Remote Data Acquisition and Control System for Mössbauer Spectroscopy Based on RT-Linux

Zhou Qing Guo*, Huang Chun Hong, Zhou Rong Jie
(Applied Magnetics Key Lab of Education Ministry, Department of Physics, Lanzhou University, Lanzhou, 730000, P.R.China)

Abstract

In this paper a remote data acquisition system for Mössbauer Spectroscopy based on RT-Linux is presented. More precisely, a kernel module is in charge of collecting the data from the data acquisition card which is self-made based on ISA, sharing the data with the normal Linux process through the module of mbuff, carrying out the remote control and returning the results to the client by building a simple and effectual communication model. It's a good sample to deal with the communication between the real time process and the normal process. This user application can access to this system by the browser, or Java program to implement the real time observation and control.

1. Introduction.

1.1 Mössbauer spectroscopy

Mössbauer spectroscopy is a nuclear spectroscopy that has an energy resolution sufficient to resolve the hyperfine structures of nuclear levels. for example ^{57}Fe ,it reaches 10^{12} (Δ E/E). It has used in many domain, such as: Material science, chemistry, biology, archaeology, and so on. Rudolph L. Mossbauer in 1958 discovered that when the emitting nucleus is bound into a lattice there is a chance that the nucleus experiences no recoil. This is the case when the recoil-energy is smaller than the required energy for phonon creation. It is this recoilless emission which is called the Mossbauer effect. And Rudolph was granted the Nobel Prize on physics for this effect in 1961.

A typical Mössbauer spectroscopy system structure is shown in Figure 1. The radioactive source is moved by a triangular of the frequency 100 Hz. The spectra are recorded as a function of velocity, using a 486DX PC. The scan through all channels is controlled by a digital waveform generator, which produces the velocity reference signal driving radioactive source and two digital pulses: the multi scaling start (STA) and the channel advance (CHA). When a γ-ray has passed through the absorber is detected, a pulse (COUNT) is produced and accumulated in the data collection card as the counts of this channel. When the next CHA is coming, the counts are collected by the PC.

* Corresponding to author: e-mail: zhouqg@lzu.edu.cn

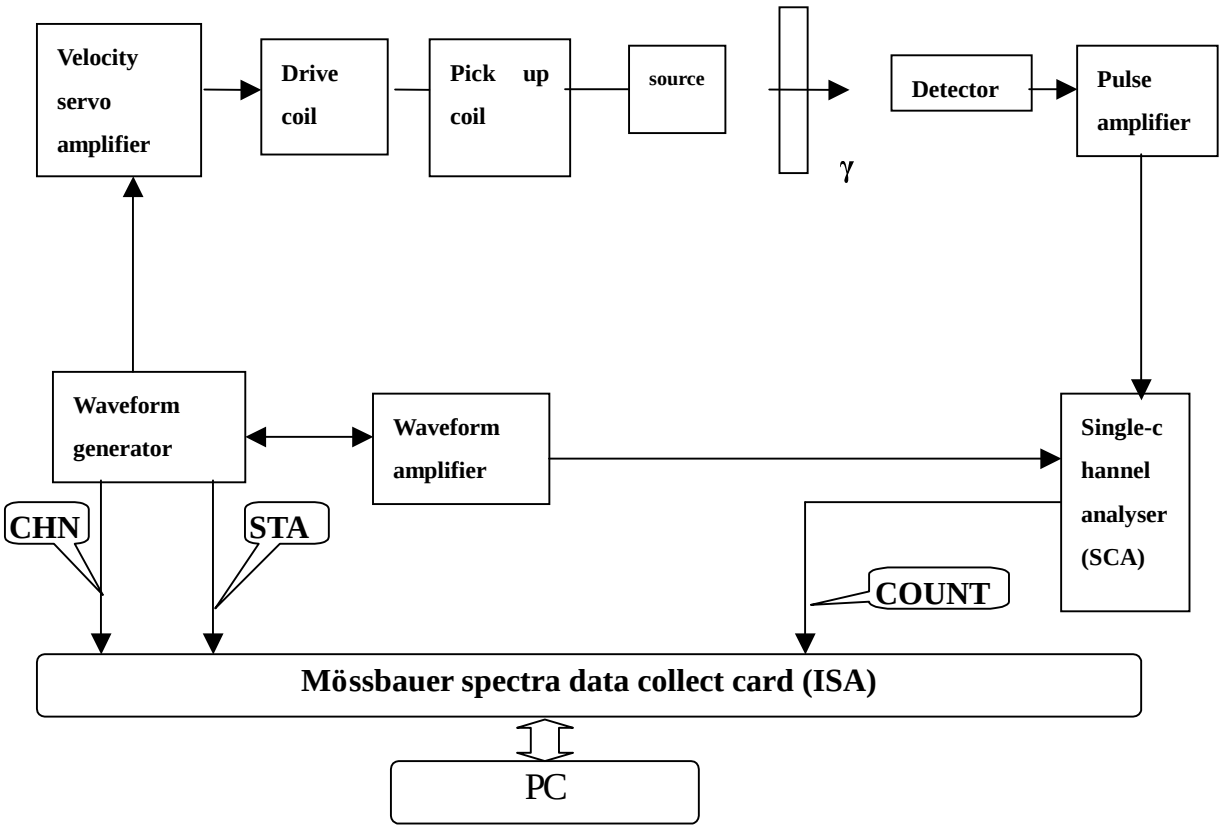

Fig.1 Mössbauer spectroscopy system structure

In this way each channel will receive those γ counts which are registered in the narrow velocity range assigned to it, and the channel number is directly proportional to the velocity. Many pulses are detected and stored during each cycle of the motion, and successive cycles over a long period of time allow the spectrum to build up as a whole. At velocities where resonance absorption occurs the accumulation rate will be slower.

1.2 System Overview

The system is the model of the client/server. It is composed of three parts: Real time collect data module, Cron process module, Client part. They are designed as module. Its structure graphic is shown in Fig.2.

1.Cron process module:

It includes web server process, and the cron process which executes listening the port, verifies the login users, and monitors status of the system. They are concurrent processes.

2.Real time collect data module:

It executes the instructions sent by the client, simultaneously deals with the hardware IRQ and run the ISR.

3.client part:

There are two methods which the authorization user could control the system and visit the data. The user could do this by the browser which is supported Java virtual machine .If they have not, they also can use the Java application program written in particularly.

To the system, only the authorization user could control the equipment by

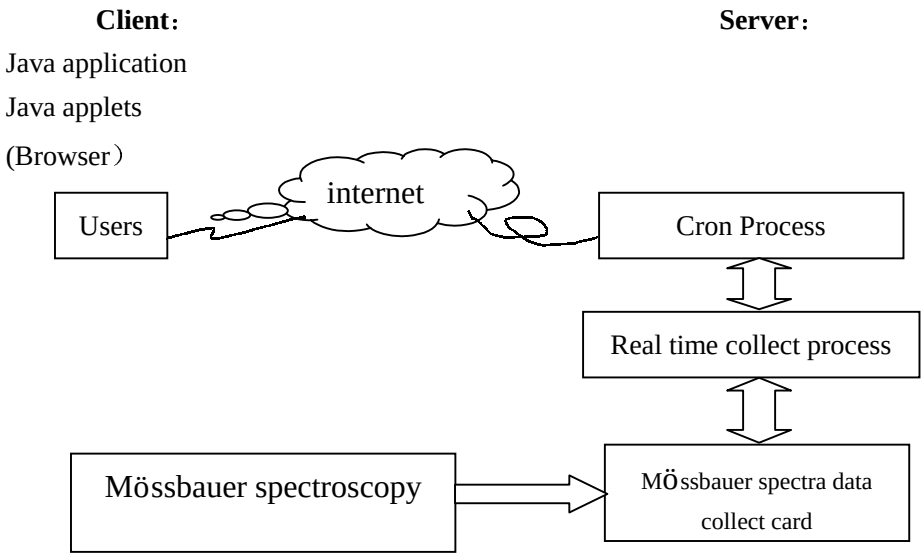

Fig.2 The graphic of the system structure

sending the instructions through the INTERNET. For example, there is a simple corresponding: The client sends the instruction request, and the cron process captures the request and verifies it. And then execute the corresponding task and return the system status to the client.

2. Communication Model

In this section the simple and effectual communication model is presented briefly. In the right (Fig.3), it's the communication model of the system. There are three types of data being exchanged between the normal process and real time module. There are the commands from the client, the result of the command being executed, and the collecting data. The first two are of the char data type. The last is of the block data type. So dealing with the first two just use the pipes and real time FIFOs. In this model it requires creating an in pipe and an out pipe, and is also the real time FIFO. One is used as receiving the commands, the other is used as returning the result. The collecting data is exchanged between *Reserve* and *Rtcollect.o* by *Mbuff.o*.

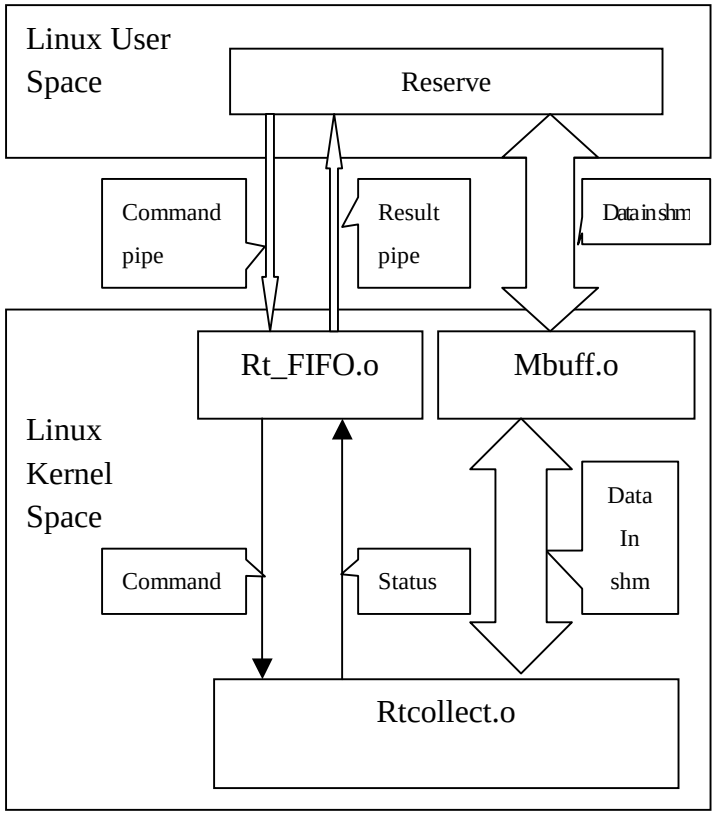

Fig.3 The Communication Model

In the following, the Java-code of the client and the C-code of the server on communication are presented below.

1.The Client (In Figure 4):

To free the application user, the client part is writing in Java. It is composed of four parts. One is the bullet of the Commands which user could send to the server through the sockets. The other is the bullet of displaying the data which comes from the shared memory buffer. Another is the utility bullet for simple dealing with the data of the displaying bullet. The fourth is a label to display the message when a action occurs.

Here given the network communication of the client (written in Java).

```
if(socket==null){
    setMsg("Connecting to
SERVER---" + Address + ":" + Port);
        socket=new
Socket(Address,Port);

socket.setSoTimeout(50000);
    setMsg("Successfully
Connected to SERVER---" + Address +
":" + Port);
    tempStatus=status;
        status=11;
        }
DataOutputStream out=new
DataOutputStream(socket.getOutputStre
am());
DataInputStream in=new
DataInputStream(socket.getInputStream
());
while(in.available()>0)
in.readByte();
```

2.The Server:

This part is divided into two parts, a cron program as running background and listening the client request from network, and a real time module responsible for dealing with IRQ, executing ISR and the command from the client.

- Reserve, the cron program running background as the normal Linux process.
- Rtcollect.o, the real time module initializing the real time thread.

User space cron process--Reserve

In the system, Reserve is the main program running as the normal process in the user address space. This part is a cron process to create the network socket, dealing with the request of network connect, verifying users and multiusers's instructions, and transiting the command and returning the status information after executing the command, etc. For example, explaining the system how to verify multiusers's instructions. In this system, there are several particular commands, such as: START, STOP, CLEAR, REFESH, SAVE. To run these commands, the normal Linux process and the real time process need to involve in at the same time. The procedure is list. First, the normal Linux process "fork" a child process to read the data of the RT_FIFO coming from the real time module in the circle. Here is the code:

```
if (!fork())
    /*Create the child process*/
{
    FILE *fp;
    char id[5],sta;
    while(1)
{
rt=     open(RTRFIFO,O_RDONLY));
```

```
    /*Read the information of RT_FIFO*/

read(rt,id,4);
read(rt,&sta,1);
close(rt);
id[4] = '\0';
/*id means the normal FIFO，*/
fp = fopen(id,"w");
fprintf(fp,"%c",sta);
fclose(fp);
    }
        return 1;
    }
```

The normal Linux process also "fork" a child process to wtite the data to the RT_FIFO coming from the pipe created by Reserve in the circle. The code is below:

```
ctl=open(RTWFIFO,O_WRONLY);
if(!fork())
{
….
umask(0);
mknod(pipe,S_IFIFO|0664,0);
    write(ctl,ss,7);
/*writing the data to the RT_FIFO*/
    readstatus(new_fd,pipe);

while(filter(new_fd,ss,user,pipe));
        remove(pipe);
            exit(0);
         }
         else close(new_fd);
            }
            close(sock_fd);
            fclose(logfp);
    exit(0);
}
```

Shared memory buffer:

The shared memory module mbuff.o is for data being exchanged between *Reserve* and *Rtcollect.o.* Here presented the two sidesof the source code are:

On *Reserve's* side:

```
#include <mbuff.h>
long *buffer;
….
buffer=
(long)
mbuff_attach("RTBuffer",MAXCHANNEL*4);
```

On *Rtcollect.o's* side:

```
#include <mbuff.h>
static long * buffer;
static long bufferA[MAXCHANNEL],bufferB[MAXCHANNEL];
    if ((buffer = mbuff_attach("RTBuffer",MAXCHANNEL*sizeof(bufferA[0]))) == NULL)
    {
        rtl_printf("RT_Collect:allocates memmory failure.\n");
        return(1);
    }
    #ifdef debug
    rtl_printf("RT_Collect:the init_module reports the buffer pointer is %ld\n",(long)buffer);
#endif
```

When a client request arrives, "Reserve" forks a child process, and this process is responsible to the client request. Now there are three processes in the system, which are the parent process (Reserve), the child process, the real time process (Rtclloect.o).

Real time module:

This is the key part of the system. The data source produces the interrupt signal every 20 μ S, so the interrupt service

should be finished in the time between the interval of the twice IRQ, otherwise could not protect the reliable data. First, interrupt initialization. When the real time task is loaded into kernel address space to execute as the Executable and Linkable Format. The procedure is listing below:

```
int init_module(void)
{
    rtl_hard_disable_irq(IRQ);
    rtf_create(FIFO_Ro,1000);
    rtf_create(FIFO_Wo,1000);
        outb(0xBF,PORT_D);
 /* set 8255 A&B port in mode 1,PC7 in read mode*/
        outb(0x09,PORT_D);
/* enable INTEA*/
        outb(0x04,PORT_D);
/*disable INTEB*/
    signal1 = STOP;
        signal2 = STOP;
    status.port[0] = STOP;
    status.port[1] = STOP;
        if (rtl_request_irq(IRQ,rt_handler) == -EBUSY)
  {rtl_printf("RT_Collect:Irq %d is busy. Maybe this programm is already running.\n",IRQ);
            return(1);
        }
    rtf_create_handler(FIFO_Ro,&fifo_handler);
if ((buffer= mbuff_attach("RTBuffer",MAXCHANNEL*sizeof(bufferA[0]))) == NULL)
    {

rtl_printf("RT_Collect:allocates memmory failure.\n");
            return(1);
    }
#ifdef debug
    rtl_printf("RT_Collect:the init_module reports the buffer pointer is %ld\n",(long)buffer);
#endif
    rtl_hard_enable_irq(IRQ);
    return(0);
}
```

3. System Hardware

In this system, there is a 486 PC whose CPU is 486DX, with 24M memory, a ISA based data acquisition card, and a NE2000 network card. In the designing, the system could be visited by the ten clients at the same time through Internet. Mössbauer spectra data collect card is composed of a 8255 chip, counting unit, logic circuit. The 8255 chip is the interface between the counting unit and pc, and it works under mode one.

It could meet the performance request as below:

1. Channel range: 0-1024
2. Maximum driver frequency: 100 HZ
3. Maximum counting frequency(the limit of 74LS393 Chip): 2×10^6 /S
4. The worst case: 20 μ S

4. Results

The remote data acquisition and control system has been developed at Applied Magnetics Key Laboratory of the Education Ministry, Lanzhou University. Especially in order to realize that the experimenter could do the test in the

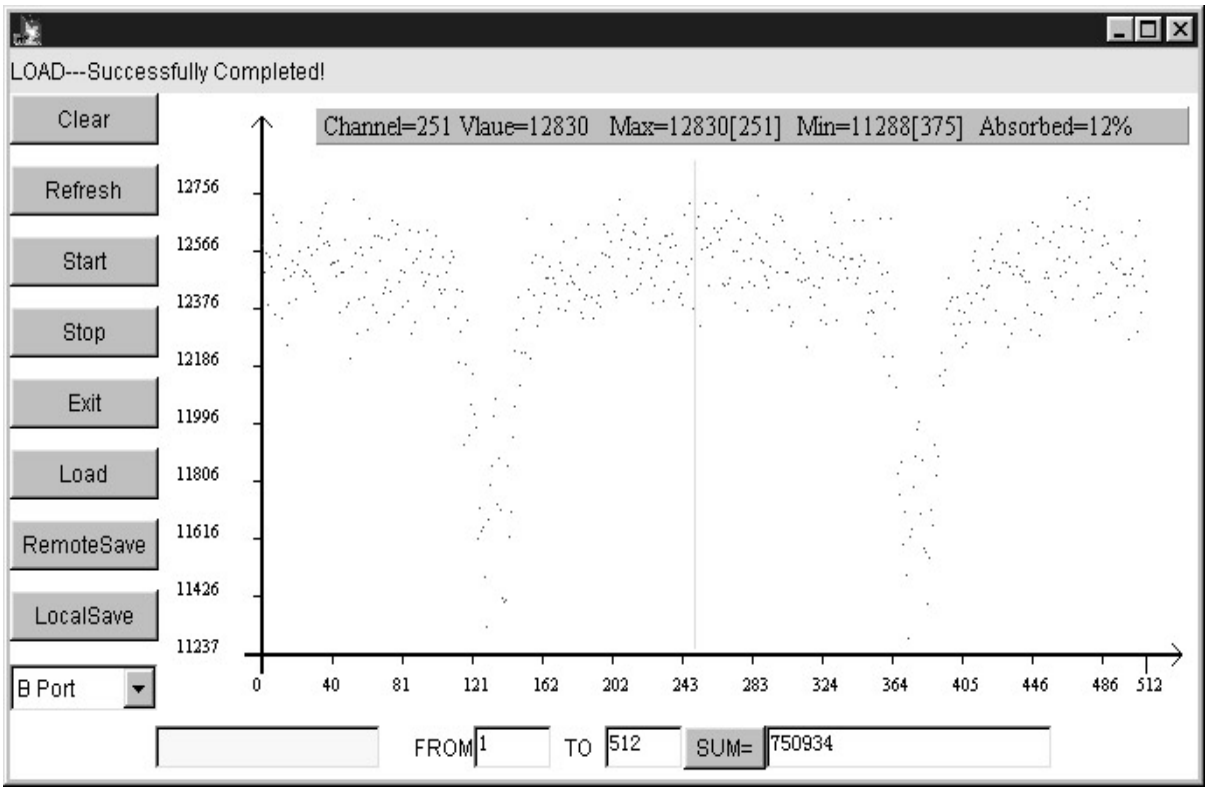

Fig.4 The result of the sample ($Ni_{0.5}Zn_{0.5}Fe_2O_4$ Nano Particle)

home through the Internet. From this system, it could prove that real time Linux is a good means to carry out this job and is a very good hard real time OS. And as known, the source is radioactive isotope which should be harm people. But now they could do the experiment anywhere if they want, just through the network, better than go to the laboratory. Here presented the result of the sample ($Ni_{0.5}Zn_{0.5}Fe_2O_4$ Nano Particle) above. The x-coordinate is the open channel, and the y-coordinate is the count corresponding to the channel.

Acknowledgements

This work has been carried out within the help of the RTLees in the real time Linux maillsit, especially the help from Nicholas Mc Guire.

References

[1]Rongjie.Zhou,Jinbo.Yang,Fashen.Li, Journal of Lanzhou University,31(2) 1995 46

[2]Motylevski, Thomasz (1999).mbuff,a kenel shared memory drivers, http://crds.chemie.unibas.ch/

[3] Yodaiken, Victor (1999). RTLinux, real time linux, FMSLab, Soccoro, New Mexico, USA
http://www.rtlinux.org/
http://www.fsmlabs.com/