Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Electrónica

Improvement of small satellite's software design with build system and continuous integration tools

para optar por el título de
Ingeniero en Electrónica con énfasis en sistemas empotrados
con el grado académico de
Maestría

Allan Granados
allangj1618@gmail.com
Cartago, Diciembre, 2015

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Tesis de Maestría
Tribunal evaluador

Tesis de maestría defendida ante el presente Tribunal Evaluador como requisito para optar por el grado académico de maestría, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Laura Cabrera Quirós,
M.Sc.
Profesor lector

Ing. Miguel Ángel Aguilar Ulloa,
M.Sc.
Profesor lector

Ing. Johan Carvajal Godínez,
M.Eng.
Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Países Bajos y Alemania,
08 de Diciembre 2015

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Tesis de Maestría

Tribunal evaluador

Acta de evaluación

Tesis de maestría defendida ante el presente Tribunal Evaluador como requisito para optar por el grado académico de maestría, del Instituto Tecnológico de Costa Rica.

Estudiante: Allan Granados Jiménez

Nombre del Proyecto: "Improvement of small satellite's software design with build system and continuous integration tools"

Miembros del Tribunal

Ing. Laura Cabrera Quirós,
M.Sc.
Profesor lector

Ing. Miguel Ángel Aguilar Ulloa,
M.Sc.
Profesor lector

Ing. Johan Carvajal Godínez,
M.Eng.
Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica

Nota Final de tesis :80

Países Bajos y Alemania, 08 de Diciembre de 2015

Contents

List of Figures

List of Tables

Abstract

The space exploration is a field that requires the interconnection of different research disciplines including medicine, biology, physics and of course electronics and embedded systems. Because of this integration of different disciplines, the development of software for this system can become a challenge. Because of this, it is important to have a common way to introduce pieces of software without alter or risk the deployment of the whole system. This work is focus on the design, development, and result data recollection of a deployment system of software focus on the space exploration field. It implement the Yocto project as its main construction system, by which the user can include and/or customize different pieces of software reducing the dependencies between different modules. It is also part of this work, the implementation of a continuous integration system (CI), in charge of reviewing and reporting the status of the final deployed image. This CI system also recollects important information about the construction of the deployed image as well as its stability during the project development. Results of the deployed images were tested on different development platforms.
Keywords:
Yocto, Linux, OBC, SDR, UML, Stakeholder, Continous Integration, Jenkins.

Chapter 1

Introduction

The space exploration is the field which study outer space by using scientific equipment. It requires the interconnection of different research disciplines including medicine, biology, physics and of course electronics and embedded systems. Space exploration is not new, for centuries the human race have dreamed about space flights and what is beyond. Since antiquity the Chinese used rockets for ceremonial and military purposes, but it was not until the last hundred years when the efforts were focus into a new frontier, the space beyond the Earth. Overcome the Earth gravity was not an easy task, but it led to an investigation race in the early 20th century between powerful nations as Germany, Russia and the United States. It was until October 4, 1957, when the Soviet Union accomplish the first successful orbital launch, the first satellite, called Sputnik 1. The success of the Soviet satellite program, led to an escalation of their competitors in order to achieve the same results. Further development in the related fields help to improve the scope of the different missions, which in the end, led to many scientific discoveries, including important research in the electronics field and communications. But why is space exploration so important?. Space exploration is important because as humans it helps to address fundamental question about our existence, our place in the Universe and the history of the universe around us. It help to expand our technological advance in different fields and speed up the communication of this advances to all human race.

The space exploration goes from sending a remote controlled or autonomous device to outer space, to planetary exploration with manned missions. The present work focus on the devices placed into orbit in order to recollect and transmit information, commonly known as satellites. Because outer space is a hazardous environment, the exploration become a difficult task. The implementation of low cost, low power and simple embedded systems becomes an ideal solution, reducing the risk of accidents and speeding up the recollection of data. The construction of such devices become an important topic, which comprehend hardware and software development. The methodologies and implementation of the software for satellites are investigated in the present work,

seeking for an implementation which improve the quality of software and speed up the development process.

The development of software for small satellites is a challenge, it includes the budget of the project, the experience of the team members, the development time and many others [9]. This challenges combines with the need of stable, efficient and reliable software, making the task even harder. Because of this, the need for a well defined design, development and construction process is required. It is a common problem on the small satellites projects, the lack of reusable and portable software, which usually is wrap in a monolithic design, making it hard to be of use in other missions in order to speed up the development or even take advantages of experience obtained in the past. If the design and development method is defined and consistent, it will help to accurate schedule the project, take advantages of the experience and reuse pieces of stable and efficient software. A good and accurate schedule in spacial exploration projects can result in the execution of the project or its cancellation, specially in a field were different investigation groups are racing for the same objective[18].

In order to achieve a consistent design and development process, the use of well know tools and methodologies can be incorporate in the project. These tools can be from the Open Source community or licensed product for a vendor. Is a common fear around developers that sometimes, good stable code is only found on commercial software, while open source code fluctuates with latest releases and improvements [24]. In reality, stable and reliable packages, tools and code can be found in either commercial or Open Source products, it depends on the developers integrating the product, to select those projects which demonstrate its stability. An example of a Open Source project which has proven its stability is the Yocto project. It balances between the dynamics of the Open Source cooperative development and the commercial requirements of time line, cost effective and stable releases [24]. This work present the development of a software construction project, using Yocto as a central build system. It takes advantage of the layer methodology to separate different packages in different levels of abstraction, where platform dependent and independent code can be separate in order to increase re-utilization. A central configuration system is designed and implemented in order to fetch, configure and construct the software in a way users not familiar to build systems or Yocto can use it and obtain an image or an SDK for a specific platform in a easy way.

1.1 Previous work focus on small satellites

The space exploration projects has rapidly increase in the last few years. Several space exploration missions has been developed using small satellites. Those missions are designed to recollect important data, not only about space, but about the project development itself. The idea is to give feedback about challenges, opportunities and what to have into account for a successful mission.

- UPMSat-2 [3]: Project aimed to develop a micro-satellite that can be used as a demonstration for several research groups. This project uses a LEON

family processor for the OBC, which is a 32 bit synthesizable VHDL processor core from SPARC V8 architecture. The design of the project is created on a high level graphical model to capture the different components, which finally are traduced into ADA. This project focuses on the analysis of the WCET (Wort's Case Execution Time) for the performance of the system. It shows the power in the uses of a VHDL specialized processor for a space exploration mission and its flexibility.

- ION Project [9]: The ION satellite are a two micro-satellite constructed by the University of Illinois. The main purpose of the project is to provide a large interdisciplinary educational project for engineer students. On this project is researched the use of off of the shell components and the difficulties on the satellite designs like cost and development time. This project implements several external components like cameras and other sensors and was an evolving project giving more than one implementation of a satellite. It is shown how the integration of different components is a very important part of the development and how it can affect the release for the final satellite.

- RinconSat [13]: This project was developed for the study of a RTOS for small satellites by the University of Arizona. The OBC used was base on a PIC16C77 micro-controller from Microchip Corporation. This project reveals the OS limitation base on the available memory, and for that reason a simple serial scheduling process was supported. Although the development time of the project was short and the cost of the satellites was very low, the hardware limitations reveal the lack of scalability for future missions using this satellite design.

- DANDE [18]: Started as a student developed nano-satellite project by the Colorado Space Grant Consortium, its main mission was to study the effects of space weather and atmospheric drag on a small satellite platform. They bring the use of a more complex OS, the uClinux version for microcontrollers. However this version of Linux brings the need for good data and program separation, because its designed for systems where a memory management unit (MMU) its not present. They implement a simple single control loop, where a process checks the memory integrity in case of cosmic rays memory corruption. Its shows the space exploration community the advantages of a more complex OS implementation and how the problems like data corruption may be overcome.

- AlcaSat [19]: This project was born with the RinconSat project, basing its design on the PIC16C77 and some peripheral ICs. Following the discovers from its brother project, the limitation on hardware bring the need of more peripherals to fulfill requirements, for example the need for external memory to store the program software. It shows how the lack of scalability on the hardware limit the scope of the project and what could be achieved.

- USS Langley [2]: The project objective was to demonstrate the ability to host a web server in a small satellite using TCP/IP internet protocols and be accessible to any internet user. In addition, testing space based networks against terrestrial networks was intended. This project is specially interesting because of the use of a Beagleboard TI development board as the host for the web server. The developers of the project choose the Beagleboard because of its small from factor, customizability and low power requirements. However the kernel of the board was a 2.6 version, been a very old kernel version. In addition, the project integrate several processing units, each one with a specific task, which increment the complexity and power consumption. For example, the Beagleboard, hosting only the web server, consumed 600mA, increasing the power requirements of the small satellite.

Each one of the projects mention before, recollected important data and invaluable experience, however, the implemented software has proven to lack in scalability and in many cases was impossible to adapt into further missions because of its platform dependent nature and the monolithic design of the whole software in the project.

1.2 Problem statement

The design and implementation of software for satellites, even for small satellites, is a time consuming and multi-disciplinary process, which require great effort [15]. The demand for more complex functionality arises day to day, in company of contractions on the development schedules, seeking faster times to market [31]. In addition, the available resources, money, equipment and personal, determine the development time required [19]. Hence, a good methodology and development process is required.

In the development of a space exploration projects, in many cases, the detailed specification of the satellite, or the scope of the mission are not known beforehand, it is necessary to be reverse engineered from the capabilities of the components and the available development vehicles [9]. In other projects, a well defined scope is defined and traduced into a System Requirement Specification (SRS), where the expected behavior and the needs are well documented [26]. On both cases, it is necessary to study the limitations of the design and the development process[13]. If the development process is not flexible enough it can not be re-used or adapted in the presence of changes. The lack of flexibility will require extra development time for its adaptation.

With the available hardware today, greater capacities on the central computer of a satellite can be achieved, given the possibility of using a more robust software [13]. For example, designs using microcontrollers, usually need external memory to store executable programs. Like it is shown on [19], a design using the PIC16F877, with the available hardware today it can be embedded in the same chip. However, with the increase of capabilities and versatility, an

increase of the software complexity and the occurrence of system errors may be present[18], more importantly when new software is incorporated. Therefore, the importance on the development software planning is reflected, which need to account for the possibility of future work [14], incorporate new technology and making development advances for future missions. The actual software development for space exploration mission is shown in figure 1.1.

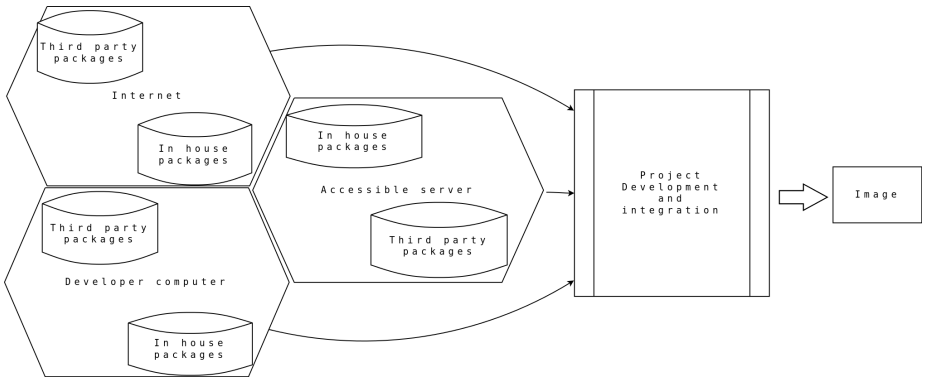

Figure 1.1: Software development methodology for space exploration projects

As shown in figure 1.1, for a specific space exploration project, many software sources are incorporated. However, the construction and integration use to be project specific, showing troubles in scalability and software re-utilization. In addition, when the scope of the project is achieved, the development and integration process is usually discarded. Its shown the necessity of a development process capable of fetching from the available software sources and construct a customize image for the satellite, but with the capability to be use in future space exploration process and incorporate new software.

To recapitulate, several problems on the actual development for small satellites are found, which can be solve by the incorporation of a scalable and reusable development methodology.

1. The increase of the software requirements for small satellites.
2. Increase on the time for software components integration.
3. Reduce of the cost and development time of a project.
4. Reduce the learning curve for new developers.
5. Breach between the design methods and the software implementation.
6. Little credibility on the use of open source tools for software development on space exploration mission.

1.3 Proposed solution

Many challenges have been stated, revealing the hard task of working on the design and development of the small satellite software. Knowing these challenges, is important to structure and implement a development methodology which can be easy to implement by developers, adopt currently available technologies, easy to adapt to the current production process and improve scalability and re-usability. Because of this, the implementation of a construction system, using the Yocto project as its internal build system is propose to be incorporate into the development process for small satellites. This construction system seeks to separate into different layers of abstraction different software packages, improving modularization and ease the re-usability of software. The use of a continuous integration system like Jenkins is added into the project. The continuous integration system is intended to collect information about the project, check the stability by schedule constructions and ease the access for users to deployed images.

1.3.1 Proposed development

One of the main objectives on this research is to study how the proposed construction system using Yocto as its build system, can be used for the software development of small satellites. In addition, the development of software for different processors and co-processors is study, incorporating the required development tools into the construction systems. On figure 1.2, the proposed construction system is shown.

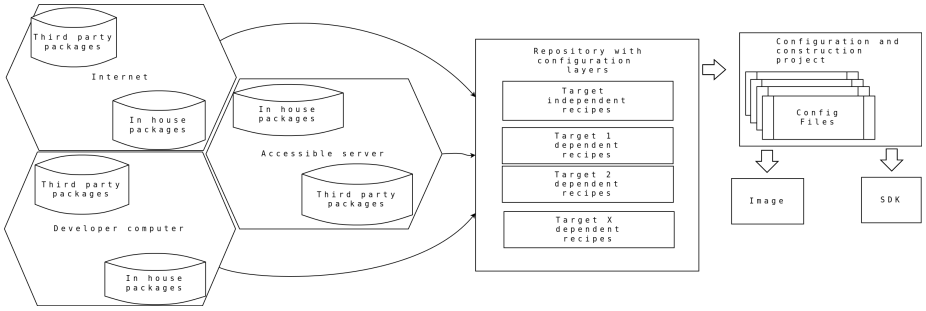

Figure 1.2: Proposed solution for software development on small satellites

The proposed solution shown in figure 1.2, separate into abstraction layers the construction and integration system, using Yocto as the central build system. The different layers are divided into platform dependent and platform independent. The platform independent layer contains software independent of the vehicle. The software contained in this layer can be easy constructed for different projects using different hardware targets, which increase the re-usability

of software. The platform dependent layers contains the specific packages implemented for the small satellite vehicle. It also is capable to perform patches and fine-tuning packages in the layer independent layer, increasing the performance in the specific satellite vehicle.

Yocto project, as the internal build system for the embedded software of a satellite, and the implementation of its layer subsystem, can overcome the problems presented before, and still give support on some of the important development process used until today. For example, Yocto allows the creation and packing of an SDK, which can be use by development teams using off-of-the-shell solution. This will reduce the compatibility risks on the creation of the different pieces of software with the main operating system of the small satellite main computer.

It is also possible to open the door on new ways for innovation without risking the project. It allows an inexperience user to construct an custom software package for a defined target in a very fast and independent way, leaving the integration problem to the build system.

Finally a configuration and construction project is proposed. The main purpose for this project is the easy configuration, selecting from pre-tested configurations the more suited for the selected platform. This configuration fetch the platform specific and platform independent layers and its in the construction process when the packages of software are fetched and constructed. The result output is an specific image or SDK for the selected platform representing the central computer for the small satellite.

Following on this document, chapter two contain a literature review on software methodologies implemented on the software development for space exploration. It discuss the hardware and software structure for small satellites. Chapter include a review of open source technologies. Chapter three explores the integration of the build system to our construction solution. It also discuss the benefits of Yocto as our central build system. Chapter four review the platform used as study for the construction system. On chapter five the design of our solution is implemented. It include the methods to add new pieces of software from the developers. Chapter six show relevant results compiled from the use of the implemented construction system. Chapter seven presents future work on which this research can expand. Finally, in chapter eight, the conclusions of the present work are stated.

Chapter 2

Software development approaches for small satellites

According to Carvajal in [6], the design gap in the field of space explorations have two dimensions, hardware and software. In the present chapter different software methodologies used on the development of software for small satellites are reviewed. Understand how the development process as evolve from the early days of space exploration will help to comprehend the arising needs in the field regarding the software development. Once the review of the actual development are completed, proposing development methodology is possible, one which not only capable to sustain the actual development, but capable to create opportunities to improve and innovate.

2.1 Software methodologies used for satellites design

The development methodology in a project can determine its success or failure. It is the one in charge of trace the design, limit the scope of what needs to be done and administrate the available resources in the best possible manner [23]. The space exploration projects do not escape from this, specially the design of small satellites, which by its size requirements and mission scope, have limited resources, and limited development time.

Because of the high requirements on quality for the satellites design, very little room exist for innovation. This causes to use space proven, though often outdated technologies [9]. Also a lot of money and effort is placed on the development of those systems, but with the limitation exposed, the development of satellites has historically been limited to first world countries with large budgets.

The high integrity of a project is reflected on the standards used. These

regulates the software development due to the high importance of the projects and the components involved on it [31]. The satellite development lacks of a well defined standard. There exist several initiatives for standards, however they are still in the process of acceptance. One example is the presented by the European Cooperation for Space Standardization (ECSS), an institution that release several standard documents for space engineering, like the one shown in [34]. I seek for harmonization of software methodologies for satellites designs. It centers in modularization of pieces of software in order to increase the re-utilization. With the modularization, the quality and stability of the code is increased, been able to corner the pieces of software that may present problems.

Several researches has been done on different methodologies for the design of small satellites, all of those having in common the search for a simple and intelligent way to expose the design and share the knowledge between a development team. Known methods used on other fields has been implemented, for example the use of UML (Unified Modeling Language) diagrams to set the requirements into a design model. The UML language, take advantage of the graphical facility to communicate the design model in order to understand the requirements. Other graphical tool used for the satellite software design is the Matlab tool Simulink [15]. It uses the blocks to describe the movement of information and processing data, giving quick simulation results and allowing the validation of the design.

A very common methodology used on the design of small satellites is the model driven design [33]. It is an approach that structures the specifications and guidelines in different models for its study. In addition, different approaches like the one in [26] are commonly developed. In this approach [26], a new paradigm is introduced by the use of OPEN-SBM modeling (a methodology using graphical interface), using the System Requirements Design (SRD) to generate the development environment.

The University of Southern California made several studies on the optimization of the tool SPIDR, which uses a graphical interface to model the system and test the validity of the model, however the most important part of the project was to transform those models into software in a automated way, something that was not achieved [15]. The use of formal languages for the description of the desire implementation was also approached [11]. This approach was hard to follow by the development teams because of the high level of knowledge required, resulting hard to include new personal

As seen, several approaches as been emerging in the last years, specially with the proliferation of small satellites projects, however, each approach tend to be an specific solution on the necessities of the immediate missions, lacking in scalability for future missions. One thing on common amount the different approaches, is the necessity to separate the software in a modular way. The proposed solution follow this line of though, encapsulating each piece of software as its own. In addition, the encapsulation of platform specific and platform independent code is achieved by the separation in different layers.

2.2 Small satellite design and structure

In order to design software for a satellite is important to get to know the hardware for what it is designed. The small size satellites, often called pico-sats, nano-sats or micro-sats, are generally less than 200kg. Its size goes from a refrigerators to small soda cans [9]. Those small satellites give potential benefits over traditional space satellites, for example, the fast development and the project cost.

Small satellites provide amazing alternatives to the developer community. They are driven by "smaller, faster, better, cheaper, smarter" mentality which allows these satellites to be build with a fraction of time and cost of traditional satellites [9].

A design who gaining a lot of acceptance in the developer community is the Cubesat design. A simple Cubesat is a small satellite, a 10cm cube with the requirement to be less than 1kg of mass [19]. The dimension of the Cubesat are one of the only hard restrictions of the design, which needs at least two dimension of 1U, 1U being 10cm, while the third dimension can be a multiple of 1U, for example 2U of 3U [8]. Nowadays the Cubesat design is the most popular because of its development low cost and structural simplicity. A example Cubesat is shown in figure 2.1.

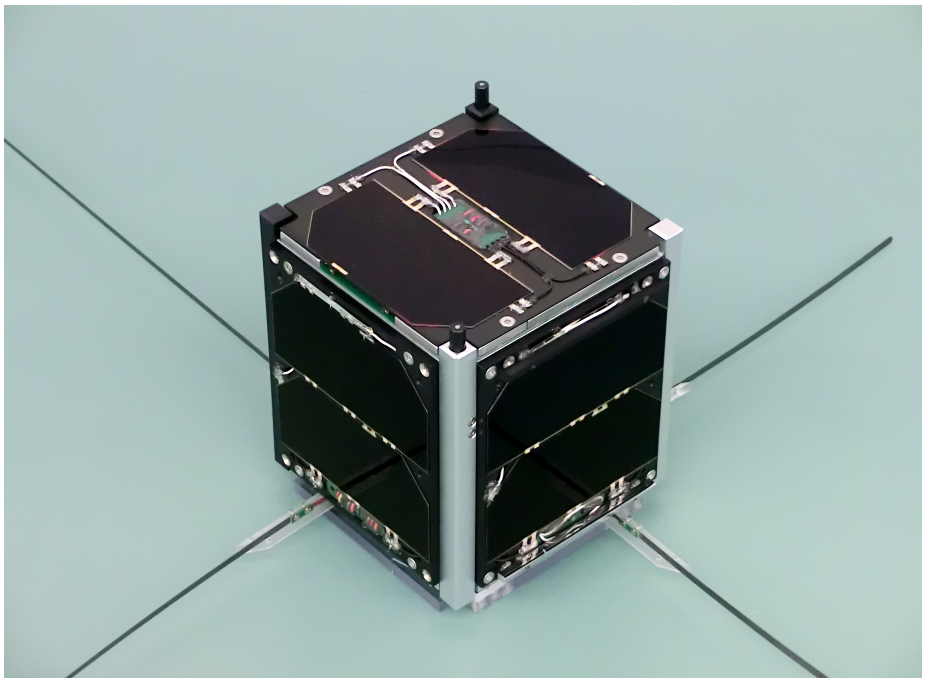

Figure 2.1: Cubesat design example[1]

Small satellites have, as the brain of the mission, an On Board Computer (OBC). The OBC is the one in charge of the main operation of the system, including the control of several subsystems. As mention in [3], the software system of the satellite may be divided in several subsystems.

- Platform monitoring: Is the subsystem in charge of review the state of the satellite by periodically reading important data and comparing it against expected performance values.
- Telemetry and telecommand: Satellites are not completely autonomous systems, they need to communicate the information collected to other station, or receive orders to perform their mission. This subsystem is in charge to interact with the telecommunication hardware in order to send or receive messages.
- Attitude determination and control system: This subsystem is in charge of the orientation of the satellite with respect to the Earth, and if needed, takes corrective actions in order to keep within specific values.

Usually all these tasks are handled and controlled by fully separate systems, but the one in charge of the control and interaction of data is the central processing unit OBC. A good integration in a platform with higher capabilities, may escalate into better performance and more flexibility. In addition the need for modular software components are shown. This modules will directly map the different subsystems of the satellite, maintaining coherency.

2.3 Central computation system in satellites. Homogeneous and Heterogeneous systems

An homogeneous architecture is one with only symmetrical processing units. It has been the simplest approach for small satellites approach, in some cases, the used of only one processing unit is prefer by the developers because of its simplicity. In counter part, the heterogeneous architecture, its a new approach gaining strength in satellites projects because of its versatility. It involve different processing units, each one specialized for a defined task, sharing information sharing information between each others. The development on heterogeneous systems is more complicate, which provoke rejection on the developers without experience, regardless of the possible advantages it may bring. Great debates have been around the use of different architectures in the space exploration projects, but all have one point in common, which relate the correct selection of the architecture to use with the study of the requirements for the final system.

[1]FUNcube-1 CubeSat Satellite in the ISIS clean room in Delft prior to being launched. Image taken by Wouter Weggelaar. FUNcube-1 image is reproduced from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License

An incorrect selection of the architecture may involve the failure of the whole project.

Homogeneous systems are simple, all the processing is performed by one kind of processing unit. The behavior and the management of data is the same, which make the things easier for the developers. The memory coherency is an example of simplicity on homogeneous systems, which has been taken for granted in homogeneous multiprocessor and multi-core systems for decades. Instead, allowing heterogeneous processors with CPU, GPU, DSP, FPGA to maintain coherency in a shared memory environment is a revolutionary concept. In addition to heterogeneous systems complexity in software is the communication between the various processing units[5]. It reveal to be not only a challenge for the software developers but also for the system designers. They need to have in mind not only how to unlock the computational performance in parallel systems by keeping a high integration of the blocks, but also to provide the system with the means of giving the separate blocks a way to access shared memory and data to communicate between each other. As mention before, the communication between different units is a very important and delicate task. As stated in [22], is important to know the path between the sender and the receiver of data, in order to determine if the bandwidth for a communication matches the requirements for an application. If not, it should be redesigned. The energy requirements of the system are very important, and become specially relevant when discussing space exploration projects. Satellites energy subsystems must provide with energy for a large amount of time and with reduce waste. An evaluation algorithm can be implemented in order to optimize the energy consumption of the system [22]. This algorithm will allow to determine if the use of a heterogeneous over a homogeneous system justify the developing effort, however this is left to future research.

For the reasons stated before, the use of heterogeneous systems has not been very common on space exploration systems, because the increase of complexity in the project. This increase of complexity has been present even in some homogeneous systems. For example, the increase on complexity on an application develop in [5] using x86 processors, provoke the lack of performance and make it susceptible to radiation, which leads to discard this kind of architectures for satellites. A 2012 survey by Ramon Chips of processors for high performance space missions space from 2012, describes the still commonly used the platform BAE RAD750 achieving 300 MIPS, which has 10.4 million transistors and its cost is around $200000 USD. Another example from [5] describes the higher end state-of-the-art Proton 200k single board computer using a TI 320C64xx DSP sporting 900MFLOP. These systems are expensive and not compatible in size smallest satellites. By contrast, most small space missions using satellites in the range of 1-10kg today use low performance and low cost parts. The simple microcontroller devices use to be commonly selected. Devices such as 8 bit microcontrollers and lower end RISC architectures are frequently used, but they are not capable of great capacities. Higher performance and capabilities can be achieve by more complex and robust architectures. Advanced FPGAs with built in ARM, PowerPC, Microblaze processors are also commonly used on

space project research, which improves the performance significantly compared to the lower end microcontrollers, but the complexity of the system and the code running on it is increased.

The increase of the complexity in the design for a heterogeneous system is a reality, but its important to have into account conclusions given by [5]. Its necessary to break the old way of thinking were all high integrated systems will be useless against radiation. Its proven that a good design make tolerate the average radiation in space exploration missions. The performance between architectures is also compared by [5], were state-of-the-art processors for space usually performs at 900 MFLOP, while the one developed and tested using a heterogeneous architecture performs at least 1510 MFLOP at a comparable power consumption. The significant speed improvement using the co-processors on a heterogeneous system is important. It lead to the investigation and implementation of better algorithms to obtain better results. For example in [5] the use of the GPU for accelerated image analysis using OpenCL, enables the hardware to reach up to 1000 PAL resolution frames per seconds for certain functions, which certainly could not be accomplished using simple homogeneous system with general purpose processing units. It reveals the need of the space exploration development to move to development procedures which allow to take advantage of new architectures with better capacities. In addition, because of the complexity to develop for more robust architecture, it is necessary to have a way to check for the develop code and its integration with the systems, a task which could be automate by the use of continuous integration systems.

2.4 Different approach on software development for small satellites

In the review of different methodologies of software design for small satellites, two software development approaches are clearly defined. The most common and used method until now is the monolithic design. With this approach, the software is develop as one entity, having clear interaction and dependency between the different algorithms implemented. The second development approach is the design by component. This approach define different components of software into a basic modules, where each algorithm can be develop in separate. Each development process is reviewed in order to understand its benefits.

2.4.1 Software development: Monolithic approach

In the development of software for small satellites, the monolithic was in the beginning, the simplest and faster development process. The main idea is to incorporate the applications of the satellite with the main core processes, including the operative system if any. On this kind of approach, the result software for the small satellite can not exist if one of they component is missing or is not executing correctly.

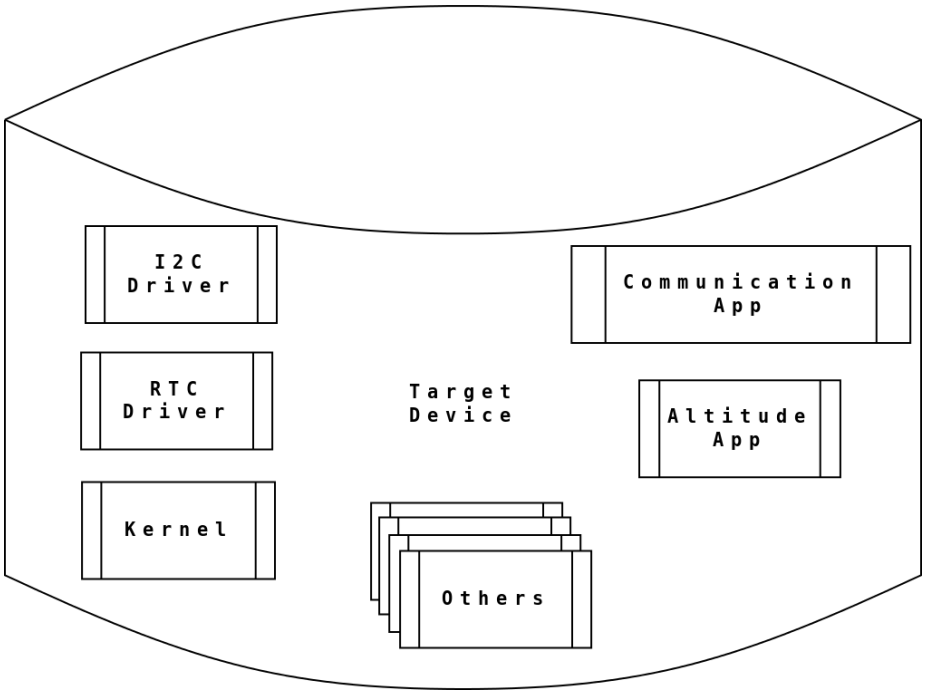

Figure 2.2: Monolithic development[2]

The monolithic software development, as shown in figure 2.2 has its advantages. Each software part of the system is designed and optimize to work taking the best capacity possible from the available resources. The designed and implemented code is highly optimize for the developed platform and strongly coupled with the other algorithms for the satellites. The code develop with this approach lacks portability. In case another space exploration is required using another platform for the satellite, it is required to redesign and re-implement the software solution. This approach use to be the best for small systems which only perform a little number of task, as each task must be highly couple with the others. Nowadays small satellites complexity grows in hardware, and software, which leads in problems implementing this development approach.

2.4.2 Software development: Development by component

The space exploration gained strength in the last years, specially with the proliferation of small satellites projects, because of this, the need for better designs arises. Different developers groups are investigating new methodologies and design paradigms. Something they all have in common is the need of modular reusable designs in order to reduce cost and development time. The main idea

[2]Original image. Exemplify the concept from [13] on monolithic development

is to separate the problem in different modules and use different pieces of software and algorithms in order to solve specific problems. This idea is widely spread in the satellite developers community. One solution is the use of off-of-the-shell components[9]. Those components are usually bought from third parties companies or created by specialized research teams inside the project. The components are later integrated in the project.

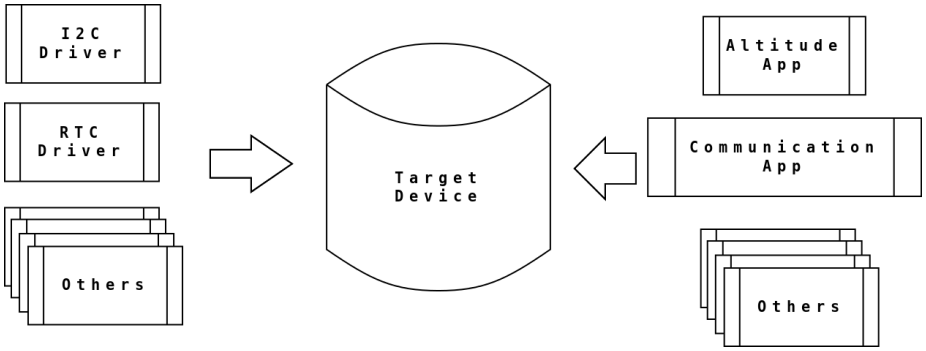

Figure 2.3: Development by component[3]

The use of components allow developers to treat the problems for a mission in a modular way, as shown in figure 2.3. It leaves the big work load of the algorithm implementation to a specialized team. The developer team in charge of each module use to have deeper understanding on the requirements of the component and how to implement it in a proper way. Development by component approach is rapidly gaining strength in the satellite development community, but also arise the need of high quality pieces of software, with high compatibility, optimize for its sustainability and performance. Development using this approach need for a integration phase, where compatibility between the different components are tested and modified in order to construct an image containing all the software packages.

Yocto as a solution follow this approach. It is a build system that take care system integration. It allows the developer to focus in the component development and rapidly test for the integration of a module with the complete system. Yocto can include third parties designers only capable of provide the source code or the binary representation of the component because Yocto build system is capable of produce an SDK. This SDK has the basic required packages and development tools to compile an application. By this way, the developers compile their software modules with the same toolchain and tools used to generate the main OS. It reduces compatibility on the integration phase.

[3]Original image. Exemplify the concept from [4] on design by component

2.5 Open Source tools on the design and implementation of software satellite

The use of Open Source tools on space exploration projects are usually rejected by the instability of packages between versions and releases. It is common to think that the required stability will only be found from commercial software, however, this is not true. There are several Open Source software with very stable lines of development. This project have big developer communities behind, which give support and provide a good quality to the product. One good example is the Yocto Project, a build system which come to bring stable releases and serves to build stable constructions systems [24].

From the presented challenges until now in the development for small satellites, the opportunity to create a better development process is available. In this research, different stable Open Source projects are proposed to address a solution for these problems. To ease the configuration and construction, a construction project is created. This construction system envelop several abstraction layers separating platform dependent and independent code. It also abstract the configuration and construction of the underlying build system, presenting only the output images to the user. The use of a Open Source build system to take care of the construction and integration of code is used by the construction systems. It takes the different layers of abstraction and integrate into one build. In addition a Open Source continuous integration system is included in the analysis of the project. Its function is to capture important data about the development process, capture possible errors introduced and report the stability of the different layers of code.

Chapter 3

Integration of build system for small satellite missions

A build system is a suite of programming tools arrange to perform a construction task without the intervention of the user. This process can be as simple as generating the structure for a project, to the hard task of creating a complex operative systems and specific pieces of software. The selection of the correct build system to use depends on the requirements of your project and how can be adjusted to it. In addition, how well supported the project is and how the developer can adapt to work under it must be take into consideration. In this chapter we focus on the study of different build systems available on the Open Source community, and determine which one is the best fit for the proposed solution on software development for small satellites.

Several types of build systems are available today, each one trying to solve a specific necessity. The problem they are designed to solve determine the way it works and how it is implemented. The study build systems are Buildroot, PtxDist, LTIB, OpenWRT, OpenEmbedded,, Geento. There are many more build systems available in the Open Source community, however, the selected are the more robust and with the biggest support by active communities. This avoid to use a solution that will be deprecated in a few mounts.

3.1 Build systems as an improvement on the design methodology

The design of a embedded system is not an easy task specially for small satellites. There are several groups of design and development for software and hardware. They are working together on the creation of a functional system which solve in the best way the specified requirements. Several design approaches can be choose in the development of a small satellite. It may go from the aspect of using available hardware or one that can be created, to which software is available

or can be implemented, basically hardware versus software approach. How to connect both aspect in a final product will define how effective the development process is. As more complex design and implementation is required, the final cost and development time for the release of a system is increase. This time can be the inflection point on the decision to launch or not a project. Because of this is important to find an approach that could coexist and improve the hardware and software development.

This research focus on the software for the small satellite, however the possible development solutions proposed doesn't have to ignore the fact that the platform running the software may not be already available. This is the reality several projects today, where the hardware of the system is under design, development, or studies to be acquired when the software begging its design and development. Because of this, a flexible approach of development should be selected. It should be able to show results on early stages of development and can be quickly ported to new platforms. On the cases where changes on the platform are perform, build systems are a great decision for a project, but it must be in companion of a good development methodology. For example, if the Y chart methodology is implemented, the software development can advance on the system level and detect malfunctions or lack of accomplishment on the requirements. The use of a build system will bring the flexibility to rearrange the software to still be functional on the next iteration of development where changes on the platform can be implemented.

As stated in [20], the Y chart it is a methodology to provide designers with quantitative data by analyzing the performance of architectures by a given set of different application. It gives the designers a concrete idea of how the final product behave, but the important part of the methodology is the flexibility on going back to stages where deep infrastructural changes can be made. As stated in [10], the methodology makes the assumption that each design can be modeled in three basic ways, which emphasize different properties, those properties are design behavior (function, specification), design structure (netlist , block diagram), and physical design (layouts, boards, packages).

This kind of methodology approach allow to develop several phases in a parallel way. More important, it allows to obtain early results on the performance of a system. By these results, the designers are able to detect problems, or change the approach of a problem solution in a early stage of the project, reducing the risk. With this approach there's room for the application software developers, to adapt in the best possible way its development, base on the performance results and the modifications performed by hardware. This is only possible with the use a good build and construction methodology. With a modular development in stages like the one shown in figure 3.1 is where a flexible build system increase the productivity of the software developers, where with small changes on the only required pieces of software, the final software can be available to be run on on the available platform on early stages of development. This kind of development will return early performance results to designers.

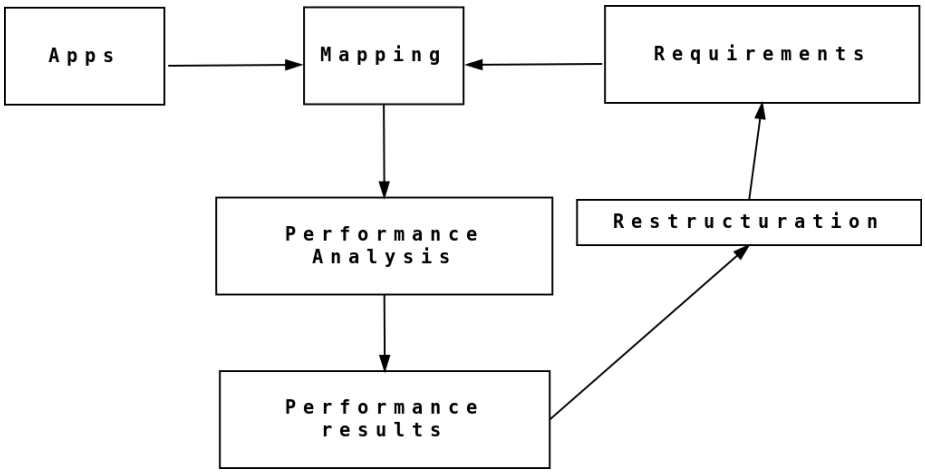

Figure 3.1: The Y chart approach with the different development stages[1]

The use of a build system can improve the development on a project, however, What is the best build system to use?. The response to this question depends on the specific requirements of the project, because a solution which work for the development of automotive projects will not be the suited on the design of software for space exploration. There are a number of build systems available which solve different necessities and they are better suited for different applications. Next we cite the different build system that were taken into consideration.

- Buildroot: It is a very easy to use build system to generate Linux systems, base on a set of Makefiles and patches, which are in charge of the generation. It can generate the toolchain, a root filesystem, kernel and bootloader images. It is very popular and mainly used between people working with small embedded systems which required customization [21]. Buildroot is better suited for kernel developer more than application developers, although they are not excluded as Buildroot has a simple structure to understand.

- PtxDist: It is a build system designed for the creation of your own Linux distribution, as they state, it is not a distribution, but what they call "a executable documentation", comprising all the steps necessary to create a customize complex distribution for the target system, been in majority easy to use for a inexperience user.[36]. PtxDist is a easy build system, however several steps on its configuration are better if they are performed by experience users. Once configured, the creation of build systems are straight forward. This build system has really good documentation about Linux porting, drivers incorporation and bootloader porting.

[1]Original image. Exemplify the concept from [20] on the early results on development

- LTIB: It stand for Linux Target Image Builder and is a tool that can be used to develop and deploy board support packages (BSP). It supports different architectures like ARM and PowerPC [16]. LTIB has a very granular configuration and incorporate over two hundred packages, however for the non experience user its configuration can be tricky and exhausting.

- OpenWRT: It is know as a Linux base distribution, which instead of providing a static single firmware, provides a fully writable filesystem with package management, freeing the user from the target application configuration [25]. Its easy to use but it stay as only a Linux base distribution.

- OpenEmbedded: It is a build framework for embedded Linux. It offers a best-in-class cross-compile environment. It allows developers to create a complete Linux Distribution for embedded systems supporting many architectures, multiple releases peer architecture, easy configuration, tools to increase the speed of construction and with a lot of supported packages which can be integrated in the final image [28]. It is the base for other build system like Yocto, on which this document discusses in further sections, however, because is so big, its structure is a little messy and developers can be easy mislead on the quantity of files and configuration knobs.

- Yocto: The Yocto Project is an open-source collaboration project focused on embedded Linux developers. Among other things, the Yocto Project uses a build system based on the OpenEmbedded (OE) project, which uses the BitBake tool, to construct complete Linux based images. The BitBake and OE components are combined together to form Poky, a reference build system [32].You can use components from the Yocto Project to design, develop, build, debug, simulate, and test the different packages for different platforms.

- Geento: It is a free operating system based on either Linux or FreeBSD that can be automatically optimized and customized for just about any application or need. Extreme configurability, performance and a top-notch user and developer community are all hallmarks of the Gentoo experience [12]. However as OpenWRT it is only a System distribution.

The construction system proposed on this research will use a build system as the central point of configuration and construction. This build system must accomplish several requirements beside fit the solution proposed. It also must support modularity of the solution, been stable and well documented and have a good community support between other characteristics. On the next table is shown a review of the different features of the mention build systems. The build systems focus as only base distribution are not taken into account as its design as only distribution systems do not fit into the proposed modular solution.

[2]Comparison between different build systems. Information taken from [12], [32], [28], [25], [16] and [36]

Table 3.1: Comparison between different build systems[2]

Feature	**Yocto**	**BuildRoot**	**LTIB**	**PtxDist**
Runtime package management	Yes	No	No	No
Requirement for full builds	No	Yes	No	Yes
Support for multiple HW	Yes	Yes	Yes	Yes
Number of sponsors	47	11	1	11
Available packages	more than 7500	more than 600	Data not available	Data not available
Base license of the project	Mix of MIT and GPLv2	GPLv2	GPLv2	GPL
Base language	Phyton	Makefiles syntax	Kconfig and Perl files	Makefiles syntax
Allow new packages	Yes	Yes	Yes	Yes

Each build system counts with their special characteristics which make them unique. Is up to the developer to choose a build system which fit the best for its application. In the case of software development for space exploration, and specially for the proposed solution, the modular capability is very important. Having runtime package management allow to apply modification to the configuration and construction of a package in a dynamic and modular way, having the build system resolve the order of modification by priorities set beforehand. It is very important the support the build system has from the community, because it assure a longest live, more documentation available and more developers capable of bringing help around the world. In addition, the development for small satellites has not grow enough on the software side, relaying on tools or third parties for the software development. Because of this seeking for a easy to learn language for the development of the build system is necessary. Yocto base its recipes of Python language, it is not the best language around, but it is very easy to learn and understand in a high level.

A very important steps of the research is the selection of a build system as center piece of the construction system. Yocto build system fulfill several of the requirements mentioned before on the construction of the small satellite software. The reason of the selection of Yocto as build system are listed as follow.

- Allow OS customization.
- Allow package customization and version selection.
- Minimal or none impact on packages customization.
- Reduce developers effort on software modification.
- Good support for the Open Source community.
- Count with stable and proven releases.
- Flexibility on platform changes, intending to reduce packages changes.

3.1.1 Yocto build system

The Yocto Project is an open-source collaboration project focused on embedded Linux developers. Among other things, the Yocto Project uses a build system based on the OpenEmbedded (OE) project, which uses the BitBake tool, to construct complete Linux based images. The BitBake and OE components are combined together to form Poky, a reference build system [32]. You can use components from the Yocto Project to design, develop, build, debug, simulate, and test the different packages for different platforms. Yocto is not just aim to develop an OS distribution, or a piece of software, Yocto is more complete than that. It allows the developer, by the use of different recipes, to describe process of construction for any kind of project. The different projects to construct with Yocto goes from the construction of a toolchain or the construction of the OS for a specific board, to the compiling process of a simple hello world application. On figure 3.2 is illustrated the development environment using Yocto.

For the development process presented on this research, the Yocto project is used as the core build system of a construction system capable to construct the operative system of the small satellite. It is also capable of construct and deploy an SDK with the available packages and the tools by which the software of the small satellite is generated. The packages of software that will be run on this OS can be also be constructed by itself, giving the choice to construct only the required package. The selection of the final output is selected by the user of the build systems. This process is exemplify in the figure 3.3.

Using a build system like Yocto for the design and implementation of small satellites will be an improvement on the development methodology used until now. It increase the availability of resources and its maintenance, taking advantage of a big community behind it. The use of the build system will impact on the development time, the cost, the scalability and the stability of the project. The modularity given by this tools will also help to improve the development process, allowing more specialized groups to include and work on different pieces of software without disrupting the integrity of the whole project.

[3] Original image. Illustration of the Yocto environment from [32]
[4] Original image. Exemplify the building output selection taken from [4]

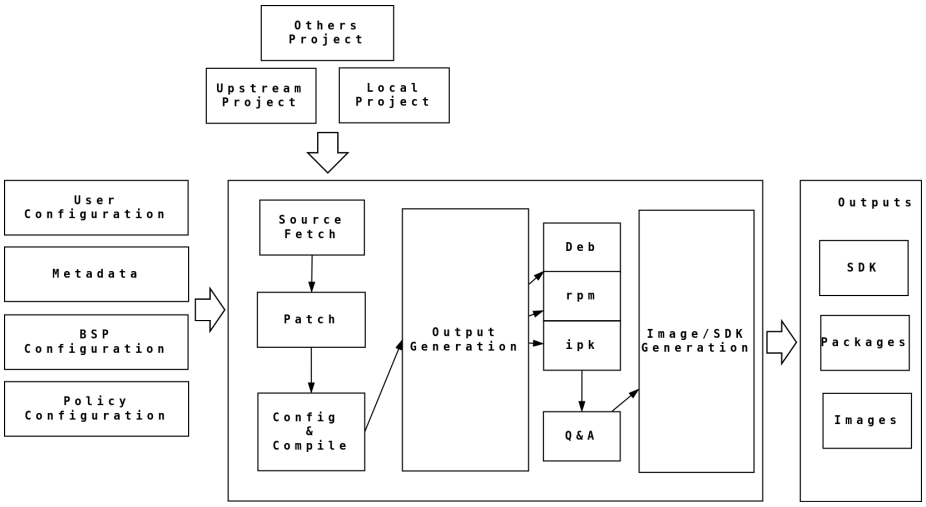

Figure 3.2: Yocto Project Development Environment[3]

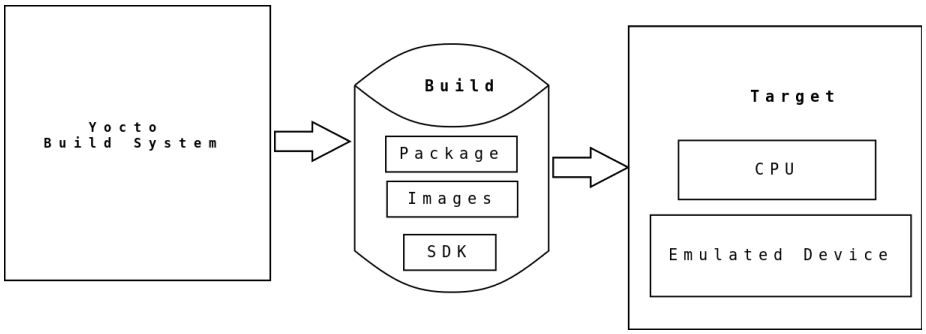

Figure 3.3: Building an Image with Yocto[4]

Yocto Layers

The Yocto project allows the creation of different layers to organize the different Metadata and recipes that will be use in a project. This layers allow to encapsulate all the necessary packages and configuration necessary. The main purpose of layers is to isolate different types of customization from each others[32]. This layer methodology allow developers to keep things in a modular way, encapsulating images with certain packages, and deploying the correct output(an image or a package) for the selected target.

Inside each the layers, the developers can keep all packages separated and well organized by using recipes. The recipes may be divided by the kind of support they give. For example you could divide the BSP configuration on one recipe, while keeping the GUI packages in other, or the networking packages in another, and so on. On the current proposed solution, the layer subsystems is an

important exploited advantage. Hardware dependent and independent pieces of code are separated and grouped in different layers. In addition, different target dependent layers are develop in parallel, which allow developers to switch from platform to platform in a easy and quick way.

Chapter 4

Development platforms

The flexibility in the support for different platforms is one of the advantages of the current implementation. It allow the support for different development boards with different capacities and different cores. It even allow to include new targets in the future. For example, platforms with single or multiple processing units in symmetric or asymmetric architecture can be added to the construction system. Supporting different targets may help in the fast development for different missions with different requirements. In addition, having support for different target can be used to compare performance and determine which one is better suited for an specific mission. The small satellite development lower the cost of the its target vehicles, however the are not available to all developers groups, having prices around $4000 USD to even $15000 USD. Because of the high prices, researching about new available vehicles which meet the desire requirements for a space mission becomes really important. Even the research on target vehicles with similar characteristic on the final available target can enable developers to start the software development. The target development boards for this project are developed by Texas Instruments, they are Open Hardware projects which allow the user to have knowledge and some specific insights on the used architecture and BSP. Some of these vehicles were used for space projects or are in the scope for possible future missions.

4.1 Beagleboard XM

The Beagleboard XM is a low cost ARM Cortex A8 board which has compatibility with several Cortex A8 processors manufactured by Texas Instrument. Which processor is used depends on the revision and the capabilities desired for the board. There are version of the board with a DM3730 and AM3715 processors. The main difference is the DSP, which is not included in the AM3715 version. The version used on this project is the one with the DM3730 processor.

The Beagleboard XM was born from the original Beagleboard project, which was a development board with a Omap3530 SoC. The newest version, the Bea-

gleboard XM, is a more powerful board, with higher operation frequency of its processor and its DSP, with new features like a uSD, camera header and over-voltage protection.

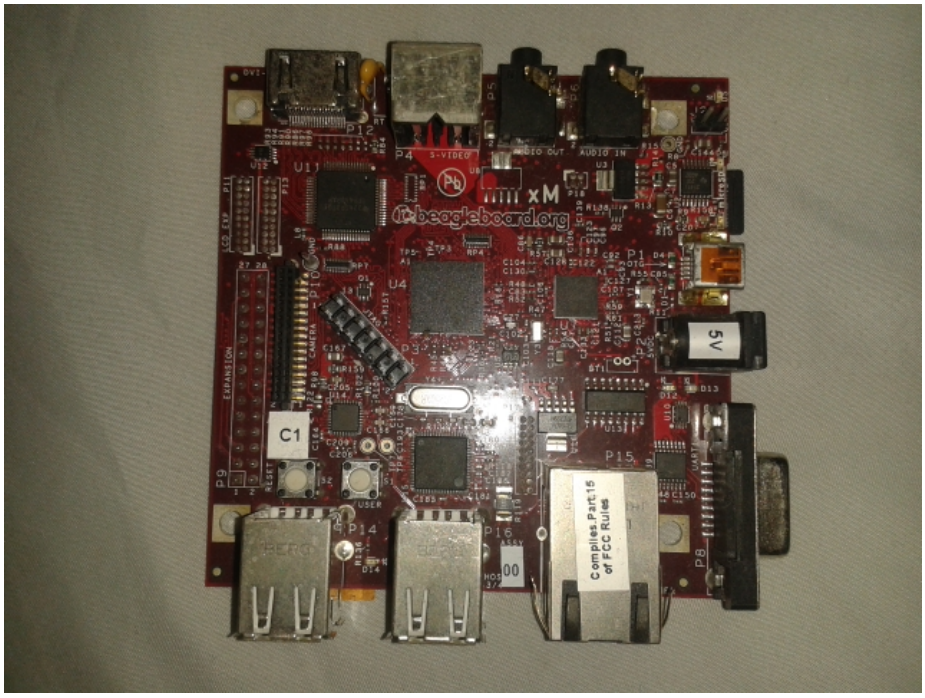

Figure 4.1: Beagleboard XM[1]

The Beagleboard XM shown in figure 4.1 is a development board aimed to the Open Source community, equipped with the many features to experience the processor capabilities but not intended as a full development platform, instead is a first step for a more complete design. This board used standard interfaces making it highly extensible to add more features. All of the design information is freely available and can be used as the basis for a product [35].

Many of the Beagleboard XM features are shown in the next table.

The Beagleboard XM used on this project is the one with the DM3730CBP 1GHz processor which comes in PoP package, a technique where the processor is mounted on top of the processor. This is a important piece of information, although the benefits of a PoP package at electrical level, also because when you look into the board, you will not see the number part of the processor, instead you will see the part number of the memory.

The Beagleboard XM has been use in space exploration missions before. One example is the USS Langley mission, on which the Beagleboard was include on

[1]Original image. Picture of Beagleboard used for development
[2]Beagleboard XM features taken from [35]

Table 4.1: Beagleboard XM features[2]

Component	**Feature**
Processor	Texas Instruments Cortex A8 1GHz processor
PoP Memory	Micron 4Gb MDDR SDRAM (512MB) 200MHz
PMIC TPS65950	Power Regulators, Audio CODEC, Reset, USB OTG PHY
Debug Support	14-pin JTAG, GPIO Pins, UART, 3 LEDs
PCB	3.1 x 3.0 (78.74 x 76.2mm), 6 layers
Indicators	Power, Power Error, 2-User Controllable, PMU, USB Power
HS USB 2.0 OTG Port	Mini AB USB connector, TPS65950 I/F
HS USB Host Port	SMSC LAN9514 Ethernet HUB, Up to 500mA per Port if adequate power is supplied, 4 FS/LS/HS
Ethernet	10/100 from USB hob
Audio Connectors	3.5mm L+R out, 3.5mm Stereo In
SD/MMC Connectors	MicroSD
User Interface	1-User defined button, Reset Button
Video	DVI-D, S-Video, Supports Leopard Imaging Module
Power Connector	USB Power, DC Power
Overvoltage Protection	Shutdown at Over voltage
2 LCD Connectors	Access to all of the LCD control signals plus I2C, 3.3V, 5V, 1.8V
Auxiliary Audio	4 pin connector, McBSP2
Auxiliary Expansion	MMC3, GPIO,ADC,HDQ

the research for an internet server to access the satellite. The research study the radiation levels on which the vehicle where still functional. The Beagleboard was fully functional to a total dose of 35 Krad(Si). After a dose of 40 Krad(Si)/s, the functions in the board start to present errors [1]. However, the Beagleboard SoC has bee detected to not be too stable working on batteries as power source. The instability of a power source can be troublesome for the OMAP family SoC's. Regard this limitation, the use of the Beagleboard caught the attention in this research as a target vehicle because it allow developers the early development of software for a space exploration.

4.2 Pandaboard

The Pandaboard is a development board designed by Texas Instrument. The SoC of the Pandaboard may vary between the Omap4430 or the Omap4460 (it depends on the board revision), been the latest the one known as the revision ES of the Pandaboard. It has a dual core ARM cortex A9, giving a Symmetric Multiprocessing (SMP) capability. It counts with a graphics core, a POWERVR SGX540, capable of supporting actual graphics API's as OpenGL or OpenVG. The SoC of the Pandaboard contain a DSP subsystem, base on TMS320DMC64x very long instruction word DSP Core. Also a IVA-HD subsystem image and video acceleration. The Pandaboard also counts with a ARM Cortex M3 processors. The M3 may be programmed for specialized purposes and communication between the different subsystems. The Pandaboard have peripherals to interact with the processing system as USB ports to connect keyboards, mouse, camera, and other USB devices, HDMI ports, a 3.5 jack for input and output audio allowing stereo input support, SD connector card, 10/100Mb capable Ethernet connection, wireless connection by WiLink, supporting 802.11 b/g/n and bluetooth, and many other peripherals[30]. The Pandaboard ES used on this research is shown in figure 4.2.

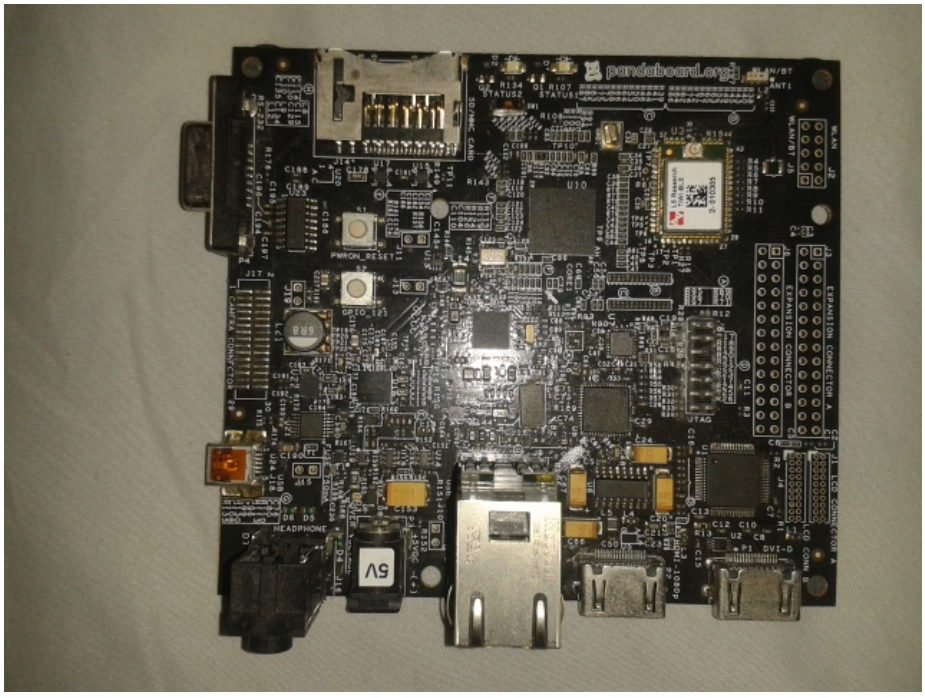

Figure 4.2: Pandaboard development board[3]

[3]Original image. Picture of Pandaboard used for development

The available Pandaboard in which the generated OS is tested is the ES revision. This revision contains an Omap4460 SoC, however the build system is intended to support the Omap4430 version. For the rest of this document we refer to the Pandaboard ES. The next table, taken from the system reference manual of the board [29], list the different features of the board.

Table 4.2: Pandaboard ES features[4]

Component	**Feature**
Processor	Omap4460
PoP Memory	Elpida 8Gb LPDDR2 (EDB8064B1PB-8D-F)
PMIC	TI (TWL6030 Power Management Companion IC)
Debug Support	14-pin JTAG, GPIO Pins, UART via DB-9 connector, LEDs
PCB	4.5 x 4.0 (114.3 x 101.6 mm), 8 layers
Indicators	3 LEDs (two user-controlled, one overvoltage indicator)
HS USB 2.0 OTG Port	Mini-AB USB connector, sourced from OMAP USB Transceiver
HS USB Host Port	Four USB HS Ports, up to 500mA current out on each, two onboard connectors, two expansion connectors
Audio Connectors	3.5mm, L+R out, 3.5mm, Stereo In
SD/MMC Connectors	6 in 1 SD/MMC/SDIO, 4/8 bit support, Dual voltage
User Interface	1-User defined button, Reset Button, SYSBOOT3 switch
Video	DVI-D or HDMI, Optional user provided plug-in display
Power Connector	USB Power, DC Power

The Pandaboard ES, using an OMAP SoC present the same problems as the Beagleboard regarding instability when powered by batteries or alternate power supplies which may present variations. However, the Pandaboard present a powerful core processor, the Arm Cortex A9, very common in the development of advance embedded systems and with great capabilities for the development of space exploration missions. It also present different subsystems, which extend the possibilities for developers to develop and test software which communicate between each other. On figure 4.3 is shown the architectural block diagram of the Pandaboard ES and its subsystems.

[4]Pandaboard features taken from [29]

[5]Image which exemplify the different subsystems in the Pandaboard [29]

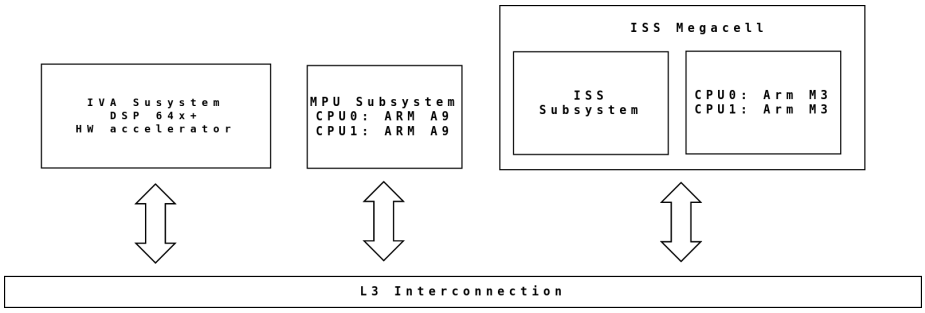

Figure 4.3: Pandaboard general block diagram[5]

As shown in figure 4.3, the OMAP4460 is composed of the following subsystems.

1. Cortex-A9 microprocessor unit (MPU) subsystem, including two ARM Cortex-A9 cores capable of operation at 1.2GHz.
2. Digital signal processor (DSP) subsystem.
3. Image and video accelerator high-definition (IVA-HD) subsystem.
4. Cortex-M3 MPU subsystem, including two ARM Cortex-M3 microprocessors.
5. Display subsystem.
6. Audio back-end (ABE) subsystem.
7. Imaging subsystem (ISS), consisting of image signal processor (ISP) and still image co-processor (SIMCOP) block.
8. 2D/3D graphic accelerator (SGX) subsystem.
9. Emulation (EMU) subsystem.

The Pandaboard has an Omap4460 processor, the heart of the development board, which implements a 38.4 MHz 1.8V CMOS square-wave oscillator share with the TWL6040 audio companion. This clock is also used as a PLL input within the OMAP processor so that it can generate all the internal clock frequencies required for the system. The processor is based on the OMAP architecture of Texas Instrument, which uses 45nm technology. Because of the high integration of the technology used for the OMAP architecture of the Pandaboard, it very susceptible to radiation. It is recommendable to perform further analysis in order to validate the effects of radiation in the board. This analysis is beyond the scope of this research. The Omap4460 support high level operating systems such as Windows CE, WinMobile, Symbian, Linux, Palm OS and Android[29].

The Pandaboard ES is a great development board with a lot of capabilities. It counts with the support of Open Source community which move with the latest news available, making it easy to use, specially for the central processing. The available diversity in different subsystems make available to developers to design, develop and test pieces of software which can be ported later if necessary.

4.3 Beaglebone

The Beaglebone development board is a platform developed by Texas Instrument, with the main purpose to create a easy to use target to experimented developers and the new developers joining to the embedded world. There are two flavors of the Beaglebone. The white version, a development board around the $89 USD, with a DSP, integrated DFI chip and on chip JTAG for debugging purposes. The black version is the latest edition, designed to address the Open Source Community, early adopters, and anyone interested in a low cost ARM Cortex-A8 based processors because its price its around the $45 USD [2]. The construction system has support for both version of the Beaglebone, as its main differences are addressed by they device tree files, however the related testing shown in this research its developed on the Beaglebone Black (BBB).

The BBB has been equipped with minimum set of features to experiment with, however its not designed to be a full development platform, neither to be the main computer for a satellite, as some features of the processor are not accessible from the BBB interfaces. Both version allow the use of add-ons boards called capes, to add many different combinations or features to the ones already provided [7]. On figure 4.4 is shown the Beaglebone Black used for the research.

Both version of the Beaglebone are design to be compatible between each other as much as possible, but there are several areas of differences between both of them. The more important difference is in the processor, which continue being part of the AM3xx family, but present an increase in its speed (1GHz). In addition, there's no JTAG emulation over USB on the BBB as no serial port by default is present. The BBB add an on-board managed NAND (eMMC) of 2GB and 512MB DDR3L, which is a increase of size and a performance boost with cost reduction. Next are shown some of the BBB features [7].

The Beaglebone Black has been widely accepted on the Open Source Community, opening the embedded world to a lot of new developers. On this project, the BBB support is added to show the scalability and the capability of multiple platform support of the build system. Its beyond of this research a radiation analysis and other requirements to validate the board as a satellite vehicle. Instead, we focus in its capability of supporting the software of the satellite and the advantages it may add to the project. In addition, its support allows the developers to use a simple board, easy to acquire, with good support and easy to program, increasing its experience with embedded systems. Furthermore, the

[6] Original image. Picture of Beaglebone Black used for development

[7] Beaglebone Black features taken from [7]

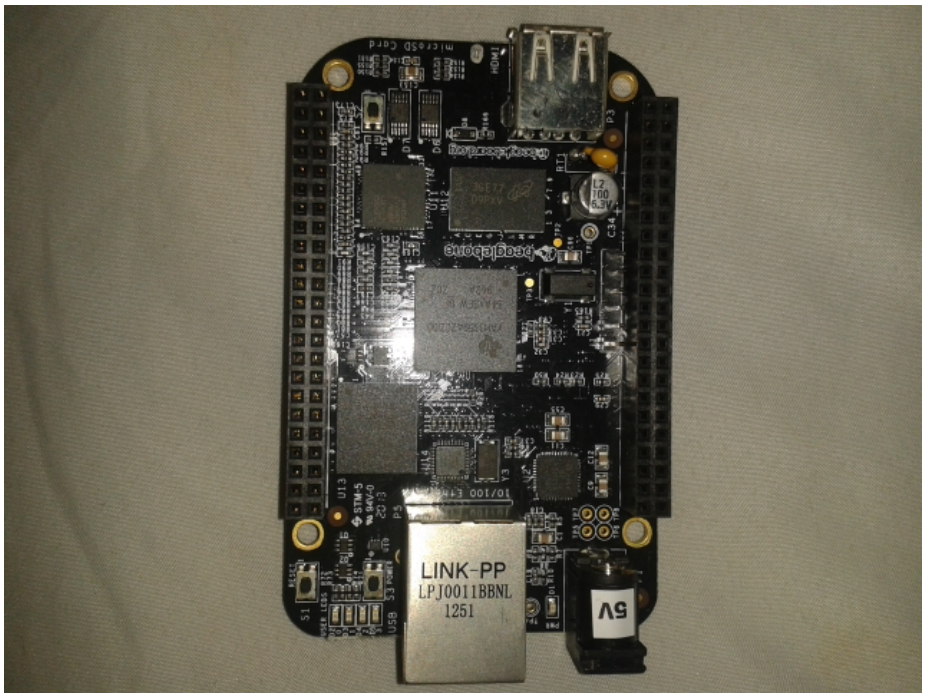

Figure 4.4: Beaglebone Black[6]

capabilities shown on the SoC of the BBB can be compared with more complex board which may be complaint with the requirements of space exploration missions.

Table 4.3: Beaglebone Black features[7]

Component	**Feature**
Processor	Sitara AM3359AZCZ100 1GHz, 2000 MIPS
Graphics Engine	SGX530 3D, 20M Polygons/S
SDRAM Memory	512MB DDR3L 800MHZ
Onboard Flash	2GB, 8bit Embedded MMC
PMIC	TPS65217C PMIC regulator and one additional LDO
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header
Power Source	miniUSB USB or DC Jack, 5VDC External Via Expansion Header
PCB	3.4 x 2.1, 6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB
HS USB 2.0 Host Port	Access to USB1, Type A Socket, 500mA LS/FS/HS
Serial Port	UART0 access via 6 pin 3.3V TTL Header. Header is populated
Ethernet	10/100, RJ45
SD/MMC Connector	microSD , 3.3V
User Input	Reset Button, Boot Button, Power Button
Video Out	16b HDMI, 1280x1024 (MAX), 1024x768,1280x720,1440x900 ,1920x1080@24Hz, w/EDID Support
Audio	Via HDMI Interface, Stereo
Weight	1.4 oz (39.68 grams)

Chapter 5

Design and implementation of the construction system

On the present chapter, the design and development of the construction system is reviewed. The construction system is constructed in a modular way to fulfill the requirements of scalability and reusable pieces of software. The construction system follow the design pattern described in figure 1.2. It is intended to take pieces of software for different sources and integrate it into one construction system in order to generate a functional image.

The construction systems uses Yocto project as its internal build system, abstracting the set up process from the user. It encapsulate the basic packages needed and the for a target board, which allow the easy selection and configuration of a custom Linux OS distribution, ready for the deploy and testing on the selected target.

5.1 Construction System

The construction system is the principal abstraction layer of the proposed solution. It is a simple but powerful encapsulation made around the principal components of the proposed solution. The basic idea of this encapsulation is to be able to overcome the difficulties of change or modification of internal components, including the build system. This abstraction layer allow developers to focus on specific tasks without worrying about integration, leaving this task to developers specialized in the construction systems itself.

The construction system structures is shown on figure **??**. The construction directory structure is designed to be as simple as possible in order to avoid proliferation of files, on which developers may get lost. The Makefile.inc file contains definitions of variables necessary for the construction of the project. It includes the URL's of the different repositories, the configuration files needed and the required structure of the project once is configured. The different configuration files are located at tools/config_files. Each pair of bblayers.conf

and local.conf files exist by available hardware target and are in charge of the full configuration of the system. On tools/scripts there is the principal script of the construction system, build_script.sh. This script source the environment of the core build system, in this case Yocto build system. It also take care of triggering the construction of the system.

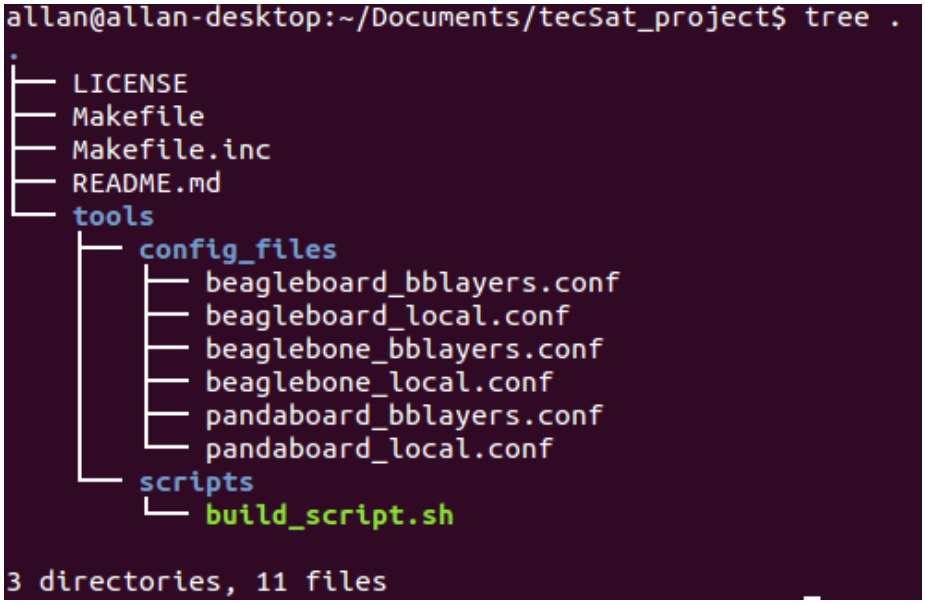

Figure 5.1: Directories of the construction system[1]

All the process is controlled by the Makefile. It is the one in charge of provide the different targets available to the user as config, build, clean, distclean and help. It is the one in charge to interact with the user of the build system. It is designed to be as simple as possible to the user, which is in charge of provide a few simple targets in order to construct an image for a target. In order for the construction of an image the only necessary targets are config and build targets. The config target will fetch all the layers necessary, dependent and independent from the hardware. The build target will trigger the build system for the construction of the image.

5.1.1 The hardware independent layer: meta-tecSat

The meta-tecSat layer is a custom Yocto layer created to keep all the hardware independent configurations and packages for the system. It allows the user add this layer and it dynamically select the wanted packages to create a simple and functional image, including the configuration for the custom distribution. The created images are base on Linux. Currently the layer support three different

[1]Original image. Directories of the construction system

image types, tecSat-image-minimal, tecSat-image and tecSat-image-dbg. It also declare an environment standard variable indicating the inclusion of packages from a hardware dependent layer. The tecSat-image-minimal is a minimal customize image base on Linux, which allow a functional system to boot up with the minimal support of a Linux system. The tecSat-image is base on the minimal image, but it add the more packages defined by the user which will represent the specialized software for the satellite. When developers add independent pieces of code for specialized functions in space exploration, they will be added to this image. The third image available is a debug image, base on the tecSat-image. It adds debug packages with capabilities to support eclipse debug and packages to review performance. For example, this images add packages capable to do tracing at user or kernel level.

Inclusion of the meta-tecSat layer allow the customization of the distribution. By specifying in the local.conf file the distribution type wanted, it may be a tecSat distribution or a normal poky distribution. The tecSat distribution contains special configuration which avoid not needed packages in order to reduce footprint of the whole image. By itself, the meta-tecSat layer is not capable of the entire construction of the system. It is require to associate this layer with a platform dependent layer in order to assure the correct construction of the image. Otherwise the target platform for which the construction is made may not be available or may contain undesired features.

The different layer implemented has its own priority. The meta-tecSat layer has lower priority than the hardware dependent layers. The reason for this design constrain is to allow platform specific modification, which should be append to the original recipes in the meta-tecSat layer by the recipe appends methodology. Those appends could include patches and new pieces of software in order to increase the performance of the package but only if necessary.

5.1.2 The hardware dependent later: meta-tecSat-target

The hardware dependent layers are the custom Yocto layers created in order to keep all configuration and special packages which depends strongly of the target platform they are executed. Example of this packages are the kernel and the bootloader. The hardware dependent layers are interchangeable between each others, been only one needed and used at the time. Which layer is used depends on the selected target and specified in the local.conf file of the Yocto build system. If other layers besides the one referring the target exist, they are simply ignore.

The convention for the name of the layer follow the pattern meta-tecSat-target, where target must be replaced with the name of our target platform. For example, if our target platform will be a Beagleboard, the name of our hardware dependent layer will be meta-tecSat-beagleboard. This approach allow developers to create new layers in the future for new platforms.

All the hardware dependent layers have the same priority and is up to the user to specified which one to use by configuration. The hardware dependent layers have higher priority than the meta-tecSat layer. This priority allow devel-

oper to include support for new targets in the future in an easy way. In addition, the higher priority in these layers allow to include modifications and patches to the code which are intended to increase the performance and efficiency in the specific target.

On the present research several development platforms are supported. To each one of these a specific layer is created. The supported development platforms supported are the Beagleboard, the Beaglebone and the Pandaboard and the created layers for each target are meta-tecSat-beagleboard, meta-tecSat-beaglebone and meta-tecSat-pandaboard respectively. On the section development platforms the capabilities of each target are reviewed.

5.1.3 Integration of the dependent and independent hardware layers in the construction system

The different layers created separate in a modular way the pieces of code which depend and do not depend in the hardware. Using any of those layers will allow the creation of a result image. However, the real powerful advantage of the construction systems comes with the integration of both layers in a common ecosystem. The layer independent layer will provide pieces of software specialize for space exploration missions and the target specific layer will provide the ones specialized in the hardware. This combination will allow the best and more flexible integration of the software. The ecosystems that will combine the layers and provide the configuration will be the tecSat-project. In the figure 5.2 show the relation of the different layers inside the build system (Yocto) wrapped by the construction system.

As seen in 5.2, the hardware independent layer will sit on top the hardware independent, meaning it has a higher priority and it is able to modify packages and configuration made by the hardware independent layer. On the bottom, the basic and always needed Yocto layers are represented as well as they can not be removed. These layers contains basic configuration of the build system itself and will be always present on our construction.

Once the construction system is in your system, by execution the configuration, all the needed layers and the build system itself is fetched. The construction systems perform the configuration within the build system to take this burden from the user, however the manual configuration is also possible. Next are shown the required steps to configure the project.

```
# Configuration for the platform
$ make config PLATFORM=<desire platform>
$ vim yocto/build/conf/bblayer.conf // Layer manual config if necessary
$ vim conf/local.conf               // Packages manual config if necessary
```

If you want to add a special available package to your recipe, edit the local.conf file and add the desire configuration. This may be accomplish by adding

[2]Original image. Representation of the layer on the build system

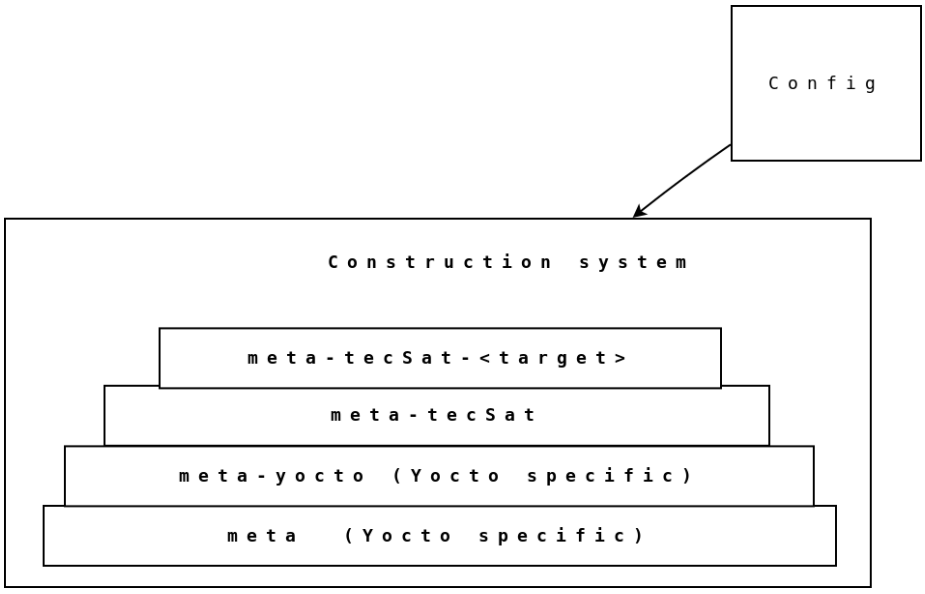

Figure 5.2: Layers inside the build system[2]

IMAGE_INSTALL_append = "package_name" on the file. After adding the required packages for your image, you may continue the build process. Is important to know that adding extra package is not necessary, it is only necessary for a special package out of the ones contained in the default configuration. When you have performed all those steps, the environment is ready to create the selected image. You only need to perform the construction by make.

```
# Execute construction for the platform
$ make
```

The build process fetch the require packages from the web, however some packages can not be downloaded directly because of access request or licensing. Those package need to be downloaded manually into the download directory. After download it, you need to add the ¡name_package¿.done file to the download directory.

At the end of the construction process, all the required images, bootloader, kernel, file system, will be at the yocto/buildDir/tmp/deploy/¡board¿/ directory. For simplicity, the construction system construct a symbolic link to this directory on the root of the project under the name of images. These images are the ones you will need to move to the SD or the internal memory of your system, it will depend on the boot up process of your system.

In addition, the construction system allow the construction of an SDK, necessary to construct your own applications. In order to create your SDK is only needed to call the correct target in the Makefile. The result of this process

will be a script containing the toolchain and scripts to configure the construction environment. The libraries and headers necessary to construct the image are included as well. To construct the SDK you need to perform the following command.

```
# Construct SDK for the platform
$ make sdk
```

5.1.4 Adding a new recipe to a layer

Adding a new recipe is a simple task following the Yocto guidelines of recipe creation. The first you need is to to evaluate in what layer your recipe belong to. In the second step, you need to evaluate the recipe directories available at the moment, and see if it fits into that recipe. If not, a new recipe directory within the corresponding layer need to be created. If the recipe directory is needed, add it and create a recipe. With this recipe BitBake can fetch, compile and deploy your package. If you need to understand the Yocto build system and the way on how to create a new recipe, you can review the Yocto tutorials [32].

Once you have created your image, you can instruct BitBake to compile your package, or you may add it to you local.conf. In that way the package will be included in the result image. If you are using the construction system you need to call the corresponding target in the Makefile to construct your package. The correct target is shown next.

```
# Construct a package for the platform
$ make RECIPE=<desire_recipe>
```

Developers creating new recipes within a layer, need to have in mind the current repository they are working on. Its necessary to remember that each layer correspond to a different repository and the construction system as well. Changes made into a specific repository need to be pushed in it in order to preserve the changes, otherwise in the process of distclean the project the developer may lose the changes.

5.2 Continuous integration system in the tecSat project

Continuous Integration (CI) systems are automated process by which several developers merge they working copies of work to a mainline several times a day and integrity check are made. On those merges several actions can take place, like review the integrity of the code, perform testing and even emailing the developers. Continuous Integration systems were first incorporated on the techniques of extreme programming as a way to increase productivity and also keep the quality of the generated code. At the beginning the CI system were

used to run test several time a day on the mainline branches of code, but later it evolve on build servers which not only run test but also build the code and review its integrity. This evolution make popular the use of continuous integration systems, not only inside the extreme programming paradigm. The main advantage of having a CI system its the proven increase of the quality of the software and reduction of the delivery time, advantages which impact in a very positive way the development of a project.

For the development of software for small satellites, a CI system capable of meet important requirements is needed. This CI system will also be use to collect important information about the construction system. Some of these requirements on the CI system are:

- Must be easy to use.
- Can construct the code to check consistency.
- Must be scalable to support future upgrades on the project.
- Must provide ways to inform about status of the project.
- Must have good documentation and references.
- Easy integration with other tools.
- Must be a stable system.

Base on the previous needs, and looking for the available continuous integration systems available, the selection of one may not be an easy task. Not only because there is a huge amount of CI systems available, but because a lot of them can fulfill our requirements. A decisive factor on our selection is the easy integration with other tools and its scalability. Favoring the fact that Jenkins CI is a Open Source tools, and its license doesn't require a fee from us, Jenkins CI is a well supported and stable project, which with is capability to include plugins its very scalable in a project that can grow and sum new requirements. Some of these plugins allow the recollection of data and statistics about our construction system.

5.2.1 Integration of Jenkins with tecSat Project

The continuous integration system used for the project is the Jenkins CI, an open source continuous integration system. It has a lot of open source plugins for customization and great flexibility to be set as the CI for a great variety of projects. In the case of the current research the CI is customize to contain a set of jobs, which represent the creation of images for the different platform targets available. Jenkins CI uses Cron jobs, a time base jobs scheduler. It is possible to schedule commands or even scripts to be executed in a selected order. In addition it allows taking results from each construction phase, which allow us to gather information about our construction system. Jenkins CI contain user security, creating different users with different permissions and accessibility, to

ensure the protection of the project for unknown intruders. For each platform target a construction job is created in order to collect statistics, those jobs are listed next.

1. **tecSat_image_beagleboard**: This jobs construct the tecSat-image for the Beagleboard platform. This include the Beagleboard and the Beagleboard XM.

2. **tecSat_image_beaglebone**: This jobs construct the tecSat-image for the Beaglebone platform. This include the Beaglebone White and the Beaglebone Black.

3. **tecSat_image_pandaboard**: This jobs construct the tecSat-image for the Pandaboard platform.

4. **tecSat_SDK**: This job construct the SDK for the selected platform and deliver the script to install this toolchain on the user machine.

The jobs created on Jenkins, specially the tecSat-image-platform jobs, test the integration of the project on the master development branch. It allow to detect possible problems on the different components involved. This feature contribute to the stability of the project, giving security on its deliverable, as it is constantly tested. In addition, the implementation of different jobs allow abstraction to the users of the construction system, as it can be configured to create output artifacts from each build of the jobs. These artifacts can be the output images created, used on the different platforms to boot (the bootloader, the device tree, the kernel image and the root file system). In the figure 5.3 the front panel in the CI page is shown.

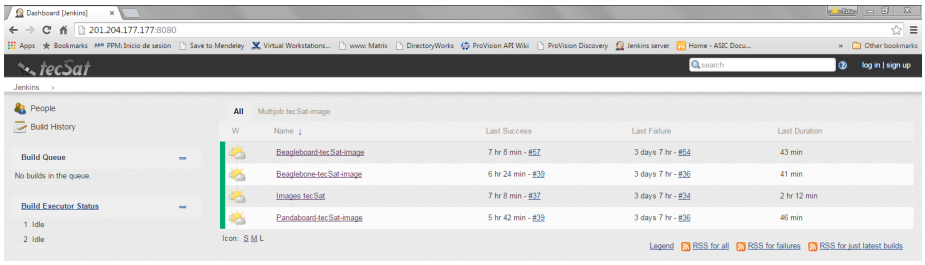

Figure 5.3: Principal Jenkins panel for tecSat project[3]

The front panel of the CI, allow users to show a job for a particular platform, review the basic statistics of construction and have access to the different output artifacts. As the CI system is installed as a build server, and can be access from any explorer, the users can connect even from their cellphones, which allow the analysis of the results from any place as long as connection to the same network as the build server is available. In figure 5.4 access from a cellphone is shown.

[3]Original image. CI front panel for the tecSat project

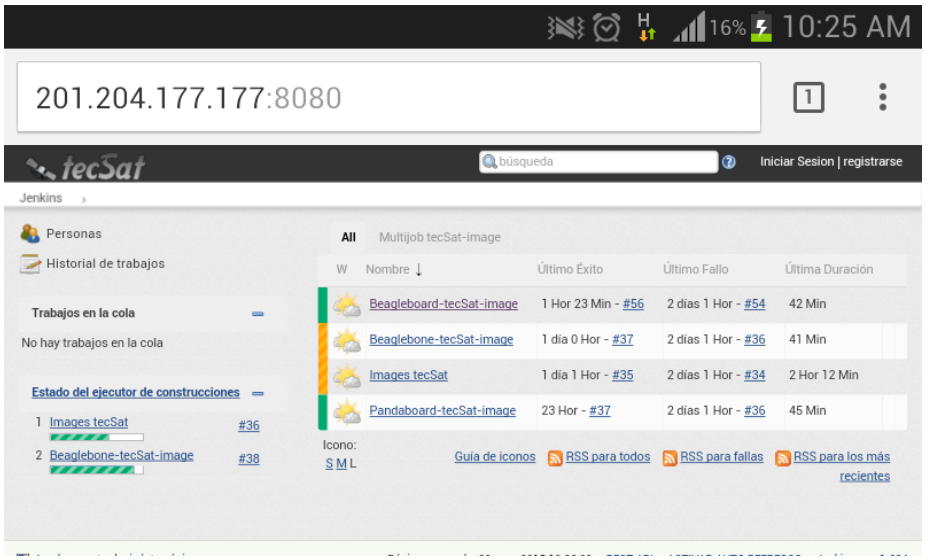

Figure 5.4: Principal Jenkins panel for tecSat project from a cellphone[4]

Each job is created to construct the SDK for the desired machine. It compress the used toolchain and all the necessary packages and configuration to set up the developing environment. The generated SDK created is base on the selected machine as the some features are platform dependent. The creation of the SDK is made taking advantage of the Yocto capabilities. It is possible to export this toolchain and create an output which contain all the necessary to develop pieces of software for the desire platform. This is very useful as one of the methods of software development for small satellite is the incorporation of third parties IP of software. It usually bring integration problems because it doesn't focus on the platform configuration and OS in which will be deployed. The creation of a SDK, can be distributed to the different developers groups, which can use it to create the binary executable that will run on the platform. Furthermore, it can be use to check for complaint of its code with the platform software infrastructure on which will be deployed, reducing compatibility issues. Because of this a job to create an SDK is made, the job panel is shown in figure 5.5.

[4]Original image. CI front panel for the tecSat project accessed from a cellphone

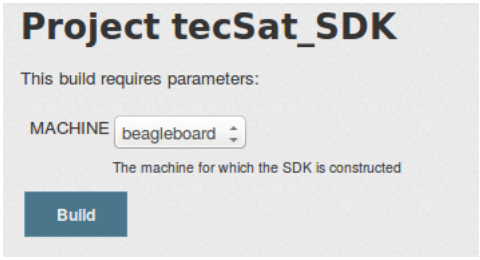

Figure 5.5: Configuration for the tecSat_SDK Jenkins job

The SDK job output its an script, which contains all the necessary to install the toolchain on your developing machine. The installation contains all the necessary for the toolchain to cross compile and deploy your package for the output platform. When you execute the script, you will be asked for the installation path where you want your SDK, then it will unpack in the selected directory. In addition the SDK contains an environment script, which can be source when you are starting your development process. This environment script will add to the environment PATH the cross compiler and set in different environment variables needed to start developing for your machine. For example CROSS_COMPILER, ARCH, LDFLAGS, and many other variables will be added to your environment.

When the user select a job for a selected platform, he can trigger its build process. This process will fetching the latest version of the tecSat project and construct and deploy the image for the selected target. Whether the construction process succeed or fail, an output log can be accessed from the CI system. This log can be also reviewed at construction time, functioning as console output of the running process. For example, on figure 5.6 is shown the displayed output of the construction of the images for the Beagleboard job.

The last function in which the CI system is configured is to access the results of the build. These outputs are configured as artifacts of the Jenkins CI job and can be archived. By archiving the image as artifacts, the Jenkins CI allow the user to access an output image constructed on a specific day, and compare it between builds. On the figure 5.7 the archived artifacts for a Beagleboard build are shown.

[4]Original image. CI job for the SDK creation
[5]Original image. CI output log for the creation of a Beagleboard image
[6]Original image. CI output artifacts for the creation of a Beagleboard image

```
Build Configuration:
BB_VERSION        = "1.23.1"
BUILD_SYS         = "x86_64-linux"
NATIVELSBSTRING   = "Ubuntu-14.04"
TARGET_SYS        = "arm-poky-linux-gnueabi"
MACHINE           = "beaglebone"
DISTRO            = "poky"
DISTRO_VERSION    = "1.6+snapshot-20150520"
TUNE_FEATURES     = "arm armv7a vfp neon callconvention-hard cortexa8"
TARGET_FPU        = "vfp-neon"
meta
meta-yocto
meta-yocto-bsp    = "meta-tecSat_stable:4d2ac6f6df2b3ef98699dd4f7afadb2d994222bb"
workspace         = "(detachedfrom18467e8):18467e87a1ba0d8de4ae6c9c0e44926a27e4a96b"

NOTE: Preparing runqueue
NOTE: Executing SetScene Tasks
NOTE: Executing RunQueue Tasks
NOTE: Running noexec task 1354 of 2025 (ID: 4, ../../workspace/recipes-core/images/tecSat-image.bb, do_fetch)
NOTE: Running noexec task 1356 of 2025 (ID: 0, ../../workspace/recipes-core/images/tecSat-image.bb, do_unpack)
NOTE: Running noexec task 1359 of 2025 (ID: 1, ../../workspace/recipes-core/images/tecSat-image.bb, do_patch)
NOTE: Running task 2021 of 2025 (ID: 10, ../../workspace/recipes-core/images/tecSat-image.bb, do_package_qa)
NOTE: Running task 2022 of 2025 (ID: 8, ../../workspace/recipes-core/images/tecSat-image.bb, do_populate_lic)
NOTE: Running task 2023 of 2025 (ID: 7, ../../workspace/recipes-core/images/tecSat-image.bb, do_rootfs)
NOTE: recipe tecSat-image-1.0-r0: task do_package_qa: Started
NOTE: recipe tecSat-image-1.0-r0: task do_populate_lic: Started
NOTE: recipe tecSat-image-1.0-r0: task do_package_qa: Succeeded
NOTE: recipe tecSat-image-1.0-r0: task do_populate_lic: Succeeded
NOTE: recipe tecSat-image-1.0-r0: task do_rootfs: Started
NOTE: recipe tecSat-image-1.0-r0: task do_rootfs: Succeeded
NOTE: Running noexec task 2025 of 2025 (ID: 11, ../../workspace/recipes-core/images/tecSat-image.bb, do_build)
NOTE: Tasks Summary: Attempted 2025 tasks of which 2018 didn't need to be rerun and all succeeded.
Archiving artifacts
Finished: SUCCESS
```

Figure 5.6: Console output for the tecSat_image_beagleboard Jenkins job[5]

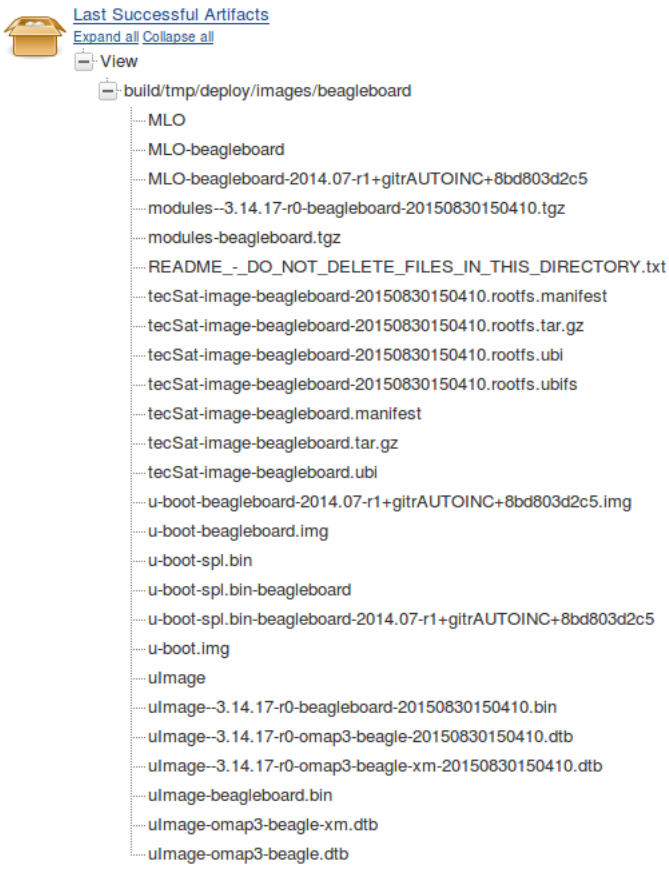

Figure 5.7: Jenkins artifacts on a build[6]

Chapter 6

Development and results using the construction system

On the last chapters, several tools, platforms and proposed solutions have been reviewed. On the present chapter important information recollected is reviewed as well as examples of how the build system can be used to measure and compare platforms, a very useful capability needed on the selection of small computers for small satellites.

6.1 Construction system information recollected from the build system

The measure of the construction system is not an easy task. First is needed the parameters by which it will be measure and it depends on the target application on which it will be used. The space exploration missions, specially the development of small satellites need clean outputs and stability of the code. Its also necessary too use as less as resources as possible to make the project profitable. Because of this the required memory size required for the project, the average construction time and influences over the project are shown.

To recollect the important data about the construction system, the Jenkins CI is used. It is configured to gather statistics about the project and display then in a human readable form.

6.1.1 Construction time

Using the Jenkins CI, data about the construction time required to deploy an image is gather. This information depends on the available resources of the machine in which the developers are making the construction, however this

data gives developers information to compare with the resources available for their projects.

The base system used as a server for the constructions is a machine with an Intel i7-4810MQ. It has eight core processors running at 2.80GHz. The system has 16Gb of RAM memory and is running Ubuntu Desktop 14.04 as the OS. The data is recollected in a period of time around three months, running the construction for all available images one time each day. The recollected data is shown in figure 6.1.

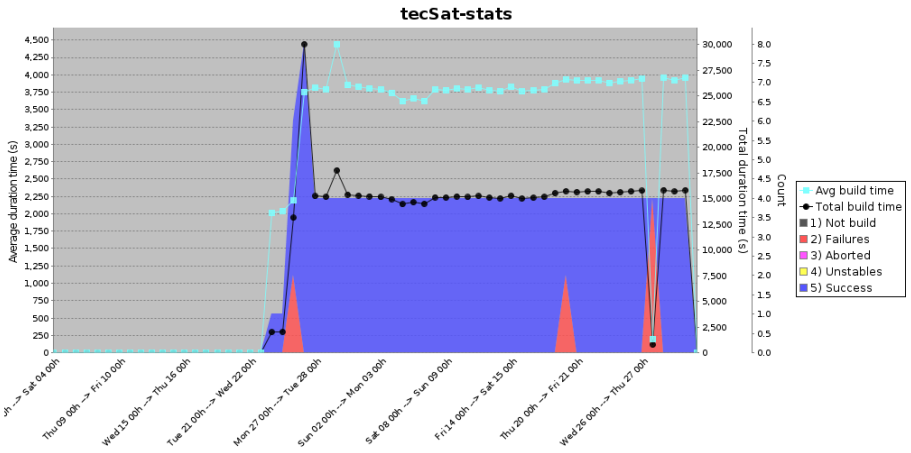

Figure 6.1: Constructions statistics of the different Jenkins jobs[1]

Important information is recollected from the construction statistics, which allow developers to plan ahead in terms of how many time will be necessary to deploy a release for a customer. This results are taking in a machine with a Intel i7-4810MQ with 8 core processor 2.80GHz. The average construction time is shown next.

Table 6.1: Construction time for every platform

Platform	**Average construction time**
Beagleboard XM	41min
Beaglebone Black	40ms
Pandaboard	44ms

The average time in the last table reflects the construction time for a release image. This time doesn't take into account the fetching time for packages, as it depends on the bandwidth of the internet connection. This may result on variations depending on users internet connections. Ignoring the fetching packages, the construction time will be a stable value around users with the

[1]Original image. Statistic data from tecSat project recollected using Jenkins CI

same construction machine resources. The construction time shown represent the whole construction process from its beginning. After the first time the construction process is made, forward construction may take few minutes to be made depending on the deep of the performed changes on the infrastructure. This is because the construction system take advantage of the construction cache of the Yocto build system know as sstates. The sstates make possible to improve the construction speed by only re-build packages with dependencies on the change recipes.

6.2 Required size for construction

A construction system require storage space in the machine its executed. The required space needed may vary between one target platform and another, nevertheless is important to have a reference of how many of my resources will be necessary, including storage. This is very important to have into account from the beginning of the project because it may represent a cost on the budget for the total project. On space exploration projects on rare cases the available storage on the building machines is a limitation, however its important to have it into account.

The average size on the construction for each platform differ because of the packages they contain, however they allow to measure an approximate required for other platforms. In this project the analysis of size for each of the supported platforms are studied and shown in the next table. The data do not include the fetch package, as they may differ strongly from platform to platform.

Table 6.2: Required size of the construction system for each platform

Platform	**Average size**
Beagleboard XM	23Gb
Beaglebone Black	17Gb
Pandaboard	19Gb

As shown in the last table, the construction system required great amount of storage space for its construction. This amount of storage is one of the downsides of Yocto as a build system, which include the construction of each package require, for example the toolchain. Nevertheless this amount of storage require, today systems may support this requirement, where this storage may be in the construction system itself or on a shared storage. If the storage needed for the downloaded packages is taken into account, it may rise up around 5Gb in total.

6.3 Stability of the construction system

The stability of the construction system refer to the reproducibility of the results. It is important to have the same results each time the developer invoke

the process of construction. Its a common error on the software production to encounter strange errors difficult to reproduce. The phrase "it works on my machine" or "it works the second time I configure it" has become a real trouble on the development community of new software packages. Space exploration projects need stable projects with configuration steps that reproduce the same output each time invoke.

The stability of the proposed construction system is analyzed by periodically constructions performed each day. One construction for each available platform is done, performing all the initial configuration from the start. The process include fetch the tecSat project itself, configure, construct and deploy the required output. This information was already shown in figure 6.1. The errors on the construction are represented as red stripes in the chart. Only three errors were detected on the period of time of analysis. The first error was detected as a error introduced on the project itself, which classify as a developer error. The error itself was the introduction of a piece of code not tested enough. It was not capture in the development phase because of the use of previous states on the construction. As our analysis wipe out the entire project and execute the construction process from start, it was possible to capture the error and resolve it right in the moment. The second and third error detected were fetching errors. The main purpose of this errors comes from problems on the connection of the system, making it unavailable to download the pieces of software. This was the case for both errors encounter, were the loss of connection from the machine performing the construction made unavailable to clone the tecSat project itself. It is important to be aware of this kind of errors, as the construction system fetch several pieces of software from the web, the movement or deprecation of those packages may produce errors on the system. Nevertheless this limitation on external packages, the construction system may be configure to fetch this packages from different mirrors on the web, and when a package get deprecated or moved uncommon that package disappear completely.

6.4 Application development of a hardware independent application

The most important feature of the construction system is the fast development and porting of code between different platforms. This allow comparison and fast interchangeability. A probe of concept developing a hardware independent application is develop. This development will show the advantages the construction system have when is necessary to measure the application in different targets in a easy and fast way. In this example the application as been executed on the GPP, allowing the measure of different GPP workloads. The select task is a color space conversion of an image, a computational application which require a lot of resources and is very used on today applications. The color space conversion is RGB to YCbCr. It is an application used mostly for the easy encoding of images and decrease the effect of noise on its transport. It is also

used on space exploration missions combined with several compression methods when an image need to be transferred from a device in the space to a ground station.

6.4.1 Color space conversion, RGB to YCbCr

The color space of an Image is a specific organization of the image color. It allows different representations, some which may be suited for the human interpretation, other for computational advantages like compression, transmission, immunity to noise and others. It allow for a reproducible representation of a color image, digital and analog. Color space is a specific term to talk about the identification and mapping of the color on images, however it can be used also as a method to identify a color model for it.

The most common color model used on image, for its simplicity, is the RGB color space. On this color space every pixel on an image is represented as the combination of three basic colors, Reed, Green and Blue which are called chromaticities. A RGB color space can be explained plain simple as all the color that can be made by the combination of a triangle defined by the three primary colors. This color space is widely used on computer graphics because of its similarity with the human visual system, however its important to make clear it is not the same. Its important to notice that for computational representation, there are finite numbers to use on the representation for each color, so if we define each color as the channels of the image, the number of bits used to represent each channel will be called the depth of the channel. For example, a depth image of 8 bits on each channel, allow the representation of each color form 0 to 255.

The developed application will take a RGB image and convert it to a color space named YCbCr. The YCbCr is a family of color spaces used for video and photography, where Y is called the luma component, Cb the blue difference of chroma component and Cr the red difference of chroma component. It is not really a color space but rather a method to encode RGB information of an image, because in this conversion, although information from the RGB image is loss, is better suited for image transmission. The resulting image do not loss quality for the human eye.

The standard used of the YCbCr image is the BT.601 conversion, which map every pixel directly from the RGB values to the YCbCr values. The mapping between this two color spaces is shown next:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = D * \begin{bmatrix} R \\ G \\ B \end{bmatrix} + C \tag{6.1}$$

Being the values of D and C define a 3x3 and 3x1 matrix in that order.

$$C = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \tag{6.2}$$

$$D = \begin{bmatrix} 0.257 & 0.502 & 0.0975 \\ -0.148 & -0.289 & 0.438 \\ 0.438 & -0.366 & -0.0711 \end{bmatrix} \tag{6.3}$$

This mapping is done on each pixel of the image, obtaining the new representation of the color space. To test the mapping, a model using Octave is created to review the representation of the image.

6.4.2 Development of a hardware independent recipe

The piece of software develop uses only the GPP of the target. It is intended to get an image from a specified location in the filesystem and them process the image. Because the software is not intended to use a specialized hardware of the board, it is possible to include this code in the hardware independent layer. In this layer a directory of recipes-tecSat-examples is created in order to contain the recipe. In addition a directory of files containing the code is created. The code was included because of its simplicity, but more robust pieces of code are recommendable to be created and fetched from its own repositories.

The recipe of this package fetch the code, compile it using the construction tools of the system and deploy it into the build system intended for our target. Next is shown the recipe use to construct this piece of code.

```
# Recipe for color space conversion: RGB to YCbCr
SUMMARY = "RGB to YCbCr image conversion"
DESCRIPTION = "RGB to YCbCr image conversion using only the GPP"

AUTHOR = "Allan Granados Jimnez"
SECTION = "bin"
LICENSE = "BSD"

LIC_FILES_CHKSUM = "file://CMakeLists.txt;md5=67c6ec0c059c284670ff186946a36d0e"

DEPENDS = "libpng"

inherit pkgconfig cmake

SRC_URI = "file://rgb2ycbcr_gpp.c \
           file://CMakeLists.txt"

S = "${WORKDIR}"

OBJECT = "rgb2ycbcr_gpp"

do_install () {
   install -d ${D}${bindir}
   install -m 0755 ${OBJECT} ${D}${bindir}
```

6.4.3 Deployed images for different targets

The construction system is intended to deploy the output images in a easy to see form for the user, because of this an image directory is created on the root of the

project for easy access. For each platform, the result images are studied as the footprint of these image are of importance for small satellites project, been the images on the storage system of the satellites. The footprint of the tecSat image resulting from the construction system depends on which packages are added, however we can measure the default set up of the image, the bootloader, the kernel and its filesystem. It can give us a general view of the required resources for our system. From the available image to construct the study image its the tecSat-image base version. Next are shown the footprint of the tecSat-image for the different platforms.

Table 6.3: Footprint of the different images (minimum packages required)

Image	**Beagleboard**	**Beaglebone**	**Pandaboard**
MLO	48K	100K	NA
u-boot.img	369K	348K	NA
u-boot.bin	47K	NA	185K
uImage	4,1M	4,9M	4,2M
device tree	57K	29K	NA
FileSystem	13M	20M	18M
Total	18M	25M	23M

6.4.4 Result of RBG to YCbCr using the GPP

A C application is created to perform the task of reading an image and convert if from RGB to YCbCr. This application is created in the most basic way and it uses resources only from the GPP of the selected target.

To first test the application, the program is executed to convert the classic image of Lena, the image has dimension for 512x512 pixels. The result are shown of figure 6.2.

Figure 6.2: Color space transformation in the Beagleboard XM. A) Base image. B) Octave representation of algorithm. C) Conversion by the GPP[3]

[3]Original image. Color space transformation using the image of Lena as base.

The first step was to test the conversion algorithm on each of the supported platforms. A custom image of our system was deployed using the Jenkins CI. The Jenkins process made use of the construction system for the image deployment. For each platform the algorithm implementation using C code was run, recollecting the execution time it took to process the whole image.

Table 6.4: Color space transformation on the GPP of the Lena image

Platform	**Execution time**
Beagleboard XM	152,069ms
Beaglebone Black	400,0ms
Pandaboard	125,100ms

As expected, having a better GPP and running a faster clock, the Pandaboard perform the image conversion faster than the other platforms, however this board has many more resources and its bigger that the others not only in size, but in price too. The Beagleboard XM and the Beaglebone run at the same speed, however the core CPU in the Beagleboard is more powerful that the one in the Beaglebone Black and its probe of that is the require time needed to process the same image.

For more detailed testing on the conversion time and the capacities of the boards, a measure using several images is made. For this, a data base of RGB image is used, you can review the used database in [27]. The image has 455 images with dimension of 160x120 pixels. The result of the conversion of the database are shown in the next table.

Table 6.5: Color space transformation on the GPP

Platform	**Average time**	**Standard deviation**
Beagleboard XM	11,102ms	89,248us
Beaglebone Black	28,835ms	9,622ms
Pandaboard	8.586ms	90,86us

This results show how is possible to compare different platforms, however the most important topic is to see how similar images for different targets can be easy deploy in a fast way. In this case three different images are deploy only by changing the hardware dependent layer.

Chapter 7

Future Work

With a stable and robust construction system, the next step on the development process is adding special recipes oriented to the space exploration and to take advantage of the resources available on the different platforms. One example is the development of software which take advantage of the DSP in the Beagleboard XM. On this research, support for the DSPLINK and basic testing is added, giving the possibility for future development using this API.

7.1 Support of DSPLINK API

DSPLINK is foundation software for the inter-processor communication across the GPP-DSP boundary. It provides a generic API that abstract the characteristics of the physical link connecting GPP and DSP from the application [17]. This abstraction layer eliminates the needs from the user to worry about the development of this link from scratch, and focus on the application development. The DSPLINK API is a package of software develop for the Texas Instrument boards, because of this it is a hardware dependent piece of software. The DSPLINK API is a very old and out of date piece of software, however, for the Beagleboard XM (one of our development board) is a very easy way to access the DSP capabilities. New pieces of software like RPM-MESSAGE and REMOTEPROC are way more efficient, however they lack in support for the Beagleboard XM.

DSPLINK support is added to the meta-tecSat-beagleboard layer. It allows development for the DSP on selected target, the Beagleboard XM. The package is encapsulated to be added to the image as a module, including example of how it can be use and how it perform operations. In addition this package include a dependency on another module package, the local power manager of Texas Instrument, therefore it is included on the meta-tecSat-beagleboard layer as well. Local power manager module allow the user to bring up and down the DSP. This operation is needed on every execution of a different application on the DSP, to ensure the memory and resources are in a clean initial state.

DSPLINK execute the DSP/BIOS from Texas Instrument on the DSP processor, while the GPP can be running any kind of OS. Not an specific OS is necessary, as DSPLINK abstract the API calls and the application using an OSAL sub-layer on its development. It means that if you provide the specific functions for the API on a specific OS, you can port DSPLINK and your applications to this new OS. Specific support is already provided to some operative systems like Linux and Windows, however if a new OS support is required on the GPP, it is up to the user to create the abstraction layer for the OS (OSAL).

Depending on the target platform, DSPLINK brings support to a different set of services. The list of the different services can be found in [17]. All these services allow communication and processing capabilities using the DSP and GPP. Some of these capabilities are:

- Basic processor control
- Shared/synchronized memory pool across multiple processors
- Notification of user events
- Mutually exclusive access to shared data structures
- Linked list based data streaming
- Data transfer over logical channels
- Messaging (based on MSGQ module of DSP/BIOS)
- Ring buffer based data streaming

Depending on the desire application, its development may use one or several of these services. It is up to the application developer to analyze and determine the best way it may perform and take advantages of its resources.

The DSPLINK architecture depends on pieces of code in the GPP side as pieces of code on the DSP side. The communication of the different pieces is what create the flow of date in order to our application to work as expected. This flow is already defined by the DSPLINK API, but the developer is the one in charge to take advantage from it. In figure 7.1 is shown the DSPLINK software architecture.

As show in figure 7.1 on the GPP side of the DSPLINK a specific OS is running, on which a OS Adaptation Layer was used to encapsulate generic OS services required by the API instead of using direct OS calls. The Link Driver encapsulates the physical link between the GPP and the DSP using a defined protocol. The Processor Manager keep information for all components and exposes control operations through the API layer. Finally the DSPLINK API layer is basically a wrapper which provide an abstraction layer to the user, it doesn't do to much other than parameter validation [17].

[1] Original image. Representation of the DSPLINK software architecture taken from [17]

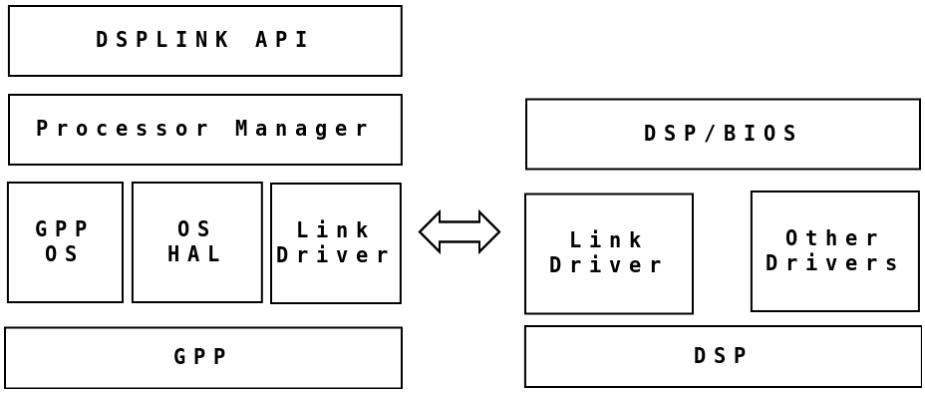

Figure 7.1: DSPLINK software architecture[1]

On the DSP side, the Link Driver is one of the DSP/BIOS drivers on the DSP, specialized in communication to the GPP over the physical link. The communication (data/ message transfer) is done using the DSP/BIOSTM modules-SIO/ GIO/ MSGQ. There are specific DSPLINK API on the DSP for the other modules RingIO, MPCS, MPLIST, NOTIFY, POOL [17].

7.1.1 Validation test on the DSP for the Beagleboard XM

The DSPLINK API from Texas Instrument contain a set of examples, so you can test the operation of the DSP and GPP as also interact with the communication system between processor and co-processor. The examples explore the different methods used to share information, like using buffers, a list or a ring buffer for example. All those examples can be modified and customize for the user to be in a selected application. The available examples are shown next, refer to [17] for further information.

- loopgpp: This sample transfer data between a task running in the GPP to the DSP, then it send it back to the GPP. This example show the creation and use of channels to send and receive data by a buffer in a pool of memory.
- messagegpp: This example show simple messaging between the GPP and the DSP. This message is send to the DSP where it is put in a queue back to the GPP where its verified.
- mplistgpp: Show the use of a MP list component to stream data between the GPP and the DSP using multiprocessor list instances.
- mpcsxfergpp: This sample illustrates data transfer between the GPP and DSP through a basic mechanism of shared buffers with mutually exclusive access protection.

- ringiogpp: This example show the use of a ring buffer to stream data between the GPP and the DSP and back to the GPP from the DSP.
- scalegpp: This sample its a combination of data streaming and messaging between the DSP, where it sends data from a task running in the GPP to another running in the DSP using channels. A message is send to communicate the scaling factor to the DSP in which the data will be scaled.

Following are the results obtained from running the examples.

Table 7.1: Result of DSPLINK examples

Sample	**Buffer size**	**Number of iteration**	**Bytes to transfer**	**Result**
loopgpp	1024	2000	NA	Successful
messagegpp	NA	10000	NA	Successful
mplistgpp	1024	128	128	Successful
mpcsxfergpp	128	1000	NA	Successful
ringiogpp	2048	NA	128	Successful
scalegpp	128	500	NA	Successful

Unfortunately, not all the examples show the average execution time and is up to the developer to analyze and decide which method is better suited for its application. Only the messagegpp sample show the execution time, which for a message to do a round trip, from GPP to DSP and then back to the GPP, is about 282 microseconds.

7.2 RBG to YCbCr using the DSP

The application of color space transformation will be analyzed using a specialized processor, the DSP. It will be intended to be in charge of the mathematical operation of the image transformation. The basic idea is to leverage the mapping to the DSP because this processor can make faster mathematical operations and also do a multiple data single instruction operations in a more efficient way. In this case is intended to perform the matrix operation for the RGB to YCbCr conversion.

The communication between the DSP and the GPP is done by using the DSPLINK API. A common memory space is reserve to pass data between each processor. The design of the application use a buffer and MSGQ API of DSPLINK to transport data between the different processors. This buffers and its channels created an access to the pool on a selected shared region of memory. On figure 7.2 is shown the chosen design approach.

[2]Original image. Representation of communication selected for the RGB to YCbCr application

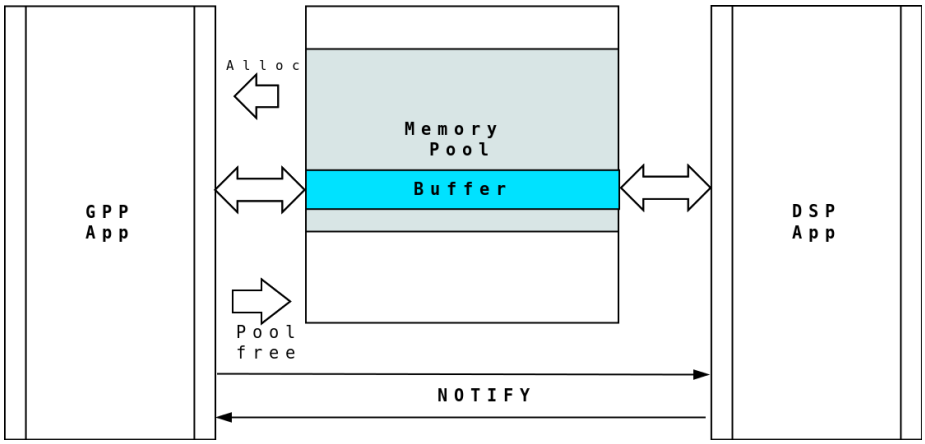

Figure 7.2: Design representation for the rgb2ycbcr_dsp application[2]

As shown on figure 7.2, a selected shared memory region is created as a pool, in which the buffer of data live. The GPP operates with a OS which uses a MMU, translation between virtual address and physical address is required. The OS of the DSP, the DSPBIOS, doesn't have a MMU, for that reason there is a direct mapping of its memory and the shared memory, performing direct access to the memory.

```
NOTE: At first the design only contemplate a single transaction of all data,
      however, on image analysis, great amount of data is required, so the
      design need to be changed to use multiple transactions. For example,
      a 512x512 image, required 786432 bytes of processing. Also the test
      of loopgpp demonstrate a maximum transfer of 16Mbytes of data in a
      single transaction of the buffer.
```

The basic flow of the application begin on the GPP. It start the DSP and send the executable it will run from the GPP to the DSP. When the DSP has the application, the GPP open a pool on the shared memory space, the size of the pool will be enough to have two communication buffer with two channel for each processor to access, one to read and another to write. The creation of all this components are perform on the setup of the application. When ready to start the execution, the GPP will open the image and fill the buffer with the pixel information of the image. On the DSP side of the application, the attach to the buffer and the pool is done in the setup. In the execution time, the DSP receive data from the GPP using the buffer. This data is obtained by the DSP and the algorithm of color space conversion is performed. The resulting data is place in the output buffer of the DSP to be send to the GPP side of the application, to finally be part of the resulting image.

The code can be located of the application is located in the next URL, however, it is not complete yet. It is available to any developer to access it and review it.

```
https://github.com/allangj/rgb2ycbcr-dsp
```

The code on the repository is divided in the GPP side and the DSP side following the structure of the DSPLINK source examples, this is because of the construction recipe deploy for this kind of application. A recipe were created on which the application code is added as one of the DSPLINK examples. A rebuild of the source examples is done in order to include the new code. This allow to generate the executable for the application and insert those on our final image. Developers implementing a new application using the DSPLINK, must create a recipe for its construction, were the next recipe can be take as example.

```
# Recipe for color space conversion: RGB to YCbCr using DSPLINK
SUMMARY = "RGB to YCbCr image conversion using the DSP"
DESCRIPTION = "RGB to YCbCr image conversion using only the DSP"
AUTHOR = "Allan Granados Jimnez"
HOMEPAGE = "https://github.com/allangj/rgb2ycbcr-dsp"
SECTION = "bin"
LICENSE = "GPLv2"

INSANE_SKIP_${PN} = "installed-vs-shipped"

require ../../recipes-ti/includes/ti-paths.inc
require ../../recipes-ti/includes/ti-staging.inc

DEPENDS = "libpng ti-dsplink-module ti-dsplink-examples ti-lpm-module"
PROVIDES = "rgb2ycbcr-dsp"


BRANCH = "master"

SRCREV = "0f631084775595acc00b9d25d2f645dc78dabc10"
LIC_FILES_CHKSUM = "file://${WORKDIR}/git/README.md;md5=d1381886e92ddc20cf4ff7e97bfa4f9a"
SRC_URI = "git://github.com/allangj/rgb2ycbcr-dsp.git;branch=${BRANCH}"


DSPLINK = "${LINK_INSTALL_DIR}/dsplink"

S = "${WORKDIR}/git"

DSPLINKPLATFORM_omap3     = "OMAP3530"
DSPLINKPLATFORM          ?= "<UNDEFINED_DSPLINKPLATFORM>"

DSPLINKDSP_omap3          = "OMAP3530_0"
DSPLINKDSP               ?= "<UNDEFINED_DSPLINKDSP>"

TECSAT_INSTALL_DIR_RECIPE = "${installdir}/tecsat-examples"
do_configure() {

    echo "------------------Configuring rgb2ycbcr-dsp-------------------"
    # Create symbolic links between this app and the DSPLINK samples folders
    ln -s -f ${S}/gpp ${DSPLINK}/gpp/src/samples/rgb2ycbcr-dsp
    ln -s -f ${S}/dsp ${DSPLINK}/dsp/src/samples/rgb2ycbcr-dsp
    # Move the DIR file to use custom ones
    mv ${DSPLINK}/gpp/src/samples/DIRS ${DSPLINK}/gpp/src/samples/DIRS.old
    mv ${DSPLINK}/dsp/src/samples/DIRS ${DSPLINK}/dsp/src/samples/DIRS.old
    ln -s ${S}/DIRS_GPP ${DSPLINK}/gpp/src/samples/DIRS
    ln -s ${S}/DIRS_DSP ${DSPLINK}/dsp/src/samples/DIRS
```

```
}

do_compile() {
    echo "------------------COMPILING-------------------"
    # Build the gpp samples
    echo "------------------GPP SAMPLES-------------------"
    cd ${DSPLINK}/gpp/src/samples && make \
      BASE_TOOLCHAIN="${TOOLCHAIN_PATH}" \
      BASE_CGTOOLS="${BASE_TOOLCHAIN}/bin" \
      OSINC_PLATFORM="${TOOLCHAIN_PATH}/lib/gcc/${TARGET_SYS}/$(${TARGET_PREFIX}gcc \
      -dumpversion)/include" \
      OSINC_TARGET="${BASE_TOOLCHAIN}/target/usr/include" \
      CROSS_COMPILE="${TARGET_PREFIX}" \
      CC="${TOOLCHAIN_PATH}/${TARGET_PREFIX}gcc" \
      LD="${TOOLCHAIN_PATH}/${TARGET_PREFIX}gcc" \
      AR="${TOOLCHAIN_PATH}/${TARGET_PREFIX}ar" \
      COMPILER="${TOOLCHAIN_PATH}/${TARGET_PREFIX}gcc" \
      LINKER="${TOOLCHAIN_PATH}/${TARGET_PREFIX}gcc" \
      ARCHIVER="${TOOLCHAIN_PATH}/${TARGET_PREFIX}ar" \
      KERNEL_DIR="${STAGING_KERNEL_DIR}" \
      DSPLINK="${LINK_INSTALL_DIR}/dsplink" \
      all

    # Build the dsp samples (debug and release)
    echo "------------------DSP SAMPLES-------------------"
    cd ${DSPLINK}/dsp/src/samples && make \
      BASE_CGTOOLS="${CODEGEN_INSTALL_DIR}" \
      BASE_SABIOS="${BIOS_INSTALL_DIR}" \
      DSPLINK="${LINK_INSTALL_DIR}/dsplink" \
      all
}

do_install () {

    # Install the example apps (gpp and dsp)
    install -d ${D}${bindir}
    install -m 0755 \
    ${DSPLINK}/gpp/export/BIN/Linux/${DSPLINKPLATFORM}/RELEASE/rgb2ycbcr-dspgpp ${D}${bindir}
    install -m 0755 \
    ${DSPLINK}/dsp/export/BIN/DspBios/${DSPLINKPLATFORM}/${DSPLINKDSP}/RELEASE/rgb2ycbcr-dsp.out \
    ${D}${bindir}
}
```

Chapter 8

Conclusion

On the software development for space exploration systems there is a great amount of possibilities in the work been develop around the world. The present research allow to implement a construction system intended to speed up the process of software development and reduce the complexity and cost of it. By the introduction of a construction system for the software development of small satellites several conclusions are made.

A construction system can reduce the complexity of the construction and integration of code. By reducing the number of commands needed for configuration and construction, the possibility of errors are reduced. This reduction is achieved by the incorporation of abstraction between the construction process, the pieces of software and the interaction with the user and developer.

The incorporation of Yocto as the core build system allow the creation of abstraction layers. It separates the pieces of software which depends on the hardware from the ones who don't. This separation include the benefit of reducing the software development necessary for the migration of the system between platforms. This increase on the escalation of software create opportunities to reduce the development and debugging time. In addition, the pieces of software can evolve to be more robust and incorporate new functionality and abstraction. The inclusion of a new supported platform only need the development of the corresponding abstraction layer. Interchangeability between different platform in a easy way depends only on the available platform dependent layers and a configuration file. On space exploration mission this allow the quick deployment of a satellite in a totally new platform.

The develop construction system prove to be stable in time. Errors coming from the developer side were hardly found, and when there were present, the use of the CI system allow for an early detection and correction. Having the construction system in the web prove to add a dependency on the internet connection. However the construction system has the functionality to work in a internal network or totally local in the system, if the packages were fetched at least one time.

The approximate construction time of the system depends on the selected

platform. It proves to required around an hour of execution. It separates the different construction packages in order to speed up the process. The construction time may be high, however this time is required only if full construction. In the cases were only some packages were change, the construction time may take only a few minutes. The analysis of the construction time prove to be stable, requiring the same amount of time for a complete build to be deployed. This predictable behavior is very desirable on the system. It allow developers to know for certain how much time is required in order to deploy an image.

Space exploration mission hardly have the available time for innovation in pieces of code. The construction systems allow the easy development and incorporation of new code. This new code can be ease develop and tested against the same compilation tools used to create the software for the satellite. It reduces the integration risk of new code and open the door for incorporation of pieces of software for the innovation. An example of this development was tested by the introduction of a image color space transformation code. Once develop this software, it was easy to test it in different platforms without the need to modify it or perform adaptations. It opens the door for new methods on the software development for small satellites, seeking to remove the dependencies on the development platforms.

Using the construction system, the possibility of build and compare a piece of software was possible. Developing a simple color image transformation as a hardware independent software, it was ease to deploy a testing image on each of the available platforms. The importance of having the possibility to compare target platforms in a easy way is a very important advantage on the design of small satellites. It allow developers to test the most important software on different platform and decide which fit best the requirements. In addition, gives the opportunity for a developer on a space exploration mission to early develop software, an later change to the final target.

The required configuration for the construction system was able to be abstracted from the user. However the possibility of adding new packages is always possible. The basic configuration for each system are intended to have all the required packages, removing the need of analysis from the user side, which may introduce errors. However it gives the possibility to developers who knows the architecture of the construction system to include new pieces of software and extend the functionality. In addition, the construction systems include the functionality of construction of only the required package, which speed of the development and testing of a particular piece of code.

Bibliography

[1] Keith Avery, Jeffery Finchel, Jesse Mee, William Kemp, Richard Netzer, Donald Elkins, Brian Zufelt, and David Alexander. Total Dose Test Results for CubeSat Electronics. *2011 IEEE Radiation Effects Data Workshop*, pages 1–8, 2010.

[2] Beagleboard.org. Beagleboard, official page. `http://beagleboard.org/`, 2015. [Online; accessed 08-January-2015].

[3] D Brosnan Blázquez. Analysis of WCET in an experimental satellite software development. i:1–10, 2012.

[4] Pieter Johannes Botma. The Design and Development of an ADCS OBC for a CubeSat. (December), 2011.

[5] Fredrik Bruhn, Lars Asplund, Kjell Brunberg, John Hines, Independent Consultant, San Francisco, and Magnus Norgren. Introducing Radiation Tolerant Heterogeneous Computers for Small Satellites. pages 1–10, 2015.

[6] Johan Carvajal, Morteza Haghaayegh, Jaan Viru, Jian Guo, and Allan Granados. Increasing computing performance of ADCS subsystems in small satellites for earth observation (IAA-B10-0901).

[7] Gerald Coley. BeagleBone Black System Reference Manual. page 239, 2013.

[8] Cubesat. Cubesat program, official page. `http://www.cubesat.org/`, 2014. [Online; accessed 25-June-2014].

[9] MJ Dabrowski. The design of a software system for a small space satellite. 2005.

[10] Daniel D Gajski. System-Level Design Methodology. 2003.

[11] Xiang Gan, Jori Dubrovin, and Keijo Heljanko. Electronic Communications of the EASST 11th International Workshop on Automated Verification of Critical Systems (AVoCS 2011) A Symbolic Model Checking Approach to Verifying Satellite Onboard Software A Symbolic Model Checking Approach to Verifying Sate. 46, 2011.

[12] Inc. Gentoo Foundation. Geento, official page. `https://www.gentoo.org/`, 2015. [Online; accessed 17-August-2015].

[13] John Gruenenfelder. Operating system for control of small satellite systems. *... AIAA/USU Conference on Small Satellites, ...*, pages 1–8, 2002.

[14] Hank Heidt, J Puig-Suari, A Moore, S Nakasuka, and R Twiggs. CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation. 2000.

[15] Lucy Hoag, Tatiana Kichkaylo, and David Barnhart. A systems architecting approach to automation and optimization of satellite design in SPIDR. *International Conference on Engineering ...*, 2010.

[16] Stuart Hughes. LTIB, official page. `http://ltib.org/home-intro`, 2015. [Online; accessed 17-August-2015].

[17] Texas Instrument. DSPLINK User guide. 2010.

[18] I Introduction. Implementation of a Real-Time Operating System on a Small. pages 1–9, 2007.

[19] Hirofumi Kawakubo. Hardware Development of a Microcontroller Board for a Small Satellite. pages 1–9.

[20] Bart Kienhuis, Deprettere Ed, Pieter Van der Wolf, and Kees Vissers. A methodology to Design Programmable Embedded Systems. *Article*, 2001.

[21] Peter Korsgaard. Buildroot, official page. `http://buildroot.uclibc.org/about.html`, 2015. [Online; accessed 13-July-2015].

[22] Marcio Kreutz, Cesar a. Marcon, Luigi Carro, Flavio Wagner, and Altamiro a. Susin. Design Space Exploration Comparing Homogeneous and Heterogeneous Network-on-Chip Architectures. *2005 18th Symposium on Integrated Circuits and Systems Design*, (August 2015):190–195, September 2005.

[23] Fu-Ren Lin, Meng-Chyn Yang, and Yu-Hua Pai. A generic structure for business process modeling. *Business Process Management Journal*, 8(1):19–41, 2002.

[24] Chase Maupin, Henry Wiechman, and Denys Dmytriyenko. The Yocto Project : Changing the way embedded Linux software.

[25] Gerry Rozema Mike Baker. OpenWRT, official page. `https://openwrt.org/`, 2015. [Online; accessed 17-August-2015].

[26] Atif Mohammad, Jeremy Straub, Christoffer Korvald, and Emanuel Grant. Model-based software engineering for an imaging CubeSat and its extrapolation to other missions. *2013 IEEE Aerospace Conference*, pages 1–6, March 2013.

[27] University of Edinburgh. afreightdata image data base. `http://homepages.inf.ed.ac.uk/amos/movies/afreightdata.zip`, 2015. [Online; accessed 08-May-2015].

[28] Openembedded.org. OpenEmbedded, official page. `http://www.openembedded.org/wiki/Main_Page`, 2015. [Online; accessed 17-August-2015].

[29] Pandaboard.org. Pandaboard ES Hardware System Reference Manual. 2013.

[30] Pandaboard.org. Pandaboard, official page. `http://pandaboard.org/`, 2015. [Online; accessed 08-January-2015].

[31] Marco Panunzio and Tullio Vardanega. A Component Model for On-board Software Applications. *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 57–64, September 2010.

[32] A Linux Foundation Collaborative Project. Yocto build system, official page. `https://www.yoctoproject.org/`, 2013. [Online; accessed 25-June-2014].

[33] Juan A De Puente. Design of On-Board Software for an Experimental Satellite. 01:1–10.

[34] European Cooperation for Space Standardization. ECSS-E-ST-40C. (March), 2009.

[35] Texas Instruments. BeagleBoard-xM System Reference Manual Rev C1. pages 1–164, 2010.

[36] Letzte nderung. Pengutronix, official page. `http://www.pengutronix.de/software/ptxdist/`, 2015. [Online; accessed 17-August-2015].