Design and Implementation of Triple Modular Redundant System on Linux-Based On-Board Computer for CubeSat

Emir Husni, *Angga Putra,* *Nazmi Febrian*
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jl. Ganesha no. 10, Bandung 40132, Indonesia
ehusni@lskk.ee.itb.ac.id

Triple modular redundant system was implemented on Intel Galileo generation 1 as satellite on-board computer development hardware. This system implemented as hardware fault tolerant system for satellite on-board computer. Redundancy implemented in on-board computer using bash shell script. Simulation is carried out by testing various possible scenario. By using this redundant system, system reliability level is improved in terms of fault tolerance of on-board computer hardware. Each on-board computer operate simultaneously and supporting each other when failure happened.

Keyword(s) : fault tolerant system, satellite, redundancy, triple modular redundancy, on board computer.

I. Introduction

One of the most important elements in the design of embedded systems on the satellite system is hardware fault tolerance. The system must be able to maintain its performance in carrying out its mission despite the wide variety of faults on the space environment.

Various kinds of disturbances on the satellite caused in by high energy particles, such as single event effects, on the satellite orbital systems can cause soft to hard failure. In the redundant system, when one of the duplicated components fails, the other redundant components designed to replace the role of the failed components.

On satellite, redundant system design can improve the reliability of data storage and data transmission from the satellite to the ground station (GS). It is necessary to ensure good synchronization between redundant components. When the system is out of sync, the rate of data sent can be significantly reduced.

Today, many orbiting satellite still using non-internet based communication technology. On this research, development of redundant system is implemented on Linux-based single board computer that is Intel Galileo generation 1. By using Linux as operating system, modern technology such as CAN-over-IP and Interplanetary Overlay Network can be implemented on satellite system. This redundancy is implemented using bash shell script on each on-board computer.

II. System Analysis and Design

One of the stages in the design of redundant systems on satellite on-board computer (OBC) is to ensure that the system can continue its operation even though there is a failure in the system for example, OBC that is not functioning due to overheating or overcurrent caused by single event effects. Data retrieval and communication with GS must still be made, therefore, the systems must be designed to be fault tolerant.

There are various methods to design a fault tolerant system, i.e. replication, redundancy and diversification. All this methods can be implemented in form of hardware or software. On this development, fault tolerant systems are implemented in hardware redundancy of on-board computer.

Redundant system can be implemented on the dimensions of space (space redundancy) and the dimension of time (time redundancy). Implementation of time redundancy for example is in the form of retransmission of the TCP protocol if the data fails to be transmitted. While the implementation of space redundancy is by duplicating the use of components with the same function. System redundancy in OBC implemented due to the following considerations:

- **How important the component is in the system?** OBC is a very critical component as the central processing of all of the satellite activities, such as data acquisition, data processing, data transmission, power settings, scheduling, etc.
- **How likely the component to be fail?** Trapped protons in LEO contribute to the occurrence of single event effects on the electronic components that can resulted in errors of the electronic components.

- **How expensive to make a component into a fault tolerant?** By applying the redundancy system in OBC, the budget become more expensive. However, this is comparable to the level of reliability obtained. This will reduce the possibility of lost contact with the satellite when it is orbiting.

A. *Hardware Design*

Triple modular redundant system is implemented by using three single board computers as on-board data handling (OBDH) connected with each other through passive switch as follows:

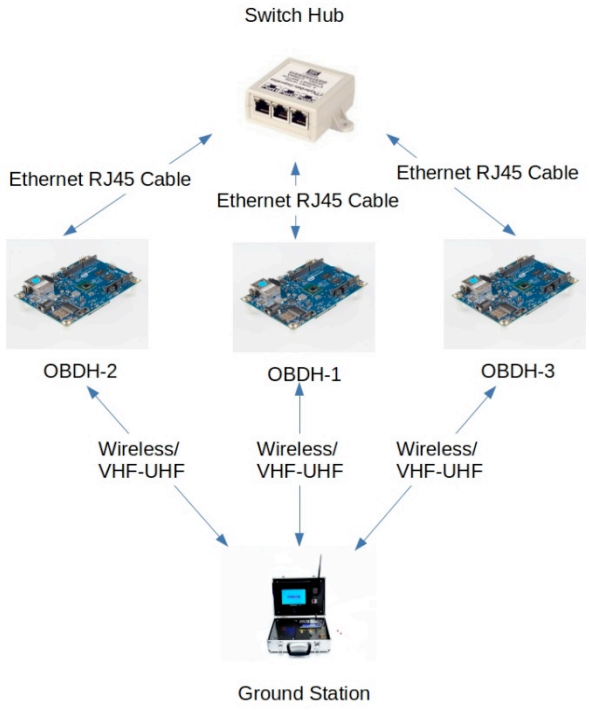

Figure 1 Hardware Design

These components are integrated to form a system. Each OBC has different priorities in transmitting and receiving data, while the order of the components of the highest priority to a low are:

1. OBC-1
2. OBC-2
3. OBC-3

Each of these OBC can replace another OBC that has a higher priority if the OBC with a higher priority fail.

B. *Software Design*

Software designed in development of this redundant system is primarily based on synchronization of connection among the three OBC. OBC-1 acts as the default primary OBC. As the primary OBC, its function is to sending data to GS and accepting and broadcasting software upgrade from GS to other OBC. Data transmitted from OBC-1 to GS is taken from other OBC. Below is a flow diagram of the OBC-1 function:

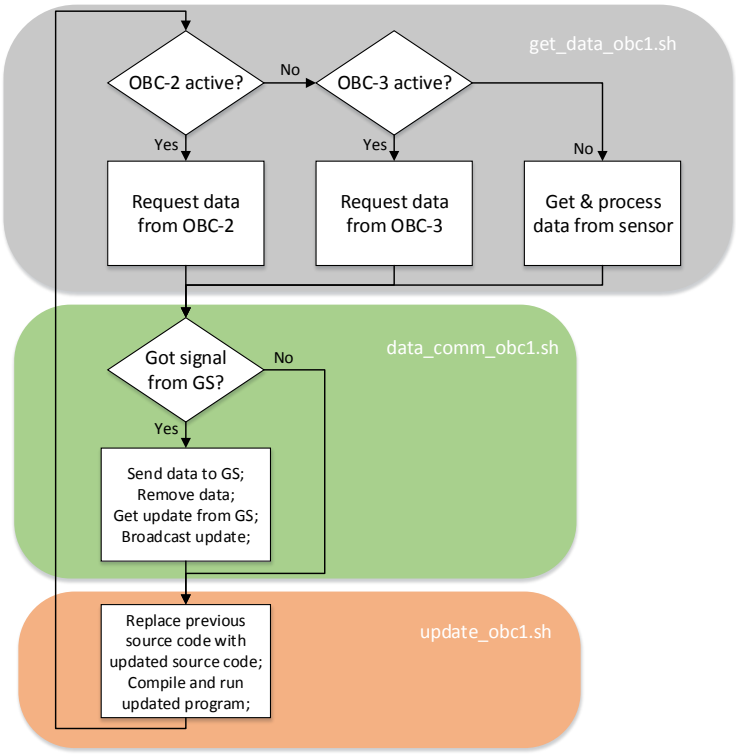

Figure 2 OBC-1 Flowchart

By default, OBC-2 is the supporting OBC that processing and transmitting platform data to OBC-1. OBC-2 will be taking OBC-1 role as primary OBC if OBC-1 fail. OBC-2 flowchart is as follows:

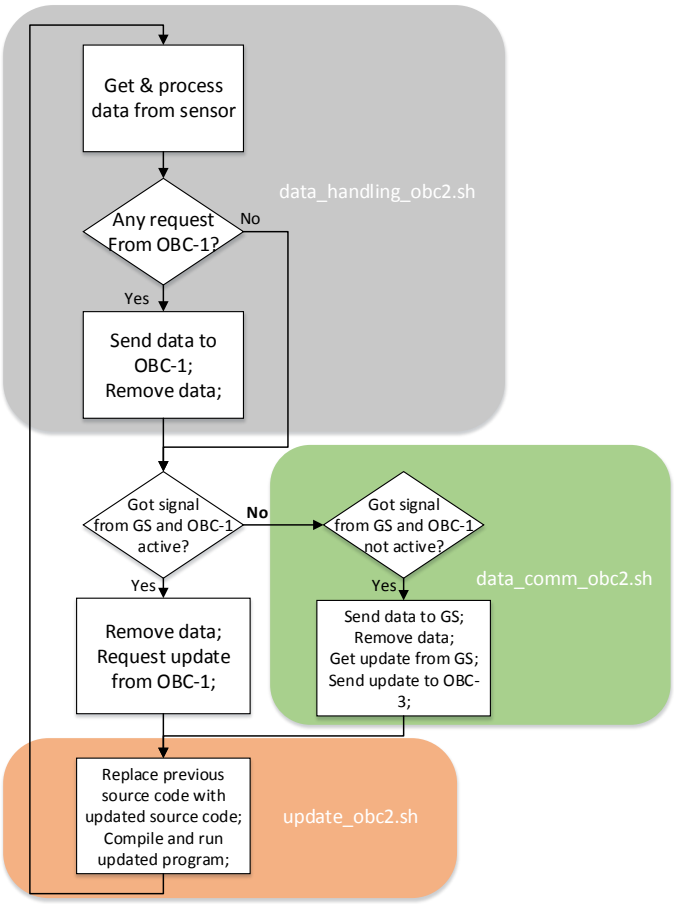

Figure 3 OBC-2 Flowchart

And then, there is OBC-3 as supporting OBC besides OBC-2. OBC-1 will request data to OBC-3 if OBC-2 fails. OBC-3 will take over OBC-1 role to communicate with GS if both OBC-1 and OBC-2 fails.

OBC-3 flowchart is as follows:

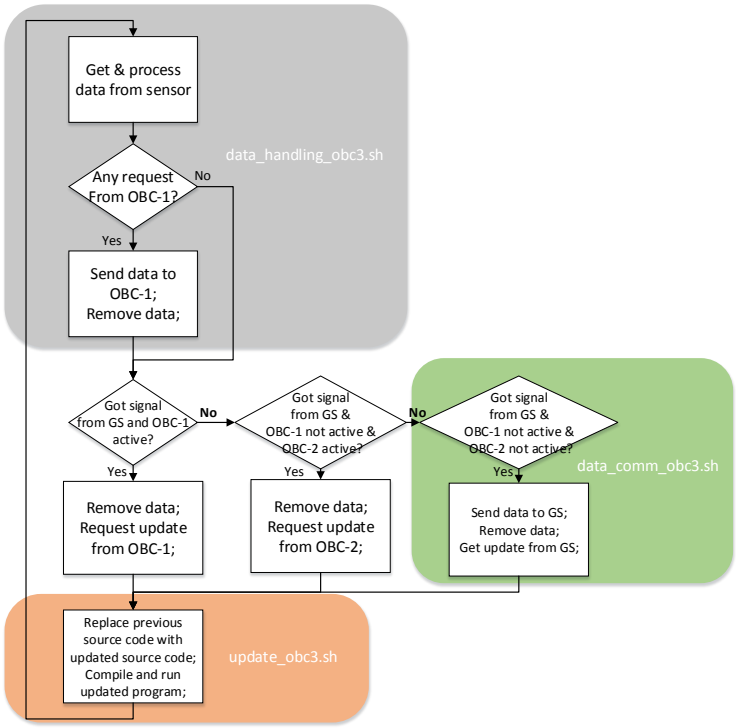

Figure 4 OBC-3 Flowchart

As seen in the flow diagram, the main script on each OBC work by calling other scripts that contains the program that runs each sub-section of each task (retrieval and dissemination of data, communication data to GS, and the process of updating the software).

In a redundant system, the system is designed to be mutually synchronized with one another via a data communication process. Through this communication exchange, each OBC mutually synchronized by knowing how other OBC circumstances and can directly take over tasks of failed OBC. Process that must be done by each of the OBC is defined in the script based on a variety of combinations possible scenarios. In short, this scenario can be divided into two different conditions, i.e. when connected with GS and when not connected. The combination of scenarios that may occur when the OBC is not connected with the GS are as follows:

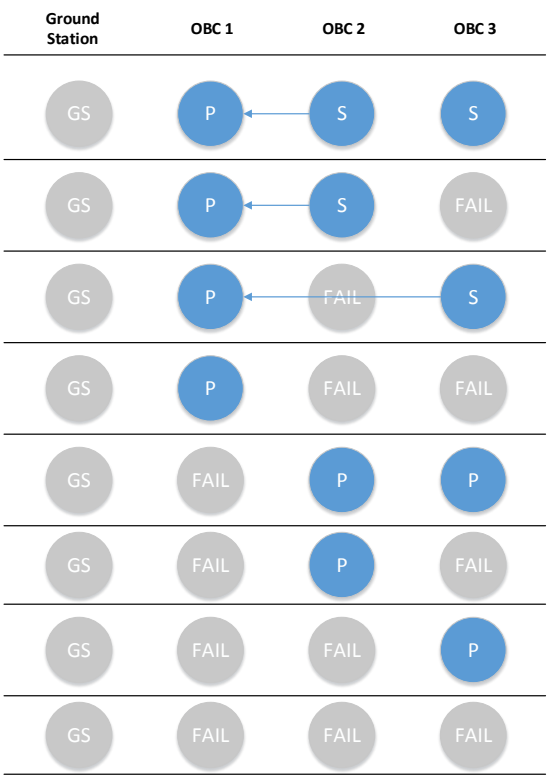

Figure 5 Redundancy Scenario : Unconnected from GS

Above picture shows us data communication path between each OBC on various scenarios (arrow sign). Primary OBC (P) will request data from secondary OBC (S) in case of scenario 1 to scenario 3.

And scenarios that may happened when OBC connected with GS are:

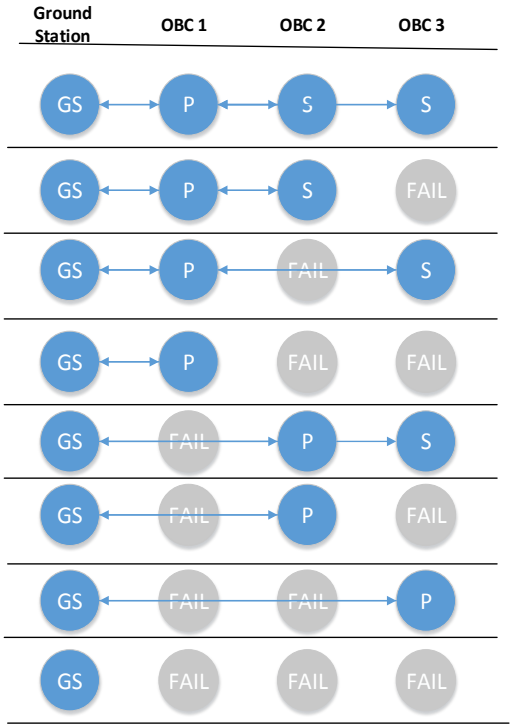

Figure 6 Redundancy Scenario : Connected from GS

III. TESTING

The testing was done by simulating data transmission between OBC via Ethernet cable, without corrupt and delay. The purpose of this test is to know the performance of data request hit and miss between OBC, total packet acquired between OBC and first accumulated packet sent time between OBC and GS on normal environment.

The testing was done on all scenario except the scenario where all OBC fail simultaneously. This test was done by transmitting dummy data and repeating each scenario testing by five times to obtain the average result. Each scenario was tested by using Low Earth Orbital time consideration, which is around 100 minutes. OBC-GS communication duration assumed to be 10 minutes. Then there is 90 minutes duration that each OBC disconnected with GS. If every scenario except when all OBC fail assumed to be happened during that time, then each scenario has around 13 minutes duration.

Before the test is done, OBC was configured beforehand in order to carry out functions in accordance with a predetermined mission. The configuration is carried out as follows:

No	Configuration	Status
1	Checking networks availability among OBC	Success
2	Checking networks availability between OBC and GS	Success
3	Giving priority to each OBC	Success
4	Task takeover when higher	Success

	priority OBC fails	
5	Filename and directory management	Success
6	Data transmission between OBC using 64-bit CAN over IP	Success
7	Data transmission between OBC and GS using interplanetary network	Success
8	Files compression in one file with tar.gz extension before sending process	Success
9	Limiting delivery time allocation for each session to avoid the cessation of the whole process in case of failure at the time of delivery	Success
10	Software update system management based on filename	Success
11	Giving information of the process phase name to shown in the terminal	Success
12	Synchronized ransitions between OBC functions through the provision of delay	Success

A. *Testing Results : OBC Disconnected from GS*

When the OBC is not connected with the GS, the operation is done in the form of data transmission between OBC operations. The results obtained in the form of data sending reliability between OBC in various situations. Each scenario is tested on 13 minutes duration.

The test results demonstrate the success of the system tolerances to failure to keep the transmission of data despite the failure that caused the malfunctioning of OBC. The data will continue to be acquired until all OBC fail.

OBC-1	OBC-2	OBC-3	Request Hit	Request Miss	Total Packet Acquired (byte)
P	S	S	13	5	6,299,264
P	S	FAIL	12	3	4,600,751
P	FAIL	S	11	10	4,600,860
P	FAIL	FAIL	no request	no request	3,827,809
FAIL	P	P	no request	no request	5,219,563
FAIL	P	FAIL	no request	no request	5,219,608
FAIL	FAIL	P	no request	no request	3,410,318

B. *Testing Results : OBC connected with GS*

At this time in addition to a process of exchanging data between OBC, is also a process of sending data to the GS. In addition, the OBC which serves as the main OBC will play spread shipment of GS data, which is the script and the source code that has been updated to then be compiled and run on each targeted OBC. On this case, the first packet size that is accumulated during OBC-GS disconnected is measured by using maximum packet size acquired, that is 6,299,264 byte per 13 minutes. Then on 90 minutes duration the accumulated packet size is around 44.1 MB. This is the first packet data size to be delivered to GS, which sent time is shown on table below.

OBC-1	OBC-2	OBC-3	First Packet Sent Time (s)	Update Success?
P	S	S	15.03	YES
P	S	FAIL	14.57	YES
P	FAIL	S	14.8	YES
P	FAIL	FAIL	14.2	YES
FAIL	P	P	15.03	YES
FAIL	P	FAIL	15.44	YES
FAIL	FAIL	P	15.57	YES

The test results show the success of system redundancy by successfully transmitting data from OBC to GS despite the failure that happened on two from three OBC at the same time. Software upgrade successfully delivered and processed, but only can be implemented on OBC that is still active during the delivery process of data from GS to OBC.

Testing appearances on Linux terminal, are shown on the following pictures:

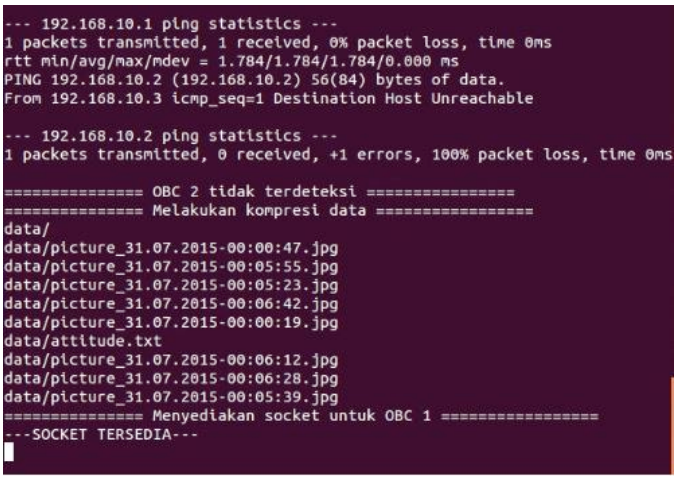

Figure 7 OBC-3 took over OBC-2 roles in Transmitting Data to OBC-1

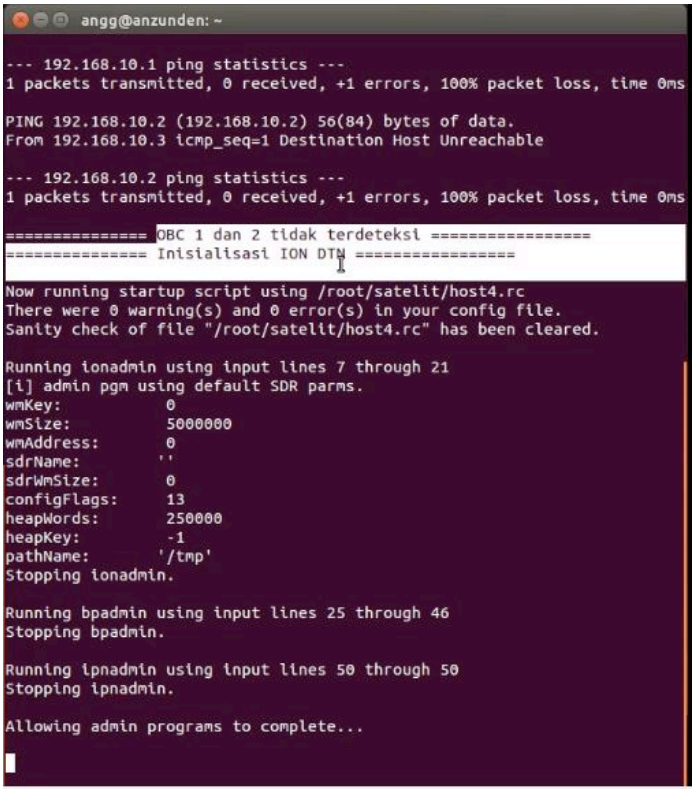

Figure 8 OBC-3 as primary OBC for communicating with GS when both OBC-2 and OBC-1 undetected (fail)

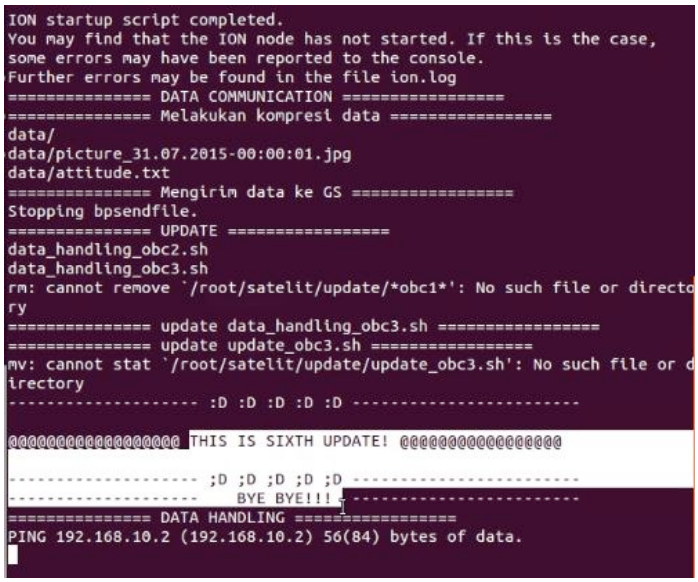

Figure 9 Script succesfully updated

IV. Conclusion

The conclusion of design and implementation of triple modular redundant system on linux-based on-board computer for LEO satellite is as follows:

- Redundant system can operate synchronously to establish data communication between OBC
- Data will always be acquired and transmitted as long as all three OBC aren't failed simultaneously
- Remote software update is delivered and processed successfully on all scenario

References

1. Dubrova, E. (2013). "Fault-Tolerant Design", Springer, 2013
2. Avizienis, A. (1976). "Fault-Tolerant Systems", IEEE Transactions on Computers, vol. 25, no. 12 Jyh Shing Roger Jang. 1997. Neuro Fuzzy and Soft Computing.
3. Johnson, B. W. (1984). "Fault-Tolerant Microprocessor-Based Systems" IEEE Micro, vol. 4, no. 6.
4. Laprie, J. C. (1985). "Dependable Computing and Fault Tolerance: Concepts and Terminology", Proceedings of 15th International Symposium on Fault-Tolerant Computing (FTSC-15).