Software Development For Command And Data Handling Subsytem of Nano-Satellite

Pooja K P[1], ArunKumar R[2], Dr.S.Sandya[3]
M.Tech Student[1], Reasearch Scholor[2], Professor & HOD. Dept. of ECE[3]
Nitte Meenakshi Institute of Technology[1]

Abstract—
The brain of a satellite is said to be the On-board computer(OBC),that is designed as the resource of the constrained environments which has to be specifically tailored for the host system and its needs, as the real time functioning system afforms as the interrupts and various time critical tasks are to be processed when it is required. For the maximization of accessibility, the functioning system has to be inexpensive, and availability should be made easy, where it demands to be as an open-source. The real time capabilities,complete accessibilty for the source code, functional improvement environment,cross platform support are used as the requirements that are said as free RTOS. Following through the literature survey, the on board computer which supports the Free RTOS through the ARM microcontroller that has low power and also high capability performance and also the STM32 ARM cortex-M4 having 168MHz CPU and also 210 DMIPS.

The controls and the coordination of the other subsystems in the satellite is done via the On-Board computer, this paper has been implemented the preemptive scheduling algorithm for OBC to monitor and control the power delivered to the loads. One-time task such as antenna deployment, solar panel deployment, satellite separation and RTC to wake up the OBC after 45sec to mimic the satellite mission sequence of 45mins delay after injection in to orbit.

Key words— STUDSAT; RTOS; FreeRTOS; Scheduling; Task management.

I INTRODUCTION

A Real-time operating system (RTOS), an operating system which helps to inscribe good embedded software for the systems with is very complex, particularly to guarantee the function that meet the processing of the execution time of the tasks in the given time limit. In general, real-time applications of embedded systems are a blend of both hard and soft real-time requirements. The state of a time deadline is deterministic said as soft real-time requirement, when breached, the system is not said as ineffective whereas hard real-time requirements are probalistic which means that requirements states a time target, that is when the specific deadline is not met the entire system failure occurs.

In the work proposed the technical explanation is illuminated via FreeRTOS in STUDSAT-2. The presentation is done by the accomplishment of FreeRTOS on STM32Cortex M4 controller to broaden the functional combination in particular the recital of small satellite. Exploitation of FreeRTOS simplifies software development, enables the code modularity, which leads in maintainable and expandable high-performance and also will reduce the efforts essential to build up the software for STUDSAT-2A/2B Mission.

III NEED FOR RTOS

So, addressing on the need for an OS, an OS required just to provide a platform for the software to get executed. It provides an abstraction to the tasks regarding the underlying resource management operations. As the mission objectives increase of become complex, more and more people contribute to the code base of the project and integrating the codes from different engineers become a daunting task if we consider the superloop. A need to add more functionality to the project might lead to complete change of the software which is insufficient way of doing the things. When one considers an OS, the contributions are in the form of tasks, which can be (in the most of the cases) run independently from other tasks. So, adding functionality to the software just means adding the tasks and maybe tweaking the priorities of the each of the tasks. No major reconfiguration is required.

And, addressing on the need for an RTOS, the satellite handles many types of operations like communicating with the ground station, inter-satellite communication, data gathering etc. Some tasks have higher priority than others. For e.g., controlling satellite attitude is more important because if not done in time, the whole mission may fail so, to make sure that the tasks will execute within the deadline. RTOS has the feature of reliability.

III CHOICE OF FREERTOS

The mission objective requires the RTOS kernel to have basic functionalities like task management, semaphores, mutex, priority inheritance capability, event management, queue etc. FreeRTOS is license free that is, FreeRTOS can be accessed freely; the code is available freely and for the user it is an open source. It supports 34 architectures and 18 tool chains till date.

FreeRTOS is regarded as a good choice among all due to the accessibility of the records for various microcontroller architectures, technical support and enormous user base.

The memory requirement for FreeRTOS is provided below, foot prints of memory:

1. 236 bytes of scheduler [can be reduced using data type of smaller bytes].

2. Queue comprises of 76 bytes along with the queue storage.

3.64bytes of tasks [4 character name] along with Task stack size.

4. 10Kb of the Kernel size.

5. 120 Kb of total memory.

So, the footprint is quite small and also has the above mentioned features the use of FreeRTOS makes sense. Additionally, FreeRTOS is royalty-free, which is good for a student satellite mission.

IV ON BOARD COMPUTER (OBC)

OBC sub system which acts as brain of the satellite by not only commanding every action of other sub system, but also by handling the data sent by the sub systems. To carry out this vital task, ARM Cortex-M4 based STM32F407 is used as the OBC. C&DH sub-system acts as the centre of all the data handling and processing.

The controllers present in OBC are 2 and they are as follows:

1) Handling of data and command

2) The control system and the attitude of determination.

The brain of the satellite is the command and data handling system. They follow the following functionalities:

1) STUDSAT OBC subsystem is to develop the working of the hardware prototype
2) Logging of the data that is telemetry in the memory that is periodically used to perform the house keeping functionalities.
3) Faults are to be detected and corrected..
4) Launcher, ground segment along with the two constellation of the space craft as the telemetry which is possibly received along the Inter-Spacecraft links which are to be interfaced along the electrical and ground sustain tools.

V OPERATING SYSTEM DESIGN

The system software programming that will manage programming assets, the computer hardware equipment and furthermore it gives the regular administrations to computer programs are an operating system (OS). For multiprocessing and multitasking, an OS is indispensable. Among many services, an OS provides a scheduler, which decides which process or tasks gets resources like CPU time, I/O etc.

A. operating system architecture

There are 3 logical layers in an OS as shown as fig1. The Boot loader and Board Support Package (BSP) are present in the lowest layer. To control computer hardware peripherals like SPI, I2C and UART the BSP incorporates basic mechanism. RTOS kernel and drivers exists in the middle most layer which can access the BSP layer straight. Upper most layers have user tasks and a sequencer that executes the control commands which are received from base station. The kernel that is RTOS or driver is used as an intermediate between the BSP loader and the upper most layer as the layer cannot access the loader directly.

Fig1 Operating system design layout

B. OS Software architecture

In the proposed OS all tasks share single stack with higher need assignments being heigher in the stack space. All continuous assignments have their own stack.The fundamental task management comprises of creating a errands, making the errands set for implementation and errands yielding. The inter task communication can be made possible by sending data or events. The tasks can vary its states as shown in the figure fig 2.

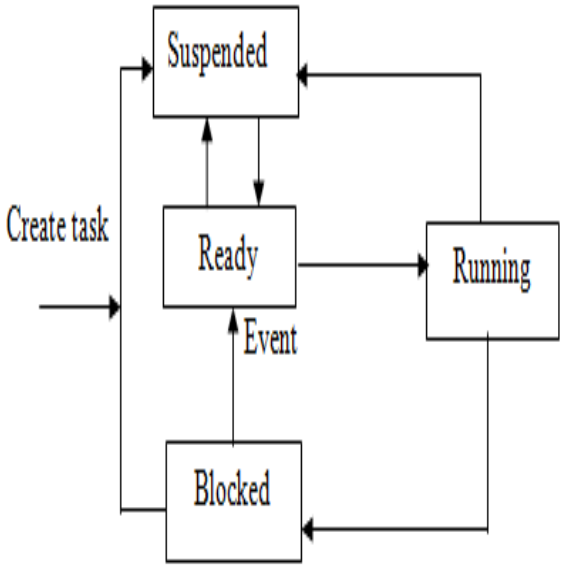

Fig.2 Task states.

Initially the task is in an IDLE state. The current state of the task can be changed to the ready (RDY) state when the OS call is located in the ISR or in the code by the user which keeps the OS in the ready queue. According to the priority the tasks are arranged in the ready queue and the highest priority task from the ready queue will be executed when OS calls the Scheduler. Due to an event the task will be in waiting condition, once the event is satisfied the errands is ready and is placed in the ready queue for the execution.

C. Layer architecture

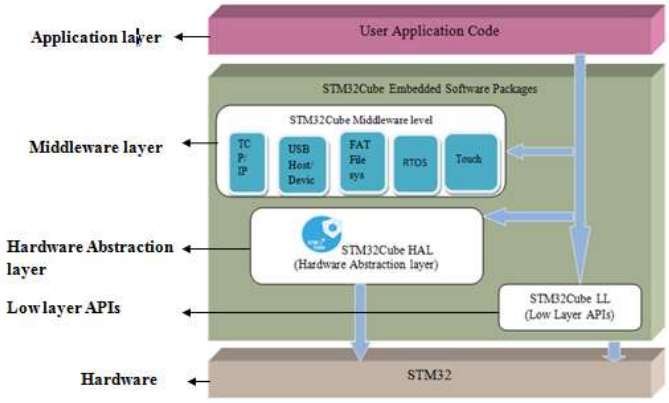

Fig.3 Layer architecture.

The architecture comprises of 4 layers namely application layer, low layer APIs, Hardware Abstraction layer and middle level layer. In hardware layer the STM32F407 Discovery Board has been used. HAL drivers and FreeRTOS have been used in Hardware abstraction layer. The Low layer APIs include STM32CubeMX. User Application code is used in Application layer. Layer architecture can be seen in the above figure

D. Algorithms used

At the point when STUDSAT-1 was created using real-time operating system which is a constant working framework and a variety of algorithms did not strike us. Nonetheless, after its dispatch we realized how usage of a RTOS could have essentially

diminished the complication of the code and spared a considerable measure of time. For the purpose of development Prioritized Preemptive Scheduling algorithm is used.

Prioritized Preemptive Scheduling

The Scheduling algorithm has the following features

• Priority is assigned for each and every errand.
• Each errand can appear in one among many states.
• At one point of time only one task can be in the Running state.
• The task can enter into the running state when the higher priority task is selected by the scheduler and this is termed as 'Fixed Priority Preemptive Scheduling'. As the priority which is assigned will remain constant and cannot be changed by the kernel itself (priorities can be changed only by the tasks) it is regarded as a 'Fixed Priority'. 'Preemptive' when a task with the lowest priority is under execution and a highest priority task enters making the current execution pre-empt and thus making the higher priority task to execute.

VI IMPLEMENTATION OF RTOS AND PRIORITY

The proposed work uses uVision Keil for implementing the RTOS. STM32F4CubeMX: A graphical software configuration tool that generates the initialization code for STM32 peripherals. STM32F4HAL for STM32 peripherals the HAL is a software abstraction layer initially, the tasks required for the operation of satellite has to be tabulated. Table 1 illustrates the list of tasks used. The basic three types of tasks are.
• Periodic task that takes action on a regularly..
• Regularly update task gathers the data and stored by the global storage area and in further it can be used by other tasks.
• Aperiodic tasks which is the result of some external command response like executing from the results of the ground command. Each task is given a priority based on depending upon the time required for the execution and task criticality.

Each errand has a priority from 0 to (configMAX_PRIORITI-1).
Based on the application, the FreeRTOSConfig.h is defined within it which is said as configMAX_PRIORITIES. The more the value assumed to configMAX_PRIORITIES the consequent RAM's the kernel FreeRTOS will be considered. The task with the lowest priority denotes the low priority number. The tskIdle_PRIORITY is defined as the default priority that is being zero. Whether the task is ready else in running state is scheduled by the scheduler that is for the given processor time which has the low priority for the processor time that is preference in the ready state. In a similar way, the highest processing time will be given to tasks with higher priority task. The priority mainly depends on up commanding, how fast the task response to the particular event.

Tasks	priority	Timing
Manager	3	Continuous
Startup delay(45min)	3	One Time
Antenna deployment	3	One Time
Solar panel deployment	3	One Time
Satellite separation	3	One Time
STUDSAT Beacon	2	Periodic
Watchdog	2	Periodic
Payload operation	2	Periodic
Power(monitoring & control)	1	Periodic
Redundant communication	1	Periodic
Inter processor communication	1	Periodic
Telemetry logging	0	Periodic
Firmware upgrade	0	Aperiodic
Uplink communication	-1	Aperiodic
Downlink communication	-1	Aperiodic
Inter-satellite communication	1	Aperiodic

Table 1 List of Tasks in the C &DH

VII INTEGRATION

The system as shown in fig4 consists of ARM Cortex based STM32F407 Discovery board connected to Current sensor, the Temperature sensor and UASRT and enable RTC. Both Current sensor and temperature sensor are interfacing to controller

through I2C communicational protocol. For synchronizing between the tasks priority preemptive scheduling algorithm will be used.

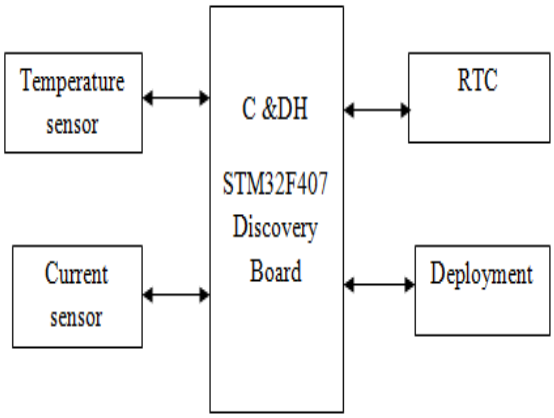

Fig.4 Block diagram

Temperature sensor TMP75 is integrated with STM32F407 via I2C communication protocol. Current sensor INA219 is integrated with STM32F407 via I2C communication protocol. And the RTC is configured with 10sec delay

The following tasks have been indentified for the project.

Task Name	Priority
RTC configration	-1
Development	0
Temperature sensor	1
Current sensor	2

- The work mainly focuses on the priority scheduling algorithm.
- The four tasks have been created using CUBEMX, Task1 for RTC, Task2 for deployment, task3 included current sensor, and task4 includes temperature sensor code.
- All 4 tasks are assigned with priorities.
- The task1 is RTC with the OsPriorityhigh, followed by the task2 is temperature sensor with OsPriorityAboveNormal priority and then task3 is current sensor with OsPriorityNormal then finally last task is task4 with OsPriorityBelowNormal.
- RTC is configured and initialized with 10 sec delay and FreeRTOS is enabled.
- The LED glows after 10sec delay and the task 1 completed.
- The deployment is done using switches concept. Initially the switch will be off state .When the button is pressed corresponding LED's glow.
    - Red LED = antenna deployment
    - Green LED= separation mechanism
    - Blue LED= satellite deployment
    - When the LED glows the task 2 gets completed.
- Task 3 contains telemetry data that is temperature sensor. The sensors have been interfaced with STM32F407 using I2C, which will sense the digital data and displayed in debug viewer in Keil IDE. The Task 4 contain current sensor and it will sense the value. Then both sensor data is transferred through USART with the frequency 11200 and results can be seen in HTerm. Then the tasks as completed.

VIII RTOS FLOW AND BLOCK DIAGRAM

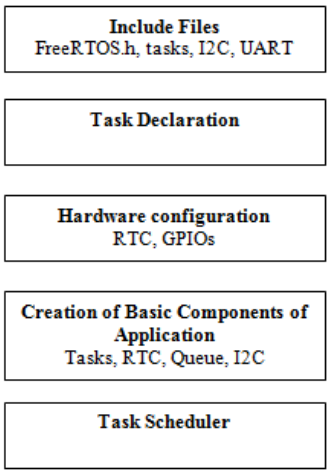

Fig 5 RTOS main structure

IX SOFTWARE TESTING

To exanimate certain errand management features of FreeRTOS, the code has to be written on Keil IDE later it has to be executed on the microcontroller STM32F4DISCOVERY board. There will be around 11Kbyte of total flash footprint which incorporates FreeRTOS code, application and library code. The code below shows few snippets that indicate various FreeRTOS features.

For the creation of a particular task using FreeRTOS the syntax is given below. For the simple task creation by using the FreeRTOS the following code snippet is used:

```
void emptyTasks(void *pvParameters)
{
char *pcTaskName = "Empty Task is
running\n "; //The task which is to be run is displayed
for(;;) //loop that is infinite
{
USART_puts(USART2, pcTaskName);
//USART2 is used for printing on the terminal software
Delay();
}
}
```

By the usage of FreeRTOS the following code snippet shows the priority scheduling. the uxPriority is assigned as the intial priority of the task which is the parameter of the xTaskCreate() function of API.

vTaskPrioritySet() is said as the priority which can be changed using this as an function of API. With the help of configMAX PRIORITIES, the maximum number of priority can be set which is present within the header file FreeRTOSConfig.h. As the config- MAX PRIORITIES value increases the RAM kernel consumes more memory.

```
void vTask1(void *pvParameters)
{
 char *pcTasksName = "The Task1
 running\n";
UportBASE_TYPE vxPriority;
vxPriority = vxTaskPriorityGet();
for(;;)
 {
Delay();
USART1_puts(USART1, pcTasksName);
USART1_puts(USART1, "About to
raise the sampleTask2
priority...\n");
vTaskPrioritySet(xTasks2Handle,
(vxPriority+1));
}
```

```
}
void  vTask2(void *pvParameters)
{
 char *pcTasksName = "sampleTask2
is running\n";
UportBASE_TYPE uxPriority;
uxPriority = vxTaskPriorityGet();
for(;;) {
Delay();
USART1_puts(USART1, pcTasksName);
USART1_puts(USART1, "About to
lower the sampleTask2
priority...\n");
vTaskPrioritySet(0,
(vxPriority-2));
}
```

```
}
int main( void )
{
init_USART2 (11200); // initialize
USART2 at the  baud of 11200
USART_puts(USART2, "USART
Initialization complete !\r\n");
//Task Creation
xTaskCreate(vTask1,
signed char * )
"vTask1", 440, NULL,
vTask1Priority, NULL);
xTaskCreate(vTask2, (
signed char * )
"vTask2", 440, NULL,
vTask2Priority,&xTask2Handle);
/Execute the task
vTaskStartScheduler(); // indicates  the execution of the tasks
```

X RESULTS

After 10sec delay the LED will glow then all the sensor value will be displayed in the debug viewer in the keil complier and that value are transmitted and received using USART communication protocol which is interface with STM32F407 controller via I2C and the values are displayed in HTerm is seen in the figure 7 with temperature value of 23.96 and vBusVolt value of 17.6640v. The figure 6 shows the complete hardware setup. The figure 8 shows the Tracealyzer output. The task scheduling is presented using colour coded rectangles, where the colour helps to identify the thread of execution - a task. The Figure 9 shows the CPU load graph. The CPU Load Graph displays the CPU usage per task and interrupts over time.

CONCLUSION AND FUTURE SCOPE

The software development for command and data handling subsystem version 1 has been implemented and checked for proper outcome. HAL drivers were used instead of standard peripherals. Further for synchronizing the tasks the FreeRTOS objectives like semaphores, queues and also event notification can be used in the higher version of software development for the command and data handling sub system of nano-satellite

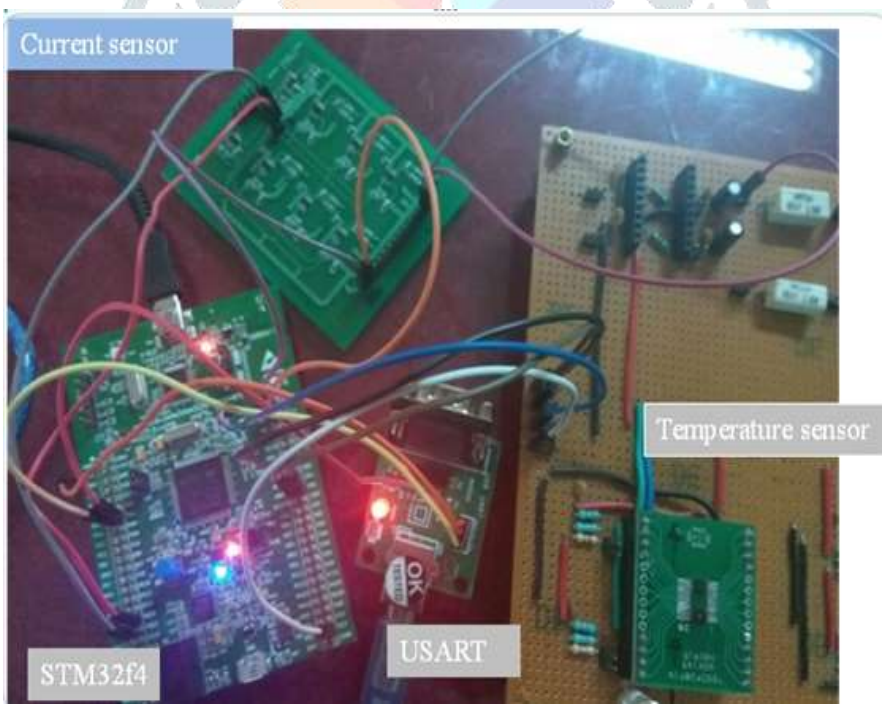

Fig. 6 Hardware setup

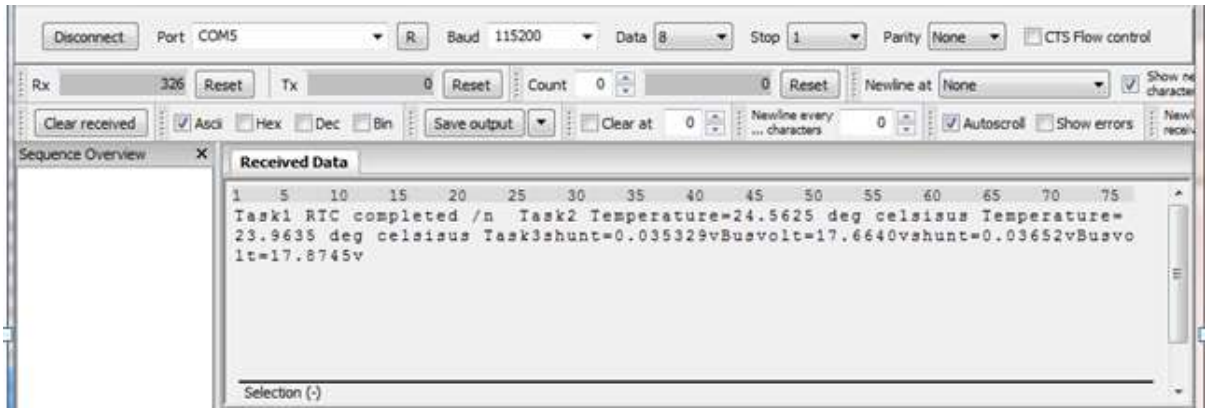

Fig. 7 HTerm Display

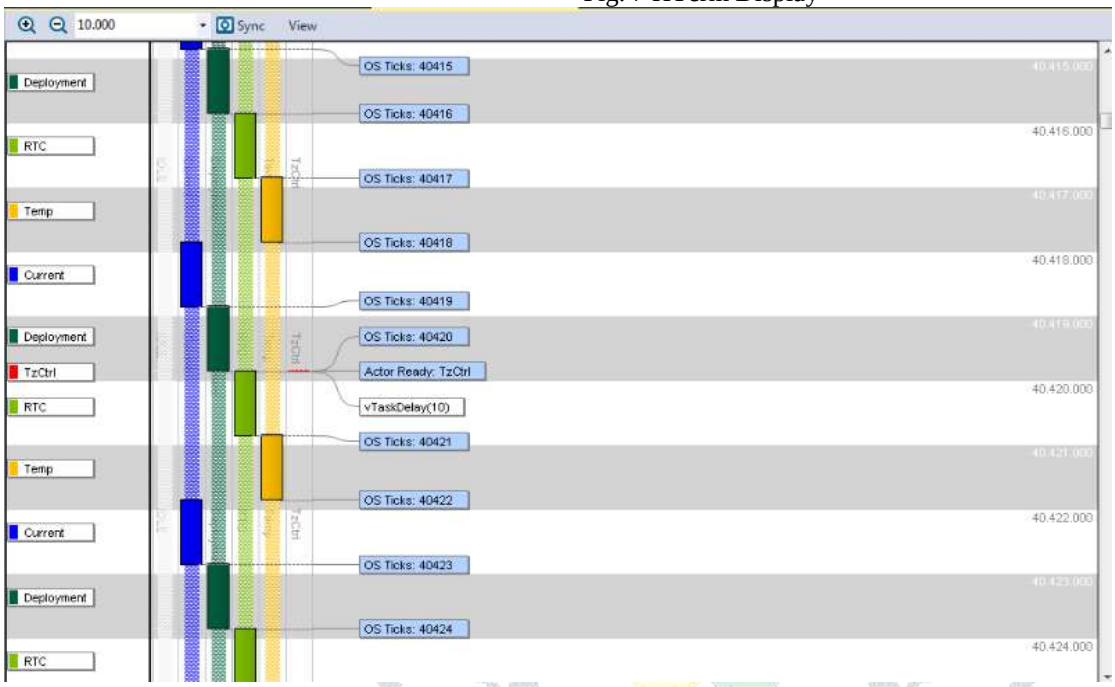

Fig. 8 Tracealyzer Output

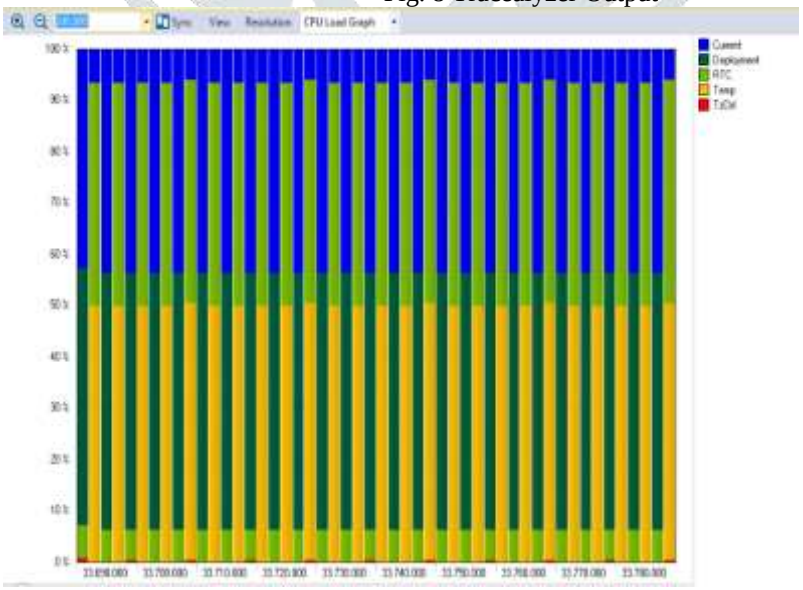

Fig. 9 CPU load graph

ACKNLOWEDGEMENT

I would like to extend my sincere regards and gratitude to the team members of STUDSAT-2 Divyanshu, Kannan T and Somnath Singh for their support and guidance in the completion of the project.

.

REFERENCE

1] B. Rajulu, S. Dasiga, and N. R. Iyer, "Open source RTOS implementation for the on-board computer (OBC) in STUDSAT-2", in 2014 IEEE Aerospace Conference, 2014, pp.1-13.

2] Agfianto Eko Putra ,Tri Kuntoro Priyambodo, and Noris Mestika,"Real-time Operating System Implementation on OBC/OBDH for UGMSat-1 Sequence" , Published by the American Institute of Physics, Citation: 1755, 170009 (2016).

3] J. Slackaa, S. Petrikb, M. Halasa, Slovak "Embedded RTOS for skcube satellite", faculty of electrical engineering and information technology slovak university of technology 2014.

4] A. E. Putra, B. A. A. Sumbodo, C. Atmaji, and M. S. H. Ahmad, "Purwarupa On-Board Data Handling (OBDH) berbasis Mikrokontroler LPC1769 untuk satelit IiNUSAT-1", in 13th Proceeding of Seminar on Intelligent Technology and Its Applications (SITIA 2012), ITS, Surabaya, 2012.

5] Juraj Slacka "Safety Critical RTOS for Space Satellites", Institute of Robotics cybernetics Faculty of Electrical Engineering and Information Technology Bratislava, Slovakia, 2015 International Conference on Process Control (PC) June 9–12, 2015

6] 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.

7]Http://www.manipalthetalk.org/colleges/mit/clubs/pariksit-student-satellite-team.