Notes on using Xilinx System Generator

Gregor Dschung

January 18, 2011

1 Installation

The *Xilinx System Generator* is part of the *Xilinx ISE Design Suite* that can be downloaded from `http://www.xilinx.com`. Currently, we are using ISE 12.2. By this time, release 12.3 and 12.4 are available too, but we haven't spent any effort in evaluation for compatibility with Wifire up to now. Licences can be obtained from Robust Network Security Group (RUNSEC).

First install MATLAB and Simulink. The ISE Design Suite Installer detects that installation and offers the installation of the System Generator Blockset. While installing, you have to select an *Edition*. Choose *ISE Design Suite: System Edition*.

2 Using System Generator

- Start a new Simulink model.
- All behaviour that is supposed to be translated into Verilog has to be designed between so called *Gateways*. You are limited to use only blocks from the Xilinx Blocksets—the whole blocks from Simulink or additional Toolboxes / Blocksets can't be used (exceptions are the In/Out-Ports and the Terminator). See Figure 1
- I recommend to use an In-Port for loading data from the workspace into the model while simulation. Furthermore, I recommend to use an fixed-step and discrete solver. That will simplify the debugging with Wave Scopes. The appropriate simulation's configuration parameters are shown in Figure 2 and 3.
- "The System Generator WaveScope block provides a powerful and easy-to-use waveform viewer for analyzing and debugging System Generator designs. The viewer allows you to observe the time-changing values of any wires in the design after the conclusion of the simulation. The signals may be formatted in a logic or analog format and may be viewed in binary, hex, or decimal radices." [System Generator Documentation]
- Every System Generator model needs a System Generator Block. The required settings for the USRP2 are shown in Figure 4.

- The simulation of your design should work with ideal generated signals as well as with recordings.
- The variable `simin` that is configured as the simulation's input in the model's configuration parameters is a structure with the vector `time` and the substructure `signals` composed by the vector `values` and the variable `dimensions`. If you simply want to load a recording (recorded with GNU Radio), something like

  ```
  simin.signals.values = read_complex_binary('../stuff/mitschnitt_25MS_nah.bin');
  simin.signals.dimensions = 1;
  simin.time = 0:1/Fs:(length(simin.signals.values)-1)/Fs;
  simin.time = simin.time.';
  ```

  should be enough, whereby `read_complex_binary` is taken from GNU Radio.
- Use Matlab's Cell Mode.
- Timing Constraints can't be neglegted. A zero-latency pipe through various blocks like adders or multipliers is possible in simulation, but not in hardware. Just generate the ISE Project from the System Generator Block, open the .xise-file with the ISE Project Navigator, synthesize the code and implement the design. This shouldn't take as long as synthesizing and implementing the integration of the own code into the USRP2 ISE Project (this can be generated from the USRP2 FPGA git repository by calling `make proj` in the appropriate directory), but it is mostly sufficient to reveal all broken timing constraints within the own model.
- To exemplify the integration of the model into the USRP2 ISE Project, I refer to `svn:wifire_react:/patches/fpga.patch`.

3 Screenshots

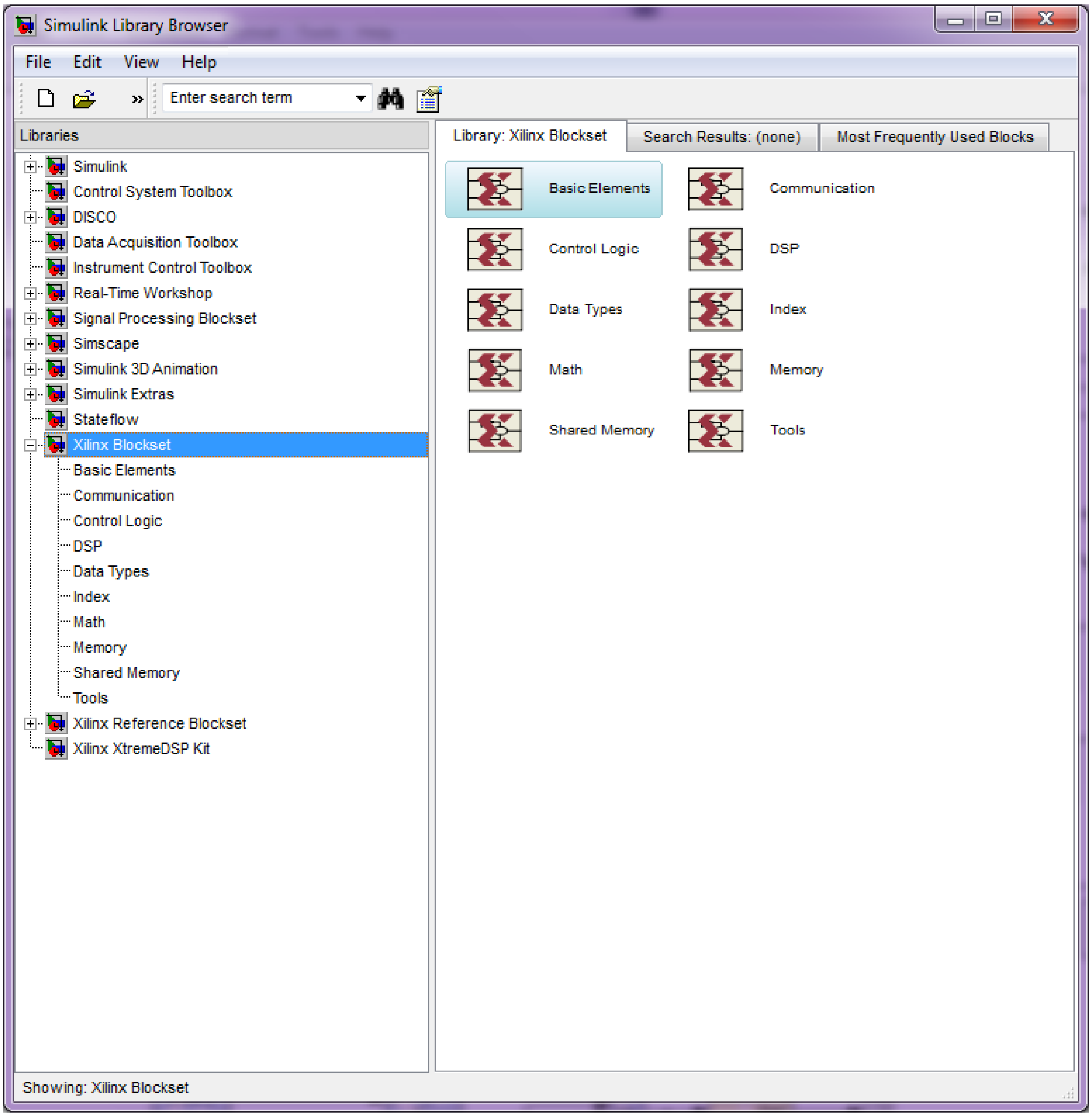
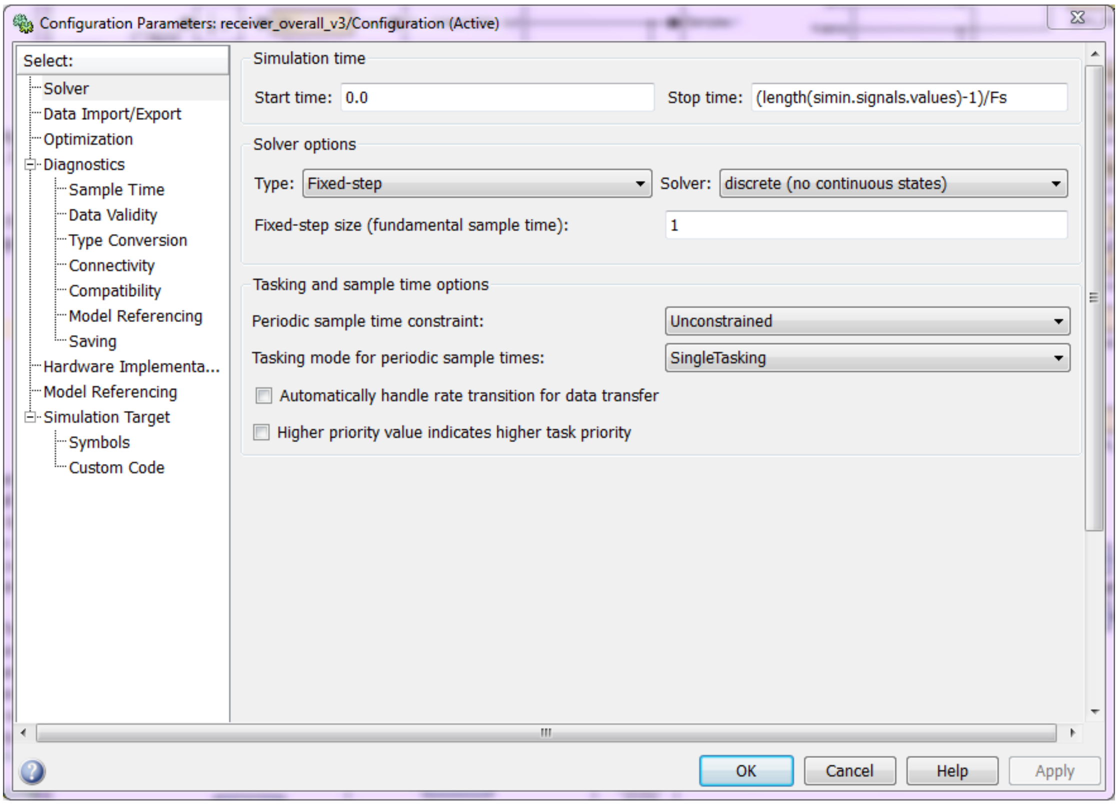


Figure 1: Simulink Library Browser



Configuration Parameters: receiver_overall_v3/Configuration (Active)

Simulation time

Start time: 0.0

Stop time: (length(simin.signals.values)-1)/Fs

Solver options

Type: Fixed-step

Solver: discrete (no continuous states)

Fixed-step size (fundamental sample time): 1

Tasking and sample time options

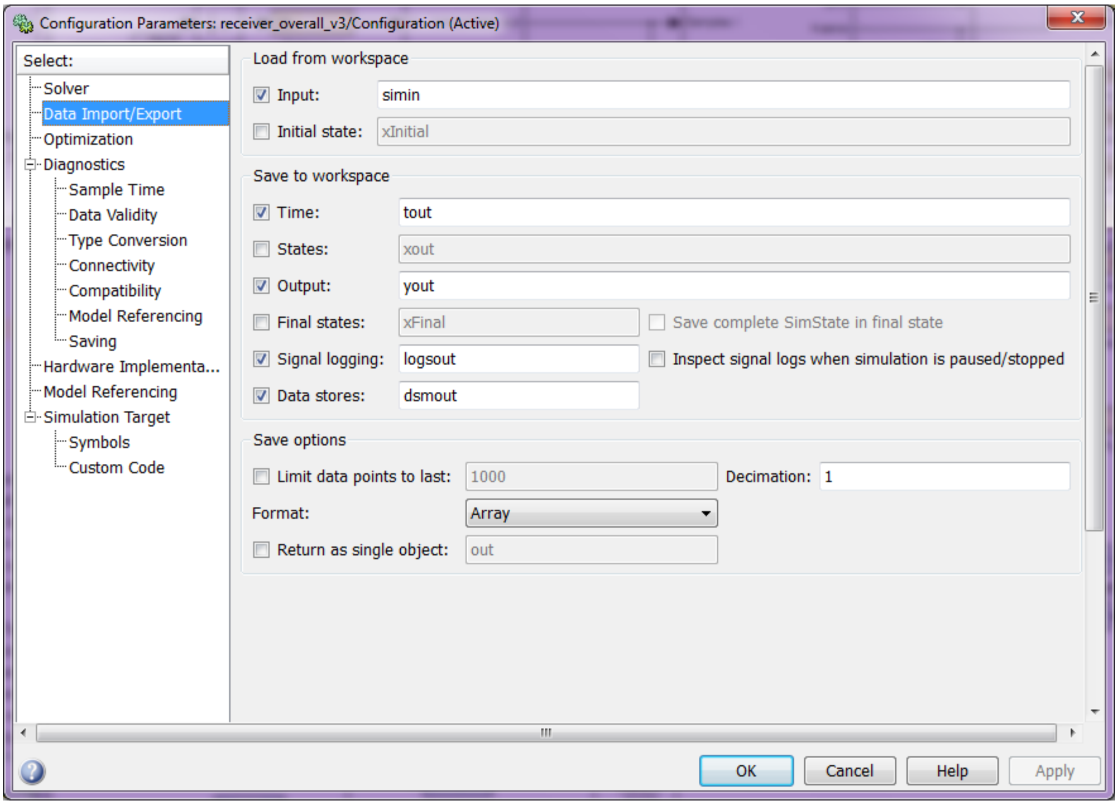
Periodic sample time constraint: Unconstrained

Tasking mode for periodic sample times: SingleTasking

□ Automatically handle rate transition for data transfer

□ Higher priority value indicates higher task priority

Figure 2: Configuration parameters – Solver



Load from workspace

☑ Input: simin

□ Initial state: xInitial

Save to workspace

☑ Time: tout

□ States: xout

☑ Output: yout

□ Final states: xFinal □ Save complete SimState in final state

☑ Signal logging: logsout □ Inspect signal logs when simulation is paused/stopped

☑ Data stores: dsmout

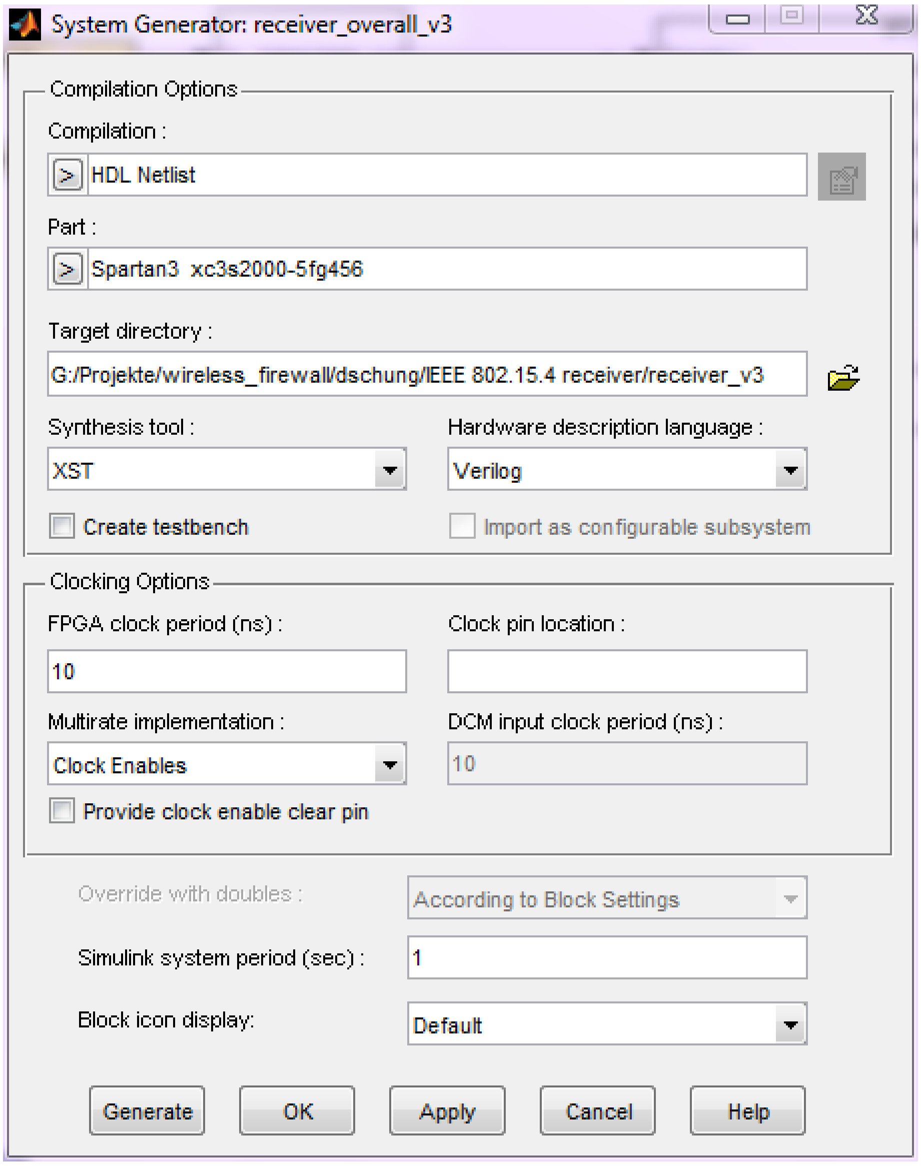
Save options

□ Limit data points to last: 1000 Decimation: 1

Format: Array

□ Return as single object: out

Figure 3: Configuration parameters – Data Import/Export



System Generator: receiver_overall_v3

Compilation Options

Compilation : HDL Netlist

Part : Spartan3 xc3s2000-5fg456

Target directory : G:/Projekte/wireless_firewall/dschung/IEEE 802.15.4 receiver/receiver_v3

Synthesis tool : XST

Hardware description language : Verilog

□ Create testbench

□ Import as configurable subsystem

Clocking Options

FPGA clock period (ns) : 10

Clock pin location :

Multirate implementation : Clock Enables

DCM input clock period (ns) : 10

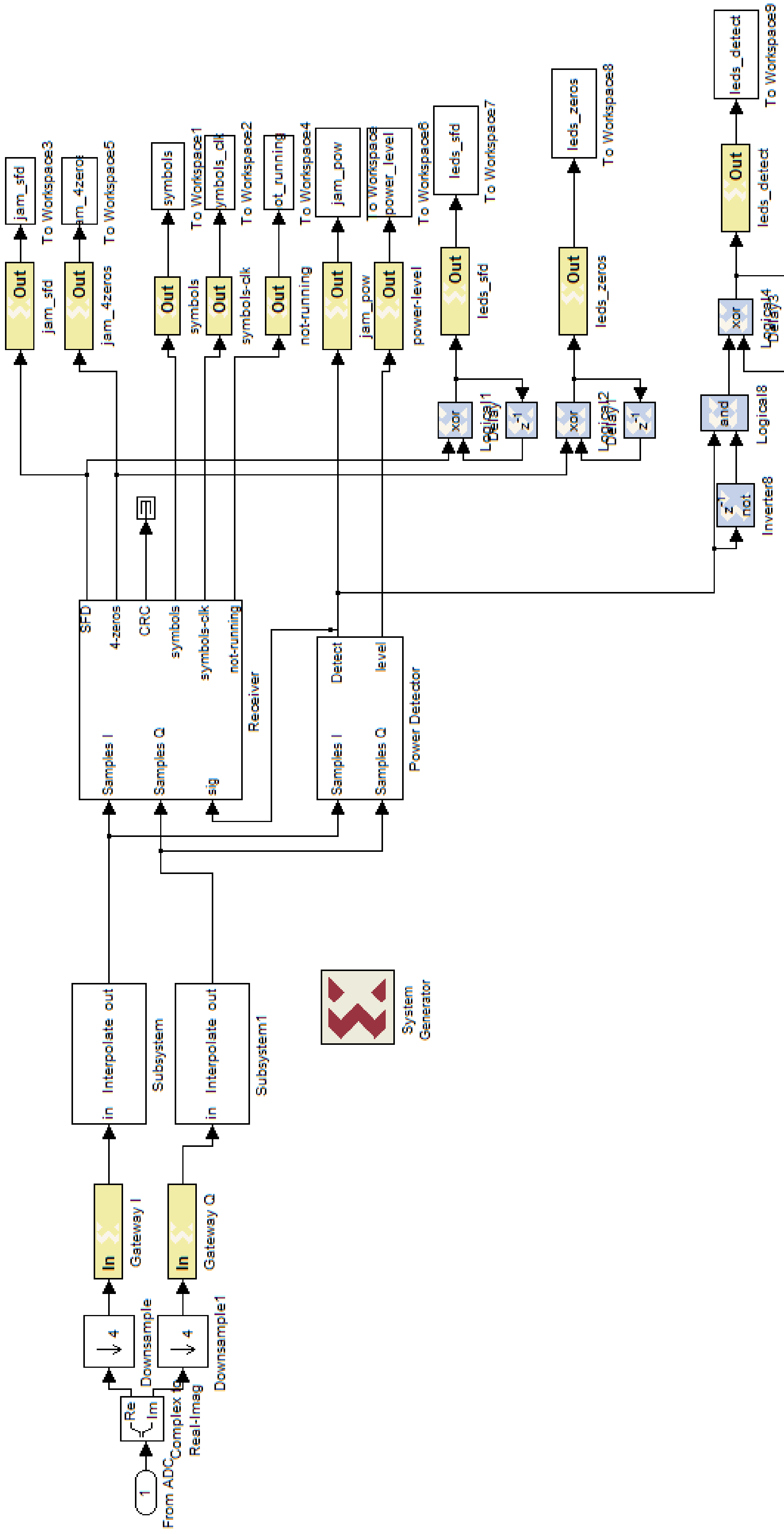
□ Provide clock enable clear pin

Override with doubles : According to Block Settings

Simulink system period (sec) : 1

Block icon display: Default

Figure 4: System Generator


1
From ADC
Re
Im
Complex to
Real-Imag
4
Downsample
4
Downsample1
In
Gateway I
In
Gateway Q
in
Interpolate out
Subsystem
in
Interpolate out
Subsystem1
System
Generator
Samples I
Samples Q
sig
SFD
4-zeros
CRC
symbols
symbols-clk
not-running
Receiver
Samples I
Samples Q
Detect
level
Power Detector
Out
jam_sfd
jam_sfd
To Workspace3
Out
jam_4zeros
jam_4zeros
To Workspace5
Out
symbols
symbols
To Workspace1
Out
symbols-clk
symbols_clk
To Workspace2
Out
not-running
not_running
To Workspace4
Out
jam_pow
jam_pow
To Workspace8
Out
power-level
power_level
To Workspace6
xor
Logical1
z-1
Delay
Out
leds_sfd
leds_sfd
To Workspace7
xor
Logical2
z-1
Delay1
Out
leds_zeros
leds_zeros
To Workspace8
z-1
not
Inverter8
and
Logical8
xor
Logical4
z-1
Delay3
Out
leds_detect
leds_detect
To Workspace9