LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

-LIGO-

CALIFORNIA INSTITUTE OF TECHNOLOGY

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Document Type	DCC Number **T080135-00-C**	May 22, 2008
AdvLigo CDS Realtime Code Generator (RCG) Application Developer's Guide		
R. Bork		

This is an internal working note of the LIGO Laboratory

California Institute of Technology Massachusetts Institute of Technology
LIGO Project – MS 18-34 LIGO Project – NW 22-295
Pasadena, CA 91125 Cambridge, MA 01239

Phone (626) 395-2129 Phone (617) 253-4824
Fax (626) 304-9834 Fax (617) 253-7014
E-mail: info@ligo.caltech.edu E-mail: info@ligo.mit.edu

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

Table of Contents

1 Introduction

For the development of realtime controls application software, the LIGO Control and Data Systems (CDS) group has developed an automated realtime code generator (RCG). This RCG uses Matlab Simulink as a graphical data entry tool to define the desired control algorithms. The resulting Matlab .mdl file is then used by the RCG to produce software to run on an AdvLigo CDS front end control computer.

The software produced by the RCG includes:

- A realtime code thread, with integrated timing, data acquisition and diagnostics.
- Network interface software, using the Experimental Physics and Industrial Control System (EPICS) software and EPICS Channel Access. This software provides a remote interface into the realtime code.

2 Document Overview

This document describes the means to develop a user application using the RCG. It contains the following sections:

- Reference Section (3): The RCG produces software which integrates with various other components of CDS software. In addition, there are various files and services which must be configured prior to code operation. These items are covered under separate documentation, listed in the reference section.
- RCG Overview (4): Provides a brief description of the RCG, its components and resulting code threads.
- Application Development (5): Provides the basics for developing an application using the RCG, with a sample application file.
- Software Execution (6): Describes how to start, stop the software application.
- RCG Parts Library (7): Describes the various components supported by the RCG.

3 References

LIGO T080136-C CDS Software Admin Guide: Describes the various computer services and configuration files which must be in place to operate software produced by the RCG.
TBD CDS Software Development Guidelines: Provides the rules and guidelines for software development for applications which are to run in AdvLigo CDS.

4 RCG Overview

The RCG uses Matlab Simulink as a 'drawing' tool to allow for applications to be developed via a GUI. A basic description of this process, the RCG itself, and resulting application software is provided in the following subsections.

4.1 Code Development

Code development is done by graphically placing and connecting blocks in the Matlab Simulink editor. The 'building blocks' supported by the RCG are included in the CDS_PARTS.mdl file. The contents of the present file are shown below, with further descriptions of the blocks listed in Section 7.

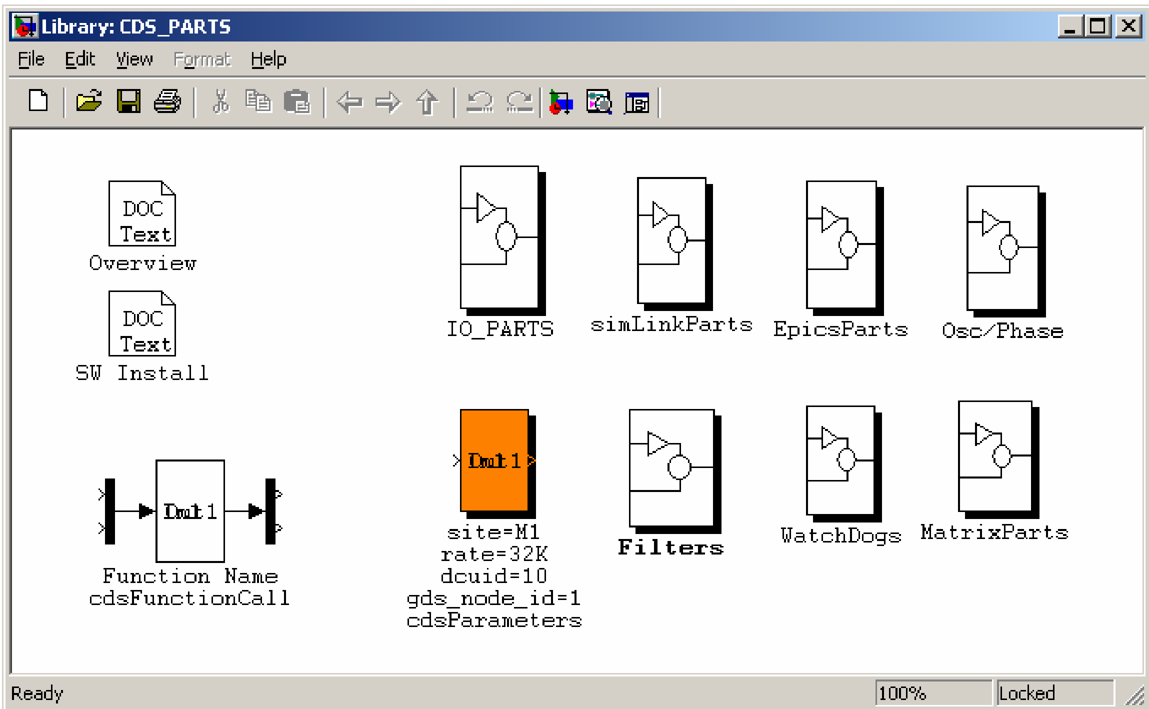


Figure 1: CDS Parts Library

Parts from the CDS library are copied (drag and drop) to the user application window and then connected to show processing/signal flow. A simple example is shown in the following figure. This example shows:

- A CDS parameter block, used to identify the desired sample rate and connection into the CDS infrastructure.
- A single, 32 channel ADC (adc_0)
- An ADC channel selector, which is here used to pick off the first 6 ADC channels.
- A Matrix part (IN_MTRX), which routes inputs to outputs with user selectable gain for each.
- Four CDS standard IIR filter modules (FM1-4)
- A single, 16 channel DAC

This Simulink diagram is then saved to a user defined .mdl file, which is then processed by the RCG to provide the final realtime and supporting software which run on a CDS front end computer.

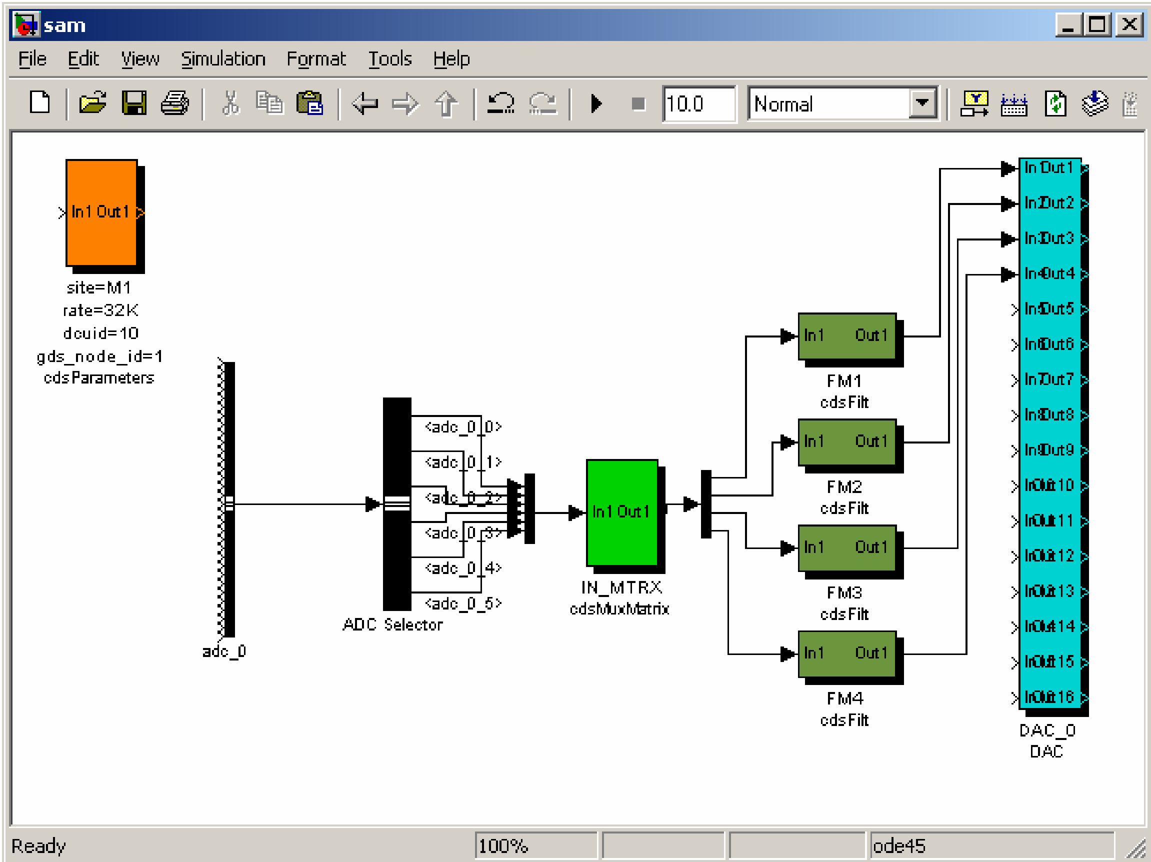


Figure 2: Sample Application

4.2 Code Generator

The code generation process is shown in the following figure and the basic process described below.

1) Once the user application is complete, it is saved to the user.mdl file in a predefined CDS software directory.

2) The 'make' command is now invoked at the top level CDS directory. This results in the following actions:

- A CDS Perl script parses the user.mdl file and creates:
  - Realtime C source code for all of the parts in the user.mdl file, in the sequence specified by the links between parts.
  - A Makefile to compile the realtime C code.
  - A text file for use by a second Perl script to generate the EPICS code.
  - An EPICS code Makefile
  - A header file, common to both the realtime code and EPICS interface code, for the communication of data between the two during runtime.
- The compiler is invoked on the application C code file, which links in the standard CDS developed C code modules, and produces a realtime executable.

- The Perl script for EPICS code generation is invoked, which:
    - Produces an EPICS database file.
    - Produces an executable code object, based on EPICS State Notation Language (SNL).

This code module provides communication between CDS workstations on the CDS Ethernet and the realtime FE code.

- Produces basic EPICS MEDM screens.
- An EPICS BURT backup file for use in saving EPICS settings.
- The header for the CDS standard filter module coefficient file.
- A list of all test points, for use by GDS tools.
- A basic DAQ file.
- A list of all EPICS channels for use by the EDCU.

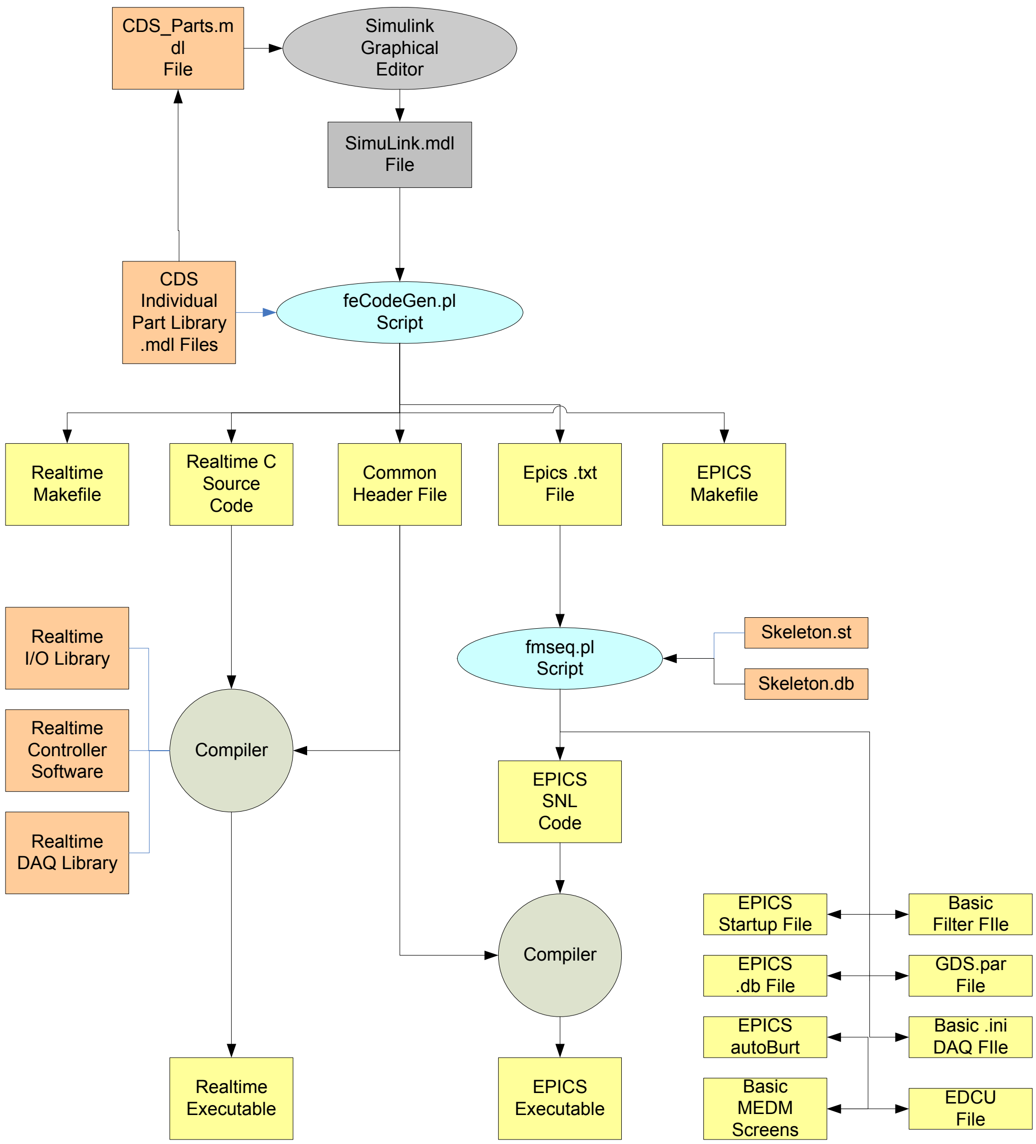


Figure 3: Code Generation

4.3 Runtime Software

The primary software modules to run on CDS FE computers are shown in the figure below. The intention is that all FE computers run the same generic code modules (highlighted in green), and that only the block labeled FE Application be specific to each FE computer.

The computer itself is to be a multi CPU / multi core computer, with up to 4 cores available. Generic Linux would be the operating system for the 'Non-Realtime' CPU, and up to 3 extra available running Realtime Linux.

The 'Non-Realtime' CPU runs the following tasks:

- GDS Test Point Manager (TPM) and Arbitrary Waveform Generator (AWG). In LIGO, one TPM and one AWG was run per IFO and communicated to the FE applications via Reflected Memory (RFM). In the AdvLigo scheme, the TPM/AWG runs on each FE computer and communicates to the FE application via internal memory space.
- EPICS based network interface. The purpose of this task is to relay realtime FE application information to/from EPICS operator interfaces. In Ligo, an EPICS interface task is run on a separate computer and communicates to the FE applications via RFM. In the AdvLigo scheme, there is an EPICS task on the FE computer to relay this information via the CDS network and internal computer memory.

Realtime CPUs in the FE computer run the realtime control and monitoring application. The code modules shown are inline compiled and run as a single task. The code modules that make up this task are:

- Synchronization software: This module controls initialization and timing of all other code modules. This code is slaved to the CDS timing clock used to synchronize the ADC modules.
- I/O Drivers: This code supports all input/output to the ADC/DAC modules in the I/O chassis, and data access to the CDS realtime network.
- DAQ/GDS: This module writes all data to the realtime network for data acquisition and handles all TP and AWG signals.
- FE Application: This code is specific to each FE and runs all of the necessary control algorithms, include CDS standard filter modules. To aid in the development of this software, a Matlab Simulink tool is provided. This allows the application to be developed through a standard GUI, then compiled with the above generic modules.

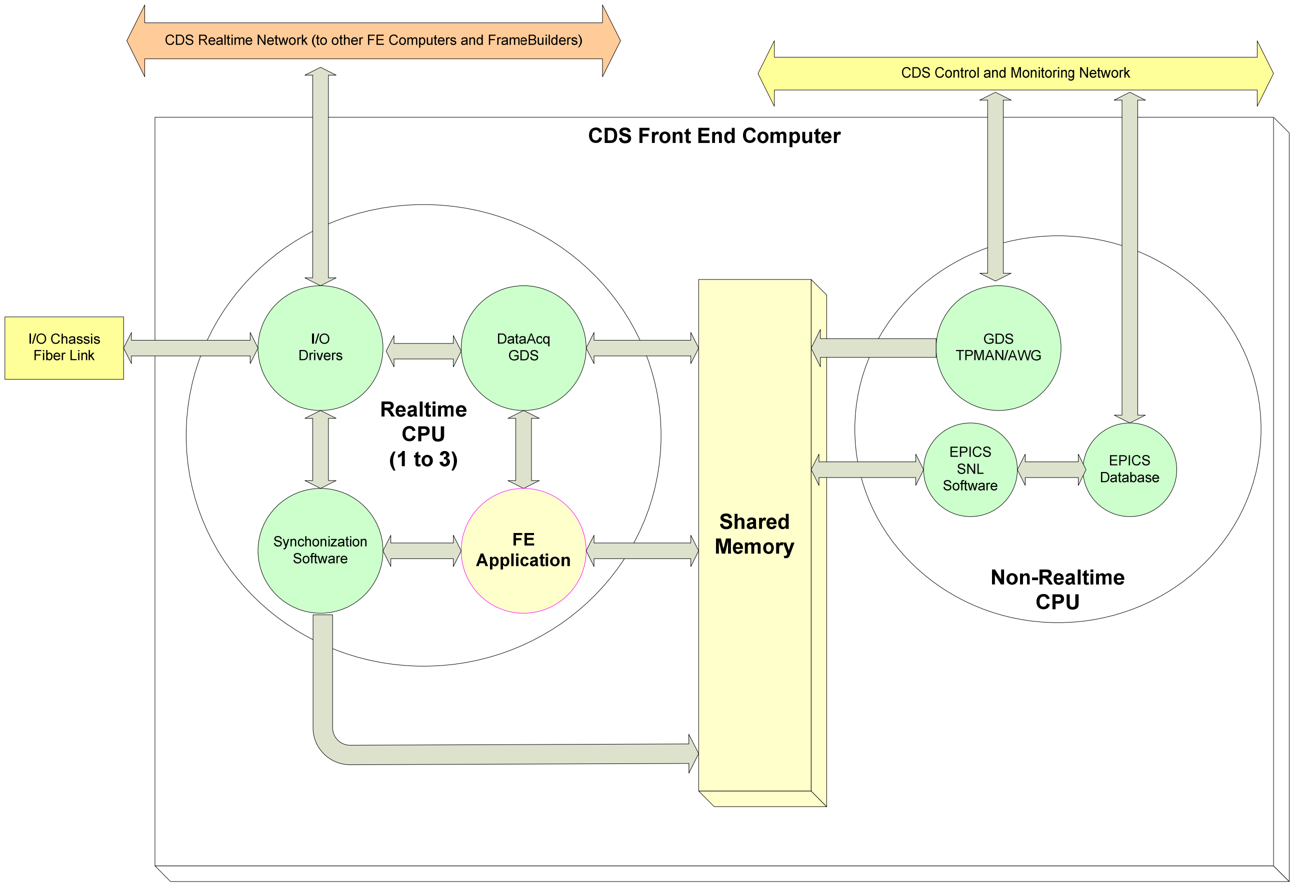


Figure 4: Runtime Software

5 RCG Application Development

This section describes how to use the RCG by stepping through a basic example.

5.1 Basic Code Development

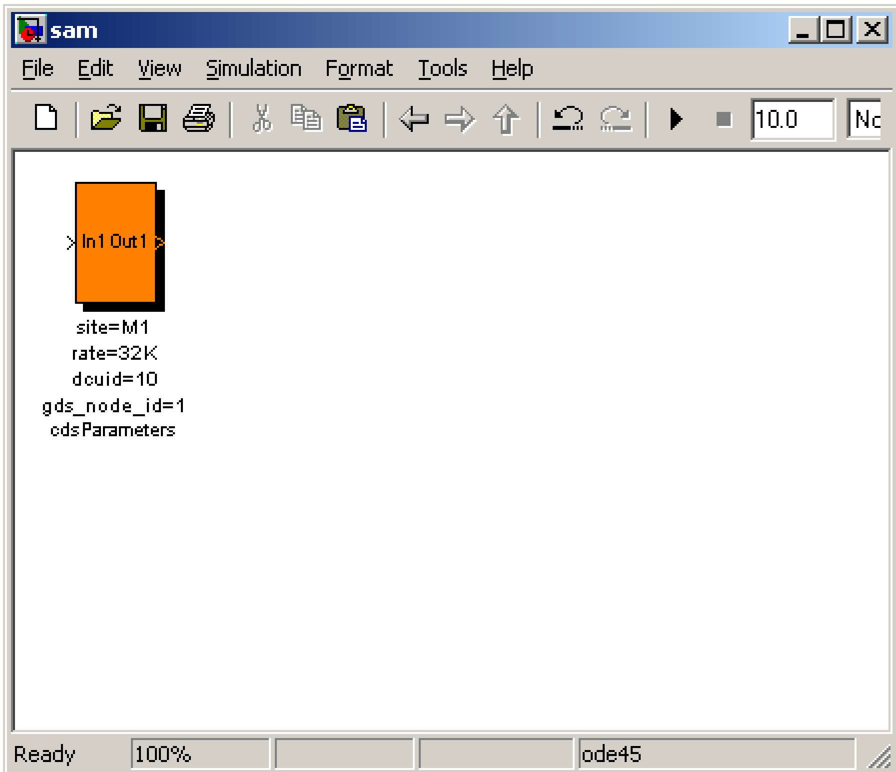
5.1.1 General Rules, Guidelines and Gotchas

Some overview notes before starting an application development process:

1) Only modules shown in the CDS_PARTS.mdl file may be used in the application development. Simulink native parts which may be used are shown in the CDS PARTS >> simLinkParts window. A description of all available parts is given in Section 7.
2) The tool is designed to work with the LIGO CDS standard naming convention, which includes:
    a. All channel names shall be upper case.
    b. All channel names shall be of the form A1:SYS-SUBSYS_XXX_YYY where:
        i. A1 is the Interferometer (IFO) site and number, such as H1, H2, L1, M1, etc., followed by a colon (:). The IFO part of the name is set using the *cdsParameters* part in the application model (see example in next section).
        ii. SYS is a three letter system designator, such as SUS, ISI, SEI, LSC, ASC, etc., followed by a dash (-).
        iii. SUBSYS and beyond are user definable, up to a maximum channel name length of 28 characters (limit set by EPICS software). Underscores are used to further break up the name, with any number of characters in between.
3) **Gotcha! THE USER APPLICATION MODEL MUST BE SAVED WITH A THREE (3) LETTER NAME!** The present release of RCG uses the .mdl file name, by default, as the three letter acronym for the SYS part of the channel names in the model. This naming 'feature' may be overridden by the use of 'subsystem' parts (see section 7.3.2).
4) All models must contain at least one ADC part and two IIR filter parts. This has to do with the compile scripts and shared memory setups running properly.

5.1.2 Example Model

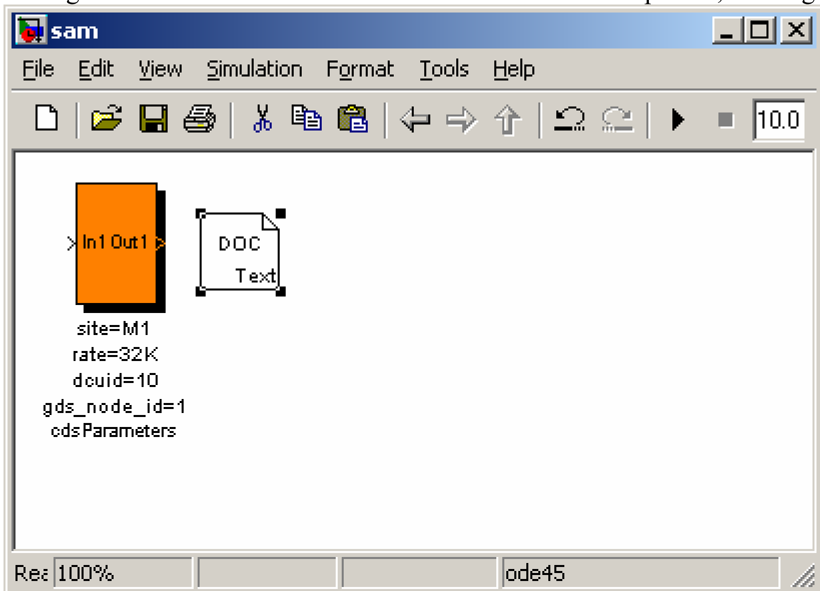
1) Start Matlab. Ensure that the advLigo/src/epics/simLink directory and subdirectories are in the Matlab path.
2) Open the CDS_PARTS.mdl file. This will be used to select parts for inclusion in the user model.
3) Select File>>New>>Model from the Matlab toolbar. This will open a new, blank Simulink window.
4) From the CDS_PARTS.mdl window, drag and drop a 'cdsParameters' block into the user model



window. One, and only one, of these blocks are required in every user application model.

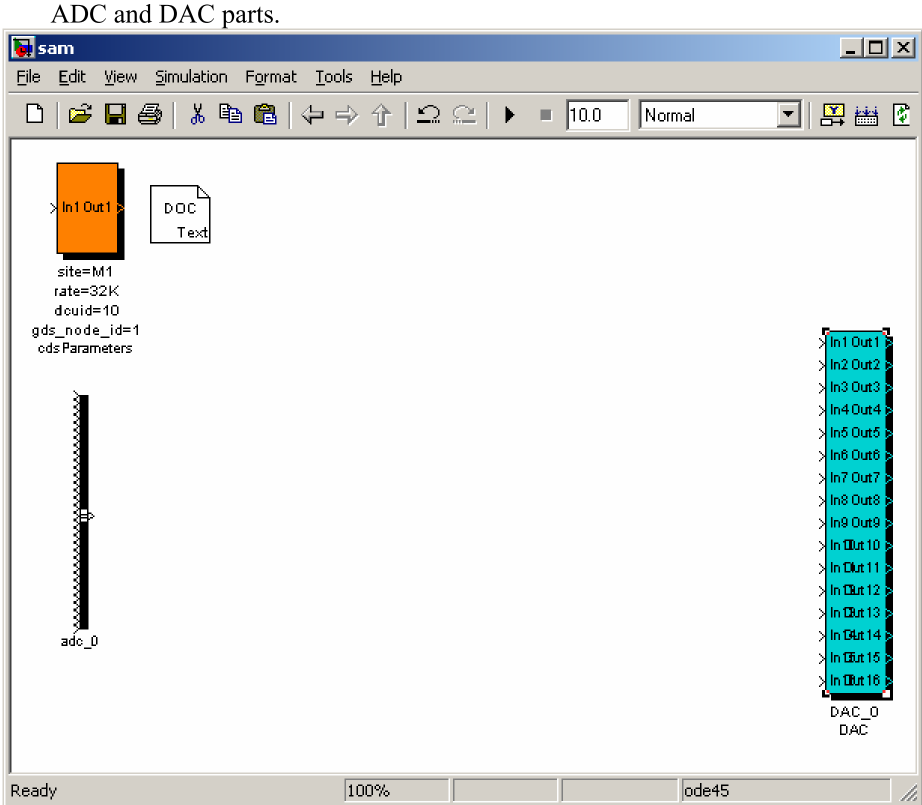
5) Define the parameters for this block by editing the text. The following is the minimal number of parameters which need to be defined. A complete list is given in section 7.1.1.
   a. Site=: The RCG will name all of the parts using the Ligo CDS standard naming convention ie IFO:SYS-SUBSYS_XXX_XXX_XXX to a maximum of 28 characters (EPICS limit). The IFO portion of all signal names for this model will be filled in by this site definition. In this example, M1: will be the prefix for all channel names in this model. If the code generated from this model is to run on multiple IFO, then multiple entries can be listed after site= eg site=H1,H2,L1.
   b. Rate: The rate field indicates the runtime sample rate of the realtime process. Presently supported are 32K (32768 Hz), 16K (16384 Hz) and 2K (2048 Hz).
   c. Dcuid: Every realtime process requires a unique id number to properly address the data acquisition system.
   d. Gds_node_id: In the same manner, a unique Global Diagnostic System (GDS) id is required for each realtime process, starting with 1 for the first model within a system. This is needed to properly attach the test point manager (TPM) and Arbitrary Waveform Generator (AWG) at runtime.
6) From the Simulink>>Model-Wide Utilities menu, drag and drop a DocBlock into the user model. Though this part is not required for the RCG, it is standard procedure to use these document

blocks throughout the model. There should be at least one at the top level, which gives an



overview of the application. These blocks should also appear at the top, left of each subsystem part, to further document what that component does.

7) Add an ADC and DAC module to the model. This is done by double clicking on the 'I/O Parts' block in the CDS_PARTS window, which opens the I/O parts window. Then, drag and drop the ADC and DAC parts.

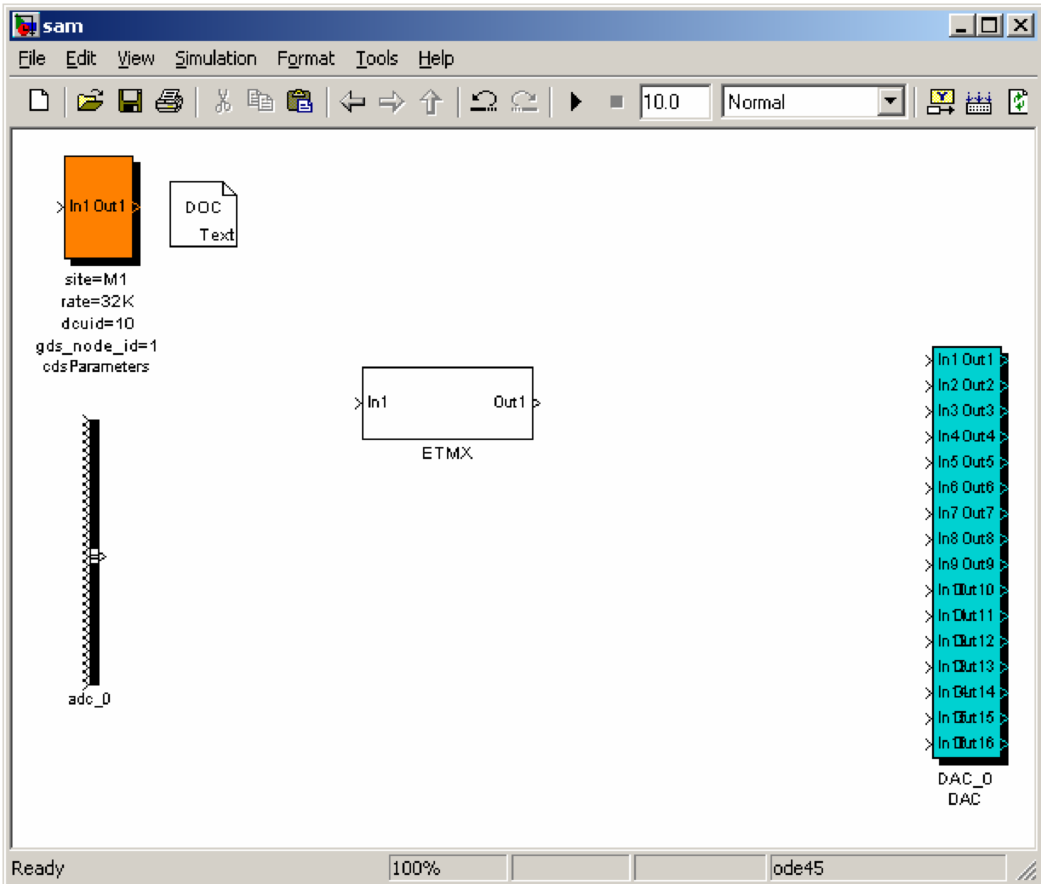


8) Save this model file as 'sam.mdl'. In the present RCG release, this must be a three letter name, as the three letters, in this case 'sam', are used as part of the signal names generated from this model. Therefore, for this model, all EPICS signal names will be prefixed by 'M1:SAM-'. It is intended

that this restriction will be removed in the next release such that the user can define multiple SYS name fields within the same model file.

9) Add a Subsystem block from the Simulink>>Commonly Used Parts Menu. While a simple 'flat' model can be used, it is more common to organize the diagram using subsystems. This is done to keep the model view from becoming too complex and also allows the reuse of subsystems as 'parts'.

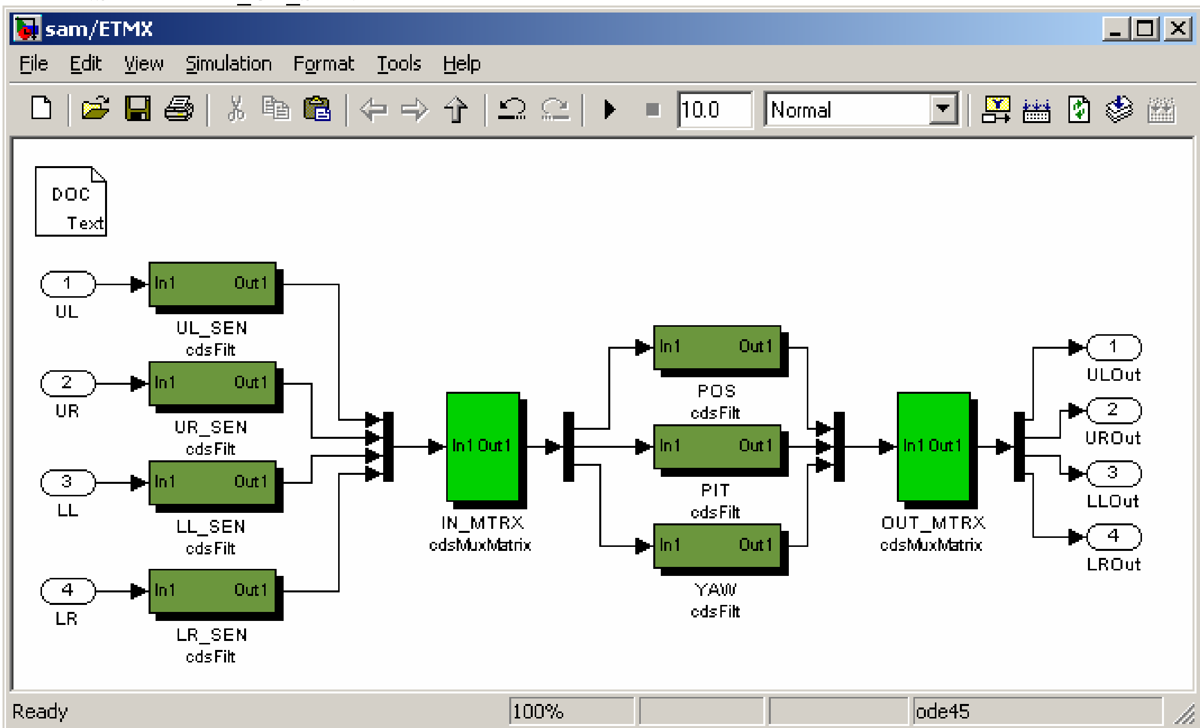
10) Change the name of the Subsystem part to 'ETMX'. Note that the convention is to name all parts in the model using upper case, in keeping with the CDS naming convention. In the following steps, blocks will be added to the Subsystem block. The name of every item within the subsystem block will later be prefixed by 'M1:SAM-ETMX_', where M1 came from the *cdsParameters* block, SAM comes from the name of the model file, and ETMX comes from the subsystem part name.



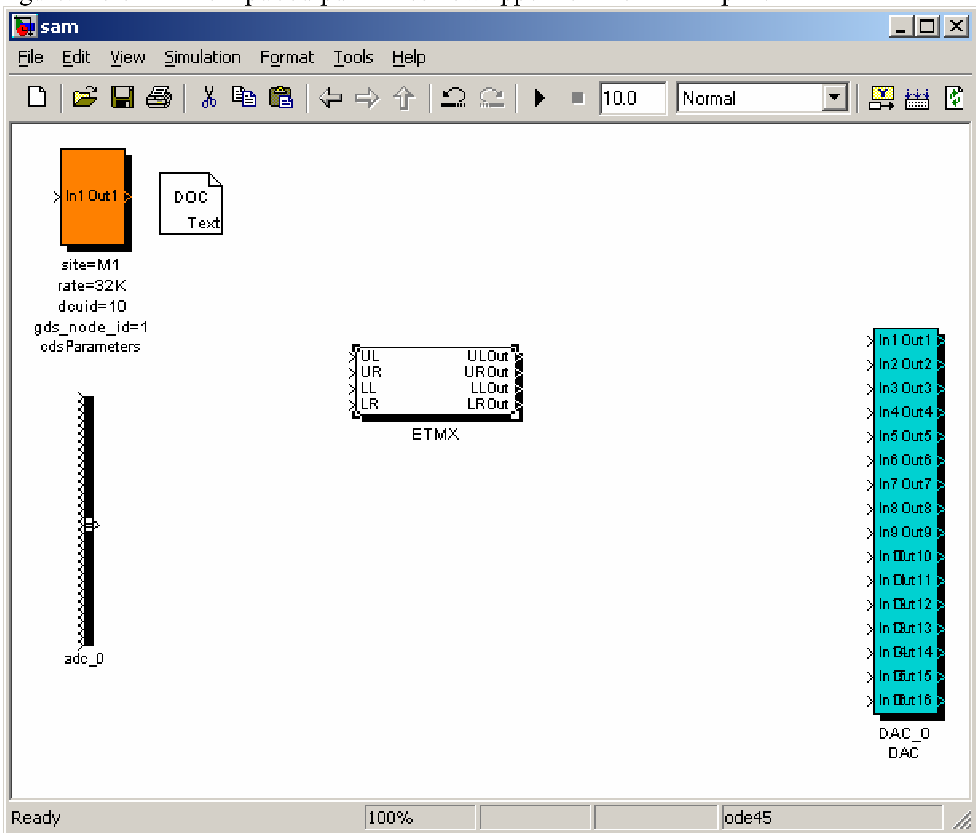
11) Double click on the 'ETMX' subsystem part, which will open a window showing an input connected to an output.
    a. Disconnect the link between In1 and Out1.
    b. Copy 'In1' several times until there are In1 thru In4. Do the same with 'Out1'. Change the names of these parts to something more meaningful, as these connection points will appear as part of the subsystem part at the top level diagram. In the case of this example, the four inputs are renamed UL, UR, LL, LR and the four outputs are ULOut, UROut, LLOut and LROut. The user is free to define any name for these input/output parts, as the RCG will discard them when it builds the code.
    c. From the CDS_PARTS, select and place filter modules and matrix parts into the subsystem window and make connections and name changes until the window appears as shown in the next figure. Note that the number of inputs and outputs to a matrix part can be changed by double clicking on the mux/demux parts and entering the number of desired ports.

d. In keeping with CDS standards, add a ‘DOC’ block in the upper left to document this code section.

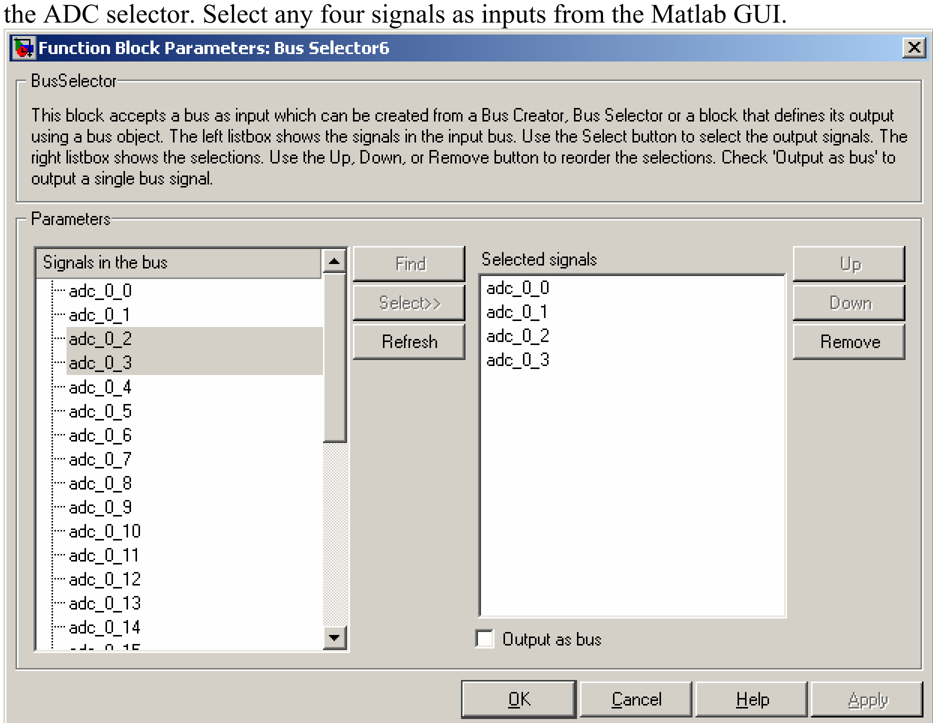
12) After step 11, the subsystem block window should look like the following figure. When the code is generated , the EPICS names of these channels will be prefixed by M1:SAM-ETMX eg M1:SAM-ETMX_UL_SEN.



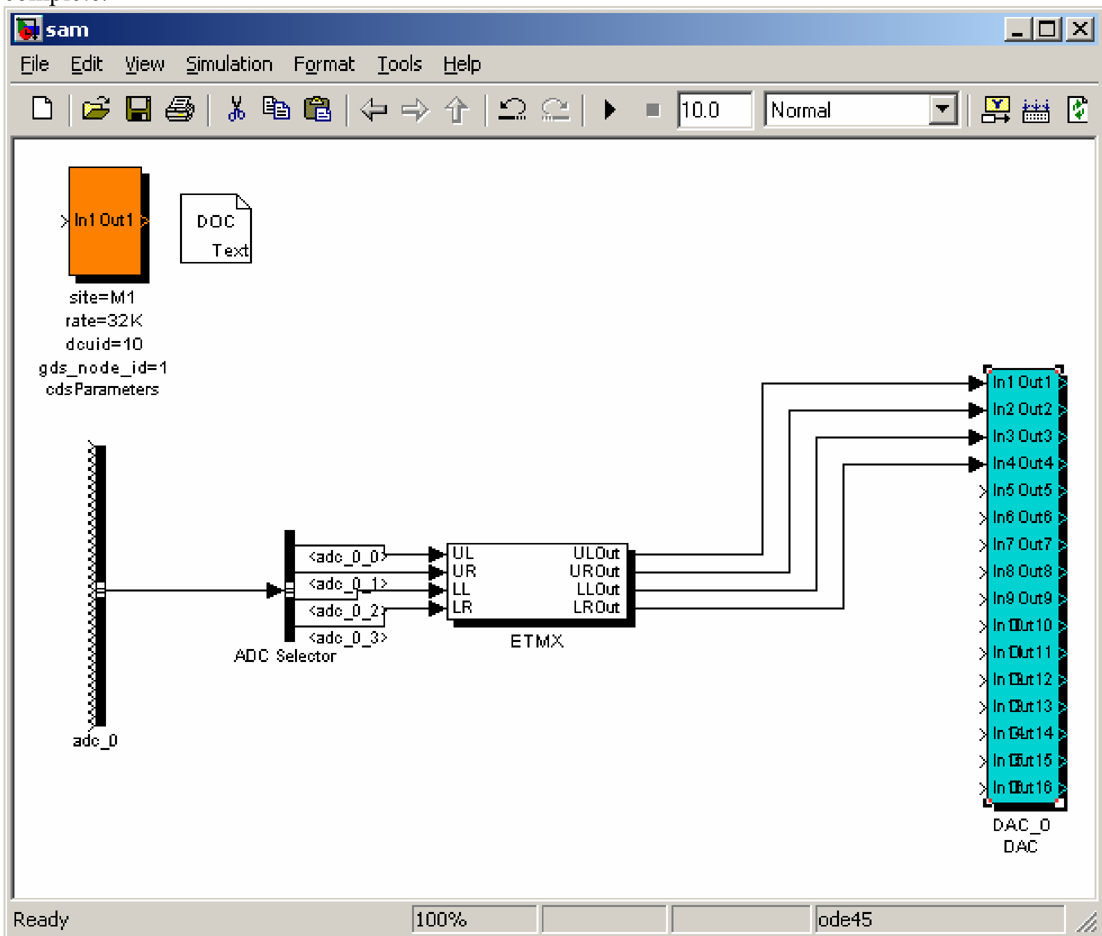
13) Close the subsystem window. The top level window should now appear as shown in the following figure. Note that the input/output names now appear on the ETMX part.



14) Next step is to connect ADC channel to the ETMX part. From the CDS_PARTS>>I/O Parts drag and drop an ADC Selector part. Connect the adc_0 part to the adc selector part. Double click on the ADC selector. Select any four signals as inputs from the Matlab GUI.



15) Connect the ADC selector to the ETMX part and ETMX part to the DAC and this sample model is complete.



5.2 Code Compilation and Installation

The software may be compiled in any user area that includes the cds/advLigo source code tree from the CDS CVS software repository. This space must be mounted to a computer which has RT Linux installed, as all compilation must be done on a realtime computer..

To compile the code:

1) Place the Matlab .mdl file in the directory *advLigo/src/epics/simLink*
2) Move to the the *advLigo* directory.
3) Type 'make SYS', where SYS is the three letter name of the .mdl file. This command will result in the compilation of all the code, including EPICS.

Once the code is compiled, a few more commands need to be run from the *advLigo* directory to install the code for execution.

1) make install-SYS : This command installs the code in the appropriate directories for execution and makes the automated startup commands. The EPICS code will be copied to /cvs/cds/site/target/SYSepics and the front end code will be moved to /cvs/cds/site/target/SYS.
2) make install-daq-SYS : The creates the data acquisition file in */cvs/cds/site/chans/daq/*.
3) Make install-screens-SYS : Installs automatically generated MEDM screens in */cvs/cds/site/medm/IFO/SYS* directory.

5.3 Defining Multiple Models For One Computer

During runtime, the RCG code requires one or more multi-core processor to operate. Core 0 is reserved for standard Linux tasks and the realtime support tasks, such as EPICS. Remaining cores may be used by the realtime code threads. By default, as in the case of the example model, at runtime, the realtime code will run on CPU 1.

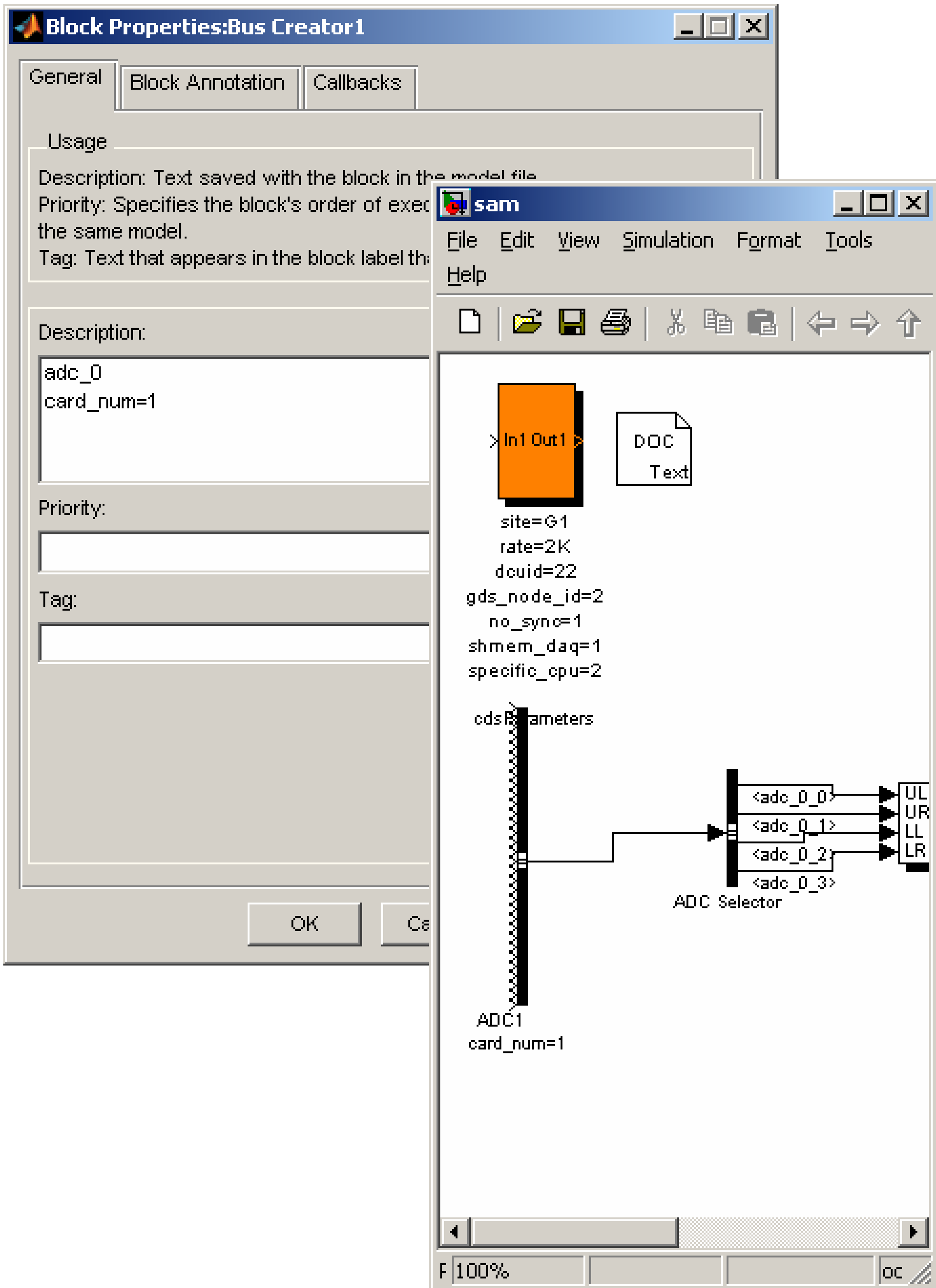
If it is desired to run multiple applications on the same computer, a couple of things need to be done:

- The support services must be configured, as described in the SysAdmin Guide.
- Applications which are destined to run on Core 2 and higher must have some additional parameters set:
  - The cdsParameter part must have specific_cpu=num, where num is the core number on which to run. This number may be 2 to 15, dependent on the number of cores on the target computer.
  - Since, in the present release, models may not share I/O cards, these cards require further definition in the model file.

Taking the previous example model as an example, to have this model run on CPU core 2 and make use of ADC card 1 (instead of the default core 1 and ADC 0 of the example model), the following changes would need to take place:

- The cdsParameter block would need to have specific_cpu=2 added.
- The adc_0 block will need card_num=1 added to the block description. This is done by right clicking on the adc_0 part and selecting Block Properties. This will bring up the following window, where card_num needs to be added to the Description field.

The Block Properties window and resulting model changes are shown below. Note the even though adc_num has been set to 1, the user application still needs to use ADC 0 and adc_0 signals for its first ADC.


Block Properties:Bus Creator1
General
Block Annotation
Callbacks
Usage
Description: Text saved with the block in the model file
Priority: Specifies the block's order of exec
the same model.
Tag: Text that appears in the block label th
Description:
adc_0
card_num=1
Priority:
Tag:
OK
sam
File Edit View Simulation Format Tools
Help
In1 Out1
DOC
Text
site=G1
rate=2K
dcuid=22
gds_node_id=2
no_sync=1
shmem_daq=1
specific_cpu=2
cdsParameters
<adc_0_0>
<adc_0_1>
<adc_0_2>
<adc_0_3>
UL
UR
LL
LR
ADC Selector
ADC1
card_num=1
100%

6 Running the RCG Application

6.1 Loading and Executing the software

When the code is compiled and installed, it is ready to run, as outlined below. However, for data acquisition and global diagnostics to function with this software, certain parameters must be set up for these services to work properly. See the RCG SysAdmin Guide for instructions on how these parameters are set.

6.1.1 Automatic Scripts

During the make install process, scripts are generated in the /cvs/cds/site/scripts area for conveniently starting and stopping the user application. This directory should be put into the user's PATH. Note that the user must have super user privileges, as the realtime code needs to be inserted into the kernel.

To start the RCG processes, type 'startsys', where sys is the name of the model file. This will result in:

- The EPICS code being started, along with an automatic restoration of the last EPICS settings (if EPICS BackUp and Restore Tool (BURT) is in the user's path and a backup had been made previously).
- The awgtpman process will be executed to provide GDS support for this system. Note again that this task will only function properly if the appropriate system parameters have been set up, as described in the SysAdmin Guide.
- The realtime code thread will be executed and inserted into the kernel of CPU 1.

To verify that the software is functioning, use the auto generated MEDM screen, described below in section 6.2.1. There are also log files produced in the target areas for the EPICS and realtime code which provide additional diagnostic information.

To stop the software, execute the *killsys* script, where again sys is the model name. This will kill all tasks associated with this model.

6.1.2 Manual Code Execution

The EPICS and realtime code may also be executed manually from a command line. This is typically only done when trying to diagnose problems or the realtime code modifications do not affect the EPICS code, such as modifications to user supplied C code modules, and it is not desired to constantly stop and start the EPICS side.

During the make install-sys process, two target directories are built in /cvs/cds/site/target, one for the EPICS components and one for the realtime code. EPICS and the realtime code may be started independently by using the startup command in those directories. Note that EPICS must be running prior to starting the realtime code.

6.1.3 Log Files

Ioc log
RT log
Dmesg

6.2 Performance Considerations

TP and DAQ loading

6.3 Auto Generated MEDM Screens

During the make process, various EPICS displays are automatically generated. These are fairly simple displays, to get the user started and to provide for quick testing and some quick 'copy paste' points to use in building custom operator displays. After the make install-screens-sys command is executed, these displays will appear in the */cvs/cds/site/medm/ifo* directory.

These display are:

- IFOSYS_GDS_TP.adl : Provides basic diagnostic information for the running application.
- IFOSYS_ADC_X: Provides a display of all ADC input channels for quick signal checkout. Note that, in the present release, this display will only show ADC channels which are directly connected to filter modules or EPICS outputs in the model file.
- Filter module displays: For every filter module in the model file, a generic filter module display is generated.
- Matrix displays: For every matrix defined in the model, an associated EPICS display is generated.

These various displays are further described in the following subsections.

6.3.1 GDS_TP Display

A basic system diagnostic display is built for each system during the build process, with an example shown below. This display includes the following:

Upper Left: DAQ data and status

- Dcu Id: The DAQ node id for this system. Each realtime process has a unique and separate id number on the network, as defined by matlab model.
- Chan Count: Number of channels presently being recorded by the DAQ system, as defined by the user in the system .ini file.
- DAQ Rate: Total data rate in Kbyte/sec for configured DAQ channels
- DAQ + TP rate: Total data rate in Kbyte/sec being transferred by this process to the framebuilders, which is a combination of DAQ channels and selected testpoints.
- CRC: This is the CRC checksum, calculated from the .ini file. This number is checked by both the framebuilder and the realtime front end to verify that they have read the same .ini file.
- DAQ Reload button: When pressed, causes the realtime front end to reload the DAQ .ini file. This is to be asserted whenever a new DAQ configuration has been set by the user. Note that the frabebuilder must also be reset at this time for DAQ configuration to be computerd.
- Framebuilder status info: The next sub block contains framebuilder status information, as it pertains to this system. In the Ligo system, two framebuilders run on the network for redundancy, but only one framebuilder is required. The fields shown beside each framebuilder are:
    - Status block, with two red/green indicators. The left most indicator is front end status and right most is framebuilder status for this system.
    - Status: A hex status number, with meaning given below this block.
    - CPS: Transmission errors per second. The framebuilder performs CRC checksums on all data received from the front end system. The number in this field should be zero, but if there are continuous errors, the count will be indicated here and in the following field.
    - Sum: The total number of transmission CRC errors since the framebuilder counter was reset.

Lower Left: Front end realtime process status:

- Coeff Reload Button: Pressing this icon will cause the front end to reload all filter coefficients listed in its coefficient file.
- Diag Reset: Cause the reset of diagnostic values, including the CPU Max Time.
- IRIGB Diff:
- 1PPS Trig:

- ADC Sync:
- USR Time: The maximum time, in usec, that it takes to cycle through the user application, which was developed using the RCG.
- CPU Max: Maximum amount of time, in usec, that it took to run through a single cycle of the software, including the user application and overhead, such as I/O and DAQ, of the front end code. This field is held to the highest value until reset using the Diag Reset icon.
- CPU: Similar to above, but this field is the maximum time during the last one second period.
- BURT Restore: When the software is started, the realtime code will wait for restoration of user setpoint values before running. This is typically done through a BURT restore. However, this can be overridden by entering a '1' into this field.

Center Section: The realtime code continuously checks for ADC and DAC overflows ie greater than 32000 counts or less than -32000 counts. If these values are exceeded, the realtime code will clamp the value to +/- 32000 and report the error via overflow counters.

- Total and Reset (top): This field reports the total number of overflows detected for all channels. This is a running count, which may be reset using the Reset icon.
- Below each ADC and DAC on this display are individual overflow counters for each channel. These fields indicate the number of overflow detected per second to help identify which channel(s) is/are having problems.

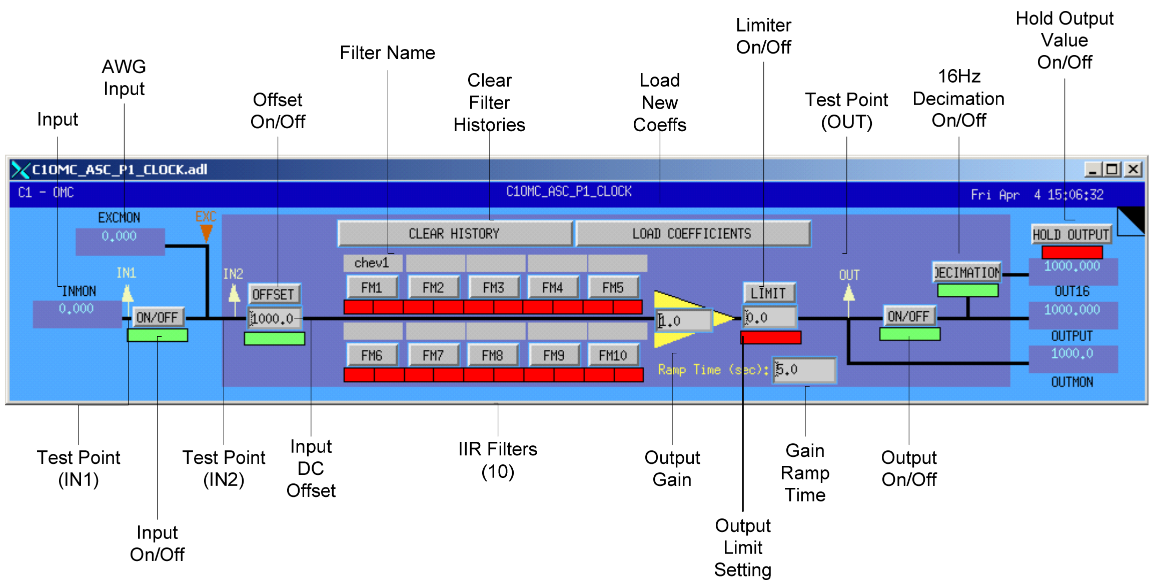
Right hand section: This section provides a list of those GDS testpoint and excitation channels which are presently selected. There is also a meter representation of the maximum CPU time, same as the value in the CPU field at the lower left of this display. The meter limit is set by the sample rate of this system. For example, this system shown was set to run at 32KS/sec, so a single code cycle must complete in under 30 usec to function properly. For 2KS/sec systems, the max time on the meter would be 480usec and 60usec for a 16KS/sec system.

6.3.2 ADC Input Display

6.3.3 Standard Filter Module Display

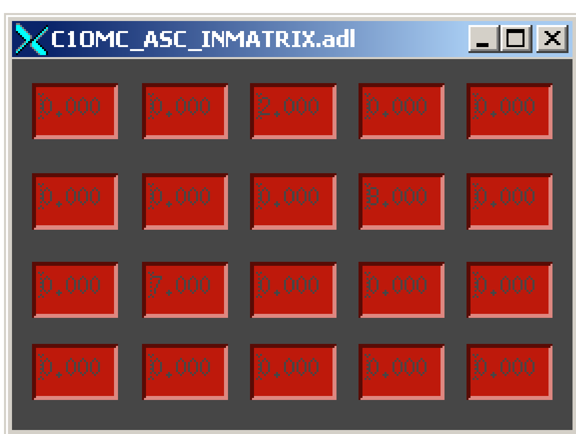
For each IIR filter module defined in the user model, a standard MEDM screen will be produced as part of the build process. An example screen is shown below. This screen contains the following EPICS I/O:

- INMON and Input On/Off: Displays the filter module input value. The following on/off switch applies/removes the input signal from the filter bank.
- EXMON: The value of an excitation input. This field is typically 0.0 except when a GDS excitation signal is being applied.
- Offset value and Offset On/Off switch: Allows the user to add a DC offset to the input prior to entering the filter bank. The indicator below the offset value will be green if turned on and red if turned off.
- Filter module names and selections: The 10 available filters per bank appear to the right of the offset value field. Names, as defined using foton, appear above each filter selection button. The filter selection buttons are used to turn the filters on/off. Below each filter button are two status indicator block. The left box indicates if a filter has been selected to be turned on (green) or off (red). The right box indicates when the realtime code has actually turned on (green) the filter or turned off (red) the filter.
- Gain and Ramping: The signal out of the filter bank may be multiplied by the gain setting. To avoid a sudden excursion of the signal when a new gain is selected, this gain may be ramped over the number of seconds entered into the ramp time setting. This ramping is performed by the realtime code. When the realtime code gain is not the same as the entered gain, ie during the ramping, the background of triangle surrounding the gain setting will be yellow. Once the ramping is complete, the triangle will become black.
- Limit setting and on/off switch: The output of the filter bank may be limited by the user by setting the limit field and turning the limit switch on (green indicator). The realtime code will then limit the output to +/- the limit setting.
- Output On/Off and OUTPUT monitor: Turns the output on/off, with the filter bank output value displayed in the OUTPUT field. Note that the OUTMON (output testpoint) will still have the output of the filter bank.
- Decimation On/Off switch and OUT16 field: The realtime code decimates the filter bank output to 16Hz, the resulting value being placed in the OUT16 field.
- Hold Output: When selected, the output of the filter module is held to the present value (seldom used).
- CLEAR HISTORY: When selected, clears the history of all filters within the filter module. This is typically used when integrators have been defined and have rung up to a large value.
- LOAD COEFFICIENTS: Reloads the filter coefficients for this filter module.



6.3.4 Matrix Display

For each matrix defined in a model, a matrix screen is automatically generated, as in the following example screen. By default, matrix elements which are set to 0.0 have their backgrounds set to red. Any other value results in a green background.



6.4 Additional Run Time Tools

Along with EPICS MEDM, various additional tools are available to support realtime applications during runtime. These are listed below, with a few described briefly in the following subsections. For more detailed information, see the appropriate user guides for these applications.

- EPICS BackUp and Restore Tool (BURT): Used to save and restore operator settings.
- EPICS StripTool: Provides strip charting for EPICS channels.
- Dataviewer: Allows users to view DAQ and GDS TP channels, either live or from disk.

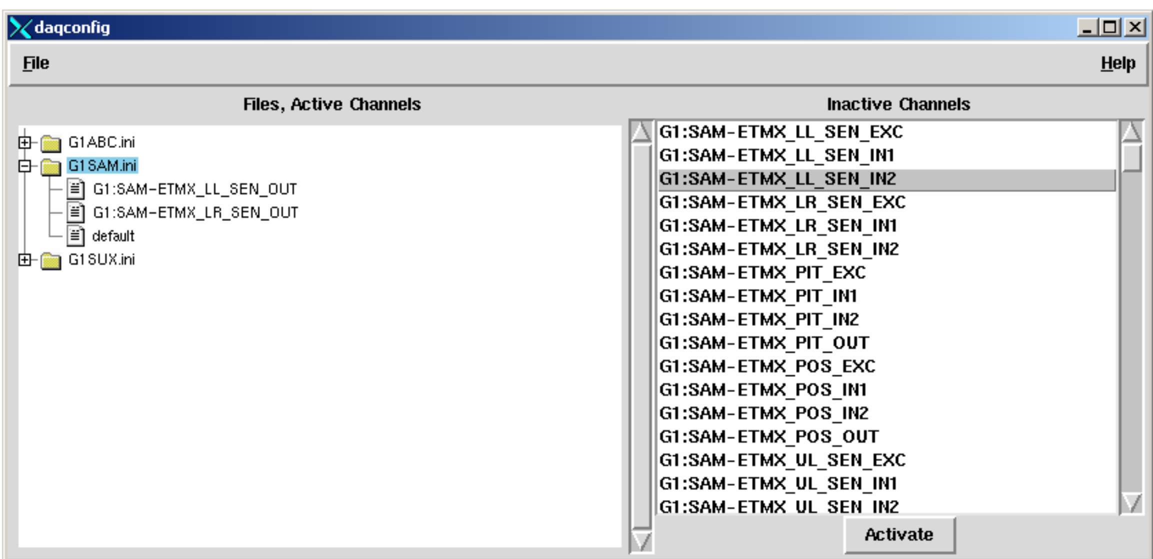
- ligoDV: Based on the GEO developed tool, this is a Matlab tool for reading, plotting and analyzing DAQ data.
- Diagnostic Test Tool (DTT): Allows for analysis of live or recorded DAQ/TP data, particularly useful for calculating and plotting transfer functions.
- DaqGui: A graphical user interface for setting up DAQ channels.
- Foton: A GUI for the development of filter coefficients for use by the realtime software.
- Ezca based scripting tools, along with TDS scripting tools. These tools allow for the addition of automated scripts which may be used to sequence through operator settings automatically.

6.4.1 DAQ GUI

Screen shots of the DAQ configuration GUI are shown below. This tool is used to configure channels which are to be stored by the DAQ system. By default, all filter module input and output testpoints are available to be recorded, but must be selected from the list and set to be stored to disk, if desired.

After the make install-daq-sys command is executed during the build phase, a DAQ file with all available channels is built in the */cvs/cds/site/chans/daq* directory (with suffix .ini). In addition, a *daqconfig* script is generated in */cvs/cds/site/scripts* to attach this file to the DAQ GUI. Running this script will bring up the following window, with a list of all .ini in the daq directory. ***Note that this GUI is only used to configure 'fast' data channels, that is channels which may be recorded at up to the sample rate set for that system. Slow (EPICS) channels may also be stored to disk at 16Hz, but must be separately configured, as described in section 6.3.2 below***.

Running the script will bring up the following display. This display will list all systems which have .ini files in the daq directory. Systems and active DAQ channels are shown in the left half of the window. A list of available channels is shown to the right.

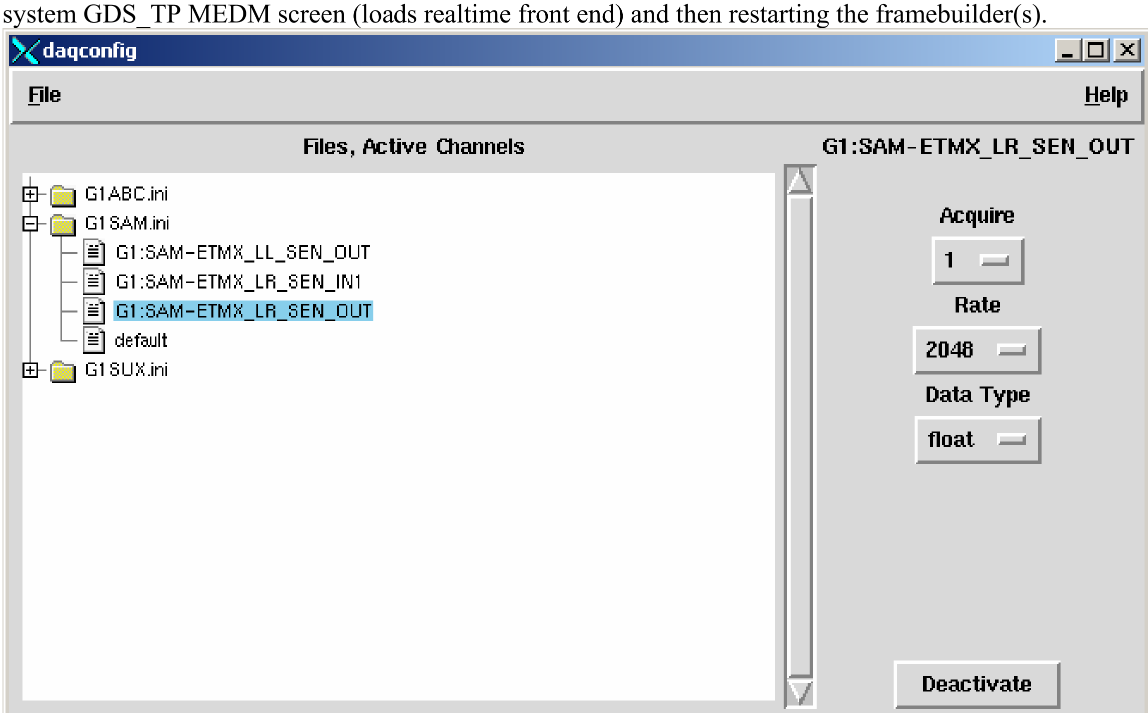


Double clicking on any signal name on the active or inactive list will result in the following window being opened. From the window, the following may be selected:

- Acquire (0 or 1): Setting this value to '1' will cause the channel to be continuously sent to the Framebuilder at the prescribed rate and stored to disk. Setting this value to '0' will result in the channel being continuously sent to the framebuilder, but it will not be recorded to disk.
- Rate: The data storage sample rate may be set from 256 samples/sec up to the native sample rate of the system, as defined during the RCG model build. Decimation filters in the front end code will properly downsample the desired channels prior to sending them to the framebuilder.
- Data Type: The data type may be set to float, int, or short. Again, the realtime front end code will perform the conversion prior to transmission.
- Deactivate: This will remove a signal from the active list.

Note that after a signal has been activated as a DAQ channel, the sample storage rate is appended to the end of the channel name. For example, if the channel name is H1:SUS-ETMX_LR_SEN_IN1 and has been set to be acquired at 256 samples/sec, the resulting DAQ channel name will becomre H1:SUS-ETMX_LR_SEN_256.

Once all of the desired changes have been made and the new file saved, it will be necessary to load the new configuration before it will become active. This is done by pressing the DAQ Reconfig button on the system GDS_TP MEDM screen (loads realtime front end) and then restarting the framebuilder(s).



6.4.2 EPICS DAQ Configuration

EPICS channels to be stored by the DAQ system are named in a single EPICS.ini file for all systems running on the same network. This file must be located in the */cvs/cds/site/chans/daq* directory, and added to the *master* file list (see SysAdmin Guide).

An example file is shown below. The header portion must be as shown. Individual channels to be recorded may then be added, one channel per line, with braces around each channel name.

*********** Sample File **********************

```
[default]
dcuid=4
datarate=16
gain=1.0
```

```
acquire=1
ifoid=0
datatype=4
slope=1.0
offset=0
units=NONE

#
# HEPI channels
#
[M1:SEI-BSC_HP_INMON]
[M1:SEI-BSC_HP_OUT16]
[M1:SEI-BSC_RX_INMON]
[M1:SEI-BSC_RX_OUT16]
[M1:SEI-BSC_RY_INMON]
```

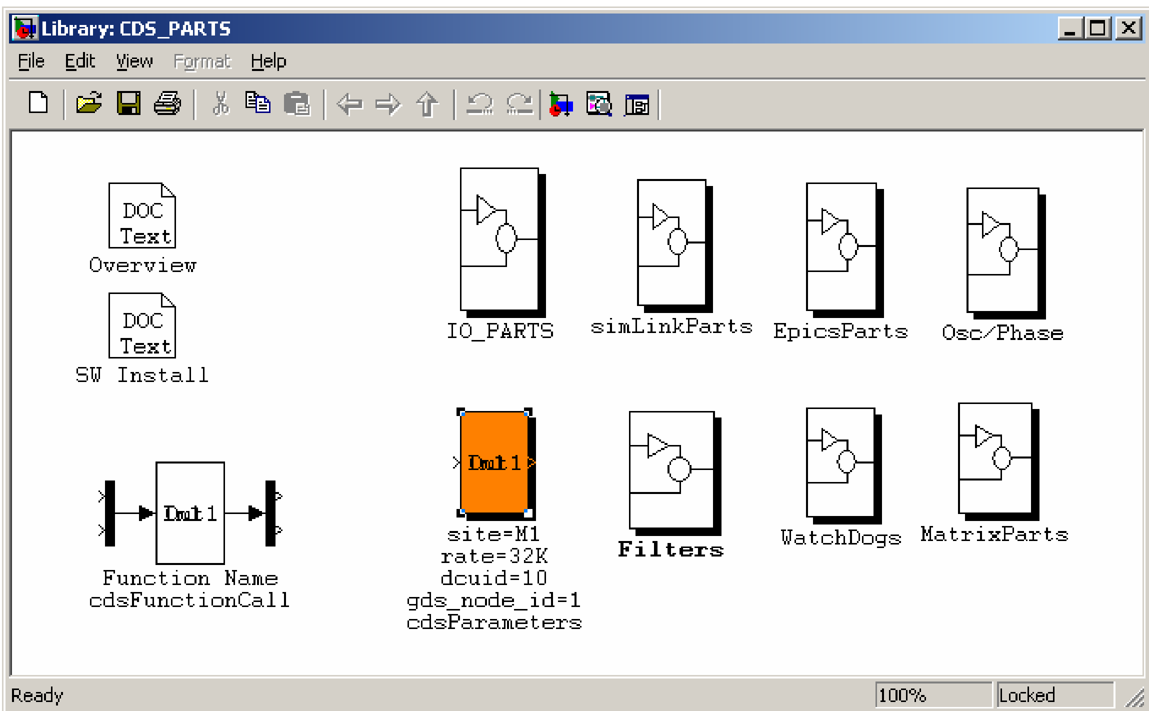
7 RCG Software Parts Library

The CDS_PARTS.mdl file contains symbols for the modules supported by the RCG. Only parts defined in this library may be used with the RCG ie the RCG does not support the full set of Simulink parts and some custom parts have been added for specific purposes.

7.1 Top Level Modules

CDS parts at the top level of the library include:

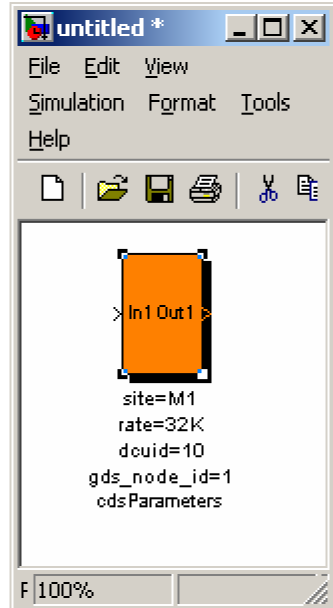
- cdsParameters
- cdsFunctionCall



7.1.1 cdsParameters

7.1.1.1 Function

The purpose of this module is to define basic runtime parameters needed by the CDS RCG.



7.1.1.2 Usage

This module must appear once, and only once, at the top level of an RCG application model, by convention usually in the upper left hand corner. It contains four fields which must be edited.

1) site: Somewhat of a misnomer, this field is actually the designator for the site and interferometer on which the code will run. This can be a single entry (as shown) or comma delimited for multiple IFO use, such as site=H1,H2,L1. In this case, the RCG will generate code for three IFO. This field will be used in the EPICS channel generation as the first two characters of the channel name. In the example at right, all channels names within this RCG model will have an M1: prefix.
2) rate: The sample rate of the generated code must be defined as one of the supported rates:
   a. 64K (65536 samples/sec)
   b. 32K (32768 samples/sec)
   c. 16K (16384 samples/sec)
   d. 2K (2048 samples/sec)
3) dcuid: All realtime processes must have a unique (per IFO) dcuid number. This is used to identify a front end process to the data acquisition system for proper communications to the FrameBuilders.
4) gds_node_id: Global Diagnostic System (GDS) functions are built into every realtime application. To operate properly, each realtime application requires a unique GDS id number.
5) Compat_initial_ligo = 1: This must be set if the computer is to run as an integrated part of initial Ligo.

For items 3 and 4 above, the site system administrator should be contacted for proper id numbers if this code is to operate on an integrated CDS computer.

In addition to the above fields, there are additional optional entries. Each of these entries must be on its own line, followed by a carriage return:

- plant_name =
- shmem_daq
  - This results in a compiler flag such that the runtime code with use shared memory to communicate with the FrameBuilder software. This argument is only set if the software is to run on a standalone computer which will run the realtime code and the DAQ code.
- no_sync
  - Set if realtime code is not to be synchronized to the GPS 1PPS signal. This flag should be set if the realtime code is to be synchronized using an IRIG-B or if the system is to run asynchrously (only in a standalone system which has no synchronization signal available).
- no_daq
  - System is to run without data acquisition capabilities.
- dac_internal_clocking
  - The DAC modules will be clocked using internal clock signal instead of external clock signal from timing system. This is typically only used in testing.
- no_oversampling

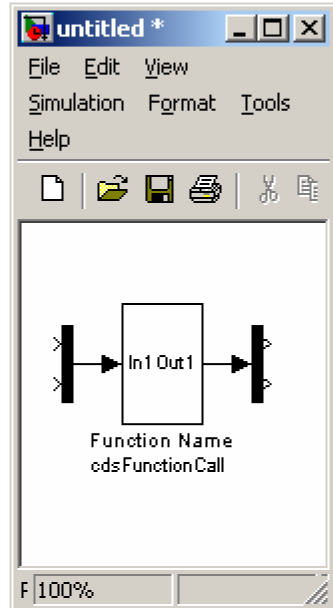
- The present default is to clock all ADC/DAC at 65536Hz, then do decimation / upsample filtering of I/O data to match the desired servo ‘rate’. With this flag set, the decimation filtering is not performed and it is expected that the timing clock will match the ‘rate’.

- no_dac_interpolation
  - As above, except this turns off the upsample filtering to 65536Hz
- specific_cpu=x
  - Without this definition, when a model is build into an application, it will run on cpu core 1. When it is desired to run multiple realtime applications, this parameter needs to be set to the cpu core to use (2-15).

7.1.1.3 Operation

This component is used solely to setup up appropriate compiler flags in the RCG. It is not linked as part of the realtime code.

7.1.2 cdsFunctionCall



7.1.2.1 Function

The purpose of this block is to allow users to link their own C code into the realtime application built by RCG. It is typically used when RCG does not support desired functions or the desired process is too complicated to be drawn in a model file.

7.1.2.2 Usage

Process variables are passed into and out of the user C function by connecting signals at the MUX inputs and DEMUX outputs. Any number of inputs or outputs may be connected by adjusting the MUX/DEMUX I/O sizes in Matlab.

The 'Function Name' must be changed to the name of the user supplied function. Keep in mind that, as with other parts, if this part is used within 'subsystem' parts, it will inherit the upper level names, the same as any other part used in the .mdl file. For example, if 'Function Name' is reentered as 'prc_inv' and this block is inside of subsystem block named LSC, the full name of the function called in realtime will be LSC_prc_inv.

The user defined C code function must be of the form:

```
Void Function_Name (double *in, int inSize, double *out, int outsize)
```

Where:

- Function_Name is the full name of the function to be called. In the example above, this would be LSC_prc_inv
- *in is a pointer to the input variables. Inputs are passed in the same order as they are connected to the input MUX.
- inSize indicates the number of parameters being passed to the function
- *out is a pointer to the output variables. Outputs are passed back to the main code in the same order as the DEMUX connections.
- outsize is the number of outputs allowed from the code module.

As a simple example of user code:

```
Void LSC_prc_inv(double *in, int inSize, double *out, int outsize)
{
        if(in[2] > in[0]) out[0] = in[1]*-1;
        else out[0] = in[1];
}
```

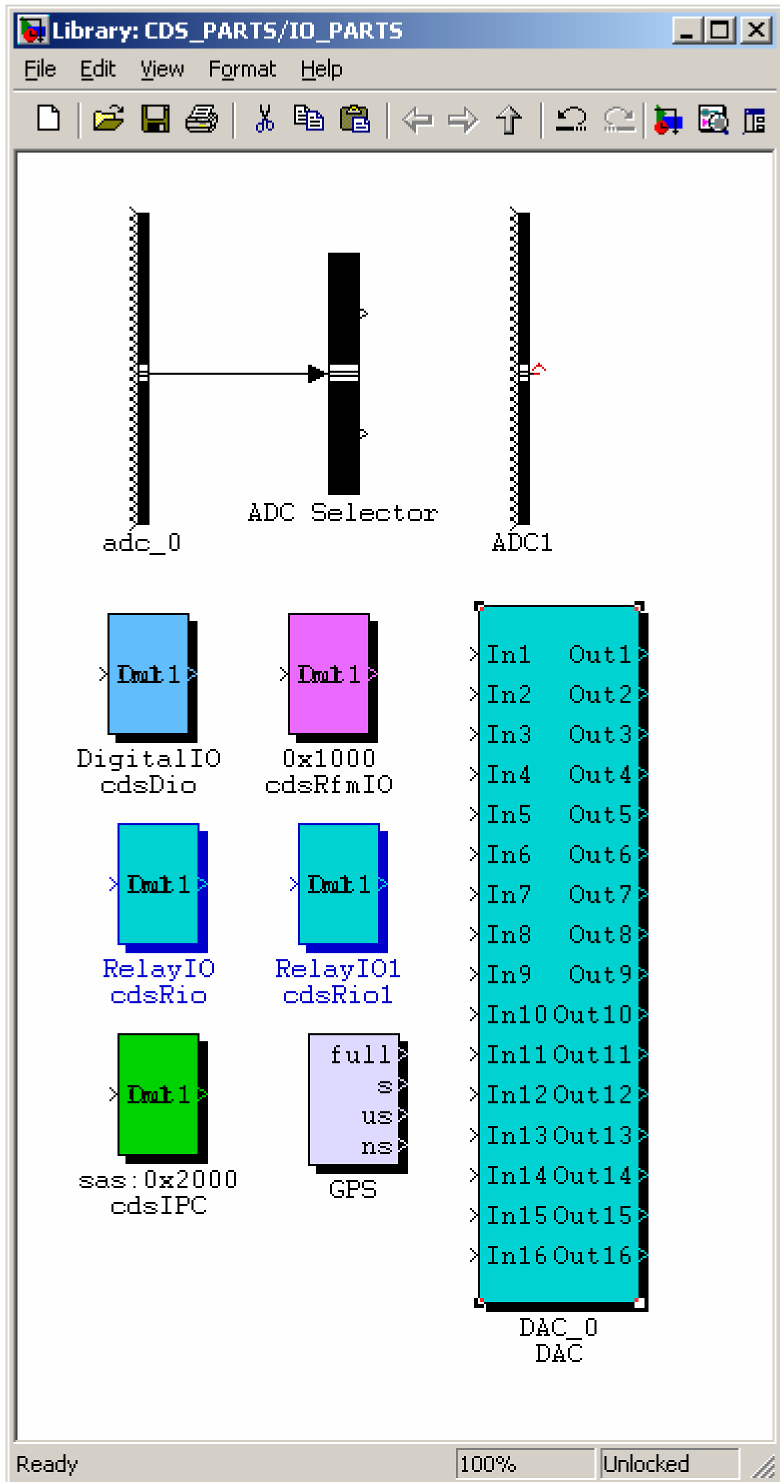
After the user code module is written, it must be placed in the appropriate directory and properly named to be compiled into the main realtime code. For example, if the above is part of a model named psl.mdl, then the code must be in the file 'LSC_pcr_inv.c' in the *advLigo/src/fe/psl* directory.

7.1.2.3 Operation

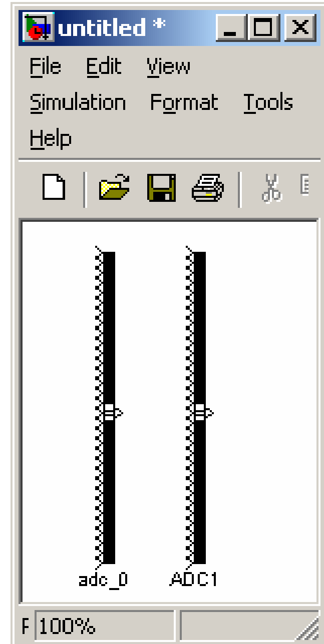
In runtime, the code operates as defined by the user provided C code.

7.2 I/O Parts

The I/O parts library contains the drivers for connecting I/O modules to the system.



7.2.1 ADC



7.2.1.1 Function

The purpose of this module is to define an ADC module. At present, only the General Standards 32 channel, 16 bit ADC is supported.

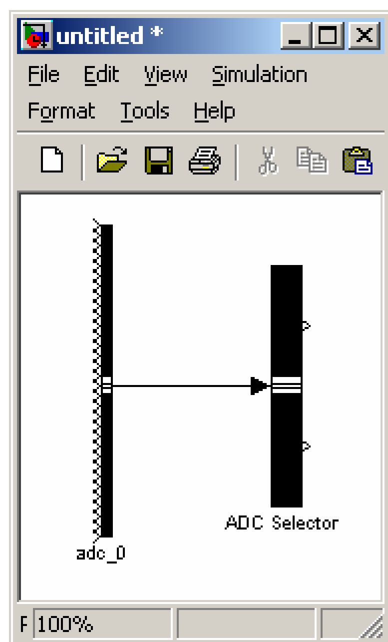
7.2.1.2 Usage

Each RCG model must include at least one (1) ADC block. The output of this block must be tied to one or more ADC Selector blocks to pick out and further connect individual ADC signal channels.

7.2.1.3 Operation

No software is directly produced for this part. Rather, it is used as an indicator of how many and of what type ADC module the realtime I/O software should expect during operation.

7.2.2 ADC Selector



7.2.2.1 Function

The function of the ADC Selector is to route selected channels from an ADC to other RCG model blocks (it is actually a Simulink Bus Selector part).

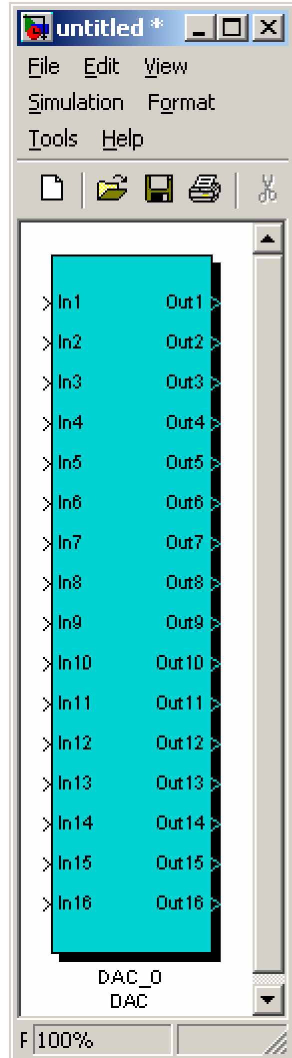
7.2.2.2 Usage

- Drag and drop the part into the model window.
- Connect the input to an ADC part.
- Double click on the ADC selector and select the desired signals from the Simulink window.
- Connect the outputs to other RCG parts.

7.2.2.3 Operation

No realtime code is directly generated to support this part. Rather, it is used by the RCG to produce appropriate signal links.

7.2.3 DAC



7.2.3.1 Function

The purpose of this block is to allow signal connections to be output to DAC output channels.

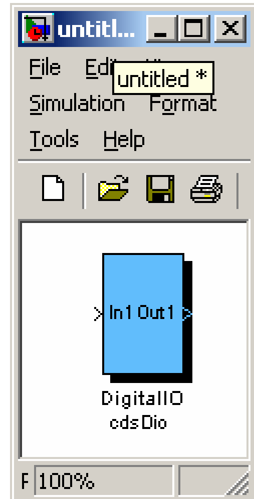
7.2.3.2 Usage

Desired output signals are connected to the 16 inputs of the DAC part. The Output connections are not used.

7.2.3.3 Operation

As with the ADC part, this block is only used by the realtime code to route signals to DAC modules.

7.2.4 cdsDio

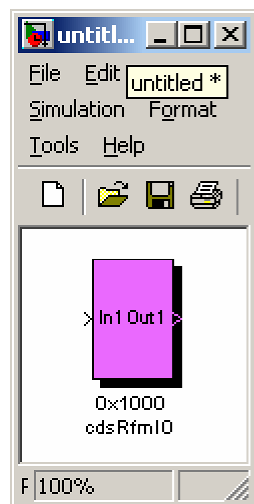


7.2.4.1 Function

7.2.4.2 Usage

7.2.4.3 Operation

7.2.5 cdsRfmIO



7.2.5.1 Function

The RCG supports communication between computers using the GEFanuc 5565 and 5979 reflected memory modules. This block allows single signal connection to/from these modules.

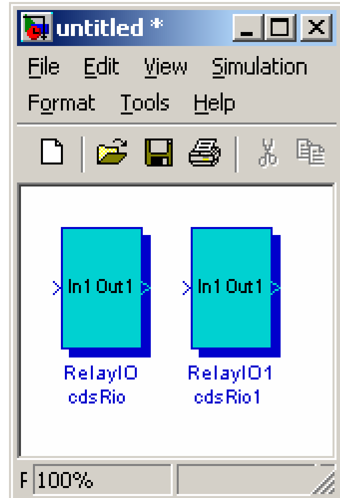
7.2.5.2 Usage

If a signal value is to be sent to the module, a signal needs to be connected to In1. If a signal is to be read from a reflected memory module, then a signal should be connected from the Out1 connection. The offset from the memory board base address is entered in the block label field. In example at right, the memory offset is set to 0x1000.

7.2.5.3 Operation

The realtime code provides a single write or read at the specified memory board offset in the form of a double precision float.

7.2.6 cdsRio

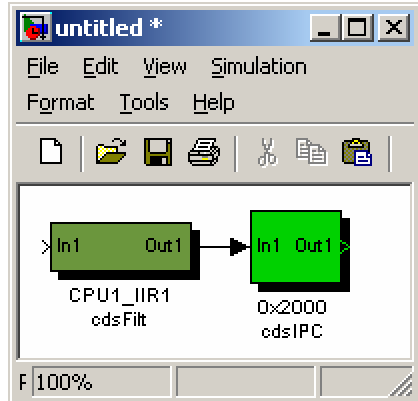


7.2.6.1 Function

7.2.6.2 Usage

7.2.6.3 Operation

7.2.7 cdsIPC

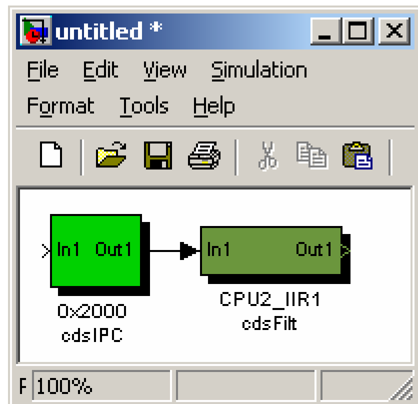


7.2.7.1 Function

The purpose of this module is to allow communications, via shared memory, between two or more realtime processes running in the same computer, but on separate CPU cores.

7.2.7.2 Usage

The user needs to change the label to a hex value, for example 0x2000. This part needs to exist in both the ‘sender’ model and the ‘receiver’ model, with the same address in both.



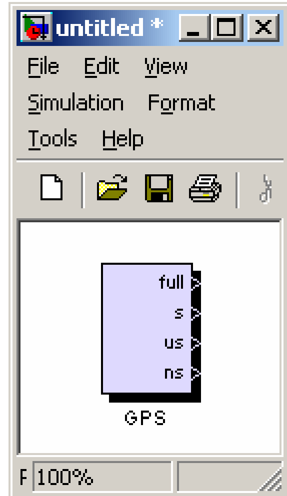
7.2.7.3 Operation

If there is a signal connected at In1, then this will result in the signal data being written to the address location as a single precision float. Conversely, if the Out1 is connected, data will be read in from the prescribed memory location as a float. Communications at runtime use the ‘ipc’ shared memory block.

Warning:
This communication is asynchronous ie the ‘reader’ will not wait for the ‘sender’. Therefore, it is up to the user to decide and take care of any synchronization needs.

Warning:
All computer cores on the same computer will use the same ipc shared memory block. Therefore care must be taken that models use unique addresses to communicate with each other.

7.2.8 GPS



7.2.8.1 Function

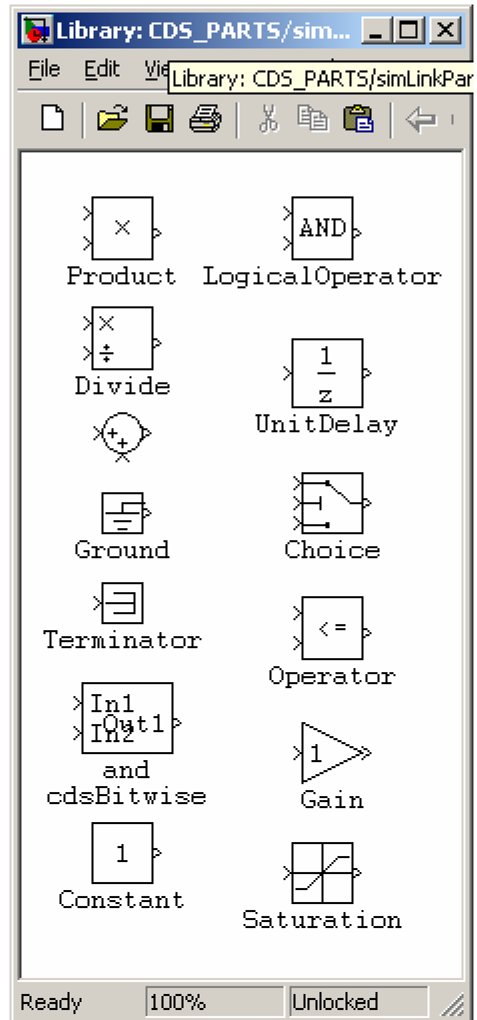
Return GPS time information from an IRIG-B interface module.

7.2.8.2 Usage

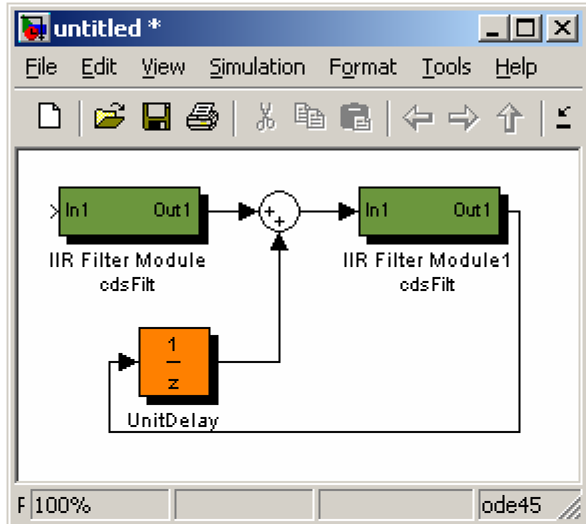
7.2.8.3 Operation

7.3 Simulink Parts

The RCG supports a number of standard SimuLink parts, as shown in the simLinkParts window (at right). In general, the code generated by the RCG behaves as it would in a SimuLink model. Special cases are described in the following subsections.



7.3.1 Unit Delay



7.3.1.1 Function

Typically, the RCG produces sequential code that starts with ADC inputs, performs the required calculations, and ends with the DAC outputs. However, there are cases where calculations performed within the code are to be fed back as inputs on the next code cycle. In these cases, the desired feedback signal must be run through a UnitDelay block to indicate to the RCG that this signal will be used on the next cycle

7.3.1.2 Usage

An example showing the use of the UnitDelay block is shown at the right. If the output of Module 1 were to be tied directly back to the summing junction at the input, it would produce an infinite loop in the code generator. By placing the UnitDelay in line, the output of Module 1 is sent back to its input on the next cycle of the software.

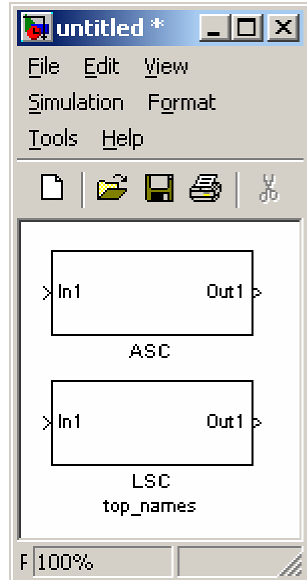
7.3.1.3 Operation

Introduces a one cycle delay between input and output.

7.3.1.4 Associated EPICS Records

None.

7.3.2 Subsystem Part



7.3.2.1 Function

This is a standard Matlab part for grouping individual parts into a subsystem.

7.3.2.2 Usage

Any number of parts within the application model may be grouped into a subsystem using the Matlab subsystem part. The RCG uses the assigned name as a prefix to all block names within the subsystem. This is done in two ways:

- In the top example at right, if it is at the top level of the model, all signal names for blocks within ASC would become SITE:ModelFileName-ASC_xxxx. So, if the model file name is omc.mdl and site defined as L1, names for parts within the ASC subsystem part would become L1:OMC-ASC_xxxxx.
- In the lower example (LSC), a tag has been added (using the Block Properties Window) “top_names”. This is a flag to the RCG to use the name of this subsystem to replace the model file name. Using the same example as above, all parts within this subsystem would be prefixed L1:LSC-xxxx.

The use of the ‘top_names’ subsystem part tags provides a couple of useful features:

1) A single model may contain parts with multiple SYS names in the Ligo naming convention. As seen in the example above, SYS is OMC (model name) for all ASC subsystem parts (L1:OMC-ASC_, but L1:LSC- for all LSC subsystem parts. In the same manner, ASC could also be defined ‘top_names’ and the results would be L1ASC-: and L1LSC-:.
2) Multiple models may contain the same SYS name. This allows models running on different processors to use the same SYS identifier in the signal names.

Warning: Since the name of all subsystems marked with the ‘top_names’ tag are used to replace the three character SYS part in the Ligo naming convention, this name must be 3 characters in length, no more, no less!

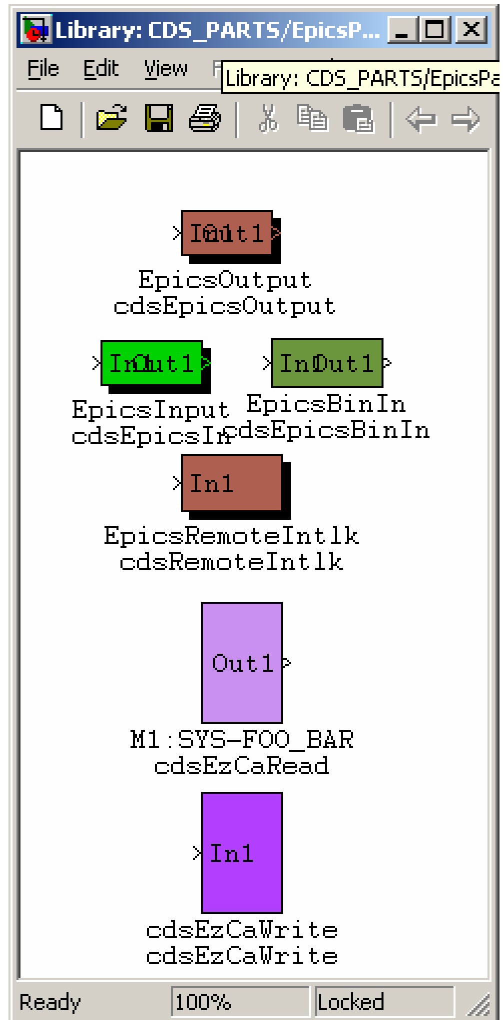
Warning: Subsystems with the ‘top_names’ tag may only appear at the highest level of the model ie they may not be nested within other subsystems.

7.3.2.3 Operation

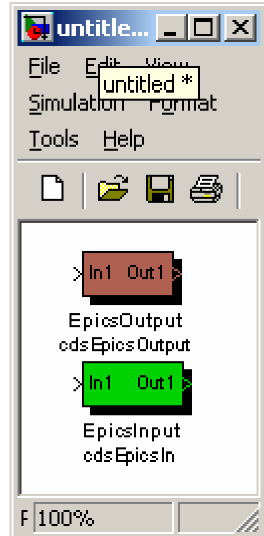
The subsystem part is only used by the RCG to produce appropriate signal names.

7.4 EPICS Parts

EPICS parts are used to input/output signals from/to the realtime application and EPICS. Some are used primarily to communicate with operator displays, while others are intended to allow multiple FE computers to communicate with each other using EPICS Channel Access (CA) via Ethernet connections.



7.4.1 cdsEpicsOutput / cdsEpicsIn



7.4.1.1 Function

The cdsEpicsOutput module is used to write data into an EPICS channel and the cdsEpicsIn module reads in data from an EPICS channel. *NOTE: The resulting EPICS channels are built on and communicate with EPICS on the local computer. To access EPICS channels on other computers, use the cdsEzcaRead/Write modules.*

7.4.1.2 Usage

For the EpicsOutput, connect the signal to be sent to EPICS via the In1 connection. The Out1 connection may be used to continue the signal into another RCG part.
For EpicsInput, use the Out1 connection to pick up the EPICS data. The In1 is not used.
For both, modify the name to the desired EPICS channel name.

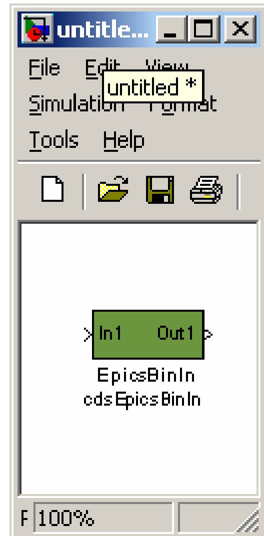
7.4.1.3 Operation

The RCG will produce local EPICS records with the assigned names and the realtime software will communicate data to/from the EPICS records via shared memory.

7.4.1.4 Associated EPICS Records

A single ai EPICS record will be produced using the assigned name.

7.4.2 cdsEpicsBinIn



7.4.2.1 Function

7.4.2.2 Usage

7.4.2.3 Operation

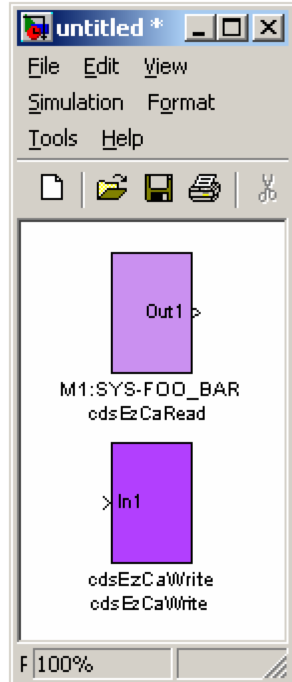
7.4.3 cdsRemoteIntlk

7.4.3.1 Function

7.4.3.2 Usage

7.4.3.3 Operation

7.4.4 cdsEzCaRead / cdsEzCaWrite



7.4.4.1 Function

These blocks are used to communicate data, via EPICS channel access, between realtime code running on separate computers.

7.4.4.2 Usage

Insert the block into the model and modify the name to be the exact name of the remote Epics channel to be accessed. This must be the full name, in Ligo standard format, including IFO:SYS-.

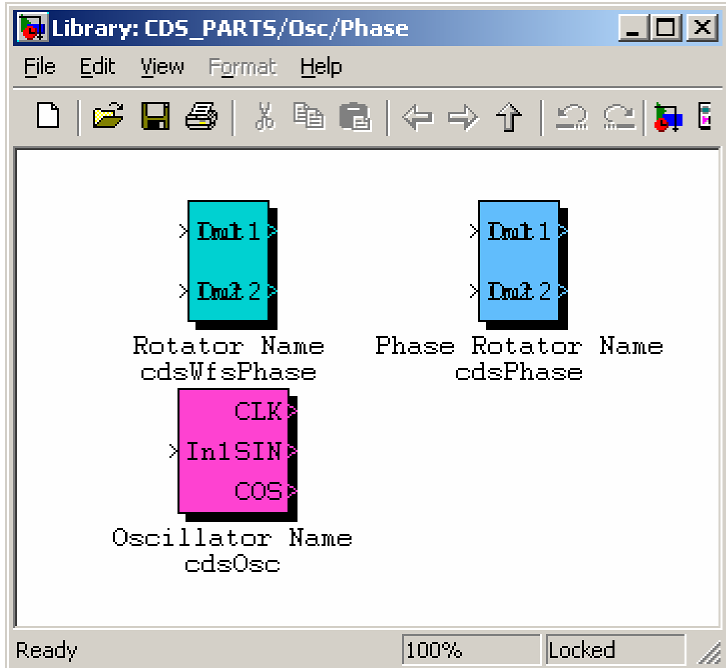
7.4.4.3 Operation

The EPICS sequencer which support the realtime code will have ezcaRead/ezcaWrite commands added to obtain/set the desired values via the Ethernet. Values are passed out of / into the realtime code via shared memory.

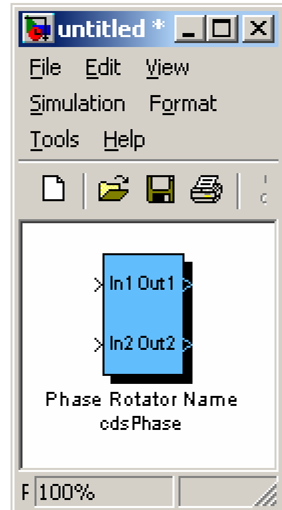
7.4.4.4 Associated EPICS Records

None.

7.5 Osc/Phase


Library: CDS_PARTS/Osc/Phase
File Edit View Format Help
Rotator Name
cdsWfsPhase
Phase Rotator Name
cdsPhase
CLK
In1 SIN
COS
Oscillator Name
cdsOsc
Ready
100%
Locked

7.5.1 cdsPhase



7.5.1.1 Function

This block replicates and I&Q phase rotator used in the Ligo LSC control software.

7.5.1.2 Usage

7.5.1.3 Operation

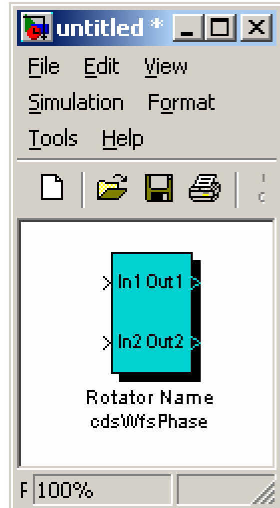
The EPICS code reads in the user variable and calculates a sin and cos for this entered value. These two values (sinPhase, cosPhase) are then passed to the realtime software, which perform the following calculation:

Out1 = In1 * cosPhase + In2 * sinPhase
Out2 = In2 * cosPhase – In1 * sinPhase

7.5.1.4 Associated EPICS Records

A single ai EPICS record is produced to support this module. Entries in this record are in units of radians.

7.5.2 cdsWfsPhase



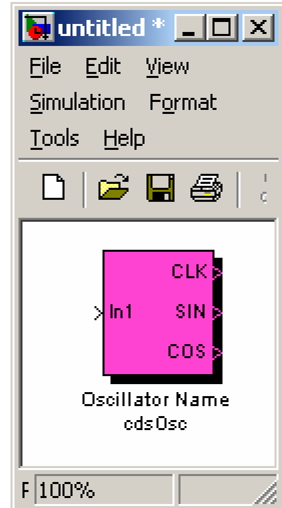
7.5.2.1 Function

7.5.2.2 Usage

7.5.2.3 Operation

7.5.2.4 Associated EPICS Records

7.5.3 cdsOsc



7.5.3.1 Function

This block is a software oscillator, developed to support dither locking where two signals with 90 phase rotation are required.

7.5.3.2 Usage

7.5.3.3 Operation

The three outputs are a sine wave at the user requested frequency. The CLK and SIN outputs are in phase with each other and the COS is 90 out of phase. The block internal sine wave varies in amplitude from -1 to 1. The three outputs are then multiplied by their individual gain settings to produce the CLK, SIN and COS outputs.

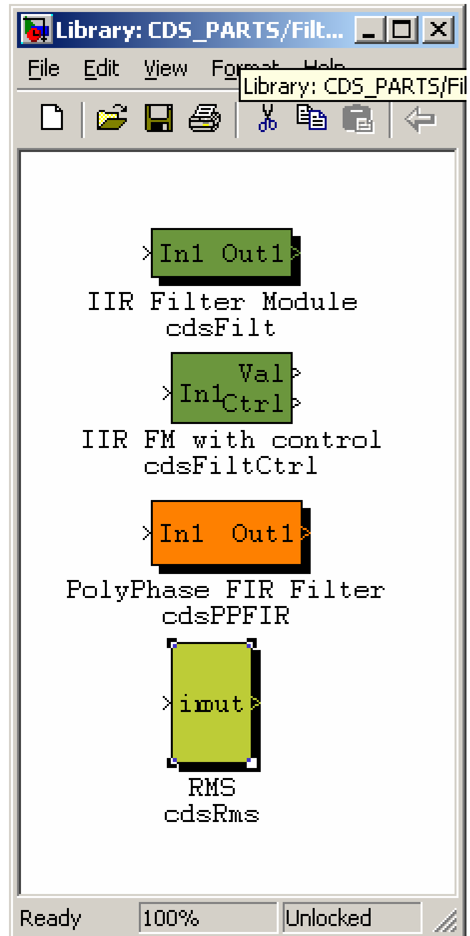
7.5.3.4 Associated EPICS Records

Four EPICS records are produced for user entries:
_FREQ: Desired frequency in Hz
_CLKGAIN:
_SINGAIN:
_COSGAIN:

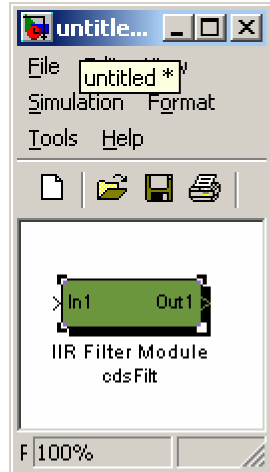
7.6 Filters

The key servo control functions provided by the RCG are in the form of digital filters, as shown in the Filter Parts section.

For most applications, the IIR Filter Module is used. The PolyPhase FIR Filter is designed only for the Ligo HEPI controls application and is not intended for general use.



7.6.1 CDS Standard IIR Filter Module



7.6.1.1 Function

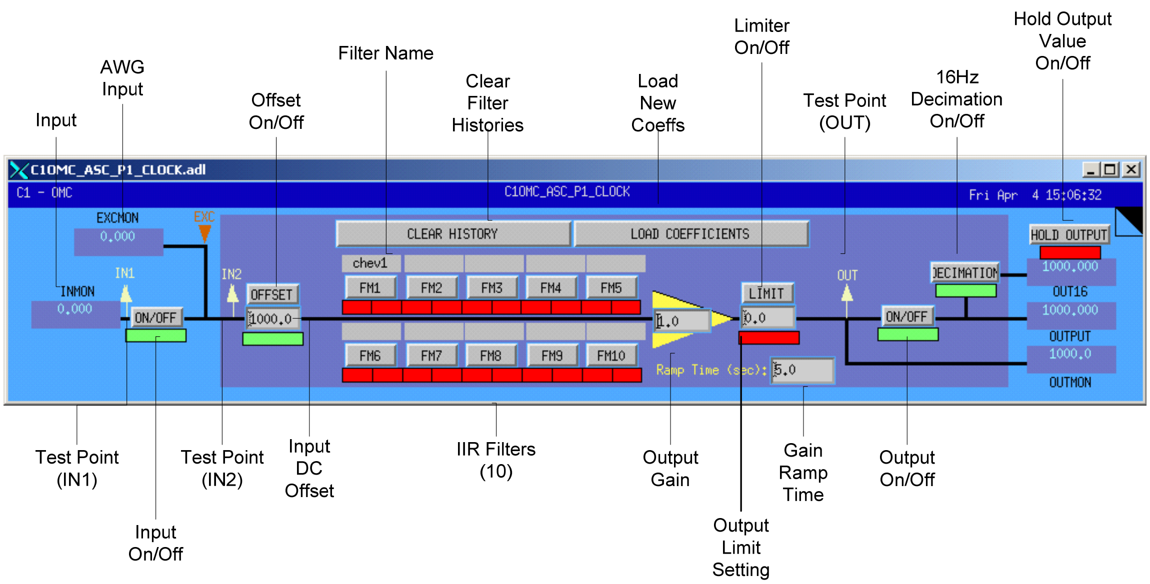
All CDS FE processors use digital Infinite Impulse Response (IIR) filters to perform a majority of their signal conditioning and control algorithm tasks. In order to facilitate their incorporation into FE software and to provide a standard set of DAQ and diagnostic capabilities, the Standard Filter Module (SFM) was developed.

7.6.1.2 Usage

Desired input signal is connected at In1 and output at Out1. 'IIR Filter Module' name tag is replaced with user name.

7.6.1.3 Operation

To help illustrate the operation of the Ligo CDS Standard Filter Module (SFM), an operator MEDM screen shot is shown below. Signal flow is from Input (left) to Output (right).

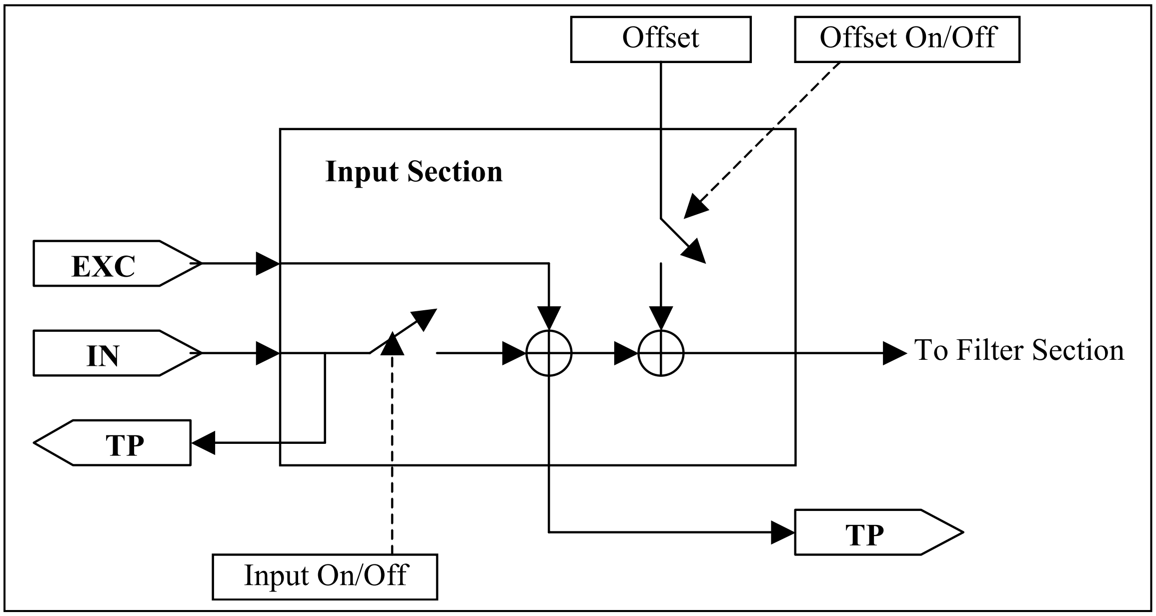


7.6.1.3.1 Input Section

The SFM input is as defined by the user in the Matlab Simulink model. At runtime, this signal is available to EPICS (_INMON) and is available to diagnostic tools as a test point (_IN1) at the sampling rate of the software. This signal may continue on or be set to zero at this point by use of the Input On/Off switch.

Each SFM also has an excitation signal input available from the Arbitrary Waveform Generator (AWG). This signal is available for EPICS (_EXCMON). The AWG signal is summed with the input signal, and available to diagnostic tools as a second test point (_IN2).

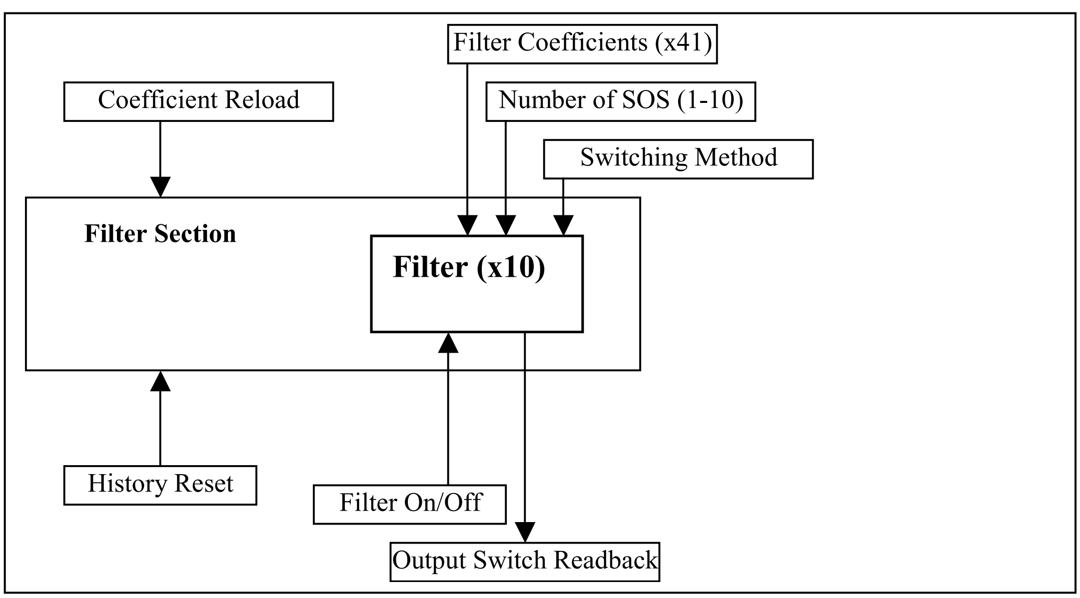
To this resulting signal, a DC offset may be added (Input DC Offset) and this offset may be turned on/off via the Offset on/off switch. The sum of the input, AWG and offset signal is then fed to the IIR filtering section.



7.6.1.3.2 Filtering Section

The filter section may have up to 10 IIR filters defined, with up to 10 Second Order Sections (SOS) each. The software allows for any/all of these filters to be redefined "on the fly" ie a FE process does not need to be rebooted, restarted or otherwise interrupted from its tasks during reconfiguration.

Each filter within a SFM may be individually turned on/off during operation. Various types of input/output switching may be defined for each individual filter.



The filter coefficients and switching properties are defined in a text file produced by the *foton* tool. Filter coefficient files used by the SFM must be located in the */cvs/cds/site/chans* directory. This file contains:

- The names of all SFM defined within a FE processor. Each SFM within a front end is given a unique name in the EPICS sequencer software used to download the SFM coefficients to the front end. These names must be provided in this file for use by *foton*. This is done by listing the SFM names after the keyword 'MODULES'. As an example, from the LSC FE file:
  - # MODULES DARM MICH PRC CARM MICH_CORR
  - # MODULES BS RM AS1_I
- A line (or lines) for each filter within an SFM, describing filter attributes and coefficients. These lines must contain the information listed in the following table, in the exact order given in the table.

Field	Description
SFM Name	The EPICS name of the filter module to which the remaining parameters are to apply.
Filter Number	The number of the filter (0-9) within the given SFM to which the remaining parameters are to apply.
Filter Switching	As previously mentioned, individual filters may have different switching capabilities set. This two digit number describes how the filter is to switch on/off. This number is calculated by input_switch_type x 10 + output switch type. The supported values for input switching are; • 0 – Input is always applied to filter. • 1 – Input switch will switch with output switch. When filter output switch goes to 'OFF', all filter history variables will be set to zero. Four types of output switching are supported. These are: • 0 – Immediate. The output will switch off as soon as commanded. • 1 – Ramp: The output will ramp up over the number of cycles defined by the RAMP field. • 2 – Input Crossing: The output will switch when the filter input and output are within a given value of each other. This value is contained in the RAMP field. • 3 – Zero Crossing: The output will switch when the filter input crosses zero.
Number of SOS	This field contains the number of Second Order Sections in this filter.
RAMP	The contents of this field are dependent on the Filter Switching type.
Timeout	For type 2 and 3 filter output switching (input and zero crossing), a timeout value must be provided (in FE cycles). If the output switching requirements are not met within this number of cycles, the output will switch anyway.
Filter Name	This name will be printed to the EPICS displays which have that filter. It is basically a comment field.
Filter Gain	Overall gain term of the filter.
Filter Coefficients	The coefficients which describe the filter design.

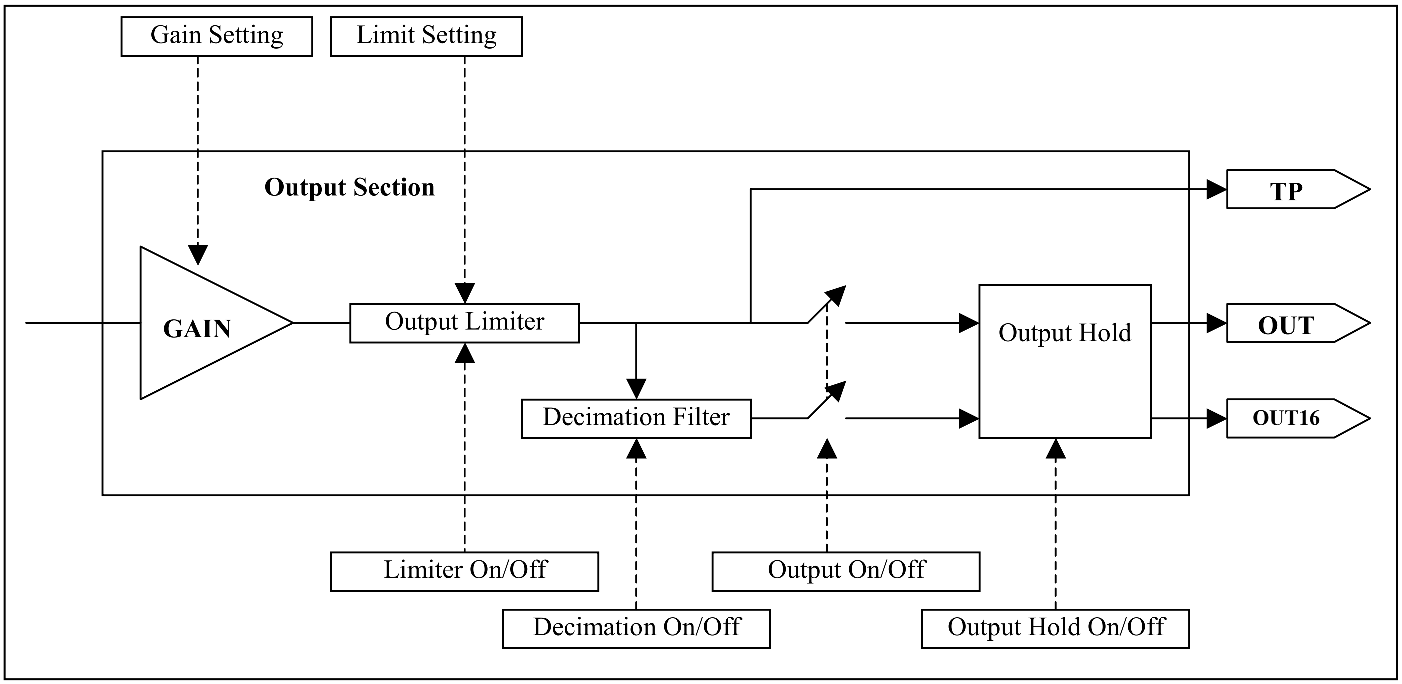
A skeleton coefficient file is produced the first time make-install is invoked after compiling a model file. Thereafter, whenever make-install is executed, the install process will make a backup of the present coefficient file, then patch the present file with any new filter modules or renaming of filter modules.

7.6.1.3.3 Output Section

The following figure shows the output section. The output section provides for:

- A variable gain to be applied to the filter section output. This gain may be ramped over time from one setting to another by setting the gain ramp time.
- This output to be limited to a selected value.

- A GDS TP. This TP is always on, regardless of whether the output is turned on or off.
- Ability to turn output on or off.
- A decimation filter to provide a 16Hz output (typically used by EPICS).
- A “hold” output feature. When enabled, the output of the SFM will be held to its present value.

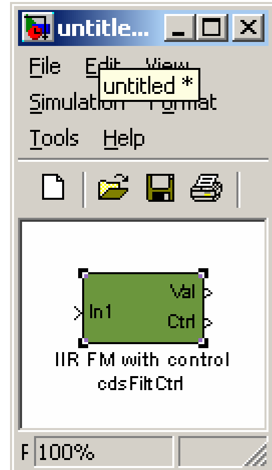


7.6.1.4 Associated EPICS Records

For each filter module, the following EPICS records are produced, with the filter name as the prefix:

_INMON = Filter module input value (RO)
_EXCMON = Filter module excitation signal input value (RO)
_OFFSET = User settable offset value (WR)
_GAIN = Filter module output gain (W/R)
_TRAMP = Gain ramping time, in seconds (W/R)
_LIMIT = User defined filter module output limit (W/R)
_OUTMON = Output testpoint value (RO)
_OUT16 = Filter module output, decimation filtered to 16Hz (RO)
_OUTPUT = Filter module output value (RO)
_SW1
_SW2
_RSET
_SW1R
_SW2R
_SW1S
_SW2S
_Name00 thru _Name09 = Individual filter names, as defined in the coefficient file (RO)

7.6.2 IIR Filter Module with Control



7.6.2.1 Function

This module is a standard filter module, with the addition that the SFM switch and filter status are output.

7.6.2.2 Usage

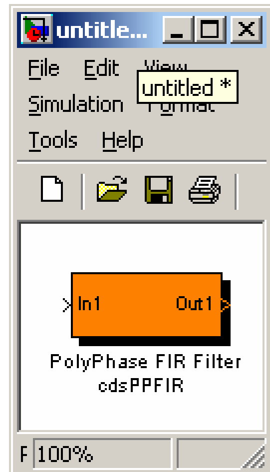
The additional control output is used to provide some downstream control or decision making based on the switch settings within the SFM. Typically this output is tied to a bitwise operator to select the desired bits, often to then go to binary output modules to switch relays based on filters being on/off.

7.6.2.3 Operation

In addition to the SFM operation, this block outputs the internal switch information in the form of a 32 bit integer. The bits of this integer are defined in the following table.

Bit	Name	Description
0	Coeff Reset	This is a momentary bit. When set, the EPICS CPU will read in new SFM coeffs from file and send this information on the FE via the RFM network. The FE SFM will read and load new filter coefficients from RFM.
1	Master Reset	Momentary, when set, SFM will reset all filter history buffers.
2	Input On/.Off	Enables/disables signal input to SFM.
3	Offset Switch	Enables/disable application of SFM input offset value.
Even bits 4-22	Filter Request	Set to one when an SFM filter is requested ON, or zero when SFM filter requested OFF.
Odd bits 5-23	Filter Status	Set to one by SFM when an SFM filter is ON, or zero when SFM filter is OFF.
24	Limiter Switch	Enables/disables application of SFM output limit value.
25	Decimation Switch	Enables/Disables application of decimation filter to SFM OUT16 calculation.
26	Output Switch	Enables/Disables SFM output (SFM OUT and OUT16 variables)
27	Hold Output	If (!bit 26 && bit27), SFM OUT will be held at last value.
28	Gain Ramp	If set, gain of filter module != requested gain. This bit is set when SFM gain is ramping to a new gain request.

7.6.3 PolyPhase FIR Filter



7.6.3.1 Function

This module allows the use of Polyphase FIR filters, typically used in seismic isolation system controls.

7.6.3.2 Usage

This part is placed into the model and functions exactly as the cdsFilter part. To load an FIR at runtime, a separate coefficient file must be provided for FIR filters (*/cvs/cds/site/chans/modelName.fir*).

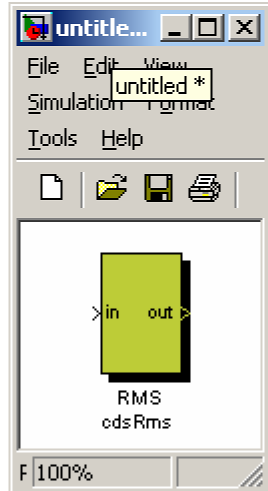
7.6.3.3 Operation

Use of this part simply sets a compiler flag to allow the use of FIR filters. In all other respects, it functions in the same way as the cdsFilter part described previously. In fact, this part allows a mix of IIR and FIR filters to be assigned to the 10 available digital filters within the module. The difference between IIR and FIR is determined by the runtime software by the number of coefficients loaded (>10 SOS = FIR).

7.6.3.4 Associated EPICS Records

Same as cdsFilt part.

7.6.4 RMS Filter



7.6.4.1 Function

This block computes the RMS value of the input signal.

7.6.4.2 Usage

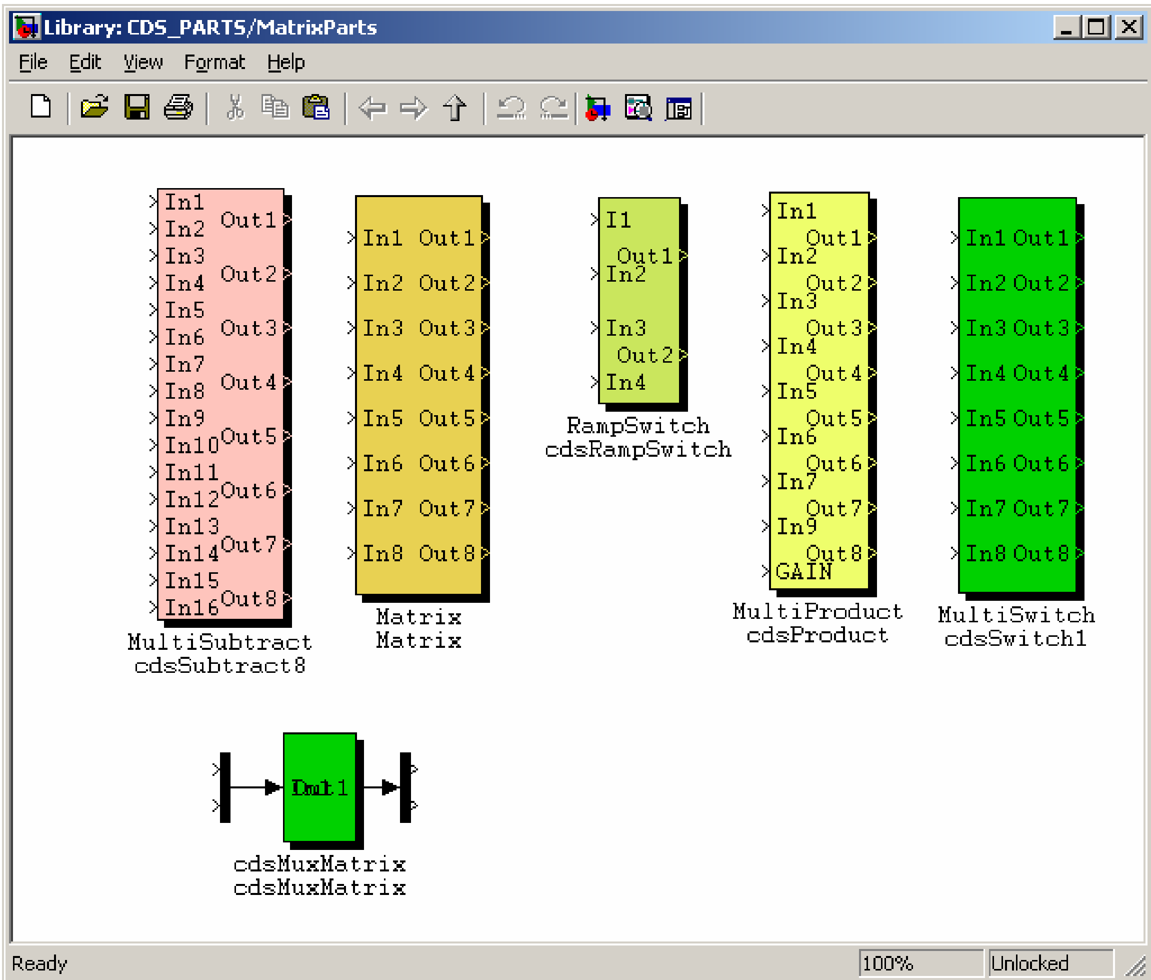
7.6.4.3 Operation

7.6.4.4 Associated EPICS Records

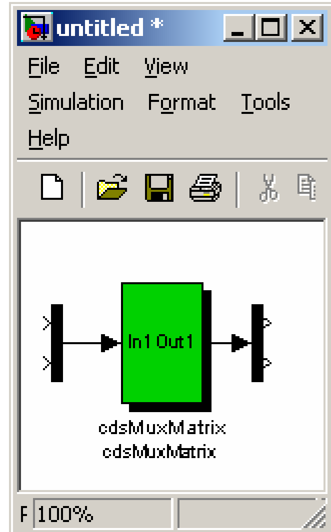
Provides a single EPICS output, using the name assigned to this block.

7.7 Matrix Parts

Matrix parts are those which perform calculations based on array data. The most commonly used is the cdsMuxMatrix part.



7.7.1 cdsMuxMatrix



7.7.1.1 Function

The primary function of this block is to produce outputs signals based on the scaling and addition of various input signals.

7.7.1.2 Usage

Inputs are connected via the MUX part and outputs are connected via the DEMUX part. The number of connections available at the input/output may be modified to any size by double clicking on the MUX/DEMUX and modifying the number of connections field in the pop up window.

7.7.1.3 Operation

Basic code function is:
Output[0] =
Input[0] * Matrix_11 + Input[1] * Matrix_12 + Input[n] * Matrix_1n,
where Matrix_xx is an EPICS entry field.

7.7.1.4 Associated EPICS Records

The RCG will produce an A x B matrix of EPICS records for use as input variables, where A is the number of inputs and B is the number of outputs. The EPICS record names will be in the form of PARTNAME_AB, starting at PARTNAME_11.

7.7.2 MultiSubtract

7.7.2.1 Function

This module is a group of subtractions, packaged into a single part.

7.7.2.2 Usage

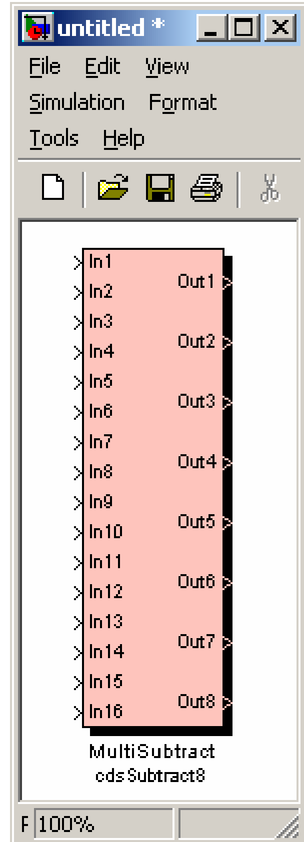
Connect all input and output connectors.

7.7.2.3 Operation

This module subtracts pairs of inputs (16) and produces 8 outputs eg Out1 = In2 – In1, Out2 = In4 – In3, etc.

7.7.2.4 Associated EPICS Records

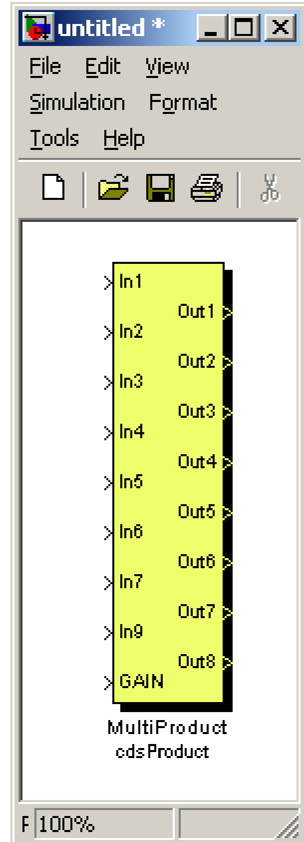
None.



7.7.3 MultiProduct

7.7.3.1 Function

The purpose of this block is to multiply up to eight inputs by a single input gain setting. Whenever a gain setting is changed, this block will ramp the gain from the present to new setting over the user defined time interval.



7.7.3.2 Usage

The 8 inputs and outputs are connected, either to other signals or terminators. If the GAIN input is connected, then the multiplier will be derived from the connected component. If the GAIN is not connected, then the multiplier will come from EPICS.

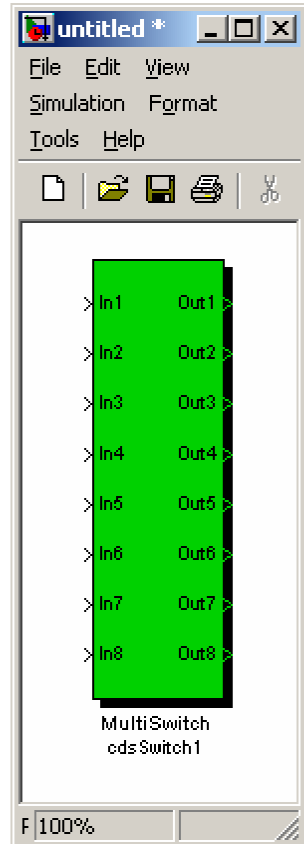
7.7.3.3 Operation

The code for this block will multiply all inputs by the gain setting and produce the results at the corresponding outputs. If the gain is changed, the code will ramp the gain value over the requested ramp time.

7.7.3.4 Associated EPICS Records

_GAIN : Gain to be applied to all channels
_TRAMP: Time (seconds) over which to ramp any gain changes.

7.7.4 MultiSwitch



7.7.4.1 Function

This block allows simultaneous on/off switching of up to 8 signals via a single EPICS input record.

7.7.4.2 Usage

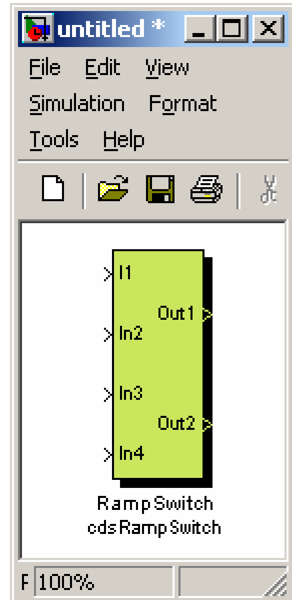
7.7.4.3 Operation

When the associated EPICS record is set to '1', In1 thru In8 are passed straight through to Out1 thru Out8. If the EPICS record is set to zero, Out1 through Out8 become zero.

7.7.4.4 Associated EPICS Records

The RCG produces a single EPICS ai record with the name given to this part by the user.

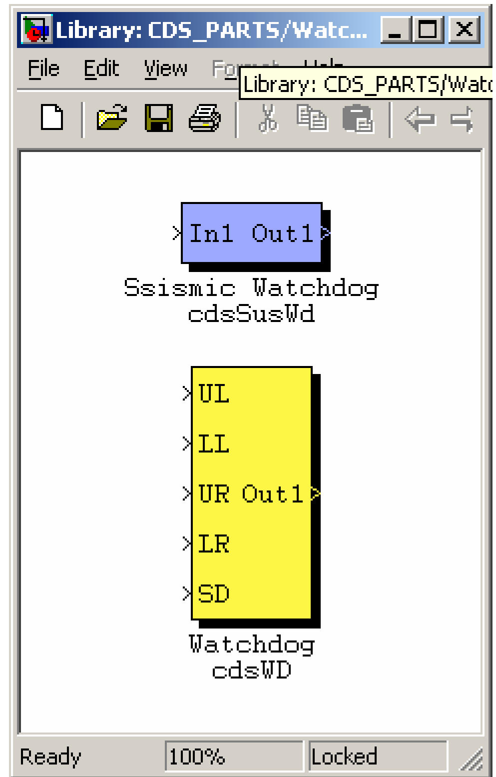
7.7.5 RampSwitch



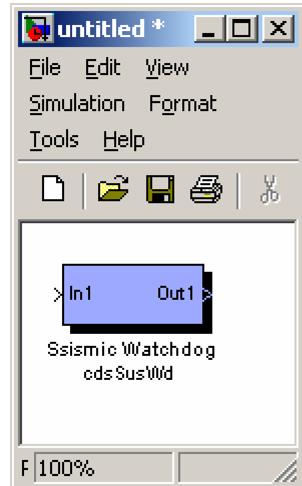
7.8 WatchDogs

Watchdogs are used to monitor their input signals and produce an error signal at their output to automatically trigger some fault handling code/modules. The modules to date were designed to implement similar tasks in initial LIGO controls.

NOTE: There is a third watchdog type (not shown), which was specifically implemented to replicate the watchdogs used in present LIGO HEPI systems. It is intended that it will be redesigned and added to the Watchdog parts library in a future release.



7.8.1 cdsSusWd



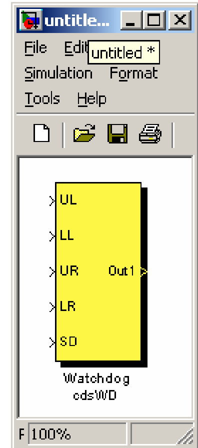
7.8.1.1 Function

This function was developed with the sole purpose of connecting a suspension trip signal to the HEPI system in the early prototyping stages at LASTI. This block should not be used in any new designs.

7.8.1.1.1 Usage

7.8.1.1.2 Operation

7.8.2 cdsWD

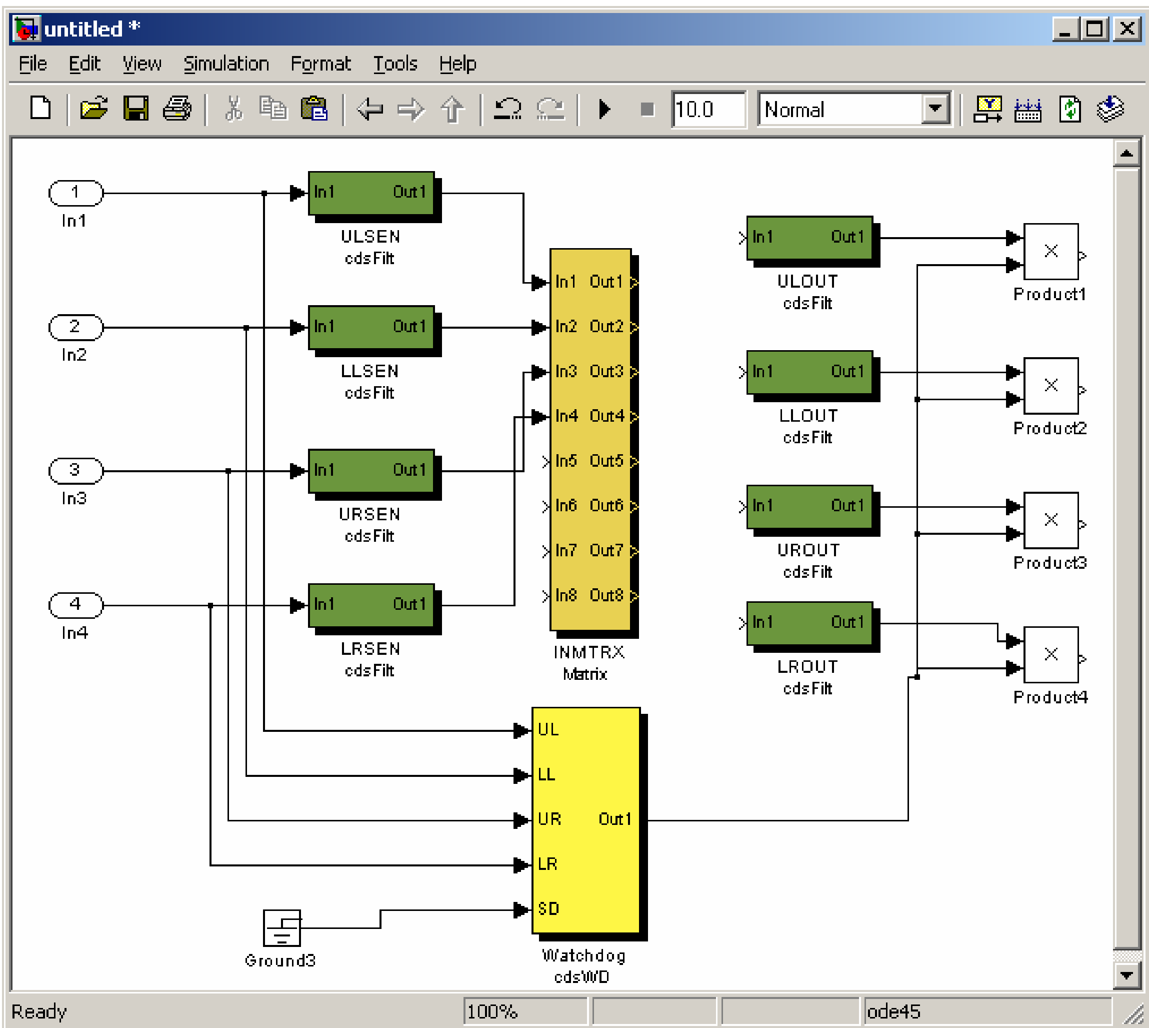


7.8.2.1 Function

This block was designed to implement the suspension watchdog function found in initial LIGO.

7.8.2.2 Usage

Typically, the raw suspension OSEM signals are input at the left of the block. The output is then connected to a product block, with the second connection of the product being the signal path which is to be turned off if the watchdog trips. An (incomplete) example is shown in the following figure.



7.8.2.3 Operation

The runtime software for this module continuously calculates an RMS and variance for each input signal. If all variances are within the tolerances, the output is 1. If the variance for any input signal exceeds the RMS value beyond the operator setpoints, the output becomes a value of 0, and remains 0 until reset by the operator.

7.8.2.4 Associated EPICS Records

To support this module, the following EPICS records are produced for operator interaction. Signal names shown in the table are based on the part being named 'WD' in the user model.:

Name	Type	Purpose
WD	Momentary ai	Used to turn the module on/off. If 'on', watchdog is operational. If 'off', the output of the watchdog code goes to 0. This is also used to 'reset' the watchdog after variances are back in tolerance.
WD_STAT	Ai	Provides watchdog status information
WD_MAX	Ai	Trip setpoint. If variance on any input exceeds its RMS value by greater than this setting, the WD will trip.
WD_VAR_1 Thru WD_VAR_5	Ai	This records provide readbacks on the present variance of all five input signals.