Basic Rocket System v1.0 | General Requirements

Rev 01

2025

Lucas Arturo Noce
lucasnoce0@gmail.com
github.com/lucasnoce/LNBRS

Table of Contents

List of Figures

List of Tables

1. Document overview

1.1. Introduction

The Basic Rocket System (BRS) is an open source device designed - as a side personal project - to implement all the most basic functionalities expected from a model rocket avionics computer.

The focus of the project is on firmware development, as hardware is heavily dependent on the rocket structure and design. However, firmware does not exist without hardware, so a block diagram and electrical schematic will be sketched.

The system will be designed to feature some basic functionalities and also to be compatible with future software and, especially, hardware upgrades that can provide further utilities for the system.

1.2. Scope

This document aims to clearly define all the BRS requirements, such as the selected part numbers and firmware structure, and also describe guidelines of how the system should perform and be built.

1.3. Disclaimer

This project is designed by me, Lucas, as a personal project. It is not intended for commercial purposes nor any other use rather than model rocketry hobby activities.

This project is open-source under the MIT license, allowing the community to explore and contribute. However, I disclaim any responsibility for any misuse, damage, or consequences resulting from the use of this project. Do it at your own risk.

Rockets can be seriously dangerous - and potentially deadly - for the ones involved, for others and nearby strangers, so be very cautious, conservative and safe when exploring the rocketry world.

2. Hardware Requirements

2.1. MoSCoW Analysis

The definition of HW specification started by listing all the components related to a standard avionics system. Then, the MoSCoW prioritisation method was implemented in order to enumerate the necessary and desired components. The criteria used to categorize each component was completely arbitrary, but focused on maintaining the lowest possible cost while also allowing the device to have all the key features needed for a basic avionics system.

Table 1 - Hardware Components MoSCoW Analysis

Hardware Component	**MoSCoW**	**Comments**
MCU	Must Have	Focus on low power and/or low cost
IMU	Must Have	Periodically sample acceleration and gyro
Barometer	Must Have	Apogee detection
Power Tree	Must Have	Includes power source, probably LiPo batteries
Recovery System	Must Have	For triggering parachute deployment
Data logging	Must Have	Primarily flash chip, maybe microSD card for cost reduction
Remove Before Flight	Must Have	Must be removed with the rocket fully assembled
LEDs	Must Have	Indicate system state
Basic Backup System	Should Have	May not be designed for cost reduction
Buzzer	Should Have	Indicate system state
GPS	Should Have	May not be used for cost reduction
Wireless Communication	Should Have	Will likely be added later, probably LoRa P2P
Camera	Could Have	Really increases system complexity and storage demand, may be added later as a separate system
Self Ignition System	Could Have	Depends on rocket motor design

Because of the mentioned criteria, some important components were categorized as “Should Have” or even “Could Have” to reduce costs. But, as the project evolves, some of them may be added on later versions of the system.

After the MoSCoW analysis, the components chosen to integrate the project are the following:

- MCU
- IMU

- Barometer
- Power Tree
- Recovery System
- Data logging
- Remove Before Flight
- LEDs
- Buzzer

2.2. Components Specification

Considering that the device will not be mass produced, many components were chosen based on availability, meaning that the focus on low power and/or low cost are often more dependent on retailers stock than on datasheet information.

2.2.1. MCU

For the MCU, the following options were considered. Component information was sourced from the corresponding datasheet. Average price is an approximate value taken from local stores.

Table 2 - MCU Options

Manufacturer	**Part #**	**Flash / RAM (bytes)**	**Max Clock (MHz)**	**Consumption (mA@$freq_{max}$)**	**Average Price**
STMicroelectronics	STM32F103C8	64k / 20k	72	50	R$ 30,00
STMicroelectronics	STM32F103RB	128k / 20k	72	50	R$ 100,00
STMicroelectronics	STM32F401CC	256k / 64k	84	22	R$ 45,00
STMicroelectronics	STM32F411CE	512k / 128k	100	23.6	R$ 55,00
Espressif	ESP32-WROOM-32	4M / 520k	240	80	R$ 40,00

Data driven analysis indicates that the best option (low cost, high density) would be the ESP32. However, previous experiences were also considered and I happen to not be a huge fan of the Espressif platform. Besides, the datasheet shows an average current consumption of 80 mA, which is the highest value amongst the options.

The second best component seems to be the STM32F411CE, which has enough flash memory to store data from one entire flight and the largest RAM compared to the remaining options. That, alongside presenting almost a quarter of

the current consumption than the ESP32, has made this MCU to be the selected one.

Finally, since I don't currently have any STM32F411 at home, I will be starting firmware development with my blue pill kit (which features an STM32F103C8) and then migrating the code to the 411CE for the final version.

2.2.2. IMU

For the IMU, the following options were considered. Component information was sourced from the corresponding datasheet. Average price is an approximate value taken from local stores.

Table 3 - IMU Options

Manufacturer	Part #	DOF	Max Acc ODR (Hz)	Max Acc Range	Supply Voltage	Consumption	Bus	Average Price
NXP	MMA8452	3	800	±8g	2.0~3.6V	<165 µA	I2C	R$ 20,00
InvenSense	MPU-6050	6	1k	±16g	2.4~3.4V	<3.9 mA	SPI / I2C	R$ 30,00
STM	LSM6DS3	6	1.6k	±16g	1.7~3.6V	<900 µA	SPI / I2C	R$ 20,00
Bosch	BNO055	9	100	±16g	2.4~3.6V	<12.3mA	I2C / UART	R$ 300,00

// TODO

2.2.3. Barometer

// TODO

2.2.4. Power Tree

// TODO

- ON/OFF switch accessible with the rocket fully assembled
- Battery
- NO battery charging
- USB type C or microB
- Solder jumper to select between power sources (helps SW dev process)
- Self shutdown feature, controlled by MCU

- Battery may need to be higher voltage, depending on recovery system design
- Probably some separate regulation for the USB supply only
- PWR ON LED

2.2.5. Recovery System

// TODO

2.2.6. Data Logging

// TODO

- Flash chip
- Rough data structure (24 bytes):
    - (4B) Timestamp
    - (6B) Acc [x,y,z]
    - (6B) Gyro [x,y,z]
    - (4B) Pressure
    - (2B) Temperature
    - (2B) Battery Voltage
- Absolute minimum data log flash size: 256 kB
- Variable sample/store rate based on state

2.2.7. Remove Before Flight

// TODO

- Must be accessible from outside, i.e. with the rocket fully assembled
- Must be possible to reinsert the tag with the rocket fully assembled

2.2.8. LEDs and Buzzers

- LEDs:
    - FW predicts a total of 11 states, so 4 LEDs to display current state (for debugging only, some may be removed for flight)
    - At least 2 more LEDs for other indications
    - 1 LED for power input indication (purely HW, may be removed for flight)
    - 1 LED for low battery indication
- Buzzers:
    - Use cases:

    - Signaling RBF
    - Facilitate recovery process (keeps beeping after landing)
    - Low battery indication
- Only 1 buzzer needed

2.2.9. Components Summary

Finally, a summary of all selected components and corresponding specification is presented below. This list is still not a Bill Of Materials, so only the most important components are shown.

Table x - IMU Options

Component	**Specification**	**Average Price**
MCU	STM32F411CE	
IMU		
BAR		
Power Tree	x V battery, USB, ON/OFF switches	
Recovery		
Backup Flash		
RBF		
LEDs	8 LEDs through IO expander	
Buzzer	1 buzzer	

2.3. Block Diagram

Based on the selected components, a high abstraction block diagram was designed to help visualize the circuits.

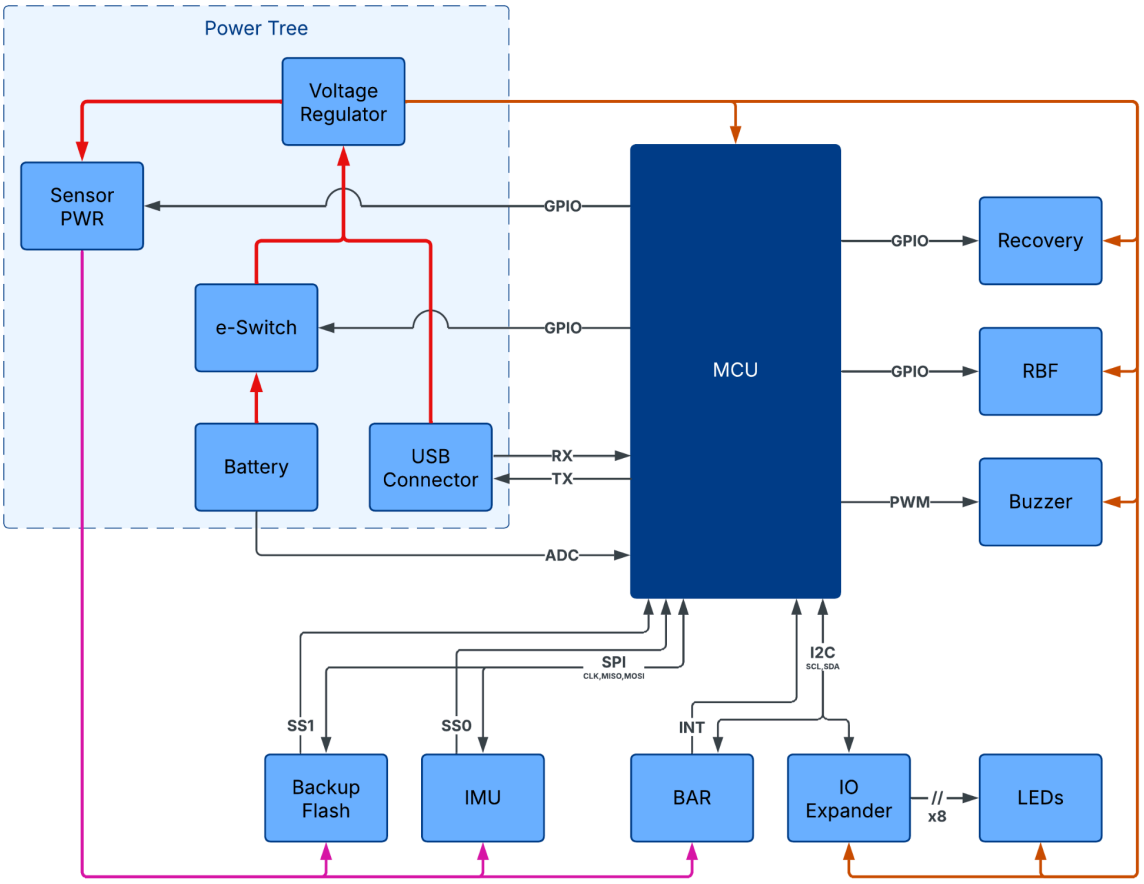

Figure x - Circuit Block Diagram

Signal	**Pins**
SPI	5
I2C	3
Recovery	1
RBF	1
Buzzer (PWM)	1
PWT Ctrl	2
Battery (ADC)	1
USB (UART)	2

2.4. Schematic Design

2.5. PCB Layout

3. Software Requirements

3.1. State Machine

3.1.1. S0

3.1.2. Sx

3.1.3. S1

3.1.4. S2

3.1.5. S3

3.1.6. S4

3.1.7. S5

3.1.8. S6

3.1.9. S7

3.1.10. S8

3.1.11. S9

3.2. Flowchart

3.3. System Architecture

3.3.1. Drivers Description