A Nonlinear Model-Based Control realized with an Open Framework for Educational Purposes

Klaus Weichinger
BIOE Open Hardware Automation System Developer
3300 Greinsfurth, Austria
snaky.1@gmx.at - http://bioe.sourceforge.net

Abstract

Today, nonlinear model-based control methods are an essential part in different control applications. To provide a complete open framework for educational purposes this contribution extends common open source software (Scilab/Scicos, Maxima and rt-preempt Linux real-time system) with a low-cost do-it-yourself open hardware interface and a web-based monitoring system embedded into Scicos blocks.

The simple concept of the open hardware interface called Bioe (Basic Input Output Elements) allows the real-time application to interact with general analog and digital signals as well as to more complex devices (e.g. resistive touch panels, RC servos, I^2C acceleration sensors). Furthermore, a prototype of a web-based monitoring system using Ajax is treated. It consists of a Http web server embedded into a Scicos block so that existing Scicos code generation packages for rt-preempt Linux can be used without modifications.

To demonstrate the applicability and usability of the proposed framework a nonlinear model-based control law for a mechatronic multi-input multi-output system is derived with the concept of input/output linearization and realized with the proposed open framework.

1 Introduction

Rapid Control Prototyping (RCP) is still associated with cost-intensive hardware and software investments. Such proprietary RPC frameworks are used in industrial applications because they are well known from education.

Mercifully, the real-time capability of the Linux kernel due to patches like Rtai [1] or rt-preempt [2] has highly increased and the usability of cost-saving open source software (OSS) for computer-aided control system design (CACSD) has improved within the last years. A fully usable RCP framework for industrial and educational applications is provided by Rtai-Lab which consists of a Linux distribution with a Rtai patched kernel, Scilab/Scicos [3] or ScicosLab [4] and the Comedi drivers collection [5] for a variety of data acquisition boards. Furthermore, the software xRtaiLab allows to scope and record signals and to edit system parameters. There also exist other real-time Linux projects like RTLinux [6] and Xenomai [7], but the comparison of real-time approaches is not the topic of this paper.

This contribution picks up two aspects that should complete the existing OSS [2, 8, 4, 9] to a cost- and time-saving open RCP framework for rt-preempt Linux real-time systems. The first aspect is discussed in section 2 and concerns the hardware to interface a RCP system with the target system. The presented solution is a low-cost and do-it-yourself system called Basic Input Output Elements (Bioe, [10]) with a simple but very flexible interface. The second aspect treats a web-based monitoring approach that implements a Http web server within a Scicos block and uses an Ajax web-application to scope signals and edit parameters. The web-based concept, first results of the prototype and further details are topic of section 3.

In section 4 the proposed open RCP framework is used to implement a nonlinear model-based control law for a mechatronic system. The multipleinput multiple-output (MIMO) mechatronic system consists of the well known mass-spring system with viscous friction that is actuated with a double-acting hydraulic piston (DAP). Beside the position control of the mass the sum-pressure of the DAP has to be stabilized at a constant value. The method of input/output exact linearization [11, 12] is used to derive the control law and a hardware-in-loop (HIL) simulation is used to test the control law and to demonstrate the usage of the BIOE system and the web-based monitoring system.

Usability and reliability of an open RCP framework are basic requirements so that open hard- and software can be used for educational purposes, a very important precondition to introduce open RCP frameworks into industrial applications.

2 Open Hardware - BIOE

The BIOE system was developed within a diploma thesis [13] and is a simple but effective piece of hardware to interface a PC with signals of a physical system. It consists of small modules that are connected parallel with the BIOE bus cable to the parallel port interface (Line Print Terminal, LPT) of the PC as shown in figure 1.

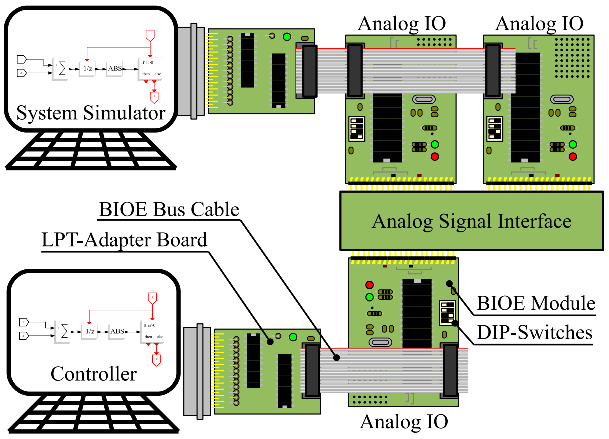


FIGURE 1: *A BIOE mock-up*

Each module can be addressed with a 4bit dip-switch and so up to 16 BIOE modules can be connected to the bus cable. An adapter board is used for signal amplification and to protect the LPT against damages. This interface is used for communication because of its real-time performance and simplicity. In addition, the LPT interface is still available on some embedded and desktop systems or can be retrofitted with e.g. PCI Express cards.

A BIOE module is equipped with an AT-Mega16 microprocessor. This 8bit microprocessor contains the functionality to deal with different types of signals and to provide the information via the BIOE bus by the use of 16 end-points (EP0, EP1, . . . , EP15). Each end-point consists of a 16bit receive and 16bit transmit register ($RX_{adr,ep}$ and $TX_{adr,ep}$ with the BIOE address $adr \in \{0, \dots, 15\}$ and the end-point number $ep \in \{0, \dots, 15\}$). These end-points are the common interface between the real-time application and the physical signal. The end-points are accessed with transactions. During a transaction a 16bit value is written from the PC to the register $RX_{adr,ep}$ and is read from the BIOE module register $TX_{adr,ep}$ back to the PC simultaneous.

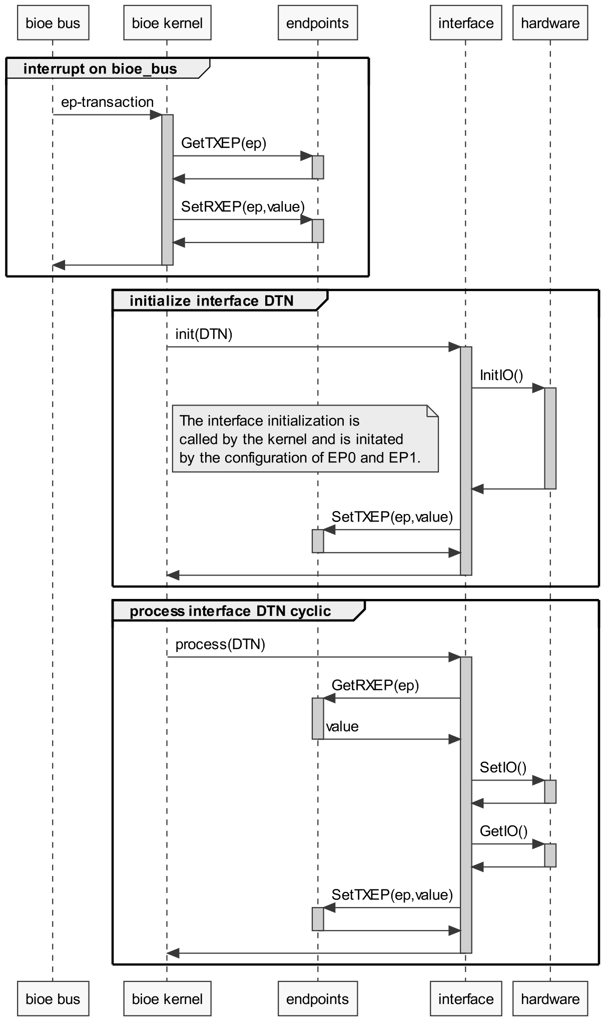


FIGURE 2: *UML sequence diagram from the BIOE module firmware*

In addition, the end-points EP0 and EP1 are used to activate the required interface type with the device type number (DTN). This approach reduces the software effort for the real-time task and so only two SCICOS blocks are required to use the whole functionality of BIOE (see section 2.2). Furthermore, each BIOE module contains all interface types that are realized for BIOE and each interface type can be activated with an unique DTN. After the initialization of the interface the corresponding interface function is processed and the informations are converted and exchanged between the end-points and the hardware interface as illustrated in figure 2.

The table 1 gives an overview of the provided interfaces:

DTN	Interface Type Description
001	16 digital outputs
002	16 digital inputs
003	8 digital inputs, 8 digital outputs
050	two 10bit PWM, four 10bit ADC
090	square signal generator
091	square signal frequency measurement
100	incremental decoder
101	incremental decoder with HCTL2022
105	16x2 LCD driver, four push buttons
106	ADC, PWM, RC-Servomotor and incremental decoder
107	UART interface
120	RC5 infrared receiver
127	4 wire resistive touch interface
128	WII-Nunchuck interface (I^2C)
129	WII-Remote IR camera interface (I^2C)
230	electrical network simulator
231	torsional oscillator simulator
250	cycle time measurement

TABLE 1: *List of available interface types*

If a new interface type is required the implementation is done in the BIOE module software. The concept to keep hardware related functionality on the BIOE modules avoids any modification of the BIOE software on the PC and reduces the efforts to keep the PC software updated. Hence, if the BIOE communication via the end-points is ensured all available DTN interfaces can be used. In the following sections further topics concerning the BIOE system are discussed.

2.1 BIOE Bus and Performance

The BIOE bus is a 14 pole flat cable (see table 2) that connects the modules with the LPT adapter board. This bus is a compromise of low-cost cables and connectors, communication speed and the microprocessor performance. For the communication with transactions a 4 bit parallel bus is used and each transaction contains the device address, the endpoint number, the register values and a very simple check-sum to detect transmission errors.

PIN	*Name*	*Description*
1	V_{cc}	+5 V supply
2	GND	ground supply
3	CS	Chip-Select (by Master)
4	CLK	Clock (by Master)
5	DO_0	Bit 0 Master→Slave
6	DO_1	Bit 1 Master→Slave
7	DO_2	Bit 2 Master→Slave
8	DO_3	Bit 3 Master→Slave
9	DI_0	Bit 0 Slave→Master
10	DI_1	Bit 1 Slave→Master
11	DI_2	Bit 2 Slave→Master
12	DI_3	Bit 3 Slave→Master
13	NC	not in use
14	NC	not in use

TABLE 2: *Description of the 14 pole* BIOE *bus flat cable*

Performance measurements done with different PC's return a transaction time T_{trans} between 25 μs and 50 μs (time where CS is high; see figure 3). This parameter depends on the PC's hardware, especially the type of parallel port (parallel port directly on the motherboard, a PCIe extension module, ...).

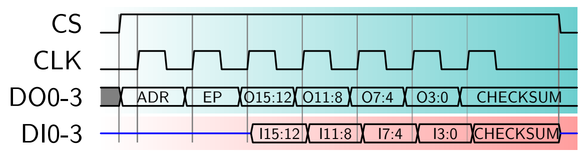


FIGURE 3: *Timing-Diagram of a* BIOE *transaction (O is the value written to $RX_{adr,ep}$, I is the value read from $TX_{adr,ep}$)*

The following example demonstrates the determination of the smallest possible sample time of a control realized with BIOE.

Example 1 *A PID control of a system with one input $n_i = 1$ and one output $n_o = 1$ should be realized with* BIOE. *This fictive example illustrates how to determine the smallest sample time for the control task. For discrete time systems it is assumed that the time delay T_{io} between reading the inputs and setting the outputs is zero. Nevertheless, due to finite calculation speed there is a delay. If a sample time T_s is used so that the relation $T_{io} \leq 0.1\,T_s$ is ensured* *then the effects of the timing error can be neglected. In this example a measurement of the* Bioe *bus resulted in $T_{trans} = 30\,\mu s$.*

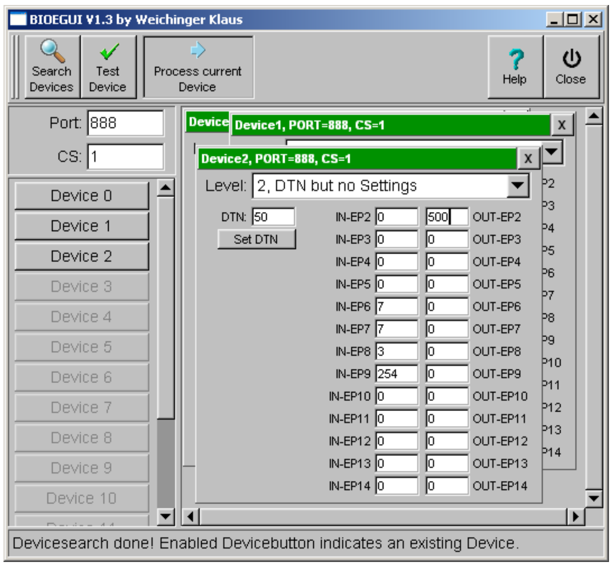
$$T_s = 10\,T_{trans}\,(n_i + n_o) \tag{1}$$

The equation (1) delivers the minimum sample time of $T_s = 0.6\,ms$ for this PID control example.

2.2 Toolchain

The Bioe toolchain consists of a set of Scicos blocks, a command-line tool and a tool with a graphical user interface. A CLisp API is also available. This toolchain uses the Bioe C library which provides the required communication functions and contains the platform abstraction layer.

For the development and testing of Bioe modules and new interface functions the command-line tool `bioedude` and the graphical user interface `bioegui` (figure 4) can be used to configure the modules and to access the end-points directly. These tools are also very useful to test sensors and actuators of a new mock-up. The toolchain can be compiled and used on WIN32 and Linux based platforms.



Port: 888
CS: 1

Device1, PORT=888, CS=1

Device2, PORT=888, CS=1

Level: 2, DTN but no Settings

DTN: 50

IN-EP2	0	500	OUT-EP2
IN-EP3	0	0	OUT-EP3
IN-EP4	0	0	OUT-EP4
IN-EP5	0	0	OUT-EP5
IN-EP6	7	0	OUT-EP6
IN-EP7	7	0	OUT-EP7
IN-EP8	3	0	OUT-EP8
IN-EP9	254	0	OUT-EP9
IN-EP10	0	0	OUT-EP10
IN-EP11	0	0	OUT-EP11
IN-EP12	0	0	OUT-EP12
IN-EP13	0	0	OUT-EP13
IN-EP14	0	0	OUT-EP14

Devicesearch done! Enabled Devicebutton indicates an existing Device.

FIGURE 4: *Screenshot of* `bioegui` *(uses FLTK, a cross-platform C++ GUI toolkit)*

Only two Scicos blocks (figure 5 are required to include the Bioe system into a Scicos model. The first block `bioe_dev` does not have inputs and outputs and it initializes the interface type of a Bioe module. The second block `bioe_ep` provides the end-point access. This block can be configured so that it reads or writes an end-point triggered by an external event.

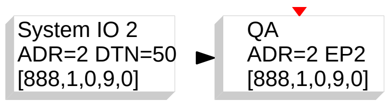
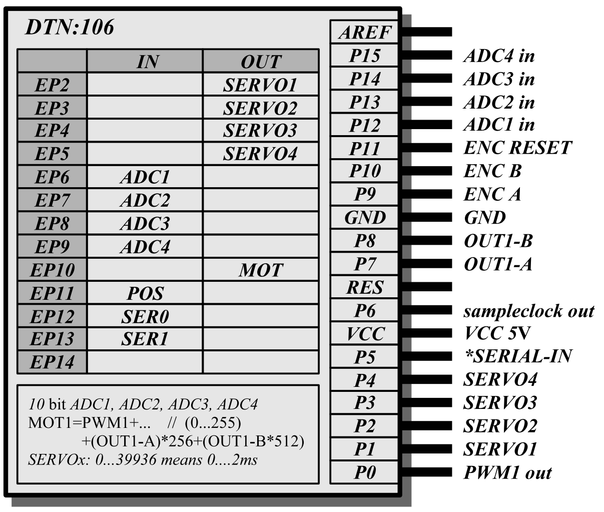


FIGURE 5: Scicos *blocks (left side: device block; right side: end-point block)*

To configure the parallel port the Bioe API requires the base-address of the parallel port (e.g. 888 dec), the channel number (e.g. 1) to define the chip-select number, the speed-parameter (e.g. 0) to trim the communication speed and the pause-parameter (e.g. 9) to avoid CPU stress of the Bioe module in consequence of a permanent Bioe bus communication. Optionally, a fifth parameter is used to define a setting number that loads the port settings from the file `bioesettings.conf` if available. This feature will be required if a compiled Bioe application has to be adapted to a different hardware platform (e.g. an other parallel port base-address).

2.3 An Example Interface



	IN	OUT
EP2		SERVO1
EP3		SERVO2
EP4		SERVO3
EP5		SERVO4
EP6	ADC1	
EP7	ADC2	
EP8	ADC3	
EP9	ADC4	
EP10		MOT
EP11	POS	
EP12	SER0	
EP13	SER1	
EP14		

10 bit *ADC1, ADC2, ADC3, ADC4*
MOT1=PWM1+... // (0...255)
+(OUT1-A)*256+(OUT1-B*512)
SERVOx: 0...39936 means 0....2ms

FIGURE 6: *Illustration of a complex* Bioe *interface*

All interfaces are described similar to DTN106 illustrated in figure 6. The concept of Bioe allows such simple documentation so that the principle of *What You See Is What You Get* (WYSIWYG) fits very well.

3 Web-Based Monitoring

This section presents the prototype of a web-based monitoring system for rt-preempt Linux real-time applications created with Scicos. The idea was to implement a HTTP server within the real-time application, to start the server as non-real-time thread and to exchange signals and parameters between the real-time application and the HTTP server (see section 3.1). A REST-style architecture [14] is used for AJAX based communication in which the signals are exchanged as XML formated byte stream. A web application or an other kind of application uses this interface to establish a connection with the real-time application to scope signals and to modify parameters. For the client side communication and a basic web-based monitoring system see section 3.2.

3.1 Server Side Component

The RTXMLSERVER illustrated in figure 7 was realized within one SCICOS block. During the initialization of this SCICOS block a new POSIX thread is created (see listing 1).

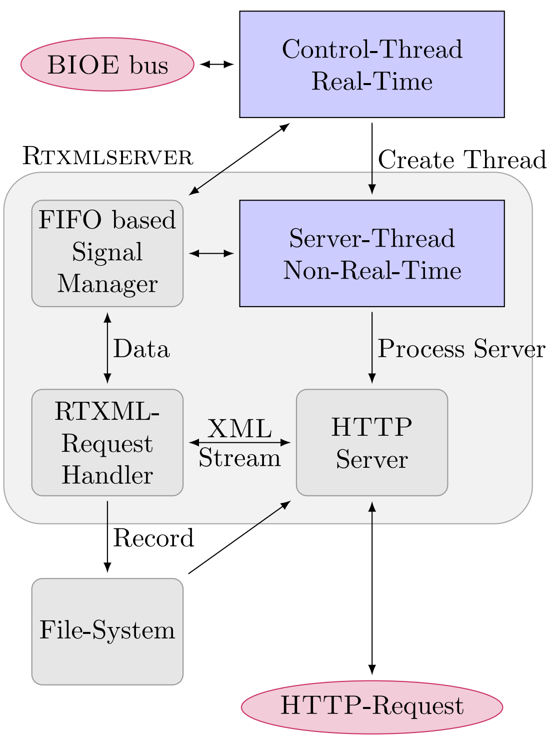


FIGURE 7: *Simplified Structure Diagram of the* RTXMLSERVER

This thread processes the data-exchange between a remote client monitoring system and the real-time task. For the communication between the real-time thread and the server thread a object called RTSignalManager was implemented. The SCICOS signal blocks for the RTXMLSERVER use this manager to register their signals and parameters and during the operation the signal value with time-stamp is passed via thread-safe FIFOs to the RTXMLSERVER. At the moment the RTXMLSERVER provides an access via HTTP requests (a reduced HTTP server is implemented according RFC-2616 [15]; TCP and UDP servers are planed).

```
static pthread_t thrd;

static void *server_task(void *p)
{
  struct sched_param param;
  param.sched_priority = 10;
  if(sched_setscheduler(0, SCHED_FIFO, &param)==-1)
  {
    exit(-1);
  }
  // ... process the server
}

void rtxmlserver_fnc(scicos_block *block, int flag)
{
  if (flag==1) {  /* set output      */ }
  if (flag==2) {  /* get input       */ }
  if (flag==5) {  /* termination     */ }
  if (flag==4)
  { /* initialisation */
    // ... block operations
    pthread_create(&thrd, NULL, server_task, NULL);
  }
}
```

LISTING 1: *Schematic structure of the* RTXMLSERVER SCICOS *block*

The HTTP server supports GET requests to access the file system and to load a web-page. In addition, a PUT-Request with the URI `/rtxml.pipe` passes the RTXML-Request within the HTTP request message-body to the RTXML-Request-Handler. The handler parses the XML byte stream and does the corresponding actions similar to SAX [16]. The XML formated RTXML-Response is sent back to the client within the message-body of the PUT-Response. The whole description of the RTXML-Requests and RTXML-Responses is not intended within this contribution but sub-section 3.3 should give an idea about the structure of the XML formated messages.

The concept to start a non-real-time thread within a SCICOS block (see listing 1) can be applied for other purposes like image processing. In this case, the function to get the image from a camera and to do the image processing can be done within a non-real-time task. The results are passed to the real-time thread. This approach reduces the effort to start the application and to share the data with inter-process communication.

3.2 Client Side Application

The prototype of a client side monitoring system shown in figure 8 is an AJAX based web-application using JavaScipt and jQuery (see [17]). AJAX (with jQuery) and the HTTP request method PUT are used to establish and configure a connection with the RTXMLSERVER and to exchange the signals and parameters. This functionality is encapsulated within the JavaScript RTXMLCLIENT class that prepares and stores the data in arrays. These arrays can be used for a visualization purposes, e.g., realized with a JavaScript plotting library to illustrate the signals continuously without refreshing the whole web-page. The library jQuery UI [17] was used to improve the look and feel of the visualization and to get a behavior comparable with a native application.

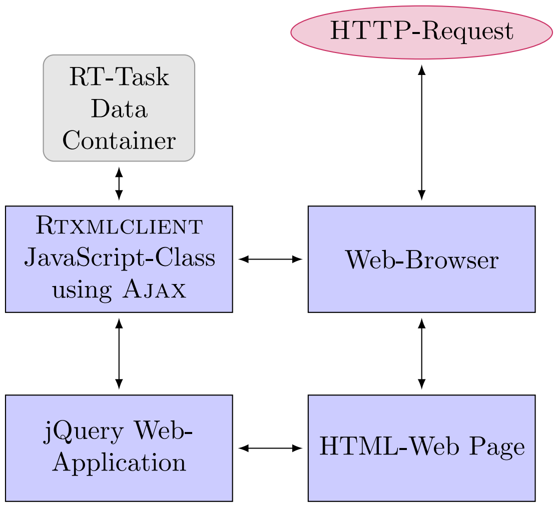
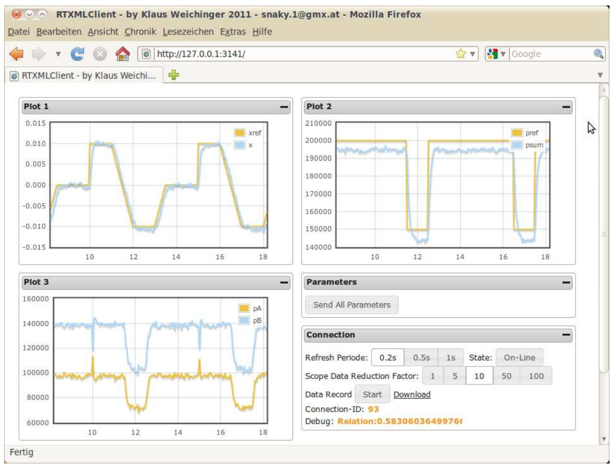


FIGURE 8: *Simplified Structure Diagram of the web-based application using* RTXMLCLIENT

Figure 9 shows the screenshot of the standard interface. It allows the configuration of the connection. If the connection to the real-time application is established, the three signal scopes and the parameter list are automatically configured. This standard interface consists of a HTML file, some CSS and JavaScript files. A custom interface can be created with a text editor and without the need of further compilers or a complex toolchain. This approach allows to prepare a demonstration system mock-up with a specialized interface for presentation purposes or to create laboratory setups where the students can configure the control-law parameters and test the performance of the control-law.

The first prototype works very well and can be used for applications, but if a high sample-rate and more signals are transmitted it is necessary to reduce the number of transmitted data-points. Otherwise, there is not enough CPU power to parse the XML formated context with JavaScript. The reduction of data-points has no significant effect on the quality of the visualization. To provide data records with all sampled informations for offline analysis the RTXMLSERVER can record all data-points into a local `*.csv` file and this file can be downloaded with the web browser.


RTXMLClient - by Klaus Weichinger 2011 - snaky.1@gmx.at - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://127.0.0.1:3141/
RTXMLClient - by Klaus Weichi...
Plot 1
xref
x
Plot 2
pref
psum
Plot 3
pA
pB
Parameters
Send All Parameters
Connection
Fertig

Refresh Periode: 0.2s 0.5s 1s State: On-Line
Scope Data Reduction Factor: 1 5 10 50 100
Data Record Start Download
Connection-ID: 93
Debug: Relation:0.58306036499760

FIGURE 9: *Screenshot of a web-based monitoring example*

3.3 About the RTXML-Request and RTXML-Response

The RTXML-Request in listing 2 contains most of the available functions. With the first XML tag `<C>...</C>` the client provides the connection number so that the RTXMLSERVER knows that it is the same client. Then some action commands are sent to the RTXMLSERVER and at the end the parameter with the id 6 is set with the value 2.44.

```
<R>
  <!-- Connection-Number -->
  <C>69</C>
  <!-- Actions: -->
  <A><N>GETPARAMETERS</N></A>
  <A><N>GETSIGNALS</N></A>
  <A><N>STARTRECORD</N><V>data.csv</V></A>
  <!-- set parameter 6 to 2.44 -->
  <P><I>6</I><V>2.44</V></P>
</R>
```

LISTING 2: *An example of a RTXML-Request*

The RTXML-Responses have a similar format (see listing 3). The RTXMLSERVER collects all signals of one sample time, creates a list that contains the time (double precision), signal id and signal value (double precision), converts this list to a HEX-ASCII string and places it with XML tags into the RTXML-Response.

The main idea was to keep all information in double precision so that there is no information lost due to truncation. The HEX-ASCII string yields to a 100% increase of the bytes to transmit. In the future the use of Base64 (RFC4648, [18]) coding is planed because this increases the bytes to sent only about 33%. Furthermore, it should be checked if the XML parsing can be improved.

```
<R>
  <!-- Connection-Number -->
  <C>69</C>
  <!-- Provide Signals and Parameters -->
  <P><I>5</I><N>Param1</N></P>
  <P><I>6</I><N>Param2</N></P>
  <S><I>1</I><N>Input</N></S>
  <S><I>2</I><N>Output</N></S>
  <!-- Signal Samples Hex-Formated -->
  <!-- time,id1,signal1,id2,signal2,..-->
  <S><V>AF43.....4B2C</V></S>
  <S><V>AF43.....4B3A</V></S>
</R>
```

LISTING 3: *An example of a RTXML-Response*

4 Application Example

To demonstrate the proposed open RCP framework this chapter is devoted to the position and sum-pressure control of a non-linear hydraulic system by the use of exact input-output linearization [11, 12]. The hydraulic system, realized with a PC and BIOE for HIL simulations, is controlled by an other PC. Both PCs use Linux with a rt-preempt patched kernel to provide real-time capability. The signals like position, pressure and volume flow are exchanged between the PCs via an analog interface and BIOE (see figure 1). According to [19, 20, 21], the mathematical model of the system illustrated in figure 10 can be described with the set of ordinary differential equations

$$\begin{aligned}
\dot{x} &= v \\
\dot{v} &= \frac{1}{m}\left[A\,p_A - \alpha\,A\,p_B - d\,v + c\,(L - L_0 - x)\right] \\
\dot{p}_A &= \frac{E_f}{V_{A0} + A\,x}\,(Q_A - A\,v) \\
\dot{p}_B &= \frac{E_f}{V_{B0} - \alpha\,A\,x}\,(Q_B + \alpha\,v\,A)\ ,
\end{aligned} \tag{2}$$

where $\dot{\square}$ denotes the total time derivative, A and $\alpha\,A$ the piston areas, m the mass, E_f the isothermal bulk modulus of the fluid (see, e.g. [21]), V_{A0} and V_{B0} are the initial volumes, c the spring coefficient and d the viscous friction coefficient. The piston position x and the pressures p_A, p_B are measurable states of the system and it is assumed that the velocity v of the piston can not be measured. The volume flows Q_A and Q_B are the inputs of the system. The length L is a geometrical parameter of the system and L_0 is the unstressed length of the spring. Here and in the following let us assume for simplicity $L = L_0$.

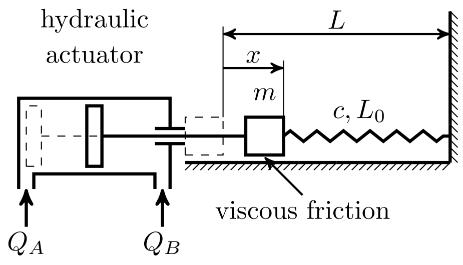


FIGURE 10: *Schematic of the hydraulic system*

4.1 Control Law Design

The design goal of this application is to keep the DAP position x and the sum-pressure $p_A + \alpha\,p_B$ as close as possible to the desired values x_d and p_d. The control deviation can be formulated with

$$e_1 = x - x_d \qquad e_2 = p_A + \alpha\,p_B - p_d\ .$$

With the concept of exact input-output linearization it can be shown that the inputs of the system appear at $\dddot{e}_1$ and $\dot{e}_2$ and in consequence the system is input-state linearizable. To keep these outputs as close as possible to zero the dynamics

$$0 = \dddot{e}_1 + \alpha_{12}\,\ddot{e}_1 + \alpha_{11}\,\dot{e}_1 + \alpha_{10}\,e_1 \tag{3}$$
$$0 = \dot{e}_2 + \alpha_{20}\,e_2 \tag{4}$$

are chosen with a set of parameters α_{10}, α_{11}, α_{12} and α_{20} so that the dynamics (3) and (4) are asymptotically stable and have the required behavior. In the next step the control law

$$\begin{aligned}
Q_A &= Q_A(x, v, p_A, p_B, x_d, \dot{x}_d, \ddot{x}_d, \dddot{x}_d, p_d, \dot{p}_d) \\
Q_B &= Q_B(x, v, p_A, p_B, x_d, \dot{x}_d, \ddot{x}_d, \dddot{x}_d, p_d, \dot{p}_d)
\end{aligned} \tag{5}$$

can be evaluated by solving the equations (3) and (4) for the volume flows Q_A and Q_B. The time derivatives $\dot{x}_d$, $\ddot{x}_d$, $\dddot{x}_d$ and $\dot{p}_d$ can be provided by a trajectory planning system and this allows a feed-forward tracking control of the system (2). Let us assume that the desired trajectories $x_d(t)$ and $p_d(t)$ are smooth and changes very slow and so the choice $\dot{x}_d = \ddot{x}_d = \dddot{x}_d = \dot{p}_d = 0$ is justifiable. Consider that the control law (5) depends on the piston velocity v which can not be measured. Hence, a velocity and disturbance observer is designed in the next step.

4.2 Observer Design

To implement the control law (5) a velocity observer can be used. The realized HIL mock-up (see figure 1) was realized with a minimum of effort and the signals have small offsets. The pressures p_A and p_B are very sensitive for signal-offsets because a wrong measured pressure deals like an external disturbance force and this irritates a reduced observer for the velocity v. To fix this problem the system (2) is extended by an unknown but constant disturbance force F_d with the dynamic $\dot{F}_d = 0$.

The following observer design (see, e.g., [12]) treats the general system

$$\begin{aligned}\dot{x} &= v \\ \dot{v} &= \frac{1}{m}\,(-c\,x - d\,v + F + F_d) \\ \dot{F}_d &= 0\end{aligned}$$

with the position x, the velocity v and the disturbance force F_d. To get the equations for the DAP system $F = A\,p_A - \alpha\,A\,p_B$ has to be used. The position x and the force F can be determined by measurements and calculations and so a reduced observer is designed to estimate the velocity v and the disturbance force F_d. In the first step the following state transformation

$$w_1 = v + k_v\,x \qquad\qquad w_2 = F_d + k_{F_d}\,x$$

is introduced and results in the linear and time invariant system

$$\dot{w} = A_{obs}\,w + u \tag{6}$$

with the new state vector $w = [w_1\; w_2]^T$, the matrix

$$A_{obs} = \begin{bmatrix} k_v - \frac{d}{m} & \frac{1}{m} \\ k_{F_d} & 0 \end{bmatrix}$$

and the input

$$u = \begin{bmatrix} \left(d\,k_v - k_{F_d} - c - k_v^2\,m\right)\,\frac{x}{m} + \frac{F}{m} \\ -k_v\,k_{F_d}\,x \end{bmatrix}\,.$$

With a specific choice of parameters k_v and k_{F_d} the matrix A_{obs} becomes a Hurwitz matrix and then the standard approach of a trivial observer for the system (6) can be applied to get an estimation $\hat{w}$ of w, where $\hat{\Box}$ indicates the estimated value of $\Box$.

$$\dot{\hat{w}} = A_{obs}\,\hat{w} + u$$

Now it is easy to prove that the estimation error for the velocity v and the disturbance force F_d

$$\begin{aligned}e_v &= v - \hat{v} = w_1 - \hat{w}_1 \\ e_{F_d} &= F_d - \hat{F}_d = w_2 - \hat{w}_2\end{aligned}$$

has the asymptotically stable dynamic

$$\dot{e} = A_{obs}\,e$$

with $e = [e_v\,e_{F_d}]^T$. Nevertheless, the proof of the stability of the observer does not allow for conclusions concerning the stability of the overall system because of the nonlinear system (2).

4.3 Implementation

To realize the HIL simulation Ubuntu 10.04 LTS [22] with the rt-preempt patched 2.6.31-11-rt kernel from the Lucid repository was installed. ScicosLab 4.4.1 [4] with the rt-preempt code generation package [9] is used for the simulation and real-time code generation. The Bioe and Rtxmlserver blocks were installed into the ScicosLab directory structure. A performance measurement of a dummy simulation with a cycle time of $T_s = 2\,\mathrm{ms}$ and a Bioe with DTN250 (see table 1) was performed on an Intel(R) Pentium(R) M 1.4 GHz notebook. During the test duration of two hours the maximum $T_{s,max} = 2.08\,\mathrm{ms}$ and the minimum $T_{s,min} = 1.928\,\mathrm{ms}$ of the cycle time were detected. It should be noted that the measured time variations cover the rt-preempt patched Linux notebook, the Bioe bus and the Bioe element software.

In addition, the rt-preempt code generation package [9] was modified so that a real-time scaling parameter can be used to slow-down or speed-up the HIL simulation. This feature can be very useful in the case of complex systems (e.g. distributed parameter systems) were the calculation of the dynamics can not be performed in real-time. In the case of the DAP system control a real-time scaling factor of three is used to keep the time of the Bioe transactions and the settling time of the analog interface circuit small compared to the cycle time. In addition, RC low-pass filters with $R = 1\,\mathrm{k}\Omega$ and $C = 1\,\mu\mathrm{F}$ were used to convert the digital PWM signals ($f_{PWM} = 15.6\,\mathrm{kHz}$) into analog signals.

For the numerical simulation and HIL simulation Scicos blocks of the systems (2) and (5) are required. For this job a Maxima toolbox was developed that allows to export a system

$$\dot{x} = f(x, u, p) \tag{7}$$

$$y = g(x, u, p) \tag{8}$$

into a ready to use Scicos block. A Scicos block normally consists of an interface-function and a computational-function (see, e.g. [23]). The interface-function is written in the Scilab language and provides an interface to edit the system parameters p and the initial values of the state x. The computational-function can be written in Scilab or C and contains the set of ordinary equations

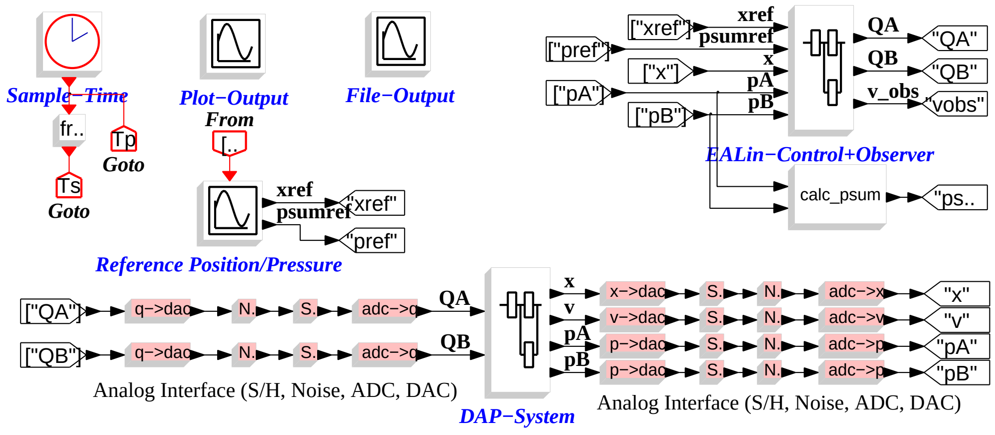


FIGURE 11: *Numerical simulation of the DAP control realized with* Scicos *($T_p = 1\,ms$, $T_s = 5\,ms$)*

$f(x, u, p)$ with the system input u and the equations $g(x, u, p)$ for the system output y.

Such a code-generation package is an important component of a complete RCP framework because it avoids the repeated implementation of equations, the search for typing errors and it is easy to keep the analytic calculations and the simulation code synchronized. The created blocks for the DAP system and the control law can be used for numerical simulations and the real-time code generation with Scicos.

To obtain realistic results the numerical simulation in figure 11 takes effects like quantization, signal ranges and noise into account. The numerical results fit very well with the HIL measurements and they are discussed in the following section.

4.4 Results

Following system parameters were used for the numerical and HIL simulation: $m = 1\,\text{kg}$, $d = 1\,\frac{\text{N s}}{\text{m}}$, $c = 1\,\frac{\text{N}}{\text{m}}$, $A = 1 \cdot 10^{-4}\,\text{m}^2$, $\alpha = 0.7$, $V_{A0} = 5 \cdot 10^{-6}\,\text{m}^3$, $V_{B0} = 3.5 \cdot 10^{-6}\,\text{m}^3$, $E_f = 1.6 \cdot 10^5\,\frac{\text{N}}{\text{m}^2}$. Some of the parameter values are not realistic because they were chosen so that bad influences caused by noise, quantization, offsets and signal limitations due to the low-cost analog interface will be reduced. The eigenvalues of the observer dynamic matrix A_{obs} are placed with the coefficients $k_v = -3$ and $k_{F_d} = -4$ to -2. Finally, the poles of the error dynamic (3) are defined with $\alpha_{12} = 90$, $\alpha_{11} = 2700$, $\alpha_{10} = 27000$ at -30 and the pole of the error dynamic (4) is aligned with $\alpha_{20} = 10$ to -10 .

The results are summarized in figure 12. The first two plots illustrates the tracking behavior of the piston position x and the sum-pressure p_{sum} control. The desired trajectories $x_d(t)$ and $p_d(t)$ are also changed dramatically in order to obtain the step response of the system. In the third plot the volume-flow Q_A into the DAP are compared and the fourth plot compares the real piston velocity (with noise) and the estimated velocity from the numerical simulation.

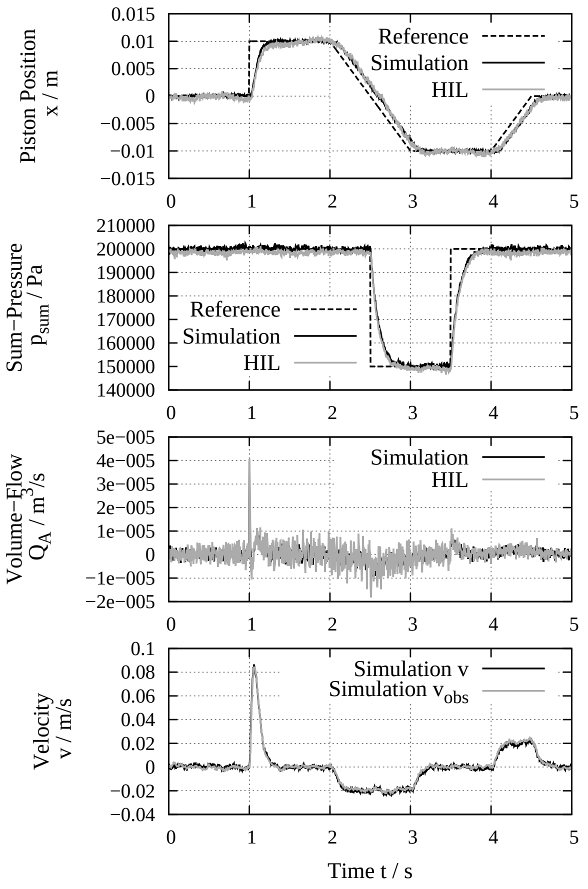


FIGURE 12: *Comparison of the numerical simulation results and the HIL measurements*

5 Conclusions and Perspectives

The HIL simulation of an industrial motivated control demonstrated that rt-preempt patched Linux kernels, ScicosLab, Maxima, the presented open hardware Bioe and the web-based monitoring system prototype build a complete open rapid control prototyping framework which can be used for educational purposes. Due to the web-based approach there is no additional software except a modern webbrowser required to interact with the real-time system. Furthermore, the web interface can be modified to the needs of the application by editing the web pages with a single text editor.

Improvements of the web-based monitoring system and the release of a ready to use version for the community with more human interface demos are planed. An other point of interest is to use this framework to realize a distributed parameter system benchmark example and to compare the results with measurements. Therefore, more complex Bioe interfaces will be used and the realization of 2D/3D visualizations are planed.

References

[1] RTAI - Real Time Application Interface Official Website: *https://www.rtai.org.* Web. 20 Aug. 2011.

[2] Real-Time Linux Wiki: *http://rt.wiki.kernel.org.* Web. 20 Aug. 2011.

[3] Scilab / Scicos Website: *http://www.scilab.org.* Web. 20 Aug. 2011.

[4] ScicosLab 4.4.1 - Project Website: *http://www.scicoslab.org.* Web. 20 Aug. 2011.

[5] COMEDI - Linux Control and Measurement Device Interface - Website: *http://www.comedi.org.* Web. 20 Aug. 2011.

[6] FSM Labs, Inc.: *RTLinux3.1 Getting Started with RTLinux*, 2001.

[7] Xenomai: Real-Time Framework for Linux Website: *http://www.xenomai.org.* Web. 20 Aug. 2011.

[8] Maxima, a Computer Algebra System; Website: *http://maxima.sourceforge.net.* Web. 20 Aug. 2011.

[9] Bucher, R.: *rt-preempt Code Generation Package for ScicosLab http://www.dti.supsi.ch/˜bucher.* Web. 20 Aug. 2011.

[10] Basic Input Output Elements Project Website: *http://bioe.sourceforge.net.* Web. 20 Aug. 2011.

[11] Isidori, A.: *Nonlinear Control Systems, 3rd Edition.* Springer, Londen, UK, 1995.

[12] Schlacher, K.; Zehetleitner, K.: *Control of Hydraulic Devices, an Internal Model Approach In: Analysis and Design of Nonlinear Control Systems.* Springer-Verlag Berlin Heidelbarg, 2008.

[13] Weichinger, K.: *Tonregelung einer Trompete mit einem Low-Cost Automatisierungssystem und RTAI Linux*, JK University Linz, Austria, 2008.

[14] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures.* Dissertation, University of California, Irvine, 2000.

[15] Fiedling, R. T. et al.: *RFC2616: Hypertext Transfer Protocol - HTTP/1.1*, June 1999, http://www.ietf.org/rfc/rfc2616.txt, Web. 20 Aug. 2011.

[16] SAX-Project Website: *http://www.saxproject.org.* Web. 20 Aug. 2011.

[17] jQuery Project Website: *http://jquery.org.* Web. 20 Aug. 2011.

[18] Josefsson, S.: *RFC4648: The Base16, Base32, and Base64 Data Encodings*, October 2006, http://www.ietf.org/rfc/rfc4648.txt, Web. 20 Aug. 2011.

[19] Grabmair, G.; Schlacher, K.; Kugi, A.: *Geometric energy based analysis and controller design of hydraulic acutators applied in rolling mills.* European Control Conference (ECC), Cambridge, 2003.

[20] Kugi, A.: *Non-linear Control Based on Physical Models, volume 260 of Lecture Notes in Control and Information Sciences.* Springer-Verlag, London, 2001.

[21] Murrenhoff, H.: *Grundlagen der Fluidtechnik, Teil 1: Hydraulik.* Shaker Verlag, Aachen, 2007.

[22] Ubuntu Linux Distribution Website: *http://www.ubuntu.com.* Web. 20 Aug. 2011.

[23] Campbell, S.; Chancelier J.; Nikoukhah, R.: *Modeling and Simulation in Scilab/Scicos.* Springer. 2006.