

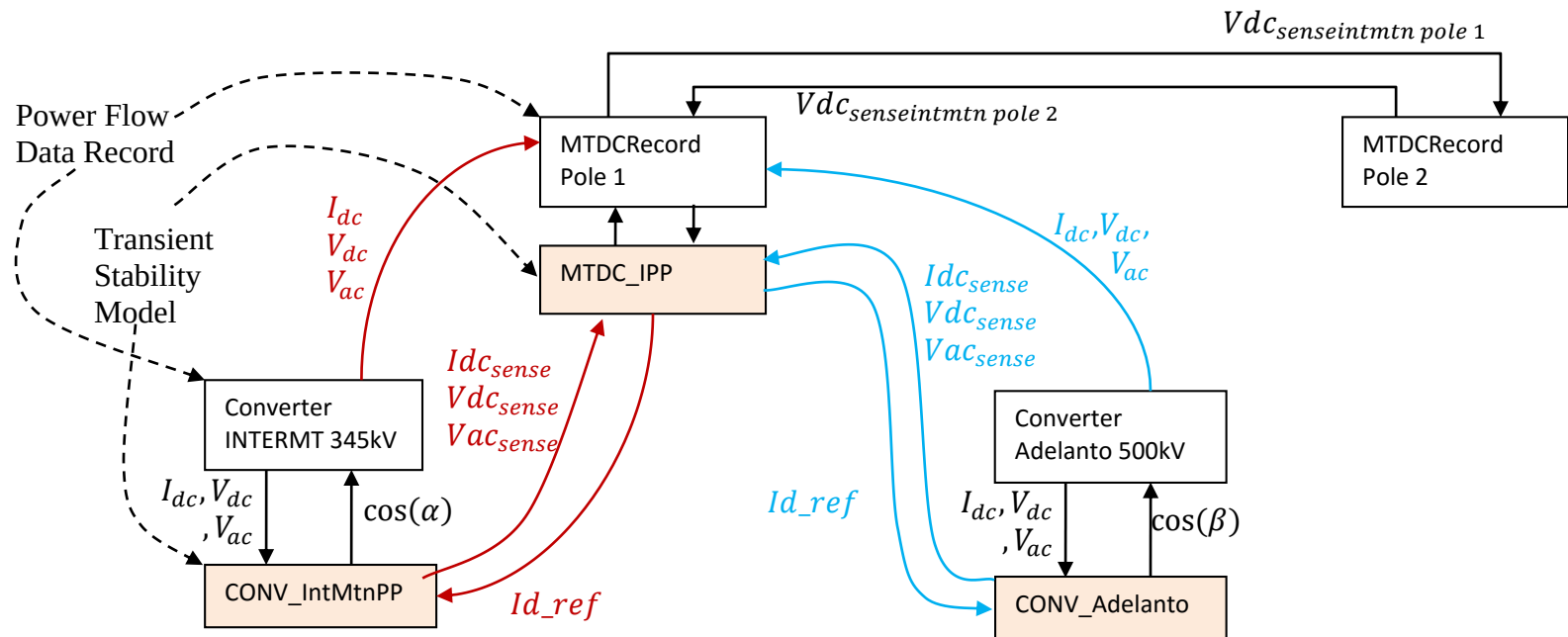
## General Overview of Multi-Terminal DC Line Model for the Intermountain Power Plant DC Line

Simulator has a multi-terminal DC (MTDC) record which represents the grouping of the dc converters, dc buses and dc lines for a single pole of a MTDC transmission line. The Intermountain DC Line (IPP) is represented by two of these records: one for each pole of the IPP. For the IPP, each MTDC record contains two DC converters with one at Intermountain (North end) and one at Adelanto (South end). In Simulator these DC converters are also represented by unique objects.

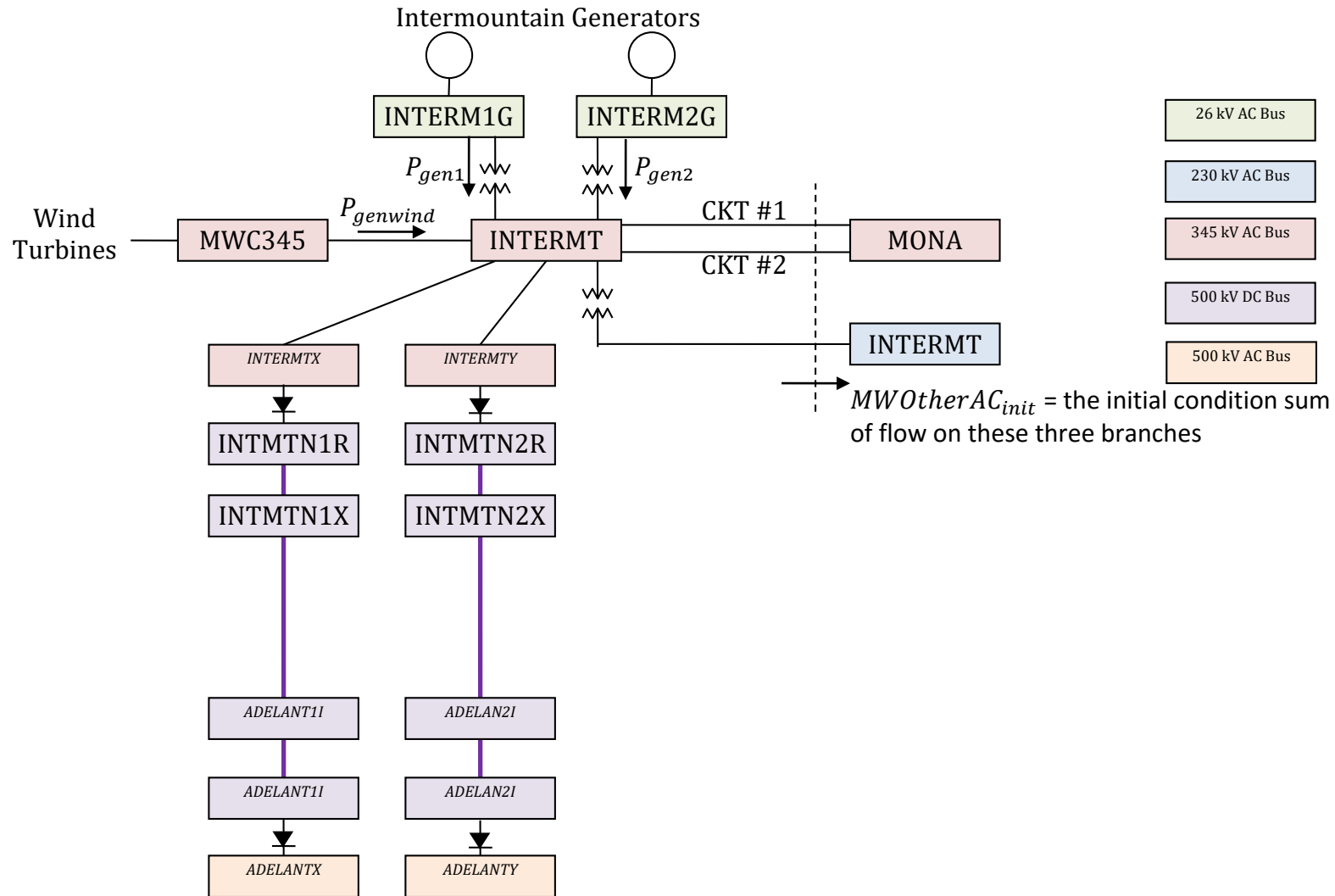
The transient stability model of the IPP works by assigning a dynamic model to each of the two MTDC records, and also assigning a dynamic model to each of the DC converter objects. To model this, there are 2 dynamic models because the converters at each end use the same dynamic model.

|               |                              |
|---------------|------------------------------|
| MTDC_IPP      | Assigned to the MTDC record. |
| CONV_IntMtnPP | Assigned to each converter   |

The flow of signals is depicted in the following image.

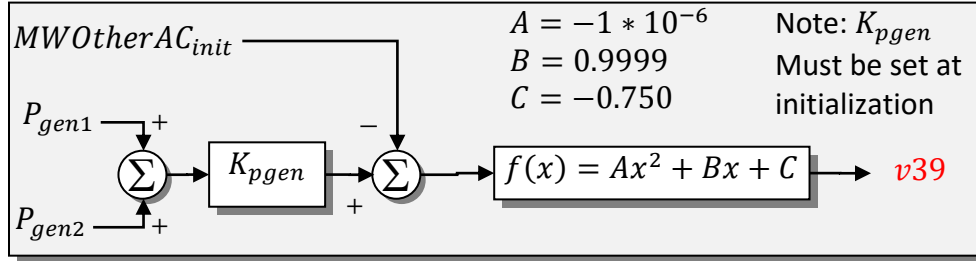


# **MTDC\_IPP** **Current Order Allocation Calculations for Intermountain/Adelanto**

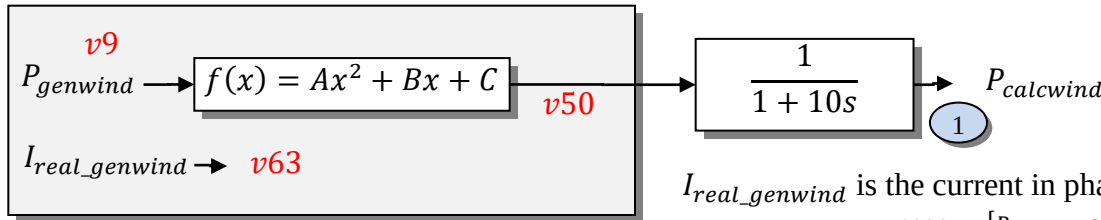


## MTDC\_IPP

### Current Order Allocation Calculations for Intermountain/Adelanto



Every  $T_{discrete}$  (0.035 seconds), values are calculated for  $v9$ ,  $v39$ ,  $v63$ ,  $v50$ ,  $v70$ , and  $v73$ . Essentially these represent a slower time-scale integration method which is going on.



$I_{real\_genwind}$  is the current in phase with the terminal voltage and equals

$$I_{real\_genwind} = \frac{1000}{345\sqrt{3}} * \frac{[P_{genwin} \cos(\theta_{term}) + Q_{genwin} \sin(\theta_{term})]}{V_{puterm}}$$

If *BothMonaACLinesOPEN*, Then *Contribution* =  $1000 * (\text{FrequencyAtInterMtnACBus} - 60)$   
 Else *Contribution* = 0

If  $\text{abs}(v70) > 2001$  then

If  $v70 > 2000$  then  $v70 = 2000$

Else  $v70 = -2000$

Else if *BothPolesAtMaxCurrentSched* then

If *OnePoleWellAboveMaxCurrent* then  $v70 = v70 + (-8)(T_{discrete}) \left( \frac{170}{1000} \right)$

Else  $v70 = v70 + (\text{Contribution})(T_{discrete}) \left( \frac{170}{1000} \right)$

If *BothMonaACLinesOPEN* then start a timer and keep pushing back the end time by 1.0 seconds as long as both are open. Thus the Timer is active for 1.0 seconds after one or both of the Mona lines closing back in. As soon as the timer becomes inactive, reset  $v70 = 0$

If *TimerActive* then  $v73 = v70$  Else  $v73 = 0$

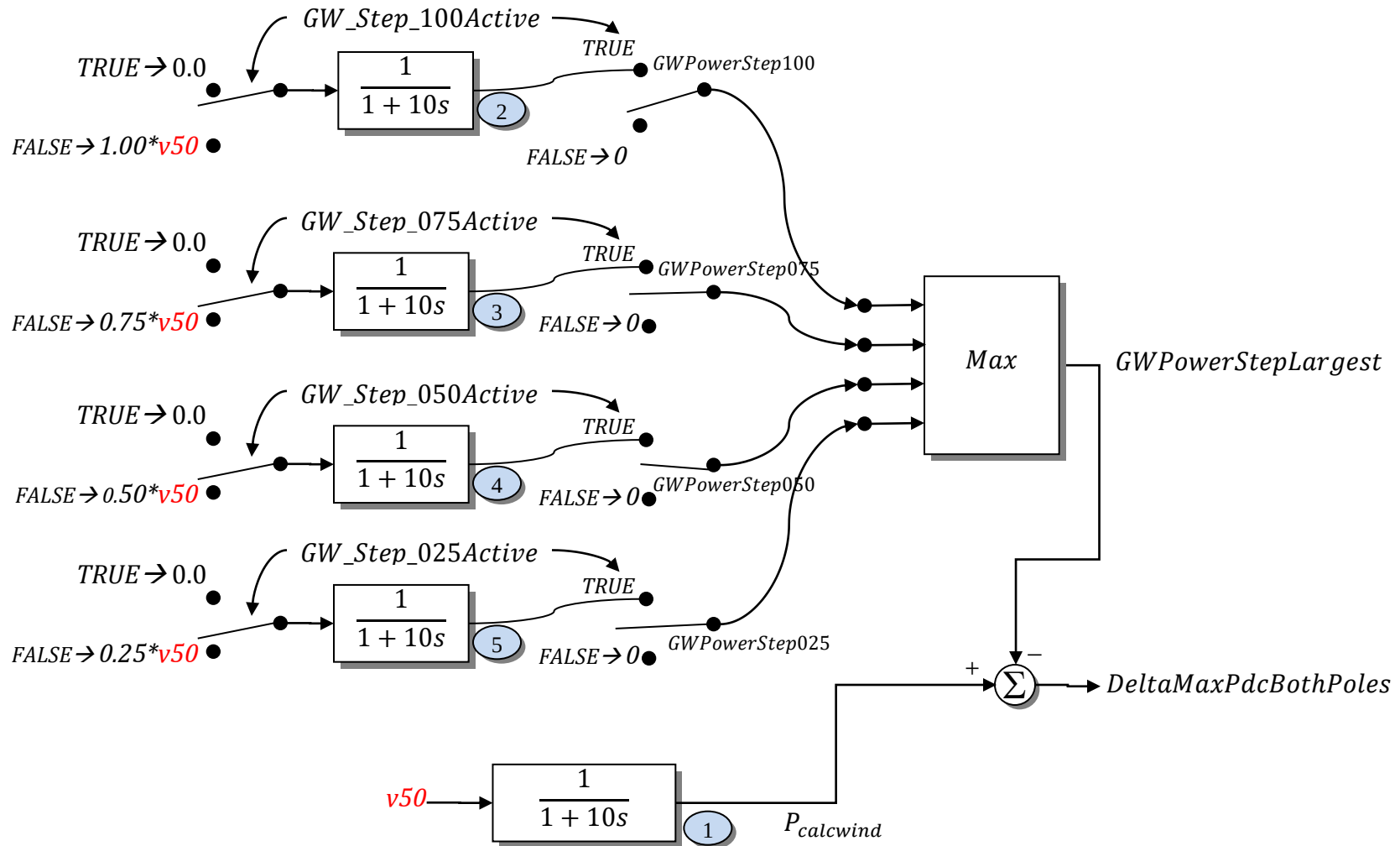
$$P_{maxgenwind} = 200$$

$$I_{maxgenwind} = \frac{1000}{345\sqrt{3}} * P_{maxgenwind}$$

## MTDC\_IPP AC WindConnection Calculations

| Boolean Variable         | Evaluation for the Boolean                                                                                                                                                                                                                                                         |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>VacOver90</i>         | is set FALSE if the AC voltage at the Intermountain terminal falls below 0.9 per unit for 0.020 seconds. It will remain FALSE until the voltage recovers above 0.90 per unit at which point it resets to TRUE immediately.                                                         |
| <i>IWindOver200</i>      | is set TRUE as soon as $I_{real\_genwind} > 2 * I_{maxgenwind}$ . It remains TRUE until the current drops below this value for 0.10 seconds.                                                                                                                                       |
| <i>PWindOver_50</i>      | $(P_{calcwind} > 0.50 * P_{maxgenwind})$ Note: $P_{calcwind}$ is State #1                                                                                                                                                                                                          |
| <i>PWindOver_25</i>      | $(P_{calcwind} > 0.25 * P_{maxgenwind})$ Note: $P_{calcwind}$ is State #1                                                                                                                                                                                                          |
| <i>PWindUnder_10</i>     | $(P_{calcwind} < 0.10 * P_{maxgenwind})$ Note: $P_{calcwind}$ is State #1                                                                                                                                                                                                          |
| <i>IWindUnder_20</i>     | $(I_{real\_genwind} < 0.20 * I_{maxgenwind})$                                                                                                                                                                                                                                      |
| <i>Do_100Active</i>      | $(PWindUnder_10 \text{ AND } IWindUnder_20)$                                                                                                                                                                                                                                       |
| <i>Do_100_OS_Active</i>  | When <i>Do_100Active</i> toggles from FALSE to TRUE, then we set the value <i>Do_100_OS_Active</i> TRUE for a period of 0.100 seconds after which it resets back to FALSE regardless. Thus if <i>Do_100Active</i> stays TRUE for a long time this won't ever get set back to TRUE. |
| <i>Do_100_ON_Active</i>  | $(PWindOver_25 \text{ AND } (VacOver90) \text{ AND } (\text{NOT } IWindOver200) \text{ AND } (\text{NOT } Do_100\_OS\_Active))$                                                                                                                                                    |
| <i>Do_100Active</i>      | If <i>Do_100_ON_Active</i> then <i>Do_100Active</i> = FALSE                                                                                                                                                                                                                        |
| <i>GW_STEP_100Active</i> | If <i>Do_100Active</i> stays TRUE for 0.50 seconds (reset timer if it every goes FALSE), then <i>GW_STEP_100Active</i> will be set and locked TRUE for 40 seconds after which it resets to FALSE.                                                                                  |
| <i>Do_075Active</i>      | $(0.25 * P_{calcwind} > v50) \text{ AND } Do_100\_ON\_Active \text{ AND } PWindOver_50$                                                                                                                                                                                            |
| <i>GW_STEP_075Active</i> | If <i>Do_075Active</i> stays TRUE for 0.40 seconds (reset timer if it every goes FALSE), then <i>GW_STEP_075Active</i> will be set and locked TRUE for 40 seconds after which it resets to FALSE.                                                                                  |
| <i>Do_050Active</i>      | $(0.50 * P_{calcwind} > v50) \text{ AND } Do_100\_ON\_Active$                                                                                                                                                                                                                      |
| <i>GW_STEP_050Active</i> | If <i>Do_050Active</i> stays TRUE for 0.30 seconds (reset timer if it every goes FALSE), then <i>GW_STEP_050Active</i> will be set and locked TRUE for 40 seconds after which it resets to FALSE.                                                                                  |
| <i>Do_025Active</i>      | $(0.75 * P_{calcwind} > v50) \text{ AND } Do_100\_ON\_Active \text{ AND } PWindOver_50$                                                                                                                                                                                            |
| <i>GW_STEP_025Active</i> | If <i>Do_025Active</i> stays TRUE for 0.20 seconds (reset timer if it every goes FALSE), then <i>GW_STEP_025Active</i> will be set and locked TRUE for 40 seconds after which it resets to FALSE.                                                                                  |

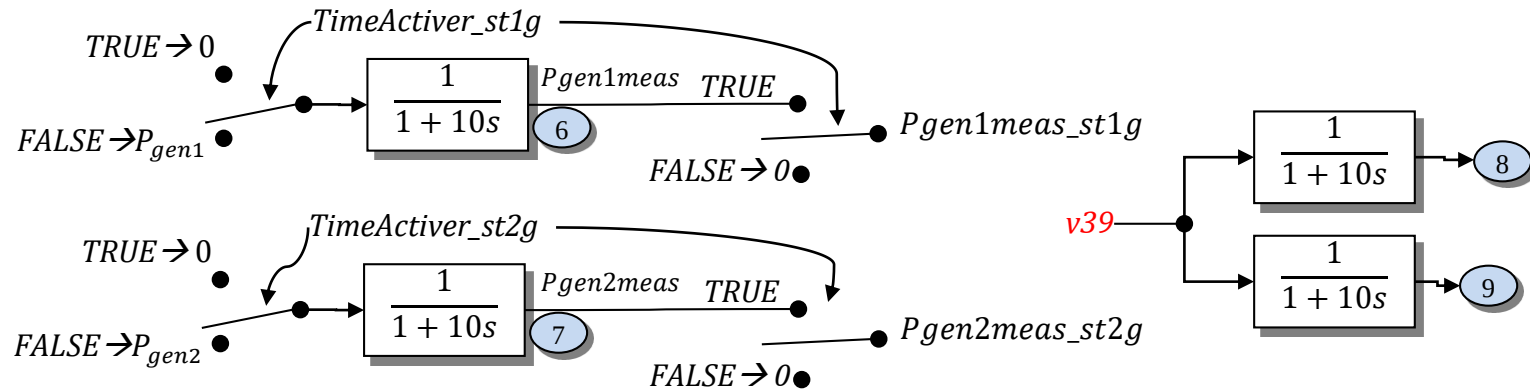
# **MTDC\_IPP** **AC Wind Connection Calculations**



**MTDC\_IPP**  
**Intermountain Generator Status Change Contribution**

| Boolean Variable         | Evaluation for the Boolean                                                                                                    |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <i>st1gChange</i>        | TRUE if status if INTERM1G is different than the initial condition generator status                                           |
| <i>TimerActive_st1g</i>  | When <i>st1gChange</i> first becomes TRUE, set this value TRUE for 40 seconds                                                 |
| <i>TimerExpire_st1g</i>  | When <i>TimerActive_st1g</i> first goes TRUE, then this value is set FALSE but then after a 1.0 second delay it becomes TRUE. |
| <i>JustChanged_st1g</i>  | <i>TimerActive_st1g</i> AND (NOT <i>TimerExpire_st1g</i> )                                                                    |
| <i>TimerActive_2not1</i> | If <i>JustChanged_st2g</i> AND (NOT <i>st1gChange</i> ) then set this TRUE for 3.0 seconds                                    |
| <i>GenBoo2not1</i>       | <i>TimerActive_2not1</i> AND (NOT <i>TimerActive_st1g</i> )                                                                   |
| <i>TimerActive_st1gB</i> | When <i>st1gChange</i> first become TRUE, set this value TRUE for 2.5 second                                                  |
| <i>GenBoo1</i>           | <i>TimerActive_st1gB</i> AND ( <i>JustChanged_st1g</i> OR (NOT <i>st1gChange</i> ))                                           |
|                          |                                                                                                                               |
| <i>st2gChange</i>        | TRUE if status if INTERM2G is different than the initial condition generator status                                           |
| <i>TimerActive_st2g</i>  | When <i>st2gChange</i> first becomes TRUE, set this value TRUE for 40 seconds                                                 |
| <i>TimerExpire_st2g</i>  | When <i>TimerActive_st2g</i> first goes TRUE, then this value is set FALSE but then after a 1.0 second delay it becomes TRUE. |
| <i>JustChanged_st2g</i>  | <i>TimerActive_st2g</i> AND (NOT <i>TimerExpire_st2g</i> )                                                                    |
| <i>TimerActive_1not2</i> | If <i>JustChanged_st1g</i> AND (NOT <i>st2gChange</i> ) then set this TRUE for 3.0 seconds                                    |
| <i>GenBoo1not2</i>       | <i>TimerActive_1not2</i> AND (NOT <i>TimerActive_st2g</i> )                                                                   |
| <i>TimerActive_st2gB</i> | When <i>st2gChange</i> first become TRUE, set this value TRUE for 2.5 second                                                  |
| <i>GenBoo2</i>           | <i>TimerActive_st2gB</i> AND ( <i>JustChanged_st2g</i> OR (NOT <i>st2gChange</i> ))                                           |

## MTDC\_IPP Intermountain Generator Status Change Contribution



Note:  $P_{gen1DiscreteTrack}$  is set to  $P_{gen1}$  during initialization

If  $(P_{gen1} - P_{gen1DiscreteTrack}) > 50$  then  $P_{gen1DiscreteTrack} = P_{gen1DiscreteTrack} + 50$

Else If  $(P_{gen1} - P_{gen1DiscreteTrack}) < -50$  then  $P_{gen1DiscreteTrack} = P_{gen1DiscreteTrack} - 50$

Else  $P_{gen1DiscreteTrack} = P_{gen1}$

$DiffP_{gen1Discrete} = P_{gen1meas} - P_{gen1DiscreteTrack}$

Note:  $P_{gen2DiscreteTrack}$  is set to  $P_{gen2}$  during initialize

If  $(P_{gen2} - P_{gen2DiscreteTrack}) > 50$  then  $P_{gen2DiscreteTrack} = P_{gen2DiscreteTrack} + 50$

Else If  $(P_{gen2} - P_{gen2DiscreteTrack}) < -50$  then  $P_{gen2DiscreteTrack} = P_{gen2DiscreteTrack} - 50$

Else  $P_{gen2DiscreteTrack} = P_{gen2}$

$DiffP_{gen2Discrete} = P_{gen2meas} - P_{gen2DiscreteTrack}$

If  $GenBoo2not1$  then  $P_{gen1Value} = DiffP_{gen1Discrete}$  Else  $P_{gen1Value} = P_{gen1Meas\_st1g}$

If  $GenBoo1not2$  then  $P_{gen2Value} = DiffP_{gen2Discrete}$  Else  $P_{gen2Value} = P_{gen2Meas\_st2g}$

## *MTDC\_IPP*

### *Calculating a new Power Schedule*

*AccumulationValue* = *State#8* + *State#1* - *Pgen1Value* - *Pgen2Value* - *GWPowerStepLargest*

If *GenBoo1* OR *GenBoo2* then

Set *State#9* = *v39* and derivative of *State #9* to 0.0

During initialization, following values are set

If *NumMonaLinesInservice* = 2 then *MinPdcBothPoles* = 600

else *MinPdcBothPoles* = 300

If *ActualPdcOnBothPoles* > *MinPdcBothPoles* then *LimitedMinPdcBothPoles* = *MinPdcBothPoles*

else *LimitedMinPdcBothPoles* = *PdcBothPoles*

*LastNumMonaLinesInservice* = -1

Assuming these are initialized in this way, then do following

*NewPschedBothPoles* = *State#9* + *State#1*

If *NewPschedBothPoles* > *AccumulationValue* then *NewPschedBothPoles* = *AccumulationValue*

If *NewPschedBothPoles* < *LimitedMinPdcBothPoles* then *NewPschedBothPoles* = *LimitedMinPdcBothPoles*

*NumMonaLinesInservice* = Count number of lines to MONA that are inservice (0, 1, or 2)

If *NumMonaLinesInservice* <> *LastNumMonaLinesInservice* then

If *NumMonaLinesInservice* = 2 then *MinPdcBothPoles* = 600 + *DeltaMaxPdcBothPoles*

else *MinPdcBothPoles* = 300 + *DeltaMaxPdcBothPoles*

If *NewPschedBothPoles* > *MinPdcBothPoles* then *LimitedMinPdcBothPoles* = *MinPdcBothPoles*

Else *LimitedMinPdcBothPoles* = *NewPschedBothPoles*

*LastNumMonaLinesInservice* = *NumMonaLinesInservice*

Every 0.104 seconds, do the following

If (*NewPschedBothPoles* > *LimitedMinPdcBothPoles*) AND (*LimitedMinPdcBothPoles* < *MinPdcBothPoles*) then

*LimitedMinPdcBothPoles* = *LimitedMinPdcBothPoles* + 1

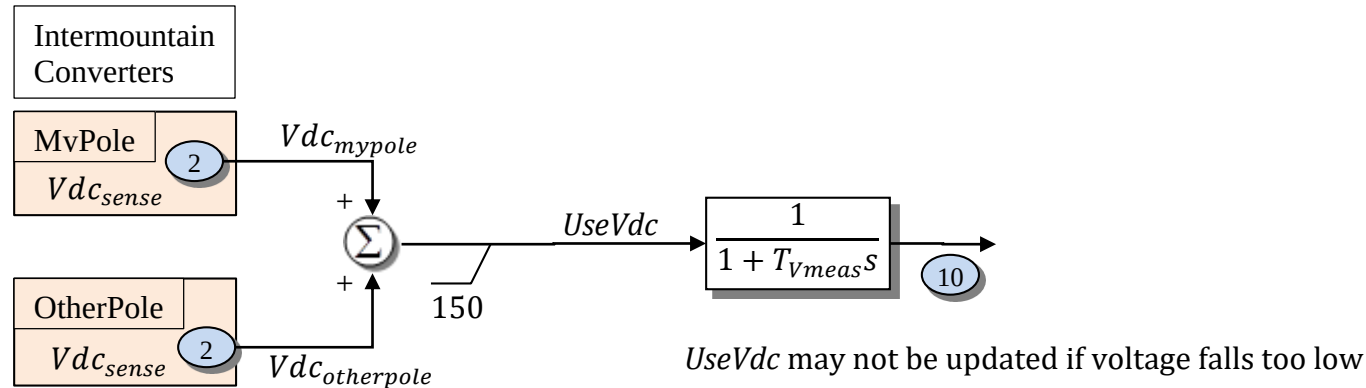
If (*NewPschedBothPoles* = *LimitedMinPdcBothPoles*) AND (*LimitedMinPdcBothPoles* > 0) then

*LimitedMinPdcBothPoles* = *LimitedMinPdcBothPoles* - 1

If *NewPschedBothPoles* > 3200 then *NewPschedBothPoles* = 3200

## MTDC\_IPP

### Conversion of new Power Schedule to a Current Schedule



Initialize  $VdcFreeze$  to FALSE and  $UseVdc = Vdc_{mypole} + Vdc_{otherpole}$

$VdcFreeze = [(NOT\ VdcFreeze)\ AND\ (Vdc_{mypole} < 425)]\ OR\ [(VdcFreeze)\ AND\ (Vdc_{mypole} < 375)]$

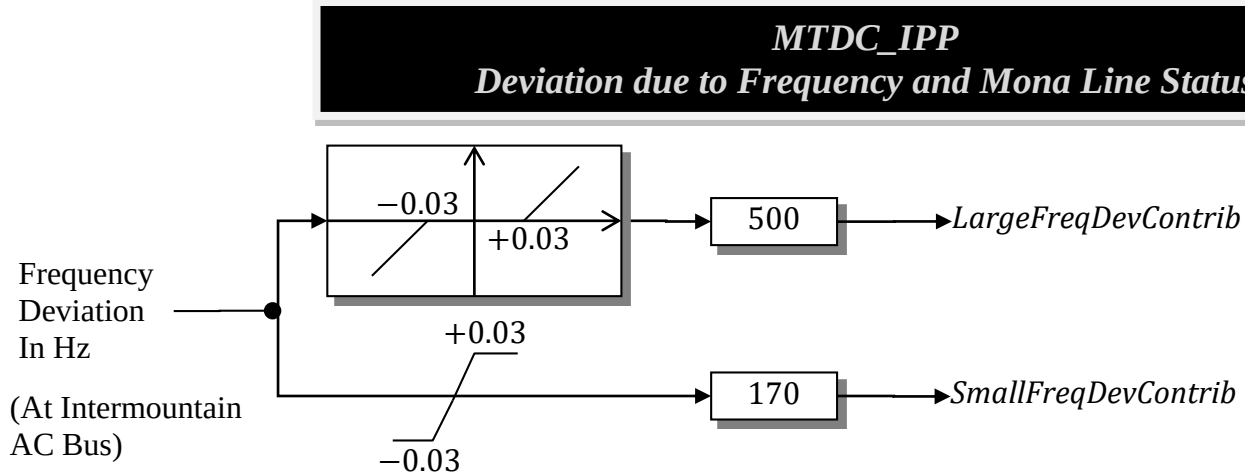
If NOT  $VdcFreeze$  then  $UseVdc = Vdc_{mypole} + Vdc_{otherpole}$

Otherwise use the previous value of  $UseVdc$

If  $[abs(UseVdc - State\#10) > 100]$  then for a period of 0.20 seconds treat the value of  $T_{vmeas} = 0.0$ . Otherwise  $T_{vmeas} = 0.30$ .

If  $T_{vmeas}$  is too small then immediately set State #10 equal to  $UseVdc$

$UseThisCurrentSched = 1000 * NewPschedBothPoles / (State\#10)$



// The following is a rate limiter on the change in this value. In the EPCL code the “4800\*TimeStep” actually just says 20.0  
 // we assume the intent was a 0.25 cycle time-step, so at 60Hz this means 4800\*TimeStep

*BothMonaOpenContrib* = **v73**

If *BothMonaOpenContrib* > (*Last\_BMOC* + 4800\**TimeStep*) then *BothMonaOpenContrib* = *Last\_BMOC* + 4800\**TimeStep*

If *BothMonaOpenContrib* < (*Last\_BMOC* - 4800\**TimeStep*) then *BothMonaOpenContrib* = *Last\_BMOC* - 4800\**TimeStep*

*Last\_BMOC* = *BothMonaOpenContrib*

If *NumMonaLinesInService*=2 then *ExtraCurrent* = *BothMonaOpenContrib* + *SmallFreqDevContrib*

Else *ExtraCurrent* = 0

*ExtraCurrent* = *ExtraCurrent* + *LargeFreqDevContrib*

*NewCurrentSched* = *UseThisCurrentSched* + *ExtraCurrent*

## *MTDC\_IPP Current Limits*

// Following are set here, but they are used way back in the integration routine for calculating v73

*AtMaxCurrentSched* = (*NewCurrentSched* > 3680)

*WellAboveMaxCurrentSched* (*NewCurrentSched* > 3720)

If *AtMaxCurrentSched* then *NewCurrentSched* = 3680

*IncreasingCurrent* = (*NewCurrentSched* ≥ *ISched\_Minus\_0.03sec*)

If (NOT *IncreasingCurrent*) AND (*abs(NewCurrentSched - ISched\_Minus\_0.03sec)* > 48) then

*NewCurrentSched* = *ISched\_Minus\_0.03sec*

Using a timer every 0.01 seconds do the following:

*ISched\_Minus\_0.03sec* = *ISched\_Minus\_0.02sec*

*ISched\_Minus\_0.02sec* = *ISched\_Minus\_0.01sec*

*ISched\_Minus\_0.01sec* = *NewCurrentSched*

Send *NewCurrentSched* as *Id\_ref* to the Intermountain Converter.

Send *ISched\_Minus\_0.02sec* as *Idref* to the Adelanto side // discrete communication delay of 0.02 seconds