Simulink Target for Real Time Linux Extension: Hardware Control using a Wrapper for Comedi

Klaus Oppermann
Institute for Measurement Technology
Johannes Kepler University, Altenbergerstraße 69, 4040 Linz, Austria
klaus.oppermann@jku.at

Daniel Schleicher
Institute for Measurement Technology
Johannes Kepler University, Altenbergerstraße 69, 4040 Linz, Austria
daniel.schleicher@jku.at

Bernhard G. Zagar
Institute for Measurement Technology
Johannes Kepler University, Altenbergerstraße 69, 4040 Linz, Austria
bernhard.zagar@jku.at

Abstract

A commonly used hardware in–the–loop environment is the dSPACE[1] system, whose strength is aimed at high flexibility and processing speed, but involves rather costly hardware and software licence fees. In our proposed paper we elaborate a scaleable hardware platform running a low–cost RTLinux open source system of comparable processing speed.

There is a great variety of applications that need such rapid control prototyping systems. Especially developers of electromechanical plants are using rapid control prototyping systems to tune less known system parameters and controller settings. These needs are well covered by the software environment MATLAB/Simulink and the "Simulink Target for Real–Time Linux" (STRTL toolbox). STRTL was developed within the Ph.D. thesis of R. M. Garcia [1] on which our system is based upon. Unfortunately STRTL is no longer actively supported and provides only certain drivers for a limited set of hardware in its initial design.

In our work we extended the driver concept of STRTL to the well known driver interface standard comedi[2]. This enables the STRTL to use all the hardware supported by comedi.

In this proposed paper we show the applicability of our package to control an electric drive used in a student lab course aimed at familiarising the students with controller systems.

The paper is organised as follows:

First the requirement for this rapid control prototyping system are detailed. Secondly the comedi–STRTL driver is discussed in detail. Finally we demonstrate the successful application of this extension by means of an electric drive controller. We conclude our paper with some remarks and the reference to our website where this extension can be downloaded freely.

[1] www.dspace.de
[2] linux control and measurement device interface (www.comedi.org).

1 Introduction

Nowadays rapid control prototyping gains more and more importance in the industry. Usually such systems cost a big amount of money for hardware and licence fees (e. g. dSPACE). Especially at universities, where the money is scarce, rapid control prototyping systems based on open source products are quite interesting. Two well known open source products are RTAI–Lab[3] and RTLinux[4] used with STRTL [1].

At our institute we use RTLinux with STRTL because we're familiar with MATLAB/Simulink[5] and its RealTimeWorkshop environment [3]. This allows our students, who are already well trained using MATLAB to learn theoretical knowledge by making experiences with live applications. Therefore we designed a student lab exercise for demonstration purposes.

A problem of RTLinux with STRTL was the insufficient support of data acquisition hardware. Therefore the decision can be made to write a STRTL–driver for each data acquisition card or to implement a driver–interface like comedi[6] in STRTL. We decided to implement comedi in STRTL, for this reason this paper describes the implementation of the comedi–interface into STRTL.

2 The comedi–STRTL–driver

2.1 Overview

The idea of the comedi–STRTL driver is to combine the advantages of the comedi driver interface with the STRTL–Target for MATLAB. The advantages of this driver interface are a wide range of supported hardware and its standardized interface. From there it is easy to write a comedi–driver, if there doesn't exist one for a special hardware. There is also a good description "Writing a comedi driver" [5]. Another helpful source is the Goetz Report [2] which gives a nice STRTL overview. Furthermore hardware which is supported by comedi can also be used in STRTL with the comedi–STRTL driver. In this sense the comedi–STRTL driver is a very useful and flexible AddOn for the STRTL–Target.

2.2 Including the driver into the STRTL toolbox

To include the comedi–STRTL–driver into the STRTL toolbox the STRTL–addon–package has to be copied over the basic STRTL–toolbox. Mention that STRTL–addon–package also includes the procctrl–STRTL–interface written by D. Schleicher [4]. In order to guide this STRTL–modification the most important steps to include the AddOn–package will be repeated below. A running comedi–interface and STRTL–system will be assumed.

1. Copy the files from the directory `\STRTL_M6.5\rtw\c\src\` of the downloaded package into `\rtw\c\src\`.

2. Copy the files from the directory `\rtlinux\` of the AddOn–Package into `\rtlinux\`.

3. Adapt the file `\rtlinux\rtlinux.tmf` so that the system automatically load the comedi–driver (e. g. s526) for your own data acquisition hardware. This is a example for the Sensoray S526 card that we use. The `rtlinux.tmf` of the AddOn–package is prepared for the Sensoray S526 card like you see it in the listing 1.

```
...
load:
@modprobe kcomedilib

# Comedi-Driver Load and Configuration
# adapt these lines
@modprobe s526
@comedi_config /dev/comedi0  s526 0x6c0,0x5
#End of Comedi

@rtlinux start $(PROGRAM)_rtl.o
@./$(PROGRAM)

unload:
@echo "### Unloading modules from the kernel..."
@rtlinux stop $(PROGRAM)_rtl

#adapt this line
@rmmod s526

@rmmod comedi
@rmmod rtl
...
```

Listing 1: extract of `rtlinux.tmf`

When you copy this two directories the package is completely installed on your system.

[3] www.rtai.org
[4] www.realtimelinuxfoundation.org
[5] www.mathworks.com
[6] comedi is an open source project from the authors David Schleef and Frank Mori Hess.

3 The comedi–STRTL–driver

The comedi–STRTL–driver is rather more a wrapper than a real driver, because it provides the comedi interface as a simulink block. The driver comprised three different new main blocks for Simulink.

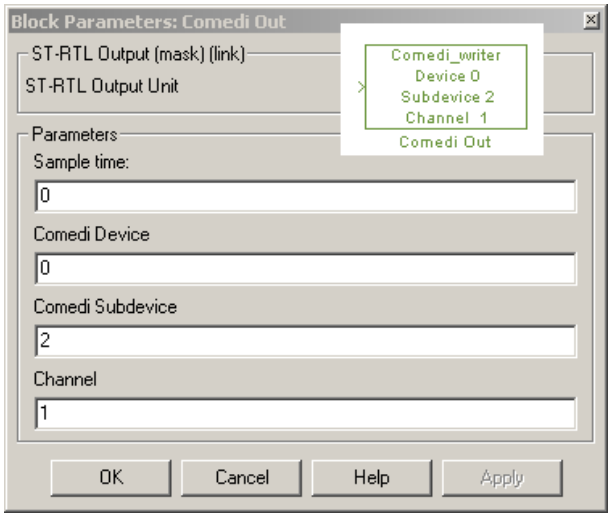

FIGURE 1: *The parameters for comedi are set in the Mask of the comedi–STRTL–driver block.*

These blocks are the:

output block (Comedi Out): writes the value to a specified output channel.

input block (Comedi In): reads the value from a specified input channel.

adjustment block (S526 Direction): defines a group of channels to output or input.

The input and output blocks have the same body to define them. By double clicking on the blocks, a table opens, where the comedi options are defined (see figure 1).
The comedi options are:

Sample time: defines the Sample time of the block, must be the same or a multiple of the Sample time of the model.

device: to select the acquisition device.

subdevice: to select the different functions of the hardware (DIOs, AOs, Counter, ...).

channel: to select the line or channel of the function.

The parameters in the mask are the same like the parameters used by the C–functions of the standard comedi–library. The input or output of the block uses the value type depending on the hardware selected by the comedi–parameters. Normally values like the resolution of the channels (e. g. 16 bit analog: $-32768 \ldots 32768$) or the digital I/O state (0, 1) are used. That's why it is recommended to use a adaptation block based on the applied measurement hardware.

The adjustment block defines a specified I/O–channelgroup to output, when the input of the block is defined to 1. Otherwise the I/O–channelgroup is defined as an input if the input of the block is defined to 0.

```
...
extern void comedi_strtl_write(int device_number,
int subdevice, int channel, real_T value)
{
  /* Lock writing mutex. */
  pthread_mutex_lock(&mutexcomedi_write);
  dev=(char*) dev_sel(device_number);
  device=comedi_open(dev);
  val = value;
  ret=comedi_data_write(device,subdevice,
  channel,0,0,val);
  ret=comedi_close(device);

  /* Unlock writing mutex. */
  pthread_mutex_unlock(&mutexcomedi_write);
}  /* End of comedi_strtl_write(...) */
...
```

Listing 2: extract of the `comedi\_strtl.c`

The secret behind these MATLAB–blocks is only a call of the original comedi C–functions, as you can see in Listing 2. This listing shows only the write–function of the comedi wrapper. But the other functions are similar to the write–function.

With the capabilities of this driver it is very easy to use comedi with STRTL.

4 Demonstration application

To show the capabilities of our AddOn we use a student lab exercise. Originally the students had to build an analog controller for a DC–motor using a tachometer (also a DC–Motor, but used in generator mode) to measure the rotational speed.

This is a typical problem where mechanic, electronic and controlling are linked. Instead of building an analog controller we implement a digital controller to control the rotational speed. The plant is simulated in Simulink and adapted for STRTL. Using the strtl_procctrl driver [4] the parameters can be controlled by a small website. A block schematic is shown in figure 2. The whole plant is shown in figure 3.

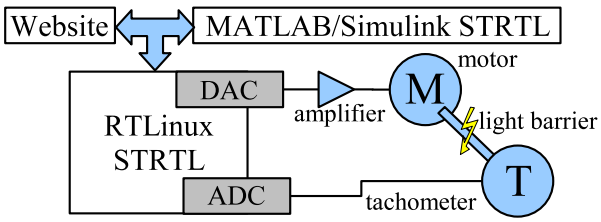

FIGURE 2: *Block schematic of the setup.*

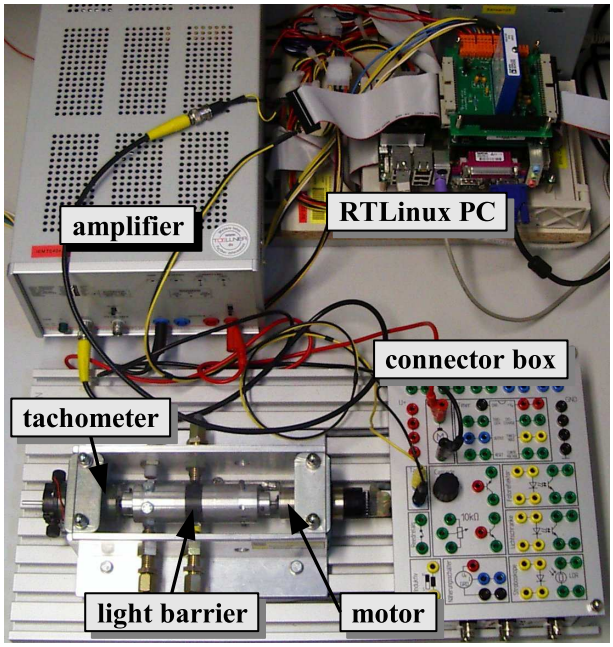

FIGURE 3: *Instead of using an analog controller, the motor is directly connected via a power amplifier to the DAC and the tachometer to the ADC.*

In the following sections the mathematical model of the plant and the implementation is performed. Furthermore the simulation is compared with real measurements acquired with the STRTL–model.

4.1 Mathematical model of the motor

Since the mechanical time constant is much bigger, than the electrical, we approximate the motor by a first order system

$$G(s) = \frac{V}{1+as} = \frac{n(s)}{u(s)}. \tag{1}$$

The tachometer can be approximated by

$$u_A = k \cdot n. \tag{2}$$

If we measure u_A and the rotational speed with a light barrier, we can determine the tachometer constant k with $k = 1000/2.45$ V/min.
A simple PI–controller with a saturated integrator part in the form of

$$R(s) = k_p + k_i \frac{1}{s} \tag{3}$$

is used to regular the system. These equations are implemented in the Simulink–model.

4.2 Implementation

As shown in figure 6 the implementation of the controller is quite simple. With the known plant constants the simulation should represent the real plant.
Using the simulation the parameters of the PI–controller can be adjusted without any damage of the real system. Nevertheless the parameter should be setup rather conservative to mind possible non-linearities which are not in the model of the plant.

To adapt the simulation model the constants are replaced by blocks of the strtl_procctrl driver and the plant is replaced by the comedi input and output blocks. The result is shown in figure 7. Comparing figure 6 and 7 we'll recognise just a few changes between the two models.

Additional to the External Mode in Simulink we can now control the model through the website (see figure 4).
One of the biggest advantage of the strtl_procctrl driver is to have an interface which is independent of MATLAB/Simulink.

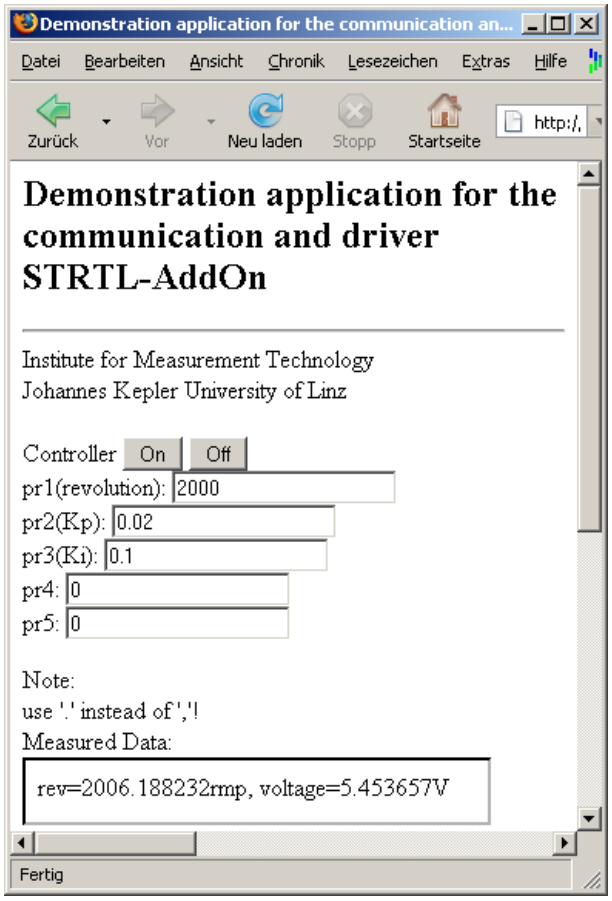

FIGURE 4: *The website is used to change controller parameters an the rotation speed.*

Since we have the problem of the broken pipe when we change a parameter in Simulink during the External mode, we use the website to change the values of the rated speed.
The rate to read values from the Simulink model depends on the webserver and the implementation of the website. In this case we only update one single value each half second and do not use graphs like Simulink does it.
Using the *SaveToWorkspace*–Block in Simulink we can save the rotation speed progression and compare it with the simulation. The comparison is shown in figure 5. We can see that the designed controller works fine on the real plant.

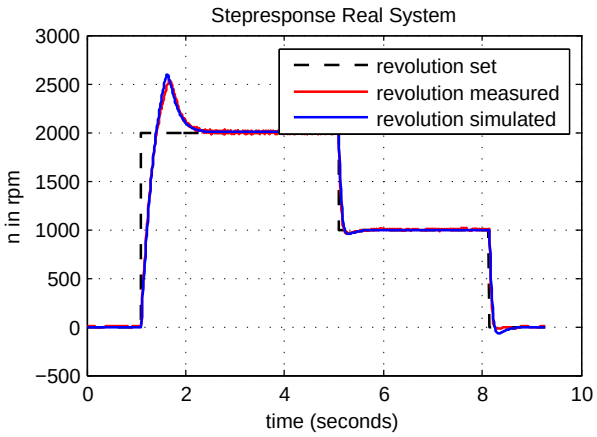

FIGURE 5: *The measured data fits very good with the simulated data.*

5 Conclusions

In this paper a comedi–driver–AddOn for STRTL was presented with a small student lab exercise. Using the comedi–STRTL driver we are able to use comedi featured devices in our Simulink–model. The blocks are easy to use and their usage is quite similar to the C–functions of comedi. With this extension STRTL features the same hardware like comedi.
For a better and stable communication with the embedded system we use the proctrl–driver [4]. MATLAB/Simulink is only used during the rapid control prototyping phase of the project while the Website is used as the user interface which is shipped out to the customer when the product is ready.
The final demonstration with the student lab exercise shows the capabilities of RTLinux, STRTL and the AddOn presented in the paper. Finally, the comedi–STRTL and strtl_procctrl driver [4] can be downloaded from `http://www.emt.uni-linz.ac.at/rtlinux/` as one complete package.

The latest source files for comedi are available at the web page `http://www.comedi.org`.

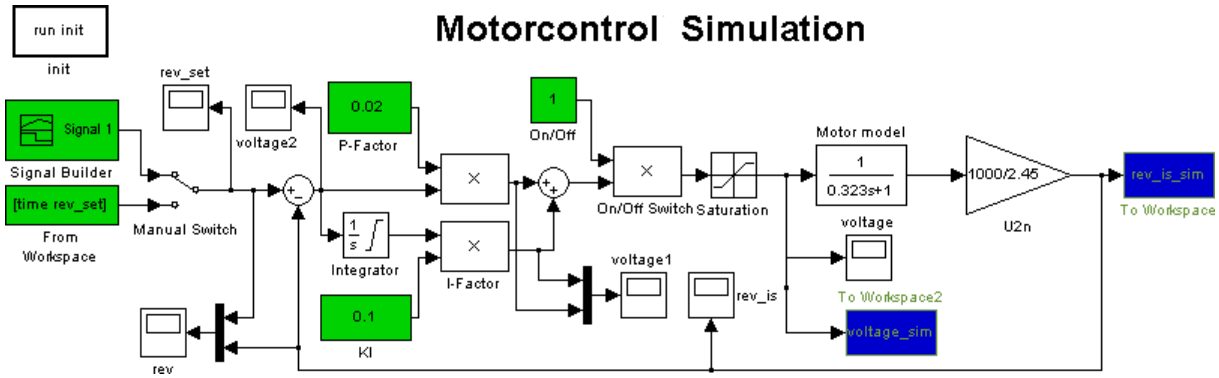

FIGURE 6: *Having all the parameters of the plant, it can easy be simulated in Simulink.*

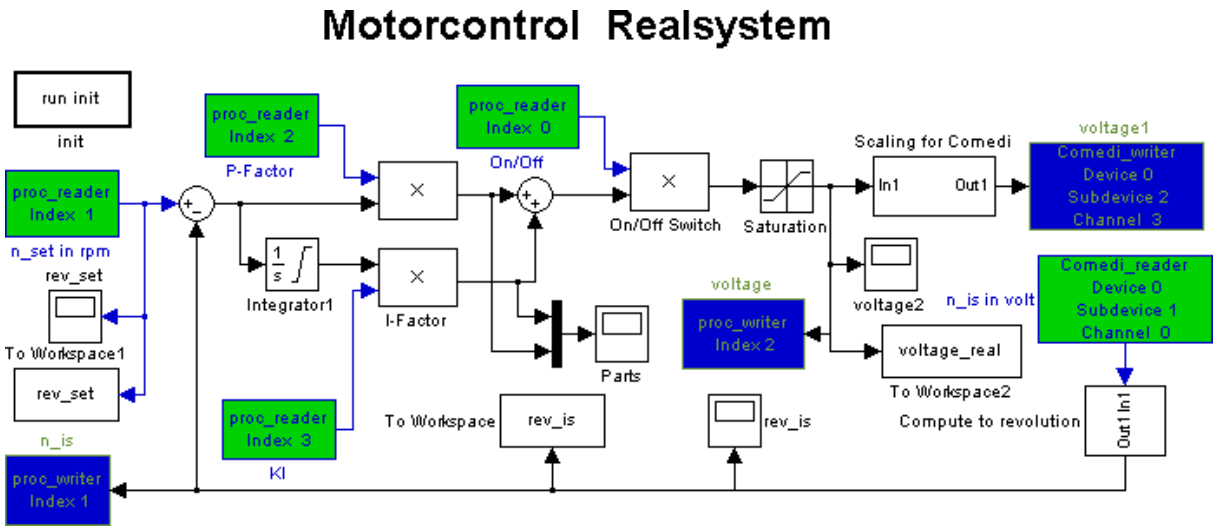

FIGURE 7: *The adaptation for the usage with STRTL only needs a few changes (compare with figure 6).*

References

[1] R. M. Garcia: *Hard Real–Time Control Using Simulink Target for Real–Time Linux*, Glasgow, Caledonian University, IJCA Vol.11 No.2, 2004.

[2] S. Goetz: *Development of Real Time Systems using Simulink–RTW and RTLinux* ,2005.

[3] MathWorks Inc., *Real-Time Workshop User's Guide, for use with SIMULINK*, 1999.

[4] Daniel Schleicher, *Simulink Target for Real Time Linux Extension: Remote Control via Command Line and Web Interface*, 2007.

[5] David Schleef & Frank Mori Hess, *Comedi "Writing a Comedi driver"* `http://www.comedi.org/doc/x1394.html`.