SynDEx for Real-Time Applications Implementations

Pierre Pomiers, Antonella Semerano
Robosoft S.A., Technopole d'Izarbel, Bidart, France

ABSTRACT

This article discusses and brings solutions to the wide range of real-time applications implementations. Specially, it discusses how a new real-time development tool, named SynDEx\(^1\), improves the state of the art. Relying on the AAA\(^2\) methodology, SynDEx is known to be both a gentle tool for distributed architectures handling and a good way of optimizing hardware resources management. SynDEx provides an easy graphical user interface for standalone critical application design and proposes users a set of standard functionalities. When interfacing low-level executives with high-level applications numbers of problems appear. Software making use of network or multimedia resources do not usually fit all hard real-time constraints needed. Through the new "linuxIO_" SynDEx macro, we bring a solution to this category of programs. Thanks to it, interfacing classical C/C++, as well as network distributed programs with SynDEx generated executives becomes possible and ease to handle.

1 SYSTEM OVERVIEW

The "linuxIO_" macros is described for use in a particular context: that is in SynDEx programs driving architectures composed of one Intel x86 Linux/RTAI machines and one or more Motorola MPC555 based control boards\(^3\) with CAN buses as communication media. This section covers both the development and embedded target environment.

1.1 Development environment

The application development method discussed here makes use of SynDEx tools. Developed by INRIA, SynDEx is a graphical interactive software (see Fig. 1) with on-line documentation

---

1 Synchronized Distributed Executive
2 Algorithm Architecture Adequation
3 Boards manufactured by Robosoft as part of his own control system products
(refer to [3]), implementing the AAA methodology. Here is the list of services offered by SynDEx:

- specification of an application algorithm as a conditioned data-flow graph (or inter-face 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 multicomponents with response time optimization
- visualization of predicted real-time performances for the multicomponents sizing
- generation of dead-lock free executives for real-time execution on the multicomponents 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.

![Fig. 1: Application design example using SynDEx CAD](image)

The distributing and scheduling heuristics as well as the predicted real-time diagram, help the user to parallelize his algorithm and to 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.

1.2 Robosoft control environment
The SynDEx development system described above is able to generated executive binaries for various types of target including the ones that compose the Robosoft control platform. Robosoft control architecture typically embeds one or more MPC555 based boards and an Intel x86 Real-time Linux computer.

1.2.1 MPC555 control board
The Robosoft MPC555 based board (see Fig. 2) is a stand-alone four 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 (cadenced at a 40 MHz frequency), an interface for controlling one axis, a set of isolated inputs outputs and all communication ports.

• 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 ad-d custom modules for future device (additional input and output ports, RAM or EEPROM extension).

Refer to Table 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™)
• 50-Channel Timer System: Dual Time Processor Units (TPU3), Modular I/O System (MIOS1)
• 32 Analog Inputs: Dual Queued Analog-to-Digital Converters (QADC64)

Fig. 2: The MPC555 based Robosoft control board

1.2.1 Real-time Linux embedded computer

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 RTOS⁴ on top of Linux standard kernel. It exists two main variants of real-time Linuses available for free:

---

⁴ Real-Time Operating System
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 nor 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.

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.

<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>connectors dedicated to axis control (including 1 analog output per axis)</td>
</tr>
</tbody>
</table>

Table 1: Robosoft control board connectors description
2 THE "LINUXIO_" SYNDEX MACRO MECHANISM

Let us explain "linuxIO_" basic principle. After a short recall about what "linuxIO_" should be used for, we present how it has been implemented and which are the underlying techniques we are making use of.

2.1 "linuxIO_" specification

The use of "linuxIO_" is for 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.

![Fig. 3: User application general principle](image)

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 "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. Fig. 3, 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 Fig. 4. On this picture, we mention that data are ex-changed using shared memory segments. These segments, created by RTAI, are parts 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).

![Fig. 4: High-level application general principle](image)

The type of 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 for details). Thanks to this data sharing method, any high-level Linux process is able to access control board hardware asynchronously (that is without disturbing hard real-time software behavior) and use it from non-SynDEx time consuming computation processes. Among the obvious advantages of this approach the following 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 supports for the hardware architecture (previously defined using SynDEx CAD
Basically, a set of low-level hardware elements (sensors and actuators needed for the application) is handled by one (or more) MPC555 controller boards.

For communication purpose, each MPC555 control board is equipped with 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). From now, let us focus on how low-level application operates. On Fig. 5, 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 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 level 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).

![Fig. 5: Low-level application general principle](image)

**CONCLUSION**

Embedding high quality video feedbacks or interactive control panels is challenging when applications are expected to be real-time compatible. This paper proposes "linuxIO_" SynDEx macro as a solution, widely used in Robosoft products, that respects such constraints.

"linuxIO_" macros is a secured way of interfacing SynDEx generated hard real-time constrained applications with user oriented high-level programs. For instance, GUIs\(^5\) are parts of these so called high-level software. Indeed GUIs should help application oriented users to drive or interact with a given hardware architecture, but it seems to be important that GUIs

\(^5\) Graphical User Interfaces
should not disturb the functioning of the real-time synchronized critical application part. Thus, it appears reasonable to develop GUIs at a user-oriented operating system level. Moreover, this way of proceeding, allows 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. Fig. 6 is a sample shot of a GUI we developed for an automated electrical vehicle application. It shows the GUI displayed on the vehicle LCD panel that provides user access to some hardware signals and reports information coming from various vehicle sensors.

![GUI Application Screenshot](image)

**Fig. 6: Screenshot of a GUI application designed using "linuxIO_"**

**REFERENCE**

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.