I.J. Engineering and Manufacturing, **2012,1, 54-61**
Published Online February 2012 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijem.2012.01.09

Available online at http://www.mecs-press.net/ijem

A General Simulation Framework Based on CAN Bus for Satellite Design[1]

Cheng Long, Cai Yuanwen, Liu Danghui

Department of Space Equipment, The Academy of Equipment Command & Technology, Beijing, 101416, China

Abstract

The running status of a satellite can be imitated realistically through the use of an interactive simulation system, in which actual hardware can join for testing and validation. To solve the problem of interfaces and communications between different members in a satellite simulation system, an on-board interactive simulation system based on CAN bus is established. Several simulation members are built according to the composition of real system. Interfaces of drivers included in simulation members are packaged into a uniform interface, thus the on-board simulation system is established with the foundation of CAN bus simulation framework. Uniform communication interface of each simulation member is designed and realized, the framework of simulation flow is established, and foundation of intercommunion and operation between simulation members is laid. Experiment results prove that the framework can work stably with great efficiency and high flexibility, and expected effect is achieved.

Index Terms: CAN bus; simulation; framework; satellite

1. Introduction

In recent years, people's demands for satellite application have risen sharply, this brought forward higher request for information transmission and processing ability of space vehicle, and these all highly lie on the performance of on-board data bus. CAN(Controller Area Network) supports a distributed control, and it can equal 1553B for data transmission rate and reliability, this has been proved in the automobile and the industrial control field[1]. Because of its advantages of performance and price, CAN is being applied to spaceflight more and more widely[2]. Especially in satellite application, CAN is capable of replacing the 1553B[3].At present CAN has been adopted for command and control bus or data bus in many satellites[4]. And it's very suitable to

[1] This work is supported by National High Technology Research and Development Program of China (863 program, No. 2008AA7045007)

* Corresponding author.
E-mail address: chenglong317@sina.com

built-in test[5] and formation flight[6] of satellites in near future. So on-board interactive simulation framework based on CAN bus must be established.

Considering the integration and complexity of simulation system, such as different CAN controllers and different driver interfaces, the interface of subsystem must be conformable with each other, so the framework of transmission has to be regulated in a general and standard way which be used in the software engineering. Another problem is the communication between subsystems. There are plenty of data and information between subsystems during simulation, how to regulate the interactive sequence and ensure the simulation time goes forward stably is very essential for system operation.

To solve the problem of interface specification and interaction in on-board interactive simulation system, a general simulation framework based on CAN bus is designed and implemented, it can be applied to simulation for various types of spacecrafts.

2. Structure of simulation system

In on-board interactive simulation system, models of subsystems could be hardware or software. Subsystems only need to accord with CAN protocol, no matter what mode it runs in. Thus a subsystem can communicate with others by CAN bus, and all subsystems compose the whole simulation system.

Satellite system usually can be divided into two main parts: useful load and platform. And platform is composed of some subsystems, such as structure, thermal control, power supply, attitude and orbit control, tracking telemetry and command, data management, electricity, reentry, etc[7]. According to the composition, a flexible and extensible on-board simulation system is established with the foundation of CAN bus simulation framework, its structure is illustrated in Fig. 1.

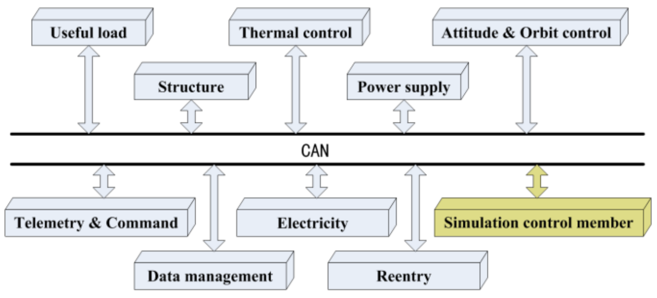


Fig. 1. Architecture of on-board simulation system

Besides all subsystem simulation members, there is also a simulation control member in system. It will manage and control the simulation process.

3. Design and implementation of CAN transmission framework based on simulation

3.1. Overall plan

The problem of communication in various members must be solved firstly in the process of system integration. Because of different design requirements, the members run in different environments, some in PC and some in embedded system. And they have different CAN devices which were produced by different manufacturers, so each member has its own CAN driver interface. In order to regulate the communication interface of system, all members' CAN driver interfaces must be packaged into a uniform CAN communication interface, as shown in Fig. 2. Using this uniform interface to perform a uniform CAN transmission, a cross-platform and cross-OS interactive simulation system can be established.

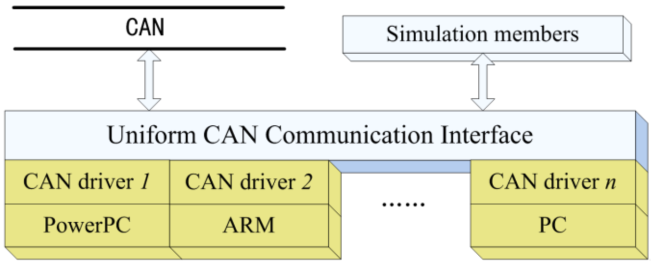


Fig. 2. Communication interface of system

The uniform CAN communication interface displayed in Fig. 2 is a CAN transmission framework, which is implemented by a C++ class CCANMemmber. CCANMemmber is packaged into a DLL in order that program developers can use it conveniently. This class is suitable for not only subsystem simulation members but also the simulation control member, just there are some differences in function calls during simulation. Class CCANMemmber's interface is shown below:

```
class _declspec(dllexport) CCANMember
{
private:
BYTE CanConfig[11];// CAN configuration parameters
public:
CCANMember(int n);//Initialize class, input is member's ID
~CCANMember(void);
int MemberID;// Member's ID
DataBuffer RevBuffer;//Data buffer
int MemberNum;//Number of members(available for control member)
void InitCAN();//Initialize CAN bus
void GetData();//Get data from bus
void SendJoinIns();//Send Join Request(available for subsystem members)
void SendDataReq();//Send Data Request(available for subsystem members)
void SendStepReq();//Send Step Request(available for subsystem members)
void WaitPermit();//Wait for Step Permission (available for subsystem members)
void WaitDataReq(); // Wait for Data Request(available for control member)
void WaitStepReq(); // Wait for Step Request(available for control member)
void SendDataPermit();//Send Data Permission(available for control member)
void SendStepPermit();//Send Step Permission(available for control member)
void SendData(int nParaIndex,float ParaDate);//Send float type data
void SendData(int nParaIndex,CString ParaDate);// Send CString type data
void SendData(int nParaIndex,double ParaDate);// Send double type data
void SendData(int nParaIndex,int ParaDate); // Send int type data
};
```

In order to ensure the convenience of framework, it is designed as simple as possible. Of course, it must meet the demand of simulation running. CAN drivers in different devices are mainly packaged in functions: InitCAN(), GetData(), and SendData(). SendData() performs a function overload for different data types. So we just need to modify these three functions in different members according to their CAN drivers. This can be done by a person who performs CAN drivers packaging specially. And other programmers, who answer for subsystem members, just use CCANMember to complete communication, disregarding what type of CAN controller is used and how the bottom driver interfaces are called.

3.2. Design and test on data transmission

To insure validity and reliability of the CAN drivers packaging, detailed design and testing should be performed according to driver interfaces provided by manufacturer. For example, one member may need to send many data frames continuously during simulation. This member runs in a PC system, which adopts a CAN communication card with PCI interface. And manufacturer drivers provide a function named CAN_Trans() for data sending. But we found that if CAN_Trans() was called continuously and repetitively, data sending would be delayed. The reason may be that PC processed the codes faster than CAN card’s response, so a block happened in CAN card when PC called CAN_Trans() repetitively, and CAN controller performed a delayed time processing in succession. Thus the transmission rate would not meet the demand. To solve this problem, we call CAN_Readreg() for a query about bus status after once CAN_Trans() function call, and perform next CAN_Trans() function call when bus is idle. Thus sending rate can match the baud rate which is set in program initialization. Main code is shown below:

```
CAN_Trans(0,CanBuffer,50);//Send a data frame
UCHAR CanReg[2];
CanReg[0]=2;
do{CAN_ReadReg(0, CanReg);}
while((CanReg[1]&12)!=12)// Query bus
```

In addition, data check has to be executed in case errors appear during data sending. These must be considered by the person who performs CAN drivers packaging.

Manufacturers generally will provide two modes of data receiving:

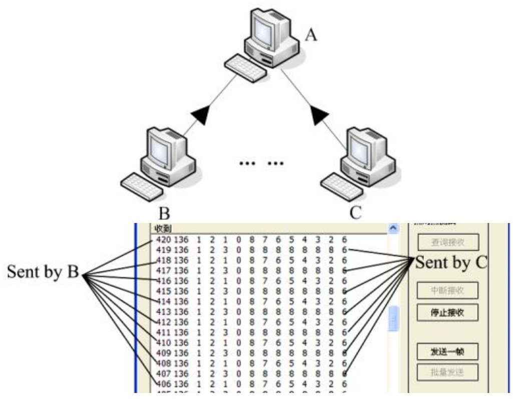
1) Immediate query mode

This mode receives data frames by a recursive call of a receiving function. But we found that some frames would be lost when high baud rate was set.

2) Buffer query mode

This mode is also called interrupt mode, because that CAN device will save arriving data to a buffer automatically by hardware interrupt. So software can get data in batches by querying buffer periodically. This mode can insure a high receiving speed and avoid loss of data.

We realized data receiving by buffer query mode, and tests of receiving performance were executed to find out whether some mistakes would happen in a high speed data receiving, such as conflict, confusion and loss. One case of tests is illustrated by Fig. 3.



```
420 136  1  2  1  0  8  7  6  5  4  3  2  6
419 136  1  2  3  0  8  8  8  8  8  8  8  6
418 136  1  2  1  0  8  7  6  5  4  3  2  6
417 136  1  2  3  0  8  8  8  8  8  8  8  6
416 136  1  2  1  0  8  7  6  5  4  3  2  6
415 136  1  2  3  0  8  8  8  8  8  8  8  6
414 136  1  2  1  0  8  7  6  5  4  3  2  6
413 136  1  2  3  0  8  8  8  8  8  8  8  6
412 136  1  2  1  0  8  7  6  5  4  3  2  6
411 136  1  2  3  0  8  8  8  8  8  8  8  6
410 136  1  2  1  0  8  7  6  5  4  3  2  6
409 136  1  2  3  0  8  8  8  8  8  8  8  6
408 136  1  2  1  0  8  7  6  5  4  3  2  6
407 136  1  2  3  0  8  8  8  8  8  8  8  6
406 136  1  2  1  0  8  7  6  5  4  3  2  6
```

Fig. 3. Receiving performance test

A is a PC with Windows XP OS and PCI CAN communication card, B is an embedded system with MCP2510 CAN controller, and C is also an embedded system but with SJA1000 CAN controller. Baud rate of CAN bus is set to 1000Kbps, then B and C send data frames to A separately at the same time. A receives data in buffer query mode, result shows that all data are received accurately.

4. Simulation flow

At the time of development of CAN transmission framework, an effective simulation flow has to be put forward. It’s the foundation of interaction and order for simulation members. Simulation flow is designed as Fig. 4.

Simulation process is divided into 3 steps:

1) Firstly, simulation control member builds simulation system environment. Then it queries bus continuously for Join Requests from subsystem members. At the same time, each subsystem member will wait for Step Permission from control member after sending its Join Request. When all subsystem members join, control member will send Step Permission.

2) Control member waits for Data Requests from subsystem members after sending Step Permission. At the same time, each subsystem member runs its own simulation model and makes result for this period, and then it sends Data Request and waits for Data Permission in succession. When control member receives all Data Requests, it will send Data Permission.

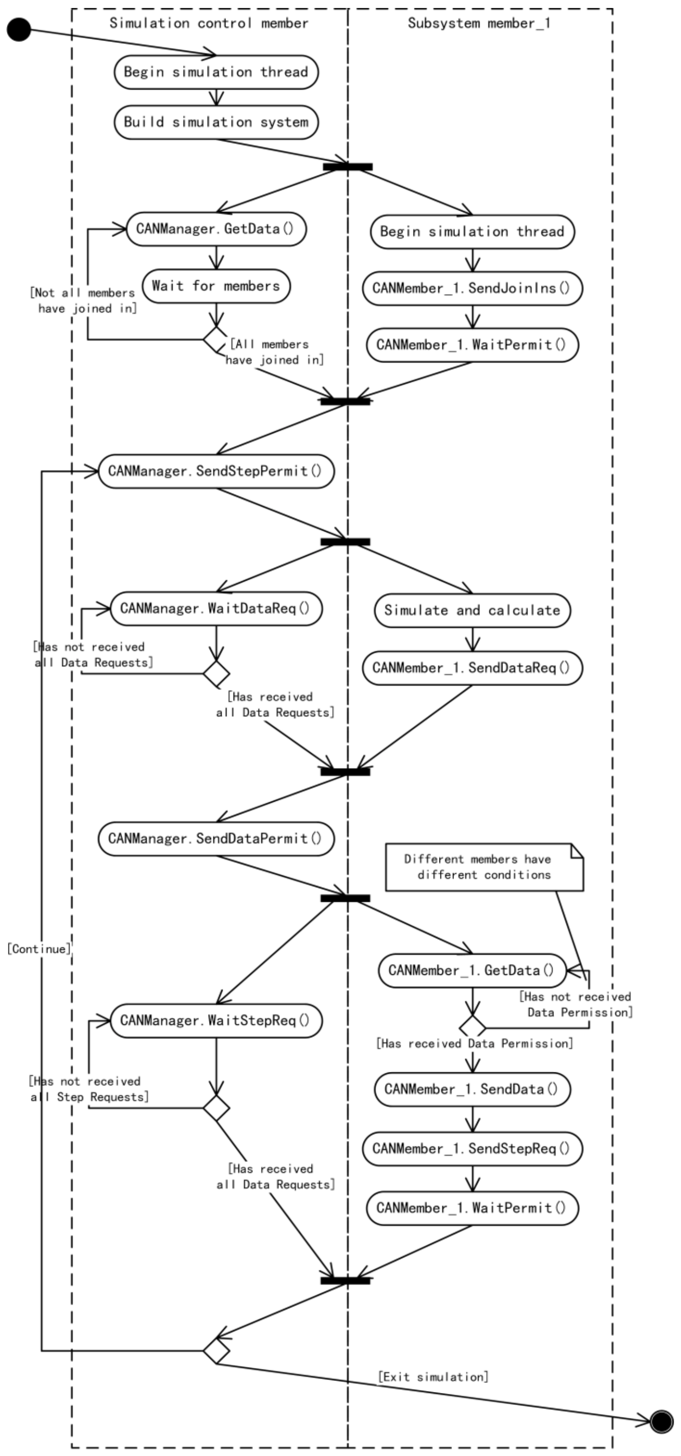

Simulation control member
Subsystem member_1
Begin simulation thread
Build simulation system
CANManager.GetData()
Wait for members
[Not all members have joined in]
[All members have joined in]
Begin simulation thread
CANMember_1.SendJoinIns()
CANMember_1.WaitPermit()
CANManager.SendStepPermit()
CANManager.WaitDataReq()
[Has not received all Data Requests]
[Has received all Data Requests]
Simulate and calculate
CANMember_1.SendDataReq()
CANManager.SendDataPermit()
Different members have different conditions
[Continue]
CANMember_1.GetData()
[Has not received Data Permission]
[Has received Data Permission]
CANManager.WaitStepReq()
[Has not received all Step Requests]
[Has received all Data Requests]
CANMember_1.SendData()
CANMember_1.SendStepReq()
CANMember_1.WaitPermit()
[Exit simulation]

Fig. 4. Simulation process

3) Control member waits for Step Requests from subsystem members after sending Data Permission. Then subsystem members will send data according to different preconditions:

a) The first subsystem member will send its simulation results after receiving Data Permission from control member, and sends its Step Request when results sending finishes. Then it waits for Step Permission. Because that function GetData() is called in function WaitPermit(), so the first subsystem member can receive the data sent by other members in its waiting time.

b) The second subsystem member will send its simulation results after receiving Step Request from the first subsystem member, and sends its Step Request when results sending finishes. Then it waits for Step Permission. The rest may be deduced by analogy, other members send its simulation results after receiving Step Request from the previous member. Thus the orderliness of data sending is insured, conflicts are avoided. And all other members are querying bus for fear of loss of data when a member sends data.

And when control member receive all Step Requests, it will send Step Permission. Then whole system will go back to step *2)*.

From above we see that simulation member plays a very important role in simulation process. It controls transfer sequence and handshake of whole simulation system. This is mainly materialized by two key nodes: Waits For Data Requests and Sends Data Permission, Waits For Step Requests and Sends Step Permission. Thus every member can not go to next step until all members finish their tasks in this step. As for the transmission mode of CAN bus, we can select either broadcast mode or filter mode according to the data demands between the members.

Simulation program frameworks of different members are similar. The framework is just an environment for simulation operation, and it’s commonly developed and maintained by a special programmer. Thus programmers answering for subsystem members can pay more attention to their modeling and simulation tasks. So development efficiency is raised and maintainability of system is enhanced.

5. Conclusion

At present the whole on-board simulation system has been established, and it meets all design requirements such as simulation management, transmission efficiency, reliability. Experiment results prove that the simulation framework can work stably with great efficiency and high flexibility. The framework can be easily accepted and used by programmers who are unfamiliar with CAN bus, thus development efficiency of system is raised. And it’s a very general CAN simulation framework, so it can be applied to simulation for various systems based on CAN bus. The following task is to extend the framework in order that actual hardware subsystems can join in.

References

[1] XIAO Longlong, CHENG Mousen, ZHANG Weihua. “Simulation and evaluation on data transmission of launch vehicle control system based on CAN bus,” *Journal of Projectiles Rockets Missiles and Guidance*, vol. 29, no. 2, pp. 256-260, 2009. (in Chinese)

[2] Liu Shufen, Sun Xin. “Study on application of CAN bus on satellite,” *Aerospace Control*, vol. 22, no. 6, pp. 79-83, 2004. (in Chinese)

[3] Gianluca Casarosa, Michele Apuzzo, Luca Fanucci, Bruno Sarti. “Characterization of the EMC Performances of the CAN Bus in a Typical System Bus Architecture for Small Satellites,” *9th EUROMICRO Conference on Digital System Design (DSD'06)*, 2006, pp. 338-345.

[4] Ferrer Albert, Parkes Steve. “Unified communication infrastructure for small satellites,” *60th International Astronautical Congress*, 2009, vol. 5, pp. 3776-3780.
[5] Yan Meizhi. “Research on built-in testing methodology of small satellite,” *60th International Astronautical Congress*, 2009, vol. 9, pp. 6991-6994.
[6] WANG Jihe, WANG Feng, LAN Shengchang, CAO Xibin. “Micro Core Based Double Satellite Real Time Simulation System,” *Journal of System Simulation*, vol. 20, no. 2, pp. 328-331, 2008. (in Chinese)
[7] Xu Fuxiang. *An introduction to satellite engineering*, Beijing: China Astronautics Press, 2003, pp. 9. (in Chinese)