Karel H.G. Walters

Versatile Architectures for Onboard Payload Signal Processing

VERSATILE ARCHITECTURES FOR ONBOARD PAYLOAD SIGNAL PROCESSING

KAREL H.G. WALTERS

Graduation committee

Prof. dr. ir. A.J. Mouthaan	University of Twente (chairman and secretary)
Prof. dr. ir. G.J.M. Smit	University of Twente (promotor)
Dr. ir. S.H. Gerez	University of Twente (assistant-promotor)
Prof. dr. ir. C.H. Slump	University of Twente
Prof. dr. ir. P.J.M. Havinga	University of Twente
Prof. dr. H. Corporaal	Eindhoven University of Technology
Dr. ir. G. K. Rauwerda	Recore Systems
Dr. R. Trautner	European Space Agency

Part of this thesis was carried out under a programme of and funded by the European Space Agency under contract no. 21986/08/NL/LvH Massively Parallel Processor Breadboard. The views expressed herein can in no way be taken to reflect the official opinion of the European Space Agency.

Computer Architecture for Embedded Systems Group
The Faculty of Electrical Engineering,
Mathematics and Computer Science
P.O. Box 217
7500 AE Enschede
The Netherlands.

CTIT Ph.D. Thesis Series No. 13-268
Centre for Telematics and Information Technology
P.O. Box 217
7500 AE Enschede
The Netherlands.

Cover design by Nymus3D - `www.nymus3d.nl`

Printed by Gildeprint Drukkerijen - The Netherlands
Typeset with LaTeX.

Electronic mail address: `k.h.g.walters@utwente.nl`
ISBN 978-90-365-0850-6
ISSN 1381-3617
DOI http://dx.doi.org/10.3990/1.9789036508506

VERSATILE ARCHITECTURES FOR ONBOARD PAYLOAD SIGNAL PROCESSING

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Wednesday, 9 October 2013 at 14:45

by

Karel Hubertus Gerardus Walters

born on 8th of December 1981,
in Naarden, The Netherlands

This dissertation is approved by

Prof. dr. ir. G.J.M Smit	University of Twente (promotor)
Dr. ir. S.H. Gerez	University of Twente (assistant-promotor)

Abstract

This thesis describes a system-on-chip (SoC) architecture for future space missions. The SoC market for deep-space missions develops slowly and is limited in features compared to the market of consumer electronics. Where consumers often cannot keep up with the features which are offered to them and sometimes even question the need for some of the options that electronics offer them, computer architectures for deep-space missions should fulfill different needs. Space is a harsh environment which requires SoCs to be shielded from radiation by hardening techniques. The missions often have a very long life-cycle: it can take more than fifteen years from the early planning stages to decommissioning of a satellite platform. The harsh environment, long lifetime and no possibility to change the hardware after launch together with the fact that mass and energy are constrained make architecture development for space more challenging than for consumer electronics.

The need for a new SoC for on-board payload processing is high. The reason is mainly due to the increased quantity of data from sensors. More sensors are mounted on board of satellites and the sensors themselves produce larger volumes of data. The bandwidth, however, to send this data back to Earth is limited. To cope with the increase in data and limitation of bandwidth, the satellite platforms need to have more processing power for distilling the information from the sensor data and compress this information sufficiently. Information distillation and compression are driving forces for algorithm development and these algorithms need to run on on-board processing platforms. Of course, algorithm development and platform development are related: algorithm development pushes platform development further while limitations of the platforms often hold back the capabilities of algorithms. This relation is explored in this thesis and from this, requirements in terms of processing power, scalability and power consumption are derived to develop a payload processing platform.

One topic of discussion related to processing platforms is often the need for hardware floating-point. This discussion is more prevalent for on-board payload processing where architecture developers want to keep the processing platform as lean as possible. Arguments in favor of small area, low power and low design complexity normally prevent inclusion of floating-point hardware

in space platforms. To overcome these arguments, part of the research reported in this thesis has focused on a light-weight floating point unit.

Another difficulty for European missions is the fact that most architectures are developed based around technology from the United States of America. The USA, however, has very strict export regulations regarding technology that potentially could be used for military purposes. Although the research done by the European Space Agency is strictly for civilian purposes, this needs to be proven each time that technology from the USA is procured.

This thesis presents a hardware fused-multiply-add floating point unit, called Sabrewing, which has properties that satisfy the needs for payload processing units. Sabrewing is BSD licensed and made in Europe such that it bypasses the export regulations of the USA. It combines floating-point and fixed-point operations such that the re-use of silicon area is increased. Fusing the multiply and add stage increases accuracy and processing speed. The number of pipeline stages is only three which makes it easier to cope with in compilers and makes the design itself less complex. The design itself is small, 0.03 mm^2 in low power ST65nm technology, and consumes very little power: 13mW at 200Mhz for floating-point multiply-add operations. The low-power ST65nm library characteristics transfer well to a library currently under development by ST which is completely radiation hardened.

In parallel to the development of the Sabrewing, a prototype platform has been developed to increase processing capabilities for on-board payload data processing. Development of this platform could not wait until the moment that Sabrewing became available as the evaluation of all other features of the platform as presented below could not be postponed. The platform is based on the trends which can also be observed in the more conventional SoC market. It consists of two VLIW processors, Xentiums, connected to a high speed network-on-chip (NoC), in combination with distributed memories, and high-speed interfaces. To satisfy requirements for space-based platforms, standard interfaces like Spacewire are connected directly to the NoC. The NoC also has direct interfaces to high-speed AD and DA converters. Since most software for space has not been developed with multi-tile and multi-cores in mind, the NoC is connected to a more conventional system, an AMBA bus. The bus system serves as an interconnect for a LEON2 processor to run legacy software and connects a real-time clock and other payload processing oriented interfaces. The whole system is prototyped on an FPGA and can be used to test and develop new algorithms, as such serving as a testbed to develop an ASIC for deep-space missions. Several benchmarks, representative for on-board processing, have been completely implemented and show that the new platform can serve as a starting point for further development.

So, a platform based on Xentiums turns out to satisfy many of the requirements set for future space missions. However, it obviously fails to meet an

important one: the incorporation of floating-point hardware. Therefore another testbed has been created, called Nectar. This testbed is based around a LEON2 that is directly connected to a modified Xentium processor which has a Sabrewing incorporated. Nectar shows that it is relatively easy to integrate Sabrewing in a modern VLIW processor and as such shows the feasibility of an on-board processing platform with hardware floating point capabilities. Furthermore, the trade-off between software floating point and hardware floating point has been evaluated in a complete system. This comparison shows that with a minimal increase in area, 15%, a decrease in power consumption of 98% can be achieved for floating-point operations. Since it is such an improvement with minimal effort and small area increase, the discussion to incorporate hardware floating becomes an easy one. The power and area downsides to hardware floating-point almost never outweigh its benefits anymore.

There is still research needed to fully satisfy the needs of an on-board payload processing platform. The final configuration of the NoC and its interfaces will depend on the requirements of the target missions and algorithms, but the scalability of the NoC makes it simpler to derive this configuration. The size of the NoC and configuration is more a constraint of the target ASIC technology than it is of the inherent capabilities of the system. Therefore, it is envisioned that future signal processing platforms, will have differentiated (VLIW) processors interconnected by a NoC of which most are suited for high speed, high throughput integer based processing and some are more suited to perform floating point operations supported by Sabrewing.

In short, this thesis shows insight into the architecture of future processing platforms such that scientists can get the most information out of an increasing amount of raw data originating from the numerous payloads for deep-space missions. The results of this thesis are used in a follow-up project of ESA in which a radiation-hardened version of a subset of the presented system-on-chip will be manufactured, intended for a deep-space mission.

Samenvatting

In dit proefschrift wordt een nieuw system-on-chip (SoC) beschreven bedoeld voor toekomstige ruimtemissies. De markt voor SoC's voor ruimtemissies ontwikkelt zich langzaam en is beperkt vergeleken met de markt voor consumentenelektronica. Consumenten kunnen amper de functionaliteit die de huidige elektronica hun biedt bijbenen, soms kan zich men zelfs afvragen of alle functionaliteit die geboden wordt wel vereist is. Computerarchitecturen die voor ruimtemissies bedoeld zijn moeten aan hele andere eisen voldoen. De ruimte is een zware werkomgeving voor SoC's, waar ze door speciale techniek beschermd moeten worden tegen straling. De missies hebben vaak een hele lange levenscyclus; het kan eenvoudig meer dan vijftien jaar duren van de eerste planningsfase tot de ingebruikname van een satelliet. De moeilijke werkomgeving, lange levensduur, geen enkele mogelijkheid om na lancering hardware te veranderen, samen met de restricties aan massa en energieverbruik maken de ontwikkeling van computerarchitecturen voor de ruimtevaart erg uitdagend.

De noodzaak voor een nieuwe SoC voor satelliet data-processing is hoog. De achterliggende reden hiervoor is de toename van de kwaliteit van de sensoren aan boord van de satelliet. Er bevinden zich meer en meer sensoren op de satelliet en de sensoren produceren zelf meer data. De bandbreedte om al deze data naar de aarde te sturen is gelimiteerd. Om toch zoveel mogelijk data naar de aarde te sturen wordt het steeds belangrijker dat in de satelliet veel informatie uit de beschikbare data gehaald wordt en dat deze informatie zo veel mogelijk gecomprimeerd wordt. De compressie en het distilleren van informatie uit de data zijn de grote aanjagers voor de ontwikkeling van nieuwe algoritmen die aan boord van de satellieten moeten werken. Algoritmeontwikkeling en platformontwikkeling gaan hand in hand: ontwikkelaars van algoritmen zullen pleiten voor de inzet van de nieuwste platformen, maar moeten zich uiteindelijk schikken naar de beperkingen van de keuzes gemaakt door platformontwikkelaars. In dit proefschrift wordt deze relatie onderzocht en worden hieruit vereisten gedestilleerd waaraan een nieuw platform zou moeten voldoen. Hierbij wordt onder andere gekeken naar functionaliteit, snelheid, schaalbaarheid en energieverbruik.

Een onderwerp van discussie bij de ontwikkeling van een nieuw te ontwik-

kelen platform is de noodzaak van floating-point hardware. De discussie is met name van belang bij de ontwikkeling van een platform voor een satelliet waar architectuurontwikkelaars het SoC zo klein mogelijk willen houden. De eisen van de ontwikkeling zijn onder andere: een zo klein mogelijk siliciumoppervlak, zo laag mogelijk energieverbruik en een zo simpel mogelijk ontwerp. Om aan de eisen van floating-point hardware tegemoet te komen wordt in dit proefschrift een nieuwe floating-point rekeneenheid voorgesteld.

Voor Europese ruimtevaartmissies is een ander groot probleem dat de meeste architecturen gebaseerd zijn op technologie afkomstig uit de Verenigde Staten. De Verenigde Staten hebben een zware regulering op de export van technologie die gebruikt zou kunnen worden voor militaire doeleinden. Ook al gebruikt de Europese Ruimtevaart Organisatie (ESA) technologie uitsluitend voor humanitaire doeleinden, moet dit elke keer aangetoond worden wanneer er nieuwe technologie gekocht wordt in de VS.

In dit proefschrift wordt een nieuwe rekeneenheid geïntroduceerd genaamd Sabrewing. Sabrewing biedt de mogelijkheid om een fused-multiply-add operatie uit te voeren en voldoet aan de eisen die gesteld worden aan data processing aan boord van een satelliet. Sabrewing valt onder de BSD licentie en is gemaakt in Europa waardoor het niet onder de exportrestricties van de VS valt. Sabrewing combineert floating- en fixed-point operaties waardoor het nuttig gebruik van het silicium oppervlak vergroot wordt. Door de vermenigvuldiging en optelling te combineren wordt de precisie vergroot en snelheid verhoogd. De diepte van de pipeline is maar drie 'stages', waardoor compilers er eenvoudiger mee om kunnen gaan en de complexiteit laag is. Het ontwerp is klein, 0.03 mm^2 in low power ST65nm technologie en verbruikt maar weinig vermogen, 13mW op 200Mhz voor fused-multiply-add operaties. De karakteristieken van de low-power ST65nm technologie vertalen goed naar een technologie t.b.v. inzet in de ruimte die momenteel bij ST in ontwikkeling is.

Tegelijkertijd met de ontwikkeling van de Sabrewing is er een prototype platform ontwikkeld om de processorcapaciteit aan boord van satellieten te vergroten. De ontwikkeling van dit platform kon niet wachten totdat de Sabrewing klaar was omdat het platform veel meer nieuwe kenmerken heeft die tijdig geëvalueerd moesten worden. Het platform is gebaseerd op de huidige trends van de meer conventionele SoC markt. Het platform bestaat uit twee VLIW processoren, Xentiums, die verbonden zijn via een network-on-chip (NoC), gecombineerd met gedistribueerd geheugen, en hogesnelheidsinterfaces. Om aan de eisen van een ruimtevaartplatform te voldoen zijn er ook interfaces als SpaceWire aanwezig, die direct verbonden zijn met het NoC. Een AD- en een DA-omzetter zijn ook direct verbonden aan het NoC. Vanwege het feit dat de meeste ruimtevaartsoftware ontwikkeld is in een periode waarin multi-tile en multi-core architecturen nog niet bestonden, is het NoC verbonden met een conventionele AMBA bus. De bus biedt ruimte aan een LEON2 processor waar

oudere software op kan draaien, een real-time klok en andere ruimtevaartgeörienteerde interfaces. Van het complete systeem is een FPGA-prototype gebouwd opdat er nieuwe en bestaande algoritmen op getest kunnen worden. Verschillende benchmarks zijn geïmplementeerd op het platform om aan te tonen dat het een goed startpunt is voor verdere ontwikkeling met uiteindelijk een ASIC als einddoel.

Het platform, gebaseerd op Xentiums, blijkt een goed startpunt te zijn dat aan de meeste eisen voor toekomstige ruimtemissies voldoet. Aan één van deze vereisten voldoet het huidige platform nog niet, en dat is de eis van 1 GFLOP. Om dit te bewerkstelligen is er een nieuw testplatform ontwikkeld, genaamd Nectar. In dit testplatform is een LEON2 direct verbonden met een aangepaste versie van de Xentium waarin een deel van de rekenhardware is vervangen door een Sabrewing. Nectar toont aan dat het relatief eenvoudig is om Sabrewing te integreren in een moderne VLIW processor en tevens dat integratie van floating-point hardware in een platform t.b.v. ruimtevaart haalbaar is. Mede met behulp van dit platform is ook de afweging tussen software en hardware floating-point geëvalueerd. Deze evaluatie laat zien dat met een minimale toename in siliciumoppervlakte, 15%, een afname van 98% energie kan worden verkregen voor een floating-point operatie. Aangezien deze verbetering verkregen kan worden met een minimale toename in oppervlakte en met een minimale inspanning in werk wordt de discussie om een dergelijke verbetering door te voeren een stuk eenvoudiger. De nadelen van hardware floating-point in termen van energie en oppervlak wegen op deze manier bijna nooit meer op tegen de voordelen.

Er is nog steeds onderzoek nodig om volledig aan alle vereisten te voldoen voor een platform aan boord van een satelliet. De configuratie van het platform in zijn uiteindelijke vorm hangt grotendeels van de vereisten van de missie af, maar de schaalbaarheid van het NoC maakt het eenvoudiger om op de gewenste configuratie uit te komen. De uiteindelijke maximale grootte van het NoC en de configuratie hangen meer af van de ASIC-technologie dan van de mogelijkheden van het systeem. Vanwege deze reden is te voorzien dat toekomstige signaalverwerkingsplatformen verschillende (VLIW) processoren aan boord hebben, die via een NoC zijn verbonden en waarvan de meeste processoren gebaseerd op integerhardware zijn en enkele op floating-point Sabrewing.

In het kort laat dit proefschrift zien hoe toekomstige hardware aan boord van satellieten er uit kan zien. Met deze hardware is het mogelijk voor onderzoekers om nog meer informatie te vergaren uit de ruwe data, die de verschillende sensoren, aan boord van satellieten voor verscheidene ruimtemissies opleveren. De resultaten van dit proefschrift worden inmiddels gebruikt in een project van ESA om een tegen straling beschermd signaalverwerkingsplatform te maken voor een toekomstige deep-space missie.

Dankwoord

Ten eerste wil ik graag Gerard bedanken die mij de mogelijkheid bood om te promoveren bij de vakgroep CAES. Hij heeft gedurende de hele periode ondersteuning geboden en soms ongevraagde, maar toch nodige kritiek. Hij is ook diegene die er voor gezorgd heeft dat ik een geweldige tijd kon doorbrengen in het centrum van de ruimtevaart in Europa bij ESA-ESTEC.

Sabih, die mij vooral in de laatste periode ontzettend heeft geholpen met het afronden van mijn promotie. Zijn verhelderende inzichten en commentaar, hielpen mij om het geheel toch af te ronden. Ik wil hem ook vooral bedanken voor zijn grote geduld met mij. Hij heeft ook grote invloed gehad om mijn voorliefde voor digitale hardware ontwerp via zijn onderwijs.

Het is ook tijdens dit onderwijs heb ik ook Tom leren kennen, die samen met andere studenten werkte aan een project dat ik begeleidde. Tom heeft tijdens zijn master onderzoek veel betekend voor mijn promotie waar ik hem voor wil bedanken. Ook hebben we menig uur aan de computer gezeten om problemen proberen op te lossen. Het lukte niet altijd maar vaak hebben we er toch veel plezier aan beleeft.

I would also like to thank Martin, Roland, Raffaele and Håkan for the great time I had during my stay at ESA-ESTEC. This was followed by the time I was part of the ESA project which Roland supervised. I want to thank him for the guidance and trust he put in a novice when it comes to ESA projects.

Sebastién who I got to know during the ESA project who helped me whenever I ran into problems with the platform. Ook wil ik graag Recore Systems als geheel bedanken voor het beschikbaar stellen van de Xentium voor mijn onderzoek. Graag wil ik Gerard bedanken voor zijn rol in het ESA project en de mogelijkheid om mijn onderzoek uit te voeren gedurende het project. Waar bedrijfsgeheim vaak voor een probleem zou kunnen zorgen, heeft Gerard er altijd voor gezorgd dat het zou moeten gaan om oplossingen en niet om extra problemen.

Mijn kamergenoten Maurice, Vincent en in het speciaal Albert die met mij, menig weekend heeft doorgebracht op de vakgroep. We hielden elkaar scherp door de nodige en soms keiharde humor waardoor het verblijf op de vakgroep verre van een sleur was.

Verder wil ik de hele vakgroep bedanken voor de leuke tijd tijdens koffie pauzes, vakgroep-uitjes, vrijdagmiddagen en om mijn cynisme een beetje in toom te houden. In het speciaal wil ik graag Thelma, Nicole en Marlous bedanken voor hun jaren lange ondersteuning en het luisterend oor dat ze menig promovendus bieden.

Natuurlijk bedank ik mijn ouders voor het feit dat ze me altijd de mogelijkheid en ondersteuning hebben geboden om te doen en laten wat ik wil. Mede door hen heb ik het via VWO, HBO, universiteit nu tot een promotie kunnen brengen. Mijn zusje die nu zelf promoveert en toch altijd een voorbeeld is geweest om het ook maar zo goed te kunnen als zij.

Vrienden, familie en huisgenoten hebben het vaak moeten ontzien gedurende mijn studie en promotie. Computers werden niet meer gerepareerd, aan huizen werd niet meer geklust, eten werd later gekookt en bij feestjes was ik vaak de afwezige. Mijn dank is dan ook groot voor hen die desondanks hun steun boden.

Allen bedankt,

Karel Walters

Contents

Chapter 1

Introduction

1.1 Computers in space

The fragility of human beings made exploration of space a field where electronics went first. It started with a series of pulses sent from the Russian Sputnik-1 satellite, which could be received all over the world. At the beginning of the space race, processing speeds were very low and data processing required huge mainframes. Early astronauts had to do calculations by hand and used slide rulers to help themselves out. The first moon landing almost failed because astronaut Buzz Aldrin wanted the landing computer to do two operations at the same time [Eyles, 2004]. Computers in space have come a long way since those days but still it is often difficult to comprehend for the general public why so much processing has to be done with so little outdated resources. Questions like, "Why has my phone more processing power than a Mars lander?" or "Why can't I directly have a live video feed of a fly-by of Jupiter?", often have to be explained in more detail to satisfy the general public.

Still, the processing done with on-board computers is primitive compared to what can be done with a current mobile phone. Small steps are being made because processing in space entails much more than just passing the sensor data to Earth for further processing. This thesis describes one such step: the architecture of a future platform for on-board processing.

1.2 Missions

There are roughly three different types of the space platforms:

- *Commercial*, these are the satellites that are used in daily communication. Television, internet and telephone are services that these satellites provide.

- *Military*, these are the 'observation' satellites and other orbiting platforms which are utilized by the military branches of some countries.
- The last and possibly most interesting class, are the *science*-based platforms and missions.

The science-based platforms deliver amazing pictures which inspire people to look further than their own backyard. Information like the weather, and the amount of sunscreen people need to put on are provided by these platforms. Images and information from the Moon, Mars and beyond are transmitted back to Earth and provide insight into our own existence and into our past. One day they might provide answers to the question whether we are the only intelligent life form in our universe.

The science missions can also be divided roughly into two different categories. Deep-space missions and Earth-orbiting missions. The reason for this subdivide, besides the platforms mission, has to do with energy availability, radiation and mass. Closer to Earth the Earth's magnetic field shields a lot of the radiation from the Sun and outer space. That is one of the reasons why the ISS (International Space Station) is in a low-earth orbit. The Earth is relatively close to the Sun which makes solar power a good source of power. Furthermore, an orbit closer to Earth makes it possible to have a larger mass of the platform with lower costs. When traveling further away from Earth, these three factors, mass, power and radiation, start to play an increasing role.

1.3 Sensors

The main reason why the amount of data to be transmitted keeps growing, are the developments in sensor technology. Cameras and other sensors are able to obtain increasingly detailed information from the outside world. This is great for scientists but difficult to deal with by engineers. The fact is that the downlink for data to Earth does not allow to send all data to Earth in real-time; the bandwidth is too small. Larger bandwidths would require more power on the platform and with it more mass. This is just not an option. For this reason, a selection on board of the satellite needs to take place on what data is required directly, how much compressions needs to be done, and what data may be delayed until there is bandwidth available.

1.4 Processing platform

A processing platform on board of a satellite has to take care of the collection of sensor data and has to send them back to Earth or another satellite. Currently these platforms only perform basic compression and transmit as much information as possible. Back on Earth the data is further processed and distilled into

usable information. Since the amount of sensor data increases and bandwidth to Earth practically does not, some of this processing needs to move to the satellite platform. Although the processing performance needs to increase, the demands on power, mass and radiation still hold.

Currently, processing platforms often consist of a general purpose processor in the 100Mhz range and a single or a few dedicated Application Specific Integrated Circuits (ASICs) which perform very specific tasks. Sometimes a Very Large Instruction Word (VLIW) processor is incorporated to do some more number crunching operations. Since the space environment is very demanding for the platforms and missions are relatively scarce, the platforms are often custom made. This means there is a wide variety of solutions available to suit the specific needs of a certain mission. Custom designs also make the missions very expensive and difficult to re-use for other missions because of the specific implementations. Scalable designs, which are more common in the consumer-electronics market, are often not done because each mission is very specific in nature.

Another difficulty with the designs is the long time that they need to be supported. From planning to launch takes several years and from launch to actually decommissioning of the platform can take more than a decade. Commercial-off-the-shelf hardware is just not built to be supported for those long periods and not up to the scrutiny that space hardware requires. Hardware revision changes can not be done when the craft is on the launch pad. Often, mass-market consumers are more than happy to change their mobile phone every other year to be up-to-date with current fashion. This is a luxury that the space science missions do not have.

1.5 Floating point

The hardware in on-board processing platforms often only supports integer operations. The reason for this is the fact that these are easier to implement, smaller in silicon area and more easily run at higher clock frequencies. Hardware floating point is more complex and requires more effort to run at higher clock frequencies. Hardware floating point has made it onto several processing platforms in space but rarely in the GigaFLOP range. Standards made this a little bit easier. The standards define how rounding should be done and what ranges of values need to be supported. When implemented completely, the hardware is often relatively large and power consuming. For these reasons, it is not that common to have high performance floating point hardware on-board of a satellite. The flexibility that hardware floating point offers, however, makes it highly desirable for algorithm developers. Having the support for hardware floating point saves development time since algorithms do not need to be converted to fixed point and integer operations. For this reason, there is a desire to

have number crunching VLIWs with support for hardware floating point.

1.6 Next generation

Part of the research presented in this thesis has been performed in close collaboration with the European Space Agency (ESA). ESA is actively exploring several design concepts to obtain a platform which can serve new missions over an extended period of time. Efforts are being made to increase processing speed allowing to deal with the demand from sensors and the bandwidth limitations. The preferred target is a design which can be scaled to the needs of a mission without building an entirely new platform. The platform should be capable of running algorithms developed after completion of the design. Reaching the one GFLOP range should be possible with this platform. Furthermore, it should have high-speed interfaces, and an easy to use development environment.

1.7 Problem statement

This thesis deals with a computing platform which is radiation hardened, can meet the performance requirements set by ESA and still keep cost, in terms of power and mass down. The platform needs also to have high-speed interfaces, should deal with legacy code, be scalable and usable in the future.

In particular this thesis tries to seek answers to the following questions:

- How should a future DSP platform for onboard processing look like, in view of current trends of multi-core architectures, massive parallelism and new NoC interconnects?
- How can a 1 GFLOP floating-point performance be achieved, given the constraints of deep-space missions?
- How can hardware floating-point be incorporated in a VLIW architecture?
- Can all requirements be combined with scalability?

1.8 Contributions

Part of this thesis has been performed within Massively Parallel Processing Breadboard (MPPB) study of ESA. This study is one of the endeavors towards a next generation on-board processing platform. Prototyped design and the implementation of the entire benchmarks done for this platform are the main contributions of Chapter 5.

Another part of this thesis deals with combined fixed-point and floating point hardware, which is named Sabrewing it is presented in Chapter 4. Sabrewing offers fused-multiply-add operations and many derivatives. It can handle the standard IEEE single precision format as well as an extended 41 bit format

as well as 32 bit fixed-point arithmetic. Four out of the five rounding modes that IEEE specifies can be used in both formats. The Sabrewing is small, low power and has only three pipeline stages. It is BSD licensed such that it can be used by many designers. In Chapter 6 of this thesis, it is demonstrated that it can be implemented relatively easily in a modern VLIW with minimal resource cost and effort.

The MPPB project and the design of the Sabrewing ALU show the potential for a scalable high-performance on-board processing platform.

1.9 Structure of this thesis

This thesis starts with describing the most commonly used number systems in digital hardware in Chapter 2. Some of the benefits and downsides are explained. An introduction as to why some number systems are easier to implement than others is also provided in that chapter. The thesis continues with outlining some of the existing payloads and architectures used in satellites in Chapter 3. The chapter concludes with the requirements that a next generation platform should satisfy. The mixed floating and fixed point core is described in Chapter 4. An evaluation is provided based on synthesis results. The prototype MPPB platform is introduced in Chapter 5. A combination of the MPPB platform and Sabrewing as well as its performance is presented in Chapter 6. The overall conclusions are finally listed in Chapter 7.

Chapter 2

Number Systems for Signal Processing

Abstract

The existence of different number systems in digital designs offers flexibility, but at the same time requires making decisions. They are about trade-offs between the required resources and the resolution. A high resolution and dynamic range is often a requirement from an algorithmic point of view while less resolution reduces resource consumption. This is especially important for space-related applications where mass and power have a direct influence on launch costs. This chapter provides insight into the different number systems that are commonly used in digital signal processing hardware.

2.1 Introduction

Development and evaluation of a new payload processing platform in the early stages often comes down to numbers. Computer architects will request numbers in terms of calculations per second and word lengths of operands from algorithm developers. These numbers then need to be matched to what is available in terms of silicon area and power budget. The amount of silicon area available will dictate how many Digital Signal Processors (DSPs) and Central Processing Units (CPUs) can be selected to meet the processing power requirement.

When a DSP is selected, there is often a clear first selection criteria. Should the DSP be able to handle floating point numbers in hardware or not. The reasoning behind this is that although floating point hardware increases flexibility, it also increases the overall resource demand of the DSP considerably. This chapter will explain some of the number systems that are used in signal processing and provide an insight into the resources they require. The following

chapter will then illustrate some current payloads and how they deal with the processing requirements as well as provide an outlook on some requirements for processing demands on future space missions.

2.2 Number systems

Before introducing the most common number systems in digital hardware, it is appropriate to list the common number systems that we are used to in mathematics.

- $\mathbb{N}$ for positive integer values e.g. $0, 1, 2...$
- $\mathbb{Z}$ for negative as well as positive integer values e.g. $... -2, 1, 0, 1, 2, ...$
- $\mathbb{Q}$ for fractions e.g. $\frac{n}{m}$ where m is not zero, the rational numbers
- $\mathbb{R}$ to express all numbers along a continuous number line, e.g. values as π and $\sqrt{2}$, the irrational numbers.

In the best case scenario, these numbers would have their identical representations in computer architectures. However computer architectures are constrained in the number of bits that can be used to represent numbers. Moving $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ and $\mathbb{R}$ to a constrained digital system is therefore impossible because of their infinite range and precision. In DSP architectures, the integers (see section 2.2.1) and fixed-point (see section 2.2.2) datatypes are mostly used. They can be considered approximations of $\mathbb{Z}$ and $\mathbb{Q}$ respectively. Moving $\mathbb{R}$ to a digital system is even more difficult to do efficiently: not only the range but also the infinite precision that is needed, is impossible to realize on a computer architecture. The floating-point number representation does not provide the full range and precision like $\mathbb{R}$ but does try to provide a practical implementation. We will show this in section 2.2.4. Any system used in digital signal processing is therefore an approximation of the number systems that we are used to in mathematics.

2.2.1 Integers

The simplest numbers that can be represented in a computer are integers. Throughout this thesis, we use the *two's complement* representation unless mentioned otherwise. Most if not all current processing elements use this as a common input and output representation. Internally sometimes deviations are used for various reasons, as shown in Chapter 4.

The number of bits or word length (WL) affects the range of numbers that can be represented. The more bits are used, the larger the *range*. Range is the term used to express the upper and lower limit of the numbers that can be expressed. In two's complement, a signed number is represented as:

$$\begin{aligned} \text{bit pattern} &= (b_{\text{WL}-1}\, b_{\text{WL}-2} \ldots b_1\, b_0) \\ N &= -2^{\text{WL}-1}\, b_{\text{WL}-1} + \sum_{k=0}^{\text{WL-2}} b_k 2^k \end{aligned} \tag{2.1}$$

The first bit of a two's complement representation determines the sign of the decimal representation. Therefore this bit is referred to as the *sign bit*, which is denoted as $b_{\text{WL}-1}$ in Equation (2.1).

As an example, 4 bits will provide a range of $[-2^3, 2^3 - 1]$ or $[-8, 7]$. Notice the inequality of the absolute value of the upper and lower bound. Adding a bit, in this case to 5 bits, will increase the range to $[-2^4, 2^4 - 1]$. This provides a practical implementation of numbers in the set $\mathbb{Z}$. To calculate the number of bits to represent a number Equation (2.2) can be used. Here α denotes the integer range and WL the word length.

$$\alpha = [-2^{\text{WL}-1}, 2^{\text{WL}-1} - 1] \tag{2.2}$$

Most DSPs operate on 16 and 32 bit numbers. A common addition to this are special registers in the DSP which can hold larger numbers with more bits e.g. 40 bit.

Note that representing a number $\mathbb{N}$ from an integer system with word length WL in a number systems with word length WL + k amount to prefixing the representation of $\mathbb{N}$ with k sign bits. This is called *sign-extension*. Reversly, if all numbers considered have k+1 identical sign bits, the k most significant bit can be removed without loss of information.

2.2.2 Signed Fixed point

Where in the previous sections the representation was limited to integers, the fixed-point representation provides the ability to represent fractions. The fixed point format is a popular format used in signal processing. It consists of an integer part with a length QI, and a fractional part, with length QF, such that the word length WL equals QI + QF. The fractional part, QF, affects the *resolution* ϵ that can be obtained:

$$\epsilon = \frac{1}{2^{\text{QF}}}$$

Resolution is a term used to express the precision of the fraction. Practically this determines the number of digits behind the fractional point, in this case the binary point.

Taking a small step back and using a similar representation as in the previous section we obtain:

$$\text{bit pattern} = (b_{\text{QI}-1}b_{\text{QI}-2}...b_1b_0.b_{-1}b_{-2}...b_{-QF})$$

$$N = -2^{\text{QI}-1}b_{\text{QI}-1} + \sum_{k=-\text{QF}}^{\text{QI}-2} b_k 2^k$$

As with the previous integer representation this representation is also two's complement in which the sign bit is now indicated by $b_{\text{QI}-1}$. The QI part (integer part) is split from the QF part (fractional part) by the *binary point*.

To know how many fractional bits are needed to obtain a certain resolution ϵ, we can use the following equation (notice the ceiling operation $\lceil\rceil$):

$$\text{QF} = \left\lceil \log_2\left(\frac{1}{\epsilon}\right)\right\rceil$$

If we then for example want to express a precision, ρ, of $\rho \leq 0.0001$ (decimal), the equation tells us that we need 14 bits as shown here:

$$\text{QF} = \left\lceil \log_2\left(\frac{1}{\rho}\right)\right\rceil$$

$$\text{QF} = \left\lceil \log_2\left(\frac{1}{0.0001}\right)\right\rceil$$

$$\text{QF} = \lceil \log_2(10000)\rceil = \lceil 13.288\rceil = 14$$

To express the fixed-point format, the notation < QI.QF >is used. For a 16-bit number in which 1 bit is used for the integer part and 15 for the fraction the notation is: < 1.15 >. Effectively the single integer bit is only used to indicate the sign of the fraction. Putting the integer range and fraction resolution together into one equation:

$$\alpha = \left[-2^{\text{QI}-1}, \left(2^{\text{QI}-1} - 2^{-\text{QF}}\right\rangle\right]$$

$$\text{resolution} = 2^{-\text{QF}}$$

The number of bits therefore equals WL = QI + QF. The fixed-point representation provides a practical implementation of a subset of $\mathbb{Q}$.

Figure 2.1 shows 100 equally spaced numbers in the −1 to 1 range in a < 1.4 > format. The limitations of this number system are clearly visible. The 100 equally space numbers are projected onto a fixed number format having a resolution of 0.02. Correct (lossless) mapping requires at least 6 bits in the fraction to represent correctly. As there are only 4 bits available, multiple numbers are mapped on the same bit-pattern. This is the *quantization* effect. Rounding explains why not all horizontal lines in Figure 2.1 have the same length.

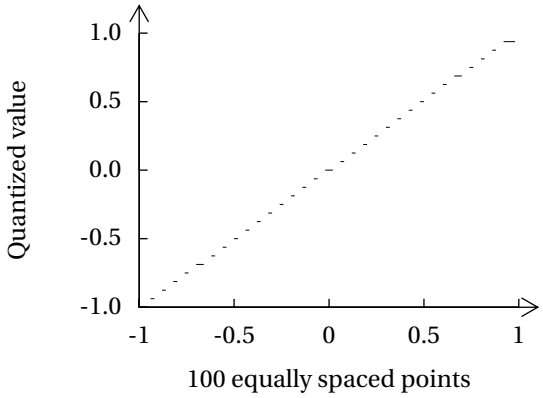


FIGURE 2.1 – *Fixed curve limitation* < 1.4 > *in the* −1 *to* 1 *range.*

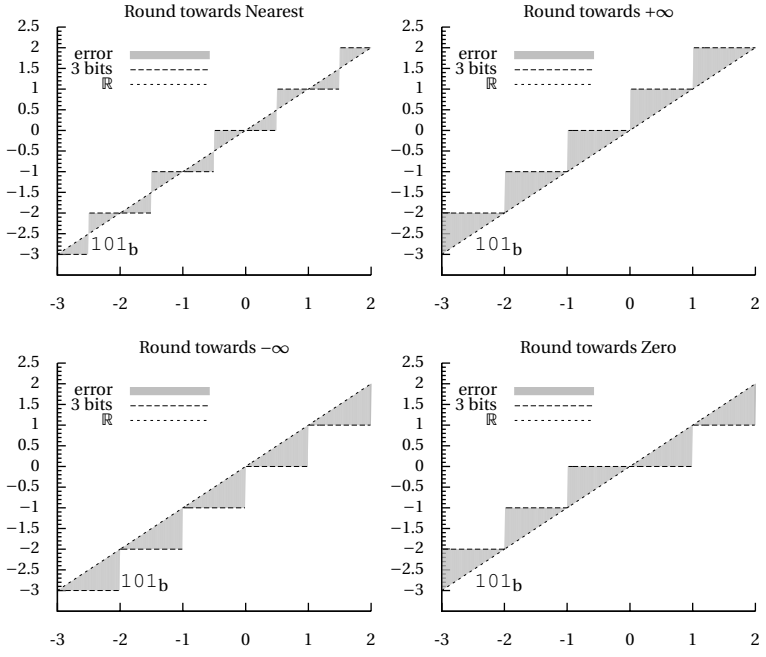


FIGURE 2.2 – *Quantization examples in a 3 bit system*

Quantization is the procedure in which a number is represented with less precision than the original representation and information is lost in the process. An example of quantization in the decimal system would be expressing 3.00 as 3.0. In this particular example there is no information loss and as such the quantization error is 0.00. However, if we quantize the number 3.05, to either 3.1 or 3.0 (depending on how the number is rounded), the quantization error is 0.05 or −0.05. Figure 2.2 shows examples of quantization in a 3 bit system. The diagonal line is a representation of $\mathbb{R}$ in which every number can be represented, in this case 500 equally spaced points in the −3 to 2 range. The horizontal lines are the numbers that can be represented using the 3 bits that are available. The gray area is the difference, quantization error, between the un-quantized number and the two's complement fixed-point number. Rounding is an operation that is applied to a number to achieve quantization. It requires a conditional addition or subtraction when implemented by a computer. Quantization that amounts to merely removing bits when reducing resolution is called *truncation*. Another form is taking into account what number is "nearest" and rounding towards that, which is what happens in Figure 2.2 top-left sub-figure. The other limitation of fixed point numbers, next to the quantization effect, is the same as with the integer system; the range is limited.

The question arises: Why would we use these fixed point systems? There are two main reasons. The hardware to implement basic operations like addition, subtraction and multiplication is not complex and it requires a small amount of resources compared to an implementation that is a closer approximation of $\mathbb{R}$.

2.2.3 Block floating point

To increase the range that can be expressed, with the same number of bits, other representations have been introduced. One of these systems is *block floating point* [Kalliojarvi and Astola, 1996] [Oppenheim, 1970]. The limitation of fixed point is the fact that the number of fractional bits is predetermined and as such fixed. Block floating point tries to alleviate this by keeping track of a value that represents the scaling factor for a certain number of fractions. The scaling is a power-of-two multiplication that effectively shifts the binary point. This allows the programmer to shift the binary point to allow for a larger and smaller fraction range at the cost of precision. The ability to store this value is not the only hardware feature that is required but one also needs the ability to count the leading sign bits efficiently in hardware. Counting the leading sign bits allows the programmer to obtain the scaling-factor and shift the value with the number of redundant sign bits. A number of DSPs provides this instruction. The C5000 (low power DSP series) from Texas Instruments [Texas Instruments, 2009], offers the EXP instruction which counts the leading zeros. The Blackfin series from Analog Devices [Analog Devices, 2012] offers the SIGNBIT instruction which

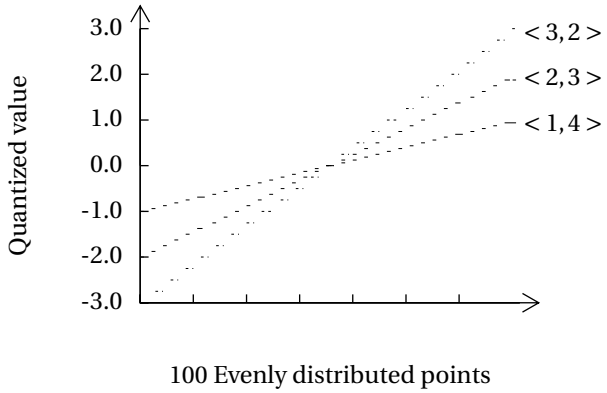


FIGURE 2.3 – *Block curve limitation for* 5 *bits in three ranges*

does the same thing. This means that these DSPs have the implicit possibility to work with block floating point although this is not always advertised as such.

Normally, in the fixed point format, all numbers would have a static format such as $< 1.4 >$ or $< 2.3 >$, but with the extra register, the location of the binary point can be varied. The register effectively holds the value with which the stored values needs to shift to get all the values aligned; one could call this value the scaling factor. The system makes it possible to have different locations of the binary point. There are some drawbacks to this system which are made clear in Figure 2.3. Figure 2.3 shows three different fixed-point formats number representations of 5 bits. The first "curve" is in a $< 2.3 >$ format, from $[-2,2)$, the register holds a 0. The second "curve" is in a $< 3.2 >$ format, from $[-3,3)$, the register holds a 1. The third "curve" is in a $< 1.4 >$ format, from $[-1,1)$, the register holds a -1.

Although the resolution is more flexible and the ranges improve a bit, the range is still fixed for the values that are in the same block. Another drawback which is less apparent, is the increased complexity of the source code in which this system is used. The software will need to keep track of the range of values in with the same exponent as well updating the register that holds the value of the exponent. The decision on the size of the block containing the values with the same exponent as well as the exponent value is completely in the hand of the programmer.

2.2.4 True Floating point

The last system we want to discuss is true *floating point*. This system is introduced to be able to approximate numbers from the set $\mathbb{R}$. In this representation the binary point has a flexible position, hence the term floating. It is almost the

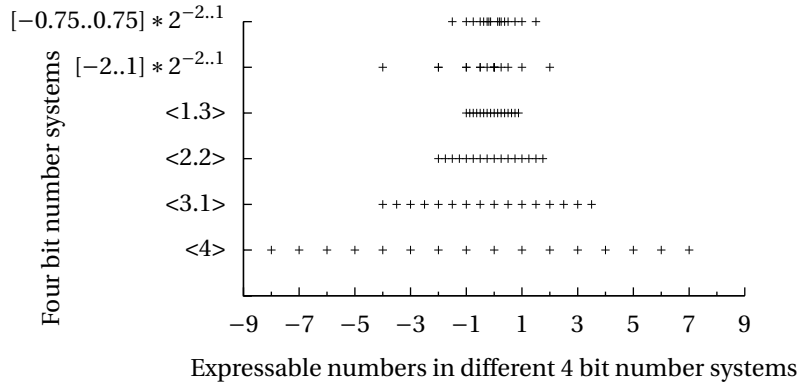


FIGURE 2.4 – *Different numbersystems with 4 bits.*

same as having block floating point in which the block only holds a single value. The register that holds the exponent is then included in the representation itself. As such this system holds the exponent as well as the fraction. In floating point the fraction is often referred to as *significand* or *mantissa*. Throughout this thesis we will refer to it as the significand. The combination of significand and exponent represents the following number:

$$\pm \mathrm{S} \times 2^{e}$$

Here S denotes the significand and e the exponent.

Since the number of bits in a word is fixed; one needs to decide how many bits are allocated to the exponent and how many to the significand. One also has different choice for the encoding of the exponent and significand (e.g. two's complicant or sign magnitude). This has been a discussion since the existence of floating point itself. Figure 2.4 shows the different possible representations of 4 bits that we have discussed sofar, as well as two floating point formats. The difference between the top two is that the first employs sign magnitude encoding for the significand and includes a so called *hidden bit*. The second uses two's complement for the significand as well as the exponent. Going further down, the three following formats are fixed-point formats which represent numbers from the $\mathbb{Q}$ set. The bottom representation encodes 4-bit integers (from $\mathbb{Z}$).

In sign-magnitude representation, there is a single bit that represents the sign of the value while the other bits represent the magnitude. This is different from two's complement in which the value after the sign-bit does not directly represent the magnitude, the difference is depicted in Table 2.1 The advantage of the sign-magnitude representation is that it is symmetrical while the two's complement representation is not. On the other hand sign-magnitude, has two

representations for zero whereas, two's complement has only one representation of zero.

TABLE 2.1 – *Sign-magnitude and two's complement representation with 3 bits.*

Decimal Representation	Sign-Magnitude Representation	Two's Complement Representation
−4	NA	100
−3	111	101
−2	110	110
−1	101	111
−0	100	000
+0	000	000
+1	001	001
+2	010	010
+3	011	011

For the top floating point representation in Figure 2.4, sign-magnitude is used for the significand part. Next to this a so called hidden bit used. The hidden bit is not part of the bit pattern since it is always set to 1. This is done to always have a so called *normalized number*.

Table 2.2 helps to explain why there is a need for normalization. The first column shows a five bit pattern in which the first two bits are the two's complement representation of the exponent and the other three bits are the sign-magnitude representation of the significand with a binary point in between. The second column shows the decimal value of this bit pattern. The third column shows the same bit pattern but with the introduction of the hidden third bit which normalizes the value, and the fourth column the decimal value corresponding with this pattern. The table shows that when the pattern is not normalized, different bit patterns represent the same decimal value which is not an efficient use of the available bits. When using a hidden bit, the significand is shifted to the left until a binary 1 is on the left most position and the exponent is modified accordingly. Since there is always a one on the left most position this bit can be discarded in the actual representation and as such become the hidden bit.

Table 2.3 depicts the floating point values of the top two rows of Figure 2.4. The first column is the decimal value that corresponds to the second column. The third column is the same binary representation as the second but now interpreted as a normalized sign-magnitude significand with a hidden bit. The fourth column is the decimal value that corresponds to the third column. What can be seen, is, that the normalization eradicates the duplicate values and makes the bit pattern more efficient.

TABLE 2.2 – *Normalization*

Not Normalized Bit Pattern	Decimal Representation	Normalized Bit Pattern	Decimal Representation
`10 0 .00`	$2^{-2} \times 0 = 0$	`10 0 .1 0`	$2^{-2} \times 0.5 = 0.125$
`11 0 .00`	$2^{-1} \times 0 = 0$	`11 0 .1 0`	$2^{-1} \times 0.5 = 0.25$
`00 1 .10`	$2^{0} \times -0.5 = -0.5$	`00 1 .1 0`	$2^{0} \times -0.5 = -0.5$
`01 1 .01`	$2^{1} \times -0.25 = -0.5$	`01 1 .1 1`	$2^{1} \times -0.75 = -0.75$

Still the question remains why sign-magnitude encoding is chosen for the significand. The normalization could indeed be done with two's complement but would be a bit more difficult. A second reason is the fact that the sign magnitude makes the range symmetrical. The third reason is most likely due to historical reasons, which probably has to do with the property that it is easier to visually see which binary representation is larger than the other.

TABLE 2.3 – *Two different floating point formats: two's complement exponent and significand (`ee SS`) and a normalized sign-magnitude with hidden bit (`ee s h S`)*

Decimal Representation	Two's Complement `ee SS`	Normalized Sign-Magnitude `ee s h S`	Decimal Representation
−4	`01 10`	`01 1.1 0`	−1
−2	`01 11`	`01 1.1 1`	−1.5
−2	`00 10`	`00 1.1 0`	−0.5
−1	`00 11`	`00 1.1 1`	−0.75
−1	`11 10`	`11 1.1 0`	−0.25
−0.5	`10 10`	`10 1.1 0`	−0.125
−0.5	`11 11`	`11 1.1 1`	−0.375
−0.25	`10 11`	`10 1.1 1`	−0.1875
0	`00 00`	`00 0.1 0`	0.5
0	`01 00`	`01 0.1 0`	1
0	`10 00`	`10 0.1 0`	0.125
0	`11 00`	`11 0.1 0`	0.25
0.25	`10 01`	`10 0.1 1`	0.1875
0.5	`11 01`	`11 0.1 1`	0.375
1	`00 01`	`00 0.1 1`	0.75
2	`01 01`	`01 0.1 1`	1.5

The first computer capable of true floating point operations was the Z1, a mechanical computer from 1938 developed by Konrad Zuse [Rojas, 1997]. So, already in those days Zuse saw the need to overcome the limitations of fixed-point representations. Interoperability between different architectures was less of a problem in that era but did become a problem after more manufacturers began to produce processing architectures incorporating floating point. Manufacturers such as IBM, HP, DEC and Cray all had their proprietary floating point

formats. IBM for instance used a base 16 representation for their exponent. Cray on the other hand decided that it was best to use at least 64 bits for floating point number to reduce the chance of overflow and underflow. This situation had to change to be able to easily exchange data between the different architectures. Several companies sat down and developed a standard which became the IEEE-754 standard and later the IEEE-754-2008 standard [IEEE Task P754, 2008] which includes some updates and a decimal format.

IEEE-754-2008

The IEEE-754-2008 standard deals with two floating point formats: the decimal and binary format. The decimal format is mainly used in the financial world where rounding errors are of greater concern than data efficiency. We will only consider the binary floating point format since we are mainly interested in compact architectures. The standard itself does not specify anything related to implementation. It describes how the operations on the floating point numbers should behave. For example it states "rounding should behave as if there is an infinite precision available". The standard does not describe how one should or can achieve this. The standard describes five basic binary formats as shown in Table 2.4. The first four formats have common word-lengths of 16,32,64 and 128. The fifth format in Table 2.4 is special. It provides the option for a custom format in which the relation between exponent and significand is fixed. In a few occasions, this is used by hardware manufacturers to allow for increased precision compared to the already defined formats.

TABLE 2.4 – *Segmentation of the different formats described by IEEE-754-2008*

Precision	**Significand** (+hidden-bit)	**Exponent** (bits)	**Bias**
Binary			
half (16-bit)	11	5	15
single (32-bit)	24	8	127
double (64-bit)	53	11	1023
quadruple (128-bit)	113	15	16383
custom (k-bit, k≥128)	k - rnd$(4\times log_2(k))+13$ *	rnd$(4\times log_2(k))-13$	$2^{(k-s-1)}$-1 **

* The rnd function rounds to the nearest integer
** s is the significand width.

The exponent is in a biased form, this means that for the single precision format a *bias* of 127 is added to the exponent before it is stored in the 8 bits. The actual range for these 8 bits is thus $0-255$ to obtain an exponent of -127 to 128. Zero and 255 represent special cases. There are two other terms related to a floating point number system: *ULP* and *machine epsilon*.

The ULP (Unit in the Last Place) is the maximum rounding error for a given exponent value. The ULP is the value that is presented by the last bit of the

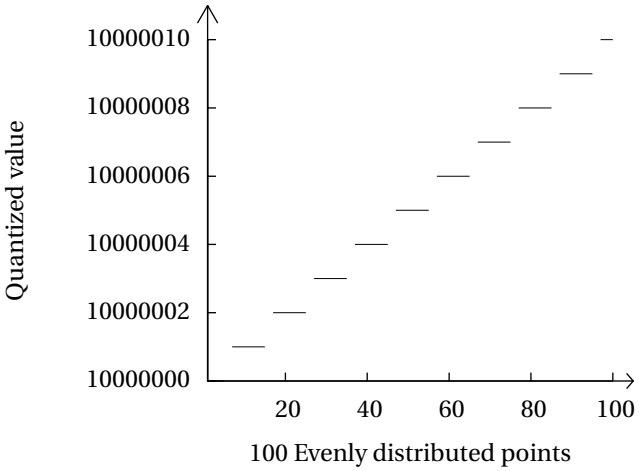


FIGURE 2.5 – *Float curve limitation ULP* ≈ 1

significand. The machine epsilon is a term coined to express the relative rounding error of a floating point number. Two values that have the same exponent and differ only the last bit of the significand are thus in 1 ULP distance of each other. The IEEE standard states that the result of an elementary mathematical operation should never be more than 0.5 ULP from the mathematically exact result.

The machine epsilon is not mentioned in the standard but is used to determine the upper bound on the relative error due to rounding. The machine epsilon for a single precision number is $\epsilon = 2^{-24} \approx 5.96046 * 10^{-8}$. This is the relative error. To determine the relative error the exponent value is set to zero. The number of significant bits in this case is 23 but the hidden bit increases this to 24 bits. Figure 2.6 and 2.5 illustrate the effect of the ULP. Both figures plot exactly the same number of points but in a different range. In Figure 2.5 from 10.000.000 to 10.000.010 and in Figure 2.6 in the range from 0 to 10. Ideally, both these lines should be straight to create the closest approximation to $\mathbb{R}$. We already know that this is not possible but the effects here make this explicitly clear. In Figure 2.5, the ULP represents a 1, while in Figure 2.6 it equals to approximately $8.9 * 10^{-16}$. The effects are clear: there is a trade-off in the precision and exponent size in floating point numbers. Consequently, problems may occur when using a number with a large exponent together with a small exponent number in the same mathematical operation, such as subtraction.

As already mentioned quantization introduces errors. To reduce the errors the IEEE-754 standard defines several rounding algorithms. The most popular is "rounding towards nearest, ties to even". This rounds towards the nearest value, if the number falls midway, it is rounded to the nearest even value. This mode

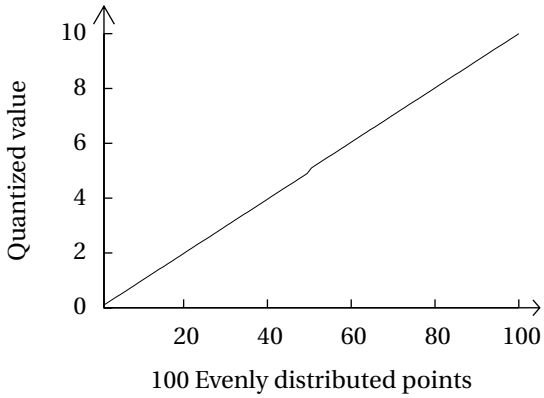


FIGURE 2.6 – *Float curve limitation ULP* $\approx 8.9 * 10^{-16}$

generally introduces the smallest error with arithmetic operations. There are four more rounding modes, depicted in Table 2.5. These modes are not always available in hardware or made easily available to the programmer. For example on most Intel architectures they are only available via the Rounding Control field (RC) in the x87 FPU control register (bits 10, 11) and MXCSR register (bits 13, 14) [Intel, 2011]. Also, the Intel architectures do not offer the "Round toward nearest, ties away from zero option", since this was added in a later version of the standard. Setting the FPU control and MXCSR register bits to other values than the default values affects the performance of the architecture, for example the pipeline flushing, which is also a reason it is not done often.

DSP manufacturers will often make statements about full IEEE-754 compliancy. Even when the hardware is fully compliant to the IEEE-754 standard, it does not necessarily mean that everything that the standard describes has been implemented. After closer inspection of the datasheet, one will conclude that multiplication, addition and some boolean logic is supported in hardware. Often this does not include all of the rounding modes. There is a logical explanation for this. In most signal processing algorithms, the most common operations are multiplication, addition and comparison. Other operations can be constructed from these operations in software. Next to that, most software developers are not aware or care about the different rounding possibilities and will use the default one, "round to nearest ties to even". Support for subnormal numbers is often omitted by DSP manufacturers. Lacking this feature makes computation less complex and the processing elements will occupy less resources. Subnormal numbers are often defaulted to zero.

The implementation of the IEE-754 floating point format in hardware requires a lot of hardware resources, area and power, compared to fixed point. This is not entirely surprising since the mathematical operations require dif-

TABLE 2.5 – *IEEE-754-2008 rounding modes*

Mode	Description
Round toward nearest, ties to even	Rounds toward the nearest value, if the number falls midway it is rounded to the nearest even value (LSB of 0)
Round toward nearest, ties away from zero	Rounds to the nearest value, if the number falls midway it is rounded to the nearest larger value (for positive numbers) or nearest smaller one (for negative numbers)
Round toward 0	Rounds toward zero (i.e., truncation)
Round toward $+\infty$	Rounds toward positive infinity
Round toward $-\infty$	Rounds toward negative infinity

ferent hardware for the different parts of the floating point operands. Addition for example requires the operand to be aligned first, then the significands can be added together and afterwards the result needs to be normalized again. Combined with updating the exponent to the correct value as well as rounding correctly increases the required resources significantly.

2.3 Conclusion

This chapter presented the most common number representations in current digital signal processors: integers, fixed-point, block floating-point, and floating-point. In practice, looking at common word lengths one finds often < 1.15 > , < 1.31 > in DSP devices. We have also shown block-floating point for cases where floating point is not available but where a larger range is desired. The main benefit of the block floating point format is the reduced amount of hardware resources compared to hardware floating point and still an increased available range compared to fixed point

We have shown that every representation has upsides and downsides and as such it is not always a clear-cut choice which representation to use. Based upon this chapter alone, one would most likely be most inclined to use as many bits as possible to get the best range an precision possible. In modern DSPs this would result in choosing IEEE-754 floating point or < 1.31 > fixed point if the range is limited. This is a fair choice but, unfortunately, precision and range are not the only factors for the choice for a number representation.

From the information in this chapter, it is already clear that more precision means more bits and that floating point is more expensive than fixed-point. The next chapters will provide more insight in hardware and power penalties involved and the trade-offs that are possible.

Chapter 3

Payload processing

Abstract

Payload processing deals with the processing of instrument data onboard of a satellite. This instrument data consists of data coming from cameras or other sensors that need to be processed before it can be sent to Earth. Processing consists mainly of compressing data or reducing the data rate so it can be transmitted via a relatively low bandwidth channel. As a consequence of developments in sensor implementations, data rate needs to be reduced more and more because the accuracy of sensors increases more than the power and bandwidth available for the transmission channels. Reducing the data rate can be done by compressing the data or discarding data which is not of interest. This chapter deals with current processing platforms as well as an example of a platform intended for a future planned mission to the icy moons of Jupiter.

3.1 Current payloads

There are many satellite launches and missions. *This chapter focuses on science missions managed by the European Space Agency, because part of the research was done in corporation with this agency.*

Current payloads can roughly be divided in two categories regarding requirements: Earth orbiting platforms and missions that travel further into space. In Earth orbiting missions, the requirements are less strict regarding radiation and often there is a higher bandwidth available to send data back which directly influences the processing requirements. The radiation tolerance requirement influences the possibility to use different kinds of hardware and packaging. Closer to the earth, the hardware is shielded with the help of the earth's magnetic field and allows for hardware like SRAM-based FPGAs to be used which are more sensitive to radiation. This reduces costs since radiation-hardened ASICs are

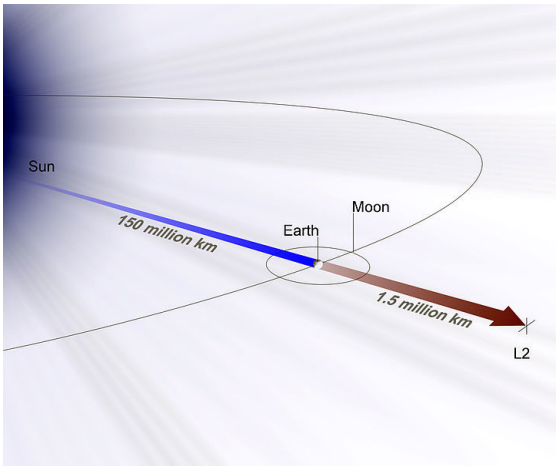


FIGURE 3.1 – *Lagrangian point 2, Image Source: NASA*

not widely available and the costs for new ASIC development is high, especially when qualified for deep space.

3.2 ITAR

Although politics should not be an issue in doing research, it unfortunately does play a role here. The processing elements that are described in the following sections might seem a bit odd to the observant reader. Why is there no Intel, TI, IBM or any other large silicon vendor on the list? This has to do with the export regulations to be observed by the USA, where most of these companies reside. In so-called International Traffic in Arms Regulations (ITAR) which makes it really difficult to export designs to be placed on satellites. Therefore companies that reside in Europe are the preferred choice for ESA to acquire processing elements.

3.3 Gaia

In the next two sections two examples of platforms for space mission are given, Gaia which is close to launch at the moment of writing this thesis and JUICE of which the launch is planned some decade later. The Gaia satellite is a platform made to create the most accurate three-dimensional star map to date. It is projected to launch in 2013 and put into an orbit at Lagrange point 2. This is a point 1.5 million kilometers from Earth opposite from the Sun as shown in Figure 3.1.

From the start of the project, it was already clear that the step in technology for ESA would be a large one to achieve the objectives of this mission.

The basic concept of the platform is a large CCD made out of 106 smaller CCDs which is similar in functionality to the ones used in consumer cameras. Each CCD contains 4500 by 1966 pixels and all 106 of them take up an area of 1 by 0.5 meter. The light of the stars is captured by the CCDs through a series of mirrors in the satellite after which the data is processed, stored and transmitted to Earth. The expected number of stars to be captured is around one billion, while the down-link is about 3-8 Megabits per second (Mb/s) depending on the weather. The captured raw data is around 7 Gigabits per second (Gbs). This means there is a need for a data reduction in the order of a factor 1000.

A nice feature of space photography is that there is a lot of black. The black in space does not contain any interesting information for this mission and is discarded, while only the relevant information is kept (feature detection). Discarding this data is where most of the data reduction comes from. Further data reduction is realized by a data compression algorithm that compresses the data before it is sent back to a ground station on Earth. These are the two main features of the processing platform: feature detection and data compression.

There are seven Video Processing Units (VPUs) on board of the Gaia spacecraft. Each of them contains two Actel RTAX2000 [Microsemi, 2013] anti-fuse FPGAs and an SCS 750 space qualified PowerPC [Maxwell, 2006]. These boards get the data directly from the CCDs and deal with the feature detection and compression of the data. The result is forwarded to the payload data handling unit of the spacecraft.

The Payload Data Handling Unit (PDHU) is responsible for packing of the data and its subsequent transmission. It also performs science data selection. Priorities are set for the features that are detected and, based on the current down-link capabilities, selections are made which features are discarded and which are sent to Earth. The PDHU is based on the LEON2 [Gaisler Research, 2005*b*] the fault tolerant processor.

The SCS 750 is indeed a product originating from the USA. It cost a lot of time to pass the ITAR regulations and be able to fly on-board. This signifies two things. There was not any part available in the European market to fulfill its job and the amount of extra time to get it on board was worth the trouble going through.

3.4 JUICE

JUICE is a mission planned to launch in 2022 and travel to the icy moons of Jupiter. It is currently in the early stages of development but there is a baseline for the instruments and mission objectives. Final decisions on the instruments and the processing platforms are expected to be made somewhere around 2015.

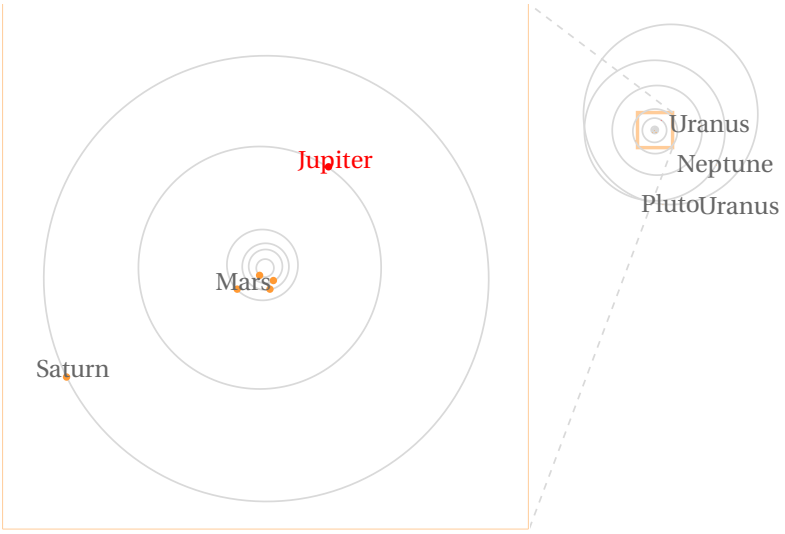


FIGURE 3.2 – *Jupiter orbit, Image source: Wolfram Alpha*

The spacecraft will take around 7.5 years to cross the ≈ 865 million km to arrive at Jupiter in the year 2030 after which the science mission of around 2.5 years starts. In this period, it will make flybys of the moons of Jupiter: Callisto, Europe and end up in orbit around Ganymede.

The following sections list the main instruments and provide an overview of the basic needs in terms of processing for the overall platform. As it currently stands, there are 11 instruments planned [ESA].

Narrow Angle Camera (NAC)

The narrow angle camera obtains high resolution and stereo images of the surfaces of the different moons and Jupiter's atmosphere. Currently a 1024 × 1024 CMOS APS sensor is the baseline. The stereo images are obtained by making two observations at a slightly different angle after each other. The average power consumption is projected at 12.7W, of which 7.7W is budgeted for the processing unit. The peak raw data rate is 5 images in 2 seconds at 36700 kbit/sec. The peak data rate compressed is (1:3) 12233.4 kbit/sec. The typical data volume of an image sequence is projected to be 24.47 Mbit in a compressed format.

Wide Angle Camera (WAC)

The Wide Angle Camera is similar to the NAC, but will operate longer and, because of the wide angle, does not have a high resolution. It contributes to the global shape mapping and multispectral mapping of the moons and in particular provides context images while in orbit around

Ganymede. Again the peak raw data rate is 5 images in 2 seconds at 36700 kbit/sec. The peak data rate compressed is (1:3) 12233.4 kbit/sec, which makes real-time transmission impossible. During its mission, it is estimated to produce around 30000 images yielding 160Gbit of raw data and compressed 20GByte data.

Laser Altimeter (LA)

The laser altimeter measures the height or elevation of the terrain underneath. This will create a 3D map of the terrain underneath the satellite. Two different units are envisioned for this instrument. The first is a transceiver unit with the instrument. The second is an electronics together with an FPGA to control the laser and electronics unit which houses the data processing units and main memory, which will also hold an FPGA to assist processing. The total data that this instrument gathers is over 2.5 gigabyte depending on the exact mission. The max raw data rate is 30Kb/sec while after compression this should be 15Kb/sec.

Magnetometer (MAG)

This instrument measures the magnetic field at a high resolution It supports two different modes: nominal and burst. The nominal produces 1700Kb/sec and the burst mode 6500Kb/s.

Ice Penetrating Radar (IPR)

The radar maps the subsurface features of Ganymede and parts of Callisto. Not much is known of the specifics of the implementation of this instrument as of this time. The projected data rate is to be 300kb/s.

Submm Wave Instrument (SWI)

The main objective of this instrument is to observe the stratosphere of Jupiter: looking at the winds, temperature as well as other characteristics. The total data is around 5 Gbyte per year, while the data rate is estimated at 11Kb/s.

UV Imaging spectrometer (UVIS)

The spectrometer is a camera with 512×512 pixels. It is meant to look at the composition of the different atmospheres. The total data volume is around 30 gigabyte a year while the datarate is 30kbytes/sec.

Visible Infrared Hyperspectral Imaging Spectrometer (VIRHIS)

This instrument looks at the surface composition in relation to geological and tectonic features. The instrument can operate at several modes and resolutions. The data rate ranges from 600Kbit/sec to 60Mbit/s. The very high data rate can only be used for a very short time due to storage and transmission limitations. The max burst data rate is 5Mbits/s and the

compression factor should be at least a factor of 5. The total data volume depends on the mission.

Particle Package Ion Neutral Mass Spectrometer (PP-INMS)
The instrument measures ions, electrons, energetic neutral atoms and neutral gas. The total data volume for this instrument is about 54 Gbyte, 20kbps for 4h each day for 4 years. The data rate is expected to be 2 to 25 kbps depending on the mode.

Table 3.1 shows the different instrument parameters which are known as of writing. The project has not been finalized yet and details can change. The table shows the number of bits that the sensor of the instrument operates on as well as the related data rate. Next to that it shows the expected data rates that can be used on the actual down-link to Earth. The weight and the power budget for the overall instrument, that includes operating the instrument as well as the power that would be needed to perform the processing can be found in the last columns.

TABLE 3.1 – *Projected Instrument specifications*

Instrument	Sensor (bits)	Raw data rate (Kb/s)	Compressed Data Rate (Kb/s)	Compression Ratio	Power (W)	Mass (Kg)
NAC	14	36700	12233	3	12.7	10
WAC	14	36700	12233	3	13	4.5
LA	10	30	30(15)	2	24	11
MAG	$3 \times 16 + 2$	2-7	2-7	TBD	2	1.8
IPR	N/A	300	300	TBD	20	10
SWI	8	11, 5GB/Year	10	1.25	48.5	9.7
UVIS	16	240, 40GB / Year	240	TBD	20	6.5
VIRHIS	14 and 16	600 to 60Mbit/s	5000	>5	20	17
PP-INMS	16	0.064 - 100	2-25	3	50	18.2

Most of these instruments are expected to have their own processing unit which compresses the data and formats it such that it can be sent back to Earth. It might seem that the overall data rate is not that high, certainly if compared to some of the devices one might have connected to a regular house computer. The data, however, all has to be sent via a link that can only transmit 1.4Gbit a day in an 8 hour window. This immediately makes it more difficult to plan which instrument can be used at what time and how long this data needs to be stored until there is time available on the link to send it back. This also means that the higher compression ratio one can obtain, the more information can be

sent back and the more efficient the instrument becomes. Compression and data reduction itself might not be such a problem but does become a problem if the hardware has to be radiation hardened and all has to work on about the 700 watts that the solar panels produce. As a comparison: a state-of-the-art personal gaming computer consumes about 1KW. Moreover, the 700W is not only for the instruments but also for the communication and control of the spacecraft itself.

3.5 Typical Algorithms

The previous section showed several instruments and most of the data processing on board revolves around data-rate reduction. The type of compression or reduction in data rate often depends on the characteristics of the data coming from the instrument. ESA has released a benchmark document which provides several typical algorithms used in payload processing [ESA/ESTEC, 2008*b*].

The first algorithms described in the document are several different size FFTs and FIRs. Later on in the document more complex benchmarks are introduced which revolve around two standardized algorithms for data compression of payload-oriented data. Straightforward data rate reduction often comes from a decimation filter which reduces the data by filtering and sub-sampling. Next some form of data compression is performed. Each of the main algorithms are described in the following sections.

3.5.1 Fast Fourier Transform

One of the first algorithms of the ESA benchmark is the Fast Fourier Transform. The Fast Fourier Transform[Cooley and Tukey, 1965] is an algorithm which implements the Fourier transform in an efficient way. Volumes have been written on the FFT: [Lyons, 2010] provides a nice introduction. It is used to transform from the time domain to the frequency domain and is often used as an analysis tool for any discrete sequence. The FFT can be implemented in many different ways but all methods are based on the so-called butterfly which is depicted in Figure 3.3. The name butterfly comes from the fact that the shape created by the arrows somewhat resembles the wings of a butterfly. Figure 3.3 depicts a radix-2 butterfly which shows up in algorithms that split their inputs into two. Other radices exist but two and four are the most common. The higher the radix the more complex the algorithm often becomes.

LISTING 3.1 – *complex multiplication*

```
A_real = (Wy_real*X_real)-(Wy_imag*X_imag);
A_imag = (W_imag*X_real)+(Wy_real*X_imag);
```

The operation itself is not that complex and consists of multiplications and additions. These operations can operate on complex values which in turn can

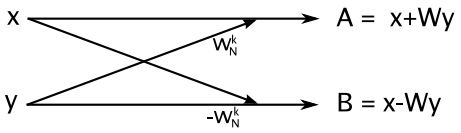


FIGURE 3.3 – *Radix-2 butterfly*

be floating-point data types as well fixed-point data types. Listing 3.1 shows the basic operations needed for a complex multiplication which is part of the butterfly shown in Figure 3.3. Most common are N-point FFTs in which N is a power of two. For these kinds of FFTs, the number of complex multiplications is $\left(\frac{N}{2}\right)\log_2 N$, (ignoring simplifications of multiplications by 1 and similar) and the number of complex additions is $N\log_2 N$.

3.5.2 Decimation filters and FIR filters

A common filter involved in space applications is called a decimation filter, a low-pass filter, which is also part of the ESA benchmark. There are many different configurations of filters, each having different properties. The selection depends on the characteristics of the input signal, the bandwidth, sample rate, and range. Although the overall design and configuration of the filter can be quite complex, the basic operation of a filter is often the same: a multiplication followed by an addition. The more multiplications and additions in series, the more so called *taps* such a filter has, since a tap corresponds directly to a multiply accumulate operation. The average filter size ranges somewhere from 16 to 256 taps in the benchmark of ESA. Multiple filters may be cascaded.

The factor by which a signal is multiplied is within a tap is the filter coefficient. These coefficients can be floating-point or fixed-point. They are often designed as floating-point and then a fixed-point approximation is made. The reason to go to fixed-point is often to overcome speed limitations of the available hardware but fixed-point conversion costs of extra development time. Listing 3.2 shows an example of a FIR filter. The datatypes here are integer but could just as well be float or double which are the equivalent of single precision and double precision floating-point in the IEEE 754 standard. Without any knowledge of the range of the input signal, `z[]`, and the behavior of the filter coefficients, `h[]`, it is difficult to know whether this algorithm will cause an overflow in the datatype. This is the exact reason why development is still often done in floating-point because it is easier and the developer has less to worry about overflow conditions.

LISTING 3.2 – *FIR example funtion*

```
int FIR(int  ntaps, int sum) {
   int  i;
```

```
   for (i=0; i<ntaps; i++)
      sum += h[i] * z[i];

   return (sum);
}
```

3.5.3 Lossless data compression

The Consultative Committee for Space Data Systems (CCSDS) has developed standards for space missions. The CCSDS was formed in 1982 by the major space agencies of the world to provide a forum for discussion of common problems in the development and operation of space data systems. Although not all payloads employ these standards, they do provide a baseline around which a lot of compression algorithms are developed. The most common ones are a lossless data compression standard [CCSDS, 2012] and an image data compression standard [CCSDS, 2005]. The following sections will describe these standards and explain what the overall main operations are that make up these algorithms. The lossless data compression standard of CCSDS consists of a preprocessor and an adaptive entropy encoder. The adaptive entropy encoder is basically a Rice encoder see Section 3.5.5. This requires many compare and bit-wise operations.

The preprocessor consists of a predictor and a mapper. The predictor tries to predict the value of the next sample in the sequence. Although many different predictors are allowed in the standard, the one that is promoted is quite simple. It uses the previous value as the predicted value. The difference between the previous value and the actual value is then used in the mapper. The prediction does not have to be compressed, because the same prediction can be made by the decompression stage. The difference between the predicted value and the actual value is most of the time smaller than the actual value which makes it a compression algorithm in the first place. The mapper maps the values from the predictor to non-negative integers suitable for the encoder to use.

3.5.4 Image data compression

The image data compression of CCSDS consists of a discrete wavelet transform (DWT) of the image and a subsequent encoder of the wavelet coefficients [Addison, 2002]. The wavelet decomposition of the image is performed by either two fixed-point filters or two floating-point filters. The floating-point-filter-based one results in lossy compression and the fixed-point based filter results in lossless compression. The floating-point filter however has better compression results. The whole image is filtered in the horizontal direction and down-sampled by a factor of 2. Then the whole image is filtered in the vertical direction and down-sampled again by a factor of 2. This process is then repeated two

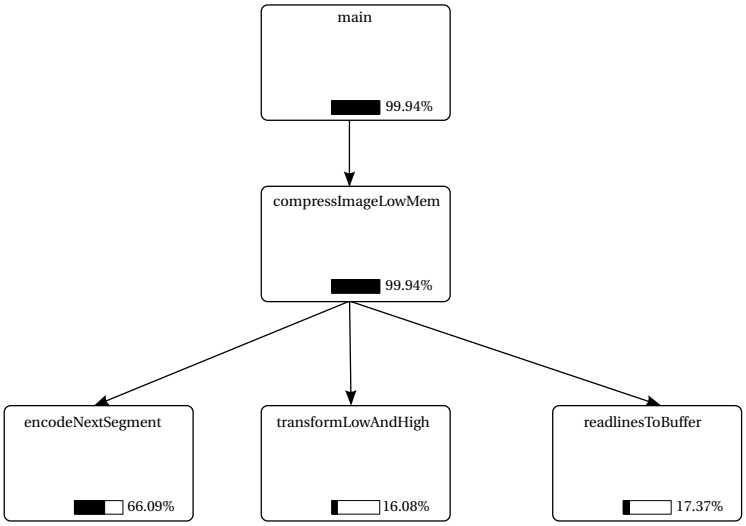


FIGURE 3.4 – *Image compression profiling information*

times more on the top left quarter of the resulting wavelet image. The resulting wavelet coefficients are then used by the encoder to produce a compressed data stream. The encoder consists of a Rice-based encoder which uses many compares, shifts and bitwise operations to produce the end result.

Figure 3.4 shows profiling information for a simple image being processed by the algorithm. This information was produced by running an implementation of the algorithm on a regular Intel architecture. The exact numbers are therefore not that representative for space qualified architectures but provides an indication which parts of the algorithm are expected to take up most of the processing time. The most interesting information in this image is the division of time required for the actual wavelet transformation, indicated by transformLowAndHigh, and the encoder, encodeNextSegment. What is interesting is that although the transform requires most of the arithmetic calculations, the bitwise operations and compares (Rice coding) are taking up the most of the processing time.

3.5.5 Hyperspectral image compression

A special kind of image is the so called hyper-spectral image. The images people are used to are either grey scale or color, exploiting the spectrum visible to a human being. Hyper-spectral and multi-spectral images are images produced by sensors that are sensitive to wavelengths outside the visible spectrum. On top of that, the term hyper comes from the fact that on a continuous spectral range there are a multitude of spectral bands for which a sensor might be sens-

itive. For example in a spectral band with a wavelength from 500nm to 700nm, a sensor might be sensitive to 20 bands that are each 10nm wide.

A standard has been developed, but this, as with many other standards, has taken a long time. There is a lot of correlation between the frequency bands and this can be exploited to achieve a better compression ratio as compared to compressing each band by itself. One of the candidates was from Abrardo et. al. [Abrardo et al., 2008] and it was based on the concept of distributed source coding. Without going into much detail one can say that it is built on the idea of estimating the next sensor value based on previous sensor values. This is not too difficult to understand as values in nearby frequency bands are not expected to change drastically. The estimation together with the difference in estimated value is then compressed into the data-stream to be sent back to Earth. The better the estimate, the smaller the difference between the actual value and the predicted value, and the better the compression. Although at first sight this algorithm looks simple, it turns out quite complex when it needs to be implemented on a fixed-point architecture [Walters et al., 2009]. It not only requires a complex addressing scheme for neighboring sensor values (pixels), it also requires an implementation of the Newton-Rhapson [Ypma, 1995] method to get the value of an estimator, because of the lack of hardware division in most architectures. Newton-Rhapson method is an iterative method to obtain roots of mathematical functions. It can be used to find a close approximation of the division result with multiplication operations and as such bypassing the need for a hardware division operation. Even though this is still all fixed-point, a floating-point implementation was suggested as well [Abrardo et al., 2008].

Although the algorithm described in [Abrardo et al., 2008] might prove to be superior in terms of compression ratio, one of the reasons the CCSDS committee discarded it was due to limitations in hardware capabilities at that time. Scientists might still want this standard implemented but due to hardware constraints it is difficult. This is also one of the reasons hardware needs to improve such that it is more future proof.

Rice Coding

Rice coding [Rice and Plaunt, 1971] is a special case of Golomb coding [Golomb, 1966]. The concept behind it is to represent a low value number with a small number of bits. Rice encoding can be explained as follows. Provided a constant M, and symbol S can be represented as a quotient Q and remainder R as follows:

$$S = Q \times M + R$$

If S is small, relative to M, then Q will also be small.

Rice Coding is designed to reduce the number of bits required to represent symbols where Q is small. Rice Coding represents Q as an unary value and R as a binary value. A unary representation is a representation whereby a number

of ones is followed by a zero to represent an integer. For example the integer 3 would be represented as `1110` and the number 5 by `111110`.

The following holds for binary representations, if $\log_2(M) = K$ where K is an integer:

```
Q = S >> K
R = S & (M-1)
```

R can be represented using K bits. Where >> indicates a shift right operation and & is a logical AND operation.

Then the encoder does the following:

```
S \& (M-1) //write to output
S >> K  //write to output in unary format.
```

As an example; encoding the 8-bit value $S = 18$ (`00010010`) when $K = 4$ ($M = 16$)

```
R = S & (M - 1)
//with S=18 and M=16
R= 18 & (16 - 1) = 00010010 & 1111 = 0010
Q = S >> K
//With S=18, K=4 and M=16
Q = 18 >> 4 = 00010010 >> 4 = 0001 //(10 in unary)
```

So the Rice encoded value is `10 0010`, saving 2 bits. This requires quite a number of bit operations for each symbol to write out, but achieves a significant compression ratio.

3.6 Architectures

The choice in hardware that have proven to work in the harsh conditions of deep-space is limited, certainly if ITAR regulations are taken into account. Therefore the payload architectures are built around architectures like the LEON2, Actel antifuse FPGA and the ADSP 201020, as seen in Section 3.3. Common architectures are the LEON2 [Gaisler Research, 2005*b*] for general purpose processing and the ADSP21020 for signal processing. A third architecture that is sometimes used is the Field Programmable Gate Array, (FPGA). Such a device provides a way to prototype an ASIC with programmable gates. The downside of a space-grade FPGA is that the size is very limited and the power consumption is fairly high when compared to an ASIC. The large available FPGAs are based on SRAM, which is very difficult to use in deep space due the fact that radiation causes single bits to flip which makes the platforms fairly sensitive to singe event upsets. Space-grade FPGAs are based on anti-fuse technology. Nevertheless, due to the fact that an FPGA is easy to use and the time to actual hardware is short, it provides a nice way to develop hardware for special purposes against a relative small budget.

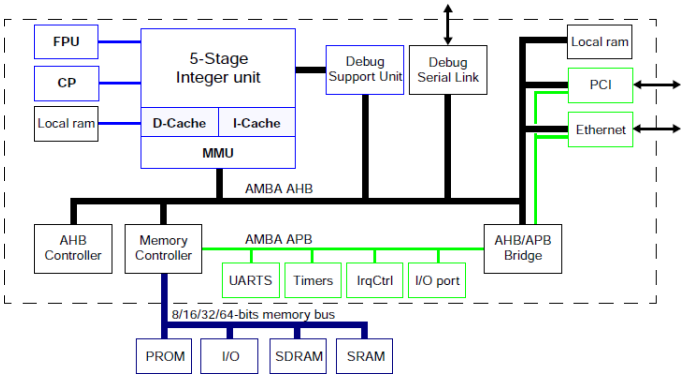


FIGURE 3.5 – *Architecture of the LEON2*

3.6.1 LEON2

The LEON2 [Gaisler Research, 2005*b*] processor is a 32-bit synthesisable general purpose processor core based on the SPARC V8 architecture [SPARC, 1991]. The LEON2 processor was designed under contract from ESA by Gaisler in 1999. Small updates have been made and the current version is 1.0.9.16.2 which was released in September 2009. ESA still licenses the IP for projects. Several succors to the design have been created by Gaisler Research, now Aeroflex Gaisler. These include the LEON3 and LEON4 and a fault tolerant version of the LEON3. Altough licensing of the LEON2 has been done via ESA the support is mainly done by Aeroflex Gaisler. The LEON2 is not under direct support by Gaisler any more. Gaisler actively encourages developers to move to the LEON3 or even LEON4 designs because these designs are better maintained. The reason the LEON2 is still used in designs is the fact that the license for it is easily obtained via ESA and free, the LEON3 and LEON4 are licensed via Aeroflex Gaisler with costs attached.

The LEON2 has the option to be extended with a dedicated floating-point core called Floating Point Unit (FPU), which can be seen in Figure 3.5. The floating-point core has also been developed by Gaisler and comes in two different versions. Either with full support in hardware for addition, multiplication, division, and square root or a lite version which only includes addition and multiplication. Both versions support single and double precision IEEE-754 floating-point.

A radiation-hardened version of the LEON2 is available from Atmel as AT697 [Atmel, 2011]. This implementation runs up to 100Mhz and achieves 86 Million Instructions Per Second (MIPS). It does come with a dedicated floating-point unit which is capable of achieving 23 Mega-FLoating-point-operations Per Second (MFLOPS).

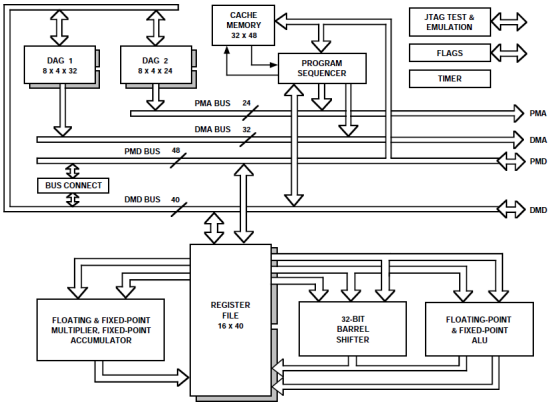


FIGURE 3.6 – *Architecture of the ADSP21020*

3.6.2 ADSP-21020

The ADSP-21020 is a superscalar floating-point processor with a Harvard Architecture [Hennessy and Patterson, 2006] from Analog Devices. It has a peak performance of 60 MFLOPs and sustained performance of 40 MFLOPs at 20Mhz. It offers 32-bit IEEE single precision and 40 bit extended precision floating-point operations. The original IP from Analog devices was licensed to ESA to produce a radiation hardened version with European technology. Atmel produces the radiation hardened version, the TSC21020F [Atmel, 2007].

The TSC21020F contains three independent computation units: an ALU, a multiplier with fixed-point accumulator, and a shifter, which can be seen in Figure 3.6. In order to meet a wide variety of processing needs, the computation units process data in three formats: 32-bit fixed-point, 32-bit floating-point and 40-bit floating-point. The floating-point operations are single-precision IEEE-compatible (IEEE Standard 754/854). The 32-bit floating-point format is the standard IEEE format, whereas the 40-bit IEEE extended- precision format has eight additional LSBs of mantissa for better accuracy.

The computation units perform single-cycle operations; there is no computation pipeline. The three units are connected in parallel rather than serially, via multiple-bus connections with the 10-port data register file. In a multifunction computation, the ALU and multiplier perform independent, simultaneous operations.

3.6.3 Actel anti-fuse

There are several types of FPGAs, most of them are based on SRAM technology which makes them more difficult to use in space, certainly for deep-space missions, because SRAM-based FPGAs are more susceptible to radiation. To

overcome this issue, anti-fuse FPGAs are more often used for these kind of applications. Anti-fuse technology is based on the fact that a permanently conductive circuit is made when a high enough voltage has been applied [Comer and T., 1996]. The downside is that they have fewer programmable elements and can be programmed only once while an SRAM based FPGA can be programmed multiple times.

Actel is one of the main suppliers of radiation-hardened anti-fuse FPGAs. Their largest option with highest capacity is currently the RTAX 4000 series, which offers the equivalent of 500.000 logic gates. To put this in perspective, Xilinx produces the Virtex 5 SRAM based FPGA which can emulate up to 4.000.000 logic gates.

There is active research and development to create better radiation tolerance for SRAM based FPGA and they are being used in space although in very specific cases and often only on low earth orbit [Ostler et al., 2009].

3.7 Speed

Although the actual required processing speeds varies with missions and the type of sensor observations, a rough estimate can be made. As an example, consider what would be required to compress images from the NAC during the JUICE mission. The near angle camera from the JUICE mission has a projected data rate of 36700 Kb/s in words of 14 bit, for 5 images of 1024×1024 in 2 seconds. The LEON2 and the TSC21020F do not support any form of Single Instruction Multiple Data (SIMD) and as such the 14 bit words from the sensor will need to be mapped on the 32 bit data types. This corresponds to an input rate of $36700 * 10^3/14 = 2622\text{Mwords}/s$. The projected output rate is 12233.4 Kbit/s which in 32-bit words relates to 385 Kword/s. This is not that much since the LEON 2 processor runs at a speed of 100Mhz which provides 86 MIPS. So as a rough estimate, around 5Mhz is spent in loading and storing the operands. When for this image the CCSDS 122 standard is used to decompose the image in wavelet components the following calculations are required. For each two samples 8 additions and 2 shifts and 1 multiplication are required for the fixed-point implementation and 12 additions, 4 shifts and 4 multiplications if the floating-point implementation is used. So in short 11 operations for 2 samples in fixed-point and 20 operations for the floating-point implementation [CCSDS, 2005]. A single image is made up out of 1024×1024 samples. All samples are processed twice, first in horizontal order and then in vertical order. The standard uses three passes. This means that the number of operations roughly can be estimated by $11 * 1024 \times 1024 + 11 * 512 \times 512 + 11 * 256 \times 256 \approx 15\text{M}$ operations for fixed-point and $20 * 1024 \times 1024 + 20 * 512 \times 512 + 20 * 256 \times 256 \approx 27\text{M}$ for floating-point. This relates to $5 * 15138816/2 \approx 38\text{M}$ operations per second for fixed-point for the 5 images in 2 seconds and $5 * 27525120/2 \approx 68\text{M}$ operations

for floating-point. This assumes that there would be no load and store operations in between and everything would be locally available which is not possible since there is 16KB of data cache available and the image itself is 1MB. Smart implementations can stream the data through the whole processing chain without the need to load and store the whole image, but can operate on a part of the image. This means that there is a need for load and store operations between the parts that can be processed.

For fixed-point 38Mcycles/s is roughly spent on calculations alone. The RICE encoder which comes after the decomposition into wavelet coefficients takes up the majority of the processing time in the current architectures due to all the bit-wise operations. From earlier estimates, which can be seen in Figure 3.4, it takes up over 60% of the processing time. The wavelet decomposition alone takes up 38Mcyles/s, it is not possible to do this in real-time on a LEON2, since by rough calculation this would need more than 125% of the processing power. Not even taking into account that the floating-point operation takes up almost twice as many cycles. Also this neglects packing, loading and storing of the data for transmission which would need to be added to this as well, so a single LEON2 does not even come close to performing the processing for the CCSDS 122 standard in near real-time.

On paper the TSC21020F is a bit better in terms of calculations per clock-cycle since there is the possibility to load multiple operands at the same time and some operations can be done in parallel. However, the algorithm is not easily split in such fine granularity that it can simply be distributed over the different ALUs of the TSC21020F. In addition the processor runs at a much lower speed which makes it even more difficult to obtain the same speed as the LEON2 in this case.

The JUICE mission does not directly specify that there would be a need to perform all processing real-time, only a burst data rate of 5 images in 2 seconds, which perfectly allows to store the data and do the processing later on. The nearest which is specified is a sustained data rate of 2 images per second. For fixed-point version this alone would require 30M cycles /s for the filtering alone.

Even though it is not directly specified, it does illustrate that even a "simple" camera can require a lot of processing time. To better understand the required computational capacity on should be aware that this is not the only instrument on board and not even the most demanding one. Just putting more LEON2 processors on board is also not a solution because it would take up too much power; the LEON2 in the Atmel implementation requires 1 Watt at 100Mhz.

3.8 Requirements

From the previous sections requirements for a future on-board payload processing platform can be derived. Most of the issues with the current architec-

tures available revolve around a few requirements.

I The current processing speeds will just not be enough for future missions and instruments. Consequently this also means that interconnect speeds, both on-chip as well as off-chip need to go up to feed the data of the instruments to and from the chip. The clock-speed is mainly limited by the technology which needs to be suited for deep space missions and comes with the added requirements of radiation hardening and power restrictions. Adding cores also comes with problems as the die size is limited.

II Current standards, like the CSSDS 122 have some parts which would benefit from floating-point operations. If floating-point operations are needed, hardware floating-point is almost undeniable, since it is difficult to maintain a high speed with software floating-point implementations. Hardware floating point comes with increased die size which is already limited. Future standards are more likely to have floating-point computations added to them because of their ease of use and general availability on desktop computers. Software development time is expensive and floating-point hardware could facilitate to bring this cost down.

III One of the more controversial requirements is that of being ITAR free. Politics should not play a role in hardware development, certainly if it strives to drive the knowledge and understanding of the world around us. Unfortunately this is not the case and politics do play a role. Therefore the technology should be developed in Europe to avoid the ITAR restrictions put up by the United States regulatory entities.

IV The power consumption and related silicon area of the platform should be as low as possible. More power consumption and larger chip area directly influence the mass of the total system. Silicon area increases package size and PCB area, while increased power consumption increases the size of the power plant of the platform, either in solar panel size or nuclear battery size.

V Of course, there are more requirements, for example regarding the interfaces which it should be able to connect to, or maybe the ability to run legacy software. Most of these requirements can be met without too much trouble, certainly if the main requirements previously mentioned are met.

As in most engineering problems the requirements are contradictory and trade-offs are needed to arrive to a good solution.

3.9 Possible development routes

From the previous requirements some possible development routes can be derived. These routes have been identified by ESA as well. Funding was found for the following three different tracks:

- Employ commercial of-the-shelf components and employ radiation hardening by special packaging and PCB design.
- License an IP from a vendor under ITAR regulation but develop the chip in Europe.
- Build an entirely new system on chip from European IP.

It is expected that the first item on the list can be employed with any chip which is commercially available. This thesis does not discuss this item any further as it focuses more on radiation hardening by repackaging existing designs.

In the past the second item on the list has been proven to be quite successful with the TSC21020F IP. Various SoCs for payload have been using this IP. As of writing this thesis, this option is being investigated further by ESA. The downside to this is the potential costs involved. Not only for the IP itself which comes at a high cost but also the possible customizations. On-board platforms do not need options which are most commonly found in these IPs like MPEG compression but benefit more from interfaces like Spacewire. Customization of the IP for space applications is not very attractive for a vendor due to the low volume of chips and small return on investment.

It is not entirely clear which IP would be best suitable in such case. Suitable choices to investigate are IPs like Texas Instruments' TMS320C674x series [Texas Instruments, 2011] and Analog Devices' AD SHARC 2147x series [Analog Devices, 2011]. Both operate in the 300Mhz range and offer similar single precision floating point performance: two single precision multiplications per clock cycle.

The TI platform offers functions like USB, multichannel audio, I^2C, SPI, 100Mbit ethernet and various other peripherals which are not needed. The AD platform is a more bare product in which there is little extra functionality, which makes it a little better to be used. Whether or not these platforms would be under consideration would depend on the willingness of the vendors. For this reason this topic is also not further discussed in this thesis.

The third item on the list is the development of a system on chip with European IP. Several companies submitted designs that complied with the budget and time constrains set by ESA. It is not entirely known what the submitted designs were since the submission process was not public, but an educated guess can be made.

The most obvious route to take would be to take a couple of general purpose LEON processors, as presented earlier in this chapter. Aeroflex Gaisler is

currently on version 4 of the LEON which adds the benefit of a deeper 7 stage pipeline, multiprocessor support as well as a wider internal bus and the capability of level 2 cache. Aeroflex Gaisler offers the LEON4 which is able to run up to 800Mhz in 65nm, and each of which can be equipped with a separate floating point unit to realize the giga flop per second requirement. The LEON4 can be combined with other space related IP with which Aeroflex Gaisler has experience as they have developed other payload processors in the past.

Other routes would have most probably been a bus design with a LEON or ARM processor together with one or several dedicated accelerators, like with the TSC21020F design. The main downside to the combined LEON4 and other designs would be the scalability and still the lack of a high performance replacement for the TSC21020F DSP.

A variation on the design presented in Chapter 5 ultimately was selected by ESA and as such this design was investigated further. The two main reasons why this design was chosen are the new VLIW type processor, the Xentium, that can be licensed separately, and the scalability of the design due to the network-on-chip. It also combines existing techniques like a LEON 2 which makes it relatively simple to run legacy code on the platform. This project would test a few different techniques at the same time which could later be used to develop a complete SoC capable of performing the tasks projected by a mission like JUICE. Further details of the platform are explained in Chapter 5.

The one thing that is missing from the Xentium DSP, further explained in Chapter 5 and Chapter 6 is hardware floating point. The scalability of the platform, however, opened the possibility to investigate the execution of the algorithms in fixed-point and software floating point. Although initial projections looked promising, at the same time the need for a hardware floating point addition to the Xentium VLIW architecture was investigated. The following section explores some of the existing hardware floating point units available and explains why a new route is taken.

3.10 Hardware floating point

Hardware floating point units pose the same problem as a VLIW by itself when it needs to be integrated into an European design. Not that many floating point units exist that can be easily licensed in a design.

Often designs are used within an FPGA, which allows the use of the Xilinx or Altera library. For ASICs, it is a little bit more difficult but the DesignWare library from Synopsys offers floating point as well [Synopsys, 2011]. Alternatively something like the FPU from Aeroflex Gaisler can be licensed [Gaisler Research, 2005*a*] for a fee.

The downside of these options is that they are dedicated solutions for floating point while a majority of the algorithms presented in this chapter rarely

use floating point. The silicon area would be wasted in such a case. Radiation hardened silicon is not cheap and as such it would be beneficial if it could be combined with integer calculations.

Although hardware floating point has been around for some time, vendors like Intel, AMD and ARM have only recently added instructions for fused multiply add. The new Intel Haswell architecture [Intel Corporation, 2013], the new AMD Bulldozer and Piledriver architectures [AMD, 2012] and the new ARM NEON/VFP architecture [ARM, 2013] all offer a form of fused multiply add. These architectures became commercially available in the year 2013. The main benefit of a fused multiply add is the added precision which is further explained in Chapter 4 and Chapter 6. The downside is that the overall design becomes a bit more complex. These designs, however, are closed and cannot be licensed, nor are they explained well in any documentation. The fused multiply add operation is something the TI and AD DSPs do not offer.

A freely available hardware floating point unit synthesizeable for ASIC which offers FMA in floating point as well as fixed point data types does not exist.

3.11 Conclusion

This chapter presented current and future requirements of space missions. It provides a clear indication in the increasing need for processing power on board space platforms. The increase in number of instruments and increasing bit rates and sample moments, increase the amount of data which needs to be sent to Earth. The down-link bandwidth, however, is not expected to increase all that much therefore the need for processing power to compress the sensor data is increasing as well.

The ITAR restrictions as well as the harsh environment of deep-space make it difficult to use commercial off the shelf components. A new design has to be made which is capable to provide the processing requirements as a well as be free of the ITAR regulations.

Chapter 4

Sabrewing *

Abstract

Even though floating-point arithmetic is costly in terms of silicon area, a single hardware design for floating-point as well as integer arithmetic is rarely seen. While components like multipliers and adders can potentially be shared, floating-point and integer units in contemporary processors are usually disjoint.

This chapter describes a new architecture which tightly integrates floating-point and integer arithmetic in a single datapath, intended for use in low power embedded digital signal processors. The architecture is tailored to digital signal processing by combining *floating-point fused multiply-add* and *integer multiply-accumulate*. It could be deployed in a multi-core system-on-chip designed to support applications where floating-point intensive algorithms are multiplexed in time with fixed-point ones.

The architecture is configurable at design-time. It has been thoroughly checked for IEEE-754 compliance by means of a floating-point test suite originating from the IBM Research Labs. A proof-of-concept has been implemented using STMicroelectronics 65nm technology. The current prototype design supports 32-bit signed two's complement integers and 41-bit (8-bit exponent and 32-bit significand) floating-point numbers as well as the more common IEEE-754 single precision floating-numbers. Implemented in ST's general-purpose CMOS technology, the design can operate at a frequency of 1.35GHz, while 667MHz can be achieved in low-power CMOS. This is competitive with current technology low-power floating-point units. Post-layout estimates indicate that the required area of a low-power implementation can be as small as 0.04mm^2. Power consumption is in the order of several milliwatts. The VHDL structural description of this architecture has been made available for download under BSD license †

*This chapter is a revised and updated version of [Bruintjes et al., 2012]

†`http://caes.ewi.utwente.nl/research/publications-by-year/accompanying-material/sabrewing.html`

4.1 Introduction

The previous chapters described the different number systems and algorithms on current on-board payloads. In 2008 ESA opened a tender for a new processing platform for payload processing [ESA/ESTEC, 2008*a*]. The development of this platform is described in Chapter 5. Although most of the requirements listed in the previous chapter are met with the platform one of them is not. That requirement is a hardware platform capable of 1 giga floating-point operations per second, which is difficult to obtain with an integer-only platform. Therefore this chapter describes a an arithmetic and logic unit, ALU, of a processor capable of integer and floating-point operations. This ALU was developed alongside the new processing platform. When both were ready the integration of this ALU in the Xentium VLIW processor was investigated as reported in Chapter 6. Section 3.5 of Chapter 3 described a number of algorithms; the main operations of these algorithms are based on integers or fixed-point. A minority of operations is based on floating-point, mainly the filtering operations. Although it is not technically required to have floating-point hardware, it could be emulated in software, it's performance does greatly benefit from it. The main reason why floating-point is often discarded from processor design is the fact that it occupies a large area on silicon while having a relatively low utilization degree.

Because the basic operators in floating-point units are the same as the ones in integer ALUs, the floating-point hardware could in essence be used to perform integer operations. The ability to schedule the floating-point unit for integer instructions would boost the integer performance and increase the overall utilization of the hardware. A combined integer and floating-point architecture does not necessarily have to replace its existing dedicated integer or floating-point counterparts. Dedicated architectures will always be slightly more energy and area efficient when fully utilized. However, when both floating-point and integer arithmetic is required and the hardware is not fully utilized, an architecture that combines both is advantageous for engineers that have to deal with tight area and energy constraints. These conditions hold for on-board payload processing and hence a mixed integer and floating-point processor is advantageous.

Bringing floating-point arithmetic, merged with integer functionality, to the embedded hardware domain presents some challenges. First of all, the design needs to be lightweight (both small and low-power), which is a rarely-seen property in floating-point hardware. Secondly, even though the basic components in floating-point and integer units are almost identical, actually sharing them is not trivial. To address these issues we present Sabrewing[‡], a lightweight but powerful architecture that combines floating-point fused multiply-add (FMA)

[‡]Sabrewings are hummingbirds, small and light birds that symbolize the properties of our architecture.

[Montoye et al., 1990] with integer multiply-accumulate. Multiply-add (A×B+C) instructions are particularly interesting for digital signal processing (DSP). Not only do they increase performance, they also produce more accurate results due to the elimination of an intermediate round operation. Another characteristic property of Sabrewing is the small number of pipeline stages. A typical floating-point datapath is deeply pipelined to attain higher clock frequencies. However, experience has taught that designing efficient compilers for deep pipelines is difficult [Hennessy and Patterson, 2006], especially if the datapath is to be shared between floating-point and integer instructions. For this reason, the depth of the pipeline is kept to three stages, while each individual stage is delay optimized as the delay in one stage is likely to determine the overall clock frequency.

The hardware description of Sabrewing in VHDL is parametrized such that the design can be configured at design time. Besides choosing the number of bits used to represent floating-point and integer numbers, users can enable/disable full IEEE-754 support for infinity and NaN (Not a Number) by simply specifying this in a configuration file. The main restriction of the design is that the number of bits used for the significand should match the number of bits used for the integers. In the default configuration the significand/integers use 32 bits, the exponent uses 8 bits and infinity and NaN are disabled for reasons that will be explained later. Single precision IEEE-754 floating-point is also supported in the current implementation.

Note that throughout this chapter, N_{sig} indicates the significand of floating-point operand N and N_{exp} the exponent.

4.2 Related Work

The FMA concept is slowly gaining popularity [Butler et al., 2011; Cornea et al., 2003; Mueller et al., 2005; Wittenbrink et al., 2011]. Where FMA was initially an extension to the existing floating-point datapaths that implemented multiplication and addition separately, we see that the conventional floating-point adders and multipliers are being replaced with FMA units in more recent processors. We believe that, since x86 general-purpose processors [Butler et al., 2011] are making the transition to FMA, it will become even more important. IBM became the pioneer of FMA [Montoye et al., 1990] when they introduced the System/6000. Successive publications mostly describe variations of the System/6000's FMA datapath. The work presented in [Quinnell et al., 2007] for example, shows how the overall latency of FMA instructions can be reduced by distinguishing multiple data flows for addition, based on the exponent difference of the input operands. Other examples include [Jessani and Putrino, 1998] where the datapath is modified such that multiple passes allow double precision instructions to be executed on a single precision datapath, and [Huang et al.,

2007] where a double precision FMA datapath is modified to support two-way single precision SIMD instructions. To our knowledge FMA has not yet found its way to low-power (embedded/DSP) processors.

Merging floating-point and integer hardware is seldom seen, yet the idea is not entirely unexplored. Thapliyal et al. [Thapliyal et al., 2006] propose to replace the standard 18x18-bit multiplier in FPGAs with a 24x24-bit multiplier, which could then be used for single precision floating-point multiplication. However, with this approach the multiplier is assigned to integer or floating-point duty at design-time, not both. The actual potential of combined integer and floating-point hardware was first recognized by Palacharla and Smith in [Palacharla and Smith, 1995]. They evaluate its feasibility and indicate that about 40% of the integer operations can be migrated to a typical floating-point datapath. In response to this work, Solihin et al. present a reconfigurable adder that can switch between integer and floating-point functionality at runtime [Lavenier et al., 2000; Solihin et al., 2001]. Some examples of merged floating-point and integer execution hardware can be found in commercially available processors as well. [Mueller et al., 2005] hints that the CELL processor is allegedly capable of performing integer multiply-additions in the synergistic processing element's single precision floating-point cores. Unfortunately no details are discussed except for the fact that such operations require preprocessing, resulting in a latency penalty. In the SPARC V2 architecture, the multiplier of the floating-point execution datapath has been isolated to serve as the main multiplication element for both integers and floating-point significands. However, the majority of the SPARC's floating-point hardware is still used exclusively for floating-point purpose. Architectures that do combine floating-point and integer hardware are still immature which shows that there is need for further research to optimize and explore the potential benefits.

4.3 Architectural Design Considerations

The following paragraphs explain the design choices made that led to the datapath which is presented afterwards in Figure 4.9 and explained in Section 4.4. The paragraphs describe the fused-multiply-add and incorporation of integers, the number representation, rounding and IEEE compliance, and instructions. The rationale behind the choices were bringing down the latency, improve operations per area where possible and provide high precision.

4.3.1 Multiply Add

There are two main ways to perform a multiply-add operation in floating point. The reason behind it is that the IEEE floating point standard states that for an addition, the smallest operand is shifted to the right until the exponents of both operands are equal. This condition requires that either the datapath makes a

decision how to handle the addition after the multiplication or makes sure that one of the operands of the addition is always smaller. This leads to either a cascaded design in which the alignment follows after the multiplication or a non-cascaded design in which the shift amount is determined alongside the multiplication.

The non-cascaded design requires a smaller shifter for the alignment but comes with the penalty of having a higher latency. The non-cascaded design would have required a minimal size of 68 bit versus the current design which has a 102 bit shifter. The choice was made to have a lower latency leading the choice for a non-cascaded design.

4.3.2 Integer integration

To increase the number of operations per mm^2 the choice was made to perform integer operations on the same datapath. Another option would be to have integer operations completely separate and performed in another ALU. The algorithms presented in Chapter 3, however, do not have floating-point as their main operations. It is only a single algorithm that may require floating-point operations in certain cases. If the Sabrewing would be solely used for floating-point operations the silicon area would be wasted most of the time.

For this reason the choice was made to not waste the silicon area and re-use it for integer operations.

4.3.3 Number Representation

In the default configuration, Sabrewing performs operations on 41-bit floating-point numbers, or IEEE-754 single precision numbers. As shown in Figure 4.1, the format closely resembles IEEE-754 single precision [IEEE, 2008], but uses 32 bits for the significand rather than 23. In this configuration, integer arithmetic is based on 32-bit two's complement notation. The reason for this combination is that 32 bits are commonly used to represent integers in today's computer architectures. In addition, our philosophy is that since 32-bit arithmetic components are needed for a full 32-bit integer datapath, we could just as well use that processing power for additional floating-point precision.

The IEEE formats also include reserved representations for infinity, NaN, zero and subnormal numbers. In the default configuration infinity, NaN and subnormals are not supported. Such deviations are not uncommon [Analog Devices, 2011; Mueller et al., 2005; Texas Instruments, 2011]. Besides the obvious performance gain by reducing the design complexity, certain applications also benefit from not having to deal with the special cases defined in IEEE-754. Infinity for example, behaves substantially different from the maximum representable number. A digital filter is much easier to implement in floating-point when the input simply saturates (interpreting infinity as the largest represent-

able number) rather than having to deal with infinity in a special clause. Since most of the projected use for this ALU is in streaming signal processing saturation would be of a larger benefit.

For these reasons, NaN and infinity are by default disabled, but can be enabled at design time. Subnormal support is a different matter. Because supporting subnormal numbers has an enormous impact on the area of Sabrewing, we feel it is justified not to implement them at all. In effect, these deviations from the standard mean that subnormal numbers are forced to zero, infinity becomes a very large number and NaN is not recognized (and also never produced). Numerically, the proposed format behaves similarly to single precision, but with more precision. In section 4.7 we will discuss the impact of full IEEE-754 support on the performance and area of Sabrewing.

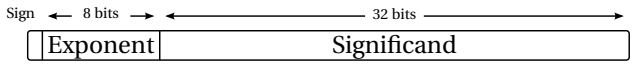


FIGURE 4.1 – *Sabrewing (41-bit, bias-127) floating-point format.*

Rounding

The IEEE-754 (2008) standard defines five rounding modes. There are designs that do not implement all these rounding modes or any at all [Analog Devices, 2011; Mueller et al., 2005; Texas Instruments, 2011]. Mainly because there is a penalty in the silicon area required to implement and because the rounding has a minimal effect on the final outcome of a single operation. The largest increase in size of the design is in the second stage which is presented in Figure 4.9 where the carry-propagate-adder is currently 101 bits wide. When no rounding would be supported at all the width of that adder could be 68 bits.

The targeted end-user of the ALU are scientists who demand data of the highest precision. Not only because the information gathered from the data is important but it can not be gathered again without very high costs e.g. launching a new satellite. So the choice was made to implement the most common four rounding modes of the IEEE standard.

The four rounding modes which have been implemented are: round to nearest, round to zero, round to positive infinity and round towards negative infinity. Rounding is implemented using a *guard*, *round* and *sticky-bit*, as shown in Figure 4.2. These additional bits indicate what the rounding direction should be after the result has been normalized. The guard and round bits are basically a 2-bit extension of the datapath. The sticky-bit acts as a control-bit to indicate whether or not the result is inexact (i.e., `1`'s are lost due to shifting or truncation) and does not participate in the floating-point operation itself. The sticky-bit is a bit that is beyond the two bit extension and is a bitwise OR of all the bits which come after these bits.

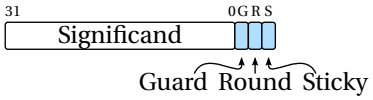


FIGURE 4.2 – *Guard, round and sticky-bit determine the round direction.*

4.3.4 Instructions

The need for floating-point signal processing in energy-efficient embedded systems has been the main drive for the development of Sabrewing. Because many DSP applications (e.g, fast Fourier transform, discrete cosine transform or digital filters such as FIR and IIR) greatly benefit from multiply-add operations, the main class of instructions implemented by Sabrewing are multiply-additions. Despite the fact that the majority of DSP algorithms is based on multiply-add, it should of course be possible to execute separate multiply and add instructions as well. These instructions are available as derivatives of multiply-add (A×B+0 and A×1+C respectively), which helps to keep the datapath structure as regular as possible and ultimately contributes to the multiply-add performance and area of Sabrewing. Input operands can be positive and negative, which provides the means to perform subtraction and multiply-subtract as well. In addition, the full range of comparisons (>, < and =) is implemented, supporting both integer and floating-point numbers. Finally, two directions of signed integer shifts are supported, because they are relatively easy to extract from a floating-point datapath. Other basic operations such as division and square root are not directly implemented in hardware because they would defeat the purpose of a lightweight floating-point solution. Moreover, they can be provided relatively easy as software routines, using FMA instructions [Markstein, 2004]. Considering that these instructions are rarely used, software emulation is feasible, as demonstrated in the Intel Itanium [Cornea et al., 2003; Robison, 2005] and CELL processor [Mueller et al., 2005]. An overview of the instruction support with corresponding latencies is shown in Table 4.1. An exhaustive table of all the instructions and rounding modes is provided in the appendix in Table A.1. Note that every cycle a new instruction can be issued and that all floating-point instructions require three cycles to complete. All integer instructions need two cycles. Even though this is a suboptimal solution from a hardware perspective (Section 4.6), the latency consistency is convenient for the compiler/scheduler.

4.4 Datapath design

Even though we limited ourselves to a few pipeline stages (in this case three), designing a reasonably performing FMA datapath is not trivial. This section shows a number of optimizations used in the Sabrewing architecture to achieve

TABLE 4.1 – *Sabrewing instructions*

Operation	**Operands**		**latency (# cycles)**	
			integer	floating-point
Multiply-Add	{A,B,C}	: floating-point or integer	2	3
Compare (>,=,<)	{A,B}	: floating-point or integer	2	3
Shift (≫,≪)	{A,n}	: integer and natural	2	n/a
Multiply	{A,B,0}	: floating-point or integer	2	3
Add	{A,1,C}	: floating-point or integer	2	3

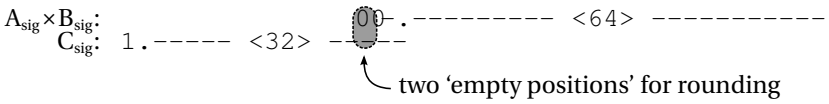


FIGURE 4.3 – *Positioning of $A_{sig} \times B_{sig}$ and C_{sig} prior to alignment.*

good performance without compromising the pipeline constraint and the area limitations of embedded systems. All examples assume the 41-bit representation presented in Section 4.3 (Figure 4.1).

4.4.1 Alignment

Floating-point addition requires that the smallest operand is shifted to the right during alignment until the exponents are equal. However, in case of FMA (A×B+C) the product (A×B) first needs to be computed before the smallest operand can be identified, which causes a major bottleneck. If C_{sig} is positioned entirely in front of the product, as shown in Figure 4.3 [Jessani and Putrino, 1998], alignment can be implemented in parallel with multiplication. By positioning C_{sig} entirely in front of $A_{sig} \times B_{sig}$, and imagining the radix points to be fixed, C_{sig} will always be the largest operand and therefore $A_{sig} \times B_{sig}$ does not have to be evaluated first. The reasoning behind this is that left shifting the significand lowers the exponent which makes it the smaller operand. This left shift at the start needs to be compensated when the values are added together.

For a basic floating-point addition (P+Q), the alignment shift is given by the absolute difference between the exponents: $|P_{exp} - Q_{exp}|$. However, multiplying two 33-bit significands (32 bits and the *hidden-bit*) yields a 66-bit significand, with two bits instead of one bit to the left of the binary point (Figure 4.3). The FMA exponent datapath must take this into account by incrementing the value of the intermediate exponent. To place C_{sig} entirely in front of the product, the exponent also needs to be adjusted by the number of bits used for the significand (32 in this case) plus the hidden-bit. Finally, as shown in Figure 4.3, two more positions may have to be taken into account for rounding. During

fused multiply-add operations, the guard and round bits can be placed between C_{sig} and $A_{sig} \times B_{sig}$. This has the advantage of no overflowing significands. The above-mentioned adjustments can be combined into a single implementation-specific offset. This offset should also include the bias of the IEEE-754 representation, which normally accumulates during the addition of A_{exp} and B_{exp}. In this particular case, the alignment shift is:

$$\begin{aligned} \text{Shift} &= A_{exp} + B_{exp} - C_{exp} + 36 - 127 \\ &= A_{exp} + B_{exp} - C_{exp} - 91. \end{aligned} \tag{4.1}$$

And the corresponding exponent:

$$\text{Exponent} = A_{exp} + B_{exp} - 91 \tag{4.2}$$

Sticky-Bit

During alignment, precision may be lost due to the shift. For rounding purpose, a *sticky-bit* keeps track of precision loss (Section 4.3.3). Whenever a `1` is shifted out of range, the sticky-bit is asserted to `1` and stays `1` during the entire computation.

4.4.2 Multiplication

Although multiplication can be performed in parallel with alignment, the multiplier design itself continues to be decisive for the latency and area of the datapath. Sabrewing uses a combination of modified *Booth encoding* [Booth, 1951] and a *Wallace tree* structure [Wallace, 1964] for its single-cycle multiplier. Two steps can be distinguished in single-cycle multiplication. Generating partial products and summing these partial products. Booth encoding helps reduce the number of partial products while a Wallace tree efficiently accumulates them.

Booth encoding

Booth states that M×`00111110` (M×(2^5+2^4+2^3+2^2+2^1)) equals M×`010000(-1)0` (M×(2^6-2^1)). The number of additions in the latter is obviously less, however, this basic Booth encoding only applies to multi-cycle multiplication. For single-cycle multiplication, Booth encoding only works when radix-4 multiplication [Parhami, 2000] is used. In radix-2 multiplication, the multiplier bits generate partial products that are either 0× or 1× multiples of the multiplicand. In radix-4 multiplication, two multiplier bits generate partial products that are 0×, 1×, 2× or 3× multiples of the multiplicand as shown in Figure 4.4.

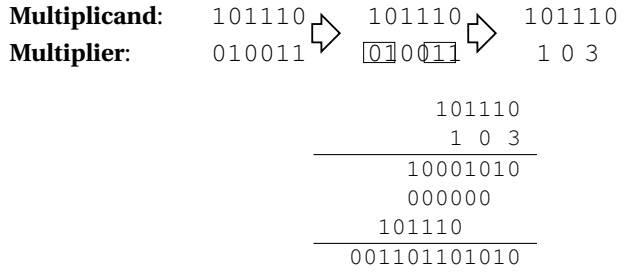


FIGURE 4.4 – *Modified Booth encoding.*

Note that 3× is not a nice regular multiple (i.e., cannot be obtained by merely shifting). However, with Booth encoding this multiple can be obtained by (4×)-(1×), which is again regular. Radix-4 multiplication reduces the number of needed adders from $n+1$ to $\lceil (n+1)/2 \rceil$. Hence, by using modified Booth encoding the area of the multiplier is reduced significantly without negatively affecting its latency. In addition Booth encoding enables us to support both two's complement and unsigned inputs in the multiplier.

Wallace tree structures

Carry propagation in adders is among the longest latencies found in most arithmetic circuits. Therefore, carry-save adders (CSA) [Earl, 1965] are used to accumulate the partial products. CSAs are a special type of adder that immediately output their carries instead of propagating them. An n-bit CSA yields an n-bit sum and an n-bit carry that can be converted to a regular n+1 bit binary number by adding the sum and carry as usual. This seemingly cumbersome way of adding numbers is very efficient for accumulation. Instead of propagating m carries for a series of m additions, only one carry propagation is required at the bottom of the multiplier array.

By placing the CSAs in a regular tree structure, a Wallace tree is obtained (Figure 4.5). As opposed to a standard multiplier array where the length of the entire path grows linearly with m, the depth of a tree ($\lceil \log_2(m) \rceil$) is much smaller and therefore a much better latency can be achieved.

4.4.3 Addition

The addition of operand C and multiplication result of A×B are elegantly *fused* when an additional CSA is inserted at the end of the multiplier tree, as shown in Figure 4.5. However, because IEEE-754 floating-point representation is sign-magnitude, this only works for positive operands. To support *subtraction* (e.g., -A×B+C or A×B-C), additional measures have to be taken. Internally converting all input to two's complement notation will work but is not efficient. *End-*

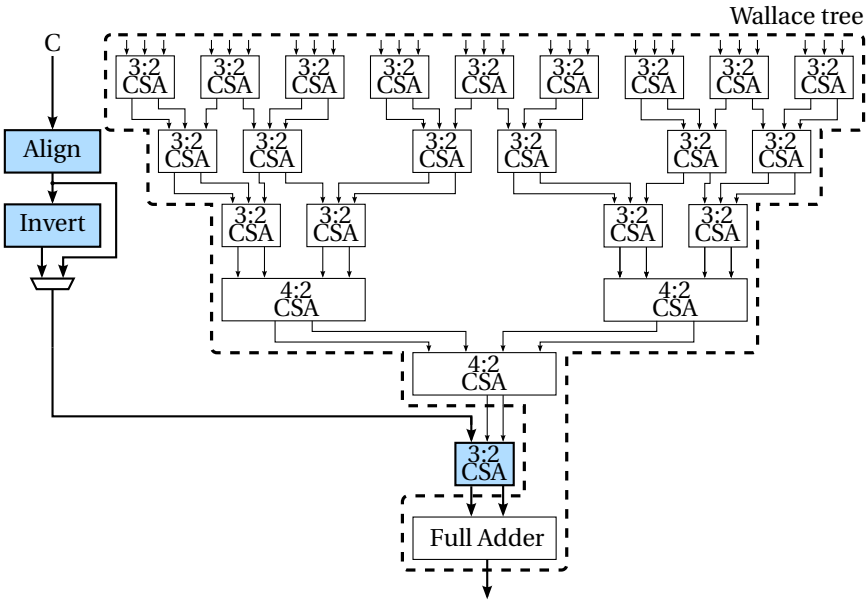


FIGURE 4.5 – *Addition merged with the (Wallace) multiplier tree.*

```
-1    1110
+5    0101
-------------------------------
+4(1)0011  -> 0011+1 = 0100
```

FIGURE 4.6 – *End-around carry addition example.*

around carry addition [Vassiliadis et al., 1989] requires much less hardware and does not contribute to the critical path since only C needs to be converted to one's complement, which overlaps with the delay of the multiplier. With end-around carry addition the two numbers are added and then the carry coming out of the highest bit is added to the result (this is what is called an "end-around-carry"). An example of this is shown in Figure 4.6.

The principle of end-around carry addition applies to the magnitude of the operands (|A×B| and |C|); the sign bits are merely used for control. If $|A\times B| > |C|$, the magnitude is given by: $|R| = (|A\times B| - |C|) = |A\times B| + \overline{|C|} + 1$. This means that |C| is inverted (converted to one's complement) before being added to |A×B|, and after addition the result must be incremented. An important property of end-around carry addition is that the carry out (C_{out}) is always `1` in this case. If $|A\times B| < |C|$, the resulting magnitude is obtained by: $|R| = \overline{|A\times B| + \overline{|C|} + 0}$, and the carry out will always be `0`. Table 4.2 provides an example of both cases: at the left |A×B| = 5 and |C| = 6 while at the right the values are swapped.

In a more generalized form, the magnitude can be found by: $\Sigma = |A\times B| +$

TABLE 4.2 – *Two examples of computing* $|A\times B| - |C|$ *with end-around carry addition in which* $|A\times B| > |C|$ *and* $|A\times B| < |C|$.

Operands	**decimal**	**binary**	**decimal**	**binary**
|A×B|	5	0101	6	0110
|C|	6	0110	5	0101
|A×B|		0101		0110
$\overline{\|C\|}$		1001		1010
end-around carry		0000		0001
difference	−1	$\overline{1110}$	+1	0001

$|C|^* + C_{out}$, where $|C|^*$ equals |C| for effective addition and $\overline{|C|}$ for effective subtraction. The sign bits combined with C_{out} provides enough information to determine if the result needs to be re-complemented after addition: $\Delta = \overline{C_{out}} \wedge$ "effective operation". The "effective operation" is either addition or subtraction. When Δ is 1, the result is re-complemented, otherwise it remains unchanged. If the final results designated by |R| then the final results is a shown in Equation (4.3) where $\oplus$ is an XOR operation.

$$|R| = \Sigma \oplus \Delta. \tag{4.3}$$

The sign-bit is defined as:

$$R_{sign} = (A_{sign} \oplus B_{sign}) \oplus \text{effective operation} \wedge \overline{C_{out}} \tag{4.4}$$

except when the result is exactly zero [Bruintjes, 2011].

4.4.4 Normalization

After addition, the result is most likely not normalized. A normalized result is obtained by counting the number of leading zeros in the significand and shifting it to the left by that number. Typically the number of zeros can only be counted after the addition has been performed, resulting in another bottleneck. To tackle this problem, *leading zero anticipation* (LZA) is used to predict the number of leading zeros.

Leading Zero Anticipation

There are different techniques to predict the number of leading zeros [Schmookler and Nowka, 2001]. Some aim for a prediction that is always exact, others settle for a close approximation. LZA describes a boolean relation between the input of the adder (A,B) and a string f of indicators: 0^k1x^* (where k≥0 and x is 1 or 0) in which f_i indicates whether position i will be a 1 or 0. Here k

represents the estimated number of leading zeros and x* is either zero or one and '*' represents zero or more repititions of 'x'. The LZA [Schwarz, 2006] used in Sabrewing predicts the leading zeros with one position uncertainty. From [Schwarz, 2006] we obtain the value of each position by:

$$f_i = \begin{cases} \overline{T_0} T_1 & (i = 0) \\ T_{i-1}(G_i \overline{Z_{i+1}} \vee Z_i \overline{G_{i+1}}) \vee \overline{T_{i-1}}(Z_i \overline{Z_{i+1}} \vee G_i \overline{G_{i+1}}) & (i > 0) \end{cases} \tag{4.5}$$

where

$$\begin{aligned} T_i &= A_i \oplus B_i \\ G_i &= A_i\ B_i \\ Z_i &= \overline{A_i}\ \overline{B_i} \end{aligned}$$

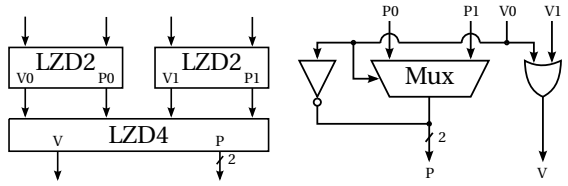
When position f_i is the least significant bit (LSB), position f_{i+1} represents the carry-in of the adder(MSB $i = 0$). This approximate LZA requires much less hardware than exact LZA [Schmookler and Nowka, 2001]. The downside is that a correction may be needed.

Handling the LZA mis-prediction The position of the leading `1` according to Equation (4.5) is either exact or it is located one position to the left of the actual leading `1`. Since the error is so consistent, it is easily corrected. In case of a mis-prediction, we can simply shift the significand one more position to the left.

Leading Zero Detection

The prediction obtained from LZA needs to be encoded into a binary number to drive a left-shifter for actual normalization. Two different techniques to count the number of leading zeros exist. One is based on monotonic string encoding, the other on a hierarchical counting tree. The monotonic string method [Hayes et al., 1985] is typically faster while the counting tree has the advantage in terms of area and energy-efficiency. Because LZA takes leading zero detection (LZD) off the critical path, counting trees are attractive for an approach like Sabrewing.

A well known LZD circuit is the one from Oklobdzija [Oklobdzija, 1994]. It is based on a hierarchical counting tree. This tree is built up from small 2-bit LZD circuits (Figure 4.7(a)) that are combined into a 4-bit LZD circuit (Figure 4.7(b)), which can be used to create 8-bit LZDs and so on. The technique does not directly apply to an input that is not a power of two, implying that a 128-bit LZD will be needed to count the number of leading zeros in the 101-bit LZA result (Figure 4.9). Because this is not very area efficient, a modified LZD algorithm that performs better in case the number of bits of the input is not a power of two, has been devised for Sabrewing [Bruintjes, 2011].



(a) *Four bit LZD circuit (P indicates the #zeros and V indicates if the result is valid)*

(b) *LZD4 logic*

FIGURE 4.7 – *Leading zero detection.*

Sticky-Bit In order to reduce the obtained 101-bit significand to the original input size (32 bits), the 66 LSBs will have to be truncated (the MSB will become a hidden-bit and the guard and round-bit will be removed after rounding). For similar reasons as explained in Section 4.4.1, these bits are OR-reduced into the sticky-bit to keep track of any precision that is lost.

4.4.5 Rounding

After truncation, the significand is either incremented (rounded up), or remains unchanged (rounded down) based on the values of the guard, round and sticky-bit. Once the sticky-bit has been determined, IEEE-754 compliant rounding requires little effort. Several pattern matching algorithms [Woo-Chan et al., 1996] such as the one for *round to nearest even*, listed in Algorithm 1, can be used to determine the rounding direction. An overview of the rounding process is schematically depicted in Figure 4.8. What is important in rounding are the last three bits. In this case bit 66 which is called the guard bit, bit 65 which is called the round bit and then bit 64 to 0 which is OR reduced into a single bit, the sticky bit. These three bits together are used as input in Algorithm 1 to determine the value of the last bit of the 33-bit significand.

4.4.6 Pipelining

Based on a combination of the design principles described in this section, fused multiply-add can be implemented in a fairly well balanced three stage pipeline (Figure 4.9).

4.5 Integer Operations and Floating-Point Hardware

A number of fundamental arithmetic operations reside in every floating-point datapath: shifts, addition, subtraction and multiplication. These are the integer operations that can be incorporated in the floating-point datapath, without

Algorithm 1: IEEE-754 round to nearest ties to even.

Input: guard-bit ***G***, round-bit ***R***, sticky-bit ***S***, the normalized result truncated up to the original input size plus the hidden-bit $\texttt{Significand}_{\texttt{normalized}}$, and the least significand bit of the normalized result ***LSB***.
Output: The rounded significand $\texttt{Significand}_{\texttt{rounded}}$.
if *(**G** = 0)* **then**
    $\texttt{Significand}_{\texttt{rounded}} \leftarrow \texttt{Significand}_{\texttt{normalized}}$
else if *(**R** = 1 ∨ **S** = 1)* **then**
    $\texttt{Significand}_{\texttt{rounded}} \leftarrow \texttt{Significand}_{\texttt{normalized}} + 1$
else if *(**LSB** = 0)* **then**
    $\texttt{Significand}_{\texttt{rounded}} \leftarrow \texttt{Significand}_{\texttt{normalized}}$
else
    $\texttt{Significand}_{\texttt{rounded}} \leftarrow \texttt{Significand}_{\texttt{normalized}} + 1$
end

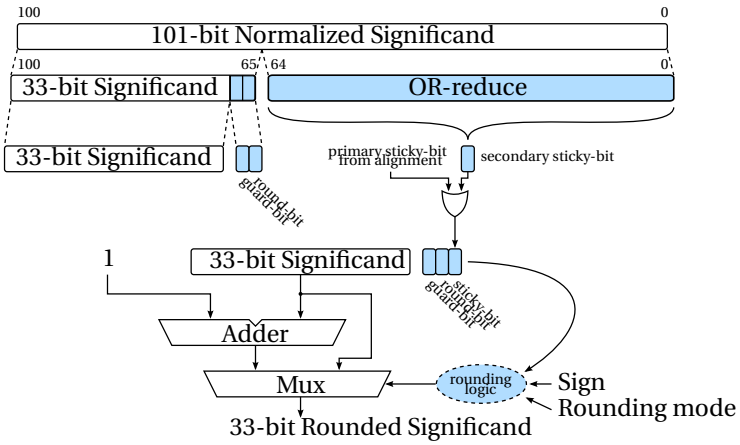


FIGURE 4.8 – *IEEE-754 compliant fused multiply-add rounding.*

noteworthy increase of silicon area or latency. The FMA datapath following from Section 4.4 is now reviewed to show which parts lend themselves to be modified for integer purpose.

Alignment requires several small adders/subtracters (Equation (4.1)) to determine the shift, and a right-shifter to actually align the operands. The adders and subtracters (and the entire exponent datapath for that matter) are not particularly useful for integer operations because they accommodate only 9-bit input (8-bit exponent plus one overflow bit). The shifter can be used for a shift-right instruction, although integers benefit most from an arithmetic shift while floating-point alignment is based on a logical shift.

The Booth encoded multiplier is directly usable for two's complement integer operation. Both full adders can in principle also directly be used for two's complement additions. However, full utilization of the multiply-accumulate potential of the FMA datapath requires a number of modifications of which a few are discussed here.

Major components in the normalization circuit are the leading zero anticipator, the leading zero detector and a left-shifter. The function of leading zero anticipation is too specific to be of any use for operations other than normalizing floating-point data. Leading zero detection is also not particularly useful outside the floating-point context, although some fixed-point instruction sets do include a leading zero count instruction. The normalization shifter on the other hand, provides the means to implement a left oriented shift. Since arithmetic shift-left is exactly the same as logical shift-left, no complications arise here. The rounding circuitry includes an incrementer, but since this does not provide any additional functionality over addition, re-using it will not yield a direct advantage worth the effort.

Although this synopsis may give the impression that merging integer and floating-point datapaths is trivial, this is certainly not the case. For example, the basic integer instructions all require at most one out of three pipeline stages. Multiply-accumulate, however, requires both the first and second stage. Performance-wise it would be best to implement the instructions with variable latency such that certain stages can be bypassed and the integer operations take up the minimum amount of clock cycles. However, this will lead to complicated instruction scheduling due to *pipeline-hazards*. For consistency we have chosen to implement all floating-point instructions in three clock cycles and integers instructions in two cycles (Section 4.3). A pipeline flush is required when changing from floating-point to integer operation. The penalty is however just one clock cycle because the pipeline is three stages deep. The next section describes the Sabrewing architecture, our approach to combine integer and floating-point hardware for low-power DSP.

4.6 The Sabrewing Architecture

The entire (default) Sabrewing datapath (configuration) is depicted schematically in Figure 4.9. To give an impression of the achieved level of reuse, all (partially) shared components are colored.

I/O Interfaces The input of the datapath consists of a 5-bit opcode and three 64-bit operands. Each 64-bit input is subdivided into two 32-bit words ('left' and 'right'), accommodating the floating-point sign and exponent, and significand/integer respectively. A 5-bit opcode is sufficient to encode all instructions from Table 4.1 and possible future extensions. The reason for making all data

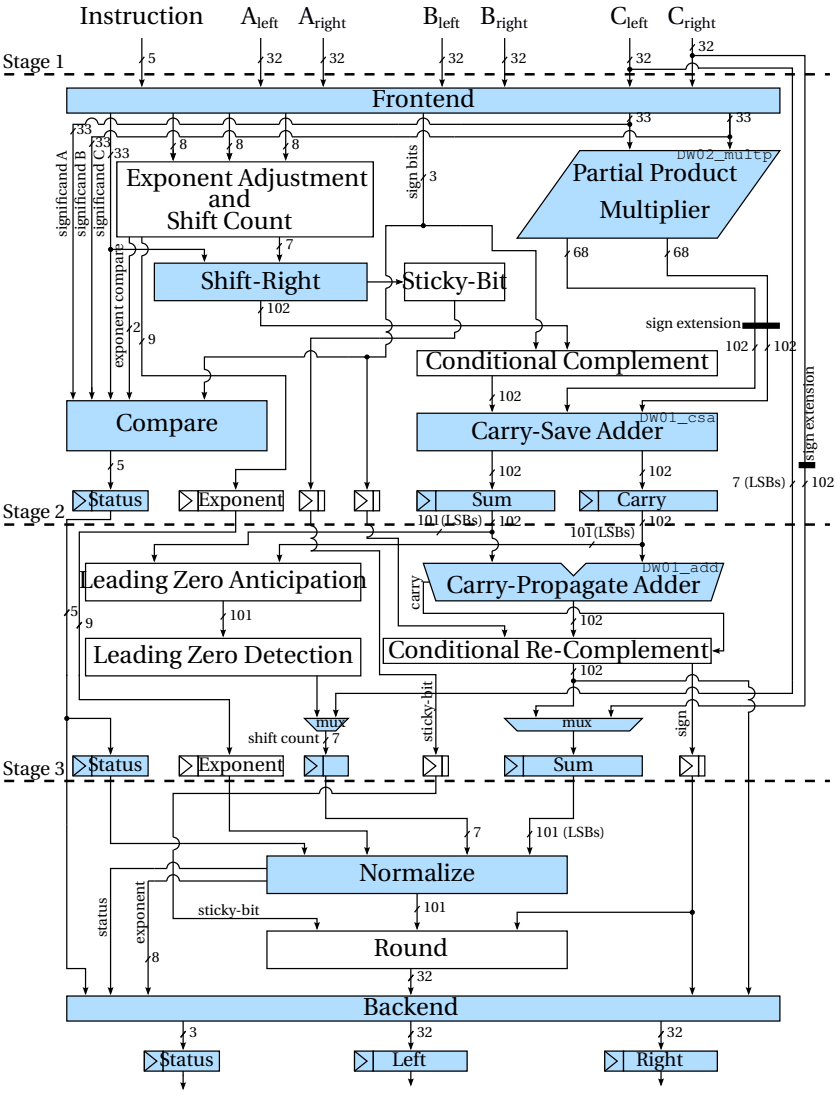


FIGURE 4.9 – *Default Sabrewing datapath overview (colored blocks indicate shared hardware).*

interfaces 32-bit (even though the exponent and sign require only 9 bits) is that this is convenient for memory access and it allows the datapath to be integrated in existing infrastructures more easily. The output of the datapath consists of three status bits acting as the exception flags, and two 32-bit data ports. In case of integer multiplication or multiply-accumulate, the higher order bits of the 64-bit result are mapped to 'left' and the lower order bits to 'right'.

Frontend The frontend performs several preparatory tasks. Each opcode is translated into control signals that configure the datapath. All floating-point significands are extended with an MSB with value `1` to make the *hidden-bit* explicit, while integers are sign extended to match the width of the new 33-bit significand. In case of single precision floating-point operations, the operands are unpacked. Each operand is checked for zero, and when the Sabrewing is configured to do so, also for infinity and NaN.

Exponent Datapath and Alignment Shifter Equation (4.1) and Equation (4.2) show how the shift for alignment and corresponding exponent can be found. An actual implementation requires one adder and two subtracters. The exponent is manipulated with 9-bit arithmetic, such that the intermediate results will never overflow. Underflow can not be prevented as easily. In order to detect (temporary) underflow, a 9-bit comparator is used to determine if the intermediate exponent is smaller than the bias. If this is the case, the exponent will underflow, which is registered for later use. Actually, raising the exception flags is postponed until after rounding because normalization could undo overflow or underflow. To save area, the comparator is multiplexed with primary input A_{left} and B_{left} to facilitate floating-point compares.

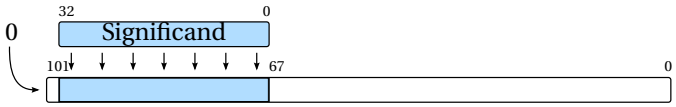
The input of the alignment shifter (shift-right) is extended by one bit, of which the value depends on the type of input: `0` for floating-point and `1` for integer. This allows the shifter to be implemented as an arithmetic shift-right, such that it can serve both integer and floating-point operations. To aid datapath regularity, the input to the shifter is mapped as shown in Figure 4.10. By mapping integer input to the LSBs, the shifter output can be routed through the carry-save adder $(A \ll n) = 0 \times 0 + (A \ll n)$ to match the two cycle latency of the other integer instructions, without using additional registers. As shown in Figure 4.9, the output of the Carry-Propagate adder is sent directly to the backend of the datapath such that the third stage is bypassed for all integer arithmetic and shift-right.

Sticky-Bit The primary sticky-bit (caused by alignment) is implemented as a 66-bit OR-gate reduction tree, fed with the bits from C that are shifted beyond the range of the datapath.

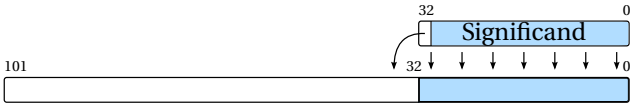
Compare With the compare operation we face similar problems as encountered in the shifter. Two's complement comparators do not operate correctly on unsigned operands and vice versa. The solution is again to perform an input extension, such that synthesis tools will infer only one (signed) comparator. Note that this modification only needs to be applied to the significand comparator. Moreover, the full range of compares (>, = and <) require only implementations of > and =, since < can easily be derived.

Multiplication and Addition The heart of Sabrewing consists of the *partial product multiplier*, a Wallace tree without the final carry-propagate adder; the complementers that invert the addend in case of subtraction; an additional CSA and the final carry-propagate adder.

Modified Booth encoding is used to support both floating-point (unsigned) and integer (signed) multiplication. Although other (possibly more efficient) techniques exist for signed multiplication [Parhami, 2000; Sjalander and Larsson-Edefors, 2008], Booth has the advantage that it is widely adopted and particularly well supported in the Synopsys design flow. In our prototype (Section 4.7), Synopsys DesignWare components [Synopsys, 2011] were used to implement parts of the core functionality. The `DW02_multp` IP implements the partial product multiplier, `DW01_csa` the carry-save adder and `DW01_add` the main adder. Both are indicated in Figure 4.9 with small letters in the carry-save adder, carry-propagate adder and partial product multiplier. A few complications arise when Booth encoded Wallace trees are used for integer and floating-point multiplication. For FMA, sign suppression needs to be applied to prevent false carry out during end-around carry addition [Schwarz, 2006]. Because we have no control over the internals of `DW02_multp`, the problem was solved otherwise. By explicitly including sign bits during end-around carry addition, we can prevent false carry out from occurring in most cases (not for input that is zero).



(a) *Floating-point alignment*



(b) *Integer shift-right*

FIGURE 4.10 – *Input mapping to right-shifter.*

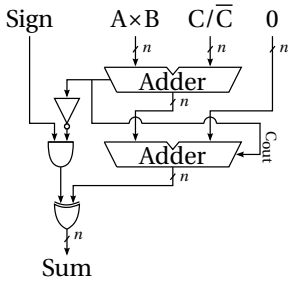


FIGURE 4.11 – *End-around carry addition.*

To share the multiplier between integer and floating-point operands in general, it should be noted that the output of a Booth multiplier is always signed, regardless of how the input is interpreted, and that the output is actually two bits wider than twice the input. Both complications are easily solved by sign extension. Since the aligned C operand including sign-bit is already 102 bits wide, the 68-bit output of the multiplier should be sign extended. End-around carry addition will now work correctly for floating-point. For integers, the complementers must be disabled and C needs to be sign extended from 101 to 102 bits.

The partial products, now including the aligned C, are added with `DW01_csa` of which the carry in is forced to `0` and the carry out can be ignored. Due to inclusion of the sign bits, end-around carry addition does not work when the input is zero. A correct result is again obtained by disabling both complementers when the input is zero. Two sequential adders implement end around carry addition (Figure 4.11), because both sum and sum+1 (Section 4.4.3) have to be known. Since the final addition takes place in stage two, a non-critical stage of the pipeline (see Section 4.7).

The modifications needed to incorporate integer multiply-accumulate in a floating-point FMA datapath are actually modest, yet not self-evident.

Leading Zero Anticipation and Leading Zero Detection The LZA and LZD hardware is not particularly interesting for integer use. LZA has been implemented exactly as shown in Section 4.5 and LZD by a variant of Oklobdzija's counting tree, which requires less area and does not affect latency.

Normalize The actual normalization circuit contains a left-shifter which is useful for integer shift-left instructions. Arithmetic shift-left is exactly the same as logical shift-left thus sharing this shifter is straightforward. To obtain a two cycle latency, the sum and shift-accrocount registers have been multiplexed.

Round Floating-point rounding (Figure 4.8) is implemented in 34-bit arithmetic, using the pattern matching logic described in [Woo-Chan et al., 1996]. The 34 bits prevent the significand from overflowing during the increment. If the carry out is `1`, the overflow is easily corrected by adjusting the exponent and using only the 33 MSBs as the significand. Since the functionality of the 66-bit primary adder supersedes that of a 34-bit incrementer, the incrementer was not modified to support integer operands. Rounding is disabled when the last stage is used for integer shift-left instructions.

Backend The backend of the datapath performs the final exception checks and formats the output. During a floating-point multiply-add operation, underflow and overflow can occur in multiple places: during alignment, normalization and rounding. It is possible that an underflowed or overflowed intermediate result can perfectly be represented after being normalized or rounded. Because the exponent datapath is implemented with 9-bit arithmetic, overflow occurs only when the rounded result is found to be outside the representable range. The backend then asserts both the status bits and the output to the corresponding IEEE-754 representation. Note that 32-bit integer multiply-accumulate will never overflow due to the exceptionally wide datapath. The most positive result is still well within the 64-bit output range, which also holds for the most negative result. For convenience, the status registers are therefore used to indicate if the integer result requires more than 32 bits. Lastly, the sign-bit as defined by Equation (4.4) does not always apply when the result, or one of the operands, is zero, infinity or NaN. The backend performs a correction in these cases.

4.7 Realization

To show the feasibility and advantages of Sabrewing, a VHDL structural description has been implemented in FPGA and ASIC technology. FPGA mapping was carried out using Synopsys Synplify, targeting a Virtex-5 LX330T device. The ASIC prototype is based on a fairly standard IC design flow, using Synopsys DesignCompiler for synthesis and Cadence Encounter for layout generation. Both the low-power high voltage threshold (LPHVT) and general-purpose standard voltage threshold (GPSVT) libraries from STMicroelectronics' 65nm CMOS technology have been evaluated.

4.7.1 FPGA

Despite the fact that an FPGA mapping of Sabrewing was never a major objective of this work, it was carried out to verify compatibility with an FPGA. We encountered no problems to synthesize for an FPGA, although the design flow

is restricted to Synopsys tools because of the DesignWare IPs. On a Virtex-5 LX330T, the design operates at 96MHz and occupies less than 2% of the device's resources (3683 LUTs).

4.7.2 ASIC

Because the potential advantages of Sabrewing matter most in ASIC technology, we present the ASIC implementations more elaborately. Every design is based on the default Sabrewing configuration (see Section 4.3) which means no support for NaN, infinity and subnormals. The differences are in the properties of the ASIC technology libraries, either low power with high voltage threshold or general purpose. Scan-chains are included and automatic retiming is applied to optimize the pipeline balance. Since power consumption is a major concern and limitation for embedded systems, the LPHVT library is the main target technology in this survey.

Timing

The critical path caused by the multiplier is located in the first stage. Implemented in LPHVT, the datapath can be clocked up to 667MHz while 1.35GHz is the limit in GPSVT technology. Although such frequencies can be achieved when desired, the clock will most likely be constrained to 200MHz (i.e., the frequency currently achieved in MPSoC DSP-cores [ter Braak et al., 2010] that are candidate for Sabrewing integration). The pipeline is fairly well balanced by itself. However, register retiming still yields an improvement of almost 100MHz in LPHVT and over 200MHz in GPSVT. Implementing IEEE-754 compliant infinity and NaN support (see Section 4.3) incurs a penalty of just 10MHz, when retiming is not taken into account. Support for subnormal numbers has a much larger impact. Even though subnormal support was not fully implemented like infinity and NaN, our initial evaluations show that at least 90MHz must be traded off to prenormalize the input operands.

Area

The area of the default Sabrewing configuration, fully placed and routed with the LPHVT library, is approximately 0.07mm^2 at 667MHz; or 41K equivalent gates. The GPSVT implementation, running at 1.35GHz, occupies close to 0.08mm^2 (54K gates). However, when the clock frequency is brought back to 200MHz, both the LPHVT and GPSVT areas are just 0.04mm^2 (22K and 20K gates respectively). A detailed pre-layout area distribution of the 200MHz LPHVT implementation is shown in Table 4.3. Roughly the same distribution can be found in the other implementations. The residual logic is a good indication of the overhead caused by merging floating-point with integer arithmetic. Since these area figures are based on pre-layout results, they should be considered as

TABLE 4.3 – *200MHz LPHVT (pre-layout) area distribution.*

Component(s)	**Area (μm^2)**	**Percentage** (%)
Sticky-Bit OR-reduction	238	0.7
Backend	381	1.2
Exponent Datapath	432	1.3
Frontend	454	1.4
Comparator	517	1.6
Round	862	2.6
Carry-Save Adder	951	2.9
Leading Zero Detection	965	2.9
Residual logic	1506	4.5
EAC Recomplement	1726	5.2
Main Adder	1852	5.6
Leading Zero Anticipation	2384	7.2
Alignment Shift & Complement	2723	8.2
Normalize	4193	12.6
Non-Combinatorial (Registers)	4223	12.7
Partial Product Multiplier	9750	29.4
Total	**33157**	**100.0**

lower bounds. Supporting infinity and NaN has no notable impact on the area. Subnormal support will increase the area by at least 30%.

Power Consumption

Taking the parasitics of the design layout into account, the power consumption of Sabrewing is summarized in Figure 4.12. This power is based on a 4-tap FIR filter under normal conditions (1.2/1.1V operating voltage and 25°C temperature). An input signal with a large amplitude was chosen to cause high internal switching activity, but more importantly, to ensure that the dynamic range of floating-point is being used. The filtering was performed both in floating-point and integer (fixed-point) arithmetic. As expected, the power consumption of the floating-point filter is higher. Close inspection shows that high clock frequencies result in power inefficiency in the integer path. Integer operations consume almost the same amount of power as floating-point operations at 667MHz because of the aggressive synthesis optimizations needed to satisfy the timing constraint. At this frequency there is also a large difference between LPHVT and GPSVT. The 667MHz frequency pushes the design to its limits in LPHVT, which means many more transistors are required. In addition the voltage threshold of GPSVT is 0.1V lower. Because of this observation one might be tempted to use GPSVT. However, leakage is several orders of magnitude higher in GPSVT. At 200MHz, the leakage is 117nW in LPHVT while it is 286µW in GPSVT. The total

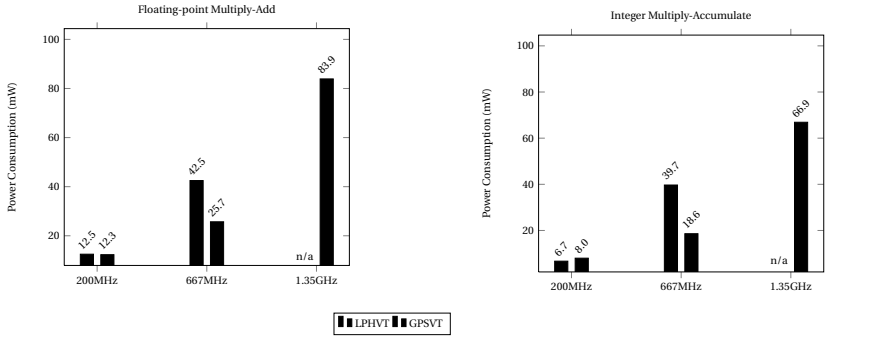


FIGURE 4.12 – *Power consumption based on a 4-tap FIR filter in floating-point (left) and integer (right).*

power consumption is almost the same for GPSVT and LPHVT. Leakage then becomes an important factor, especially in combination with clock gating.

Clock Gating To reduce/eliminate power inefficiency, clock gating can be applied to the datapath. In the 667MHz low-power design, automatically inserted clock gates save approximately 4.5mW in the FMA case and a little over 3mW for integer MAC. We expect that much better results can be obtained by manual clock gate insertion, or by providing the EDA tools optimization directions based on switching activity.

4.8 Evaluation

A realization of the Sabrewing architecture has demonstrated that it has attractive physical properties. This section will provide an indication of how it compares to similar floating-point solutions. At the end of this section we also discuss possible improvements and further research planned for Sabrewing.

4.8.1 Comparable low-power floating-point DSP solutions

It is difficult to make a fair assessment of our new architecture because of its rather unique functionality (Section 4.2). However, to put Sabrewing in the right perspective, some properties of commercially available low-power floating-point DSPs have been listed in Table 4.4. The low-power DSPs of Texas Instruments [Texas Instruments, 2011] and Analog Devices [Analog Devices, 2011] are believed to be the current state-of-the-art.

Although both Texas Instruments (TI) and Analog Devices (AD) claim to be IEEE-754 compliant, thorough inspection of their documentation reveals that this is not entirely true. Like Sabrewing, some shortcuts were taken in both

TABLE 4.4 – *Low-power floating-point DSP overview*

	TI TMS320C674x	**AD SHARC 2147x**
Clock Frequency	200-456MHz	200-266MHz
Floating-Point Support	IEEE SP/(DP) MUL and ADD *- No subnormals* *- Four rounding modes* *- DP not native*	IEEE SP/40-bit MUL and ADD *- No subnormals* *- Only two rounding modes*
Pipelining	Two stage SP MUL and ADD → *four-cycle MAC*	Single stage MUL and ADD → *two-cycle MAC*
Power Consumption	141.2mW at 200MHz 187.8mW at 266MHz 377.8mW at 456MHz	235.9mW at 200MHz 310.0mW at 266MHz

architectures. Neither support subnormal numbers and SHARC only supports half of the rounding modes. Concerning the physical properties, Sabrewing appears to be a step ahead of both DSPs. Either measured in raw frequency or instructions per cycle, both TI and AD are outperformed by Sabrewing. And even though it is impossible to isolate the power consumption of just the arithmetic datapath in these DSPs, the fact that Sabrewing uses less than 10% looks quite favorable. This is why we implemented a reference design to do baseline comparisons.

A baseline conventional approach

Our baseline circuit offers functionality comparable to Sabrewing, but is implemented in a conventional way (no sharing of integer and floating-point hardware). The integer datapath is designed by straightforward VHDL coding and the floating-point datapath is based on a synthesizable version [Bishop, 2011] of the new IEEE-1076 VHDL floating-point support [IEEE, 2009]. The design is not pipe-lined by default. Three registers were instantiated at the input of the design and the tool-flow was used to perform automatic register re-timing of the design. Because the same 65nm process technology and tool flow are used, no assumptions have to be made and a direct comparison is possible. There is, however, one caveat. Although [Bishop, 2011] provides very similar floating-point multiply-add functionality, the results it produces do not adhere to IEEE-754 (about 8% of the tests in [IBM Haifa Research Lab, 2011] fail) and only one rounding mode can be chosen per instance (round to nearest was chosen here). This implementation is not ideal but unfortunately the best that is currently available to us. Sabrewing does produce IEEE-compliant results and is therefore a much more complete design. This has notable impact on its performance and to an even greater degree its area. That is why we expect that the improvements made by Sabrewing are better than portrayed by this comparison.

TABLE 4.5 – *Sabrewing compared to a conventional solution*

		Baseline	Sabrewing	Improvement
Maximum Clock Frequency	LPHVT	425MHz	667MHz	57%
Maximum Clock Frequency	GPSVT	713MHz	1.35GHz	89%
Area at 200MHz	LPHVT	51790μm^2	43640μm^2	19%
Area at max. Frequency	LPHVT	76210μm^2	66653μm^2	14%
Area at max. Frequency	GPSVT	65215μm^2	78086μm^2	-
Integer Power at 200MHz	LPHVT	24.9mW	6.7mW	74%
Floating-Point Power at 200MHz	LPHVT	37.3mW	12.5mW	67%
Normalized Power at max. Frequency	LPHVT	220µW/MHz	64µW/MHz	71%

4.8.2 Performance Area and Energy Comparison

Compared to the baseline, Sabrewing shows improvement in all areas: clock frequency, silicon area and power consumption. Table 4.5 shows the differences in detail. The performance difference can be explained by the many delay optimizations applied to the Sabrewing datapath (Section 4.4). The difference in area is mostly a result of eliminating integer specific logic. The reduced power consumption results mostly from a lower transistor count. No clock gating was applied for this comparison. Note that although the area of Sabrewing is larger at maximum frequency in GPSVT, it should be taken into account that the clock frequency is also practically twice as high.

In addition to the baseline comparison, we have compared Sabrewing to a dedicated and area optimized integer MAC. This MAC was implemented in LPHVT with the clock frequency constrained to 200MHz. According to our expectations, the area of the MAC is smaller compared to the Sabrewing: 11858 μm^2 which is about 27% of the area that Sabrewing occupies. The dedicated MAC consumes approximately 4.3mW to perform the FIR filter that we used to determine Sabrewing's power consumption (Section 4.7.2). This confirms that inefficiencies still reside in the current Sabrewing design. We expect that applying clock gating will be adequate to remedy these inefficiencies. Note that, despite the comparison being made here, Sabrewing is not meant to compete with its dedicated counterparts.

4.8.3 IEEE Compliance

There exists some controversy over what is IEEE-754 compliance. Many manufacturers and researchers label their work as IEEE compliant while in fact they just implement a subset of the functionality defined in [IEEE, 2008]. We would therefore not call Sabrewing IEEE compliant, but it does produce IEEE compliant results. In order to guarantee this, extensive testing was done using a floating-point verification framework from IBM [IBM Haifa Research Lab, 2011]. In contrast to many others, IBM's floating-point test is contemporary and not

randomly generated. It specifically targets the error prone parts of the datapath and includes many test cases for FMA. IBM is pioneer in the field of FMA and they have many years of experience in testing such datapaths. The fact that Sabrewing passes all of IBM's test cases leads us to say that we are confident the results produced by our design are compliant with IEEE-754.

4.8.4 Future Work

The feasibility and advantages of combined floating-point and integer hardware have been demonstrated in this chapter. An important question that remains to be answered is what the architectural implications of this approach are. The first major challenge is instruction scheduling. Further research will focus on scheduling techniques to determine how beneficial combining floating-point and integer arithmetic will be in practice. Exploration of additional hardware features and optimizations is also planned. Although modified Booth multiplication suits the Sabrewing design well, the Baugh-Wooley algorithm might yield even better results [Sjalander and Larsson-Edefors, 2008]. In terms of features, we think vectorized (SIMD) integer instructions and flexible rounding will be interesting for Sabrewing. The Sabrewing datapath is very wide because of the parallel alignment optimization. This makes it suitable to perform SIMD instructions with smaller (integer) operands. Whether these extension are worthwhile or not remains to be seen as they will increase the area and complexity of the circuit and thus partially eliminate the purpose of Sabrewing: being lightweight.

4.9 Conclusion

In this chapter we presented Sabrewing: a new combined floating-point and integer arithmetic architecture designed primarily for low-power embedded digital signal processors. Many DSP applications will map nicely to this new architecture because it offers various floating-point and integer multiply-add operations. In addition, the number of pipeline stages is kept low (three), such that developing efficient compilers for this architecture will not be a very difficult task.

It was shown that a prototype supporting 41-bit floating-point and 32-bit integer operands, implemented in 65nm low-power technology and operating at 200MHz, has an area of just 0.04mm^2. This design's power consumption ranges from about 6.7mW to 12.5mW based on a 4-tap FIR filter performed in integer (fixed-point) and floating-point respectively. Sabrewing can be clocked up to 667MHz in low-power technology and even at 1.35GHz when implemented in general-purpose technology.

By tightly integrating integer functionality in a fused-multiply-add floating-point datapath, at least 19% less area is required compared to a conventional ap-

proach where the same functionality is implemented in two separate datapaths. Compared to the same conventional baseline, a power reduction of 67% can be observed. The amount of area overhead created by augmenting the FMA datapath with integer functionality is less than 5%. Considering all these aspects, we conclude that the Sabrewing architecture will be a valuable asset for hardware platforms that require floating-point arithmetic but are on a tight area and power budget for on-board payload processing.

Chapter 5

Massively Parallel Prototyping Breadboard *

Abstract

Development of payload processing platforms is challenging because of a number of reasons. Their exact use is often not known at the start of the development and they should be usable and maintained over a period of at least a decade. It is not a surprise that development for the harsh space environment takes a long time since there are many procedures involved to guarantee the device works as intended in space conditions. If this cycle is finally completed it is desirable to use the platform for many space missions since development of a new device takes up much time and money and the number of space missions is relatively low. This chapter presents a prototype platform called MPPB, in which different new technologies are incorporated to be used in a future payload processing platform. The platform has been developed as a new starting point from which a future space-grade ASIC could be created and to expose early in the development process possible shortcomings when this platform is used for typical space-related applications.

5.1 Introduction

This chapter describes the MPPB platform that was developed in the Massively Parallel Prototyping Breadboard project, funded by ESA [ESA/ESTEC, 2008*a*]. The project had Recore Systems as its prime contractor and the University of Twente as a sub-contractor.

The project was started as part of a larger project to develop next generation processing platforms for satellite payloads. Several different approaches

*Parts of this chapter have been published in [Walters et al., 2011]

were investigated of which the MPPB platform was one. Other approaches were to use a commercial of the shelf (COTS) DSP and redesign it with radiation hardened chip technology, and to repackage existing hardware to better mitigate radiation. Among the different projects, the MPPB project was the project in which more advanced technologies were explored. Key features to drive the development of the platform were: performance of 1 giga-floating-point-operations per second, highly parallel, ease of use and high throughput I/O. Other features encompassed scalability, energy efficiency and European technology (see also Section 3.8).

All approaches were evaluated and graded in terms of how well they performed, maturity, and when they could be made available as a space grade ASIC by Astrium. Astrium is an EADS company with a long experience in building various satellites and other spacecraft for both science and commercial missions [Astrium-EADS, 2006].

The first section of this chapter describes the platform itself as it is prototyped on an FPGA. Next, several of the subsystems are described in more detail, including the considerations of the design choices. This is followed by a discussion of results from benchmarks that were run on the platform. The chapter ends with work that would be required to get closer to a radiation-hardened platform that can be used in space.

5.2 Prototype Platform

Figure 5.1 depicts a simplified representation of the MPPB platform and Figure 5.2 shows one of three assembled units. The parts depicted in Figure 5.1 are explained in more detail in the following sections. For now its, sufficient to know that the platform consists of the LEON2 and Xentiums, which are the processing elements, the NoC and AHB, which are the main interconnects, and then a lot of peripherals of which the Spacewire (SpW) and gigabit interface (GbIF) have the highest throughput.

The platform itself was defined to test various technologies within a limited time and money budget. The platform was developed on a Virtex 5 board (XpressLX330T) [PLDA, 2008] manufactured by PLDA, chosen for having the largest FPGA with the largest number resources available at the time of development as well as the largest number of I/O pins. Besides the common goal of getting the most in terms of processing power, the project partners had their own focus. The NoC and Xentiums were proposed by Recore and the university, the LEON2 and several of the external interfaces were mainly upon request from ESA.

The reason why the LEON2 [Gaisler Research, 2005*b*] was chosen, is mainly because of the availability of legacy code that exists for this processor and the fact that ESA has a license for the IP. The NoC was proposed because of its ease

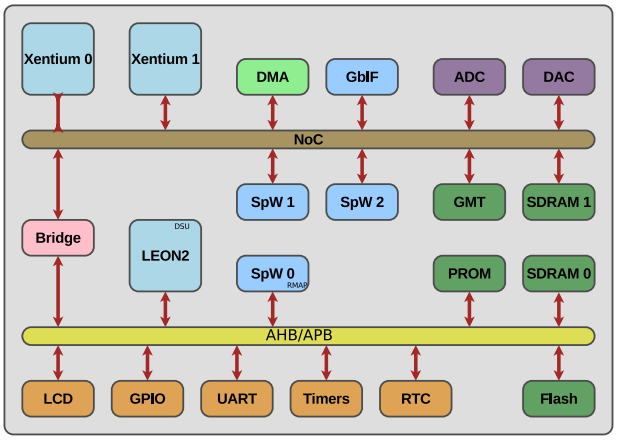


FIGURE 5.1 – *Simple representation of the MPPB platform*

of scaling while the Xentium is one of Recore Systems' DSP technologies. The NoC and Xentiums were also chosen to push new European technology onto a space platform.

It is well known that I/O bandwidth and data transport are critical issues in high-performance multi-core computing. Therefore the platform has quite a few high speed interfaces with the goal of investigating the issues of data transport from the start of the project.

The platform runs at a speed of 50Mhz on the FPGA, while the ASIC is projected to run at a speed of 200Mhz in 65nm technology. The FPGA performance can be extrapolated to estimate the performance of an ASIC realization. As of writing this thesis, an ASIC has been designed in a follow-up project to collect more data on performance and radiation sensitivity. This is done with the DARE180 technology [imec, 2004], within the DARE+ activity [imec, 2012].

As is shown in Figure 5.1, the platform is divided in two main parts, the NoC part and the AHB/LEON2 part, connected by a bridge which takes care of endian [Hennessy and Patterson, 2006] differences. The platform offers high throughput and high processing power on the NoC side. The older and slower interfaces can be found on the AHB interface. The exception is the SpaceWire RMAP connection, SpW 0 in the figure, which was a fairly late addition to the project upon request by ESA. There was already an interconnect fabric available for SpaceWire RMAP [ECSS, 2008] to an AHB and therefore the decision was made to connect it to the AHB within the MPPB platform.

The implemented AHB interconnect is a fairly common connect matrix in which each master has its own layer. This implementation of AHB is also referred to as AHB-Lite [ARM, 2006], without the need for a multiple master ne-

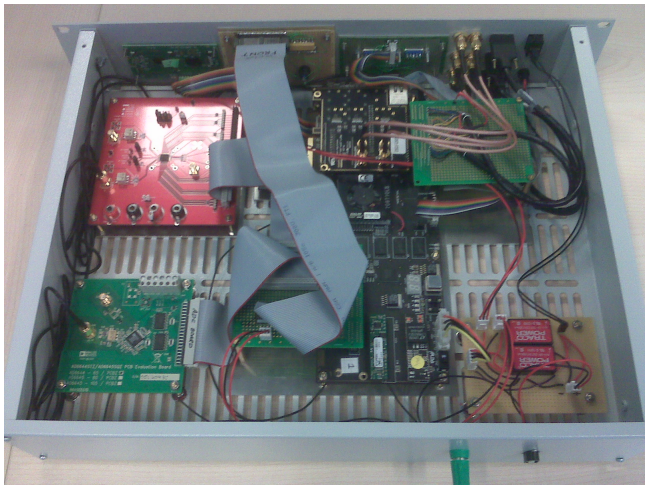


FIGURE 5.2 – *Insides of an assembled MPPB prototype*

TABLE 5.1 – *Device Utilization for 5VLX330TFF1738 (Complete MPPB @50Mhz)*

Resource	Used	Avail	Utilization
IOS	244	960	25.42%
Global Buffers	11	32	34.38%
Function Generators	141677	207360	68.32%
CLB Slices	35420	51840	68.33%
Dffs or Latches	71216	209280	34.03%
Block RAMs	77	324	23.77%
DSP48Es	18	192	9.38%

gotiation scheme since each layer has only a single master. This simplifies the design at the cost of an increase in silicon.

The syntesis results for the FPGA implementation of the MPPB platform are shown in Table 5.1. Table 5.1 indicates that the utilization of the whole FPGA is fairly high, ≈ 69%. Some space is still left but routing constraints make it difficult to map more hardware onto the platform.

5.2.1 Processing

The platform incorporates two different types of processing cores. The LEON2 [Gaisler Research, 2005*b*] and the Xentium [Recore Systems, 2012*a*,*b*] processor. The LEON2 is included because of the availability of space related legacy code and familiarity of use within ESA and space related applications. Also its tool-

chain was already mature while the Xentium toolchain was still in development at the start of the project. While programming on assembly level was not a problem for the Xentium, higher level code and especially control code was an issue at the start of the project. These considerations led to the decision to include a LEON2 as a safe way to implement existing code and use it to offload processing intensive parts of the computations to the Xentiums. At the end of the project an early C compiler for the Xentium was available. This compiler is capable of compiling the full ANSI-C standard, but is, not completely optimized yet.

LEON 2

Two reasons led to the decision to incorporate the big-endian Sparc based LEON 2 processor [Gaisler Research, 2005*b*]. The first reason is that the processor is free to use, and the VHDL is freely available as well. The second reason is that ESA participated in the development of the LEON 2 and already uses the LEON 2 in several other space related projects.

The chosen version of the five stage pipelined LEON2 does not have support for hardware floating-point and uses 16Kb instruction cache and 16Kb data cache. Both caches are configured as two-way associative caches. The LEON2 runs the main control code and is also used to boot the platform. A small bootloader starts either loading code from the UART or from the flash based storage.

Xentium

The Xentium is a little-endian VLIW type processor developed by Recore Systems and available as a separate IP [Recore Systems, 2012*a*]. The MPPB platform currently contains two Xentium tile processors. The Xentiums consume most resources in the platform, each utilizing close to 12% of the FPGA resources in term of LUTs. Synthesis results for a single Xentium for the Virtex 5 FPGA can be seen in Table 5.2. Table 5.2 shows that the resources needed for a Xentium are fairly high, around 12%. What can also be noted from Table 5.2 is that the number of DSP slices used is fairly low. This is because the Xentium is optimized towards ASIC technology and as such the FPGA synthesizer has a more difficult time to extract the DSP tiles which could alleviate some of the CLB slice utilization. One of the main topics to evaluate was the parallelism within the platform. From this perspective, more Xentiums would be desirable. Tables 5.1 and 5.2 show that from a resource perspective it would be possible to add another Xentium and still have some resources to spare. The problem, however, is that it makes routing the design on the chosen FPGA very difficult and when all routing constraints need to be met, the speed would go down considerably.

The Xentium tile as depicted in Figure 5.3 consists of three main parts: the Xentium local bus, the datapath (Core) and the memory (Bank 0 to 3). The Xen-

TABLE 5.2 – *Device Utilization for 5VLX330TFF1738 (Single Xentium @50Mhz)*

Resource	Used	Avail	Utilization
IOS	479	960	49.90%
Global Buffers	1	32	3.12%
Function Generators	24663	207360	11.89%
CLB Slices	6166	51840	11.89%
Dffs or Latches	4145	209280	1.98%
Block RAMs	13	324	4.01%
DSP48Es	8	192	4.17%

tium local bus is an AHB implementation to allow the use of already existing hardware when needed. The datapath is depicted in Figure 5.4. Most operations work on 32-bit or 16-bit two's complement data. The different units offer different functionality.

- The A units offer 2 x 32-bit or 4 x 16-bit adders with 40-bit wide addition registers.
- The C unit is for loop control instructions.
- The M units offer the multiplications for 2 x 32-bit operands or 4 x 16-bit operands.
- The P unit focuses on packing instructions for the different configurations of 16-bit words into 32-bit words.
- The S unit also offer the adders and wide registers like the A units but also includes shift operations.
- The E units offer the LOAD and STORE functions.

The memory provides access to 4 different banks at the same time. This allows the datapath to load and or store 4 32-bit values simultaneously. This bandwidth is needed to make use of all different parallel execution units in the Xentium.

5.2.2 Peripherals

The platform offers several interfaces to peripherals. The main building blocks will be discussed in more detail. Several different interfaces have been placed on the platform to see what their degree of utilization is when they are operated together in a complete system. It is not expected that all of them will be active at the same time or that all of them will be implemented on a future ASIC.

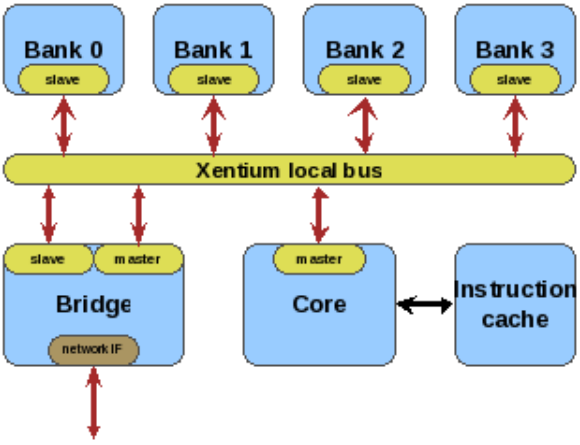


FIGURE 5.3 – *Xentium localbus*

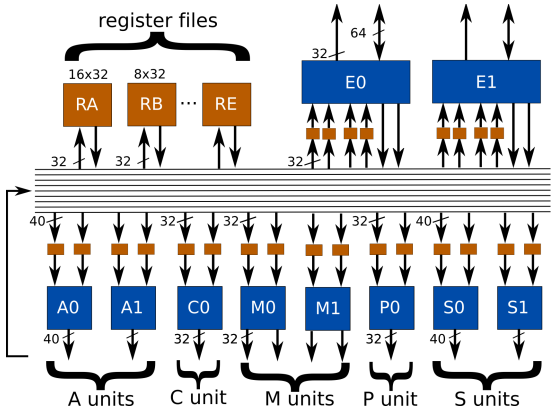


FIGURE 5.4 – *Xentium datapath, Copyright Recore Systems.*

Network on Chip

Before discussing the peripherals, the NoC is presented. Figure 5.5 shows a simple representation of the NoC part of the MPPB as it is used in the platform. It connects the following devices:

- twelve routers (R),
- two Xentiums (Xentium 0 and Xentium 1),
- a bridge to connect the ADC and DAC (ADC/DAC bridge),
- an eight channel DMA engine (DMA),
- two Spacewire connections (SpW1, SpW2),

- a gigabit Aurora interface (GbIF),
- a DDR controller (SDRAM controller),
- and a default slave which acts as a sink in case an illegal NoC address is used,
- AMBA Subsystem (Bridge in Figure 5.1).

All high bandwidth demanding peripherals are connected to the NoC, with the exception of the LEON2, Spacewire Remote Memory Access Protocol interface (Spw0) and some of the DDR (SDRAM 0) which are connected to the AHB.

The idea is that the LEON2 controls the flow of the data while the data itself never leaves the NoC part of the platform. The control of the flow is nothing more than configuring the DMA transfers and restarting them when they complete.

The network-on-chip is a packet switched network with XY routing [Wolkotte, 2009]. XY routing is a simple way of routing packets through the network, where first the X (horizontal) direction is chosen in the network and the Y (vertical) direction afterwards. For this reason, the forward and return path can be different. Each router has 5 ports and serves 4 channels on each port, each of them with different priorities. Each channel offers a bandwidth of 1.6Gbit/s at 50Mhz. Two channels are dedicated for DMA transfers (high priority) while the two others are for single read and write operations and interrupts.

All memories and registers are accessible via memory mapped I/O using a single memory map. Both NoC and AHB support this feature which simplifies programming considerably.

SpaceWire

SpaceWire [ECSS, 2008] is a serial link based on LVDS developed on the initiative of ESA to provide a robust link between different instruments and processing elements on board of a satellite payload. It runs at speeds of 2Mbit to 400Mbit/s. It offers point to point connections as well as options to connect several devices together via a network. The protocol is fairly simple in which an absolute or relative target address preceded the data. It offers flow control such that speed is throttled down in case receiving hosts are not fast enough to process the incoming data.

The FPGA implementation on the platform offers a speed of 100Mbit/s, due to limitations of the prototype technology. The SpaceWire connection on the AHB bus also offers a so called *target* Remote Memory Access Protocol (RMAP) functionality [ECSS, 2008]. The RMAP on SpaceWire makes it possible to remotely access any of the readable memory locations by only providing the memory address and size of the desired transfer. The term *target* indicates that the device can only be accessed from another device and that the device itself cannot initiate RMAP transfers, a bit like the master and slave relation on a bus.

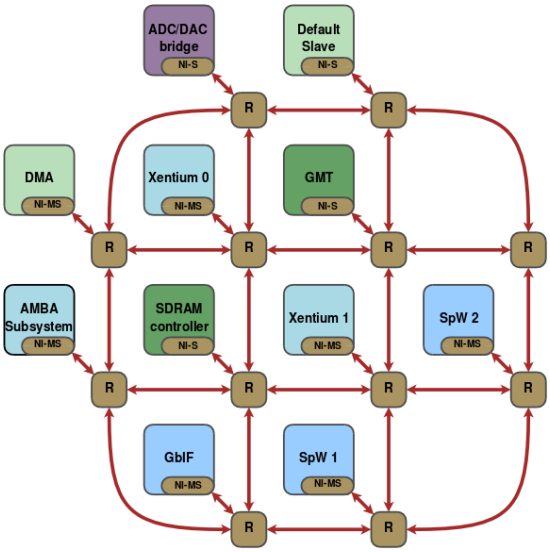


FIGURE 5.5 – *Simple representation of NoC*

Memory

There are five different types of memory for general use on the heterogeneous platform, which are listed in Table 5.3. Each memory type has different performance properties. The fastest memories are the SRAM memories and the slowest the DDR and FLASH based storage. Selection of what memory to use is mainly based upon the performance of the algorithm that needs to run on the platform, location on the NoC and, in the end, preference of the programmer. Table 5.3 specifies all memories and their respective delays in cycle counts from the Xentium. The cycles are the cycles required to retrieve data from their respective memory locations. Each of the routers in the network adds a cycle of delay as well. The delays of the DDR memories are a worst case single read indication since they support burst transfers as well as single word transfers which makes the cycles mentioned more dependent on what kind of transfer is performed.

Bridge

The bridge takes care of the translation from NoC traffic to AHB traffic as well as taking care of the endianness conversion. The data and processing elements on the NoC are little-endian while the data and elements on the AHB operate in big-endian. The conversion between both is done in hardware to make it easier

TABLE 5.3 – *Memories and delays from the Xentium*

Memory	Cycles delay from the Xentium
Xentium local memory (2x) 32kB SRAM	1
Generic Memory Tile 256kB SRAM (GMT)	32
256MB DDR on the NoC (SDRAM 0)	44
256MB DDR on the AHB (SDRAM 1)	170
1GB NAND flash on the AHB	>500

for the user of the platform.

Gigabit interface

The gigabit interface (GbIF) is an implementation of the Aurora protocol [Xilinx, 2010] which is available on the FPGA. In theory, any other device which implements this protocol could be connected to this interface at a speed of 1.1 Gb/s. On the platform, it is used to connect two boards and as such to extend the platform and increase the processing power and I/O capabilities of the whole setup.

Data Converters

The Analog to Digital (ADC) and Digital to Analog converter (DAC) on the board are there to interact with high speed sensors and actuators. Both interfaces are on separate daughter boards connected to the FPGA. The ADC is an Analog Devices AD6644 [Analog Devices, 2007], which samples at a maximum rate of 40 MS/s where each sample consists of 14-bit. The DAC is a Texas Instruments DAC5662 [Texas Instruments, 2006], which runs at a maximum speed of 40 MS/s as well but with a width of 12-bit. The interfaces of the NoC allow them to be controlled at different sample rates, varying from 190.74 S/s to 40MS/s.

Miscellaneous interfaces

The LCD, GPIO, UART, Timers and RTC, depicted in Figure 5.1 are not discussed here. A small explanation as to what they do and why they are on the platform is provided in Appendix B.3.

5.2.3 Tooling

Although tooling is not the main topic of this thesis, this section is included for completeness. Software development for the platform requires the use of two toolflows: one for the code to be executed on the LEON2 and one for the code to be run on the Xentiums.

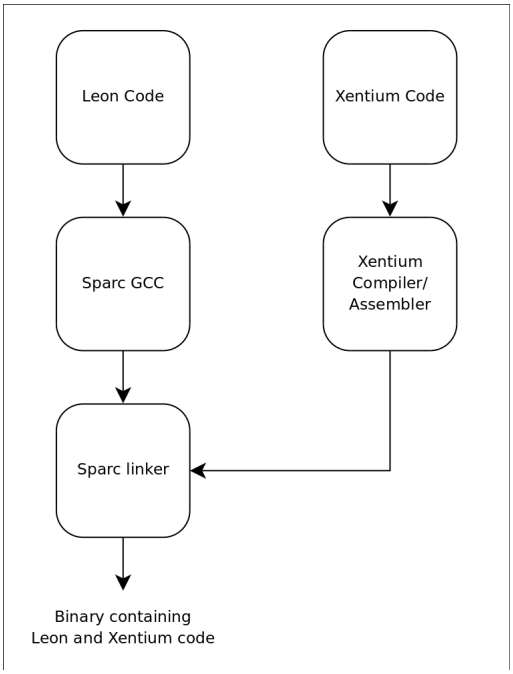


FIGURE 5.6 – *MPPB code development flow*

The two toolflows can be combined in the Eclipse IDE [Eclipse, 2002] or in a Makefile. This makes it relatively easy to use both tools even though they are intended for different architectures.

Both the Xentium and the LEON2 tooling offer a simulator to simulate the generated code. Both the LEON2 and the Xentium simulators offer interaction via file I/O.

This makes it possible to simulate stand-alone binaries. For debugging and optimizing single data processing kernels on the Xentiums, this is quite sufficient. Appendix B.2 provides a bit more insight into the current compilation process.

5.2.4 Requirement compliance

One of the requirements mentioned at the end of Chapter 3 is a performance of 1 GFlops per second. However, as explained in Chapter 3, no floating-point IP was directly available. For this reason, it was decided to start two developments in parallel: a prototyping platform based on integer arithmetic, but incorporating a modern VLIW processor and a network on chip on one hand and a floating-point ALU on the other hand.

The VLIWs offer a high integer performance. Also the NoC offers relatively simple design time scalability. The high integer performance can still offer enough software floating point performance in certain cases. And foremost if the platform is a success, a core with a dedicated hardware floating point unit could still be added. Testing the new technologies, especially radiation hardening requires a lot of time, therefore starting these tests a soon as possible was essential.

5.3 Benchmarks

Some typical space applications have already been discussed in section 3.5. This section provides performance numbers for these algorithms run on the MPPB platform. The chosen FIR lengths and FFT sizes and other parameters are based on the specifications provided by ESA in their benchmark document [ESA/ESTEC, 2008*b*].

5.3.1 ADC / DAC processing

In this benchmark several FFTs and FIRs of different size are run as well as a benchmark with unprocessed data in which the input of the ADC is directly routed to the DAC. The data streams are set up and maintained by the LEON2 processor while the Xentium does the actual processing of the data. The results of the different ADC / DAC benchmarks can be found in Table 5.4.

Unprocessed data

For this test the data is sent from the ADC to the GMT (Generic Memory Tile) and from there to the DAC. The data rate obtained is indicating by how fast the system can handle the incoming data. Since there is no data processing involved, a data rate of 40MS/s can be achieved, which is also the maximum sample speed of the ADC. All buffer sizes are 1KWords in size.

FIR Lowpass

Three low-pass filters of lengths respectively 16, 64 and 256 taps are used. For these tests the ADC fills a buffer of 1KB that is located in the memory tile (GMT) of the MPPB platform. The data is processed consecutively by the Xentium performing the FIR filtering. The filtered results are stored in the NoC-DDR (SDRAM1) and subsequently sent to the DAC interface. The ADC switches between two buffers (ping-pong buffers), as does the Xentium. The FIR filter has state, the state is saved in the Xentium itself such that continuous behaviour of the FIR filter is maintained. Figure 5.7 depicts the flow of the data. The

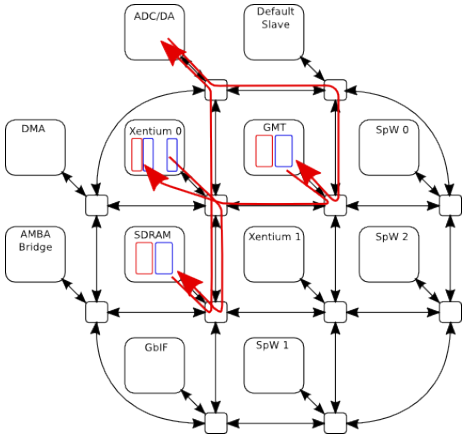


FIGURE 5.7 – *FIR data flow on the NoC*

control flow is not depicted but it is dealt with by the LEON2 and only involves several bytes of control data.

FFT 1024 / 4096

For these tests the ADC fills a buffer of 1 Kwords or 4 Kwords of the GMT for FFT-1024 or FFT-4096, respectively (1024 respectively 4096 point FFT). The buffered data, located in the memory tile of the MPPB platform, are processed by the Xentium, performing the required FFT. The output results of the FFT processing are stored in the NoC-DDR. When a total of 5*64Kbyte of output data is stored in the NoC-DDR, the data is sent to the SpaceWire 0 interface. The benchmark PC is connected to the SpaceWire 0 interface, where the output results are collected and analyzed. The data stream is depicted in Figure 5.8. The FFT algorithm implemented on the Xentium performs a complex FFT. Each 32-bit word from the ADC contains two real samples. During FFT processing in the Xentium, the ADC input data is used for the in-phase (I) component of the input data, while zeros are introduced for the quadrature (Q) component of the input data. The zeros are introduced since the input consists of only real data and the FFT assumes complex values. This means that the data rate of the ADC is half the data rate of the actual throughput of the FFT.

FFT 1960

The test with the 1960 FFT is similar to the tests with the FFT-1024/4096 in terms of data flow, which is depicted in Figure 5.8. However, the Xentium now performs a 1960-point FFT. The 1960 FFT is used in the benchmark to determine the ability of the architecture to handle higher radices as well as high irregular-

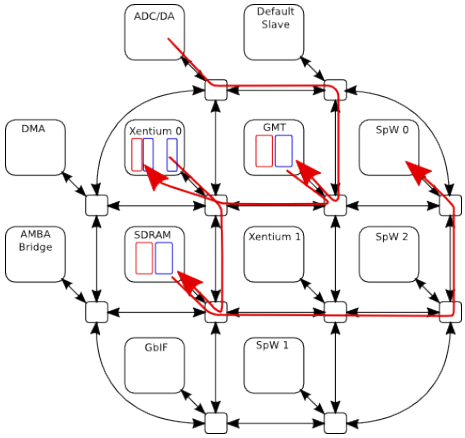


FIGURE 5.8 – *FFT data flow on the NoC*

TABLE 5.4 – *Benchmark FFT and FIR on MPPB platform*

DAQ processing	Max Data rate	FIR exec time (1k samples)	FFT exec time
Unprocessed data	40 MS/s	NA	NA
FIR 16 taps	6 MS/s	$1.7 * 10^{-4}$s	NA
FIR 64 taps	2.5 MS/s	$4.1 * 10^{-4}$s	NA
FIR 256 taps	730KS/s	$1.3 * 10^{-3}$s	NA
FFT 1024	15.3MS/s	NA	$6.6 * 10^{-5}$s
FFT 1960	8KS/s	NA	0.2s
FFT 4096	865KS/s	NA	$5.0 * 10^{-3}$s
FFT 4096[a]	7MS/s	NA	$1.2 * 10^{-4}$s

[a] theoretical speed in case of sufficient memory

ities in data access. The 1960 FFT is implemented as a mixed-radix FFT with a mix of power-of-2 FFTs and non-power-of-2 FFTs. The 1960 FFT algorithm code running on the Xentium has been generated using the Xentium C compiler software.

Evaluation

The unprocessed data processing in which the data from the ADC is directly routed towards the DAC shows that the NoC can easily deal with the throughput which is required. The bottleneck in this case are the ADC and the DAC themselves.

The FIR filters are efficiently implemented for the Xentiums. For the FIR filters the performance of the Xentiums is the limiting factor on the MPPB. For

a future ASIC, the FIR filter processing speed is expected to scale with the speed of the Xentium up to the point where the throughput speed of the target data location becomes the limiting factor.

Power-of-2 FFT algorithms are efficiently implemented on the Xentium processor and, hence, on the MPPB platform. However, the FFT-1960 is a mixed-radix implementation which factorizes poorly; for example 2x2x5x7 or 4x5x7 makes this really difficult to map on a regular DSP architecture such as the Xentium. It was concluded that it is still possible to implement these mixed-radix FFT algorithms on the Xentium, based on a high-level C description. The results are listed in Table 5.4. However, it is expected that the performance of the 1960 point FFT can be improved quite a bit with more development effort.

For efficient implementation of power-of-2 FFT algorithms it is beneficial that the local data memory of the Xentium matches with the needs of the algorithm. For instance, the FFT-4096 requires more data storage than the internal memory of the Xentiums can offer in the MPPB platform. Therefore, the memory tile in the MPPB platform is used as an external buffer (i.e. external for the Xentium). Since data transfers outside the Xentium are required, this reduces the performance of the FFT-4096 significantly compared to e.g. the FFT-1024 implementation. A solution to circumvent the problem of insufficient local data memory size would be to distribute the FFT code itself over multiple Xentiums in the MPPB platform or make the internal memory of the Xentium larger to accommodate for larger FFTs. In the current MPPB platform, the local data memory of the Xentium processor is sufficient for power of two FFTs up to 2048 points. The last row in Table 5.4 shows the theoretical performance of an FFT 4096 when the Xentium would have had double the memory, 64kB. The theoretical value is extrapolated based on the value of the 1024 FFT. It would be possible to get close to this performance when the 4096 FFT would be split into two 2048 FFTs over the two Xentiums. In this case two 2048-FFTs would be performed after which an extra recombination stage would have to be performed.

5.3.2 Image data compression

In this benchmark an image is compressed which is loaded via SpaceWire as if it was coming from an on-board camera.

Two different image sizes are processed. A 1024 by 1024 image, which is a representative size for an image coming from a large CMOS sensor. The second image is a 5148 by 8 pixels image which is representative for a *push broom* sensor.

The push broom sensors used in observation satellites move across the surface of a planet. By the movement of the sensor in orbit, it can construct a larger image. In this case, the sensor samples 8 rows of 5148 pixels at a time and as the satellite platform moves, the next 8 rows can be read and as such creating a theoretically unlimited size image.

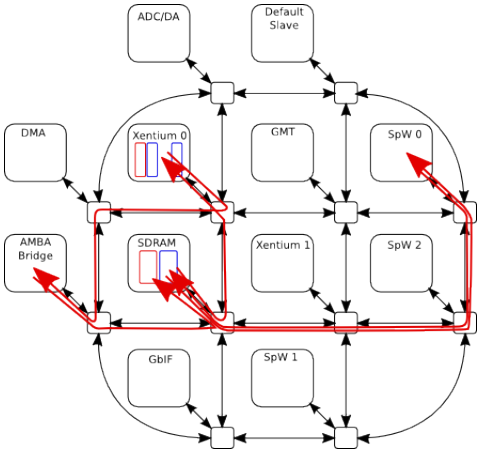


FIGURE 5.9 – *DWT data flow on the NoC*

Figure 5.9 shows the data flow of this benchmark. The data for the benchmark is loaded from SpaceWire 0 and put into the NoC-DDR. The data is 8 bit unsigned integers in a binary format.

From the NoC-DDR data is transferred via DMA to the Xentium in segments of 256x256 pixels for the 1k image and segments of 32x32 for the 5184 image. The 5148 image is transformed from 8 rows to 32 rows. This is done since the standard that is used for processing requires a minimum number of rows of 17 [CCSDS, 2005]. The standard also allows to replicate the values on the edges to acquire the necessary number of rows and columns. The algorithm that is implemented on the Xentium only works for powers of two, since that is most efficient. This makes 32 the closest power of two which still processes all the data. In case of reconstructing the image from the wavelet coefficients, which is not included in this benchmark, the extra rows are discarded. Although this seems like an inefficient procedure, a practical implementation would need to process more than 8 rows of data and as such make full use of the 32 row processing.

The Xentium then computes the wavelet transform on these pieces of the image. Subsequently the results are sent to the main memory for encoding and compression by the LEON2. The results of this benchmark are listed in Table 5.5.

Evaluation

The main bottleneck for this benchmark is the encoder which is run on the LEON. There are a lot of bit-wise operations and a lot of searches over all the data which makes this a relative slow part of the benchmark. The code was

TABLE 5.5 – *Benchmark Wavelet transform*

	Execution time DWT (Xentium)	Execution time encoder (LEON2)	Compression ratio	Block transfers
1024 × 1024	591ms	87s	1.4	16
5148 × 8	175ms	15s	1.2	163

TABLE 5.6 – *Benchmark decimation and compression*

	Execution Time
Filtering (Xentium) 5.8MS/s	1.7 s (10M samples)
Compression (LEON2) 44KS/s	732ms (32.768 samples)

written from scratch with only the standard as a reference [CCSDS, 2005]. Due to time constraints and overall lack of experience with this kind of code, it is far from optimal. What can also be noted is the fact that the amount of data transfers needed, influences the benchmark considerably. The 5148 × 8 image is done with a lot of small transfers (163) which increases the overall processing time while the 1024 × 1024 pixel image is done with only 16 transfers of data.

5.3.3 Decimation and compression

During this benchmark, the data coming via SpaceWire is filtered and decimated, after which it is compressed and sent back via SpaceWire, see Section 3.5.2, Section 3.5.3 and Section 3.5.4. The compressor is a Rice based compressor [Rice and Plaunt, 1971]. For this benchmark 10M samples of input data is generated by Matlab and stored in a binary file.

The data is sent via a PC on SpaceWire port number 0. The PC waits until the processing is done and receives the data back on the same SpaceWire port. The platform receives and stores the data in the NoC-DDR. When all the data has been received it is processed in blocks of 1k bytes by the Xentium. Digital filtering and, subsequently down-conversion, of the input data is done by means of a 128-taps FIR filter. When the decimation of the data is done, the results are put back into the NoC-DDR. Finally the data is compressed by the LEON2 via Rice Coding. The dataflow is the same as the previous benchmark as depicted in Figure 5.9.

Evaluation

The results of this benchmark are listed in Table 5.6, which makes clear that the LEON2 is the bottleneck in the processing chain. This is not entirely surprising since the LEON2 does not have such a high throughput in the first place and has to perform an algorithm which requires a lot of operations. An implementation

TABLE 5.7 – *Benchmark demodulation and decimation*

	Maximum throughput	**Processing Latency**
Overall (Demodulation and decimation)	4.8 MS/s	17246 cycles per 1000 samples

of a Rice encoder could be made for the Xentium but it would require a lot of code optimization by hand to achieve a large advantage over the LEON2. Nevertheless this might be worth the effort since the Rice encoder is used in quite a number of processing chains, each of which would benefit from a high throughput implementation.

5.3.4 Demodulation and digital down conversion

The benchmark consists of two parts, I/Q demodulation and digital down conversion. The I/Q demodulation function demodulates the input signal in a real and imaginary part (i.e. in-phase and quadrature component). The I and Q parts are subsequently filtered, up-sampled and down-sampled.

For this test, the input data is generated by Matlab and stored in a binary file. The data is sent via the benchmark PC on SpaceWire port no. 0. The benchmark PC waits until the processing is done and receives the data back on the same Spacewire port.

The platform receives the data which is stored in the NoC-DDR. When all the data is received it is processed in blocks of 1k bytes by the Xentium. When the demodulation and decimation of the data is done, the results are put back into the NoC-DDR. Then the data is sent back to the benchmark PC, again the data flow is similar to the two previous benchmarks, as shown in Figure 5.9.

The I/Q demodulation is implemented with an 8-point Hilbert filter [Lyons, 2010]. The 80% data reduction is implemented as an optimized 64-point FIR filter in which some of the samples are discarded. The I/Q demodulation and Decimation Filter have been implemented in a single Xentium program.

The synthetic data, 10M 16-bit samples, required 2.07 seconds to demodulate and decimate. This excludes the actual transfer time of the data via the SpaceWire port.

Evaluation

As with the previous benchmark, there might be a possibility to further optimize the algorithm. One can think of using two Xentiums and pipelining the implementation. This could speed up the process but more DMA transfers would be needed. This overhead could cancel the possible speed-up.

5.3.5 Software floating-point

Although the ESA benchmark does not explicitly include a floating-point performance benchmark it is of great importance to them. Therefore this section describes the software floating-point performance of the MPPB platform. Currently the Xentiums and the LEON2 on the platform support software floating-point operations via linking in the appropriate library. The integer hardware is then employed to perform the floating-point operations, and hiding the actual mapping for the programmer. This is a perfectly viable strategy and it has been done successfully on a wide variety of hardware architectures where hardware floating-point support is missing. Section 2.2.4 together with Chapter 4 outlined the operations needed to perform a floating-point operation. Most of these operations can be done in software without any problem. An issue arises when the bit-width needs to be wider than the actual datapath. As a reminder, the Xentium has a datapath width of 32 bits. For a single precision floating-point operation in software there is not so much of a problem. The 23 bits significand and 8 bits exponent fit inside the 32 bits datapath. The most troublesome part is the alignment together with the rounding. This requires bits beyond the 23 bit significand. With a 32 bit datapath this can still be solved since there are 9 extra bits beyond the 23 bits of which 3 can be used to do the rounding with.

In case of double precision floating-point numbers, this changes the situation a lot. For double precision, in which case the significand is 52 bits wide, the calculations take up multiple registers and operations in the 32 bit datapath. A single bit shift suddenly requires a lot of calculations; extra loads to get the parts of the significand and checking overlapping parts for the bits that shift out of one part of the significand into the next part. This is only some of the overhead involved regarding a single bit shift. Multiplications, additions and other more complex operations take a lot more integer operations to complete. These simple examples already show that employing software floating-point should not be taken lightly, and may take more effort than expected.

Table 5.8 illustrates the overhead of single precision floating-point a bit more. There are some caveats with these numbers though. Currently the compiler of the Xentium does not take full advantage of all the parallel arithmetic units. Therefore the presented numbers are a first indication of what to expect and improvement can be expected.

LLVM refers to the compiler toolkit which supplies a library that takes care of the floating-point emulation. FLIP it the Floating-point Library for Integer Processors [Jeannerod and Revy, 2012; Jeannerod et al., 2010]. This is a library especially targeted towards VLIW type processors. The library has been developed by the Laboratoire de l'Informatique du Parallélisme in Lyon France. They used the ST200 family of STMicroelectronics as a validation platform. The ST231 [STMicroelectronics, 2009], a four issue slot VLIW, is used as a reference for this library.

TABLE 5.8 – *Cycle counts for single precision floating-point*

	Xentium [a]						ST231	Leon GRFPU
	LLVM			FLIP			FLIP	FP Hardware
Opr.	Min	Max	Avg	Min	Max	Avg	Max=Min=Avg	Throughput
add	47	101	85	85	105	92	26	1
sub	57	114	95	87	108	94	26	1
mul	44	70	54	78	96	85	21	1
div	47	107	80	88	124	117	32	16
sqrt	47	339	247	36	80	60	23	24

[a] Not optimized, preliminary indication

In both cases an minimum, maximum and average is provided for the Xentium because the number of steps in the algorithm is data dependent. This is not that difficult to understand, for example adding two zero values together takes far fewer cycles than adding two values together with a high difference in exponent value. In case of the ST231, only a single set of values was available in which they were listed as max, min and avg. There was no clear explanation for this, it could be that the code is optimized in such a way that there is little to no branching in the code and as such creating a single execution path. A single path of execution in the code would explain that avg, min as well as max provide the same number of cycles.

The last column is the hardware floating-point unit from Gaisler which can be used in combination with a LEON2 processor. This hardware floating-point unit is not used on the current MPPB platform.

Texas Instruments also provides software floating-point libraries with their integer DSPs [Texas Instruments, 2003]. The results of these libraries are provided in Table 5.9. Texas Instrument's default libraries offers the normal floating-point operations, the FastRTS library, however, is optimized assembly code. The FastRTS library does not handle underflow or de-normalized numbers, it returns zero in this case. It also does not handle overflow, it returns plus or minus infinity in that case. Exceptions are also not given. This allows for very optimized code which is reflected by the lower cycle count for this library.

From Table 5.8 and Table 5.9 it is clear that, although it is nice to have the ability to do single precision floating-point operation, it should be used with care. Another disadvantage of software floating-point is code size. The binary which employs floating-point operations is larger and therefore takes up more space in memory, which by itself is not such a problem. However, the jumps in the code together with the large binary are. The reason is that DSPs come with a limited amount of instruction memory. Ideally everything of the computation kernel should fit inside the instruction cache. This way, no extra code fetches

TABLE 5.9 – *Single precision float cycle counts Texas Instruments 62x architecture*

Instruction	Default	FastRTS lib
addf	81	29
sub	37	18
mpy	35	18
div	109	32

are needed and the algorithm runs at full speed. With floating-point emulation, however, depending on the operation and the input operands, new code needs to be fetched every now and then. Fetching new code into the cache takes time and can stall processing. So, while a floating-point operation might be in the order of a 100 cycles, it might take longer if the processor is stalled due to instruction cache misses. Table 5.3 gives an impression of the additional delays that might occur depending on where the instructions are located.

5.4 Future work

The MPPB platform has successfully contributed to more insight in the design of a multi-core DSP architecture for space applications. This section describes main issues which would need attention for a future ASIC.

5.4.1 Data transfers

Data transfers in the platform are performed with the help of separate DMA engines. This construction frees up processor time and also allows for more efficient data transfers. Currently, this is implemented in such a way that the LEON2 processor assigns a source, destination and size after which the DMA engine takes care of the actual transfer and reports back when it completes. The notification comes in the form of an interrupt back to the LEON2. In a typical scenario the LEON2 restarts the same transfer or similar one with an address offset. At first glance this might appear like an elegant solution because the LEON2 does not need to take care of the transfer and gets notified when the transfer is done. In practice, however, this mechanism proves to be troublesome.

A common scenario is that data coming from the ADC is processed by the Xentium and subsequently sent out to an interface. Notice that this scenario does not include the LEON2 itself. The LEON2, however, is responsible for setting up the DMA transfers from the ADC to some buffer memory, from that memory to the Xentium and from the Xentium to the interface. Already with this simple scenario there are three different DMA transfers involved, which

might be even more if the target interface requires some buffering or when double buffering scenarios are preferred. The rate at which the LEON2 gets notified of the completion of the transfers can be quite high. For example the ADC can run at a speed of 40 MS/s with an effective word width of 32bits and a target size of 8Kb (size of a single Xentium memory bank) which already triggers 10k interrupts each second. With 10k interrupts each second the LEON2 has roughly 5k cycles to handle each interrupt. While this seems enough this example only covers a single transfer and the simple example already requires three transfers. This makes it clear that it will not work if high data rates are required, small buffers are needed and if on the LEON 2 a time-consuming algorithm is running. This is quite prominent when a scenario is employed which actually uses the LEON2 in a software pipelined fashion in the processing chain.

To alleviate the bottleneck of the LEON2 controlling all the DMA transfers in this scenario, it would be better to have each processing element be responsible for its own data transfer. In this way, the LEON2 could still be used to set the parameters of the DMA transfers but the subsequent repeat transfers would be set by the source or target of the DMA transfer. By handling the transfers in this way, there is never a single point which needs to handle all the interrupts. It would require some extra hardware but a simple restart possibility in the engine itself would only have a minimal impact on size.

Another solution would be to get rid of interrupts all together. A way to implement this is to have pre-calculated buffer sizes in such a way that there is never an overflow or under-run of the buffer itself e.g. via a back-pressure mechanism. This is a common way of implementing streaming applications. [Bijlsma, 2011]. However, the pre-calculation of the buffer sizes requires an almost fully deterministic system. The current system is difficult to profile and therefore it is difficult to exactly determine what the sizes of the buffers should be. Certainly in cases where access is required to a shared physical memory location, which could hold multiple buffers, this solution is limited since the arbitration is difficult to determine. A cycle-accurate simulator for the whole platform would be needed to completely trace a program and determine its behavior.

5.4.2 Simulator

A cycle-accurate simulator for the current platform takes a lot of development effort. Besides the fact that a simulator would take a lot of time to develop, this platform is a prototype and certainly not a production ASIC. For this reason, a cycle-accurate simulator is probably not worth the effort unless a simulator could be developed that is flexible enough to be altered when a production architecture has been decided upon. If, however, a cycle-accurate simulator is required, there is the possibility to use the RTL of the system and simulate it this way. The downside to this is that when distributing the RTL code, there is a large

risk that IP modules become available for a broad community unintentionally. Another reason for not wanting to do a full RTL simulation is the simulation speed, which is really low.

5.4.3 Memory

The platform offers different memories with different performance properties which makes it not directly intuitive which memory to choose as a programmer. The intuitive step to choose a memory is in most cases to select the one that is closest to the processing and go from there. The penalty, however, for selecting a memory that is further away is not always clear, especially in combination with the aforementioned data transfer issues in section 5.4.1.

For the different memory locations, the performance of the algorithm needs to be measured and, more importantly, improved where possible. This ties in with the tooling which is currently available. There is no way to determine the exact timing of an algorithm deployed onto the platform and in that way select the best buffer sizes and memory locations. The absolute minimum duration of a separate kernel running on the Xentium is easy to determine via the simulator but difficult to predict when actually running this kernel as part of a complete program or system.

Most of the time the run-time of an algorithm can easily overlap with the data transfers. However, when this becomes time critical, with high data rates and short run times of the kernel, overlapping data transfers prove to be more difficult. Currently a program is optimized until it runs fast enough, which may take some iterations and deployments of the code onto the platform. But what happens when "fast-enough", just is not fast enough? This is where more feedback for the programmer would be needed. Currently, it is not possible to know if or when a memory transfer is blocking another transfer. This information is needed to achieve a better performance [Akesson, 2010].

The exact timings of the memory could be made available to the programmer but this still would not make programming any simpler for the platform. It would provide programmers some more information and maybe some more insight into which memory would be best for their algorithms. This does, however, still make it difficult to figure out conflicting memory accesses. As discussed in section 5.4.1, a full simulator would be nice to have.

5.5 Conclusion

The MPPB platform as has been prototyped on the FPGA is a good technology demonstrator for future payload processing. The ability to have a modern SoC with technologies as Spacewire together with a modern VLIW and NoC make it possible to assess current payload processing software as well as future ones.

The benchmarks as they have currently been implemented, can still be optimized further.

As it stands now, the platform, although it comes with a strong set of tools, is difficult to profile. This means that it is difficult to predict how well an algorithm actually performs on the platform and if there is any room for improvement.

Although an operational space signal processing application has not been developed for the platform, a large set of benchmarks consisting of representative parts of such a system has been successfully implemented.

The platform shows that relatively older hardware like the LEON2 and AMBA subsystem can be combined well with newer technologies like a NoC. It also turns out that Spacewire can be attached to a NoC while giving a good performance. The programming interfaces combined in Eclipse make it relatively easy to develop software for the platform even though the system is quite complex.

The fact that a subset of MPPB is being developed in a radiation hardened technology makes it possible to further test and profile the platform.

Chapter 6

Nectar: Integrating Sabrewing

Abstract

New hardware platforms are often developed with the help of FPGAs, but the step towards ASIC has to be made for deep-space missions. This chapter presents ASIC synthesis results for parts of the MPPB platform as well as for a small SoC platform incorporating the Sabering ALU, called Nectar. The Nectar test platform shows the value of Sabrewing when integrated into a DSP. The results show that the integration of the Sabrewing leads to a marginal area increase (15%) while providing a tremendous reduction of energy for floating-point operations (98%). Suggestion for further improvement of Sabrewing as learned from Nectar are also presented, which mainly revolve around 16-bit SIMD type operations. Based on the experience with Nectar, the chapter concludes with suggestions for a full-fledged multi-core architecture for payload signal processing.

6.1 Introduction

Chapter 5 provided the results for an FPGA implementation of the platform. This chapter continues with results targeting a future ASIC. One of the requirements for the next generation DSP for ESA is performance of one giga-floating-point-operations per second. The following sections will show why this is a difficult metric to obtain with the current MPPB platform based on the results of the ASIC synthesis.

The main topic of this chapter is the integration of the Sabrewing ALU, introduced in Chapter 4, into a Xentium as used in Chapter 5. The modified Xentium together with a LEON2 CPU haven been used for a testbed called Nectar. The results from Nectar as well as the synthesis results from Sabrewing are presented. They show that it is possible to get beyond the giga-floating-point-operations per second. Based on the experiences of the Sabrewing inside the Xentium some alterations to the Sabrewing ALU are proposed. This chapter

concludes with a proposed platform which is sufficiently powerful to serve in future space missions.

6.2 Towards an ASIC

There are two aspects concerning the evaluation of the MPPB: on one side, the direct functional evaluation of the FPGA prototype platform (Chapter 5) and, on the other side, the projection towards a space-grade ASIC, to be presented below.

Space-grade technology libraries are not that easy to obtain, and especially when the technology should be European. Common technology libraries often used in commercially available products are a little bit easier to obtain and therefore used as a starting point.

Three technologies were used to synthesize parts of the platform. The low-power (LP), and general-purpose (GP) libraries (65nm) from ST Microelectronics and the 90nm from UMC. The UMC library is listed since a successful project incorporating the Xentium was already completed with that technology [Ahonen et al., 2011]. The ST libraries were chosen since ESA is working with ST to develop a radiation-hardened library based on the ST65nm library [Denis Lehongre, 2012]. By taking the ST technology as a reference, an extrapolation can be made using the characteristics of this technology. Early characterization of the library shows, that for a radiation-hardened library, the same speed is to be expected as for the LP variant of the commercial ST 65nm library. This is also the reason for not doing a multi-voltage threshold synthesis in which libraries are mixed, which would be more representative for a commercial ASIC [Puri, 2004; Synopsys, 2010].

6.2.1 Area and power figures

This section shows the synthesis results for different technologies of different parts of MPPB platform. Not all parts of the platform can be synthesized for the target technology. This is because some components are specific for the FPGA technology like the Aurora high speed link and some components are proprietary technology like the DDR controller.

The system is also split into parts because the synthesis software does not cope well with such a large system at once. It is common practice to synthesize parts of the system and combine the synthesized parts later on.

The ASIC synthesis runs have been performed on the same RTL as used in the MPPB FPGA prototype where possible. Synopsys Design Compiler, DFT Compiler and Power Compiler C-2009.06-SP5 [Synopsis, 2005] were used in all runs.

The following configuration was used for synthesis.

TABLE 6.1 – *Single Xentium synthesis results*

	UMC90	ST65LP	ST65GP
Total area (@ 200MHz)	1.180mm^2	0.823mm^2	0.715mm^2
Memories area	0.588mm^2	0.305mm^2	0.317mm^2
Max clock speed	330MHz	220Mhz	444Mhz
Power consumption (1k FFT @ 200Mhz)	12.03mW (2.17mW leakage)	8.70mW (0.5mW leakage)	18.64mW (10.65mW leakage)

- 0 mm2 target area *;
- 0 W target power consumption †;
- 200 MHz target clock speed;
- Enabled re-timing;
- Clock-gating insertion;
- Full scan-chain insertion (DFT);
- Single VT optimization (High VT libraries).

Xentium without Sabrewing

Table 6.1 shows the synthesis results for the Xentium with 8Kb of instruction memory and 16Kbyte of data-memory. Important to note is that the RTL of the Xentium can be configured to utilize different size memories if specific algorithms would require this. The power figures are obtained with a gate level SAIF (Switching Activity Interchange format) simulation, except for the ST65LP one, because of lack of support from the library at that time. The numbers for ST65LP are therefore less accurate. Estimating power is quite difficult without a complete configuration and running simulations. Therefore, Table 6.1 shows the simulation results for the power consumption of a Xentium running a 1k FFT at 200Mhz.

The differences between the low power (LP) and general purpose libraries (GP) show a clear trade-off between power and speed.

Table 6.2 shows the are and speed results for twelve NoC routers. Since a router itself is quite small, the gains between the different technologies are small.

Tables 6.3, 6.4 and 6.5 show the results of the different interfaces connected to the NoC. From this the Spacewire interface, the DDR controller and Aurora interface are excluded. This is because the DDR and Aurora controller that were used, are Xilinx proprietary technology and they can only be used for their FPGAs. The Spacewire interface has not been synthesized for the target technologies since it had already proven itself in radiation-hardened technology.

*Minimum size to trigger aggressive area optimization
†Minimum power to trigger aggressive power optimization

TABLE 6.2 – *NoC mesh synthesis results for 12 routers*

	UMC90	ST65LP	ST65GP
Total area (@200MHz)	0.788mm^2	0.520mm^2	0.504mm^2
Max clock speed	550MHz	460MHz	870MHz
Clock gated registers	87.61%	87.55%	87.55%

TABLE 6.3 – *Slave NoC interface*

	UMC90	ST65LP	ST65GP
Total area (@200MHz)	0.167mm^2	0.085mm^2	0.073mm^2
Max clock speed	450MHz	345MHz	625MHz

TABLE 6.4 – *Master / Slave NoC interface*

	UMC90	ST65LP	ST65GP
Total area (@200MHz)	0.147mm^2	0.086mm^2	0.078mm^2
Max clock speed	460MHz	350MHz	654MHz

Although the complete platform can not be constructed from only these parts, it does provide an insight into what can be expected in terms of speed and area.

Taking the ST low power variant as a baseline, a maximum clock frequency of 200MHz can be expected from the preliminary results. The Xentium is the limiting factor in terms of speed. The Xentium also occupies the largest area on an ASIC. The final results depend on the technology, area limit, and the desired power consumption and dissipation.

Based on the projected speed, an estimate can be made concerning the computational capacity of a Xentium. A single 200Mhz Xentium would be able to perform

- 800 16-bit MMAC[‡]/s;

[‡]Mega Multiply Accumulate

TABLE 6.5 – *ADC/DAC bridge interface*

	UMC90	ST65LP	ST65GP
Total area (@200MHz)	0.379mm^2	0.222mm^2	0.218mm^2
Max clock speed	527MHz	366MHz	630MHz

- 400 32-bit MMAC/s;
- 400 16-bit complex MMAC/s;

The actual performance depends on the algorithm and available memory but these figures provide a rough estimate. Based on these figures and the required computational capacity, the number of Xentiums on a future DSP can be estimated. The other parts of the NoC (Spacewire etc.) are less timing critical and should also be an to run at a clock frequency 200Mhz. The parts that do not scale are the interfaces to the outside world, since they are required to run at a standardized speed.

Another part that does not scale as well is the interface towards the DDR memory. The interface will be the slowest in the system since the NoC would allow to have a single link speed of 6.4Gb/s while the memory might be slower depending on what kind of DDR is available. This does not necessarily have to be a problem since a very wide memory bus for the memory could alleviate this bottleneck. Widening the memory bus is possible since most of the memory transfers are large block based-transfers and only rarely single word read and writes are used.

6.2.2 Giga-flops

One of the requirement within the project was the ability to achieve a performance of a giga-floating-point-operations per second. The number of Xentiums that would be needed for this can be determined with the help of the synthesis results from section 6.2 and the software floating-point results from section 5.3.5. This estimate will be pessimistic as the number of clock-cycles per floating-point operation is expected to go down with the optimization of the software floating-point library. However, any penalties that might occur due to the size limitation of the instruction cache (see Section 5.3.5) or any other performance penalty that might occur are not taken into account.

Taking the average number of 70 cycles required for a multiplication, running at a speed 200 MHz for a future ASIC, would mean a 2.86 million floating-point operations a second. To ultimately get to 1 Giga-flops this would imply a processor count of 350 in an ideal scenario. In current process technology this is clearly not possible due to limitations on die size. Also managing such a large number of processors in software is extremely difficult.

When more optimizations are made to the library it can be expected to reach the performance figures of the ST231 (see Table 5.8) because the Xentium offers more integer operations per second than the ST231. If 21 cycles per floating-point operation is to be achieved the number of Xentiums would drop to 105 processors, which is still too high to be practical.

To get to the Giga-flops, either the frequency of the ASIC needs to go up or the cycle count has to go down. The first option is currently not feasible

due to technology constraints in combination with radiation hardening and the critical path length of parts of the MPPB. The cycle count can go down further with the help of hardware support for floating-point.

When the Sabrewing ALU from Chapter 4 is implemented, a throughput of 1 cycle per floating-point operation can be achieved, or even 0.5 cycles per floating-point operation if fused multiply add would be counted as 2 instructions. Taking that into account, the performance requirement would be met with 5 Sabrewing ALUs or 5 Xentiums each with a single Sabrewing integrated. If FMA were to be counted as 2 instructions, only 3 Xentiums (with a Sabrewing integrated) would be needed.

By using the numbers provided in Table 4.5 of Chapter 4 this is certainly feasible in terms of die size. For this reason, the Sabrewing has been implemented in the Xentium in a test platform named Nectar. Nectar is presented in the next section.

6.3 Platform

The Nectar platform, depicted in Figure 6.1, is a very basic configuration which allows to test the Sabrewing in a minimalistic environment. It consists of a LEON2 processor and Xentium processor with Sabrewing integrated, connected to an AMBA subsystem (AHB). A complete system is built with a small amount of SRAM memory (RAM) and a UART interface. It has been synthesized for a Xilinx Virtex 4 LX 160 FPGA on an AVnet LX development board.

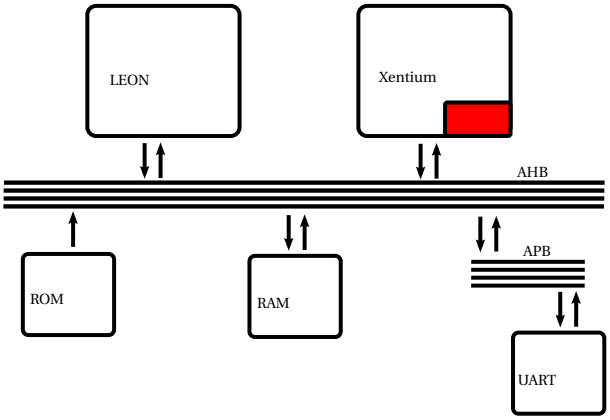


FIGURE 6.1 – *Nectar test platform, in which the marked area represents the Sabrewing.*

To cope with the difference in endianess, of the LEON2 and Xentium processors, the connection of the Xentium to the AHB takes care of re-ordering the

words.

6.4 Integration of Sabrewing and Xentium

The Sabrewing ALU was developed having a 32-bit architecture in mind in which it would be integrated. Therefore, the input operands of the Sabrewing are all 32-bit. To accommodate the 41-bit floating-point operands, the decision was made to split the operand into a 32-bit part and a 9-bit part. The 9-bit part itself was then mapped onto a 32-bit operand. The idea behind this was that a synthesis tool would remove 23 bits which are not used and it would still allow an easy integration if there is only a 32-bit interconnect available.

Recore Systems allowed access to their proprietary hardware model of the Xentium in such a way that it was possible to integrate the Sabrewing into the datapath of the Xentium. To integrate the Sabrewing with the Xentium, the A-units were replaced with a single Sabrewing ALU.Figure 6.2 and Figure 6.3 respectively show the original and modified Xentium data-paths. Ideally the choice would be to replace only a single ALU. However, this was not possible, because Sabrewing requires three input operands and the ALUs in the Xentium all use two input operands. The A unit was chosen because it has addition as its main function which is an operation the Sabrewing can perform as well. It has the ability to deal with 40-bit operands as well.

The 40-bit input operands of the A unit can be loaded from a 32-bit register bank of which the addressing is done in pairs. A pair consists of two locations in the same bank where 32-bit of one location acts as the lower 32-bit and the other location is used for the 8 most significants bits of the complete 40-bit word. To be able to produce the 41-bits which are actually needed for a single extended floating-point word for the Sabrewing, the 8 bits are extended to 9 bits.

Since two A-units are replaced, there are two 40-bit output ports available. The 40 bits of one output operand are used for the 9 bits of the exponent and the other 40 bit output is used for the 32 bits of the significand. The status bits are also output as the MSBs of the operand which contains the 9 bits exponent and sign bit. Although the status bits are not needed for basic functionality, in this way, they are still available to test the full functionality of the Sabrewing ALU.

Ideally, if the hardware description of the Xentium and its software tools were fully parameterizable, the Sabrewing could have been integrated seamlessly. The interconnect could have been changed at design time, with considerable effort, but this would defy the purpose of doing a "fast" integration of the Sabrewing ALU.

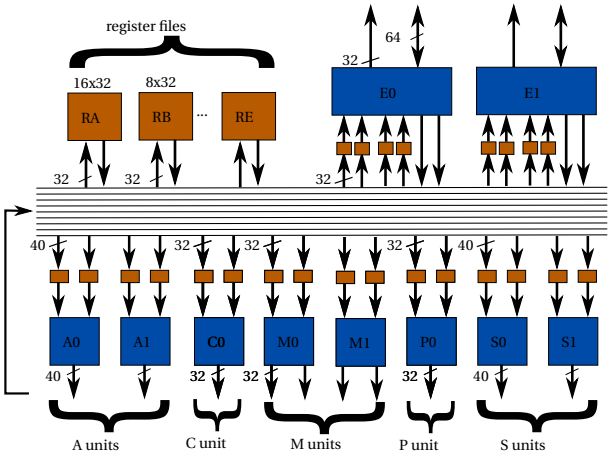


FIGURE 6.2 – *Original Xentium Datapath, Copyright Recore Systems.*

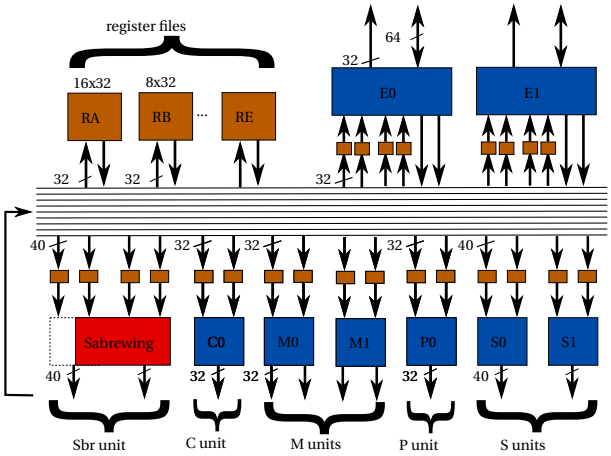


FIGURE 6.3 – *Xentium Datapath with Sabrewing ALU, Copyright Recore Systems.*

6.4.1 Synthesis

Although the Nectar platform itself is only intended as a test platform, it is still interesting to provide the synthesis results for this platform. The platform was mapped with the Xilinx tools release 12.2 for a Virtex 4 LX160 and synthesized with the 64 bit version of Precision RTL Synthesis 2010a. The results are shown in Table 6.6. It is quite difficult to provide and compare the numbers of a Sabrewing-extended Xentium with a standard Xentium in this technology even though the results are available for a Virtex 5 in Table 5.2. The reason is that

TABLE 6.6 – *Nectar prototype platform on Xilinx Virtex 4 LX 160 utilization*

Resource	**Utilization**	**Utilization %**
BSCANs	1 out of 4	25%
BUFGs	4 out of 32	12%
DCM_ADVs	1 out of 12	8%
DSP48s	9 out of 96	9%
External IOBs	27 out of 960	2%
LOCed IOBs	5 out of 27	18%
RAMB16s	85 out of 288	29%
Slices	51839 out of 67584	76%
SLICEMs	539 out of 33792	1%

TABLE 6.7 – *Nectar prototype platform on Xilinx Virtex 5 LX 330 utilization (including LEON2 and memory)*

Resource	Used	Avail	Utilization
Resource	Used	Avail	Utilization
IOS	23	960	2.40%
Global Buffers	4	32	12.50%
Function Generators	51640	207360	24.90%
CLB Slices	12910	51840	24.90%
Dffs or Latches	27934	207360	13.47%
Block RAMs	43	324	13.27%
DSP48Es	9	192	4.69%

Xilinx made a fundamental change in the underlying technology of the FPGAs between the Virtex 4 and 5. A Virtex 5 slice contains four 6-input LUTs while a Virtex 4 slice contains two 4-input LUTs. Although there is an overall increase in resources in terms of slices, not all logic benefits from the increase in inputs per LUT. For example a 32-bit XOR gate consumes seven 6-input LUTs or eleven 4-input LUTs. Therefore, it is difficult to just subtract the numbers provided in Table 5.2 from the ones provided in Table 6.6. To provide some indication the synthesis has been done for the same Virtex 5 of which the results are in Table 6.7. This has been done for comparison, but the platform has never been functionally tested.

Since the platform is intended to be used as an ASIC, the rest of this chapter focuses on ASIC synthesis. Synthesis for the Xentium is done similarly as in Section 6.2.1. There are two additions to this, because more details of the tooling became available. Synthesis was done with the ST LP libraries and with multi-VT in which SVT, HVT and LVT were mixed. Mixing the different libraries makes

TABLE 6.8 – *Xentium ALUs without Sabrewing*

ALU	**Area mm^2**	**Area Percentage**
Datapath Xentium	0.129	100.0%
A0	0.010	7.9%
A1	0.010	7.9%
C0	0.009	7.1%
E0	0.014	11.2%
E1	0.014	11.2%
S0	0.012	9.0%
S1	0.011	9.0%
P0	0.009	6.7%
M0	0.019	15.1%
M1	0.019	15.1%

TABLE 6.9 – *Xentium ALUs including Sabrewing*

ALU	**Area mm^2**	**Area Percentage**
Datapath Xentium	0.147	100.0%
FMA	0.040	27.0%
FMA/Sabrewing	0.030	20.1%
C0	0.009	6.0%
E0	0.014	9.7%
E1	0.014	9.8%
S0	0.011	7.7%
S1	0.011	7.7%
P0	0.008	5.6%
M0	0.019	13.1%
M1	0.019	13.1%

it possible to select the most power-efficient logic outside the critical path and higher speed but less power efficient logic in the critical paths. The other addition is that results are obtained after place-and-route which was done with Cadence Encounter Digital XL 11.0. Both these additions were made to be able to provide more accurate power and area figures.

Chapter 4 provided the numbers for the size of the Sabrewing ALU, but now the actual costs in terms of silicon area are compared to the integer logic of the original Xentium. By synthesizing the Xentium used for the Nectar platform which incorporates the Sabrewing ALU an answer can be provided. The baseline is the Xentium excluding the Sabrewing ALU including the A0 and A1 units. This means that the register banks and the memory are not taken into account.

Table 6.8 and Table 6.9 show the synthesis results of both versions of the

TABLE 6.10 – *Xentium Non FMA ALUs*

Static Power Analysis (1.3V, 200Mhz, 20%) NO FMA	
Internal Power	16.4mW
Switching Power	7.9mW
Leakage Power	0.1mW
Total Power	24.4mW

Xentium. In Table 6.9 a subdivision has been made between the FMA unit itself and the Sabrewing logic behind it. The difference in size between them, 0.01mm^2, is the glue logic needed to incorporate the Sabrewing ALU with the Xentium itself. The differences in Table 6.9 and Table 6.8 between the same units of the same Xentium are the result of the sightly different place they have within the datapath of the Xentium.

The synthesis results in both tables show that the overall size increases which is to be expected. The A units do not contain any multiply logic and certainly not the three pipeline stages. The increase in size, however, is not that large. Both A units jointly have a size of approximately 0.020mm^2 and the complete FMA unit has a size of 0.040mm^2, which is an increase of $\approx$ 96%. The total increase in ALU area is from 0.128mm^2 to 0.147 mm^2 which is an increase of $\approx$ 15%. So with this 15% increase in area, a three stage pipelined floating-point fused multiply add function is gained at the expense of 16-bit vector integer operations of the A-units.

The difference in power consumption is a bit more difficult to determine. The synthesis tools provide the extra cost in power directly related to the gate count of the design. The tooling will take an average power consumption based on a certain average switching activity, often of 20%. The results of such static power analysis is shown in Table 6.10 and Table 6.11. Static power analysis is based on an average switching activity which the tool uses to determine an approximation of the power consumption.

Such estimations provide a decent average and can be used to make a judgment on the overall power consumption. Clearly, the datapath which includes the Sabrewing consumes more power, and in a direct comparison this would be an increase of $\approx$ 24%. However, the functionality between both sets of ALUs differs so much that it is unfair to compare both designs directly only on the average power consumption.

For this reason, a power consumption comparison has been made based on specific computations. On the original Xentium a software floating-point operation has been executed which is comparable to the functionality of the Sabrewing ALU. An activity trace has been made in QuestaSim, which is then used in the power estimation tools of Cadence Encounter.

The results from the power estimations are as close to actual ASIC power

TABLE 6.11 – *Xentium FMA ALUs*

Static Power Analysis (1.3V, 200Mhz, 20%) FMA	
Internal Power	20.8mW
Switching Power	9.4mW
Leakage Power	0.1mW
Total Power	30.3mW

FIGURE 6.4 – *Power estimation FMA datapath*

measurements as the tooling can get. Graphs like the ones in Figure 6.4 and Figure 6.5 are the result. In Figure 6.5 the waveforms represent components of the power estimation, the top one being the overall total power. The lines beneath represent the internal power, the switching power and the leakage power. Power is measured on a mW scale. The time on the horizontal axis is in a ps scale. In Figure 6.4, the specific values on the ALU are also shown. The square waves are an average over a whole clock cycle while the top graph in Figure 6.4 provides an estimate on a near continuous basis. For longer sequences, like the number of clock cycles for the software floating-point of which a part is shown in Figure 6.5, this becomes unreadable on such a scale.

The integral of the graph provides the energy consumption over time in Joules which is a better comparison between both datapaths, for the same operation. For a single precision ADD, this results in a value of 877 over a period of 2925ns for the software floating-point operation, which results in 2567mJ energy consumption. For the same single precision ADD with the Sabrewing ALU, this results in a value of 5.8 over a period of 20ns which is 117.7pJ, which is several orders of magnitude improvement over software floating-point. The reason for the very large increase in energy consumption is because of cache misses and subsequent stall cycles in which the Xentium waits for the new instructions

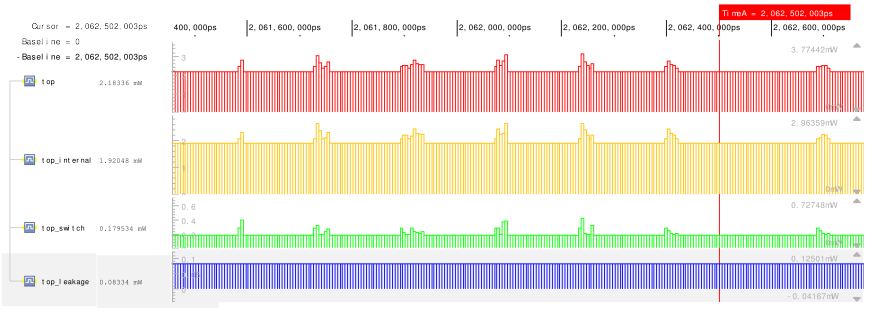


FIGURE 6.5 – *Power estimation non FMA datapath*

to be fetched. Fetching of new instructions costs energy as well. Next to the power benefit, the design with Sabrewing has the advantage that the other parts of the datapath remain available for other operations since they can still run in parallel with the Sabrewing ALU. Also the operation is computed in a fraction of the time needed for the same operation performed as software floating-point.

The previous chapter explained the goal of ESA to move towards 1 giga floating-point operations per second, which for software floating-point is not achievable within the given requirements concerning die-size and speed. With the integration of Sabrewing in the Xentium and the possibility to run it at 200Mhz, this goal has become a possibility. A single Sabrewing ALU provides two floating-point operations, add and multiply, with a throughput of 1 operation each cycle. This means that a single Sabrewing Xentium in this configuration can provide 400 MFlops. Three Xentiums in this configuration would then be able to provide 1.2 GFlops of raw processing power. In addition to that, integer operations can be performed in parallel.

6.4.2 Embedding in the Xentium software

The opcodes for the A units were re-used for the Sabrewing ALU. This made it possible to use the Sabrewing ALU without making any changes to the assembler. The downside to this approach is that the actual assembly becomes difficult to read and write due to the new meaning assigned to the mnemonics. Another downside is that the C compiler itself could not be used. Still this '*hack*' allows fast testing and evaluation of the Sabrewing in a VLIW architecture.

By making use of the the assembler's feature of "macros" and "defines", it was still possible to make a more readable assembly possible. Listing 6.1 shows a single macro. The macro makes it possible to call a single function, *epfman*, with 4 arguments to perform an "extended precision fused multiply add" with round-to-nearest rounding. The programmer, however, would still need to take

into account the pipeline delays as well as the possible hazards when switching between integer and floating-point operations.

LISTING 6.1 – *Macro example Sabrewing Xentium*

```
;extended precision fma round to nearest
%macro epfman(a, b, cl, cr)
  A0 ADD a, b
  A1 ADD cl, cr
%endmacro
```

To be able to make use of the FMA operation in the compiler requires more work. A fast solution would be to make an intrinsic available such that the operation can be explicitly called by the programmer. Much like how the FMA operation is now made available to programmers in GCC and other compilers. Another solution would be to make the compiler and scheduler aware of the possibility to map the combination of an addition and multiplication together as a single operation on the Sabrewing ALU.

6.5 Precision

6.5.1 Quantization Noise for Floating-Point

The hardware obviously has advantages in terms of performance and power consumption compared to software solutions. The question remains how well the hardware performs in terms of accuracy. The precision of floating-point itself was already discussed in Chapter 2. There are two separate factors that contribute to the improved accuracy with a Sabrewing ALU. The first is that a fused multiply-add operation lacks an intermediate rounding operation. The seconds is the possibility to employ more bits in the significand. To determine the benefit of the extra bits in the significand, a comparison has been made with a system which has even more bits in the significand and at least as many in the exponent.

The Signal to Noise Ration (SNR) for the quantization noise in floating-point itself is independent of the input sequence which is different from a uniform quantization like in fixed-point representations. If p is the number of bits in the significand including the hidden bit.

$$SNR = 5.55 \cdot 2^{2p} \tag{6.1}$$

$$SNR, dB \approx 10\log((5.55)2^{2p}) = 7.55 + 6.02p \tag{6.2}$$

With the help of Equation (6.1) and Equation (6.2) [Widrow and Koll'ar, 2008] the SNR of the floating-point formats can be determined with the original non-quantized input sequence is the reference. This results in the dB values listed in Table 6.12.

TABLE 6.12 – *SNR of Floating-point formats*

Format	SNR[dB]
Single precision p=24	152.0dB
Extended precision p=33	206.2dB
Double precision p=53	326.6dB

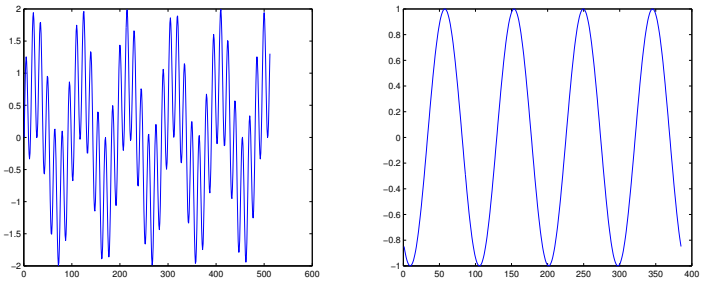


FIGURE 6.6 – *Input and output waveforms for the FIR filter used for SNR computations*

Rounding once instead of twice is better, but the actual improvement itself can be difficult to measure. This is not that surprising since it involves at most the two least significant bits. The effect becomes more clear when a lot of calculations are performed where the output is used as input again.

6.5.2 Quantization noise Experiment

Three benchmarks from Chapter 5, decimation, demodulation and compression, relied heavily on a FIR filter. A 128 taps FIR filter was run on the Nectar platform with normal single precision floating-point and with extended precision floating-point. This included the fused multiply add, so with a single rounding stage. An identical experiment was done in Matlab with single precision but without fused multiply add. As a reference the same FIR filter was run with double precision floating-point without fused multiply add. As an input the superposition of two different signals are used which are processed by a low-pass FIR filter. The input and output of the filter are shown in which can be seen in Figure 6.6.

The difference between the four outputs and the reference double precision floating-point output is considered to be quantization noise. In this way the signal to noise ratios are derived with the results shown in Table 6.13.

TABLE 6.13 – *128 FIR SNR compared to double precision*

Precision	SNR
Extended precision without FMA	186.29dB
Extended precision with FMA	186.37dB
Single precision without FMA	132.58dB
Single precision with FMA	132.64dB

6.5.3 Fused-multiply-add theory

It is also possible to calculate the gain in SNR the fused-multiply-add operation provides in certain cases. Although Chapter 2 provided a small insight into quantization, the theory behind quantization of operations and how they affect algorithms goes beyond the scope of this thesis. A large part of quantization theory in digital systems is explained in [Widrow and Koll'ar, 2008].

With the help of equations listed and explained in [Widrow and Koll'ar, 2008], the SNR can be calculated for the FIR example in the previous example. The quantization in the previous example has three sources. The quantization noise from the quantization of the input, the quantization noise from the from the 128 weighted signals, and the quantization noise from the final output quantizer.

The quantization noise power due at the input is calculated with Equation (6.3) in which p is the number of bits (including hidden bit) of the significand. The *sum of squares* is the summed square of the impulse response of the filter.

$$(0.180 \cdot 2^{-2p}) \frac{1}{2} (2^{31})^2 \cdot (\text{sum of squares}) \tag{6.3}$$

The quantization noise power due to the 128 quantizers is the same as Equation (6.3). The quantization noise power due to the output quantizer is calculated with Equation (6.4).

The output quantization is calculated as in Equation (6.4)

$$\left(0.180 \cdot 2^{-2p}\right) \frac{1}{2} \left(2^{31}\right)^2 \tag{6.4}$$

The total output SNR can then be calculated with Equation (6.5).

$$\left(\text{output SNR}\right) = 10 \log_{10} \left(\frac{\frac{1}{2} \left(2^{31}\right)^2}{\left(\text{total output noise power}\right)} \right) \tag{6.5}$$

For the filter used in Section 6.5 the *sum of squares* of the impulse response is 0.0629.

TABLE 6.14 – *Theory 128 FIR SNR*

Precision	SNR
Extended precision without FMA	205.61dB
Extended precision with FMA	205.86dB
Single precision without FMA	151.42dB
Single precision with FMA	151.67dB

So for this example the total noise power for single precision floating-point would be:

$$(0.180 * 2^{-48}) * \frac{1}{2} * (2^{31})^2 * 0.0629 +$$
$$(0.180 * 2^{-48}) * \frac{1}{2} * (2^{31})^2 +$$
$$(0.180 * 2^{-48}) * \frac{1}{2} * (2^{31})^2 * 0.0629 = 1660$$

The SNR then becomes:

$$10 * log_{10}\left(\frac{\left(\frac{1}{2} * 2^{62}\right)}{1660}\right) = 151.4\text{dB}$$

This holds in the case that there is rounding after addition as well as multiplication. In the case of FMA, it is a bit different. The output quantization can be omitted. For this reason the FMA dB calculation becomes:

For this same example it is:

$$(0.180 * 2^{-48}) * \frac{1}{2} * (2^{31})^2 * 0.0629 +$$
$$(0.180 * 2^{-48}) * \frac{1}{2} * (2^{31})^2 = 1567$$

$$10 * log_{10}\left(\frac{\left(\frac{1}{2} * 2^{62}\right)}{1567}\right) = 151.6\text{dB}$$

For single precision and the extended precision the results can be found in Table 6.14

The difference for FMA and non-FMA is a bit higher in these results. The main reason is the fact that an ideal input signal is assumed and not an already quantized signal.

Table 6.14 is compared to non-quantized values as apposed to Table 6.13 which used double precision as a reference. For other algorithms, the SNR calculations becomes more complicated and sometimes simulation is required if the exact SNR needs to be known.

6.6 Future work

6.6.1 Vector operation and word length

Based on the experiences obtained from integrating the Sabrewing ALU, some improvements can be made. In the current design, the 16-bit integer vector operations (SIMD) are sacrificed. To some extent, these could be integrated into the Sabrewing as most of the hardware is already. Adding them would make it more attractive to use the Sabrewing.

Currently the Sabrewing ALU is implemented as a replacement of two A units while it provides the functionality of one M unit and one A unit. When the Sabrewing ALU supports the 16-bit vector operations, it would be better to replace those units instead of replacing two A units. Another solution would be to add the Sabrewing ALU to the datapath next to the existing units. It would require some research to determine whether the design will still be somewhat balanced in terms of available register banks and I/O.

If the extended precision format is of no use at all, it would be a possibility to add a design-time generic to the Sabrewing ALU which allows it to be single precision only. This change would allow integration with an existing datapath even better since there would be no word-length mismatch.

6.6.2 SoCs for Payload Signal Processing

A future SoC will most likely still not need floating-point precision in every core. Suggested SoCs will put the higher throughput cores near the edges such that transport of data on the NoC is reduced to a minimum. Single cores can be clock-gated completely or even power-gated when they are not in use.

Figure 6.7 illustrates this. The darker areas are the number-crunching cores which operate in fixed-point. The lighter colored tiles are still VLIW number crunchers but have the Sabrewing ALUs to do the floating-point calculations. Then the lightest core is suggested to be a more general purpose CPU for more control dominated tasks.

A future SoC will require on-chip memory as well as I/O to off-chip memory. This could lead a layout shown in Figure 6.8. Here, the middle row of the SoC is replaced with memories. In such a way, it is possible to put buffer memory between the number crunching cores and higher throughput interfaces on one side and the more control code oriented general purpose processor on the other side.

The final configuration would again depend on the actual application and the need for floating-point hardware.

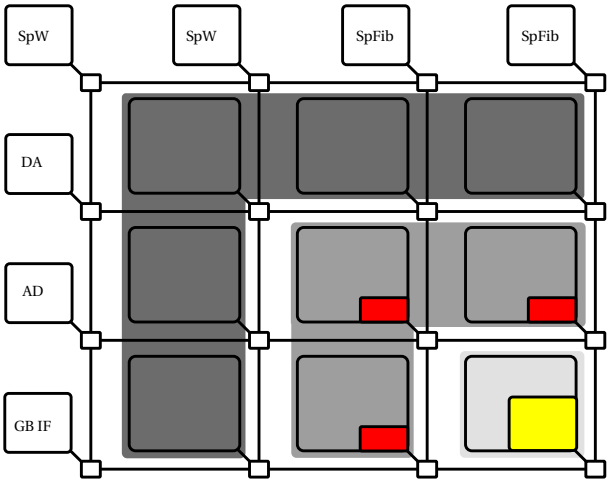


FIGURE 6.7 – *Example NoC including IO and sabrewing*

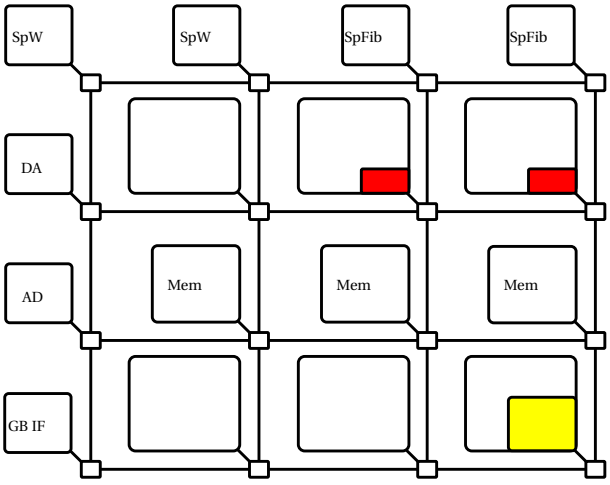


FIGURE 6.8 – *Example NoC including IO, memory, and different cores*

6.7 Conclusion

The current MPPB is very well suited for any instrument data processing when a space qualified ASIC would be developed based on this hardware. The synthesis results show that it is possible to develop an ASIC based on the 65nm technology of ST. A future radiation-hardened version of this technology makes it possible to develop a completely radiation-hardened SoC. When the characteristics of this technology are known, an estimate of performance metrics can easily be

made based on the synthesis results provided in this chapter.

From the original requirements of the MPPB the only one that is difficult to fulfill is the 1 GFLOPS one. To fulfill this requirement, a hardware floating-point unit like the Sabrewing is needed.

This chapter has shown that the Sabrewing can be integrated in a VLIW type processor. Whether or not it is required in the SoC itself, is still up to the designer. This chapter shows that the impact with respect to using the Sabrewing ALU is minimal while the gains can be significant, especially when power and speed are on a tight budget, since software floating-point operations not only take longer, but they also consume a lot more energy.

The two extra features which Sabrewing offers are extended precision. The benefit of these features are very specific in nature. Extended precision reduces quantization noise by some 50dB. Fused multiply add is a feature which is always an improvement but the amount of improvement it offers depends a lot on the algorithm. The more multiply additions on the subsequent outcomes of the calculations the more the overall result improves compared to a non-FMA architecture.

An architecture with only fixed-point hardware will often be more energy efficient. Running software floating-point algorithms on such architecture requires more energy than dedicated floating-point. Cache-misses and subsequent fetch cycles together with waits on instruction memory can cause the energy consumption to increase by several orders of magnitude. Therefore the addition of hardware support for floating-point operations can be more easily made and defended.

A future SoC for payload processing will most likely revolve around a high though-put interconnect like a NoC. Processing cores, of which some are equipped with hardware floating-point support are connected to the NoC. By having support for hardware floating-point as well as more efficient SIMD number-crunching architectures are wide variety of algorithms are supported and a more future proof SoC is envisioned.

Chapter 7

Conclusion

System-on-chip development has always been a time consuming and complex process, but this is even more true for deep-space missions. The development of ASICs for deep-space missions adds several degrees of complexity to the design process. The harsh environment in terms of temperature and radiation, high costs and long duration of the missions are main contributers to this. Due to the high complexity and demands, the design of a new processing platform for future missions is therefore not a trivial effort. New technologies, which have not proven themselves in a space environment, but still would need to be incorporated to meet certain performance requirements, add to this complexity. However, the design effort is paid back when a SoC has been designed that can be reused in multiple missions. An important step towards a new processing platform for on-board processing has been made with this thesis.

7.1 Contribution

Science and engineering are working together on new missions. From a deep-space science perspective there is a need for high accuracy and high precision, to acquire as much information from sensors as possible. From an engineering perspective there is a need for low hardware-resource usage. Chapter 2 explained the operation of the most common number systems in digital hardware. The main concern of these number systems is that application designers and hardware designers need to be aware of the fact that no digital number system fully represents all numbers. Even though standards like IEEE-754 help to provide a solid basis for representing numbers across different architectures, it is not the ultimate representation.

New missions demand more performance from hardware architectures to improve the quality of measurements and facilitate new discoveries. The limited bandwidth down-link to Earth makes transmitting all the information to Earth difficult. Therefore, the processing performance needs to increase to be

able to support the instruments projected on new missions. Although politics (e.g. ITAR regulations) should not be part of architecture development or science, it unfortunately is the case which makes development even more challenging. Chapter 3 provides requirements derived from the current and near future missions. It gives insight into where future missions might be heading and why certain requirements are needed.

With the requirements from Chapter 3 and the information from Chapter 2, a new arithmetic floating-point unit called Sabrewing, has been developed. The fact that floating-point is a requirement, combined with low power and low mass make architectures like the Sabrewing introduced in Chapter 4 an interesting design option. The Sabrewing ALU combines the multiply and add instruction in a single fused-multiply-add instruction which saves clock cycles with respect to the cycles needed by two separate instructions. An added benefit is the increased accuracy since there is only a single rounding stage. To further improve the silicon area utilization of the architecture, it combines fixed-point and floating-point operations which further improves the utilization of silicon area. It provides IEEE-754 single precision and four out of the five rounding modes. It supports an extended precision floating-point format such that precision can be increased when an algorithm demands it. The low number of pipeline stages, only three stages, makes it easier for compilers to work with it. The architecture requires 0.04 mm^2 for 200Mhz (65nm ST LP) and consumes less than 13 mW for extended floating-point operations. Sabrewing itself is available under BSD license. Because it is made in Europe, it can be used in future ESA space missions without political issues.

Chapter 5 provided the design of a new on-board signal processing platform called MPPB. It showed how new technologies can be combined with older ones in a completely implemented FPGA prototype. The prototype provides a testbed and benchmark platform. Within the constraints in terms of budget and time of the MPPB project, it proved that a heterogeneous architecture can be of benefit for the benchmarks presented. The different processing cores all have their own strong and weak points in the system and complement each other in the tasks outlined by the different benchmarks. Because the whole system is implemented and not just a theoretical study, it provided much insight into what the actual performance figures of such a system are. Furthermore parts of the MPPB are developed in DARE180 technology such that tests can be performed on a radiation hardened ASIC.

An FPGA implementation can provide an evaluation of the functional behavior, but it gives limited information about a future ASIC implementation. Therefore synthesis results of different parts of the MPPB platform are presented in Chapter 6. The synthesis results estimate a low power consumption of 8.7 mW for an FFT at a speed of 200Mhz for a single Xentium with integrated Sabrewing in 65nm STLP technology. It is expected that a similar performance

can be achieved in the projected radiation hardened technology. This is a bit of a contrast to the 13 mW of Sabrewing, but this is due to the fact that parts of the Xentium are not used for an FFT while a Sabrewing offers a lot of combined functionality.

Based on the information from the implemented benchmarks presented in Chapter 5 and the requirements gathered in Chapter 3, it is clear that hardware floating-point is a necessity when 1 GFLOP is needed. Software-based floating-point can sometimes be a solution, but in the case of the image compression benchmark, it becomes clear that the performance of software floating-point is not sufficient. Because Chapter 4 shows great promise, Sabrewing has been integrated in a Xentium (VLIW type) processor to provide detailed information about its performance.

The results presented in Chapter 6 show that the implementation of Sabrewing in a VLIW is a great success. The relatively low complexity of the interfaces and the low number of pipeline stages make it easy to implement. Integration within the Xentium did not make Sabrewing part of the critical path of the design and it is not likely that it would be critical for other implementation cases. Compiling from a high-level language was not yet possible but with macro support in the assembler, the ALU has been tested extensively. IBM research provides an extensive test for floating-point hardware and especially for fused multiply-add floating-point operation. The Sabrewing passes all test vectors for the supported operations.

7.2 Answers to research questions

In Chapter 1, four research questions were posed. Answers to these questions are provided in this section.

How should a future DSP platform for onboard processing look like, in view of current trends of multi-core architectures, massive parallelism, new interconnects and high-speed interfaces?

With the lessons learned from the MPPB project and the integration of Sabrewing in the Xentium, a future platform would preferably look like a combination of tiles described in Chapter 6. There is a caveat to this though, and this has to do to with legacy hardware and costs of development. While a future platform benefits most from a NoC-like interconnect and the simple way of attaching interfaces to the NoC, most of these interfaces need to be developed. This development cost and more importantly the testing that is needed to fully qualify such a plaform may be a step too far at this time. Therefore a platform more in the style of the MPPB where legacy hardware works together with more recent hardware is more likely to be a viable approach and might work just as well for near future missions.

How can 1 GFLOP performance be achieved, given the constraints of deep-space missions?

Chapter 2 showed that without defining exactly what is meant with GFLOP performance, a lot is open to interpretation. Even when GFLOP is defined in terms of IEEE-754 single precision operations, it still leaves a lot open for the actual implementation. One can think of rounding, de-normalized numbers. Another aspect are the operations themselves, support for hardware-division and square-root for example. Nevertheless, taking the politics into account by using European technology and having a reasonable assumption about what the operations should be, it is surely possible. A single Sabrewing and Xentium can perform 400MFLOPS. A future platform would need three Sabrewings or a similar architecture to go beyond the 1 GFLOPS which is certainly possible given the current constraints.

How can hardware floating-point be incorporated in a VLIW architecture?

For a large part that question is answered in Chapter 6. By having the interfaces at the same word size as the architecture and not impacting the critical path it is reasonably easy to implement. The extended format made it a little bit more difficult but overall the implementation is still fairly easy.

It mostly comes down to resources. Hardware floating-point simply costs chip area. On the other hand, the development time for an application programmer to switch from a floating-point algorithm to a fixed point one is high. This will most likely take more time when the algorithms become more complex. Ever shrinking feature sizes on chip enable the integration of more functions. So more often the choice can be made to incorporate hardware floating-point if the choice for a new ASIC has already been made, both for measured performance and convenience of the application developer in an already time-constrained world. Chapter 4 also makes clear that there is no need to sacrifice chip area which is used for fixed-point arithmetic for floating-point arithmetic since both can be implemented in the same area for high re-use of silicon space.

Can all requirements be combined with scalability?

From a hardware perspective the system is very scalable. The NoC part presented in Chapter 5 can be extended easily and so can the number of tiles. These tiles can be memories or processing elements. The main limitation is the silicon area of the target technology.

From a software point of view scaling is more difficult. The platform as presented in Chapter 5 needs to change if it requires more than 4 Xentiums. The way interrupts are dealt with in the current setup works well, in a future design this requires some engineering effort. Interrupts would need to be dealt with more locally or the system needs to be re-designed to do without them.

The MPPB has always been intended as a technology test platform and as such it provided a lot of valuable information. Better understanding of target applications would be needed to configure a system, based on this technology, such that it can really meet the requirements that engineers and scientist have for the new missions.

7.3 Recommendations

The main continuation of this thesis is focusing towards building a real ASIC and trying to get as close to an actual deep-space approved ASIC as possible. For building such a system one should keep in mind what actual algorithms or complete programs need to run on it. In this way the final NoC / SoC configuration comes as close to what is desired by the final user.

The currently implementation of Sabrewing in the Xentium requires work to make it into an actual commercial product. The evaluation of this effort might lead to a re-design of the Xentium interconnect if the extended precision combined with the three operand Sabrewing ALU leads to a throughput limitation.

The actual increase in precision FMA and extended format provide is something that still requires research to determine its exact implications in space related algorithms. Although extended precision is an improvement over single precision, its actual benefit to space related data processing is still unknown. If the benefit is marginal it might lead to omitting this feature, such that the design becomes even smaller and integration becomes even simpler.

In terms of floating-point performance, it is important to make clear to users that floating-point is not the ideal end-solution that many claim it to be. Statements like *"We need x amount of FLOPS"* are made all too often, without having an argument or fundamental requirements to backup this statement. Software floating-point can be a nice alternative in certain cases but since Sabrewing has many benefits it is difficult to make an argument not to implement such an ALU. For this reason an attempt should be made to have at least some form of hardware floating-point in the future ASIC. Not only for performance reasons but also to learn more from the integration process and complexity that might come from the algorithms for the future missions.

It is expected that research will advance further into the multi-core and scalable architectures such that we can re-use ASICs for several deep-space missions. Even though these missions rely heavily on public funding and as such might be difficult to pleads its cause in public. There is however a need for re-usable hardware architectures that can operate in these harsh environments and can be used for a wide variety of applications, such that deep-space science missions can continue to find answers.

Appendices

Appendix A

A.1 Sabrewing instructions

Table A.1 lists the instructions currently supported in the default configuration by the Sabrewing ALU. The integer and floating-point instructions are easily distinguishable from each other by the first bit. There are still some opcodes available for the integer instructions and it is envisioned that these can be used for SIMD like operations to make better use of the silicon.

The instructions with the extended precision work with the 41-bit format, but this changes when the Sabrewing is configured for a different format at design time. The opcodes stay the same but work a different number of bits.

TABLE A.1 – *Sabrewing instructions and opcodes*

Operation	Opcode	Description
Extended precision floating-point.		
EPFMAN	"00001"	41-bit fused multiply-add round to nearest even
EPFMAZ	"00010"	41-bit fused multiply-add round to zero
EPFMAP	"00011"	41-bit fused multiply-add round to positive infinity
EPFMAM	"00100"	41-bit fused multiply-add round to negative infinity
EPFLTV	"00101"	41-bit floating-point compare, smaller than
EPFGTV	"00110"	41-bit floating-point compare, greater than
EPFETV	"00111"	41-bit floating-point compare, equal
IEEE-754 single precision floating-point.		
SPFMAN	"01000"	32-bit fused multiply-add round to nearest even
SPFMAZ	"01001"	32-bit fused multiply-add round to zero
SPFMAP	"01010"	32-bit fused multiply-add round to positive infinity
SPFMAM	"01011"	32-bit fused multiply-add round to negative infinity
SPFLTV	"01100"	32-bit floating-point compare, smaller than
SPFGTV	"01101"	32-bit floating-point compare, greater than
SPFETV	"01111"	32-bit floating-point compare, equal
Integer.		
INTMAC	"10000"	multiply-accumulate integer
INTSLV	"10001"	integer arithmetic shift-left
INTSRV	"10011"	integer arithmetic shift-right
INTLTV	"10100"	integer compare, smaller than
INTGTV	"10101"	integer compare, greater than
INTETV	"10110"	integer compare, equal

Appendix B

B.1 Introduction

This appendix supplies extra information on two themes from Chapter 5. Section B.2 explains how the current toolchain works for the platform. It highlights two main topics. The first is that the toolchain is very flexible as it uses the well known GNU compiler as a model. Makefiles and command-line options are all similar. The other theme that is highlighted is the fact that two different architectures in the same platform can make compiling and running code quite complex. Some *hacks* are used to get the toolchain working together and make sure everything works properly.

The other part that is discussed are the other peripherals which are present in the platform. These peripherals are very generic and do not add anything in the discussion of Chapter 5 but are added here for the sake of completion. This makes clear that the platform has a lot of features to offer such that many different programs can be prototyped.

B.2 Compiling

Programs for the LEON2 are compiled with the sparc-gcc compiler, while the Xentium comes with an LLVM [Lattner and Adve, 2004] based C compiler. Nevertheless, a single binary can be compiled by having the sparc-linker, link in the Xentium binary and reference this in the LEON2 program. This process is depicted in Figure 5.6.

Most of the process is via makefiles but to elaborate a bit more on the process itself listing B.1 walks trough the compile process of a simple program. File `elf_test.c` is the input file for the Xentium and `leonelf.c` as shown in Listing B.2 is a simplified version for the LEON2. The command-line syntax for both compilers is quite similar, which makes usage of both simple.

The externals in Listing B.2 come from the `sparc-elf-objcopy` and subsequent link step which provides the programmer with these globals. These can be used to put the binary from the Xentium in the correct memory place

which is done with the `memcpy` operation in Listing B.2. The following write to the mailbox `p_xen->mlbx[0] =` provides the Xentium with the starting point of the code. This will trigger the Xentium to start fetching and executing this code.

LISTING B.1 – *Compile and linking sequence for the MPPB platform*

```
xentium-clang -O3 -Wall -c elf_test.o elf_test.c //compile xentium-c
xentium-asm -e -o test_asm.o test_asm.s //compile xentium-asm
xentium-clang -o elf_test elf_test.o test_asm.o //link xentium elf
xentium-objcopy --reverse-bytes=4 -O binary elf_test elf_test.bin //copy elf
      to binary
sparc-elf-objcopy --input binary --output elf32-sparc --binary-architecture
     sparc elf_test.bin elf_test.bin.o //create sparc-elf from xentium
     binary
sparc-elf-gcc -o leon_elf leonelf.o elf_test.bin.o //link all elf binaries
     into a single file
sparc-elf-objcopy -O srec --srec-len=4 leonelf.o leonelf.srec //convert to
     SREC for upload
```

LISTING B.2 – *Simplified C code for the LEON2 to start the Xentium*

```
extern int _binary_elf_test_bin_start;
extern int _binary_elf_test_bin_end;
extern int _binary_elf_test_bin_size;

int main()
{
  memcpy((int*)0x57000000,_binary_elf_test_bin_start,
       _binary_elf_test_bin_size); //copy xentium binary
  p_xen->mlbx[0] = (unsigned int)(0x57000000); //start xentium
}
```

The Sparc linker is "abused" to pack the Xentium and Leon binaries into a simple ELF file which is then uploaded to the LEON2. The LEON2 then takes care of copying the Xentium code to the correct memory location.

B.3 MPPB peripherals

B.3.1 LCD

There is a small 24x2 character display attached to the platform. This is mainly used for status messages when the system runs standalone.

B.3.2 General Purpose IO

The general purpose IO (GPIO) has several functions. Three IO pins are connected to buttons. The buttons can be configured in terms of interrupt triggering and de-bouncing. They are mainly used for simple interaction with the programs running on the platform. Five of the IO pins are connected to LEDs,

which are mainly used to convey simple status information. Eight of the IO pins can be configured as input or output whichever is desired. All the IO pins can be used in combination with 4 different interrupts. Uses for these IO pins vary from custom interfaces at low speeds to simple control of external devices.

B.3.3 UART

The UART interface implements the RS-232 standard and is used for communication with an attached computer. The communication varies from uploading complete programs to relaying simple debug information.

B.3.4 Timers

The timers which are implemented are used as counters to determine the speed of certain processes. This peripheral also implements the watchdog timer which can reset the system in cases of deadlocks.

B.3.5 Real-Time Clock

The real-time clock that is implemented on the platform, obeys the CCSDS [CCSDS, 2010] standard. It is used in the packet format for telemetry.

Bibliography

Abrardo A, Barni M, Bertoli A, Garzelli A, Magli E, Nencini F, Penna B, Vitulli R, 2008
"Low-complexity, secure and error-resilient hyper-spectral image compression"
In: *On-board payload data compression workshop 2008 ESA*,

Addison P S, 2002
The illustrated wavelet transform handbook: introductory theory and applications in science, engineering, medicine, and finance.,
Institute of Physics Publishing
ISBN 075-03-069-20

Ahonen T, ter Braak T D, Burgess S T, Geißler R, Heysters P M, Hurskainen H, Kerkhoff H G, Kokkeler A B J, Nurmi J, Rauwerda G K, Smit G J M, Zhang X, 2011
"CRISP: Cutting Edge Reconfigurable ICs for Stream Processing"
In: *Reconfigurable Computing: From FPGAs to Hardware/Software Codesign*, pp. 211–238,
Springer Verlag

Akesson B, February 2010
Predictable and Composable System-on-Chip Memory Controllers
Ph.D. thesis, Eindhoven University of Technology

AMD, October 2012
"New Bulldozer and Piledriver Instructions"
Available online: `http://developer.amd.com/wordpress/media/2012/10/New-Bulldozer-and-Piledriver-Instructions.pdf`

Analog Devices, 2007
"14-bit 4MS/s Analog-to-Digital Converter"
Available online: `http://www.analog.com/static/imported-files/data_sheets/AD6644.pdf`
Datasheet

Analog Devices, April 2011
"SHARC ADSP-21478/ADSP-21479 (Rev. 0)"
Available online: `http://www.analog.com/static/imported-files/data_sheets/ADSP-21478_21479.pdf`
Datasheet

Analog Devices, March 2012
"Blackfin Processor Programming Reference"
Available online: `http://www.analog.com/static/imported-files/processor_manuals/Blackfin_pgr_rev2.0.pdf`
Manual

ARM, 2006
"AMBA 3 AHB-Lite Protocol"
Available online: `https://silver.arm.com/download/download.tm?pv=1085658`
Datasheet

ARM, January 2013
"Assembler Reference version 5.03"
Available online: `http://developer.amd.com/wordpress/media/2012/10/New-Bulldozer-and-Piledriver-Instructions.pdf`

Astrium-EADS, December 2006
"The Company"
Available online: `http://www.astrium.eads.net/en/who-is-astrium/`

Atmel, April 2007
"Rad. Hard 32/40-bit IEEE Floating Point DSP"
Available online: `http://www.atmel.com/Images/doc4153.pdf`
Datasheet

Atmel, August 2011
"Rad-Hard 32 bit SPARC V8 Processor"
Available online: `http://www.atmel.com/Images/doc4226.pdf`
Datasheet

Bijlsma T, July 2011
Automatic parallelization of nested loop programs for non-manifest real-time stream processing applications
Ph.D. thesis, University of Twente

Bishop D W, May 2011
"VHDL-2008 Support Library"
Available online: `http://www.vhdl.org/fphdl`

Booth A D, 1951
"A Signed Multiplication Technique (Part 2)"
Q. J. Mech. Appl. Math **4**, pp. 236–240

Bruintjes T M, 2011
Design of a Fused Multiply-Add Floating-Point and Integer Datapath
Master's thesis, University of Twente, the Netherlands
Available online: `http://eprints.eemcs.utwente.nl/20466/`

Bruintjes T M, Walters K H G, Gerez S H, Molenkamp E, Smit G J M, January 2012
"Sabrewing: A lightweight architecture for combined floating-point and integer arithmetic"
ACM Transactions on Architecture and Code Optimization (TACO) **8**, pp. 1–22

Butler M, Barnes L, Sarma D D, Gelinas B, 2011
"Bulldozer: An Approach to Multithreaded Compute Performance"
IEEE Micro **31**, pp. 6–15, doi: 10.1109/MM.2011.23

CCSDS, November 2005
"Image Data Compression. Blue Book. Issue 1."
Available online: `http://public.ccsds.org/publications/archive/122x0b1c3.pdf`

CCSDS, November 2010
"Time Code Formats"
Available online: `http://public.ccsds.org/publications/archive/301x0b4.pdf`

CCSDS, May 2012
"Lossless Data Compression. Blue Book. Issue 2"
Available online: `http://public.ccsds.org/publications/archive/121x0b2.pdf`

Comer A, T D, 1996
"Zener Zap Anti-Fuse Trim in VLSI Circuits"
VLSI Design **5**, pp. 89–100, doi: 10.1155/1996/23706

Cooley J W, Tukey J W, 1965
"An Algorithm for the machine calculation of the complex Fourier series"
Math. Comput. **19**, pp. 297–301

Cornea M, Harrison J, Tang P T P, 2003
"Intel Itanium Floating-Point Architecture"
In: *Proceedings of the 2003 workshop on Computer architecture education: Held in conjunction with the 30th International Symposium on Computer Architecture*, WCAE '03, ACM, New York, NY, USA, doi: http://doi.acm.org/10.1145/1275521.1275526
URL `http://doi.acm.org/10.1145/1275521.1275526`

Denis Lehongre, 2012
"Esa deep sub micron program 65nm"
`http://congrexprojects.com/docs/12c25_2510/13hili-lehongre_dsm65nm_v3.pdf`

Earl J, 1965
"Latched Carry-Save Adder"
IBM Technical Disclosure Bulletin **10**, pp. 909–910

Eclipse, December 2002
"Eclipse IDE"
Available online: `http://projects.eclipse.org/projects/eclipse`

ECSS, December 2008
"SpaceWire - Remote memory access protocol"
Available online: `http://www.ecss.nl`

ESA,
"Model payload definition document for the jupiter icy moons explorer"
Available online: `http://sci.esa.int/juice_ao`

ESA/ESTEC, May 2008*a*
"ITT Massively Parallel Processor Breadboarding"
ITT, undisclosed

ESA/ESTEC, December 2008*b*
"Next Generation Space Digital Signal Processor Software Benchmark"
Availaible upon request at ESA-ESTEC

Eyles D, February 2004
"Tales From The Lunar Module Guidance Computer"
In: *Proceedings 27th Guidance and Control Conference,*

Gaisler Research, July 2005*a*
"GRFPU High Performance Floating Point Unit"
Available online: `http://www.gaisler.com/doc/grfpu_product_sheet.pdf`

Gaisler Research, July 2005*b*
"LEON2 Processor User Manual"
Available upon request, leon2-1.0.30-xst.pdf

Golomb S, jul 1966
"Run-length encodings (Corresp.)"
Information Theory, IEEE Transactions on **12**, pp. 399 – 401, doi: 10.1109/TIT.1966.1053907

Hayes W P, Kershaw R N, Bays L E, Boddie J R, Fields E M, Freyman R L, Garen C J, Hartung J, Klinikowski J J, Miller C R, Mondal K, Moscovitz H S, Rotblum Y, Stocker W A, Tow J, Tran L V, October 1985
"A 32-bit VLSI Digital Signal Processor"
IEEE Journal of Solid-State Circuits **20**, pp. 998–1004, doi: 10.1109/JSSC.1985.1052427

Hennessy J L, Patterson D A, 2006
Computer Architecture, Fourth Edition: A Quantitative Approach, chapter A, pp. A2–A77, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
ISBN 0123704901

Huang L, Shen L, Dai K, Wang Z, 2007
"A New Architecture For Multiple-Precision Floating-Point Multiply-Add Fused Unit Design"
In: *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pp. 69–76, IEEE Computer Society, Washington, DC, USA
ISBN 0-7695-2854-6, doi: 10.1109/ARITH.2007.5
URL `http://portal.acm.org/citation.cfm?id=1270377.1270450`

IBM Haifa Research Lab, June 2011
"FPgen: A Deep-Knowledge Coverage-Driven Floating-Point Test Generator"
Available online:: `https://www.research.ibm.com/haifa/projects/verification/fpgen`

IEEE, 2008
"IEEE 754-2008, Standard for Floating-Point Arithmetic"
Available online: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4610935`, doi: http://doi.ieeecomputersociety.org/10.1109/IEEESTD.2008.4610935

IEEE, 2009
"IEEE Standard VHDL Language Reference Manual"
Available online: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4772740`, doi: 10.1109/IEEESTD.2009.4772740

IEEE Task P754, August 2008
"IEEE 754-2008, Standard for Floating-Point Arithmetic"
Available Online: `http://ieeexplore.ieee.org/servlet/opac?punumber=4610933;http://en.wikipedia.org/wiki/IEEE_754-2008`, doi: http://doi.ieeecomputersociety.org/10.1109/IEEESTD.2008.4610935

imec, August 2004
"Radiation Hardening By Design"
Available online: `http://microelectronics.esa.int/mpd2004/DARE-ESA-MPD2004.pdf`

imec, August 2012
"Radiation Hardened Mixed-Signal IP with DARE Technology"
Available online: http://microelectronics.esa.int/amicsa/2012/pdf/S3_02_Thys_slides.pdf

Intel, 2011
"Intel 64 and IA-32 Architectures Software DeveloperâĂŹs Manual"
Available online: http://download.intel.com/design/processor/manuals/253665.pdf

Intel Corporation, June 2013
"Desktop 4th Generation Intel Core Processor Family"
Available online: http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/4th-gen-core-family-desktop-vol-1-datasheet.pdf

Jeannerod C P, Revy G, March 2012
"Floating-point Library for Integer Processors"
Website
Available online: http://flip.gforge.inria.fr/

Jeannerod C P, Mouilleron C, Muller J M, Revy G, Bertin C, Jourdan-Lu J, Knochel H, Monat C, 2010
"Techniques and tools for implementing IEEE 754 floating-point arithmetic on VLIW integer processors"
In: *Proceedings of the 4th International Workshop on Parallel and Symbolic Computation*, PASCO '10, pp. 1–9, ACM, New York, NY, USA
ISBN 978-1-4503-0067-4, doi: 10.1145/1837210.1837212
URL http://doi.acm.org/10.1145/1837210.1837212

Jessani R M, Putrino M, September 1998
"Comparison of Single- and Dual-Pass Multiply-Add Fused Floating-Point Units"
IEEE Transactions on Computers **47**, pp. 927–937, doi: 10.1109/12.713312

Kalliojarvi K, Astola J, April 1996
"Roundoff errors in block-floating-point systems"
Signal Processing, IEEE Transactions on **44**, pp. 783 –790, doi: 10.1109/78.492531

Lattner C, Adve V, Mar 2004
"LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation"
In: *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California

Lavenier D, Solihin Y, Cameron K, 2000
"Integer/Floating-point Reconfigurable ALU"
In: *Proceedings of the 6th Symposium on New Machine Architectures (SympA'6)*, Besancon, France

Lyons R, 2010
Understanding Digital Signal Processing,
Pearson Education
ISBN 9780137028528

Markstein P, 2004
"Software Division and Square Root Using Goldschmidt's Algorithms"

In: *Proceedings of the 6th Conference on Real Numbers and Computers*, pp. 146–157, Schloss Dagstuhl, Wadern, Germany
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.85.9648&rep=rep1&type=pdf

Maxwell, September 2006
"SCS750"
Available online: http://www.maxwell.com/products/microelectronics/docs/scs750_rev7.pdf
Datasheet

Microsemi, January 2013
"RTAX-S/SL and RTAX-DSP Radiation-Tolerant FPGAs"
Available online: http://www.actel.com/documents/RTAXS_DS.pdf
Datasheet

Montoye R K, Hokenek E, Runyon S L, January 1990
"Design of the IBM RISC System/6000 Floating-Point Execution Unit"
IBM Journal of Research and Development **34**, pp. 59–70, doi: http://dx.doi.org/10.1147/rd.341.0059
URL http://dx.doi.org/10.1147/rd.341.0059

Mueller S M, Jacobi C, Oh H J, Tran K D, Cottier S R, Michael B W, Nishikawa H, Totsuka Y, Namatame T, Yano N, Machida T, Dhong S H, 2005
"The Vector Floating-Point Unit in a Synergistic Processor Element of a CELL Processor"
In: *17th IEEE Symposium on Computer Arithmetic, 2005. ARITH-17 2005.*, pp. 59–67, doi: 10.1109/ARITH.2005.45

Oklobdzija V G, March 1994
"An Algorithmic and Novel Design of a Leading Zero Detector Circuit: Comparison With Logic Synthesis"
IEEE Transactions on Very Large Scale Integration (VLSI) Systems **2**, pp. 124–128, doi: 10.1109/92.273153

Oppenheim A, June 1970
"Realization of digital filters using block-floating-point arithmetic"
Audio and Electroacoustics, IEEE Transactions on **18**, pp. 130 – 136, doi: 10.1109/TAU.1970.1162085

Ostler P, Caffrey M, Gibelyou D, Graham P, Morgan K, Pratt B, Quinn H, Wirthlin M, dec. 2009
"SRAM FPGA Reliability Analysis for Harsh Radiation Environments"
Nuclear Science, IEEE Transactions on **56**, pp. 3519 –3526, doi: 10.1109/TNS.2009.2033381

Palacharla S, Smith J E, 1995
"Decoupling Integer Execution in Superscalar Processors"
In: *MICRO 28: Proceedings of the 28th annual international symposium on Microarchitecture*, pp. 285–290, IEEE Computer Society Press, Los Alamitos, CA, USA
ISBN 0-8186-7349-4

Parhami B, 2000
Computer Arithmetic: Algorithms and Hardware Designs,
Oxford University Press, Oxford, UK
ISBN 0-19-512583-5

PLDA, March 2008
"XpressGEN2V5 - Reference Manual"
Available online: http://www.plda.com/documentations/78

Puri R, 2004
"Minimizing power under performance constraint"
In: *International Conference on Integrated Circuit Design and Technology ICICDT '04.*, pp. 159–163, doi: 10.1109/ICICDT.2004.1309935

Quinnell E, Swartzlander E E, Lemonds C, 2007
"Floating-Point Fused Multiply-Add Architectures"
In: *Conference Record of the Forty-First Asilomar Conference on Signals, Systems and Computers, 2007. ACSSC 2007.*, pp. 331–337, doi: 10.1109/ACSSC.2007.4487224

Recore Systems, December 2012*a*
"Xentium Product Brief"
Available online: `http://www.recoresystems.com/fileadmin/downloads/Product_briefs/2012-2.0_Xentium_Product_Brief.pdf`

Recore Systems, June 2012*b*
"Xentium User Guide"
Available under NDA at Recore Systems

Rice R, Plaunt J, december 1971
"Adaptive Variable-Length Coding for Efficient Compression of Spacecraft Television Data"
Communication Technology, IEEE Transactions on **19**, pp. 889 –897, doi: 10.1109/TCOM.1971.1090789

Robison A D, June 2005
"N-bit Unsigned Division Via N-Bit Multiply-Add"
In: *17th IEEE Symposium on Computer Arithmetic, 2005. ARITH-17 2005.*, pp. 131–139, doi: 10.1109/ARITH.2005.31

Rojas R, april 1997
"Konrad Zuse's legacy: the architecture of the Z1 and Z3"
Annals of the History of Computing, IEEE **19**, pp. 5 –16, doi: 10.1109/85.586067

Schmookler M S, Nowka K J, 2001
"Leading Zero Anticipation and Detection - A Comparison of Methods"
In: *15th IEEE Symposium on Computer Arithmetic, 2001.*, pp. 7–12, doi: 10.1109/ARITH.2001.930098

Schwarz E M, 2006
"Binary Floating-Point Unit Design"
In: *High-Performance Energy-Efficient Microprocessor Design*, Series on Integrated Circuits and Systems, pp. 189–208, Springer US

Sjalander M, Larsson-Edefors P, september 2008
"High-Speed and Low-Power Multipliers Using the Baugh-Wooley Algorithm and HPM Reduction Tree"
In: *15th IEEE International Conference on Electronics, Circuits and Systems, 2008. ICECS 2008.*, pp. 33 –36, doi: 10.1109/ICECS.2008.4674784

Solihin Y, Cameron K, Luo Y, Lavenier D, Gokhale M, 2001
"Mutable Functional Units and Their Applications on Microprocessors"
In: *Proceedings of the International Conference on Computer Design: VLSI in Computers & Processors*, pp. 234 –239, IEEE Computer Society, Washington, DC, USA, doi: 10.1109/ICCD.2001.955034
URL `http://portal.acm.org/citation.cfm?id=876877.879218`

SPARC, 1991
"The SPARC Architecture Manual"
Available online: `http://www.sparc.com/standards/V8.pdf`

STMicroelectronics, September 2009
"ST231 core and instruction set architecture"
Reference Manual
Available online: `http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/REFERENCE_MANUAL/CD17645929.pdf`

Synopsis, May 2005
"Design Compiler Ultra"
Available online: `http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DCUltra/Documents/DCUltra-ds.pdf`

Synopsys, 2010
Low-Power Flow User Guide
Available with Design Compiler

Synopsys, June 2011
DesignWare Building Block IP Overview
Available with Design Compiler

ter Braak T D, Burgess S T, Hurskainen H, Kerkhoff H G, Vermeulen B, Zhang X, September 2010
"On-Line Dependability Enhancement of Multiprocessor SoCs by Resource Management"
In: *Proceedings of the 2010 International Symposium on System-on-Chip, Tampere, Finland*, pp. 103–110, IEEE Circuits and Systems Society, Piscataway

Texas Instruments, 2003
"TMS320C62x/64x FastRTS Library ProgrammerâĂŹs Reference"
Available online: Available online: `http://www.ti.com/lit/ug/spru653/spru653.pdf`

Texas Instruments, 2006
"DAC5562 User's Guide"
Available online: `http://www.ti.com/lit/ug/slau139b/slau139b.pdf`

Texas Instruments, 2009
"C55x v3.x CPU Algebraic Instruction Set Reference Guide"
Available online: `http://www.ti.com/lit/ug/swpu068e/swpu068e.pdf`

Texas Instruments, October 2011
"TMS320C6742 Fixed/Floating Point Digital Signal Processor (Rev. C)"
Available online: `http://www.ti.com/lit/ds/symlink/tms320c6742.pdf`

Thapliyal H, Arabnia H R, Vinod A P, 2006
"Combined Integer and Floating Point Multiplication Architecture (CIFM) for FPGAs and Its Reversible Logic Implementation"
In: *49th IEEE International Midwest Symposium on Circuits and Systems, 2006. MWSCAS '06.*, volume 2, pp. 438–442, doi: 10.1109/MWSCAS.2006.382306

Vassiliadis S, Lemon D S, Putrino M, August 1989
"S/370 Sign-Magnitude Floating-Point Adder"
IEEE Journal of Solid-State Circuits **24**, pp. 1062–1070, doi: 10.1109/4.34093

Wallace C S, February 1964
"A Suggestion for a Fast Multiplier"
IEEE Transactions on Electronic Computers **EC-13**, pp. 14–17, doi: 10.1109/PGEC.1964.263830

Walters K, Kokkeler A B J, Gerez S, Smit G J M, 2009
"Low-complexity hyperspectral image compression on a multi-tiled architecture"
In: *Adaptive Hardware and Systems, 2009. AHS 2009. NASA/ESA Conference on*, pp. 330–335, doi: 10.1109/AHS.2009.28

Walters K H G, Gerez S H, Smit G J M, Rauwerda G K, Baillou S, Trautner R, July 2011
"Multicore soc for on-board payload signal processing"
In: *CNASA/ESA Conference on Adaptive Hardware and Systems, AHS 2011, San Diego*, pp. 17–21, IEEE Computer Society, USA

Widrow B, Koll'ar I, 2008
Quantization Noise,
Camebridge University Press, 1st edition
ISBN 978-0-521-886710

Wittenbrink C M, Kilgariff E, Prabhu A, 2011
"Fermi GF100 GPU Architecture"
IEEE Micro **31**, pp. 50–59, doi: 10.1109/MM.2011.24

Wolkotte P T, January 2009
Exploration within the Network-on-Chip Paradigm
Ph.D. thesis, University of Twente, Enschede

Woo-Chan P, Shi-Wha L E E, Oh-Young K, Tack-Don H A N, Shin-Dug K I M, 1996
"Floating Point Adder/Subtractor Performing IEEE Rounding and Addition/Subtraction in Parallel"
IEICE transactions on information and systems **79**, pp. 297–305

Xilinx, April 2010
"Aurora 8B/10B Protocol Specification"
Available online: `http://www.xilinx.com/support/documentation/ip_documentation/aurora_8b10b_protocol_spec_sp002.pdf`

Ypma T J, December 1995
"Historical development of the newton-raphson method"
SIAM Rev. **37**, pp. 531–551, doi: 10.1137/1037125
URL `http://dx.doi.org/10.1137/1037125`