DOI: 10.13009/EUCASS2019-940

Analysis of scalable distributed on-board computer architecture for suborbital rockets and micro launchers

Jakub Rachucki and Piotr Rugor***
** Łukasiewicz Research Network - Institute of Aviation*
jakub.rachucki@ilot.edu.pl
*** Łukasiewicz Research Network - Institute of Aviation*
piotr.rugor@ilot.edu.pl

Abstract

This paper describes the architecture of the on-board computer (OBC) that meets all of the stated objectives for ILR-33 AMBER suborbital rocket. A modular OBC architecture is presented, based on requirements of scalability and ease of expansion. The merits of independent modules based on COTS components used in the system are described with exemplary system configurations given. Considerations on adapting the architecture to specific needs are presented as well as concepts for managing reliability and testing. Feasibility of such a system for use in microlaunchers and conclusions from the current development stage are discussed.

1. Introduction

The ILR-33 AMBER project aims to develop a highly scalable, cost-effective platform for in-flight validation of microlauncher technologies as well as for suborbital experiments. The OBC team is responsible for providing reconfigurable and reusable avionics that aim to make the AMBER rocket a competitive and reusable solution for scientific and technical research. Special approach during the OBC design needed to be adopted in order to make the avionics reusable for different missions performed by the rocket. The avionics need to be able to act as a service module providing the on-board experiments with power, logging and transmission capabilities while also performing a wide set of functions demanded by rocket flight.

Centralised architectures have proven to be successful while being implemented in suborbital rocket missions [1] as well as CubeSat missions [2]. Perhaps biggest drawback of such solutions is its decreased reusability. Centralised hardware optimised to serve specific purpose might be impossible to scale without significant changes. Thus this architecture was deemed not suitable for the AMBER rocket and alternatives were considered.

On the contrary a distributed system of modular avionics offers the possibility of creation of a scalable, reconfigurable system that can be easily adapted to changing mission purposes [3]. As a result the scalable OBC is able to be used during different suborbital missions and can act as an adaptable service module for suborbital experiments. Such a solution can be easily scaled in order to provide essential functionality and redundancy on-board of a rocket.

2. Requirements.

The presented environmental and functional requirements were used as a guideline in the process of defining the system requirements. Thereafter the system architecture was derived from the system requirements

2.1. Environmental and functional requirements

During a rocket flight on-board instruments are subject to harsh environmental conditions such as extreme accelerations, shocks, vibrations and temperatures [4]. The conditions of such a flight were summarised in Table 1 based on previous flights and experiences of the Institute of Aviation's ILR-33 Amber engineering team. While some

DOI: 10.13009/EUCASS2019-940

of these challenges, such as humidity and temperature, can be resolved at a mechanical level by designing a proper enclosure, many of them require the use of proper design rules to mitigate risk of damage and assure proper resistance to electrical failures. During a suborbital flight the rocket and its avionics is exposed for space environment for very short time. Additionally it reaches relatively low attitudes- below 200 km, thus the cosmic radiation problem is negligible in this case [5].

Table 1: Typical environmental conditions for sounding rocket flight

	Min.	**Max.**	**Unit**
Temperature	-40	80	°C
Pressure	0	110	kPa
Humidity	0	100	%
Vibration (20 – 2000Hz)	-	10	g
Shock (duration 0,5 ms)	-	1000	g
Shock Response Spectrum (Q=10; 100-500Hz)	-	500	g

Aside from resisting environmental effects, space electronics must meet strict criteria defined by the functions it must perform as well as reliability that must be assured. The ILR-33 Amber rocket computer was required to perform a wide set of tasks, from configuration and battery management before launch, through operating under radio silence, to recording numerous parameters and controlling actuators in various modes after launch.
Typically, before launch a flight computer is connected to an auxiliary ground system that provides it with power and communication with a ground station, enabling charging of batteries and limited control and configuration by a user at a ground station. Under such conditions safety must be assured to prevent unplanned events from being triggered. The OBC must then detect the rocket launch and acquire data from numerous sensors while controlling actuators and staging. The system is also required to detect the end of the mission in order to backup flight data and switched to standby mode.

The purpose of ILR-33 rocket is to provide a reusable carrier for various suborbital experiments. The OBC should be designed in a way to provide necessary power, logging and transmission capabilities for the experiments to be mounted on the rocket without introducing big changes in the hardware or software.

2.2. System requirements

In order to address the aforementioned demands, requirements for the computer avionics implementation were adopted. Due to the large number of tasks performed by the computer designing a single highly integrated system would prove complex and difficult to test. Sufficient performance of such a system would also be difficult to assure in early design phases because of large expected communication overhead between the controller of such a system and connectors or actuators. Moreover, a highly integrated system with a controller under high workload would likely be difficult to expand with additional functionalities should the requirements change, necessitating software and hardware changes for every such adaptation.

For these reasons a distributed open system would seem preferable. Distributing the functions of a flight computer across different devices allows to reduce the complexity of each device, making design easier. Moreover, expanding the functionalities of such a system could be achieved by adding more devices to the open distributed network. Therefore assuring demanded configurability and reusability. However, a major problem of such a design would come from size and mass constraints on-board the ILR-33 Amber rocket. Modular architectures used in nanosatellites are heavily constrained by defined connectors and assigned interfaces which highly limit form factors and performance. It has been thus decided that a custom architecture that would enable easy scalability and openness via modularity should be developed. The implementation of such an architecture demands assignment of specialised

DOI: 10.13009/EUCASS2019-940

interface for communication and power distribution. An appropriate target hardware and software platform which contributes to increasing the reusability need to be chosen.

3. System architecture

Achieving the stated requirements for modularity and scalability of the system motivated the definition of an architecture that enables both independent operation of subsystems and future expansion without sacrificing performance. While several such architectures that are based on standards such as PC/104 already exist and are commonly used in nanosatellites [6], the limitations inherent to them such as form factor can invalidate their use in more constrained applications.

3.1 Architecture overview

The developed architecture relies on the separation of subsystems by dividing the on-board computer (OBC) into independent, heterogeneous modules of matching form-factor connected by a common stack interface. The purpose of the interface is to provide data connectivity between modules via multipoint interfaces as well as power distribution and possibility of implementation of inter-modules mission specific functions. Since a predefined form factor limits the possible complexity of each board subsystems can consist of numerous cross-dependant boards, with specific intra-module interfaces to allow for easy expansion. Each module is responsible for a specific task and should be fully self-contained to minimize the cross-dependence of modules.

The architecture is similar to PC/104 in that it has been designed with stacking the boards in mind. However, in contrast to PC/104 systems and similar architectures, no master node is permitted and nodes are as independent as possible, allowing for failures of individual nodes while still maintaining remaining system functionality. As a result adding new nodes should be possible without introducing modifications to existing ones, which could simplify expansion.

Depending on the required performance of inter-module communication two types of interfaces are predicted in this architecture. A multiplexed interface on the common stack interface is always present to enable reliable, low-throughput connectivity with easy integration of additional modules. A common choice for such an interface is the CAN bus [7][8] due to its low latency, high reliability and availability of COTS solutions suited for high-reliability applications. Consequently it has been chosen as part of the basic configuration of a system in this architecture. While CAN can be used as a data bus its low bit rates and strict message format limit its throughput, making it more suited for real-time control than exchanging raw data. To accommodate for high data rates necessity e.g. for communication between data acquisition and memory modules a second interface can be utilized, existing outside of the common interface implemented as part of the module stack. Point-to-point network interfaces are highly preferred for this application, with a separate module operating as a network controller for such an interface. Their comparatively large latency must be considered when deciding which interface should be utilized for a particular transmission, with CAN being ideally used only for control and diagnostics. A summary of characteristics expected of each bus has been made in Table 2.

Table 2: Comparison of bus requirements

	Control bus	**Data bus**
Latency	Low	Uncontrolled
Data rate	Low	High
Implementation complexity	Low	High
Incorporation into modules	Mandatory	Optional
Suggested implementation	CAN	-

High speed interfaces used in today's electrical systems such as require point-to-point low-voltage differential signalling connections due to limitations of arbitration methods used in multiplexing buses. As a result they cannot be incorporated into the stack's common interface without limiting scalability of the system. The proposed architecture requires each board utilizing a high-speed data connection to incorporate a separate wire-to-board connector to interface either with a dedicated network controller module or directly with another module. This allows for a degree of flexibility when designing new modules in that the high-speed bus can be omitted from simpler modules, reducing design complexity.

DOI: 10.13009/EUCASS2019-940

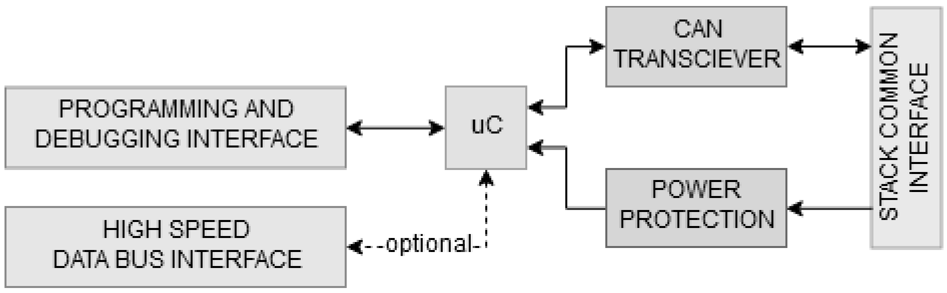
PROGRAMMING AND DEBUGGING INTERFACE
uC
CAN TRANSCIEVER
STACK COMMON INTERFACE
POWER PROTECTION
HIGH SPEED DATA BUS INTERFACE
optional

Figure 1: Basic module building blocks

Concluding the architecture the basic building block diagram of a single module was created and it is presented on the Figure 1. As it can be seen the core of each module is microcontroller with dedicated programming and debugging interface.

Although each module should be as independent as possible, semi-independent modules can be incorporated into designs when necessary due to limitations of the chosen form-factor. In such a case the high-speed bus may be used not only to provide network connectivity with the whole system but also as a way of providing a direct connection between interdependent modules. As the high speed interface is unconstrained from PCB design various different network topologies can be utilized and reconfigurations can be performed to best suit mission requirements. An example of a configuration consisting of individual modules, multi-board modules and different data connections is shown on Figure 2.

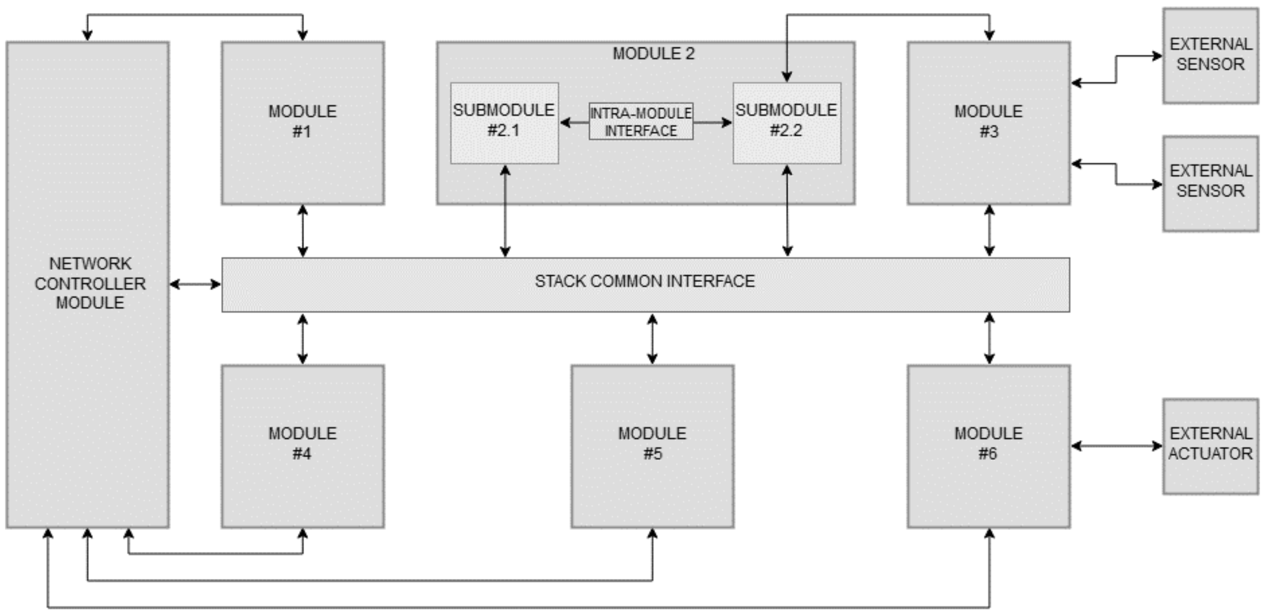

Figure 2: Example system configuration

3.2 Hardware considerations

The architecture is intended for use with COTS hardware components in order to implement self-contained, highly functional modules in a small form factor while also reducing cost. The suggested architecture uses a COTS mezzanine connector to implement an electronic stack similar to PC/104 in a much smaller form factor without sacrificing the capability of the common interface. Current high-density board to board connectors offer much lower pitch than the 2.54mm used in PC/104 while also allowing multiple board-to-board heights to be incorporated into a single design, which helps reduce board size and stack height, subsequently reducing enclosure size and weight.

Most connectors are also suited for high-speed interfaces, which allows more expansion possibilities for future designs.
Although not necessitated by the architecture the current implementation is based on ARM microcontrollers with each module having the exact same microcontroller, which allows for reusability of hardware and software between modules. ARM microcontrollers additionally offer an improvement in cost, performance and power consumption over CPUs currently used in the space industry [9] and substantial effort is being made in order to adapt them to high-reliability applications [10]. Furthermore, the proposed architecture allows to compensate for the reduced reliability of COTS components by easy implementation of cold-redundancy, as discussed further.
Due to heavy emphasis on openness the modules share common power supply lines. However, due to harsh environmental conditions during rocket flight, power failures due to phenomena such as capacitor cracking might cause power shorts which could cause a system failure if not protected against. Overcurrent protection has been incorporated into each module limiting the current to twice the maximum nominal operating value to switch a faulty module off from the power supply. This safety feature can be coupled with various forms of redundancy to detect failures and adapt the operation of the system to assure mission completion.

3.3 Software considerations

Due to the modular approach taken and incorporation of highly repetitive building blocks of the modules much of the code driving each module's microcontroller can be reused. For this reason the structure of the developed code has also been developed with modularity in mind, utilizing numerous layers of abstraction to account for future changes in hardware. The aim of the applied software design philosophy was to reduce repetitive workload while maintaining high code clarity and performance.
C++ has been chosen as best for this purpose due to its good performance and heavy support of object-oriented programming (OOP). The use of OOP allows for clean and efficient implementation of numerous abstraction layers as well as creation of easy to use drivers enabling easy and safe code reuse with the use of encapsulation. The created software architecture can be seen on Figure 3.

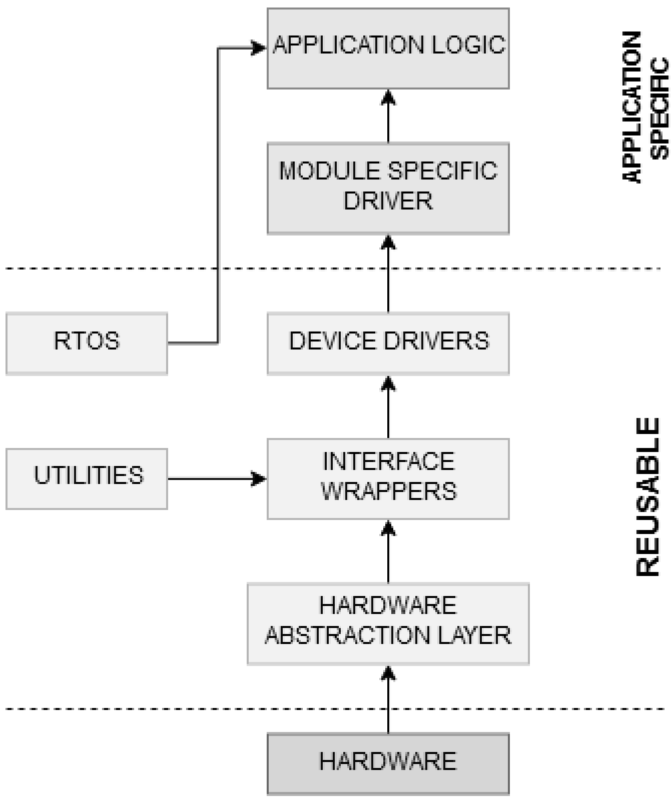

Figure 3: Software architecture summary

DOI: 10.13009/EUCASS2019-940

Although a bare-metal approach has been tested and deemed feasible for achieving desired performance and software modularity, the application of real time operating systems such as FreeRTOS has allowed for further improvements as several functions are performed uniformly across modules.

Since tasks relating to the basic building blocks are performed on each module a basic code portion is present across all nodes and can be, similarly as in hardware, considered a basic building block, the structure of which is shown on Figure 4. In the proposed implementation the basic tasks would be:

- CAN TX, responsible for sending messages from a queue updated from other threads
- CAN RX, responsible for receiving and parsing CAN messages
- LOG UART TX, responsible for sending debug information via a serial interface
- DEFAULT, which is the main logic state implementing the state machine

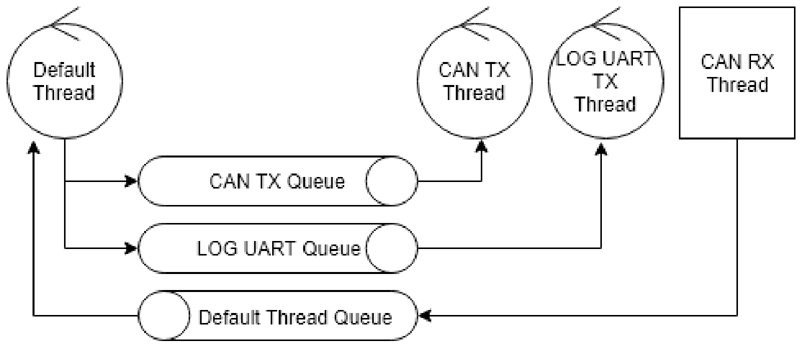

Figure 4: Example basic thread diagram for proposed configuration

Thus a new module can be created with a greatly reduced workload by expanding on a ready and tested framework utilized on previous modules.

3.4 Redundancy considerations

Reliability is a major concern for space avionics due to harsh operating environment causing difficult to predict failures, most often associated with space radiation. Due to the use of COTS components reliability of a single module is likely to be inferior to current space systems utilizing specialized components with space heritage. The proposed architecture has been created to reduce the impact of single module failures on the rest of the system by focusing on the independency of each module. However, reliability can be further improved by adding redundancy at the module level or system level.
Cold- or hot-spare redundancy in the form of multiple identical modules performing the same function can be implemented with little workload due to the independent operation of modules. The CAN bus utilized in the system can be used for heartbeat messages that would allow for use of cold redundancy or to implement voting in the case of critical command messages. The proposed architecture thus provides a flexible method of applying redundancy according to mission risk and required reliability.
Redundancy can be also applied in the form of multiple instances of the same system operating independently, with either cold-spare or voting implemented as a separate system. Such systems have been already described in aircraft avionics [11].

DOI: 10.13009/EUCASS2019-940

4. Testing

For the purpose of increasing the reliability of the system a proper testing scheme has to be applied. Required tests can be divided into 2 groups: hardware tests and software and integration tests. Hardware needs to be qualified before it will be integrated with the software. Qualification of the hardware should be conducted using a dedicated test stand. To reduce development effort test stands should not be dedicated for each module but instead able to service all modules designed according to this architecture. Continuous integration practices should be applied during the hardware and software integration as it guarantees rapid bug discovery and produces tested and reliable code. The testing procedure should enable testing of independent isolated modules as well as a whole stack.

4.1. Hardware tests

Hardware tests should be conducted as in-circuit tests using a test stand able to test different kinds of OBC modules. For the purpose of reducing the test overhead the construction of the test stand cannot require any significant modification of the tested modules. Consequently the bed of nail test approach was adopted for this purpose. This solution provides a compact interface between the test stand and the device under test (DUT). Such a solution has a low impact on the hardware design of a module. During PCB design only small testing points for signals, power lines and interfaces have to be included which results in a minimally invasive testing interface on each module.

Hardware in the loop testing, simulation of digital and analog inputs and outputs as well as simulation of different protocols and resistive loads are required in order to conduct comprehensive tests of every module. National Instruments PXI computer was chosen as a very convenient solution with intuitive software, LabVIEW, available for easy creation of custom testing user interfaces.

The heterogeneity of each module introduces some challenges to the process of standardisation of the test procedure. In order to reduce workload it is necessary to provide a common interface which operates as a bridge between the PXI and DUT. Thus an adapter PCB was designed. Its purpose is to gather all of the signals of the PXI and all of the loads on one common connector. Loads can be control by PXI using digital outputs that controls relays. The relays are responsible for connecting power resistors in parallel. The more power resistors are connected the bigger the load is, which allows for varying the load during testing. It also increases the amount of digital and analog channels that can be measured or simulated by providing channel multiplexing. Owing to this solution the testing procedure can be unified. Testing overhead comes down to designing a custom PCB interfacing with pogo pins dedicated for every DUT. The PCB is connected to the adapter PCB using previously mentioned common connector. The whole architecture of the test stand is sum up on the Figure 5:. Testing PCB provides the possibility to test a module as completely isolated system. Every interface and signal used by the module can be simulated by connecting it to common connector.

With aim of simplifying and automating the testing procedure, a LabVIEW application provides possibility of reading and writing every test stand I/O. It is also possible to program the test sequence and save the test results. What’s more the test result of a specific module can be employed as a qualification model for the production of this module.

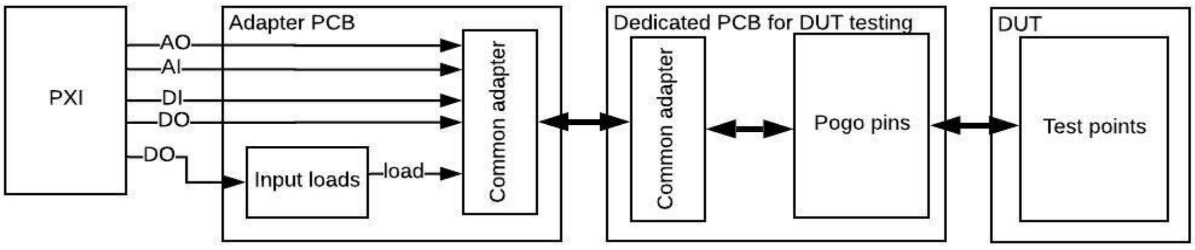

Figure 5: Diagram of the test stand architecture

4.2. Software and integration tests

The modular and reusable character of the OBC software makes it possible to develop each chunk of the code independently from the rest of the software. Eventually the part of the code that was under development need to be integrated with the rest of the code and further the new version of the software needs to be integrated with the

DOI: 10.13009/EUCASS2019-940

hardware. Usually integration is risky and demanding process [12]. The practise of continuous integration (CI) has proven to be very effective addressing that issue. CI means that software developers integrate the code often, at least once per day [13]. Every integration includes running the test and software verification. This practice contributes to early discovery of integration problems.

In order to habituate this practise the workflow described in Figure 6:. was introduced. First the developer needs to cover the majority of his code with unit tests. Unit testing is one of the practises of test driven development approach (TDD). This technique enables to test the software incrementally function by function. This approach leads to significant decrease in development time [12] as it helps to find bugs quickly which also contributes to decrease of debugging time. Adaptation of TDD in embedded systems helps to write and test the code independently from hardware which reduces the debugging time on target where software issues might be harder to find. In order to run unit test each developer need to set up a unit test harness [12]. Only when all of the unit test were successfully passed the code can be committed. Commit is detected by an automatic build server. The server tries to integrate it by running the integration unit tests and then building it.

To prevent from project delay due to late hardware development it must be possible to test the software completely on the emulated target. Even if hardware is already qualified it is require to first test it on emulated hardware as this may lead to early detection of the bugs that might cause hardware issues. Software can be uploaded to dedicated qualified module only after passing all of the test. If the hardware is not qualified yet but the software passed the tests on emulated environment it is deployed to the repository and waits for hardware to be qualified. The last step is a hardware in the loop test conducted on the isolated module. All of the interaction with the external devices and components are simulated by the test stand, which also monitors and enables direct interaction with the DUT e.g. changing state or simulating errors. If software pass all of the test in can be deployed into production repo.

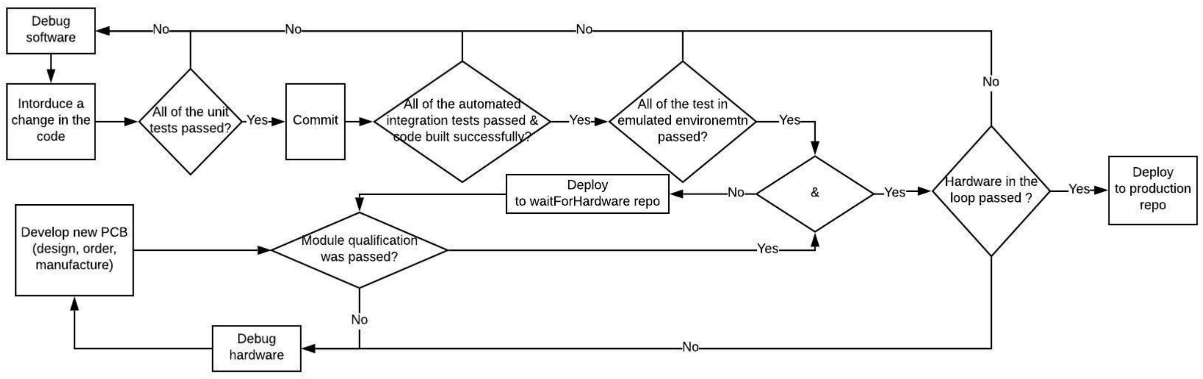
Debug software
Intorduce a change in the code
All of the unit tests passed?
Yes
No
Commit
All of the automated integration tests passed & code built successfully?
All of the test in emulated environemtn passed?
&
Deploy to waitForHardware repo
Hardware in the loop passed ?
Deploy to production repo
Develop new PCB (design, order, manufacture)
Module qualification was passed?
Debug hardware

Figure 6: Software and integration test workflow

The test stand gives the possibility of conducting whole stack tests. In the stack configuration it is not possible to access all of the modules using pogo pins. Therefore a flat configuration must be adopted providing the possibility of simulating the external devices and different testing scenarios using the test stand LabVIEW application. In such a configuration a dedicated PCB provides the common interface using interconnected mezzanine connectors.

5. Discussion of results

Providing the basic hardware and software tools, like configuration of CI process and building of the reusable test stand the team could begin to establish reusable hardware and software building blocks. The basic template of the module PCB, including the microcontroller, mezzanine connector and CAN module, was established and utilized during further development of the first three prototype modules. A similar approach was undertaken during the software development. Reusable software components like drivers, wrappers, utilities as well as classes for task creation and management were created. This caused the software development process time to be shortened as only module specific functionality had to be developed while the rest functionalities were reused.

Finally after establishing of this framework the reliable system of delivering the ready to flight OBC was created. Due to the high modularity and independence of modules as well as hardware-independent software testing patterns application of agile project management methodologies was attempted. The qualified hardware integrated in dedicated housing with tested software acted as adopted definition of done (DOD) of each sprint. In this way there is always qualified OBC available for flight test. New hardware and software changes are introduced on the development version of the system. After qualification they are being integrated with the latest version of the system creating new flight ready iteration. The developed architecture and design method could allow for further implementation of agile methodologies to better adapt to changing requirements posed by experiment owners.

So far power, logging and data acquisition modules were developed and are ready for flight test. They create the first iteration of the system. The development process of first iteration successfully verified the presented architecture and provide the OBC team with conclusion and plans for future improvements.

Adaptation of the architecture to microlaunchers, satellites or other more complex high-reliability systems should be more closely analysed in a separate case study with a derivation of a more precise set of functional requirements and solutions enabled by the presented architecture. Although features useful in such systems such as redundancy and high scalability should be possible in this architecture. They have not yet been tested and remain to be considered in the future.

6. Conclusion

The process of the ILR-33 Amber OBC development according to the presented architecture and methodologies is still ongoing, however the adopted strategy has already proved to be effective during the design of a prototype. A modular architecture with highly independent modules has been developed and its advantages in high-performance, scalable designs have been assessed. Basic module structure both in terms of hardware and software has been defined which allowed for reduction of workload when designing new modules. Methodologies for hardware and software development and testing were presented.

The architecture has already been used to develop a working 3-module prototype used for data acquisition and recording on the ILR-33 Amber rocket. Though true effectiveness of the architecture remains to be seen as new needs for functionalities arise it has thus far been useful in increasing design reuse, thus reducing development time.

References

[1] Schüttauf, K. (2018). REXUS User Manual. Retrieved from http://rexusbexus.net/wp-content/uploads/2018/11/RX_UserManual_v7-16_24Oct18.pdf

[2] Nielsen, J. D. & Larsen, J. A. (2011). A Decentralized Design Philosophy for Satellites. 5th International Conference on Recent Advances in Space Technologies

[3] Tagawa, G. B., & Souza, M. L. (2011). An Overview of the Integrated Modular Avionics (IMA) Concept. DINCON

[4] Szpakowska- Peas, E., & Mariusz, K. (2016). Selected problems of electronic equipment design in rocketry. Transactions of the Institute of Aviation, pp. 209- 217

[5] Martines M. L. (2012). Analysis of LEO Radiation Environment and its effects on Spacecraft's Critical Electronic Devices. Dissertations and Theses. 102. Embry-Riddle Aeronautical University

[6] Bouwmeester, J., Langer, M., & Gill, E. (2017, 06). Survey on the implementation and reliability of CubeSat electrical bus interfaces. CEAS Space Journal

[7] Elston, J., Argrow, B., & Frew, E. (2005). A Distributed Avionics Package for Small UAVs. Infotech@Aerospace.

[8] Boniol, F., & Wiels, V. (2014, 12). Towards Modular and Certified Avionics for UAV. Aerospace Lab.

[9] Poupat, J. L., Leroy, B., & T., H. (2016). TCLS ARM for Space. Proc. of the Data Systems In Aerospace Conference.

[10] Iturbe, X., Venu, B., Ozer, E., & Das, S. (2016). A Triple Core Lock-Step (TCLS) ARM® Cortex®-R5 Processor for Safety-Critical and Ultra-Reliable Applications. International Conference on Dependable Systems and Networks Workshops.

[11] Yeh, Y. (1998). Design considerations in Boeing 777 fly-by-wire computer. Proceeding of Third IEEE International High-Assurance Systems Engineering Symposium

DOI: 10.13009/EUCASS2019-940

[12] .Grenning, W. J. (2011). Test-Driven Development for Embedded C. Dallas, Texas: The Pragmatic BookshelfUsually integration is risky and demanding process

[13] Fowler, M. (2006). Continuous Integration. Retrieved from https://martinfowler.com/articles/continuousIntegration.html.