**UnB On-Board Computer Prototype for CubeSats**

**G. Silva Lionço(1), G. Santilli (2), L. Aguayo (3)**

Universidade de Brasília

Faculdade Gama Brasília, Brazil

1. Phone: +55 61 9 9227-8739, Mail: guilherme.lionzo@gmail.com
2. Phone: +55 61 9 8355-0174, Mail: santilli@aerospace.unb.br
3. Phone: +55 61 phone, Mail: aguayo@unb.br

**Abstract**: Researchers from the University of Brasília (UnB) are studying the feasibility of a CubeSat 3U mission, as a technology demonstrator. Some studies are already being carried out, in order to offer solutions for this future mission. The present research is aimed at the construction of an Onboard Computer (OBC) for this future mission. During the development of the OBC, it was used the co-design methodology, which allowed for the development of hardware and software at the same time. During the design of the theoretical project, it was chosen the microcontroller and another devices to compose the OBC’s hardware. For the embedded software, the FreeRTOS operating system was defined as the operating system. During the protoboard test, it was possible to verify: the consumption of the microcontroller; modes of operation of the embedded software; the acquisition and data storage; etc. It was concluded that the use of the TI MSP432 is a great choice for low-power and intermediate performance scenarios. The use of FreeRTOS as a real-time operating system for low memory systems, as well as the use of watchdog utilization at software level has been ratified.

<< meus comentários ao longo do texto. Converti para o Word para poder editar e propor uma primeira versão em inglês. Em geral está OK, sugiro focar nas contribuições e no detalhamento dos testes>>

**<< pessoal, termino na semana entrante >>**

# 1. INTRODUCTION

The feasibility of the deployment of short CubeSat 3U missions is under study at the University of Brasília (UnB). These missions are aimed to validate concepts and perform system/ subsystem/component testing for future proof-of-concepts prototypes. Initial subsystem requirements for these missions are (a) use of optical cameras for polar shields monitoring, (b) use of a Pulsed Plasma Thruster(PPT) for orbital control studies and (c) the use of an accelerometer to provide a mapping of the Earth gravitational field.

Some studies are already ongoing, but all missions will require an Onboard Computer (OBC), a mandatory subsystem responsible for control, management and monitoring of all relevant signals that needed for correct operation of the CubeSat. The following text comprises the details of a development of an OBC, from its requirements to the hardware and software specifications, as well some results from a prototype.

## 2. OBC REQUIREMENTS FOR CUBESAT3U MISSIONS

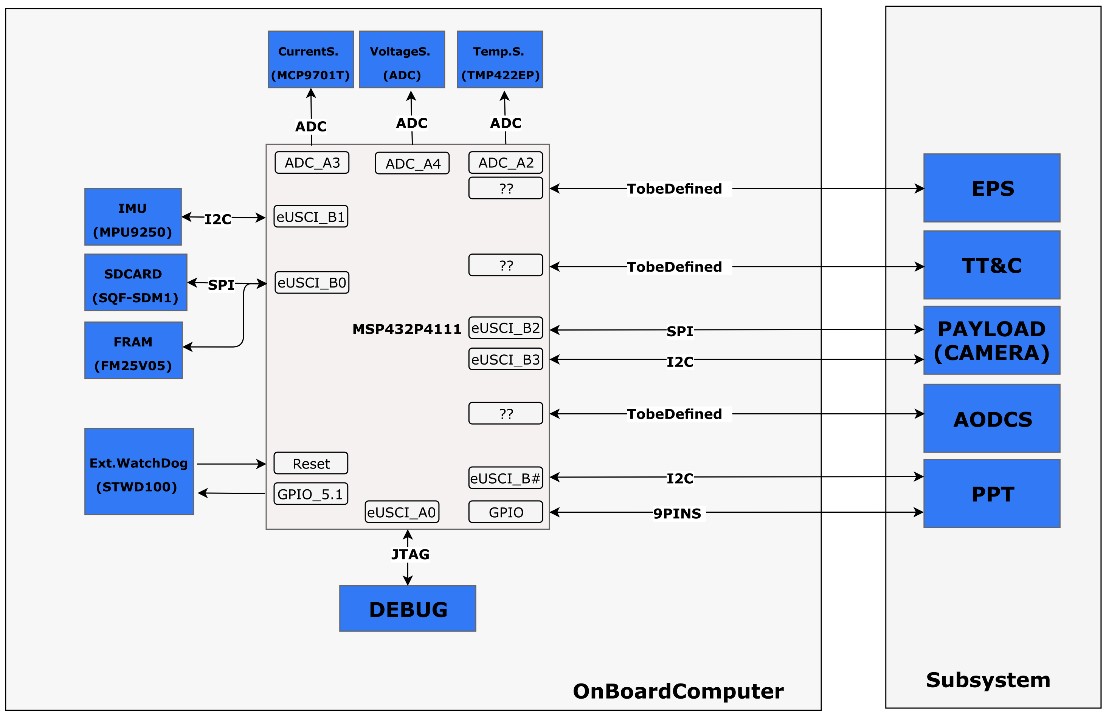
Among the primary requirements for the initial phase of the OBC subsystem project, some are depicted as follows. The list is not exhaustive but contain the major functions to be performed at this stage of prototyping.

* To control an optical camera based on CMOS (*Complementary Metal-Oxide-Semiconductor*) technology;
* To provide control for a Pulsed Plasma Thruster;
* To control satellite’s position via control and processing of signals from of an inertial sensor;
* To guarantee a high degree of overall system reliability, even with the use of commercial off-the-shelf (COTS) components;
* To store useful data in a non-volatile memory, for adequate transmission to a ground station;
* To have an anti-locking system;
* Change operation modes according to the energy availability.

## 3. OBC HARDWARE AND SOFTWARE ARCHITECTURE

A *co-design* methodology was used for the development of the OBC, i.e., there was a simultaneous and interactive development of hardware and software. Figure 1 depicts the general architecture, as well the interfaces with other subsystems. The next subsections provide details on the hardware subsystems and software functionalities.

<< sugestões: deixar o fundo branco e não cinza; usar fonte branca para os blocos azuis>>



**Figure 1 –** Block diagram of the OBC, with interfaces.

### 3.1. Hardware Architecture

<< para economizar espaço, não detalharia as especificações “do manual“, mas justificaria apenas as vantagens dos principais parâmetros >>

#### 3.1.1. Microcontroller

The first step was the assessment of different options for the microcontroller, as it is the component that imposes the main restrictions on the OBC performance. Selection criteria has considered: low power consumption, presence of and analog-to-digital converter (ADC), standard communication interfaces such as General-Purpose Input/output (GPIO), serial peripheral interface (SPI) and processing speed (clock rate). Usual COTS components were considered, and to reduce the time for the prototyping, we also required that a development kit was ready to be acquired from the corresponding manufacturer.

Among a list of candidates, the final selection was Texas Instruments MSP432P4111, due to its low power consumption and satisfactory speed performance. Its power consumption is 520 W/MHz, has memory size compatible with the mission requirements: Furthermore, it has 18 different operation modes, timers, ADCs, several I/O pins and a block of real-time clock [1].

#### 3.1.2. Memory Unit

The memory unit is responsible to store data from OBS processing and it has two subunits. The first stores telemetry data and payload, and the second stores a backup of the embedded software. For the estimative of data requirements, the team has used the parameters of *SWISSCube* mission [2]: an amount of 176 MB of data, 10 MB from telemetry and 166 MB from the optical camera. The total capacity is a function of the spatial resolution for the images only, as the radiometric and spectral resolutions are fixed. Worst and best cases for the autonomy (in days) are depicted at Table 1.

<< inserir tabela. concordo com o comentário do Giancarlo >>

<< Este cálculo dependerá da resolução espacial escolhida para as imagens (a radiométrica e espectral é fixa). Seria aconselhável inserir aqui uma tabela, com o cálculo da autonomia (em dias), na melhor das hipóteses e pior em função da resolução espacial.>>

Como não se sabe a quantidade de estações terrenas disponíveis para descarregar os dados, escolheu-se o tamanho máximo que o microcontrolador poderia suportar, que no caso é de 4GB, dando 22 dias de armazenamento.

Já para o armazenamento do backup do software embarcado, escolheu-se uma memória FRAM, devido sua resistência à efeitos TID e SEU [3]. O tamanho definido foi o dobro da memória SRAM do microcontrolador, resultando em 512KB.

<< não alterei o texto abaixo (3.1.3 e 3.1.4) por hoje, mas posso fazer na segunda. Novamente, acho importante colocar apenas os parâmetros relevantes. Minha sugestão é reduzir para deixar mais espaço para as figuras e a para as contribuições do trabalho>>

#### 3.1.3. Peripherical

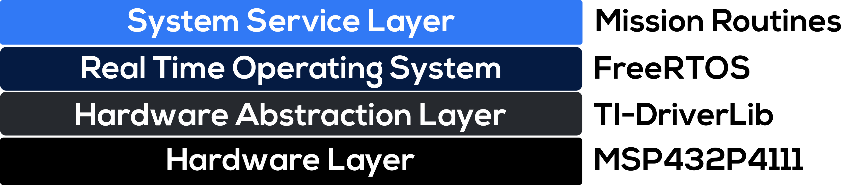
Os periféricos são componentes que, assim como a unidade de armazenamento, tem o objetivo de auxiliar o microprocessador a cumprir os requisitos do OBC. Essa unidade é composta basicamente de quatro componentes: sensor de corrente, sensor de inercial, sensor de temperatura, Watchdog externo (contador). O sensor de corrente escolhido foi o ACS70331, sensor que também serve para proteção de *overcurrent* [4]*.* O sensor inercial MPU9250 é destinado à aferição da aceleração, campo magnético e rotação nos três eixos acelerômetro, magnetômetro nos 3 eixos [5]. O sensor de temperatura MCP9701T é de fácil usabilidade e baixo consumo, sendo capaz de medir temperaturas com uma precisão de 2°C em uma faixa de temperatura de -40°C a 125ºC [6]. Por fim, o contador STWD100 tem o objetivo de trazer maior robustez ao OBC, em casos de evento SEE, servindo como Watchdog externo [7].

#### 3.1.4. Interfaces

O OBC possui duas interfaces principais, a primeira é destinada ao “debug” do microcontrolador e a segunda é destinada à comunicação com outros subsistemas do satélite. A interface de debug é destinada à fase de projeto e testes do OBC, e é composta pelos pinos do MSP432 de JTAG (*Joint Test Action Group*) e SWD (*Serial Wire Debug*) [8]. A segunda interface é reservada para a comunicação com os subsistemas do satélite, sendo feita por meio do barramento ISA de 16bit, comumente utilizado em placas do padrão PC104 [9].

### 3.2. Software Architecture

As depicted in Figure 2, a layered approach architecture was defined, in order to provide better software usability and maintenance. Mission parameters may be set at the top layer, System Service Layer (SSL). The Hardware Abstraction Layer (HAL) is provided by the microprocessor manufacturer [10] and the Free RTOS was used to meet the requirements of Hard RTOS [11].



**Figure 2 –** Layered software architecture.

<< sugestão: como nos periféricos e interfaces, retirar. Se houver algo relevante sobre a interfaces, mencionar rapidamente >>

## Camada de Abstração de Hardware

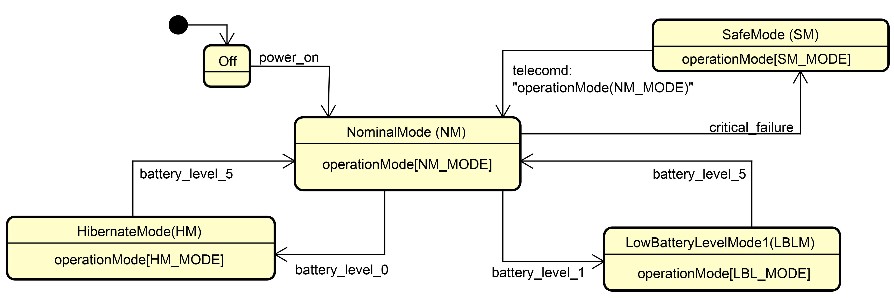
A camada de HAL é desempenhada pelo pacote *Driver Library* (*DriverLib*), desenvolvido pela Texas Instruments, que tem o intuito de facilitar o desenvolvimento de projetos embarcados e ajudar na portabilidade dos códigos. Utilizando esse pacote, o desenvolvedor não necessita saber o que acontece a nível de registrador, tornando o desenvolvimento mais amigável e rápido. [10]

## Camada do Sistema Operacional

O RTOS escolhido para o software embarcado foi o *FreeRTOS* devido sua vasta utilização em missões CubeSat. Esse kernel, desenvolvido e mantido pela Real Time Engineers Ltd, é distribuído gratuitamente sobre a licença General Public License (GPL). No contexto do FreeRTOS, cada tarefa em execução é chamada de ‘task’. No contexto do projeto, o uso das tasks é fundamental para criar um certo nível de abstração e garantir o requisito de HardRTOS. [11]

Here we focus only in the SSL, the layer with the main development effort. An UML extension, *FuncionalC* [12] allows the modeling of systems using C language, the one chosen for the code writing and testing. From the software perspective, two major representations were produced: a State Machine Diagram and a File Diagram.

The state machine has four modes of operation, depicted at Figure 3: **Nominal Mode**, **Safe Mode**, **Battery Low Level Mode** and **Hibernate Mode**.

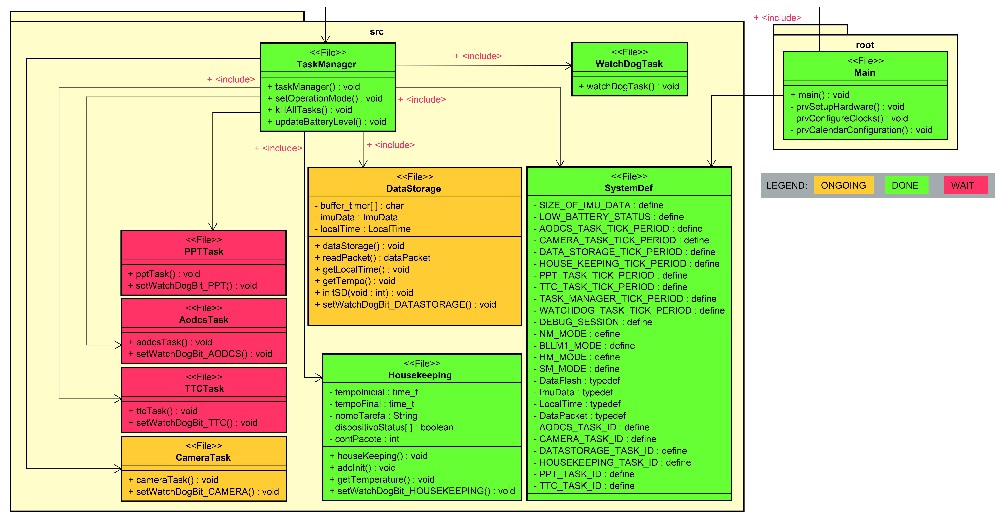


**Figure 3** – System Service Layer State Machine Diagram.

It is worth mentioning that, at this stage of prototyping, the pre-launching and deployment stages were not considered.

The File Diagram has 9 routines, presented in Figure 5: one for control (***TaskManager***), one for data collection (***HouseKeeping***), one for data storage (***DataStorage***), one for SW locking control (***WatchDogTask***), and 5 to handle CubeSat subsystems functionalities, such as telemetry, tracking and command (TTC), among others.

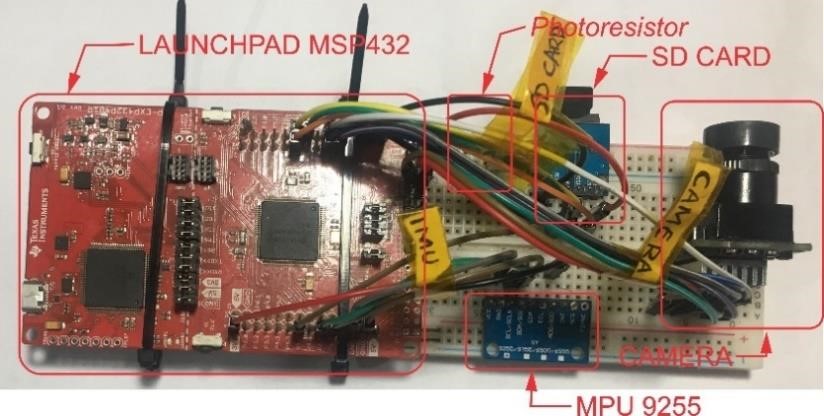
<< sugestão: trocar fontes pretas no fundo vermelho por fontes brancas >>



**Figure 4 –** System File Diagram. Green modules are completed and tested.

## 4. RESULTS

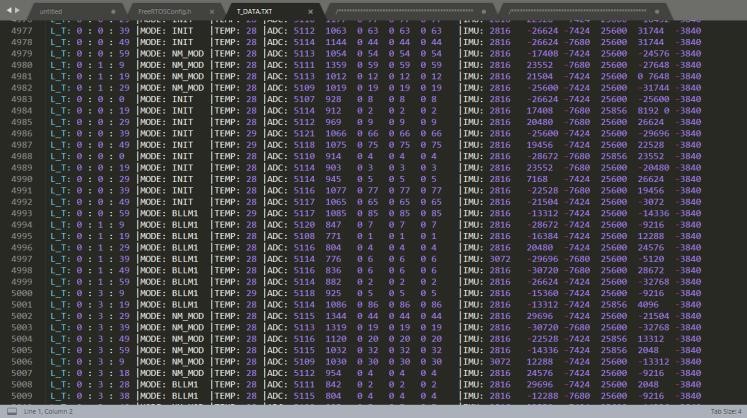
Como o hardware do OBC estava em fase de construção, o software embarcado foi simulado na *LaunchPad* do MSP432. Utilizou-se um *photoresistor* para simular o nível de bateria do EPS do CubeSat. Essa abordagem condiz, de certa forma, com a realidade pois a incidência de luz interfere na quantidade de energia armazenada no *EPS*. A Figura 5 mostra a conexão dos componentes na *LaunchPad*. Nesta foto há os módulos COTS utilizados (SD Card, MPU9255 e *photoresistor*) e a câmera (*Paylod*).



**Figura 5 –** Protoboard com os componentes COTS e a *LaunchPad*.

### 4.1. Aquisição e Armazenamento de Dados

O serviço de armazenamento de aquisição e armazenamento de dados é feito pelas tasks ***HouseKeeping*** *e* ***DataStorage***, respectivamente. Durante os testes realizados, os dados foram salvados em formato de *ASCII*, para facilitar o debug. A Figura 6 mostra o arquivo de telemetria salvado no SD Card.



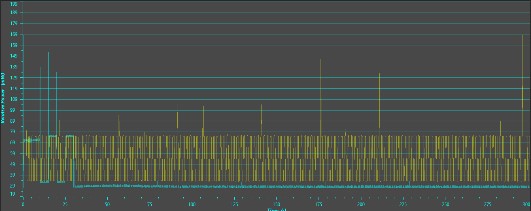
**Figura 6 –** Dados da telemetria armazenados no cartão de memória.

Em relação as fotos, [TESTES COM A CAMERA]

### 4.2. Consumo

Os testes de consumo foram realizados utilizando a ferramenta *EnergyTracer* do *Code Composer Studio*. Essa ferramenta mede a corrente sendo consumida no barramento JTAG/SW, sendo assim, ela permite calcular a corrente consumida pelos módulos/sensores alimentados pela *Launchpad*. Houveram três baterias de testes de cinco minutos, um para cada estado.

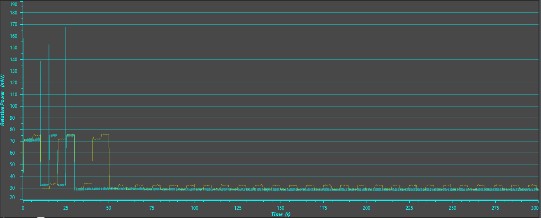
Comparando o estado nominal com o estado de hibernação, observa-se que houve uma economia de mais de 40%, aumentando a vida da bateria em dois dias. A Figura 7 mostra a comparação em forma gráfica, hibernação em azul e nominal em amarelo.



<< se possível, inverter fundo 🡪 deixar branco, curvas em Vermelho e azul >>

**Figura 7 –** Comparação entre os Modos de hibernação (azul) e Pouca Bateria (amarelo).

Não foi possível concluir a mesma melhora comparando o modo de baixo consumo com o modo de hibernação. Houve apenas uma economia de 9%, 0.6 dias. Esse fato ocorre porque o modo de hibernação não está completamente otimizado. A fonte de sincronismo durante esse estado ainda continua sendo 48MHz. O ideal seria utilizar o *clock* externo de 32KHz para realizar as interrupções do *kernel*. Infelizmente devido à inexperiência do aluno com o FreeRTOS, não foi possível adicionar uma segunda fonte de sincronismo no modo de baixo consumo. A Figura 8 mostra a comparação em forma gráfica, hibernação em azul e nominal em amarelo.

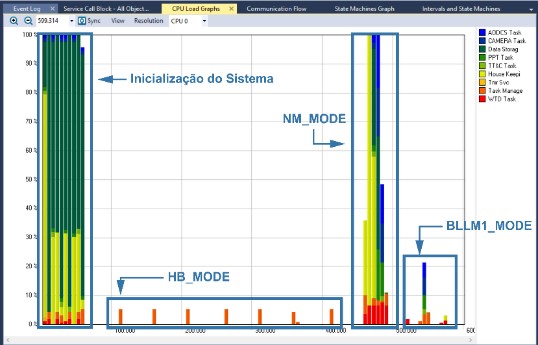


<< se possível, inverter fundo 🡪 deixar branco, curvas em Vermelho e azul >>

**Figura 8 –** Comparação entre os Modos de hibernação (azul) e Pouca Bateria (amarelo).

### 4.3. Modos de Operação

Para testar a máquina de estados do OBC, foi utilizado o *software* Tracealyzer, da empresa *Percepio* [13]. No decorrer dos testes foi possível observar vários fenômenos interessantes. Na inicialização do sistema há um grande uso de CPU, logo em seguida o sistema entra em modo de baixo consumo e apenas o ***TaskManager*** fica ativo e sendo executado mais lentamente. Após o modo de hibernação, o sistema entra em modo nominal e todas as tasks são executadas sem limite de CPU. E por fim, o sistema foi colocado em modo de baixo consumo, e apenas as *tasks* de controle(***WTDTask***e ***TaskManager***) e manipulação de dados (***HouseKeeeping***e ***DataManager***) foram executadas.

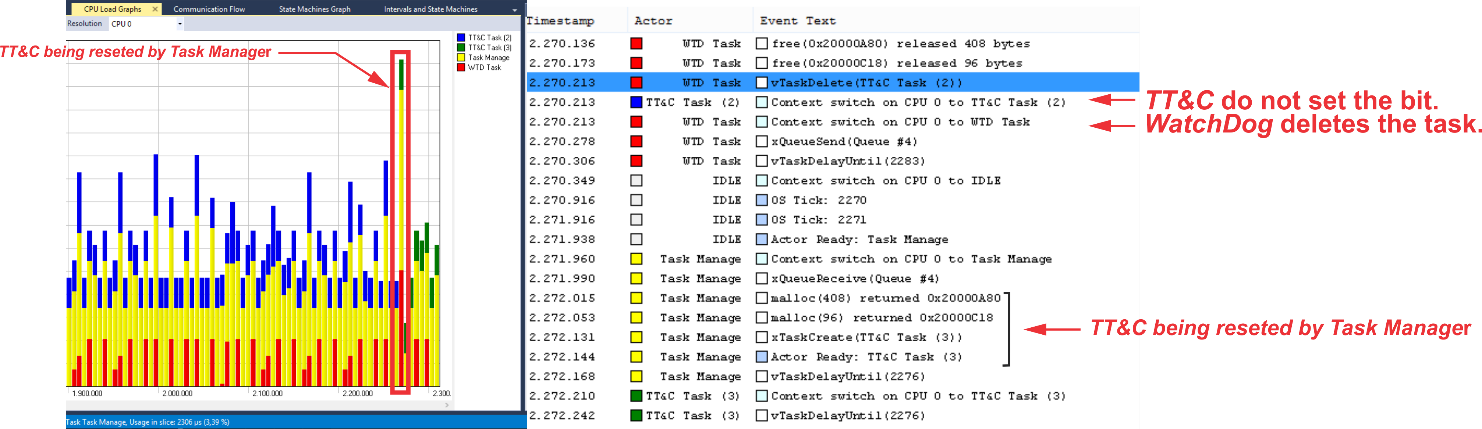


**Figura 9 –** Snapshoot no Tracealyzer do sistema executando em todos os modos.

<< esta figura é muito mais legível do que a com fundo escuro >>

### 4.4. Sistema Antitravamento

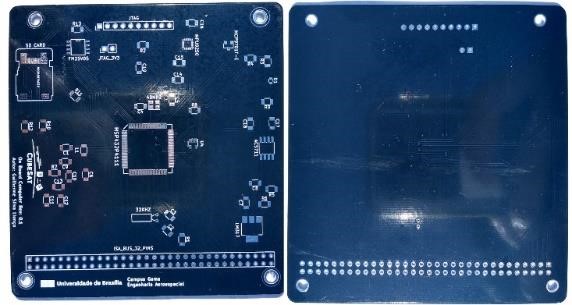
Para testar o Watchdog a nível de software, simulou-se um travamento no ***TT&C Task***, fazendo com que um bit não fosse setado no handler do ***WatchDogTask***. Como era de se esperar, a ***WatchDogTask***conferiu os bits e deletou a ***TT&C Task***, conforme mostrado nas Figuras 10 e 44. Antes de ser delatada, a *task* do TT&C possuía a label **TT&C Task (2)** [azul] e, após ser reiniciada, mudou para **TT&C Task (3)** [verde].



**Figura 10 –** Simulação do travamento do ***TT&C Task***.

### 4.5. Printed Circuit Board

A partir do layout realizado no KiCad, gerou-se os arquivos .gerbers que contêm todas as informações para a fabricação da PCB. Utilizou-se a fabricante PCBWay para realizar a fabricação da PCB. O resultado final é mostrado na Figura 11.



**Figura 11 –** Vista superior e inferior da PCB.

A fabricação da PCB representa a atual etapa em que o projeto se encontra. Os próximos passos, no desenvolvimento do OBC, serão mostrados na seção 7.

## 5. CONCLUSION

Este trabalho apresentou os principais avanços obtidos, até o presente momento, na prototipagem do UNB *On Board Computer,* para a futura missão CubeSat da Universidade de Brasília. O desenvolvimento foi dividido em duas partes: Hardware, onde foi selecionado o microcontrolador e os periféricos; Software, onde foi definido a arquitetura e componentes para cada camada. De acordo com os resultados obtidos, chegou-se à conclusão o MSP432P4111 é uma ótima opção para cenários de desempenho intermediário e de baixa potência. O uso do FreeRTOS como um sistema operacional em tempo real é uma ótima opção para microcontroladores com pouca memória RAM. Também foi visto que o watchdog a nível de software funcionou como forma de redundância em casos de travamento parcial do código. Além disso, o software preliminar já consegue trocar de estado de acordo com algum input, por exemplo a luminosidade.

## 6. FUTURE WORK

Alguns requisitos estabelecidos no projeto ainda não foram atendidos devido à complexidade do projeto e ao tempo limitado disponível. Todos esses pontos serão retomados, desenvolvidos e aprofundados durante os próximos meses para chegar à conclusão deste protótipo de computador de bordo da UnB para o CubeSats. Um ponto de extrema prioridade, a ser retomado, é a compra e soldagem dos componentes, pois não foi possível testar a placa desenvolvida no projeto. Outro ponto que deve ser aprofundado é a utilização do *clock* de 32KHz como fonte de sincronismo do *SysTick*, durante o modo de hibernação do satélite. Foi visto que a utilização de um único *clock*, tanto para o modo de alto desempenho quanto para o modo de hibernação, não torna o OBC robusto em cenários de baixa bateria. O uso de vários níveis de *watchdog* não é suficiente para diminuição do risco dos efeitos da radiação sobre o OBC. A utilização de componentes COTS diminuem a confiabilidade do sistema e outras formas de proteção devem ser analisadas.

## 7. REFERENCES

1. TEXAS INSTRUMENTS, MSP432P411x, MSP432P401x SimpleLinkTM Mixed Signal Microcontrollers. p.214. (2018)
2. F. George, SwissCube HouseKeeping Parameters. p.1. Space Center EPFL, Switzerland (2009)
3. C. Frost and E. Agasid, Small Spacecraft Technology State of the Art. p.97. NASA Ames Research Center, California (2015)
4. ALLEGRO, High Sensitivity, 1 MHz, GMR-Based Current Sensor IC in SpaceSaving, Low Resistance QFN and SOIC-8 Packages. p.1. Manchester, New Hampshire (2018)
5. INVENSENSE, MPU-9250 Product Specification Revision 1.1. p.28. San Jose, California (2018)
6. MICROCHIP, Low-Power Linear Active Thermistor ICs. p.1. Chandler, Arizona (2016)
7. C. Frost and E. Agasid, Small Spacecraft Technology State of the Art. p.95. NASA Ames Research Center, California (2015)
8. TEXAS INSTRUMENTS, MSP432P4111 SimpleLink™ microcontroller LaunchPad™ development kit user's guide (Rev. B). p.11. (2019)
9. PC/104 Embedded Consortium. PC/104 Embedded Consortium. 2nd. p.25. (2008)
10. TEXAS INSTRUMENTS, USER'S GUIDE MSP432® Peripheral Driver Library. p.14. (2015)
11. R. Barry, Mastering the FreeRTOS STM Real Time Kernel: A Hands-On Tutorial Guide. p.17. Real Time Engineers Ltd., USA (2016)
12. B. P. Douglass, UML for the C programming language. p.12. IBM Corporation, USA (2009) [13] P. AB, Tracealyzer for FreeRTOS. p.1. Percepio AB, USA (2019)