Optimization of Execution Time in Onboard Software for Human-Rated Missions

Harshavardhan Reddy Y	Ranjani K	Manju CR	Jayalekshmy L
Flight Software Integration Division	*Flight Software Integration Division*	*Flight Software Integration Division*	*Flight Computers Group*
AVN/VSSC	*AVN/VSSC*	*AVN/VSSC*	*AVN/VSSC*
yennamharshavardhan@gmail.com	ranjani141@gmail.com	cr_manju@vssc.gov.in	l_jayalekshmy@vssc.gov.in

doi: https://doi.org/10.37285/bsp.SACAD2025.32

***Abstract*— In human space missions, the safety of the human passengers is to be ensured in all situations. To enhance the reliability of the system, four redundant systems are employed in the avionics. Functions such as navigation, guidance, digital auto pilot, fuel tank vent algorithm and sequencing are executed as application tasks in the onboard computer (OBC) of the launch vehicle. For man-rated rockets, certain additional tasks such as data voting, health monitoring of the overall vehicle and interface with the escape system for crew are to be carried out. All these tasks should be completed within the typical cycle time of 20ms, with a margin of at least 2ms per cycle. As this timing requirement was not met initially, the software was updated by applying some optimization techniques. These included code optimizations, re-ordering of the tasks and better memory management. A reduction of 3ms was achieved in the execution time with these steps, ensuring optimum utilization of memory and time for the OBC software in the man-rated mission.**

***Keywords*—Man-rated space mission, Launch vehicle, Onboard computer, Execution time, Code optimization.**

I. INTRODUCTION

The number of launches for transporting humans to space is steadily on the rise. Government space agencies and private enterprises are conducting manned missions, as part of scientific studies, as well as space tourism. For such missions, the primary objective is to ensure the safety of the human passengers at all costs. This calls for stringent safety checks and validations to be built-in throughout the development cycle, for each subsystem in the rockets and crew capsules.

To enhance the reliability of the system, multiple redundant systems are employed in the avionics or electronics system in the launch vehicles. For instance, a quadruple chain configuration is adopted in many rockets for manned missions, as against a single or dual chain system in other missions. In case of a failure in the main system, one of the redundant systems takes over, thus increasing the fault tolerance capability. In the Indian scenario, a quadruple chain avionics configuration is proposed for the human rated mission, with four independent communication buses. It consists of different sub-systems in quadruple redundancy, including the onboard computer, sensors, navigation systems, control systems and health monitoring system (HMS). Typically, functions such as navigation, guidance, digital auto pilot, fuel tank vent algorithm and sequencing are executed as application tasks in the onboard computer (OBC). These tasks are to be executed in different periodicities, based on the system requirements. The tasks such as vehicle attitude update, attitude control and sequencing are to be executed at a faster rate, whereas guidance task is to be executed at a slower rate. In addition, I/O management, voting of data, consolidation of health for HMS and fault handling are carried out by the OS-like component, referred to as the real-time executive (REX).

The real time task scheduler, part of the REX software, determines the exact moment at which a particular task is to be executed in the requisite periodicities. The shorter time period, of the order of 20ms is referred to as a minor cycle and the longer period of 500ms is called a major cycle. An indigenous 16-bit microprocessor is used as the OBC, with a custom architecture and instruction set. To ensure stable control of the vehicle, the control outputs should be available from the OBC to the actuation systems every 20ms. Therefore, all the tasks, executed sequentially, should be completed within this time period, with a margin of at least 2ms per cycle.

In the initial version of the software for the manned mission, this execution time requirement could not be satisfied. Subsequently, some optimization techniques were applied to minimize the execution time by modifying the software. The techniques include optimizations in code, re-ordering of the tasks to minimize idling time and memory management methods. This resulted in considerable reduction in the

execution time of the software, providing sufficient margin in the minor cycle.

This paper explains the strategies adopted to optimize the running time of the various tasks in the onboard computer of a human space mission, thus mitigating the risk of a task over-run in flight. The details of the optimization techniques, the results of testing and the performance of the final software are presented.

The remaining sections in this paper are organised as follows: section II gives the necessary background, including the hardware and software organization of the onboard computer. The details of the changes implemented in the OBC software for optimization of execution time are explained in section III. In section IV, the testing and results are described and the paper concludes in section V.

II. BACKGROUND

A. Onboard computer (OBC)

The Onboard Computer (OBC) serves as the brain of a launch vehicle in the distributed avionics configuration. It carries out complex computations through execution of application software, meets the communication requirements of the avionics systems spread across the length of the launch vehicle, manages redundancies and handles errors that may occur during the launch. The design of the software for the OBC is thus extremely important for the successful completion of a mission.

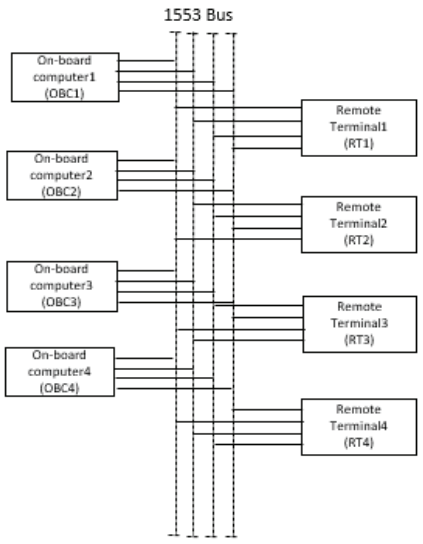

Fig. 1. Quadruple chain 1553B configuration

The OBC communicates with other units of the avionics system over a reliable communication bus, typically the MIL-STD-1553 bus[1]. Input acquisition units (like navigation sensors) and output distribution units for posting sequencing and control commands are handled by different subsystems located in each stage of the vehicle. All these packages are configured as Remote Terminals (RT) on the MIL-STD-1553B bus and OBC is the Bus Controller (BC) for these RTs. A typical quadruple redundant configuration, with 4 buses, and the OBCs and RTs connected to the buses is shown in Fig.1. Here, OBC1 acts as the bus controller or master in bus1, OBC2 in bus2 and so on.

B. Onboard software

The Onboard Computer (OBC) is based on an indigenously developed 16-bit microprocessor, with a custom instruction set. The embedded software in the OBC, particularly the application software, is developed in Ada language. Tasks such as self-check, clock synchronization, telemetry and interfaces to application tasks which are the functions of REX software are also coded in Ada. On the other hand, the portions of the REX software which involve interacting with the hardware and the boot loader, are programmed in the assembly language of the16-bit processor.

The embedded software in the launch vehicle OBC consists of different components. These components, along with their main functions, inputs and outputs are shown in Table I. The navigation task acquires inputs from sensors, processes the data and passes the outputs to other software. Guidance task uses the navigation outputs to decide the optimum trajectory for the vehicle and generates the quaternion command angles to achieve the target in each execution cycle. The control or autopilot task uses these command angles to generate error commands to the actuators to achieve the desired attitude orientation. The sequencing task does the real time event detection and generates the critical commands required for stage ignition, satellite separation and others. The fuel tank vent algorithm maintains the pressure of the fuel tank within defined bounds.

In addition to the application tasks described above, there are some system tasks to be performed by REX. These include self test of the processor, clock synchronization of OBCs, monitoring and transmitting data to ground through telemetry, filling of data in output buffers etc.

TABLE I. OBC SOFTWARE COMPONENTS

Software	Inputs	Outputs
Navigation	Angle and velocity increments, analog rates and acceleration	Quaternions ,body rates, velocity ,position and orbital parameters
Guidance	Altitude, position, velocity and stage information flags	Commanded quaternions
Autopilot	Attitude, body rates, Commanded quaternions	Control Commands
Sequencing	Vehicle acceleration, flight time and fuel tank valve status	Sequencing Commands
Fuel tank vent algorithm	Commands to start the algorithm and fuel tank pressures	Commands for close/open of valves

For scheduling the various tasks in the OBC software, the rate monotonic scheduling (RMS) algorithm is followed. It is a priority-driven pre-emptive scheduling algorithm which assigns static priorities to tasks based on their periods. Higher priority is given for tasks with shorter periods. Thus in the OBC software, minor cycle tasks have higher priority than the major cycle tasks. All minor cycle tasks are executed sequentially one after the other and will not be pre-empted. The overall flow of execution of minor cycle tasks is shown in Fig.2. All major cycle tasks are also executed sequentially one after another. But whenever minor cycle tasks are ready to execute, the major cycle task in execution will be pre-empted and control will be transferred to the minor cycle tasks. The timing diagram for such a rate monotonic scheduling scheme is shown in Fig.3.

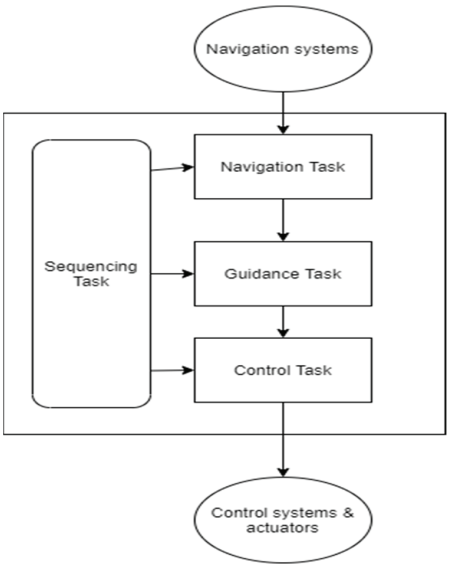

Fig. 2 Execution flow of minor cycle tasks

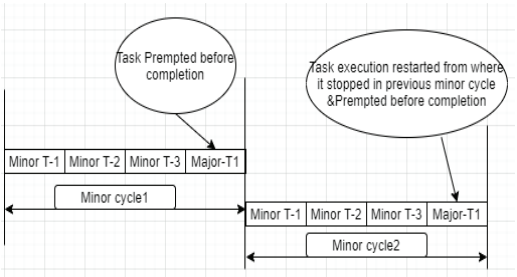

Fig. 3. RMS Timeline diagram

C. New requirements in manned mission

Compared to a regular launch vehicle mission, the onboard software has to carry out some additional functions in the manned mission. This is due to the quadruple redundancy and safety requirements associated with a manned mission. Also, in case of grave failures, the mission should be aborted to ensure the safety of the crew. These extra requirements called for new software modules which contributed to the increased time of execution. The details of these additional modules are presented in this section.

The first new requirement is the introduction of voting algorithms in software. Suitable data selection logic is required for voting of inputs from the redundant systems before usage in algorithms to ensure that at least 2 outputs match. This can bring out errors undetected by self-check of each individual system. Voting on inputs is implemented in the OBC software for selecting the functional data, such as navigation parameters and pressure values. Inability to vote is indicated to the health management system (HMS) for abort decision making.

The second addition is the interface to the HMS for monitoring the health of the overall vehicle. Synthesized health of each element in the avionics system is communicated periodically to the HMS. This is formed based on communication status, system health and other errors detected by the fault-detection logic in different software components. After mission abort, to the extent possible, essential commands are to be issued using the remaining healthy elements to ensure the safety of the vehicle. The overall logic in HMS is shown in Fig.4.

Fault handling for all the new redundant systems by validating the communication and health status is the third requirement. The number of subsystems on the 1553B bus in man-rated missions is around 30, which is the maximum possible, compared to 15-16 in normal missions.

Yet another essential addition in the software was the interface with the escape system for crew (ESC). The ESC is used for taking the crew to safety in case of any emergency situation during the atmospheric phase of the launch. Communicating with ESC to initiate abort during a contingency is vital for safe evacuation of the humans onboard. The messages from ESC are validated periodically for proper communication status, besides checking the health of the system.

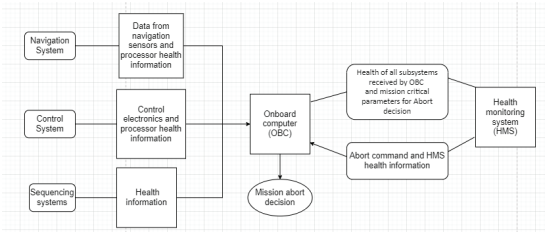

Fig. 4. Flow chart for abort processing through HMS

III. Techniques For Reduction of Execution Time

A. Code optimization

To minimize the execution time and increase the margin in the minor cycle, the OBC software was modified by applying few optimization techniques. The details of these changes are explained in this section, with some illustrative examples.

The first method involved manual translation of computation-intensive routines from high level language to assembly language. Programming directly in assembly language provides more optimized code with fewer lines, compared to the assembly language program generated from high level Ada code by a compiler. An example code snippet in Ada, with its corresponding assembly module written manually is shown in Fig. 5. The Ada module uses a loop to sort three input data and then returns the middle value of the array as the median. The pseudo code in assembly carries out the same function and returns the median value at an address location, in a shorter time.

```
-- Arrange in descending order and find the
median
          i :=1;
          while (i<4)
          loop
            j:=i+1;
              while j<4
              loop
                if (Data(i)<Data(j)) then
                  temp := Data(i);
                  Data(i) := Data(j);
                  Data(j) :=temp;
                end if;
                j:=j+1;
              end loop;
            i := i+1;
          end loop;

          Median := Data(2);
```

Assembly code after manual translation:

```
LOAD R1, X
LOAD R2, Y
LOAD R3, Z
if R2 >= R1
    if R3>= R2
     STORE [MEDIAN],R2;
    else
     if R3>= R1
       STORE [MEDIAN],R3;
     else
       STORE [MEDIAN],R1;
     end if;
    end if;
else
 if R3>= R2
   if R3>= R1
       STORE [MEDIAN],R1;
     else
       STORE [MEDIAN],R3;
     end if;
   end if;
return [MEDIAN]
```

Fig. 5. Ada module and assembly code

In the second approach, some of the onboard software algorithms were optimized by replacing recursive functions with iterative counterparts, reducing stack usage and improving predictability.

A third optimization was attempted in one of the tasks to reduce the overhead caused by linear search of large arrays during execution. In this module, during each stage of the flight, the array entry corresponding to the current altitude was being derived by searching the arrays from the start. Each of these arrays consists of ~80 elements and if the last element is to be selected in a particular cycle, the execution time was correspondingly more. Hence, this was modified by invoking a module for computing the array index as per the current altitude in a major cycle task.

The fourth strategy adopted was the rearrangement of the various options under switch-case statements. This was done in such a way that the condition which is most often and most likely to be satisfied is placed first. This reduced the overhead of checking each condition till the end and executing the corresponding statements during most of the flight duration. This is illustrated with an example in Fig. 6. In this example, variable i has a value of 1 for most of the launch time. Therefore the first statement would be executed immediately, avoiding the overhead of checking the less likely second condition.

```
Case i is
    when 0 => x := sin(y);
    when 1 => x := 1.5;
    when others => null;
end;
```

After modification:

```
Case i is
    when 1 => x := 1.5;
    when 0 => x := sin(y);
    when others => null;
end;
```

Fig. 6. Rearrangement of switch-case options

In the fifth technique, a few changes were done in the sequencing commands software. The processing to decode the exact command to be posted to the hardware elements was removed from the software. Instead, this was processed and decoded offline from the ground and the commands in the final format were uploaded as data in RAM. In addition, all the monitoring data associated with the sequence software, required for analysis at ground is now consolidated and transferred for telemetry as one packet. Earlier, depending on the number of commands, this was repeated multiple times in each cycle.

B. Task re-ordering

In the onboard computer for the launch vehicle, it is highly essential to ensure the proper order of precedence of the tasks and the availability of necessary inputs for each task while fixing the scheduling. For example, the navigation task can be

scheduled only after inputs from navigation sensors are received. The guidance task then uses the outputs from navigation task for deciding the optimum trajectory of the vehicle. The sequencing task gives information about the ongoing stage of the launch vehicle to all the tasks so that guidance and control algorithms are appropriately selected.

It is the responsibility of the scheduler to ensure that idle waiting time of the processor is minimized. Thus, in the software for the man-rated mission, system tasks such as clock synchronization, telemetry and self test of the processor are scheduled at the start. Therefore, rather than waiting for reception of navigation inputs at around 2 ms in a minor cycle of 20 ms, this time is utilized effectively. Subsequently, tasks such as navigation, guidance and autopilot are executed one after another.

The control commands to be posted to the actuators are transferred to the control electronics system through a 1553B message at a fixed time in every execution cycle. The autopilot task which computes these commands should be completed before the time at which this message is transmitted. Other tasks such as initializing and filling of message buffers for the next cycle, fuel tank vent algorithm, HMS interfacing and the processing for escape system for crew (ESC) are scheduled only after the execution of the autopilot task. With the above re-ordering of tasks, it was checked and confirmed that the autopilot computations are finished well in advance of the message transmission time. This revised ordering of the tasks is listed in Table II.

The data required for functional operations such as those from navigation sensors and pressure sensors are acquired by the various remote terminals located along the length of the rocket. These are then transferred via 1553B messages to the OBC acting as the bus controller for further computations. The OBC software validates each message to ensure error-free communication by checking 1553B parameters such as block status word and status word. In addition, to ensure integrity of the input data, change in a running counter and checksum are also validated.

TABLE II. TASK ORDERING

Task order - revised
Self test
Telemetry
Clock synchronization
Input data processing
Navigation
Sequencing
Guidance
Autopilot
Posting of control commands
HMS interfacing
ESC interfacing
Fuel tank vent algorithm

In the man-rated mission, it was mandated that all the functional information should be packed in a single message. This avoids unnecessary processing time in the bus controller as validating and processing multiple short messages takes up more time compared to checking a single, albeit larger message. Some of the functional inputs would be available only with a delay in this case, but it was ensured that this staggering of one execution cycle has no impact.

In addition, all mission requirements which are non-time critical were executed in the longer periodicity of major cycle. Together, the techniques adopted w.r.t re-ordering of the tasks in the OBC resulted in a reduction in the execution time of 3ms.

C. Memory management

Certain principles of memory management were also applied to optimize the time of software execution. Static memory allocation was implemented to avoid dynamic memory allocation during runtime, preventing fragmentation and ensuring deterministic behavior. Data structures were optimized for size, and unused code paths were eliminated to conserve memory. Also, wherever possible, global variables were substituted with local variables. This is because accessing global variables requires more time-consuming address calculation whereas local variables are available in the stack. The global variables in the OBC processor are stored in data RAM and the address calculation involves accessing the file in which they are declared.

IV. TESTING AND RESULTS

As described in the previous section, various changes were done in the OBC software components to reduce the total time of execution. These included optimizations under different categories such as code changes, task re-ordering and better memory management. The updated application software modules were integrated with the REX software and the generated binaries were programmed in the OBC hardware. Simulation runs were executed with this updated software and the execution times of all tasks were monitored through telemetry.

TABLE III. EXECUTION TIMES OF TASKS

Task number	Execution time in ms (original)	Execution time in ms (revised)
1	2.56	2.4
2	2.5	1.65
3	3.4	2.93
4	3.9	2.57
5	0.8	0.7
6	4.6	4.05
7	0.5	0.5
8	0.4	0.4
9	2.0	1.47

The execution times of the various tasks are tabulated for comparison before and after the optimizations in Table III. As seen from the table, significant improvement was observed in the time of execution of most tasks, varying from 0.1 ms to 1.4

ms. The overall time also reduced by more than 3 ms, thus providing sufficient margin in each execution cycle of 20 ms. These results thus illustrated the effectiveness of the optimizations implemented in software. Similar techniques can be adopted by software designers of other, similar systems where strict timing deadlines have to be satisfied.

V. Related Work

Some of the related work from literature which propose various techniques to optimize execution time of tasks in microprocessors is examined here.

The work in [2] proposes the "xfor" programming structure to control data locality, reducing stalled processor cycles and improving execution time and energy efficiency in processor-based systems. It demonstrates a software approach to mitigate pipeline hazards impacting performance. In [3], compiler-based optimization techniques for generating efficient machine code tailored to embedded processor architectures is explored. System software optimizations for heterogeneous processor-based systems, focusing on reducing execution times through efficient resource management and scheduling are proposed in [4].

In [5], the authors present GameTime, a toolkit combining game-theoretic learning and SMT solvers to estimate worst-case execution time (WCET) and optimize software performance in real-time embedded systems. [6] presents an enhanced round-robin scheduling algorithm to optimize task execution times in processor-based systems, particularly for distributed and real-time environments. The work in [7] introduces static worst-case execution time (WCET) analysis optimized for multiple issue architectures, crucial for real-time systems.

Since the OBC in the rocket is based on an indigenous processor with a single core and no pipelining, many of the existing methods for optimization would not be helpful. Therefore, in this paper, we have conceptualized and implemented some standard as well as some novel methods to optimize the onboard flight software and reduce the time of execution.

VI. Conclusion

As the safety of human crew is of paramount importance, the avionics systems in human-rated rockets typically employ quadruple or triple redundancy. To cater to the enhanced functional and safety requirements, extra software modules are developed for such missions, compared to other launch vehicle missions. This, in turn, demands increased resources for computation, including time and memory. Therefore the software should be optimized to minimize these resources.

In this work, the application of a few classes of optimization, to make the code more efficient, primarily in terms of execution time, is presented, in the context of the onboard computer. This includes use of assembly code instead of Ada code for some modules, scheduling the execution of non-essential functions in the longer periodicity, ordering of tasks to ensure complete utilization of the processor, packing of all necessary data in a single frame and other techniques. This enabled all the tasks to be completed faster, providing a gain of ~3 ms in time. For real-time, safety critical systems with short execution cycles, this reduction is highly significant.

In future, more optimizations are proposed to be attempted, to further reduce the requirements of time and memory. Also, the use of assistive coding is planned to be explored for exploiting the capabilities of AI in software programming.

Acknowledgment

The authors wish to express their sincere thanks to their colleagues Sri. Akash Niladri Ganguly, Syam Kumar S and Syamlal LS and Deputy Director, Sri. Biju SR for the help and advice rendered in successfully completing this work. We would also like to thank Sri.Gadhave Amol Nanasaheb, for reviewing the paper and giving suggestions for improvement.

References

[1] MIL-STD-1553 Designer’s Guide Sixth Edition, 1998, Data Device Corporation (DDC).

[2] Clauss, P., Fassi, I., &Jimborean, A. (2014). "Software-controlled Processor Stalls for Time and Energy Efficient Data Locality Optimization." International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS XIV).

[3] Stirling, S., Rocha, R. C. O., Hazelwood, K., Leather, H., O'Boyle, M. F. P., &Petoumenos, P. (2022). "Optimizing Code Generation for Embedded Systems." 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO).

[4] Lefeuvre, H., Bădoiu, V.-A., Jung, A., Teodorescu, Ș., Rauch, S., Huici, F., Raiciu, C., & Olivier, P. (2022). "Optimizing System Software for Heterogeneous Architectures." ASPLOS 2022 - Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems.

[5] Seshia, S. A., &Rakhlin, A. (2011). "GameTime: A Toolkit for Execution Time Analysis of Software." Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11/ETAPS'11).

[6] Fiad, A., Maaza, Z. M., &Bendoukha, H. (2020). "Improved Version of Round Robin Scheduling Algorithm Based on Analytic Model." International Journal of Networked and Distributed Computing.

[7] S. S. Lim, J. H. Han, J. Kim, S. L. Min (1998), "A Worst-Case Timing Analysis Technique for Multiple-Issue Machines", 19th IEEE Real-Time Systems Symposium (RTSS 1998)