answers: Prob #1 & #2

- Prob #1: located in **prims/prob1/answer/prob1.sv**

```
pwl_func #(.data('{-2.5, -5, 2.5, 5})) gen_y(.in(x), .out(y));
```

  - You can also use a combination of *scale*, *add*, and *power* primitives

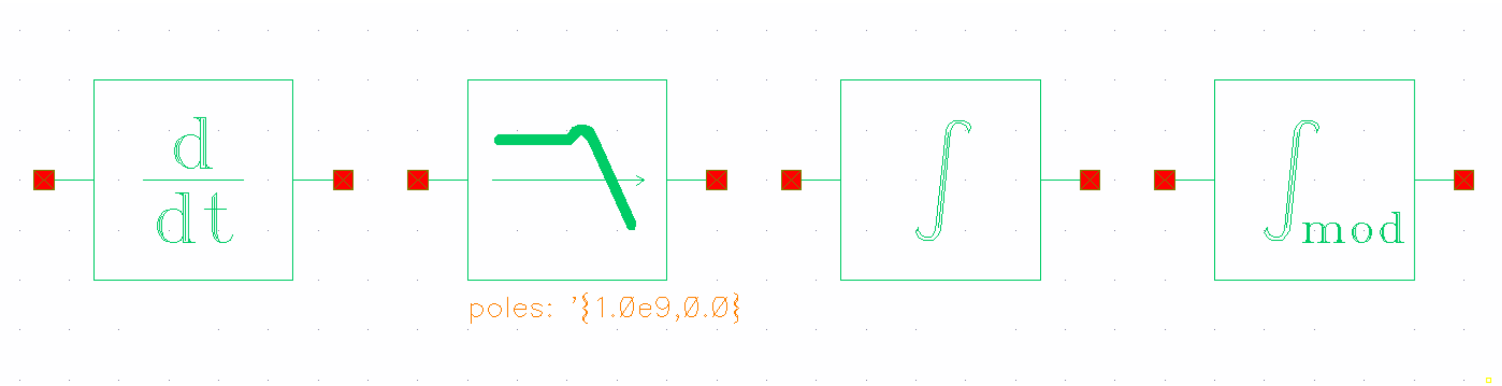
- Prob #2: located in **prims/prob2/answer/prob2.sv**

```
poly_func #(.data('{9, 12, 4})) pf(.in(x), .out(y));
```

  - You can also use a combination of *scale* and *limit* primitives

Modeling Linear Systems

- Available primitives:
  - deriv, filter, filter_var, integ, integ_mod



- A ***filter*** primitive can model any LTI systems
  - Others are special cases of the *filter* primitive
  - e.g. *deriv*: H(s) = s, *integ*: H(s) = 1/s

filter Primitive

- *filter* primitive describes an LTI system with:
  - Transfer function: H(s)
  - Transport delay: delay

- The transfer function H(s) can be described in two forms:
  - (1) $H(s) = gain \times \frac{\left(1+\frac{s}{2\pi z_1}\right)\left(1+\frac{s}{2\pi z_2}\right)\ldots(1+\frac{s}{2\pi z_N})}{\left(1+\frac{s}{2\pi p_1}\right)\left(1+\frac{s}{2\pi p_2}\right)\ldots(1+\frac{s}{2\pi p_M})}$
  - (2) $H(s) = \sum_{i=1}^{n} \frac{b_i}{(s+a_i)^{m_i}}$
  - All coefficients can be complex numbers

filter Primitive (2)

- When using method #1 to describe H(s)
  - Need a list of poles and zeros
  - Format: a list of real and imaginary parts of poles/zeros
  - Also possible to use a file when the list is long

```
poles = ‘{real(p1), imag(p1), real(p2), imag(p2), …}
zeros = ‘{real(z1), imag(z1), real(z2), imag(z2), …}
```

- Example: a 1st order filter with a 400MHz pole

```
filter #(.poles(‘{4e8, 0}), .zeros(‘{0})) filter1(.in(in),
.out(out))
```

Pole at 4e8

No zeros

filter Primitive (3)

- List of parameters:

Name	Type	Default	Description
filename	string	" "	Filter parameter files
poles	Real array	'{1e9, 0.0}	Pole list
zeros	Real array	'{0}	Zero list
delay	Real	0	Transport delay
gain	Real	1	DC gain

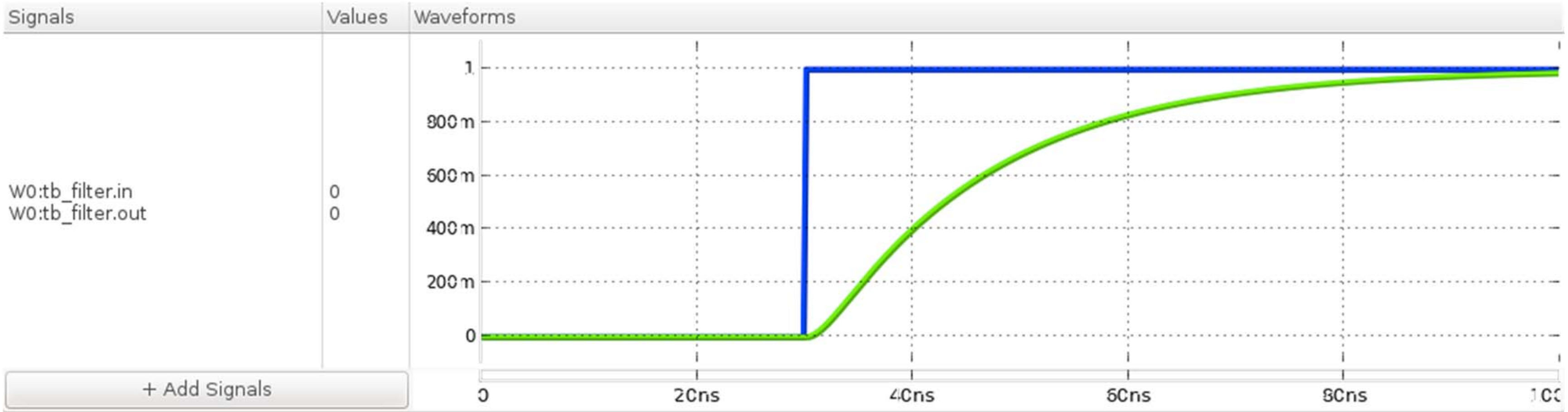
Exercise #5

- Simulate a step response of a linear filter that has two poles at 10MHz and 100MHz
  - Complete **prims/tb_filter/tb_filter.sv**

Answer #5

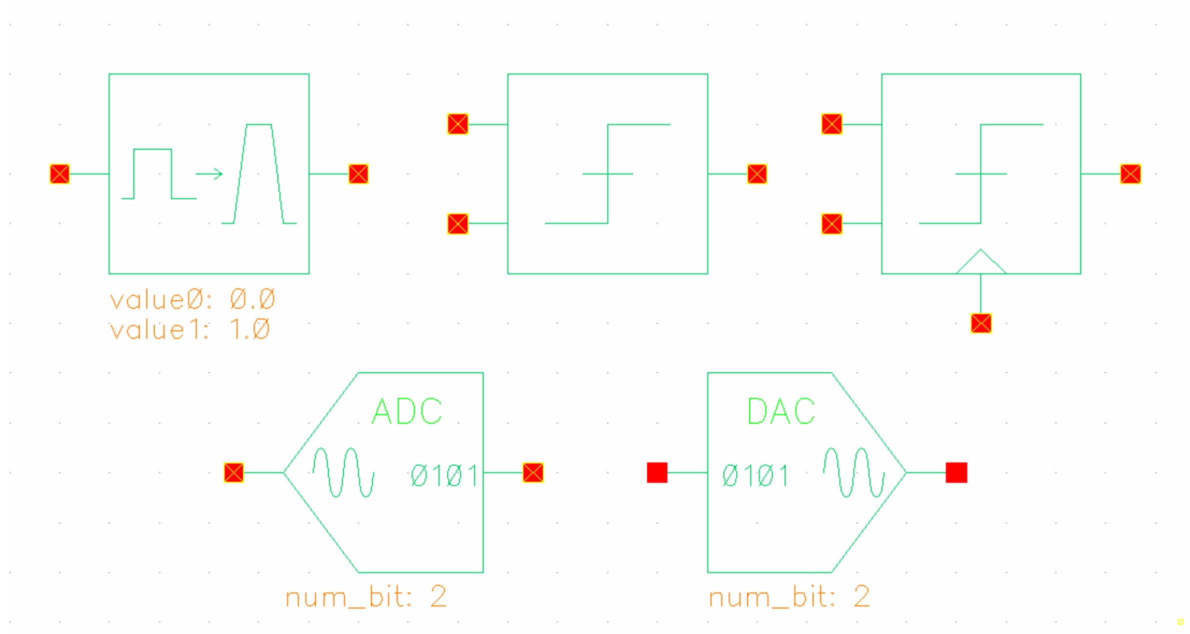
- Located at **prims/tb_filter/answer/tb_filter.sv**

```
filter      #(.poles('{10e6, 0, 100e6, 0}))
            filter(.out(out_signal), .in(in_signal));
```



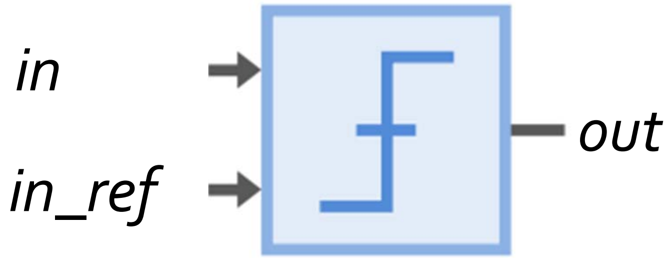
Primitive for A/D Conversion

- Primitives for converting between analog and digital signals
  - transition, slice, compare, dac, adc



slice Primitive

- Slice compares an ***xreal***-typed input signal to an ***xreal***-typed reference signal, and gives an ***xbit*** result



- Parameters:

Name	Type	Default	Description
threshold	real	0.0	Threshold for comparison

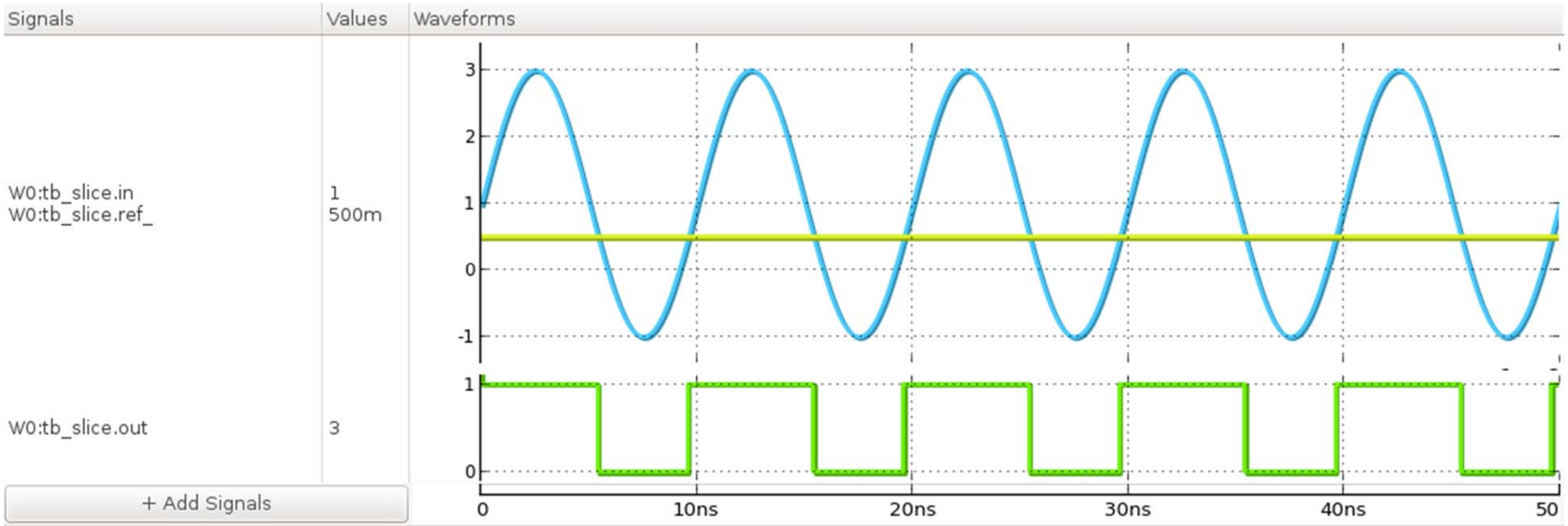
Exercise #6

- Generate a sinusoidal signal with a 2-V amplitude and 1-V offset and slice it at 0.5V
  - Complete **prims/tb_slice/tb_slice.sv**

Answer #6

- Located at **prims/tb_slice/ answer/tb_slice.sv**

```
xreal       in_signal, ref_signal;
xbit        out_signal;

sin_gen     #(.freq(100e6), .offset(1.0), .amp(2.0))
            in_sig_gen(in_signal);
dc_gen      #(.value(0.5))
            ref_sig_gen(ref_signal);
slice       #(.threshold(0.0))
            slice(.out(out_signal), .in(in_signal), .in_ref(ref_signal));
```

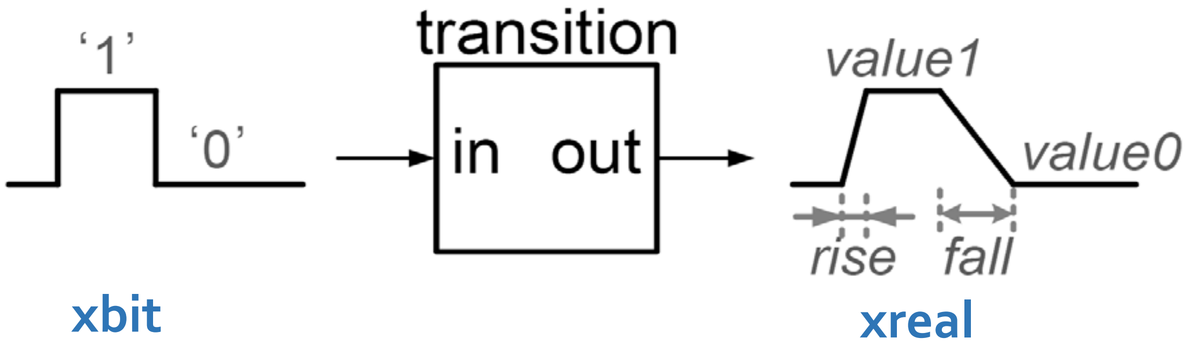


compare vs. *slice* Primitives

- The **compare** primitive is similar to **slice** primitive except that **compare** requires a triggering clock
  - ***compare*** models a clocked comparator
  - ***slice*** models a continuous slicer
- The **compare** primitive can also model time-varying characteristics such as:
  - Finite sampling aperture (limited bandwidth)
  - Finite regeneration time (limited gain; metastability)
  - Latency varying with input magnitude
  - See more details in the *XMODEL reference manual*

transition Primitive

- Converts an xbit-typed signal to an xreal-typed signal with finite rise and fall transition times



- Parameters:

Name	Type	Default	Description
value0	real	0.0	Signal level for an xbit input '0'
value1	real	1.0	Signal level for an xbit input '1'
rise_time	real	0.0	Transition time from the input level '0' to '1'
fall_time	real	0.0	Transition time from the input level '1' to '0'
delay	real	0.0	Propagation delay

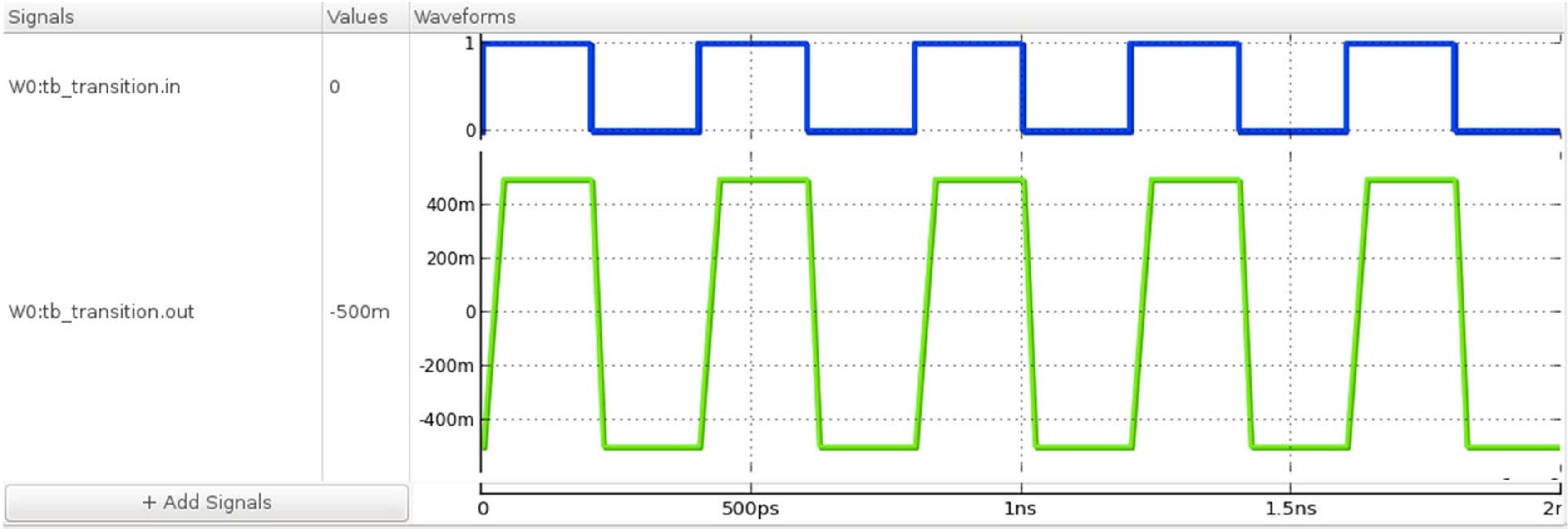
Exercise #7

- Convert a periodic clock signal into an analog signal that swings between **-5** and **+5** with a rise time of **37ps** and fall time of **22ps**
  - Complete **prims/tb_transition/tb_transition.sv**

Answer #7

- Located at **prims/ tb_transition/answer/ tb_transition.sv**

```
transition  #(.value0(-0.5), .value1(0.5),
              .rise_time(37e-12), .fall_time(22e-12))
            transition(.out(out_signal), .in(in_signal));
```



Exercises with Function Primitives (2)

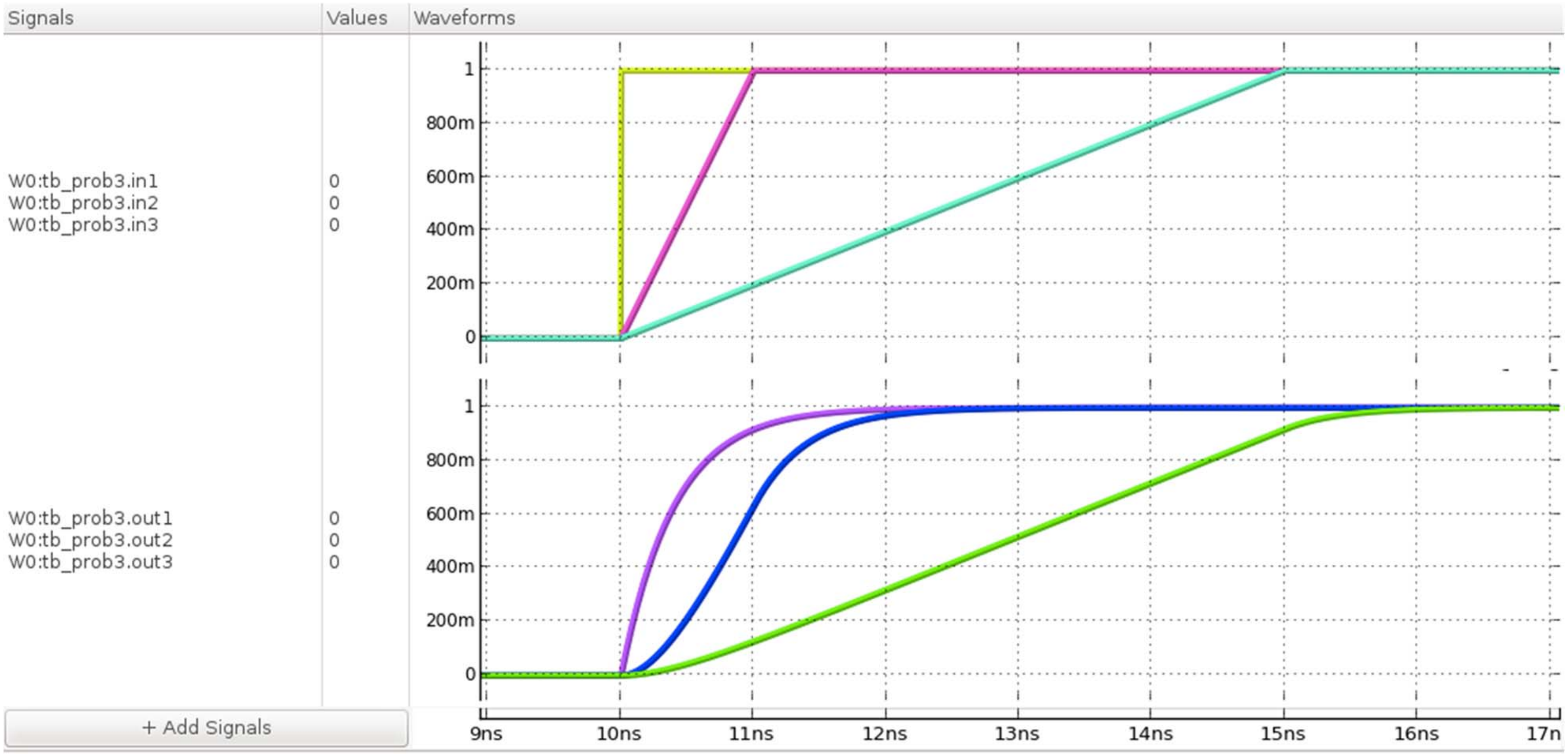
- Prob. #3: Simulate the responses of a 1st-order filter with a 400-MHz bandwidth to the following inputs
  - An ideal step
  - A ramp with a slope of 1V/1ns
  - A ramp with a slope of 1V/5ns
- Prob. #4: Design an 3-bit ADC with arbitrary threshold levels given as:
  - 0.5, 1.1, 1.8, 2.5, 3.2, 4.0, 4.7
- Complete the skeletons in **prims/prob3** and **prims/prob4**

Answer: Prob #3

- Located at: **prims/prob3/answer/prob3.sv**

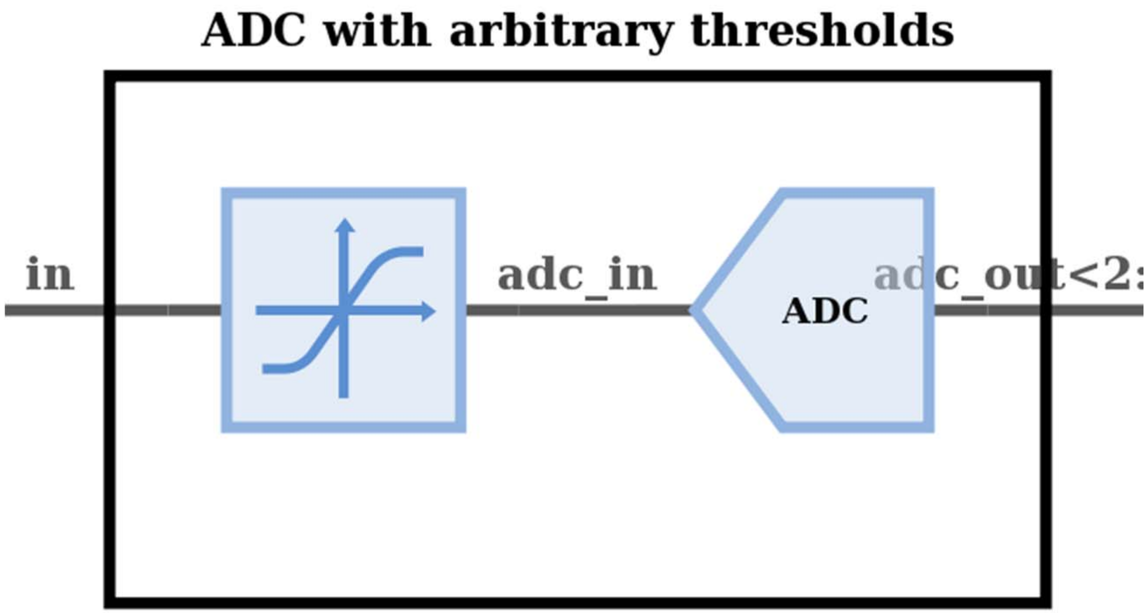
```
`include "xmodel.h"

module prob3;
    xreal in0 ,in1, in2;
    xreal out0, out1, out2;

    step_gen    #(.change(1)) p0(in0);
    pwl_gen     #(.data('{0, 0, 1e-9, 1})) p1(in1);
    pwl_gen     #(.data('{0, 0, 5e-9, 1})) p2(in2);

    filter      #(.poles('{4e8, 0})) filter0(.in(in0), .out(out0));
    filter      #(.poles('{4e8, 0})) filter1(.in(in1), .out(out1));
    filter      #(.poles('{4e8, 0})) filter2(.in(in2), .out(out2));

    initial begin
        $xmodel_dumpfile();
        $xmodel_dumpvars();
    end
endmodule
```

Answer: Prob #3 (2)



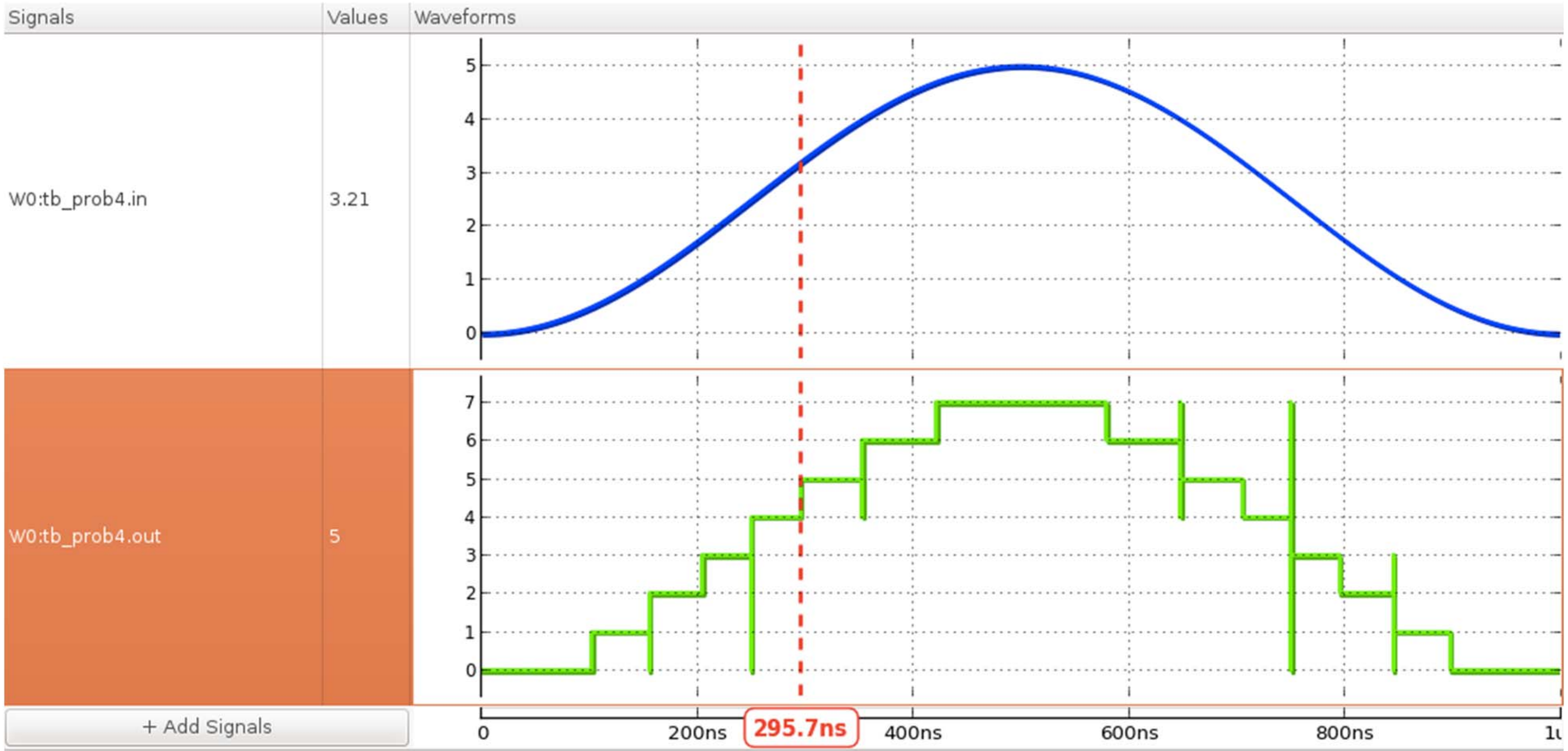
Answer: Prob #4

- An ideal ***adc*** preceded by a ***pwl_func*** primitive can model any non-ideal ADCs
- Q: how would you determine the piecewise linear function?



Answer: Prob #4 (2)

- Located at: **prims/prob4/answer/prob4.sv**

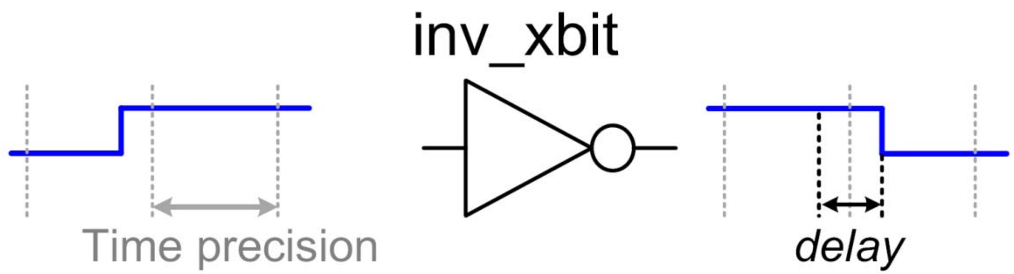


Logic Gate Primitives

- Logic gate primitives perform the logical operation on ***xbit***-typed digital signals
  - Available ones include: inv_xbit, buf_xbit, and_xbit, or_xbit, xor_xbit, delay_xbit, interp_xbit, mux_xbit, dff_xbit, tribuf_xbit, …
- Note that ***xbit***-typed signals model digital signals whose timing must be accruate (e.g. clocks & pulses)
  - e.g. performing analog operations in time domain
- For other Boolean signals, use ***wire*** or ***reg*** types in Verilog

inv_xbit Primitive

- Inverts the input signal with an accurate delay



- Parameter:

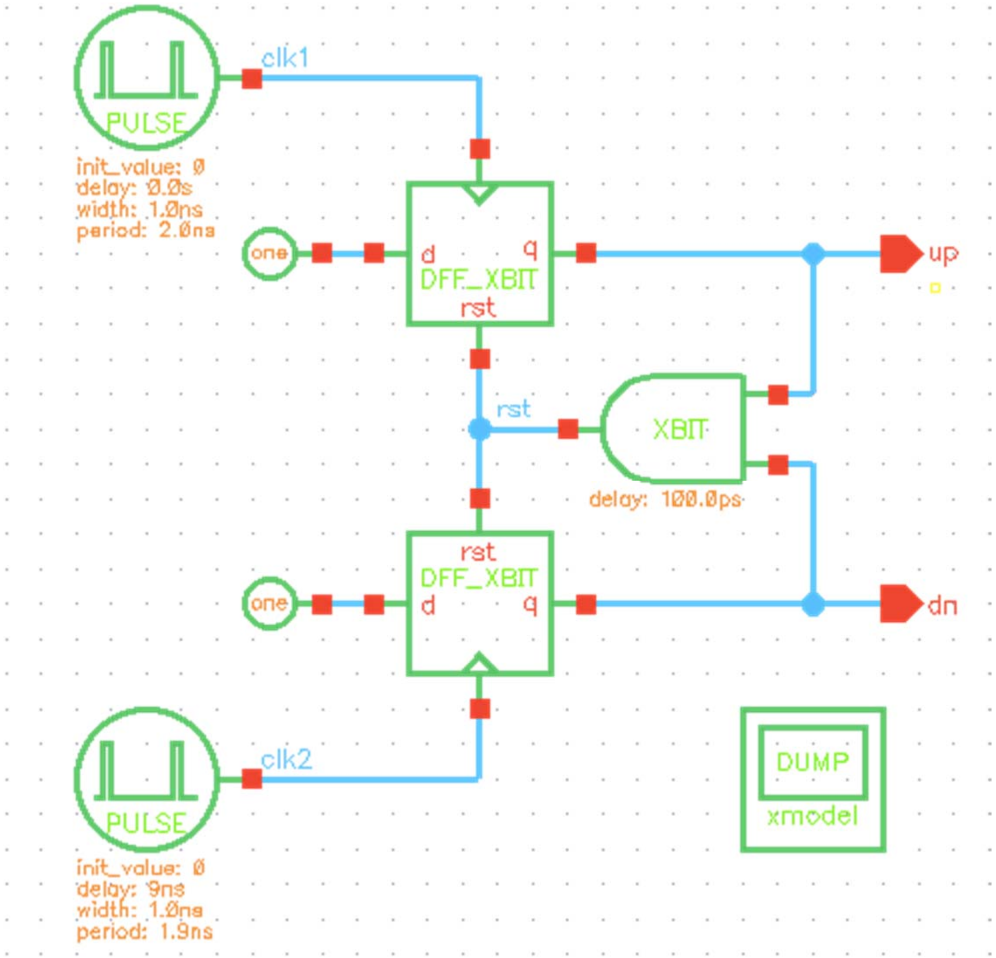
Name	Type	Default	Description
delay	real	0.0	Propagation delay

- Example: an inverter with 120ps delay:

```
inv_xbit  #(.delay(120e-12))  inv1 (.out(out),.in(in));
```

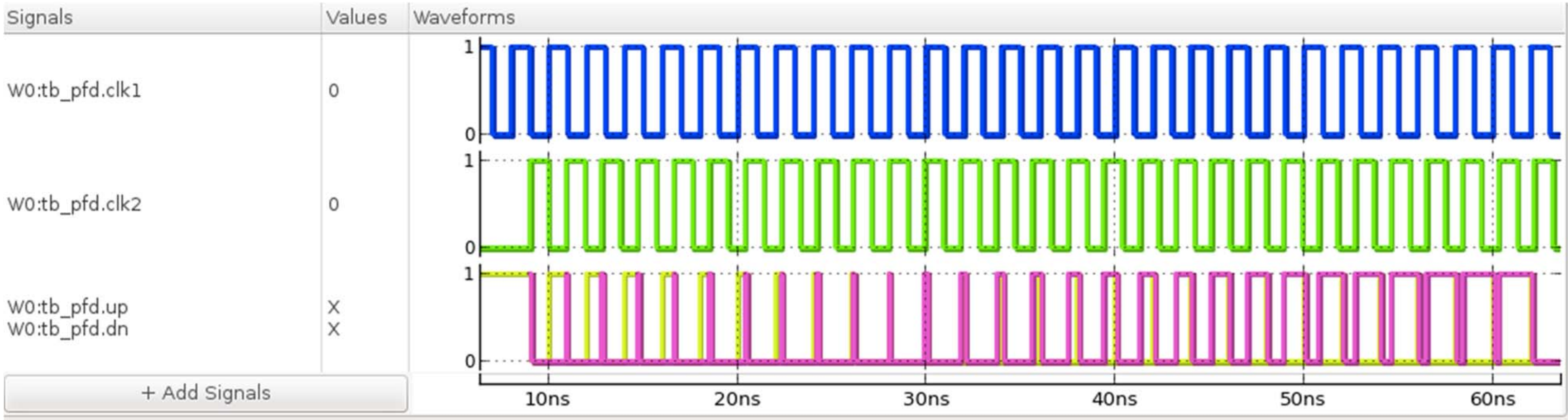
Exercise #8

- Simulate the responses of a phase-frequency detector (PFD) to two input clocks with a small frequency difference
  - Testbench is in **prims/tb_pfd**



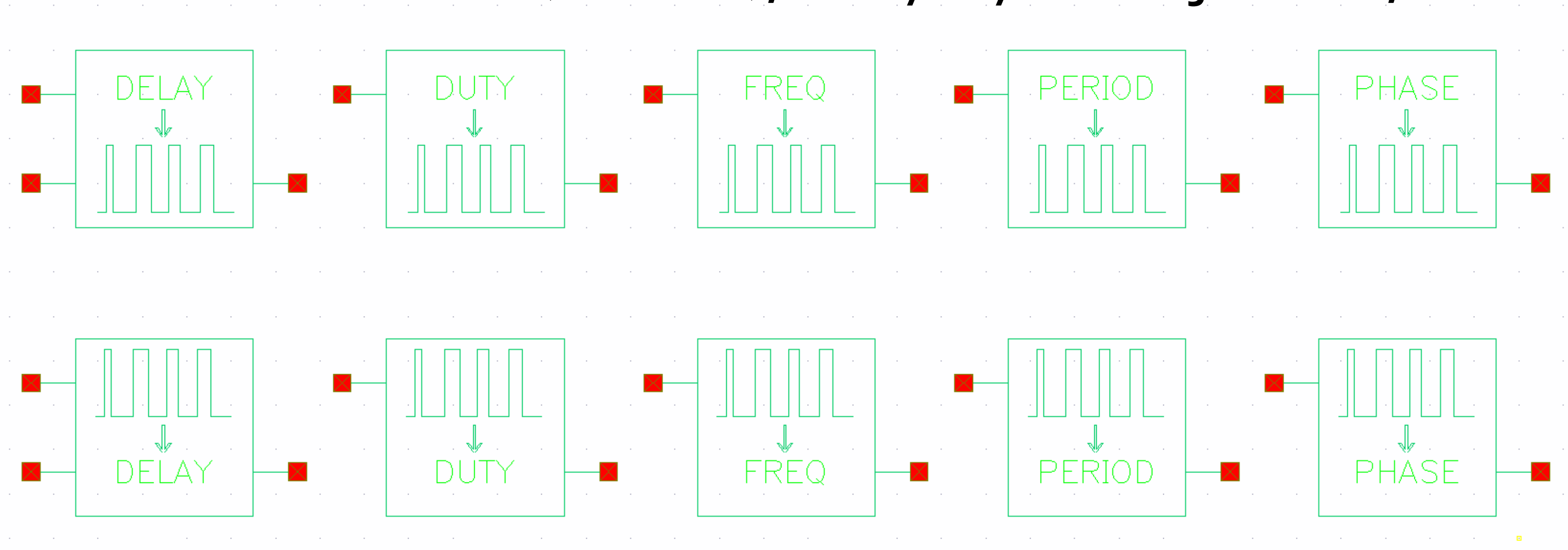
Answer #8: Simulation Waveforms

- The up/down output signals have net pulsewidths that correspond to the timing error between the two input clocks



Variable Domain Translators (VDT)

- VDT primitives convert between a clock and its property (such as frequency, period, phase, duty-cycle, delay, etc.)
- Useful when modeling oscillators, delay-lines, pulse-width modulators (PWMs), duty-cycle adjusters, …

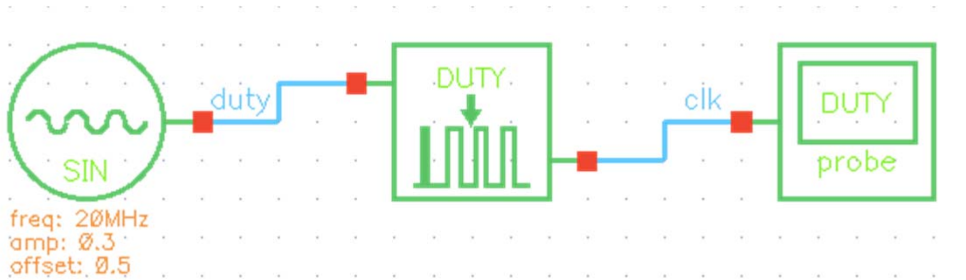


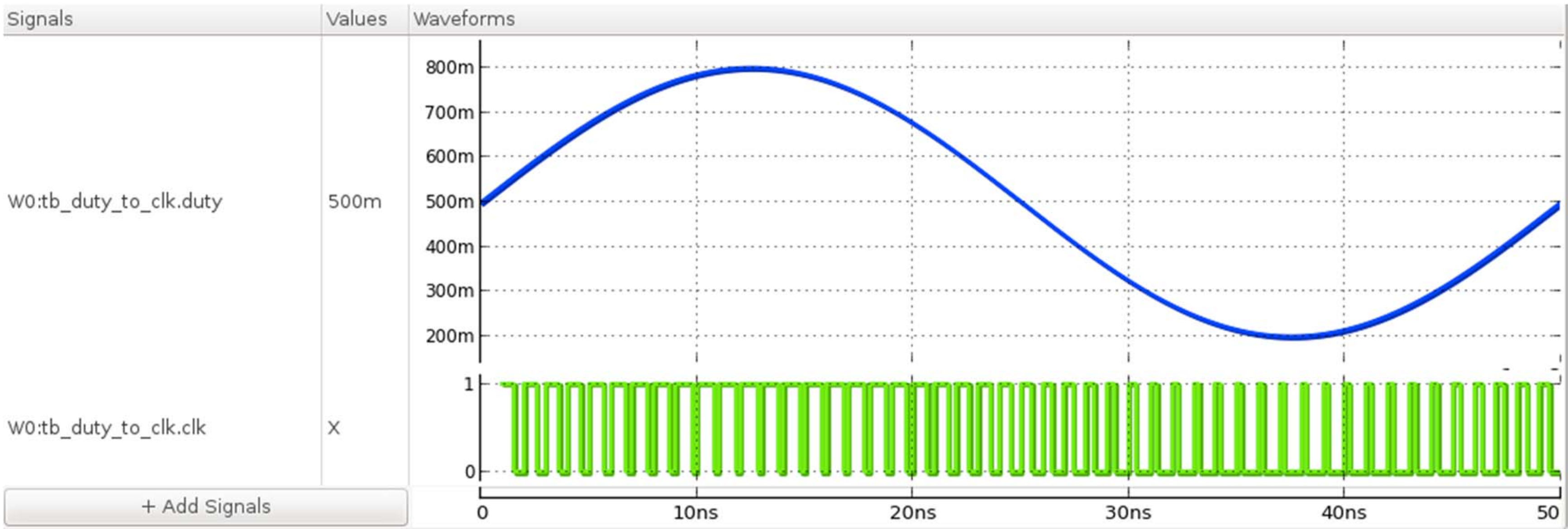
Exercise #9

- Generate a 1-GHz clock whose duty-cycle varies as a 20-MHz sinusoid ranging from 20% to 80%
  - Complete **prims/tb_duty_to_clk/tb_duty_to_clk.sv**

Answer #9

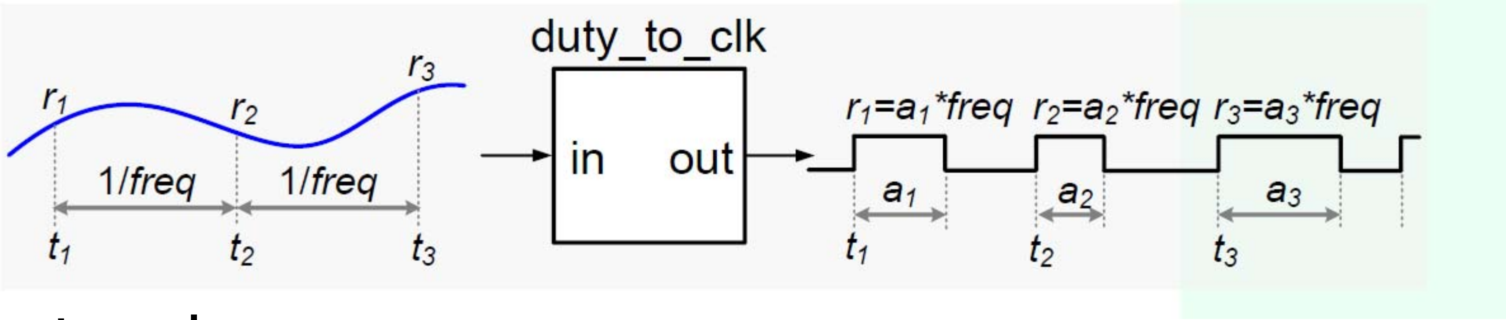
- Located at **prims/ tb_duty_to_clk/ answer/tb_duty_to_clk.sv**





duty_to_clk Primitive

- Generates a clock signal whose duty-cycle varies with the input signal



- I/O signals

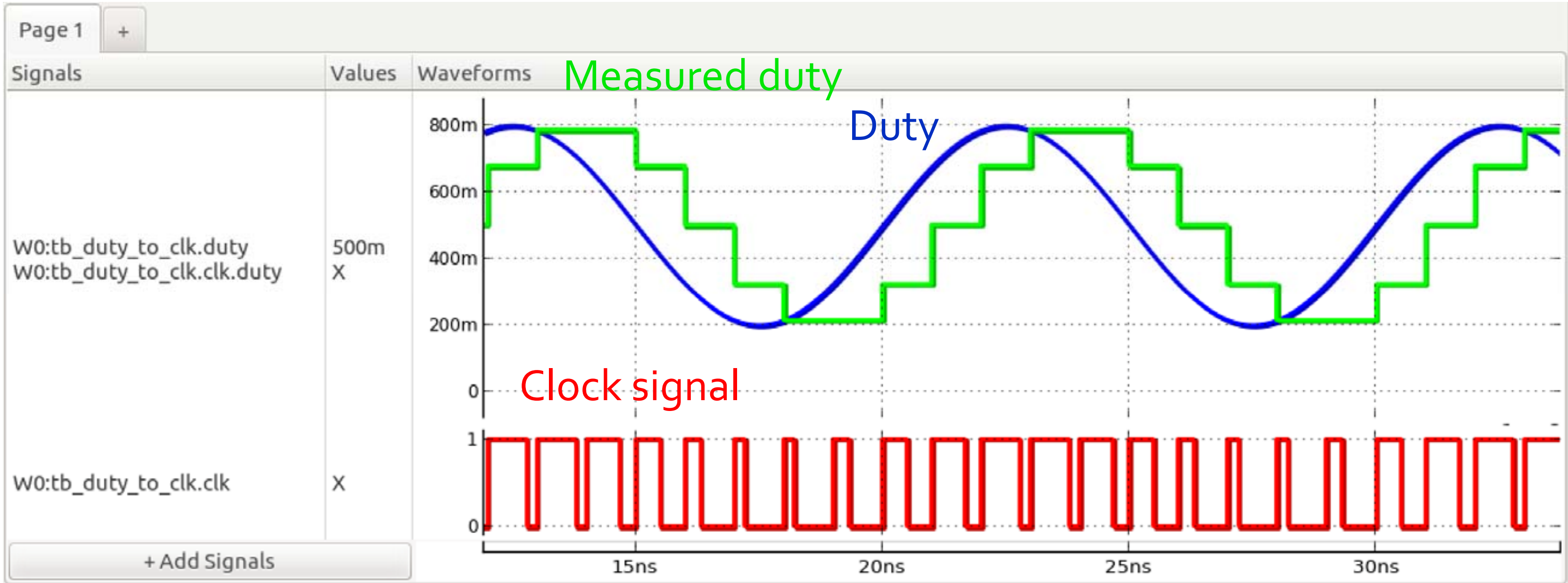
Name	I/O	Type	Description
out	output	xbit array	Clock output
in	Input	xreal	Duty-cycle input

- Parameters

Name	Type	Default	Description
freq	real	1e9	Frequency
...			

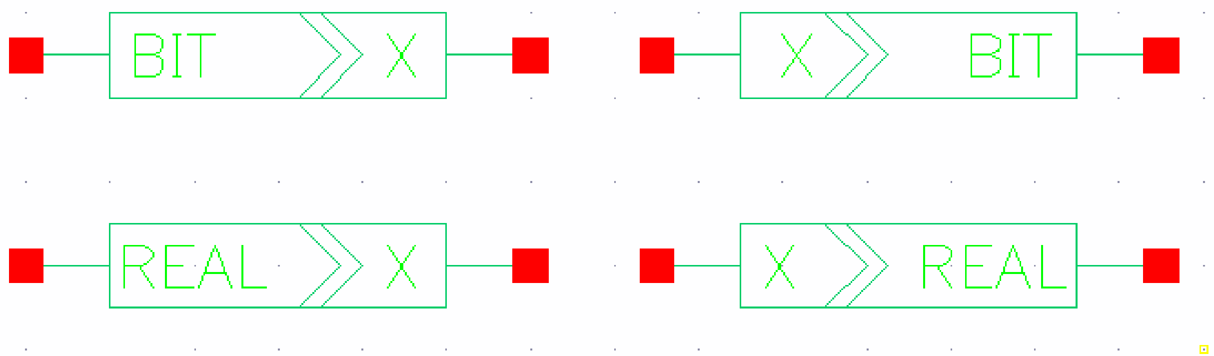
VDT Probe Primitives

- Some probes can measure the properties of a clock directly
  - Examples: probe_freq, probe_period, probe_phase, probe_delay, probe_duty, ...



Connect Primitives

- Connect primitives convert between xreal <-> real or xbit <-> bit
  - Useful for interfacing non-XMODEL models (e.g. Verilog models, SPICE models, ...)



real_to_xreal / xreal_to_real Primitive

- Convert an real-type / xreal-type signal to a xreal-type / real type signal, respectively
- I/O description:

Name	I/O	Type	Description
out	output	xreal/real	xreal/ real output signal.
in	Input	real/xreal	real/ xreal Input signal.

- Parameters (for *xreal_to_real*):

Name	Type	Default	Description
mode	string	“variable”	Sampling mode (“variable” or “fixed”)
period	real	0.0	Sampling period (for “fixed”)
abstol	real	1e-6	Absolute tolerance
reltol	real	1e-3	Relative tolerance

Notes on Converting *xreal* to *real*

- *XMODEL* gives you fast speed because it generates very small number of events while describing accurate analog waveforms
- However, if you convert an actively changing signal to a real-typed variable, many events will be generated
  - The very reason why Real-Number Verilog is slow
- Spare-use real-typed variables only for signals that does not vary (DC) or vary in a discrete fashion

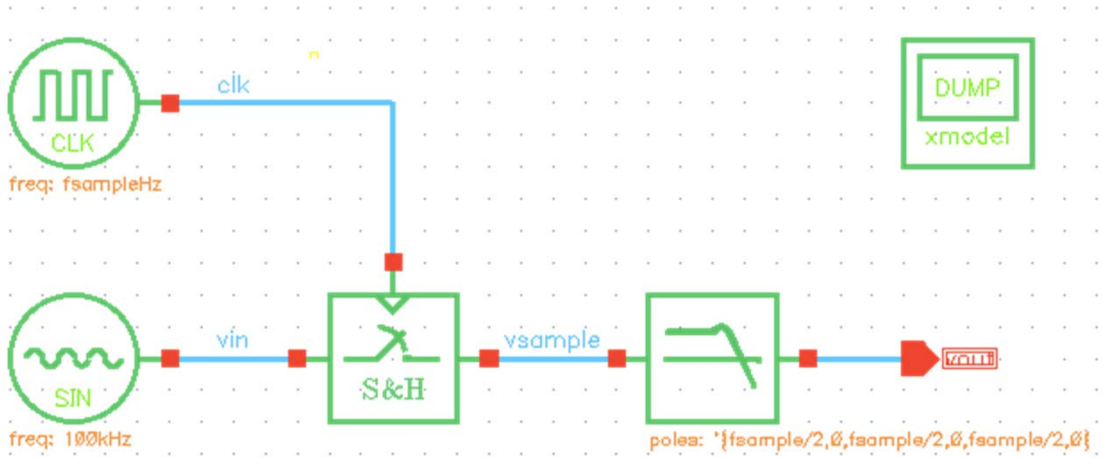
Exercises with VDT Primitives

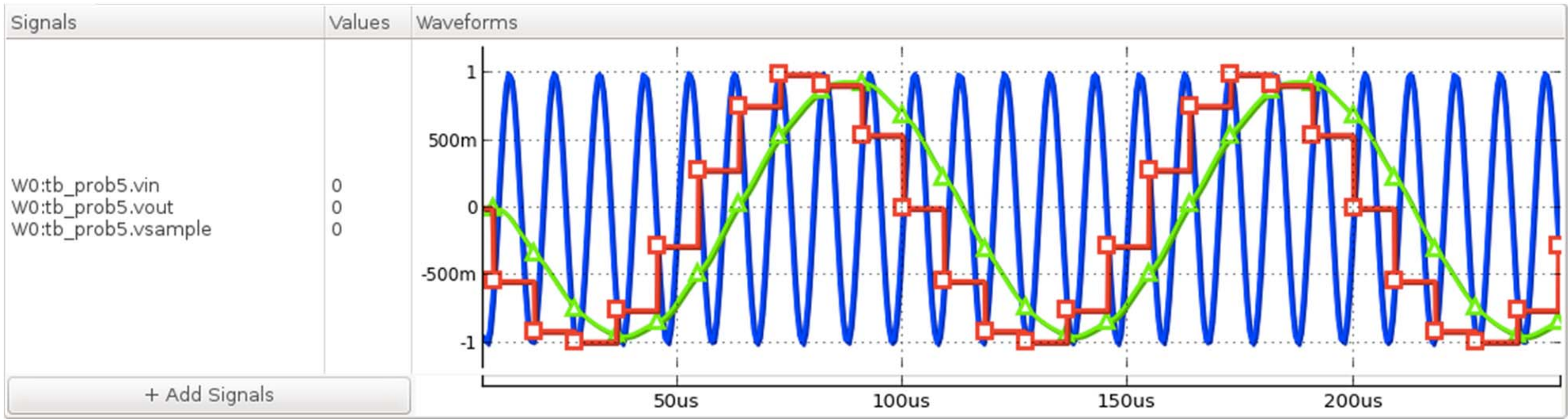
- Prob. 5: sample a 100-kHz sinusoidal signal with various rates and try to reconstruct the original signal with a low-pass filter
  - Try sampling rates of 110KHz, 210KHz, 500KHz, 10MHz
  - Observe any aliasing effects
- Prob. 6: generate a 100-kHz clock whose duty varies as x
  - x is a clipped signal of y within the range of [0.05, 0.9]
  - y is the absolute value of a 60-Hz, unit-amplitude sinusoid
- Complete **prims/prob5** and **prims/prob6**

Answer: Prob #5

- Located in **prims/prob5/ answer/ prob5.sv**

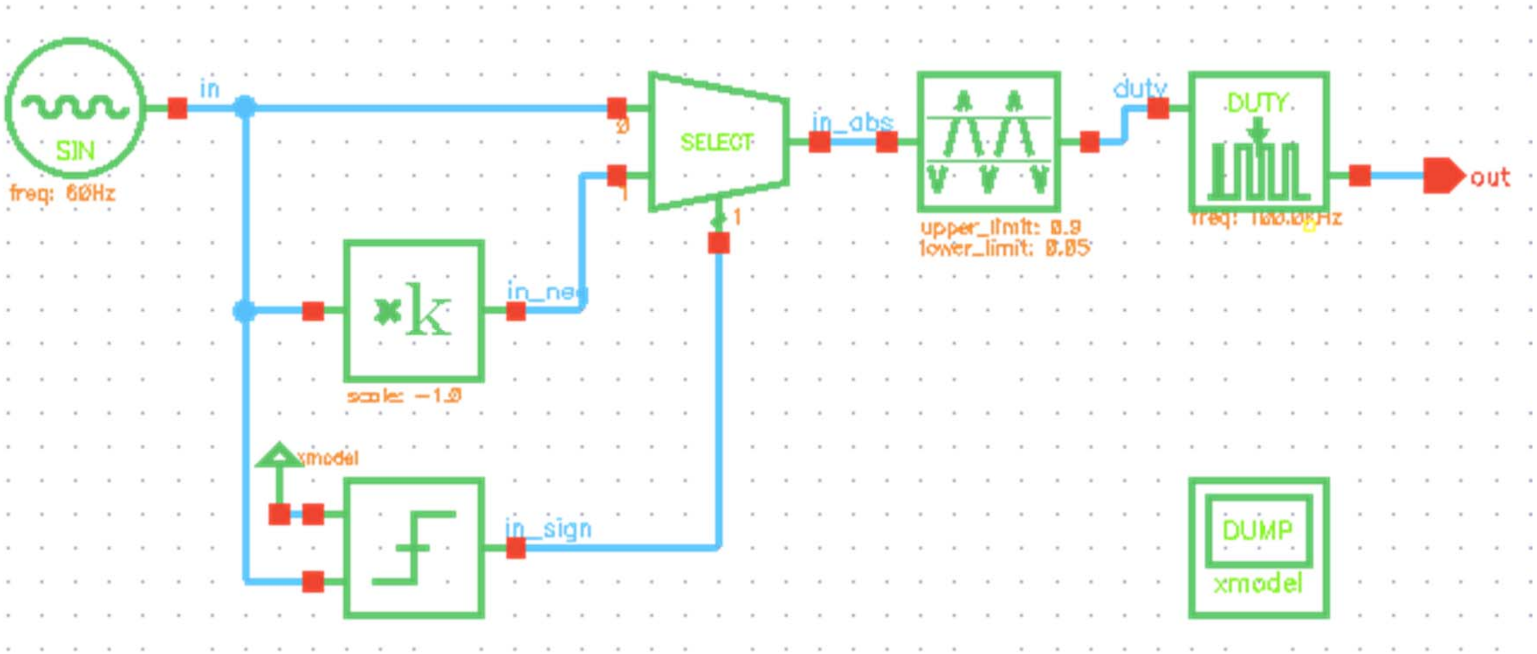
  fsample = 110kHz





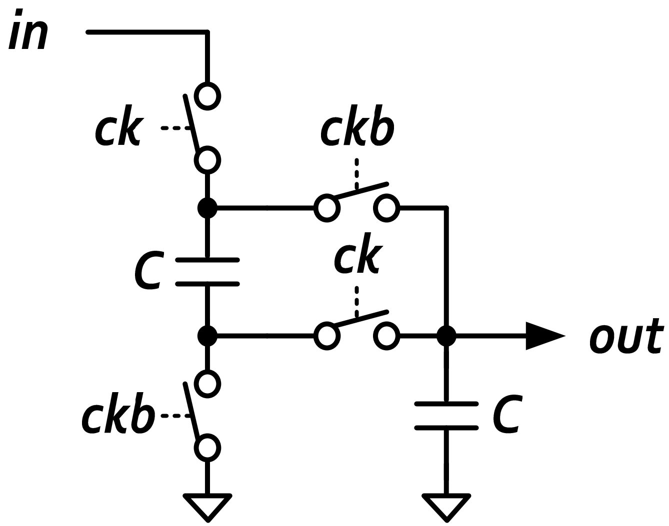
Answer: Prob #6

- Located in **prims/prob6/answer/prob6.sv**
- Other alternative ways of computing the absolute value exist



Circuit Primitives

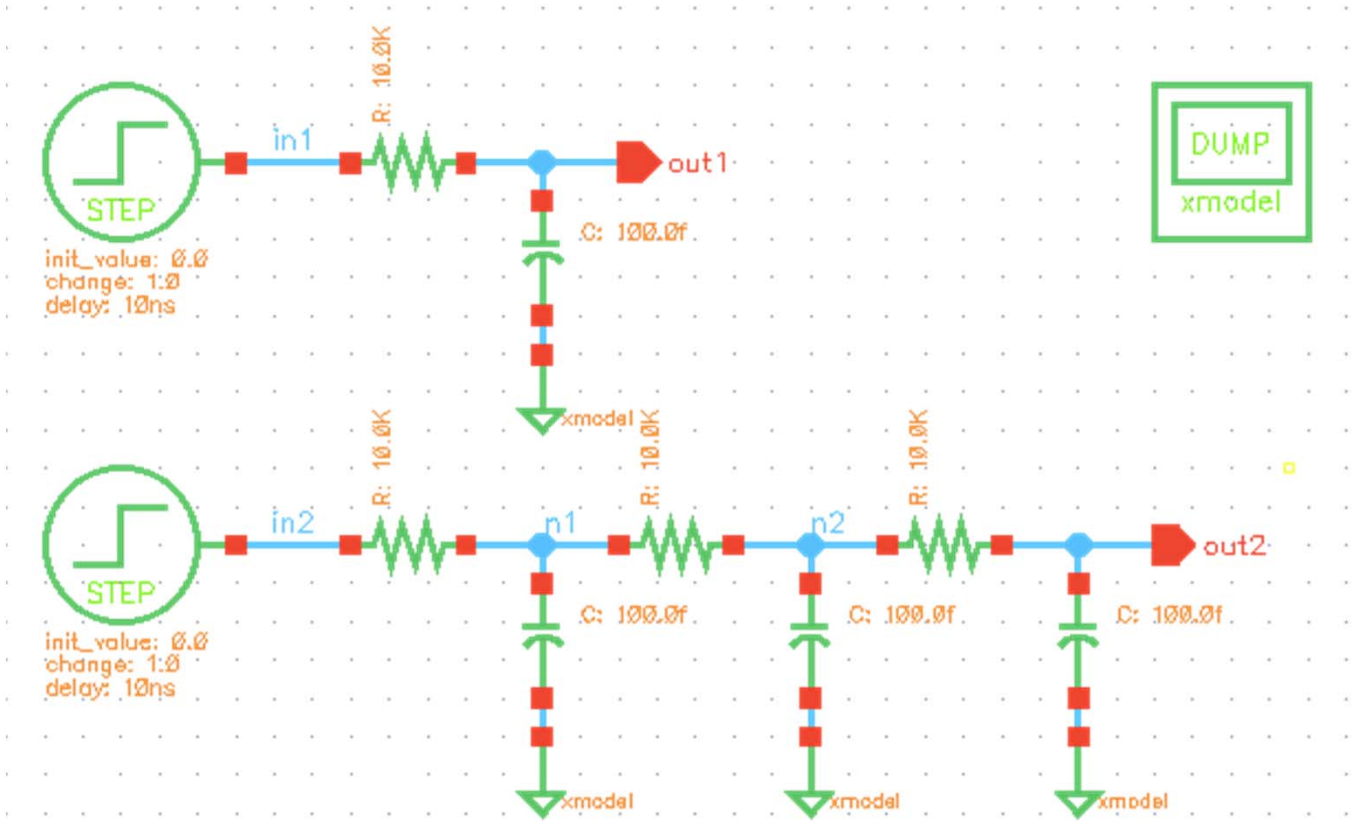
- With circuit primitives, you can model circuits directly in SystemVerilog
  - Available ones: resistor, capacitor, inductor, diode, nmosfet, ...
  - Useful when modeling loading effects, nonlinear behaviors, time-varying (switching) behaviors, ...




```
module sc_converter(
      input xreal in,
      output xreal out,
      input xbit ck, ckb
);
xreal     n1, n2;
switch    sw1(.pos(in), .neg(n1), .ctrl(ck));
switch    sw2(.pos(n1), .neg(out), .ctrl(ckb));
switch    sw3(.pos(n2), .neg(out), .ctrl(ck));
switch    sw4(.pos(n2), .neg(`ground), .ctrl(ckb));
capacitor #(.C(1e-12)) C1(.pos(n1), .neg(n2));
capacitor #(.C(1e-12)) C2(.pos(n2), .neg(`ground));
endmodule
```

Exercise #10

- Simulate the step responses of RC-filters
- Testbench: **prims/tb_rc_filter**



capacitor Primitive

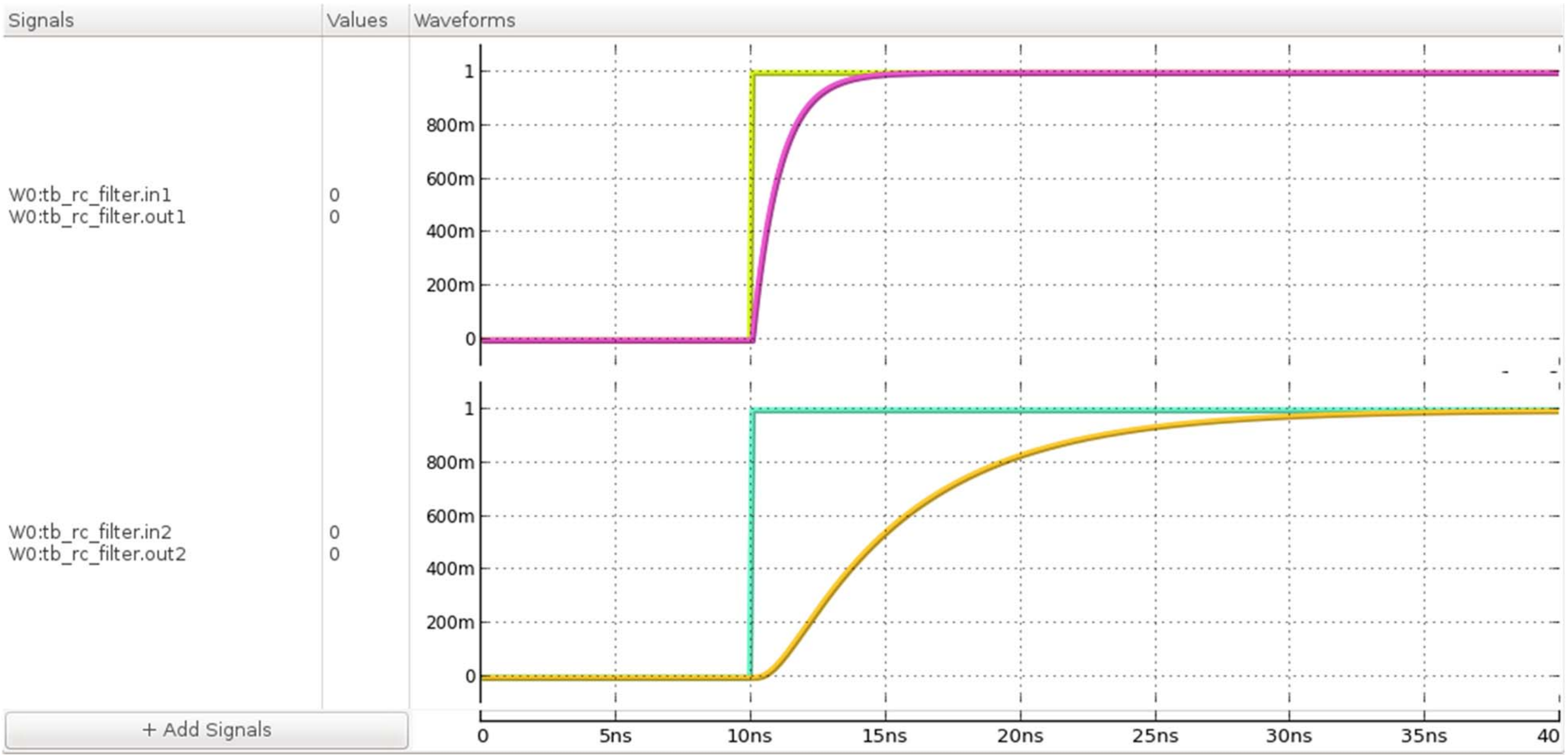
- This primitive models a two-terminal capacitor
- I/O description:

Name	I/O	Type	Description
pos	Input	xreal	Positive terminal
neg	Input	xreal	Negative terminal

- List of parameters:

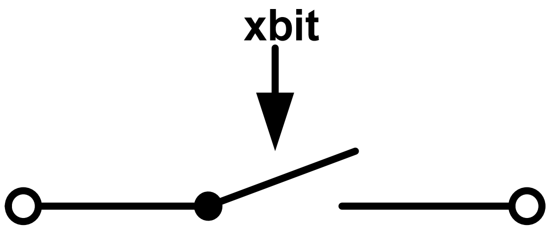
Name	Type	Default	Description
C	real	1F	Capacitor size
ic	real	0	Initial condition

Answer #10: Simulated Waveforms



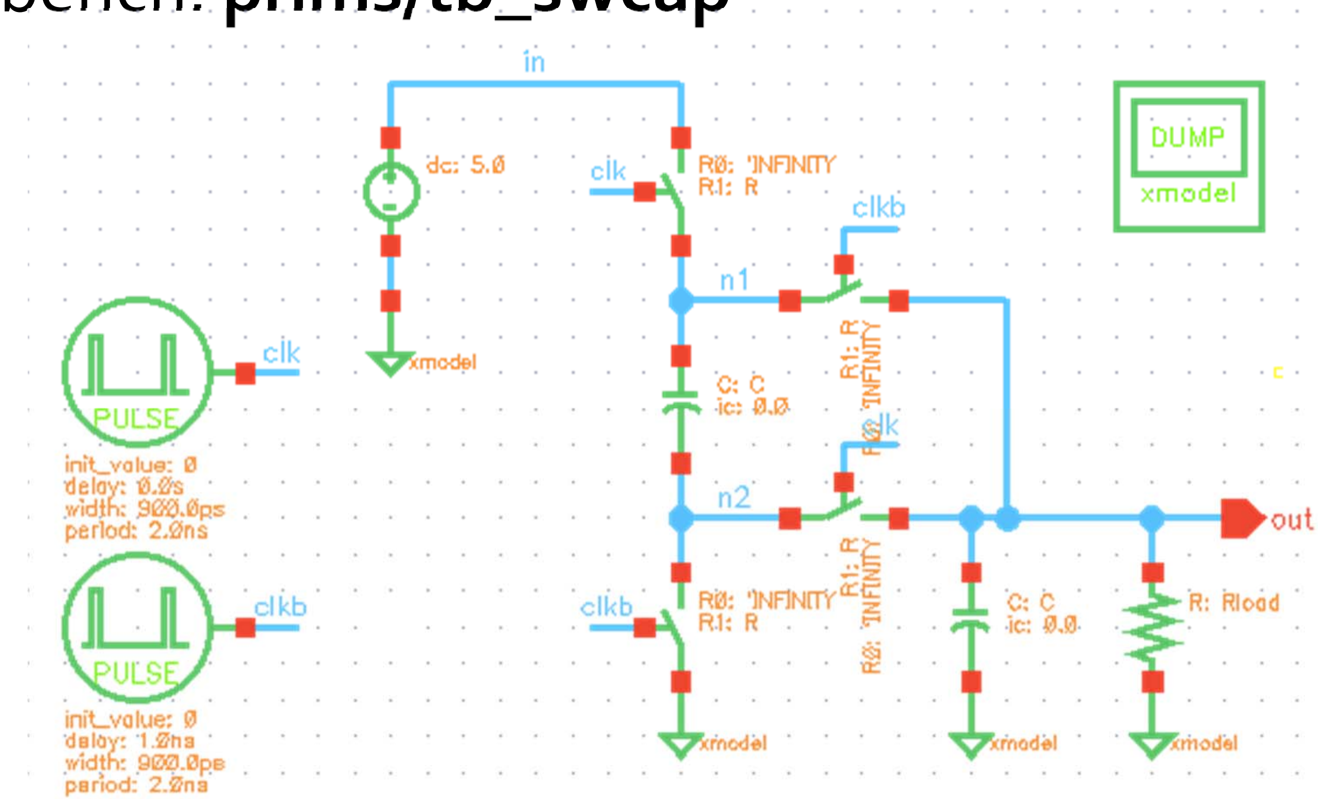
switch Primitive

- The ***switch*** primitive models a variable resistance controlled by a digital, ***xbit***-typed input (**ctrl**)
  - ***R0*** is the resistance when the **ctrl**=0
  - ***R1*** is the resistance when the **ctrl**=1
  - One of them is the on-resistance (low value) while other is the off-resistance (high value)



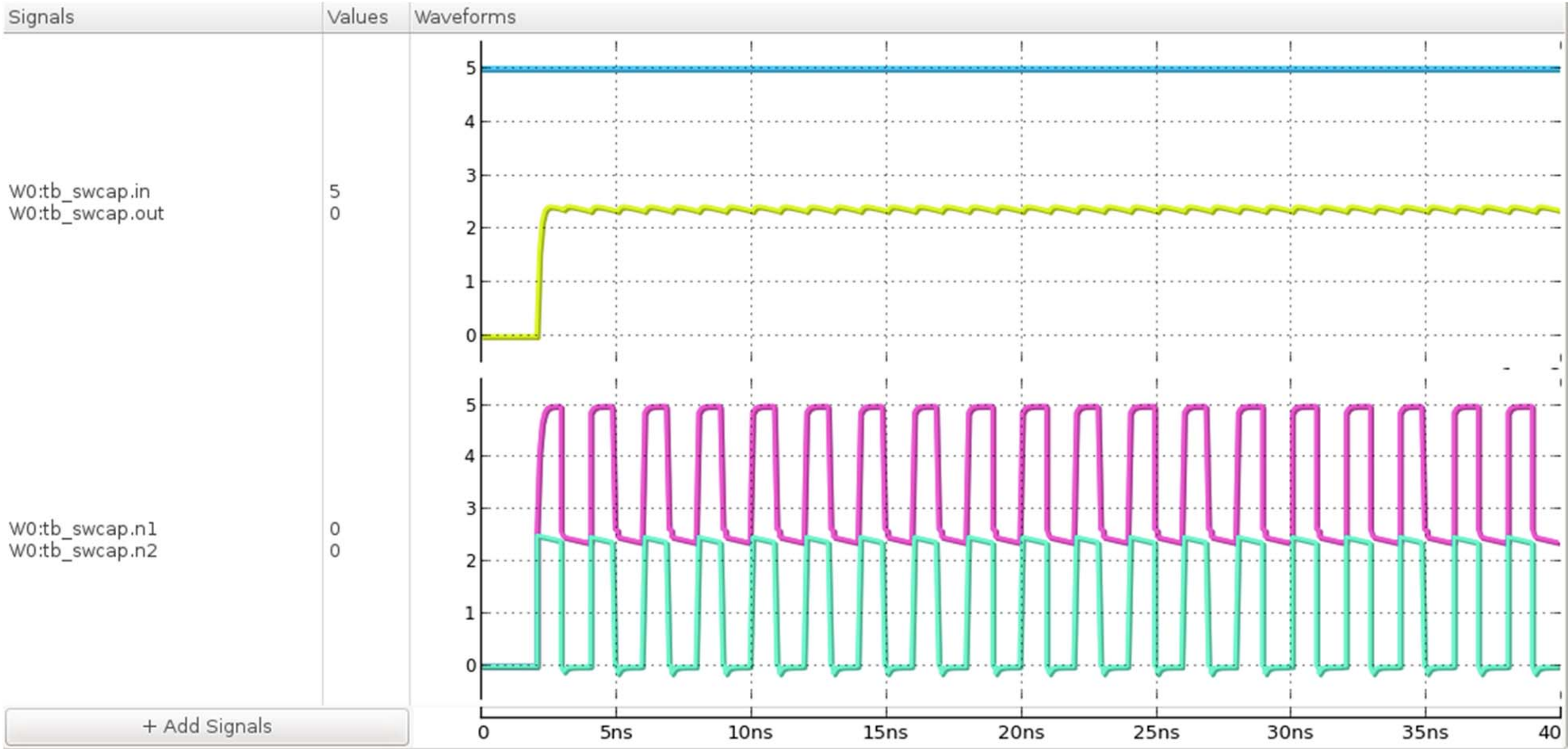
Exercise #11

- Simulate the settling response of a switched-capacitor 2:1 step-down DC-DC converter
- Testbench: **prims/tb_swcap**



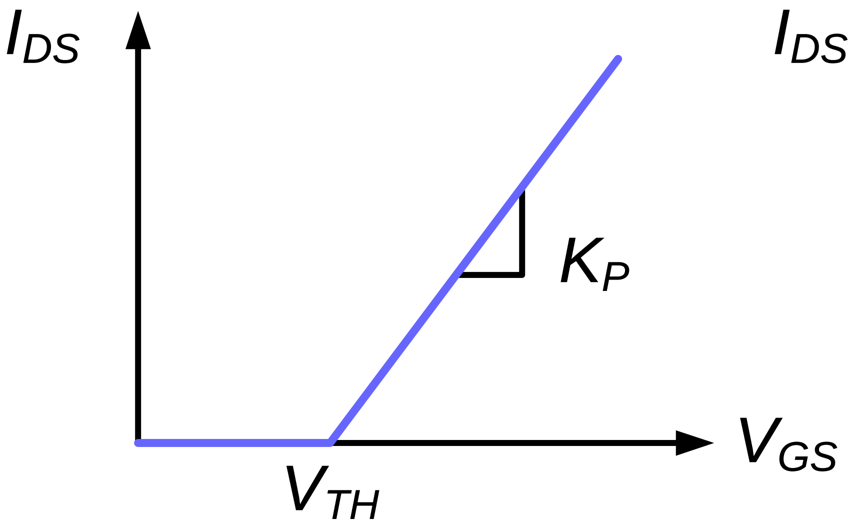
Answer #11

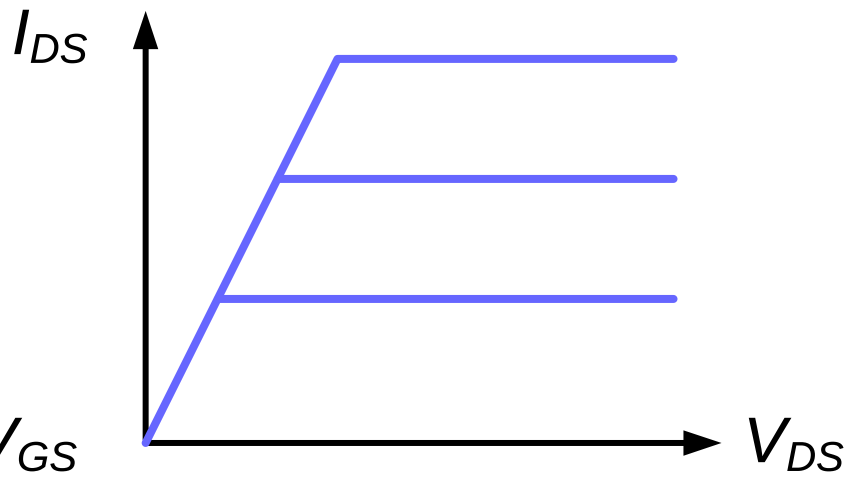
- The output voltage settles to ~2.5V as expected
- Try seeing the event markers by pressing ‘M’



nmosfet / pmosfet Primitive

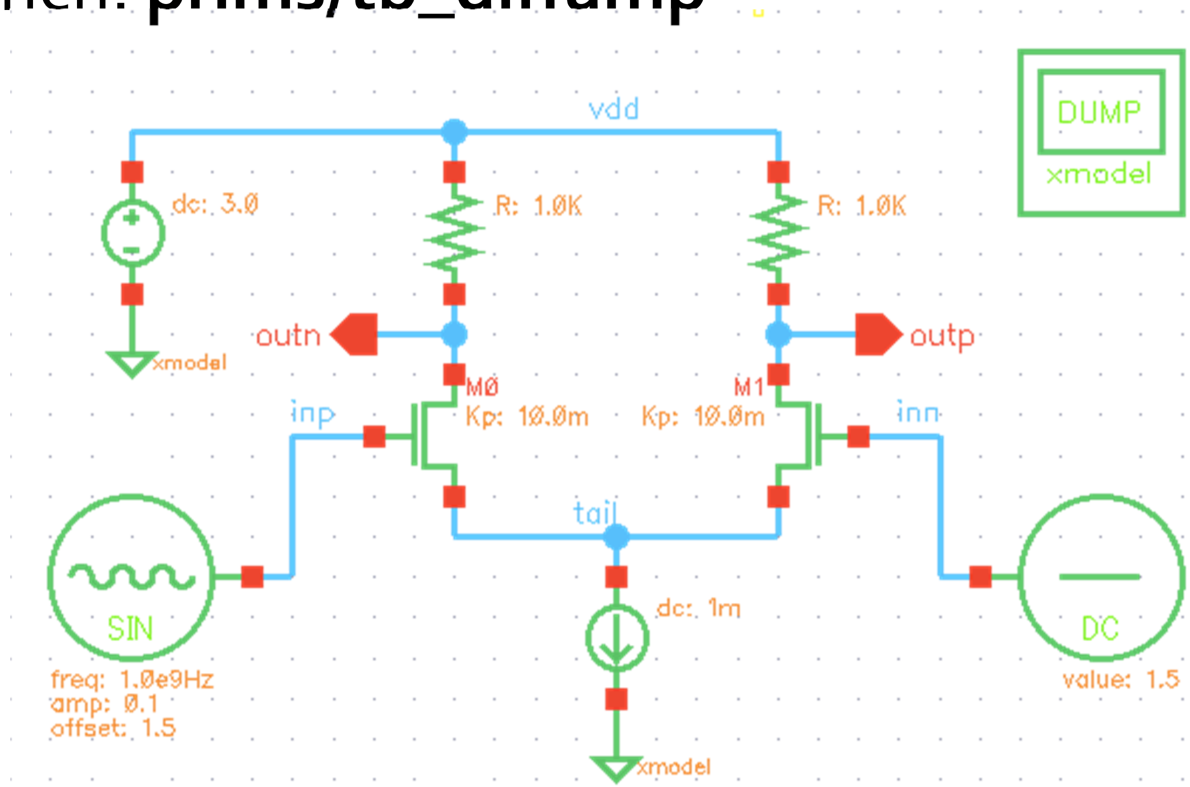
- *nmosfet* and *pmosfet* primitives approximate the MOSFET transistor behaviors as linear I_{DSAT} model
- Accurate enough for high-V_{GS}, velocity-saturated devices but not for devices near thresholds





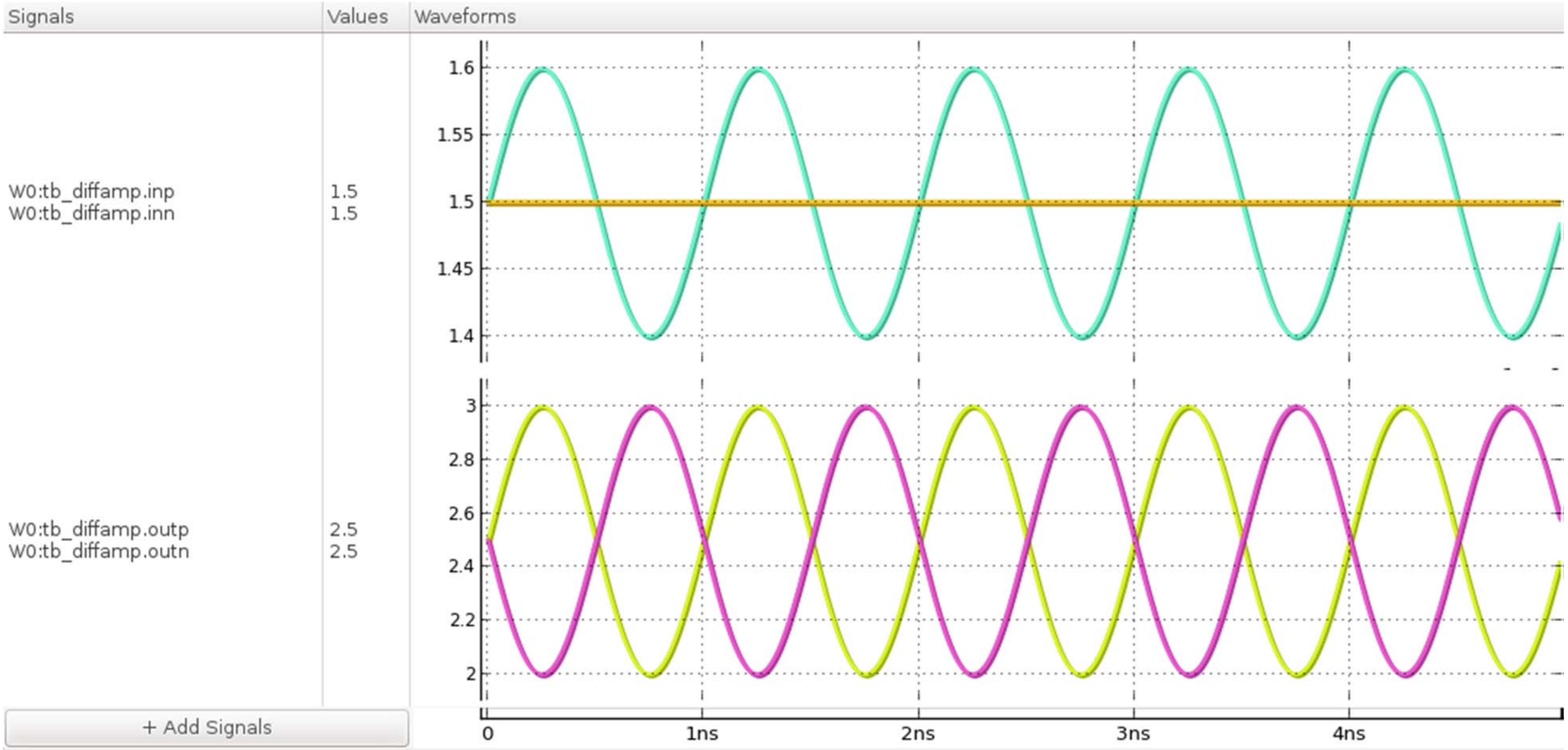
Exercise #12

- Simulate the response of a differential amplifier with different input amplitudes
- Testbench: **prims/tb_diffamp**



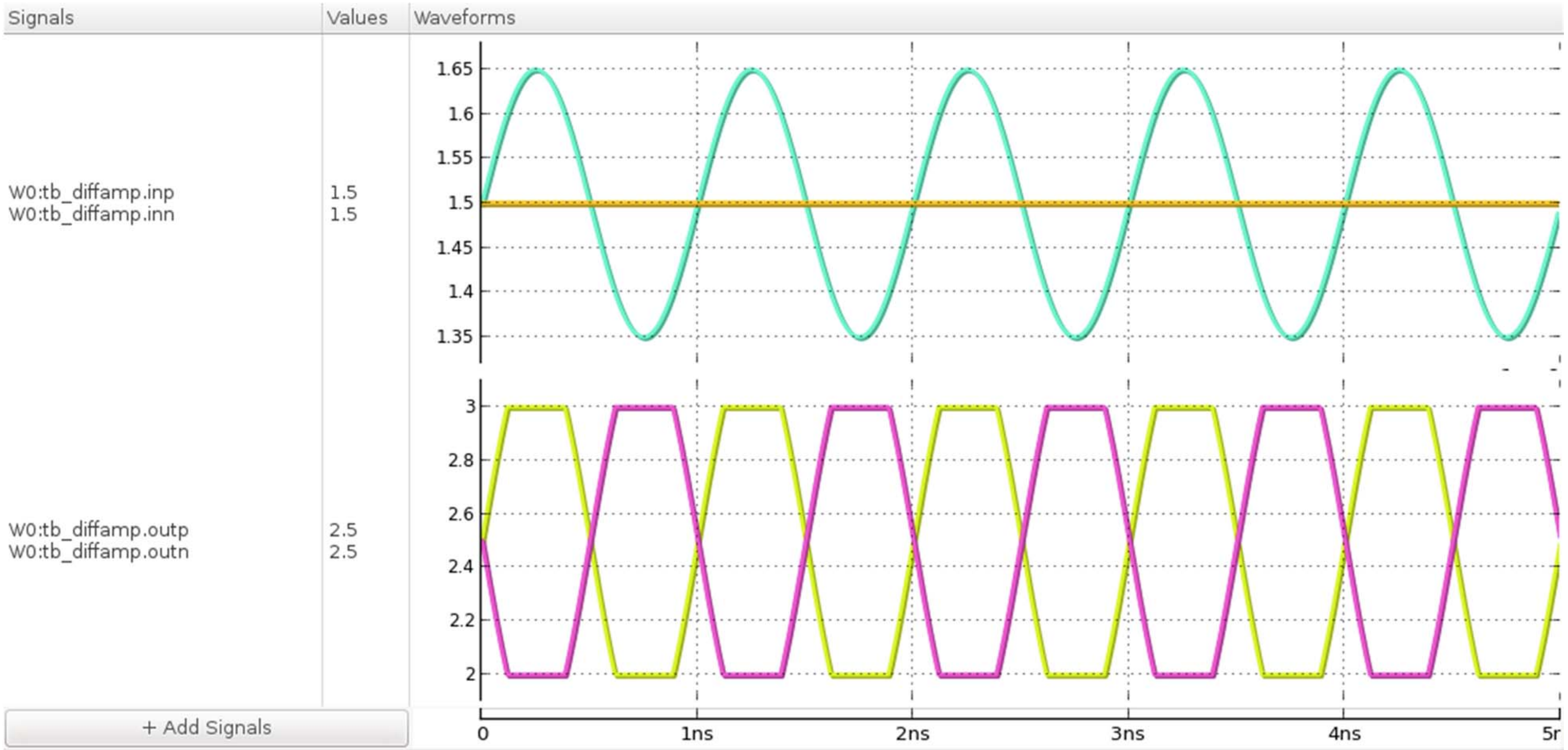
Answer #12

- With 0.1V-amplitude swing on the input *inp*
- Press ‘M’ to see how many events are generated



Answer #12 (2)

- With 0.15V-amplitude swing on the input *inp*
- How many events are generated this time?



scientific
analog

Waveform Dumping in *XMODEL*

Scientific Analog, Inc.

November 2016

Overview

- *XMODEL* provides a set of system calls to facilitate waveform dumping
  - $xmodel_dumpfile, $xmodel_dumpvars, …
  - Analogous to Verilog's $dumpfile, $dumpvars, …
- The waveform can be dumped either in JEZ format or in FSDB format
  - JEZ is the proprietary format that gives the fastest simulation speed (can be viewed using *XWAVE*)
  - FSDB-format files can be viewed using other commercial waveform viewers

Available Commands

- **$xmodel_dumpfile()**
  - Defines the dump file name and format
- **$xmodel_dumpvars()**
  - Defines the variables to be dumped
- **$xmodel_dumpon() / $xmodel_dumpoff()**
  - Enables/disables dumping
- **$xmodel_dumpall()**
  - Dumps all the variable values being monitored
- **$xmodel_dumpflush()**
  - Flushes the memory content to the file

$xmodel_dumpfile()

- Defines the name and format of the dump file
- **Usage: $xmodel_dumpfile(*filename*, [*version*])**
  - *filename* : name of the dump file; its extension defines the file format (e.g. ".jez" for JEZ and ".fsdb" for FSDB format)
  - *version* : file format version; currently used only for JEZ format files (e.g. "jezbinary" for binary and "jezascii" for ASCII format)
  - [...] denotes optional arguments
- **Examples**
  - $xmodel_dumpfile("xmodel.jez", "jezascii");
  - $xmodel_dumpfile("xmodel.fsdb");

$xmodel_dumpvars()

- Defines the variables to be monitored and dumped
- **Usage: $xmodel_dumpvars([*option spec*]*, [*module or variable*]*)**
  - *option spec*: can be a string of "*arg=value*" or a pair of arguments (i.e. "*arg=*" and *value*). Multiple argument/value pairs can appear in one string argument using comma separators
  - *modules or variables*: a list of modules or variables of which value-changes are to be monitored. If no modules or variables are given, the current module is assumed
  - * denotes that arbitrary number of arguments can be used

$xmodel_dumpvars() (2)

- **Available options:**
  - ***level=<depth>*** : the level of monitoring depth. For instance, "level=0" means the current level and all lower levels below. "level=1" means only the current level and "level=2" means the current level and one level below.
  - ***type=<vartype1>,<vartype2>, …*** : a comma-separated list of variable types to be monitored. Possible types are *xbit*, *xreal*, *reg*, *wire*, *bit*, *int*, *integer*, and *real*.
  - ***stat=<statistical mode (1 or 0)>*** : a flag to enable/disable statistical data recording; it's used only for JEZ format.
  - ***start=<start time>*** : absolute time (in seconds) to start dumping
  - ***stop=<stop time>*** : absolute time (in seconds) to stop dumping

Examples with $xmodel_dumpvars()

- **$xmodel_dumpvars();** : dumps all the variables in the current scope and below
- **$xmodel_dumpvars("type=xbit,xreal");** : dumps all the xbit and xreal-typed variables in current scope and below
- **$xmodel_dumpvars("level=1", module1);** : dumps only the variables in module1
- **$xmodel_dumpvars("start=10e-9:stop=200e-9", var1, var2, var3);** : dumps var1, var2, var3 from 10ns to 200ns
- **$xmodel_dumpvars("level=", 0, "stat=", 1);** : dumps all the variables in the current scope and below with the statistical recording option on

Miscellaneous Commands

- The following commands take no arguments and have the same functionalities with the corresponding Verilog system calls (e.g. $dumpon, $dumpoff, …)
  - **$xmodel_dumpon()** : enables waveform dumping
  - **$xmodel_dumpoff()** : disables waveform dumping
  - **$xmodel_dumpall()** : dumps all the variables at the current time step
  - **$xmodel_dumpflush()** : flushes the buffer content to file

Measurement & Checker Primitives in *XMODEL*

Scientific Analog, Inc.

November 2016

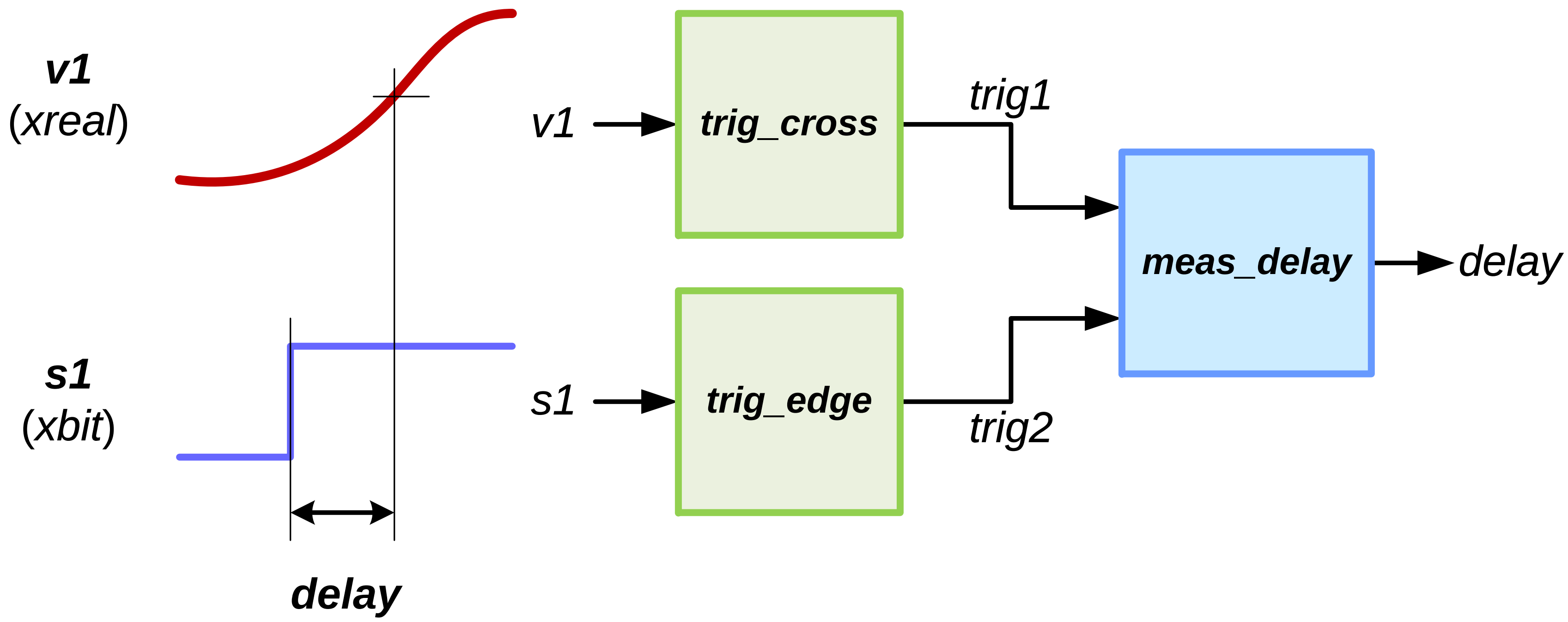
Overview

- The first set of *XMODEL* primitives to measure the waveform characteristics during simulation:
  - Trigger primitives
  - Measurement primitives
  - Checker primitives

- One can compose a variety of measurement/checker statements by putting together these primitives
  - Like MIT's scratch

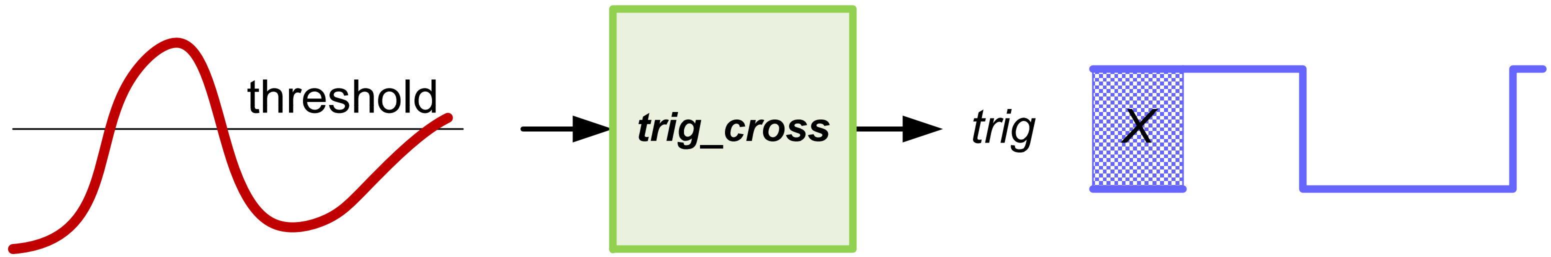
Quick Example

- Measuring the delay from s1's rising to the v1's rising by combining ***trig_**** primitives with ***meas_**** primitives



Trigger Primitives

- Trigger primitives generate a time trigger when a specific event occurs for the input signal
- Trigger is an xbit-typed variable that:
  - Initially "X"
  - Changes to 1 and subsequently toggles between 1 and 0 whenever the event occurs



trig_cross: Trigger for Voltage Crossing

- ***trig_cross*** #(.threshold, .delay, .times) inst (.in, .out);

  : triggers *out* when xreal-typed *in* crosses *threshold*
  N times after *delay*
- Default parameter values:
  - threshold = 0.0
  - delay = 0.0
  - times = 0 (<=0 means whenever)
  - direction = 0 (+1: rising, -1: falling, 0: both)

trig_rise/fall: Trigger for Rising/Falling

- ***trig_rise*** #(.threshold, .delay, .times) inst (.in, .out);
  : triggers *out* when xreal-typed *in* rises above *threshold N* times after *delay*
- ***trig_fall*** #(.threshold, .delay, .times) inst (.in, .out);
  : triggers *out* when xreal-typed *in* falls below *threshold N* times after *delay*

- NOTE: ***trig_rise*** and ***trig_fall*** are equivalent to ***trig_cross*** with the parameter *direction* set to +1 and -1, respectively

trig_edge: Trigger for Bit Transitions

- ***trig_edge*** #(.delay, .times) inst (.in, .out);

  : triggers *out* when xbit-typed *in* has *N*-th transitions after *delay*

- Default parameter values:
  - delay = 0.0
  - times = 0 (<=0 means whenever)
  - direction = 0 (+1: rising, -1: falling, 0: both)

trig_posedge/negedge

- ***trig_posedge*** #(.delay, .times) inst (.in, .out);

  : triggers *out* when xbit-typed *in* has N-th rising transition after *delay*
- ***trig_negedge*** #(.delay, .times) inst (.in, .out);

  : triggers *out* when xbit-typed *in* has N-th falling transition after *delay*
- NOTE: ***trig_posedge*** and ***trig_negedge*** are equivalent to ***trig_edge*** with the parameter *direction* set to +1 and -1, respectively

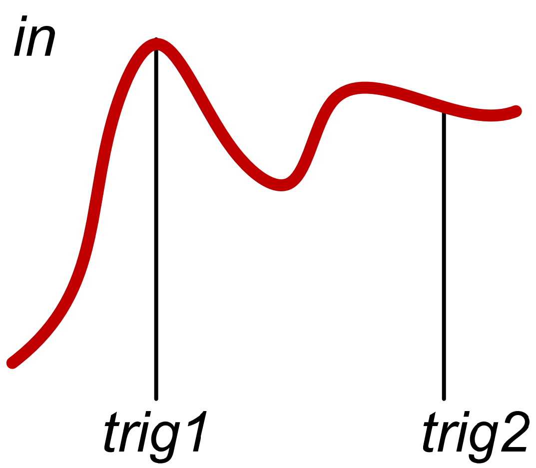
trig_time: Trigger at Specific Times

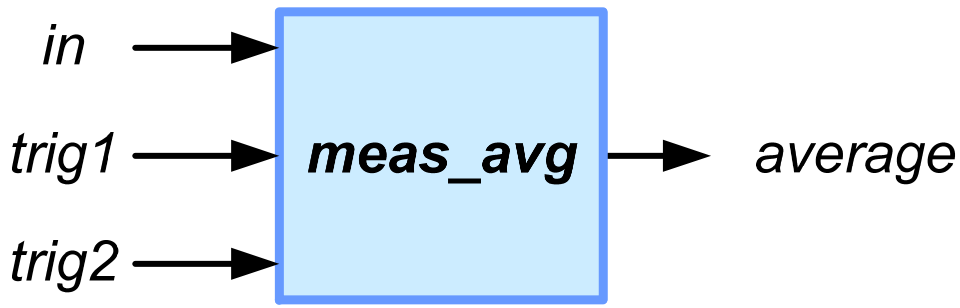
- ***trig_time*** #(.delay, .period) inst (.out);

  : triggers *out* when time = *delay* + N**period*
- Default parameter values:
  - delay = -1.0 (delay < 0 means end of simulation)
  - period = 0 (period <= 0 means no repeating)

Measurement Primitives

- Measurement primitives measure the properties of signals over a time interval indicated by triggers
  - e.g. measuring the average of a signal within $t = [t_1, t_2]$
- The measurement result is a ***real***-typed value





meas_value, slope, deriv

- ***meas_value*** inst(.in, .trig, .out);
  meas_slope inst(.in, .trig, .out);
  meas_deriv #(.order) inst(.in, .trig, .out);
  : measures the value, slope, or N-th derivative of *in* at the time instant indicated by *trig*

- NOTE: ***meas_slope*** is equivalent to ***meas_deriv*** with the parameter *order* set to 1

meas_max, min, avg, integ, pp, rms

- ***meas_max*** inst(.in, .out, .from, .to);
  meas_min inst(.in, .out, .from, .to);
  meas_avg inst(.in, .out, .from, .to);
  meas_integ inst(.in, .out, .from, .to);
  meas_pp inst(.in, .out, .from, .to);
  meas_rms inst(.in, .out, .from, .to);

  : measures the maximum, minimum, average, integral, peak-to-peak, and root-mean-squared (RMS) values of in over a time interval marked by two triggers [*from*, *to*]

meas_time, delay, period

- ***meas_time*** inst(.trig, .out);

  : measures the time instant of the trigger *trig*

- ***meas_delay*** inst(.from, .to, .out);

  : measures the time difference between two triggers *from* and *to*

- ***meas_period*** inst(.trig, .out);

  : measures the time difference between two consecutive trigger events of *trig*

meas_cross, rise, fall

- ***meas_cross*** #(.threshold, .delay, .times) inst (.in, .out);
  meas_rise #(.threshold, .delay, .times) inst (.in, .out);
  meas_fall #(.threshold, .delay, .times) inst (.in, .out);
  : measures time when xreal-typed *in* crosses, rises above, or falls below *threshold N* times after *delay*

- NOTE: these primitives are short-cuts to using ***trig_cross***, ***rise***, ***fall*** with ***meas_time***, *e.g.:*

```
trig_cross #(.threshold, .delay, .times) inst1 (.in(in), .out(trig));
meas_time inst2 (.trig(trig), .out(out));
```

meas_edge, posedge, negedge

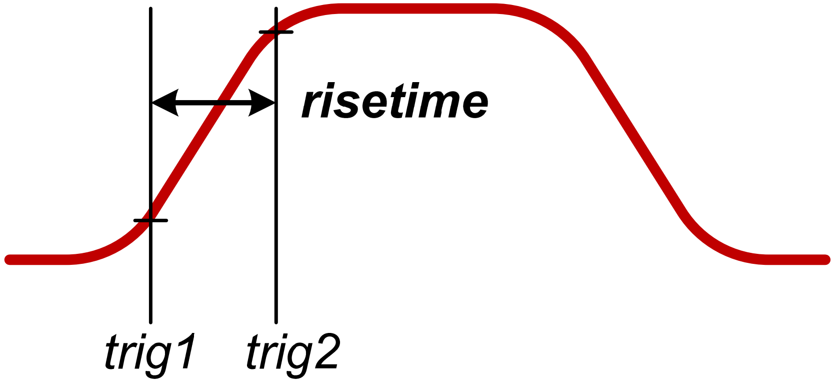
- ***meas_edge*** #(.delay, .times) inst (.in, .out);
  meas_posedge #(.delay, .times) inst (.in, .out);
  meas_negedge #(.delay, .times) inst (.in, .out);
  : measures time when xbit-typed *in* has *N*-th rising, falling, or both transitions after *delay*

- NOTE: these primitives are short-cuts to using ***trig_edge***, ***posedge***, ***negedge*** with ***meas_time***, *e.g.:*

```
trig_edge #(.delay, .times) inst1 (.in(in), .out(trig));
meas_time inst2 (.trig(trig), .out(out));
```

Example: Risetime Measurement

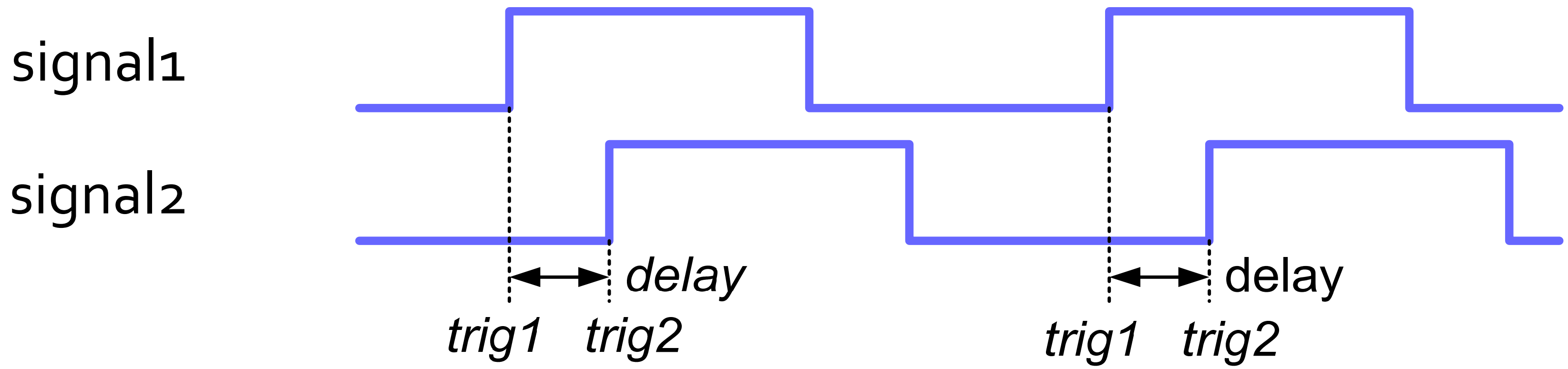
- Measuring the 10-to-90% risetime of a signal




```
xreal signal;
xbit trig1, trig2;
real risetime;

trig_rise #(.threshold(0.1*vdd)) inst1 (.in(signal), .out(trig1));
trig_rise #(.threshold(0.9*vdd)) inst2 (.in(signal), .out(trig2));
meas_delay inst3(.from(trig1), .to(trig2), .out(risetime));
```

Example: Delay Measurement

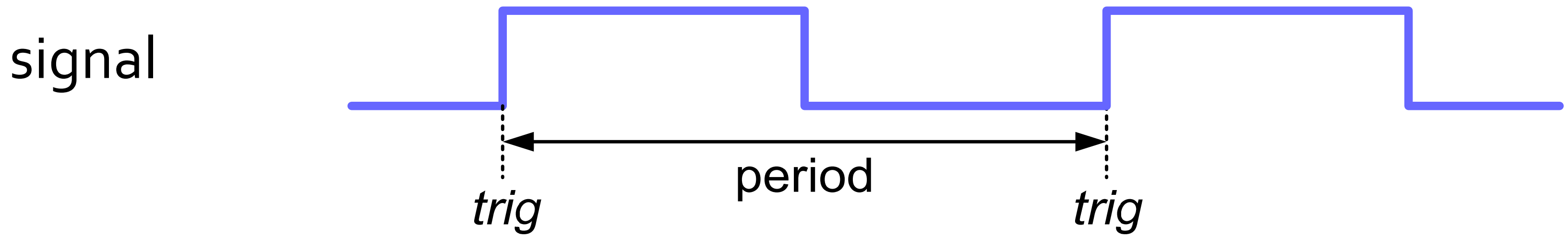
- Measuring the delay between signal1 and signal2:




```
xbit signal1, signal2, trig1, trig2;
real delay;

trig_posedge inst1 (.in(signal1), .out(trig1));
trig_posedge inst2 (.in(signal2), .out(trig2));
meas_delay inst3 (.from(trig1), .to(trig2), .out(delay));
```

Example: Period Measurement

- Measuring the time between two adjacent rising edges




```
xbit signal, trig;
real period;

trig_posedge inst1 (.in(signal), .out(trig));
meas_period inst2 (.in(trig), .out(period));
```