14

The Innovative Design of Low Cost Embedded Controller for Complex Control Systems

Meng Shao[1], Zhe Peng[2] and Longhua Ma[2]
[1]*Computer Centre, Hangzhou First People's Hospital, Hangzhou,*
[2]*School of Aeronautics and Astronautics, Zhejiang University, Hangzhou, China*

1. Introduction

With the availability of ever more powerful and cheaper products, the number of embedded devices deployed in the real world has been far greater than that of the various general-purpose computers such as desktop PCs. An embedded system is an application-specific computer system that is physically encapsulated by the device it controls. It is generally a part of a larger system and is hidden from end users. There are a few different architectures for embedded processors, such as ARM, PowerPC, x86, MIPS, etc. Some embedded systems have no operating system, while many more run real-time operating systems and complex multithreaded programs. Nowadays embedded systems are used in numerous application areas, for example, aerospace, instrument, industrial control, transportation, military, consumer electronics, and sensor networks. In particular, embedded controllers that implement control functions of various physical processes have become unprecedentedly popular in computer-controlled systems (Wittenmark et al., 2002 ; Xia, F. & Sun, Y.X., 2008). The use of embedded processors has the potential of reducing the size and cost, increasing the reliability, and improving the performance of control systems.

The majority of embedded control systems in use today are implemented on microcontrollers or programmable logic controllers (PLC). Although microcontrollers and programmable logic controllers provide most of the essential features to implement basic control systems, the programming languages for embedded control software have not evolved as in other software technologies (Albertos, P. 2005). A large number of embedded control systems are programmed using special programming languages such as sequential function charts (SFC), function block languages, or ladder diagram languages, which generally provide poor programming structures. On the other hand, the complexity of control software is growing rapidly due to expanding requirements on the system functionalities. As this trend continues, the old way of developing embedded control software is becoming less and less efficient.

There are quite a lot of efforts in both industry and academia to address the above-mentioned problem. One example is the ARTIST2 network of excellence on embedded systems design (http://www.artist-embedded.org). Another example is the CEMACS project (http://www.hamilton.ie/cemacs/) that aims to devise a systematic, modular, model-based approach for designing complex automotive control systems. From a technical

point of view, a classical solution for developing complex embedded control software is to use the Matlab/Simulink platform that has been commercially available for many years. For instance, Bucher and Balemi (Bucher, R.; Balemi, S., 2006) developed a rapid controller prototyping system based on Matlab, Simulink and the Real-Time Workshop toolbox; Chindris and Muresan (Chindris, G.; Muresan, M., 2006) presented a method for using Simulink along with code generation software to build control applications on programmable system-on-chip devices. However, these solutions are often complicated and expensive. Automatic generation of executable codes directly from Matlab/Simulink models may not always be supported. It is also possible that the generated codes do not perform satisfactorily on embedded platforms, even if the corresponding Matlab/Simulink models are able to achieve very good performance in simulations on PC. Consequently, the developers often have to spend significant time dealing with such situations. As computer hardware is becoming cheaper and cheaper, embedded software dominates the development cost in most cases. In this context, more affordable solutions that use low-cost, even free, software tools rather than expensive proprietary counterparts are preferable.

The main contributions of this book are multifold. First, a design methodology that features the integration of controller design and its implementation is introduced for embedded control systems. Secondly, a low-cost, reusable, reconfigurable platform is developed for designing and implementing embedded control systems based on Scilab and Linux, which are freely available along with source code. Finally, a case study is conducted to test the performance of the developed platform, with preliminary results presented.

The platform is built on the Cirrus Logic EP9315 (ARM9) development board running a Linux operating system. Since Scilab was originally designed for general-purpose computers such as PCs, we port Scilab to the embedded ARM-Linux platform. To enable data acquisition from sensors and control of physical processes, the drivers for interfacing Scilab with several communication protocols including serial, Ethernet, and Modbus are implemented, respectively. The developed platform has the following main features:

- It enables developers to perform all phases of the development cycle of control systems within a unified environment, thus facilitating rapid development of embedded control software. This has the potential of improving the performance of the resulting system.
- It makes possible to implement complex control strategies on embedded platforms, for example, robust control, model predictive control, optimal control, and online system optimization. With this capability, the embedded platform can be used to control complex physical processes.
- It significantly reduces system development cost thanks to the use of free and open source software packages. Both Scilab and Linux can be freely downloaded from the Internet, thus minimizing the cost of software.

While Scilab has attracted significant attention around the world, limited work has been conducted in applying it to the development/implementation of practically applicable control applications. Bucher et al. presented a rapid control prototyping environment based on Scilab/Scicos, where the executable code is automatically generated for Linux RTAI(Bucher, R.; Balemi, S, 2005). The generated code runs as a hard real-time user space application on a standard PC. The changes in the Scilab/Scicos environment needed to interface the generated code to the RTAI Linux OS are described. Hladowski et al. (Hladowski et al., 2006) developed a Scilab-compatible software package for the analysis

and control of repetitive processes. The main features of the implemented toolkit include visualization of the process dynamics, system stability analysis, control law design, and a user-friendly interface. Considering a control law designed with Scicos and implemented on a distributed architecture with the SynDEx tool, Ben Gaid et al. proposed a design methodology for improving the software development cycle of embedded control systems(Ben Gaid et al., 2008). Mannori et al. presented a complete development chain, from the design tools to the automatic code generation of standalone embedded control and user interface program, for industrial control systems based on Scilab/Scicos (Mannori et al., 2008).

2. Embedded control systems design

In this paper, we develop an embedded controller for complex control applications. The key software used is the Scilab/Scicos package, a free and open source alternative to commercial packages for dynamical system modeling and simulation such as Matlab/Simulink. Since hardware devices are becoming cheaper by the day, software development cost has dominated the cost of most embedded systems. As a consequence, the use of the free and open source software minimizes the cost of the embedded controller. On the other hand, Scilab is a software package providing a powerful open computing environment for engineering and scientific applications. It features a variety of powerful primitives for numerical computations. There exist a number of mature Scilab toolboxes, such as Scicos, fuzzy logic control, genetic algorithm, artificial neural network, model predictive control, etc. All these features of Scilab make it possible, and quite easy, to implement complex control algorithms on the embedded platform we develop in this work.

To satisfy the ever-increasing requirement of complex control systems with respect to computational capability, we use the Cirrus Logic EP9315 ARM chip in this project. The platform runs on an ARM-Linux system. Since Scilab and Scicos were originally developed for general-purpose computers such as desktop PCs, we port Scilab/Scicos to the ARM-Linux platform (Longhua Ma, et al., 2008 ; Feng Xia, et al., 2008). Several interfaces and toolboxes are implemented to facilitate embedded control.

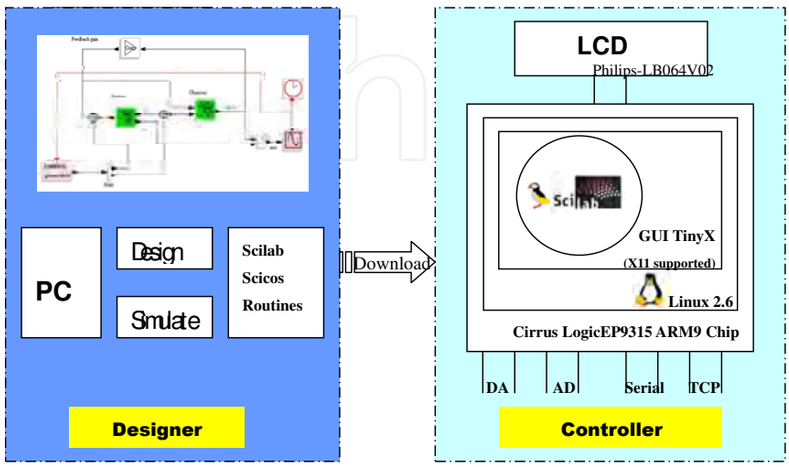


Fig. 1. Design of an embedded controller.

With the developed platform, the design and implementation of a complex control system will become relatively simple, as shown in Figure 1. The main procedures involved in this process are as follows: model, design, and simulate the control system with Scilab/Scicos on a host PC, then download the well designed control algorithm(s) to the target embedded system. The Scilab code on the embedded platform is completely compatible with that on the PC. Consequently, the development time can be significantly reduced.

2.1 Architecture

As control systems increase in complexity and functionality, it becomes impossible in many cases to use analog controllers. At present almost all controllers are digitally implemented on computers. The introduction of computers in the control loop has many advantages. For instance, it makes possible to execute advanced algorithms with complicated computations, and to build user-friendly GUI. The general structure of an embedded control system with one single control loop is shown in Figure 2. The main components consist of the physical process being controlled, a sensor that contains an A/D (Analog-to-Digital) converter, an embedded computer/controller, an actuator that contains a D/A (Digital-to-Analog) converter, and, in some cases, a network.

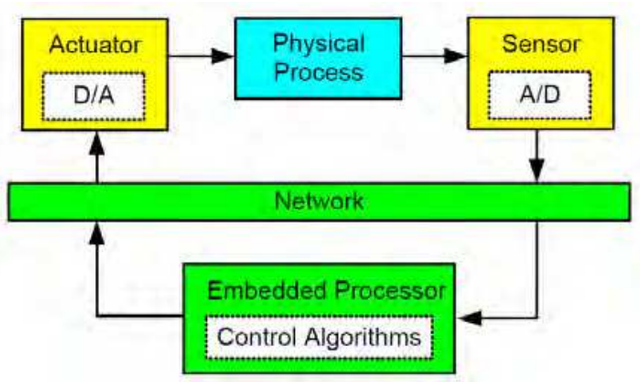


Fig. 2. General structure of embedded control systems.

The most basic operations within the control loop are sensing, control, and actuation. The controlled system is usually a continuous-time physical process, e.g. DC motor, inverted pendulum, etc. The inputs and outputs of the process are continuous-time signals. The A/D converter transforms the outputs of the process into digital signals at sampling instants. It can be either a separated unit, or embedded into the sensor. The controller takes charge of executing software programs that process the sequence of sampled data according to specific control algorithms and then produce the sequence of control commands. To make these digital signals applicable to the physical process, the D/A converter transforms them into continuous-time signals with the help of a hold circuit that determines the input to the process until a new control command is available from the controller. The most common method is the zero-order-hold that holds the input constant over the sampling period. In a networked environment, the sequences of sampled data and the control commands need to be transmitted from the sensor to the controller and from the controller to the actuator, respectively, over the communication network. The network could either be wire line (e.g. field bus, Ethernet, and Internet) or be wireless (e.g. WLAN, ZigBee, and Bluetooth). In a

multitasking/multi-loop environment, as illustrated in Figure 3, different tasks will have to compete for the use of the same embedded processor on which they run concurrently.

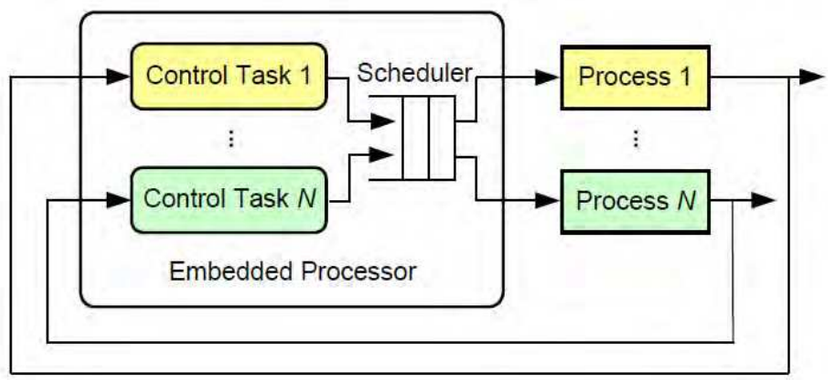


Fig. 3. A multitasking embedded control system.

2.2 Design methodology

There is no doubt that embedded control systems constitute an important subclass of real-time systems in which the value of the task depends not only on the correctness of the computation but also on the time at which the results are available. From a real-time systems point of view, the temporal behavior of a system highly relies on the availability of resources. Therefore, it is compulsory for the system to gain sufficient resources within a certain time interval in order that the execution of individual tasks can be completed in time. Unfortunately, most embedded platforms are suffering from resource limitations, which is in contrast to general-purpose computer systems. There are many reasons behind. For instance, embedded devices are often subject to various limitations on physical factors such as size and weight due to the stringent application requirements. In this context, care must be taken when developing embedded control systems such that the timing requirements of the target application can be satisfied.

Traditionally, the development cycle of a control system consists of two main steps: controller design and its implementation. These two steps are often separated, as shown in Figure 4, where the so-called V-model is given. While the controller design is usually done by control engineers, the implementation is the responsibility of system (software) engineers. In the first step, the control engineers model the physical processes using mathematical equations. According to the requirements specification, the control engineers then design the control algorithms. The parameters of the control algorithms are often determined through extensive simulations to achieve the best possible performance. A widely used tool in this step is Matlab/Simulink that supports modeling, synthesis, and simulation of control systems. In this environment the physical processes are usually modeled in continuous time while the control algorithms are to facilitate digital implementation. In the second step, the software engineers produce the programs executing the control algorithms with the parameters designed in the first step. There are a number of mature programming languages available for the implementation. The system will be tested, possibly many times before the satisfactory performance is achieved.

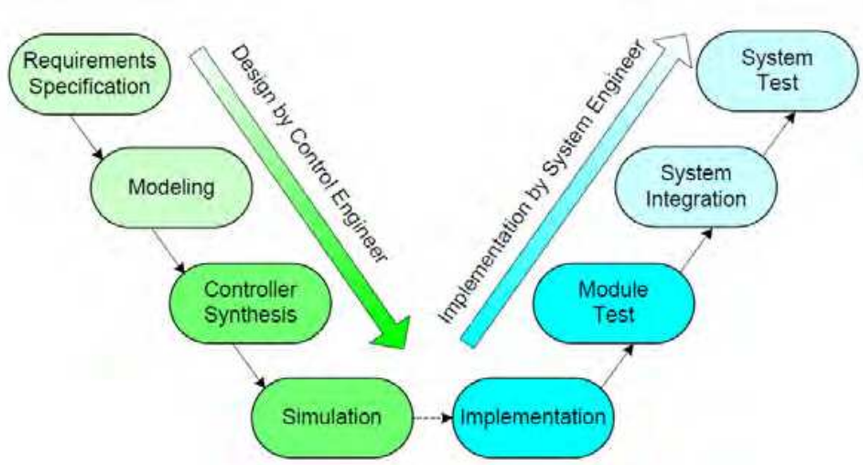


Fig. 4. Traditional development process of control software.

The traditional development process features separation of control and scheduling. The control engineers pay no attention to how the designed control algorithms will be implemented, while the software engineers have no idea about the requirements of the control applications with respect to temporal attributes. In resource-constrained embedded environments, the traditional design methodology cannot guarantee that the desired temporal behavior is achieved, which may lead to much worse-than-possible control performance. Furthermore, the development cycle of a system that can deliver good performance may potentially take a long time, making it difficult to support rapid development that is increasingly important for commercial embedded products.

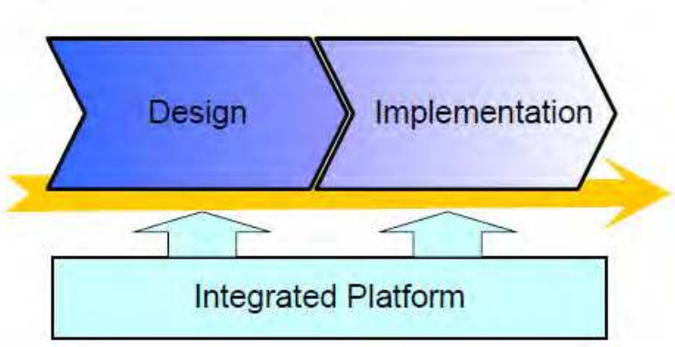


Fig. 5. Integrated design and implementation on a unified platform.

In this paper we adopt a design methodology that bridges the gaps between the traditionally separated two steps of the development process. As shown in Figure 5, we develop an integrated platform that provides support for all phases of the whole development cycle of embedded control systems. With this platform, the modeling, synthesis, simulation, implementation, and test of control software can be performed in a unified environment. Thanks to the seamless integration of the controller design and its implementation, this design methodology enables rapid development of high quality embedded controllers that can be used in real-world systems.

3. Hardware platform

3.1 SoC system

SoC is believed to be more cost effective than a system in package, particularly in large volumes. One of the most typical application areas of SoC is embedded systems. In this work, the processor of SoC is chosen to be the Cirrus Logic EP9315 ARM9 chip, which contains a Maverick Crunch coprocessor. A snapshot of the hardware board is shown in Figure 6.

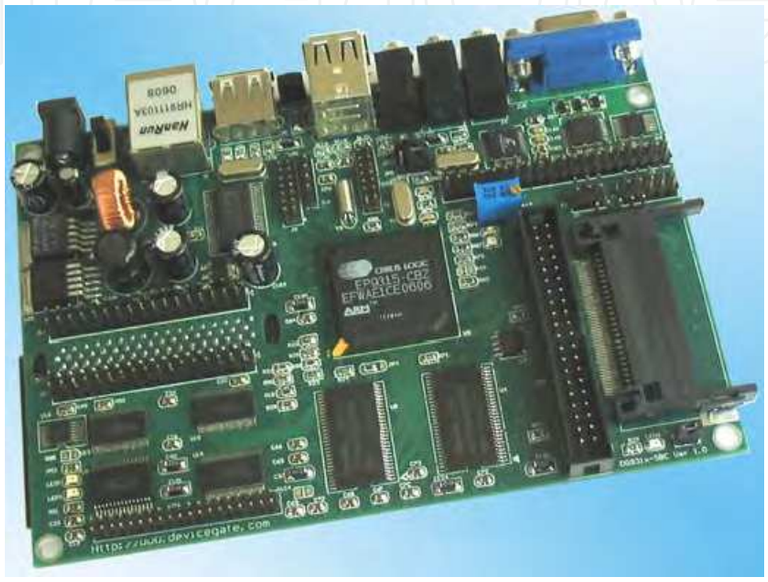


Fig. 6. Hardware platform.

Using this SoC board, it is easy to communicate with other components of the system, for example, to sample data from sensors and to send control commands to actuators, thanks to its support for A/D, D/A, Serial and Ethernet interfaces, etc. To keep the system user-friendly, the embedded controller also includes a LCD with touch screen.

3.2 Maverick crunch coprocessor

The Maverick Crunch coprocessor accelerates IEEE-754 floating point arithmetic and 32-bit fixed point arithmetic operations such as addition, subtraction, multiplication, etc. It provides an integer multiply-accumulate (MAC) that is considerably faster than the native MAC implementation in the ARM920T. The single-cycle integer multiply-accumulate instruction in the Maverick Crunch coprocessor allows the EP9315 to offer unique speed and performance while dealing with math-intensive computing and data processing functions in industrial electronics. The computational speed of the system becomes 10 to 100 times faster when the Maverick Crunch coprocessor is used.

In Table 1 we list the time needed to execute every test function 360,000 times, both with the Maverick Crunch coprocessor and without it. Compared with the case without the Maverick Crunch coprocessor, the computational speed of the system becomes 10 to 100 times faster when the Maverick Crunch coprocessor is used.

Functions	ADD	SUB	MUL	SIN	LOG	EXP
HPF (ms) With Maverick Crunch	1	1	25	950	950	902
SFP (ms) Without Maverick Crunch	187	190	310	7155	7468	6879
Ratio	1:187	1:190	1:12.8	1:7.6	1:7.8	1:7.6

Table 1. Comparison of computational capability of PC and ARM.

The reason of this coprocessor selection is due to its high computation performance compared to normal embedded coprocessor.

4. Software design

There are a number of considerations in implementing control algorithms on embedded platforms including the ARM9 board we use. One of the most important is that embedded platforms are usually limited in resource such as processor speed and memory. Therefore, control software must be designed in a resource-efficient fashion, in a sense that the limited resources are efficiently used.

The key software packages used in this paper includes Linux, TinyX, JWM, Scilab/Scicos, the Scilab SCADA (Supervisory Control and Data Acquisition) toolbox we develop, and other related Scilab toolboxes. The system software architecture is shown in Figure 7. In the following, we detail the software design of the embedded controller.

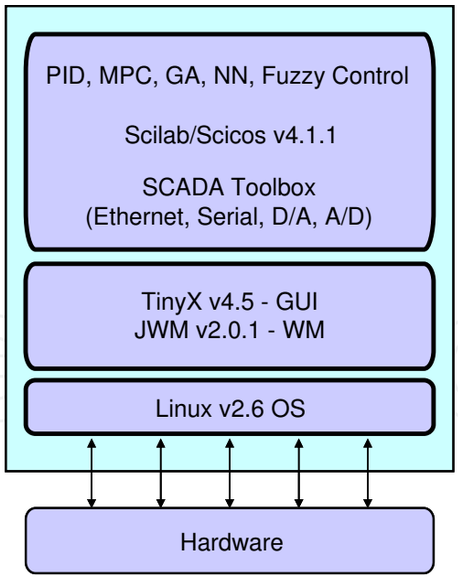


Fig. 7. Software architecture.

4.1 The Scilab/Scicos environment

Scilab is a free and open source scientific software package for numerical computations, which provides a powerful open computing environment for engineering and scientific applications.

It has been developed by researchers from INRIA and ENPC, France, since 1990 and distributed freely and in open source via the Internet since 1994. It is currently the responsibility of the Scilab Consortium, which was launched in 2003. Scilab is becoming increasingly popular in both educational/academic and industrial environments worldwide. Scilab provides hundreds of built-in powerful primitives in the form of mathematical functions. It supports all basic operations on matrices such as addition, multiplication, concatenation, extraction, and transpose, etc. It has an open programming environment in which the user can define new data types and operations on these data types. In particular, it supports a character string type that allows the online creation of functions. It is easy to interface Scilab with FORTRAN, C, C++, Java, Tcl/Tk, LabView, and Maple, for example, to add interactively FORTRAN or C programs. Scilab has sophisticated and transparent data structures including matrices, lists, polynomials, rational functions, linear systems, among others. It includes a high-level programming language, an interpreter, and a number of toolboxes for linear algebra, signal processing, classic and robust control, optimization, graphs and networks, etc. In addition, a large (and increasing) number of contributions can be downloaded from the Scilab website. The latest stable release of Scilab (version 4.1.2) can work on GNU/Linux, Windows 2000/XP/VISTA, HP-UX, and Mac OS.

Scilab includes a graphical system modeler and simulator toolbox called Scicos (http://www.scicos.org), which corresponds to Simulink in Matlab. Scicos is particularly useful in signal processing, systems control, and study of queuing, physical, and biological systems. It enables the user to model and simulate the dynamics of hybrid dynamical systems through creating block diagrams using a GUI-based editor and to compile models into executable codes. There are a large number of standard blocks available in the palettes. It is possible for the user to program new blocks in C, FORTRAN, or Scilab Language and constructs a library of reusable blocks that can be used in different systems. Scicos allows running simulations in real time and generating C code from Scicos model using a code generator. Scilab/Scicos is the open source alternative to commercial software packages for system modeling and simulation such as Matlab/Simulink. Figure 8 gives a screen shot of the Scilab/Scicos package.

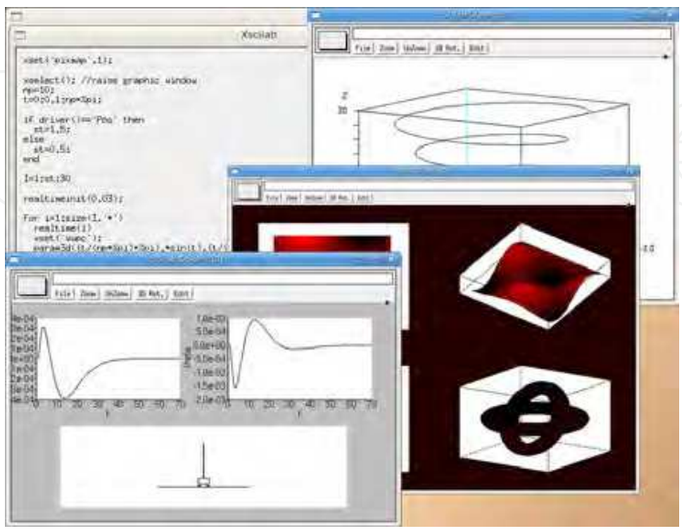


Fig. 8. Scilab environment.

4.2 Software packages

Underneath is the list of the software packages.

- Linux. The developed embedded controller is built on the Linux kernel (www.linux.org). The Linux kernel provides a level of flexibility and reliability simply impossible to achieve with any other operating system such as Windows, UNIX and Mac OS. Mainly for this reason, many embedded systems choose Linux OS.
- TinyX. TinyX is an X server written by Keith Packard. It was designed for low memory environments. On Linux/x86, a TinyX server with RENDER support but without support for scalable fonts compiles into less than 700 KB of text. TinyX tends to avoid large memory allocations at runtime, and tries to perform operations on-the-fly whenever possible. Unlike the usual XFree86 server, a TinyX server is completely self-contained: it does not require any configuration files, and will function even if no on disk fonts are available. All configurations are done at compile time and through command-line flags. It is easy to build user-specified GUI applications with TinyX. More information about TinyX can be found at http://www.xfree86.org.
- Scilab. Scilab/Scicos is utilized in this work to build the development environment for control software executing control algorithms. Developed initially by researchers from INRIA and ENPC, France, since 1990, Scilab is currently a free and open source scientific software package for numerical computations. Scilab has many toolboxes for modelling, designing, simulating, implementing, and evaluating hybrid control systems. It is now used in academic, educational, and industrial environments around the world. Scilab includes hundreds of mathematical functions with the possibility to add interactively programs from various languages, e.g., FORTRAN, C, C++, and Java. It has sophisticated data structures including, among others, lists, polynomials, rational functions, and linear systems, an interpreter, and a high level programming language, i.e., the Scilab language.
- Scicos. Although it is possible to model and design a hybrid dynamical system through writing scripts using the primitives of the Scilab language, this is often time consuming and the developers are prone to insert bugs during the manual coding. To simplify this task, Scilab includes a graphical dynamical system modeller and simulator toolbox called Scicos. Scicos can be used for applications in control, communication, signal processing, queuing systems, and study of physical and biological systems, etc. Using the Scicos graphical editor, it is possible to model and simulate hybrid dynamical systems by simply placing, configuring, and connect blocks. To achieve complete integration with Scilab, easy customization, and the maximum flexibility, most of the Scicos GUIs are written in the Scilab language.
- Scilab SCADA toolbox. To facilitate data acquisition and control operations, we develop the Scilab SCADA toolbox that interfaces Scilab with several kinds of I/O ports including serial port, Ethernet, and Modbus on the embedded Linux system. These communication interfaces make it possible to connect the embedded controller with other entities in the system, e.g., sensors, actuators, and the controlled physical process, using various communication mechanisms/networks. In a complex, possibly large-scale, control system in industry, a huge amount of data, e.g. system output samples and control commands, will be produced during run time. These data usually has to be stored in order to provide support for, e.g., historical data query and higher-layer system optimization. To meet this requirement, we develop the interface to MySQL

database in the Scilab SCADA toolbox. In addition, to provide a standard-compatible solution for the industrial control field, the Scilab SCADA toolbox conforms to the OPC (OLE for Process Control) standard. OPC is a widely accepted industrial communication standard that enables the exchange of data between multi-vendor devices and control applications. It helps provide solutions that are truly open, which in turn gives users more choices in their control applications. The interoperability between heterogeneous entities is assured through the support for non-proprietary specifications. A GUI of the OPC toolbox we develop is shown in Figure 9. With this OPC interface, it is possible to use Scilab as the core control software, and the communications with other (third-party) hardware devices and software tools will be effortless. These help to fully exploit the powerful functionalities of Scilab in complex control applications.

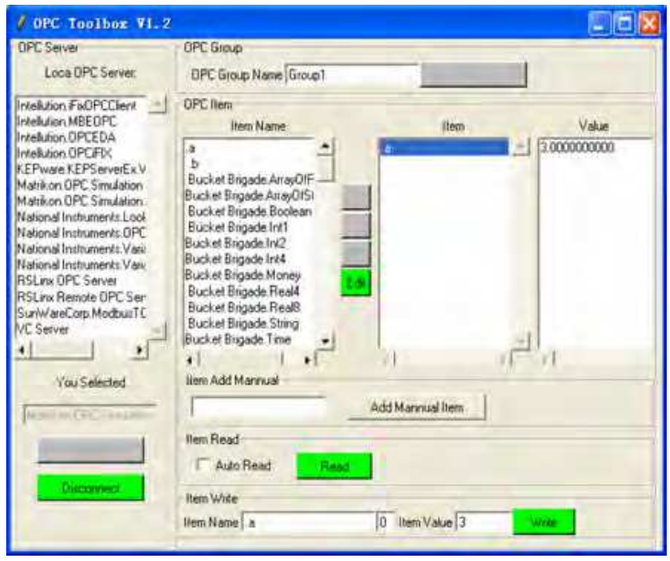


Fig. 9. OPC interface.

4.3 Building cross-compilation tool chain

A cross compiler is a compiler that is able to create executable code for a platform other than the one on which it is run. The basic role of a cross compiler is to separate the build environment from the target environment. This will be particularly useful for the development of the embedded controller based on Scilab/Scicos, which typically works in a general purpose computing environment other than the embedded platform. To port related software packages from PC to the ARM-Linux system, it is essential to build the cross compilation tool chain environment first. There exist several approaches to setting up a cross-compilation tool chain. In this work, we build the cross compiler for the ARM-Linux system using the build root toolkits. Build root is a set of Makefiles and patches that allow to easily generating both a cross-compilation tool chain and a root file for the target system. The cross compilation tool chain makes use of uClibc, a tiny C standard library. Several tools, such as bison, flex, and build-essential, are also exploited. It is worth mentioning that the g77 compiler option should be enabled during this process. Since most of the Scilab code is written in FORTRAN, the g77 compiler is necessary when compiling Scilab.

4.4 Porting Scilab/Scicos to ARM-Linux

Scilab/Scicos was originally designed for PC-based systems but not embedded ARM-Linux systems. Therefore, it is necessary to port Scilab/Scicos onto the embedded platform. Since the majority of core codes of Scilab are written in FORTRAN, we first build a cross-compiler for g77 in order to support cross-compilation of GUI, for example. The GUI system of Scilab/Scicos is based on X11, and therefore the X11 server TinyX is included. To reduce runtime overheads, we optimize/modify some programs in Scilab/Scicos. We have successfully ported Scilab/Scicos to the ARM-Linux system (see Figure 14). To achieve this goal, a number of files in Scilab and Linux have been modified. The main tasks involved in this process are as follows:

- Port Linux to the ARM platform;
- Port TinyX to ARM-Linux;
- Port JWM to ARM-Linux;
- Port Scilab/Scicos to ARM-Linux;
- Configure and optimize the embedded Scilab/Scicos.

The more details of how to porting Scilab/Scicos can be found at Book The embedded ARM-Linux computation develop based Scilab(Ma Longhua, Peng Zhe, 2008).

4.5 Software programming

Once all the necessary software packages are ported to ARM Linux, programming with Scilab in the embedded ARM Linux environment will be the same as on a PC. In this section we address some key issues closely related to embed software programming using Scilab in the ARM Linux platform. Scilab supports numerous data types, such as list, matrix, polynomial, scalar, string, and vector, among others. The syntax is designed to be natural and easy to use. The basic data type is a matrix. All basic operations on matrices, e.g., addition, multiplication, concatenation, and extraction, are provided by means of built-in functions. Scilab can also handle more complex objects such as polynomial matrices and transfer matrices. The syntax for manipulating these matrices is identical with that for constant matrices. This powerful capability of Scilab to handle matrices makes it particularly useful for systems control and signal processing. For instance, it is easy to obtain a natural symbolic representation of complicated mathematical objects such as transfer functions, dynamic systems, and graphs.

In addition, the Scicos toolbox allows users to model and simulate the dynamics of complex hybrid systems using a block-diagram graphical editor. Scilab is composed of three main parts: an interpreter, libraries of functions and libraries of FORTRAN and C routines. It provides an open programming environment in which users can easily create new functions and libraries of functions. In Scilab, functions are treated as data objects. As a consequence, they can be created and manipulated as other data objects. For instance, it is possible to define and/or treat a Scilab function as an input or output argument of other functions. In particular, Scilab supports a character string data type allowing for on-line creation of functions. Scilab has a high level programming language, i.e., the Scilab language. It can be easily interfaced with external FORTRAN or C programs by using dynamic links, or by building an interface program. Dynamic links can be realized using the link primitive. The linked routine can then be interactively called by the call primitive, which transmits Scilab

variables to the linked program and transforms back the output parameters into Scilab variables. In the next section, we will use this technique in developing the interfaces to hardware devices. The interface program can be produced by intersci, which is a built-in Scilab program for building an interface file between Scilab and external functions. It describes the routine called and the associated Scilab function. In addition, the interface program can also be written by the user using mexfiles. With an appropriate interface, it is possible to add a permanent new primitive to Scilab through making a new executable code for Scilab. In addition to the Scilab language and the interface program, Scilab includes hundreds of powerful primitives in the form of mathematical functions. A large number of toolboxes for simulation, control, optimization, signal processing, graphics and networks, etc., are also available. These built-in functions and toolboxes allow users to program software with ease. Figure 10 gives an example of Scilab scripts in which a PID controller is implemented. In this program, GetSample() and UpdateState() are user-defined functions, which may be built by exploiting the I/O port drivers to be presented in the next section. The former obtains the sampled data from sensors, while the latter sends the new control command to actuators.

```
Digital PID Controller
//SP: Setpoint; y: System output; u: Control input
//Ts: Sampling period
//Kc, Td, Ti: Controller parameters
mode(-1)
Ts=2; Kc=1; Td=1; Ti=1; SP=1; u=0;
e(1)=0; e(2)=0; i=3;
Ki=Kc*Td/Ti;
Kd=Kc*Td/Ts;
realtimeinit(Ts);
realtime(0);
while 1
y=GetSample();
e(i)=SP-y;
du=Kc*(e(i)-e(i-1))+Ki*e(i)+Kd*(e(i)-2*e(i-1)+e(i-2));
u=du+u;
UpdateState(u);
e(i-2)=e(i-1);
e(i-1)=e(i);
i=i+1;
realtime(i-3);
end
```

Fig. 10. Example of Scilab scripts in which a PID controller.

5. Platform performance & interface

5.1 Rapid prototyping of control algorithms

The use of Scilab makes it easy to model, design, and implement complex control algorithms in the embedded controller developed in this work. Scilab has a variety of powerful

primitives for programming control applications. Additionally, there are several different ways to realize a control algorithm in the Scilab/Scicos environment. For instance, it can be programmed as a Scilab .sci file using the Scilab language, or visualized as a Scicos block linked to a specific function written in FORTRAN or C. In addition, there are an increasing number of contributions that provide support for implementing advanced control strategies in Scilab using, e.g., fuzzy logic, genetic algorithm, neural networks, and online optimization. As a simple example for system modeling and simulation in Scicos, Figure 11 shows a control system for a water tank. The models of the controller and the water tank are highlighted by the dashed and solid rectangles, respectively. The step response of the control system is depicted in Figure 12.

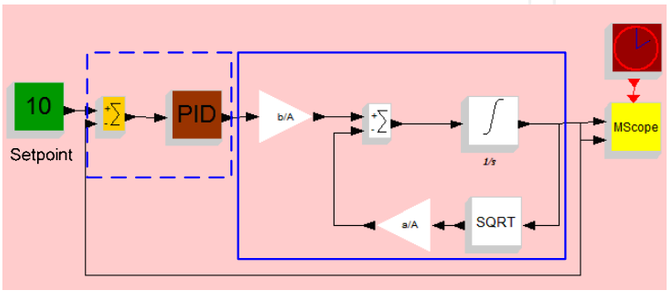


Fig. 11. An example control system in Scicos.

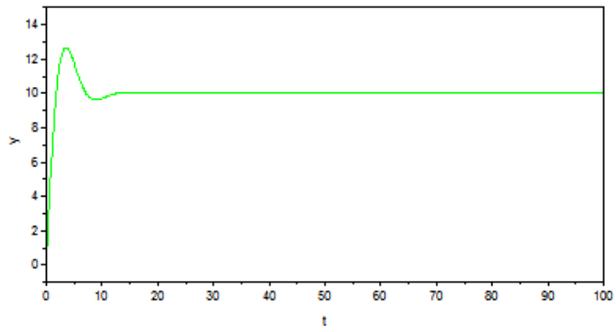


Fig. 12. Step response of the example control system.

5.2 Hardware drivers

Almost all embedded systems in practice need to interact with other related components (i.e. hardware devices) via I/O ports. In order for the developers to build practically useful embedded software with communication ability, it is necessary to provide hardware drivers in the embedded Scilab environment. To address this issue, we have developed the drivers for several types of communication interfaces including serial port, Ethernet, and Modbus. Illustrated below is how to program these drivers using Scilab in ARM Linux, while taking the serial port interface as an example. In the process of communication via a serial port, there are several basic operations, including open connection, set communication parameters, read data, write data, and close connection. Each basic operation is implemented as a separate C function. To facilitate dynamic links with Scilab, all arguments of the C functions are defined as pointers, as shown in the following example figure 13 where the function for reading and writing data from a serial port is implemented.

```
int serialread(int *handle, char *readbuff)
{
int nread;
readbuff[0]='\0';
while((nread=read(*handle,buff,512))>0)
{
printf('\nLen %d\n',nread);
buff[nread]='\0';
strcat(readbuff, buff);
}
}

int serialwrite(int *handle, char *writebuff)
{
int nwrite;
nwrite = write(*handle, writebuff,
strlen(writebuff));
printf('serialwrite%d\n %d\n %d\n', *handle,
nwrite, strlen(writebuff));
if (nwrite==strlen(writebuff))
printf('%d successfully written!\n',
nwrite);
else printf('write error!\n');
}
```

Fig. 13. Example of serial port reading and writing script.

As such, the hardware drivers are implemented as Scilab functions. These functions can be used by Scilab software programs in the same way as using other built-in Scilab functions. The developed hardware drivers, in the form of functions, serve as the gateway linking the different entities. Figure 14 gives a snapshot of the Scilab-based embedded ARM Linux system we develop using the programming techniques described in this Book(Peng, Z, 2008).

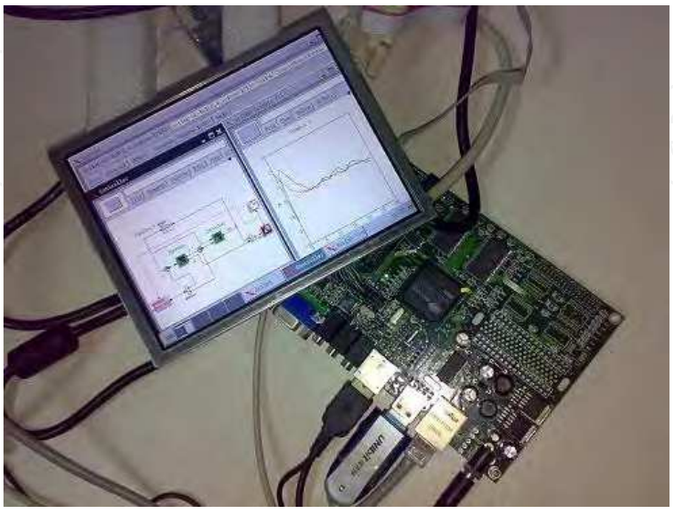


Fig. 14. The embedded control developed.

5.3 Computational capability analysis

Computational capability is a critical attribute of the embedded controller since the execution of the control program affects the temporal behavior of the control system, especially when complex control algorithms are employed. Therefore, we assess the computational capability of the developed embedded controller in comparison with that of a PC (Intel Pentium M CPU @1.60 GHz, with 760 MB of RAM) running Linux. The time for executing different algorithms is summarized in Table 2.

	Rand(800, 800)	DeJoy Algorithm
PC (s)	0.029	3.486
ARM (s)	1.176	92.3
Ratio	1:40	1:30

Table 2. Comparison of computational capability of PC and ARM.

6. Experimental test

In this section, we will test the performance of the developed embedded controller via experiments. For a research laboratory, however, it is very costly, if not impossible, to build the real controlled physical processes for experiments on complex control applications. For this reason, we construct a virtual control laboratory to facilitate the experiments on the embedded controller.

6.1 Virtual control platform

The schematic diagram of the structure of the experimental system is shown in Figure 15. The basic idea behind the virtual control laboratory is to use a PC running a dynamical system modeling software to simulate the physical process to be controlled. The control algorithms are implemented on the embedded controller, which exchanges data with the PC via a certain communication protocol, e.g., serial, Ethernet, or Modbus.

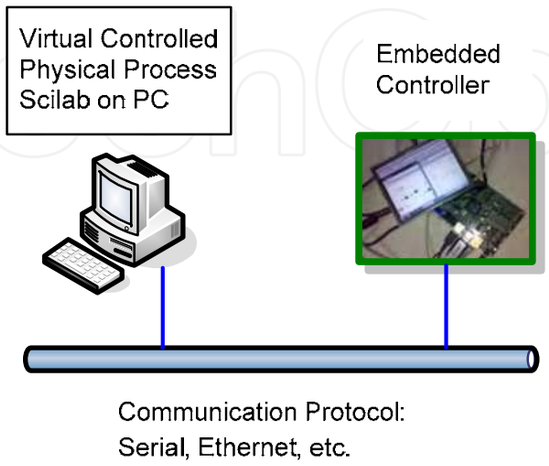


Fig. 15. Experimental system.

Both of the PC and the embedded controller use Scilab/Scicos as core software. Using this virtual control platform, experiments on various (virtual) physical processes are possible given that they can be modeled using Scilab/Scicos.

6.2 Case study

In the following, the control of a water tank is taken as an example for the experimental study. The water tank is modeled as shown in Figure 15 and implemented on the PC (Figure 16). The controller implemented on the embedded controller is shown in Figure 17. The control objective is to keep the water level (denoted y) in the tank to 10. The PC and the embedded controller are connected using Ethernet, and they communicate based on the UDP protocol. The PID algorithm is used for control

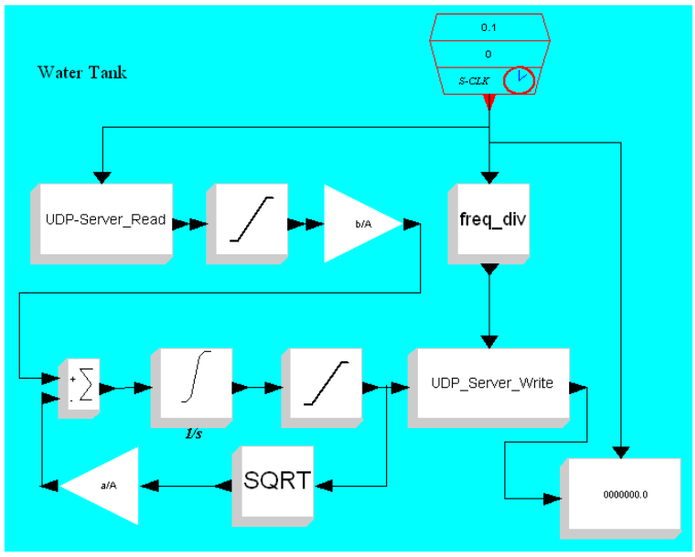


Fig. 16. Controlled process.

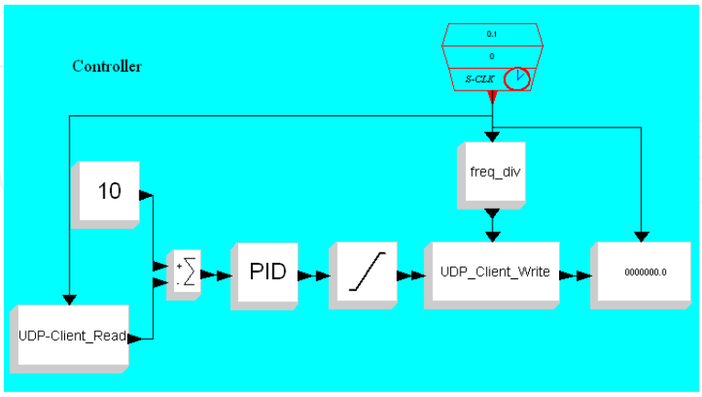
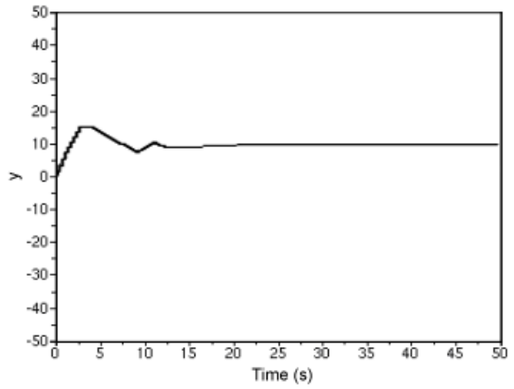
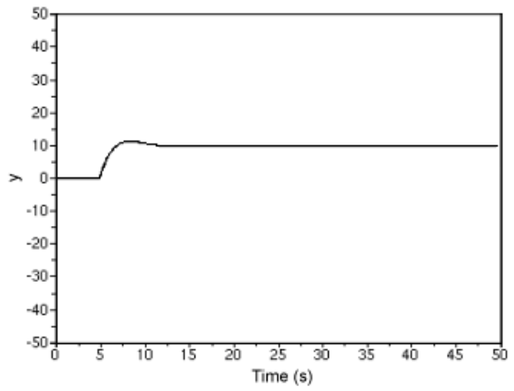


Fig. 17. Controller.

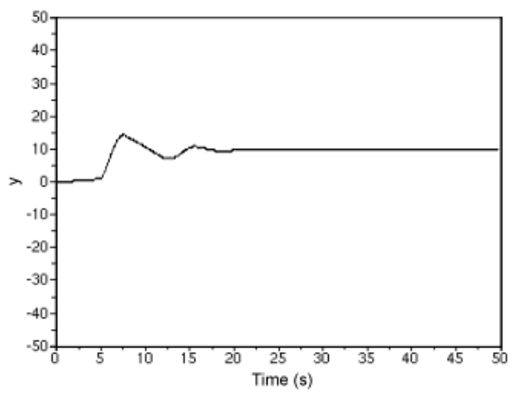
Figure 18 depicts the water level in the tank when different sampling periods are used, i.e., h = 0.1s, 0.2s and 0.5s, respectively. It can be seen that the control system achieve satisfactory performance. The water level is successfully controlled at the desired value in all cases.



(a) h = 0.1s



(b) h = 0.2s



(c) h = 0.5s

Fig. 18. Control performance.

7. Conclusion

We have developed an embedded platform that can be used to design and implement embedded control systems in a rapid and cost-efficient fashion. This platform is built on free and open source software such as Scilab and Linux. Therefore, the system development cost

can be minimized. Since the platform provides a unified environment in which the users are able to perform all phases of the development cycle of control systems, the development time can be reduced while the resulting performance may potentially be improved. In addition to industrial control, the platform can also be applied to many other areas such as optimization, image processing, instrument, and education. Our future work includes test and application of the developed platform in real-world systems where real sensors and actuators are deployed.

8. Acknowledgment

This work is supported in part by Natural Science Foundation of China under Grant No. 61070003, and Zhejiang Provincial Natural Science Foundation of China under Grant No. R1090052 and Grant No. Y108685.

9. References

Albertos, P.; Crespo, A.; Vallés, M. & Ripoll, I. Embedded control systems: some issues and solutions, Proc. of the 16th IFAC World Congress, pp. 257-262, Prague, 2005

Ben Gaid, M.; Kocik, R.; Sorel, Y.; Hamouche, R. A methodology for improving software design lifecycle in embedded control systems, Proc. of Design, Automation and Test in Europe (DATE), Munich, Germany, March 2008

Bucher, R.; Balemi, S. Rapid controller prototyping with Matlab/Simulink and Linux, Control Eng. Pract. , pp. 185-192, 2006

Bucher, R.; Balemi, S. Scilab/Scicos and Linux RTAI - a unified approach, Proc. of the IEEE Conf. on Control Applications, pp. 1121-1126, Toronto, Canada, August 2005

Chindris, G.; Muresan, M. Deploying Simulink Models into System-On-Chip Structures, Proc. of 29th Int. Spring Seminar on Electronics Technology, 2006

Feng Xia, Longhua Ma, Zhe Peng, Programming Scilab in ARM Linux, ACM SIGSOFT Software Engineering Notes, Volume 33 number 5, 2008

Hladowski, L.; Cichy, B.; Galkowski, K.; Sulikowski B.; Rogers, E. SCILAB compatible software for analysis and control of repetitive processes, Proc. of the IEEE Conf. on Computer Aided Control Systems Design, pp. 3024-3029, Munich, Germany, October 2006

Longhua Ma, Feng Xia, and Zhe Peng, Integrated Design and Implementation of Embedded Control Systems with Scilab, Sensors, vol.8, no.9, pp. 5501- 5515, 2008.

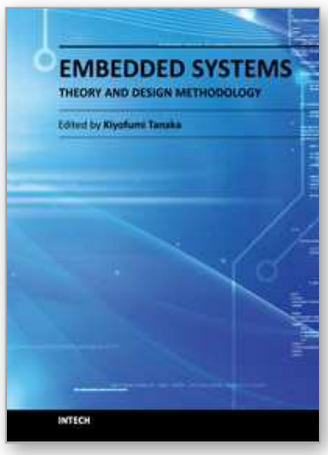
Mannori, S.; Nikoukhah, R.; Steer, S. Free and Open Source Software for Industrial Process Control Systems, 2008, Available from http://www.scicos.org/ScicosHIL/angers2006eng.pdf

Ma Longhua, Peng Zhe, Embedded ARM-Linux computation develop based Scilab, China Science publication, Beijing, China, 2008

Peng, Z. Research and Development of the Embedded Computing Platform Scilab-EMB Based on ARM-Linux, Master Thesis, Zhejiang University, Hangzhou, 2008.

Wittenmark, B.; Åström, K.J.; Årzén, K.-E. Computer control: An Overview, IFAC Professional Brief, 2002

Xia, F. & Sun, Y.X. Control and Scheduling Codesign: Flexible Resource Management in Real Time Control Systems, Springer, Heidelberg, Germany, 2008



Embedded Systems - Theory and Design Methodology
Edited by Dr. Kiyofumi Tanaka

ISBN 978-953-51-0167-3
Hard cover, 430 pages
Publisher InTech
Published online 02, March, 2012
Published in print edition March, 2012

Nowadays, embedded systems - the computer systems that are embedded in various kinds of devices and play an important role of specific control functions, have permitted various aspects of industry. Therefore, we can hardly discuss our life and society from now onwards without referring to embedded systems. For wide-ranging embedded systems to continue their growth, a number of high-quality fundamental and applied researches are indispensable. This book contains 19 excellent chapters and addresses a wide spectrum of research topics on embedded systems, including basic researches, theoretical studies, and practical work. Embedded systems can be made only after fusing miscellaneous technologies together. Various technologies condensed in this book will be helpful to researchers and engineers around the world.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen