Scicos as an alternative for Simulink

M.G.J.M. Maassen

DCT 2006.054

Migrating to from Simulink to Scicos with respect to real time programs

Supervisor: M.J.G. vd Molengraft

Technische Universiteit Eindhoven
Department Mechanical Engineering
Dynamics and Control Technology Group

Eindhoven, May, 2006
## Contents

1. Introduction ........................................... 1

2. Installation .............................................. 2
   2.1. Introduction ........................................ 2
   2.2. Install Knoppix to the hard disk ............... 2
       2.2.1. General notes/hints regarding Knoppix .... 3
   2.3. Installation steps in Linux ..................... 3
       2.3.1. Mesa (display drivers) ....................... 3
       2.3.2. EFLTK ......................................... 4
       2.3.3. A new patched kernel ....................... 4
       2.3.4. Rtai ........................................... 5
       2.3.5. Scilab (source version) ..................... 5
       2.3.6. Installing Scilab/Scicos & Rtai add-ons .... 6

3. Using Scilab and Scicos ................................ 7
   3.1. Differences compared to Matlab .................. 7
   3.2. Scilab, generals hints and tips ................. 7
   3.3. Scicos ............................................. 8
       3.3.1. General hints and tips ...................... 8
       3.3.2. An example of working with Scicos ........ 10
       3.3.3. Notes ......................................... 12
   3.4. Creating and running real time programs ........ 12
       3.4.1. Building real time code ..................... 12
       3.4.2. Loading modules ............................. 13
       3.4.3. Running applications ....................... 13
       3.4.4. Xrtailab ..................................... 14

4. Writing new blocks .................................... 16
   4.1. Block and code structure ....................... 16
   4.2. Steps needed to implement a new block ........ 17
       4.2.1. Customize the C code ....................... 18
       4.2.2. Explanation of new block template ......... 20
   4.3. Example of converting existing Matlab S-function, implementation of TUEDACs ..... 21
       4.3.1. Modify the main file ......................... 21
       4.3.2. Modify the code generation file: .......... 23
       4.3.3. Building the C code ......................... 23
       4.3.4. Create the Scicos script .................... 27
   4.4. Building the pato01 model ...................... 27
       4.4.1. Example of translating other blocks, DCT PD and Ref3 ... 28
       4.4.2. Not explained errors ....................... 28
4.5. Customizing blocks 29
  4.5.1. Change appearance of blocks 29
  4.5.2. Xrtailab labels 30

5. Results 31
  5.1. The pato01 model 31
  5.2. Testing the pato01 model 32
    5.2.1. Measurement on a dynamic system 32
    5.2.2. Sample time errors 36
    5.2.3. CPU usage 38

6. Conclusions 39

7. References 40

Appendix A: errors and possible solutions A.1
  Installation errors A.1
    Kernel (make menuconfig) A.1
    Kernel (make) A.1
    Kernel (restarting using the new kernel) A.3
    RTAI (make) A.3
    Scilab (./configure) A.4
    Scilab (make all) A.4
    Rtailab (make install) A.5
  General errors A.6
    Starting an xwindow application (xlib) A.6
    Running Scilab A.6
    Using a universal Actuator/Sensor block (Scicos) A.6
    Building a real time program (RTAI CodGen) A.7
    Running a real time program A.7

Appendix B: software download locations B.1

Appendix C: Scicos block template C.1

Appendix D: source codes examples D.1
  Translated td_outports.c file D.1
  Modifying the td_outports.sci script D.4
  Generated functions of the DCT PD and the Ref3 block D.9

Appendix E: contents of CD E.1
1. Introduction

Matlab is a well known computational software package with a lot of toolboxes, which can be used for many applications. It also contains Simulink and combined with the Real Time Workshop (RTW) it offers functionality to control real time processes. On the other hand, the disadvantage of the Matlab package is its price; it is very expensive. For that reason Scilab, an open source variant of Matlab, might be a good alternative. Scicos, a module of the Scilab package, offers similar functionality as Simulink.

In this report migrating from Simulink to Scicos with respect to real time applications will be discussed.

Currently the TUe Data Acquisition & Control System (the TUeDACS) is supported by Simulink RTW. This implementation allows users to use a block in Simulink representing the in- or output ports of the TUeDACS devices. By means of the real time workshop it is possible to build a real time program from a Simulink model containing the TUeDACS blocks. Such program can measure (sensor) inputs and control the target system using (actuator) outputs and control rules. In Scicos, also real time programs can be built.

The goal of this project is to explore implementation possibilities for the TUeDACS devices in Scicos. As Scicos offers similar functionality as Simulink it might be possible to use Scicos for building the same kind of real time programs. For this exploration the pato01 setup is used as a case.

Linux will be used as operation system because of its excellent real time properties, especially when combined with the Real Time Application Interface (RTAI) extension.

This report contains three sections. The first section will describe the complete installation. Knowledge of Linux is not required; when everything is properly installed it is fairly self-evident. The TUeDACS disk with a working Linux distribution will be the starting point.

In the second section, the usage of the new Scilab package will be explained. Knowledge of Matlab and Simulink will be assumed.

In the last part describes how new Scicos blocks are programmed by translating existing Simulink blocks. These new blocks, including the TUeDACS blocks, are needed when trying to build the pato01 setup in Scicos.
2. Installation

2.1. Introduction

Before Scicos will run in combination with RTAI it is necessary to execute some installation steps.
Here the installation as written down in [1] and adapted to the TUeDACS CD (Linux Live DVD for TUeDACS QAD/AQI devices) version 3.1 will be described. This is just a short list of the commands needed to execute when using the same CD. See [1] for more details. Of course these installation steps can be performed in combination with a different Linux distribution version but then (different) errors will occur. For easy reference the errors that may occur using the TUDACS CD and the possible solutions are described in appendix A.

The user has to pass three significant installation steps. First, the Knoppix cd has to be installed to the hard disk. This may take some time. Then, in order to build and run real time programs after installing some drivers, two major installations steps in Linux have to be executed.
In Scicos the user can build real time programs. These programs make use of the Real Time Application Interface (RTAI). In order to build and load the RTAI modules a new patched kernel must be installed. This patch comes with RTAI. So the next step is to build a new kernel. This also can take a while and may not work on the first try. The last step is to install RTAI and of course Scilab.

Before starting, notice the difference in text formatting:
This format means a command to execute. Just type these commands exactly in a console window.
This format displays the name of a file, path, program or option. It can be seen as a quote.
At last, this format stands for output code printed on the screen. It is generally used in appendix A to reflect an error code as seen in the console window.

2.2. Install Knoppix to the hard disk

To install Knoppix to the hard disk make a Linux Swap, Ext3 and Fat32 partition. In Windows this can be achieved using a program like PartitionMagic. In Linux qparted, a PartitionMagic clone, can be used.
The Fat 32 partition is useful for exchanging files between both operating systems. Knoppix will be installed on the Ext3 (reformatted with the Reiserfs files system) partition.
This is a list of commands needed to execute when using the same CD:

- Start Knoppix from the TUeDACS CD.
- Open Konsole (Terminal Program) and type:
  - `sudo knoppix-installer`
Wait a moment till the program starts and configure a new installation:

- Choose the system: beginner: Multi-User System with hw-detection
- Make a new user account and choose a short administration password because this password is needed more often.
- Install grub to the mbr: Master Boot Record.
- Start the installation and make a boot diskette (might be helpful in case the system will not start anymore because the startup file is messed up).
- Restart the computer without the cd (start the hard disk copy of Knoppix)

2.2.1. General notes/hints regarding Knoppix

In Linux almost every file and folder is protected and can not be modified without “root” privileges. Also operations like loading RTAI modules can not be preformed without those privileges.

In Knoppix it is not possible to login as root instead of normal user. Therefore the administration password is used often. In a console window type su (super user) and enter the administration password to get root privileges. To modify protected files use the File Manager open File Manager – Super User Mode (K-menu > System > More Applications > File Manager – Super User Mode)

A lot of USB devices such as mouses and memory sticks are supported in Knoppix. Attach those devices before starting the computer otherwise they will not be automatically recognized.

Sometimes internet is not available even though the network cable is plugged in. To activate the internet connection open the Network card configuration program (K-menu > KNOPPIX > Network/Internet > Network card configuration).

Note: execute all commands in this document as root (super user) unless told otherwise.

2.3. Installation steps in Linux

2.3.1. Mesa (display drivers)

- Download MesaLib-6.3.2.tar.gz and untar it to a temporary directory. In appendix B the download locations of all source files are listened. These files are also available on the enclosed CD (see appendix E for the contents of the cd).
- Open Konsole and enter su and the administration password. Then go to the Mesa directory (cd /tmp/Mesa-6.3.2).
- make linux-x86-static
- make install (enter-enter-enter)
2.3.2. EFLTK

- Download (see appendix B) efltk-2.0.5.tar.bz2 and also untar it to a temporary directory.
- `cd /tmp/efltk`
- `.efltk-config.in --prefix=/usr/local --multithread`
- When the message: `bash: ./efltk-config.in: Permission denied` appears:
  - Open File Manager – Super User Mode, go to /tmp/efltk and open the properties of efltk-config.in. Open the permissions tab, change all permissions to Can Read & Write, check Is Executable and try again.
- `./emake`
- `./emake install`
- Add `/usr/local/lib` to the file `/etc/ld.so.conf` (already done when using the TUEDAX cd) and run `ldconfig` to update the database.

2.3.3. A new patched kernel

- Download `linux-2.6.10.tar.bz2` and untar it to `/usr/src/linux-2.6.10-rtai-3.2`.
- Download `rtai-3.2.tar.tar` and untar it to `/usr/src/rtai-3.2`.
  - Instead of the file `rtai-3.2.tar.tar` it is also possible to use the rtai file from the cd, in this file the TUEDACS implementation is already included (see appendix E).
- Make symbolic links:
  - `cd /usr/src`
  - `rm linux`
    - `rm: remove symbolic link `linux'? y`
  - `ln -s /usr/src/linux-2.6.10-rtai-3.2 linux`
  - `ln -s /usr/src/rtai-3.2 rtai`

- Patch the kernel:
  - `cd /usr/src/linux`
  - `patch -p1 < /usr/src/rtai/base/arch/i386/patches/hal-linux-2.6.10-i386-r9.patch`

- Configure the kernel.
  - To configure the kernel there are two options: it is possible to make a new config file or change an old (working) config file. For more information about making a new config file or kernel options see appendix A of [1].
  - Here the old config file that comes with the TUEDAX cd is used and some changes were made (just type `make menuconfig` to make a new config file):
    - Copy the old config file:
      - `cp /usr/src/linux-2.6.11-fusion-0.7/.config /usr/src/linux-2.6.10-rtai-3.2` (hidden file)
    - `make menuconfig` (possible error, see appendix A)
  - Change the following options:
    - Change General setup >> Local version to -rtai-3.2
    - Exclude Processor type and features >> Symmetric multi-processing support
- Include Processor type and features >> Preemptible Kernel
- Exclude Device Drivers >> Parallel Port support (needed option for direct i/o access to the parallel port when making the i/o led example from [1])
- Exclude Device Drivers >> Multi-device support
- Exclude Device Drivers >> Fusion MPT device support
- Exclude Power management options >> APM
- Include Power management options >> ACPI >> needed functions (for a laptop include AC Adapter, Battery, Button, Fan, Processor, Thermal Zone and other needed features).
- Include File systems >> Reiserfs support
- Save and exit

- `make` (possible errors, see appendix A)
- `make modules_install`
- `make install`
- Add the new kernel to `/boot/grub/menu.lst` by example:
  ```
  title Debian GNU/Linux, kernel 2.6.10-rtai-3.2, Default
  root (hd0,6)
  kernel /boot/vmlinuz-2.6.10-rtai-3.2 ro ramdisk_size=100000
  init=/etc/init lang=us acpi=off noapic noapm xmodule=fbdev nomce quiet
  vga=791
  ```
- Reboot using the new kernel

### 2.3.4. Rtai

- `cd /usr/src/rtai`
- `make menuconfig`
  - Change General >> Installation directory to `/usr/realtime-rtai-3.2`
  - Include RTAI-Lab
- Save and exit
- `make` (possible error, see appendix A)
- `make install`
- Open `/etc/profile` and add `/usr/realtime-rtai-3.2/bin` to the PATH variable.

### 2.3.5. Scilab (source version)

**Important:** do not use the binary version of Scilab because this version does not contain library files needed by the RTAI-lab automatic code generation.

If Tls and Tk are not installed or the libraries are missing, first (re)install those (in this case the configure script will show an error, for more information see appendix A, section Scilab (`./configure`)).

- Download `scilab-3.1.1.tar.tar` and untar it to `/usr/local/scilab-3.1.1`
- `cd /usr/local/scilab-3.1.1`
- `/configure`
- `make all` (possible errors, see appendix A)
2.3.6. Installing Scilab/Scicos & Rtai add-ons

- `cd /usr/src/rtai/rtai-lab/scilab/macros`
- Open `/usr/src/rtai/rtai-lab/macros/Makefile` and edit the line:
  `SCILAB_DIR = /usr/local/scilab-3.1.1`
- `make install`
- Create the Scilab startup file:
  - `exit` (exit super user mode, the `make user` command must be run as normal user)
  - `cd /usr/src/rtai/rtai-lab/scilab/macros`
  - `make user`
- Create a link to `/usr/local/Scilab-3.1.1/bin/scilab` in `/usr/local/bin`
  (`ln -s /usr/local/scilab-3.1.1/bin/scilab /usr/local/bin/scilab`).

**Important:** always start Scilab as a super user (using the File Manager – Super User Mode or type `scilab` as super user in a console window).

To open Scicos type `scicos` in the Scilab window. The RTAI button (for code generation) and palette (for real time blocks) will be visible.

Otherwise, when Scilab is started as normal user (using a desktop icon for example) the RTAI add-ons will not be available. These add-ons will be visible when Scilab is started as normal user by moving or copying the (hidden) startup file `/home/<username>/.scilab` to `/home/<username>/Scilab/scilab-3.1.1/.scilab`. Now the RTAI add-ons will be loaded when Scilab is started as normal user, only this makes no sense because the RTAI code generation script needs super user authority to create and compile files.

Remark: even when the program is started as super user using the console, the RTAI button is occasionally still invisible. In this case use the file manager to start Scilab.
3. Using Scilab and Scicos

3.1. Differences compared to Matlab

When starting Scilab, the first thing that strikes the most compared to Matlab is the absence of a few windows. Scilab only has one command window. The workspace, working directory, launch path and so on are not present. Also, the first impression of Scicos makes the user think Scicos is not very user friendly.

When taking a closer look, Scilab basically works the same as the user was used to in Matlab. Only some functions are renamed or work differently and some functions are missing. Furthermore, the structure of the script files (m files in Matlab and sci files in Scilab) is different. In Scicos a sci file can contain one or more functions and those functions have to be loaded in the command window before they can be used. For more information and to see a list of differences read the Matlab section of the Scilab manual or read one of the Scilab for Matlab users documents commonly available from the internet. Furthermore, in these documents some syntax differences are explained.

To conclude, the Scilab package is less extensive, help is limited and the user interface is not as good-looking and user-friendly as in Matlab.

3.2. Scilab, generals hints and tips

This report will not explain in depth the usage of the Scilab program itself and how to make scripts. To get started only some hints are given, use the manual to get more information:

- Both in Scilab and in Matlab a graphical window is available to show variables just like the workspace window. Type `browsevar` to show this window. Note that this window does not automatically refresh itself and that not all variables are shown (see options to change this ignore list). The command `who` also shows a list of variables.

- In Matlab the current working directory is visible on the toolbar. In Scilab not; there are two functions to find and change this directory:
  - To find it use the command `pwd` (it stands for print working directory).
  - To change it use the command `chdir(<new directory>)`.

- To get some help on a certain function, type `help <function name>` just like in Matlab. The Scilab help will be opened in a separate help browser instead of printed on the screen.

If an error message regarding missing help files is given it is possible to manually download these help files from the Scilab website (`man-eng-scilab-3.1.1`). First rename or (re)move the directory `/usr/local/scilab-3.1.1/man/eng`. Secondly untar
the new help files to /usr/local/scilab-3.1.1/man/eng (or place the files in another
directory and make a link to it named eng in the folder /usr/local/scilab-
3.1.1/man).

- To translate Matlab m-files Scilab has a special utility named m-file2sci. In
  Scilab, type mfile2sci to translate a single file or a whole directory. In the Scilab
  script editor it is also possible to open and translate a Matlab file. Note that not all
  functions are translated; open the new sci file to see the translation errors and
  warnings. See the help for more information, also look to m2scideclare. With this
  the user can help the m-file2sci script to find out of what kind of type a variable is
  supposed to be.

- Just two more things to get used to. In contrast to other applications in
  Scilab/Scicos the mouse has to be hold above a text field, for example a parameter
  dialog or the main command prompt, when typing in it. Also a scrollbar works
  differently, use the left and right mouse button to scroll up and down.

- To learn more about working in Scilab, look at the demo’s or read one of the
  online tutorials. More information about the application of Scilab and Scicos is
  also available in a recent book, Modeling and Simulation in Scilab/Scicos, written
  by Stephen L. Campbell, Jean-Philippe Chancelier and Ramine Nikoukhah

3.3. **Scicos**

3.3.1. General hints and tips

As already mentioned the Scilab package also contains a Simulink-like tool, named
Scicos. To open Scicos type scicos in the Scilab main window.

The Scicos user interface is not as intuitive as Simulink’s. At the start it takes some
time to understand the behaviour of several mouse operations because they are
different from Simulink. In contrast to Simulink, Scicos has different "mouse modes".
For instance when moving a block and then clicking on another block that other block
will move, too. Mind the useful white hint line on top of the Scicos diagram. This line
shows which mouse operation is selected.

To perform a concrete operation, right click on a block to show a menu containing the
possible operations. Instead of using this menu each time a different operation has to
be preformed one can also save a lot of time by using the shortcuts instead of the
menu to select an operation. To view the currently defined shortcuts go back to Scilab
using Calc option in the Misc menu. Now, type in the Scilab command window:

%scicos_short

The following list of (default) defined shortcuts will appear:
To go back to Scicos use the *return* command. Choose *Shortcuts* in the Scicos menu *Misc* to define a new shortcut. For instance after selecting *Mics > Shortcuts*, click *Object > Identification* and give a letter *y* for example.

In Scicos there are two types of block input/outputs and signals. The black in and outputs are data signals just as in Simulink. Instead of Simulink some Scicos blocks have another type of input or output signal. These red signals are activations signals. Blocks with an activation input are triggered each time an activation signal is send over the activation line. All real time sources, sinks, in- or outputs blocks in the RTAI-lib palette are triggered by using this red activation signal. It is needed to have a Scicos diagram where a red activation clock is connected to a super block as shown in figure 1 to make a RTAI standalone real time program. In this super block other (super) blocks are placed. To generate the code simply select *RTAI CodGen* in the *RTAI* menu and click on the super block.

![Figure 1: base diagram](image)
### 3.3.2. An example of working with Scicos

Several options and commands will be explained while showing a simple example diagram. This example explains how to use Scicos simple and fast. Of course this is the author’s personal approach and one can prefer another way of working.

Example:

Open Scicos.

To place blocks inside the new diagram open a palette using the context or edit menu. Click on a block in the palette, release the mouse button and click somewhere in the new model. Drag and drop as in Simulink will not work.

From the sources palette copy the red clock to the diagram. Also copy a sine, scope (both in the RTAI-lib palette) and gain (Linear palette) block to the project as shown in figure 2. Do not use the sine block from the sources palette.

![Figure 2: blocks inside example diagram](image)

Hold the mouse above a block to move it and press `m` (move shortcut key). Now drag it and click or press `m` again to place the block on its new location. The white hint line implicates that clicking on another block now will also result in a moving operation. Move all the blocks to the right location.

Hold the mouse above the scope block and press the `o` button to open it. Set the number of scope inputs to 2 and optionally change the name.

Before connecting the blocks the representations looks better if the blocks are aligned. In Scicos ports can be aligned; this will avoid oblique lines. Hold the mouse above the output port of the sine block. Now press the `a` key and click on the first black input port of the scope. Now the scope block will be aligned vertically with the sine block. Horizontal alignment is also possible.
The next step is to connect the blocks. Always make a link from an output port to an input port. Hold the mouse above the sine block and press L. Now click on the first black input of the scope block or hold the mouse above it and press L or space. Use the right mouse button to cancel a link operation. To connect the input of the gain block click on the link between the sine and the scope, move the mouse down and click on the background, press L or press space. Now click on the gain block and a nice straight line will appear.
Make the other links like the diagram in figure 3.

![Figure 3: connected example diagram](image)

Just like Simulink Scicos also has the ability to make use of symbolic parameters. In Scicos it only works differently. Open the edit menu and select context. A dialog will appear, in that dialog write:

gain = 3

Open the gain block and enter for gain value the name of the defined variable (gain). In this example the use of a symbolic parameter is not relevant but in a complex system it is powerful to change the value of more than one block just by changing one variable. To change the gain value open the context dialog again, enter a different value and the gain value of the gain block will change without opening the block. Symbolic parameters are also visible inside a super block, so it is also possible to define the variable gain in the main diagram (figure 1).

Once the real time program is built and running it is possible to change parameters when the program is running by using the xrtailab application. To identify blocks in this application it is recommended to give each block an identification string. This string will be visible in xrtailab.
Select identification in the object menu or use a shortcut if defined. Now enter for the sine and gain block an identification string.

To change the color of a block or a link click right on it and select color in the properties sub menu.
To place all blocks, except the red timer block, in a super block choose region to super block in the diagram menu. Then, select all blocks except the timer so the result will look like figure 1.

To change the sample time open the red timer block and set the period for example to 0.01 for 100 samples/seconds.

Before generating a program change the name of the corresponding super block. Open the super block and choose rename in the diagram menu. This will be the default filename for the code generation.

3.3.3. Notes

Do not give two similar blocks the same name. In most blocks, the block interface function is identified by a block type specific prefix and the block name. The RTAI code generator does not test for duplicate names causing the resulting real time application to be incorrect.

In Simulink the user is able to build a parameter dialog for a super block using Simulink’s GUI. In Scicos it is not possible to make a mask for a super block in an easy way using the Scicos GUI. Principally it can be done by writing an own super block script. Another way is to save a diagram as a super block interface function (in the Diagram menu choose Save as Interf Func). This function can be loaded using the Add new block function in the Edit menu. Now it is possible to edit the created sci file containing the block interface. In this way a parameter dialog for a super block is created by changing the open/set code (job is equal to ‘set’) of the interface function.

In the misc menu (default) colours like the background colour can be set or new colours can be defined. By default the blocks have a 3D appearance. The aspects option in the misc menu controls this behaviour.

When saving a diagram for the first time, use the save as function. Always save a model when all super blocks are closed so the origin of the model (like figure 1) is visible. When saving a model by applying the save button in an open super block, only the contents of the opened super block will be saved, replacing the original model.

3.4. Creating and running real time programs

3.4.1. Building real time code

Building a real time program is easy. Like already mentioned just click the RTAI CodGen button in the RTAI menu and click the regarding super block. This will activate the RTAI code generation script, situated in the file RTAICodeGen_.sci.
The Scilab main screen monitors the progress and the recognized errors. When an error occurs, Scicos is closed so save the work done before starting the code generator. The files `unix.err` and if available `unix.out` may contain useful information in case of an error. These files are located in the folder `/tmp/SD_<any number>/unix.err`. To find the right folder, look at the modification dates or remove all temp folders before starting Scilab.
See appendix A for two possible errors and solutions regarding code generation.

### 3.4.2. Loading modules

Before it is possible to run a real time program the RTAI modules have to be loaded. The RTAI modules can be loaded by using the `rtaiload` script included in the RTAI package (first create a `.runinfo` file) or by loading the modules manually. The easiest way to load those modules manually is to insert the following code in a text file, name it `rl` (rtai load) and make sure it is executable (in properties, check _Is Executable_):

```
sync
insmod /usr/realtime-rtai-3.2/modules/rtai_hal.ko
insmod /usr/realtime-rtai-3.2/modules/rtai_lxrt.ko
insmod /usr/realtime-rtai-3.2/modules/rtai_fifos.ko
insmod /usr/realtime-rtai-3.2/modules/rtai_sem.ko
insmod /usr/realtime-rtai-3.2/modules/rtai_mbx.ko
insmod /usr/realtime-rtai-3.2/modules/rtai_msg.ko
insmod /usr/realtime-rtai-3.2/modules/rtai_shm.ko
insmod /usr/realtime-rtai-3.2/modules/rtai_netrpc.ko
ThisNode="127.0.0.1"
sync
```

Also make a script named `rr` (rtai remove):

```
sync
rmmod /usr/realtime-rtai-3.2/modules/rtai_shm.ko
rmmod /usr/realtime-rtai-3.2/modules/rtai_msg.ko
rmmod /usr/realtime-rtai-3.2/modules/rtai_netrpc.ko
rmmod /usr/realtime-rtai-3.2/modules/rtai_fifos.ko
rmmod /usr/realtime-rtai-3.2/modules/rtai_mbx.ko
rmmod /usr/realtime-rtai-3.2/modules/rtai_sem.ko
rmmod /usr/realtime-rtai-3.2/modules/rtai_lxrt.ko
rmmod /usr/realtime-rtai-3.2/modules/rtai_hal.ko
sync
```

Now simply load or unload the modules by executing `./rl` or `./rr`.

Note: also load these modules to run Matlab real time programs using the new RTAI patched kernel.

### 3.4.3. Running applications

When the modules are loaded, the program can be executed applying the command:

`./<modelname>` (for example `./pato01`).
Also some command line options are available. Open the main file (*rtmain.c*) or use the command line option `-u` for an overview of the following list of options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-u</code></td>
<td>usage, print usage</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>verbose, verbose output</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>version, print rtmain version</td>
</tr>
<tr>
<td><code>-s</code></td>
<td>soft, run RT-model in soft real time (default hard RT)</td>
</tr>
<tr>
<td><code>-w</code></td>
<td>wait, wait to start</td>
</tr>
<tr>
<td><code>-p &lt;priority&gt;</code></td>
<td>priority <code>&lt;priority&gt;</code>, set the priority at which the RT-model's highest priority task will run (default 0)</td>
</tr>
<tr>
<td><code>-f &lt;finaltime&gt;</code></td>
<td>finaltime <code>&lt;finaltime&gt;</code>, set the final time (default infinite)</td>
</tr>
<tr>
<td><code>-n &lt;ifname&gt;</code></td>
<td>name <code>&lt;ifname&gt;</code>, set the name of the host interface task (default IFTASK)</td>
</tr>
<tr>
<td><code>-i &lt;scopeid&gt;</code></td>
<td>ids cope <code>&lt;scopeid&gt;</code>, set the scope mailboxes identifier (default RTS)</td>
</tr>
<tr>
<td><code>-l &lt;logid&gt;</code></td>
<td>idlog <code>&lt;logid&gt;</code>, set the log mailboxes identifier (default RTL)</td>
</tr>
<tr>
<td><code>-t &lt;meterid&gt;</code></td>
<td>idmeter <code>&lt;meterid&gt;</code>, set the meter mailboxes identifier (default RTM)</td>
</tr>
<tr>
<td><code>-d &lt;ledid&gt;</code></td>
<td>idled <code>&lt;ledid&gt;</code>, set the led mailboxes identifier (default RTE)</td>
</tr>
<tr>
<td><code>-y &lt;synchid&gt;</code></td>
<td>idsynch <code>&lt;synchid&gt;</code>, set the synchronoscope mailboxes identifier (default RTY)</td>
</tr>
<tr>
<td><code>-c &lt;cpumap&gt;</code></td>
<td>cpumap <code>&lt;cpumap&gt;</code>, (1 &lt;&lt; cpunum) on which the RT-model runs (default: let RTAI choose)</td>
</tr>
<tr>
<td><code>-e</code></td>
<td>external, RT-model timed by an external resume (default internal)</td>
</tr>
<tr>
<td><code>-o</code></td>
<td>oneshot, the hard timer will run in oneshot mode (default periodic)</td>
</tr>
<tr>
<td><code>-m &lt;stack&gt;</code></td>
<td>stack <code>&lt;stack&gt;</code>, set a guaranteed stack size extension (default 30000)</td>
</tr>
</tbody>
</table>

### 3.4.4. Xrtailab

After executing the real time program, execute xrtailab by typing `xrtailab`. Connect to the program by using the default parameters (ip is 127.0.0.1). Alternatively a profile can be made, but after modifying the Scicos diagram and building a new program this profile might not work anymore.

Scopes, meters and leds can be displayed and it is possible to change parameters. When the real time program has been started with `-w` as command line option, the program can be activated by clicking the start arrow on the toolbar. By means of a scope it is possible to log data for a predefined period. This function can be selected before activating the program (`-w` command line option).

Remarks:

- Sometimes the scopes are displayed twice in xrtailab. In this case close the programs and try to remove and reload all the modules (`./rr` and `./rl`).

- Also other errors might occur. Sometimes xrtailab can not connect to the real time program or sometimes xrtailab suddenly disappears. In those cases try to restart the programs, try to unload and reload the modules or if nothing seems to work restart the computer.
Save all open projects before loading the RTAI modules or starting the real time application. In some cases the system might stop responding.

Xrtailab might give the message: Cannot find default scheme. This can be ignored.

In the real time loop, do not print text on the screen by using the printf function when xrtailab is connected because in this case xrtailab does not work properly (xrtailab can abruptly disappear).

Figure 4: xrtailab screenshot
4. **Writing new blocks**

To implement for example the TUnDACS system in Scicos new blocks representing this system must be written. In Simulink these TUnDACS blocks already exists. Each block consists of a standard S-function block calling a user made S-function which is defined in a Simulink MEX-file.

In Scicos the procedure is somewhat different.

4.1. **Block and code structure**

To make a new Scicos block the user has to write a Scilab script file. This *.sci file instructs Scicos about the appearance and properties of the block. This script also generates the C-code to be used in the standalone real time program.

The base for a new block consists of three fundamentals, summarized in the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Defined in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block properties like states, block type, number of inputs/outputs and parameters</td>
<td>Scicos: Block script  Simulink: S-function (compiled into a mexglx-file)</td>
</tr>
<tr>
<td>The appearance of the block</td>
<td>Scicos: Block script, can be changed using the Scicos GUI when using the <code>standard_draw</code> method in the block script  Simulink GUI</td>
</tr>
<tr>
<td>Real time code</td>
<td>Scicos: Block script must generate code and store it in a block property (as string in a variable)  Simulink: S-function contains this c-code</td>
</tr>
</tbody>
</table>

A new block is specified in only one file, called a *.sci file. It contains a block interface function which provides all necessary information. This interface function also provides a C-code function, used when a real time program is build. As this C-code must be offered as a string variable, for large block codes it is easier to generate small functions. In that case the block interface function calls different functions located in a separate file which need not to be generated by the block script.

At last the Scilab interface function has to be loaded in Scicos. To avoid that Scicos can not find the corresponding file it is better to include the new *.sci file in the RTAI add-ons so the new file will be loaded automatically in Scilab. The separate C-file can be included in the RTAI add-ons too, so the C-compiler (which will compile the real time code) always can find the new user defined function.
In order to build the standalone real time application the RTAI code generator (RTAICodeGen_.sci) creates or uses the following files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;program name&gt;.c.c</code></td>
<td>In this file the C functions of each block generated by the block script are placed.</td>
</tr>
<tr>
<td><code>&lt;program name&gt;_&lt;standalone.c</code></td>
<td>In the file the properties like xrtailab labels and connections between blocks are stored.</td>
</tr>
<tr>
<td><code>rtmain.c</code></td>
<td>The block’s main file (standard file, the code generator does not modify this file).</td>
</tr>
</tbody>
</table>

In combination with the main file (`rtmain.c`) a standalone real time program is build. The main file performs the following tasks:
- initialization and finalization of all blocks,
- switch to hard real time,
- execution of the main program loop and
- providing the communication protocol for xrtailab.

### 4.2. Steps needed to implement a new block

Summarizing the previous paragraph, the following steps have to be taken in order to implement a new block:

- Create a C file for the real time block code.
- Include this file to the RTAI add-ons.
- Create a *.sci file containing the block interface function.
- Also include this file to the RTAI add-ons.
- Load the new block in Scicos and eventually add it to a palette.

First create a C file and add it to the RTAI add-ons:

The simplest way to do this is to use the `gen_dev` utility. This utility creates a new C file (`/usr/src/rtai-3.2/rtai-lab/scilab/devices/<blockname>.c`, using `template.c`) and includes the function prototypes to the `devices.h` file:

```
  cd /usr/src/rtai-3.2/rtai-lab/scilab/devices
  gen_dev <blockname>
  Open /usr/src/rtai/rtai-lab/scilab/devices/GNUmakefile.am and add <blockname>.c to the libsciblk_a_SOURCES list.
  cd /usr/src/rtai-3.2
  aclocal (optional, only when needed)
  automake rtai-lab/scilab/devices/GNUmakefile
  make
  make install
```

Remark: after previous steps the new C file (`<blockname>.c`) should be included to the RTAI package and it should be compiled by the `make` command. However sometimes this does not work. To test this include a syntax error in the new C file to monitor whether the compilation process halts. If the files are not included check and try the following:
In the directory `rtai-lab/scilab/devices` the files `GNUmakefile` and `GNUmakefile.in` should be rebuilt now, check the modification dates. Also these files must contain `<blockname>.c`.

- Open and resave `GNUmakefile.am` (modify the last modification date).
- Try to delete `GNUmakefile.in` and recreate it using the `automake` command.
- In `/usr/src/rtai` try `make menuinstall`, exit and save the configuration, `make` and `make install` (maybe it is necessary to fist `make clean` or `make distclean`).

How the C code can be customized will be discussed in the next paragraph.

Next create a Scicos interface function (*.sci file):

- Go to `/usr/src/rtai-3.2/rtai-lab/scilab/macros/RTAI` and create a new *.sci file. Use a copy of the file `scicos_block_template.sci`.
- Open `/usr/src/rtai-3.2/rtai-lab/scilab/macros/RTAI/Makefile` and add the new file to the `MACROS` list.
- Customize the *.sci file (see the Explanation of new block template paragraph)
- `cd /usr/src/rtai-3.2/rtai-lab/scilab/macros`
- `make`

Finally the new block can be loaded in Scicos:

- Open Scicos and select the Add new block function in the Edit menu. Insert the name of the interface function, this is the name selected in the *.sci file.
- A diagram can be loaded or saved as palette. It is also possible to modify palettes.

### 4.2.1. Customize the C code

In the standard form, the new C file (copied from the template) contains 7 functions. These functions are called by the (main) C function generated in the *.sci file. Standard this main function only looks at the flag parameter and depending on this parameter it calls one of the 7 functions:

<table>
<thead>
<tr>
<th>Function name</th>
<th>Flag value</th>
<th>Data direction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inp_&lt;blockname&gt;_init</code></td>
<td>4</td>
<td>n/a, initialization</td>
</tr>
<tr>
<td><code>inp_&lt;blockname&gt;_input</code></td>
<td>1</td>
<td>C/hardware input to block output</td>
</tr>
<tr>
<td><code>inp_&lt;blockname&gt;_update</code></td>
<td>2</td>
<td>block input to C/hardware</td>
</tr>
<tr>
<td><code>inp_&lt;blockname&gt;_end</code></td>
<td>5</td>
<td>n/a finalization</td>
</tr>
<tr>
<td><code>out_&lt;blockname&gt;_init</code></td>
<td>4</td>
<td>n/a, initialization</td>
</tr>
<tr>
<td><code>out_&lt;blockname&gt;_output</code></td>
<td>2</td>
<td>block input to C/hardware output</td>
</tr>
<tr>
<td><code>out_&lt;blockname&gt;_end</code></td>
<td>5</td>
<td>n/a finalization</td>
</tr>
</tbody>
</table>
Depending of the type of block only a selection of these functions can be applied and the remaining functions removed. When the standard format is not useful, the next step is to customize the function headers. For example block parameters required in the initialization function of a block must be implemented in that function. In the (main) C function generated by the *.sci file make sure that those parameter values are passed through.

Some blocks require memory, for example to store the last output value or an internal state. As a block can be applied more than once the specification of static variables is not allowed for one of these 7 standard functions. Therefore these static variables must be included in the block’s main function generated by the *.sci file. The name of each block has to be unique and standard the name of the block’s main function is dependent on the block name. So this function is unique and it is possible to safely use static variables.

The real time standalone main file (rtmain.c) includes two arrays of variables of the type devStr:

```c
devStr inpDevStr[40];
devStr outDevStr[40];
int pinp_cnt = 0;
int pout_cnt = 0;
```

Defined in the file /usr/src/rtai-3.2/rtai-lab/scilab/devices/devstruct.h:

```c
typedef struct devStr{
    int nch;
    char IOName[20];
    char sName[20];
    char sParam[20];
    double dParam[5];
    int i1;
    long l1;
    long l2;
    void * ptr1;
    void * ptr2;
}devStr;
```

A block can use one of those structures to store local data. In the template C file the use of this procedure is shown. The static variable `port` in the main function remembers which variable in the array is reserved. Of course it is also possible to define another static variable instead of the `port` variable in the main function for instance a pointer variable. In this case the initialization function can dynamically allocate memory and after closing the program the finalization function can free it.

When all functions are specified, some functions might be removed or some headers might be changed. Open the file `devices.h` and correct the function declarations in this file.

Finally build the new code and install it:

- `make` (check for syntax errors and warnings)
- `make install`
4.2.2. Explanation of new block template

To easily build a new Scicos block, as part of this investigation, a universal block template is designed (see appendix C). This block template is derived from the example blocks as described in [1] and from existing RTAI blocks. The advantage of this template is the presence of parameter explanations and options so it will not take too much time to figure out how things work. These explanations are copied from the Scilab manual. Also this block code is universal and can be used for blocks with several purposes.

To build a new block this template can be adapted. Building an own new block script is also possible. The application of the block template will be explained. See the Scilab manual for help on a certain function. To use the template the following modifications are necessary:

- **First editable block between the EDIT THIS PART and END EDIT lines:**
  - The script function name has to be the same as the filename of the script.
  - Furthermore block properties are defined, see comments or the Scilab manual for more information.

- **Next (by default non editable) part:**
  - The job variable instructs the script which operation to perform.
  - ‘plot’: assign the first parameter (block name) to the variable *name*, standard_draw will display the block using the instructions as given in initial_icon (these instructions make use of the variable *name*).
  - ‘getinputs’, ‘getoutputs’ and ‘getorigin’: use standard functions to determinate the positions of the in/outputs and origin.
  - ‘set’: when the job parameter is equal to ‘set’ the script has to show his parameter dialog and it has to generate the (real time) C-code.

- **Next step between the EDIT THIS PART and END EDIT lines:**
  - This is the code to show a parameter dialog. Two standard Scilab dialogs (*xchoices* and *getvalue*) are implemented. Use one of those dialogs and remove the other code or write an own dialog code. Notice that in both dialogs the block name is the first parameter. Values can be optionally assigned to the *rpar* and *ipar* parameters. These parameters can be changed by xrtailab.

- **Next (by default non editable) part:**
  - A name for the block’s real time main function is created (*funam*).
  - The function *getCode* is defined at the end of the file. This function generates the real time C-code using *funam* as function name.
  - After the C-code is successfully generated the block will be checked. Although already done in the ‘define’ part next block properties are (re)assigned. Almost every standard rtai block reassigns all its parameters. It is unknown why this is done. The blocks also work without that reassignment. As the reason for reassignment is unknown in this block template this reassignment is also performed. The variable temporary *tt* contains the C-code and is assigned to the *exprs(2)* parameter (the *expr(1)* parameters contains a list of all parameter values).
- When job is equal to ‘define’, a new block structure is created and block properties as well as initial parameters are assigned.

- At last the code to generate the C-code has to be adapted (function `getCode`):
  - The real time C-code is generated and stored in variable `tt`. The standard code only looks at the variable `flag` which tells what action to perform and executes a corresponding external function. See the previous paragraph for more information about this function. For a standard new block enter the `cfunctionname` as used in the `gen_dev` utility.
  - For debugging purposes it is possible to measure the time it takes to call the external function and the number of function calls. In this case those data are printed on the screen after closing the real time program. Assign `%t` or `%f` to the variables `measure_input_time` and `measure_output_time` to measure those times or not.
  - After that some variables are declared; the program standard uses a static variable called `port` to remember block properties, see also previous paragraph for more information.
  - Then, at last, the code only reads the variable `flag` and depending on that variable it executes initialization, in- and output and finalization functions.
  - Use block parameters as function parameters to pass through those parameters. The tunable (xrtailab) parameters can be accessed by the instructions `block->ipar[x]` (integer parameters) or `block->rpar[x]` (real parameters) where `x = 0...n-1` for `n` parameters.

4.3. **Example of converting existing Matlab S-function, implementation of TUeDACS**

In this paragraph the implementation of the TUeDACS system will be discussed. In order to use this system in Scicos two new blocks are written representing the in- and output ports. The new files will be translated from the Matlab S-function files. Only the translation of the output file will be explained, see the CD for the result of the other file.

Before the TUeDACS can be used make sure the right drivers (for RTAI) are installed.

To implement the TUeDACS two files have to be modified before making the new blocks.

4.3.1. **Modify the main file**

When using the TUeDACS device it is necessary to initialize the driver before processing input/output instructions. This initialization function must be called only once in soft real time. Immediately after ending the real time application the finalization function must be called. This must be done as soon as possible to avoid uncontrolled motion and it is no problem to call it more than once or in hard real time.
In the standalone main file the program switches between soft and hard real time. To avoid errors when not using the TUeDACS devices this initialization has been implemented in the Scicos blocks representing the TUeDACS in/output functions. As the exit function has to be called immediately after stopping the target the real time main file (rtmain.c) must be modified:

- Open /usr/src/rtai/rtai-lab/scilab/rtmain.c
- Below the line (around line 120):
  ```c
  int pout_cnt = 0;
  ```
  add:
  ```c
  //################ ADDED ########################
  int td_exit(void);
  int td_already_initialized = 0;
  //############################################
  ```
- Search for the line (around line 321):
  ```c
  NAME(MODEL,_init_blk)();
  ```
  and replace it with:
  ```c
  //################ ADDED ########################
  td_already_initialized = 0;
  //############################################
  NAME(MODEL,_init_blk)();
  //################ ADDED ########################
  if (td_already_initialized == -1) {
    printf("TUeDACS error: devices could not be initialized (rtmain.c) \n");
    return (void *)1;
  }
  //############################################
  ```
- Finally implement the exit code immediately after ending the program, before the program switches back to soft real time, so before the line (around line 348):
  ```c
  if (UseHRT) {
    rt_make_soft_real_time();
  }
  ```
  add the exit code so the code will look like:
  ```c
  //################ ADDED ########################
  if (td_already_initialized != 0) {
    td_exit();
    printf("td_exit called (rtmain.c) \n");
  }
  //############################################
  if (UseHRT) {
    rt_make_soft_real_time();
  }
  ```
- Save and close
- Now the new main file has to be installed:
  - `cd /usr/src/rtai-3.2`
  - `make install`

Explanation: the initialization function of each TUeDACS block first checks whether `td_already_initialized` (global variable) is equal to 0. If this variable is equal to 0, it
tries to initialize the TUeDACS system and it changes `td_already_initialized` to 1 or in case of an error to -1.

### 4.3.2. Modify the code generation file:

To link the shared TUeDACS library a small modification of the RTAI code generation file must be made:

- Open `/usr/src/rtai/rtai-lab/scilab/macros/RTAI/RTAICodeGen_.sci` and search for the following line (line 1316):
  
  ```asci(9)+"gcc -static -o $@ $(OBJSSTAN) $(SCILIBS) $(ULIBRARY) -lpthread -lm"
  ```

- Remove `-static` (otherwise it is not possible to link against the dynamic TUeDACS library) and add `-ltd` (the shared TUeDACS library) to this line so it will look like:
  
  ```asci(9)+"gcc -o $@ $(OBJSSTAN) $(SCILIBS) $(ULIBRARY) -lpthread -lm -ltd"
  ```

- Install the new file:
  - `cd /usr/src/rtai/rtai-lab/scilab/macros`
  - `make`

### 4.3.3. Building the C code

Before making a new block script the block code will be translated. Here all the steps that have to be made are enumerated. This might seem a bit complex, see appendix D for the complete resulting file.

- Create a new C file, `td_outports.c`, using the `gen_dev` utility and add it to the RTAI add-ons as mentioned early in this report.

- Open `devices.h`, remove `inp_td_outports_*` functions and change the other 3 `td_outport` functions to:

  ```
  void out_td_outports_init(const int QADID, const int DAC_ACTIVE_CH1, const int DAC_HOLD_ORDER_SELECT_CH1, const int DAC_ACTIVE_CH2, const int DAC_HOLD_ORDER_SELECT_CH2, const int DIO_ACTIVE, const int PWM_ACTIVE_CH1, const int PWM_FUNCTION_SELECT_CH1, const int PWM_FREQ_SELECT_CH1, const int PWM_ACTIVE_CH2, const int PWM_FUNCTION_SELECT_CH2, const int PWM_FREQ_SELECT_CH2);
  void out_td_outports_output(double * u, const int QADID);
  void out_td_outports_end(const int QADID);
  ```

As can be seen, all the TUeDACS parameters are implemented as input parameters of the `init` function. The QADID parameter is also needed in the other functions.
- Open the new generated file `td_outports.c` and delete all its code.
- Open the existing Matlab source file to be translated, `td_outports.c`.
- Select all and copy paste it to the new (Scicos) `td_outports.c`.
- Replace the following function headers to the corresponding headers as defined in `devices.h`, thus (remove the semicolon's (;)):

Replace:

```c
static void mdlInitializeSizes(SimStruct *S)
```

with

```c
void out_td_outports_init(const int QADID,
    const int DAC_ACTIVE_CH1, const int
    DAC_HOLD_ORDER_SELECT_CH1,
    const int DAC_ACTIVE_CH2, const int
    DAC_HOLD_ORDER_SELECT_CH2,
    const int DIO_ACTIVE,
    const int PWM_ACTIVE_CH1, const int
    PWM_FUNCTION_SELECT_CH1, const int
    PWM_FREQ_SELECT_CH1,
    const int PWM_ACTIVE_CH2, const int
    PWM_FUNCTION_SELECT_CH2,
    const int PWM_FREQ_SELECT_CH2)
```

```c
static void mdlOutputs(SimStruct *S, int_T tid)
```

with

```c
void out_td_outports_output(double * u, const int QADID)
```

and

```c
static void mdlTerminate(SimStruct *S)
```

with

```c
void out_td_outports_end(const int QADID)
```

- Statements like:

  ```c
  #define QADID                           ssGetSFcnParam(S,0)
  ```

  are not needed because the quad parameters are directly given as parameters of the init function. Remove the original `#define` and `#include` statements. Also remove the prototypes comment. So remove all the lines from

  ```c
  #define S_FUNCTION_NAME td_outports
  ```

  till the beginning of the `out_td_outports init` function.

- The matlab types `int_T` and `real_T` are not supported in the new c file. The `int_T` and the `real_T` type can be replaced with the `int` and `double` type.

  To avoid changing large pieces of the code it is easier to define the following macros:

  ```c
  #define int_T   int
  #define real_T  double
  ```

  Also define the TUeDACS functions, enter the following lines:

  ```c
  #define TD_DIRECT       0
  #include <stdlib.h>
  #include <stdio.h>

  /* prototypes */
  int td_init(void);
  ```
Initialization code:

- Add the TUeDACS initialization code:
  Before the initialization function insert the following function:

  ```c
  void td_exit_td_outports(void)
  {
    td_exit();
    printf("td_exit called using atexit (td_outports.c)\n");
  }
  ```

  In the initialization function, at the start, add the following code:

  ```c
  extern int td_already_initialized;
  int nDevs = 0;
  if (td_already_initialized == 0) {
    td_already_initialized = 1;
    printf("Initializing TUeDACS/1 devices... (td_outports.c)\n");
    nDevs=td_init();
    if (nDevs>10) {
      printf("TUeDACS error: devices could not be initialized (td_outports.c)\n");
      td_already_initialized = -1;
      exit(EXIT_FAILURE);
    }
    atexit( td_exit_td_outports );
    printf("TUeDACS/1 devices ready (td_outports.c)\n");
  }
  ```

- Remove all the lines outside the #ifndef MATLAB_MEX_FILE .... #endif blocks because these definitions in Scicos are placed in the block script (sci file). In this block script the same values will be used. So remove the following lines:

  ```c
  ssSetNumSFcnParams(S,NPARAMS);
  ssSetNumContStates(S,NSTATES);
  ssSetNumDiscStates(S,0);
  if (!ssSetNumInputPorts(S,1)) return;
  ssSetInputPortWidth(S,0,NINPUTS);
  ssSetInputPortDirectFeedThrough(S,0,0);
  if (!ssSetNumOutputPorts(S,0)) return;
  ssSetOutputPortWidth(S,0,NOUTPUTS);
  /*
  ssSetNumSampleTimes(S,1);
  */
  ```
ssSetNumRWork(S,0);
ssSetNumIWork(S,0);
ssSetNumPWork(S,0);
ssSetNumModes(S,0);
ssSetNumNonsampledZCs(S,0);

- Next remove all the #ifndef MATLAB_MEX_FILE and #endif lines (do not remove the lines in those #define blocks)

- As the block parameters are directly passed to the init function the Matlab mxGetPr(parameter) function is not needed. Simply those variables are accessed directly, so the line:
  
  \[ \text{ilink} = (\text{int}_T) \left( \text{mxGetPr} (\text{QADID}) \right) - 1; \]

  is changed in:
  
  \[ \text{ilink} = \text{QADID} - 1; \]

- In the total file, remove all these constructions, for example, change:
  
  \[
  \text{ipar} = (\text{int}_T) \left( \text{mxGetPr} (\text{DAC_ACTIVE_CH1}) \right); \]

  \[
  \text{if} \ (\text{ipar} == 1) \ \{ \]
  
  \[ \text{enable channel} \]
  
  \[ \text{td\_dac\_set\_enable\_chan} (1, 0, \text{ilink}); \]
  
  \[ \text{ipar2} = (\text{int}_T) \left( \text{mxGetPr} (\text{DAC\_HOLD\_ORDER\_SELECT\_CH1}) \right) - 1; \]
  
  \[ \text{td\_dac\_set\_filter\_order\_chan} (\text{ipar2}, 0, \text{ilink}, \text{TD\_DIRECT}); \]

  \[
  \} \text{ else } \{ \]
  
  \[ \text{disable channel} \]
  
  \[ \text{td\_dac\_set\_enable\_chan} (0, 0, \text{ilink}); \]

  \[
  \}
  \]

  In to:

  \[
  \text{if} \ (\text{DAC\_ACTIVE\_CH1} == 1) \ \{ \]
  
  \[ \text{enable channel} \]
  
  \[ \text{td\_dac\_set\_enable\_chan} (1, 0, \text{ilink}); \]

  \[
  \text{td\_dac\_set\_filter\_order\_chan} (\text{DAC\_HOLD\_ORDER\_SELECT\_CH1}, 0, \text{ilink}, \text{TD\_DIRECT}); \]

  \[
  \} \text{ else } \{ \]
  
  \[ \text{disable channel} \]
  
  \[ \text{td\_dac\_set\_enable\_chan} (0, 0, \text{ilink}); \]

  \[
  \}
  \]

- The entire function `mdlInitializeSampleTimes` is not needed in Scilab, remove it.

Output code:

- The line:
  
  \[
  \text{InputRealPtrsType uPtrs} = \text{ssGetInputPortRealSignalPtrs} (S, 0); \]

  Is not needed because the block input signal can be accessed directly (function parameter u). Remove that line.

- The uppercase `U` variables where previously defined as a macro. Now it is possible to replace those uppercase `U` variables manually with the lowercase `u` function parameter. Also replace the round () with square [] brackets. It is easier to use a macro (#define U(x) u[x]) to get this done.
Finalization code:

- Only the same adoptions as mentioned above have to be applied.
- At the end of the file some other files are include. Remove these include lines.
- At last install the new file.

4.3.4. Create the Scicos script

To create the Scicos script for the new block use the Scicos block template, add it to the RTAI add-ons and make the changes as described in appendix D. Finally install the new file.

4.4. Building the pato01 model

In order to build the pato01 model in Scicos additional blocks have to be translated. As an example some parts of the code for the DCT PD block and the Ref3 block are given. After the translation of all needed blocks the new blocks are added to a new palette.

Using this new blocks the pato01 diagram was built and tested. To test the model the bode diagram of a dynamic model is determined by using both a Scicos and a Simulink real time program. See chapter 5 for the test characteristics and results.
4.4.1. Example of translating other blocks, DCT PD and Ref3

As an example to show the procedure used in the other blocks, two functions
generated by the Scicos block script are given in appendix D. The first function is the
generated function for the DCT PD block. This block uses the xrtailab tunable
parameters. The original Matlab code uses the \texttt{real\_T *rwrkpr=ssGetRWork(S);} construction to store block data. The Scicos program applies a static variable named
\texttt{rwrkpr} as a pointer to a dynamic array. This array is created in the initialization
function ( \texttt{rwrkpr = out\_dpd\_init();} ) and used in the i/o function ( \texttt{out\_dpd\_output(u, y, rwrkpr, block->rpar[0], block->rpar[1]);} ).

The ref3 function is special because it has a variable number of parameters. These
parameters are passed to the initialization function using a pointer to a temporary
dynamic array.

4.4.2. Not explained errors

The new TUEDACs input and output blocks generate two not explained errors.

In the pato01 model those blocks are included within a super block. In the main
diagram, when changing the context variables (\texttt{edit > context}) the parameter dialogs
of those two TUEDACs blocks are opened. This is awkward because these blocks do
not use a context variable.

An explanation of this phenomenon is that the C code of these blocks needs to be
regenerated by opening each block. The parameters are directly implemented in the C
code. In order to be sure the code is regenerated using the new parameters all the
blocks are opened. Maybe the use of a different type of dialog (\texttt{xchoices} instead of
\texttt{getvalue}) is an explanation why the dialogs of other blocks are not visible. In this case
Scicos is able to prevent that those \texttt{getvalue} dialogs are shown.

Another not explained error in the pato01 model concerns the block type and block
dependence of the TUEDACs output block:

\begin{verbatim}
blocktype           = "d"   // blocktype:a character with
possible values:
//: 'c', block output depend continuously of the time.
//: 'd', block output changes only on input events.
//: 'z', zero crossing block
//: 'l', logical block
// Default value: "c"

dep_u               = %t    // dep_u must be true if output
depends continuously of the input

dep_t               = %f    // dep_t must be true if output
depends continuously of the time

dep_ut              = [dep_u, dep_t] // dep_ut:1x 2 vector of
boolean [dep_u, dep_t], default value: [%f,%f]
\end{verbatim}

This usually works but in case this output block is placed in a super block and input
signals are not connected directly but through super block channels an error occurred.
The measure output option of the Scicos block template indicates that the output
function is not called anymore. No data is sent to the device. To solve this error in the
pato01 model a dummy sine block is included inside the super block connected
directly to an unused input port of the TUeDACS output block.
This is bizarre but perhaps a signal passed through a super block channel is not able to
activate a block.
Using the dummy sine the pato model seems to function only for debugging purposes,
when including an extra scope inside another super block (the controller block) again
the same error occurred. The device’s output function is not called anymore. This is
not predictable and probably a bug.
A solution is to change the block type to c, make this block continuously depending
on time.

4.5. Customizing blocks

4.5.1. Change appearance of blocks

In Matlab the appearance of a block is changed by using the GUI. The user can place
pictures, label in- and output ports, change the colour of a block and of course modify
the caption of a block.
In Scicos these possibilities are limited. Using special commands it is possible to draw
figures and print text on a block. See the Graphics Library section of the Scilab
manual for a list of those functions. These commands must be entered in the icon
property of a block.
There are two ways to change the colour of a block; the first way is to use the color
option in the properties sub menu of a block menu. When customizing a new block,
the colour of the block must be modified after loading the block in Scicos. Use a
palette to prevent changing this colour each time a block is loaded.
In the second way, the colour of a block can be modified using the xrect draw
command. For new blocks the block colour can be specified in the block script. The
size of the rectangle and the block do not have to be exactly the same. The
disadvantage of this code is that the colour of the text printed on the block is merged
with the background colour.

```plaintext
initial_icon = ['xrects([orig(1);orig(2)+sz(2);sz(1);sz(2)],scs_color(11));'
    'xstringb(orig(1),orig(2),[''Block caption''],
    sz(1),sz(2),''fill'');']
```

Another way to create icon expressions instead of using the Scilab manual is to use
the icon editor (Block menu > Properties > Icon Editor) in Scicos. Note that original
icon expressions will be overwritten.

In Scicos, placing picture on a block is not supported.
4.5.2. Xrtailab labels

In xrtailab each block and each parameter is labelled.

The assignment of the block label can be found in the code generator script, see /usr/src/rtai-3.2/rtai-lab/scilab/macros/RTAI/RTAICodeGen_.sci, at line 1897:

```scheme
if stripblanks(OO.graphics.id)~=emptystr() then
    str_id = string(OO.graphics.id);
else
    str_id = 'RPARAM[' + string(nbrpa) + ']' + string(OO.graphics.id);
end
RCode=[RCode;
cformatline('Identification: '+strcat(str_id),70)];
```

The `graphics.id` is the identification string of the block as defined in Scicos (`Object > Identification`). As noticeable in xrtailab the default label for block (when no identification string is defined) is `RPARAM[x].`

Modification of a parameter label is not (yet) supported in RTAI-lab. By default all parameters use the label `Value[x]`. In the main file, `rtmain.c`, these default values are assigned (search for `Value[0]` and find the block given below two times):

```c
for (i = 0; i < NRPAR1; i++) {
    sprintf(rtParam.blockName, "%s/%s", rtParam.modelName, strRPAR1[i]);
    if(i==0) Idx = 0;
    else Idx += lenRPAR1[i-1];
    for(j=0;j<lenRPAR1[i];j++) {
        rt_receivex(task, &Request, 1, &len);
        sprintf(rtParam.paramName, "Value[%d]", j);
        rtParam.dataValue[0] = RPAR1[Idx+j];
        rt_returnx(task, &rtParam, sizeof(rtParam));
    }
}
for (i = 0; i < NIPAR1; i++) {
    sprintf(rtParam.blockName, "%s/%s", rtParam.modelName, strIPAR1[i]);
    if(i==0) Idx = 0;
    else Idx += lenIPAR1[i-1];
    for(j=0;j<lenIPAR1[i];j++) {
        rt_receivex(task, &Request, 1, &len);
        sprintf(rtParam.paramName, "Value[%d]", j);
        rtParam.dataValue[0] = IPAR1[Idx+j];
        rt_returnx(task, &rtParam, sizeof(rtParam));
    }
}
```
5. Results

5.1. The pat01 model

After creating new blocks the pat01 model could be built as shown in Figure 6. This model is similar to the original Simulink pat01 setup. Controller parameters can be set using the Context option in the Edit menu. A parameter dialog (see Figure 7) will popup.

![Figure 6: the pat01 setup](image)

![Figure 7: controller parameters](image)
5.2. **Testing the pato01 model**

To test the Scicos pato01 setup and the new Scicos blocks three tests are performed:

- First, the transfer function of a dynamic system is measured and compared to the results of the same experiment and using Simulink.
- Next for the Scicos real time program, sample time errors are determined.
- At last, during a measurement on the same dynamic system, the CPU load of the Scicos and the Simulink real time programs is compared.

For all the tests a TU/e, generation 2002 & 2003, notebook is used. This is a Nec Versa P520 computer with a 1400 GHz Pentium M processor and 512 Mb ram.

5.2.1. **Measurement on a dynamic system**

In this section the pato01 setup will be tested. In the previous chapter this setup was build in Scicos. The transfer function of a dynamic system will be determined using both the new Scicos and the original Simulink model.

![Figure 8: the pato setup for the experiment](image-url)
In figure 8 the dynamic system is shown. This is a fourth order system with two rotating masses and one spring connecting both masses. The transfer function of the first mass (directly connected to the motor) will be measured in close loop using the following parameters:

<table>
<thead>
<tr>
<th>Controller:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Kp</td>
<td>0.1</td>
</tr>
<tr>
<td>Kv</td>
<td>0.01</td>
</tr>
<tr>
<td>Notch zero</td>
<td>300</td>
</tr>
<tr>
<td>Notch damping zero</td>
<td>0.1</td>
</tr>
<tr>
<td>Notch pole</td>
<td>300</td>
</tr>
<tr>
<td>Notch damping pole</td>
<td>0.1</td>
</tr>
<tr>
<td>Roll off poles</td>
<td>300</td>
</tr>
<tr>
<td>Roll off damping poles</td>
<td>0.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Noise:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance</td>
<td>1</td>
</tr>
<tr>
<td>Mean</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ref3 parameters:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 500 49.95 1.0081 615</td>
<td></td>
</tr>
<tr>
<td>500 11.0081 0 49.95 1.0081 615</td>
<td></td>
</tr>
<tr>
<td>0 -1 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

Measure time 180 seconds
Sample rate 3000 Hz

The experiment starts with the command:

```
pato01 -v -w
```

The options –v (verbose output) and –w (wait till the program is started by Xrtailab) are optional. Also Xrtailab must be started.

When the experiment is started the following output is printed:

```
Target settings:
  Real-time : HARD;
  Internal Clock is OneShot;
  Priority : 0;
  RUN FOR EVER.

TARGET STARTS.
Initializing TUEdacs/1 devices... (td_outports.c)
pbSetUserModeIO: switched link 0 to user mode IO.
TUEdacs/1 QAD revision 2 found at link 0.
TUEdacs/1 devices ready (td_outports.c)
Model : pato01 .
Executes on CPU map : f.
Sampling time : 3.333000e-04 (s).
Target is waiting to start ... 
```

And after activation in Xrtailab:

```
Target is running.
```
After ending the experiment (press ctrl + c) the first action the program executes is the finalization code for the TUeDACS devices (in the project’s main file):

```c
td_exit called (rtmain.c)
```

And the finalization code of the other blocks is executed:

```c
Scope closed
Scope closed
```

In the Scicos block template (Appendix C) there are two selectable options to measure the time it took for executing the input or output code in that block (measure_input_time and measure_output_time). If one or more of those options are selected, the relevant measure times are printed on the screen. For the TUeDACS blocks these times are:

```plaintext
//==== td_outports blk-input >> c-output measurement times =====
|| Total runtime in seconds: 246.550122 ||
|| Number of function calls: 558087 ||
|| Mean time (ns) in function: 23883 ||
\==============================================================//

//===== td_inports c-input >> blk-output measurement times =====
|| Total runtime in seconds: 246.549703 ||
|| Number of function calls: 558087 ||
|| Mean time (ns) in function: 29737 ||
\==============================================================//

For empty scripts (just calling two times the function `get_cpu_time_ns()` ) these times are around 3200 ns. See figure 9 for a screenshot of the command window after ending the pato01 program.

![Figure 9: screenshot after ending the real time program](image-url)
To calculate the bode diagram of the dynamic system; the process sensitivity is divided by the sensitivity. The results are shown in figure 10.

![Bode diagram of the fourth order system](image)

**Figure 10: bode diagram of the fourth order system**

As indicated in the diagrams the results of the Scicos and the Simulink real time programs are equal.

To test the controller blocks the transfer functions of the controllers in both systems are calculated, see figure 11. Due to the fact that this is not a physic measurement both lines are exactly equal.
These results indicate that the Scicos pato01 setup is working properly.

5.2.2. Sample time errors

For this experiment, a block that retrieves the sample time error in nanoseconds is created. This block makes use of the RTAI function `get_cpu_time_ns()`. This block is placed in a new, empty, Scicos diagram. Then the rest of a simple setup, consisting of a ref3 block and a scope, is placed. At last, after all other blocks were placed and connected, another measure block is added. In this diagram, for each sample, first the code of the first measurement block is called followed by the code of the ref3 and the next measurement block.

Several experiments show a mean error of about 150 ns. Compared to the sample time of 1 ms (1000 Hz) this error is negligible:

\[
\frac{150 \cdot 10^{-9}}{1 \cdot 10^{-3}} \cdot 100\% = 0.015\%
\]

The variance of the errors measured by the second block is a bit higher; this is because the execution time of the ref3 block also may vary.
In figure 12 a histogram of errors measured by the fist block is represented. The histogram of the second block looks almost the same but only the diagram is a little bit lower and wider. Almost all samples showed an error of 150 ns. Only a few samples created much bigger errors. The maximum error before the ref3 block is below 5200 ns, equivalent to an error below 0.5 % (after the ref3 block 8000 ns; equivalent to 0.8 % error rate).

Considering the time (1600 ns) it takes to only call the function RTAI function `get_cpu_time_ns()` this error is really small.

These results demonstrate the Scicos real time program is very accurate for measuring and controlling dynamic systems.
5.2.3. CPU usage

At last the CPU load of the same experiment as described in paragraph 5.2.1 (determination of the transfer function of a dynamic system) is captured. Each second the CPU load is logged by the KSysGuard program (K Menu > System > KSysGuard).

During the experiments data logging was enabled and all scopes were closed. The mean CPU load caused by the Scilab and the Simulink pato01 setup differs a lot:

<table>
<thead>
<tr>
<th></th>
<th>Scicos program:</th>
<th>Simulink program:</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU usage (%)</td>
<td>23.5</td>
<td>66.8</td>
</tr>
</tbody>
</table>

Partly these differences are explained by the fact that although in both programs the scopes were shut down in the Simulink setup two additional meters were present. It takes some CPU time to update these meters real time.
6. Conclusions

The installation process turned out to be more difficult than expected. Some unexpected errors appeared. When starting to use RTAI/Scicos be aware that it takes time to get everything installed properly. When installed properly and working normally, Scilab and Scicos are good working programs and both provide much functionality. Although at the start it might be difficult to become used to it.

The next step is to build new blocks. Compared to Simulink in Scicos the structure to build new blocks is more transparent. There are no standard procedures to access for example the values of block parameters. It is simple, when building a block the user can define its own code construction to pass through, for example block parameters. In this way it is easier for the starting user to find out how to create new blocks. But being a more advanced user and knowing the standard procedures compared to Matlab it takes more time in Scilab to create new blocks.

To simply create new Scicos blocks a block template was written, which made the creation of new Scicos blocks much easier.

One of the goals of this project was to translate the pato01 model. This goal was achieved. Based on new (translated from Simulink) blocks a new pato01 model for Scicos was created. It should be noticed that the new model does not contain the graphical user interface for the ref3 block. This interface can not be built in Scicos in the same way as Matlab supports. Another procedure has to be considered.

A real time program was built using Scicos. According to a performed test this new program provides the same results as a program created with Simulink. Also the interval between two samples is precisely and the CPU load is lower than a Simulink program for the same setup.

The overall conclusion is that it takes some time before the total program is running properly and Scilab is not as user friendly and extensive as Matlab. But even with these disadvantages, for real time applications, Scilab is a good and cheap alternative for Matlab.
7. References

[1] Roberto Bucher and Simone Mannori, *USING SCILAB/SCICOS WITH RTAI-LAB*, Version 0.48


[9] Scilab manuals:
http://scilabsoft.inria.fr/product/man/html/eng


To solve problems articles from internet and newsgroups are used, like:


Appendix A: errors and possible solutions

Installation errors

Kernel (make menuconfig)

- The following error might occur:
  /usr/include/bits/socket.h:305:24: asm/socket.h: No such file or directory
  make[1]: *** [scripts/basic/fixedep] Error 1
  make: *** [scripts_basic] Error 2

  Solution:
  - Make a link to the files needed:
    - Goto /usr/include and remove the broken asm link.
    (or rm /usr/include/asm; rm: remove symbolic link /usr/include/asm)?
  - Make a new link to /usr/src/linux/include/asm-i386
    (ln -s /usr/src/linux/include/asm-i386 /usr/include/asm)
  - Try make menuconfig again.

Kernel (make)

- The following error might occur:
  include/linux/kernel.h:10:20: stdarg.h: No such file or directory
  In file included from include/linux/cpumask.h:76, from include/asm/adeos.h:27, from include/linux/adeos.h:25, from include/linux/sched.h:8, from arch/i386/kernel/asm-offsets.c:7:
  include/linux/kernel.h:82: syntax error before "va_list"
  include/linux/kernel.h:82: warning: function declaration isn't a prototype
  include/linux/kernel.h:85: syntax error before "va_list"
  include/linux/kernel.h:85: warning: function declaration isn't a prototype
  include/linux/kernel.h:88: syntax error before "va_list"
  include/linux/kernel.h:88: warning: function declaration isn't a prototype
  include/linux/kernel.h:92: syntax error before "va_list"
  include/linux/kernel.h:92: warning: function declaration isn't a prototype
  include/linux/kernel.h:102: syntax error before "va_list"
  include/linux/kernel.h:102: warning: function declaration isn't a prototype
  make[1]: *** [arch/i386/kernel/asm-offsets.s] Error 1
  make: *** [arch/i386/kernel/asm-offsets.s] Error 2

  Solution:
  - Copy /usr/local/gcc-3.2.3/lib/gcc-lib/i686-pc-linux-gnu/3.2.3/include/stdlib
to /usr/src/linux-2.6.10-rtai-3.2/include/stdlib.h
Try `make` again

The following error might occur:

drivers/block/ps2esdi.c:42:30: `linux/mca-legacy.h': No such file or directory

drivers/block/ps2esdi.c: In function `cleanup_module':

drivers/block/ps2esdi.c:209: warning: implicit declaration of function `mca_mark_as_unuse'
d'

drivers/block/ps2esdi.c:210: warning: implicit declaration of function `mca_set_adapter_procfn'

drivers/block/ps2esdi.c: In function `ps2esdi_geninit':

drivers/block/ps2esdi.c:300: warning: implicit declaration of function `mca_find_main file included from
/usr/include/sys/types.h:31,
   from /usr/src/rtai/base/math/mathP.h:21,
   from /usr/src/rtai/base/math/e_acos.c:42:
   /usr/include/bits/types.h:31:20: `stddef.h': No such file or directory

In file included from /usr/src/rtai/base/math/mathP.h:21,
   from /usr/src/rtai/base/math/e_acos.c:42:
   /usr/include/sys/types.h:147:20: `stddef.h': No such file or directory

In file included from /usr/include/sys/types.h:266,
   from /usr/src/rtai/base/math/mathP.h:21,
   from /usr/src/rtai/base/math/e_acos.c:42:
   /usr/include/bits/posixtypes.h:50: `size_t'
    /usr/include/bits/posixtypes.h:50: warning: no semicolon at end of struct or union
   /usr/include/bits/posixtypes.h:53: syntax error before "stacksize"

`/usr/include/bits/posixtypes.h:53: warning: type defaults to `int' in declaration of __stacksize'

`/usr/include/bits/posixtypes.h:53: warning: data definition has no type or storage class

`/usr/include/bits/posixtypes.h:54: warning: type defaults to `int' in declaration of `pthread_attr_t'

`/usr/include/bits/posixtypes.h:54: warning: data definition has no type or storage class

make[5]: *** `/usr/src/rtai/base/math/e_acos.o' Error 1
make[4]: *** `[module] `/usr/src/rtai/base/math' Error 2
make[4]: Leaving directory `/usr/src/linux-2.6.10-rtai-3.2'
make[3]: *** `[rtai_math.ko]' Error 2
make[3]: Leaving directory `/usr/src/rtai-3.2/base/math'
make[2]: *** `[all-recursive]' Error 1
make[2]: Leaving directory `/usr/src/rtai-3.2/base'
make[1]: *** `[all-recursive]' Error 1
make[1]: Leaving directory `/usr/src/rtai-3.2'
make: *** `[all]' Error 2pter'

drivers/block/ps2esdi.c:300: `MCA_NOTFOUND' undeclared (first use in this function)

drivers/block/ps2esdi.c:300: (Each undeclared identifier is reported only once

drivers/block/ps2esdi.c:300: for each function it appears in.)

drivers/block/ps2esdi.c:311: warning: implicit declaration of function `mca_set_adapter_name'

drivers/block/ps2esdi.c:317: warning: implicit declaration of function `mca_mark_as_used'

drivers/block/ps2esdi.c:318: `MCA_ProcFn' undeclared (first use in this function)

drivers/block/ps2esdi.c:318: syntax error before "ps2esdi_getinfo"

drivers/block/ps2esdi.c:335: warning: implicit declaration of function `mca_read_stored_p
Appendix A

include/asm/mca_dma.h: At top level: drivers/block/ps2esdi.c:267: warning: `ps2esdi_getinfo' defined but not used
make[2]: *** [drivers/block/ps2esdi.o] Error 1
make[1]: *** [drivers/block] Error 2
make: *** [drivers] Error 2

Solution:
- Copy /usr/src/linux-2.6.10-rtai-3.2/include/mca-legacy.h to /usr/src/linux-2.6.10-rtai-3.2/include/linux/mca-legacy.h (cp /usr/src/linux-2.6.10-rtai-3.2/include/mca-legacy /usr/src/linux-2.6.10-rtai-3.2/include/linux/mca-legacy.h)
- Try make again.

Kernel (restarting using the new kernel)

The following error might occur (when restarting the computer using the new kernel):
  Kernel panic: VFS: Unable to mount root fs
Solution:
- Check whether File systems >> Reiserfs support is included in the configuration of the kernel.

RTAI (make)

The following error might occur:
In file included from /usr/include/sys/types.h:31, from /usr/src/rtai/base/math/mathP.h:21, from /usr/src/rtai/base/math/e_acos.c:42: /usr/include/bits/types.h:31:20: stddef.h: No such file or directory
In file included from /usr/src/rtai/base/math/mathP.h:21, from /usr/src/rtai/base/math/e_acos.c:42: /usr/include/sys/types.h:147:20: stddef.h: No such file or directory
In file included from /usr/include/sys/types.h:266, from /usr/src/rtai/base/math/mathP.h:21, from /usr/src/rtai/base/math/e_acos.c:42: /usr/include/bits/pthreadtypes.h:50: syntax error before "size_t"
/usr/include/bits/pthreadtypes.h:50: warning: no semicolon at end of struct or union
/usr/include/bits/pthreadtypes.h:53: syntax error before "/size_t"
/usr/include/bits/pthreadtypes.h:53: warning: type defaults to 'int' in declaration of '__stacksize'
/usr/include/bits/pthreadtypes.h:53: warning: data definition has no type or storage class
/usr/include/bits/pthreadtypes.h:54: warning: type defaults to 'int' in declaration of 'pthread_attr_t'
/usr/include/bits/pthreadtypes.h:54: warning: data definition has no type or storage class
make[5]: *** [/usr/src/rtai/base/math/e_acos.o] Error 1
make[4]: *** [_module_/usr/src/rtai/base/math] Error 2
make[4]: Leaving directory /usr/src/linux-2.6.10-rtai-3.2'
make[3]: *** [rtai_math.ko] Error 2
make[3]: Leaving directory /usr/src/rtai-3.2/base'
make[2]: *** [all-recursive] Error 1
make[2]: Leaving directory /usr/src/rtai-3.2/base'

Appendix A
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/usr/src/rtai-3.2'
make: *** [all] Error 2

- **Solution:**
  - Copy `/usr/local/gcc-3.2.3/lib/gcc-lib/i686-pc-linux-gnu/3.2.3/include/stddef.h`
to `/usr/include/stddef.h`
  - (cp `/usr/local/gcc-3.2.3/lib/gcc-lib/i686-pc-linux-gnu/3.2.3/include/stddef.h`
    `/usr/include/stddef.h`)

**Scilab** (/configure)

- **If the installation script cannot find Fortran** make sure Fortran is installed and
  if so make a link to the Fortran (g77) compiler in `/usr/bin` (applicable to
  TUeDACS cd):
  - Link `/usr/local/gcc-3.2.3/bin/g77` to `/usr/bin/g77`
    (ln `/usr/local/gcc-3.2.3/bin/g77` `/usr/bin/g77`)

- **When Tcl and Tk are not installed** or the shared libraries are missing it is
  necessary to (re)install Tcl and Tk:
  - download and untar `tcl8.4.11-src.tar.gz` and `tk8.4.11-src.tar.gz` to `/usr/src`
  - Tcl:
    - `cd /usr/src/tcl8.4.11/unix`
    - `./configure --enable-shared`
    - `make`
    - `make install`
  - Tk:
    - `cd /usr/src/tk8.4.11/unix`
    - `./configure --enable-shared`
    - `make`
    - `make install`

**Scilab** (make all)

- **The following error might occur:**
  
  In file included from `../graphics/Math.h:12`,
  from `../stack-c.h:8`,
  from `javasciGlobals.h:16`,
  from `javasciGlobals.c:1`:
  
  /usr/include/stdlib.h:583: syntax error before "__size"
  /usr/include/stdlib.h:739: syntax error before "size_t"
  /usr/include/stdlib.h:743: syntax error before "size_t"
  /usr/include/stdlib.h:812: syntax error before "size_t"
  /usr/include/stdlib.h:815: syntax error before "size_t"
  /usr/include/stdlib.h:819: syntax error before "size_t"
  /usr/include/stdlib.h:822: syntax error before "size_t"
  /usr/include/stdlib.h:830: syntax error before "size_t"
  /usr/include/stdlib.h:833: syntax error before '/*' token
  /usr/include/stdlib.h:837: syntax error before "wchar_t"
  /usr/include/stdlib.h:841: syntax error before "mbstowcs"

Appendix A

 SOLUTION:

 Do not use (not essential [1]) Java components:

 /configure --without-java

 make all

Rtai-lab (make install)

 The following error might occur:

 Starting Compilation
 Running Scilab
 .../../bin/scilex: error while loading shared libraries:
 libtk8.4.so: cannot open shared object file: No such file or directory
 generating lib and names
 Running Scilab
 .../../bin/scilex: error while loading shared libraries:
 libtk8.4.so: cannot open shared object file: No such file or directory
 End of compilation
 .../../util/Mak2VCMak Makefile
 .../../util/Mak2ABSMak Makefile
 make[1]: Leaving directory `/usr/local/scilab-3.1.1/macros/RTAI'

 Solution:

 Cannot find libtk8.4.so that comes with tk (should be in folder /usr/local/lib), reinstall tk or when tk is just installed run ldconfig and try again.

 The following error might occur:

 Starting Compilation
 Running Scilab
 .../../bin/scilex: error while loading shared libraries:
 libg2c.so.0: cannot open shared object file: No such file or directory
 generating lib and names
 Running Scilab
 .../../bin/scilex: error while loading shared libraries:
 libg2c.so.0: cannot open shared object file: No such file or directory
 End of compilation
 make[1]: Leaving directory `/usr/local/scilab-3.1.1/macros/RTAI'

 Solution:

 Cannot find libg2c.so and libg2c.so.0. These files are situated in the folder /usr/local/gcc-3.2.3/lib. Make a link to these files in /usr/local/lib
 (ln -s /usr/local/gcc-3.2.3/lib/libg2c.so /usr/local/lib/libg2c.so
 ln -s /usr/local/gcc-3.2.3/lib/libg2c.so.0 /usr/local/lib/libg2c.so.0) and run ldconfig.
General errors

Starting an xwindow application (xlib)

- The following error might occur:
  Xlib: connection to ":0.0" refused by server
  Xlib: No protocol specified
  Can't open display "":0.0"
- Solution:
  - As normal user (not a super user) type:
    \texttt{xhost +}

Running Scilab

- When running Scilab for the first time the following error might occur in the Scilab window:
  File SCI/macros/mtlb/lib does not exist or read access denied
  at line 42 of exec file called by:
  exec('SCI/scilab.star',-1);
- Solution:
  - In the Scilab window type \texttt{quit} (possible 2 times) to close the program.
  - Maybe something went wrong during the installation. It is possible go to \texttt{cd /usr/local/scilab-3.1.1/macros/mtlb} and type \texttt{make} but when starting Scilab again the same error with a different file can be seen. Maybe this is because the building process hesitated a few times because of errors.
  - One way to successfully solve the problem is to remove all the Scicos files (the source and build directories) and go through the whole process (untar and configure/make) again. This time the build process will not stop because all previous errors will be fixed (use \texttt{./configure --without-java}).

- If the computer is restarted without manually closing Scilab first, Scilab is also executed next time the computer starts. An error will occur in the Scilab window and it is not possible to close Scilab in a normal way.
  - Close Scilab by typing \texttt{quit} (or \texttt{exit}) two times in the Scilab window and restart Scilab (as superuser). Now it will work normal.

Using a universal Actuator/Sensor block (Scicos)

- When using a universal actuator or sensor block (rtai-palette) and trying to modify the code Scicos stops given the following error:
  undefined variable : tt ACTUATOR/SENSOR block
- Solution:
  - The block script looks at variable \texttt{tt} without declaring it:
    \begin{verbatim}
    in: function [ok,tt]=getCode(funam)
    if tt==[] then
    \end{verbatim}
In Scicos declaring a variable \( tt \) using the command \( tt = [] \) seems to work but then it is not possible to change the C code. Maybe this bug is fixed in a next version of RTAI.

**Building a real time program** (RTAI CodGen)

- **The following error message might popup:**
  
  
  
  sorry C file name not defined

  **Solution:**
  
  - When opening/setting parameters of a block the block script generates the C-code and a C-function name for that block. This message occurs when this code is missing, so the solution is to open / set all blocks in the diagram to make sure all C-code is generated.

- **In the Scilab command prompt the following message might occur:**
  
  
  dpd_Pd1.c:6:22: rtai_msg.h: No such file or directory

  **Solution:**
  
  - The compiler can not find the rtai include files. This is awkward because these files are present. To simply solve this error, copy all files from /usr/realtime/include individually to /usr/local/include (do not place them in a subfolder).

**Running a real time program**

- **The following error message might popup:**
  
  
  segmentation fault

  **Solution:**
  
  - Load the RTAI modules
## Appendix B: software download locations

<table>
<thead>
<tr>
<th>Software</th>
<th>Homepage</th>
<th>Download page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesalib-6.3.2.tar.gz</td>
<td><a href="http://mesa3d.sourceforge.net/">http://mesa3d.sourceforge.net/</a></td>
<td><a href="http://sourceforge.net/project/showfiles.php?group_id=3&amp;package_id=2436">http://sourceforge.net/project/showfiles.php?group_id=3&amp;package_id=2436</a></td>
</tr>
<tr>
<td>rtai-3.2.tar.tar</td>
<td><a href="http://www.rtai.org">www.rtai.org</a></td>
<td><a href="https://www.rtai.org/RTAI/">https://www.rtai.org/RTAI/</a></td>
</tr>
<tr>
<td>tcl8.4.11-src.tar.gz / tk8.4.11-src.tar.gz</td>
<td><a href="http://tcl.sourceforge.net/">http://tcl.sourceforge.net/</a></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C: Scicos block template

//########################## EDIT THIS PART
##########################

function [x,y,typ]=FunctionName_CHANGE_THIS(job,arg1,arg2)
    if job == 'set' | job == 'define' //only assign values to variables when needed
        num_inputs          = 1     //the number of input ports
        num_outputs         = 1     //the number of output ports
        num_evtin           = 1     //the number of event input ports
        num_evtout          = 0     //the number of event output ports
        firing              = []    // firing: a vector whose size is equal to the size of evtout> It contains output initial event dates (Events generated before any input event arises). Negative values stands for no initial event on the corresponding port. // Default value: []
        state               = []    // state: column vector, the initial continuous state of the block. Must be [] if no continuous state. // Default value: []
        dstate              = []    // dstate: column vector, the initial discrete state of the block. Must be [] if no discrete state. // Default value: []
        blocktype           = "c"   // blocktype:a character with possible values:
                                //:'c' block output depend continuously of the time.
                                //:'d' block output changes only on input events.
                                //:'z' zero crossing block
                                //:'l' logical block
                                // Default value: "c"
        dep_u               = %t    //dep_u must be true if output depends continuously of the input
        dep_t               = %f    //dep_t must be true if output depends continuously of the time
        dep_ut              = [dep_u, dep_t] // dep_ut:1x 2 vector of boolean [dep_u, dep_t], default value: [%f,%f]
        nzcross             = 0     // nzcross: Number of zero crossing surfaces . // Default value: 0
        nmode               = 0     // nmode: Number of different modes . // Default value: 0
        initial_label       = ""    // label: a character string, used as a label. // Default value: ""
        initial_name        = "BlockName"           //default blockname
        cfunctionname_prefix  = "FunctionName_"  
                                //functionname = cfunctionname_prefix + blockname
        initial_parameters  = [initial_name; '1'; '2'; '3'] //vector of
character strings, initial parameters expressions
// in this example the blockname is the first parameter (default value = initial_name), you can modify the dialog code to change this
// separate parameters with a semicolon (;)!

initial_icon = 'xstringb(orig(1),orig(2),["Block Caption";name],sz(1),sz(2),"fill");'

// vector of character strings, initial
icon definition instructions
// mind the name variable (name is not quoted), a value to this variable is assigned when job == 'plot'

initial_blocksize = [3 3] //2 vector, giving the initial block width and height
end

//########################## END EDIT ############################

x=[];y=[];typ=[];

select job
  case 'plot' then
    graphics=arg1.graphics; exprs=graphics.exprs;
    name=exprs(1)(1);
    standard_draw(arg1)
  case 'getinputs' then
    [x,y,typ]=standard_inputs(arg1)
  case 'getoutputs' then
    [x,y,typ]=standard_outputs(arg1)
  case 'getorigin' then
    [x,y]=standard_origin(arg1)
  case 'set' then
    x=arg1
    model=arg1.model; graphics=arg1.graphics;
    exprs=graphics.exprs;
    while %t do
      //########################## EDIT THIS PART
      description=["'
      'Parameter dialog description' 
      'next line' 
      "];
      // CHOOSE between a xchoices or getvalue dialog and REMOVE the other code or build your own gui dialog

      //##################### xchoice dialog ############################
      // name, default value, options
      l1=list('Name',1,[exprs(1)(1),"change name (after closing this dialog)"]);
      l2=list('Parameter1',evstr(exprs(1)(2)),['option1','option2']);
      l3=list('Parameter2',evstr(exprs(1)(3)),['option1','option2']);
      l4=list('Parameter3',evstr(exprs(1)(4)),['option1','option2','option 3']);
      rep=x_choices(description, list(l1,l2,l3,l4));
      if rep == [] then break,end
      name = exprs(1)(1);
      exprs(1) = string(rep');
      if rep(1) ~= 1
[ok,nametemp]=getValue("","Name:",list("str",1),[name]);
if ok
    name=nametemp;
end
end
exprs(1)(1)= name;

rpar = [];      // rpar:column vector, the vector of
    // floating point block parameters. Must be [] if no floating point
    // parameters.
    // Default value: []

ipar = [exprs(1)(2); exprs(1)(3); exprs(1)(4)];  // ipar:column vector, the vector of integer block parameters. Must be []
    // if no integer parameters.
    // Default value: []

//########### end xchoice dialog ####################
//########### getvalue dialog ####################
[ok, name, param1, param2, param3, allans] = ..
    getvalue(description, ..
    ['Name'; 'Parameter1'; 'Parameter2'; 'Parameter3'], ..
    list('str',1, 'vec',1, 'vec',1, 'vec',1), exprs(1))
if ~ok then break,end
exprs(1)=allans
rpar = [param1; param2];  // rpar:column vector, the vector of
    // floating point block parameters. Must be [] if no floating point
    // parameters.
    // Default value: []

ipar = [param3];  // ipar:column vector, the vector of integer
    // block parameters. Must be [] if no integer parameters.
    // Default value: []

//########### end getvalue dialog ####################
//########################## END EDIT ############################
funam = cfunctionname_prefix + name;

[ok,tt]=getCode(funam)
if ~ok then break,end

[model,graphics,ok]=check_io(model,graphics,ones(num_inputs,
    1), ..
    ones(num_outputs, 1),ones(num_evtin, 1),ones(num_evtout,
    1))
if ok then
    model.sim = list(funam,2004)
    model.in  = ones(num_inputs, 1)
    model.out = ones(num_outputs, 1)
    model.evtin = ones(num_evtin, 1)
    model.evtout = ones(num_evtout, 1)
    model.state = state
    model.dstate = dstate
    model.rpar  = rpar
    model.ipar  = ipar
    model.blocktype = blocktype
    model.firing  = firing
    model.dep_ut = dep_ut
    model.nzcross = nzcross
    model.nmode  = nmode
    x.model = model
    exprs(2) = tt
    graphics.exprs = exprs
    x.graphics = graphics
    break
case 'define' then
model = scicos_model()
model.sim = list(' ',2004)
model.in = ones(num_inputs, 1)
model.out = ones(num_outputs, 1)
model.evtin = ones(num_evtin, 1)
model.evtout = ones(num_evtout, 1)
model.state = state
model.dstate = dstate
model.rpar = []
model.ipar = []
model.blocktype = blocktype
model.firing = firing
model.dep_ut = dep_ut
model.nzcross = nzcross
model.nmode = nmode
model.label = initial_label
exprs=list(initial_parameters,[])
x=standard_define(initial_blocksize,model,exprs,initial_icon)
end
endfunction

function [ok,tt]=getCode(funam)
  //########################## EDIT THIS PART
  #Aquaint the device name you choose using the "gen_dev" utility
  cfunctionsname = 'C_Function_Name'  //the device name you
  measure_input_time = %t  //measure the time in the function
  inp_<cfunctionsname>_input,  //in this function data is
  measure_output_time = %t  //measure the time in the function
  out_<cfunctionsname>_output,  //in this function data is
  textmp = '#ifndef MODEL'
  textmp($+1),'#include <math.h>'
  textmp($+1),'#include <stdlib.h>'
  textmp($+1),'#include <scicos/scicos_block.h>'
  textmp($+1),'#endif'
  textmp($+1)='';
  if (measure_input_time | measure_output_time) textmp($+1)='  
  includes <rtai_msg.h>', end
  textmp($+1)='void '+funam+'(scicos_block *block,int flag)';
  textmp($+1)='{'
  textmp($+1)='  #ifdef MODEL'
  if measure_input_time textmp($+1)='  static int mi_count;'
  if measure_output_time textmp($+1)='  static int mo_count;'
  textmp($+1)='  static RTIME mi_time, mi_totaltime;'
  if measure_input_time textmp($+1)='  static RTIME mo_time, mo_totaltime;'
  else
  textmp($+1)='  static double y[' + string(num_outputs) + ']';
  textmp($+1)='  static double u[' + string(num_inputs) + ']';
  textmp($+1)='  static RTIME t = get_scicos_time();'
  textmp($+1)='  static int port;'

Appendix C
```c
switch(flag) {
  case 4:
    mi_count = 0;
    mi_time = 0;
    mi_totaltime = rt_get_cpu_time_ns();
    mo_count = 0;
    mo_time = 0;
    mo_totaltime = rt_get_cpu_time_ns();
    port = inp_' + cfunctionsname + '_init();
    port = out_' + cfunctionsname + '_init();
    break;

  case 1:
    mi_count++;
    mi_time = mi_time - rt_get_cpu_time_ns();
    inp_' + cfunctionsname + '_input(port, y, t);
    for (i=0;i<' + string(num_outputs) + ';i++)
      block->outptr[i][0] = y[i];
    break;

  case 2:
    for (i=0;i<' + string(num_inputs) + ';i++)
      u[i]=block->inptr[i][0];
    if measure_output_time
      mo_count++;
    mo_time = mo_time - rt_get_cpu_time_ns();
    break;

  case 3:
    break;
}
```
Appendix C

//############ END BLOCK INPUT to C-FILE/HARDWARE OUTPUT CODE
######

//############ FINALIZATION CODE
#############################

textmp($+1)='  case 5:

  //function calls

  textmp($+1)='    inp_' + cfunctionsname + '_end(port);
  textmp($+1)='    out_' + cfunctionsname + '_end(port);

  //end function calls

  if measure_input_time
    len = (16 - length(cfunctionsname)) / 2; s1 = []; for i=0:len-1
    s1=s1+"="; end; s2 = []; for i=0:len-0.5 s2=s2+"="; end;
    textmp($+1)='    printf("\n//=' + s1 + ' = ' + cfunctionsname +
      ' blk-input >> c-input >> blk-output measurement times ==' + s2 + '=
      "| Total runtime in seconds: %16f ||
     
     printf("Number of function calls:
      |\n\n||    Total runtime in seconds: %16f ||
     
     textmp($+1)='    printf("||    Number of function calls:
      |\n\n||    Total runtime in seconds: %16f ||
     
     end

  if measure_output_time
    len = (16 - length(cfunctionsname)) / 2; s1 = []; for i=0:len-1
    s1=s1+"="; end; s2 = []; for i=0:len-0.5 s2=s2+"="; end;
    textmp($+1)='    printf("\n//=' + s1 + ' = ' + cfunctionsname +
      ' blk-input >> c-output measurement times ==' + s2 + '=
      "|| Total runtime in seconds: %16f ||
     
     printf("Number of function calls:
      |\n\n||    Total runtime in seconds: %16f ||
     
     end

  textmp($+1)='    break;

  //############ END FINALIZATION CODE
#############################

  textmp($+1)='  }
  textmp($+1)='#endif'
  textmp($+1)'}

//################### END EDIT #####################

tt=textmp
ok = %t
endfunction
Appendix D: source codes examples

Translated td_outports.c file

/*
 * td_outports (Matlab 6 version)
 * (c) Rene' van de Molengraft, 2002 - 2004
 * last update: April, 18th, 2002
 * July, 9th, 2003
 * December, 11th, 2003: encoder function select
 * December, 12th, 2003: adc input range select
 * December, 14th, 2003: individual channel select
 * December, 17th, 2003: pwm support
 * August, 26th, 2004: in- and outports separated
 * again (no cached data anymore...)
 * August, 30th, 2004: do improved
 * December, 13th, 2004: hoh support
 *
 * Inputs : u[0], u[1] = da channels 0 and 1
 * u[2], u[3], u[4], u[5] = do bits 4, 5, 6, 7
 * u[6], u[7] = pwm duty cycle channels 0 and 1
 */

#define int_T   int
#define real_T  double
#include <stdlib.h>
#include <stdio.h>
#define TD_DIRECT       0

/* prototypes */
int td_init(void);
int td_exit(void);
int td_dac_set_enable_chan(int, int, int);
int td_dac_set_filter_order_chan(int, int, int, int);
int td_pwm_start_chan(int, int, int);
int td_pwm_stop_chan(int, int, int);
int td_pwm_set_mode_chan(int, int, int, int);
int td_dac_write_chan(double, int, int, int);
int td_do_set_enable(int, int);
int td_do_select_bit(int, int, int);
int td_do_write_bit(double, int, int, int);
int td_dac_write_chan(double, int, int, int);

void td_exit_td_outports(void)
{
    td_exit();
    printf( "td_exit called using atexit (td_outports.c).\n" );
}

void out_td_outports_init(const int QADID,
                        const int DAC_ACTIVE_CH1, const int
DAC_HOLD_ORDER_SELECT_CH1, const int DAC_ACTIVE_CH2, const int
DAC_HOLD_ORDER_SELECT_CH2,
const int DIO_ACTIVE, const int PWM_ACTIVE_CH1, const int
PWM_FUNCTION_SELECT_CH1, const int PWM_ACTIVE_CH2, const int
PWM_FUNCTION_SELECT_CH2,
const int PWM_FREQ_SELECT_CH1, const int PWM_FREQ_SELECT_CH2)
{
    extern int td_already_initialized;
    int nDevs = 0;
    if (td_already_initialized == 0) {
        td_already_initialized = 1;
        printf("Initializing TueDACS/1 devices...
(td_outports.c)\n");
        nDevs=td_init();
        if (nDevs>10) {
            printf("TueDACS error: devices could not be
initialized (td_outports.c)\n");
td_already_initialized = -1;
            exit(EXIT_FAILURE);
        }
        atexit( td_exit_td_outports );
        printf("TueDACS/1 devices ready (td_outports.c)\n");
    }
    int_T ilink,i;
    real_T
    pwm_freq[8]={1.0,10.0,50.0,100.0,1000.0,10000.0,100000.0,100000.0};
    ilink= QADID -1;
    if (DAC_ACTIVE_CH1==1) {
        //              enable channel
        td_dac_set_enable_chan(1,0,ilink);
        td_dac_set_filter_order_chan(DAC_HOLD_ORDER_SELECT_CH1,0,ilink,TD_DIRECT);
    } else {
        //              disable channel
        td_dac_set_enable_chan(0,0,ilink);
    }
    if (DAC_ACTIVE_CH2==1) {
        //              enable channel
        td_dac_set_enable_chan(1,1,ilink);
        td_dac_set_filter_order_chan(DAC_HOLD_ORDER_SELECT_CH2,1,ilink,TD_DIRECT);
    } else {
        //              disable channel
        td_dac_set_enable_chan(0,1,ilink);
    }
    if (DIO_ACTIVE==1) {
        //              enable channels
        td_do_set_enable(1,ilink);
        for (i=0;i<4;i++) {
            td_do_select_bit(4+i,ilink,0);
        }
    } else {
        //              disable channels
        td_do_set_enable(0,ilink);
    }
    if (PWM_ACTIVE_CH1==1) {
        //              enable channels
        td_pwm_set_enable_chan(1,0,ilink);
        for (i=0;i<8;i++) {
            td_do_set_enable(1,ilink);
        }
    } else {
        //              disable channels
        td_do_set_enable(0,ilink);
    }
    if (PWM_ACTIVE_CH2==1) {
        //              enable channels
        td_pwm_set_enable_chan(1,0,ilink);
        td_pwm_write_freq_chan(pwm_freq[PWM_FREQ_SELECT_CH1-1],ilink,0,0);
    }
void out_td_outports_output(double * u, const int QADID) {
#define U(x) u[x]
    int_T icnt, ilink;
    ilink=QADID-1;
    /* write outputs */
    /*
     * only assign values to variables when needed
     */
    for (icnt=0;icnt<3;icnt++) {
        td_do_write_bit(U(2+icnt),icnt+4,ilink,TD_DIRECT+1);
    }
    td_dac_write_chan(U(0),0,ilink,TD_DIRECT);
    td_dac_write_chan(U(1),1,ilink,TD_DIRECT);
    td_pwm_write_duty_chan(U(6),0,ilink,TD_DIRECT);
    td_pwm_write_duty_chan(U(7),1,ilink,TD_DIRECT);
}
void out_td_outports_end(const int QADID) {
    int_T ilink;
    ilink=QADID-1;
    td_dac_write_chan(0.0,0,ilink,TD_DIRECT);
    td_dac_write_chan(0.0,1,ilink,TD_DIRECT);
}

Modifying the td_outports.sci script

- Change FunctionName to rtai_td_outports.
- Change the definitions to:

```
if job == 'set' | job == 'define' //only assign values to variables when needed
    num_inputs          = 8 //the number of input ports
    num_outputs         = 0 //the number of output ports
    num_evtin           = 1 //the number of event input ports
    num_evtout          = 0 //the number of event output ports
    firing              = [] // firing: a vector whose size is equal to the size of evtout. It contains output initial event dates (events generated before any input event arises). Negative values stands for no initial event on the corresponding port. Default value: []
    state               = [] // state: column vector, the initial continuous state of the block. Must be [] if no continuous state. Default value: []
    dstate              = [] // dstate: column vector, the initial discrete state of the block. Must be [] if no discrete state. Default value: []
    blocktype           = "d" // blocktype: a character with possible values:
                           // :'c' block output depend continuously of the time.
                           // :'d' block output changes only on input events.
                           // :"z" zero crossing block
                           // Default value: "c"
    dep_u               = %t // dep_u must be true if output depends continuously of the input
    dep_t               = %f // dep_t must be true if output depends continuously of the time
    dep_ut              = [dep_u, dep_t] // dep_ut: lx 2 vector of boolean [dep_u, dep_t], default value: [%f,%f]
    nzcross             = 0 // nzcross: Number of zero crossing surfaces. Default value: 0
    nmode               = 0 // nmode: Number of different modes. Default value: 0
    initial_label       = "" // label: a character string,
```
used as a label.

// Default value: ""
initial_name = "td_outports1"  //default blockname
cfunctionname_prefix = "td_outports."  //functionname = functionname_prefix + blockname

//vector of character strings, initial parameters expressions
//in this example the blockname is the first parameter
//default value = initial_name), you can modify the dialog code to change this
//separate parameters with a semicolon (;)
initial_parameters = [initial_name;'1';'1';'0';'1';'0';'0';'1';'6';'0';'1';'6']

//vector of character strings, initial icon definition instructions
//mind the name variable (name is not quoted), a value to this variable is assigned when job == 'plot'
initial_icon = ['xstringb(orig(1),orig(2),[''dac 1'';''dac 2'';''bit 4'';''bit 5'';''bit 6'';''bit 7'';''pwm 1'';''pwm 2''],sz(1)*0.3,sz(2),''fill'');''xstringb(orig(1)+sz(1)*0.3,orig(2),[''TUeDACS/1 QAD/AQI'';''Outports block'';name],sz(1)*0.7,sz(2)*1.15,''fill'');']

//2 vector, giving the initial block width and height
initial_blocksize = [6 5]

end

//########################## END EDIT ############################

• Change dialog code to:

function [ok,tt]=getCode(funam)

% Change dialog code to:

//########################## EDIT THIS PART ############################

description= 'This block models the TuE DAC/1 QAD/AQI outputs.';

'QAD:';
- Front panel channels are named 0/1 instead of 1/2';

'DAC output range is +/- 2.5 V';

'AQI:';
- DAC output range is +/- 5V';
- Scalable ADC input range';

// CHOOSE between a xchoices or getvalue dialog, REMOVE the other code or build your own gui dialog

end


• The last part that will be modified is the block code generation part, change it to:

function [ok,tt]=getCode(funam)
choose using the "gen_dev" utility
measure_input_time = %t
//measure the time in the
function inp_<cfunctionsname>_input
 transferred from you c-file / hardware input to scicos block output ports
measure_output_time = %t
//measure the time in the
function out_<cfunctionsname>_output,
//in this function data is
transferred from scicos block input to your c-file / hardware output ports

#include <math.h>
#include <stdlib.h>
#include <scicos/scicos_block.h>
#include <rtai_msg.h>

void '+funam+'(
scicos_block *block,
int flag);
{
  #ifdef MODEL
  switch(flag) {
  case 4:
    if measure_input_time
      mi_count = 0;
    mi_time = 0;
    mi_totaltime = rt_get_cpu_time_ns();
    if measure_output_time
      mo_count = 0;
    mo_time = 0;
    mo_totaltime = rt_get_cpu_time_ns();
    out_' + cfunctionsname + '_init(' + quad_params + ');' +
    //function calls
    //end function calls
    #ifdef MODEL
    break;
  }

  case 1:
    if measure_input_time
      mi_count++;;
    mi_time = mi_time - rt_get_cpu_time_ns();
    if measure_output_time
      mo_count++;;
    mo_time = mo_time - rt_get_cpu_time_ns();
    break;

  case 2:
    for (i=0;i< string(num_inputs) + ';i++)
      u[i]=block->inptr[i][0];
    if measure_output_time
      mo_count++;;
    mo_time = mo_time - rt_get_cpu_time_ns();
    break;

  case 5:
    out_' + cfunctionsname + '_end(' + QADID + ');' +
    //function calls
    //end function calls
  }
  #ifdef MODEL

  //############ INITIALIZATION CODE
  #ifdef MODEL
    mi_count = 0;
    mi_time = 0;
    mi_totaltime = rt_get_cpu_time_ns();
    if measure_output_time
      mo_count = 0;
    mo_time = 0;
    mo_totaltime = rt_get_cpu_time_ns();
  #ifdef MODEL
    out_' + cfunctionsname + '_init(' + quad_params + ');' +
    //function calls
    //end function calls
    #ifdef MODEL
    break;
  }

  //end initialization code

//end function calls

if measure_input_time
  mi_time = mi_time +
  rt_get_cpu_time_ns();',
  textmp($+1)='break;
}
Generated functions of the DCT PD and the Ref3 block

These two functions are generated by the two block scripts. They are included to show the constructions used in those blocks.

```c
#include <math.h>
#include <stdlib.h>
#include <scicos/scicos_block.h>
#include <rtai_msg.h>
void dpd_Pd1(scicos_block *block, int flag)
{
    static int mi_count;
    static RTIME mi_time, mi_totaltime;
    static int mo_count;
    static RTIME mo_time, mo_totaltime;
    int i;
    double y[1];
    double u[1];
    static double * rwrkpr;
    switch(flag) {
    case 4:
        mi_count = 0;
        mi_time = 0;
        mi_totaltime = rt_get_cpu_time_ns();
        mo_count = 0;
        mo_time = 0;
        rwrkpr = out_dpd_init();
        printf("\\========================================================\n\n//======== dpd c-input >> blk-output measurement times =========\\
||    Total runtime in seconds:   %16f  ||
", (rt_get_cpu_time_ns() - mi_totaltime) / 1e9 );
        printf("||    Number of function calls:   %16d  ||
", mi_count);
        if (mi_count != 0) { printf("||    Mean time (ns) in function: %16d  ||
", mi_time / mi_count );}
        printf("\\========================================================\n\n//======== dpd blk-input >> c-output measurement times =========\\
||    Total runtime in seconds:   %16f  ||
", (rt_get_cpu_time_ns() - mo_totaltime) / 1e9 );
        printf("||    Number of function calls:   %16d  ||
", mo_count);
        if (mo_count != 0) { printf("||    Mean time (ns) in function: %16d  ||
", mo_time / mo_count );}
        break;
    case 5:
        out_dpd_end(rwrkpr);
        printf("\\========================================================\n\n//########### END FINALIZATION CODE
#

#ifndef MODEL
#include <math.h>
#include <stdlib.h> #include <scicos/scicos_block.h>
#endif
#include <rtai_msg.h>
void ref3b_ref3_1(scicos_block *block, int flag)
{
    static int mi_count;
    static RTIME mi_time, mi_totaltime;
    static int mo_count;
    static RTIME mo_time, mo_totaltime;
    int i;
    double y[3];
    double u[1];
    double t = get_scicos_time();
    static int port;
    double * temp;
    switch(flag) {
    case 1:
        mi_count++;
        mi_time = mi_time - rt_get_cpu_time_ns();
        for (i=0; i<3; i++) u[i] = block->inptr[i][0];
        out_dpd_output(u, y, rwrkpr, block->rpar[0], block->rpar[1]);
        mi_time = mi_time + rt_get_cpu_time_ns();
        for (i=0; i<3; i++) block->outptr[i][0] = y[i];
        break;
    case 2:
        mo_count++;
        mo_time = mo_time - rt_get_cpu_time_ns();
        mo_time = mo_time + rt_get_cpu_time_ns();
        break;
    case 3:
        out_dpd_end(rwrkpr);
        printf("\\========================================================\n\n//======== dpd c-input >> blk-output measurement times =========\\
||    Total runtime in seconds:   %16f  ||
", (rt_get_cpu_time_ns() - mi_totaltime) / 1e9 );
        printf("||    Number of function calls:   %16d  ||
", mi_count);
        if (mi_count != 0) { printf("||    Mean time (ns) in function: %16d  ||
", mi_time / mi_count );}
        printf("\\========================================================\n\n//======== dpd blk-input >> c-output measurement times =========\\
||    Total runtime in seconds:   %16f  ||
", (rt_get_cpu_time_ns() - mo_totaltime) / 1e9 );
        printf("||    Number of function calls:   %16d  ||
", mo_count);
        if (mo_count != 0) { printf("||    Mean time (ns) in function: %16d  ||
", mo_time / mo_count );}
        printf("\\========================================================\n\n//########### END FINALIZATION CODE
#
```

Appendix D 9

Appendix D 10
case 4:
mi_count = 0;
mi_time = 0;
mi_total_time = rt_get_cpu_time_ns();
mo_count = 0;
mo_time = 0;
mo_total_time = rt_get_cpu_time_ns();
temp = calloc(18, sizeof(double));

//#### define parameters: ####
temp[0] = 0;
temp[1] = 500;
temp[2] = 0;
temp[3] = 0;
temp[4] = 11.0081;
temp[5] = -1;
temp[6] = 500;
temp[7] = 0;
temp[8] = 0;
temp[9] = 49.95;
temp[10] = 49.95;
temp[11] = 0;
temp[12] = 1.0081;
temp[13] = 1.0081;
temp[14] = 0;
temp[15] = 615;
temp[16] = 615;
temp[17] = 0;

//############################
port = out_ref3b_init("ref3_1", 3, temp);
free(temp);
break;

case 1:
mi_count++;
mi_time = mi_time - rt_get_cpu_time_ns();
for (i=0;i<1;i++) u[i] = block->inptr[i][0];
out_ref3b_input(port, y, t, u);
mi_time = mi_time + rt_get_cpu_time_ns();
for (i=0;i<3;i++) block->outptr[i][0] = y[i];
break;

case 2:
mo_count++;
mo_time = mo_time - rt_get_cpu_time_ns();
mo_time = mo_time + rt_get_cpu_time_ns();
break;

case 3:
out_ref3b_end(port);
print("//### ref3b c-input >> blk-output measurement times =\n");
print("| Total runtime in seconds: %16f
| (rt_get_cpu_time_ns() - mi_total_time) / 1e9 |");
print("| Number of function calls: %16d
| mi_count |");
if (mi_count != 0) { printf("| Mean time (ns) in function:
| %16d
| mi_time / mi_count |");}
print("\n");

print("\n\n#-------------------------------------#\n\n\n//### ref3b blk-input >> c-output measurement times =\n");
print("| Total runtime in seconds: %16f
| (rt_get_cpu_time_ns() - mo_total_time) / 1e9 |");
print("| Number of function calls: %16d
| mo_count |");
if (mo_count != 0) { printf("| Mean time (ns) in function:
| %16d
| mo_time / mo_count |");}
print("\n\n#-------------------------------------#\n\n\nbreak;
Appendix E: contents of CD

In order use the new blocks that were written a CD is included. This CD contains all needed files to use the new blocks and the pato01 model. Also all files needed to install RTAI/Scilab and some useful documents are included.

The CD contains the following files:

<table>
<thead>
<tr>
<th>Report.doc and Report.pdf</th>
<th>this report</th>
</tr>
</thead>
<tbody>
<tr>
<td>New blocks/macros (folder)</td>
<td>the new Scicos blocks</td>
</tr>
<tr>
<td>New blocks/devices (folder)</td>
<td>c source files</td>
</tr>
</tbody>
</table>

Scicos files – pato01 (folder):

<table>
<thead>
<tr>
<th>pato01.cos</th>
<th>the pato01 model</th>
</tr>
</thead>
<tbody>
<tr>
<td>pato01</td>
<td>pato01 real program</td>
</tr>
<tr>
<td>New Blocks.cos</td>
<td>a palette containing all new blocks, in Scicos use load/save as palette to include it</td>
</tr>
<tr>
<td>scicos_block_template.cos</td>
<td>template written in order to easy create new blocks</td>
</tr>
<tr>
<td>rl and rr</td>
<td>scripts to load and unload (remove) the RTAI modules (rtai load and rtai remove)</td>
</tr>
<tr>
<td>pato01_rtai (folder)</td>
<td>source files of pato01 (created by the RTAI code generator)</td>
</tr>
</tbody>
</table>

Software (folder):

<table>
<thead>
<tr>
<th>rtai-3.2-modified to use tuedacs devices.tar.gz</th>
<th>rtai-3.2 package including the new blocks, for changes see below.</th>
</tr>
</thead>
<tbody>
<tr>
<td>other files</td>
<td>source files needed for installation</td>
</tr>
</tbody>
</table>

Useful documents (folder) Some useful pdf files

In `rtai-3.2-modified to use tuedacs devices.tar.gz` the following files are added or modified (*):

- `rtai-3.2/rtai-lab/scilab/rtmain.c *`
- `rtai-3.2/rtai-lab/scilab/macros/Makefile *`
- `rtai-3.2/rtai-lab/scilab/macros/RTAI/Makefile *`
- `rtai-3.2/rtai-lab/scilab/macros/RTAI/RTAICODEGEN.sci *`
- `rtai-3.2/rtai-lab/scilab/macros/RTAI/scicos_block_template.sci`
- `rtai-3.2/rtai-lab/scilab/macros/RTAI/td_outports.sci`
- `rtai-3.2/rtai-lab/scilab/macros/RTAI/td_inports.sci`