A Complex Hardware Control System Based on Micro Control Units

Boling He[1,*], Wen Xu[2], Yipu Zhang[2], and Lei Liu[3]

[1]*Department of Computer Science*, Changchun Finance Academy, Changchun, Jilin, 130028, China
[2]Simulation Institute, Aviation University of Air Force, Changchun, Jilin, 130022, China
[3]College of Computer Science and Technology, Jilin University, Changchun, Jilin, 130012, China

***Abstract* — A reusable software system called hardware driver is designed and implemented to control complex hardware systems controlled by micro control units based on embedded UDP/IP, with focus on the scheduling of micro control units. The number of the hardware drivers are based on the scale of the devices. Usually,one hardware driver with one or more network interface cards is enough. Each device is controlled by a micro control unit to sample the hardware items that sampled data is delivered to clients through the hardware driver that receives the driving data from the clients and transmits them to the micro control unit driving the device. Each hardware driver runs on an industrial personal computer and communicates with the micro control units through the lower network, a UDP/IP network, while it communicates with the clients through the upper network, a UDP/IP and(or) a reflective memory networks. The hardware driver with a reusable software architecture makes any number of micro control units integrated seamlessly in dynamic link libraries without any need to modify the architecture. The only task for developing a new device control system is to implement the embedded routines of the micro control units that must be done with any other development methods. This control method fits the man-in-loop simulators best with the advantages of less wiring than analogue-digital cards usually used and has some other beneficial features of developing efficient, run-time, software reusability and precision maintenance.**

Keywords* — *Micro control unit; Embedded UDP/IP; Drive hardware

I. INTRODUCTION

For a simulator with a small-scale devices, analogue-digital cards fit better for sampling and driving hardware devices. But for the systems with large-scale devices, the wiring would be most dense and mass leading troubles for both development and maintenance. Besides, in practices with controlling devices, the software reusability and developing mode are not be paid attentions that lead to many problems, such as repeated development of hardware controlling software, mutual interdependence between hardware developers and software developers, device precision maintenance invalid due to device wearing. Now the technology of embedded UDP/IP or integrating UDP/IP stacks into embedded system insures the connection between embedded systems and Ethernet based on UDP/IP. This technology is used widely in industry control fields[1,2,3,4,5,6] but its application in simulators is not reported. This paper presents a complex control system in which a huge mass of devices controlled by micro controller units based on embedded UDP/IP, with emphasis on the management of those micro control units through one or more reusable hardware drivers. This device control method has the features of reusable driving architecture, more developing efficient, easily maintenance, run-time and reliable performance and fits the man-in-loop simulators best with the most advantage of making the system wiring simple,elegant and clean than AD cards usually used.

II. SYSTEM STRUCTURE

Fig.(1) shows the overall structure of the device control system based on embedded UDP/IP. There are three types of partners: the micro control units[7], the hardware drivers and the applications. The number of the hardware drivers bases on the mass of the devices

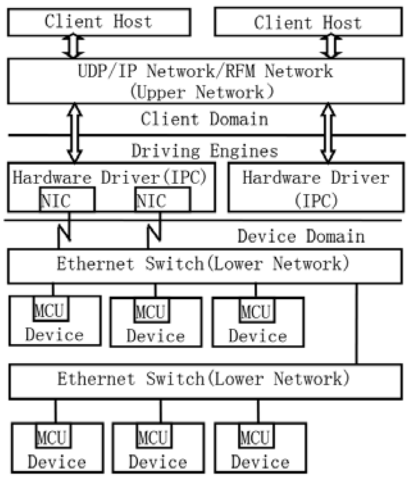

Fig.1 Hardware System

and the real-time ability. For most cases, one hardware driver with one or more NICs(Network Interface Cards) is enough. Each device is controlled by a micro control unit to sample the device and the sampled data is delivered to applications through the hardware driver that receives the driving data from the applications and transmits it to the micro control unit to drive device. The hardware driver running on an industrial personal computer(IPC) communicates with the micro control units through the lower network, a UDP/IP network,

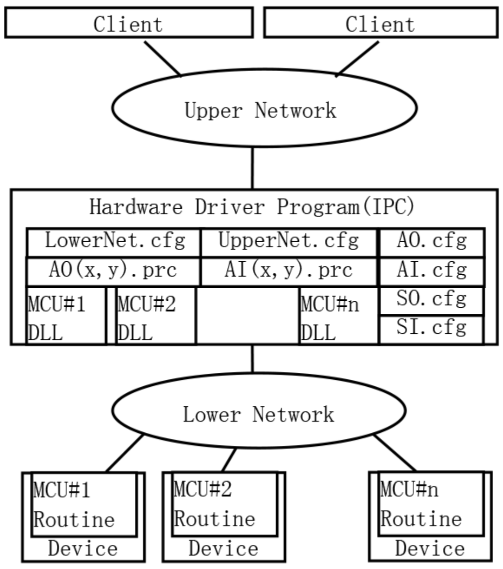

Fig.2 Software System

while it communicates with the applications through the upper network, a UDP/IP network and(or) a reflective memory (RFM) network. The hardware driver is the proxy of all micro control units and the applications communicate only with the hardware driver but not any micro control units. Usually the device domain, such as aircraft cabin, is far off the applications and(or) the hardware drivers. Fortunately, the embedded UDP/IP applied ensures the wiring simple and elegant. Fig.(2) shows the software topology structure including the configuration files that insulate special devices and communication node topology from the reusable hardware driver.

III. PROTOCOL DATA

There are two types of protocol data[8,9]:

- **Protocol data from micro control unit to hardware driver**
  Each micro control unit sends UDP packet to the hardware driver as follows:

```
struct ToDriver
{
char type;/*Micro control unit type:0-sampling and driving; 1-sampling; 2-driving.*/
char data[];/*The valid sampled data according to the micro control unit.*/
};
```

  This type of packet must be sent even the micro control unit is driving only. In this case the packet is regarded as a heartbeat packet.

- **Protocol data from hardware driver to micro control unit**
  The hardware driver sends UDP packet to each driving micro control unit as follows:

```
struct ToMicro
{
char data[];/*The driving data according to the micro control unit.*/
};
```

IV. HARDWARE ITEM NOTATION

A. Hardware Item Identifiers

Each controlled hardware item on a device is identified in two-dimensional notation. An analogue output(such as a meter needle) is marked as AO(x,y). An analogue input(such as a knob) is marked as AI(x,y). An digital output(such as a light with two statuses:ON or OFF) is marked SO(x,y). An digit input(such as a switch) is marked as SI(x,y). Where x is the number identifying the panel that the item on and y is the number identifying the item.

B. Hardware Item Reference

Each hardware item can be sampled or drove in a C/C++ program as follows:

- **Driving hardware items**
  AO(x,y)=*a;*/*AO(x,y) such as a meter needle is drove to value *a**/
  SO(x,y)=0;/*SO(x,y) such as a light is turned off.*/
- **Sampling hardware items**
  float *a*=AI(x,y);/*AI(x,y) such as a knob is sampled and is saved in variable *a*.*/
  int *s*=SI(x,y);/*SI(x,y) such as a switch is sampled and is saved in variable *s*.*/

C. Notation Implementation

Fig.(3) shows how to locate the hardware items in its buffer through the packed index. The hardware items lay on the panels from i_0 to i_m. The packed index is built based on the files AO.cfg, AI.cfg, SO.cfg and SI.cfg. These files must match the following syntax:

ItemList:=Item ItemList

Item:=AItem | SItem

AItem:=AO(*x,y*)[*:T*] | AI(*x,y*)[*:T*]

SItem:=SO(*x,y*)[*:S*]|SI(*x,y*)[*:S*]

S:= number .. number

x:= number

y:=number

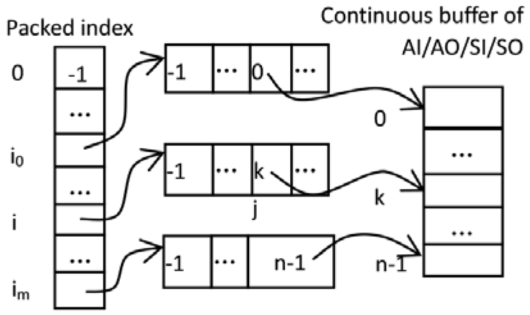

Fig.3 Time-Efficient Index

V. PRECISION CONTROLS

A. Analogue-Output Precision List

The drive process is as follows when a micro control unit controls an instrument needle (analogue output) to indicate scale *x*:

z=cvt(x), z⟶ MicroControlUnit.

where, the conversion calculation function cvt(x) is determined by hardware drive mode. The common problem is the error in terms of instrument indication instead of correction indication to *x*. To correct the error, the drive process can be modified to be:

y=f(x), z=cvt(y), z→ MicroControlUnit.

That is, first to get *y* by means of proper turning of *x* through the function *f* and then *y* is used for driving. The function *f* can be materialized by means of interpolation based on the precision list.

B. Analogue-Input Precision List

Sample of current value *y* of an analogue input (such as an knob) can be described by:

Receiving sample value z, y=cvt(z).

In the same way, the function cvt(z) is determined by hardware drive mode. Just like analogue output, sample of an analogue input is characterized by error, to correct which, sampling process can be improved as:

Receive sample value z from micro control unit, x=cvt(z), y=f(x).

The function *f* can be materialized by interpolation based on the precision list.

C. Precision Lists On Hard Disk

If hardware is maintained, the precision table files will be modified and the program code must not need to be modified. Thus the precision lists must be stored in external hard disk. For every item *AO(x,y)*, its precision list is stored in the file *AO(x,y).prc* and for every *AI(x,y)* , the list is stored in the file *AI(x,y) .prc*(see Fig.(3)). These precision lists will be loaded to memory by the hardware driver in initializing phrase. The precision lists in memory can be accessed by the interface *PrcLists*:

```
struct PrcLists
{
    virtual double AO(int x, int y, double a);
     /*Return the interpolated value of AO(x,y) according to a.*/
     virtual double AI(int x, int  y, double  a);
     /*Return the interpolated value of AI(x,y) according to a.*/
};
```

This interface will be passed to the micro control unit DLLs(Dynamic Link Library) in order to enable the DLLs accessing the precision lists.

VI. MICRO CONTROL UNIT DLLS

According to each micro control unit, a DLL exporting two functions Sample() and Drvive() as follows must be provided:

- **Algorithm of Sample(char data[],PrcLists DoItp)** where,data is buffer to save sampled data in micro control unit format and DoItp is the interface to access the precision lists.

  Step 1 Get AI(x,y) by parsing data:
  a= Get(data).
  Step 2 Do precision operation and update AI(x,y):
  AI(x,y)= DoItp.AI(x,y,a).
  Step 3 Parse data then update SI(x,y):
  SI(x,y)= Get(data).

- **Algorithm of Drive(char data[],PrcLists DoItp)** where,data is driving data in micro control unit format and DoItp is the interface to access the precision lists.

  Step 1 Do precision operation on AO(x,y):
  a = DoItp.AO(x,y, AO(x,y)).

Step 2 Set data with a: Set(data,a).
Step 3 Set data with with SI(x,y): Set(data, SI(x,y)).

VII. HARDWARE DRIVER

A. Lower Net Configuration

The lower communication refers to the hardware driver communicates with the micro control units based on UDP/IP. The communications are defined as a list with each item in following formats in the file *LowerNet.cfg*:

(MicroName, MicroIP, MicroPort, DrvIP, DrvPort) The hardware driver receives sample packets from *(DrvIP, DrvPort)* and sends driving packets from *(DrvIP, DrvPort)* to *(MicroIP, MicroPort)*. In the hardware driver endpoint, the DLL must be named as *MicroName.dll* so that the hardware driver can find the proper functions *Sample()*and *Drive()*.

B. Upper Net Configuration

The upper communication refers to the hardware driver communicates with the clients of the micro control unit data through a network of UDP/IP[10,11] and(or) reflective memory boards. The communications are defined as a list with each item in following formats in the file *UpperNet.cfg*:

(DrvIP,DrvPort,AppIP,AppPort)
(DrvRfmID,DrvRfmAddr,AppRfmID,AppRfmAddr)

The hardware driver receives driving packets from *(DrvIP, DrvPort)(DrvRfmID,DrvRfmAddr)* and sends sample packets from *(DrvIP, DrvPort)* to *(AppIP, AppPort)(AppRfmID,AppRfmAddr)*.

C. Driving Algorithm

The key steps of the algorithm of the hardware driver are as follows:

Step 1 Load files *AO.cfg, AIcfg , SO.cfg , SI.cfg,AO(x,y).prc,AI(x,y).prc,LowerNet.cfg, UpperNet.cfg.*

Step 2 FOR each *item* in *LowerNet.cfg* DO

Load DLL by *LoadLibrary()*[12,13].

Search *Sample()* and *Drive()* by *GetProcAddress()[12,13]*.

Step 3 FOR each *item* in *LowerNet.cfg* DO

IF Receive from *(DrvIP, DrvPort)* ok THEN Call *Sample(}*.

Call *Drive()*.

Send the driving packet from *(DrvIP, DrvPort)* to *(MicroIP, MicroPort)*.

Step 4 FOR each *item* in *UpperNet.cfg* DO

Receive AO(x,y)/SO(x,y) from *(DrvIP, DrvPort)(DrvRfmID, DrvRfmAddr)*.

Send AI(x,y)/SI(x,y) from *(DrvIP, DrvPort)* to *(AppIP, AppPort)(AppRfmID, AppRfmAddr)*.

Step 5]Goto **Step 3**.

VIII. CONCLUSIONS

It is a better way to apply embedded UDP/IP technology to sample and drive devices in man-in-loop simulators with a mass of hardware items. This method has advantages as follows:

- The system wiring is simple and elegant than AD cards usually used.
- The maintenance is easy due to each device controlled by its micro control unit inside.
- The hardware driver is a reusable software architecture and any number of micro control units can be integrated seamlessly in DLLs without any need to modify the architecture.
- The development is more efficient. To develop a new device control system, the only tasks are to implement the micro control unit embedded routines that must be done whenever any development methods applied and the micro control unit DLLs and then integrate these DLLs with the architecture. In addition, the two-dimensional notation makes hardware developers and software developers developing parallel.
- The external precision lists make the precision maintenance valid.
- The external lower and upper communication configuration files make the communication performing without any communication code to be developed.
- The number of the hardware drivers bases on the mass of the devices and the real-time ability. For most cases， one hardware driver with one or more NICs is enough.
- All clients of the device data communicate with the hardware driver but not any micro control units and that make the upper communication simple.
- The system perform run-time and reliable. Obviously, this method can be used in hardware-related systems especially in man-in-loop simulators to make the developing efficiently

CONFLICT OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGMENT

This work is supported by the 12th Five-year Science Foundation of the Education Department of Jilin, China (No. JJKHZ[2012]387).

REFERENCES

[1] Tong Z and Jingtao H., “Design and realization of EtherCAT master, ” 2011 International Conference on Electronic and Mechanical Engineering and Information Technology. pp.173-177, 2011.

[2] LIU Xin-shun and YAN Jian-guo, “Data communication and acquisition of mode calculation computer for UAV’s hardware-in-loop simulation in VxWorks, ” Modern Electronics Technique,No. 01, pp.7-9,2012.

[3] Schickhuber G and McCarthy O., “Distributed fieldbus and control network systems,” Computing&Control Engineering, Vol.8, no.1, pp.20-33,1997.

[4] Zhao Liang, Zhang Jili, and Liang Ruobing, “Remote Ethernet data transmission system based on hardware protocol stack chip,” Journal of Network, Vol.8, no.6, pp.1285-1291,2013.

[5] Knezic M, Dokic B, and Ivanovic Z, “Increasing EtherCAT performance using frame size optimization algorithm,” 2011 IEEE 16th Conference on Emerging Technologies & Factory Automation, pp.1-4, 2011.

[6] XU Hai,CUI Lianhu, and XU Guangyao, “Research on realtime collection method of timing information in RTX environment,” Ship Electronic Engineering,No. 4, pp.59-61,2012.

[7] Da-hua Li, Hang Li, and Kai-lin Zhang, “Development and realization of real-time data exchange between OPC client multiple remote servers,”Journal of Computers, Vol.9,no.1, pp.168-175,2014.

[8] Lei Wang and Junyan Qi, “The real-time networked data gathering systems based on EtherCAT,” IEEE International Conference on Environmental Science and Information Application Technology, pp. 513-515, 2009.

[9] Harry F., “Digital communication provides,” Control Engineering, Vol.41,no.1,pp.55-56,1994.

[10] Buck Graham, UDP/IP addressing : designing and optimizing your IP addressing scheme, Morgan Kaufmann, San Diego, Calif. , 2001.

[11] Michael J. Donahoo and Kenneth L. Calvert., UDP/IP sockets in C : practical guide for programmers (2nd Edition), Morgan Kaufmann Publishers, San Francisco, 2009.

[12] SUN Yahong, “Windows-based ditributed real-time simulation system,” Electronic Science and Technology, No.3,pp. 7-9,2012.

[13] Jeffrey Richter and Christophe Nasarre, Windows Via C/C++, Microsoft Press, Washington, 2011.