Multi-Scale, Multi-Fidelity Systems Design and Simulation Environment

Peter Menegay – Director, Funded R&D
pete@syncad.com, 540-953-3390, 540-557-7556
Dan Notestein – President
SynaptiCAD

National Defense Industrial Association (NDIA)
19th Annual Systems Engineering Conference
Oct. 26, 2016

NDIA
National Defense Industrial Association
Promoting National Security Since 1919

SYNAPTICAD

Technology Motivation

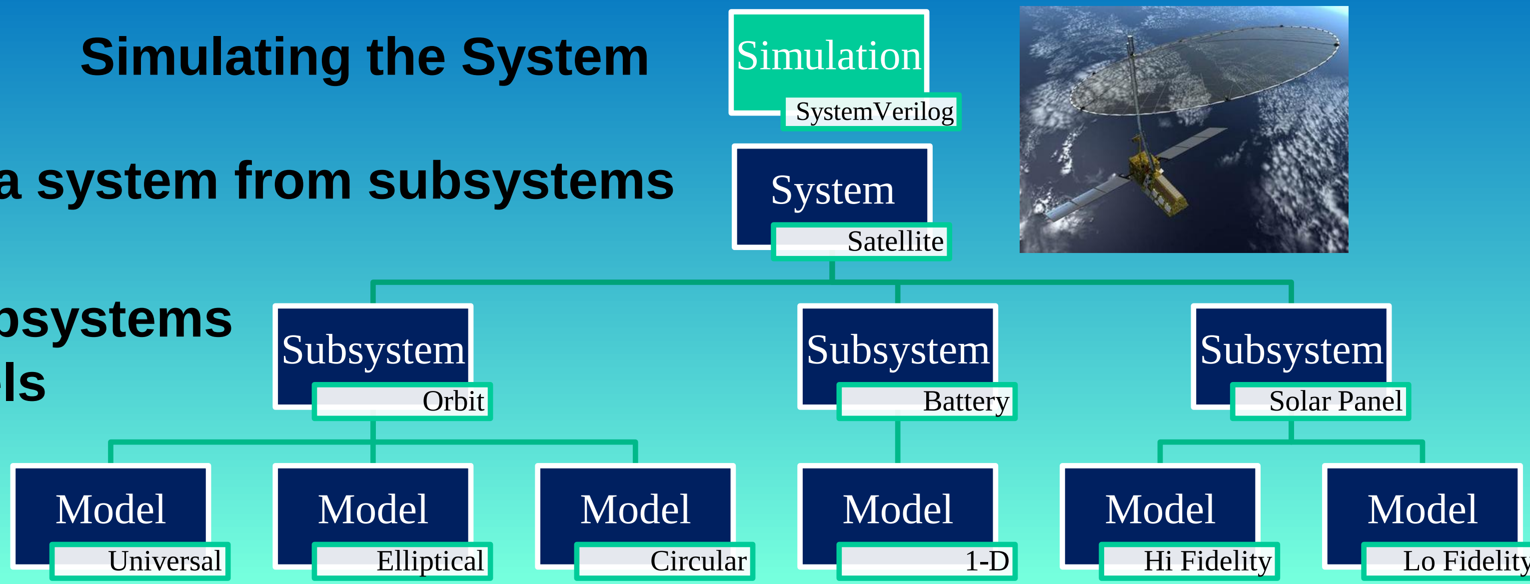
- DoD systems are increasingly complex and challenge human cognitive, and organizational, abilities.
  - Model fidelity and connecting multi-fidelity models to coherent system views.
    - Using the lowest, most appropriate level of fidelity.
  - Engineering model robustness.
  - Reuse of models.
  - Handling highly scaled simulation problems – “digital twin”.
  - Discovering unforeseen behavior.
  - Understanding complex results.
  - Accurate simulations, well before we commit.
  - Simulations & models that live with us throughout the program’s life.

The Basics

Simulating the System

Creating a system from subsystems

Making subsystems from models



Building models from 1st principles

SYNAPTICAD

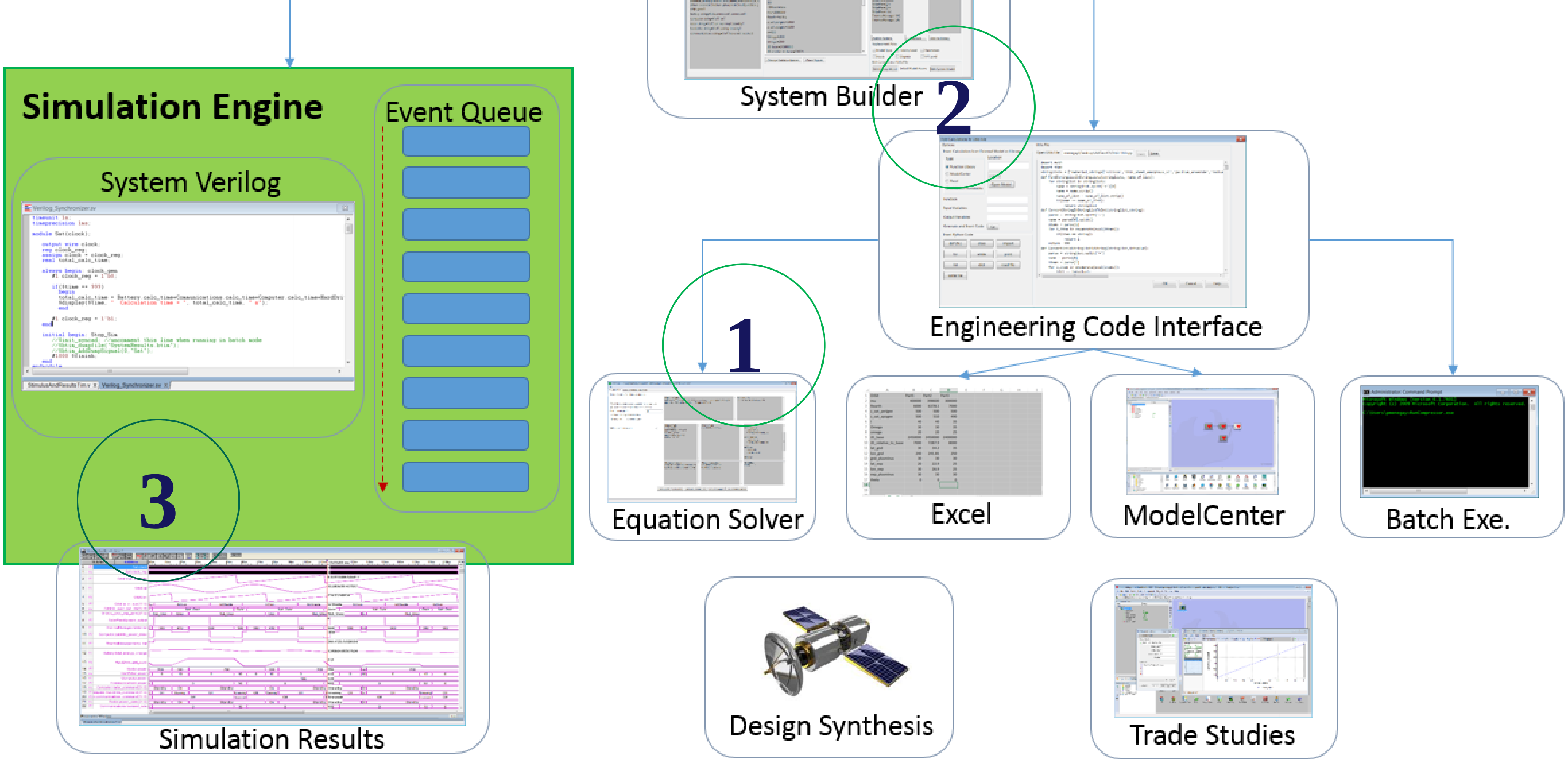
Some Observations

- Most simulation systems are not scalable. But those in the EDA space are. Why not use a paradigm like SystemVerilog to achieve scale?
- Most model-building tools are not generic, and if they are, are not robust enough to be used within simulation. We need to improve the areas where generic numerical methods are weak.
- Most system building tools have trouble handling fidelity transitions as models get more complex. Tools also tend to be domain specific. This leads to bifurcation of modeling efforts.
- System simulation needs to be able to *drive* other tools & models. It also needs to be *drivable* from other tools & models.
- When results from large scale simulations come in, they are often hard to interpret. How can we create views that make sense?

SYNAPTICAD

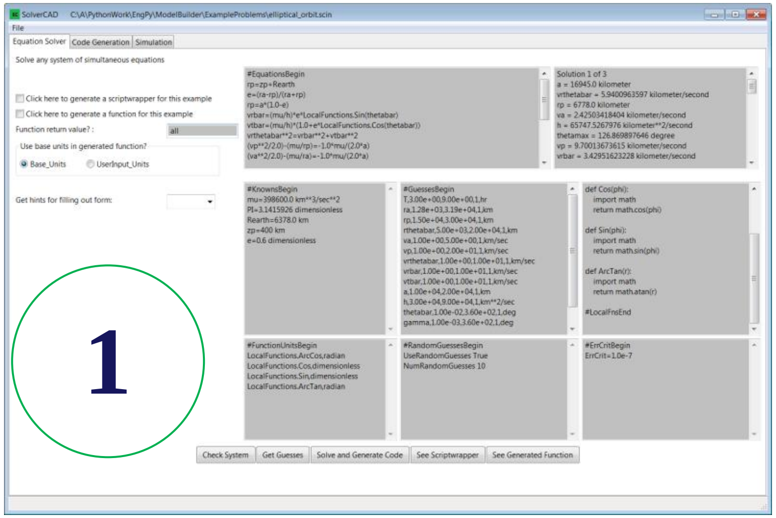
System Computation Platform

Infrastructure for building complex systems for simulation

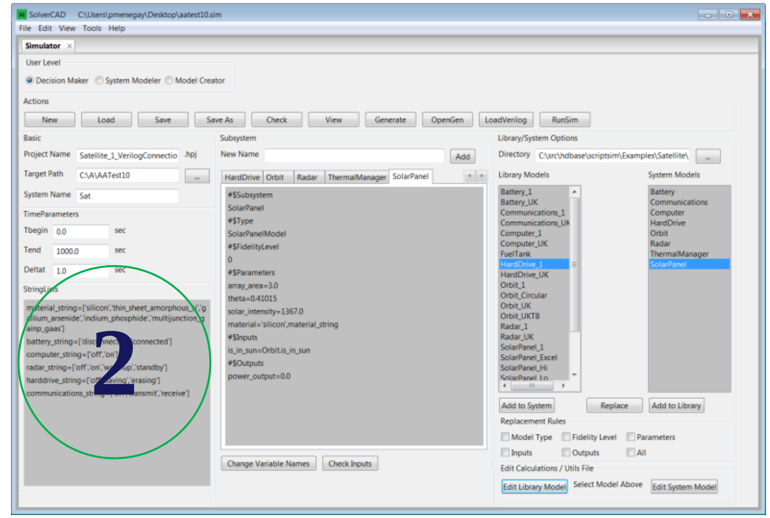


Our approach is to fix weaknesses in the 3 key areas

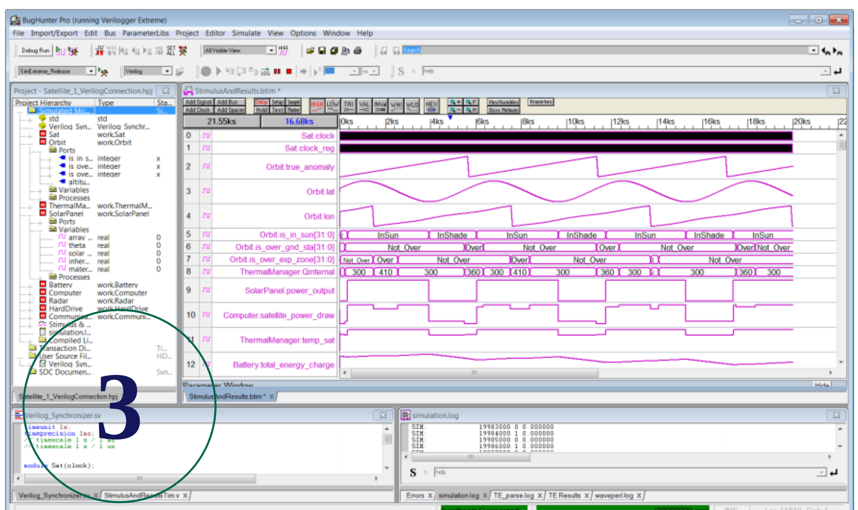
3 Core Elements & Workflow



Eqn. Based Model Creator -- Simultaneous equation solver for creating new engineering models for subsystems.



System Builder -- User interface for creating new system models.

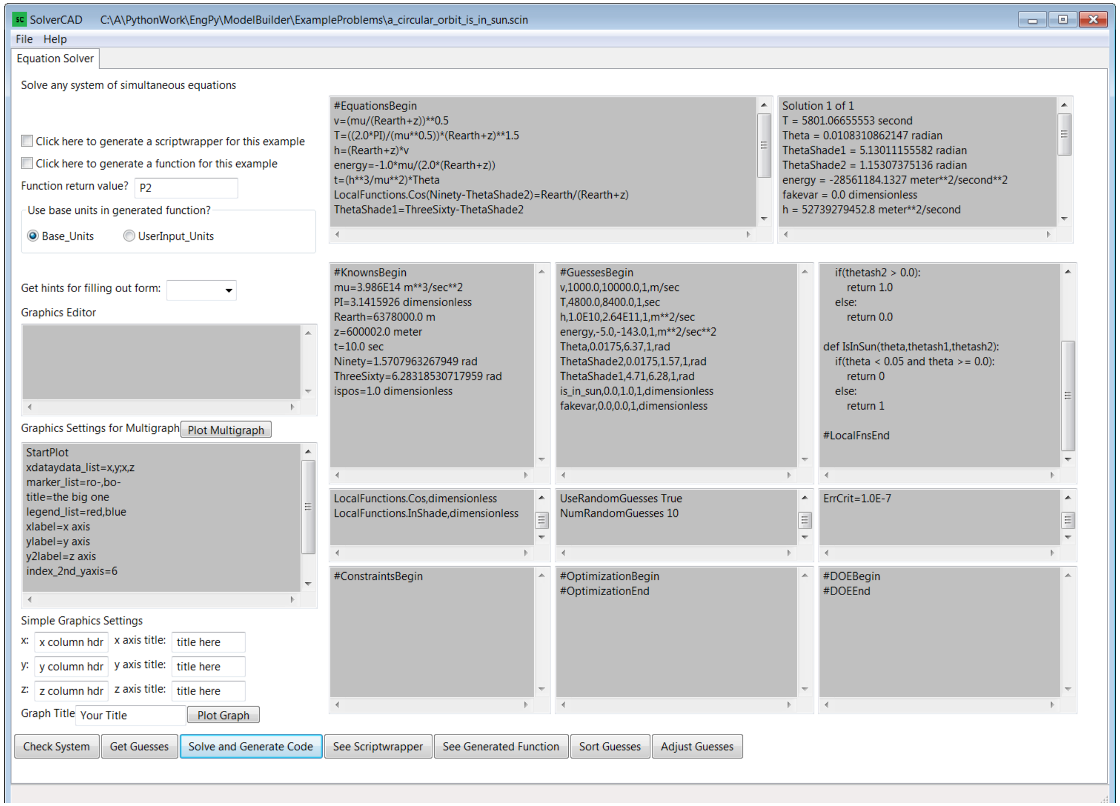


System Simulator -- SystemVerilog simulation engine for system model execution and results.

1. Eqn. Based Model Creator

A general purpose model-creation environment for engineering analysis. Under development.

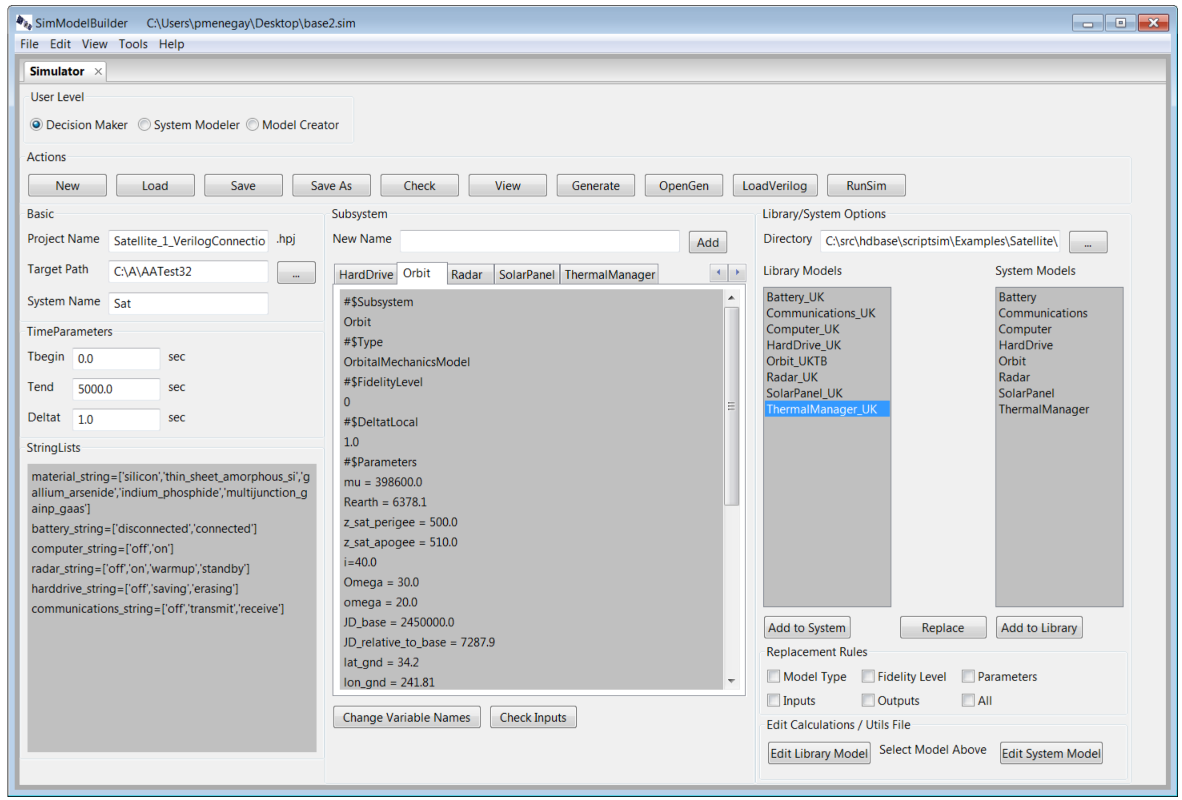
- Solves any system of nonlinear simultaneous equations.
- Manages the core numerical library to achieve robustness.
- Generates Python functions to use in simulation code.
- Uses a library concept for storing functions for later use.



2. System Builder

A GUI that helps you build, connect, and modify sub-system models for simulation.

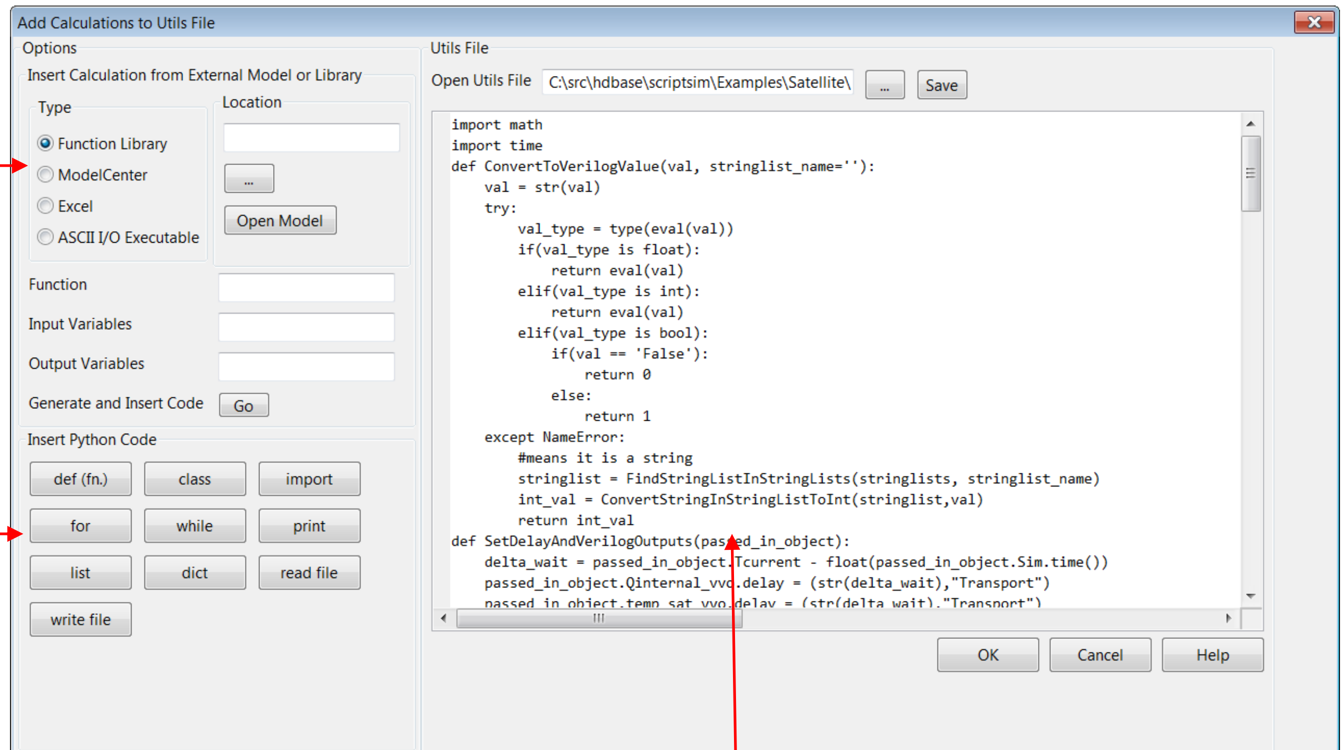
- Create system models from a library of pre-built subsystems.
- Create subsystem models from equation solver and external tools.
- Publish subsystems to library.
- Easily replace subsystems with higher/lower fidelity ones.



SYNAPTICAD

2. System Builder, cont.

Includes a feature to help you edit calculations or link to them from external tools.



Link to pre-built models in SolverCAD, ModelCenter, Excel, or your own program.

Buttons for quickly creating common Python constructs.

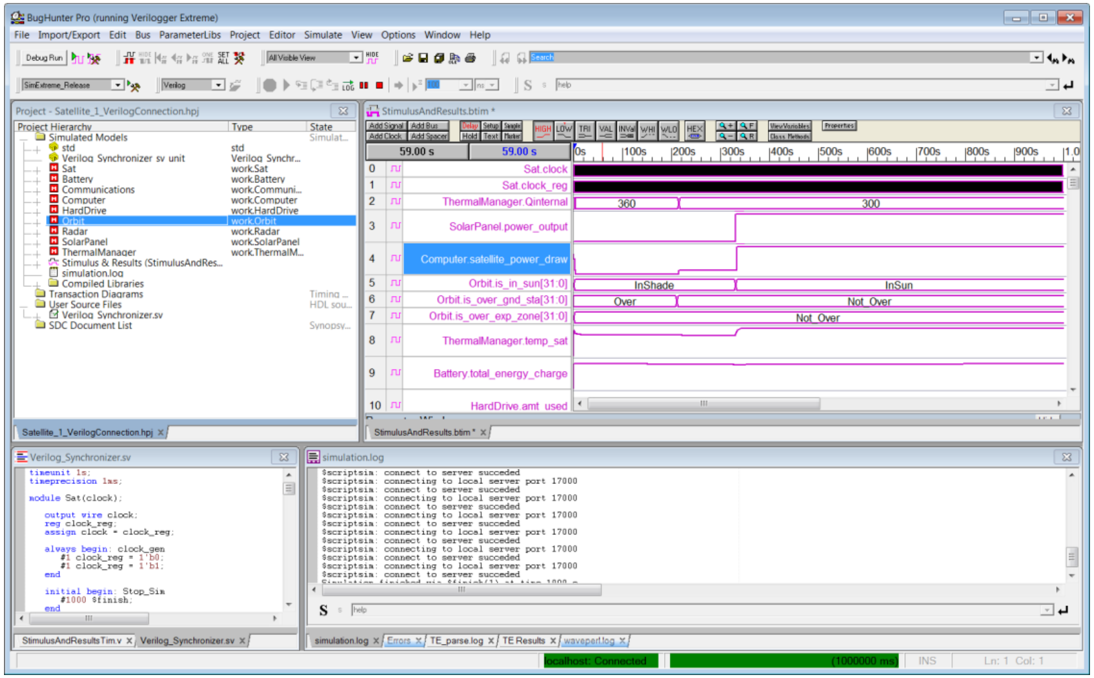
Edit the calculation file for any subsystem in Python.

SYNAPTICAD

3. System Simulator

The system simulator combines a high-performance compiled-code SystemVerilog simulator with a Python interpreter to enable engineering level modeling of real world systems.

- Timing diagram with simulation results
- Generated SystemVerilog code.
- Hierarchical view of subsystems and components.
- Full IDE including single step debugging, breakpoints, etc.
- Design browsing & navigation.
- Various output formats

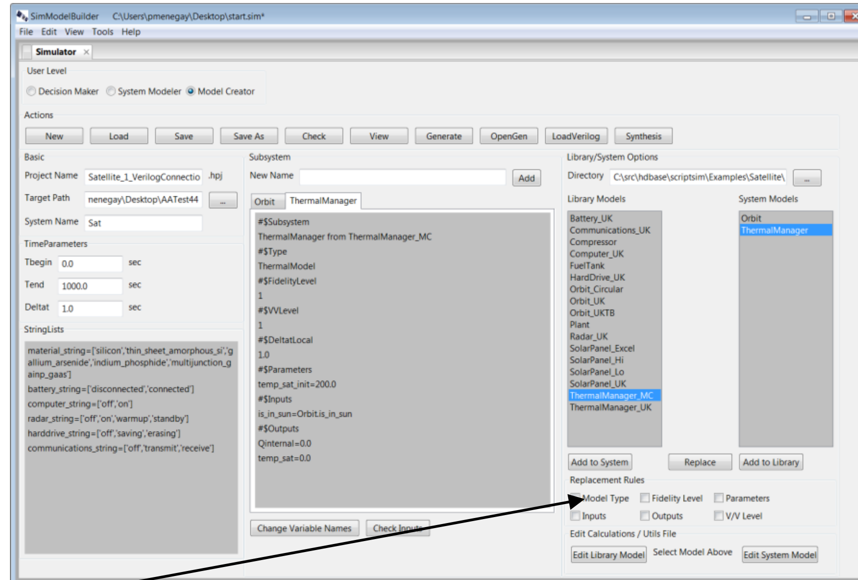


VeriLogger Extreme

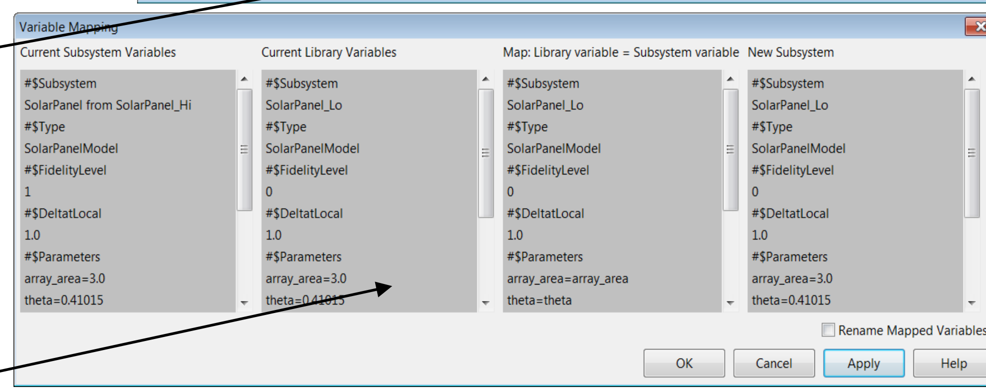
SYNAPTICAD

Multifidelity Modeling

- Models of different fidelity can be switched on the fly.
- As the project advances, the simulation environment remains in place, and maintains connectiveity with previous models.



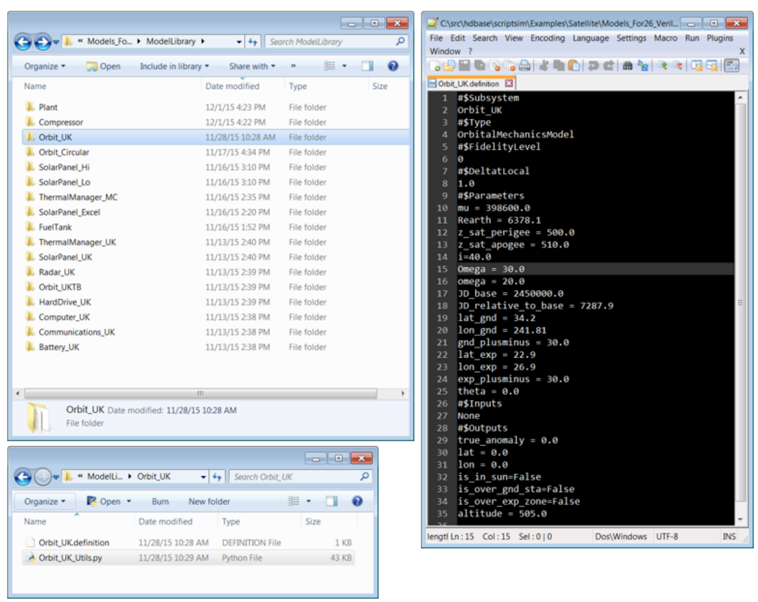
Replacement rules for switching model fidelity



Variable mapping to ensure continuity between models

Model Libraries

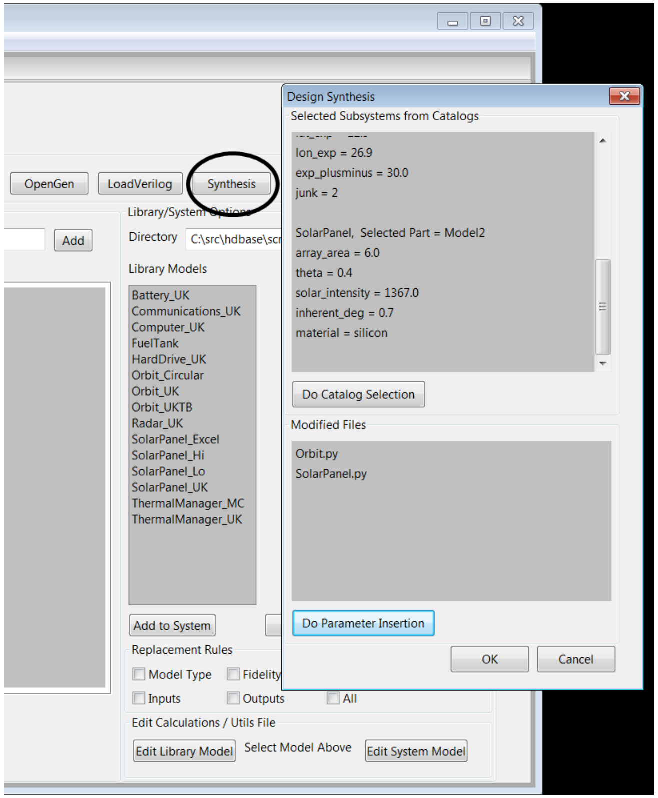
- Subsystem model library.
  - Orbit calculations, solar panel, battery, etc. are publishable and retrievable from library.
- Function library for equation solver.
  - Generated functions can be accessed by System Builder.



SYNAPTICAD

Synthesis

- Compares simulation subsystems with a catalog of parts.
  - User has presumably optimized the subsystem and now wants to select hardware.
  - Software will choose the closest part from catalog and resimulate.



Satellite Model

Satellite circles the earth in a standard elliptical orbit. It's mission is to collect earth data over an experimental zone and download it to a ground station at another location. It charges a battery in the sun and depletes the battery in the shade. The simulation objective is to understand if the subsystems are sized properly.

Orbit
PAR mu=398600.0
PAR Rearth=6378.1
PAR z_sat_perigee=500.0
PAR z_sat_apogee=510.0
PAR i=40.0
PAR Omega=30.0
PAR omega=20.0
PAR JD_base=2450000.0
PAR JD_relative_to_base=7287.9
PAR lat_gnd=34.2
PAR lon_gnd=241.81
PAR gnd_plusminus=30.0
PAR lat_exp=22.9
PAR lon_exp=26.9
PAR exp_plusminus=30.0
PAR theta=0.0
INP None
OUT true_anomaly=0.0
OUT lat=0.0
OUT lon=0.0
OUT is_in_sun=False
OUT is_over_gnd_sta=False
OUT is_over_exp_zone=False
OUT altitude=505.0

Sat
tbegin=0.0
tend=1000.0
deltat=100.0

SolarPanel
PAR array_area=3.0
PAR theta=0.41015
PAR solar_intensity=1367.0
PAR inherent_deg=0.77
PAR material='silicon',material_string
INP is_in_sun=Orbit.is_in_sun
OUT power_output=0.0

Computer
PAR power_rating=100.0
PAR on_state_init='on',computer_string
INP solar_panel_power_output=SolarPanel.power_output
INP radar_power=Radar.power
INP harddrive_power=HardDrive.power
INP comm_power=Communications.power
INP is_over_gnd_sta=Orbit.is_over_gnd_sta
INP is_over_exp_zone=Orbit.is_over_exp_zone
OUT satellite_power_draw=0.0
OUT on_state='off',computer_string
OUT power=0.0
OUT radar_command='off',radar_string
OUT harddrive_command='off',harddrive_string
OUT communications_command='off',communications_string

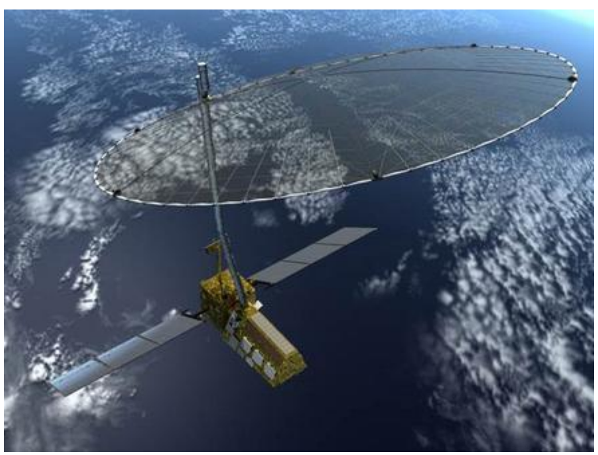
Battery
PAR capacity_rating=360000.0
PAR voltage_rating=12.0
PAR total_energy_charge_init=2160000.0
PAR con_state='connected',battery_string
INP satellite_power_draw=Computer.satellite_power_draw
OUT total_energy_charge=0.0

Radar
PAR power_on=300.0
PAR power_standby=200.0
PAR power_warmup=100.0
PAR power_off=0.0
INP command=Computer.radar_command
OUT power_state='off',radar_string
OUT power=0.0

HardDrive
PAR storage_capacity=650.0
PAR store_speed=0.5
PAR erase_speed=1.0
PAR power_rating=10.0
PAR amt_used_init=300.0
INP command=Computer.harddrive_command
OUT on_state='off',harddrive_string
OUT store_rate=0.0
OUT erase_rate=0.0
OUT power=0.0
OUT amt_used=0.0

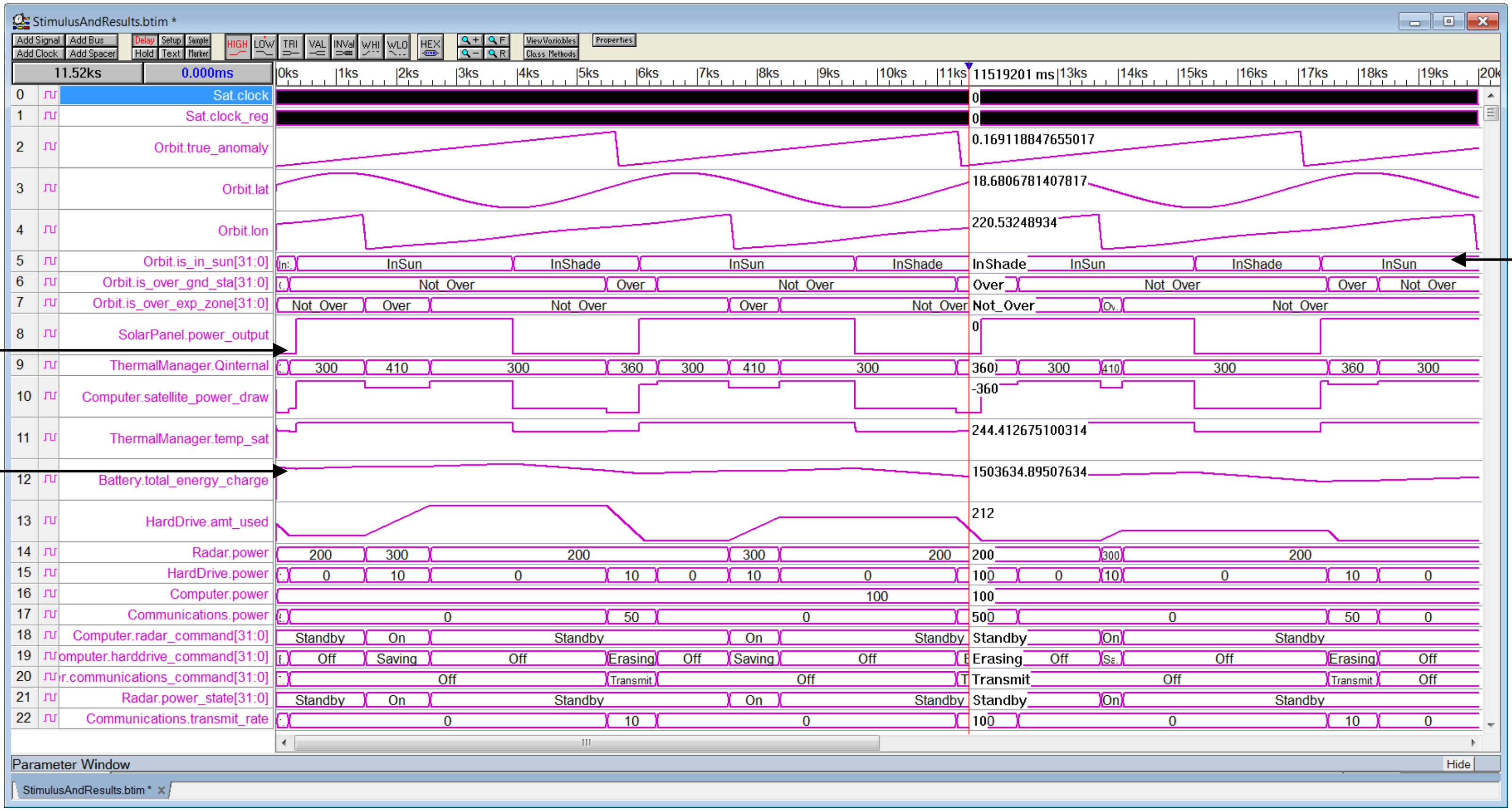
Communications
PAR transmit_speed=10.0
PAR receive_speed=20.0
PAR power_rating=50.0
INP command=Computer.communications_command
OUT on_state='off',communications_string
OUT transmit_rate=0.0
OUT receive_rate=0.0
OUT power=0.0

ThermalManager
PAR earth_albedo=0.3
PAR solar_intensity=1388.0
PAR stefan_boltzmann_const=5.67e-8
PAR temp_earth=290.0
PAR temp_sun=5800.0
PAR temp_space=2.7
PAR radius_earth=6378.1*1000.0
PAR sat_diameter=1.0
PAR sat_absorptivity=0.7
PAR sat_emissivity=0.9
PAR earth_emissivity=0.5
PAR sat_thermal_capacity=50.0
PAR temp_sat_init=300.0
PAR sat_weight=10.0
INP is_in_sun=Orbit.is_in_sun
INP sat_altitude=Orbit.altitude
INP radar_power=Radar.power
INP computer_power=Computer.power
INP harddrive_power=HardDrive.power
INP comm_power=Communications.power
OUT Qinternal=0.0
OUT temp_sat=0.0



NASA / JPL

Results



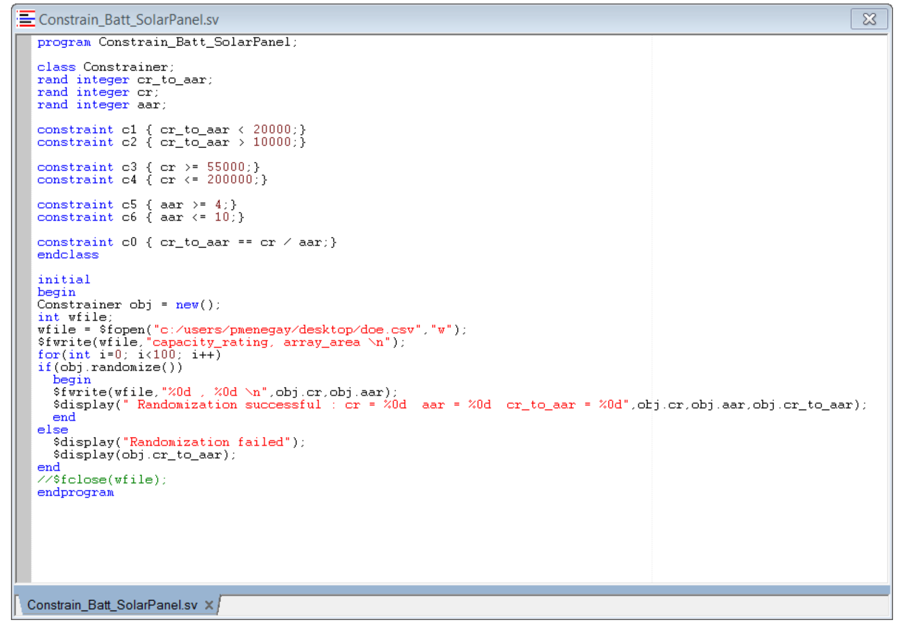
Battery slowly drains to 0

Solar Panel does not recharge it when exposed to sun

Ie, the Solar Panel is undersized. Battery is oversized.

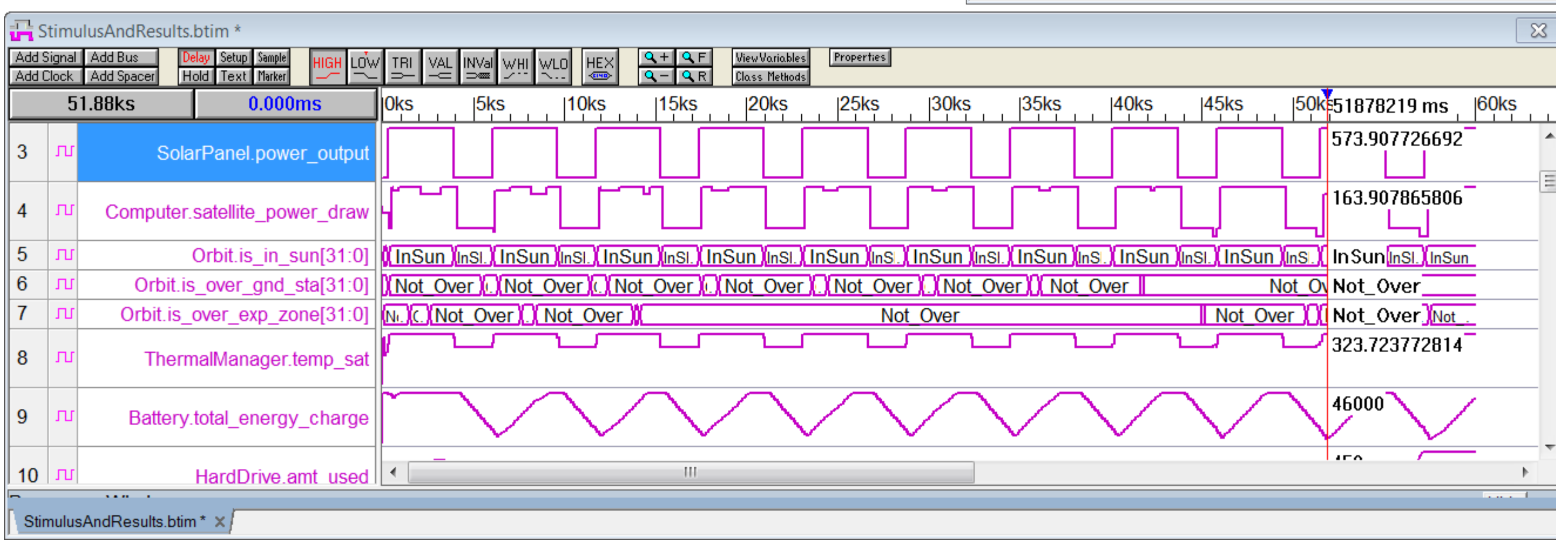
Results, cont.

- One way to vary the solar panel / battery size is to use constrained randomization.
- Solution was to increase the solar panel area from 3.0 to 4.0 m**2 and decrease the battery capacity from 360,000 to 60,000 amp-sec.



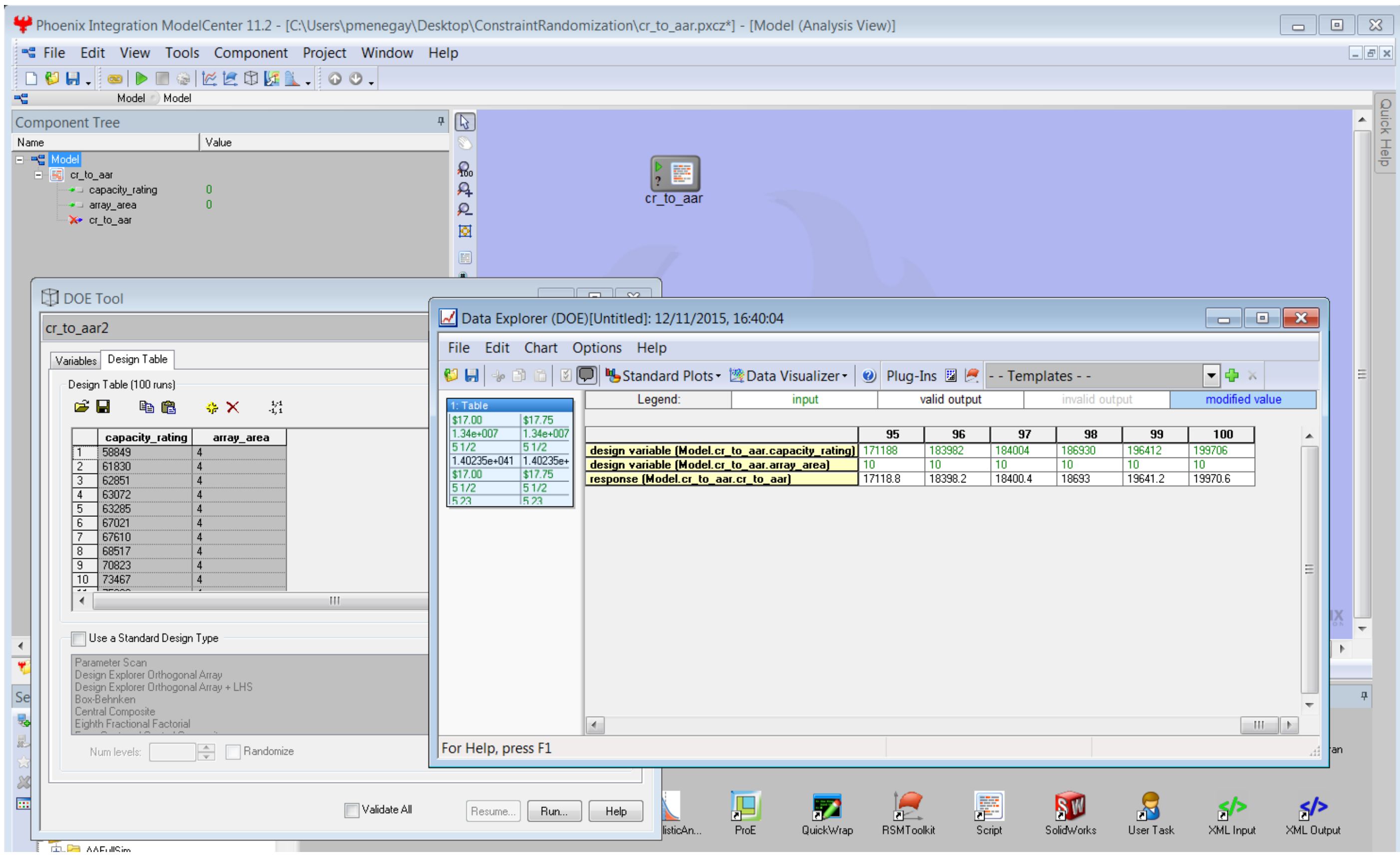
Constrain_Batt_SolarPanel.sv

```
program Constrain_Batt_SolarPanel;

class Constrainer;
rand integer cr_to_aar;
rand integer cr;
rand integer aar;

constraint c1 { cr_to_aar < 20000;}
constraint c2 { cr_to_aar > 10000;}

constraint c3 { cr >= 55000;}
constraint c4 { cr <= 200000;}

constraint c5 { aar >= 4;}
constraint c6 { aar <= 10;}

constraint c0 { cr_to_aar == cr / aar;}
endclass

initial
begin
Constrainer obj = new();
int wfile;
wfile = $fopen("c:/users/pmenegay/desktop/doe.csv","w");
$fwrite(wfile,"capacity_rating, array_area \n");
for(int i=0; i<100; i++)
if(obj.randomize())
  begin
  $fwrite(wfile,"%0d , %0d \n",obj.cr,obj.aar);
  $display(" Randomization successful : cr = %0d  aar = %0d  cr_to_aar = %0d",obj.cr,obj.aar,obj.cr_to_aar);
  end
else
  $display("Randomization failed");
  $display(obj.cr_to_aar);
end
//$fclose(wfile);
endprogram
```



Results, cont.

- This could also have been achieved by driving the simulator from a ModelCenter DOE.



Overall Results

- Once engineering models were made, system integration was fast, 1-2 days for this case.
  - Model libraries were key.
- Provision for multi-fidelity model switching allowed project to remain within a single environment throughout its life.
- Scalability tests on a simple vehicle object lends credence to the SystemVerilog approach.
  - SystemVerilog can simulate up to memory limits of computer. 18 million vehicles for 32-bit and 40 million for 64 bit.
  - SimPy by contrast could simulate 900,000 such objects.
- Runs could be made faster by using event-driven simulation. A 10 fold speed up was achieved this way.
  - Important for long run times over the life of the system.

We thank the NASA SBIR program for sponsoring this work.
Contract NNX15CP26P.

SYNAPTICAD