Joar Andreas Gjersund

# A Reconfigurable Fault-Tolerant On-Board Processing System For The HYPSO CubeSat

Master's thesis in Electronic Systems Design
Supervisor: Milica Orlandic
June 2020

HYPSO
· NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY · NTNU SMALLSAT ·

Joar Andreas Gjersund

# A Reconfigurable Fault-Tolerant On-Board Processing System For The HYPSO CubeSat

HYPSO
· NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY · NTNU SMALLSAT ·

Master's thesis in Electronic Systems Design
Supervisor: Milica Orlandic
June 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

# Summary

This thesis documents the development of the on-board processing system for a small satellite with high throughput, dynamically re-configurable, image processing capabilities. The system consisted of a dual ARM core Zynq-7000 SoC that was made to run a customized Linux operating system loaded using a customized U-Boot bootloader. The final system was proven to provide a resilient framework for over-the-air firmware and software updates by applying redundancy and fallback mechanisms along with checksum algorithms such as CRC-32, SHA-1, and MD5 for integrity validation of data files. The processing system was also able to prove support for both full and partial dynamic reconfiguration of the on-chip Artix-7 grade FPGA.

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

**ASIC** Application Specific Integrated Circuit

**AT&T** American Telephone and Teligraph Company

**BoB** Breakout Board

**BSP** Board Support Package

**CAN** Controller Area Network

**CLI** Command Line Interface

**CPLD** Complex Programmable Logic Device

**CPU** Central Processing Unit

**CRC** Cyclic Redundancy Check

**CSP** CubeSatProtocol

**DMA** Direct Memory Access

**DRAM** Dynamic Random Access Memory

**DSP** Digital Signal Processing

**ECC** Error-Correcting Code

**ECD** Error Correction and Detection

**EMIO** Expanded Multiplexed I/O

**EPS** Electronic Power System

**FIT** Flattened Image Tree

**FPGA** Field Programmable Gate Array

**FSBL** First-Stage Bootloader

**GAL** Generic Array Logic

**GPIO** General-Purpose Input/Output

| | |
|---|---|
| **GUI** | Graphical User Interface |
| **HDF** | Hardware Defintion File |
| **HDL** | Hardware Description Language |
| **HSI** | Hyperspectral Imaging |
| **HYPSO** | Hyper Spectral Imager for Oceanographic Applications |
| **IP** | Intellectual Property |
| **ISS** | International Space Station |
| **MPGA** | Mask-Programmable Gate Array |
| **MPSoC** | Multi-Processor System-on-chip |
| **NTNU** | Norwegian University of Science and Technology |
| **OCM** | On Chip Memory |
| **OPU** | On-board Processing Unit |
| **OS** | Operating System |
| **OSL** | Open Source Linux |
| **PAL** | Programmable Array Logic |
| **PC** | Payload Controller |
| **PHY** | Physical Layer |
| **PLA** | Programmable Logic Array |
| **PLD** | Programmable Logic Device |
| **QSPI** | Queued Serial Peripheral Interface |
| **RAM** | Random Access Memory |
| **ROM** | Read Only Memory |
| **RTL** | Register Transfer Logic |
| **SDK** | Software Development Kit |
| **SEE** | Single Event Effects |
| **SIMD** | Single Instruction Multiple Data |
| **SISD** | Single Instruction Single Data |
| **SoC** | System-on-chip |

**SoM** System-on-Module

**SSBL** Second-Stage Bootloader

**SSH** Secure Shell

**Tcl** Tool Command Language

**TID** Total Ionizing Dose

**TMR** Triple Modular Redundancy

**VHDL** Very High Speed Integrated Circuit Hardware Description Language

# Part I

# Introduction and background

Chapter 1

# Introduction

It is now 20 years since the concept of CubeSats was formally introduced as an educational platform. By 2019, in total 1317 nanosatellites and CubeSats have been launched worldwide. In Norway, there have still not been any successful student satellites in orbit. The Norwegian student-satellites that have been closest to success are nCube-1, which had a launch failure and nCube-2 which had a deployment failure, both developed at Norwegian University of Science and Technology (NTNU), and HiNCube, developed at Narvik University, which got lost after deployment. Additionally, some missions have been canceled such as NUTS (NTNU) and CubeSTAR (University of Oslo). Since the launch of the HiNCube in 2013, there have according to the NanoSats database not been made any further attempt at launching a Norwegian made student-satellite to orbit [24].

## 1.1 HYPSO Mission

Hyper Spectral Imager for Oceanographic Applications (HYPSO) is the first space mission at the SmallSatLab at NTNU in Trondheim. SmallSatLab is a student-driven multidisciplinary research incubator, initiated as an effort to promote space-related technology and build competence on the field within the academic community at NTNU. The HYPSO mission statement is to ”provide oceanographic data to monitor the effects of climate change and human impact on the world”. By analyzing the spectral signature of light reflected from the earth’s surface it is possible to detect and measure the presence of biological and chemical materials such as algae blooms, seaweed, salt content, forest health, etc. The HYPSO missions goal is to collect and process hyper spectral data on a 6unit CubeSat which will be deployed to low earth orbit where it should stay operational and collect data for 7-8 years before it will be decommissioned as atmospheric friction will gradually slow the vehicle down and finally let it burn up.

## 1.2 Aims and objectives

This thesis will focus on the On-board Processing Unit (OPU) which is responsible for capturing and processing hyperspectral images and other mission data. This thesis will particularly focus on the underlying firmware which makes up the interface between hardware and software, and how it can be modified to protect against potential failures by facilitating for over-the-air software and firmware updates and implement fail-safe fallback mechanisms to protect the system.

## 1.3 Outline

The following chapter will give some background about the problems associated with processing hyperspectral images and a brief description of available technology for high throughput data processing before diving deeper into various methods for assuring error-free data in the system, particularly focusing on instructions data which if left unchecked can cause unpredictable behavior of the OPU. This chapter also gives an introduction to embedded operating systems, their use, and how it can be built and customized according to need.
Part II of this thesis covers the proposed design of the OPU and how it is implemented on the satellite. This part is introduced in chapter 3 with an overview of the challenges the design must overcome. In chapter 4 the actual design is presented, by first giving an overview of the physical system, before diving deeper into the OPU and presenting various details of the design and how it meets the challenges presented in chapter 3. Chapter 5 covers the integration of the design, with a more detailed description of what practical work had to be done to integrate the design, how it behaves, and how it was tested. Finally, in chapter 6, the work is summarized and concluded, and some topics for future work are presented.

# Chapter 2

# Background

## 2.1 Programmable logic devices

Image and video processing are usually characterized by high computational load and strict timing requirements. With super-resolution and hyperspectral imaging techniques, the computational load on the processing system can be significant and beyond what the traditional von Neumann architecture based computer processor can handle. In place of the traditional Single Instruction Single Data (SISD) processing architecture, modern processing systems usually include Digital Signal Processing (DSP) - extensions for performing Single Instruction Multiple Data (SIMD) arithmetic which enables some image and video processing capabilities. For more flexibility and better performance, customized Application Specific Integrated Circuit (ASIC) are sometimes also used to accelerate specific computations, but these are expensive to develop and therefore usually more suitable for batch productions [26]. A less expensive, and increasingly popular approach is to use a Programmable Logic Device (PLD), which makes it possible to quickly implement custom combinational circuits that enable better control of the data path and flexibility with regards to parallelization of the workload.

### 2.1.1 Technologies

Programmable logic devices first appeared in the mid-1970s. At that time the devices could only be configured by hard-wiring a handful of uncommitted logic gates together. Various methods to achieve better programmability and performance have since been developed such as Programmable Logic Array (PLA), Programmable Array Logic (PAL), Generic Array Logic (GAL), Complex Programmable Logic Device (CPLD), and most recently the Field Programmable Gate Array (FPGA) [42, 31]. There also exist some factory programmable devices which are not reconfigurable, such as Read Only Memory (ROM) and Mask-Programmable Gate Array (MPGA). An overview of available technologies can be seen in Figure 2.1. The most promising family of PLD's and currently the most attractive alternative to ASIC is the FPGA, which can provide several million re-configurable gates on one chip. I addition, modern FPGAs today are usually also enhanced with other hardware components such as DSPs, networking cores, and complete multi-core processors making up a complete system often referred to as a System-on-chip (SoC) or a Multi-Processor System-on-chip (MPSoC).

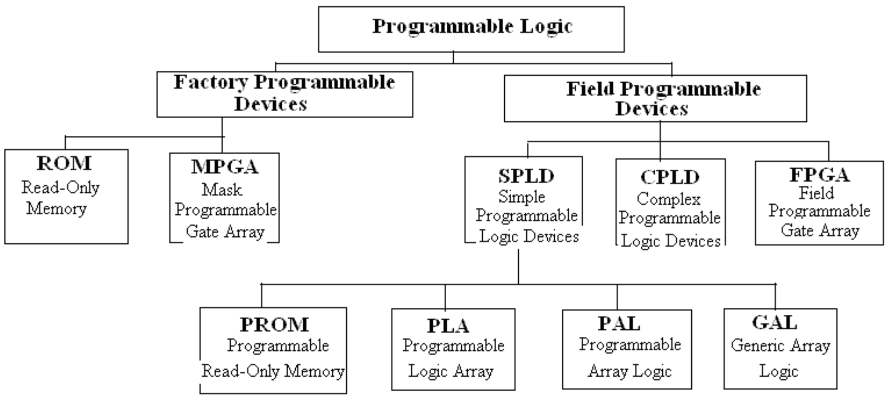

Programmable Logic
Factory Programmable Devices
Field Programmable Devices
ROM Read-Only Memory
MPGA Mask Programmable Gate Array
SPLD Simple Programmable Logic Devices
CPLD Complex Programmable Logic Devices
FPGA Field Programmable Gate Array
PROM Programmable Read-Only Memory
PLA Programmable Logic Array
PAL Programmable Array Logic
GAL Generic Array Logic

**Figure 2.1:** Overview of programmable logic devices [41].

### 2.1.2 State of the art and performance considerations

Current state of the art PLD's rely on the FPGA technology which usually is implemented as a SoC together with multi-core processors and multiple-level on-chip memory [40]. This art enables deployment of software which can re-program the PLD according to need, thereby optimizing the area utilization by taking advantage of temporal mutual exclusiveness [30]. The two top players in the field of FPGA manufacturers are currently Xilinx and Intel, [40]. The two most recent flagship FPGAs from Xilinx and Intel is Xilinx's Virtex and Intel's Agilex. The Virtex, which is found in the Ultrascale+ SoC uses a 16 nm fabrication technology, and Intel's Agilex also found on some recent SoC's from Intel is on the other hand built on 10 nm technology.

Finding a suitable FPGA candidate to use when implementing a processing system is however not just a matter of finding the one who uses the smallest technology node. Although smaller technology usually is a good indication of the overall performance, other aspects such as what hardware components are available, what development platform is supported, and how well documented it is should also be considered. Generally, state-of-the-art technology is often lacking in both documentation and community support which both need time to grow. How close to the state-of-the-art one choose to operate should therefore also be defined by the developers' experience and available resources.

### 2.1.3 Partial reconfiguration

Partial reconfiguration is the ability to reconfigure only selected areas of the FPGA after its initial configuration. This facilitates the idle parts of the programmable logic to be swapped out while active parts are still running, thereby enabling increased utilization of the available area of the FPGA [30]. Partial reconfiguration can also contribute to increased fault-tolerance towards single-event upsets by using it in conjunction with read-back to detect and replace corruption in the configuration memory [20]. A more in-depth study on FPGA design workflow and practical applications can be found in section 2.4 and in appendix A.

### 2.1.4 Limitations and fault tolerance

Digital devices such as FPGAs and ASICs are commonly affected by radiation-induced faults. These faults can be both permanent and transient. With the rapid down-scaling and resulting in reduced noise-margins, the susceptibility to radiation-induced transient faults has increased, while the susceptibility to permanent damage has decreased [7, 11]. The necessity to implement techniques for automatic error detection and correction of data is therefore increasing and can be expected to increase in the future.

## 2.2 Error correction and detection

Data in a processing system usually have a varying level of criticality associated with it. Data with high criticality is usually what would be labeled as instructions data, which is data containing instructions to be executed by the processing system. Corruption of such data can cause the processing system to become unpredictable and in some cases even stop responding. Less critical data, where some corruption may be acceptable are often what would be labeled as payload data. Corruption of such data would just result in the quality of that data getting lowered.

The primary reason that not all data is considered critical is due to the overhead associated with correction and detection. To be able to detect a flipped bit, information about the initial value of the bit must be known. This can be done by appending an extra copy of the bit and assure that both bits are equivalent. To both check and store, each bit in a message is expensive, another option is, therefore, to encode the message, such that it can be represented by a fewer number of bits, this encoding of data is also known as hashing and is an important component of most integrity checking mechanisms.

### 2.2.1 Error correcting code

All mechanisms for Error Correction and Detection (ECD) must rely on two fundamental concepts; i) hashing for detection and ii) redundancy for correction. This inhibits both performances due to the overhead associated with hashing, and the informational density that can be achieved.

The efficiency of the ECD scheme can be expressed as a fraction of data bits $N_D$, over the total number of data bits and redundant bits, $N_D + N_R$. Equation 2.1

$$E = \frac{N_D}{N_D + N_R} \tag{2.1}$$

The most basic method for performing ECD is Triple Modular Redundancy (TMR) which repeats every bit of data three times and uses simple bit-voting to determine the correct value. In this scheme, the total number of bytes needed to represent one byte of information is three bytes, and the efficiency will thus be 1/3 (Equation 2.1). The reliability of triple modular redundancy is also questionable, as it strongly depends on a week correlation of error between modules. If two of the redundant bytes are stuck at the same value, that value will always be perceived as the correct value. ECD can however also be implemented in more clever ways. The number of redundant bits can be reduced when the hashing is not done over each bit individually as it is done with TMR, but rather done over blocks of multiple bits. One such method is the Hamming Code [16]. Hamming code is based on parity bits, which in the example of even parity is a redundant bit appended to a block of multiple bits to assure that the block contains an even number of high bits. This way, if the number of high bits in a block is found to be an odd number, at least one bit must have been flipped. Parity bits can thus be used to detect the presence of one-bit errors in a block but has no way of knowing which of the bits in the block contains the error, at least unless the number of bits in the block is more than 1, consequently, a parity bit alone can therefore not be used for error correction of multi-bit messages. What Richard Hamming presented in 1950 was however a method to minimize the required number of parity bits required in a block while still being able to detect which bit had flipped. Hamming found mathematically that the required number of redundant parity bits to detect and correct a flipped data bit was given by Equation 2.2. Here r is the number of redundant/parity bits and m is the number of data bits. From this equation, we can see that the minimum required redundant bits when the Error-Correcting Code (ECC) is applied to each bit individually is 2, which is also the same as the number of redundant bits in triple modular redundancy.

$$2^r \geq m + r + 1 \tag{2.2}$$

With Hamming coding applied to a whole byte, the minimum required number of redundant bits are 4, leading to the efficiency of 2/3, and significantly better than for triple modular redundancy. The efficiency can be increased even further by increasing the bit-width of the hamming code. A graph of the efficiency of the Hamming code can be seen in Figure 2.2. Although the efficiency of the code increase with increasing block size, the reliability will decrease. The reliability of the ECC scheme is dependent on the block size and the hamming distance of the code, which is the minimal number of bit changes needed to go from one codeword to another. With a hamming distance of three, as used in this example, the code can only correct one-bit errors in a block [16].

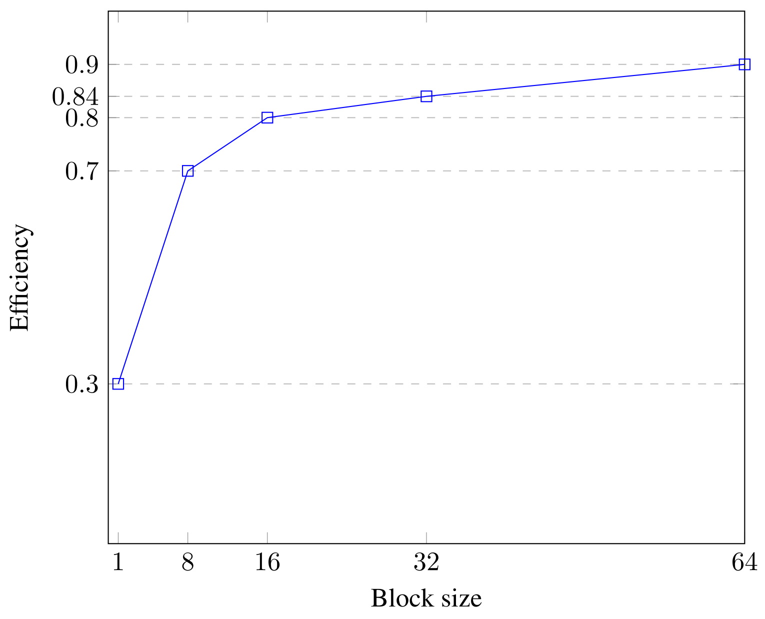


**Figure 2.2:** Efficiency of Hamming code with minimum hamming distance

### 2.2.2 Error detecting code

ECC is useful for recovering permanently corrupted data, but sometimes the error might either just be a transient fault, or a redundant copy of the data may available. In those cases, backward error recovery can be performed, for example by asking for a repeated read of data. In such cases, knowing exactly what bit(s) in a block contains the error is not needed since the whole block will be re-read and re-transmitted either way. To achieve a good efficiency for error detection, the hashing is then often done over a much larger block, often even over complete multi-megabyte data files. For such error detection schemes a hash value, of a certain bit-width, commonly referred to as a checksum, is calculated and appended to the data block. Upon receiving the data, the receiver recalculates the checksum, and if the checksum does not match the appended checksum, the block must be corrupted.

#### Parity

Parity check is the simplest method for error detection and can be used to detect an odd number of corrupted data bits in a message. Parity checks work by appending a single bit to a message to assure that the number of active bits in a message is a pair number (even parity), or odd (odd parity). The receiver can then sum up the total number of active bits in the message including the parity bit and know that if this number is odd, but parity bit is even (or sum is even and the parity bit is odd) at least one bit in the message must be corrupt. Single bit parity checks can only detect an odd number of corrupted bits, thus the probability of detecting a random error in the message can be as low as 50%.

### CRC

Cyclic Redundancy Check (CRC) uses the reminder of polynomial division to calculate a hash which can be used to validate that the message is error-free. CRC uses an n+1 -bit CRC polynomial which is XORed with the message in a cyclic manner starting from the most significant bits and proceeding with the result shifted one bit with every cycle until the resulting message is only zeros. The hash is then the resulting n latest bits discarded during the latest bit shift [34].

The simplest form of CRC is 1 bit CRC, also known as CRC-1. With CRC-1 the CRC polynomial is two bits wide, resulting in a 1-bit hash after the CRC algorithm has completed. this hash indicates if the number of active bits in the message is an even number. CRC-1 is thus the same as even parity described in section 2.2.2. By increasing the number of bits used for hashing the corresponding probability of detecting errors will increase. The ability to detect errors can be seen in Figure 2.3. To stay within a Hamming distance of d which ensures that up to d-1 flipped bits in a message are guaranteed to be detected, the number of bits in the message must be no more than k, [10].

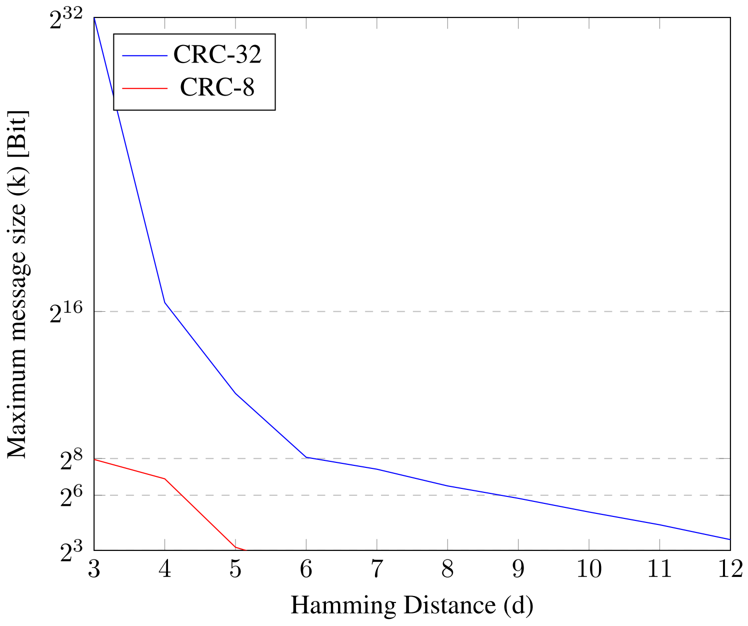


**Figure 2.3:** Maximum bits of payload by Hamming distance [22]

### SHA, MD5, and other cryptographic hashing algorithms

SHA and MD5 are two of many cryptographic hash functions commonly used for detecting data integrity. The ability to detect errors using cryptographic hash functions are usually described as the probability that two unique messages will generate the same hash, which is equivalent to the probability of a random error not getting detected. Such matching hashes are often called a collision and their probability is strongly correlated with the size of the hash, and the block size used by the hashing algorithm. For SHA-1, the hash size is 160 bits, and the block size 512 bits, while for SHA-256 the hash size is 256 bits. Assuming the probability of a collision is uniformly distributed, the probability of a collision can be approximated using Equation 2.3 where **n is the number of data blocks in the message, b is the size of the hash, and P is the probability of a collision** [35].

$$P \leq \frac{n(n-1)}{2} \times \frac{1}{2^b} \tag{2.3}$$

For long messages hashed with small blocks, the number of hashes must increase to cover the whole message, what hashing algorithm to use should, therefore, be partly decided by the length of the message. As can be seen in Figure 2.4, the probability of a random error not getting detected is in most practical cases negligible for hashes with a size of 32-bit. Note that the number of hashes required will be determined by the message size divided by the block size.

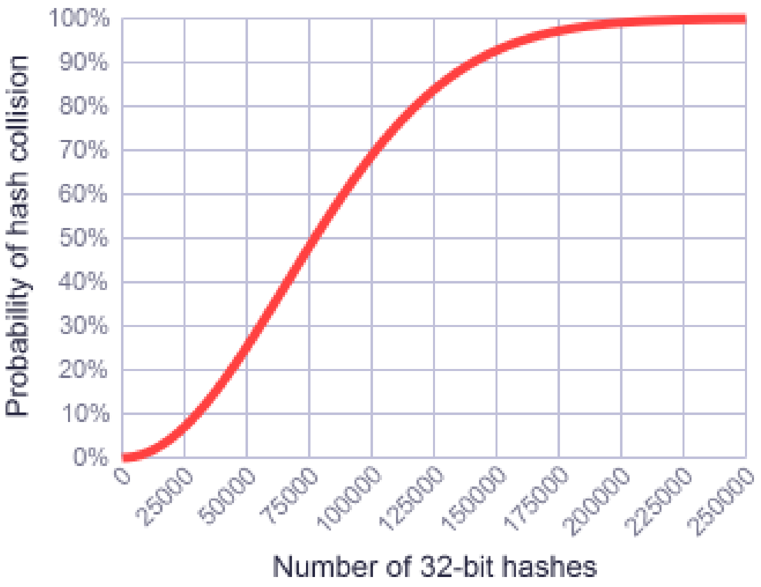


**Figure 2.4:** Probability of a collision given a number of 32-it hashes [35].

Given the low probability of missing an error when using cryptography hashing algorithms like SHA and MD5, it is easy to assume that those are better to use. It is however also important to consider extra overhead that more complex hashing algorithms introduce, and taking the probability of an actual error happening in the first place, less complicated algorithms like CRC-32 may as well be more than good enough to detect non-intentional and random corruption of data.

## 2.3 Embedded operating systems

Embedded processing systems are, contrary to general processing systems, designed to perform a specific task. While general-purpose processing systems often have multiple processes running, and can kill and spawn new processes according to need during run-time, embedded processing systems are often limited to just a few running processes which are automatically spawned when the system starts up and killed when the system shuts down. Besides, embedded processing systems are also often characterized by more strict and predictable response time (Table 2.1).

| Real-Time requirement | Constraints |
|---|---|
| Hard | Missed deadline is system failure |
| Firm | Value of output after deadline is zero |
| Soft | Value of output after deadline degrades over time |

**Table 2.1:** Classifications of real-time requirements

General-purpose processing systems will always run some kind of Operating System (OS) which manage the systems resources and schedule running processes. Depending on the complexity and number of concurrent tasks running, may embedded processing systems also run a light-weight OS, but in many cases, the program running on the system can be programmed bare-metal without any OS. Bare-metal programming gives full access to the systems resources and thus also full control of the systems response time. The trade-off between running an OS and going bare-metal is given by the complexity of the system and the real-time requirements. A complex system may be difficult to implement bare-metal without losing control of how the systems resources are scheduled between processes, which essentially is what the OS does for you.

An OS can be described as a three-layered system; the hardware or physical space layer, the kernel space layer, and the userspace layer (Figure 2.5). Each layer will only interface with the neighboring layer, such that all access to the hardware must go via the kernel space layer. This assures that the kernel can have full control of what hardware resources are accessed by software and thereby protect and schedule resources between software running concurrently on the system.

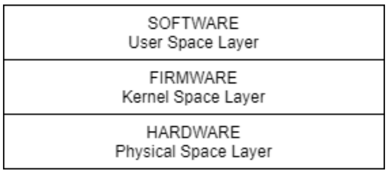


**Figure 2.5:** The three layer model describing the structure of operating systems.

### 2.3.1 Linux

Linux is a free and open-source general-purpose operating system developed in the model of and emulating the Unix architecture whose family of operating systems originates from development done at the Bell Labs research Centre by American Telephone and Teligraph Company (AT&T) [1];[39]. Linux was originally developed as a research project by Linus Torvalds as he studied for a master's degree in computer science at Helsingfors University in Finland. Linux was originally only developed for the Intel x86 family of processors but was from the ground up designed to be easy to transfer to other types of processors by making a clear distinction between hardware-dependent code and code that could easily be ported to new processor architectures by simply recompiling it. As a consequence, Linux has since been ported to a wide range of processor architectures including the popular ARM processor found in many of today's embedded systems. Linux is today included in a majority of all embedded device unit shipments worldwide [1] and has become increasingly popular for use in space. Linux is today found both in satellites and in more critical systems such as SpaceX's Falcon 9, Dragon spacecraft recently used for delivering people and cargo to the International Space Station (ISS) [25].

### 2.3.2 The Yocto Project

Yocto is an open-source development platform for creating custom embedded Linux distributions. Although being widely adopted across the industry, and used by top processing system vendors such as Intel and Xilinx, it is considered to have a steep learning curve, confusing workflow, and long build-time [38]. Some vendors choose to wrap the Yocto building process in their proprietary Software Development Kit (SDK) and accompanying work-flow. Although this is great for usability and decreases the barrier of setting up and customizing a Linux distribution for the vendors' hardware, it may also add extra overhead to an already slow building process. One example of an SDK's which uses Yocto for building Linux Distributions is Xilinx's Petalinux SDK [46].

#### Layer Model

Configuration settings and instructions for distribution builds, are organized in different layers to logically separate information to help simplify future customization and reuse. Examples of such layers are the Board Support Package (BSP) layer which contains primarily target hardware-specific configurations given by vendors such as device-tree and bootloaders, the kernel layer which contains instructions about which kernel drivers to install such as communication ports and memory controllers, and application layers which contain applications and dependencies and instructions on how and where to install it on the root file system. When multiple instances of the same layer are present in a project, the last built layer will overwrite previous builds, this workflow makes it easy to customize and reuse existing layers to suit the requirements of a particular product without making changes to the already existing layer itself. This approach facilitates the isolation of hardware-specific configurations such that configurations that are common across different hardware can be easily shared. [38]

### Project Components

The configuration settings and instructions for Yocto projects may also be referred to as project components. These components consist of three different data types:
i) Recipes (files with a .bb suffix) provides details about pieces of software, such as where to get the source code, which patched to apply, and where on the file system software components shall be placed.
ii) Class data (files with a .bbclass suffix) contains abstract information about how to build the component.
iii) Configuration data (files with a .conf suffix) contains configuration definitions and settings to customize the recipe and class data for the particular build. [38].

### Packaging

For parsing, executing, and building the different layers and their underlying components that together make up the OS, a make-like build tool called BitBake is used. BitBake will also package all components of the OS into various image files. These image files usually consist of
i) Kernel image: The program that runs in the background and schedules how system resources accessed by running processes.
ii) device tree: A file describing the target-dependent hardware.
iii)root-filesystem-image: Contains configuration files, software, scripts, and other user-files that are mounted to the root/top directory of the OS at startup.
iv) Bootloaders: Program that is responsible for setting up memory and other peripherals, extracting i, ii, and iii to the prepared memory, and updating the program counter to the memory address of the kernel (see section 2.3.3).

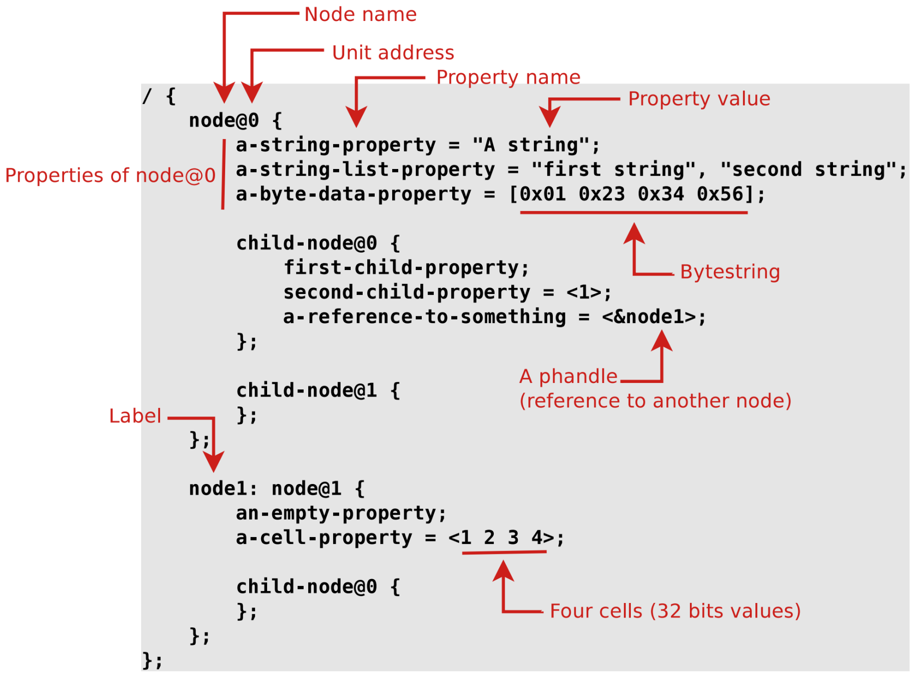
## 2.3.3 Linux components

### The Linux kernel

The kernel is the program that runs in the background and schedules access to the available hardware in the system. In the OS layer model (Figure 2.5) the kernel is the interface between hardware and userspace. This also implies that the kernel is hardware specific, and thus not necessarily portable between different target hardware. In efforts to increase the portability and reduce the number of forks on the Linux Kernel, device drivers are often made configurable via kernel modules that can be used to dynamically extend the kernel and a separate configuration file called a device tree that contains information about the underlying hardware [32]. This for instance makes it possible for Xilinx to have only one fork of the Linux Kernel which supports all their products [45].

### Device tree

A device tree is a file containing information about what hardware is available to the kernel. The device tree's source file (.dts suffix) is formatted as a tree where each hardware components is described in a separate node as can be seen in Figure 2.6.



```
/ {
    node@0 {
        a-string-property = "A string";
        a-string-list-property = "first string", "second string";
        a-byte-data-property = [0x01 0x23 0x34 0x56];

        child-node@0 {
            first-child-property;
            second-child-property = <1>;
            a-reference-to-something = <&node1>;
        };

        child-node@1 {
        };
    };

    node1: node@1 {
        an-empty-property;
        a-cell-property = <1 2 3 4>;

        child-node@0 {
        };
    };
};
```

**Figure 2.6:** Basic device tree syntax [33].

For deployment, the device tree source files are converted to a device tree blob file (.dtb suffix) which is a binary file that can be loaded by the bootloader and parsed by the kernel at boot time [33]. To support dynamically configurable hardware such as programmable logic, Linux has also added support for device tree overlays, which enables the device tree to be dynamically configured at run time.

### Bootloaders

Bootloaders can be defined as a small program that initializes hardware before handing off execution to a more complex program. To start up a Linux OS a chain of individual bootloader stages is usually needed. The startup procedure usually begins with a hardware-specific read-only bootloader implemented by the hardware manufacturer, and proceeds with a First-Stage Bootloader (FSBL) which is small enough to reside on the On Chip Memory (OCM) but complex enough to initialize external memory and extract the Second-Stage Bootloader (SSBL) to it. The SSBL then handles the extraction and hand-off of the Linux kernel, root file system, and device tree as well as an integrity check and other mechanisms for selecting what boot configuration to use. The most well popular SSBL used for embedded Linux Das U-Boot which is described in more detail in section 2.5.

### Boot images

A boot image is a type of computer-file which encapsulates a complete description of one or more components of a system such as the bootloader (section 2.3.3), kernel (section 2.3.3), device tree (section 2.3.3). or root file system (section 2.3.3). The most common format for Linux images is the zImage, which is a minimalistic image only containing a small header, some code for performing decompression, and the compressed data (see Table 2.2) Because it can only contain one image it is sometimes referred to as a single component image [28].

| |
|---|
| Header |
| Decompression Code |
| Compressed Data |

**Table 2.2:** zImage format [14]

Depending on what bootloader is used, the rootfs, device tree, and kernel can be encapsulated into a single file, known as a multi-component image. This file often also comes with additional fields for checksums to detect corrupted images. Checksum tests together with redundant images can be used for both forward and backward error correction, which is critical for reliability in noisy environments. The two most popular image formats bootable with the U-Boot bootloader (section 5.2.1) are the legacy uImage and the more recent Flattened Image Tree (FIT) [14]. Both can be generated with the mkimage utility that comes with U-Boot [19] and is also part of the Yocto workflow.

The uImage format includes all images in a single block check-summed with CR32 (see section 2.2.2 for more details). The layout of the uImage can be seen in Table 2.3.

| |
|---|
| Header |
| Header Checksum |
| Data size |
| Data load address |
| Entry point address |
| Data CRC |
| OS, CPU |
| Image type |
| Compression type |
| Image name |
| Image data |

**Table 2.3:** uImage format [12]

The other popular image format is FIT, which is a more flexible multi-component image. For example will FIT type images support multiple different hardware, software, or kernel configurations in the same image. This image format also supports Integrity protection for each image with various hash algorithms such as sha1, sha256, and md5 as well as the CRC32 algorithm. The FIT image structure is described with an Image Tree source (.its file) inspired by how the device tree (section 2.3.3) is structured, an example of this can be found in appendix D. The .its file is taken as an input by the mkimage utility which outputs an Image Tree Blob (.itb file) which is bootable from U-Boot. [14] [28].

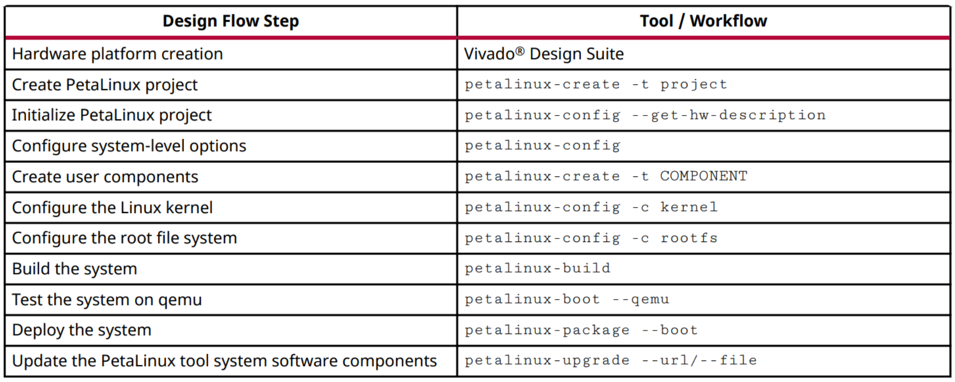
### Root file system

The root file system is the file system mounted to the upmost directory in Linux during boot. It contains all user-space applications and configurations, such as command-line utilities, kernel modules for additional device drivers not already included with the kernel, and various configuration files for setting up networking parameters, startup behavior, and other critical files needed for the system to boot up correctly.

Depending on system requirements the root file system can be mounted on either volatile Random Access Memory (RAM) or non-volatile flash memory. If mounted to volatile ram it is usually done so in a driver-less *tmpfs* configuration opposed to the earlier method of Ramdev where the root file system was mounted on a simulated hard drive on ram [18].

## 2.3.4 Petalinux-tools workflow

Petalinux-tools are according to Xilinx the recommended flow for building Linux systems for Zynq chips. Petalinux-tools follow a sequential workflow model, making it easy to use, but giving little room for minimal rebuilds for efficient prototyping and testing. The sequential workflow follows the steps shown in Table 2.4 [44].



| Design Flow Step | Tool / Workflow |
|---|---|
| Hardware platform creation | Vivado® Design Suite |
| Create PetaLinux project | `petalinux-create -t project` |
| Initialize PetaLinux project | `petalinux-config --get-hw-description` |
| Configure system-level options | `petalinux-config` |
| Create user components | `petalinux-create -t COMPONENT` |
| Configure the Linux kernel | `petalinux-config -c kernel` |
| Configure the root file system | `petalinux-config -c rootfs` |
| Build the system | `petalinux-build` |
| Test the system on qemu | `petalinux-boot --qemu` |
| Deploy the system | `petalinux-package --boot` |
| Update the PetaLinux tool system software components | `petalinux-upgrade --url/--file` |

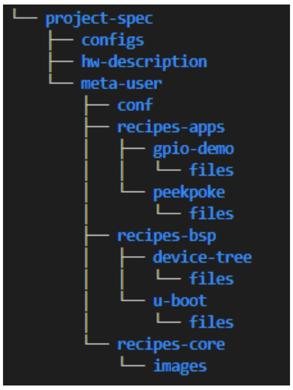
**Table 2.4:** Petalinux-tools design flow overview [44].

### Vivado

Vivado Design Suite is a hardware platform design tool for configuring hardware available, and for the development and configuration of the FPGA. This tool's work-flow which will be described in more detail in subsection 2.4.2, is used to export a Hardware Defintion File (HDF) for configuring the processing system, and a bitstream for configuring the FPGA.

### petalinux-create

The `petalinux-create` command creates a new app, module, or project using either template or source files depending on what is specified when running the command. When creating a project using a standard template, a set of configuration files and a set of Yocto components such as a kernel, bootloader, and device tree, are created and added to a layer called meta-user as shown in Figure 2.7. When an app or module is created, they will be placed under the recipes-apps and recipes-modules respectively.



```
└── project-spec
    ├── configs
    ├── hw-description
    └── meta-user
        ├── conf
        ├── recipes-apps
        │   ├── gpio-demo
        │   │   └── files
        │   └── peekpoke
        │       └── files
        ├── recipes-bsp
        │   ├── device-tree
        │   │   └── files
        │   └── u-boot
        │       └── files
        └── recipes-core
            └── images
```

**Figure 2.7:** Template zynq project generated with petalinux-tools.

### petalinux-configs

Each `petalinux-config` step automatically spawns a dialog menu for configuring the project or any of its underlying components, which in turn edits, or adds files and components in the corresponding meta-layer in the projects project-specs folder, or either of the configuration files. In most cases, this usually involves defining a macro describing the particular configuration which will then be used to include that feature or setting when the particular component is being built. Although the menu is easy to use, it does not properly structure the configurations made so that it is easily portable across different hardware as discussed in section 2.3.2 and should, therefore, be avoided. The menu is however great for displaying the available settings to show which macros and files are available, which then can be added to the project using a bash script, for example through the find and replace command ”`sed`”, or the append to file command ”>>”.

### petalinux-build

The `petalinux-build` command is used to build either the whole project or individual components. It will use the Yocto Project to parse the various project components and packages them accordingly.

**petalinux-package**

When the four main components; kernel, root-file-system, device-tree, and the bootloader are built, they can be packaged into a format suitable for deployment using the `petalinux-package` command. The two most interesting files generated by this command is a `BOOT.BIN` file containing the first and second stage bootloader, and a `image.ub` file containing the kernel, root-file-system, and device tree in a FIT image.

### 2.3.5 Open Source Linux Workflow

The Open Source Linux (OSL) workflow is an alternative to the much simpler Petalinux-Tools workflow (subsection 2.3.4) and involves working directly with the source files, thereby giving full transparency of how the system is set-up. It gives more flexibility for customization of the kernel and bootloader but is usually not needed unless working with unsupported state-of-the-art hardware.

### 2.3.6 File Systems for Block Devices

Mounted media devices that are accessed by an operating system and provide non-volatile data storage are usually managed by a file system. File systems provide a way of structuring data into files and folders, and additional features such as access control mechanisms, metadata for keeping track of accesses and modification of data files, and in some cases mechanisms for preventing corruption of data via ECC. There exists today hundreds of different file systems with different strengths and weaknesses, particularly regarding speed, and reliability. The default file system used with most Linux hosts is Ext -type file systems (extended file system), while Microsoft Windows usually uses NTFS (New Technology File system) or FAT-types (File Allocation Table). Performance analysis of the mentioned file systems has shown that FAT32 which has a simpler structure and smaller overhead, thus performing faster block allocation, while Ext3 and Ext4 performed better on more complex elements such as fragmentation and journaling for better protection against data corruption [9].

## 2.4 FPGA design and work flow for Xilinx devices

As discussed in section 2.1 algorithms implemented on programmable logic have a more flexible datapath which is particularly useful for algorithms that do not rely on sequential dependencies and that can be decomposed into smaller independent tasks. This is because programmable logic can implement true pipelining and parallelism allowing more work to be done on a single clock cycle then what would be the case on a sequential processing device such as the ARM-based processor found on the Zynq SoC.

### 2.4.1 General Use-case

An FPGA accelerated algorithm often referred to as a ”hardware accelerator” is controlled by a Central Processing Unit (CPU) which is responsible for loading the design onto the

FPGA using a bit-stream, preparing the input data to the accelerator, and finally read the output when the job is done. Control signals and data are transmitted between the CPU and FPGA using an AXI communication interface [44]. The general flow can be seen in Figure 2.8. In cases where the FPGA design is contained in a partial configuration, the full/static bit-stream must be configured first.

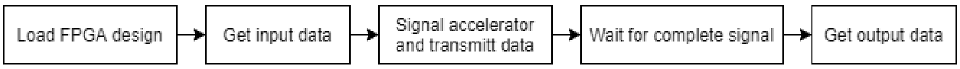


**Figure 2.8:** General use-case flow.

### 2.4.2 Vivado work flow

Vivado is a tool by Xilinx used for hardware design and development, including synthesis and analysis of Hardware Description Language (HDL) designs and SoC development such as pin mappings and peripheral device interface configurations for the processing system. Vivado is a Graphical User Interface (GUI) tool but supports scripting using Tool Command Language (Tcl), which provides good support for automated builds and transparent version control, making it compatible with automated continuous integration testing frameworks. This section will give a summary of the workflow used for developing projects that want to utilize the capabilities of partial reconfiguration. A more detailed tutorial was also made as a part of this literature study and can be found in appendix A.

#### Overview

Vivado supports a hierarchical module based project workflow, where interconnections between modules are defined in a hierarchical block diagram. This allows complex designs to be managed with high-level abstraction. Modules in a block diagram can be either closed source Intellectual Property (IP) or an open-source Register Transfer Logic (RTL) design, written using a HDL language like Verilog or Very High Speed Integrated Circuit Hardware Description Language (VHDL).

#### Workflow for configuring static design and reconfigurable partitions

i.i) Create a top-level block design including the processing system block.
i.ii) add IP and modules to be included in the static partition.
i.iii) add one black box module (a module that has not been instantiated inside the project) for each re-configurable partition to be used as a wrapper for all modules on that partition.
i.iv) generate global output products and add a RTL wrapper for top level block design
i.v) synthesize project and export hardware definition file.
i.vi) open the synthesized design and define the size of the re-configurable partition for each black box module and make a checkpoint

**Workflow for implementing a module to be loaded to a re-configurable partition**

ii.i) add source files, making sure the top module is compatible with the black box top module of the partition.
ii.ii) synthesize as an out of context module by referencing checkpoint from i.vi and defining the black box module as modules top.
ii.iii) Run verification utility to confirm that design is compatible with the reconfigurable partition.
ii.iv) generate a full and partial bit-streams.

## 2.5 Das U-Boot

Das U-Boot is a portable open-source bootloader. It uses a Command Line Interface (CLI), usually accessed over a serial port which can be used to execute small procedures for accessing various devices such as memory, programmable logic, and for toggling the voltage on General-Purpose Input/Output (GPIO) ports. The bootloader is also able to load bootable images (section 2.3.3) and perform the necessary integrity checks described in the image file. A list of all supported commands can be found by typing `help` in the U-boot CLI. Additional commands and drivers can be installed by defining the proper macro in the build options (see subsection 2.3.2). An overview of all the available macros can be found in the README-file available in the U-Boot Github repository [2]. Along with the commands, U-Boot also uses environmental variables, which can be used to store sequence of commands and memory information such as memory offsets and size. Variables containing sub commands can be executed as commands by calling the run command before the variable name. The bootloader is by default configured to run the sub-commands found in the bootcmd variable at startup, making it possible to fully automate the booting procedure.

# Part II

# Design and implementation

Chapter 3

# Development of a student CubeSat

## 3.1 Physical challenges

### 3.1.1 Space environment

Planet earth's magnetic field deflects high energy ionizing particles that could otherwise cause transient voltage peaks in electronics capable of temporally corrupting electronic signals or flipping a bit of memory known as a Single Event Effects (SEE). High doses of ionizing radiation can also cause permanent damage to electronics known as Total Ionizing Dose (TID). In space, low energy radiation may also cause problems. Earth's atmosphere also provides an environment with high heat capacity, effectively low-pass filtering the extreme temperature fluctuations caused by periodic exposure to the sun. Outside the earth's atmosphere the temperature difference between day and night is intense, and likewise will any heat source, such as the heat generated by power-hungry electronics be difficult to dissipate due to the insulating properties of the vacuum.

### 3.1.2 Power

Student CubeSats are usually powered using solar panels. This requires both that the satellite is angled correctly towards the sun, and that the energy harvested can be stored to provide power during nighttime. To stay within the power budget, the satellite must be able to turn on and off electronic components on demand. The satellite should also handle power outage or brownouts without causing permanent damage to components.

### 3.1.3 Reliability

A student CubeSat is challenging in the sense that it is often both developed and operated by students with limited prior knowledge in the field. It must, therefore, be expected that mistakes will be made both during development, testing, and operation. Nevertheless, it is important to plan for what can go wrong and implement systems for remote patching and automatic recovery.

### 3.1.4 Size and weight constraints

CubeSat is an open standard that defines various constraints for miniature satellites. CubeSat dimensions are specified in units, where one unit has a size of $10cm^3$ and about $1kg$ weight. By adhering to the CubeSat standard when developing a satellite, the cost associated with launching to orbit is drastically reduced, largely because the deployment mechanism does not have to be designed from scratch, and because multiple independent satellites can be deployed simultaneously [37].

## 3.2 Practical challanges

### 3.2.1 Parallel development

One challenge when working in a multidisciplinary team is to coordinate how to do co-operative development across multiple disciplines. For this to work smoothly it is important to have a clear plan for the different requirements that makes up the system. These requirements must be defined in a system design plan before development can begin. Requirements that have not been defined in the system design plan, and thus also not properly tested, is likely to cause problems at a later stage in the development phase when components of the system are being integrated. For inexperienced developers the system design plan is likely to be incomplete before development begins, this is because the development in itself is an educational process where new limitations and not yet thought of possibilities are constantly being discovered. An example of a co-operative design flow between hardware, software, and firmware can be seen in Figure 3.1.

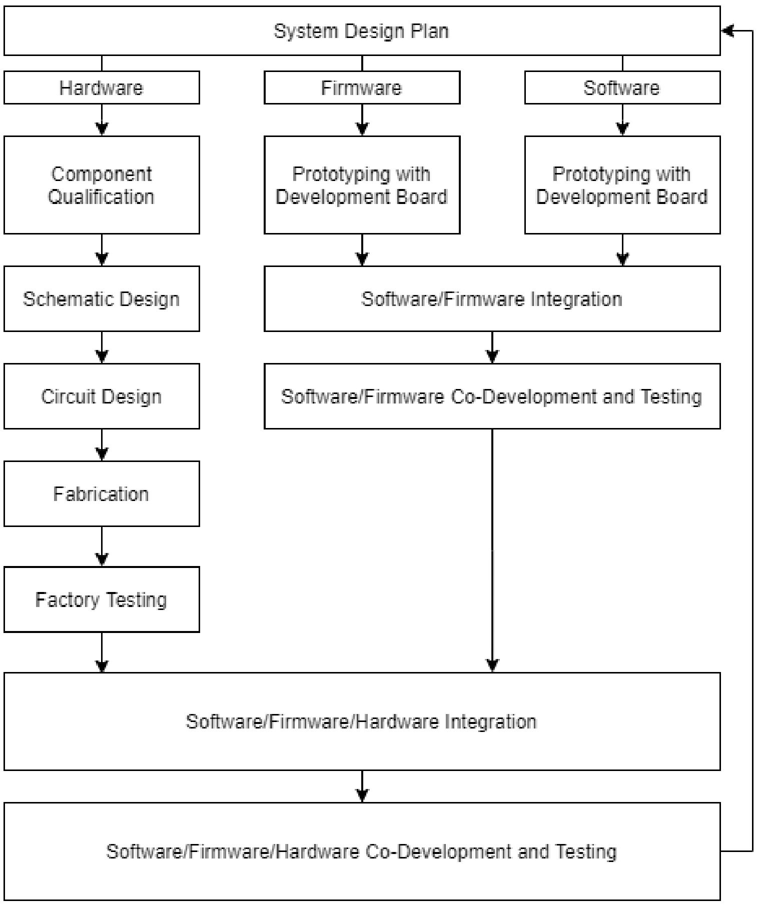

System Design Plan
Hardware
Firmware
Software
Component Qualification
Prototyping with Development Board
Prototyping with Development Board
Schematic Design
Software/Firmware Integration
Circuit Design
Software/Firmware Co-Development and Testing
Fabrication
Factory Testing
Software/Firmware/Hardware Integration
Software/Firmware/Hardware Co-Development and Testing

**Figure 3.1:** Hardware-Firmware-Software co-design flow.

### 3.2.2 Continuity and knowledge transfer

Long term student projects, in general, suffer from high throughput of involved students and a lack of long-term commitment, often limited to 1-2 semesters of work. Students participating in the project must therefore often rely on work done by previous students no longer involved in the project. To succeed it is therefore important to implement good practices for how knowledge is passed on, and to split up larges tasks in smaller sub-tasks with clearly defined boundaries and requirements.

### 3.2.3 Budget

The CubeSat platform was developed as a response to rapidly evolving technology and tight budgets, which required shorter and cheaper mission timelines, especially concerning development. To meet the budget requirements, CubeSats usually take advantage of commercially available technology. The combination of rapid development and the use of commercial-grade hardware not properly tested for the space environment have largely contributed to the low success rate of CubeSat missions [3].

Chapter 4

# On-board processing System for the HYPSO mission

This chapter describes the main considerations and solutions chosen for the firmware of the OPU to be implemented on the payload HYPSO spacecraft. The first section gives an overview of all the modules on the satellite, their purpose, and their interconnects. Finally, a more in-depth look at the hardware, firmware, and software of the OPU are then presented.

## 4.1 System architecture

### 4.1.1 Overview

The HYPSO spacecraft is built on the development platform M6P provided by NanoAvionics, which is a company specialized in providing the subsystems and integrated spacecraft buses for custom CubeSat missions. The M6P bus is confirming to the 6U CubeSat specification [36]. Approximately one cube of the six-unit CubeSat is allocated for the OPU (shown in Figure 4.2), while the remaining space is allocated for sensors and communication. A model of the satellite can be seen in Figure 4.1.

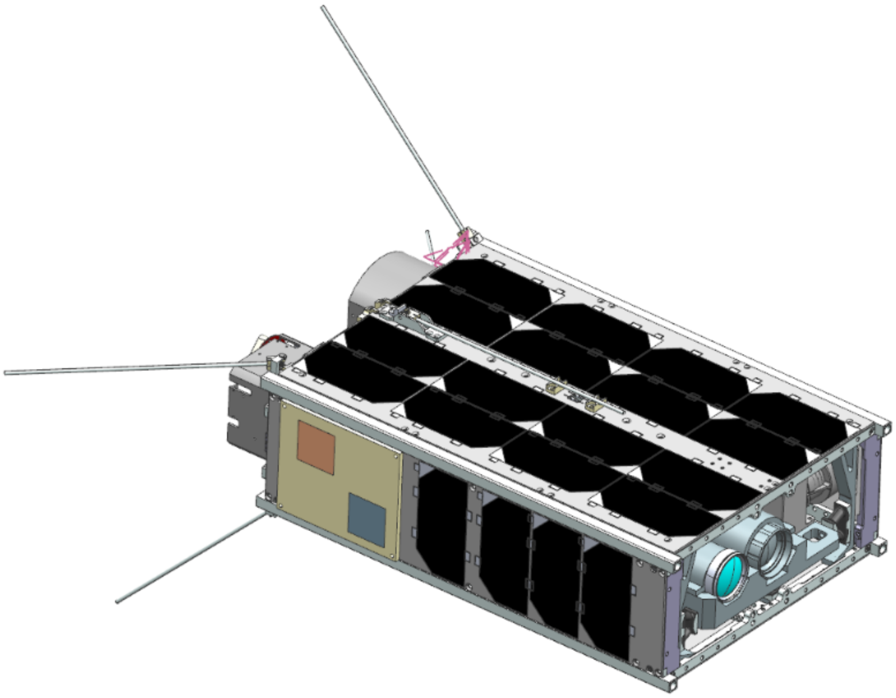


**Figure 4.1:** The HYPSO Satellite, showing features such as the HSI and RGB camera, solar arrays, and UHF antennas.

An overview of the internal system components of the satellite can be seen in Figure 4.2 and shows the physical placement of some of the subsystems such as the Hyperspectral Imaging (HSI) and RGB camera, radio communication, attitude determination and control, and the OPU. This thesis will primarily focus on the OPU also referred to as PicoBoB, which is responsible for controlling the payload instruments and processing the payload data.

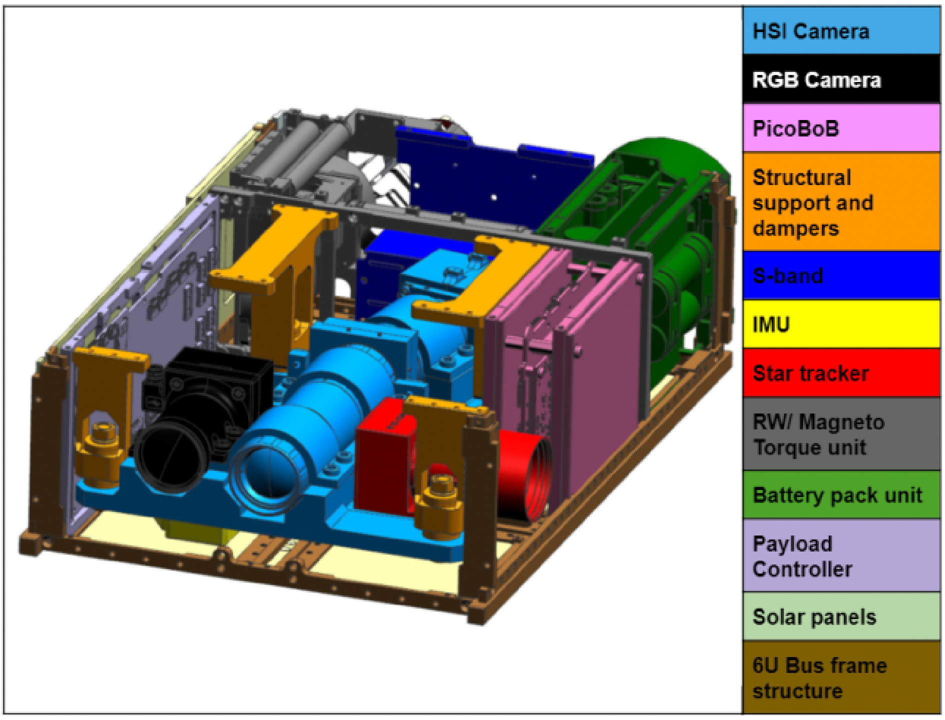


**Figure 4.2:** Overview of the different modules on the satellite and their placement.

## 4.2 On board processing unit

### 4.2.1 Overview

The OPU consists of a PicoZed System-on-Module (SoM) and a Breakout Board (BoB) together usually referred to as PicoBoB. It is responsible for collecting and processing data from sensors on board the payload of the satellite, primarily the hyper spectral imaging sensor and RGB-camera. Most modules on the spacecraft are off the shelf products developed by NanoAvionics, except for the payload consisting of the OPU, HSI and RGB cameras (Figure 4.3) which are customized by students as part of the HYPSO research project. Both instruction data and payload data are transmitted between modules via Controller Area Network (CAN) using the CubeSatProtocol (CSP), or by properitarian protocols via ethernet or USB.

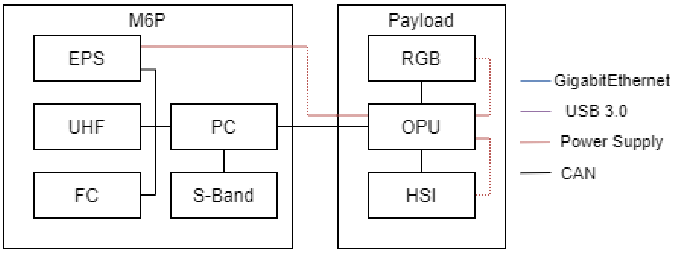


**Figure 4.3:** Overview of modules interfacing with the OPU [15].

The OPU is powered by the Electronic Power System (EPS) which can be controlled via the Payload Controller (PC) such that a power reset can still be made if the OPU stops responding. Additionally all communication between ground and OPU is piped through the PC according to the M6P Platform standard [43].

### 4.2.2 Hardware

The PicoBoB consists of a PicoZed 7030 Rev. E SoM and a BoB to act as a mechanical, electrical, communication, and the thermal interface between the PicoZed SoM and the M6P satellite platform and payload instruments [15]. The PicoZed SoM was manufactured by Avnet and consists of a Xilinx Zynq XC7Z030-1SBG485 AP, commonly referred to as Zynq-7000, a 128 Mb QSPI NOR and 8 GB eMMC Flash storage, 1 GB DDR3 RAM, and a USB and ethernet Physical Layer (PHY) interface controller [4]. The student developed BoB contains interface connectors, SD-Card readers, voltage regulators, and logic level shifters. A 3D rendering of the BoB developed by Amund Gjersvik can be seen in Figure 4.4.

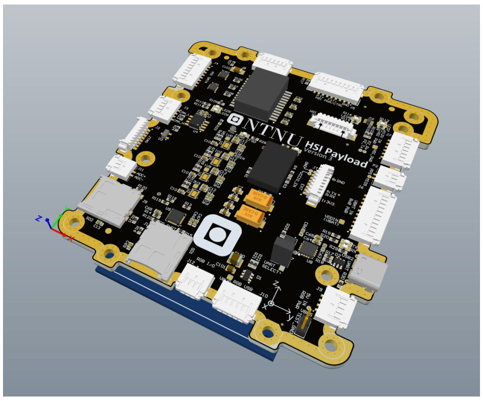


**Figure 4.4:** 3D rendering of the breakout board for the onboard processing system [15].

### 4.2.3 Operating system and services

The OPU runs Linux, which enables a command-line interface for monitoring and managing the systems resources and for starting or stopping services. The command-line interface is accessed through software called opu-services which communicates using CAN through the PC. In addition to the command-line interface, the software also provides file transfer services and commands for capturing and processing data. Figure 4.5 shows the layering of software, firmware, and hardware components according to the layer model presented in section 2.3. Note that opu-services is not the only service that will be running on the system. Other running services include dropbear for ssh access used during development and various camera-related services included with the camera drivers. Kernel modules running on the system include drivers for interfacing with the Direct Memory Access (DMA) reserved memory region, timestamping driver for accurate image frame synchronization, and userspace IO for direct access to hardware that does not require a separate kernel module [21].

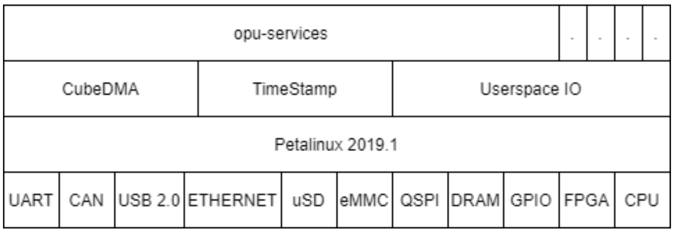


**Figure 4.5:** Simplified layer model of hardware, firmware, and software on the OPU.

### 4.2.4 Software and firmware updates

Software updates are updates of executable file(s) stores on mounted flash memory and executed by the OS after booting. Firmware updates are updates of the whole FIT image, which consists of the kernel, device tree, and root file system (section 2.3.3), which may also include updates of the software and programmable logic. The reasoning behind distinguishing firmware and software updates are primarily due to their size, and the level of risk associated with performing such updates. As shown in Figure 4.6, backward error recovery for firmware updates are done by automatically falling back to the fallback/golden/non-updateable image if the primary/updateable image fails to boot, or software fails to load after the image has booted a set number of times in a row. For software updates, the exact fallback mechanism is defined by the firmware, but is by default set so that software is first loaded from the firmware's root file system, and only if exited loads the software which originates from a software update. This is done to ensure that the system will always be in a known state, defined only by the FIT image after a reboot.

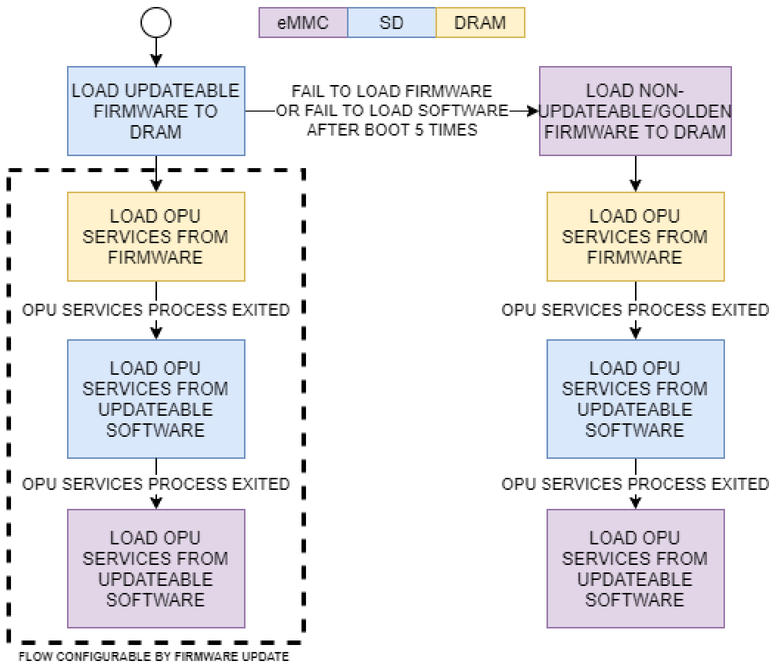

eMMC
SD
DRAM
LOAD UPDATEABLE FIRMWARE TO DRAM
FAIL TO LOAD FIRMWARE OR FAIL TO LOAD SOFTWARE AFTER BOOT 5 TIMES
LOAD NON-UPDATEABLE/GOLDEN FIRMWARE TO DRAM
LOAD OPU SERVICES FROM FIRMWARE
OPU SERVICES PROCESS EXITED
LOAD OPU SERVICES FROM UPDATEABLE SOFTWARE
OPU SERVICES PROCESS EXITED
LOAD OPU SERVICES FROM UPDATEABLE SOFTWARE
LOAD OPU SERVICES FROM FIRMWARE
OPU SERVICES PROCESS EXITED
LOAD OPU SERVICES FROM UPDATEABLE SOFTWARE
OPU SERVICES PROCESS EXITED
LOAD OPU SERVICES FROM UPDATEABLE SOFTWARE
FLOW CONFIGURABLE BY FIRMWARE UPDATE

**Figure 4.6:** Firmware and software execution flow and their memory locations

### 4.2.5 Programmable logic

The Zynq-7000 SoC includes an Artix-7 grade FPGA which is used to do accelerate most of the heavy image processing steps. The FPGA is configured to have one static region consisting of a timer module for timestamping of HSI frames [23], a DMA module optimized for three dimensional DMA access [13], and the general routing of various Expanded Multiplexed I/O (EMIO) pins on the Zynq; and a dynamic region consisting of various image processing accelerators to be reconfigured on the run in the image processing pipeline such as compression [8, 29], smile and keystone correction [27], and various implementations of dimensionality reduction and target detection to filter out the most useful data to transfer back down to the ground station [5].

Deciding on the size and resources to be included in the reconfigurable partition of the FPGA is defined by the most resource-demanding reconfigurable module and is limited by the number of resources available on the FPGA, and how much time it is allowed to take to perform a reconfiguration.

# Chapter 5

# Integration and testing

Integration and Testing of planned designs are perhaps the most time-consuming part of any development project. This chapter will describe some of the details of how the design was implemented on the hardware and what tools and methods that were used to make this process less painful, both for current and future students.

## 5.1 Development framework

The development framework describes what tools, methods, and best practices that are used during development. Having a proper framework prepared before starting development can be what decides how successful the development project will be. A good development framework is something that should help with structuring the project and encourage cooperation between the project's team members. A good development framework should also be quick and easy to set up and be strict in defining what version of software to be used to avoid incompatibility issues between different team member's setup.

### 5.1.1 Macro management

To help with managing the work done by developers at the macro level a method called Kanban was used. Kanban is a way of keeping track of work that has to be done and to better uncover bottlenecks and dependencies across subtasks and thereby making it easier for team members to prioritize what to work with. Figure 5.1 shows the basic layout of the Kanban board used for this development project, a snapshot of the actual Kanban board mid-development can be seen in appendix M. One potential pitfall with using the Kanban method is that it can act as a replacement of the initial design plan, this is especially true for design plans that are lacking details about the various requirements, or if the design plan is buried in a documents archive and where the perceived threshold to update it is too high. As a result of deviating from the initial design plan more responsibility is put on the developers who make non-documented changes to properly define requirements for their design and verify that it is working as expected.

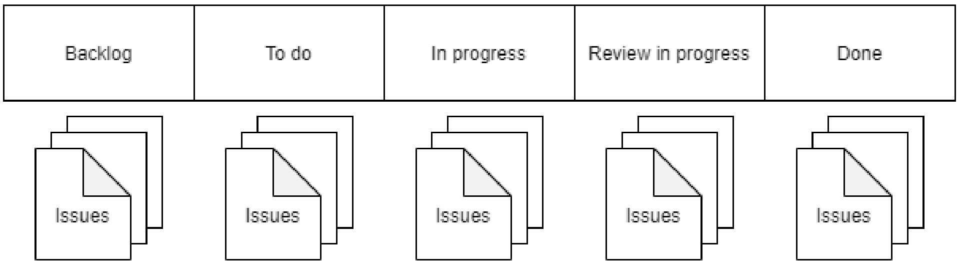


**Figure 5.1:** Layout of the Kanban board. Issues represents detected problems or missing features.

### 5.1.2 Source control

To keep track changes made to the source code and hardware design the distributed version control system Git was used. Git allows developers to work on separate branches dedicated to solving a particular issue (See subsection 5.1.1 for details about how issues are organized). By keeping ongoing work on a separate branch one can assure that the developer can work in a contained environment and that ongoing changes do not cause extra problems for other developers working on separate issues. The common branch which contains all completed work is normally called the master branch. Sub-projects/modules that can be developed and tested individually can be separated into what is called repositories which contains their own set of files, branches, and version history. Repositories are recursive which means repositories can contain child repositories, making it easy to structure large projects. All repositories were hosted on GitHub which also provides integration with issues and Kanban. A typical workflow when developing using Git is:
1) Download latest version of master branch from remote repository
```
git checkout master
```
2) Create a new feature branch to work on
```
git checkout -b branch_name
```
3) Stage all changed files.
```
git add .
```
4) Commit all staged files
```
git commit -m "commit message describing what is changed."
```
5) Create a pull request in GitHub to request committing work to mater branch.
5) Merge and delete feature branch.

### 5.1.3 Knowledge transfer considerations

Knowledge transfer was primarily done through issues and readme files available in the repositories. By using the Kanban method, cooperation between developers was strongly encouraged, which forced developers to write understandable documentation on the work done and how to recreate it. Besides, documentation covering the initial design plan, test plans, and test results were also created, some of which can be found in appendix B and C.

### 5.1.4 Development software

Development tools and software necessary for software and firmware development was automatically set up using Docker. Docker is a virtualization tool for setting up and containing the development environment, assuring full control of what packages and versions are installed on the development workstation. This was used to aid productivity and reproducibility of work done across team members. For the firmware of the OPU which is the primary focus of this thesis, the Petalinux SDK version 2019 was used for building the Yocto Project and packaging the resulting Linux distribution and bootloader to run on the OPU system. The Petalinux SDK has around 50 additional package dependencies which are required for the SDK to work properly. The installation of these is automatically done through a docker file (appendix H), which is a script that can be executed via docker to set up the work environment.

### 5.1.5 Remote access

The ability to work on a development project remotely can sometimes be necessary. Due to very restrictive access to the lab which was enforced by the University as a response to the ongoing pandemic, having remote access to hardware in the lab that was necessary for doing development was important. For accessing the lab computer remotely, the cryptographic network protocol Secure Shell (SSH) was used. It was also found to work particularly well when used together with Visual Studio Code and the Remote Explorer extension.

In addition to having remote access to hardware, it was also important to have a channel for communication between team members. For this purpose, the communication platform Slack was used for text-based and one-to-one communication, and the video conferencing software Zoom was used as a replacement for larger gatherings such as design reviews and weekly Kanban meetings.

## 5.2 Integration

### 5.2.1 Firmware

#### Bootloader and error detection and recovery

The OPU uses the bootloader *Das U-Boot* for booting up the OS. This bootloader binary is stored on the bootloader partition of the on chip NOR flash memory which is accessible by the processing system through Queued Serial Peripheral Interface (QSPI) ((Table 5.1). The bootloader is configured to load a FIT image containing a kernel, device tree, and root file system. Configurations of the bootloader are primarily done through environmental variables stored either on file or included in the bootloader binary. At power-on, the bootloader will enter a setup sequence where memory and drivers are loaded, and where it will try to load an environment from the environment partition on the NOR flash memory shown in Table 5.1. The environment is protected with a CRC32 checksum, with automatic fallback to a clean environment present in the bootloader image. Some of the most important variables which are defined in the environment are `bootcmd, bootm_low, bootm_size, loadaddr, bootcounter, bootlimit`, and `altbootcmd`.

| Offset | Partition Name/Content |
|---|---|
| | Bootloader |
| 0x000000 | BOOT.BIN |
| 0x200000 | BOOT.BIN |
| 0x400000 | BOOT.BIN |
| | Environment |
| 0x600000 | env_blob.bin |

**Table 5.1:** On-chip QSPI NOR flash memory partition layout as defined in the device tree (appendix F)

The content of the bootcmd variables will automatically be executed after the initial bootloader setup sequence. The bootcmd and altbootcmd variables contain ordered lists of boot commands for various images at various memory locations such that if the first boot command fails to load, the next one in the list will be executed. The bootcounter variable holds the number of times the environment is loaded by the bootloader, if no valid environment is present, it will default to 0. To keep track of this number, the environment is automatically saved to flash during each boot, and erased from within the operating system when the system has booted up. The bootlimit variable defines the maximum value of the bootcounter until the bootloader switches from executing the content of bootcmd at startup to the content of altbootcmd, thus making it possible to invoke an alternative booting priority, making it possible to perform backwards error recovery in the event of a software bug on the primary image. The layout of the `bootcmd` and `altbootcmd` variables can be seen in Table 5.2 and 5.3.

| Command | Description |
|---|---|
| `fatload mmc 1 $loadaddr bitstream.bit;`<br>`fpga loadb 0 $loadaddr $filesize;` | Program FPGA. |
| `fatload mmc 0 $loadaddr image.ub;`<br>`bootm $loadaddr;` | Boot image from primary SD. |
| `gpio toggle 46;`<br>`fatload mmc 0 $loadaddr image.ub;`<br>`bootm $loadaddr;` | Toggle SD-Card arbiter and boot image from fallback SD. |
| `fatload mmc 1 $loadaddr image_golden.ub;`<br>`bootm $loadaddr;` | Boot golden image from eMMC. |
| `reset;` | Restart boot sequence. |

**Table 5.2:** Sequence of commands executed by U-Boot as defined in the `bootcmd` variable.

| Command | Description |
|---|---|
| `fatload mmc 1 $loadaddr bitstream.bit;` `fpga loadb 0 $loadaddr $filesize;` | Program FPGA. |
| `fatload mmc 1 $loadaddr image_golden.ub;` `bootm $loadaddr;` | Boot golden image from eMMC. |
| `gpio toggle 46;` `fatload mmc 0 $loadaddr image.ub;` `bootm $loadaddr;` | Toggle SD-Card arbiter and boot image from fallback SD. |
| `gpio toggle 46;` `fatload mmc 0 $loadaddr image.ub;` `bootm $loadaddr;` | Toggle SD-Card arbiter and boot image from primary SD. |
| `reset;` | Restart boot sequence. |

**Table 5.3:** Sequence of commands executed by U-Boot as defined in the `altbootcmd` variable.

FIT images are together with payload data, software and configuration files kept on NAND Flash block memory devices. For simplicity it was decided to use FAT32 filesystem for the FIT images as this was the only supported file system when booting directly from sd card. The file-system and partition layout can be seen in Table 5.4 and 5.5.

| Parition | File-system type | Content |
|---|---|---|
| 1 | FAT32 | image.ub |
| 2 | Ext4 | opu-services + payload data |

**Table 5.4:** SD-Card (mmc 0) partition layout and file-system.

| Parition | File-system type | Content |
|---|---|---|
| 1 | FAT32 | image_golden.ub + bistream.bit |
| 2 | Ext4 | opu-services (fallback) + payload data |

**Table 5.5:** eMMC (mmc 1) partition layout and file-system.

The `loadaddr` and `bootm_size` variables defines the location on RAM where the FIT image is to be loaded and extracted to during boot. The image will be loaded somewhere between `loadaddr` and the end of the memory and extracted somewhere between bootm_low and bootm_size. The memory address is a hexadecimal number, with each bit representing one byte of memory. Table 5.6 shows how the memory is mapped by U-Boot during booting, while Table 5.7 shows how the memory is mapped after being handed over to the OS.

| Address | Content |
|---|---|
| `0x00000000` | U-Boot |
| `0x00008000` | Kernel code and data |
| `0x0b68fb28` | Device Tree |
| `0x0ffffa4a` | Ramdisk |
| `0x10000000` | `$loadaddr` and `$bootm_size` |
| `0x10000104` | Linux Kernel Image |
| `0x103f6cd8` | RAMDisk Image (gzip compressed) |
| `0x103f30ec` | Flattened Device Tree blob |
| `0x40000000` | *END* |

**Table 5.6:** DRAM memory mapping during booting.

| Address | Content |
|---|---|
| `0x00000000` | System RAM |
| `0x30000000` | Reserved Cube DMA |
| `0x40000000` | *END* |

**Table 5.7:** DRAM memory mapping after booting.

As discussed in subsection 2.1.4 data may become corrupt after deployment. Mechanisms for error detection and recovery was therefore implemented both for bootloader and FIT images. The bootloader was configured to use MD5 checksum (section 2.2.2) for verification of the content of the U-Boot image with three copies flashed to the QSPI NOR Flash memory for redundancy. The FIT image was checksummed with SHA1 (section 2.2.2) which was the default algorithm for FIT images built with the Petalinux SDK. In the event of a failed checksum, the fallback golden FIT was set to be loaded. It was not decided to add any extra redundancy as corruption of the primary FIT image can be replaced following the procedure of a firmware update. For protection of temporary data stored on the external Dynamic Random Access Memory (DRAM) during processing a proprietary ECC was available on the on-chip memory controller of the Zynq-7000, and could be enabled in the FSBL through Vivado. Enabling the ECC would however limit the amount of usable DRAM to 50% of its original capacity which was not within the requirements of the image-processing software running on the OS. FPGA solutions for ECC was also considered, but would not allow protection of the memory managed by the kernel, as this would either require support for partial reconfiguration of the FPGA which was still under development; or removal of the already implemented support for full reconfiguration of the FPGA within the OS as discussed in subsection 5.2.2.

### Operating system

The OS on the OPU uses a customized Linux distribution called Petalinux provided by Xilinx and developed with Yocto via the Petalinux SDK (subsection 2.3.2). The OS uses version 2019.1 of the official Linux kernel from Xilinx [47]. The Petalinux SDK automatically configures the kernel according to a hardware description file. The hardware description file was exported from Vivado, which is the tool used for managing what hardware resources that are available to the zynq chip and for setting up the programmable logic. An overview of the available resources on the OPU can be seen in Figure 5.2. Most of the hardware-specific configurations are not directly configured in the kernel, but rather included in the device tree blob as discussed in section 2.3.3. The complete device tree source of the OPU can be found in appendix F.

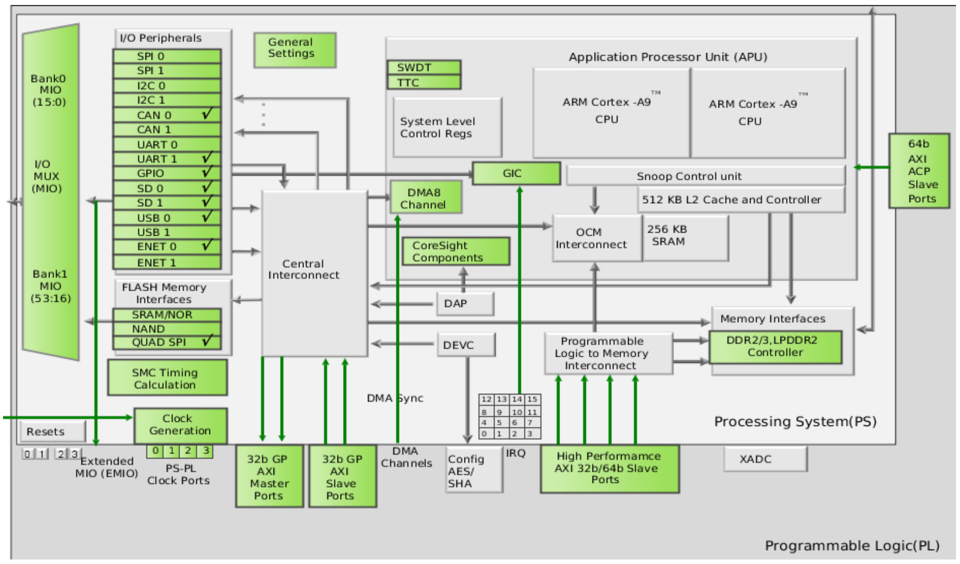


**Figure 5.2:** Overview of the resources available on the Zynq SoC as configured with Vivado.

### File system

The OS was set up to use a RAM based root file system, implemented with a tmpfs configuration. The file system is integrated into the upgradable FIT image stored on NAND flash memory, and loaded to RAM during booting, as described in section 5.2.1. Loading the root file system to volatile memory during booting assures that all files are quickly accessible and that any configuration of the file system that would affect the state of the system is reset after each boot. One of the potential drawbacks of storing the file system on RAM is that it decreases the amount of data memory available for the applications running on the OS. To solve this and decrease the chance of running out of memory, memory swapping was enabled. By allocating a certain size of the slower flash memory for swapping, the kernel would extend the virtual data memory available, and automatically swap the less frequently used data to the flash memory whenever there is no more space left on the RAM.

Setting up the kernel to use tmpfs configuration was enabled by default, and not many changes needed to be performed. However, to assure enough memory was allocated by the bootloader when loading the rootfs, the variable bootm_size had to be defined to a proper value. Additionally, the macro `CONFIG_SYS_BOOTMAPSZ` had to be undefined in the u-boot's configuration file before the bootloader was built. Additionally, the swapspace had to be manually enabled during booting using the swaponoff package which was installed on the OS by defining the `CONFIG_util-linux-swaponoff` macro in the kernels configuration file (appendix G).

### 5.2.2 Programmable logic

The FPGA on the Zynq SoC is programmed with a bitstream file which can be exported from Vivado. To support dynamic reconfigurability of the FPGA during operations from within the OS userspace, an FPGA driver and the fpga-manager package had to be installed on the OS. This was done by defining the macro `CONFIG_SUBSYSTEM_FPGA_MANAGER, CONFIG_SUBSYSTEM_DTB_OVERLAY, CONFIG_CMA`, and `CONFIG_DMA_CM` in the kernels's configuration files. Also, the bitstreams exported from Vivado had to be converted to a bin file to be supported by the FPGA Manager. This could be done using the bootgen utility included with Yocto and Petalinux SDK. The Procedure for this conversion was added to the automatic build script which is discussed in a bit more detail in subsection 5.3.2. As part of this thesis, a quick guide on how to build re-configurable modules and bitstreams was made and can be found in appendix A.

### 5.2.3 Software

The software running on the OPU is packaged into one executable called opu-services. opu-services provides the necessary communication interface for direct access to the OS unix style command-line interface, for up and downlink file transfer, for accessing cameras, and for controlling the image processing pipeline. It is therefore critical that opu-services automatically starts up when the OS has booted. This was assured by adding two commands in the startup script for starting up opu-services, with the first command trying to start from the ramdisk image, and the second one for starting from the mounted flash media in case the first one fails to start or is exited. This also assured that the second opu-services binary which was executed from flash memory could easily be updated without needing to perform a complete update of the whole FIT image (appendix G).

#### File transfer and remote shell access

For uploading and downloading files to the satellite, opu-services implements a file transfer sub-service using the CSP over the CANBUS network. The design and implementation of the file transfer system were done by Magne Hov as part of his master thesis, details regarding functionality and performance are documented in [17]. In addition to file transfer, a sub-service for executing shell commands directly on the OPU's Linux terminal was also implemented. Both file transfer and remote shell access give the ground station full access to all data files on the OPU, including the access needed for performing over the air firmware and software updates and for performing reconfiguration of the FPGA. **/*-/*-*/

## 5.3 Testing

Testing should be done frequently in all stages of development. Failing to do proper testing early in a project will result in more time spent debugging problems at a later stage of the development, and may even result in a flawed design getting deployed. It is therefore important that a plan for how testing shall be performed is established early, and before development has begun. This plan should consider the boundary of the tests to be performed, such that i) the tests can be performed regularly without stealing to much time from developer and ii) that the tests accurately considers the limitations of the surrounding system. The surrounding system is often simplified, either to save time or because the surrounding system is incomplete. It is therefore often necessary to do extra tests called integration testing when all separate parts of the design are complete. The amount of work associated with integration will depend on how accurate the initial development tests are performed.

### 5.3.1 Test methods and setup

The test methods used during this development project can be split into two general types; feature testing and system testing. Feature testing is a private test setup used for testing ongoing work that is not yet ready to be integrated into the test setup of co-developers. System testing is the public test setup and considered the latest and greatest stable release and used for design reviews, integration tests with outside systems, and generally for monitoring the overall progress of the project. The system testing setup should always run the latest release on the project's repository's master branch.

Due to limited available hardware for feature testing, some developers had to use development boards such as ZedBoard for feature testing or coordinate their use of the available PicoBoBs. The development boards had slightly different hardware and thus required some modifications to the firmware which was developed for PicoBoB.

#### Feature test setup

The feature test setup shown in Figure 5.3 had the test boundary defined at the CAN interface between PC and OPU (see Figure 4.3). In addition to the CAN interface, a UART interface was included in the test setup to show additional information about the status of the OPU before the system had booted up and was accessible using the CSP interface established over CAN with opu-services. For certain software development tasks, an extra ethernet connection to the operator computer was also used for faster file transfer capabilities. More details of this test setup can be found in appendix B.

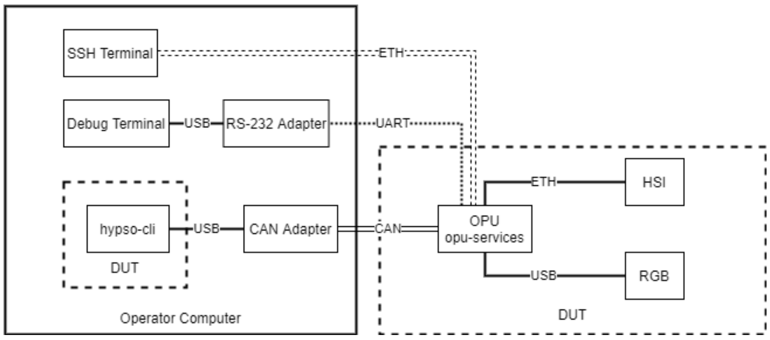


**Figure 5.3:** Feature testing setup used for firmware development.

### System Test Setup

The system test setup shown in Figure 5.4 had the boundary defined at the CAN interface between the PC and the EPS, and other modules on the M6P bus (see Figure 4.3). A more detailed description of this setup can be found in [6].

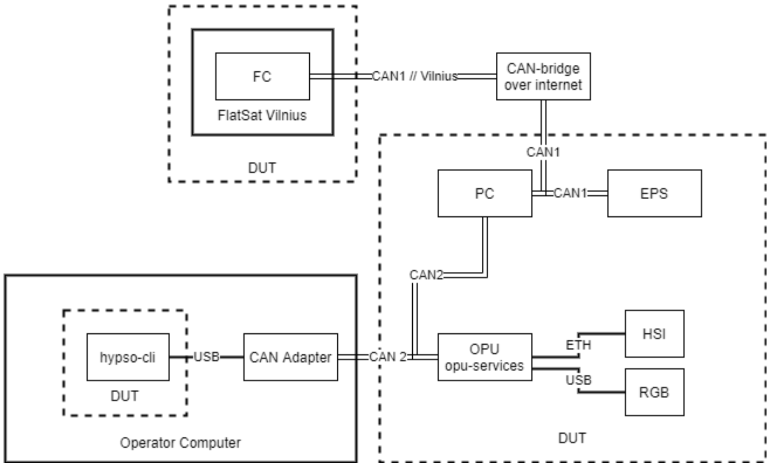


**Figure 5.4:** System testing setup used for integration testing.

## 5.3.2 Building

To easily reproduce and keep track of the configurations made to the system the whole build process was scripted in bash. The script consisted of a nested set of sub-scripts to raise the level of abstraction and thus make the process more systematic. The flow of these scripts can be seen in Figure 5.5, with each separate sub-script is shown in grey. Using this flow the whole build process could be performed running the single command; `build_all`.

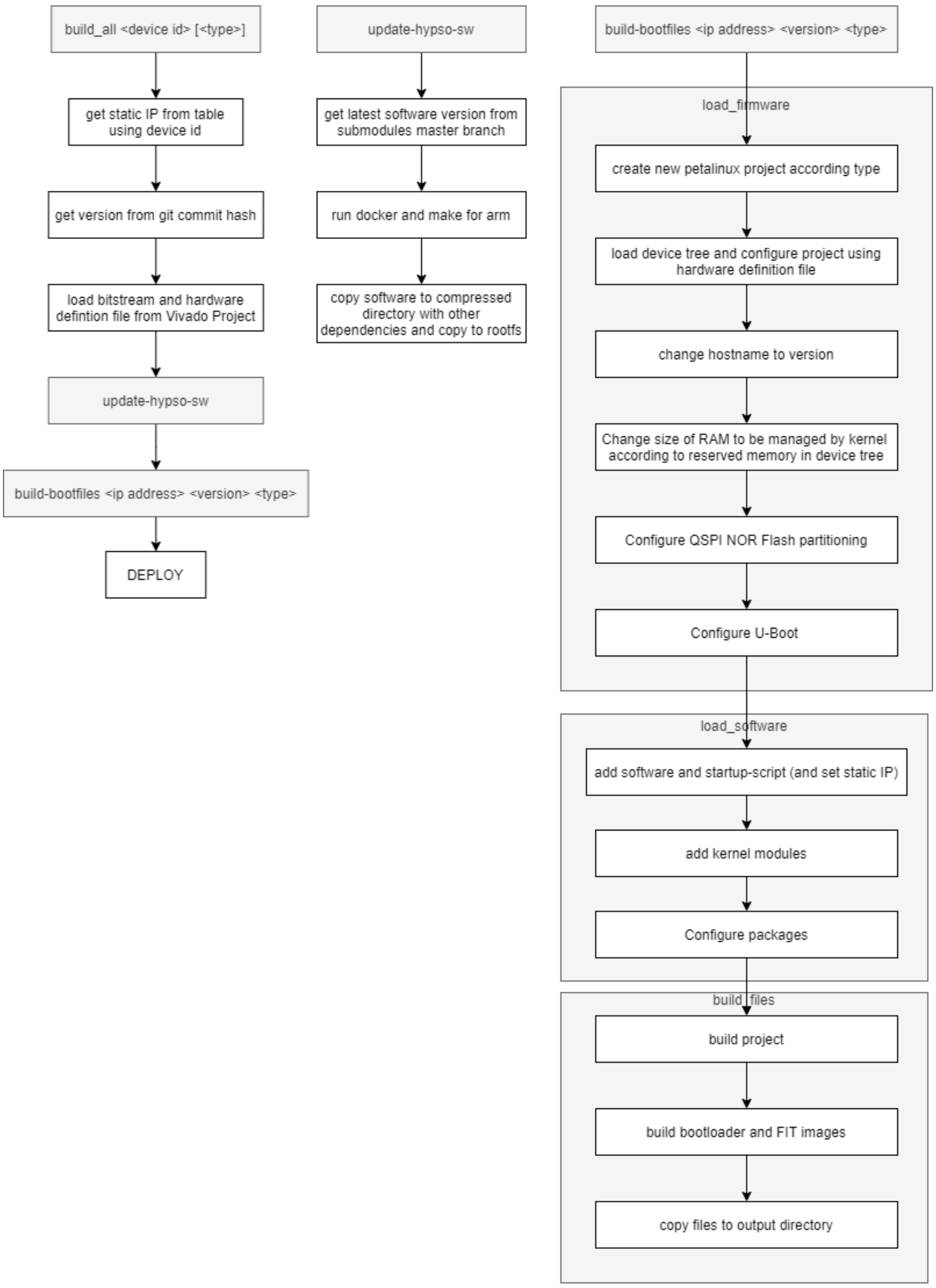

build_all <device id> [<type>]
get static IP from table using device id
get version from git commit hash
load bitstream and hardware defintion file from Vivado Project
update-hypso-sw
build-bootfiles <ip address> <version> <type>
DEPLOY
update-hypso-sw
get latest software version from submodules master branch
run docker and make for arm
copy software to compressed directory with other dependencies and copy to rootfs
build-bootfiles <ip address> <version> <type>
load_firmware
create new petalinux project according type
load device tree and configure project using hardware definition file
change hostname to version
Change size of RAM to be managed by kernel according to reserved memory in device tree
Configure QSPI NOR Flash partitioning
Configure U-Boot
load_software
add software and startup-script (and set static IP)
add kernel modules
Configure packages
build_files
build project
build bootloader and FIT images
copy files to output directory

**Figure 5.5:** Flow diagram of the scripted build process

The complete building process takes 5-30 min depending on build configuration and workstation used. The script supports input arguments for selecting between building for the prototyping ZedBoard or for the PicoZed which will be used for the HYPSO-1 mission, it also supports options for building both the primary and golden image, or for just building the primary. A breakdown of the sub-scripts relevant to the petalinux-tools workflow can be seen in Table 5.8.

| **Script name** | **Petalinux-tools commands executed** |
|---|---|
| load_firmware | petalinux-create |
| | petalinux-config –get-hw-description |
| load_software | petalinux-create -t apps |
| | petalinux-create -t modules |
| build_files | petalinux-build |
| | petalinux-package |

**Table 5.8:** Script and petalinux-tools relationship.

Using bash scripts for automating builds are generally not considered a good practice as it is unable to automatically detect dependencies across builds and to minimize what has to be rebuilt. However, after running the script once, the complete project is set up and can be changed and rebuilt more efficiently using the petalinux-tools makefile commands described in subsection 2.3.4. Making changes to the actual underlying hardware defined by the hardware definition file exported from Vivado would however require a complete rebuilt of the project.

## 5.3.3 File system performance test

Memory access latency is often the main bottleneck in processing systems. It was therefore thought interesting to measure and compare the performance of the three largest memory devices on the system; the external DRAM where the root file system was mounted, the eMMC, and the SD-Card. It was also interesting to see if there was any measurable difference in performance between the FAT or Ext4 File System. To test the performance, various read and write operations with a 64MB large test data spread across 1, 64, and 65K blocks, with a block size of 64MB, 1MB, and 1KB, respectively was performed, while the real-time latency between each operation was measured. The full test script made for this test can be found in appendix I, and the recorded raw data can be found in appendix J.

#### General performance overview

To get the general performance across the different devices and filesystems the average throughput (both read and write) with no caching was calculated and plotted as shown in Figure 5.6. The same graph also shows the overall best-recorded performance, which for all devices was when the 64MB test data was divided into 64 1MB blocks.

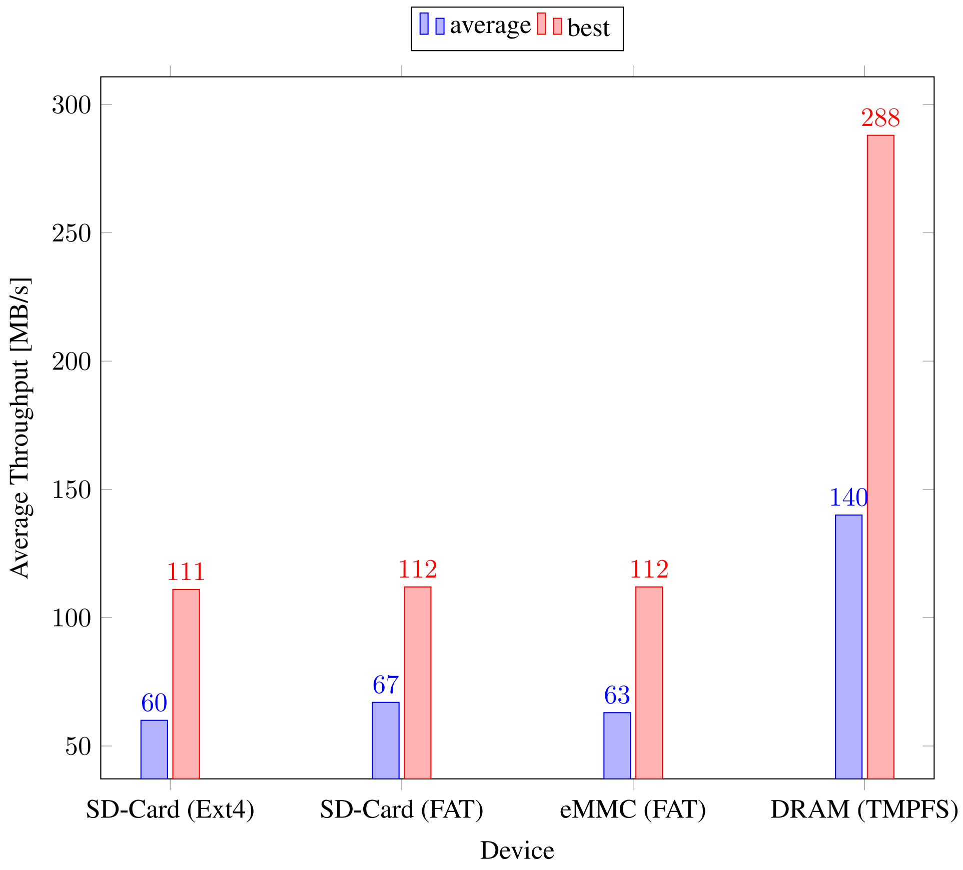


**Figure 5.6:** Average and fastest recorded throughput for various devices and file systems on the OPU.

### Read vs. write performance

The average and best recorded read and write performance plotted in Figure 5.7 shows an up to 50% difference in the recorded throughput between reading and writing data. Comparing the two file-systems FAT and Ext4 indicated that the FAT file-system might be slightly faster on average, supporting the findings of previous research summarized in subsection 2.3.6.

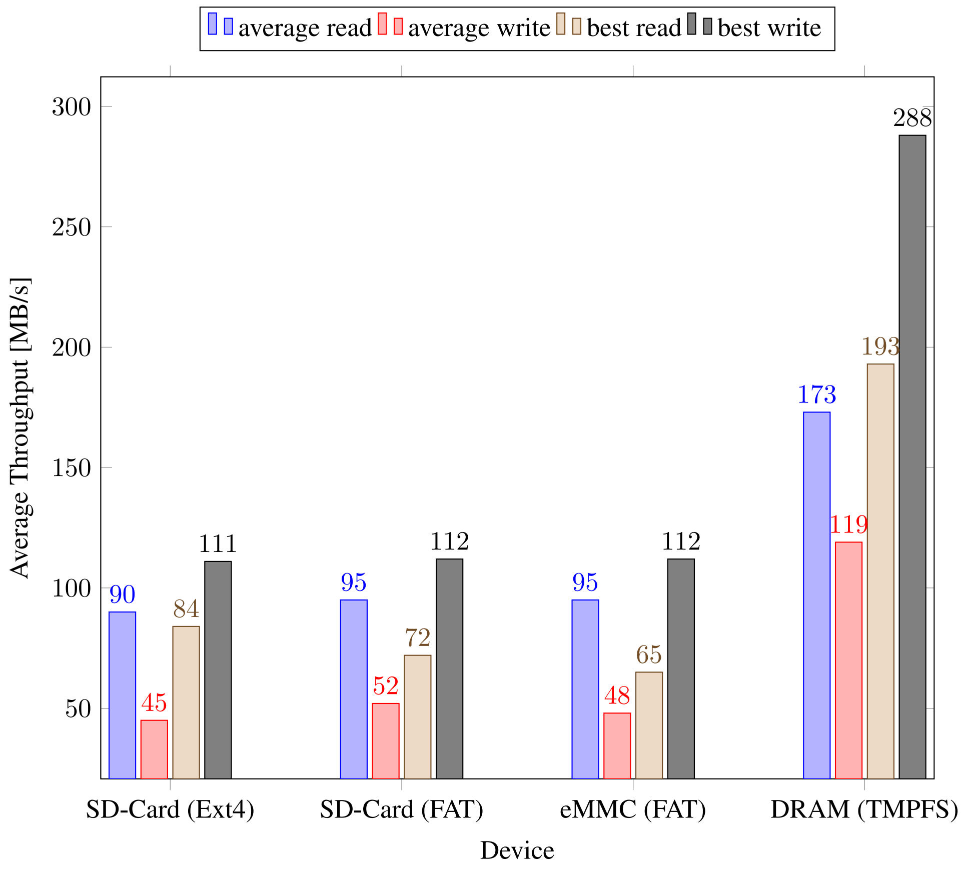


**Figure 5.7:** Average read and write throughput for various devices and file systems on the OPU.

#### Performance with caching enabled

Often the same data is accessed multiple times, in such cases, the Linux Kernel can drastically improve the read throughput by temporally storing a copy of previously accessed data on a RAM. Measuring the read performance with caching enabled showed that the read throughput for all devices increased to the same speed as for the DRAM (TMPFS) device. For more details on these results see appendix J.

### 5.3.4 Dynamic reconfiguration of FPGA performance

The re-configurable modules briefly mentioned in subsection 4.2.5 were not ready to be integrated into the image processing pipeline on the system and could therefore not be used to benchmark and compare the throughput using full and partial reconfiguration to toggle between the different processing cores. However, as a proof-of-concept and a way to gather some performance data, a dummy hyper-spectral imaging processing pipeline was developed. The pipeline consisted of four steps, with each step simulating one operation performed on the complete image, such as compression, correction, and filtering. However, since only the achieved throughput and not the actual result of the data was relevant for the test, the various steps were simply replaced with alternating bitwise left and right shift operation (Figure 5.8).

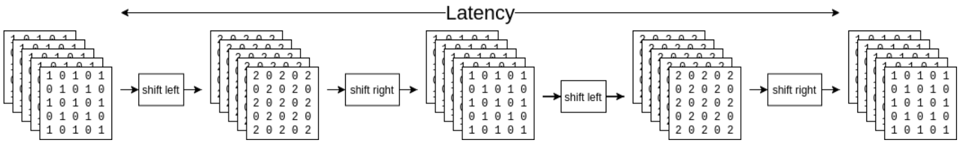


**Figure 5.8:** The dummy hyper-spectral image processing pipeline used for testing.

The pipeline was tested using hardware accelerators implemented on the FPGA and through software (appendix K). To simulate a more complex operation where the resources on the FPGA would have to be time-multiplexed, the FPGA was dynamically reconfigured between each step, one of these tests performed full reconfiguration while the other performed partial reconfiguration using a predefined reconfigurable partition. The resulting latency for various cube dimensions using the three methods can be seen in Figure 5.9. Note that in this particular case the configuration data/bitstreams are first loaded from the SD-Card before loaded onto the FPGA, the extra latency for the smallest cube is likely related to underlying processes running on the OS and possibly the use of caching and prefetching/read-ahead of the files on the SD-Card to reduce the load time, and is not related to the size of the data cube.

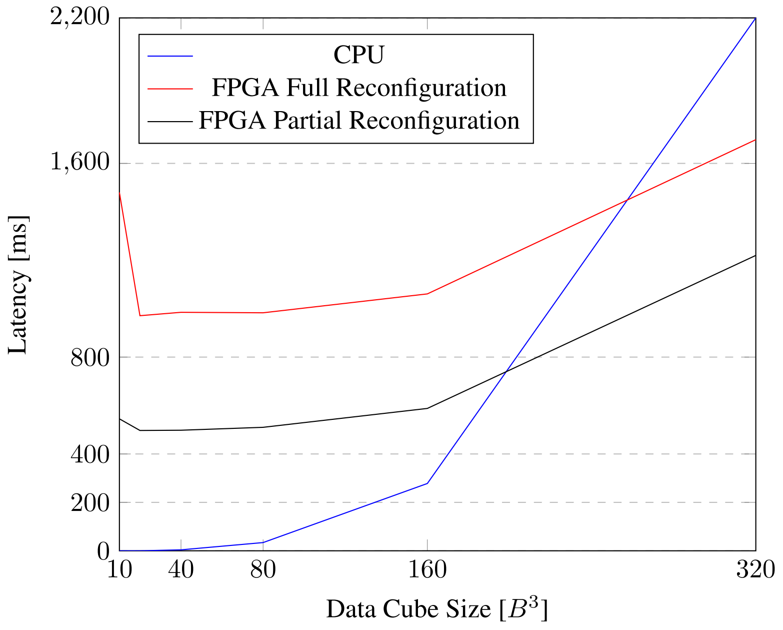


**Figure 5.9:** Comparison between software and FPGA accelerated hyper-spectral image processing.

As shown in Figure 5.9, partial reconfiguration enables faster switching between processing cores and will thus achieve lower latency compared to full reconfiguration. This offset is defined by the size of the re-configurable partition which again must be large enough to fit the largest core to be loaded onto that partition.

### 5.3.5 Error detection and recovery tests

#### Bootloader Image

To protect against the possible corruption of the bootloader image, triple redundancy was implemented. To test that this was working as intended, each of the bootloader images were sequentially made corrupt by erasing a small part (10 byte) of the file as shown in Figure 5.11. The bootloader file named BOOT.BIN consisted of a FSBL and the u-boot SSBL according to the boot information file shown in Figure 5.10.

```
the_ROM_image:
{
    [bootloader] images/linux/zynq_fsbl.elf
    [checksum=md5] images/linux/u-boot.elf
}
```

**Figure 5.10:** The bif file describing the layout the bootloader image.

The procedure for performing this test was the following:

1) build deployment images using the build_all script found in the opu-system repository.
2) Copy the resulting boot files to the SD-Card of the DUT, assure boot mode is set to QSPI (see appendix B), and turn on the power and wait for the procedure to complete, indicated by a ”`FLASHING FINISHED. YOU CAN TURN OFF THE DEVICE`” message on the UART serial terminal.
3) Switch to SD boot mode and assure that the system boots up.
4) turn off the power, change the bootmode to QSPI, and enter the U-boot terminal by hitting enter multiple times as soon as power is turned on. Once in the terminal, write the following commands to erase a portion of the bootloader:

```
sf probe 0 0 0
sf erase 0x100000 +0x10
```

5) Repeat step 3.4 with increasing offset for the data to erase, such as `0x220000` and `0x430000` as illustrated in Figure 5.11. 6) After all three bootloader files have been corrupted the device will fail to start.

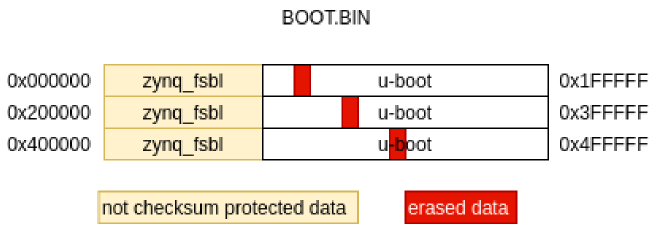


**Figure 5.11:** Overview of BOOT.BIN content and how the bootloader was corrupted to test the automatic fallback mechanism.

The test found that the md5 checksum was able to detect the corrupted bootloader and proceed to load the backup. It was however also found that corrupting data within the zynq_fsbl caused the system to fail to start or become unresponsive. This was expected as this region is not validated with a checksum.

### Bootloader environment

The SSBL (U-boot) uses an environment file to store bootloader information such as the number of attempted boots. This data is automatically checked using CRC-32 before beeing loaded by U-boot. To validate that the CRC-32 checksum is working as intended a small chunk of the env_blob file (Table 5.1) was erased by running the following commands in the u-boot terminal:

```
sf probe 0 0 0
sf erase 0x600000 +0x10
```

This erases 10 bytes of data from the U-boot environment, which causes the CRC test during booting to fail and thereby the boot environment to be reset to the original values (shown in Figure 5.12). The complete log of the test can be found in appendix L.

```
Loading Environment from SPI Flash...OK
Warning: Bootlimit (5) exceeded. Using altbootcmd.
Hit any key to stop autoboot:  0
Zynq> sf erase 0x600000 +0x10
SF: 65536 bytes @ 0x600000 Erased: OK
Zynq> reset
resetting ...
Loading Environment from SPI Flash... *** Warning - bad
 ↪  CRC, using default environment
Hit any key to stop autoboot...
```

**Figure 5.12:** Summary of the test result of the CRC-32 checksum and fallback in U-boot from appendix L

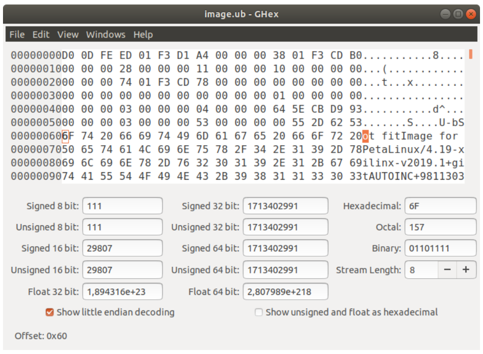
### FIT image

Each of the three components (kernel, ramdisk, and device tree) of the FIT image is protected by a SHA-1 checksum as discussed in section 5.2.1 and shown in appendix D. This test was done to validate that this functionality was working and that the bootloader was able to automatically fall back on the golden backup image. To test this the necessary files were first built by running the build_all script, copying the generated files to the SD card of the DUT, and let it automatically flash the bootloader and golden image to the correct flash devices as specified in Table 5.5 and Table 5.1. A successful flashing was confirmed by no error messages in the u-boot terminal as shown in Figure 5.15. Once this was complete the DUT was changed to boot from QSPI.

```
Flashing UBOOT
SF: Detected n25q128 with page size 256 Bytes, erase size
 ↪  64 KiB, total 16 MiB
687056 bytes read in 60 ms (10.9 MiB/s)
SF: 720896 bytes @ 0x0 Erased: OK
device 0 offset 0x0, size 0xa7bd0
SF: 687056 bytes @ 0x0 Written: OK
SF: 720896 bytes @ 0x200000 Erased: OK
device 0 offset 0x200000, size 0xa7bd0
SF: 687056 bytes @ 0x200000 Written: OK
SF: 720896 bytes @ 0x400000 Erased: OK
device 0 offset 0x400000, size 0xa7bd0
SF: 687056 bytes @ 0x400000 Written: OK
Flashing golden image
32756112 bytes read in 2075 ms (15.1 MiB/s)
32756112 bytes written
Flashing bitstream
5980026 bytes read in 394 ms (14.5 MiB/s)
5980026 bytes written
SF: Detected n25q128 with page size 256 Bytes, erase size
 ↪  64 KiB, total 16 MiB
SF: 131072 bytes @ 0x500000 Erased: OK
FLASHING FINISHED. YOU CAN TURN OFF THE DEVICE
```

**Figure 5.13:** U-Boot log indicating the successful flashing of bootloader and golden image to QSPI and eMMC.

The primary image located on the SD-Card as described in Table 5.4 was then corrupted using a hex editor to simulate a bitflip as shown in Figure 5.14. Turning on the DUT and a reading of the serial terminal (Figure 5.15) reported a bad data hash for the kernel and proceeded to load the golden image.



```
00000000D0 0D FE ED 01 F3 D1 A4 00 00 00 38 01 F3 CD B0...........8....
0000001000 00 00 28 00 00 00 11 00 00 00 10 00 00 00 00...(............
0000002000 00 00 74 01 F3 CD 78 00 00 00 00 00 00 00 00...t...x........
0000003000 00 00 00 00 00 00 00 00 00 00 01 00 00 00 00................
0000004000 00 00 03 00 00 00 04 00 00 00 64 5E CB D9 93...........d^...
0000005000 00 00 03 00 00 00 53 00 00 00 00 55 2D 62 53.......S....U-bS
000000606F 74 20 66 69 74 49 6D 61 67 65 20 66 6F 72 20ot fitImage for 
0000007050 65 74 61 4C 69 6E 75 78 2F 34 2E 31 39 2D 78PetaLinux/4.19-x
0000008069 6C 69 6E 78 2D 76 32 30 31 39 2E 31 2B 67 69ilinx-v2019.1+gi
0000009074 41 55 54 4F 49 4E 43 2B 39 38 31 31 33 30 33tAUTOINC+9811303
```

Signed 8 bit: 111
Unsigned 8 bit: 111
Signed 16 bit: 29807
Unsigned 16 bit: 29807
Float 32 bit: 1,894316e+23
Signed 32 bit: 1713402991
Unsigned 32 bit: 1713402991
Signed 64 bit: 1713402991
Unsigned 64 bit: 1713402991
Float 64 bit: 2,807989e+218
Hexadecimal: 6F
Octal: 157
Binary: 01101111
Stream Length: 8

[✓] Show little endian decoding
[ ] Show unsigned and float as hexadecimal

Offset: 0x60

**Figure 5.14:** The bit flip on the primary image was simulated by editing the file content using a hex editor.

```
## Loading kernel from FIT Image at 10000000 ...
   Using 'conf@system-top.dtb' configuration
   Verifying Hash Integrity ... OK
   Trying 'kernel@1' kernel subimage
     Description:  Linux kernel
     Type:         Kernel Image
     Compression:  uncompressed
     Data Start:   0x10000104
     Data Size:    4140776 Bytes = 3.9 MiB
     Architecture: ARM
     OS:           Linux
     Load Address: 0x00008000
     Entry Point:  0x00008000
     Hash algo:    sha1
     Hash value:
      ↪  85181abd564c701c7ffeb78b9898d9744a6dbd4c
   Verifying Hash Integrity ... sha1 error!
Bad hash value for 'hash@1' hash node in 'kernel@1' image
 ↪  node
Bad Data Hash
ERROR: cant get kernel image!
Booting from eMMC
```

**Figure 5.15:** U-Boot sucessfully detecting a corruption of the FIT image and proceeding to boot from eMMC.

**File integrity checking with MD5**

Sometimes it is useful to verify the integrity of the remote files to assure that the files have not become corrupted or have been tampered with. To test this functionality one file that was identical on both the local and remote system, and one file that had been tampered with was check-summed. The tampered file was made corrupt using the same method as shown in Figure 5.14. As shown in Figure 5.16 the hash value was identical for the two identical files and different for the tampered file as expected. The test also showed that changing the name of the file did not have any effect on the hash.

Local:

```
hypso@WS1:~/Desktop/tmp$ md5sum image.ub
c6d52f71da4a7674b795e8b204f95e4c  image.ub
```

Remote (OPU/DUT):

```
root@f82945d-primary:/media/sd-img# md5sum image.ub
c6d52f71da4a7674b795e8b204f95e4c  image.ub
root@f82945d-primary:/media/sd-img# md5sum
 ↪  image_tampered.ub
ae0d446615a6df1107461fd9aa35f8a9 image_tampered.ub
root@f82945d-primary:/media/sd-img# cp image.ub
 ↪  image_renamed.ub
root@f82945d-primary:/media/sd-img# md5sum
 ↪  image_renamed.ub
c6d52f71da4a7674b795e8b204f95e4c  image_renamed.ub
```

**Figure 5.16:** Computing md5 checksum of local and remote file to confirm identical files.

### 5.3.6 Manual recovery test

For extra security in case an undetected bug in a firmware update would cause the system to become unresponsive a method for manually forcing the system to boot the golden image was also developed. To test this the boot counter had to be incremented to 5, which would activate the alternative boot sequence in U-boot. The boot counter was incremented by manually toggling the power to the system on and off 5 times, with a short delay of 2 sec between each toggle to give time for the bootloader to successfully update the boot counter variable as depicted in Figure 5.17.

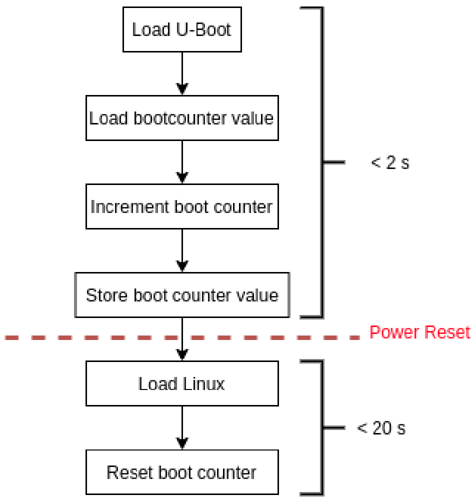


**Figure 5.17:** Illustration on when and why the power was toggled to increment the boot-counter.

A short summary of the results in U-boot can be seen in Figure 5.18. The test also showed that after the golden image had successfully booted, the bootcounter was reset to 0 again.

```
HYPSO-1 Booting. Current bootcount is 1 of 5
HYPSO-1 Booting. Current bootcount is 2 of 5
HYPSO-1 Booting. Current bootcount is 3 of 5
HYPSO-1 Booting. Current bootcount is 4 of 5
HYPSO-1 Booting. Current bootcount is 5 of 5
Warning: Bootlimit (5) exceeded. Using altbootcmd.
.... Booting into Petalinux
.... after rebooting again:
HYPSO-1 Booting. Current bootcount is 1 of 5
```

**Figure 5.18:** Summary of the U-Boot log showing the bootcounter successfully counting the number of power cycles and activating the alternative boot sequence after reaching the defined boot limit.

Chapter 6

# Summary and conclusion

This thesis presented the work done to set up a fault-tolerant and re-configurable on-board processing system for the HYPSO CubeSat. The processing system was equipped with a Zynq-7000 SoC, which was set up to run the customizable embedded Linux distribution, Petalinux. A large portion of the work documented in this thesis is associated with system-level design and development workflows for this system, with a focus on ease-of-use and portability. The implemented processing system that resulted from this work was, in addition to performance tweaks such as the implementation of a RAM-based root file system, and the extension of the virtual RAM using swap, also able to prove to support the dynamic scheduling of full and partial reconfiguration of the on-chip programmable logic and to provide a reliable framework for performing remote software and firmware updates by detecting and automatically recovering from the corruption of critical files on the processing system.

## 6.1 Future work

The implemented system successfully provides the framework for performing reliable updates and is therefore ready for deployment as soon as the overlying minimal baseline software for the file-transfer and remote access interface has been properly tested in the system test setup. Future work should therefore primarily be on the topic of continuous testing and implementation of automated continuous system integration testing, using Jenkins or similar tool for setting up an automation server. Other topics that can be an interesting area for further research is to do a more extensive file-system and SD-Card performance comparisons with a particular focus on speed and fault-tolerance and to do further research on potential applications for partial reconfiguration such as SEU scrubbing, ECC protection of system RAM, and adaptive FPGA designs for high throughput image processing.

# Bibliography

[1] Abbott, D., 2017. Linux for Embedded and Real-time Applications. Newnes. Google-Books-ID: zNomDwAAQBAJ.

[2] et. al., 2020. u-boot/u-boot. URL: https://github.com/u-boot/u-boot. original-date: 2014-11-12T13:29:02Z.

[3] Alanazi, A., Straub, J., 2018. Statistical Analysis of CubeSat Mission Failure , 8.

[4] AVNET, 2018. PicoZed™ 7Z015 / 7Z030 SOM (System-On-Module) Hardware User Guide. URL: http://zedboard.org/sites/default/files/documentations/5279-UG-PicoZed-7015-7030-V2_0.pdf.

[5] Bakken, S., Orlandic, M., Johansen, T.A., 2019. The effect of dimensionality reduction on signature-based target detection for hyperspectral remote sensing, in: CubeSats and SmallSats for Remote Sensing III, International Society for Optics and Photonics. p. 111310L.

[6] Birkeland, R., Langer, D., 2020. HYPSO-UM-004 Manual for FlatSat and LidSat.

[7] Bolchini, C., Miele, A., Santambrogio, M.D., 2007. TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs, in: 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007), pp. 87–95. doi:10.1109/DFT.2007.25. iSSN: 2377-7966.

[8] Boothby, C., 2019. TFE4580 Project Thesis - A software implementation of the ccsds 123 Issue 2 compression standard. A low-complexity lossless and near-lossless compression standard of multispectral and hyperspectral images. Technical Report. NTNU.

[9] Choi, J.O., 2015. Performance Analysis of Block Write Operation of File Systems on Linux Environment. Journal of the Korea Institute of Information and Communication Engineering 19, 136–140. URL: http://www.koreascience.or.kr/article/JAKO201506565684274.do, doi:10.6109/jkiice.2015.19.1.136. publisher: The Korea Institute of Information and Commucation Engineering.

[10] Derek J.S., R., 2003. An Introduction to Abstract Algebra. Walter de Gruyter, pp. 255-257.

[11] Dodd, P.E., Shaneyfelt, M.R., Schwank, J.R., Felix, J.A., 2010. Current and Future Challenges in Radiation Effects on CMOS Electronics. IEEE Transactions on Nuclear Science 57, 1747–1763. URL: http://ieeexplore.ieee.org/document/5550487/, doi:10.1109/TNS.2010.2042613.

[12] Fernandes, J.A., 2013. A powerful kernel image format , 38.

[13] Fjeldtvedt, J., Orlandić, M., 2019. CubeDMA – Optimizing three-dimensional DMA transfers for hyperspectral imaging applications. Microprocessors and Microsystems 65, 23–36. URL: http://www.sciencedirect.com/science/article/pii/S014193311830228X, doi:10.1016/j.micpro.2018.12.009.

[14] Folkesson, M., 2017. FIT vs legacy image format. URL: https://www.marcusfolkesson.se/blog/fit-vs-legacy-image-format/.

[15] Gjersvik, A., 2020. HYPSO-DR-011: BoB V3 Design Report.

[16] Hamming, R.W., 1950. Error detecting and error correcting codes. The Bell System Technical Journal 29, 147–160. doi:10.1002/j.1538-7305.1950.tb00463.x.

[17] Hov, M., 2019. Design and Implementation of Hardware and Software Interfaces for a Hyperspectral Payload in a Small URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2625750. accepted: 2019-10-31T15:12:35Z Publisher: NTNU.

[18] Hudec, J., 2012. linux - The difference between initrd and initramfs. URL: https://stackoverflow.com/a/10604667. library Catalog: stackoverflow.com.

[19] Iwamatsu, N., Denk, W., 2020. mkimage - Generate image for U-Boot. URL: https://linux.die.net/man/1/mkimage.

[20] Kao, C., 2005. Benefits of Partial Reconfiguration , 3.

[21] Koch, H.J., 2006. The Userspace I/O HOWTO — The Linux Kernel documentation. URL: https://www.kernel.org/doc/html/v4.12/driver-api/uio-howto.html.

[22] Koopman, P., 2018. Best CRC Polynomials. URL: http://users.ece.cmu.edu/~koopman/crc/.

[23] Kornberg, J.A., 2020. Time Synchronization of Hyperspectral Image Capture on board a Nanosatellite. Ph.D. thesis. NTNU.

[24] Kulu, E., 2020. Nanosats Database. URL: https://www.nanosats.eu/index.html. library Catalog: www.nanosats.eu.

[25] Leppinen, H., 2017. Current use of linux in spacecraft flight software. IEEE Aerospace and Electronic Systems Magazine 32, 4–13. doi:10.1109/MAES.2017.160182. conference Name: IEEE Aerospace and Electronic Systems Magazine.

[26] Meyer-Baese, U., 2007. Digital Signal Processing with Field Programmable Gate Arrays. Springer Science & Business Media. Google-Books-ID: wzYuOF6HFX0C.

[27] Montzka, M., 2020. Fast Spectrograph Corrections on Programmable Logic. Ph.D. thesis. NTNU.

[28] O'Neal, T., 2019. U-Boot Images - Xilinx Wiki - Confluence. URL: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842374/U-Boot+Images.

[29] Orlandic, M., Fjeldtvedt, J.A., Johansen, T.A., 2019. A Parallel FPGA Implementation of the CCSDS-123 Compression Algorithm. 11 URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2595160, doi:https://doi.org/10.3390/rs11060673. accepted: 2019-04-24T07:24:44Z Publisher: MDPI.

[30] Orlandić, M., Svarstad, K., 2019. An adaptive high-throughput edge detection filtering system using dynamic partial reconfiguration | SpringerLink. URL: https://link.springer.com/article/10.1007/s11554-018-0753-4.

[31] Paul Horowitz, Winfield, H., 2016. The Art of Electronics. 3 ed. URL: https://books.google.no/books?id=LAiWPwAACAAJ&dq=0521809266,9780521809269&hl=no&sa=X&ved=0ahUKEwih8JGnq9jnAhVywqYKHeXtCdQQ6AEIKTAA.

[32] Petazzoni, T., 2012. Embedded Linux Conference Europe 2012 - Your new ARM SoC Linux support check-list! URL: https://www.elinux.org/images/a/ad/Arm-soc-checklist.pdf.

[33] Petazzoni, T., 2013. Embedded Linux Conference Europe - device tree for dummies. URL: http://events17.linuxfoundation.org/sites/events/files/slides/petazzoni-device-tree-dummies.pdf.

[34] Peterson, W.W., Brown, D.T., 1961. Cyclic Codes for Error Detection. Proceedings of the IRE 49, 228–235. doi:10.1109/JRPROC.1961.287814. conference Name: Proceedings of the IRE.

[35] Preshing, J., 2011. Hash Collision Probabilities. URL: https://preshing.com/20110504/hash-collision-probabilities/.

[36] Puig-Suari, J., 2018. 6U CubeSat Design Specification Revision 1.0. URL: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/5b75dfcd70a6adbee5908fd9/1534451664215/6U_CDS_2018-06-07_rev_1.0.pdf.

[37] Razzaghi, E., 2012. Design and Qualification of On-Board Computer for Aalto-1 CubeSat. Ph.D. thesis. Aalto University.

[38] Rifenbark, S., 2019. Yocto Project Overview and Concepts Manual (Revision 3.0.1). URL: https://www.yoctoproject.org/docs/3.0.2/overview-manual/overview-manual.html.

[39] Ritchie, D., Thompson, K., 1978. The UNIX Time-Sharing System. Bell System Technical Journal. URL: http://archive.org/details/bstj57-6-1905.

[40] Rössler, P., Höller, R., 2020. Programmable logic devices – key components for today's and tomorrow's electronic-based systems. e & i Elektrotechnik und Informationstechnik 137, 45–51. URL: http://link.springer.com/10.1007/s00502-019-00781-w, doi:10.1007/s00502-019-00781-w.

[41] Shet, R., 2020. Programmable Logic Devices - A summary of all types of PLDs. URL: https://www.technobyte.org/programmable-logic-devices/. library Catalog: www.technobyte.org Section: Digital Electronics and Digital Logic Design.

[42] Teubner, J., Woods, L., 2013. Data Processing on FPGAs. Morgan & Claypool Publishers. URL: https://books.google.no/books?id=qbJdAQAAQBAJ&printsec=frontcover&hl=no#v=onepage&q&f=false.

[43] Volkova, J., Kneižys, E., Kalabuckas, E., 2018. M6P Platform/Payload Interface Control Document - Rev. NA-IC6P-001/1.

[44] Xilinx, 2019. UG1157: PetaLinux Tools Documentation: Command Line Reference Guide.

[45] Xilinx, 2020. Xilinx/linux-xlnx. URL: https://github.com/Xilinx/linux-xlnx. original-date: 2013-03-19T22:15:21Z.

[46] Xilinx Inc., 2019. PetaLinux Tools Documentation: Reference Guide.

[47] Šimek, M., 2019. Release xilinx-v2019.1 · Xilinx/linux-xlnx. URL: https://github.com/Xilinx/linux-xlnx/releases/tag/xilinx-v2019.1.

# Appendices

Appendix A

# Partial Reconfiguration of Programmable Logic in Linux

# Partial Reconfiguration of Programmable Logic in Linux

Joar Andreas Gjersund

**Abstract**

This document is intended as a guide for getting started with dynamic partial reconfiguration of a Zynq-7000 FPGA running Embedded Linux. This document goes through all the steps from setting up the processing system in Vivado, to defining reconfigurable partions and generating bitstreams, it also gives a short guide on how to how to set up the Embedded Linux distribution to enable partial reconfiguration.

## 1 Introduction

Partial Reconfiguration enables us to increase the utilization of the resources available on the FPGA by taking advantage of the fact that often not all programmable logic is active at the same time. Partial Reconfiguration can thus reprogram these regions of the fpga, while other regions are running, allowing time multiplexing the available resources. Applications such as SEU Scrubbing for correcting configuration cell upsets and real time dynamic image processing pipe-lining for quickly togling between various image processing steps such as compression, correction, and filtering is just the tip of the iceberg of what possibilities are realizable with partial reconfiguration.

In this report I will go through the simplest example possible intended as a proof of concept of how this can be actually implemented such that is as easy as possible for the reader to follow. In this example I will build a simple one-gate logic reconfigurable module that reads data from a DMA interface, inverts the data, and returns the data to the DMA see fig. **??**. I will also build another module using the same reconfigurable partion, that simply forwards the data without performing any manipulation. I will then show how these two modules can be reconfigured at runtime. All sources including a simple app to test the features will be made available on SmallSatLab Github repository opu-system under the branch bitmap_testing. For access to this repository, please contact the SmallSatLab at NTNU

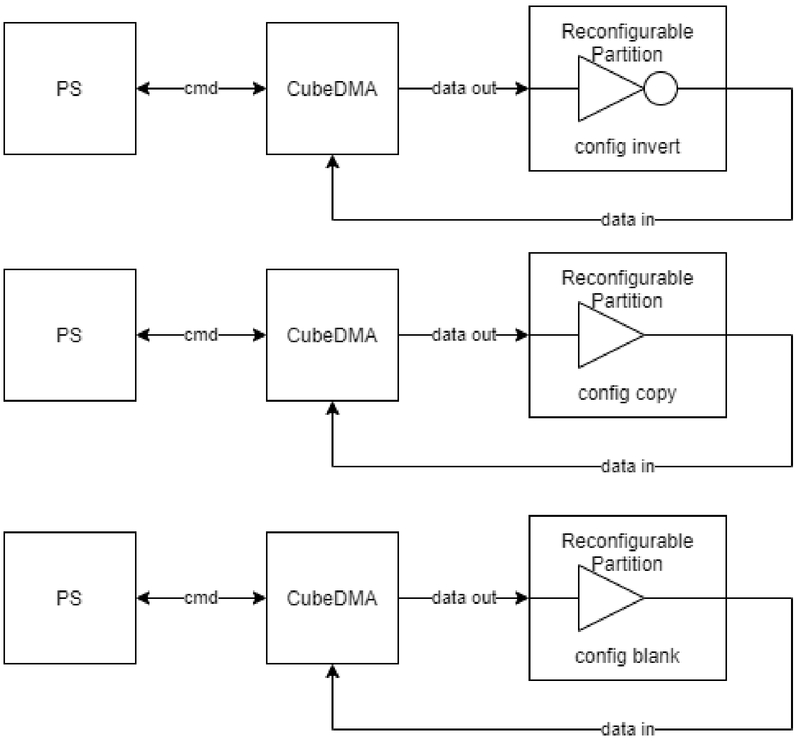


Figure 1: The three different configurations for this example

## 2 Hardware and software Requirements

Zynq-7000 SoC. Petalinux 2019.1 Vivado 2019.1 Docker

## 3 Building the embedded Linux operating System

follow the README guide found in repository.

## 4 Making a reconfigurable module

### 4.1 Building The Static Part

Open the vivado project file. make a new source file. verilog.

```
module test(
  s_axis_tdata,
  m_axis_tdata,
    );
```

right click in block diagram and add module. hook up the module

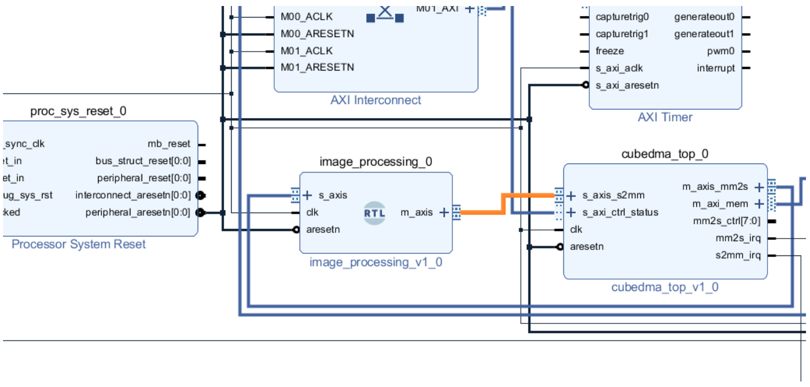


Figure 2: Caption

In the flow navigator press Run Synthesis, when done click cancel and exit the project. Navigate the the synthesis output folder and make sure that a dcp checkpoint file has been generated.

## 4.2 Building The Reconfigurable Modules

Navigate to the source folder of the reconfigurable modules and synthesize the modules using the following tcl script (make sure the part matches the hardware and projects part):

```
read_verilog test_copy/test_copy.v
synth_design -mode out_of_context -flatten_hierarchy rebuilt
↪  -top test -part xc7z030sbg485-1
write_checkpoint Synth/reconfig_modules/test_copy_synth.dcp
↪  -force
close_design
close_project
read_verilog  test_invert/test_invert.v
synth_design -mode out_of_context -flatten_hierarchy rebuilt
↪  -top test -part xc7z030sbg485-1
write_checkpoint Synth/reconfig_modules/test_invert_synth.dcp
↪  -force
close_design
```

```
close_project
```

The reconfigurable modules consists of the following:
test_copy.v:

```
module test(
    input [63:0] s_axis_data,
    output [63:0] m_axis_data
    );
    assign m_axis_data = s_axis_data;
endmodule
```

test_invert.v:

```
module test(
    input [63:0] s_axis_data,
    output [63:0] m_axis_data
    );
    assign m_axis_data = ~s_axis_data;
endmodule
```

### 4.3 Drawing The Reconfigurable Partition Layout

The next step is to assign a block on the fpga for the modules to be loaded to, also called a reconfigurable partition block or simply pblock.

Run the following tcl commands to load the partion planning tool:

```
open_checkpoint Synth/Static/System_wrapper.dcp
read_checkpoint -cell System_i/test_0/inst
 ↪  Synth/reconfig_modules/test_copy_synth.dcp
set_property HD.RECONFIGURABLE 1 [get_cells
 ↪  System_i/test_0/inst]
write_checkpoint Checkpoint/top_link_add.dcp -force
```

find the module and draw the pblock. (It is also possible to use a xdc constraint file that sets the requirements of the pblock and automatically draws it). Make sure to include enough resources for all reconfigurable modules to be included in the partition. Once done, run the DRC Report tool with partial reconfiguration enabled to make sure everything looks okey. If it fails, you might have to enable snapping of the pblock. This is done by selecting the pBlock, and toggle the SNAPPING_MODE to ON, found at the bottom of the pBlock Properties.

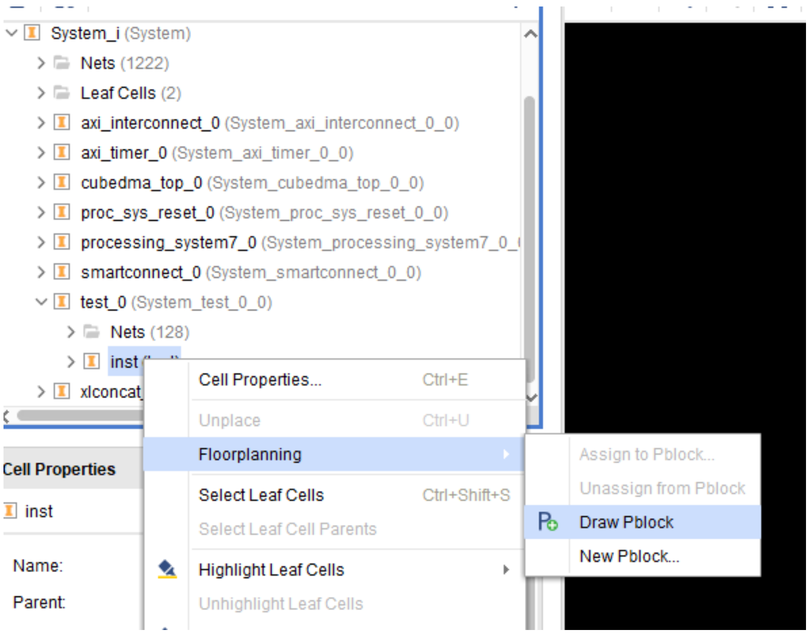

System_i (System)
Nets (1222)
Leaf Cells (2)
axi_interconnect_0 (System_axi_interconnect_0_0)
axi_timer_0 (System_axi_timer_0_0)
cubedma_top_0 (System_cubedma_top_0_0)
proc_sys_reset_0 (System_proc_sys_reset_0_0)
processing_system7_0 (System_processing_system7_0_
smartconnect_0 (System_smartconnect_0_0)
test_0 (System_test_0_0)
Nets (128)
inst
xlconcat
Cell Properties
inst
Name:
Parent:
Cell Properties... Ctrl+E
Unplace Ctrl+U
Floorplanning
Select Leaf Cells Ctrl+Shift+S
Select Leaf Cell Parents
Highlight Leaf Cells
Unhighlight Leaf Cells
Assign to Pblock...
Unassign from Pblock
Draw Pblock
New Pblock...

Figure 3: Caption

Once done. Run the following script to optimize, place, and route design:

```
opt_design
place_design
route_design
write_checkpoint Implement/config_copy_top_route_design.dcp
 ↪  -force
write_checkpoint -force -cell System_i/test_0/inst
 ↪  Checkpoint/test0_copy_route_design.dcp
update_design -cell System_i/test_0/inst -black_box
lock_design -level routing
write_checkpoint -force Checkpoint/static_route_design.dcp
```

The first module is now done. For any successive modules that we would like to add to the partition the procedure is a bit faster as the pBlock is already defined. Simply run the following tcl commands:

```
read_checkpoint -cell System_i/test_0/inst
 ↪  Synth/reconfig_modules/test_invert_synth.dcp
opt_design
```

```
place_design
route_design
write_checkpoint Implement/config_invert_top_route_design.dcp
↪  -force
write_checkpoint -force -cell System_i/test_0/inst
↪  Checkpoint/test0_invert_route_design.dcp
close_project
```

Do this for all reconfigurable modules. When done we need to make a blanking module, which is used to when transistion between modules. This can be done by running the following tcl commands:

```
open_checkpoint Checkpoint/static_route_design.dcp
update_design -buffer_ports -cell System_i/test_0/inst
place_design
route_design
write_checkpoint -force
↪  Implement/config_blank_top_route_design.dcp
close_project
```

Finnally, we should run the following command to verify that all reconfigurable modules and partion is configured correctly:

```
pr_verify -initial Implement/config_copy_top_route_design.dcp
↪  -additional {Implement/config_invert_top_route_design.dcp
↪  Implement/config_blank_top_route_design.dcp}
close_project
```

## 4.4 Export partial bitstreams

To export the partial bitstreams run the following tcl command:

```
open_checkpoint Implement/config_copy_top_route_design.dcp
write_bitstream -file Bitstreams/test_copy.bit -force
close_project
open_checkpoint Implement/config_invert_top_route_design.dcp
write_bitstream -file Bitstreams/test_invert.bit -force
close_project
open_checkpoint Implement/config_blank_top_route_design.dcp
write_bitstream -file Bitstreams/test_blank.bit -force
close_project
```

The bitstreams are now exported in bit format. Note that there both a partial and a full bitstream is exported for each module, the full bitstream

also includes the static part and one of these must be loaded onto the fpga before partial bitstreams can be used.
To be able to load bitstreams onto the fpga in linux, the files must be in bin format. This is achieved by running the bootgen utillity found in the petalinux sdk. For each of the bitstreams do the following, replacing bitstreamname with the name of the actual bitstream to convert to bin.:

make an empty file called bitstream.bif with the following content:

```
all:
{
        bitstreamname.bit /* Bitstream file name */
}
```

run this command in the same folder as the bif file and bitsteram file. Make sure you have Petalinux SDK installed.

```
bootgen -image bitstream.bif -arch zynq -process_bitstream
↪  bin
```

The bitstreams in correct format will then be generated and saved to the same folder.

## 4.5 Loading bitstream onto the FPGA in linux

For loading the bitstream onto the FPGA in linux we use a driver via the sysfs interface. Make sure the bitstream bin files are copied to the sd card along with the bootloader and linux image and boot up the system. Once booted up the bitstreams can be loaded by running the following commands via the terminal:

for full reconfiguration run:

```
echo 0 > /sys/class/fpga_manager/fpga0/flags
mkdir -p /lib/firmware
cp /media/bitstream_full.bit.bin /lib/firmware/
echo bitstream_full.bit.bin >
↪  /sys/class/fpga_manager/fpga0/firmware
```

for partial reconfiguration run:

```
echo 1 > /sys/class/fpga_manager/fpga0/flags
mkdir -p /lib/firmware
cp /media/bitstream_partial.bit.bin /lib/firmware/
echo bitstream_partial.bit.bin >
↪  /sys/class/fpga_manager/fpga0/firmware
```

To make this process easier, a tool called fpgautil can also be installed on the system. The source file can be found in the reference files. loading bitstream with this tool works like this:

```
root@38787f2-primary:~# fpgautil

fpgautil: FPGA Utility for Loading/reading PL Configuration

Usage:  fpgautil -b <bin file path> -o <dtbo file path>

Options: -b <binfile>           (Bin file path)
         -o <dtbofile>          (DTBO file path)
         -f <flags>             Optional: <Bitstream type
          ↪  flags>
                                   f := <Full | Partial >

Examples:
(Load Full bitstream using Overlay)
fpgautil -b top.bit.bin -o can.dtbo
(Load Partial bitstream through the sysfs interface)
fpgautil -b top.bit.bin -f Partial
```

### 4.6 Verifcation Testing

The design can be validated using the tool bitmap-test as follows:

```
root@b43e570-primary:~# fpgautil -b
 ↪  /media/sd-img/bitstreams/bitstreams/test_blank.bit.bin
fpga_manager fpga0: writing test_blank.bit.bin to Xilinx Zynq
 ↪  FPGA Manager
Time taken to load BIN is 113.000000 Milli Seconds
BIN FILE loaded through FPGA manager successfully
root@b43e570-primary:~# bitmap-test
Iterating
Configure cubeDMA
Starting transfer
 ding
received length 2000

 Sent:

0101010101
0101010101
```

```
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
 Received:

0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000
0000000000Success
root@b43e570-primary:~# fpgautil -b
↪  /media/sd-img/bitstreams/bitstreams/test_copy_pblock_inst_partial.bit.bin
↪  -f Partial
fpga_manager fpga0: writing
↪  test_copy_pblock_inst_partial.bit.bin to Xilinx Zynq FPGA
↪  Manager
Time taken to load BIN is 18.000000 Milli Seconds
BIN FILE loaded through FPGA manager successfully
root@b43e570-primary:~# bitmap-test
Iterating
Configure cubeDMA
Starting transfer
 ding
received length 2000

 Sent:

0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
```

```
0101010101
0101010101
0101010101
0101010101
 Received:

0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101Success
root@b43e570-primary:~# fpgautil -b
↪  /media/sd-img/bitstreams/bitstreams/test_invert_pblock_inst_partial.bit.bin
↪  -f Partial
fpga_manager fpga0: writing
↪  test_invert_pblock_inst_partial.bit.bin to Xilinx Zynq
↪  FPGA Manager
Time taken to load BIN is 18.000000 Milli Seconds
BIN FILE loaded through FPGA manager successfully
root@b43e570-primary:~# bitmap-test
Iterating
Configure cubeDMA
Starting transfer
 ding
received length 2000

 Sent:

0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
0101010101
```

```
Received:

6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534
6553565534655356553465535655346553565534655356553465535655346553565534Success
```

## 5 Conclusion

The example gone through in this report managed to reduce the time needed to reconfigure the fpga from 113 ms for a full reconfiguration, to only 18 ms for the partial reconfiguration. The work flow for generating partial reconfiguration modules was heavly inspired by [Kajekar(2020)].

## References

[Kajekar(2020)] Kajekar, N.N., 2020. Tutorial on Partial Reconfiguration of Image Processing Blocks using VIvado and SDK. Technical Report. University of New Mexico School Of Engineering - Department of Electrical and Computer Engineering. URL: http://ivpcl.unm.edu/ivpclpages/Research/drastic/PRWebPage/PR$_{S}$*ub.php*.

Appendix B

# HYPSO-DSW-008: Documentation for The Petalinux Bootloader and the Generation of system Images for Performing Software Updates

# Documentation for The Petalinux Bootloader and the generation of system images for performing software updates

HYPSO-DSW-008

**Prepared by:** HYPSO Project Team

**Reference:** HYPSO-DSW-008
**Revision:** 1
**Date of issue:** 27.11.19
**Status:** Issued
**Document Type:** 02.12.2019

## Table Of Contents

Table 1: Table of Changes

| Rev. | Summary of Changes | Author(s) | Effective Date |
|---|---|---|---|
| *1* | *First issue* | *Joar Andreas Gjersund* | *27.11.2019* |
| *2* | *updated test procedure.* | *Joar Andreas Gjersund* | *12.12.2019* |

# 1 Overview

The HYPSO Mission will primarily be a science-oriented technology demonstrator. It will enable low-cost & high-performance hyperspectral imaging and autonomous onboard processing that fulfill science requirements in ocean color remote sensing and oceanography. NTNU SmallSat is prospected to be the first SmallSat developed at NTNU with launch planned for Q4 2020 followed by a second mission later. Furthermore, the vision of a constellation of remote-sensing focused SmallSat will constitute a space-asset platform added to the multi-agent architecture of UAVs, USVs, AUVs, and buoys that have similar ocean characterization objectives.

## 1.1 Purpose

The purpose of the Petalinux bootloader is to load the Linux kernel and root file system (system image) from flash memory to RAM, perform integrity checks to make sure data is not corrupted, and finally boot up Linux. The bootloader should include some level of redundancy, so that if the system image becomes corrupted it should try to load a backup image from another memory device.

## 1.2 Scope

This document covers the requirements for the booting procedure, an overview of the commands that is automatically executed in u-boot during booting and where these commands are defined, how to build the system images with Docker, and finally the process of deploying the system to the Zedboard and the Picozed. This document also proposes a testing plan for validating that the system behaves according to the requirements.

## 1.4 Referenced Documents

The documents listed have been used as a reference in the creation of this document.

Table 2: Referenced Documents

| ID | Author | Title |
|---|---|---|
| UG1144 | Xilinx | Petalinux Tools Documentation |
| | Xilinx | Zedboard HW User guide |
| [RD03] | | |
| HYPSO-DSW-004 | *Marion VRIGNAUD* | Using HYPSO SW for  Zedboard |

| | | |
|---|---|---|
| | *Xilinx* | PicoZed Datasheet |

## 2 Description of opu-system

The opu-system repository is used to generate the necessary files to boot up Petalinux according to the requirements mentioned in 3.1. The repository consists of a set of scripts for setting up the proper work environment via Docker, and to automatically generate system images containing the Petalinux kernel, root file system and all the necessary software components and dependencies for the payload. The main purpose of the repository is to make and document the process of generating bootable files according to requirements and to make this process easy to replicate and configure in the future. An overview of the booting procedure can be seen in fig. 1. The two system states (green and red) is a seperate OS image (kernel and root file system), making it possible to tailor permissions, and what software components to automatically run. The green state, represented by the file image.ub can be updated (e.g. rewritten) with a new version when performing a software update. The red state, is considered a golden image and serves as a backup in case of a failed software update or corrupt primary image. This image (image_golden.ub) should under no circumstances be edited or replaced during a software update and preferably reside on read-only-memory. The bootloader is configured to use the initramfs root file system, which means that the root file system will be loaded to the volatile RAM upon boot. This enables all changes made to the rootfs to be reset during a reboot.

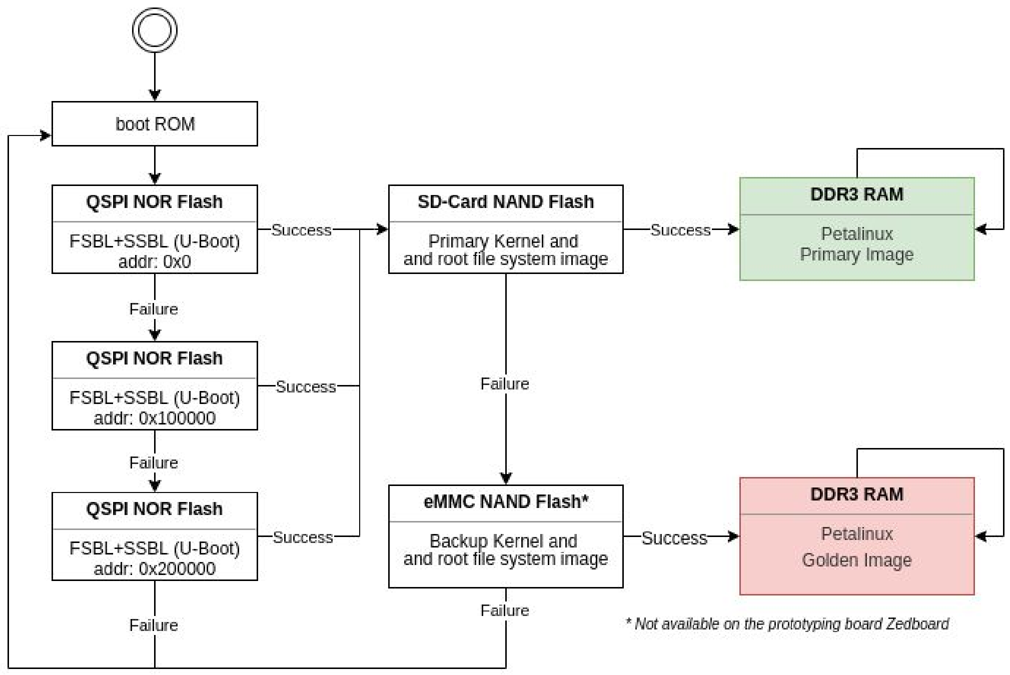

boot ROM
QSPI NOR Flash
FSBL+SSBL (U-Boot) addr: 0x0
QSPI NOR Flash
FSBL+SSBL (U-Boot) addr: 0x100000
QSPI NOR Flash
FSBL+SSBL (U-Boot) addr: 0x200000
Success
Failure
SD-Card NAND Flash
Primary Kernel and and root file system image
eMMC NAND Flash*
Backup Kernel and and root file system image
DDR3 RAM
Petalinux Primary Image
DDR3 RAM
Petalinux Golden Image
* Not available on the prototyping board Zedboard

Fig. 1: Booting procedure.

## 2.1 Interfaces with other modules

### 2.1.1 Inputs

This module uses the latest release of the hypso-sw repository. This release should be structured as it would on the target root file system and compressed to a .tar.xz file. This file will then automatically be extracted and merged with the root file system upon boot. Use the prerelease 0.3 in the hypso-sw repository as an example.

### 2.1.2 Outputs

A successful build should generate the following boot-files specified in this table:

| Location/Filename | Description |
|---|---|
| opu-system/petalinux/projects/bootfiles/<name>/BOOT_QSPI.BIN | Bootloader: FSBL+SSBL (UBOOT) |

| | |
|---|---|
| opu-system/petalinux/projects/bootfiles/<name>/BOOT.BIN | For flashing bootloader to QSPI. (will automatically run when boot mode is sd) |
| opu-system/petalinux/projects/bootfiles/<name>/Image.ub | Primary Image file |
| opu-system/petalinux/projects/bootfiles/<name>/Image_golden.ub | Golden Image file |

## 2.2 How to generate boot files (bootloader and system images)

- Requirements: Linux, Docker, git

1) Clone the opu-system repository from GitHub in the home directory in linux.
2) Download petalinux-v2019.1-final-installer.run to the directory opu-system/Docker
3) Run the command `sudo ./setup-petalinux-docker` inside the directory opu-system/Docker.
4) Run the command `sudo ./run-petalinux-docker.` If repository was not cloned in the home directory, change the script to reflect the location of the repository.
5)
    a) Run the command `./makeproject <project-name> -picozed -all` to generate all boot files for **picozed**. *
    b) Run the command `./makeproject <project-name> -all` to generate all boot files for **zedboard**.*

    * Exclude the -all flag to only generate the primary image file
6) Wait for a while…. Grab a coffee, yes, it will take quite some time. (around 30 min)
7) When done, you can find all the necessary files for the system inside opu-system/petalinux/projects/bootfiles/<project-name>

## 2.3 How to deploy

- Requirements: Zedboard or Picozed with PicoBOB, boot files (see 2.2.), SD-Card.

1) Format the SD-Card with filesystem Ext4.
2) Copy the file BOOT.BIN, BOOT_QSPI.BIN, image.ub and image_golden.ub located in the bootfiles folder to the SD-Card.
3) Insert the SD-Card into the SD-Card slot on the board, make sure the boot mode pin straps on the Zedboard or Picozed are set to SD-Card (see fig. 2 and fig.3), and turn on the power.

4) Wait for 15 minutes, then turn off power and change the boot mode pin straps to QSPI (see fig. 2 and fig.3). The file BOOT_QSPI.BIN and image_golden.ub can now be deleted from the SD-Card. (All files except image.ub have now been flashed to other memory devices according to the procedure described in fig. 1)
5) Done.

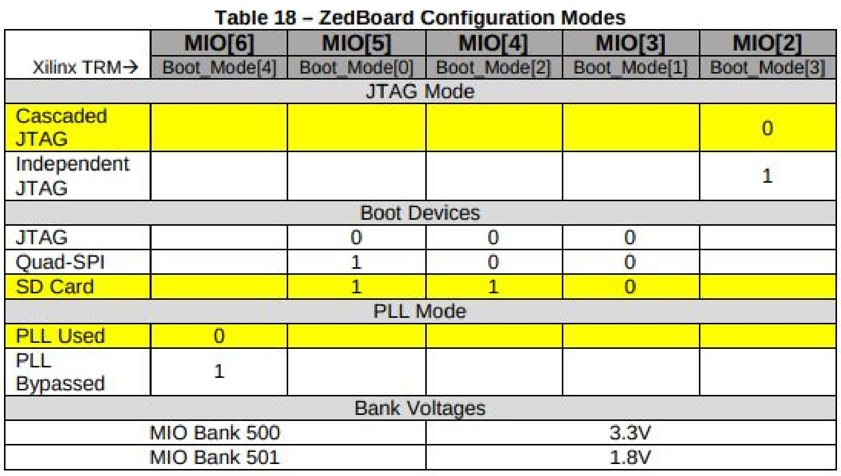
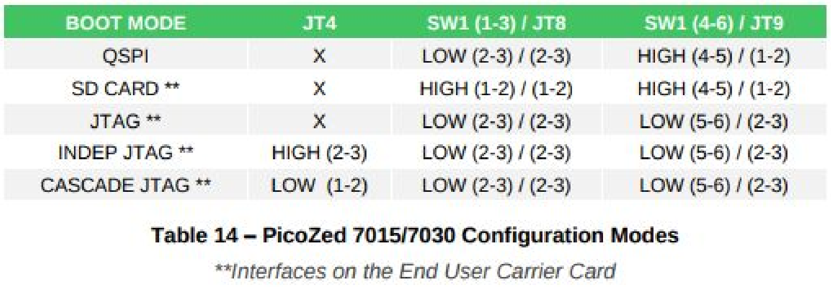


**Table 18 – ZedBoard Configuration Modes**

| | MIO[6] | MIO[5] | MIO[4] | MIO[3] | MIO[2] |
|---|---|---|---|---|---|
| Xilinx TRM→ | Boot_Mode[4] | Boot_Mode[0] | Boot_Mode[2] | Boot_Mode[1] | Boot_Mode[3] |
| JTAG Mode | | | | | |
| Cascaded JTAG | | | | | 0 |
| Independent JTAG | | | | | 1 |
| Boot Devices | | | | | |
| JTAG | | 0 | 0 | 0 | |
| Quad-SPI | | 1 | 0 | 0 | |
| SD Card | | 1 | 1 | 0 | |
| PLL Mode | | | | | |
| PLL Used | 0 | | | | |
| PLL Bypassed | 1 | | | | |
| Bank Voltages | | | | | |
| MIO Bank 500 | | | 3.3V | | |
| MIO Bank 501 | | | 1.8V | | |

Fig. 2 : Boot mode pin strap settings for Zedboard



| BOOT MODE | JT4 | SW1 (1-3) / JT8 | SW1 (4-6) / JT9 |
|---|---|---|---|
| QSPI | X | LOW (2-3) / (2-3) | HIGH (4-5) / (1-2) |
| SD CARD ** | X | HIGH (1-2) / (1-2) | HIGH (4-5) / (1-2) |
| JTAG ** | X | LOW (2-3) / (2-3) | LOW (5-6) / (2-3) |
| INDEP JTAG ** | HIGH (2-3) | LOW (2-3) / (2-3) | LOW (5-6) / (2-3) |
| CASCADE JTAG ** | LOW (1-2) | LOW (2-3) / (2-3) | LOW (5-6) / (2-3) |

**Table 14 – PicoZed 7015/7030 Configuration Modes**

***Interfaces on the End User Carrier Card*

Fig 3: Boot mode pin strap settings for Picozed

## 2.4 How to add software and configure startup script

● Software components, libraries, and applications can be added to the compressed folder software.tar.xz in the projects/software directory. All these files will automatically be extracted to the petalinux root file system upon boot.

- A startup script, located in the projects folder under the name `linux_startup_script.sh` will automatically be executed when the booting process has finished. This script is responsible of starting opu-services and mounting flash memory devices. Note that opu-services must be the last program to run in the script  since this never exit.

## 2.4 How to perform a software update

1) Copy and replace projects/software/software.tar.xz with the new software modules and libraries and optionally update the startup script as described in 2.4.
2) Optional: Configure the makeproject script.
3) Generate an OS image as described in 2.2 (exclude the -all flag)
4) Replace the newly generated image.ub file with the image.ub file present in media/sd on the satellite using the hypso-cli software.
5) Perform a reboot. If system is in green state the software update was successful.

# 3 Testplan for the Petalinux bootloader

## 3.1 Requirements

| ID | Refines | Refined by | Definition |
|---|---|---|---|
| BL-5-10 | | SYS.PA.100 | The system shall still function in the event of a corrupt or inaccessible primary system image. |
| BL-5-20 | | SBUS.3.17 | The primary system image shall be possible to update |
| BL-5-30 | | HSI.4.110<br>HSI.4.110 | The booting procedure shall be fully automated . |
| BL-5-40 | | OPU.4.80 | The system image shall be loaded to volatile memory during booting so that the system is restored to its initial state after a reboot. |
| BL-5-50 | | HSI.4.110 | The worst case execution time of the complete booting procedure should be no more than 60 (TBC) s |

## 3.2 Test Procedure

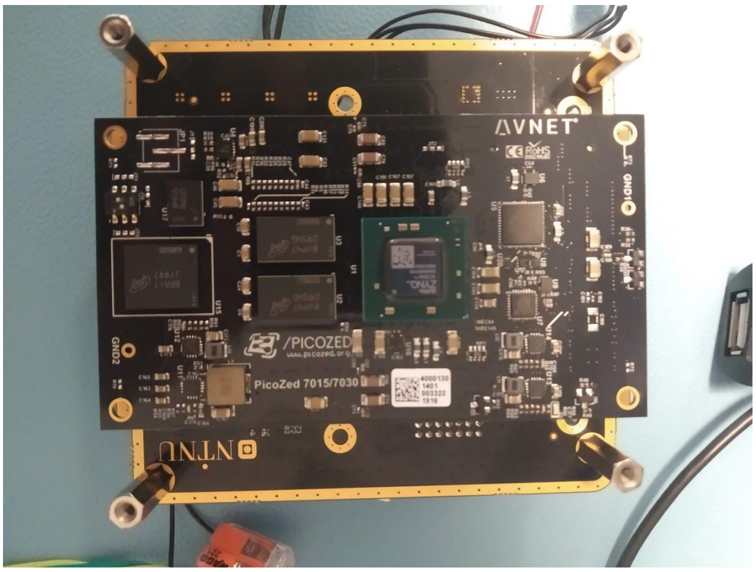
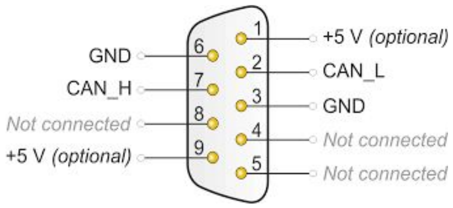
The following procedures are used to verify that the requirements in 3.1 are met. A program that guides the tester throughout the test procedure and automatically verify if a test pass can be found in opu-systems/petalinux/verification and can be run in linux by writing the command sudo `./checkkall <path-to-hypso-sw-build-directory>`.

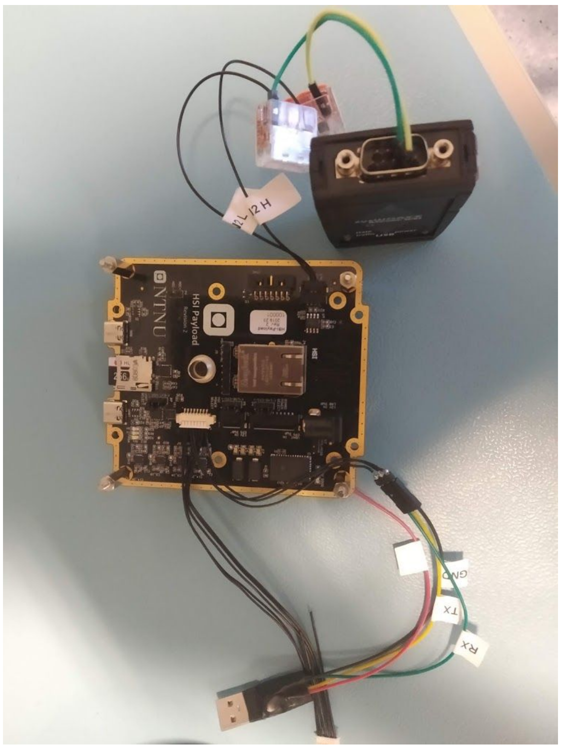
**Prerequisites**

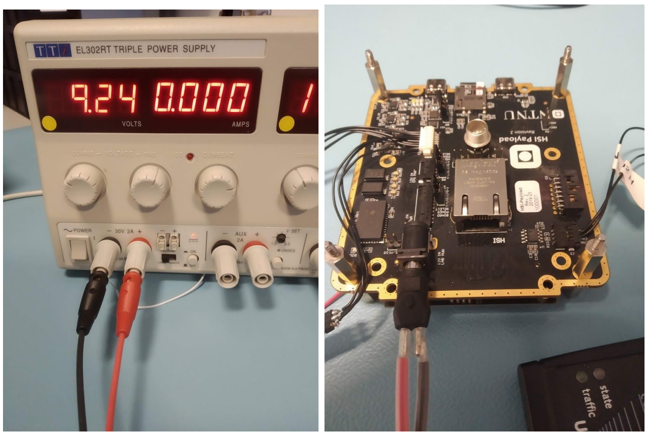
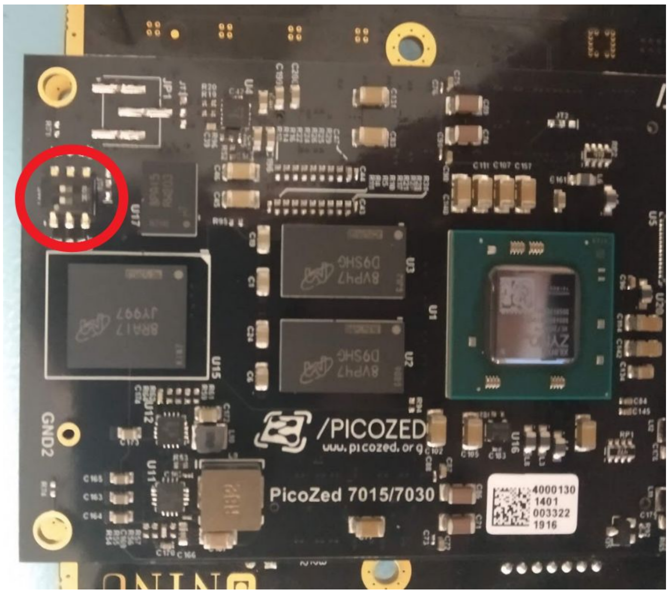
- A computer with hypso-cli readily set up. (Including necessary CAN-usb converter)
  - See HYPSO-DSW-004 for instruction on how to set up the hypso-cli on the computer.
- PicoBOB (DUT)
- Micro SD card.

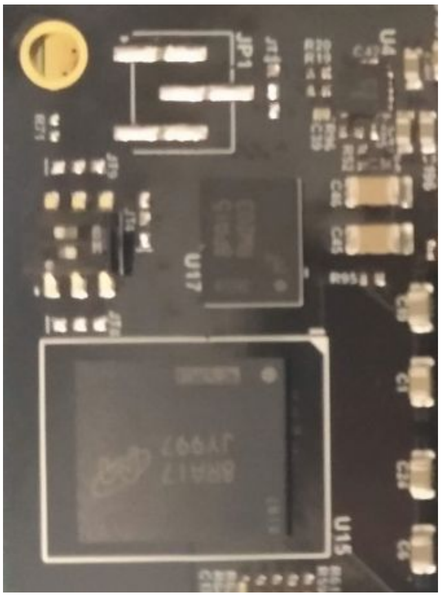
**Hardware Setup and Test procedure.**
Note: Remember to follow the labs guidelines/rules regarding ESD protection and clean-room regulations.
This section explains how to set up the DUT (device under test) and gives a more in-depth explanation of what is performed “under-the-hood” by the test script.

| | |
|---|---|
| Find a clean (preferably unopened) picozed board. | The picozed used for this test was the 7Z030. (PicoZed Datasheet) If no factory/clean picozed board is available, make sure that the u-boot environment is erased before procedure. Where the u-boot environment is placed in memory depends on the previous bootloaders configurations, but should by default be on partition 1 of the QSPI. In that case, run the following command in the petalinux command line after the picozed as booted up: `$ flash_eraseall -j /dev/mtd1` |

| | |
|---|---|
| Connect the picozed to a breakout board to make a PicoBoB. |  The breakout board used for this test was V1R2. |
| Connect the CAN USB converter to the PicoBoB CAN bus 2 connector |   GND<br>CAN_H<br>Not connected<br>+5 V (optional)<br>1 +5 V (optional)<br>2 CAN_L<br>3 GND<br>4 Not connected<br>5 Not connected<br>6<br>7<br>8<br>9 The CAN USB converter used for this test was systec CAN (IEC 61131-3) |

| | |
|---|---|
| |  The CAN USB converter pins can be connected directly to the PicoBoB CAN bus 2 connecter. The bottom connection is used for serial UART and is not needed for the test. |
| Connect the power supply to the PicoBoB | Connect a power supply ( 5.9-14.5 V) to the power connector on the PicoBoB. |

| | |
|---|---|
| |  An inbuilt buck-converter in the PicoBoB will fine-adjust the voltage for us. For this test we kept it at around 9V. |
| Set the boot mode pin strap switches to SD-Card (according to fig. 3) |  The boot mode pin straps switched all the way towards the PCB border indicate a SD-Card boot. For a mint picozed board there might be a film covering this switch that must be peeled before it can be configured. |
| Find a fresh Micro-SD card and make sure it is | Generate all necessary files using the procedure described in Chapter 2.2. And copy them to the Micro SD-Card. If more than one partition is present, either merge all partitions or copy the files to the first partition. |

| | |
|---|---|
| formatted to FAT/MS-DOS. (Should be standard formatting for most SD-Cards). | |
| Insert the SD Card and turn on power | Navigate to opu-systems/petalinux/verification on the client computer and run the command:<br>`Sudo ./checkall <path to folder of hypso-cli>`<br>Wait for 30s, then turn of the power. |
| Change the mood mode pin strap switch to QSPI mode (according to fig. 3) | <br>The boot mode pin strap switches set to QSPI. |
| Turn on power | The test will now ping the picozed to check that the system has booted up correctly. If it has it will output a “1” on the terminal screen of the client computer.<br>- Pass<br>  - Test 1 of 2 passed for requirement: BL-5-10.<br>  - Test 1 of 4 passed for requirement: BL-5-30<br>  - Test 2 of 2 passed for requirement: BL-5-50<br>- Fail<br>  - requirement BL-5-10 failed. |
| Turn off power and corrupt the primary | Make a copy of the image.ub present on the SD-Card. Then edit the original file image.ub on the micro SD-Card by opening the file up in a |

| | |
|---|---|
| image | text editor on a computer with a Micro SD-Card reader and delete/reorder some small chunks of the machine code.<br>Insert the Micro SD-Card into the PicoBoB again. |
| Turn on power | The test will again perform a ping test to make sure the bootloader has been able to detect that the primary image is corrupted and that the golden image correctly boots up. It will again output a “1” on the terminal screen of the client computer if the test has passed.<br>- Pass:<br>    - Test 2 of 2 passed for requirement: BL-5-10<br>    - Test 2 of 4 passed for requirement: BL-5-30<br>    - Test 2 of 2 passed for requirement: BL-5-50<br>- Fail:<br>    - requirement BL-5-10 failed. |
| Perform a “in-flight” software update and turn on power again. | Upload the image.ub-copy from previous step to the mounted micro SD-Card on the picozed using the hypso-cli terminal (see HYPSO-DSW-004 for details on how this is done). Make sure the uploaded copy has the same name and overwrites the image.ub file already present. When completed, a reboot and an automated ping-test is executed to confirm that the primary image boots up correctly.<br>- Pass:<br>    - Test 3 of 4 passed for requirement: BL-5-30<br>    - Test 1 of 1 passed for requirment BL-5-20<br>- Fail:<br>    - requirement BL-5-30 failed<br>    - requirement BL-5-20 failed. |
| 6 x (Turn on power, wait 1 sec. Turn off power wait 1 sec.). Then turn on and leave power on. | This test will check that the alternative boot procedure is working when bootcounter > bootlimit (5), and correctly boots up the golden image.<br>- Pass:<br>    - Test 4 of 4 passed for requirement: BL-5-30<br>- Fail:<br>    - requirement BL-5-30 failed |
| Continue the script. When told so, perform the same power on-off procedure as in the previous step | The test script will now delete all files on the root file system of the golden image. Reboot, and delete all files on the primary image. |

| | |
|---|---|
| | The testscript will now output a "1" if the golden and primary image meets the requirements in BL-5-40<br>- Pass<br>  - Tes passed for requirement BL-5-40<br>- Fail<br>  - Requirement BL-5-40 failed |
| Test complete | If all tests have completed sucessfully (no error messages and only 1s outputted) the system is conforming to the defined requirements. |

# 4 List of Abbreviations

Table 3: List of Abbreviations

| Abbrv. | Description |
|---|---|
| DUT | Device Under Test |
| PCB | Printed Circuit Board |
| OS | Operating System |

Appendix C

# Test of Bootloader and Firmware Updates

Upload new boot image test report

Date

| 2 | 0 | 2 | 0 | 0 | 3 | 1 | 0 |
|---|---|---|---|---|---|---|---|

User(s): Write down the names of the people involved with this test

Joar Gjersund

Test Method: Describe what kind of methods were used during the test

Test was performed using an automated test script which included instructions for physical actions needed to be performed during the test

Test Equipment & Set-Up Description: Describe what kind of equipment was used and the set-up. Use pictures if possible

HYPSO-DSW-008 (Rev. 3) chapter 3.2

As-Ran Test Procedure: Describe how the test was performed, make especially note of an discrepancies from original plan

HYPSO-DSW-008 (Rev. 3) chapter 3.2

(VR.BL.5-10) Used bash command *shred image.ub* to inject errors in primary image.

() Used the following procedure and commands in hypso-cli to update image:
*copy image.ub to be uploaded in same folder as hypso-cli.*

*make sure ft list is empty*
*ft list 12*

*ft register 12 image.ub 1*
*ft prepare local image.ub image.ub.fmt 1 240*
*ft format 12 1 240 338755*
*ft upload file 12 image.ub.fmt 1*
*(wait 1 hour)*

*ft extract 12 1 image.ub*

*shell remote 12 10000*
*cp -fr image.ub /media/sd/image.ub*
*Reboot*

*UPDATE:*

*The file upload seems to work, but due to some bugs it is more complicated than it should be. This is how I was able to perform a firmware upgrade:*

```
(hypso) ft list 12
(hypso) ft register 12 image.ub 1
(hypso) ft prepare local image.ub image.ub.fmt 1 240
file_name:      image.ub.fmt
```

```
file_type:       STATIC
file_id:         1
entry_sz:        240
max_entries:     338755
first_entry_id:  1
total_entries:   338755
(hypso) ft format 12 1 240 338755
File ID: 1, status: 0
(hypso) ft upload file 12 image.ub.fmt 1
Uploading 1 missing ranges:
[       1-338755  ]: 100% [=========================]
File is complete.
Upload finished.
(hypso) ft extract 12 1 image.ub
/home/hypso/src/ft/ft_client.c:722:ft_client_extract: response length: 0, expected 3.
cli_ft_extract failed with ret: -71
`ft extract` failed: 71 (Protocol error)
(hypso) shell remote 12 10000000
Enter "exit", "quit" or "q" to exit remote shell.
(OPU) ls image.ub-e
```

*(I then had to press tab for autocomplete) getting the following result*

```
(OPU) ls image.ub-e\360
```

*I then had remove ls prepend cp -fr and append /media/sd/image.ub, like this:*

```
(OPU) cp -fr image.ub-e\360 /media/sd/image.ub
```

*then I did a reboot*

```
(OPU) reboot
```

*The new image (version 092a743) was confirmed by printing the hostname*

```
(hypso) shell remote 12 10000
Enter "exit", "quit" or "q" to exit remote shell.
(OPU) hostname
092a743-primary
```

Pass or Fail Criteria: Describe shortly what the test is supposed to determine, i.e. what would constitute success and failure.

HYPSO-DSW-008 (Rev. 3) chapter 3.2

Test Results: Quantify results of the test. Use tables and graphs to the extent it is applicable

| Requirement | Short description | Tests passed |
|---|---|---|
| VR.BL.10 | | 3/3 |
| - BL-5-10 | Corrupt image | 1ft l |
| - BL.5.30 | Automatic boot | 1 |
| - BL.5.50 | Bootingtime < 10s | 1 (recorded booting time was 1 min, but this was TBC) |
| VR.PROC.30/VR.BL.3 | Read online rootfs | 1 |

| | | |
|---|---|---|
| - BL-5-40 | | 1 |
| VR.PROC.50 | 36GB available | 0 |
| VR.BL.20 | Image update | 1 |

Discussion of Results: *Discuss the results in light of the pass / fail criteria of the test*

- Requirement VR.PROC.50 does not pass because the chosen SD-Card does not meet requirement. Current Available space is 8 GB. This Choice of SD-Card is based on tolerance towards radiation.

- None of the tests are done via the payload controller (PC), which might affect the test results. It is assumed that future tests of the PC will uncover these faults. If the same tests fail when done via the PC, the fault(s) should first be assumed to be with the PC.

- BL.5.50 does not pass or is invalid. Booting time was recorded to be 60s. This is primarily due to the DUT not being connected to ethernet during the test, and ssh-server waits for connection to establish. Actual booting time should be tested on a complete setup in the future. SSH server is also primarily a tool for development and could be considered disabled when deploying to the satellite.

- VR.BL.20: The time it takes to upload one image (ca. 80MB) just over the canbus is significant (at least 1 hour). This is without overhead associated with uplink to the PC and PC buffering which was not part of the test. Hypso-cli also reports some errors during the upload procedure (also mentioned in HYPSO-TRP-EL-006) and the ft file extract does not appear to be working as intended.

Conclusion: *If success, describe the impact of the results. If Failure, describe the remedial measures that should be taken.*

File upload displays error messages mostly related to timeout. Those bugs should be fixed.
BL.5.50 does not pass, but was TBC. This requirement should be updated to 100s to account for possible future updates which can further increase the boot time.

# Appendix D

# Example Image Tree Source for a FIT image

```
/dts-v1/;

/ {
    description = "U-Boot fitImage for plnx_aarch64
     ↪  kernel";
    #address-cells = <1>;

    images {
        kernel@0 {
            description = "Linux Kernel";
            data = /incbin/("./Image");
            type = "kernel";
            arch = "arm64";
            os = "linux";
            compression = "none";
            load = <0x80000>;
            entry = <0x80000>;
            hash@1 {
                algo = "sha1";
            };
        };
        fdt@0 {
            description = "Flattened Device Tree blob";
            data = /incbin/("./system.dtb");
            type = "flat_dt";
            arch = "arm64";
            compression = "none";
            hash@1 {
                algo = "sha1";
            };
        };
        ramdisk@0 {
            description = "ramdisk";
            data = /incbin/("./ramdisk.cpio");
            type = "ramdisk";
            arch = "arm64";
            os = "linux";
            compression = "none";
            hash@1 {
                algo = "sha1";
            };
        };
    };
    configurations {
        default = "conf@1";
        conf@1 {
            description = "Boot Linux kernel with FDT blob
             ↪   + ramdisk";
            kernel = "kernel@0";
            fdt = "fdt@0";
            ramdisk = "ramdisk@0";
            hash@1 {
                algo = "sha1";
            };
        };
    };
};
```

Source: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842374/U-Boot+Images

Appendix E

# Failure mode, effects and criticality analysis for the HYPSO Mission

| Subsystems | SubSubSystem | SubSubSubSystem | Reference letter | Function & requirements | Responsible | open/closed |
|---|---|---|---|---|---|---|
| **Payload** | HSI | OPU (on board processing unit) | A | Payload control and data processing, this is the interface between HSI+BoB and the spacecraft bus | Joe | open |
|  |  | Imager | B | High resolution hyperspectral imaging. Stores data in three dimensions (spatial x, spatial y, spectral) | Joe × Sivert | open |
|  |  | BOB(break out board) | C | The interface between OPU and PC/EPS/Cameras | Magne | Lukket |
|  |  |  |  |  |  |  |
|  | Software defined radio | RF front end | D | Receives and converts analog/digital signal to be processed by mother-board | Gara | open |
|  |  | RF motherboard | E | Controls the front-end and processes the digital signal received from front-end | Gara | open |
|  |  |  |  |  |  |  |
|  | RGB camera |  | F | Used for validating HSI images spatially and utilized for geo-referencing. Has larger FoV than HSI but lower spatial resolution. | Dennis | open |
| **Communications** | S-band radio |  | G | Recieving and sending HSI data (large amounts) | Roger | open |
|  | UHF radio |  | H | Used for sending telemetry (housekeeping) and receiving mission plan updates, spacecraft SW updates and camera parameters. Always ON | Roger | open |
| **ADCS** | AC |  | I | Controlling the spacecrafts physical motions. Orients the spacecraft towards desired attitude and actuates reaction wheels to perform a slew maneuver. Estimates the orientation, position and velocity of spacecraft autonomously | Bjørn x Mariusz | open |
|  | AD |  | J |  | Bjørn x Mariusz | open |
| **Power** |  |  | K | Providing electrical energy during operation from solar panels and stores in batteries which is managed at the Electrical Power System to distribute to other subsystems | Magne | open |
| **Ground Station** | NTNU |  | L | Daily monitoring. Operators identify, track and propagate based on TLEs. Sends commands and mission plan updates on request. Also receives HSI data through S-band. Will see the spacecraft about 6-7 times per day | Roger | open |
|  | Svalbard |  | M | Receives HSI data through S-band. Will see the spacecraft 10-11 times per day | Roger | open |
|  |  |  |  | Uplink of mission data (S-band) |  |  |
|  |  |  |  | Download of payload data (S-band) |  |  |

| Mission requirements | | Comments(feel free to edit or add) |
|---|---|---|
| 1.MS-0-001 | S/C shall successfully launch, deploy, detumble and initialize operations (LEOP and commissioning) in LEO within 3 weeks | Outside of the scope |
| 2.MS-0-002 | Shall observe Case 1 and Case 2 waters off the Norwegian of at least 70x70 km^2 area | Case 1: one type of water (far out to sea). Case 2: optically complex waters (close to the coast) |
| 3.MS-0-004 | Should image same target at least 3 passes per day | The satelitte orbits earth 15 times per day |
| 4.MS-0-006 | Should take at least 1 image with less than(at least?) 160 spectral bands in VIS-NIR with <10 nm spectral resolution | 160 spectral bands, the challenge is software management |
| 5.MS-0-008 | HSI images should have at least 100 m spatial resolution | Optical performance |
| 6.MS-0-010 | S/C should perform cross-track slew maneuver at a angular velocity with magnitude of 0.01 deg stability over 60 s | Angle, magnetorquer and rotary wheel adjustment, important that it is precise enough! |
| 7.MS-0-011 | Shall downlink 1 hyperspectral images in L1A data format containing detectable optical signatures (Chl-a, CDOM etc.) to be processed on ground | Software problem |
| 8.MS-0-012 | Should downlink 1 operational hyperspectral images in less than 1 hr after successful onboard dimensionality reduction, classification and target detection with certainty of 10 % of positive optical signatures (Chl-a, CDOM etc.) to be ground truthed | Software problem |
| 9.MS-0-013 | Shall enable flexible mission planning & scheduling and subsystem updates through successfully integrated uplinked mission data, FPGA programming logic and codes | Satellite tray, UHF. Data package size. time requirements, confirm success or whether it created software problems |
| 10.MS-0-016 | Should be operational for at least 5 years with weekly mission updates during peak-season | Ground station, monitoring. current |

- **What is FMECA, and what is the purpose?**

  FMECA — failure mode, effects and criticality analysis — is a tool for identifying potential problems, their causes, impact (effect) and criticality, and systematize this information in a standardized manner.

  The FMECA is usually created within a spreadsheet. In order to conduct a FMECA the analyst needs a deep understanding of the system to be analysed, but the tool itself is fairly simple to learn.

  The main purpose of a FMECA is to find out which components are the most critical, meaning which is the most likely to prevent the main functions and requirements. In addition, FMECA is a great tool to systematize and present a large amount of information about a system.

- **What is a failure mode?**

  <u>A failure mode is the partial or full absence of a function.</u>

  A failure mode occur when:

  - Preferred action is not happening, because of an unintentional error.
  - Preferred action is happening at the wrong time or with wrong duration, causing an unwanted effect.
  - There is an error related to the instruments. An instrument is for some

reason giving incorrect data or reading.

When finding failure modes, one assume that all the other components in the system are working perfectly.

*Example of a function:* A smoke detector's function is to alert when there is smoke, and not alert when there isn't smoke.

*Example of a failure mode:* The smoke detector not alerting during a fire is a failure mode. The smoke detector alerting when there is no fire is another failure mode.

- **What is a failure effect?**

  The failure effect is the consequence of the failure mode. Which effect will the failure mode have on other units and the main function?
  *Example of a failure effect:* If the fire detector doesn't alert during a fire, the failure effect can be a burned down house.

- **What is a failure cause?**

  A failure cause is the circumstances during specification, design, manufacture, installation, use or maintenance that result in failure.
  *Example of failure cause:* Lack of batteries in the fire detector, or error in the

speaker.

- **What is a RPN?**

  RPN stands for Risk Priority Number. Components with a high RPN should be paid special attention to. The RPN is calculated by this formula:

  RPN = Severity X Occurrence X Detection

  We have chosen to only use severity and occurrence to calculate the RPN:

  RPN = Severity X Occurrence

| Severity number | Severity class | Severity description |
|---|---|---|
| 1 | Negligible | Operating conditions are such that personnel error, environment, design deficiencies, subsystem or component failure or procedural deficiencies will result in no effect on the systems function |
| 2 | Marginal | Failure may commonly cause minor effect on the systems function. |
| 3 | Considerable | Failure may in some cases cause functions to stop system from fulfilling mission success requirements |
| 4 | Critical | Failure causes serious absence of required functions. Most mission success requirements will not be met. |
| 5 | Catastrophic | System ceases to function, no mission success requirements can be met |

| Occurrence number | Occurrence class | Occurrence description |
|---|---|---|
| 1 | Improbable | So unlikely that occurrence is negligible |
| 2 | Remote | Occurrence possible but unlikely |
| 3 | Occasional | Likely to occur at some point in lifetime |
| 4 | Probable | Will occur several times in lifetime |
| 5 | Frequent | Likely to occur often |

| **(A) Payload:HSI:Onboard processing Unit (System on Cip SoC)** | Sivert | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operational mode | Description of operation | Function | Failure mode nr. | Failure Mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
| Cleanup | Delete frames and prepare for new | | A1 | Does not delete the old image cube | Less memory left | Wrong OS path | | Less space for more images | 3 | 3 | 9 | Test, monitor, housekeeping | It's important that we are able to take several images |
| | | | | | | Hardware failure | | | | | | | |
| | | | A17 | Deletes wrong area of memory | with multiple boot images, detect as falling back to boot 2 | Bad programming, not setting the OS partitions of the memory as read-only | | Main function temporarily unavailable | 2 | 4 | 8 | Redundant boot images | |
| | | | | | checksum changes | | | | | | | Mount as read only filesystem | |
| | | | | | | | | | | | | Set up things in RAI | |
| Process image | | | A2 | Modules run in the wrong order | Corrupted data | Bad scheduling on operator's side | modules receive corrupt data | wrongly processed data is downlinked | 2 | 1 | 2 | Double check command schedule prior to uplink | |
| | | | | | Error executing module | Erronous end-to-end testing | modules do not receive expected metadata | | | | | Test the end-to-en testing? | Not introduced as mitigation action |
| | | | | | | Erronous or limited descriptions of interfaces between modules | | | | | | metadata module flags | |
| | | | | | | Bad programming | | | | | | Better programming standards, testing and verification procedures | |
| Inititilization | Same as Cleanup? But requires reboot of BoB, payload controller and HSI | | A3 | Datacube is corrupted in a malfunctioning module | data not passed to next module | A2 | | Main function temporarily unavailable. If the data is not passed to te next module, The satellite may transmit some error message, and the operators need to investigate the falure | 2 | 3 | 6 | have programs that verify data inbetween modules? | |
| | | | | | downlinked data observed to be corrupt | Undetected bug in module | | | | | | | |
| | | | | | | Logic gates in FPGA flipped due to solar in-flux and cosmic rays | | | | | | | |
| | | | A4 | Missing input metadata | error thrown by module | ADCS data not received | data processed in metadata agnostic way | simplified image processing | 3 | 1 | 3 | check input metadata | |
| | | | | | | timing is not received | | | | | | write code to work even without metadata | |
| | | | | | | previous module does not send appropriate metadata | | | | | | | |
| | | | A5 | Software crash | no response when controller is pinged | Untested software | nothing works | Main function temporarily not availably. Restart payload | 3 | 3 | 9 | Use the software a lot in different ways | |
| | | | | | Keep uptime in housekeeping telemetry | Heisen bugs | | | | | | Watchdog (implemented in bus, but not in payload) | |
| | | | | | | Erronous parameters | | | | | | | |
| | | | | | | Cosmic rays flipping bits | | | | | | | |
| | | | A6 | corrupted data received from other subsystems not handeld (skip) | | | | | | | | | |
| | | | A7 | wrong / unexpected input meta data leads erronous processing (skip) | | | | | | | | | |
| | | | A18 | OPU doesnt boot up | No response to csp ping from OPU | Error in software | Nothing regarding imaging works | imaging fails | 1 | 5 | 5 | secondary boot image | |
| | | | | | | damage from space environment | | | | | | | |
| | | | A8 | Image & analysis not transferred from OPU | empty data packages received at payload controller or ground | module in imaging pipeline does not transfer image | payload controller does not receive image | image is not downlinked | 2 | 2 | 4 | look at image after downlinking to ensure that it is present | |
| | | | A9 | | | | | | | | | | |
| | | | A10 | Unexpected metadata cause crash (skip) | software crash | unex | | | | | | | |
| | | | A11 | bit flip due to solar-influx causes erronous processing (skip) | | | | | | | | | |
| | | | A12 | corrupted processed data, intended for downlink | Data fail reliability test | bad module in image processing pipeline | wrong data downlinked | data downlinked is wrong | 3 | 1 | 3 | Check downlinked data | |
| | | | A13 | outdated calibration coefficients | data look suspicious | calibration coefficients not updated | image sent to payload controller is inaccurate | downlinked image is inaccurate | 3 | 2 | 6 | update calibration coefficients | |
| | | | A14 | data not transferred to the payload controller | data is not received at pc | bad transfer | no image to downlink | no image downlinked | 2 | 2 | 4 | Check to see if image is downlinked | |
| | | | | | | | | | | | | Adjust parameters accordingly | |
| | | | A15 | does not store data | data is not stored by camera | bad transfer from HSI to OPU | no image to process | image not downlinked | 1 | 2 | 2 | alert groundstation if no image | |
| | | | | | | or data not stored on HSi | | | | | | | |
| | | | A16 | initialization goes wrong | camera does not take image | imager not initiallized properly | no image to process | image not downlinked | 1 | 1 | 1 | alert groundstation if no image | |

**(B)Payload: HSI: imager**

| Operational mode | Description of operational mode | Function | Failure mode nr. | Failure Mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Record datacube | Camera on, camera initialized, record a datacube with a set sensor AOI, frame rate, exposure, spectral sampling rate | Record datacube | B1 | No image | Erronous or unexpected Image cube | Software Malfunction | Errnous data can be sent to further processing modules | Unable to capture image cube | 2 | 4 | 8 | Test Software for all possible parameter configurations that are expected | |
| | | | | | | Hardware Malfunction | Errnous data can be sent to further processing modules | Unable to capture image cube | 3 | 4 | 12 | Make sure that HW is designed for expected envirnoment | See hardware FMECA |
| Example options: | Camera ON, Needs to initialize for x seconds,, Needs to initialize, medium AOI on sensor, High frame rate per second, low exposure (high data rate throughput). Camera settings need to be adjusted to this | Transfer datacube | B2 | Offset/skewed image on sensor | Erronous or unexpected Image Cube | Hardware Malfunction | Errnous data can be sent to further processing modules | The entire image cube will not be useful for scientific purposes | 4 | 2 | 8 | Make sure that HW is designed for expected envirnoment | See hardware FMECA |
| i. High Frame Rate | Camera ON, Needs to initialize for x seconds,, Needs to initialize, medium AOI on sensor, High frame rate per second, low exposure (high data rate throughput). Camera settings need to be adjusted to this | | B3 | Offset spectral bands | Thorugh analysis and calibration of sensor | degradation or mechanical movement of optical components | Errnous data can be sent to further processing modules | The data in the image cube is erronous | 3 | 3 | 9 | Make sure that HW is designed for expected envirnoment, perform in-flight calibration | See hardware FMECA |
| | | | B4 | Artifacts in image | Erronous or unexpected Image Cube | degradation or mechanical movement of optical components | Errnous data or data of poor quality can be sent to further processing modules | The data in the image cube is erronous | 2 | 3 | 6 | Make sure that HW is designed for expected envirnoment, perform in-flight calibration | See hardware FMECA |
| | | | B5 | Changing focus between orbits/imaging sessions | Erronous or unexpected Image Cube | degradation or mechanical movement of optical components | data of poor quality can be sent to further processing modules | The data in the image cube is of poor quality | 2 | 3 | 6 | Make sure that HW is designed for expected envirnoment, perform in-flight calibration | See hardware FMECA |
| ii. High Image Resolution | Camera ON, Needs to initialize for x seconds, large AOI on sensor, medium frame rate per second, medium exposure (medium data rate throughput). Camera settings need to be adjusted to this | | B6 | Permanent lack of focus | Erronous or unexpected Image Cube | degradation or mechanical movement of optical components | data of poor quality can be sent to further processing modules | The data in the image cube is of poor quality | 2 | 4 | 8 | Make sure that HW is designed for expected envirnoment, perform in-flight calibration | See hardware FMECA |
| | | | B7 | Image of wrong area (pointing is offset) | Erronous or unexpected Image Cube | Worngly calibrated navigation subsystems | georefrencing and image registration will be of poor quality | Autnomous Assets can be sent to wrong destination | 2 | 4 | 8 | Perform in-flight calibration | |
| | | | | | Erronous or unexpected Image Cube | mechanical shift of system | georefrencing and image registration will be of poor quality | Autnomous Assets can be sent to wrong destination | 2 | 3 | 6 | Make sure that HW is designed for expected envirnoment, perform in-flight calibration | see hardware fmeca |
| | | | B8 | Noise in image | Erronous or unexpected Image Cube | Imaging Sensor degradation | data of poor quality can be sent to further processing modules | The data in the image cube is erronous | 4 | 3 | 12 | Make sure that HW is designed for expected envirnoment, testing | see hardware fmeca |
| | | | | | Erronous or unexpected Image Cube | Optical degradation | data of poor quality can be sent to further processing modules | The data in the image cube is erronous | 4 | 3 | 12 | Make sure that HW is designed for expected envirnoment, perform in-flight calibration | see hardware fmeca |
| | | | | | Erronous or unexpected Image Cube | Software Malfunction | data of poor quality can be sent to further processing modules | The data in the image cube is erronous | 2 | 2 | 4 | Do testing and verification of software | |
| | | | | | Erronous or unexpected Image Cube | Outgassing causes lenses to be clogged | | | | | | | |
| iii. Low Bit Image | Camera ON, Needs to initialize for x seconds, small AOI on sensor, low frame rate per second, high exposure (low data rate throughput). Camera settings need to be adjusted to this | | B9 | Under/overexposed image | Erronous or unexpected Image Cube, underexposed | Too little light reaches the sensor | data of poor quality can be sent to further processing modules | The data in the image cube is of poor qualitty | 3 | 4 | 12 | Increase software gain | |
| | | | | | | | | | | | | increase hardware gain | |
| | | | | | Erronous or unexpected Image Cube, overexposed | Too much light reaches the sensor | data of poor quality can be sent to further processing modules | The data in the image cube is of poor qualitty | 2 | 2 | 4 | | |
| | | | | | | | | | | | | Lower SW/HW gain | |
| | | | | | | | | | | | | Increase framerate | |
| | | | B10 | Image stuck in buffer | Erronous, unexpected or no Image Cube | | | | | | | | |
| | | | B11 | Only partial data cube transferred | Erronous or unexpected Image Cube | | | | | | 0 | | |
| iv. Subsampled | Camera ON, Needs to initialize for x seconds,, Needs to initialize, High AOI on sensor, High frame rate per second, low exposure (high data rate throughput). | | B12 | Timing metadata are not collected | Missing timing values | I/O pin does not read timing signal | Will hamper other processing steps | Not enough data to do timign accuratly | 2 | 4 | 8 | Make sure that the connectors are designed to survive the space envirnoment | |
| | | | | | | Software not read timing signal | Will hamper other processing steps | | 2 | 4 | 4 | Do testing and verification of software | |
| | | | B13 | Dark/underxposed image | Light does not reach sensor | Outgassing causes lenses to be clogged | | | | | | | |
| | | | B14 | Data corrupted while being transmitted to BoB | | | | | | | 0 | | |
| | | | B15 | Unexpected camera shutdown | | | | | | | | | |
| | | | | | | | | | | | 0 | | |
| | | | B17 | camera initialization fails | | Camera will not turn back on after being in safe mode | | | | | | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | | | |
| Idle | Camera is off. But connected to the BoB and power (still alive) | Save power | | | | | | | | | 0 | | |
| | | Camera is startable | B20 | Camera is still drawing too much power | | | | | | | | | |

| **(C) Paylod: HSI:Break Out Board** | Magne | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operational mode | Description of operational mode | Function | Failure mode nr. | Failure Mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
| Nominal | Nominal behavior | Provides power to BOB systems | C1 | OPU power transmission fails | No response from OUP | 5V regulator fails | Cameras and OPU can't operate | No image acquistiion capability | 2 | 5 | 10 | | |
| | | | | | | Cables from EPS are damaged | | | | | | | |
| | | | C2 | HSI power transmission fails | No response from HSI camera | EPS routing is damaged | | No HSI image acquisition | 2 | 4 | 8 | | |
| | | | | | | Cables from EPS are damaged | | | | | | | |
| | | | C3 | RGB power transmission fails | No response from RGB camera | 5V regulator fails | ADCS validation and georeferencing can't get RGB data | No RGB image acquisition | 2 | 3 | 6 | | |
| | | | | | | Cables from EPS are damaged | | | | | | | |
| | | Image data is transferred from OP to SD card on BoB | C4 | Unable to store images on SD-card | | data corrupted during transfer | | Loss of one image | 3 | 2 | 6 | Verify image from non-volatile memory before clearing image from volatile memory. | |
| | | | | | | does not store data | | Unable to store any data, can only hold image in DDR memory | 2 | 4 | 8 | Have backup memory storage (PC buffering, eMMC) | The picozed has an embedded eMMC which could be repurposed to hold image data if SD card is damaged |
| | | | | | | data store is corrupted | | Loss of multiple current images | 3 | 2 | 6 | Use ECC when storing images | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | The OPU is able to communicate over the CAN-bus via BoB | C5 | Cannot receive or transmit CSP data | No response from OPU | CAN transceiver damaged from overvoltage | Other units can't get any data or response from OPU | No HSI/RGB image acquisition | 2 | 5 | 10 | replacing transceiver with one with overvoltage protection | |
| | | | | | | CAN transceiver damaged from cosmic rays | | | 2 | 5 | 10 | sourcing a space grade transceiver | |
| | | | | | | Faulty circuit design | | | 2 | 5 | 10 | performing design reviews | |
| | | The OPU is able to communicate with the HSI Camera | C6 | Cannot receive HSI data/metadata | No response from HSI camera | Ethernet connector mechanicalically damaged | Other units can't get HSI data | No HSI image acquisition | 1 | 4 | 4 | Creating and following assembly procedures | |
| | | | | | | Ethernet cables mechanically damaged | | | 2 | 4 | 8 | Creating and following assembly procedures | |
| | | The OPU is able to communicate with the RGB Camera | C7 | Cannot receive RGB data/metadata, or transmit RGB commands | No response from RGB camera | USB connector mechanicalically damaged | ADCS validation and georeferencing can't get RGB data | No RGB image acquisition | 1 | 3 | 3 | Creating and following assembly procedures | |
| | | | C8 | | | USB cables mechanically damaged | | | 2 | 3 | 6 | Creating and following assembly procedures | |
| | | | | | | | | | | | | | |

| **(F) Payload: RGB** | Dennis | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operational modes | Description of operational mode | Function | Failure mode nr. | Failure mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
| Off | Camera is disconnected from power and uses no energy. | Using no power | F1 | Module uses power when it should not | If camera is connected to its own 5V power line, EPS can measure how much power the camera uses. | mechanical failure causing short circuit | more power use than desired, draining battery, and potentially hindering the main payload and other operational functions | if power use is too high for too long, can drain battery so that no more energy is left for normal operation | 2 | 3 | 6 | Prevention: Stringent shock and vibration tests on rgb camera related wiring. Mitigation: EPS disabling power to module | |
| | | Being ready to be turned on | F2 | Module does not turn on when operations expect | RGB camera service reports an error that camera is not connected | mechanical failure causing short circuit inside module that breaks the camera or mechanical failure that disconnects a vital circuit inside camera module or of the camera module | Verification of atitude is harder for humans to see | No failure effect on main function | 2 | 2 | 4 | Prevention: Stringent shock and vibration tests on rgb camera related wiring. | |
| Initializing | Process where power is connected to camera and RGB service opens the camera and applies parameters from config file. Takes about 15 seconds | Making camera ready for taking images and communicating with computer | F3 | One step in initialization sequence fails | RGB camera service reports an error and cancels initialization | Cosmic/stellar ray or particles causing upset | RGB image delayed or missing | No failure effect on main function | 3 | 2 | 6 | Restart sequence | |
| Initialized / Opened / Standby | Camera initialized and ready to be confgured further and take pictures, waiting for capture command. | Ready to respond to commands | F4 | Camera changing its mode unexpectedly | The next command executed is failing | Upset, loose wiring? | RGB image delayed or missing | No failure effect on main function | 2 | 2 | 4 | Resending command or turning camera off and on again | |
| | | | F5 | Command execution fails | The next command executed is failing | | RGB image delayed or missing | No failure effect on main function | 1 | 2 | 2 | Restart initialization sequence | |
| Taking pictures | Camera is currently capturing an image: 1. Allocating memory 2. Waiting for HW or SW trigger 3. Exposing 4. Reading image from camera 5. saving image to SD card | | F6 | Noise in image | Visual inspetion of downlinked image | Too much space radiation hitting sensor during exposure, upsets changing camera gain settings | Verification of atitude is harder for humans to see | No failure effect on main function | 4 | 1 | 4 | Resetting camera parameters | |
| | | | F7 | Underexposed image | Visual inspetion of downlinked image | Upset changing exposure, pixel clock or framerate settings | Verification of atitude is harder for humans to see | No failure effect on main function | 2 | 1 | 2 | Resetting camera parameters | |
| | | | F8 | Overexposed image | Visual inspetion of downlinked image | Upset changing exposure, pixel clock or framerate settings | Verification of atitude is harder for humans to see | No failure effect on main function | 2 | 1 | 2 | Resetting camera parameters | |
| | | | F9 | Unexpected camera shutdown | The next executed command is failing | Latchup or upset | RGB image delayed or missing | No failure effect on main function | 2 | 2 | 4 | Restarting camera | |
| | | | F10 | Artifacts in image | Visual inspetion of downlinked image | Space radiation? | Verification of atitude is harder for humans to see | No failure effect on main function | 2 | 2 | 4 | | |
| | | | F11 | Changing focus between orbits/imaging sessions | Visual inspetion of downlinked images | Probably temperature variations causing change of optical system | Verification of atitude is harder for humans to see | No failure effect on main function | 3 | 2 | 6 | | |
| | | | F12 | Permanent lack of focus | Visual inspetion of downlinked images | Damage to optical system during launch or permanent damage from temperature variations, low pressure environment? | Verification of atitude is harder for humans to see | No failure effect on main function | 2 | 3 | 6 | Proper vibration and shock testing procedure | |
| | | | F13 | Transfer Error | RGB Camera service reports an error | Unknown bu common error encountered during testing | RGB image delayed or missing | No failure effect on main function | 3 | 1 | 3 | Restarting camera | |

**(G)Payload: Comm S-band Radio**

| Operational m | Description of operational m | Function | Failure mode nr. | Failure Mode | Failure detection | Failure cause | Failure effect on | Failure effect on | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing n | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S-band RX | Receiving data from ground. ALWAYS ON | Receive analog signal from ground | G1 | Does not recieve data at all | | Noisy signal, Wrong pointing of antenna, | | Dont recives misson planning schedule | 3 | 3 | 9 | Testing | |
| | | Convert signal from analog to digital data | G2 | Receives corrupt data | | Noisy signal | | dont recives misson planning schedule | 3 | 3 | 9 | Have default mission planning onboard | |
| | | Send digital data from radio to payload controller | *G3* | Does not send any data | | Databus is saturated | | Lose the mission planning | 2 | 3 | 6 | | |
| | | (Data corruption check) | | N/A | | | | | | | 0 | | |
| | | Acknowledge that packages are recived | G4 | Does not send an acknowledgement | | Outside the communication window | | Ground station wont be sure that we got tha last package | 3 | 3 | 9 | Send periodic acknowledgement s | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| S-band TX | Transmits data from payload controller to ground. | Transmit analog signal from satellite to ground | G5 | Does not send data from satellite to ground | | Bad pointing of the antenna | | It will not downlink images, and housekeeping data | 2 | 5 | 10 | Switch to UHF radio | |
| | | Convert data from digital to analog data | G6 | Sends corrupt data | | Corrupted encoder | | Downlink corrupt images and housekeeping data | 2 | 4 | 8 | | |
| | | Receive digital data from payload controller | G7 | doesnt recieve digital data from payload controller | | Payload controller is busy with other tasks | | It will not downlink images, and housekeeping data | 2 | 4 | 8 | | |
| | | Handshake with ground | G8 | Resends old data without being asked to | | | | | | | 0 | | |
| | | | | No handshake | | Bad pointing of antenna, strong interference | | it will not downlink images and housekeeping data | 2 | 5 | | | |
| | | Packet transmission control | G13 | cant resume transmitting the missing packets | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| Safe Mode | Reboot and diagnostics; subsystem check-out. Initialize RX and TX | Reboots | G9 | Does not reboot | | frozen software | | Cannot control S-band radio | 1 | 4 | 4 | switch to backup UHF radio | |
| | | | G10 | Sends wrong telemetry | | Sensors are not well calibrated | | Does not have telemetry from satellite | 2 | 3 | 6 | | |
| | | Initialize RX | G11 | Does not initialize RX | | Does not have power, software corrupted | | Cannot recieve | 2 | 4 | 8 | | |
| | | Initialize TX | G12 | Does not initialize TX | | Does not have power, software corrupted | | Cannot transmit | 2 | 4 | 8 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |

**(H)Comm UHF radio**

| Operational mode | Description of operational mode | Function | Failure mode nr. | Failure Mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UHF RX | Receiving data from ground. ALWAYS ON | | H1 | Does not recieve data at all | | | | | | | 0 | | |
| | | | H2 | Receives corrupt data | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| UHF TX | Transmits data from Flight computer to ground. | | H3 | Does not send data | | | | | | | 0 | | |
| | | | H4 | Sends corrupt data | | | | | | | 0 | | |
| | | | H5 | Send partial data | | | | | | | 0 | | |
| | | | H6 | Resends old data without being asked to | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| Idle | Same as RX | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| Safe Mode | Reboot and diagnostics; subsystem check-out. Initialize RX and TX | | H7 | Does not reboot | | | | | | | 0 | | |
| | | | H8 | Sends wrong telemetry | | | | | | | 0 | | |
| | | | H9 | Does not initialize RX | | | | | | | 0 | | |
| | | | H10 | Does not initialize TX | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |

**(I)ADCS**

| Operational mode | Description of operational mode | Function | Failure mode nr. | Failure mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Orbit mode | Orbit mode happens right after deployment, OFF for 45 min until detumbling automatically turns on | ADCS is off for 45 min right after deployment | I11 | Attitude Control Failure Mode | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| Detumble mode | Magnetorquers are on, de-spins the satellite after deployment. Automatically turns on 45 min after deployment; utilizes Earth's magnetic field | ADCS stops the satellite from spinning | I12 | Magnetorquers do not dump reaction wheel momentum | | Magnetorquers malfunctioning | | | 2 | 5 | 10 | | |
| | | | I13 | | | | | | | | 0 | | |
| | | | I14 | Wrong actuator signals to magnetorquers | | Signaling error | | | 2 | 4 | 8 | | |
| | | | I15 | Magnetorquers have wrong direction in magnetc field | | Mistake in setup | | | 1 | 5 | 5 | | |
| Idle mode | Magnetorquers are ON all the time. All sensors except for star-tracker and IMU are on. Constantly controls spacecraft against perturbations. Magnetorquers are on to dump momentum from reaction wheels | Makes the satellite orbit with a constant attitude | I16 | Reaction wheels saturate | | Magnetorquers malfunctioning, wrong measurements from sensors (AD) | | | 2 | 3 | 6 | | |
| | | | I17 | Reaction wheel does not spin up | | Reaction wheels malfunction | | | 1 | 3 | 3 | | |
| | | | I18 | Magnetorquers dont work | | Magnetorquers malfunction | | | 1 | 3 | 3 | | |
| Velocity mode | Aligns the S/C z-axis along the velocity vector to minimize drag. Reaction wheels are ON | Attitude control | I19 | Desired attitude not achieved | | Reaction wheels malfunctioning | | | 2 | 2 | 4 | | |
| User supplied vector | Uses reaction wheels and magnetorquers to align against an arbitrary user supplied position (i.e. point towards the Moon) | Pointing | I20 | Does not point in right direction | | Magnetorquers malfunctioning, reaction wheels malfunctioning, AD malfunctioning | | MS-0-002, MS-0-004, MS-0-011 | 2 | 3 | 6 | | |
| Sun pointing | Nominal mode when not in eclipse. When sun is detected by sun sensors, the S/C uses reaction wheels to align against the Sun with the +Y-face vector parallel to sun vector (largest face 90 degrees) | Pointing | I21 | Does not point in right direction | | Magnetorquers malfunctioning, reaction wheels malfunctioning, AD malfunctioning | | | 2 | 3 | 6 | | |
| Nadir | Points the camera straight down towards Earth's surface (center) | Pointing | I22 | Does not point in right direction | | Magnetorquers malfunctioning, reaction wheels malfunctioning, AD malfunctioning | | MS-0-002 | 1 | 4 | 4 | | |
| Prepare slew | S/C uses reaction wheels to point to an user-specified pitch angle (e.g. 60 deg) along orbit track, while keeping roll and yaw angles (almost zero) | Pointing | I23 | Wrong angle | | Reaction wheels not accurate enough, AD malfunctioning | | MS-0-002, MS-0-004, MS-0-010 | 1 | 4 | 4 | | |
| | | | I24 | roll and Yaw angles not almost zero | | Reaction wheels not accurate enough, AD malfunctioning | | MS-0-002, MS-0-004, MS-0-010 | 2 | 3 | 6 | | |
| In track slew | S/C actuates the reaction wheels opposite way of orbit track about S/C y-axis, uses Reaction wheels to achieve <1 deg/s angular velocity. Angular velocities in x and y-axes are kept to a minimal | Slew | I25 | Does not slew at correct rate | | Reaction wheels not accurate enough | | MS-0-002, MS-0-004, MS-0-010 | 1 | 4 | 4 | | |
| | | | I26 | Does not keep y and x axes at minimal angular velocities | | Reaction wheels not accurate enough | | | 2 | 3 | 6 | | |
| Cross-track | S/C actuates the reaction wheels opposite way of orbit track about S/C y-axis while pointing at a roll angle (e.g. 10 deg), uses Reaction wheels to achieve <1 deg/s angular velocity. Angular velocities in x and z-axes are kept to a minimal | Slew | I27 | Does not slew at correct rate | | Reaction wheels not accurate enough | | MS-0-002, MS-0-004, MS-0-010 | 1 | 4 | 4 | | |
| | | | I28 | Does not keep z and x axes at minimal angular velocities | | Reaction wheels not accurate enough | | | 2 | 3 | 6 | | |
| Safe mode | Anomalies in subsystems, saturation. Turns off, reboots all sensors/reaction wheels/magnetorquers, goes back to detumbling mode | Reset | | | Does not reset | | | | | | 0 | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

**(J)ADCS** Attitude Determination Failure Mode ADCS Failure Mode (total)

| Function | Failure mode nr. | Failure Mode | Failure effect | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Estimate angular velocity | J1 | Fails to estimate angular velocity | | | Flaw in inertial measurement unit (IMU), error in signals between IMU and AD | AC: Most tasks | | 1 | 4 | 4 | | |
| Estimate magnetic field | J2 | Fails to estimate magnetic field | | | Flaw in magnetometers, error in signals between magnetometers and AD | AC: Detumbling, Idle | MS-0-001 | 1 | 3 | 3 | | |
| Estimate sun vector | J3 | Fails to estimate sun vector | | | Flaw in sun sensors, error in signals between sun sensors and AD | AC: Sun pointing | | 1 | 2 | 2 | | |
| Estimate position | J4 | Fails to estimate position | | | Star tracker covered/blocked | AC: Any that requires pointing somewhere (sun pointing, user vector) | MS-0-004, MS-0-011 | 1 | 3 | 3 | | |
| Estimate attitude | J5 | Fails to estimate attitude | | | Flaw in inertial measurement unit (IMU), error in signals between IMU and AD | Needed for all AC tasks | MS-0-002, MS-0-004, MS-0-010, MS-0-011 | 1 | 5 | 5 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |
| | | | | | | | | | | 0 | | |

| **(K)Power** | Magne | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Operational mode | Description of operational mode | Function | Failure mode nr. | Failure Mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizing measures | Notes |
| Safe Mode | Reboots EPS, happens when too much power is drawn | Convert solar energy to power | | | | | | | | | 0 | | |
| | | Reboot EPS | K1 | EPS is not rebooted | No response from satellite | | All units fail | Total failure | 2 | 5 | 10 | | |
| | | Store power in batteries | | | | | | | | | 0 | | |
| | | Limit discharge of batteries | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| Critical Mode | Power consumption is too high, dept of discharge is too high | Distribute power to critical subsystems | K2 | Subsystems do not get power | | | | | | 4 | 0 | | |
| | | Convert solar energy to power | K3 | Solar energy is not converted | | | | | 2 | 5 | 10 | | |
| | | Charge batteries | K4 | Batteries are not charged (same as solar energy not converted?) | | | | | 2 | 5 | 10 | | |
| | | Store power in batteries | K5 | Some batteries fail | | | | | 3 | 4 | 12 | | |
| | | Cut power to non-critical subsystems | K6 | All batteries fail | | | | | 2 | 5 | 10 | | |
| | | Limit discharge of batteries | K7 | Batteries are discharged more than depth-of-discharge | | | | | | | 0 | | |
| | | | K8 | Subsystems that are not supposed to get power in critical mode get power(sdr) | | | | | 1 | 4 | 4 | | |
| | | | | | | | | | | | 0 | | |
| Nominal Mode | Normal operation. Draws current and voltage to a certain level | Distribute power to active subsystems | K9 | Subsystems do not get power | | | | | | | 0 | | |
| | | Convert solar energy to power | K10 | Solar energy is not converted | | | | | | | 0 | | |
| | | Store power in batteries | K11 | Some batteries fail | | | | | | | 0 | | |
| | | Charge batteries | K12 | Batteries are not charged (same as solar energy not converted?) | | | | | | | 0 | | |
| | | Limit discharge of batteries | K13 | Batteries are discharged more than depth-of-discharge | | | | | | | 0 | | |
| | All power channels are monitored and found to be within current consumption limits. | Monitor current consumption of subsystems. | K14 | A subsystem consumes more current than the threshold on the power channel | Monitoring of EPS telemetry | Short-circuit in subsystem | Affected subsystem will not be able to provide other systems with services | No image acquisition if the wrong subsystem fails | 2 | 4 | 8 | Following best practices when designing circuits | |
| | | Disconnect a power channel if it consumes too much power. | | | | Unexpected high peak power concumption in subsystem | Affected subsystem will be temporarily unabvail | Temporarily unavailable | 3 | 2 | 6 | Performing power analysis and measurements | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| Performance Mode | Depth-of-discharge allowance is expanded (>15%) | Distribute power to active subsystems | | | | | | | | | 0 | | |
| | | Convert solar energy to power | | | | | | | | | 0 | | |
| | | Charge batteries | | | | | | | | | 0 | | |
| | | Store power in batteries | | | | | | | | | 0 | | |
| | | Allow more discharge of batteries, but not let it get to safe mode level | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |

**(L)Ground station: NTNU Trondheim**

| Operational mode | Description of operational mode | Function | Failure mode nr. | Failure Mode | Failure | Failure cause | Failure ef | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizin | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Standby/Idle | | Ready to receive commands from ground station operator | L1 | Does not react | | Frozen software | | Lose track of the satelite, no comms | 2 | 2 | 4 | | |
| | | | | | | Internet does not work | | same | 2 | 2 | 4 | | |
| | | | | | | | | | | | 0 | | |
| RX | Receives telemetry, HSI data, tracks acitvely the spacecraft | Track satellite | L2 | Tracks wrong satellite | | Satellite points wrong way or recive weaker signal from our satellite than from an other stallite | | Lose track of the satelite, no comms | 2 | 4 | 8 | | |
| | | Receives analog signal from satellite | L3 | Does not recieve analog signal from satelite | | Signal too weak, or signal too noisy or the antenna pionts wrong direction | | We have no data or comms | 1 | 4 | 4 | | |
| | | Converts analog signal from satellite to digital data | L4 | Receives noisy signal | | Antenna not pointing correctly or a lot of interference | | recive corrupted data | 4 | 4 | 16 | Use another ground station | |
| | | | | | | | | Does not know health status of satelite | | | | | |
| | | Decode digital signal | L5 | Does not decode digital signal | | Bug in decoder or noisy signal | | Not contracting image, does not know health status of satellite | 1 | 4 | 4 | | |
| | | save recived raw data | L6 | Does not save recieved data | | Memory is full, or the memory is corrupted, can't access online data base | | Losing data packages, noes not know health status, does not contract image | 2 | 5 | 10 | Have different places to save data, and always different places to save data | |
| | | Sends data to operator | L7 | does not send data to operator | | No internet connection | | does not get image(but its saved) | 2 | 2 | 4 | | |
| | | Provides operator with user interface | L8 | does not provide operator with user interface | | Frozen software | | Image difficult to analyze, the operator will have to chek the lokal memory | 2 | 2 | 4 | | |
| | | | | | | | | | | | 0 | | |
| | | | | | | | | | | | 0 | | |
| TX | Transmits mission plan data, software updates, does not need to track (?) | Transmits analog signal to satellite | L9 | Transmit corrupted signal | | Doesn't read command properly | | satellite could go to safemode | 1 | 2 | 2 | | |
| | | Converts digital data to analog signal | L10 | Does not convert digital data to analog signal | | analog to digital converter not working properly | | No signal is transmitted | 1 | 2 | 2 | | |
| | | Tracks satellite | L11 | Does not track satellite | | Bad pointing of antenna, weaker signal than from another satellite | | No signal, does not recieve data | 3 | 3 | 9 | Two antennas tracking the satellite, bigger antenna | |
| | | Provides operator with user interface | L12 | Does not provide operator with user interface | | Frozen software | | Difficulties sending commands to the satellite | 2 | 2 | 4 | | |
| | | Code data | L13 | Does not encode data | | encoding does not work properly | | Can not send commands, or commands are not understood by the satellite | 1 | 2 | 2 | | |
| | | Create mission plan | L14 | Does not create mission plan | | Wrong inputs | | satellite does not know what to do/does nothing | 2 | 4 | 8 | | |
| | | Ensure software mechanism | L15 | Can not ensure software update mechanism | | no acknowledgement from satellate when sending update | | Onboard software could be corrupted | 4 | 3 | 12 | Test updatemechanism on ground as if it was the real setup | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

**(N)Place failure modes which does not fit in any other subsystem/ component here**

| Operational mode | Description of operational mode | Function | Failure mode nr. | Failure Mode | Failure detection | Failure cause | Failure effect on other units | Failure effect on main function | Occurence (1-5) | Severity (1-5) | RPN=O*S | Failure minimizin | Notes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | Scheduler | Schedules operations for the day | N1 | Does only work for one orbit |  |  |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |  |  |  | 0 |  |  |
|  |  |  |  |  |  |  |  |  |  |  | 0 |  |  |

Appendix F

# Device Tree Source for the OPU

The following device tree source was decompiled from the final device tree blob generated with Petalinux SDK.

```
/dts-v1/;

/ {
        #address-cells = <0x1>;
        #size-cells = <0x1>;
        compatible = "avnet,picozed", "xlnx,zynq-7000";
        model = "Avnet picoZed";

        cpus {
                #address-cells = <0x1>;
                #size-cells = <0x0>;

                cpu@0 {
                        compatible = "arm,cortex-a9";
                        device_type = "cpu";
                        reg = <0x0>;
                        clocks = <0x1 0x3>;
                        clock-latency = <0x3e8>;
                        cpu0-supply = <0x2>;
                        operating-points = <0x7a120 0xf4240
                         ↪  0x3d090 0xf4240>;
                };

                cpu@1 {
                        compatible = "arm,cortex-a9";
                        device_type = "cpu";
                        reg = <0x1>;
                        clocks = <0x1 0x3>;
                };
        };

        fpga-full {
                compatible = "fpga-region";
                fpga-mgr = <0x3>;
                #address-cells = <0x1>;
                #size-cells = <0x1>;
                ranges;
        };

        pmu@f8891000 {
                compatible = "arm,cortex-a9-pmu";
                interrupts = <0x0 0x5 0x4 0x0 0x6 0x4>;
                interrupt-parent = <0x4>;
                reg = <0xf8891000 0x1000 0xf8893000
                 ↪  0x1000>;
        };

        fixedregulator {
                compatible = "regulator-fixed";
                regulator-name = "VCCPINT";
                regulator-min-microvolt = <0xf4240>;
                regulator-max-microvolt = <0xf4240>;
                regulator-boot-on;
                regulator-always-on;
                phandle = <0x2>;
        };

        amba {
                u-boot,dm-pre-reloc;
                compatible = "simple-bus";
                #address-cells = <0x1>;
                #size-cells = <0x1>;
                interrupt-parent = <0x4>;
                ranges;

                adc@f8007100 {
                        compatible =
                         ↪  "xlnx,zynq-xadc-1.00.a";
                        reg = <0xf8007100 0x20>;
                        interrupts = <0x0 0x7 0x4>;
                        interrupt-parent = <0x4>;
                        clocks = <0x1 0xc>;
```

```
};

can@e0008000 {
        compatible = "xlnx,zynq-can-1.0";
        status = "okay";
        clocks = <0x1 0x13 0x1 0x24>;
        clock-names = "can_clk", "pclk";
        reg = <0xe0008000 0x1000>;
        interrupts = <0x0 0x1c 0x4>;
        interrupt-parent = <0x4>;
        tx-fifo-depth = <0x40>;
        rx-fifo-depth = <0x40>;
};

can@e0009000 {
        compatible = "xlnx,zynq-can-1.0";
        status = "disabled";
        clocks = <0x1 0x14 0x1 0x25>;
        clock-names = "can_clk", "pclk";
        reg = <0xe0009000 0x1000>;
        interrupts = <0x0 0x33 0x4>;
        interrupt-parent = <0x4>;
        tx-fifo-depth = <0x40>;
        rx-fifo-depth = <0x40>;
};

gpio@e000a000 {
        compatible = "xlnx,zynq-gpio-1.0";
        #gpio-cells = <0x2>;
        clocks = <0x1 0x2a>;
        gpio-controller;
        interrupt-controller;
        #interrupt-cells = <0x2>;
        interrupt-parent = <0x4>;
        interrupts = <0x0 0x14 0x4>;
        reg = <0xe000a000 0x1000>;
        emio-gpio-width = <0x40>;
        gpio-mask-high = <0x0>;
        gpio-mask-low = <0x5600>;
        phandle = <0x8>;
};

i2c@e0004000 {
        compatible = "cdns,i2c-r1p10";
        status = "disabled";
```

```
		clocks = <0x1 0x26>;
		interrupt-parent = <0x4>;
		interrupts = <0x0 0x19 0x4>;
		reg = <0xe0004000 0x1000>;
		#address-cells = <0x1>;
		#size-cells = <0x0>;
	};

	i2c@e0005000 {
		compatible = "cdns,i2c-r1p10";
		status = "disabled";
		clocks = <0x1 0x27>;
		interrupt-parent = <0x4>;
		interrupts = <0x0 0x30 0x4>;
		reg = <0xe0005000 0x1000>;
		#address-cells = <0x1>;
		#size-cells = <0x0>;
	};

	interrupt-controller@f8f01000 {
		compatible = "arm,cortex-a9-gic";
		#interrupt-cells = <0x3>;
		interrupt-controller;
		reg = <0xf8f01000 0x1000 0xf8f00100
		↪  0x100>;
		num_cpus = <0x2>;
		num_interrupts = <0x60>;
		phandle = <0x4>;
	};

	cache-controller@f8f02000 {
		compatible = "arm,pl310-cache";
		reg = <0xf8f02000 0x1000>;
		interrupts = <0x0 0x2 0x4>;
		arm,data-latency = <0x3 0x2 0x2>;
		arm,tag-latency = <0x2 0x2 0x2>;
		cache-unified;
		cache-level = <0x2>;
	};

	memory-controller@f8006000 {
		compatible = "xlnx,zynq-ddrc-a05";
		reg = <0xf8006000 0x1000>;
	};
```

```
ocmc@f800c000 {
        compatible = "xlnx,zynq-ocmc-1.0";
        interrupt-parent = <0x4>;
        interrupts = <0x0 0x3 0x4>;
        reg = <0xf800c000 0x1000>;
};

serial@e0000000 {
        compatible = "xlnx,xuartps",
         ↪  "cdns,uart-r1p8";
        status = "disabled";
        clocks = <0x1 0x17 0x1 0x28>;
        clock-names = "uart_clk", "pclk";
        reg = <0xe0000000 0x1000>;
        interrupts = <0x0 0x1b 0x4>;
};

serial@e0001000 {
        compatible = "xlnx,xuartps",
         ↪  "cdns,uart-r1p8";
        status = "okay";
        clocks = <0x1 0x18 0x1 0x29>;
        clock-names = "uart_clk", "pclk";
        reg = <0xe0001000 0x1000>;
        interrupts = <0x0 0x32 0x4>;
        cts-override;
        device_type = "serial";
        port-number = <0x0>;
};

spi@e0006000 {
        compatible = "xlnx,zynq-spi-r1p6";
        reg = <0xe0006000 0x1000>;
        status = "disabled";
        interrupt-parent = <0x4>;
        interrupts = <0x0 0x1a 0x4>;
        clocks = <0x1 0x19 0x1 0x22>;
        clock-names = "ref_clk", "pclk";
        #address-cells = <0x1>;
        #size-cells = <0x0>;
};

spi@e0007000 {
        compatible = "xlnx,zynq-spi-r1p6";
        reg = <0xe0007000 0x1000>;
```

```
                status = "disabled";
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x31 0x4>;
                clocks = <0x1 0x1a 0x1 0x23>;
                clock-names = "ref_clk", "pclk";
                #address-cells = <0x1>;
                #size-cells = <0x0>;
        };

        spi@e000d000 {
                clock-names = "ref_clk", "pclk";
                clocks = <0x1 0xa 0x1 0x2b>;
                compatible = "xlnx,zynq-qspi-1.0";
                status = "okay";
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x13 0x4>;
                reg = <0xe000d000 0x1000>;
                #address-cells = <0x1>;
                #size-cells = <0x0>;
                is-dual = <0x0>;
                num-cs = <0x1>;
                spi-rx-bus-width = <0x4>;
                spi-tx-bus-width = <0x4>;

                flash@0 {
                        compatible = "n25q512a",
                         ↪   "micron,m25p80";
                        reg = <0x0>;
                        #address-cells = <0x1>;
                        #size-cells = <0x1>;
                        spi-max-frequency =
                         ↪   <0x2faf080>;

                        partition@0x00000000 {
                                label = "boot";
                                reg = <0x0
                                 ↪   0x500000>;
                        };

                        partition@0x00500000 {
                                label = "bootenv";
                                reg = <0x500000
                                 ↪   0x20000>;
                        };
                        partition@0x00520000 {
                                label = "kernel";
                                reg = <0x520000
                                 ↪  0xa80000>;
                        };

                        partition@0x00fa0000 {
                                label = "spare";
                                reg = <0xfa0000
                                 ↪  0x0>;
                        };
                };
        };

        memory-controller@e000e000 {
                #address-cells = <0x1>;
                #size-cells = <0x1>;
                status = "disabled";
                clock-names = "memclk", "apb_pclk";
                clocks = <0x1 0xb 0x1 0x2c>;
                compatible = "arm,pl353-smc-r2p1",
                 ↪  "arm,primecell";
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x12 0x4>;
                ranges;
                reg = <0xe000e000 0x1000>;

                flash@e1000000 {
                        status = "disabled";
                        compatible =
                         ↪  "arm,pl353-nand-r2p1";
                        reg = <0xe1000000
                         ↪  0x1000000>;
                        #address-cells = <0x1>;
                        #size-cells = <0x1>;
                };

                flash@e2000000 {
                        status = "disabled";
                        compatible = "cfi-flash";
                        reg = <0xe2000000
                         ↪  0x2000000>;
                        #address-cells = <0x1>;
                        #size-cells = <0x1>;
                };
```

```
};

ethernet@e000b000 {
        compatible = "cdns,zynq-gem",
         ↪   "cdns,gem";
        reg = <0xe000b000 0x1000>;
        status = "okay";
        interrupts = <0x0 0x16 0x4>;
        clocks = <0x1 0x1e 0x1 0x1e 0x1
         ↪   0xd>;
        clock-names = "pclk", "hclk",
         ↪   "tx_clk";
        #address-cells = <0x1>;
        #size-cells = <0x0>;
        phy-mode = "rgmii-id";
        xlnx,ptp-enet-clock = <0x4f790d8>;
        local-mac-address = [00 0a 35 00 1e
         ↪   53];
};

ethernet@e000c000 {
        compatible = "cdns,zynq-gem",
         ↪   "cdns,gem";
        reg = <0xe000c000 0x1000>;
        status = "disabled";
        interrupts = <0x0 0x2d 0x4>;
        clocks = <0x1 0x1f 0x1 0x1f 0x1
         ↪   0xe>;
        clock-names = "pclk", "hclk",
         ↪   "tx_clk";
        #address-cells = <0x1>;
        #size-cells = <0x0>;
};

mmc@e0100000 {
        compatible = "arasan,sdhci-8.9a";
        status = "okay";
        clock-names = "clk_xin", "clk_ahb";
        clocks = <0x1 0x15 0x1 0x20>;
        interrupt-parent = <0x4>;
        interrupts = <0x0 0x18 0x4>;
        reg = <0xe0100000 0x1000>;
        xlnx,has-cd = <0x0>;
        xlnx,has-power = <0x0>;
        xlnx,has-wp = <0x0>;
```

```
                broken-cd;
        };

        mmc@e0101000 {
                compatible = "arasan,sdhci-8.9a";
                status = "okay";
                clock-names = "clk_xin", "clk_ahb";
                clocks = <0x1 0x16 0x1 0x21>;
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x2f 0x4>;
                reg = <0xe0101000 0x1000>;
                xlnx,has-cd = <0x1>;
                xlnx,has-power = <0x0>;
                xlnx,has-wp = <0x0>;
                non-removable;
        };

        slcr@f8000000 {
                u-boot,dm-pre-reloc;
                #address-cells = <0x1>;
                #size-cells = <0x1>;
                compatible = "xlnx,zynq-slcr",
                 ↪   "syscon", "simple-mfd";
                reg = <0xf8000000 0x1000>;
                ranges;
                phandle = <0x5>;

                clkc@100 {
                        u-boot,dm-pre-reloc;
                        #clock-cells = <0x1>;
                        compatible =
                         ↪   "xlnx,ps7-clkc";
                        fclk-enable = <0x1>;
                        clock-output-names =
                            ↪  "armpll", "ddrpll",
                            ↪  "iopll", "cpu_6or4x",
                            ↪  "cpu_3or2x", "cpu_2x",
                            ↪  "cpu_1x", "ddr2x",
                            ↪  "ddr3x", "dci",
                            ↪  "lqspi", "smc", "pcap",
                            ↪  "gem0", "gem1",
                            ↪  "fclk0", "fclk1",
                            ↪  "fclk2", "fclk3",
                            ↪  "can0", "can1",
                            ↪  "sdio0", "sdio1",
                            ↪  "uart0", "uart1",
                            ↪  "spi0", "spi1", "dma",
                            ↪  "usb0_aper",
                            ↪  "usb1_aper",
                            ↪  "gem0_aper",
                            ↪  "gem1_aper",
                            ↪  "sdio0_aper",
                            ↪  "sdio1_aper",
                            ↪  "spi0_aper",
                            ↪  "spi1_aper",
                            ↪  "can0_aper",
                            ↪  "can1_aper",
                            ↪  "i2c0_aper",
                            ↪  "i2c1_aper",
                            ↪  "uart0_aper",
                            ↪  "uart1_aper",
                            ↪  "gpio_aper",
                            ↪  "lqspi_aper",
                            ↪  "smc_aper", "swdt",
                            ↪  "dbg_trc", "dbg_apb";
                        reg = <0x100 0x100>;
                        ps-clk-frequency =
                         ↪  <0x1fca055>;
                        phandle = <0x1>;
                };

                rstc@200 {
                        compatible =
                         ↪  "xlnx,zynq-reset";
                        reg = <0x200 0x48>;
                        #reset-cells = <0x1>;
                        syscon = <0x5>;
                };
```

```
                pinctrl@700 {
                        compatible =
                         ↪  "xlnx,pinctrl-zynq";
                        reg = <0x700 0x200>;
                        syscon = <0x5>;
                };
        };

        dmac@f8003000 {
                compatible = "arm,pl330",
                 ↪  "arm,primecell";
                reg = <0xf8003000 0x1000>;
                interrupt-parent = <0x4>;
                interrupt-names = "abort", "dma0",
                 ↪  "dma1", "dma2", "dma3", "dma4",
                 ↪  "dma5", "dma6", "dma7";
                interrupts = <0x0 0xd 0x4 0x0 0xe
                 ↪  0x4 0x0 0xf 0x4 0x0 0x10 0x4
                 ↪  0x0 0x11 0x4 0x0 0x28 0x4 0x0
                 ↪  0x29 0x4 0x0 0x2a 0x4 0x0 0x2b
                 ↪  0x4>;
                #dma-cells = <0x1>;
                #dma-channels = <0x8>;
                #dma-requests = <0x4>;
                clocks = <0x1 0x1b>;
                clock-names = "apb_pclk";
        };

        devcfg@f8007000 {
                compatible =
                 ↪  "xlnx,zynq-devcfg-1.0";
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x8 0x4>;
                reg = <0xf8007000 0x100>;
                clocks = <0x1 0xc 0x1 0xf 0x1 0x10
                 ↪  0x1 0x11 0x1 0x12>;
                clock-names = "ref_clk", "fclk0",
                 ↪  "fclk1", "fclk2", "fclk3";
                syscon = <0x5>;
                phandle = <0x3>;
        };

        efuse@f800d000 {
                compatible = "xlnx,zynq-efuse";
                reg = <0xf800d000 0x20>;
        };

        timer@f8f00200 {
                compatible =
                 ↪  "arm,cortex-a9-global-timer";
                reg = <0xf8f00200 0x20>;
                interrupts = <0x1 0xb 0x301>;
                interrupt-parent = <0x4>;
                clocks = <0x1 0x4>;
        };

        timer@f8001000 {
                interrupt-parent = <0x4>;
                interrupts = <0x0 0xa 0x4 0x0 0xb
                 ↪  0x4 0x0 0xc 0x4>;
                compatible = "cdns,ttc";
                clocks = <0x1 0x6>;
                reg = <0xf8001000 0x1000>;
        };

        timer@f8002000 {
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x25 0x4 0x0 0x26
                 ↪  0x4 0x0 0x27 0x4>;
                compatible = "cdns,ttc";
                clocks = <0x1 0x6>;
                reg = <0xf8002000 0x1000>;
        };

        timer@f8f00600 {
                interrupt-parent = <0x4>;
                interrupts = <0x1 0xd 0x301>;
                compatible =
                 ↪  "arm,cortex-a9-twd-timer";
                reg = <0xf8f00600 0x20>;
                clocks = <0x1 0x4>;
        };

        usb@e0002000 {
                compatible = "xlnx,zynq-usb-2.20a",
                 ↪  "chipidea,usb2";
                status = "okay";
                clocks = <0x1 0x1c>;
                interrupt-parent = <0x4>;
```

```
                interrupts = <0x0 0x15 0x4>;
                reg = <0xe0002000 0x1000>;
                phy_type = "ulpi";
                dr_mode = "host";
                usb-phy = <0x6>;
        };

        usb@e0003000 {
                compatible = "xlnx,zynq-usb-2.20a",
                 ↪  "chipidea,usb2";
                status = "disabled";
                clocks = <0x1 0x1d>;
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x2c 0x4>;
                reg = <0xe0003000 0x1000>;
                phy_type = "ulpi";
        };

        watchdog@f8005000 {
                clocks = <0x1 0x2d>;
                compatible = "cdns,wdt-r1p2";
                interrupt-parent = <0x4>;
                interrupts = <0x0 0x9 0x1>;
                reg = <0xf8005000 0x1000>;
                timeout-sec = <0xa>;
        };
};

amba_pl {
        #address-cells = <0x1>;
        #size-cells = <0x1>;
        compatible = "simple-bus";
        ranges;

        timer@42800000 {
                clock-frequency = <0x5e69ec0>;
                clock-names = "s_axi_aclk";
                clocks = <0x1 0xf>;
                compatible = "xlnx,axi-timer-2.0",
                 ↪  "xlnx,xps-timer-1.00.a";
                reg = <0x42800000 0x1000>;
                xlnx,count-width = <0x20>;
                xlnx,gen0-assert = <0x1>;
                xlnx,gen1-assert = <0x1>;
                xlnx,one-timer-only = <0x0>;
```

```
                        xlnx,trig0-assert = <0x1>;
                        xlnx,trig1-assert = <0x1>;
                };

                cubedma_top@43c00000 {
                        clock-names = "clk";
                        clocks = <0x1 0xf>;
                        compatible =
                         ↪  "xlnx,cubedma-top-1.0";
                        interrupt-names = "mm2s_irq",
                         ↪  "s2mm_irq";
                        interrupt-parent = <0x4>;
                        interrupts = <0x0 0x1d 0x4 0x0 0x1e
                         ↪  0x4>;
                        reg = <0x43c00000 0x10000>;
                        xlnx,mm2s-comp-width = <0x10>;
                        xlnx,mm2s-num-comp = <0x4>;
                        xlnx,s2mm-comp-width = <0x10>;
                        xlnx,s2mm-num-comp = <0x4>;
                        xlnx,tinymover = "false";
                };
        };

        chosen {
                bootargs = "earlyprintk";
                stdout-path = "serial0:115200n8";
        };

        aliases {
                ethernet0 = "/amba/ethernet@e000b000";
                serial0 = "/amba/serial@e0001000";
                spi0 = "/amba/spi@e000d000";
        };

        memory {
                device_type = "memory";
                reg = <0x0 0x40000000>;
        };

        reserved-memory {
                #address-cells = <0x1>;
                #size-cells = <0x1>;
                ranges;

                buffer@0x30000000 {
                        reg = <0x30000000 0x10000000>;
                        no-map;
                        phandle = <0x7>;
                };
        };

        cubedma@0 {
                memory-region = <0x7>;
        };

        phy0 {
                compatible = "usb-nop-xceiv";
                #phy-cells = <0x0>;
                reset-gpios = <0x8 0x7 0x1>;
                phandle = <0x6>;
        };
};
```

Appendix G

# Startup Script

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          myapp-init
# Required-Start: $ALL
# Should-Start:
# Required-Stop:
# Should-Stop:
# Default-Start:  2 3 5
# Default-Stop:
# Description:    Linux Startup Script
### END INIT INFO
start ()
{
        echo "################################ HYPSO
         ↪  STARTUP SCRIPT BELOW
         ↪  ################################"


        #Set static ip address.
        echo "Setting static ip address."
        ip addr flush dev eth0
        ip addr add 129.241.2.42/23 dev eth0

        echo "mounting media..."
        mkdir -m 755 /media/sd
        mount /dev/mmcblk0p1 /media/sd || mount
         ↪  /dev/mmcblk0 /media/sd
        mkdir -m 755 /media/emmc
        mount /dev/mmcblk1p2 /media/emmc
```

```
echo "setting up swap..."
if mountpoint -q /media/emmc
then
        if test ! -f /media/emmc/swapfile
        then
                dd if=/dev/zero
                 ↪  of=/media/emmc/swapfile bs=1024
                 ↪  count=1048576
        fi
        mkswap /media/emmc/swapfile
        swapon /media/emmc/swapfile
fi
if mountpoint -q /media/sd
then
        if test ! -f /media/sd/swapfile
        then
                dd if=/dev/zero
                 ↪  of=/media/sd/swapfile bs=1024
                 ↪  count=1048576
        fi
        mkswap /media/sd/swapfile
        swapon /media/sd/swapfile
fi
echo "remounting rootfs size to 1G..."
mount -o remount,size=1G /

echo "mounting gpio"
echo 960 > /sys/class/gpio/export

echo "extracting software to rootfs"
tar -xf /software.tar.xz -C /
rm /software.tar.xz

echo "running ueye setup script"
# run ueye setup script
/usr/bin/ueyesetup -i usb
/usr/bin/ueyesetup -i eth

echo "starting ueye drivers"
#start eueye ethernet & usb
/etc/init.d/ueyeethdrc start
/etc/init.d/ueyeusbdrc start

echo "resetting uboot environment (bootcounter)"
```

```
        flash_eraseall -j /dev/mtd1

        echo "starting cubeDMA driver"
        insmod
         ↪  /lib/modules/4.19.0-xilinx-v2019.1/extra/cubedma.ko

        #&&&#
        echo "starting opu-services from /home/root".
        if mountpoint -q /media/sd
        then
                /home/root/opu-services 12 can0 -m
                 ↪  /media/sd/
                echo "opu-services exited... restarting
                 ↪  from flash..".
                /media/sd/opu-services 12 can0 -m
                 ↪  /media/sd/
        else
                /home/root/opu-services 12 can0 -m
                 ↪  /media/emmc/
                echo "opu-services exited... restarting
                 ↪  from flash..".
                /media/emmc/opu-services 12 can0 -m
                 ↪  /media/emmc/
        fi

        echo "THIS TEXT SHOULD BE REPLACED WITH A REBOOT
         ↪  COMMAND."

}

stop ()
{


echo "Bye, bye hypso."
}
restart()
{
stop
start
}
case "$1" in
start)
start; ;;
stop)
```

```
stop; ;;
restart)
restart; ;;
*)
echo "Usage: $0 {start|stop|restart}"
exit 1
esac
exit $?
```

Appendix H

# Dockerfile for setting up the Petalinux SDK

```
FROM ubuntu:18.04

# build with docker build --build-arg PETA_VERSION=2019.1
 ↪  --build-arg
 ↪  PETA_RUN_FILE=petalinux-v2019.1-final-installer.run -t
 ↪  petalinux:2019.1 .

ENV DEBIAN_FRONTEND=noninteractive

# Install PetaLinux Installer Dependences:
RUN dpkg --add-architecture i386 && apt-get update &&
 ↪  apt-get install -y \
  build-essential \
  sudo \
  tofrodos \
  iproute2 \
  gawk \
  net-tools \
  expect \
  libncurses5-dev \
  tftpd \
  libssl-dev \
  flex \
  bison \
  libselinux1 \
  gnupg \
  wget \
  socat \
  gcc-multilib \
  libsdl1.2-dev \
  libglib2.0-dev \
  lib32z1-dev \
  zlib1g:i386 \
  libgtk2.0-0 \
  screen \
  pax \
  diffstat \
  xvfb \
  xterm \
  texinfo \
  gzip \
  unzip \
  cpio \
  chrpath \
  autoconf \
  lsb-release \
  libtool \
  libtool-bin \
  locales \
  kmod \
  git \
  python \
  vim \
  nano

ARG PETA_VERSION
ARG PETA_RUN_FILE

RUN locale-gen en_US.UTF-8 && update-locale

#make a HYPSO user
RUN adduser --disabled-password --gecos '' hypso && \
  usermod -aG sudo hypso && \
  echo "hypso ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

COPY accept-eula.sh ${PETA_RUN_FILE} /

# Run Petalinux installer
RUN chmod a+x /${PETA_RUN_FILE} && \
  mkdir -p /opt/Xilinx && \
  chmod 777 /tmp /opt/Xilinx && \
  cd /tmp && \
  sudo -u hypso /accept-eula.sh /${PETA_RUN_FILE}
   ↪  /opt/Xilinx/petalinux && \
  rm -f /${PETA_RUN_FILE} /accept-eula.sh

# Install tools required by PetaLinux
RUN apt-get install -y \
    ssh rsync

USER hypso
ENV HOME /home/hypso
ENV LANG en_US.UTF-8
WORKDIR /home/hypso/

#add petalinux tools to path
RUN echo "source /opt/Xilinx/petalinux/settings.sh" >>
 ↪  /home/hypso/.bashrc
```

Appendix I

# Filesystem Performance Test Script

```
#/bin/bash
count=$1
block_size=$2

echo 3 | tee /proc/sys/vm/drop_caches
echo "Cache flused."

echo "write 1 $block_size block to SD Card"

time dd if=/dev/zero of=/media/sd-pl/garbage.test
 ↪  bs=$block_size count=$count


echo 3 | tee /proc/sys/vm/drop_caches
echo "Cache flused."

echo "read 1 $block_size block  from SD Card (flushed
 ↪  cache)"
time dd if=/media/sd-pl/garbage.test of=/dev/null
 ↪  bs=$block_size count=$count
echo "read 1 $block_size block from SD Card"
time dd if=/media/sd-pl/garbage.test of=/dev/null
 ↪  bs=$block_size count=$count

rm /media/sd-pl/garbage.test

echo 3 | tee /proc/sys/vm/drop_caches
echo "Cache flused."
echo "write 1 $block_size block file to EMMC."
time dd if=/dev/zero of=/media/emmc-pl/garbage.test
 ↪  bs=$block_size count=$count

echo 3 | tee /proc/sys/vm/drop_caches
echo "Cache flused."

echo "read 1 $block_size block from  EMMC (flushed cache)"
time dd if=/media/emmc-pl/garbage.test of=/dev/null
 ↪  bs=$block_size count=$count
echo "read 1 $block_size block from  EMMC"
time dd if=/media/emmc-pl/garbage.test of=/dev/null
 ↪  bs=$block_size count=$count

rm /media/emmc-pl/garbage.test

echo 3 | tee /proc/sys/vm/drop_caches
echo "Cache flused."

echo "write 1 $block_size block file to rootfs."
time dd if=/dev/zero of=/garbage.test bs=$block_size
 ↪  count=$count

echo 3 | tee /proc/sys/vm/drop_caches
echo "Cache flused."

echo "read 1 $block_size block from rootfs (flushed cache)"
time dd if=/garbage.test of=/dev/null bs=$block_size
 ↪  count=$count
echo "read 1 $block_size block from rootfs"
time dd if=/garbage.test of=/dev/null bs=$block_size
 ↪  count=$count

rm /garbage.test
```

Appendix J

# Filesystem Performance Test Result

| test | SD (ext4) | SD (ext4) cached | SD (FAT) | SD (FAT) cached |
|---|---|---|---|---|
| 1 64MB write | 1.01 | | 1 | |
| 1 64MB read | 0.867 | 0.336 | 0.772 | 0.379 |
| 64 1MB write | 0.76 | | 0.891 | |
| 64 1MB read | 0.575 | 0.237 | 0.572 | 0.247 |
| 64K 1KB write | 2.496 | | 1.79 | |
| 64K 1KB read | 0.686 | 0.646 | 0.687 | 0.663 |
| | | | | |
| 1 1KB write | 0.01 | | 0.011 | |
| 1 1KB read | 0.008 | 0.008 | 0.009 | 0.008 |
| 1 10KB write | 0.009 | | 0.009 | |
| 1 10KB read | 0.009 | 0.008 | 0.009 | 0.009 |
| 1 100KB write | 0.01 | | 0.011 | |
| 1 100KB read | 0.009 | 0.009 | 0.01 | 0.008 |
| 1 1MB write | 0.024 | | 0.026 | |
| 1 1MB read | 0.018 | 0.014 | 0.019 | 0.014 |
| 1 10MB write | 0.144 | | 0.166 | |
| 1 10MB read | 0.108 | 0.064 | 0.123 | 0.057 |
| 1 100MB write | 1.442 | | 5.699 | |
| 1 100MB read | 7.513 | 0.577 | 2.225 | 0.513 |
| | | | | |
| avg MB/s | 60.0563027838599 | 157.50615258409 | 67.22689076 | 148.9526764934 |
| Fastest MB/s | 111.304347826087 | 270.04219409283 | 111.8881119 | 259.1093117409 |
| | | | | |
| avg read MB/s | 90.2255639097744 | 157.50615258409 | 94.53471196 | 148.9526764934 |
| avg write MB/s | 45.0070323488045 | | 52.1597392 | |
| | | | | |
| Fastest read MB/s | 111.304347826087 | | 111.8881119 | |
| Fastest write MB/s | 84.2105263157895 | | 71.82940516 | |

| eMMC (FAT) | eMMC (FAT) cached | rootfs (RAM) | rootfs cached |
|---|---|---|---|
| 1.115 | | 0.443 | |
| 0.773 | 0.365 | 0.317 | 0.37 |
| 0.989 | | 0.332 | |
| 0.572 | 0.237 | 0.222 | 0.22 |
| 1.926 | | 0.845 | |
| 0.681 | 0.65 | 0.57 | 0.595 |
| | | | |
| 0.008 | | 0.008 | |
| 0.008 | 0.008 | 0.008 | 0.008 |
| 0.009 | | 0.008 | |
| 0.008 | 0.008 | 0.008 | 0.008 |
| 0.01 | | 0.009 | |
| 0.009 | 0.008 | 0.008 | 0.008 |
| 0.027 | | 0.015 | |
| 0.017 | 0.014 | 0.013 | 0.014 |
| 0.182 | | 0.076 | |
| 0.108 | 0.064 | 0.056 | 0.065 |
| 4.99 | | 0.686 | |
| 3.113 | 0.566 | 0.543 | 0.511 |
| | | | |
| 63.4081902246 | 153.354632587859 | 140.7108831 | 162.0253165 |
| 111.888111888 | 270.042194092827 | 288.2882883 | 290.9090909 |
| | | | |
| 94.7680157947 | 153.354632587859 | 173.128945 | 162.0253165 |
| 47.6426799007 | | 118.5185185 | |
| | | | |
| 111.888111888 | | 288.2882883 | |
| 64.7118301314 | | 192.7710843 | |

Appendix K

# Dynamic Reconfiguration of FPGA Performance Test Program

```
// usage:
// ./test.o <width> <height> <depth>

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <time.h>

#define CUBEDMA_BASE 0x43C00000
#define SR_DONE_MSK 0x1

#define MM2S_CTRL_REG                0//0x00
#define MM2S_STAT_REG                1//0x04
#define MM2S_ADDR_REG                2//0x08
#define MM2S_CUBE_DIM_REG        3//0x0C
#define MM2S_BLOCK_DIM_REG       4//0x10
#define MM2S_ROW_DIM_REG        5//0x14

#define S2MM_CTRL_REG 8//0x20
```

```
#define S2MM_STAT_REG 9//0x24
#define S2MM_ADDR_REG 10//0x28
#define S2MM_LEN_REG  11//0x2C

#define SEND_PHYS_ADDR 0x30000000
#define RECEIVE_PHYS_ADDR 0x38000000

#define cubedma_RegWrite(offset, val) deviceMem[offset] =
 ↪  val;
#define cubedma_RegRead(offset) deviceMem[offset]

char *p;
int num;


typedef struct
{
        uint8_t error:1;
        uint8_t complete:1;
} cubedma_init_enable_irq_t;

typedef enum
{
        MM2S,
        S2MM
} transfer_t;

typedef enum {
        SUCCESS,
        ERR_TIMEOUT,
        ERR_BUSY,
        ERR_INV_PARAM
} cubedma_error_t;

typedef struct {
        struct{
                uint32_t source;
                uint32_t destination;
        } address;
        struct {
                uint8_t n_planes;
                uint8_t c_offset;
                uint8_t planewise:1;
                struct {
                        uint8_t enabled:1;
```

```
                                struct {
                                        uint8_t width:4;
                                        uint8_t height:4;
                                        uint32_t size_last_row:20;
                                } dims;
                        } blocks;
                        struct {
                                uint16_t width:12;
                                uint16_t height:12;
                                uint16_t depth:12;
                                uint32_t size_row:20;
                        } dims;
                } cube;
                struct {
                        cubedma_init_enable_irq_t mm2s;
                        cubedma_init_enable_irq_t s2mm;
                } interrupt_enable;
} cubedma_init_t;

#define CTRL_REG_OFFSET(mode) (mode==MM2S)? \
         (MM2S_CTRL_REG): \
        (S2MM_CTRL_REG)

#define STAT_REG_OFFSET(mode) (mode==MM2S)? \
        (MM2S_STAT_REG): \
        (S2MM_STAT_REG)

volatile uint8_t * send_channel;
volatile uint8_t * receive_channel;

static int fd_send;
static int fd_receive;

volatile uint32_t* deviceMem;

int cubedma_TransferDone(transfer_t transfer)
{
        if (cubedma_RegRead(STAT_REG_OFFSET(transfer)) &
         ↪  SR_DONE_MSK)
        {
                if (transfer == S2MM)
                {
                        printf("received length %u\n",
                         ↪  deviceMem[S2MM_LEN_REG]);
                }
                return 1;
        }
        return 0;
}

int gettime(struct timeval  t0, struct timeval t1)
{
        return ((t1.tv_sec - t0.tv_sec) * 1000.0f +
         ↪  (t1.tv_usec -t0.tv_usec) / 1000.0f);
}

int main(int argc, char **argv) {
    // Get cubesize parameters defined by user.
        const int width = strtol(argv[1], &p, 10);
        const int height = strtol(argv[2], &p, 10);
        const int depth = strtol(argv[3], &p, 10);
        const int cube_size = (depth * height * width);

        cubedma_init_t param =
            {
                .address =
                {
                    .source      =
                     ↪  (uint32_t)(SEND_PHYS_ADDR),
                    .destination =
                     ↪  (uint32_t)(RECEIVE_PHYS_ADDR)
                },
                .cube =
                {
                    .n_planes  = 0,
                    .c_offset  = 0,
                    .planewise = 0,
                    .blocks    =
                    {
                        .enabled = 0,
                        .dims = { 0, 0, 0 }
                    },
                    .dims =
                    {   //TODO: Get the numbers the camera
                     ↪  uses
                        .width      = width, //Number of
                         ↪  frames
                        .height     = height, //Number of
                         ↪  rows
                        .depth      = depth, //Number of
                         ↪  cols
                        .size_row    = height*width
                    }
                },
                .interrupt_enable =
                {
                    {0, 0}, {0, 0}
                }
            };

        int fd = open("/dev/mem", O_RDWR|O_SYNC);

        if (fd < 0)
        {
                perror("/dev/mem");
                exit(-1);
        }

        fd_send = open("/dev/cubedmasend", O_RDWR);
          if(fd_send < 1)
          {
                  printf("Unable to open CubeDMA send
                   ↪  channel");
          }

           fd_receive = open("/dev/cubedmarecieve",
            ↪  O_RDWR);
    if (fd_receive < 1) {
      printf("Unable to open receive channel");
    }

    send_channel = mmap(0, cube_size*sizeof(uint8_t),
                    PROT_READ | PROT_WRITE, MAP_SHARED,
                     ↪  fd_send, SEND_PHYS_ADDR);

    receive_channel = mmap(0, cube_size*sizeof(uint8_t),
                    PROT_READ | PROT_WRITE, MAP_SHARED,
                     ↪  fd_receive, RECEIVE_PHYS_ADDR);

         deviceMem = (uint32_t *) mmap(NULL, getpagesize(),
             ↪  PROT_READ|PROT_WRITE, MAP_SHARED, fd,
             ↪  CUBEDMA_BASE);

        if(receive_channel == NULL) {
                perror("fuck");
                exit(-1);
        }


        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;
        printf("GENERATING TEST DATA \n");
        for(int i = 0; i < cube_size; i++) {
                if (i%2)
                send_channel[i] = 0x1;
                else
                send_channel[i] = 0x0;
        }

        struct timeval t1, t0;





        ioctl(fd_send, 0);

        printf("Starting FPGA full reconfig test\n");
        gettimeofday(&t0, NULL);
        system("fpgautil -b
         ↪  /media/sd-img/fpga/image_processing_light.bit.bin");
        printf("Configure cubeDMA \n");
         deviceMem[MM2S_ADDR_REG] = param.address.source;
        deviceMem[MM2S_CUBE_DIM_REG] =
                (param.cube.dims.width & 0xFFF) << 0 |
                (param.cube.dims.height & 0xFFF) << 12 |
                (param.cube.dims.depth & 0xFF) << 24;
        deviceMem[MM2S_BLOCK_DIM_REG] = 0x0;
        deviceMem[MM2S_ROW_DIM_REG] =
↪   param.cube.dims.size_row;
        deviceMem[S2MM_ADDR_REG] =
↪   param.address.destination;
        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;

        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;
        memcpy(send_channel, receive_channel,
↪   cube_size*sizeof(uint8_t));
```

```
        system("fpgautil -b
↪   /media/sd-img/fpga/image_processing_dark.bit.bin");
        printf("Configure cubeDMA \n");
         deviceMem[MM2S_ADDR_REG] = param.address.source;
        deviceMem[MM2S_CUBE_DIM_REG] =
                (param.cube.dims.width & 0xFFF) << 0 |
                (param.cube.dims.height & 0xFFF) << 12 |
                (param.cube.dims.depth & 0xFF) << 24;
        deviceMem[MM2S_BLOCK_DIM_REG] = 0x0;
        deviceMem[MM2S_ROW_DIM_REG] =
↪   param.cube.dims.size_row;
        deviceMem[S2MM_ADDR_REG] =
↪   param.address.destination;

        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;
        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;
        memcpy(send_channel, receive_channel,
↪   cube_size*sizeof(uint8_t));
        system("fpgautil -b
↪   /media/sd-img/fpga/image_processing_light.bit.bin");
        printf("Configure cubeDMA \n");
         deviceMem[MM2S_ADDR_REG] = param.address.source;
        deviceMem[MM2S_CUBE_DIM_REG] =
                (param.cube.dims.width & 0xFFF) << 0 |
                (param.cube.dims.height & 0xFFF) << 12 |
                (param.cube.dims.depth & 0xFF) << 24;
        deviceMem[MM2S_BLOCK_DIM_REG] = 0x0;
        deviceMem[MM2S_ROW_DIM_REG] =
         ↪   param.cube.dims.size_row;
        deviceMem[S2MM_ADDR_REG] =
         ↪   param.address.destination;

        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;
        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;
        memcpy(send_channel, receive_channel,
         ↪   cube_size*sizeof(uint8_t));
        system("fpgautil -b
         ↪   /media/sd-img/fpga/image_processing_dark.bit.bin");
        printf("Configure cubeDMA \n");
         deviceMem[MM2S_ADDR_REG] = param.address.source;
        deviceMem[MM2S_CUBE_DIM_REG] =
                (param.cube.dims.width & 0xFFF) << 0 |
                (param.cube.dims.height & 0xFFF) << 12 |
                (param.cube.dims.depth & 0xFF) << 24;
        deviceMem[MM2S_BLOCK_DIM_REG] = 0x0;
        deviceMem[MM2S_ROW_DIM_REG] =
↪   param.cube.dims.size_row;
        deviceMem[S2MM_ADDR_REG] =
↪   param.address.destination;

        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;
        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;

    gettimeofday(&t1, NULL);
        double time_spent_fpga = gettime(t0, t1);
        printf("\n fpga done \n");

        memcpy(send_channel, receive_channel,
         ↪  cube_size*sizeof(uint8_t));

        printf("Starting FPGA partial reconfig test\n");
        gettimeofday(&t0, NULL);
    system("fpgautil -b
     ↪  /media/sd-img/fpga/image_processing_light_pblock_image_processin
     ↪  -f Partial");
        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;
        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;
        memcpy(send_channel, receive_channel,
         ↪  cube_size*sizeof(uint8_t));
        system("fpgautil -b
         ↪  /media/sd-img/fpga/image_processing_dark_pblock_image_proces
         ↪  -f Partial");
        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;
        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;
        memcpy(send_channel, receive_channel,
         ↪  cube_size*sizeof(uint8_t));
        system("fpgautil -b
         ↪  /media/sd-img/fpga/image_processing_light_pblock_image_proce
         ↪  -f Partial");
        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;
        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;
        memcpy(send_channel, receive_channel,
         ↪  cube_size*sizeof(uint8_t));
        system("fpgautil -b
         ↪  /media/sd-img/fpga/image_processing_dark_pblock_image_proces
         ↪  -f Partial");
        deviceMem[MM2S_CTRL_REG] = 0x1;
        deviceMem[S2MM_CTRL_REG] = 0x1;
        while(!cubedma_TransferDone(MM2S));
        while(!cubedma_TransferDone(S2MM));
        deviceMem[MM2S_CTRL_REG] = 0x0;
        deviceMem[S2MM_CTRL_REG] = 0x0;

    gettimeofday(&t1, NULL);
        printf("\n fpga done \n");
        double time_spent_fpga_part = gettime(t0, t1);


        for(int i = 0; i < cube_size; i++) {
                if (i%2)
                send_channel[i] = 0x1;
                else
                send_channel[i] = 0x0;
        }


        printf("\n Starting SOFTWARE test \n");
        gettimeofday(&t0, NULL);
        for(int i = 0; i < cube_size; i++) {
                send_channel[i] = send_channel[i]*2;
        }
        for(int i = 0; i < cube_size; i++) {
                send_channel[i] = send_channel[i]/2;
        }
        for(int i = 0; i < cube_size; i++) {
                send_channel[i] = send_channel[i]*2;
        }
        for(int i = 0; i < cube_size; i++) {
                send_channel[i] = send_channel[i]/2;
        }
        gettimeofday(&t1, NULL);
        printf("\n software done \n");
        double time_spent_sw = gettime(t0, t1);

        printf("\n comparing result \n");
        for (int i = 0; i < cube_size; i++) {

                if(receive_channel[i] != send_channel[i]) {
                        printf("not same result for %d.
                         ↪  send: %u, reseived: %u \n", i,
                         ↪  send_channel[i],
                         ↪  receive_channel[i]);
                        break;
                }
        }

        printf("FPGA (full): %f \n FPGA (partial): %f \n
         ↪  SW: %f \n", time_spent_fpga,
         ↪  time_spent_fpga_part, time_spent_sw);


        return 0;
}
```

Appendix L

# U-Boot environment Fallback Test and Test Results

```
U-Boot 2019.01 (May 25 2020 - 11:50:29 +0000) Xilinx Zynq
 ↪  ZC702

CPU:   Zynq 7z030
Silicon: v3.1
Model: Avnet picoZed
DRAM:  ECC disabled 1 GiB
MMC:   mmc@e0100000: 0, mmc@e0101000: 1
Loading Environment from SPI Flash... SF: Detected n25q128
 ↪  with page size 256 Bytes, erase size 64 KiB, total 16
 ↪  MiB
OK
In:    serial@e0001000
Out:   serial@e0001000
Err:   serial@e0001000
Model: Avnet picoZed
Net:   ZYNQ GEM: e000b000, phyaddr ffffffff, interface
 ↪  rgmii-id
eth0: ethernet@e000b000
Saving Environment to SPI Flash... SF: Detected n25q128
 ↪  with page size 256 Bytes, erase size 64 KiB, total 16
 ↪  MiB
Erasing SPI flash...Writing to SPI flash...done
OK
Warning: Bootlimit (5) exceeded. Using altbootcmd.
Hit any key to stop autoboot:  0
Zynq> sf probe 0 0 0
```

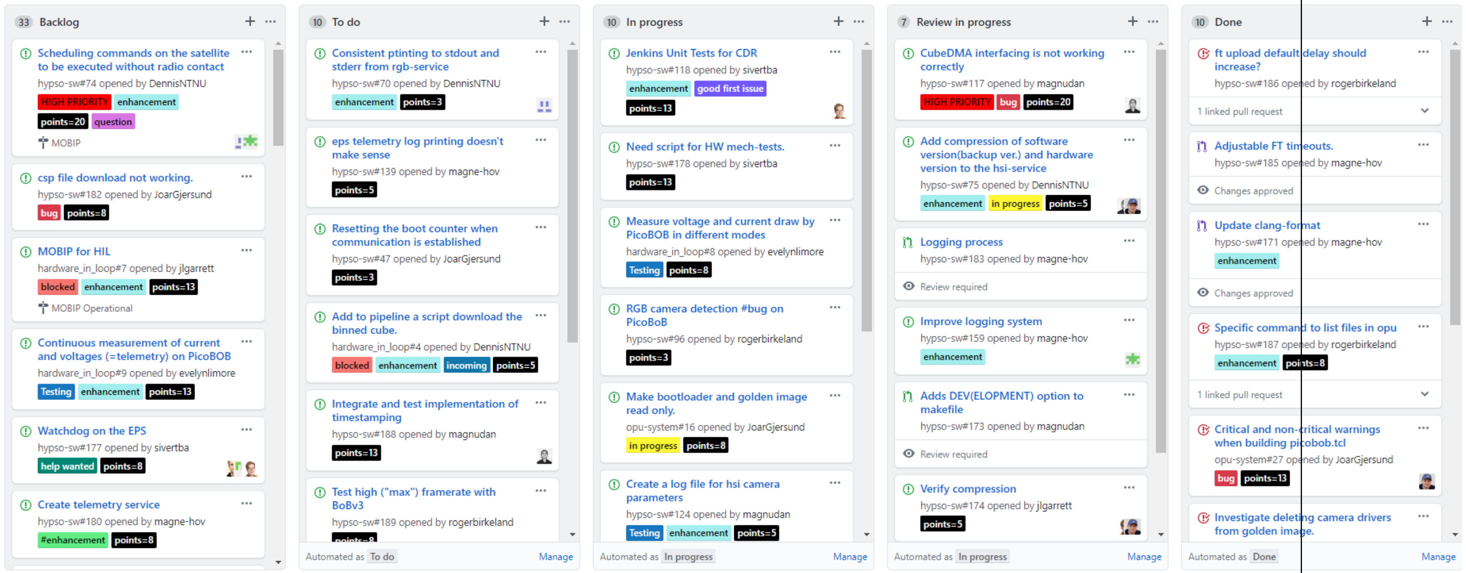
```
SF: Detected n25q128 with page size 256 Bytes, erase size
 ↪  64 KiB, total 16 MiB
Zynq> sf erase 0x600000 +0x10
SF: 65536 bytes @ 0x600000 Erased: OK
Zynq> reset
resetting ...


U-Boot 2019.01 (May 25 2020 - 11:50:29 +0000) Xilinx Zynq
 ↪  ZC702

CPU:   Zynq 7z030
Silicon: v3.1
Model: Avnet picoZed
DRAM:  ECC disabled 1 GiB
MMC:   mmc@e0100000: 0, mmc@e0101000: 1
Loading Environment from SPI Flash... SF: Detected n25q128
 ↪  with page size 256 Bytes, erase size 64 KiB, total 16
 ↪  MiB
*** Warning - bad CRC, using default environment
```

# Appendix M

# Snapshop of Kanban Board



33 Backlog

- Scheduling commands on the satellite to be executed without radio contact
  hypso-sw#74 opened by DennisNTNU
  HIGH PRIORITY enhancement points=20 question
  MOBIP
- csp file download not working.
  hypso-sw#182 opened by JoarGjersund
  bug points=8
- MOBIP for HIL
  hardware_in_loop#7 opened by jlgarrett
  blocked enhancement points=13
  MOBIP Operational
- Continuous measurement of current and voltages (=telemetry) on PicoBOB
  hardware_in_loop#9 opened by evelynlimore
  Testing enhancement points=13
- Watchdog on the EPS
  hypso-sw#177 opened by sivertba
  help wanted points=8
- Create telemetry service
  hypso-sw#180 opened by magne-hov
  #enhancement points=8

10 To do

- Consistent ptinting to stdout and stderr from rgb-service
  hypso-sw#70 opened by DennisNTNU
  enhancement points=3
- eps telemetry log printing doesn't make sense
  hypso-sw#139 opened by magne-hov
  points=5
- Resetting the boot counter when communication is established
  hypso-sw#47 opened by JoarGjersund
  points=3
- Add to pipeline a script download the binned cube.
  hardware_in_loop#4 opened by DennisNTNU
  blocked enhancement incoming points=5
- Integrate and test implementation of timestamping
  hypso-sw#188 opened by magnudan
  points=13
- Test high ("max") framerate with BoBv3
  hypso-sw#189 opened by rogerbirkeland
  points=8

Automated as To do — Manage

10 In progress

- Jenkins Unit Tests for CDR
  hypso-sw#118 opened by sivertba
  enhancement good first issue points=13
- Need script for HW mech-tests.
  hypso-sw#178 opened by sivertba
  points=13
- Measure voltage and current draw by PicoBOB in different modes
  hardware_in_loop#8 opened by evelynlimore
  Testing points=8
- RGB camera detection #bug on PicoBoB
  hypso-sw#96 opened by rogerbirkeland
  points=3
- Make bootloader and golden image read only.
  opu-system#16 opened by JoarGjersund
  in progress points=8
- Create a log file for hsi camera parameters
  hypso-sw#124 opened by magnudan
  Testing enhancement points=5

Automated as In progress — Manage

7 Review in progress

- CubeDMA interfacing is not working correctly
  hypso-sw#117 opened by magnudan
  HIGH PRIORITY bug points=20
- Add compression of software version(backup ver.) and hardware version to the hsi-service
  hypso-sw#75 opened by DennisNTNU
  enhancement in progress points=5
- Logging process
  hypso-sw#183 opened by magne-hov
  Review required
- Improve logging system
  hypso-sw#159 opened by magne-hov
  enhancement
- Adds DEV(ELOPMENT) option to makefile
  hypso-sw#173 opened by magnudan
  Review required
- Verify compression
  hypso-sw#174 opened by jlgarrett
  points=5

Automated as In progress — Manage

10 Done

- ft upload default delay should increase?
  hypso-sw#186 opened by rogerbirkeland
  1 linked pull request
- Adjustable FT timeouts.
  hypso-sw#185 opened by magne-hov
  Changes approved
- Update clang-format
  hypso-sw#171 opened by magne-hov
  enhancement
  Changes approved
- Specific command to list files in opu
  hypso-sw#187 opened by rogerbirkeland
  enhancement points=8
  1 linked pull request
- Critical and non-critical warnings when building picobob.tcl
  opu-system#27 opened by JoarGjersund
  bug points=13
- Investigate deleting camera drivers from golden image.

Automated as Done — Manage