STUDY OF DETERMINISM OF DATA INTEGRITY DURING I/O DATA EXCHANGE BETWEEN TASKS AND DEVICE

Cheol-hea Koo*, Su-hyun Park*, Soo-yeon Kang*, Koon-ho Yang*, Sung-bong Choi**

*Communication Satellites Dept, COMS Program Office, Korea Aerospace Research Institute, Daejeon, 305-333, Korea
E-mail : chkoo@kari.re.kr
**Communication Satellites Dept, Korea Aerospace Research Institute, Daejeon, 305-333, Korea

ABSTRACT In this paper, the method which can protect the situation of possible data corruption when collision has happened during I/O data exchange between device and tasks is presented. Also, an example diagram of mechanism according to this introduced method is shown and the effect and merits and demerits of the method is evaluated.

KEY WORDS: Determinism, Data processing, Predictability, Data integrity

1. INTRODUCTION

Most processing power is consumed by improper handling of input/output process on device. So I/O interface between software and device is crucial for overall software performance evaluation. Many research has been performed so that this kind of overhead could be minimized. And improper handling can cause stuck of overall processing like as deadlock. Priority inversion is the most representative behaviour when a corruption between the usage of shared resources is happened.

These problem can be cured by proper scheduling between task and devices. First of all, predictability should be guaranteed for the periodic and sporadic I/O processing of devices. This paper presents the method how to save the consumption of processing power during acquiring the I/O data from the devices through tasks.

2. FUNDAMENTAL PROCESISNG OF TASK

The behaviour of tasks which control the I/O processing from device consists of three parts as we can see the Figure 1.

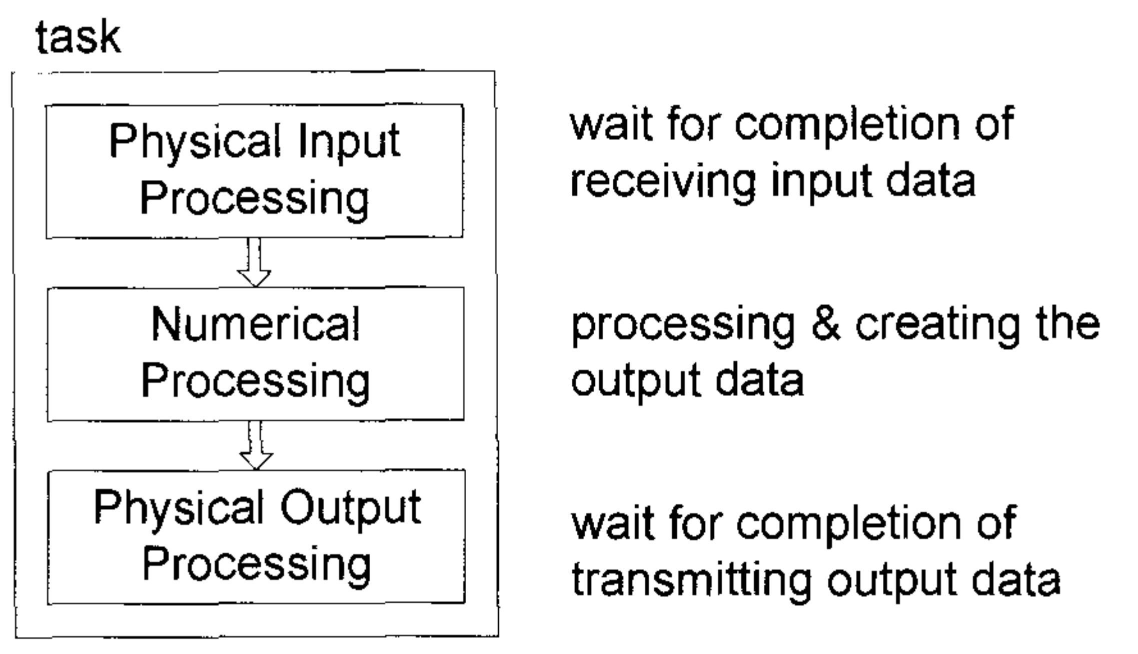
task
Physical Input Processing
Numerical Processing
Physical Output Processing
wait for completion of receiving input data
processing & creating the output data
wait for completion of transmitting output data

Figure 1. General structure of control task

First part is Physical Input Processing. At this part, task will wait for completion of receiving input data. The waiting time should be minimized because it is the main source of loosing processing power. Second part is Numerical Processing. At this part, task computes the received data and commands the proper action. The last

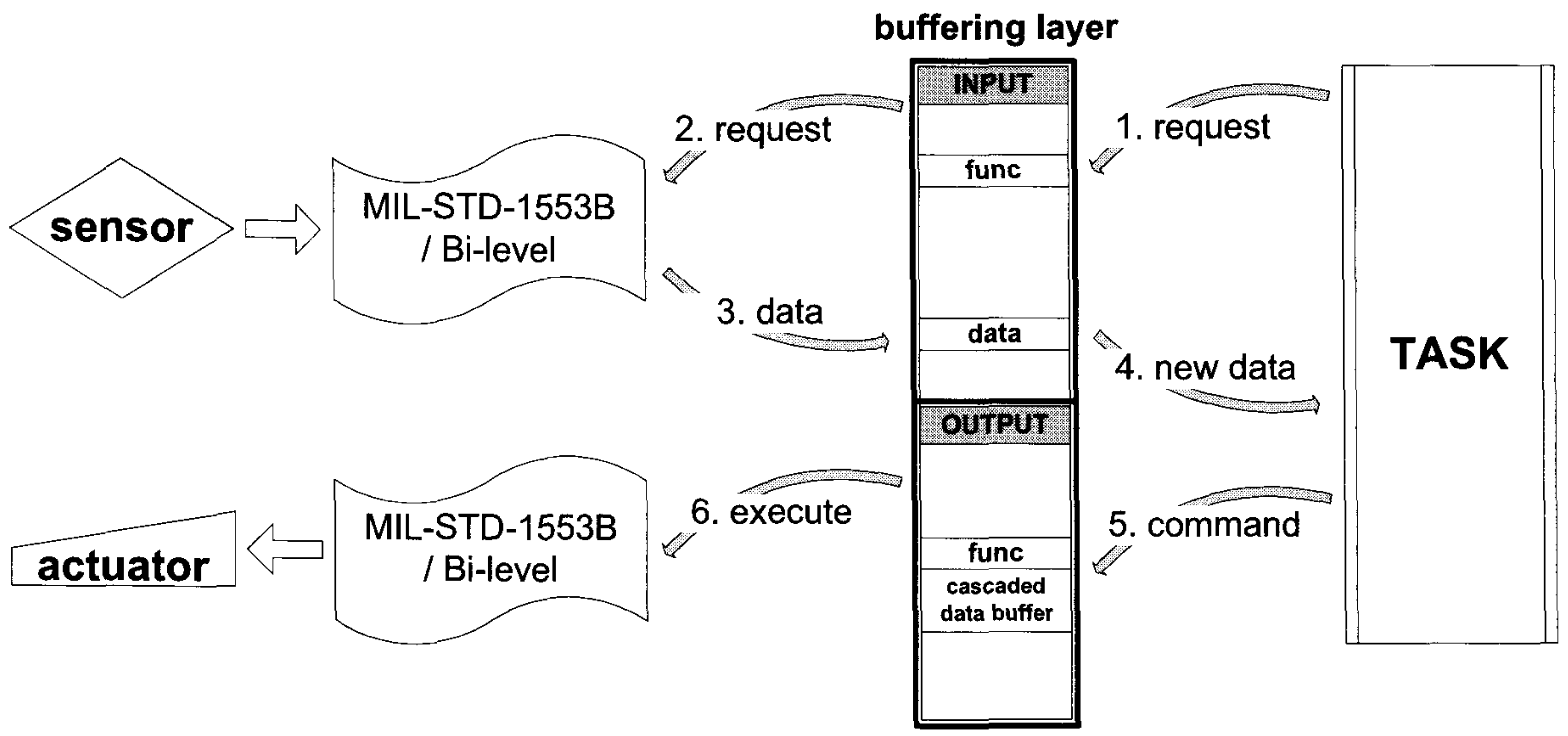

Figure 2. Data exchange diagram

Third part is Physical Output Processing. At this part, task transmits the command through the I/O device. For the fastest response of interface with hardware task shall not wait the completion of hardware commanding.

During all of these three process, waiting time should be minimized to use the processor effectively. So, input data should be prepared at the input buffer before task needs the input data and output data should be processed without any intervention of task.

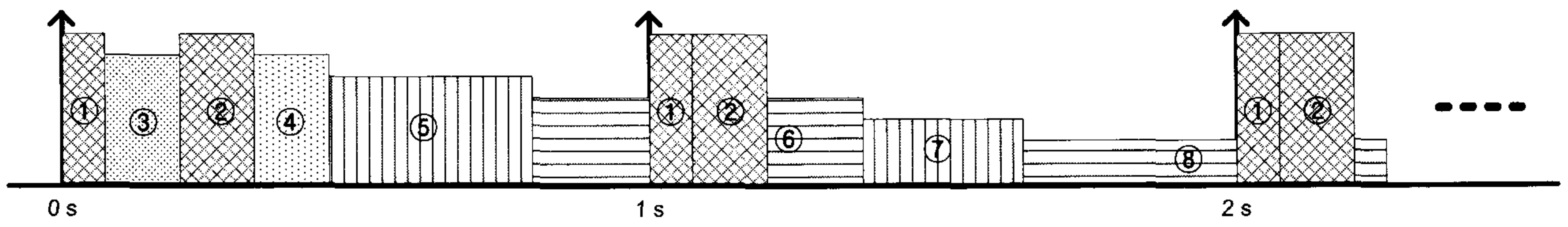

Figure 3. Task scheduling allocation

As shown in Figure 2, task will receive the needed data from sensor through several application layer. In satellite flight software, most of data line is MIL-STD-1553 or bi-level bus.

With this multiple interfacing layer, sensor and task is divided into independent operational unit. This interfacing layer will manage the whole process to ensure the predictability.

3. PREDICTABILITY ENHANCEMENTS

The enhancement of predictability during I/O data exchange between task and device could be achieved through scheduling of task with regard to minimizing hazard and avoiding data collision.

3.1 Scheduling

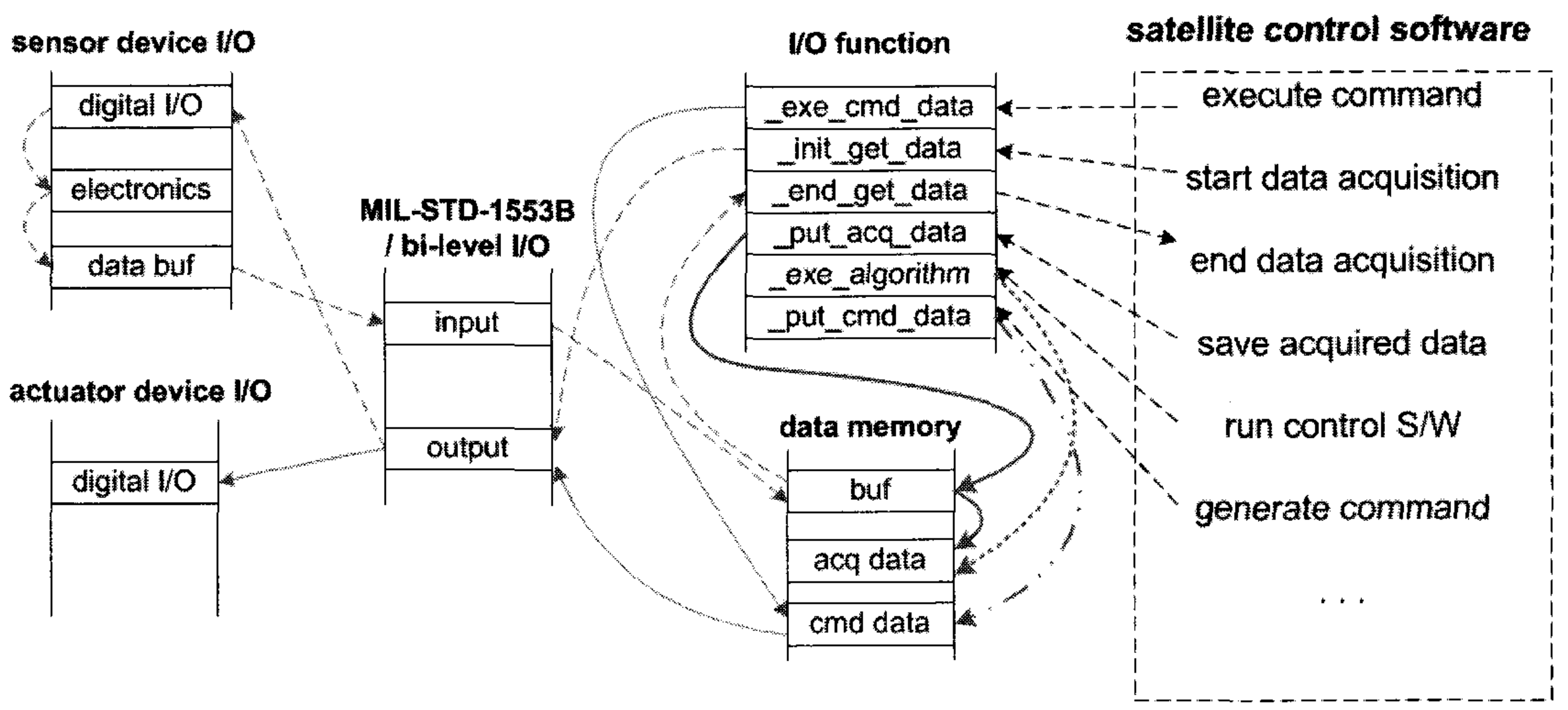

Figure 4. Task scheduling allocation

In real time systems, race between tasks which own the shared resource is the main cause of various problem. So, the problematic hazard is below,

- Potential bad data integrity when shared resource is pre-empted by another task
- Unnecessary dead time from data creation to data processing

These potential hazards are mainly introduced by race condition between tasks for acquiring CPU or shared resources. So, the possibility of these hazards must be excluded in the application. Consequently the optimal scheduling is needed at the expense of reactivity of I/O data processing. But this approach is very good for satellite application.

In satellite operation software, predictability has a points rather than reactivity. So, this paper presents the method which separate the data creating part from data reading part completely to ensure the predictability.

Basically tasks which are related to data processing consists in task which use data and task which read data. So, these tasks can not be interpolated in time domain. The scheduling of these tasks should be configured by below manner.

The tasks in the Figure 3 are divided to fast periodic and slow periodic execution rate. And the internal operation requirement are listed below,

- fast execution rate task has higher priority
  - (1), (2) : it is not related to data acquisition, or related to command data processing
  - (3) : it starts the acquisition of data which is needed by task (6) ~ (8)
  - (4) : it waits the completion of acquisition of data which is needed by task (6) ~ (8)
- slow execution rate task has lower priority
  - (5) : it saves the acquired data to the buffered memory, or processes command data

- (6), (7), (8) : it is satellite control task. It fetches the acquired data from the buffered memory, performs control algorithm with this data, and generates command data

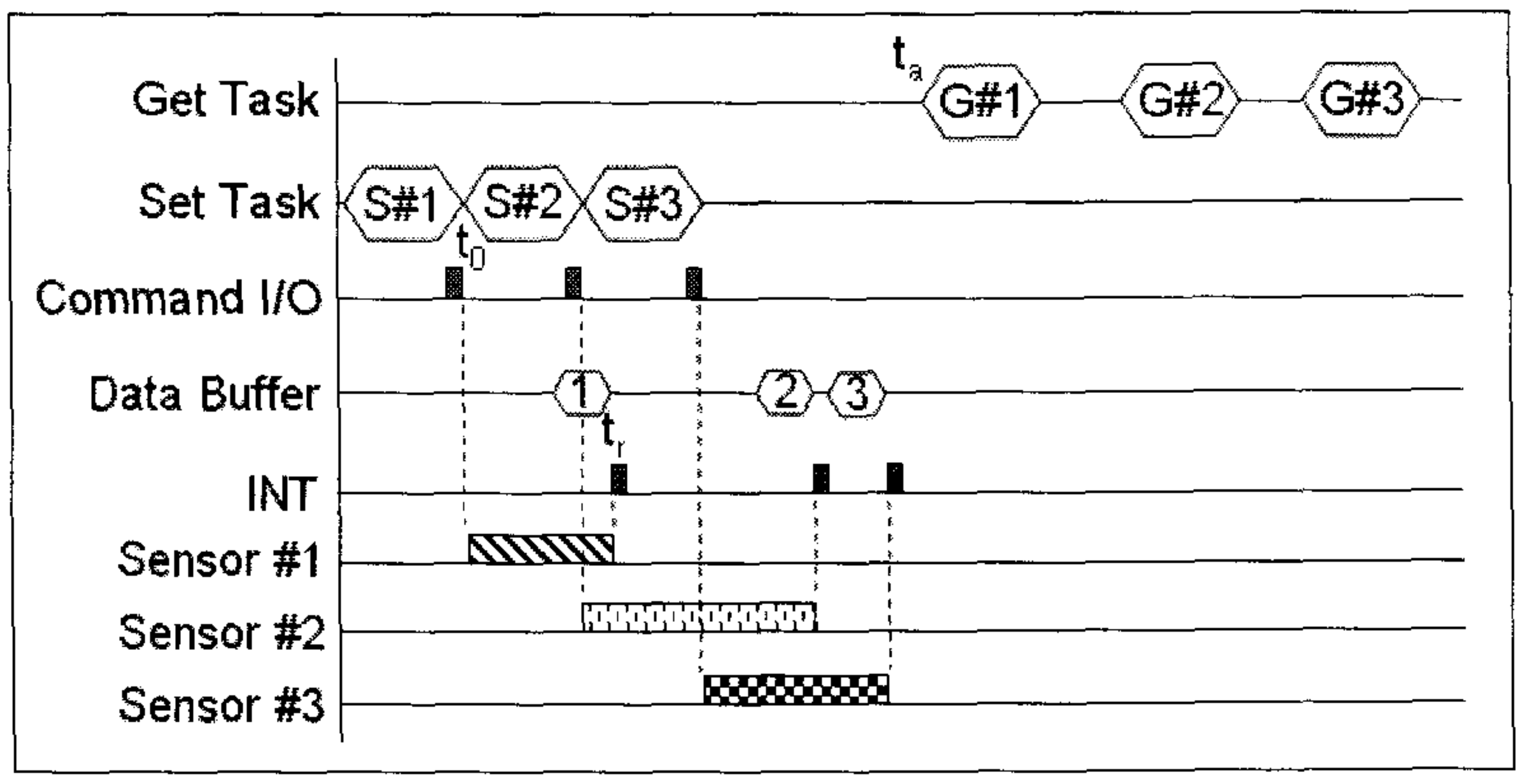

Figure 5. Task scheduling timing diagram

The tasks which handle data creation and management is higher priority task. So it can pre-empt the lower priority task during the lower priority task processes data. It should be ensured that the task which handles data creation and management does not handle the same data with the data which is handled by the pre-empted task.

Task (3), (4), and (5) should be performed by sequential manner because the sequence must be maintained.

3.2 Input data processing

Attitude control task of satellite flight software receives data from Sun sensor, Earth sensor, Gyro and etc devices. It is not important how the data reach to the software through various way. Real thing is that data should be sighted by the software as a local variable. It reflects the need of some special task to manage the acquired data. This special task should perform the buffering of these acquired data before the software need the data. This task needs specific function to interface and map the data to consumer task. As shown in Figure 4, various function will be needed to transport the sensored data from sensor to task. The complex design of memory for data buffering is crucial for each input/output data handling.

3.3 Output data processing

Attitude control task of satellite flight software performs specific algorithms to control the satellite attitude via wheel or thrusters after receiving the raw data which is needed to attitude control from sensor equipment. Finally the task creates command data and the data will be sent via MIL-STD-1553B or bi-level command data line.

If various data can be distributed to each customer, these data should be buffered because it is not efficient to perform the transaction every time data is arrived. If the number of N is the value to perform the transaction efficiently on a system, data should be stored until N_{th} data is arrived. Therefore a concept which processes the data as block unit level as described in Figure 6 is required and it will considerably contribute to raise the performance of data acquisition logic.

In Figure 5, an example is shown as detailed timing diagram during block unit data acquisition.

Data can be read from the buffer(t_a) by data process task after receiving interrupt signal(t_r) which notifies the save finish of sensor data to data buffer form sensor #1.

So, "Get Task" can be invoked after or before $\triangle$t of "Set Task". "Get Task" is just waiting the signal which notifies the event that sensor save the data to buffer successfully (C.H.Koo, 2005).

Therefore, "Get Task" is independent with "Set Task" with regard to time behavior and the focus of function. Actually, "Get Task" can neglect the existence of "Set Task" during the operation. The more the interrelationship between "Get Task" and "Set Task" is decreased, the more the operability and maintainability of data acquisition become simple.

Through this concept, the ambiguity between the time of data ready of sensors and the time of data acquisition is removed. And time dependency between "Set Task" and "Get Task" is lowered than the other mechanism.

Actually, to implement this concept, a specific interrupt controller is needed to process the interrupt signals from multi sensors. So, special controller is mandatory for nice processing of multiple data acquisition and distribution and shall support main micro-controller because this kind of activity can be significant burden in the general purpose micro-controller even if it is present state of the art processor.

4. CONCLUSION

Data predictability and integrity will be ensured by scheduling which avoids the collision between data acquisition task and data usage task. But, hardware I/O channel design is more delicate and complex because in previous I/O management will be ok by using periodic I/O operation, but now process after request is essential. So internal scheduling for the next I/O operation in the device is indispensable for the purpose of this approach to ensure the predictability.

In the real time systems and priority based systems, various race conditions are possible and all these conditions can not be removed dynamically. So, static and deterministic task allocation is essential for ensuring the predictability of data during I/O data exchange.

References from Journals:
C. H. Koo, 2005. Method of data processing through polling and interrupt driven I/O on device data. KSAS, Vol. 33, No. 9, 2005

A. Burns, A. J. Wellings, 1995, SOFTWARE-PRACTICE AND EXPERIENCE, VOL. 25(7), 705-726

K. Tindell, A. Burns, and A. J. Wellings, 1995, Real-Time Systems, Vol. 9, pp. 147-171

Lui Sha, Shirish S. Sathaye, 1995, Technical Report(CMU/SEI-95-TR-011), SEI, p.p 15-19

Acknowledgements
This paper is part of COMS project supported by MOST.