The HAM ISI Watchdog
LIGO-T0900011-x0

J. S. Kissel

January 14, 2009

Contents

List of Figures

1 Introduction

This document serves as the source for all information concerning the eLIGO HAM ISI Watchdog. Section 2 defines the watchdog in a more abstract sense, where section 3 describes each portion of how the Watchdog is actually implemented in the CDS system. This includes the simulink model, the non-standard front end code, and medm interface.

2 Functionality

There are two paths which the watchdog oversees. The damping loop path is the local damping control system that uses only inertial sensors (geophones), designed to be unconditionally stable, and very robust against perturbations. The isolation loop path provides most of the seismic isolation using both displacement and inertial sensor signals. This path is only conditionally stable. The point at which the signals are blocked is just after the control filter banks, before the two paths are added together and fed to the actuators.

The watchdog triggers on user-defined thresholds of five inputs: the displacement sensors, the geophones, the ground STS-2, the actuator drive level, and payload's (suspension) watchdog. The displacement, geophones, and ground STS-2 are observed directly from the inputs into the active control system. The actuator signal is sampled just before the digital control signal is sent back to the isolation system. Finally, the payload trigger is piped in directly from payload's watchdog. All watching is done on the front end computers, which has a 2048 Hz sampling rate.

The watchdog is a four state, finite state machine. Figure 1 is a state diagram showing the flow of the machine. In state 1, the watchdog is armed, sampling each of the four signals. In this state, the digital signals for both the isolation loop and damping loop paths are allowed to pass, i.e. "enabled." If any of the signals exceed threshold, the watchdog moves to state 2 and immediately blocks the isolation path. To be clear, a single sample of any signal that exceeds threshold will trip the watchdog and block the isolation path (as opposed to the typical suspension watchdogs which trigger on the the exponentially averaged RMS values of the watched signals). State 2 is simply a "cooling period," where a pre-determined amount of time passes (we've chosen three seconds, but the period is somewhat arbitrary). The isolation loops are blocked, but the damping loops are still enabled. One hopes, that while in state 2, whatever external drive that caused the initial saturation, has been squelched by the robust damping loops. Regardless, after the time period has passed, the watchdog moves to state 3.

In state 3, the watchdog resumes checking for saturations, with the isolation loops blocked and the damping loops enabled. If there are no saturations, the watchdog remains in this state until the user manually resets the watchdog. If there are any further saturations once in state 3, then the damping loops are blocked and the watchdog transitions to state 4. Again, the saturation need only be a single sample above threshold for any signal for the watchdog to move into state 4 and block the damping path. Once entered, the only way to leave state 4 is a user reset. Indeed, state 4 still checks for saturations, so if the user asks for a reset and any saturations are still present then the watchdog will not exit state 4.

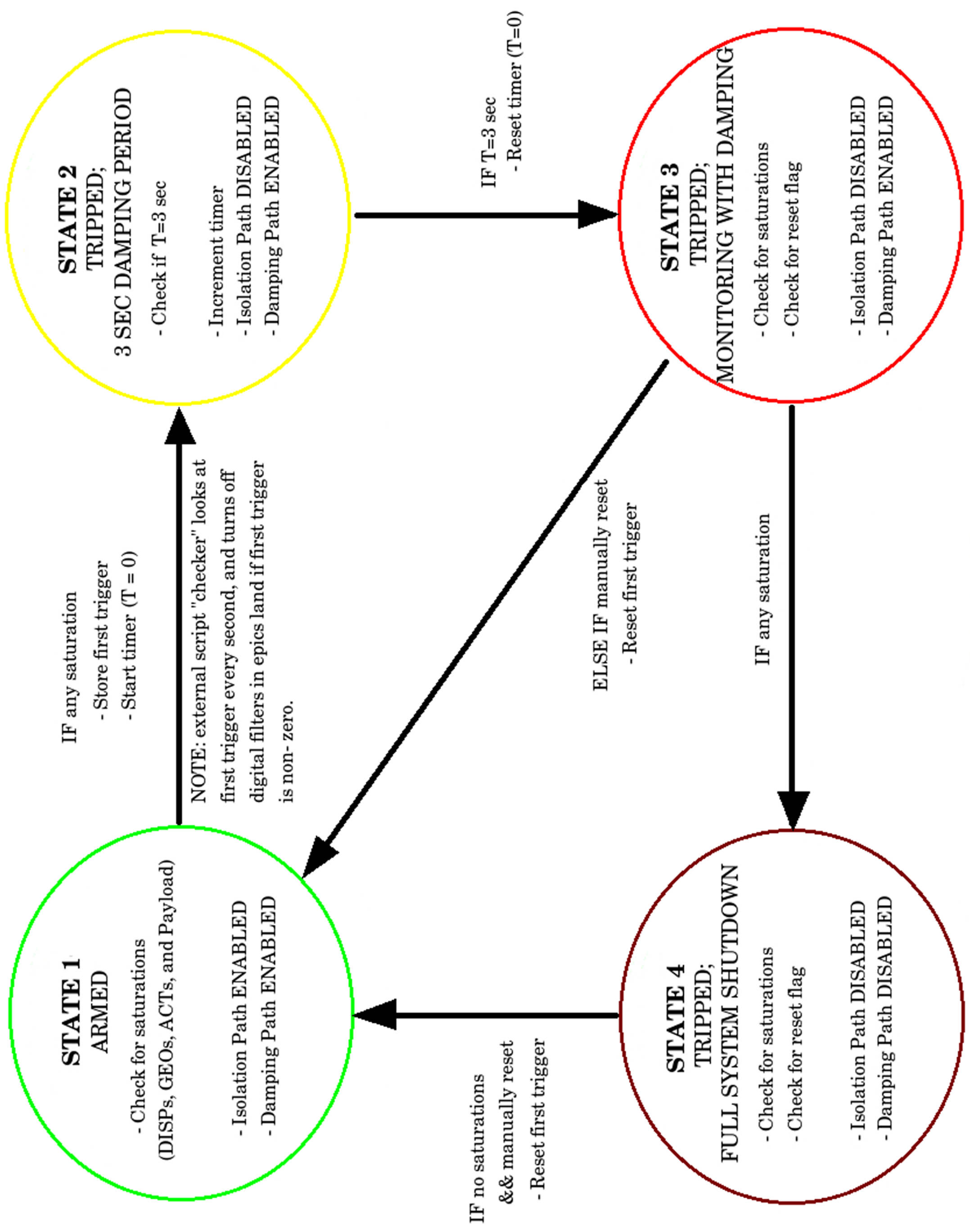


Figure 1: State diagram for HAM ISI watchdog.

3 Implementation

3.1 Simulink Model

Figures 2 - 9 show screen shots of the Simulink model for the entire ISI front end system. Both the OMC suspensions and the HAM ISI are on the same front end. The digital signals flow from left to right across the model.

Figure 2 shows the overall flow of data; there are two 32-channel ADCs, and one 16 channel DAC. The only connection relevant to the scope of this document is the connection between the ISI-SUS (OMC suspension) and ISI-OMC (HAM ISI): the last output of the SUS block is the binary flag indicating status of the SUS watchdog. This is connected into the HAM ISI's block, for use with the watchdog. The SUS watchdog flag is 1 if not tripped, and 0 if tripped. The HAM ISI watchdog has been written such that "not-tripped" is 0, so the logic is inverted between the SUS and OMC blocks at this level.

Figure 3 shows the ISI-OMC (HAM ISI) subsystem. The watchdog elements of this subsystem run along the bottom of the screen. After the inputs from the ADC, the first block encountered (WD) converts the raw displacement sensor, geophone, and STS-2 signals into saturation flags. This is done by comparing the absolute value of each signal with the user threshold using a logical greater-than operation (with the ADC signal as the first operand). The result of each logical comparison is added to form a single flag: 0 if there no saturations, non-zero if a saturation has occurred. These two flags, in addition to the payload flag, are then input into the HAMISI-WATCHDOG block, which represents the C-code OMC_HAMISIWATCHDOG.c (described in detail in section 3.2).

The five outputs of the watchdog are also integer flags. The outputs are monitored in the ISI-WDMON block (shown in Figure 6) From top to bottom, these outputs are "DAMP, CONT, STATE, FIRSTTRIG, and CURRENTTRIG." DAMP is a binary flag (0 or 1) that determines whether the damping loops should be blocked. CONT is a similar flag that determines whether the isolation is should be blocked. STATE is an integer from 1 to 4, indicating in what state the watchdog is. FIRSTTRIG is a integer whose bits indicate which sensor was the first to trip the watchdog. CURRENTTRIG is a similar integer whose bits indicate which sensor is currently saturating. STATE, FIRSTTRIG, and CURRENTTRIG are only used for monitoring purposes. However, DAMP and CONT are picked off and fed into ISI-OMC-MUXS, where the flags are used to block or enable the damping loop path and isolation loop path.

Figure 7 shows the innards of ISI-OMC-MUX5. Inputs 1 - 12 are the damping loop and isolation loop signals over which the watchdog has control. Inputs 13 and 14 are the control flags from the watchdog for damping and isolation loops, respectively. The choice blocks with constants as their first and third input multiply the control and damping loop signals by 0 or 1. If the watchdog flags are tripped, the choice clock outputs a 0, multiplying the drive signals by zero, effectively blocking the signals. If the watchdog flags are not tripped, then the choice block sends out a 1, effectively enabling the signals. Figure 8 shows the configuration of the choice block which achieves this behavior.

Finally, the blocked or enabled signals are added together in a colocated fashion and sent out to the DAC. Just before the DAC outputs, however, is a pick-off to watch the actuator drive levels. It is here where the actuator flag of the watchdog system is set, via the HAMISIDRIVEMON block, which represents the C-code described in section 3.2.

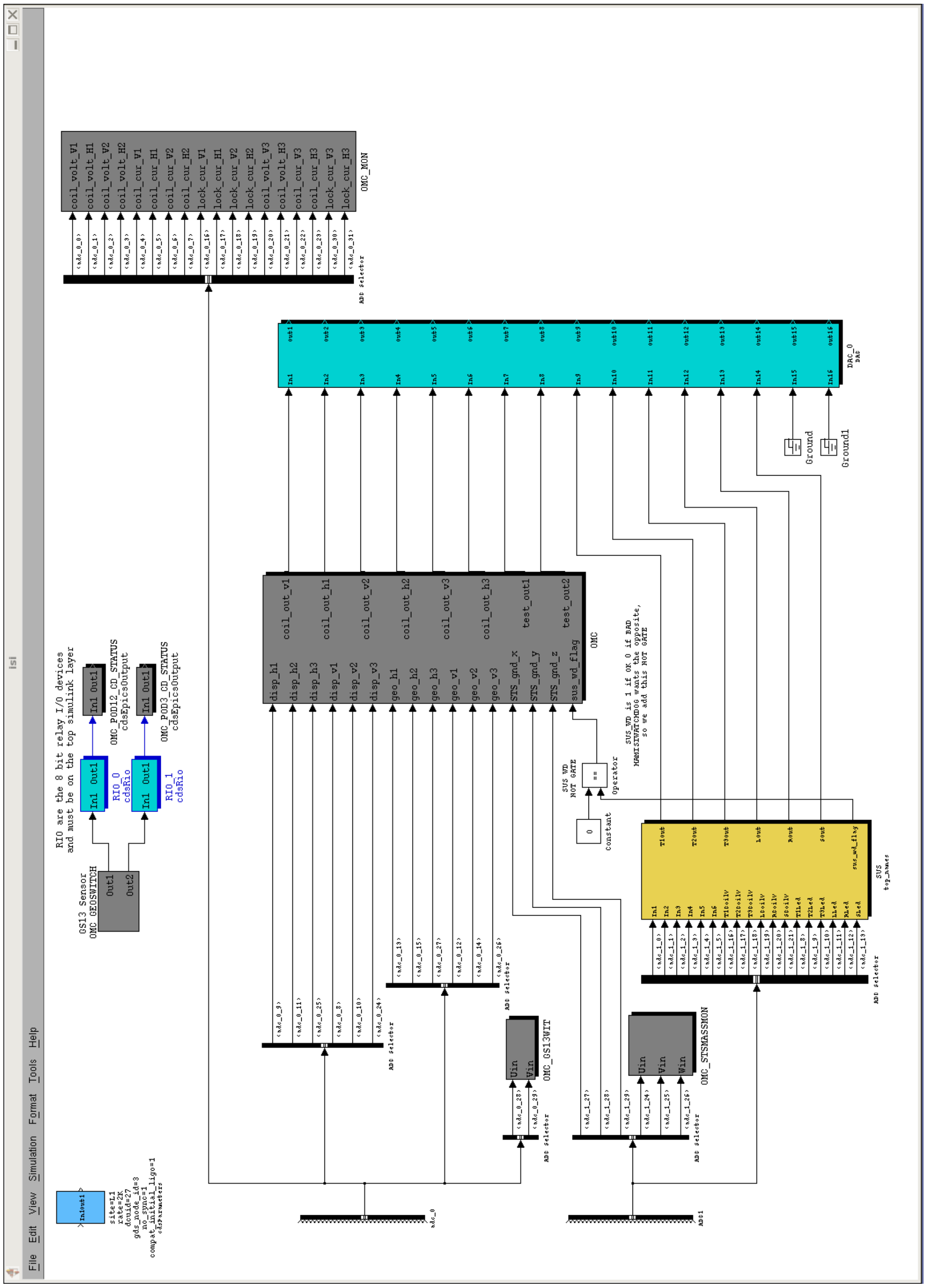
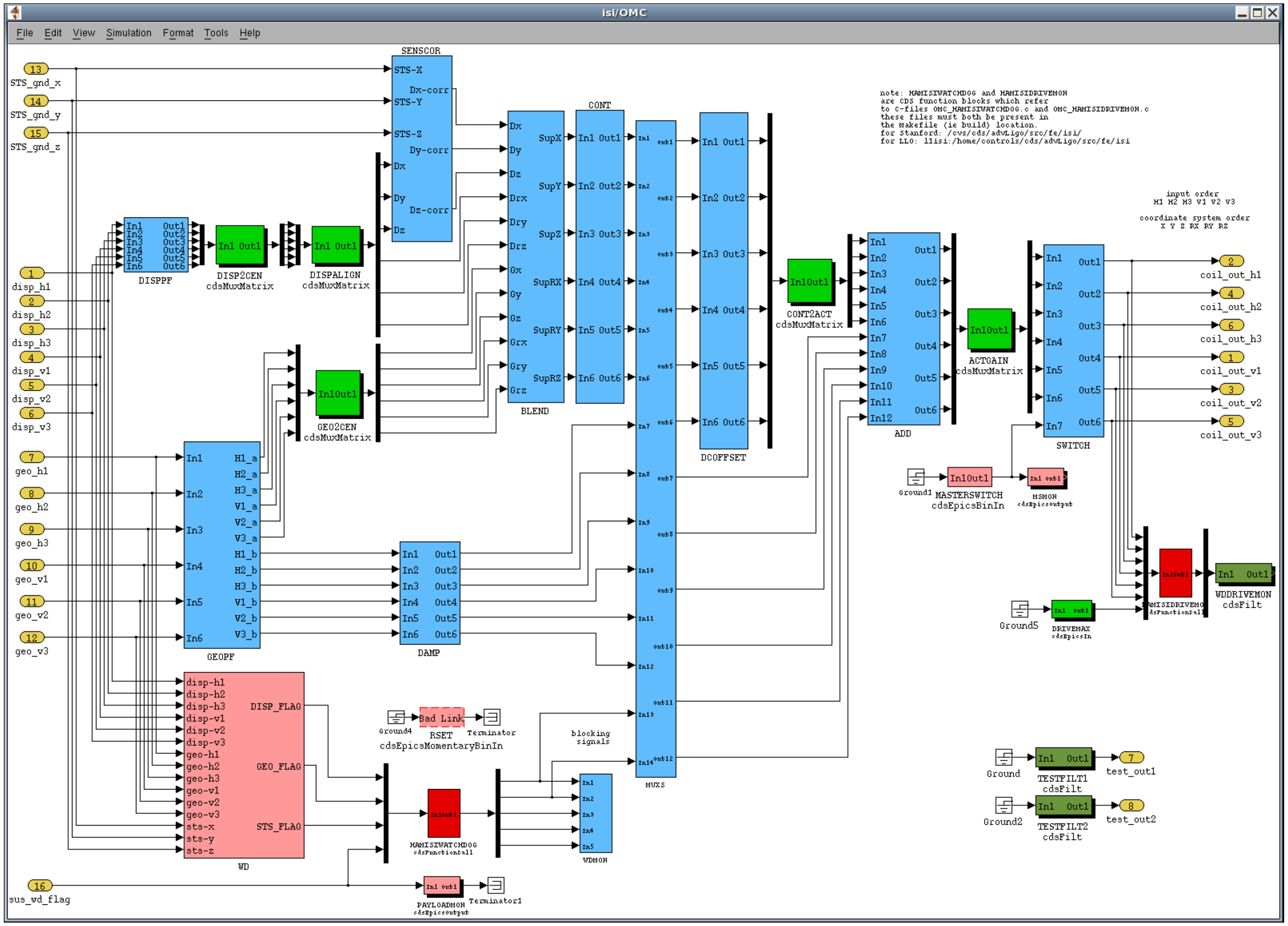


Figure 2: Overall ISI system diagram in Simulink.



```
note: HAMISIWATCHDOG and HAMISIDRIVEMON
are CDS function blocks which refer
to C-files OMC_HAMISIWATCHDOG.c and OMC_HAMISIDRIVEMON.c
these files must both be present in
the Makefile (ie build) location.
for Stanford: /cvs/cds/advLigo/src/fe/isi/
for LLO: l1isi:/home/controls/cds/advLigo/src/fe/isi
```

Figure 3: ISI-OMC subsystem diagram in Simulink.

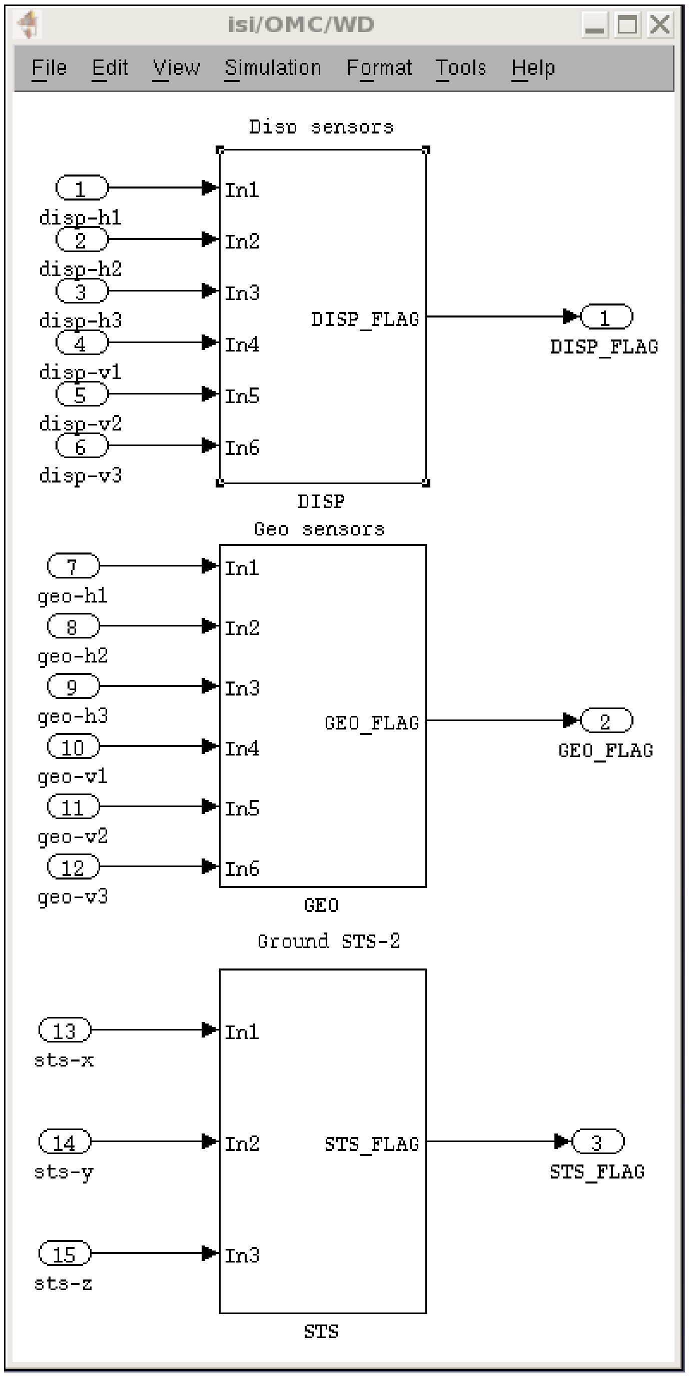


Figure 4: ISI-OMC-WD subsystem diagram in Simulink.

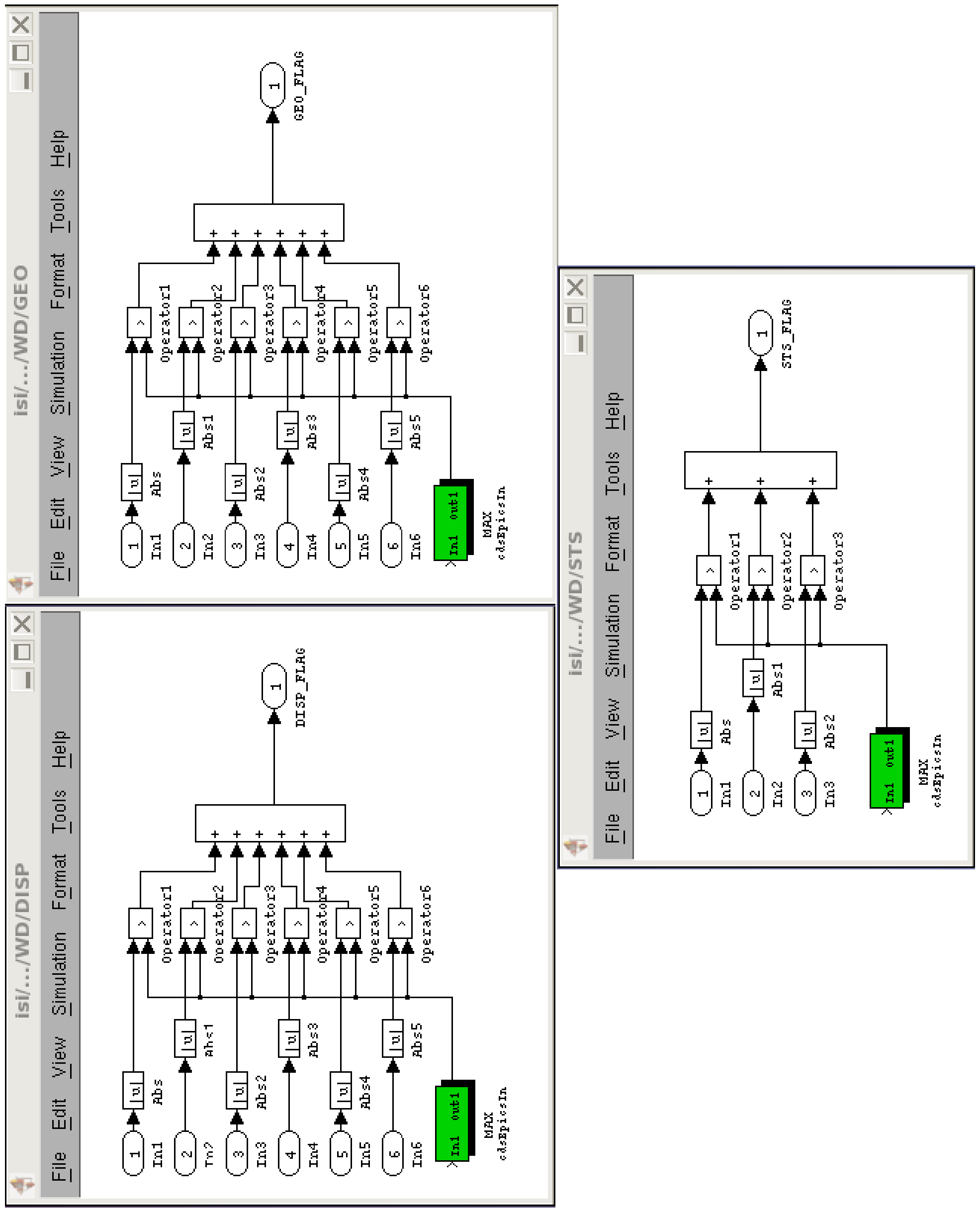

isi/.../WD/GEO
File Edit View Simulation Format Tools Help
In1 In2 In3 In4 In5 In6
Abs Abs1 Abs2 Abs3 Abs4 Abs5
Operator1 Operator2 Operator3 Operator4 Operator5 Operator6
GEO_FLAG
MAX cdsEpicsIn
isi/.../WD/DISP
DISP_FLAG
isi/.../WD/STS
STS_FLAG

Figure 5: ISI-OMC-WD-DISP and GEO subsystem diagram in Simulink.

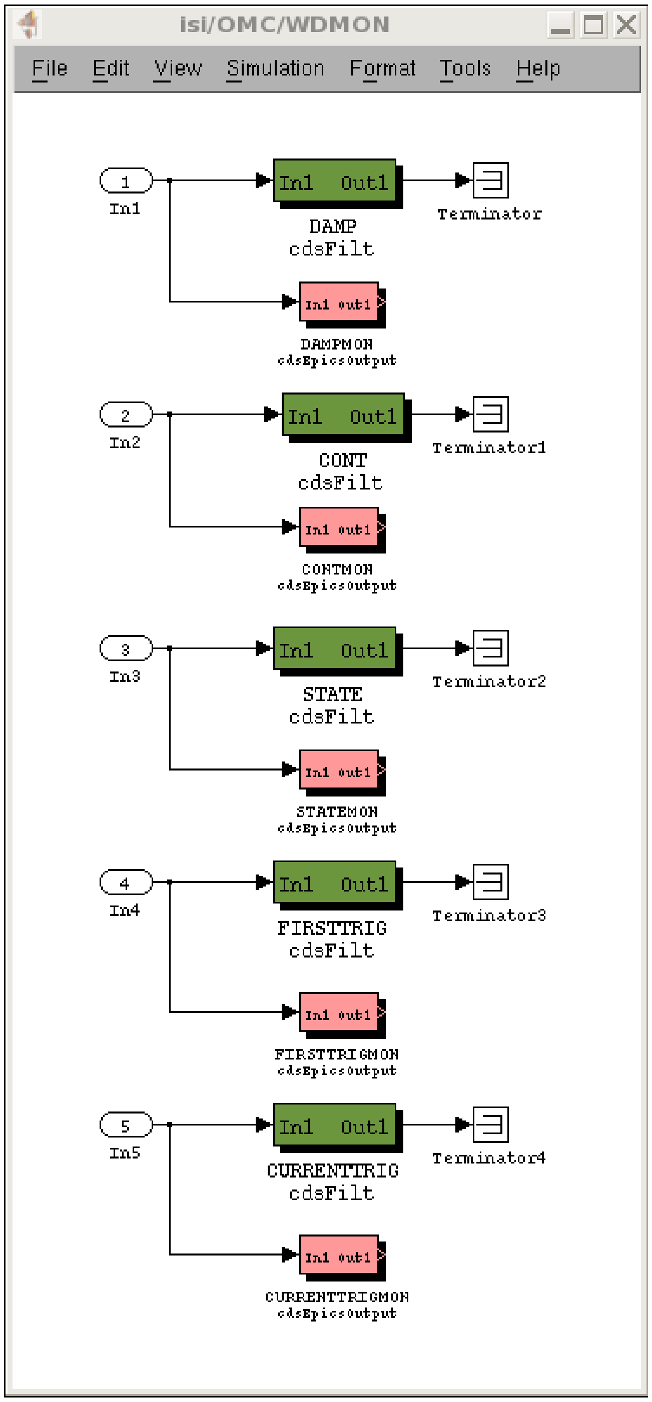


Figure 6: ISI-OMC-WDMON subsystem diagram in Simulink.

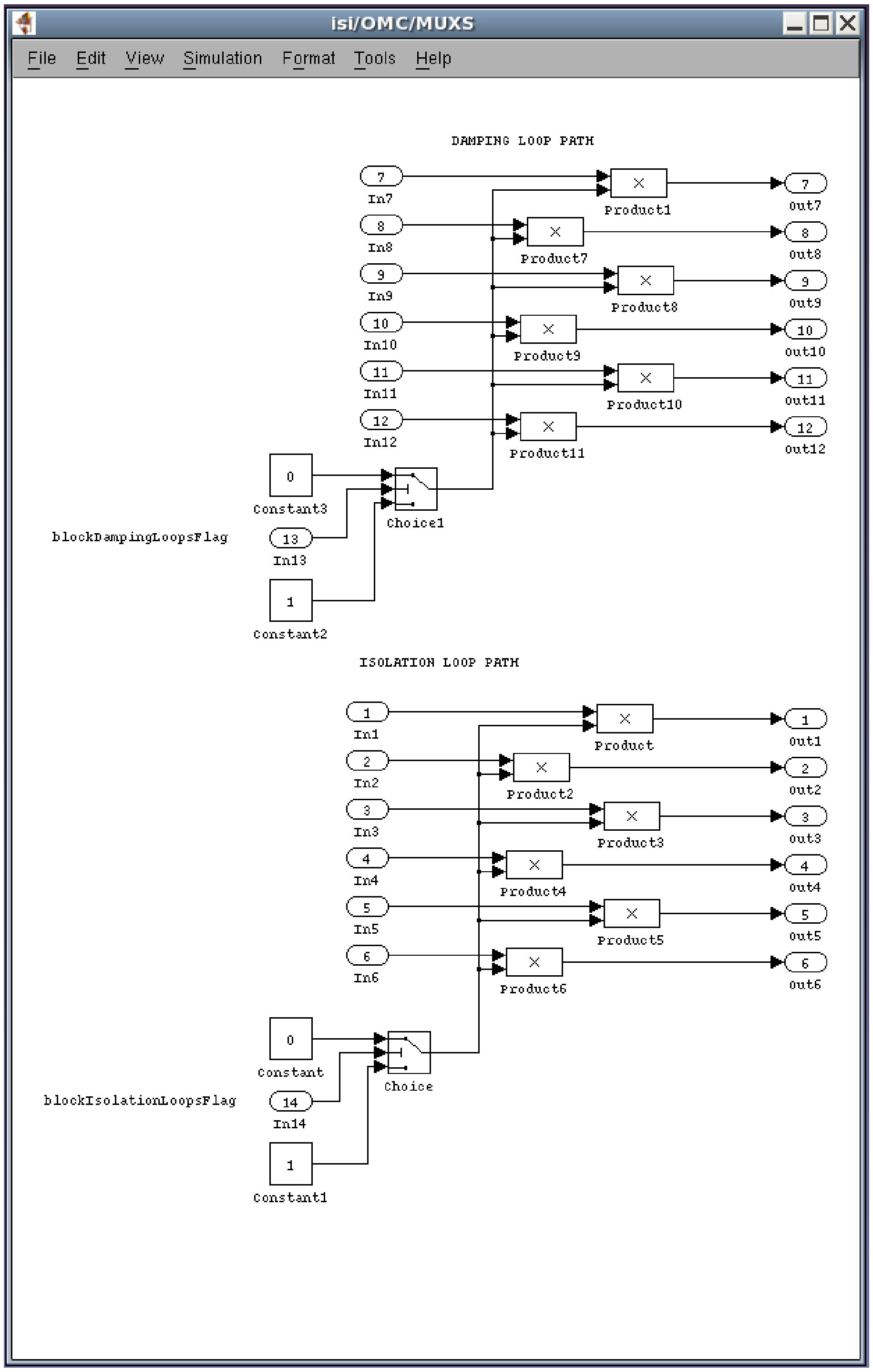
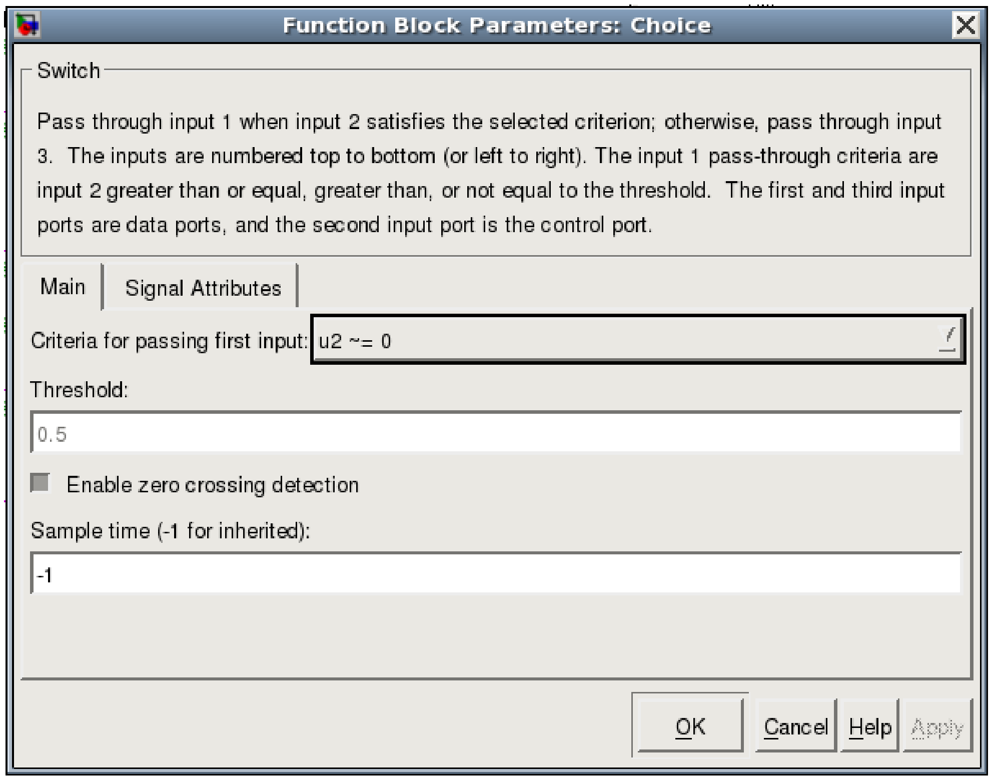


Figure 7: ISI-OMC-MUXS subsystem diagram in Simulink.


Function Block Parameters: Choice

Switch

Pass through input 1 when input 2 satisfies the selected criterion; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right). The input 1 pass-through criteria are input 2 greater than or equal, greater than, or not equal to the threshold. The first and third input ports are data ports, and the second input port is the control port.

Main | Signal Attributes

Criteria for passing first input: u2 ~= 0

Threshold:

0.5

Enable zero crossing detection

Sample time (-1 for inherited):

-1

OK | Cancel | Help | Apply

Figure 8: ISI-OMC-MUXS-CHOICE block menu under MUXS subsystem in Simulink.

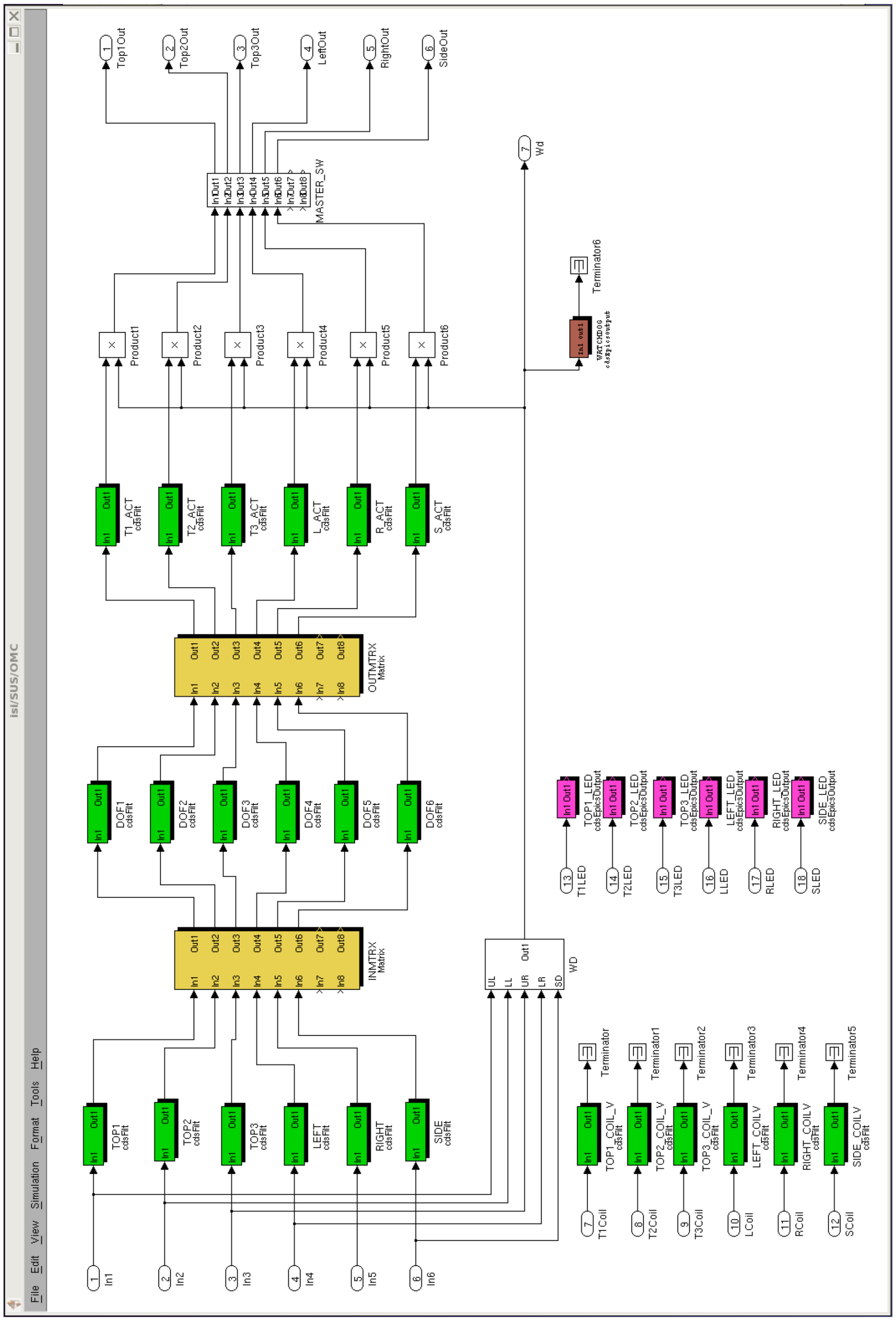


Figure 9: ISI-SUS-OMC subsystem diagram in Simulink.

3.2 C Code

This section describes the front-end code (written in C) that is responsible for the HAMISI-WATCHDOG and HAMISIDRIVEMON blocks in the simulink diagram. This code gets folded into the overall isi.c front end code during the "make" process.

As described in section 2, the watchdog is a finite state machine. OMC_HAMISIWATCHDOG.c is the C code on the front end which runs the state machine. The basic structure starts with the persistent variable "`state`," and the flags that determine whether the isolation and damping loop paths are blocked, "`blockIsolationLoopsFlag`," and "`blockDampingLoopsFlag`," respectively. `state` is initialized in state 4, and both `block*LoopsFlag`s are initialized as blocked (i.e. = 1). This way, anytime the code is started or restarted, the watchdog begins in state 4 with both paths are blocked until the user resets the watchdog.

After the initialization of a few more variables, the code sets "`triggers`" by creating a five digit bit-field based on the five input flags for displacement sensor, geophone, STS-2, actuator, and payload. `triggers` is a bit-field instead of simply a flag in case more than one input flag has tripped. Also, a bit-field is easily stored and read by MEDM for the user interfaces described in section 3.4.

Once `triggers` has been set, the watchdog code determines any state transitions needed by using a switch on `state`. The transitions include changing `state`, starting and stopping the arbitrary counter for use in state 2, storing the triggers that first cause the watchdog to trip, etc. Finally, the actions are performed in the last switch over `state`. Here, `blockIsolationLoopFlag` and `blockDampingLoopFlag` are set according to the value of `triggers` and `state`.

```
/*
 * OMC_HAMISIWATCHDOG.c
 *
 * HAMISIWATCHDOG is a state machine that watches the displacement
 * sensor, geophone, actuator, and payload watchdog flags, and blocks
 * the isolation loop path and/or damping loop path according to the
 * flags.
 *
 *
 * INPUTS:
 * argin[0] = displacement sensor saturation flag
 * argin[1] = geophone sensor saturation flag
 * argin[2] = Ground STS-2 sensor saturation flag
 * argin[3] = payload saturation flag
 * global variables
 *    actLevel    = actuator saturation flag (from OMC_HAMISIDRIVEMON.c)
 *    pLocalEpics = variable in control of CDS_EPICS (from EPICS database)
 *
 * OUTPUTS:
 * argout[0] = flag to block damping loop path
 * argout[1] = flag to block isolation loop path
 * argout[2] = state of watchdog
 * argout[3] = 4 character bit-field state after the first trigger
 *             (1 = displacement sensors; 2 = geophones;
 *              4 = actuators; 8 = payload;)
 * argout[4] = triggers;
 *
 * Written by Jeff Kissel and Tobin Fricke
 * Nov 26 2008
 * $Id: OMC_HAMISIWATCHDOG.c 318 2009-01-14 22:23:06Z seismic $
 */

void OMC_HAMISIWATCHDOG(double *argin, int nargin, double *argout, int nargout) {

  static int state           = 4;             // Start in STATE 4 (FULL SYSTEM SHUTDOWN)
  static int firstTrigger    = 0;             // Start with no indication of triggers
  static int cycleClock      = 0;             // Used to count cycles when in state 2.
  const int state2HoldCycles = 3 * 2048;      // Wait for [seconds] X (cycles/second)

  int blockIsolationLoopsFlag = 1;            // Start script with isolation path BLOCKED
  int blockDampingLoopsFlag   = 1;            // Start script with damping path BLOCKED

  // Read inputs
  int dispTriggered    = argin[0];            // Displacement Sensor Flag
  int geoTriggered     = argin[1];            // Geophone flag
  int stsTriggered     = argin[2];            // Ground STS-2 flag
  int payloadTriggered = argin[3];            // Payload flag
  int actTriggered     = actFlag;             // Actuator flag, defined in OMC_HAMISIDRIVEMON

  // Check and record the RESET button value
  int resetFlag =  pLocalEpics->isi.OMC_RSET; // Epics variable for reset button
  pLocalEpics->isi.OMC_RSET = 0;              // Set reset button epics variable to 0

  // Check for triggers - make a 5 character bitfield so we can tell which triggers triggered
  // The !! (double inverse) syntax guarantees that the given value is 0 or 1.
  // The << is a bitwise shift left, i.e. 1 << 2 == 4 and 1 << 3 == 8.
  // So, if the displacement sensors and actuators trigger, triggers = 01001;
  int triggers =
    (!!dispTriggered) | (!!geoTriggered << 1) | (!!stsTriggered << 2) | (!!actTriggered << 3) | (!!pa

  // State transitions
  switch (state) {
  case 1: // To STATE 1 (ARMED)
    if (triggers) {
      firstTrigger = triggers; // Record the bit-field state after the first trigger
      cycleClock = 0;
      state = 2;
    }
    break;
```

```
case 2: // To STATE 2 (TRIGGERED; HOLDING 3 SECONDS WITH DAMPING ENABLED)
  if (cycleClock >= state2HoldCycles) {
    cycleClock = 0;
    state = 3;
      // If three seconds haven't passed yet, we just stay here in state 2.
  }
  break;

case 3: // To STATE 3 (TRIGGERED; MONITORING WITH DAMPING ENABLED)
  if (triggers)
    state = 4;
  else if (resetFlag) {
    firstTrigger = 0; // Reset first trigger
    state = 1;
  }
  break;

default:
case 4: // To STATE 4 (TRIGGERED; FULL SHUTDOWN)
  if (!triggers && resetFlag) {
    firstTrigger = 0; // Reset first trigger
    state = 1;
  }
  break;
}

// State actions
switch (state) {
case 1: // STATE 1 (ARMED)
  blockIsolationLoopsFlag = 0;     // Leave isolation loop path open
  blockDampingLoopsFlag = 0;       // Leave damping loop path open
  break;

case 2: // STATE 2 (TRIGGERED; HOLDING 3 SECONDS WITH DAMPING ENABLED)
  blockIsolationLoopsFlag = 1;    // Block isolation loops path
  blockDampingLoopsFlag = 0;      // Leave damping loops path open
  cycleClock = cycleClock + 1;    // Keep counting, it hasn't been long enough yet
  break;

case 3: // STATE 3 (TRIGGERED; MONITORING WITH DAMPING ENABLED)
  blockIsolationLoopsFlag = 1;    // Block isolation loops path
  blockDampingLoopsFlag = 0;      // Leave damping loops path open
  break;

default:
  case 4: // STATE 4 (TRIGGERED; FULL SHUTDOWN)
    blockIsolationLoopsFlag = 1;     // Block isolation loops path
    blockDampingLoopsFlag = 1;       // Block damping loop path
    break;

  }

  // Output
  argout[0] = blockDampingLoopsFlag;
  argout[1] = blockIsolationLoopsFlag;
  argout[2] = state;
  argout[3] = firstTrigger;
  argout[4] = triggers;

  return;
}
```

The only other piece of C code for the watchdog is OMC_HAMISIDRIVEMON.c, which represents the block where actuator drive levels are monitored. Similar to the “WD” block described in section 3.1, the code monitors the digital signal sent out to the DAC, and sets a flag (the global variable “actFlag”) accordingly based on the user-defined actuator drive threshold.

```
/*
 *  OMC_HAMISIDRIVEMON.c
 *
 *  INPUTS
 *  mux3[0] = digital output to H1 actuator
 *  mux3[1] = digital output to H2 actuator
 *  mux3[2] = digital output to H3 actuator
 *  mux3[3] = digital output to V1 actuator
 *  mux3[4] = digital output to V2 actuator
 *  mux3[5] = digital output to V3 actuator
 *  mux3[6] = user defined threshold (DRIVEMAX)
 *
 *  OUTPUTS:
 *  demux4[0] = saturation flag
 *
 *  actFlag (nominally 0) is set to 1 if any digital
 *  signal exceeds user defined threshold. actFlag is used
 *  for a flag read by OMC_HAMISIWATCHDOG.c.
 *
 *  Written by Jeff Kissel
 *  Nov 17 2008
 *  $Id: OMC_HAMISIDRIVEMON.c 265 2008-12-01 18:16:05Z seismic $
 */

static int actFlag = 0; // must be static to persist between cycles,
                        //in case WATCHDOG is called before DRIVEMON

void OMC_HAMISIDRIVEMON(double *argin, int nargin, double *argout, int nargout) {

  int numActuators = 6;       // all but the last input
  int userThreshold = argin[6];  // the last input

  actFlag = 0; // no saturations yet

  // look for saturations
  int ii;
  for (ii = 0; ii < numActuators; ii++)
    // Set actuator flag to either 0 or 1, using the |= (or-equals) bitwise
    // OR assignment operator. The statement below is equivalent to
    //          actFlag = actFlag | (argin[ii] >= userThreshold);
    // such that the flag is sit if ANY actuators have tripped.
    actFlag |= (argin[ii] >= userThreshold);

  argout[0] = actFlag;       //Send flag value to output
}
```

3.3 Scripts

There are several auxiliary scripts that must be running at all times. They include "*ctrlDOWN*," "*checker*," ""*chk_daemon*." These scripts serve to prevent further digital requests to drive the HAM-ISI after the watchdog has been tripped, more specifically isolation loop signals.

The digital portion of the isolation loops run independently after a watchdog trip; the watchdog simply denies any of that control signal to reach the actuators. Because the isolation loops are only conditionally stable, when the input control signal sudden vanishes after a watchdog trip, the loops saturate (if they have not saturated already). The auxiliary scripts serve to turn off the independent isolation loop control system. If it were not in place, the user could mistakenly reset the watchdog while the isolation loops are still requesting huge actuation signals, and slam the HAM ISI around. They control higher level epics variables, and do not don't need to cycle as quickly as the front end code, so they are run on a separate Linux machine.

The functionality is as follows: *checker* is a bash script that must be run continually. It checks every second whether the channel L1:ISI-OMC_WDMON_FIRSTTRIGMON is non-zero (as described in section 3.1, FIRSTTRIGMON is an integer representing the bit-field `triggers` of the first saturation that cause the watchdog to trip). If FIRSTTRIGMON is non-zero, *checker* calls *ctrlDOWN*. When the bash script *ctrlDOWN* is called, it immediately does the following to every isolation loop degree of freedom:

1. Sets the gain ramp time to ZERO.
2. Turns the output OFF.
3. Turns the input OFF.
4. Turns the isolation loop boost filter OFF.

5. Turns the isolation loop gain to ZERO.
6. Clears the filter bank history.
7. Sets to the gain ramp time to five seconds.

which totally turns off the digital isolation loop system, as desired.

The final script, *chk_daemon* are merely in place to make sure *checker* is running. This check is done once a minute, placed under the Linux machine's crontab.

The following are copies of the scripts currently running at the Livingston observatory.

checker:

```
#!/bin/bash

# CHECKER checks the status of ISI watchdog every second. If tripped, calls
# the "ctrlDOWN" script. This function will read status of output lines of
# HAMISIWATCHDOG cdsFunction in the isi.mdl simLink diagram. If the output
# of the function is less than one, this function will call the "ctrlDOWN" script
# that goes through proper sequences to safely bring down the LLO HAM6 ISI
# isolation loops.

# $Id: checker 269 2008-12-01 23:18:35Z seismic $

SITE='L1';
SCRIPTSPATH='/cvs/cds/llo/scripts/l1/ISI/';
SYSTEM=':ISI-OMC_';
LOG=${SCRIPTSPATH}checker_status.log;

while true
    do
    sleep 1  # run this checker every one second
    echo $0 >> ${LOG}
    echo "Checking watchdog status ..."
    echo "Checking watchdog status ..." >> ${LOG}
    STATUS=`caget -t ${SITE}${SYSTEM}WDMON_FIRSTTRIGMON`;

    # check if ${STATUS} is empty string
    while [ -z ${STATUS} ]; #
        do
        echo "Check of status failed!"
echo "Rechecking status!"
        echo "Check of status failed!" >> ${LOG}
echo "Rechecking status!" >> ${LOG}
        STATUS=`caget -t ${SITE}${SYSTEM}WDMON_FIRSTTRIGMON`;
    done
    # Report status of check to log
    DATE=`date`
    echo "${DATE} STATUS = ${STATUS} PID = $PPID HOST = ${HOST} SITE = ${SITE}"
    echo "${DATE} STATUS = ${STATUS} PID = $PPID HOST = ${HOST} SITE = ${SITE}" >> ${LOG}

    # If status is non-zero
    if [ "${STATUS}" -ne "0" ]; then
echo "Watchdog tripped! Calling ctrlDOWN ..."
        echo "Watchdog tripped! Calling ctrlDOWN ..." >> ${LOG}
        cd ${SCRIPTSPATH}
        ./ctrlDOWN >> ${LOG}
echo "ctrlDOWN done."
echo "ctrlDOWN done." >> ${LOG}
    else
echo "Status OK ..."
        echo "Status OK ..." >> ${LOG}
    fi

done

exit 0
```

ctrlDOWN

```
#!/bin/bash

# ctrlDOWN is designed as an automatic, quick methods to turn off
# the isolation loops in case of saturations.
#
#

SITE='L1';
SYSTEM=':ISI-OMC_';
boostButton='FM10';
scriptsPath='/cvs/cds/llo/scripts/l1/ISI/';

for iCoordDOFs in X Y Z RX RY RZ
    do
    caput ${SITE}${SYSTEM}CONT_${iCoordDOFs}_TRAMP 0.0                # Set gain ramp to zero seconds
    ezcaswitch ${SITE}${SYSTEM}CONT_${iCoordDOFs} OUTPUT OFF          # Turn off output
    ezcaswitch ${SITE}${SYSTEM}CONT_${iCoordDOFs} INPUT OFF           # Turn off input
    ezcaswitch ${SITE}${SYSTEM}CONT_${iCoordDOFs} ${boostButton} OFF  # Turn off boost filter
    caput ${SITE}${SYSTEM}CONT_${iCoordDOFs}_GAIN 0.0                 # Set gain to zero
    caput ${SITE}${SYSTEM}CONT_${iCoordDOFs}_RSET 2                   # Clear history on filter bank
    caput ${SITE}${SYSTEM}CONT_${iCoordDOFs}_TRAMP 5.0                # Set gain ramp back to five se
done
```

chk_daemon

```
#!/bin/bash

# CHK_DAEMON
# Purpose:  This daemon script is to be linked to cron daemon running every
#           minute. This script checks if checker script is running, and if
#           not, then runs the script.
#
# $Id: chk_daemon 299 2009-01-13 01:05:17Z seismic $

scriptsPath='/cvs/cds/llo/scripts/l1/ISI/';

# Report the current processes (ps), all of them (-e),
# and search for the checker process (| grep checker),
# but remove the search from that list (| grep -v grep).
ps -ef | grep -v grep | grep checker

# The variable $? is the return
# value of the last executed program.
# It is zero if a match was found, and
# 1 if no match was found.

if [ "$?" -eq 1 ]
      then
      echo "Watchdog function checker not running - restarting ..."
      ${scriptsPath}checker &
      echo "Sleeping for 60 seconds..."
else
      echo "Watchdog function checker running - waiting until next check ..."
      echo "Sleeping for 60 seconds..."
fi

return
```

3.4 Medm Screens

Considerable effort has been made to make the MEDM control screens look and feel very much like the simulink model. Figure 10 shows the overview screen for control of the HAM ISI. A detailed explanation of the the overview screen is not in the scope of this document, but the flow of data is from left to right (or bottom to top since the figure is rotated). Just after the blue "isolation filters" and tan "damp" block, there is a tan dotted line connected to a box titled "watchdog." This box is actually a link to a sub-screen shown in figure 12. The green inner box inside the watchdog box on the overview screen indicates that watchdog is armed and in state 1. In any other state, this inner box shows red.

The overview's progression through states shown in figure 11. Notice, that in state 1 (TOP

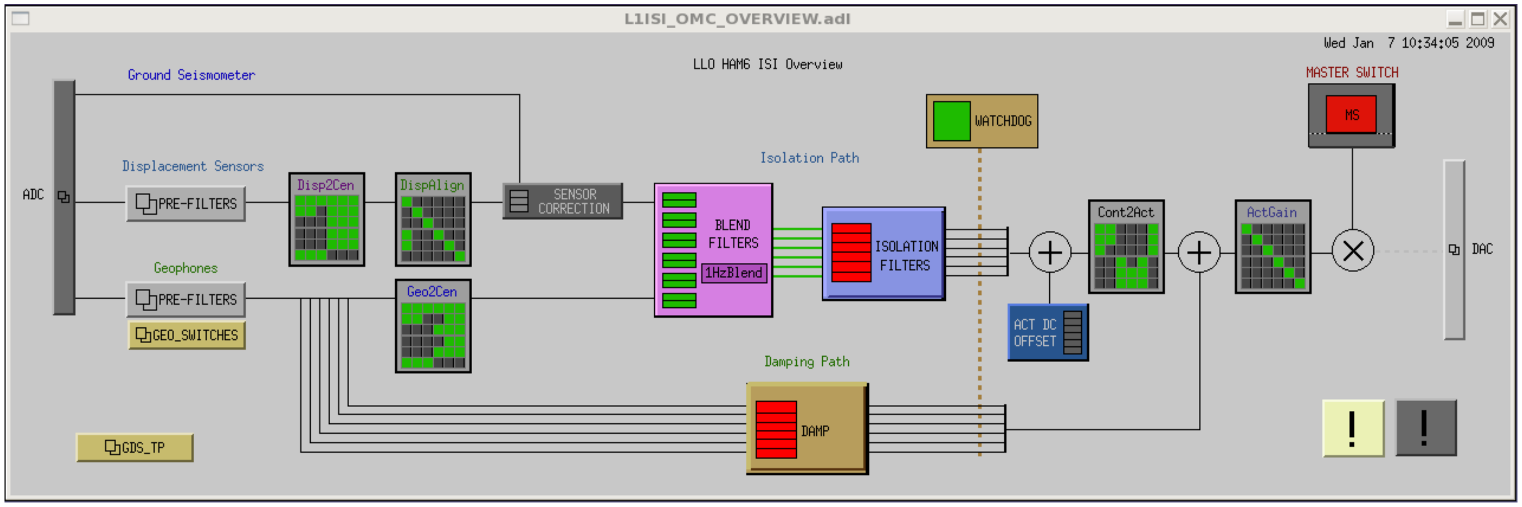


Figure 10: HAM ISI Watchdog MEDM screen

panel), the inner box in the watchdog block shows green, indicating the watchdog is in state 1, or armed. In states 2 and 3 (MIDDEL panel), the inner box is red, and only the isolation path has a red blocker in on top of the tan dash line. In state 3 (BOTTOM), the inner box is again red, but now both the isolation and the damping paths are blocked.

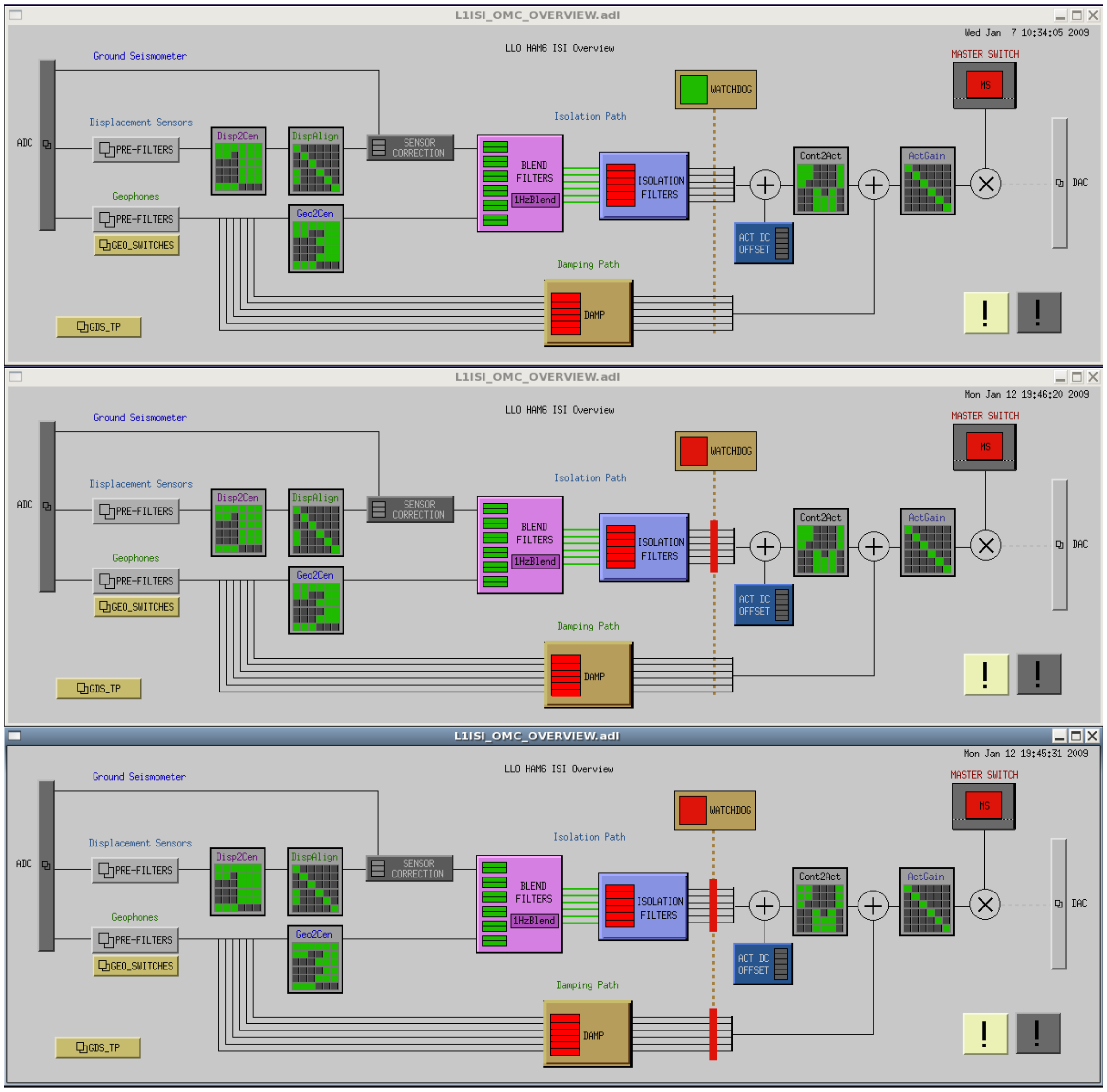


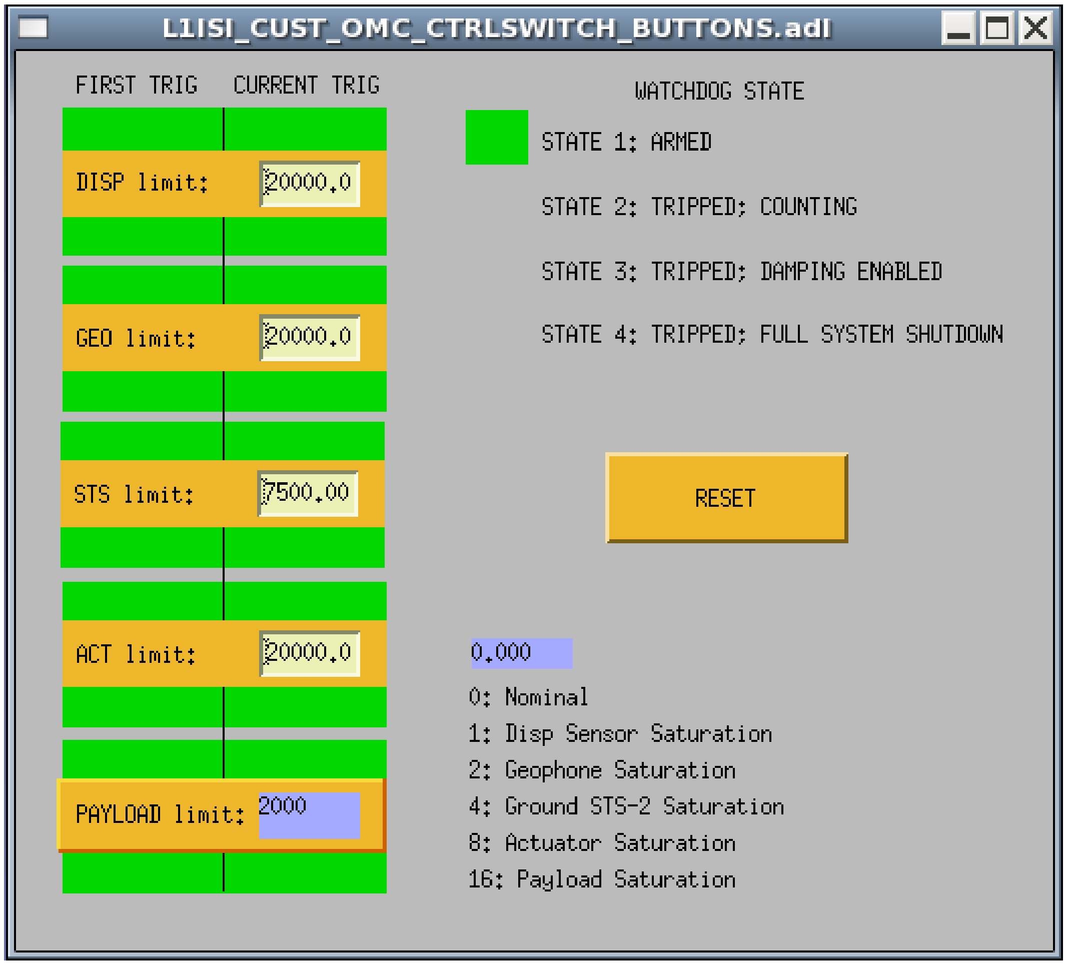
Figure 11: HAM ISI Watchdog MEDM screen. The top, middle, and bottom panels show the overview screen in state 1, 2&3, and 4, respectively.

Figure 12 shows the watchdog MEDM screen. In tan on the left are where the user can define the thresholds for displacement sensors (DISP limit:), geophones (GEO limit:), and actuators

(ACT limit). Note that the fourth threshold, the PAYLOAD limit, is a read-only display of the payload threshold which is set by another, intentionally independent, MEDM screen. Surrounding these limits are bars indicating the status of the triggers. These bars are green if their respective flags are not triggered, and red if they are. There are two columns of bars, the left column shows the flags that saturated causing the initial trip of the watchdog, and the right coumn shows what is currently being flagged for saturation.

In the top right corner right, under "watchdog state," the current state of the watchdog is shown. When the watchdog is in state 1, a green box appears near "state 1: armed." For state 2, a yellow box appears near "state 2: tripped, counting." For state 3, a bright red box appears near "state 3: tripped, damping enabled." Finally, when in state 4, a dark red box appears near "tripped; full system shutdown." This progression is show in figure 13.

In the bottom left corner are a field which reports the value of CURRENT TRIG (corresponding to `triggers` in the front end code) with explanation of the bit field below and the reset button, which will reset the watchdog from state 3 or 4 into 1, as long as there are no current trigger flags.

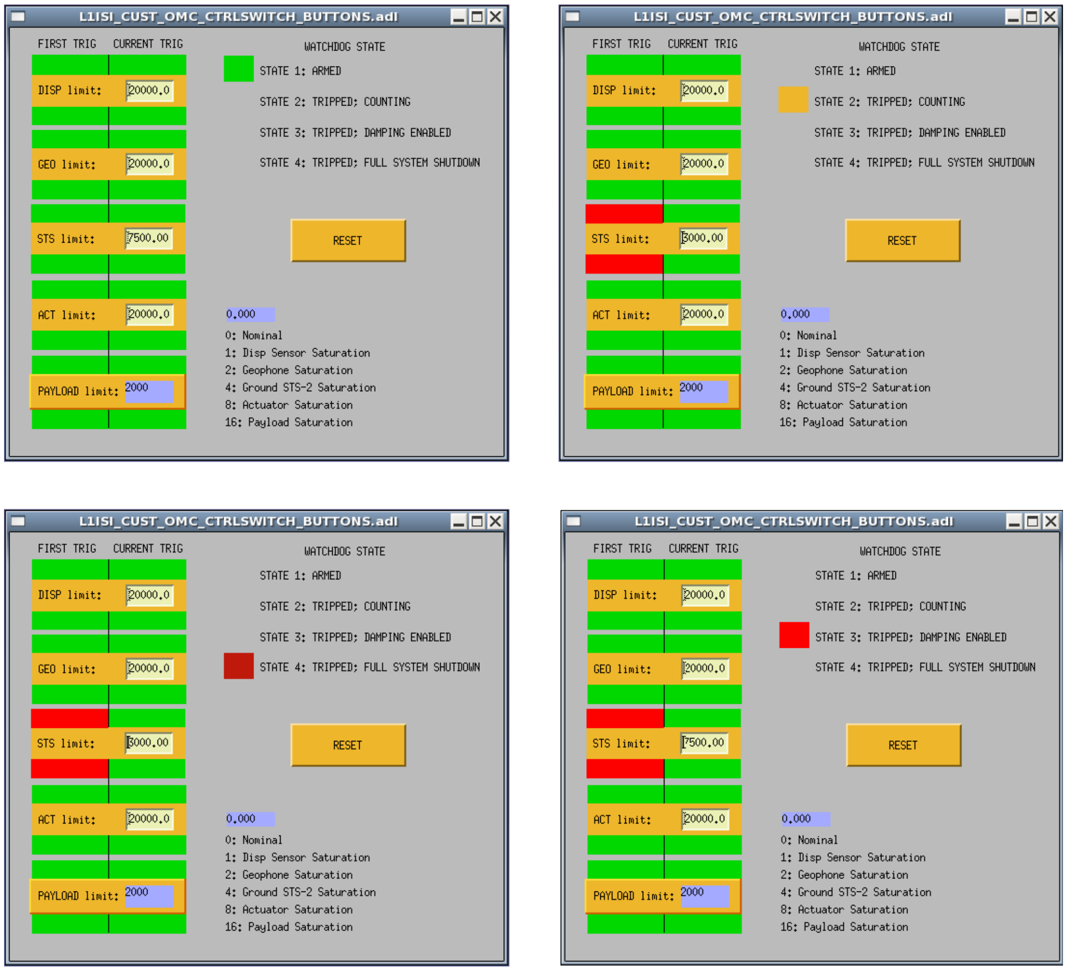


DISP limit: 20000.0
GEO limit: 20000.0
STS limit: 7500.00
ACT limit: 20000.0
PAYLOAD limit: 2000

STATE 1: ARMED
STATE 2: TRIPPED: COUNTING
STATE 3: TRIPPED: DAMPING ENABLED
STATE 4: TRIPPED: FULL SYSTEM SHUTDOWN

0.000
0: Nominal
1: Disp Sensor Saturation
2: Geophone Saturation
4: Ground STS-2 Saturation
8: Actuator Saturation
16: Payload Saturation

Figure 12: HAM ISI Watchdog MEDM screen.



L1ISI_CUST_OMC_CTRLSWITCH_BUTTONS.adl

FIRST TRIG CURRENT TRIG
DISP limit: 20000.0
GEO limit: 20000.0
STS limit: 7500.00
ACT limit: 20000.0
PAYLOAD limit: 2000

WATCHDOG STATE
STATE 1: ARMED
STATE 2: TRIPPED; COUNTING
STATE 3: TRIPPED; DAMPING ENABLED
STATE 4: TRIPPED; FULL SYSTEM SHUTDOWN

RESET

0.000
0: Nominal
1: Disp Sensor Saturation
2: Geophone Saturation
4: Ground STS-2 Saturation
8: Actuator Saturation
16: Payload Saturation

L1ISI_CUST_OMC_CTRLSWITCH_BUTTONS.adl

FIRST TRIG CURRENT TRIG
DISP limit: 20000.0
GEO limit: 20000.0
STS limit: 3000.00
ACT limit: 20000.0
PAYLOAD limit: 2000

WATCHDOG STATE
STATE 1: ARMED
STATE 2: TRIPPED; COUNTING
STATE 3: TRIPPED; DAMPING ENABLED
STATE 4: TRIPPED; FULL SYSTEM SHUTDOWN

RESET

0.000
0: Nominal
1: Disp Sensor Saturation
2: Geophone Saturation
4: Ground STS-2 Saturation
8: Actuator Saturation
16: Payload Saturation

L1ISI_CUST_OMC_CTRLSWITCH_BUTTONS.adl

FIRST TRIG CURRENT TRIG
DISP limit: 20000.0
GEO limit: 20000.0
STS limit: 3000.00
ACT limit: 20000.0
PAYLOAD limit: 2000

WATCHDOG STATE
STATE 1: ARMED
STATE 2: TRIPPED; COUNTING
STATE 3: TRIPPED; DAMPING ENABLED
STATE 4: TRIPPED; FULL SYSTEM SHUTDOWN

RESET

0.000
0: Nominal
1: Disp Sensor Saturation
2: Geophone Saturation
4: Ground STS-2 Saturation
8: Actuator Saturation
16: Payload Saturation

L1ISI_CUST_OMC_CTRLSWITCH_BUTTONS.adl

FIRST TRIG CURRENT TRIG
DISP limit: 20000.0
GEO limit: 20000.0
STS limit: 7500.00
ACT limit: 20000.0
PAYLOAD limit: 2000

WATCHDOG STATE
STATE 1: ARMED
STATE 2: TRIPPED; COUNTING
STATE 3: TRIPPED; DAMPING ENABLED
STATE 4: TRIPPED; FULL SYSTEM SHUTDOWN

RESET

0.000
0: Nominal
1: Disp Sensor Saturation
2: Geophone Saturation
4: Ground STS-2 Saturation
8: Actuator Saturation
16: Payload Saturation

Figure 13: HAM ISI Watchdog MEDM screen. The four panels, starting in the bottom left and continuing clockwise (as in figure 1) show the screen in state 1, 2, 3, and 4.