LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY
- LIGO -
CALIFORNIA INSTITUTE OF TECHNOLOGY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Technical Note **LIGO-T1400349–v11** 2016/09/16

The Advanced LIGO Hardware Injection Infrastructure

A. Mullavey, E. Thrane, C. Biwer, K. Riles, P. Shawhan, J. Kissel, J. Betzwieser, E. Goetz

Distribution of this document:

Hardware Injection Group

California Institute of Technology
LIGO Project, MS 18-34
Pasadena, CA 91125
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

Massachusetts Institute of Technology
LIGO Project, Room NW17-161
Cambridge, MA 02139
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

LIGO Hanford Observatory
Route 10, Mile Marker 2
Richland, WA 99352
Phone (509) 372-8106
Fax (509) 372-8137
E-mail: info@ligo.caltech.edu

LIGO Livingston Observatory
19100 LIGO Lane
Livingston, LA 70754
Phone (225) 686-3100
Fax (225) 686-7189
E-mail: info@ligo.caltech.edu

http://www.ligo.caltech.edu/

Contents

1 Introduction and Overview

Hardware injections are simulated gravitational-wave signals added to LIGO and Virgo's strain channel by physically actuating on the test masses. By testing to see that we can observe injected signals, hardware injections provide an end-to-end validation of our ability to detect gravitational waves: from the detector through to the interpretation of results from data analysis pipelines.

1.1 Mission and scope

The hardware injection group is tasked with the development, testing, and maintenance of hardware injection infrastructure. This includes on-site software, which carries out the injections at specified times. We also work with the data analysis groups to maintain software used to generate gravitational waveforms suitable for injection.

1.2 Search Groups

Each data analysis group liaises with the hardware injection subgroup. The Burst and CBC groups work with the subgroup to provide transient waveforms and to determine suitable injection rates. The CW group selects the parameters for neutron star signals, which persist throughout the observation run. The stochastic group typically carries out one or two $\approx$10 min injections during each observation run. The data analysis groups analyze hardware injections during observation and engineering runs to identify and solve problems as they come up. The results of these studies are reported back to the hardware injection team so that adjustments can be made.

1.3 Injection Plans for Observation Runs

Each group's hardware injection plans for each observation run are kept on the DCC and linked to the hardware injection documentation tree. The CBC injection plans are linked to LIGO-E1600215, the Burst injection plans are linked to LIGO-E1600216, the CW plans are linked to LIGO-E1600217 and the Stochastic plans are linked to LIGO-E1600218.

1.4 This document

This document describes the infrastructure that makes up the hardware injection system. It serves as a first stop for those wanting to learn how the hardware injection system is set-up.

1.5 Other reading

This document has been developed in concert with calibration (see LIGO-E1500281) and noise budget efforts. Please see these references for additional details.

1.6 Overview

The hardware injection system is summarized by the flowchart in Fig. 1. It is a combination of astro-physically motivated waveforms and the on-site infrastructure used to inject them into the interferometer. The different blocks in this flowchart are covered in the following sections.

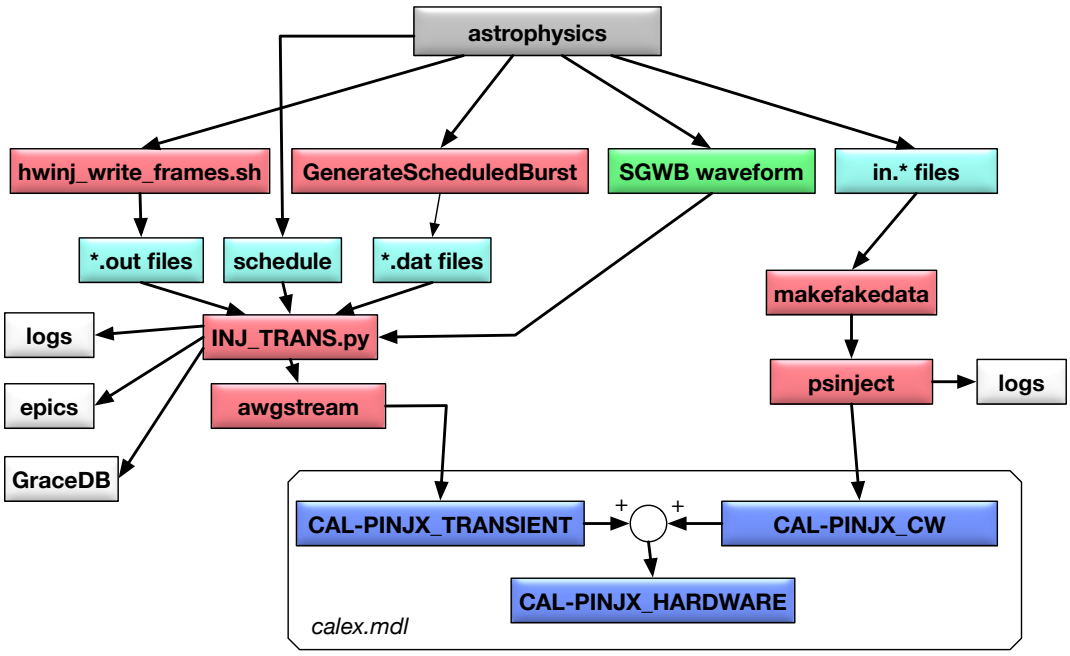

Figure 1: Flowchart of aLIGO hardware injection infrastructure.

2 Waveform generation

The decisions of which waveforms to inject and at what amplitude are made by the four data-analysis groups. Each group is responsible for their own waveform generation.

2.1 CBC waveform generation

The CBC group maintains waveform generation code called `hwinj_write_frames.sh`, which generates injection files consisting of $h(t)$. It takes as input an XML file, provided by the CBC group, which specifies the properties of each waveform (e.g., injection time, chirp mass, etc.).

Documentation for running the waveform generation code is kept in the PyCBC repository at https://github.com/ligo-cbc/pycbc/blob/master/docs/hwinj.rst.

One of the outputs of waveform generation are LIGOLW XML files with a `sim_inspiral` table. Documentation for uploading `sim_inspiral` rows to GraceDB can be found at https://svn.ligo.caltech.edu/svn/dac/hwinj/cbc/README.

2.2 Burst waveform generation

The Burst group maintains analogous waveform generation code called `GenerateScheduledBurst`.

2.3 CW waveform generation

In addition to the `.cfg` files described in LIGO-T1600421, each CW injection has a `in.$` file, which is a one-line shell script wrapper to call lalapps_makefkaedata with the appropriate parameters. Here is an example of an `in.#` file:

../../bin/lalapps_Makefakedata_v4 Pulsar0_StrainAmp.cfg –Tsft=20 –duration=51 84000 -b –ephemEarth "/home/eric.thrane/master/opt/lscsoft/lalpulsar/share/lalp ulsar/earth00-19-DE405.dat.gz" –ephemSun "/home/eric.thrane lalpulsar/share/lalpulsar/sun00-19-DE405.dat.gz" -l pulsar0.log

Some of the arguments are identical for each injection while others are different (such as the log file).

There are a few steps to add additional CW injections. The first step is to create new `.cfg` and in files for each injection. Then, edit the file `number_of_signals` (in the same directory as the .cfg and in files) in order to account for the added injections.

3 On-site Infrastructure

The on-site hardware injection infrastructure can be divided into three parts. The physical hardware that actuates the mirror, i.e. the photon calibrator, the real-time code that controls it, the software that is ultimately responsible for injecting the waveform.

3.1 The Photon Calibrator

The Advanced LIGO photon calibrator system uses a power-modulated auxiliary laser to modulate the position of a test mass via radiation pressure. The system provides a calibrated readback of the induced force on the end mirror. This force is translated into a calibrated length from the known force-to-length transfer function of the suspended mirror. The drive input can also be calibrated, although it should be realized that this is less robust so all analyses should verify the requested waveform is measured by the photon calibrator readback at CAL-PCAL_TX_PD_DQ (or RX).

At each site there are two photon calibrators, one for each end mirror. The photon calibrator's intended use is to inject calibration lines into DARM which are used for precise calibration of the detector. This function can be performed by one photon calibrator alone,

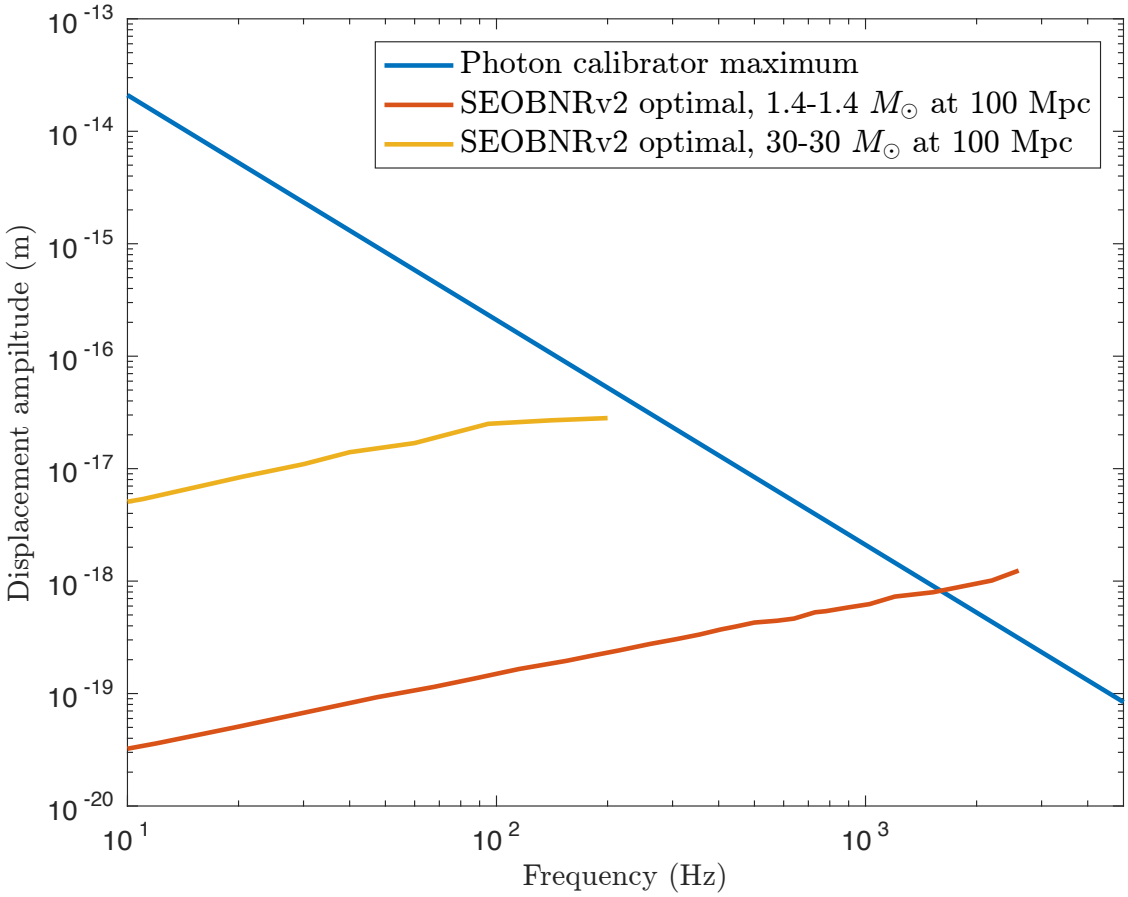

Figure 2: Injection capability of the aLIGO photon calibrators.

leaving the second calibrator as a spare. The hardware injection system takes advantage of this free photon calibrator and uses it to inject all waveforms. At both sites, the X end photon calinbrator is used for hardware injections. For extensive documentation of the Advanced LIGO photon calibrator system, see LIGO-E1300707.

Some of the most demanding injections are CBC waveforms. These cover a broad frequency span, ∼10 Hz to ∼2 kHz and with potentially large amplitudes. In figure 2 we illustrate two different waveforms, one a black hole-black hole merger, and one a neutron star-neutron star merger, both optimally oriented and at a distance of 100 Mpc. This required amplitude injection is compared against the maximum possible drive using a photon calibrator.

3.2 Real-time Code

The real-time or front-end code is compiled C-code that controls the Photon Calibrator and provides the interface between the user or injection software with the photon calibrator. In our case we can send excitations in via this code. The photon calibrator front-end code is called l1calex at LLO and h1calex at LHO, and the simulink diagram of this code can be seen in Fig. 3. At the bottom of this diagram, there is a block called PINJX. This block contains the front-end code from which the injection waveforms are passed through. The

'HARDWARE_INJ_OUT' output from this block is summed with the regular 'PCAL Exc' output on the right side of this diagram and sent out of the Digital-to-Analog-Converter (DAC) to the photon calibrator.

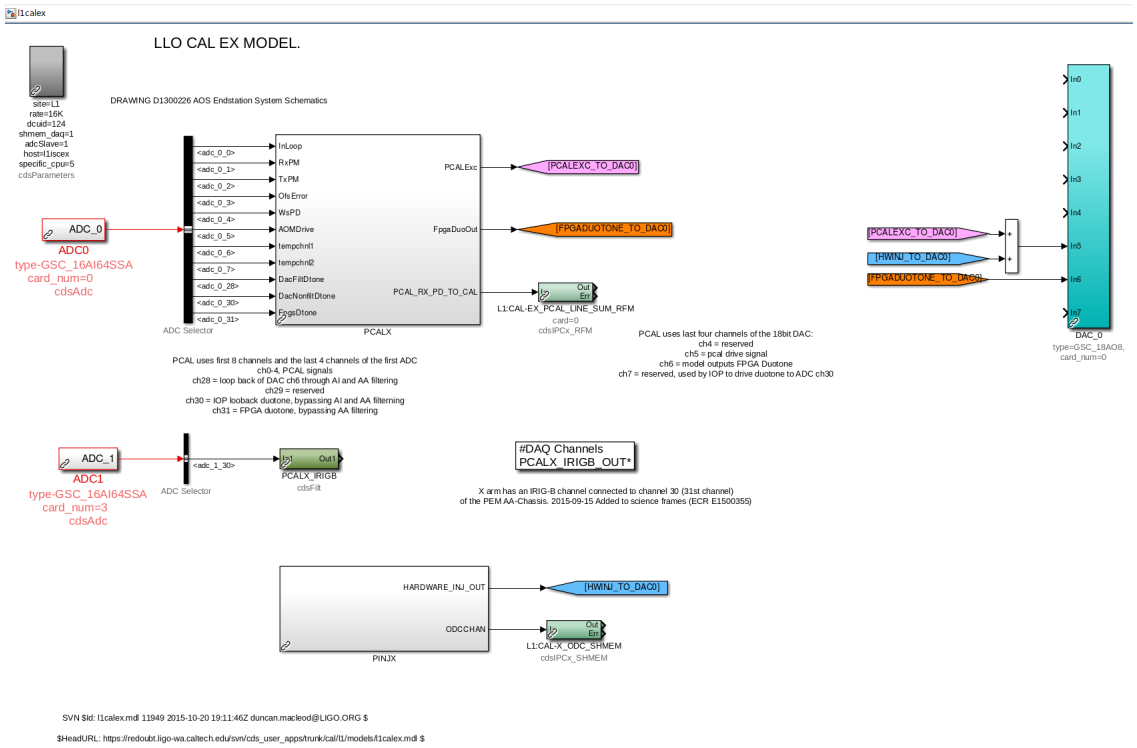

Figure 3: Simulink drawing of the l1calex front end code.

A simulink diagram of the code inside the PINJX block is shown in Fig. 4

At the top left corner of this diagram, there are three green blocks labelled CW, TRANSIENT and HARDWARE. These green blocks are filter banks and they provide excitation channels where the hardware injection waveforms are sent into the front end. The $IFO:CAL-INJ_TRANSIENT_EXC is the name of the channel that provides the input for transient injections, whereas the $IFO:CAL-INJ_CW_EXC is the channel where CW injections are sent in, $IFO equals L1 at LLO and H1 at LHO.

The excitations pass through there relative filter banks. These filter banks provide a means to apply an inverse actuation filter that adapts the waveform (which is in units of strain) so that it actuates the test mass correctly. Currently the transient injections make use of an inverse actuation filter, whereas the CW group apply the inverse actuation function directly to there waveforms before injection. The output of these filter banks can be read-out in the channels $IFO:CAL-INJ_TRANSIENT_OUT and $IFO:CAL-INJ_CW_OUT. These outputs are summed and sent through the HARDWARE filter bank. The output of the HARDWARE filter bank then passes to the 'HARDWARE_INJ_OUT' block output, which as described at the start of this section goes out of the DAC to the photon calibrator.

There is another block on the right side of PINJX labelled ODC. This is part of record keeping and is explained in section 5.4

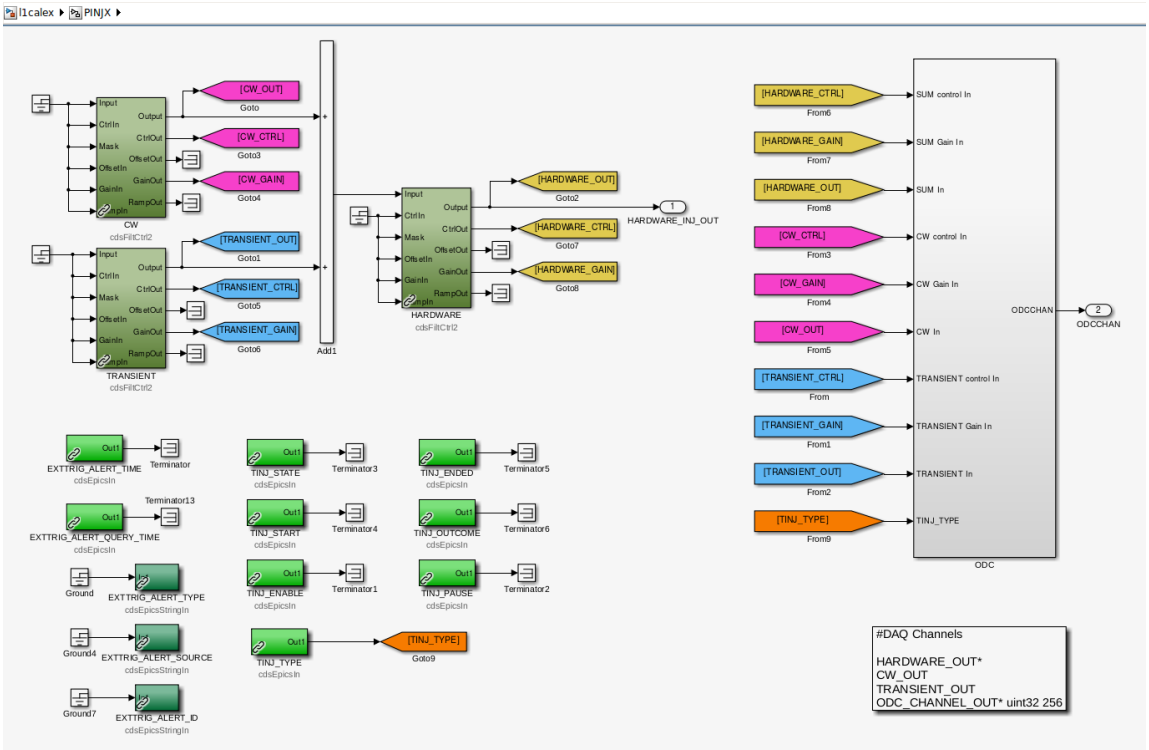

Figure 4: Simulink drawing of the PINJX front end code.

3.3 Injection Software

There are two main programs that perform the injections: the Transient Injection Guardian (INJ_TRANS) and the Pulsar Injection code (psinject). These programs have been written by members of the LSC, but are maintained by the site system administrators (please see LIGO-E1500265 for on-site software installation and set-up details).

3.3.1 Transient Injection Guardian

Both Burst and CBC injection files are carried out at the appropriate time by the transient injection guardian (`INJ_TRANS.py`). Guardian is the name given to the automation code that controls all interferometer sub-systems. `INJ_TRANS` at its core is just a wrapper for `awgstream`. The waveform files injected by `INJ_TRANS` are in units of strain and are saved in ascii format.

Most of the time, the transient injection code `INJ_TRANS` waits in standby mode. When reloaded it will parse a schedule file, which lists information about upcoming injections: filename, GPS time, and injection type (burst or CBC). When the GPS time for a scheduled injection approaches, it injects the signal into the transient injection channel: $IFO:CAL-PINJX_TRANSIENT_EXC. Transient injections are always canceled immediately following GRB alerts and may be optionally canceled by operators. A detailed description on the transient injection guardian can be found at LIGO-T1600183. This document also provides detailed instructions on how to schedule a transient injection.

Stochastic injections, although not technically transient injections, are also injected via `INJ_TRANS`.

3.3.2 Pulsar Injection Code

The CW group maintains on-the-fly injection software (part of lalsuite), which is designed to run continuously. Simulated CW signals are generated by `makefakedata` and piped to `psinject`, which combines the data streams from ≈14 signals before sending them to the excitation point. The CW injections go to the channel: $IFO:CAL-INJ_CW_EXC.

4 Scheduling, starting, and stopping

4.1 Scheduling

If you would like to schedule a transient injection, please contact the Hardware Injection co-chairs and or the site liaisons (contact details can be found at the wiki page). The site liaisons can also be reached by calling the appropriate control room:

- LHO: (509) 372-8202
- LLO: (225) 686-3131

CBC, Burst and Stochastic injections are carried out by the Transient Injection Guardian, see section 3.3.1 for details.

4.2 Starting and stopping

4.2.1 CW injections

The CW injection code (psinject) is run using Monit. If on-site at one of the observatories, starting and stopping the code is trivial. To start and stop CW injections:

1. Log onto a workstation in the control room.
2. Open an internet browser.
3. In the address bar type "l1hwinj/" if at LLO or "h1hwinj/" if at LHO.
4. Under 'Process' select psinject.
5. Select 'Start service' to start the code, and 'Stop service' to stop the code.

4.2.2 Transient Injections

Transient injections can be stopped by requesting the `KILL_INJECT` state in the transient injection guardian. If for some reason the guardian code needs to be stopped completely, select stop in the 'OP' pull-down menu. The code can also be paused via this pull-down menu.

5 Record Keeping

5.1 Logs

The applications `INJ_TRANS`, `psinject`, `makefakedata` all keep log files, which are helpful for debugging.

The transient injection log files, are stored with all the other guardian log files. There is a suite of guardian log tools that can be used to parse the guardian logs.

The starting and stopping of CW injections is logged by psinject in files like this one in `HardwareInjection/Details/log/`: `psinject_L1.14_09_11_09:26:37`. A new one is created every time psinject is restarted. This is where the state of the entire injection system is recorded. When the CW injection code crashses, it is usually recorded here.

5.2 EPICS

Injection properties are also recorded in EPICS channels. All EPICS channels described here begin with "H1:" or "L1:" depending on the interferometer.

- CAL-PINJX_EXTTRIG_ALERT_TIME: this channel is populated with the GPS time of the latest external trigger event (GRB/supernova), which is obtained by queries to GraceDB. The process is described in LIGO-T1500197. `tinj` skips any injections scheduled within 1 hour following that time.
- CAL-PINJX_TINJ_ENABLE: This is a switch used by the operator to turn injections on (1) or off (0). Thus, `tinj` skips the injection if this channel is equal to 0.
- CAL-PINJX_TINJ_PAUSE: This allows the operator to disable injections for a limited amount of time, and then resume them. When a GPS time is put into this channel, tinj should skip any injection scheduled for before that time.
- CAL-PINJX_TINJ_STATE: takes on the following values:
  - 1 = pending. set 5 minutes before the time of a scheduled injection (regardless of the control channel values described in the previous section).
  - 2 = streaming. sets immediately before calling awgstream to send the waveform.
  - 3 = completed. set immediately after awgstream returns. This state code does not attempt to indicate success or failure, only that the streaming of the waveform is complete. One minute after completion, `tinj` sets the channel back to 0 (idle).
- CAL-PINJX_TINJ_START: The GPS time of the scheduled injection. set 5 minutes before the scheduled injection, i.e. when entering state 1. It refers to the upcoming or in-progress injection when the state is 1 or 2, and to the most recent past injection when the state is 0.
- CAL-PINJX_TINJ_TYPE: A type code: CBC=1, Burst=2, DetChar=3, Stochastic=4. `tinj` sets this when entering state 1, i.e. at the same time it sets CAL-INJ_TINJ_START.

- CAL-PINJX_TINJ_ENDED: `tinj` clears this (sets it equal to 0) when entering state 1, 5 minutes before a scheduled injection. It sets this to the GPS time of the end of the injection regardless of outcome. Note that tinj does not clear this channel when starting up or shutting down, but instead let it keep its value to reflect the last attempted injection.
- CAL-PINJX_TINJ_OUTCOME. Set as follows:
  - 1 = success
  - -1 = skipped because interferometer is not operating normally
  - -2 = skipped due to GRB alert
  - -3 = skipped due to operator override (pause OR override)
  - -4 = injection failed
  - -5 = skipped due to detector not being locked
  - -6 = skipped due to intent bit off (but detector locked)

These definitions are adapted from Peter's EPICS planning page: https://wiki.ligo.org/viewauth/Calibration/HWInjBookkeeping. However, please note that this DCC copy is the official copy, and has evolved since Peter's wiki page.

5.3 Raw data

The total injection, CAL-PINJX_HARDWARE_OUT, is recorded so that there is a record of the injected signal at all times.

5.4 ODC Bit

ODC stands for "Online Detector Characterization." The ODC records a hardware injection bit to monitor if a transient injection is active.

The ODC bit runs completely independently from the hardware injection software such as `psinject` and `tinj`. The logic that informs the ODC bit exists at the front-end level. A simulink diagram of this ODC front-end code can be seen in Fig. 5. Here is a summary (courtesy of Ryan Fisher) of how the ODC bit works.

1. The $IFO:CAL-PINJX_ODC_CHANNEL_OUT_DQ channel generated at 16kHz, is recorded to the frames, downsampled to 256 Hz using the internal downsampling algoritm for all ODC channels (preserving 0's where possible by taking the bitwise AND across all samples on the input to form each output sample), and is also sent via IPC link to the ODC MASTER front end model (at full rate).
2. The $IFO:ODC-MASTER_CHANNEL_OUT_DQ channel is generated from the ODC MASTER front end code and written to frames at 16 kHz (full rate). Currently, this channel includes a bit for each of CBC, Burst and DetChar injections (the bits are 1 if there is no injection present). We have written an ECR that we will submit shortly that will change this to add a bit indicating Stochastic injections.

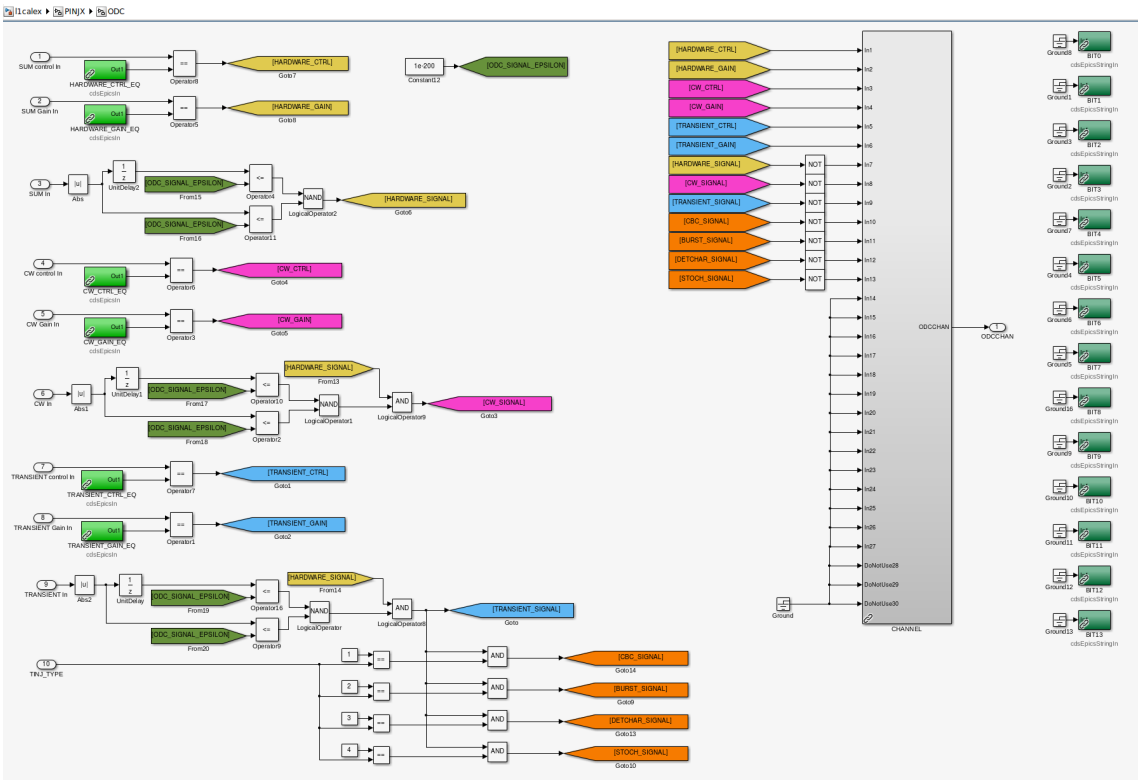

Figure 5: Simulink drawing of the ODC front end code for hardware injection record keeping.

3. The $IFO:ODC-MASTER_CHANNEL_OUT_DQ and $IFO:CAL-PINJX_ODC_CHANNEL_OUT_DQ channel are also captured by the the dedicated GDS frame broadcaster and sent to the GDS/DMT machine on the CDS network.

4. The SegGener monitor code that is managed by the DMT process manager is run in a loop on the incoming frames and converts the bits in the $IFO:CAL-PINJX_ODC_CHANNEL_OUT_DQ to segments that get written to DQXML files on a shared mount point via the TrigGener program (also managed via the DMT process manager). The communication between the two processes is UDP. The calculation for the generation of the segments is (example):
   H1:ODC-INJECTION_CBC_s bitnand “H1:CAL-PINJX_ODC_CHANNEL_OUT_DQ” mask=0x400 fraction=0
   That translates to looking at the channel’s 10th (counting from zero) bit (mask=0x400) and taking the bitwise NAND over 1 second’s worth of data, such that if any of the input bits are 0 (indicating an injection via the ODC system), we get an active segment output for that 1 second.

5. The DQXML is rsync’d from an LDAS headnode that shares the same mount point described in step 4 to a shared filesystem at CIT.

6. Right now, a publishing job is run from my account on the ldas-grid machine via cron that publishes the information contained in the DQXML files into the ER segment

database. In the future, this publication process will be moved to the dedicated hardware that is also used to store the O1 segment database (the publisher client and server for the database will be on the same machine).

6 Debugging

This section is in progress. For now, please refer to http://www.ligo.caltech.edu/~ethrane/hwinj/aligo/ for debugging tips. See also, http://www.ligo.caltech.edu/~ethrane/hwinj/ for tips from initial LIGO.

7 Editing this document

This is a living document, meant to evolve and grow the development of Advanced LIGO hardware injection infrastructure. In order to facilitate editing by multiple authors, please observe the following stylistic conventions:

- All graphics should be added as .pdf files (not .eps files). In order to compile the document, use `pdflatex`.
- When quoting text from computer code and/or parameter files, please use the "footnotesize" and "bigskip" commands.
- Try to avoid quoting large portions of computing code.