Introducing the C-API Simulink Target for RT-Linux

Arthur Siro
Universidad de Navarra, Escuela Superior de Ingenieros, TECNUN.
Paseo de Manuel Lardizabal 13, 20018, Donostia-San Sebastian, SPAIN
asiro@ceit.es

Iñaki Diaz
Centro de Estudios e Investigaciones Tecnicas de Gipuzkoa, CEIT.
Paseo de Manuel Lardizabal 15, 20009, SPAIN
idiaz@ceit.es

Abstract

The 'C-API Simulink Target for RT-Linux' (CAPI_STRTL) is an application developed to extend the functionality of the original 'Simulink target for RT-Linux' (STRTL) by providing the user with full parameter tuning and signal access, of a real time application running in a RT-Linux kernel, independent of the Simulink/MATLAB interface.

In this manner, CAPI_STRTL extends the utility of the original STRTL from its conventional use in control applications to more exotic fields such as haptics/virtual reality and even augmented reality - not to mention the many other possible implementations users can realize by having signal access and parameter tuning at the disposal of their custom applications.

The CAPI_STRTL is an add-on feature to the STRTL and is not intended to replace it in anyway.

The purpose of this paper is to introduce the CAPI_STRTL and demonstrate how its functionality can extend the use of STRTL. This paper assumes some previous knowledge of the workings of STRTL.

Keywords: RT-Linux, Simulink, Real-Time Workshop

1 Introduction

Rapid prototyping tools e.g. Simulink Target for RT-Linux (STRTL)[1] and RTAI-Lab[2], used for the development and testing of real-time applications executing on RT Linux variants[3] have been around for the last few years. These utilities in turn harness code generating and signal monitoring tools such as Real Time Workshop and Simulink/MATLAB[4], and Scilab/Scicos[5].

A user - via a few mouse clicks on a modelling application's GUI[1] e.g. Simulink - generates code that will execute on a RT Linux variant's kernel. All that is left is to be done is compilation and execution in the machine running the RT Linux variant and viola! The user can then monitor signals and adjust parameters via the interface provided to them by Simulink or RTAI-Lab. This is by no means a trivial achievement on the part of the designers but imposes some limitations on the part of the user.

For instance, users might require that the signals obtained 'on the fly' during program execution be channelled to a custom application instead of the interface provided by, say, Simulink. Now, though it would be a worthy undertaking to try and list the number of possible things users are capable of doing if they had these signals at their complete disposal, some 'immediate' areas of application can be in haptics and even robotic tele-oparation applications in augmented reality.

In other words, although most of these Rapid Prototyping tools are incredibly wonderful utilities, they generally target conventional control and robotic applications. It is this restriction that CAPI_STRTL attempts to purge.

[1] Graphical User Interface

2 Software Description

STRTL is the outcome of a PhD research project carried out at the Caledonian University, School of Engineering, Design and Science by a certain Dr. Raul Garcia Murillo[1]. STRTL is one of the few widely tested and proven alternatives to the expensive commercial software required for developing and testing prototype hard real-time control applications. It is a modification of the RTW target Generic Real-Time Target used for UNIX based computers[6]. STRTL offers the capability of remote monitoring and control using a client-server model. In Simulink/RTW jargon, a client/host is a computer running the Simulink process while the machine running the real-time application is the server/target. Simulink implements host/target communication via the TCP/IP protocol.

Now, CAPI_STRTL is a new software feature that is partially based on the original STRTL framework. It differs with respect to STRTL mainly in the manner signals and parameters of the real-time application are handled. CAPI_STRTL strives to offer mechanism and not policy in signal and parameter handling on the part of the user. By mechanism and not policy, we mean that the users are free to do whatever they wish with the signals and parameters of a running instance of a real-time application.

In other words, while STRTL offers a great deal of ease in parameter tuning and/or signal viewing (via Simulink scope / display blocks) of the real-time process(users practically require no need of knowledge of the C/C++ programming language) users are restricted to the interface offered by Simulink. This is fine for control engineers but is not of much help for somebody wishing to play with signals his own way. The down side of the latter case(and yes, nothing comes for free!) is that the user has to have some C/C++ programming know how since an API to access the signals and parameters of the real-time application has to be used. But the programming interface is quite straight forward.

3 CAPI_STRTL Versus STRTL

Below are some similarities and differences in the workings of CAPI_STRTL and STRTL. Some of the features (as regards CAPI_STRTL) mentioned below are subject to change.[2]

- Parameter Tuning and Signal Handling

With STRTL, parameter tuning and signal handling is done via the interface provided by Simulink. Scope blocks are used to view the signals generated by a running instance of the real-time process in the (remote) computer while parameters are passed to the kernel module via Dialog boxes of the corresponding blocks of the Simulink model version of the real-time application. Once the user configures Simulinks Configuration /Simulation parameters Dialog Box and generates code all that is left to be done is compilation in the target machine, connection to, and execution control from the host computer. This mode of operation in Simulink/RTW jargon is referred to as External Mode; the low-level transport layer handling physical transmission of messages between the client and server being the TCP/IP protocol.

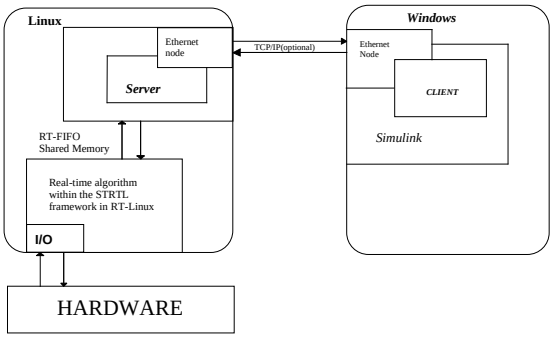

FIGURE 1: *STRTL Operation and Usage in External Mode*

In the case of CAPI_STRTL, the user employs a specific signal naming protocol. This protocol marks the signals to be read and specifies the order in which these signals will be passed to the users custom application. The use of scope blocks in the Simulink model is simply ignored in the code generation process. Parameter tuning - at runtime- is done via a consistent API. Thus, the user has to write a custom user land[3] program to handle these signals and parameters.

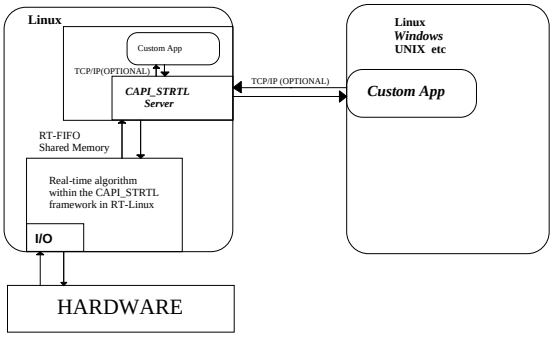

FIGURE 2: *CAPI_STRTL Operation and Usage*

[2]See future direction below
[3]See future direction below

STRTL implements a Discarding Algorithm to improve real-time monitoring in Local Area Networks. It discards signal points (SIMULINK packets) according to the hardware-network infrastructure[7]. Now as CAPI_STRTL delivers signals and parameters to custom applications, each with varying implementation goals a variety of communication mechanisms arise from monitored non-buffered reads in real-time to non-synchronised buffered signal acquisition. At the time of the writing of this paper, a simple synchronisation mechanism has been implemented that enables users perform non-buffered reads (of signals)in real-time(i.e. synchronisation is maintained by the kernel mode task); the user - via a CAPI_STRTL simulink library M S-function Simulink block,*GetSignals* - specifies the rate at which the userland application can handle signals generated by the real-time process (much in the same way as would have been done with scope blocks in STRTL) and later on, during real-time program execution, the kernel mode task writes these signals into shared memory at the rate specified while counter checking whether the user space process is reading this data in real-time. If not, a synchronisation error is flagged. The solution in this case would be to specify a lower frequency value in the M S-function, *GetSignals* and redo the code generation and build process.

- Code Generation

When generating C code from a Simulink model for any target platform, RTW requires a script file (or a .tlc file) called a System target file for its Target Language Compiler (TLC). When targeting the STRTL platform, the strtl.tlc System target file is used to control the code generation process for the Simulink model and the strtfl.tmf template makefile to generate a Makefile for the target machine. In the case of the CAPI_STRTL target, the capi_strtl.tlc and capi_strtl.tmf file are usedinstead as the System target file and the template makefile respectively. However, this implementation style is a temporary phase.[4]

- Compilation and Execution

Applications based on either of the CAPI_STRTL and STRTL frameworks are compiled and loaded into the RT-Linux kernel in an identical fashion, i.e. by use of the make f MODEL.mk command. However, to get up and running, the Simulink interface is used with STRTL in 'External Mode' while a custom application is needed to handle signals and parameters in the CAPI_STRTL framework.

- Mischalleneous

[4]See future direction below

CAPI_STRTL implements the same watchdog algorithm used in STRTL to ensure that the target platform can handle the requested sample rate(s). CAPI_STRTL like STRTL supports both Single and Multi-tasking modes.

4 EXAMPLE

1. A Simple Signal Data Read Example.

   Below is a Simulink model that contains a wrapper S-Function that invokes some custom control algorithm which we can consider as a black box as the details of its implementation are trivial. What is of interest to us here is how the user will obtain signals. The signal lines labelled "x_1", "x_2" and "x_3" will be captured by CAPI_STRTL at the rate specified by the sample time of the "ln1" block that drives the *GetSignals* M S-function. A MultiQ-3 DAQ board is used here. You could replace this with your card interface or the Comedi (kernel mode)library interface.

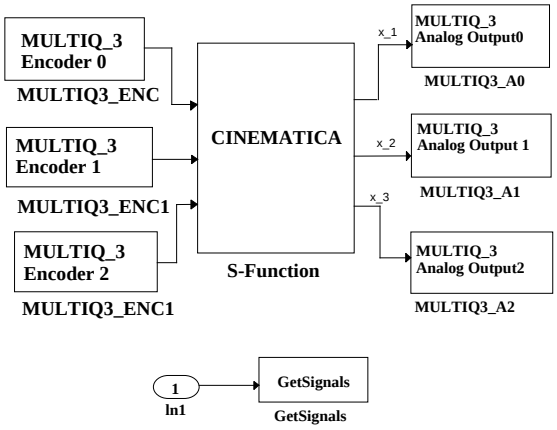

FIGURE 3: *Simulink model with a 'Black Box' S-function*

Now there exist two small but important structures which store signal and parameter information. These structs are defined in the params_signls.h header file. The user application should use these structures to extract signal info passed from the real-time process or send parameters to it. These structs are defined as follows.

```
typedef _rtSignls{
     char signlsName[SZ_SIGNLSNAME];
```

```
     double signlsValue;
}rtSignls;

typedef _rtParams{
     char paramsName[SZ_PARAMSNAME];
     double paramsValue;
}rtParams;
```

Below is the gist of a typical signal reading application to be used with the above implementation of a Simulink model.

```
/* required Unix headers here */
#include ‘‘params_signls.h’’
#include ‘‘c_api_common.h’’
int main( )
{
rtSignls *userSignls;
/*
 * socket creation, connection,
 * buffer size initialization,
 * misc. pointer sizes etc etc
 */
while(1){
  capi_strtlRead(userSignls);
  /* signals now in userSignls buffer i.e.
   * a pointer now to an array of rtSignls
   * structs...so do something (or nothing)
   * with signals
   */
  }
}
```

2. A Real World Demo:

   CAPI in HAPTICS

   Haptics is the science of applying touch (tactile) sensation and control to interaction with computer applications. By using special input/output devices (joysticks, data gloves or other devices), users can recieve feedback from computer applications in the form of felt sensations in the hands or other parts of the body. In combination with a visual display, haptics technology can be used to train people for tasks requiring hand-eye coordination, such as surgery and space ship maneuvers. It can also be used for games in which you feel as well as see your interactions with images. Haptics offer an additional dimension to virtual reality or 3-D environment. A number of universities are experimenting with haptics.

   REVIMA is a haptic system developed at the CEIT[5] Applied Mechanics Department, Spain[8]. REVIMA is a multidisciplinary enterprise that encompasses mechanical design, control theory, computer graphics, computational geometry and human-machine interaction. The system runs on 2 PC's. One computer is charged with executing the control loop (that controls the haptic interface) while the other handles the graphics engine, GUI and the algorithms of the collision solver. Currently, a 500MHz Celeron CPU executing the system control algoritm at 2KHz within the CAPI_STRTL framework is being implemented. The scheme of the system architecture is presented in the figure below.

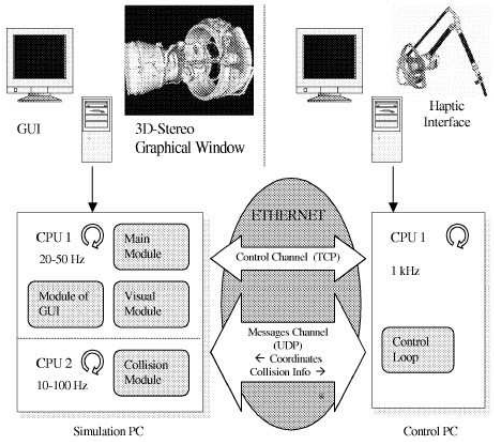

FIGURE 4: *REVIMA Schematic*

The above control system has been implemented in CAPI_STRTL for MATLAB 6.5, SIMULINK/RTW 5.0.1. A port to MATLAB 7.0, SIMULINK/RTW 6.0 is currently underway.

5 STATE OF AFFAIRS

1. The stable and tested version of CAPI_STRTL is for MATLAB 6.5, SIMULINK/RTW 5.x. A port to MATLAB 7.0, SIMULINK/RTW 6.x is currently underway and should be ready by November or December 2005. This was only due to lack of availability.

2. The TCP/IP communication implemenation between the custom applications and the CAPI_STRTL framework needs a more rigorous analysis and testing. In addition, once CAPI_STRTL is ready for download, more communication mechamisms will be augmented according user feedback.

[5]Centro de Estudios e Investigacoines Tecnico de Gipuzkoa

3. CAPI_STRTL may not work well (or not work at all) with the more obscure Simulink blocks and Non-Inlined S-functions. Wrapper S-functions are the best alternatives in such situations.

6 FUTURE DIRECTION

Currently, CAPI_STRTL and STRTL are separate applications. However, their execution engines for the control algorithms generated by RTW are practically identical. With a few #ifdefs and such precompiler directives, there is no (known) reason why CAPI_STRTL cannot be merged into STRTL. As far as the authors are concerned, STRTL is only MATLAB 6.x and SIMULINK/RTW 5.x compatible. A port may be neccessary to MATLAB 7.x and SIMULINK/RTW 6.x. As mentioned before, IPC[6] mechanisms between CAPI_STRTL and user applications are subject to further study e.g. use of real-time sockets.

7 DOWNLOAD

CAPI_STRTL should be ready for free download and usage with associated documentation and tutorials by December or thereabouts from this web site: www.tecnun.es/bitcrib/asiro/

[6]Inter Process Communication

References

[1] Raul Murillo Garcia, 2002, *Simulink Target for RT-Linux*, Caledonian University

[2] Roberto Butcher, University of Applied Sciences of South Switzerland, Lorenzo Dozio, Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano, 2003,*CACSD under RTAI Linux with RTAI-LAB*, Fifth Real-Timw Linux Workshop, Valencia, Spain,

[3] www.realtimelinuxfoundation.org

[4] www.mathworks.com/ products/product_listing/index.html

[5] www.scilab.org

[6] Vishal J.Desai DA-IICT Near Indroda Circle, Gandhinagar, Gujarat, India, vishal_desai@da-iict.org, 2003, *Interfacing RT-Linux and Simulink*,Fifth Real-Time Linux Workshop, Valencia, Spain.

[7] The Mathworks Inc., *Real-Time Workshop User's Guide*

[8] Savall.J.Borro, D.Gil J.J, Matey.L, 2002, CEIT,*Description of a Haptic System for Virtual Maintainability in Aeronautics*,Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), EPFL, Lausanne, Switzerland pp2887–2892.