This is a power conversion element that interfaces a nonlinear time domain model, in the Hammerstein-Weiner (HW) framework, to the OpenDSS dynamics mode. The model senses terminal voltage and updates the injected current at each OpenDSS time step. Figure 1 shows the relationship between HW and OpenDSS domains, and it defines the key signal locations to reference when reading the Delphi code. In a snapshot mode, the model injected current will be constant, as determined by its ratings and initial power output level. The initial application will be for photovoltaic inverter modeling, and the parameters are defined in those terms.



Figure : Interface between HW and OpenDSS solution variables

Input parameters are:

* Basefreq: as other PC elements (assumed 60 in Figure 1)
* BP1: XYCurve defining the input piecewise linear block in Figure 1
* BP2: XYCurve defining the output piecewise linear block in Figure 1
* Bus1: as other PC elements
* Enabled: as other PC elements
* Filter: XYCurve defining the infinite impulse response (IIR) digital filter coefficients in Figure 1. X values are the numerator coefficients and Y values are the denominator coefficients, and note that a0=1 must be input explicitly. This also enforces equal order in numerator and denominator.
* Fsample: sampling rate for H(z), either 5 kHz (PortoSag) or 10 kHz (Dranetz) in this project.
* Like: as other PC elements
* Phases: as other PC elements, but only 1-phase implemented now. 3-phase under development.
* Ppct: real power output in snapshot mode, based on Prated. Unity power factor assumed.
* Prated: inverter’s rated power in Watts
* Spectrum: as other PC elements, but not implemented here.
* Vrated: rated line-to-line voltage (3-phase) or line-to-neutral voltage (1-phase) in volts

## HWTest.DSS

This file illustrates two inverter models in a snapshot solution. The inverters are connected to a 360-volt source, which matches the lab tests done at 208 volts. The short circuit current supplied by the source is 7354 amps. The VCCS component works in snapshot mode, but not yet in dynamics mode, when phases=3. BP1, BP2, Filter and Fsample have no effect in snapshot mode. The initial currents will be  and 

Clear

new circuit.HWTest

~ basekv=0.360 pu=1.0 angle=0 phases=3 bus1=SourceBus r1=0.02 r0=0.02 x1=0.02 x0=0.02

redirect xy\_data.txt

New VCCS.SMA1 Phases=1 Bus1=SourceBus.1 Prated=3000 Vrated=208

~ Ppct=100 bp1='bp1' bp2='bp2' filter='coeffs' fsample=10000

New VCCS.SMA3 Phases=3 Bus1=SourceBus Prated=3000 Vrated=360

~ Ppct= 50 bp1='bp1' bp2='bp2' filter='coeffs' fsample=10000

Set Voltagebases=[0.360]

set maxiterations=100

calcv

Solve

## HWDynTest.dss

The file that runs a dynamics test has been set up to run either the SMA 3-kW or the Enphase M190 inverter, feeding a bolted fault with source impedance adjusted to match that of Pitt’s electric power lab. The fault’s ontime value will determine when the HW model departs from its initial condition. In this case, Fsample is 10 kHz, so the OpenDSS time step must be at least 0.1 ms, but it can be longer. The HW model can run at a faster time step within each OpenDSS time step.

Clear

// EPSL impedance is 10% on 75 kVA positive sequence, 5% zero sequence, assume X/R = 2

new circuit.HWDynamic

~ basekv=0.360 pu=1.0 angle=0 phases=3 bus1=SourceBus r1=0.029 x1=0.058 r0=0.014 x0=0.029

redirect xy\_data.txt

redirect Enphase\_M190.txt

New VCCS.SMA Phases=1 Bus1=SourceBus.1 Prated=3000 Vrated=208 Ppct=100

~ bp1='bp1' bp2='bp2' filter='coeffs' fsample=10000

// New VCCS.SMA Phases=1 Bus1=SourceBus.1 Prated=190 Vrated=208 Ppct=89.5

// ~bp1='m190\_bp1' bp2='m190\_bp2' filter='m190\_z' fsample=10000

new fault.flt bus1=Sourcebus.1 phases=1 r=0.001 temporary=yes ontime=0.0821 // ontime=0.0768

new monitor.invvi element=vccs.sma terminal=1 mode=0

new monitor.invpq element=vccs.sma terminal=1 mode=1

new monitor.invst element=vccs.sma terminal=1 mode=3

new monitor.fltvi element=fault.flt terminal=1 mode=0

Set Voltagebases=[0.360]

set maxiterations=100

calcv

set mode=snap

solve

set mode=dynamic

set stepsize=0.0002

set number=1250

//set stepsize=0.0001 // matches fsample

//set number=2500

Solve

export monitors invst

## Enphase\_M190.txt

This included file defines BP1, BP2, and a 4th-order filter for the M190. The SMA included file is formatted the same way, but it has a 22-order filter, which means it’s transient response will last longer.

// library model for M190

New XYcurve.m190\_bp1 npts=10 xarray=[-264.247209691872 -186.797096749935 -113.999158872285 -82.6870353867463 -24.8137408902437 -11.9148284369882 27.0192252128813 135.149825825011 196.543272046783 241.739830586434]

~ yarray=[-56.6007772352605 18.4686074648929 90.5436353349481 121.436176790441 177.882600484537 191.555600465171 231.689431511064 332.957826097434 384.696875082151 420.213174126393]

New XYcurve.m190\_bp2 npts=10 xarray=[-185.670885956717 -162.398919495745 -139.126953016290 -115.852878563658 -92.5758885674808 -47.0112883890579 -35.3480971557727 -27.0710140718451 13.4654018280332 23.7955752913491]

~ yarray=[39.9993291871476 18.6092700884592 0.271831180935148 0.476925527659269 15.4575150026444 14.1667309380808 7.00879990229796 3.58373421812038 -11.3765765505691 -38.8152025782624]

New XYcurve.m190\_z npts=4 xarray=[1 -0.275515728217058 -0.0975731084864074 0.0275483933466857] // denominator

~ yarray=[0 -0.198600301682772 1 -0.856652694252797] // numerator

## Example BP1 and BP2

Sample non-linear blocks for the SMA 3000 are shown in Figure 2. They do not necessarily display appropriate symmetry and offset characteristics; work is ongoing to improve this. Note that BP2 decreases with X, which required a bug fix to the XYCurve object in OpenDSS for reverse lookup. It’s possible that InvControl is also affected by this fix.

Figure : Sample 10-point nonlinear blocks BP1 and BP2

## SMA Sample Outputs

Figures 3 and 4 show the SMA model running at OpenDSS time steps of 0.1 to 10 ms. Current waveforms are in black, RMS in red and peak in blue. The model initialization is smooth, but steady state is not well established. In fact, the response is acausal; the breakpoint block data needs to be refined.

When OpenDSS runs at a “slow” time step, the HW model runs internally at the “fast” filter time step. However, the output is only sampled at the “slow” time step. The OpenDSS current injections are only updated at the “slow” time step.

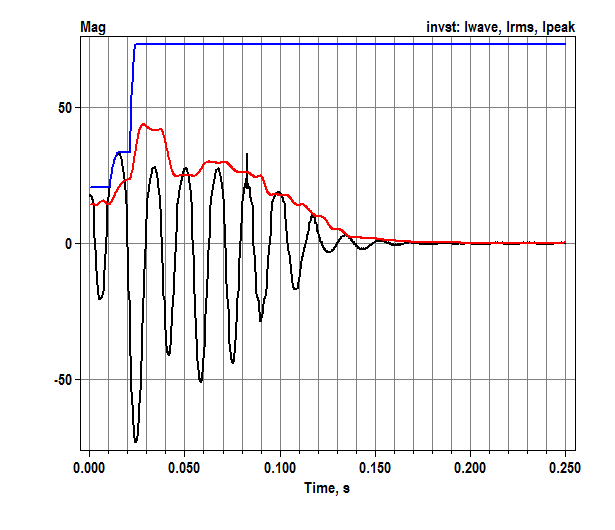
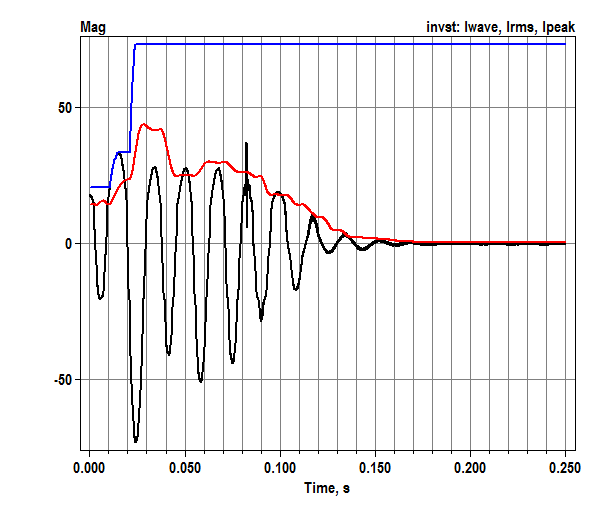


Figure 3: Current waveform, RMS and peak state variables for SMA 3000. h=0.1 ms (left) and 1 ms (right)

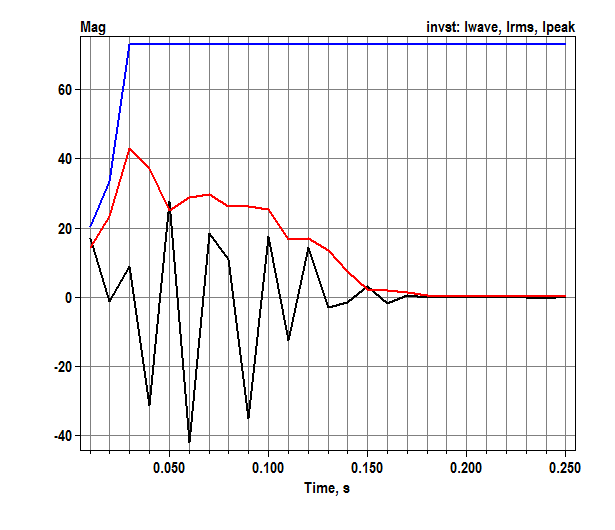
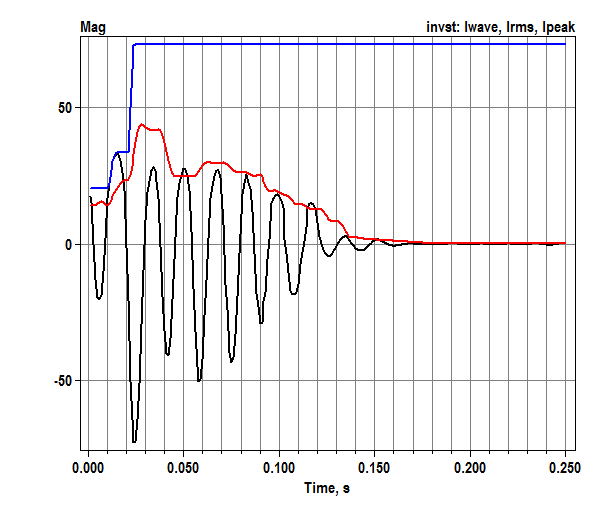


Figure : Current waveform, RMS and peak state variables for SMA 3000. h=2 ms (left) and 10 ms (right)

## Enphase Sample Outputs

Figure 5 shows an M190 response to the fault. With a 4th-order filter at 10 kHz sampling rate, the transient response is much shorter in duration. Initialization is smooth, but incorrect, because the BP1 output was scaled by 10. The model captures a transient current peak that loads the inverter terminal voltage (Figure 5 right). When OpenDSS runs at a step longer than 0.1 ms, this loading effect is not captured because the RMS function smooths the current peak. Therefore, the peak current (Figure 5 left) is reduced when the OpenDSS time step increases. This loading effect on peak current does not occur in Figures 3 and 4.

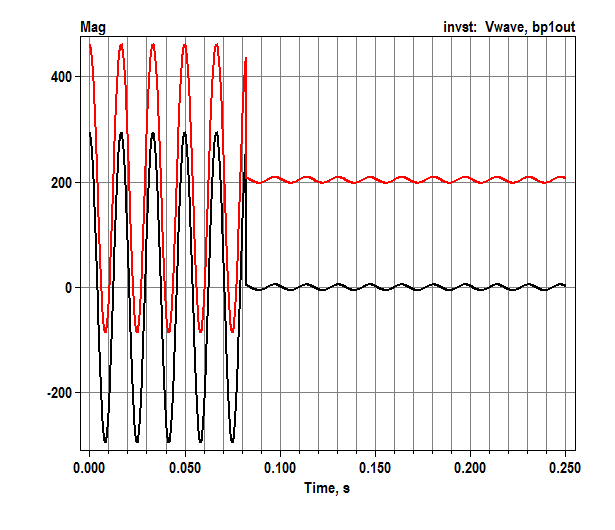
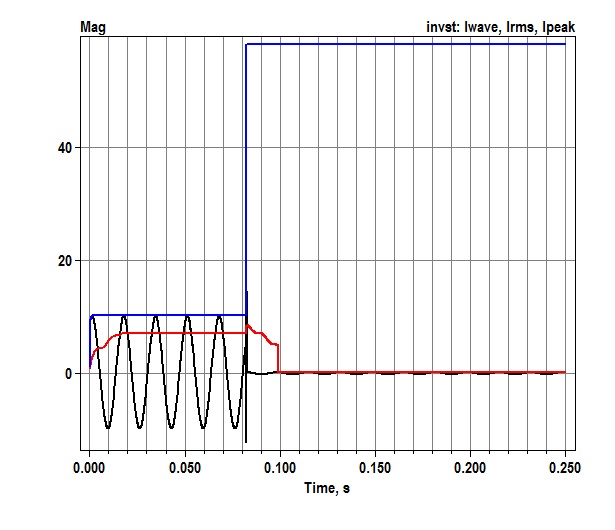


Figure : Enphase M190 current state variables (left) and voltage state variables (right) at h=0.1 ms.

## Code Listings

The most significant parts of the VCCS code define, initialize and integrate the state variables. Listings follow.

Evaluating H(z) in Figure 1 requires history storage for both input and output. A second copy of both history terms is required for the predictor and corrector steps. Otherwise, when the OpenDSS and HW steps are unequal, the last step’s history could be corrupted by a partial update before the corrector step initiates. The history terms are z, whist, zlast and wlast, allocated to a size determined from the base and sampling frequencies. A ring buffer indexing scheme is used to manage this storage, with helper functions MapIdx and OffsetIdx.

The overloaded InitStateVars procedure extracts terminal voltage and current from the most recent snapshot solution. Then it steps back in time at the sampling frequency to initialize the history terms. The history of z is determined from the history of current, i, passed in reverse through BP2. The HW model current uses generator convention while the OpenDSS current uses passive convention, leading to a negative sign on the initial values of z.

The overloaded IntegrateStates procedure is called twice in each OpenDSS time step, first as predictor and then as corrector. Each time, it extracts terminal voltage and ensures that whist and z begin afresh from the previous OpenDSS time step. Then, it runs through the BP1, z, and BP2 evaluations defined in Figure 1 for the number of sampling steps contained in an OpenDSS time step. RMS current is calculated “brute force” at the end of the OpenDSS time step, for both predictor and corrector. This value, sIrms, is later injected into the grid via the overloaded procedure GetInjCurrent (not listed below). Note that the current angle updates are not yet implemented. The peak current state variable is checked every HW step, in order to catch “fast” peaks that might not appear in monitor output. The other state variables, and the wlast / zlast history terms, are only updated at each OpenDSS corrector time step.

**private**

Fbp1: TXYcurveObj;

Fbp1\_name: String;

Fbp2: TXYcurveObj;

Fbp2\_name: String;

Ffilter: TXYcurveObj;

Ffilter\_name: String;

BaseCurr: double; // line current at Ppct

FsampleFreq: double; // discretization frequency for Z filter

Fwinlen: integer;

Ffiltlen: integer;

// Support for Dynamics Mode

sVwave: double;

sIwave: double;

sIrms: double;

sIpeak: double;

sBP1out: double;

sFilterout: double;

vlast: complex;

y2: pDoubleArray;

z: pDoubleArray; // current digital filter history terms

whist: pDoubleArray;

zlast: pDoubleArray; // update only after the corrector step

wlast: pDoubleArray;

sIdxU: integer; // ring buffer index for z and whist

sIdxY: integer; // ring buffer index for y2 (rms current)

y2sum: double;

// helper functions for ring buffer indexing, 1..len

**function MapIdx(idx, len: integer):integer;**

begin

while idx <= 0 do idx := idx + len;

Result := idx mod (len + 1);

if Result = 0 then Result := 1;

end;

**function OffsetIdx(idx, offset, len: integer):integer;**

begin

Result := MapIdx(idx+offset, len);

end;

// support for DYNAMICMODE

// NB: The test data and HW model used source convention (I and V in phase)

// However, OpenDSS uses the load convention

**procedure TVCCSObj.InitStateVars;**

var

d, wt, wd, val, iang, vang, pk: double;

i, k: integer;

begin

// initialize outputs from the terminal conditions

ComputeIterminal;

iang := cang(Iterminal^[1]);

vang := cang(Vterminal^[1]);

pk := sqrt(2);

sVwave := cabs(Vterminal^[1]) \* pk;

sIrms := cabs(Iterminal^[1]);

sIwave := sIrms \* pk;

sIpeak := sIrms \* pk;

sBP1out := 0;

sFilterout := 0;

vlast := Vterminal^[1];

// initialize the history terms for HW model source convention

d := 1 / FsampleFreq;

wd := 2 \* Pi \* ActiveSolutionObj.Frequency \* d;

for i := 1 to Ffiltlen do begin

wt := vang - wd \* (Ffiltlen - i);

whist[i] := 0;

whist[i] := Fbp1.GetYValue(sVwave \* cos(wt));

wlast[i] := whist[i];

end;

for i := 1 to Fwinlen do begin

wt := iang - wd \* (Fwinlen - i);

val := pk \* sIrms \* cos(wt); // current by passive sign convention

y2[i] := val \* val;

k := i - Fwinlen + Ffiltlen;

if k > 0 then begin

z[k] := -Fbp2.GetXvalue (val); // HW history with generator convention

zlast[k] := z[k];

end;

end;

// initialize the ring buffer indices; these increment by 1 before actual use

sIdxU := 0;

sIdxY := 0;

end;

// this is called twice per dynamic time step; predictor then corrector

**procedure TVCCSObj.IntegrateStates;**

var

t, h, d, f, w, wt, pk: double;

vre, vim, vin, scale, y: double;

nstep, i, k, corrector: integer;

vnow: complex;

iu, iy: integer; // local copies of sIdxU and sIdxY for predictor

begin

ComputeIterminal;

t := ActiveSolutionObj.DynaVars.t;

h := ActiveSolutionObj.DynaVars.h;

f := ActiveSolutionObj.Frequency;

corrector := ActiveSolutionObj.DynaVars.IterationFlag;

d := 1 / FSampleFreq;

nstep := trunc (1e-6 + h/d);

w := 2 \* Pi \* f;

pk := sqrt(2);

vnow := Vterminal^[1];

vin := 0;

y := 0;

iu := sIdxU;

iy := sIdxY;

for k := 1 to FFiltlen do begin

z[k] := zlast[k];

whist[k] := wlast[k];

end;

for i:=1 to nstep do begin

iu := OffsetIdx (iu, 1, Ffiltlen);

// push input voltage waveform through the first PWL block

scale := 1.0 \* i / nstep;

vre := vlast.re + (vnow.re - vlast.re) \* scale;

vim := vlast.im + (vnow.im - vlast.im) \* scale;

wt := w \* (t - h + i \* d);

vin := pk \* (vre \* cos(wt) + vim \* sin(wt));

whist[iu] := Fbp1.GetYValue(vin);

// apply the filter and second PWL block

z[iu] := 0;

for k := 1 to Ffiltlen do begin

z[iu] := z[iu] + Ffilter.Yvalue\_pt[k] \* whist[MapIdx(iu-k+1,Ffiltlen)];

end;

for k := 2 to Ffiltlen do begin

z[iu] := z[iu] - Ffilter.Xvalue\_pt[k] \* z[MapIdx(iu-k+1,Ffiltlen)];

end;

y := Fbp2.GetYValue(z[iu]);

// updating outputs

if (corrector = 1) and (abs(y) > sIpeak) then

sIpeak := abs(y); // catching the fastest peaks

// update the RMS

iy := OffsetIdx (iy, 1, Fwinlen);

y2[iy] := y \* y; // brute-force RMS update

if i = nstep then begin

y2sum := 0.0;

for k := 1 to Fwinlen do y2sum := y2sum + y2[k];

sIrms := sqrt(y2sum / Fwinlen); // TODO - this is the magnitude, what about angle?

end;

end;

if corrector = 1 then begin

sIdxU := iu;

sIdxY := iy;

vlast := vnow;

sVwave := vin;

sBP1out := whist[sIdxU];

sFilterout := z[sIdxU];

sIwave := y;

for k := 1 to FFiltlen do begin

zlast[k] := z[k];

wlast[k] := whist[k];

end;

end;

end;