Sensor Correction Update to the ISI models
Implement a Fader and Allow Common mode control

Brian Lantz
T1800414-v3, Oct 2018

1 Summary

This document describes the installation, function, and use of the ISI Sensor Correction block for correction from the ground sensors to stage 1 (for the BSC-ISI) or ground sensors to the platform (for the HAM-ISI). The general ECR for all of the Common-mode earthquake control is ECR E1800268 and FRS ticket 11553. The common-mode earthquake control also involves changes to how the ground motion signals are processed and distributed. Those changes are discussed in T1800420.

2 Major Changes

2.1 Channel Fading

The Channel Fader is a new tool for smoothly changing between channels. It is similar to the blend-switching, except that now each possible filter is run all the time, and the Fader simply ramps from one running filter to the next. This means the switching is faster because, unlike the blend-switcher, there is no start-up transient associated with selecting a new filter, and you only have to transition 1 time per fade (the blend-switcher had to switch back to the CUR filter). This requires more computing time per cycle and more memory, but the ISI models still complete with plenty of time. This also means that each sensor correction filter can use an entire filter bank, rather than just a single module.

For the Sensor Correction, this should be a big improvement, because one be able to choose between running correction filters, rather than having to turn one off and turn another one on to change the correction filter.

2.2 Ground motion calculation

The ground motion signals are no longer calculated by each model. Going forward, each ISI and HPI model will have the calibrated ground motion signals as inputs. These signals are calculated by the SEI-PROC model in the corner station, and the top-level ISI-ETMX and ISI-ETMY models in the end stations. **These signals all need to be calculated and distributed before this model update is implemented.**

The new inputs to the ISI models are:

inputs 1-3 Normal ground motion channels. These are the X, Y, and Z channels used to monitor the ground motion during normal operation for the building. The BRS-corrected channels are used if they exist. For LHO End-X, these would be SUPER-X, STS-Y, and STS-Z.

inputs 4-6 Common-mode motion for X, Y, and Z. These are calculated by the SEI-PROC model in the corner station.

inputs 7 & 8 These are the uncorrected ground motion signals for X and Y. If the BRS is used, these channels will be different than those for 1-2. If there is not a tilt-corrected channel, then these will be the same as those used for 1-2. These back-up channels are available for use if the BRS is not working correctly. For LHO End-X, these would be STS-X and STS-Y.

input 9 Spare. Left for future use. Having nine inputs keeps the input numbering the same.

2.3 STS outputs from master model have been removed

Since the ground motion signals are no longer calibrated within isi2stagemaster.mdl, the outputs of that have been removed. This will require re-wiring the outputs at the top level of the BSC models.

2.4 Weiner Filtering

The Weiner filters still exist, and have been extended slightly. For X, Y, and Z there are now filters for the Normal ground signal, the Uncorrected ground signal, and the local mode motion.

2.5 RX, RY, and RZ

No changes have been made to the sensor correction for the rotational degrees of freedom. There seemed to be little motivation at the time of the writing, so Brian Lantz and Jim Warner decided to leave this unchanged for now.

3 Installation

3.1 Overview

Here is a summary of the installation steps:

1. Before you begin, make sure that the calibrated ground motion signals and common-mode signals are being calculated by the End Station ISI models (T1800420) or by the SEI-PROC model.
2. The isi/common part of the userapps svn needs to be updated.
3. In each chamber model, the new ground motion signals need to be connected, and the master library part, isi2stagemaster.mdl will be replaced with the new part isi2stagemaster_SC_2018.mdl. The wiring to the new part does change, but not much.
4. The framewriter needs to be restarted because the filter list is different. The DQ channels do not change.
5. The location of the Sensor Correction filters gets changed; the filters are the same but they need to be installed into the new locations.

6. Once the filters have been installed, set them up in the plethora of new medm screens.
7. There are a few epics channels which need to be updated and saved in SDF.
8. Finally, the guardian control for the Sensor correction needs to be changed.

3.2 SVN updates

There are several files in various folders of the seismic userapps which have been updated for the BSC-ISI. You need to SVN up the following files:

```
code update is in rev 18007
{userapps}/release/isi/common/src/
   CHANNEL_FADER.c   / this code has been updated

medm screens appear in rev 18034
{userapps}/release/isi/common/medm/bscisi/
   ISI_CUST_CHAMBER_GND_BRS.adl   / added new block to calibrate the STS
   ISI_CUST_CHAMBER_OVERVIEW.adl  / new link to the sensor correction controls
   ISI_CUST_CHAMBER_ST1_SENSCOR_MATCH_ALL_2018.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_MATCH_ALL_OLD.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_OVERVIEW_2018.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_OVERVIEW_OLD.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_SELECT_XYZ.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_WNR_OVERVIEW.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_WNR_X.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_WNR_Y.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_WNR_Z.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_X_EQ_ALL.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_X_NORM_ALL.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_Y_EQ_ALL.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_Y_NORM_ALL.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_Z_EQ_ALL.adl
   ISI_CUST_CHAMBER_ST1_SENSCOR_Z_NORM_ALL.adl

models appear in rev 18036
{userapps}/release/isi/common/models/
  isi2stagemaster_SC_2018.mdl    / new library part for the BSC-ISI
  blend_switch_library.mdl   / new library part for the sensor correction.

The models and 2 corresponding medm screens were updated in rev 18066.
```

There are several more files for the HAM-ISI update. These were committed on (not yet). The HAM-ISI updates are in rev- (not yet). The HAM and the BSC source the same fader c-code and library parts, but the HAM-ISI master model has been updated to a new master part.

```
{userapps}/release/isi/common/medm/
(not yet)

{userapps}/release/isi/common/models/
  (not yet)
```

3.3 Model updates

The inputs to the ground motion channels need to be updated, and the models have new master library parts to minimize confusion and prevent accidental updates.

3.3.1 BSC-ISI model update

The master library part for the ISI needs to be replaced.

1. Commit the working chamber model to the SVN.
2. Leave all the inputs to the master part unchanged. The number and order of the inputs to the new part are the same, although the 9 GND inputs get different signals.
3. Delete the output for the 9 STS_{A/B/C}_{X/Y/Z} signals from the master part. These outputs have been removed from the new part. These signals are now all calculated elsewhere.
4. Delete the master isi2stagemaster part.
5. Bring in the new block from the isi2stagemaster_SC_2018.mdl. Rename the block per the chamber.
6. Connect all the inputs and the outputs in the same order as the previous version (except the outputs removed earlier). The outputs are all now connected correctly.
7. Fix the GND inputs. The 9 old STS inputs have been replaced by 9 new GND inputs. For the End Station models, connect these as shown in figure 1. For the other models, these get connected to signals from the SEI-PROC model.
8. save, commit, pray, compile.

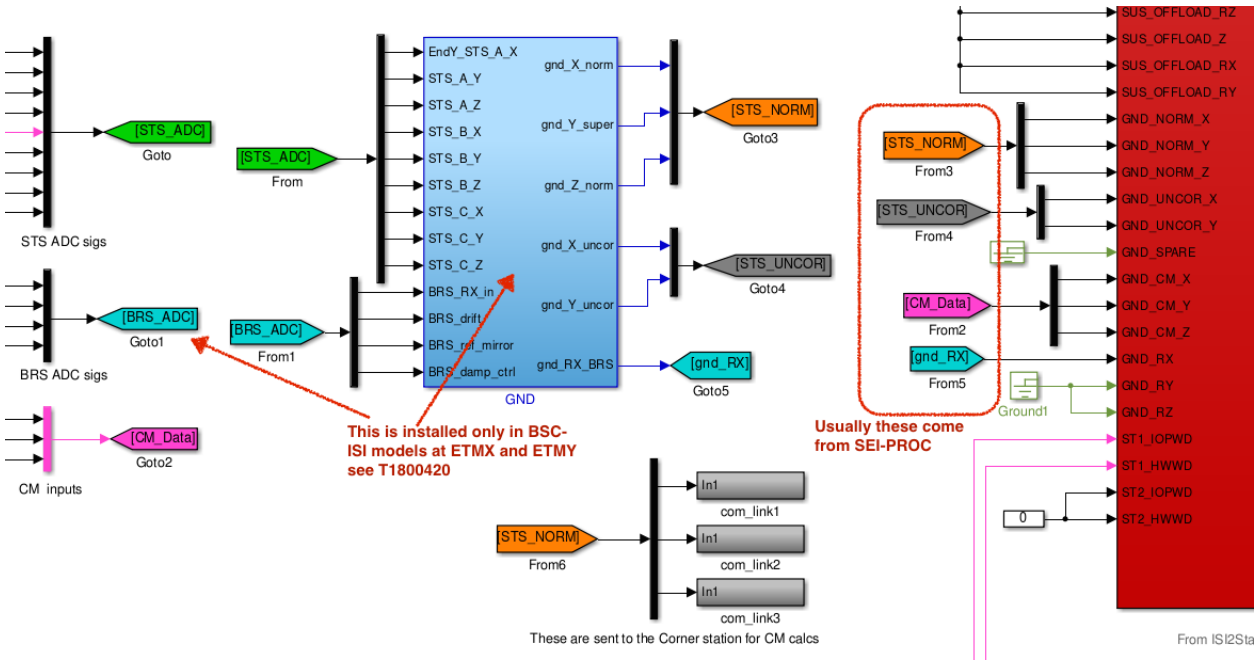

Figure 1: New ground inputs for the isi2stagemaster_SC_2018 library part.

3.3.2 HAM-ISI model update

TBD

3.3.3 HPI model update

also TBD

3.4 Framewriter restart

Oddly, the DQ channel list has not changed. The sensor correction signals which are saved are all saved from the SCSUM block, and this block has not been touched. However, the list of filters has been changed pretty dramatically, so the PAR and INI files are quite different, and the framewriter will need to be restarted.

3.5 MEDM updates

There are several new MEDM screens. The overview screen is the same except that there is a new block called 'Gnd -> St1 SENSCOR', as shown in figure 2. The new screens are described in section 4.

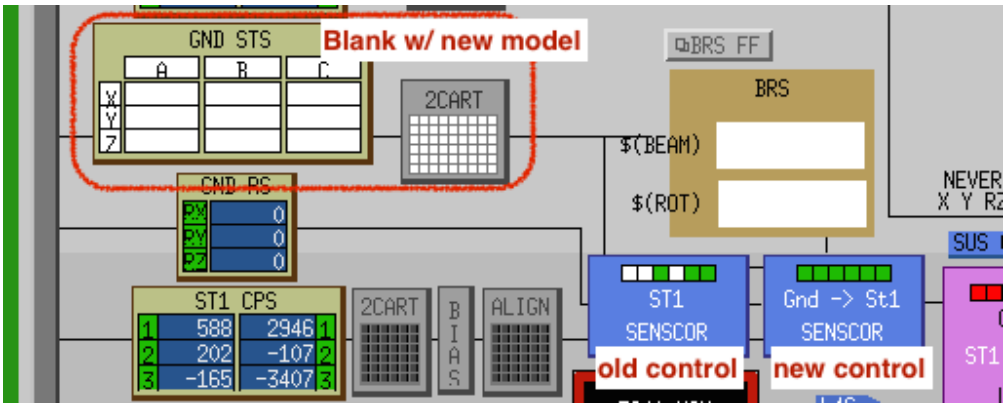

Figure 2: Change to the BSC-ISI overview screen. There is a new GND -> St1 SENSCOR part, which is linked to the new sensor correction screens. The ground STS calibrations are all gone, as is the Ground STS selection matrix, so these displays are all white.

3.6 Filter arrangement

3.6.1 Foton filters

Several Foton filters need to be installed for each chamber. First, the filter which is most commonly used for sensor correction, e.g. CML_BB_SC should be installed in the first filter bank for the Normal ground signals, and also in the first filter bank for the uncorrected ground filters. Open the foton file for the chamber, and copy the filter *into the first module* for:
$(CHAMBER)_ST1_SENSCOR_{X, Y, and Z}_NORM_FILT1, and
$(CHAMBER)_ST1_SENSCOR_{X, Y, and Z}_UNCOR_FILT1

If there is a second Sensor correction filter which is currently used, that should be installed into $(CHAMBER)_ST1_SENSCOR_{X, Y, and Z}_NORM_FILT2.

There is (soon to be) a new sensor correction filter for use during earthquakes called EQ_trial1. That filter will be in the seismic userapps as soon as I make it. When it is, install EQ_trial1 into the first module of the 3 filters
$(CHAMBER)_ST1_SENSCOR_{X, Y, and Z}_EQ_FILT1

3.6.2 Epics Parameters

We'll talk about setting up the various epics variables shortly. Remember to put the filter settings, and the new epics vars for ramp time and initial sensor correction filter under SDF control.

There are 9 new filters for each of X, Y and Z. These are
...ST1_SENSCOR_{X, Y, and Z}_NORM_FILT1 to NORM_FILT4,
...ST1_SENSCOR_{X, Y, and Z}_EQ_FILT1 to EQ_FILT3, and
...ST1_SENSCOR_{X, Y, and Z}_UNCOR_FILT1 and UNCOR_FILT2.

The new ramp time is ...ST1_SENSCOR_{X, Y, and Z}_TRAMP.

The Initial Filter is selected by ...ST1_SENSCOR_{X, Y, and Z}_INIT_CHAN.

3.7 Guardian Changes

To select a new sensor correction filter, update the value of the ...ST1_SENSCOR_{X, Y, Z}_NEXT_CHAN epics variable. The ramp starts automatically when this value changes. The front-end prevents changes to this variable during a ramp. The values 1-9 select the 9 possible filter banks.
1-4 select NORM_FILT1-4,
5-7 select EQ_FILT1-3, and
8 & 9 select UNCOR_FILT1 and 2.
Setting NEXT_CHAN to 0 ramps the sensor correction output to 0.

The NEXT_CHAN epics variable is a bit special. It gets set to the value of INIT_CHAN when the code starts (cycle 2) and it can not be changed while the filter is transitioning. You also can not set it outside the range of 0-9. I've attached the code to achieve this later in the document.

Please add docs for the guardian control here.

4 Using the new Sensor Correction

4.1 Setting up the Sensor Correction

When the foton file is ready, install it and then set up the Sensor Correction filters. From the overview screen, press the new GND -> St1 SENSCOR module. This opens the new screen. Figure 3 shows the new sensor correction overview screen. Each DOF has 9 filter banks: 4 for normal operation, 3 for common-mode control during earthquakes, and 2 additional backup filters which can be used if the BRS is not working correctly. These have descriptive names, but are just numbered 1-9 for switching. To setup

4.2 Fade Time

I suggest starting with a fade time of 30 seconds. I've tried 30 seconds for blend time on long duration filters, and the simulations look fine, see SEI log 890, Brian Lantz, from Dec. 7, 2015. This is also comparable with the characteristic period of the surface waves. Feel free to try shorter transitions, and let us know how it goes.

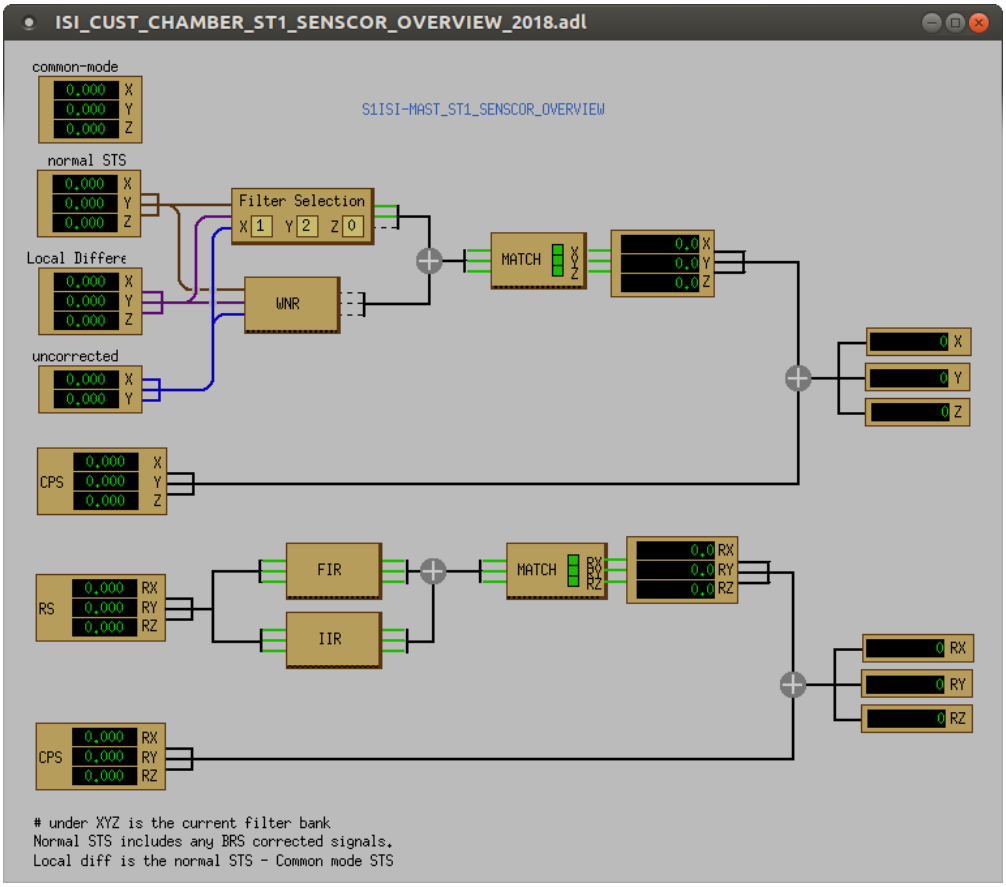

Figure 3: Updated Sensor Correction overview screen. The numbers on the 'Filter Selection' block refer to the channel which is in use. 0 means it is off.

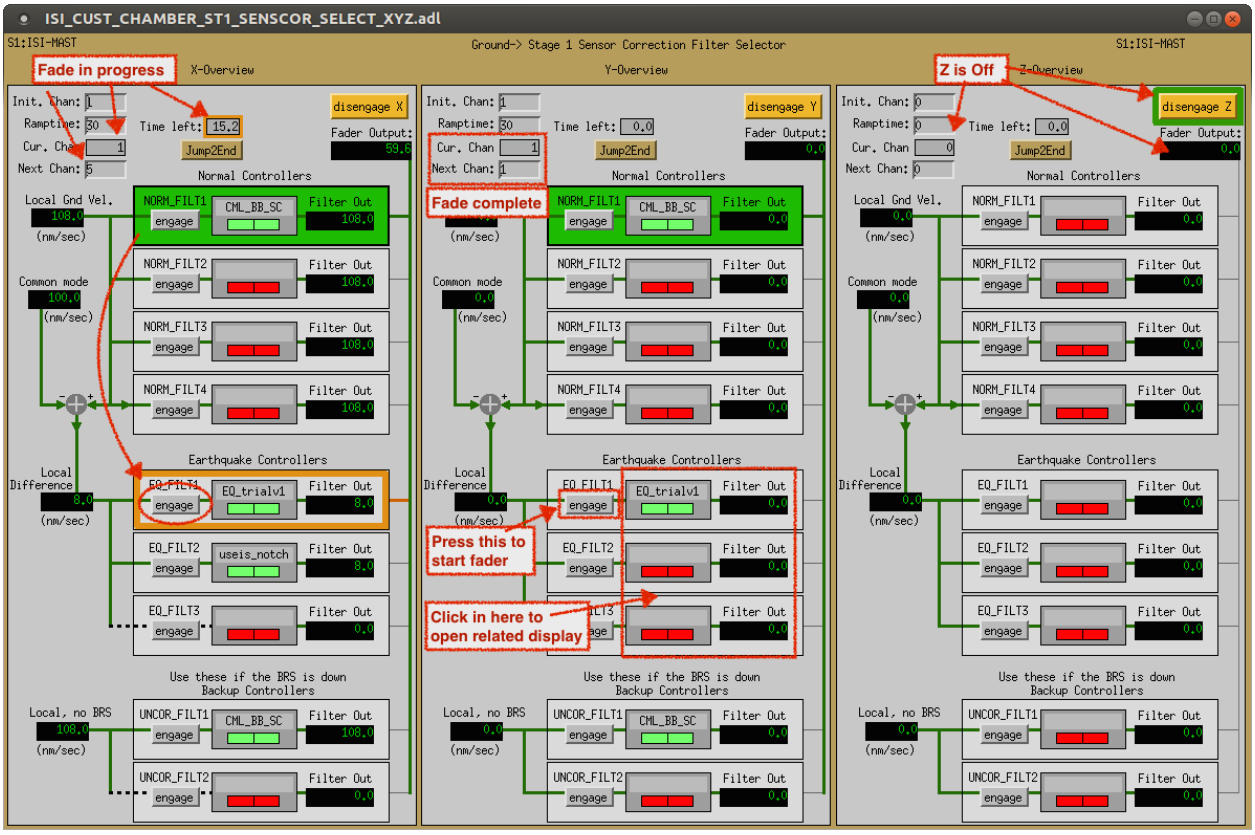

Figure 4: Selection Screen for the Sensor correction. The filters for each DOF are selected by pressing the 'engage' button. You can also select a new filter by typing a number in the 'Next Chan' variable. The filters are numbered 1-9 where 1= NORM_FILT1 and 9 = UNCOR_FILT2. Here, the X DOF has channel 1, NORM_FILT1, selected, but is part way through a transition from 1 to 5 (EQ_FILT1). Y running happily on channel 1, and Z is off. At the top of each bank, you need to set the initial channel to use. 0 is fine, it means the sensor correction starts OFF. Also set the ramp times. 30 seconds is fine.

5 Additional MEDM screens

There are a few other medm screens which have been created.

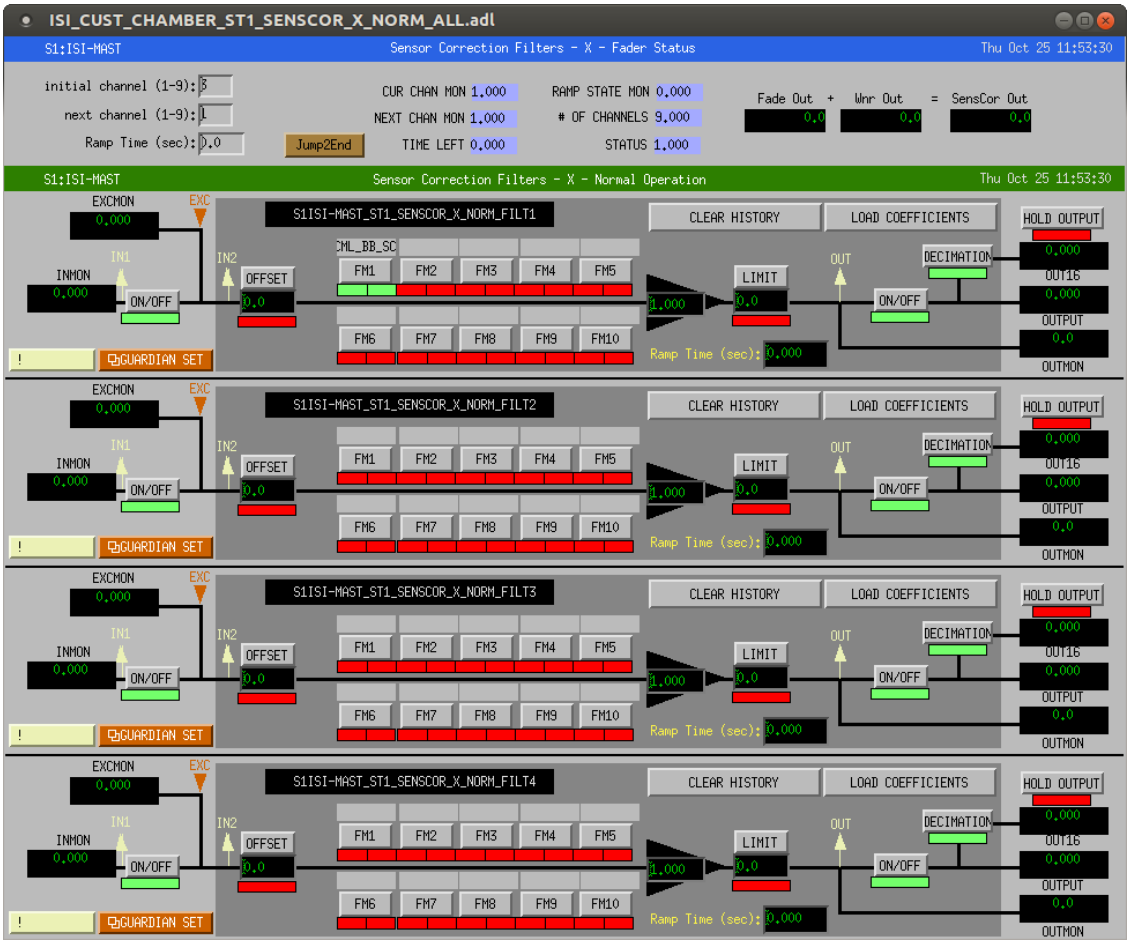

Figure 5: New screen showing additional info on the fader status and the full filter set for the Normal filters.

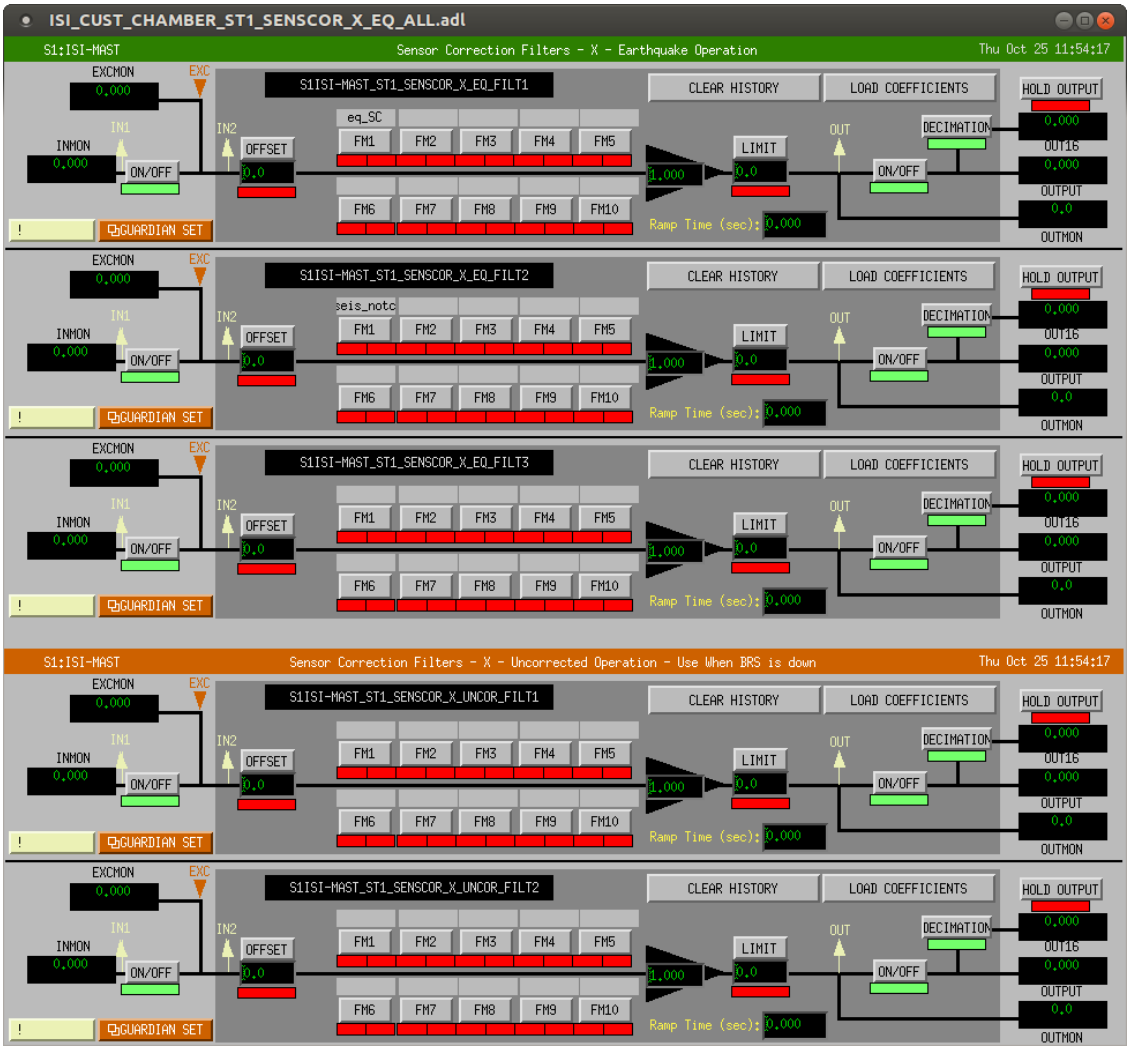

Figure 6: New screen showing full filter set for the Earthquake and the Uncorrected (backup) filters.

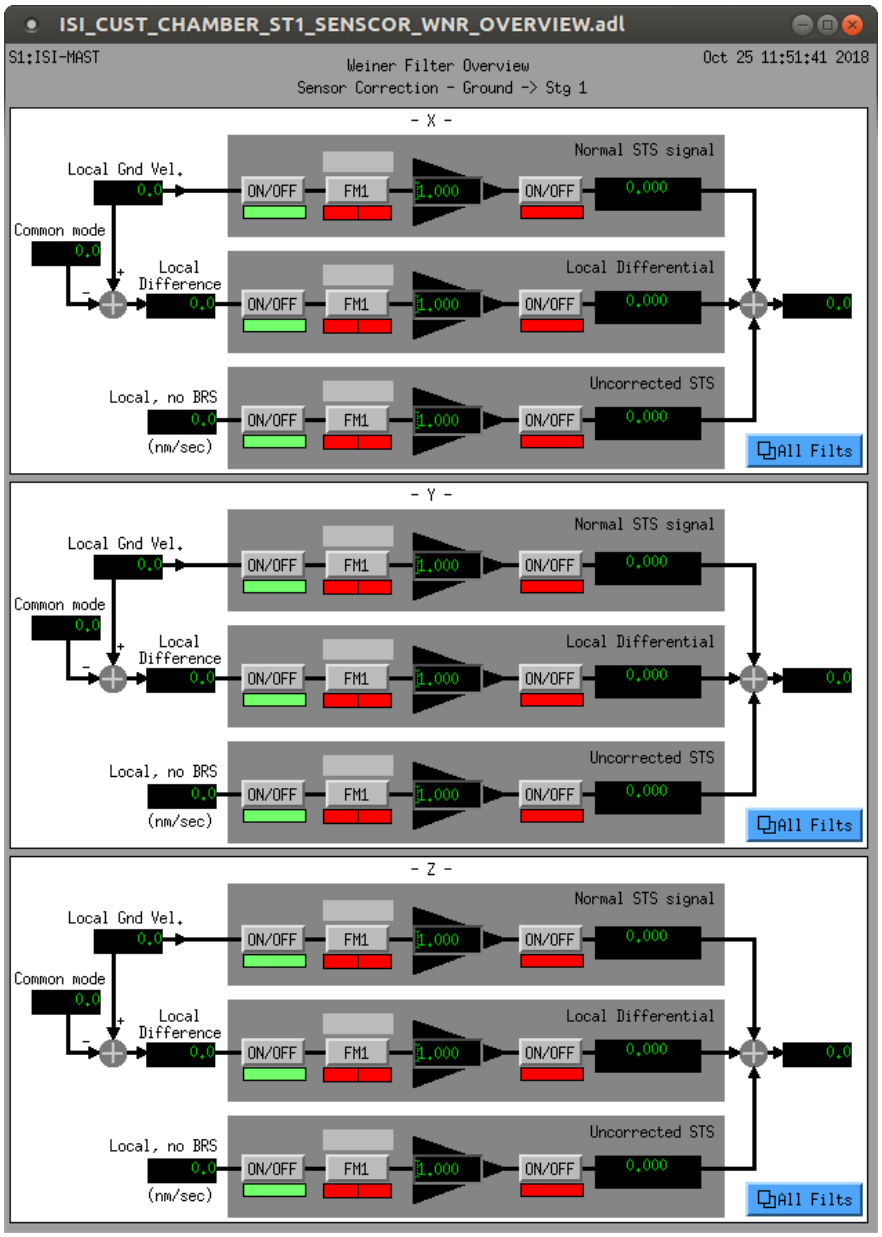

Figure 7: New Weiner Filter Sensor Correction overview screen

6 Simulink diagrams

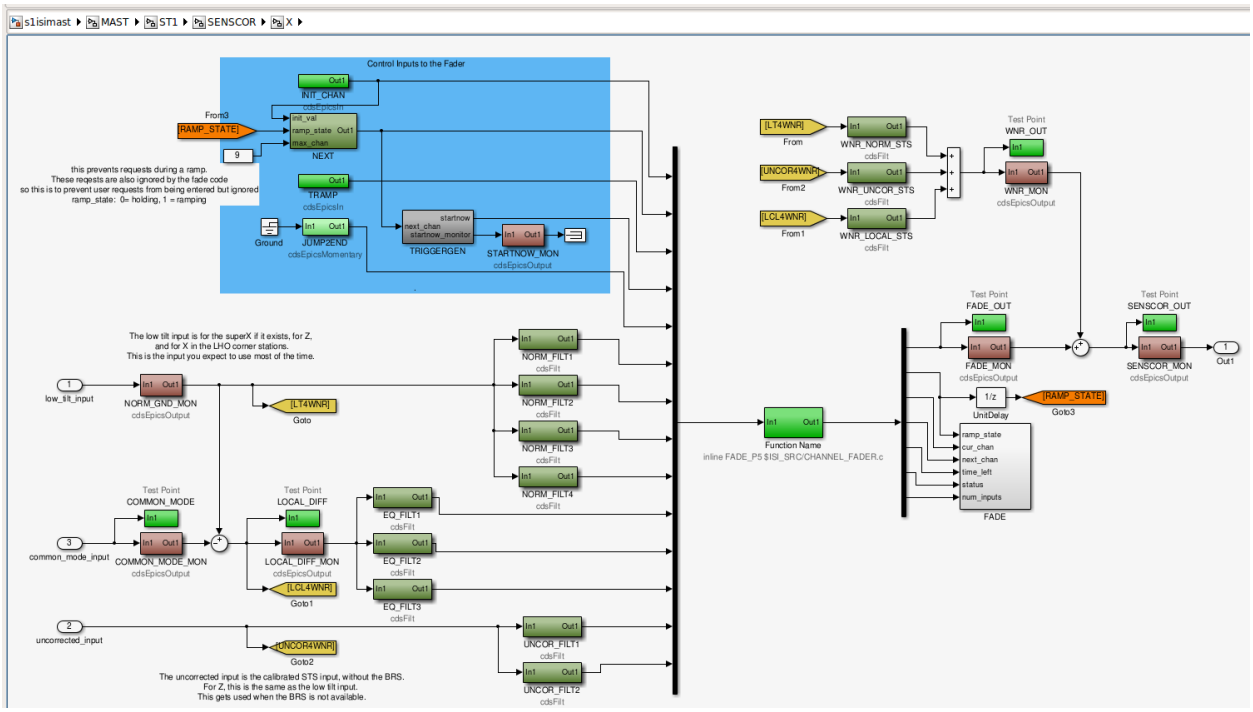

Figure 8: New Sensor Correction diagram (for x). Channel selection is on the top left in the blue, note that the NEXT block has some funky stuff going on. The 9 signals to select between come in on the left. The new weiner filters are top right. The FADE block (bottom right) contains several status monitors for the CHANNEL_FADER code (center). This part is the X_SENSCOR part from the new blend_switch_library.mdl

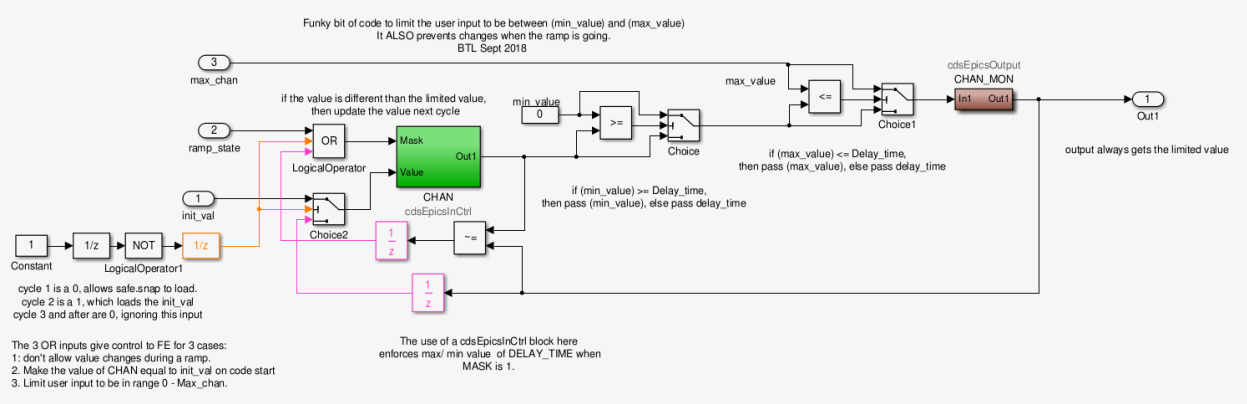

Figure 9: Code for the NEXT block. This is based on the epics with limiter, T1800278, and also includes a 2 cycle thing to set the initial value to match the INIT_CHAN. This needs to be 2 cycles, because the safe.snap value will be loaded at some point. This code allows it to be loaded on cycle 1, then replaced on cycle 2. When the c-code is ramping, this epics part also just gets the value from last cycle. These things are done to keep the NEXT_CHAN epics variable in sync with the c-code. The code switches when the NEXT_CHAN value updates, so the sync is important.

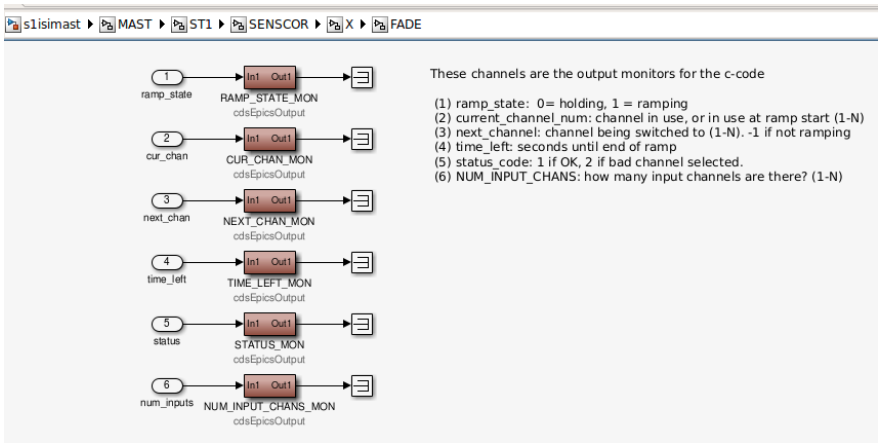

Figure 10: These are the epics channels which monitor the state of the FADE code

7 Source Code

There is a new piece of c-code called **CHANNEL_FADER.c**. It lives in userapps/trunk/isi/common/src/. Testing of the code was done by Brian Lantz in March 2018, please see SEI log 1320 for details.

The code is:

```
/* CHANNEL_FADER.c Function: FADE_P5.c
 *
 * Transition smoothly (fade) from one input channel to another.
 *  Used to switch between sensor correction filters or between blend filters
 *
 * Inputs:
 *
 * (0) int initial_channel: at code start, select this channel (numbered 1-N, or 0
    for OFF)
 * (1) int next_channel: this is the channel to switch to (numbered 1-N, or 0 for
    OFF).
 *      if the channel is 0 = fade output to 0. fader is running, but output is 0.
 * (2) double T_ramp: ramp time in seconds
 * (3) int start_now:  (epics momentary) when this channel is 1, start a transition
 * (4) int jump_to_end: (epics momentary)  when this channel is 1, go to ramp end
 * (5 - end) double: input channels to switch between
 *
 * Outputs:
 *
 * (0) double output_val: the current output value
 * (1) double ramp_state: 0 = holding, 1 = ramping
 * (2) int current_channel_num: channel in use, or in use at ramp start (1-N)
 * (3) int next_channel: channel being switched to (1-N). -1 if not ramping
 * (4) double time_left: seconds until end of ramp
 * (5) int status_code: 1 if OK, 2 if bad channel selected.
 * (6) int NUM_INPUT_CHANS: how many input channels are there? (1-N)
 *
 * note: current_channel_num and target_channel are the internal vars used to track
 *       what is going on. These are only updated at the start and end of
    transitions
 *
 * Authors: BTL
 * March, April 2018, adapted from ramp_bias.c
 * see T1300510 for a derivation of the ramp - BTL June 12, 2013
 *
 * pseudo_code
 * start with no ramp, select the current_channel_num = initial_channel as the
    channel to use
 *
 * if ramping
 *   continue the ramp unless the jump_to_end is selected
 *   at end of ramp, select RampState = HOLDING set current_channel_num =
    target_channel
 * else (ie if not ramping)
 *   if the start_now is selected and ramptime > 0 start a new ramp (set state to
    RAMPING)
 *       elseif start_now is selected and ramptime = 0, just switch.
 *     current_channel_num is not changed, make target_channel = channel_next.
 *        we only read the channel_next at the ramp start, and
 *        we stick with target_channel until the end of the ramp.
 * end
```

```
 * Set the outputs:
 * if HOLDING, output = input from current_channel_num
 * if RAMPING output = ramp * input from current_channel_num + (1-ramp) * input from
     target_channel
 */

#define MODEL_RATE FE_RATE
#define MIN_CHANS 0      // if channel is 0, then use 0 as input, so output fades to
    0
#define MAX_CHANS 20     // maximum number of allowed input channels (N<=this number)

#define NUM_CONT_PARAMS 5  // number of inputs which are used as control parameters
#define IN_INIT_CHAN     0  // these are the control input channel numbers (for
    convenience)
#define IN_NEXT_CHAN     1
#define IN_TRAMP         2
#define IN_START_NOW     3
#define IN_JUMP_2_END    4

#define OUT_SIGNAL       0  // these are the output channel numbers (for convenience)
#define OUT_RAMP_STATE   1
#define OUT_CUR_CHAN     2
#define OUT_NEXT_CHAN    3
#define OUT_TIME_LEFT    4
#define OUT_STATUS       5
#define OUT_NUM_CHANS    6

#define ALL_GOOD_CODE 1             // status codes for the user;
#define BAD_CHANNEL_REQUEST_CODE 2
#define BAD_NUMBER_CHAN_CODE 3

#define INITIAL_VALUE 1             // these are the end values for the ramp (from 1
    down to 0);
#define FINAL_VALUE   0
#define MIN_RAMP_TIME 0.001      // ramptime > 0 but < min_ramp_time will be set to
    this value
#define MAX_RAMP_TIME 100.0       // maximum time in seconds.

typedef enum {HOLDING, RAMPING} RampStates;


void FADE_P5(double *argin, int nargin, double *argout, int nargout){
        static int RampTimer = 0;  // How far along the ramp are we, in cycles
        static int TotalRampCycles = 0;  // Number of cycles in the ramp
        static RampStates CurrentState = HOLDING;
        static int FirstCycle = 2;  // 1 or more will initialize things, 1 doesn't
            seem to work, so try 2 cycles.

    static int current_channel_num; // this is the number (1-N) of the input chan to
        send out
    static int next_channel_num;  // channel to switch to
    static int NUM_INPUT_CHANS; // assumes that nargin starts at 1, not 0
    static bool major_error = false;  // used for error monitoring;
    int   requested_channel;  // this is what the user requested. check it before
        using it.
    float requested_tramp;    // these are the other control inputs;
    float  requested_start;
    float  requested_end;
    static int error_code = ALL_GOOD_CODE;
```

```
double ThisOutput;
double current_channel_value;
double next_channel_value;

    static double Tramp;  // ramptime (sec) read only on new ramp start;
    static double RpC[6];  // these are the polynomial Ramp Coefs.
    double Xdiff;  // Total change for the ramp
    double Vmax;   // max velocity, computed from dX and dT
    double tt;     // time from ramp start, but scaled as -T/2 -> T/2.
double ThisRampVal;  // value of the ramp (from 1 down to 0)
    double time_left;   //number of seconds left in the ramp. for the GUI
int monkey = 1;

// on code start, get the initial channel to use, and check that number of
    inputs is OK
// this requires 2 cycles, because the NEXT_CHAN epics var is set to the
    INIT_CHAN value
// by the simulink diagram code. that assignment takes 2 cycles,
// first is to let the epics code set NEXT_CHAN to the value in safe.snap
// second to set it to the correct val.

//    if (FirstCycle >=1) {
if ((FirstCycle == 2) || (FirstCycle ==1)) {
    FirstCycle--; // needs 2 cycles to get epics vars all set.
    NUM_INPUT_CHANS = (nargin - NUM_CONT_PARAMS);
    if ((NUM_INPUT_CHANS > MAX_CHANS) || (NUM_INPUT_CHANS < MIN_CHANS)) {
        major_error = true; // this is a fatal error. just skip to the end
    } else {
        current_channel_num = argin[IN_INIT_CHAN];
        if (current_channel_num < MIN_CHANS)   {current_channel_num = MIN_CHANS
            ;}
        if (current_channel_num > NUM_INPUT_CHANS) {current_channel_num =
            NUM_INPUT_CHANS;}
        next_channel_num = current_channel_num;
        //set the initial outputs.
        if (current_channel_num == 0) {current_channel_value = 0;} else {
            current_channel_value = argin[current_channel_num + NUM_CONT_PARAMS
                - 1];}
             ThisOutput = current_channel_value;
             time_left = 0;
             // so we have vals for ThisOutput, CurrentState,
                 current_channel_num, next_channel_num,
             // time_left, error_code, and NUM_INPUT_CHANS
        }
} else if (major_error == true) {
    //just skip to the end, the outputs are set there

} else {
    // this is the normal piece of the code which we expect to run on every
        normal iteration
    // part 1:
    // start by just reading ALL various control parameters and giving them
        useful names
    requested_channel  = argin[IN_NEXT_CHAN];
    requested_tramp    = (double) argin[IN_TRAMP];
    requested_start    = argin[IN_START_NOW];
    requested_end      = argin[IN_JUMP_2_END];
```

```
// part 2:
// now we examine the state and the inputs and decide what to do
// in this piece of the code, we will set the states and update the ramp co-
    effs.
// but we do not calculate the outputs yet. that is in part 4.


if ((CurrentState == RAMPING) && (requested_end == 1)) {
    // abort the current ramp and jump to the end
    CurrentState  = HOLDING;
    current_channel_num = next_channel_num;
    time_left       = 0;
} else if (CurrentState == RAMPING) {
    // continue the ramp. this will ignore new requests until the ramp is
        complete.
} else if (requested_start == 1) {
    // start a new ramp
    // first, be sure the requested channel is OK
    if ((requested_channel > NUM_INPUT_CHANS) || (requested_channel <
        MIN_CHANS)) {
        // the user has selected an invalid channel. don't switch.
        error_code  = BAD_CHANNEL_REQUEST_CODE;  // bad channel selected;
        // so we are not going to accept the request.
        // continue holding, but set the code to bad.
    } else if (requested_tramp <= 0.0 ){
        // valid channel, but ramp time of 0 (or negative):
        // just switch without ramping
        error_code    = ALL_GOOD_CODE;
        CurrentState = HOLDING;
        current_channel_num = requested_channel;
        next_channel_num    = requested_channel;
    } else {
        // user has selected a valid channel and finite ramp time
        // start the switching process
        error_code    = ALL_GOOD_CODE;
        next_channel_num = requested_channel;
        // now set up the ramp.
        Tramp     =  requested_tramp;
        if (Tramp < MIN_RAMP_TIME) { Tramp = MIN_RAMP_TIME; }
        if (Tramp > MAX_RAMP_TIME) { Tramp = MAX_RAMP_TIME; }
        RampTimer = 0;
        TotalRampCycles = (int) (MODEL_RATE * Tramp);
        CurrentState  = RAMPING;
        Xdiff = (double) FINAL_VALUE - INITIAL_VALUE;
        Vmax = (1.875) * Xdiff/Tramp;
        // RC are the Ramp Coefficients
        RpC[0] = INITIAL_VALUE + (0.5 * Xdiff);
        RpC[1] = Vmax;
        RpC[2] = 0.0;
        RpC[3] = (-2.6666666667/(Tramp* Tramp)) * Vmax;
        RpC[4] = 0.0;
        RpC[5] = (3.20/(Tramp*Tramp*Tramp*Tramp)) * Vmax;
    }  // end of the new ramp section
} else {
    // we are just holding. dont do anything;
}

// part 3: if we are now in the ramping state, update the ramp parameters
if (CurrentState == RAMPING){
```

```
                    RampTimer++;
                    // make this back into a time which goes from -T/2 to +T/2;
                    tt = (double) 2*RampTimer - TotalRampCycles;
                    tt = (0.5 * tt)/(1.0 * MODEL_RATE);  // cast to double
                        before the divide
                    // RC[5]*tt^5 + RC[4]*tt^4 + ... RC[0]
                    ThisRampVal = ((((RpC[5]*tt + RpC[4])*tt + RpC[3])*tt + RpC
                        [2])*tt + RpC[1])*tt + RpC[0];
                    if(RampTimer >= TotalRampCycles){
            // the ramp is done! update the ramp state
                        CurrentState     = HOLDING;
            current_channel_num = next_channel_num;
            // next_channel_num     = 0;  // no - just leave this at the current
                channel
            time_left       = 0;
                    } else {
                        CurrentState = RAMPING;
            time_left     = Tramp - (RampTimer / (1.0 * MODEL_RATE));
        }
    } else {
        // nothing to do in the holding state
    }

    // part 4: update the main output based on the ramp state.
    if (CurrentState == RAMPING) {
        // here is where we do the mixing of the two input channels
        if (current_channel_num == 0) {current_channel_value = 0;} else {
            current_channel_value = argin[current_channel_num + NUM_CONT_PARAMS
                - 1];}
        if (next_channel_num == 0) {next_channel_value = 0;} else {
            next_channel_value     = argin[    next_channel_num + NUM_CONT_PARAMS
                - 1];}
        ThisOutput = ThisRampVal * current_channel_value + (1.0 - ThisRampVal) *
             next_channel_value;
    } else {
        // just make the output equal to the input
        if (current_channel_num == 0) {current_channel_value = 0;} else {
            current_channel_value = argin[current_channel_num + NUM_CONT_PARAMS
                - 1];}
                ThisOutput = current_channel_value;
    }
}
// part 5: update all the output parameters, this runs every cycle, including
    first and major error.
if (major_error == true) {
    argout[OUT_SIGNAL]       = -1;
    argout[OUT_RAMP_STATE]   = HOLDING;
    argout[OUT_CUR_CHAN]     = -1;
    argout[OUT_NEXT_CHAN]    = -1;
    argout[OUT_TIME_LEFT]    = 0;
    argout[OUT_STATUS]       = BAD_NUMBER_CHAN_CODE;
    argout[OUT_NUM_CHANS]    = NUM_INPUT_CHANS;
} else {
    argout[OUT_SIGNAL]       = ThisOutput;
    argout[OUT_RAMP_STATE]   = CurrentState;
    argout[OUT_CUR_CHAN]     = current_channel_num;
    argout[OUT_NEXT_CHAN]    = next_channel_num;
    argout[OUT_TIME_LEFT]    = time_left;
    argout[OUT_STATUS]       = error_code;
```

```
        argout[OUT_NUM_CHANS]      = NUM_INPUT_CHANS;
    }
}
```