ANALYSIS OF VARIOUS APPROACHES FOR C&DH FOR STUDSAT-2

Bhuvanesh S K
Dept. of Electronics and Communication
NMAMIT, Nitte
Karkala, Karnataka, India
bhuvanskk11@gmail.com

Pooja Pai
Dept. of Electronics and Communication
NMAMIT, Nitte
Karkala, Karnataka, India
paipooja99.pp@gmail.com

Shraddha S
Dept. of Electronics and Communication
NMAMIT, Nitte
Karkala, Karnataka, India
shraddha091998@gmail.com

Durga Prasad
Associate Professor
Dept. of Electronics and Communication
NMAMIT, Nitte
Karkala, Karnataka, India
durgaprasad@nitte.edu.in

***Abstract*—Project STUDSAT-2 is India's Twin Nano Satellite Mission. It hosts four payloads namely Automatic Identification System(AIS) for ship Identification and Tracking, InterSatellite Link (ISL) to prove in orbit Communication between satellites, Drag Sail Mechanism to de-orbit the satellite at the end of mission and a Beacon data relay for HAMs. The satellite system of STUDSAT-2 consists of six subsystems i.e.Payload, Attitude Determination and Control System(ADCS), Electrical Power System(EPS), Command and Data Handling (C & DH), Mechanical Structure and Communication sub-system are the brain of the satellite which is responsible for on-board control and processing. The software deployed on top of FreeRTOS operating system should handle multiple tasks based on their priority levels. The project aims at designing such software on top of real-time Operating system. Multiple tasks which are supposed to be demonstrated are Timer call back function with ten threads for six current sensors and four temperature sensors. The designing of the software where tasks are scheduled using preemptive priority scheduling algorithm. Thus the project aims at building a software and demonstrating the command and data handling ability of the system using six current Sensors and four temperature sensors.**

***Index Terms*—RTOS, shunt current, bus voltage, temparature, TMP75 temparature sensor, INA219 current sensor, Secure Digital Card(SD card), STM CubeMX, Keil uVisionMDK5**

I. Introduction

STUDSAT-2, a twin nano satellite, aims at proving in-orbit separation, establish inter-satellite communication, implementation of drag sail technology to de-orbit the slave satellite and capturing earth image using CMOS camera. The satellite system of STUDSAT-2 consists of six subsystems i.e. Structure, Attitude Determination and Control System (ADCS), Electrical Power System (EPS), Command and Data Handling (C & DH), Communication and Payload. The ADCS is responsible for maintaining the desired orientation of the satellite in the orbit. The satellite employs a combination of sensors and actuators to achieve the same. EPS is the major subsystem which supplies regulated power for proper functioning of all the subsystems. The solar energy is harnessed by the solar panels and converted to electrical energy which is stored in Li-ion batteries. The objective of Communication and Payload system is to establish communication link with ground station and transfer the payload and telemetry data to the ground station using full-duplex mode of communication. The structure of the satellite is designed considering the dimensions 30 cm x 30 cm x 20 cm. C & DH subsystem is the brain of satellite, i.e. responsible for on board control and processing. All the subsystem operations are mastered by an On-Board Computer (OBC) system. The sensor data and other important information about the outer space environment has to be down-linked to the earth station. C & DH will manoeuvre the transmission of all these information. Real-time computing is where system correctness not only depends on the correctness of the logical result but also on the result delivery time. So the operating system should have features to support this critical requirement to render it to be termed a Real-time Operating System (RTOS).

II. Literature Survey

To deal with numerous processes one can utilize distinctive calculation for booking the assignments. A Process Scheduler plans various procedures to be allocated to the CPU dependent on specific booking calculations. A couple of them are
1. First-Come, First-Served (FCFS) Scheduling
2. Shortest-Job-Next (SJN) Scheduling
3. Priority Scheduling
4. Shortest Remaining Time
5. Round Robin(RR) Scheduling
6. Multiple-Level Queues Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive calculations are structured with the goal that once a procedure enters the running state, it can't be aborted until it finishes its distributed time, while the preemptive planning depends on need where a scheduler may acquire a low need running procedure whenever a high need procedure goes into a ready state.

*1) **First Come First Serve (FCFS)**:*

- Jobs are executed on first come, first serve premise.
- It is a non-preemptive, pre-emptive scheduling algorithm
- Easy to comprehend and actualize.
- Its execution depends on FIFO queue.
- Poor in execution as normal hold up time is high.

*2) **Shortest Job Next (SJN)**:*

- This is otherwise called most brief employment first or shortest job first, or SJF.
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best way to deal with limit holding up time.
- Easy to actualize in Batch frameworks where required CPU time is known ahead of time.
- Impossible to actualize in intelligent frameworks where required CPU time isn't known.
- The processer should know ahead of time how much time process will take.

*3) **Priority Based Scheduling**:*

- Priority scheduling is a non-preemptive algorithm and one of the most widely recognized planning calculations in cluster frameworks.
- Each processes is allocated a need. Process with most noteworthy need is to be executed first, etc.
- Processes with same need are executed on first start things out served premise.
- Priority can be chosen dependent on memory necessities, time prerequisites or some other asset prerequisite.

*4) **Priority Based Scheduling**:*

- Shortest remaining time (SRT) is the preemptive form of the SJN algorithm.
- The processor is allotted to the activity nearest to consummation yet it very well may be acquired by a more up to date prepared occupation with shorter time to finish.
- Impossible to execute in intelligent frameworks where required CPU time isn't known.
- It is regularly utilized in group conditions where short occupations need to give inclination.

*5) **Robin Scheduling**:*

- Round Robin is the preemptive process scheduling algorithm.
- Each process is given a fixed time to execute, it is known as a quantum.
- Once a process is executed for a given time-frame, it is terminated and different procedure executes for a given time-span.
- Context exchanging is utilized to spare conditions of acquired process.

*6) **Multiple-Level Queues Scheduling**:*

- Multiple-level lines are not a free planning algorithm. They utilize other existing algorithms to gathering and schedule tasks with normal qualities.
- Multiple queues are kept up for procedures with normal attributes.
- Different types of scheduling algorithms can be implemented to different queues.
- Each queues are allocated based on predefined priorities.

The background work discusses the need for RTOS for the project also aims at a case study on RTOS selection, software development for Telecommand and Telemetry handling for STUDSAT - 2. A good RTOS should have these properties: Multi-tasking and preemptable, should be able to identify the deadline within which the task should be executed, should be capable of predictable synchronization, should have sufficient priority levels and should possess predefined latencies. A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. Following are the scheduler plans that are studied: First-Come, First- Served (FCFS) Scheduling, Shortest Job-Next (SJN) Scheduling, Priority Scheduling, Shortest Remaining Time, Round Robin(RR) Scheduling, Multiple- Level Queues Scheduling. Case study of RTOS selection is done through referring different papers on satellites and its RTOS and comparative analysis is done.

- **Customized RTOS**: Some have used a customized RTOS is used where it is felt that the need for a heavy RTOS is unnecessary. However, basic features are retained and additional features are added such as RTOS used in skCube [1] Satellite uses only one static pointer and also provides emulation of lockstep with double scheduling whereas DANDE [2] has preemptive scheduling driven by watchDog reset and also has an interrupt to be raised in case of overvoltage in the hardware parts.
- **XilKernel**: XilKernel RTOS implemented on Spartan-6 FPGA [3]. The disadvantages are FPGA sensitivity to ionizing radiation is not fully known and research and testing techniques for tolerance to error are yet to be determined.
- **CooOx CoOs**: CooOx CoOs used in IiNUSAT 1 [8] and UGMSat 1 [7] is an embedded RTOS which is free and open-source but being specially designed for Cortex M4 makes it difficult to interface with other devices and also the platforms supported by the OS are ICCARM, ARM GCC, and GCC. Otherwise, it is fairly good RTOS to be used for nano-satellites.
- **Micrium III RTOS**: One major advantage of Micrium III is that it is highly scalable [5]. This RTOS occupies very little memory and also does not impose restrictions on the declaration on the number of Kernel objects as long as it does not go out of memory range of the RTOS. Considering this feature, to avoid priority inversion it also has 256 priority levels which make the preemptive scheduling more effective and efficient.
- **FreeRTOS**: FreeRTOS is a free and open-source RTOS which implies that it can be tweaked as per the user's needs. FreeRTOS also supports a wide range of platforms also which makes it more versatile and desirable [6]. Its

design has been developed to fit on very small embedded systems and implements only a very minimalist set of functions, a very basic handle of tasks and memory management, just sufficient Application Programming Interface (API) concerning synchronization, and absolutely nothing is provided for network communication, drivers for the external hardware, or access to a files system. Since most of the code is written in the c programming language, it is highly portable and has been ported to many different platforms. Its strength is its small size, making it possible to run where most (or all) other real-time operating systems won't fit [4]. FreeRTOS is designed to be simple, portable and concise. FreeRTOS has much smaller requirements when it comes to the amount of memory. FreeRTOS has a simple highest priority first scheduler. It has much less scheduling overhead.

III. Methodology and Implemenation

A. Block Diagram

The project is built around the STM32F407VG Discovery board. The six INA219 current sensors and four temperature sensors are interfaced to the board. The shunt current values and bus voltage values read by INA219 current sensors and the temperature values read by TMP75 temperature sensors are sent to STM32F407VG Discovery board.This communication follows Inter-IC (I2C) protocol. The values so received by the board is sent to the computer via UART bridge where it is displayed on serial monitor display. This helps for debugging purposes. The data read by the board is also stored in the SD card to achieve telemetry logging of the obtained sensor values. The block diagram of the project is as shown in Figure 1.

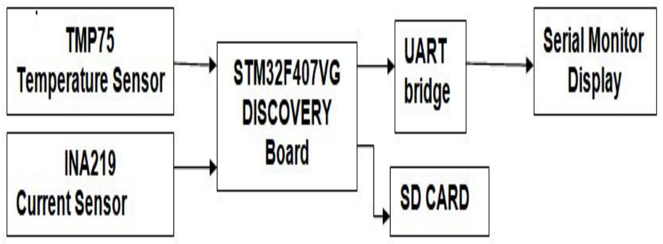

Fig. 1. Block Diagram

B. Flow chart

The first task is to use an inbuilt Real-time clock to generate a delay of 45 minutes. This is the duration where the satellite switches to normal mode, post ejection. The second task is to interface the four temperature sensor with microcontroller to collect temperature data. After confirming the attitude stability, an event-based separation of satellites is initiated.The third task is to interface six current sensors in the I2C protocol format to communicate with the microcontroller as master and collect the bus voltages and shunt current through the shunt resistor. The flow chart of the project is as depicted in Figure 2.

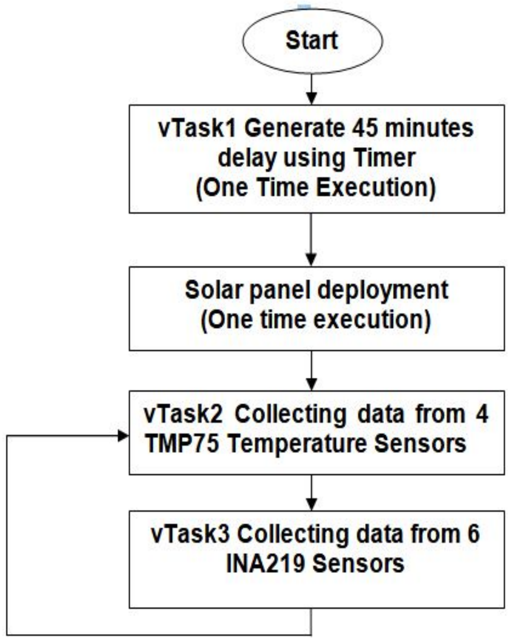
Start
vTask1 Generate 45 minutes delay using Timer (One Time Execution)
Solar panel deployment (One time execution)
vTask2 Collecting data from 4 TMP75 Temperature Sensors
vTask3 Collecting data from 6 INA219 Sensors

Fig. 2. Flow Chart

C. Procedure for implementation

To interface the four temperature sensors and six current sensors the pinouts, communication interfaces like serial peripheral interface (SPI), I2C, Universal asynchronous receiver-transmitter (UART), clock configuration is initialised using STMCubeMX. Along with this, middlewares like FATFS file systems, FreeRTOS, timers, semaphores are declared. This generates a code with input-output pin declarations and empty threads declarations. Library files for current sensor and temperature sensor which constitutes of necessary function declaration and definitions are written. These library files are included in the main program and tasks are called into the empty threads. This is programmed using Keil uvision5 or STM32 Cube IDE. There are 11 threads: one default thread, six threads for six current sensors and four threads for four temperature sensors. The delay of 45 minutes is achieved using the timer inbuilt in STM32F407VG Discovery board as it takes 45 minutes switch on to the normal mode post ejection. During this duration, this task has priority aboveNormal hence, none of the other tasks run during this time. Post the delay, the four TMP75 temperature sensors read temperature of the surroundings and the six INA219 current sensors read shunt currents and bus voltages. The so read values are displayed on the computer using HTerm software using the UART protocol. The temperature sensor and current sensor data so collected are stored into an SD card and hence telemetry logging of data can be achieved and task analysis can be done using middlewares.

IV. Results and Conclusions

When the satellite is ejected, a 45 minutes delay is measured before the satellite switches to the normal mode and the OBC on the satellite starts executing its tasks. In the present work, performance of various sessions are checked. The delay of 45 minutes is achieved using the timer inbuilt in STM32F407VG Discovery board and during this duration, this task has priority aboveNormal hence, none of the other tasks run during this time. After the delay, the satellite switches to normal mode and thus starts collecting temparature data using the four TMP75 temparature sesors modules and shunt current values using six INA219 current sensor modules. The ability to schedule multiple tasks for six current sensor INA219 modules four TMP75 temparature sensor modules is demonstrated. The log file records to an SD card using FATFS system on Middleware layer and SEGGER systemView used for Task analysis.

References

[1] J. Slaˇcka and M. Halás, "Safety critical rtos for space satellites," in 2015 20th International Conference on Process Control (PC), 2015, pp. 250–254.

[2] C. M. Cooke, "Implementation of a real-time operating system on a small satellite platform," Ph.D. dissertation, Colorado Space Grant Consortium, Boulder, CO, 2012.

[3] A. Hanafi, M. Karim, I. Latachi, T. Rachidi, S. Dahbi, and S. Zouggar, "Fpga-based secondary on-board computer system for low-earth-orbit nanosatellite," in 2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), 2017, pp. 1–6.

[4] B. Rajulu, S. Dasiga, and N. R. Iyer, "Open source rtos implementation for onboard computer (obc) in studsat-2," in 2014 IEEE Aerospace Conference, 2014, pp. 1–13.

[5] C. Nagarajan, K. Kinger, F. Haider, and R. Agarwal, "Performance analysis of micrium rtos in the computer of a nano-satellite," in 2015 IEEE Aerospace Conference, 2015, pp. 1–9.

[6] K. Lamichhane, M. Kiran, T. Kannan, D. Sahay, H. G. Ranjith, and S. Sandya, "Embedded rtos implementation for twin nano-satellite studsat-2," in 2015 IEEE Metrology for Aerospace (MetroAeroSpace), 2015, pp. 541–546.

[7] A. Putra, T. Priyambodo, and N. Mestika, "Real-time operating system implementation on obc/obdh for ugmsat-1 sequence," vol. 1755, 07 2016, p. 170009.

[8] A. Tomar. (2014) Coocox:user guide for coocox coos operating system. [Online]. Available: https://www.element14.com/community/docs/DOC-42435/l/coocoxuser-guide-forcoocox-coos-operating-system