Experiences in Firmware Development for a CubeSat Instrument Payload

Joseph Mangan[1], David Murphy[2], Rachel Dunwoody[2], Maeve Doyle[2], Alexey Ulyanov[2], Mike Hibbett[3], Sai Krishna Reddy Akarapu[2], Jessica Erkal[2], Gabriel Finneran[2], Fergal Marshall[2], Jack Reilly[2], Lána Salmon[2], Eoghan Somers[4], Joseph Thompson[4], Sarah Walsh[2], Lorraine Hanlon[2], David McKeown[4], William O'Connor[4], Brian Shortt[5], Ronan Wall[2] and Sheila McBreen[2]

Abstract

Recent advancements in gamma-ray detector technology have brought new opportunities to study gamma-ray bursts and other high-energy phenomena. However, there is a lack of dissemination on the development methods, tools and techniques used in the production of instrument flight firmware. This is understandable as firmware for spacecraft payloads may be proprietary or exceptionally hardware specific and so is not always published. However, this leaves a gap in the knowledge for CubeSat teams, especially those consisting of university students who may be building a custom spacecraft payload with limited initial experience. The Gamma-Ray Module (GMOD) on-board EIRSAT-1, a 2U CubeSat in the 2nd European Space Agency Fly Your Satellite! programme, is one such instrument. GMOD features a 25x25x40mm Scionix CeBr3 scintillator, coupled to an array of 16 (4x4) JSeries OnSemiconductor MicroFJ-60035-TSV silicon photomultipliers (SiPMs) with readout provided by the SIPHRA IDE3380 application specific integrated circuit. The instrument is supported by the Gamma-Ray Module motherboard which controls and configures the instrument, providing regulated voltage and current sources as well as generating time tagged event packets and a temporary on-board flash storage. At the core of this system is the Texas Instruments MSP430FR5994 microcontroller. A custom firmware was produced for the instrument by the EIRSAT-1 team over numerous cycles of testing and development to reliably perform the long duration tasks of readout, storage and transfer of time tagged event data to the EIRSAT-1 on-board computer. Recognising the value of sharing our experiences and pitfalls on firmware development with the wider CubeSat community, this paper will provide an introduction to GMOD, with focus primarily on the development approach of the firmware. The development, testing, version control, essential tools and an overview of how the resources provided by the device manufacturer were used will be examined, such that the lessons learned may be extended to other payloads from student-led missions.

Keywords

EIRSAT-1, GMOD, FYS!, Gamma-Ray Detector, Firmware, Software, MSP430

[1] Corresponding author: University College Dublin, Ireland, joseph.mangan@ucdconnect.ie
[2] School of Physics, University College Dublin, Ireland
[3] Irish Manufacturing Research, Ireland
[4] School of Mechanical and Materials Engineering, University College Dublin, Ireland
[5] ESA European Space Research and Technology Centre ESTEC, Netherlands

1. Introduction

The simultaneous detection of the gravitational wave (GW) GW170817 [1], and its coincident electromagnetic counterpart GRB170817A [2] has resulted in a new era of gravitational and gamma-ray burst (GRB) astronomy. However, this comes at a time when the main GRB detecting flagship missions are operating close to and beyond their intended operational lifetime [3]. One method for enhanced capabilities is the use of CubeSats, which can be developed for relatively low cost, on short timescales, with reduced risk and technology acceptance levels. This provides opportunities for industry and university teams to develop and test hardware, to progress the maturation of novel technology and contribute to the scientific community.

The Gamma-Ray Module (GMOD) is a <1U, gamma-ray detecting instrument which has been developed based on a heritage of tests (e.g. [4] and references therein). GMOD is the primary science payload on the 2U CubeSat EIRSAT-1 (Educational Irish Research Satellite) [5], a participant of the 2nd European Space Agency (ESA) Fly Your Satellite! (FYS!) programme, consisting of three scientific payloads, custom software and off-the-shelf hardware.

From our experience, there is a lack of dissemination of firmware development methods for payloads, on large missions and CubeSats. This is understandable as there may be legal issues surrounding publication of software for proprietary hardware, or simply that the firmware solutions are seen as too specific to the mission. However, this leads to a gap in the literature and a paucity of references on common techniques when it comes to firmware development and testing. To encourage further sharing of knowledge in this area we present an overview of GMOD (§2), our firmware development approach (§3), the tools used (§4), as well as an overview of some pitfalls and lessons learned (§5) throughout its development.

2. The Gamma-Ray Module

GMOD [6] consists of a 25x25x40mm Scionix CeBr3 scintillator coupled to a 4x4 tiled array of 16 OnSemiconductor MicroFJ-60035 silicon photomultipliers (SiPMs) readout using the SIPHRA application specific integrated circuit (ASIC). GMOD is a novel instrument primarily developed to study GRB events in the GW era while also demonstrating the use of SiPMs in earth orbit and their role in future missions.

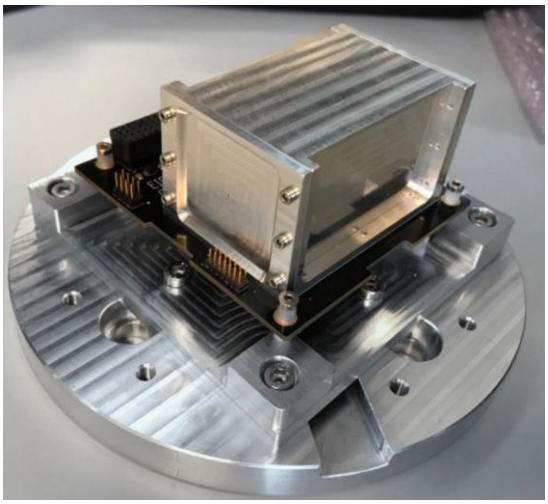

Figure 1. GMOD, the Gamma-Ray Module, the primary experiment payload on EIRSAT-1 mounted on an aluminium test fixture (lower)

The GMOD motherboard is a readout and control system responsible for carrying out the operation-critical duties of the instrument. This is accomplished using a custom firmware [7] developed in C/C++ for the Texas Instruments (TI) MSP430FR5994 16MHz microcontroller. The firmware requirements include:

- Readout of the detector assembly.
- Generation of time tagged event (TTE) data from the detector readout.
- Processing and temporary on-board storage of TTEs in 128MBit flash.
- Constant current supply (ASIC) and bias voltage generation (SiPMs).
- Transmission of TTE data to the EIRSAT-1 on-board computer (OBC).
- Configuration of the instrument.

3. Firmware Development Approach

This development cycle, as demonstrated in Figure 2, begins with an assessment of the requirements followed by implementation and testing on a subsystem level. If satisfactory, testing is then done with the OBC in the loop. If either test fails, the firmware is revised and retested. This approach was selected as it provides a structured approach to development and testing from the perspective of the OBC, payload and the interface between them.

3.1. Firmware Requirements

It is necessary to order the production of firmware in a structured fashion. Typically instrument firmware would be framed around the requirements set out in the preliminary design review (PDR) and critical design review (CDR) products. These are developed as part of phases B/C of the typical project lifecycle and encompass the design specification of the

spacecraft and its payloads. For ESA FYS! the project structure closely adheres to the ECSS standards set out in ECSS-M-ST-10C as the “Design Your Satellite” phase, which include producing the design definition file (DDF) and design justification file (DJF) documents. These documents form the bedrock of firmware development for the payload and are the fundamental starting point and reference when developing any new functionality.

3.2. Outputs From Testing

Throughout the design and qualification stage of the life cycle, the spacecraft and payloads undergo numerous tests. Environmental testing may be conducted on the subsystem [4] and system level [8], which may also require pre-campaign functional and mission tests [9]. All of these tests can be used to inform additions or amendments to the existing functionality, providing feedback not found within the scope of the basic payload firmware requirements. Furthermore, these tests should be used as key milestones along the path of development. Certain functionality will be required at these milestones (ie. pre/post vibration functional tests), thus the build up to these tests can be used to assign priority and pacing to certain blocks of functionality. It is important to ensure that priority is maintained and the appropriate time is allocated where needed. As in Figure 2, requirements, documentation and testing all feed into development of new functionality, with some being introduced as outputs from test campaigns throughout the project life cycle.

3.3. Introducing Functionality

Payload requirements are initially distilled into individual deliverables during a review of the documentation or through team meetings post testing. In some cases, these deliverables may be so low level that they do not appear in the design documents. Interface documents (between the OBC and payload) can then be produced which outline the operations and form of communications. For instance, an interface document may describe the command structure to activate certain functionality on the payload. In more intricate situations, an interface document may explicitly outline how this functionality is expected to behave if that operation impacts the interface.

For example, on GMOD, a “Function List” exists with all commands, descriptions of their operation, the size of data expected to be received and transmitted and any fail scenarios listed.

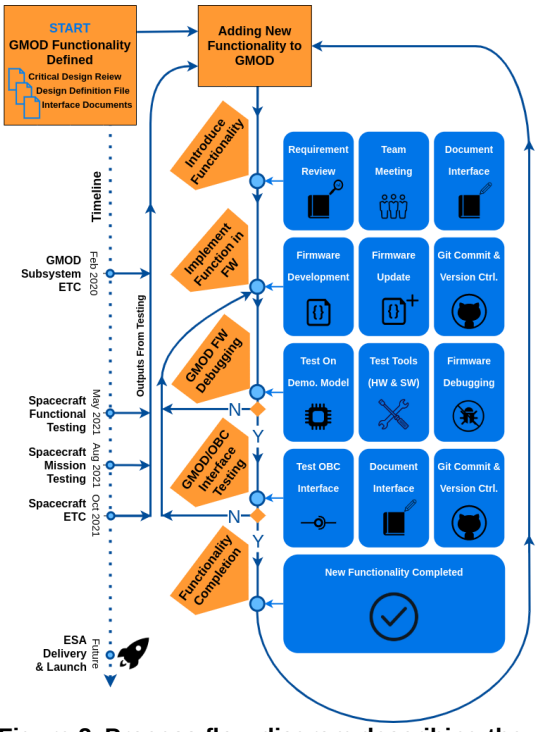

Figure 2. Process flow diagram describing the addition and development of new functionality

Similarly, in the case of complex operations like the serving of full channel TTE data from GMOD to the OBC, a more detailed overview of how the functionality should be implemented was produced. We have found in our experience that it is sufficient to produce these documents as required, without firm document control, but that the final iteration would be appropriately documented for OBC interfacing.

3.4. Implementing Functionality

This functionality is then implemented in C/C++ using the *Code Composer Studio* (CCS)[1] integrated development environment (IDE) provided by TI. An important part of this development is version control. This is performed using the Git version control tool. Upon completion of the new functionality, committing and pushing to the Git remote, the firmware is then tested.

3.5. Firmware Debugging

Before testing new functionality with the OBC, it is tested and debugged separately on a test setup (as shown in Figure 3) replicating the OBC interface to confirm all functional requirements are met. This is done using ground support equipment (GSE), some having been developed and used during the EIRSAT-1 environmental test campaigns (ETCs). Additionally, it is possible to drill

[1] https://www.ti.com/tool/CCSTUDIO

further into the firmware operation mid-execution using the CCS IDE debugging capabilities, which offer numerous breakpoint, timing and variable reporting tools. If a bug is found or the functionality does not behave as expected the firmware is revised (from §3.3) with edits to the firmware code, as seen in Figure 2.

When testing is complete, a binary file is generated after compilation, which is renamed with the version ID and timestamp of creation. The version ID code records *major, minor* and *patch* updates in a two byte ID as in Table 1:

Table 1. The version ID convention adopted for the GMOD firmware development

Versions	**Major** 4 bits (0-15)	**Minor** 8 bits (0-255)	**Patch** 4 bits (0-15)
DM	0	-	-
EQM	0x1	0x00 - 0xFF	0x0 - 0xF
FM	0x2	0x00 - 0xFF	0x0 - 0xF

The version ID is also hardcoded into the firmware and is accessible over I2C. A *major* increment indicates the firmware model. A *minor* increment may be any change judged to be larger than a patch. A *patch* update could be something as small as a bug fix or comment correction (a rollover of the patch increment from 0xF to 0x0 implies a minor increment has occurred). For example, V1.2.3 is encoded as two bytes as 0x1023. When completed the firmware is tagged on Git using its version ID for later reference.

3.6. GMOD/OBC Interface Testing

Once there is confidence in the newly developed firmware the interface between the OBC and GMOD is tested. The binary file is reformatted into serial bootloader commands used by the OBC to program GMOD. The OBC can then command and control GMOD using the *Mission Control Software* (MCS), provided by *Bright Ascension*. Throughout these tests the general operation of the new functionality can be confirmed for both OBC and GMOD while any deviations or changes to the interface can be discussed in a post test context. Any deviations from the original interface can be agreed and amended by revision of functionality (from §3.3). Once satisfied, the new functionality is considered complete and the cycle begins anew, as in Figure 2. For instance, development of GMOD's demonstration model firmware was produced in a closed loop without input from the OBC software team. This was acceptable for this iteration, as it was intended to simply demonstrate the instrument operation as a standalone system. However future iterations have strict interfacing between the OBC and GMOD and require close collaboration with the OBC software team. The EQM version was redeveloped from the ground up using this flow.

4. Development Tools and Resources

It's not possible to describe a universal configuration for testing firmware applicable to all. However, many of the tools used with GMOD are generic, off the shelf and for the most part relatively accessible. Similarly, while other payloads use different hardware and may be locked into a given device manufacturer's documentation, compilers and drivers, many reputable manufacturers share the same level of high quality in these areas. For this reason, we describe the development and test tools used, such that other teams might find similar services provided by their chosen device manufacturer.

4.1. MSP-EXP430FR5994 Launchpad

Development kits are intended to provide a starting point for unfamiliar audiences. These kits allow the users to develop skills when working with embedded devices as well as in understanding the fundamentals of the device and its peripherals. For instance, TI produces the *MSP-EXP430FR5994 LaunchPad,* with similar kits for their range of products. These are invaluable tools to begin firmware development by exploring the limits and potential quirks of the chosen device while also allowing the user to evaluate its suitability, potential design ideas and capabilities before incorporating them into their application. By far its most useful application is its ability to act as a programmer for any MSP430 device with accessible SBW pins. The LaunchPad PCB consists of two blocks: the MSP430 target and the eZ-FET debug probe. The eZ-FET, which is connected to the target device via jumper pins, can be broken out from the LaunchPad and connected to any MSP430FR5994 device. This provides access to TI's debugging tools in CCS, TI's *EnergyTrace* capabilities (for live power management analysis) and a low-cost programmer, which as compared to a generic flashing device would normally be priced up to €200 per unit. The LaunchPad can be seen in Figure 3.

Figure 3. The hardware setup used to test GMOD firmware, including the motherboard, interface board, Total Phase Aardvark, MSP-EXP430FR5994 LaunchPad and FTDI LC234X UART module

4.2. TI's Code Composer Studio

Code Composer Studio is an Eclipse based IDE which supports TI's MSP430 devices. As such, CCS allows the creation of project profiles which can be integrated with Git, a source code editor, compilation and even register and memory browsing on active debugging code along with all the other general debugging operations. Furthermore, given that CCS is built upon the Eclipse open framework, there is no licence fee to access or develop using CCS, a quality which is especially advantageous for university teams.

4.3. TI's MSP430 Driverlib

Depending on the application, a choice can be made on how the firmware developer interacts with the device peripherals in C/C++. For the MSP430 there is the option for direct register level interaction (the developer directly sets/resets bits in the device/peripheral control registers) or indirect interaction where the developer may use a set of abstracted libraries developed by TI called *Driverlib*. This is very useful for several reasons as it allows almost immediate access to the device peripherals without in-depth knowledge of the MSP430, but also may be used as a reference when looking for an example implementation.

4.4. TI Documentation and Examples

Developing for hardware of course requires reference to device documentation. This is a must for successful development of any application. TI have compiled a comprehensive user manual and data sheet for the MSP430FR5994 while also providing application reports describing best practice and the use of the device peripherals. TI provides a number of example scripts in the TI *Resource Explorer*, describing in detail how to operate the device and the internal peripherals for both register level and Driverlib applications.

4.5. Ground Support Equipment

Testing of firmware requires simulation of the intended interface between the payload and the spacecraft. A GMOD interface board was built to break out the PC/104 header for the ETC. GMODs primary channels of communication are through I2C (command and control) and 128k baud asynchronous serial (science data). To allow control of GMOD in the absence of the OBC, a *Total Phase Aardvark I2C/SPI Adapter* is used to interface with GMOD using Python scripts. Readout of science data over serial can be achieved using any generic USB to serial interface, in the case of GMOD an *FTDI LC234X* UART module was selected. To emulate the role of the OBC and MCS during testing, a Python Jupyter Notebook was developed which provides access to the GMOD command set. Both the Total Phase Aardvark and FTDI LC234X modules can be seen in Figure 3.

5. Discussion and Lessons Learned

5.1. Structure is Essential

As mentioned in §3.1, firmware development needs structure, primarily based around the requirements of the payload, but also in terms of the scheduling of deliverables and priority assigned to functional blocks as described in §3.2. This structure is informed by the existing documentation from mission planning, testing and design.

5.2. Know When Enough is Enough

In §3.2, a reference is made to time allocation regarding firmware development. There are

diminishing returns when fixating on the implementation of functionality, particularly when it comes to the impact on schedules. Typically if firmware works as expected, satisfies the requirement criteria and has been tested - little can be gained by further modification.

5.3. Balancing Schedules

As a student team consisting mostly of PhD and masters students, working on a demanding project such as CubeSat development means maintaining a balance between academic work and work related to EIRSAT-1. This is not at all straightforward and the amount of time required to be dedicated to the project should not be underestimated.

5.4. Testing Firmware

There is no “one size fits all” when it comes to firmware/hardware testing. However, from our experience, we have found it is important during testing to introduce a level of “randomness” as expected during actual operation, to better stress the firmware. For one example, externally triggering GMOD with periodic pulses does not test the robustness of the firmware in the same way as triggering randomly in time, which better simulates realistic detections and strains detector readout, access to flash and transmission of data to the OBC.

Another aspect is unit testing firmware. Typically firmware developed for embedded systems is not usually unit tested. This is often due to the belief that it is impossible or impractical due to hardware specifics, or simply because it is acceptable on embedded systems for there to be no defined boundary between pure software and hardware/register manipulation. However, unit testing during development on embedded systems is possible. Test driven development (TDD) has a number of benefits, including reduced time spent debugging, confidence in the final product, continued confidence after modification of the firmware and well-structured code, which stands as objectively reliable. While unit test development is underway with the GMOD firmware, it would have been advisable, and saved time, to have begun test driven development from the start.

6. Conclusion

This paper presents the payload firmware development approach of a student-led CubeSat team. The importance of a structured development cycle, availing of manufacturer resources, balancing PhD and project related work as well as adequate stress testing of the firmware have all been highlighted. In conclusion, there is no single way to develop payload firmware, however it is hoped this may offer some advice and a starting point to other teams to begin their own payload firmware development.

7. Acknowledgements

EIRSAT-1 is carried out with the support of ESA’s Education Office under the Fly Your Satellite! 2 programme. This study was supported by The ESA’s Science Programme under contract 4000104771/11/NL/CBi. JM and SMB acknowledge support from Science Foundation Ireland (SFI) under grant number 17/CDA/4723. We acknowledge all students who have contributed to EIRSAT-1.

References

[1] Abbott, et al., “Gravitational Waves and Gamma-Rays from a Binary Neutron Star Merger: GW170817 and GRB 170817A”, ApJL, 848 L13, 2017

[2] Goldstein, et al., “An Ordinary Short Gamma-Ray Burst with Extraordinary Implications: Fermi-GBM Detection of GRB 170817A”, ApJL, 848 L14, 2017

[3] Perkins et al., “Burstcube A Cubesat for Gravitational Wave Counterparts”, PoS ICRC2017, 760 (2017).

[4] Mangan et al., “The Environmental Test Campaign of GMOD, a Novel Gamma-Ray Detector,” Proc. SPIE 11852 ICSO, 2021

[5] Doyle et al., “Update on the Status of the Educational Irish Research Satellite (EIRSAT-1)”, Proc. 4th SSEA, 2022

[6] Murphy et al., “A compact instrument for gamma-ray burst detection on a CubeSat platform I”, EA, 52 59–84, 2021

[7] Mangan et al., "Embedded Firmware Development for a Novel CubeSat Gamma-Ray Detector," IEEE (SMC-IT), 14-22, 2021

[8] Dunwoody et al., “Thermal Vacuum Test Campaign of the EIRSAT-1 Engineering Qualification Model”, Aerospace 2022, 9, 99. 2022

[9] Doyle et al., “Mission Test Campaign for the EIRSAT-1 Engineering Qualification Model”, Aerospace 9(2), 100, 2022