Advances in Economics, Business and Management Research (AEBMR), volume 23
2017 2nd International Conference on Politics, Economics and Law (ICPEL 2017)

Factory Test Management System Based on Embedded Microprocessor

Li Tang
Department of Information Science and Technology
Tianjin University of Finance and Economics
Tianjin, China
tangli0831@tjufe.edu.cn

Baosheng Wang
Department of Information Science and Technology
Tianjin University of Finance and Economics
Tianjin, China
905494664@qq.com

Li He
Department of Information Science and Technology
Tianjin University of Finance and Economics
Tianjin, China
renkeheli@163.com

Shuhua ZHANG
Coordinated Innovation Center for Computable Modeling in Management Science
Tianjin University of Finance and Economics
Tianjin, China
shuhua55@126.com

***Abstract*—The factory test management system is introduced to reduce the failure rate of intelligent instrument. This paper analyzes the structure of foreign factory test management system. Based on it, this paper introduces the system architecture, testing processes, communications protocols and the software realization of the new factory test management system developed independently, and especially focuses on the design of embedded GUI (MiniGUI) and Linux device drivers. Compared with the foreign factory test system, this factory test management system with 32-bit microprocessor has the advantage of low-cost, small size and strong flexibility.**

Keywords—factory test management system; embedded microprocessor; intelligent instrument; Linux device driver

I. INTRODUCTION

Instrument is the core component of industrial control system. With the development of computer technology, more and more embedded systems are applied to the field of intelligent instrument. However, intelligent instruments also have their shortcomings. There is a high failure rate and rework rate. This undoubtedly adds to the cost of the business. To solve this problem, the concept of factory testing is introduced, and based on it, the factory test management system is generated.

The factory testing infers to the testing within the product assembly and packaging period, testing each component functions of products, to check whether they work well or not. This requires a special system that includes specific programs running in the product, and some detection devices. Factory testing management system in China is almost rare, and in some large multinational enterprises is widely used, such as SIEMENS (Germany), GE and Honeywell (USA). Some foreign factory test systems are expensive and versatile, and some functions are not practical for Chinese users. Therefore, it is necessary to develop a factory test management system with Chinese characteristics.

This research is supported by the Tianjin Natural Science Foundation of China (15JCYBJC16000) and Tianjin innovative training program for college students (201610070023).

II. SYSTEM INTRODUCTION

A. Foreign Factory Test System Structure

The foreign factory testing system includes the control end and the test end. The control end is PC, and the test end is composed of test fixture and test bench. The control end sends the factory test command to the test end through the RS-232, and receives the data replied by the test end.

The foreign factory test system is listed in Fig. 1.

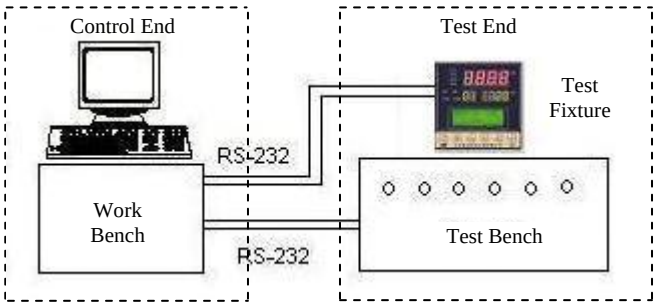

Fig. 1. Foreign factory test system.

B. A Brief Introduction to the Structure and Function of the New Factory Test Management System

The control end sends the command to the test end through the serial port and tells the test end to do the corresponding operation. At the same time, it receives the information returned by the test end through the serial port. The development platform of control end is ARM9+Linux.

The test end includes test fixture and test bench. The test fixture is used for placing the product, and the chassis has a

thimble connected with each test point of the product. The test bench mainly has the following functions:

- Supply power to test the product.
- Provide up to 20 LED lights to simulate the I/O output of real products.
- Provide A/D sampling thermistor.
- Provide the pneumatic devices to simulate the press button.
- Provide frequency meter to check internal RC calibration result.

The new factory test management system is listed in Fig. 2.

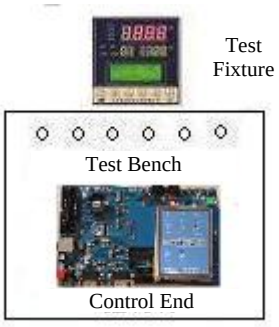

Fig. 2. The new factory test management system.

Compared with the foreign factory test system, the new factory test management system has the lower cost because it reduces some expensive function like man-machine interface image recognition system and replaces PC with the 32 bit embedded microprocessor in the control end. It is directly embedded into the test bench, greatly reducing the size. The embedded Linux operating system and the embedded graphical user interface MiniGUI make the control end software almost has the same friendly user interface and functions as the foreign factory test system software. It is developed by VB.NET.

III. Test Flow of Factory Test Management System

The main processes of factory test management system are shown as the following:

- Power on the test end.
- Check the A/D sampling module of the product. The input of the A/D is provided by the thermistor of the test bench.
- Check product LED. Test products, all LED digital tubes are all turned on, they are turned off after five seconds. This process is manually detected.
- Check product BUTTON. The input of the human finger is simulated by the pneumatic device of the test bench.
- Check the I/O module of the product. The status of each I/O is shown by the light of the tester.
- Check product internal RC calibration result. The output frequency of the CLKOUT pin of the product is detected by the frequency meter of the tester, and the result is the basis of whether the internal RC calibration is accurate.
- Finish the factory test.

IV. Communication Protocol for Factory Test Management System

A. Protocol Format Table of Factory Test

TABLE I. PROTOCOL FORMAT TABLE OF FACTORY TEST

START BYTE	LENGTH	DATA	CRC8
0XE9	1 BYTE	N BYTES	1 BYTE

B. Communication Command Table of Control End and Test End

TABLE II. COMMUNICATION COMMAND TABLE OF CONTROL END AND TEST END

Control End Command	DATA	Test End Command	DATA
START	0X00	A/D TEST RESULT	0X80 + 4 BYTE
TEST LED	0X01	TEST LED END	0X81
TEST BUTTON	0X02	TEST BUTTON END	0X82
TEST I/O	0X03	TEST I/O END	0X83
TEST RC	0X04	TEST RC END	0X84
END	0X05		

V. Software Implementation of Factory Test Management System

A. Software Implementation of Control End

The software of control end is implemented by C and embedded MiniGUI which are based on embedded Linux[1-2].

1) Embedded Linux Customization and Migration

The design of embedded Linux includes the customization and migration of Linux kernel[3]. The main purpose is to remove the all the useless modules, reduce the size of the Linux as much as possible. The purpose of Linux migration is to make Linux run on the ARM platform, which requires the arm-linux-gcc cross compiler. Add the content in the path of the Makefile:

CROSS_COMPILE=/usr/local/arm/2.95.3/bin/arm-Linux-gcc.

Then, run “Make Menuconfig” command to graphically configure the kernel menu option, customize, save and exit, run “Make dep”, “Make Clean”, “Make zImage”, and generate the Linux kernel image file that can run on ARM.

2) Development of Linux device drivers

In this project, the hardware needs to be operated. In Linux, hardware operations are implemented by device drivers. In the Linux operating system, the driver is the direct interface between the operating system kernel and the hardware device, and the driver shields the detail information of the hardware (such as bus protocol, DMA operation, etc.). From the view of application layer, the hardware device is just a device file, and the application can operate the hardware device just like an ordinary file. Device drivers are part of the kernel[4].

In this project, the embedded microprocessor sends the factory test command to the test end, and receives the data transmitted from the test port through the USART port, so it is necessary to design a USART Linux device driver. Because the user processes are handled hardware through device files, the manner of operating to the device files is called system calls such as open, read, write, close, and so on. But if associate system calls with drivers, it is necessary to use a very critical data structure:

```
static struct file_operations usart_fops = {
        read:       usart_read,
        write:      usart_write,
        ioctl:      usart_ioctl,
        open:       usart_open,
        releas:     usart_close,
}
```

The device driver is loaded in the form of module to the kernel dynamically, each loading process will call init_module automatically. This function applies a major device number from the operation system, and creates a device file in the default directory. The cleanup_module will be called once the module is un-installed, and this function will cancel the major device number, and delete the device file in the default directory. By using the “open” function in the application to open the device file, it then uses the “IOCTL” function to set the communication mode of USART. Finally, it uses “read” and “write” function to read and write from/to the devices.

3) Design of embedded MiniGUI

MiniGUI is a graphical user interface support system, and all the general GUI programming concepts can be applied to MiniGUI programming, such as windows and event driven programming, etc. [5-6].

This project requires MiniGUI to run on the ARM platform, so MiniGUI should be configured for it. Configuration method is to enter the directory of the source code /libminigui, and execute the “make menuconfig” command to configure:

- Compiler:arm-Linux-gcc;
- Path prefix:/usr/local/arm/2.95.3/arm-Linux;
- GAL engine option: choose “NewGAL engine on Linux FrameBuffer console”;
- IAL engine option: choose “Ipaq h3600(also H3800)”;
- Font options: choose “QT Prerendered Font”, don’t choose “Var bitmap font”;
- After configuration is finished, modify SRC/mybmp/Makefile, add source code shown as below:

  INCLUDES=-I/opt/host/armv4l/armv4l-redhat-Linux/include;

  LDFLAGS=-L/opt/host/armv4l/armv4l-redhat-Linux/lib;
- And then execute “Make” to compile; finally execute “Make install”.

The user interface designed by MiniGUI is shown as Fig.3.

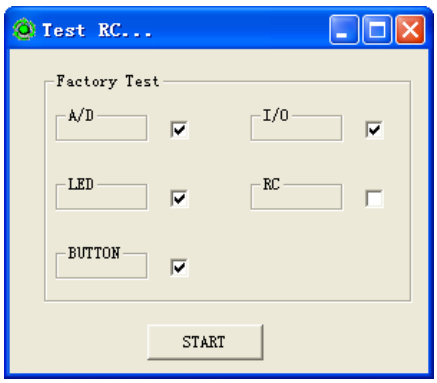

Fig. 3. The user interface designed by MiniGUI.

B. The Software Implementation of Test End

The test end software is running on the product and it is different from the software designed for the customer. The core of the software is a state machine function:

```
void FactoryTestMachineState(STATE_TYPE status)
{
    switch (status)
     {
            case Test_AD_Status:
                // Do AD verification
            break;
            case Test_LED_Status:
                // Turn on LED step by step
             break;
             ……
       }
}
```

After factory test is finished and the product works well, the final customer software will be programmed into the product.

VI. CONCLUSION

From the test result, the new factory test management system can reduce the failure rate and rework rate apparently, and save a lot of cost for the enterprise management. Comparing with foreign factory test system, the new factory test management system has the advantages such as lower cost, smaller size and so on. 32-bit embedded system makes the

whole factory test management system with higher extendibility, and makes it easy to be upgraded in the future. So in the field of control instruments management, it is important both in theoretical and practical application.

ACKNOWLEDGMENT

The research is supported by the Tianjin Natural Science Foundation of China (15JCYBJC16000) and Tianjin innovative training program for college students (201610070023).

REFERENCES

[1] Y. Tang, Real Time Operating System Application Development Guide, Beijing: China Electric Power Press, 2002, pp. 13-25. (In Chinese)

[2] H.X. Jiang, "Programming C on ARM Embedded Platform," Computer Applications and Software, vol. 20, pp. 15-17, 2003. (In Chinese)

[3] C. L. Du, ARM Architecture Stuction and Programme, 2rd ed., Beijing: Tsinghua University Press, 2015, pp. 384-411. (In Chinese)

[4] X. H. Lan, "The Design of Interrupt Device Driver in Embedded Linux Operating System," Application Research of Computers, vol. 20, pp. 96-98, 2003. (In Chinese)

[5] S. Y. Zou, Design and application of Linux embedded system, Beijing: Tsinghua University Press, 2002. (In Chinese)

[6] W. X. Tian, L. L. Zhang, Embedded Linux programming, Beijing: Tsinghua University Press, 2017. (In Chinese)