EtherLab

Version 1.2

created by

Ingenieurgemeinschaft IgH

Essen

February 4, 2009

Ingenieurgemeinschaft IgH
Heinz-Bäcker-Str. 34
D-45356 Essen
Tel.: +49-201-360 14-0
Fax.: +49-201-360 14-14
E-mail: igh@igh-essen.com

Contents

List of Figures

1 Introduction

The name EtherLab once stood for an open-source solution of using the EtherCAT fieldbus technology with MATLAB/Simulink for automation and test purposes. Today, EtherLab is much more than this: It is neither depending on EtherCAT, nor on MATLAB/Simulink any more. Below is a short summary of what EtherLab is capable of:

- Realtime execution of multiple applications consisting of multiple tasks with signals and parameters.
- Accessing the applications' signals and parameters via TCP/IP. There is a generic C++ library called PdCom and many tools for visualisation, data logging, etc. available from `http://etherlab.org/en/components.php`.
- Creation of realtime applications with a C-API (see section 4).
- Creation of realtime applications using MATLAB/Simulink and the EtherLab blockset (see section 5).

2 Architecture

The EtherLab package can be subdivided into the components listed below:

- The realtime application core (represented by the Linux kernel module `rt_appcore`), that is responsible for the execution of the realtime applications. It manages the applications tasks, signals and parameters.
- The buddy process, acting as a userspace counterpart to the `rt_appcore`. It maintains internal representations of the loaded applications and is responsible for presenting signals and parameters via TCP/IP (see figure 2.1). Each application is accessible[1] via an own TCP port.
- The EtherLab C-API, that is used to create realtime applications, basically consisting of tasks, signals and parameters. The C-API provides a generic way to describe applications using an EtherLab realtime application description XML file that is a source for the application's data structures and the appropriate information for the buddy process, that is necessary for presenting the data.
- The MATLAB/Simulink integration of EtherLab in terms of the `etherlab_lib` for Simulink (containing all necessary blocks) and the EtherLab target for the Realtime Workshop, that is able to create EtherLab applications from Simulink models.

[1] To access the realtime application data via TCP/IP, there is a platform-independent library called PdCom, that is not part of this document. See http://etherlab.org/en/components.php.

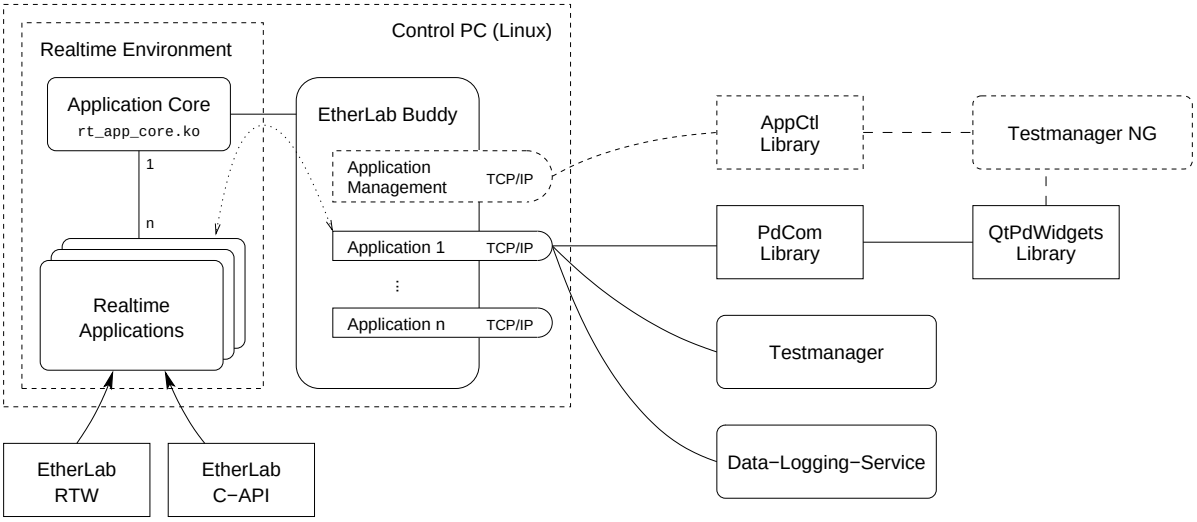

Figure 2.1: EtherLab Component Overview

3 Installation

3.1 Prerequisites

Prerequisites for the installation of EtherLab:

- Linux kernel 2.6 (including kernel sources)
- RTAI $\geq$ 3.3
- *optional:* MATLAB/Simulink with Realtime Workshop, if Simulink shall be used to create EtherLab realtime applications. Recommended are MATLAB versions 2006x – 2007x.
- *optional:* EtherCAT master 1.4, if the Simulink EtherCAT blocks shall be used. See http://etherlab.org/en/ethercat.

It is recommended to install EtherLab components to `/opt/etherlab`. If this directory does not exist, it can be created with the commands below:

```
# mkdir /opt/etherlab
# chown root:users /opt/etherlab
# chmod 775 /opt/etherlab
```

3.2 Installation

Installation is done after copying the EtherLab tarball from the EtherLab CD (or downloading from http://etherlab.org) as normal user according to the commands below. The last command installs EtherLab into the directory `/opt/etherlab`. A different target directory can be selected via the `--prefix` parameter of the `configure` script (see `configure --help`).

```
$ tar xzf etherlab-1.2.tar.gz
$ cd etherlab-1.2/
$ ./configure
$ make
$ make install
```

3.3 Installation of the Simulink Library

As a prerequisite for the `setup_etherlab` command (see below) to succeed, the file `toolbox/local/pathdef.m` inside the MATLAB installation directory must be writable for the user running MATLAB. Enter the below commands (as *root*) to make it writable for all users in the group *users* (substitute the variable `$MATLABDIR` with the correct path first):

```
# chown :users $MATLABDIR/toolbox/local/pathdef.m
# chmod 664 $MATLABDIR/toolbox/local/pathdef.m
```

To setup the EtherLab Simulink library `etherlab_lib` for the use in MATLAB, the below commands must be entered (out of MATLAB).

```
>> cd /opt/etherlab/rtw
>> setup_etherlab
```

3.4 Starting the Application Core as a Service

If the EtherLab application core shall be started as a service, the provided init script `etherlab` can be used. However, the insertion of the service is dependent on the GNU/Linux distribution used. The command sequence below is intended for openSUSE Linux:

```
# ln -s /opt/etherlab/etc/init.d/etherlab /etc/init.d/
# insserv etherlab
```

4 The EtherLab C-API

4.1 Concept

EtherLab provides a generic way of executing realtime applications in the Linux kernel and accessing their signals and parameters from userspace. The EtherLab C-API provides functions to

- Initialize and clean up realtime applications.
- Accept information about the internal structure of a realtime application, i. e. application attributes, different tasks, signals and parameters, to allow access from userspace.
- Execute application tasks with different sample times.

The realtime application description is done via an XML file (see section 4.2). This file is parsed to create the C data structures for signals and parameters (see fig. 4.1). This means the user only has to provide the application description XML and the C-code algorithms operating on the given data structures to create an EtherLab realtime application. The application core and the buddy process provide mechanisms to make the application's variables visible and accessible from userspace.

4.2 Realtime Application Description XML

Listing 4.1 shows a minimal realtime application description.

Listing 4.1: Minimal realtime application description

```
<?xml version="1.0"?>
<application>
  <description>EtherLab example application</description>
  <version>Version string</version>
  <task basetick="1000"/>
  <data>
    <signal name="signal1" datatype="double_T"/>
    <parameter name="parameter1" datatype="double_T"/>
  </data>
</application>
```

4.2.1 Application

The root of the realtime application description XML is the `<application>` element. It may contain the following elements (the order is mandatory).

`<description>` (*mandatory*) May contain an arbitrary textual description of the application. Please note, that this results in a C string, so please avoid double quotes and `\0`.

`<version>` (*mandatory*) May contain an arbitrary version string. The same constraints as for the `<description>` element apply.

`<task>` (*mandatory*) This sets the base period of the application's main task. The period is set via the `basetick` attribute, that is interpreted as time period in µs and will be later referenced as sample time index 0. The element may optionally have `<subtask>` child elements with a mandatory `decimation` attribute. Each of them will create a subtask with an execution period of the application's base period divided by the given decimation.

`<refsystem>` (*optional*) Reference systems serve as target systems for `<reference>` elements, that can appear in system-like elements (see section 4.2.2). The element itsself is a system-like element. An arbitrary number of reference systems can be defined in this place.

`<data>` (*mandatory*) This is the root of the application's data structure and can be considered as a system-like element (see section 4.2.2).

4.2.2 Systems

The elements `<data>`, `<subsystem>` and `<refsystem>` can be considered as system-like elements, that behave similar in a certain way. Below is a list of elements that can appear as child elements in system-like elements. System-like elements can contain subsystems in terms of the `<subsystem>` or `<reference>` elements, so that nested tree structures can be created. Each node is identified by a name. As a result, any children of system-like elements must have a unique `name` attribute.

`<signal>` Defines a signal as part of the parent system. A signal is a variable, that is calculated at a given sample time. See section 4.2.3 for more information on variables. The element can accept the following attributes:

`name` (*mandatory*) The name of the signal. This must be a valid C variable name.

`datatype` (*mandatory*) One of the variable data types listed in section 4.2.3.

`sampleTimeIndex` (*optional*) The index of the sample time referencing the task or subtask defined in the application (see section 4.2.1). This also specifies the fastest rate of change for the signal. If `sampleTimeIndex` is omitted, the signal will belong to the fastest sample time.

`alias` (*optional*) An alternate global name for the signal.

<parameter> Defines a parameter variable as part of the parent system. As far as the application is concerned, a parameter is a constant value. It is only possible to change the value via external mechanisms provided by `rt_kernel`. See section 4.2.3 for more information on variables. The following attributes are possible for this element:

name (*mandatory*) The name of the parameter. This must be a valid C variable name.

datatype (*mandatory*) One of the variable data types listed in section 4.2.3.

<subsystem> A subsystem will create a deeper nesting level and a new name space for signals, parameters and further subsystems. The `name` attribute defines the name of the subsystem and is therefore mandatory. A subsystem will result in an unnamed C structure. If a named data structure is required, use `<reference>` instead.

<reference> A reference will create a subsystem as an instance of a `<refsystem>` (see section 4.2.1). The `name` attribute defines the name of the subsystem and therefore is mandatory. A reference will create a named C structure, that can be used as a data type in C code (for example as a function parameter's type).

4.2.3 Variables

Signals and parameters are considered as variables. Variables are the "leaves" of the tree of systems defined in the application's `<data>` section and are the elements that make up the realtime application's data.

The available variable data types are listed below:

EtherLab name	Size in byte
`uint8_T`	1
`sint8_T`	1
`int16_T`	2
`sint16_T`	2
`uint32_T`	4
`sint32_T`	4
`float_T`	4
`double_T`	8

Due to their special character, variables may be described with the below child elements in the application description XML. The order is mandatory again.

<dim> (*optional*) A variable can be multi-dimensional. Any `<dim>` element will add another dimension to the variable. If omitted, the variable is considered as a scalar. Every `<dim>` element has a mandatory `value` attribute, that specifies the size of the dimension. For example

```
<signal name="matrix" datatype="double_T">
  <dim value="5"/>
  <dim value="7"/>
</signal>
```

will create a signal named `matrix` that is a 5×7 matrix of double-precision floating-point values, so the resulting C variable will be defined as `double_T matrix[5][7];`.

`<value>` (*optional*) Default values may be specified for variables on application startup. If the variable is a scalar, a single element (for example `<value>8</value>`) will be sufficient. If the variable is multi-dimensional, the elements can be nested according to the dimensions, like in the example below. If a `<value>` element is ommitted, if will default to zero.

```
<parameter name="matrixParam" datatype="uint8_T">
  <dim value="2"/>
  <dim value="3"/>
  <value>
    <value>
      <value>1</value>
      <value>2</value>
      <value>3</value>
    </value>
    <value>
      <value>4</value>
      <value>5</value>
      <value>6</value>
    </value>
  </value>
</signal>
```

`<meta>` (*optional*) Variables can have additional information attached. These are not precisely specified and can be arbitrary strings. This has to be defined in the future.

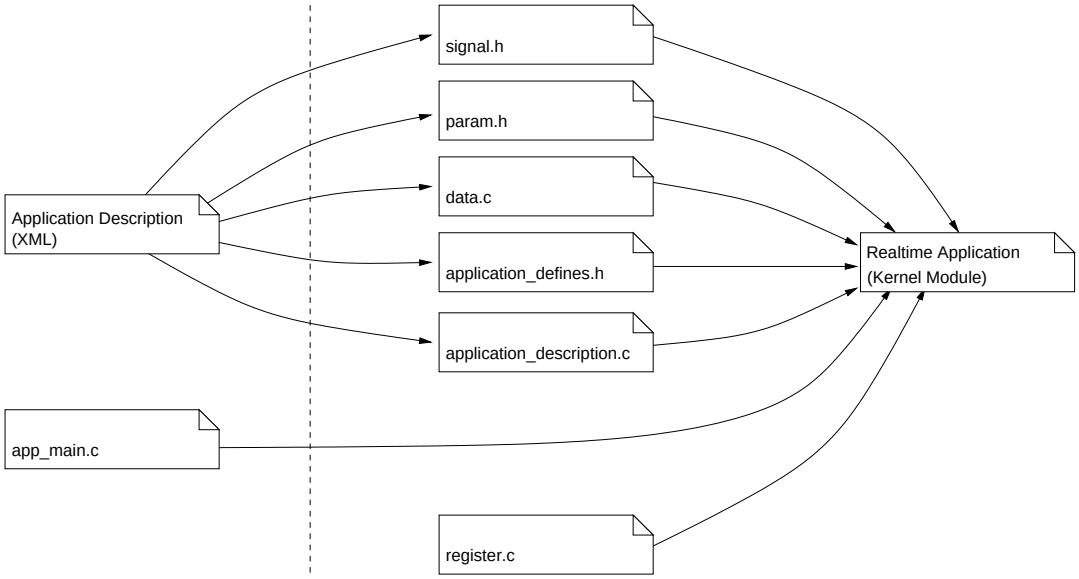

Figure 4.1: Creation of a Realtime Application

5 Simulink Integration

5.1 The Simulink Library

The EtherLab library `etherlab_lib` was designed to allow the creation of EtherLab applications using MATLAB/Simulink models. For example, it contains Simulink blocks for all supported EtherCAT slaves. To show the library window (see figure 5.1), the command below have to be entered in MATLAB:

```
>> etherlab_lib
```

Figure 5.1 shows the window containing the `etherlab_lib` with the EtherCAT blockset. Each block has a configuration dialog, that shows up after double-clicking on the block area. There is a "Help" button in all of the configuration dialogs, that provides detailed help concerning the dialog elements.

Figure 5.2 exemplary shows the configuration dialog for the analog-input EtherCAT slaves of the Beckhoff EL31XX series.

5.2 Creation of EtherLab Applications with Simulink

After creating a new model sheet within Simulink, a few parameters have to be set up (menu *Simulation/Configuration Parameters...*):

- The input field *Realtime Workshop* → *System target file* must contain the file name `etherlab.tlc`. It can be chosen via the *Browse...* button (see figure 5.3).
- The input field *Solver* → *Fixed-step size* must contain the period time of the real time cycle, for example `0.01` meaning 100 Hz (see figure 5.4).

After the model has been parametrized, it can be edited as usual. To use EtherLab blocks, open the `etherlab_lib` (see sec. 5.1) and drag the desired blocks on the new model sheet. When finished, *Ctrl-B* will generate the application code contained in a Linux kernel module.

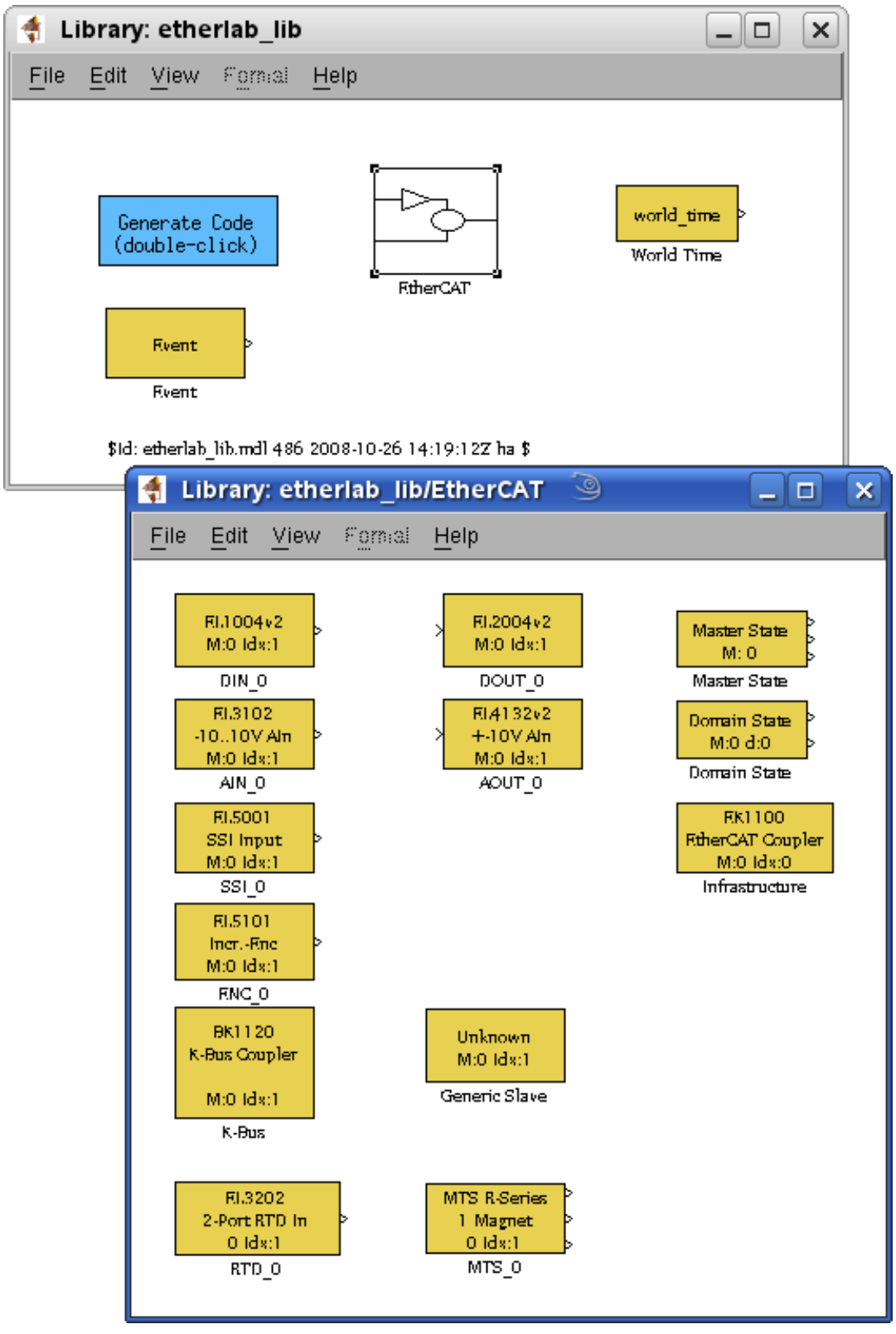

Figure 5.1: EtherLab Simulink Blockset

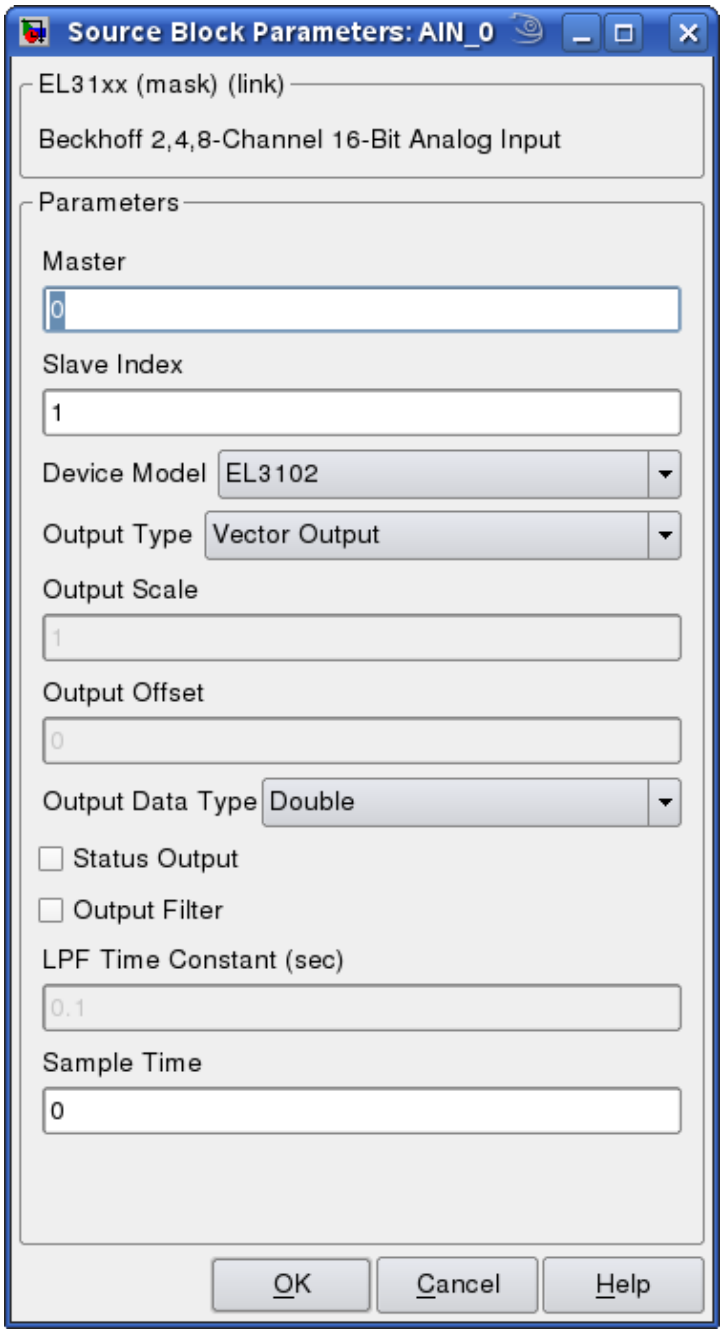

Figure 5.2: EL31xx Configuration Dialog

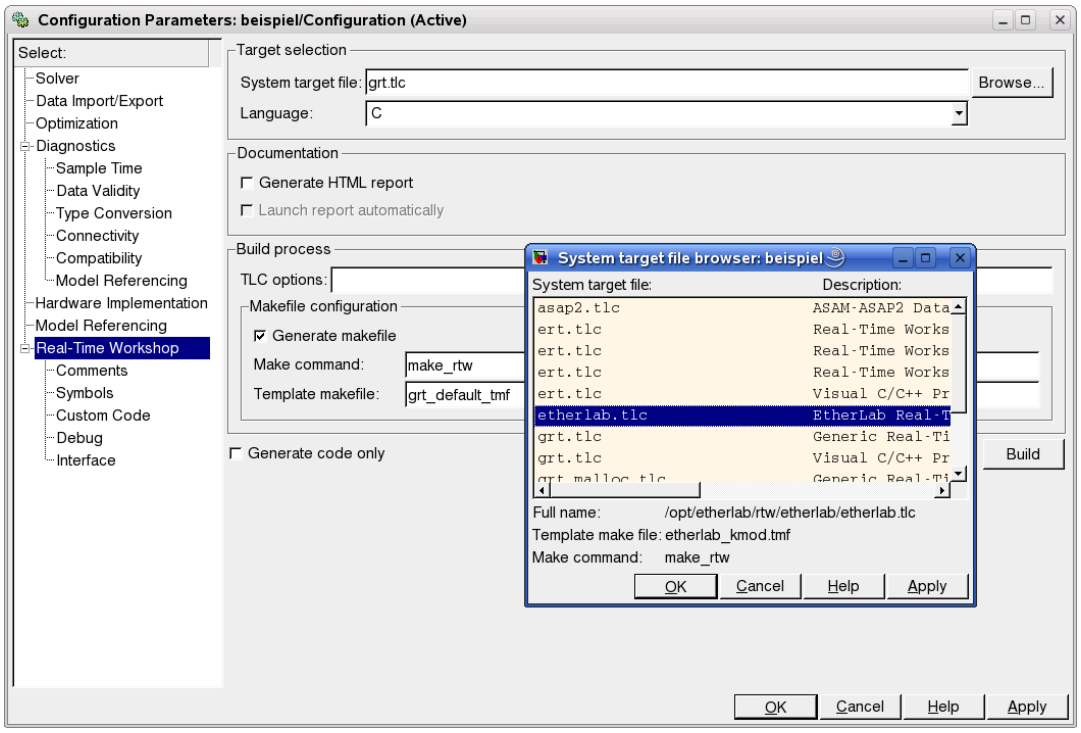

Figure 5.3: Configuration of model parameters

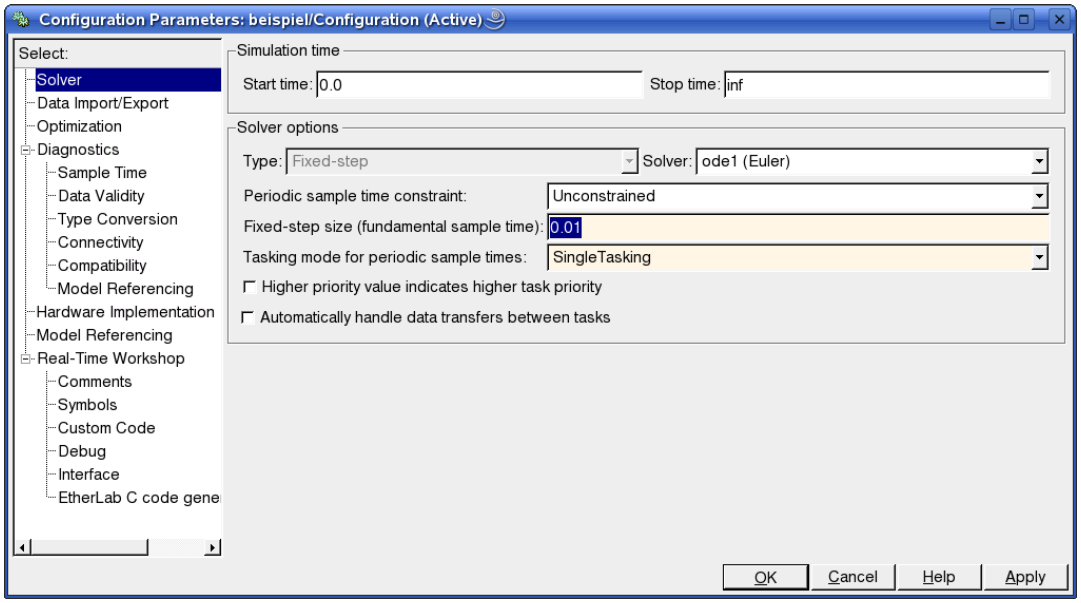

Figure 5.4: Configuration of model parameters (2)

6 Starting Realtime Applications

The application core (`rt_appcore`) must be running before any realtime applications can be started. If it is configured as a service (see section 3.4), this is done automatically at system startup. Otherwise the application core module can be inserted manually. It is recommended to start the application core using the init script, because the necessary device files are then created automatically:

```
# /opt/etherlab/etc/init.d/etherlab start
```

After that, a realtime application can be started by inserting the generated kernel module.

```
# insmod <application>_kmod.ko
```

If the `insmod` command fails, the kernel ring buffer can be analyzed for possible software or hardware misconfiguration:

```
# dmesg | less
```

If the `insmod` command succeeds (no output), the application is running. To access its data from userspace, the buddy process has to be started:

```
# etherlab_buddy
```

The buddy process now acts as a TCP/IP server[1], so the application data can be accessed via the network with EtherLab tools like Testmanager, DLS or the PdCom library (have a look at http://etherlab.org/en/components.php).

[1] The buddy uses an XML-based protocol called MSR; there is a German documentation at http://etherlab.org/download/m-igh_rt_api.pdf