Update to HEPI controls: Bias-hold and Watchdog

Brian Lantz, Hugo Paris, et. al.

Dec 10, 2014

Plan:

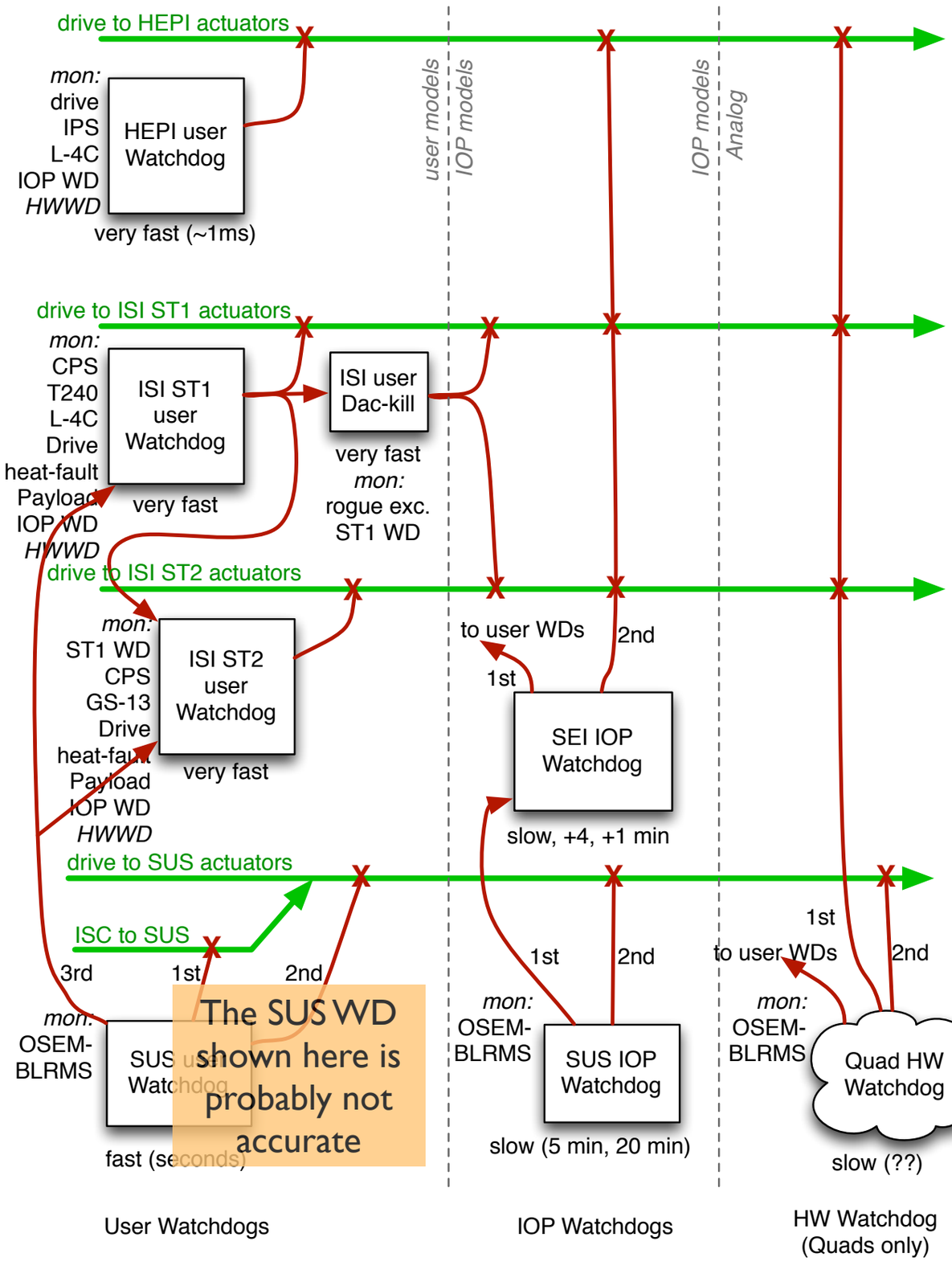
1. Describe suggested changes (today)
2. Update parameters (and plans) based on comments.
   - many parameters set by metric of "seems OK"
3. File ECR
4. Test at LASTI
5. Implement at sites

advancedligo

Review of current Watchdogs

Watchdog Interactions - Nov. 2014

adapted from G1301210



Discussed in WD wiki:
https://awiki.ligo-wa.caltech.edu/aLIGO/SeiSusWatchDogCommissioning

Some tweaks suggested:

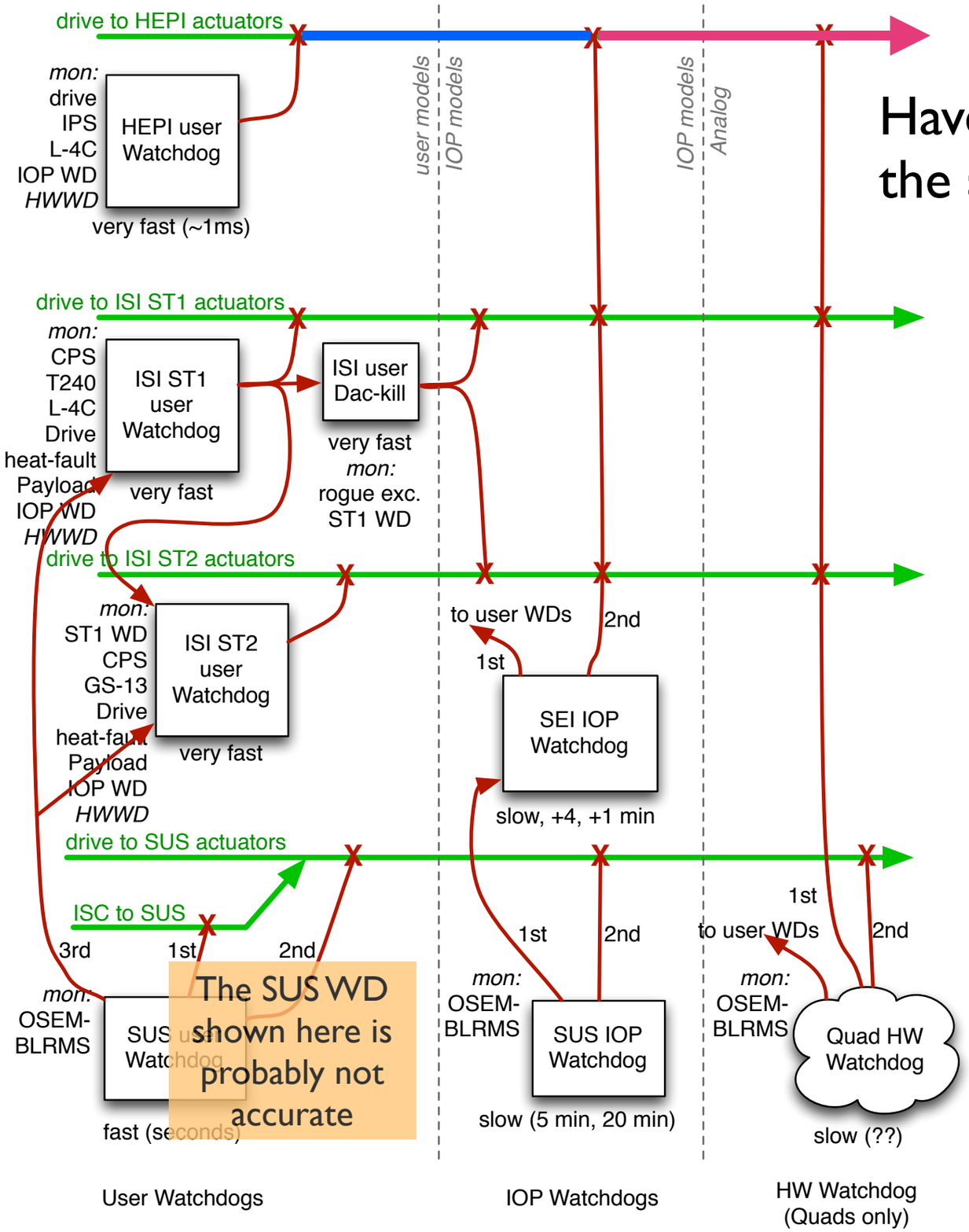
1) Increase ISI damping-hold-on time from 3 sec to 5 sec. to allow full damping of optical table from big whacks.
2) Increase GS-13 allowed sat. from 10 to 400 (1/400 sec to 1/10 of sec).
3) debug rogue excitation mon.

Actuator signals in Green
Watchdog signals in Red

advancedligo

The change...

adapted from G1301210



Have the HEPI user WD hold the output to the actuator, rather than change it to zero.

Actuator signals in Green

Watchdog signals in Red

Summary of changes

- HEPI diagram updates
  - Install bias-hold/ update Guardian to support it.
  - Remove HEPI user DAC kill.
  - Remove HEPI WD-KillAll feature.
    - (Confirm IOP DAC-kill installed for HEPI)
    - Only the IOP WD can shut off all HEPI output.

- IOP WD
  - Adjust timing of SEI IOP WD.
  - mod SEI IOP DAC Kill to sequence the ISI/ HPI trips.

Bias Hold

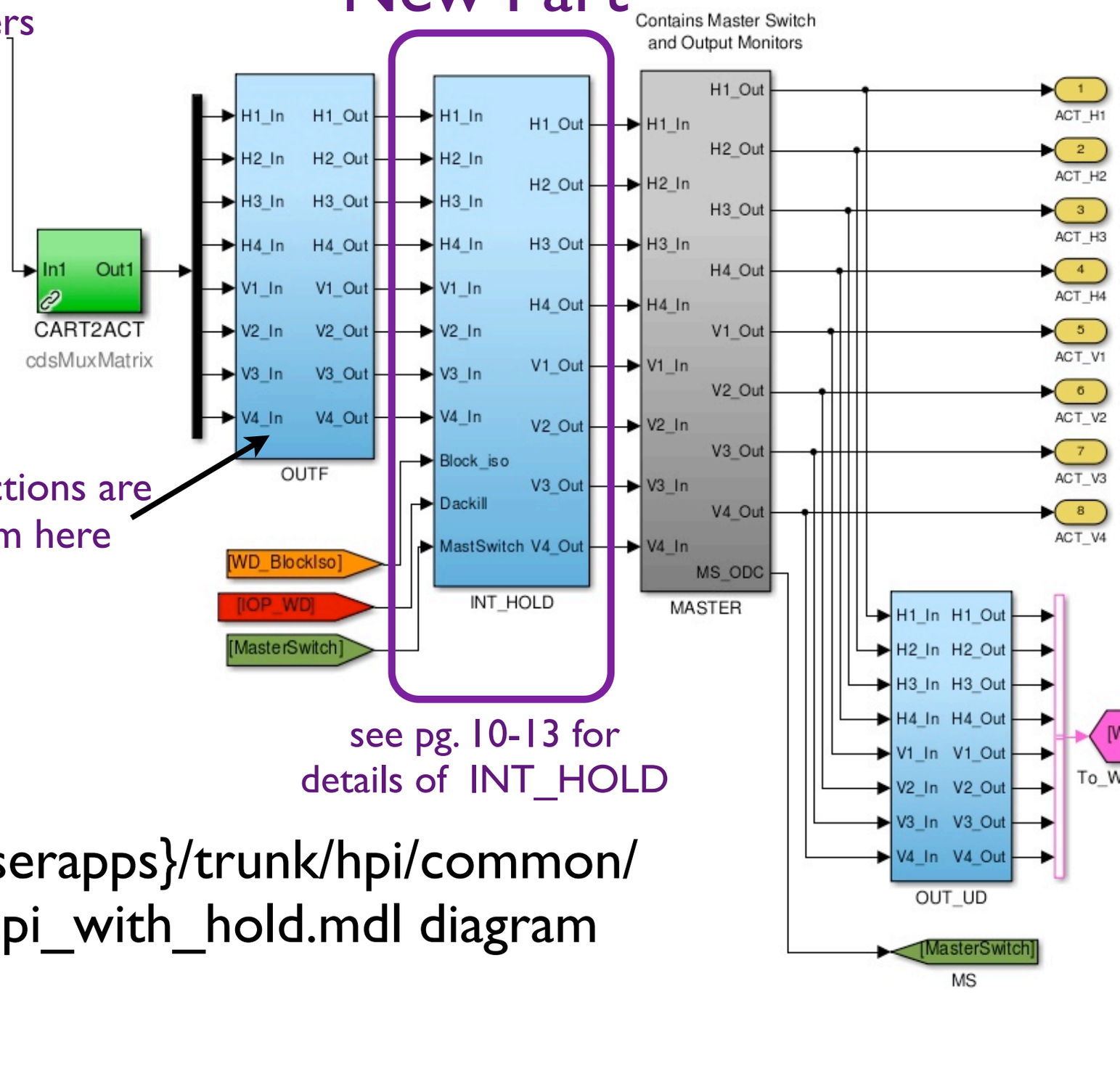
- Idea - Last stage of the HEPI control will be a P+I, like a 'boost' stage. The integrator part carries all the low frequency control (below about 0.1 Hz) and saturates at 30k counts. (see T1300073)
- On a Watchdog trip, we shut off the control loops at the input to this last part - so the low freq. "control" just becomes a "bias" as it integrates the input of 0.
- So HEPI user WD trip = position hold. Drives sit at some value run by C-code.
- Rely on SEI IOP WD to shut off all outputs.
- Comments?

P+I = Proportional + Integral, P gain is one, P/I corner at 0.1 Hz

Bias-Hold diagram 1

HEPI controllers

New Part



Transfer functions are driven from here

see pg. 10-13 for details of INT_HOLD

Detail of {userapps}/trunk/hpi/common/models/hepi_with_hold.mdl diagram

Guardian update for HEPI

- Proposal- Restore all 6 DOF HEPI locations to stored position because HEPI plant has hysteresis. 8 actuators taken close to position.
- This will cause tilt, on startup and make T240s ring - therefore we want to minimize number of times this happens.
- On computer start, or IOP WD trip, need to restore "correct" position.
- "Correct" position is determined by the displacement sensors.
  e.g. H1:HPI-ETMX_IPS_X_TARGET
- "Good Guess" for the bias-hold can is saved.
  e.g. H1:HPI-ETMX_INT_HOLD_H1_DRIVE_BIAS_REFERENCE
- At computer start or IOP trip, drive = 0. Use Guardian to load bias-hold integrators to "Good guess" location in 5 sec. HEPI moves for about 30 sec.
- From this "good guess" bias-hold location, start isolation controllers.
- HPI User WD trips result in a bias-hold, and guardian restores the "good guess" location - provides repeatable starting point. (c.f from bias-hold)

IOP WD timing

↓ = WD signal
X = IOP DAC Kill
X = ISI DAC Kill

Current

check if SUS RMS too big

SUS IOP WD

SEI IOP WD

*

4 min

1 min

Signal to ISI + HPI user WD
- currently disconnected at EX?
if connected would have ISI + HPI DAC kill in 3-5 seconds

Proposed

check if SUS RMS too big

SUS IOP WD

Kill SEI control sooner, but hold bias longer

SEI IOP WD

*

2 min

*

1 min

Signal to ISI user WD

Signal to HPI user WD - bias hold

t=0 5 min 10 min 15 min 20 min

* - The signal to the SEI IOP WD is a latched signal output from the SUS IOP WD & requires human to clear

Notes on timing

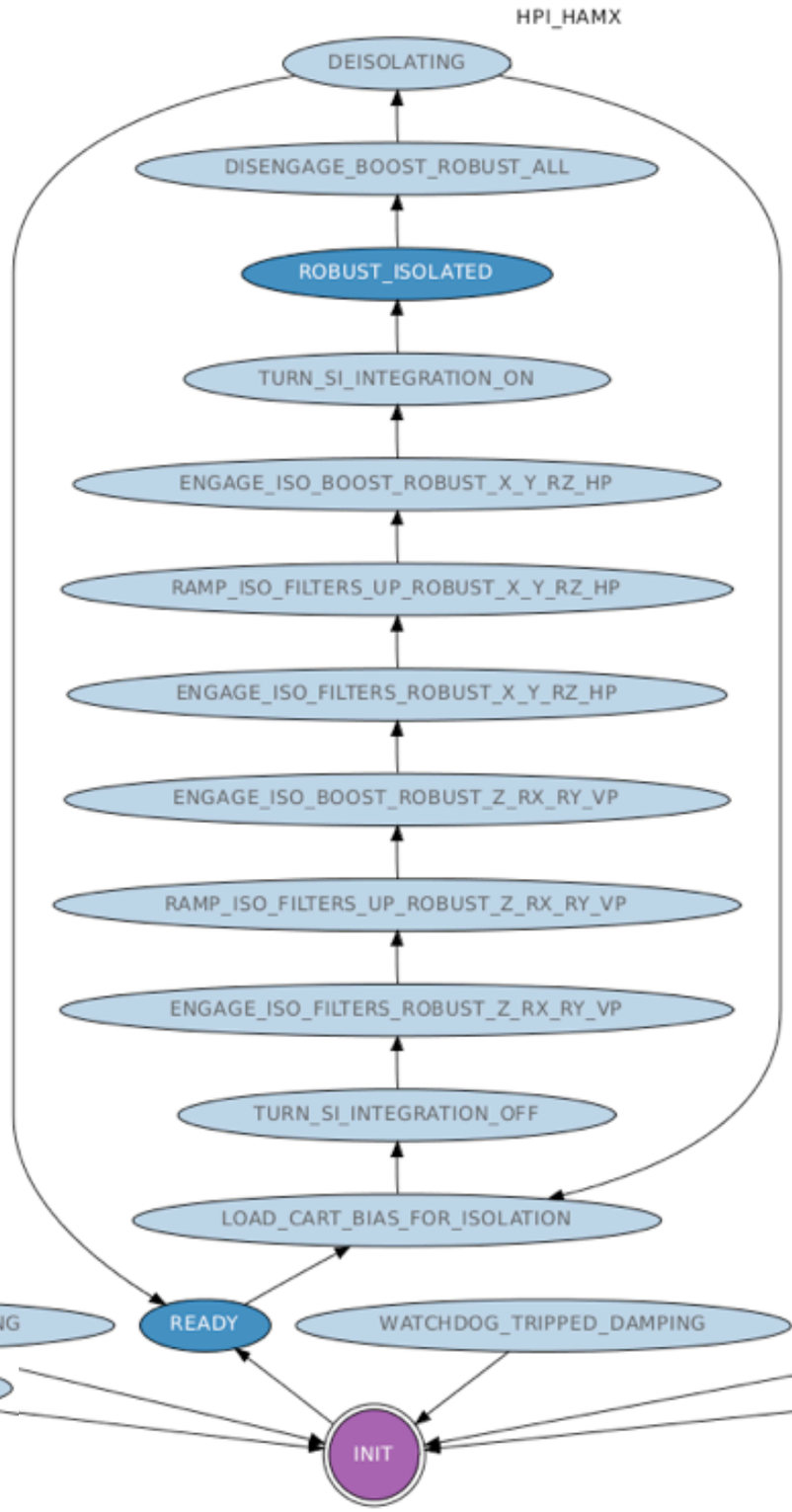
- The IOP watchdog "warning" signal is latched, and will remain on until a human clears it.
- Since the IOP signal to the SEI IOP is latched, the whole SEI will be turned off unless someone intervenes.
- 2 mins. chosen for signal to ISI - picked to be short because it seems likely the SEI is responsible.
- HPI tripped first so that ISI can absorb the shake. Chosen philosophically, not based on data.
- 8 min delay chosen so that a human has a chance to reset the IOP WD before the bias-hold is release.
- 5 + 15 min for SUS IOP not changed.

Next

- Agree on first cut of order and timing parameters.
- Agree on Guardian sequencing.
- Test at MIT (Sebastien, Hugo)
- Implement at sites.

Guardian Info

from SEI log 657

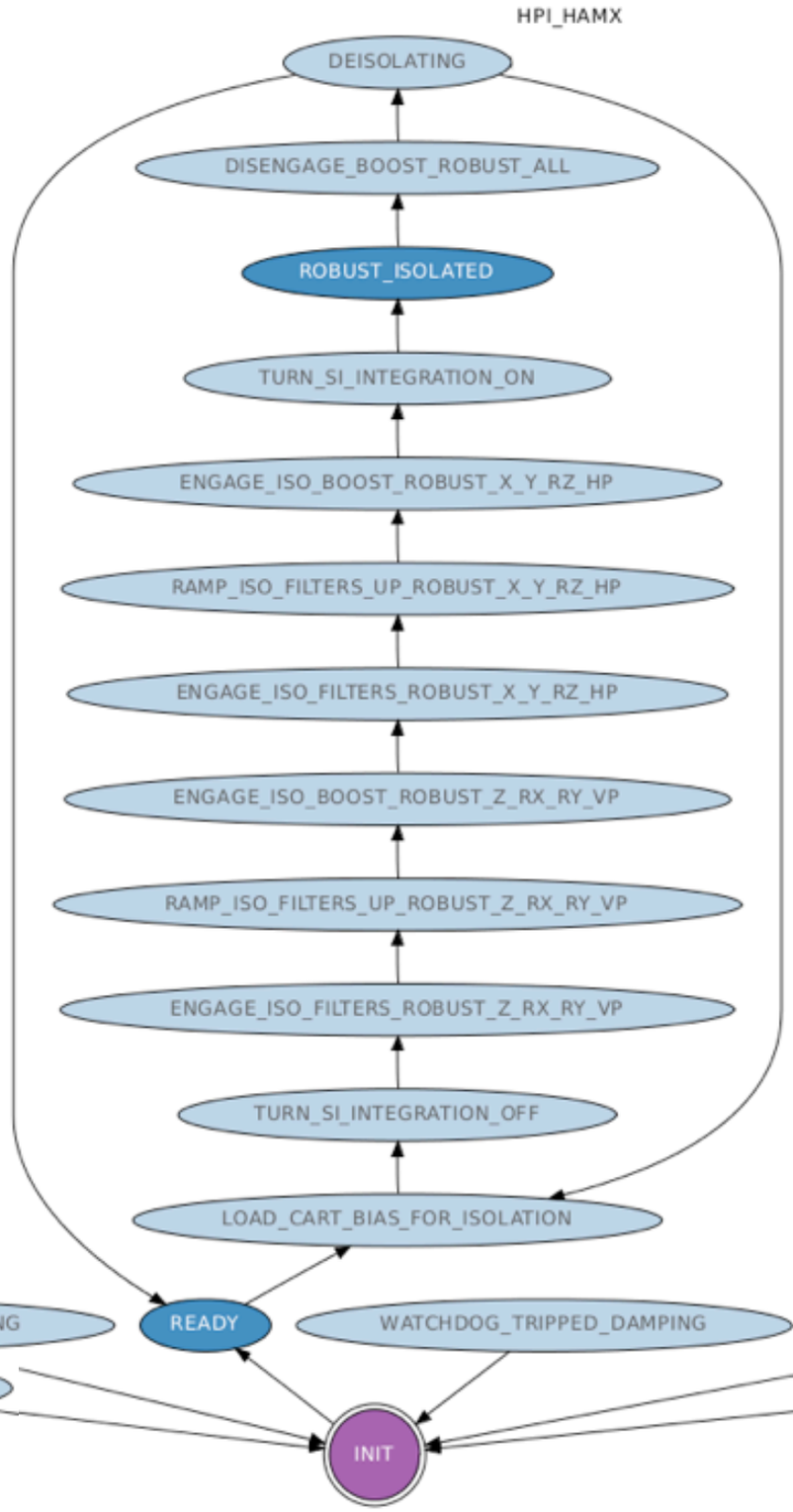


The model (re)starts:

1. **User WD is tripped**
2. FE code prevents BIAS HOLD from being driven
3. DRIVE_BIAS_REFERENCE(s) were previously stored in the safe.snap.
4. DRIVE_BIAS_REFERENCE epics values get loaded when the safe.snap is read at model (re)start
5. BIAS HOLD is zero
6. **The operator untrips the user WD**
7. **HEPI is in INIT state**
8. FE code allows BIAS HOLD to be driven
9. HEPI starts going from INIT to READY
10. Guardian reads the DRIVE_BIAS_REFERENCE epics values
11. Guardian computes a drive signal to bring HEPI to its reference alignment
12. Guardian drives a smooth ramp into each DRIVE_BIAS_EXC (1 per DOF) to bring the BIAS HOLD, and thus HEPI, to the desired aligned position
13. **HEPI is in READY state**
14. If/when requested, HEPI starts transitioning from READY to ROBUST_ISOLATED
15. Guardian loads the CART BIAS
16. Guardian stops the BIAS HOLD integration. Integrated vaules are maintained
17. Guardian ramps up the isolation loops, and turns the boosts on
18. All the isolation loops, and boosts are on
19. Guardian starts the BIAS HOLD integration again
20. **HEPI is in ROBUST_ISOLATED state**

Guardian Info

from SEI log 657



READY is requested, from ROBUST_ISOLATED:

1. The isolation loops are turned off
2. **HEPI is in READY state**
3. The BIAS HOLD keeps HEPI aligned
4. Main steps 13-20 are available

HEPI **user WD trips**:

1. The isolation loops are turned off by the FE code (feature already there)
2. The BIAS HOLD keeps HEPI aligned
3. FE code prevents BIAS HOLD from being driven
4. Main steps 6-20 become available
   as soon as the user WD is manually untripped

The **IOP DacKill is tripped**:

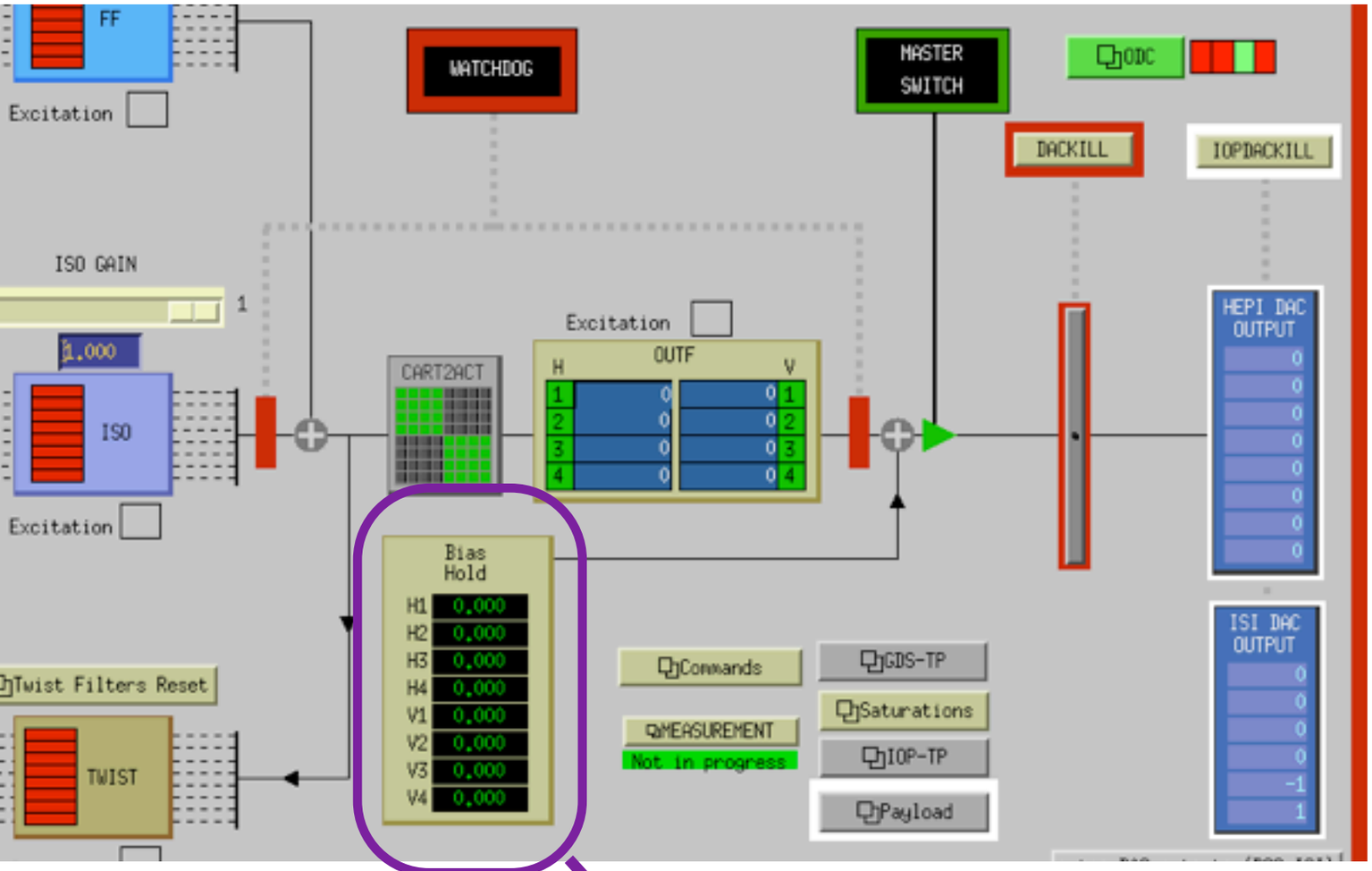
1. The **user WD trips** (standard SEI FE code behavior)
2. The Isolation loops are turned off (standard SEI FE code behavior)
3. The BIAS HOLD is reset to zero by the FE code
4. FE code prevents BIAS HOLD from being driven, until
   user WD is manually untripped
5. Main steps 6-20 become available as soon as the IOP dackill, and the
   user WD are manually untripped (standard Guardian behavior)

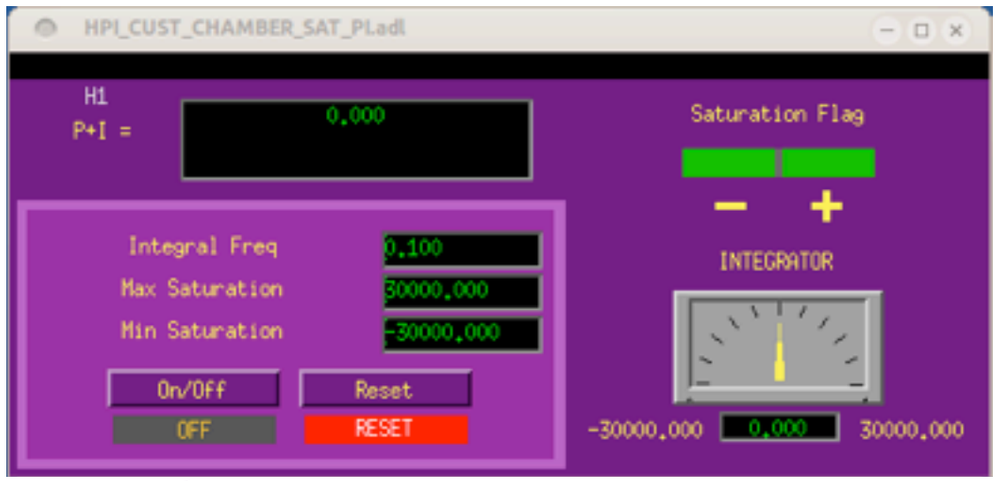
The **MS is turned OFF**:

1. The BIAS HOLD is reset to zero by the FE code
2. Main steps 6-20 become available as soon as the MS is turned ON
   (standard guardian behavior)

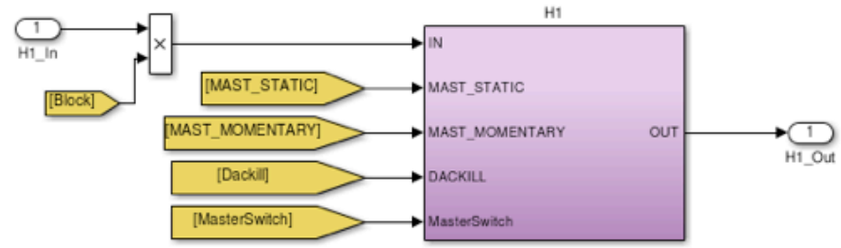
MEDM screens

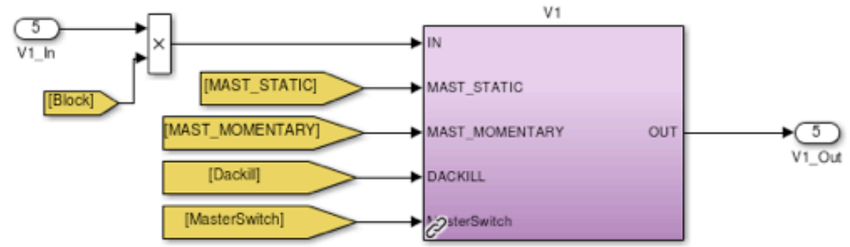
HEPI Overview screen, and 2 levels of detail

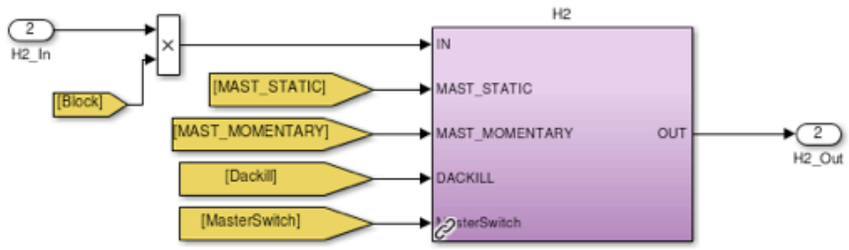


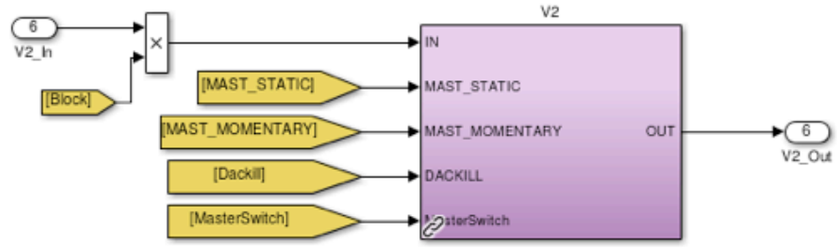


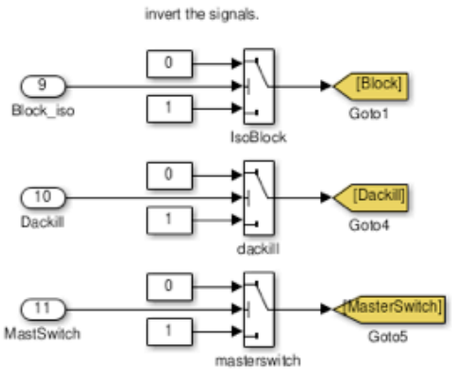
Detail of INT_HOLD

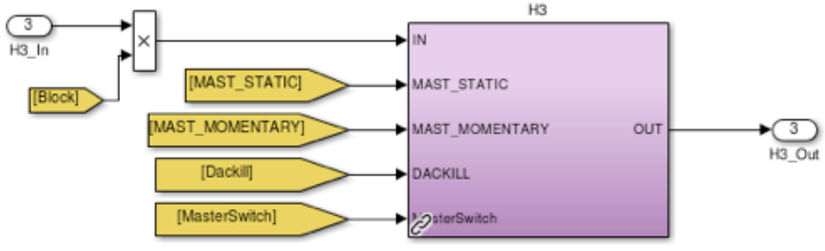


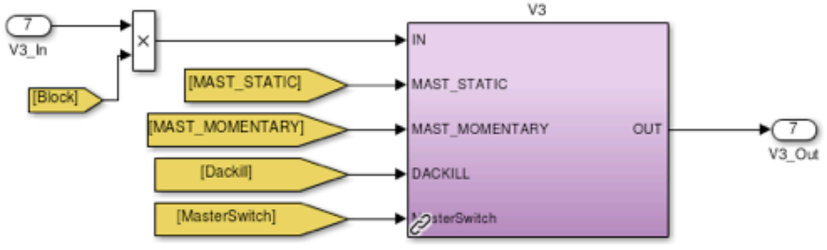


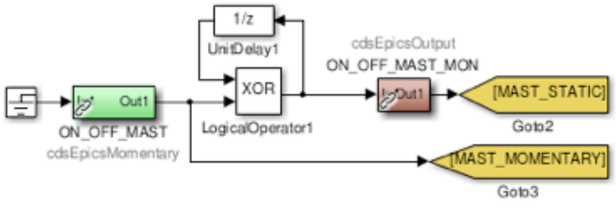


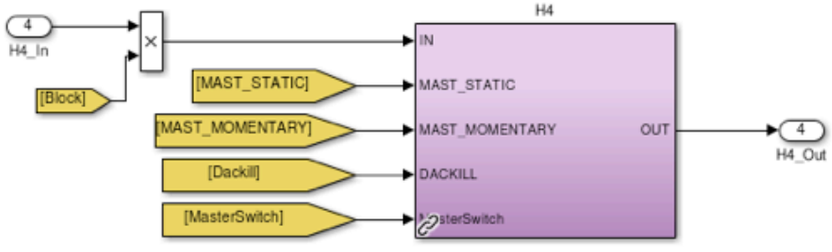


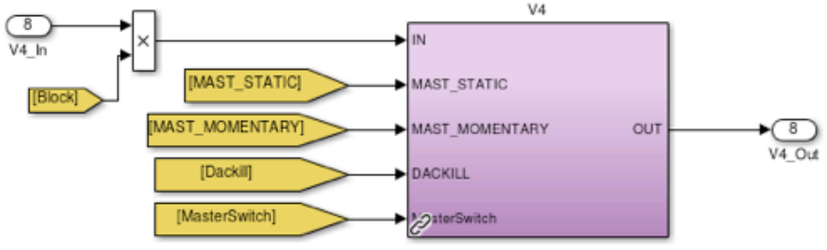




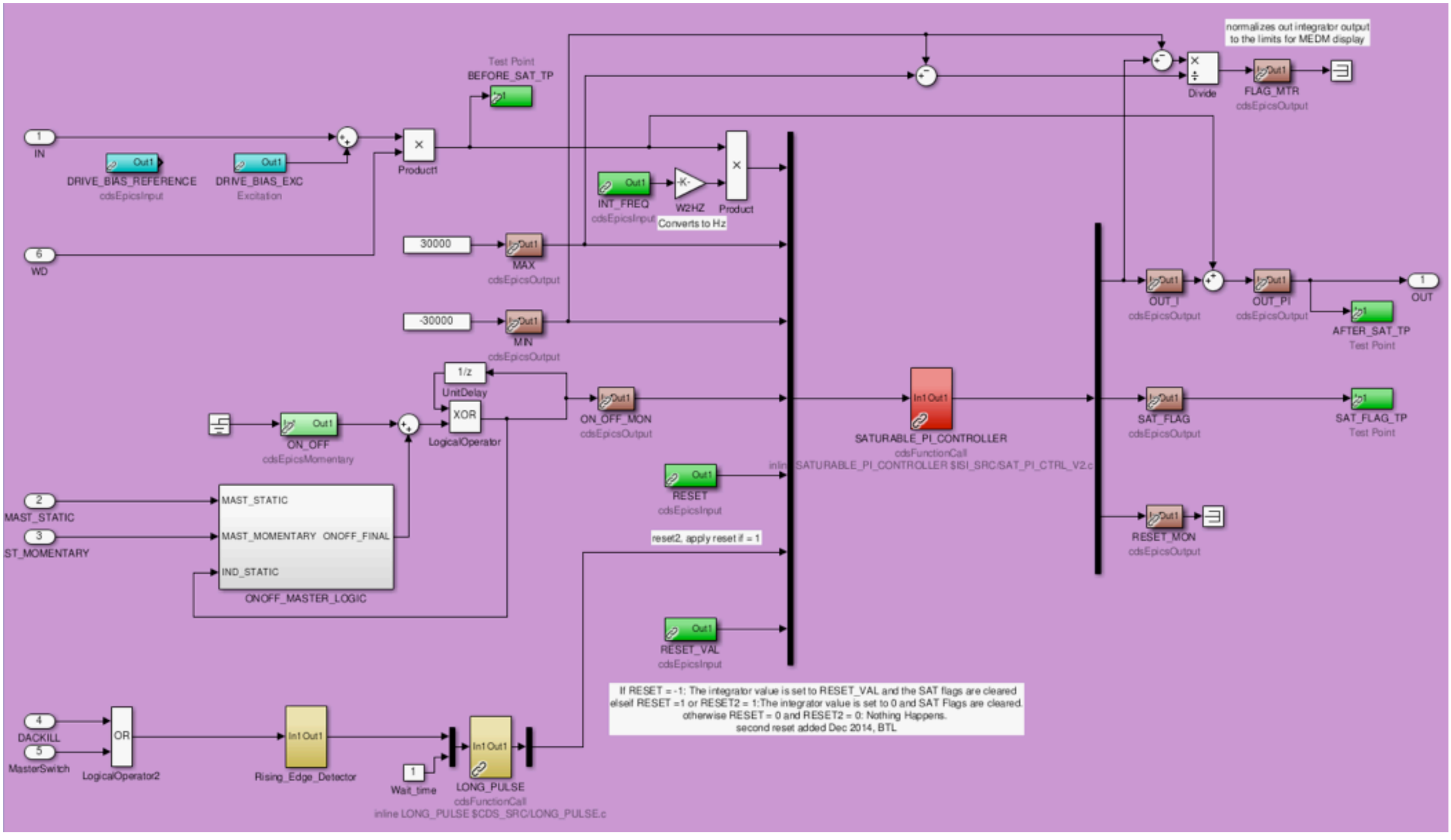








Detail of Saturable Integrator



SAT PI Code

```
/* SAT_PI_CTRL.c
 *
 * SATURABLE_PI_CONTROLLER is a standard control system integrator with the added
 * effect of saturating (or appearing as a bias) once the integrator reaches
 * client specified maximum and minimum values.
 *
 * BTL - Sept 18, 2014 _V2 adjustments:
 *
 * This function will add the measured input/cycle_rate at each cycle to
 * measured_input_sec, hence the units of measured_input_secs is
 * count_seconds. The output of the function is the error history in the same units
 *  This is a change - it used to store in input (error) - cycles
 *
 * ========================================================================
 *
 * INPUTS:
 * argin[0] = measured_input
 * argin[1] = max saturation value
 * argin[2] = min saturation value
 * argin[3] = on/off switch (non-zero is on and zero is off)
 * argin[4] = user reset switch, the -1 from this has priority over the +1 from the second reset.
 *                          (0 does nothing,
 *                         > 0, or +1 preffered, sets the integrator to zero, and clears the saturations
 *                         < 0 or -1 preffered, sets the integrator to the preset val on input 6, and
 *                                                              clears the saturations
 * argin[5] = reset switch 2, from master switch and DAC kill (0 do nothing, 1, reset)
 * argin[6] = new integrator value
 *
 * static local variables
 *    integrated_input_secs = accumulated measured input (possibly saturated)
 *    saturation_flag       = -1 or +1 for (min or max saturation)
 *
 * OUTPUTS:
 * argout[0] = measured_input_secs (count-seconds)
 * argout[1] = saturation_flag
 * argout[2] = reset_flag
 *
 * Written by Charles Celerier and Brian Lantz
 * February 2013 LIGO-T1300073-v1
 * BTL added reset 2 on Dec 5, 2014
 */

#define CYCLE_RATE FE_RATE
#define CORRECT_INPUT_NUMBER 7
#define CORRECT_OUTPUT_NUMBER 3

 void SATURABLE_PI_CONTROLLER(double *argin, int nargin, double *argout, int nargout)
{
  /* initialize the local static variables
   */
  static double integrated_input_secs = 0;
  static signed char saturation_flag = 0;

/* name the argin variables
 * many of these were const double, but BTL made them just double on Dec 8, 2014.
 */
double measured_input = argin[0];
double max_saturation_value = argin[1];
double min_saturation_value = argin[2];
double isOn           = argin[3];
double reset_flag     = argin[4];
double reset_flag2    = argin[5];
double reset_int_val  = argin[6];
double reset_flag_out = 0;


if (nargin != CORRECT_INPUT_NUMBER) {
       /* kill the process and report */
}
/* if the client wants to reset, set the error history to zero
 * and turn off the saturation flag.
 */
if (reset_flag < 0) {
  integrated_input_secs = reset_int_val;
  argout[0] = integrated_input_secs;
  saturation_flag = 0;
  reset_flag_out = -1;
} else if ((reset_flag > 0) || (reset_flag2 > 0))  {
  integrated_input_secs = 0;
  argout[0] = 0;
  saturation_flag = 0;
  reset_flag_out = 1;
}
else {
  if (isOn) {
    integrated_input_secs += measured_input / CYCLE_RATE;

    if (integrated_input_secs  <= min_saturation_value) {
      argout[0] = min_saturation_value;
      integrated_input_secs =  min_saturation_value;
      saturation_flag = -1;
    }
    else if (integrated_input_secs  >= max_saturation_value) {
      argout[0] = max_saturation_value;
      integrated_input_secs =  max_saturation_value;
      saturation_flag = 1;
    }
    else {
      argout[0] = integrated_input_secs;
      saturation_flag = 0;
    }
  }
}

/* Output flags
 */
argout[1] = saturation_flag;
argout[2] = reset_flag_out;

return;
}
```