Putting the Pieces Together: The Systematic Development of a Software Defined Radio Toolflow for the Rhino Project

Gordon Eric Inggs

A dissertation submitted to the Department of Electrical Engineering, University of Cape Town, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Cape Town, May 2011

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in Engineering in the University of Cape Town. It has not been submitted before for any degree or examination in any other university.

Signature of Author . . . . . .

Cape Town

12/05/2011

Abstract

This dissertation is concerned with the thesis that it is possible for a software defined radio system that has been described in accordance with synchronous dataflow theory to be implemented upon a reconfigurable computing platform. This thesis is addressed by considering the systematic development of a software defined radio toolflow for the Rhino System, a set of tools that allows end-users to define such software defined radio systems for the Rhino reconfigurable computing platform.

The development process begins by considering the problem posed by such a toolflow, and the context of such a problem. Relevant topics in systematic development and software defined radio utilising reconfigurable computing are then considered. That systems analysis theory is then applied to the Rhino System development process to derive the specification for a software defined radio toolflow for the Rhino context, which is comprises a subsystem of the Rhino System. In response to this specification, and considering the relevant software defined radio work, a software defined radio toolflow is conceptualised and a development model outlined. This development model requires the development and implementation of two software defined radio prototype systems, a low pass filter and discrete Fourier transform, as vehicles for the development of the toolflow. It concludes by considering the nature of the fulfilment of the specification, and implications for Systems Engineering, Software Defined Radio and the Rhino project.

Dedication

The popular poem by Robert Frost, The Road Not Taken speaks to the joy of exploration and the search for significance as we move through life. I think that this seeking is one of the fundemental aspects of the academic endeavour, and it defines us all as scientists and engineers.

However, I do not believe that this seeking can occur without the support of others. I would like to dedicate this dissertation to everyone in my life who has supported me over the past 17 months, especially my understanding parents, Mike and Trish, and my amazing girlfriend, Rebecca.

Acknowledgements

Firstly I must acknowledge the good work and assistance provided by my colleagues in the Software Defined Radio Research Group, as well as the Rhino Project Leader, Dr Alan Langman. Especially thanks must go to my excellent supervisor, Dr Simon Winberg.

Technical support was provided by both Mr Graham Inggs of the Department of Chemical Engineering at UCT, as well as Ms Janet Hewitson of the Council for Scientific Investigation and Research's Centre for High Performance Computing, Cape Town, and for this I am extremely grateful.

Finally generous funding support has been provided from the South African National Defense Force's Project Ledger, the Square Kilometre Array South Africa Project and the University of Cape Town.

Contents

List of Figures

Chapter 1

Introduction

The objective of this dissertation is to show that *it is possible to describe a software defined system using synchronous dataflow theory, and then implement that model upon a reconfigurable computing platform*. In order to demonstrate that this objective has been achieved, a set of tools will be structured to perform the translation from an abstract model described using synchronous dataflow theory into a practical implementation that runs on the Rhino reconfigurable computer platform. As this set of tools shall facilitate a transistion from the abstract to a practical implementation, the term toolflow shall be used to describe them.

The development of this toolflow has occurred in the context of Rhino[1], a project carried out by Software Defined Radio Group at the University of Cape Town. The Rhino Project seeks to provide a system for research and training in the field of software defined radio for the application domains of Radar, Telecommunications and Radio Astronomy[2]. The tools created as part of this dissertation are intended to form part of the Rhino System. Being part of such a project provides a complete ecosystem within which the aforementioned objective may be comprehensively evaluated, and a real world problem against which the underlying theoretical question may be considered.

This chapter further elaborates upon the nature and scope of the practical need suggested by the thesis under investigation. As expressed above the practical need inferred may be conceptualised as a structured set of tools that enable and facilitate the implementation of software defined radio systems that have been described using the theoretical description methodology of synchronous dataflow modelling. Furthermore the theoretical approach and research methodology adopted and employed in addressing this need will be discussed. It begins by stating the problem that initiated this research clearly and simply, and then frames it within the context of the

[1] **R**econfigurable **H**ardware **I**nterface for computi**N**g and radi**O**

[2] The specifics of the requirements for the project, and hence the scope of Software Defined Radio applications will be discussed in Chapter 3 as well as in Appendix A

current state of the fields of software defined radio and heterogeneous computing. This context is key to identifying the placement of the project within the subject domain, and the scope of the topics that were considered. Then the theoretical approach, the academic strategy employed in formulating the dissertation's research methodology, and hence the means by which the thesis will be evaluated. Finally the chapter concludes by outlining the research methodology employed, the series of interrelated research activities that comprise the investigation of this dissertation.

1.1 Problem Statement

As the extent of digital electronics expands, both in terms of functionality and performance, it has become increasingly possible to implement signal processing tasks in the digital domain. Progress in this field of digital signal processing has been rapid, and of late, a new paradigm named software defined radio has been proposed for processing signals in terms of their application domain. This new approach will be discussed in greater deal in the Project Context section and throughout this dissertation, however what should be noted at this point is that this paradigm seeks to leverage as many general data processing techniques and technologies throughout any particular radio frequency signal processing system as is possible.

While many research projects have focused on developing platforms and tools for prototyping software defined radio systems [52, 11, 31, 48], little attention has been paid to providing platforms that prioritise education and training in this field. Instead of a focus on skills training, existing development platforms tend to rely heavily on the users having experience in the broader area of this subject domain, such as a good understanding of high performance embedded computing and signal processing theory, which are then built upon to utilise these tools[11, 52, 13]. The Rhino project at the University of Cape Town seeks rather to focus on providing a development platform that prioritises training and research in software defined radio, and makes no assumptions as to knowledge of the nature of the tools being used.

An assumption that is safe to make is that this system for software defined radio education will comprise of a significant software component, and in particular software that will facilitate the implementation of the user's software defined radio system on the computing platform of choice. Such collections of software that map abstract, user-defined systems to practical systems are typically called toolflows or toolchains[11, 31, 13][3], and it is the development of such a toolflow for the Rhino System that this dissertation is concerned.

Furthermore, in seeking to create a toolflow for the Rhino Project, the fundamental features of such tools would be derived and implemented. And that implementation would be assessed in

[3] In this dissertation the term toolflow shall be used

terms of the features derived for the toolflow. Doing so would not only provide insight into the software defined radio tools developed for the Rhino platform, but also to the broader viability of implementing software defined radio systems upon reconfigurable platforms, the underlying thesis of this dissertation.

1.2 Project Context

This endeavour has not occurred in a vacuum, and indeed it is at the intersection of two emerging fields:

- *Software defined radio*, the use of computing technologies to perform signal processing operations for the radio frequency application domain[4].
- *Heterogeneous computing*, the use of a combination of distinct computing technologies in order to leverage not only improved performance, but also a greater net performance than merely the aggregate performance of the individual components of the system[9, 23].

Rhino is a platform targeted at training and research in software defined radio, consequently it is the current state of the field that will inform the context of any tools developed that pertain to the platform. In particular the current uses of the paradigm will contextualise the existing software defined radio tools discussed in the Literature Review. And that discussion will in turn reflect upon the tools developed as part of this dissertation.

Heterogeneous computing does not appear to be as immediately relevant, yet is in fact deeply so. As software defined radio, and hence the Rhino platform, seeks to perform signal processing operations in the digital domain at radio frequencies, highly flexible computing systems are required. Furthermore these computing systems will need to perform a vast array of functions optimally, often in constrained operating environments. As a result, it is highly unlikely that one computing technology will provide these functions at the level of performance required, and so out of necessity a variety of cooperative computational technologies are necessary. Thus the current state of the study of such heterogeneous computing technologies will inform the context within which the proposed toolflow will be developed.

1.2.1 Software Defined Radio

Software defined radio is the broad term used to describe an electromagnetic signal processing design methodology to complete as much of a signal processing system[4]using general purpose

[4]***System*** in this case is the term used to refer to an entity with a clearly defined set of input and output signals[46]. More generally, a system is grouping of entities which serve a defined set of purposes[10]

computing[5] technologies as is possible. Thus this approach is inherently a form of digital signal processing, as it requires the signal under consideration to be in digital form and all of the operations which are being applied to it be defined in terms of the underlying architecture. The ideal that is being striven for is a truly flexible, multi-functional technology that can freely interact with the radio electromagnetic spectrum[31, 7, 13]. However due to constraints upon the sampling of analogue electromagnetic waveforms, as well as the limitations of data processing technologies, this ideal is some way off[31]. The practical realities of the field complicate the definition of the field somewhat, and the full semantic rationale as to why software defined radio can be called a digital signal processing paradigm shall be discussed further in the literature review.

The relevance in determining the scope of the project is that the performance currently achievable for software defined radio systems have severely limited its direct application in broader commercial and industrial activities, particularly in mobile devices[31]. However there has been widespread adoption as a technique for rapidly prototyping signal processing systems intended for these application domains, which are then later "crystallised" into Application Specific Integrated Circuits (ASIC). There is also recognition that the constraints on signal sampling and embedded processing are rapidly being overcome, and that the field application of software defined radio particularly in fields such as Cognitive Radio is rapidly approaching. Thus this project is placed squarely in the context of a field which is on the cusp of commercial adoption, with considerable use as a development space currently.

1.2.2 Heterogeneous Computing

Digital signal processing is not the only technical domain which has been experiencing a paradigm shift of late. Due to several physical limitations reached with regards to heat dissipation and transmission line effects, the oft-quoted "Moore's Law" is no longer translating into the reliable improvement in monolithic processing unit speed that it has for the last 40 years[3]. In order for the conventional Von Neumann machine to maintain the improvement in data processing, multiple processing "cores" have been introduced into central processing units. The transition from simply doing computer processing operations faster to doing multiple operations in parallel has stopped being a specialised branch of high performance computing, and become an issue of import for computing in general. Furthermore, this has presented an opportunity for several other computing technologies to come to the fore, such as the use of graphics processing units for general purpose computing and reconfigurable computers such as field programmable gate arrays, because of the equalisation of development time and effort[23, 3]. This

[5] **General Purpose** computing technologies are those which implement a standardised set of operations, usually the Reduced Instruction Set Computing (RISC) or x86 instruction set processors popularised by Intel[3]

has given rise to the sub-field of heterogeneous computing, which is concerned with enabling and optimising the interactions between the various computing technologies available.

Thus the challenge presented in developing in the context of the Rhino project is to ensure that this trend in general purpose computing is acknowledged, and taken advantage of when developing a platform for software defined radio research and training. In particular, this pertains to the scope of the software toolflow, as it is typically the "glue" which holds the different technologies together into a coherent system. So not only will the software toolflow for Rhino allow for development of software radio systems, but it should also one which should consider and accommodate these new and emerging technologies.

1.3 Theoretical Approach

This section outlines the academic strategy employed in addressing the problem stated above, and hence the underlying thesis of this dissertation. This strategy, in conjunction with the problem statement and context, was used to formulate the research methodology. First the key conceptual tool of Systems Engineering is introduced, then the broader development process of the Rhino System. Finally the relationship between the development process undertaken as part of this dissertation and the Rhino System is clarified, and the theoretical approach of the work undertaken is apparent.

1.3.1 The Systems Engineering Approach

Central to the analysis of the problem, and the development of the proposed software defined radio toolflow is the concept of Systems Engineering. This theoretical approach was adopted because the problem under consideration is not only multi-faceted, but the facets are also strongly interrelated. Systems Engineering arose in response to addressing such problems during the 20th century, and provides a comprehensive framework for not only analysis and understanding, but also for addressing the needs that suggest such problems[10, 53].

As the name implies, Systems Engineering is the solving of a problem by rigorously analysing all of its facets in a structured manner, and then conceptualising a solution to that problem that incorporates only the necessary elements or components into a cohesive unit or *system*, as suggested by that analysis [10]. Systems engineering is a vast topic, with many nuances particularly relating to its relationship with the discipline of project management. The broader field of systems engineering and several topics within it that are of relevance to this project will be addressed further in the literature review.

1.3.2 The Rhino Development Process

An overview of the systems engineering process developed for Rhino is presented in Figure 1.1. The process is comprised as three distinct phases: requirements analysis, system architecture description and the detailed subsystem design; with 2 bridging elements between the phases, the System Requirements and System Architecture.

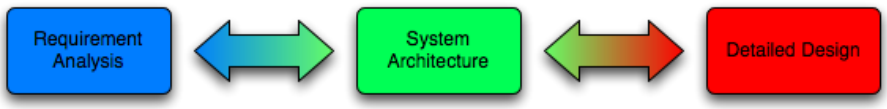

Figure 1.1: Rhino Project Development Process

A brief overview of the three phases of the Rhino development process[6]:

- The *Requirements Analysis* seeks to characterise the problem under consideration in full, forming a clear link from the needs of the stakeholders to a set of deliverables for the system. These deliverables are the system requirements, providing a comprehensive and unambiguous description of what is required of the system.
- The *System Architecture* description brings together the functions and physical structure of the system, so as to create a system architecture that provides an outline of the constituent components of the system that will fulfil the system requirements, and the relationship between these components.
- The *Detailed Design* is the realisation of the system architecture through the development and detailed documentation of the subsystems identified as comprising the system architecture. These subsystems might be developed in particular groupings, such as the software defined radio toolflow or the computing platform, for practical reasons.

While it is beyond the scope of this dissertation to conduct an exhaustive systematic study of the Rhino project, elements of that analysis are relevant to the development of the toolflow.

1.3.3 Relationship between the Rhino Development Process and the Toolflow Development

The development of a software defined radio toolflow as described by this dissertation forms part of the third phase of the development of the Rhino System, as the toolflow is demonstrated

[6]The full Rhino Development Process Document may be found in Appendix A

to be a subsystems within the system architecture. As a result of this relationship, the Rhino development process provides a robust analytical framework for further analysis of the problem under consideration.

By revisiting and deepening the analysis conducted in the first two phases upon the system architecture and requirements relevant to the software defined toolflow[7], further insight may be provided into not only possible solutions to the problem, but also the fundamental nature of the problem under consideration. The positivist nature of the analytical process allows for the context of implementation and the academic question to be clearly differentiated between, and hence allows for the latter to be addressed by comprehensively addressing the former.

Thus the technical approach adopted by this dissertation is to develop systematically a software defined radio toolflow within the broader development process of the Rhino System. And as a result of the analysis undertaken during this development process, the thesis of the dissertation will be comprehensively addressed.

1.4 Research Methodology

The Section above described the Systems Engineering-based theoretical approach that this dissertation makes use of in considering the problem stated above. This methodology describes how this strategy is translated into a series of interrelated research activities[8] in service of achieving the objective of this dissertation, and provides an overview of those activities. Figures 1.2 and 1.3 show the flow of the arguments through the dissertation, both within and between the chapters, as described below.

[7]The precedent for doing so, with regards to linearity in the systems engineering process shall be discussed in Chapter 2

[8]Each major, distinct activity comprises a chapter

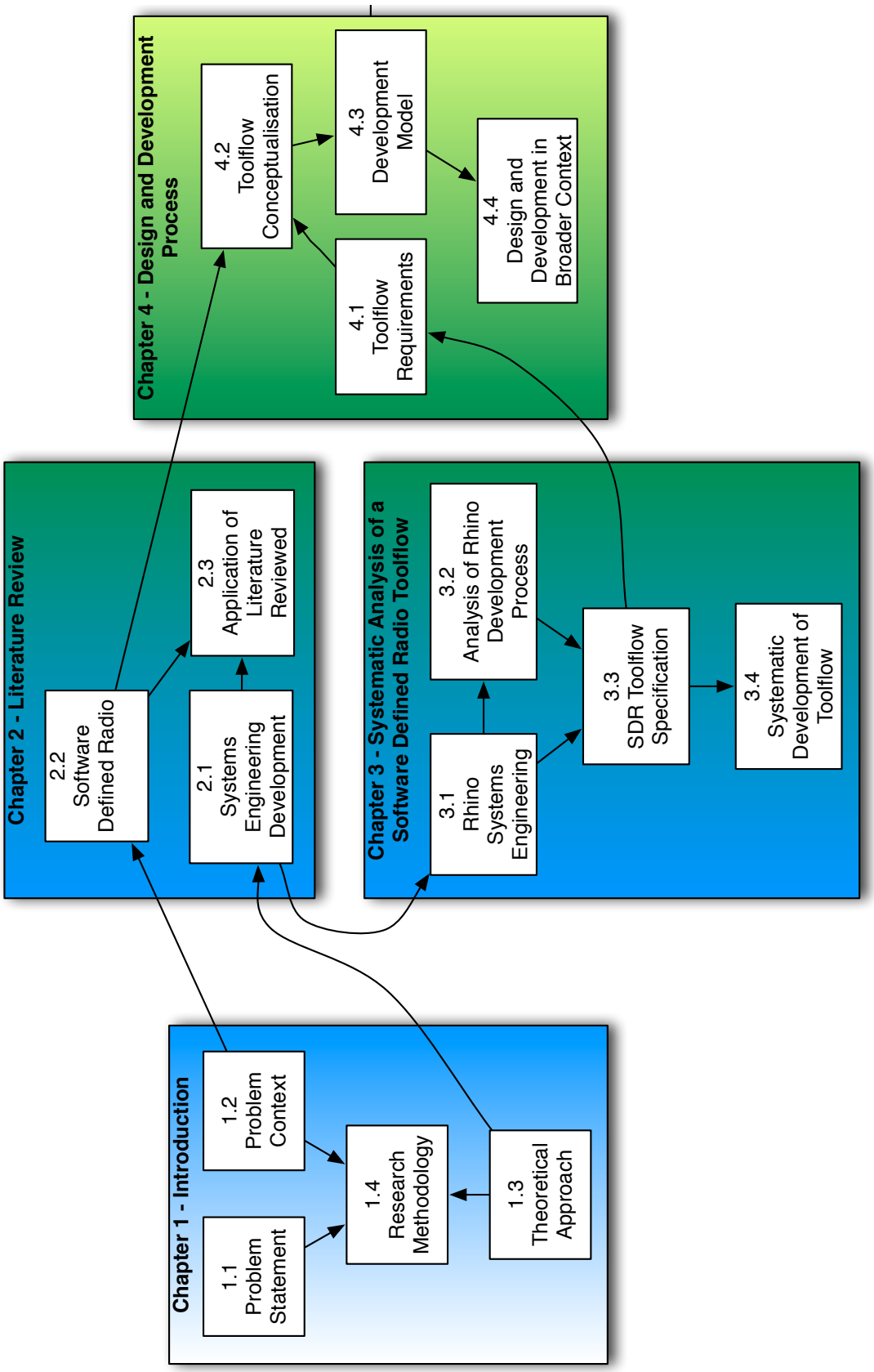

Figure 1.2: Flow of Arguments in the Analytical Chapters

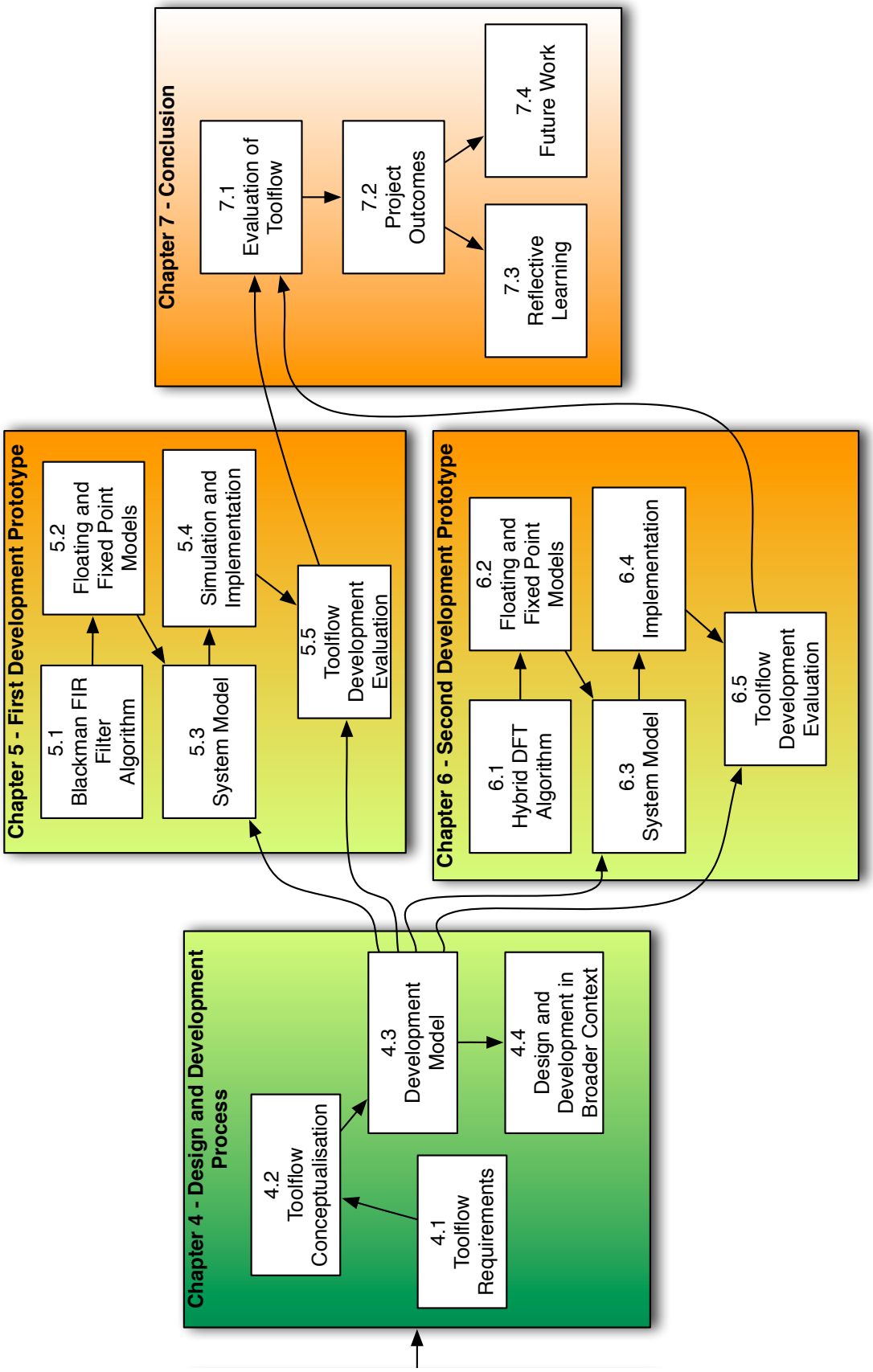

Figure 1.3: Flow of Arguments in the Development Chapters

1.4.1 Chapter 2 - Review of Relevant Literature

Chapter 2 presents a review of current literature in order to more fully elaborate the context of the field of software defined radio as well as the systems-based development theory, both introduced in this chapter. This elaboration is grouped under two themes: Systems Engineering Development and Software Defined Radio.

Under the theme of Systems Engineering Development, the concepts of systematic development, the systems modelling language (SysML) and Traceability will be introduced. The purpose of this analysis will be to focus the extremely broad field of Systems Engineering into several topics of relevance to this dissertation. Systematic development concerns the use of systems engineering in the development of a system, as opposed to the other stages in the lifecycle of the system. SysML, a diagrammatic modelling language useful in representing systems engineering analysis is discussed as means by which both the Rhino system development process, as well as the toolflow process may be succinctly articulated. Finally the Subsection on Traceability and Agility describes the nature of the logical links between the various phases of the systems analysis, and the important distinction between linearity and sequentiality with regards to these links, as is increasingly recognised in modern Systems Engineering methods. The theory discussed throughout this Section directly inform the Systematic Analysis conducted in Chapter 3.

The analysis of the literature pertaining to the topic of software defined radio will be comprised of a discussion of the software defined radio paradigm and hence the "ideal" software radio system, current frameworks for software defined radio utilising a reconfigurable computing technology, as well as synchronous dataflow modelling, an example of mathematical modelling technique of relevance to the field. This analysis will provide insight into the current state of field, both theoretically and practically. The software defined radio paradigm and the description of an idealised form of a software defined radio system constitutes the theoretical discussion on the current thinking within the field. This discussion also clearly defines the field within the broader classification of digital signal processing, relating its emergence to that of digital signal processing's earlier rise to prominence. It also highlights a critical concession made in most practical software defined radio systems, with regards to the use of a generic up and down conversion stages, and the implications of this concession.

The practical aspect of the discussion of software defined radio considers an example of a method for mathematically modelling digital signal processing systems, synchronous dataflow modelling, and its utility in describing software defined radio systems. Existing tools for creating software defined radio systems upon reconfigurable platforms are also presented. The popularity of reconfigurable platforms for software defined radio warrants this analysis, as this allows for several different types of tool sets, in terms of their varying degrees of abstraction to

be considered for one class of implementation platform. The discussion of the current state of the field of software defined radio directly informs the conceptualisation undertaken in Chapter 4, in development of a toolflow for the Rhino Project.

1.4.2 Chapter 3 - Systematic Analysis of a Software Defined Radio Toolflow as part of the Rhino System

Chapter 3 is concerned with the analysis of a software defined radio toolflow, within the analytical framework of the system engineering analysis performed for the Rhino System. This is done so as to leverage the insight provided by the analysis performed upon the broader Rhino System for the problem represented by a software defined radio toolflow.

The Chapter begins by considering the placement of the software defined radio within the Rhino System. The software defined radio toolflow is shown to be a subsystem within the Rhino System's physical architecture, primarily orientated around the system function of *configuration*. This *configuration* function is derived from the end-user needs expressed within the Problem Statement of this Chapter, hence providing a link the between the fundamental problem considered by this dissertation and the analysis being conducted.

Having established the relationship of the software defined radio toolflow to the Rhino System, as well as the problem statement to the analysis being conducted, the relevant analytical processes conducted upon the Rhino System are revisited and further developed. This is done by considering the phases of the Rhino development process phases of development: the System Architecture Elaboration and Requirement Analysis, and deepening the analysis conducted upon those features relevant to the software defined radio toolflow and the *configuration* system function. While these are analytical activities within the Rhino Development Process, by focusing on that which is relevant, insight into the nature of the required toolflow is furthered in accordance with the broader development process.

A specification for a software defined radio toolflow is derived as a result of this analysis. This specification contains the key features and constraints of the software defined radio toolflow as required by the Rhino system, and hence the fundamental concepts of the toolflow developed. These features also provide a means by which the toolflow developed may be assessed.

1.4.3 Chapter 4 - Proposed Design of the Toolflow and Implications for Development Process

Chapter 4 describes the conceptualisation of, and a development plan for this proposed toolflow design in light of the analysis conducted in Chapter 3, as well as the current trends in software

defined radio tools as also outlined in the Literature Review.

The foundations of the proposed toolflow are described by first reviewing the requirements for the toolflow as derived from the toolflow specification in Chapter 3. This re-expression of the functional features and constraints of the software defined radio subsystem grouping as requirements isolates the development of the toolflow from the rest of the Rhino System, and allows for its development to be focused on as a project in its own right. This reformulation represents the end point of the analytical decomposition process, and the beginning of the inductive creation process as described in the System Engineering Development theory in Chapter 2.

The critical design issues are then discussed, as identified in the review of other tools for creating software defined radio systems also in Chapter 2. A toolflow proposal is then made, that traverses the spectrum from the formalised, abstraction of the end-user to the Rhino system via a series of interrelated models. It is this concept design that will be implemented and evaluated in Chapters 5,6 and 7.

Finally a path of development is outlined for this toolflow concept, utilising the popular Spiral developmental process model. As required by the Spiral model, the key components of the toolflow are identified, and then prototype implementations are proposed that implement these components, which are progressive realised in an increasing degree of sophistication. Chapters 5 and 6 describe the prototype implementations of the toolflow proposed in accordance with this development process.

1.4.4 Chapters 5 and 6 - Developmental Prototypes

With the proposed toolflow outlined, Chapters 5 and 6 describe two development prototypes used to implement the toolflow concept and its components as mapped out in the toolflow development plan in Chapter 4. These prototype implementations demonstrated the validity of the toolflow concept, that in turn allowed the objective of this dissertation to be evaluated. Both development prototypes implement common software defined radio applications as vehicles to direct the development of the proposed toolflow.

It is critical to note that these developmental prototypes stop short of what some would consider a fully operational toolflow, as the purpose of this dissertation is to address its thesis, that an abstract-defined software defined radio systems may be implemented upon practical systems. As described below this purpose is achieved by the two prototypes described, and as such it is not necessary to implement a toolflow that exceeds the core features identified[9].

Chapter 5 covers a practical implementation of a Blackman Low Pass Filter, and primarily focuses on exploring candidate technologies that are used in implementing the toolflow proposal.

[9] Although possible directions for such work are discussed in the Conclusion

The technologies focused on were those concerned with the transitions between the levels of abstraction in the proposed toolflow. The various models of the filter implemented between these transitions are validated using accurate mathematical models, ensuring that the results from the various models are correct, and hence that the translation is occurring correctly. The technologies utilised are evaluated and discussed.

Chapter 6 describes a practical implementation of an algorithm for performing the Discrete Fourier Transform, especially optimised for reconfigurable computing. Beyond demonstrating the full functionality of the proposed toolflow concept, this demonstration shows the utility of the toolflow for prototyping novel software defined radio systems. Again mathematical models are used to validate the results from the various models created by toolflow concept. The performance of the implemented algorithm is assessed against that achieved by an optimised implementation, assessing the efficiency of the toolflow concept.

1.4.5 Chapter 7 - Evaluation and Conclusion

Chapter 7 concludes this dissertation, evaluating the proposed toolflow developed in terms of the systematic analysis which resulted in it, as well as the theoretical question that prompted this analysis. Then general reflective learning from undertaking this project will be considered, as well as future areas of work for the further development of a software defined radio toolflow, both generally and for the Rhino System's toolflow.

The most mature version of the proposed toolflow, as implemented for the second development prototype[10], will be evaluated in terms of the key components as identified in the Toolflow Specification in Chapters 3 and 4. Not only will it be demonstrated that the proposed toolflow has the features and constraints described in the specification, but also the insight gained with regards to those features through implementing the proposed toolflow will be described. From this evaluation, more general implications for the fields of systems engineering and software defined radio will be discussed, particularly in relation to the underlying hypothesis of this dissertation.

Extrapolating further than even the hypothesis of this dissertation, the lessons learned from the undertaking of this project will be discussed reflectively. The key insight gained is with regards to the manner in which engineering problems are addressed, differentiating between meeting a technical need and fundamentally addressing the root cause of that specific technical problem.

Finally future work is proposed in terms of the toolflow proposed, implemented and evaluated as part of this dissertation and the consideration of its thesis, as well as in the field of software defined radio. The proposals suggest the broadening and deepening of the toolflow "library", so

[10] as described in Chapter 6

that a greater variety and complexity of software defined radio systems might be implemented using it, hence creating a fully operational toolflow prototype. Furthermore, more broadly, recommendations are made with regards to the automation of the software defined toolflow libraries so as to open up the exploration of the electromagnetic spectrum to a wider group of engineers.

Chapter 2

Literature Review

The purpose of this literature review is to provide synthetic analysis of the theoretical foundations of this dissertation, so as to provide insight into not only the nature of the problem of a software defined radio toolflow that is being addressed, but also the means by which this problem has been understood, and the manner in which it shall be addressed. It does so by investigating the central theoretical concept to the project's methodology, the Systems Engineering Developmental Approach, as well as the the key technical area of interest, Software Defined Radio.

The section on Systems Engineering Development discusses the field of systems engineering broadly as it pertains to development, the use of the system description language[35], SysML and distinguishes between linearity and coherency in the Systems Engineering Process. These topics directly inform the analytical model used to develop the requirements for this project, as well as providing the means by which it may be assessed.

The discussion of Software Defined Radio (SDR) addresses the question of software defined radio as a paradigm, within in the broader trends of radio frequency technologies. It also describes existing frameworks for performing software radio upon a particular computational technology, providing analysis with regards to their commonalities when located upon a common platform. It concludes by considering a possible theoretical underpinning for modeling SDR systems, synchronous dataflow modeling, while acknowledging several alternative methods of modeling SDR systems.

2.1 Systems Engineering Development

As described in the previous chapter, Systems Engineering is an extremely broad field that conceivably could touch upon all areas of human technological-orientated endeavour. The section

below seeks to describe only those aspects of the field which are relevant to this project, as it is being applied as an analytical tool in order to guide the development of the software defined radio toolflow for the Rhino system.

Firstly, what is considered systematic conception and development, as outlined by current, internationally agreed standards[18, 10, 44] shall be presented, then SysML, the System Modeling Language, as relevant in documenting systematic development shall be introduced. Finally an issue critical to this project, traceability and the implications for the development timeline, shall be discussed.

2.1.1 Systematic Development

Systems Engineering is the discipline which seeks to inform all stages of the lifecycle of any complicated, technology-rich undertaking. It is regarded as both mature and extremely functional, with an international governing body, the International Council on Systems Engineering (INCOSE), and a large volume of best practice and standards defined[18, 10, 53, 6, 44, 45], including a description language, the Systems Modeling Language (SysML)[53, 35] being promulgated through the Open Management Group[35].

Systems Engineering is often associated with the development of a grouping of entities for achieving a specific set of purposes or functions, although it is not limited to development [10, 45, 44]. This development process of deductive analysis and inductive development is encapsulated by the "Vee" diagram, presented in Figure 2.1. To begin the development process, it proscribes a process of translating the needs of the stakeholders[1] of the system under consideration to a set of requirements that those charged with implementing that system may aim to fulfill. This process is often described as translating the requirements of the system from the language of the users' to that of the system's engineers' [53]. As is the case with any translation, this process is non-trivial, and indeed a whole field of requirements engineering has arisen to facilitate this process. The result is not only a hierarchical list of requirements of various types, but also a clear sense of the system boundary and the operational scenarios that the system will be employed within.

[1] Stakeholders are any people, systems or groups of people or systems that have a vested interest in the existence and functioning of a system. It should be noted that these interests are not necessarily aligned across all stakeholders - for example, the victims of a military weapons system are indeed stakeholders in it, who have a very different interest in its functioning than its operators or commissioners[10].

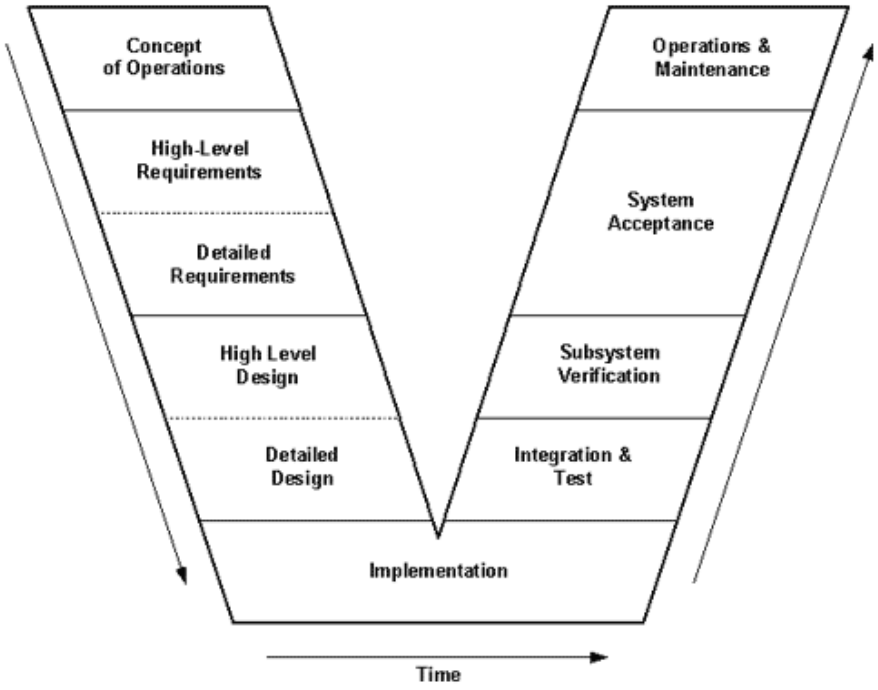

Figure 2.1: The Systems Engineering "Vee" diagram, describing systematic development[10]

Having engaged with the proposed system's stakeholders and context, and elicited a set of system requirements, the next process in systematic development is to create a system architecture that will satisfy the system requirements, that will be capable of performing in the operational scenarios envisioned while remaining within the system boundary. This is typically done by fully describing the behaviour(s) of the system in parallel with the structure of the elements of the systems[10]. The result is a deconstruction of the system under consideration into subsystems, with system requirements allocated to these subsystems[10, 53, 44, 45].

This deconstructive process is recursive, in that the subsystems derived by the system architecture elaboration may be considered fully self-contained systems in their own-right, and all of the analytical tools described above may be applied to them. The point at which the analysis is completed, and the subsystems are implemented is at the discretion of the development engineers[10, 53]. Often an iterative model is applied, where the analysis is completed to a certain depth, and then a partial implementation of the system or prototype is constructed to help guide further analysis[45, 8, 10].

2.1.2 The Use of SysML

SysML aims to replicate the descriptive role that the Unified Modelling Language (UML) plays in the Software Engineering domain more broadly[35, 53], allowing for the description of any

systems involving any combination of physical, virtual and human components[53]. As the Rhino project envisions a computational system comprising of a variety of digital hardware, as well as software and human elements, the project is an ideal candidate for being described using SysML.

SysML starts from the same premise as UML, that one diagram or one type can not encapsulate all of the details of the object under description[53]. Rather several different diagram types are used to reflect different aspects of such an object. SysML departs from UML as it moves to support systems in general, as opposed to merely object orientated software. In line with the generalisation of the concept of the software class, flows of items are supported, as well as continuous functions. SysML also introduces diagrammatic means for describing requirements and performance parameters, critical to fully mapping the lifecycles of a particular system[53].

Through the initial Rhino requirement analysis phase, wide use is made of the requirement diagram, as well as the use case and block diagram in order to describe not only the stakeholder's requirements, but also the stakeholders themselves, as well as operational concepts for the system and its boundary. In describing the architecture of the system, the block diagram and activity diagram are employed to describe the behaviour and structure of the system concept.

2.1.3 Traceability and Agility

The key concept that is inherent in the systems engineering development process, and is extremely explicit in SysML is that of traceability[10, 53]. From the capture of the stakeholders' needs to the subsystems implementation, the systems engineering process seeks to provide a clear mapping of the analytical and creative processes that have resulted in the system developed. This mapping is critical to making the development process transparent, comprehensible, and hence accessible to not only the developers and maintainers of the system, but also relevant stakeholders and any person that might take an interest.

This traceability may be interpreted as coherency or harmony between the different elements of the system[2], throughout its various lifecycles. By having the system, and of its' element being clearly traceable, the relationships and transitions between the analytical and inductive stages of the system are hence clearly defined. However it is important to note that this traceability characteristic imposes no structure on the nature of the development process, only that the process is clearly defined, and that all of the subprocesses within it are realisable. The implications of this distinction are considerable for a project such as Rhino, which is being developed in a unique academic context. Thus development of system elements[3] might occur concurrently

[2]The term interface is avoided in this instance, because of its very particular meaning in Systems theory
[3]And indeed the system concept itself

and largely independently. However the systems engineering approach predicts that if sufficient work is done to relate and describe the relationships between the analysis that gave rise to it, the broader system and its elements, then the system should integrate and function as effectively as if it had been developed by a single developer.

Such thinking is not revolutionary, and reflects a trend over the past decade towards the concept of Agility in Systems Engineering[50, 21]. Agile software development methods place importance on people, collaborative relationships and approach in developing software, and conceptualise the development process as containing short iterations that deliver value[4] to the stakeholders on a regular basis[1, 50]. This paradigm advocates a responsiveness and a focus on the outcomes of the development above an unnecessary adherence to formalised engineering methodologies[1, 50, 21].

A crucial distinction that needs to be made in the Systems Engineering domain with respect to Agile development is between Agile systems and Agile systems engineering[21], as these are frequently confused. The latter is a development process that responds to new information and learning that occurs, and delays system crystalisation decisions as long as possible, while the former is a system that may rapidly be reconfigured to respond to a change in the needs that inform it[21, 50].

The concept of traceability is critical in adopting such a reactive approach in any Systems Engineering context. By clearly and coherently linking together the assorted analytical and development activities in the creation of a system, any change that needs to occur may be properly mapped through and accounted for in all aspects of the system. Furthermore by facilitating the process of identifying and understanding the relationships between systems elements, traceability inherently enables Agile systems to be developed.

2.2 Software Defined Radio

The Rhino project seeks to provide a platform for research and training in software defined radio, immediately begging the questions as to what exactly software defined radio is, what are the implications of this new way of performing radio applications, and finally what work has, or is currently being done in this field. This section answers these questions by examining the software defined radio paradigm, the ideal conception of a software defined radio system and the existing practical frameworks for working within this area. It also considers a theoretical tool for the rigorous description of such systems, synchronous dataflow modeling.

[4] in the form of quality software

2.2.1 The Software Defined Radio Paradigm

Radio frequencies are defined as signals oscillating between 30 kHz and 300 GHz, that are generally implied to be electromagnetic in nature[46]. As these signals typically form the basis for all wireless electrical systems used to convey or sense information, the term has become synonymous for any technology that makes use of such signals. Broadly these technologies fall into two classes: sensing technologies, such as those employed in radio astronomy or Radar, or telecommunication systems, that make use of radio frequency signals to convey information.

Over the century that has seen the application of this technology, the manipulation of these signals has largely been performed using analogue electronics, i.e. directly interacting with, and affecting the electromagnetic signal that has been received, or that is to be transmitted. This approach has its merits: the signal does not have to be converted from some other form, typically an energy intensive process, and advances in general electronics drove improvement in the performance of signal processing electronics. However this approach was largely born out of practicality, there simply was no other widely available means by which information could be processed at such a high rate[46].

With the rise of portable digital electronics in the late 1970s and early 1980s, another means did become available, digital signal processing[46, 42]. This new approach converted electromagnetic signals into mathematical representations, and performed signal processing operations utilising digital computers. Rather than RF engineers using mathematical models for the operations being performed upon signals by analogue components, computational instructions that represent those operations could be applied directly[42]. Not only did this allow greater control over how the signal was being manipulated, it also allowed for greater flexibility, as changing the operations being performed required simply changing the instructions issued to the digital processing device[5] that was acting upon the signal[42].

With the proliferation of mobile cellular devices, the need for more sophisticated communication encryption techniques and finer control over sensing signals, this new digital means of interacting with signals has become common place[42]. Digital signal processors, dedicated digital computing chips were developed for purely performing signal processing operations[42]. In the mid-1990s, the field of Software Defined Radio emerged, as the distinct field that studied the application of digital computing technologies to radio frequency systems[17, 4]. While the trend of specialised computing hardware for these systems continued, the rapidly expanding capabilities of general purpose computing and economies of scale has shifted development focus into software systems which are implemented on the aforementioned general purpose hardware.

[5] of course the sophistication and the size of the instruction set of such a device was a practical constraint

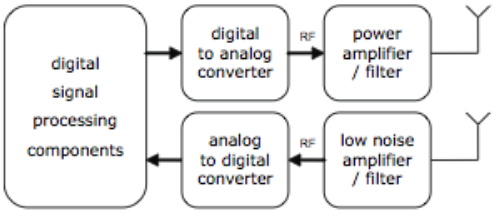

Figure 2.2: Ideal Software Defined Radio Architecture[31]

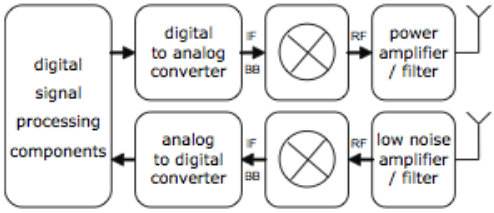

Figure 2.3: Practical Software Defined Radio Architecture[31]

2.2.2 The "Ideal" Software Radio System

Software Defined Radio is most definitely a paradigm that is coming of age[31, 17, 4], however from the outset it has been acknowledged that the conversion between electromagnetic RF signals and digital representations thereof are a key technical constraint in such systems.

Figure 2.2 presents a representation of the ideal SDR architecture, one in which the signal is converted to digital/analogue form directly, and minimal analogue circuitry is required, beyond receiving and transmitting such signals.

Figure 2.3 presents the more realistic case, acknowledging that both the conversion and digital processing technologies are not yet at the point at which a higher frequency signal[6] may be dealt with directly. And through the use of a single down-conversion stage[46], it is unnecessary to do so, as the signal may be converted to a lower, more manageable frequency signal for digitisation. However the practical implementation of the up-and-down conversion stages do constrain the portion of the electromagnetic spectrum that may be worked with. A further practical constraint that has to be considered is the bandwidth[7] of the signal under consideration[46, 40]. However all conventional systems operate within a defined bandwidth[46, 40], and provided that the digital and analogue conversion processes allow for the band required, this will not a constraint on the information that may be extracted from the signal. Furthermore by "selecting" only the desired bandwidth using analogue components[8], signal processing resources are optimised.

[6] It is difficult to succinctly describe the exact current capabilities, but broadly speaking the higher the conversion rate, the lower the resolution of the signal under consideration

[7] the width of the range of frequencies that may be converted simultaneously

[8] which are fairly sophisticated after a century of development

In essence, it is acknowledged that it is necessary that the developers of any software defined radio system engage with the applications under consideration, and hence choose hardware appropriately, or provide sufficient scope for end users to do so[31, 13, 17, 4].

2.2.3 Synchronous Data Flow Modeling

Synchronous Dataflow Modeling (SDF) is a theoretical means by which concurrent signal processing applications may be described abstractly as directed graphs[26, 25]. It is a special case of dataflow processing, as all the functions within the graph consume and produce data at rates that are known *a priori.* This maps well to conventional digital signal processing theory, as this is always the case, and is in fact a colliery to systems theory, which stipulates that a function maps a known number of inputs to a known number of outputs in a predictable fashion[46, 26]. There is implicit recognition of the suitability of the dataflow paradigm in industry and academia currently, as evidenced by the popularity of MathWorks Simulink and National Instruments LabVIEW. Both of these software programs employ dataflow representations as the primary means to engage with the processing of signals.

It has been argued that it is the natural method for representing digital signal processing systems [37, 14, 26], and that through this modeling, embedded signal processing systems such as software defined radios can be better characterised, understood, and hence improved[14]

The relevant facts, in terms of digital signal processing as applied in software defined radio:

- Signal Processing Systems may be modelled as interconnected dataflow graphs. Data samples traverse this network, having operations performed at processing nodes. Thus a whole software defined radio system may be described within as a single SDT graph.
- Synchronous Processing Elements or "Actors", which have varying degrees of granularity (i.e. a simple adder may be considered a processing element, as may a Fast Fourier Transform). The number of samples that are consumed and produced by the element during a single operation or "firing" are both known and constant. It is assumed that all processing elements work off a common clock, although they might run at different rates, which is an integer divisor of this fundamental frequency.
- Connecting the Actors are "Arcs" which represents the intercommunication and buffering that occurs between processing elements.
- A Topology Matrix may be constructed which represents the consumption and production of samples within the dataflow network, with the rows representing Arcs and the columns representing Nodes. This topology matrix is significant in that it describes the network in a form which allows for the body of linear algebra analysis to be performed upon it.

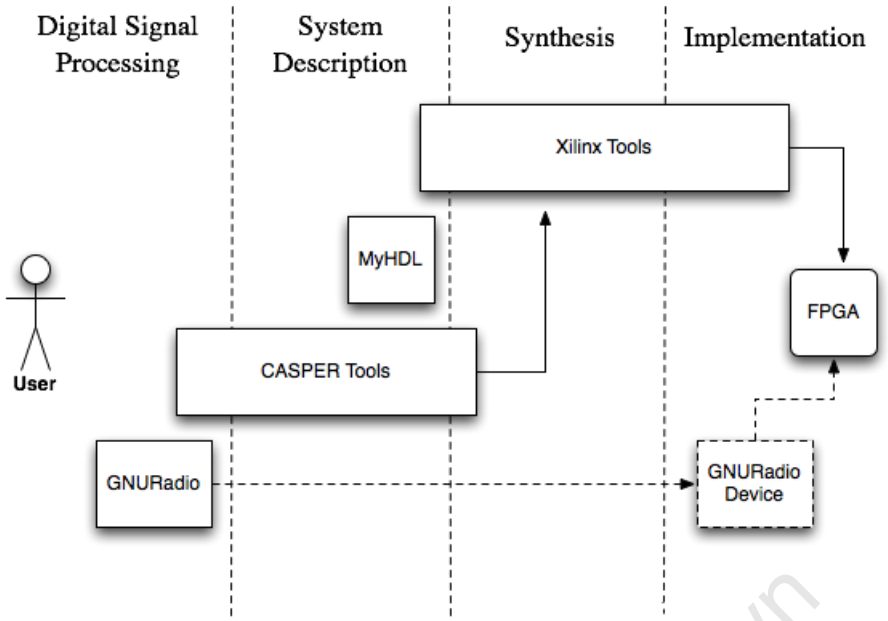

Figure 2.4: Overview of the spectrum between digital signal processing system descriptions and implementations on FPGAs

There are two primary benefits to the application of synchronous dataflow theory to software defined radio:

1. It is mathematically complete, with a wealth of supporting literature. This suggests that any toolflow or framework that is implemented in accordance with this theory can be considered to be so.

2. There are a variety of scheduling algorithms and analytical techniques associated with this modeling technique[25, 26, 27] which can aid in the formulation and implementation of software defined radio systems.

2.2.4 Existing Software Defined Radio Frameworks for Reconfigurable Computing

As Synchronous Dataflow Modeling is but one theoretical model for considering Software Defined Radio systems, there are also many different approaches to implementing software defined radio systems[48, 34, 29, 28, 31]. Those making use of reconfigurable technology have proven overwhelmingly popular, and hence warrant further investigation.

This section describes several relevant methodologies which make use of a particular class of such technologies, Field Programmable Gate Array (FPGA) devices, which span the spectrum

between the signal processing system[9] described by the user and the implementation on the programmable device, as is illustrated in figure 2.4. Hence, this spectrum spans from the abstract theoretical description of software defined radio system by the end user[10], in terms of the operations being performed, to the logic that would be specified on a field programmable gate array. Furthermore this survey is by no means exhaustive, but intends to give a "flavour" of the various frameworks available, within this particular subclass of implementation.

Firstly the Xilinx tool family is described. Xilinx provides a variety of software tools to facilitate programming of the FPGAs which they manufacture, which could be used to implement signal processing on their devices.

MyHDL, an Open Source library for the popular programming language Python is described. MyHDL provides a high level manner for hardware architectures to be described, as well as various tools for simulating and testing the architectures developed. This high level description can then be converted into more commonly supported hardware description languages (HDL). Similarly to the Xilinx tools, although MyHDL is not directly intended for digital signal processing, it may be used to describe hardware architectures which perform digital signal processing operations.

Then the Centre for Astronomy Signal Processing and Electronics Research (CASPER) toolflow is described, which is a large, easily accessible and wide ranging high performance signal processing toolsets which targets the ROACH, reconfigurable computing platform[34, 11]. Of all the approaches described, the CASPER tools are probably the most complete toolflow, in that it transitions from the end user to implementation in hardware[11], although the implementation in hardware is constrained to relatively few devices.

Finally, as an alternative approach to the models described above, the GNURadio project is described. While the GNURadio project makes use of FPGA devices, it constrains the user largely to a general purpose microprocessor computing environment in order to perform their signal processing.

Xilinx Tools

Xilinx Inc. is on of the largest manufacturers and vendors of reconfigurable computing. Xilinx provide the Integrated Software Environment (ISE) to facilitate the configuration of their FPGAs[57]. Within this environment, there are a variety of tools supporting the primary flow

[9] i.e. a signal processing system described in terms of digital signal processing operations, independent of implementation

[10] Although none use as rigorous a mathematical underpinning as synchronous dataflow modeling

[11] although it does comprise a significant subset of the Xilinx tools, as is demonstrated in the diagram above

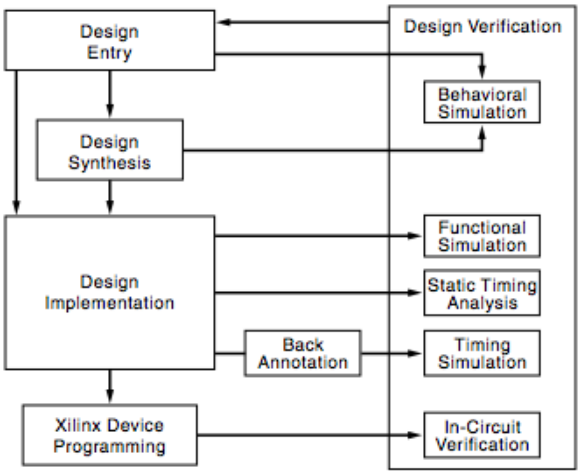

Figure 2.5: Xilinx Integrated Software Environment Tool Features [57]

of the design, i.e. design entry, synthesis and implementation, as well as device programming. Alongside this primary flow, there are a set of verification tools available.

A broad overview of the tools, in terms of the design flow[57]:

- Design Entry: Xilinx allows users to specify designs utilising two methods: text and graphical. The text interface requires the user to specify the design in VHDL or Verilog, the dominant hardware design languages. The graphical design tools make use of a simple block diagram interface, which can incorporate modules which have been described by the user in text The System Generator tool is an elaboration upon this, allowing for DSP systems to be specified within the Simulink environment from Mathworks. Furthermore the Core Generator software[54], which provides a standardised interface to large library of parameterisable, predefined "IP Cores", which are essentially optimised, modular implementations of a variety of operations, including digital signal processing tasks.

- Design Synthesis: The synthesis of the design is the process whereby the user design is prepared for implementation upon the target FPGA. A variety of tools are available to interact with this process which is largely done automatically. In particular there is support for the addition of user-defined constraints to this process, so as to ensure that the user specified system performs as expected. Furthermore this process is necessary for creating models for simulation of the user design.

- Design Implementation: This stage of the design tools perform the critical 'place and route' operations, which is the layout of the logic specified on the FPGA, as well as the interconnection routes between it. Again, Xilinx provides tools for the user to manually place modules, as well as specify routes.

- Device Programming: Support is provided for connecting to, and configuring a FPGA device from Xilinx with the now fully elaborated design. Further tools are available for assessing performance of the design on the chip itself.

Although the tools specified above are particular to Xilinx, similar tools are available from FPGA manufacturers. These tools are generally proprietary, and require payment.

MyHDL

MyHDL seeks to provide a higher level hardware description language than is currently available. It is implemented as an Open Source library of the popular programming language, Python. By choosing this path of implementation, it fully leverages the expressive power of the language, as well as the abundance of supporting libraries available[15].

MyHDL recognises that hardware may be modeled as a massively concurrent system, which can be described as many threads of execution which have a high degree of intercommunication. These are practically implemented using Python generator functions, which are standard programming functions with a non-fatal return behaviour, allowing for state to be maintained between calls to the function[15, 5]. Hardware modules are implemented using these generator functions, while special signal objects are used to provide interconnection and communication.

Simulation and conversion functionalities are available for the hardware systems described. The simulator allows for the behaviour of the systems described to be mapped out and analysed using standardised vector waveform files. The conversion feature allows for the system described to be converted into either VHDL or Verilog code, which can then be synthesised and implemented using vendor tools.

CASPER Toolflow

CASPER have produced a series of hardware and software design tools for creating signal processing equipment, which have been used to create a variety instruments for radio astronomy[11, 34]. The primary purpose of the CASPER hardware and tools is to reduce the time taken to produce instrumentation for Radio Astronomy.

This is achieved by providing a standardised development framework to work within, so as to facilitate collaborative development and reduce engineering design overhead. This is further encouraged by making as much of the software and hardware designs Open Source[36].

There are three distinct steps in the CASPER toolflow:

1. System specification and simulation using Mathworks Simulink (Block Diagram interface)

2. System simulation and conversion using Xilinx System Generator (Block Diagram to Hardware Description Code conversion)

3. Hardware description code to bitstream conversion using Embedded Development Kit (EDK)

The CASPER platforms and tools have been relatively successful in the radio astronomy community, greatly reducing the time and cost of developing and commissioning instruments[11]. However, problems with the toolflow as it is currently have been identified, namely:

- The low level of standardisation between libraries has inhibited the interoperability of code
- The tools and platform are restricted to Xilinx-produced FPGAs.
- The proprietary nature of much of the toolchain prevents the user community from customising and improving the performance and functionality of the tools.
- Furthermore the cost of the proprietary tools is prohibitively high, representing a sizable portion of development costs.

The GNURadio framework

GNURadio is a open source framework for creating software defined radio systems upon conventional processing units[52, 7].

It makes use of functional signal processing units or "Blocks" to perform processing on stream of data. These blocks are written in C++, one of the better performing languages on general purpose computers[7, 52], and are then "glued" together using Python[12].

As a framework, GNURadio is not explicitly targeted at reconfigurable computing, rather focused on promoting greater access to the electromagnetic spectrum[13]. This goal is interpreted to allow the casual end user to interact with, and process a signal conceivably from any part of spectrum. An implication of this goal is that it is expected that the user will primarily be making use of general purpose computing, thus not requiring much beyond the most widely available off the shelf computing equipment to perform software defined radio processing .

[12] An interface between C++ and Python is used, so that a high level Python may make use of, and interact with code written and compiled in C++

[13] This is extremely similar to the goal of the Rhino project

Users typically do not modify the FPGA processor which is the core of the Universal Software Radio Peripheral (USRP), which are a practically necessary part of such systems given the limitations of current computing[31]. Rather, certain common signal processing operations are performed upon it, and the user's defined DSP system thus occurs upon a standard, general purpose processor.

This is a alternative approach to those mentioned above in that the end user, who is interested in performing software defined radio operations, does not necessarily have to modify the FPGA utilised. This approaches chief merits are it's simplicity and low cost. The end user only has to purchase and connect up a USRP to start interacting and working with the electromagnetic spectrum. The detractor is of course that the end user can not fully leverage the full possible advantages of performing signal processing upon the FPGA for their particular desired system, unless they purchase and learn to use the vendor specific tools for configuring the FPGA and the interface created for GNU Radio and the USRP.

That being said, if an end user is prepared to do so, they have the fundamentals of a heterogeneous system for software defined radio available.

2.3 Application of the Literature Reviewed

This chapter described and analysed the current academic thinking in both the Software Defined Radio Domain, as well as Systems Engineering as applied to development. This Section will briefly describe the relevance of this academic thinking throughout the rest of this dissertation.

The application of Systems Engineering to development is the foundation of the theoretical approach adopted in this dissertation, and hence the nature and structure of the research methodology employed. The theory was addressed by considering the topics of systematic development, SysML and the role of traceability in development. The following chapter will comment upon the System Engineering performed upon the development of the Rhino system, and further it so as to better characterise the software toolflow. Upon the basis of this analysis, a toolflow is designed and developed as described in Chapters 4, 5 and 6. SysML is used throughout, allowing for succinct representation of the different aspect and levels of this analytical and inductive process. Finally tracebility is relevant to illustrate that seeming "scattershot" approach is in fact in line with Systems Engineering theory.

Software Defined Radio is both the context and the object of this dissertation, and so its definition as well as current trends needed to be considered while working in this domain. This was done by considering first the theoretical conception and definition of software defined radio. Then a technique for modeling such systems, synchronous dataflow modeling, was described,

as well as current frameworks for working within software defined radio upon reconfigurable computing platforms. While neither of these investigations were exhaustive discussions of the broad fields of software defined radio modeling and implementations, both provided insight into the nature and scope of the current work that has been done. Furthermore both describe technologies that are used in the practical investigations described in Chapters 4, 5 and 6.

Chapter 3

Systematic Analysis of a Software Defined Radio Toolflow

This chapter comments on the Systems Engineering process followed in the development of Rhino system, as relevant to the development of the software defined radio toolflow. The software defined toolflow is a subsystem within the Rhino System, and so this further analysis is fundamental to the toolflow's development. This activity is in service of the objective of this disseration because it provides a rigorous framework within which to engage with the academic question of a software defined radio toolflow.

The purpose of this analysis is to provide a complete, unambiguous description of the practical problem introduced in the first chapter. This complete description is often called a specification, and in an ideal world, it contains all of the performance and constraining requirements for a specific entity within the system[10] such as the software defined radio subsystem. However, in recent years it has been recognised that it often represents the starting point for further analysis and consideration of the role of that entity within the system, and indeed the nature of the system itself[50]. It is this analysis that that will be conducted below, so as to provide clarity on what exactly is required of the software defined radio toolflow at the iniation of its development.

This analysis begins by describing the Rhino systems engineering development process as a whole and the placement of the software defined radio toolflow within it. It then considers the relevant elements within the Requirements Analysis and System Design phases of that analytical process that affect the software defined radio subsystem, as identified by the lines of traceability. The software defined radio subsystem specification and constraints are then formulated, outlining exactly the nature and limitations of the project under consideration[10]. Not only will this specification form the basis for the development and implementation of the toolflow as described in Chapters 4, 5 and 6, but it will also form the basis of the assesment conducted in Chapter 7.

3.1 The Rhino System Engineering Process and Software Defined Radio Subsystem

The overarching objective of the Rhino system is to provide a reconfigurable computational platform for research and training in software defined radio for the South African tertiary education context[24]. Thus, from the outset, the project has sought to service a complex set of objectives for several groups of diverse stakeholders. The software defined radio toolflow is a critical part of trying to support the achievement of these objectives.

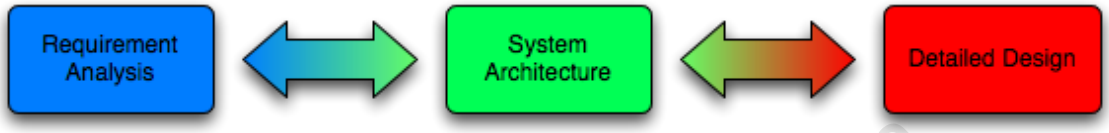

Figure 3.1: Rhino Project Development Process

The Rhino Development Process is based upon requirements-driven, responsive design approach commonly advocated[10, 18, 50], of which an overview is presented in Figure 3.1. It is important to note that the directional flow of the diagram only indicates coherency and traceability, as it is entirely possibly for multiple phases of development to occur simultaneously, and for relevant information within the various phases to be feed into each other[18, 10, 21]. And so the analysis considering the software defined radio toolflow conducted below features in the current iteration of the systems engineering document, as it is an extension and an enhancement of the work undertaken in the general Rhino Systems Engineering process. The full 4th revision of the report on the Systems Engineering of the Rhino Project may be found in Appendix A of this dissertation, that incorporates the full systems development process analysis, including the work conducted for this particular problem.

The Software Defined Radio toolflow represents a subsystem of the Rhino system, identified during the requirements analysis and systems design. This grouping was done for practical reasons, bringing together several related components for practical implementation, in essence forming the bridge between the System and Detailed design phases of the development process. However, as the detailed design[1] is undertaken the relevant system elements and requirements shall be revisited and reconsidered, providing further depth to the analysis conducted.

[1] in the form of this dissertation

3.2 Analysis of the Rhino development process

This analysis considers the rhino development process analysis undertaken, and traces the functions, components and ultimately the requirements allocated to the software defined radio toolflow subsystem. The system architecture phase considers the system elements that form the Software Defined Toolflow and the functional and physical architectural elements that gave rise to those elements. These elements are then traced back to the system requirements that gave rise to them. The crux of this analysis is the linking of the active end-user stakeholder group and the signal processing system[2], and the nature and features of this relationship.

3.2.1 Rhino System Architecture Design Phase

The Physical Architecture

The Software Defined Radio Toolflow subsystem has been identified as being primarily comprised of the development tools component, with the middleware and signal processing components providing support to the envisioned tools, and being the target of its functioning respectively. It is these system elements that encapsulate the translation and linking of the end-user's desired software defined radio system to the Rhino System's computational platform, and hence the usage of the system to perform software defined radio operations. The development tools components is primarily concerned with the capturing of the end-user's desired operations and translation into a form for implementation in the Signal Processing subsystem, however elements of the middleware that system that shall allow access to this subsystem, as well as the critical system interfaces. This relationship is represented by the interconnections in the physical architecture's internal block diagram, shown in Figure 3.2.

[2]Indeed, these two concepts represent either end of the toolflow

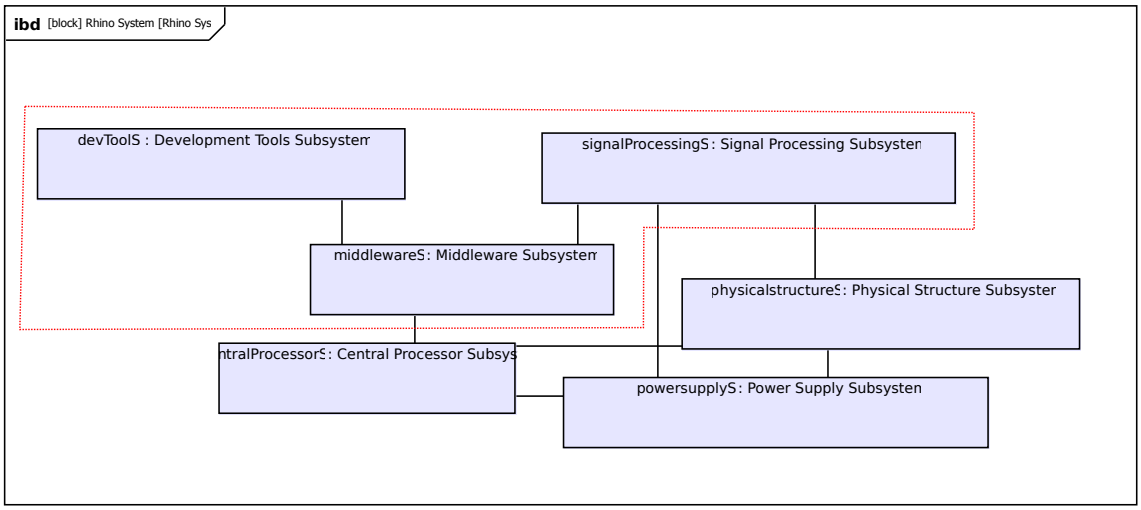

Figure 3.2: Interconnection relationships within the software defined radio subsystem grouping (outlined in red)

The Functional Architecture

The Rhino System function of configuration encapsulates the linking of the development end-user to the Rhino System, and so matches to the problem of this dissertation well. Figure 3.3 is a SysML Activity diagram that further decomposes the function further[53, 35]. It is recognised that the end-user will need to specify some sort of software defined radio topology[3], and that system needs to be validated, indicating that the system is capable of implementing it. Upon the completion of this validation process, and provided it is successful, the desired software defined radio system should be implemented by the Rhino system. Figure 3.4 makes use of a SysML Sequence diagram to further explore the interaction between the system and the end-user, as part of the Configuration Operational Scenario. There are three distinct exchanges that would comprise the configuration activity: the initiation of the configuration, the specification of the topology and its validation, and finally the deployment of the desired system. Thus it is critical that the subsystems provide not only the means to accept and deploy a software radio topology, but that it also has the capability to do so in concert with the end-user, by validating the desired operations.

[3]Which is a Digital Signal Processing system, as argued in the literature review. The term *topology* is used to differentiate from the more general *system* term used in systems engineering

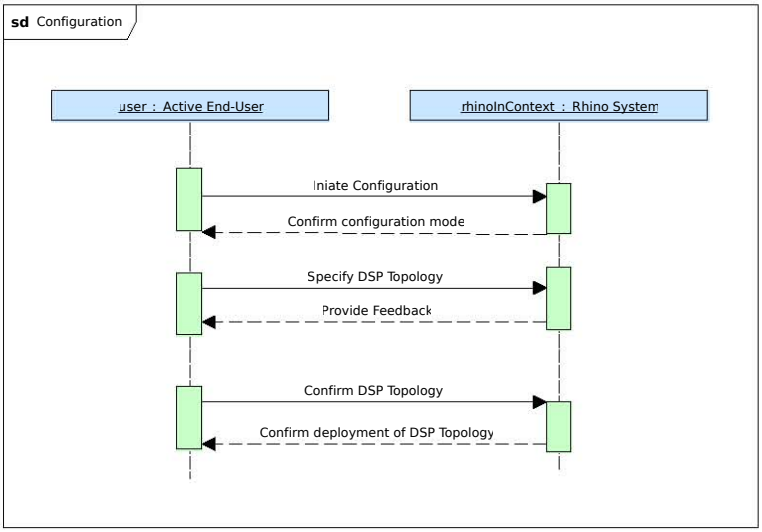

Figure 3.4: Sequence diagram further detailing the interaction that occurs between the end-user and the Rhino system during the configuration task

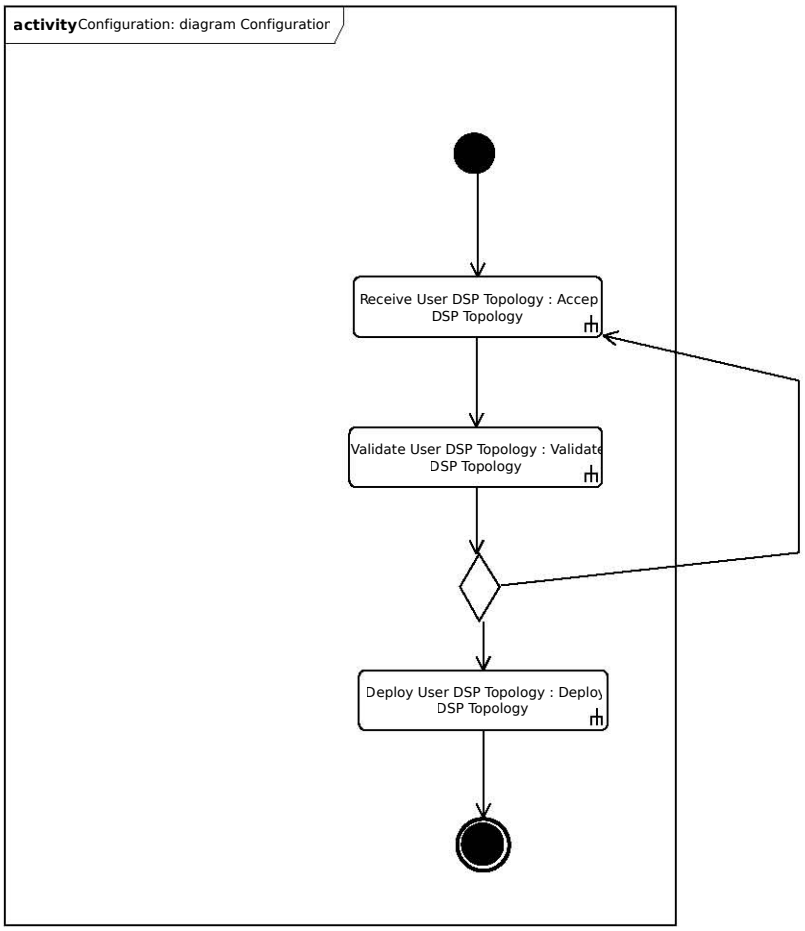

Figure 3.3: Configuration function of the Rhino System

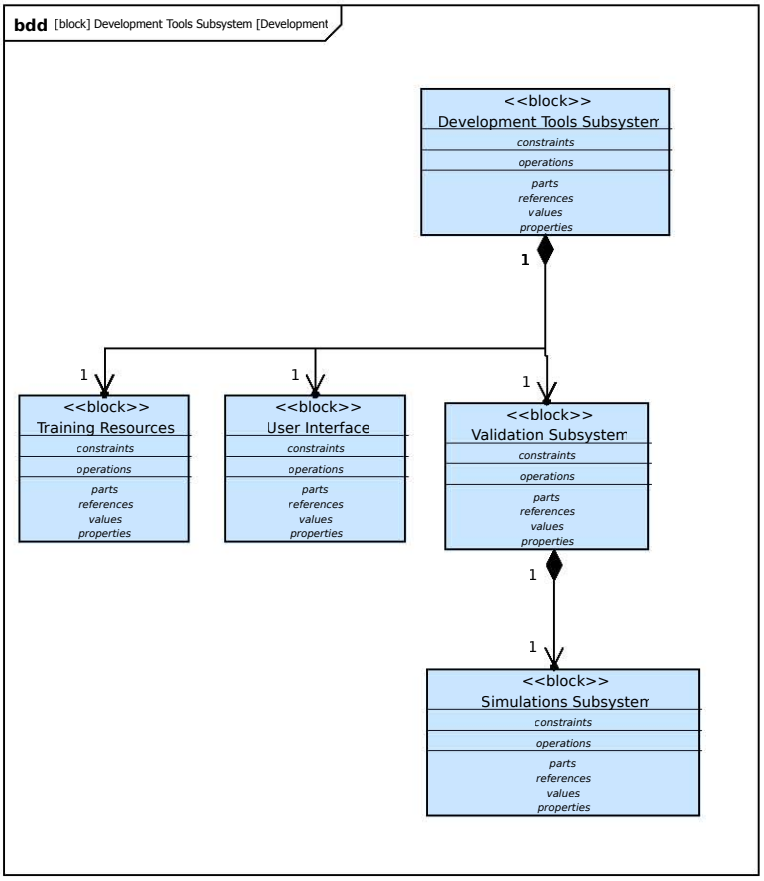

Figure 3.5: Further decomposition of the Development Tools Subsystem

The System Architecture

Thus, as Figure 3.5 describes, the development tools subsystem requires 3 key components:

1. A user interface that provides the means to solicit and accept interaction with the end user.

2. A validation subsystem that is capable of providing feedback that would provide the end-user of the nature of their specified system in the Rhino System. As indicated in Figure 3.4, simulation is seen as a key component of this, providing a representation of the behaviour of the specified system, which may be used in service of this validation.

3. Training resources are also required, that will better enable and facilitate the end-user performing the configuration task. The middleware subsystem is composed solely of the deployment mechanism, that will take the software defined radio topology received and verified by the development tools subsystem, and implements it in the Signal Processing Subsystem.

3.2.2 Rhino System Requirements Analysis Phase

In the analysis of the system function that is most relevant to the software defined radio toolflow, Configuration, it became clear that consideration of the system's End-User is critical, as it is the translation of their software defined radio topology into a form that functions within the signal processing subsystem that is the primary function of the toolflow. As a result, the requirements relevant to the end-user and its support, as well as the capabilities that are required in order for the desired software defined radio topologies they may wish to implement are considered below.

However, engineering is not only the enabling of capacity, it is also the pragmatic acknowledgement of the realities and limitations of the context of implementation. So those system requirements that constrain the software defined radio toolflow are also discussed. Figures 3.6, 3.7 and 3.8 all present decompositions of system requirements that pertain to the functions of software defined radio subsystem as derived in the analysis above[4]. Figures 3.6 and 3.7 present the decomposition of the critical digital signal processing requirement.

The Digital Signal Processing Input/Output System Requirement

As described in the Literature Review, software defined radio is a special case of digital signal processing, and it is thus a critical function of the Rhino System. This input/output requirement may be decomposed into two requirements: the types of processing that are required to be possible in order to be able to perform digital signal process, as well as the level of performance at which these tasks can be performed.

The level of performance required is difficult to evaluate, because of the extremely broad range of applications that are targeted by the Rhino System. The performance required by high-end research in the relevant domains were collected, and may be used to assess the scope of applications that may be implemented on the Rhino System[5].

Furthermore the types of operations that need to be supported in order to support software defined radio need to be considered. Beyond classifying these operations broadly into two categories, linear and non-linear, as done in Figure 3.7, it is clear that the scope required of the toolflow is enormous. There are literally thousands of signal processing operations that might in fact be required by the vast array of applications in the software defined radio application domain.

[4] details of the linking between the subsystems and the requirements may be found in Appendix A

[5] These performance requirements are presented in the Rhino Development Document in Appendix A

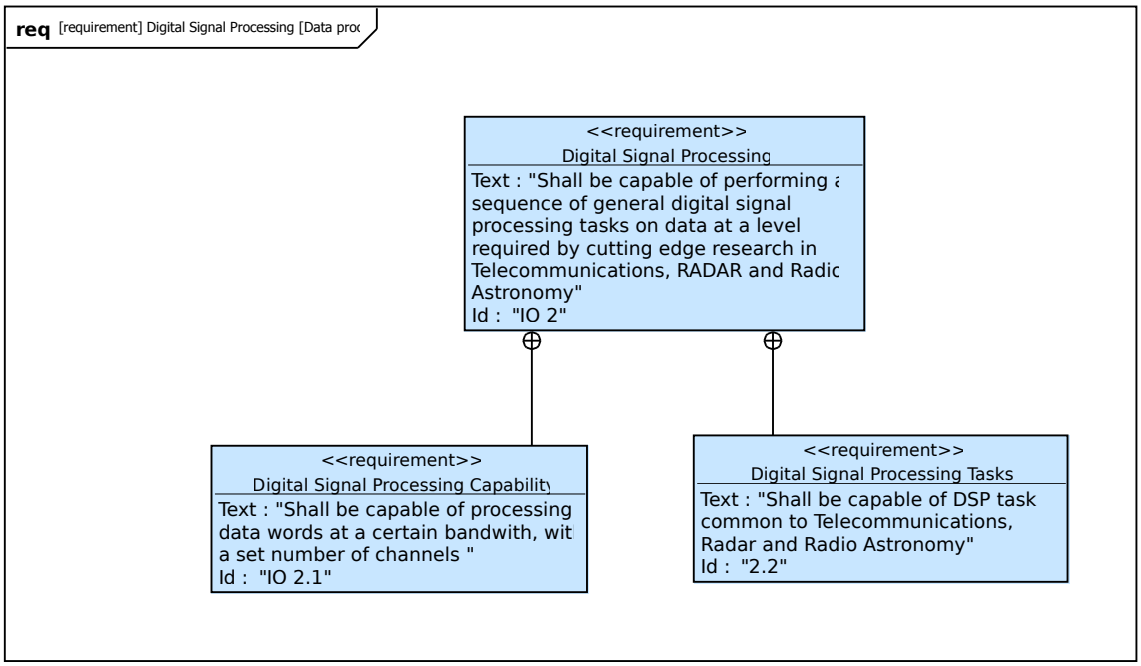

Figure 3.6: Decomposition of the Digital Signal Processing System Input/Output Requirement

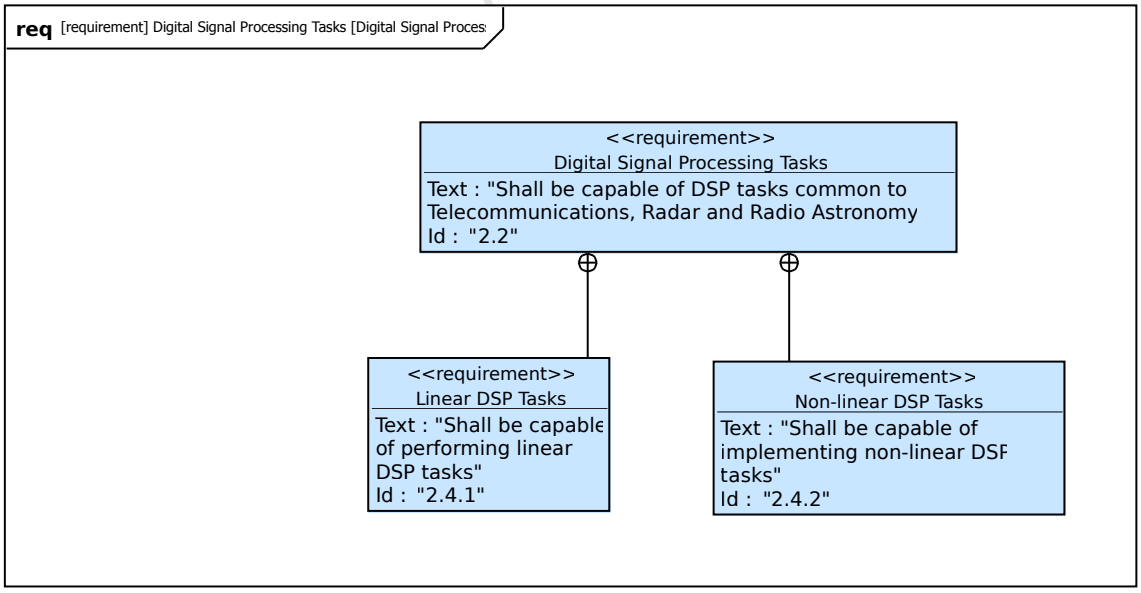

Figure 3.7: Further decomposition of the Digital Signal Processing Input/Output System Requirement

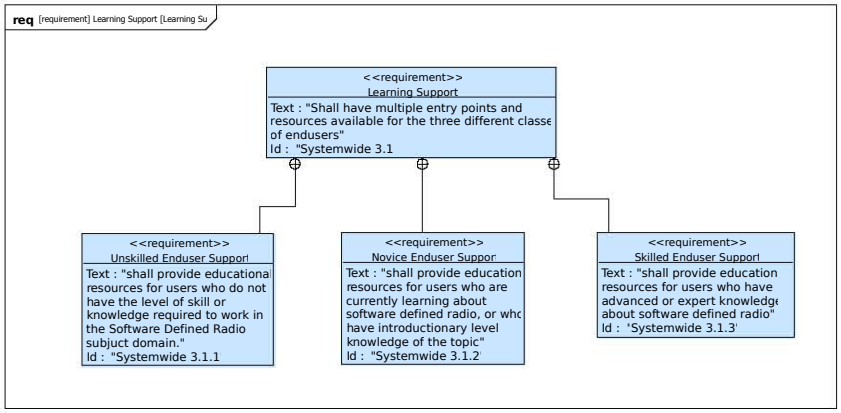

Figure 3.8: Subrequirements of Learning Support Systemwide System Requirement

The Learning Support Systemwide System Requirement

Another facet of the capabilities which the toolflow must provide is the nature of the end-users that shall be interacting with the toolflow. Figure 3.8 further decomposes the Learning Support Systemwide Requirement to show the various types of end-users that must be catered too. It becomes clear that there must be some notion of depth in the toolflow, allowing for all three categories of end-user to be accommodated. However the degree to which this broad support of the end-use is intrinsic to the toolflow, and to external resources provided has to be mediated.

The Interoperability Input/Output System Requirement

The requirements analysed above speak to the functionality or capability of the software defined radio subsystem grouping, however there are additional requirements that will limit or constrain the system formulated in response to these functions. Figure 3.9 decomposes the interoperability requirement, which suggests, as far as is possible, the toolflow should be capable of interacting with the hardware and software developed for the GNU Radio and Casper projects. The system requirement of reconfigurable computing technology is extremely relevant, as this fundamentally defines the end-point of the toolflow. The requirements around low cost and maintenance will further constrain the toolflow solution.

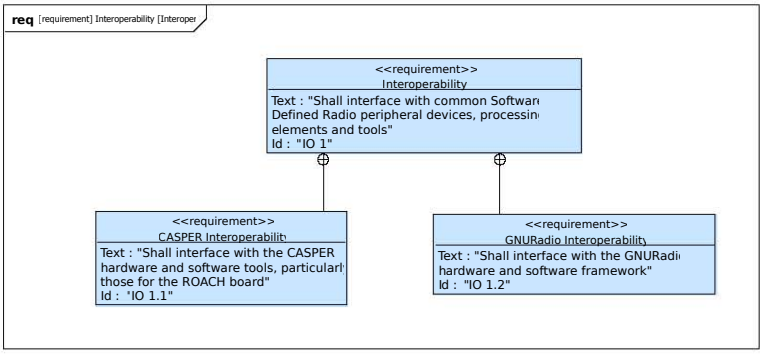

Figure 3.9: Decomposition of Interoperability Requirement

3.3 The Software Defined Radio Toolflow Specification

Having reviewed and furthered the analysis conducted by the Rhino development process, the specification, the features and constraints for a Software Defined Radio Toolflow, the subject of this project, may be described. The performance or capabilities required by the toolflow are described below as the Toolflow Features, broad groupings of functional requirements. The relevant limitations upon the toolflow are described as constraints. It is upon this basis of this specification that the toolflow shall be conceptualised and developed in Chapter 4, so as to perform the features while remaining within the boundaries of the constraints.

3.3.1 Toolflow Features

The table below describes that which the toolflow must achieve or the elements of its nature, as derived from the analysis conducted in the sections above. By performing these features, the subsystems identified as making up the software defined radio toolflow within the Rhino system will either be directly fulfilling the requirements of the stakeholders, or part thereof. Furthermore the degree of fulfillment of these requirements will reflect upon the success with which the objective of this dissertation has been achieved.

Feature	*Description*	Evaluation
Layered User Interface	The Toolflow will present an interface that allows for the unambiguous and consistent descriptions of the desired software defined radio topology within a broad range of level of detail, as well as any feedback information that it is necessary to be presented.	As this involves users, to a degree the evaluation of this feature is subjective. However by comparing what is implemented to established best practice, the measure of the interface may be found.
Levels of Validated Abstraction	The Toolflow will have varying levels between the abstraction of the end-user defined topology and the details of implementation in the Rhino System, with translation between each level validated.	If demonstration software defined radio applications are able to fully transition from abstract model to implementation, with verified validation performed.
Modularity	The Toolflow should theoretically be capable of being able to support any digital signal processing operation in a standardised manner.	This is self-evident, but the implemented Toolflow framework must demonstrate that it is possible to implement signal processing operations of varying degrees of granularity.

Table 3.1: The features of the proposed Software Defined Radio Toolflow for the Rhino

3.3.2 Constraints

The contraints identified below are the "realities" of the context of the design of the toolflow that must be considered when creating a toolflow with the features described above. These contraints are derived from other design decisions made as part of the rest of the Rhino system, as well as constraining requirements ascertained from the system stakeholders.

Use of Reconfigurable Computing

The use of reconfigurable computing, and the chosen implementation of the Xilinx Spartan 6 XLS150T[56] dramatically affects the nature of the toolflow, as it represents the end point of the translation of the end-user defined signal processing topology. It thus requires that the toolflow supports this translation process, and creates configurations that are compatible with this FPGA chip.

Use of the Borph Operating System

The use of the Borph Operation System in the command and control and deployment of the system affects the nature of the interfaces that the toolflow must traverse in order to deploy to the user-defined topology to the system.

Casper and GNURadio interoperability

A further constraint on the technology choices, the toolflow implemented should be done in such a manner that it could interface and interact with the Casper and GNU Radio libraries as much as is possible.

Openness

The requirements for low cost and support of varying levels of interaction user suggest to the point of necessitate that the toolflow makes use of as open technologies as possible in its implementation.

3.4 Systematic Analysis within the Toolflow Development Process

This chapter applied the theory of systematic development developed in Chapter 2, drawing upon the techniques and methods of Systems Engineering to rigorously characterise the practical problem under consideration, within the context of the Rhino System. This characterisation and analysis of the practical problem of a toolflow is one of the fundemental steps in the dissertation's research methodology in service of the the objective of showing that it is possible to implement software defined radio systems described in accordance with synchronous dataflow theory upon reconfigurable computing platforms.

Characterisation was achieved by first describing the Rhino System development process, and how this analysis process would utilise the analysis already done to develop a constrained specification for a software defined radio toolflow. This was achieved by iterating over the first two phases of analysis in the Rhino Development Model, the Design and Requirements phases. This analysis was then synthesised into a set of tooflow features and constraints, representing the nature of a software defined radio toolflow as part of the Rhino System, completely abstracted from any implementation.

Based upon this specification, the Chapter that follows will describe the conceptualisation and propose a design of such a toolflow, while also drawing upon the current work in developing these tools described in Chapter 2.

Chapter 4

Proposed Design and Development Process

This Chapter details the conceptualisation of a coherent set of tools for implementing software defined radio systems, intended for the Rhino System platform, commonly referred to as a toolflow. It also describes the development process that was undertaken to realise this toolflow to a point where it may be assessed meaningfully. This conceptualisation and development is also in service of an investigation of the implementation of software defined radio systems described in accordance with synchronous dataflow theory upon reconfigurable computing platforms.

The Chapter begins by considering the implications of the three features specified at the end of the previous chapter, in terms of the realisation of the toolflow. The two stage conceptualisation process is then introduced. Firstly, the key design issues are discussed, as derived from the specification as well as the software defined radio literature reviewed in Chapter 2, then the multi-step toolflow concept is proposed as the culmination of the analysis conducted thus far. Finally the development plan for realising and evaluating the proposed toolflow design is outlined. A key feature of this development plan is the degree to which the toolflow will be implemented for the purposes of this disseration, as required by its scope.

4.1 Toolflow Requirements

The previous chapter detailed the analysis that culminated in the specification of the critical features of the software defined toolflow for the Rhino System. These features may be reformulated in terms of system requirements, divorcing the development of the toolflow from the rest of the Rhino System.

As such the follow requirements have been derived for the software defined radio toolflow:

1. **Clear User Interface** - this characteristic suggests that the end-user needs to be provided with an interface to the toolflow that is not only easy-to-use, but also easy-to-learn. Furthermore this interface must be layered in some manner, so as to provide varying degrees or depth of access to details of the implementation.

2. **Validated Levels of Abstraction** - if an abstractly defined system is correct with regards to the theory informing it, it should work on the platform of implementation. These dual processes of translation from some description of the desired operations to an implementation on the Rhino System, as well as some manner of verification throughout this translation. This verification may take the form of *a priori* and *a posterori* measures, preemptive means that constrain the inputs to a desired subset, as well measures to provide feedback once the current level of abstraction has been realised.

3. **Modularity** - the toolflow must support generalised descriptions of signal processing operations, with the generalities firmly based in the nature of digital signal processing. It must be possible for any digital signal processing operation pertaining to radio frequencies to be implemented within the toolflow.

4.2 Rhino Toolflow Conceptualisation

This Section describes the software defined radio toolflow conceptualised and developed for the Rhino System in fulfillment of the specification developed above, considering the constraints also developed. The conceptualisation is comprised of two distinct phases: the key design issues, the critical questions which have to be answered in the toolflow design, and the toolflow concept itself.

4.2.1 Key Toolflow Design Issues

Several common challenges have been identified in creating a software defined toolflows for reconfigurable computing[31, 19, 29, 48]. A brief discussion of these issues, as is relevant to the development of such a toolflow follows. These common design issues span the spectrum from abstract description of software defined radio system to the details of implementation of such systems.

Description of user's SDR system

The interface between the end-user and the toolflow is the point at which they describe the software defined radio operations they wish to implement upon the platform. From the require-

ments derived around accessibility, the toolflow needs to cater for a broad range of abilities on the part of the end user in the programming of reconfigurable computers, in particular the FPGA chosen for the Rhino platform.

Thus it is logical for a toolflow to allow users to describe their intended operations as signal processing systems in accordance with a formalised description methodology for doing so, such as Synchronous Dataflow Theory. As an critical added advantage to doing so, all of the analytical tools developed for such systems may be applied to the system described, providing another means of validation for the toolflow[32, 26, 25]. Furthermore by creating a toolflow that is in accordance with synchronous dataflow theory, the underlying objective of this disseration may be achieved.

The role of simulations

Simulation is increasingly playing an important part of the verification mechanisms of the development of any system [39]. As a set of tools that is intended to aid in the development of complicated systems on a particular platform, it is critical that the simulations performed provide accurate feedback with respect to the particular stage of the development that is being performed. It is also crucial that accurate simulations are performed as soon as is possible, to reduce any wasted time or effort within the development process, as several stages in configuring reconfigurable computers are known for being time-consuming[57, 55].

Nature of the implemented system

As the toolflow will be used to implement a system abstractly defined by the end user, certain general principles must be adopted with regards to how those systems are implemented[31]. A key issue is the nature of the communication links between the various operational modules. These links must not only provide a standardised manner with which to link modules, but also the control information required to run the system.

Nature of arithmetic

A key concern in computing is the accuracy of the computations performed, essentially keeping track of the errors introduced by rounding off of numbers due to the practically limited amount of memory space available [20]. This manifests within signal processing in a variety of forms, relevant in software defined radio as noise[1]. Figures of merit such as quantisation noise and the signal-to-noise ratio are used to characterise this phenomenon [46].

[1]Extraneous data introduced into the system that impairs the processing of the desired data or signal

By default the software toolflow should construct the system that shall perform the end-user's defined software defined radio system in such a manner that the signal-to-noise ratio of the input signal is preserved. Furthermore the end-user should be provided the options to manage resource allocation in the system, allowing for the trade-off between noise performance and resource usage to be managed. The exact amount of noise[2] that may be tolerated is entirely dependent upon the end-user's requirement, hence the need for noise to be minimised generally, but allow for the end-user to determine this.

An example of how this might be quantified is in the use case of modern tracking radars. These radar system require 70 dBs of dynamic range in order to achieve the desired tracking resolution of 1m. As a result of this, all of the signal processing done requires at least 25 bits in order to accomodate the sensing range required.

4.2.2 Proposed Toolflow Concept

The proposed toolflow identifies several stages in traversing the spectrum between the end-user and Rhino system, as identified in the systematic analysis. It is proposed that each one of these stages incorporates some means of validation. The rationale is to ensure that an end-user's desired software defined radio system only proceeds further along the deployment process if there is the possibility (as adjudicated by the results of various verification mechanisms) that the design is capable of being implemented, and will perform as desired. Each of these stages are described in order, as presented in Figure 4.1

[2] due to the signal processing system

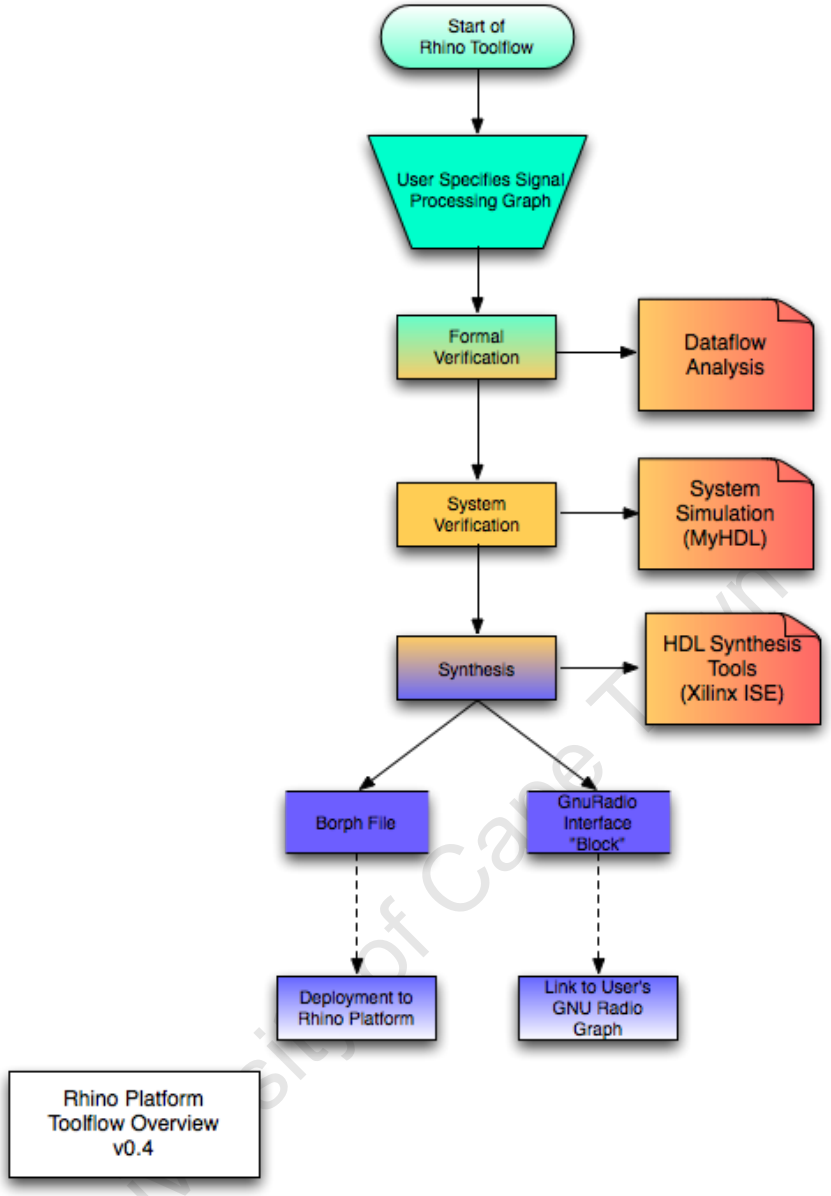

Figure 4.1: Overview of proposed tools for implementing software defined radio systems on the Rhino System

Formal Model

It is envisioned that the end-user will have conceptualised the desired software defined radio system that they would like to be implemented on the Rhino Platform as a model in accordance with a formal model description such as Synchronous Dataflow Theory. Thus it is proposed that the user specify the desired system in terms of a formal mathematical model of a signal processing system. This allows the user to take advantage of the wealth of tools available in order to accurately characterise the behaviour of the signal processing system[14] prior to entry into the toolflow. It should be noted that as much as this approach enables the easier description

of software defined radio systems, it does the limit the expression of problems that are outside of this paradigm.

This draws heavily on the approach adopted by GNU Radio [52, 7], in that the user creates a simple Python script utilising a library of Python programming objects in order to describe a signal processing graph with the desired operational parameters. This graph will consist of signal processing actors, performing both atomic and non-atomic signal processing operations[25, 26]. Thus there is a clear separation of the upper level description of the graph, and the lower level signal processing operations (which are described within the object library). This allows for more advanced users to directly access the digital signal processing if desired, but still provides the relative simpler notion of the system graph for the novice user.

This approach was adopted because of the widespread success and perceived accessibility of the GNU Radio project concept, which is usable by those with even a basic knowledge of signal processing and programming. Where this proposed toolflow differs is that the user specified model can be subjected to model verification utilising the linear algebraic techniques mentioned above[25, 26, 27]. This verification can determine:

1. That an acyclic graph exists for the specified system, and hence the system does not cause an "infinite loop", suggesting that data being processed will never leave the system.
2. The maximum number of samples that shall at any given point exist on an arc between two actors. This will ensure that there will not be an infinitely growing link between any two modules in the system.

System Model

From the formal model of the signal processing graph proposed by the user, a model of the system is proposed to be implemented in a simulatable system level language, such as MyHDL or SystemC[47, 15], verifying the proposed graph as a functional, implementable system[44]. This will provide the user with an opportunity to asses that the signal processing graph is actually providing the functionality desired, within the requisite level of arithmetic accuracy and performance by means of simulation.

For the case of MyHDL, each signal processing actor object available would have to have associated MyHDL generator functions that would provide the same functionality of the associated IP Core/HDL code which will eventually be implemented on the Rhino platform. The benefit of the greatly reduced overhead by simulating in a system level-language such as MyHDL versus other hardware description languages makes this step a useful tool to allow the user to get the design desired entirely correct, before engaging in computationally expensive hardware synthesis tasks.

Also at this level of the model, the interconnections between signal processing operations would be specified. Utilising a standard such as Wishbone [33] would satisfy the modularity and flexibility requirements for the toolflow framework, as well as provide a means for the control infrastructure, and so would also be advisable.

Hardware Implementation

Upon the formal and system elaboration and verification of the signal processing graph, as detailed above, these libraries will then either interact with the Core Generator Software[54] to produce the optimised hardware description code or make use of the high level hardware description language's built-in HDL converter to create the Hardware Description Language description which will actually perform the desired operations upon the Rhino platform. An advantage of this level of abstraction is that it lays the foundation for custom core creation and platform independence[3], as additional cores may be implemented utilising the standardised higher level description framework such as Python/MyHDL. This system will then be converted into Verilog/VHDL, with the relevant vendor-created/third party cores linked into place within the system.

Once the system is described in a common HDL, it can be simulated using one of the many simulators available, a prime candidate being the Xilinx iSim[55, 57]. This will allow for a final level of largely redundant bit level verification of the system before implementation on the platform.

Upon the completion of the optional bit-level simulation, two outputs from the toolflow are proposed:

- Bitstream File - Upon verification of the user specified signal processing graph on both the formal and system level, the system will be written to a hardware description and will then be placed and routed into a bitstream file targeted at the Rhino Platform using the Xilinx EDK described above[55, 57]. In addition to the desired signal processing graph, additional middleware elements will have to be specified such as any interfaces required or external memory controllers, referred to as Board Support Packages in the CASPER toolflow. This completed bitstream will then be wrapped in the required files to be understood by the BORPH operating system, running on the Rhino co-processor. This file will then be ready for deployment to the Rhino Processing Platform.

- Linking to GNU Radio - In additional to the final bit file being produced, the toolflow system could also produce a GNU Radio block [7], which will allow describe to the

[3] Within the reconfigurable computing domain, of course

GNU Radio framework how to interact with the signal processing graph implemented on the Rhino Processing Platform. This will allow for heterogeneous software defined radio processing networks to be created, using a combination of processing on the Rhino Platform and GNU Radio's signal processing blocks[19, 30].

4.3 Development Model

This section describes the development process of the proposed toolflow as outlined in the previous section, so as to realise the critical features of the proposed software defined radio toolflow. It it important to locate the toolflow development process within the broader Rhino development process. The development process described below fits into the Detailed Design phase of the Rhino Development process, as described in Chapter 3 as well as Appendix A. It draws upon the analysis of the System Requirements and Architecture phase to guide it, but the insights gained throughout this process are feed back into these phases. Thus the toolflow development process may be described as a major subprocess of the broader development of the Rhino System.

This description begins by addressing the core development process methodology, the Spiral Model, and how it is applied to the practical problem of this dissertation. The principal components of the Toolflow are then described as will be implemented for the purposes of this dissertation. This includes the degree of implementation, as well the technology used. Finally the developmental applications are described, the two applications of the toolflow that are developed using the prototype toolflow concepts. These applications are development prototypes, used as vehicles to develop the toolflow concept as described.

4.3.1 The Spiral Development Model applied to the Software Defined Radio Toolflow Concept

The developmental process methodology employed is based upon the popular Spiral Model [8], in which the stages or "arms" of development: analysis, design, implementation and verification are iterated over several times in increasing degrees of sophistication, as presented in Figure 4.2. As the project iterates over these arms, it increases in sophistication of implementation, hence creating a development process that allows for reliable, complicated functionality by building only upon reliable pre-existing functionality, while in accordance with the problem being addressed. This iterative, "learning" approach is recognised as being one of the precursers to the Agile methodologies discussed in the Literature Review[1, 50].

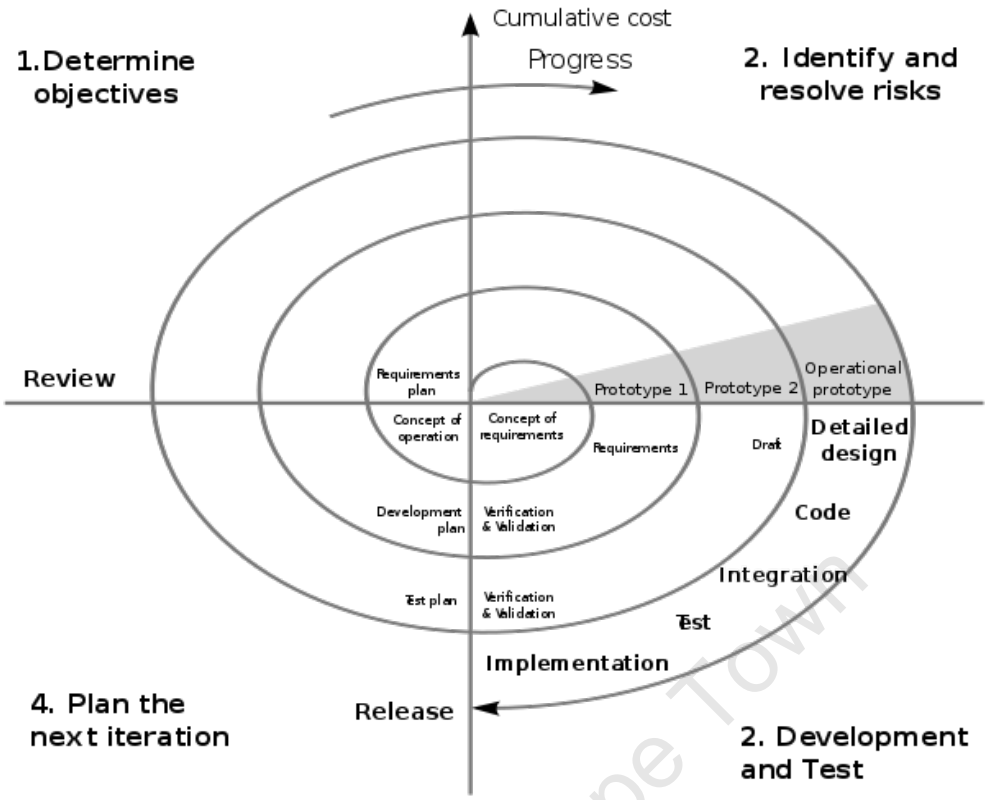

Figure 4.2: Overview of Boehm's Spiral Model[8]

In terms of applying the Spiral model to this dissertation, the first iteration around the centre of spiral, which is primarily concerned with conceptualising the requirements for the toolflow has been covered in Chapter 3 and this Chapter by drawing upon the Rhino Development Process. The second and third iterations around the Spiral will be conducted in the two chapters that follow this one, refining the concept of the software defined radio toolflow presented in this Chapter through two developmental prototypes of the toolflow, as used to implement software defined radio systems that are described using synchronous dataflow theory. Not only does this process of refinement improve the implemented toolflow, but it also improves understanding of the toolflow concept, and hence be of utility in addressing the objective of this dissertation.

It should be noted, as mentioned in the Introduction, this dissertation stops short of a full operational prototype toolflow. This is beyond both the theoretical and practical scope of this dissertation. The thesis under consideration may be adequately evaluated using the two development prototype, both in terms of functionality as well as performance. Practically, a full operational prototype would require not only a significant amount of time to develop, but also functionality of the rest of the Rhino System that is not yet in existence.

4.3.2 Toolflow Principal Components

As identified in the conceptualisation, the proposed toolflow has several stages in the translation from the end-user to the Rhino platform. Each of these stages may be translated concievably into a distinct tool or process that forms part of the larger toolflow.

Below is a brief description of the nature of these components, and the technology that was used:

- ***Formal Description***: A means for describing a software defined radio system, that is in accordance with a formal description methodology for such systems. Synchronous dataflow theory will be used for this dissertation.
- ***System Model***: A system model that is based upon the abstract system description, that is both simulatable and convertible to a common HDL. This will be implemented using the MyHDL hardware library[16] and the Python high level programming language for this dissertation.
- ***Hardware Implementation***: A means to implement the HDL description created from the system model, as well as simulation support for this implementation. This will be performed using the Xilinx ISE tools for the Spartan 6 FPGA used in the Rhino System[56, 57].
- ***Flow Coherency***: A viable mechanism for tying the above features together into a coherent flow. For the purposes of this disseration this is done manually by the end-user, this was done to allow greater development effort to be concentrate on the toolflow stages.
- ***Deployment Mechanisms***: Deployment mechanisms for various frameworks. Again, for the purposes of this disseration this is done manually so as to concentrate development effort on the other components.

4.3.3 Development Applications

Two pathfinder software defined applications were chosen to use as vehicles for the development of the toolflow prototypes during the Spiral development process, placing the development of the toolflow *in situ*. These applications help guide the development of the toolflow in a manner that simply implementing the major components will not, as it forces the development of the toolflow in terms of its overarching purpose, the creation of software defined radio applications.

The first was the implementation of a Blackman low pass FIR filter algorithm. The objective of this implementation was to create a simulatable model of the algorithm in MyHDL, and then

implement it upon the actual FPGA used in the Rhino Platform, the Xilinx Spartan 6[56]. The primary purpose of this implementation was to demonstrate that a MyHDL model of a DSP component could be created and implemented on the actual hardware, and that the system level simulations of the filter predicted the behaviour of the hardware filter implemented. Thus this prototype was exploratory, demonstrating the validity of the technology used. Thus the system was in accordance with synchronous dataflow theory, but this was not implemented in a manner that was particularly modular nor did it present the necessary layered interface.

The second pathfinder application was a scalable Fast Fourier Transform algorithm. In addition to being a more complex algorithm than the low pass filter, this application also incorporated the use of optimised "cores" from the FPGA vendor, Xilinx. Furthermore an object orientated approach was used in constructing the MyHDL system model, allowing for the high level system model envisioned in the toolflow features. Thus this second prototype demonstrates a prototype toolflow developed that is more in accordance with the features outlined in Chapter 3 and the principal components described above.

By completing both prototypes, the technology of the toolflow, as well as the concept toolflow itself shall be demonstrated to functional and valid. And in the evaluation of toolflow implemented, the objectives of this dissertation may be assessed.

4.4 Design and Development Process in the Broader Toolflow Context

The constrained specification developed in the previous chapter, based upon the systematic analysis of the Rhino Development Process were reformulated as requirements at the beginning of this chapter. By teasing apart the development of a software defined radio toolflow from the broader Rhino System, and focusing the practical component of this dissertation towards its theoretical question. A Toolflow concept was then proposed, firstly by considering the key design issues identified and then by introducing the concepts of seperate, but interconnected software defined radio models. Finally the "Spiral" development process for substantially realising this concept was outlined, identifying the key components of the toolflow and the two development prototype applications that would be used to drive the development of the toolflow for the purposes of this disseration.

Chapter 5 will detail the exploration and validation of much of the technology that would comprise the toolflow concept proposed, while implementing a simple Low Pass Filter. The critical technology investigated is that pertaining to the translation from the various models, as outlined in the toolflow concept in this chapter. Chapter 6 will focus on consolidating these technologies

into a coherent flow, while also characterising the performance of the toolflow again optimised implementations of the algorithm under consideration, the Fast Fourier Transform.

Chapter 5

First Development Prototype - The Blackman Low Pass Filter

The first application developed using the first prototype of the proposed toolflow was a relatively simple, yet widely used element in signal processing systems, a low pass filter. First the theory informing this particular filter's implementation and the motivation for selecting it, then the mathematical models constructed to provide a reference for the system and hardware implementations of the filter are discussed. This is followed by a discussion of these implementations, being the system and ensuing hardware implementations, as outlined in Chapter 4. The prototype discussion concludes with an evaluation of the primary goal of this protoype, validated levels of abstraction[1] while demonstrating the technologies used to make the transition from end-user model to the implementation.

5.1 The Blackman Finite Impulse Response Filter Algorithm

The filter algorithm used for this demonstration of the low pass filter was the Blackman finite impulse response (FIR) algorithm. This filter algorithm is classified as a windowed-sinc filter algorithm, in that it attempts to approximate as closely as possible the ideal frequency domain filter[2], of which the real time representation is a sinc function[3][42, 46]. This is done not by simply applying a "block" window to the sinc function, but rather a Blackman window, which smoothes out the ripple in the filter passband by tapering off the edge of the windowing function, as is demonstrated in Figure 5.1[42].

[1] as developed in Chapter 3 as one of the key toolflow requirements

[2] i.e. a step function in the frequency domain

[3] a sinc function is defined as $f(x) = \frac{\sin(x)}{x}$

The Blackman window class of filter was chosen because it represents a refinement of a standard digital signal processing operation, stopping this prototype from merely being a "toy problem" and actually concievably having some practical application.

The equation for the Blackman Window is given by

$$w[i] = 0.42 - \cos(\frac{2\pi i}{M}) + 0.08\cos(\frac{4\pi i}{M})$$

where i is the sample number of the window and M is the number of samples being used to construct the filter.

In terms of designing a filter using this window function and the sinc function, two parameters are necessary:

- The cut-off frequency desired, f_c. This is the frequency at which the output of the filter is half the amplitude of the input at that same frequency, relative to the gain of the filter. This frequency is expressed as a fraction of the sampling frequency of the data, f_S, which is in term described in terms of $^{\text{Samples}}/_{\text{Second}}$.
- The size of the filter window or order of the filter in samples, M. The order of filter is inversely proportional to the size of the transition bandwidth, given by the formula $B_t = \frac{4}{M}$. The transition bandwidth is a fraction of the Sampling Frequency.

Thus the equation representing the impulse response of the filter, and hence the kernel of the filter algorithm:

$$h[i] = K\frac{\sin(2\pi f_c(i-\frac{M}{2}))}{i-\frac{M}{2}}(0.42 - \cos(\frac{2\pi i}{M}) + 0.08\cos(\frac{4\pi i}{M}))\ i \neq \frac{M}{2}$$
$$h[i] = 2\pi f_c K\ i = \frac{M}{2}$$

Where K is the constant gain of the filter. The special case is given to avoid implementation issues around the apparent division by *0* in the sinc function. Algorithm 5.1 provides an overview of the filter algorithm in sequential instructions.

5.2 Floating and Fixed Point Arithmetic Models

A key consideration with regards to development is verification of the design[10, 18]. Verification implies that points of comparison must exist in whatever is being implemented, allowing for the implementation to be considered valid in reference to the methods of validation.

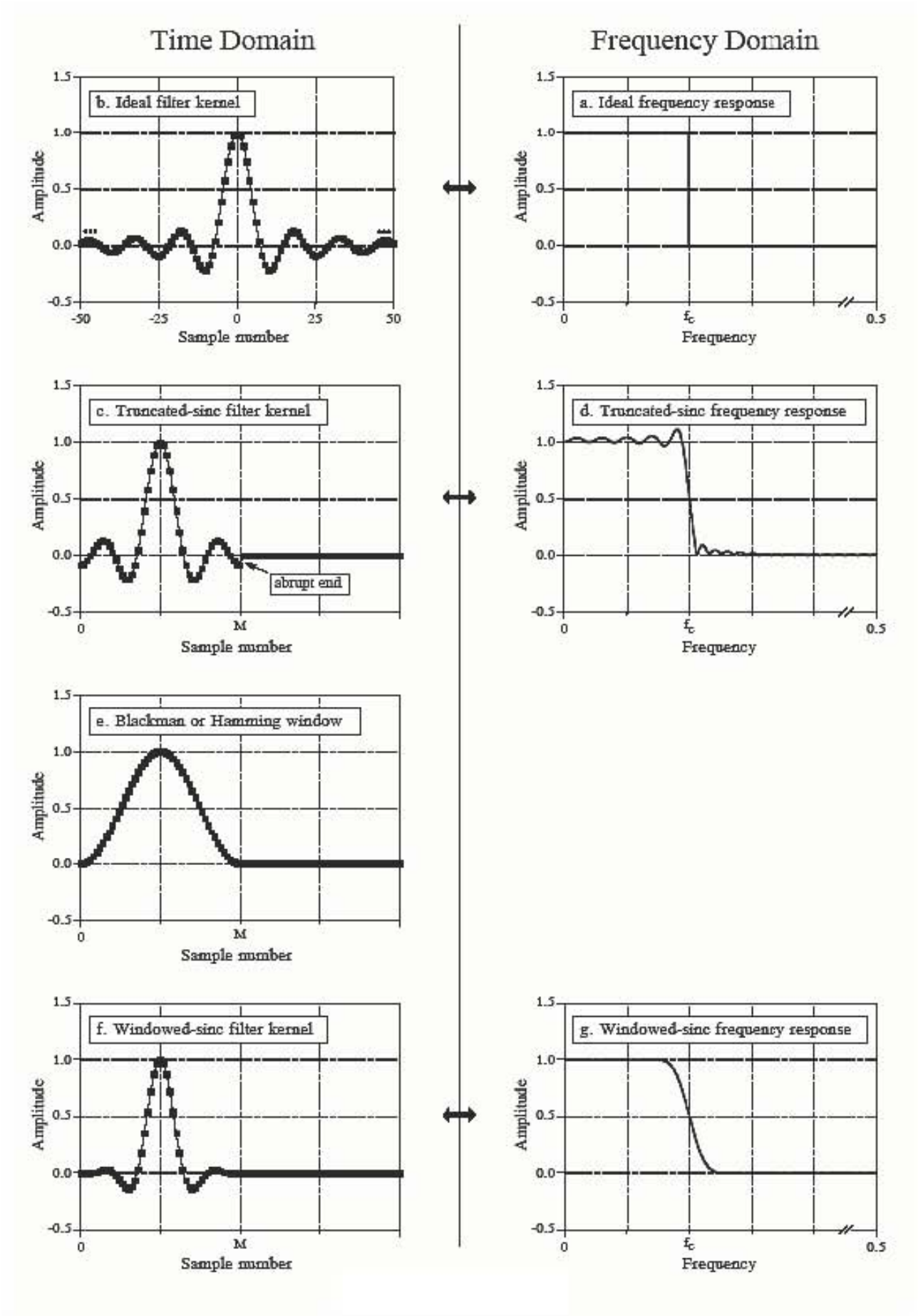

Figure 5.1: Windowed-Sinc Filter Theory[42]

Algorithm 5.1 Pythonic description of Low Pass Filter algorithm

```
input_dataset #Assuming input waveform is stored in input
   dataset array
output_dataset #Empty array for output waveform
cutoff_freq #Assuming cutoff frequency is a fraction between 0
   and 0.5
filter_coefficients #Array containing

#Generating Filter Coefficients
for i in range(0,filter_order):
        if (i != filter_order/2):
                x = i - filter_order/2
                filter_coefficients[i] = sin(2*pi*cutoff_freq*x
                   )/x * (0.42-0.5cos(2*pi*i/filter_order)-0.08
                   cos(4*pi*i/filter_order))
        else:
                filter_coefficients[i] = 2*pi*cutoff_freq

#Normalising the filter coefficients
gain = max(abs(filter_coefficients))
filter_coefficients = filter_coefficients/gain

#Iterating over input waveform
for i from filter_order to size(input_dataset):
        #Actual convolution operation
        for j from 0 to filter_order:
                output_value += input_dataset[i-j]*
                   filter_coefficients[j]

        output_dataset[i-filter_order] = output_value
```

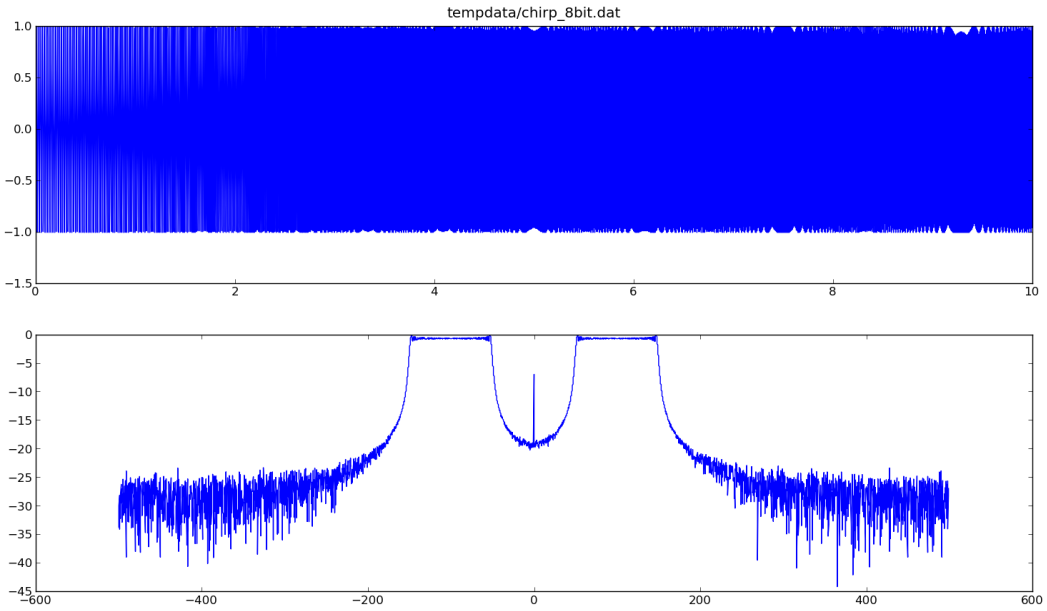

Figure 5.2: Example of the time and frequency domain of an input waveform used, in this case a linear chirp. The waveform was restricted to 8 bits, so note the quantisation noise.

5.2.1 Points of Comparison

These points of comparison may take many forms; however, in this case it takes the form of the mathematical results returned from the filtering operation. Hence a model needed to be created that could verify the output of the system model simulations of the low pass filter, so as to verify that it is fact returning valid results. Critically this model would have to account for the constrained bit arithmetic that would be employed within the system model. However, first this fixed point model would itself need to be verified itself. Hence a double precision floating point arithmetic model was created to first verify the fixed point model. A typical input signal into the mathematical models, as well as the two outputs for this signal are presented in Figures 5.2, 5.3 and 5.4

In order for a direct comparison between the results from fixed point and floating point model to be undertaken, the output from the two filter models had to be normalised. With the output values normalised, it was assumed that the double precision floating point implementation would be more accurate, and that the rounding error introduced in the fixed point model could be quantified by examining the mean square difference between corresponding values in both output datasets, i.e. $E_{system-filter} = \overline{(Y_{system-filter} - Y_{ideal-filter})^2}$, with $Y_{system-filter}$ and $Y_{ideal-filter}$ representing the output datasets of the fixed point system and floating point filter models respectively. To make this comparison meaningful, this error introduced by the fil-

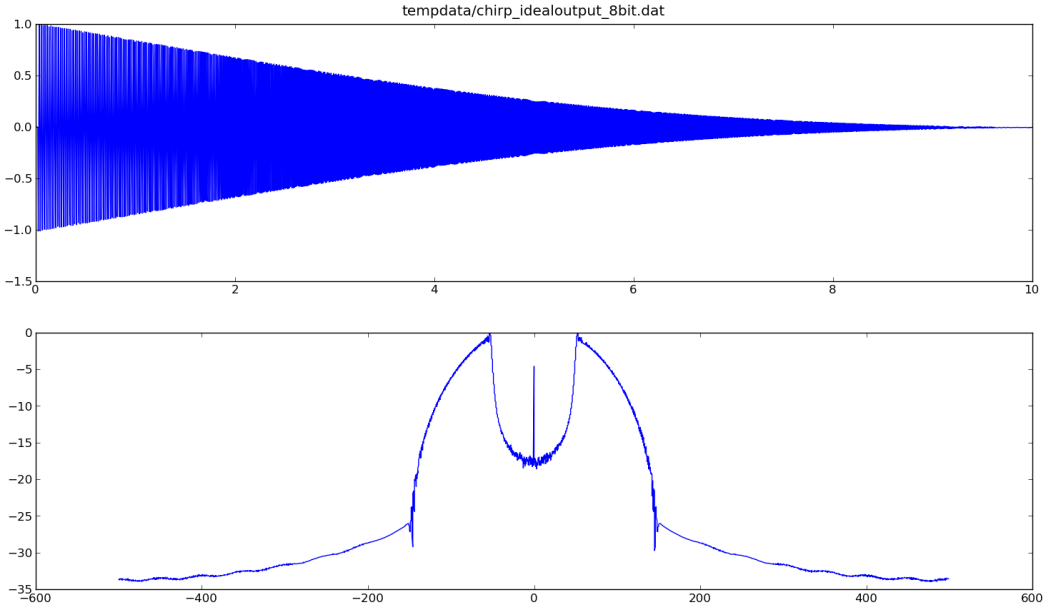

Figure 5.3: Example of the time and frequency domain of the output from the floating point low pass filter model

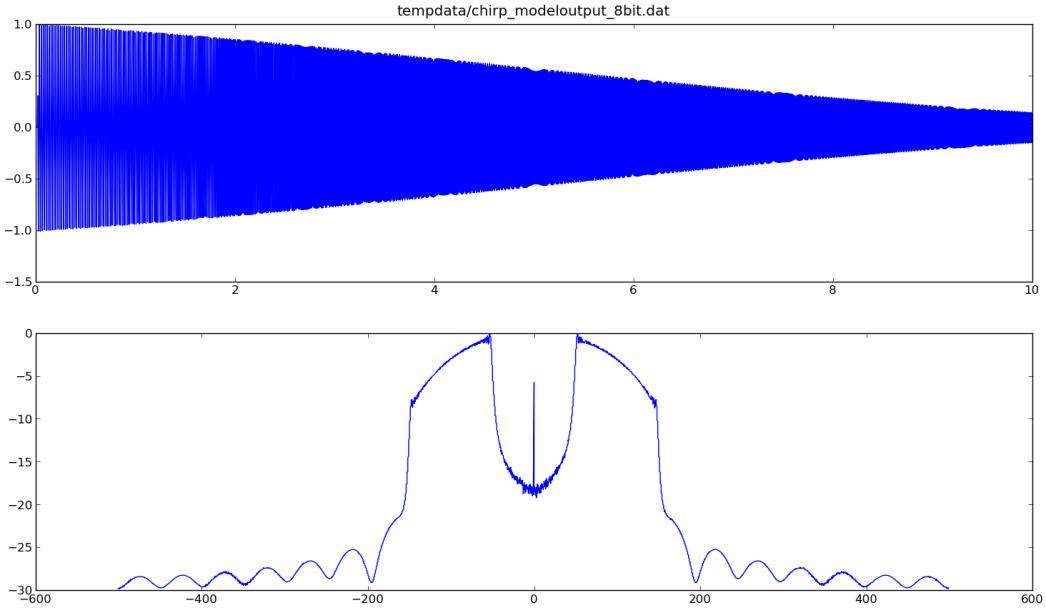

Figure 5.4: Example of the time and frequency domain of the output from a fixed point low pass filter model

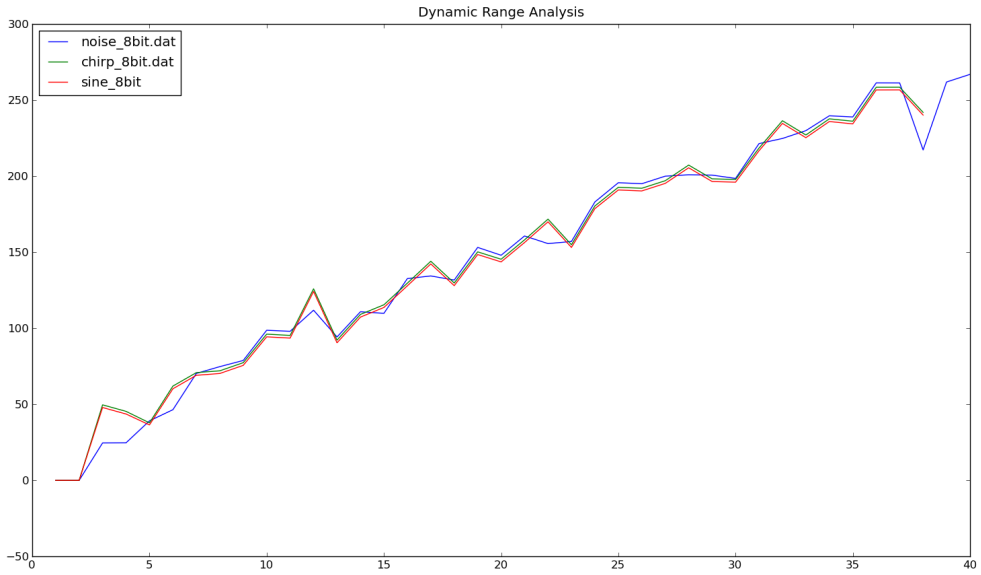

Figure 5.5: Plot of the effect of increasing dataword size used in filter coefficients

ter was compared to the mean signal power of the normalised output from the floating point filter, i.e. $P_{ideal-filter} = \overline{Y^2_{ideal-filter}}$, essentially calculating the dynamic range of the filter implementation[46, 42],

$$DR_{system-filter} = P_{ideal-filter}/E_{system-filter}$$

5.2.2 Experimental Verification

Experiments were performed using three waveforms: a sine wave, a linear chirp waveform and Gaussian noise. As is to be expected, the dynamic range was shown to be directly proportional to the bit size of the primary filter dataword, the filter coefficients, as is demonstrated in figures 5.5. As a result, it can be assumed that the fixed point arithmetic model implemented was valid, and could be used to meaningfully assess the system model created.

The full source code for the implementation of the arithmetic models, as well as the experimental code can be found in Appendix B. The arithmetic models were largely based upon the filter algorithm as described in Algorithm 5.1. There are two steps to calculation: Firstly the filter coefficients must be generated, based upon the patermeters of the desired filter. Then that filter "kernal" is used to perform the convoluation operation that yields the filtered values. The filter coefficients only have to be calculated once[4], but the convolution operation must be performed upon each value in the dataset that is being processed.

[4] a fact exploited later in the implementation of the system

This code is written in the Python programming language, in the modular, object orientated style which encourages clarity and reusability. The SciPy Scientific Computing libraries as well as the MyHDL were used to provide support for various Mathematical and fixed point arithmetic operations performed[2, 16].

5.3 System Model

Having investigated the theory underpinning the prototype system, as well as ensuring that there is a viable means to validate the results of the system, the next step was for the system model to be created. As envisioned in Chapter 4, this was done using the MyHDL library for the high level language, Python [16].

5.3.1 Systematising of the theoretical filter model

The key conceptual translation between the theoretical models and the system implementation was that the low pass filter comprised two primary elements, besides the input and output interfaces: a coefficient memory read operation[5] and a convolution operation. This suggests that the system should comprise two elements, a static coefficient memory and a convolution functional unit. The MyHDL documentation[16] suggests effective templates for both types of elements, and as such the core filter system was relatively simple to implement. Two global functions, each with a local generator function were used to represent each module.

The MyHDL signal mechanism was used to create a rudimentary input and output interface, along with a common clock. A slight complication was the use of the output serial data interface, which only allowed a limited number of output bits, and hence the output logic of the system was adjusted to account for this, incorporating an output FIFO, which takes in the output from the filter, and breaks it down into 8 bit data words. The full source code for the MyHDL filter implementation may be found in Appendix B.

5.3.2 Critical System Features

Two key requirements of the toolflow are modularity and validated abstraction. These are discussed below, in the context of the system model implemented for the first development prototype.

Python is modern, high level programming language that provides special support in its structure and syntax for iterative types and complex objects. This allowed the prototype filter to be

[5] the coefficients would have to be generated at some point, of course

written generally, so that it could in theory represent a low pass filter with any parameters. In addition to this flexibility manifesting in variable data word sizes, generalised functions were used to generate system elements such as filter coefficients, and read and write of the input and output datasets from files, supporting this flexibility of implementation. Thus it is evident that wider support for modular signal processing components may be implemented, although it was not done in this case in a manner that was in line with synchronous dataflow theory.

In order to evaluate the performance of the system implemented using MyHDL, a system of nestled test benched were created, which allowed for the functionality of the system to be simulated by the built-in MyHDL simulator. The first test bench was wrapped around the core low pass filter system (and is represented by figure 5.6, and was done to simplify the system. This test bench comprised a large input memory module (labeled Input Waveform Rom in figure 5.6), which contained the input waveform, as well as a counter, which provides the memory address to the input waveform memory. The second test bench provided the stimulus signals required for the system to run inside the simulation (and is represented in figure 5.7), being an initial reset signal, as well as the common clock. Additional randomised delay logic was applied to the data output to simulate the behaviour of the output serial link.

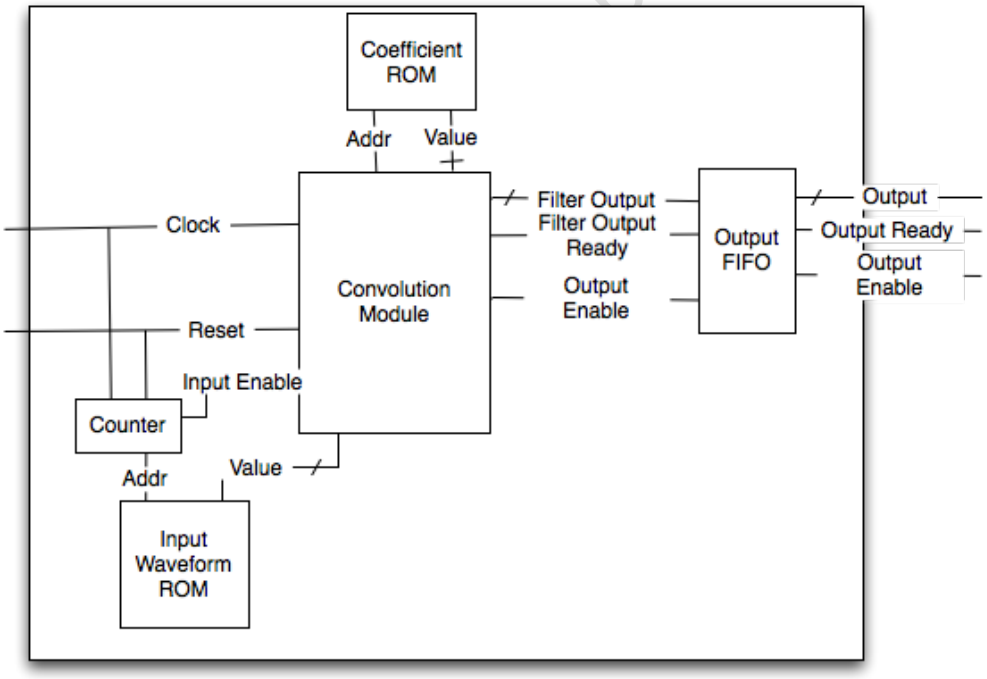

Figure 5.6: Block Diagram of core MyHDL Low Pass Filter System

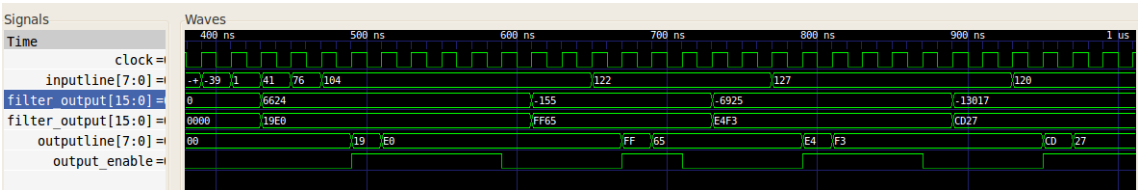

Figure 5.8: Plot of Output waveform in GTKWave.

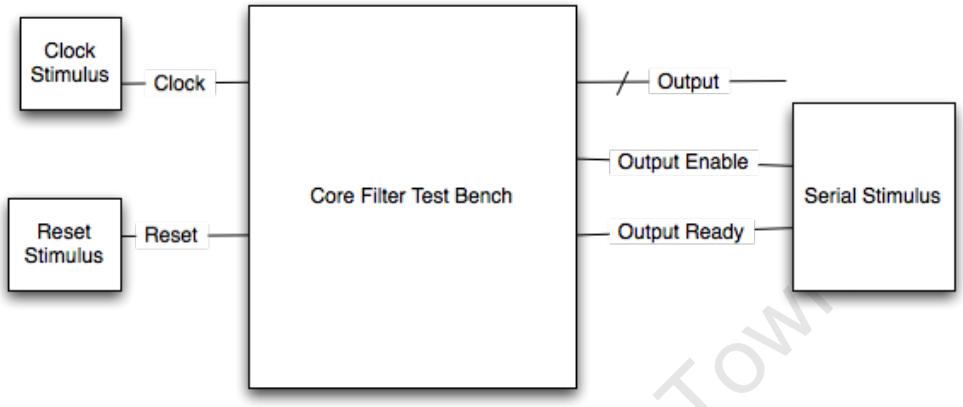

Figure 5.7: Block Diagram of Low Pass Filter System with Test Benches

The output from the MyHDL simulations of the system was found to match the output from the fixed point arithmetic model, and so it can be concluded that on a system level, the implementation was true to the planned prototype. Figure 5.8 shows a section of the output waveforms from the simulator. This is well exhibited by Figure 5.9, showing an output waveform from the System Level simulation, similar to those plotted in figures 5.2, 5.3 and 5.4.These results suggest a succesful implementation of validated abstraction, in that the higher level system model was able to be evaluated and simulated before being converted to lower level hardware descriptions.

5.4 Hardware Simulation and Implementation

Upon the successful implementation and testing of the System model, the MyHDL core test bench was converted to a common HDL, namely Verilog, and imported into a Xilinx ISE Project[57]. Within the Xilinx ISE, the inner test bench was incorporated into another test bench, in a similar manner to the MyHDL implementation of the outer test bench (see figure 5.7), with a simulated clock and reset signal applied. The key difference was that the system was synthesised, and prepared for implementation upon an actual FPGA, which took a considerable amount of time for each run of the simulation. This additional level of simulation was proven unnecessary, as it produced results identical to the MyHDL simulator and bit-constrained mathematical model. This does however verify the conversion from the MyHDL system model to

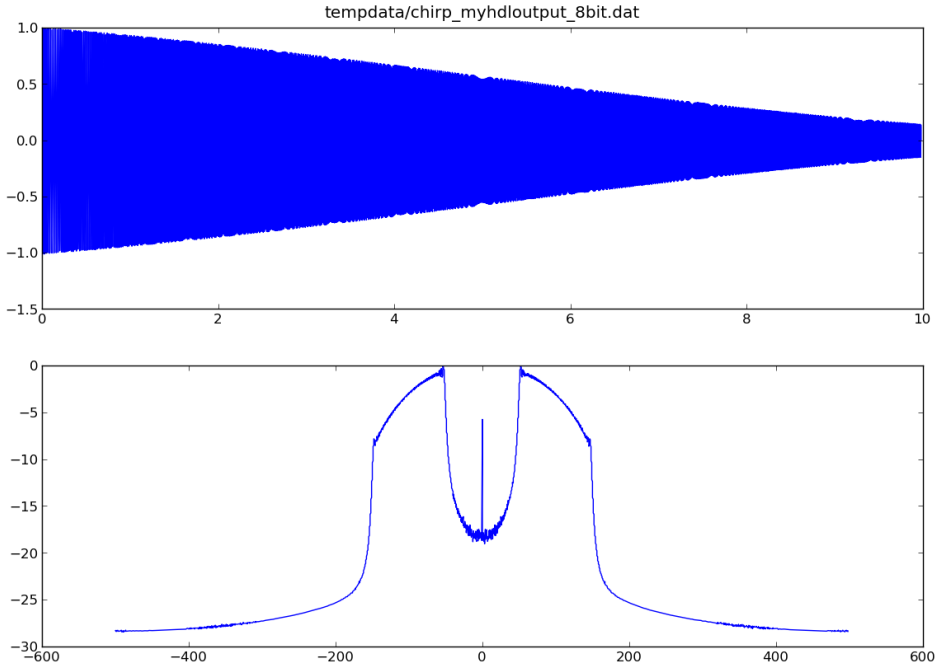

Figure 5.9: Example of output waveform from filter system model

Verilog. The abridged, converted code for a linear chirp implementation may be found in Appendix B, section B.1. Again, this lends credance to the claim that validated abstraction is possible.

The inner test bench was then incorporated into another ISE Project, wherein actual hardware components and pins were specified, as well as a serial communication core[57]. These were connected together using the high level, graphical block diagram interface, as shown in Figure 5.10. This system was then synthesised and implemented on the Xilinx SP605 Evaluation board. A USB serial port connection was used to collect the data from the Evaluation board. This data was compared to the results of the fixed point model, and was found to be identical (utilising the same mechanism as the system model), an example of this output is shown by Figure 5.11. Hence it can be concluded that the system model has been successfully implemented on the FPGA, and that this prototype had achieved its objective.

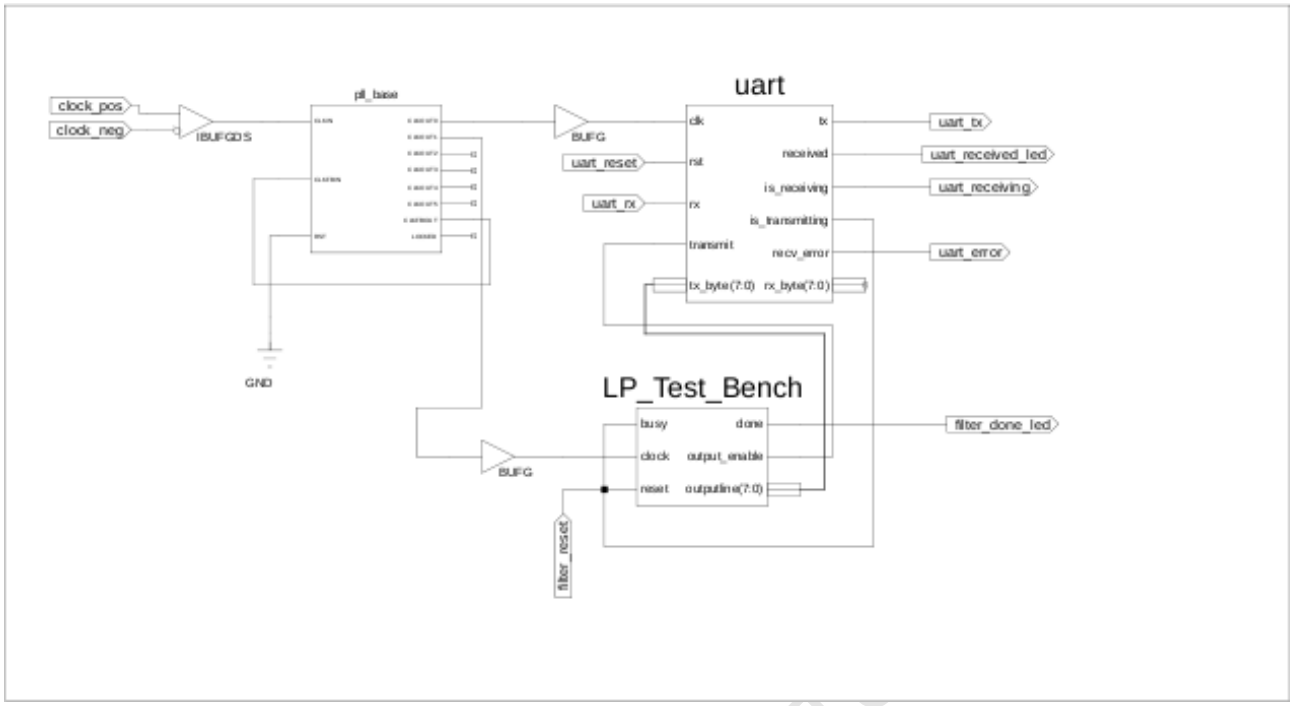

Figure 5.10: System Overview of ISE block diagram interface. The module labeled **LP_Test_Bench** is the Low Pass Filter Test bench, as described in Figure 5.6

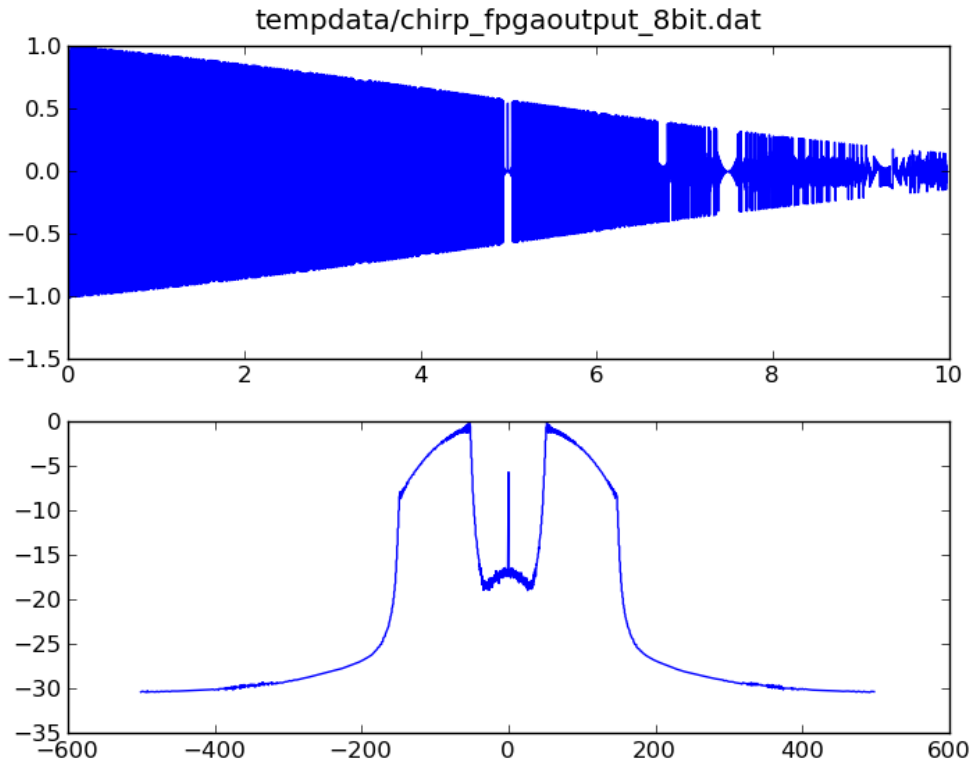

Figure 5.11: Plot of output data from FPGA Low Pass Filter Experiment

5.5 Tooflow Development Evaluation

This section presents analysis as to the state of development of the toolflow, after the successful implementation of the low pass filter prototype, in terms of the major components identified in Chapter 4, as well as the broader toolflow development. This represents the end of the first iteration around the Spiral Model, as described in Chapter 4. It also represents the sort of iterative "learning" conceptualised as part of the Agile approach[1].

This prototype of the toolflow demonstrated a flow, albeit labour-intensive, from an abstract, mathematically defined model[6] of a software defined radio system to a practical implementation upon a reconfigurable platform. While this doesn't prove that any abstractly defined system could be implemented, it does suggest the technological translations required by this process are functional. Further work will be performed in the chapter that follows in terms of characterising the performance of these practically implemented systems, as well as conforming to constraints around interoperability with existing tools for software defined radio upon reconfigurable computing platforms. Furthermore further work will be done to achieve the key features of modularity and layers of abstraction.

5.5.1 Formal Description

The high level system description was largely unimplemented at this stage of the toolflow development, in that the necessary signal processing theory was researched and implemented "manually". This was due to the undefined nature of the underlying system model at this stage in the toolflow development. It became clear that the fixed point arithmetic is a critical consideration in the design of the toolflow, as it directly relates to the performance of the implemented system, and hence the expected behaviour of the system on the part of the end-user.

5.5.2 System Model

A MyHDL system was successfully implemented, in that the desired software defined radio system was described and implemented using a high level programming language which approximated and predicted the behaviour of the system as expected. In the development of the system model, the need for standardised data and control interfaces between the modules implemented became apparent, as a considerable amount of time was spent "wiring" up the system. As systems which the toolflow implements grow in complexity, this need will become even more pressing. Furthermore, at this point in the process, no support was implemented for native, optimised "cores".

[6] that is in accordance wtih synchronous dataflow theory

5.5.3 Hardware Implementation

The implementation of the system model in the FPGA hardware was largely done using Graphical Xilinx ISE tools[57, 55]. As these tools are required to program the FPGA platform being targeted, this is not a major departure from the requirements of the toolflow. The conversion mechanism from MyHDL to Verilog was better understood, and found to be reliable.

5.5.4 Flow Coherency and Deployment Mechanism

As in the case of the high level system description, little effort was made with regards to the overarching toolflow coherency or the deployment mechanisms for the target platforms, as too many of the toolflow components were undefined. Transitions between the different phases throughout the prototype development were largely developer driven. However, it was recognised that by involving the developer in these transitions, greater insight into the system being developed could be created.

Chapter 6

Second Development Prototype - A Hybrid DFT Algorithm

The second development prototype implemented, in order to further the development of the Rhino toolflow to the cusp of an operational prototype, was that of a scalable algorithm for computing the Discrete Fourier Transform.

The first prototype application, the Blackman Low Pass Filter, verified that not only could a MyHDL-based system model of a Signal Processing operation be created, but that it could be simulated and implemented accurately upon a FPGA similar to that being used in the Rhino System. This development prototype will build on these findings, by demonstrating that a description of a software defined radio system that is in accordance with synchronous data-flow theory may be created and implemented upon the Rhino system. Furthermore, optimised, vendor-specific IP cores may be incorporated in this framework, demonstrating the modularity of the toolflow developed and its interoperability with exisiting tools in the reconfigurable computing domain.

As was the case with the filter prototype, firstly the algorithm under consideration will be described. Then the development of floating and fixed point arithmetic models for validation purposes will be discussed. Having developed the accurate means to verify the system model, the creation of the system model shall be described. Importantly, the approach adopted that allows for the abstract model sought in the specification will be described, as well as the module and system test bench development concepts that solidify the modularity of the Toolflow. The implementation of the system model developed onto the Rhino platform is then described. This implementation not only verifies the system model simulations, but also compares its performance to Xilinx's own LogiCore FFT IP core[58], which in turn demonstrates that a software defined radio application developed with the toolflow compares well to even extremely optimised implementations of this same application. The chapter concludes by describing the

pertinent facts from this prototype implementation.

6.1 The Hybrid Discrete Fourier Transform Algorithm

The algorithm considers the scalable, 1 dimensional case of the Cooley-Tukey algorithm for computing the dimensional Discrete Fourier Transform (DFT), popularly known as the Fast Fourier Transform (FFT). Scalability in the context of this algorithm has two dimensions: not only to the size of the input set of discrete samples, but also the resources employed in implementing the design, hence this algorithm is suited for use on reconfigurable computing platforms[22].

6.1.1 The Cooley-Tukey FFT Algorithm

First however, the mathematical description of the DFT and FFT:

Let $x[n]$ be a finite length, digital signal, sampled at a rate of f_s Hz. Thus there are N samples with a time period T_s between each sample. The Fourier Transform Coefficients for the discrete case is given by:

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi kn}{N}}\ k\varepsilon\mathbb{Z}$$

And by multiplying by a factor of N, the k Discrete Fourier Transform may be found (k may be related to frequency by $f_k = \frac{f_s}{N}k$), i.e.

$$X(k) = \sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi kn}{N}} = \sum_{n=0}^{N-1} x[n]W_N^{nk}\ k = 0,1,2,...,N-1$$

This algorithm is computational intensive, requiring N^2 complex multiplications, a particularly resource intensive operation. In order to reduce the number of these operations, the Coolely-Tukey algorithm exploits the cyclical value of the complex coefficient (or twiddle factor), i.e.

$$W_N^{\frac{N}{2}} = -1$$

And expresses the DFT formulation with $k = 2m$ and $k = 2m+1$, while making use of $W_N^2 = W_{\frac{N}{2}}$ so that

$$X(2m) = \sum_{n=0}^{\frac{N}{2}-1} (x[n] + x[n+\frac{N}{2}])W_{\frac{N}{2}}^{mn} \; m = 0,1,2,...,\frac{N}{2}-1$$

$$X(2m+1) = \sum_{n=0}^{\frac{N}{2}-1} (x[n] - x[n+\frac{N}{2}])W_N^n W_{\frac{N}{2}}^{mn} \; m = 0,1,2,...,\frac{N}{2}-1$$

This technique is applied recursively, and for the radix 2 case, until the N point DFT has been broken down into $\frac{N}{2}$ 2 point DFTs, hence greatly reducing the number of complex multiplications required to complete the calculation. This deconstruction technique is popularly called a butterfly or divide and conquer operation[22, 12].

Making use of this "division of labour" of the greater computational task, the FFT algorithm greatly reduces the number of complex multiplications required to compute the DFT, from N^2 to $Nlog_2(N)$, while modestly increasing the number of complex additions and subtractions[12]. However, crucially in this context, it also decreases the proportion of the algorithm that may be computed in parallel[22].

Hypothetically, the DFT could be computed in $log_2(N)+1$ sequential operations[1]. The radix-2 FFT algorithm would take $log_2(N)$ stages to deconstruct the data set down to 2 point DFTs, with at most 2 sequential operations per stage[2], resulting in $2log_2(N)$ sequential operations required for the full DFT to be calculated. The key limitation is caused by the bottlenecks introduced by the division stages, with the operations of further stages dependent upon the results of the previous stage.

6.1.2 The Hybrid FFT-DFT Algorithm

It is proposed that the Cooley-Tukey algorithm could be generalised, so that after a variable number of divide and conquer stages, a series of DFTs of size d may be computed upon the outputs from the decomposition, with d being a power of 2 factor of N[22]. Depending on the choice of d, the number of divide and conquer stages, and hence sequential operations may be limited, and performance optimised. The number of sequential operations required would be $2(log_2N) - log_2d + 1$. This algorithm may be described as a hybrid of the FFT and DFT algorithms, as it incorporates elements from both. Figure 6.1 contrasts this Hybrid DFT (HDFT) to the sequential requirements of the FFT and DFT, with $d = 16$.

[1] comprised of a single parallel set of complex multiplications and a parallel adder tree that takes $log_2(N)$ steps to traverse

[2] a parallel set of addition/subtraction operations and a set of parallel multiplications for the lower set

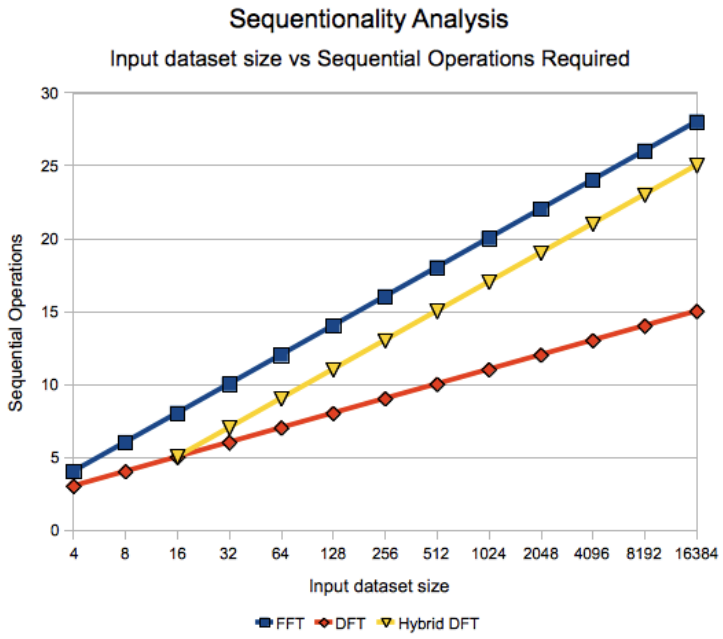

Figure 6.1: Sequential Operations plot, with the proposed Hybrid Algorithm for a direct DFT of size 16

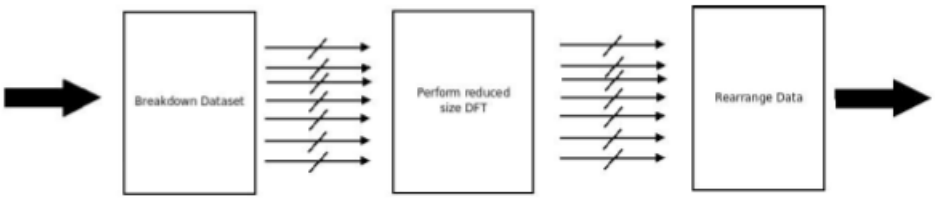

Figure 6.2: Simplistic Overview of the Proposed Algorithm's operation

However, obviously the greater the size of d, the greater the resources required to compute the algorithm, as a greater proportion of the calculation would be performed in parallel. Additional parameters for such an implementation would be the size of the input data set and the data words sizes for the input and output values, as well as the internal coefficients. Figure 6.2 presents a simplistic overview of the proposed HDFT algorithm. In addition to the decomposition operation required, there is obviously a reordering of the output data.

6.2 Floating and Fixed Point Arithmetic Validation Models

Similar to the Filter prototype developed in the previous Chapter, a fixed point arithmetic model of the algorithm under development was required so as to verify the output values from the system model and hardware implementation of the algorithm. This was implemented in MyHDL

and Python, based upon a previously developed floating point version of the algorithm[22]. The reference values for computing the error value made use of the FFTW library found within SciPy[2].

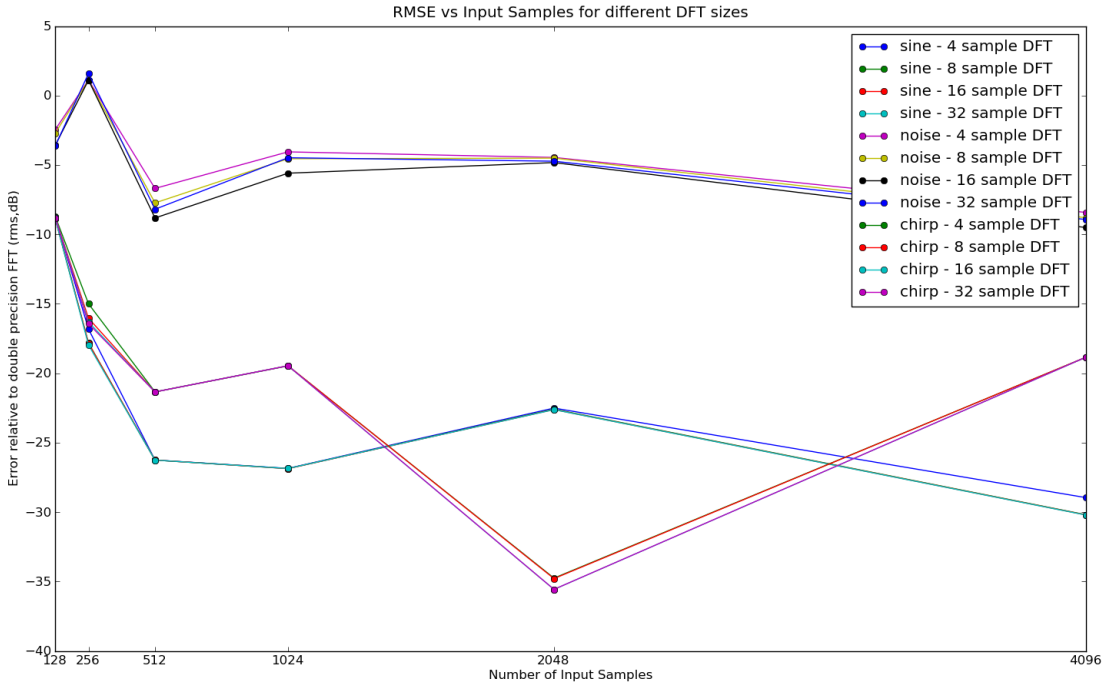

Figure 6.3: Plot presenting error performance of the implemented model

Figure 6.3 shows the results of the error characterisation experiments performed, performed over a range of data set and DFT sizes using the fixed point arithmetic model of the proposed algorithm. The range covered goes from 128 to 2048 input samples, making use of direct DFTs between 4 and 32 samples in size. 56 bit input data words were used (i.e. a 28 bit real and 28 bit imaginary values), along with 8 bit internal twiddle factors[3] for a Gaussian Noise, Linear Chirp and Sine test waveforms.

The graph shows that the error performance is fairly stable, albeit non-linear, with initially an improvement as the input data set size increases. However the consistency and stability of the error for the different sizes of DFT utilised suggests that the algorithm has a predictable error performance for a given DFT size.

A notable exception is the case of 256 Input Samples for the noise waveforms. In this case, the internal bit thresholds of the DFT calculation were exceeded, and the values produced were not

[3]This is in line with data word sizes used in the Xilinx FFT implementation used later to characterise the performance [58]

an even vaguely correct Fourier Transform of the input data. However set against the stability of the other noise waveform experiements, this appears to be an aberation, and was possibly a chance occurance. Considering that the point of this experiment was to prototype a novel Fourier Transform experiement, the results were decided to be sufficient for further use.

6.3 System Model

Having completed the mathematical characterisation of the algorithm, it is now necessary to describe it in terms of a software defined radio system in accordance with synchronous dataflow theory. First the critical elements that allow for this generalised description to occur must be introduced - a software defined radio system software library. This library was created making use of the MyHDL Python library, both in the internal structure of its members as well as the concept of module and system "test benches" used to develop library elements. After the toolflow library has been described, the process undertaken in creating a software defined radio system that implements the HDFT algorithm, making use of the software defined radio library, is discussed.

6.3.1 Critical System Description Features

In order for the Hybrid DFT algorithm to be described as a software defined radio system in accordance with synchronous dataflow theory, a library of programming objects had to be created that allowed for this description to occur. The structuring of this library is described, as well as the details of the internal structure. Furthermore the built-in verification mechanism, the "test-benches" are detailed.

The Prototype Library for the Rhino Software Defined Radio Toolflow

In order to fulfil the specification outlined at the end of Chapter 3, and to enable the creation of the Hybrid DFT algorithm, a software library written in Python, that makes use of the MyHDL library to describe operations was created. This library is made up of a set of classes, each representing different signal processing operations. Software Defined Radio systems are then comprised of groupings of instantiated object of these classes, linked together in a standardised manner. This concept is extremely close to the highly successful GNU Radio model[7, 41, 52], with the crucial difference being that hardware descriptions underly the processing nodes, as opposed to C++ in the GNU Radio case.

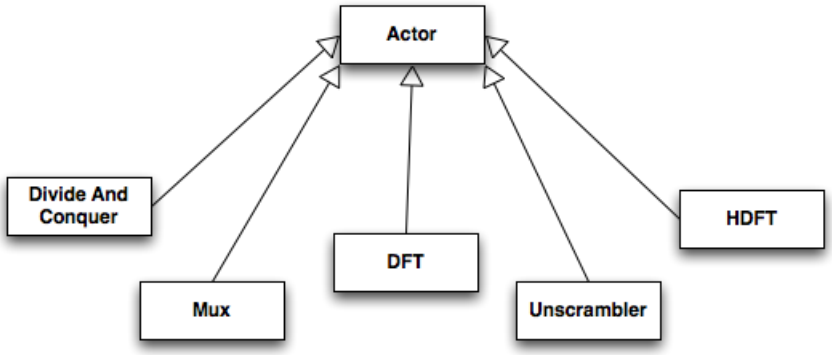

Figure 6.4: The inheritance structure of the proposed Rhino Software Defined Radio Toolflow Library for the case of the Hybrid DFT

Furthermore this toolflow library was created in accordance with synchronous dataflow theory. As a result at its core is a single base class, the Actor, that is a self-contained unit that is capable of accepting and transmitting data units or Samples via Arcs, groupings of data links that carry control information as well as the actual signal data that needs to be processed. This Actor class is also capable of performing a trivial signal processing operation, allowing a sample or group of samples to pass through it, provided the required number of input samples present are present on its input buffer. All other signal processing operations, represented as classes inherit this super class, and modify the operation performed as required, but still exhibit the same behaviour with regards to the input and output of data, as is shown in figure 6.4 using a UML class diagram. If additional input or output interfaces are required, they may be added, but the underlying structure behaviour of the Actor class is unaltered. The details of this behaviour are considered below.

The Internal Structure of an Actor, and extending the Toolflow Library

As described above, the Actor class represents the atomic behaviour of any signal processing element within the Rhino Toolflow. Thus by describing its internal structure in some detail, insight may be given as to how all signal processing elements could exist within the Rhino framework, a key characteristic of the developed system. Figure 6.5 shows the full UML class description of the Actor class, as implemented in Rhino Toolflow Library. A subclass used in the implementation of the HDFT algorithm, the DFT is also shown, so as to demonstrate how the Actor class may be used to create any signal processing operation. The Attributes and Methods of the Actor class are described in table form below, so as to provide insight as to how a instance of this class could represent a signal processing operations in accordance with synchronous dataflow theory.

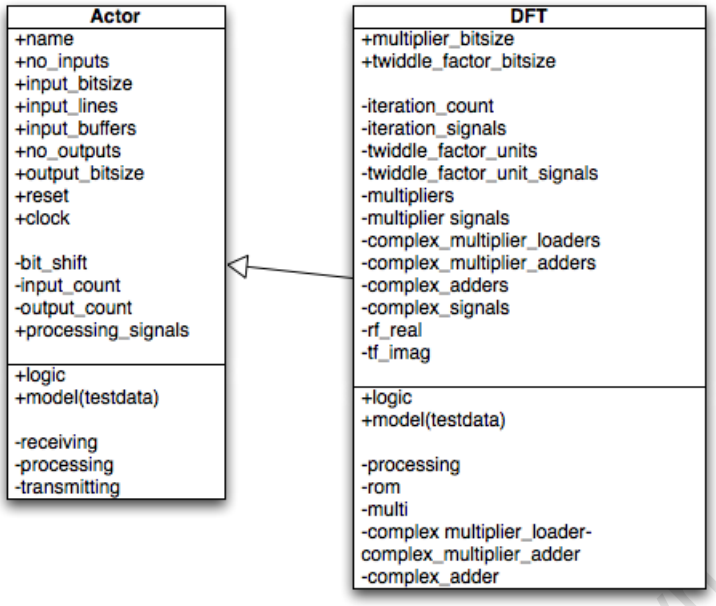

Figure 6.5: A UML class diagram, showing the base class of the library and one its subclasses, the DFT

Attribute Name	***Use***	***Designation***	***Purpose***
name	Public	Identification	As mentioned above, the name attribute provides a unique type descriptor for the class with the Rhino Toolflow Framework.
no_inputs	Public	Parametrisation	The number of input samples required before the Actor must activate
input_bitsize	Public	Parametrisation	The bit width size of the input data line[4]
no_outputs	Public	Parametrisation	The number of output samples produced by the Actor when processing is complete[5]
output_bitsize	Public	Parametrisation	The bit width of the output data line. A positive difference between this and the input bit width implies a truncation of the output data word.

[4]It is assumed that there is a real and complex data line, and are of equal bit width

[5]For the case of the trivial operation performed by the Actor base class, this value must equal the number of input samples to remain in accordance with Synchronous Dataflow Theory

input_lines	Public	Functional	These are MyHDL signal objects that represent the input data lines as well as the control lines.
input_count	Private	Functional	This internal index value is used to populate the internal, input buffer
input_buffer	Public	Functional	The input buffer is the MyHDL intbv object data value array which data values are read into, before being processed.
output_lines	Public	Functional	The MyHDL signal objects that represent the output data line as well as control information required for transferring the processed data out of the class object.
output_count	Private	Functional	This internal index value is used to ensure that the full output data set is transmitted
output_buffer	Public	Functional	The input buffer is the MyHDL intbv object data value array which data values are written into, after being processed.
Clock	Public	Functional	The MyHDL clock signal is the underlying control of the system that triggers its operations.
Reset	Public	Functional	The MyHDL reset signal is the asynchronous reset that triggers of the system that triggers its operations.
Internal mode and communication signals	Private	Functional	Assorted MyHDL signals are used internally to control the operation of the Actor class.

The attributes above are classified according to their use, Public or Private[6] and designation, either Identification, Parametrisation or Functional. Identification variables provide a unique type identifier within the Rhino toolflow library. The Parametrisation variables define the behaviour of the Actor in terms of how it performs its designated signal processing. And finally functional variables are the allocations of memory that are actually used during the class's operation. The class Methods below are classified according to their use, "Public" or "Private"

[6]in Python, all attributes of a class are accessible, so this description is purely for organisational purposes, i.e. private attributes are only intended for use within the class, while public attributes are use passing information in and out of the class object.

and their designation, either behavioural or procedural. Behavioural methods are those which describe some sort of signal processing behaviour using MyHDL generator functions. Procedural methods are those that perform some functional use to the class, in the context of the Rhino toolflow.

Name	*Use*	*Designation*	*Description*
logic	Public	Procedural	This method returns instances[7] of the signal processing behavioural methods of that particular class, as is required for various operations such as simulation and conversion to a HDL by the MyHDL libraries
model	Public	Procedural	This method mimics the behaviour of the signal processing element that the class represents, but performs it upon a list passed to it as an object. This is used in the modular development concept described below.
receiving	Private	Behavioural	This method contains the MyHDL generator function that pertains to the reception and storage of data in the internal data buffer. It may occur in parallel with the transmitting behaviour. When the correct number of inputs has been received, the processing behaviour is automatically triggered.
processing	Private	Behavioural	MyHDL generator that describes the behaviour that happens when the correct number of input samples have been received. In the Actor class, the samples in the input buffer are merely transferred to the output buffer. Upon its completion it indicates to the receiving behaviour that it may occur again, as well as triggering the output behaviour.
transmitting	Private	Behavioural	This is governs the outputting of the data in the output buffer, upon completion of the processing operation. Its behaviour is also governed by the ability of the external interface to receive the data.

As mentioned before, and suggested by Figures 6.4 and 6.5, additional signal processing operations are added to the proposed toolflow library by inheriting the Actor base class, and overriding the behavioural methods and logic procedural methods as needed. Additional attributes may be also be added as necessary, if additional parametrisation or functional memories are required. However, duplication of code is avoided, and coherency with synchronous dataflow theory is ensured by following this implicit structuring of the Toolflow Library. Thus be leveraging the expressive power of a high level programming language, a means for describing abstracted software defined radio systems has been enabled.

[7] pointers to instantiated methods tied to a particular instantiation of an object

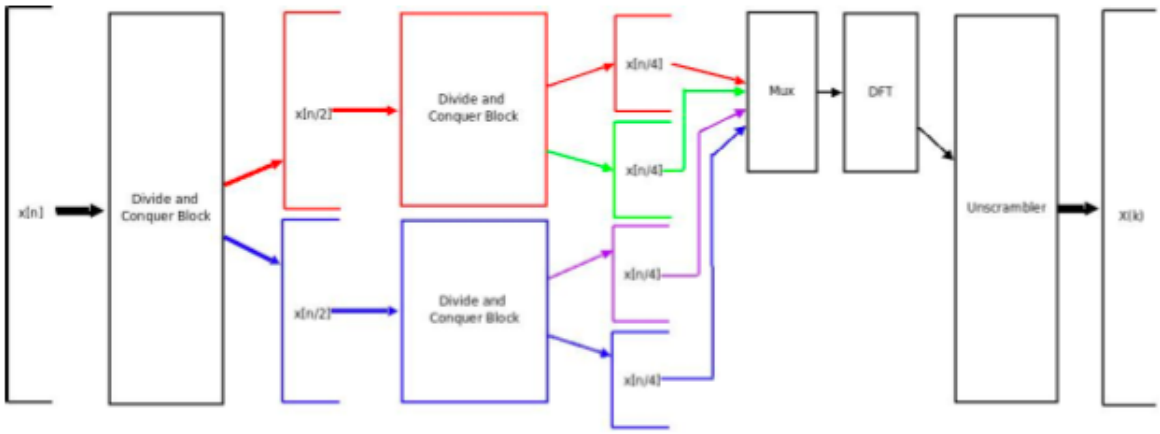

Figure 6.6: Implementation of Proposed System for the case of $d = \frac{N}{4}$

Built-in Verification Mechanism and Modular Development

As described in the description above, the Actor class contains a model method that can provide a reference for which the behaviour methods of the class may be compared to. This is line with the toolflow requirement of modular completeness, in that each element created contains within it the tools necessary to validate its correct functioning. This allows for a general purpose test bench to be developed that may test the facilitate the development of new signal processing operations, providing support to the advanced application developer in line with the broad user interface requirement. A description of the modular test benches for the HDFT algorithm implementation may be found in Appendix B.

6.3.2 Systematising of the HDFT algorithm

Now that the nature of the toolflow library has been elaborated upon, the description of how the HDFT algorithm was converted into a software defined radio system within the toolflow is given. It begins by considering the sequencing of operations, which suggests the signal processing operations, and hence modules to implement the algorithm in the toolflow. These modules are then elaborated upon, and a description of how one operation, the HDFT may contain other processing operations is provided.

Analysis of the Signal Processing Operations Required

As discussed in the section describing of the HDFT algorithm, the algorithm begins with a series of "divide and conquer" operations, each of which perform the standard radix-2 butterfly operation, dividing the data into two smaller data sets, one being the addition of the top half by

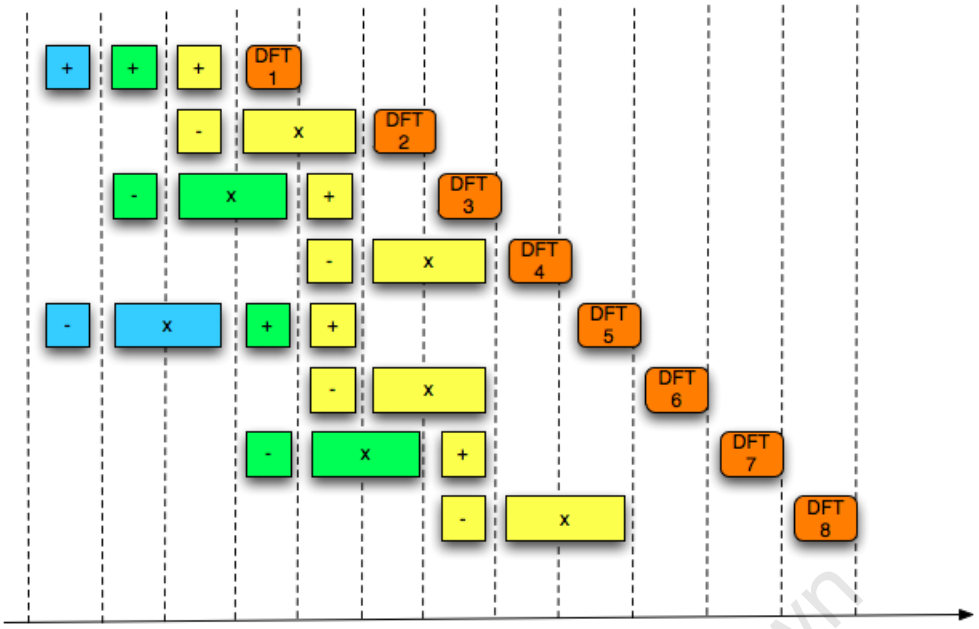

Figure 6.7: Scheduling of operations within system, for the case when $d = \frac{N}{8}$, and thus there are 3 divide and conquer stages before the DFT stages

the bottom half of the data set, while the other is the subtraction followed by the multiplication by complex coefficients (or twiddle factors). Upon completion of the divide and conquer stages, the results are then fed sequentially fed into a direct DFT computation module. The outputs from the DFT unit are then collected, and when the entire data set has been processed, the data is reordered to correctly produce the complete DFT. The system employs the inherent pipelining introduced by the irregular output of the datasets from the divide and conquer stages, allowing for a single DFT module to be utilised (as is demonstrated in Figures 6.6 and 6.7). The use of a single DFT module is desirable, as it allows for this computational intensive operation to be more easily optimised[22].

The Resulting Toolflow Modules

As a result of the operations analysis, the following software defined radio modules were required and hence developed utilising the modularised development model described above[8]:

- A *Divide and Conquer* module which performs the butterfly operation. It will have to contain memory to store the required twiddle factors, as well as a second set of output variables to represent the two operations being performed.
- A *Mux* module that takes in a specified number of sets of input signals and sequentially route the results of these input signals straight through it. This module would take in a set

[8]The full commented source code may be found in appendix B

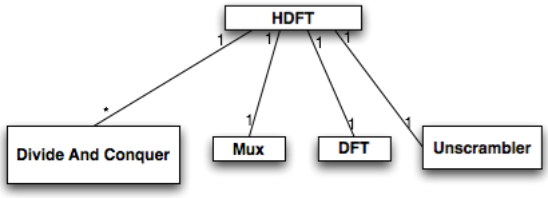

Figure 6.8: Structure of Hybrid DFT, in terms of Association

of input signals, and provide the logic to connect these signals into the system iteratively. This required little modification to the behavioural methods of the Actor superclass, but rather necessitated the use of a asynchronous logic tree of 2 input muxes to provide the parameterisable mux required.

- A *DFT* module that performs the direct DFT operation required. Of all the modules implemented, this required the most complex processing method modification, with the use of additional synchronous and asynchronous logic in completing the operations over several clock cycles. As a point of optimisation for the algorithm, the default hardware conversion code for the multiplications required in the DFT were overwritten with a Xilinx DSP IP core, so as to make use of the Rhino FPGA's built-in multiplication units. This was done both to converse resource usage and optimise performance of the algorithm.

- An *Unscrambler* module, that receives multiple sets of subsets of inputs, and depending on the an index mapping passed to it, remaps this data accordingly.

One of the characteristics of a system defined in terms of synchronous dataflow modelling is that of granularity[26, 27], in that operations should be able to be grouped together, and presented as a single operation. This was accomplished for the case of the HDFT algorithm by describing the system that would implement the algorithm, and then containing those descriptions within the logic procedural method of the HDFT container class, as suggested by figure 6.8. The input signals for this HDFT container class were then mapped onto the first stage of the required Divide and Conquer Modules, while the output signals were linked to the output signals of the unscrambler module that is the last required module in the HDFT system. The arithmetic model used above to provide insight into the error performance of the algorithm was used to provide the logic procedural method, while the behavioural methods of the class were disabled.

Thus a system level version of the module development test could be used in the development and evaluation of the HDFT on the system level, demonstrating the completeness of the toolflow prototype.

6.4 Hardware Implementation

This section describes the implementation and evaluation of the HDFT algorithm upon the Rhino platform, and hence the capability of the proposed toolflow to implement software defined radio system upon the targeted Rhino system. It begins by considering an extremely optimised implementation of the FFT upon the Spartan 6 FPGA from Xilinx[58]. It then describes the experimental setup used to evaluate the HDFT algorithm both in order to verify its implementation, as well as to asses its latency performance. Finally the performance evaluation of the algorithm is described, as compared to the optimised Xilinx FFT.

6.4.1 The Xilinx FFT

The Xilinx FFT parameterisable module or LogiCore library provides the de-facto standard for scalable FFT implementations on Xilinx FPGAs[38, 51, 58]. The core is capable of implementing the Cooley-Tukey algorithm for input sample set sizes of the range 8-65536, making use of both radix 2 and radix 4[9] versions of the algorithm. It also provides several arithmetic modes and data word width specifications and options within those modes, allowing for coarse as well as fine control over the precision of the algorithm. Finally it provides options with regards to implementation upon Xilinx FPGA chips, in terms of the usage of features such as memory and dedicated DSP units[10].

The IP core seeks to provide a system designer with clear control of the trade off between performance (both arithmetic precision and latency) and resource usage. And what is clearly acknowledged is that improved performance and higher resource usage are correlated in a linear fashion, and that it is up to the design engineer to manage this tradeoff. Appendix C contains the pertinent performance and resource usage results for the Spartan 6 SLX150T used in the Rhino System.

6.4.2 Experimental setup for the HDFT Algorithm Hardware Evaluation

The verification of the HDFT algorithm implementation upon the actual reconfigurable computing hardware was conducted in a similar manner to that of the low pass filter in Chapter 5, and was conducted upon the SP605 evaluation board, as none of the Rhino system's data interface ports were operational at the time of experimentation. A specialised test bench was constructed that contained a test waveform in a ROM that was then fed into the system. The resulting wave-

[9] i.e. deconstructing the algorithm down to DFTs of size 2 or 4

[10] called slices in the Xilinx nomenclature

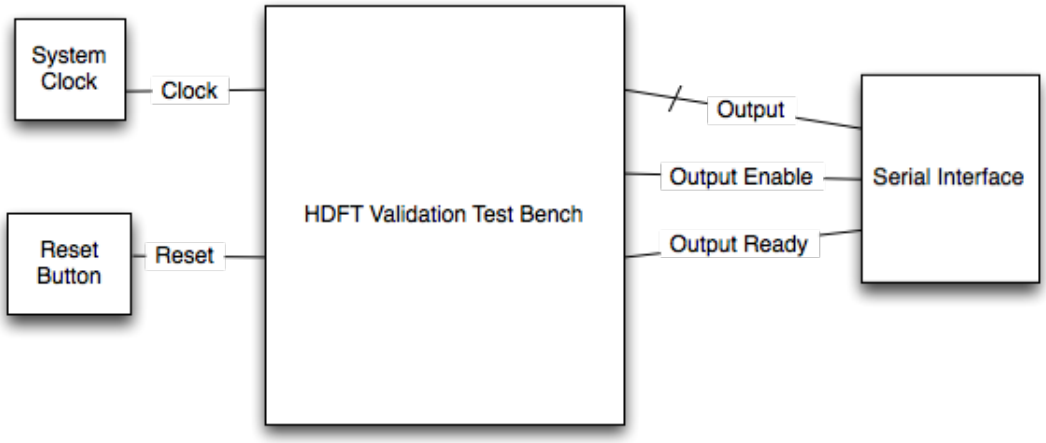

Figure 6.9: HDFT System Verification Test Bench

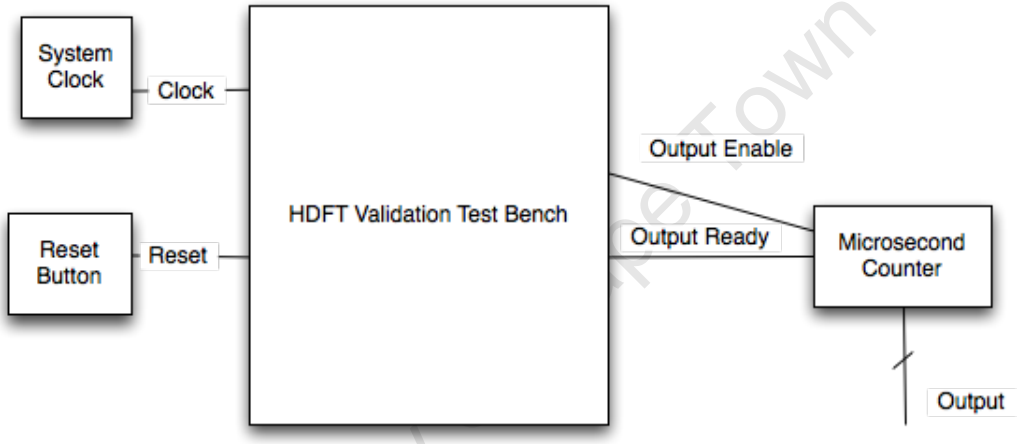

Figure 6.10: Performance Experiment Test Bench

form was then output to the USB UART interface, and verified against the arithmetic model of the algorithm prepared.

The performance characterisation was performed on the actual Rhino Computing Platform[11] by making use of the system validation test bench described above, coupled with a microsecond counter as shown in figure 6.10, that used the system clock to test the number of microseconds to process the test waveform. The output value was then output to the status indicator LEDs on the Rhino Platform.

6.4.3 Performance Evaluation

A set of experiments were conducted in order to evaluate the performance of the practical implementation described above, implementing a 128 input sample form of the algorithm upon

[11] Which has a Spartan 6 SLX150T FPGA with far greater capability than that of the SP605 evaluation board[56]

the Rhino system, as well as numerous simulations of larger sizes of the algorithm. The Rhino system features a Xilinx Spartan 6 SLX150T FPGA. This allowed for direct DFTs between sizes of 4 and 32, as 3 hardware multipliers are needed per complex multiplication, and there are 180 hardware multipliers available on the SLX150T FPGA (in the form of DSP48A1 logic slices).

This dual approach of a limited prototype implementation along with a larger volume of simulation work was adopted due to the large synthesis times required for these designs. However, the simulations were verified by comparing the implementation prototype to the simulation results for 128 sample versions of the algorithm, and it was found that there was agreement between the simulations and the actual implementation.

Two metrics were considered:

1. The resource usage, in terms of the available resources on the FPGA. This was measured from the synthesis reports of the ISE software for the prototype system[57].
2. The performance latency, measuring how long it takes the algorithm to compute the DFT. This was measured in microseconds, utilising the specialised evaluation test bench described in the section above.

Resource Usage

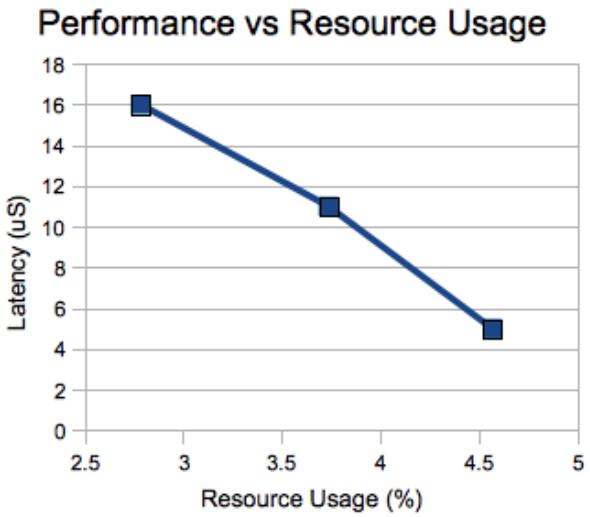

Figure 6.11: Latency Performance vs Resource Usage from the actual Rhino HDFT Implementation

The trends established by the Xilinx LogiCore and other scalable FFT implementations on the FPGA are born out by these experiments[38, 51, 58]. It is shown in Figure 6.11 that as the

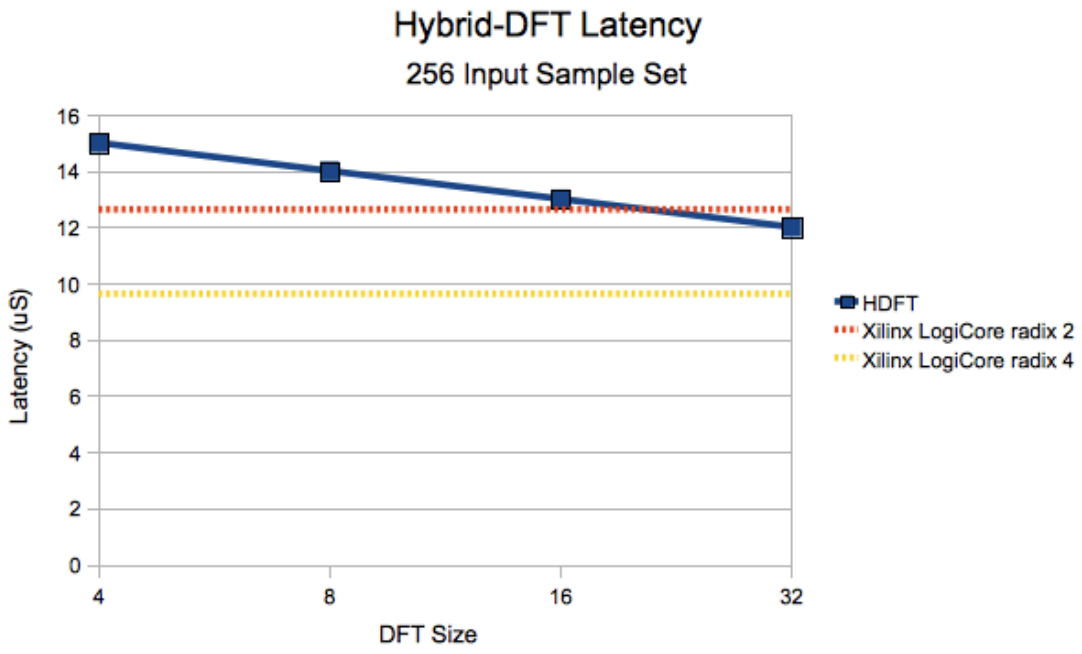

Figure 6.12: Simulated Latency Performance for Hybrid DFT Algorithm for sample set of size 256

throughput rate is increased (i.e. increasing the size of the direct DFT in the case of the proposed algorithm), the resource usage increases. Further experimentation is required to establish the exact nature of the trend, but it does appear to be an exponential growth, which is expected given the exponential growth in required operations. The full results for the experimentation may be found in Appendix C.

Latency Performance

As predicted, Figures 6.12 and 6.13 show that the larger the size of the DFT operation performed, the fewer the divide and conquer operations required, and hence the lower the sequential run time. With an optimisation of the clock frequency, it is predicted that even the algorithm's prototype implementation would compare favourably with the Xilinx LogiCore library. Figure 6.11 compared the performance to the resource usage for the implemented system, demonstrating a clear, linear relationship between latency and resource usage, which is also in line with Xilinx LogiCore library.

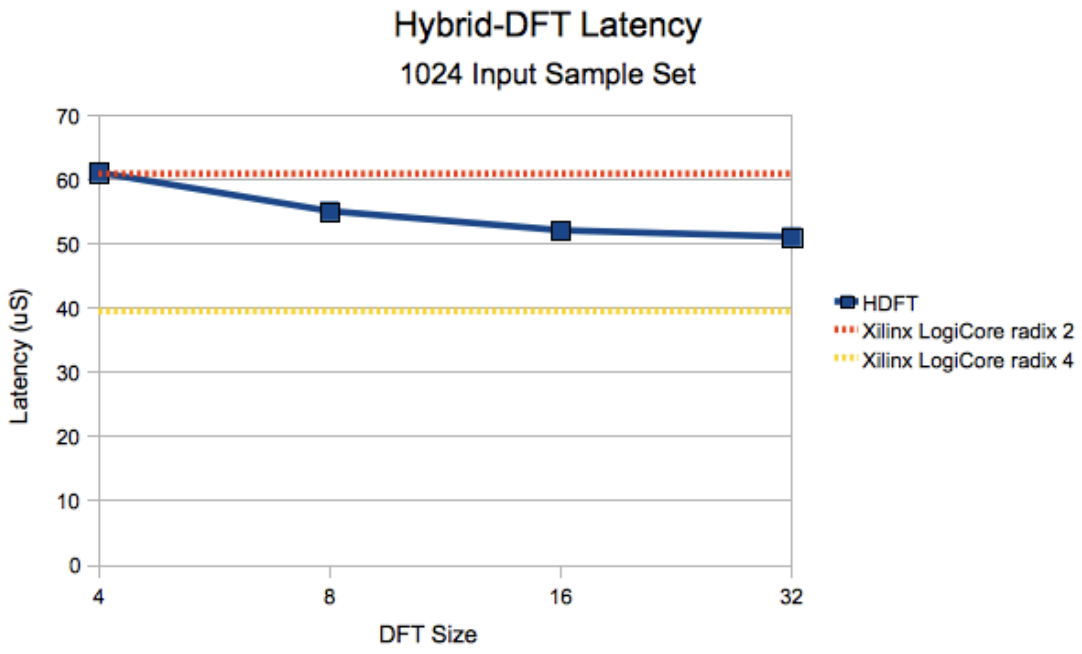

Figure 6.13: Simulated Latency Performance for Hybrid DFT Algorithm for sample set of size 1024

6.5 Evaluation of Second Development Prototype

This chapter described the prototyping and development of an experimental, scalable Discrete Fourier Transform Algorithm utilising the Toolflow library developed, in line with the toolflow concept developed in Chapter 4 and exploratory work done by the previous development prototype in Chapter 5. This section describes the progress achieved by this prototype in terms of the components identified in Chapter 4.

This Chapter began by considering the algorithm under development, and linked this to the development of a fixed point arithmetic model of it, which showed consistent error stability across a variety of implementations. It then developed the System level model of the prototype, as envisioned by the toolflow concept. This was done by developing the required signal processing operations in conformance with the Synchronous Dataflow Modelling informed Toolflow library methodology. Finally the algorithm was implemented in hardware and not only verified, but also its performance was compared to that of an industrial implementation.

The chapter that follows will evaluate the practical efforts described in both this chapter as well as the previous one, in terms of not only achieving the specification derived in Chapter 3, but also linking back to the fundemental objective of this dissertation. Further recommendations will also be made with regards to the further development of software defined radio toolflows.

6.5.1 Formal Description

Significant work was done in this Chapter to provide support for the description of software defined radio systems in accordance with synchronous dataflow theory. This support primarily took the form of how the toolflow library objects were structured, inherently making the systems created using the framework in accordance with synchronous dataflow theory. Although analysis based upon the theory is not utilised in the rest of the toolflow, by ensuring compliance with it the possiblity remains open.

6.5.2 System Model

Similarily to the previous prototype, a MyHDL model of the desired system was created that predicited the behaviour of the hardware implementation of the system. However several critical developments in this prototype deepened the utility of system models within the toolflow. The expressive power of the Python language was harnessed to create a library of signal processing actors, standardising behaviour while reducing the amount of new code that had to written to introduce new signal processing operations. Furthermore the concept of actor and system testbenches that make use of validation methods within the toolflow library objects further facilitates the development of new signal processing operations within the toolflow library.

6.5.3 Hardware Implementation

The implementation of the algorithm upon the reconfigurable platform, Rhino was undertaken, along with considerable characterisation of those implementations. It was found that the implementation compared favourably to optimised implementations of the algorithm upon similar hardware, in terms of latency and resource usage.

6.5.4 Flow Coherency and Deployment Mechanisms

As before, little work was done to automate the toolflow processes. However with the improved structure of the system model introduced by the toolflow library, the coherency and deployment mechanisms were made considerably easier.

Chapter 7

Conclusion

This chapter concludes this dissertation, and evaluates the validity of the claim that *it is possible to describe a software defined system using synchronous dataflow theory, and then implement that model of the system upon a reconfigurable computing platform* in light of the work undertaken.

It does so by evaluating the toolflow concept and its components developed in Chapters 4, 5 and 6, in terms of the project specification and evaluation criteria derived using the systematic analysis of the problem and its context in Chapters 1, 2 and 3. By demonstrating the achievement of the specification developed, the direct objective of this dissertation is shown to have been realised.

It then considers the project outcomes, in terms of the application of systems engineering to a project of this nature, as well as the conclusions with regards to the implementation of software defined radio systems as a result of this work. More broadly, reflective learning is considered, on the outcomes of the project that do not directly pertain to systems engineering or Software Defined Radio.

It concludes by considering the possible avenues of future work that might be conducted upon this topic, building upon the work described in this dissertation.

7.1 Evaluation of Toolflow

The bulk of this dissertation was concerned with the conceptualisation, development and refinement of the tools and support systems for assisting the end-user in configuring the Rhino System to perform their desired software defined radio operations. This was done in service of a hypothesis that it was possible to make such a transition; going from the user-defined sys-

tem described in terms of synchronous dataflow theory to a practical implementation upon the Rhino platform.

A specification was derived in the analysis performed in Chapter 3 that drove this detailed design process described in Chapter 4, as per the broader Rhino Development Process and the review of current Literature in Chapter 2. The toolflow that evolved as part of an iterative, "Sprial" development process through two development prototypes, will now be assessed in terms of the three critical features identified by that specification and the evaluation criteria identified therein. The constraints derived by the same analytical process are inherent in how these features were implemented.

The most mature form that the toolflow developed took was that of a software library of signal processing operations. This library was created using the Python programming language, with the MyHDL library providing support for much of the operations performed. The library was structure to be in accordance with synchronous dataflow theory, and was demonstrated to be able to be used to implement software defined radio systems upon the Rhino reconfigurable platform.

Having assessed the implementation in terms of the specification derived from the systematic analysis of a software defined radio toolflow, the ultimate objective of this dissertation may be shown to have been achieved xomprehensively.

7.1.1 Layered User Interface

The essence of the user interface conceptualised is the use of the expressive power of Python, the object orientated programming languages used to represent the user's desired software defined radio systems, in a manner that maps equally to their understanding as it does to implementation. The further imposition of a formalised, mathematically-verifiable paradigm in shaping the toolflow library of object strengthens the mapping of the system described to the platform concerned, while only requiring the most minimal additional knowledge on the part of the end-user.

This choice of interface to the end-user also supports the varying levels of knowledge and training in the end-user group, as the novice user may interact on a simplistic level using predefined toolflow library objects, creating simple systems, while the more advanced end-user may delve deeper into the creation of the objects themselves, creating their own, as well as optimising by implementing optimised IP cores, such as what has been done in the second development prototype. The open nature of the interface is in direct alignment with the constraint.

As previously acknowledged, the fulfilment of this feature is somewhat subjective, but the ease with which fairly complex signal processing algorithms were implemented in both prototypes

suggests the strength of this interface. The final validation of this feature lies in the future, as to where this toolflow library, or further iterations of it see widespread adoption and use.

7.1.2 Validated Abstraction

The second toolflow feature considers the notion of multiple levels of abstraction, between the end-user and implementation of their desired software defined radio systems. Three levels of abstraction were proposed and implemented, spanning the spectrum from abstract model to hardware implementation.

The first level of abstraction was implicit, in that the end-user is constrained into describing the software defined radio system they desire in terms of synchronous dataflow theory, as the software objects used conform to this theory. Whether the user makes use of the large body of support work in this theory to inform their design or not, its mathematical completeness results in systems that may be translated to further levels of abstraction.

The next level of abstraction and validation is on the system level, having translated the user defined topology into a coherent system that may be simulated using the supporting MyHDL library. These simulations form a vital point of verification, as it provides the end-user with information pertaining to the performance of their system, and utilises the fact that they will be the best judge of this information. These simulations have also been shown to predict the behaviour of the toolflow elements when implemented on the FPGA platform extremely robustly in Chapters 5 and 6, at a fraction of the required preparation time to actually implement the desired operations on the FPGA. This is an extremely important result, as timely yet accurate simulation results are critical in FPGA application development.

The final level of abstraction is not abstract at all, as it is the implementation of the system upon the platform. Here the vendor tools have to be used to implement the system upon the platform, however these tools do provide accurate feedback with regards to the implementation of the design that might be of use to the end-user, and so the utility of them should not be dismissed, simply because it is "forced" component of the toolflow.

The three levels of abstraction implemented constitute a valid and smooth "flow", that translates from the desired, abstract software defined radio system of the end user, to something that may be implemented upon the Rhino platform, and so this feature has been fulfilled.

7.1.3 Modularity

The final feature requires that the Toolflow makes provision for diversity, so that it hypothetically could support any signal processing operation.

And the implemented toolflow library, through the use of the implicit standardisation of object inheritance makes allowance for this. Several extremely distinct operations have been developed through the framework in the second development framework, and the concept has held up well. Furthermore the natural support seen for varying levels of operation granularity within the framework suggest that the toolflow library supports an extremely broad range of operations.

However, the additional support provided by the built-in module and system validation features suggest that the toolflow goes beyond merely supporting any operation, and in fact facilitates further development of these operations within the framework. So the toolflow library is fact not only modular, but it in fact encourages the development of a diversity of modules.

7.2 Project Outcomes

This section reflects upon the work done as part of this dissertation, and the implications for the academic fields of systems engineering as applied to projects of this nature, as well as the software defined radio thesis.

7.2.1 Systems Engineering

This project demonstrated a novel application of Systems Engineering, in its use in application to the development of a tool for fairly broad application in an academic context of development. Typically Systems Engineering is restricted to application in very particular contexts of operation, and development occurs in tightly controlled, usually commercial development environments.

The broad applications of the tool were a considerable challenge, but it prompted further deep analysis of the stakeholders of the system, particularly the end-users. The resulting consideration of the different classes of users provided to be critical in the conceptualisation of the toolflow, and hence the toolflow developed and implemented. Thus it suggests that the application of systems engineering in this project reinforces the utility of the analytical and development framework provided by the field.

The academic context which this project was completed in, in parallel with the other projects that made up the development of the Rhino System meant that often development was occurring in parallel, along very different timelines. However, the deep insight provided into the system as a whole allowed for design decisions to be made that were not only coherent with other aspects of the project, but could also respond to design decisions made in parallel. Thus the support during the development process that the application of systems engineering provided proved

to be important, and hence advocates its general utility. This responsiveness to change is in line with current thinking in Systems Engineering, particularly those advocating a more Agile approach.

7.2.2 Software Defined Radio

As identified in the initial consideration of the problem context, Software Defined Radio is a field that is rapidly coming of age, and is on the brink of transitioning from the academic and experimental to that of practical application. With this widespread adoption, resources and tools are required that shall facilitate training with this domain, and indeed it is with this need that the Rhino System is concerned.

The toolflow developed, as it is the means by which the end-user will describe their desired operations to the Rhino System, so that these operations may be implemented, is obviously crucial in this facilitation and training role. The toolflow developed demonstrated that it is possible to make use of a formalised digital signal description methodology, synchronous dataflow modelling, and by translating the crucial elements of that methodology into a practical framework that may be implemented, it is possible to create a general tool for implementing software defined radio systems, provided that they are described using that formal, abstract methodology. This outcome has value beyond easing the creation of software defined radio systems, as it separates the description of the systems from its implementation, crucial to understanding the theory that underpins the field.

This translation of the abstract system to the practical is the thesis that this dissertation is concerned with, and the successful analysis and development of the toolflow described not only validates it, but has also enhanced characterisation of its facets: The abstract description of software defined radio systems has shown to link not only to a large body of digital signal processing theory, but also provide an opportunity for providing an easier entrance point for engineers new to the field. The practical implementation of these abstract systems necessitated a need for flexibility that has been shown to be of great utility in creating scalable systems, possibly better suited to reconfigurable computing. Furthermore, the toolflow developed enhances the goal of the Rhino System, which in turn enhances the field of Software Defined Radio more generally.

7.3 Reflective Learning

The outcomes of this dissertation were not confined to software defined radio nor systematic development, and this section encapsulates the “lessons learned” in pursuing the thesis.

It is proposed that good engineering is seeking to create something, and doing it well. Great engineering is first understanding why something may be needed first, and then using that knowledge to create something in response to that need.

The difference between the two is the conscious acknowledgement and analysis of the problem under consideration. This might seem obvious, however as has been observed by social scientists of recent years, humans are not entirely rational beings. And engineers, despite appearances at time to the contrary, are human. Human faculties such as intuition and experiential learning are enormously helpful to engineers in solving problems, but it is these faculties that are also enormously helpful in allowing engineers to avoid consciously engaging with problems.

This project has sought to consciously engage with a problem, systematically analyse it, and develop a solution while continuously reflecting upon the nature of the need being addressed. Doing so was a constant struggle not only against the challenge faced, but also the methods employed in facing that challenge. However, to do otherwise would have resulted in a solution that could possibly have met the need that prompted the project, but not addressed it.

7.4 Future Work

In conclusion, this project has gone some of the way in creating a software defined radio system for the South African tertiary education context while addressing a serious academic question in the field, however further work remains to be done. Two broad paths of further development are proposed, based upon the solid foundation of work completed in this thesis. This work would both bring the proposed toolflow to the operational prototype phase of its development, as well as further the field of software defined radio.

1. Firstly, the existing toolflow library of signal processing objects may be better populated signal processing operations, as required by other applications. A substantial project would be to survey the general operations required by signal processing, and create a comprehensive subset of these operations within the library. Further accompanying resources, for the use of the toolflow in certain contexts such as Digital Signal Processing or Radar research may be developed.

2. Secondly, the toolflow may be further automated, tying the three stages of abstract development into a more uniform structure, so as to better facilitate its use. It will however be critical to do so in manner which does not compromise its flexibility or openness.

Bibliography

[1] 2001. URL `http://agilemanifesto.org/`.

[2] Pearu Peterson and others. Eric Jones, Travis Oliphant. SciPy: Open Source Scientific Tools for Python.

[3] Krste Asanovic, R Bodik, BC Catanzaro, JJ Gebis, and P. The landscape of parallel computing research: A view from berkeley. *EECS Department, University of California, Berkeley*, 2006.

[4] John Bard and Vincent J. Kovarik. *Software defined radio: the software communications architecture*. John Wiley and Sons, 2007.

[5] David M. Beazley. *Python Essential Reference*. Addison-Wesley, 2009.

[6] B.S. Blanchard and W.J. Fabrycky. *Systems engineering and analysis*. Prentice Hall, Upper Saddle River, NJ, 3rd edition, December 1998.

[7] Eric Blossom. GNU radio: tools for exploring the radio frequency spectrum. *Linux Journal*, 2004(122), 2004.

[8] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21 (5):61–72, May 1988.

[9] T Braun. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, June 2001.

[10] D M Buede. *Engineering Design of Systems - Models and Methods*. John Wiley & Sons, 2000.

[11] CASPER. Rapid Development of Radio Astronomy Instrumentation Using Open Source FPGA Boards, Tools and Libraries, 2009.

[12] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput*, pages 297–301, 1965.

[13] Stephen Craven and Peter Athanas. Dynamically Reconfigurable Radios from a High-Level Specification. *Computer*, 2008.

[14] John Davis, Christopher Hylands, Jorn Janneck, Edward Lee, Jie Liu, Xiaojun Lui, Steve Neuendorffer, Sonia Sachs, Mary Stewart, Kees Vissers, Paul Whitaker, and Yuhong Xiong. Overview of the Ptolemy Project, 2001.

[15] Jan Decaluwe. MyHDL: a python-based hardware description language. *Linux Journal*, 2004(127), 2004.

[16] Jan Decaluwe. MyHDL - From Python to Silicon!, 2010.

[17] Markus Dillinger, Kambiz Madani, and Nancy Alonistioti. *Software defined radio: architectures, systems, and functions*. John Wiley and Sons, 2003.

[18] T. Doran. IEEE 1220: For Practical Systems Engineering. *Computer*, 39(5):92–94, May 2006.

[19] Linda Doyle, Robert Esser, Jorg Lotze, Suhaib Fahmy, Juanjo Noguera, and Baris Ozgul. Development Framework for Implementing FPGA-Based Cognitive Network Nodes. In *IEEE GLOBECOM 2009 proceedings*. IEEE Communications Society, 2009.

[20] Richard W. Hamming and Richard Wesley Hamming. *Numerical methods for scientists and engineers*. Courier Dover Publications, 1986.

[21] *Agile SYSTEMS ENGINEERING versus AGILE SYSTEMS engineering*, July 2005. INCOSE.

[22] Gordon Eric Inggs. *An Economy of Logic*. Undergraduate final year project, University of Cape Town, October 2009.

[23] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C.-L. Wang. Heterogeneous computing: challenges and opportunities. *Computer*, 26(6):18–27, June 1993.

[24] Alan Langman, Michael Inggs, Simon Winberg, and Et al. Rhino Project, 2011.

[25] E A Lee and D G Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on Computers*, 36(1):24–35, 1987.

[26] E A (IEEE) Lee and D G Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[27] Edward A Lee and Thomas M Parks. Dataflow Process Networks. *Proceedings of the IEEE*, 83(5):773–799, 1995.

[28] Yanbing Li, Tim Callahan, Ervan Darnell, Randolph Harr, Uday Kurkure, and Jon Stockwood. Hardware-Software Co-Design of Embedded Reconfigurable Architectures. In *Proceedings of the 37th Annual Design Automation Conference*, number c, pages 507 – 512. ACM, 2000.

[29] J Lotze, S Fahmy, J Noguera, L Doyle, and R Esser. An FPGA-based cognitive radio framework. In *ISSC 2008, Galway, June 18-19*, pages 138–143, 2008.

[30] W Luk, JGF Coutinho, T Todman, YM Lam, W Osborne, KW Susanto, Q Liu, and WS Wong. A high-level compilation toolchain for heterogeneous systems. In *Proceedings of the IEEE International SOC Conference*, pages 9–18.

[31] Christophe Moy and Mickael Raulet. High-Level Design Methodology for Ultra-Fast Software Defined Radio Prototyping on Heterogeneous Platforms. *Advances in Electronics and Telecommunications*, 1(1):67–85, 2010.

[32] Praveen K Murthy, Edward A Lee, and Shuvra S Bhattacharyya. Synthesis of Embedded Software from Synchronous Dataflow Specifications. *VLSI Signal Processing*, 21:151–166, 1999.

[33] OpenCores Organisation. Wishbone B4 - WISHBONE System-on-Chip(SoC) Interconnection Architecturefor Portable IP Cores, 2010.

[34] Aaron Parsons, Dan Werthimer, Donald Backer, Tim Bastian, Geoffrey Bower, Walter Brisken, Henry Chen, Adam Deller, Terry Filiba, Dale Gary, Lincoln Greenhill, David Hawkins, Glenn Jones, Glen Langston, Joseph Lazio, Joeri van Leeuwen, Daniel Mitchell, Jason Manley, Andrew Siemion, Hayden Kwok-Hay So, Alan Whitney, Dave Woody, Melvyn Wright, and Kristian Zarb-Adami. Digital Instrumentation for the Radio Astronomy Community. page 11, April 2009.

[35] SysML Partners. Systems Modeling Language (SysML) Specification. syseng.omg.org.

[36] Eric S. Raymond and Bob Young. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. page 241, 2001.

[37] N Sane, J Ford, A Harris, and S S Bhattacharyya. Prototyping scalable digital signal processing systems for radio astronomy using dataflow modeling, 2009.

[38] T. Sansaloni, A. Pérez-Pascual, and J. Valls. Area-efficient FPGA-based FFT processor. *Electronics Letters*, 39(19):1369, September 2003.

[39] Robert G. Sargent. Verification and validation of simulation models. In *Proceedings of the 37th conference on Winter simulation*, WSC '05, pages 130–143. Winter Simulation Conference, 2005.

[40] C. Shannon. The zero error capacity of a noisy channel. *Information Theory, IRE Transactions on*, 2(3):8 –19, september 1956. ISSN 0096-1000. doi: 10.1109/TIT.1956.1056798.

[41] Timothy J O Shea, N C State, T Charles Clancy, College Park, and Hani J Ebeid. Practical signal detection and classification in gnu radio. *Energy*.

[42] SW Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1998.

[43] Hayden Kwok-Hay So, Artem Tkachenko, and Robert W Brodersen. A unified hardware/software runtime environment for FPGA-based reconfigurable computers using BORPH. *ACM Transactions on Embedded . . .*, 2008.

[44] IEEE Computer Society Software & Systems Engineering Standards Committee. ISO/IEC 15288, 2008.

[45] IEEE Computer Society Software Engineering Committee. 1220-2005 IEEE Standard for Application and Management of the Systems Engineering Process, 2005.

[46] Ferrel G. Stremler. *Introduction to communication systems*. Addison-Wesley Pub. Co., 1982.

[47] S Swan. An introduction to system level modeling in SystemC 2.0. *Cadence Design Systems, Inc., draft report*, 2001.

[48] By Kun Tan, He Liu, Jiansong Zhang, Yongguang Zhang, Ji Fang, and Geoffrey M Voelker. Sora : High-Performance Software Radio Using General-Purpose Multi-Core Processors. *Communications of the ACM*, 54(1):99–107, 2011.

[49] *OMAP3530 / 25 Applications Processor*. Texas Instruments.

[50] Richard Turner. Towards agile systems engineering processes. *Defense Software Engineering*, pages 11–15, April 2007.

[51] I.S. Uzun, A. Amira, and A. Bouridane. FPGA implementations of fast Fourier transforms for real-time signal and image processing. *IEEE Proceedings - Vision, Image, and Signal Processing*, 152(3):283, June 2005.

[52] Danilo Valerio. Open Source Software-Defined Radio : A survey on GNUradio and its applications, 2008.

[53] T Weilkiens. *Systems engineering with SysML/UML : modeling, analysis, design : OMG Press series*. Morgan Kaufmann OMG Press, Burlington, 2007.

[54] *CORE Generator Guide*. Xilinx Inc., January 2009.

[55] *EDK Concepts, Tools, and Techniques*. Xilinx Inc., 2009.

[56] Xilinx Inc. Spartan-6 Family Overview, 2009.

[57] Xilinx Inc. ISE Design Suite Software Manuals and Help. 681:1–13, 2010.

[58] Xilinx Inc. LogiCORE IP Fast Fourier Transform v7.1, 2011.

Appendix A

Rhino Development Process Document

Revision History

Revision Number	Notes
0.1	Initial draft of development process document. Only contains the Systems Engineering Process overview, as well as the originating requirements analysis
0.2	Refinement of the Requirements Analysis phase. System Operational Concepts and Context added, as well as the System Requirements.
0.3	Further refinement of the Requirements Analysis, adding traceability as well as further details to the system requirements. Initial System Design phase structure proposed
0.4	System Design phase described in full, with full traceability. Ongoing and completed work in the subsystem design detailed.

Introduction and Overview of Development Process

This document describes the Systems Engineering process followed throughout the development of Rhino project. The overarching objective of the Rhino is to provide a computational platform for research and training in software defined radio for the South African tertiary education context. Thus, from the outset, the project has sought to service a complex set of objectives for several groups of diverse stakeholders. It was recognised that a thorough design process and clarity of thought had to be a priority throughout the development of the computational platform and its accompanying software tools.

In service of thoroughness and clarity, the Systems Modelling Language (SysML) was employed. As the Rhino project envisions a computational system comprising of a variety of digital hardware, as well as software and human elements, the project is an ideal candidate for description using SysML.

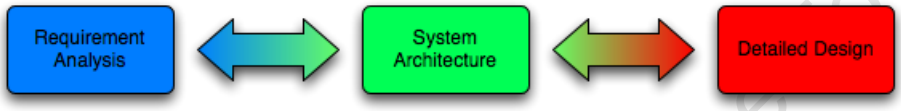

Figure A.1: Rhino Project Development Process

The Rhino Systems Engineering Development Process is based upon requirements-driven design approach commonly advocated[10, 18]. Each major developmental phase is identified in figure 1. It is also important to note that the directional flow of the diagram only indicates coherency, as it is entirely possibly for multiple phases of development to occur simultaneously. This document details the systems engineering of the project as described in figure 1, broken down according to the three development phases:

- The requirements analysis phase describes the characterisation of the problem concerned, i.e. the translation of the multiple objectives of the various stakeholders into unambiguous and clear requirements understood by the development team and ratified by the stakeholders[18]. This was undertaken by considering not only the originally stated requirements of the stakeholders, but also the context of the project as well as the envisioned operational use of the system[10].

- The system architecture elaboration phase details the conceptualisation, investigation and decisions which shaped the resulting Rhino system, in terms of requirements derived during the Requirements Analysis phase, as well as defining the key system elements. This process of iterative decomposition of the system into subsystems and subsystems of subsystems is an expected result in the system design process[44, 45].

- Detailed design phase describes the resulting system design, at a level necessary for implementation. These detailed plans for implementation are organised according to the subsystems identified by the block diagrams produced during the high level design phase, with possibly further logistical grouping. By in large this process is described in terms of the implementation, and as such does not make use of SysML.

Requirements Analysis

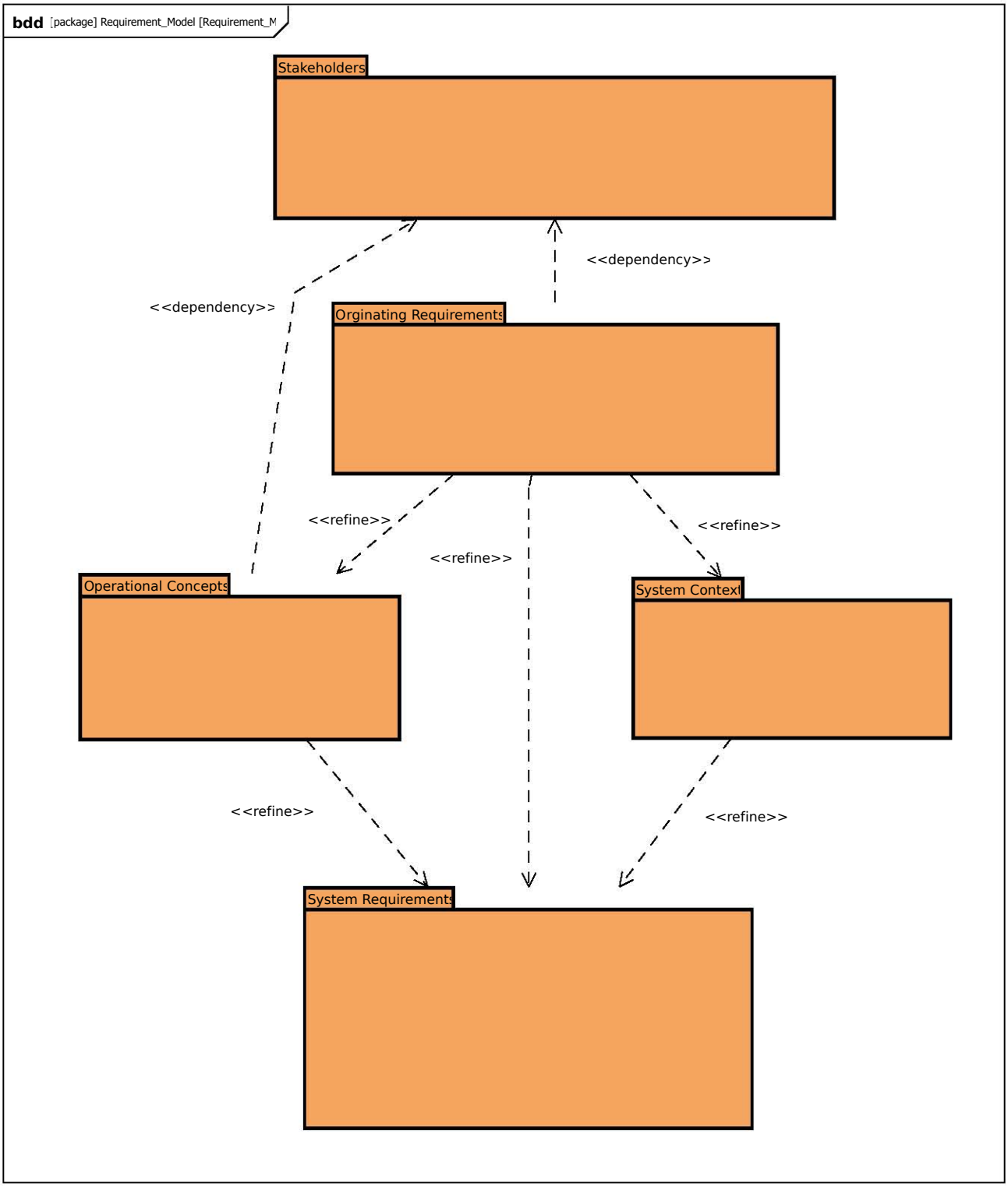

Figure A.2: Overview of Requirements Analysis phase of the development process

Figure 2 describes the Requirement Analysis phase of the project in terms of a SysML block definition diagram[53]. Considering the process analogous to an industrial production line, it takes in several inputs relating to the project's stakeholders and produces a set of coherent system objectives which can be mapped against a system design. Several SysML relationship stereotypes are used to express the relationship between the elements of the requirements analysis.

Firstly the project stakeholders were identified and grouped according to roles that each entity would play in relation to the project. Implicit in this grouping operation is consideration of the hierarchy of stakeholders - stakeholders have varying degrees of relevance as well as authority with respect to different aspects of the project.

From these group of stakeholders, a set of originating requirements were solicited, which are a clear statement from the stakeholders, expressing their need(s)[10].

From this set of originating requirements, a set of operational concepts were constructed, requiring additional consultation with the stakeholder (and often subgroups of that particular stakeholder group).

A system context diagram was also produced, based upon analysis of the operational context of the stakeholders.

From these three alternative descriptions of the requirements of the system, the system requirements were finally derived, describing the system in terms of what is required, in manner compatible with the rest of the design process.

Stakeholder Identification

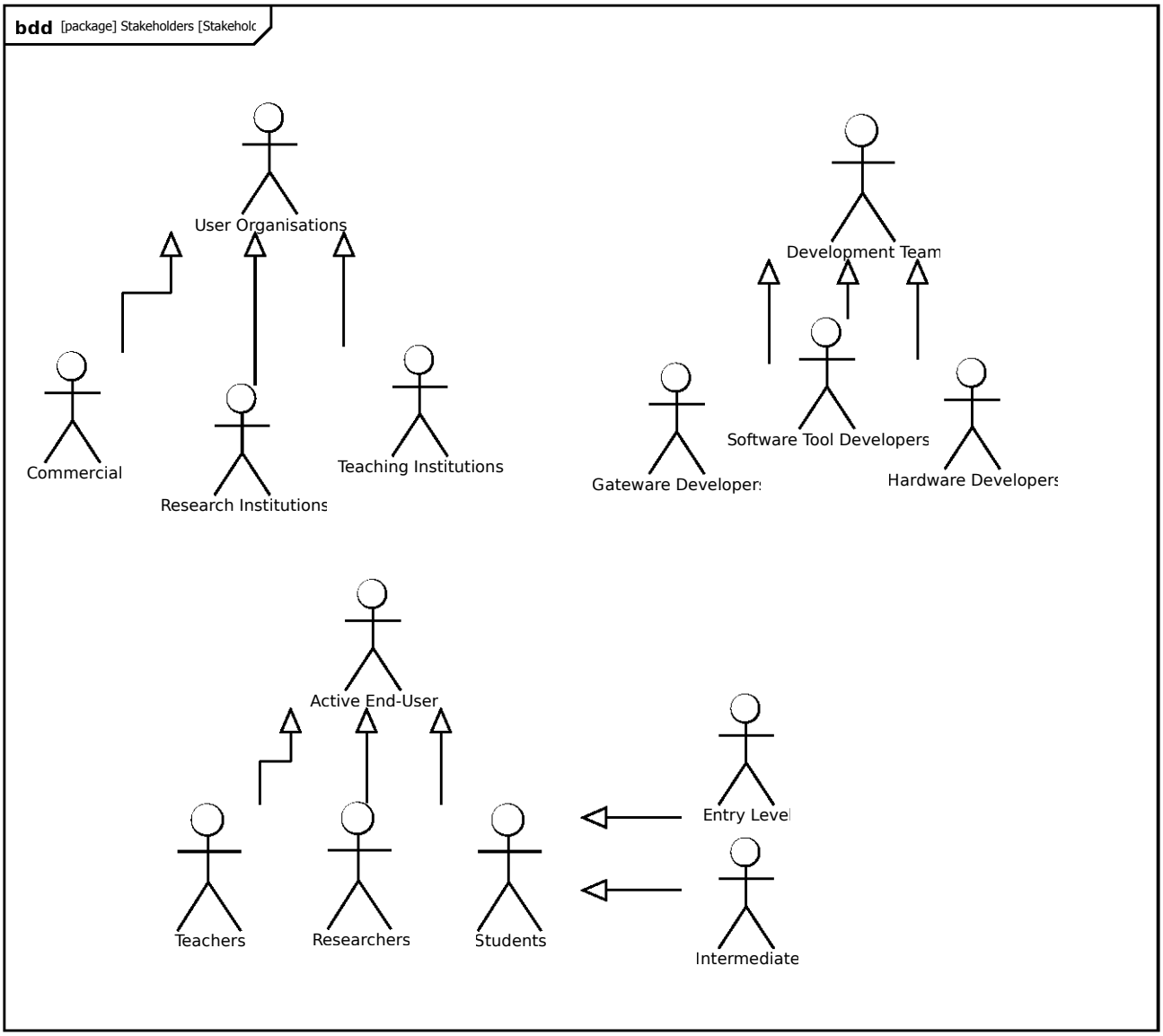

Figure A.3: Identification and grouping of Rhino Project Stakeholders

Stakeholders are defined in this context as any entity that has an interest or relationship to the system under development. The first task when considering the engineering of any system is identifying these stakeholders, and classifying them in terms of the role that they would play with regards to the system[10, 53, 44]. As a result, it will then be possible to rank the needs of these stakeholders into a hierarchy based upon the roles identified, an invaluable tool when making design trade-off decisions.

The Rhino project has multiple stakeholders with a number of requirements, however there are large degrees of overlap with regards to what the different groups require from the system. Hence, a SysML block definition diagram has been used to express this process of reducing

the nine main stakeholders into three main groups: Active end-users, user organisations and the development team members. This process of grouping stakeholders is described by figure 3, with the stakeholders represented using the actor stereotype and the grouping relationship represented using the generalisation relationship stereotype[53, 35].

These stakeholder groups are elaborated on below:

- **Active End-Users** - this stakeholder group is comprised of those which will make active use of the Rhino platform to meet their computational signal processing needs. As a result of this, this is the group which is further elaborated upon, in terms of deriving the operational concepts and system contexts within which the Rhino platform will operate. Three distinct roles comprise this group - the teachers and student who will make use of the platform for Software Defined Radio education, and the researchers who will make use of the platform for experimentation and development in the telecommunication, RADAR and Radio Astronomy application domains. Meeting the needs of this group are thus paramount in terms of the long-term success of the platform. By in large, the requirements from this group form the baseline for the Rhino Project's requirements.

- **User Organisation** - this group is made up of the organisations that support the active end-users and the development team that is developing the Rhino platform. In the short term, meeting the needs of this group is necessary to support the further development of the platform. In the longer term, fulfilling the requirements of this group will ensure the continuity and longevity of the Rhino System.

- **Development Team** - an oft-neglected group, when considering stakeholders in systems development, this group includes the engineers and scientists who are directly involved in developing the Rhino platform. In addition to conceptualising, designing and implementing the system, this group of people also have requirements for the system. Given the role played by this group in shaping the system, it is thus vital that these requirements are acknowledged and met in a manner which does not conflict with the other requirements under consideration.

Originating Requirements

The originating requirement statement is often where the formal system engineering process begins, representing the solicitation from the system's stakeholders what they require from the system, in their own words[10, 18, 53]. In formulating these originating requirements, it is important to note the underlying statement of intent above, that the Rhino project seeks to create a computational platform for software defined radio applications. All of the requirements

are hence written in response to this overarching goal, so it could be said that these are the requirements of the stakeholders for a software defined radio computational platform.

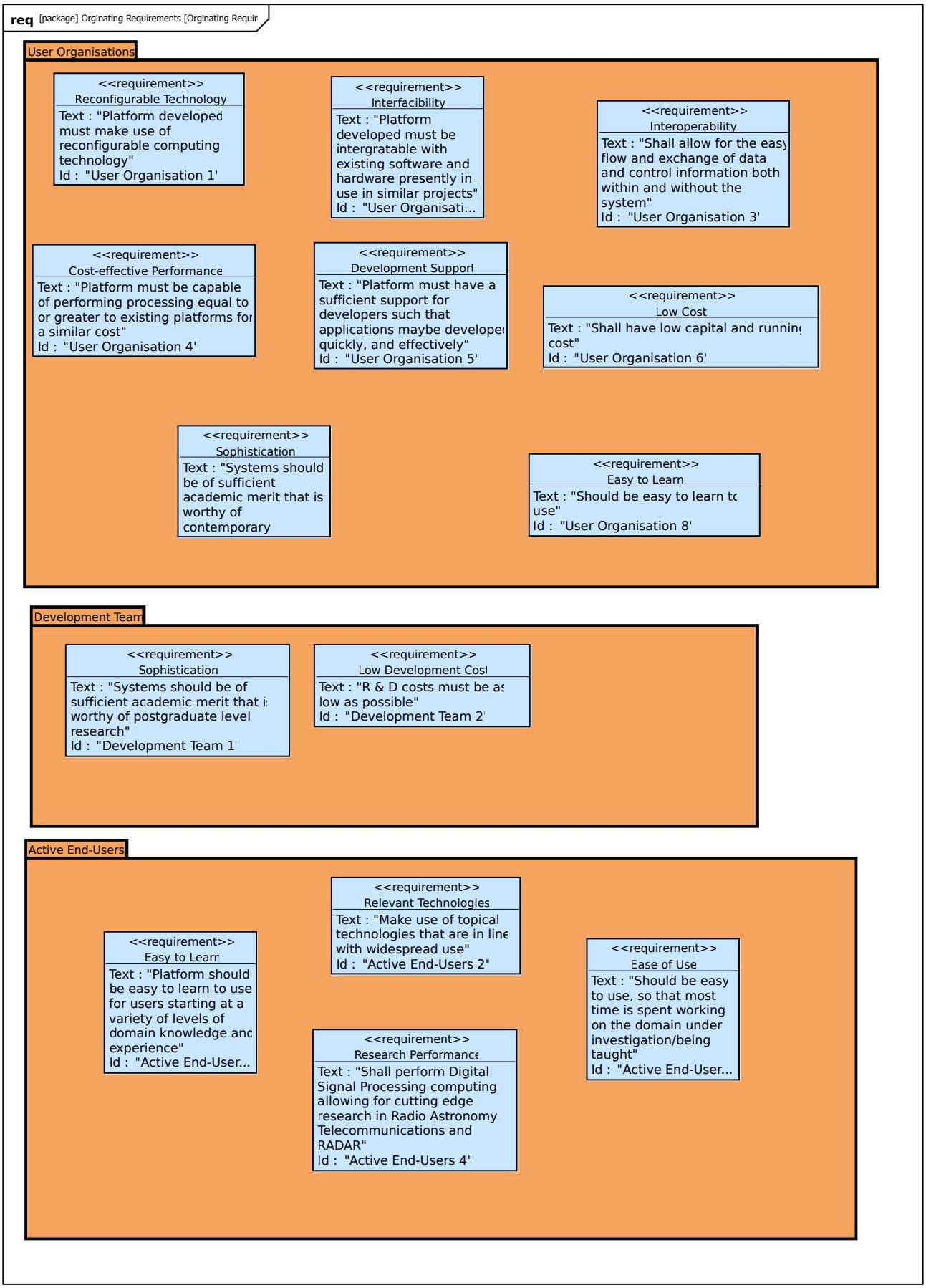

Figure A.4: Original stakeholder requirements for the Rhino Project

Figure 4 details the high-level originating system requirements for the three stakeholder groups identified above. These requirements were solicited from each group of stakeholders by means of requirements meetings and analysis of existing software defined radio platforms in use. Appendix A contains the derivation diagram for the requirements for each of the identified stakeholder groups[1]. In addition to the derivation, Appendix A also details the further expansion upon certain requirements.

All of these requirements represent one end of the coherent systems engineering process[18, 45, 44], meaning that every single one can be traced through the Rhino design to an eventual process which monitors and evaluates the systems effectiveness at meeting them.

Operational Concepts

As a candidate for further analysis with a view to informing design, the active end-user stakeholder group is an obvious choice. By describing the intended usage of the system by its actual users, many of the necessary functionality requirements for the system can be derived. In SysML, as in UML, this usage can represented using a Use Case Diagram[53]. However a major challenge in deriving the operational concept(s) was that the platform under consideration is a system intended for general use. To perform an exhaustive analysis of every possible usage scenario would be extremely time-consuming and the law of diminishing returns would certainly apply.

Rather three "extreme" operational concepts were captured from researchers in each of these fields at the University of Cape Town, in the domains of 4th generation telecommunications protocols, wide-band radar and radio astronomy. By considering these boundary cases of high end research, all other usage scenarios would fall within the performance of the system. The required information was gathered by engaging in further discussions with leading researchers within the identified domains, describing the types of computation required, as well as the performance targets.

[1] By in large this grouping process is merely a categorisation, several cases where stakeholders' requirements are similar/the same, and so could be derived into a requirement which satisfies both. In the case where a stakeholder requirement is simply carried through, the SysML <<copy>> relationship stereotype is used. In the case when several stakeholders' needs are conflated into one, the <<deriveReqt>> relationship stereotype is used[53].

	RADAR	Telecommunications	Radio Astronomy
Word Size	14 bit	14 bit	8 bit
Bandwidth	100 MHz - 500 MHz	20 MHz - 100 MHz	256 MHz - 12 GHz
Channels	8	8	1000 - 64000

Table A.1: Data Processing Requirements, derived from high-end research operational scenarios

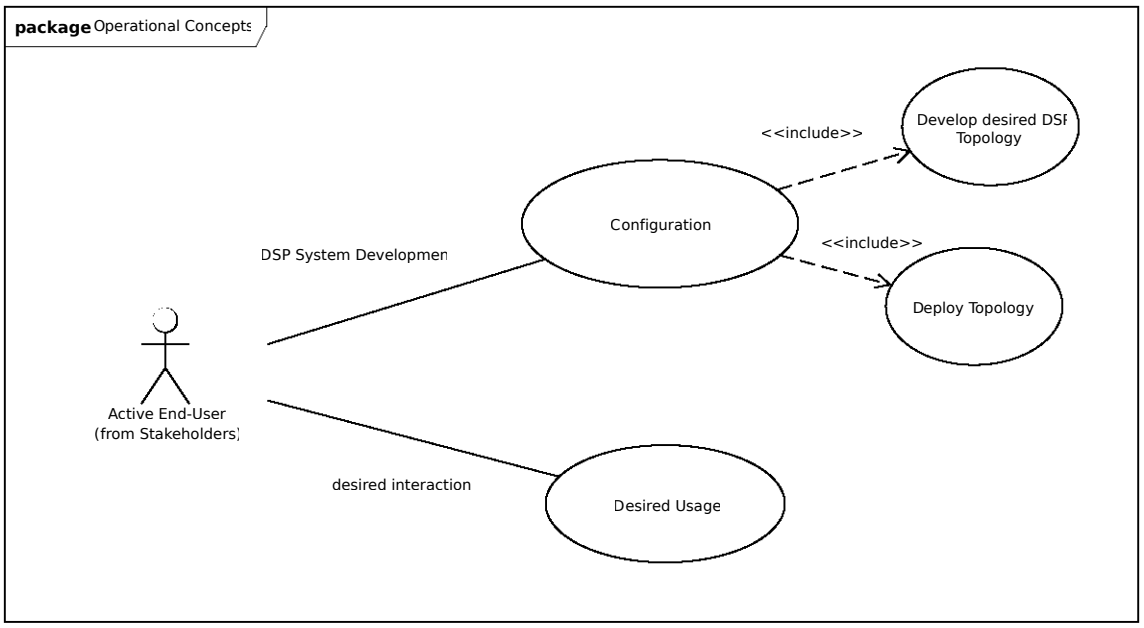

Figure A.5: Use Case diagrams for Rhino System

Figure 5 presents the overview of the two main usage scenarios envisaged:

1. Configuration - the user of the system describes to the platform the exact signal processing configuration desired.

2. Desired Usage - the user is making use of the configuration specified to process signal processing data. Software Defined Radio requires processing to be done upon a certain bandwidth within the electromagnetic spectrum. It also requires this processing to be done at a certain resolution, which translates into a processing word in the context of computation. A further consideration is that multiple sets of the bandwidth or channels need to be processed in parallel. See the table 1 for the high end requirements for the three target applications.

System Context

Analysis of the system context is another useful strategy when considering the requirements of the stakeholders, and attempting to map these requirements into a set of coherent system requirements. Its intended purpose is to explore the environment within which the system will function, especially with regards to identifying points of interaction, i.e. the necessary interfaces into the system[10, 53]. This leads to development of the system boundary, encapsulating all which the system is expected contain, within its context.

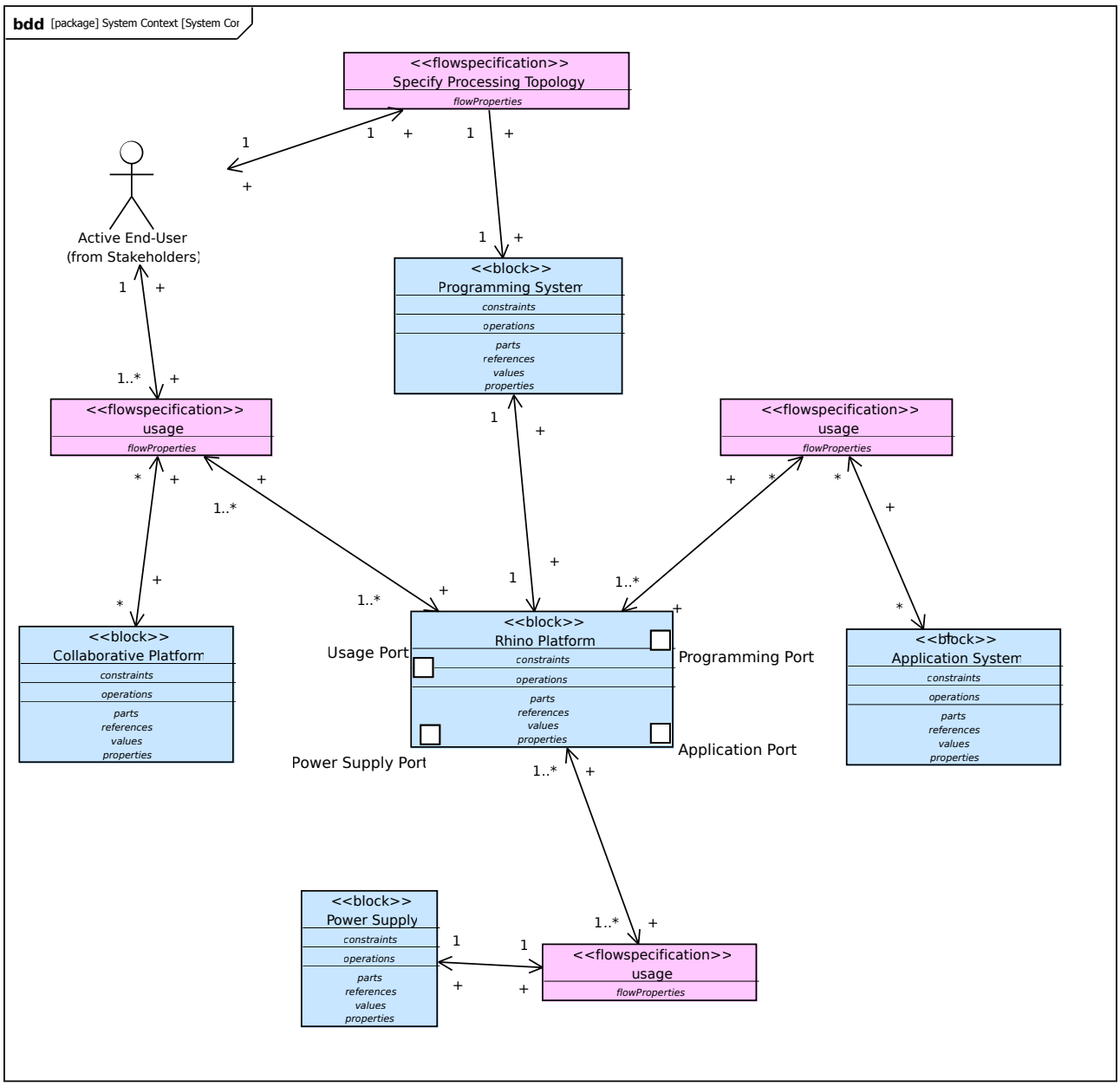

Figure A.6: System Context of the Rhino Project

The block definition diagram figure 6 is used to describe the Rhino system context. Systems,

including the Rhino platform, are represented using the <<block>> stereotype, a generalisation of the <<class>> stereotype from UML. Interactions are represented as <<flowspecification>> stereotypes, representing an interaction of some sort. Distinct, but related to these interactions, are the ports identified on the Rhino platform, which are a type of attribute[53].

As documented in the diagram, in addition to the end user, three adjacent, complicated systems are present. The point of interaction between these systems and Rhino will most likely comprise both a physical connection or interface of some sort, which will allow for the flow of data between the two system, as well as the supporting software infrastructure, which will control the flow of data between the systems.

These three systems are:

1. Programming System - the mechanism by which the end-user interacts with the Rhino System, so as to fulfil the configuration operational concept described above. The degree to which this system forms part of Rhino directly pertains to its accessibility and user interface.

2. Collaborative System - in many software defined radio systems, in fulfilling the data processing operational concept, typically multiple computational platforms are used. This has developed into an expectation on the part of the end-users that the system will not only be capable of cooperating with other computational systems, but also be able to do so effectively. It is entirely possible that another Rhino system could be a collaborative system in relation to another Rhino system.

3. Application System - software defined radio is a very broad application domain, and as a result a variety of systems will be required to be used along with the Rhino system to access the bandwidth within the electromagnetic section which the end-user desires to work with[31]. It is expected that these application systems will perform certain desired "front-end" operations, which will allow the Rhino System to be more generally applicable. The degree of modularity for the application system in conjunction with the Rhino system will directly pertain to the general applicability of the Rhino system.

System Requirements

The System Requirements represent the outcome of the Requirements Analysis phase of the Rhino project. Also referred to as the Requirements Baseline [10, 44, 45], it is the set of requirements to which the system design will attempt to meet, and to which it will be evaluated against. It may be considered as the requirements of the system, as described in terminology understood by the development team[53].

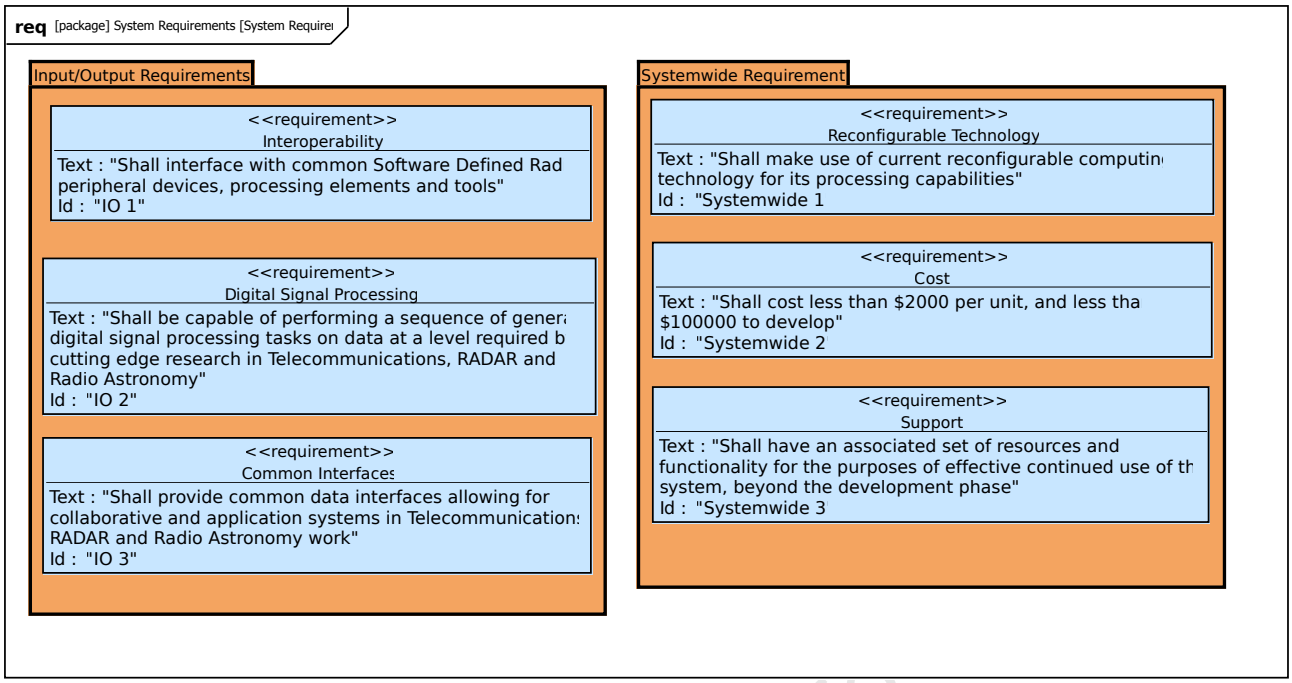

Figure A.7: Resulting System Objectives

The requirements have been grouped into Input/Output requirements which relate directly to the behaviour of the system with regards to data and other inputs, and Systemwide requirements which are those that apply to the system as a whole. Figure 7 presents these requirements in terms of these two categories. Tables 2 and 3 describe the mapping from the originating to system requirements, using an **X** to show where a <<deriveReqt>> relationship exists between two requirements. Of particular note, figure 8 describes the expansion of the high level data processing functional requirement, detailing this requirement in further detail, in terms of quantitative performance requirements.

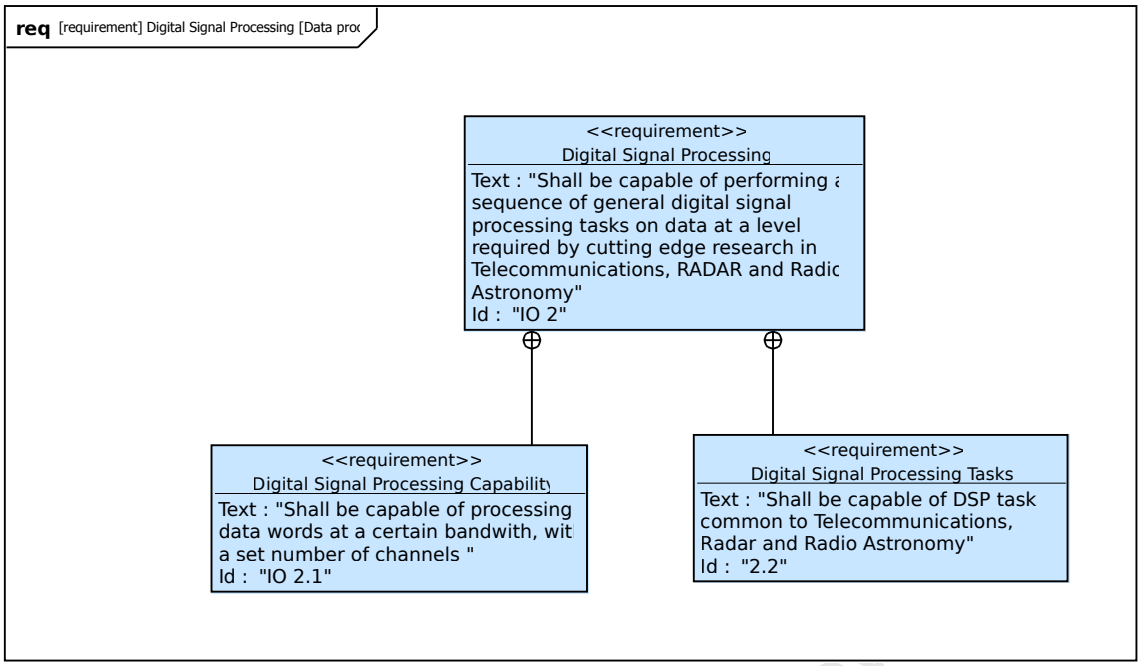

Figure A.8: Composition of System Processing Requirement

In Appendix A of this document, the derivation of these requirements may be found in full SysML, in terms of the originating requirements (making use of the <<deriveReqt>> relationship mentioned above). Input which is not represented in the figures below comes from the operational concepts and system context analysis, as well as further consultation and review of the requirements with the stakeholders in question.

Requirement ID		IO Requirement 1	IO Requirement 2	IO Requirement 3
	Requirement Name	Interoperability	Data processing	Common interfaces
User Organisation 2	Interfacibility	**X**		
User Organisation 3	Interoperability	**X**		**X**
Development Team 1	Sophistication		**X**	
End-Users 2	Relevant Technologies		**X**	**X**
End-Users 3	Ease of Use			
End-Users 4	Research Performance		**X**	**X**

Table A.3: Mapping of Originating Requirements to Input/Output System Requirements

Requirement ID		Systemwide 1	Systemwide 2	Systemwide 3	Systemwide 4
	Requirement Name	Reconfigurable Technology	Capital Cost	Development Cost	Support
User Organisation 1	Reconfigurable Technology	**X**			
User Organisation 5	Development Support				**X**
User Organisation 6	Low Cost		**X**		**X**
User Organisation 7	Sophistication	**X**			
User Organisation 8	Easy to Learn				**X**
Development Team 1	Sophistication	**X**			
Development Team 2	Low Development Cost			**X**	
End-Users 1	Easy to Learn				**X**
End-Users 2	Relevant Technologies	**X**			
End-Users 3	Ease of Use				**X**

Table A.4: Mapping of Originating Requirements to Systemwide System Requirements

System Design

Figure 9 presents the analytical processes undertaken in designing the Rhino System. The endpoint of this phase of the process is a clear description of the principal elements of the Rhino System and the functions provided by those subsystems, as well as the relationships or interfaces between those subsystems[10]. Again there also needs to be complete traceability, so that the functions and system elements can be traced back to requirements. These system elements may then be further developed and implemented, to the point of being capable of performing the functions allocated. These elements may then be integrated, and the aggregate result of their combination should meet the system requirements.

This process is a two stage one, with the first stage using the system requirements to suggest the conditional behaviours and physical groupings of elements of and within the system under consideration[10]. The mappings of certain system inputs to certain system outputs, as identified in the I/O requirements subset of the System Requirements are called the system functions[2], and it is the comprehensive description of these that comprise the functional architecture of the system. These functions are however informed in part by the elements performing them, and as such the parallel physical architecture is relevant. In parallel, the systemwide requirements, as well as the developing functional architecture suggest those elements that the system requires. This process is analogous to algebraic factorisation, in that the system requirements provide descriptions or formulations of the problem under consideration, and the process of elaborating the architectures helps reduce these descriptions down to more compact, atomic formulations.

The second stage is where the operational, or system architecture represents the marriage between the functional and physical architectures, bringing the entire design of the system together. This stage is crucial, as it contains all of the information pertaining to subsystems, and hence is essential for further development. To continue the algebraic analogy developed above, the system architecture provides the proposed solution to the formulations represented by the functional and physical architectures.

Functional and Physical Architectures

A decomposition strategy was largely used detailing both the functional and physical architectures[10], deducing subfunctions and behaviours from the high level functions and elements, as it is easier to do so at this general stage in the system conception. During the detailed design phase, a composition strategy is typically employed. Implementing multiple simpler functions or elements in order to realise the architectures described by this phase. This allows for intuition, creativity and experience on the part of the designer in creating the subsystems, while still ensuring traceability and reliability at the end point of the development process.

[2] if one considers the term input and output broadly enough, this definition is consistent with the mathematical definition of functions

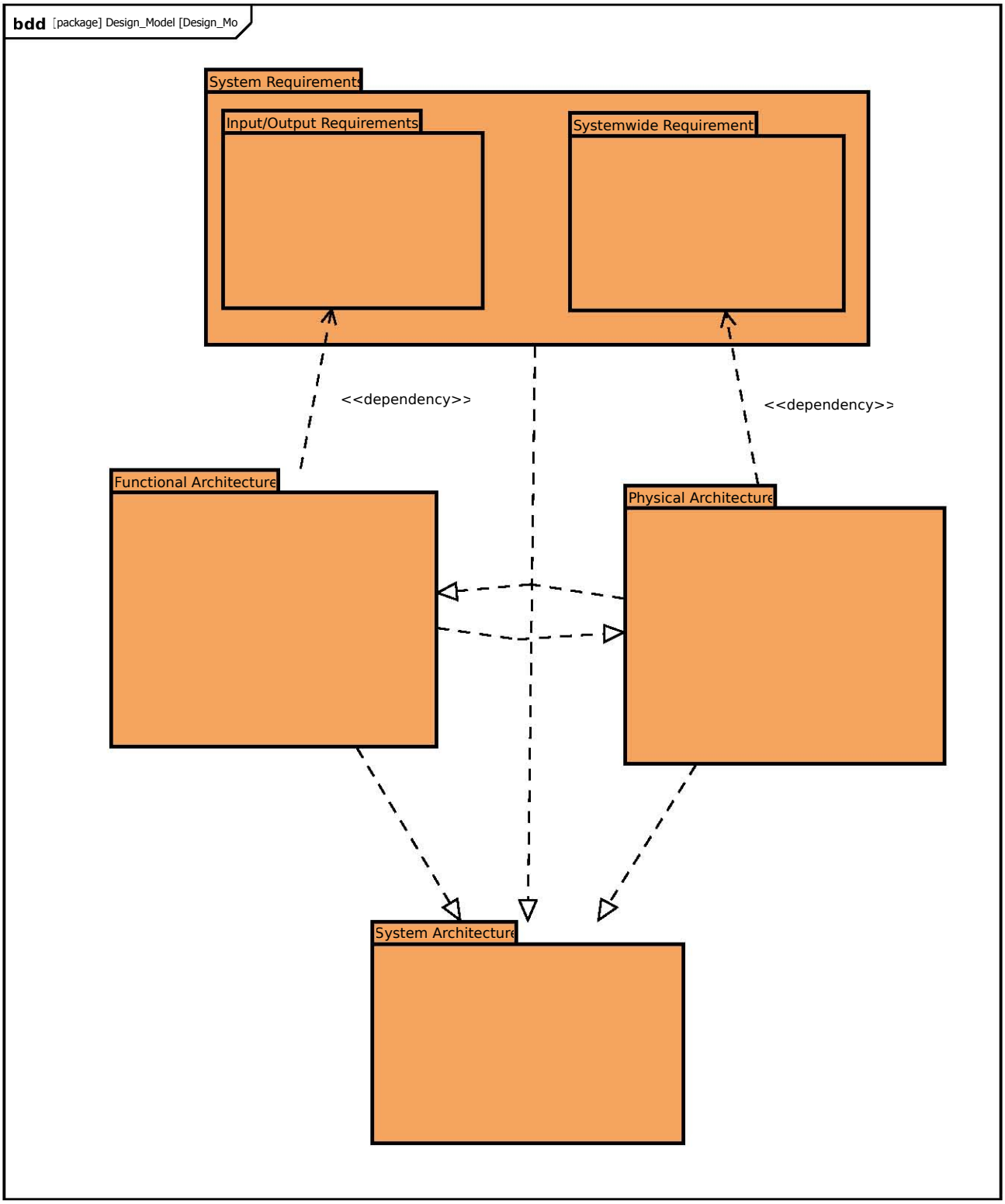

Figure A.9: Overview of Design phase of system architecture

Functional Architecture

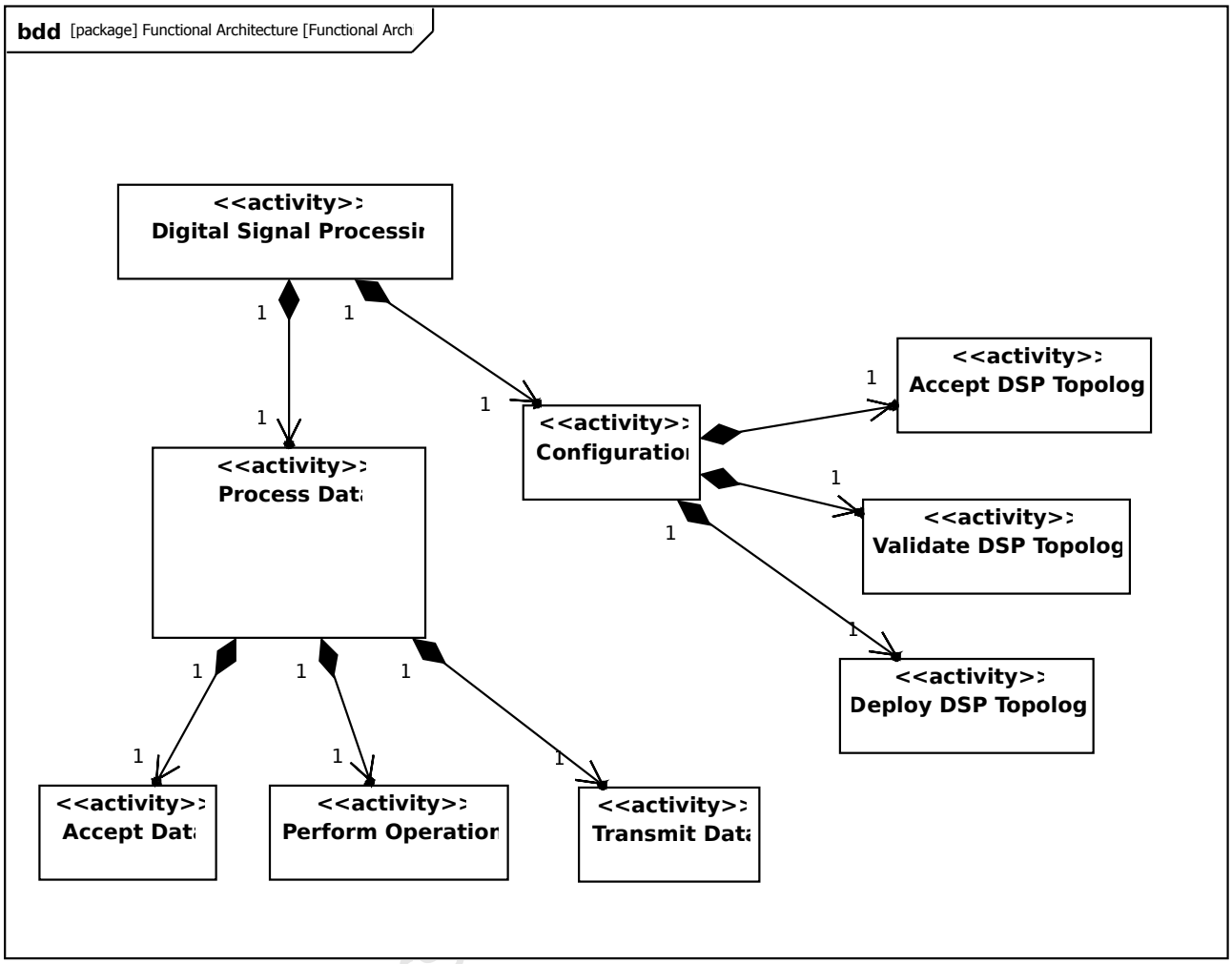

Figure A.10: Functional decomposition diagram of digital signal processing system activity

The functional architecture, as described above, is an analysis of what outputs must be produced, given certain inputs for the system. Functionality can be considered in terms of simple functions, mapping one single input to a single output, or it might be considered complete, mapping multiple coordinated inputs to multiple functions[10]. Another dimension to consider is the temporal - functions may be grouped into system modes, which are tied to a particular mode of the system. This is represented using several diagrams in SysML, with the block diagram providing the system modes, as in figure 10[53]. The sequencing of these activities is provided in subsequent SysML activity diagrams.

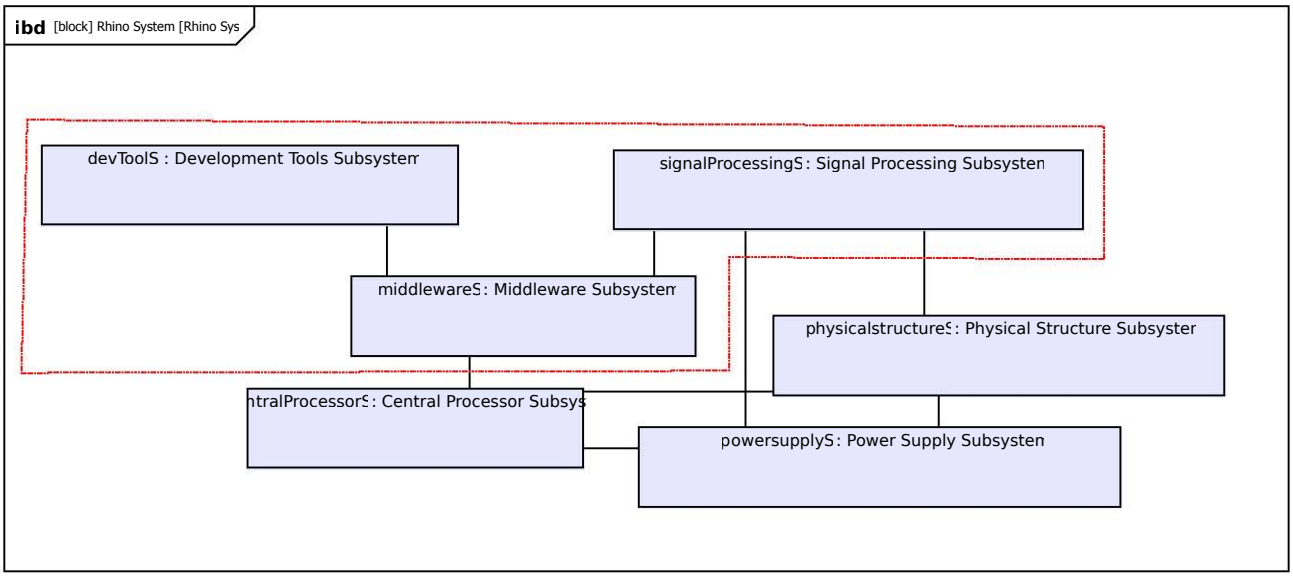

Figure A.12: Internal composition of Rhino system

Physical Architecture

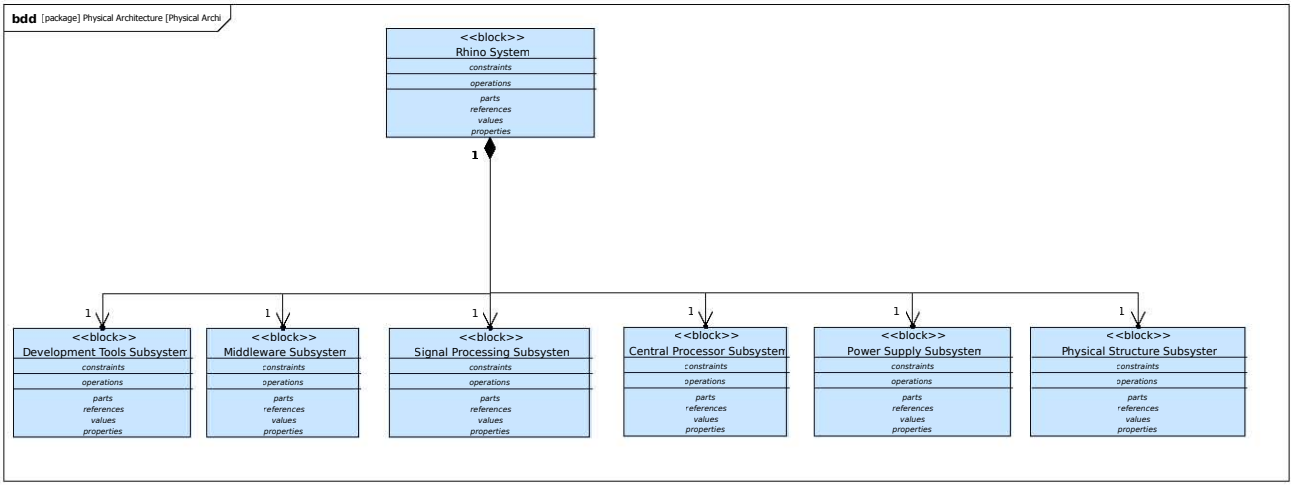

Figure A.11: Structural decomposition of Rhino physical architecture

If the functional architecture is what the system does, given certain inputs, the physical architecture is a description of what the system is comprised of, and the relationships between those components. A critical distinction is to be made between generic and instantiated architectural elements. Generic architectural elements are those described in general terms, such as broad function. The instantiated architectural elements outline the specific element for completing the task[10]. Figure 11 and 12 provide a hierarchical and interconnected descriptions of the generic subsystems of Rhino, making use of the SysML block definition diagrams and internal block diagram. At this stage in the analysis, it would not would not be possible to describe the instan-

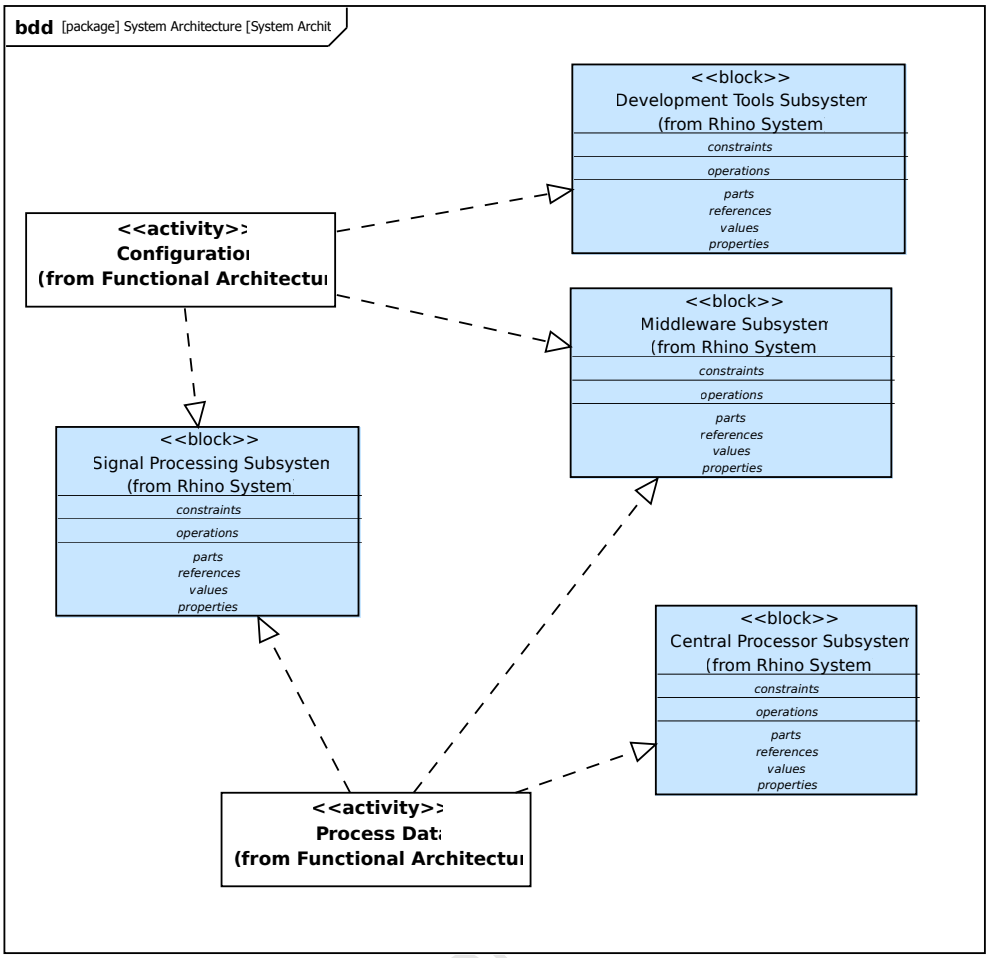

Figure A.13: Allocation of System functions to elements of Physical Architecture

tiated architectures, as it is the task of the detailed design phase of the development process to populate this system.

System Architecture

As described before, the system architecture marries the physical and functional architectures to the system requirements, as well as eachother[10]. The Input/Output requirements are mapped to the functions, while the Systemwide requirements are mapped to the physical architectures. Then functions described in the functional architecture are mapped to the subsystems described in the physical system. Thus the whole system is described in a coherent manner, in terms of its composite elements and its behaviours, which in turn may be traced back to the requirements[10]. Figure 13 describes the allocation process for the functions to subsystem elements, while figure 14 describes the allocation of the system requirements to the system elements.

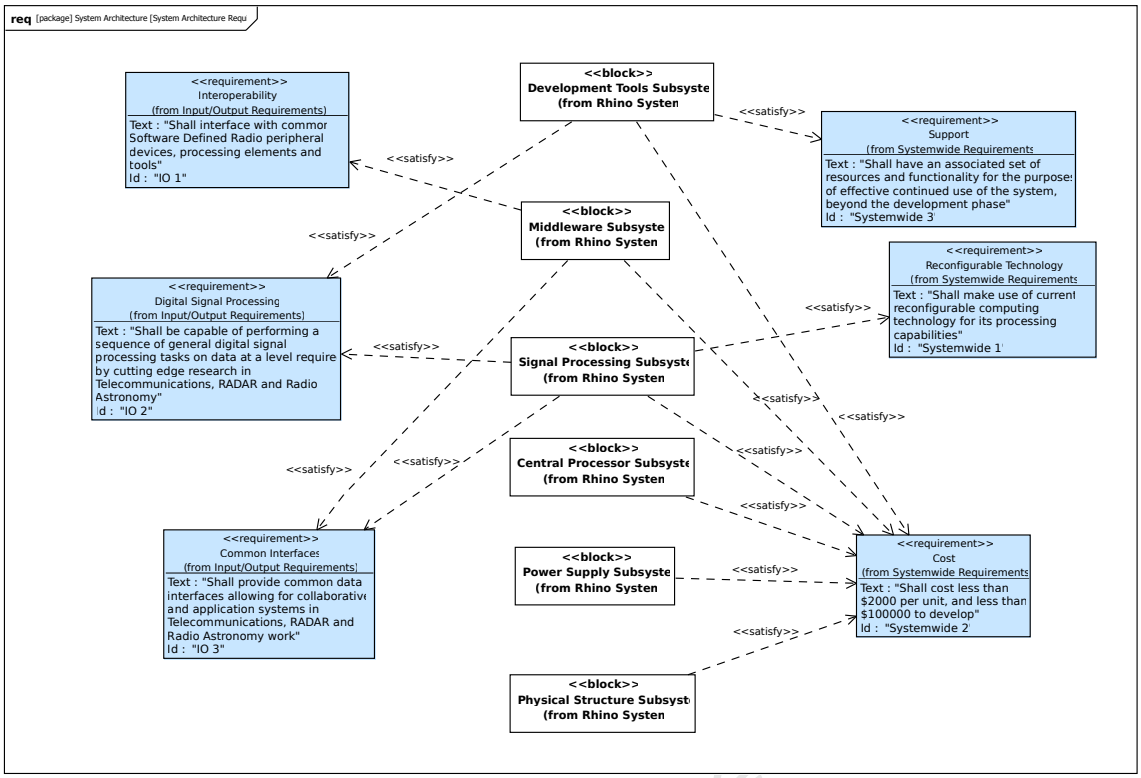

Figure A.14: Satisfaction of System Requirements by the System's Architecture

This tying together of the generalised system is a critical stage for the project. It is from this point that the various subsystems may be further analysed and implemented, so as to create the instantiated system architecture[10]. Furthermore the links between the various subsystems make it clear where defined interfaces are required, which is another detail of the implementation. Finally, it also enables performance measurements of the system to be performed, as with knowledge of the system architecture, the metrics may be decomposed and applied to each subsystem.

Detailed Design

This is the phase of development that the Rhino project is currently preoccupied with[24]. As such the efforts underway are described below. The various subsystems identified have been allocated to into three groups or self-contained projects for further development, as is to be expected during the practical management of the project.

Each of these groupings represent the starting point for the specification of the subsystem elements, which may be derived from using the allocation and traceability of the Systems Engineering Process. These groupings are:

1. The Rhino Platform - this includes the physical computational resources of the system, as well as the physical infrastructure required to support it, i.e. The Signal Processing and Central Processing subsystems, as well as the physical and power supply subsystems that support these elements. Also a significant number of interfaces are required to be implemented, as required by the system boundary and architecture.

2. The Operating System and Gateware Support packages - this includes all of the software and firmware required to support computation upon the Rhino platform, i.e. the static components of the middleware subsystem, as well as the central processing and development tools subsystems. In addition to supporting the control and operation of the identified activities, this grouping must also

3. The Software Defined Radio Toolflow - this includes the software and firmware tools that translate from an abstract signal processing description to one which may be implemented through the Rhino system. Primarily composed of the software development tools subsystem, it also will probably includes elements of the firmware.

The Rhino Platform

This project has largely been completed, and the designs are available[24]. In addition to a full design of the subsystem, it has been manufactured and tested.

Key instantiations of the subsystem architecture:

- The Digital Signal Processing subsystem - a Xilinx XC6SLX150T FPGA is the primary means of performing the desired digital signal processing, with the desired degree of flexibility[56]. 512MB of DDR3 RAM was implemented in support of the FPGA intended for signal processing, making use of the FPGA's built-in Memory Controllers. 2 FMC interconnection buses for accessing application systems and 2 10 Gbps Ethernet connections for connecting to collaborative systems.

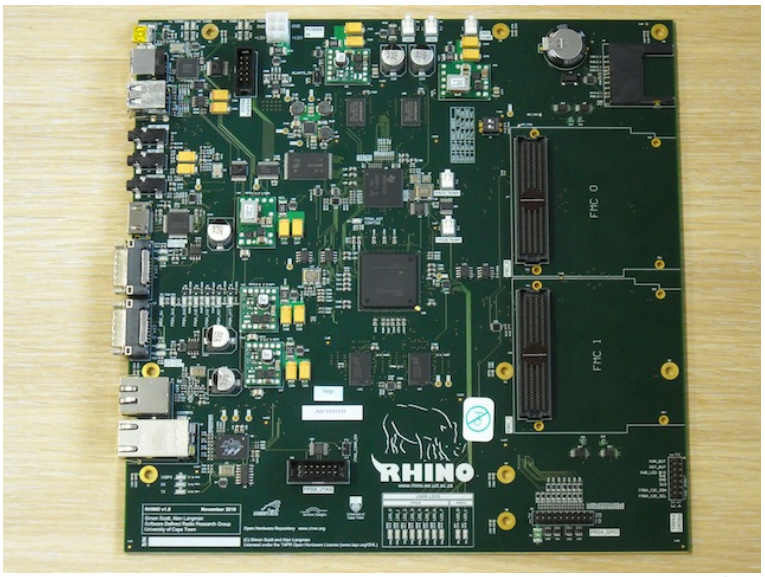

Figure A.15: Photo of completed Rhino Platform

- The Central Processing subsystem - a Texas Instrument Sitara ARM Cortex A8 processor was implemented to provide the command and control support subsystem[49]. This processor has 256 MB of permanent storage, as well as 256 MB of DDR2 RAM available to it. Additionally it supports USB (both host and UART), SD Cards, HDMI and Audio (in-/out) peripheral interfaces. 100 Mbps Ethernet support is provided and an interconnection bus with the FPGA.

Operating System and gateware support packages

The middleware subsystem is still largely under development, having required the hardware mostly to be in place, and functional before serious implementation could occur.

As suggested by the activities allocated to the relevant subsystems, the major design decisions taken:

- The use of BORPH Linux in managing the central processing system and the signal processing system[43]. As part of this effort, a fully functional communication bus between the central processing subsystem as well as the signal processing system is being implemented.
- Work is ongoing to provide full speed access to the FMC ports and 10 Gbps interface ports from the signal processing subsystem.

Software Defined Radio Toolflow

The software defined toolflow is currently under development. It is largely concerned with development of the tools that will assist the active end-users in using the Rhino system.

As suggested by the activities allocated to it:

- A toolflow must be implemented that allows the end user to configure the desired digital signal processing system upon the digital signal processing subsystem.
- This implemented signal processing system should behave in accordance with what the user has described.

Appendix A - Rhino Development SysML Representations

This appendix contains the details of those SysML diagrams from the Rhino development model not already presented above. The full system model is available as an interactive, HTML model may be found in the directory "systems_engineering_sysml_model". The top level of the diagram is in the file "index.html".

Appendix B - Additional Records of Systems Engineering

The following interviews and discussions occurred in relation to the systems engineering of the project:

- In January, 2010, an interview was conducted with Dr Yoann Paichard then of the Department of Electrical Engineering of the University of Cape Town, providing the wideband Radar operational scenario. Additional use cases were gathered from expertise within the Rhino group, particularly relating to 4th generation telecommunications and radio astronomy research communities.
- In July, 2010, Mr Paul Prozesky of the MeerKAT project presented on the Casper Signal Processing Toolset.
- In September, 2010, Mr Andrew Martens of the Centre for Astronomy Signal Processing and Electronics Research circulated an email entitled CASPER Toolflow MyHDL migration attempt, providing a suggested list of requirements for a software defined radio toolflow for radio astronomy.

Appendix B

Description of Software Source Code

This appendix contains the overview of the software source code from the two development prototypes implemented in chapter 5 and 6 of this dissertation. In addition to the title and name of each file, there is a brief explaination of that particular file's purpose.

B.1 Required Libraries

The following software libraries are required by the software tools below:

- SciPy - Scientific Computing Libraries for Pythonand others. Eric Jones [2]
- MyHDL - Hardware Description Libraries for PythonDecaluwe [16]
- MatPlotLib - Graphical Plotting Libraries for Python
- PySerial - Serial Port Libraries for Python

B.2 Low Pass Filter Source Code

B.2.1 Supporting and Analytical Tools

- Signal File Generator ("signal_file_gnerator.py") - a simple command line script for creating real-valued signal files for use in experimentation
- Waveform View Code ("time_signal_file_viewer.py") - another command line script for viewing real-valued signal files

- Serial Monitor Code (“serial_dump.py”) - another command line script for monitoring a particular serial port and storing the output as numbers of a specified data word size
- Error Analysis Code (“error_analysis.py”) - script for calculating and plotting the dynamic range of the filter implementation for a specified range of parameters

B.2.2 Arithmetic Models

- Blackman Low Pass Filter Model Class (“lowPassFilter_Model.py”) - executable class that represents a mathematical model of the Blackman Low Pass Filter. Includes parameters that allow fixed or floating pointing values to be used.

B.2.3 System Models

- System Model Code with nestled test benches (“low_pass_filter_myhdl_verilog_generator.py”) - MyHDL system model of the Low Pass Filter, comprising the filter system, as well as the inner and outer test benches, as described in Chapter 5.
- Example of converted Verilog code (“LP_Test_Bench.v”) - An example of the converted output from the file above.
- Xilinx Low Pass Filter ISE Project (“board_test_3.zip”) - this archive contains all the files used in the Xilinx ISE to program the SP605 Evaluation Board with the converted Verilog Code.

B.3 Hybrid Fourier Transform Source Code

B.3.1 Supporting and Analytical Tools

- Complex Signal File Generator (“complex_signal_file_generator.py”) - modified version of the signal file generator script used in Chapter 5, that creates complex versions of test waveforms.
- Frequency domain waveform viewer code (“frequency_specturm_viewer”) - simple script for viewing the magnitude of frequency domain waveforms, as produced by the HDFT algorithm
- Error Analyiss script (“dft_error_analyis.py”) - script for comparing error from fixed point algorithm implementation to that of a floating point FFT implementation.

B.3.2 Arithmetic Point Models

- Arithmetic model (“DFT_Model.py”) - executable class that contains sequential code for creating both fixed and floating point versions of the algorithm under consideration.

B.3.3 System Models

- Member class code (“Actor.py”, “DivideAndConquer.py”, “Mux.py”, “DFT.py”, “Unscrambler.py” and “HDFT.py”) - Baseclass, and member class for the toolflow library that implements the HDFT algorithm in MyHDL. The DFT class is notable for its overriding of the default conversion process.
- Module Development test bench (“mod_dev_testbench.py”) - script used as a MyHDL testbench to develop the modules used in the HDFT algorithm. It is capable of testing each module against the model method within it
- System Development test bench ()”system_dev_testbench.py”) - similar to the script above, except used for the HDFT module, which contains within it other modules.
- Nestled test benchs for verification and performance (“hdft_performance_testbench.py”) - similar to the verification test benches used previously, these testbenches were aimed at characterising the latency performance of the HDFT, and so implement modules to achieve this end.
- Example of Converted Verilog Code (“HDFT_Performance_Testbench.v”) - Verilog code for one of the performance evaluation experiment conducted in Chapter 6.

Appendix C

Experimental Results

C.1 Hybrid Discrete Fourier Prototype Experiment

C.1.1 Optimised FFT Core Performance

Radix Size	Transform Size	Logic Usage	Block RAM Usage	DSP Slices	Latency (μS)
2	256	1.45%	5.31%	11%	12.63
2	1024	1.49%	11.96%	11%	60.81
4	256	2.4%	10.63%	35.56%	9.63
4	1024	2.48%	42.51%	35.56%	39.43

Table C.1: Xilinx FFT IP Core resource vs latency performance statistics, for input bit width of 56 Bit on the Spartan 6 SLX150T FPGA

C.1.2 FPGA Performance Results

The performance of the prototype system is presented in Table C.2.. The metrics presented are similar to those presented for the Xilinx LogiCore library in Table C.1, but for the 128 input sample size used in this prototype system. Table C.3 includes the results from what is a reliable simulation model, using similiar parameters to those used in Table C.1 for the Xilinx LogicCore.

DFT	Input Sample Set Size	Logic Usage	DSP Slices	Latency (μS)
8	128	2.78%	13.33%	16
16	128	3.74%	26.67%	11
32	128	33.98%	53.33%	5

Table C.2: Resource and latency results by prototype system implemented upon the Rhino Platform

DFT	Input Size	DSP Slices	Latency (μS)
4	256	6.67%	15
8	256	13.33%	14
16	256	26.67%	13
32	256	53.33%	12
4	1024	6.67%	61
8	1024	13.33%	55
16	1024	26.67%	52
32	1024	53.33%	51

Table C.3: Resource and latency results from simulations of larger datasets

The throughput time are presented in table 4 and 5, as measured in microseconds, from a specialised performance measurement system implemented upon the Rhino system, as well as verified simulation models. The runtime is the time from the first value being input, to last value being output. The system clock was run at 50MHz for the prototype system, which is a considerably reduced rate, and possible far below the maximum that the system could perform at. The simulations were run with a clock rate of 125MHz.