

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

<b>Document Type</b> <b>LIGO-T980051-00 - E</b> June. 1998
<b>Getting Started with End-to-End model</b>
Biplab Bhawal, Matt Evans, Ed Maros, Malik Rahman and Hiro Yamamoto

*Distribution of this draft:*

xyz

This is an internal working note  
of the LIGO Project..

**California Institute of Technology**  
**LIGO Project - MS 51-33**  
**Pasadena CA 91125**  
Phone (626) 395-2129  
Fax (626) 304-9834  
E-mail: info@ligo.caltech.edu

**Massachusetts Institute of Technology**  
**LIGO Project - MS 20B-145**  
**Cambridge, MA 01239**  
Phone (617) 253-4824  
Fax (617) 253-7014  
E-mail: info@ligo.mit.edu

WWW: <http://www.ligo.caltech.edu/>

# 1 ABSTRACT

This document is intended to help you start using the End to End model by going through simple examples step by step. Appendix 2 summarizes the overall structure of the End to End package. This tutorial includes

- how to setup your computer to use the End to End model,
- how to run e2e to generate time series of the data and frequency spectrum,
- how to analyze the data, and
- how to write a simple module to add a new functionality to e2e.

The main purpose of this document is to go through all of these key steps as quickly as possible to let you have an idea what e2e is and let you judge yourself how e2e can be used. The details of the e2e environment are provided in the e2e core documents:

- **Overview of End-to-End Model** (LIGO-T970193)
- **Organization of End-to-End model** (LIGO-T970194)
- **Code Reference for End-to-End Model** (LIGO-T970195)
- **Physics of End-to-End Model** (LIGO-T970196)
- **Alfi - the GUI of the End-to-End Model** (LIGO-T980014)
- **Extending E2E Model - The How to Guide to Coding Modules** (LIGO-T980067)

The main components of e2e are (1) the time domain simulation engine, named Adlib, and (2) the front-end to configure the setup you are going to simulate, named Alfi. The first four documents describe Adlib, an application framework using Adlib and physics behind, and the last document is the manual for Alfi, which provides the Graphical User Interface front-end of the simulation package.

This document is organized as follows. In Sec. 4, the purpose and functionality of the End to End model are briefly explained. Sec. 5 summarizes what you have to do before you start this tutorial. Sec. 6 explains how to build a setup you want to simulate, followed by Sec. 7 which explains how to run e2e to simulate the setup you prepared. The output format of the data are discussed also in Sec. 7. Sec. 8 briefly explains how to add a new functionality to the e2e environment. Examples using C++, C and Fortran are given in a separate document (LIGO-T980067).

## 2 KEYWORDS

End to End model, time-domain, tutorial, e2e, introduction

## 3 DEFINITION OF WORDS

Adlib	Adlib : Digial Instrument Builder - C++ simulation engine used in e2e
Alfi	AdLib Friendly Interface - GUI front end of e2e
CVS	Concurrent Version System
e2e	End to End model
egcs	Free gnu compiler package, including C++,C,Fortran, available at egcs.cygnus.com.

## 4 WHAT IS END TO END MODEL

End to End model is a program package designed to simulate the LIGO interferometer in the time domain. The motivation and the main purpose of e2e are given in “**Overview of End-to-End Model**”. The underlying design is made so that

- new functionality can be easily added, and
- the program can be setup easily to simulate wide range of systems.

Because of this design, this can be used for many different purposes.

e2e is not a replacement of matlab or like, but rather a complement. The strength of these packages is the generality of their use. The high level language is flexible enough that many kind of calculations can be done much easier than the software development using low level languages, like C++. This “high level generality” introduces the overhead in the performance. This prohibited SMAC programmers to use the matlab controls in conjunction with the optics code written in fortran and implemented as a mexfile. The control system had to be written using fortran. e2e does not have this kind of generality, but is designed to avoid this kind of overhead.

## 5 PREPARING FOR E2E

The End to End simulation package has several programs and files in it. To let it work, you need to define some environmental variables summarized in the following table. PATH and LD\_LIBRARY\_PATH should be appended to the existing paths, and CVSROOT and E2E\_MAKEFILE\_CFG are needed when you want to add a new feature to e2e.

**Table 1: Environmental Variables on CIT-LIGO network**

<i>purpose</i>	<i>variable name</i>	<i>Value</i>
<i>All</i>	E2E_SOFT_HOME	/home/e2e/Software
	PATH	\${E2E_SOFT_HOME}/bin (append)
	LD_LIBRARY_PATH	\${E2E_SOFT_HOME}/lib (append)
	E2E_PATH	::\${E2E_SOFT_HOME}/lib
	E2E_LD_PATH	\${E2E_SOFT_HOME}/lib/modules
<i>code development</i>	CVSROOT	\${E2E_SOFT_HOME}/cvsroot
	E2E_MAKEFILE_CFG	\${E2E_SOFT_HOME}/config/Makefile.cf

One example of the setup under C shell is given below. Copy those lines in your .cshrc file for permanent use. If this does not work, ask your system manager.

```

setenv E2E_SOFT_HOME      /home/e2e/Software
setenv PATH ${E2E_SOFT_HOME}/bin/:${PATH}
setenv LD_LIBRARY_PATH ${E2E_SOFT_HOME}/lib:${LD_LIBRARY_PATH}
setenv E2E_PATH .:${E2E_SOFT_HOME}/lib
setenv E2E_LD_PATH ${E2E_SOFT_HOME}/lib/modules
setenv CVSROOT /home/e2e/Software/cvsroot
setenv E2E_MAKEFILE_CFG /home/e2e/Software/config/Makefile.cf

```

You don't need any programming, in order to do the simulation using the existing functionality of e2e ( Sec. 6 to Sec. 7 in this tutorial ). If want to add a new functionality, or if you want to customize the output ( Sec. 8 ) you will need compilers. The software is developed using egcs, free compilers including C++, C and FORTRAN (each called g++, gcc and g77, respectively). C++ compilers are now moving to the ANSI standard, and g++ compiler is close to that goal (but not quite). Other compilers, including SUN CC C++ compiler, may fail to compile e2e code because their implementation is not current. So, use g++, gcc and f77 for C++, C and FORTRAN compilers to avoid compatibility problems.

## 6 DEFINING WHAT TO SIMULATE

### 6.1. What is this step for

In order to do the simulation work using e2e, the first thing you have to do is to describe the setup you want to simulate. This step is the same as you write a source code in c, like

```

double foo( int in_i, double in_d )
{
    double ddd;
    d2 = exp(in_i) + sin(in_d);
    return ddd;
}

```

In stead of using a text editor to write the c source code, you use the program “alfi”, part of the e2e package, to write a file describing the setup, called a description file, using a GUI (graphical user interface). Instead of using built-in c functions, like sin or exp, you use primitive modules like “**field\_gen**”, which generates a monochromatic laser, or “**sideband\_gen**” which adds a sideband to the incoming laser. Just as c functions have input parameters of various types and return function values, you can define input and output ports of various predefined types to your description file. And just as your c function can call other functions you defined, one description file can use other description files.

In this tutorial, a simulation will be setup, step-by-step, to generate a simple thermal noise,  $f^{-1/2}$  tail plus a resonant peak. Also provided is a set of files which can be used to calculate responses of a Fabry-Perot cavity.

## 6.2. Alfi - the front end of the simulation engine

The details of this software is given in the Alfi manual. In the windows shown below, all texts written in italic and arrows are comments, and will not exit in the real window. When you have completed the setup explained in Section 5, just type **alfi** in the shell window to run the program. You will see the main window in Figure 1. Select “*Box*” from the “*File*” menu. A new window, “*Box selection window*” comes up. In this window, you create a new description file - called box file for short - or open an exiting one. Type *f\_half* for the name and click OK. Another window, *Box window*, comes up. Choose “*Save*” from the “*File*” menu in the main window. Congratulations, you have written a program using alfi. Make sure that there is a new file created in your directory named *f\_half.box*.

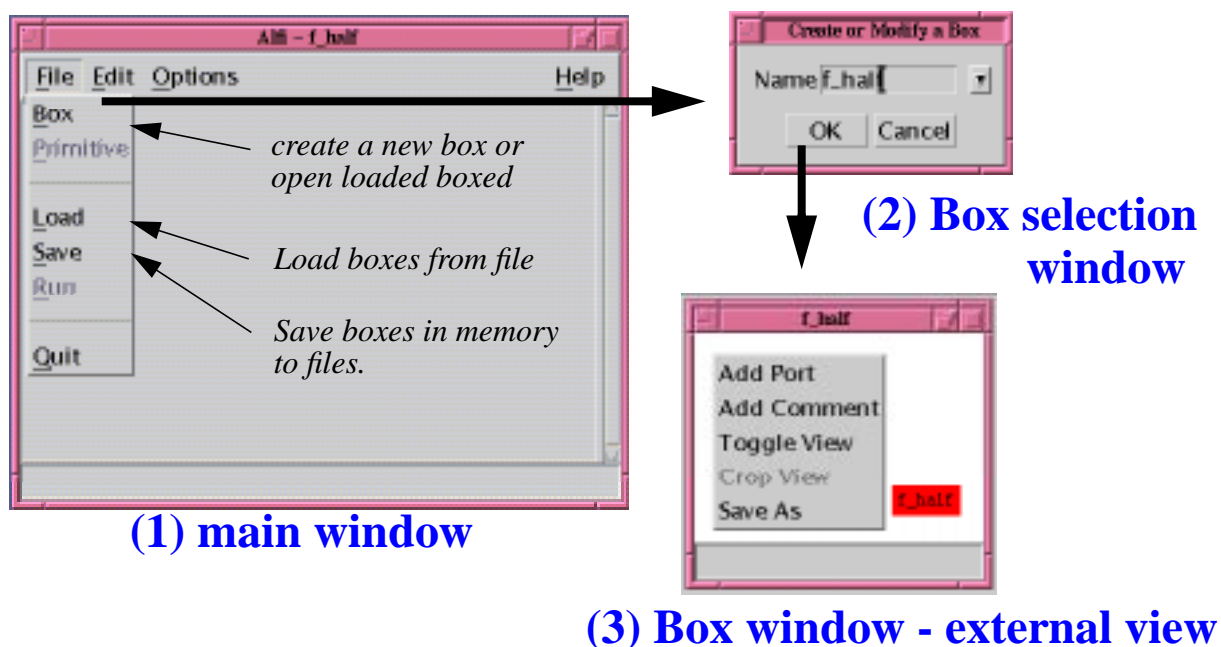
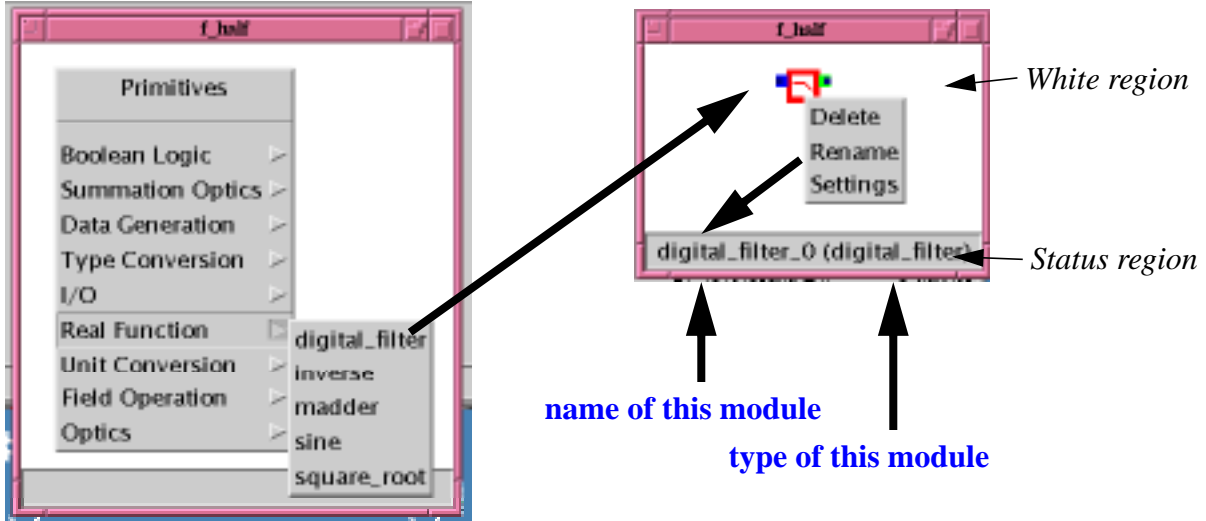


Figure 1: Starting up alfi

This box window is a view of the box from outside. Click in the white region in the window using the right mouse button. A menu shows up, as is shown in the external view in Figure 1. Select “*Toggle View*” from the menu. The window becomes empty. Now you are looking inside of the box, where you are going to put pieces together, or write a program.

Click inside the empty window using left mouse button. A popup menu of primitive modules shows up, as is shown Figure 2. Available primitives are summarized in Appendix 1 primitives. Primitives are grouped into different categories. Choose “*digital\_filter*” from the submenu at “*Real function*”. An icon is added to the window. Move the mouse on top of the icon. The narrow field at the bottom of the window is the status region, where information about the object of interest are shown. Now, two names are displayed in the zone. The first one is the name of this instance of the module, which is automatically assigned, and the type of the module. In terms of c programming, you have written

```
digital_filter digital_filter_0;
```



**Figure 2: Adding primitives in the internal view**

just as you would write

```
double ddd;
```

when writing c source code. If you want to name the module yourself, click the icon using the right button. A menu pops up as is shown in the right window in Figure 2. Select “Rename” and you can change it. Just as the c function name does not directly affect the real performance, the naming is not crucial. But the name of the port, which corresponds to input and output parameters of c functions and will be explained soon, should be named in such a way that one can easily understand the meaning of the port. To remove the box, select “Delete”.

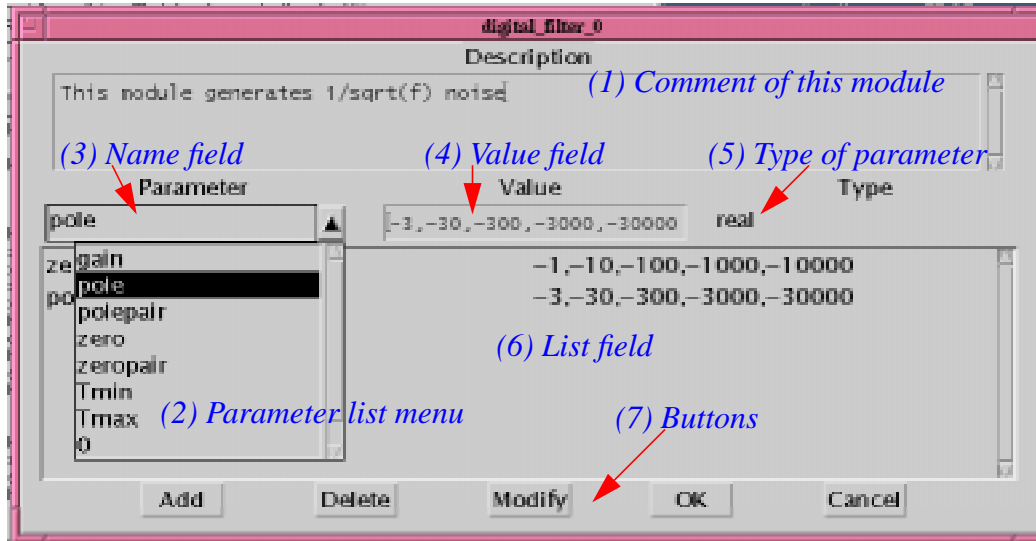
The digital\_filter in e2e is characterized by the following form, i.e., a overall normalization, real zeros ( $z_i$ ), complex zero pairs ( $z_{p_i}$ ), real poles ( $p_j$ ) and complex pole pairs ( $pp_j$ ).

$$G(s) = A \cdot \frac{\prod_{i1} (s - z_{i1}) \prod_{i2} (s - z_{p_{i2}}) \cdot (s - \overline{z_{p_{i2}}})}{\prod_{j1} (s - p_{j1}) \prod_{j2} (s - pp_{j2}) \cdot (s - \overline{pp_{j2}})} \quad (1)$$

Click the icon with the right button and you will see a popup menu in the right window in Figure 2. Select “Settings” from the menu, which brings up a window to define the settings of this module. Figure 3 is the settings window for the digital\_filter.

At the top of the window is a comment of this module. You select the parameter to set from the parameter list menu. To generate the  $1/f^{1/2}$  spectrum, the method used by VIRGO will be used, i.e., the filter is

$$G(s) = \prod_i \frac{s - z_i}{s - p_i} \quad (2)$$



**Figure 3: Setting window**

where  $z_i = (-3\text{Hz}, -30\text{Hz}, -300\text{Hz}, -3000\text{Hz}, -30000\text{Hz})$  and  $p_i = (-1\text{Hz}, -10\text{Hz}, -100\text{Hz}, -1000\text{Hz}, -10000\text{Hz})$ . To set a parameter, select the parameter, say pole, from the menu, and type in the value in the value field. When you select a parameter from the menu, the type of the parameter, e.g., real or integer, is displayed next the box where you enter the value. After you type in the value, you click “Add” button at the bottom. Then that setting appears in the List field at the bottom of the window. If you want to change the setting, click the parameter in the List field, change the value and click “Modify”. If you want to remove a setting, select the parameter in the List field and click “Remove” button.

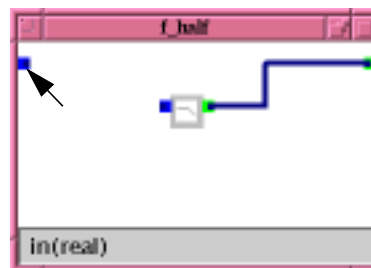
When all settings are done, **click “OK”. DON’T FORGET TO DO THIS. THEN SELECT SAVE FROM THE FILE MENU.** All modifications are done in the memory, and when you select “Save”, the content is saved in a file, whose name is the box name with “box” as the extension. E.g., the file name of the box “f\_half” is “f\_half.box”. The content in the file can be loaded into memory again using “Load” in the file menu. If you want to edit “h\_half” box later, choose “Load” from the menu, select “h\_half.box” file in the file selection dialog, select “Box” from the file menu, and choose “f\_half” from the popup menu in the window. When you load a box file, all related box files are automatically loaded and you don’t need to open each box file individually.

After closing the settings dialog, you will find the color of the box has changed from red to gray. This indicates that you have modified the box and customized the module.

Next, we define the input and output for this box, through which this module communicates with other components. Click the white area in the window with the right button. A menu in Figure 1 (3) appears. Select “Add Port”. The Port definition dialog appears as is shown in Figure 4. This dialog lets you define the input and output ports. Type the name of the port in the name field. Select “input”, make the data type to be “real”, and choose two “left”s and click OK. The blue box on the left edge of the window appears. The orientation buttons define where the port appears.



(1) Port definition dialog



(2) Ports and connections

Figure 4: Adding ports to box

Repeat the process to create another port, but this time, make it an output port on the right side of the box window.

**SELECT SAVE FROM THE FILE MENU, DO IT, DO IT NOW.**

As Figure 4 (2) shows, the `digital_filter` box has its own ports, one on the left, one on the right. An input port is represented by blue and an output port by green. Connect the output port of the box to the output port of the window, by clicking one port followed by another click on the other port. Then a line is drawn as in Figure 4 (2), showing the data flow chain. Do the same for the input link. When you want to remove a link line, move the mouse pointer over the line, watching the status region to confirm that a proper link is under the pointer, click the right mouse button, and select delete from the popup menu, just as you do to remove modules.

Toggle back to the external view of the `f_half` box. The `f_half` box now has two ports. Move the pointer to one of the ports. The name and the data type of that port is displayed in the status region of the window.

What you have done corresponds to the following pseudo c code.

```
void f_half( real in, real *out )
{
    digital_filter digital_filter_0;
    digital_filter_0( zeros = (-3,...), poles = (-1,...) );
    *out = digital_filter_0( in );
}
```

If you are interested in trying out how this filter works, go to Section 7.3. and Section 7.3. and follow the instructions.

### 6.3. Thermal noise module

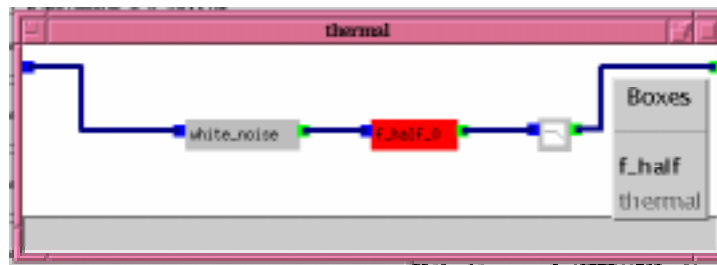
You have learned the basic technique to use `alfi`. In this section, a module is constructed which generates thermal noise containing  $1/f^{1/2}$  tail plus one resonance peak, i.e.,

$$x(\omega)^2 = \frac{1}{\omega \left( (\omega^2 - \omega_0^2)^2 + \frac{\omega_0^4}{Q^2} \right)} \quad (3)$$

The resonant part is approximated by a double-pole low-pass filter with poles at  $(-\omega_0/2Q_0 \pm i\omega_0)$ .

Let us create a configuration box to generate a time series of data which has the distribution given in Eqn.(3). Choose “Box” from the “File” menu, create a box named “thermal”, and switch to the internal view. Use the primitive menu and create a copy of module “rnd\_norm”, which is in “Data Generation” submenu, and “digital\_filter”. “rnd\_norm” module generates a random numbers with a normal distribution whose width is defined by the input “width”. This module is used to generate a white noise, so rename it to “white\_noise”.

We need to use “f\_half” box. Click in the white area in the “thermal” window using the left button with the control key down. You will see “Boxes” menu which has “f\_half” in it. Select it to create an instance of “f\_half”. This is like calling functions in c, but it is more powerful - see the alfi manual. Create an input port of type real and name it “noise\_width”, an output port of type real and name it “out”. Then link starting from the input port to rnd\_norm to f\_half to digital\_filter module to the output port.



**Figure 5: Thermal box and “Boxes” menu**

The white noise goes through the f\_half module to generate the  $1/f^{1/2}$  spectrum, then goes through the digital\_filter to add the resonance structure.

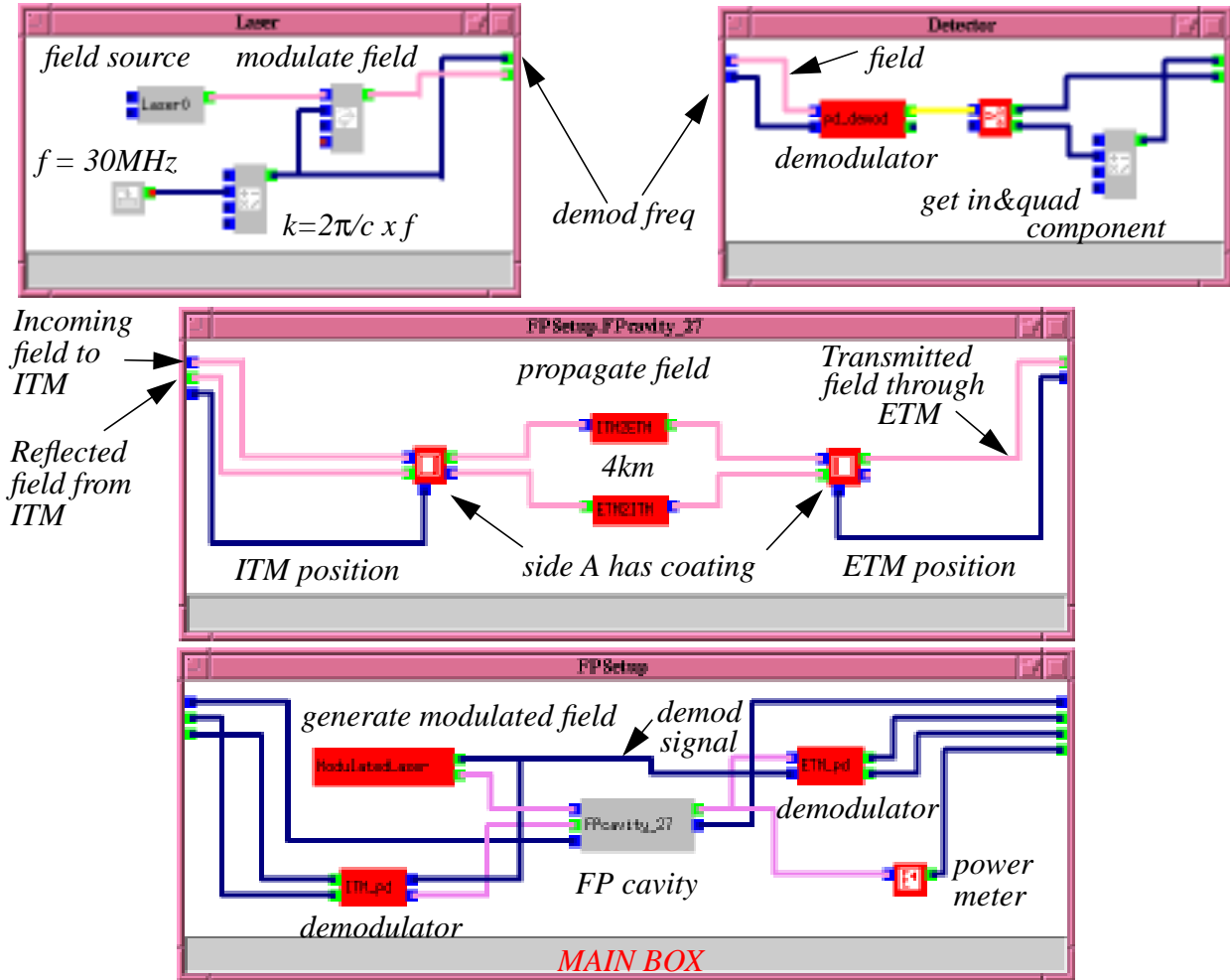
We need to set some parameters. First for white\_noise module. There is one parameter, “width”. This is the name of an input port of “rnd\_norm” module, meaning that the value is being passed from other module. Set it to 1. What this does is to provide a default value if no input is connected.

Next, open the digital\_filter setting, and set “polepair” to (-0.5, 1000). You need to specify only one of the complex conjugate pair, and you use a syntax (real, imaginary) to represent a complex number, and you can enter more than one value by putting “,” between values.

**SELECT SAVE FROM THE FILE MENU.**

## 6.4. Fabry-Perot example

In this tutorial, the following 4 box files are included: Laser.box, Detector.box, FPcavity.box and FPSetup.box. In this configuration, laser field is generated and modulated in Laser.box, passed into FPcavity.box, in which time evolution of the field is calculated with given values of the mirror positions, and the reflected and transmitted fields are demodulated and detected.



**Figure 6: Fabry-Perot cavity configuration**

One thing you have to be careful about is handling the optics, i.e., the side of the coating. By convention, the “**mirror**” primitive has the coating on side A. In order to make the FP cavity, you have to link - let field propagates back and forth - sides A of both mirrors. For convenience, the module can be rotated, so that side A, which usually faces to the left as ETM is now, can face to right. To rotate the module, move the mouse pointer on top of the module, and type the right or left arrow key. If you want to flip a module, instead of rotate, type x key or y key.

Using these box files, you can calculate the transfer functions or the field response for a swinging mirror. Some results will be shown in the following section.

## 7 RUNNING E2E AND ANALYZING THE OUTPUT

### 7.1. What is this step for

In order to simulate the configuration which you have just created, you run programs provided as a part of the e2e package. The current version of these programs produces ascii file outputs of data

of type real. The output is written each time new data are generated. The e2e package does not have data visualization software included yet. To see the data, use existing software like gnuplot or matlab or whatever you like which accepts the format explained in this section.

If you want to change the output format, you need to modify a software source code, `modeler_base` and `modeler_freq`. This is an advanced topic, and consult one of the core documents.

## 7.2. `modeler` and `modeler_freq`

`modeler` and `modeler_freq` are programs provided as a part of e2e package. The schematic structure is shown in Appendix 2 schematic view of e2e. `modeler` reads in the main box file, perform the time domain simulation with a specified time step, and writes the time series of data coming out from the output ports of the main box file.

`modeler_freq` is a software spectrum analyzer. It generates a signal of the form  $A \sin(2\pi f t)$  with a fixed frequency  $f$ , feeds that into one input port of a box, do the time domain simulation exactly the same as `modeler`, wait until the output becomes stable, i.e., the output behaves as  $B \sin(2\pi f t + \phi) + C$  with  $B$ ,  $\phi$  and  $C$  being time independent, print out  $B/A$  and  $\phi$ , then advance the frequency.

Both of these programs are derived classes of `modeler_base`. The “Overview” manual explains the details of the main program.

In the following, we will run these two programs, explaining how to interact with the program and how to analyze the output. In the following examples runs, **blue** text corresponds to prompts from the program, **red bold** are the inputs you type. *Black italic* text between `<<` and `>>` are the explanation for the inputs and outputs, so don't type.

## 7.3. $1/f^{1/2}$ filter

In the directory where you have the `f_half.box`, type “`modeler_freq`”. The following summarizes the inputs to the program.

```
<< box file name >>
Description file = f_half.box
<< just ., will be explained in the following example >>
Parameter File ( '.' for none): .

<< frequency response is written here >>
Output file name ( '.' to halt) = f_half.dat

<< You can save all setting values in one file for later review >>
Do you want to save the settings ?
Setting File Name ( '.' for no output): setting.dat

<< modeler_freq changes the time step depending on the frequency being analyzed.
Provide a widest safe range. There is no intrinsic time scale in this spectrum,
so it does not matter. >>
```

```

largest time step (range=[0:+INF],def=1) >> 1
smallest time step (range=[0:1],def=1e-05) >> 1e-8

<< just 0 >>
Convergence algorithm (0:accurate,1:fast) (range=[0:1],def=0) >> 0

<< you define which input port should be shaken and with what magnitude >>
Sin signal input port name (? to list all) [def="in"] >> ?
Input port names
    in
Sin signal input port name (? to list all) [def="in"] >> in
Amplitude of the sine signal (range=[-INF:+INF],def=1) >> 1

<< you decide which output ports are to be analyzed. >>
Enter output port number to be analyzed. -1 for all.
0 out
Port Number (-2 to end) (range=[-2:0],def=-1) >> -1

<< accuracy of result >>
convergence tolerance (range=[1e-05:1],def=0.01) >> 0.01

<< writes time series of the out from the output ports. This is useful
    mostly for debugging purpose - takes too long to converge, etc.
    So recommended to type n for no output. Remember, this may become
    very large. >>
file to dump the time series ( n for no dump ) = f_half.dump

<< frequencies are changed from the lower limit to the upper limit
    given below, with equal spacing in log. >>
lower limit of frequency (range=[1e-05:+INF],def=1) >> 10
upper limit of frequency (range=[10:1e+09],def=10000) >> 100000
number of points in this frequency range (range=[1:+INF],def=10) >> 100

<< some information printed on screen for each frequency >>
frequency = 10.000000, time step = 9.765625e-04 = 9.765625e-03 / freq ( 1 of 20 )
Data is analyzed at every 1 th event and is dumped to file at every 5 th event
    1.27051
frequency = 14.384499, time step = 4.882812e-04 = 7.023681e-03 / freq ( 2 of 20 )
Data is analyzed at every 1 th event and is dumped to file at every 7 th event
    2.17822
....
....

<< If you want to continue more, say yes >>
Continue? (y or n) n

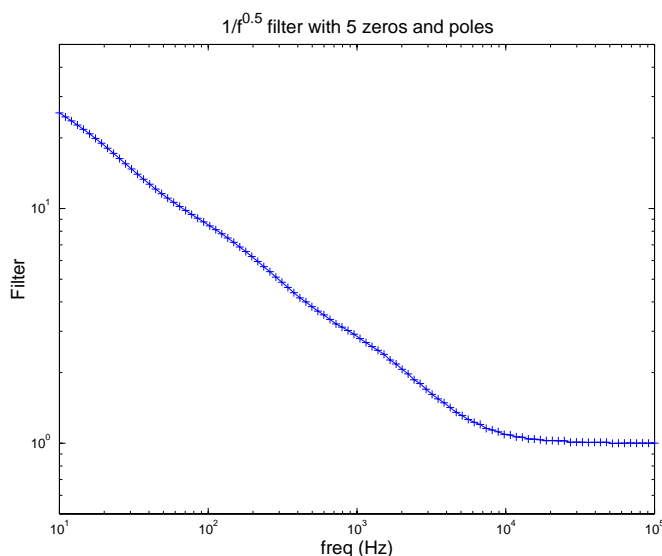
```

LIGO-DRAFT

The output file `f_half.dat` is a text file which contains the following data (see Section 7.2. for the definition of  $B/A$  and  $\phi$ ).

input frequency	B/A	$\phi$	estimated freq	#time steps
1.0000000000e+01	2.5556754892e+01	-7.1993819177e-01	1.0000008455e+01	20669
1.0974987655e+01	2.4578555289e+01	-7.2354438064e-01	1.0974985260e+01	100194
1.2045035403e+01	2.3629155218e+01	-7.2852960511e-01	1.2045031697e+01	93334
....				

The first row explains the meaning of each column and the actual data file has only numbers, no headers. The first column is the frequency of the source fed into the `f_half` “in” port, second is the magnitude of the transfer function, third is the phase. If there are more than one output port, this pair, magnitude and phase of the transfer function, of each output is repeated. The last two columns are performance measures. The column second from the last is the estimated frequency from the output, and this should be the same as the input frequency. The last column is the number of time steps needed to get the stabilized output. And the following is a plot with column 1 for  $x$  and column 2 for  $y$ .



**Figure 7: Transfer function of `f_half` box**

## 7.4. Thermal noise data

Type “`modeler`” to run the program to generate the time series of the output data. The following is the interaction with the program, using the same color convention as above.

```
<< the box file you want to simulate >>
```

```
Description file = thermal.box
```

```
<< thermal.box has one input, "noise_width". You can set the value here.
```

```
You prepare a file which contains a line
```

```

    noise_width = 1.0
    and type the file name here which has this setting. >>
Parameter File (\'\' for none): thermal.parm

<< output port data will be written in this file >>
Output file name (\'\' to halt) = thermal.dat

<< You can save all setting values in one file for later review >>
Do you want to save the settings ?
Setting File Name (\'\' for no output): setting.dat

<< The time step of the simulation. If you need a resolution at frequency f,
    choose the time step < 0.1/f. >>
Model time step = 1e-5

<< how many seconds do you want to simulate. If you have chosen 1e-5 for the
    time step, and if you choose 10 seconds here, there will be 10^6 time
    evolution loop. >>
Simulation time (s) = 10

<< You may not need to store all the data points simulated.
    If you have chosen the time step to 10^-5, and if you choose N here to 100,
    then the data is stored to the output file at every 10^-3 seconds. >>
Write one data point every N steps. N (0 to halt) = 100

<< If you want to run more, you can say yes. If you have changed
    the setting in the thermal.parm, the simulation uses new values
    in this data file for the input of the simulation. >>
Continue? (y or n) n

```

The contend of the thermal.dat looks like the following.

```

time      value at output
0.0010    1.0034109326e-08
0.0020    1.5753470856e-08
0.0030    -4.7999670067e-07
0.0040    -1.0085298240e-06
...

```

Essentially time and the value at the output port. If there are more than one output, they will be repeated in the same row.

The FFTed result using matlab is shown in Fig. 8.

In the same figure, the result obtained running modeler\_freq and the analytic form Eqn. (3) are also shown. Also superposed are two shapes,  $1/f^{1/2}$  and  $1/f^{5/2}$  for the reference. As can be seen from this figure, the model reproduce the shape well.

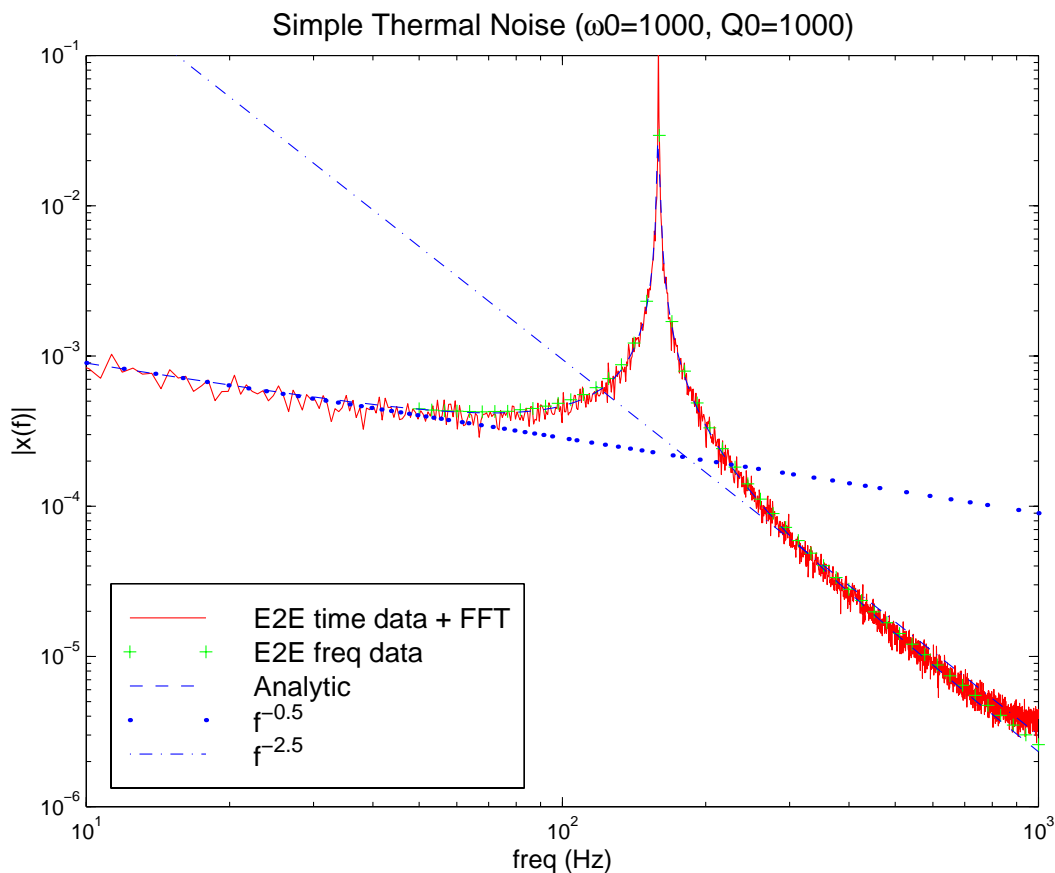


Figure 8: Frequency spectrum of thermal noise

## 7.5. Fabry-Perot - X func

Using the box files discussed in Sec. 6.4., we will calculate the transfer function and the response when the mirror swings. First the transfer function from the ETM to the reflected field. You run `modeler_freq` and specify the `FPSsetup.box` as the description file. In the following, some parts of the input are explained, using the same coloring convention.

```
<< When using optics, the time step is important. You have to choose
  an integer fraction of one-way-trip-time. modeler_freq adjusts the
  time step depending on the audio modulation frequency. For the upper
  limit, enter a number close to 0.01/f_max, where f_max is the maximum
  frequency of interest. >>
```

```
Enter the range of acceptable time steps
```

```
largest time step (range=[0:+INF],def=1) >> 0.6667e-5
```

```
smallest time step (range=[0:6.667e-06],def=6.667e-11) >> 0.6667e-7
```

```
<< If the worst-case estimated error using the input time step exceeds
  10%, the following warning is printed. This message could appear
  during the run time when the frequency is changed and the time
  step had to be changed. If you see messages, check if the frequency
```

```

    range (the third line of the message) is satisfied. >>
-> Root.FPcavity_27.ITM2ETM * 200 internal delay steps.
-> Root.FPcavity_27.ITM2ETM * 12.8451% time step error.
-> Root.FPcavity_27.ITM2ETM * Use this for frequency << 1.1677e+06

<< There are several inputs, and you have to choose which one to shake >>
Sin signal input port name (? to list all) [def="ETMx"] >> ?
Input port names
    ETMx
    ITMx
    Root.ModulatedLaser.freq
Sin signal input port name (? to list all) [def="ETMx"] >> ETMx

<< The amplitude should be chosen small enough to stay in the linear region >>
Amplitude of the sine signal (range=[-INF:+INF],def=1) >> 1e-13

<< There are 5 outputs, and you choose which one to be analyzed.
    In the final output, a pair of data, magnitude and phase of the
    transfer function, for the selected output ports are placed
    in this order in one row. >>
Enter output port number to be analyzed. -1 for all.
0 TransPower
1 InETM
2 QuETM
3 InITM
4 QuITM
Port Number (-2 to end) (range=[-2:4],def=-1) >> 1
Port Number (-2 to end) (range=[-2:4],def=-2) >> 2
Port Number (-2 to end) (range=[-2:4],def=-2) >> 3
Port Number (-2 to end) (range=[-2:4],def=-2) >> 4
Port Number (-2 to end) (range=[-2:4],def=-2) >>

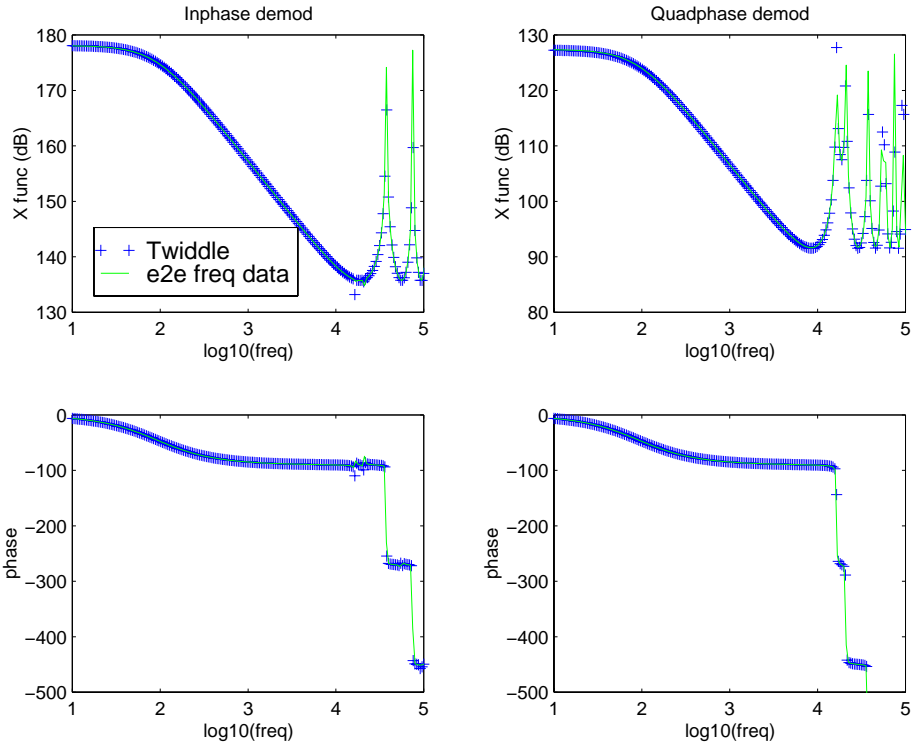
```

The transfer function from the ETM motion to the reflected field is shown in Fig. 9, together with the calculation of twiddle. Both the magnitude and phase agree well.

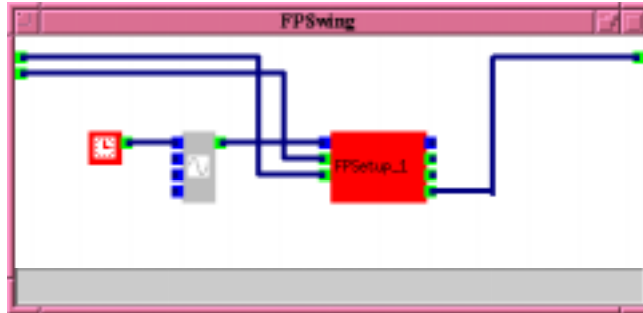
## 7.6. Fabry-Perot - swinging mirror

In order to swing the mirror, we generate the x value and connect the value to the port ITMx of the box FPSsetup, which internally is connected to the ITM mass position. The x value is generated using a “sine” module, which generates the sinusoidal shape, with a “clock” module which provides the current time. The settings of the sine module is done so that it swings with an amplitude of  $2\mu$  with frequency of 1Hz.

Run modeler with the FPSwing.box as the description file, time step of  $0.6667e-5$ , and the result shown in Fig. 11 is obtained. This is consistent with the measurement of the 40m one arm open loop measurement.



**Figure 9: Transfer function from ETM to ITM reflected field**

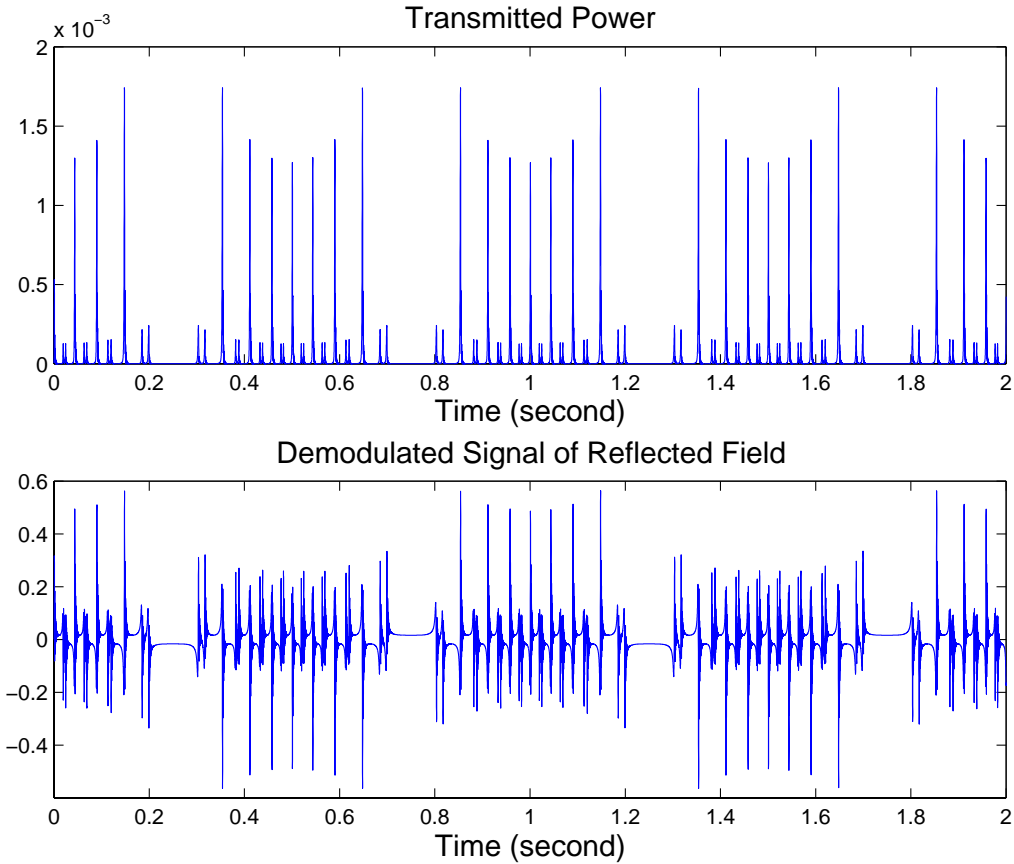


**Figure 10: mirror motion input**

## 8 ADDING A NEW FUNCTION

### 8.1. What is this step for

Adlib, the simulation engine of e2e included in modeler and modeler\_freq, has many functionalities included, like “field\_gen” or “sideband\_gen”. When some new functionality is needed, one can add the new capability. The C++-based modular design allows you to concentrate on the physics you want to add anew, not programming issues like the interaction with the time-



**Figure 11: Swinging mirror**

loop. In this tutorial, the “limiter” primitive, which simulates a saturation effect, will be exemplified in details. Another document, **Extending E2E Model - The How to Guide to Coding Modules** (LIGO-T980067) gives examples written in C++, C and FORTRAN.

## 8.2. simple example

The following is an excerpt from `math_function.h` and `math_function.cc`. This is a module to add a new function,  $a*x + b*y$ . If you need to add a new module which accepts several real value inputs and generates a new value, only things you have to do is to modify the place shown in red-italic.

```
// class declaration. Use as it is, just simply substitute your class name
class madder : public real_function
{
public:
madder(const string& name_arg = "", const module* parent_arg = NULL);
~madder();
module* new_type(const string& name_arg, const module* parent_arg) const;
void action();
};
```

```

// code implementation
madder::madder
(const string& name_arg, const module* parent_arg)
// In the following, the third argument is the module name
// and the last argument is the number of inputs.
: real_function(name_arg, parent_arg, "madder", 4)
{
// set_input_name(id,name) gives a name to the input port.
// set_default_input(id,val) gives a default value when nothing is connected.
set_input_name(0, "a");
set_default_input(0, 1.0);
set_input_name(1, "x");
set_default_input(1, 0.0);
set_input_name(2, "b");
set_default_input(2, 1.0);
set_input_name(3, "y");
set_default_input(3, 0.0);
}

// just that
madder::~~madder()
{}

// just copy as it is, but use your module class name
module* madder::new_type
(const string& name_arg, const module* parent_arg) const
{ return new madder(name_arg, parent_arg); }

// This is where you spend most of your intelligence.
void madder::action()
// in(0) = a, in(1) = x, in(2) = b, in(3) = y
{ output = in(0) * in(1) + in(2) * in(3); }

```

In short, to implement a new module which looks like

```
double func( double a1, double a2, ... )
```

what you need to do is to specify the number of inputs, their names and default values, and of course the content of the function.

### 8.3. Real story

If you want to do more, you need to do more, but the class hierarchy is designed to minimize the overhead. Once the e2e environment is setup, not only the environment variables discussed in

Section 5, but all necessary directories and cvs setup, you need to change 2 to 3 lines in Makefile, possibly in modeler\_base.cc, and type make. No fine prints anywhere else.

You can create a new application which can generate different kind of outputs. modeler\_base is a base for the application framework, which is explained in the “Overview” document. You can create a derived class of modeler\_base, and you can concentrate on the change you want to make, without bothering how to make a time loop or read in the description file.

## **9 SUMMARY**

Please give us feed back what are missing. We want to make this document useful.

LIGO-DRAFT

# APPENDIX 1 PRIMITIVES

The latest version of this table is available in T970193\_Overview.fm5.

<i>Name</i>	<i>Function</i>	<i>in</i>	<i>out</i>	<i>setting</i>
<b>I/O</b>				
data_in	used to get data into the simulation	none	"0" variable type	"type" string ("real"), "init" output type (???)
data_out	used to get data out of the simulation (a "probe")	"0" variable type	none	none
<b>Real Function</b>				
madder	implements $z = a*x + b*y$	"a" "x" "b" "y" real	"0" real	none
sine	the sine function	"0" "amplitude" "frequency" "phase" real	"0" real	none
square_root	the square root function	"0" real	"0" real	none
inverse	the inverse function	"0" real	"0" real	none
digital_filter	a digital filter	"0" real	"0" real	"zero" "pole" "gain" real "zeropair" "polepair" complex
limiter	models a circuit with rails	"0" "upper" "lower" real	"0" real	none
<b>Logic functions</b>				
<b>Input "val" is evaluated to be true if <math>val &gt; threshold</math>, otherwise false. Output is true_val if the result is logical true, false_val otherwise.</b>				
and	logical AND	"a","b" real	"0" real	"threshold" (0.9), "true_val" (5), "false_val" (0.0) real
or	logical OR	"a","b" real	"0" real	same as above
a>b	comparison	"a","b" real	"0" real	same as above
not	negation	"0" real	"0" real	same as above
switch	if bool is true, return high, else return low.	"bool" "low" "high" real	"0" real	same as above
<b>Data Generation</b>				
rnd_flat	generates random numbers with a flat distribution	"range" real	"0" real	none
rnd_norm	generates random numbers with a normal distribution	"width" real	"0" real	none
clock	generates the time	none	"0" real	none
<b>Unit Conversion</b>				
lam2k	converts wavelength to wavenumber	"0" real	"0" real	none

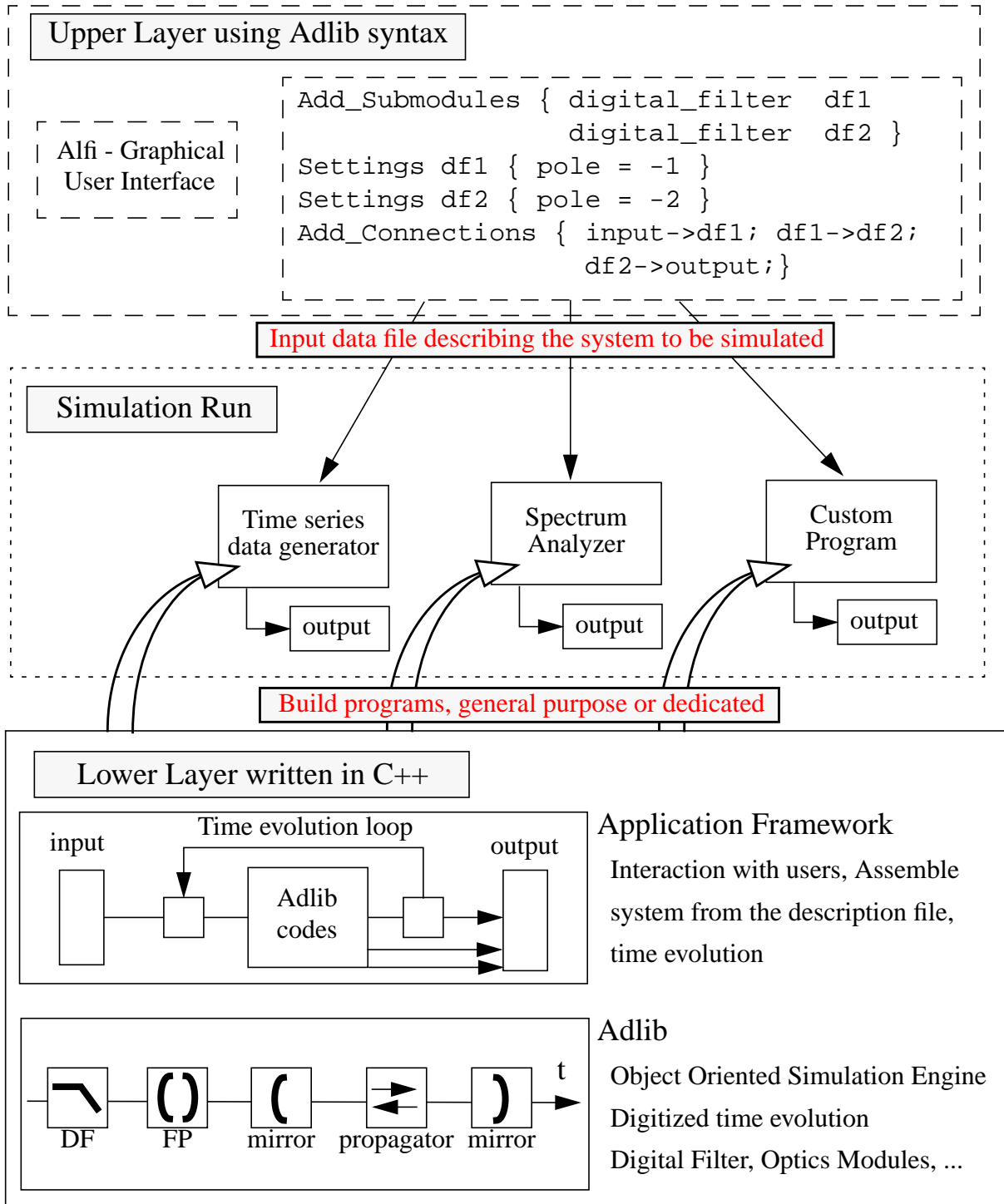
<i>Name</i>	<i>Function</i>	<i>in</i>	<i>out</i>	<i>setting</i>
f2k	converts frequency to wavenumber	"0" real	"0" real	none
<b>Type Conversion</b>				
field2complex	converts a field to a complex number	"0" single_mode_field, "dk" real	"0" complex	none
complex2reim	converts a complex number to real and imaginary	"0" complex, "phi" real	"real" "imag" real	none
complex2aphi	converts a complex number to amplitude and phase	"0" complex	"amp" "phi" real	none
<b>Field Operation</b>				
field_gen	generates a field	"power" "phi" real	"0" single_mode_field	"lambda" real (1.0)
sideband_gen	phase modulates a field (uses sideband approximation)	"0" single_mode_field, "k_mod" "gamma" real, "order" integer	"0" single_mode_field	none
phase_adder	phase modulates a field directly	"0" single_mode_field, "phi" real	"0" single_mode_field	none
power_meter	outputs the power in a field	"0" single_mode_field	"0" real	none
pd_demod	demodulates a field	"0" single_mode_field, "k_demod" real	"demod" complex, "power" real	none
<b>Optics</b>				
prop	propagates a field over a macroscopic distance	"0" single_mode_field	"0" single_mode_field	"length" real (1.0) "dphi" real (0.0) (f)
mirror2 (a)	a 2-input 2-output mirror (cavity end mirror)	"x" real "Ain" "Bin" single_mode_field	"Aout" "Bout" single_mode_field	"r" "t" "R" "T" "L" real (2.0), (d) "angle" real (0.0)
mirror4  void don't use	a 4-input 4-output mirror (beam splitter, pickoff)	"x" real "A+in" "A-in" "B+in" "B-in" single_mode_field	"A+out" "A-out" "B+out" "B-out" single_mode_field	"r" "t" "R" "T" "L" real (2.0), (d),(e) "angle" real (M_PI/4.0)

LIGO-DRAFT

<i>Name</i>	<i>Function</i>	<i>in</i>	<i>out</i>	<i>setting</i>
<b>Summation Optics:</b>				
cav_sum (b)	represents a short FP cavity	"xA" "xB" "xC" real, "Ain" "Bin" single_mode_field	"Aout" "Bout" "Cout" single_mode_field	"length" real (1.0), "dphi" real (0.0) "dirA" real (1.0), "dirB" real (1.0) "rA" "tA" "RA" "TA" "LA" real (2.0), (d) "rB" "tB" "RB" "TB" "LB" real (2.0), "rC" "tC" "RC" "TC" "LC" real (2.0)
rec_sum (c)	represents a recycled MIFO	"xA" "xB" "xC" "xD" real, "Ain" "Bin" "Cin" "Din" single_mode_field	"Aout" "Bout" "Cout" "Dout" "Bpick" "Cpick" "Dpick" single_mode_field	"lengthA","lengthB","lengthC" real (1.0), "dphiA","dphiB","dphiC" real (0.0) "dirA","dirB","dirC","dirD" real (1.0) "rA" "tA" "RA" "TA" "LA" real (2.0), (d) "rB" "tB" "RB" "TB" "LB" real (2.0), "rC" "tC" "RC" "TC" "LC" real (2.0), "rD" "tD" "RD" "TD" "LD" real (2.0),

LIGO-DRAFT

## APPENDIX 2 SCHEMATIC VIEW OF E2E



**Figure 12: Schematic view of the End to End model**