The SynDEx \texttt{“linuxIO_”} Macro:
An Easy C/C++ Linux User’s Application Interface

Pierre Pomiers
Robosoft S.A.
Technopole d'Izarbel, 64210 Bidart, France.
Tel.: +33-5-59 41 53 66; Fax.: +33-5-59 41 53 79
E-mail: pierre@robosoft.fr

Draft 0 printing, April 20, 2001
The SynDEx “linuxIO.” Macro:
An Easy C/C++ Linux User's Application Interface
Contents

Introduction 9

1 System requirements 11
  1.1 Development platform ............................................. 11
    1.1.1 Computer hardware ............................................. 11
    1.1.2 Linux environment ............................................. 12
    1.1.3 Linux real-time extension (RTAI) .............................. 12
    1.1.4 SynDEx tools .................................................. 13
  1.2 Robosoft control environment ...................................... 14
    1.2.1 Robosoft MPC555 control board ............................... 14
    1.2.2 Cross compiler tools ......................................... 16

2 “linuxIO” overview 19
  2.1 Acknowledgments .................................................. 19
  2.2 General principle ................................................ 20
    2.2.1 “linuxIO” specification ..................................... 20
    2.2.2 “linuxIO” implementation ................................... 20

3 “linuxIO” programming basics 25
  3.1 The “linuxIO” SynDEx interface .................................. 25
    3.1.1 How to include it from application macro executive .......... 25
    3.1.2 How to use it from SynDEx CAD ................................ 25
    3.1.3 The generated headers file interface .......................... 26
  3.2 How to use the generated “linuxIO” interface from C/C++ programs .............................................. 29

4 User’s code development techniques 31
  4.1 Initializing RTAI shared memory .................................. 31
  4.2 The read/write operations ......................................... 31
  4.3 RTAI shared memory finalization .................................. 32

5 How to program graphical user interfaces (GUIs) 33
  5.1 About real-time embedded GUI .................................... 33
  5.2 The Qt/Embedded alternative ....................................... 34
A Sample code using “linuxIO_” C interface
List of Figures

1.1 SynDEx CAD software example .................................................. 14
1.2 The MPC555 based Robosoft control board .................................. 15

2.1 User application general principle ................................................. 21
2.2 High-level application general principle ....................................... 22
2.3 Low-level application general principle ....................................... 23

3.1 New operation type declaration for “linuxIO_” ................................ 27
3.2 New operation call for “linuxIO_” ................................................. 28
3.3 Linio operations inserted in the data-flow graph .............................. 29

5.1 Screenshot of a GUI application written with Qt/Embedded .................. 35
List of Tables

1.1 Robosoft control board connectors description .......................... 15
The SynDEx “linuxIO_” Macro:
An Easy C/C++ Linux User’s Application Interface
Introduction

This document applies to programmers intended to use SynDEx for real-time applications implementations. Relying on the AAA\(^1\) methodology, SynDEx is known to be both a gentle tool for distributed architectures handling and a good way of optimizing hardware resources management. Nevertheless, even if SynDEx provides an easy graphical user interface for application design, when users need extra functionalities (not implemented in standard) thing are getting harder. Indeed, before creating new real SynDEx function (called “macros” in the SynDEx world), user is intended to know about the SynDEx system structure and the philosophy of its low level programming. In fact, this effort is not so different from what is necessary for other type of operating system programming, but this deep understanding is probably something common users do not want or at least do not really need. The “linuxIO_” SynDEx macro brings a solution to this category of programmers. Thanks to it, interfacing classical C/C++ programs with SynDEx generated executives becomes possible and very easy.

\(^1\)Algorithm Architecture Adequation
Chapter 1

System requirements

This chapter is dedicated to the set of hardware and tools you need to use the “linuxIO_” C/C++ interface. In this document, the “linuxIO_” macros is described for use in a particular context: that is to say in SynDEx programs driving architectures composed of Intel i386 Linux/RTAI machines and Motorola MPC555 based control boards\(^1\) with CAN buses as communication media. The following section cover both development and the embedded target platform.

1.1 Development platform

Creating real-time embedded solutions with SynDEx requires a correctly configured tools chain for development. All required components are listed and detailed below.

1.1.1 Computer hardware

Before installing the development environment, you need an Intel i386 family based PC (normally Intel processors from 80386 to recent P4 and others AMD or Cyrix clones are supported). Other platforms are not supported as the tools we use has not been ported yet, or do not work properly.

This PC also has to be equipped with a CAN bus board. We recommend to use a NuDACQ PCI-7841 CAN Interface. This card is based on Philips SJA1000 chip. Actually, SynDEx routines only support this type of device. Ports for other devices are possible and should be rather easy for experimented programmers.

No other specific device is needed for real-time applications development and support.

\(^1\)Boards manufactured by Robosoft as part of his own control system environment
1.1.2 Linux environment

The development platform we propose, is running under Linux operating system. Any Linux distribution (Redhat, Debian, Slackware...) can fit our needs. The only constraint concerns the kernel version. Your Linux must run the 2.2.14 kernel version. The reason for this is explained in subsection 1.1.3. For those reasons, Robosoft recommends to install a Redhat 6.1 distribution with a 2.2.14 kernel update or a Redhat 6.2. Compatibility between tools is insured with these two distributions.

Concerning standard tools, you will need at least “gmake”, “m4”, “sed” and a complete native GNU compiler (with “gcc”, “gas”, “ld”...) installed on your computer. Tools dedicated to cross compiling operations are also needed for MPC555 target programming; information about them is given in subsection 1.2.2.

1.1.3 Linux real-time extension (RTAI)

The context of real-time embedded applications programming is quite different from the classical one, user usually meets. This notion of “real-time” is not present in normal Linuses.

Such real-time dedicated mechanisms can be added by installing an RTOS\(^2\) on top of Linux standard kernel. It exists two main variants of real-time Linuses available for free:

- RTAI (refer to [1])
- RTLinux (refer to [2])

Robosoft based its product on RTAI version which is widely used in embedded industry for prototyping and which is supported by very active companies.

RTAI basic principle is rather simple. RTAI provides deterministic and preemptive performance in addition to allowing the use of all standard Linux drivers, applications and functions. To this aim, the real-time Linux scheduler treats the Linux operating system kernel as an idle task. Linux only executes when there are no real-time tasks to run, and the real-time kernel is inactive. That is to say that the Linux task can never block interrupts or prevent itself from being preempted. RTAI decouples the mechanisms of the real-time kernel from the mechanisms of the general purpose Linux kernel so that each can be optimized independently and so that the real-time kernel can be kept small and simple. In other words, the fundamental design philosophy of RTAI is to let the Linux operate as far as possible with regards to the real-time application constraints.

RTAI’s performance is very competitive with commercial RTOS, such as VxWorks or QNX, offering typical context switch times of 4 microseconds, 20 microseconds interrupt response and up to 100 kHz periodic tasks rate. The main limitation being imposed by the system hardware, rather than the real-time software itself.

\(^2\)Real-Time Operating System
RTAI relies on the Linux loadable module mechanism to install components of the RTOS, which keeps it extensible and modular. Let us note that loading a real-time module is an operation Linux user can do only if he has root permissions. The primary function of the real-time kernel is to provide real-time tasks with direct access to the raw hardware, so that they can execute with minimal latency and maximal processing resource, when required.

Even if RTAI versions are expected to be compatible, keep in mind that Robosoft development tools rely on RTAI version 1.3. If major changes occurred between version 1.3 and newer versions, we can not guaranty our product to work properly. That is why we highly recommend to use RTAI version 1.3 in conjunction with a version 2.2.14 of the Linux kernel.

1.1.4 SynDEx tools

The application development method discussed in this document makes use of SynDEx tools. SynDEx, developed by INRIA, is a graphical interactive software (see figure 1.1) with on-line documentation (refer to [3]), implementing the AAA methodology. As the “linuxIO...” mechanism is managed through SynDEx generated executives, SynDEx must be installed on the development platform. For information, here is the list of services offered by SynDEx:

- specification of an application algorithm as a conditioned data-flow graph (or interface with the compiler of one of the Synchronous languages ESTEREL, LUSTRE, SIGNAL through the common format DC)
- specification of a multicomponent as a graph
- heuristic for distributing and scheduling the algorithm on the multicomponent with response time optimization
- visualization of predicted real-time performances for the multicomponent sizing
- generation of dead-lock free executives for real-time execution on the multicomponent with optional real-time performance measurement. These executives are built from a processor-dependent executive kernel. SynDEx comes presently with executives kernels for various digital signal processors, microprocessors and microcontrollers.

The distributing and scheduling heuristics as well as the predicted real-time diagram, help the user to parallelize his algorithm and size the hardware while satisfying real-time constraints. Moreover, as the executives are automatically generated with SynDEx, the user is relieved from low level system programming and from distributed debugging. This allows optimized rapid prototyping and dramatically reduces the development cycle of distributed real-time applications.
The SynDEx “linuxIO.” Macro: An Easy C/C++ Linux User’s Application Interface

The latest version of SynDEx CAD can be downloaded from the SynDEx web site (http://www-rocq.inria.fr/syndex/download.htm) and the latest on-line documentation can be consulted at the following address: http://www-rocq.inria.fr/syndex/doc/index.htm. Additional macro-executives for MPC555 based Robosoft control board and RTAI i386 computer support are distributed and maintained by Robosoft (contact us for more information about availability and distribution conditions).

1.2 Robosoft control environment

This section is about the target embedded system environment. This system is a part of Robosoft product. It is composed of a MPC555 based control board and a set of tools dedicated to development for this type of machine.

1.2.1 Robosoft MPC555 control board

The Robosoft MPC555 based board (see figure 1.2) is a stand-alone 4-axis controller designed for critical industrial process handling. Including a 32-bit PowerPC architecture, it provides high computation performance. This controller is split into two boards:

- A 100 millimeters by 165 millimeters main board including the processor (cadeden at a 10 MHz frequency), an interface for controlling one axis, a set of isolated inputs outputs and all communication ports.
Table 1.1: Robosoft control board connectors description

<table>
<thead>
<tr>
<th>Figure label</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MPC555</td>
</tr>
<tr>
<td>2</td>
<td>Power supply: 18-60V DC (from batteries)</td>
</tr>
<tr>
<td>3</td>
<td>BDM Interface (Basic Debug Interface)</td>
</tr>
<tr>
<td>4</td>
<td>7 analog inputs</td>
</tr>
<tr>
<td>5</td>
<td>16 logical inputs and 20 logical outputs</td>
</tr>
<tr>
<td>6</td>
<td>Synchronous serial line (SPI)</td>
</tr>
<tr>
<td>7</td>
<td>Asynchronous serial lines (port 0)</td>
</tr>
<tr>
<td>8</td>
<td>Asynchronous serial lines (port 1)</td>
</tr>
<tr>
<td>9</td>
<td>CAN bus (port 0)</td>
</tr>
<tr>
<td>10</td>
<td>CAN bus (port 1)</td>
</tr>
<tr>
<td>from 11 to 14</td>
<td>4 connectors dedicated to axis control (including 1 analog output per axis)</td>
</tr>
</tbody>
</table>
The SynDEx “linuxIO.” Macro:
An Easy C/C++ Linux User’s Application Interface

- A 100 millimeters by 100 millimeters “piggy back” card extending the main board features by adding support for three axis. The PC104 like connector allow to add custom modules for future device (additional input and output ports, RAM or EEPROM extension).

Refer to table 1.1 for a detailed description of boards connectors.

For information, an overview of the MPC555 microcontroller features is given above (for details, consult the manufacturer datasheet [4]):

- PowerPC TM Core with Floating-Point Unit
- 26 Kbytes Fast RAM and 6 Kbytes TPU Microcode RAM
- 448 Kbytes Flash EEPROM with 5V Programming
- 5V I/O System
- Serial System: Queued Serial Multi-Channel Module (QSMCM), Dual CAN 2.0B Controller Modules (TouCAN TM)
- 50-Channel Timer System: Dual Time Processor Units (TPU3), Modular I/O System (MIOS1)
- 32 Analog Inputs: Dual Queued Analog-to-Digital Converters (QADC64)

1.2.2 Cross compiler tools

Developing applications for bare boards, like the Robosoft one, requires the use of a cross compiler (as, in most cases, no native compiler exists for this stuff). The GNUPro Toolchain is a set of free tools distributed by GNU under GPL license, providing cross-platform development tools for 32 and 64-bit microcontrollers. This chain fully supports ELF 32-bit PowerPC platforms, which fits our needs for MPC555 embedded software development.

An equivalent of each classical Linux native tools is found in GNUPro Toolchain packages. The commonly used ones are:

```bash
ppc-elf32-gcc  (C compiler)
ppc-elf32-gas   (assembler)
ppc-elf32-cpp   (C preprocessor)
ppc-elf32-ld    (linker)
```

It also contains a set of binary utilities such as:

```bash
ppc-elf32-objdump  (for object file information display)
```
The SynDEx “linuxIO” Macro: An Easy C/C++ Linux User’s Application Interface

**ppc-elf32-ar** (for object code archives management)

**ppc-elf32-ranlib** (for archive index generation)

**ppc-elf32-cmp** (for byte-by-byte files comparison).

Let us note that each of these tools is prefixed by “ppc-elf32-” in order to differentiate it from the native ones (for instance, when writing makefiles). The GNUPro Toolchain documentation is included in the packages sources and can be consulted through text files or manpages. For on-line HTML documentation refer to the related Redhat web site: [http://www.redhat.com/support/manuals/gnupro98r2/](http://www.redhat.com/support/manuals/gnupro98r2/). A specific part of the documentation related to the MPC555 and more generally the PowerPC family can also be found in HTML format under the “doc” directory of the source tree.

A cdrom copy of the complete toolchain can be obtained from Robosoft. This archive includes a user friendly installation script that configures tools sources for a correct Robosoft MPC555 controller board support. It also includes our own target specific linker scripts file, containing crucial board related information for the GNU compiler.
The SynDEx “linuxIO_” Macro:
An Easy C/C++ Linux User’s Application Interface
Chapter 2

“linuxIO_” overview

This chapter intends to explain “linuxIO_” basic principle. After a short recall about required programmer skills, we explain what “linuxIO_” should be used for. Finally, in the last section, dedicated to more technical information, we present how it has been implemented and which are the underlying techniques we are making use of.

2.1 Acknowledgments

From now, we assume that “linuxIO_” users, intending to develop C/C++ applications compatible with SynDEx executives, are familiar with the following notions:

- C/C++ programming (writing, compiling and running programs)
- specifying algorithm with SynDEx CAD
- specifying architecture with SynDEx CAD
- specifying operations and communication characteristics with SynDEx CAD
- specifying distribution constraints with SynDEx CAD
- obtaining scheduling with SynDEx CAD
- generating executives with SynDEx CAD
- running a distributed application using SynDEx procedures

As mentioned in a previous section, it exists a web page that fully explains SynDEx techniques. Thus, we highly recommend users to refer to this address (http://www-rocq.inria.fr/syndex/doc/index.htm) for any question about one of these points.
2.2 General principle

2.2.1 “linuxIO_” specification

The use of “linuxIO_” takes place in a particular context: interfacing C/C++ user’s application with critical real-time embedded applications developed with SynDEx. With respect to this programming context, application development organization falls into two main parts. The first one consists in developing the low-level software part of the application, while the other one concerns the high-level software architecture. Let us detail their respective contents.

The low-level part of the user application is intended to offer a total control under hard real-time constraints of both actuators and sensors embedded in the controlled platform (for instance motor driving, analog input acquisition and so on). In our specific case, the platform architecture is composed of one x86 machine (running Linux and RTAI) and one or more MPC555 control boards. Then, all elements of this hardware architecture have to be linked each others through a CAN bus network in order to allow data exchanges between them. This part is intended to be exclusively developed using SynDEx tools. Indeed, SynDEx macros set provides optimized and reliable services able to fit the real-time application constraints. First, both hardware and software architecture are designed with SynDEx CAD tools. Afterwards, the Motorola MPC555, x86 RTAI and CAN bus media SynDEx macro executives are used for generating a specific application executive, able to run on the previously described hardware architecture. This generated critical application will run both under RTAI and MPC555 control board.

On the opposite, the high-level part concerns all the software in charge of time consuming processes, such as trajectory planning, navigation strategies, network oriented applications or any other non critical, user friendly and users oriented stuff. Such tasks are expected to be run under a standard Linux session on the controlled platform x86 computer. At this level, a user application is developed independently from the low-level part.

As a conclusion, with “linuxIO_” SynDEx macro, our intention is to allow user to make its pieces of software act on the hard real-time controlled system, as far as possible.

2.2.2 “linuxIO_” implementation

In this subsection, we intend to explain how those two main parts are organized inside a process\(^1\) control application and what kind of relation exists between them. Figure 2.1, shows the general concept of a user application where the two separated high-level and low-level parts are represented.

High-level part is detailed on figure 2.2. On this picture, we mention that data are exchanged using shared memory segments. These segments, created by RTAI, are parts

\(^1\)By process we mean any system aiming to operate and transform its own physical environment.
of both the RTAI kernel and the Linux users space. The shared memory is accessed by kernel services allowing programs to manipulate it (allocate, read, write or free).

Nature of the data, Linux user space programs are allowed to access, is chosen by the application designer. With no restriction, all resources present on the Robosoft control board can be shared (refer to table 1.1 for details). Thanks to this data sharing method, any high-level Linux process is able to access control board hardware asynchronously (that is to say without disturbing hard real-time software behavior) and use it from non-SynDEx time consuming computation processes. Several obvious advantages for this approach appear:

- From a security point of view, it is important to guaranty that making changes on a part of the high-level process will not cause any bugs on the low-level process, neither lead to any modifications of a previously bug-free low-level API developed separately (and generated automatically by SynDEx).

- User can take benefits of using Linux kernel classical services (such as network, video or sound supports for instance) which is really helpful when performing customers applications.

Now, let us focus on the description of the low-level user application. This part directly implements handling for the hardware architecture (previously defined using SynDEx CAD tool). Basically, a set of low-level hardware elements (sensors and actuators needed for the application) is handled by one (or more) MPC555 controller boards. Besides,
for communication purpose, each MPC555 control board is equipped with two CAN bus interfaces. These communication ports allow controllers to exchange data with distant units plugged on the bus. This is what is used for sending and receiving data operations between MPC555 controllers and also between MPC555 controllers and x86 computer (including the CAN device which references are given in section 1.1.1).

From now, let us focus on how low-level application operates. On figure 2.3, each inputs and outputs (depicted as the micro-controllers I/O layer) are handled by their respective low-latency assembly pieces of software, executed under a 1 ms timer interrupt. Thus, data flow get from I/O layer can be processed by MPC555 micro-controllers under hard real-time constraints. In the described architecture, still remains the x86 computer running under RTAI/Linux kernel. This computer aims to handle real-time processes (executed each 10 ms period) less critical than the ones running on the MPC555 micro-controllers. As part of the "linuxIO_" macro, the x86 side of the architecture is used for updating the shared memory for Linux user space data exchanges (which is considered to be a time consuming task regarding to the rest of the application).
The SynDEx “linuxIO_” Macro:
An Easy C/C++ Linux User’s Application Interface

Figure 2.3: Low-level application general principle
Chapter 3

“linuxIO_” programming basics

Now “linuxIO_” principles are known, we naturally come to this new chapter where is explained the way to use “linuxIO_” macro from SynDEx CAD. In the last chapter section, we give users the method for interfacing high-level Linux applications with a SynDEx designed algorithm via “linuxIO_”.

3.1 The “linuxIO_” SynDEx interface

3.1.1 How to include it from application macro executive

The “linuxIO_” macro is provided by the macro executive file named: “linio.m4x”. Before designing with SynDEx an application that make use of “linuxIO_”, you have to include the “linio.m4x” file from your own application dedicated macro executive file. For instance if you want to create an application named “my_app”, you have to include “linio.m4x” from your “my_app.m4x” file. This is done inserting the following M4 command line before your own definitions:

```
include(linio.m4x)
```

This step both validate the “linuxIO_” macro definition for later executive compilation and make it accessible from the SynDEx CAD interface.

3.1.2 How to use it from SynDEx CAD

Before you begin to specify the dependence graph corresponding to your application algorithm, you have to add “linuxIO_” as an existing operation for operator type “555” (see in toolbar: “menu/Adequation/AIOp-ArOp”). At that time, when “Enter the new operation type you need” is requested, answer by: “linuxIO_{X,Y}” where “X” (respectively “Y”) is the number of inputs (respectively outputs) you want to use (see figure 3.1).
These specified numbers should correspond to the number of inputs and outputs you indicate in fields “Number of in_ports” and “Number of out_ports” of the “Algorithm/New operation” dialog box (see figure 3.2). Let us remark that “linuxIO_” operation is of generic type “extio” for initialization purposes. This will probably change in next version of SynDEx, which will implement a specific generic type for this kind of asynchronous operations. If “linuxIO_” boxes of different numbers of inputs and/or outputs are needed, create as many new “linuxIO_{X,Y}” operations as you need, from the “menu/Adequation/AOp-ArOp” menu.

The box you obtain, as a result of “Algorithm/New operation”, aims to give you an access both to signals entering the box (inputs) and signals leaving the box (outputs) from any Linux user space program (see figure 3.3).

### 3.1.3 The generated headers file interface

Once design of your application including one (or more) “linuxIO_” operation(s) is done, you can compile it. For compilation procedure, nothing changes, just do it as you are used to.

During compilation time, an header file, named “linio.h”, is automatically generated. This header file contains all information you need to access input (for reading) and output (for writing) signals related to all “linuxIO_” operations inserted in the algorithm graph. Let us see how is composed the header file generated for the application depicted on figure 3.3. The content of this file, listed below, falls into two parts: one for each “linuxIO_” operation present in the graph.

```c
/* linio.h: auto-generated C users API frontend. *
 * Copyright Robosoft 2001, Pierre Pomiers <pierre@robosoft.fr>. */

/* LINUX IO shared memory data segments. */

#define SEGO_BASE 0x10000

typedef struct {
    int f_spi_o0;
    int f_tpu1_o0;
    int f_tpu2_o0;
    int f_qadc0_o0;
    int f_qadc1_o0;
    int f_linio_o0;
    int f_linio_o1;
    int f_linio_o2;
    int time;
    unsigned char sem;
} shm_seg0_def;
```
The SynDEx “linuxIO” Macro: An Easy C/C++ Linux User’s Application Interface

```c
#define SEG1_BASE 0x10025
typedef struct {
    int r_spi_o0;
    int r_tpu1_o0;
    int r_tpu2_o0;
    int r_qadc0_o0;
    int r_qadc1_o0;
    int r_linio_o0;
    int r_linio_o1;
    int r_linio_o2;
    int time;
    unsigned char sem;
} shm_seg1_def;
```

Figure 3.1: New operation type declaration for “linuxIO.”

For instance, let us explain what is found in the first part, corresponding to the “f_linio” operation. The first defined macro ("#define SEG0_BASE 0x10000") corresponds to the name of the shared memory segment allocated for this “f_linio” operation. It will be used later in user’s program for getting a pointer on structure where data are located. Then, following this macro, is defined the type of the structure that contains the concerned data.

The first five structure fields (from “f_spi_o0” to “f_qadc1_o0” in the example) correspond to the five operation input signals. These values aim to be read and are periodically refreshed by a part of the SynDEx generated RTAI application running on the
x86 computer. Reading one of these fields gives the corresponding current input signal value.

The following three structure fields ("f_linio_00", "f_linio_01" and "f_linio_02" in the example) represent the three operation output signals. These values aim to be written and are periodically taken in account by an other part of the SynDEx generated RTAI application. Asserting one of these fields updates the corresponding signal value on the hardware architecture. Note that updates are not bufferized: that is to say that, only the last one will be read by the periodical hardware updating process.

Let us remark that two extra fields are mentioned in the structure: “time”, corresponding to the time (expressed in nanoseconds) when the data were get from the architecture, and “sem”, aiming to be used as a semaphore for under user’s control data locking/unlocking mechanism. These two fields can freely be used by user and have no influence on the hardware architecture.
3.2 How to use the generated “linuxIO” interface from C/C++ programs

This step is quite direct. Basically, using the “linuxIO” interface from C/C++ codes consists of including the automatically generated header file. This can be done from a C program with the following lines:

```c
#include <stdio.h>
#include <unistd.h>
#include <sched.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <rtai_shm.h>
```
The SynDEx “linuxIO” Macro: An Easy C/C++ Linux User’s Application Interface

```c
#include "linio.h"

and can be done from a C++ program with these other ones:

```c
#include "linio.h"
extern "C" {
    #include <stdio.h>
    #include <unistd.h>
    #include <sys/ioctl.h>
    #include <sys/types.h>
    #include <sys/mman.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    #include <rtai_shm.h>
}
```

where “rtai_shm.h” is the RTAI shared memory header file provided with the standard RTAI distribution.

Thus all information for shared memory access are available from user’s program. Actions to be done for initializing, reading, writing and terminating shared memory access are discussed in the chapter 4.
Chapter 4

User’s code development techniques

This chapter presents operations needed by users who want to work with RTAI shared memory and, by extension, who want to make use of the “linuxIO_” shared memory segment. These operations are available for C and C++ implementations. Other languages are, a priori, not supported yet (for more information refer to the RTAI web site [1]).

4.1 Initializing RTAI shared memory

Let us go back to the “linuxIO_” example given in subsection 3.1.3. For initializing segments reserved by RTAI for “linuxIO_” operations, we use information reported in the “linio.h” header file. Initializing consist of using an RTAI service available from Linux user space in order to access the segments. Here is the piece of code used for initializing the first segment identified by “SEG0_BASE”:

```c
/* seg0 structure declaration */
static shm_seg0_def *seg0;
...
seg0 = rtai_malloc(SEG0_BASE, sizeof(shm_seg0_def));
```

This is actually not a real memory allocation, that has already been done by the running SynDEx generated RTAI process. In fact, called from the Linux user space, “rtai_malloc” just maps the “linuxIO_” RTAI shared memory area to the user space and returns the related pointer (in the example this pointer is “seg0”).

4.2 The read/write operations

Reading data from, as well as writing data down to, the hardware architecture is extremely easy as it is just a matter of manipulating classical C/C++ typed variables. For instance, if we intend to read the value of the “f_tpu2” signal (linked to one of the “f_linio” inputs), let us proceed as follow:
The SynDEx “linuxIO_” Macro:
An Easy C/C++ Linux User’s Application Interface

```
int val;
...
/* Read "f_tpu2" signal value, put it into "val" for later use... */
val = seg0->f_tpu2_o0;
/* ... and display it on the screen. */
printf("Read value of \"f_tpu2\" is: %d\n", seg0->f_tpu2_o0);
```

Do keep in mind that signal data are available through the structure pointer, that means that you can use it as you want and not only as it is presented in the above sample of code (in other word it is nothing but a classical pointer, without any restriction).

Writing procedure is as simple as reading. For this, user just have to assert the value of the desired structure field. Let us illustrate this by the following code:

```
/* Set "f_linio_o0" output signal value to 700... */
seg0->f_linio_o0 = 700;
/* ... and display it on the screen. */
printf("Written value of \"f_linio_o0\" is: %d\n", seg0->f_linio_o0);
```

Note that asserting the structure “output” fields cause hardware environment modifications. In the above example, we set “f_linio_o0” field to 700. As “f_linio_o0” signal is connected to a PWM operation, the user could measure (with an oscilloscope) a physical 70% duty cycle PWM signal on the corresponding hardware output connector pin.

### 4.3 RTAI shared memory finalization

Before leaving his application written in C or C++, user have to free the memory segments access he previously got with “rtai_malloc” call. Back to our example, here is the code line needed to free the segment identified by “SEG0_BASE” and which current pointer is “seg0”:

```
rtai_free(SEG0_BASE, seg0);
```

Functioning for freeing memory is basically the same as the allocation one. Analogously the freeing calls have just the effect of unmapping the “linuxIO_” RTAI shared memory area from the Linux user space. As before for allocation, the real memory free is done when the running SynDEx generated RTAI process terminates.
Chapter 5

How to program graphical user interfaces (GUIs)

This chapter deals with something getting more and more important in modern user oriented products: the GUIs\(^1\). Embedding high quality video feedbacks or interactive control panels is challenging when applications are expected to be real-time compatible. We propose here a solution, we use in our own products, that respects such constraints.

5.1 About real-time embedded GUI

In the previous chapters, we present “linuxIO...” macros as an easy mean for interfacing SynDEx generated hard real-time constrained applications with user oriented high-level programs. GUIs are parts of these so called high-level software. Indeed GUIs should help application user to drive or interact with the hardware architecture, but it seems to be important that GUIs should not disturb the functioning of the real-time synchronized application part. Thus, it appears reasonable to develop GUIs at a user-oriented operating system level. Moreover, this way of proceeding, allow GUI programmer to dispose of all the up to date applications, drivers or libraries available on this operating system, which represent a gain of quality and development time.

As running under RTAI, SynDEx generated x86 computer processes can only be accessed via Linux. Thus, high-level user applications have to be developed under Linux. We decided to focus on GUIs (as high-level software example) cause GUIs management implies non-trivial problems.

When talking about GUIs under Linux environment, an evident parallel is made with XFree86\(^2\)[6]. Nevertheless, XFree86 approach is absolutely not appropriate to real-time context. A dirty pieces of code is found in XFree86 sources, for graphic accelerator based video boards handling. Basically this code aims to suspend all interrupts when

\(^1\)Graphical User Interfaces

\(^2\)An open-source implementation of the X Window
graphic display load becomes too important. Then, XFree86 only assign the video related interrupts, which effect it to give graphic accelerator the absolute priority for absorbing rapidly the requested graphical updates. Once this display load processed, XFree86 restore hardware interrupt context as it was just before it enters the “accelerating” mode.

Such procedure has dramatical effects on the real-time application. When such an application is programmed, we assume that all interrupts are mastered in order to avoid non-predictable functioning. Obviously, using XFree86 introduces periods during which it is totally impossible to know what is happening on the hardware. For instance, if during one of these time periods a real-time communication event appears on a port of the x86 computer, it will be lost as no interrupt is assigned for handling it. This directly causes a blocking loss of synchronization in the real-time system.

Thus, the choice of tools for GUIs design is crucial when programming real-time user-oriented applications. Robosoft managed to solve this problem by using a product specially dedicated to user interfaces for embedded devices.

5.2 The Qt/Embedded alternative

The Qt/Embedded[5] product provides you with all you need to create high quality video interfaces for interacting with embedded architectures. Qt/Embedded installs and runs with a very small memory footprint on any device running Linux, without making use of XFree86, which perfectly fits our requirements. Qt/Embedded features the same API as the non-embedded Qt libraries available under Microsoft Windows and X11/Linux. Moreover, embedded code versions are fully compatible with desktop versions; that is to say that you can imagine writing your Qt application in your favorite desktop environment and just recompiling to move it to your embedded device.

All of these points make Qt/Embedded be a very flexible and reliable tool. Besides, let us note that Qt/Embedded is supported by the world wide known TROLLTECH company which is very active in Linux world and participate to numbers of industrial software development projects. This insure Qt/Embedded to be one of the most durable solution for GUIs programming.

Figure 5.1 is a sample shot of a GUI we developed for an automated electrical vehicle application. We see the GUI displayed on the vehicle LCD panel. It provides user access to some hardware signals and reports information coming from various vehicle sensors. This GUI is running asynchronously (in opposition to the synchronized functioning of the real-time constrained vehicle control application).
The SynDEx “linuxIO” Macro: An Easy C/C++ Linux User’s Application Interface

Figure 5.1: Screenshot of a GUI application written with Qt/Embedded
Appendix A

Sample code using “linuxIO_” C interface

The C code listed below implements a user program, interfaced with the SynDEx designed application example given in chapter 3 (see figure 3.3).

/* (c)ROBOSOFT pierre@robosoft.fr
 * This code aims to show how it is possible to exchange data between a
 * x86 Linux/RTAI SynDEx kernel based executive and a classical C API.
 * IMPORTANT! ! !: this procedure is exclusively a part of the SynDEx
 * RTAI Linux port. */

#include <stdio.h>
#include <unistd.h>
#include <sched.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <../rtai/rtai-1.3/shmem/rtai_shm.h>
#include "linio.h"

static shm_seg0_def *seg0;
static shm_seg1_def *seg1;

static int end;
static void endme(int dummy)
{
    end = 1;
}
```c
int main(int argc, char **argv)
{
    shm_seg0_def front;
    shm_seg1_def rear;

    int out = 0;
    int f_old_time = 0;
    int r_old_time = 0;

    seg0 = rtai_malloc(SEG0_BASE, sizeof(shm_seg0_def));
    seg1 = rtai_malloc(SEG1_BASE, sizeof(shm_seg1_def));

    /* PP: Allow return only at SIGINT reception (SIGINT is emitted when
     * user press Ctrl-C). This will stop the execution of the user C
     * application loop. */
    signal(SIGINT, endme);

    /* ##################################################
     * Here begins the user C application code! ! !
     * ++++++++++++++++++++++++++++++++++++++++++++++++++ */

    while (!end) {
        if (seg0->sem != 0) {
            out = out++ % 1000;
            bcopy(seg0, &front, sizeof(shm_seg0_def));
            printf
                ("FRONT> DIFF:%10d SPI:%5d TPU1:%5d TPU2:%5d ADC0:%5d ADC2:%5d\n",
                 front.time - f_old_time, front.f_spi_o0, front.f_tpu1_o0,
                 front.f_tpu2_o0, front.f_qadc0_o0, front.f_qadc1_o0,
                 front.f_linio_o0, front.f_linio_o1, front.f_linio_o2);
            f_old_time = front.time;
            front.f_linio_o0 = out % 700;
            front.f_linio_o1 = out % 800;
            front.f_linio_o2 = out % 900;
            front.sem = 0;
            bcopy(&front, seg0, sizeof(shm_seg0_def));
            fflush(stdout);
        }
        if (seg1->sem != 0) {
            bcopy(seg1, &rear, sizeof(shm_seg1_def));
            printf
                ("REAR > DIFF:%10d SPI:%5d TPU1:%5d TPU2:%5d ADC0:%5d ADC2:%5d\n",
```
rear.time - r_old_time, rear.r_spi_o0, rear.r_tpu1_o0,
rear.r_tpu2_o0, rear.r_qadc0_o0, rear.r_qadc1_o0,
rear.r_linio_o0, rear.r_linio_o1, rear.r_linio_o2);
    r_old_time = rear.time;
    rear.r_linio_o0 = rear.r_spi_o0;
    rear.r_linio_o1 = out % 500;
    rear.r_linio_o2 = out % 600;
    rear.sem = 0;
    bcopy(&rear, seg1, sizeof(shm_seg1_def));
    fflush(stdout);
    }
}

/* ++++++++++++++++++++++++++++++++++++++++++++++++++
 * Here ends the user C application code! ! !
 * #%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

front.f_linio_o0 = 0;
front.f_linio_o1 = 0;
front.f_linio_o2 = 0;
front.sem = 0;
    bcopy(&front, seg0, sizeof(shm_seg0_def));
rear.r_linio_o0 = 0;
rear.r_linio_o1 = 0;
rear.r_linio_o2 = 0;
rear.sem = 0;
    bcopy(&rear, seg1, sizeof(shm_seg1_def));

rtai_free(SEG1_BASE, seg1);
rtai_free(SEG0_BASE, seg0);

    return 0;
}
The SynDEx “linuxIO.” Macro:
An Easy C/C++ Linux User’s Application Interface
Bibliography

   DIAPM RTAI Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano Real 
   Time Application Interface. 
   All information available at the RTAI web site http://www.aero.polimi.it/projects/rtai/.

   All information available at the RTLinux web site http://www.rtlinux.org.

   SynDEx. 
   All information available at the SynDEx web site http://www-rocq.inria.fr/syndex

[4] MOTOROLA SEMICONDUCTOR TECHNICAL DATA. 

   PRODUCT-Qt/Embedded. 
   All information available at the TROLLTECH web site 

   The XFree86 Project,Inc. 
   All information available at the XFree86 web site 
   http://www.xfree86.org