Error Detection and Correction System (EDAC) of On Board Data Handling (OBDH) in Real Time Operating System Behaviour

Haryono[1], Agfianto Eko Putra[2], Jazi Eko Istiyanto[2] and Agus Harjoko[2]

Satellite Centre[1], National Institute of Aeronautics and Space (LAPAN),
Jl Satelit KM 04 Rancabungur Bogor
Department of Computer Science, Faculty of Mathematics and Natural Sciences[2],
Gadjah Mada University (UGM), Sekip Utara Kotak Pos 21 Bulaksumur Yogkayakarta
[1]*Email: haryono@hryn.net*

ABSTRACT

The satellite requires the support of a robust sub system. On Board Data Handling (OBDH) is the core function of the satellite subsystem and has to be error free in managing the operation of the satellite. It should withstand the harsh environmental conditions in space that has a lot of hazards caused by radiations. In view of these two conditions, the OBDH design should be able to manage the operation and overcome the hazards of radiation. In order to manage the operation Real Time Operating System (RTOS) was applied. RTOS was able to manage the task efficiently and effectively. In the aerospace domain, RTOS has become popular because of its strength in managing the operating system. Error Detection and Correction System (EDAC) system was applied to make OBDH more robust. This paper will discuss the implementation of the EDAC system in tandem with the RTOS behaviour to manage the operation and increase the robustness of the system. The findings show that OBDH can be programmed successfully using RTOS to handle critical and robust operations.

***Keywords**: EDAC, uCos, Real Time Operating System, SH7145, OBDH*

ISSN 1675-7009

INTRODUCTION

Super loop can be replaced by a multitasking system, referred to as Real Time Operating System (RTOS). In the aerospace domain, RTOS has become popular because of its reputable capability in managing the operation system [1]. When programming the embedded system initially, the super loop method is used, but this super loop method displayed some weaknesses. The super loop had trouble when a parallel manner function was executed. This could be because in the super loop the executing function is naturally in serial program; the time taken to switch to other functions is influenced by ISR [2]. However, with RTOS, each task can run concurrently. The task also gives guarantees that the function of ISR on both sides to be in an active state all the time [2].

There are some other types of RTOS, Pre-emptive and Cooperative Scheduling. Both systems have their advantages and disadvantages. This study will look at the possibility of applying the most compatible system to apply in the OBDH. Cooperative Scheduling implies working together, the resource of the system should be maintained cooperatively [2]. It has to be cooperative in switching the task, because of the cooperative aspect, the disadvantage is the highest priority task cannot run directly until the low priority task gives the resources to other task explicitly [2]. Pre-emptive Scheduling, low priority operations can be switched directly to the higher priority or ISR without permission from the task being performed. This method will lead to complexity of the system, the task is pre-empted and all information should be saved properly before moving on to the next task, hence requiring more resources [2]. The efficiency of the responsiveness reduces waiting time for other operations to use the system. In terms of our OBDH, we need the system to be responsive and react immediately therefore, the Pre-emptive Scheduling was selected and incorporated in the design.

Hamming error correcting codes was used to protect memory because it is relatively inexpensive, very efficient, and creates less redundancy [3]. The advantage of the Hamming code is that, it is faster and simple, coding does not require a lot of calculation. The other reasons are, it is useful on short messages, synchronizes with this task to handle command from ground segment and handle Global Positioning Satellite data. It has been used by HAUSAT-2 satellite [4] and ISTNanosat-1 satellite [5]. Therefore, the Hamming code was chosen to store the data in to memory for this project.

ISSN 1675-7009

By applying the RTOS and EDAC, the satellite can be programmed easily, efficiently and more robust.

EDAC System Connectivity Design

This section will discuss the method of EDAC software in handling Global Positioning Satellite (GPS) and Ground Segment (GS) command data, the detail flowchart is shown in figure 1.

1. Receive data command from GS trough Telemetry Tracking and Command (TTC) and from GPS receiver.
2. Decode and Save data from GS and GPS receiver using Hamming Code in memory.
3. Open and Decode data from memory and then send to appropriate destination,

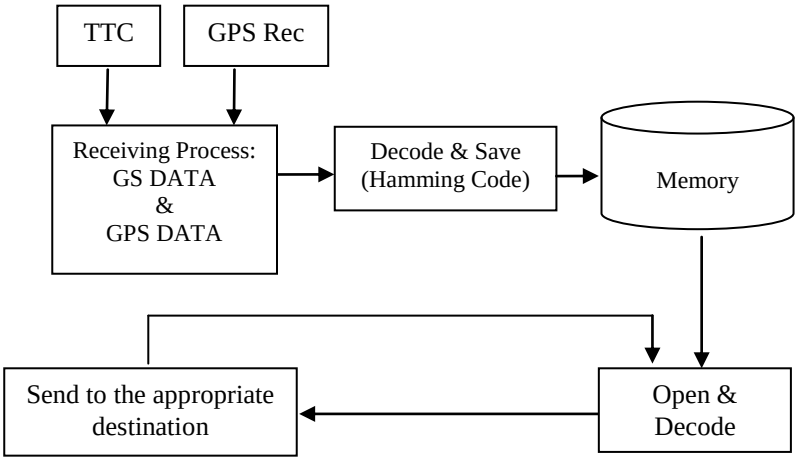

Figure 1. Software Flowchart of Hamming Code Implementation

RTOS Design

This section will discuss the design of RTOS in OBDH in handling the various tasks. This project is focused on three important task; first is handling the GPS data, the Second is handling the GS data and the third is handling the EDAC system for the GS data and GPS data. SH7145 from Renesas is the microcontroller used, the tool is High-performance Embedded Workshop (IDE) HEW and the tool chain is Renesas sh2. Figure two describes the operation.

ISSN 1675-7009

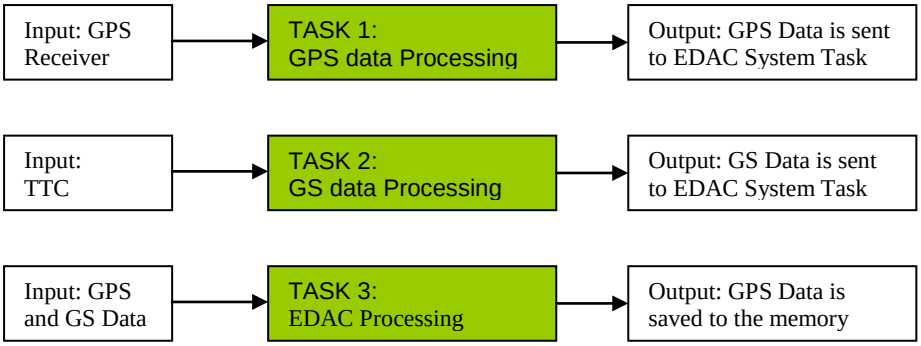

Figure 2: Task being running

EDAC Implementation

GS and GPS Data received will be encoded and saved in the banks memory. In order to protect the data from errors the hamming code was implemented. This section will discuss algorithm of the Hamming Code that as in [6]. Number of hamming bits depends on the number of message bits. "n" is total number of bits transmitted. "m" is total number of hamming bit. The formula is given below;

$$\mathbf{2^m \geq n + 1}$$

Example: Message has 4 data bits

$2^m \geq (4 + m) + 1$

$2^3 \geq (4 + 3) + 1$

Therefore, 3 is the smallest number of hamming bit.

Encoding Part

Original message => 1101 (4 bits)

Step 1: Compute the no. of hamming bits (m)

$2^3 \geq (4 + 3) + 1$; m = 3 ; therefore n =7

Step 2: Insert the hamming bits (H) into the original message stream. By following the formula in the table 1 and calculating the data in to the table 2.

P1 (Parity 1) = D3 XOR D5 XOR D7

P2 (Parity 2) = D3 XOR D6 XOR D7

P4 (Parity 3) = D5 XOR D6 XOR D7

Code stream that will be sent: D7 D6 D5 **P4** D3 **P2** **P1**

Example: Message bit: **1101**

Step 3: The code bit stream: **1100110**

ISSN 1675-7009

Decode Part

Bit Position : 7 6 5 **4** 3 **2 1** (7 bits)

Actual Data : 1 1 0 **0** 1 **1 0**

Occurring error : 1 1 1 **0** 1 **1 0**

Supposed position number 5 is error.

Step 1: Calculating the message stream

C1 = P1 XOR D3 XOR D5 XOR D7

C2 = P2 XOR D3 XOR D6 XOR D7

C4 = P4 XOR D5 XOR D6 XOR D7

C1 =0^1^1^1=1

C2 =1^1^1^1=0

C4 =0^1^1^1=1

Step 2: Finding the error, if the calculation result is not "0" the data error, so flip the data according the data calculation. **C4C2C1 = $1x2^2 + 0x2^1 + 1x2^0 = 5$**

Now, we can **correct the error** bit in position 5, so definitely it is not 1 but 0.

RTOS Implementation

In this project, three important tasks were created: TASK 1: GPS data Processing, TASK 2: GS data Processing, and TASK 3: EDAC Processing.

The RTOS we have selected is **uCOS** from Micirum, it is Pre-empted system, well known RTOS, has been validated by FAA [1] and proper for safety-critical systems for leading aerospace. Each TASK will run simultaneously.

Below is the source for the uCOS RTOS in creating each task:

```
OSTaskCreateExt(GS_Data_Process,    // task name
(void *)0,
(OS_STK *)&AppTask2Stk[APP_START_TASK_STK_SIZE - 1],
APP_START_TASK1_PRIO, //Task Priority
APP_START_TASK1_ID, //Task ID
(OS_STK *)&AppTask2Stk[0], // pointer to the lowest stack
APP_START_TASK_STK_SIZE,  // stack task size
(void *)0,
OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR); //allow to check stack

OSTaskCreateExt(GPS_Data_Process,
(void *)0,
(OS_STK *)&AppTask1Stk[APP_START_TASK_STK_SIZE - 1],
 APP_START_TASK2_PRIO,
 APP_START_TASK2_ ID,
```

ISSN 1675-7009

```
(OS_STK *)&AppTask1Stk[0],
 APP_START_TASK_STK_SIZE,
(void *)0,
 OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

OSTaskCreateExt(EDAC_Process,
(void *)0,
(OS_STK *)&AppTask3Stk[APP_START_TASK_STK_SIZE - 1],
APP_START_TASK3_PRIO,
APP_START_TASK3_ ID,
(OS_STK *)&AppTask2Stk[0],
APP_START_TASK_STK_SIZE,
(void *)0,
OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);
```

On this source code above, each task created and given different priority, below are the detail priority:

TASK 1: GS data Processing (Highest Priority): to ensure the command data from the ground should be handled first, the command is critical for example to shutdown the sub system because the battery is low.

TASK 2: GPS data Processing (Middle Priority): to consider GPS Data Processing and to ensure that the data can be acquired properly.

TASK 3: EDAC Processing (Low Priority) considered as low priority, this task should run correctly, but the priority impact should be set to the natural sequence. We consider the two tasks above more risky if the task is not comparable to the EDAC Processing. Data received in the buffer can be interrupted by other task, upon completion of the task the system will continue the EDAC Processing.

Programming each Task

GS_Data_Process TASK will work upon receiving the message from the ISR receiver data (Rx2), the task will be suspended and delayed in each 100 millisecond. The GS data will be sent to the **EDAC_Process** TASK to be encoded with hamming code.

```
void  GS_Data_Process (void *p_arg)
{
        unsigned char *msg;
        char err;
        while (1) {
                msg = (INT8U *)OSQPend(CommQRX2, 0, &err); //waiting the message from ISR
                if (err == OS_NO_ERR) {
                        //FUNCTION PROCESS HERE
                        OSQPost(GS_DATA, (void *)&data); //post the data to EDAC task
                        OSTimeDlyHMSM(0, 0, 0, 100);
                } else { sci1(err_gs); }
```

ISSN 1675-7009

```
        }
}
```

GPS_Data_Process TASK will work after receiving the message from the ISR receiver data (Rx3), the task will be suspended and delay in each 100 millisecond. After calculating, the GPS data will be sent to the **EDAC_Process** TASK to be encoded with hamming code.

```
static  void  GPS_Data_Process (void *p_arg)
{
        unsigned char *msg;
        char err;
        while (1) {
                msg = (INT8U *)OSQPend(CommQRX3, 0, &err); //waiting the message from
        ISR
                if (err == OS_NO_ERR) {
                        //FUNCTION PROCESS HERE
                        OSQPost(GPS_DATA, (void *)&data); //post the data to EDAC task
                        OSTimeDlyHMSM(0, 0, 0, 100);
                } else { sci1(err_gs); }
           }
}
```

EDAC_Process TASK will work after received the message from the **GS_Data_Process TASK and GPS_Data_Process TASK**, before that the task will be suspended and having delay each 10 milisecond.

```
static  void  GPS_Data_Process (void *p_arg)
{
        unsigned char *msg_gs;
        unsigned char *msg_gps;
        char err_gs;
        char err_gps;
        while (1) {
                *msg_gs = (INT8U *)OSQPend(GS_DATA, 0, & err_gs);
                if (err_gs == OS_NO_ERR) {
                        //FUNCTION PROCESS HERE
                        OSTimeDlyHMSM(0, 0, 0, 10);
                } else { sci1(err_gs); }

                *msg_gps = (INT8U *)OSQPend(GPS_DATA, 0, &err_gps);
                if (err_gps == OS_NO_ERR) {
                        //FUNCTION PROCESS HERE
                        OSTimeDlyHMSM(0, 0, 0, 10);
                } else { sci1(err_gs); }
           }
}
```

ISSN 1675-7009

TESTING AND RESULTS

In super loop method, it is easy to face data corruption due to the neglect of the non *re-entrancy* function. As shown in figure 3, the data becomes corrupt because simultaneous use of the non *re-entrancy* function, the process is interrupted while sending the data which has not finished transmitting yet. We can see data that "abABCDEF..." lower case letter sent until "z", is interrupted by upper case letter. We have analyzed the EDAC super loop method [7], we need to be more careful in using non *re-entrancy* function in super loop method.

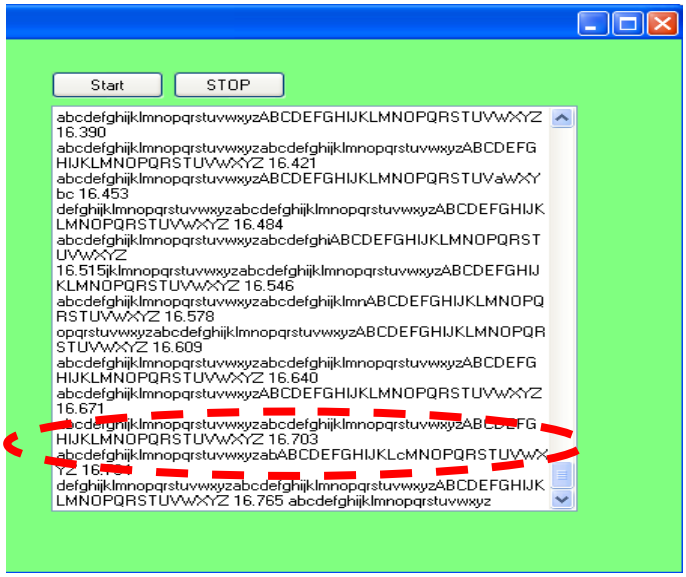

Figure 3: Super loop Method facing error

The difference when using RTOS as described in this paper, data that comes in to be processed, should be encoded /decoded and sent to the appropriate destination and will be easier to manage. The aim of this test was to check whether acquisition, storing and opening GS and GPS data using hamming code method is working correctly or not. Moreover can RTOS manage operation of the OBDH in applying the EDAC system? Test diagram in Figure 4. Figure 4 shows GS data and GPS data entered to the OBDH at the same time and saved to the memory, before saving to memory the data introduces a random error and then opens and sends to the Computer to be observed.

ISSN 1675-7009

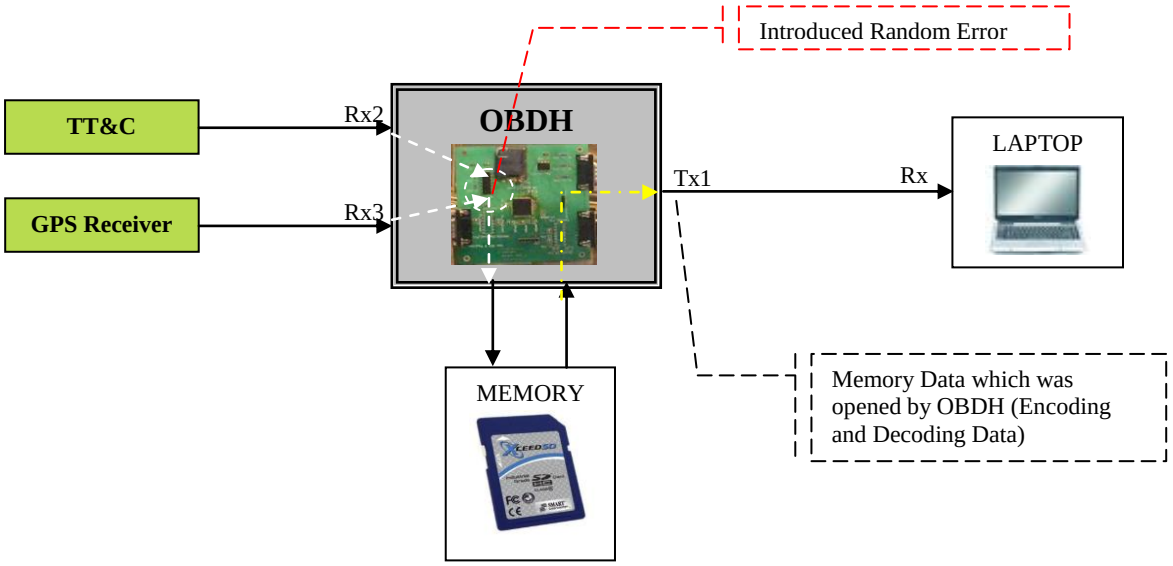

Figure 4: Block Schema Software Test of EDAC System

From the test applied: the RTOS can handle EDAC System calculation and perform each task correctly. The GPS receiver and TTC data was acquired concurrently. Even when errors were introduced, the hamming code can correct the data which was correctly saved in the memory by saving the data and each time getting the data from both destination and applying the EDAC system to the data successfully as well. EDAC system test can be found in the [7].

CONCLUSION

Using RTOS the data received concurrently can be managed and processed safely. In RTOS behaviour can be classify the task easier whether in super loop method, care need to be taken when using non re-entrance function to avoid data corruption. The EDAC system to handle GS Command and GPS data that have been implemented can detect and correct the

ISSN 1675-7009

data successfully in RTOS behaviour. A character can be divided into two hexadecimal forms to implement Hamming Code, therefore, the data error can be corrected up to two bits.

REFERENCES

[1] Mingsong, L.v., Guan, N., Zhang, Y., Chen, R. (2009). WCET Analysis of the uC/OS-II Real-Time Kernel, *in the Proceedings of the 2009 International Conference on Computational Science and Engineering*, Vancouver, Canada, Vol. 2, pp. 2.

[2] Andrew E. Kalman. (1995-2010). *SALVO User Manual version 4.2.2*, Pumpkin, Inc, San Francisco, USA, pp. 14-21.

[3] McFarland, C.A., December 11, 1994, Computer Subsystem, http://citeseerx.ist.psu.edu/pdf, pp. 13.

[4] Jung, Wan, J., Chang, Keun, Y., (2005). HAUSAT-2 Satellite Radiation Environment Analysis and Software Hamming Code EDAC Implementation, *Journal of Astronomy and Space Sciences*, Vol. 22, no. 4, pp. 537-558

[5] Jo˜ao, A.H.F. (2012). *Istnanosat-1 Heart Processing and Digital Communications Unit*, Dissertation of the Master Degree in Communication Networks Engineering, Lisboa Portugal.

[6] M. Dipperstein, September 15, 2012, Hamming (7,4) Code Discussion and Implementation, http://michael.dipperstein.com/hamming/index.html.

[7] Haryono (2010). Hamming Code Implementation for Satellite On Board Data Handling (OBDH), *in the Proceedings of the 2010 VIII International Scientific-Research Conference of Student*, Bishkek, pp. 148-150.

ISSN 1675-7009