ISSN 0280-5316
ISRN LUTFD2/TFRT--5799--SE

TrueTime in Scicos

Daniel Kusnadi

Department of Automatic Control
Lund University
June 2007

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* June 2007 *Document Number* ISRNLUTFD2/TFRT--5799--SE
Author(s) Daniel Kusnadi	*Supervisor* Karl-Erik Årzén at Automatic Control in Lund Anton Cervin at Automatic Control in Lund (Examiner) *Sponsoring organization*

Title and subtitle
TrueTime in Scicos (TrueTime I Scicos)

Abstract

TrueTime is a MATLAB/Simulink-based simulator for real-time control systems that has been developed at the Department of Automatic Control, LTH. The aim of this thesis is to port the TrueTime kernel block to the open-source Scilab/Scicos simulation environment. The original kernel block is a variable-step, discrete S-function written in C++. In this work, the Scicos engine should replace the Simulink engine that is being used for timing and interfacing with the rest of the simulation model. The scope of work does not include the network related implementation in TrueTime kernel block, since the TrueTime network block is not part of this thesis project. Neither is the implementation of the user interface is included, since Scilab is planning to reorganize the implementation on its graphical editor. The results show that it is possible to port TrueTime to the Scilab/Scicos environment, although the developed Scilab version does not provide as many simulation features as the MATLAB version. The simulation outputs from tests of TrueTime in Scilab/Scicos give the same results as those simulated in MATLAB/Simulink.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316	*ISBN*

Language English	*Number of pages* 72	*Recipient’s notes*
Security classification		

http://www.control.lth.se/publications/

Contents

1 Introduction

1.1 Aim of the thesis

The aim of the thesis is to port the TrueTime kernel block to the Scilab/Scicos simulation environment. The original block is a variable-step, discrete, MATLAB S-function written in C++. The Scicos engine shall replace the Simulink engine that is being used for timing and interfacing with the rest of the model (the continuous dynamics).

Two main problems to be solved in this thesis are:

- To translate as many TrueTime functions as possible into the Scicos/Scilab environment
- To make Scilab/Scicos collaborate with the translated TrueTime application and produce as similar results as possible compared to MATLAB/Simulink.

The output figures of the provided process control examples from TrueTime will be used to measure the result of this porting work.

Those two problems are important to open the possibilities to port other MATLAB/Simulink based tool blocks, so that the tool can also be accessible by the Scilab user, with consideration that Scilab is a free software package and has a quite strong market. In other words, solving this problem will give possibilities to expand the market area for the developed tools.

The scope of work does not include the network related implementation in TrueTime kernel block, since the TrueTime network block is not part of this project. Neither the implementation of the user setting interface is included, since Scilab is planning to reorganize the implementation on its graphical editor.

1.2 Results

The results show that it is possible to port the MATLAB/Simulink based tool TrueTime to the Scilab/Scicos environment, although Scilab/Scicos does not provide as many simulation features as MATLAB/Simulink. The simulation outputs from the test of TrueTime on Scilab/Scicos give the same figures as those simulated on MATLAB/Simulink. These figures confirm that the assignments given in this project are accomplished.

Although the user setting interface is not included in the scope of work, it is successfully completed in the last stage of this thesis project. The main purpose is to improve the user friendliness of the setting interface of the ported TrueTime Kernel. As in Simulink, the kernel block is available in a TrueTime library that is accessible from Scicos palettes.

2 Background

2.1 The TrueTime simulator

TrueTime is a MATLAB/Simulink-based simulator written in C++ MEX for real-time control systems. It facilitates the simulations of:

- Controller task execution using a multitasking real-time kernel, where the tasks are controlling processes that are modelled as ordinary Simulink blocks
- Simple models of communication networks and their influence on networked control loops

The TrueTime simulator was developed by Department of Automatic Control, LTH[1], Sweden, and commonly used in:

- Investigation on the effects of timing non-determinism caused, for example, by pre-emption or transmission delays, on control performance.
- Development of compensation schemes that adjust the controller dynamically based on measurements of actual timing variations
- Experiment with new, more flexible approaches to dynamic scheduling, such as feedback scheduling of CPU time and communication bandwidth and quality-of-service (QoS)-based scheduling approaches.
- Simulation of event-driven control system, e.g. engine controllers and distributed controllers.

TrueTime version 1.4 that is used in this project is delivered as a Simulink library, as shown in Figure 1, consisting of the following blocks:

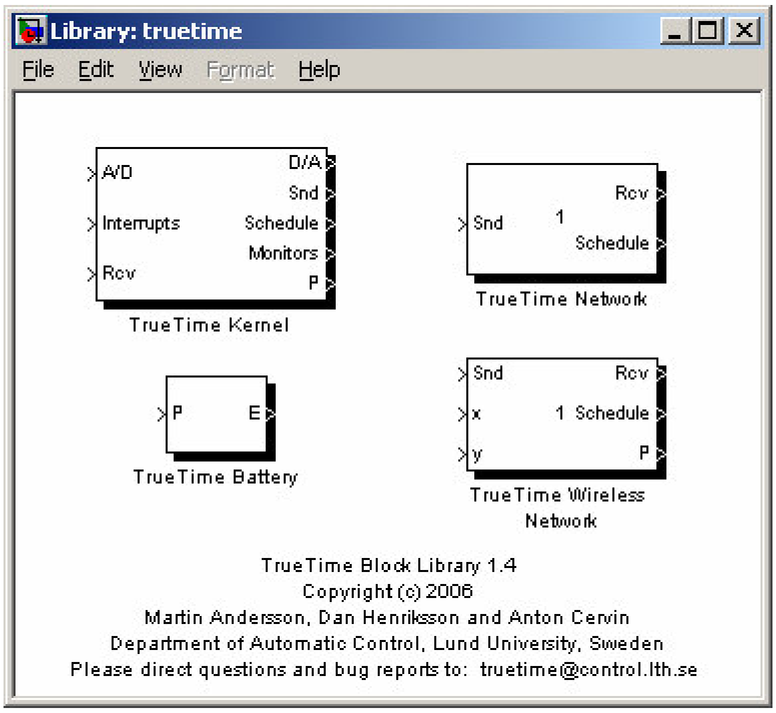
- Kernel block supporting external interruptions, possibility to write tasks as M-files or C++ functions and possibility to call Simulink block diagrams from within the code functions
- Network block with protocol settings available for Ethernet, CAN, TDMA, FDMA, Round Robin, and switched Ethernet networks
- Wireless network block that supports 802.11b WLAN and 802.15.4 ZigBee
- Battery block supporting Dynamic Voltage Scaling

Since the project scope of work is to port only the kernel block, the other three blocks will not be discussed in details. Further information about these blocks can be found in the TrueTime Reference Manual [5]. The porting of the network-related implementation in the kernel block will include only the configuration of network I/O ports (Receive/Send) to maintain the compatibility in future development. In other words, there is no functionality available behind these network I/O ports.

The kernel block works as a computer with a simple but flexible real-time kernel equipped with A/D and D/A converters, a network interface, and external interrupt channels. As common real-time kernels, it maintains data structures for a ready-queue, a time queue, and records for tasks, interrupt handlers, monitors, and timers that have been created for the

[1] Lund Tekniska Högskolan

simulation. The execution of tasks and interrupt handlers is configured by user-defined code functions that can be written in C++ (for speed) or as MATLAB m-files (for ease of use). Control algorithm is modelled graphically using ordinary Simulink block diagram.



TrueTime Block Library 1.4
Copyright (c) 2006
Martin Andersson, Dan Henriksson and Anton Cervin
Department of Automatic Control, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se

Figure 1 - The TrueTime block Library

Figure 1 shows three groups of inputs on the kernel block: A/Ds, Interrupts and Network Receive. The A/Ds are continuous-time inputs with A/D converters and the rest are discrete-time inputs. There is one more discrete-time input for energy supply, but it is hidden and only accessible from the configuration interface window. All outputs are discrete-time signals. The schedulers and monitors display the allocation of kernel resources during the simulation, such as tasks and handlers.

An arbitrary number of tasks can be created to run in TrueTime kernel and they can be both periodic (such as controller and I/O tasks) and a-periodic (such as communication tasks). Each task has a set of attributes and a code function. The attributes include a name, a release time, a worst-case execution time, an execution time budget, relative and absolute deadlines, a priority, and a period (if task is periodic). The priority scheduling is dynamic given as a user-defined priority function. There are four priority functions supported by TrueTime:

- prioFP (fixed priority)
- prioRM (rate monotonic)
- prioDM (deadline monotonic)
- prioEDF (earliest deadline first)

Interrupts can be generated externally or internally. An external interrupt is triggered by signal connected to the corresponding interrupt input channel of the kernel block. Internal interrupt is activated by timers; either periodic timers or one-shot timers. When an interrupt occurs, a user-defined interrupt handler is scheduled to serve the interrupt according to fixed priorities. This interrupt handler has higher priority than a task although it works similarly. It has simpler attribute consisting of a name, a priority and a code function.

The code function mentioned above is a function defining the activities to be performed by the task or the interrupt. This code function is scheduled and executed by the kernel that is prioritized lower than the interrupts but higher than the tasks. The code is normally divided into several segments and it returns the simulated execution time of each segment. The value of the execution time can be constant, random or data-dependent. The kernel calls the code functions with the proper arguments and keeps track of the segment execution. The assignments in a segment are normally built using kernel primitives. Some examples of kernel primitives called by the code functions are listed in Table 1, the rest can be found in the TrueTime documentation [5].

Kernel primitives (called from code functions)	Description
ttCurrentTime()	Get current time in simulation
ttAnalogIn(ch)	Get the value of an input channel
ttAnalogOut(ch, val)	Set value to an output channel
ttCreateTimer(time, ih)	Trigger interrupt handler at a specific time
ttSleepUntil(time)	Wait until a specific time
ttWait(ev)	Await an event
ttSetPriority(val)	Change the priority of a task
ttSetPeriod(val)	Change the period of a task

Table 1 - Examples of kernel primitives

Figure 2 below shows an example of a code function implementing a simple controller. The first segment (indicated by case 1) samples the input signal at channel 1 and computes the control signal. The second segment actuates the control signal to the output channel 1 and updates the controller states. The third segment indicates the end of execution by returning a negative value. The functions *calculateOutput* and *updateState* are assumed to represent the implementation of an arbitrary controller. The data structure *data* represents the local memory of the task and is used to store the control signals and measured variable between calls to different segments. A/D and D/A conversions are performed using kernel primitives: *ttAnalogIn* and *ttAnalogOut*.

```
function [exectime, data] = myControler(seg, data)
switch seg,
  case 1,
    data.y = ttAnalogIn(1);
    data.u = calculateOutput(data.y);
    exectime = 0.002;
  case 2,
    ttAnalogOut(1, data.u);
    updateState(data.y);
    exectime = 0.003;
  case 3,
    exectime = -1; % finished
```

Figure 2 - Example of a simple code function

In this project the code functions are written in C++ instead of MATLAB m-file code, like the example above. The code structure is more or less similar.

2.2 MATLAB/Simulink

MATLAB is a software program package for technical calculations and data visualization. The result from reliable calculations can be presented with advanced graphics. The program package is flexible, user-friendly, and can be applied within, for example, mathematics, physics, engineering, chemistry, biology, economics.

MATLAB is maintained and sold by The MathWorks, Inc., of Natick, Massachusetts, and is available for MS Windows, Macintosh, Unix and Open VMS systems. It was developed from the beginning in the 1970s for applications involving matrices, linear algebra, and numerical analysis. The name MATLAB stands for "Matrix Laboratory". Nowadays, the MATLAB's numerical routines have been thoroughly tested and improved through many years of use in different wide sectors, which in turn contributes the fact that MATLAB's capabilities have been greatly expanded during that time. Below are some wide sectors where MATLAB is used:

- Research and development in industry
- Teaching in mathematics such as linear algebra and numerical analysis, where different calculation methods can be studied and compared in details
- Teaching and research in engineering subjects such as electronics, signal processing and automatic control engineering
- Teaching and research in all subjects where calculation problem or data processing problem exist, for example statistics, economics, physics, chemistry and biology

MATLAB has a number of add-on software modules, called toolboxes that perform more specialized computations for specific application areas, such as digital signal processing and automatic control system design. One of them is Simulink that is used by TrueTime. The algorithm in the software modules is written by the experts on mathematical software and often optimized for the present computer types and applications.

The graphics in MATLAB is object-oriented and provides a powerfull environment to develop advanced graphics in both two and three dimensions. Since MATLAB are both an interactive environment and a matrix/vector-oriented programming language, it is possible to develop user-defined functions that can be used exactly as those built-in commands.

2.3 Scilab/Scicos

Scilab is a free open-source software package for general purpose numerical systems performing numerical computations. It is designed specifically for scientific applications. Scilab has the same market that is dominated by MATLAB. Widely used at universities and engineering schools makes it gaining ground in industrial environments.

The software is available for downloading at http://www.scilab.org. It is in binary format and runs on the main available platforms: Unix/Linux (the main software development is performed on Linux), Windows and MacOSX.

Basile was the original name of Scilab and was developed at INRIA[2] as part of Meta2 project. The development continued under the name of Scilab by the Scilab group, which was a team

[2] Institut National de Recherche en Informatique et en Automatique

of researchers from INRIA Metalau and ENPC[3]. Since 2004, a consortium coordinates the Scilab development.

As it is mainly dedicated to scientific computing, Scilab provides access to large numerical libraries from such areas as linear algebra, numerical integration and optimization. It is possible to extend the Scilab environment by importing new functionalities from external libraries using either static or dynamic links.

One important Scilab toolbox is Scicos (`http://www.scicos.org`), which corresponds to Simulink in MATLAB. Scilab/Scicos is the open-source alternative to commercial packages for dynamical system modelling and simulation packages such as MATLAB/Simulink and MATRIXx/SystemBuild. Scicos provides a modular way to construct and simulate complex dynamical systems using a block-diagram graphical editor. It handles, in particular, the interaction between continuous-time dynamics and system events including events associated with the timing of a discrete-time clock.

Using Scicos, the user can construct a library of reusable modules (blocks) that can be used in different models in different projects. A large number of blocks are already available in Scicos palettes for elementary operations. There are quite many useful functionalities for the designer to optimize model parameters, validate models, generate C code, etc. The most important facility used in this project is linking to a new library constructed and compiled in C++.

Running some of the examples presented in the delivered Scilab product requires C compiler, which is usually no problem under Linux, MacOSX and most Unix workstations. Under windows operating system, Scilab requires the installation of Visual C++. Scilab version 4.1 running on Windows XP Professional SP2 operating system is used during this project.

[3] Ecole Nationale des Ponts et des Chaussées

3 Porting TrueTime to Scilab

3.1 Writing blocks for Simulink and Scicos - a Comparison

This section summarizes important differences between MATLAB/Simulink and Scilab/Scicos in the way to write blocks, to load them, and use them in the models.

3.1.1 Activation signals

The behaviour of simulation block is primarily specified by the way it is activated. There are three ways to activate a block: event activation, continuous-time activation and internal mode/zero-crossing activation.

Event activation

A block with this activation updates its output only when it receives external activation signal. It is illustrated by the formula below for the implementation in Scicos:

$$y(t_e) = f_1(t_e, x(t_e^-), z(t_e^-), u(t_e), \mu(t_e))$$

t_e denotes the event time, $x(t_e^-)$ and $z(t_e^-)$ denote respectively the continuous-time and discrete-time states.

Scicos has different approach compared to Simulink on how to design the simulation model regarding to this matter. One simple and clear example that can describe the difference is the modelling of a Square-Wave Generator block with amplitude 1 and period 1.

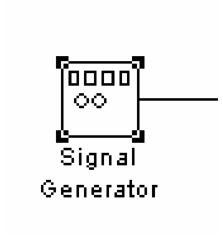
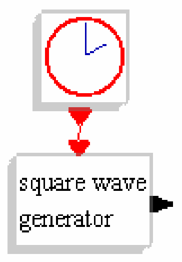
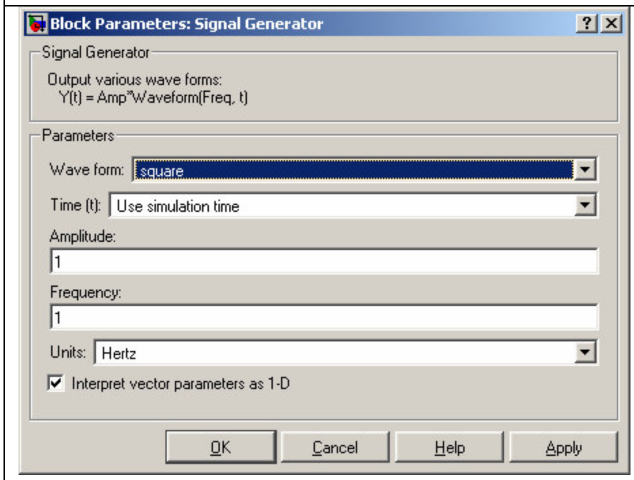
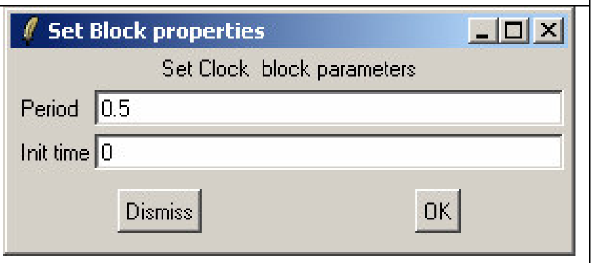
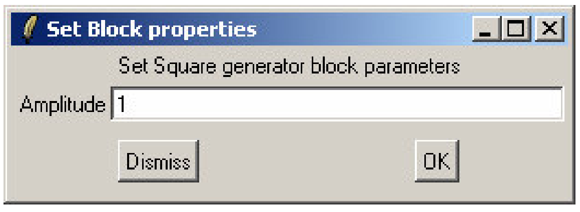
Simulink Approach	Scicos Approach
	
 Block Parameters: Signal Generator Signal Generator Output various wave forms: Y(t) = Amp*Waveform(Freq, t) Parameters Wave form: square Time (t): Use simulation time Amplitude: 1 Frequency: 1 Units: Hertz ☑ Interpret vector parameters as 1-D 	 Set Block properties Set Clock block parameters Period 0.5 Init time 0  Set Block properties Set Square generator block parameters Amplitude 1

Figure 3 – Comparison of Event Activation of Square-Wave Generator Block

In Simulink this can be constructed by a Signal Generator block and completed with those parameters required. In Scicos, two blocks are needed: Square-Wave Generator block with its parameter 'Amplitude' and Event Generator block with its parameters 'Period' and 'Init Time'.

The Scicos block can have event output activation ports, where upon activation the block can program events on the ports by providing the delay time to event firing on each port. If the block is configured to have internal states, it can update them, both continuous-time states and discrete time states. The output activation ports and internal states are not used in these porting activities.

Continuous-time activation

In this case, the activation occurs over time intervals and not at specific times as in the case of event activation. No further discussion on this topic since it is similar as in MATLAB.

Internal mode and zero-crossing

The mode parameter is not part of the Scicos formalism. It is introduced to facilitate the implementation of the numerical solver. The mode is not used in these porting activities.

When zero-crossing occurs inside the block, this block is activated at the time of crossing. To make sure the simulation stops at the time of crossing, a zero-crossing surface is introduced at zero. After the zero-crossing the simulation continues. The computation of the zero-crossing is performed by the block. In Scicos, if the simulation phase is 1, then

$$[m(t), s(t)] = f_9(t, x(t), z(t), u(t), \mu(t))$$

m(t) denotes the mode and the zero-crossing surfaces s(t) may be present even if the block has no mode.

Zero-crossing is the most important feature used in the TrueTime implementation. The zero crossing surface function is defined generally by the following equation:

$$s(t) = nextHit - t$$

This surface function s(t) can be illustrated against the current simulation time t as shown in Figure 4. The value of nextHit is updated and calculated by the kernel at each zero crossing points pointed by the green arrows in the figure.

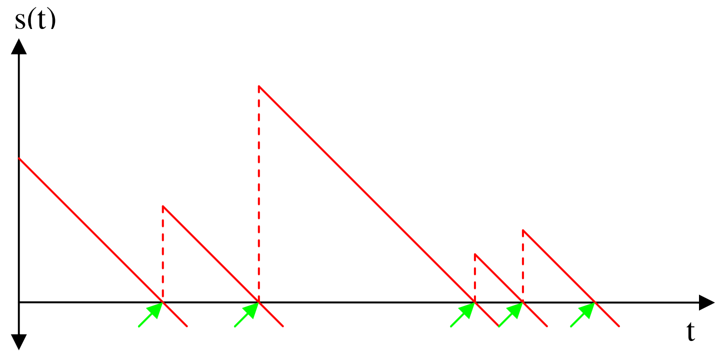


Figure 4 – The zero crossing surface function in TrueTime, the green arrows show the crossing point

The illustration shows that zero crossing is the key for TrueTime to simulate the different execution times of the scheduled tasks and interrupts. The method of zero-crossing between Scilab/Scicos and MATLAB/Simulink is similar, which makes it possible to port the TrueTime implementation into the Scilab environment.

3.1.2 Collaboration with C files

The idea behind the collaboration with C files is quite similar between Scilab and MATLAB but the implementations are slightly different as shown in Figure 5. Both Scilab and MATLAB are written in C and they use pointers to exchange data with the C files.

MATLAB Approach	Scilab Approach
To compile and link C files	
mex –f [options] filename can be used outside MATLAB environment	ilib_for_link(names,files,libs,flag) can only be used inside Scilab environment

Figure 5 – Compiling and linking C files

The function *ilib_for_link* is not used in this project since it compiles only C code. TrueTime code is written in C++ which can not pass the compilation restrictions used in *ilib_for_link*. One approach is done by compiling the TrueTime code using Visual C++ to generate truetime.dll file and then link the .dll file from Scilab using command below:

```
link(myblock_path+'ttkernel.dll',['truetimekernel'],'c');
```

Figure 6 – Scicos command to link the function *truetimekernel* in the file *ttkernel.dll*

The way the block is called in Scicos is characterized by the type of interfacing function. Since the TrueTime program is written in C/C++ language, Type 4 of Scicos block is used. Type 4 C functions receive two arguments: a structure containing block information and a flag.

```
#include "scicos_block.h"
void truetimekernel(scicos_block *block, int flag) {
    ..
}
```

Figure 7 – Main structure of Scicos computational function

3.1.3 Function call structure

The computational function is called in various ways by Scicos simulator and flag is used to indicate the job that must be performed. The valid flags are listed in the table below.

flag	input	output	Description
0	t, nervprt, x, z, inptr, mode, phase	xd	Compute the derivative of continuous time state
1	t, nervprt, x, z, inptr, mode, phase	outptr	Compute the outputs of the block
2	t, nervprt>0, x, z, inptr	x, z	Update states due to external activation
2	t, nervprt=-1, x, z, inptr, jroot	x, z	Update states due to internal zero-crossing
3	t, x, z, inptr, jroot	evout	Program activation output delay times
4	t, x, z	x, z, outptr	Initialize states and other initializations
5	X, z, inptr	x, z, outptr	Final call to block for ending the simulation
6			Not needed

flag	input	output	Description
7			Only used for internally implicit blocks
9	t, phase=1, nervprt, x, z, inptr	g, mode	Compute zero-crossing surfaces and set modes
9	t, phase=2, nervprt, x, z, inptr	g	Compute zero-crossing surfaces

Table 2 – The jobs that the computational function must perform for different flag

Instead of using flag, Simulink distributes the code into several pre-defined modules with dedicated functions. Figure 8 shows the comparison of the programming structure between those two simulators.

Simulink Approach	Scicos Approach
Program code structure	

```
#define S_FUNCTION_NAME
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

static void mdlInitializeSizes(SimStruct *S) {
}
static void mdlInitializeSampleTimes(SimStruct *S) {
}
static void mdlStart(SimStruct *S) {
}
static void mdlInitializeConditions(SimStruct *S) {
}
static void mdlOutputs(SimStruct *S, int_T tid) {
}
static void mdlZeroCrossings(SimStruct *S) {
}
static void mdlTerminate(SimStruct *S) {
}

#ifdef MATLAB_MEX_FILE
/* Is this file being compiled as a MEX-file? */
#include "simulink.c"
/* MEX-file interface mechanism */
#else
#include "cg_sfun.h"
/* Code generation registration function */
#endif
```

```
#include "scicos_block.h"

void function_name(scicos_block *block, int flag) {

switch (flag) {

  case 4:  // ***** Initalization *****

    /* << mdlInitializeSizes >> */
     ..
    /* << mdlInitializeSampleTimes >> */
     ..
    /* << mdlStart >> */
     ..
    /* << mdlInitializeConditions >> */
     ..

  case 1:  // ***** Update outputs *****
    /* << mdlOutputs >> */
     ..

  case 9:  // ***** Calculate zero-crossings *****
    /* << mdlZeroCrossings >> */
     ..

  case 5:  // ***** Termination *****
    /* << mdlTerminate >> */
     ..

}
```

Figure 8 – Comparison between Simulink and Scicos program structure

3.1.4 Block input and output sizes

When interfacing with Simulink, it is possible to define all input and output sizes from the computational function written in C++. This is not the case for Scicos since the sizing is expected to be done from the corresponding interface function, entered manually either from its interaction window or in its interface function code.

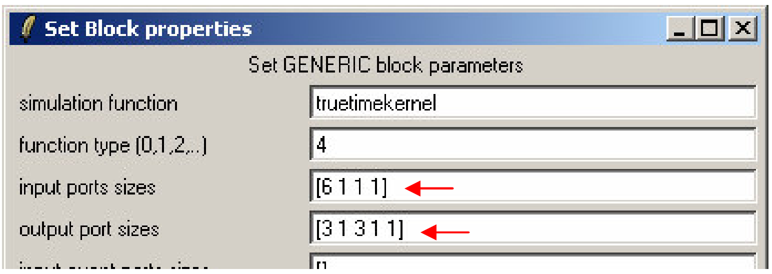
MATLAB Approach	Scilab Approach
To define the block input sizes	

ssSetNumInputPorts(S, 4);
ssSetInputPortWidth(S, 0, rtsys->nbrOfInputs);
ssSetInputPortWidth(S, 1, rtsys->nbrOfTriggers);
ssSetInputPortWidth(S, 2, rtsys->nbrOfNetworks);
ssSetInputPortWidth(S, 3, 1);
The functions are called from computational function written in C++ to set the input sizes
	if (block->nin==4) { … }
	if (block->insz[0] != rtsys->nbrOfInputs) { … }
	if (block->insz[1] != rtsys->nbrOfTriggers) { … }
	if (block->insz[2] != rtsys->nbrOfNetworks) { … }
	if (block->insz[3] != 1) { … }
	The block structure can only be used to read values from the C computational function. To set the input sizes is done from the setting interface of the block as shown in Figure 10.

To define the block output sizes	

ssSetNumOutputPorts(S, 5);
ssSetOutputPortWidth(S, 0, rtsys->nbrOfOutputs);
ssSetOutputPortWidth(S, 1, (rtsys->nbrOfNetworks));
ssSetOutputPortWidth(S, 2, rtsys->nbrOfSchedTasks+rtsys->nbrOfSchedHandlers);
ssSetOutputPortWidth(S, 3, rtsys->nbrOfSchedMonitors*rtsys->nbrOfTasks);
ssSetOutputPortWidth(S, 4, 1);
	if (block->nout==5) { … }
	if (rtsys->nbrOfOutputs > 0) { … }
	if (block->outsz[1] != rtsys->nbrOfNetworks) { … }
	if (block->outsz[2] != rtsys->nbrOfSchedTasks+rtsys->nbrOfSchedHandlers) { … }
	if (block->outsz[3] != rtsys->nbrOfSchedMonitors*rtsys->nbrOfTasks) { … }
	if (block->outsz[4] != 1) { … }
	The block structure can only be used to read values from the C computational function. To set the output sizes is done from the setting interface of the block as shown in Figure 10.

Figure 9 – Comparison in setting the block input and ouput sizes


Set Block properties

Set GENERIC block parameters

simulation function	truetimekernel
function type (0,1,2,..)	4
input ports sizes	[6 1 1 1]
output port sizes	[3 1 3 1 1]

Figure 10 – User setting interface using GENERIC block in Scicos

Under testing, GENERIC block is used to set the block input/output sizes via its interaction window. Other alternative is to create user define interface block corresponding to this Generic block and set the sizes in its interface function code truetimekernel.sci using commands model.in=[6;1;1;1] and model.out=[3;1;3;1;1]. Further details are explained in work performed under chapter 3.2.

3.1.5 Sample times

Both M-file and C MEX S-functions allow a high degree of flexibility in specifying when an S-function executes. Simulink provides the following options for sample times:

- Continuous sample time (For S-functions that have continuous states or non-sampled zero crossings. For this type of S-function, the output changes in minor time steps)
- Continuous but fixed in minor time step sample time (For S-functions that need to execute at every major simulation step, but do not during minor time steps)
- Discrete sample time (For S-Function that is a function of discrete time intervals, so that the block is called by Simulink based on a sample time defined by the user. An offset that delays each sample time hit can also be defined.)
- Variable sample time (A discrete sample time where the intervals can vary)
- Inherited sample time (For S-Function block that is depending on the sample time of some other block in the system).

The functions that are used for this purpose in TrueTime are as follows:

```
ssSetNumSampleTimes(S, 1);
ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);
```

Figure 11 – Simulink functions related to sample time setting in TrueTime kernel block

In order to have the same purpose in Scicos, the user need to design the computational function code by placing the code to be executed (e.g. update of output signals) under the right scicos execution flag combined with either simulation phase (called by using scicos command *get_phase_simulation()*) or external event (event input must be define in this case). Fortunately, TrueTime uses the option CONTINUOUS_SAMPLE_TIME, which requires the least amount of code to program.

3.1.6 Error messages

Simulink provides a built-in function to report errors that occur in its S-function (corresponds to computational function in Scicos) as shown in Figure 12. It sends any message entered into its function parameter to the simulator.

MATLAB Approach	Scilab Approach
To report error message to simulator from the computational function	
void ssSetErrorStatus(SimStruct *S, const char_T *msg)	void set_block_error(int)

Figure 12 – Error message reporting from computational function

The corresponding built-in function in Scicos has limitation to a number of predefined messages (selected by entering the ID of the desired message), and only three messages that are related to errors that can occur in the computational function. Those three are as follows:

- -1 : This mean that the block input does not have the expected value

- -2 : Occurs if the block encounters singularity, such as division by zero
- -16 : Corresponds to the case when the block fails to allocate memory

3.1.7 Getting system time

Both Scicos and Simulink have their own built-in functions to get the system time into the computational function.

MATLAB Approach	Scilab Approach
To get system time from the simulator	
ssGetT(S)	Get_scicos_time()

Figure 13 – Different built-in functions to get simulator's time

The only small difference is the placement of this built-in function. Porting the code directly from Simulink will result that this function will land under flag 1, execution flag to update the output signals. Scicos recommends that any time calculation with zero-crossing enabled shall be executed under flag 9, the execution flag to calculate zero-crossing surface. Experience during porting of TrueTime shows that built-in function *get_scicos_time()* can fetch a wrong time value under flag 1.

3.1.8 Simulation option for computational function

Simulink provides a built-in function to specify the S-function options. A corresponding built-in function in Scicos that match the functionality could not be found. The function is used in TrueTime, and excluding this from the ported code does not introduce any big issue during the project.

```
ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE |
SS_OPTION_CALL_TERMINATE_ON_EXIT);
```

Figure 14 – Simulink function to set simulation options

3.1.9 Abort the simulation

There is one part of the computational function that requires simulation abortion when the kernel has fatal error. Simulink has built-in function for this purpose as shown in Figure 15. Scicos has a similar function but it is in Scicos diagram block type. The way to activate the block is to create a new event activation output signal in TrueTime kernel block to be linked to this STOP block. Upon the arrival of an event to this block, the simulation is stopped.

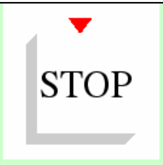
MATLAB Approach	Scilab Approach
To abort the running simulation	
ssSetStopRequested(S, 1);	

Figure 15 – Different method to abort simulation

The project decided not to implement this with the following considerations:

- The TrueTime kernel block will not have the same look and feel as the MATLAB version
- Sending an error message to the Scicos simulator will stop the simulation anyway, and this can be assumed as a workaround.

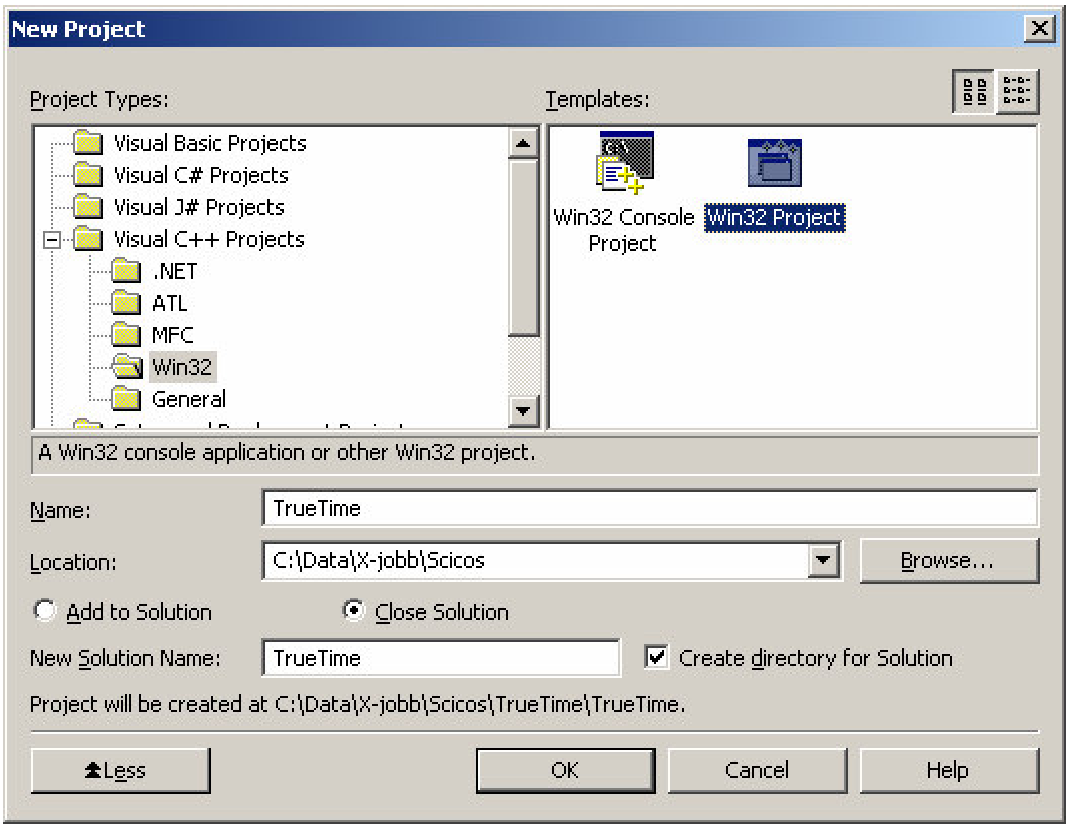
3.2 Work performed

This is the main section detailing how to solve the compilation problem, how the block function was finally written, details about reading inputs and outputs, and which parts of TrueTime are excluded.

3.2.1 Visual C++ configuration

Since Windows XP is chosen as the platform to perform the porting in this project, Microsoft Visual C++ is required by Scilab to compile the user-defined library written in C/C++. Some adjustments must be done so that the compiled library code can be linked to Scilab.

First of all, a new Visual C++ project shall be created in the Microsoft Visual C++ Editor with type *Win32 Project* as shown in Figure 16 below. In the Application Setting window, select *DLL* for application type and *Empty project* as additional options.


New Project

Project Types: Visual Basic Projects, Visual C# Projects, Visual J# Projects, Visual C++ Projects (.NET, ATL, MFC, Win32, General)

Templates: Win32 Console Project, Win32 Project

A Win32 console application or other Win32 project.

Name: TrueTime

Location: C:\Data\X-jobb\Scicos

Add to Solution [✓ Close Solution]

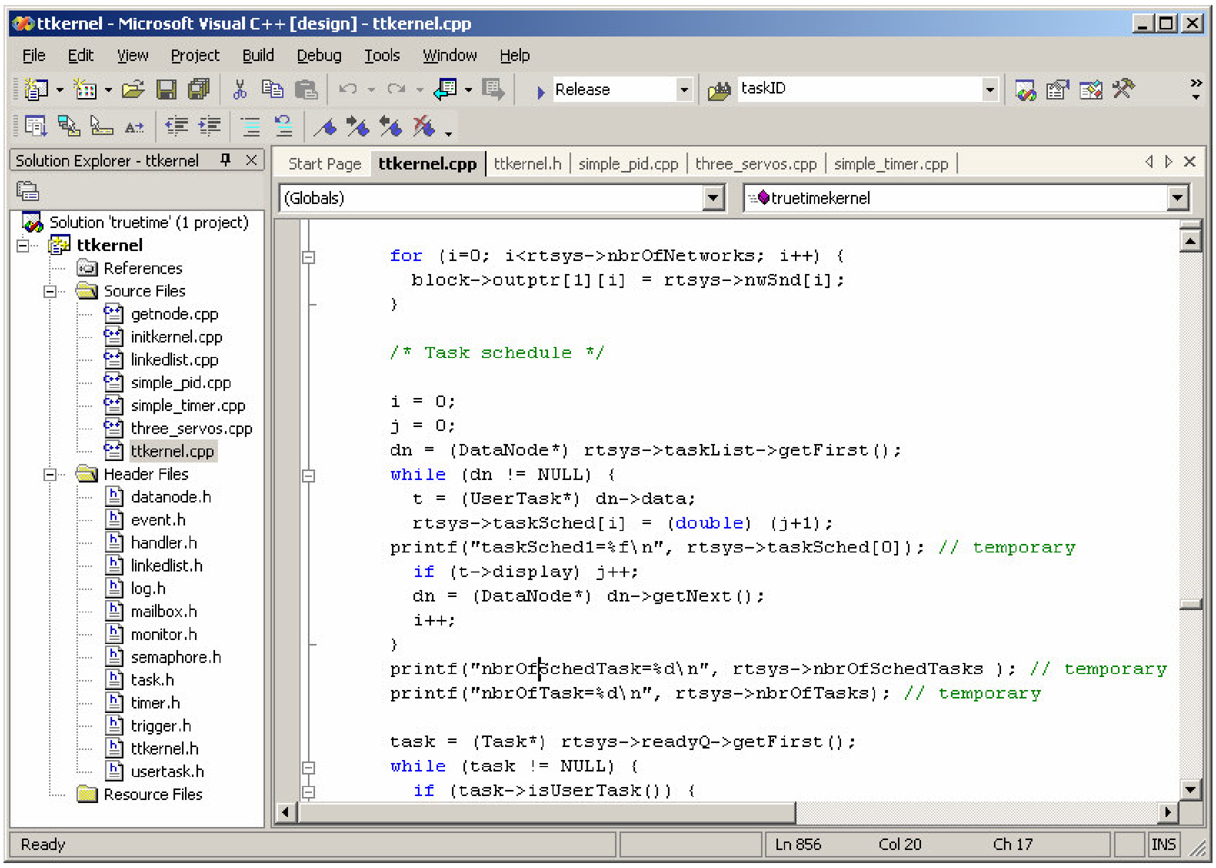
New Solution Name: TrueTime

[✓] Create directory for Solution

Project will be created at C:\Data\X-jobb\Scicos\TrueTime\TrueTime.

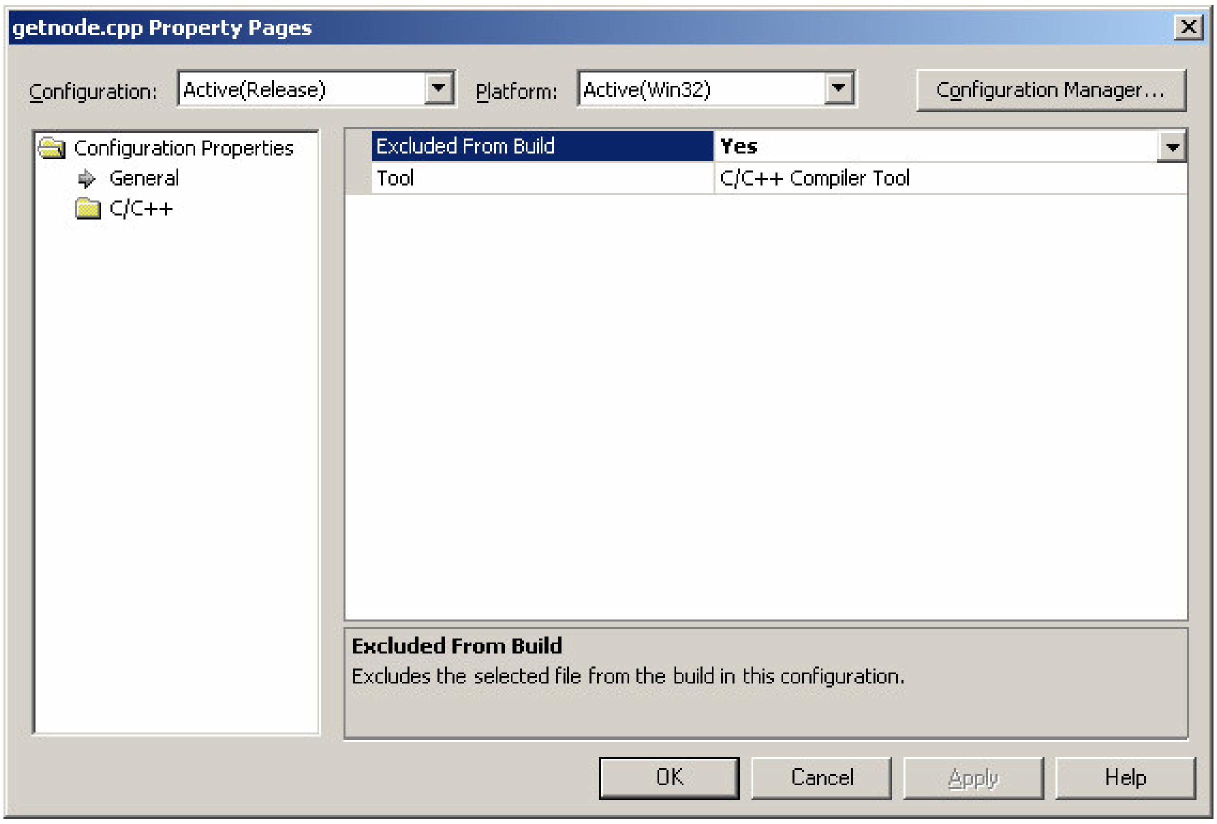
Figure 16 - Creating Visual C++ Project for TrueTime

The *.cpp files of TrueTime that are planned to be edited are added into folder 'Source Files' and *.h files into folder 'Header Files'. Since the main code is in ttkernel.cpp, all files included in those two folders except ttkernel.cpp shall be marked 'Excluded from Build', otherwise compilation error can occur. See example in Figure 18 below to exclude getnode.cpp from the build.



```
    for (i=0; i<rtsys->nbrOfNetworks; i++) {
      block->outptr[1][i] = rtsys->nwSnd[i];
    }

    /* Task schedule */

    i = 0;
    j = 0;
    dn = (DataNode*) rtsys->taskList->getFirst();
    while (dn != NULL) {
      t = (UserTask*) dn->data;
      rtsys->taskSched[i] = (double) (j+1);
    printf("taskSched1=%f\n", rtsys->taskSched[0]); // temporary
      if (t->display) j++;
      dn = (DataNode*) dn->getNext();
      i++;
    }
    printf("nbrOfSchedTask=%d\n", rtsys->nbrOfSchedTasks ); // temporary
    printf("nbrOfTask=%d\n", rtsys->nbrOfTasks); // temporary

    task = (Task*) rtsys->readyQ->getFirst();
    while (task != NULL) {
      if (task->isUserTask()) {
```

Figure 17 - Microsoft Visual C++ Editor with *ttkernel.cpp* code



Configuration: Active(Release) Platform: Active(Win32)

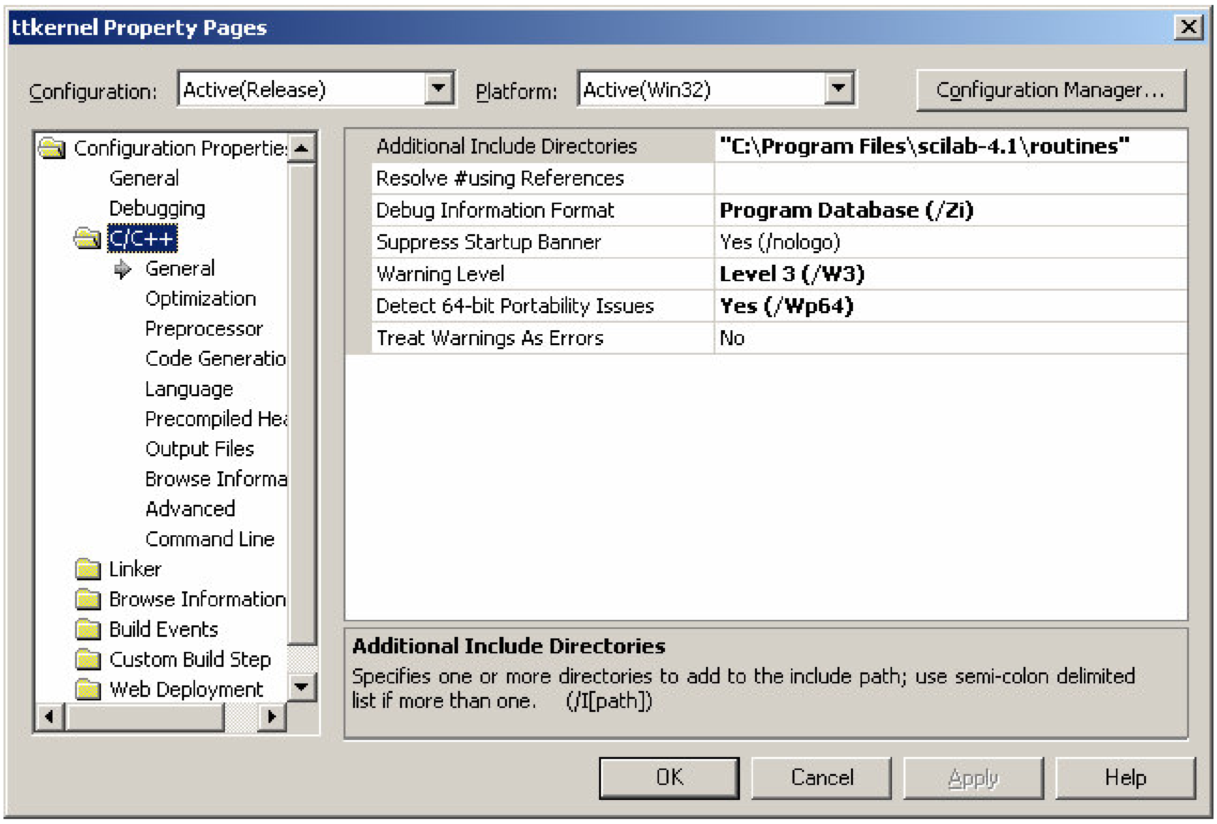
Excluded From Build	**Yes**
Tool	C/C++ Compiler Tool

Excluded From Build
Excludes the selected file from the build in this configuration.

Figure 18 - Example of getnode.cpp excluded from build

The next challenging activity is to adjust some of the configuration properties on ttkernel object. There are quite many properties here but normally most of the properties can be kept as they are. Only the change-required configuration properties are described below.

The first part is C/C++->General as shown in Figure 19. The folder "..\scilab-4.1\routines" shall be stated as Additonal Include Directories since it contains many important files included in the project, one of them is \scicos\scicos_block.h. In C/C++->Preprocessor, some pre-processor definitions are added based on file "Makefile.incl.mak" located under folder "..\scilab-4.1" and file "Makelib.mak" under working directory.



Additional Include Directories	"C:\Program Files\scilab-4.1\routines"
Resolve #using References	
Debug Information Format	Program Database (/Zi)
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level 3 (/W3)
Detect 64-bit Portability Issues	Yes (/Wp64)
Treat Warnings As Errors	No

Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than one. (/I[path])

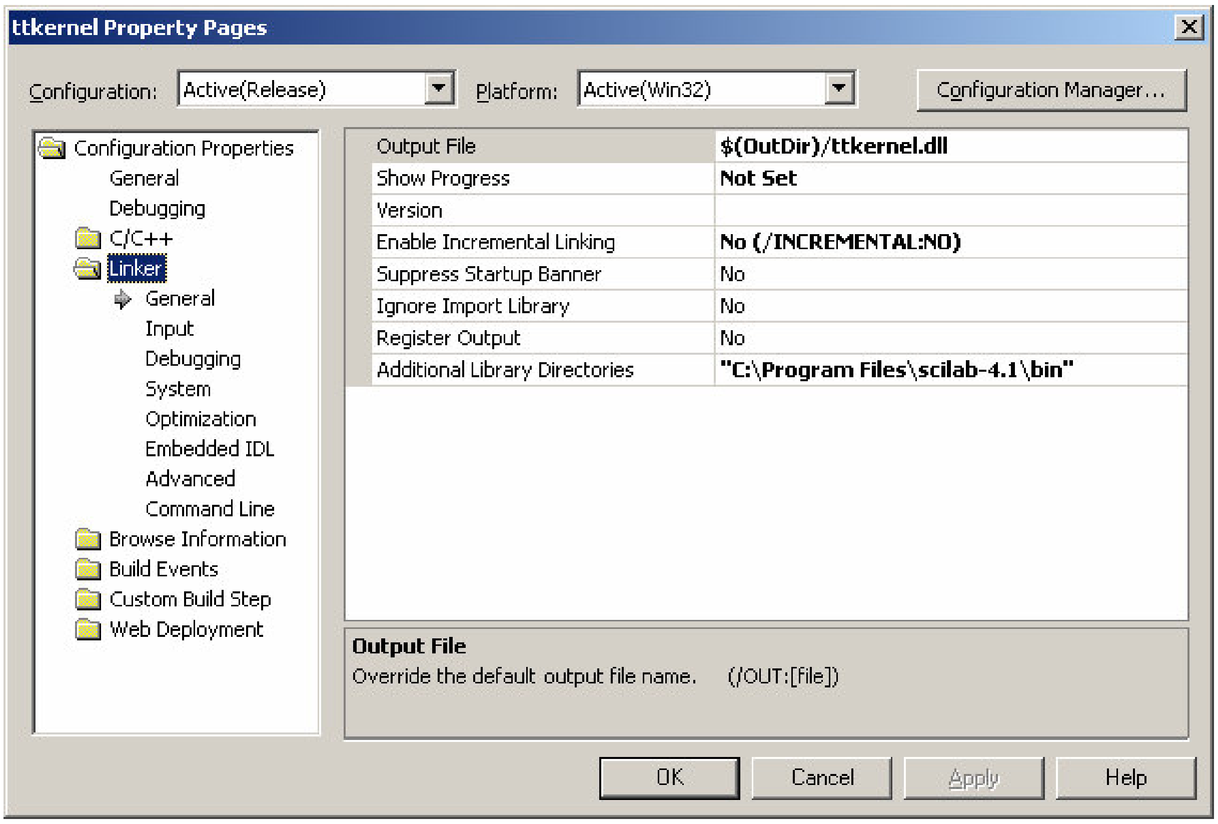
Figure 19 - Configuration Properties -> C/C++ -> General

The complete configuration properties for C/C++ can be seen as text in C/C++->Command Line.

```
/O2 /I "C:\Program Files\scilab-4.1\routines" /D "__MSC__" /D "WIN32" /D
"NDEBUG" /D "_WINDOWS" /D "_USRDLL" /D "STRICT" /D "__MAKEFILEVC__"
/D "mexFunction_=mex$*_" /D "mexFunction=mex_$*" /D "_WINDLL" /D "_MBCS"
/FD /EHsc /MT /GS /Fo"Release/" /Fd"Release/vc70.pdb" /W3 /nologo /c /Wp64 /Zi /TP
```

Figure 20 - Command line for the C/C++ configuration properties

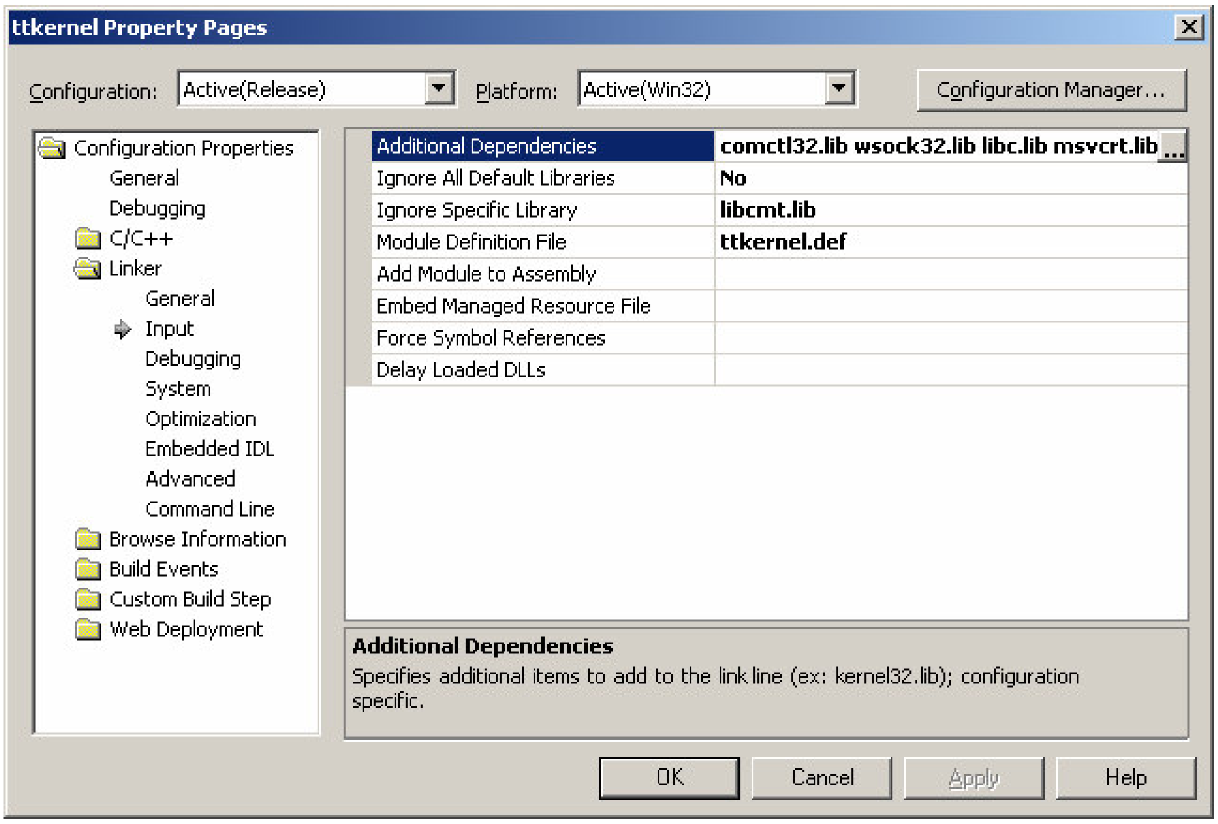
The next section of configuration properties is Linker. Under General properties, folder "..\scilab-4.1\bin" shall be added as Additional Library Directories, because several important libraries from Scilab need to be linked when generating ttkernel.dll. One of them is *LibScilab.lib* according to file "Makefile.mak" under folder "..\scilab-4.1\config".



Output File	$(OutDir)/ttkernel.dll
Show Progress	Not Set
Version	
Enable Incremental Linking	No (/INCREMENTAL:NO)
Suppress Startup Banner	No
Ignore Import Library	No
Register Output	No
Additional Library Directories	"C:\Program Files\scilab-4.1\bin"

Output File
Override the default output file name. (/OUT:[file])

Figure 21 - Configuration Properties -> Linker -> General



Additional Dependencies	comctl32.lib wsock32.lib libc.lib msvcrt.lib
Ignore All Default Libraries	No
Ignore Specific Library	libcmt.lib
Module Definition File	ttkernel.def
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded DLLs	

Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.

Figure 22 - Configuration Properties -> Linker -> Input

Under properties Input, sub-properties Additional Dependencies and Ignore Specific Library shall be updated according to file "Makefile.incl.mak" under folder "..\scilab-4.1". See Figure

22 for details. In this project, sub properties Module Definition File is set to ttkernel.def, which will be described further in chapter 3.2.2.

Under properties Advanced, the file extension of the import library can be changed to *ttkernel.ilib* to synchronize with Scilab library. To keep it as it is will not be a problem.

Likewise the properties for C/C++, the complete command line for Linker can be displayed also, and in this project it shows as below. The rest of the properties can be kept as they are.

```
/OUT:"Release/ttkernel.dll" /INCREMENTAL:NO /NOLOGO /LIBPATH:"C:\Program
Files\scilab-4.1\bin" /DLL /NODEFAULTLIB:"libcmt.lib" /DEF:"ttkernel.def" /DEBUG
/PDB:"Release/ttkernel.pdb" /SUBSYSTEM:WINDOWS /OPT:REF /OPT:ICF
/IMPLIB:"Release/ttkernel.ilib" /MACHINE:X86 comctl32.lib wsock32.lib libc.lib
msvcrt.lib LibScilab.lib  kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib "\Program
Files\scilab-4.1\bin\LibScilab.lib"
```

Figure 23 - Command line for the Linker configuration properties

3.2.2 Adjustments to succeed compilation and linking

Since Scilab has different method than MATLAB to collaborate with a user-defined library written in C++ as described in chapter 3.1.3, some adjustments are required to succeed the linking between them. The major works performed for this purpose are described below.

Removing all MATLAB specific code that Scilab can not replace

The challenge in this work is to have as few excluded functions as possible by finding as many Scicos replacement functions as possible. Figure 24 lists all the .cpp files used to develop TrueTime kernel block. Each file is marked either with black text, red text or blue text.

The black marked files are those files that were not changed due to one of the reasons below:

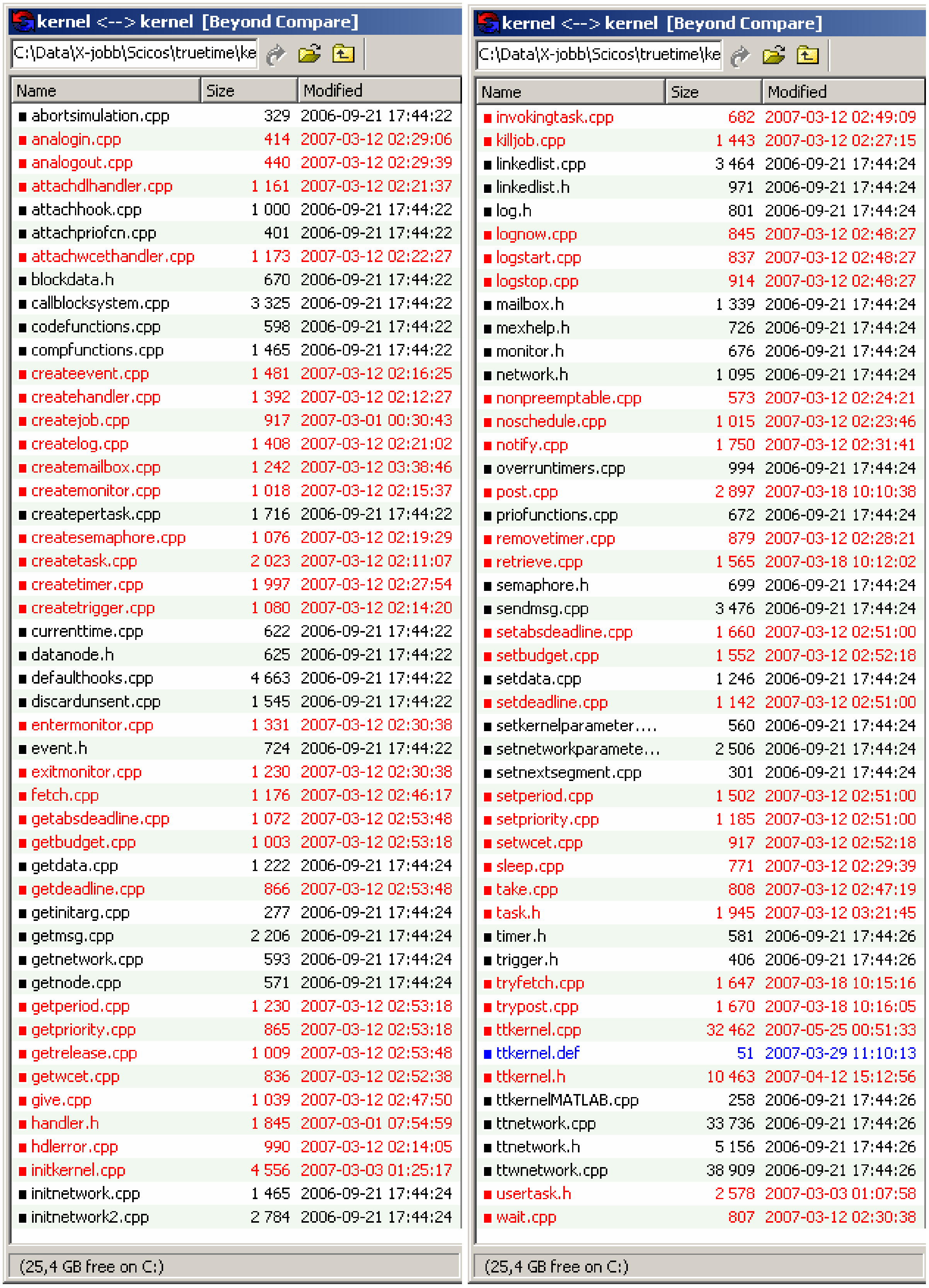
- No change required since they can be used as they are
- Excluded due to network related functions
- Excluded due to MATLAB specific functions

The list of functions or files that are excluded can be found in chapter 3.2.6.

The red marked files are those files that were changed due to adjustment to Scicos environment. There is only one blue marked file (*ttkernel.def*) that was newly created to define Scicos exported functions.

The most frequent error that occurs in most of the files is *MEX_ERROR();*. The function *printf()* is used to replace this since there is no specific Scicos function that corresponds to this function. This work did not take much effort but it was done for all changed files.

The file that has the most changes is *ttkernel.cpp* since all kernel code is located in this file. The detail changes will be described in chapter 3.2.3. The next file that has important changes is *ttkernel.h*. There are several signs that can be used to ease the identification of MATLAB specific code, such as the following: ‘MATLAB’, ‘mx’, ‘MEX’, ‘ssSetxxx’.



kernel <--> kernel [Beyond Compare]

C:\Data\X-jobb\Scicos\truetime\ke

Name	Size	Modified
abortsimulation.cpp	329	2006-09-21 17:44:22
analogin.cpp	414	2007-03-12 02:29:06
analogout.cpp	440	2007-03-12 02:29:39
attachdlhandler.cpp	1 161	2007-03-12 02:21:37
attachhook.cpp	1 000	2006-09-21 17:44:22
attachpriofcn.cpp	401	2006-09-21 17:44:22
attachwcethandler.cpp	1 173	2007-03-12 02:22:27
blockdata.h	670	2006-09-21 17:44:22
callblocksystem.cpp	3 325	2006-09-21 17:44:22
codefunctions.cpp	598	2006-09-21 17:44:22
compfunctions.cpp	1 465	2006-09-21 17:44:22
createevent.cpp	1 481	2007-03-12 02:16:25
createhandler.cpp	1 392	2007-03-12 02:12:27
createjob.cpp	917	2007-03-01 00:30:43
createlog.cpp	1 408	2007-03-12 02:21:02
createmailbox.cpp	1 242	2007-03-12 03:38:46
createmonitor.cpp	1 018	2007-03-12 02:15:37
createpertask.cpp	1 716	2006-09-21 17:44:22
createsemaphore.cpp	1 076	2007-03-12 02:19:29
createtask.cpp	2 023	2007-03-12 02:11:07
createtimer.cpp	1 997	2007-03-12 02:27:54
createtrigger.cpp	1 080	2007-03-12 02:14:20
currenttime.cpp	622	2006-09-21 17:44:22
datanode.h	625	2006-09-21 17:44:22
defaulthooks.cpp	4 663	2006-09-21 17:44:22
discardunsent.cpp	1 545	2006-09-21 17:44:22
entermonitor.cpp	1 331	2007-03-12 02:30:38
event.h	724	2006-09-21 17:44:22
exitmonitor.cpp	1 230	2007-03-12 02:30:38
fetch.cpp	1 176	2007-03-12 02:46:17
getabsdeadline.cpp	1 072	2007-03-12 02:53:48
getbudget.cpp	1 003	2007-03-12 02:53:18
getdata.cpp	1 222	2006-09-21 17:44:24
getdeadline.cpp	866	2007-03-12 02:53:48
getinitarg.cpp	277	2006-09-21 17:44:24
getmsg.cpp	2 206	2006-09-21 17:44:24
getnetwork.cpp	593	2006-09-21 17:44:24
getnode.cpp	571	2006-09-21 17:44:24
getperiod.cpp	1 230	2007-03-12 02:53:18
getpriority.cpp	865	2007-03-12 02:53:18
getrelease.cpp	1 009	2007-03-12 02:53:48
getwcet.cpp	836	2007-03-12 02:52:38
give.cpp	1 039	2007-03-12 02:47:50
handler.h	1 845	2007-03-01 07:54:59
hdlerror.cpp	990	2007-03-12 02:14:05
initkernel.cpp	4 556	2007-03-03 01:25:17
initnetwork.cpp	1 465	2006-09-21 17:44:24
initnetwork2.cpp	2 784	2006-09-21 17:44:24

(25,4 GB free on C:)

kernel <--> kernel [Beyond Compare]

C:\Data\X-jobb\Scicos\truetime\ke

Name	Size	Modified
invokingtask.cpp	682	2007-03-12 02:49:09
killjob.cpp	1 443	2007-03-12 02:27:15
linkedlist.cpp	3 464	2006-09-21 17:44:24
linkedlist.h	971	2006-09-21 17:44:24
log.h	801	2006-09-21 17:44:24
lognow.cpp	845	2007-03-12 02:48:27
logstart.cpp	837	2007-03-12 02:48:27
logstop.cpp	914	2007-03-12 02:48:27
mailbox.h	1 339	2006-09-21 17:44:24
mexhelp.h	726	2006-09-21 17:44:24
monitor.h	676	2006-09-21 17:44:24
network.h	1 095	2006-09-21 17:44:24
nonpreemptable.cpp	573	2007-03-12 02:24:21
noschedule.cpp	1 015	2007-03-12 02:23:46
notify.cpp	1 750	2007-03-12 02:31:41
overruntimers.cpp	994	2006-09-21 17:44:24
post.cpp	2 897	2007-03-18 10:10:38
priofunctions.cpp	672	2006-09-21 17:44:24
removetimer.cpp	879	2007-03-12 02:28:21
retrieve.cpp	1 565	2007-03-18 10:12:02
semaphore.h	699	2006-09-21 17:44:24
sendmsg.cpp	3 476	2006-09-21 17:44:24
setabsdeadline.cpp	1 660	2007-03-12 02:51:00
setbudget.cpp	1 552	2007-03-12 02:52:18
setdata.cpp	1 246	2006-09-21 17:44:24
setdeadline.cpp	1 142	2007-03-12 02:51:00
setkernelparameter....	560	2006-09-21 17:44:24
setnetworkparamete...	2 506	2006-09-21 17:44:24
setnextsegment.cpp	301	2006-09-21 17:44:24
setperiod.cpp	1 502	2007-03-12 02:51:00
setpriority.cpp	1 185	2007-03-12 02:51:00
setwcet.cpp	917	2007-03-12 02:52:18
sleep.cpp	771	2007-03-12 02:29:39
take.cpp	808	2007-03-12 02:47:19
task.h	1 945	2007-03-12 03:21:45
timer.h	581	2006-09-21 17:44:26
trigger.h	406	2006-09-21 17:44:26
tryfetch.cpp	1 647	2007-03-18 10:15:16
trypost.cpp	1 670	2007-03-18 10:16:05
ttkernel.cpp	32 462	2007-05-25 00:51:33
ttkernel.def	51	2007-03-29 11:10:13
ttkernel.h	10 463	2007-04-12 15:12:56
ttkernelMATLAB.cpp	258	2006-09-21 17:44:26
ttnetwork.cpp	33 736	2006-09-21 17:44:26
ttnetwork.h	5 156	2006-09-21 17:44:26
ttwnetwork.cpp	38 909	2006-09-21 17:44:26
usertask.h	2 578	2007-03-03 01:07:58
wait.cpp	807	2007-03-12 02:30:38

(25,4 GB free on C:)

Figure 24 – List of all .cpp and .h files for TrueTime kernel block (red: modified; blue: new; black: unmodified)

Statement of extern “C”

Since Scicos is written in C, the data structure and functions defined in file “scicos_block.h” need to stated as extern “C”. Without this statement, calling to any of the predefined function, i.e. get_scicos_time(), will introduce compilation error in Visual C++ compilation.

```
extern "C" {
#include "scicos/scicos_block.h"
}
```

Figure 25 – Declare scicos_block.h in extern “C” statement

Creation of ttkernel.def

The purpose of this file is to be attached to the Visual C++ linker so that it exports the “truetimekernel” function, so that the function is accessible from Scilab/Scicos.

```
LIBRARY    ttkernel.dll
EXPORTS
                 truetimekernel
```

Figure 26 – Content of ttkernel.def

Modification in loader.sce

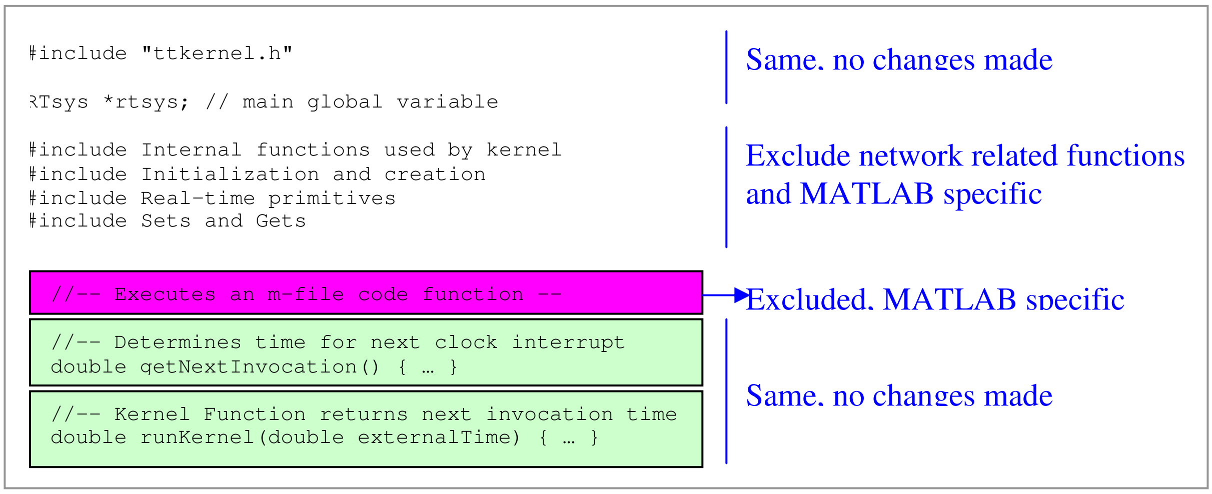
The script in the file loader.sce is intended to link the ttkernel.dll to Scicos so that the exported function “truetimekernel” is visible from Scicos.

```
myblock_path=get_absolute_file_path('loader.sce');
link(myblock_path+'ttkernel.dll',['truetimekernel'],'c');
scicos SimpleTimer.cos;
```

Figur 27 – Content of loader.sce

3.2.3 Code restructuring

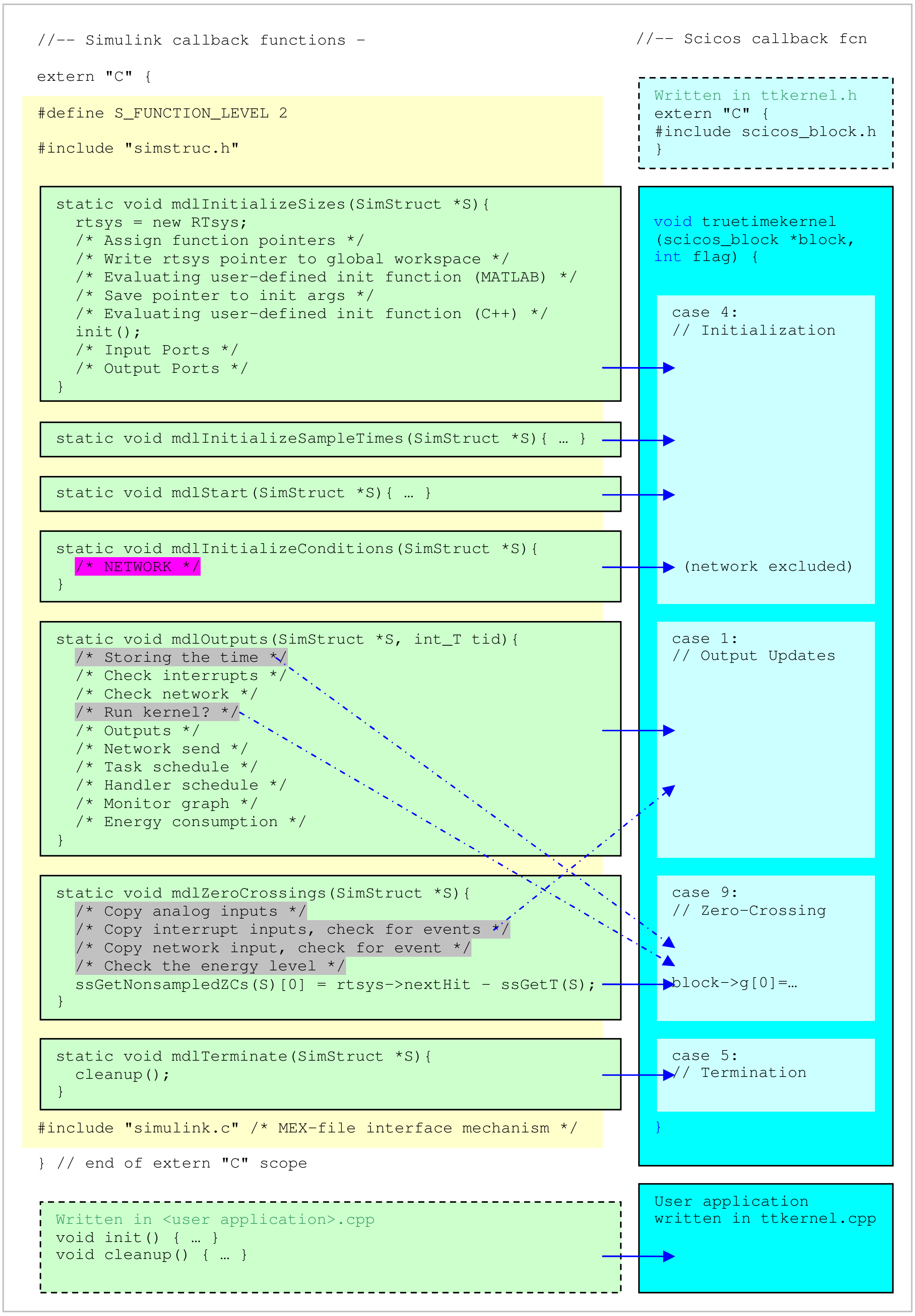
Following up the discussion on the differences on computational function structure in chapter 3.1.3, some necessary changes were made in file ttkernel.cpp.



```
#include "ttkernel.h"

RTsys *rtsys; // main global variable

#include Internal functions used by kernel
#include Initialization and creation
#include Real-time primitives
#include Sets and Gets

//-- Executes an m-file code function --

//-- Determines time for next clock interrupt
double getNextInvocation() { … }

//-- Kernel Function returns next invocation time
double runKernel(double externalTime) { … }
```

Same, no changes made
Exclude network related functions and MATLAB specific
Excluded, MATLAB specific
Same, no changes made

Figure 28 – Program code restructuring (part 1)



```
//-- Simulink callback functions -

extern "C" {

#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

static void mdlInitializeSizes(SimStruct *S){
  rtsys = new RTsys;
  /* Assign function pointers */
  /* Write rtsys pointer to global workspace */
  /* Evaluating user-defined init function (MATLAB) */
  /* Save pointer to init args */
  /* Evaluating user-defined init function (C++) */
  init();
  /* Input Ports */
  /* Output Ports */
}

static void mdlInitializeSampleTimes(SimStruct *S){ … }

static void mdlStart(SimStruct *S){ … }

static void mdlInitializeConditions(SimStruct *S){
  /* NETWORK */
}

static void mdlOutputs(SimStruct *S, int_T tid){
  /* Storing the time */
  /* Check interrupts */
  /* Check network */
  /* Run kernel? */
  /* Outputs */
  /* Network send */
  /* Task schedule */
  /* Handler schedule */
  /* Monitor graph */
  /* Energy consumption */
}

static void mdlZeroCrossings(SimStruct *S){
  /* Copy analog inputs */
  /* Copy interrupt inputs, check for events */
  /* Copy network input, check for event */
  /* Check the energy level */
  ssGetNonsampledZCs(S)[0] = rtsys->nextHit - ssGetT(S);
}

static void mdlTerminate(SimStruct *S){
  cleanup();
}

#include "simulink.c" /* MEX-file interface mechanism */

} // end of extern "C" scope
```

```
Written in <user application>.cpp
void init() { … }
void cleanup() { … }
```

```
//-- Scicos callback fcn
```

```
Written in ttkernel.h
extern "C" {
#include scicos_block.h
}
```

```
void truetimekernel
(scicos_block *block,
int flag) {

  case 4:
  // Initialization

  (network excluded)

  case 1:
  // Output Updates

  case 9:
  // Zero-Crossing

  block->g[0]=…

  case 5:
  // Termination

}
```

```
User application
written in ttkernel.cpp
```

Figure 29 – Program code restructuring (part 2)

As described in Figure 28 above, this part is not affected by restructuring. The work performed here is excluding code that is related to network implementation and MATLAB specific function.

The second part in Figure 29 shows that it is sufficient to put statement #include “scicos_block.h” into extern “C” scope, instead of the whole function calls as done for MATLAB. This section of code is moved to the header file ttkernel.h, due to programming consistency reason.

Initialization job (Scicos flag 4)

The section to initialize the simulation is represented by four modules in MATLAB, but Scicos does not have such detail. The code in those four modules was merged into one and executed under Scicos initialization job in case flag 4. Some adjustments are needed, for example, to collect all local variables in each original module into one place and to adapt variables with the same name to avoid access collision.

Under the testing, it showed that variable *nbrOfTask* =1 after declaration of *rtsys=new RTsys*. This was causing problem for the scheduler. Code to set this variable to zero was added.

Under module *mdlInitializeConditions*, there is code to initialize the network local variables in the kernel. This code is excluded.

Updating output job (Scicos flag 1)

There is only one module from MATLAB here, which corresponds to updating output job in case flag 1 in Scicos simulation. The code to store the simulation time to global variable *rtsys* and the code to call the function *runkernel()* are moved to zero-crossing job in case 9. The reason is that Scicos recommends that time-related code shall be allocated under this job. When trying to port this code directly under flag 1, the simulation gave strange timing problem, which was caused by a jump in time after certain period.

Another deviation made in this section is removing the intermediate pointers *y*, *n*, *s*, *m* and connecting the output port directly to the application code, based on the consideration that only pointer s has one-to-two-point connection. The rest of the pointers have one-to-one-point connections.

Zero-crossing calculation job (Scicos flag 9)

There is only one module here from MATLAB, which corresponds to zero-crossing calculation job in Scicos in case flag 9. The most important thing in this section is to accommodate the zero-crossing calculations and in Scicos the code look like:

```
block->g[0] = rtsys->nextHit - get_scicos_time();
```

As mentioned in the previous paragraph, the simulation time storing and calling to the function *runkernel()* are moved to this section. Important issue here is that the zero-crossing calculation must precede the calling to the function *runkernel()*, otherwise the next invocation time return by the function will give wrong simulation time. This is because the zero-crossing calculation is supposed to make simulation time adjustment before it is used to calculate the next invocation time.

One deviation was made to move the code to copy input values to updating output job under case flag 1. The decision was taken based on the Scicos examples.

Termination job (Scicos flag 5)

Nothing is strange here since Scicos has the corresponding job to terminate the simulation.

User application

User application code such as PID controller and Three Servos controllers were put at the end of the code in file ttkernel.cpp to simplify the testing. Future work to put the user application to one separate .cpp file can be planned.

3.2.4 Input reading and output settings

The computational function receives most of its inputs from the block structure written in C language and defined as C struct with components as follows:

```
typedef struct {
  int nevprt;
  voidg funpt ;
  int type;
  int scsptr;
  int nz;
  double *z;
  int nx;
  double *x;
  double *xd;
  double *res;
  int nin;
  int *insz;
  double **inptr;
  int nout;
  int *outsz;
  double **outptr;
  int nevout;
  double *evout;
  int nrpar;
  double *rpar;
  int nipar;
  int *ipar;
  int ng;
  double *g;
  int ztyp;
  int *jroot;
  char *label;
  void **work;
  int nmode;
  int *mode;
} scicos_block;
```

Figure 30 – Components of Scicos data block structure

Since it is not possible to set the sizes of Scicos block inputs and outputs from C++ code, different approach has to be done when porting the I/O handling part of the existing TrueTime code. Changes are applied to module *mdlInitializeSizes* in file “ttkernel.cpp” and described in Table 3 below. Scicos requires interface function to set those sizing parameters. In this project, the provided generic interface function as shown in Figure 31 is used. The block is available in palette “Others” in Scicos editor. To have a better user interaction, a user-defined interface function shall be build. The Generic block parameters in Figure 32 must be adjusted to match the compiled TrueTime function. The name of the simulation function is *truetimekernel*, and the function type is 4 since it is written in C++.

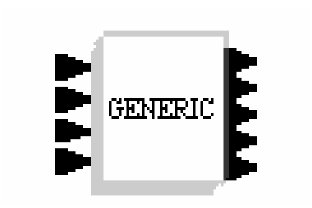
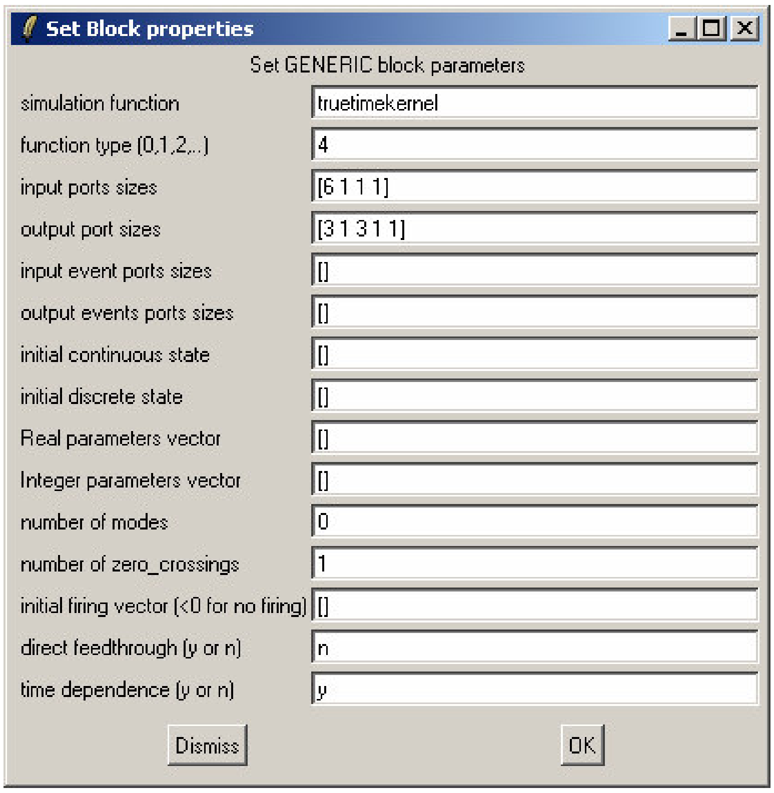


Figure 31 – Generic Interface Function Block



Set GENERIC block parameters

simulation function	truetimekernel
function type (0,1,2,..)	4
input ports sizes	[6 1 1 1]
output port sizes	[3 1 3 1 1]
input event ports sizes	[]
output events ports sizes	[]
initial continuous state	[]
initial discrete state	[]
Real parameters vector	[]
Integer parameters vector	[]
number of modes	0
number of zero_crossings	1
initial firing vector (<0 for no firing)	[]
direct feedthrough (y or n)	n
time dependence (y or n)	y

Figure 32 – Properties of the Generic Interface Function Block

The table below describes the Scilab approach done in this project corresponding to the MATLAB approach made in the existing TrueTime Kernel code (ttkernel.cpp).

No.	MATLAB Approach	Scilab Approach
1	Setting number of input ports	
	if (!ssSetNumInputPorts(S, 4)) return;	Use GENERIC block. Set automatically when parameter “input ports sizes” is defined e.g. [6 1 1 1] -> size = 4. Additional code to do value check: if (block->nin!=4) return;
2	Setting number of output ports	
	if (!ssSetNumOutputPorts(S, 5)) return;	Use GENERIC block. Set automatically when parameter “output ports sizes” is defined e.g. [3 1 3 1 1] -> size = 5. Additional code to do value check: if (block->nout!=5) return;

No.	MATLAB Approach	Scilab Approach
3	Setting input port sizes	

	ssSetInputPortWidth(S, 0, rtsys->nbrOfInputs);
	ssSetInputPortWidth(S, 1, rtsys->nbrOfTriggers);
	ssSetInputPortWidth(S, 2, rtsys->nbrOfNetworks);
	ssSetInputPortWidth(S, 3, 1);
		Use GENERIC block.
		Set parameter "input ports sizes" to e.g. [6 1 1 1].
		Additional code to do value check on each input port size are provided.

4	Setting output port sizes	

	ssSetOutputPortWidth(S, 0, rtsys->nbrOfOutputs);
	ssSetOutputPortWidth(S, 1, rtsys-> nbrOfNetworks);
	ssSetOutputPortWidth(S, 2, rtsys-> nbrOfSchedTasks+rtsys->nbrOfSchedHandlers);
	ssSetOutputPortWidth(S, 3, rtsys-> nbrOfSchedMonitors*rtsys->nbrOfTasks);
	ssSetOutputPortWidth(S, 4, 1); //Energy consumption
		Use GENERIC block.
		Set parameter "output ports sizes" to e.g. [3 1 3 1 1].
		Additional code to do value check on each input port size are provided.

5	Setting number of continuous states	
	ssSetNumContStates(S, 0);	Use GENERIC block. Set parameter "initial continuous state" to [ ]. Additional code to do value check on this parameter are provided.
6	Setting number of discrete states	
	ssSetNumDiscStates(S, 0);	Use GENERIC block. Set parameter "initial discrete state" to [ ]. Additional code to do value check on this parameter are provided.
7	Setting offset time	
	ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);	Excluded since default offset time = 0 is used.
8	Setting number of sample times	
	ssSetNumSampleTimes(S, 1);	Set in simulation setup and use default value.
9	Setting number of modes	
	ssSetNumModes(S, 0);	Use GENERIC block. Set parameter "number of modes" to 0. Additional code is provided to do value check to make sure that no mode is configured for this module.
10	Setting number of non sampled zero-crossings	
	ssSetNumNonsampledZCs(S, 1);	Use GENERIC block. Set parameter "number of zero-crossings" to 1. Additional code is provided to do value check to make sure that only one zero-crossing is set.

No.	MATLAB Approach	Scilab Approach
11	Setting input port direct feed through	

	ssSetInputPortDirectFeedThrough(S, 0, 0);
	ssSetInputPortDirectFeedThrough(S, 1, 0);
	ssSetInputPortDirectFeedThrough(S, 2, 0);
	ssSetInputPortDirectFeedThrough(S, 3, 0);
		Use GENERIC block.
		Set parameter "direct feedthrough (y or n)" to n.

12	Setting number of workplaces	

	ssSetNumRWork(S, 0);
	ssSetNumIWork(S, 0);
	ssSetNumPWork(S, 0);
		Use void **work from the scicos_block.h that accepts different data types and no size definition required.

13	Setting user data to workplace	
	ssSetUserData(S, rtsys);	*block->work = rtsys;
14	Setting number of expected parameters	
	ssSetNumSFcnParams(S, 4);	Not used since corresponding code for reading of m-file is not ported. Possible future work.
15	Assigning values to output ports	

	real_T *y = ssGetOutputPortRealSignal(S,0);
	y[i] = rtsys->outputs[i];
	real_T *n = ssGetOutputPortRealSignal(S,1);
	n[i] = rtsys->nwSnd[i];
	real_T *s = ssGetOutputPortRealSignal(S,2);
	s[j] = rtsys->taskSched[i];
	real_T *m = ssGetOutputPortRealSignal(S,3);
	m[j+k*rtsys->nbrOfTasks] = rtsys-> monitorGraph[j];
	real_T *energyConsumption = ssGetOutputPortRealSignal(S,4);
	energyConsumption[0] = rtsys-> energyConsumption;
		Intermediate variables y, n, s, m and energyConsumption are removed and make direct connections instead.
		block->outptr[0][i] = rtsys->outputs[i];
		block->outptr[1][i] = rtsys->nwSnd[i];
		block->outptr[2][j] = rtsys->taskSched[i];
		block->outptr[3][j+k*rtsys->nbrOfTasks] = rtsys-> monitorGraph[j];
		block->outptr[4][0] = rtsys->energyConsumption;

16	Reading values from input ports	

	rtsys->inputs[i] = *ssGetInputPortRealSignalPtrs(S,0)[i];
	rtsys->interruptinputs[i] = *ssGetInputPortRealSignalPtrs(S,1)[i];
	rtsys->networkinputs[i] = *ssGetInputPortRealSignalPtrs(S,2)[i];
	rtsys->energyLevel = *ssGetInputPortRealSignalPtrs(S,3)[0];
		rtsys->inputs[i] = block->inptr[0][i];
		rtsys->interruptinputs[i] = block->inptr[1][i];
		rtsys->networkinputs[i] = block->inptr[2][i];
		rtsys->energyLevel = block->inptr[3][0];

17	Calculating zero-crossing surface	
	ssGetNonsampledZCs(S)[0] = rtsys->nextHit - ssGetT(S);	block->g[0] = rtsys->nextHit - get_scicos_time();

Table 3 - Porting Input and Output Handling

3.2.5 Scicos user interface for TrueTime kernel block

The implementation of Scicos user interface for the TrueTime kernel block is completed even if it is not part of the scope of work. In Simulink, the TrueTime kernel block is delivered together with three other blocks in a package of TrueTime library as shown in Figure 1. To achieve the same feature, a TrueTime library is created in the Scicos environment, which is accessible from the list of palettes (see Figure 33).

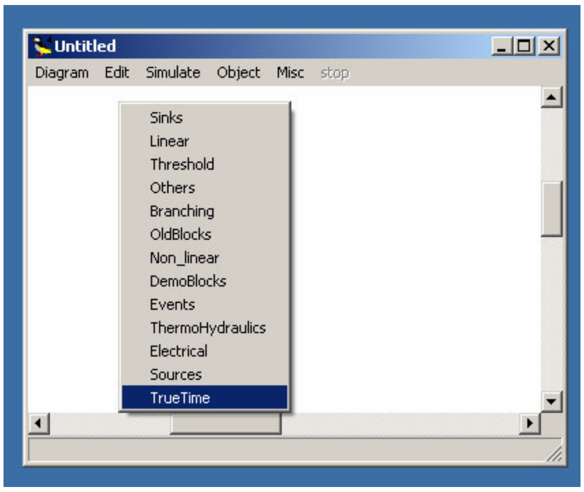


Figure 33 – Scicos list of palettes including the TrueTime library

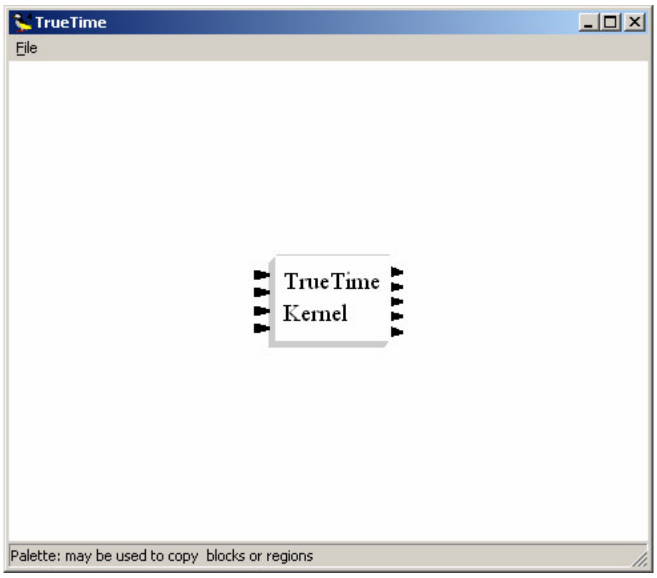


Figure 34 – The TrueTime library consisting the kernel block

As seen on Figure 34 the Scicos version of TrueTime kernel block can not indicate the names of the input/output channels like Simulink. The fourth input channel for battery and the fifth output channel for energy consumption are visible in Scicos TrueTime block, which is not in the Simulink case. Under the simulation testing, the last three input channels are connected to constant value 0, 0 and 1 respectively, corresponding respectively to grounded-connections for Simulink as shown in Figure 43 and battery selected option in Figure 35.

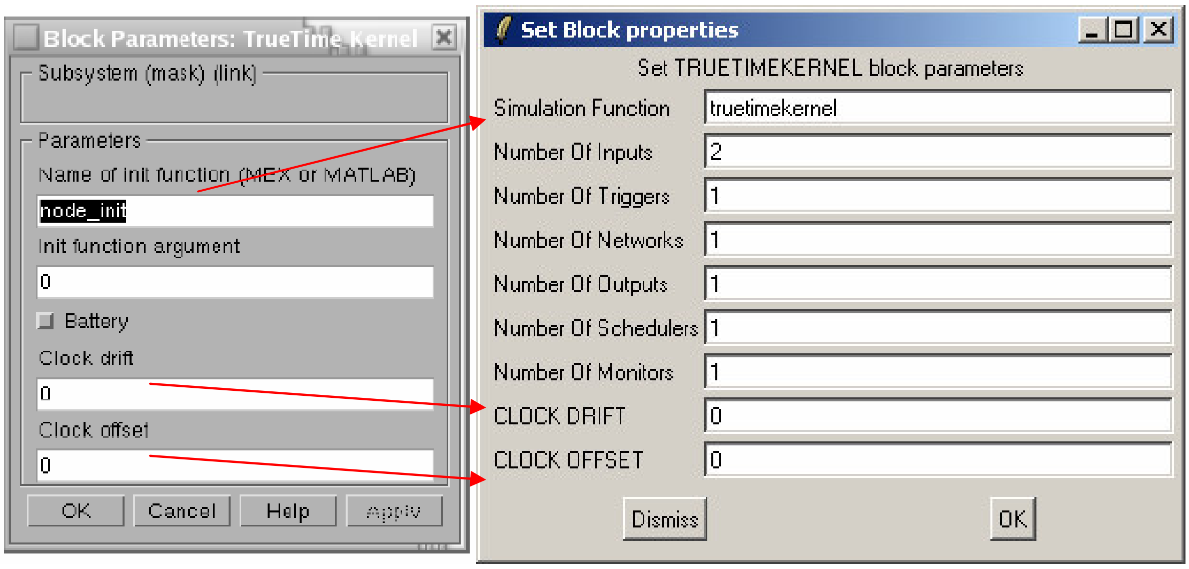
To implement the user setting interface window, the corresponding design in Simulink is used as reference. There are some deviations made during the implementation but the major function is covered. The deviation can be observed using Figure 35.

The first parameter is *simulation function*. The definition is the same for Simulink and Scicos, but a small deviation exists because the user application in Scicos is included in the file ttkernel.cpp. It means for the current version of implementation in Scicos, the parameter *simulation function* should always be *truetimekernel*, instead of user application function as in Simulink. Separation work to have the same interaction can be done in future work.

The parameter *init function argument* is not implemented in Scicos interface since the corresponding feature does not exist.

The parameter Battery in the Simulink interface is implemented as common input channel in Scicos. So, it does not exist as a parameter in the Scicos interface.

The last two parameters in Simulink, *clock drift* and *clock offset*, are implemented without any deviation.



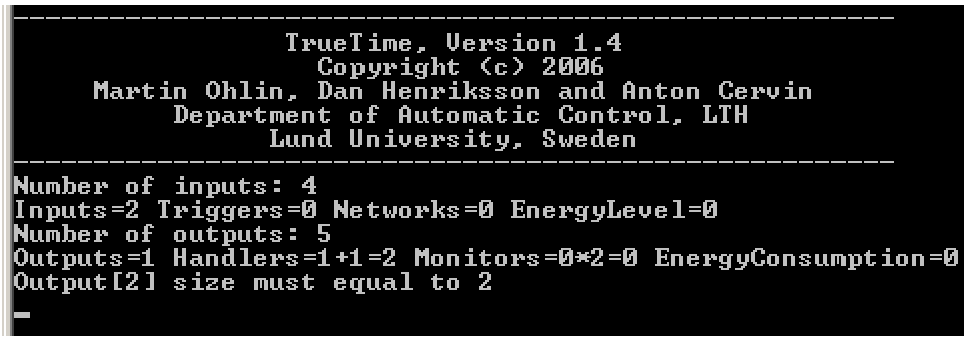
Name of init function (MEX or MATLAB): node_init
Init function argument: 0
Battery
Clock drift: 0
Clock offset: 0

Simulation Function	truetimekernel
Number Of Inputs	2
Number Of Triggers	1
Number Of Networks	1
Number Of Outputs	1
Number Of Schedulers	1
Number Of Monitors	1
CLOCK DRIFT	0
CLOCK OFFSET	0

Figure 35 – User setting interfaces for TrueTime kernel in Simulink (left) and in Scicos (right)

There are six parameters in Scicos interface (Number of Inputs, Number of Triggers, etc.), which are not available in Simulink interface. They are used to configure the block input and output sizes as discussed in chapter 3.1.4. All of them have 1 as default value and 0 is not acceptable by the simulation engine (same as Simulink). In other words, these values must match the setting values defined by the user application, if the values are greater than 0. If the value required by the setting is 0, the corresponding parameter must be set to 1. Simulink does not requie these since the settings can be done directly from the user application.

In order to help the user to put the expected values, a printout of those setting values from user application is provided when starting the simulation.



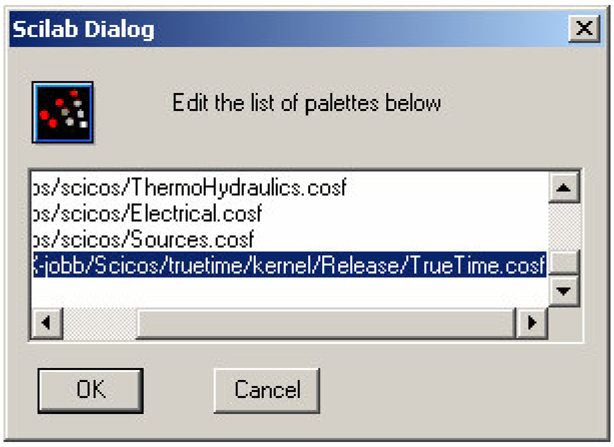
```
-------------------------------------------------------
                TrueTime, Version 1.4
                  Copyright (c) 2006
     Martin Ohlin, Dan Henriksson and Anton Cervin
          Department of Automatic Control, LTH
                Lund University, Sweden
-------------------------------------------------------
Number of inputs: 4
Inputs=2 Triggers=0 Networks=0 EnergyLevel=0
Number of outputs: 5
Outputs=1 Handlers=1+1=2 Monitors=0*2=0 EnergyConsumption=0
Output[2] size must equal to 2
_
```

Figure 36 – Printout of input/output sizes set in the user applicatiion

Any value that does not match the required setting will generate a message for the user to change to the expected value. In Figure 36, it is Output[2]: number of scheduler must be 2 instead of 1.

The works performed to develop this TrueTime kernel user interface in Scicos are as follows:

1. It is assumed that only one working directory is used to put all working files. Set this working directory as the Scicos current directory. The purpose is only to simplify the work.
2. Program the interface code function based on the Scicos standard and name the program file as *truetimekernel.sci*. It is important to have the same name on the file as the name of the function exposed from the compiled C/C++. The code listing is in chapter 6.3.
3. Make the *ttkernel.dll* available and link the function *truetimekernel* in that .dll file from Scilab. Use the same procedures mentioned in chapter 3.1.2.
4. Give command *genlib('scstruetimelib',pwd());* in Scilab to generate the executable library *truetimekernel.bin* from the file *truetimekernel.sci*. The command itself will actually convert all available .sci files in the current directory. Now, the truetimekernel block is available as a library object.
5. To make a TrueTime library to collect all TrueTime objects/blocks in the future (in this case only one available: TrueTime Kernel), a new Scicos diagram is created and the intended TrueTime blocks are added into it (use pulldown menu Edit->Add new block). Save the diagram as .cosf file, e. g. truetime.cosf. This will be the TrueTime library.
6. To configure so that the TrueTime library is accessible from Scicos palette list, use pulldown menu Edit->Pal editor, and update the list with reference to the library complete with its path. See Figure 37.


Scilab Dialog

Edit the list of palettes below

```
os/scicos/ThermoHydraulics.cosf
os/scicos/Electrical.cosf
os/scicos/Sources.cosf
-jobb/Scicos/truetime/kernel/Release/TrueTime.cosf
```

Figure 37 – Scilab dialog window for the list of palettes.

7. The TrueTime library must be reloaded everytime Scilab is restarted to make it available in the Scicos diagram editor. For this purpose, the best alternative is to make Scilab init file called *scilab.ini* and include the command *load('lib').* The path of the library must be added in case the current directory does not contain the library. The init file must be located in the startup path of Scilab.

3.2.6 Excluded functions

The functions of TrueTime kernel below are excluded when porting to Scilab/Scicos. The reasons of exclusion are either due to MATLAB specific function or network block related since the network block is not part of the porting either.

Functions	Allocation	Description	Reason of Exclusion
ttInitNetwork(I)	initnetwork.cpp initnetwork2.cpp	Initialize the TrueTime network interface	Network block related
ttGetInitArg()	getinitarg.cpp	Get initial argument from MATLAB interface	MATLAB specific function
ttCallBlockSystem(TH)	callblocksystem.cpp	Call a Simulink block diagram from within a code function	MATLAB specific function
ttSendMsg(TH)	sendmsg.cpp	Send a message over a network	Network block related
ttGetMsg(TH)	getmsg.cpp	Get a message that has been received over a network.	Network block related
ttSendNetwork-Parameter(ITH)	setnetwork-parameter.cpp	Set a specific network parameter on a per node basis	Network block related
ttAbortSimulation(TH)	abortsimulation.cpp	Stop the current simulation, causing an error	MATLAB specific function
ttDiscardUnsent-Messages(IT)	discardunsent.cpp	Delete unset messages in the network queue	Network block related
ttSetData(TH)	setdata.cpp	Update the local memory data structure associated with a specific task	MATLAB specific function
ttGetData(TH)	getdata.cpp	Retrieve the local memory data structure associated with a specific task	MATLAB specific function

3.3 Examples

3.3.1 Simple Timer – Sinus Sampler

This example is intended to give simple test on the TrueTime porting result at the beginning. There is no task involved in this case since the sampling was done using timer, but the timer activates an interrupt handler. The user application code is:

```
double mycode(int seg, void* data) {
  switch(seg) {
    case 1:
      printf("Task MYCODE running at %f\n", ttCurrentTime());
      ttAnalogOut(1,ttAnalogIn(1));
      printf("AnalogIn ch1=%f\n", ttAnalogIn(1));
      return 0.1;
    default:
      return FINISHED;
  }
}

void init() {
  ttInitKernel(1, 1, FP);
  ttCreateInterruptHandler("handler1", 1.0, mycode);
  ttCreatePeriodicTimer("timer1",0.0,1.0,"handler1");
}

void cleanup() {}
```

Figure 38 – Code listing of Simple Timer (Sinus Sampler)

The function *init()* initialize the kernel with settings 1 input, 1 output and priority function FP. It creates a periodic timer called "timer1" with period=1.0 and start time at 0.0, which activates an interrupt handler called "handler1" with priority=1 to execute the function *mycode*.
The user function *mycode* consists of two segments. The first segment makes a print out of the given text, reads the input channel 1 and copies its value to the output channel 1. The segment returns its execution time=0.1. The second segment does nothing and returns negative values (FINISHED) indicating end of the function.
The Scicos simulation diagram is modeled as follows:

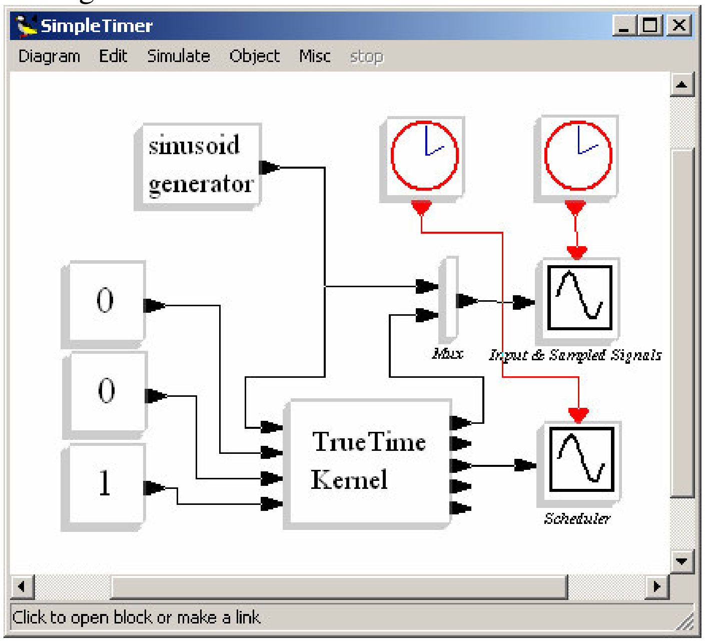


Figure 39 – Simulation diagram for Simple Timer (Sinus Sampler)

The displays for input/output signals and for the scheduler are connected to different activator blocks since they require different frequencies of activation signals. The scheduler needs higher frequency than the input/output signals.

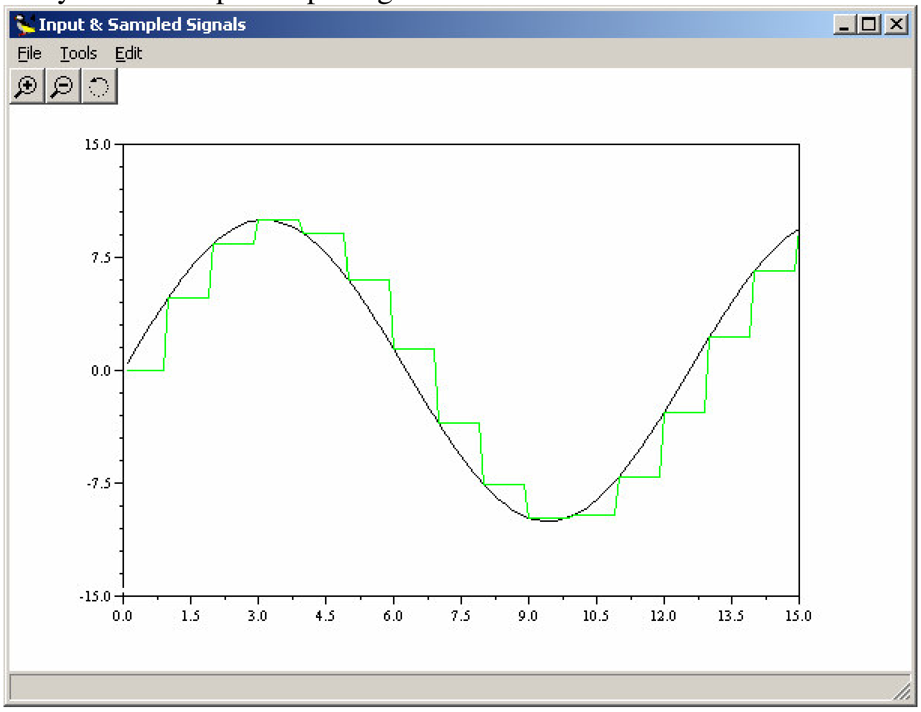


Figure 40 – The input signal (black color) and the output signal (green color)

Figure 40 shows the graphical result by mapping the output signal on the original sinusoidal input signal. The result indicates that the input signal is sampled with sampling time = 1. This is confirmed by the scheduler output showing the activity of the scheduled handler in Figure 41. There is no scheduled task displayed in the scheduler since the sampler was built using Periodic Timer and Interrupt Handler instead of Periodic Task.

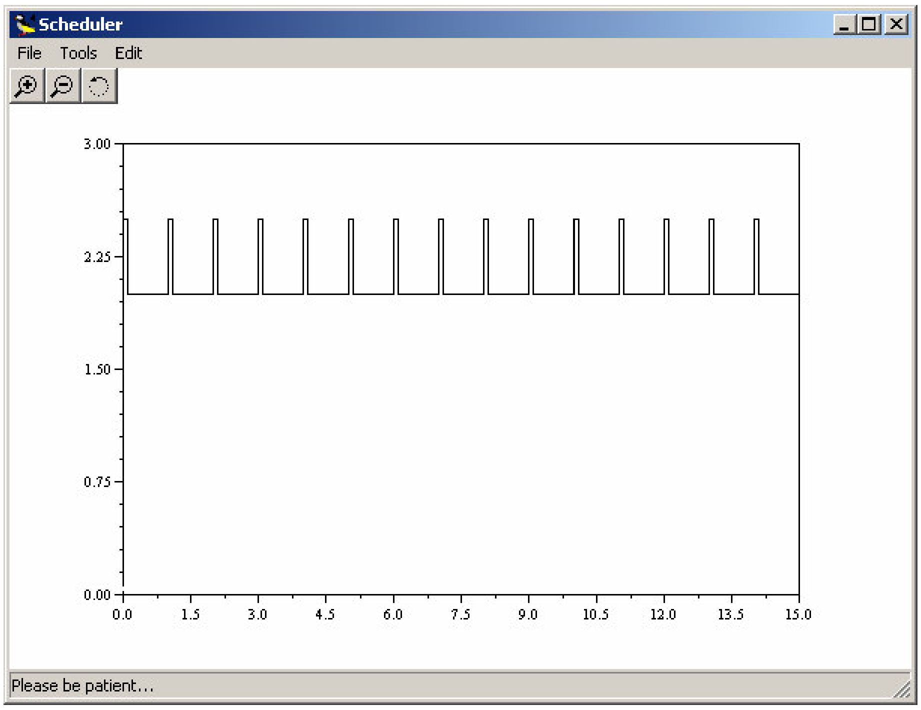


Figure 41 – Scheduler output showing the scheduled interrupt handler

3.3.2 SimplePID

This example considers simple PID control of a DC-servo process, and is intended to give a basic introduction to the TrueTime simulation environment. The process is controlled by a controller task implemented in a TrueTime kernel block. Three different implementations of the controller task are provided to show the different ways to implement periodic activities.

The DC-servo is described by the continuous-time transfer function

$$G(s) = \frac{1000}{s(s+1)}$$

The PID-controller is implemented according to the following equations

$$P(k) = K \cdot (r(k) - y(k))$$

$$I(k+1) = I(k) + \frac{Kh}{T_i}(r(k) - y(k)) \qquad a_d = \frac{T_d}{Nh + T_d}$$

$$D(k) = a_d D(k-1) + b_d(y(k-1) - y(k)) \qquad b_d = \frac{NKT_d}{Nh + T_d}$$

$$u(k) = P(k) + I(k) + D(k)$$

The controller parameters were chosen to give the system closed-loop bandwidth w$_c$=20rad/s, and relative damping, ζ=0.7.

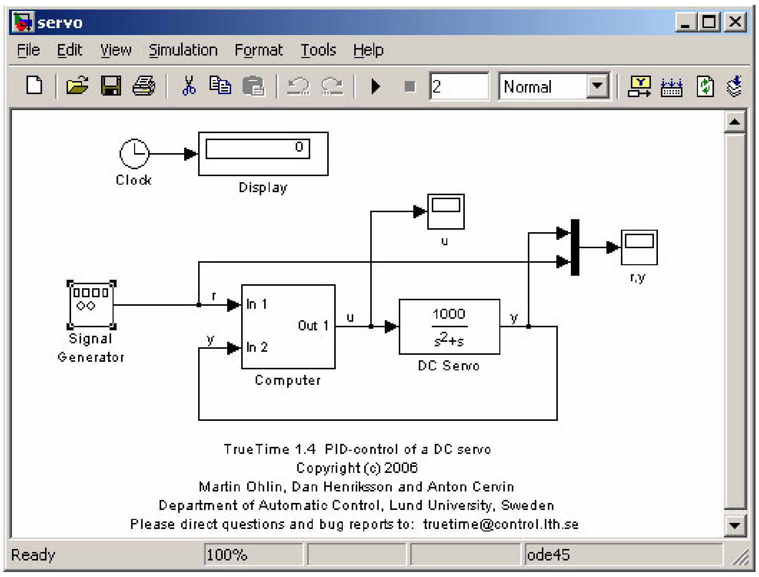
To choose the different implementations is done by uncommenting the desired implementation code. Those three implementations are:

- PID code 1 uses the TrueTime built-in support for periodic task.
- PID code 2 implements the periodic task by using the TrueTime primitive ttSleepUntil.
- PID code 3 implements the periodic task by using a periodic timer. The associated interrupt handler samples the process and triggers the task jobs. The handler and the controller task communicate using a mailbox.

The written code for user application is:

```
// PID-control of a DC servo process.
//
// This example shows four ways to implement a periodic controller
// activity in TrueTime. The task implements a standard
// PID-controller to control a DC-servo process (2nd order system).

// PID data structure used in Implementations 1a, 2, and 3 below.
class PID_Data {
public:
  struct { // states
    double u, Iold, Dold, yold, t; // t used only in Implementation 2 below
  } s;

  struct { // params
    double K, Ti, Td, N, h;
    int rChan, yChan, uChan;
  } p;
};

// calculate PID control signal and update states
void pidcalc(PID_Data* d, double r, double y) {
  double P = d->p.K*(r-y);
  double I = d->s.Iold;
  double D = d->p.Td/(d->p.N*d->p.h+d->p.Td)*d->s.Dold+d->p.N*d->p.K*d->p.Td/(d->p.N*d->p.h+d-
>p.Td)*(d->s.yold-y);

  d->s.u = P + I + D;
  d->s.Iold = d->s.Iold + d->p.K*d->p.h/d->p.Ti*(r-y);
  d->s.Dold = D;
  d->s.yold = y;
};

// ---- PID code function for Implementation 1 ----
double pidcode1(int seg, void* data) {

  double r, y;
  PID_Data* d = (PID_Data*) data;

  switch (seg) {
  case 1:
    r = ttAnalogIn(d->p.rChan);
    y = ttAnalogIn(d->p.yChan);
    pidcalc(d, r, y);
    return 0.002;
  default:
    ttAnalogOut(d->p.uChan, d->s.u);
    return FINISHED;
  }
}

// ---- PID code function for Implementation 2 ----
double pidcode2(int seg, void* data) {

  double r, y;
  PID_Data* d = (PID_Data*) data;

  switch (seg) {
  case 1:
    d->s.t = ttCurrentTime();
    return 0.0;
  case 2:
    r = ttAnalogIn(d->p.rChan);
    y = ttAnalogIn(d->p.yChan);
    pidcalc(d, r, y);
    return 0.002;
  case 3:
    ttAnalogOut(d->p.uChan, d->s.u);
    // Sleep
    d->s.t += d->p.h;
    ttSleepUntil(d->s.t);
    return 0.0;
  case 4:
    ttSetNextSegment(2); // loop
    return 0.0;
  }

  return FINISHED; // to supress compilation warnings
}

// ---- PID code function for Implementation 3 ----
double pidcode3(int seg, void* data) {

  double r;
  double *y;
  PID_Data* d = (PID_Data*) data;

  switch (seg) {
  case 1:
    r = ttAnalogIn(d->p.rChan);
    y = (double*) ttTryFetch("Samples");
    pidcalc(d, r, *y);
    delete y;
    return 0.0018;
  case 2:
    ttAnalogOut(d->p.uChan, d->s.u);
    return FINISHED;
  }

  return FINISHED; // to supress compilation warnings

}

// ---- Sampler code function for Implementation 3 ----
double samplercode(int seg, void* data) {

  double y;
  int* d = (int*) data;

  switch (seg) {
  case 1:
    y = ttAnalogIn(*d);
    ttTryPost("Samples", new double(y)); // put sample in mailbox
    ttCreateJob("pid_task");  // trigger task job
    return 0.0002;
  case 2:
    return FINISHED;
  }

  return FINISHED; // to supress compilation warnings
}

#define NBROFINPUTS 2
#define NBROFOUTPUTS 1
#define SCHEDULER prioFP

PID_Data *d;
int *hdl_data = NULL;

void init() {

  // Initialize TrueTime kernel
  ttInitKernel(NBROFINPUTS, NBROFOUTPUTS, SCHEDULER);

  // Task attributes
  double period = 0.006; // original = 0.006;
  double offset = 0.0; // start of first task instance
  double prio = 1.0;
  double deadline = 0.003; // only used by implementation 2 and 3

  // Create task data (local memory)
  d = new PID_Data;
  d->p.K = 0.96;
  d->p.Ti = 0.12;
  d->p.Td = 0.049;
  d->p.N = 10.0;
  d->p.h = period;
  d->s.u = 0.0;
  d->s.t = 0.0; // only used in Implementation 2 below
  d->s.Iold = 0.0;
  d->s.Dold = 0.0;
  d->s.yold = 0.0;
  d->p.rChan = 1;
  d->p.yChan = 2;
  d->p.uChan = 1;

  // IMPLEMENTATION 1: using the built-in support for periodic tasks
  //
  // ttCreatePeriodicTask("pid_task", offset, period, prio, pidcode1, d);

  // IMPLEMENTATION 2: sleepUntil and loop back
  //
  // ttCreateTask("pid_task", deadline, prio, pidcode2, d);
  // ttCreateJob("pid_task");

  // IMPLEMENTATION 3: sampling in timer handler, triggers task job

   hdl_data = new int(2); // y_chan for reading samples
   ttCreateInterruptHandler("timer_handler", prio, samplercode, hdl_data);
   ttCreatePeriodicTimer("timer", offset, period, "timer_handler");

   ttCreateMailbox("Samples", 10);

   ttCreateTask("pid_task", deadline, prio, pidcode3, d);

}

void cleanup() {

  delete d;
  if (hdl_data) delete hdl_data;

}
```

The Simulink simulation diagram is shown in Figure 42 below. The Computer is super block using TrueTime kernel block as shown in Figure 43 and the results are displayed in Figure 44.



TrueTime 1.4 PID-control of a DC servo
Copyright (c) 2006
Martin Ohlin, Dan Henriksson and Anton Cervin
Department of Automatic Control, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se

Figure 42 – Simulink simulation diagram for Simple PID

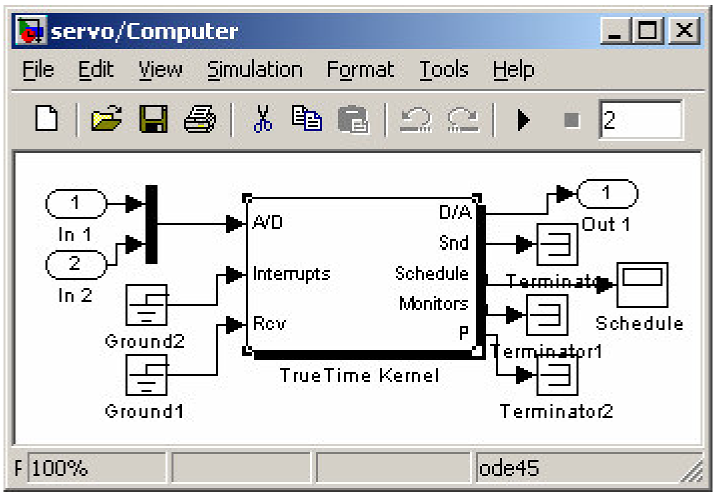


Figure 43 – Simulink computer block designed using TrueTime kernel block

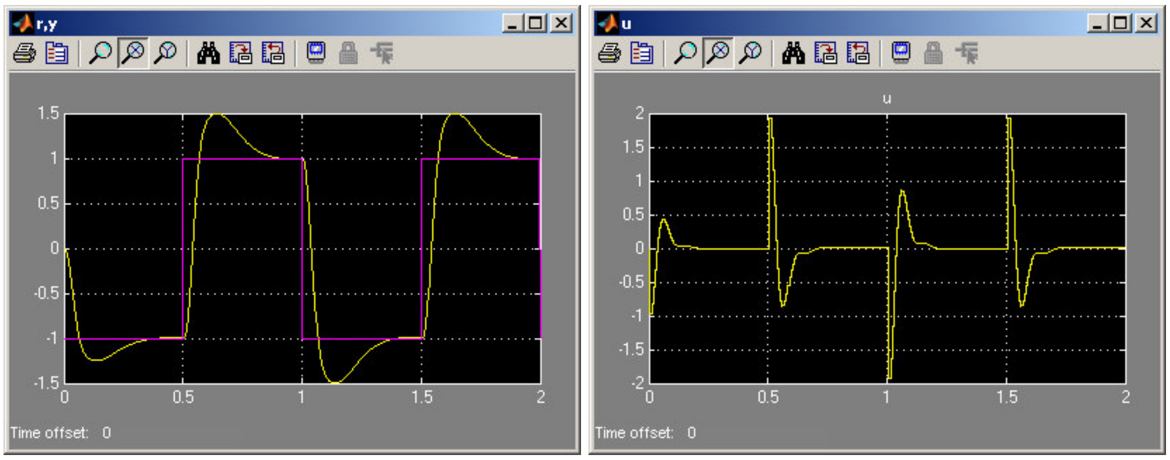


Figure 44 – The result of Simple PID (PID code 1, 2 and 3) displaying reference signal *r* (left pink color), measured signal *y* (left yellow color) and the control signal *u* (right yellow color)

The corresponding simulation diagram for Simple PID in the Scicos environment is shown in Figure 45.

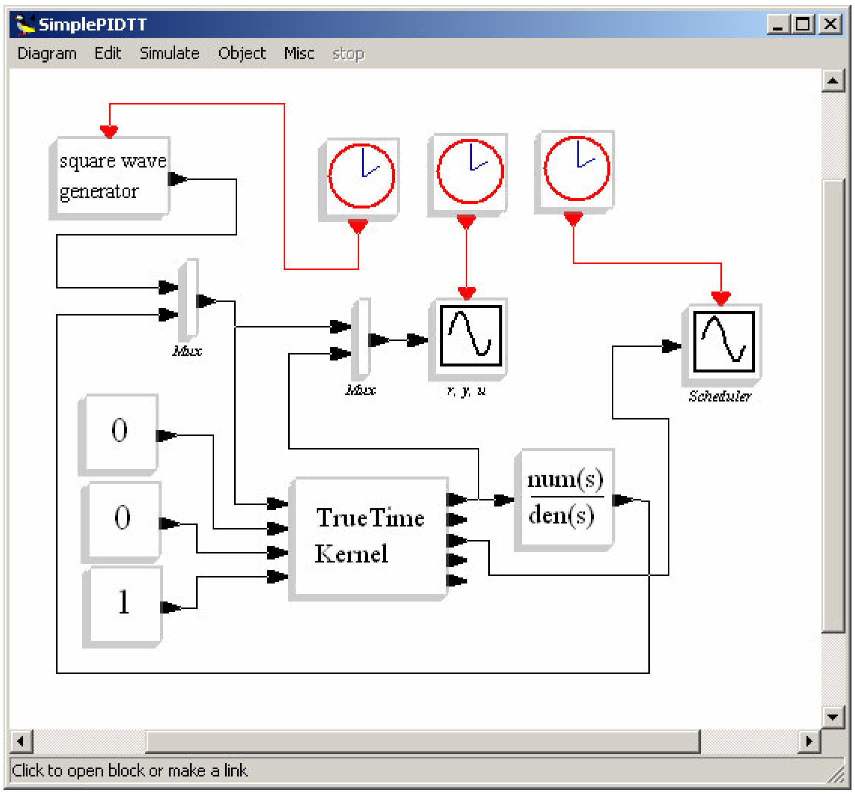
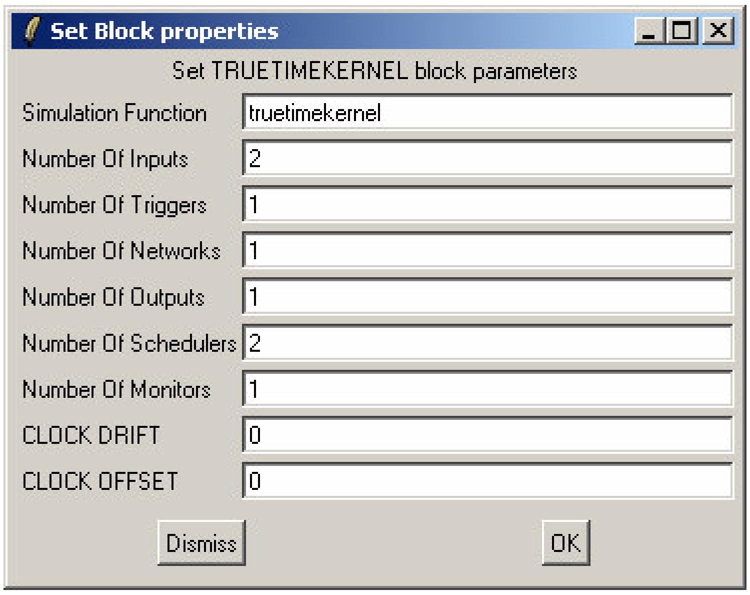


Figure 45 – Scicos simulation diagram for Simple PID

The diagram is not designed using super block feature as in Simulink, although Scicos has the feature. The simulated process block in Scicos can not display the actual equations as in Simulink. Regardless of apparent lack of the graphical features, the simulation diagram above is configured with the same settings as the Simulink simulation diagram to obtain comparable results.

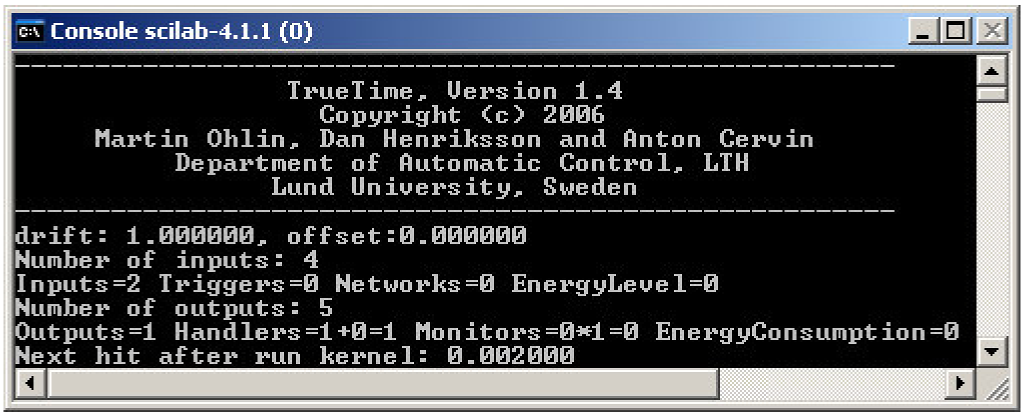

Set Block properties

Set TRUETIMEKERNEL block parameters

Simulation Function	truetimekernel
Number Of Inputs	2
Number Of Triggers	1
Number Of Networks	1
Number Of Outputs	1
Number Of Schedulers	2
Number Of Monitors	1
CLOCK DRIFT	0
CLOCK OFFSET	0

Figure 46 – TrueTime kernel settings for Simple PID simulations

Figure 46 shows the settings for the TrueTime block to simulate the Simple PID. Number of inputs is 2 since the block receives a reference signal r and measured signal from the process. There are no triggers and networks, but the settings have to be 1 as discussed in chapter 3.2.5. Number of output is only one, the control signal u. The handler is 1 since one task is created and no interrupt used. No monitor is applied. All these numbers are confirmed by the printout below in the Scicos console.



```
-------------------------------------------------------
                 TrueTime, Version 1.4
                   Copyright (c) 2006
     Martin Ohlin, Dan Henriksson and Anton Cervin
          Department of Automatic Control, LTH
                Lund University, Sweden
-------------------------------------------------------
drift: 1.000000, offset:0.000000
Number of inputs: 4
Inputs=2 Triggers=0 Networks=0 EnergyLevel=0
Number of outputs: 5
Outputs=1 Handlers=1+0=1 Monitors=0*1=0 EnergyConsumption=0
Next hit after run kernel: 0.002000
```

Figure 47 – Printout of initial settings for Simple PID

The results on PID code 1 and PID code 2 in Scicos are the same shown in Figure 48, although they have different implementations. Simulink confirms the correctness of these results (refer to Figure 44).

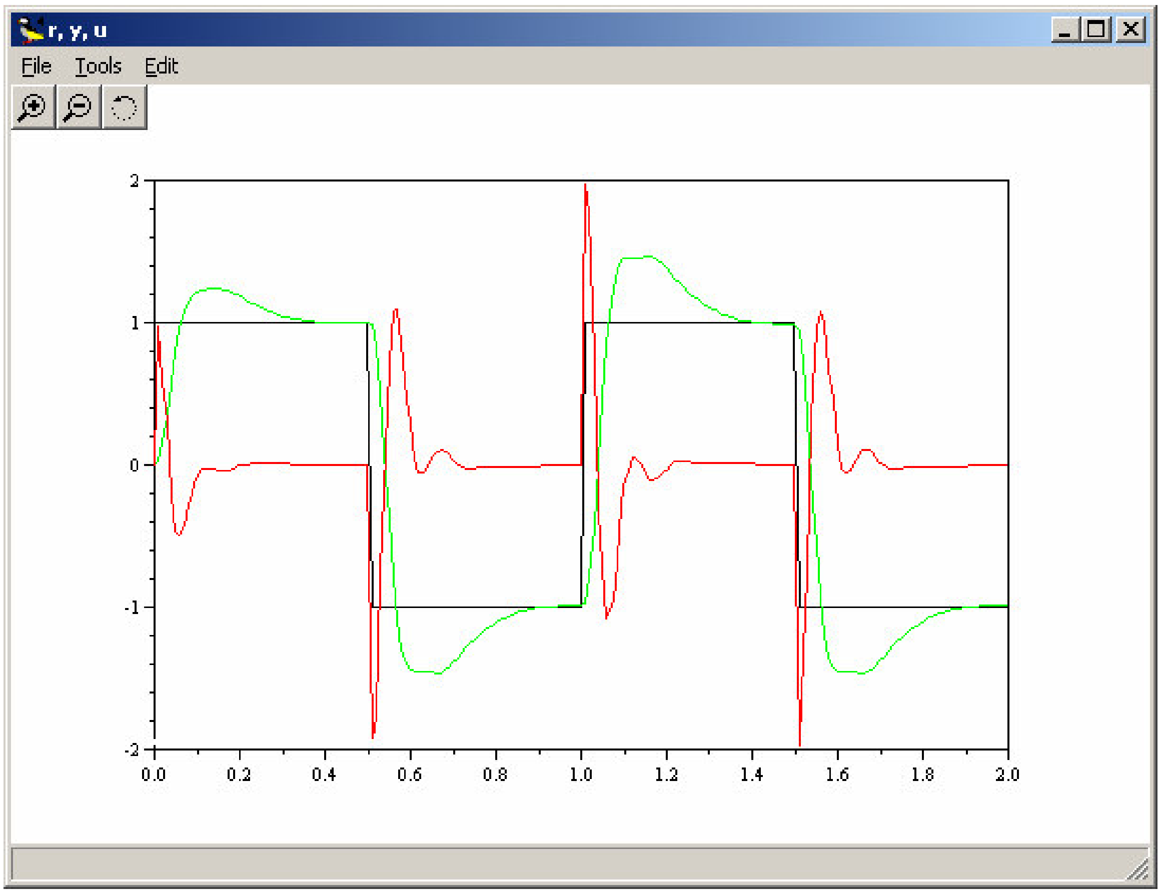


Figure 48 – The output results of PID code 1 and 2, reference signal *r* (black color), measured signal *y* (green color) and control signal *u* (red color)

As mentioned before, only one task is introduced here and this is confirmed by the output of scheduler as shown in Figure 49.

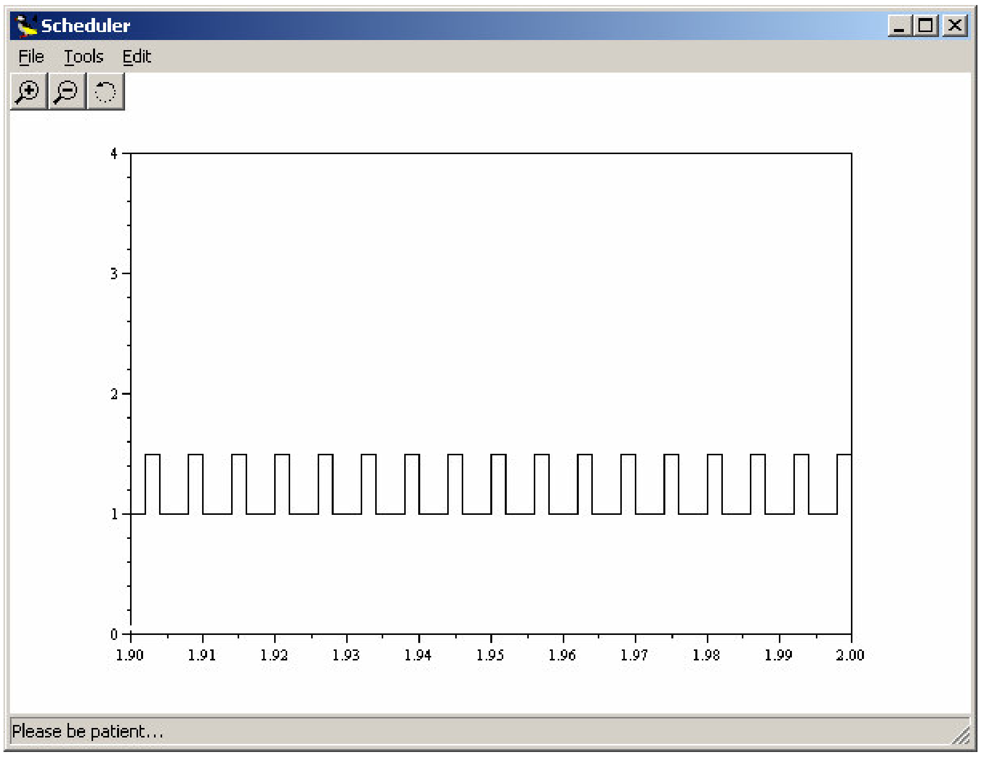


Figure 49 – Output of scheduler when simulating PID code 1 and 2

When simulating PID code 3, the setting of the number of Scheduler must be changed to 2, because in this case one task and one interrupt are scheduled.

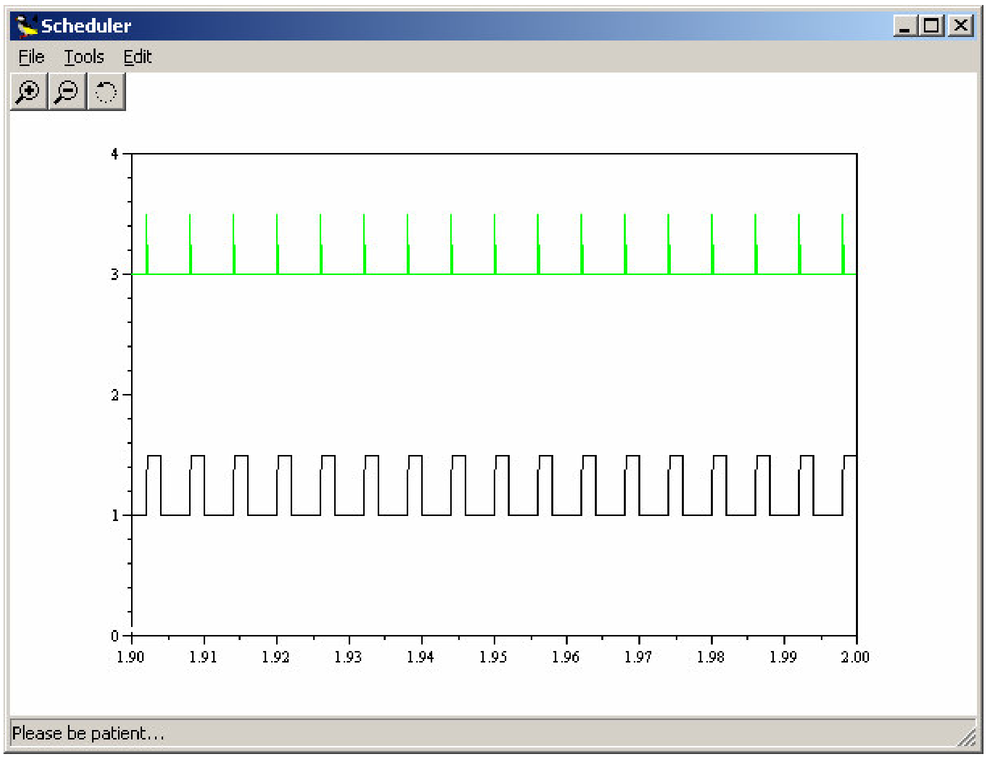


Figure 50 – Output of scheduler when simulating PID code 3 displaying one task (black color) and one interrupt (green color)

3.3.3 ThreeServos

This example extends the simple PID control example from the previous section to the case of three PID-tasks running concurrently on the same CPU controlling three different DC-servo systems. The effect of the scheduling policy on the global control performance is demonstrated. The user application code is:

```
// Task scheduling and control.
//
// This example extends the simple PID control example (located in
// $DIR/examples/simple_pid) to the case of three PID-tasks running
// concurrently on the same CPU controlling three different servo
// systems. The effect of the scheduling policy on the global control
// performance is demonstrated.

// PID data structure
class PID_Data {
public:
  struct { // states
    double u, Iold, Dold, yold;
  } s;

  struct { // params
    double K, Ti, Td, N, h;
    int rChan, yChan, uChan;
  } p;
};

// calculate PID control signal and update states
void pidcalc(PID_Data* d, double r, double y) {
  double P = d->p.K*(r-y);
  double I = d->s.Iold;
  double D = d->p.Td/(d->p.N*d->p.h+d->p.Td)*d->s.Dold+d->p.N*d->p.K*d->p.Td/(d->p.N*d->p.h+d-
>p.Td)*(d->s.yold-y);

  d->s.u = P + I + D;
  d->s.Iold = d->s.Iold + d->p.K*d->p.h/d->p.Ti*(r-y);
  d->s.Dold = D;
  d->s.yold = y;
};

// --------- Generic code function ----------
double pidcode(int seg, void* data) {

  PID_Data* d = (PID_Data*) data;

  switch (seg) {
  case 1:
    pidcalc(d, ttAnalogIn(d->p.rChan), ttAnalogIn(d->p.yChan));
    return 0.001;
  case 2:
    ttAnalogOut(d->p.uChan, d->s.u);
    return FINISHED;
  }

  return FINISHED; // to supress compilation warnings
}

#define NBROFINPUTS 6
#define NBROFOUTPUTS 3
#define SCHEDULER prioRM

// Task parameters
double periods[] = {0.006, 0.005, 0.004};
char* names[] = {"pid_task1", "pid_task2", "pid_task3"};
int rChans[] = {1, 3, 5};
int yChans[] = {2, 4, 6};
int uChans[] = {1, 2, 3};

PID_Data *d[3];

void init() {

  // Initialize TrueTime kernel
  ttInitKernel(NBROFINPUTS, NBROFOUTPUTS, SCHEDULER);
```

```
    // Create the three tasks
    for (int i = 0; i < 3; i++) {
      d[i] = new PID_Data;
      d[i]->p.K = 0.96;
      d[i]->p.Ti = 0.12;
      d[i]->p.Td = 0.049;
      d[i]->p.N = 10;
      d[i]->p.h = periods[i];
      d[i]->s.u = 0.0;
      d[i]->s.Iold = 0.0;
      d[i]->s.Dold = 0.0;
      d[i]->s.yold = 0.0;
      d[i]->p.rChan = rChans[i];
      d[i]->p.yChan = yChans[i];
      d[i]->p.uChan = uChans[i];

      // Offset=0 and prio=1 for all tasks
      ttCreatePeriodicTask(names[i], 0.0, periods[i], 1.0, pidcode, d[i]);
    }
  }

  void cleanup() {

    for (int i = 0; i < 3; i++) {
      delete d[i];
    }
  }
```

The simulation diagram modelled in Simulink is shown in Figure 51 below.

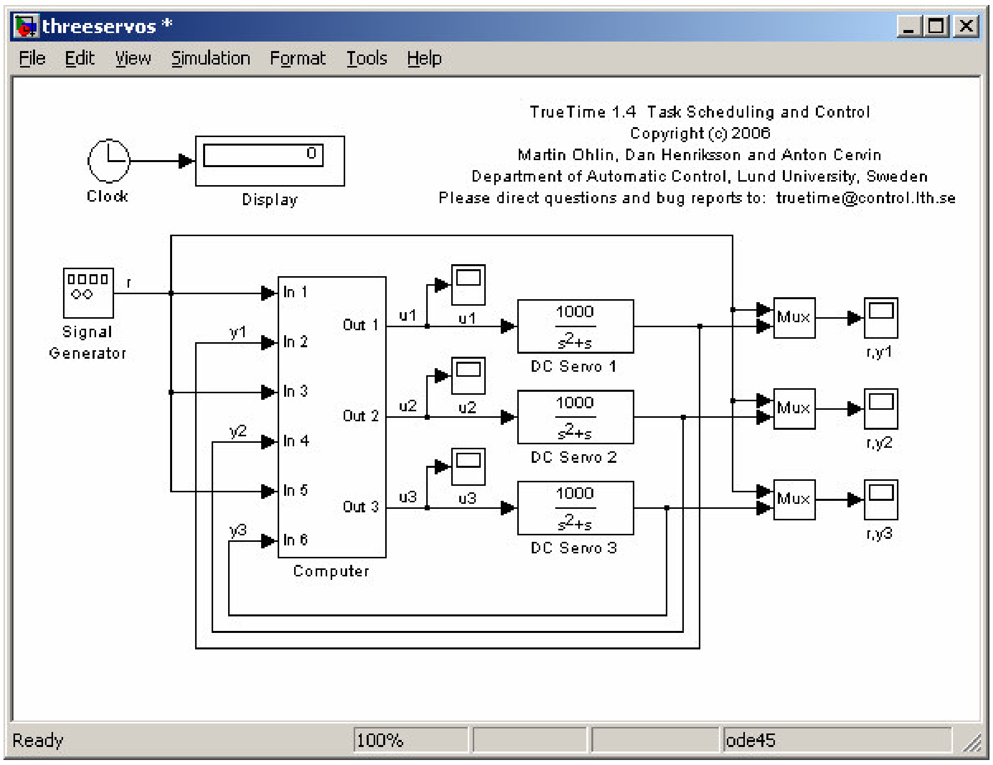

threeservos *
File Edit View Simulation Format Tools Help
TrueTime 1.4 Task Scheduling and Control
Copyright (c) 2006
Martin Ohlin, Dan Henriksson and Anton Cervin
Department of Automatic Control, Lund University, Sweden
Please direct questions and bug reports to: truetime@control.lth.se
Clock
0
Display
Signal Generator
r
y1
y2
y3
In 1
In 2
In 3
In 4
In 5
In 6
Out 1
Out 2
Out 3
Computer
u1
u2
u3
1000
s^2+s
DC Servo 1
DC Servo 2
DC Servo 3
Mux
r,y1
r,y2
r,y3
Ready
100%
ode45

Figure 51 - Simulation diagram for Three Servos in Simulink

As in the diagram of Simple PID (Figure 42), the computer block is a super block using TrueTime kernel and the diagram is quite similar with differences in the number of inputs and outputs. The corresponding simulation diagram in the Scicos environment is shown in Figure 52. One display block is provided for each servo to show the involved signals r, y and u. One more display is used to show the output of shedulers.

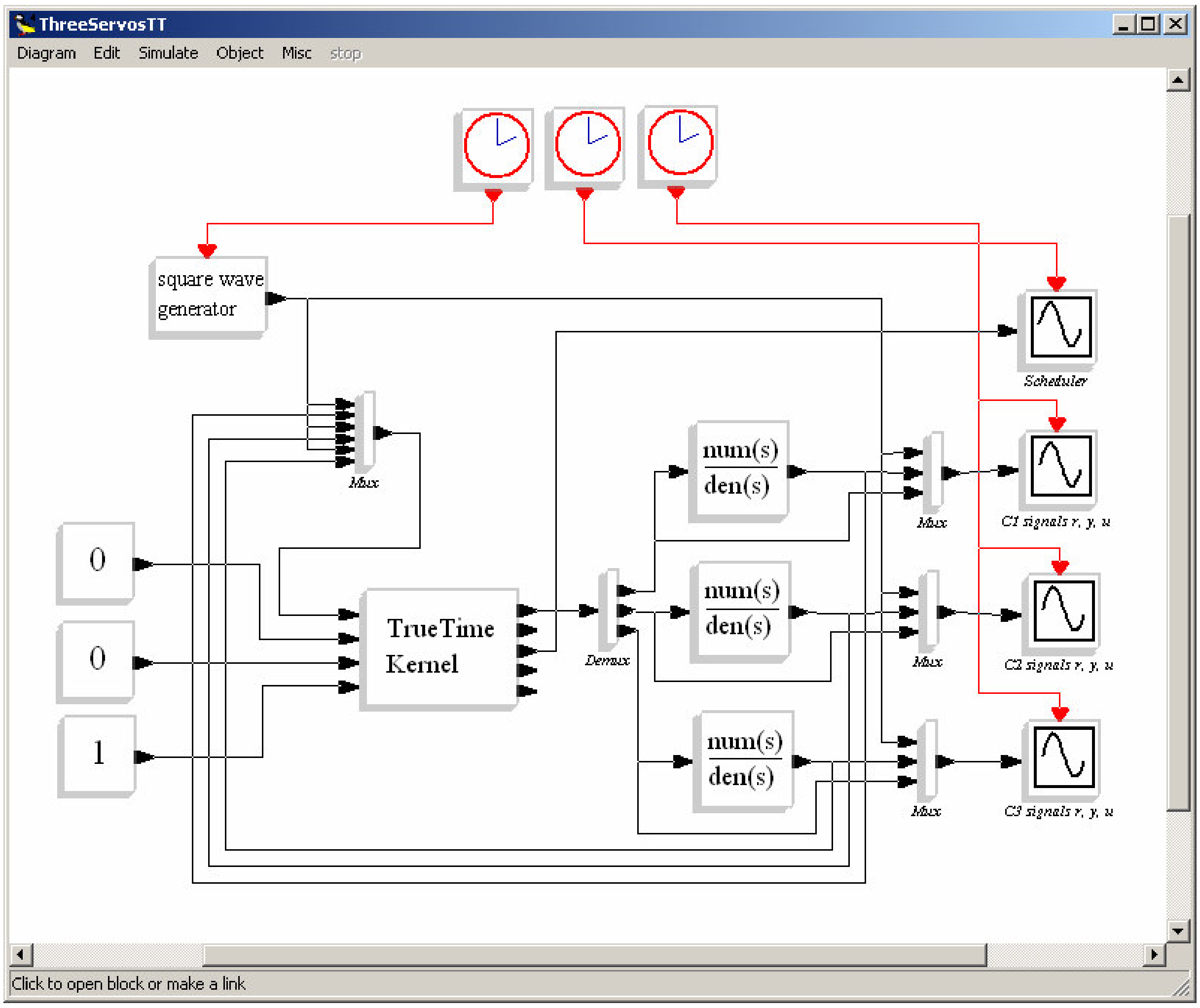
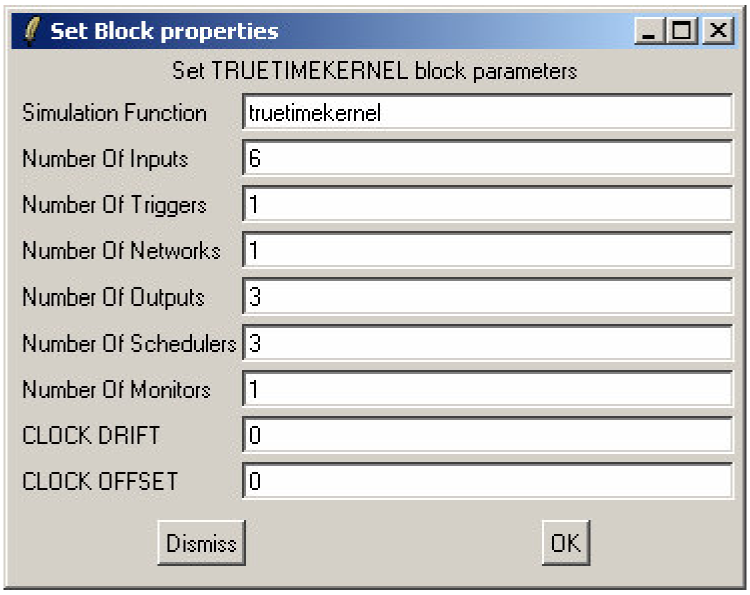


Figure 52 – Simulation diagram for Three Servos in Scicos

The settings of the kernel block must be adjusted as shown in the figure below.



Set TRUETIMEKERNEL block parameters

Simulation Function	truetimekernel
Number Of Inputs	6
Number Of Triggers	1
Number Of Networks	1
Number Of Outputs	3
Number Of Schedulers	3
Number Of Monitors	1
CLOCK DRIFT	0
CLOCK OFFSET	0

Figure 53 – Settings of the kernel block to simulate Three Servos

In the first test, the user application has the original PID execution time set to 0.002. There are three tasks created during initialization with periods set to 0.006, 0.005 and 0.004 to execute Controller 1, Controller 2 and Controller 3 respectively. The priority function is set to 'prioRM'. With these settings, the results show that Controller 1 has an unstable control, but the other two controllers work properly. Figure 54 and Figure 55 display the result.

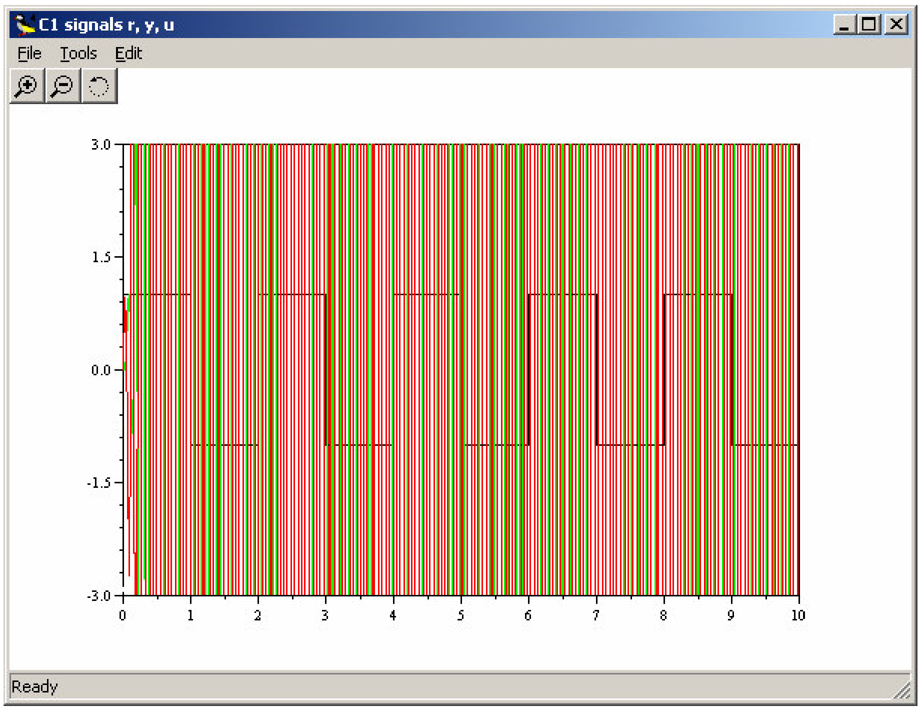


Figure 54 – Controller 1 with instable control

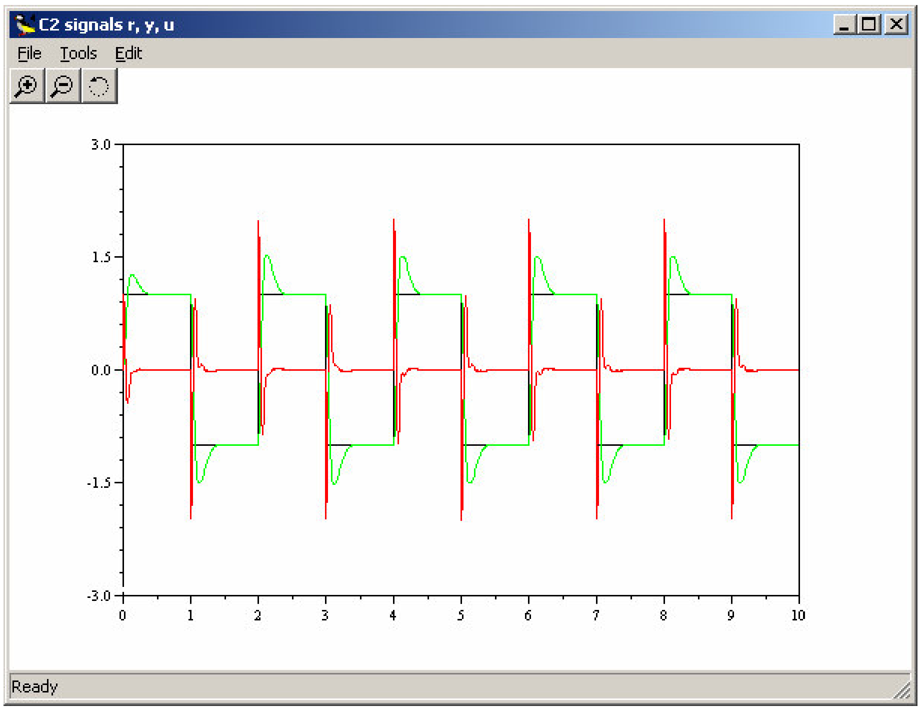


Figure 55 – Controller 2 with stable control, the same figure obtained for Controller 3

To understand the problem in Controller 1, the output of the schedulers need to be observed. Figure 56 below shows that the task in Controller 3 is working properly as designed and

Controller 2 is still working as scheduled even if it sometimes gets disturbances. Controller 1 is hardly possible to have a chance to complete its assignments. The kernel does not have sufficient time to accomplish all three tasks properly with the given execution times. This first test proves the capability of TrueTime to analyze problems in automatic control from the kernel perspective (and off course in the Scicos environment).

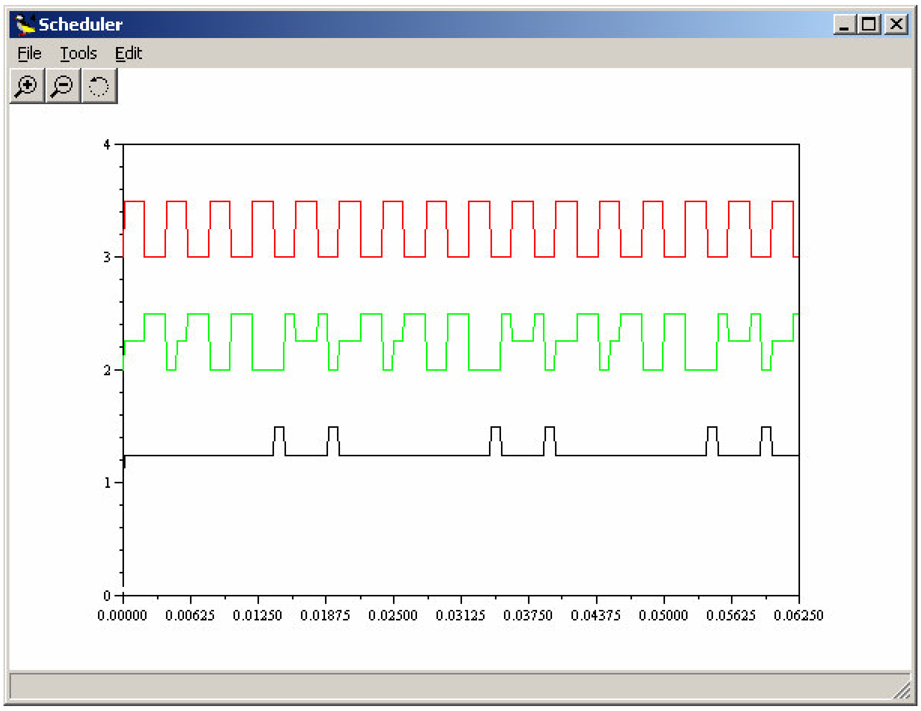


Figure 56 – Output of schedulers when simulating Three Servos with PID execution time 0.002, tasks of Controller 1, 2 and 3 indicated with black, green and red lines respectively

The next test is performed by changing the execution time for each PID to 0.001. The results show how Controller 1 becomes stable as the other two controllers.

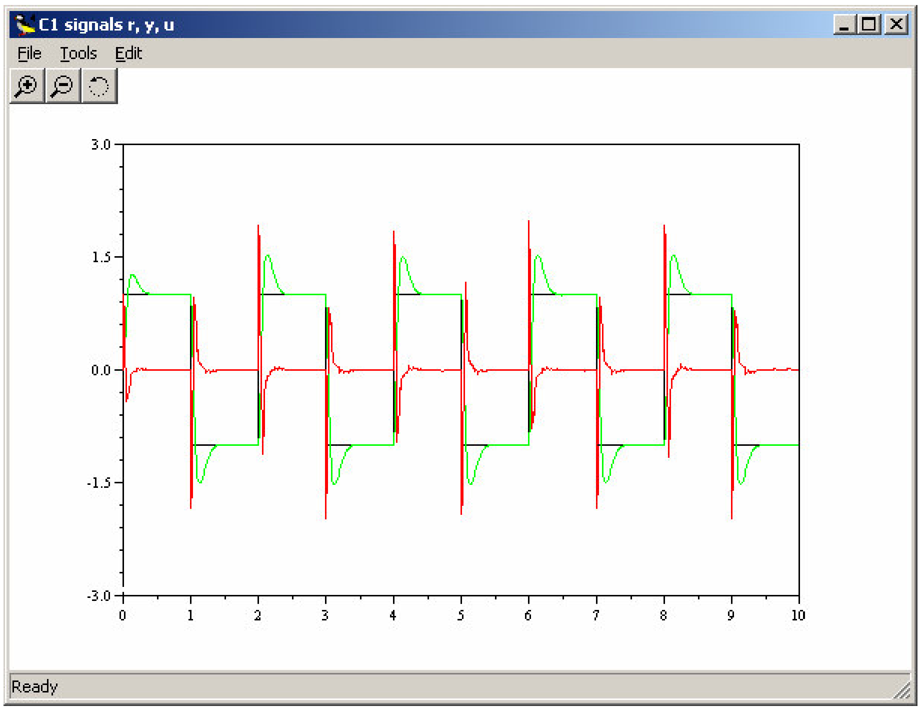


Figure 57 - Controller 1 gives stable control and so do Controller 2 and Controller 3

The kernel has enogh time to let each task executes its assignment properly as shown in Figure 58.

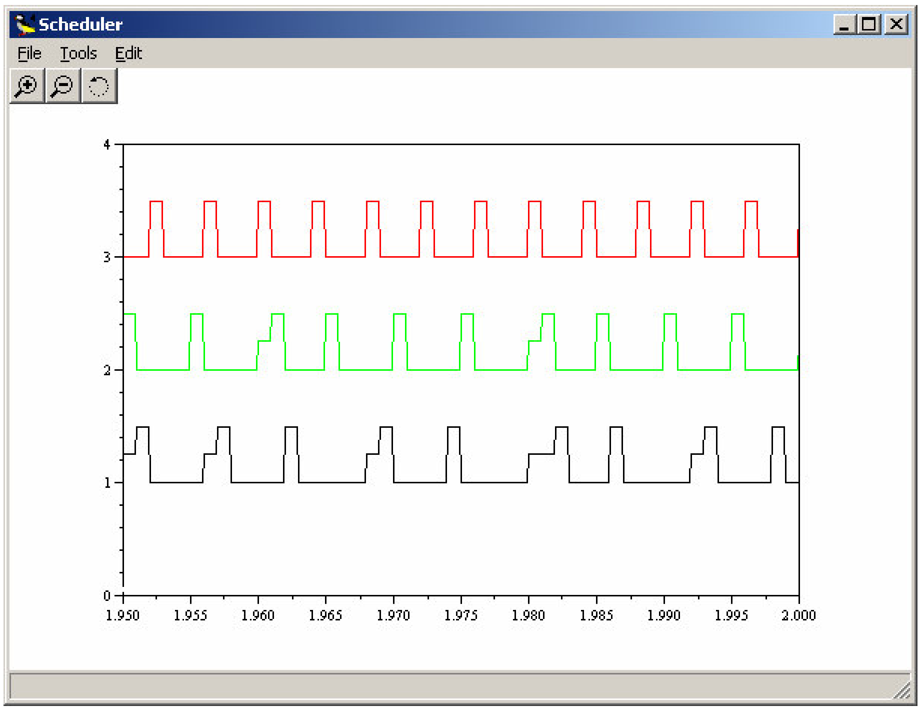


Figure 58 - Output of schedulers when simulating Three Servos with PID execution time 0.001, tasks of Controller 1, 2 and 3 indicated with black, green and red lines respectively

All results for simulating Three Servos as described above are the same as in Simulink.

4 Conclusions

4.1 Good story

The good story from this thesis is that all efforts made in this work yield a positive result at the end. The shortage of information about Scilab/Scicos in this particular subject is the main challenge to solve the problems given in this scope of work. Regular meeting with supervisor, Anton Cervin, is the main important key behind this success.

4.2 Difficulties

Some difficulties were met during accomplishing the project as follows:

- Insufficient Scilab/Scicos documentation for the subject focused in the project
- Product instability occurs sometimes in certain circumstances
- Not user-friendly compilation warning/message, which makes it difficult to trace the problems that occur.
- Sometime the command does not give any error message even if it is wrong. The error is identified when Scilab does not give any respond to any given commands.
- The graphics editor in Scicos is not supported by user-friendly tools either for development or modification.

4.3 Possible future work

Below are the possible future works to extend this porting work:

4.3.1 Improvement on the ported TrueTime kernel block

Improvement on the ported TrueTime kernel block is needed, especially regarding the user-friendliness aspect. One future work is to split the user application from the file ttkernel.cpp. The other improvement is to apply super block feature in the examples used above.

4.3.2 Calling PID parameters written in Scicos

The TrueTime original program contains code to read in data written in a MATLAB m-file. This seems possible to perform in Scicos with its .sce file. Further investigation is required.

4.3.3 Porting of TrueTime network blocks

The TrueTime network blocks are the next challenging works to do. There are two blocks available for MATLAB: network block and wireless network block.

4.3.4 Completing the network part in TrueTime Kernel block

As mentioned in chapter 3.2.6, there are several TrueTime kernel functions that are not included when porting the TrueTime kernel block. Completing the porting of those functions that related to the networking is an interesting future work.

5 References

1. Pärt-Enander, Eva, Anders Sjöberg, “Användarhandledning för MATLAB® 6.5”, Elanders Gotab, Stockholm, 2003, ISBN 91-506-1690-0.
2. Palm III, William J., ”Introduction to MATLAB® for Engineers”, The McGraw-Hill Companies, Inc., United States of America, 1998, ISBN 0-07-047328-5.
3. Campbell, Stephen L., Jean-Philippe Chancelier, Ramine Nikoukhah, “Modelling and Simulation in Scilab/Scicos”, Springer Science+Bussiness Media, New York, 2006, ISBN-10: 0-387-27802-8 and ISBN-13: 978-0387278025.
4. Cervin, A., D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén, “How does control timing affect performance?”, IEEE Control System Magazine, 23:3, pp. 16-30, 2003.
5. Ohlin, Martin, Dan Henriksson, Anton Cervin, “TrueTime 1.4 – Reference Manual”, Department of Automatic Control, Lund University, September 2006.
6. Åström, K. J. and T. Hägglund, ” PID Controllers: Theory, Design, and Tuning”, Instrument Society of America, Research Triangle Park, North Carolina, 1995.
7. Bates, Jonathan, Timothy Tompkins, “Using Visual C++ 6”, Que, United States of America, 1998, ISBN 0-7897-1635-6.
8. Templeman, Julian, Andy Olsen, “Microsoft Visual C++ .NET: Steg för steg”, Elanders Graphic Systems AB, Göteborg, 2002.
9. Kerninghan, Brian W., Dennis M Ritchie, ”The C Programming Language”, Prentice Hall, USA, 1988, ISBN 9780131103627.

6 Appendices

6.1 Appendix A: Code listing of ttkernel.h

```
/*
 * TrueTime, Version 1.4
 * Copyright (c) 2006
 * Martin Ohlin, Dan Henriksson and Anton Cervin
 * Department of Automatic Control, LTH
 * Lund University, Sweden
 */

extern "C" {
#include "scicos/scicos_block.h"
}

#ifndef __TT_KERNEL_H__
#define __TT_KERNEL_H__

// old="mex.h"
#include <stdio.h>
#include <string.h>
#include <math.h>

// old=#include "mexhelp.h"
#include "linkedlist.cpp"
#include "datanode.h"

#define EPS 1.0E-5  // Timing precision
#define INF 1.0E+5  // Maximum simulation time
#define FINISHED -1.0

enum { FP, RM, DM, EDF }; // supported scheduling policies
enum { ARRIVAL, RELEASE, START, SUSPEND, RESUME, FINISH }; // hooks
enum { IDLE, READY, RUNNING, WAITING, SLEEPING, SUSPENDED };   // usertask states
enum { UNUSED, OVERRUN, TIMER, NETWORK, EXTERNAL }; // handler types

#define NBRLOGS 10 // Maximum number of log entries for each task
// Log types (5 pre-defined)
enum { RESPONSETIMELOG=1, RELEASELATENCYLOG, STARTLATENCYLOG, EXECTIMELOG, CONTEXTRESTORELOG,
USERLOG };
#define NBRUSERLOGS (NBRLOGS - 5) // Number of user-defined log types

/* temp ===============
#include "ttnetwork.h"
======================= */

#include "task.h"
#include "log.h"
// old=#include "blockdata.h"
#include "usertask.h"


#include "timer.h"
#include "trigger.h"
/* temp ===============
#include "network.h"
======================= */
#include "handler.h"

#include "monitor.h"
#include "event.h"
#include "mailbox.h"
#include "semaphore.h"

void init();
void cleanup();
void truetimekernel(scicos_block *block, int flag);

class RTsys {
 public:
  char* name;

  bool init_phase;    // false when the simulation is running
  bool initialized;   // true if ttInitKernel has been called
  bool error;         // true if simulation should stop
  bool started;       // true after time zero
  bool mdlzerocalled; // true if mdlZeroCrossings has been called

  int nbrOfInputs;
  int nbrOfOutputs;
  int nbrOfTasks;
  int nbrOfHandlers;
  int nbrOfMonitors;
  int nbrOfTriggers;
  int nbrOfSchedTasks;
  int nbrOfSchedHandlers;
  int nbrOfSchedMonitors;

  double time;      // Current time in simulation
  double prevHit;   // Previous invocation of kernel
  double nextHit;   // Next invocation of kernel

  double *inputs;          // Vector of input port (analogin) values
  double *outputs;         // Vector of output port (analogout) values
  double *interruptinputs;    // External interrupt trigger signals
  double *oldinterruptinputs;

  double *taskSched;      // Vector of values for the usertask schedule
  double *handlerSched;   // Vector of values for the handler schedule
  double *monitorGraph;   // Vector of values for the monitor graph

  Task* running;          // Currently running task

  List* readyQ;   // usertasks and handlers ready for execution, prio-sorted
  List* timeQ;    // usertasks and handlers waiting for release, time-sorted

  List *taskList;      // List of datanodes with pointers to created tasks
  List *handlerList;   // List of datanodes with pointers to created handlers
  List *timerList;     // List of datanodes with pointers to timer handlers
  List *monitorList;   // List of datanodes with pointers to created monitors
  List *eventList;     // List of datanodes with pointers to created event
  List *triggerList;   // List of datanodes with pointers to ext. interrupt handlers
  List *mailboxList;   // List of datanodes with pointers to created mailboxes
  List *semaphoreList; // List of datanodes with pointers to created semaphores

  // temp=mxArray *initarg;                // Pointer to Kernel block init arg

  double (*prioFcn)(UserTask*);    // Priority function (see priofunctions.cpp)

  double contextSwitchTime;         // Time for a full context save/restore
  InterruptHandler* kernelHandler;  // Handler simulating context switches
  double contextSimTime;            // Execution time of handler code function
  UserTask* suspended;              // Last suspended usertask (context switch
                                    // if another task is resumed or started)

  // NETWORK
  int nbrOfNetworks;         // Number of TrueTime networks to which the
                             // kernel (node) is connected
  double *nwSnd;             // Send output
  double *networkinputs;     // Network interrupt trigger signals
  double *oldnetworkinputs;
  List *networkList;         // List of datanodes with pointers to network interrupt
                             // handlers. One handler for each network to which the
                             // kernel (node) is connected.

  // Function pointers
  double (*contextSwitchCode)(int, void*); // Code function for context switch handler
  double (*periodicTaskHandlerCode)(int, void*);  // Code function for handler
                                                  // generating periodic task jobs.

  int (*prioSort)(Node* , Node*);  // Sorting function for priority-sorted lists
  int (*timeSort)(Node* , Node*);  // Sorting function for time-sorted lists
                                   // (see compfunctions.cpp)

  void (*default_arrival)(UserTask*);  // Default kernel hooks
  void (*default_release)(UserTask*);  // (see defaulthooks.cpp)
  void (*default_start)(UserTask*);
  void (*default_suspend)(UserTask*);
  void (*default_resume)(UserTask*);
  void (*default_finish)(UserTask*);
```

```
  double (*prioFP)(UserTask*);          // Standard priority functions
  double (*prioRM)(UserTask*);          // (see priofunctions.cpp)
  double (*prioEDF)(UserTask*);
  double (*prioDM)(UserTask*);

  double energyLevel;       // Input from the battery
  double energyConsumption; // How much we consume
  double cpuScaling;        // How fast we run
  double clockDrift;        //
  double clockOffset;       // Offset from the nominal time

#ifdef KERNEL_MATLAB
  mxArray* rtsysptr;        // Pointer to global variable "_rtsys"
#endif

  RTsys();  // constructor
  ~RTsys(); // deconstructor
};


/**
 * RTsys Constructor
 */
RTsys::RTsys() {

  name = "NoName";

  init_phase = true;
  initialized = false;
  error = false;
  started = false;
  mdlzerocalled = false;

  nbrOfInputs = 0;
  nbrOfOutputs = 0;
  nbrOfTasks = 1;
  nbrOfHandlers = 0;
  nbrOfMonitors = 0;
  nbrOfTriggers = 0;
  nbrOfSchedTasks = 0;
  nbrOfSchedHandlers = 0;
  nbrOfSchedMonitors = 0;

  time = 0.0;
  prevHit = 0.0;
  nextHit = 0.0;

  inputs = NULL;
  outputs = NULL;
  interruptinputs = NULL;
  oldinterruptinputs = NULL;

  taskSched = NULL;
  handlerSched = NULL;
  monitorGraph = NULL;

  running = NULL;

  readyQ = NULL;
  timeQ = NULL;

  taskList = NULL;
  handlerList = NULL;
  timerList = NULL;
  monitorList = NULL;
  eventList = NULL;
  triggerList = NULL;
  mailboxList = NULL;
  semaphoreList = NULL;

  // old=initarg = NULL;

  prioFcn = NULL;

  contextSwitchTime = 0.0;
  contextSimTime = 0.0;
  kernelHandler = NULL;
  suspended = NULL;

  nbrOfNetworks = 0;
  nwSnd = NULL;
  networkinputs = NULL;
  oldnetworkinputs = NULL;
  networkList = NULL;

  energyLevel = 1;       // Energy level from the battery
  energyConsumption = 0; // How much we consume
  cpuScaling = 1;        // No scaling as default
  clockDrift = 1;        // No drift as default
  clockOffset = 0;       // No offset as default

#ifdef KERNEL_MATLAB
  rtsysptr = NULL;
#endif
}

/**
 * RTsys Destructor
 */
RTsys::~RTsys() {
  DataNode *dn, *tmp;

  if (inputs) delete[] inputs;
  if (outputs) delete[] outputs;
  if (interruptinputs) delete[] interruptinputs;
  if (oldinterruptinputs) delete[] oldinterruptinputs;
  if (taskSched) delete[] taskSched;
  if (handlerSched) delete[] handlerSched;
  if (monitorGraph) delete[] monitorGraph;
  if (nwSnd) delete[] nwSnd;
  if (networkinputs) delete[] networkinputs;
  if (oldnetworkinputs) delete[] oldnetworkinputs;

  if (readyQ) delete readyQ;
  if (timeQ) delete timeQ;

  // Delete all tasks and the tasklist
  if (taskList) {
    dn = (DataNode*) taskList->getFirst();
    while (dn != NULL) {
      UserTask* task = (UserTask*) dn->data;
      delete task;
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete taskList;
  }
  // Delete all handlers and the handlerlist
  if (handlerList) {
    dn = (DataNode*) handlerList->getFirst();
    while (dn != NULL) {
      InterruptHandler* hdl = (InterruptHandler*) dn->data;
      delete hdl;
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete handlerList;
  }
  // Delete the timerlist
  if (timerList) {
    dn = (DataNode*) timerList->getFirst();
    while (dn != NULL) {
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete timerList;
  }

  // Delete all monitors and the monitorlist
  if (monitorList) {
    dn = (DataNode*) monitorList->getFirst();
    while (dn != NULL) {
      Monitor* mon = (Monitor*) dn->data;
      delete mon;
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete monitorList;
  }
  // Delete all events and the eventlist
  if (eventList) {
    dn = (DataNode*) eventList->getFirst();
    while (dn != NULL) {
      Event* ev = (Event*) dn->data;
      delete ev;
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete eventList;
  }
  // Delete triggerlist
  if (triggerList) {
    dn = (DataNode*) triggerList->getFirst();
    while (dn != NULL) {
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete triggerList;
  }
  // Delete all mailboxes and the mailboxlist
  if (mailboxList) {
    dn = (DataNode*) mailboxList->getFirst();
    while (dn != NULL) {
      Mailbox* mb = (Mailbox*) dn->data;
      delete mb;
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete mailboxList;
  }
  // Delete all semaphores and the semaphorelist
  if (semaphoreList) {
    dn = (DataNode*) semaphoreList->getFirst();
    while (dn != NULL) {
      Semaphore* sem = (Semaphore*) dn->data;
      delete sem;
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete semaphoreList;
  }
  // Delete networklist
  /* temp ===============
  if (networkList) {
    dn = (DataNode*) networkList->getFirst();
    while (dn != NULL) {
      tmp = dn;
      dn = (DataNode*) dn->getNext();
      delete tmp;
    }
    delete networkList;
  }
             ======================= */

}

#endif // __TT_KERNEL_H__
```

6.2 Appendix B: Code listing of ttkernel.cpp

```
/*
 * TrueTime, Version 1.4
 * Copyright (c) 2006
 * Martin Ohlin, Dan Henriksson and Anton Cervin
 * Department of Automatic Control, LTH
 * Lund University, Sweden
 */

#include "ttkernel.h"

// ----- Main data structure ------

RTsys *rtsys; // global variable used by all instances of ttkernel

// ------- Internal functions used by kernel -------

#include "compfunctions.cpp"
#include "codefunctions.cpp"
#include "priofunctions.cpp"
#include "defaulthooks.cpp"
/* temp ===============
#include "initnetwork2.cpp"

#ifndef KERNEL_MATLAB

#define KERNEL_C
======================= */

// --- Initialization and creation ----

#include "initkernel.cpp"
#include "createtask.cpp"
#include "createpertask.cpp"
#include "createhandler.cpp"
#include "createtrigger.cpp"
/* temp ===============
#include "initnetwork.cpp"
======================= */
#include "createmonitor.cpp"
#include "createevent.cpp"
#include "createmailbox.cpp"
#include "createsemaphore.cpp"
#include "createlog.cpp"
#include "attachdlhandler.cpp"
#include "attachwcethandler.cpp"
#include "attachpriofcn.cpp"
#include "attachhook.cpp"
#include "noschedule.cpp"
#include "nonpreemptable.cpp"
/* temp ===============
#include "getinitarg.cpp"
======================= */

// ------- Real-time primitives -------

#include "createjob.cpp"
#include "killjob.cpp"
#include "createtimer.cpp"
#include "removetimer.cpp"
#include "analogin.cpp"
#include "analogout.cpp"
#include "sleep.cpp"
#include "entermonitor.cpp"
#include "exitmonitor.cpp"
#include "wait.cpp"
#include "notify.cpp"
#include "tryfetch.cpp"
#include "trypost.cpp"
#include "fetch.cpp"
#include "post.cpp"
#include "retrieve.cpp"
#include "take.cpp"
#include "give.cpp"
```

```
#include "lognow.cpp"
#include "logstart.cpp"
#include "logstop.cpp"
#include "currenttime.cpp"
#include "invokingtask.cpp"
#include "setnextsegment.cpp"
/* temp ===============
#include "callblocksystem.cpp"
#include "sendmsg.cpp"
#include "getmsg.cpp"
#include "setnetworkparameter.cpp"
#include "abortsimulation.cpp"
#include "discardunsent.cpp"
======================= */

// ---------- Sets and Gets ------------

#include "setdeadline.cpp"
#include "setabsdeadline.cpp"
#include "setpriority.cpp"
#include "setperiod.cpp"
#include "setbudget.cpp"
#include "setwcet.cpp"
/* temp ===============
#include "setdata.cpp"
======================= */
#include "getrelease.cpp"
#include "getdeadline.cpp"
#include "getabsdeadline.cpp"
#include "getpriority.cpp"
#include "getperiod.cpp"
#include "getbudget.cpp"
#include "getwcet.cpp"
/* temp ===============
#include "getdata.cpp"

#endif
======================= */

// temp=#ifdef KERNEL_MATLAB

// ----------------------------------------------
// ------ Executes an m-file code function ------
// ----------------------------------------------

// mxArray used to pass the segment to the code function
/* temp ===============
mxArray *segArray;
bool destroyed;

double executeCode(char *codeName, int seg, char* dataName) {
  double retval;
  mxArray *lhs[2];
  mxArray *rhs[2];

  *mxGetPr(segArray) = (double) seg;

  rhs[0] = segArray;
  if (dataName) {
    rhs[1] = mexGetVariable("global", dataName);
  } else {
    rhs[1] = mxCreateDoubleMatrix(1, 1, mxREAL);
    *mxGetPr(rhs[1]) = 0.0;
  }

  mexSetTrapFlag(1); // return control to the MEX file after an error
  lhs[0] = NULL;     // needed not to crash Matlab after an error
  lhs[1] = NULL;     // needed not to crash Matlab after an error
  if (mexCallMATLAB(2, lhs, 2, rhs, codeName) != 0) {
    rtsys->error = true;
    return 0.0;
  }

  if (mxGetClassID(lhs[0]) == mxUNKNOWN_CLASS) {
    printf("??? executeCode: execution time not assigned\n\n");
    printf("Error in ==> code function '%s', segment %d\n", codeName, seg);
    rtsys->error = true;
    return 0.0;
  }

  if (!mxIsDoubleScalar(lhs[0])) {
    printf("??? executeCode: illegal execution time\n\n");
    printf("Error in ==> code function '%s', segment %d\n", codeName, seg);
    rtsys->error = true;
    return 0.0;
  }

  if (mxGetClassID(lhs[1]) == mxUNKNOWN_CLASS) {
    printf("??? executeCode: data not assigned\n\n");
    printf("Error in ==> code function '%s', segment %d\n", codeName, seg);
    rtsys->error = true;
    return 0.0;
  }

  if (dataName) {
    mexPutVariable("global", dataName, lhs[1]);
  }

  retval = *mxGetPr(lhs[0]);

  mxDestroyArray(rhs[1]);
  mxDestroyArray(lhs[0]);
  mxDestroyArray(lhs[1]);

  return retval;
}
#endif
======================= */

// ----------------------------------------------
// -- Determines time for next clock interrupt --
// ----- used in the kernel function below ------
// ----------------------------------------------

double getNextInvocation() {

  double compTime;
  double nextHit = INF;

  // Next release from timeQ
  if (rtsys->timeQ->getFirst() != NULL) {
    Task* t = (Task*) rtsys->timeQ->getFirst();
    nextHit = t->wakeupTime() - rtsys->time;
  }

  // Remaining execution time of running task
  if (rtsys->running != NULL) {
    compTime = rtsys->running->execTime / rtsys->cpuScaling;
    nextHit = (nextHit < compTime) ? nextHit : compTime;
  }

  return nextHit;
}


// -------------------------------------------------------------
// --------------------- Kernel Function -----------------------
// ----  Called from the Simulink callback functions during ----
// -- simulation and returns the time for its next invocation --
// -------------------------------------------------------------

double runKernel(double externalTime) {

  double nextHit, timeElapsed;

  Task *task, *temp, *oldrunning, *newrunning;
  UserTask *usertask;
  InterruptHandler *hdl;
  DataNode* dn;

  // If no energy, then we can not run
  if (rtsys->energyLevel <= 0) {
    printf("Energy is out at time: %f\n", rtsys->time);
    return INF;
  }

  timeElapsed = externalTime - rtsys->prevHit; // time since last invocation
  rtsys->prevHit = externalTime;  // update previous invocation time
  nextHit = 0.0;

  //printf("runkernel at %f\n", rtsys->time);

#ifdef KERNEL_MATLAB
    /* Write rtsys pointer to global workspace */
  *((int *)mxGetPr(rtsys->rtsysptr)) = (int)rtsys;
#endif

  while (nextHit < EPS) {

    // Count down execution time for current task (usertask or handler)
    // and check if it has finished its execution

    task = rtsys->running;
    if (task != NULL) {
      // Count down execution time
      task->execTime -= timeElapsed * rtsys->cpuScaling;
      if (task->execTime < EPS) {
        // Execute next segment
        task->segment++;

        if (task->isUserTask()) {
          usertask = (UserTask*) task;
          // Update budget and lastStart variable at segment change
          usertask->budget -= (rtsys->time - usertask->lastStart);
          usertask->lastStart = rtsys->time;
        }

        // Execute next segment of the code function

#ifndef KERNEL_MATLAB
        task->execTime = task->codeFcn(task->segment, task->data);
        if (rtsys->error) {
          printf("Error in ==> task '%s', segment %d\n", task->name, task->segment);
          return 0.0;
        }
#else
        if (task->codeFcnMATLAB == NULL) {
          task->execTime = task->codeFcn(task->segment, task->data);
        } else {
          task->execTime = executeCode(task->codeFcnMATLAB, task->segment, task->dataMATLAB);
        }
        if (rtsys->error) {
          printf("Error in ==> task '%s', segment %d\n", task->name, task->segment);
          return 0.0;
        }
#endif

        if (task->execTime < 0.0) {
          // Negative execution time = task finished
          task->execTime = 0.0;
          task->segment = 0;

          if (task->myList == rtsys->readyQ) {
            // Remove task from readyQ
            task->remove();
          }

          if (!(task->isUserTask())) {
            hdl = (InterruptHandler*) task;

            if (hdl->type == TIMER) {
              if (hdl->timer->isPeriodic) {
                    // if periodic timer put back in timeQ
                    hdl->timer->time += hdl->timer->period;
                    hdl->moveToList(rtsys->timeQ);
              } else {
                    // Remove timer and free up handler
                    dn = getNode(hdl->timer->name, rtsys->timerList);
                    rtsys->timerList->deleteNode(dn);
                    delete hdl->timer;
                    hdl->timer = NULL;
```

```
                hdl->type = UNUSED;
          }
        }
        if (hdl->type == EXTERNAL) {
          if (hdl->pending > 0) {
                // new external interrupt occured before handler finished
                hdl->pending--;
                hdl->moveToList(rtsys->readyQ);
          }
        }

      } else { // the finished task is a usertask
        usertask = (UserTask*) task;

        // Execute finish-hook
        usertask->finish_hook(usertask);
        usertask->state = IDLE;

        // Release next job if any
        usertask->nbrJobs--;
        if (usertask->nbrJobs > 0) {
          // next pending release
          dn = (DataNode*) usertask->pending->getFirst();
          double* release = (double*) dn->data;
          usertask->release = *release;
          usertask->absDeadline = *release + usertask->deadline;
          usertask->moveToList(rtsys->timeQ);
          usertask->pending->deleteNode(dn);
          delete release;
          // Execute release-hook
          usertask->release_hook(usertask);
          usertask->state = SLEEPING;

        }
      }
    }
  }
} // end: counting down execution time of running task


// Check time queue for possible releases

task = (Task*) rtsys->timeQ->getFirst();
while (task != NULL) {
  if ((task->wakeupTime() - rtsys->time) < EPS) {

    // Task to be released
    temp = task;
    task = (Task*) task->getNext();
    temp->moveToList(rtsys->readyQ);

    if (temp->isUserTask()) {
      usertask = (UserTask*) temp;
      usertask->state = READY;

    }
  } else {
    break;
  }
} // end: checking timeQ for releases


// Determine task with highest priority and make it running task

newrunning = (Task*) rtsys->readyQ->getFirst();
oldrunning = rtsys->running;

if (newrunning != NULL) {

  // Check for suspend- and resume-hooks

  if (oldrunning != NULL) {

    // Is oldrunning being suspended?
    if (oldrunning->isUserTask()) {
      if (newrunning != oldrunning && ((UserTask*) oldrunning)->state == RUNNING) {
        usertask = (UserTask*) oldrunning;
```

```
            usertask->state = SUSPENDED;
            usertask->suspend_hook(usertask);
          }
        }
      }

      // invocation of hooks may have triggered kernelHandler
      newrunning = (Task*) rtsys->readyQ->getFirst();

      // Is newrunning being resumed?
      if (newrunning->isUserTask()) {
        if ( (((UserTask*) newrunning)->state == READY) ||
             (((UserTask*) newrunning)->state == SUSPENDED) ) {
          // newrunning is being resumed or started
          usertask = (UserTask*) newrunning;
          usertask->state = RUNNING;
          if (usertask->segment == 0) {
            usertask->start_hook(usertask);
          } else {
            usertask->resume_hook(usertask);
          }
        }
      }

      // invocation of hooks may have triggered kernelHandler
      rtsys->running = (Task*) rtsys->readyQ->getFirst();

    } else { // No tasks in readyQ

      rtsys->running = NULL;

    } // end: task dispatching


    // Determine next invocation of kernel
    nextHit = getNextInvocation();
    timeElapsed = 0.0;

  } // end: loop while nextHit < EPS

  return nextHit;
}

// ------- Scicos callback functions -------

void truetimekernel(scicos_block *block, int flag) {

  static int printed;  // original printed = 0 and located before if (!printed)
  // double externTime;  // original under mdlOutputs double externTime = ...
  int i, j, k, detected; //original i under << mdlInitializeConditions >>
  double dTime;            // original others under output calculation
  DataNode *dn;
  Task* task;
  UserTask* t;
  InterruptHandler* hdl;
  Monitor *mon;

  // printf("%d at %f\n", flag, get_scicos_time()); // only for debugging

  switch (flag) {

  case 4:     // ***** Initalialization *****

    /* << mdlInitializeSizes >> */
    printed = 0;
    if (!printed) {
             printed = 1;
             printf("-------------------------------------------------------\n");
             printf("                 TrueTime, Version 1.4\n");
             printf("                   Copyright (c) 2006\n");
             printf("     Martin Ohlin, Dan Henriksson and Anton Cervin\n");
             printf("          Department of Automatic Control, LTH\n");
             printf("                Lund University, Sweden\n");
             printf("-------------------------------------------------------\n");
    }

    rtsys = new RTsys;
    /* Assign function pointers */
    rtsys->contextSwitchCode = contextSwitchCode;
    rtsys->periodicTaskHandlerCode = periodicTaskHandlerCode;

    rtsys->timeSort = timeSort;
    rtsys->prioSort = prioSort;

    rtsys->default_arrival = default_arrival;
    rtsys->default_release = default_release;
    rtsys->default_start = default_start;
    rtsys->default_suspend = default_suspend;
    rtsys->default_resume = default_resume;
    rtsys->default_finish = default_finish;

    rtsys->prioFP = prioFP;
    rtsys->prioRM = prioRM;
    rtsys->prioEDF = prioEDF;
    rtsys->prioDM = prioDM;

    rtsys->nbrOfTasks = 0;

    /* Save pointer to init args */
    // rtsys->initarg = (mxArray *)ssGetSFcnParam(S, 1);
    /* Evaluating user-defined init function (C++) */
    init();
    rtsys->init_phase = false;

    if (!rtsys->initialized) {
              printf("ttInitKernel was not called in init function!\n");
              set_block_error(-1);
              return;
    }

    // Clock drift parameters
    rtsys->clockDrift = 1.0; // originally = *mxGetPr(arg) + 1;
    rtsys->clockOffset = 0.0; // originally = *mxGetPr(arg);
    // printf("drift: %f, offset:%f\n", rtsys->clockDrift, rtsys->clockOffset);

    /* Input Ports */

    printf("Number of inputs: %d\n", block->nin);
    if (block->nin==4) {
      printf("Inputs=%d Triggers=%d Networks=%d EnergyLevel=%d\n", rtsys->nbrOfInputs, rtsys-
>nbrOfTriggers, rtsys->nbrOfNetworks, rtsys->energyLevel);
      if (rtsys->nbrOfInputs > 0) {
        // ssSetInputPortWidth(S, 0, rtsys->nbrOfInputs);
        if (block->insz[0] != rtsys->nbrOfInputs) {
          printf("Input[0] size must equal to %d\n", rtsys->nbrOfInputs);
          rtsys->error = true;
          set_block_error(-1);
          return;
      }}
      else {
        // ssSetInputPortWidth(S, 0, 1);
        if (block->insz[0] != 1) {
          printf("Input[0] size must be 1!\n");
          set_block_error(-1);
          return;
      }}

      if (rtsys->nbrOfTriggers > 0) {
        // ssSetInputPortWidth(S, 1, rtsys->nbrOfTriggers);
        if (block->insz[1] != rtsys->nbrOfTriggers) {
          printf("Input[1] size must equal to %d\n", rtsys->nbrOfTriggers);
          set_block_error(-1);
          return;
      }}
      else {
        // ssSetInputPortWidth(S, 1, 1);
        if (block->insz[1] != 1) {
          printf("Input[1] size must be 1!\n");
          set_block_error(-1);
          return;
      }}

      if (rtsys->nbrOfNetworks > 0) {
        // ssSetInputPortWidth(S, 2, rtsys->nbrOfNetworks); // Network received
        if (block->insz[2] != rtsys->nbrOfNetworks) {
          printf("Input[2] size must equal to %d\n", rtsys->nbrOfNetworks);
          set_block_error(-1);
          return;
      }}
      else {
        // ssSetInputPortWidth(S, 2, 1);
        if (block->insz[2] != 1) {
          printf("Input[2] size must be 1!\n");
          set_block_error(-1);
          return;
      }}

      // ssSetInputPortWidth(S, 3, 1); //battery
      if (block->insz[3] != 1) {
        printf("Input[3] size must be 1!\n");
        set_block_error(-1);
        return;
      }
    }
    else {
      printf("ttInitKernel input number must be 4!\n");
      set_block_error(-1);
      return;
    }

    /* Output Ports */

    printf("Number of outputs: %d\n", block->nout);
    if (block->nout==5) {
      printf("Outputs=%d Handlers=%d+%d=%d Monitors=%d*%d=%d EnergyConsumption=%d\n",
rtsys->nbrOfOutputs, rtsys->nbrOfSchedTasks, rtsys->nbrOfSchedHandlers, rtsys-
>nbrOfSchedTasks+rtsys->nbrOfSchedHandlers, rtsys->nbrOfSchedMonitors, rtsys->nbrOfTasks,
rtsys->nbrOfSchedMonitors*rtsys->nbrOfTasks, rtsys->energyConsumption);
    if (rtsys->nbrOfOutputs > 0) {
      // ssSetOutputPortWidth(S, 0, rtsys->nbrOfOutputs);
      if (block->outsz[0] != rtsys->nbrOfOutputs) {
        printf("Output[0] size must equal to %d\n", rtsys->nbrOfOutputs);
        set_block_error(-1);
        return;
    }}
    else {
      // ssSetOutputPortWidth(S, 0, 1);
      if (block->outsz[0] != 1) {
        printf("Output[0] size must be 1!\n");
        set_block_error(-1);
        return;
    }}
    if (rtsys->nbrOfNetworks > 0) {
      // ssSetOutputPortWidth(S, 1, (rtsys->nbrOfNetworks)); // Network send
      if (block->outsz[1] != rtsys->nbrOfNetworks) {
        printf("Output[1] size must equal to %d\n", rtsys->nbrOfNetworks);
        set_block_error(-1);
        return;
    }}
    else {
      // ssSetOutputPortWidth(S, 1, 1);
      if (block->outsz[1] != 1) {
        printf("Output[1] size must be 1!\n");
        set_block_error(-1);
        return;
    }}

    if (rtsys->nbrOfSchedTasks+rtsys->nbrOfSchedHandlers > 0) {
      // ssSetOutputPortWidth(S, 2, rtsys->nbrOfSchedTasks+rtsys->nbrOfSchedHandlers);
      if (block->outsz[2] != rtsys->nbrOfSchedTasks+rtsys->nbrOfSchedHandlers) {
        printf("Output[2] size must equal to %d\n", rtsys->nbrOfSchedTasks+rtsys-
>nbrOfSchedHandlers);
        set_block_error(-1);
        return;
    }}
    else {
      // ssSetOutputPortWidth(S, 2, 1);
      if (block->outsz[2] != 1) {
        printf("Output[2] size must be 1!\n");
        set_block_error(-1);
        return;
    }}

    if (rtsys->nbrOfSchedMonitors > 0) {
      // ssSetOutputPortWidth(S, 3, rtsys->nbrOfSchedMonitors*rtsys->nbrOfTasks);
      if (block->outsz[3] != rtsys->nbrOfSchedMonitors*rtsys->nbrOfTasks) {
        printf("Output[3] size must equal to %d\n", rtsys->nbrOfSchedMonitors*rtsys-
>nbrOfTasks);
        set_block_error(-1);
        return;
    }}
    else {
      // ssSetOutputPortWidth(S, 3, 1);
      if (block->outsz[3] != 1) {
        printf("Output[3] size must be 1!\n");
        set_block_error(-1);
        return;
    }}

    //ssSetOutputPortWidth(S, 4, 1); //Energy consumption
    if (block->outsz[4] != 1) {
      printf("Output[4] size must be 1!\n");
      set_block_error(-1);
      return;
    }
  }
  else {
    printf("ttInitKernel output number must be 5!\n");
    set_block_error(-1);
    return;
  }

  // ssSetNumContStates(S, 0);
  if (block->nx != 0) {
            printf("Number of Continuous States must be 0!\n");
            set_block_error(-1);
            return;
   }

   // ssSetNumDiscStates(S, 0);
   if (block->nz != 0) {
            printf("Number of Discrete States must be 0!\n");
            set_block_error(-1);
            return;
   }

   // ?? ssSetNumSampleTimes(S, 1);

   // ssSetNumRWork(S, 0);
   if (block->nrpar != 0) {
            printf("Number of Real Parameters must be 0!\n");
            set_block_error(-1);
            return;
   }

   block->nipar = 0; // equal to ssSetNumIWork(S, 0);
   if (block->nipar != 0) {
            printf("Number of Integer Parameters must be 0!\n");
            set_block_error(-1);
            return;
   }

   // ?? ssSetNumPWork(S, 0);

   block->nmode = 0; // equal to ssSetNumModes(S, 0);
   if (block->nmode != 0) {
            printf("Number of Modes must be 0!\n");
            set_block_error(-1);
            return;
   }

   block->ng = 1; // equal to ssSetNumNonsampledZCs(S, 1);
   if (block->ng != 1) {
            printf("Number of Zero Crossings must be 1!\n");
            set_block_error(-1);
            return;
   }
   *block->work = rtsys; // equal to ssSetUserData(S, rtsys);

   // ?? ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE | SS_OPTION_CALL_TERMINATE_ON_EXIT);

   /* << mdlInitializeSampleTimes >> */
   // ?? ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
   // ?? ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);

   /* << mdlStart >> */

   // not necessary - rtsys = (RTsys*) ssGetUserData(S);

   if (rtsys->init_phase) {
             /* Failure during initialization */
             return;
   }

   /* DATA ALLOCATION */

   if (rtsys->nbrOfTriggers > 0) {
             rtsys->interruptinputs = new double[rtsys->nbrOfTriggers];
             rtsys->oldinterruptinputs = new double[rtsys->nbrOfTriggers];
   }

   if (rtsys->nbrOfTasks > 0) {
             rtsys->taskSched = new double[rtsys->nbrOfTasks];
   }

   if (rtsys->nbrOfHandlers > 0) {
             rtsys->handlerSched = new double[rtsys->nbrOfHandlers];
   }

   if (rtsys->nbrOfMonitors > 0) {
             rtsys->monitorGraph = new double[rtsys->nbrOfTasks];
   }
   if (rtsys->nbrOfNetworks > 0) {
             rtsys->nwSnd = new double[rtsys->nbrOfNetworks];
             rtsys->networkinputs = new double[rtsys->nbrOfNetworks];
             rtsys->oldnetworkinputs = new double[rtsys->nbrOfNetworks];
   }

   /* << mdlInitializeConditions >> */

   //printf("mdlInit\n");
   //int i; collected in the beginning of scicos block
   // repeated - rtsys = (RTsys*) ssGetUserData(S);

   if (rtsys->init_phase) {
             /* Failure during initialization */
             return;
   }

   for (i=0; i<rtsys->nbrOfInputs; i++)
             rtsys->inputs[i] = block->inptr[0][i]; // originally =
*ssGetInputPortRealSignalPtrs(S,0)[i];

   for (i=0; i<rtsys->nbrOfTriggers; i++) {
             rtsys->interruptinputs[i] = 0.0;
             rtsys->oldinterruptinputs[i] = 0.0;
   }

   /* NETWORK */
   for (i=0; i<rtsys->nbrOfNetworks; i++) {
             rtsys->nwSnd[i] = 0.0;
             rtsys->networkinputs[i] = 0.0;
             rtsys->oldnetworkinputs[i] = 0.0;
   }
   if (rtsys->nbrOfNetworks > 0) {
             //temp=ttInitNetwork2(); /* do the rest of the network initialization */
   }
   break;


  case 1:     // ***** Output Calculation *****

    if (get_phase_simulation()==1) {
      rtsys = (RTsys*) *block->work;

      if (rtsys->init_phase) {
        /* Failure during initialization */
        return;
      }

      // directly connect to block->outptr[0] real_T *y = ssGetOutputPortRealSignal(S,0);
      // directly connect to block->outptr[1] real_T *n = ssGetOutputPortRealSignal(S,1);
      // directly connect to block->outptr[2] real_T *s = ssGetOutputPortRealSignal(S,2);
      // directly connect to block->outptr[3] real_T *m = ssGetOutputPortRealSignal(S,3);
      // directly connect to block->outptr[4] real_T *energyConsumption =
ssGetOutputPortRealSignal(S,4);
      /* ====================== collected in the beginning of scicos block
      int i, j, k, detected;
      double dTime;

      DataNode *dn;
      Task* task;
      UserTask* t;
      InterruptHandler* hdl;
      Monitor *mon;
      ====================== */

      //rtsys = (RTsys*) *block->work; // equal to rtsys = (RTsys*) ssGetUserData(S);

      // if (rtsys->init_phase) { // duplicated code
         /* Failure during initialization */
      // return;
      // }


      if (!rtsys->started && get_scicos_time() == 0.0) {
        rtsys->started = true;
        return;
      }

      /*if (!rtsys->mdlzerocalled) {
        printf("Zero crossing detection must be turned on in order to run TrueTime!\n");
        return;
      }
      */

      /* Storing the time */
      /* ========= Contents in between are moved to case 9 ========== */

      detected = 0;

      /* Check interrupts */

      i = 0;
      dn = (DataNode*) rtsys->triggerList->getFirst();
      while (dn != NULL) {
        if (fabs(rtsys->interruptinputs[i]-rtsys->oldinterruptinputs[i]) > 0.1) {
          hdl = (InterruptHandler*) dn->data;
          Trigger* trig = hdl->trigger;
          if (rtsys->time - trig->prevHit > trig->latency) {
            // Trigger interrupt handler
            if (hdl->myList == rtsys->readyQ) {
              // handler serving older interrupts
              hdl->pending++;
            } else {
              hdl->moveToList(rtsys->readyQ);
              detected = 1;
            }
            trig->prevHit = rtsys->time;
          } else {
              //printf("Call to interrupt handler %s ignored at time %f. Within interrupt
latency!\n", hdl->name, rtsys->time);
          }
          rtsys->oldinterruptinputs[i] = rtsys->interruptinputs[i];
        }
        i++;
        dn = (DataNode*) dn->getNext();
      }
      /* Check network */

      dn = (DataNode*) rtsys->networkList->getFirst();
      while (dn != NULL) {
        hdl = (InterruptHandler*) dn->data;
       /* temp ===============
        Network* network = hdl->network;
        i = network->networkID - 1;
        //printf("mdlOutputs: checking network #%d inp: %d oldinp: %d\n",i,rtsys-
>networkinputs[i],rtsys->oldnetworkinputs[i]);
        if (fabs(rtsys->networkinputs[i] - rtsys->oldnetworkinputs[i]) > 0.1) {
          hdl->moveToList(rtsys->readyQ);
          detected = 1;
          rtsys->oldnetworkinputs[i] = rtsys->networkinputs[i];
        }
        ==================== */
        dn = (DataNode*) dn->getNext();
      }

      /* Run kernel? */
      /* ========= Contents in between are moved to case 9 ========== */

      /* Outputs */

      for (i=0; i<rtsys->nbrOfOutputs; i++) {
        block->outptr[0][i] = rtsys->outputs[i];
      }


      /* Network send */

      for (i=0; i<rtsys->nbrOfNetworks; i++) {
        block->outptr[1][i] = rtsys->nwSnd[i];
      }

      /* Task schedule */

      i = 0;
      j = 0;
      dn = (DataNode*) rtsys->taskList->getFirst();
      while (dn != NULL) {
        t = (UserTask*) dn->data;
        rtsys->taskSched[i] = (double) (j+1);
        if (t->display) j++;
        dn = (DataNode*) dn->getNext();
        i++;
      }

      task = (Task*) rtsys->readyQ->getFirst();
      while (task != NULL) {
        if (task->isUserTask()) {
          t = (UserTask*) task;
          rtsys->taskSched[t->taskID - 1] += 0.25;
        }
        task = (Task*) task->getNext();
      }

      if ((rtsys->running != NULL) && (rtsys->running->isUserTask())) {
        t = (UserTask*) rtsys->running;
        rtsys->taskSched[t->taskID - 1] += 0.25;
      }

      i = 0;
      j = 0;
      dn = (DataNode*) rtsys->taskList->getFirst();
      while (dn != NULL) {
        t = (UserTask*) dn->data;
        if (t->display) {
          block->outptr[2][j] = rtsys->taskSched[i];
          j++;
        }
        dn = (DataNode*) dn->getNext();
        i++;
      }


      /* Handler schedule */
```

```
i = 0;
j = 0;
dn = (DataNode*) rtsys->handlerList->getFirst();
while (dn != NULL) {
  rtsys->handlerSched[i] = (double) (j+rtsys->nbrOfSchedTasks+2);
  if (i==0 && rtsys->contextSwitchTime > EPS) {
    // Context switch schedule, move graph down to task level
    rtsys->handlerSched[i] = rtsys->handlerSched[i] - 1;
  }
  hdl = (InterruptHandler*) dn->data;
  if (hdl->display) j++;
  dn = (DataNode*) dn->getNext();
  i++;
}

task = (Task*) rtsys->readyQ->getFirst();
while (task != NULL) {
  if (!(task->isUserTask())) {
    hdl = (InterruptHandler*) task;
    rtsys->handlerSched[hdl->handlerID - 1] += 0.25;
  }
  task = (Task*) task->getNext();
}

if ((rtsys->running != NULL) && (!(rtsys->running->isUserTask()))) {
  hdl = (InterruptHandler*) rtsys->running;
  rtsys->handlerSched[hdl->handlerID - 1] += 0.25;
}

i = 0;
j = 0;
dn = (DataNode*) rtsys->handlerList->getFirst();
while (dn != NULL) {
  hdl = (InterruptHandler*) dn->data;
  if (hdl->display) {
    block->outptr[2][j+rtsys->nbrOfSchedTasks] = rtsys->handlerSched[i];
    j++;
  }
  dn = (DataNode*) dn->getNext();
  i++;
}

/* Monitor graph */

k = 0;
dn = (DataNode*) rtsys->monitorList->getFirst();
while (dn != NULL) {
  mon = (Monitor*) dn->data;

  for (j=0; j<rtsys->nbrOfTasks; j++)
  rtsys->monitorGraph[j] = (double) (j+1+k*(1+rtsys->nbrOfTasks));

  t = (UserTask*) mon->waitingQ->getFirst();
  while (t != NULL) {
    i = t->taskID;
    rtsys->monitorGraph[i-1] += 0.25;
    t = (UserTask*) t->getNext();
  }
  if (mon->heldBy != NULL) {
    i = mon->heldBy->taskID;
    rtsys->monitorGraph[i-1] += 0.5;
  }
  if (mon->display) {
    for (j=0; j<rtsys->nbrOfTasks; j++)
    block->outptr[3][j+k*rtsys->nbrOfTasks] = rtsys->monitorGraph[j];
                                k++;
  }
  dn = (DataNode*) dn->getNext();
}

/* Energy consumption */
block->outptr[4][0] = rtsys->energyConsumption;

/* Copy analog inputs */
for (i=0; i<rtsys->nbrOfInputs; i++) {
        rtsys->inputs[i] = block->inptr[0][i]; //
original=*ssGetInputPortRealSignalPtrs(S,0)[i];
      }

      /* Copy interrupt inputs, check for events */
      for (i=0; i<rtsys->nbrOfTriggers; i++) {
        if (fabs(block->inptr[1][i]-rtsys->interruptinputs[i]) > 0.1) {
          if (get_scicos_time() < rtsys->nextHit) {
            rtsys->nextHit = get_scicos_time();
            printf("Next hit at copy interrupt %f\n", rtsys->nextHit);

          }
          //printf("mdlZeroCrossings: interrupt detected at %2.20g\n", get_scicos_time());
        }
        rtsys->interruptinputs[i] = block->inptr[1][i]; //
original=*ssGetInputPortRealSignalPtrs(S,1)[i];
      }

      /* Copy network input, check for event */
      for (i=0; i<rtsys->nbrOfNetworks; i++) {
        if (fabs(block->inptr[2][i]-rtsys->networkinputs[i]) > 0.1) {
          if (get_scicos_time() < rtsys->nextHit) {
            rtsys->nextHit = get_scicos_time();
            printf("Next hit at copy network input %f\n", rtsys->nextHit);
          }
        }
        rtsys->networkinputs[i] = block->inptr[2][i]; //
original=*ssGetInputPortRealSignalPtrs(S,2)[i];
      }

      /* Check the energy level */
      rtsys->energyLevel = block->inptr[3][0]; //
original=*ssGetInputPortRealSignalPtrs(S,3)[0];

    }

    break;

  case 9:     // ***** Calculate zero crossing surface *****

    rtsys = (RTsys*) *block->work;

    /* Updating zero-crossing surface */

    block->g[0] = rtsys->nextHit - get_scicos_time();
    // equal to ssGetNonsampledZCs(S)[0] = rtsys->nextHit - ssGetT(S);

    /* Storing the time */

    rtsys->time = get_scicos_time() * rtsys->clockDrift + rtsys->clockOffset;

    /* Run kernel? */

    // externTime =  (rtsys->time- rtsys->clockOffset) / rtsys->clockDrift; //originally
declared here
    if ((get_scicos_time() >= rtsys->nextHit) || (detected > 0)) { // replaced externTime with
get_scicos_time()
        if (!rtsys->mdlzerocalled) {
              rtsys->mdlzerocalled = true;
        }
        dTime = runKernel(get_scicos_time());
        if (rtsys->error) {
              // Something went wrong executing a code function
              /* temp ===============
              mxArray *bn[1];
              mexCallMATLAB(1, bn, 0, 0, "gcs"); // get current system
              char buf[200];
              mxGetString(bn[0], buf, 200);
              for (unsigned int i=0; i<strlen(buf); i++) if (buf[i]=='\n') buf[i]=' ';
              ==================== */
              printf("In block ==> '%s'\nSimulation aborted!\n");
              //ssSetStopRequested(S, 1);
        } else {
              rtsys->nextHit = (rtsys->time + dTime - rtsys->clockOffset) / rtsys->clockDrift;
              printf("Next hit after run kernel: %f\n", rtsys->nextHit);
        }
    }
    break;

  case 5:     // ***** Termination *****

    /* << mdlTerminate >> */

    rtsys = (RTsys*) *block->work; // equal to rtsys = ssGetUserData(S);

    if (rtsys == NULL) {
        return;
    }

    if (rtsys->taskList != NULL) {
        // write logs to the MATLAB workspace
        DataNode *dn = (DataNode*) rtsys->taskList->getFirst();
        while (dn != NULL) {
            UserTask *task = (UserTask*) dn->data;
            for (int j=0; j<NBRLOGS; j++) {
                Log *log = task->logs[j];
                if (log) {
                    // printf("Dumping log %d for task %s\n", j, task->name);
                    /* temp ===============
                    mxArray *ptr = mxCreateDoubleMatrix(log->entries, 1, mxREAL);
                    for (int n=0; n<log->entries; n++) {
                        mxGetPr(ptr)[n] = log->vals[n];
                    }
                    mexMakeArrayPersistent(ptr);
                    mexPutVariable("base",log->variable,ptr);
                    ==================== */
                }
            }
            dn = (DataNode*) dn->getNext();
        }
    }

    // Cleanup
    delete *block->work;
    cleanup();

    // Delete rtsys and all data structures within
    delete rtsys;
  }
}

/* >>>>>>>>>>>>>>>>>>>>> Write the applications below this line <<<<<<<<<<<<<<<<<<<<< */

// Task scheduling and control.
//
// This example extends the simple PID control example (located in
// $DIR/examples/simple_pid) to the case of three PID-tasks running
// concurrently on the same CPU controlling three different servo
// systems. The effect of the scheduling policy on the global control
// performance is demonstrated.

// PID data structure
class PID_Data {
public:
  struct { // states
    double u, Iold, Dold, yold;
  } s;

  struct { // params
    double K, Ti, Td, N, h;
    int rChan, yChan, uChan;
  } p;
};

// calculate PID control signal and update states
void pidcalc(PID_Data* d, double r, double y) {
  double P = d->p.K*(r-y);
  double I = d->s.Iold;
  double D = d->p.Td/(d->p.N*d->p.h+d->p.Td)*d->s.Dold+d->p.N*d->p.K*d->p.Td/(d->p.N*d->p.h+d-
>p.Td)*(d->s.yold-y);

  d->s.u = P + I + D;
  d->s.Iold = d->s.Iold + d->p.K*d->p.h/d->p.Ti*(r-y);
  d->s.Dold = D;
  d->s.yold = y;
};


// --------- Generic code function ----------
double pidcode(int seg, void* data) {

  PID_Data* d = (PID_Data*) data;

  switch (seg) {
  case 1:
    pidcalc(d, ttAnalogIn(d->p.rChan), ttAnalogIn(d->p.yChan));
    return 0.001;
  case 2:
    ttAnalogOut(d->p.uChan, d->s.u);
    return FINISHED;
  }

  return FINISHED; // to supress compilation warnings
}

#define NBROFINPUTS 6
#define NBROFOUTPUTS 3
#define SCHEDULER prioRM

// Task parameters
double periods[] = {0.006, 0.005, 0.004};
char* names[] = {"pid_task1", "pid_task2", "pid_task3"};
int rChans[] = {1, 3, 5};
int yChans[] = {2, 4, 6};
int uChans[] = {1, 2, 3};

PID_Data *d[3];

void init() {

  // Initialize TrueTime kernel
  ttInitKernel(NBROFINPUTS, NBROFOUTPUTS, SCHEDULER);

  // Create the three tasks
  for (int i = 0; i < 3; i++) {
    d[i] = new PID_Data;
    d[i]->p.K = 0.96;
    d[i]->p.Ti = 0.12;
    d[i]->p.Td = 0.049;
    d[i]->p.N = 10;
    d[i]->p.h = periods[i];
    d[i]->s.u = 0.0;
    d[i]->s.Iold = 0.0;
    d[i]->s.Dold = 0.0;
    d[i]->s.yold = 0.0;
    d[i]->p.rChan = rChans[i];
    d[i]->p.yChan = yChans[i];
    d[i]->p.uChan = uChans[i];

    // Offset=0 and prio=1 for all tasks
    ttCreatePeriodicTask(names[i], 0.0, periods[i], 1.0, pidcode, d[i]);
  }
}

void cleanup() {

  for (int i = 0; i < 3; i++) {
    delete d[i];
  }
}
```

6.3 Appendix C: Code listing of truetimekernel.sci

```
function [x,y,typ]=truetimekernel(job,arg1,arg2)
//
  x=[];y=[];typ=[]
  select job
    case 'plot' then
      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;
      graphics=arg1.graphics;exprs=graphics.exprs;
      model=arg1.model;
      while %t do
        [ok,junction_name,ni,nt,nn,no,ns,nm,clkd,clko,exprs]=..
          getvalue('Set TRUETIMEKERNEL block parameters',..
            ['Simulation Function';
             'Number Of Inputs';
             'Number Of Triggers';
             'Number Of Networks';
             'Number Of Outputs';
             'Number Of Schedulers';
             'Number Of Monitors';
             'CLOCK DRIFT';
             'CLOCK OFFSET'],..

list('str',1,'vec',1,'vec',1,'vec',1,'vec',1,'vec',1,'vec',1,'vec',1,'vec',1),exprs)
        if ~ok then break,end
        junction_name=stripblanks(junction_name)
        if ok then
          graphics.exprs=exprs
          model.sim=list(junction_name,4)
          model.in=[ni;nt;nn;1]
          model.out=[no;nn;ns;nm;1]
          model.rpar=[clkd;clko]
          x.model=model
          x.graphics=graphics
          break
        end
      end
    case 'define' then
      model=scicos_model()
      junction_name='truetimekernel';
      //ni=1,nt=1,nn=1,no=1,ns=1,nm=1;
      model.sim=list(junction_name,4)
      model.in=[1;1;1;1]
      model.out=[1;1;1;1;1]
      model.evtin=[]
      model.evtout=[]
      model.state=[]
      model.dstate=[]
      model.rpar=[0.0;0.0]
      model.ipar=[]
      model.nmode=0
      model.nzcross=1
      model.blocktype='c'
      model.firing=[]
      model.dep_ut=[%f %t] //no direct feedthrough, time dependence
      exprs=[junction_name;sci2exp(1);sci2exp(1);sci2exp(1);
                           sci2exp(1);sci2exp(1);sci2exp(1);
                           sci2exp(0.0);sci2exp(0.0)]
      gr_i=['txt=['' TrueTime '';'' Kernel ''];';
                      'xstringb(orig(1),orig(2),txt,sz(1),sz(2),''fill'')']
      x=standard_define([4 3],model,exprs,gr_i)
    end
endfunction
```