OA-II VEH COM System Design

DR00003

Rev: A02

Jinzhi Cai

2019–08–15

Contents

1 Introduction

1.1 Scope

This document outlines the design for the OA-II VEG Computing and Operation Module.

1.2 Purpose

The design goal is to build a system that allows real time operation and multimedia processing simultaneously during flight.

2 Revision History

Rev#	Editor	Delta	Date
A01	Jinzhi Cai	Initialize	2019-7-20

Table 1: Summary of Revision History

3 Requirement Analysis

3.1 Failure Recovery

In the OA-OII VEH requirements, failure recovery is the most critical component. Proper recovery and redundancy requires the following:

Backup Each mission critical system must have at least one backup unit.

Fast Switch When an emergency scenario is detected, the system switches to backup unit quickly, and is able to keep that system operating.

Hierarchy Failure Recovery System For each failure scenario, the recovery system has a plan to protect the device and preserve data.

3.2 Realtime Operating System

In the OA-II VEH requirements, a realtime operating system is key to performing critical procedures.[1]

Predictable Execution Time Each operation needs to be completed within a limited time and with minimal errors.

Interface with the Main System The RTOS needs to be able to receive commands from other systems.

Execution Feedback When a procedure is completed, it will need confirmation from sensor data.

4 Current Controller Analysis

4.1 Arduino

Aruino is a common and beginner friendly microcontroller used in DIY electronics, and has abundant libraries for different IC.[2] However, most of these libraries are inefficient and the interface options are limited. Its core clock runs at 16MHz which is not fast enough to support a functional Linux system.

4.2 STM32 Serise

SYM32 Series is another common microcontroller that supports up to 400MHz core clock, and also supports Linux operating systems.[3] However, like the Arduino, it lacks support for a custom interface.

4.3 Market SoC System

SoC systems such as the Raspberry Pi and BeagleBone have a variety of interface options, but whose interfaces lack the high speed required for connections between systems.[4]

4.4 FPGA Soft Core Processor

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence the term "field-programmable".[5] Almost every FPGA company will have it own soft core processor. This allows the core to be easily reprogrammed, as it is formed by "logical synthesis" and allow the user to rebuild them in the FPGA chip as long as the chip has enough resources. Even the soft core processor allow user to customize the structure, but the speed of soft core processor will not as fast as the SoC processor. Although, it is fairly easy to interface to other systems.

4.5 SoC FPGA

The SoC FPGA is another kind of FPGA. The difference is the embedded, more powerful processor know as hard core processor. It helps fuse the benefit between the SoC chip and FPGA chip designs.[6] On one hand, it allows a big system to run in the powerful hard core processor, and, at the same time, the flexibility of an FPGA chip allows a custom interface and upgradibility. However, SoC FPGAs have less logical resources compared with a classic FPGA and weak proessing power when compared with SoC which is the same cost.

5 Recommended System Design

5.1 SoC FPGA

In this design, the main processor will be a SoC FPGA that have about 70K LUT logic resources and a dual core 800MHz - 1GHz hard core processor.

IP core or Logic	Resource Cost
Blackbone Bus Controller	50K(Max)
Low Speed Sensor Controller	10K
Soft Core Processor	9K
Debug JTAG Core	1K
Summery	70K

Table 2: Table of Logic Resource

The hard core processor will be collecting data from the bus and sensor controller. It will store that data and use it to resolve the rocket attitude and location. The result will then transfer to command and be sent to the realtime soft core processor to control the PAM. The FRU will have the same hardware structure as the MCU, but constantly get data from the MCU for error checking.

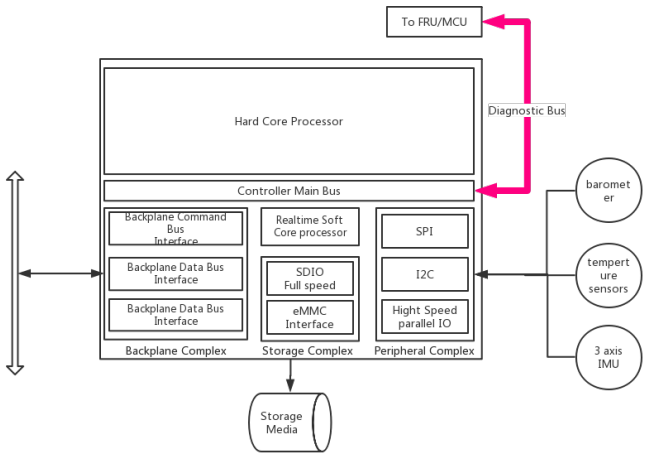

Figure 1: Block Diagram For COM System

6 Bibliography

[1] Wikipedia, *Real-time operating system*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Real-time_operating_system.

[2] Arduino, *Arduino*, 2019. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction.

[3] STMicroelectronics, *STM32 32-bit Arm Cortex MCUs*, 2019. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.

[4] beagleboard, *BeagleBone Black*, 2019. [Online]. Available: http://beagleboard.org/black.

[5] Wikipedia, *Field Programmable Gate Array*. [Online]. Available: https://en.wikipedia.org/wiki/Field-programmable_gate_array.

[6] aliyan, *SoC FPGA*, 2019. [Online]. Available: https://www.eefocus.com/aliyan/blog/18-06/429516_36b28.html.