LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

*LIGO Laboratory / LIGO Scientific Collaboration*

LIGO-T1200291-v1 *LIGO* June 11, 2012

# Real-time Code Generator (RCG) Software Component Overview

R. Bork

Distribution of this document:
LIGO Scientific Collaboration

This is an internal working note
of the LIGO Laboratory.

**California Institute of Technology**
**LIGO Project – MS 18-34**
**1200 E. California Blvd.**
**Pasadena, CA 91125**
Phone (626) 395-2129
Fax (626) 304-9834
E-mail: info@ligo.caltech.edu

**Massachusetts Institute of Technology**
**LIGO Project – NW22-295**
**185 Albany St**
**Cambridge, MA 02139**
Phone (617) 253-4824
Fax (617) 253-7014
E-mail: info@ligo.mit.edu

**LIGO Hanford Observatory**
**P.O. Box 1970**
**Mail Stop S9-02**
**Richland WA 99352**
Phone 509-372-8106
Fax 509-372-8137

**LIGO Livingston Observatory**
**P.O. Box 940**
**Livingston, LA 70754**
Phone 225-686-3100
Fax 225-686-7189

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

# 1 Introduction

The purpose of this document is to provide a description of the Real-time Code Generator (RCG) software. It is intended to provide sufficient detail for experienced software developers to modify, enhance or maintain the core RCG software. This includes the RCG compiler functions and the various RCG perl and C code components necessary to produce and compile the code that runs on the aLIGO real-time control computers. This document does not cover the aLIGO data concentrator, framebuilder and network data server software.

# 2 References

1) aLIGO CDS Design Overview LIGO-T0900612

2) aLIGO CDS Real-time Sequencer Software LIGO-T0900607-v5.pdf

3) CDS Real-time Data Acquisition Software LIGO-T0900638

4) aLIGO CDS Inter-Process Communications Software LIGO-T1000587

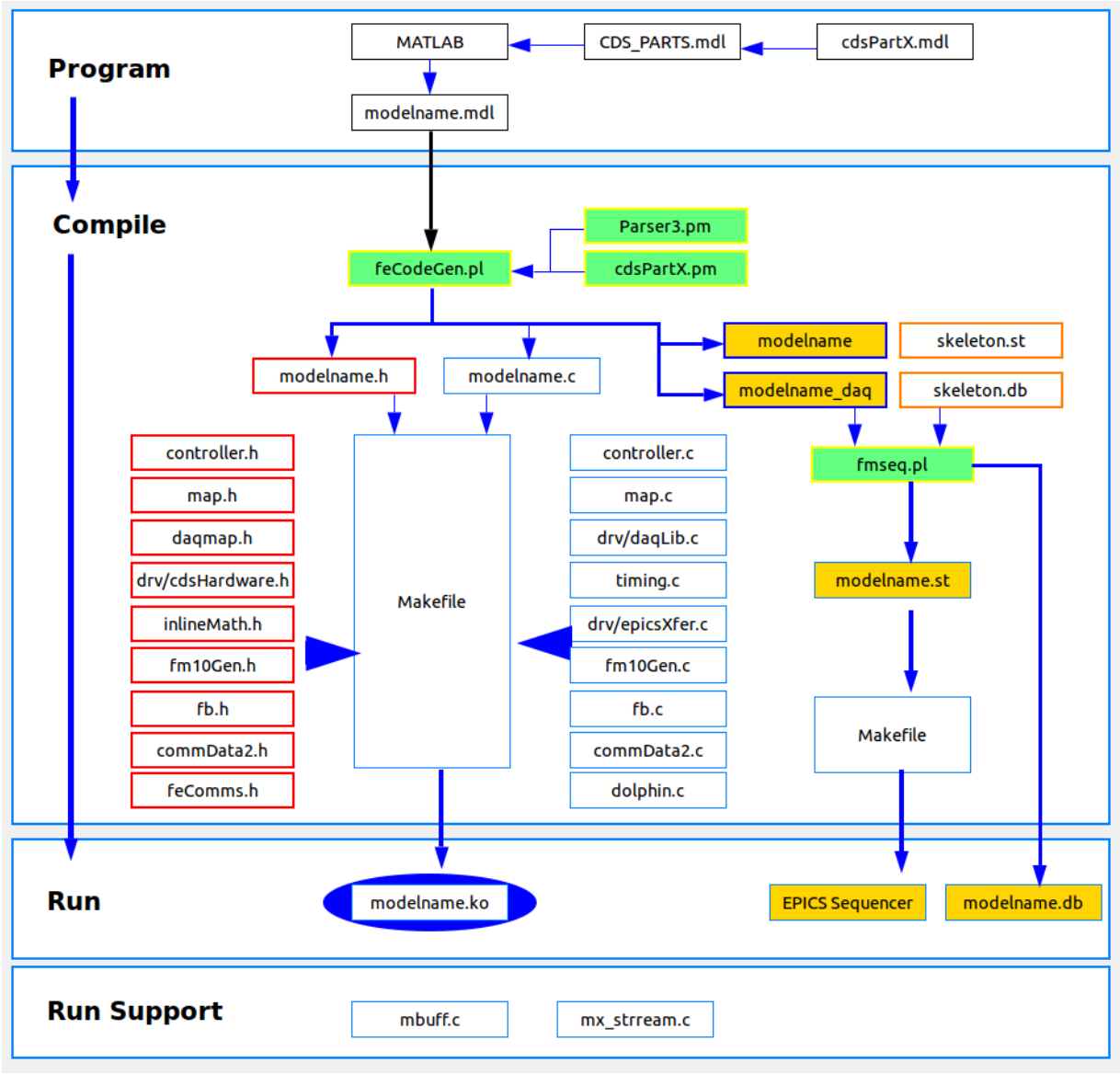
5) CDS Standard IIR Filter Software LIGO-T0900606

# 3 Overview

The RCG consists of three primary components:

1) Matlab Simulink, which provides the Graphical User Interface (GUI) and text files which represent the code to be generated.
2) Perl parser scripts, which read the Matlab .mdl files and produce:
    a. C code to be compiled for real-time execution.
    b. EPICS State Notation Language (SNL) code, which acts as the interface between the real-time software EPICS database records.
    c. EPICS database records, which act as the interface to the controls network via EPICS Channel Access (CA).
    d. Various header and Make files to support the compilation of code via the Linux GNU compilers.
3) Core header and C code files that are compiled as part of every application built by the RCG.

The following diagram depicts the various code modules that make up the RCG, from programming through to executable code. All RCG software may be found in the CDS SVN code repository under the advLigoRTS area. Note, that while this diagram shows the key components, it is not totally inclusive of all source and header files.




# 4 Matlab Simulink

Matlab is used as the GUI to allow the "application programmer" to define a control process via a graphical diagram. The output of Matlab is a text file, which describes the various parts and their connections. This text (.mdl) file is then used by the RCG in the compilation process.

All parts that RCG supports are contained in the CDS_PARTS.mdl library file in the $ROOT/src/epics/simLink directory, where $ROOT is the top level directory of the particular advLigoRTS checkout. Graphical representations of the individual parts in this library are contained in the simLink/lib directory. Note that the CDS_PARTS are only graphical parts which contain information necessary for the RCG parsers in compiling code. They are not true Matlab parts, as they have no Matlab code associated with them.

# 5 RCG Perl Scripts

Once the Matlab model has been built and .mdl file saved, the RCG build process is executed using a standard "make" command. Prior to this process, a "build" directory needs to be established and configured. This is done by:

1) Creating a directory, typically in the /opt/rtcds/<site>/<ifo> directory. This area is designated $BUILD for the remainder of this document.
2) Moving into that directory and, from the command line, executing $ROOT/configure.

The latter will create a Makefile and produce subdirectories needed in the make process. Once this is done, executing 'make modelname' will invoke the various RCG Perl scripts and compilers to produce runtime executable code and various supporting configuration files. Invoking 'make install-modelname will then install all of the generated objects to appropriate target directories to load the software onto the real-time control computers.

## 5.1 Model Parsing Code

To develop executable software from the Matlab file, the RCG basically has to do two things:

1) Find all of the parts defined in the file, for which code will be substituted.
2) Find all the links between parts to determine the processing sequence.

The Perl scripts and modules that perform these functions are located in the $ROOT/src/epics/util (Perl scripts) and $ROOT/src/epics/util/lib directories (Perl modules). The first script invoked by the make command is feCodeGen.pl. This script essentially has three parts:

1) Find all the parts and links between parts. This is done with the help of the lib/Parser3.pm module. At the end of this step, a diags.txt file is produced in the build directory under src/epics/util. This file contains a list of all parts found and all inputs/outputs to/from each part. This file is not used in the remaining build process, but may be a useful diagnostic tool for code developers if the build process does not complete successfully.
2) The script produces an ordered execution list for all of the parts. The basic concept here is that a part gets added to the execution sequence list when, and only when, all of the parts that provide input for this part are already on the execution list. The first items placed on the execution list are those parts which either do not require an input from another part, or have only one input connection and that input connection is from an ADC module. Once these items are placed in the list, the script continues to loop over the remaining parts until all part input requirements have been satisfied or the code finds that there are connection errors in the Matlab model.
3) Writing of source code and header files. The script now runs through the execution parts list, essentially substituting source code for each part. Some of this substitution is handled directly by feCodeGen.pl, and others are handled by Perl modules located in the epics/util/lib directory. As a general rule, parts that are supported by Matlab itself, as depicted in the CDS_PARTS.mdl, Simulink Parts subsystem block, are handled by the feCodeGen.pl script itself, with all other parts using the supporting Perl modules.

All of the cdsPart.pm modules contain the same sub components, such that they can be easily developed and "plugged into" the RCG package. These sub components are:

1) sub partType: Returns the part type information to the RCG for use in later code generation calls. This part type must be unique.
2) sub printHeaderStruct: Includes EPICS definitions to be installed in the modelname.h file, which is later used to develop the EPICS to real-time interface definition.
3) sub printEpics: List of EPICS channels associated with this application, in the form need by fmseq.pl (described in next section).
4) sub printFrontEndVars: List of variables to be used by this part that need to be defined in the real-time C source.
5) sub frontEndInitCode: Any code required during the initialization of the real-time code.
6) sub fromExp: Code to be inserted when another part requires data from this part.
7) sub frontEndCode: Source that defines the processing to be performed when this part is executed.

The principal products of the feCodeGen.pl script are:

1) Real-time C source and Makefile. These are placed in the build directory under $BUILD/src/fe/modelname directory.
2) A modelname.h header file in the $BUILD/src/include directory. This header file primarily defines the structures that allow the real-time code to communicate with the EPICS SNL code, which will in turn communicate with the EPICS database.
3) A text file, using the model name, in $BUILD/src/epics/fmseq directory. This file lists all filter modules and EPICS inputs/outputs. This file will be used in the generation of the EPICS State Notation Language (SNL) code and EPICS database records by another script, fmseq.pl, described below.

## 5.2 EPICS Code and Database Generation

Another Perl script, $ROOT/src/epics/util/fmseq.pl, is invoked to generate the EPICS interface side of the code. This script reads in the following files to obtain system information and produce the EPICS products:

1) The model name text file produced by feCodeGen.pl in the $BUILD/src/epics/fmseq directory.
2) The skeleton.st and skeleton.db files located in the $ROOT/src/epics/util directory. The skeleton.st file is the template for generation of EPICS SNL code, which fmseq will fill in with model specific items. The skeleton.db file is a template of all EPICS records to be produced for filter modules.

From the text file in epics/fmseq, the fmseq script will develop the following products in the $BUILD/build/<modelname>epics directory.

1) EPICS SNL code (modelname.st). This code will be used to move data between the EPICS database records and the real-time code via shared memory on the real-time computer.
2) EPICS database (modelname.db file). The database records will allow data to be communicated on the control system networks to various EPICS compatible software tools, such as operator display graphical user interfaces.
3) A Data Acquisition (DAQ) channel list file (modelname.ini). This file is read by both the real-time code and DAQ software to acquire data at runtime. As of RCG V2.5,

channels to be acquired and saved to disk are listed within the application models via a new DAQ part.

4) A Global Diagnostics channel list (modelname.par). This file is used by the real-time code and DAQ system to provide a list of all data channels available "on demand" as testpoints or excitation entry points.

As the make process continues, the SNL code will be run through the SNL precompiler and then GNU compiler to produce exectuables in the $BUILD/target/<modelname>epics directory. The final startup scripts and databases are also generated here, for later movement to the runtime target directory when make install is invoked.

# 6 Real-time Core C code modules

The RCG Perl scripts produce an application specific C code file for each control model. This code will be inline compiled with various RCG C code sources to produce a final kernel module object. The key C code modules provided as part of the RCG are shown in Figure 1 above and described further in the following subsections.

## 6.1 controller.c

This code, referred to as the Real-time Sequencer, is described in detail in Reference 2. Once the real-time code gets installed as a kernel object, it divorces itself from the Linux OS and its scheduler and interrupt generator. This code now takes over those functions for this user application.

Along with timing and scheduling, this code module takes care of all code initialization and I/O transactions for the user application, either:

1) Directly. This code performs all I/O transactions for ADC, DAC and binary I/O modules. Note that this is dependent on the compile options:
    a. adc_master: On each real-time computer, one application, referred to as the I/O Processor (IOP) performs all I/O initialization and mapping, and directly communicates data from/to ADC and DAC modules. It passes this data, along with addresses for binary I/O modules and real-time networks, via shared memory to the remaining real-time applications on this computer. It is intended to run at the full 65536 clock rate of ADC/DAC modules, and makes use of the ADC data as a timing trigger for the next code cycle.
    b. adc_slave: Actual control applications are built with this compile option. In this case, the controller code communicates and synchronizes with the IOP. It handles binary I/O module transactions directly with the PCIe devices only.
2) Indirectly. Address locations are passed to commData2, described later, for real-time communications between real-time processes.

## 6.2 map.c

The RCG runtime code accesses all I/O devices directly through bus addresses ie does not typically use vendor provided Linux drivers. This is done for both performance reasons and, since the RCG code becomes a kernel object, it cannot access the application developer's routine calls typically provided by vendor drivers (intended to be accessed from user space). There are a few cases, such

as with the Dolphin real-time network, where the vendor has provided calls at the kernel level and are used by the RCG runtime code.

All of the routines used in mapping of I/O devices, along with calls to read/write RCG supported PCIe modules, are contained in the map.c source file. On initialization, the controller.c code (compiled as an IOP) calls the mapPciModules routine in the map.c file. This code contains the methods necessary to search for and find all devices on the PCIe bus and make appropriate mappings to CPU bus addresses, such that the remainder of the PCIe read/write routines can address the PCIe modules directly.

A list of all PCIe devices, addressing, and register setup definitions are provided in the $ROOT/src/include/drv/cdsHardware.h file.

## 6.3 daqLib.c

The file contains all of the routines used in acquiring data from the real-time application, as well as extracting/injecting GDS testpoint data and excitation signals. This code is called once per cycle by the controller.c software. This code is described in further detail in Reference 3.

## 6.4 commData2.c

The RCG provides software for synchronous communication of data between real-time processes, either on the same computer or, via real-time networks, to processes on other computers. The source code routines for this are located in the commData2.c file. Calls to these routines are placed directly into the user application produced by the RCG, with a call to read all inputs at the beginning and a call to write all variables at the end. A detailed description of this software is provided in Reference 4.

## 6.5 fm10Gen.c

Primarily, control is provided by the definition IIR filters to provide the control transfer functions. Code to support IIR filtering is included in this file, with a detailed description provided in Reference 5.

## 6.6 timing.c

The real-time code performs various timing measurements for diagnostic purposes. Routines to read time data from IRIG-B time code receiver modules and perform timing calculations on duotone signals are included in this source file.

## 6.7 epicsXfer.c

The bulk of the data that needs to be exchanged at runtime between the real-time code and EPICS are filter module variables. With applications often containing hundreds of filter modules and about a dozen variables per module to be exchanged, this can lead to heavy loading if all done at once. Therefore, the transfer is spread across many code cycles via the routine in this source file to provide more consistent timing of the real-time code.

### 6.8 inlineMath.h

Though not a C source file, this file is of particular note. Since the real-time application is a kernel object, it cannot be compiled using the standard math.h file for performing standard math functions. Therefore, the real-time code uses math assembly code instructions, as defined in this file.

## 7 Runtime Support Code

In addition to source that gets compiled into the applications as they are build, there also exists various source files to provide runtime capabilities.

### 7.1 Linux Kernel Patch

The RCG produced real-time code must be precisely (within a few μseconds) and consistently triggered at a known time (ADC data ready). This type of precision is not available from a GPL Linux OS, even with the present real-time extensions for Linux. Therefore, a Linux kernel patch was developed which:

1) Releases the application target CPU core from the Linux OS. Basically, it reports to Linux that the particular core is powering down and no longer available for scheduling tasks or handling interrupts.
2) It loads and executes the RCG real-time application on the Linux released CPU core.

These steps ensure that the desired CPU core is under the sole ownership of the real-time application and the Linux OS will not interfere. At this point, scheduling is done exclusively by ADC data arrival, which is in turn triggered precisely by the aLIGO Timing Distribution System (TDS).

Linux patch files for various Linux release versions are maintained in the $ROOT/patches directory.

### 7.2 mbuf

Communications between the real-time process and EPICS and DAQ routines, along with IPC to other real-time processes on the same computer, are done through computer shared memory. This memory allocation and management is performed by mbuf, which is also developed as a kernel module, one object per computer. The source code is located in the $ROOT/src/drv/mbuf directory.

### 7.3 mx_stream

The real-time code produces a DAQ data buffer, in shared memory, every 1/16 second. Upon completion, a non-realtime program is triggered to pass this data on to the DAQ via a separate Ethernet. Source code for this communication program is located in the $ROOT/src/mx_stream directory.