# Design and Implementation of Mixed Criticality Systems for CubeSat Applications

Undergraduate Thesis

*Submitted in partial fulfillment of the requirements of*
*BITS F421T Thesis*

*By*

Gayatri Kedar Patankar
ID No. 2019A7TS1006G

*Under the supervision of:*

Dr. Kunal Kishore Korgaonkar


BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, GOA CAMPUS
May 2023

# Declaration of Authorship

I, Gayatri Kedar Patankar, declare that this Undergraduate Thesis titled, 'Design and Implementation of Mixed Criticality Systems for CubeSat Applications' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Gayatri Kedar Patankar

Date: 17/5/23

# Certificate

This is to certify that the thesis entitled, "*Design and Implementation of Mixed Criticality Systems for CubeSat Applications*" and submitted by Gayatri Kedar Patankar ID No. 2019A7TS1006G in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

---

*Supervisor*
Dr. Kunal Kishore Korgaonkar
Assistant Professor,
BITS-Pilani Goa Campus
Date:

# *Acknowledgements*

I would like to thank my supervisor Prof. Kunal Korgaonkar, for his support and input in this thesis. This work has been a part of Project EINSat, the CubeSat project of BITS Goa. I am deeply grateful for the resources I have received as a member of this project, such as access to Sandbox, a student operated lab at the BITS Goa campus. I am also indebted to the support I received from the members, past and present, of Project EINSat. Without my peers and my family I would not have been able to complete this work to a degree of satisfaction.

# Contents

# List of Figures

# Abbreviations

| | |
|---|---|
| **WCET** | **W**orst **C**ase **E**xecution **T**ime |
| **MCS** | **M**ixed **C**riticality **S**ystem |
| **ISA** | **I**nstruction **S**et **A**rchitecture |

# Chapter 1

# Understanding Mixed Criticality Systems

## 1.1 Introduction

Mixed criticality systems are those which have multiple applications with different degrees of criticality running on the same computational resources or sharing the same communication framework. This notion of criticality is primarily taken in the safety context, with more critical tasks as those which are more vital to the safe continued operation of the system. Often, mixed criticality systems are used for real time applications and are formally defined in real time analysis. In practice, this can refer to many systems in real life, including vehicles and other embedded systems. Mixed criticality requires that the applications maintain sufficient independence or freedom from interference among themselves. The study of mixed criticality systems often focuses on mechanisms that can create independence of tasks in the functional and time domains, and the architecture and design of the components required for this.[5]

## 1.2 Formal model

Mixed criticality systems are often real time systems, as the different applications to be coordinated are usually to be synchronized in the time domain. In the study of real time systems, a task $\tau_i$ in a mixed criticality system is characterized by the following attributes:

$\tau_i = \{c_i, T_i, \pi_i, C_i, D_i\}$

Here,

$c_i$ represents the criticality level of the task,
$T_i$ represents the period of the task,
$\pi_i$ represents the priority of the task,
$C_i$ represents the worst case execution time of the task, and
$D_i$ represents the deadline of the task.

For a task, $C_i$ is a vector of length $c_i$, each value corresponding to the criticality level of its index. For lower criticality levels, a simpler estimation process is used. This results in a WCET which is not usually exceeded during regular operation. Higher criticality levels correspond to more and more stringent values of WCET, which may not be exceeded in practice but are required for certification purposes. Often, a dual criticality level model is used for simplicity.[6][7]

However, this model focuses on obtaining graceful degradation of performance, that is, sacrificing lower criticality tasks for higher ones when required. The spirit of mixed criticality is for sufficient independence to be brought about in the functional and time domains, which is not duly honoured in this model alone.

## 1.3 Safety integrity level determination

Different standards have different methods of determining the criticality levels. The following is the risk matrix for the standard ISO 26262.[4] Here, severity is determined by the extent of harm that can occur in the situation resultant from a failure of this degree, probability corresponds to each level of harm, while controllability is the degree to which harm can be avoided.

| Severity class | Probability | Controllability class | | |
|---|---|---|---|---|
| | | C1 | C2 | C3 |
| S1 | E1 | QM | QM | QM |
| | E2 | QM | QM | QM |
| | E3 | QM | QM | A |
| | E4 | QM | A | B |
| S2 | E1 | QM | QM | QM |
| | E2 | QM | QM | A |
| | E3 | QM | A | B |
| | E4 | A | B | C |
| S3 | E1 | QM | QM | A |
| | E2 | QM | A | B |
| | E3 | A | B | C |
| | E4 | B | C | D |

Here,
S1-3 represent the severity class of the event, corresponding to the extent of harm that can occur. S1 is least severe while S3 is most severe.
E1-4 represent the frequency of exposure to this type of event. E1 is most unlikely while E4 is highly likely.
C1-3 represent the controllability class of the event, corresponding to what can be done to mitigate the harm caused in the case of such an event. C1 is most controllable while C3 is least controllable.
QM or Quality Management indicates that no additional safety measures are required,
A-D are ASILs (Automobile Safety Integrity Levels) which represent the overall degree of hazard from lowest to highest.

These levels can extend up to S0-S3, E0-E4, C0-C3. However we only consider the partitions given above in the majority of cases.

As this standard is ordinarily used in automobile applications, the categories are given as below:
S0: No injuries
S1: Light to moderate injuries
S2: Severe to life-threatening (survival probable) injuries
S3: Life-threatening (survival uncertain) to fatal injuries

E0: Incredibly unlikely

E1: Very low probability (injury could happen only in rare operating conditions)
E2: Low probability
E3: Medium probability
E4: High probability (injury could happen under most operating conditions)

C0: Controllable in general
C1: Simply controllable
C2: Normally controllable (most drivers could act to prevent injury)
C3: Difficult to control or uncontrollable

The ASIL can generally be expressed as
ASIL = Severity × (Exposure × Controllability)

For example, in an automative vehicle, rear lights are classed under ASIL A, while head lights and brake lights would fall under ASIL B, cruise control would be under ASIL C. The most important components in terms of safety such as air bags, power steering, and brakes would be categorized under ASIL D.[4] These would be subjected to the highest degree of scrutiny.

Considering the case of cruise control, this process would be estimated to fall under the severity class S3. This is because the injuries caused by the failure of the same can easily become life threatening. Cruise control when used in the wrong circumstances can increase the risk of an accident and cause sudden unintended acceleration. The control of the speed of the vehicle is incredibly vital in the prevention of accidents. The probability of the an accident occurring would be medium and hence it can be classed under E3. In this case it would be difficult to control and hence would be considered as C3. This indicates that it indeed would be of the ASIL C.

The case of brakes would clearly fall under S3, E4, and C3. This is because the failure of brakes can cause the most catastrophic of accidents which become very probable. It is also incredibly difficult to control these events through any action of the driver. Hence, it is considered ASIL D.

Each of the ASILs have a different set of requirements which must be adhered to in order to conform to the standard. These become more and more strict from A-D. The requirements for QM are not safety based but rather focused on improving the quality of the product and the user's experience. The mechanisms used to ensure mixed criticality must meet the requirements

of these ASILs. Mechanisms which meet the criteria for ASIL D also meet all those lower than it, hence many manufacturers get the most critical aspects of their product certified with this level.

A dual criticality mode system, as is commonly found in most theoretical models of mixed criticality systems, can be created by condensing ASILs into two overarching levels of criticality. For example, the criticality levels A and B can be classed as low criticality while C and D may be considered high criticality. Similar splits are also possible. Hence, the point at which the division between the criticality levels is done is important in the determination of the extent of importance of the ASILs.

## 1.4 Modes

The behaviour of the system upon a mode change request is extremely important. Normally, when a mode change request comes in, there is a certain time period between the two modes referred to as the offset. Offsets for each pair of modes can be represented in the format of a matrix where the row and column each correspond to modes. Not every element of the matrix will be used in practice, rather, only those which correspond to the flowchart for the specific use case would be relevant. Deciding the length of the offset depends upon the tradeoff between completing old mode jobs to ensure that the transition is smooth and error free and both maintaining the periodicity of mode independent jobs as well as completing new mode jobs as soon as possible. The latter is especially important for emergency modes, when every instant is critical to the success of the mission.

The protocols followed in this transition period can be classified as synchronous and asynchronous protocols.[8] In synchronous protocols, new mode jobs are only released after the transition period while in asynchronous protocols, new mode jobs are released upon the mode change request and are handled along with the old mode jobs in the interim phase. The offset is calculated based on the protocol followed in the transition phase and the schedulability analysis of the same.

## 1.5 Implications Upon Scheduling

In essence, the ASIL table assists us in determining the criticality of each task. The majority of scheduling algorithms used for mixed criticality systems focus on graceful degradation of performance.

In practice, this means that they prioritize higher criticality tasks over lower ones and will kill the latter if a sacrifice needs to be made in a scheduling scenario. At every mode switch, a decision must be made about whether the new job released is to be prioritized over whichever job may be running at that instant, and whether this would cause any deadlines to be missed. One of the principles of mixed criticality systems is that every task is independent, which is to be upheld as far as possible, but when a choice needs to be made between two jobs which will miss their deadlines, the lower criticality job is sacrificed.

If no mode switch occurs, which is an unlikely case, no sacrifices need be made and as long as certain criteria are met it is guaranteed that the system will be schedulable.

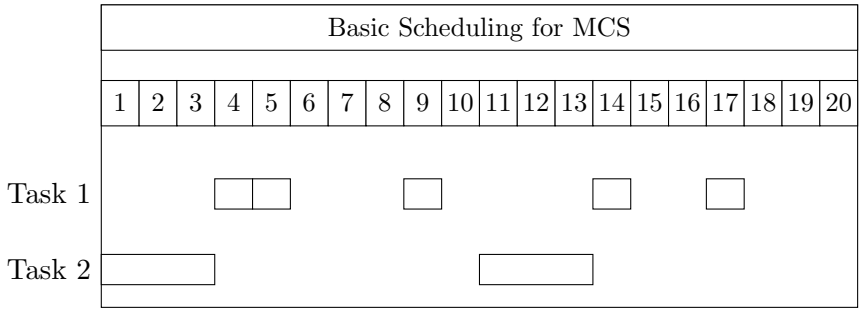
## 1.6 Examples of Scheduling

For example, we can have two tasks in a given system, one at the higher criticality level while the other is at the lower criticality level:

$\tau_1 = \{1, 4, 2, (1), 6\}$
$\tau_2 = \{2, 10, 1, (2, 3), 4\}$

These two tasks exist in a dual criticality mode system, as is seen due to the presence of only 2 criticality levels.

We assume at the release times of the tasks are synchronized. Using a simple priority based algorithm, which is often used for mixed criticality systems, we get the resulting Gantt chart:



This is the simplest possible example. It is possible to have multiple tasks at the same criticality level as well as more than 2 criticality levels.

Several algorithms are used for scheduling mixed criticality tasks in real time systems, including hybrid modified EDF such as EDF-VD (EDF with Virtual Deadlines), OCBP (Own Criticality Based Priority), AMC (Adaptive Mixed Criticality), among others. Here, the OCBP algorithm is used. We will see more examples of other scheduling algorithms below.

It is possible for mode switches to occur during scheduling, which can require the use of a protocol to handle situations resulting from the same. It is also possible for situations to arise wherein lower criticality tasks must be sacrificed in order to complete high criticality tasks. We will see an example of these scenarios below:

$\tau_1 = \{1, 4, 2, (1), 6\}$
$\tau_2 = \{2, 10, 1, (1, 2), 4\}$
$\tau_3 = \{1, 5, 2, (2), 8\}$
$\tau_4 = \{2, 20, 1, (2, 3), 17\}$

For scheduling these tasks, we will use the EDF-VD algorithm. This algorithm includes a pre-processing stage. We will calculate modified period of each task and further determine virtual deadlines, which are used for scheduling until high criticality mode is activated.[2]

First, we will calculate the utilization of each level of jobs at each criticality level:

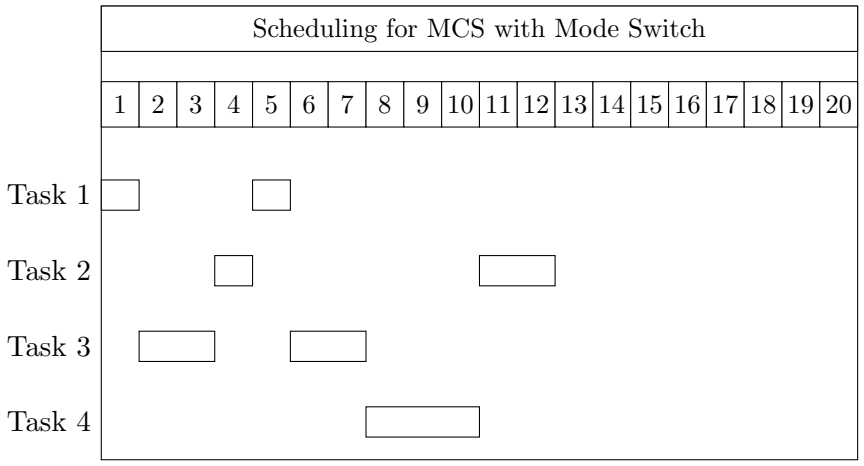
$U_1(1) = u_1(1) + u_3(1) = \frac{c_1(1)}{p_1} + \frac{c_3(1)}{p_3} = \frac{1}{4} + \frac{2}{5} = 0.65$
$U_2(1) = u_2(1) + u_4(1) = \frac{c_2(1)}{p_2} + \frac{c_4(1)}{p_4} = \frac{1}{10} + \frac{2}{20} = 0.2$
$U_2(2) = u_2(2) + u_4(2) = \frac{c_2(2)}{p_2} + \frac{c_4(2)}{p_4} = \frac{2}{10} + \frac{3}{20} = 0.35$
$x = \frac{U_2(1)}{1-U_1(1)} = \frac{0.2}{1-0.65} = 0.57$
$xU_1(1) + U_2(2) = 0.57 * 0.65 + 0.35 = 0.72$
Hence, we see that the system is schedulable.

Further, we find the modified periods:
$\hat{T_1} = 0.57 * 4 = 2.28$
$\hat{T_2} = 0.57 * 10 = 5.7$
$\hat{T_3} = 0.57 * 5 = 2.85$

$\hat{T_4} = 0.57 * 20 = 11.4$

Hence, the relative virtual deadlines of each task are:
$\tau_1 \rightarrow 4$
$\tau_2 \rightarrow 5.7$
$\tau_3 \rightarrow 5$
$\tau_4 \rightarrow 11.4$

Now, we will follow the EDF algorithm apart from the fact that we will switch into high criticality mode if any job executes beyond its low criticality WCET.




Upon the release of the first jobs of every task, we consider the job with the earliest deadline for execution. This is that of $\tau_1$. At the beginning of the second time unit, the closest deadline is that of $\tau_3$, which will then execute for the next 2 time units. Then, we proceed to complete the job of $\tau_2$ which would then continue for the next 1-2 time units. In this case it has taken 1 unit of time. At this point, the second job of $\tau_1$ is released. This has an absolute deadline at 8 time units. Hence, it is taken as the next job, executing for the next time unit. Now, the second job of $\tau_3$ is released which has a deadline of 10 units. It is taken to be executed for the next 2 time units. Then, the job of $\tau_4$ is next for execution. If it crosses its low criticality execution time of 2 units and instead executes for 3 units, we switch into high criticality mode. We see that this occurs at the 9th time unit. From this point on, we will not service jobs of $\tau_1$ and $\tau_3$ as they are low criticality tasks. For $\tau_2$ and $\tau_4$, we use the regular deadlines rather than the virtual ones. At the beginning of the 11th time unit, the second job of $\tau_2$ is released. It is then scheduled for execution for the next two time units.

Hence we see that low criticality jobs are sacrificed in order to complete high criticality jobs. In this case, it resulted in the processor remaining idle. This is not an ideal outcome but it is deemed necessary by the algorithm so as to maintain safety and mission integrity.

## 1.7 Worst Case Execution Time

The calculation of WCET is a nontrivial problem in the analysis of mixed criticality systems as well as real time systems by extension.[3] It must be estimated as accurately as possible so that scheduling can be done efficiently. With the additional constraint of meeting the mixed criticality standard in use, multiple values of execution time with increasing strictness must be calculated. For this, static as well as measurement based methods can be used.

Static methods are preferred in the case that the processor used has predictable and well understood behaviour which can be analyzed along with the code of the task to be executed to produce the WCET. Measurement based methods on the other hand are often used to find estimates rather than bounds and can be used for code snippets and then be integrated for the whole program.

# Chapter 2

# Application to EINSat system

## 2.1 Introduction

The mixed criticality paradigm can be applied to real time systems such as a CubeSat with relative ease. In this case, the CubeSat project aims to detect exoplanets using the transit method. This system operates in various modes depending on the position of the satellite with respect to its orbit as well as over the duration of the mission. In this case, there is the added complexity of the modes. At each instance when the mode is to be switched, there is a certain protocol to be followed with respect to the job that was executing in the previous mode. Its requirements and termination must be carefully handled.

It is important to realize that the scientific observations can only be carried out during eclipse, which accounts for 32.6 minutes of the 92.6 minute duration of the orbit. Each observation period lasts 20 minutes, and is undertaken 2-4 times in 16 orbits, which amounts to a day.

## 2.2 Modes and related behaviour

The modes of the system are deployment mode, orbiting and calibration mode, nominal day mode, nominal night mode, safety mode, and deorbiting mode. They can be represented as the following:

Here, deployment mode involves the release of the CubeSat by its launching apparatus, a poly picosatellite orbital deployer (P-POD). Prior to this, it is housed within the P-POD and is ejected into space once it reaches its orbit. Then, detumbling is done using the B-dot algorithm. Here, the time derivative of the magnetic field vector $B\cdot$ is used as feedback to the coil output

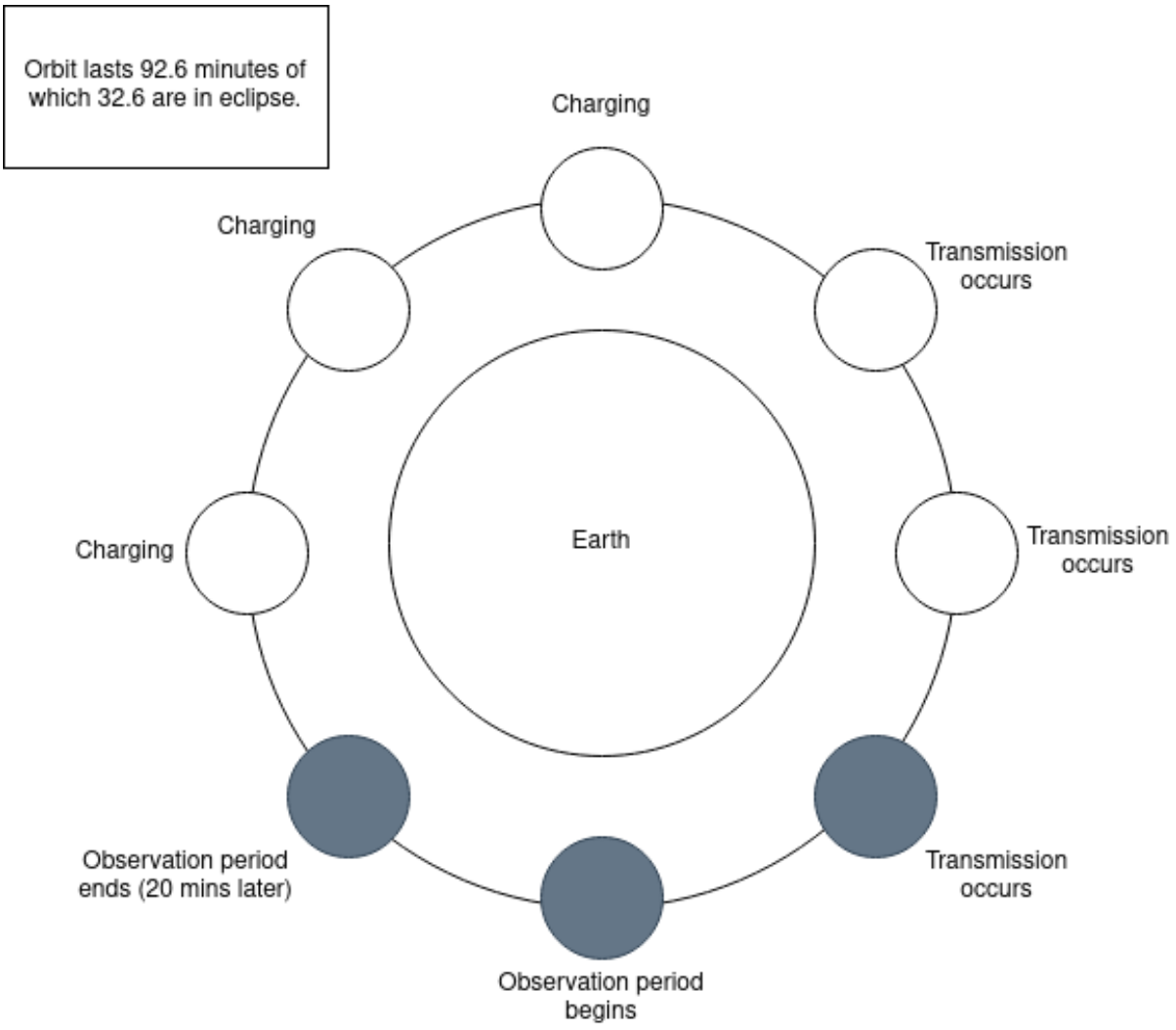



FIGURE 2.1: An orbit of the CubeSat

to slow the rotation of the satellite. This is done by magnetic torquers.

After this comes orbiting and calibration mode. Here, all the sensors involved in the attitude determination and control system, such as the sun sensors, gyroscope, accelerometers, and magnetometer, are calibrated so that they will be able to operate with minimal error.

Then, the CubeSat enters the beginning of the control loop. This involves recording readings from the sensors at a period of the average of the refresh rates of each. These readings are merged using sensor fusion with the QUEST algorithm meant to convert the readings to quaternions. This is then further processed using Kalman filtering and used as feedback for the actuators, including PID controllers and reaction wheels. Based on this, the CubeSat is able to maintain a stable orientation. This is called the nominal mode. It has two variants based on whether the

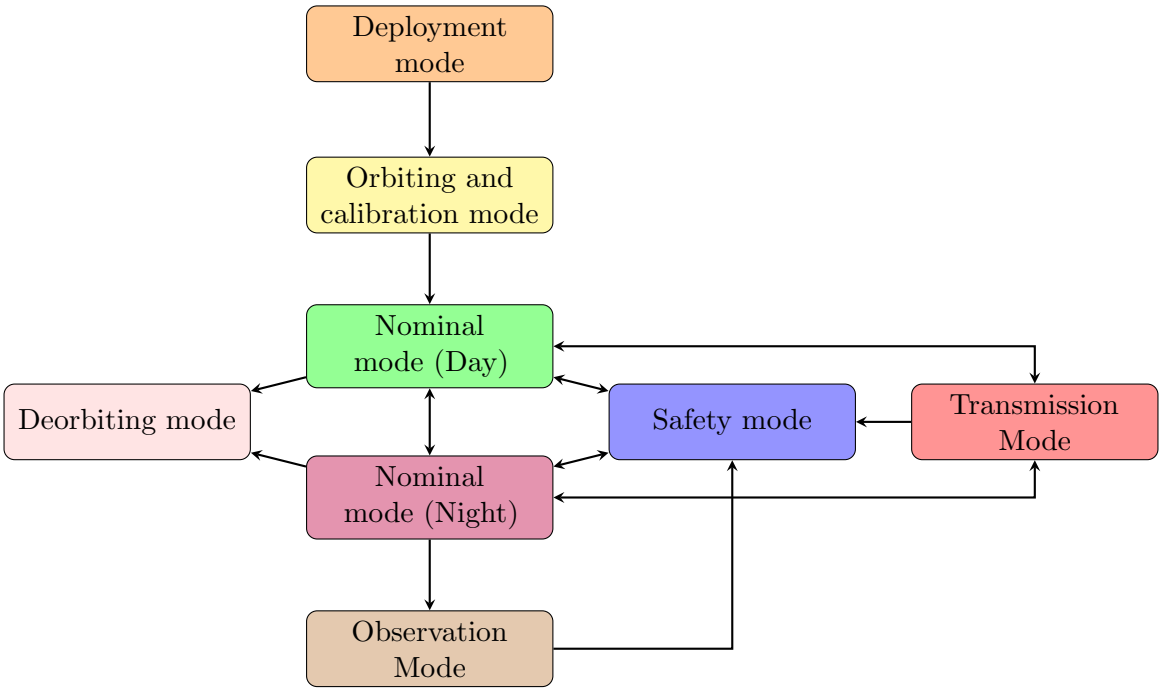



FIGURE 2.2: Flowchart describing the progression of the CubeSat's modes

CubeSat has access to sunlight. In the day, when sunlight is available, the battery is charged if necessary using solar panels. The sun sensor would be used along with the other sensors. In the night time, the star tracker is used in lieu of the sun sensor along with the remaining sensors.

It is possible for the CubeSat to reach saturation in a relatively rare case, where the actuators would be unable to control the attitude of the system. In this version of nominal mode, the attitude determination and control system of the CubeSat takes a more passive approach, using only a single sensor and actuator. This is called the safety mode. It can also be called the battery saving mode.

## 2.3 Task Set

We have seen the modes present in the CubeSat and their transitions between them. This can be elaborated to describe the full behaviour of the CubeSat system in terms of tasks. Hence, each mode can be represented as purely a set of tasks that characterize it. Hence, transitions between modes can be carried out through the action of mode change jobs.

**Deployment Mode:**

- On board computer powered up.
- Solar panels deployed.

These are jobs rather than tasks as they need only be done once. In this sense, deployment mode falls solely to the initialization phase of the CubeSat.

**Orbiting and Calibration Mode:**

- Detumbling using B-dot algorithm. This is required as the CubeSat must be stabilized so that it can be more precisely controlled in terms of attitude. This occurs after deployment of the system.
- Initial reading of sensors taken for calibration.

Similar to the previous mode, these are jobs rather than tasks and are part of the set-up of the system prior to entering the attitude determination and control loop.

**Nominal mode (Day):**

1. Sensors, excluding sun sensor and star tracker.
2. Sun sensor.
3. Sensor fusion by Kalman filtering.
4. QuEst (Quaternion Estimation) algorithm performed to calculate future course of action.
5. Output to actuators.

Each of these tasks has a dependency on the task preceding them. These together form the attitude determination and control loop for the day phase of the nominal mode.

- Charging.
- Stacking images.

**Nominal mode (Night):**

1. Sensors, excluding sun sensor and star tracker.

2. Star tracker.
3. Sensor fusion by Kalman filtering.
4. QuEst (Quaternion Estimation) algorithm performed to calculate future course of action.
5. Output to actuators.

Each of these tasks has a dependency on the task preceding them. These together form the attitude determination and control loop for the night phase of the nominal mode.

- Stacking images.

**Observation Mode:**

- Pointing to star to be observed. (Job)
- Imaging using payload.

1. Sensors, excluding sun sensor and star tracker.
2. Star tracker.
3. Sensor fusion by Kalman filtering.
4. QuEst (Quaternion Estimation) algorithm performed to calculate future course of action.
5. Output to actuators.

Each of these tasks has a dependency on the task preceding them. These together form the attitude determination and control loop.

**Transmission Mode:**

- Transferring data to be transmitted over bus.

1. Sensors, excluding sun sensor and star tracker.
2. Star tracker.
3. Sensor fusion by Kalman filtering.
4. QuEst (Quaternion Estimation) algorithm performed to calculate future course of action.

5. Output to actuators.

Each of these tasks has a dependency on the task preceding them. These together form the attitude determination and control loop.

**Safety Mode:**

1. Single sensor.
2. Sensor fusion by Kalman filtering.
3. QuEst (Quaternion Estimation) algorithm performed to calculate future course of action.
4. Output to actuator.

Each of these tasks has a dependency on the task preceding them. These together form the attitude determination and control loop.

**Deorbiting Mode:**

- Orient towards atmosphere.

## 2.4 Scheduling

Scheduling tasks in the EINSat system draws greatly from pure EDF-VD scheduling as seen the in previous chapter. However, problems arising from the nature of the mission are possible, as we will see.

For example, if one or more reaction wheels fail, the corresponding task will overrun its virtual deadline, which will trigger saturation or high criticality mode. We will transition to safety mode and if functionality is restored, we will be able to return to nominal mode and normal functioning of the mission as intended.

If saturation is reached otherwise, due to a malfunction in the hardware of the attitude determination and control apparatus, we will still proceed to safety mode. If the battery is unable to maintain normal functioning of the system, we will again proceed to safety mode until the

battery is charged sufficiently. A similar case will occur in the event of overheating.

In the case of irreparable damage, the mission parameters and objectives can be reduced in accordance with the lost functionality. An offline backup schedule can be prepared for these cases.

As seen previously, the EDF-VD algorithm is favoured for mixed criticality systems, particularly of the kind we see in a CubeSat, where the majority of tasks are periodic in nature. This schedules tasks within a mode of the system as seen in the flowchart above. This is done for a task set at a time. The pseudocode of the same is given below.

**Scheduler:**
1) Collate set of tasks with parameters.
2) Calculate utilization of each level of jobs at each criticality level for task set.
3) Verify schedulability of the system by ensuring that $x$ is less than or equal to 1 along with the net utilization.
4) Find modified periods and allocate virtual deadlines accordingly.
5) Proceed with EDF algorithm using virtual deadlines and actual deadlines as per requirement.

In addition to this, we also need to switch between system modes when required. This meta scheduler is represented by a finite state machine of the system modes.

**Meta Scheduler:**
1) Deployment mode:

- If all tasks are completed, proceed to orbiting and calibration mode.

2) Orbiting and calibration mode:

- If all tasks are completed, proceed to the day phase of nominal mode.

2) Day phase of nominal mode:

- If night phase begins, proceed to night phase of nominal mode.
- If signal is received to begin transmission, proceed to transmission mode.

- If signal is received to end mission, proceed to deorbiting mode.
- If signal is received indicating saturation is reached, proceed to safety mode.

3) Night phase of nominal mode:

- If day phase begins, proceed to day phase of nominal mode.
- If signal is received to begin observation, proceed to observation mode.
- If signal is received to begin transmission, proceed to transmission mode.
- If signal is received to end mission, proceed to deorbiting mode.
- If signal is received indicating saturation is reached, proceed to safety mode.

4) Observation mode:

- If all tasks are completed proceed to safety mode.

5) Safety mode:

- If system is stabilized and it is day phase, proceed to day phase of nominal mode.
- If system is stabilized and it is night phase, proceed to night phase of nominal mode.

6) Transmission mode:

- If all tasks are completed, proceed to safety mode.

7) Deorbiting mode:

- End mission.

## 2.5 Worst Case Execution Time

Finding the WCET of tasks is extremely important for timing analysis and scheduling of the system, as we have seen in the previous chapter. Here, the OTAWA toolset developed at the University of Toulouse is used for the same.[1] Before performing WCET analysis, the task must be converted into the Executable and Linkable Format for the target processor. Hence, it is compiled and statically linked for a 32 bit ARM processor, using the corresponding GCC

compiler. Once the ELF form of the program is obtained, we can perform WCET analysis using the OTAWA set of tools. An example of WCET analysis for the benchmark to check whether a number is prime and an example program for the same is taken. Here, the most basic ISA is used, as indicated by 'trivial'.

```
g@compy:~/otawa_home/arm-elf/prime$ owcet -s trivial prime.elf
INFO: plugged otawa::trivial (/home/g/otawa_home/otawa/lib/otawa/otawa/trivial.s
o)
WARNING: otawa::FlowFactLoader 1.4.0:no flow fact file for prime.elf
WARNING: otawa::ipet::FlowFactLoader 2.0.0:no limit for the loop at prime + 0x4c
 (000082d8).
WARNING: otawa::ipet::FlowFactLoader 2.0.0: in the context [FUN(0000828c)]
WARNING: otawa::ipet::FlowFactLoader 2.0.0:no limit for the loop at __umodsi3 +
0x20 (00008440).
WARNING: otawa::ipet::FlowFactLoader 2.0.0: in the context [FUN(00008420)]
WARNING: otawa::ipet::FlowFactLoader 2.0.0:no limit for the loop at __umodsi3 +
0x34 (00008454).
WARNING: otawa::ipet::FlowFactLoader 2.0.0: in the context [FUN(00008420)]
WARNING: otawa::ipet::FlowFactLoader 2.0.0:no limit for the loop at __umodsi3 +
0x50 (00008470).
WARNING: otawa::ipet::FlowFactLoader 2.0.0: in the context [FUN(00008420)]
WARNING: otawa::ipet::FlowFactConstraintBuilder 1.1.0:no flow fact constraint fo
r loop at BB 8 (000082d8)
WARNING: otawa::ipet::FlowFactConstraintBuilder 1.1.0:no flow fact constraint fo
r loop at BB 6 (00008440)
WARNING: otawa::ipet::FlowFactConstraintBuilder 1.1.0:no flow fact constraint fo
r loop at BB 7 (00008454)
WARNING: otawa::ipet::FlowFactConstraintBuilder 1.1.0:no flow fact constraint fo
r loop at BB 14 (00008470)
```

An issue arises as the bounds of the loops must be specified precisely for accurate estimation of WCET. For this, we use the 'mkff' command to generate the flow-fact file for the same. Then, we are able to enter the bounds of the loop by hand.

```
g@compy:~/otawa_home/arm-elf/prime$ mkff prime.elf >> prime.ff
mkff: 15722 micro-seconds
g@compy:~/otawa_home/arm-elf/prime$ gvim prime.ff
```

Then, we can proceed to calculate the WCET without error.

```
g@compy:~/otawa_home/arm-elf/prime$ owcet -s trivial prime.elf
INFO: plugged otawa::trivial (/home/g/otawa_home/otawa/lib/otawa/otawa/trivial.s
o)
WCET[main] = 13740 cycles
```

## 2.6 Implementation

The implementation of the full system has been done using FreeRTOS and tested using the Posix port for Linux. Specifically, the version v202112.00 of FreeRTOS was used on Ubuntu 22.04. Both the meta scheduler as well as the scheduler can be represented as a set of tasks which can be created and destroyed as the need arises. The common tasks which are dependent on each other are considered together in a set. The scheduling algorithm is carried out with the tasks which can be taken in any order. A heap is used as the priority queue for the same. Whenever a task is taken for execution, it is taken out of the heap and assigned the highest priority, allowing it to execute. If needed, the next instance of the same is added to the heap at its release time.

```
g@compy:~/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/my_dir$ make
mkdir -p build
gcc -I. -I/home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Source/include
-I/home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Source/portable/ThirdPa
rty/GCC/Posix -I/home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Source/po
rtable/ThirdParty/GCC/Posix/utils -I/home/g/workspace/obc/code/FreeRTOSv202112.0
0/FreeRTOS/Demo/Common/include -I/home/g/workspace/obc/code/FreeRTOSv202112.00/F
reeRTOS-Plus/Source/FreeRTOS-Plus-Trace/Include -DBUILD_DIR=\"/home/g/workspace/
obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/my_dir/build\" -D_WINDOWS_ -DTRACE_ON_
ENTER=0 -DprojCOVERAGE_TEST=0 -ggdb3 -O3 -MMD -c test.c -o build/test.o
mkdir -p build
gcc build/code_coverage_additions.o build/console.o build/main.o build/run-time-
stats-utils.o build/test.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/F
reeRTOS/Source/croutine.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/Fr
eeRTOS/Source/event_groups.o build//home/g/workspace/obc/code/FreeRTOSv202112.00
/FreeRTOS/Source/list.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/Free
RTOS/Source/queue.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS
/Source/stream_buffer.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/Free
RTOS/Source/tasks.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS
/Source/timers.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/So
urce/portable/MemMang/heap_3.o build//home/g/workspace/obc/code/FreeRTOSv202112.
00/FreeRTOS/Source/portable/ThirdParty/GCC/Posix/utils/wait_for_event.o build//h
ome/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Source/portable/ThirdParty/
GCC/Posix/port.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/De
```

```
mo/Common/Minimal/AbortDelay.o build//home/g/workspace/obc/code/FreeRTOSv202112.
00/FreeRTOS/Demo/Common/Minimal/BlockQ.o build//home/g/workspace/obc/code/FreeRT
OSv202112.00/FreeRTOS/Demo/Common/Minimal/blocktim.o build//home/g/workspace/obc
/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/countsem.o build//home/g/w
orkspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/death.o build/
/home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/dynam
ic.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Mi
nimal/EventGroupsDemo.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/Free
RTOS/Demo/Common/Minimal/flop.o build//home/g/workspace/obc/code/FreeRTOSv202112
.00/FreeRTOS/Demo/Common/Minimal/GenQTest.o build//home/g/workspace/obc/code/Fre
eRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/integer.o build//home/g/workspace/o
bc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/IntSemTest.o build//home
/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/MessageBuf
ferAMP.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Commo
n/Minimal/MessageBufferDemo.o build//home/g/workspace/obc/code/FreeRTOSv202112.0
0/FreeRTOS/Demo/Common/Minimal/PollQ.o build//home/g/workspace/obc/code/FreeRTOS
v202112.00/FreeRTOS/Demo/Common/Minimal/QPeek.o build//home/g/workspace/obc/code
/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/QueueOverwrite.o build//home/g/
workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/QueueSet.o bu
ild//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal/Q
ueueSetPolling.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/De
mo/Common/Minimal/recmutex.o build//home/g/workspace/obc/code/FreeRTOSv202112.00
/FreeRTOS/Demo/Common/Minimal/semtest.o build//home/g/workspace/obc/code/FreeRTO
Sv202112.00/FreeRTOS/Demo/Common/Minimal/StaticAllocation.o build//home/g/worksp
```

```
build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/Common/Minimal
/StreamBufferInterrupt.o build//home/g/workspace/obc/code/FreeRTOSv202112.00/Fre
eRTOS/Demo/Common/Minimal/TaskNotify.o build//home/g/workspace/obc/code/FreeRTOS
v202112.00/FreeRTOS/Demo/Common/Minimal/TimerDemo.o build//home/g/workspace/obc/
code/FreeRTOSv202112.00/FreeRTOS-Plus/Source/FreeRTOS-Plus-Trace/trcKernelPort.o
 build//home/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS-Plus/Source/FreeRT
OS-Plus-Trace/trcSnapshotRecorder.o build//home/g/workspace/obc/code/FreeRTOSv20
2112.00/FreeRTOS-Plus/Source/FreeRTOS-Plus-Trace/trcStreamingRecorder.o build//h
ome/g/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS-Plus/Source/FreeRTOS-Plus-T
race/streamports/File/trcStreamingPort.o -ggdb3 -pthread -O3 -o build/posix_demo
g@compy:~/workspace/obc/code/FreeRTOSv202112.00/FreeRTOS/Demo/my_dir$ ./build/po
six_demo

Trace started.
The trace will be dumped to disk if a call to configASSERT() fails.
Deployment mode started
Solar panels deployed
Orbiting and calibration mode started
B-dot results: q0 = 0.500000, q1 = -0.500000, q2 = -0.500000, q3 = -0.500000
Nominal day mode
Nominal mode started
Sensor reading: -1685737414
Sun sensor
```

```
x_hat = [0.000000, 0.000000]
Output to actuators
Nominal night mode
Nominal mode started
Sensor reading: -965005846
Sun sensor
x_hat = [0.000000, 0.000000]
Output to actuators
Sensors calibrated

Trace output saved to Trace.dump
```

## 2.7 Conclusion

Mixed criticality systems are a very important emerging tool for modeling systems as complex of that of a CubeSat and other nanosatellites, and even larger satellites can use a similar system and benefit from this work. Although the standards in this field may be somewhat out of date for the use of the space technology industry, we can hope that this quickly growing area will soon see development. We see that FreeRTOS is an effective tool used as an operating system for systems of this size, particularly with the help of mixed criticality scheduling algorithms.

# Bibliography


[1] Clement Ballabriga et al. “OTAWA: An Open Toolbox for Adaptive WCET Analysis”. In: (2010).

[2] Sanjoy Baruah et al. “Mixed-Criticality Scheduling of Sporadic Task Systems”. In: (2011).

[3] Wilhelm Reinhard et al. “The Worst-Case Execution Time Problem — Overview of Methods and Survey of Tools”. In: (2008).

[4] Richard Bellairs. “What is ISO 26262? Overview + ASIL”. In: (2019).

[5] Alan Burns and Robert I. Davis. “Mixed Criticality Systems—A Review”. In: (2022).

[6] Rolf Ernst and Marco Di Natale. “Mixed Criticality Systems—A History of Misconceptions?” In: (2016).

[7] et al Esper Alexandre. “How realistic is the mixed-criticality real-time system model?” In: (2015).

[8] Jorge Real and Alfons Crespo. “Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal”. In: (2004).