LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

- LIGO -

CALIFORNIA INSTITUTE OF TECHNOLOGY

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Document Type **LIGO-T980051-00 - E** June. 1998

Getting Started with End-to-End model

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 documemt (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

purpose	*variable name*	*Value*
All	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
code development	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.

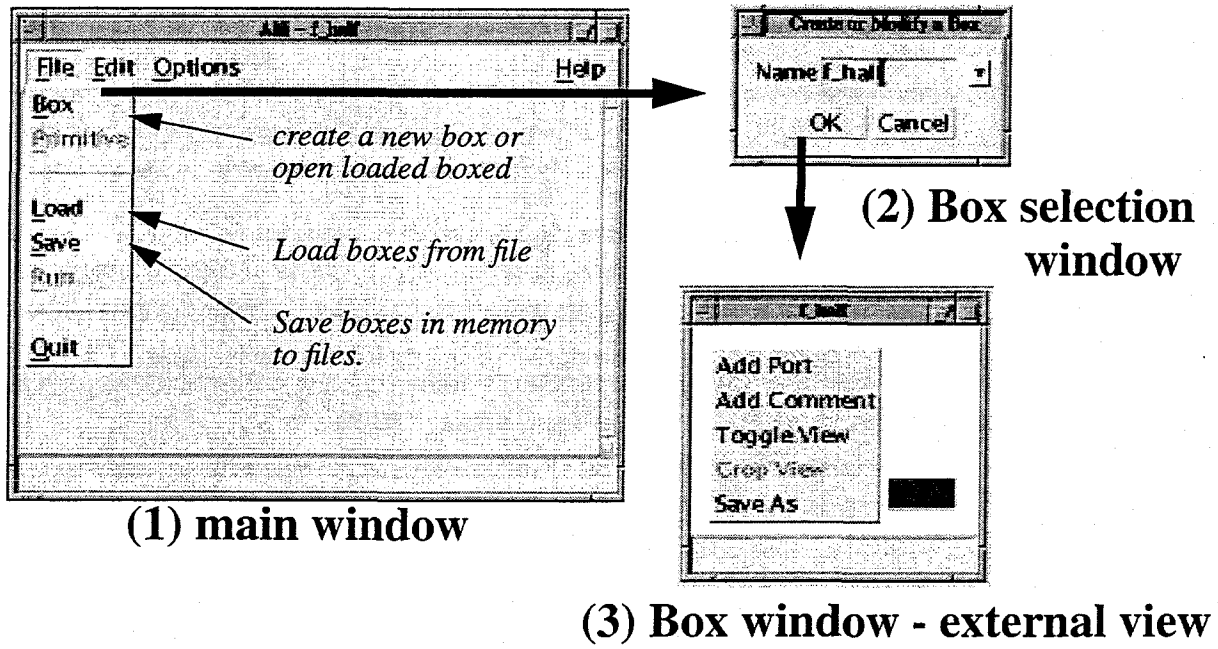


Figure 1: Starting up alfi

This box window is a view of the box from out side. 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;
```

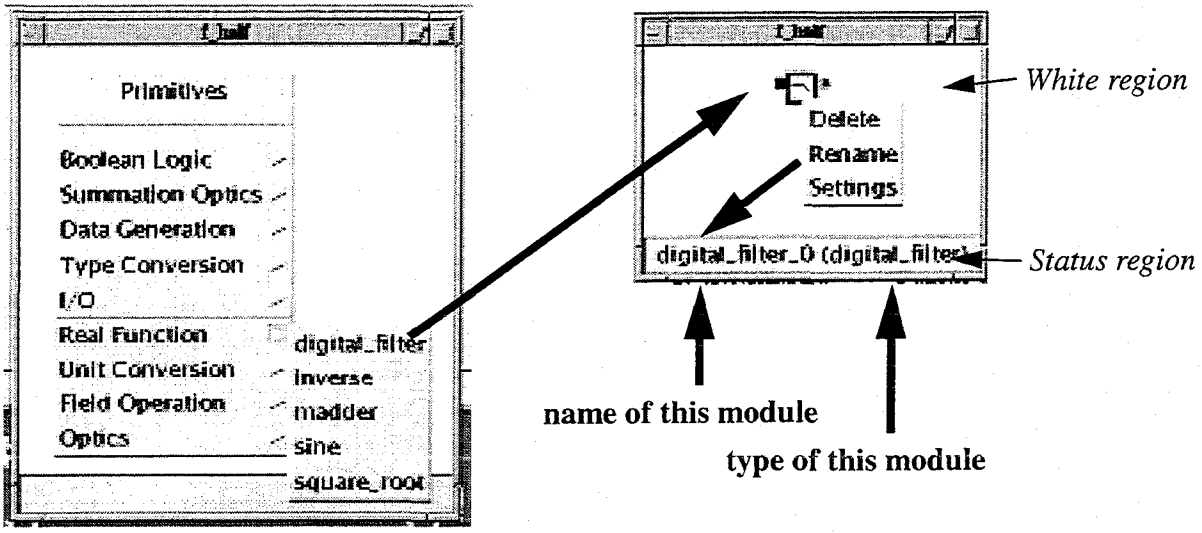


Figure 2: Adding primitives in the internal view

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 (zp_i), real poles (p_j) and complex pole pairs (pp_j).

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

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

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

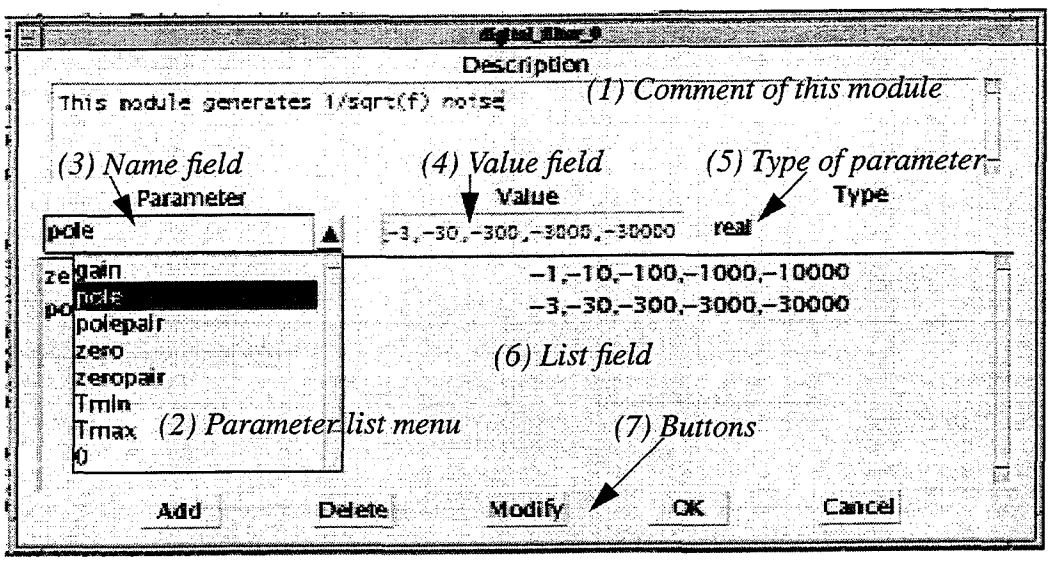


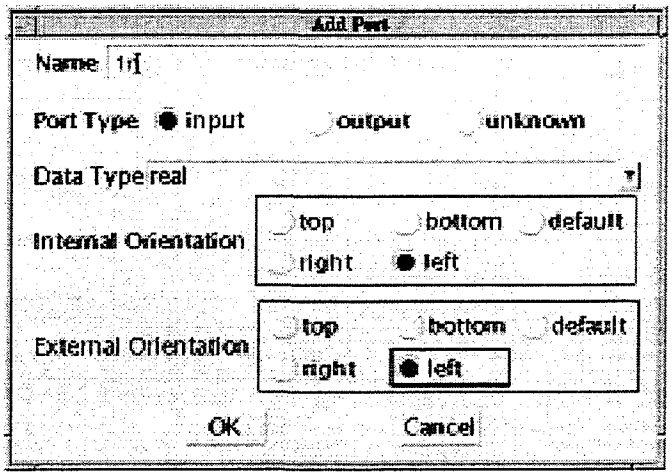
Figure 3: Setting window

where z_i=(-3Hz, -30Hz, -300Hz, -3000Hz, -30000Hz) and p_i=(-1Hz, -10Hz, -100Hz, -1000Hz, -10000Hz). 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 clock "Remove" button.

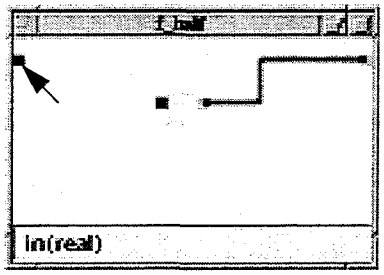
When all settings are done, click "OK". DON'T FORGET TO DO THIS. THEN SELECT SAVE FROM THE FILE MENU. All modifications are done in the memory, and when you select "Save", the content is saved in a file, whose name is the box name with "box" as the extension. E.g., the file name of the box "f_half" is "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 popupemnu 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 ($-\omega_0/2Q_0$ ω_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.

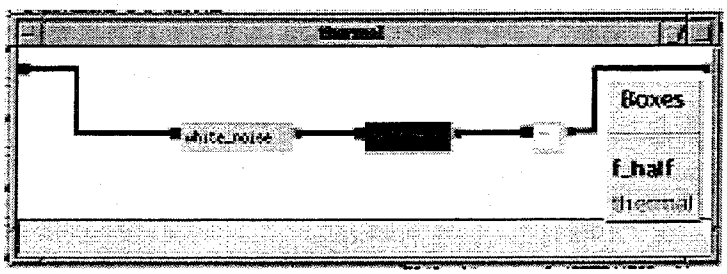


Figure 5: Thermal box and "Boxes" menu

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.

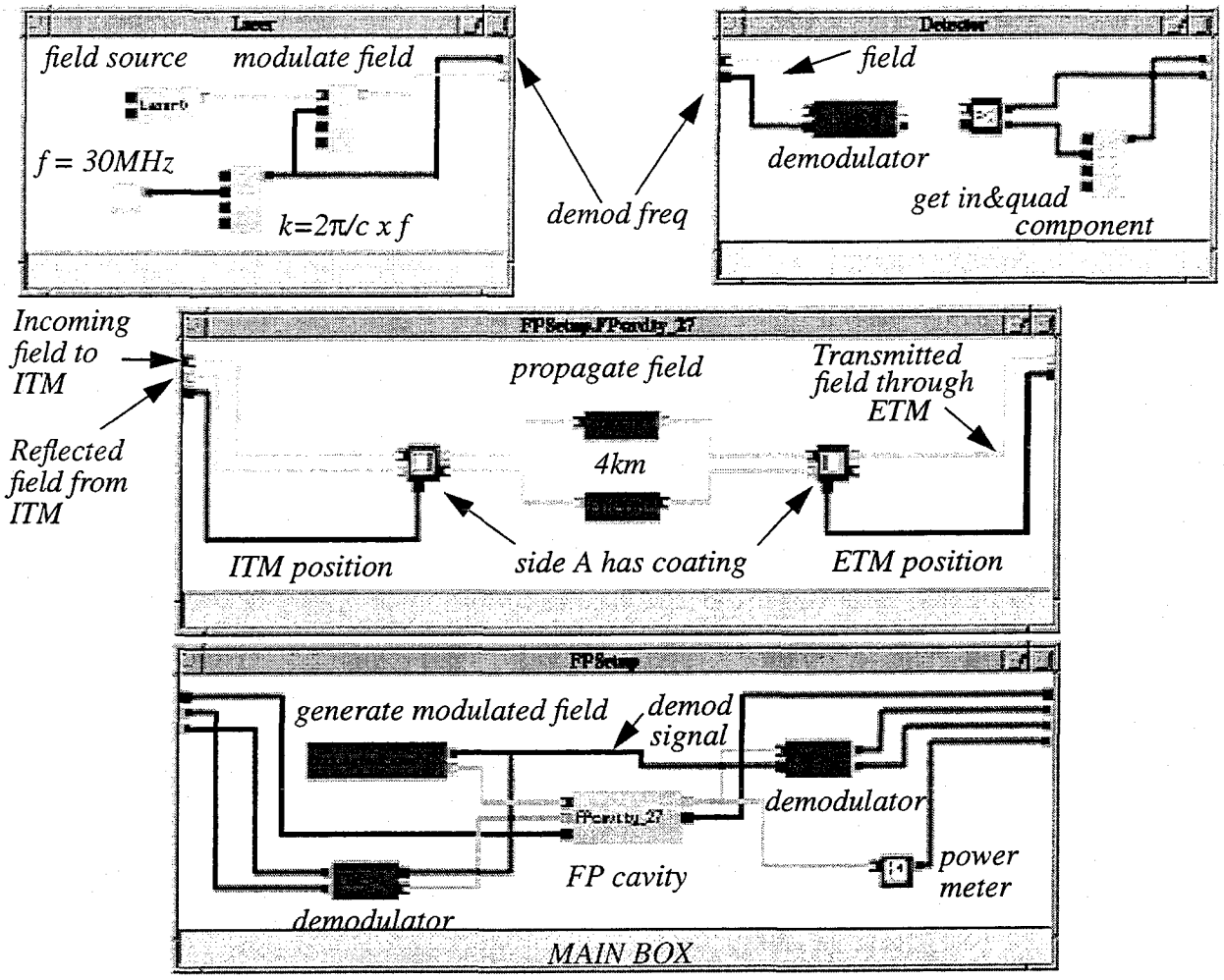


Figure 6: Fabry-Perot cavity configuration

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, in stead 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, ϕ and C being time independent, print out B/A and ϕ, 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. >>
```