Simulink Target for Real Time Linux Extension: Remote Control via Command Line and Web Interface

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

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

Bernhard G. Zagar
Institute for Measurement Technology
Johannes Kepler University, Altenbergerstr. 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.

Because of a bug in the parameter exchange function of STRTL and the desire to control our Simulink model via a website, we created an extension called strtl_procctrl. This extension is a special driver that enables the user to exchange the MATLAB/Simulink model parameters via the linux build in proc–filesystem between the model and a website. Using this method it is quite easy to create a MATLAB/Simulink independent user interface. This interface can either be a command line application or a website that accesses the driver generated proc–file.

The paper is organised as follows:

First the requirement for this rapid control prototyping system and common drawbacks of the STRTL toolbox are detailed. Secondly the strtl_procctrl driver is discussed in detail. Finally we demonstrate the usage of the driver by means of a small application. We conclude our paper with some remarks and the reference to our website where this extension can be downloaded freely.

[1] www.dspace.de

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 know products are RTAI–Lab[2] and RTLinux[3] used with STRTL.

At our institute we use RTLinux with STRTL for MATLAB 6.5 because we are familiar with MATLAB/Simulink and its RealTimeWorkshop environment [2]. Using this setup during the prototyping phase of an application is quite powerful and leads to good solutions. But offering this solution to the customer isn't really possible, because they would need an easy to use interface and wouldn't like to pay licence fees to Mathworks in general.

During the work with STRTL we identified an error called "broken pipe" while exchanging parameters from Simulink–model within external mode. The development of the strtl_procctrl driver was a workaround for this identified problem, which enables the exchange of parameter through the command line, respectively through a website. Especially the latter brings up an interface which is easy to use and universally applicable.

Another way of exchanging parameters was devised by A. Siro [3] who wrote the C–API. The C–API is an application interface to access and change the STRTL parameters. Unfortunately this package wasn't yet available during our own development phase.

In comparison to the C–API our approach to use a html–website allows quick creation of an user interface. Maybe this two approaches would be combined one day.

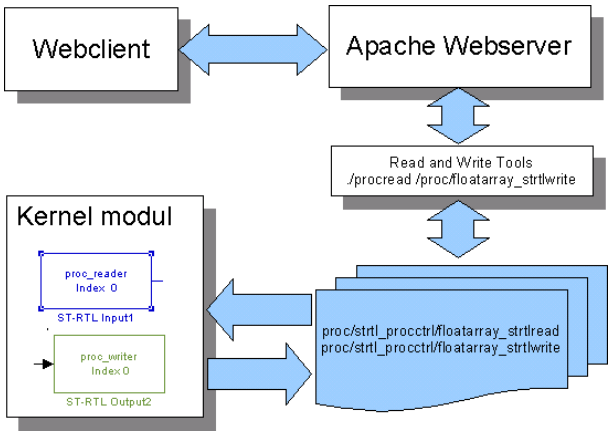

FIGURE 1: *Block diagram of an application using the strtl_procctrl driver*

The block diagram 1 shows the functionality of a STRTL–model using the strtl_procctrl driver. The two proc–files (`floatarray_strtlread` and `floatarray_strtlwrite`) are used to communicate with the STRTL–module which is running on the embedded target system.

The data in the proc–file, written by the kernel module, can be accessed by a webclient through an webserver and two small C–programs, which are explained later in section 2.4.

While the MATLAB/Simulink blocks `proc_reader` and `proc_writer` are updated continuously with every taken sample, the webclient updates its data periodically on demand.

2 The strtl_procctrl driver

2.1 Overview

The basic idea of the strtl_procctrl driver is to use the proc–filesystem of linux to change important settings in the STRTL–model. Well edited information about the proc–filesystem was found in [4].

[2] www.rtai.org
[3] www.realtimelinuxfoundation.org

2.2 Driver source

The proc–files are created during the driver initialisation when `procctrl_init()` is executed. Each proc–file represents one array that is directly accessed by MATLAB/Simulink through the driver. The coherencies between the driver files and their functions is pictured in fig. 2.

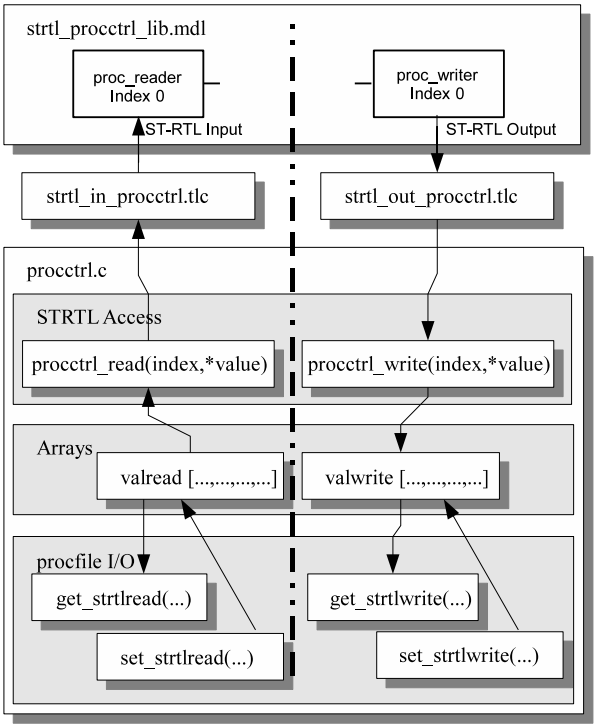

FIGURE 2: *Connections between the files and functions of the driver*

The Simulink blocks of strtl_procctrl_lib.mdl can be placed in the MATLAB/Simulink model with a preferred channel index. The signal value of the proc_writer Simulink block is stored in the `valwrite` array using the `procctrl_write`–function with `valwrite[index%ARRAYLEN]=value`, where `index` is the index parameter and `value` is the signal value. To read the value of the array with the proc_reader block, we use the `procctrl_read`–function with `*value=valread[index%ARRAYLEN]`.
Using this functions, the STRTL–output block is connected to the float array `valwrite[ARRAYLEN]` which is updated each samplestep. That is why the second array `valread[ARRAYLEN]` is needed for the STRTL–input block, otherwise the value of the STRTL–input block would be overwritten each samplestep.

External programs have access to the arrays via the get– and set–functions. Each proc–file has one get– and set–function. The following source shows how the `valread`–array is read by external programs reading the proc–file via standard file–I/O functions.

```
...
// char pointer on the first
// valread value
p=(char*)&valread[0];
for(i=0;i<ARRAYLEN*sizeof(float);i++){
  //print bytes
  size+=sprintf(page+size,"%c",*p);
  p++;
}
...
```

Because of the implementation of the proc–files, the float value is transmitted bytewise using a pointer (see fig. 3).

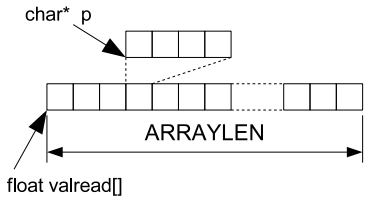

FIGURE 3: *Using a char pointer on the float array for the proc interface*

These bytes have to be merged to a float value again when an external program writes on the proc–file. This is done in the `set_strtlread` function shown below:

```
// startpointer
p=(char*) &valread[0];
for (i=0; i < count; i++){
  //save user_buffer values
  *p=user_buffer[i];
  p++;// pointer inc
}
...
```

The functions `set_strtlread` and `get_strtlread` belong to the proc–file floatarray_strtlread. The functions for the proc–file floatarray_strtlwrite, which is accessed by the functions `set_strtlwrite` and `get_strtlwrite`, are working in the same manner.

2.3 Including the driver into the STRTL toolbox

To include the strtl_procctrl driver into the STRTL toolbox it's only necessary to follow the guideline written by R. M. Garcia [1] (look at README_IOCard.txt contained by the STRTL toolbox). Similar information about drivers for STRTL can be found in [5]. Nevertheless we'll repeat the most important steps to include the strtl_procctrl driver. A running STRTL system will be assumed.

1. Copy the files procctrl.h and procctrl.c into `\rtw\c\src\`.
2. Copy strtl_procctrl.mdl, strtl_in_procctrl.m, strtl_in_procctrl.tlc, strtl_out_procctrl.m and strtl_out_procctrl.tlc into `\rtlinux\`.
3. Include the strtl_procctrl.mdl blocks into strtl_lib.mdl.

4. Add the driver name in rtlinux.tmf (DAC = procctrl.c)
5. Modify krnl_main.c
   - Add `#include "procctrl.h"`
   - Add `if (procctrl_init() == -ENODEV) PRINT_W("\nprocctrl didn't initialize\n");` in `init_module(void)`
   - Add `procctrl_close();` in `cleanup_module(void)`

Instead of doing all this changes by yourself, it might be easier to download the modified package from our website (`http://www.emt.uni-linz.ac.at/rtlinux/`).

2.4 Driver interface via command line

To exchange parameters with a Simulink model using the strtl_proccrtl driver we need to write at `/proc/strtl_procctrl/floatarray_strtlread` or read from `/proc/strtl_procctrl/floatarray_strtlwrite`.

Due to minimize the processing time of the driver, the output–function doesn't make any conversation from hex to string or similar of the array. The values are submitted directly as `char` and are not converted to a string. Because of this manner, reading the proc–file with `cat <procfiletoread>` will only provide nonsense and should not be used. Therefore some small C–programs which do the conversations in the userspace were written. These read and write tools can be used to get understandable results from the proc–file and are listed below:

- `pr_fa` ... ProcRead FloatArray
  `./pr_fa <procfiletorread>`
- `pw_fa` ... ProcWrite FloatArray
  `./pw_fa <procfiletorwrite>`
- `pr_faind` ... ProcRead FloatArray on Index
  `./pr_faind <procfiletorread> <index>`
- `pw_faind` ... ProcWrite FloatArray on Index
  `./pw_faind <procfiletowrite> <index> <value>`

The pw_fa C–program is shown in the following listing:

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
// look at procctrl.c
#define ARRAYLEN 64
int main(int argc, char** argv)
{
  FILE* fp; char* arg; float* input=NULL;
  int i,k,index,x; float res;
  if ( argc < 2 ) {
   printf("pw_fa ProcWrite Floatarray!!\n");
   printf("Usage: ./pw_fa <procfiletorwrite>  \n");
   printf("Usage: ./pw_fa /proc/procfile 3.4 7.3 \n");
   return 0;
  }
  // determine nr of values
  x = argc - 2;
  input = (float*)malloc(x*sizeof(float));
  for ( k = 0; k < x; k++ ) { // parse input
   sscanf(argv[2+k],"%f",&res);
   input[k] = (float)res;
  }
  fp=fopen(argv[1],"w");
  //transferring the data as float
  fwrite(input,sizeof(float),x,fp); t
  free(input);
  fclose(fp);
  return 0;
}
```

In the following an example for writing the values (1.23,5.55,6.66) into `/proc/strtl_procctrl/floatarray_strtlread` is given.
The following line is typed on the console:

```
root#>./pr_fa /proc/strtl_procctrl/float...
            ...array_strtlread 1.23 5.55 6.66
```

Looking at the listing above, it is quite easy to transfer data to the proc–file, since standard file–I/O functions are used. The main part is parsing the command line parameters.

2.5 Driver interface via website

To control the model via website we only need to execute our programs to write on the proc–files. Therefore we've to take care that our programs and files have the correct permissions to be assure the usability via the webserver.

The following source is an extract of such a website, where a float number (pr2) is written on index 2. When the page is loaded the first time, pr2 is initialised with 0. After changing this parameter the `system()`–function is executed with the appropriate options.

The usage of the PHP internal write–functions doesn't work because of the PHP automatic typecastig. An other way would be the usage of cgi–scipts, where the c–write functions can be directly implemented.

```
<?php // set global variables
// files
$pf_read='/proc/strtl_procctrl/floatarray_strtlread';
$pf_write='/proc/strtl_procctrl/floatarray_strtlwrite';
// commands
$pw_faind='./c-files/pw_faind';
?>
<form name="number"  method="GET" action="index.html">
(pr2) A:
<input type="text" name="pr2" value="
<?php
```

```
  if(isset($_GET[pr2])){
   echo "$_GET[pr2]";
   }else{
   echo "0"; // default value
}
?>"
action="index.html">
<?php
  if (isset($_GET[pr2])){
   system($pw_faind.' '.$pf_read.' 2
   '.$_GET[pr2]   , $ret);
 }
?><br>
</form>
```

Of course, for the real homepage source, this code will be formatted to use only two lines. A demonstration will be shown in the following chapter.

3 Demonstration application

To demonstrate the capabilities of our extensions we build a small calculator application. Fig. 4 shows the MATLAB/Simulink model. Instead of constant blocks the proc_reader block is used. Instead of a display block the proc_writer block is used.

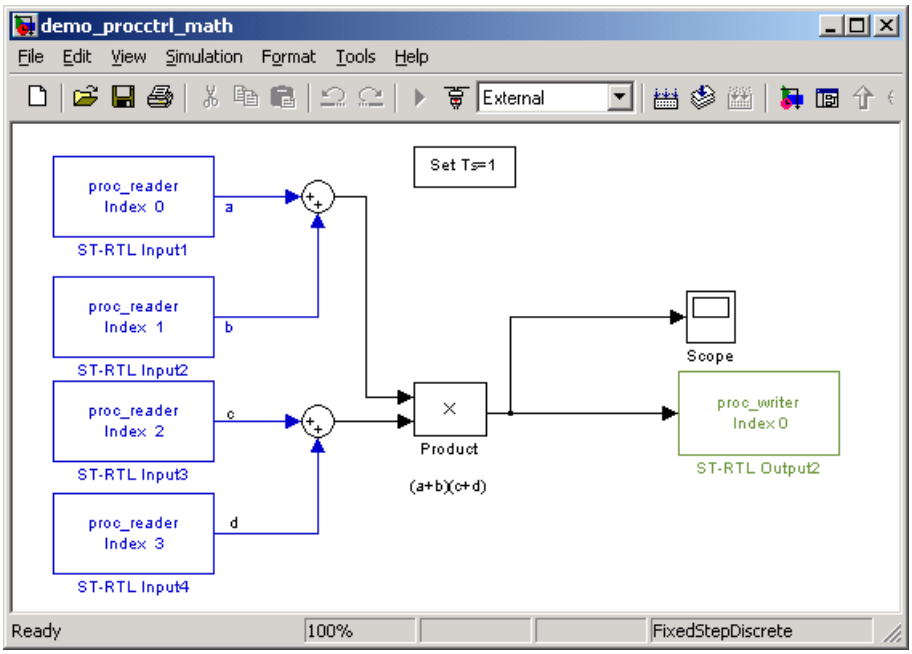

FIGURE 4: *MATLAB/Simulink model using the strtl_procctrl driver for a simple calculation*

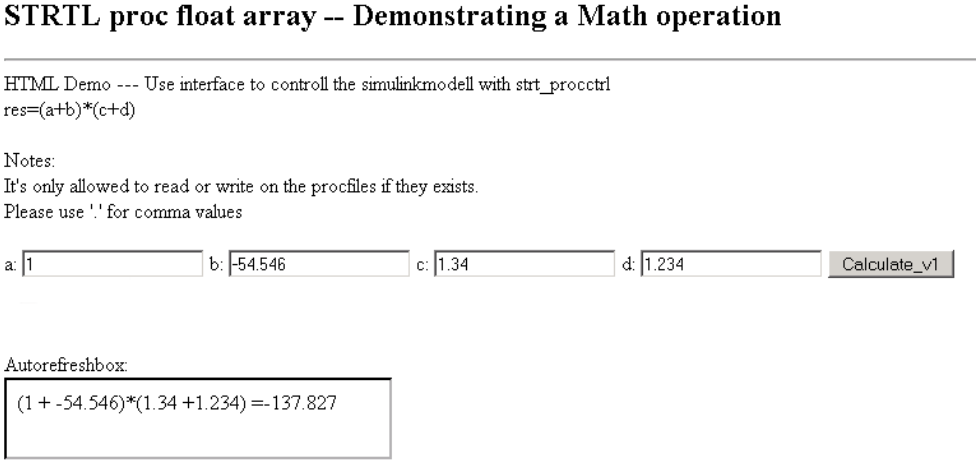

FIGURE 5: *The values for the calculation are inserted via html form*

Mention that the calculation is done in real time each sample, the parameter exchange does not. Fig. 4 shows the web interface which uses the PHP code snipplets shown in section. 2.5. The autorefreshbox is updated by the webbrowser continuously.

4 Conclusions

In this paper a remote control for STRTL was presented, that was demonstrated to work nicely with a small MATLAB/Simulink model. It enables a quick communication with the kernel module completely independent from MATLAB/Simulink.

MATLAB/Simulink is only used during the rapid control prototyping phase of the project while the website provides the user interface which is shipped out to the customer when the product is ready.

Of course a website is not mandatoryly needed, because the proc–files can also be read and written by a simple C–program. The appearance of the user interface is still very flexible which is a result of the usage of the proc–filesystem. This enables the developer to create his own userinterface using standard file–I/O functions.

The final simple and easy to understand Simulink model demonstrates nicely the capabilities of RTLinux, STRTL and our remote control extension. A much more practical demonstration is given in [6], where we combined this approach with a student lab exercise.
Finally, the strtl_procctrl driver can be downloaded from `http://www.emt.uni-linz.ac.at/rtlinux/`.

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] The MathWorks, Inc., *Real–Time Workshop User's Guide for use with SIMULINK*, 1999.

[3] A. Siro, I. Diaz: *Introducing the C–API Simulink Target for RT–Linux*, 2005.

[4] J. Quade, E. K. Kunst: *Linux–Treiber entwickeln*, dpunk.verlag, 2004.

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

[6] K. Oppermann, D. Schleicher, B. G. Zagar: *Simulink Target for Real Time Linux Extension: Hardware Control using a Wrapper for Comedi*, 2007.