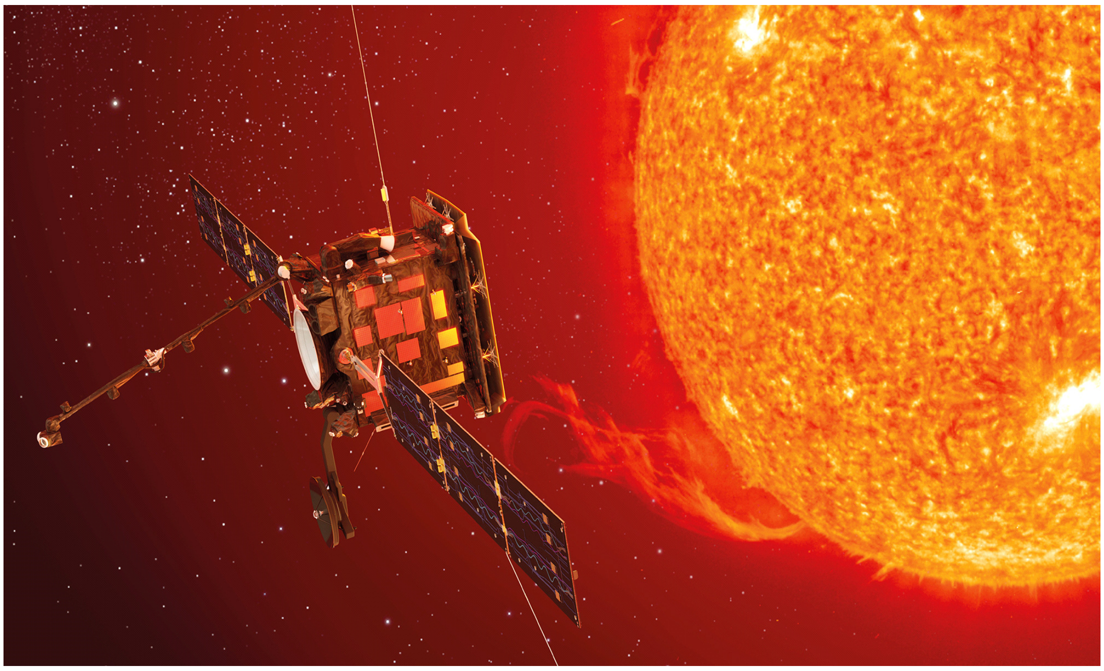
High-Performance Scientific Computing on FPGA aboard the *Solar Orbiter* PHI Instrument



Tesis Doctoral

Programa Oficial de Posgrado Ciencias de la Computación y Tecnología Informática

Juan Pedro Cobos Carrascosa

Granada, Noviembre de 2015

**INSTITUTO DE ASTROFÍSICA DE ANDALUCÍA
CONSEJO SUPERIOR DE INVESTIGACIONES CIENTÍFICAS**

**DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE COMPUTADORES
UNIVERSIDAD DE GRANADA**

Editor: Universidad de Granada. Tesis Doctorales
Autor: Juan Pedro Cobos Carrascosa
ISBN: 978-84-9125-776-9
URI: http://hdl.handle.net/10481/43508

Tesis Doctoral

Programa Oficial de Posgrado Ciencias de la Computación
y Tecnología Informática

High-Performance Scientific Computing on FPGA aboard the *Solar Orbiter* PHI Instrument

Autor:

D. Juan Pedro Cobos Carrascosa

Directores:

Dr. D. Antonio C. López Jiménez   Dr. D. Christian A. Morillas Gutiérrez

Granada, Noviembre de 2015

Instituto de Astrofísica de Andalucía
Consejo Superior de Investigaciones
Científicas

Departamento de Arquitectura y
Tecnología de Computadores
Universidad de Granada

El doctorando, Juan Pedro Cobos Carrascosa, y los directores de la tesis, Antonio C. López Jiménez y Christian A. Morillas Gutiérrez, garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, 12 de noviembre de 2015

Director/es de la Tesis	Doctorando
Fdo.: Antonio C. López Jiménez Christian A. Morillas Gutiérrez	Fdo.: Juan Pedro Cobos Carrascosa

A Pablo, Adrián y Esther

Acknowledgements

Quiero aprovechar esta oportunidad para agradecer de la forma más sincera, y por igual, a todas las personas que por su ayuda, o ánimos, han hecho posible este trabajo. Aun así, me gustaría hacer algunas menciones especiales de agradecimiento a quienes han sido fundamentales.

Al Dr. Jose Carlos del Toro, por poner sobre mis hombros toda la responsabilidad que conlleva este apasionante e increíble reto, por toda la confianza que siempre me ha dado para llevarlo a cabo y, por supuesto, por su inconmensurable ayuda. Todo ello ha sido la motivación que me ha hecho esforzarme al máximo en este trabajo y, sobre todo, disfrutar haciéndolo.

A Ramos y Bea, a Bea y Ramos, compañeros infatigables en esta aventura. Junto a los que empujar hacia delante codo con codo ha sido un placer.

Al Dr. Antonio López, por la dirección de esta tesis, por sus siempre certeros consejos y por el espléndido ambiente de trabajo que ha creado en el IAA.

Al Dr. Christian Morillas y al profesor Francisco Pelayo, por la motivación que me dieron siendo estudiante, por llevarme al mundo de la investigación en tecnología y, por supuesto, por sus revisiones y consejos.

Al IAA SO/PHI Team quienes contagian su ilusión y que con su ingente trabajo harán que un instrumento tan complejo, y de tal magnitud, como lo es SO/PHI sea un éxito. A María, Dani, Fernando, Pierre, Joaquín, Miguel, Berta y Luis.

A todos los compañeros del IAA con quienes es inspirador el trabajar, compartir despacho y convivir. A pesar de todas las barreras, trabajan desmesuradamente por hacer la mejor ciencia y tecnología posibles.

Al Dr. David Orozco por toda su ayuda y paciencia.

Para llevar a cabo esta tesis no ha sido suficiente toda esa ayuda en el plano profesional, porque sin todas las personas que me rodean nunca hubiese podido llegar hasta aquí.

A Esther, por estar siempre a mi lado y darme todo su apoyo. Por toda la alegría que me trasmite y sin quien no hubiera sido posible realizar todo este esfuerzo.

A Pablo y Adrián, cuyas risas, imaginación y energía hacen de cada día el mejor de los días.

A mi madre cuyo ejemplo de lucha y esfuerzo siempre ha sido el mejor de sus consejos.

Casi por último, y no por eso menos importante, a mis hermanos y familia, quienes siempre me han cubierto de cariño.

Y para finalizar, agradezco al Plan Nacional del Espacio y a la Junta de Andalucía por subvencionar los proyectos que han hecho posible este trabajo. Así como al Consejo Superior de Investigaciones Científicas por brindarme esta magnífica oportunidad.

Abstract

SO/PHI (*Solar Orbiter* Polarimetric and Helioseismic Imager) is a filtergraph-based, solar magnetograph aimed at mapping the vector magnetic field and the line-of-sight (LOS) velocity of the solar photospheric plasma. It belongs to the scientific payload of the European Space Agency's *Solar Orbiter* mission which will orbit the Sun at 0.28 astronomical units.

The limited telemetry rate combined with the large amount of scientific information retrieved by the SO/PHI instrument demand a sophisticated on-board data reduction and scientific analysis through the study of the polarization state of a specific spectral line. The main aim is to perform the complicated algorithm needed to translate the polarization state of the light spectrum in terms of some specific solar parameters like the magnetic field vector and velocity. Technically speaking, the inference of the solar physical quantities through a spectropolarimetric study is based on the inversion of the Radiative Transfer Equation (RTE) and these tasks require the processing of a huge quantity of data in parallel.

The RTE inverter is the core of the on-board scientific data analysis and, probably, one of the most innovative parts of the instrument. Due to the unavailability of qualified for space processors, DSPs, or GPGPUs that fulfil the stringent computational requirements with the limited room and power consumption allocated to the instrument, a specifically designed hardware device has been implemented in SO/PHI. This device is in charge of inverting the RTE aboard *Solar Orbiter* under narrow time and power constraints.

The main aim of this thesis is to design, build, and test such a hardware device for SO/PHI. With that goal in mind, we propose two different high-performance computing architectures for carrying out the RTE inversion using FPGA devices embedded in the SO/PHI instrument.

The first of these proposals is a distributed-memory MIMD multiprocessor architecture on a Virtex-5 FPGA that exploits the functional and data fine parallelism. It uses a pipelined execution based on a novel MIMD programming method. The processors within the architecture are simplified for saving resources but they are able of eliminating latency and exploiting the computing power that the FPGA provides. The synchronization and the communication network between processors have been simplified using this proposal.

The second proposal consists of a SIMD multiprocessor architecture to reach high performance in floating point operations. This architecture on a Virtex-4 FPGA squeezes the FPGA resources in order to reach the time constraints. It is focused in exploiting the data parallelism using several processors working together and using different data streams. One of the most important contributions of this architecture is the ability of saving resources allocating operation cores in a shared operation block, which is accessed by every processor. Some details for extending the architecture to other problems are pointed out. A study of how the radiation induced errors affect each block of the architecture is detailed, and two fault mitigation strategies are described.

We also present a novel software tool, which automates the entire design process and system settings from an input C-like pseudo-code. This tool uses advanced techniques of software pipelining and parallelizing scientific algorithms in multicore systems. A compiler within the tool makes it easier the use and programming of the proposed MIMD and SIMD architectures.

As a byproduct of our development, a specific, novel Singular Value Decomposition (SVD) architecture within the SIMD architecture is proposed as well. SVD is one of the steps in the RTE inversion but can be of interest to other developments as is a fairly common mathematical tool.

The achieved FPGA systems improve the time and power consumption of ground-based systems based on commercial CPUs.

The final system is tested using synthetic and real data. It satisfies the scientific precision requirements and the engineering computing time and power consumption requirements.

Resumen

SO/PHI (*Solar Orbiter* Polarimetric and Helioseismic Imager) es un magnetógrafo, basado en un filtrógrafo, destinado a crear mapas del vector campo magnético y de la velocidad del plasma solar a lo largo de la línea de visión. Pertenece a la carga útil científica de la misión *Solar Orbiter* de la Agencia Espacial Europea, la cual orbitará el Sol a una distancia de 0,28 unidades astronómicas.

La limitación en la tasa de telemetría combinada con la gran cantidad de información científica obtenida por el instrumento SO/PHI requieren que se realice a bordo una sofisticada reducción de datos y un análisis científico a través del estudio del estado de polarización de una específica línea espectral. El principal objetivo es realizar el complicado algoritmo necesario para trasladar el estado de polarización del espectro de la luz a términos de algunos parámetros solares específicos como el campo magnético y velocidad. Técnicamente hablando, la inferencia de las magnitudes físicas solares a través de un estudio espectropolarimétrico está basada en la inversión de la ecuación de transporte radiativo (RTE) y estas tareas requieren del procesamiento de una gran cantidad de datos en paralelo.

El inversor RTE es el núcleo del análisis científico a bordo y, probablemente, una de las partes más innovadoras del instrumento. Debido a la inexistencia de procesadores, DSPs o GPGPUs cualificados para el espacio que cumplan con los exigentes requisitos de cómputo, de limitación de espacio y de consumo de potencia del instrumento, ha sido necesario el desarrollado un dispositivo hardware específico para SO/PHI. Este dispositivo es el encargado de invertir la ecuación RTE a bordo de Solar Orbiter bajo estrechas limitaciones de tiempo y potencia.

El objetivo principal de esta tesis es diseñar, construir y probar tal dispositivo hardware para SO/PHI. Con ese objetivo en mente, proponemos dos arquitecturas de cálculo científico de altas prestaciones para llevar a cabo la inversión RTE usando dispositivos FPGA embebidos en el instrumento SO/PHI.

La primera de estas propuestas es un arquitectura multiprocesador tipo MIMD de memoria distribuida en FPGA que explota el paralelismo fino a nivel de datos y funcional. Usa una ejecución segmentada basada en un innovador método de programación MIMD. Los procesadores de la arquitectura son simplificados para ahorrar recursos pero son capaces de eliminar latencias y de explotar la potencia de cálculo que la FPGA provee. Usando esta propuesta se simplifica la sincronización entre procesadores y la red de comunicaciones.

La segunda propuesta consiste de una arquitectura multiprocesador tipo SIMD para alcanzar alto rendimiento en las operaciones en coma flotante. Esta arquitectura en la FPGA Virtex-4 exprime los recursos de la FPGA para alcanzar las restricciones de tiempo. Está enfocada en explotar el paralelismo de datos usando varios procesadores que trabajan juntos y usan diferentes flujos de datos. Una de las contribuciones más importantes de esta arquitectura es la habilidad de ahorrar recursos alojando núcleos de operación en un bloque de operaciones compartidas, el cual es accedido por todos los procesadores. También se dan propuestas para extender la arquitectura a otros problemas y se hace un estudio detallado de como los errores inducidos por radiación afectan a cada bloque de la arquitectura. Además se describen dos estrategias para mitigar errores.

También presentamos una innovadora herramienta software, la cual automatiza completamente el proceso de diseño y la configuración del sistema para un seudocódigo de entrada tipo C. Esta herramienta usa técnicas avanzadas de segmentación de software y paralelización de algoritmos científicos en sistemas multiprocesador. El compilador que forma parte de la herramienta facilita el uso y programación de las arquitecturas propuestas SIMD y MIMD.

Como un subproducto de este desarrollo también se ha generado una arquitectura innovadora para el cálculo de la descomposición en valores singulares (SVD) de una matriz que se aloja en la arquitectura SIMD. El cálculo del SVD es uno de los pasos dentro de la inversión RTE pero puede ser de interés para otros desarrollos ya que es justamente una herramienta matemática muy común.

El sistema en FPGA obtenido mejora el tiempo y consumo de potencia de los sistemas terrestres basados en ordenadores comerciales.

Finalmente, se presentan los resultados obtenidos usando datos sintéticos y reales. Se ha comprobado que se satisfacen los requisitos de precisión científica, de tiempo de cálculo y de consumo de potencia.

High-Performance Scientific Computing on FPGA Aboard the *Solar Orbiter* PHI Instrument

Table of Contents

1

Introduction

From the early days of humankind, the Sun has unquestionably been the main celestial body, and the one that breathes life into the Earth. Soon, the Sun was ascended to a god category. Inti, in the Inca Empire, Ra, for Egyptians, and Helios, in the Greek culture, were deifications of the Sun.

The first astronomic attempt to explain how the universe works located the Earth as its center. Several centuries after that paradigm, Nicolaus Copernicus shifted the center of the known universe to the Sun. Back then, the Sun was not a god any more. The road had been opened for a scientific thought that was able of mathematically explaining the universe. Newton, after Galileo and Kepler, laid the foundations of modern science with its formal models and methodology. Four centuries later, the same thirst for knowledge has carried us to space exploration for searching the principles that manage the universe.

This PhD dissertation is an example of how the everlasting dichotomy between science and technology is old fashioned. It is a technological step forward with the background goal of helping science to increase our basic knowledge. The main goal of this dissertation is obtaining more efficient and powerful computing models to fulfil the stringent requirements of a solar physics problem. Hence, it can be considered advancement in the computer science field (thus improving computing technology) that can be used as a tool for astrophysics. As a by-product, many of the developments described in this dissertation can be used in other applications where high-performance scientific computing is needed.

1.1 The *Solar Orbiter* mission

The Earth is within the extended atmosphere of the Sun, called Heliosphere. Advances in our understanding of both the Sun and the Heliosphere are therefore expected to improve not only our knowledge but our life on Earth as well. This is the goal of *Solar Orbiter*, an ESA mission in collaboration with NASA that was selected as the first medium (M)-class mission of ESA's Cosmic Vision 2015 – 2025 program.

An outline about the scientific objectives of *Solar Orbiter* is adopted from [Müller (2013)]:

"The results from current and past solar and heliospheric missions such as Helios [Porsche (1977); Schwenn (1990); Schwenn (1991)], Voyager [Stone (1977)], Ulysses [Wenzel (1992)], Yohkoh [Acton (1992)], SOHO [Domingo (1995)], TRACE [Handy (1999)], RHESSI [Lin (2002)], Hinode [Kosugi (2007)], STEREO [Kaiser (2008)] and SDO [Pesnell (2012)] have greatly improved our knowledge of the solar corona, the solar wind, and the three-dimensional Heliosphere. Each of these missions had a specific focus, being part of an overall strategy of coordinated solar and heliospheric research. However, none of these missions have been able to fully explore the interface region where the solar wind is born and heliospheric structures are formed with sufficient instrumentation to link solar wind structures back to their source regions at the Sun.

With a combination of in-situ and remote-sensing instruments and its inner-heliospheric mission design, Solar Orbiter will address the central question of heliophysics: How does the Sun create and control the Heliosphere? This primary, overarching scientific objective can be expanded into four interrelated top-level scientific questions that will be addressed by Solar Orbiter:

- *What drives the solar wind and where does the coronal magnetic field originate from?*
- *How do solar transients drive heliospheric variability?*
- *How do solar eruptions produce energetic particle radiation that fills the Heliosphere?*
- *How does the solar dynamo work and drive connections between the Sun and the Heliosphere?*

These questions represent fundamental challenges in solar and heliospheric physics today. By addressing them, we expect to make major breakthroughs in our understanding of how the inner solar system works and is driven by solar activity. To answer these questions, it is essential to make in-situ measurements of the solar wind plasma, fields, waves, and energetic particles close enough to the Sun that they are still relatively pristine and have not had their properties modified by subsequent transport and propagation processes. This is one of the *fundamental drivers for the Solar Orbiter mission, which will approach the Sun to as close as 0.28 Astronomical Unit (AU)"*

It is important to remark that *Solar Orbiter* will also have the possibility of observing the solar poles. An orbital inclination of 25° will be reached during the nominal mission, and a maximum inclination about the ecliptic of 34° – 36° in the extended mission.

The scientific payload of the Solar Orbiter mission is formed by ten instruments. In Figure 1 the instrument accommodation onboard Solar Orbiter is shown. In short, there are in-situ instrument for measuring the solar wind, energetic particles and magnetometers. Also there are remote-sensing instruments that basically are imagers for providing visual images, and for doing spectroscopy with different aims as study solar wind or X-ray emissions. One of them is the Polarimetric and Helioseismic Imager instrument (PHI).

Also in [Müller (2013)] are introduced the instruments that compose the scientific payload of Solar Orbiter. We adopted this introduction below.

"Relating these in-situ measurements back to their source regions and structures on the Sun requires simultaneous, high-resolution imaging and spectroscopic observations of the Sun in and out of the ecliptic plane. The resulting combination of in-situ and remote-sensing instruments on the same spacecraft, together with the new, inner-heliospheric perspective, distinguishes Solar Orbiter from all previous and current missions, enabling science which can be achieved in no other way.

The scientific payload elements of Solar Orbiter will be provided by ESA member states, NASA and ESA and have been selected and funded through a competitive selection process. These are:

The in-situ instruments:

- *The Energetic Particle Detector (EPD) experiment (J. Rodriguez-Pacheco, PI, Spain; R. F. Wimmer-Schweingruber, co-PI, Germany) will measure the properties of suprathermal ions and energetic particles in the energy range of a few keVn−1 to relativistic electrons and high-energy ions (100 MeVn−1 protons, 200 MeVn−1 heavy ions).*
- *The Magnetometer (MAG) experiment (T.S. Horbury, PI, UK) will provide detailed in-situ measurements of the heliospheric magnetic field.*
- *The Radio and Plasma Waves (RPW) experiment (M. Maksimovic, PI, France) will measure magnetic and electric fields at high time resolution and determine the characteristics of electromagnetic and electrostatic waves in the solar wind from almost DC to 20 MHz.*
- *The Solar Wind Analyser (SWA) instrument suite (C.J. Owen, PI, UK) will fully characterize the major constituents of the solar wind plasma (protons, alpha particles, electrons, heavy ions) between 0.28 and 1.2 AU.*

The remote-sensing instruments:

• The Extreme Ultraviolet Imager (EUI, P. Rochus, PI, Belgium) will provide image sequences of the solar atmospheric layers from the photosphere into the corona.

• The Multi Element Telescope for Imaging and Spectroscopy (METIS) Coronagraph (E. Antonucci, PI, Italy) will perform broad-band and polarized imaging of the visible K-corona, narrow-band imaging of the UV and EUV corona and spectroscopy of the most intense lines of the outer corona.

• The Polarimetric and Helioseismic Imager (PHI, S.K. Solanki, PI, Germany; J.C. del Toro Iniesta, co-PI, Spain) will provide high-resolution and full-disk measurements of the photospheric vector magnetic field and line-of-sight velocity as well as the continuum intensity in the visible wavelength range.

• The Solar Orbiter Heliospheric Imager (SoloHI, R.A. Howard, PI, USA) will image both the quasi-steady flow and transient disturbances in the solar wind over a wide field-of-view by observing visible sunlight scattered by solar wind electrons.

• A European-lead extreme ultraviolet imaging spectrograph Spectral Imaging of the Coronal Environment (SPICE) with contributions from ESA member states and ESA. This instrument will remotely characterize plasma properties of regions at and near the Sun.

• The Spectrometer/Telescope for Imaging X-rays (STIX) (S. Krucker, PI, Switzerland) provides imaging spectroscopy of solar thermal and non-thermal X-ray emission from ~4 – 150 keV".

Detailed descriptions of the payload elements, as well as traceability matrices of the science goals are given in [Marsden (2011)]."

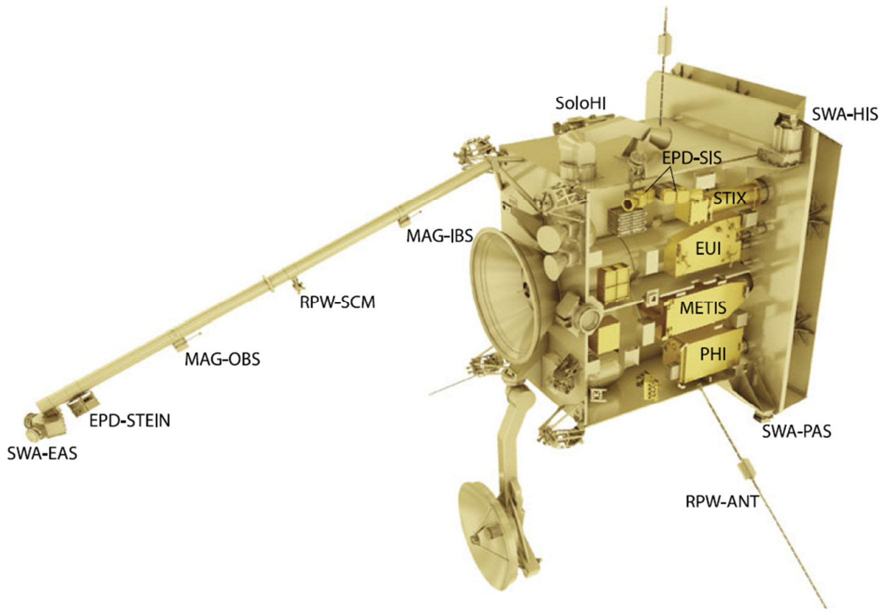


Figure 1 Payload accommodation onboard Solar Orbiter. (Adapted from [Müller (2013)])

1.2 The PHI instrument

The Polarimetric and Helioseismic Imager (SO/PHI or simply PHI) instrument will provide high-resolution and full-disk measurements of the photospheric vector magnetic field and line-of-sight (LOS) velocity as well as the continuum intensity in the visible wavelength range. The measurement principle of SO/PHI is based on imaging spectropolarimetric observations of a photospheric absorption line in the solar visible-light spectrum. More details can be found in [Solanki (2015)].

The instrument is made up of two main units, namely, an Optics Unit and an Electronics Unit (E-Unit). Figure 2 illustrates the conceptual design.

The Optics Unit contains several subsystems: two telescopes, a tunable filtergraph, two polarimetric packages, an image stabilization system, and the focal plane assembly. The, so-called High Resolution Telescope (HRT) is an off-axis Ritchey-Chrétien that will image a fraction of the solar disk at a resolution of 150 km over the solar surface at perihelion (the same resolution as the Extreme Ultraviolet Imager's high resolution channel). The refractor, Full Disk Telescope (FDT), will be able to image the full solar disk at all phases of the orbit. Each telescope has its own Polarization Modulation Package (PMP) located early in the optical path in order to minimize crosstalk effects among the four Stokes parameters. Polarimetry with a signal-to-noise ratio of 10^3 is baselined for SO/PHI. HRT or FDT will sequentially send light to a Fabry-Perot filtergraph system (100mÅ spectral resolution) and on one 2048×2048 pixels CMOS sensor. The Image Stabilization System (ISS) will compensate spacecraft jitter or other disturbances. This system is composed of a correlation tracker and a rapid tip-tilt mirror for the HRT [Hirzberger (2012)]]. Although the Heat Rejecting Entrance Window (HREW) is not visible in Figure 1, it contains the two openings for both telescopes that Figure 2 shows.

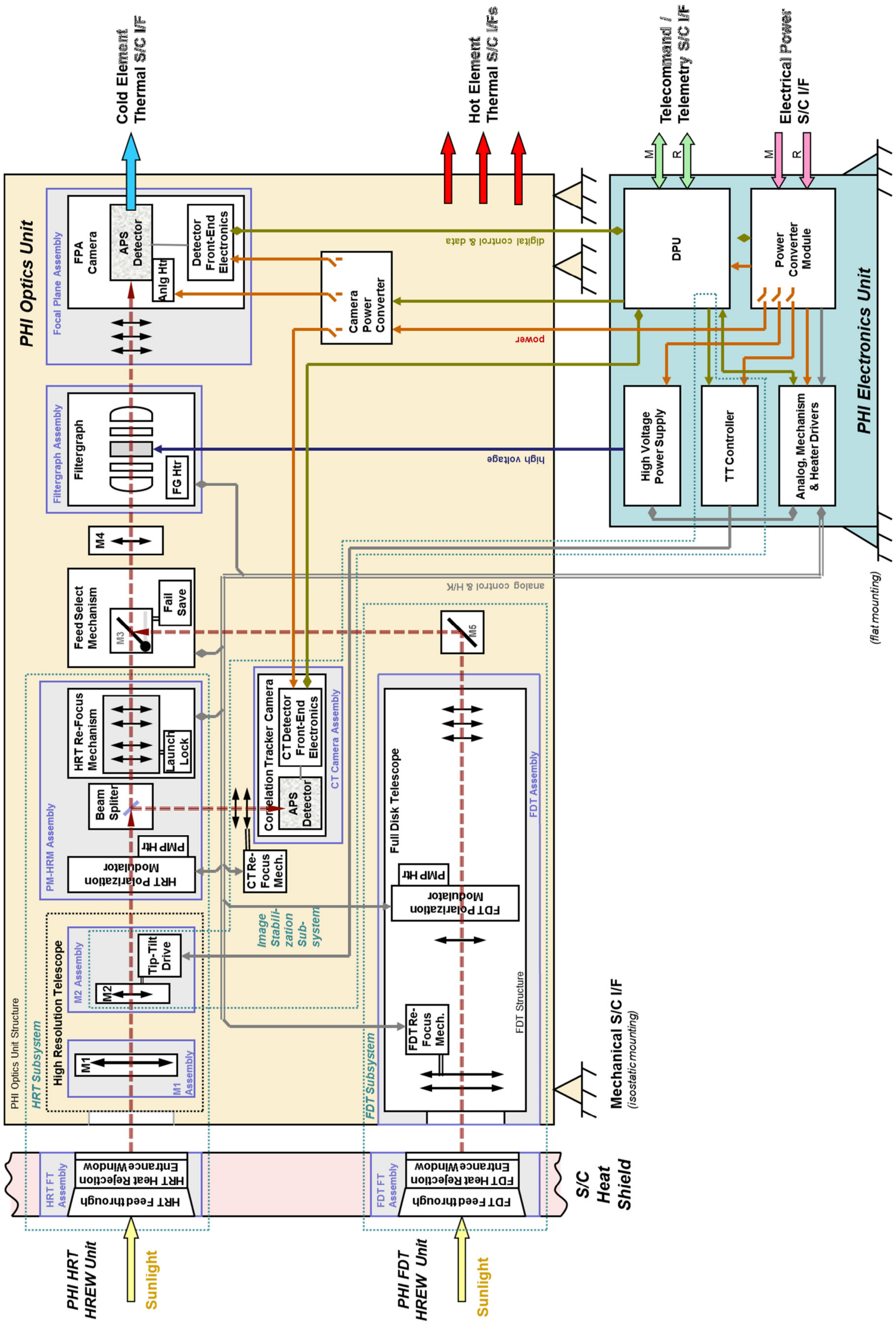


Figure 2 PHI Functional Diagram. (Adapted from [Meller (2013)])

The E-Unit is the main electronics system and is in charge of controlling the whole instrument. It is based on a modular system with individual boards (subsystems) interconnected with an external motherboard called EDS (Electrical Distribution System). It consists of five subsystems: the DPU (Digital Processing Unit for image accumulation, preprocessing, Stokes inversion of physical magnitudes, data compression, control of the instrument and of the interfaces with the spacecraft; the PCM (main and redundant Power Converter Module for providing the corresponding power supply to the subsystems); the AMHD (for housekeeping parameter acquisition, motors controller, and heater drivers) board; the TTC (Tip-Tilt Controller); and the HVPS (High Voltage Power supply for the filtergraph).

SO/PHI will observe intensively during the Science Operations Windows of each orbit. During other phases of the orbit it will continue to provide, as necessary, synoptic and context observations of the magnetic field at a considerably lower cadence. It will also provide helioseismic data at the full cadence but at a vastly reduced spatial resolution [Hirzberger (2012)]]. Specific operational modes have been defined to meet the scientific objectives during all mission phases within the allocated telemetry resources.

Severe threats have to be dealt in the SO/PHI instrument as a good power management or heat dissipation, all of them, with a mass limitation of 31 Kg for the entire instrument. The total power consumption limitation for the instrument is around 45 W during the science operation in the nominal orbit (0.28 AU). There is a great radial temperature in the nominal orbit that supposes a great challenge for the thermal engineers. This issue could affect the optical performance and the entire instrument.

A big technical challenge is the data transmission of the scientific data to ground since the available telemetry for SO/PHI is in average 20kbps, or 211 MB/day. To reduce the telemetry needed by SO/PHI, an initial reduction and the scientific analysis of the data will be carried out already onboard the spacecraft using a Milne-Eddington inversion (or RTE inversion) of the recorded data prior to their transmission to ground [Hirzberger (2012)]]. This analysis is highly computing demanding: in fact, when this type of observations is analyzed post facto on Earth, typical PCs spend almost two and a half hours [Borrero (2010)].

That is precisely one of the features that stand out among all the instrumentation in the *Solar Orbiter* mission, and maybe among all the space instruments of all times. Such an exhaustive scientific analysis has never been carried out aboard any space instrument.

Specifically, the challenge of carrying out this scientific analysis on flight is the origin and motivation of this thesis.

1.3 SO/PHI Measurement principles

The primary goal of SO/PHI is to map the vector magnetic field of the Sun (magnetography) and the velocity of the solar plasma along the line of sight (tachography). Therefore, both polarimetric and spectroscopic measurements are needed. Magnetography is made through using liquid crystal variable retarders (LCVRs) as polarization modulators and a linear polarizer as the analyzer. With them, all four Stokes parameters of light, namely, *I*, *Q*, *U*, and *V* are measured. Stokes *I* gives the intensity of the light beam; Stokes *Q* and *U* give the linear polarization; and Stokes *V* speaks about the circular polarization. Spectroscopy is made by scanning a solar photospheric absorption line at a few wavelengths with a tunable, narrow band Filtergraph (FG), which is made up of a broader band pre-filter and a narrow band $LiNbO_3$, solid Fabry-Perot etalon. With SO/PHI, thus, the spectral line profile at each point over the solar surface and in all four Stokes parameters is obtained: the so-called Stokes profiles of the spectral line are sampled at several wavelengths. The sought-for solar information is encoded in these Stokes profile samples.

The Fe I line at 617.3 nm was chosen as the SO/PHI science target after a careful comparison with a number of other widely used Zeeman-sensitive lines due to its ideal combination of properties allowing both vector magnetic field and helioseismic observations [Hirzberger (2012)]. The polarization properties of this spectral line depend on the magnetic field structure in the line-forming regions of the solar atmosphere through the Zeeman effect. In addition, the exact wavelength position of the spectral line depends on the relative velocity between the observing solar point and the spacecraft through the Doppler effect.

Figure 3 illustrates the measurement principle. The intensity solar spectrum around 617 nm is shown in panel *a* (red solid line) along with the pre-filter (orange solid line) and the etalon transmission profiles (blue solid line). The vertical dotted lines mark the central wavelength position of the various transmission peaks of the etalon. Their full width at half maximum (FWHM) is denoted by $\Delta\lambda_{FWHM}$, their separation is the so-called free spectral range (FSR), and the FWHM of the pre-filter is $\Delta\lambda_{OSPF}$. The FG spectral transmission is given by the product of the pre-filter and etalon transmission profiles, so that the side peaks of the FG are almost suppressed to zero. Panels *b* through *e* display the four Stokes profiles of the Fe I line as resulting from a simulation. The original profiles are plotted in red, the same profiles after convolution with the FG transmission profile are plotted in light blue; the asterisks mark the instrument wavelength samples.

As commented above, the information about the solar magnetic field and plasma velocity is encoded in the spectrum of these Stokes profiles. The SO/PHI data products, then, will be extracted from the primary observables by onboard processing. Without going into details, we can say that the process of getting the solar atmospheric parameters from the Stokes parameters is known as the inversion of the Radiative Transfer Equation (RTE) [del Toro (2003)]. This process will be detailed in Chapter 2.

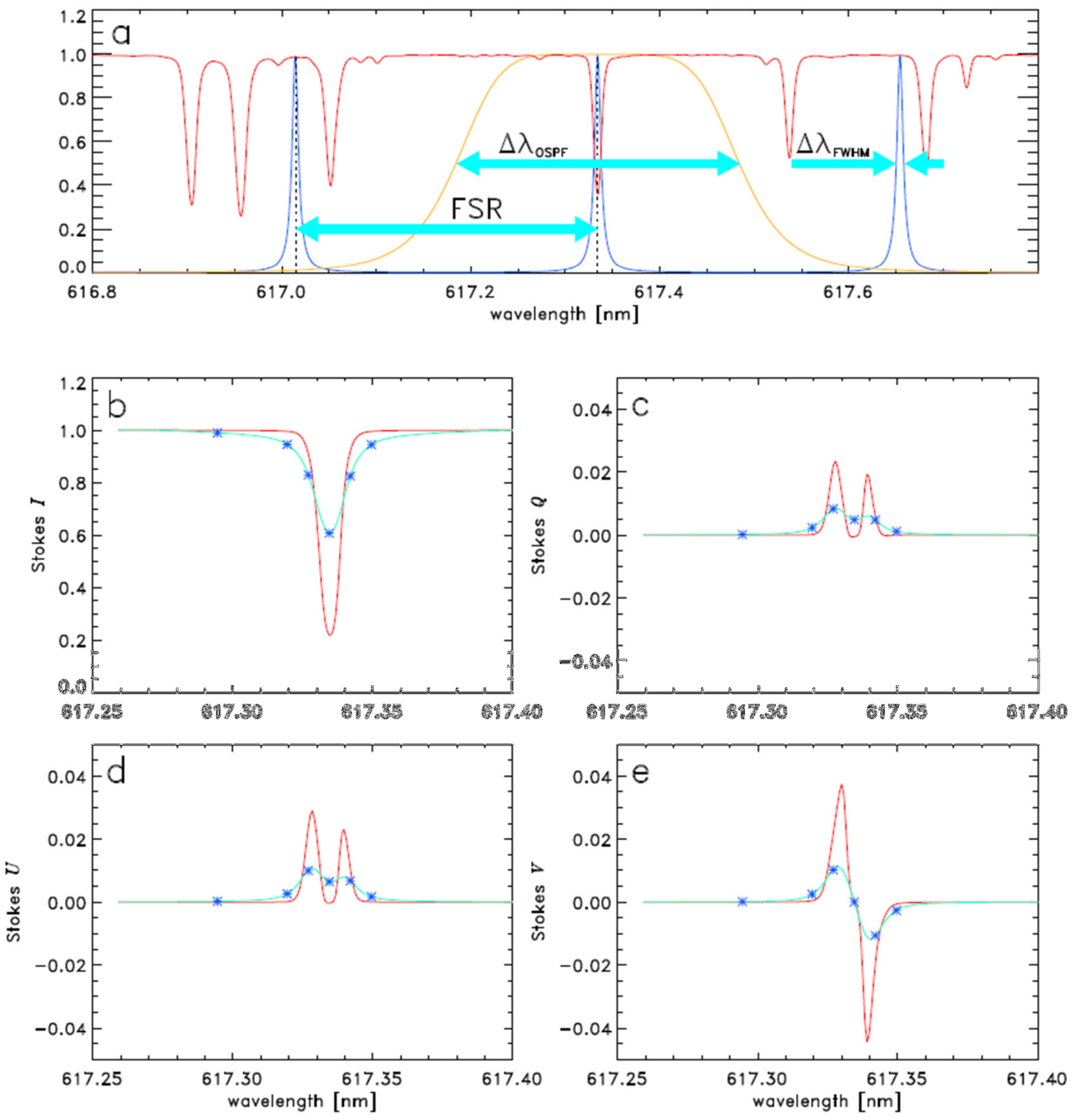


Figure 3 Measurement principle of SO/PHI. (Image adapted from [Hirzberger (2015)]).

1.4 The SO/PHI data processing pipeline

The conceptual light path is depicted in Figure 2 with brown dotted lines. The light is recorded by the CMOS sensor after a polarization analysis by the polarization modulation package and a selection of wavelength within the spectral line by the filtergraph. Raw data, as they will be produced by the SO/PHI science detector within the focal plane assembly (FPA), are built up of narrow band filtergrams (images) obtained in the spectral region around the Fe I absorption line at 617.3 nm (the light level will be modulated by the PMPs).

Several operation principles can be tuned for acquisition of data. The science observing modes define the telescope to use (HRT or FDT), the number (and wavelengths) of the spectral scan positions (6 as a baseline) the number of polarization states at each spectral position (4 baselined for *I*, *Q*, *U*, and *V*), and the number of images to be accumulated at each polarization state and spectral position.

A conceptual overview of the data pipeline is shown in Figure 4 with a block diagram. Solid black lines denote the baseline data processing pipeline; dashed red lines represent additional/alternative processing modules, and denote the possibility to store and downlink raw and partially processed data. We refer the interested reader to [Hirzberger (2012)] for details.

In order to achieve a signal-to-noise ratio (SNR) of 10^3, several images (separately for each polarization sate) have to be accumulated. An exposure time of the order of 1 s per polarization state is required to obtain the required SNR. This means that around 22 images have to be accumulated. The size of the frames is 2048x2048 pixels. However, the number of rows can be specified in the science operation mode (0–2047). At every spectral position and polarimetric state a number of frames will be accumulated in real time. Single frames have a digital depth of 14 bits and accumulated images have 32 bits digital depth. A frame rate of 10 s^{-1} corresponds to a read-out rate of 560 Mbits/s.

For each single science data set (one scan across the Fe I line at 617.3 nm), 24 accumulated images (6 spectral scan positions and 4 polarization states) are acquired, accounting for a raw data size of 3.2Gbit (for a frame size of 2048 × 2048 pixels).

After data accumulation, several basic data calibration procedures have to be applied, so dark current and flat field calibrations are carried out. The polarization of the incoming light is modulated and analyzed by the PMPs. This means that that the fraction of light with a certain polarization state is transformed into a given intensity level, which is a linear combination of the four Stokes parameters. The process of recovering the Stokes parameters from those combinations is known as demodulation. That polarization demodulation does not affect to the size of a raw data set: 3.2 Gbit.

Through the study of the polarization state of the specific spectral line, we can infer the magnetic field vector and other quantities that define the physical state of the solar photosphere as the continuum intensity and the velocity of the solar plasma in the light-of-sight [del Toro (2003)]. Technically speaking, the inference of the solar physical quantities from the spectropolarimetric observations is based on the RTE inversion.

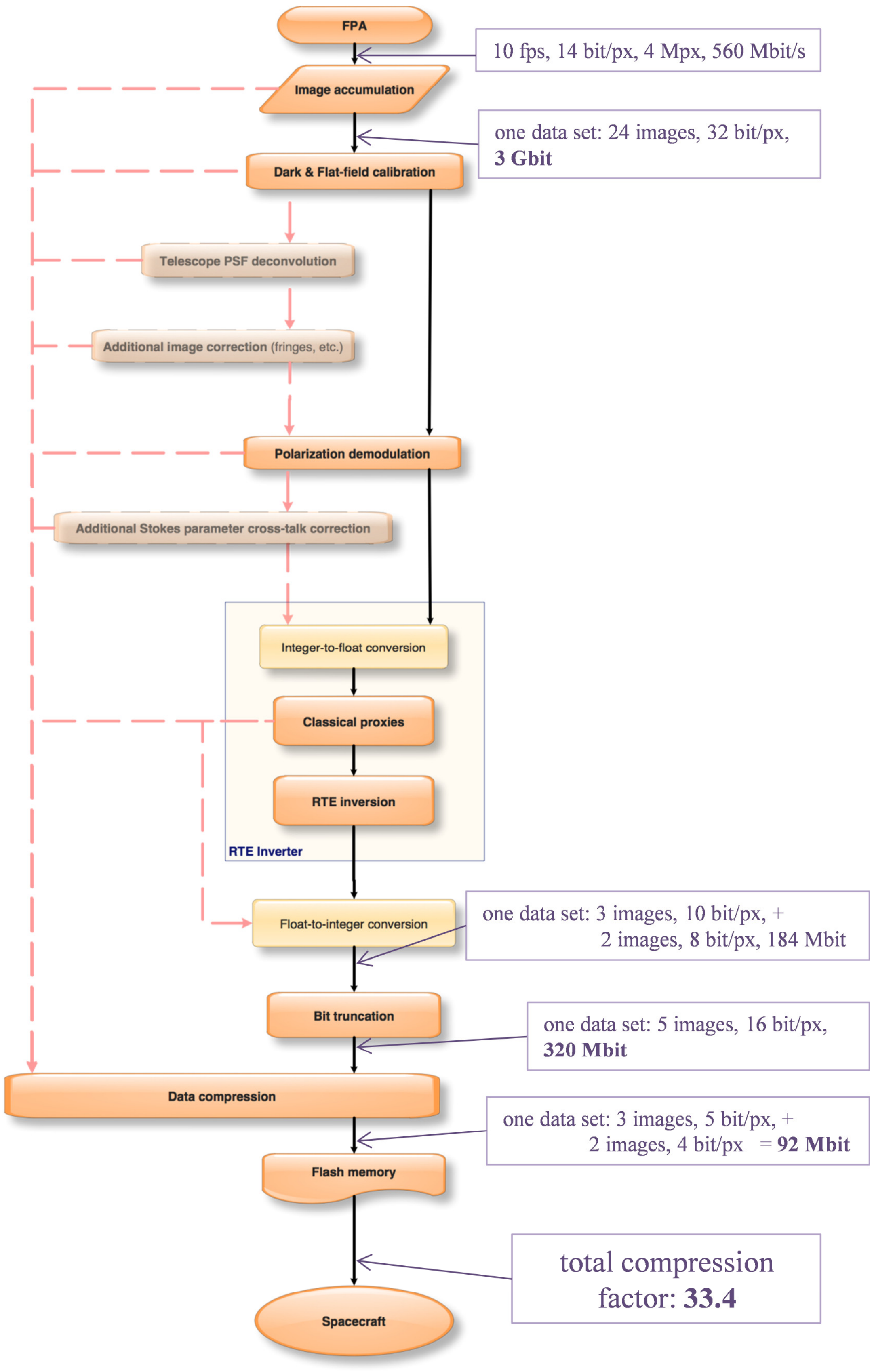

FPA
10 fps, 14 bit/px, 4 Mpx, 560 Mbit/s
Image accumulation
one data set: 24 images, 32 bit/px, 3 Gbit
Dark & Flat-field calibration
Telescope PSF deconvolution
Additional image correction (fringes, etc.)
Polarization demodulation
Additional Stokes parameter cross-talk correction
Integer-to-float conversion
Classical proxies
RTE inversion
RTE Inverter
Float-to-integer conversion
one data set: 3 images, 10 bit/px, + 2 images, 8 bit/px, 184 Mbit
Bit truncation
one data set: 5 images, 16 bit/px, 320 Mbit
Data compression
one data set: 3 images, 5 bit/px, + 2 images, 4 bit/px = 92 Mbit
Flash memory
total compression factor: 33.4
Spacecraft

Figure 4 On-board data processing pipeline.

Thanks to the RTE inversion the raw data set can be reduced from 24 images to only 4 images: the line-of-sight (LOS) flow velocity and the strength, inclination, and azimuth of the magnetic field vector. The continuum intensity, which is not an output from the inversion procedure, is also a valuable fifth image. The inversion results enable the representation of the physical quantities with a shorter fixed point precision than the raw polarized images. Therefore, these quantities are candidates of a bit truncation.

The last step in the data processing pipeline is a lossless compression procedure. The RTE inversion results also help in increasing the effective data compression. As illustrated in Figure 4, data after the RTE inversion are truncated and compressed. Hence, the amount of data is reduced to only 92 Mbit per individual set. In short, we can say that the SO/PHI data processing pipeline, using **the on-board scientific analysis**, **reduces the data** to be transferred from the spacecraft to ground **in a factor higher than 33**.

During normal science operation, only fully-processed and compressed data are expected to be downlinked. During commissioning and check-out phases it is, however, planned to downlink raw and/or partly processed data, as is illustrated with dashed red lines in Figure 4.

In summary, the RTE inversion on board implies to do the scientific analysis of 24 images of 2048x2048 pixels, which makes it easy a final compression factor of around 33. In this thesis we study the RTE inversion algorithm and we will propose two computational architectures for that task not to be a bottle neck in the instrument regular pipeline.

1.5 The Data Processing Unit in SO/PHI

The complexities of the acquisition tasks and of the data processing pipeline imply that steps like scientific data extraction and data evaluation have to be conducted within an instrument Data Processing Unit (DPU), which is otherwise responsible of the overall behavior of the whole instrument.

The basic structure proposed for the PHI DPU design is based on the results of the ESA study for a Dynamically Reconfigurable Processing Module (DRPM) [Bubenhagen (2010); Fossati (2011)]. This design utilizes a combination of a processor ASIC (GR712C from Aeroflex-Gaisler), containing a LEON-3FT-based main processor system, together with a fixed, radiation hardened and TMR by design, one-time programmable Microsemi RTAX FPGA as system supervisor plus a set of dedicated, real-time function cores implemented within in-flight reconfigurable Xilinx Virtex-4 FPGAs (XQR4VSX55) [Fiethe (2012)]. A DPU schematics is shown in Figure 5. Other alternatives of DPU architectures were rejected due to availability and performance because of power consumption reasons, as demonstrated in [Bubenhagen (2013)].

A combination of a small amount of volatile (e.g. 8 Gbit SDRAM) and large capacity of non-volatile (e.g. 4 Tbit NAND Flash) image memory provides significant storage capacity, which fulfills all needs of intermediate data storage at very low resources [Fiethe (2012)]. The design of the NAND-Flash-based system has complete error correction, taking into account the Flash handling. NAND-Flash-based mass storages for space have been intensively studied by Institute of Computer and Communication Engineering (IDA) in the Safe Guard Data Recorder (SGDR) study for ESA and have already been implemented for the Sentinel-2 SSMM [Cassel (2011)].

It is important to remark that the 4 Tbit non-volatile NAND flash memory allows storing data in order to schedule and postpone the data processing. In the nominal science observing mode, one full set of observables will be obtained within 60s. This time range includes frame exposure and data accumulation. The high cadence of data acquisition in most operation modes and/or the lack of DPU processing capacity for carrying out all necessary tasks in due time implies for the pipeline to have an option for storing raw or partially calibrated data in an image storage. The full data processing pipeline will thus be executed during periods where no observations are carried out. This idea of postponed processing is described in Figure 5, and detailed in [Fiethe (2012)].

Thanks to the sufficient flash memory storage available, and to the Time-Space Partitioning (TSP) proposed by IDA [Fiethe (2012)], the entire advanced data processing pipeline is done only with the two radiation-tolerant in-flight reconfigurable Xilinx Virtex-4 FPGAs. For the PHI DPU, the seamless reconfigurability of these FPGAs enables multiple uses during different modes of operation:

- the first configuration uses the two FPGAs for image acquisition; it is remarked in Figure 5 using red blocks:

  • image stabilization system (ISS) in FPGA #1

  • data accumulation in FPGA #2

- a second, different configuration for the subsequent data processing is depicted in Figure 5 using yellow and blue remarks:

  • RTE inversion in FPGA #1

  • data pre-processing and compression in FPGA #2

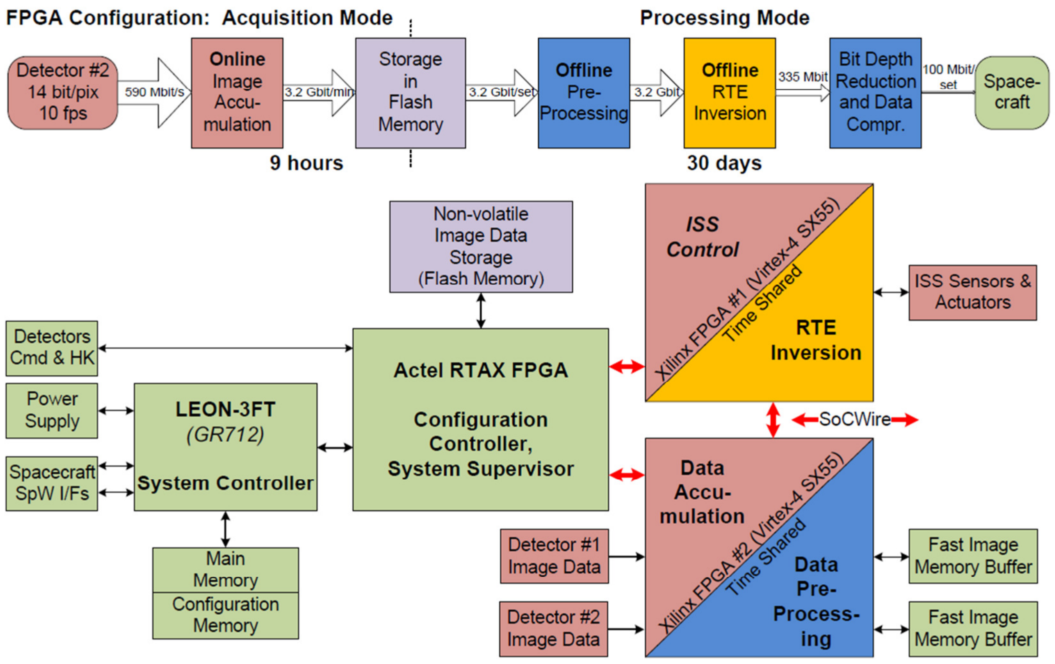


Figure 5 PHI basic data flow (up) and DPU architecture block diagram with FPGA time sharing (down). Adapted from [Bubenhagen (2013)].

In short, to allow the DPU to work with only two Virtex-4 FPGAs, and make it feasible the Time-Space Partitioning within the SO/PHI instrument, the RTE inversion must be carried out using one of the two Virtex-4 FPGAs embedded in the PHI DPU. That is the same that is used to perform the Image Stabilization System (ISS) during the data acquisition phase. In the context of the SO/PHI instrument, the subsystem for carrying out the RTE inversion is also called RTE inverter and will also be referred to so along this work.

In early stages of the PHI DPU design, the use of the space-qualified Virtex-5 QRFX130T FPGA was considered. Unfortunately, this part is not yet qualified by ESA, mainly for packaging and soldering issues and, finally, the Virtex-4 XQR4VSX55 was the accepted device by ESA. Nevertheless, a specific qualification process about the assembly of the Virtex-4 XQR4VSX55 package (CF1140 package) had to be carried out by the SO/PHI team [Fiethe (2014)]. This mission-specific qualification was needed since no qualified process manufacturer for such a ceramic flip-chip column grid array package assembly is currently available in Europe.

The first DPU design, using Virtex-5, did not count with the enormous massive memory, and the first attempt was to process the raw data at the same time they were taken (online), in a minute. In the final DPU design using two Virtex-4 FPGA, thanks to the instrument internal memory, which allows storing up to one observation cycle (about 9 hours) [Hirzberger (2013-

B)], approximately 15 minutes are available to the system to perform the inversion of the RTE for each single observational data set (obtained in one minute).

The Virtex-5 device was the baseline for the DPU design (and so for RTE inverter) during more than two years. We designed and developed a first prototype for that device. Finally, after the ESA decision, we had to redesign all the system for arranging the inversion to the Virtex-4 device. As Table 1 shows, both devices are very different, not only in number and size of available resources but also in the technology of these resources. Virtex-5 Configurable Logic Block slice contains four LUTs and four flip-flops (in Virtex 4 they have two LUTs and two flip-flops). In addition, each DSP48E slice of Virtex-5 contains 25 x 18 multipliers, meanwhile each DSP slice in Virtex-4 contains 18 x 18 ones. A study about the feasibility of using other space-qualified devices is given in [Cobos (2011)], where higher members on the qualified Virtex-4 family had to be discarded due to power consumption issues.

The Virtex-4 is based on 90 nm copper CMOS Process and it uses a 1.2 V Core Voltage. On the other hand, Virtex-5 is based on a 65 nm copper CMOS process technology, and it uses a 1.0 V core voltage. That means that Virtex-5 is a better, faster device and more efficient in power consumption. However the final system will use the Virtex-4, emphasizing even more the processing challenge in the space.

The device change in the DPU design is the reason why in this thesis two processing architectures are proposed, one for each DPU design, and both are original architectures.

	Devices	
Resource	Virtex 4 XQR4VSX55	Virtex 5 XQR5VFX130
Slices	24,576	20,480
Block Ram	320 (18 Kb)	596 (18 Kb)
Math Blocks	512 (400 MHz 18 bit x 18 bit)	320 (360 MHz 25 bit x 18 bit)
CMOS Tech.	90 nm	65 nm
Core Voltage	1.2 V	1 V

Table 1 Elements that compose the Virtex-4 and Virtex-5 FPGAs

Regarding the engineering requirements, it is important to remark that the whole DPU has to operate with a maximum of 35 watts, and the assignation for the FPGA when the RTE inversion is running is only 7 watts for the Virtex-5 device and 5 watts for the Virtex-4 one.

1.6 The PHI Team

To build such a complex, challenging, and novel instrument is an enormous task that would not be possible without a big consortium. However, its core elements either have space heritage or are based on technologies already developed by the SO/PHI team for the successful balloon-borne *Sunrise* mission. The PHI consortium has a long-standing expertise in designing and building scientific instruments for space missions and their scientific exploitation. The PHI instrument is being developed with major contributions from institutes in Germany, Spain and France, and with smaller contributions from Sweden, and Norway.

The hardware and/or software contributing institutes are (in alphabetical order):

- Grupo de Astronomía y Ciencias del Espacio (GACE), Universidad de Valencia, Spain
- Institute of Computer and Communication Engineering (IDA), Braunschweig, Germany
- Institut d'Astrophysique Spatiale (IAS), Paris, France
- Instituto de Astrofísica de Andalucía (IAA), Granada, Spain
- Instituto de Astrofísica de Canarias (IAC), Tenerife, Spain
- Instituto Nacional de Técnica Aeroespacial (INTA), Madrid, Spain
- Instituto Universitario de Microgravedad "Ignacio Da Riva" (IDR), Universidad Politécnica de Madrid, Spain
- Institutt for Teoretisk Astrofysikk (ITA), Oslo, Norway
- Kiepenheuer-Institut für Sonnenphysik (KIS), Freiburg, Germany
- Max-Planck-Institut für Sonnensystemforschung (MPS), Göttingen, Germany
- Universidad de Barcelona (UB), Spain

The PHI consortium is represented by the Principal Investigator Prof. S. K. Solanki (MPS) and the Co- Principal Investigator Dr. Jose Carlos del Toro Iniesta (IAA).

The IAA has an important role in SO/PHI which includes the Co-Principal Investigator as well as the coordination of the Spanish consortium which is formed by the following institutions: INTA, GACE, IDR-UPM, UB and IAC. The level of participation of the Spanish consortium in the PHI instrument is slightly above 40%, including responsibilities at system level which are performed at IAA.

The IAA, as the coordinating institute, acts as a contact point with the PI project office at the MPS in Göttingen (Germany), which guarantees a high visibility of the contribution. Apart from an important contribution to the scientific side, the IAA has also a relevant contribution to the technical team.

In addition to these high level tasks, the IAA is also responsible for two of the three SO/PHI instrument's Units: the Electronics Unit and the Harness as well as several important work packages at lower level as are the Electrical Distribution System (EDS), the Analog and Mechanisms Driver control board (AMHD) and the inversion of the Radiative Transfer Equation (RTE), all of them included in the E-Unit.

The EDS is an electronic board in which the distribution of the signals between the E-Unit subsystems is implemented. The AMHD is another board included in the E-Unit which is in charge of performing the PHI instrument motor and heaters control as well as the housekeeping acquisition. Finally, the RTE is the core of the on-board scientific analysis and, probably, one of the most innovative parts of the instrument. It consists on a device specifically designed to invert the Radiative Transfer Equation that will be on board *Solar Orbiter*. That denotes the importance of this thesis work within the SO/PHI instrument.

The responsibility of the IAA about the E-Unit as a whole implies the coordination of the several institutions that contribute to this unit; these are IDA (at the TU Braunschweig), which develops the Digital Processing unit (DPU); IAS (at the University of Paris-Sud), which contributes the High Voltage Power Supply (HVPS); Universitat de Barcelona (UB), which is responsible for the image stabilization system electronics (Tip Tilt Controller TTC); and GACE (at University of Valencia), which develops the Power Converter Module (PCM) of PHI.

As the E-Unit responsible, the IAA is in charge of supervising the development of all E-Unit subsystems from management and quality assurance points of view, controlling the schedule and leading the integration and verifications campaigns of the different E-Unit models.

1.7 Objectives of this thesis

In the last years, in the high-performance computing realm, the trend is to use multi-core architectures for accelerating scientific applications. However, in an extreme environment like the outer space, we cannot use the common devices like clusters of PCs or General Purpose Graphics Processing Unit devices (GPGPU) due to radiation problems and to power limitations.

The hypothesis of this thesis is to demonstrate that a space-qualified FPGA device can be used as a high-performance computing element and which will be able of carrying out the RTE inversion aboard the SO/PHI instrument. As we mentioned above, this will be the first time that such a computing demanding process is performed aboard a spacecraft. The final FPGA system can be considered the heart of the entire instrument data pipeline.

Therefore, the main aim of this thesis is to design and develop embedded architectures using the current high-performance computing techniques like multi-core architectures, code optimization, or specific-domain efficient processors, which would be able of reaching high-performance computing on an FPGA. In addition, the final design must satisfy the scientific requirement of computing precision and the engineering requirements of computing time and power consumption, all of that, without forgetting the final space environment for which the system is envisaged.

The proposed computing architectures, despite of focusing in the RTE inversion, shall be enough versatile for arranging to future changes in the scientific requirements, as spectral line, number of wavelengths, etc. That will lead to architectures with some capacity of configuration. Therefore, a secondary objective will be to provide the architectures with software tools that facilitate their configuration.

This thesis in computer science is aimed to further the research in high-performance computing and to propose innovative architectures for embedded computing. It is not an objective of this thesis to generate a detailed engineering document, or manual, with all the features and requirements of the proposed architectures focused in the space mission. For that, a great amount of technical notes and documentation has been generated within the framework of the SO/PHI international project, which will be referred to along this work. These documents are available upon request to the SO/PHI team. The requests will be examined on a case by case basis. However, we have tried to give a self-consistent document with all the necessary details, as well as, a summary of several technical notes in the appendices. Therefore, this thesis will mainly focus in explaining new computation strategies and how they are implemented using FPGA devices for carrying out scientific algorithms.

1.8 Methodology and thesis structure

The development of this thesis is bounded to the development of the SO/PHI instrument. So the confirmation of the initial hypothesis is conditioned to the success of SO/PHI and of the RTE inverter subsystem. Since the space instrumentation context imposes a rigorous methodology based on the ESA standard for ASIC and FPGA Development [ESA (2008)] the RTE inverter development shall follow this standard.

The methodology for designing and developing the RTE inverter is detailed in the RTE inverter FPGA Development Plan [Aparicio (2013)]. There, all the generated engineering documentation is detailed, and the entire followed design flow, for the RTE inverter within the space instrument context. The RTE inverter FPGA development shall follow the life cycle shown in the Figure 6, where once the specifications and requirements are established, a detailed design will be developed. This design will be simulated at a block level, and the different blocks will also be tested on the FPGA. That is, the FPGA will be programmed with the different blocks and they will later be tested. Furthermore, this process will be repeated at top level. This is reflected in the figure as RTE simulations and RTE post programming test. Then, the RTE functional test will be done. All these tests will be carried out using the commercial FPGA version of our space-qualified Virtex-4. Finally, the electrical tests and functional test will be carried out over the flight FPGA, once integrated in the DPU board. All these steps will generate all the associated documentation appearing in Figure 6.

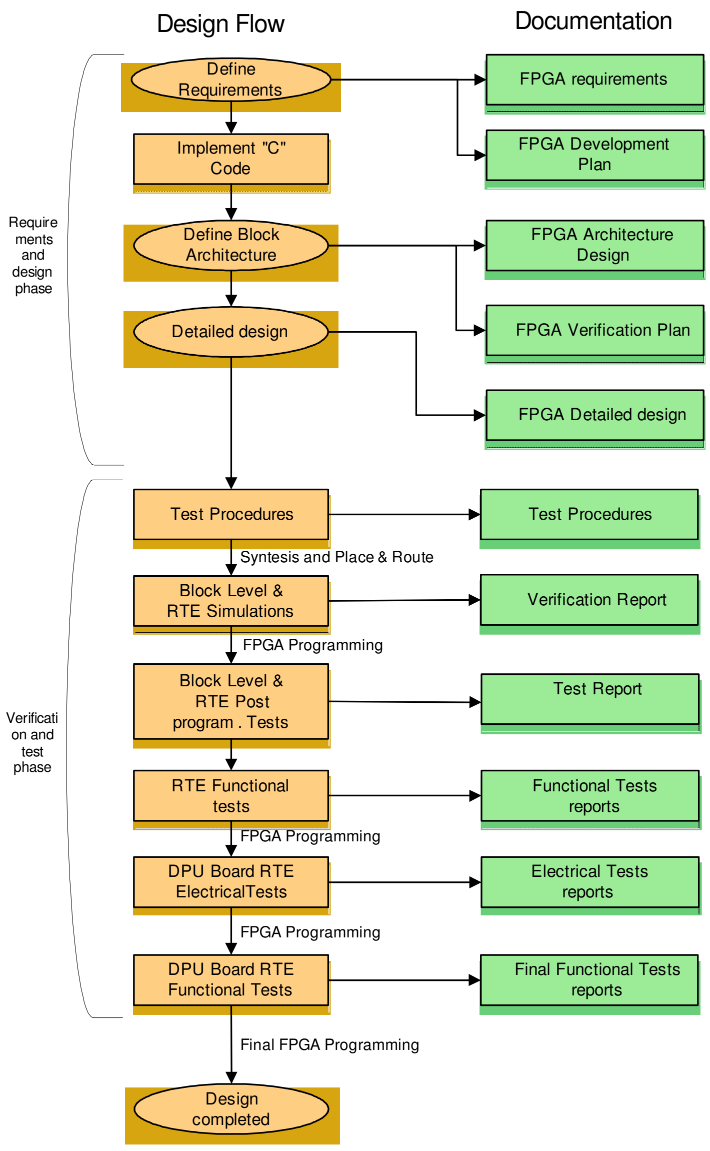


Figure 6 RTE inverter FPGA design flow

As Figure 6 explains, the first activity is to define the RTE requirements, including both scientific and technical. These requirements shall be documented within the RTE inverter FPGA requirement specification [Aparicio (2014)]. This document shall include a minimum set of items according to the ESA standard [ESA (2008)], so it specifies the electrical, operational, functional, and science requirements of the RTE inverter with high grade of detail. Every requirement shall have its corresponding verification procedure and test reports. The main requirements for understanding the work in this thesis are outlined in Table 2.

IDENTIFIER	Requirement Definition
RTE-R-0010	RTE shall produce maps of : • i.- the photospheric line-of-sight (LOS) flow velocity (v_{LOS}), obtained from the polarized profiles of that line, • ii.- the photospheric magnetic field strength (B_{LOS}), obtained from the polarized profiles of the line, • iii.- the photospheric magnetic field inclination (γ) with respect to the LOS, obtained from the polarized profiles of the line, and, • iv.- the photospheric magnetic field azimuth (ϕ) in a plane perpendicular to the LOS, obtained from the polarized profiles of the above mentioned line.
RTE-R-0020	The RTE inverter FPGA maximum power dissipation shall be 7 W for the Virtex-5 and 5 W for the Virtex-4
RTE-R-0030	The RTE inverter shall process a full set of images of each observation in 1 minute using the Virtex-5 device and less than 15 minutes using the Virtex-4 device

Table 2 Outline of the main RTE inverter requirement specifications

After this phase, a C code shall be implemented with the RTE operations. This way is easier and faster to make a first approach of RTE functionality. Then the block architecture shall be defined. This phase will be documented with the FPGA architecture design, and shall have a verification plan associated, explaining the procedures and test to verify the requirements. In [Cobos (2013)] is presented the RTE inverter FPGA validation and verification plan to follow in the RTE inverter development. It explains the needs for demonstrating whether the functional and non-functional requirements stated in the definition phase documentation are accomplished at all levels of modeling, starting from the behavioral level down to the RTL level.

It is important to remark that the Project Manager and the Assistant Manager supervise the validation and implementation tasks according to ESA standards for FPGA/ASIC developments, under the leadership and scientific assessment of the SO/PHI co-PI. Finally, an external company, SENER, with a long tradition in space missions, supervises and certificates all the development process and the documentation.

Thus, beginning from the state of the art in the field of high-performance computing embedded on FPGA, innovative architecture proposals will be generated that tackle the RTE inversion on FPGA. These architectures will be compared with the current ones. The final validation of the architecture proposals will be mainly based on the execution of the RTE inversion algorithm and on the requirements verification. If that validation is satisfactory the initial hypothesis will be corroborated. Anyway, the SO/PHI documentation will be constantly referred to as the powerful complement that it represents.

This document is structured in 9 main chapters. The next one deeply studies the RTE inversion algorithm from a point of view of computation, although some scientific concepts have to be introduced in order to clarify the most important issues. This study is necessary in order to know all the parallelization and optimization possibilities that the code intrinsically possesses. In addition, we will show the necessity of developing a core for the Singular Value Decomposition of a correlation matrix.

The third chapter presents the most relevant trends and works in the high-performance computing on the FPGA field. Based on it, two architectures, conceptually different, are proposed, but both aimed to the RTE inversion problem, and focused in obtaining high-performance computing. Each architecture is devised to one specific family of FPGAs considered in the DPU design. The fourth chapter details the design for the Virtex-5 FPGA, and the fifth one does the same for the Virtex-4 FPGA version. Since, the version for the Virtex-4 is the finally accepted by the ESA, the architecture is described with more detail.

In chapter number six, a software tool that allows the configuration and modification of the computing architectures is presented. This tool supplies a compiler that facilitates the programming tasks, and also the migration of the RTE inversion algorithm from a high-level programming language to the embedded architecture level. The tool counts with two different versions, one for each possible architecture and device.

The seventh chapter shows the design for the mathematical operation of Singular Value Decomposition of a correlation matrix.

In the eighth chapter some aspects related to the final RTE inverter systems like fault mitigation, algorithm scheduling, and configuration and use from an instrument point of view are explained.

Chapter nine shows how the RTE inversion algorithm is executed within the proposed architectures. Tests using artificial and real images are presented. A comparative with other RTE inversion approaches is done.

The tenth chapter concludes with comments and remarks about the most important innovations of the proposed solutions in this thesis.

2

The Radiative Transfer Equation inversion

The limited telemetry rate combined with the large amount of scientific information retrieved from the SO/PHI instrument demand a sophisticated on-board data reduction and analysis. As pointed out in the introduction, the RTE inversion is the heart of that analysis. In this chapter, we describe the RTE inversion basics, the process, and its algorithmic implementation.

2.1 The RTE inversion in SO/PHI

Through a spectropolarimetric study of a particular spectral line formed in the photosphere we can infer the magnetic field and other quantities that define the physical state of the solar atmosphere [del Toro]. Spectropolarimetry is a spectroscopic analysis of light whose polarization state has been previously analyzed. As a result of this analysis, quasi-monochromatic images of the Stokes parameters (I, Q, U, V) are obtained at different wavelengths across the line and its nearby continuum -six wavelength in the SO/PHI case. The information about the solar magnetic field and plasma velocity is encoded in the spectrum of the Stokes parameters. Without going into details, we can say that the process of getting the solar atmospheric parameters from the Stokes parameters is known as the inversion of the Radiative Transfer Equation (RTE) [del Toro (2003)].

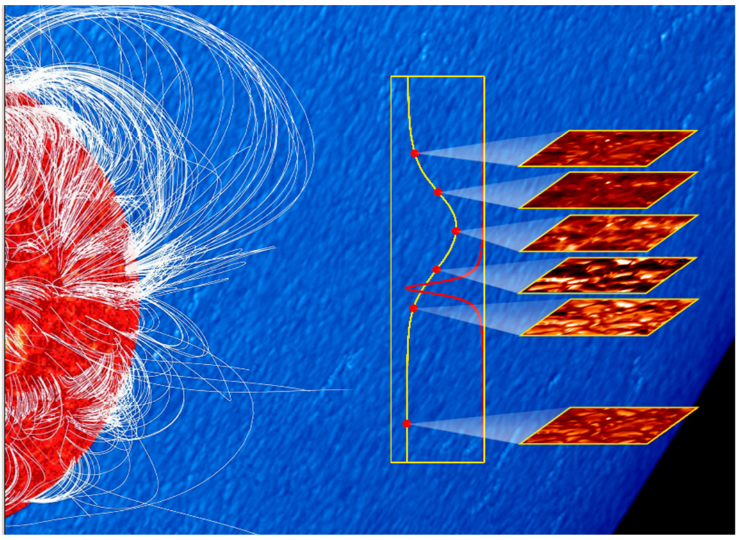


Figure 7 Illustration of the wavelength sampling

Specifically, we shall employ an inversion method based on the Milne-Eddington (ME) solution of the RTE. In it, the atmospheric parameters are assumed to be constant with depth. This approximation assumes a linear dependence of the source function on the optical depth and that several physical quantities like the magnetic field strength, the flow velocity, the microturbulence, etc. do not change with height. No macro- or microturbulent velocities will be assumed throughout this study. The ME approximation reduces significantly the computation time. However, the fit quality deteriorates for asymmetric Stokes profiles, where vertical gradients of the physical quantities are present in the solar photosphere.

In a Milne-Eddington atmosphere model, besides the so-called thermodynamic parameters (the line-to-continuum absorption coefficient ratio, η_0, the Doppler width of the line, $\Delta\lambda_D$, the damping parameter, a, and the two coefficients defining the source function, S_0 andS_1) that govern the shape of the Stokes spectrum, the line-of-sight (LOS) velocity (v_{LOS}), and the strength (B), inclination (γ), and azimuth (ϕ) of the vector magnetic field fully describe the atmosphere.

A single broadening, corresponding to the instrumental point-spread-function will be considered in the development. Since it is likely very wide (approximately 10^{-1} Å), its effect on the profiles overrides other possible broadenings.

In Figure 8, a sketch is presented about how the Stokes profiles are obtained from the initial polarized images for each spatial pixel. For each point on the solar surface, codified in one spatial pixel, we have four Stokes "profiles" of six elements each (the Stokes parameters depend on wavelength). Thus, for each point (i, j) on the surface of the Sun, we have an Stokes profile with the shape of I={I(i, j, $\lambda_{0..5}$), Q(i, j, $\lambda_{0..5}$), U(i, j, $\lambda_{0..5}$), V(i, j, $\lambda_{0..5}$)}. The lower panels of Figure 8 show an example of a sampled Stokes profile Note that the sixth wavelength sample is not taken within the spectral line but on its nearby continuum. The image composed by I(i, j, $\lambda_{0..5}$) is also called continuum intensity image, or simply I_c.

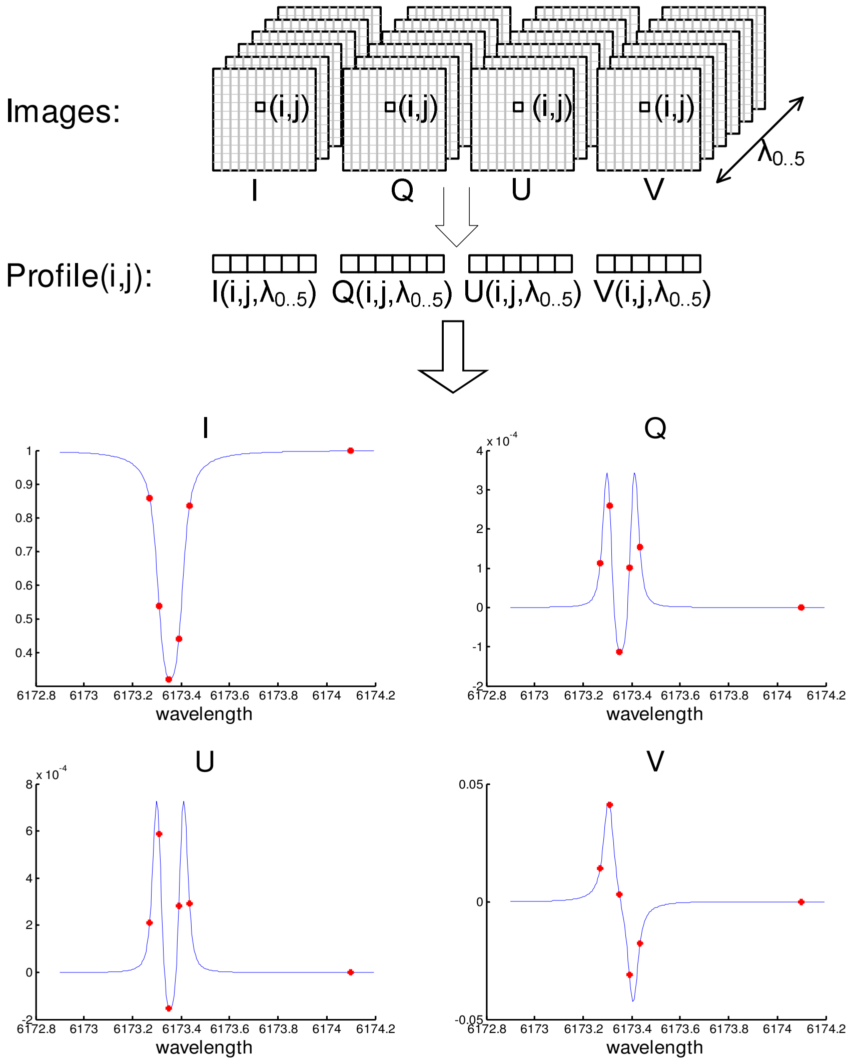


Figure 8 Diagram about how Stokes profiles are obtained from the initial images (UP). Example of a sampled Stokes profile (red dots) of the SO/PHI spectral line (blue line).

Such data set consists, then, of 24 images of 2048x2048 pixels, corresponding to the four Stokes parameters recorded at six wavelengths. From each one of these 2^{22} sets of four Stokes profiles we will extract the relevant solar information, since each of them can be represented using a ME atmosphere model. For each Stoke profile the RTE inversion has to be carried out. Figure 9 shows a simplified RTE inversion block diagram, where the iterative character of the algorithm is apparent. Indeed, it is a non-linear least-squares procedure through which synthetic

Stokes profiles are iteratively fit to the observed ones. The inversion procedure starts with an initial guess of the physical parameters. Then a synthesis of the corresponding Stokes spectrum by numerically solving the RTE is carried out. By iteratively changing the initial guess for the parameters in a closed feedback loop, the differences between synthetic and observed Stokes profiles are minimized and, thus, the desired parameters are obtained. Such a process is known as the inversion method of the Radiative Transfer Equation, or simply, RTE inversion and is illustrated in Figure 9.

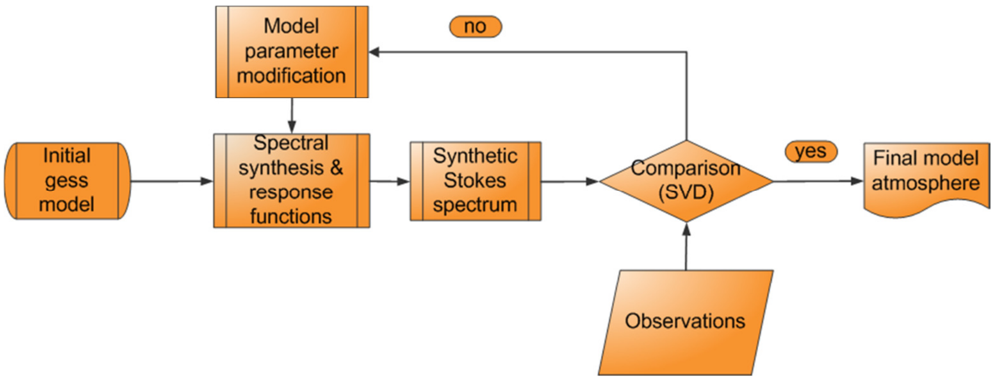

Model parameter modification
no
Initial gess model
Spectral synthesis & response functions
Synthetic Stokes spectrum
Comparison (SVD)
yes
Final model atmosphere
Observations

Figure 9 RTE inversion process block diagram

As a result of inverting the 2^{22} Stokes profiles, instead of the original 24 images, only 4 will remain corresponding to v_{LOS}, B_{LOS}, γ, and ϕ (as summarized in Table 2), hence reducing the amount of data by a factor almost 6. Subsequent truncation and conventional compression tasks will help to reach a final reduction factor higher than 30, as detailed section 1.3.

Figure 10 shows an RTE inversion result. The original *Q*, *U*, and *V* images are not shown for simplicity. Only the continuum intensity is shown. These original images correspond to the largest active region of the current solar cycle. They were taken on the 25th of October, 2014 using the spectropolarimeter aboard the *Hinode* spacecraft [Kosugi (2007)] in the Fe I spectral line at 630.2 nm.

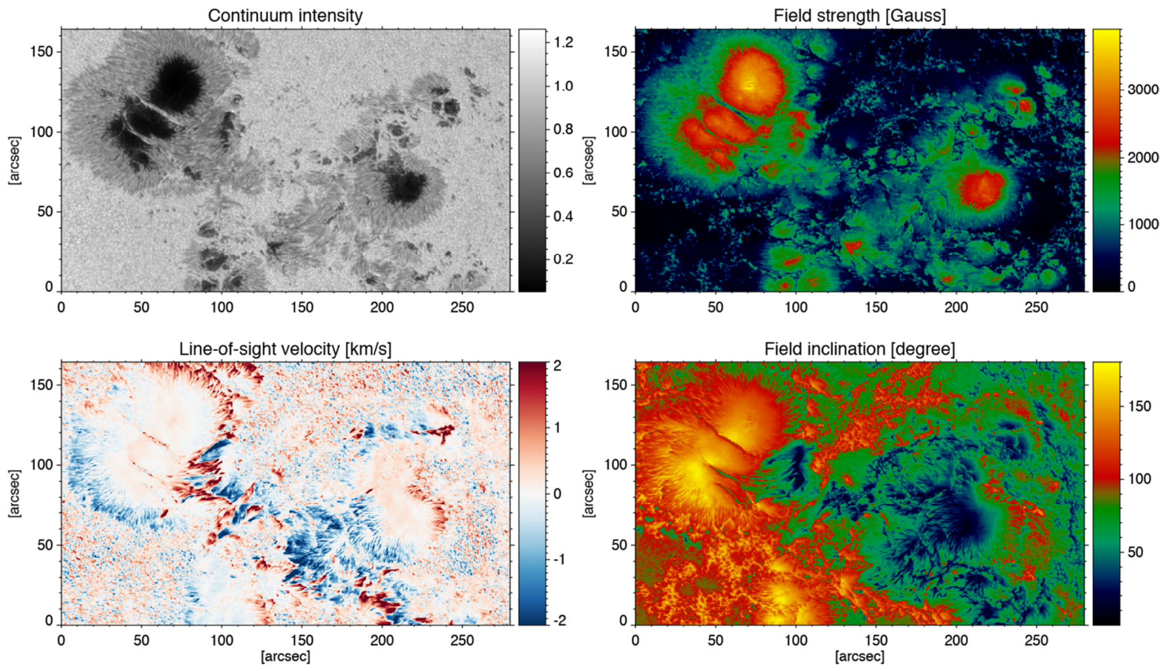


Figure 10 Example of an RTE inversion of Hinode images. Continuum intensity, magnetic field strength (Gauss), field inclination (degree), and LOS velocity (km*s-1) are shown clockwise from top left.

2.2 The RTE inversion algorithm

The RTE inversion process used in this work is thoroughly described in [Orozco (2007, 2008)], where the MILne-Eddington inversion of pOlarized Spectra (MILOS) code was presented. This inversion code, written in IDL language, uses a Levenberg-Marquardt iterative scheme based on the minimization of a merit function.

Figure 11 details the steps (or tasks) included in every single turn of the RTE iterative loop. Those tasks are:

- Stokes spectral Synthesis from a model atmosphere, Model$_i$ (hereafter referred to as SYNTHESIS),
- calculation of the Response Functions (RFs). They provide the sensitivity of Stokes profiles to the model free parameters. Hence, there are nine RFs per Stokes profile,
- convolution of the synthesized spectrum and the RFs with the filter transmission curve – the instrumental profile, which represents the broadening distortion produced by the instrument (CONV),
- estimation of the fit quality by means of the root mean square difference (χ^2) between the observed and synthetic Stokes profiles (CHISQR) for the current model,

- calculation of the correlation matrix between the observed and synthetic profiles (COVARM),
- Singular Value Decomposition (SVD) of the correlation matrix in order to modify the initial model,
- modification of model atmospheric free parameters using eigenvalues and eigenvectors (MODPAR).

If χ^2 is larger for Model$_i$ than for Model$_{i-1}$ in CHISQR, then the latter is recovered as the current one. In addition, the iterative process can be stopped when χ^2 reaches a value below a given threshold, in order to reduce the final number of iterations. This possibility is not shown in Figure 11 for simplicity

An out-of-the-loop block that obtains classical estimations of the physical parameters to initialize the model atmosphere is also shown in Figure 11. Those estimations also provide quick approximations to the parameters. This block is not in the original MILOS code, but it has been introduced later as result of this work. Thanks to the use of this block, the number of iterations to reach convergence is lower than using a previously fixed, or random, initial model. The typical number of iterations to achieve convergence is around 15. This number can even be reduced through the use of classical estimations, as shown in Section 2.4. The classical estimate formulation is presented in [del Toro (2011)].

In addition, the classical estimation task will be an excellent backup solution to the inversion, just in case the challenging goal of inverting the Stokes profiles has not enough allocated time for some specific operational modes. In fact, one of them is already prepared for having just classical estimations as output.

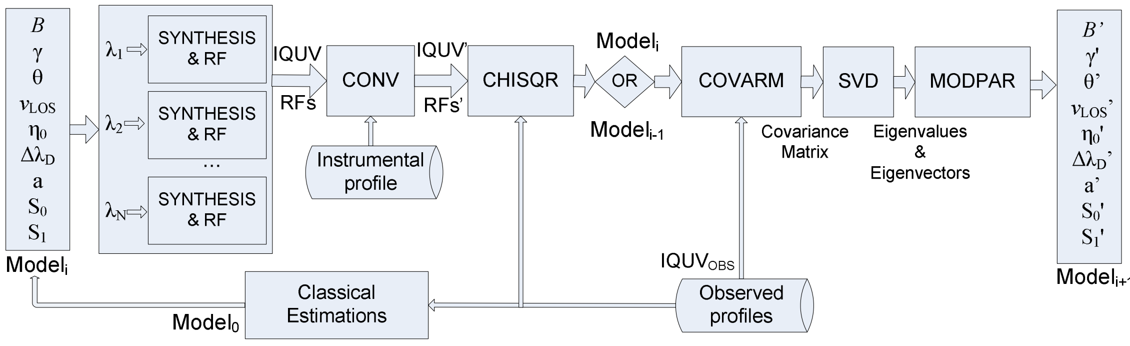


Figure 11 RTE inversion loop task block diagram

2.3 Computational cost of the RTE inversion code

In order to develop the inversion code, the MILOS code (the IDL version) was re-written in C. Its reliability is discussed in next sub-section. It has been called C-MILOS [Cobos (2010-A)]. Since C-MILOS is lower level and faster than MILOS, it was chosen for doing the computational cost study which was presented in [Cobos (2010-B)]. In Table 3, we show the number of operations per type and for the various parts of the inversion code, where:

- N is the used number of wavelength samples in the spectral line,
- NTERM is the number of free parameters of the Milne-Eddington model atmosphere[1],
- N_SIG is the number of sigma red (or blue) Zeeman components of the line,[2]
- N_PI is the number of pi Zeeman components of the line.

	MULTIPLICATION	DIVISION	ADDITION , SUBTRACTION	N_FVOIGT
RF	14*N*(N_PI+2*N_SIG) +511*N+16+ 2*N_SIG+N_PI	2*N* (2*N_SIG+N_PI) +18*N+7	11*N*(2*N_SIG+N_PI) +371*N	
SYNTHESIS	4*N*(2*N_SIG+N_PI) +45*N+16	3*N+2	3*N*(2*N_SIG+N_PI) +37*N	2*N_SIG +N_PI
FVOIGT	50*N	2*N	36*N	
COVARM	4*N*(NTERM*NTERM +2*NTERM+1)	4*NTERM	4*N* (NTERM*NTERM+NTERM+1) +4*NTERM+4*NTERM*NTERM	
CHISQR	4*N+4	5	8*N+4	
MODPAR	2*NTERM*NTERM +NTERM	NTERM	2*NTERM*NTERM	

Table 3 Number of operations for each inversion block.

The SO/PHI spectral line at 617.3 nm is a pure Zeeman triplet so that N_PI and N_SIG are equal to one. The number of samples in the spectral line will be six. And the free parameters in the model atmosphere, NTERMS, will be 9, as commented in section 2.1. In order to perform the SYNTHESIS, four more trigonometric operations are needed, that are not shown in Table 3 for simplicity.

Besides the inversion tasks previously cited, the Voigt function calculation, widely used in spectroscopy (FVOIGT in Table 3), is specified. This function is only used in the SYNTHESIS

[1] Although the nominal number of free parameters is nine, some of them can be chosen to be fixed so that the number of free parameters can be modified.

[2] As mentioned in the Introduction, the interpretation of the Stokes spectrum in terms of the solar physical parameters is through the Zeeman and Doppler effects. Discussing the physics of these effects is certainly far from the scope of this thesis. We refer the interested reader, e.g., to [del Toro (2003)] and references therein.

task but it is regarded as one of the most computationally expensive blocks in the whole code. Its total computational cost depends on the Zeeman pattern of the line that governs the number of executions, N_FVOIGT.

There are some methods that reduce the number of operations in the Voigt function evaluation, as those proposed in [Borrero (2010)], where a Taylor expansion is used to approximate the Voigt function with enough accuracy. However, we will take the original function for computing the spectral synthesis with full precision.

The SVD block is not reflected in Table 3 because the chosen algorithm does not count with a fixed number of operations. It usually depends on the initial matrix. To perform the SVD there are several methods but here we study the two most frequently used and referenced methods. These two methods are explained in [Press (1992)]. Following [Press (1992)], we will call SVDCMP (Singular Value DeCoMPosition) the first one, and TRED+TQLI (Householder maTrix REDuction + Tridiagonal QL Implicit) the second one.

SVDCMP is based on the works by [Golub (1974); Forsythe (1977)] where the authors developed the Jacobi iterative method [Jacobi (1846)] which performs the singular value decomposition on an arbitrary matrix, giving back its eigenvalues and its eigenvectors. An optimum strategy for finding eigenvalues and eigenvectors is, first, to reduce the matrix to a simple form, and then begin an iterative process.

SVDCMP starts from an original matrix and performs the next tasks: a Householder reduction to bidiagonal form [Householder (1970)], accumulation of right-hand and left-hand transformations, and finally a diagonalization of the bidiagonal form. It is important to point out that the last step consists in a loop over singular values to convergence; therefore, SCDCMP is indeed an iterative method on its own. Usually, a maximum number of iterations is established although the method is very robust and that maximum may not be needed for convergence [Press (1992)]. In fact, this method is very stable and that it is very unusual that SVDCMP misbehaves.

	SVDCMP	TRED+TQLI
Addition/Subst.	6358	5625
Multiplication	4372	2830
Division	467	360
Square root	10	7
Total	~11.200	~8.820

Table 4 Average number of operations for 15.000 SVDs performed

On the other hand, the TRED+TQLI method is the fastest known computing technique for finding all the eigenvalues and eigenvectors (or just all the eigenvalues) of a real, symmetric matrix as is pointed out by [Press (1992)]. Since in our case the initial matrix is a correlation matrix, hence a symmetric matrix, we have to study this method.

Instead of trying to reduce the matrix to a diagonal form, for symmetric matrices the preferred simple form is tridiagonal. This allows the procedure to be carried out in a finite number of steps, unlike the Jacobi method, which requires iteration to convergence. Therefore TRED+TQLI firstly reduces the matrix to tridiagonal form and then performs a QL matrix decomposition that consists of a sequence of orthogonal transformations.

As explained before, the initial matrix is the correlation matrix between the observed and synthetic profiles; therefore the SVD computational cost does not depend on the spectral line. However, it depends on the number of free parameters of the Milne-Eddington atmosphere model (NTERM), because the correlation matrix has a size of NTERM rows and NTERM columns. Table 4 gives an estimation of the computational cost of the SVD using both methods. This estimation has been obtained as the average cost of 15,000 SVD executions in a real RTE inversion. It can be seen that the TRED+TQLI method is computationally lighter than the SVDCMP method. Therefore, we take the TRED+TQLI method to carry out the RTE inversion within C-MILOS. In any case, the importance of this mathematical operation has been evidenced.

As mentioned in Section 2.2, the convolution of the synthesized Stokes profiles (*IQUV*) and its response functions with the filter transmission curve, or simply the instrumental profile, is carried out. Regarding the convolution computational cost, it obviously depends on the number of convolutions to perform. A specific study of the computational cost of convolution for our problem was carried out in [Cobos (2010-B)]. Such a cost does not depend on the spectral line either, but it does on the number of wavelength samples for the spectral line, N, and those for the instrumental profile, M. Since both N and M are small in our case, direct convolution in the measurement domain (as opposed to the Fourier domain) is chosen.

Taking into account the expressions in Table 3 detailing every part of the algorithm, the number of operations for the inversion in SO/PHI is calculated in Table 5. We have assumed that the number of inversion iterations is 15. Also, we have assumed 10 samples for the instrumental profile.

The SVD task represents around a third part of the whole algorithm. Moreover, this is a complex algorithm with a lot of branches and loops. These are the reasons why this block is usually a specific block in other problems and why we have decided to design a specific core on the FPGA for performing and accelerating the SVD. This design is presented in Chapter 7.

As mentioned in the Introduction, the engineering requirement for the RTE inversion computation time was 1 minute for the first DPU design and 15 minutes for the second one.

This means that the RTE inverter needs a real performance of almost 27 GFLOPS in the first scenario, or 1.8 GFLOPS in the second one for inverting the 2048x2048 profiles. It is important to remark that this is a real performance requirement –not a theoretical peak–, which is tremendously complex to reach by the current computing devices. In fact, the same inversion problem on ground would need a set of almost 50 CPUs [Borrero (2010)], and a last generation PC (Intel Xeon at 3.4 GHz) needs around 2.5 hours for completing the task [Borrero (2010)].

	Mult	Div	Add,Sub	Sin,Cos	Sqrt	Total	%
SVD	2830	360	5625	0	7	8815	32.09
SYNTHESIS	1258	56	924	4	0	2238	8.15
CHISQR	28	5	52	0	0	85	0.31
MODPAR	171	9	162	0	0	342	1.25
RF	3469	187	3072	0	0	6728	24.49
COVARM	2400	36	2544	0	0	4980	18.13
CONV	2160	0	2120	0	0	4280	15.58
TOTAL/ iteration	12316	653	14499	4	7	27468	100.00
Total Inversion	184740	9795	217485	60	105	384552	

Table 5 Computational cost (number of operations) for the RTE inversion in SO/PHI

2.4 Reliability and robustness of C-MILOS

Since our reference, IDL MILOS code [Orozco (2008)], for the inversion algorithm has already been checked in practice, here we only have to test C-MILOS as compared to it.

To do that, we have generated a reference basis of Milne-Eddington Stokes profiles for the Fe I 617.3 nm line using the IDL code. This set of profiles has been obtained from 10,000 ME model atmospheres with a uniform random distribution of vector magnetic fields (B from 0 to 1500 G, inclination and azimuth from 0 to 180º) and LOS velocities (between -2 and 2 km s^{-1}), and without macroturbulence. The wavelength sampling has been 0.5 *p*m, with a total of 150 samples across the spectral line. We have added noise to the profiles at the level of 10^{-3} I_c.

Figure 12 shows the difference between the results of the inversion carried out with the IDL and the C versions for the magnetic field strength, field inclination and LOS velocity (left column and top to down).

A second test uses the MELANIE code by [Socas-Navarro (2001)] for generating a profile set with the same characteristics of the former. The MELANIE code is an independent ME code which uses a different inversion method.

Figure 12 also shows the difference between the results obtained with both versions for these MELANIE profiles (right column). In this case, the similarity between the C code and

IDL code results are as good as before. These differences are totally negligible and can be attributed to different rounding errors between the two programming languages.

These tests demonstrate that C-MILOS and MILOS are totally equivalent for doing the RTE inversion; therefore we can assume that C-MILOS is as reliable and robust as MILOS as far as synthetic profiles are concerned.

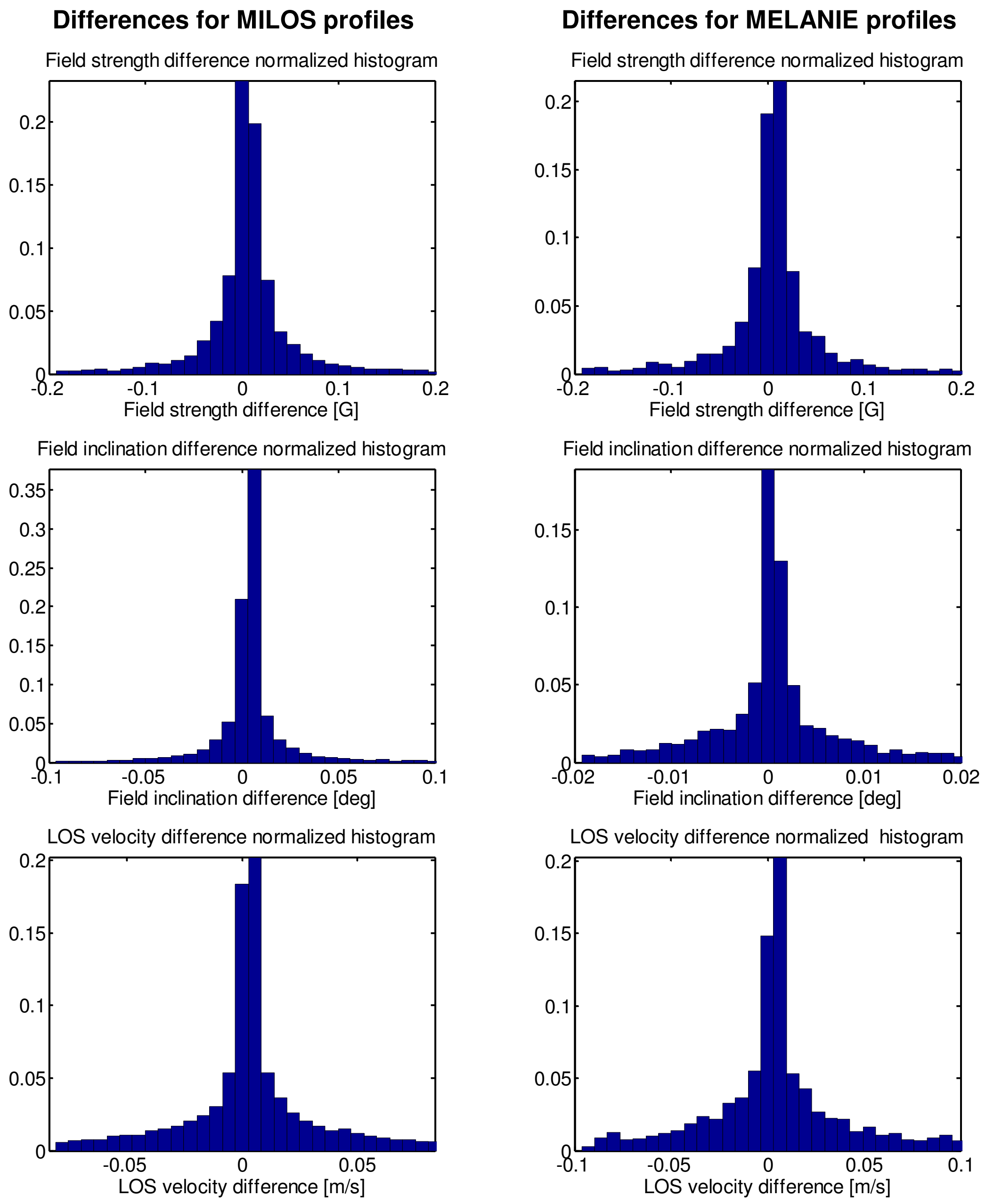


Figure 12 Difference between the inversions carried out with the IDL and the C versions for the magnetic field strength, field inclination and LOS velocity. Stokes profiles with added noise of 10^{-3} I_c generated by MILOS (left column) and by MELANIE (right column).

2.5 Precision of the calculations in C-MILOS

The previous versions of MILOS are developed with floating-point, double-precision calculations. However, C-MILOS can work as well in single precision. As can be seen in Figure 13, working in single precision C-MILOS produce as small errors as with double precision C-MILOS. The Stokes profiles were generated by MELANIE as in the previous section with added noise. The root mean square difference (RMS) between the MILOS and C-MILOS inversions run with MELANIE synthetic Stokes profiles is given in Table 6. Double and single precision results are compared.

In short, in this section the feasibility of carrying out the RTE inversion using single floating point precision has been shown because the achieved differences using the latter are comparable to those achieved using the former. All results together demonstrate the reliability of the C inversion code. Besides, single floating-point precision can now be accepted for the RTE inversion.

Nevertheless, taking into account that the final device that will execute this algorithm is an FPGA, we have studied the possibility of using fixed-point precision. As is common knowledge, fixed-point implementations tend to spend less resources that floating-point precision ones. However, we soon understood that due to the huge amount of operations for executing the RTE algorithm it is necessary to re-use most individual computation units on the FPGA along the entire data path. That implies that each computation unit has to be precise enough for every task it is involved on. Several parts in the algorithm need a huge range in its execution that imposes a big range in the fixed-point format to use.

For instance, in Appendix I we show the precision study that we performed about the Voigt function code. That function is computed for complex numbers, but in the diagram is already shown in a real number decomposition. In this study, we show the necessary fixed-point format along the data path for getting a precision similar to the floating point precision: 10^{-6}. This study clearly denotes the huge range that this function needs. A final decision for working in floating point was made. The reason is twofold: first, fixed-point calculations needed too many bits for the required precision; second, DSP blocks in our FPGA are optimized for implementing floating-point operation cores.

	Double-floating point precision	Single-floating point precision
	RMS	RMS
Field strength (G)	2.6908	3.5537
Inclination (º)	4.2260	4.1999
Velocity (ms^{-1})	0.6152	1.4846

Table 6 RMSE for inversions carried out with double precision and single precision. MELANIE profiles with added noise at 10^{-3} I$_c$.

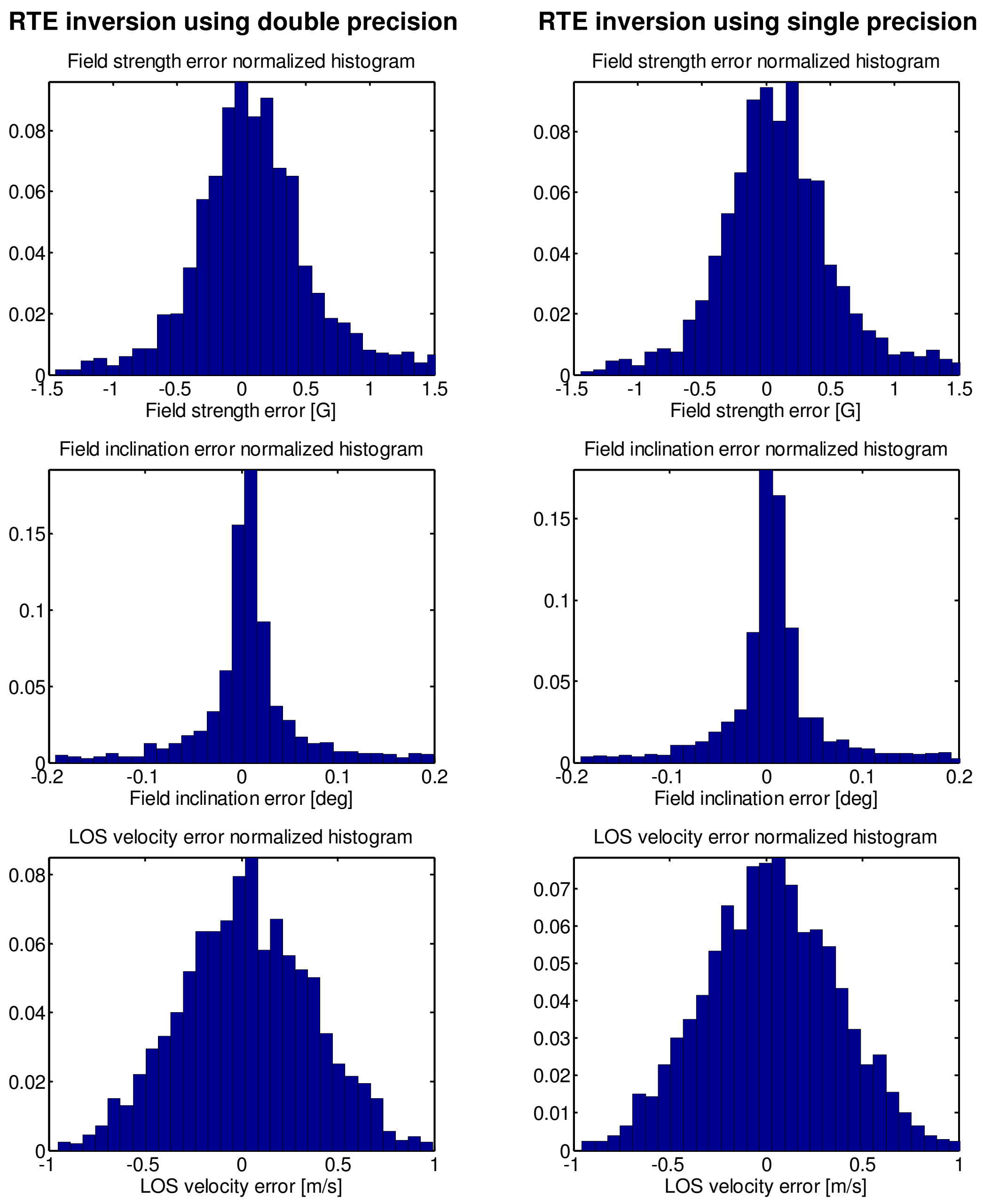


Figure 13 Errors produced by the inversions carried out with the C version for the magnetic field strength, field inclination and LOS velocity. Stokes profiles generated by MELANIE with noise at level $10^{-3}I_c$. Double-precision floating point (left column) and single-precision floating point calculations (right column).

2.6 Improving the RTE inversion code

Taking the initial number of operations as a reference, and without forgetting that the final RTE inverter device is a FPGA, it is necessary to go into details for trying to reduce the number of operations. After studying the RTE inversion code, some modifications have been proposed in order to make its computation easier. Already during the development of the C-MILOS inversion code version, some improvements were introduced. For instance, sharing those operations previously computed in different blocks is enabled. Division is costly in computation. Therefore, as many division operations as possible have been eliminated through refactoring of the mathematical expressions (using the common factor). Also, division by constants was replaced for multiplication by its inverses.

The first modification about using single floating point precision is very important since it means to use roughly half of the hardware resources.

Since the most consuming block in the inversion is the SVD block, then we consider necessary to study existing alternatives for carrying it out on an FPGA. Specific SVD blocks has been proposed by [Brent et al. (1983)] and [Bravo et al. (2006)] in order to accelerate the SVD computation and save as many hardware resources as possible. This theme is dealt deeply in Chapter 7.

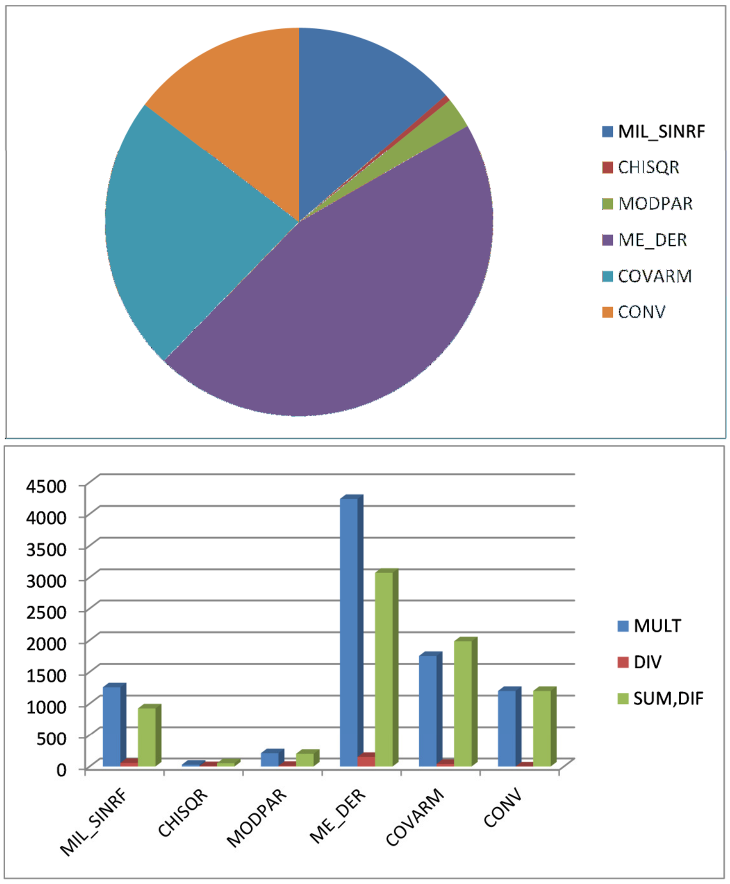


Figure 14 Operation distribution of tasks in the RTE inversion code after the C-MILOS improvements.

The calculation of the correlation matrix between the observed and synthetic profiles is improved by taking into account that is symmetric. Thus, only a triangular matrix has to be evaluated Besides, the SVD block works more efficiently using triangular matrices. The almost 5,000 operations that a full correlation matrix needs to be calculated are reduced to a little more than 3,000 operations.

The convolution computation is made using a direct convolution instead of the Fourier transform because of the small number of wavelength samples.

The final amount of operations, and its distribution in the different tasks, is shown in Figure 14.

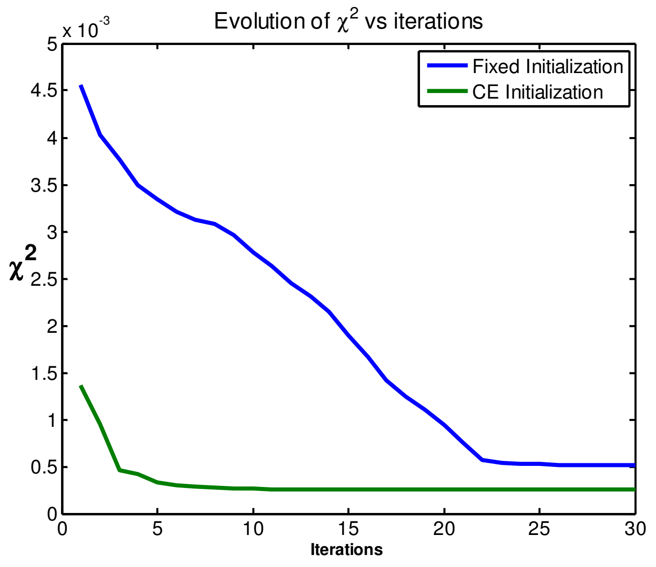


Figure 15 Convergence rate (χ^2 evolution) as a function of the number of iterations

Initialization is always an issue in any inversion procedure. Some initial models may lead to too slow convergence because the guess parameters are too far away to the searched-for ones. In ground applications where no requirements in operation time or in the use of hardware resources apply, this problem may not be as big as for a space application as ours. Since we have to establish a fixed number of iteration cycles, we cannot afford such long searches when departing from wrong guesses. As barely commented in Section 2.2, we have decided to use classical estimations of the free parameters that are obtained from simple calculations using direct operations. A demonstration of the improvement in convergence of the iterative process when using classical estimations for initialization is shown in Figure 15. χ^2 is plotted as a function of the number of iterations. The graph shows the mean value of the obtained χ^2 after inverting a thousand ME profiles with different iterations. The synthetic profiles have been generated with MELANIE. They have later been convoluted with a theoretical instrumental profile (a Gaussian of 10^{-1} pm); noise has been added at a level of 10^{-3} I_c. As can be seen, convergence is much faster and even reaches better final value when classical estimations (green line) are used.

Although this empirical result is very promising, we have to acknowledge that it is only based on synthetic ME profiles. When real, asymmetric Stokes profiles are analyzed, the results could be different. Therefore, as a precautionary measure, we will continue assuming around 12-15 iterations for the RTE inversion. The final decision will be made when more tests with real images are done.

2.7 C-MILOS inversion time

Obviously, since IDL is higher level than C, its computational time is bigger. Besides, C-MILOS has been developed thinking on achieving the minimum computational time. We have chosen the best case of testing with several compiler optimizations like SSE-MMX[3] extensions, different loop-unrolling methods, etc. C-MILOS avoids unnecessary calculations, re-uses some calculations, simplifies, and orders mathematical operations, and uses tricks of low-level programming for improving the computational time. In addition, we do not lose accuracy in the results, as we have seen before.

In Table 7, we can see a comparison of computational time with both code versions. We have inverted a reference basis of 1000 profiles generated with MELANIE in a similar way as in Section 2.5, with 150 wavelength samples.

In both cases, we have used the same initialization and the same 'stop conditions'. In this test, the classical estimate initialization is disabled in C-MILOS. This proves that C-MILOS is faster than IDL-MILOS more than around 16 times. The used CPU is a Pentium 4 running at 3.4GHz.

		IDL-MILOS	C-MILOS
Double precision	1000 profiles	329.76 s	20.41 s
	1000 profiles with noise	277.13 s	18.30 s
Single precision	1000 profiles	-	15.85 s
	1000 profiles with noise	-	15.74 s

Table 7 Computational times for inverting RTE using IDL-MILOS and C-MILOS

[3] The Streaming SIMD Extensions (SSE) was introduced in the original SIMD multimedia extension instruction set (MMX) by Intel in 1997.

3

High performance scientific computing on FPGA

As defined in [Sravanthi (2014)]:
"High Performance Computing (HPC) most generally refers to the practice of aggregating computing power in a way that delivers much higher performance than one could get out of a typical desktop computer or workstation in order to solve large problems in science, engineering, or business"

3.1 Introduction

The RTE inversion has to be carried out aboard the space-borne SO/PHI instrument. The corresponding mathematical algorithm is an iterative, non-linear, least-squares process. In this work, we start from a C-programed version of the code, where more than 17,000 operations per iteration and around 15 iterations per inversion for each of 2,048 x 2,048 spatial pixels are necessary. In addition, the singular value decomposition of a covariance matrix is included in the algorithm, which increases the high computational demand still further. The required operation accuracy, the data range involved, and the iterative method for convergence make us to work in simple precision floating point.

No qualified for space processor, or DSPs, are available that fulfills the stringent computational requirements with the limited room and power consumption allocated within the instrument. As explained in [Fiethe (2012)], the only alternative is to use a space-qualified FPGA and develop a tailored design for it. Along the course of the SO/PHI project, two versions of DPU have been proposed. The first one used the Virtex-5 FPGA for a real-time RTE inversion. The ideal scientific goal was to invert the 2^{22} profiles in a minute, the typical shortest time foreseen for recording the data. The second DPU employed a Virtex-4 XQR4VSX55 and was accepted by ESA. In this second version, massive flash memory was included that is able of storing up to 9 hours of observations. That permits the RTE inversion to take 15 minutes for the process. Thus, the computing power that those FPGA devices have to reach is 27 GFLOPS or 1.8 GFLOPS, respectively.

Bearing in mind that the RTE inversion is currently carried out using cluster of PCs because of the required computing power, we can advance that our RTE inverter has to be a High-Performance Computing system. As such, HPC is the field to propose valid solutions to the inversion problem aboard SO/PHI. Specifically, HPC embedded systems on FPGA will be taken as reference in our proposals.

Such demanding computing tasks are not usually performed aboard space instruments. Therefore, there is no useful reference. However, we can find several works where FPGAs are used for accelerating calculus in ground systems [Altera (2007)]. FPGAs are usually employed as custom machines or as specific-domain processors. In the current multi-core era [Borkar (2010)], the trend is to use several computation cores. A similar perspective is also applied in the embedded computing realm, where several works attempt to use multi-core architectures in FPGAs.

In this chapter we discuss the possibility of using a customized design in FPGA for the RTE inversion. We start by giving an outline of the state of the art in HPC for settling the background upon which our architectures have been devised. We also glance at some works about multi-processors on FPGA that explain, and condition, several design decisions we made in this dissertation.

3.2 Custom architectures on FPGA

FPGAs have become very good devices in recent years for improving computing performance and, overall, for getting power-efficient computing. Good examples of practical experience with FPGA-based co-processors are shown in [Altera (2007)]. Compared with usual processors, as Pentium 4 or Opteron with clock frequencies of several GHz, acceleration factors from 10 (Monte Carlo simulations) up to 370 (Inverse Hough processing) are reached.

The complexity of the RTE algorithm drives our design to a great extent A custom architecture would imply re-using most individual computation units on the FPGA along the entire data path, since the RTE inversion is a complex iterative algorithm. To illustrate the large amount of operations in the RTE inversion, Appendix I shows how the Voigt function (see Sections 2.3 and 2.5) would be extremely unfolded in a custom architecture. Therefore, a control unit for a large custom architecture would be as complex, or even more, as a control unit for usual processors. On the other hand, the RTE algorithm has many configurable elements. Among them we can find the number of free parameters in the iterative least-squares process; the number of wavelengths in the input; specific spectral line parameters; and other scientific approximations whose treatment in depth is out of the scope of this thesis. Let us mention, however, that the Zeeman pattern of the spectral can increase significantly the number of final operations [del Toro (2003)].

At the beginning of studying our problem, we attempted to design a custom architecture for the RTE inverter. We used a High-Level Synthesis language (HLS) -System Generator [Xilinx]- for rapid prototyping, but the first results were discouraging since the high number of operations could not be included on the FPGA. As an illustration, we show the top scheme for the Spectral Synthesis block generated in System Generator in Appendix II. This block has to be executed as many times as the number of wavelengths. In this scheme, inside the *fcomponent* block, is computed the Voigt function using the implementation shown in Appendix I.

Another example of the difficulties in customizing our algorithm can be found in the Classical Estimates block (see Figure 11). This block was also implemented by [Torné (2012)] using System Generator. Only that implementation occupied around a 9% of Flip-Flop slices and an 18% of LUT slices of a Virtex-5 FX130T FPGA (commercial version of the radiation-hardened one). As commented in Section 1.5, this FPGA logic cells are in equivalence around 4 times larger than those in the Virtex-4.

These are nothing but two small examples of how the extremely high computing demanding algorithm hampers the development of a custom architecture. It would simply be too expensive.

For all these reasons we propose a computing architecture based in a multi-core approach rather than a custom one. This design, apart from allowing a simplified control unit, enables the dynamical introduction of modifications in the RTE inversion code. In this way, the development of the RTE inversion architecture can be carried out at the same time that the final scientific algorithm is fully defined, or validated. In addition, the RTE inversion architecture could easily be adapted to other future instruments with particular necessities, or even for other applications. In fact, the multicore architecture proposed in Chapter 5 is now being arranged for compressing images on FPGA [Hernandez (2015)].

3.3 Parallel computing using multi-processor architectures

To find parallelism in a single instruction stream is based on improving the Instruction-Level Parallelism (ILP) in order to reach high performance in a processor. During years, Moore's Law made it possible to increase the Instruction-Level Parallelism in the pipelined, von Neumann processor simply by increasing its frequency. Costs reasons and technical issues made that this progress did not continue, because the practical power limit was reached (high power consumptions and notable heat dissipation). In addition, the memory technology had not followed the same processor frequency growth. Thus, latencies between processors and memories are higher and higher [Patterson (2000)].

Processors were provided with very complex mechanisms for dynamically discovering and exploit ILP like re-order buffers, branch prediction, superscalar pipelines, etc. Static approaches were also developed where the task of exploiting parallelism relies on the compiler, although these are the minority of cases. In addition, the multithread execution was introduced for improving the processor efficiency, and not wasting empty processor cycles.

To reach HPC, supercomputers have traditionally been organized in architectures with several processors working together: clusters, Massively Parallel Processing (MPP), Symmetric multiprocessing (SMP), etc. These arrangements are characterized by the type of network among processors, memory distribution, etc., but they are out of the scope of this thesis. We only want to remark that, taking the list of 500 best supercomputers in the world that TOP500 publishes [TOP500] as a reference, we have to go back to 1996 for finding an single-processor supercomputer architecture in that list. Today, the majority of them are based on clusters or MPPs.

However, the definitive solution for continuing increasing the performance of one processor was the use of several cores in a chip, or multi-core architectures. The birth of this new technology was decisive. In fact, the first supercomputers with more than a processor per chip appeared in the TOP500 list in 2002. Already in 2007, the 87% of supercomputers had more than one processor per chip. Today all supercomputers use multi-core processors.

In short, multicore architectures provide a good tool for parallelizing algorithms and have been used for long in computer engineering [mimd8, mimd9]. They have now become the most useful tool for improving modern supercomputers [Elliot (1988)].

Thus, multi-core architectures were chosen for carrying out the RTE inversion. Along this dissertation, we explain how they can be used. The difficulties they present when an algorithm has to be executed in them are also pointed out, since the software crisis [Dijkstra (1972)] highlighted the difficulties of getting all the benefit that multicore system can provide. This means that we should also pay attention to the RTE inversion software development for achieving HPC.

Sidelining the difficulties of getting optimal programs, Amdahl’s law remembers us that the reachable speedups are limited by the non-parallelizable piece of the algorithm [Amdahl (1967)], and this will be taken into account in our proposals.

When implementing a time-critical algorithm in a multi-core system, it is essential to take advantage of the intrinsic parallelism of the algorithm [Amdahl (1967)]. The Flynn's taxonomy [Flyn (1966)] shows that multi-core can be used to achieve data and functional parallelism of the algorithm to run. The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) streams and data streams available in the architecture. There are four possibilities that we outline below, and some examples for each type are given. A more extended description can be found in [Patterson (2000)].

- SISD: Single Instruction stream, Single Data stream. The standard von Neumann model.
- SIMD: Single Instruction stream, Multiple Data stream. The original systems from Thinking Machines or MasPar, and GPGPUs.
- MISD: Multiple Instruction stream, Single Data stream: No well-known systems fit this designation.
- MIMD: Multiple Instruction stream, Multiple Data stream: MPPs, workstation clusters, shared-memory SMPs.

Thus, the parallel multi-core systems are roughly divided in SIMD and MIMD parallel computers.

Firstly, a MIMD parallel architecture is a set of processors that execute multiple instruction sequences (MI) that taken altogether constitute a single stream of instructions. Each individual process corresponds to a single instruction sequence that is applied to a given set of data on an individual computer. Therefore, there are multiple streams of data (MD) that are processed at the same time. Hence, to exploit the data parallelism of a given algorithm, the MIMD architecture should be fed with instruction sequences that belong to this algorithm. As mentioned above, this is the current trend in supercomputers, where multiple multi-core processors that work on different parts of the program are coordinated. Each processor uses its own operating system and memory. Typically, processors communicate using some messaging interface.

On the other hand, the SIMD architectures (Single instruction stream, multiple data streams) were proposed in the dawn of the computer era aimed at exploiting the data-level parallelism. The underlying idea is to apply the same operations to multiple items of data in parallel.

Taking again the TOP500 list as a good reference for understanding the state of the art in computer architecture, we can see that the last supercomputer with SIMD architecture was in the list in 1997. However, last years these architectures are still very alive, as General Purpose Graphics Processing Units (GPGPU, or simply GPU) have demonstrated. Now these devices are used for accelerating specific high-parallelizable problems in a SIMD way. Specifically, this fact marks the new computation trend that has been called heterogeneous computing, where different architectures work together for improving the computing capabilities of a system. Usually, these architectures are placed as co-processors. Again, the TOP500 list reflects this fact. In its last issue of June of 2015, 20% of supercomputers use co-processors based on GPU architectures, while in 2007 there was no one. It is remarkable that these new incorporations are directly to the high part of the list. In fact, the first position in June of 2015 is co-processor accelerated. Intel and Nvidia provide most of GPU-like accelerators, with Intel Xeon Phi [Intel] or Kepler [Nvidia] as the more indicative families respectively.

GPUs supply such high performance due to its novel model of execution based on a fusion between the SIMD models and multithread execution models. GPUs are composed of hundreds of computation cores with cooperative sharing control units (instruction fetch, decode, and memory manage units) and with a huge number of threads, contrary to multithread processors which replicate all the hardware for each thread. This model has been called SIMT (Single Instruction stream, Multiple Threads), where threads are executed in a SIMD way for an instruction stream, but the flow control is maintained for each thread.

A high ILP is reached through an intensive switching between threads for avoiding empty cycles in the computation cores. Threads are organized in groups, or warps in the Nvidia nomenclature. All threads in a warp execute together using a common program counter, and sharing the control units, and all of them inside a warp are conditioned by their conditional branches.

To conclude this section, we remark that we have taken well established, state-of-the-art, computing models like MIMD and SIMD in this thesis. Besides, we have applied novel ideas on them for getting their computing capabilities enhanced. In the next two chapters we present in depth the designed solutions for dealing with the RTE inversion from both approaches, MIMD and SIMD. Along each architecture description, multiple works will be referred to either as an inspiration or as a reference. In Chapter 4, we propose an MIMD execution scheme where we have introduced a novel operation distribution in a full-fine-grain way and where we have augmented extremely the ILP. As mentioned above, the enormous data-parallelism in our RTE inversion problem makes the SIMD model very suitable to deal with it. In Chapter 5 we propose a SIMD-flavor architecture, which is not a classical SIMD vector unit but follows the SIMT fashion of the GPUs. Our proposal presents several novel ideas that modify this model for adapting it to the RTE problem.

3.4 Multi-processors architecture in FPGAs

There are several works focused in parallel multi-core architectures for embedded systems in FPGAs [Baklouti (2010); Stanley (2003); Tong (2006); [25] Yiannacouras (2012)]. However, none is suitable for the RTE inversion either because they only use fixed point precision, or because they employ commercial processors which are not able of tackling the RTE inversion.

We have already argued that the use of floating point precision is necessary to our purposes. Hence, we need to gauge the use of existing processors with floating point capabilities. In [Learn (2011)], the floating-point computing performance of the Leon 3, the Power PC 440, and the MicroBlaze embedded processors on a Virtex-5 FX-70T FPGA are studied. Using the Whetstone benchmarks [Randell (1964)], they concluded that the Leon 3 has the best floating-point performance of all of them.

According to that study, the Leon 3 processor is able of executing around 57 WMIPS (Whetstone Million Instruction Per Second). The NIOS II processor can overtake the Leon 3 processor in floating point computation performance (see [Tong (2006)]). On the other hand, we know that a usual desktop processor, of the Intel Xeon family, needs more the two hours and half for doing the RTE inversion problem. This processor usually reaches, or even exceeds, the mark of 1000 WMIPS.

Hence, a network of commercial embedded processors as LEON, NIOS, ARM or Power PC cannot be used, because several tens of them would be needed to carry out the RTE inversion in the less stringent scenario, 15 minutes. Such number of processors cannot be allocated in the proposed FPGA device. As is shown in [Learn (2011); Tong (2006)], less than a dozen could be allocated in the qualified Virtex-5 device and even less in the Virtex-4. In any case, none of those processors reach its theoretical computation peak

Since we are looking for a design with a maximum productivity of its floating-point units and with a minimum use of resources and power, we conclude that the existing processors cannot be used in FPGA for SO/PHI. Thus, we have to design new processor architectures for adapting the parallel computing paradigms to the RTE inversion problem. FPGAs offer a lot of possibilities for reaching these objectives. For instance, floating-point capabilities are an excellent feature of the Virtex family FPGA ([floating point FPGA]). FPGAs also allow customizing the processors architectures to include only the necessary instructions that an specific algorithm uses.

4

A MIMD architecture

A MIMD parallel computing architecture is proposed in this chapter. It is aimed at being implemented in a space-qualified Virtex-5 FPGA, the first DPU design considered, as commented in Section 1.5. The main objective of this version is the real-time processing of the RTE inversion, that is, 2^{22} profiles in a minute.

Since the SVD has been proposed to be implemented in an independent core, in this chapter we only address the calculation of the main block Spectral Synthesis and Response Functions, abbreviated SSRF for simplicity.

4.1 The multicore MIMD architecture

MIMD multi-computers are an excellent option for being implemented on FPGA due to the fine-parallelism grain that they provide. Specifically, a distributed memory MIMD system is proposed as shown in Figure 16. Each processor (pPi) executes a subset of instructions of the original algorithm. Therefore each one has its own control unit and operates with different data (which come through its corresponding Memoryi). Besides, they can be versatile since the SIMD architecture (single instruction stream, multiple data) can be virtualized in a MIMD architecture.

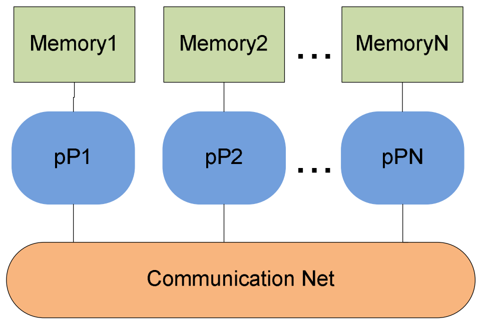


Figure 16 Distributed memory MIMD multiprocessor

Each simple processor is called a picoprocessor (pProcessor) (see Figure 17). The pProcessor calculation core is a floating-point operation core of a single type. This minimizes the latency penalty associated with the operation to be performed. To supply each calculation unit with its two operands, the data memory is unfolded in two parts. Therefore, both operands can be obtained at the same time. The motivation for distributing the memory among pProcessors is to minimize the access latency, placing the data needed to operate in each pProcessor.

In a multi-core system, where different processors cooperate in the execution of an application and where each memory is not accessible by other processors, it is necessary to establish a data communication network. In the proposed architecture the data produced in a pProcessor are directly sent to other pProcessors where they are used. This procedure is a simplified, passing message method [Patterson (2000)]. Data are sent through the output ports and through the specific links for each pair of output/input ports. Note that an output port of a pProcessor can be connected in a loop to an input port of the same pProcessor. The instruction memory (Instructions ROM in Figure 17) defines the data used by each pProcessor. Within the instruction memory there is a fixed subset of operations properly assigned, as discussed below.

The distribution of operations in different pProcessors highlights the need for cooperating among them. In fact, in order to solve the problem, pProcessors must communicate (pass on data to each other) and must be synchronized (wait for data from the others before continuing with the execution). These tasks are performed through a specific communications network, where the pProcessors only have direct communication with others if they need to exchange results. The pProcessor input and output ports are the links to other pProcessors, and the whole link set establishes the communication network. Moreover, to achieve the dataflow that the algorithm dictates, a communication network between processors must be established.

In order to allow easy reading of this thesis, tedious details about instruction codification have not been introduced here and can be found in Appendix III.

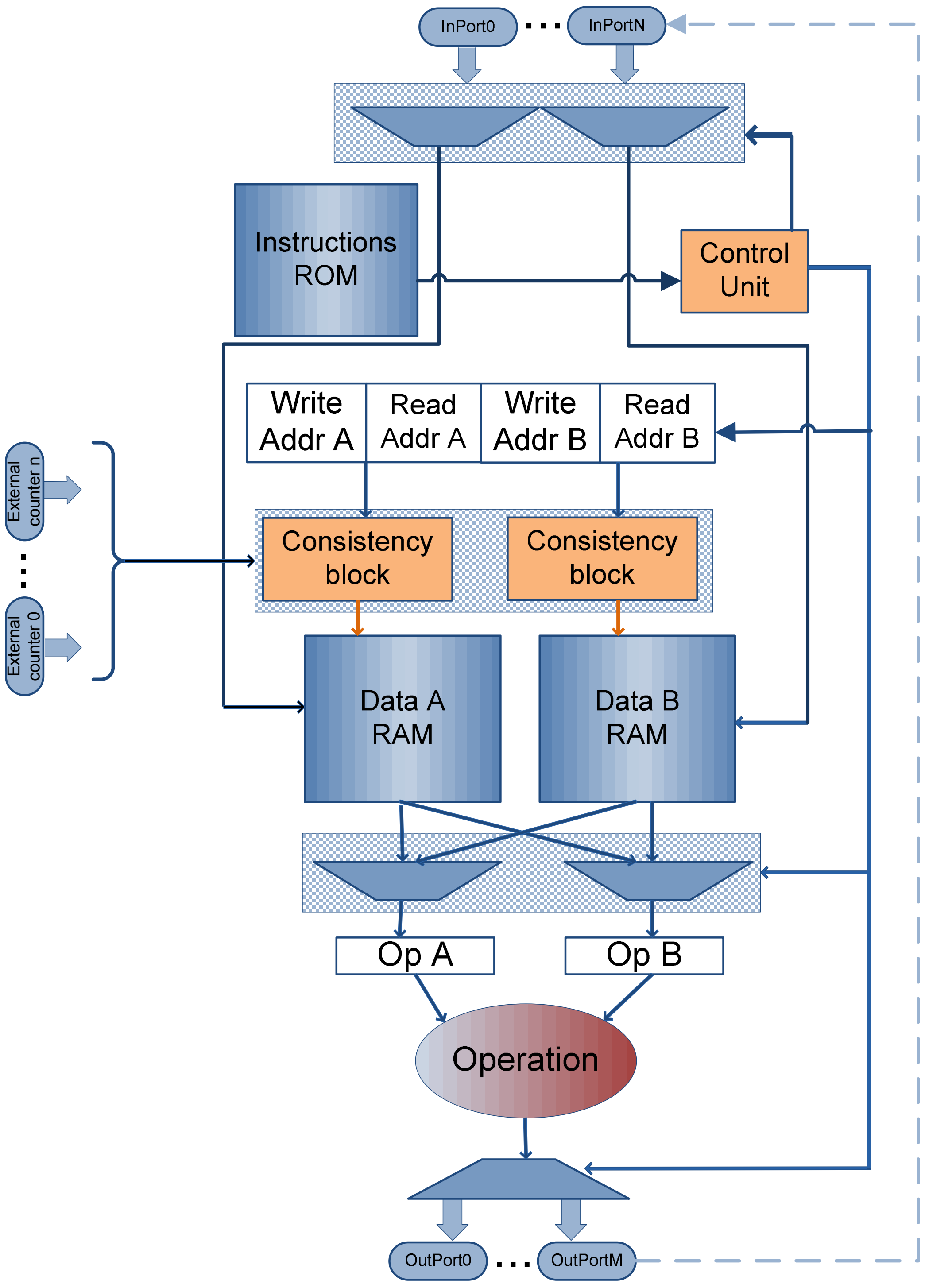


Figure 17 pProcessor's architecture

4.2 A novel MIMD programming model

Usual programming techniques for MIMD architectures attempt to determine the coarse-grained –data or instruction– parallelism and to distribute the algorithm tasks in a set of processors. This process can be done manually or assisted by some libraries [Zima (1988)]. Besides, the pieces of code in each processor can exploit the Instruction-Level Parallelism (ILP), thanks to pipelined execution, out-of-order scheduling, etc.

Some programming techniques, as loop-unrolling and software pipelining [Allan (1995); Lam (1988)], have become popular because of their performance improving, so they are very used in modern compilers. These techniques take advantage of the ILP for creating parallelization opportunities thanks to the out-of-order and speculative execution of instructions in a processor [Lam (1988)]. Beyond of execution scheduling, these techniques have been used in order to extract parallelism [Ottoni (2005)].

Consider a scientific problem, as the one addressed in this work, with a large data parallelism at algorithm level. That is, the same operations are executed for different input data. In our case, the RTE inversion algorithm is executed for each spatial pixel (2^{22} times in total). For this kind of problems we propose software pipelining and loop-unrolling techniques at the algorithm level. Traditional software pipelining searches a processing kernel [Allan (1995)] and generates a processing graph. We, however, propose a new software pipelining method. Our proposal is actually a hardware-like software pipelining by doing an intensive memory use.

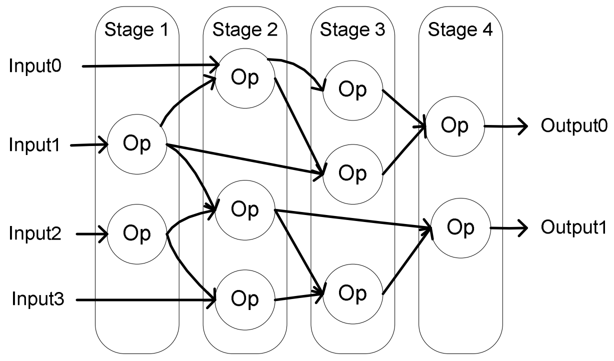


Figure 18 Example of an operation tree

One of the basic ideas in the hardware pipelining is to exploit the implicit parallelism in the algorithm. In Figure 18, a tree of operations for an illustration algorithm can be seen. If a suitable number of computing elements is used, all the operations in each stage can be executed in parallel, hence reducing the total execution time. We define a stage as the set of instructions that can be executed at the same time because of its data dependence. Therefore, the virtual X axis corresponds to a discrete time axis in which each instant corresponds to a given stage.

Anyway, in a hardware pipelined model all the stages are executed at the same moment. We will call that moment a synchronous stage.

In addition, if the algorithm execution is within an iterative process and the number of times to execute the algorithm is considerable, it is possible to add more computing elements to pipeline the system. This improves the output ratio of results, since all the stages are executed at the same time. The improvement depends on the number of times to execute the algorithm. Pipeline techniques imply an increase in the memory needed to store temporary results, as well as to synchronize the data flow –generally, through FIFO memories. By doing so, and assuming the ideal case of having the required number of computing elements (as many elements as operations), the system with minimum execution time would be achieved.

The number of operations in our SSRF algorithm is larger than the number of computing elements. Therefore, re-use of the same computing element is necessary to perform various operations –gather operations in processors. In any case, achieving the system with minimum execution time is not our target, but satisfying a time requirement: we must invert 2^{22} profiles in a minute.

Knowing the usual hardware pipelining method, the key idea is to distribute the algorithm instructions set to the set of processors starting from the hardware pipelining operation tree and including the necessary FIFO memories.

The result is a set of totally decoupled instructions in each processor, where they can be executed out of order [Dwyer (1992)] in a synchronous stage, opposite to a scheduled execution with usual software pipelining method. Like happens with a hardware pipelining method, our system has a latency time for starting to generate correct results depending on the number of algorithm stages.

In short, to exploit the parallelism of the algorithm, a MIMD architecture is used where several arithmetic operations of the same type are clustered in each pProcessor. We also carry out a hardware-like pipelined execution of the algorithm –all stages are executed simultaneously in a synchronous way. The intensive-pipelined execution of the algorithm conditions and simplifies the synchronization in the MIMD architecture. In the pipeline process, buffers are introduced that allow pProcessors not to have to wait for others before executing their instructions, since they have the data they need. It is desirable that all pProcessors start and end their instruction execution in a known time interval. This is achieved by introducing global synchronization.

Another implication of assigning a set of operations to a particular pProcessor is the need to set up its memory space. As indicated in Figure 17, the memory space is split into two in order to be able to obtain two operands at the same time. Therefore each pair of data involved in an operation has to be assigned to separate memories. Each data occupy several locations in memory due to the intensive-pipelined execution; when the data go directly from one stage to another they occupy two positions and more when it is necessary to keep the data for more stages. The memory consistency [Hennessy (1999)] is guaranteed thanks to the pipelined execution and introducing a global synchronization signal. This means that all pProcessors see the same read and write order in positions of data that they share. Thus it is not necessary to use a message passing communication, which is used in other multi-computer architectures where shared memory is used [Wilkinson (1999)].

Conceptually, memory consistency is only an emulation of FIFO memories that guarantees each data will be stored or read in the correct position and will wait for its turn. It is a software implementation of the traditional hardware pipeline. These FIFO memories are introduced when pipelining any system to transfer data between stages. But this apparent simplicity hides a problem. As noted in Section 4.1, each pProcessor encodes the instructions in a ROM, and therefore the position of each operand is fixed at given time. The "memory consistency" blocks are introduced to set the positions of the operands and to achieve a FIFO access type at runtime. Also, an external counter is added to the basic position obtained from the ROM. These global external counters are increased every synchronization period and there is a counter for every possible "value of consistency" (FIFO length). The counter to be used for every position is coded in the instruction ROM. It has to be clarified that the memory consistency model is applied when reading and writing data in memory. This model of counters has been developed in opposition to using a module operation in each pProcessor to a universal counter, since the module operation is very high resources demanding. Appendix III details how the "memory consistency" block works at a bit level.

Another key aspect of this design is to achieve the maximum performance of the calculation units. This is reached through a balanced allocation of the operations into pProcessors. Note that the allocation does not necessarily match with the operation distribution in stages (any operation can be allocated to any pProcessor). One could pretend to drive all the operations of a stage to a single pProcessor, but an unbalanced distribution can very likely occur. The proper distribution of the instructions is a key question to minimize resource consumption, as this action has a direct impact to the communication network. Tests of operation distribution starting from the operation tree have been done using complicated heuristics, which allows detecting a similarity with the typical traveling salesman problem. This problem does not have an optimal solution. Another solution has been proposed instead: to visit every operation in order of appearance and stage by stage. Then, each operation is allocated in a pProcessor of the correct type until the latter is filled up. Apparently this is a random procedure but an acceptable communication network is achieved, thanks to the principles of temporal and spatial locality.

As a consequence, the communications between pProcessors will be conditioned by this allocation since each pProcessor will need many input and output ports as its instruction set needs. The pProcessor design has a limitation in the amount of data that can be accepted each cycle at input ports. This limitation is given by the technological impossibility of writing the RAM more than once in a cycle. The architecture design, the proposed synchronization method, the achieved out-of-order execution, and the implementation of the memory consistency model allow us to freely order the execution of the operations assigned to a pProcessor, since we have relieved the data dependencies between stages. Acting on the execution order of all instructions in all pProcessors, we can find situations where collisions disappear at pProcessor input ports.

Following the proposed MIMD design guidelines and programming method we can obtain the optimal multi-core architecture that satisfies the temporal constraint. Furthermore, the system scalability supports to sidestep the Amdahl's law achieving substantial speedups higher than the theoretical ones. Our scalability is only limited by the technology used and its available resources. For instance, the Virtex-5 FPGA resources limit the number of pProcessors and the communication net.

All the tasks of architecture configuration and compilation of the input code are managed by a specific tool, called TAPAS, which will be presented in the sixth chapter.

4.3 The RTE algorithm in the MIMD architecture

In this Section we explain the way to implement a pipelined version of the SSRF algorithm using the MIMD architecture. The algorithm is an iterative process (typically 15 iterations) to be performed 2^{22} times a minute, as described in Section 2. Therefore, the pipelined system should give a result every 0.95 microseconds. Assuming a system frequency of 200 MHz, it is necessary to have a result every 190 clock cycles. That is, the minimum number of pProcessors will be the one that allows executing all operations of the algorithm in 190 cycles. By extension, to execute all the operations means to perform all stages. In this calculation, we omit the initial latency.

The TAPAS software tool generates a tree of operations from the input code with almost 9,500 operations (remember that only SSRF are considered in this section). In this case, the synchronization signal period is of 190 cycles. The first step is to establish how many pProcessors are needed.

Table 8 shows the size of the instruction set for each iteration of the SSRF algorithm. Each pProcessor has 190 cycles to execute its instruction set. Taking into account the initial latency of the floating-point core and the time needed to communicate the last result, the pProcessor net will consist of 20 pProcessors for addition and subtraction, 32 for multiplication, and 1 for division. Each type of pProcessor contains a different number of operations due to the latency.

Also, there are special pProcessors for input/output tasks and other that perform trigonometric calculations. In average, four input ports, four exit ports and 5 counters are needed by each pProcessor. About 30 counters are necessary in total.

	+ or -	*****	**÷**
Operations	3574	5712	117
Latency + Communic.	10	9	31
Operations / pP	190 - 10 = 180	190 - 9 = 181	190 - 31 = 159
Number of pP	3574/180 < **20**	5712/181 < **32**	117/159 < **1**

Table 8 Number of operations and number of necessary pProcessor

4.4 MIMD architecture in FPGA

We have shown that it is possible to obtain an MIMD architecture starting with the SSRF pseudo-code. This MIMD architecture consists of 53 pProcessors. Thanks to it, each pProcessor is fully configured: memory space (with consistency), instruction order, instruction ROM that contains the operations to execute, and all control signals to apply.

As discussed previously, the proposed MIMD architecture is implemented in an FPGA, specifically in the Virtex XC5VFX130T for this test, which is a commercial version of the radiation-hardened Virtex-5 for space. Each basic unit of the system, i.e., each pProcessor, makes use of the embedded DSP48E computing units [Learn (2011)] for its assigned floating-point operation. The number of DSP48Es to use depends on the type of operation. In addition, both the data RAM and the instruction ROM use Block RAM resources. The MIMD architecture presented here and implemented in the mentioned FPGA is capable of operating at 200 MHz with a number of pProcessors close to one hundred.

Table 9 shows the resources consumed by the architecture in a Xilinx Virtex XC5VFX130T FPGA. The system is RTL designed in VHDL and was compiled by the Xilinx

ISE 14.1 tool. Computational cores of addition, subtraction and multiplication (in 32-bit floating point, IEEE 754), are generated by Xilinx Core Generator tool [Xilinx]. They use the embedded DSP48E, which speeds up operations with minimal consumption of resources and power. More information about resources occupation can be found in Appendix III.

Running at 200 MHz with the proposed architecture it is possible to obtain a result every 0.95 microseconds, leaving out the initial latency that is lower than 50 microseconds. Thus, the time requirement for calculating the synthesis and spectral response functions for the 15 iterations of the 2^{22} inversions of Radiative Transfer Equation in a minute is reached. In the Results chapter we will detail the performed tests and these results.

Resource	Occupation	%
Occupied Slices	9,639	47%
Slices LUTs	21,745	26%
Slices FFs	28,399	34%
BRAM (36kb)	106	36%
DSP48E	136	43%
Maximum Frequency	201 MHz	
Power Consumption	5.17 W	

Table 9 Virtex-5 resource occupation

Regarding power consumption, all of results achieve the power consumption requirements since only 5.17 watts are needed (the initial requirements were 7 W, RTE-R-0030 in Table 2). The power consumption as a function of the pProcessor frequency is detailed in Table 10. The consumptions are split into quiescent and dynamic power. A graphical representation of these consumptions is shown in Figure 19. It can be seen how the total power consumption is linear with the pProcessor frequency. Thus, if adjusting the power consumption were necessary, it could be done by adjusting the clock frequency. All these simulations about power consumption are done with the Xilinx X-Power Simulator.

Frequency	Dynamic	Quiescent Power (W)	Total Power (W)
80	1.15	2.31	3.46
100	1.6	2.32	3.92
150	2.17	2.38	4.55
200	2.73	2.44	5.17

Table 10 Details about the power consumption of the MIMD architecture with respect to the frequency

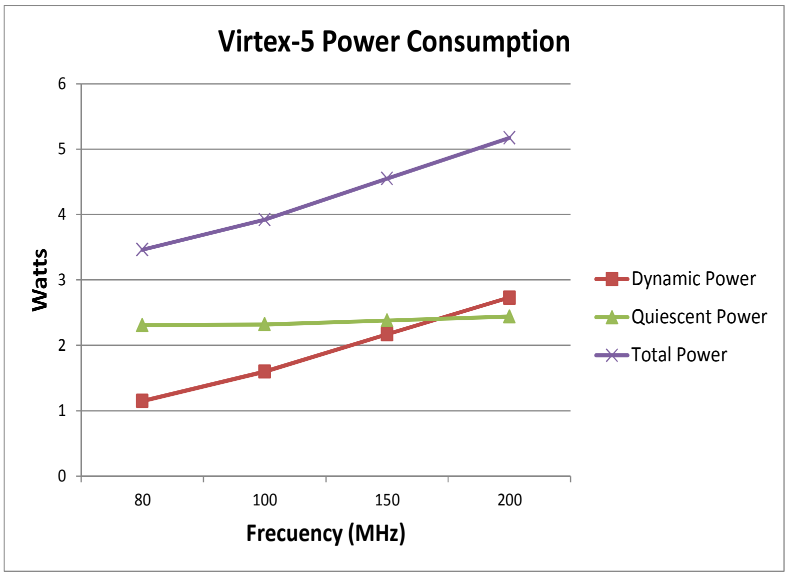


Figure 19 Representation of the power consumption of the MIMD architecture with respect to the frequency

5

A SIMD architecture

The change in the DPU design and the mandatory use of the Virtex-4 FPGA, made it impossible to use the MIMD architecture. This is mainly because there are not enough logic elements in the Virtex-4 FPGA for allocating the necessary pProcessors. Remember that the Virtex-4 FPGA is around four times smaller than the Virtex-5 with respect to the configurable logic block. Besides, and as shown in Table 9, the MIMD architecture occupation is already around 33% of the Virtex-5 device. In addition, the SVD block is to be added yet to that occupation.

Hence, another multi-processor architecture is proposed for dealing with the RTE inversion on the Virtex-4 FPGA. Since this is the final architecture to be embedded in the instrument, its design state is more advanced than the MIMD architecture; in fact, it has been integrated and tested within the DPU as shown in Chapter 9.

The RTE inversion problem can be classified as an embarrassingly parallel problem due to the enormous data parallelism [Moler (1986)]. Our algorithm can indeed be executed in parallel for each of the 2048x2048 spatial pixels using different processors. These problems are efficiently dealt with using SIMD architectures where multiple processors work almost independently. However, no commercial embedded processors can be used, as pointed in Chapter 3. The scientific requirements for accuracy demand several calculations to be carried out in floating point and these processors do not provide the necessary performance. In other words, we would need many more such commercial processors than those physically feasible to allocate within the FPGA.

In the SIMD on-chip design realm there are several works [Baklouti (2010), Stanley (2003), Stanley (2003), Yiannacouras (2012)] that present a classic approach: they take advantage from intrinsic data parallelism into the algorithm in a vector level. These works remind the original supercomputers ILLIAC IV and CRAY flavor or the MMX and SSE extensions in contemporary processors. They are interesting since they present novel communication networks between processing nodes that enhance the computing performance. Nevertheless, in the RTE inversion problem the processors do not need to exchange any data between them. In this work we present a distributed-memory SIMD architecture which is optimized for taking advantage from data parallelism where a communication network is not necessary because there are no dependencies among input data. Thus, the architecture is totally focused in obtaining the best instruction-per-cycle rate as possible.

Along this chapter, we will describe how this architecture design is inspired on SIMD and GPU architectures, and how it tailors some aspects to improve the performance for embarrassingly parallel problems. The architecture is not a classical vector unit but follows the SIMT fashion of the GPUs. Despite processors have been released from a decode unit, each of them has the control over the flow of its threads. However, there is only a general program counter and this means that only a *warp* (see section 3.3) is in progress at a given time.

The processors are tailored into the architecture for reaching a high rate of executed instructions, trying to execute one instruction per clock cycle, or in other words, for getting a high ILP. In this way, the lack of several warps is compensated. The memory address space of every processor is much reduced. It works as if it was a cache and statically scheduled by the compiler. In short, to save hardware resources the usual GPU architecture has been modified for improving performance. Other important contributions of this work are the ability of saving resources allocating operation cores in a shared operation block which is accessed by every processor. This is simply the introduction of the new computation trend, heterogeneous computing, to the embedded in GPU-like architectures.

In next subsections we present the SIMD proposal, detail the different blocks that compose the architecture, and how the RTE inversion is implemented. Relevant contributions of this architecture, regarding scalability or reconfigurability, are also discussed in this chapter.

5.1 The SIMD architecture

The SIMD architecture for carrying out the RTE inversion is presented in Figure 20. Two main blocks can be distinguished, namely, the communications block and the RTE inversion core.

The first one is obviously in charge of the communications with the instrument's DPU. It is based on a proprietary System-on-Chip protocol and uses a single communication channel with the rest of the instrument through a SoCWire bus [Osterloh (2008); SoCWire]. In Chapter 8 we will give more information about the communication block, and about the communication between the RTE inverter core and the DPU. The second block is in charge of the scientific calculation and is made up of 12 individual processors, called nano-processors or nProcessors, their net control unit, and a special block devoted to carry out some specific operations.

In our SIMD architecture, each one of the 12 processors, using only its local memory, executes the whole RTE inversion algorithm without exchanging data. This is possible since each input spatial pixel has no data dependencies with the others. Thus, we want to remark that this SIMD architecture is not a classical vector unit, but it can be considered more similar to a GPU architecture and to its SIMT model.

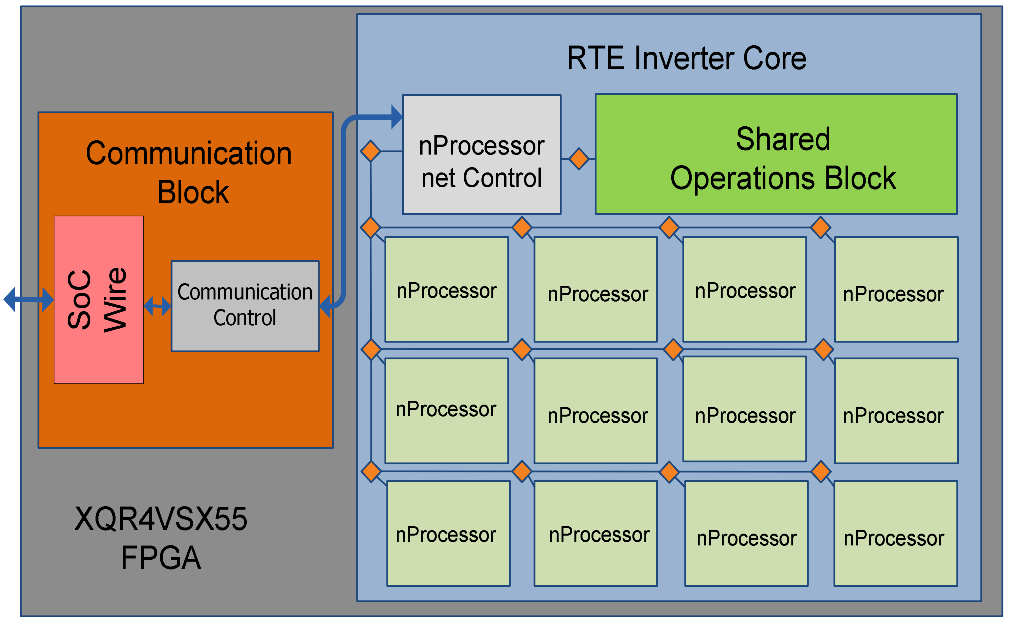


Figure 20 Main blocks of the proposed SIMD architecture within the FPGA

The processors work in IEEE 754 floating point precision. In order to save FPGA resources, the nProcessor has been relieved of executing special operations (e.g., trigonometric ones, divisions or square roots) since they are not much used along the algorithm. One division core, for instance, implies almost a 3% of the FPGA resources. Thus, having a division core in each nProcessor is not feasible. All the removed operation cores from the processors are brought together in a shared operation block, which is used by all of them.

5.1.1 **The Net Control Unit**

In other designs [Baklouti (2010),], the architecture Control Unit is another processor acting as a host, which usually executes scalar operations. In our design, however, the Net Control Unit (see Figure 21) is simplified and only delivers instructions and data to the processors. It carries out fetch and emission of instructions from an Instruction ROM and drives the input and output data flow. SIMD architectures save resources having only one centralized copy of the code. We make the Control Unit even lighter, hence enabling more FPGA free resources for introducing more processors.

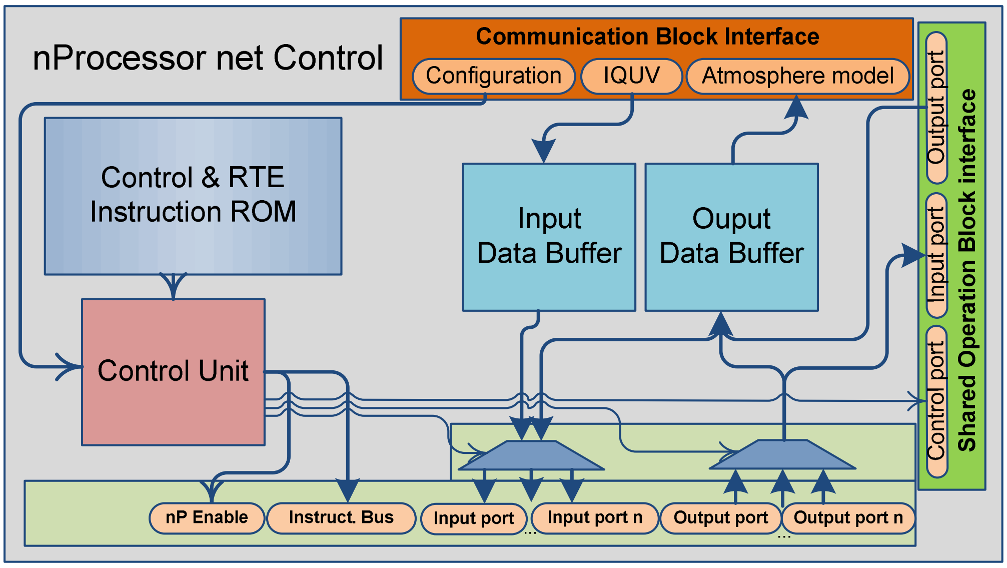


Figure 21 nProcessor Net Control scheme

The instruction flow goes from the Net Control Unit to every processor and all of them receive the same instructions through the nProcessor interface block. However each processor receives different data for operating. Then, there are as many connections between the Net Control Unit and the processors as the number of processors. As shown in Figure 21, the Net Control Unit shares the input data (Input data Buffer) and gathers the output data delivered by each processor (Output Data Buffer), including the shared operation block through the corresponding communication interface. As shown in Figure 21, the Net Control Unit operation is controlled itself by a local Control Unit. SIMD architectures tend to use mask techniques to choose which processors have to execute the instruction coming from the Net Control Unit [Siegel (1979)]. In our case, the mask technique is nothing but a set of enable signals from the Net Control Unit to every processor (nP Enable ports in Figure 21). Usually, these techniques are used to model the architecture behavior according to the instruction or data flow algorithm necessities. However, we only use the mask capability for serializing the access to the shared operation block or to gather and scatter data within the Net Control Unit.

The Instruction ROM contains two main fields: control commands and nProcessor instructions. The first one codes the Control Unit behavior and the second one contains instructions aimed to be executed in the nProcessors. All the instruction codification details have been omitted for doing more readable this document, since they are not necessary for understanding the SIMD proposal. In any case, these details can be found in Appendix IV.

Basically, the control commands are involved in the nProcessor instructions that need to exchange data with the Net Control Unit, as Input and Output instructions, or instruction to be executed in the Shared Operation Block. Using these commands, the Control Unit is micro-programmed to arbitrate the access of the different nProcessors to the shared resources in a serial way.

The Net Control Unit has buffers that the nProcessors have to access one by one for reading or writing data. The control commands are in charge of managing this task masking the nProcessors and allowing that only one nProcessor is enabled at a given time. The access to the Shared Operation Block is carried out in the same way. So, the nProcessors are serialized for sending operands and, later on, for receiving the results.

On the other hand, the nProcessor instruction field stores the nProcessor instructions which control the nProcessor internal working: input and output managing, write and read address, operands load, and the ALU configuration signals. Deep details about instruction fields can be found in Appendix IV.

5.1.2 **The processor**

Using most of the FPGA resources at maximum is essential to fulfill the high computational requirements of our system. Moreover, this type of devices does not usually bear high frequency clocks. Then, the use of pipelined processors and the avoidance of data hazards –that may cause empty cycles– are necessary in order to exploit every operation core at maximum. We, thus, propose a processor design called nanoProcessor (see Figure 22. With this name we want to emphasize that we are designing a new kind of processor. We do not use generic processors, like MIPS or NIOS, as in other SIMD architectures [Baklouti (2010), Xu (2003)]. Rather, we propose the nProcessor which has a pipeline design and which is optimized to execute one instruction every clock cycle in floating point precision.

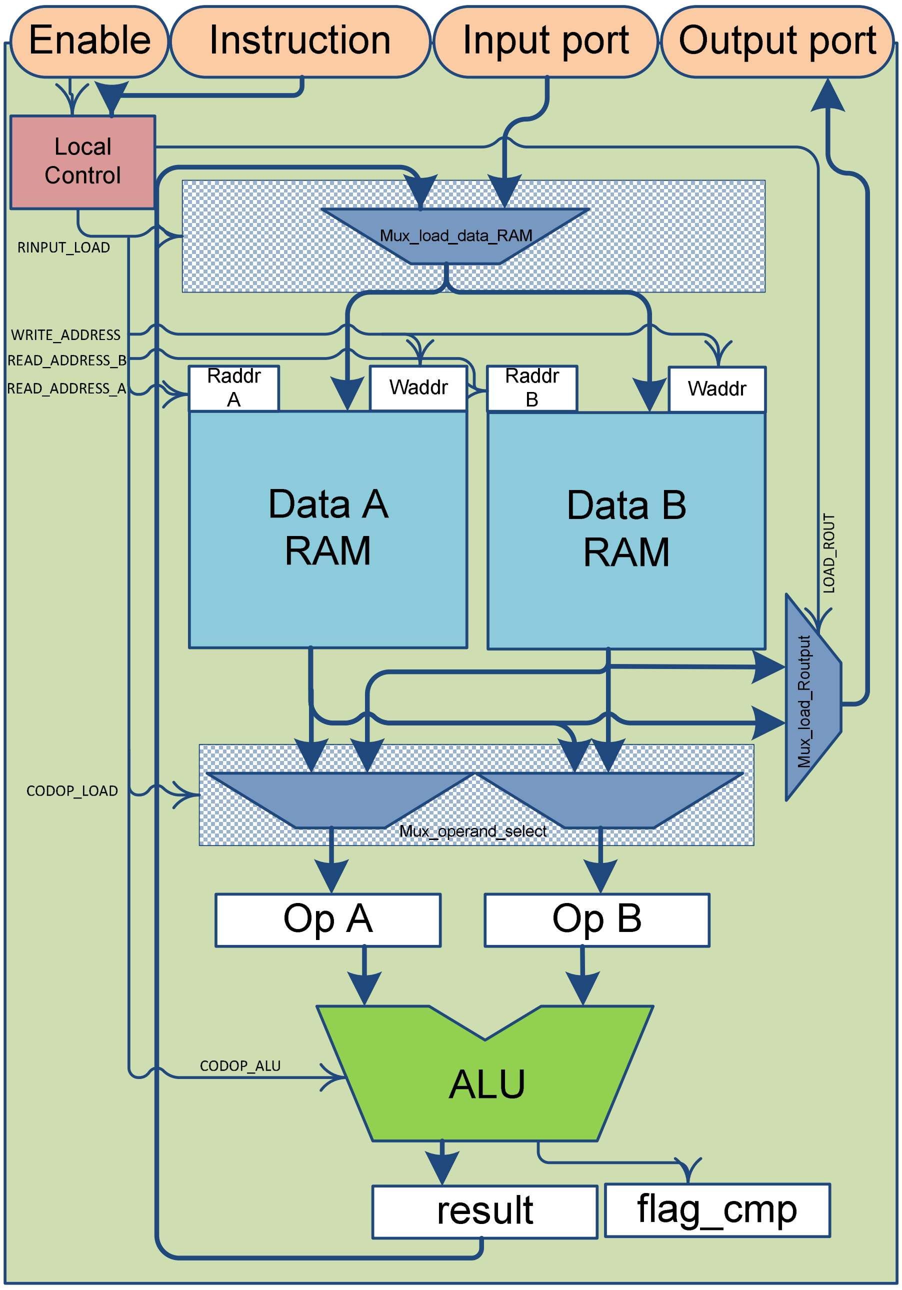


Figure 22 nProcessor architecture

The nProcessor is fed with instructions through the instruction port. Instead of receiving RISC instructions, like in most implementations [Baklouti (2010); Xu (2003); Yiannacouras (2012)], the nProcessor is not micro-programed, but every instruction contains directly the internal control signals decoded: write/read address, load of operands in the ALU, store of the result in the memories, input/output data. Since the instructions arrive decoded, decoding units within the nProcessor are not necessary. The names of the entire field sets within each instruction are shown in Figure 22. The codification is detailed in Appendix IV.

The nProcessor has two data communication ports, one for inputs and another for outputs; both of them use single floating point data. The input port is used to introduce data into the address space of each nProcessor, through a multiplexer for selecting the destination memory. A demultiplexer selects which memory can send data to the output port. As mentioned before, the nProcessor does not have communication with the other nProcessors, but both communication ports are connected to the Net Control Unit.

Regarding the computation capabilities, the nProcessor ALU can execute multiplications, additions, and subtractions in single floating point precision. The subtraction and addition operations are performed using the same physical core, since the subtraction operation is changed into an addition operation changing the sign of the second operand. The ALU is also able to execute comparison operations: equal to, greater than, greater than or equal to, lower than, and lower than or equal to.

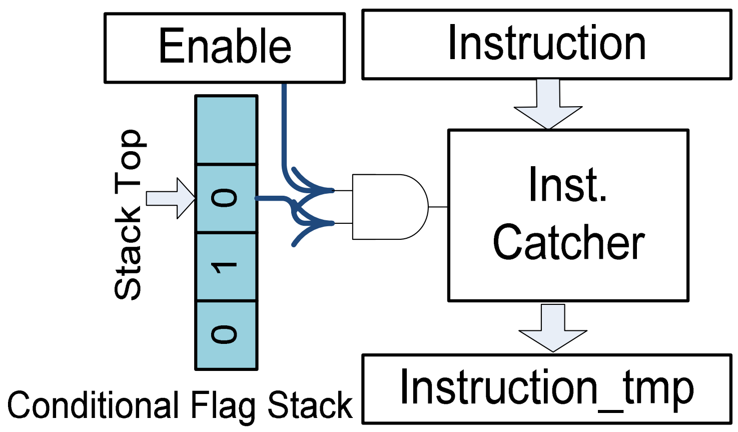


Figure 23 Instruction catch scheme within the Local Control block

However, the comparison operations do not produce any numeric result, but a logic result –into the *flag_cmp* register that is used to produce control signals in order to customize the incoming instruction flow. The local control unit catches the instructions from the input instruction port attending to the previous conditional results and to the Net Control Unit permissions –through the Enable port. Usually the Net control unit broadcasts the algorithm instructions and each processor will follow the conditional path that the previous conditional results have traced. For this purpose, a conditional flag stack (see Figure 23) can be found within the nProcessors. The conditional flag stack stores the results obtained from the *flag_cmp* register, which provides the conditional instruction results. The stack top commands when the nProcessor has to catch the incoming instructions. The else-type instructions invert the stack top value and the endif-type ones remove the stack top. Quite obviously, if the stack is empty there are no previous conditional results and, hence, all the instructions are caught.

Most of the proposed SIMD architectures so far do not use cache memory into the processors, but they are provided with registers, limited data memories [Baklouti (2010), Yiannacouras (2012)], or shared memory where the processors share an address space between all or several of them [Roma (2012)]. In our work, we propose to split the address space in two cache memories (see Figure 22). This way, the processor can be supplied with the two needed operands in one clock cycle and a full pipelined processor design is achieved. The processor register can produce bottle-neck problems and data hazards in the processor performance. Using our proposal based on a double cache scheme, we avoid those problems.

In any case, in pipelined architectures with cache memory others problems could appear that affect the performance, as cache faults due to non-ideal data locality. To deal with this issue, the cache memory size can be configured in our architecture in order to contain all the data used along the algorithm. The cache management is scheduled statically by the compiler as will be detailed in Section 6.2.

5.1.3 **The Shared Operation Block**

As introduced before, to save FPGA resources, some less used functions are centralized into a Shared Operations Blocks (SOB), shown in Figure 24. For example, arithmetic division in floating point is into the SOB since this operation is used less than one per cent of the whole RTE inversion algorithm. That supposes a big saving of resources since only one division core need around 3% of the device resources, meanwhile this operation is used less of 1% of all the operations in the algorithm.

The operation cores in the Shared Operation Block are designed to be pipelined and all the processors can access them in a serial way. They are:

- Division core: it does the floating point division.
- Arctangent core: it does the arctangent operation.
- SinCos core: it does sine and cosine operations.
- SQRT core: it does the square root operation.
- SVD: it does the Singular Value Decomposition for the covariance matrix.
- Normalize and division controller: it normalizes a covariance matrix for being processed by the SVD to the interval [-1,1]. Besides it works as Division Controller.

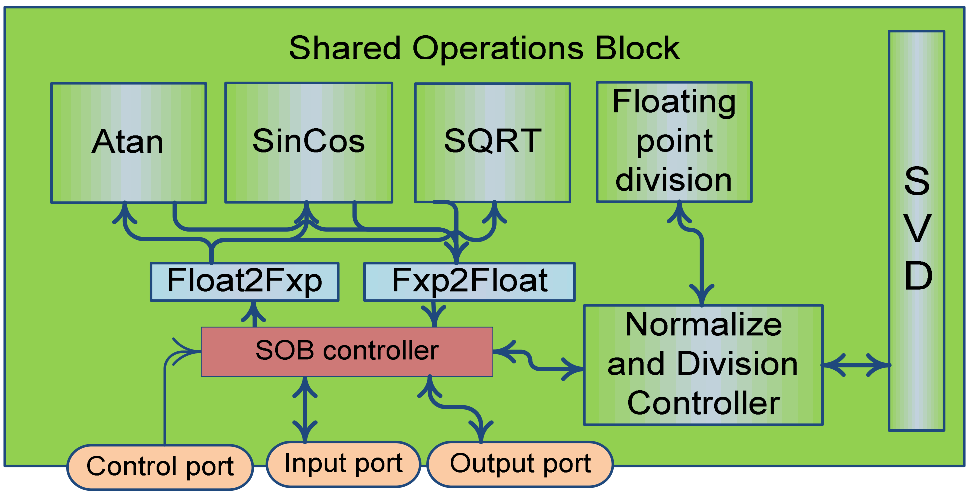


Figure 24 Shared operations block diagram

Since this unit only has an input and an output port, a bus is necessary to connect them with the different operation cores. This bus is inside the SOB control. The SOB behavior is also controlled by the Net Control Unit using the Control port, the SOB receives the necessary control signals in order to control the input/output ports and the operation code (see Appendix IV).

As Figure 24 reflects, there are two cores called Fxp2Float and Float2Fxp. They do a conversion from fixed-point data to floating-point data and the reverse conversion, respectively. This block is necessary because RTE inversion works in 32 bits floating-point precision, but it needs to perform some calculation in fixed-point precision (20 bit) as arctangent, sine, cosine, and square root. The reached precision of these cores, and its resource occupation, is shown in Appendix IV.

In the diagram, the block size is not proportional to its resource occupation since, for instance, the SVD block occupies more than 30% of all the FPGA resources. As mentioned, the SVD block will be studied specifically in Chapter 7.

The SVD needs a normalization of the input correlation matrix since the internal SVD architecture works in the range of data [-1, 1]. The de-normalization of its eigenvalues is made in the nProcessors, but for that, the de-normalization values have to be returned to the nProcessors. Each correlation matrix size consists of 60 values (upper triangular matrix of the covariance matrix for ten free parameters), and its result size is 111 values (the de-normalization value, the 10 eigenvalues, and the 100 eigenvector values).

As Figure 25 shows, the Normalize and Division Controller block performs the normalization of such matrices. They are stored in a FIFO memory (*FIFO_to_SVD*) until the

SVD block is able for processing them. The SVD results are also stored in a FIFO memory until the pProcessors are able for reading them (*FIFO_from_SVD*).

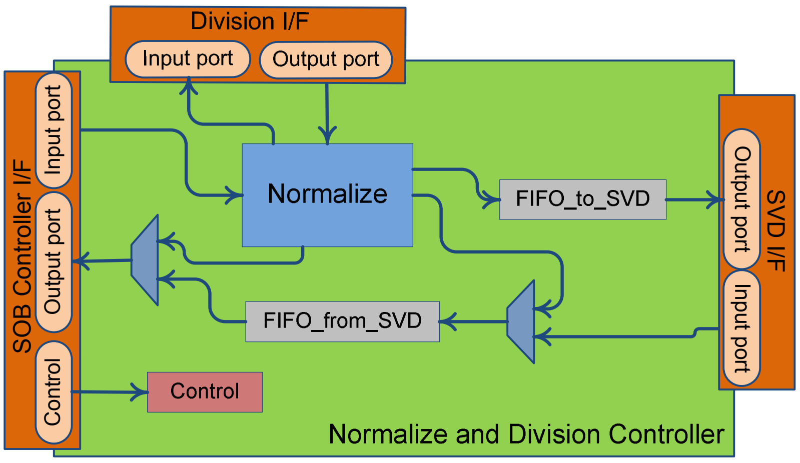


Figure 25 Normalize and Division Controller block within the SOB

It is very important to remark that the Normalize block does not contain a division core, but it uses the division core inside the SOB. The Normalize block is detailed in Figure 26. As can be seen, the division operands can be directly channeled to the Division block. The Normalize block receives the correlation matrix elements in a serial way. While such elements are received and stored in the *FIFO_MATRIX* buffer, their maximum is computed. Once the 60 matrix values are received, the entire matrix is divided by its maximum. The normalized matrix is sent to the *FIFO_to_SVD* through the *Output port1*, and the maximum of each matrix is stored in the *FIFO_MAXS* buffer. This is, the normalization value is not sent to the SVD block, but it has to wait until the SVD is performed. Later this value will be put at the top of its corresponding SVD result. Every time that the SVD block returns a result of a correlation matrix, it is stored in the *FIFO_from_SVD* buffer. Just in that moment, its normalization value is read from the *FIFO_MAXS* buffer and allocated as the first value of the SVD results.

Figures 24 through 26 about SOB and its sub-blocks are shown with great detail for providing a faithful idea of how they work. Control signals, however, have been omitted and communication interfaces simplified. Schematics at RTL level with all these signals are given in Appendix IV (Figure 70 and Figure 71).

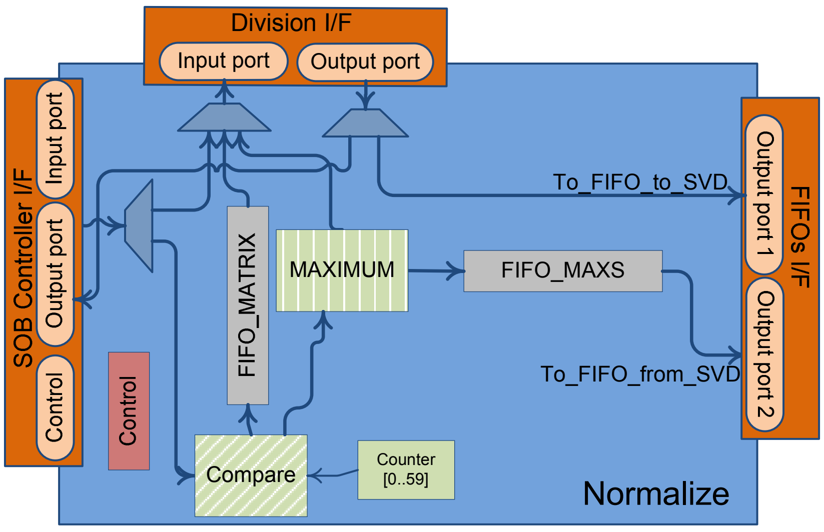


Figure 26 Normalize block within the Normalize and Division Controller block

5.2 The SIMD architecture on the FPGA

The proposed architecture to carry out the RTE inversion inside the FPGA XQR4VSX55 contains 12 nProcessors. This number is not random but the maximum possible number after introducing the remaining necessary elements. In fact, Table 11 shows the total FPGA resource occupation and how it was exhaustive in order to obtain the maximum performance from the proposed architecture.

Resources	Occupation	%
Occupied Slices	24.574	99%
LUTs	30.918	62%
FFs	34.502	70%
BRAM (18kb)	223	69%
DSP48	168	32%
Maximum Frequency	150 MHz	
Power Consumption	4.5 W	

Table 11 SIMD architecture resource occupation

The occupation distribution for each one of the main blocks in the architecture is shown in Figure 27. Percentages are rounded for simplicity. This graph reveals the difficulty of reaching high performance computing using this specific device, after taking overall account of the limitation imposed by the SVD core, which occupies 31% of the logic resources. The SoCWire bus [Osterloh (2008)], connecting the SIMD architecture to the rest of the instrument, takes around a 5% of the FPGA. Finally, discounting shared operations and communications, there is only a 48% of the FPGA for instancing nProcessors. All these facts also justify the use of a SIMD architecture and our “obsession” for saving resources.

Plenty of details about the resources that each element needs in the architecture can be found in [Cobos (2014)]. But we can summarize, regarding others FPGA logic resources, that the BRAM embedded memories are used mainly by the Input and Output Buffers (8 BRAMs), the Instruction ROM (86 BRAMs), and the cache memories in the nProcessors (48 BRAMs). The Instruction ROM occupies 96 BRAM because each instruction word has a width of 54 bits and the maximum program depth has been fixed to 32k words. In the current version only 22k words are used but we have kept this size for future widening. The nProcessors spend 48 BRAM in its cache memories. Although they only use address space of up to 1024 positions, we have kept 2048 positions for future widening.

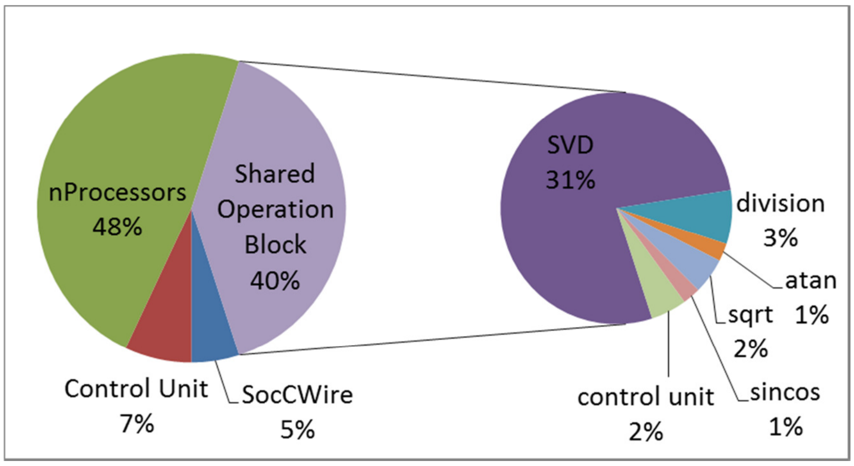


Figure 27 Occupation distribution of the main blocks within the SIMD architecture

The embedded DSP48s are used basically for the ALU operation cores. This kind of resources is essential to implement the floating point operation optimally, in a space and power consumption senses, as was pointed in [Aparicio (2010)] and shown in Appendix III for the MIMD architecture.

The trigonometric shared operation cores and square root core have been generated by the Xilinx Core Generator tools. They use a CORDIC algorithm. The SVD core is designed and is based on the Jacobi method. It will be discussed in Chapter 7.

The system is RTL designed in VHDL and was compiled by the Xilinx ISE 14.1 tool. The SIMD architecture is able to work with a maximum frequency of 150 MHz. However, the SVD and Atan blocks of the shared operation blocks only reach 115 MHz and the SoCWire bus works at 50 MHz. Therefore, there are three different clock domains which are communicating through FIFO memories. These synchronization elements are not shown in the figures for simplicity.

The power consumption is around 4.5 watts in the most demanding case; it is when all the input data are getting to the FPGA without any delay. Anyway, this power is below the system requirement of 5 watts. A deep study about the power consumption in the test device can be found in Chapter 9.

5.3 **The RTE inversion algorithm within the SIMD architecture**

Although details of the scientific inversion algorithm are far from the scope of this dissertation, the main tasks within the iterative process were outlined in Chapter 2. However, in order to simplify the understanding of the execution of the algorithm with the proposed SIMD architecture, we now use a sequence diagram with a simpler task schedule. In short, we consider the main tasks within the RTE inversion algorithm are four: calculation of synthetic spectral synthesis and response functions (SYNTH), calculation of the correlation matrix between the observed and synthetic profiles (CORR), a singular value decomposition (SVD) of the correlation matrix in order to modify the initial model, and the modification of the parameters of the initial model using the eigenvalues and eigenvectors (MODP).

The RTE inversion code for the SIMD architecture is composed basically for these four task codes. It is written in a traditional and sequential C-style way, and it can be consulted in the attached CD. The operation cores in the SOB are called from the code using a function calls style, except for the division case where the mathematical symbol is used. These issues will be described in the next chapter.

In a nominal state, all the nProcessors are executing all the RTE algorithm tasks in parallel. Then they can be blocked for using the shared operation cores: division, arctangent, sine, cosine or square root. In Figure 28 shows an example of using the division and sine cores by the nProcessors, for computing the SYNTH task. In the division case, since it is a non-blocking instruction, the nProcessors send the data to operate to the SOB in a serial way. However, they will receive the results at the same time while they are executing other instructions. However, in the access to other cores, each nProcessor blocks the others until it has finished computing the operation. Once all they finish the shared operations, they will continue executing in parallel the rest of the SYNTH task. How the Sine core is accessed by the nProcessors is illustrated in this example, but is the same procedure for the arctangent and square root cores.

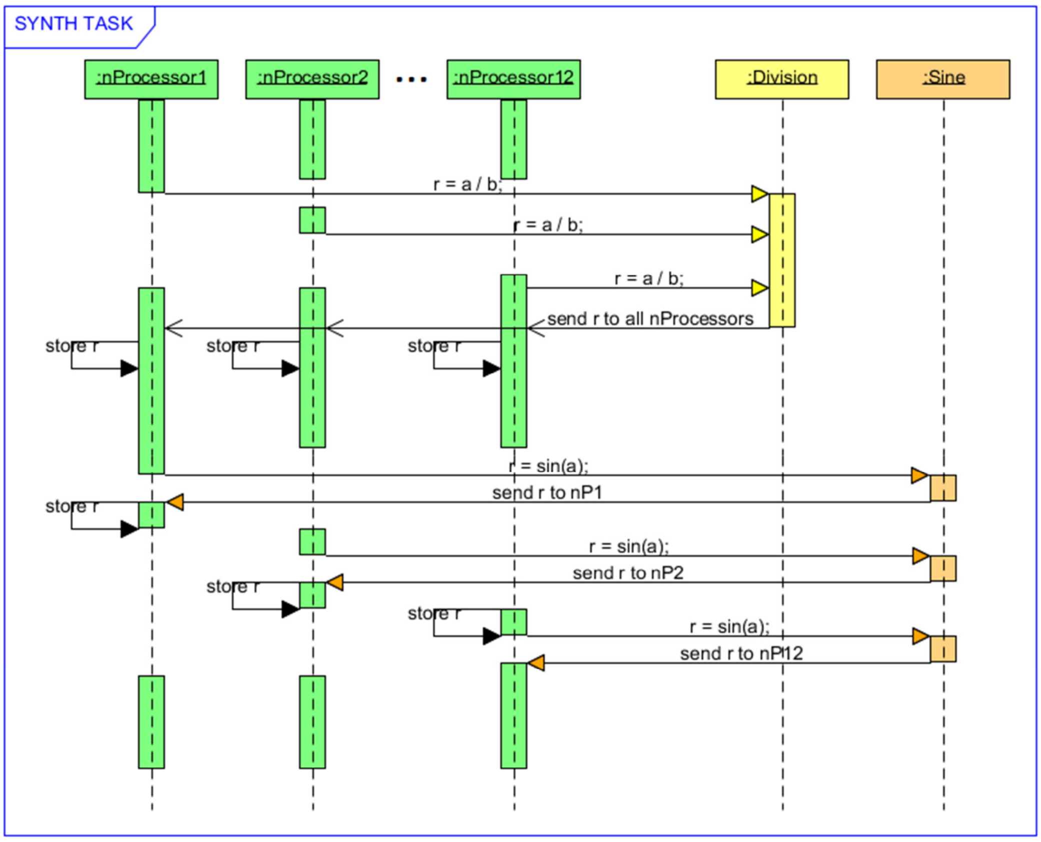


Figure 28 Sequence diagram that illustrates the calls to some SOB functions.

As it has been explained, each one of the 12 processors executes the whole RTE inversion algorithm on different input spatial pixels without exchanging any data. However, the 12 SVD tasks have to be performed by the same SVD core. To avoid a bottleneck in the SVD core use, memory buffers to store the 12 correlation matrices and their 12 results have been implemented. Thus, the nProcessors are able to carry out the other RTE main tasks in parallel. Of course, these parallel tasks are executed on different spatial pixels. With this way of processing the SVD in parallel we reach the optimal use of the device and the architecture, since it is using virtually hundred per cent of the computing elements.

In Figure 29 the task scheduling for taking advantage of the data parallelism and for optimizing the resources utilization is illustrated. In this picture we focus on the access to the SVD core within the SOB, we have removed the calls to other cores for simplicity. In any case, Figure 28 details how all of them are accessed by the nProcessors. In this example we show the schedule of the RTE inversion task according to the iterative process. In iteration 1, the nProcessors load a set of 12 pixels, Pixel_set(I) in the diagram, and all of them carry out the two first tasks (SYNTH and CORR) in parallel. They send the 12 correlation matrices to the SVD core while loading and processing another set of 12 pixels, Pixel_set(J). Note that the nProcessors use the SVD block through the *SVD_in* and *SVD_out* calls, as will be explained in section 6.2. In a nominal iteration, all the nProcessors read previous SVD results and carry out the rest of the RTE algorithm (MODP, SYNTH and CORR) on alternative sets of pixels. In the last iteration, the nProcessors have to read the last SVD results and execute the final modification of parameters. All this scheduling is possible because the SVD core is able of processing the 12 correlation matrices faster than each nProcessor executes the rest of the algorithm.

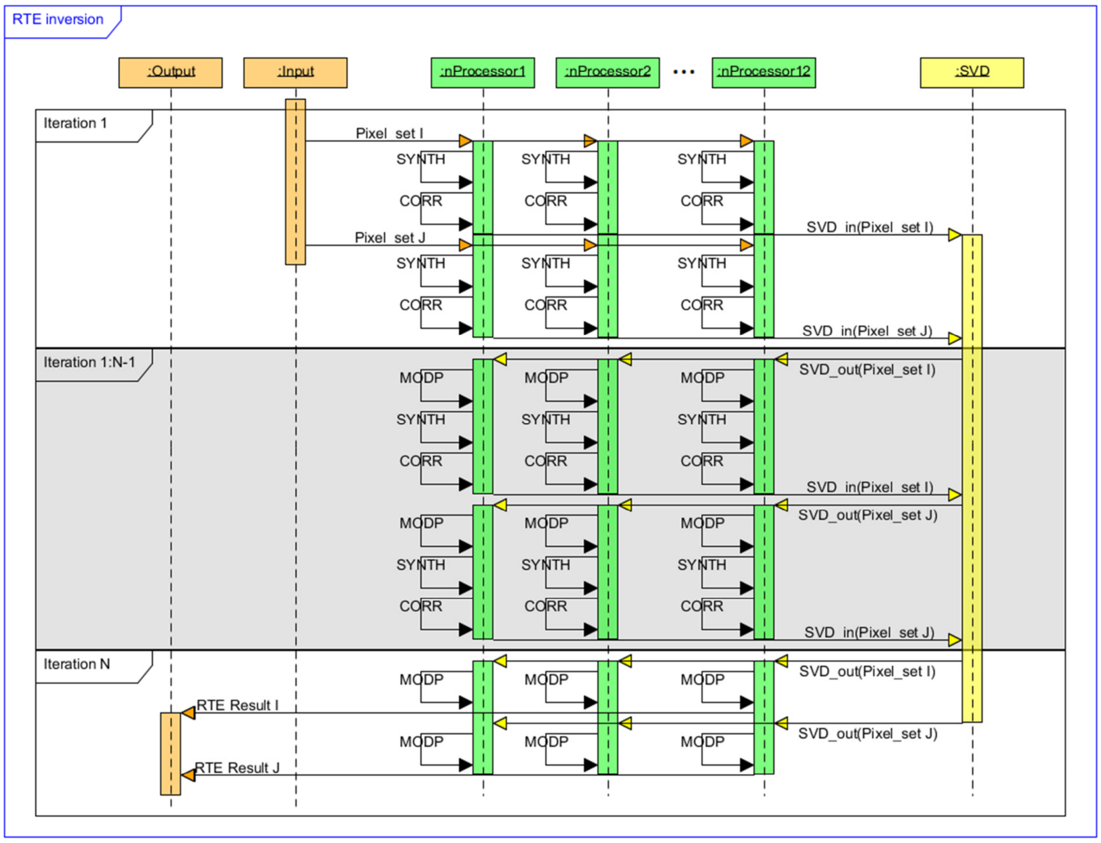


Figure 29 Sequence diagram that illustrates the RTE inversion algorithm tasks working in parallel

As Figure 28 and Figure 29 denote, the data transfer times between nProcessors and the SOB are far shorter than the computational time. The time for completing the various transferences is negligible since most operations in the SOB use only one or two operands. nProcessors send and receive data in a serial way with a rate of one datum per cycle. The Instruction Set Architecture (ISA) will be explained later in detail.

The *SVD_in* and *SVD_out* instructions are different since they have to send the input correlation matrices and receive the obtained results. Each correlation matrix consists of 60 values (upper triangular matrix of the covariance matrix for ten free parameters), and its result size is 111 values (the de-normalization value, the 10 eigenvalues, and the 100 eigenvector values). These instructions are only executed once per iteration. In short, the total amount of time that nProcessors spend doing data transfers with the SOB is less than a 0.5 per cent of the total computation time. This fact also corroborates that sharing resources using SOB (in a serial and pipelined way) is a very good alternative for saving hardware resources with a small loss of time.

In summary, the SIMD architecture is able of executing all the RTE inversion tasks in parallel on different pixels making use of the Shared Operation Block for reaching high temporal utilization of the resources. Since the number of instructions to be executed in the SOB is lower than 1.5 per cent and the SVD core is around a 65 per cent faster than the other RTE inversion main tasks (SYNTH, CORR, and MODP), we can estimate the scalability of the RTE inversion in the SIMD architecture. Introducing more nProcessors in the architecture, the speed up will be linear until 18 nProcessors. If that number is exceeded, the SVD core would be the slowest core and it would block the data pipeline. Then, to reach lineal performance scalability, the system must increase the number of nProcessors in a proportional way to the number of SVD cores (with a ratio 18 to 1). In Section 5.4 we will emphasize other issues about the scalability of the system.

5.4 A reconfigurable and scalable SIMD architecture

Despite the proposed SIMD architecture is clearly focused to the RTE inversion problem, it is enough generic to be applied in other problems, provided they show enough data parallelism.

The architecture can be reconfigured in several features. Such a reconfiguration can be of interest for attacking other problems. A more generic SIMD architecture is presented in Figure 30. Obviously, in the ideal case with enough resources, we can obtain SIMD architectures 100% scalable for embarrassingly parallel problems, simply using a higher number of nProcessors. However, the resource limitation forces us to propose a solution using a shared operation block that moves away from the ideal SIMD scalability.

To minimize the impact of sharing operation cores, we make design decisions as pipelining the shared cores and the data path between the nProcessor and them, as was explained in the previous section. Note in Figure 30 that there is one bus for sending data and another for receiving them.

Other design milestone to increase scalability is to implement the instruction to be executed in the SOB as a non-blocking instruction. In this way the nProcessor can follow executing instructions in parallel to the SOB ones. For example, the division is implemented in a pipelined and non-blocking way. This example is relevant because the division core takes about 30 clock cycles. The SVD case is even more important since the SVD core takes as long as almost the rest of the algorithm. Implementing the SVD instruction as a non-blocking instruction allows us to execute in parallel the rest of the RTE inversion algorithm in the nProcessors –obviously for different input pixels.

An obvious remark is that the scalability of our system depends on the percentage of the algorithm operations which uses cores into the shared operation block. In the RTE inversion case, this number is very low, approximately 1.5 per cent, and then the scalability that can be reached is very high. In any case, for other algorithms the final system performance would be calculated taking into account the code to execute (percentage of operation to share, data dependencies, etc.).

Knowing the resource necessities of each element in the system (see Figure 30) and aiming at using the architecture for other algorithms, the FPGA could contain around 22 nProcessor if the SVD core were not used. It could almost double its theoretical arithmetical capacity. Another possible configuration, to be taken into account for using the system in other problems, is to use the 12 nProcessor with the division core into them.

The precision to use in the architecture can be redefined, in a non-standard floating point or even in fixed point, by simply changing the data bus width and by introducing new operation cores within the ALU.

In the RTE inversion problem a communication network is not necessary. However, the architecture is able to implement a communication network by simply enabling the communication channel bridge (dashed line arrow in Figure 30). Although that resulting network would be rudimentary, it allows executing more complex problems within the architecture.

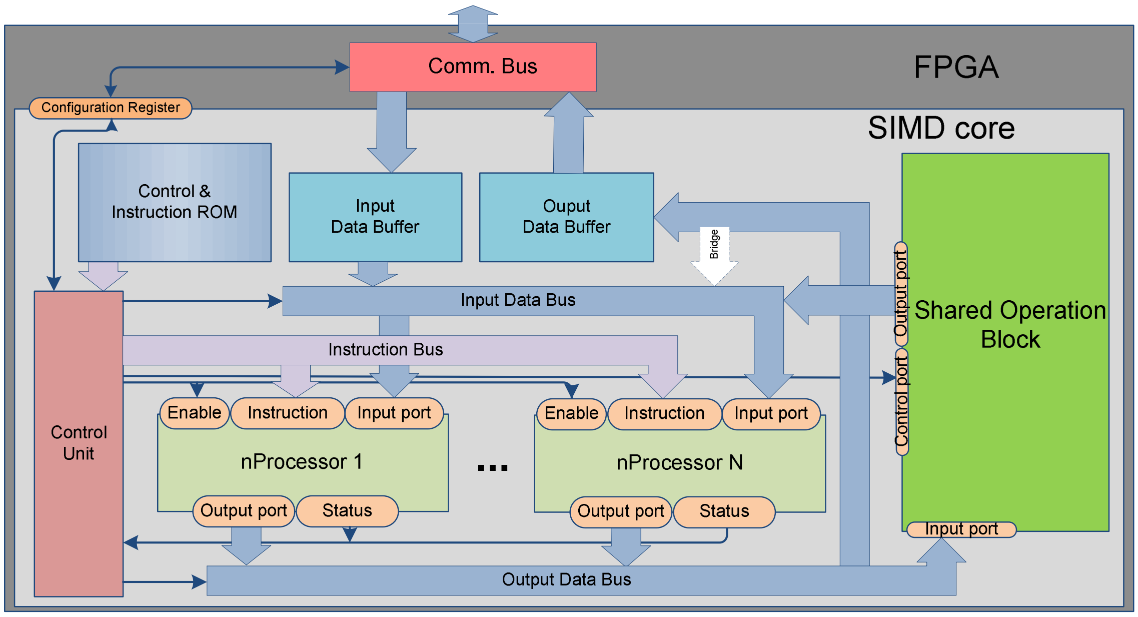


Figure 30 Generic SIMD architecture

6

The TAPAS tool

As mentioned along this work, two different DPU designs have been proposed for the SO/PHI instrument. Each design allocates two different FPGA devices: Virtex-4 and Virtex-5. For each device two parallel computing architectures have been developed in two different flavors: SIMD and MIMD respectively. A software tool to automatize the system configuration processes has been developed. The tool provides a compiler to make it easier the use and programming of the architecture.

The tool is called TAPAS (Tool for Auto-generating Processor Architecture and Simulation). From an input pseudo-code (similar to C, although it only contains operations of two operands), it generates the necessary binaries for configuring both architectures.

6.1 MIMD TAPAS

The software tool takes the initial code and automates the following tasks: to calculate the required number of pProcessors; to allocate the operations into pProcessors; to design the memory space; to design the communication network; and to generate a ROM of instructions for each pProcessor. All this automatization process is carried out taking into account the real-time requirement that applies to this version. In short, given an entry code, TAPAS automatically sets an optimal MIMD architecture using a software intensive-pipelining method. Currently, the pseudo-code is generated using a set of scripts that, starting with the scientific algorithm code, unroll loops and decompose the complex instruction in a chain of two-operand instructions.

Besides, it is necessary to establish the ROM for each pProcessor. Each ROM instruction defines the operations to execute and the control signal to enable at every instant. That is, the control unit (Figure 17) simply extracts the values to establish for every control signal: input port, output port, the operand order and needed counters for consistency. The ROM has as many instructions as the length of a synchronization stage. Each instruction contains control signals, but not all instructions contain operations.

The TAPAS tool counts with a module that from a real C-code is able to decompose complex mathematical operations in several operations with only two operands. This final decomposed code is the TAPAS code. This module is also able of loop unrolling. An example of operations decomposition an unrolling is shown in Code 1. There can be seen how new intermediate variables are created to allow the tree operation generation.

In short, the software tool TAPAS starts with an entry code and carries out the following tasks:

- decomposes the complex mathematical operations in simpler ones,
- makes an operation tree,
- counts the operations of each arithmetic type,
- calculates the minimum number of necessary pProcessors for achieving a temporal constraint,
- allocates the operations in the pProcessors,
- creates the communication network according to the pProcessor operations and the data flow defined by them,
- generates each pProcessor memory space,
- assigns a position and enough space for each data in order to ensure memory consistency,
- sorts the instructions of all pProcessors for preventing collisions at input ports,
- generates the ROM of all pProcessor,
- and generates consistency counters.

An operation tree is usually generated for studying an input code and proposing optimizations over it. Often, such an operation tree is called Data Flow Graph (DFG) [Veen (1986)]. We shall follow this common recipe and employ our own DFG. An example of DFG, which TAPAS generates is shown in Figure 31. The algorithm execution is assumed from left to right. Therefore, inputs are on the left. Those operations that are allowed to be executed simultaneously are in a given column. The data dependencies are indicated using arrows. From this algorithm expression the MIMD methodology exposed in section 4.2 is applied.

```
for(i=0;i<numl;i++){
   parcial1[i]=fi_b[i]+fi_r[i];
   parcial2[i]=(E0_2)*(fi_p[i]-(parcial1[i])/2);
}
```

```
parcial1_0=fi_b_0+fi_r_0;
parcial1_1=fi_b_1+fi_r_1;
parcial1_2=fi_b_2+fi_r_2;
parcial1_3=fi_b_3+fi_r_3;
parcial1_4=fi_b_4+fi_r_4;
parcial1_5=fi_b_5+fi_r_5;
parcial2_0_a=parcial1_0/2;
parcial2_0_b=fi_p_0-parcial2_0_a;
parcial2_0=E0_2*parcial2_0_b;
parcial2_1_a=parcial1_1/2;
parcial2_1_b=fi_p_1-parcial2_1_a;
parcial2_1=E0_2*parcial2_1_b;
parcial2_2_a=parcial1_2/2;
parcial2_2_b=fi_p_2-parcial2_2_a;
parcial2_2=E0_2*parcial2_2_b;
parcial2_3_a=parcial1_3/2;
parcial2_3_b=fi_p_3-parcial2_3_a;
parcial2_3=E0_2*parcial2_3_b;
parcial2_4_a=parcial1_4/2;
parcial2_4_b=fi_p_4-parcial2_4_a;
parcial2_4=E0_2*parcial2_4_b;
parcial2_5_a=parcial1_5/2;
parcial2_5_b=fi_p_5-parcial2_5_a;
parcial2_5=E0_2*parcial2_5_b;
```

Code 1 Example of decomposition in simpler operations and loop unrolling. Initial C-code (up) and decomposed and unrolled code (TAPAS code) (down)

Once TAPAS has processed the input code and has generated a configuration for the pProcessor net, it can be synthetized and tested. For that purpose, the test and validation flow shown in Figure 32 is followed. The MIMD architecture configuration can be synthetized using two different languages: VHDL and Handel-C.

Handel-C was used at the beginning of this project for fast prototyping. However it was left because of a twofold reason: first, because the commercial company gave up supporting it; second, because we wanted to have more control over the synthetized hardware. Overall, once we started to use operation cores from Xilinx, the integration task within the Handel-C code made it harder. In any case, the Handel-C code is in the attached CD and can be revised. It is remarkable that the code length is around a factor 10 shorter than the VHDL code.

The Xilinx ISE tool is used for simulating at VHDL level: generating waveforms or generating files with a list of results. The TAPAS tool generates also a simulation of the input code, these simulation details the pProcessor execution cycle by cycle, and so this simulation can be compared with the VHDL simulation. This comparison is made in Matlab, as is depicted in Figure 32. It is nothing but a script that compares lists of numbers. This simulation is very valuable for debugging the input code, but it also was very valuable for debugging the MIMD architecture, and pProcessor, in the developing phase.

The ISE tool is also used for generating programming files for the FPGA. Once the FPGA is programmed, Matlab is also used for transmitting data through the RS-232 port. The FPGA design for using the RS-232 port is shown in a schematic in Appendix V. The results that FPGA sends are compared in Matlab with the TAPAS simulation result, in execution time. The data are transmitted in ASCII mode but Matlab translates then to a floating-point representation.

In short, following the flow proposed in Figure 32 the MIMD architecture generated for a specific input code can be executed and tested on the FPGA. This will be used in Chapter 9 for explaining the results obtained for the RTE inversion code.

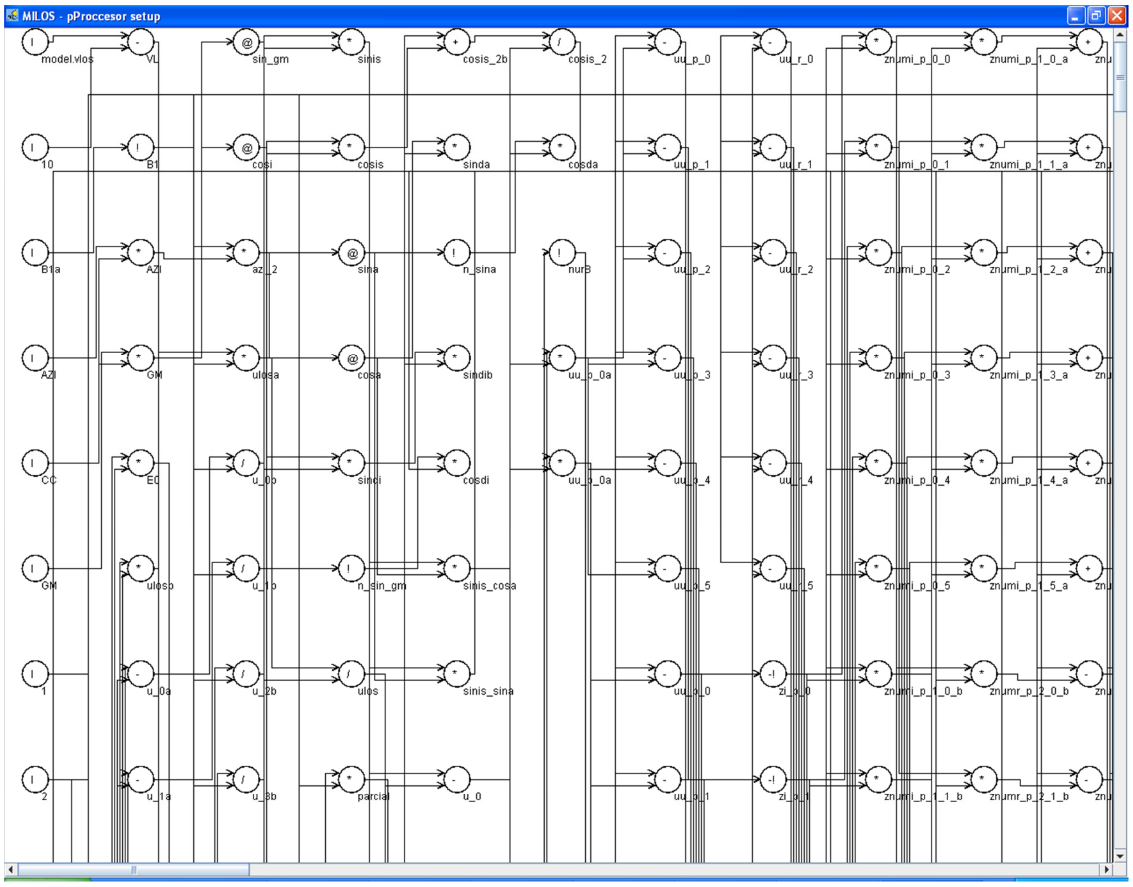


Figure 31 Operation tree exaple using by the MIMD TAPAS version.

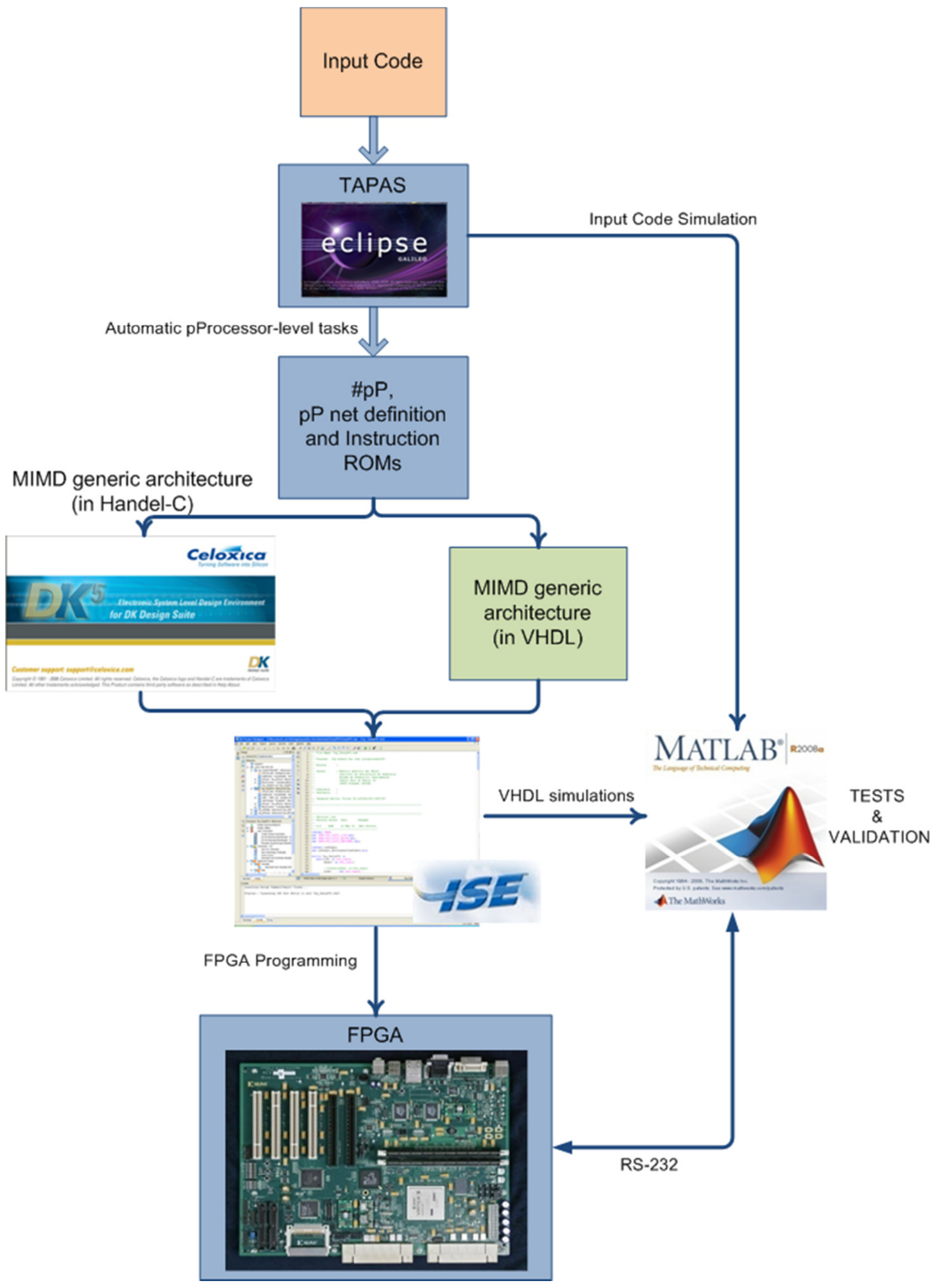


Figure 32 MIMD-TAPAS developing and debugging flow

6.2 The SIMD TAPAS

The second architecture proposed in this work is focused on carrying out the RTE inversion using a SIMD computation model in the Virtex-4 FPGA device. In this section, the TAPAS tool design for compilation and configuration of the SIMD tasks is described.

6.2.1 The programming language

TAPAS is able of decomposing an input code in the so-called TAPAS code for the MIMD architecture (Code 1). However, the TAPAS code for the SIMD architecture needs other features that are described in this section. In addition, since the MIMD had to be frozen due to change of design, the SIMD-TAPAS version is now more advanced. A unification of both versions reveals as an interesting work for the future.

Since the SIMD architectures appeared, some libraries and languages have emerged to program them [Gokhale (1995)]. Tools have also been created [Zima (1988)] to model algorithms in such architectures and offering the capacity to mask processors and create data flow between processors, etc.

In the case of embarrassingly parallel problems where there is no need to decide which tasks are carried out by each processor or to communicate data between processors, or to share memory, this sort of extensions is not required. Thus, the SIMD architecture gets closer to a traditional sequence implementation and turns out to be easy to program. Therefore, as explained in section 5.2, the SIMD architecture does not need either indicate any synchronization commands. These issues are totally transparent to the programmer, since the compiler generates the Instruction ROM which contains the nProcessor instructions and all the control commands, which are able of selecting the processors when it is necessary.

For the architecture presented here, we propose a high level language able to program the device without worrying about the number of processors used, the required data flows, or the internally used architecture. This language takes its name from the architecture compiler itself: TAPAS. We call it a high level language because of its ability for making transparent the architecture assembler to the programmer. There is no need, for instance, to define tags, to manage jumps or function calls, or physical data addresses, etc. Table 12 summarizes the Instruction Set Architecture for giving a programmer view of the SIMD architecture explained in Chapter 5. Two instruction subsets can be identified: instructions executed in parallel within the nProcessors, and instructions executed on a serialized way in the Shared Operation Block.

Instruction Set Architecture (All operands and results are floating point data)				
Instruction	**Allocation**	**Syntax**	**Blocking execution**	**Description**
Addition	**nProcessor ALU (Parallel)**	r = a + b;	Not affected	Arithmetic operation : addition
Subtraction		r = a - b;		Arithmetic operation : subtraction
Multiplication		r = a * b;		Arithmetic operation : multiplication
Greater than		ifgt(a,b)		Comparison Greater than
Greater than or equal		ifgte(a,b)		Comparison Greater than or equal
Lower than		iflt(a,b)		Comparison Lower than
Lower than or equal		iflte(a,b)		Comparison Lower than or equal
Equal		ifeq(a,b)		Comparison Equal
Division	**Shared Operation Block (Serialized)**	r = a / b;	No	Arithmetic operation : division
Arctangent		r=arctan(a/b);	Yes	Trigonometric arctangent operation
Sine		r=sin(a);	Yes	Trigonometric sine operation
Cosine		r=cos(a);	Yes	Trigonometric cosine operation
Square root		r=sqroot(a);	Yes	Square root
SVD_in		SVD_in(a1) … SVD_in(a60)	No	Inputs to the Singular Value Decomposition. The input matrix is given in a serial way.
SVD_out		r1=SVD_out() … r111=SVD_out()	No	Outputs of the Singular Value Decomposition. The eigenvalues and eigenvectors matrix are given in a serial way.

Table 12 Instruction Set Architecture

For the programmer, the use of any operation from the SOB is transparent since they all are perceived as function calls, except for the division where the classic operator is used. The cores within the SOB are custom cores and they do not have to be micro-programed from TAPAS. They only have to implement a serial interface for getting inputs and outputs. This allows carrying out future co-design tasks easily by adding co-processors through the SOB.

The cores within the SOB also have to define if they are implemented in a pipelined and non-blocking way. The division and SVD cores are non-blocking as can be seen in Table 12. The division core only generates one result per instruction and it is easily managed by the Net Control Unit for introducing it to the nProcessor. However the SVD core generates a set of results from a set of inputs, so the input data task and output data task are split in two sets of instructions: *SVD_in* and *SVD_out*, as shown in Table 12.

We cannot forget that the final target is the SIMD architecture. Thus, the TAPAS programming language is oriented to this final architecture and has a series of special features and limitations. For instance, there are no bit-level operations and all the arithmetical operation instructions should have two operands.

A code example of this language is shown in Code 2. With this example code, we attempt to show the main features of the TAPAS programming language. For example, the use of the SOB is carried out using only calls to special functions *–arctan* and *sqroot–*. The input and output are also calls to special functions –Input and Output–; in this case the order is important and corresponds with the order in which the data are supplied to the system, and in which the data are provided by the system. The number of declared inputs is used by the architecture for launching the algorithm execution when there are enough data in the Input Buffer to feed all the nProcessors. Each time the algorithm is executed, the input data are loaded into all the processors in a serial way. Finally, the Output Buffer gathers the outputs generated by the nProcessors, corresponding in number to the Output function calls carried out in the code.

We can see as well how there are no variable declarations, only constant variables are declared with no other intention than assigning an initial constant value to them. All variables starting with *cte_static* are considered constant and static in the cache memory.

The conditional instructions are greater than, *–ifg–*, greater than or equal to *–ifgte–*, lower than *–iflt–*, lower than or equal to *–iflte–*, equal to *–ifeq–*. According to the conditional flag stack seen above, the conditional instruction can be nested in the current version up to eight levels.

Finally, a function is declared in the code example. The function parameters are always passed by reference. In the current version, all functions are converted into an inline code –similar to the C inline code– when generating the executable code.

There are other language sentences although they are not shown in the example. The sentence *call_file file_name* is similar to the include one of the C language.

Also not shown in Code 2 it is possible using loops. The loop bodies, opposite to the conditional instructions, are generic for all the nProcessors, so all the nProcessors execute the code into the body. In any case, the nProcessors can specify a piece of code using conditional instructions into the loop. The loops are defined as for i (0:N-1) and the body is delimited by using endfor. It is important to note that the loops generate hardware counters into the architecture, whose control lies in the architecture control unit.

```
%Constants
cte_static_2 = 2;
cte_static_pi = 3.141593;

%Inputs declaration
Input(input_a);
Input(input_b);

aux = input_a + input_b;
median(aux,cte_static_pi,out1);

out2=arctan(out1,cte_static_2);

iflt(out2,aux)
    out3 = sqroot(aux2);
else
    iflte(out1,cte_static_2)
            out3= cte_static_pi / out1;
    endif
endif

%Program outputs
Output(out3);

%function declaration
function median(val1,val2,rslt)
    aux = val1 + val2;
    rslt = aux / cte_static_2;
endfunction
```

Code 2 Example of TAPAS programming language code

To interact with external hardware signals from the code, the conditional instruction *if_control* signal is introduced. This instruction, like *for*, is global for all the nProcessors. This kind of instruction is used, for example, for deciding if a piece of code is executed depending on an external signal status.

Note that there is no construction for selecting processors or the number or processors in the code. These issues are totally transparent to the programmer.

The simplicity of the proposed language, which allows programming as if it were a totally sequential code, facilitates a compiler. Translation from other languages such as C, Matlab, or IDL is possible. In this sense, TAPAS can be considered as an intermediate language in the future. At present, to translate from C-MILOS into TAPAS we have generated some scripts that make the complex operation decomposition process, variable rename, input/output declaration and loop-unrolling in a semi-automatic way. These tasks are based in the previously presented version for the MIMD architecture.

6.2.2 **The compiler**

To generate the assembler code for the SIMD architecture from the TAPAS programming language we have developed the TAPAS compiler. Unlike other multi-processor architectures, in a SIMD architecture the distribution of the instructions between different processors is not necessary, since all the processors execute the same code. So, TAPAS basically generates the only Instruction ROM and configures some architecture features according to an input program (for instance the maximum value of the program counter).

As mentioned before, the Instruction ROM contains two main fields: the nProcessor instructions and the control commands aimed at managing the Net Control Unit. In order to generate the Instruction ROM, the compiler looks for all the arithmetic instructions from an input code. It translates them to nProcessor instructions. However, the compiler also generates the control commands field for the shared operation function call and the inputs and output instructions case. The details about instruction codification have been omitted for keeping this work readable. We refer the interested reader to Appendix IV.

The nProcessor instructions are stored once decoded, so the compiler has to generate all the nProcessor internal control signals. For this purpose, the nProcessor instructions have different fields for coding the behavior of the different multiplexers, the read and write address of both cache memories.

The control command field codes the behavior of the control unit for executing all the Instructions ROM. The Net Control Unit has to broadcast the instructions to the nProcessors, but also needs to manage the data flow between the nProcessor and the Input and Output Buffer, and between the nProcessor and the Shared Operation Block.

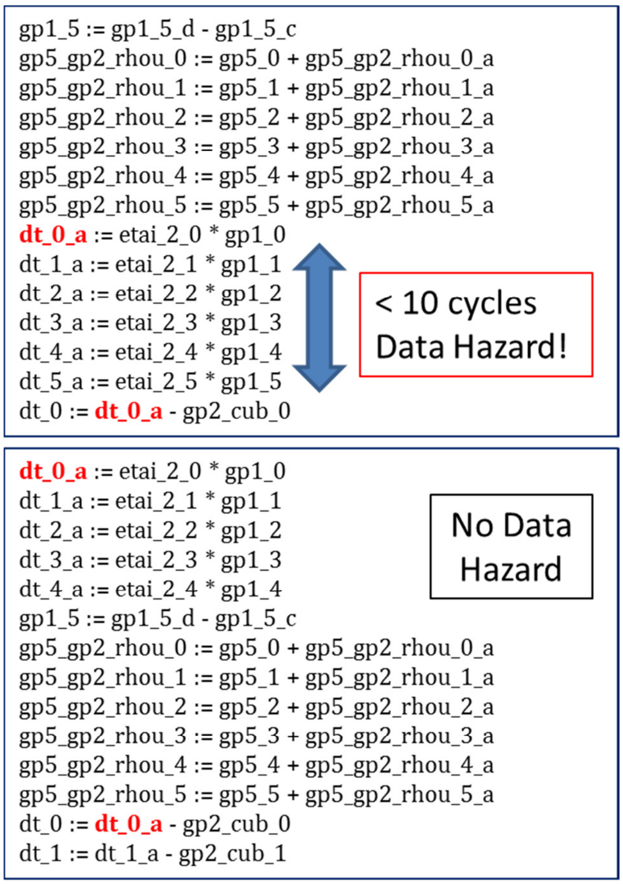
Without going into bit-level details, it is important to remark that the Instruction ROM word width is defined by the compiler. And it depends on the input algorithm features as address space requirement, number of cores in the SOB, or number of nProcessors in the architecture.

Before generating the final object-code for the Instruction ROM, the compiler has to perform some tasks in the input program. The first task is to replace the *call_file* sentences by the corresponding file content and the function calls by the corresponding inline code.

The most important task of the compiler is to reorganize the operations to avoid data hazards and introducing null operations in the cases when this reorganizing is not possible. This means that TAPAS performs a static schedule of the instruction execution. In Code 3, we can see an example of instruction reorganization for avoiding a data hazard. It assumes that the multiplication instruction lasts 10 cycles. Thank to that reorganization, there is no data hazard in that code example and the processor is never stalled.

As previously commented on, by doubling the memory space and using it as a cache memory, we try to ensure the availability of the required data for every operation in one clock cycle. Furthermore, through the static scheduling carried out by the compiler, the smallest size of usable memory is determined.

The procedure has two steps: firstly, the compiler has to assign all the data to one memory and, secondly, it has to assign positions into the memories.



```
gp1_5 := gp1_5_d - gp1_5_c
gp5_gp2_rhou_0 := gp5_0 + gp5_gp2_rhou_0_a
gp5_gp2_rhou_1 := gp5_1 + gp5_gp2_rhou_1_a
gp5_gp2_rhou_2 := gp5_2 + gp5_gp2_rhou_2_a
gp5_gp2_rhou_3 := gp5_3 + gp5_gp2_rhou_3_a
gp5_gp2_rhou_4 := gp5_4 + gp5_gp2_rhou_4_a
gp5_gp2_rhou_5 := gp5_5 + gp5_gp2_rhou_5_a
dt_0_a := etai_2_0 * gp1_0
dt_1_a := etai_2_1 * gp1_1
dt_2_a := etai_2_2 * gp1_2
dt_3_a := etai_2_3 * gp1_3
dt_4_a := etai_2_4 * gp1_4
dt_5_a := etai_2_5 * gp1_5
dt_0 := dt_0_a - gp2_cub_0
```

< 10 cycles
Data Hazard!

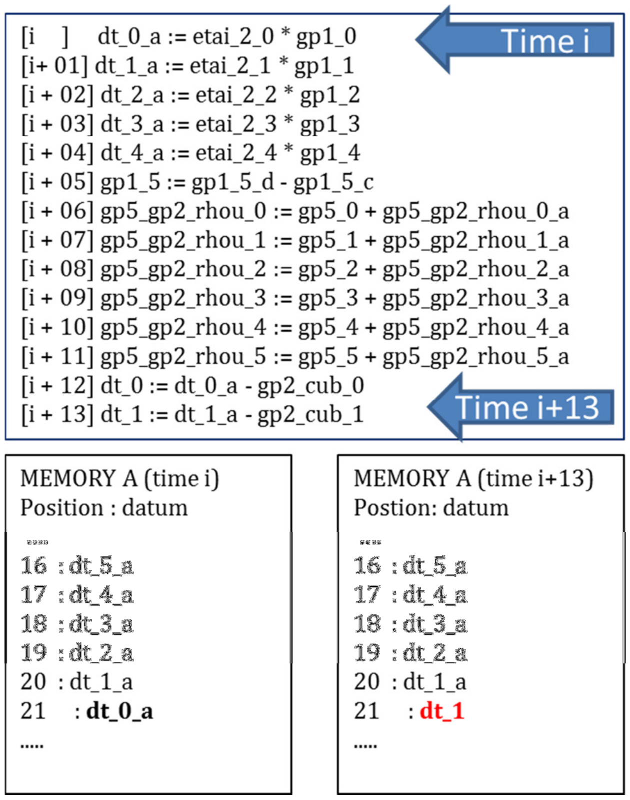
```
dt_0_a := etai_2_0 * gp1_0
dt_1_a := etai_2_1 * gp1_1
dt_2_a := etai_2_2 * gp1_2
dt_3_a := etai_2_3 * gp1_3
dt_4_a := etai_2_4 * gp1_4
gp1_5 := gp1_5_d - gp1_5_c
gp5_gp2_rhou_0 := gp5_0 + gp5_gp2_rhou_0_a
gp5_gp2_rhou_1 := gp5_1 + gp5_gp2_rhou_1_a
gp5_gp2_rhou_2 := gp5_2 + gp5_gp2_rhou_2_a
gp5_gp2_rhou_3 := gp5_3 + gp5_gp2_rhou_3_a
gp5_gp2_rhou_4 := gp5_4 + gp5_gp2_rhou_4_a
gp5_gp2_rhou_5 := gp5_5 + gp5_gp2_rhou_5_a
dt_0 := dt_0_a - gp2_cub_0
dt_1 := dt_1_a - gp2_cub_1
```

No Data
Hazard

Code 3 Example of instruction reorganization for avoiding a data hazard in TAPAS

When assigning a datum to a memory, the compiler needs to study which operations will be using it and if these are commutative or not. The compiler also tries to balance the fill of both memories for practical reasons. Once all the data are assigned to memory, the compiler will assign a virtual position to each datum. The first positions in each memory are reserved for the constant data. Afterwards, the compiler carries out an algorithm virtual execution, assigning each result to the first position with an erasable datum. Erasable position will be the one occupied by a datum not to be used again –except the variables declared as constant; if there is no erasable datum, it will go to the first available position. An example about this is shown in Code 4, where datum *dt_0_a* is written in position 21 of the Memory A at given instant, *i*. This datum is valid in that position until instant *i+12* since this datum will not be used again. For that, the position 21 is occupied in instant *i+13* by another erasable datum and it can be occupied for the *dt_1 datum*.

Following this procedure, the compiler will determine how much memory is required by each processor to be able of executing the algorithm in an optimal scenario. In a general purpose processor, the programmer, using the compiler or handmade optimizations, could reach a balance between loop-unrolling and cache fault, while using our proposal the cache is statically reconfigured to contain all the algorithm address space.


```
[i    ]  dt_0_a := etai_2_0 * gp1_0
[i+ 01] dt_1_a := etai_2_1 * gp1_1
[i + 02] dt_2_a := etai_2_2 * gp1_2
[i + 03] dt_3_a := etai_2_3 * gp1_3
[i + 04] dt_4_a := etai_2_4 * gp1_4
[i + 05] gp1_5 := gp1_5_d - gp1_5_c
[i + 06] gp5_gp2_rhou_0 := gp5_0 + gp5_gp2_rhou_0_a
[i + 07] gp5_gp2_rhou_1 := gp5_1 + gp5_gp2_rhou_1_a
[i + 08] gp5_gp2_rhou_2 := gp5_2 + gp5_gp2_rhou_2_a
[i + 09] gp5_gp2_rhou_3 := gp5_3 + gp5_gp2_rhou_3_a
[i + 10] gp5_gp2_rhou_4 := gp5_4 + gp5_gp2_rhou_4_a
[i + 11] gp5_gp2_rhou_5 := gp5_5 + gp5_gp2_rhou_5_a
[i + 12] dt_0 := dt_0_a - gp2_cub_0
[i + 13] dt_1 := dt_1_a - gp2_cub_1
```

```
MEMORY A (time i)
Position : datum
.....
16  : dt_5_a
17  : dt_4_a
18  : dt_3_a
19  : dt_2_a
20  : dt_1_a
21    : dt_0_a
.....
```

```
MEMORY A (time i+13)
Postion: datum
.....
16  : dt_5_a
17  : dt_4_a
18  : dt_3_a
19  : dt_2_a
20  : dt_1_a
21    : dt_1
.....
```

Code 4 Example of cache static scheduled

To conclude, the amount of variables generated in an algorithm like the RTE inversion is proportional to the amount of operations: around 17000. Applying the procedure seen above, the compiler has determined that using cache memories of 1024 positions memory is enough to carry out the algorithm in an optimal way.

Once TAPAS has processed the input code and has statically scheduled the instruction set execution (the Instruction ROM), the code can be synthetized and tested. For that, the test and validation flow shown in Figure 33 is followed. The SIMD architecture is programmed in VHDL, and it is configured with the Instruction ROM generated by TAPAS.

The same way than the MIMD architecture, the Xilinx ISE tool is used for simulating at VHDL level: generating waveforms or generating files with a list of results. The TAPAS tool generates also a simulation of the input code. This simulation details the nProcessor execution cycle bye cycle. Hence, this simulation can be compared with the VHDL simulation. The comparison is made in Matlab, as shown in Figure 33. It is nothing but a script that compares lists of numbers. This simulation is very valuable for debugging the input code, but it also was very valuable for debugging the SIMD architecture, and the nProcessor, in its developing phase.

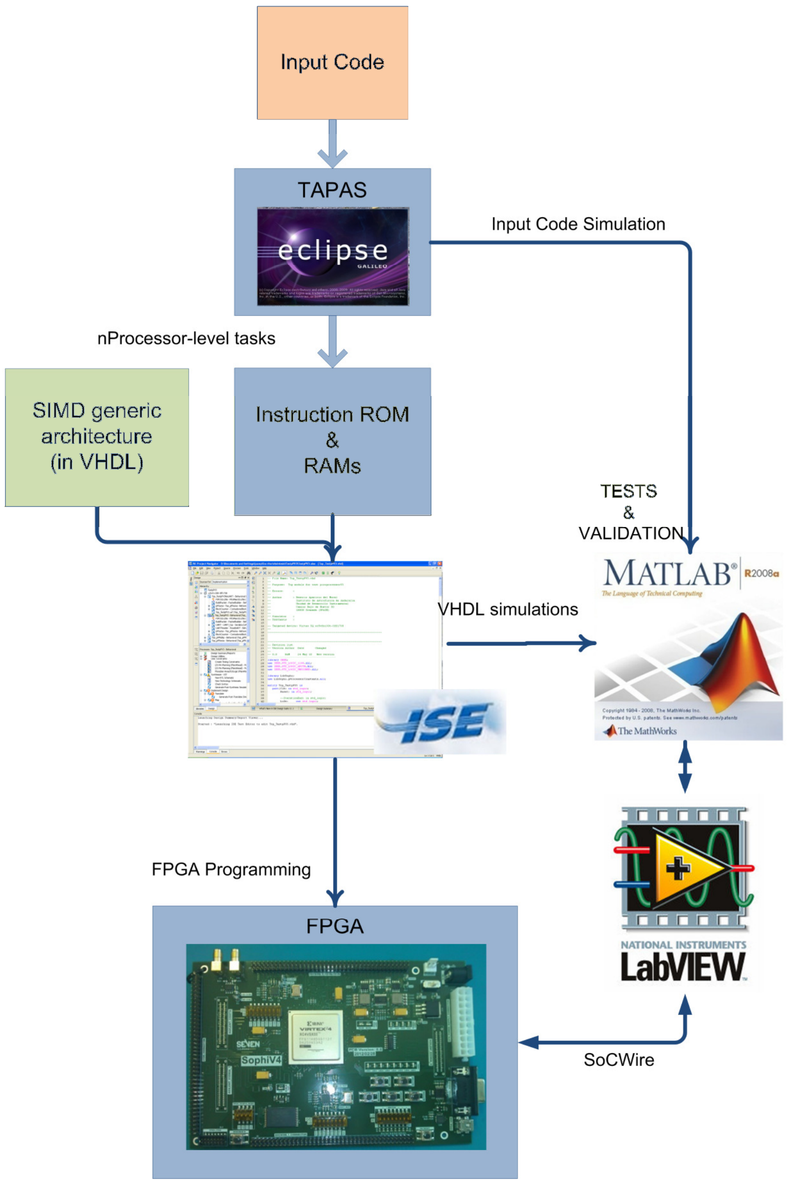


Figure 33 SIMD-TAPAS developing and debugging flow

The ISE tool is also used for generating programming files for the FPGA. Once the FPGA is programmed, LabView is used for transmitting data through the SoCWire bus. The FPGA design of the SoCWire will be detailed in Chapter 8. The results that the FPGA sends are compared, in execution time, in LabView with the TAPAS simulation results. The data are transmitted in floating-point representation.

In short, following the flow proposed in Figure 33, the SIMD architecture generated for a specific input code can be executed and tested on the FPGA. This will be used in Chapter 8 for explaining the obtained results for the RTE inversion code.

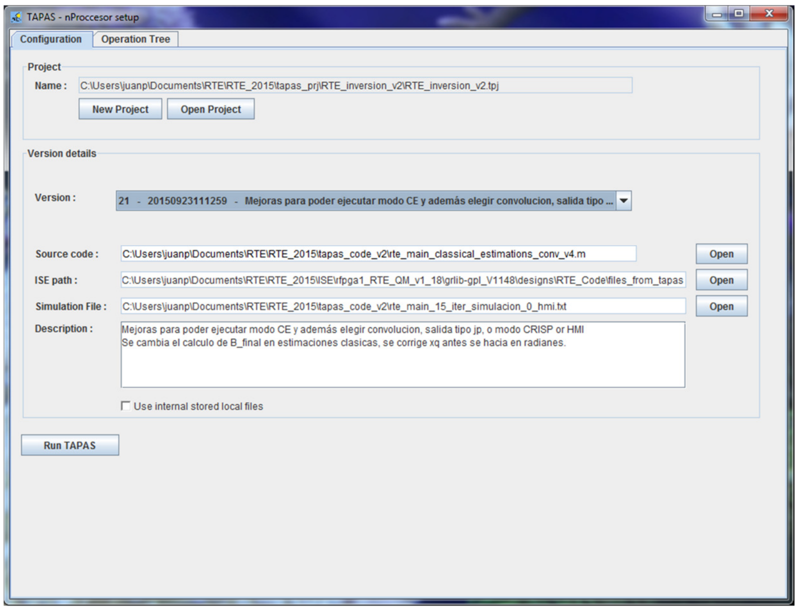
A Graphical User Interface (GUI) has been developed for the TAPAS compiler. It is shown in Figure 34. This GUI provides a friendly way of organizing the compilations in projects. For each project, TAPAS creates different versions automatically. For each version we can introduce:

- the Source code to use as input code,
- the ISE Path where the Instruction ROM, and RAMs will be generated,
- the Simulation file to use,
- and a description about the version.

Every time TAPAS is executed for a version, it stores a copy of that version in a specific directory within the project directory. In this way, we have an automatic log, and backup, of all versions in that directory.

In addition, if we want to use a specific version, we can mark the "Use internal stored local files" in the GUI, and TAPAS executes with the logged version. This is very useful during the testing and developing phases.

The operation tree shown in Figure 31 appears in the Operation Tree Tab of the GUI. This operation tree is very useful for studying the compilation results.


TAPAS - nProccesor setup

Configuration | Operation Tree

Project

Name : C:\Users\juanp\Documents\RTE\RTE_2015\tapas_prj\RTE_inversion_v2\RTE_inversion_v2.tpj

New Project | Open Project

Version details

Version : 21 - 20150923111259 - Mejoras para poder ejecutar modo CE y además elegir convolucion, salida tipo ...

Source code : C:\Users\juanp\Documents\RTE\RTE_2015\tapas_code_v2\rte_main_classical_estimations_conv_v4.m | Open

ISE path : C:\Users\juanp\Documents\RTE\RTE_2015\ISE\rfpga1_RTE_QM_v1_18\grlib-gpl_V1148\designs\RTE_Code\files_from_tapas | Open

Simulation File : C:\Users\juanp\Documents\RTE\RTE_2015\tapas_code_v2\rte_main_15_iter_simulacion_0_hmi.txt | Open

Description : Mejoras para poder ejecutar modo CE y además elegir convolucion, salida tipo jp, o modo CRISP or HMI
Se cambia el calculo de B_final en estimaciones clasicas, se corrige xq antes se hacia en radianes.

☐ Use internal stored local files

Run TAPAS

Figure 34 The TAPAS Graphical User Interface

7

The Singular Value Decomposition within the RTE inverter

In this Chapter we address the design of an SVD architecture to be allocated inside the SIMD architecture; specifically, inside the Shared Operation Block (see Figure 24). Since the RTE inverter based on an MIMD architecture is incomplete, a specific SVD has not yet been developed for it. The SVD architecture presented in this chapter can nevertheless be adapted to the MIMD architecture in the future.

7.1 Introduction to the SVD

The SVD of a correlation matrix is one of the crucial and heavier tasks in the RTE inversion. This task implies more than a 30 % of the total necessary operations, as shown in Table 5. This is even more critical when we study the usual methods for computing the SVD like SVDCMP, TRED+TQLI, etc. (see Section 2.3). Prior to running an iterative procedure, these techniques use matrix transformations that are computationally complex. Among them we find loops, square roots, branches, and other. Such matrix transformations are aimed, however, at reducing the iterative method tasks. The bibliography is full of attempts for accelerating these SVD algorithms in several ways (cluster, multithreads, GPUs). However, in spite of being fairly old, the Jacobi method [Jacobi (1846)] − based on iterative rotations− is well suited for being implemented by parallel processes in hardware as we are going to see.

The Jacobi method obtains a diagonal matrix from an original Hermitian matrix through a sequence of rotations as is explained in [Wilkinson (1988)]. The final non-zero values in the diagonal matrix are the singular values of the original matrix. These singular values are the eigenvalues of the original Hermitian matrix. Since our covariance matrices are Hermitian matrices by construction, we can use the Jacobi method.

The works of Brent and Cavallaro [Brent (1983, 1985), Cavallaro (1987)] have shown how systolic hardware architectures using the Jacobi method can be implemented for carrying out the SVD in parallel. They proposed to de-compose the initial matrix in 2x2 interconnected sub-matrices operating in parallel to get the result. Their proposal is not very efficient for our purposes since they need as many resources in the FPGA as possible for maximizing the parallelism. More efficient architectures, however, were proposed by [Bravo (2006, 2007, 2008)] and [Ahmedsaid (2004)] clearly inspired in the works by [Brent (1983, 1985)]. These architectures are focused in saving hardware resources by re-using as many elements as possible.

To study the performance of our proposed SVD architecture, we have coded the Brent's algorithm in C and carried out a battery of RTE inversion tests using this implementation. Following [Bravo (2006], we assume that our covariance matrix is normalized to its maximum matrix element. We have started by using a long representation in fixed-point precision of 48 bits. Precision has been iteratively reduced as much as possible while keeping the equivalent RTE inversion accuracy to the single floating-point version. After all these iterations, an average accuracy of around 5×10^{-8} in the computed eigenvalues has been found necessary.

In summary, we propose an architecture on FPGA similar to [Bravo (2006)] and [Ahmedsaid (2004)], which are inspired as well in the Brent's proposal. The fastest architecture takes 16.5 µs for carrying out the SVD on a 10 x 10 matrix with an 18-bit precision [Bravo (2006)]. Errors with this precision are of the order of $4\text{x}10^{-6}$ in units of the largest eigenvalue. Our challenge, hence, is bigger than his because we can only afford errors of $5\text{x}10^{-8}$ in less than 14.3 µs (according to the 15 minute time requirement for the whole RTE inversion).

The SVD design shown in this dissertation has not been possible without the works by [Ramos (2011)] and [Aparicio (2013)], where thorough discussions and details can be found.

7.2 The Jacobi iterative method

Several computational solutions for SVD are usually based in the Jacobi method. In general terms, this method consists in appropriately rotating matrices to get diagonal matrices [Jacobi (1846)]. After such operations, the diagonal elements of the obtained matrix correspond to the singular values. This is a sequential process where the previous stage has to be finished in order to complete any stage in the algorithm.

The method consists in the calculation of a proper angle to rotate the matrix so that every non-diagonal element will be equal to zero. Every rotation affects the majority of the matrix elements (exactly those placed in the same row and in the same column). Jacobi showed that each rotation takes us closer to the solution, as each rotation reduces the quadratic sum of the non-diagonal elements. An extended outline about the Jacobi method can be found in [Bravo (2007)].

Note that the initial matrix is finally diagonalized after several rotations and its diagonal elements are the eigenvalues. We call that matrix the eigenvalue matrix along the SVD process.

The final eigenvector matrix is obtained through the product of all the rotations applied in the eigenvalue computation. That is, we depart from the identity matrix and then apply on it the same rotations as in the eigenvalue matrix –the rotation angles for the identity matrix correspond, column by column, with the angles applied to get the eigenvalues in that column. This way, both the eigenvectors and the eigenvalues are obtained simultaneously.

The number of iterations depends on the required accuracy in the eigenvalues. [Brent (1983)] finds that *Mxlog(M)* iterations are needed, where *MxM* is the size of the matrix.

Here, the algorithm input matrix is a covariance square matrix, that is, a symmetrical matrix of 9 x 9 elements. The matrix symmetry reduces the amount of rotations needed for the diagonalization [Götze (1993)].

7.3 The Brent's algorithm

A hardware-focused implementation of the Jacobi method was presented by [Brent (1983, 1985)]. Such an implementation exploits the implicit parallelism that the method provides.

For square matrices and with an even number of rows and columns, Brent proposed a half-parallelizable algorithm for the SVD process. He suggested to de-compose the initial matrix in 2 x 2 sub-matrices that operate interconnected to get the result. Brent named "processors" each of these sub-matrices according to their operation mode. Table 13 specifies the three types of such processors.

Since our correlation matrix has a size of 9 x 9 elements, we have to fulfill the matrix with zeros for getting a squared matrix with an even number of rows and columns. Hence, our problem matrix has 10 x 10 elements. The 10 x 10 covariance matrix is divided in 2 x 2 sub-matrices. The diagram in Figure 35 shows an illustration of such a decomposition. Likewise, the identity matrix is decomposed in PV processors to get the corresponding eigenvectors in Figure 36

Name	Description
Diagonal Processor (PD)	The (2x2) sub-matrices with elements from the initial matrix main diagonal
Non-Diagonal Processor (PND)	The (2x2) sub-matrices with elements that do not appear in the initial matrix main diagonal
Vectorial Processor (PV)	The (2x2) sub-matrices which operate on the identity matrix for the eigenvector calculation. The operations on these sub-matrices are related, column by column, with the operations applied to get the eigenvalues in the PD

Table 13 Types of processors in the Brent's algorithm

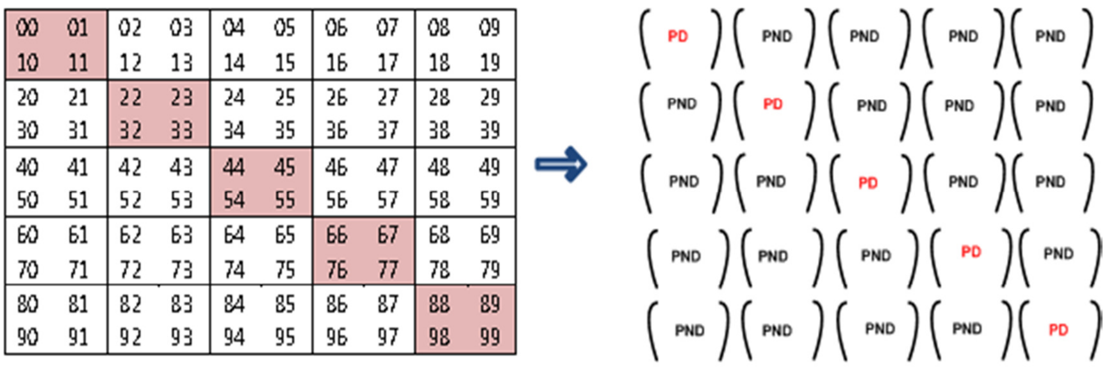


Figure 35 Eigenvalues matrix decomposition into processors

.

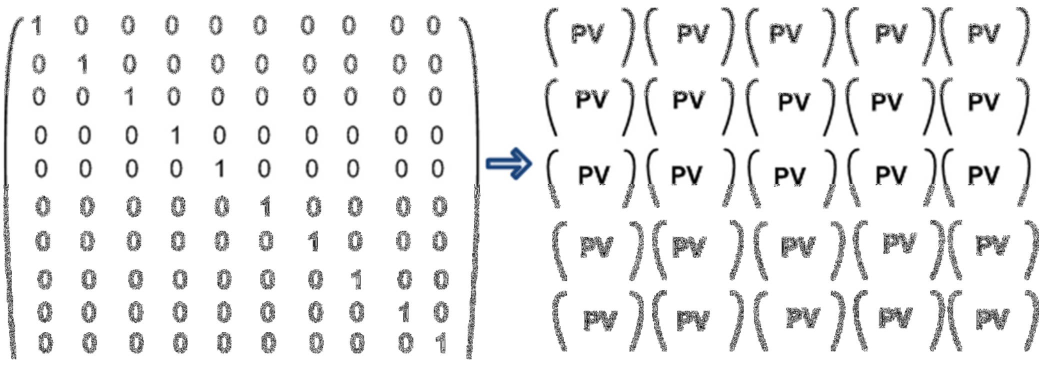


Figure 36 Eigenvector matrix decomposition into processors

Therefore, the Brent algorithm consists of a mesh of rotation matrices iteratively working in parallel. It is important to note that we are working with symmetric matrices. Hence, the number of "processors" can be reduced, as proposed by [Götze (1993)] and adopted by [Bravo (2006)], by taking the upper (or lower) triangular matrix. In our case we have finally 5 PD, 10 PND, and 25 PV processors.

The algorithm is iterative and the solution is closer after each iteration. Any iteration has four stages. In each of them, the PD, PND, and PV processors have to perform a number of operations as detailed in the following sections. The precision requirements will determine the amount of iterations that we should apply in the process.

7.3.1 Stage I: angle calculation

Each PD calculates the rotation angle required to cancel its corresponding non-diagonal elements. At this stage, all PDs work simultaneously. The diagram in Figure 37 shows the involved PD processors (one color per processor). Only the necessary processors for a symmetric matrix are shown.

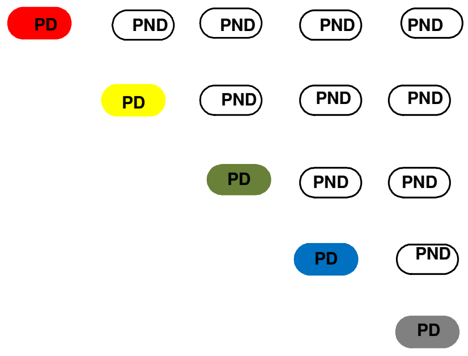


Figure 37 Stage I Brent's algorithm (angle calculation)

Assuming a diagonal processor like $\mathbf{PD} = \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix}$,

the rotation angle is given by

$$\propto = \frac{1}{2}\tan^{-1}\left(\frac{2\,X_{12}}{X_{22} - X_{11}}\right). \tag{1}$$

Note that $X_{12} = X_{21}$ because our matrix is symmetric.

7.3.2 Stage II: angle transmission

In the next stage, the diagonal processors transmit the value of the calculated angle, α, to all the processors being on their rows and columns (see Figure 38).

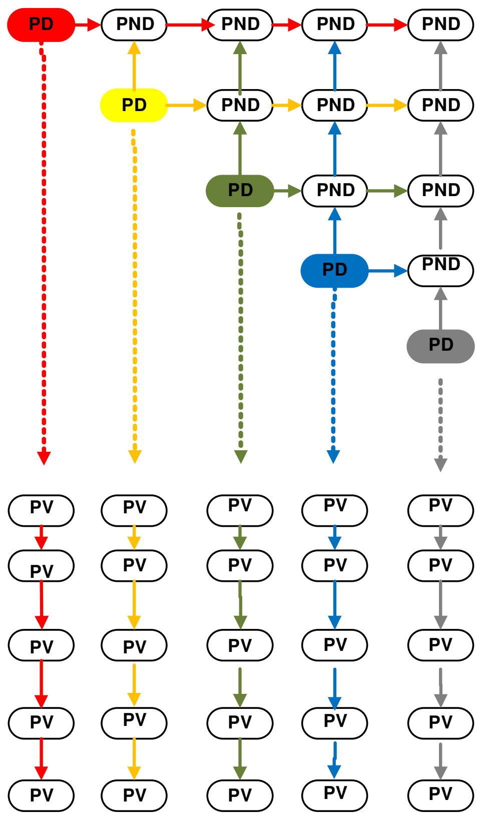


Figure 38 Stage II Brent's algorithm (angle transmission)

The PND processors receive the corresponding angle from their row PD and column PD while the PV processors receive the angle from their corresponding column PD. Once all the angles are received, all the processors are ready for the next stage.

7.3.3 Stage III: simultaneous operations

In this stage, all the processors have the angles to carry out their corresponding operations simultaneously. PD and PND both carry out a double operation. PV performs a single operation. Below we detail how these operations are performed inside them.

- (Stage III - PD) Diagonal processors

  The PD processors are transformed by rotation of an angle α of their original expression, according to

$$\mathbf{PD}' = \mathbf{R}(\alpha_{\mathrm{column}})^{\mathrm{T}} \cdot \mathbf{PD} \cdot \mathbf{R}(\alpha_{\mathrm{column}}), \tag{2}$$

where $\mathbf{R}(\alpha) = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$, and α_{column} being the angle previously calculated in Stage I.

- (Stage III - PND) Non-Diagonal Processors

  Instead of a pure rotation, PNDs transform according to

$$\mathbf{PND}' = \mathbf{R}(\alpha_{row})^{\mathrm{T}} \cdot \mathbf{PND} \cdot \mathbf{R}(\alpha_{\mathrm{column}}), \tag{3}$$

  where α_{row} and α_{column} are the angles received from their row and column PDs, respectively.

- (Stage III - PV) Vector Processors

  The vector processors transform according to

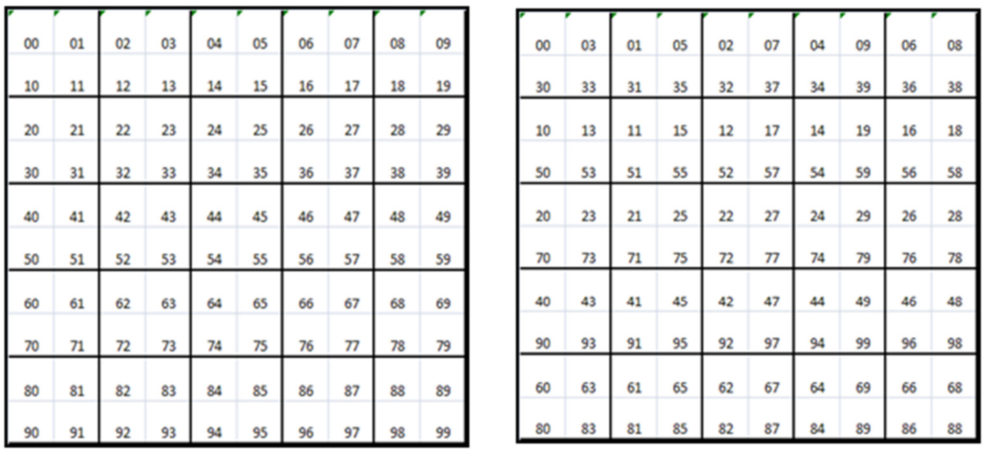
$$\mathbf{PV}' = \mathbf{PV} \cdot \mathbf{R}(\alpha_{\mathrm{column}}). \tag{4}$$

7.3.4 Stage IV: re-sorting elements

In this stage, we re-sort all the matrix elements in order for all the possible pairs of data to pass through the diagonal processors. Likewise, we carry out the corresponding sorting of the data affecting the PV so that we do not lose the correlation between eigenvectors and eigenvalues.

As described in [Brent (1983)], the re-sorting of data corresponds in fact to a re-sorting of data within the processors themselves and an exchange of data between the different processors. The general idea is to fix the element (0,0) of the matrix and exchange rows and columns within each processor; then, later exchange rows and columns between processors.

These operations can be specified in a clearer way by indicating which is the new position of the matrix elements after having re-sorted the initial one. The diagrams of Figure 39 show the initial position (a) and the position where the data must be written to be re-arranged (b). Once the matrices are re-sorted, the four stages of the algorithm can be repeated again.



00	01	02	03	04	05	06	07	08	09
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

(a)

00	03	01	05	02	07	04	09	06	08
30	33	31	35	32	37	34	39	36	38
10	13	11	15	12	17	14	19	16	18
50	53	51	55	52	57	54	59	56	58
20	23	21	25	22	27	24	29	26	28
70	73	71	75	72	77	74	79	76	78
40	43	41	45	42	47	44	49	46	48
90	93	91	95	92	97	94	99	96	98
60	63	61	65	62	67	64	69	66	68
80	83	81	85	82	87	84	89	86	88

(b)

Figure 39 Positions of covariance matrix elements, before and after the rearrangement process

7.4 The SVD architecture design

The proposals by [Brent (1983, 1985)] use specific hardware for each sub-matrix using a systolic architecture. However, this approach is obviously too demanding in terms of hardware resources. This issue was improved in a recent proposal by [Bravo (2006, 2007, 2008)], where specific pipelined cores are used. Bravo suggested using only one unit for calculating the angle and another one for rotations in a pipelined way. Since we need to save as many resources as possible we have taken the proposals by Bravo as a reference.

The necessary tasks in the Brent's algorithm are sketched in Figure 40. The eigenvalue and eigenvector matrices are iteratively introduced into the four stages of processing explained in the former section. In this picture, the task dependencies that the Brent's algorithm requires are reflected. It clearly illustrates how the angle calculation stage stalls the following stages, so that they cannot be executed in parallel. Each sub-matrix of the eigenvalues matrix needs two transformations (PDs and PNDs; see stage III and Equations (2) and (3)), while sub-matrices of the eigenvectors matrix only need one (PVs; see stage III and Equation (4)).

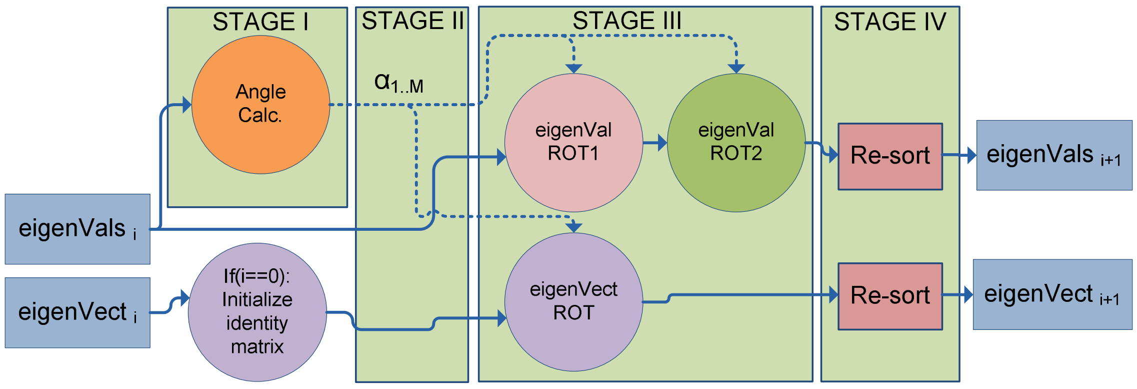


Figure 40 SVD algorithm task dependencies

The use of pipelined cores with a high throughput, like one result per cycle, is justified by [Bravo (2006)]. This is a clear advantage with respect to the Brent's architecture where several angle and rotation cores are needed in stages I and IV. We follow the same strategy in our proposal. On the other hand, Bravo modified the angle unit for being used also as a second rotation unit during stage III in order to accelerate the processing of that stage. However, we are going to arrange the architecture in an alternative configuration.

The task dependencies do not permit to execute both main stages in parallel over a unique correlation matrix, as Figure 40 illustrates. This obviously limits the performance of the architectures in [Bravo (2006)]. Hence, we suggest computing two Singular Value

Decompositions over two different correlation matrices at the same time. In this way, Stage I and Stage III can be executed over each matrix in parallel without any conflictive task dependence.

As we explained in Section 5.3, the SVD block is inside the Shared Operation Block. This means that the 12 nProcessors can use it in a shared way in the SIMD architecture. Nevertheless, only two such correlation matrices can enter the SVD block at a time.

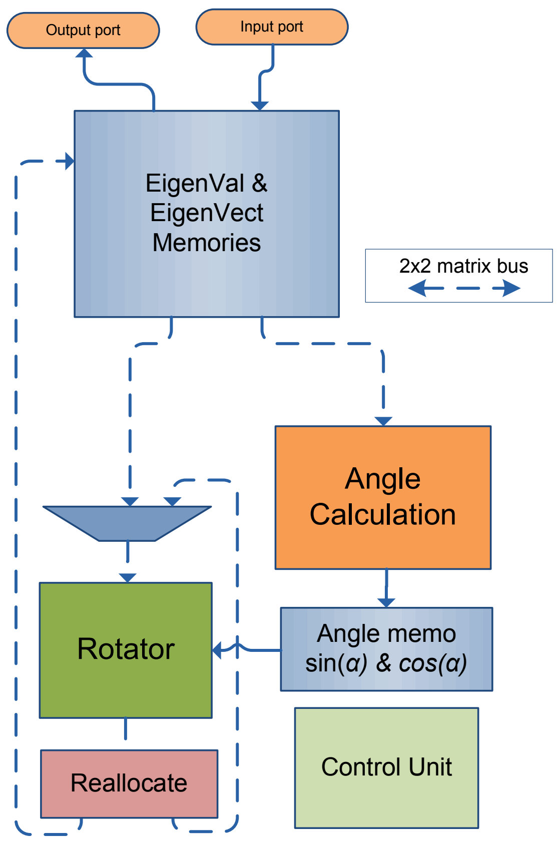


Figure 41 SVD architecture

So, in the same way as Bravo, we propose an SVD architecture that uses only one pipelined angle calculation core and one pipelined rotation core. This is illustrated in Figure 41, where the architecture is sketched. Our improvement to former proposals is the simultaneous calculation of the SVD for a second matrix. Thus, our SVD architecture is improved and optimized for carrying out the decomposition of two correlation matrices at the same time in a pipeline way. Figure 42 tries to illustrate how both cores are always working in parallel and the matrices use each operation core alternatively. That is, while a matrix M_B is executing Stages I and II (angle calculation and angle transmission), other matrix M_A is executing the Stages III and IV (rotations and re-allocation) using angles previously computed. We have called Angle Stage and Rotation Stage to these parallel tasks. The Angle Memo block in Figure 42 is the buffer that permits to disconnect the two main tasks. This block is provided with memories to store the computed angles in the Angle Calculation block that are later used in the Rotator block.

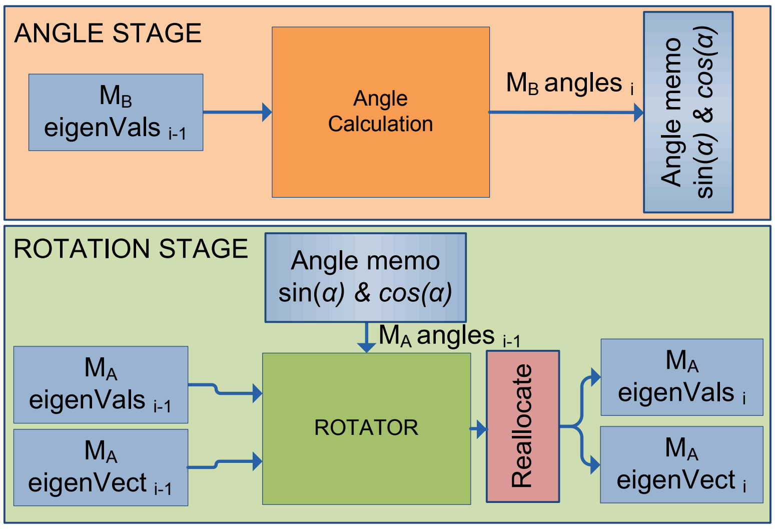


Figure 42 Pipelined Scheme of the SVD architecture

The SVD block performance and resource occupation obviously depend on the desired precision. Defining a target precision is not an immediate decision because the SVD algorithm implies an iterative procedure nested within the iterative RTE algorithm. Besides, the SVD result conditions the convergence along the RTE inversion. According to our numerical experiments in Section 7.1, we have concluded that 27 bits are enough in the Angle Calculation block for getting the target SVD precision (5×10^{-8} in the computed eigenvalues). The rotation core works in single floating point precision like most parts of the RTE system, as we are going to show later. This means that data paths and memories have to support this word length. The consequences will be analyzed later.

A Control Unit block is also shown in Figure 41. This block handles all multiplexers, decoders, enable signals for memories, and the rest of control signals in order to control the pipelined SVD process. Most of these signals are omitted in the diagram for simplicity. We describe the most important control issues in the next subsections for a better understanding of our proposal. In any case, the Finite State Machine of the Control Unit Block has been thoroughly detailed in [Cobos (2013)].

7.4.1 Eigenvalue and eigenvector memories

Working with two correlation matrices in parallel implies to store both matrices in memory. This increases the needs for memory size. But this issue is not a severe problem since the matrices are not enormous and the FPGA has enough Block RAM to allocate them (see Table 5).

Both computation cores need a sub-matrix of 2 x 2 elements for operating (buses for transmitting 2 x 2 matrices are used as indicated in Figure 41). This means that it is necessary to read four data in a cycle for an optimal performance. Bravo uses double-port memories (which provide two data in one cycle). The work frequency of these memories is double than the rest of the SVD system. So, he obtains the four data in one cycle of the main SVD system. He can also write four data in the same cycle but he does not need to write and read simultaneously because the angle calculation stage does not work in parallel with rotations in his architecture. Then, until the eigenvalues matrix is not completely finished, the angle calculation cannot be executed again.

The solution in [Bravo (2006)] has two drawbacks for us. The first one is that we need up to eight data in one cycle (remember that both cores are working in parallel), but the double-port memory technology in the Virtex 4 FPGA does not permit that. The double-port memories only permit write or read two data in one cycle at maximum. The second inconvenience is that using the double frequency in the Block RAMs increases the power consumption significantly. Using high frequency clocks in a space system could add other problems in the device since the probability of having an error due to radiation increases with increasing clock frequency [Engel (2006)].

We have solved the first issue by using two memory address spaces for allocating the two matrices: Memory A and Memory B. To get four data in one cycle without using two different frequencies, we propose to split each memory address space in two dual-port Block RAMs. In Figure 43, this splitting is indicated with different colors and with a suffix to the memory name: MemoryA_1, MemoryA_2, MemoryB_1, and MemoryB_2. Since each dual-port Block RAM is able to provide two data, four of them are able to provide 8 data.

Since each matrix is doing a different task with a different memory, it is never stalled by the other matrix. But it is important to remark that this memory organization also solves the problem of writing the four rotation results because it can write them in one cycle without any problem.

Storage of an initial matrix into a specific memory address space, which has been split in two memories, is not straightforward. The sub-matrices are always read in the same order but the writings are done in the Reallocate block in different positions. To guarantee that four elements can be read in one cycle, they always have to come two by two from the two memories. In the same way, to write four elements in one cycle, then, they have to go two by two to the two memories. In Figure 44, we show the splitting that we have proposed for the input matrix and for the initial identity matrix. Colors correspond to the two different dual-port memories. There are several combinatorial possibilities, but we have selected a solution that permits some homogeneity in the process, for that there only are two types of readings and writings, namely, readings and writings of crossed elements in the sub-matrix. We called them *crossed readings* and *writings* hereafter. This scheme is easily scaled for other matrix sizes.

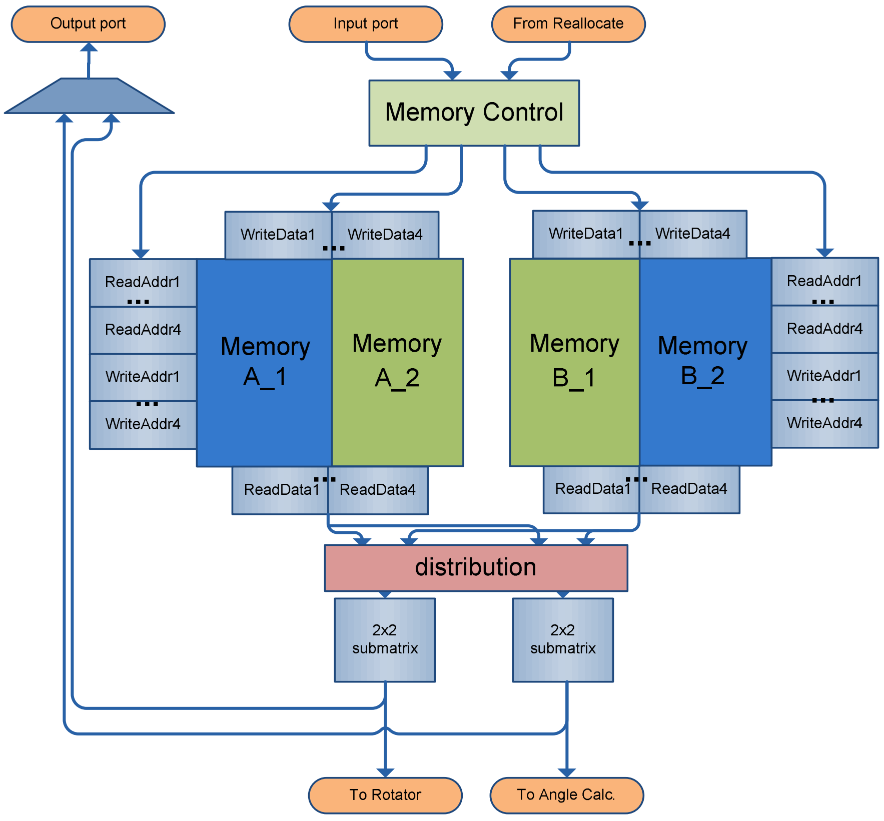


Figure 43 Matrix memories implementation detail

The matrix elements of the different processors are stored in order, since all of them are introduced one by one and one per cycle to the rotation core. PD matrix elements come first, then PND elements and, finally, the PV elements are stored. Only PD matrices have to be introduced to the angle calculation core in Stage I. The memory organization is shown in detail in Appendix VI. Following this organization within the memories, the Distribution block (see Figure 43) can easily access the different processors using just one counter. The Distribution block uses a reading block that compensates the kind of crossed reading of each processor. The crossed reading depends on its position within the matrix as Figure 44 illustrates. This internal reading block is shown in Appendix VI.

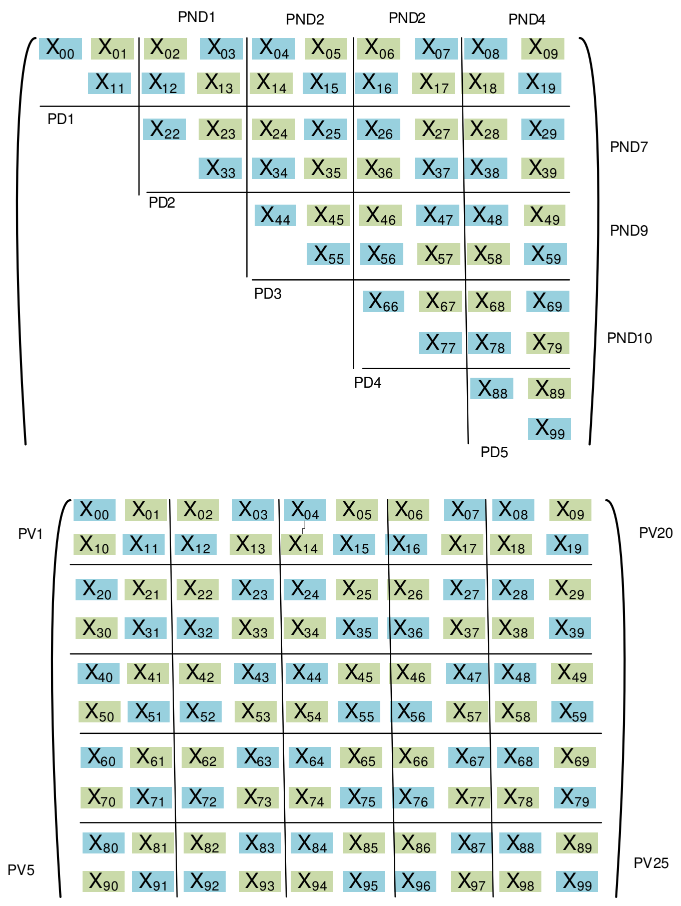


Figure 44: Eigenvalue and eigenvector memory splitting

In summary, each eigenvalue matrix consists of 60 elements (upper triangular matrix of the correlation matrix of 10 x 10) and the eigenvector matrix is composed of 100 elements. Thus, each memory block, Memory A and Memory B, will have 80 positions in total because 30 positions are for eigenvalues and 50 for eigenvectors. Each Block RAM on the FPGA has a length of 1024 positions. Therefore, using this same organization, we can compute matrices up to 25x25 elements without using additional Block RAMs.

7.4.2 **Rotator block**

This block has to carry out the rotation of the processors using the corresponding angle. CORDIC (Coordinate Rotation Digital Computer) blocks were proposed by [Cavallaro (1987)] to perform the angle calculation and rotations within a Brent's architecture. CORDIC is a simple and efficient algorithm to calculate trigonometric functions, in fixed-point precision, through an iterative process based on small steps that only use addition, subtraction, bit shift, and look-up-tables (LUT) [Volder(1959)]. In addition, CORDIC blocks can also be used for resolving rotations. The use of pipelined CORDIC block in the SVD computation was introduced by [Bravo (2006)].

Vector rotations are primitive CORDIC operations. The diagonalization of each processor can be performed by treating each processor as a pair of vectors and using the rotation angles to transform the processor [Cavallaro (1987)]. Then, two CORDIC blocks are used to carry out the two vector rotations of each processor in parallel. They called this structure two-sided CORDIC. [Cavallaro (1987)] uses two two-sided CORDIC (four CORDIC blocks) for the double rotation necessary in the PD and PND processors and one two-side CORDIC in the single rotation of the PV processors.

[Bravo (2006)] also uses two two-sided CORDIC but in a pipeline way and one of them has been modified for computing the angle calculation. This is possible because, as mentioned in Section 7.3, he does not carry out the angle calculation stage in parallel with the rotation stage and he configures the two-sided CORDIC for angle calculation or rotation.

CORDIC is extremely useful when no hardware multipliers are available. Note that Cavallaro's proposal is almost thirty year old. However, the Virtex family has embedded DSP cores which are able of implementing multiplication in an optimal manner. Bearing in mind this fact, and in an attempt to find the rotation core that consumes less hardware resources, we propose to use a particular implementation of vector rotation on FPGA better than using CORDIC blocks.

Our implementation lies on that rotation of the vector (*X, Y*) by an angle *α*:

$$\begin{aligned} X' &= X\cos \propto \ - Y\sin \propto, \\ Y' &= X\sin \propto \ + Y\cos \propto. \end{aligned} \tag{5}$$

That equation can be solved on the FPGA through floating-point arithmetic operations. The basic idea is to extract the trigonometric operations of each vector rotation calculation, which will be computed previously to the rotation. That significantly reduces the amount of trigonometric operations. In fact, the *sin(α)* and *cos(α)* operations should be calculated in all the rotations (55 in total, two for each PD and PND), but using our proposal they have only to be computed as many times as angles (5 in total, one for each PD). Then, the angles are now used for calculating the trigonometric operations directly instead of being used as rotation angles like in [Cavallaro (1987), Bravo (2006)]. This proposal of a simplified vector rotation core is illustrated in Figure 45, where *sin(α)* and *cos(α)* are explicitly received instead of *α*.

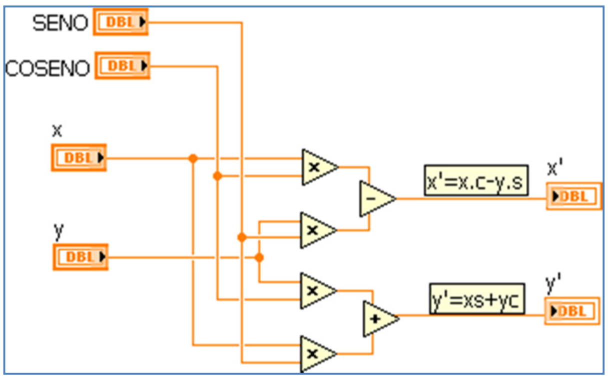


Figure 45 Simplified vector rotation based on arithmetic operations

For computing the trigonometric operations, a CORDIC block has been introduced in the Angle Calculation block to use the angles at the same time they are produced (see Section 7.4.3). This new CORDIC block is able of computing the sine and cosine at the same time. The Angle Calculation block does not return an angle but trigonometric values. Later, in the *angle transmission stage*, *sin(α)* and *cos(α)* are transmitted directly instead of *α*. This transmission is done by using the intermediate memory Angle Memo (see Figure 41). Note that, despite this new functionality, we have kept the name of these blocks as Angle Calculation block and Angle Memo block.

To summarize, the Rotator block is composed of two simplified rotation blocks (see Figure 45), which substitute the two CORDIC blocks within the two-sided CORDIC block of [Cavallaro (1987)] and [Bravo (2006)].

Figure 46 clearly indicates that the resource occupation in our proposal is smaller than that in Cavallaro's. In our simplified version we have added the resources occupation of the new rotator core and the trigonometric CORDIC core for understanding how the new strategy affects the total amount of resource consumption. The differences are not very significant in favor of the simplified version since they only represent around 2 % of the total FPGA logic resources (Flip-Flop and LUTs). However, the new simplified rotation core does not expand in resource occupation as faster as the CORDIC-based one. Our proposal is also more efficient as far as a possible precision increase is concerned. The cases for 27 bit and 32 bit precision are displayed. Differences between the two methods are only about 2 % for the first case but increase up to 5% in the second case. The reason for the high resource occupation of these blocks lies in that they are in a pipelined implementation, as we commented on before. This justifies our decision of using a simplified Rotator block instead of a CORDIC-based one like in [Bravo (2006)].

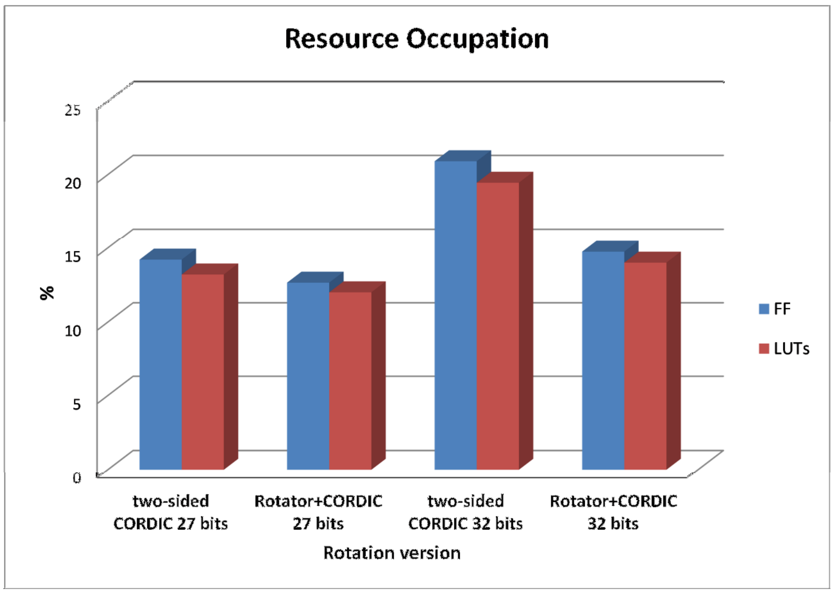


Figure 46 Two-sided CORDIC rotator vs. simplified Rotator block plus the trigonometric CORDIC. Comparison of resource occupations according to the number of bits

Introducing a CORDIC core in the Angle Calculation block for releasing another similar block from the Rotator block could seem a paradox. But, apart from saving hardware resources as we have argued, there are other advantages of using our simplified rotation. The new rotator version admits more freedom upon adjusting the final precision. Its resource occupation growth is not as dramatic in proportion to the target precision. Thanks to working in floating point precision, which provides a wider dynamic range than fixed point precision, we can partially avoid the rounding errors that are usually introduced by a CORDIC rotator, as was documented in [Bravo (2006)]. These rounding errors are due to the limit that its fixed-point precision can reach. In fixed-point operations the relative errors are always equal or greater than those using floating point operations, under the same conditions in number of bits [Kahan (1987)].

The proposed simplified Rotator block, occupies around a 5 % of the total FPGA resources (Flip-Flop and LUTs). On the other hand, rotations of eigenvalue and eigenvector matrices can be performed in parallel. Thus, we can introduce a second Rotator block for performing these rotations in parallel with a hardware cost of around a 5 %. Introducing another two-sided CORDIC in the architecture by Bravo means a hardware cost of around a 14 %. We can conclude that our SVD proposal provides a higher theoretical scalability than Bravo's.

To take the Rotator block to its maximum performance, the rotation process is in a pipeline, that is, when the rotation starts and we provide it with inputs, it takes a number of cycles in yielding the rotated data (initial latency). Fortunately, thanks to the pipelined implementation, we do not have to wait until the rotator has finished each rotation. That would be too slow and not efficient. Rather, we give a new input to the rotator block in each clock cycle. The result is such that when rotation is finished after a latency period, we have a new result every clock cycle.

To compensate for having a unique Rotator block and the initial latency we have attempted to introduce the processors as soon as they are available. We start the first rotation of PD and PND processors (eigenvalue matrix). When all the eigenvalue data are already given to the Rotator, we start with the unique rotation of PV processors (eigenvector matrix). When the result of the first rotation of eigenvalues is ready, we can go with the second rotation for them if all the eigenvectors have been rotated. Thus, the Rotator block needs a data feedback as is illustrated in Figure 41.

Each processor has to be rotated by the angle that corresponds to its position. To choose the right sine and cosine values from the Angle Memo memory, according to the proposed memory distribution, the Control Unit uses a table which codifies the corresponding angle for each processor position. This table is shown in Appendix VI.

7.4.3 The Angle calculation block

The other main task in the SVD process is the angle calculation for the PD processor. So, this block is intended to do the angle calculation expressed in Equation (1) as well as to calculate the sine and the cosine of each angle. This block is illustrated in Figure 47.

As commented on in Section 7.4.2, we are using floating point operations in the Rotator block in the same way that memories and buses work. Then, the only part that works in fixed-point precision is the CORDIC blocks in the Angle calculation block. However, the inputs to the Angle calculation block, PD processor, are data in floating point representation. As seen in Figure 47, the inputs are operated also in floating point to calculate the difference of the PD processor diagonal, as Equation (1) requires.

CORDIC blocks work in fixed point and a format conversion is necessary. Hence, the block *flp2fxp* performs this conversion. The fixed point format uses 27 bits of precision. As is shown in Figure 47, the necessary multiplication by 2 is performed after the conversion using a shift-register of 1 bit to the left.

The arctangent of a division is necessary in Equation (1). To implement this arctangent operation we use a CORDIC core synthesized for arctangent(y/x) solving. In this manner, we do not have to carry out the division separately. This operation is reflected in Figure 47 as *ATAN2_CORDIC(Y, X)*. This CORDIC block actually implements the well-known *atan2* function. Later, a quadrant translation is necessary to get the desired angle. It is carried out in the *Quadrant_Translation* block. The last step to complete the computation in Equation (1) is a division by 2. This is performed through a shift-register of 1 bit to the right.

As we have argued in the last subsection, another CORDIC block was introduced for the trigonometric calculations, which is called *SIN_COS*_CORDIC. It calculates the sine and cosine in fixed-point precision. The outputs of this core are immediately converted to 32 bit floating point data, in the *fxp2flp* block, in order to be stored in the Angle Memo block.

CORDIC cores are Intellectual Property (IP) cores by Xilinx and they have been generated using the Xilinx Core Generator tool [Xilinx]. More details about implementation of these specific cores are given in Appendix VI.

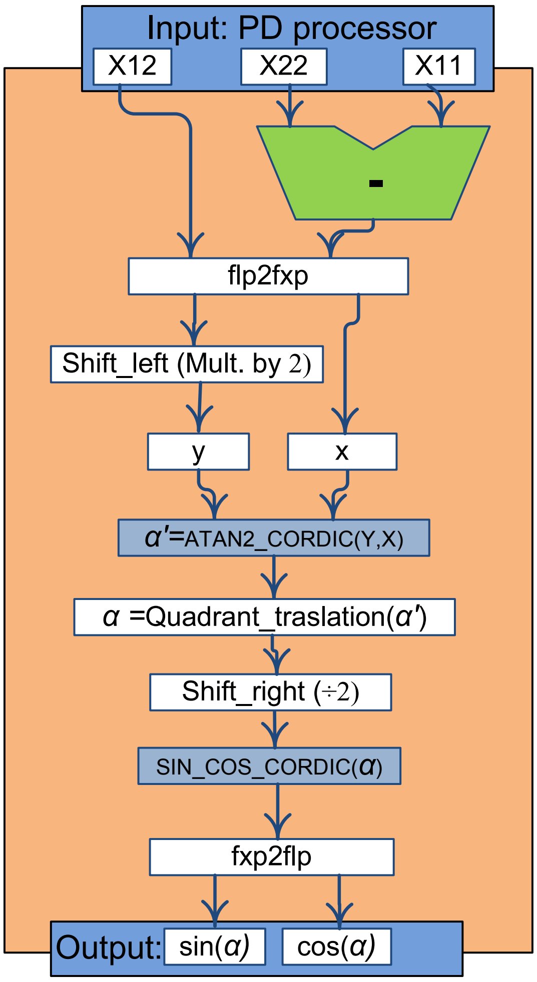


Figure 47 Angle Calculation block

7.4.4 Reallocate block

As Figure 41 illustrates, the Reallocate block gathers the outputs from the Rotator block and it is in charge of writing them in the Eigenvalue and Eigenvector Memories (EEM). During the writing process the re-sorting of elements is carried out.

The Rotator output is composed of four elements (a rotated processor) and the EEM is designed for writing these four values in the same clock cycle. The Reallocate block contains a ROM memory inside that codifies the write address for each element. This ROM is prepared according to the re-sorting stage and the memory organization explained in Section 7.4.1. A table with the ROM contents is given in Appendix VI.

7.5 Results

In this section we are going to outline the most important results of the SVD proposal as the reached precision, the speed performance and hardware resource occupation. We discuss the architecture scalability as well.

It is important to remark that the SVD design presented in this thesis was implemented and tested by [Aparicio (2013)]. In addition, specific studies and tests about the CORDIC blocks were presented in [Ramos], who has provided several software tools for debugging and testing the SVD architecture. However, these tools are currently unpublished.

7.5.1 Hardware resource occupation

In Table 14, the hardware resource occupation of the proposed architecture is shown. The required precision imposes a high occupation of resources. Most of the occupied elements are due to the CORDIC blocks since this implementation needs a lot of LUTs and Flip-Flops, as was discussed in last subsection. The DSP elements are used in the floating point operations within the Rotator block and the subtraction operation in the Angle Calculation block.

The maximum frequency of the SVD subsystem is limited by the CORDIC arctangent. It forced us to have two different clock domains within the RTE inverter since the rest of the RTE inverter works at 150 MHz.

Resources	Occupatio	%
Occupied Slices	10,754	43%
LUTs	16,266	33%
FFs	14,735	29%
BRAM (18kb)	9	2%
DSP48	60	11%
Maximum	116 MHz	

Table 14 SVD architecture resource occupation

7.5.2 **Precision**

As we mentioned in 7.1, a target average precision of 5x10^{-8} was decided after simulations of the RTE inversion using the SVD algorithm coded in C.

To test that the SVD is able of reaching this precision, we carried out the SVD on FPGA over 10^{5} correlation matrices obtained from an execution of the RTE inversion algorithm using MELANIE profiles. Then, we compared the eigenvalue results from the FPGA with the ones from the SVD algorithm in C in single floating point precision. The average root-mean-square differences between the obtained eigenvalues are shown in Table 15.

SVD Iterations	Average root-mean-square differences
9	2,61 x 10^{-8}
18	2,63 x 10^{-8}
27	2,63 x 10^{-8}

Table 15 Root-mean-square differences in the computed eigenvalues (FPGA vs C)

As we mentioned in Section 7.2, the study in [Brent (1983)] about the Jacobi iterative method concluded that the SVD solution is reached approximately after *Mxlog(M)* iterations, where *M* is one dimension of the square input matrix. On the other hand, the eigenvalue matrix elements are re-sorted in the Brent's algorithm after every iteration and the position of the final eigenvalues are exchanged along the diagonal. After *M-1* iterations the elements in the diagonal are sorted according to their initial order. Thus, we have restricted the number of iterations to multiples of this number, that is, to 9, 18, and 27 iterations but alternative numbers can be used.

Table 15 shows that the target average precision has been reached. Also, the FPGA results have the same behavior than those in the C version along the number of iterations because the differences are constant independently of that number.

This test has verified that the SVD block works properly. However, to compare how well the initial eigenvalue matrix is diagonalized and how well the eigenvectors are computed, we carried out another test based on the fulfilment of Equation (6):

$$\mathbf{A}\overrightarrow{x_k} = v_k\overrightarrow{x_k}, \tag{6}$$

where **A** is the input matrix to the SVD, $\overrightarrow{x_k}$ stands for any of the 10 calculated eigenvectors and v_k its corresponding eigenvalue. We can compute the quadratic differences, χ^2. The value of χ^2 indicates the goodness of the eigenvalue problem solution.

$$\chi^2 = \frac{\Sigma\,|\mathbf{A}\overrightarrow{x_k} - v_k\overrightarrow{x_k}|^2}{M \times M} \tag{7}$$

Table 16 shows the results from computing χ^2 for both SVD methods, C-MILOS and FPGA. For 9 iterations, the FPGA version is able of diagonalizing the input matrices as good as the C-MILOS version. This is not the case for 18 and 27 iterations, however, where C-MILOS reaches better results. The differences can be explained by the higher precision that the C version reaches in the computation of the arctangent and sine and cosine functions. This is indeed the only implementation difference. Remember that the FPGA uses a fixed point CORDIC approximation to the trigonometric functions, 27 bits specifically. This implies that elements in the eigenvalue matrix can be neglected on the FPGA if they are smaller than the used precision by CORDIC.

SVD Iterations	**Average χ^2**	
	C SVD	**FPGA SVD**
9	9.04×10^{-9}	9.04×10^{-9}
18	9.79×10^{-16}	3.57×10^{-15}
27	4.82×10^{-16}	3.18×10^{-15}

Table 16 Average χ^2 for C and FPGA versions

In any case, according to these results we conclude that the FPGA SVD architecture diagonalizes the input matrix correctly and provides root-mean-square differences of the order that we have estimated adequate. The final validation will be carried out using this architecture within the RTE inverter in order to know whether the reached precision is good enough for SO/PHI. This test will be discussed in Chapter 9.

7.5.3 Time performance

As is well known, the execution time of a pipeline system is computed taking into account the number of pipelined stages and the duration of the longest one. We have designed a pipelined SVD architecture where the two main stages are the Angle Stage and the Rotation Stage. In this section we are going to discuss the time performance of the proposed SVD architecture taking into account that is focused to 10 x 10 matrices. So, we calculate the time for this size of matrix first and speak about the architecture scalability later.

The Rotation Stage duration is conditioned by the floating-point operation trees within each vector rotator. Taking into account a latency of 18 cycles in the Rotator block, the 55 rotations performed in total (double rotations in PD and PND, and single in PV processors), and a latency in the Reallocate block of 6 cycles, the total duration of this Stage is 79 cycles.

On the other hand, the duration of executing the Angle Stage is mainly conditioned by the two CORDIC blocks and the floating-point subtraction within the Angle Calculation block. The duration of both blocks is proportional to the target precision since CORDIC performs as many iterations as those required for the desired precision. This final duration is of 30 cycles for each block. Then, the Angle Stage takes 75 cycles.

We can then compute the total duration (in cycles) of the SVD execution, T_{SVD}, as

$$T_{SVD} = \left(L_{ini} + T_{max}\,S\,(N_{iter} + 1) + L_{fin}\right)Clk_{period}, \tag{8}$$

where L_{ini} is an initial latency of 60 cycles in which an initial eigenvalue matrix is introduced to the SVD architecture, L_{fin} is the final latency of 110 cycles for releasing the eigenvalue and the eigenvector results, T_{max} is the longest duration of all the Stages (79 cycles), S is the number of the Stages in the pipelined architecture (2), N_{iter} is the number of iterations in the SVD algorithm, and *Clk_period* is the inverse of the clock frequency in the system. Note that the number of iterations is increased in one for the latencies produced for filling and emptying the two main stages to be taken into account: at the beginning of the execution only the first matrix (M_A) is inside the Angle Stage and the Rotation Stage is empty; finally, the Rotation Stage is occupied by the second matrix (M_B) and the Angle Stage is empty.

The maximum frequency at which the SVD block is able to work is limited by the arctangent CORDIC block which is around 116 MHZ. We are using a frequency of 110 MHz in this test and in the final RTE inverter.

Times obtained applying Equation (8) are shown in Table 17 for different number of iterations. Note that the times in this table are for each SVD execution, which indeed computes the SVD for two matrices at a time. As was mentioned in 7.1, the time requirement for the SVD within the RTE is 14.3 µs per matrix. Thus, we can conclude that only the cases with 9 and 18 iterations can be used for reaching that requirement. Their times are 7.38 and 13.77µs per matrix, respectively.

	Iterations		
	9	18	27
2 matrices of 10x10	14.76 µs	27.55 µs	40.35 µs

Table 17 Execution time of the SVD (2 matrices)

When compared to Bravo's performance [Bravo (2006)], whose architecture takes 16.53 μs using 19 iterations for a 10 x 10 matrix, we can see that our proposal improves the execution time when using either 9 or 18 iterations. The root-mean-square differences that are reached in [Bravo (2006)] are around 4×10^{-6} while we have reached root-mean-square differences around two orders of magnitude better.

7.5.4 About scalability and improvements

The SVD architecture is able of computing the singular value decomposition of matrices with a size up to 25 x 25 without introducing any important change. Only more Block RAMs for larger matrices would be necessary but this is almost direct as well. The main change for adapting the SVD proposal to other matrix sizes is to carry out a configuration of the memory address space and the reading and re-allocation tables.

On the other hand, a second Rotator block can be introduced for improving the time performance at the expense of around 5% of the FPGA resources (see Section 7.4). This could be interesting because the Rotation stage is the longest one. Additionally, T_{SVD} depends on the maximum duration time, T_{max}, which in turn depends on the size of the computed matrix. If we consider using different matrix sizes, the Rotation Stage will increase its duration proportionally since the number of sub-matrices grows. The Angle Stage duration is less affected by the matrix growth since it only processes the diagonal processors (PD) instead of the Rotation Stage which has to rotate all the processors (PD, PND and PV).

The final SVD times using one Rotator and two Rotators are compared in Figure 48, where T_{SVD} is plotted as a function of the matrix dimension. In that comparison the number of iterations is 9. The improvement, however, can be assumed similar for other number of iterations, as the time for different iterations is proportional to N_{iter} .

Contrary to expectations, the SVD execution time with two rotators is almost constant until the matrix reaches a 14 x 14 dimension. It is even longer than that with only one rotator for matrices smaller than approximately 10 x 10. The explanation is to be found in the Angle Calculation Stage, which becomes the longest stage for matrices of these sizes. It conditions T_{max} and, thus, the total time in Equation (8).

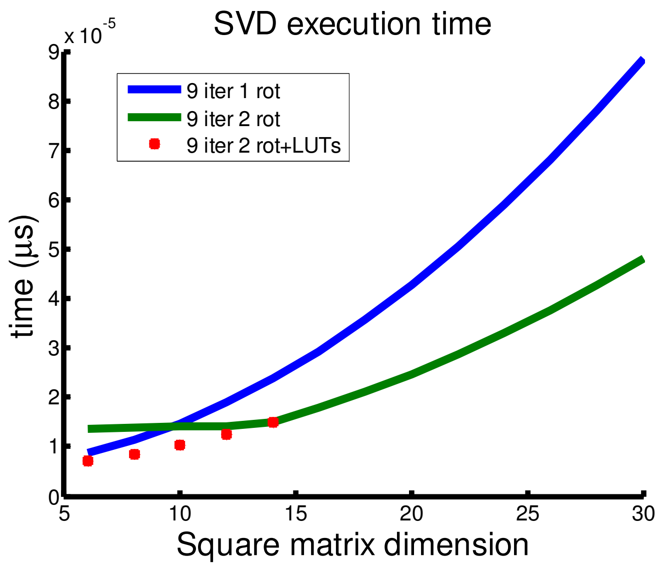


Figure 48 SVD time execution for different matrix sizes

Therefore, the only means for improving the execution time for matrices of size less than 14 x 14 with 2 rotators is to reduce the time for the Angle Calculation block. To reach this improvement, the best option is to implement the arctangent and/or the sine and cosine blocks using Look-Up-Tables (LUTs). These operations are usually implemented through a LUT-based method for improving time and space on FPGA [Florent (2005)]. Then, using two rotators and introducing a sine and cosine block in the Angle Calculation block with a shorter latency of about 8-10 cycles, an improvement is reached, as shown with red dots in Figure 48. This improvement, however, has not been implemented in the final system because the time performance already fulfilled the requirements.

8

The RTE inverter aboard SOPHI

Until now, we have omitted some information about the communication with the DPU and the error mitigation in the radiative space environment. In this chapter we discuss these important issues in the final SIMD architecture. We will also give some more details about the integration of the RTE inverter within the DPU from a double point of view, namely, the hardware and software interfaces.

8.1 Fault mitigation, detection, and correction on the FPGA

The radiation environment in which *Solar Orbiter* will orbit has been studied in [Michel (2013)], where authors have confirmed that the Virtex-4 are out of permanent and non-reversible damages, since the Total Ionizing Dose Effect requirement are completely fulfilled. In that work, the fabrication of these FPGAs is summarized with emphasis in the fact that no latch-up effects occur because of any radiation effect. The testing process is described in [Swift (2008)]. According to these authors, the only radiation effect to take into account for our FPGA is the Single Events Upset (SEU) induced by heavy ion and protons.

Under normal solar quiet conditions the upset and resulting error rates were estimated to keep inside the reasonable limits allowing a normal operation. In the specific case when Virtex-4 FPGAs are used only for processing, these quiet conditions permitted an unmitigated design [Michel (2013)]. However when flare conditions appear, data processing cannot be allowed with an unmitigated design. Since data are stored elsewhere, they can be processed when conditions permit.

The Xilinx test consortium has identified a list of functional interrupts which can be triggered by SEUs [Allen (2008)]. These Single Event Functional Interrupts (SEFI) are caused by SEUs within control logic elements. This kind of errors will not be discussed here because of the Virtex-4 FPGA does not have leeway for action against these errors, but the System Supervisor (RTAX FPGA in Figure 5) has to take the control in these cases. It usually will have to reset and reconfigure the FPGA to recover from these functional interrupts. The SEFI cases are seldom [Allen (2008)] and their impact in the system functionality is insignificant as pointed out in [Michel (2013)].

The radiation tolerant Virtex-4 series FPGAs have been tested for single event upset by [Swift (2008); Allen (2008)]. SEUs can affect basically the main elements on the FPGA: configuration cells, block of memory elements (BRAM), and distributed memory (flips flops, FF). Taking into account the estimations by [Michel (2013)], in quite solar conditions, less than five SEUs to configuration cells and BRAM per day are expected for *Solar Orbiter*. Around two SEUs per month are expected to affect the FF. In addition, if all resources in an FPGA were used, only about 15 % of configuration cell upsets lead to an incorrect behavior, as a worst-case scenario [Fuller (2000)]. Even more, the System Supervisor FPGA contains a JTAG interface to perform a configuration memory scrubbing, that is, the process of detecting and correcting upsets in configuration memory. It is running cyclically and each cycle takes around 10 seconds for the entire FPGA.

In short, the RTE inversion could be carried out without any fault mitigation strategy, where the System Supervisor simply monitors and controls the Xilinx FPGA. This would be, however, an unwise alternative. The final strategy to perform is not still decided, and it will be chosen depending on the DPU availability for attending and controlling the information about errors that the RTE inverter provides. However, we are working on some techniques to improve the fault mitigation capabilities of our system. In the next subsections, we propose two fault mitigation strategies, taking always into account the high resources occupation on the FPGA. In the last subsection we point out some improvements that could convert the SIMD architecture in a fault tolerant one.

8.1.1 Optimistic fault mitigation strategy

In the most optimistic case, the System Supervisor is the only responsible for the data and results integrity. It will send the entire image of 4 mega-spatial-pixels to the Virtex-4, row by row, in a burst mode (hereafter, we refer spatial pixel simply as pixel, but remember that a spatial pixel is composed by 24 pixels from 24 different images). After each row, it will send a pattern of 12 known pixels, one for each nProcessor. If the result of processing these 12 pixels matches with the previously off-line calculated pattern result, then the System Supervisor will assume that the entire row was well computed and will send another one. In the opposite case, the Virtex-4 will be reconfigured and the entire row will be recalculated. The same procedure could be performed using one half of a row.

The System Supervisor will reconfigure the FPGA and restart the row in course, in the presence of some evidence of error, like timeouts, error in the number of results received, or errors in a communication packet through the SoCWire bus.

In this strategy, the configuration memory scrubbing will be disabled because it could correct an error, and then, the System Supervisor would not be able to detect any error in the previous pixels using the pattern. Other alternative would be to reconfigure all the FPGA every time the scrubbing detects an upset.

This strategy could be a good candidate if the SEU rate coincides with the estimated one. However, if the SEU exceeds the estimations, the number of repeated pixels could result in an important overload.

8.1.2 **Fault mitigation strategy with FPGA error detection and correction**

In this strategy we try to be fault tolerant and to give more information about the RTE inversion process to the System Supervisor, but introducing the minimum hardware elements as possible, and only monitoring the nProcessors by software.

We propose to add a label to the head of each pixel with an identifier. The nProcessor adds a label with two fields to each result: the original PIXEL identifier and a well-processed mark, where the nProcessor indicates if the result is correct. The root mean square error (RMSE) between the observed and the synthetic profiles is a good proxy for the fitting quality. If the final RMSE is under a fixed threshold the pixel is considered to be well computed.

Using this information exchange the System Supervisor can confirm pixel by pixel, instead of "row by row" as in the optimistic case. The System Supervisor can make decisions about the RTE inversion process depending on the error cadence. In this strategy, the SIMD architecture can also make decisions about its internal performance and even correct internal errors by itself.

The SEUs affect the FPGA elements, and this is translated into specific errors in the SIMD architecture. In Table 18 the main blocks on the FPGA, the type of FPGA elements used, and how SEUs can affect the internal performance are summarized. How all the errors can be detected and the architecture corrected is also shown. Basically, there are two main evidences for detecting errors: good results (that indicates that the algorithm is converging) and good communications operation between the two FPGAs within the DPU (see Section 1.5).

The error detection and correction is organized in two levels, an External Recovery Protocol from the System Supervisor point of view, and an Internal Recovery Protocol from the SIMD architecture point of view. Each one of these protocols is able to detect errors in the results and correct the architecture in order to avoid new errors. Each protocol acts over different blocks of the FPGA as is indicated in Table 18.

In addition, the FPGA will make use of a configuration register, provided by the SoCWire Bus (see Section 8.2), for communicating its internal status to the System Controller. In this register, the FPGA gives information about its proper functioning (OK) or, in the contrary, it can ask for a reprogramming. Furthermore, the FPGA reports about where the error was: Net Control Unit, SOB, or nP. This information is useful for future evaluations.

The External Recovery Protocol, from the System Controller point of view, consists on:

- The System Controller is typically sending bursts of pixels and confirming the results through a checking in its header. There is a timeout alert. If the FPGA does not reply in a given time interval, it has to be reconfigured.

- If the header indicates that a pixel is not well computed, that pixel is resent again to the FPGA, only if the FPGA status is OK. In this way, if there was a temporal error, or the error was internally corrected, the pixel will be well computed. Each pixel is only sent twice, if it is returned as a wrong result, the wrong result is stored.

This step can be improved for the case where there are noisy input images (with dead pixels, salt-and-pepper noise, etc.). If the System Supervisor detects that there is a high number of pixels with wrong results (e.g. more than 3% or 5% per line), then it will disable the resent of pixels. In this way, the overload of computing repeated pixels is eliminated. Nevertheless, the instrument will be able to have a statistics about the RTE inversion process and data acquisition.

- If a result with an unformatted header is received, that result will be eliminated. The System Controller could know which pixels are affected using the reception order as indicator.

- If a burst of wrong results (for instance more than 30 pixels) is received, the FPGA must be reconfigured. Note that if the FPGA status register is OK, it could mean that the input data are not well calibrated, or the original images are corrupted. Then, using a pattern, like in the optimistic case, the System Controller can conclude about that. This important report, which gives us information about the images, is not possible to be achieved using the optimistic strategy.

Using the External Recovery Protocol, temporal errors within the SoCWire and Input/Output Buffers FIFOs will be corrected. In general, the protocol provides a mechanism for recovering from situations where communications have been interrupted or when the FPGA is having a wrong behavior.

As mentioned in Section 5.2, the nProcessors and the Shared Operation Block occupy almost the 90 % of the FPGA. All the error affecting these two blocks can be detected and corrected if needed by an Internal Recovery Protocol. This is explicit in Table 18.

The Internal Recovery Protocol consists on:

- The nProcessor carries out the RTE inversion for the incoming pixel and marks the result according to the calculated RMSE. When one nProcessor marks the header as a wrong result, then an error flag is activated in its nProcessor status register. This register will be checked by the Net Control Unit.

- The nProcessor carries out a short software check after processing every pixel. This check is nothing but a few operations using its constants data in memory. If the check fails, first the nProcessor is reset, and then the constant data from a reserved memory positions are re-written. Then they repeat the software check. If the software check still fails, the FPGA asks for a reconfiguration to the System Supervisor. This process is able to correct temporal errors in the internal elements of the nProcessor as Flip Flop or BRAM or it can finish in a total FPGA reconfiguration.

- If several error flags are activated in the nProcessors, then the Net Control Unit, besides of performing the software check in the nProcessors, will check the Shared Operation Block. This check consists in resetting the SOB and executing an instruction set in the nProcessors that does use of the SOB. This check is able to correct temporal errors in the SOB. If the check fails, the FPGA asks for a reconfiguration.

- If the watchdog detects an error, it interrupts the Net Control Unit operation, and asks for a reconfiguration to the System Supervisor.

- The Instruction ROM can be equipped with an Error Detection and Correction Code (EDAC) technique. Currently we are considering using a BCH codification able to correct errors in 2 bits [Bose (1960)]. Considerations about the hardware resources that are spent by this technique are given in [Cobos (2015)].

Applying the Internal Recovery Protocol, almost all the errors in the nProcessor and the SOB, within BRAM or FF, are detected and the architecture is recovered. In BRAM within the SVD block this is not possible, since that BRAM does not have any protection mechanism so far.

Block	FPGA Element	Internal Affected Block	Effect	Detection	Solution
ALL FPGA	Entired device by SEFI	Configuration line	Functional Interrupts	System Supervisor	Reconfiguration
SoCWire	BRAM	FIFOs	Input or Output Data corruption	System Supervisor or nPs: no algorithm convergence	PIXEL Resending
	FF	Control State Machine	Communication interruption	System Supervisor	External Recovery Protocol
	Conf. Cell			Configuration Memory Scrubbing	Scrubbing
Input/Output Buffers	BRAM	FIFOs	Input or Output Data corruption	nPs: no algorithm convergence	PIXEL Resending
Net Control Unit	BRAM	Instruction ROM	Less or 2 bits / word	EDAC	EDAC correction
			More than 2 / word	Watchdog nPs: no algorithm convergence	Internal Recovery Protocol: Reconfiguration
	FF	Control State Machine	Net Control Unit Wrong Operation	Watchdog nPs: no algorithm convergence	Internal Recovery Protocol: Reconfiguration
	Conf. Cell			Configuration Memory Scrubbing	Scrubbing
nP	BRAM	nP Caches	Internal data corruption	nPs: no algorithm convergence	Internal Recovery Protocol: Corrected
	FF	Local Control Unit	nP wrong operation		
	Conf. Cell			Configuration Memory Scrubbing	Scrubbing
SOB	BRAM	SVD internal Reallocation Memory	SOB cores with wrong operation	nPs: no convergence	Internal Recovery Protocol: Corrected
	FF	Control State Machine		nPs: no convergence	Internal Recovery Protocol: Corrected
	Conf. Cell			Configuration Memory Scrubbing	Scrubbing

Table 18 Summary of the main blocks in the architecture and the error detection and correction in each of them

The configuration memory scrubbing is continuously checking for correcting errors on the FPGA. In any case the recovery protocols are able to detect errors in the behavior before the scrubbing could do it. The scrubbing cycle is around 10 seconds, while the recovery protocol is working with a time resolution of milliseconds. Nevertheless, the scrubbing is kept working in background and correcting errors in case it finds them.

All the exposed details supply the FPGA a robust operation behavior in the case of SEU induced errors. In addition, the FPGA will provide statistical information about the RTE inversion and the scientific data quality, and, even, about the robustness of the architecture.

Both strategies, optimistic and fault mitigation, are being tested with faults injection. That makes it possible to find which features have to be improved.

8.1.3 A Fault tolerant architecture

Taking into account the high resource occupation of the main blocks in the SIMD architecture as the SVD, it is impossible to suggest a fully Triple Modular Redundant (TMR) architecture. However, we have to explain how most of the errors in the SVD block and in the nProcessors can be detected and the architecture recovered using only software supervision.

It would be possible to have a hardened architecture version to be used in the worst radiation conditions or in most critical applications. For instance, we could take the nProcessors in sets of two or three of them. With a set of three and a voter that connect the three ALUs the architecture could correct errors during the data processing. With a set of two nProcessors, one voter could alert about fault or discard it. Taking into account the final use of the RTE inverter, obviously, we can afford this kind of strategies.

A totally fault tolerant mechanism could be used in the Instruction ROM by employing the memory scrubbing proposed in [Rollins (2010)]. For it, the use of BRAM would be triple plus an expensive control logic. Combining the fault tolerant Instruction ROM, a TMR'ed Control Unit and set of three nProcessor the reaching architecture would be enough fault tolerant for most applications.

8.2 Integration of the RTE inverter in the DPU

8.2.1 Hardware interfaces with the DPU

This section puts the RTE inverter in context within the Data Processing Unit (DPU). Hence, the Communication block within the general diagram of the RTE inverter is described here because itis the only means for data transfer between the two systems.

A detailed Communication block diagram is shown in Figure 49. It depicts its functional sub-system blocks. We can distinguish the SoCWire elements, the Communication Control block, and the RTE inverter Core Interface. This Communication block is, therefore, managing the connection between the RTE FPGA device with the Data Flash Memory (through the Actel RTAX FPGA) and the other Virtex FPGA (Xilinx FPGA #2) (see Figure 5 for details).

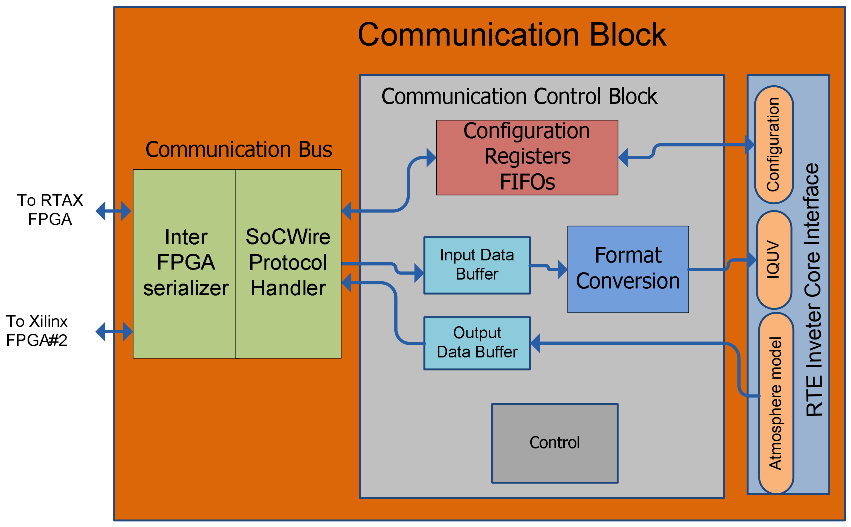


Figure 49 Communication block diagram

On the other hand, this communication block manages the input/output data to ensure an efficient work of the RTE inverter core. The Communication block is directly communicated with the nProcessor Net Control, through the RTE inverter Core Interface, to provide the necessary raw data (I,Q,U,V) and to reach the final model atmospheres to the DPU.

The RTE inverter Core Interface is nothing but a set of input and output registers of the RTE inverter core. The other two blocks are described in the next sub-sections.

8.2.1.1 The Communication Bus

The Communication Bus is based on SoCWire, which is a Network-on-Chip (NoC) approach based on the ESA SpaceWire interface standard to support dynamic partial reconfigurable System-on-Chip (SoC). It is developed by IDA and the whole information about this core can be found in [Fiethe (2010)]. SoCWire is an asynchronous communication, parallel data, bidirectional (full-duplex) interface including flow control, error detection and recovery in hardware, hot-plug ability and automatic reconnection after a link disconnection.

The protocol that controls SoCWire core is called SoCWire protocol (SoCP). The SoCWire access is done by using a serializer which saves some pins on the FPGA. The serializer-deserializer is transparent for the SoCP. The SoCP provides the RTE inverter Core with a full-duplex data communication link and, the SoCWire Protocol Write Registers that are used as configuration registers. These configuration registers are written by the DPU in order to configure the desired RTE inverter performance, as we will describe later. Also there are

SoCWire Protocol Read Registers which are used by the RTE inverter Core for communicating the RTE Internal Status Registers to the DPU.

As it is shown in Figure 49, the RTE FPGA has two connections with two DPU devices, namely, the RTAX FPGA and the Xilinx FPGA #2. With the connection to the RTAX FPGA, the device has access to pre-processed data ready to be used by the RTE inverter. The connection to the Xilinx FPGA #2 allows us to know the status of the data pre-processing and provides a control link on the operating status of the RTE inverter. This link can also be used for communicating data between both FPGAs.

The DPU generates a specific communication clock, defined at 50 MHz, and it also generates and controls a reset signal specifically for this communication core. The SoCWire bus width is defined in 16 bit but, thanks to the serializer, the FPGA only use 4 lines: three lines for control and another one for the clock. The internal SoCWire Protocol Registers are 16 bits wide.

8.2.1.2 The Communication Control Block

The most relevant functions in the Communication Control block are the following:

- Input data buffer for the RTE inverter core: it stores the input data into a First In First Out (FIFO) memory until the set of input data is complete. An input data set consists of one single spatial pixel from all the 24 images.
- Format conversion: it changes the representation of the data received from the DPU in fixed-point format into single precision floating point format, as Figure 4 illustrated. This is the operation format of the RTE inverter Core in order to fulfill the scientific requirements of the algorithm as we have seen in Chapter 2.
- Output data buffer for the RTE inverter core: it stores the result data from the inversion in a FIFO in order to send them to the DPU. The core output data are in single precision floating point format and this way they are sent to the DPU.
- The Configuration Registers FIFOs: it is composed of a full-duplex channel that reads the SoCWire Protocol Write Registers and introduces the values to the RTE inverter core using the configuration ports. And it also reads the RTE Internal Status Registers and writes them in the SoCWire Protocol Read Registers.
- Control: The Control block manages the input/output data FIFOs providing the data to SoCP and the RTE inverter. One of its main functions is to avoid the data overflow in these FIFOs by controlling the SoCP activity.

The Input/Output Data Buffers and the Configuration Registers FIFOs are stored in FIFO memories. These FIFOs are synthesized from double port Block RAM with independent writing and reading clock. The communication core data are in a clock domain at 50 Mhz and the RTE inverter Core's at 150 Mhz, so that these FIFOs perform a proper interface between these two clock domains. These FIFOs are generated by the Xilinx Core Generator software and they use BRAM resources.

8.2.2 The software interfaces with the DPU

The RTE inverter core programming structure is a pipeline structure. In every operation cycle, this core needs a complete set of data to be entered in this pipeline structure. It also needs to get rid of some results in order to be able to go on with its normal working pattern.

As it is a pipeline structure, there is a latency time while the core stream is filled with data. In this filling period, the RTE doesn't provide results. The RTE inverter core delivers the set of results in the same order as the set of operands was entered. This is an important point for the re-construction of the result images from the sets of results/pixels that the RTE inverter core delivered; anyway the pixel identifier makes it easier that task.

As it was deeply discussed in Section 5.3, the RTE inverter core starts after receiving a 12 input data set. It later receives sets in rows of 12 data. Remember that an input data set consists on one single spatial pixel from all the 24 images. After an initial latency, the RTE inverter Core starts to deliver its scientific results (model atmospheres).

So, the RTE inverter follows a clear master-slave process where the DPU acts as the master. To carry out the RTE inversion some parameters have to be defined by the DPU where the number of iterations and the specific scientific inversion are the most important. So far, we have spoken about the RTE inversion as the main target scientific problem to deal with but the RTE inverter Core also carries out other scientific calculations like the classical estimations and the no-polarization-modulation and longitudinal modes. These other calculations are much simpler than the RTE inversion and we are not going into details about them. However, the DPU has to indicate the RTE inverter core which calculation has to be carried out. This implies a change in the input pixel format and in the result one. Beside, the DPU has to monitor the RTE performance.

This bi-directional interaction between DPU and RTE is established using the SoCWire Protocol Write Registers, hereafter Configuration Register, and the SoCWire Protocol Read Registers, hereafter, Status Register. The description and use of each register is plenty of scientific details which are not important to be explained for understanding this thesis. An ample explanation about the use of this register and of the software protocol can be read in [Ramos (2015)]. In any case, in Appendix VII some details about the registers can be found.

Regarding the interaction protocol, the DPU can always know the RTE inverter internal status by accessing its Status Registers, and it can also change the RTE inverter performance writing on its Configuration registers. In summary, the RTE inverter can work in one of the following internal status: Waiting Configuration, RTE Ready, RTE Inverting, and Info Mode (in blue box in Figure 50). This figure details how to change the device state and which is the device behavior the DPU can expect in each state. The remaining annotations in the diagram are explained in the subsequent parenthesis.

The *Waiting Configuration* state is the internal state to which the RTE inverter accesses after carrying out the FPGA programmed or “Reset” process. In this state, the device is waiting to receive the configuration from the DPU. This programming is carried out by the DPU through writing in the RTE device Configuration Registers.

As the diagram in Figure 50 shows, if the device is in the *RTE Inverting state* it will not review the new configuration flag until it moves to the *RTE Ready* state which will only take place when the internal data stream is totally empty. Therefore, the RTE inverter configuration only will be attended in the *RTE Ready* and *Waiting configuration* states.

When the RTE inverter detects that the DPU has written a new configuration, it goes on analyzing such a configuration. The result of the analysis (configuration accepted or not) is written in the RTE Internal Status in order to inform the DPU. After this analysis, the RTE inverter moves to the *Info Mode* or *RTE Ready* state depending on whether the configuration was accepted. If the configuration is not accepted due to some error in the configuration process, the device will remain in the “*Waiting Configuration*” state and will inform the DPU.

Therefore, the DPU, after programming a new configuration, must check through the RTE Internal Status register that the device accepted it and that it was placed in the proper state (*RTE Ready* or *RTE Info Mode*) before sending the data.

Once in the *RTE Ready* state, the device has a valid configuration programmed and the input data stream is empty or the number of input data set is lower than 12. In this state, the RTE inverter is waiting to receive the data to start operation but the DPU can also change the configuration if desired. As soon as it detects a rising edge on the flag of new configuration has changed, the RTE inverter reads and processes the whole new configuration.

As Figure 50 illustrates, the *RTE Ready* state can be reached:

- from the *Waiting configuration* state after programming a valid configuration or
- from the *RTE Inverting* state after having calculated the results and having sent them to the DPU.

In the *RTE Inverting* state the RTE inverter is operating. These are the features of this state:

a) It can receive more input data from the DPU,
b) It is calculating, as it has data inside its internal operation stream and
c) The results are sent to the DPU as soon as they are processed.

When it has finished operating and sending all the results, the device will automatically move to the *RTE Ready* state.

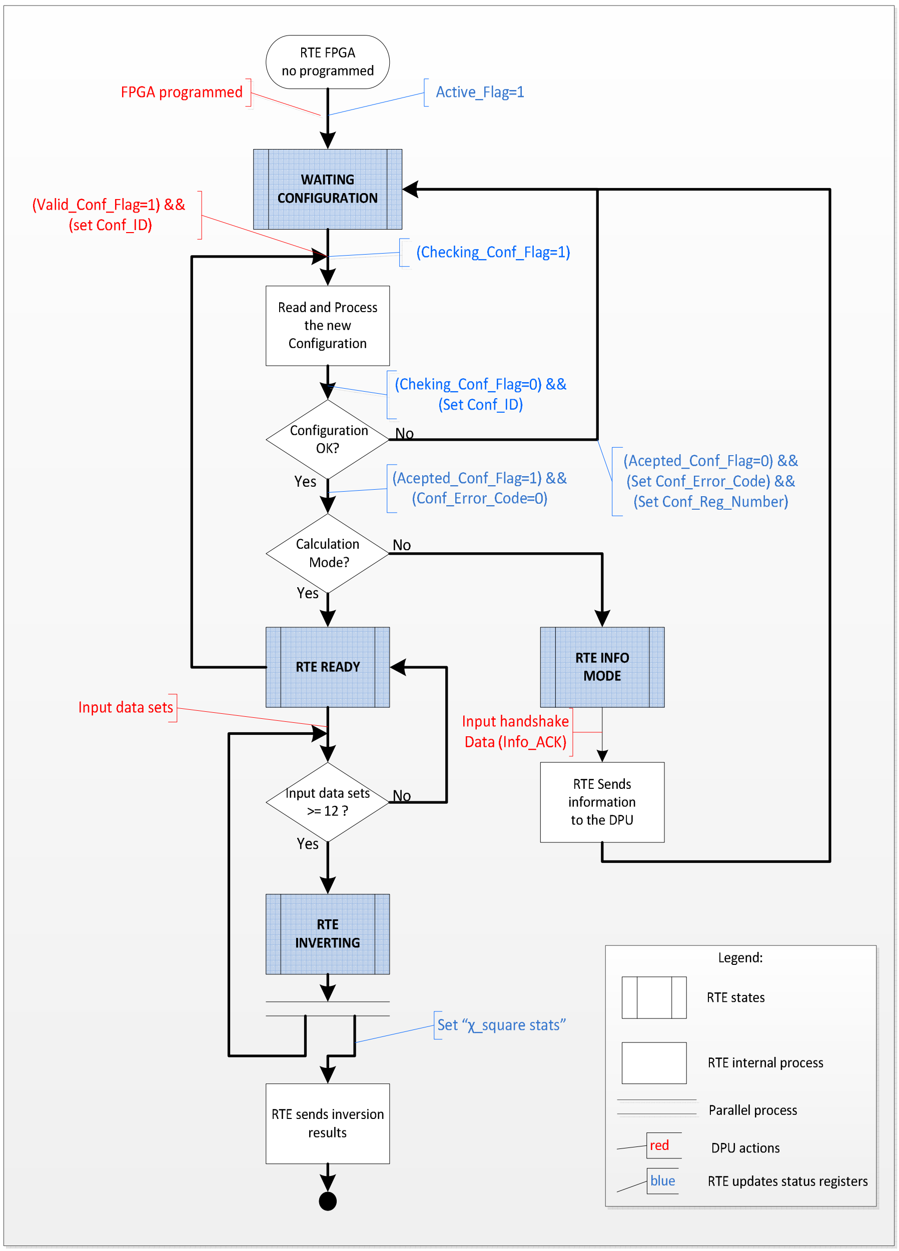

RTE FPGA no programmed
FPGA programmed
Active_Flag=1
WAITING CONFIGURATION
(Valid_Conf_Flag=1) && (set Conf_ID)
(Checking_Conf_Flag=1)
Read and Process the new Configuration
(Cheking_Conf_Flag=0) && (Set Conf_ID)
Configuration OK?
No
Yes
(Acepted_Conf_Flag=1) && (Conf_Error_Code=0)
(Acepted_Conf_Flag=0) && (Set Conf_Error_Code) && (Set Conf_Reg_Number)
Calculation Mode?
No
Yes
RTE READY
RTE INFO MODE
Input data sets
Input handshake Data (Info_ACK)
Input data sets >= 12 ?
No
Yes
RTE Sends information to the DPU
RTE INVERTING
Set "χ_square stats"
RTE sends inversion results
Legend:
RTE states
RTE internal process
Parallel process
red
DPU actions
blue
RTE updates status registers

Figure 50 RTE inverter internal states and interaction flow with the DPU.

In Figure 50 the "Parallel Process" symbol means that two processes always depart from the *RTE Inverting* state after an inversion is finished:

- One process is the reception of the input data which remains waiting the reception of the new 12 data packets in order to invert them.
- Another process is the delivery of results which is finished when all the results are sent to the DPU.

That is, these two processes take place every time an inversion is finished: one for reception in order to wait for additional data and another one for transmission that ends when all the results are sent.

Exiting *Inverting* state is carried out automatically (after operating and sending all the results) therefore the DPU might require knowing a "Time Out" for this state. While in the *RTE inverting* state, the DPU can re- program the device configuration, but this new configuration will not be analyzed until the processing of all the data is totally finished and, therefore, the device moves to the *RTE Ready* state.

After having finished the processing of all pixels in an entire image, the DPU can program the device in the *RTE Info mode*. This way, the DPU can get the statistical information produced during the data processing and indicate the RTE inverter that it is going to send a new image. Then the RTE inverter re-starts the internal statistical variables which are accumulative for the whole image.

As a good practice, the DPU should re-program all the configuration registers when it has received the "Info packet" in order to ensure the integrity of all the values in the registers (this process is carried out after processing the whole image).

9

Results

Two architecture proposals have been presented in Chapters 4 and 5 for carrying out the RTE inversion on FPGA aboard the SO/PHI instrument. As explained in those chapters, they are two different approaches that are inspired in two well established computational models. They introduce innovation in the high-performance embedded computing on FPGA field and we will show their main results in this chapter.

To demonstrate the valuable results of this thesis we are using two different points of view: one from the engineering requirement verification and another from the solar physics scientific verification. We will show how the main RTE Inverter requirements outlined in Table 2 are satisfied.

First, in the next section we are going to put the RTE inverter in context within the SO/PHI development status since, as a subsystem of the instrument, it is one of the main results of this thesis. In Section 9.2, we show how both computing architectures reach the computational power that the RTE inversion requires and, thus, how the time requirements are reached. The RTE inverter power consumption is discussed in Section 9.3.

Later, we explain how the RTE inversion on FPGA works in Section 9.4. This verification is done through a comparison of inverted scientific data with the FPGA and usual computers. This verification is only done for the SIMD architecture, which is the final design being embedded in the instrument DPU.

Other merits of this thesis like the scientific publications and oral presentations in the computer science field are mentioned in Appendix X.

9.1 The RTE inverter within the SO/PHI development status

The model philosophy followed by SO/PHI is [Meller (2013-B)]:

- A Breadboard (BB) Model was developed for some subsystems. It was aimed at having an early definition and verification of the internal interfaces. This model was not delivered to ESA.
- A Structural Thermal Model (STM) was used to qualify the PHI structure, to verify the mechanical and thermal interfaces at system level, and to validate the structural and thermal mathematical models. It was delivered to ESA on May, 2014 to support the spacecraft STM program.
- An Electrical and Functional Model (EFM) consisted on a set of modules with similar electrical behavior to the flight models. It was used to qualify the design in electrical and functional terms. This model is aimed to test the telecommands and telemetry interfaces to the spacecraft. It will also be used to test the loads at the spacecraft primary power interfaces.
- The EFM was delivered to ESA on September, 2015 to support the spacecraft EM program.
- An Instrument Qualification Model (QM) will be flight representative and will be subjected to qualification and performance testing. This model has the purpose of verifying the electrical and software interfaces as well as the on-board software and operational procedures. It will be checked for Electromagnetic compatibility (EMC) and will validate the performance and calibration activities. The QM will not be delivered to ESA.
- The Instrument Flight Model (FM), as well as the Spare model (FS), will only be subjected to acceptance test levels. The FM will be delivered to ESA two years before launch. The FS will be delivered to ESA two months later.

The RTE inverter design flow (see Figure 6) is in correspondence with this model philosophy. While the Breadboard and the Structural Thermal Model were developed, the RTE inverter finished two requirement and design phases, which were documented for a Preliminary Design Review and a Critical Design Review. A documentation package was generated for each review. After that, an iterative cycle of development, verification and test was carried out, which culminated with the generation of a new documentation package. It has been delivered along with the Electrical and Functional Model.

The first functional test of the RTE Inverter within the DPU was performed during the EFM battery of tests. Such a test is documented in [Ramos (2014)]. We are currently preparing the next integration in the QM, where more exhaustive functional tests will be done. This integration will be carried out by the end of 2015.

A list of all generated documents is shown in Appendix VIII. All the documents in every package have been reviewed by the Project Manager, the Assistant Manager, the Quality Assurance team, and by an external supervision company, SENER. Finally, the documents have been approved by the SO/PHI co-Principal Investigator and released to ESA for final review.

Reaching the specifications for an electronic device that carries out the inversion of the radiative transfer equation for the first time is a major success, which can be attributed to this thesis.

9.2 Time requirements and computing performance

Since the two proposed architectures are in fact the first two trials for inverting the RTE on FPGA, comparison of the performance has to be carried out with other regular software versions of the problem. One of such reference systems is VFISV [Borrero (2010)], which analyzes data coming from the Helioseismic and Magnetic Imager instrument [Scherrer (2002)] of the NASA's *Solar Dynamics Observatory* [Pesnell (2012)]. Such an analysis is made off line when the data have already downloaded on ground. The code runs in a cluster of 50 CPUs. The goal of this system is to carry out the RTE inversion of 16 megapixel images in 10 minutes. That is, four times more pixels in a shorter time than SO/PHI (our time requirement is 15 min for 4 megapixel images; see Table 2). For that, VFISV has 50 times more computing devices than ours and, what is more important, they do not have any limitation of power consumption.

9.2.1 Processing power of the MIMD architecture

As explained in Chapter 4, the MIMD approach only addresses the Spectral Synthesis and Response Functions blocks (SSRF) (see Chapter 2 for details about these blocks). The MIMD architecture was consequently tailored for these parts of the inversion algorithm and automatically adapted for the specific temporal requirement (RTE-R-0030) using TAPAS. TAPAS is able of arranging an optimal pProcessor architecture for the SSRF blocks and that requirement, as shown in Section 4.4.

Following the scheme in Figure 32, the proposed architecture was tested in a Virtex XC5VFX130T FPGA, which is a commercial version of the radiation hardened one. The development board used in this test was the ML510 board by Xilinx [Xilinx]. Since this board does not have external pins, we had to develop an ad-hoc expansion board for debugging the design. This expansion board provides enough pins for using logic analyzers. It was later used for implementing the SoCWire bus. A picture about the final configuration of these devices is shown in Figure 51.

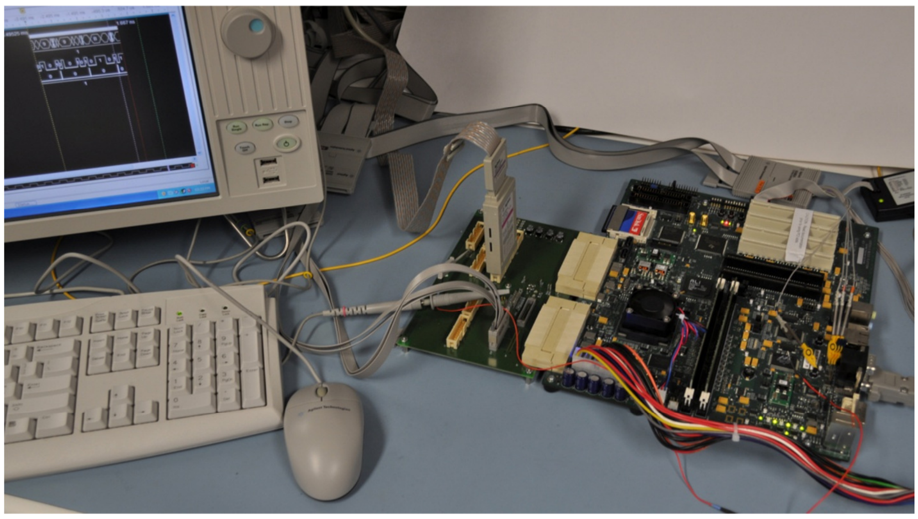


Figure 51 From right to left: ML510 board, the ad-hoc expansion board, and the logic analyzer used for testing the Virtex-5 prototype

As Figure 52 outlines, once the FPGA is programmed, Matlab is used for transmitting data through the RS-232 port to the FPGA. A detailed schematic about this test configuration is shown in Appendix V. The data are transmitted in ASCII mode but MATLAB translates then to a floating-point representation. The UART core is responsible for packing and unpacking the data packet. The results sent by the FPGA are compared with the TAPAS simulation results in Matlab and in runtime.

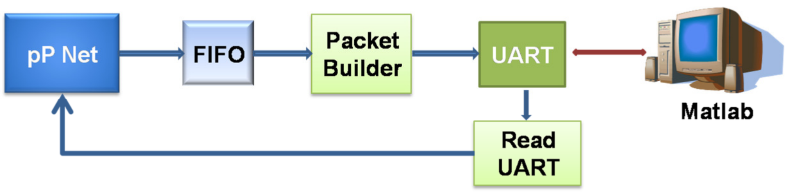


Figure 52 Scheme about the FPGA communication configuration used in the test of the MIMD architecture

This subsection is focused to test the processing power of the proposed architecture but we also take this opportunity for showing its precision because the same configuration test is used.

Using a set of 10,000 MELANIE atmospheric models, the SSRF was computed in the FPGA and was compared with the C-MILOS results. The maximum relative difference between both calculations was of the order of 10^{-7} which is the maximum precision attainable with floating-point, single precision calculations. In any case, such a difference is well below the 10^{-3} rms noise expected for the data. To illustrate the comparison, an example with a specific model atmosphere is shown in Figure 53. More information about this test can be found in [Cobos (2011)].

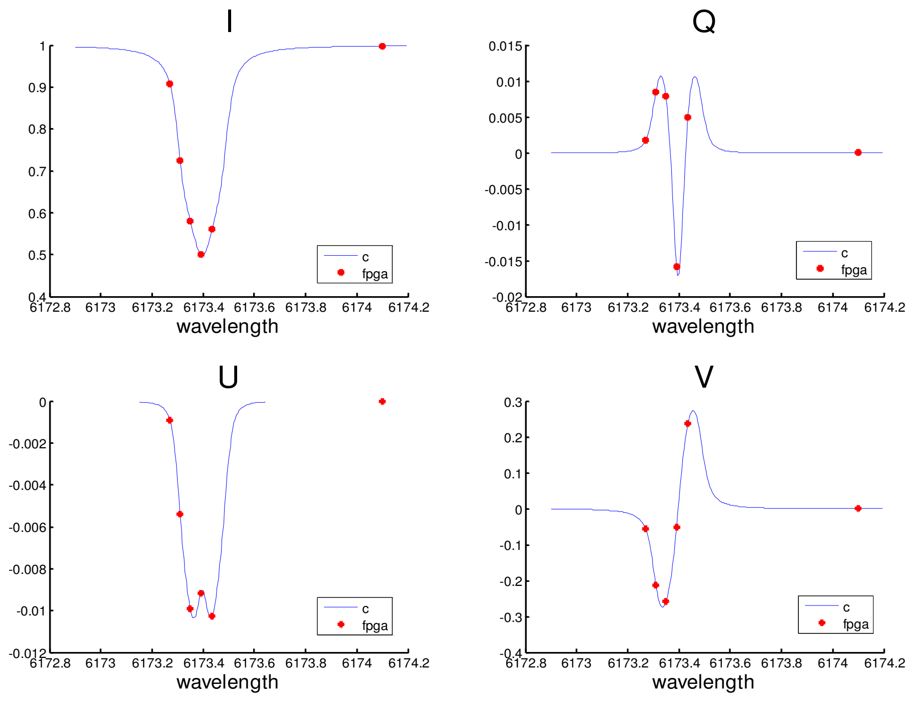


Figure 53 Synthetic Stokes profiles using C-MILOS and the pProcessor net (MIMD architecture) for one selected model atmosphere. (B = 1000 G, inclination = 160º, azimuth = 170º, and v_{LOS} = 2.25 km .s^{-1})

The SSRF block has to be computed as many times as the number of pixels multiplied by the number of RTE iterations: 2048x2048x15, that is, almost 63 million times. Therefore, the 10,000 MELANIE model sets were sent to the FPGA 6,300 times, approximately. The proposed system calculates the Spectral Synthesis and Response Functions almost 63 million times in a minute, hence achieving the engineering goal. Computationally, the system reaches high calculation performance in floating point with single precision. Over 10 GFLOPS running at 200 MHz are reached and using less than 50% of all available resources in the FPGA (see Section 4.4).

On the other hand, Table 19 compares the SSRF calculation times in the FPGA version with the C-MILOS inversion software. They are for one SSRF calculation. Shown is the average time for 10,000 C-MILOS calculations. So far, the FPGA acceleration with respect to a leading PC is almost forty times. However, the possible final acceleration factor for the entire RTE inversion algorithm is more than eighty times. This is possible because the FPGA runs all the inversion tasks in parallel at the same time than the spectral synthesis and response functions are calculated. For instance, the SVD task will be carried in parallel in a similar way to the SIMD architecture does.

Resource	MIMD FPGA	C-MILOS
Device	XC5VFX130T	Intel Xeon
Clock	200 MHz	2,6 GHz
Execution time	1,1 μs	43.35 μs

Table 19 Time comparison of MIMD architecture on FPGA vs PC

Unfortunately, it is no possible to make a specific and more accurate comparison with the system by [Borrero (2010)], or VFISV, because it uses simplifications of the algorithm that we do not consider. We cannot then compute the exact time for their SSRF block. We can assume, however, that the SSRF time is longer in the VFISV case than the C-MILOS one because their total inversion time is 1.57 times longer than ours.

According to the mentioned hardware resource occupation of the MIMD architecture (see Section 4.4), our MIMD system on FPGA is able to run the SSRF twice as fast as a cluster of almost fifty PCs in the VFISF system. Although it is not a severe comparison since they are not using exactly the same algorithm, this quantitative information shows the high performance of the MIMD multicore architecture proposed in this work. In any case, taken as a full system, the magnitude of the reached acceleration is enough to make the MIMD computing capabilities clear, and overall, it reaches the initial objective of doing real-time RTE inversions.

9.2.2 **Processing power of the SIMD architecture**

The final RTE inverter system is based upon the SIMD architecture proposed in Chapter 5 together with the Communication Block introduced in Chapter 8.2. Following the test flow shown in Figure 33, a valid SIMD architecture for carrying out the inversion is obtained and programmed on the FPGA. Once programmed, a battery of tests must be done to ensure the correct functionality of the system. The tests are intended to verify the requirements, which are detailed in the FPGA requirements document [Aparicio (2014)].

To test the RTE inverter at the IAA without using a DPU board, a test platform has been developed [Ramos (2013)]. The software and hardware system designed to provide support during the development and validation of the RTE inverter is outlined in Figure 53. The system allows the delivery and reception of data to/from the SIMD architecture, which is being run in the FPGA device (Virtex-4). The system guarantees the transmission through the SoCWire communication bus in the conditions specified by the DPU.

The data control is performed through data files in ASCII format and a first analysis of the results can be carried out in real time so that the consistence of the results can be assessed. The final analysis of the results will be done, later on, on data files with a specific programming for each test written in the necessary software language (e.g., Matlab or LabView).

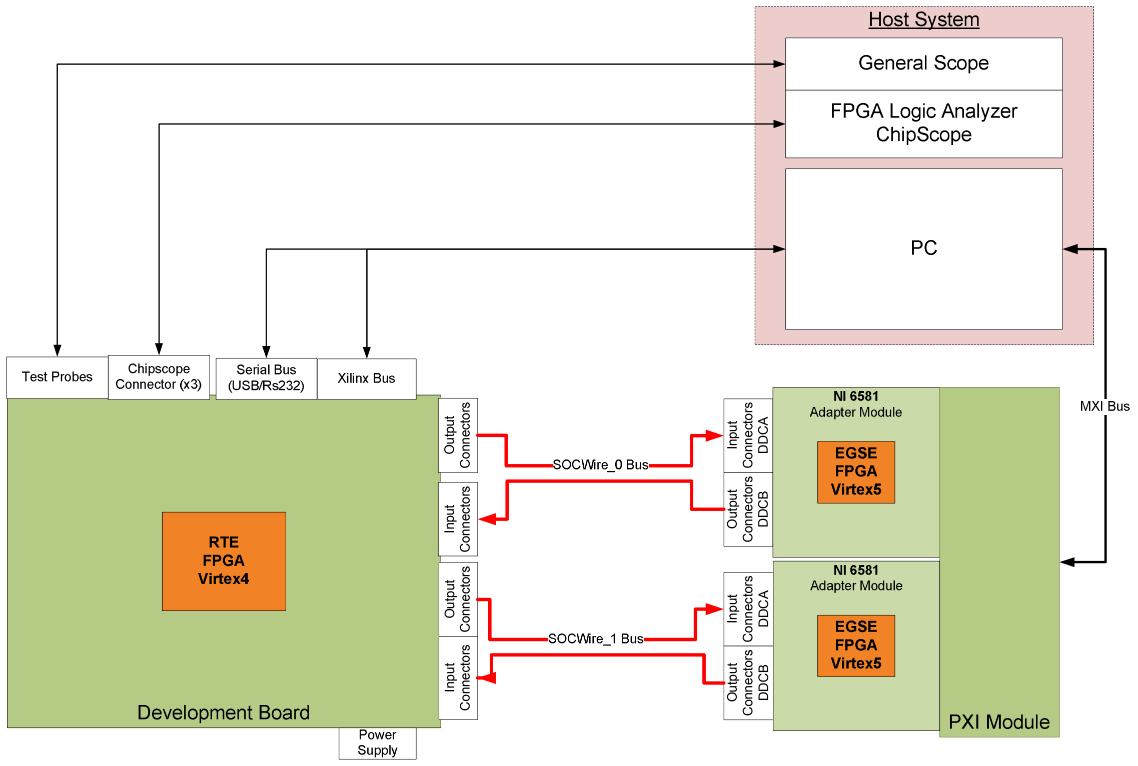


Figure 54 General block diagram of the test platform

The test platform is composed by three main blocks: the development board for the Virtex-4, the PXI Module [National Instrument], and the host PC. No development boards using the XCV4VSX55 FPGA, which is the commercial version of the radiation tolerant one, were in the market. Therefore, a custom development board was ordered to a third party company. The main features of this development board are shown in Appendix IX. In addition, the test platform is extensively detailed in [Ramos (2013)]. A picture of how the test platform is set up in the laboratory is shown in Figure 55.

To test the RTE inversion, the host PC sends a set of profiles to the FPGA in the development board through the PXI module which emulates the SoCWire bus. All the process is controlled by a LabView program which is also presented in [Ramos (2013)].

Using a set of 100,000 MELANIE synthetic profiles we have tested the RTE inversion speed. Table 20 shows that we have achieved a speed almost ten times faster than the commercial CPU used for the RTE inversion in VFISV [Borrero (2010)]. This is even more remarkable since we have used an FPGA that was released to the market in 2004. Obviously the whole computer cluster still remains better than the FPGA, over a factor of five times. Regarding the C-MILOS code we have achieved a speed up of 6.32 times, since this code had already been optimized to low level by us. In the C-MILOS case we have chosen the best case after testing with several compiler optimizations like SSE-MMX extensions, different loop-unrolling methods, etc.

It is important to remark that the responsible for that speed up are not only the nProcessors, but all the architecture involved, since it allows executing the SVD in parallel with the rest of the algorithm. While in a commercial CPU all the RTE inversion is executed in a serial way, our architecture basically does a double-thread execution, as was explained in section 5.2 and depicted in Figure 29.

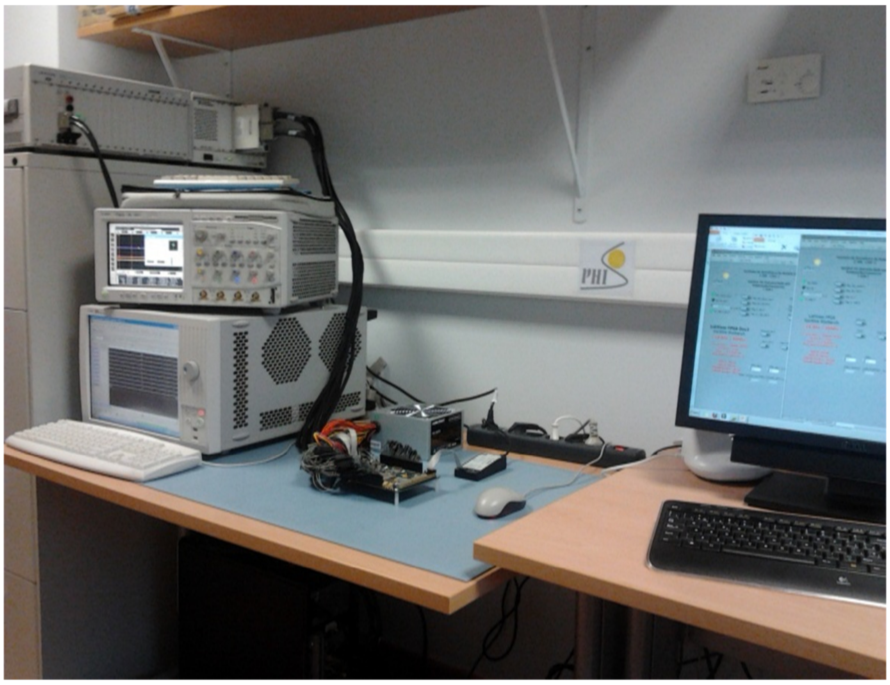


Figure 55 Test platform at the IAA laboratory: (from the upper left to the right) PXI module, oscilloscope, logic analyzer, FPGA development board wired to the PXI module, and the host desktop PC running the LabView test software

	SIMD FPGA	GPU	C-MILOS	1 PC of VFISV
Device	XQR4VSX55	Quadro 600	Intel Xeon	Intel Xeon
Clock GHz	0.15	0.64	2,6	2,6
Profiles/seconds	4765.9	1131.1	754	479.75
Minutes / image	14.6	61.8	92.7	145.7
Speedup vs VFISV	**9.93**	2.35	1.57	1

Table 20 Time comparison of RTE inversion using SIMD architecture and conventional computers.

The RTE inversion algorithm has also been developed in CUDA and executed in a GPU, specifically in a Nvidia Quadro 600 which uses the Fermi architecture [Nvidia]. We have adopted the same execution strategy used in the SIMD architecture where several pixels are executed in parallel in different cores. Actually, several groups of pixels are gathered in blocks of threads and they are distributed in different cores, as is the usual execution schedule in a GPU. As Table 20 shows, despite using 96 cores and a faster clock, the GPU is not able of outperforming the FPGA. It is around four times slower. This GPU device is not in the state of the art. Currently, there are new devices with 20 times more processing power. Nevertheless, this comparison gives us a good evidence of the outstanding performance of our SIMD architecture.

Thinking on ground-based system versions, we could perform the VFISV tasks [Borrero (2010)] using four Virtex-4 FPGA devices instead of a computer cluster of 50 CPUs. Besides, based on our simulations, with only one device of the last Virtex-7 family we could accelerate in a factor two the computer time of VFISV and its 50 CPUs.

Without forgetting the main aim of this system, the time to invert an image in FPGA is around 14.5 minutes. Then we have achieved the target of not exceeding 15 minutes. Thus, we conclude that the SIMD architecture is suitable for being used in the RTE inverter.

9.2.3 MIMD versus SIMD architecture

At a glance, the MIMD architecture overtakes the SIMD computation capabilities. In fact, the MIMD architecture is able of speeding up the C-MILOS version (commercial personal computer) in a factor almost 40, while the SIMD architecture does it only by a factor of little more than 6. Nevertheless, it is necessary to remark that the MIMD architecture uses a bigger device.

Other features of the MIMD architecture make that proposal more innovative and interesting. Among them, we can mention the adaptability to a specific temporal requirement or the easiness for finding the optimal system for that requirement. Even so, a minimal change in the input code implies that the entire MIMD architecture has to be re-compiled, both at a TAPAS level and at a VHDL level since MIMD is customized for each input code. However, changes in the SIMD input code only imply a re-compilation with TAPAS.

On the other hand, the SIMD is a simpler architecture that makes the task of scaling the number of processors easy and exploring different possibilities of co-design hardware/software through the use of shared resources.

9.3 Power consumption requirements

So far, we have given estimations about power consumption but realistic measurements were made with a specific test [Ramos (2014)].

In order to carry out this test, the DPU emulator LabView software must send and receive a data flow that guarantees the RTE inverter is at maximum speed operation. This is made by means of the bus and the SoCP protocol.

The FPGA device has an internal storage memory able to host 120 input spatial pixels in a queue to be processed and another memory permits to queue the corresponding results. The software must guarantee that the input storage memory does not remain empty for a given period. During this period we can measure the device power consumption. After all the input pixels are processed the results are taken by the DPU emulator from the results queue. This data sending and reception is cyclically repeated by the system software during the time necessary to adjust the measurement instrumentation.

The power consumption is measured through the input current reaching the development board. The board power supply is provided with a +5 V DC signal. Inside the development board this voltage is used to generate the supply voltages required for the FPGA trough a specific regulator whose efficiency is around 90%.

Once the RTE inversion cycle starts, the FPGA receives input data and sends results at a maximum internal activity. The scope screen capture in Figure 56 shows how the status of the inverter internal activity translates to device power consumption. It can be noted how, after having received the 120 input pixels, the device internal activity continues for 57 ms. The 15 ms of low consumption status is the time the DPU emulator takes to get the results and process this information.

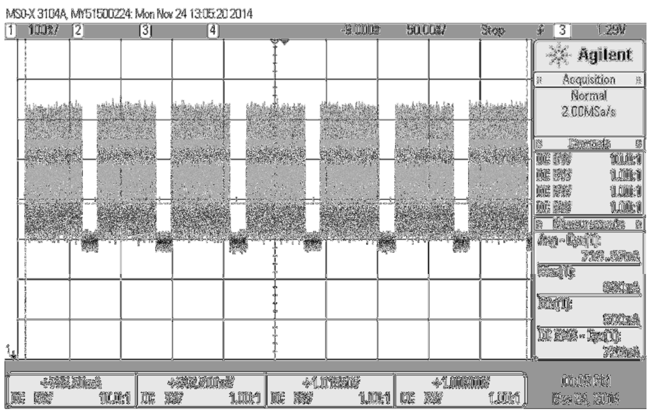


Figure 56 Scope screen capture that shows the RTE Inverter power consumption cycles

Using this test configuration we have taken measurements of power consumption in different FPGA status: before and after the FPGA is programmed and once the RTE inverter is working at its maximum load. Table 21 reflects the average, maximum, and minimum power where the regulator efficiency has been already applied. When the FPGA is programmed and the RTE inverter is waiting for data, the maximum power consumption is around 3.2 W and when the RTE inverter is carrying out the inversion this maximum power is near 5 W. Since these measurements include other small electronic devices in the development board we can conclude that the power consumption estimations given in Table 11 are realistic. We estimate 4.5 W only for the FPGA device using the Xilinx XPower tool. In any case, we conclude that the RTE inverter has fulfilled the power requirement of using less than 5 W.

FPGA Status	Power (Watts)		
	Average	Max	Min
FPGA not programmed	1.63	1.75	1.51
FPGA programmed	3.05	3.21	2.91
RTE Inverter Working	3.59	4.95	3.00

Table 21 Power consumption measurements for the FPGA not programmed, programmed and the RTE Inverter working

The power consumption of a commercial CPU like those used in VFISV is around 100 Watt [Intel]. Therefore our power savings is of the order of 1000 with respect to VFISV. It is also remarkable that the GPU we used in the former subsection needs around 40 Watts, hence almost 10 times as much power as that of the FPGA.

9.4 RTE inversion scientific requirements

Only the SIMD architecture will be subject of scientific validation. This is so because the complete problem has not been addressed with MIMD. To check the scientific results, our test bench is C-MILOS. On the one hand, we have demonstrated (Chap. 2) that it is equivalent to the original IDL-MILOS. On the other hand, the FPGA code is based on C-MILOS.

First, we have inverted with the FPGA and C-MILOS the same reference basis of Milne-Eddington synthetic Stokes noisy profiles used in Section 2.4. In Figure 57, a scatter plot of the results from both inversions is shown. The fitting is almost perfect for these Milne-Eddington profiles since the points hardly scatter away the straight line of slope unity. Table 22 summarizes the root mean square differences (RMSE) among the results of both versions. For the sake of comparison, the RMSE between C-MILOS and IDL-MILOS are also shown. One can clearly see that the contribution to these RMSEs by the FPGA programming alone can be considered negligible.

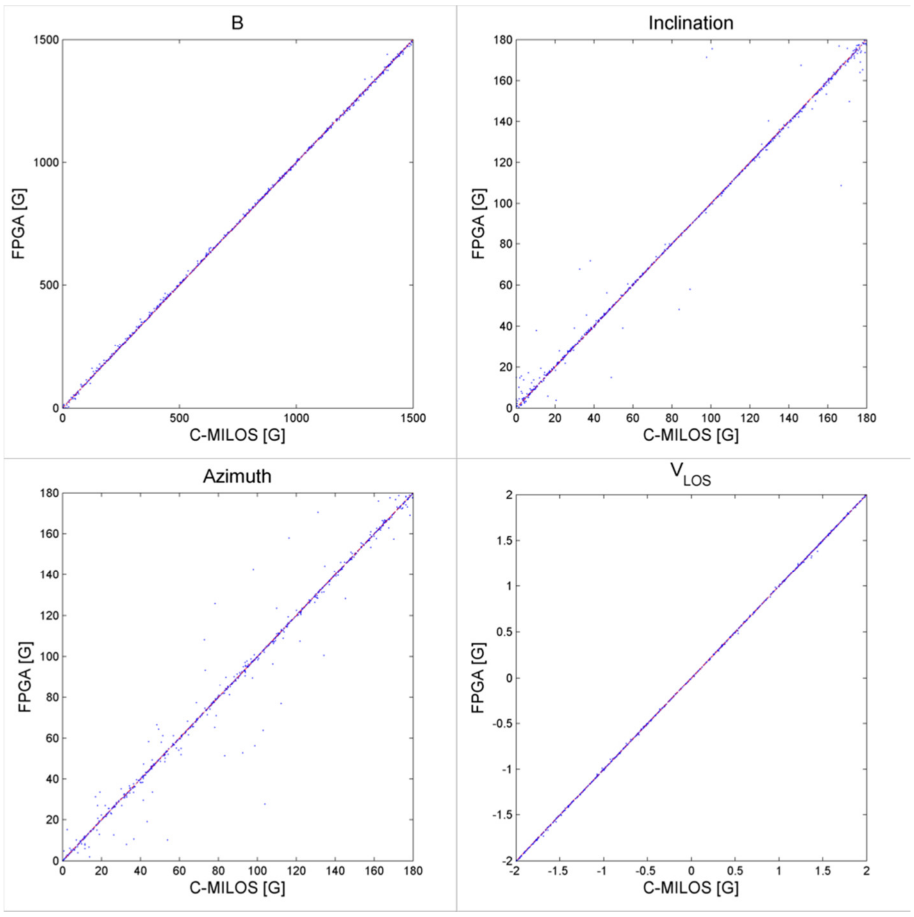


Figure 57 Scatter plot for the magnetic field strength, field inclination, field azimuth, and LOS velocity inverted on the FPGA and C-MILOS. The red dashed line shows a one-to-one correspondence

	FPGA vs C-MILOS	C-MILOS vs IDL-MILOS
	RMSE	RMSE
Field strength (G)	5.30	4.55
Inclination (º)	4.86	4.20
Azimuth (º)	5.77	5.39
Velocity (ms^{-1})	5.90	6.10

Table 22 RMSE for inversions carried out with FPGA and C-MILOS

Unlike ME profiles, real solar Stokes profiles are not symmetric. Therefore, a ME inversion that tries to fit them to symmetric profiles will necessarily fail. This is intrinsic to the inversion itself and is accepted by solar researchers that still continue using the Milne-Eddington approximation to analyze their results. What is crystal clear is that the misfits obtained with the FPGA must be as close as possible to those obtained with the post-facto, software codes of the RTE inversion. The nature itself of the calculation already heralds that a comparison between the software (C-MILOS) and hardware (FPGA) versions with real profiles may not be as ideally good as that in Figure 57. Many iterative procedures are nested and decisions on precision have to be taken in order to cope with the device resources. But indeed such a comparison is the needed cornerstone to qualify the electronic inverter of the RTE from a scientific point of view.

The first test has been performed using an image set including a big sunspot and an area of quiet Sun. The image was taken at the Swedish Solar Telescope in the Roque de los Muchachos Observatory, La Palma, on September 28, 2011. The spectral line is sampled at 35 mÅ. We have only chosen those samples at [-140, -70, 0, +70, +140, +420] mÅ far from the line core (6173.335 Å), in order to simulate the SO/PHI measurements. The continuum intensity image is shown in Figure 58. It has an 894 x 883 pixel size, each 56 arcsec wide.

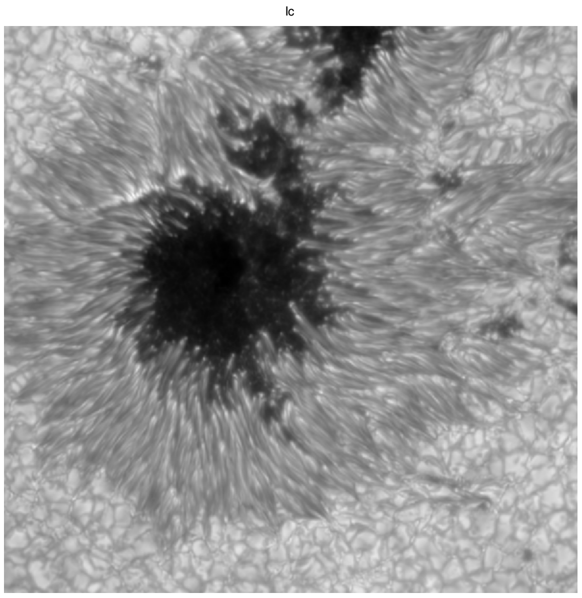


Figure 58 Continuum intensity image of the sunspot image from the CRISP instrument

Figure 59 shows the result of inverting the CRISP data set using the FPGA (left column) and C-MILOS (right column). We show the magnetic field strength, field inclination, field azimuth, and LOS velocity results because these are the model atmospheric parameters that the RTE inverter has to provide (see Table 2). Both inversions have run 15 iterations.

IDL-MILOS uses a convolution with the instrument profile with a previous interpolation of the samples. We cannot afford this interpolation in our system and simply calculate the direct convolution with our five samples within the line. Therefore, comparisons are shown without convolution.

Although the graphical comparison in Figure 59 looks very nice except for a few locations in the umbra (inner, darker part of the spot), the differences between the two inversions are shown in Figure 60. Differences in the umbra can be attributed to the lower level signal in that area, which is, thus, more liable to errors induced by noise.

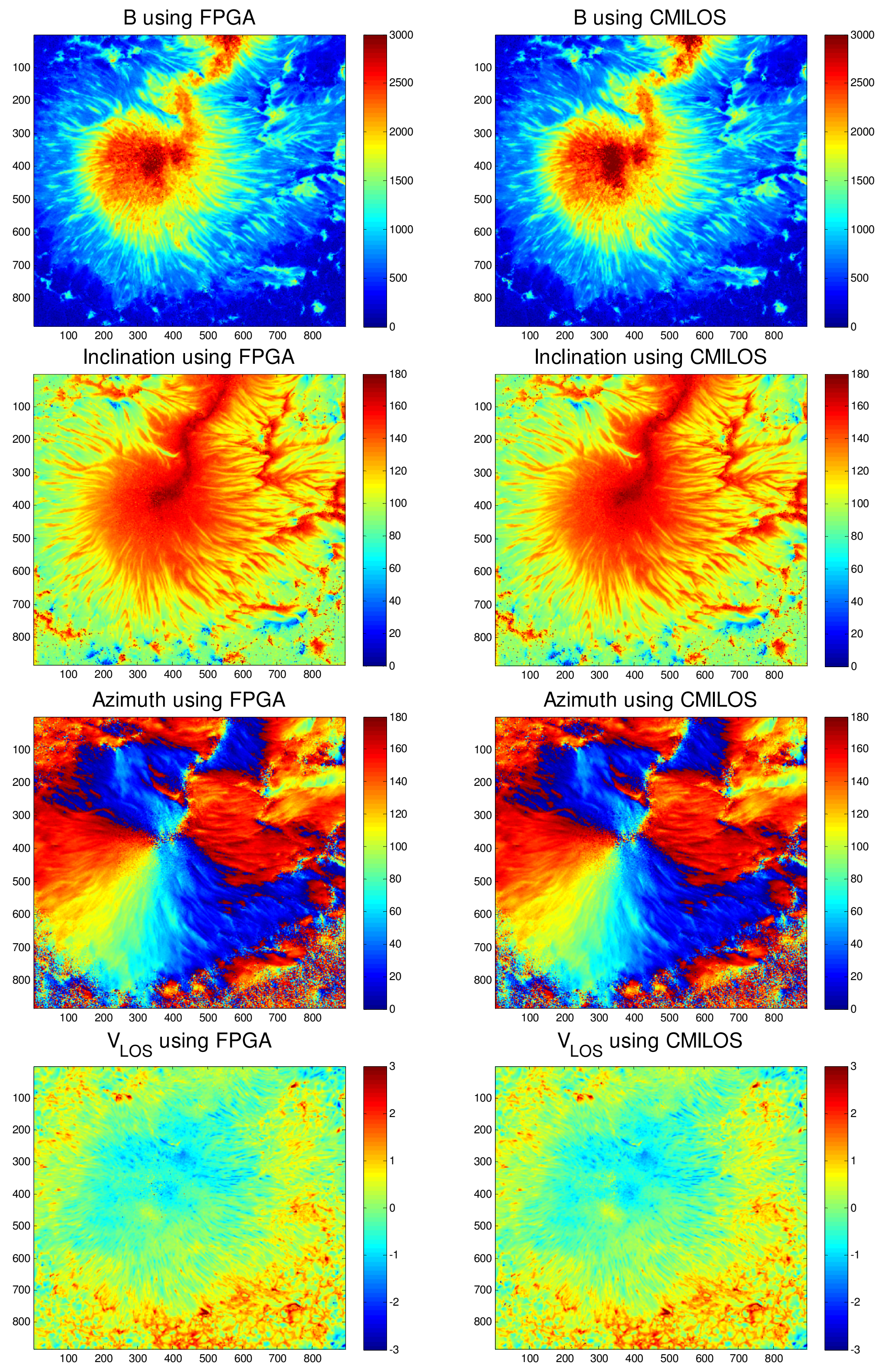


Figure 59 Magnetic field strength (Gauss), field inclination (degree), field azimuth (degree), and LOS velocity (km·s^{-1}) as a result of inverting the sunspot image using FPGA (left) and C-MILOS (right)

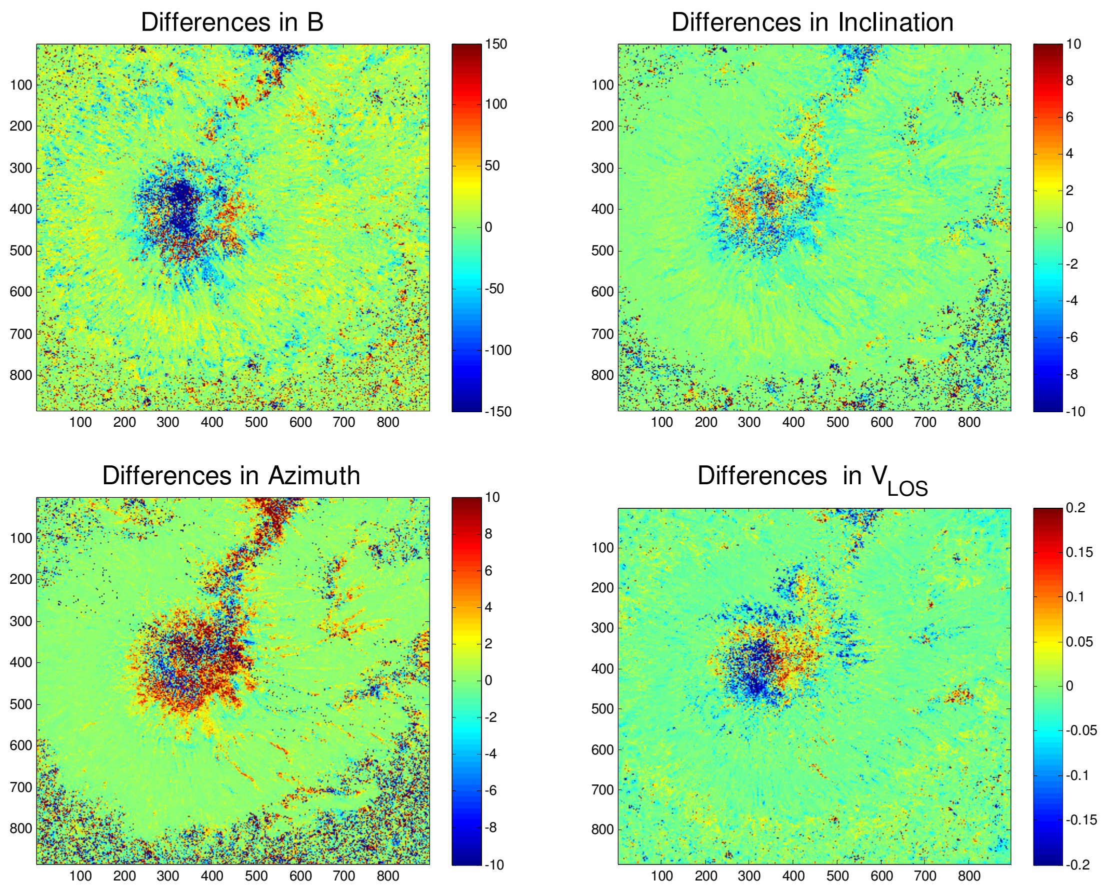


Figure 60 Differences between the magnetic field strength (Gauss), field inclination (degree), field azimuth (degree), and LOS velocity (km·s^{-1}) after inversion of the sunspot image using FPGA and C-MILOS

A scatter plot of the two inversion results and the corresponding histogram of their differences are shown in Figures 62 and 63, respectively.

In general, the fits are very good except perhaps in the strong field regime, where some excess scattering can be seen. Such stronger fields belong indeed to umbral points. The largest negative differences in magnetic field strength that appear in the histogram of Figure 63 also correspond to them. The differences, however, do not seem very large when compared with those between C- and IDL-MILOS (see Table 23).

	Entire sunspot image	
	FPGA vs. C-MILOS	C-MILOS vs. IDL-MILOS
	RMSE	RMSE
Field strength (G)	69.2	86.47
Inclination (º)	5.47	4.51
Azimuth (º)	6.50	4.50
Velocity (ms^{-1})	79.1	77.8

Table 23 RMSE for the differences between the FPGA and C-MILOS inversions, and C-MILOS and IDL-MILOS, for the entire sunspot image.

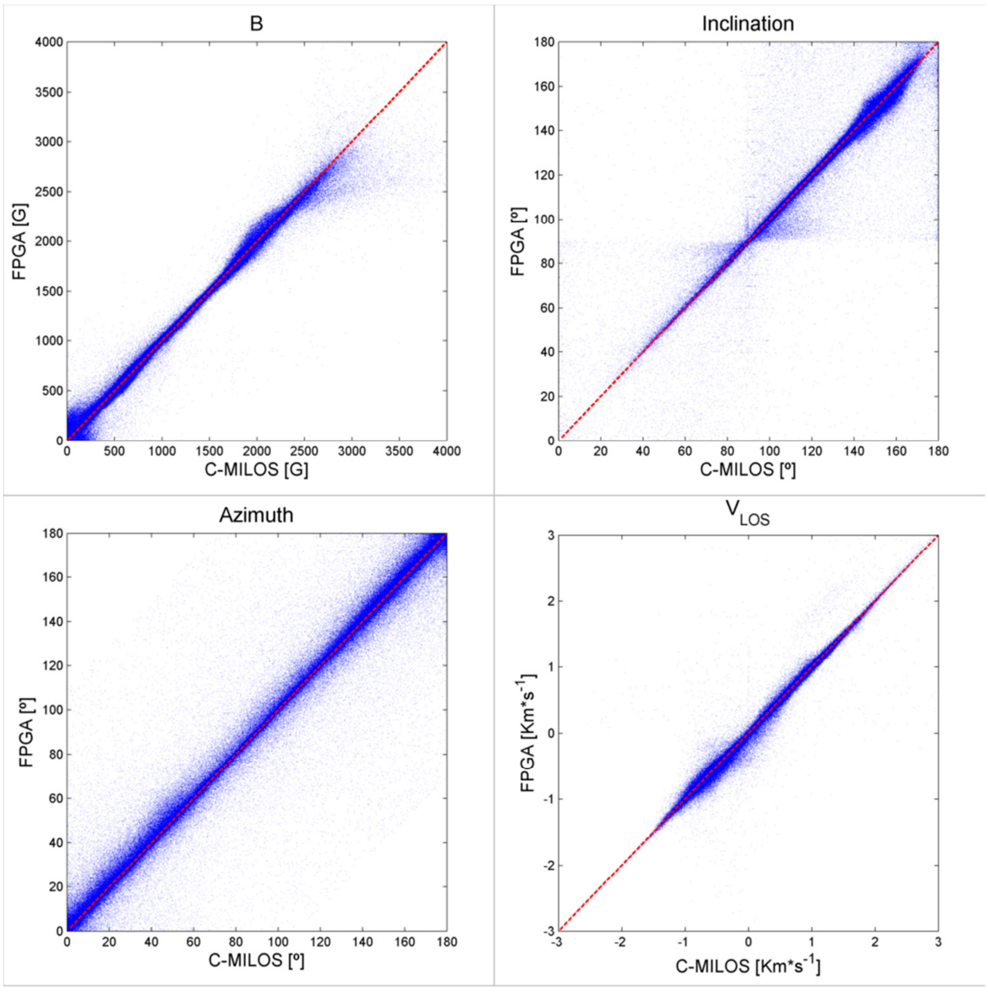


Figure 61 Scatter plot for the magnetic field strength, field inclination, field azimuth, and LOS velocity inverted on the FPGA and C-MILOS for the sunspot image. The red dashed line shows a one-to-one correspondence.

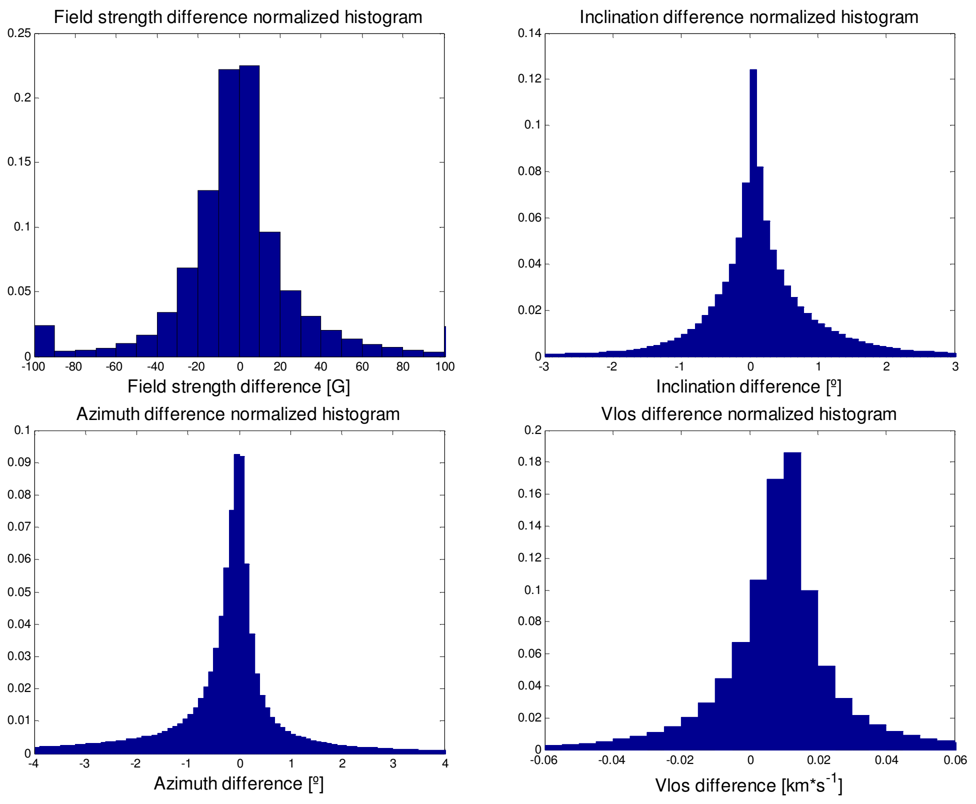


Figure 62 Normalized histograms of the differences in the magnetic field strength, field inclination, field azimuth, and LOS velocity results from inversion on the FPGA and C-MILOS

If our hypothesis is true that noise is directly influencing the largest deviations in the umbra, restricting our test to less noisy areas of the image should produce better results. This is indeed shown in the scatter plots in Figure 65, where inversions of the selected area in Figure 63 are displayed. Table 24 summarizes the RMSEs for these scatter plots. The improvement in the results is clear. Note that the RMSEs between the FPGA and C-MILOS are even smaller than those between the two software versions.

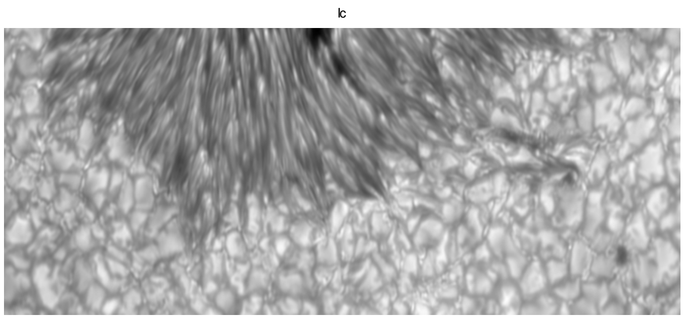


Figure 63 Selected area of the sunspot image without umbra

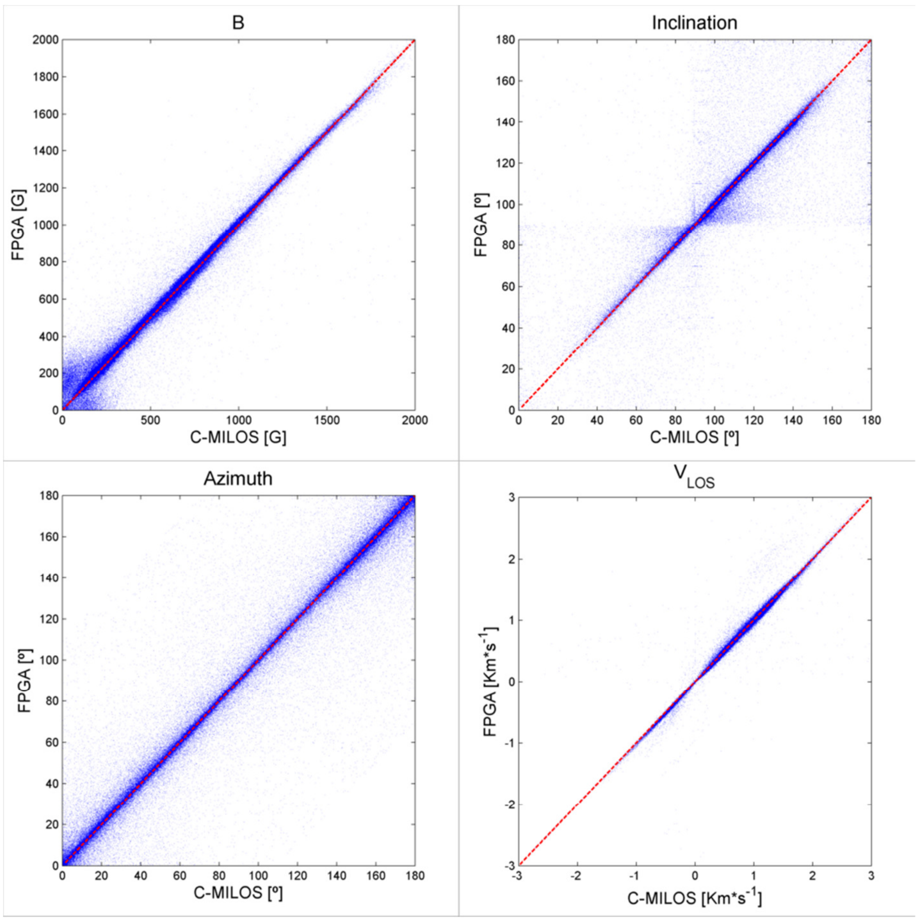


Figure 64 Scatter plot for the magnetic field strength, field inclination, field azimuth, and LOS velocity of correspondences between FPGA and C-MILOS results for the selected part of the sunspot image. The red dashed line shows a one-to-one correspondence.

	Bottom part of the sunspot image	
	FPGA vs. C-MILOS	C-MILOS vs. IDL-MILOS
	RMSE	RMSE
Field strength (G)	46.34	46.90
Inclination (º)	6.09	6.92
Azimuth (º)	7.45	5.71
Velocity (ms^{-1})	61.1	86.3

Table 24 RMSE for the differences between the FPGA and C-MILOS inversions, and C-MILOS and IDL-MILOS, for the selected part of the sunspot image.

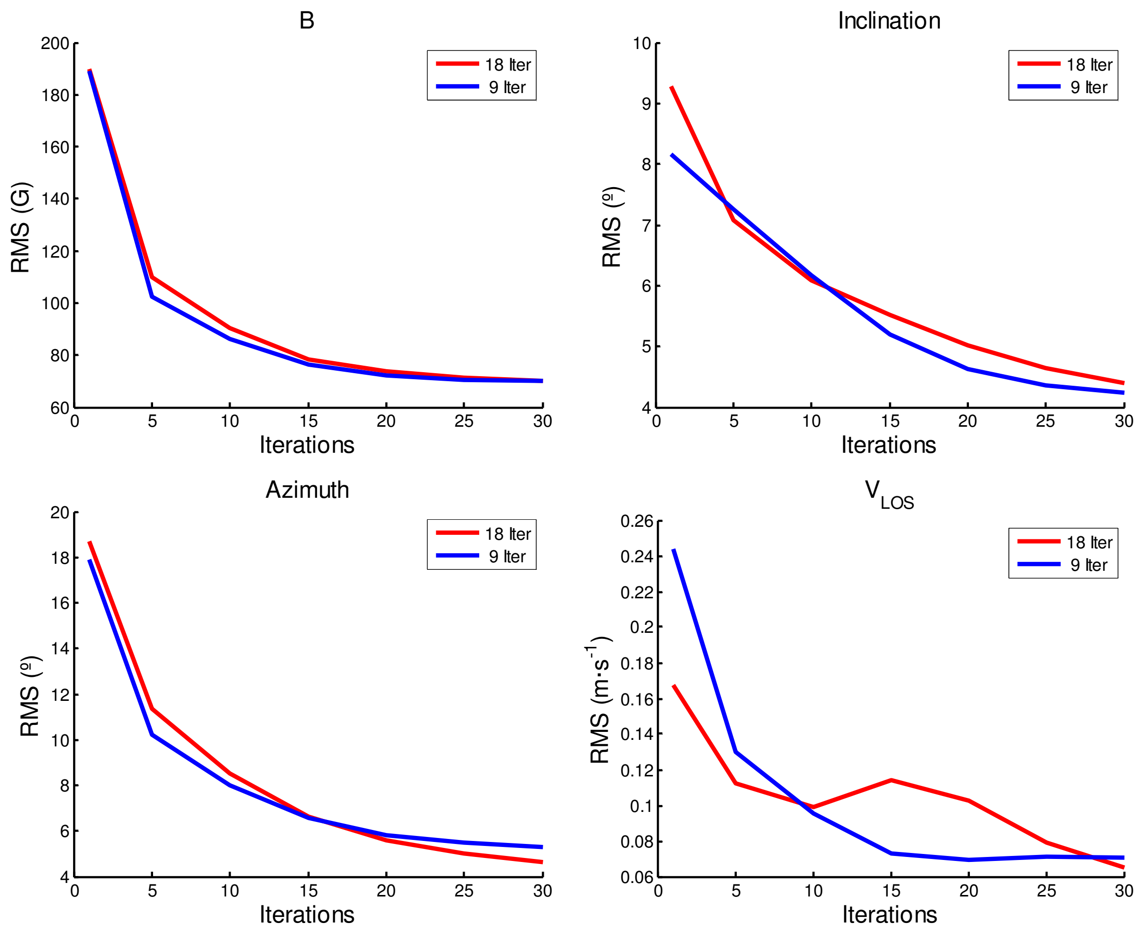


Figure 65 Evolution of the RMS difference between the FPGA and C-MILOS as a function of the number of RTE iterations for both 9 (blue lines) and 18 (red lines) SVD iterations.

All the tests in this section have been carried out with an SVD version including only 9 iterations. As commented on in Chapter 7, the SVD iterative procedure can be carried out with any multiple of 9 iterations in order to avoid further re-sorting of the eigenvalues and eigenvectors. Thus, the question arises as to why not to use 18 or a higher number of internal iterations. The results in Figure 64 already indicate that 9 iterations are enough for providing results within the scientific requirements. Nevertheless, we further address this question in Figure 65, where the evolution of the RMS difference between the FPGA and C-MILOS is shown as a function of the number of RTE iterations for both 9 (blue lines) and 18 (red lines) SVD iterations. The reference (C-MILOS) is taken with 30 iterations, which is assumed to give the optimum results. The convergence to the optimum result is apparent. Both 9 and 18 iterations seem to provide similar results, so that we decide to keep the smaller number in order to speed up the whole procedure. The small bump that appears in the LOS velocity plot for 18 iterations is due to a few points close to the umbra but we still do not have a real reason for it.

10

Conclusions and future prospects

The RTE inverter has been presented as the first device ever specifically designed to invert the radiative transfer equation aboard a space-borne instrument. The stringent time and power consumption constraints of space instrumentation, as the Polarimetric and Helioseismic Imager for *Solar Orbiter,* made the development a real challenge, which has been finally successful. Two high-performance scientific computing architectures on FPGA have been proposed, one of them will be implemented in the real instrument.

The RTE inversion is an involved, iterative, non-linear least squares minimization of a merit function. Such a merit function measures a distance (goodness of fit) between the observed and synthetic Stokes profiles of a given spectral line. An optimized RTE inversion code called, C-MILOS, has been presented that is based on a previous version (MILOS) written by scientists in IDL. We have demonstrated that C-MILOS working in single floating point precision is as reliable and robust as MILOS, which works in double precision. We have assessed the computational cost and how performance is affected by the working spectral line, the number of wavelength samples, and the convolutions related with the broadening instrumental profile.

We have designed two new processor architectures for adapting the parallel computer paradigms to the RTE inversion problem using current high-performance computing techniques like multi-core architectures, code optimization, and specific-domain efficient processors. We have taken well established, state-of-the-art, computing models like MIMD and SIMD, and have applied novel ideas on them for getting enhanced their computing capabilities.

Both multiprocessor architectures are proposed in order to achieve high performance in floating point precision using the Xilinx FPGA Virtex-5 and Virtex-4 respectively and trying to make the best out of all the FPGA resources.

We have proposed an MIMD multiprocessor architecture as a firm candidate to be part of embedded systems in an FPGA, mainly due to its ability for exploiting the functional and data parallel algorithms. This architecture is original because of its pipelined execution based on a novel programming method, called intensive-pipelining software. Using this method, the architecture can increase the system performance. With the proposed design, the synchronization and the communication between processors have been simplified. The implementation of this architecture using simplified processors, pProcessors, has been shown. Such pProcessors work for eliminating latency and for exploiting the computing power that the FPGA provides.

On the other hand, a SIMD multiprocessor architecture has been presented and it is finally in charge of carrying out the scientific analysis aboard the SO/PHI instrument within the DPU instrument. The new proposal was developed because the Virtex-5 FGA was not finally accepted by ESA. Despite the SIMD architecture is slower than the MIMD version, it provides a good scheme to save resources having a unique control unit for all the processors. Besides, one of the main contributions of this work is the ability of saving resources allocating operation cores in a shared operation block. The pipeline-designed processors of this architecture are tailored for reaching a high rate of executed instructions, trying to execute one instruction per clock cycle. An innovative memory address space has been introduced in order to feed the processor with its operands as fast as possible. The memory works as if it was a cache and it is statically scheduled by the compiler.

The proposed architectures are very focused on the RTE inversion problem, but we have pointed out that they can be used in other embarrassingly parallel problems since the number of processors in the architectures can be configured an adapted. Thus, the architectures are presented as scalable and configurable.

We have presented a software tool, TAPAS, which follows given design guidelines and makes it easier the use and programming of the proposed MIMD and SIMD architectures. It also makes the debugging and test tasks easier because it provides simulations of both the architectures and its running code.

This tool uses advanced techniques of software pipelining. Specifically, the compiler is decisive in this work, since it is responsible for re-ordering the instructions and organizing the memories in order to exploit the architectures at maximum. The associate programming language makes it easier to program the architecture using a C-like style and isolating the code from the under system.

The RTE inversion algorithm needs to perform the Singular Value Decomposition of a correlation matrix within its iterative procedure. A specific SVD pipelined architecture which is able of diagonalizing two correlation matrices at the same time has been developed. The final SVD architecture has been integrated within the SIMD architecture; it exceeds the best systems on FPGA in time and precision performance.

The impact of SEU induced errors in the proposed architecture has been discussed and two different strategies for detecting and mitigating errors within the RTE inverter have been proposed. Furthermore, one of the strategies is able to detect and correct error for most elements within the architecture.

A software protocol for communication between the RTE inverter and the DPU, based on register operations, has been detailed.

By using TAPAS and a Virtex-5 FPGA, a fully-configured pProcessor MIMD architecture has been obtained along with the communication network inside it. The proposed system calculates the synthesis and spectral response functions almost 63 million times in a minute, achieving the science goal. Computationally, the system reaches high calculation performance in floating point with single precision: over 10 GFLOPS running at 200 MHz and using less than 50% of all available resources on the FPGA. This means that it is able of exceeding to commercial desktop processor by a factor 40.

Using the SIMD architecture, the challenge of carrying out the RTE inversion in less than 15 minutes has been reached. The architecture has not only demonstrated that is able to do it but it is also improves the computing capabilities of ground systems by more than ten times using a relatively slow (and 10 year-old) Virtex-4 FPGA device.

This dissertation has demonstrated that FPGAs offer enough floating-point capabilities and enable allocation of specific-domain processors to solve high demanding scientific problems even embedded aboard a space-borne instrument.

The RTE inverter prototype has been tested using real images taken by another instrument. It is able of working as accurately as usual computers regarding the scientific precision. In addition, it has satisfied the stringent requirements of power consumption and processing time.

In summary, this thesis has provided the scientific community with high-performance computing architectures, compilers, configuration and simulation tools, and specific mathematic cores like the SVD core, that all of them assembled are able of carrying out the same computing problem than a cluster of PCs but using only FPGA devices.

Future work

Many are the research opportunities that this thesis generates. Among them let us enumerate the following:

1) Development of a fault tolerant architecture.

2) Implementation of the proven solutions into other devices of the Virtex family.

3) Use of another type of bus (e.g. AMBA) that allows an easy integration in other systems.

4) Extensive comparison with GPU solutions.

5) Extend this type of architecture to other algorithms and benchmarks.

6) Convergence of the two SIMD and MIMD architectures in TAPAS.

7) Comparison of power consumption between MIMD and TopGreen500.

Appendices

Appendix I. The Voigt function

The original Voigt function code is shown in code 5. In Figure 66, we show the pipeline design for the Voigt function (see Chapter 2). This function originally was in complex operations, like in code 5. Here it is translated to real ones. We show the necessary fixed-point format in each stage of the calculation. Is important to remark that this function is executed 18 times (see Table 2) along each RTE iteration (once for each wavelength by the cuantic number (2*N_PI+N_SIG)).

```
%fvoigt function for RTE inversion
function [h,f] = fvoigt(damp,vv)

    A=[122.6087931777104326,
          214.382388694706425,
          181.928533092181549,
          93.155580458138441,
          30.180142196210589 ,
          5.912626209773153,
          0.564189583562615];
    B=[122.60793177387535,
        352.730625110963558,
        457.334478783897737,
        348.703917719495792,
        170.354001821091472,
        53.992906912940207,
        10.479857114260399,
        1];

    Z=complex(damp,-abs(vv));

    Z=((((((A(7)*Z+A(6)).*Z+A(5)).*Z+A(4)).*Z+A(3)).*Z+A(2)).*Z+A(1))./ ...
  ((((((Z+B(7)).*Z+B(6)).*Z+B(5)).*Z+B(4)).*Z+B(3)).*Z+B(2)).*Z+B(1));

    h=real(Z);
    f=sign(vv).*imag(Z) / 2;
```

Code 5 Matlab code for the Voigt function

R=((((((A(6)*Z+A(5))*Z+A(4))*Z+A(3))*Z+A(2))*Z+A(1))*Z+A(0))/
(((((((Z+B(6))*Z+B(5))*Z+B(4))*Z+B(3))*Z+B(2))*Z+B(1))*Z+B(0))

a=[122.607931777104326,
214.382388694706425,
181.928533092181549,
93.155580458138441,
30.180142196210589,
5.912626209773153,
0.564189583562615]

b=[122.60793177387535,
352.730625110963558,
457.334478783897737,
348.703917719495792,
170.354001821091472,
53.992906912940207,
10.479857114260399]

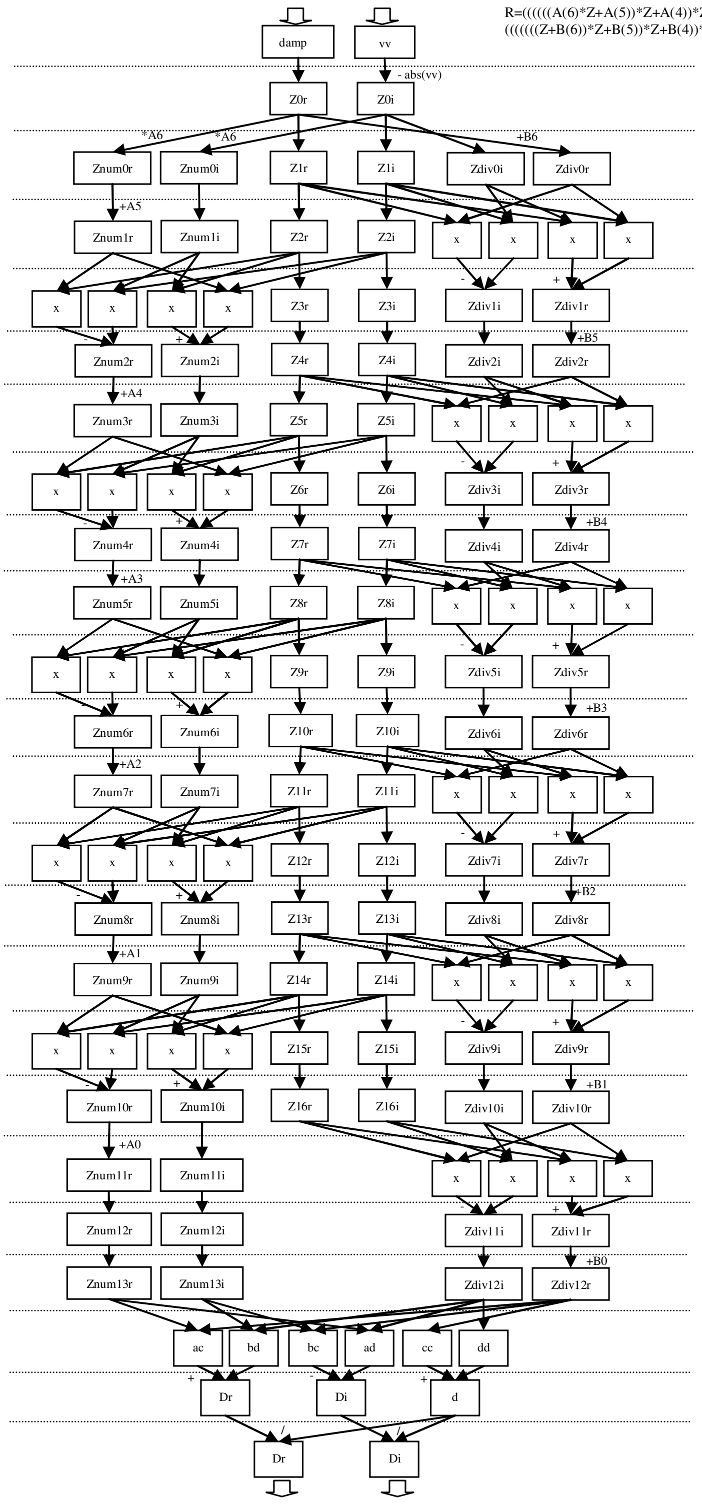


Figure 66 Pipelined Voigt function

Appendix II. HLS description of a RTE inversion block

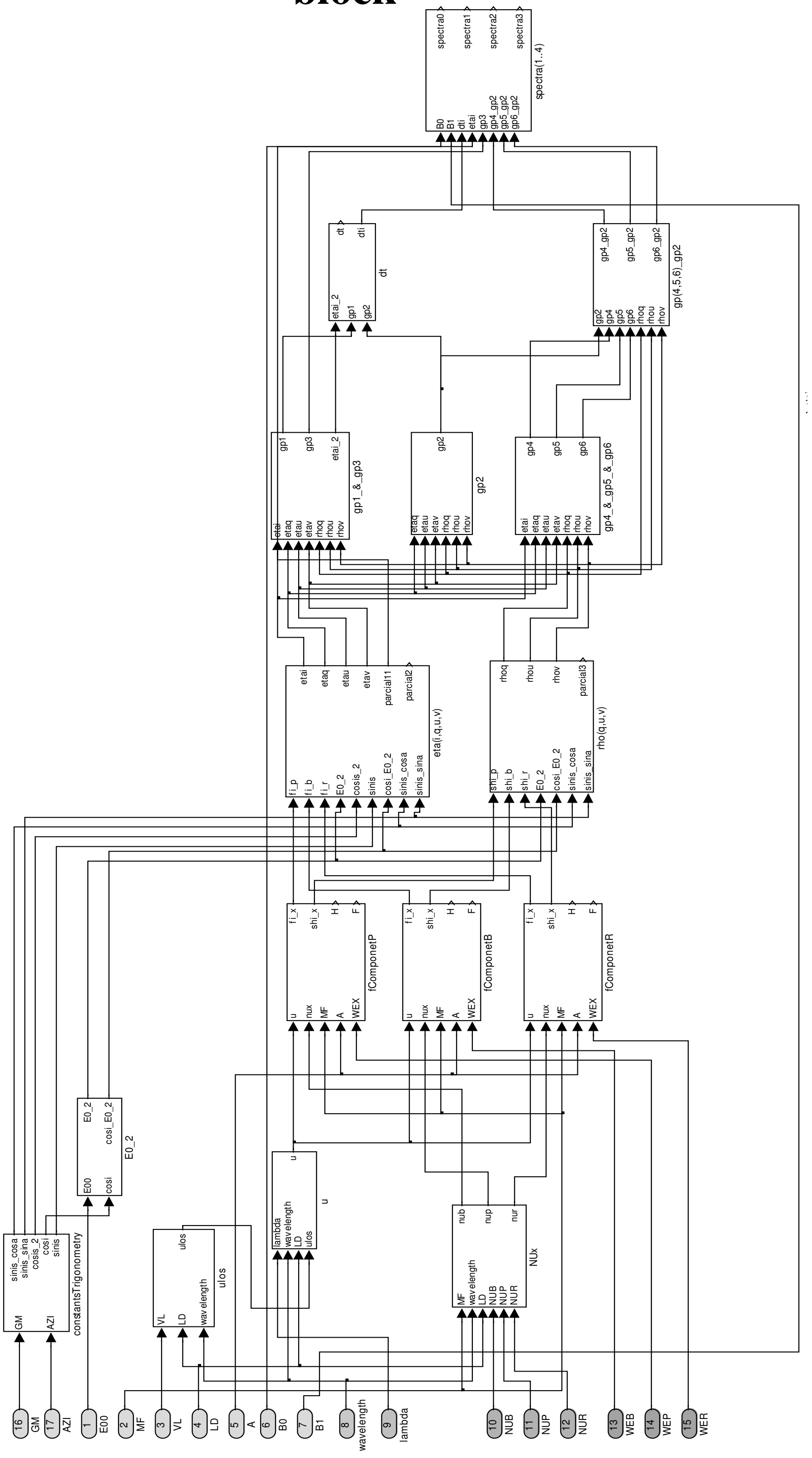


Figure 67 Spectral Synthesis block using System Generator.

Appendix III. MIMD architecture notes

In the MIMD architecture it is necessary to store a result for several stages before using it. This is a requirement to guarantee the memory consistency: all pPs exchanging data should see the same writing/reading order in the right positions referred to every data. For example, if data A (generated at iteration 1) is needed in iteration 3, it must be stored during iteration 2, without using it.

To properly keep the results, each datum is assigned to several memory addresses. It is important to take into account that the instructions to control the writing and reading addresses are stored in a ROM; this means that they cannot be modified. However, there are some data that have to be kept in RAM because they are needed in later iterations, therefore their position in RAM must change at execution time. To control these special data we have added external counters to every pPs. The counters are incremented on each iteration. The difference among them is that each counter has a different maximum. The number of external counters to use is different for each pP and is driven by the dataflow. Looking at the data that a pP has to handle, it is possible to see the maximum number of times that a datum has to be stored in RAM and the maximum number of data to be stored between iterations. Empirical observations looking the dataflow show that the maximum number of counters for a pProcessor is 8. The minimum is 1.

Therefore when we read from the ROM the addresses to write to or to read from, we have to change this value according to the consistency of the data in that iteration. That is, we have to add the value of the corresponding counter.

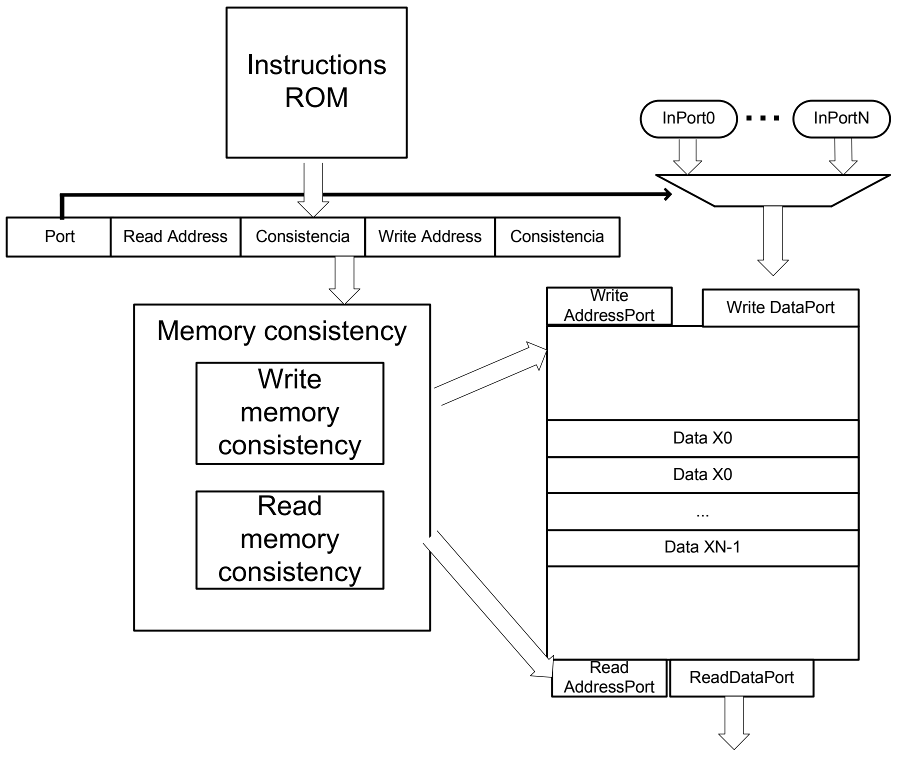


Figure 68 Illustration about the memory consistency process

When we read from the ROM the write address, first we have to add the value of the write counter to this address. When we read from the ROM the read address, we have to add the value of the corresponding counter and an offset to avoid reading from and writing in the same address at the same time. This is expressed by Equations III.1

Final Write Address = Base Write Address + write counter

Final Read Address = Base Read Address + read counter + offset (III.1)

In order to calculate the offset, it is necessary to take into account the maximum consistency of each datum (maximum counter value). In fact, when a datum has consistency then it must have some amount of memory reserved to be stored.

The offset is used to avoid writing and reading simultaneously on the same address, and to read always the last written position in the memory. Therefore, the way to calculate it is expressed in Equations III.2

Memory reserved for a datum = Maximum consistency +1 (III.2)

Offset = Memory reserved for a datum – Actual Consistency value

Instruction Codification in the nProcessor architecture

Every pP has a ROM to store the set of instructions to execute. Every instruction contains data indicating input ports for each RAM, the write/read addresses, and the output port. In Table 25 we show the codification for each 72 bits in the instruction word stored in the ROM.

RAM A			RAM B		
Input port	Write Address	Consist write	Input port	Write Address	Consist write
3 bits	10 bits	3 bit	3 bits	10 bits	3 bit
Bits 71-69	Bit 68-59	Bit 58-56	Bits 55-53	Bits 52-43	Bit 42-40

Operation code
3 bits
Bits 39-37

RAM A			RAM B		
Read Address	Consist Read	Offset Read	Read Address	Consist read	Offset Read
10 bits	3 bit	3 bit	10 bits	3 bits	3 bit
Bit 36-27	Bit 26-24	Bit 23-20	Bits 19-10	Bit 9-7	Bit 6-3

Output port
3 bit
Bit 2-0

Table 25: Instruction encoding for each word stored in ROM

Each instruction has several fields:

1. Input port A (bits 71-69): is the input port to the RAM A. We can have up to 8 different inputs for each RAM memory. Therefore, the field is 3-bit wide.
2. Write Address A (bits 68-59): is the address where the data is written in the RAM A. The field is 10-bit wide. As explained before, the final address depends on the memory consistency.
3. Consist Write A (bits 58-56): it indicates the counter whose value is necessary to add to the write address A to obtain the final write address. As the maximum number of counters for a pP is 8, the field is 3-bit wide. The input counters are numbered from 0 to 7 in each pP.
4. Input port B (bits 55-53): is the input port to the RAM B. As said before, we can have up to 8 different inputs in each RAM.
5. Write Address B (bits 52-43): is the write address to the RAM B. The final writing address depends on the memory consistency
6. Consist Write B (bits 42-40): it indicates the counter whose value is necessary to add to the write address B to obtain the final write address.
7. Operation code (bits 39-37): this value is used to control the multiplexer to change the ALU operands (see Fig 5). When Operation Code is 000 then the operands are not interchanged. When Operation Code is 001 the operands are interchanged.
8. Read Address A (bits 36-27): is the read address for the RAM A. The final reading address depends on the memory consistency and on the offset.
9. Consist Read A (bits 26-24): it indicates the counter whose value is necessary to add to the read address A to obtain the final reading address.

10. Offset Read A (bits 23-20): it indicates the offset to add to the read address A to obtain the final reading address.
11. Read Address B (bits 19-10): is the read address for the RAM B. The final reading address depends on the memory consistency and on the offset.
12. Consist Read B (bits 9-7): it indicates the counter whose value is necessary to add to the read address B before obtaining the final reading address.
13. Offset Read B (bits 6-3): it indicates the offset to add to the read address B to obtain the final reading address.
14. Output Port (bits 2-0): it indicates the pP output port. As each pP can have up to 8 different output ports, this field is 3-bit wide.

In this section we present the results for the synthesis of only one pProcessor. These results pretend to be only an orientation, since the inclusion of several pPs on the FPGA will alter the results because of the space occupied by the interconnection routing.

As said before, the pPs are classified in several types according to the operation they perform. The logic utilization and the latency are different for each type. In order to minimize the used logic, the operation core of each pP uses the DSP48Es available within the Virtex5 device [Xilinx]. The number of DSP48Es used in each case depends on the operation.

Logic utilization	Used	Available	Utilization
Number of slice registers	572	81920	0,7%
Number of slice LUTs	575	81920	0,7%
Number of occupied SLICEMs	9	6320	0,14%
Number of BlockRAM/FIFO	3	298	1%
Number of 36k Block used	1		
Number of 18k Block used	2		
Number of DSP48Es	2	320	0,62%

Table 26: Device usage summary for the addition picoprocessor only

Logic utilization	Used	Available	Utilization
Number of slice registers	572	81920	0,7%
Number of slice LUTs	575	81920	0,7%
Number of occupied SLICEMs	9	6320	0,14%
Number of BlockRAM/FIFO	3	298	1%
Number of 36k Block used	1		
Number of 18k Block used	2		
Number of DSP48Es	2	320	0,62%

Table 27: Device usage summary for the subtraction picoprocessor only

Logic utilization	Used	Available	Utilization
Number of slice registers	433	81920	0,53%
Number of slice LUTs	434	81920	0,53%
Number of occupied SLICEMs	1	6320	0,01%
Number of BlockRAM/FIFO	3	298	1%
Number of 36k Block used	1		
Number of 18k Block used	2		
Number of DSP48Es	3	320	0,94%

Table 28: Device usage summary for the multiplication picoprocessor only

Logic utilization	Used	Available	Utilization
Number of slice registers	1730	81920	2,1%
Number of slice LUTs	1102	81920	1,34%
Number of occupied SLICEMs	17	6320	0,27%
Number of BlockRAM/FIFO	3	298	1%
Number of 36k Block used	1		
Number of 18k Block used	2		
Number of DSP48Es	0	320	0%

Table 29: Device usage summary for the division picoprocessor only

As is shown in the tables, the use of DSP48Es significantly reduces the logic usage and the power consumption. Unfortunately, the division operation cannot be performed using DSP48Es and this increases the logic.

The latency depends on the arithmetic operation to perform and the desired clock speed for the FPGA. The faster the clock, the longer the latency. The number of resources is increased also with the clock speed. Figure 69 shows the evolution of the latency depending on the clock speed. The graph has been obtained from [Xilinx]

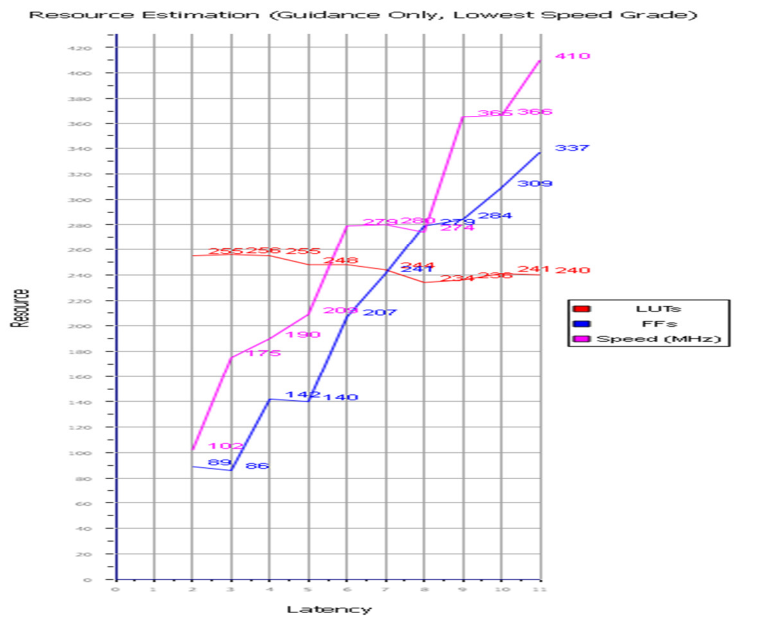


Figure 69 Latency evolution with resources and speed for addition operation.

Table 30 shows the different latencies according to the operation to achieve 200 MHz:

Operation	Latency
Addition	6
Substraction	6
Product	3
Division	28

Table 30: Latencies according to operation

Appendix IV. SIMD architecture notes

The nProcessor ALU can make multiplication and addition-subtraction arithmetic operations in the 32 bit single-precision floating-point standard. The ALU also performs the comparison operations: greater, greater or equal, lower, lower or equal, and equal than. As shown in Figure 22, the "codop_alu" signal codifies the operation to execute in the ALU. The four bit codification is shown in Table 31. Besides arithmetic and comparison operation codes, the "codop_alu" codifies the *else* and *end* operations which are used to generate instruction flow control signals.

The comparison operations do not produce a numeric result, but logic results through the "*flag_cmp"* register (see figure 3). The "*flag_cmp"* register is a cluster of four one-bit signals and it is detailed in Table 32. The "*flag_cmp"* register is used, together with the Enable input port, by the Local Control unit for managing the instruction catch, and this will be detailed below.

ALU operation	Codop_alu value
Addition	0000
Subtraction	0001
Multiply	0010
Greater than	1000
Greater or equal than	1001
Lower than	1010
Lower or equal than	1011
Equal than	1100
Else	1101
End	1110

Table 31 ALU operation codification using the "codop_alu" signal.

Flag_cmp field	Description
Flagif	Result of a comparison instruction. (1 true, 0 false)
Flagif_ready	It means when *flagif* is valid.
Else_rdy	It means when an *else* operation has been executed.
End_rdy	It means when an *end* operation has been executed.

Table 32 flag_cmp register fields

The mux_operand_select multiplexor (see figure 3) provides the data to the ALU from the data memories. In addition, this multiplexor sorts the order of the operand if it were needed.

This multiplexor is managed by the "codop_load" three-bit signal whose codification is shown in the Table 33.

Codop_load value	**Mux_operand_select function**		**Comments**
000	*Data A RAM* -> Op A	*Data B RAM* -> Op A	Parallel load
001	*Data A RAM* -> Op B	*Data B RAM* -> Op B	Crossed load
010	*Data A RAM* -> Op A	-	Op B keeps the previous value
011	-	*Data B RAM* -> Op A	Op B keeps the previous value
100	-	*Data B RAM* -> Op B	Op A keeps the previous value
101	*Data A RAM* -> Op B	-	Op A keeps the previous value
110	*Data A RAM* -> Op A	*Data B RAM* -> Op A	Op A square
111	*Data A RAM* -> Op B	*Data B RAM* -> Op B	Op B square

Table 33 Mux_operand_select codification

The Input port is used to introduce new data to the nProcessor memory space, therefore the Input port accept 32-bits floating-point data. An input data can be addressed to both memories. The *Mux_load_data_RAM* multiplexor is used for loading the input data or the ALU result in memory, this choice depends on the signal "*rinput_load*", see Table 34.

Rinput_load signal value	**Mux_load_data_RAM function**
0	The ALU result is loaded in memory
1	The Input port is loaded in memory

Table 34 Function Selection of Mux_load_data_ram

The Output port is used to get out data. The Mux_load_Routput multiplexor is used for choosing between data from Data A RAM or Data B RAM. It is managed by the "load_rout" signal (see Table 35).

Load_rout signal value	**Mux_load_Routput function**
0	The output port is loaded with a data from Data A RAM
1	The output port is loaded with a data from Data B RAM

Table 35 Function Selection of mux_load_routput

The Instruction input port is used to provide of instructions to the nProcessor. The instruction has 48 bits of width and defines the nProcessor behavior every instant. That is, each instruction defines all the necessary control signals at a given time.

Instruction fields (from MSB to LSB)	Field width (bits)
RINPUT_LOAD	1
LOAD_ROUTPUT	1
ROUTPUT_READY	1
CODOP_ALU	4
CODOP_LOAD	3
READ_ADDRESS_A	12
READ_ADDRESS_B	12
WRITE_ADDRESS_RAMA	1
WRITE_ADDRESS_RAMB	1
WRITE_ADDRESS	12

Table 36 Instruction fields

In the Conditional Flag Stack the results of previous ALU comparison operations are stored because Local Control reads the ALU comparison operation results using the flag_cmp register. In Table 37 indicates how the *flag_cmp* is used in the Conditional Flag Stack management.

Flag_cmp field	Conditional flag stack action
Flagif	Adds the result of a ALU comparison operation to the stack top
Flagif_ready	Indicates when Flaig is valid
Else_rdy	Changes the value of the stack top
End_rdy	Remove the value on the stack top

Table 37 Conditional flag stack behaviour

At FPGA technology level, it is important to point out that the nProcessor architecture projects very well with the FPGA resources. nProcessors use embedded mathematical cores, DSP48, and use Block RAM to implement each data RAM

<u>nProcessor net Control Unit</u>

The nProcessor net Control Unit behaviour is also managed by the instruction ROM which contains the control signals to control this unit –and also the RTE inverter at the time. The instructions are of 54 bits of width and they have three fields: loop, command and cargo (they are detailed in Table 38). The nPCU reads the instructions, one by one, and always interprets the Loop and Command fields. The Cargo field content depends on the command in the Command field.

ROM Instruction fields (from MSB to LSB)	Field width (bits)
Loop	2
Command	4
Cargo	48

Table 38 Fields of the Instruction ROM

The Loop field can have three values: loop starts ("01"), loop ends ("10"), and no loop indications ("00"). The instructions between loop starts and loop ends are sent to each nProcessor in different turns. Therefore, the instructions in a loop are executed twelve times, one time by each nProcessor. The control unit implements a counter –control_np- that set up the nProcessors using the nP Enable ports.

The Command field has two main utilities: instruction flow control and data flow control, as can be seen in Table 39. In the instruction flow control case, the Cargo field of the Instruction ROM has an auxiliary field of the commands. However, in the data flow control case, the Cargo field contains nP instructions. The commands are detailed in Table 39.

The commands of the instruction flow control type manage a counter –program counter- that governs the read of instructions from the Instruction ROM. On the other hand, the commands of the data flow control type manage the entire data flow in the nProcessor net: input/output data flow and the data flow with the SOB unit. For this task, the data flow control commands act over the Data_in_demux demultiplexor, over the Data_out_mux multiplexor, and over the SOB control port –according to the command. Besides, since only one nProcessor can access the SOB, and to the input/output buffers, the commands of the data flow control type work together with the control_np counter, generated by the Loop commands, in order to arrange in sequence the nProcessor access.

Command code	Command type	Command name	Cargo (From MSB to LSB, adjusted to the right)
0000	No command	nP_instruction	nP instruction
1000	Instruction flow control	If_control	Condition_code(5 bits) Jump_direction(16 bits)
1001		Reset_counter	Count_code(5 bits)
1010		Inc_counter	Count_code(5 bits)
1011		Jump	Jump_direction(16 bits)
1111	Data flow control	SVD_in	nP instruction
1110		SVD_out	nP instruction
0001		Input	nP instruction
0010		Output	nP instruction
0011		Division	nP instruction
0100		Trig	nP instruction
0101		SQRT	nP instruction
0110		ATAN	nP instruction

Table 39 Command set

Shared Operations Block

Control port code	SOB Function	Comments
001	SVD_in	It introduces an upper diagonal covariance matrix for the SVD to be performed. Sixty values must be provided in the input port.
010	SVD_out	It gets out the SVD result through the output port: 1 normalization value, 10 eigenvalues, and 10 eigenvectors (100 values).
011	Division	It performs the division of two operands. The two operands are provided in a sequence of dividend and divisor in the input port. The division result is provided using the Output port.
100	Atan	It performs the Arctangent de Y/X. The two operands are provided in a sequence of Y and X through input port. The result is provided using the Output port.
101	SQRT	It performs the square root of an operand. The square root result is provided using the Output port.
110	SinCos	It performs the sine and cosine of an operand. The sine and cosine are provided using the Output port.

Table 40 SOB functionality

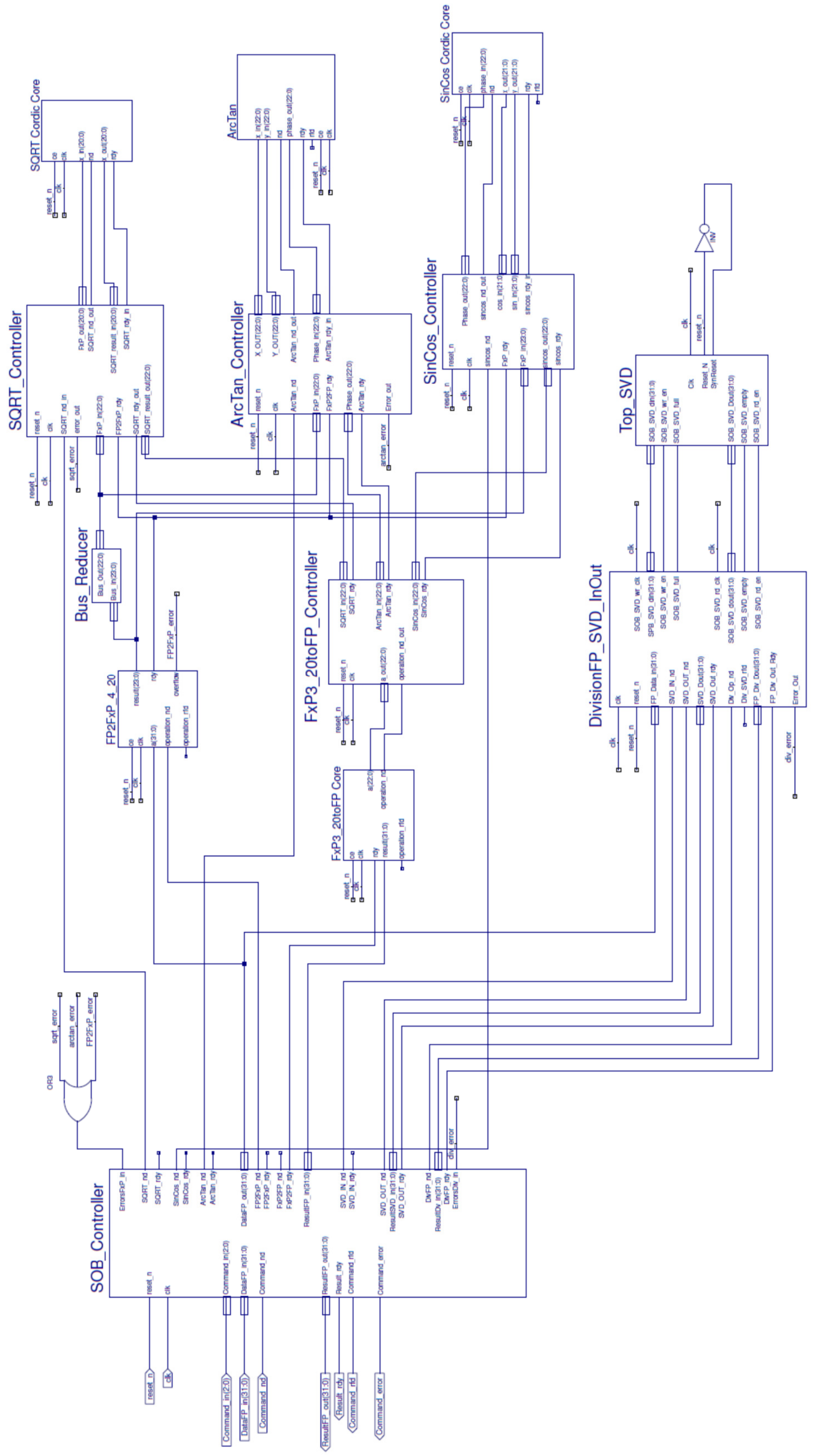


Figure 70 SOB schematic

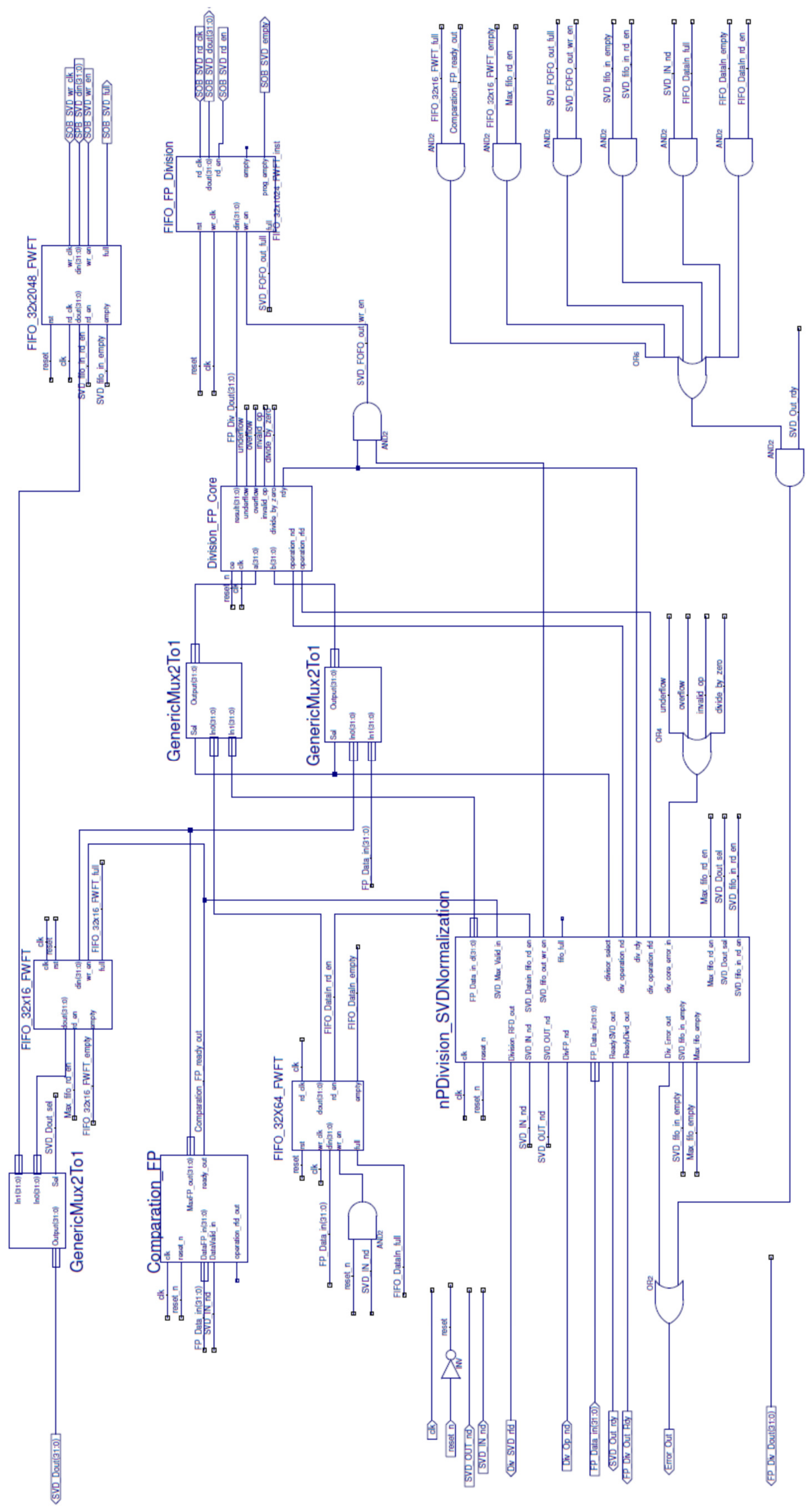


Figure 71 Normalize block within SOB

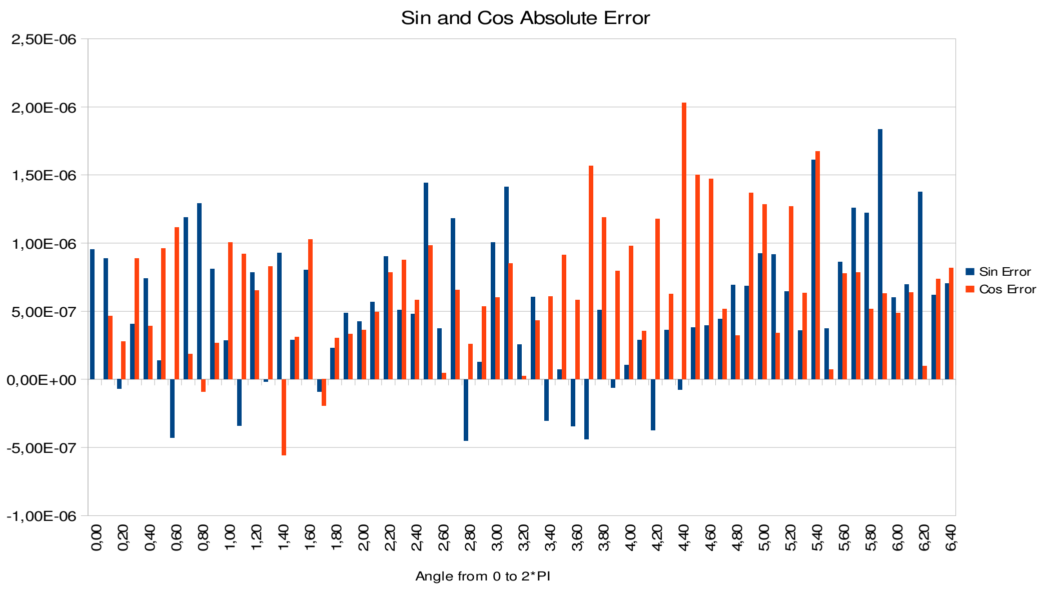


Figure 72: Sin and Cos Error (SOB)

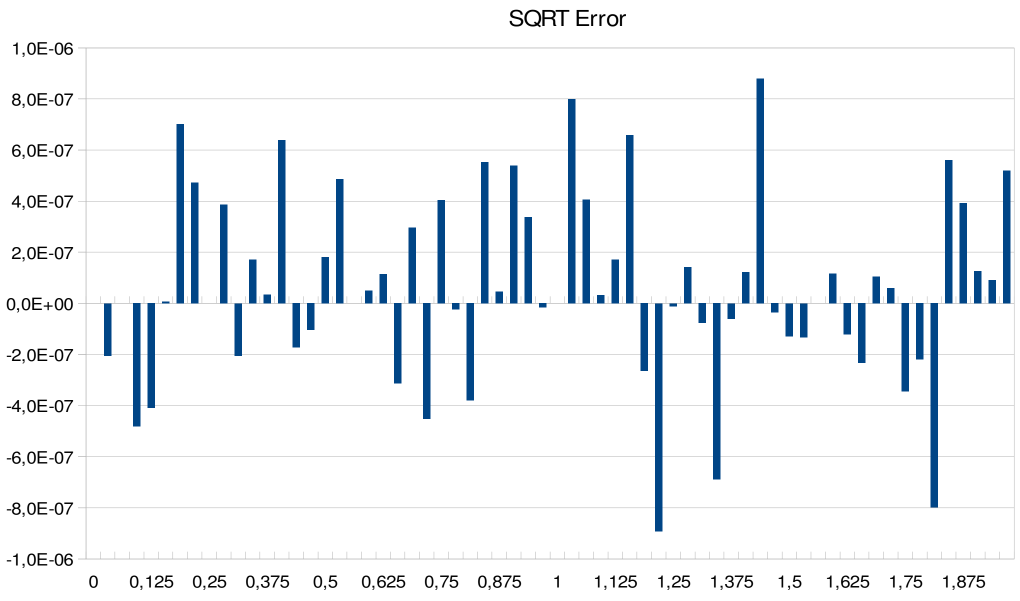


Figure 73: SQRT Error in (SOB)

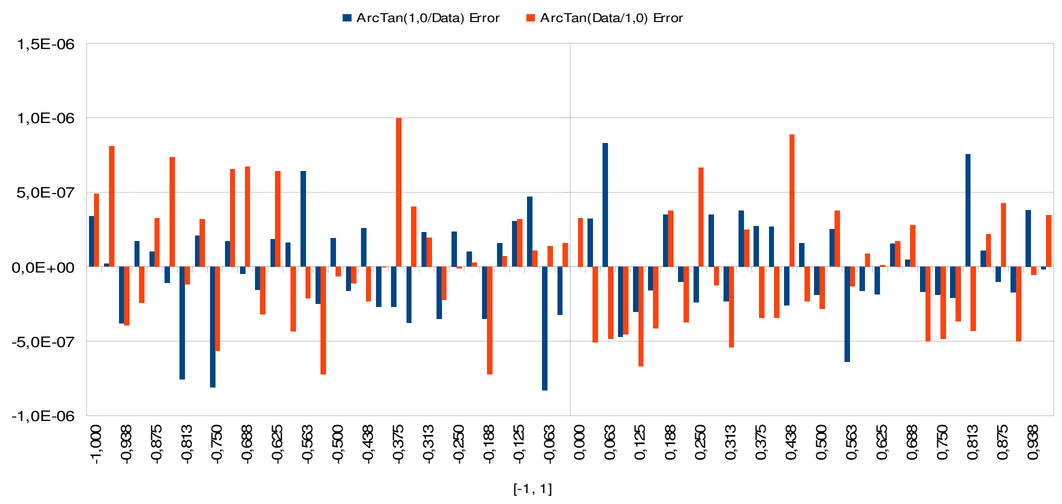


Figure 74: ArcTan Error (SOB).

Core	LUTs	BRAM	Slices	% Total	Max. Frequenc.
Arc_Tan_Controller	46	0	26	0%	354,00
SinCos_Controller	25	0	14	0%	-
FxP_20toFP_Controller	48	0	24	0%	-
SOB_Controller	99	0	50	0%	474,24
SQRT_Controller	5	0	3	0%	-
ArcTan.xco	681	0	377	1%	142,59
FxP_3_20toFP.xco	174	0	97	0%	183,25
DivisionFP.xco	828	0	789	3%	268,00
SQRT_FxP.xco	604	0	503	2%	249,19
SinCos.xco	757	0	435	1%	158,35
FP2FxP_3_20.xco	195	0	108	0%	170,11
DivisionFP_SVD_InOut:	1505	7	1406	5%	161,66
DivisionFP.xco	828	0	789	3%	268,00
FIFO_32x64_FWFT.xco	62	1	77	0%	328,19
FIFO_32x16_FWFT.xco	96	0	118	0%	315,11
FIFO_32x2048_FWFT.xco	106	4	123	0%	307,97
Comparator_FP.xco	71	0	35	0%	
FIFO_FP_Division.xco (32x1024)	90	2	110	0%	310,72
Total Cores en DivisionFP_SVD_InOut	1253	7	1252		
SOB_Top	**4132**	**7**	**3043**	**12,00%**	**142,59**

Figure 75 Resources occupation of SOB and sub-blocks. See section 5.3.

The floating point division core is generated by Xilinx Core Generator software [IR08]. Those cores accomplish with the floating point IEEE-754 Standard [NR01], as was required in RTE-R-0180 [Aparicio (2014)].

Appendix V. Communications based on RS-232 schematic

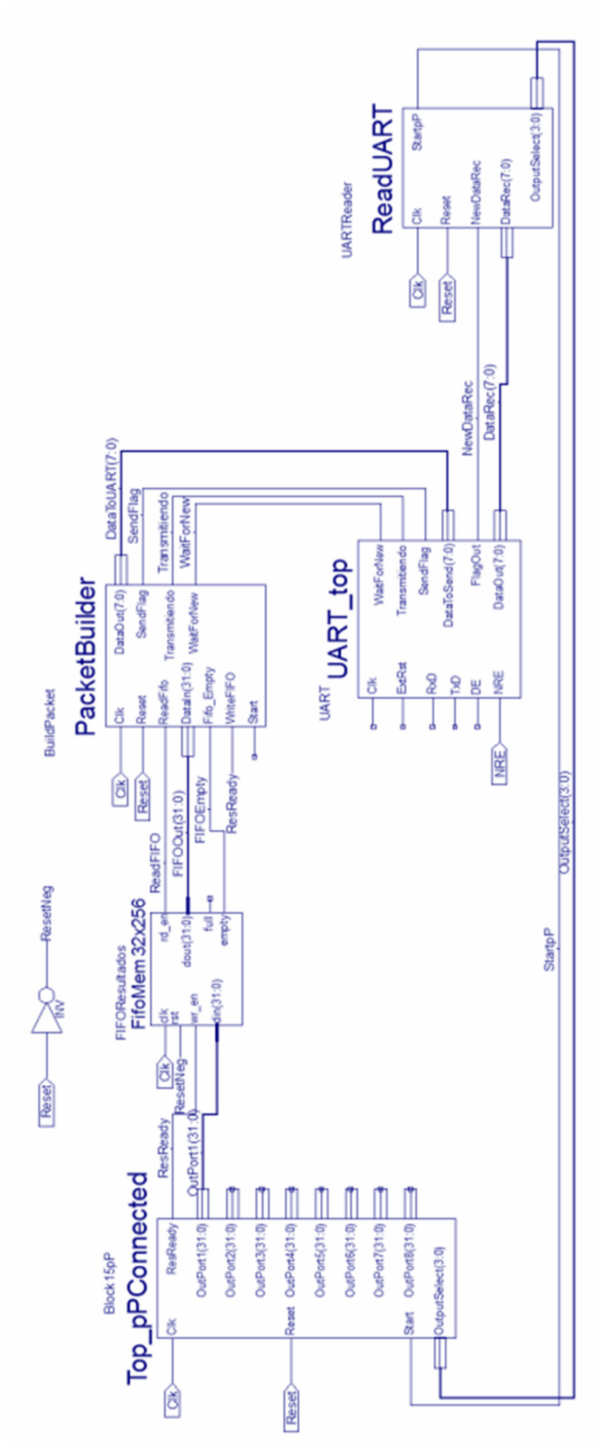


Figure 76 FPGA design for using the RS-232

Appendix VI. SVD notes

The following tables show the memory organization:

READ Processor	MemoryA_1		MemoryA_2	
	DATO	POS	DATO	POS
PD1	X00	0	X01	0
	X11	1	NoData	1
PD2	X22	2	X23	2
	X33	3	NoData	3
PD3	X44	4	X45	4
	X55	5	NoData	5
PD4	X66	6	X67	6
	X77	7	NoData	7
PD5	X88	8	X89	8
	X99	9	NoData	9
PND1	X03	10	X13	10
	X12	11	X02	11
PND2	X04	12	X05	12
	X15	13	X14	13
PND3	X07	14	X17	14
	X16	15	X06	15
PND4	X08	16	X09	16
	X19	17	X18	17
PND5	X25	18	X35	18
	X34	19	X24	19
PND6	X26	20	X27	20
	X37	21	X36	21
PND7	X29	22	X39	22
	X38	23	X28	23
PND8	X47	24	X57	24
	X56	25	X46	25
PND9	X48	26	X49	26
	X59	27	X58	27
PND10	X69	28	X79	28
	X78	29	X68	29

Table 41: Eigenvalue positions

The next table shows the eigenvector positions (taking by columns)

READ Processor	MemoryA_1		MemoryA_2	
	DATO	**POS**	**DATO**	**POS**
PV1	X00	30	X10	30
	X11	31	X01	31
PV2	X20	32	X30	32
	X31	33	X21	33
PV3	X40	34	X50	34
	X51	35	X41	35
PV4	X60	36	X70	36
	X71	37	X61	37
PV5	X80	38	X90	38
	X91	39	X81	39
PV6	X12	40	X13	40
	X03	41	X02	41
PV7	X32	42	X33	42
	X23	43	X22	43

PV8	X52	44	X53	44
	X43	45	X42	45
PV9	X72	46	X73	46
	X63	47	X62	47
PV10	X92	48	X93	48
	X83	49	X82	49
PV11	X04	50	X14	50
	X15	51	X05	51
PV12	X24	52	X34	52
	X35	53	X25	53
PV13	X44	54	X54	54
	X55	55	X45	55
PV14	X64	56	X74	56
	X75	57	X65	57
PV15	X84	58	X94	58
	X95	59	X85	59
PV16	X16	60	X17	60
	X07	61	X06	61
PV17	X36	62	X37	62
	X27	63	X26	63
PV18	X56	64	X57	64
	X47	65	X46	65
PV19	X76	66	X77	66
	X67	67	X66	67
PV20	X96	68	X97	68
	X87	69	X86	69
PV21	X08	70	X18	70
	X19	71	X09	71
PV22	X28	72	X38	72
	X39	73	X29	73
PV23	X48	74	X58	74
	X59	75	X49	75
PV24	X68	76	X78	76
	X79	77	X69	77
PV25	X88	78	X98	78
	X99	79	X89	79

Table 42: Eigenvector position

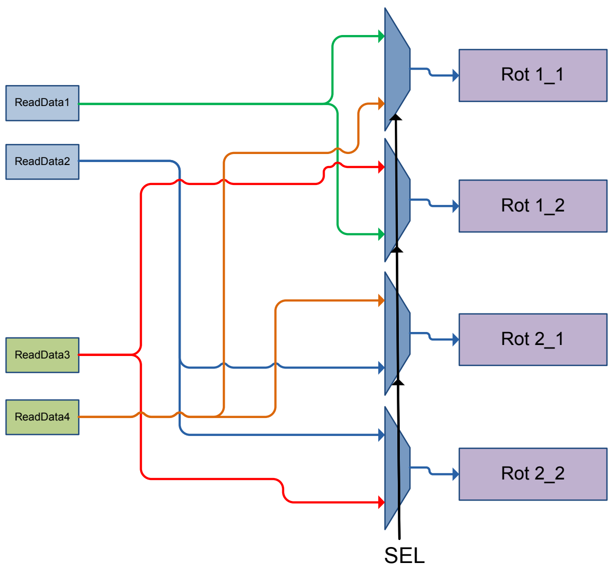


Figure 77 : Reading block: Logic for submatrix reading

The signal “sel” in Figure 77 must change the value in every reading. Values for signals “sel” are shown in Figure 78:

$$\underbrace{\boxed{\text{0-0-0-0-0}}}_{\text{ANGLE}}\underbrace{\boxed{\text{0-0-0-0-0-1-0-1-0-1-0-1-1-0-1}}}_{\text{EIGENVAL}}\underbrace{\boxed{\text{0-0-0-0-0-1-1-1-1-1-0-0-0-0-0-1-1-1-1-1-0-0-0-0-0}}}_{\text{EIGENVECT}}$$

Figure 78: States of the *SEL* signal within the Reading block (see Figure 77)

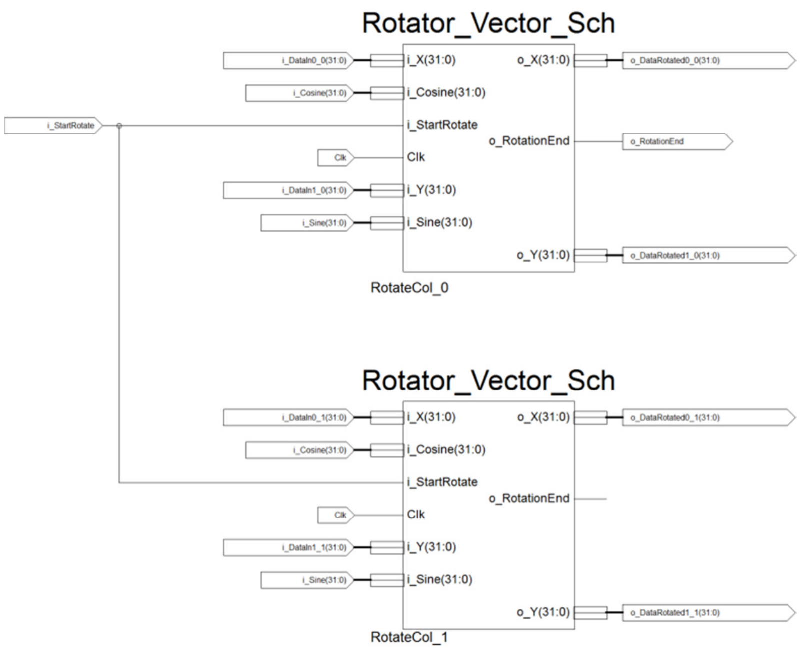


Figure 79: Rotator scheme

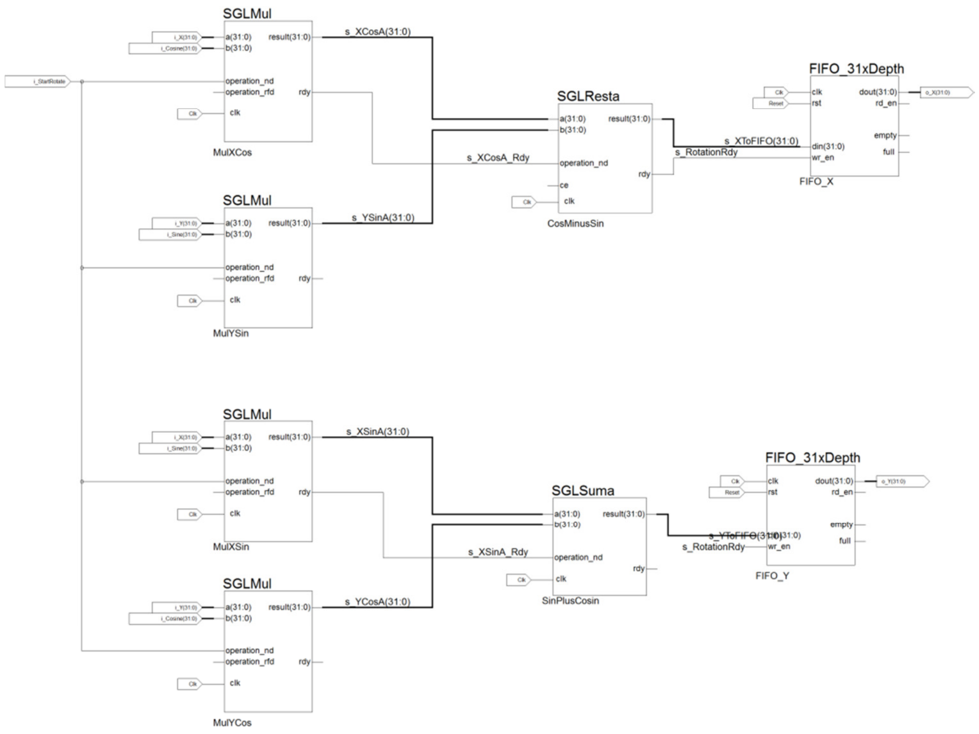


Figure 80: Rotation vector core. Schematic version of Figure 45.

	submatrix	Angle
Eigen Value rotation 1	PD1	1
	PD2	2
	PD3	3
	PD4	4
	PD5	5
	PND1	1
	PND2	1
	PND3	1
	PND4	1
	PND5	2
	PND6	2
	PND7	2
	PND8	3
	PND9	3
	PND10	4
Eigen Vector	PV1	1
	PV2	1
	PV3	1
	PV4	1
	PV5	1
	PV6	2
	PV7	2
	PV8	2
	PV9	2
	PV10	2

	PV11	3
	PV12	3
	PV13	3
	PV14	3
	PV15	3
	PV16	4
	PV17	4
	PV18	4
	PV19	4
	PV20	4
	PV21	5
	PV22	5
	PV23	5
	PV24	5
	PV25	5
Eigen value rotation 2	PD1	1
	PD2	2
	PD3	3
	PD4	4
	PD5	5
	PND1	2
	PND2	3
	PND3	4
	PND4	5
	PND5	3
	PND6	4
	PND7	5
	PND8	4
	PND9	5
	PND10	5

Table 43: Order of the angle selection within the Rotator block

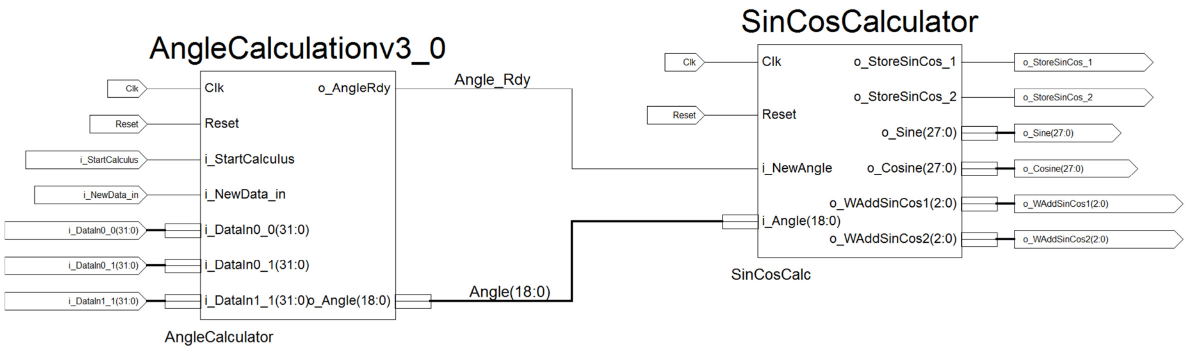


Figure 81: Schematic of the Angle Calculation Block,

To calculate the angle, the block “AngleCalculation” uses the CORDIC core. Therefore the arctangent is obtained. But the CORDIC core can only work with fixed point data. In order to avoid loss of precision, all the necessary operations before going into the CORDIC block are in 32bits floating point data. Before going into the CORDIC, a conversion is done to 40 bits fixed point data with two bits integer. The angle obtained from CORDIC is 27 bits fixed point data with three bits integer. Parameters used in the CORDIC core can be found in Table 44

CORDIC Parameter	**Value**
Functional Selection	Arc Tan
Architectural configuration	Parallel
Pipelining mode	Maximum
Phase format	Radians
Input width	40
Register inputs	Yes
Output width	27
Register outputs	Yes
Round mode	Nearest Even
Iterations	0
Precision	48
Coarse rotation	Yes
Optional pins	CE, ND, RDY, Phase Output

Table 44: CORDIC for arctangent implementation options

After calculating the angle, the FPGA calculates the sine and cosine to be stored. This avoids calculating sine and cosine in every rotation. Sine and cosine are calculated by using another CORDIC core. Parameters used in this CORDIC core can be found in Table 45

CORDIC Parameter	**Value**
Functional selection	Sin and Cos
Architectural configuration	Parallel
Pipelining mode	Maximum
Phase Format	Radians
Input width	27
Register inputs	No
Output width	25
Register outputs	No
Round mode	Nearest even
Iterations	0
Precision	48
Coarse rotation	No
Optional pins	ND, RDY, X Out, Y Out

Table 45: Cordic for sine and cosine implementation options

Write Processor	OUT ROT ->Submatriz 2x2			
PD1	1X00	0	NoData	1
	2X02 = ‘0’	11	1X22	2
PD2	1X44	4	NoData	3
	2X14= ‘0’	13	1X11	1
PD3	1X66	6	NoData	5
	2X36= ‘0’	21	1X33	3
PD4	1X88	8	NoData	7
	2X58= ‘0’	27	1X55	5
PD5	1X99	9	NoData	9
	2X79= ‘0’	28	1X77	7
PND1	1X04	12	2X24	19
	2X01	0	1X12	11
PND2	2X06	15	1X26	20
	1X03	10	2X23	2
PND3	1X08	16	2X28	23
	2X05	12	1X25	18
PND4	2X09	16	1X29	22
	1X07	14	2X27	20
PND5	2X46	25	1X16	15
	1X34	19	2X13	10
PND6	1X48	26	2X18	17
	2X45	4	1X15	13
PND7	2X49	26	1X19	17
	1X47	24	2X17	14
PND8	2X68	29	1X38	23
	1X56	25	2X35	18
PND9	1X69	28	2X39	22
	2X67	6	1X37	21
PND10	2X89	8	1X59	27
	1X78	29	2X57	24

Write Processor	MemoryA_1		MemoryA_2	
PV1	1X00	30	1X20 *	32
	2X02 *	40	2X22	42
PV2	1X40	34	2X10 *	30
	2X42 *	44	1X12	40
PV3	1X60	36	2X30 *	32
	2X62 *	46	1X32	42
PV4	1X80	38	2X50 *	34
	2X82 *	48	1X52	44
PV5	2X90	38	2X70 *	36
	1X92 *	48	1X72	46
PV6	1X04	50	1X24 *	52
	2X01 *	31	2X21	33
PV7	1X44	54	2X14 *	50
	2X41 *	35	1X11	31
PV8	1X64	56	2X34 *	52
	2X61 *	37	1X31	33
PV9	1X84	58	2X54 *	54
	2X81 *	39	1X51	35
PV10	2X94	58	2X74 *	56
	1X91 *	39	1X71	37
PV11	2X06	60	2X26 *	62
	1X03 *	41	1X23	43
PV12	2X46	64	1X16 *	60

	1X43 *	45	2X13	41
PV13	2X66	66	1X36 *	62
	1X63 *	47	2X33	43
PV14	2X86	68	1X56 *	64
	1X83 *	49	2X53	45
PV15	1X96	68	1X76 *	66
	2X93 *	49	2X73	47
PV16	1X08	70	1X28 *	72
	2X05 *	51	2X25	53
PV17	1X48	74	2X18 *	70
	2X45 *	55	1X15	51
PV18	1X68	76	2X38 *	72
	2X65 *	57	1X35	53
PV19	1X88	78	2X58 *	74
	2X85 *	59	1X55	55
PV20	2X98	78	2X78 *	76
	1X95 *	59	1X75	57
PV21	2X09	71	2X29 *	73
	1X07 *	61	1X27	63
PV22	2X49	75	1X19 *	71
	1X47 *	65	2X17	61
PV23	2X69	77	1X39 *	73
	1X67 *	67	2X37	63
PV24	2X89	79	1X59 *	75
	1X87 *	69	2X57	65
PV25	1X99	79	1X79 *	77
	2X97 *	69	2X77	67

Table 46 Reallocation ROM

Appendix VII. Communication notes

SoCWire Protocol Write Register ID	RTE Configuration Register name (16 bits)	Description
0	RTE_MODE	Format : unsigned fixed point Values: 0 : Classical estimations + RTE 1 : Classical estimations 2 : RTE (without Classical estimations)
1	RTE_ITERATIONS	Format: unsigned fixed point Values: integer in [0,32]
2	RTE_OUTPUT_MODE	Format: unsigned fixed point Values: Mask of eleven lest significant bits. Each bit position codes if the referenced parameter is in the RTE output values set. One is true and zero is false. Bit 0: the line-to-continuum absorption coefficient ratio (η_0) Bit 1: the strength of the vector magnetic field (B) Bit 2: the line-of-sight (LOS) velocity (v_{LOS}) Bit 3: Doppler width of the line (λ_D) Bit 4: the damping parameter (a) Bit 5: inclination (γ) of the vector magnetic field Bit 6: azimuth (ϕ) of the vector magnetic field Bit 7: source function (S_0) Bit 8: source function (S_1) Bit 9: the continuum intensity (Ic)
3	RTE_INVERTED _PARAMETERS	Format: unsigned fixed point Values: Mask of ten least significant bits. Each bit position codes if the referenced parameter is used in the RTE inversion process. One is true and zero is false. Bit 0: the line-to-continuum absorption coefficient ratio (η_0) Bit 1: the strength of the vector magnetic field (B) Bit 2: the line-of-sight (LOS) velocity (v_{LOS}) Bit 3: Doppler width of the line (λ_D) Bit 4: the damping parameter (a) Bit 5: inclination (γ) of the vector magnetic field Bit 6: azimuth (ϕ) of the vector magnetic field Bit 7: source function (S_0) Bit 8: source function (S_1)
4 **5**	IM_ETHA0	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [8.0,12.0]

		The line-to-continuum absorption coefficient ratio (η_0) is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
6 **7**	IM_B	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [0,4500.0] The strength of the vector magnetic field (B) is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
8 **9**	IM_VL	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [-5.0,5.0] The line-of-sight (LOS) velocity (v_{LOS}) is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
10 **11**	IM_LD	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [2.9e-2,3.2e-2] The Doppler width of the line (λ_D) is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
12 **13**	IM_A	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [2.5e-3,6e-1] The damping parameter (a) is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
14 **15**	IM_GM	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [0,180] The inclination (γ) of the vector magnetic field is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
16 **17**	IM_AZI	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [0,180.0] The azimuth (ϕ) of the vector magnetic field is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
18 **19**	IM_S0	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [1e-1,1e0]

		The source function (S_0) is used in the initial atmosphere model for the RTE inversion mode without classical estimations.
20 **21**	IM_S1	Format: floating point of 32 bits split in two SoCWire Register of 16 bits. The LSB in the lower register and MSB in the higher register. Values: [1e-1,9e-1] The source function (S_1) is used in the initial atmosphere model for the RTE inversion mode without classical estimations.

Table 47 SoCWire Protocol Write Registers used to configure the RTE inverter (see Chap. 8)

SoCP Read Register ID	RTE Status Register (16 bits)	Description
28	Status_Register_1	The Internal and the accepted configuration flag
29	Status_Register_2	The result of the check process for the configuration
30,31	Statistical_Register_1	The current statistical value "χ_i Square" (32 bits floating point)

Table 48 RTE status register (see Chapter 8)

Appendix VIII. Documentation packets

Preliminary Design Review (PDR) documentation:

- SOL-PHI-IAA-SW3100-PL-1 RTE FPGA Development Plan
- SOL-PHI-IAA-SW3100-PL-2 RTE inverter FPGA validation and verification plan
- SOL-PHI-IAA-SW3100-RP-1 RTE inverter FPGA Architecture Design Report
- SOL-PHI-IAA-SW3100-SP-1 RTE inverter FPGA requirement specification

Critical Design Review (CDR) documentation

- SOL-PHI-IAA-SW3100-PR-1 nProcessor net module post programming test procedure
- SOL-PHI-IAA-SW3100-PR-1 Classical estimations functional post programming test procedure
- SOL-PHI-IAA-SW3100-RP-1 Classical estimations functional post programming test report
- SOL-PHI-IAA-SW3100-RP-1 nProcessor net module post programming test report
- SOL-PHI-IAA-SW3100-RP-5 RTE inverter FPGA detailed Design
- SOL-PHI-IAA-SW3100-PL-1 RTE inverter FPGA Development Plan
- SOL-PHI-IAA-SW3100-RP-7 RTE inverter FPGA progress report
- SOL-PHI-IAA-SW3100-SP-1 RTE inverter FPGA requirement specification
- SOL-PHI-IAA-SW3100-PL-2 RTE inverter FPGA validation and verification plan
- SOL-PHI-IAA-SW3100-PR-2 nProcessor module test procedure
- SOL-PHI-IAA-SW3100-PR-5 nProcessor net module test procedure
- SOL-PHI-IAA-SW3100-PR-6 spectral synthesis and response functions functional test procedure
- SOL-PHI-IAA-SW3100-PR-7 classical estimations functional test procedure
- SOL-PHI-IAA-SW3100-PR-8 convolution functional test procedure
- SOL-PHI-IAA-SW3100-PR-9 covariance matrix functional test procedure
- SOL-PHI-IAA-SW3100-RP-6 nProcessor module verification report
- SOL-PHI-IAA-SW3100-RP-10 nProcessor net module verification report
- SOL-PHI-IAA-SW3100-RP-11 spectral synthesis and response functions functional verification report
- SOL-PHI-IAA-SW3100-RP-12 classical estimations functional verification report
- SOL-PHI-IAA-SW3100-RP-13 convolution functional verification report
- SOL-PHI-IAA-SW3100-RP-14 covariance matrix functional verification report

Electrical and Functional Model (EFM) documentation:

- SOL-PHI-IAA-SW3100-LI-1 RTE FPGA EFM CIDL
- SOL-PHI-IAA-SW3100-LI-2 RTE inverter FPGA EFM Traceability, Verification and Compliance Matrix for RTE inverter FPGA specification
- SOL-PHI-IAA-SW3100-PL-1 RTE inverter FPGA Development Plan
- SOL-PHI-IAA-SW3100-PR-18 RTE inverter EFM Model Test Procedure
- SOL-PHI-IAA-SW3100-RP-5 RTE inverter FPGA detailed Design
- SOL-PHI-IAA-SW3100-RP-17 RTE Inverter FPGA Power Test Report
- SOL-PHI-IAA-SW3100-RP-19 RTE inverter EFM Model Test Report
- SOL-PHI-IAA-SW3100-SP-1 RTE inverter FPGA requirement specification

Appendix IX. Virtex-4 FPGA development board

This card has been developed specifically for integration into this system. The design was developed jointly with the company ‘Seven Solutions SL’.

The board implements a commercial device Virtex 4, in particular the device XC4VSX55, package FF1148, speed: - 10. It is equipped with the most common laboratory elements to perform a debugging of a FPGA device programming as Figure 81 illustrates.

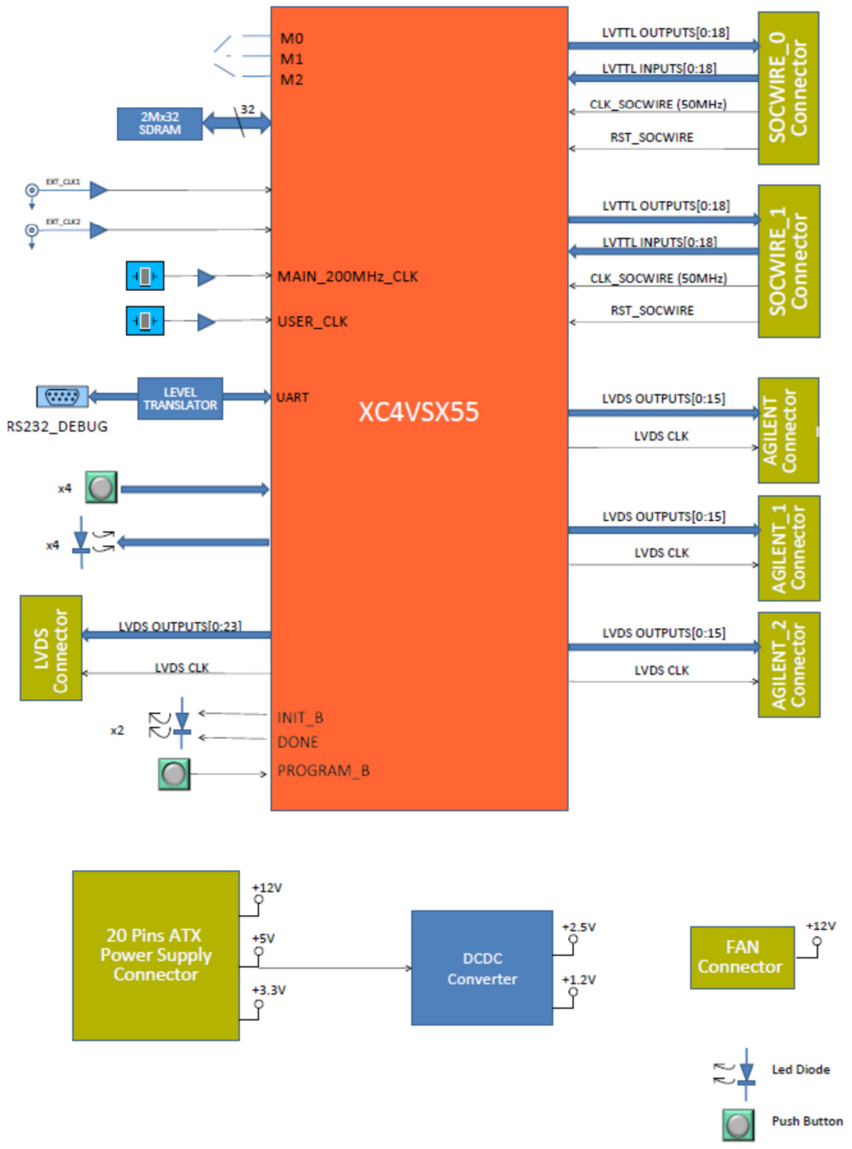


Figure 82: Block diagram development board

We highlight the following characteristics:

1) Different programmable configurations of clock. The board has two clocks of programmable frequency via PPL (cdcm61002) devices. The PLL devices, in their configuration of LVCMOS outputs, allow us to select, via switch, on frequencies in a range from 60 to 250 MHZ of clock signal. It also has possibility for connection of two external clock inputs through SMA socket.
2) It allows us to integrate communications buses serial, RS-232 (MAX3232) and USB (CP2102) for debugging.
3) Makes it possible to integrate 2 SoCWire communication buses as required by the final design . The connection is made through a typical logic analyser cable, in particular the National Instruments SHC68-H1X38, specified to 100 MHz.
4) It has 1 Connector for 34 commonly used LVDS signals.
5) It has 1 Connector for one probe-differential (E5387A), by Agilent.
6) It has 2 specific ports for probe-single-ended (E5390A) by Agilent (ChipScope, Xilinx FPGA Logic Analyser) [Xilinx]
7) It has switches, leds and buttons of general use for programming debugging.
8) The VHDL test can be loaded through a Xilinx JTAG bus or can be loaded automatically after power up from a SDRAM (2Mx32) memory bank.

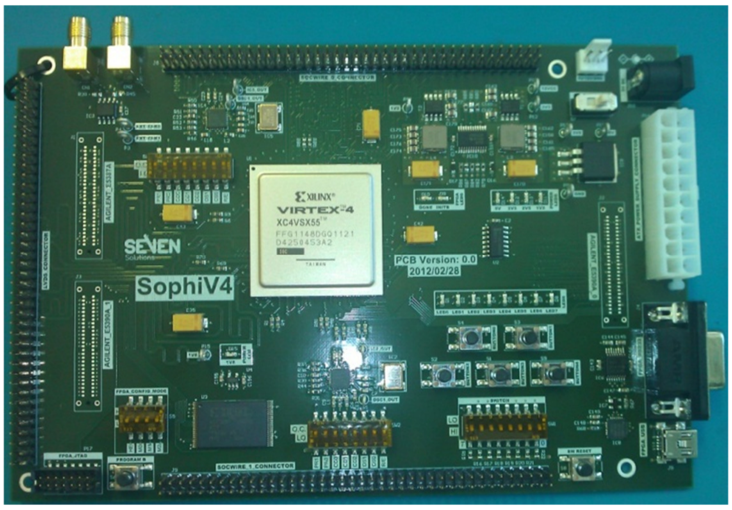


Figure 83: Virtex-4 Development Board

Appendix X. Publications and other merits

Almost 30 technical notes have been released within the SO/PHI project. Below is a list of some publications and contributions to scientific meetings.

Papers in JCR journals:

- **Cobos Carrascosa, J.P.;** Ramos, J.L.; Aparicio del Moral, B.; Balaguer M.; Lopez Jimenez, A.C.; del Toro Iniesta, J.C. "SIMD architecture on FPGA for scientific computing aboard a space instrument". Journal of Systems Architecture (Elsevier). In press.

  Sited in Q2, in 2014, in the category of "Hardware and Architecture" of the SCImago Journal Rank.

Papers in international conferences with double blind review:

- **Cobos Carrascosa, J.P.;** Aparicio del Moral, B.; Ramos, J.L.; Balaguer M.; Lopez Jimenez, A.C.; del Toro Iniesta, J.C. "Scientific Computing and Fault Mitigation on FPGA Aboard the Solar Orbiter PHI Instrument**".** In Proc. of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS'15), Montreal(CA), vol., no., pp.1-8, 15-18 June 2015. Doi: 10.1109/AHS.2015.7231150. IEEE Explore Publisher.
- **Cobos Carrascosa, J.P**.; Aparicio del Moral, B.; Ramos, J.L.; Lopez Jimenez, A.C.; del Toro Iniesta, J.C., "A Multicore Architecture for High-Performance Scientific Computing Using FPGAs," Embedded Multicore/Manycore SoCs (MCSoc), 2014 IEEE 8th International Symposium on , vol., no., pp.223-228, 23-25 Sept. 2014 doi: 10.1109/MCSoC.2014.39 IEEE Explore Publisher.

Paper in national workshops:

- D. Hernández Expósito, **J.P. Cobos Carrascosa**, M. Rodríguez Valido and V. Martínez Pillet, Arquitectura multiprocesador SIMD en FPGA para el cálculo de la 2D-DWT, Jornadas de Computación Reconfigurable y Aplicaciones, Córdoba (España) 2015
- **J.P. Cobos Carrascosa,** J.L. Ramos Mas, B. Aparicio del Moral, A.C. López Jiménez, J.C del Toro Iniesta. *Arquitectura multiprocesador SIMD en FPGA para cálculo científico en instrumentación espacial* Actas del JCRA 2014. "Jornadas de Computación Reconfigurable y Aplicaciones" Valladolid, Septiembre de 2014. ISBN : 978-84-697-0971-9 , pp 106-113.
  (Awarded like the "**Best Paper**")
- **J.P. Cobos**, R. Sanz, J.L. Ramos, J.C. del Toro Iniesta, A.C. López Jiménez, *Diseño de una arquitectura segmentada para computación de altas prestaciones en FPGA*. Actas del JCRA 2009. "IX Jornadas de Computación Reconfigurable y Aplicaciones" Alcalá de Henares, 9-11 de Septiembre de 2009. ISBN : 978-84-8138-833-6 , pp 245-254.

- **J.P. Cobos**, B. Aparicio del Moral, J.L. Ramos, A.C. López Jiménez, *Configuración automática de una arquitectura MIMD empotrada en FPGA*. Actas del SiCE 2010. "I Simposio de Computación Empotrada". Valencia, 7-10 Septiembre 2010. ISBN: 978-84-92812-69-1, pp 57-64

Presentations in workshop without printed proceedings:

- **"FPGAs for embedded computing on space instrumentation"**
  II Jornadas de Computación Empotrada
  Granada, Octubre 2011
  Organized by Universidad de Granada

- **"Progress on the RTE electronic inverter for SO/PHI"**
  Congreso Desarrollo de Instrumentación Espacial
  Madrid, Junio 2011
  Organized by the Centro de Astrobiología (INTA-CSIC)

- **"Progress on the RTE electronic inverter for SO/PHI"**
  III Reunión Española de Física Solar y Heliosférica
  Granada, Junio 2011
  Organized by Instituto de Astrofísica de Andalucía.

Papers as a member of the SO/PHI Team

- Sami K. Solanki, Jose Carlos del Toro Iniesta, Joachim Woch1, Achim Gandorfer, Johann Hirzberger1, Wolfgang Schmidt, Thierry Appourchaux, Alberto Alvarez-Herrero and the **SO/PHI team**, in Polarimetry: From the Sun to Stars and Stellar Environments, K.N. Nagendra, S. Bagnulo, R. Centeno, & M.J. Martínez González (eds.), Proceedings of the International Astronomical Union, Volume 305, pp. 108-113, CUP. 2015.

Academic works related to this thesis in which the author has been advisor:

- "Diseño de Módulos de cómputo en tiempo real para FPGA". José Manuel Martín Rodríguez. Project for obtaining the grade of Ingeniería en Telecomunicaciones. 2013. University of Granada.

- "Implementation of classical methods of magnetic field measurement in an FPGA". Pablo Torné Torres. Master thesis. 2012. University of Granada.

Acronyms

ALU	Arithmetric Logic Unit
ASIC	Application-Specific Integrated Circuit
BCH	Bose and Chaudhuri code
BRAM	Block random access memory
CORDIC	Coordinate Rotation DIgital Computer
DPU	Data Processing Unit
DSP	Digital Signal Processing
ESA	European Space Agency
ESA	European Space Agency
FFT	Fast Fourier Transform
FIFO	First Input First Output
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
I/F	InterFace
I/O	Input/Output
IDL	Interactive Data Language
IP	Intelectual property
IQUV	Stockes profiles
ISA	Instruction Set Architecture
ISS	Image Stabilization System
LOS	Line-Of-Sight
LSB	Least significant bits
ME	Milne-Eddington
MILOS	MILne-Eddington inversion of pOlarized Spectra
MIMD	Single Instruction stream Multiple Data stream
MMC	Multiport Memory Controller
MPP	Massive Parallel Processing
MSB	Most significant bits
NoC	Network On Chip
nP	nanoProcessor
nPCU	nano Processor Control Unit
nProcessor	nanoProcessor
PD	Processor-Diagonal (Brent's Algorithm)
PMP	Polarization Modulation Package
PND	Processor-Non-Diagonal (Brent's Algorithm)
pP	pProcessor
pProcessor	picoProcessor
PV	Processor-Vectorial (Brent's Algorithm)
QA/PA	Quality assurance/Product Assurance
RAM	Random Access Memory
RF	Response Functions
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTE	Radiative Transfer Equation
RTL	Register Transfer Level
SIMD	Single Instruction stream Multiple Data stream
SO	Solar Orbiter

SO/PHI	Polarimetric and Helioseismic Imager for Solar Orbiter
SOB	Shared Operations Block
SoC	System On Chip
SoCP	SoCWire Protocol
SSRFA	Spectral Synthesis and Response Functions Algorithm
SVD	Singular Value Decomposition
VHDL	Very high speed integrated circuit Hardware Description Language

References

[Allan (1995)] Allan Vicki H. “Software pipelining”. ACM Computing Surveys, 1995. Vols. 27 Issue 3: pp. 367 - 432.

[Amdahl (1967)] Amdahl , Amdahl G., Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities // AFIPS Conference Proceedings. - 1967. - Vol. 30. - págs. 483-485.

[Acton (1992)] Acton, L., Tsuneta, S., Ogawara, Y., Bentley, R., Bruner, M., Canfield, R., Culhane, L., Doschek, G., Hiei,

E., Hirayama, T.: 1992, The Yohkoh mission for high-energy solar physics. Science 258, 618 – 625.

[Allen (2008)] Allen G., Swift G., and Carmichael C., “Virtex-4qv static seu characterization summary,” Jet Propulsion Laboratory Pasadena, California, Tech. Rep., 2008.

[Altera (2007)] Altera Inc. Accelerating High-Performance Computing With FPGAs. White Paper WP-01029-1.1. 2007

[Aparicio (2010)] B. Aparicio, J.P. Cobos. picoProcessor architecture. SOL-PHI-IAA-SW3100-RP-5. 2010

[Aparicio (2013)] B. Aparicio, J.P. Cobos, J.L. Ramos. RTE inverter FPGA development plan. SOL-PHI-IAA-SW3100-PL-1

[Aparicio (2014)] B. Aparicio, J.P. Cobos, J.L. Ramos. RTE inverter FPGA requirement specification. SOL-PHI-IAA-SW3100-SP-1

[Baklouti (2010)] Baklouti M. et al. Scalable mpNoC for massively parallel systems – Design and implementation on FPGA. Journal of Systems Architecture, July 2010.

[Bolotski (1994)] Bolotski M., et al. Unifying FPGAs and SIMD Arrays. Workshop on Field Programmable Gate Arrays 1994.

[Borkar (2010)]S. Borkar, “Thousand Core Chips—A Technology Perspective,” Proc. ACM/IEEE 44th Design Automation Conf. (DAC), ACM Press, 2007, pp. 746-749.

[Borrero (2010)] Borrero J.M. e al. VFISV: Very Fast Inversion of the Stokes Vector for the Helioseismic and Magnetic Imager. 2010, Sol. Phys., 35.

[Bose (1960)] Bose, R. C.; Ray-Chaudhuri, D. K. (March 1960), "On A Class of Error Correcting Binary Group Codes", Information and Control 3 (1): 68–79,

[Bubenhagen (2010)] F. Bubenhagen, B. Fiethe, J. Ilstad, H. Michalik, P. Norridge, B. Osterloh, W. Sullivan, C. Topping, Enhanced Dynamic Reconfigurable Processing Module for

Future Space Applications, International SpaceWire Conference, pp. 475-482, June 2010

[Bubenhagen (2013)] Bubenhagen, F.; Fiethe, B.; Lange, T.; Michalik, H.; Michel, H., "Reconfigurable platforms for Data Processing on scientific space instruments," in Adaptive Hardware and Systems (AHS), 2013 NASA/ESA Conference on , vol., no., pp.63-70, 24-27 June 2013

[Bravo (2006)]. Bravo, I.; Jimenez, P.; Mazo, M.; Lazaro, J.L.; Gardel, A., "Implementation in Fpgas of Jacobi Method to Solve the Eigenvalue and Eigenvector Problem," in Field Programmable Logic and Applications, 2006. FPL '06. International Conference on , vol., no., pp.1-4, 28-30 Aug. 2006

[Bravo (2007)] Bravo I., Arquitectura basada en FPGA para la detección de objetos en movimiento, utilizando visión computacional y PCA. Thesis dissertation, 2007.

[Bravo (2008)] Bravo, I., Mazo, M., et al., "Novel HW Architecture Based on FPGAs Oriented to Solve the Eigen Problem," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.16, no.12, pp.1722,1725, Dec. 2008

[Brent (1983)] R. P. Brent, F. T. Luk and C. F. Van Loan, “Computation of the singular value decomposition using meshconnected processors”, J. of VLSI and Computer Systems 1, 3 (1983–1985), 242–270. Also appeared as Report TR 82-528, Department of Computer Science, Cornell University, November 1982; and as Report TR-CS-83-05, Department of Computer Science, ANU, January 1983, 34 pp.

[Brent (1985)] R. P. Brent and F. T. Luk, “The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays”, SIAM J. Scientific and Statistical Computing 6 (1985), 69–84.

[Cavallaro (1987] Cavallaro, 1987. "CORDIC Arithmetic for an SVD processor". Journal of parallel and distributed computing

[Cassel (2011)] M. Cassel, M. Stähle, U. Lonsdorfer, F. Gliem, D. Walter, T. Fichna, The First European Spaceborne Mass Memory System based on NAND-Flash Technology: The Sentinel-2 MMFU, ReSpace / MAPLD 2011 Conference, August 2011

[Cobos (2015)] Cobos Carrascosa, J.P.; Aparicio del Moral, B.; Ramos, J.L.; Balaguer M.; Lopez Jimenez, A.C.; del Toro Iniesta, J.C. “Scientific Computing and Fault Mitigation on FPGA Aboard the Solar Orbiter PHI Instrument”. In Proc. of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS’15), Montreal(CA), vol., no., pp.1-8, 15-18 June 2015. IEEE Explore Publisher.

[Cobos (2014)] Cobos Carrascosa, J.P.; Aparicio del Moral, B.; Ramos, J.L.; Lopez Jimenez, A.C.; del Toro Iniesta, J.C., "A Multicore Architecture for High-Performance Scientific

Computing Using FPGAs," Embedded Multicore/Manycore SoCs (MCSoc), 2014 IEEE 8th International Symposium on , vol., no., pp.223-228, 23-25 Sept. 2014 IEEE Explore Publisher.

[Cobos (2011)] J.P. Cobos, B. Aparicio, RTE inversion on space- qualified configurable devices, Technical Note SOL-PHI-IAA-SW3100-TN-2, September 2011

[Cobos (2013)] J.P. Cobos , J.L. Ramos, B. Aparicio. RTE inverter FPGA validation and verification plan. SOL-PHI-IAA-SW3100-PL-2

[Cobos (2010-A)] Cobos J.P. C-MILOS. SOL-PHI-IAA-SW2500-TN-1. 2010.

[Cobos (2010-B)] Cobos J.P. RTE Inversion: computational cost. SOL-PHI-IAA-SW2500-TN-3. 2010.

[Cobos (2014)] Cobos J.P, Aparicio B., Ramos J.L. RTE inverter FPGA detailed Design. SOL-PHI-IAA-SW3100-RP-5. 2014

[Cobos (2015-A)] Cobos Carrascosa, J.P.; Aparicio del Moral, B.; Ramos, J.L.; Balaguer M.; Lopez Jimenez, A.C.; del Toro Iniesta, J.C. "Scientific Computing and Fault Mitigation on FPGA Aboard the Solar Orbiter PHI Instrument". In Proc. of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS'15), Montreal(CA), vol., no., pp.1-8, 15-18 June 2015. IEEE Explore Publisher.

[Cobos (2015-B)] Cobos Carrascosa, J.P.; Ramos, J.L.; Aparicio del Moral, B.; Balaguer M.; Lopez Jimenez, A.C.; del Toro Iniesta, J.C. "SIMD architecture on FPGA for scientific computing aboard a space instrument". Journal of Systems Architecture (Elsevier). In press.

[del Toro (2011] del Toro Iniesta J.C., Torné Torres P., Cobos Carrascosa J.P. Magnetographic and tachographic estimates trhough classical methods. SOL-PHI-IAA- SW3100-TN-1

[del Toro (2003)] del Toro Iniesta, J.C, Introduction to Spectropolarimetry, Cambridge University Press: Cambridge, 2003.

[Dijkstra (1972)] Edsger W. Dijkstra – "The Humble Programmer" AM Turing Award. ACM Turing Lecture. 1972

[Domingo (1995)] Domingo, V., Fleck, B., Poland, A.I.: 1995, The SOHO mission: an overview. Solar Phys. 162, 1 – 37.

[Dwyer (1992)] Dwyer H., Torng H. C., "An out-of-order superscalar processor with speculative execution and fast, precise interrupts". MICRO 25 Proceedings of the 25th annual international symposium on Microarchitecture. 1992.

[Elliot (1988)] C. J. Elliott and J. H. Davenport. “Very-High-Performance Multiple-Instruction Multiple-Data Applications”. Philosophical Transactions of the Royal Society of London.

[Engel (2006) J. Engel, M. Wirthlin, K. Morgan, and P. Graham. Predicting On-Orbit Static Single Event Upset Rates in Xilinx Virtex FPGAs. Los Alamos National Laboratory, September 2006. Pp. 34-51]

[ESA (2008)] Space product assurance ASIC and FPGA development (ECSS-Q-ST-60-02C)

[Fiethe (2012)] Fiethe, B.; Bubenhagen, et al., "Adaptive hardware by dynamic reconfiguration for the Solar Orbiter PHI instrument," NASA/ESA Conference on Adaptive Hardware and Systems (AHS), 2012

[Fiethe (2014)] Björn Fiethe SOL-PHI-IDA-PA4200-PL-1 Virtex-4 FPGA CF1140 assembly qualification plan - Company Confidential -

[Flyn (1966)] Flyn M. J., Very High-Speed Computing Systems // Proc of the IEEE. - 1966. - Vol. 54. - págs. 1901-1909. - 12.

[Florent (2005)] Florent de Dinechin, Arnaud Tisserand: Multipartite Table Methods, 2005. IEEE Transactions on Computers, vol. 54, no. 3, marzo, 2005.

[Forsythe (1977)] Forsythe, G.E., Malcolm, M.A., and Moler, C.B. 1977, Computer Methods for Mathematical Computations (Englewood Cliffs, NJ: Prentice-Hall), Chapter 9.

[Fossati (2011)] L. Fossati, J. Illstad, The Future of Embedded Systems at ESA: towards Adaptability and Reconfigurability, In NASA/ESA Conference on Adaptive Hardware and Systems, Volume (AHS-2011), pp 113-120, June 2011

[Fuller (2000)] Fuller E., et al., Radiation testing update, seu mitigation, and availability analysis of the Virtex FPGA for space re-configurable computing. International Conference on Military and Aerospace Programmable Logic Devices, 2000.

[Green500] Top Green 500. http://www.green500.org/

[Gokhale (1995)] Gokhale M., Schott B.. Data parallel C on a reconfigurable logic array. Journal of Supercomputing 9(3), 1995.

[Golub (1974)] Golub, G.H., and Van Loan, C.F. 1989, Matrix Computations, 2nd ed. (Baltimore: Johns Hopkins University Press), §8.3 and Chapter 12. Lawson, C.L., and Hanson, R. 1974, Solving Least Squares Problems (Englewood Cliffs, NJ: Prentice-Hall), Chapter 18.

[Götze (1993)] Gotze, J.; Paul, S.; Sauer, M., "An efficient Jacobi-like algorithm for parallel eigenvalue computation," in Computers, IEEE Transactions on , vol.42, no.9, pp.1058-1065, Sep 1993

[Handy (1999)] Handy, B.N., Acton, L.W., Kankelborg, C.C.,Wolfson, C.J., Akin, D.J., Bruner, M.E., Caravalho, R., Catura, R.C., Chevalier, R., Duncan, D.W., Edwards, C.G., Feinstein, C.N., Freeland, S.L., Friedlaender, F.M., Hoffmann, C.H., Hurlburt, N.E., Jurcevich, B.K., Katz, N.L., Kelly, G.A., Lemen, J.R., Levay,M., Lindgren, R.W., Mathur, D.P., Meyer, S.B., Morrison, S.J., Morrison, M.D., Nightingale, R.W., Pope, T.P., Rehse, R.A., Schrijver, C.J., Shine, R.A., Shing, L., Strong, K.T., Tarbell, T.D., Title, A.M., Torgerson, D.D., Golub, L., Bookbinder, J.A., Caldwell, D., Cheimets, P.N., Davis,W.N., Deluca, E.E., McMullen, R.A., Warren, H.P., Amato, D., Fisher, R., Maldonado, H., Parkinson, C.: 1999, The transition region and coronal explorer. Solar Phys. 187, 229 – 260.

[Hernández (2015)] D. Hernández Expósito, J.P. Cobos Carrascosa, M. Rodríguez Valido and V. Martínez Pillet, Arquitectura multiprocesador SIMD en FPGA para el cálculo de la 2D-DWT, Jornadas de Computación Reconfigurable y Aplicaciones, Córdoba (España) 2015

[Hennessy (1999)] Hennessy J. Gupta A., Heinrich M. Cache,Coherent Distributed Shared Memory: Perspectives on Its Development and Future Challenges // Proceedings of the IEEE. Special Issue on Distributed Shared Memory. - 1999.

[Hirzberger (2012)] J. Hirzberger. 2012. PHI data processing pipeline. SOL-PHI-MPS-SW3200-TN-1

[Hirzberger (2013-A)] J. Hirzberger, 2013, PHI Instrument Requirements Specification. SOL-PHI-MPS-DE2000-SP-1

[Hirzberger (2013-B)] J. Hirzberger, D. Orozco Suarez, Julian Blanco Rodríguez PHI Science Performance Report. SOL-PHI-MPS-DE1000-RP-1, issue 1, revision 0, 2013-11-11.

[Hirzberger (2015)] Hirzberger J. 2015. SOL-PHI-MPS-OP3000-MA-1. PHI Instrument User Manual

[Householder (1970)] Householder, A.S. 1970, The Numerical Treatment of a Single Nonlinear Equation (New York: McGraw-Hill).

[Intel] Intel Corporation. http://ark.intel.com/

[Jacobi (1846)] Jacobi, C. G. J.: Über ein leichtes Verfahren, die in der Theorie der Säkularstörungen vorkommenden Gleichungen numerisch aufzulösen. Crelle's Journal 30, 51--94 (1846)

[Kahan (1987)] William Kahan. "Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic" 1987

[Kaiser (2008)] Kaiser, M.L., Kucera, T.A., Davila, J.M., St. Cyr, O.C., Guhathakurta, M., Christian, E.: 2008, The STEREO mission: An introduction. *Space Sci. Rev.* **136**, 5 – 16.

[Kosugi (2007)] Kosugi, T., Matsuzaki, K., Sakao, T., Shimizu, T., Sone, Y., Tachikawa, S., Hashimoto, T., Minesugi, K., Ohnishi, A., Yamada, T., Tsuneta, S., Hara, H., Ichimoto, K., Suematsu, Y., Shimojo, M., Watanabe, T., Shimada, S., Davis, J.M., Hill, L.D., Owens, J.K., Title, A.M., Culhane, J.L., Harra, L.K., Doschek, G.A., Golub, L.: 2007, The Hinode (Solar-B) mission: An overview. Solar Phys. 243, 3 – 17.

[Lam (1988)] Lam M. “Software pipelining: An effective scheduling technique for VLIW machines”. Conference on Programming Language Design and Implementation. - 1988. - pp. 318-328.

[Learn (2011)] Learn M. Evaluation of Soft-Core Processors on a Xilinx Virtex-5. Sandia National Laboratories. SAND2011-2733.

[Lin (2002)] Lin, R.P., et al.: 2002, The Reuven Ramaty High-Energy Solar Spectroscopic Imager (RHESSI). Solar Phys. 210, 3 – 32.

[Mahmood (2011)] Mahmood, B.S.; Al Jbaar, M.A. Design and implementation of SIMD Vector Processor on FPGA. ISIICT, 2011

[Marsden (2011)]Marsden, R.G., Müller, D.: 2011, Solar Orbiter definition study report, ESA/SRE(2011)14

[Meller (2013)] Meller R. 2013. SO-PHI-MPS-SY1200-DR-01, Functional Diagram

[Meller (2013-B)] Meller R. 2013. SOL-PHI-MPS-SY1800-PL-1_3_3 Engineering Plan

[Michel (2013)] Michel, H.; Bubenhagen, et al., "Worst case error rate predictions and mitigation schemes for Virtex-4 FPGAs on solar orbiter," NASA/ESA Adaptive Hardware and Systems, 2013, pp.1,8, 24-27 June 2013

[Moler (1986)] Moler, C. "Matrix Computation on Distributed Memory Multiprocessors". Hypercube Multiprocessors 1986

[Müller (2013)] D. Müller · R.G. Marsden · O.C. St. Cyr · H.R. Gilbert · The Solar Orbiter Team. Solar Orbiter. Exploring the Sun–Heliosphere Connection Solar Phys (2013) 285:25–70

[Nvidia] NVIDIA Corporation. www.nvidia.com

[Orozco (2007)] Orozco Suárez, D., and del Toro Iniesta, J.C., The usefulness of analytic response functions, 2007, Astronomy and Astrophysics, 462, 1137

[Orozco (2008)] Orozco D. Diffraction-limited spectropolarimetry of quiet-Sun magnetic fields”. Doctoral Thesis. 2008.

[Osterloh (2008)] Osterloh, B.; Michalik, et al., SoCWire: A Network-on-Chip Approach for Reconfigurable System-on-Chip Designs in Space Applications, Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on , June 2008

[Ottoni (2005)]Ottoni, G. et al. “Automatic thread extraction with decoupled software pipelining”. MICRO-38. Proceedings. 38th Annual IEEE/ACM International Symposium on Microarchitecture, 2005

[Orozco (2008)] Orozco D. Diffraction-limited spectropolarimetry of quiet-Sun magnetic fields”. Doctoral dissertation. 2008.

[Pesnell (2012)] Pesnell,W.D., Thompson, B.J., Chamberlin, P.C.: 2012, The Solar Dynamics Observatory (SDO). Solar Phys. 275, 3 – 15.

[Patterson (2000)] Patterson David A., Hennessy John L. and Canal Corretger Ramón, Estructura y Diseño de Computadores: Editorial Reverte S.A, 2000. - Cap 7. Pp 529 - 539. - 8429126171.

[Porsche (1977)] Porsche, H.: 1977, General aspects of the mission *Helios 1* and *2*. Introduction to a special issue on initial scientific results of the *Helios* mission. *J. Geophys.* **42**, 551 – 559.

[Press (1992)] Press W.H. et al., Numerical recipes in C (2nd ed.): the art of scientific computing. Cambridge University Press. 1992.

[Randell (1964)] Randell, B. and Russell, L.J. "Algol 60 Implementation" London: Academic Press, 1964. ISBN 0-12-578150-4.

[Ramos (2011)] Jose Luis Ramos, Juan Pedro Cobos, Beatriz Aparicio, FPGA: Trigonometric Functions. SOL-PHI-IAA-SW2500-TN-2

[Ramos (2011)] Jose Luis Ramos, Beatriz Aparicio , Juan Pedro Cobos, (SVD-FPGA): Brent’s Algorithm Description SOL-PHI-IAA-SW3100-TN-9

[Ramos (2013)] RTE inverter FPGA: test system in hardware. 2013. SOL-PHI-IAA-SW3100-TN-10

[Ramos (2014)] J.L. Ramos, J.P. Cobos, B. Aparicio, RTE inverter FPGA: hardware and software I/F. SOL-PHI-IAA-SW3100-TN-11. 2014

[Ramos (2014-B)] José Luis Ramos, Juan Pedro Cobos, Beatriz Aparicio. RTE inverter FPGA: power consumption test report. SOL-PHI-IAA-SW3100-RP-17

[Rollins (2010)] N. Rollins, M. Fuller, and M.J. Wirthlin. A Comparison of FaultTolerant Memories in SRAM-based FPGAs. In 2010 IEEE Aerospace Conference

[Roma (2012)] Roma, N., Magalhaes, P., System-level prototyping framework for heterogeneous multi-core architecture applied to biological sequence analysis. IEEE ISRSR 2012.

[Scherrer (2002)] Scherrer, P. H., et al, Sol. Phys., 2002. P. 275- 207

[Schwenn (1990)] Schwenn, R., Marsch, E.: 1990, Physics of the Inner Heliosphere I. Large-Scale Phenomena, Physics and Chemistry in Space 20, Springer, Berlin.

[Schwenn (1991)] Schwenn, R.,Marsch, E.: 1991, Physics of the Inner Heliosphere II. Particles, Waves and Turbulence, Physics and Chemistry in Space 21, Springer, Berlin. Sheeley, N.R. Jr.: 1991, Polar faculae – 1906 – 1990.

[Siegel (1979)] Siegel, H.J. A Model of SIMD Machines and a Comparison of Various Interconnection Networks., Transactions on Computers, no.12, pp.907,917, Dec. 1979

[Socas-Navarro (2001)] Socas-Navarro, H. 2001, in Advanced Solar Polarimetry - Theory, observation, and instrumentation, ASP Conf. Ser. 236, p.487

[SoCWire] System-on-chip Wire. http://socwire.org/

[Solanki (2015)] Sami K. Solanki, Jose Carlos del Toro Iniesta, Joachim Woch1, Achim Gandorfer, Johann Hirzberger1, Wolfgang Schmidt, Thierry Appourchaux, Alberto Alvarez-Herrero and the SO/PHI team, in Polarimetry: From the Sun to Stars and Stellar Environments, K.N. Nagendra, S. Bagnulo, R. Centeno, & M.J. Martínez González (eds.), Proceedings of the International Astronomical Union, Volume 305, pp. 108-113, CUP. 2015

[Stanley (2003)] Li Stanley Y. C. et al. FPGA-based SIMD Processor. 11th IEEE S FPCCM 2003.

[Stone (1977)] Stone, E.C.: 1977, The *Voyager* missions to the outer system. *Space Sci. Rev.* **21**, 75.

[Swift (2008)] G. Swift, G. Allen, C. W. Tseng, C. Carmichael, G. Miller, and J. George, “Static upset characteristics of the 90nm Virtex-4QV FPGAs,” in Radiation Effects Data Workshop, 2008 IEEE, July 2008, pp. 98–105.

[SoCWire] System-on-chip Wire. http://socwire.org/

[Sravanthi (2014)]A review of High Performance Computing. G.Sravanthi , B.Grace2 , V.kamakshamma3 IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 1, Ver. VII (Feb. 2014), PP 36-43

[Torné (2012)] Torné P. “Implementation of classical methods of magnetic field measurement in an FPGA” Master thesis. 2012. University of Granada

[Tong (2006)] Tong, J.G., et al. Soft-Core Processors for Embedded Systems, International Conference on Microelectronics, 2006.

[TOP500] TOP500 Supercomputer Sites. www.top500.org

[Volder (1959]Jack E. Volder, The CORDIC Trigonometric Computing Technique, IRE Transactions on Electronic Computers, pp. 330–334, September 1959

[Veen (1986)] Veen A. Dataflow machine architecture. ACM Computing Surveys, Vol. 18, No. 4, December 1986

[Wenzel (1992)] Wenzel, K.P., Marsden, R.G., Page, D.E., Smith, E.J.: 1992, The Ulysses mission. Astron. Astrophys. Suppl. Ser. 92, 207.

[Wilkinson (1999)] Wilkinson B. And Michael Allen, “Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers”, Prentice Hall, 1999.

[Wilkinson 1988] The Algebraic Eigenvalue Problem. J. H. Wilkinson (Ed.). Oxford University Press, Inc., New York, NY, USA. 1988

[Xilinc] Xilinx Inc. www.xilinx.com

[Xu (2003)] Xu X. and Ziavras S.G. A Configurable and Scalable SIMD Machine for Computation-Intensive Applications. Transactions on Computers, Oct. 2003.

[Yiannacouras (2012)] Yiannacouras, P. et al. Portable, Flexible, and Scalable Soft Vector Processors. IEEE Transactions on VLSI. Aug. 2012

[Zima (1988)] Zima H., et al. SUPERB: A tool for semi-automatic MIMD/SIMD parallelization. Parallel Computing, January 1988.