AGGA-4 User Manual

Approval Information:

Name/Role:	Date:
BECKER, Thomas (OTN) / Product Assurance	May 02, 2019 03:01:07 PM GMT
LEMBKE, Erik [DE] / Project Manager	Apr 30, 2019 01:14:39 PM GMT

Left intentionally blank

AT7991 (AGGA-4)
User Manual

Prepared by:	Jens Heim – TSPTC6 *System Engineer GNSS*		**Date:**
Checked by:	Isaac Tejerina – TSPTC *System Engineer ASIC*		**Date:**
Approved by:	Thomas Becker – TOQSP1 Quality *Assurance*		**Date:**
Released by:	Erik Lembke – TSPAC3 *Project Manager*		**Date:**

Document Revision History

Revision	Date	Responsible	Modifications / Reasons for Change
Issue 1	26. Apr. 2019	Jens Heim	Initial Release (based on AGGA-4 Datasheet 1.1) • Update of Section 1.1 Introduction • Added Note about cache snooping issue to 4.9.2.3 • Added Cache Snooping Know Issue chapter 8.4.11 • SYS_CLK_DIV Clarification in chapter 6.3 • Removed min. timings from **Table 6-2**, covered by ATMEL datasheet • Added new applicable document to section 1.5.1 • Added new reference document to section 1.5.2 • Added link to AGGA-4 Radiation test to 8.2 • Update of Section 8.4.7 • Corrected Table reference in 3.4.3.1 • Renamed space to taps in 3.4.4.4 to be consistent • Editorial Correction in 7.4.5.17 • Corrected Code RAM description in 3.4.3.2 • Added ME/IE CarrObs/Phase to **Figure 3-31** • Update of section 1.2

Table of Contents

Contributions

This document contains contributions by

- Deimos Portugal
- Gustavo López-Risueño (ESA)
- Isaac Tejerina (Airbus DS)
- Jens Heim (Airbus DS)
- Josep Rosello (ESA)
- Roland Weigand (ESA)
- Paul Rastetter (Airbus DS)
- RUAG Austria
- Stephan Fischer (Airbus DS)

1 Introduction

1.1 Purpose

The purpose of this document is to describe how the AGGA-4 works and how its features can be used by software.

Please be aware that this document complements the ATMEL AT7991 datasheet. Timings, Electrical and Mechanical specifications as well as pinout is given in the ATMEL datasheet. In case there is overlap, the ATMEL datasheet shall prevail.

Note: Usually the description in the following chapters only wants to give a description of how the individual functional blocks work. It typically does not state every detail how e.g. a bit field can be programmed. This information is listed in the programming section of the corresponding functional block. Therefore it is recommended in order to get the full picture of a functional block to look at both information sources, the functional description as well the programming table. The user is also advised to read chapter 8.4 Known Implementation Issues and Suggested Work Arounds before actually making use of the AGGA-4.

The history of the development is detailed in [RD-09].

1.2 Open Points

The following points are coming out of the AGGA-4 ASIC Validation. Before usage of these functions, Airbus shall be contacted.

1. Phase missaligment between DDC main and DDC aux
2. Real to Complex FIR Filter
3. Dual UART operation

1.3 Notation

Normal Times New Roman font is used in this document.

When *Italics* and Capital and Small letters are used it refers to a *register* (e.g. *DACtrl*) or to a *bitfield* in a register or to an internal clock domain (e.g. *CoreClk*, *HalfSampleClk*, *SysClk*).

When *Italics* and only Capital letters are used it refers to a *pin* (e.g. *EXT_SYS_CLK)*.

Sometimes **Bold** is used to highlight something

1.4 Acronyms

A/D	Analogue to Digital
ADC	A/D Converter
AGC	Automatic Gain Control
AGGA	Advanced GPS Galileo ASIC
AHB	AMBA High Performance Bus
AltBOC	Alternative BOC
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
ASC	Antenna Switch Control
ASE	Antenna Switch Epoch
ASEI	Antenna Switch Epoch Input
ASEO	Antenna Switch Epoch Output
ASIC	Application Specific Integrated Circuit

ATPG	Automated Test Pattern Generation
AUT	Aiding Unit Trigger
BIST	Build In Self Test
BOC	Binary Offset Carrier
CBOC	Complementary BOC
CDL	Code Delay Line
CRC	Cyclic Redundancy Check
D/A	Digital to Analogue
DBF	Digital Beam Forming
DDC	Digital Down Conversion
DMA	Direct Memory Access
EC	Epoch Clock
EDAC	Error Detection And Correction
FDC	Final Down Converter
FE	Front End
FFT	Fast Fourier Transform
FIFO	First In First Out
FIT	Failure in Time
FP	Floating Point
FPU	Floating Point Unit
GNSS	Global Navigation Satellite System
GPIO	General Purpose Input / Output
GRFPU	(Cobham) Gaisler Research Floating Point Unit
I /O	Input / Output
I/F	Interface
IE	Integration Epoch
IF	Intermediate Frequency
IFC	Input Format Converter
IM	Input Module
IMT	Instrument Measurement Time
IMU	Inertial Measurement Unit
IPN	Internal Problem Notice
JTAG	Joint Test Action Group
LE	Long Epoch
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look Up Table
ME	Measurement Epoch
MEI	Measurement Epoch Input
MEO	Measurement Epoch Output
MILBUS	MIL-STD-1553B
MSB	Most Significant Bit
MUX	Multiplexer
NCO	Numerically Controlled Oscillator
PIC	Primary Interrupt Controller
PLD	Power Level Detector
PPS	Pulse Per Second

PPSI	Pulse Per Second Input
PPSO	Pulse Per Second Output
PROM	Programmable Read Only Memory
PVT	Position Velocity Time
R2C	Real to Complex Converter
RF	Radio Frequency
SCK	Serial Clock
SCKM	SCK Master
SCKS	SCK Slave
SDRAM	Synchronous dynamic random access memory
SEU	Single Event Upset
SGPO	Serial General Purpose Output
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SV	Space Vehicle
TBG	Time Based Generator
TMBOC	Time Multiplexed BOC
UART	Universal asynchronous receiver/transmitter
VFCG	Very Flexible Code Generator

1.5 References

1.5.1 Applicable Documents

Id	Document	Reference
[AD-01]	The SPARC Architecture Manual http://sparc.org/wp-content/uploads/2014/01/v8.pdf.gz	
[AD-02]	Cobham Gaisler Floating Point Unit (GRFPU) Manual http://gaisler.com/doc/grfpu_product_sheet.pdf	
[AD-03]	AGGA-3 Technical Note, Implementation Loss Analysis, Issue 1.0, 04. Nov. 2004	AGGA3-ASTD-TN-0001
[AD-04]	ATMEL AT7991 Datasheet	latest
[AD-05]	AGGA-4 Radiation Test Report	APL/DOC/2018/0625 (latest)

1.5.2 Reference Documents

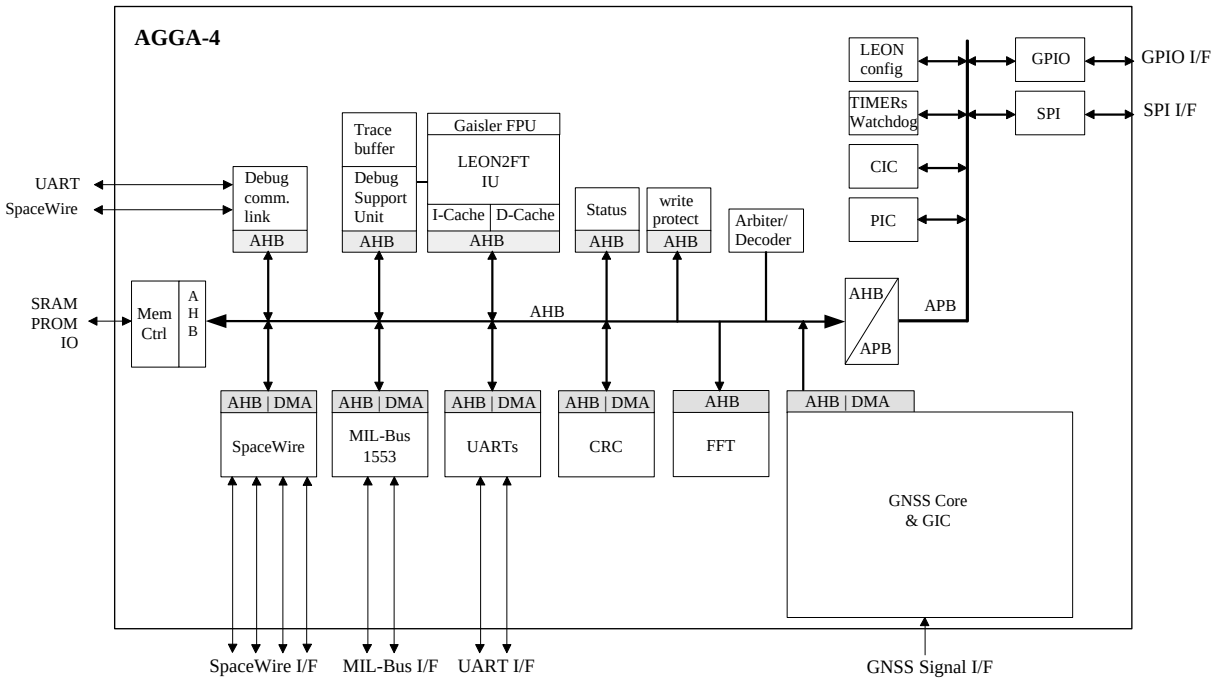
Id	Document	Reference
[RD-01]	"Galileo Open Service Signal-In-Space Interface Control Document, Draft 0 (GAL OS SIS ICD/D.0)," *Interface Control Document*, European Space Agency / Galileo Joint Undertaking, May 2006 - http://www.galileoju.com/page2.cfm	
[RD-02]	"Navstar GPS Space Segment / Navigation User Interfaces ICD-GPS-200C, Rev C," *Interface Control Document*, January 2003, available at http://www.navcen.uscg.gov/gps/modernization/	
[RD-03]	"Navstar GPS Space Segment / User Segment L5 Interfaces ICD-GPS-705, Rev 2," *Interface Control Document*, September 2005, available at http://www.navcen.uscg.gov/gps/modernization/	
[RD-04]	"Navstar GPS Space Segment / User Segment L1C Interfaces Draft IS-GPS-800," *Interface Specification*, April 2006, available at http://www.navcen.uscg.gov/gps/modernization/	
[RD-05]	GNSS Receiver Design for Attitude Determination, Proceedings of Third ESA International Conference on Spacecraft Guidance, Navigation and Control Systems, ref. SP-381, pp. 275-285, February 1997, European Space Agency	
[RD-06]	AMBA Specification 2.0, 13.05.1999 http://infocenter.arm.com/help/topic/com.arm.doc.ihi0011a/index.html	
[RD-07]	Mil Bus Standard with Notices	MIL-STD-1553B
[RD-08]	Mil Bus Standard Handbook	MIL-HDBK-1553A
[RD-09]	AGGA-4 Final Report, ESA Contract 16831/03/NL/FF, 12.12.2014	AGGA4-ASTD-RP-0003
[RD-10]	Anomaly in LEON2FT data cache snooper	ESA-TEC-TN-012539

2 AGGA-4 Overview

The AGGA-4 (Advanced GPS/GALILEO ASIC) is a radiation tolerant GNSS baseband ASIC capable of processing the modernized GPS and Galileo Signals. Due to its flexibility it is also able to process not only GPS and Galileo but also other GNSS systems like Glonass, Compass, etc.

While the predecessor AGGA-2 "only" incorporated the GNSS core inside the ASIC, AGGA-4 also incorporates the processor, the communication interfaces and other support functions (e.g. FFT, CRC) inside the ASIC [RD-09]. This simplifies the GNSS receiver board design, while at the same time increasing the capabilities.

Figure 2-1 gives an overview to the functional blocks inside the AGGA-4:



Legend:
GIC: GNSS Interrupt Controller
CIC: Communication Interrupt Controller
PIC: Primary Interrupt Controller

Figure 2-1: AGGA-4 System Overview

3 GNSS Core

The AGGA-4 GNSS Core is depicted in Figure 3-1. It consists of 4 Input Modules (IM), one Power Level Detector (PLD) Module, two Beam Forming (DBF) Modules, a Time Base Generator (TBG), an Antenna Switch Controller (ASC) and 36 channels. Please note that the connections in Figure 3-1 just depict functional connections. They are not representative concerning their width.

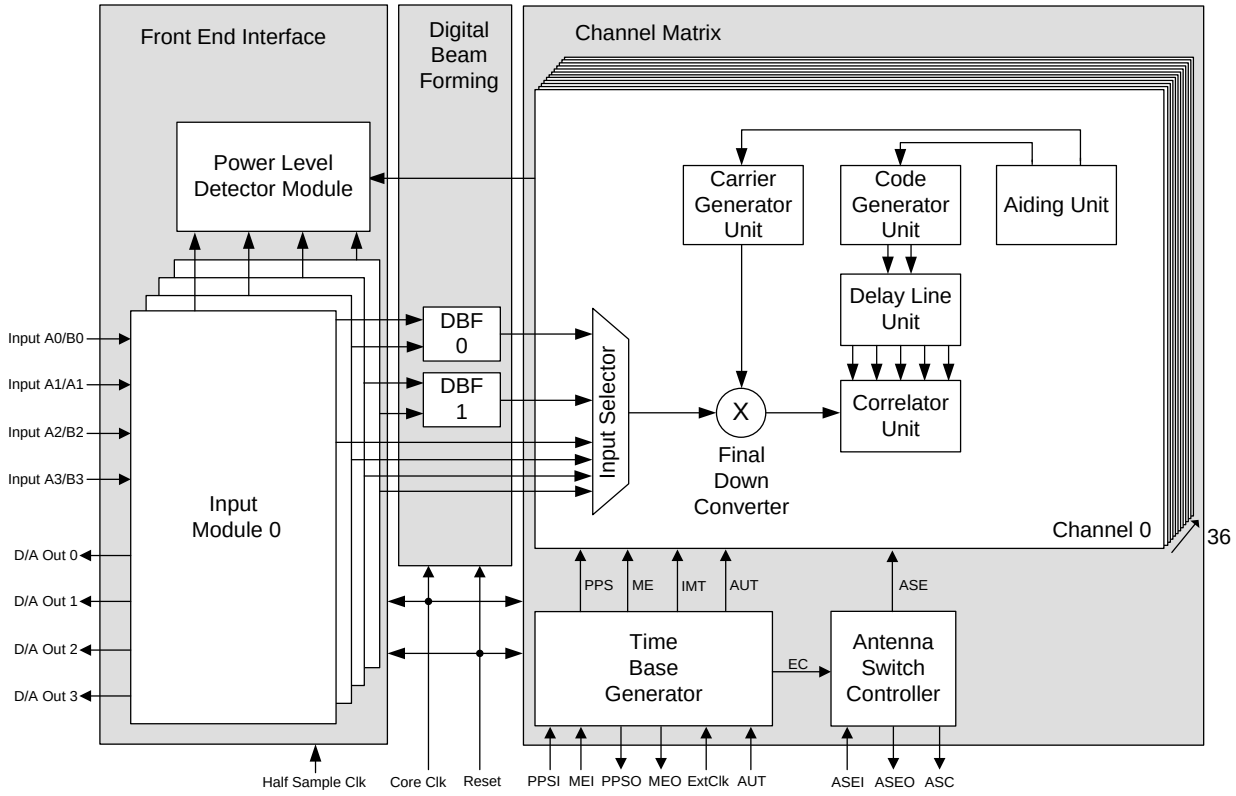


Figure 3-1: AGGA-4 GNSS Core

Up to 4 RF Front End's can be connected to the AGGA-4. The data enters in AGGA-4 through the Input Module(s) where it can be further down converted, if necessary. From the Input Module(s) the data goes either through the Digital Beam Forming (DBF) Network or directly into the channels. Each channel can select its input signal out of several possibilities as further detailed in chapter 3.4.1. The GNSS Core of the AGGA-4 uses two clock domains, the GNSS Core Clock (*CoreClk*) and the *HalfSampleClock* which has always to be 2.5 times the *CoreClk* frequency. Minimum periods for these clocks and the way how the internal clock domains are derived from the external pins *EXT_CORE_CLK* and *HALF_SAMPLE_CLK* are shown in section 6.4. The GNSS Core has its own reset input. If it is used, all registers within the GNSS core are reset to their default values.

Note: The present chapter of the datasheet often refers to a CoreClk frequency of 40 MHz, because several parts of the GNSS core like the Delay Line Settings, Aiding Unit Resolution, Integration Lengths, Accumulator Widths, etc have been optimized for this frequency.

Note also that the SystemClock always has to be equal or greater than the GNSS CoreClk frequency. Otherwise a stable operation of the AGGA-4 system is not guaranteed.

3.1 Input Module

The AGGA-4 supports the type of interfaces supported by the AGGA-2. Additionally, it offers Digital Down Conversion (DDC) modules for wide bandwidth navigation signals and a D/A converters for AGC control of the RF Front Ends. In the DDC mode it is also possible to slave Input Modules together in order to separate several frequency bands. Figure 3-2 shows the block diagram of the Input Modules. The AGGA-4 has four Input Modules referred to as 0, 1, 2 and 3 to support up to 4 antennas.

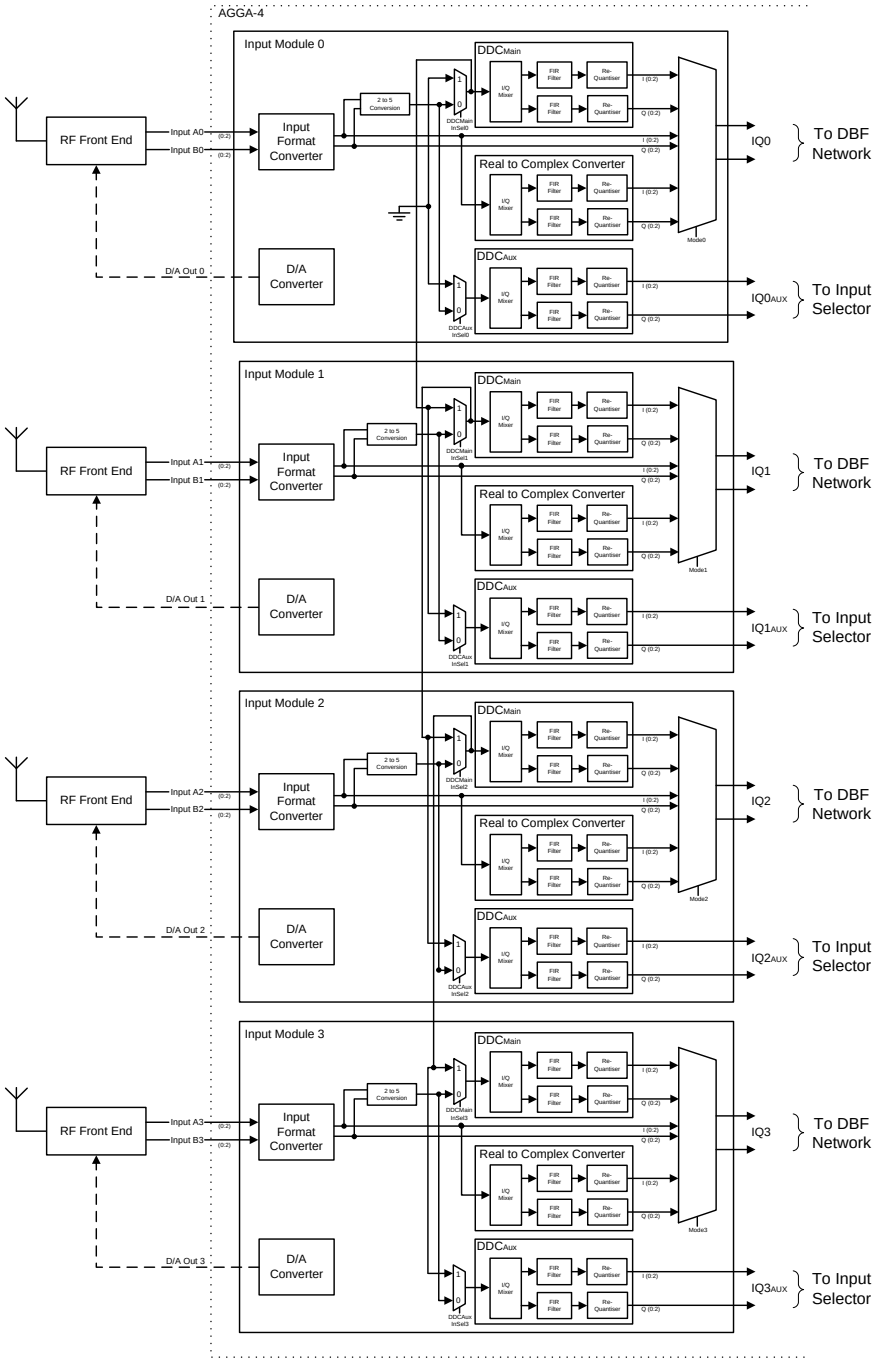


Figure 3-2: AGGA-4 Input Modules

Every input module contains an Input Format Converter (IFC), two Digital Down Converter (DDCs), a Real to Complex Converter (R2C) and a D/A Converter in order to control an AGC in the RF Front End. The Front End Interface is capable of processing wide-band signals and a variety of RF Front End frequency plans.

The AGGA-4 has four (0 to 3) separate 2 (A,B) x3-bit-wide inputs (A0 – A3 and B0 – B3), supporting ADC data from:

- Four chains of complex data (I and Q) on inputs A_n and B_n (three bits each for I and Q) in the IFC mode for input signal sampled on the rising edge of ***CoreClk***.
- Four chains of real data on input A_n (three bits) to be further processed in the R2C mode for input signal sampled on the rising and falling edge of ***CoreClk***.
- Four chains of real demultiplexed data on input A_n and B_n (each three bits) to be further processed in the DDC mode, supporting a high-speed sampling scheme: Input A_n is sampled on the rising edge and Input B_n on the falling edge of the ***HalfSampleClk***.

This information is summarized in the following table:

Mode	Data Input on Pins	Input Data Rate	*CoreClk*	*HalfSampleClk*	Sampling On	Required Format
IFC	A_n (0) = I-Sample LSB A_n (1) = I-Sample A_n (2) = I-Sample MSB B_n (0) = Q-Sample LSB B_n (1) = Q-Sample B_n (2) = Q-Sample MSB	1 x *CoreClk*	*CoreClk*	N/A	Rising edge	Sign/magnitude, Unsigned, Two's complement
R2C	A_n (0) = I-Sample LSB A_n (1) = I-Sample A_n (2) = I-Sample MSB	2 x *CoreClk*	*CoreClk*	N/A	Rising and falling edge	Sign/magnitude, Unsigned, Two's complement
DDC	A_n (0) = I-Sample-1 LSB A_n (1) = I-Sample-1 A_n (2) = I-Sample-1 MSB B_n (0) = I-Sample-2 LSB B_n (1) = I-Sample-2 B_n (2) = I-Sample-2 MSB	2.5 x *CoreClk*	*CoreClk*	2.5 x *CoreClk*	Input A_n on rising edge Input B_n on falling edge	Sign/magnitude, Unsigned, Two's complement

Table 3-1: AGGA-4 Signal Input Schemes

The following figures give an example for each of the three modes. Note that for easier readability only two of the four input modules are depicted.

Figure 3-3 gives an example for the IFC mode. This is the easiest input mode, since the data coming from the RF Front End is only formatted if needed and passed through to the Beam Forming Module. In this mode the Input Module requires complex (I/Q) Input data. The ADC for the Input Data as well as the GNSS core is driven by the *CoreClk* which can be up to 40 MHz (recommended). Figure 3-3 depicts an example with two RF Front End's (dual frequency receiver). The green path in the figure depicts the *CoreClk* domain. Note that the *HalfSampleClk* is not used in the IFC mode.

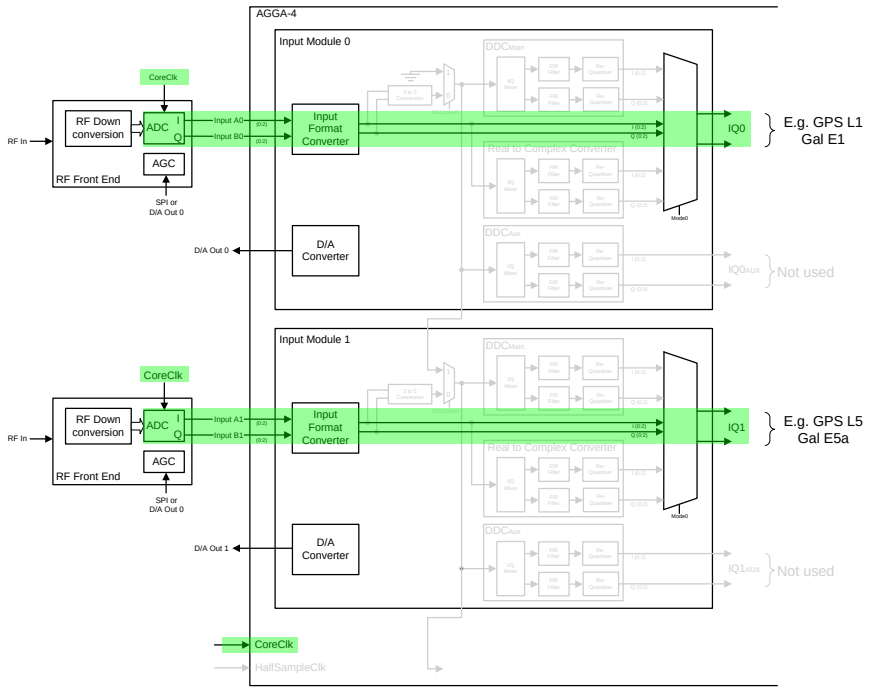


Figure 3-3: Input Format Converter (IFC) Mode Example

Figure 3-4 gives an example for a dual frequency case using the R2C Mode. In this case the ADC is sampling two IF chains (ch0 and ch1) and provides real samples with two times the *CoreClk* frequency on each path (ch0 and ch1). The data is fed into the AGGA only via the A_n inputs. Inside the AGGA the data is sampled with the rising and falling edge of the *CoreClk*. The data can be re-formatted in the IFC if needed and is then mixed with one quarter of the sampling frequency to near baseband and converted from a real to a complex I/Q signal. The ratio between the Sample frequency and the *CoreClk* frequency has to "2". In the drawn example the red color denotes the Sample Clock domain and the green color the *CoreClk* domain. Note that the *HalfSampleClk* is not used in the R2C mode.

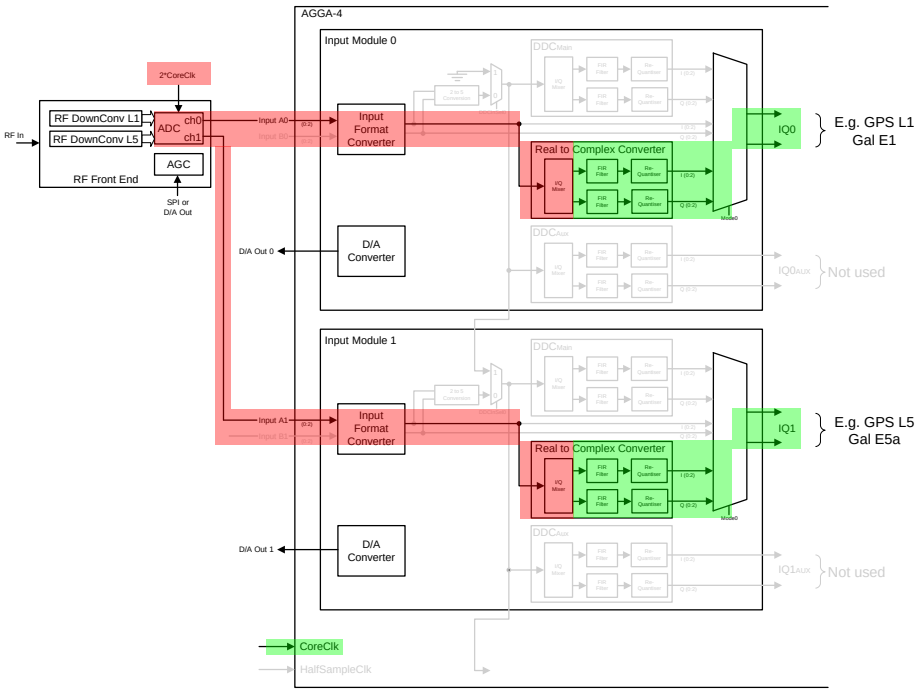


Figure 3-4: Real to Complex (R2C) Mode Example

Figure 3-5 gives an example for the DDC mode. In this mode a real signal is sampled with a sampling frequency up to 200 MHz. The data is then fed in an interleaved mode on ports A and B to the AGGA with the HalfSampling Clock and is then parallelized into five streams. Therefore after this parallelizing process (2to5 converter) the functional blocks are running with *CoreClk*. In Figure 3-5 three DDC's have been slaved in order to separate the different GNSS bands contained within the ADC data. The ratio between the Half Sample Clock and the *CoreClk* has to be "2.5". In the drawn example the red color denotes the Half Sample Clock domain and the green color the *CoreClk* domain.

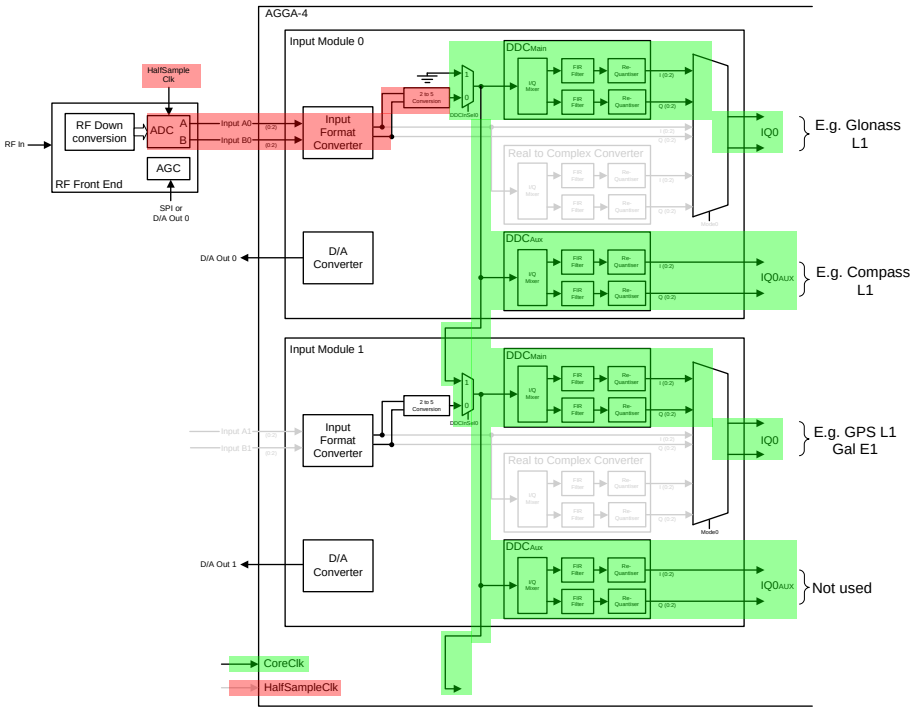


Figure 3-5: Digital Down Converter (DDC) Mode Example

3.1.1 Input Format Converter (IFC)

The input format converter samples the data from the Input A_n (3bit) and B_n (3bit) and translates it to the internally used 3 bit representation (unsigned). The unsigned representation is used within the whole AGGA-4. It supports different input formats: sign/magnitude, unsigned and two's complement as defined in Table 3-2.

Value	Sign/magnitude	Unsigned	Two's complement
-7	011	000	100
-5	010	001	101
-3	001	010	110
-1	000	011	111
+1	100	100	000
+3	101	101	001
+5	110	110	010
+7	111	111	011

Table 3-2: Supported input formats

3.1.2 Digital Down Converter (DDC) Module

Each Input Module contains two identical DDC modules (main and auxiliary). Both are always processing the same input. Each Input Module can support demodulation of different frequency bands within one data stream.

The main function of the Digital Down Converter module is to down convert a GNSS signal band to near baseband frequency. Moreover, each signal band is separated into its in-phase (I) and quadrature (Q) components. If two DDCs are used (e.g. main(0) & auxiliary(0) or main(0) and main(1)) then they can separate different frequency bands.

Only the DDCmain is fed to the Digital Beam Forming Module. The DDCaux is directly fed to the Input Selector of each channel. Note that there is no clock cycle delay between the main and the aux stream, when arriving in the channel.

DDCMainInSel and *DDCAuxInSel* in the *InputModuleCtrl* register determine whether a DDC Module shall process its own input data or the input data of the predecessor Input Module (slaving mode). Note that the first Input Module (IM0) can only process its own data.

In the DDC mode the 3-bit samples are entering the AGGA-4 via the inputs A_n and B_n (e.g. A0/B0 for IM0) and are read with *HalfSampleClk*..

Note that the ratio between *HalfSampleClk* and *CoreClk* shall exactly be 2.5 (with constant phase relationship).

3.1.2.1 Two to Five Converter

The function of this module is the reduction of clock frequency by parallel data processing. In the DDC mode a multiplexed data stream on the inputs A_n and B_n is used, where n is an integer from 0..3. After the two to five conversion the sample clock frequency is reduced by a factor of 5 through processing of the divided data stream in 5 streams simultaneously. The functional block diagram is given in Figure 3-6.

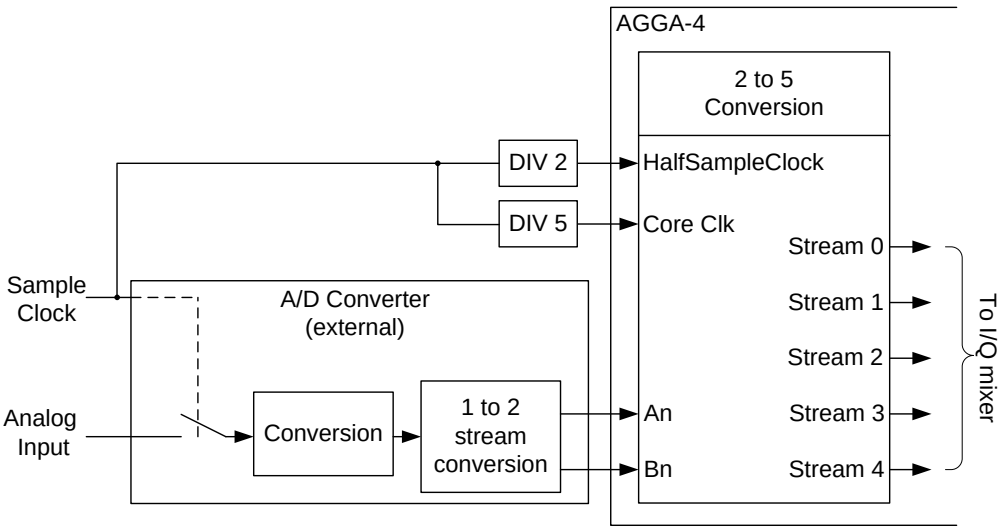


Figure 3-6: Functional block diagram of the Two to Five conversion

As it is shown in Figure 3-6, the external A/D Converter divides the sampled data into two streams. The two to five channel conversion block uses the *HalfSampleClk* to input the data into a 16 stage buffer register and *CoreClk* to read it out in 5 parallel streams. In order to avoid a conflict between in and out pointer the data is written into the shift register with *HalfSampleClk* starting from position "0" after reset of the AGGA-4 and is read with *CoreClk* starting from position "8" after reset of AGGA-4. From there on the two pointers are maintained by the two clock domains (*HalfSampleClk* and *CoreClk*) and it is guaranteed that the pointers will never interfere to each other as long as the *HalfSampleClk* and *CoreClk* applied to the AGGA-4 show no phase drifting against each other. This can either be guaranteed by chosing the AGGA-4 internal "2.5 *HalfSampleClk* to *CoreClk*" divider (see **Figure 6-5**) or by making sure that the *HalfSampleClk* and *CoreClk* are derived from the same frequency source.

As a consequence of this synchronization the data stream is delayed by 8 *HalfSampleClk* cycles by the "Two to Five Converter".

3.1.2.2 DDC I/Q Mixer Stage

The I/Q mixer separates the data stream into the inphase and quadrature components and converts the sampled navigation signal from IF to near baseband. Both operations are carried out by the multiplication of the incoming IF signal data stream with a programmable mixing frequency. Figure 3-7 shows the signals associated with the I/Q mixer.

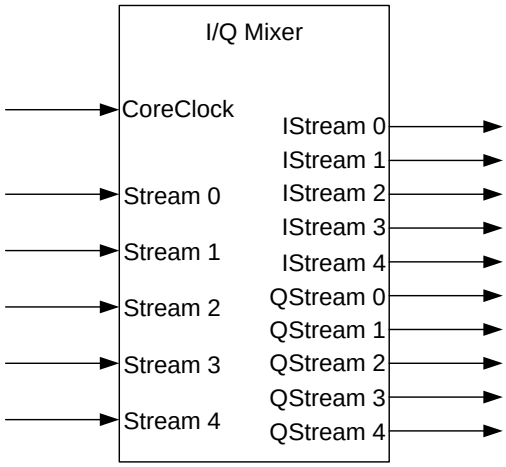


Figure 3-7: I/Q mixer module

The programmable complex mixing frequency f_{LO} is generated by a Numerically Controlled Oscillator (NCO). The input signal frequency (f_{in}) is transposed according to f_{in}-f_{Lo}. The NCO frequency f_{LO} is given by the following equation:

$$f_{LO} = \pm \frac{CoreClk \cdot 5 \cdot \Delta\Phi}{2^7}$$

where $\Delta\Phi$ is the signed phase register increment each *CoreClk* period. The accumulator is 7 bit wide. The frequency range is therefore $\pm$ 5*CoreClk/2* with a resolution of 5*CoreClk*/2^7 New phase increment values written into *DDCMainPhaseInc* or *DDCAuxPhaseInc* become effective with the next Measurement Epoch (ME). Besides this, also the phase of the NCO is reset to zero with the next ME. This allows phase alignment of multiple DDC NCO's. The phase increment is a programmed value between -180 deg. and + 180 deg. As the data is paralleled into 5 bit streams, the NCO produces a phase to each of the branches simultaneously. The NCO is incremented by $5 \cdot \Delta\Phi$ and the NCO phase is distributed to the 5 branches where each branch performs an additional phase increment in order to achieve the appropriated LO phase (see Figure 3-8).

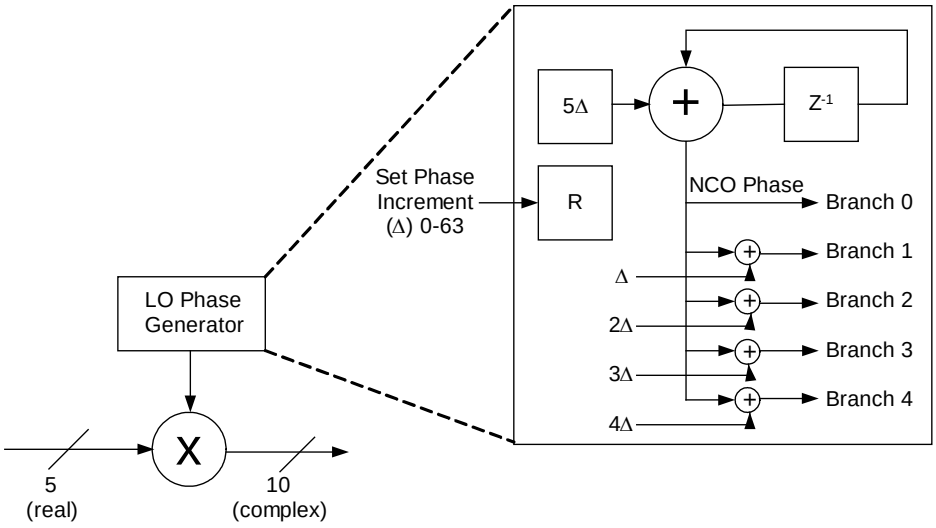


Figure 3-8: DDC mixer stage including the NCO. The delay of z^{-1} refers to the *CoreClk*

The mixer stage converts the real input data stream, centered at IF between - *SampleClk*/2 and *SampleClk*/2 frequency, to a complex signal near baseband. The incoming signal is therefore phase rotated into a complex signal. The real 3 bit input signal and the complex output signal use the internal representation, corresponding to the values [-7, -5, -3, -1, 1, 3, 5, 7]. The 8 possible representations for the real signal and the resulting possible combinations for the in-phase component and the quadrature component are shown in Figure 3-9.

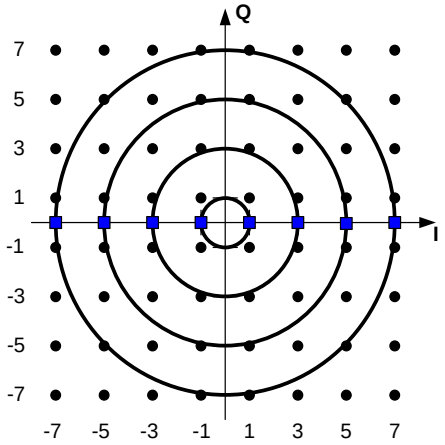


Figure 3-9: 3 bit signal constellation for the real signal (blue squares) and for the complex signal (black points). The black circles show the rotation of the input frequency. The appropriated rotation is rounded to the nearest complex state.

For the down conversion the real 3 bit input values and the 7 bit phase register values are considered. With these inputs the resulting complex signal point can be read out of a Look-Up Table. In this table for each real input value and each possible phase shift the corresponding (minimum Euclidian Distance) complex value (I and Q component, each 3 bit) is saved.

In Figure 3-10 the 8 possible real 3 bit input samples are rotated with every possible phase shift (128; each phase shift has a size of 360°/128=2,8°) resulting in the small black dots. The light red circles represent the possible complex samples after the phase rotation. Each rotated real input sample (black dot) is connected with the nearest (minimum Euclidian Distance) possible complex sample by a blue line. The figure shows to which complex point the rotated real input samples are mapped.

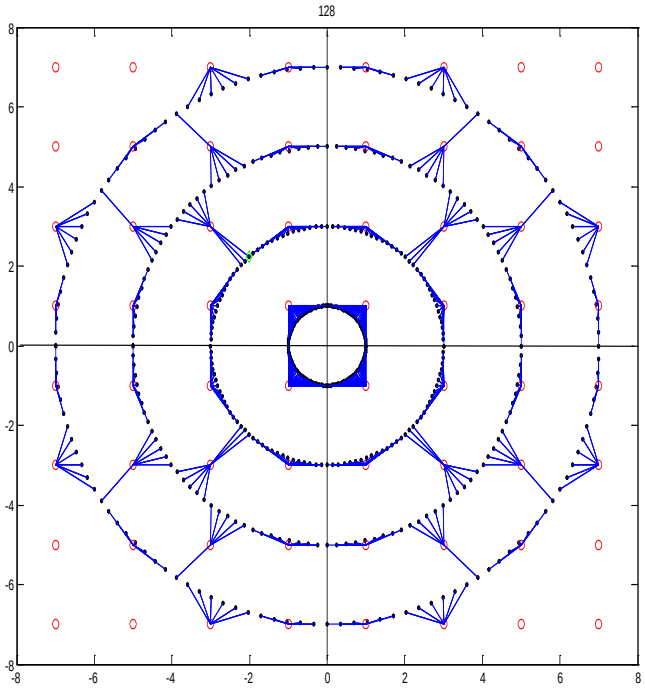


Figure 3-10: Complex 3 bit IQ signal constellation

3.1.2.3 Decimating FIR Filter

In the DDC mode, the complex signal from the I/Q mixer is subsequently filtered to suppress undesired frequency components and frequency images. The FIR filter extracts the desired frequency components of the sampled navigation signal and decimates the sample rate by a factor of 5. As each FIR filter operates at 5 data streams, only one output sample of 5 input samples is generated. The output is synchronous with *CoreClk* and consists of one stream of data. Figure 3-11 shows the Q filter. The I filter looks the same.

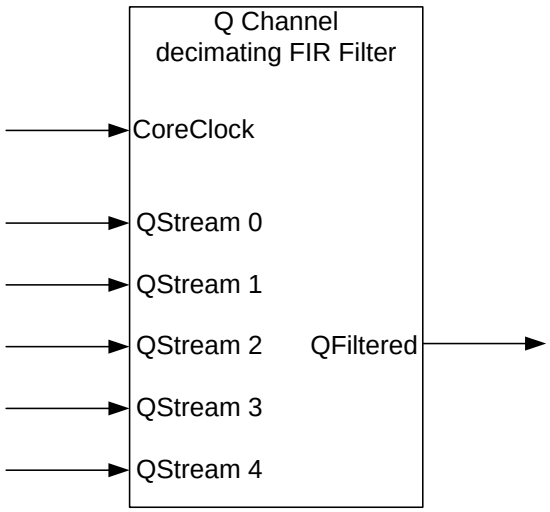


Figure 3-11: Decimating FIR filter block

The input signals *IStream0 – IStream4* and *QStream0 – QStream4* are filtered in the FIR filter block. The output signals *IFiltered* and *QFiltered*, respectively, contain the filtered and decimated data. Both, input and output signals are synchronous with *CoreClk*.

Fixed FIR filter coefficients and a fixed decimation factor of 5 are used. The main characteristics of the FIR filter are to maintain a flat amplitude response within the pass band to minimize signal distortion while suppressing out-of band signals and noise (see Figure 3-12). The hard coded (not programmable) FIR filter coefficients are [-1, -1, 0, 1, 3, 5, 6, 7, 6, 5, 3, 1, 0, -1, -1].

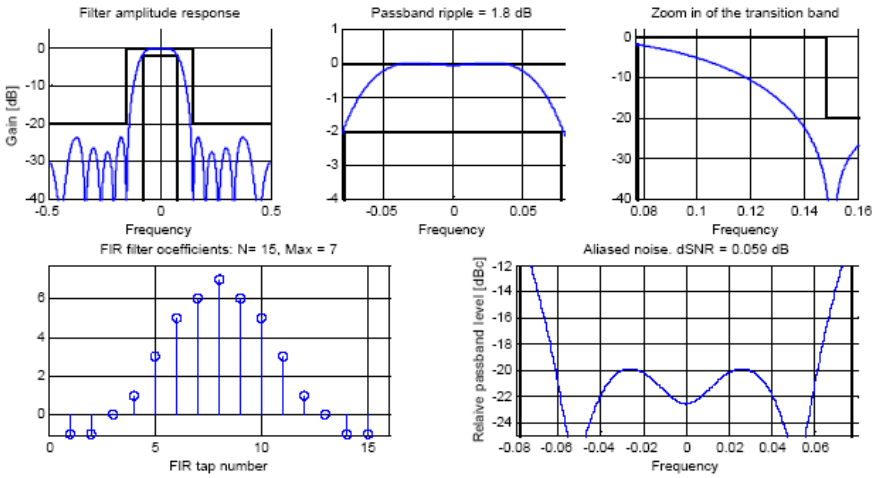


Figure 3-12: DDC/DDC FIR filter. ADC sample rate = 1, output sample rate = 0.2.

The architecture of the FIR filter in the DDC is shown in Figure 3-13, where the 15 tap transposed direct form FIR filter is given. The output data rate is decimated by a factor of 5.

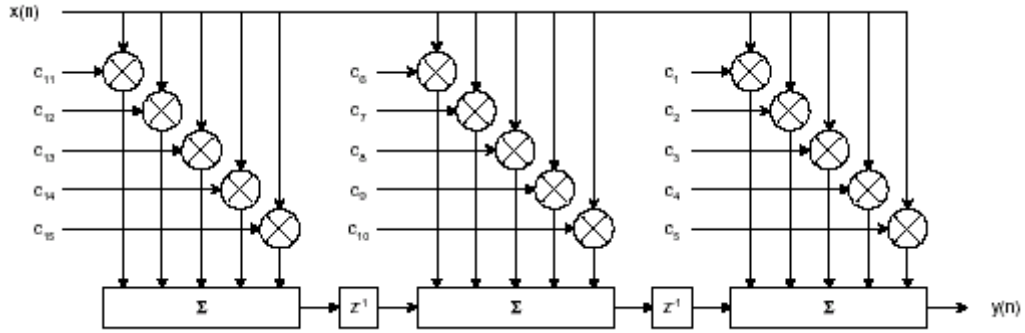


Figure 3-13: 15 tap transposed direct form decimating FIR filter

3.1.2.4 Re-Quantizer

The filtered data streams are fed to the re-quantiser and coding block. It performs rounding of the FIR filter samples to 3 bits for the following stages of GNSS data processing. The re-quantisation thresholds are programmable.

The re-quantiser block accepts the signal *IFiltered* and the *Qfiltered* and outputs the signal *IOut* and *QOut*. Although the following figure depicts only the Q re-quantiser, the I re-quantiser looks the same.

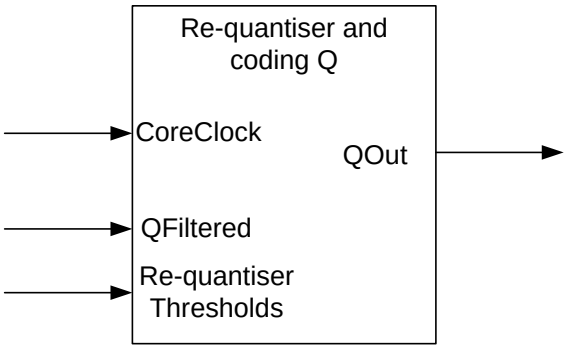


Figure 3-14: Re-quantiser and coding block (figure with quadrature part only)

The three 10-bit threshold registers *DDCMainFIRQuantThres* and *DDCAuxFIRQuantThres* can be programmed. The thresholds are going to be interpreted as follows:

Data Filtered				Data Out (Formatted)
max	-	Threshold2 + 1	=>	**+7**
Threshold2	-	Threshold1 + 1	=>	**+5**
Threshold1	-	Threshold0 + 1	=>	**+3**
Threshold0	-	0	=>	**+1**
-.1	-	-Threshold0	=>	**-1**
-Threshold0 - 1		-Threshold1	=>	**-3**
-Threshold1 - 1	-	-Threshold2	=>	**-5**
-Threshold2 - 1	-	min	=>	**-7**

Table 3-3: Quantization Thresholds

Note that the max/min FIR output value is +/- 287 and only 9-bits of the 10-bit threshold register are needed.

3.1.3 Real to Complex Converter (R2C)

The Real to Complex Converter (R2C) is the same as used in the AGGA-2. It can do a real to complex conversion. In comparison to the DDC module it has the limitation that the incoming IF has to be quarter of the sampling frequency or half of the *CoreClk* frequency, while the DDC can be used with a wide variety of IF's.

The real input signal is down converted in frequency and it is separated into the corresponding in-phase and quadrature component using a complex *CoreClk*/2 heterodyne as it is shown in Figure 3-15.

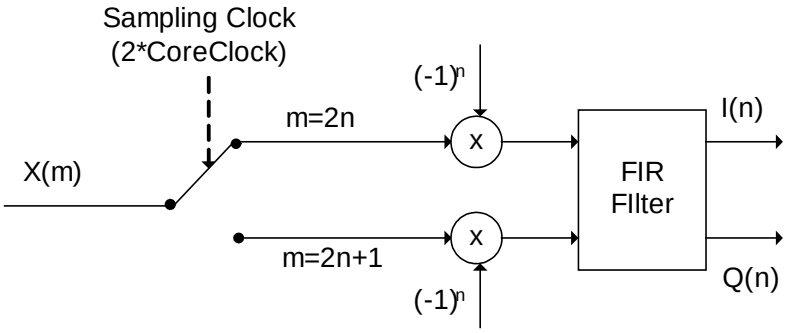


Figure 3-15: Complex *CoreClk*/2 heterodyne

It consists of a mixer stage and a low pass filter to suppress undesired frequency components and frequency images and to time align the I and Q path. The FIR filter extracts the desired frequency components of the sampled navigation signal and decimates the sample rate with a factor of 2.

In the R2C mode the 3-bit samples are entering the AGGA-4 only via the inputs Ax (e.g. A0 for IM0) and are read with the rising and falling edge of the *CoreClk*. The *HalfSampleClk* is not involved in the R2C mode.

3.1.3.1 R2C I/Q Mixer Stage

The down conversion of the intermediate frequency signal and the division into the real and imaginary part requires a complex multiplication of the sampled signal with a local generated complex carrier. The sampling frequency is to be selected so that the intermediate frequency is equal to *CoreClk*/2 which ensures the samples of the carrier wave taken at the sampling rate are samples of sine and cosine functions with 4 samples per cycle i.e. the samples at the zero crossings with value 0, maxima with value 1 and minima with value -1. The heterodyning function becomes one of multiplexing and selective inversion, as it is depicted in Figure 3-15. The complex heterodyne process corresponds to a down conversion of the input signal by a fixed frequency equal to one quarter of the sampling frequency. The input signal frequency (f_{in}) is transposed according to f_{in}-f_{Lo}.

3.1.3.2 FIR Filter

The complex baseband signal is subsequently low-pass filtered to remove undesired frequency images. As f_{IF} = *CoreClk*/2 and alternate samples are zero, the filtering operation is implemented at a sampling rate of *CoreClk*/2. The low pass filter characteristics are:

- Passband ripple peak-to-peak: < 0.6 dB
- Stoppband suppression: 30 dB
- The passband ends at ± 0.42 times *CoreClk* frequency
- The stopband starts at ± 0.58 times *CoreClk* frequency

The incoming real signal frequency is centered around half *CoreClk* and should have a bandwidth not exceeding 0.84 times *CoreClk* frequency. A real input signal between 0.08**CoreClk* and 0.92**CoreClk* results in an internal complex signal between -0.42**CoreClk* and 0.42* *CoreClk*.

The FIR filter is symmetrical with the following hard-coded (not programmable) coefficients related to the input sampling rate:[2, 0,-3, 0, 5, 0, -9, 0, 30, 47, 30, 0, -9, 0, 5, 0, -3, 0, 2]. The overall filter response related to the down converted signal is given in Figure 3-16.

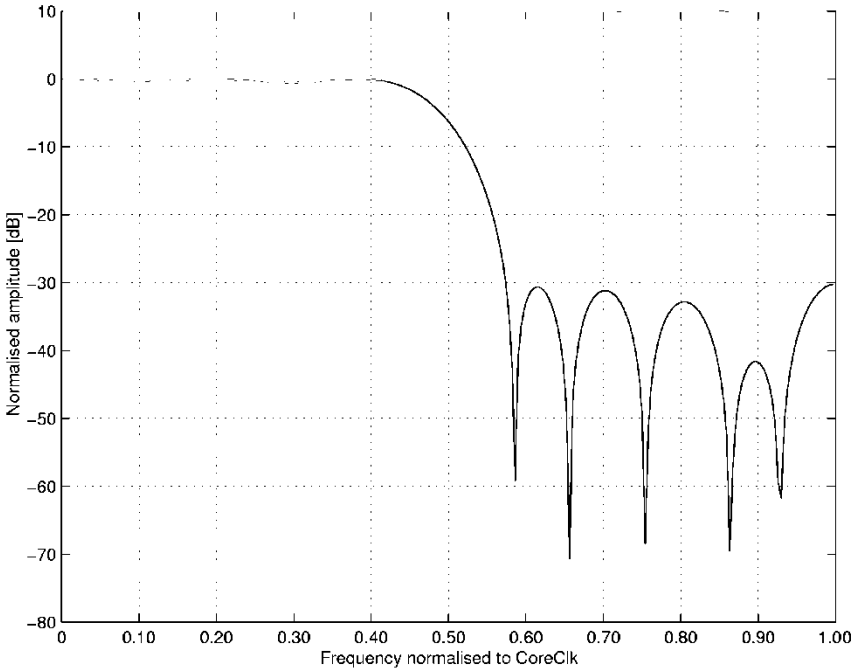


Figure 3-16: Overall filter response for the low pass filter

The implementation of this filter is simplified by the fact that only the operations with non-zero coefficients are performed.

The Q component is received with half *CoreClk* cycle delay respect the I component, being multiplied by the odd coefficients. In the same way, the I component is multiplied by even coefficients.

Defining the coefficient list as c1 - c19 and In and Qn the data of IMixed and Qmixed respectively, the resulting operations are:

*IFiltered = c1*I1 + c2*I2 + c3*I3 + c4*I4 + … + c10*I10 + c11*I11 + …*

where *C2n* = 0, except *c10*;

where *I2n-1* = 0.

*IFiltered = c10*I10*

*QFiltered = c1*Q1 + c2*Q2 + c3*Q3 + c4*Q4 + … + c10*Q10 + c11*Q11 + …*

where *C2n* = 0, except *c10;*

where *Q2n* = 0.

*QFiltered = c1*Q1 + c3*Q3 + … + c17*Q17 + c19*Q19*

where *cn = c20-n,*

QFiltered = c1 (Q1 + Q19) + c3 (Q3 + Q17) + c5 (Q5 + Q15) + c7 (Q7 + Q13) + c9 (Q9 + Q11)

In order to realise this calculation it is required to have shift register to store the previous data (only even or odd data).

The filtered I signal is not processed, because only one coefficient should be multiplied by the signal and in a next stage, the result will be requantised dividing by C_{10}. Therefore, this multiplication and the corresponding requantisation can be optimized.

In **Figure 3-17** and **Figure 3-18** it is detailed how these operations are implemented.

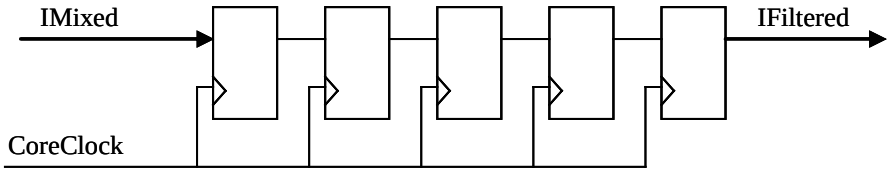


Figure 3-17. I component filter calculation

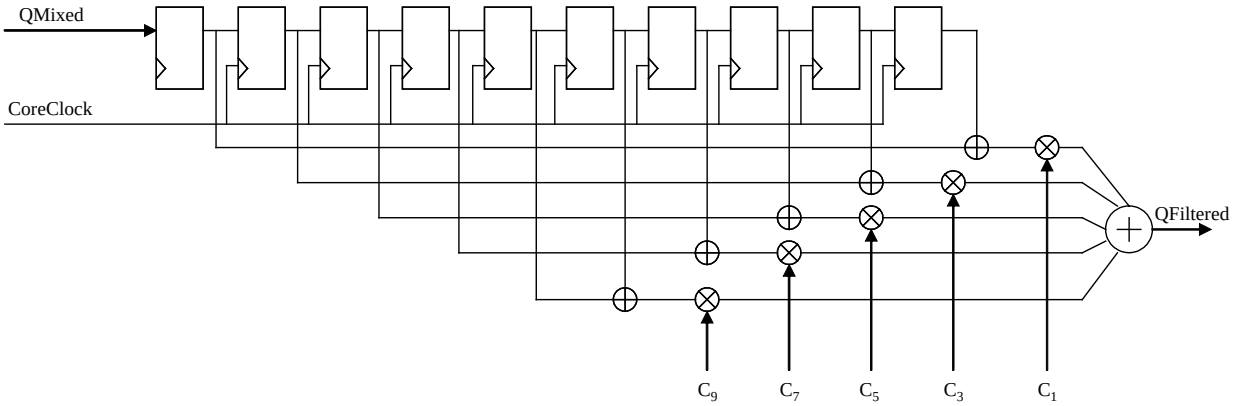


Figure 3-18. Q component filter calculation

3.1.3.3 R2C Re-Quantizer

The filtered data is fed to the re-quantiser and coding block. It performs rounding of the FIR filter samples to 3 bits for the following stages of GNSS data processing. Although the following figure depicts only the Q re-quantiser, the I re-quantiser looks the same.

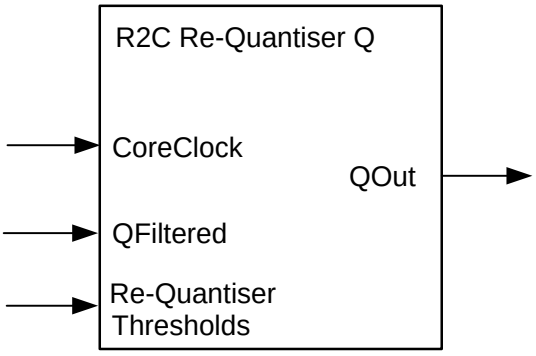


Figure 3-19: R2C Re-quantiser (figure with quadrature part only)

The output is quantized into the internal 3 bit format by comparing the filter output values to constant thresholds. The IFiltered and QFiltered component from the previous filter stage can have values ranged in between -343 to +343. For a uniform distribution of this range, the thresholds are set to constant values according to the following table:

Q (Filtered)				Q (Formatted)
max	-	283	=>	**+7**
282	-	189	=>	**+5**
188	-	95	=>	**+3**
94	-	0	=>	**+1**
-1	-	-94	=>	**-1**
-95		-188	=>	**-3**
-189	-	-282	=>	**-5**
-283	-	Min	=>	**-7**

Table 3-4: Real-To-Complex (Q part only) thresholds

3.1.4 D/A Converter

In order to be able to adjust the input power level or offset level of the ADCs, sometimes a voltage is required to control attenuators or automatic gain control (AGC) amplifiers prior to the ADCs. The D/A converter block produces a pulse width modulated digital bit stream, corresponding to the amplitude of the input word. Analogue low pass filtering of this bit stream outside of the AGGA-4 generates the control voltage.

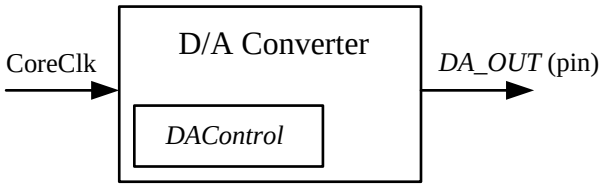


Figure 3-20: D/A Converter Block

The 12 bit wide register *DACtrl* contains the input value to the D/A converter. It can be programmed via software. Each input module contains one D/A Converter. The outgoing signal is routed to the pins *DA_OUT[0:3]*, where *DA_OUT[0]* corresponds to Input Module #0 and *DA_OUT[3]* to Input Module #3. The 12 bit input signal is used as an increment input for a 12 bit wide incrementer. The carry-bit of this incrementer gives the output bit stream of the D/A converter, making sure that the average number of ones in the output bit-stream is equal to the value which is programmed to *DACtrl*. Furthermore the ones are evenly distributed.

The incrementer and its output are synchronized to the GNSS *CoreClk*.

Example:

DACtrl = 0 => All '0'

DACtrl = 1 => 4095 '0' followed by one '1'

DACtrl = 1024 => 000100010001…

DACtrl = 2048 => 010101010101…

DACtrl = 4095 => one '0' followed by 4095 '1'

3.2 Power Level Detector

AGGA-4 has two Power Level Detector (PLD) modules. One PLD 5I and one PLD I/Q (see also Figure 3-21). The PLD 5I can count five data streams in parallel. Therefore it shall be used for the DDC modules. The PLD I/Q is used to count the sample occurrence either behind the Input Format Converter (IFC), before the Final Down Converter (FDC) or after the Final Down Converter. Note that the PLD I/Q only works for frequencies up to *CoreClk*.

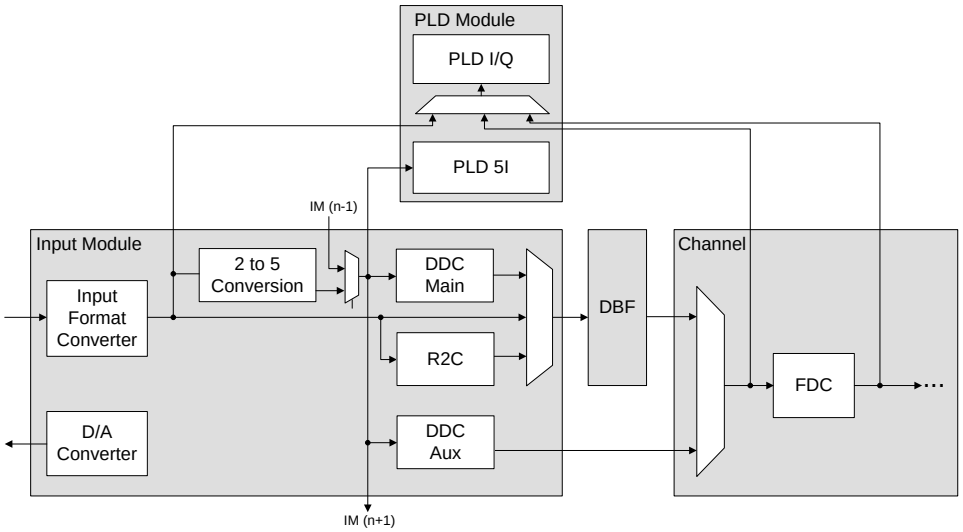


Figure 3-21: Power Level Detectors

3.2.1 Power Level Detector 5I

The PLD-5I detects the signal levels of the 5 input samples on the input data streams *SampleStream 0 – SampleStream 4* stemming from one of the DDC's before it's I/Q mixer.

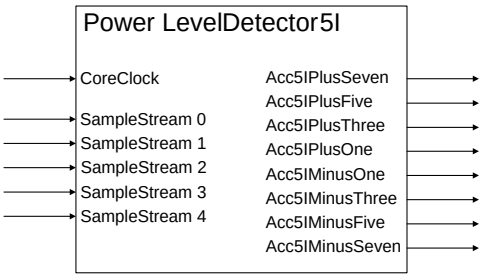


Figure 3-22: Power Level Detector 5I block

The numbers of occurrences of each signal level are accumulated in the eight 24bit accumulators in parallel. The accumulation time is either determined by each Measurement Epoch (ME) or by a programmable 24bit counter. The counter can be programmed via the *AccTime* field in the *PLD5ICtrl* register. The 24bit counter is incremented with each sample. Since the PLD is counting always 5 data streams, the counter is always incremented in steps of 5.

Therefore note that a programmed *AccTime* of 0..4 causes the PLD to accumulate 5 samples, a programmed *AccTime* of 5..9 causes the PLD to accumulate 10 samples, and so on.

The register *PLD5IInputSel* determines on which input module the PLD shall count.

An interrupt (GNSS PLD-5I) is generated as soon as the level results are available. It can be masked/unmasked in the GNSS Interrupt Controller (GIC).

When the PLD 5I latches its values, the corresponding time stamp is stored in the registers *PLD_5I_IMT_MSW* and *PLD_5I_IMT_LSW*.

Using the maximum possible value of 2^{24}-1 for *AccTime* can cause an overflow in one of the accumulation registers under worst case conditions (all samples are equal over the measuring period). Therefore it is recommended not to use this value.

Also note that the user has to take care that the accumulation registers have no overflow in case of ME latching.

3.2.2 Power Level Detector IQ

3.2.2.1 General Overview

The second type of power level detector is called 'Power Level Detector I/Q'. This detector consists of two identical power level detectors for the I and Q signal, which are processed in parallel.

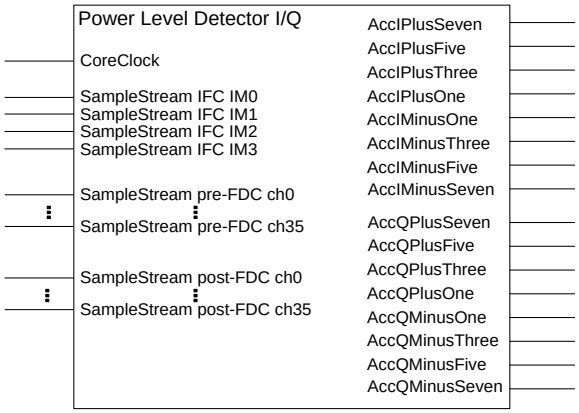


Figure 3-23: Power Level Detector I/Q block

In the PLD I/Q the upper branch processes the I signal and the lower branch processes the Q signal in parallel (see also Figure 3-24). By programming of the the *InputSel* and *SignalSel* bitfields of the *PLDIQInputSel* register a signal is selected from the incoming signals. The incoming 3 bit I/Q signal can be pre-accumulated in a signed pre-accumulator before the actual level detection. The pre-accumulator acts as low-pass filter with a 3dB cut-off frequency of:

$$f_{cutoff} = \frac{0.44}{2^{2\,ExpFactor}} \cdot f_{CoreClk}$$

The pre-accumulation can be useful to reduce the effect of out-of-band interference on the power level measurements.

It is possible to pre-accumulate over $2^{2ExpFactor}$ samples of the incoming signal, where *ExpFactor* can be selected between 0 and 5. If the *ExpFactor* is set to zero, the pre-accumulation and subsequent re-quantization is disabled.

The pre-accumulation window can be shifted by 0..1023 samples by the programming of the *Pause* field in the *PLDIQPreAccCtrl* register. The pause shift is applied once and immediately.

The integration result is scaled either by $2^{ExpFactor}$ or $2^{2ExpFactor}$. This can be selected by the programming of the *ReQuantFactor* bit in the *PLDIQPreAccCtrl* register. The output of the re-quantization stage is 3 bit wide.

The numbers of occurrences of each signal level are accumulated in the two times eight 24-bit accumulators in parallel for I and Q and for all signal levels separately. The accumulation time is either determined by each Measurement Epoch (ME) or by a programmable 24-bit counter. The counter can be programmed via the *AccTime* field in the *PLDIQCtrl* register. The 24bit counter is incremented with each sample after pre-accumulation. Therefore, if a pre accumulation is enabled, the effective accumulation time, measured in CoreClk cycles, will be AccTime * 2^(2*ExpFactor).

Note that the PLD I/Q can count the occurrences according to the following truth table:

		Input Module Operating Mode		
		IFC Mode	R2C Mode	DDC Mode
Spy Point	after IFC	YES	NO	NO
	before FDC	YES	YES	YES
	after FDC	YES	YES	YES

Table 3-5: PLD IQ Possible Working Modes

An interrupt (GNSS PLD I/Q) is generated as soon as the level results are available. It can be masked/unmasked in the GNSS Interrupt Controller. If masked, then it can also be used as status flag by polling the GNSS PLD IQ pending bit.

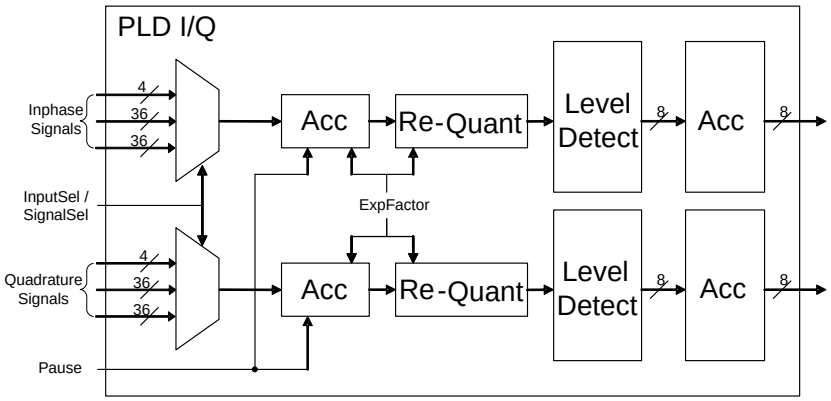


Figure 3-24: Power Level Detector I/Q

The time instance when the PLD I/Q latches its values is stored in the registers *PLD_IQ_IMT_MSW* and *PLD_IQ_IMT_LSW*.

3.2.2.2 Insights to Pre-Accumulator and Re-Quantization

Before being accumulated in the pre-accumulator the samples are transformed to 13-bit two's complement format numbers, in order to be able to handle negatives accumulations. This transformation is realised according to Table 3-6:

Sample	2's compl.
+7	+3
+5	+2
+3	+1
+1	0
-1	-1
-3	-2
-5	-3
-7	-4

Table 3-6: Sample Transformation

This transformation is asymmetric with respect to zero, inserting a negative offset of -0.5 with every sample, i.e. by an accumulation of 2^{2N} samples, an "unsymetry offset" of $-(2^{(2N-1)})$ will be accumulated. In order to compensate this, at the beginning of the accumulation, the register is set with an initial value equal to $2^{(2N-1)}$, setting to '1' the bit in the position 2N-1. Therefore after the pre-accumulation the offset is compensated.

Figure 3-25 depictes how the re-quantization back to 3bits is done after the pre-accumulation. In case of the re-quantization selection "ReQuant with ExpFactor $2^{2ExpFactor}$", the re-quantization is simply by shifting the result by

$2^{2ExpFactor}$ to the right. In case of the re-quantization selection "ReQuant with ExpFactor $2^{ExpFactor}$" one has to know that if one of the saturation bits is different from the sign bit (bit13), the Value[1:0] is also saturated by the hardware, no matter what the actual bits of [1:0] have been.

Example: ExpFactor =2, therefore 16 samples are pre-accumulated

Assumption: 16x -7 is accumulated as 16x(-4) = -64, with offset pre condition +8 = -56

-56 is binary 1111111001000. According to Figure 3-25 the grey marked bits (bit12, bit3:2) would be taken. However in this case at least one of the saturation bits (bit 5:4) is different from the sign bit (bit12). Therefore the result will not be "110" but "100"

In the end the two's complement values are back transformed to the 3bit unsigned representation according to Table 3-6.

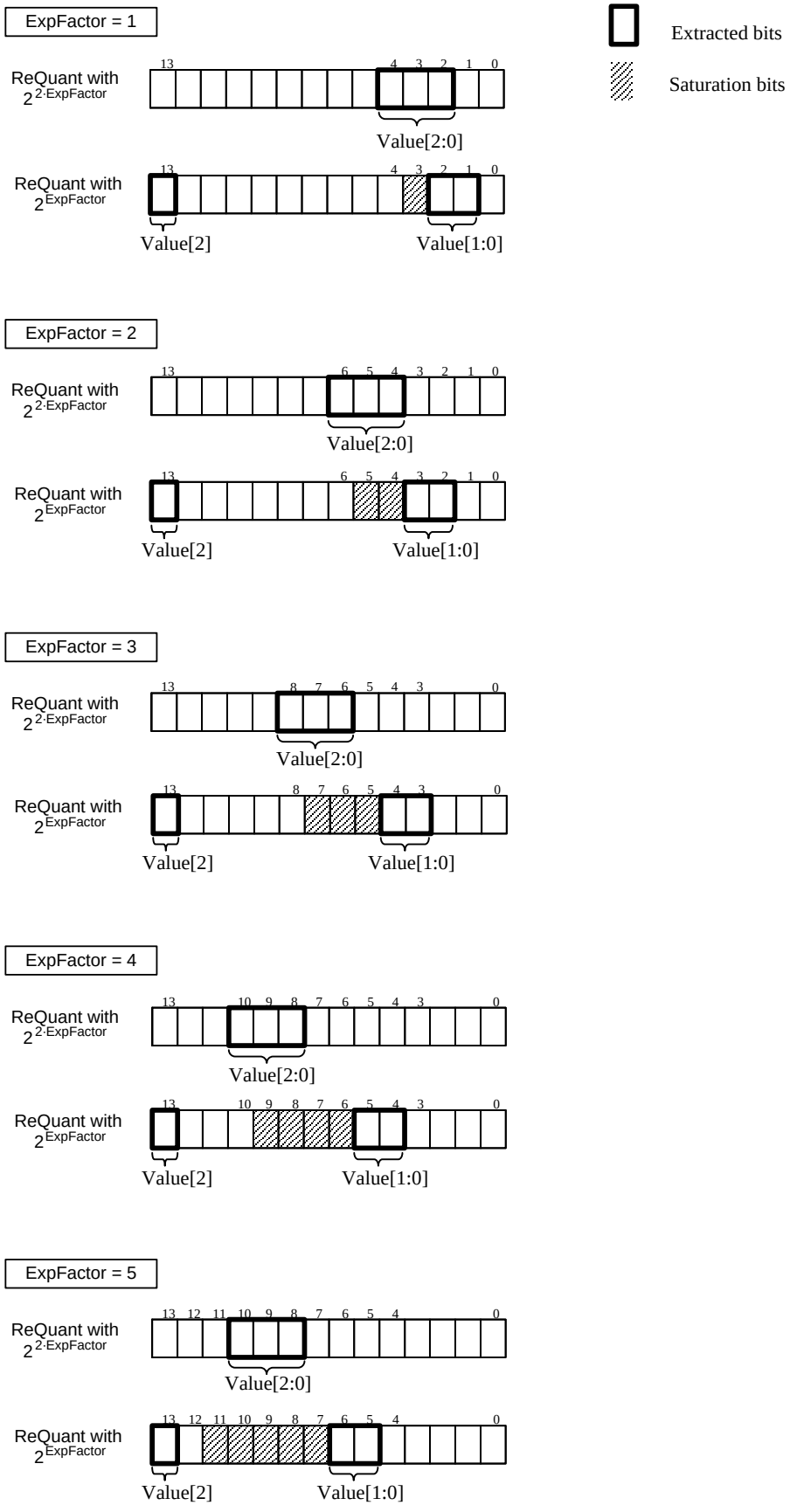


Figure 3-25: Re-Quantisation of PLD I/Q Pre-Accumulation Results

3.3 Digital Beam Forming (DBF)

The Digital Beam Forming (DBF) module combines the signals of two antenna elements in the digital domain in order to steer the antenna beam. In the AGGA-4, digital beam-forming is carried out before the correlation (pre-correlation beam forming). It performs phase shifting and combination of two antenna signals after the first coarse down-conversion in the Input Modules prior to the correlators. The main advantage is that only the combined signals are fed to the final down-conversion stage and the de-spreading correlators.

Two Digital Beam Forming (DBF) modules for the processing of two antenna inputs each are implemented. Generally, the first DBF module (DBF-0) processes the inputs from antenna A0 and antenna A1 and the second DBF module (DBF-1) processes the inputs from antenna A2 and A3 as it is shown in Figure 3-26.

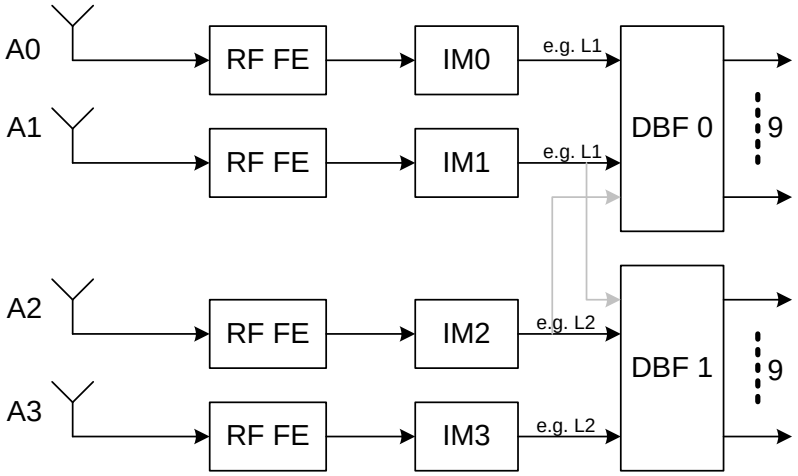


Figure 3-26: Beam forming for single frequency operation

However, for dual frequency operation the input from one antenna can be used for two Input Modules which process different frequencies. Since a second antenna is needed for the beam forming, the input of antenna A2 is send to the DBF-0 module and the input of antenna A0 is send to the DBF-1 module. By this architecture the antenna inputs of the same frequency are combined in the DBF modules. This is shown in Figure 3-27.

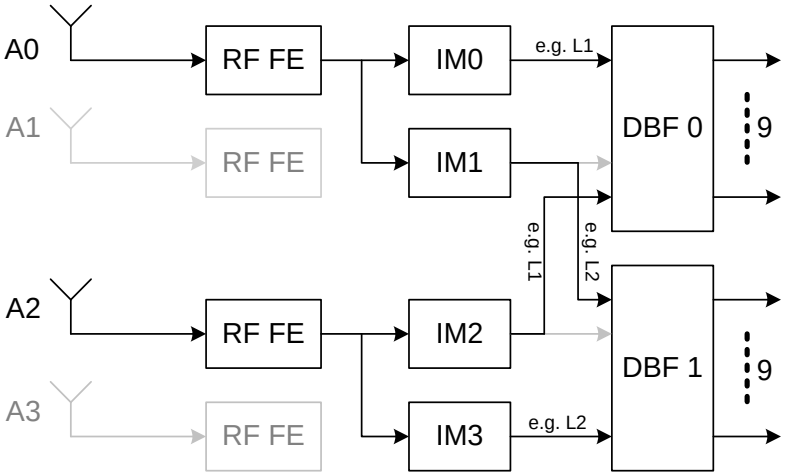


Figure 3-27: Beam forming for dual frequency operation

A fixed set of beam-forming coefficients supporting the rotation angles of 0, +/- 45°, +/- 90° and +/- 135° is used in the AGGA-4. All 7 possible beams are generated simultaneously and these beams and the direct antenna signals can be routed to any one of the channels in the GNSS core of the AGGA-4.

Beam Forming Coefficient	Rotation Angle
φ_0	0°
φ_1	45°
φ_2	90°
φ_3	135°

Table 3-7: Beam forming coefficients

Figure 3-28 shows the architecture of the DBF modules. If the DBF-0 is used as shown in Figure 3-26 then the inputs IM 0 and IM 1 are used. If the DBF-0 is used as shown in Figure 3-27 then the inputs IM 0 and IM2 are processed. By the programming of *DBF0Sel* and *DBF1Sel* in the *DBFInputSel* register, the user can select for each DBF module which Input Module input (e.g. IM1 or IM2 for DBF0) shall be selected. The inputs are phase rotated and combined or fed directly to the channel matrix.

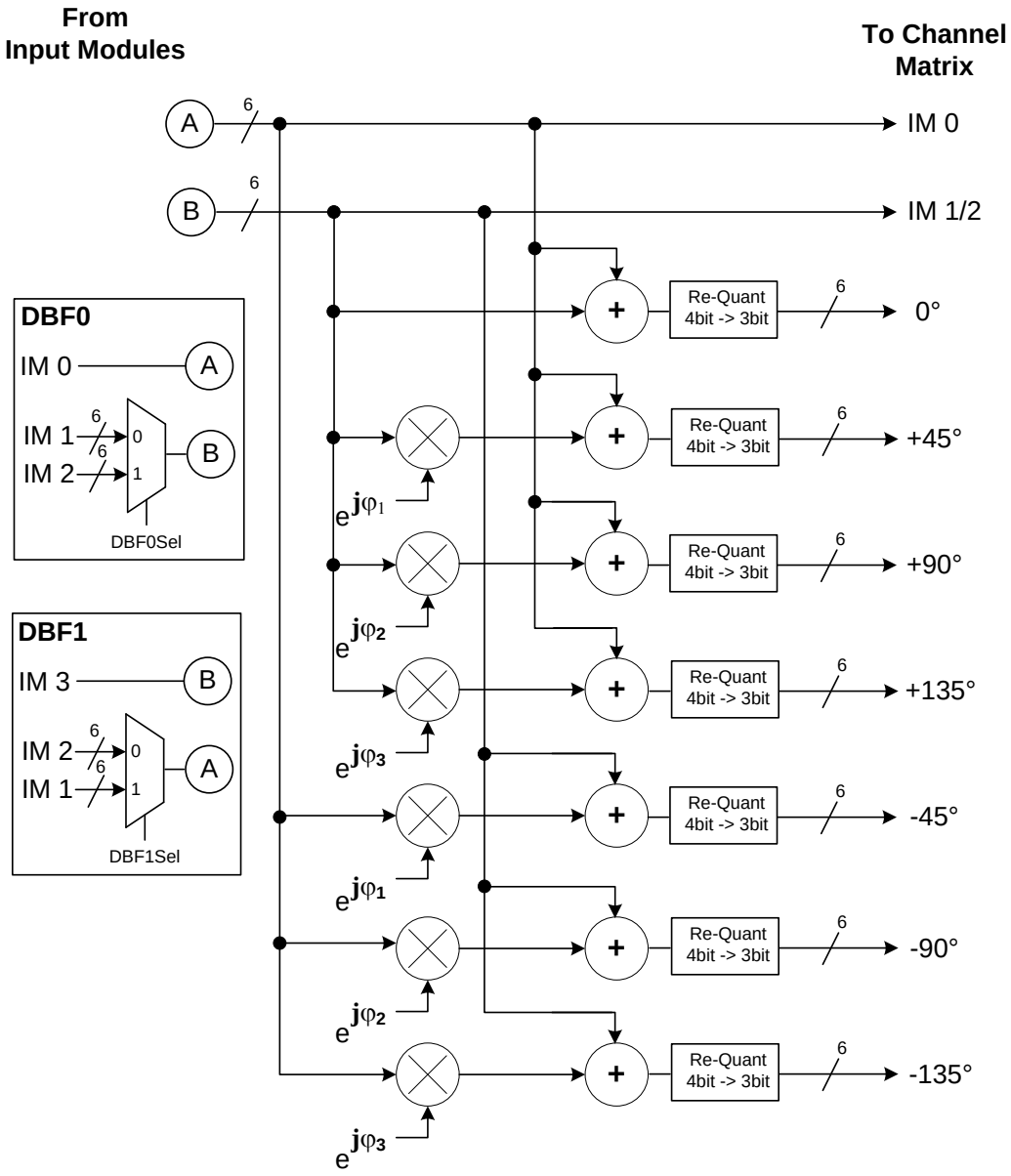


Figure 3-28: Digital Beam Forming Module

The carrier phase of the IM 1/2 signal is shifted in the mixer stage by φ_0 through φ_3 (whereas φ_0 is not shown in the figure below, since: φ_0 =0) and added to the IM 0 signal to obtain the desired beam forming, here referred to as +45°, +90° and +135°. Further, shifting the IM 0 signal by φ_1 through φ_3 and combining the shifted signal with the IM 1/2 signal causes the beams referred to as -45°, -90° and -135°.

It has to be noted that the +/- 90° rotation is lossless and needs no quantization, since every input point in the AGGA-4 notation (-7, -5, -3, -1, +1, +3, +5, +7) before the rotation can be directly be mapped to a rotated point in the AGGA-4 notation.

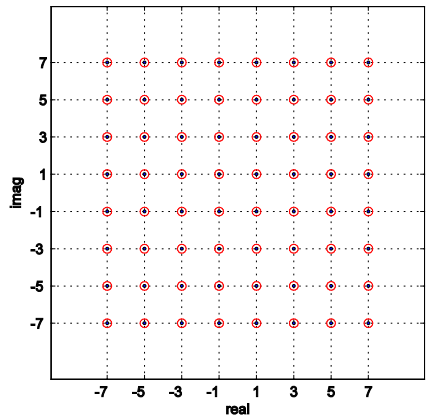
If a rotation of +/-45° is selected then the points before the rotation (in AGGA-4 notation) are mapped to the nearest possible point in the AGGA-4 notation after the rotation.

Example: An input point of (-7/-3) rotated with 45° results in (-2.83/-7.07) which is then mapped to (-3/-7). The mapping is done with a look up table.

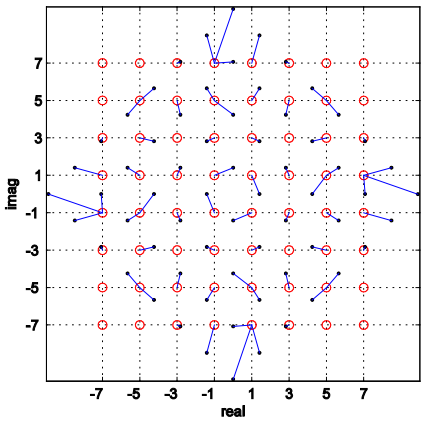
Figure 3-29 depicts the rotation diagrams for every possible rotation. The red circles represent the possible points in AGGA-4 notation. The thicker black dots (not the black dots from the axes grid) are the points after rotation. The blue line shows to which possible AGGA-4 point the rotated point was mapped.

Example: Assuming an input point of (-7/-7). With a 45° rotation it moves to (0/-9.9). This point is then mapped to (1/7).

Note that rotations of 135° are done by doing first a 90° rotation followed by a 45° rotation. That's why the right hand diagram of Figure 3-29 is applicable for 45° and 135° rotations.



+/- 90° rotation



+/- 45° and +/- 135° rotation

Figure 3-29: DBF Rotation operation

After the rotation, the signals from the two input modules are added. The input signals to the beam forming are 3-bit wide (I and Q). After the addition of two signals (e.g. IM0+ IM1) they have 4-bit. They are immediately re-quantized to 3-bit by taking the following LUT (Table 3-8):

Input to Requantization	-14	-12	-10	-8	-6	-4	-2	0	2	4	6	8	10	12	14
Output of Requantization	-7	-7	-5	-5	-3	-3	-1	+1	+1	+3	+3	+5	+5	+7	+7

Table 3-8: Requantization Table of Beamforming Addition

3.4 Channel Matrix with Channels

As anticipated in Figure 3-1, the Channel Matrix can be understood as the housing of all channels, the Time Base Generator (TBG) and the Antenna Switch Controller (ASC). The functionality of these modules is described in the corresponding chapters.

Figure 3-30 gives an overview to an AGGA-4 channel. The whole AGGA-4 inhabits 36 channels. By the programming of the corresponding channel bit in the *ChActivation0 and ChActivation1* registers, the clock to each channel can be gated on or off. Note that a channel's registers are automatically reset to their default values defined in chapter 7 if gated on/off. The *PrimaryRAM1/2* and *SecondaryRAM1/2* memories are not reset.

AGGA-4 channels can be slaved in a hardware manner (like it was the case in AGGA-2 and AGGA-3) or in a software manner (which is new compared to AGGA-2 and AGGA-3). For further details about software slaving function see 3.4.8.

For the hardware slaving each channel outputs a variety of signals (see also Figure 3-30) which can be used by the next channel as input signals. Note that with hardware slaving only subsequent channels can be slaved. Therefore if the wording "previous channel" is used in the following chapters it always means channel-1. If the wording "next channel" is used it always means channel+1. There is one speciality which is the first and the last channel. The "next channel" seen from channel 36 is channel 0 (wrap around functionality). Also the previous channel seen from channel 0 is channel 36.

Software slaving has not been possible in the AGGA-2 and AGGA-3 since only the MSBs of the Carrier and Code NCOs have been visible to the software. Therefore it was not possible for the software to make sure that two independent channels are running coherently not only in frequency, but also in phase. This is now possible since the full 32bit NCO states of Code and Carrier can be made visible to the software. The advantage of software slaving is that every channel can be slaved with every channel (not only the next and previous one). The disadvantage is more effort for the software, since the control words for the channels have to be copied not only in the master channel, but also to the slave channels and that also in time (typically before the next Integration Epoch). Figure 3-30 also shows the internal connections to the Test Support Signals, see 6.7

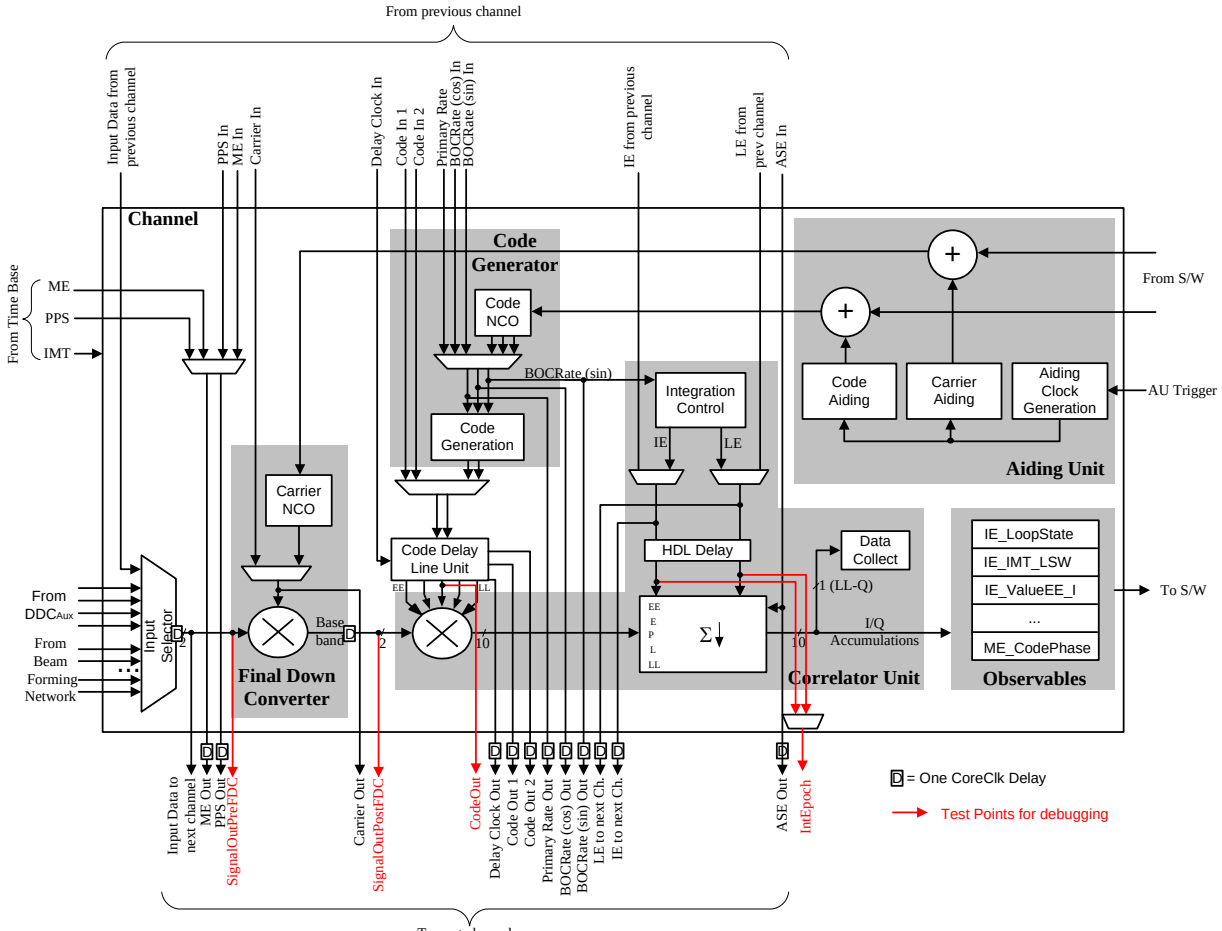


Figure 3-30: Channel Overview

3.4.1 Input Selector

The Input Selector selects an I/Q signal

- from the previous channel
- directly from one of the four Input Modules (DDC_{Aux} streams)
- one beam out of the beam forming network.

Each I/Q signal provides a complex sample every *CoreClk*, consisting of the in-phase component and the quadrature component. Each I and Q component is represented in the internal 3 bit format corresponding to the values [-7, -5, -3, -1, +1, +3, +5, +7]. In order to select the input signal, the *InputSel* field of the *ChannelCtrl* register has to be programmed.

3.4.2 Final Down Converter (FDC)

The Final Down Converter consists of the Carrier Generator and the Carrier Mixer as shown in Figure 3-31. It removes the residual carrier from the 2*3bit complex input signal selected from the Input Selector by subtracting the value of the Carrier Generator from the angle represented by the input signal. The residual frequency can include the Intermediate Frequency after the DDC, the R2C or after the IFC. Furthermore it can include the Doppler shift and the local oscillator offset.

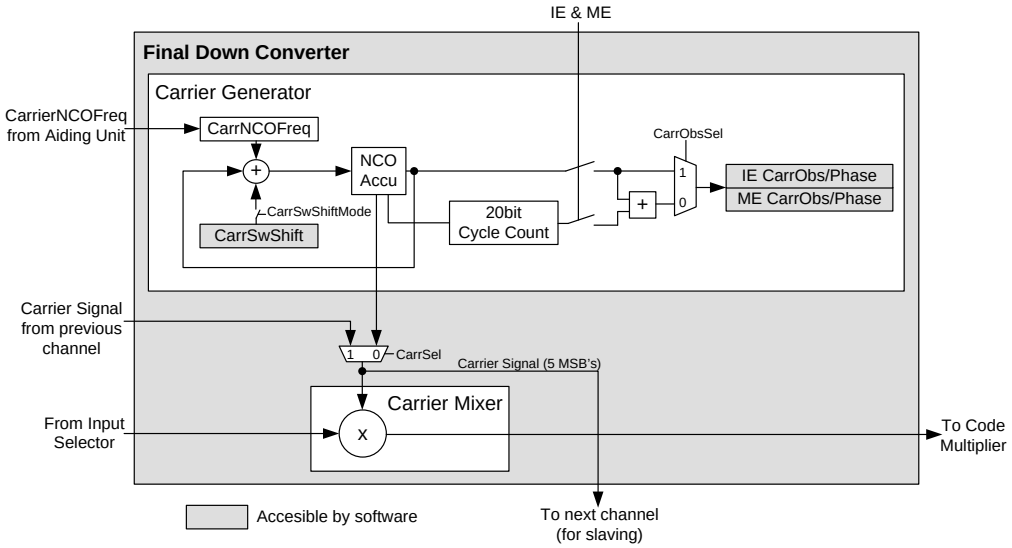


Figure 3-31: Final Down Converter

3.4.2.1 Carrier Generator

The Carrier Generator provides the carrier replica to convert the navigation signal to baseband and to strip off the Doppler frequency. The frequency of the carrier NCO is determined by the signed 32 bit *CarrNCOFreq* increment which comes from the Carrier Aiding Unit. The *CarrNCOFreq* increment gets accumulated with every *CoreClk* cycle in the NCO accumulator. Additionally the phase can be adjusted by the signed 32bit *CarrSwShift* register. A written value to *CarrSwShift* is added once to the NCO accumulator either immediate or with the next integration epoch, depending on the programming of the *CarrSwShiftMode* bit in the *NCOSettings* register. Note that the current phase (32 bit NCO accumulator value) is returned in the *CarrSwShift* register if it is read.

The valid range of *CarrSwShift* is from -2^{31} to 2^{31}-1, corresponding to phase shifts of up to ±180 deg. with a resolution of approx. 84 ndeg.

As soon as the NCO accumulator wraps around the unsigned 20bit Cycle Counter is either incremented or decremented, depending whether the NCO generates a positive or negative frequency.

At each Integration and Measurement Epoch the cycle count and the NCO accumulator value are latched. They are transferred to the *IE_CarrObsPhase* and *ME_CarrObsPhase* registers. By programming the *IE_CarrObsSel* and the

ME_CarrObsSel bit in the *CorrUnitCtrl* register, the user can select whether he wants to have all 32bit of the NCO accumulator latched in the observable or if he wants a mixture of cycle count and NCO accumulator value. If mixture is chosen, the 12MSBs of the NCO accumulator are written in the lower bits of the observable and the 20 bit cycle count is written in the upper bits of the observable. Note that the observables are updated every IE and ME either in mixture OR pure phase mode.

By the programming of the *CarrSel* bit in the *ChannelCtrl* register it can be selected whether the Carrier Mixer shall take the local generated *Carrier* signal or the *Carrier* signal of the previous channel (hardware slaving). Note that the *IE_CarrObsPhase* and the *ME_CarrObsPhase* observables are always driven by the local NCO, no matter if the Carrier Mixer uses the local or the slaved *Carrier* signal.

The 5 MSBs of the carrier NCO accumulator are output as *Carrier* signal, having 32 possible values.

The NCO output frequency is calculated according to the following equation:

$$f_{out} = \frac{CoreClk \cdot \Delta\Phi}{2^{27}} \cdot \frac{11.25\,\mathrm{deg}}{360\,\mathrm{deg}} = \frac{CoreClk \cdot \Delta\Phi}{2^{32}}$$

where f_{out} is the output frequency and $\Delta\Phi$ is the signed phase register increment (*CarrNCOFreq*). The frequency range is therefore $\pm CoreClk/2$ with a resolution of $CoreClk/2^{32}$.

3.4.2.2 Carrier Mixer

The complex input and output signals use the internal representation, corresponding to the values [-7, -5, -3, -1, 1, 3, 5, 7] for the in-phase component and the quadrature component.

Down conversion is performed in 3 steps. First the input sample given by the I and Q value is transformed into polar coordinates with the help of a LUT. Then, the phase shift given by the *Carrier* signal is subtracted from the angle of the input signal. In the third step the shifted input value given by the magnitude and the new angle is retransformed to Cartesian coordinates with the help of a LUT. The entries of the LUT give for each possible magnitude and each possible phase the corresponding (minimum Euclidian Distance) IQ value of the possible set of IQ values.

The LUTs only exist for the first quadrant of the signal constellation. Every signal which is not in this area has to be transformed to this section. For example, if the input signal is detected to be in the second quadrant it is rotated back for 90° by applying the following equation:

$$I_r = Q_i$$
$$Q_r = -I_i,$$

where the subscript *r* identifies the rotated signal and *i* the input signal.

Figure 3-32 illustrates the down conversion. The red circles are the possible values of the signal space. The black squares are all possible samples after the rotation of the input samples. The blue lines show for each rotated sample to which signal space point it is mapped.

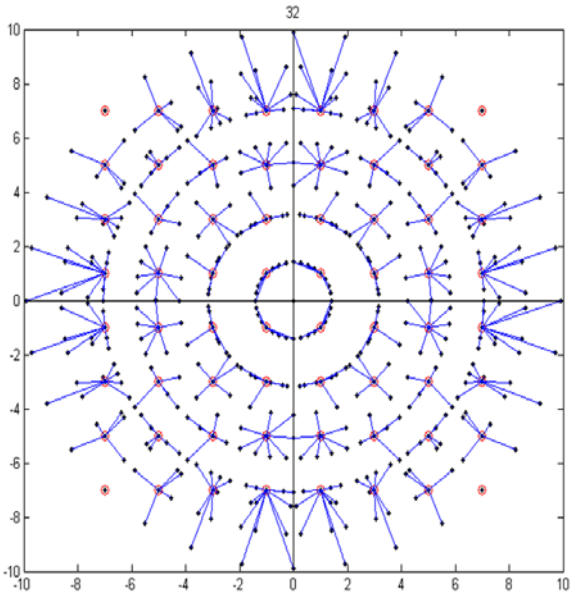


Figure 3-32: Complex 3 bit signal constellation with 32 possible NCO values

3.4.3 Code Generator Unit

The Code Generator Unit is capable of generating two independent codes. Each code can consist of primary and secondary code. Additionally these codes can be multiplied with a 2 bit Binary Offset Carrier (BOC) sin or cosine pattern. Also time multiplexed codes (like GPS L2C) are possible. In this case the time multiplexed code is routed to Output 1, while the enable/disable signal is routed to Output 2. Note that no AltBOC is supported. TMBOC or CBOC signals can be tracked with minimal losses (e.g. the low frequency part of the BOC is tracked, while the high frequency part of the BOC is not correlated).

The Code Generator Unit can be started either at the next Integration Epoch or Measurement Epoch. This can be selected by the configuration bit *Trigger* in the *CodeGenUnitCtrl* register.

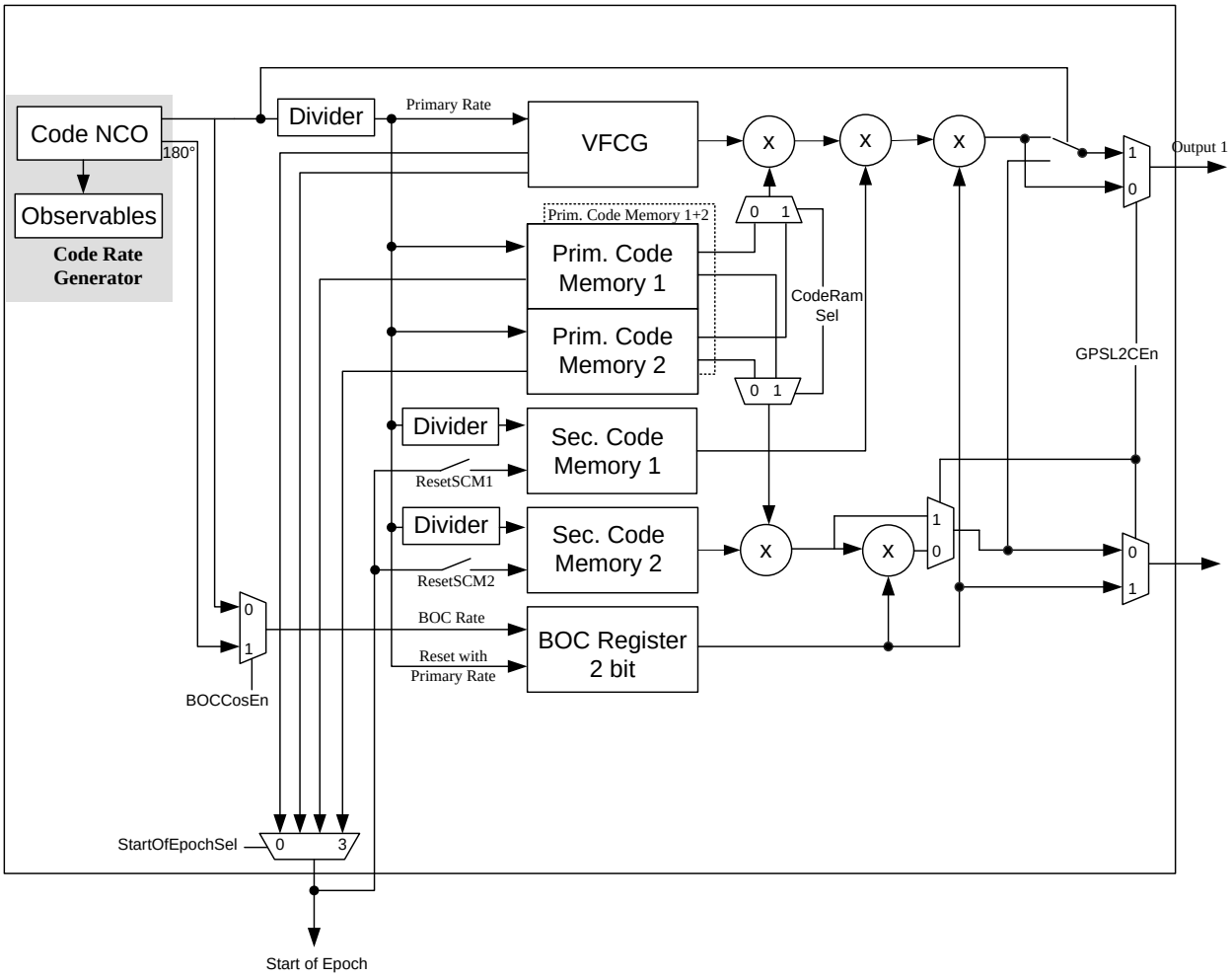


Figure 3-33: Code Generator Unit

The Very Flexible Code Generator (VFCG) consists of a 2x14 stages Linear Feedback Shift Register (LFSR) which can be coupled together to a 1x28 stage LFSR. With this configuration all today known shift register codes can be generated, except the GPS P-code.

Each primary code memory consists of 5120 bits. Each secondary code memory consists of 100 bits. It is possible to combine primary code memory 1 and primary code memory 2 to one primary code memory with 10240 bits. The output of the combined memory is then either routed to Output 1 or to Output 2 (see Figure 3-33).

With this configuration all today known pilot/data code combinations can be processed within one channel. Table 3-9 shall give an idea how the Code Generator Unit can be configured in order to generate the signals. The numbers below the column VFCG refer to the LFSR stages needed to generate the signal. The numbers below the column of the code memories refer to the number of bits needed in order to generate the signal. Note that the examples on this table shall only give an idea how the Code Generator can be configured. It is by far not complete, since there are almost unlimited combinations to generate the different signals.

	VFCG	Prim. Code 1	Sec. Code 1	Prim. Code 2	Sec Code 2	BOC
GPS CA Code	2x10	---	---	---	---	---
	---	1023	---	---	---	---
GPS L2C	1x27	---	---	10230	---	„1“„0“or„0“„1“
GPS L5 I+Q	2x13	---	20	10230	10	---
Galileo E1B+C	---	4092	25	4092	0	„1“ „0“
Galileo E5A I+Q	2x14	---	100	10230	20	---

Table 3-9: Example with possible Code Generator Unit configuration

In general it can be said that the Code Generator Unit is able to produce codes with the following properties

Single Component Code (consisting of either pilot or data)

- Shift register codes with an length up to 2^{28}-1 bit
- Memory codes (primary) with a length up to 10240 bit
- Memory codes (secondary) with a length up to 100 bit
- BOC code with a sequence length up to 2 bit

Double Component Code (consisting of pilot and data)

- One component 2^{28}-1 bit, the other 10240 if one component can be generated with LFSR
- Both components of 5120 bit each, if both components are memory based codes
- Memory codes (secondary) with a length up to 100 bit
- BOC code with a sequence length up to 2 bit

Reloading of secondary codes is not the preferred solution, but it is possible due to the low data rate of secondary codes. Therefore, any future signal (e.g. GPS L1C) with secondary codes longer than 100 bits, but still fulfilling the other criteria's mentioned above, can still be processed.

Having a look at the signals where the properties are already known, the Code Generator Unit will be able to generate the following signals within each of the 36 channels:

Signal	Yes	No
GPS L1 CA	x	
GPS L1 P-Code		x
GPS L2 P-Code		x
GPS L2CM	x	
GPS L2CL	x	
GPS L2C (M+L)	x	
GPS L5 I	x	
GPS L5 Q	x	
GPS L5 (I+Q)	x	
Galileo E1 B	x	
Galileo E1 C	x	
Galileo E1 (B+C)	x	
Galileo E5B I	x	
Galileo E5B Q	x	
Galileo E5B (I+Q)	x	
Galileo E5A I	x	
Galileo E5A Q	x	
Galileo E5A (I+Q)	x	
Galileo E5Q AltBOC		x
Beidou B1 I	x	
Beidou B2 I	x	
Beidou B3 I	x	

Table 3-10: December 2009 Signal Summary

3.4.3.1 Very Flexible Code Generator (VFCG)

The Very Flexible Code Generator consists of a 2x14 stage LFSR, which can be coupled together to a 1x28 stage LFSR and a 2x14 bit counter, which can be coupled together to a 1x28 bit counter. With this configuration all today known shift register codes can be generated, except the GPS P-code.

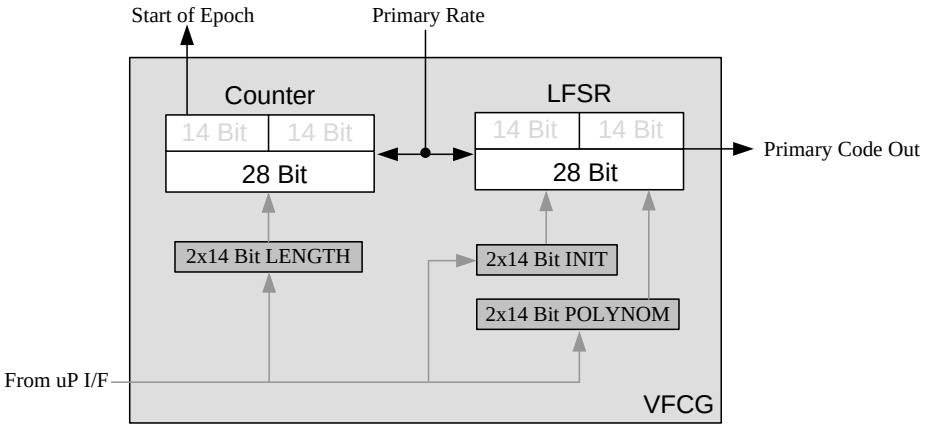


Figure 3-34: Very Flexible Code Generator (Long Mode)

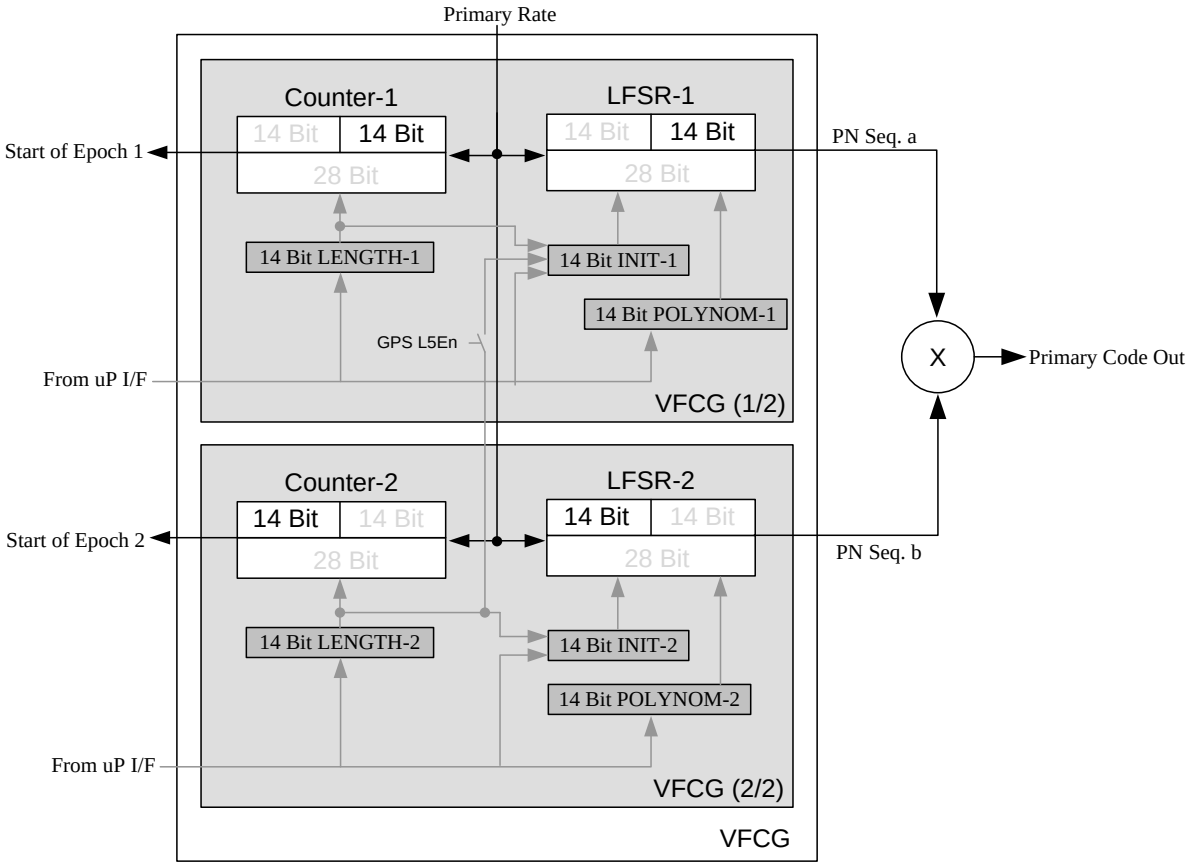


Figure 3-35 : Very Flexible Code Generator (Normal Mode)

The *StartOfEpoch* Signal arises between the edge of the last and the first chip of a code epoch. It is used to reload the VFCG LFSR's with the init patterns. Also it can be used to synchronize the Integration Epoch of the correlators with

the code epoch. The VFCG length registers can be reloaded at any time to perform a quick L2CL adjustment once the L2CM code has been acquired.

In principal the VFCG can be operated in two modes:

In the "long mode" (*VFCGLongEn* = 1, **Figure 3-34**) the VFCG appears to have:

- one 28 bit LFSR
- one 28 bit length counter
- one 28 bit init register
- one 28 bit polynom register

In the "normal mode" (*VFCGLongEn* = 0, **Figure 3-35**) the VFCG appears to have:

- two 14 bit LFSR
- two 14 bit length counter
- two 14 bit init register
- two 14 bit polynom register

In "long mode" the LFSR is reloaded with the init value when either the VFCG is started or the length counter has elapsed.

In "normal mode" the reloading of the sift registers is as following:

- LFSR 1 and 2 are reloaded when theVFCG is started
- LFSR 1 is reloaded when the length counter 1 has elapsed
- LFSR 2 is reloaded when the length counter 2 has elapsed
- LFSR 1 is reloaded when the length counter 2 has elapsed (this is only done when the configuration bit *GPSL5En* is set to 1)

The architecture of the LFSR in normal mode is depicted in Figure 3-36. The architecture of the LFSR in long mode is depicted in Figure 3-37. The init register is not shown in this figure as it simply preloads the LFSR register. The "+" in the two figures are corresponding to XOR logic. The shift direction of the LFSR is left wise. The feedback taps in the figure have been arbitrarily chosen. They are determined by the programming of the *VFCGExtTaps* register. With this knowledge all existing PRN Codes can be concluded for the AGGA-4. **Table 3-11** gives the initialization vectors for all Galileo and GPS shift register codes described in [RD-01] to [RD4]. The numbers are represented in 32bit hexadecimal format and can therefore directly be pasted into source code.

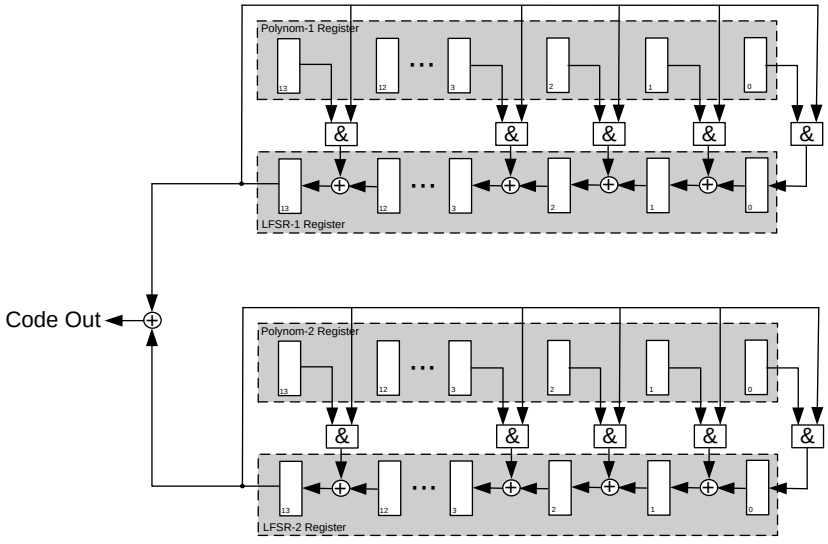


Figure 3-36: LFSR architecture in "normal mode"

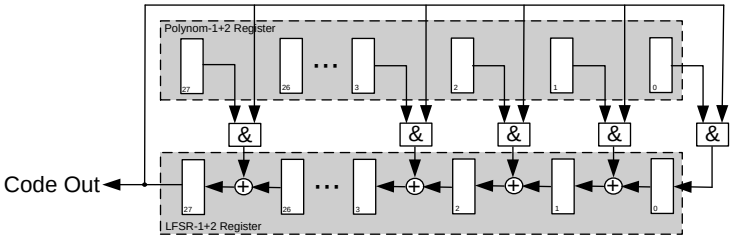


Figure 3-37: LFSR architecture in "long mode"

PRN	Galileo				GPS				
	E5A-I	E5A-Q	E5B-I	E5B-Q	CA	L2CM	L2CL	L5-I	L5-Q
1	0x0C8760C0	0x0A0CE0C0	0x03527C06	0x0166BC06	0x03C03800	0x02DF851E	0x05FD30A6	0x03293FEC	0x02F2BFEC
2	0x067E60C0	0x02A260C0	0x099DFC06	0x02817C06	0x01E03800	0x0B8181DE	0x04F08D8A	0x0E61BFEC	0x05283FEC
3	0x0B59A0C0	0x0A9E20C0	0x00217C06	0x0F967C06	0x00F03800	0x02673D00	0x07007824	0x01B6BFEC	0x0B0DBFEC
4	0x0880A0C0	0x0C47E0C0	0x06423C06	0x0D0CFC06	0x00783800	0x02BD69B0	0x0223044E	0x05DABFEC	0x07383FEC
5	0x097E60C0	0x0B47E0C0	0x07AEFC06	0x0060BC06	0x07F83800	0x09CE0606	0x0A763200	0x08F33FEC	0x06793FEC
6	0x0D68E0C0	0x0A87A0C0	0x01053C06	0x08F53C06	0x03FC3800	0x0DBACB0E	0x06B64350	0x07F23FEC	0x0C7E3FEC
7	0x06B160C0	0x0D4C60C0	0x00E43C06	0x04253C06	0x07F03800	0x01C094A8	0x07D45556	0x097EBFEC	0x0DAABFEC
8	0x0527A0C0	0x0B3420C0	0x0993FC06	0x035E7C06	0x03F83800	0x08F39BC6	0x0FF95184	0x03103FEC	0x01E33FEC
9	0x00DF60C0	0x03C0E0C0	0x00CF3C06	0x07787C06	0x01FC3800	0x089C3790	0x03F676C0	0x0AB73FEC	0x0B5ABFEC
10	0x0C9060C0	0x00F8E0C0	0x0A31FC06	0x0812BC06	0x02003800	0x0644C36E	0x07C2563A	0x04FABFEC	0x05AA3FEC
11	0x0B4320C0	0x0500A0C0	0x017A3C06	0x0A437C06	0x01003800	0x0A62974E	0x0AAE7320	0x07D0BFEC	0x0ED03FEC
12	0x038DE0C0	0x0DBAA0C0	0x095AFC06	0x0C53BC06	0x00403800	0x061FC4A0	0x014BFFC8	0x0C963FEC	0x0EFC3FEC
13	0x0FA460C0	0x074EA0C0	0x0F1BFC06	0x0123BC06	0x00203800	0x0C016A20	0x0B739586	0x0212BFEC	0x0EAC3FEC
14	0x07C5A0C0	0x072D20C0	0x0DE1FC06	0x0ACB7C06	0x00103800	0x0975AC64	0x0C77C300	0x0866BFEC	0x09083FEC
15	0x036CA0C0	0x0F4860C0	0x0386FC06	0x0E20FC06	0x00083800	0x00099A00	0x0AED9590	0x07893FEC	0x09AE3FEC
16	0x069060C0	0x0152A0C0	0x03093C06	0x03A97C06	0x00043800	0x062C4124	0x089C9676	0x0C293FEC	0x09573FEC
17	0x0AC7A0C0	0x0135A0C0	0x0E0DBC06	0x01E1FC06	0x07003800	0x0681681A	0x00920686	0x09AABFEC	0x0198BFEC
18	0x048260C0	0x036DA0C0	0x0513FC06	0x0A56BC06	0x03803800	0x0A3C5684	0x0E13F500	0x04DA3FEC	0x032B3FEC
19	0x0A95A0C0	0x0A6A60C0	0x00663C06	0x002FBC06	0x01C03800	0x026240B0	0x094E72AA	0x096ABFEC	0x0A3B3FEC
20	0x0063E0C0	0x050460C0	0x099A7C06	0x06737C06	0x00E03800	0x03D47028	0x0AFF55B4	0x0F35BFEC	0x0D393FEC
21	0x04BA20C0	0x021960C0	0x05F2FC06	0x0EB27C06	0x00703800	0x0DAE4090	0x0C387D80	0x01B43FEC	0x00CFBFEC
22	0x07E1A0C0	0x04C4E0C0	0x05C27C06	0x08C1FC06	0x00383800	0x0EB13CAE	0x0769E60A	0x09F23FEC	0x0410BFEC
23	0x0B3760C0	0x0007A0C0	0x0D4A3C06	0x0A1CFC06	0x06003800	0x0FEE3E90	0x0C4F3F1E	0x049E3FEC	0x0FEC3FEC
24	0x0C4DE0C0	0x033B20C0	0x0A16BC06	0x020B3C06	0x00C03800	0x00DC0A1E	0x06B9D6C6	0x03C23FEC	0x0BF03FEC
25	0x05F860C0	0x020660C0	0x0D89FC06	0x0CF1FC06	0x00603800	0x03A1E80E	0x008C4F3E	0x0F3D3FEC	0x0A12BFEC
26	0x061CE0C0	0x08BCA0C0	0x080E3C06	0x0C7A7C06	0x00303800	0x08D72840	0x03CF5C2E	0x057CBFEC	0x02733FEC
27	0x0EF360C0	0x04FC60C0	0x0ABAFC06	0x06CE7C06	0x00183800	0x0A4EC74E	0x0C44540E	0x04F43FEC	0x07F03FEC
28	0x09A3A0C0	0x09EAA0C0	0x0E027C06	0x06783C06	0x000C3800	0x04E15BEE	0x0CD77524	0x0556BFEC	0x06E3BFEC
29	0x0CD5E0C0	0x0CE520C0	0x01723C06	0x0958BC06	0x07C03800	0x038F4E4C	0x02357D68	0x0DF23FEC	0x0304BFEC
30	0x0B5020C0	0x0A8820C0	0x0F097C06	0x06FA7C06	0x03E03800	0x0E02A44E	0x0A52D99E	0x08A73FEC	0x0C8FBFEC
31	0x00DEA0C0	0x03D9E0C0	0x0923FC06	0x0FD0BC06	0x01F03800	0x0DA2652E	0x06C81908	0x04C33FEC	0x0E9C3FEC
32	0x067B60C0	0x0DDAA0C0	0x0B34FC06	0x04393C06	0x00F83800	0x0D12F050	0x073CCAD4	0x0CA2BFEC	0x0700BFEC
33	0x0C7260C0	0x02B1E0C0	0x0CA83C06	0x0BFE7C06	0x007C3800	0x0C3EAC0A	0x083C6FE2	0x0DA43FEC	0x035DBFEC
34	0x09DFA0C0	0x0B2560C0	0x0E8B7C06	0x0C94FC06	0x00FC3800	0x078FE2DE	0x043723CE	0x0C9E3FEC	0x035C3FEC
35	0x0D76A0C0	0x01E0E0C0	0x0E9BBC06	0x0C73FC06	0x07E03800	0x08F79DE8	0x0E118D24	0x022BBFEC	0x0BCB3FEC
36	0x0168A0C0	0x0113A0C0	0x04D97C06	0x06D6BC06	0x03F03800	0x0D4DEDDE	0x0E36523A	0x019CBFEC	0x09C33FEC
37	0x086460C0	0x0EF920C0	0x032B7C06	0x083ABC06	0x00FC3800	0x052316E2	0x0DC69C14	0x00C53FEC	0x0D4DBFEC
38	0x0E4CA0C0	0x029760C0	0x050C3C06	0x0482FC06					
39	0x0E6C60C0	0x0F2DE0C0	0x0D01FC06	0x05E87C06					
40	0x0A80A0C0	0x076360C0	0x01083C06	0x0FD3FC06					
41	0x0651A0C0	0x0FB420C0	0x0202FC06	0x0FE67C06					
42	0x0B7DA0C0	0x0348A0C0	0x0F7B3C06	0x0DFC7C06					
43	0x0349A0C0	0x0B8FE0C0	0x07B7FC06	0x0334BC06					
44	0x0C7420C0	0x07F1E0C0	0x0E69BC06	0x04813C06					
45	0x0EC9E0C0	0x0E2160C0	0x0F53BC06	0x03113C06					

46	0x0A4E20C0	0x0FB120C0	0x0C41BC06	0x01D93C06
47	0x08A0E0C0	0x057EA0C0	0x0993BC06	0x06BE7C06
48	0x0C4960C0	0x0BBB20C0	0x0D3F3C06	0x08057C06
49	0x07FCA0C0	0x0834A0C0	0x03AC3C06	0x0E393C06
50	0x05D6E0C0	0x0A27A0C0	0x046CFC06	0x0A94FC06

Table 3-11: LFSR Init Patterns for different codes

3.4.3.2 Primary Code Memory

Each channel contains 2x5120 bit memory for primary codes. The memory has a width of 8 bit, which results in the smallest ASIC die area and therefore saving gate count compared to a 32bit organized on chip memory. The two memories can be coupled together in order to have one large code memory with 10240 bits. This can be done by configuring the *Length* field in the *PrimCodeRAM1* register beyond 5120bits. The primary code memory is clocked with the Primary Rate. Whenever the current pointer reaches the length pointer, the current pointer returns to zero and the Start of Epoch signal is generated. When the *StartCM1* or *StartCM2* bit of the *CodeGenUnitCtrl* register is set, the current pointer is set equal to the offset pointer once and the current pointer starts moving with the next code generator trigger, which can be either the Integration Epoch or the Measurement Epoch depending on the programming of the *Trigger* bit in the *CodeGenUnitCtrl* register. In the time between setting the *StartCM1* or *StartCM2* bit and waiting for the trigger (IE or ME), the code memory is paused and is outputting the value of the chip corresponding to *Offset*. Note that typically the user wants to align the Integration Epoch (IE) with the Code Epoch. This can be achieved by setting the bit *ResetAtStartofEpoch* in the *CorrUnitCtrl* register to reset the accumulators and the counters for the Integration Epoch at the time instance "Start of Epoch". Please take into account that two consecutives Integration Epoch (IE) can occur close together if synchronising to the Code Epoch. It is valid to set the two bits *StartCM1*/*StartCM2* together with *ResetAtStartofEpoch* in order to minimize the setup time of the code memory. However it is important that the *StartCM1* or *StartCM2* bit is written before the *ResetAtStartofEpoch* bit.

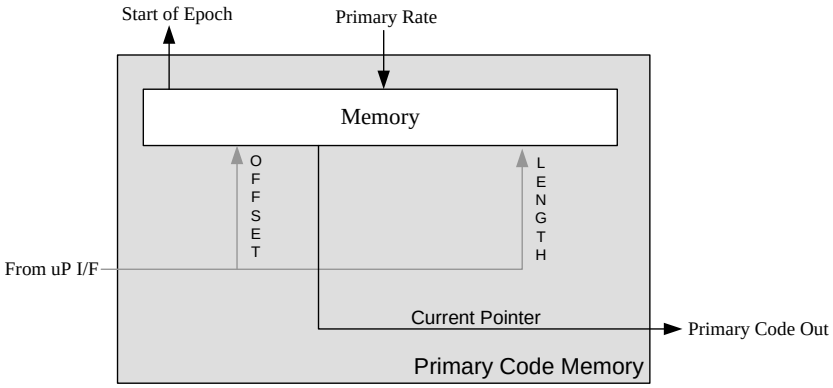


Figure 3-38: Primary Code Memory

The offset pointers as well as the length pointers are located in the *PrimCodeRam1Ctrl* respectively *PrimCodeRam2Ctrl* register and can be set by software. If the two RAM's are coupled together, the length and offset pointer are configured via *PrimCodeRam1Ctrl*. The settings in *PrimCodeRam2Ctrl* have no influence in this case. Note that the length as well as the offset pointer can address not only byte wise, but also bit wise. Note also that the offset pointer always has to be smaller or equal than the length pointer. Otherwise the output of the Code Generator is invalid.

Figure 3-39 depicts how the code chips have to be written into the PrimaryRAM1/2 in order that they are output in the correct order.

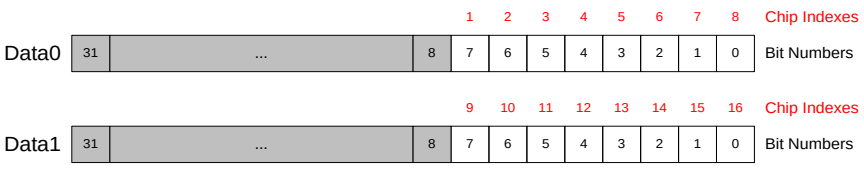


Figure 3-39: Data Organization of Code RAM

To ensure proper operation, the primary RAM should be configured as follow: first the length and offset pointers have to be set and after this the start bit has to be set. From there on it takes 2 *CoreClk* cycles until one single code RAM or both RAM's coupled to one single RAM are ready to start. If both code RAM's are used separately it takes 4 *CoreClk* cycles until they are both ready to start.

It is guaranteed by the hardware that the trigger can not start the code memories until they are not ready. However if the time between setting the start bit and the next trigger is less than 2 *CoreClk* cycles (one RAM single/coupled) respectively 4 *CoreClk* cycles (two single RAM's), the trigger is missed and the next trigger starts the Code RAM(s).

Note that for the code RAM, code lengths smaller than 16 chips are not possible due to timing constraints. That means if the code length is programmed to values 0..15 the code RAM is disabled. In that case the output of the code RAM is a constant one, no matter what is written in the code RAM.

Note that a proper use of the Code RAM's is only guaranteed if a channel is enabled in the *ChActivation0 and ChActivation1* registers.

3.4.3.3 Secondary Code Memory

Each channel contains 2x100 bit memory for secondary codes. The memory has a width of 8 bit, which results in the smallest ASIC die area and therefore saving gate count compared to a 32bit organized on chip memory. Each memory is clocked with the Secondary Rate. The Secondary Rate is generated by dividing the Primary Rate by a 14 bit integer number. The divider can be programmed via the *Divider* field in the *SecCodeRamCtrl* register. Each secondary memory has its own divider in order to have independent Secondary Rates for the two memories (see also Figure 3-33). Whenever the current pointer reaches the length pointer, the current pointer returns to zero. Additionally the current pointer of each secondary code memory can be set to the offset pointer with the next *StartOfEpoch* signal. If an offset is written (by writing the the *Offset* field in the *SecCodeRam1Ctrl* or *SecCodeRam2Ctrl* register), the next step is to set the *ResetSCM1* or *ResetSCM2* bit in the *CodeGenUnitCtrl* register. Then at the next *StartOfEpoch* signal, the current pointer will start reading the secondary code from the offset on. The offset value always is the offset from the start of the secondary code RAM. If the offset pointer is read it returns the current pointer position. This is helpful for the secondary code acquisition. Note that the offset pointer always has to be smaller or equal than the length pointer. Otherwise the output of the Secondary Code is invalid.

Also the length pointer can be set by the programming of the *Length* field in the *SecCodeRam1Ctrl* or *SecCodeRam2Ctrl* register. Other than the offset pointer, the length pointer becomes effective immediately. It is also valid to change the length pointer during runtime. A programmed length of "0" disables the secondary code RAM and a constant "0" is coming out.

The data is organized the same way as it is done for the primary code. Therefore Figure 3-39 also applies for the SecondaryRAM1/2 memory.

Note that a proper use of the Secondary Code RAM's is only guaranteed if a channel is enabled in the *ChActivation0 and ChActivation1* registers.

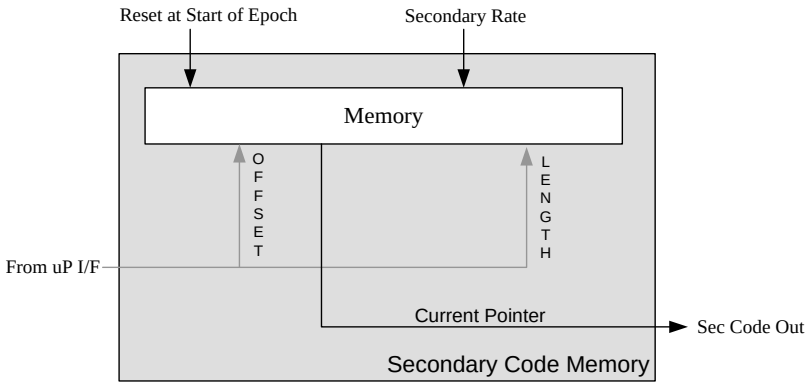
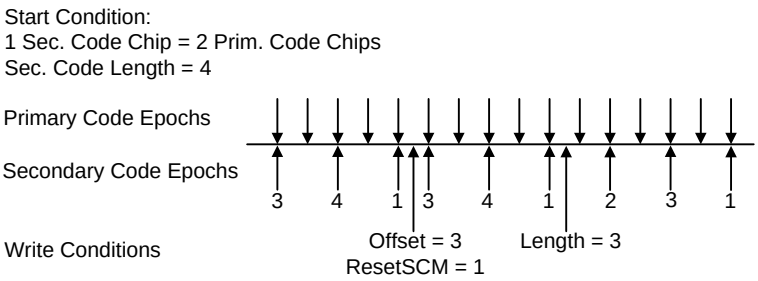


Figure 3-40: Secondary Code Memory



3.4.3.4 BOC Processing

The BOC register can hold up to 2 bit of a BOC sequence (e.g. “10”). Sin and cosine BOC patterns can be generated (more details see Figure 3-41). The BOC/Primary chip ratio can be programmed by the 4 bit *BOCDivider* in the *CodeGenUnitCtrl* register. That way a wide range of BOC(n,m) codes can be processed.

Values written in the following registers fields: *BocPattern, BOCDivider* become effective either immediately or with the next Integration Epoch, depending on the *BOCEffective* bit. This allows switching mode without losing track, e.g. from single side band processing to dual side band processing.

In case of even BOC division ratios (e.g. 2,4,6,…) the BOC sequence is synchronized with every primary chip, while for odd BOC division ratios (e.g. 1,3,5), the BOC sequence is synchronized only with every even primary chip (e.g. 0,2,4,…). Note that in this case the continuous synchronization works properly starting at the next *StartOfEpoch* event, since the hardware uses this to distinguish between even and odd primary chips.

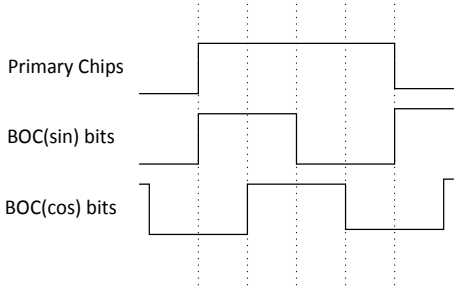


Figure 3-41: BOC Sine and Cosine Logic

The BOCRate(sin) is generated whenever the code NCO accumulator is full and has a wrap around at the 32bit border. The BOCRate(cos) is generated whenever the code NCO accumulator MSB toggles after it has wrapped around at the 32bit border. Therefore BOCRate(sin) and BOCRate(cos) are shifted 180° against each other or 90° relative to the primary chip as depicted in Figure 3-41.

3.4.3.5 Code Rate Generator

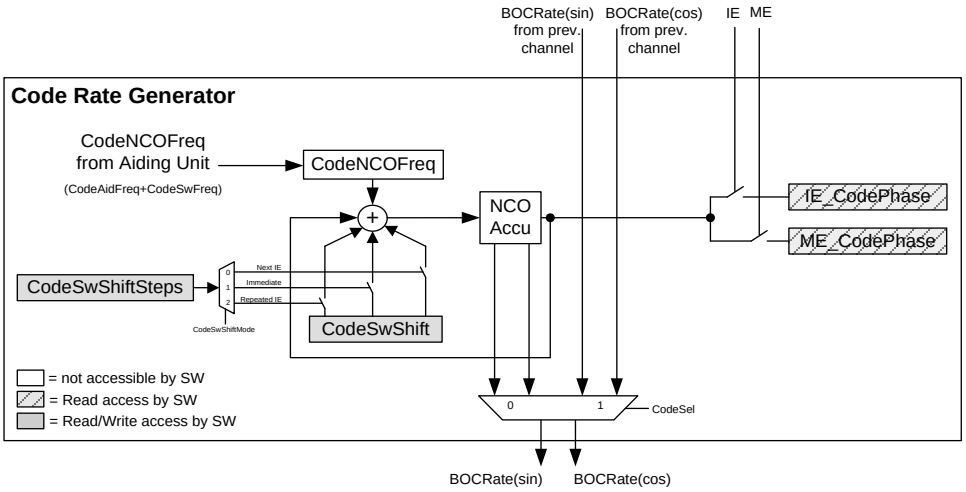


Figure 3-42: Code Rate Generator Architecture

The Code Rate Generator architecture is depicted in Figure 3-42. The code frequency increment (CodeNCOFreq) value is coming from the Code Aiding Unit. It is the summation of *CodeAidFreq* and *CodeSwFreq*. (further details see Code Aiding Unit chapter 3.4.6.1). Besides the code frequency also the code phase can be adjusted. Other than AGGA-2/3, the code phase shifts are not limited to 1/4 and 1/16 of a chip. The code phase shift can be programmed via the register *CodeSwShift* and can be any signed number within 32bit. However the user has to take care that the summation of CodeNCOFreq and *CodeSwShift* is not less than 0 and not more than 2^{32}. Otherwise the outcoming BOCRate is invalid. Also the code phase is observable with all 32bits. Therefore AGGA-4 offers the possibility of software slaving. That means that the code phase of two or multiple channels (which can, but don't have to be adjacent) can be exactly synchronized. The 32bit code phase is latched at Integration and Measurement Epoch in the corresponding registers (*IE_CodePhase* and *ME_CodePhase*). Besides this the current code phase can be read at any time by reading the *CodeSwShift* register.

For acquisition it is helpful to have the possibility to shift the code phase for several chips. This can be achieved with the 22bit *CodeSwShiftSteps* field of the *NCOSettings* register. . If *CodeSwShiftSteps* is not zero, than the *CodeSwShift* value is added *CodeSwShiftSteps* times to the Code NCO accumulator with every *CoreClk*. Therefore it might take *CodeSwShiftSteps CoreClk* cycles until the whole code phase shift is done. Also the user has the possibility to select when the single or multiple code phase shift is applied. This can be selected by the *CodeSwShiftMode* field in the *NCOSettings* register. There is the possibility to do the single/multiple code phase shift "once immediately" or "once at next integration epoch" or "repeated at integration epoch". If "repeated at integration epoch" is selected the user has to take care that the single/multiple code phase shift lasts shorter than the integration epoch period. Otherwise the code phase shift operation can not work as intended. The repeated mode can be stopped by programming either the *CodeSwShift* or the *CodeSwShiftSteps* register to zero.

The *CodeSwShift* is a signed value and can therefore be applied in both directions (forward and backwards) with the above mentioned limitations (*CodeSwShift* + *CodeNCOFreq* between 0 and 2^{32}).

Figure 3-43 is a timing example showing how the different modes behave. Note that the example is not representative concerning the distance between the adjacent IE's. In reality the distance would change due to the Code Phase Shift. Note also that the example assumes a programmed value of "2" for the *CodeSwShiftSteps* register.

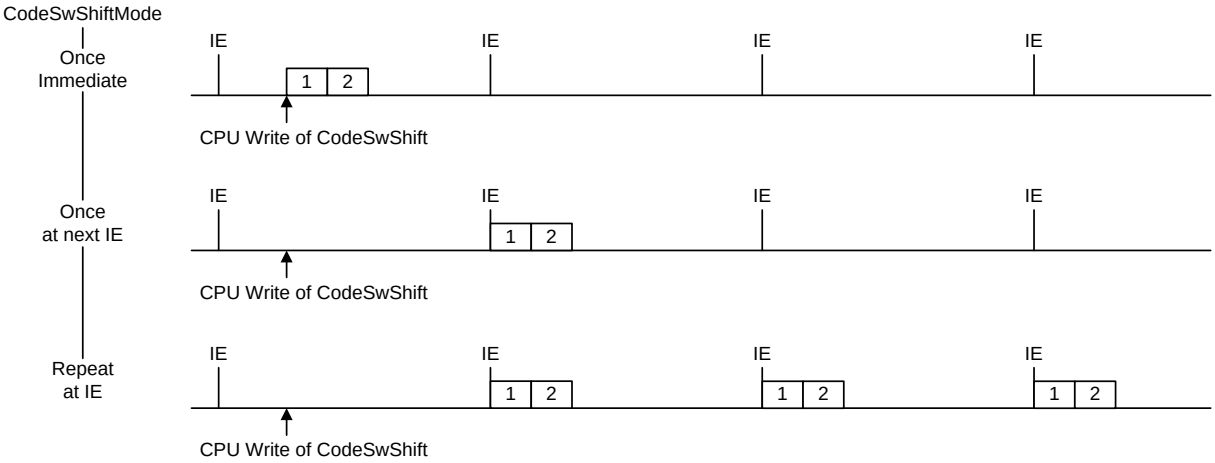


Figure 3-43: Code SW Shift Operation Timing Example

Note: If the CodeNCOFreq is zero (which is the case after a channel reset or if the *CodeNCOFreq* was intentionally set to zero), there is no more *BOCRate* coming out of the Code Rate Generator. Therefore also the Integration Counter can't count *BOCRate* cycles and thus an Integration Epoch will never occur. If in this case *CodeSwShiftMode* would be set to "0-new values become effective with next IE", then new values for the code NCO would never become effective, since no Integration Epoch would ever occur. Therefore in this particular case the *CodeSwShiftMode* has to be set to "1-new values become effective immediately" until the CodeNCOFreq has the desired value. Then Integration Epochs will occur and the *CodeSwShiftMode* can be set back to "0" if desired.

3.4.3.6 Code Generator Unit Configuration Examples:

Please note that in the following example tables (Table 3-12 - Table 3-17) a "X" indicates that the setting does not necessarily have to be "0" or "1", but can be chosen upon the users desires (e.g. Delay Line Spacing, etc).

3.4.3.6.1 GPS C/A Code Generation

The GPS C/A Code is 1023 chips long and can be generated either with the LFSRs of the VFCG, the primary code memory1 or the primary code memory2. If produced via LFSR the VFCG is configured to use two LFSR stages with a length of 10 at a rate of 1.023 MHz. The polynomials to generate GPS C/A code are the following:

- $G1 = 1 + x^3 + x^{10}$
- $G2 = 1 + x^2 + x^3 + x^6 + x^8 + x^9 + x^{10}$

The following settings depict an example of how to configure the Code Generator Unit in order to generate GPS C/A code PRN-1 via the VFCG.

Register	MSB	Setting						LSB
CodeGenUnitCtrl (20bit)				0000	0000	0000	0000	100X
PrimCodeRam1Ctrl (28bit)		0000	0000	0000	0000	0000	0000	0000
PrimCodeRam2Ctrl (26bit)		00	0000	0000	0000	0000	0000	0000
SecCodeRam1Ctrl (28bit)		0000	0000	0000	0000	0000	0000	0000
SecCodeRam2Ctrl (28bit)		0000	0000	0000	0000	0000	0000	0000
VFCGExtTaps (28bit)		0110	0101	1100	0000	1000	0001	0000
VFCGInit (28bit)		0011	1100	0000	0011	1000	0000	0000
VFCGLength (28bit)		0000	1111	1111	1000	0011	1111	1110
DelayLineCtrl (26bit)		XX	XXXX	XXXX	XXXX	XXXX	XX00	00XX
CorrUnitCtrl (17bit)				X	XXXX	XXXX	XX00	0001

Table 3-12: Example settings for GPS C/A code PRN-1 via VFCG

Note that in the previous example it was assumed that the user wanted to synchronize the integration count with the code epoch. Therefore the "*ResetAtStartofEpoch*" bit in *CorrUnitCtrl* register was set.

3.4.3.6.2 GPS L2C Code Generation

For time multiplexed codes, the BOC registers can be used. For GPS L2C for example the BOC Clock Divider will be programmed in a way that the BOC Rate is twice the Primary Rate. The VFCG is configured to produce L2CL while Code Memory2 is configures to produce L2CM. Additionally the *GPSL2CEn* bit in the *CodeGenUnitCtrl* register has to be set. This causes that the BOC register output is not multiplied onto the primary codes, but is forwarded as enable/disable control into Code Output2. Also it causes that the Outputs of the VFCG and the Primary Code RAM are time multiplexed and routed to Code Output1.

By programming the *BOCPattern* field in the *CodeGenUnitCtrl* register it can be determined which chip of the two multiplexed chips is enabled and which is disabled later on in the multiplier. A "1" in the *BOCPattern* field enables the chip later on in the multiplier, a "0" disables it.

The L2C signal contains two codes of different length:

- L2CM containing 10.230 chips with a code epoch length of 20 msec
- L2CL containing 767.250 chips with a code epoch length of 1.5 sec and synchronized to the 1.5 sec Z-count

In principle both codes can be generated via the VFCG with the following polynomial:

- $1 + x^3 + x^4 + x^5 + x^6 + x^9 + x^{11} + x^{13} + x^{16} + x^{19} + x^{21} + x^{24} + x^{27}$

The following table depicts an example of how to configure the Code Generator Unit in order to generate GPS L2CM and L2CL code PRN-1. The pilot (L2CL) signal with the VFCG, the data (L2CM) signal with the coupled primary memory. Only the second half of the chip (L2CL) has been enabled. Therefore in the Correlator Unit all correlators would correlate only on the L2CL part, and not on the L2CM part. The only exception is the LL-Q correlator shown in Figure 3-30, which in this example would correlate only on the L2CM part and therefore demodulate the data. Also in this example the *StartOfEpoch* signal is generated from the code memory and therefore occurs every 20ms rather than every 1.5s as it would be the case for the L2CL signal.

Register	MSB	Setting						LSB
CodeGenUnitCtrl (20bit)				1000	0010	1101	0001	110X
PrimCodeRam1Ctrl (28bit)		0000	0000	0000	0010	0111	1111	0101
PrimCodeRam2Ctrl (26bit)		00	0000	0000	0000	0000	0000	0000
SecCodeRam1Ctrl (28bit)		0000	0000	0000	0000	0000	0000	0000
SecCodeRam2Ctrl (28bit)		0000	0000	0000	0000	0000	0000	0000
VFCGExtTaps (28bit)		0011	1100	1010	1001	0010	1001	0010
VFCGInit (28bit)		0101	1111	1101	0011	0000	1010	0110
VFCGLength (28bit)		0000	0000	1011	1011	0101	0001	0001
DelayLineCtrl (26bit)		XX	XXXX	XXXX	XXXX	XXXX	XX00	00XX
CorrUnitCtrl (17bit)				X	XXXX	XXXX	XX11	1001

Table 3-13: Example settings for GPS L2C PRN-1 via VFCG and coupled Primary Memory

3.4.3.6.3 GPS L5 Code Generation

The GPS L5 signal is composed of an in-phase data signal and a quadrature pilot signal. Both signals have a 10.23 MHz chip rate. Both L5 codes can be generated with two 13 stage LFSRs with the following polynomials:

- $P_{XA} = 1 + x^9 + x^{10} + x^{12} + x^{13}$
- $P_{XB} = 1 + x + x^3 + x^4 + x^6 + x^7 + x^8 + x^{12} + x^{13}$

The L5 in-phase and the L5 quadrature channel share the same generating polynomials. The codes only differ in the selection of the initialization vector. Additionally the L5-Q signals carries a 20bit secondary code, while the L5-I signal carries a 10bit secondary code.

In order to generate the L5 signal with the VFCG, the shift register1 generating P_{XA} is shortened to 8190 chips using its 14 bit counter. The shift register2 generating P_{XB} is shortened to 10230 chips using its 14 bit counter.

The reload signal of the second shift register is also used for the first shift register since this generator must also be reloaded at 10230 chips. This is done by setting the *GPSL5En* bit of the *CodeGenUnitCtrl* register.

Note that in the case of GPS L5 the *StartOfEpoch* signal should be derived from LFSR-2 or the coupled primary memory (primary memory1) but not from LFSR-1, since the *StartOfEpoch* signal would then be generated every 8190 chips and not every 10230 chips as intended.

The following example gives an overview of how to configure the Code Generator Unit in order to generate GPS L5 Q code PRN-1 with the VFCG and the GPS L5 I code PRN-1 in the coupled primary memory. Also it is assumed that the LL-Q correlator correlates the data signal, while all other correlators correlate on the pilot signal. It is assumed that the *StartOfEpoch* signal is generated from the LFSR-2 of the VFCG. However it would also work if the *StartOfEpoch* signal is derived from the coupled primary memory. In that case the *StartOfEpochSel* bit would have to be set to "2" (primary code memory1), since the coupled primary memory is always addressed by primary memory1.

Register	MSB	Setting						LSB
CodeGenUnitCtrl (20bit)				1000	0000	0010	1101	101X
PrimCodeRam1Ctrl (28bit)		0000	0000	0000	0010	0111	1111	0101
PrimCodeRam2Ctrl (26bit)		00	0000	0000	0000	0000	0000	0000
SecCodeRam1Ctrl (28bit)		1001	1111	1101	0100	0000	0001	0011
SecCodeRam2Ctrl (28bit)		1001	1111	1101	0100	0000	0000	1001
VFCGExtTaps (28bit)		1011	0111	0001	1000	0000	0011	0110
VFCGInit (28bit)		0010	1111	0010	1011	1111	1110	1100
VFCGLength (28bit)		1001	1111	1101	0101	1111	1111	1101
DelayLineCtrl (26bit)		XX	XXXX	XXXX	XXXX	XXXX	XX00	00XX
CorrUnitCtrl (17bit)				X	XXXX	XXXX	XX10	0001

Table 3-14: Example settings for GPS L5 PRN-1 via VFCG and Primary Memory 1

Besides the above configuration the primary code RAM would have to be loaded with GPS L5 I PRN#1, the secondary RAM1 would have to be loaded with the 20bit Neuman Hoffman Code for the pilot and the secondary RAM2 would have to be loaded with the 10bit Neuman Hoffman Code for the data channel.

3.4.3.6.4 Galileo E1 Code Generation

The Galileo E1-B and E1-C code are memory based codes. Therefore the following example shows how the Code Generator Unit of a channel can be set up in order to generate E1-C with primary memory 1 and E1-B with primary memory 2.

Other than for shift register based codes, this example is valid for all PRNs of the Galileo E1-B/C family, since the PRN code number is only determined by the content of the memory, not by the content of the AGGA configuration register.

Register	MSB	Setting						LSB
CodeGenUnitCtrl (20bit)				0010	0011	0000	1111	010X
PrimCodeRam1Ctrl (28bit)		0000	0000	0000	0000	1111	1111	1011
PrimCodeRam2Ctrl (26bit)		0000	0000	0000	0000	1111	1111	1011
SecCodeRam1Ctrl (28bit)		0011	1111	1110	1100	0000	0001	1000
SecCodeRam2Ctrl (28bit)		0000	0000	0000	0000	0000	0000	0000
VFCGExtTaps (28bit)		0000	0000	0000	0000	0000	0000	0000
VFCGInit (28bit)		0000	0000	0000	0000	0000	0000	0000
VFCGLength (28bit)		0000	0000	0000	0000	0000	0000	0000
DelayLineCtrl (26bit)		XX	XXXX	XXXX	XXXX	XXXX	XX00	00XX
CorrUnitCtrl (17bit)				X	XXXX	XXXX	XX10	1001

Table 3-15: Example settings for Galileo E1 B+C via Primary Memory 1&2

3.4.3.6.5 Galileo E5 Code Generation

The E5A and E5B PRN codes are generated with two 14 stage LFSRs with the following polynomials (see also [RD-01]):

E5AI and E5AQ share the same polynomials:

- $P_{E5AI1/Q1} = 1 + x + x^6 + x^8 + x^{14}$
- $P_{E5AI2/Q2} = 1 + x^4 + x^5 + x^7 + x^8 + x^{12} + x^{14}$

E5BI:

- $P_{E5BI1} = 1 + x^4 + x^{11} + x^{13} + x^{14}$
- $P_{E5BI2} = 1 + x^2 + x^5 + x^8 + x^9 + x^{12} + x^{14}$

E5BQ:

- $P_{E5BQ1} = P_{E5BI1} = 1 + x^4 + x^{11} + x^{13} + x^{14}$
- $P_{E5BQ2} = 1 + x + x^5 + x^6 + x^9 + x^{10} + x^{14}$

All codes have a 10.23 MHz chip rate. Each code is truncated at a length of 10230 chips resulting in a 1 ms code epoch.

Table 3-16 gives an example how the Code Generator Unit can be configured in order to generate Galileo E5A-I and Galileo E5A-Q in one channel. The example uses the VFCG for E5A-Q and the coupled memory code RAM for E5A-I. The example shows the configuration for PRN-1.

Register	MSB	Setting						LSB
CodeGenUnitCtrl (20bit)				1000	0000	0000	1101	100X
PrimCodeRam1Ctrl (28bit)		0000	0000	0000	0010	0111	1111	0101
PrimCodeRam2Ctrl (26bit)		00	0000	0000	0000	0000	0000	0000
SecCodeRam1Ctrl (28bit)		1001	1111	1101	0100	0000	0110	0011
SecCodeRam2Ctrl (28bit)		1001	1111	1101	0100	0000	0001	0011
VFCGExtTaps (28bit)		0001	1011	0001	0110	0001	0100	0001
VFCGInit (28bit)		1010	0000	1100	1110	0000	1100	0000
VFCGLength (28bit)		1001	1111	1101	0110	0111	1111	0101
DelayLineCtrl (26bit)		XX	XXXX	XXXX	XXXX	XXXX	XX00	00XX
CorrUnitCtrl (17bit)				X	XXXX	XXXX	XX10	0001

Table 3-16: Example settings for Galileo E5A (I+Q) PRN-1 code via VFCG and Primary Memory

Table 3-17 gives an example how the Code Generator Unit can be configured in order to generate Galileo E5B-I and Galileo E5B-Q in one channel. The example uses the VFCG for E5B-Q and the coupled memory code RAM for E5B-I. The example shows the configuration for PRN-1.

Register	MSB	Setting						LSB
CodeGenUnitCtrl (20bit)				1000	0000	0000	1101	100X
PrimCodeRam1Ctrl (28bit)		0000	0000	0000	0010	0111	1111	0101
PrimCodeRam2Ctrl (26bit)		00	0000	0000	0000	0000	0000	0000
SecCodeRam1Ctrl (28bit)		1001	1111	1101	0100	0000	0110	0011
SecCodeRam2Ctrl (28bit)		1001	1111	1101	0100	0000	0000	0011
VFCGExtTaps (28bit)		1000	1100	1100	0100	0100	0000	1011
VFCGInit (28bit)		0001	0110	0110	1011	1100	0000	0110
VFCGLength (28bit)		1001	1111	1101	0110	0111	1111	0101
DelayLineCtrl (26bit)		XX	XXXX	XXXX	XXXX	XXXX	XX00	00XX
CorrUnitCtrl (17bit)				X	XXXX	XXXX	XX10	0001

Table 3-17: Example settings for Galileo E5B (I+Q) PRN-1 code via VFCG and Primary Memory

3.4.4 Code Delay Line Unit (CDLU)

The Code Delay Line Unit in each channel as anticipated in Figure 3-30 and shown in more detail in Figure 3-44 selects and controls the delay to be applied to the code correlators. It generates the Early Early, Early, Punctual, Late and Late Late (i.e. EE, E, P, L, LL) version of the selected code sequence within the channel. Two Code Delay Lines are implemented within each CDLU allowing the processing of pilot and data signals within one channel and the processing of time multiplexed signals (e.g. GPS L2C) within one channel.

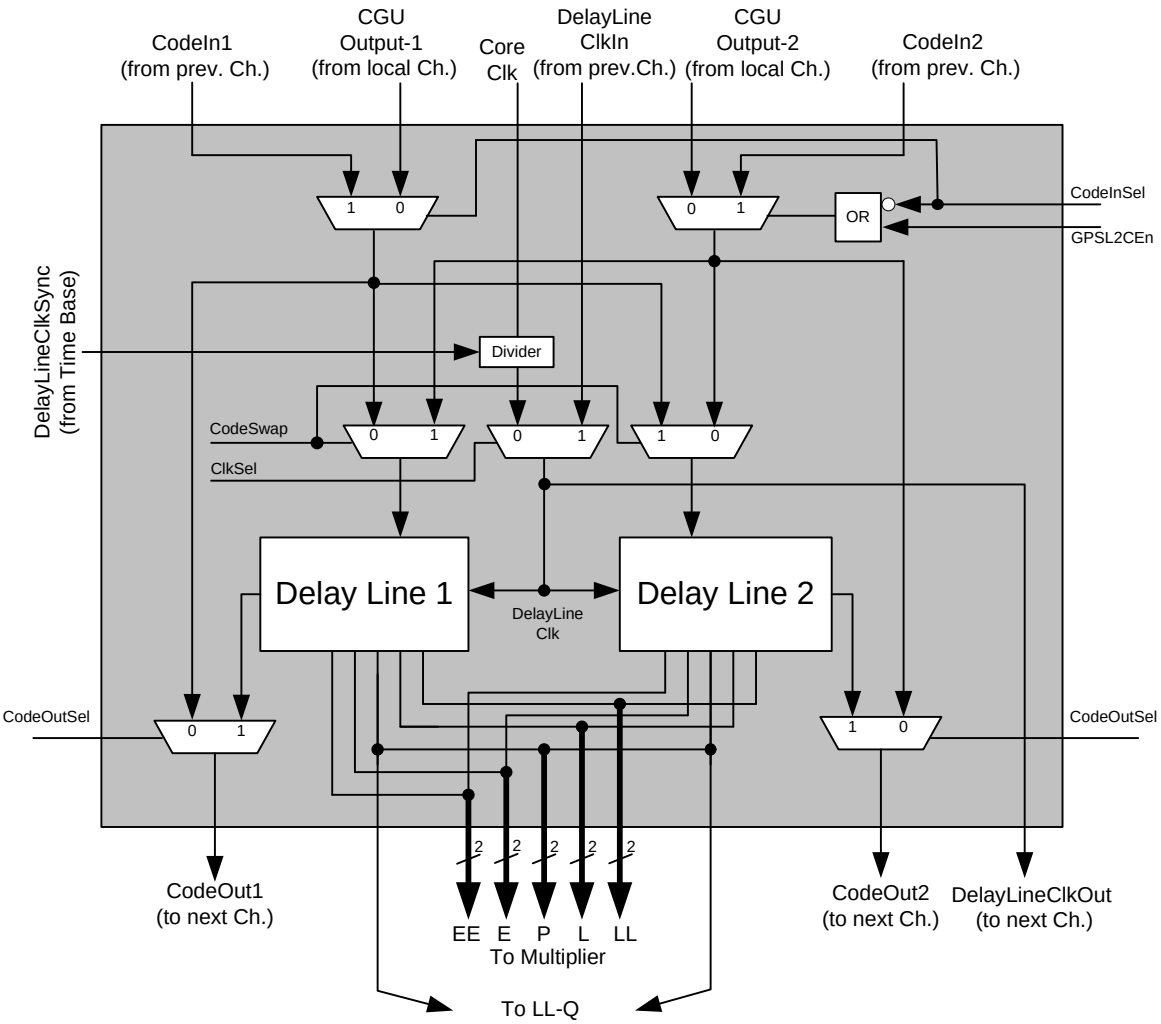


Figure 3-44: Code Delay Line Unit (CDLU)

3.4.4.1 Code Input Select

In the control register *DelayLineCtrl* there is a bit *CodeInSel*. If set, the code from the previous channel is throughput through the delay lines, otherwise the local generated code from the Code Generator Unit (CGU).

In the case of processing GPS-L2C signal (*GPSL2CEn* = '1' in *CodeGenUnitCtrl* register), the Code Delay Line 2 will always take the code generated in the local Channel, regardless of *CodeInSel*.

3.4.4.2 Code Swap

In the control register *DelayLineCtrl* there is a bit *CodeSwap*. If set, the input 1 is routed to Delay Line 2 and the input 2 is routed to Delay Line 1, otherwise input 1 is routed to Delay Line 1 and the input 2 is routed to Delay Line 2.

3.4.4.3 Delay Line Clock Select

In the control register *DelayLineCtrl* there is a bit *ClkSel*. If set, the *DelayLineClk* for the delay line is coming from the previous channel, otherwise the *DelayLineClk* is coming from the local channel. If the clock comes from the local channel, the DelayLineClk signal is computed as follow:

$$DelayLineClk = \frac{CoreClk}{DivRatio + 1}$$

where the *DivRatio* is 2 bit wide and can be found in the *DelayLineCtrl* register. Possible values of *DivRatio* are [0, 1, 2, 3].

In order to make sure that the delay line clock divider counter state is phase coherent over all channels, a synchronization signal *DelayLineClkSync* is generated every 12 *CoreClk* cycles in the Time Base module.

Note that this synchronization signal only can make sure that the Delay Line clock divider state of different channels is synchronous among the different channels. It can not prohibit the under sampling effect. The under sampling effect is present if a Delay Line Divider other than zero is used. Then the PRN code coming form the Code Generator is sampled with a slowed down clock (which can be asynchronous to the chip edges) and this causes that the PRN code edges can vary up to 3 *CoreClk* cycles behind the Delay Line compared to the true edge of the PRN code. This jitter is dependant of the programmed Delay Line Divider. For the following Delay Line Divider values the following jitter has to be taken into account:

Delay Line Divider (programmed value)	Jitter [CoreClk cycles]
0	0
1	1
2	2
3	3

Table 3-18: Delay Line Divider caused Jitter

3.4.4.4 Delay Line

Each Code Delay Line Unit consists of two identical Delay Lines. Each Delay Line is clocked by the *DelayLineClk* signal and consists of five 22 stage shift registers each fed with the selected code sequence. The phase delay between 2 adjacent shift register cells is referred to as one tap. The maximum length of 1 delay line is 110 taps. The delay between Early Early(EE), Early(E), Punctual(P), Late(L) and Late Late(LL) can be programmed individually in 22 steps i.e. 1, 2, 3, … 21, 22 taps. This can be done with the *Spacing* bit field in the *DelayLineCtrl* register.

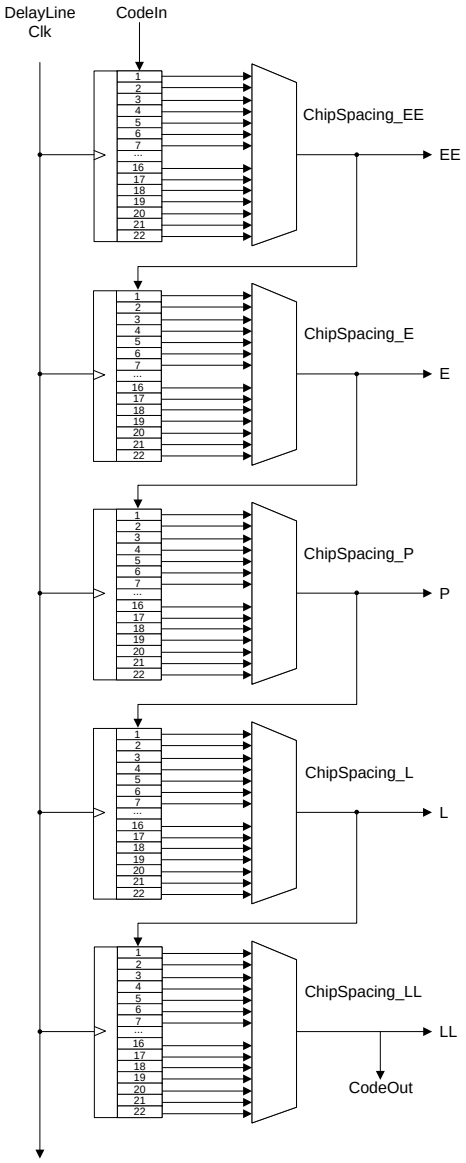


Figure 3-45: Delay Line

3.4.4.5 Code Output Select

In the control register *DelayLineCtrl* there is a bit called *CodeOutSel*. If set, the output from Delay Line 1 is routed to Code Out 1 of the Code Delay Line Unit and the output from Delay Line 2 is routed to Code Out 2 of the Code Delay Line Unit.

If the *CodeOutSel* bit is zero, there is a direct bypass from CodeIn1 (coming from the previous channel) to *CodeOut1* and from CodeIn2 (coming from the previous channel) to *CodeOut2*. This is used for hardware slaving of channels.

3.4.5 Correlator Unit

The Correlator Unit anticipated in Figure 3-30 and shown in more detail in Figure 3-46 consists of the Multiplier, the 29bit wide Integrator, the 32bit wide Data Collector, the 21bit Integration Counter and the 32bit Continuous Counter. The DMA transfer of the integration results is described in chapter 3.4.7.3. The Antenna Switch Controller (ASE) interaction with the Correlator Unit is described in more detail in chapter 3.6.3.

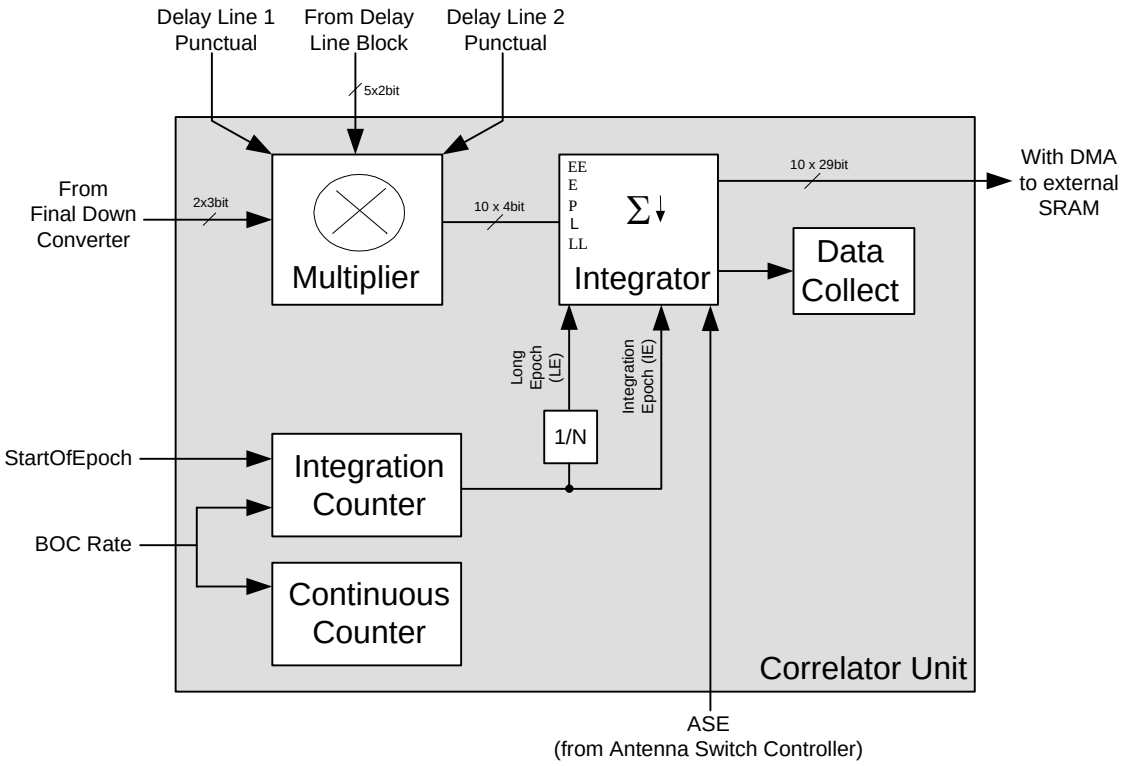


Figure 3-46: Correlator Unit

3.4.5.1 Multiplier

In the Multiplier the sequences coming from the Delay Line Block are multiplied with the signal from the Final Down Converter. Ten multiplications are processed in parallel since the I and Q part of the signal from the Final Down Converter are multiplied with each of the 5 (EE, E, P, L, LL) code sequences. The paths from the Delay Line Block to the Multiplier are 2bit wide. The MSB is always driven by the output of Delay Line 1. If *GPSL2CEn* = 0, the LSB is always 0, hence multiplication is by 1 or -1. If *GPSL2CEn* = 1, the LSB is connected to the output of Delay Line 2 as an enable / disable signal, hence multiplication is by -1, 1 or 0. With this information (enable/disable) the multiplier can also output "0". Therefore the multiplier output is 4bit wide, since the output can not only have the values [-7, -5, -3, -1, +1, +3, +5, +7] but also "0".

Please note that the multiplication scheme is slightly different for Antenna Switch Epoch enabled (*ASEEn* = 1 in *CorrUnitCtrl* register). This is shown in Figure 3-47 and described in detail in chapter 3.6.3.

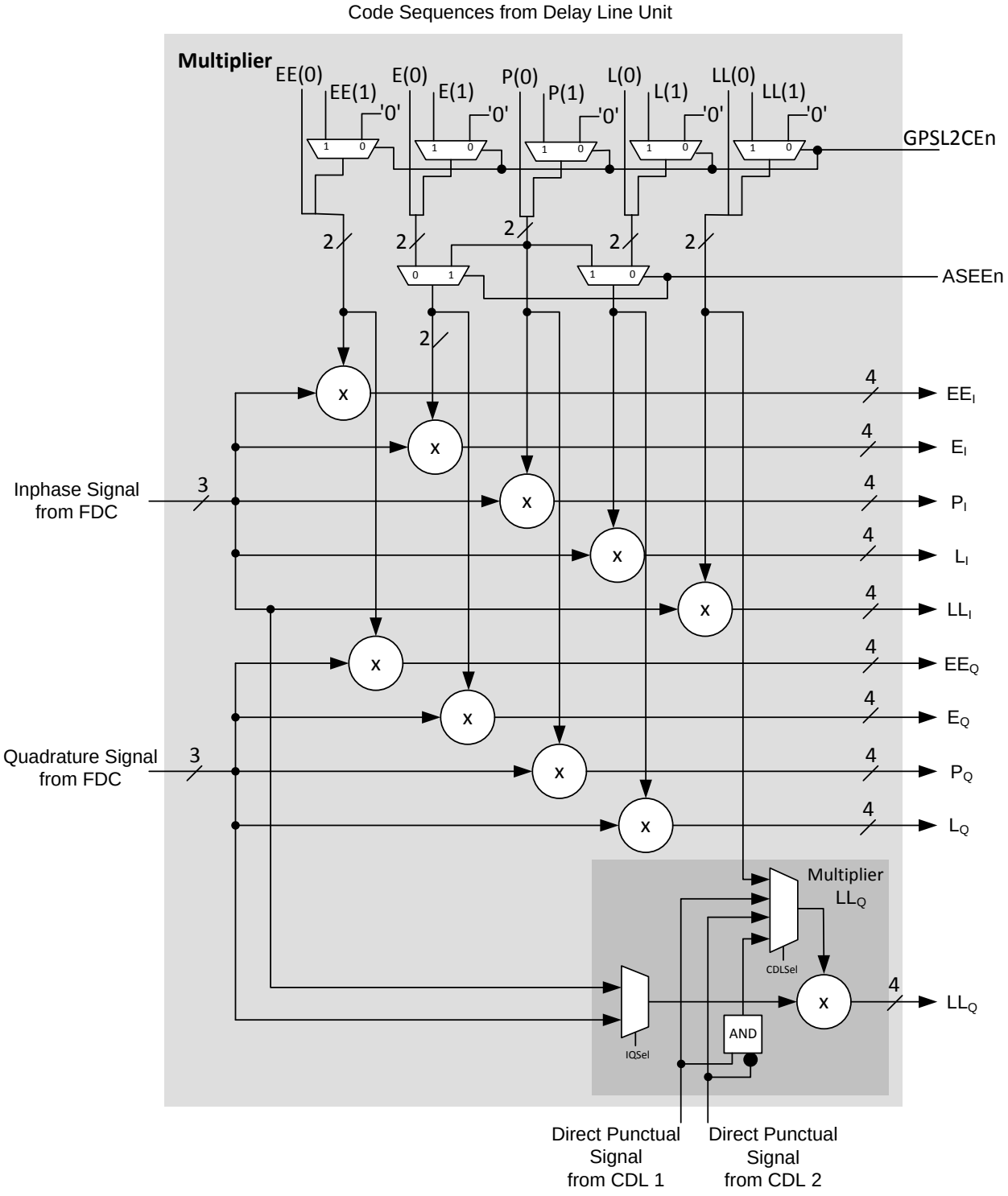


Figure 3-47: Multiplier

There is one special multiplier (LL-Q) which can be used for various functions. The following table gives an overview to these functions:

Purpose	IQSel	CDLSel	Multiplication of	with …	Comment
LL-Q normal operation	0	0	incoming Q signal	LL of DL 1	LL-Q is working like all other multipliers
B/C operation	1	2	incoming I signal	P of DL 2	LL-Q is demodulating the data of e.g. E1B
I/Q operation	0	2	incoming Q signal	P of DL 2	LL-Q is demodulating the data of e.g. E5A-I
L2C second half operation	1	3	incoming I signal	P of DL 1 AND inverse of DL2	LL-Q is demodulating e.g. L2CM

Table 3-19: Possibilities of LL-Q

3.4.5.2 Integration Counter

The Integration Counter shown in Figure 3-46 is 21bit wide and clocked with *BOCRate*. Once it elapses, the Integration Epoch occurs. The Integration Epoch can be phase aligned with the Code Epoch by resetting the Integration Counter with the Reset at Start of Epoch signal.

Note also that the Integration Epoch is automatically phase aligned to the punctual correlator with the help of a half delay line.

3.4.5.3 Continuous Counter

The Continuous Counter is clocked with *BOCRate* and is 32 bit wide. A 32bit offset can be added to the Continuous Counter by writing to the *ContCntOffset* register. The offset is added with the next Integration Epoch.

3.4.5.4 Long Epoch (LE)

If the Long Epoch Divider (i.e. 1/N in Figure 3-46) is programmed to zero then the Long Epoch is disabled and all correlators are working with Integration Epoch as usual. The accumulator values are latched at Integration Epoch, transferred via DMA and are reset back to zero.

The Long Epoch Divider can be programmed by writing the *LongEpochDiv* field in the *IntCountCtrl* register.

If only the Long Epoch Divider is programmed to any other value than zero, then all accumulators are still latched with the Integration Epoch, transferred via DMA but the accumulators are not reset back to zero. They are reset back to zero with the next Long Epoch.

If the bit *LongEpochLLQ* of *CorrUnitCtrl* register is set in addition to the Long Epoch Divider other than zero, then the LL-Q accumulator is reset back to zero with the Integration Epoch, although the Long Epoch is active. All other accumulators are reset back whenever the Long Epoch occurs.

This allows independent integration times for the pilot and the data component. Example: Taking the Galileo E1B+C Signal, the data part carries a message @ 250sps. The LL-Q multiplier would process the data part, all other multipliers would process the pilot part. Since a navigation message bit has 4ms width, the Integration Epoch has to be set to values equal or smaller than 4ms in order to catch all symbols correctly. However for the pilot part it might be of interest to have longer integration times than 4ms. Therefore the Long Epoch Divider could for example be set to ten. In this case all accumulator values (besides LL-Q) are integrating over 40ms until the accumulator value gets reset back to zero.

In order to know when a Long Epoch has occurred, the firmware can read the *LongEpochReady* status bit in the *CorrUnitCtrl* register. If the bit is set the firmware can know that the IE which just occurred was also a LE. The bit is on for one IE, then switched off and switched on at the next LE again and so on.

By setting the bit *IntSourceSel* bit in the *CorrUnitCtrl* register to "1", the DMA and GIC are not invoked at every IE, but at every LE. This allows invoking the firmware at the lower LE rate, while collecting autonomously the Navigation Data Bits via the Data Collect function (see also 3.4.5.6). The mechanism of the Interrupt Source Selection is depicted in Figure 3-48.

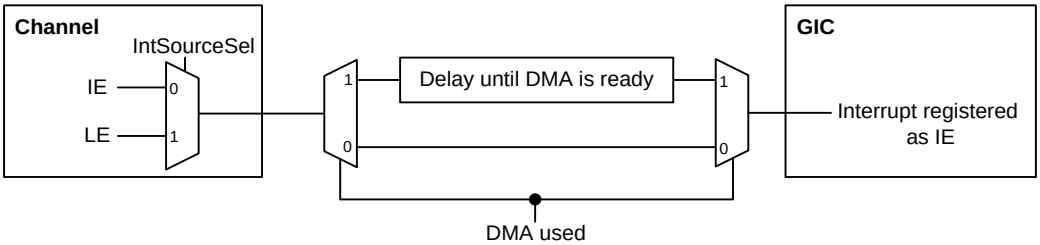


Figure 3-48: Interrupt Source (IE or LE) Selection

3.4.5.5 Integrator

The Integrator (shown in Figure 3-46) has ten 29bit signed accumulation registers for all 10 correlators. An improvement compared to AGGA-2/3 is that the sign bit (bit 28 when counting is started with 0) is also copied to bit 29-31 by the AGGA-4. Therefore the software does not have to do a sign extension at a high rate after fetching the correlation values.

The accumulators can be latched and reset at the following time instances:

- Integration Epoch (IE)
- Long Epoch (LE)
- Antenna Switch Epoch (ASE)

Table 3-20 gives a detailed overview how the accumulation registers are working under all conditions.

Latched in this context means that a snapshot of the current accumulation values is taken at a certain time instance (IE; LE, ASE) and is written into the Integration Epoch Observables (i.e. *IE_ValueEE_I, IE_ValueE_I,…, IE_ValueLL_Q)*.

Reset in this context means that the accumulators are reset back to zero.

The character "+" in this context means a logical OR.

				reset @						latch @					
ASE_En	LongEpochLLQ	LongEpochDiv = 0	LongEpochDiv != 0	EE (I/Q)	E (I/Q)	P (I/Q)	L (I/Q)	LL (I)	LL(Q)	EE (I/Q)	E (I/Q)	P (I/Q)	L (I/Q)	LL (I)	LL(Q)
0	0	true	false	IE	IE	IE	IE	IE	IE	IE	IE	IE	IE	IE	IE
0	0	false	true	LE	LE	LE	LE	LE	LE	IE	IE	IE	IE	IE	IE
0	1	true	false	IE	IE	IE	IE	IE	IE	IE	IE	IE	IE	IE	IE
0	1	false	true	LE	LE	LE	LE	LE	IE	IE	IE	IE	IE	IE	IE
1	0	true	false	IE	IE+ASE	IE+ASE	ASE	IE	IE	IE	ASE	IE	ASE	IE	IE
1	0	false	true	LE	LE+ASE	LE+ASE	ASE	LE	LE	IE	ASE	IE	ASE	IE	IE
1	1	true	false	IE	IE+ASE	IE+ASE	ASE	IE	IE	IE	ASE	IE	ASE	IE	IE
1	1	false	true	LE	LE+ASE	LE+ASE	ASE	LE	IE	IE	ASE	IE	ASE	IE	IE

Table 3-20: Action Table for Integrator

3.4.5.6 Data Collect

The sign of the LL-Q accumulator is entering a 32bit FIFO with every Integration Epoch. The FIFO is shifted left wise. The user can adjust a collecting length from 1..32 bits depending on the programming of the *DataCollectLength* field of the *CorrUnitCtrl* register. Whenever the data collect length counter elapses, the 32 bit FIFO is copied to the register *DataCollect*. Due to the left shift, the latest bit is aligned on the right. The software gets informed about an update of the *DataCollect* register via the flag *DataCollectReady* in the *CorrUnitCtrl* register. Once the software reads the *DataCollect* register, the *DataCollectReady* flag is cleared by the hardware. The IMT value of the corresponding Integration Epoch is also the IMT time stamp for the latest bit.

By setting the bit *ResetDataCollect* in the *CorrUnitCtrl* register, the current state of the data collect length counter as well as the 32bit FIFO is cleared with the next Integration Epoch.

3.4.6 Aiding Unit

The Aiding Unit can be used for various purposes. If the code and carrier NCOs are updated at a low rate due to long integration epochs an autonomous update of the loop frequency setting according to expected carrier and code dynamics can be performed by the aiding unit. The Aiding Unit can also be coupled together with an external Inertial Measurement Unit (IMU). Then the AGGA-4 pin *AU_TRIGGER* can be used as timing signal from the IMU system to the AGGA-4 Aiding Unit in order to coordinate the update of the Aiding Unit coefficients together with the external IMU. In case the aiding unit is updated with the *AU_TRIGGER* pin, the time instance of AU Trigger is latched into the registers *AUT_IMT_MSW* and *AUT_IMT_LSW*. As the signal dynamic influences the carrier frequency and the code frequency to a different extent, two autonomous aiding modules are implemented.

The aiding unit shown in Figure 3-49 consists of an autonomous carrier aiding module, an autonomous code aiding module and a clock generation module.

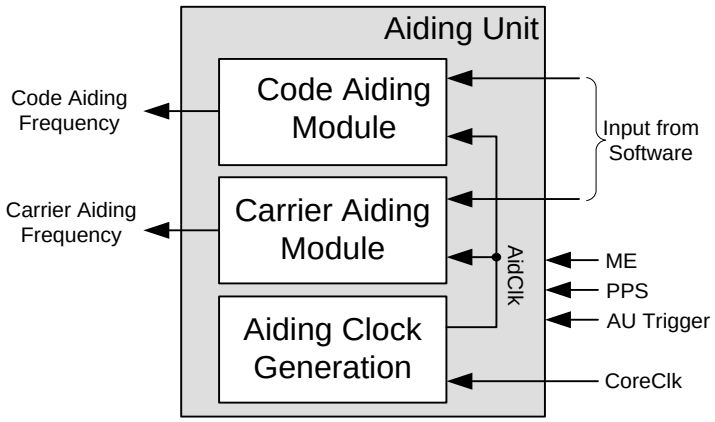


Figure 3-49: Aiding Unit

3.4.6.1 Code Aiding Unit

The code aiding unit structure is shown in Figure 3-50 with the following registers:

- *CodeNCOFreq* is the actual code NCO frequency
- *CodeSwFreq* is the code frequency provided by the acquisition or tracking loop programmed in software (SW)
- *CodeAidFreq* is the code aiding frequency
- *CodeAidAcc* is the code aiding acceleration

Note: Please also see chapter 3.4.3.5 to get the relation between the Code Aiding Unit and the *CodeSwShift* function.

In a first step (this is indicated with the "1" next to the addition circle in Figure 3-50) with every *AidClk*, the *CodeAidAcc* value is accumulated in the *CodeAidFreq* register according to Figure 3-50. As soon as this is done, in a second step (this is indicated with the "2" next to the addition circle in Figure 3-50) the content of *CodeAidFreq* and *CodeSwFreq* is added together and written to the register *CodeNCOFreq* as a second step.

By programming of the *CodeAidEn* bit in the *AidingUnitCtrl* register the Code Aiding Unit can be switched on and off, as shown in Figure 3-50. The switch command becomes effective with the next Integration Epoch.

Note: New parameters written into the registers *CodeAidAcc* and *CodeAidFreq* become effective either with the next Measurement Epoch (ME), Pulse Per Second (PPS), or rising edge of the *AU_TRIGGER*, depending on the programming of the *TriggerSel* bit in the *AidingUnitCtrl* register. Note that the external *AU_TRIGGER* signal has to last at least 4 *CoreClk* cycles in order to be correctly detected.

Also note that reading back the registers *CodeAidAcc* and *CodeAidFreq* give not back the value just written into these registers but the actual effective value. That means if one writes a value into the register, the system waits for the next applicable trigger event (either ME, PPS or *AU_TRIGGER*). Then the value becomes effective and then the read-back value equals the value written before.

A new parameter written into the register *CodeSwFreq* becomes effective either immediately or with the next Integration Epoch, depending on the programming of the *CodeSwFreqMode* bit in the *NCOSettings* register.

If new parameters have been written to *CodeAidFreq* and/or to *CodeAidAcc*, then *CodeAidFreq* without the addition of *CodeAidAcc* is added to *CodeNCOFreq* with the next trigger event. From there on a combination of *CodeAidAcc* and *CodeAidFreq* (according to Figure 3-50) is added to *CodeNCOFreq* with every *AidClk*.

Note: It takes 2 *CoreClk* cycles after the trigger until the new value (*CodeSwFreq* or *CodeAidFreq*) is added (applicable) in the NCO phase. Example: The SW changes the value *CodeSwFreq* from 10 to 20 and the trigger is set to IE. The NCO phase accumulator would do …10+10+10 IE 10+10+20+20+20+…

The following table gives the registers, which belong to the Code Aiding Unit:

Register Name	Register Length	Accessible Bits	Comment
CodeAidAcc	17 bit	17 bit (R/W)	
CodeAidFreq	31 bit	25 MSB (R/W)	6 LSB cleared at write access to the MSB
CodeNCOFreq	32 bit	32 bit (R)	
CodeSwFreq	32 bit	32 bit (R/W)	

Table 3-21: Registers of the Code Aiding Unit

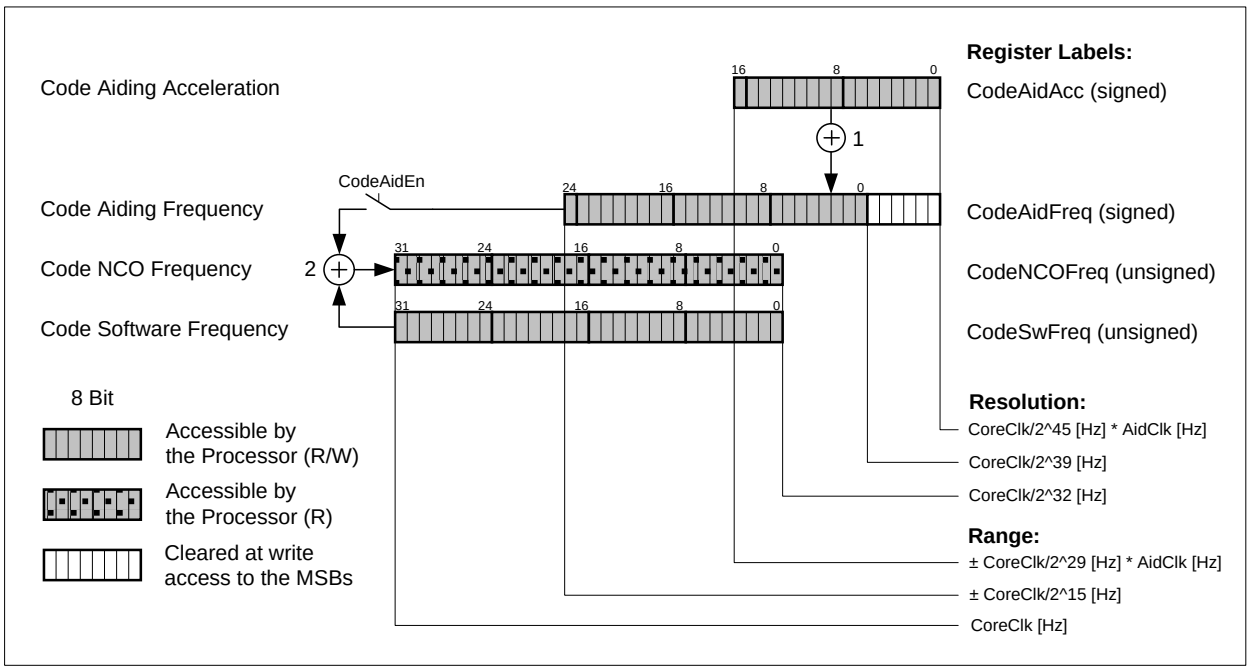


Figure 3-50: Code Aiding Unit

The Code Aiding Unit as depicted in Figure 3-50 can cope with the following Doppler's and Accelerations:

Ranges of Code Aiding Unit @ 100Hz AidClk				
Core Clock [MHz]		**2,5**	**25**	**40**
max Doppler [Hz]	+/-	76,29	762,94	1220,70
max Acceleration [Hz/s]	+/-	0,47	4,66	7,45

Table 3-22: Code Aiding Unit Ranges

The following formulas have been used to calculate the numbers in Table 3-22:

$$Max\ Doppler\ [Hz] = \pm \frac{CoreClk}{2^{15}}$$

$$Max\ Acceleration\left[\frac{Hz}{s}\right] = \pm \frac{CoreClk}{2^{29}} \cdot T_{AidClk}$$

The resolution is chosen such that the maximum range error due to finite Code Aiding Unit resolution is the following:

Error of Code Aiding Unit @ 1kHz AidClk after 100ms due to finite resolution				
Core Clock [MHz]		**2,5**	**25**	**40**
Range Error [mm] due to CodeAidFreq with 1MChip Code	+/-	0,13	1,33	2,13
Range Error [mm] due to CodeAidFreq with 10MChip Code	+/-	0,01	0,13	0,21
Range Error [mm] due to CodeAidAcc with 1MChip Code	+/-	0,10	1,04	1,67
Range Error [mm] due to CodeAidAcc with 10MChip Code	+/-	0,01	0,10	0,17

Table 3-23: Code Aiding Unit Errors due to finite resolution

The following formulas have been used to calculate the numbers of Table 3-23:

$$CodePhaseError\ [mm]\ due\ to\ CodeAidFreq = \frac{CoreClk}{2^{39}} \cdot \Delta t_{update} \cdot \frac{SpeedOfLight}{CodeFreq} \cdot 1e3$$

$$CodePhaseError\ [mm]\ due\ to\ CodeAidAcc = \frac{CoreClk}{2^{45} \cdot T_{AidClk}} \cdot \frac{1}{2} \cdot \Delta t_{update}^{2} \cdot \frac{SpeedOfLight}{CodeFreq} \cdot 1e3$$

Where Δt_{update} is the update rate at which new aiding values are written by the software. The factor 1e3 is there to bring the result from meter to millimeter.

3.4.6.2 Carrier Aiding Unit

The carrier aiding unit structure is shown in Figure 3-51 with the following registers:

- *CarrNCOFreq* is the actual carrier NCO frequency
- *CarrSwFreq* is the carrier frequency provided by the acquisition or tracking loop programmed in software
- *CarrAidFreq* is the carrier aiding frequency
- *CarrAidAcc* is the carrier aiding acceleration

Note: Please also see chapter 3.4.2 to get the relation between the Carrier Aiding Unit and the *CarrSwShift* function.

In a first step (this is indicated with the "1" next to the addition circle in Figure 3-51) with every *AidClk*, the *CarrAidAcc* value is accumulated in the *CarrAidFreq* register according to Figure 3-51. As soon as this is done, in a second step (this is indicated with the "2" next to the addition circle in Figure 3-51) the content of *CarrAidFreq* and *CarrSwFreq* is added together and written to the register *CarrNCOFreq* as a second step.

By programming of the *CarrAidEn* bit in the *AidingUnitCtrl* register the Carrier Aiding Unit can be switched on and off as shown in Figure 3-51. The switch command becomes effective with the next Integration Epoch.

Note: New parameters written into the registers *CarrAidAcc* and *CarrAidFreq* become effective either with the next Measurement Epoch (ME), Pulse Per Second (PPS), or rising edge of the *AU_TRIGGER* pin, depending on the programming of the *TriggerSel* bit in the *AidingUnitCtrl* register. Note that the external *AU_TRIGGER* signal has to last at least 4 *CoreClk* cycles in order to be correctly detected.

Also note that reading back the registers *CarrAidAcc* and *CarrAidFreq* does not give back the value just written into these registers, but the actual effective value. That means if one writes a value into the register, the system waits for the next applicable trigger event (either ME, PPS or *AU_TRIGGER*). Then the value becomes effective and then the read-back value equals the value written before.

A new parameter written into the register *CarrSwFreq* becomes effective either immediately or with the next Integration Epoch, depending on the programming of the *CarrSwFreqMode* bit in the *NCOSettings* register.

If new parameters have been written to *CarrAidFreq* and/or to *CarrAidAcc*, then *CarrAidFreq* without the addition of *CarrAidAcc* is added to *CarrNCOFreq* with the next trigger event. As shown in Figure 3-51 from there on a combination of *CarrAidAcc* and *CarrAidFreq* is added to *CarrNCOFreq* with every *AidClk*.

Note: It takes 2 *CoreClk* cycles after the trigger until the new value (*CarrSwFreq* or *CarrAidFreq*) is added (applicable) in the NCO phase. Example: The SW changes the value *CarrSwFreq* from 10 to 20 and the trigger is set to IE. The NCO phase accumulator would do …10+10+10 IE 10+10+20+20+20+…

The following table gives the registers, which belong to the Carrier Aiding Unit:

Register Name	Register Length	Accessible Bits	Comment
CarrAidAcc	24 bit	24 bit (R/W)	
CarrAidFreq	38 bit	32 MSB (R/W)	6 LSB cleared at write access to the MSB
CarrNCOFreq	32 bit	32 bit (R)	
CarrSwFreq	32 bit	32 bit (R/W)	

Table 3-24: Registers of the Carrier Aiding Unit

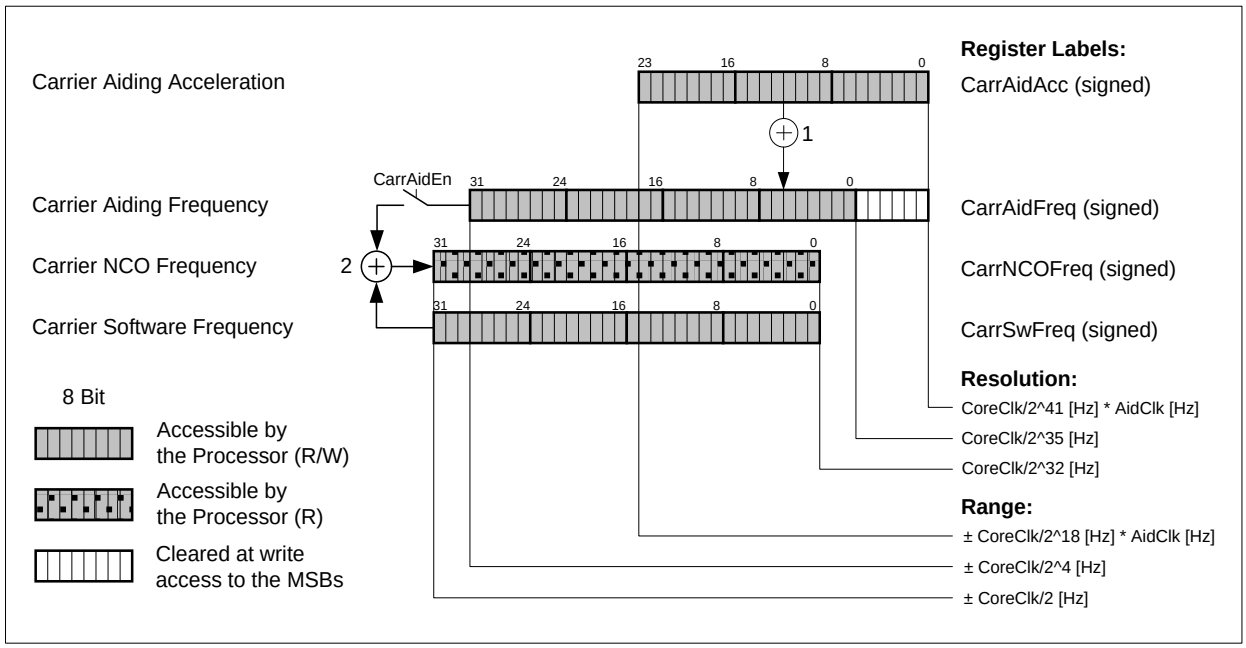


Figure 3-51: Carrier Aiding Unit

The Carrier Aiding Unit as depicted in Figure 3-51 can cope with the following Doppler's and Accelerations:

Ranges of Carrier Aiding Unit @ 100Hz AidClk				
Core Clock [MHz]		**2,5**	**25**	**40**
max Doppler [MHz]	+/-	0,16	1,56	2,50
max Acceleration [Hz/s]	+/-	953,67	9536,74	15258,79

Table 3-25: Carrier Aiding Unit Ranges

The following formulas have been used to calculate the numbers in Table 3-25:

$$Max\ Doppler\ [Hz] = \pm\frac{CoreClk}{2^4}$$

$$Max\ Acceleration\left[\frac{Hz}{s}\right] = \pm\frac{CoreClk}{2^{18}}\cdot T_{AidClk}$$

The resolution is chosen such that the maximum range error due to finite Carrier Aiding Unit resolution is the following:

Error of Carrier Aiding Unit @ 1kHz AidClk after 100ms due to finite resolution				
Core Clock [MHz]		**2,5**	**25**	**40**
Phase Error [um] due to *CarrAidFreq* @L1	+/-	1,38	13,85	22,15
Phase Error [um] due to *CarrAidFreq* @L5	+/-	1,85	18,54	29,67
Phase Error [um] due to *CarrAidAcc* @L1	+/-	1,08	10,82	17,31
Phase Error [um] due to *CarrAidAcc* @L5	+/-	1,45	14,49	23,18

Table 3-26: Carrier Aiding Unit Errors due to finite resolution

The following formulas have been used to calculate the numbers Table 3-26:

$$CarrierPhaseError\ [\mu m]\ due\ to\ CarrAidFreq = \frac{CoreClk}{2^{35}}\cdot\Delta t_{update}\cdot\frac{SpeedOfLight}{CarrierFreq}\cdot 1e6$$

$$CarrierPhaseError\ [\mu m]\ due\ to\ CarrAidAcc = \frac{CoreClk}{2^{41}\cdot T_{AidClk}}\cdot\frac{1}{2}\cdot\Delta t_{update}^2\cdot\frac{SpeedOfLight}{CarrierFreq}\cdot 1e6$$

Note: CarrierFreq in the two equations above are the physical carrier frequency (e.g. 1.57542 GHz for L1) and not the value of the register *CarrAidFreq*.

Where Δt_{update} is the update rate at which new aiding values are written by the software. The factor 1e6 is there to bring the result from meter to micrometer.

3.4.6.3 Aiding Clock Generation

The Aiding Unit clock (*AidClk*) is derived by dividing the *CoreClk* by a 19bit divider. It can be programmed via the *DivRatio* field in the *AidingUnitCtrl* register. New written settings become effective immediately. In order to synchronize the *AidClk* with either the Measurement Epoch (ME), the Pulse Per second (PPS) or AU_TRIGGER, the internal *AidClk* divider reset is armed whenever the software writes to one of the following registers: *CodeAidFreq*, *CodeAidAcc*, *CarrAidFreq*, *CarrAidAcc*. Once the reset is armed it will reset at the next trigger event (ME, PPS or AU_TRIGGER). The timing event to which the *AidClk* shall be synchronized to, can be selected by the *TriggerSel* field in the *AidingUnitCtrl* register.

3.4.6.4 Behaviour Examples

The following two figures shall give some examples in order to fully understand the behaviour of the Aiding Unit. The figures use the Code Aiding Unit as an example. However the behaviour is exactly the same for the Carrier Aiding Unit.

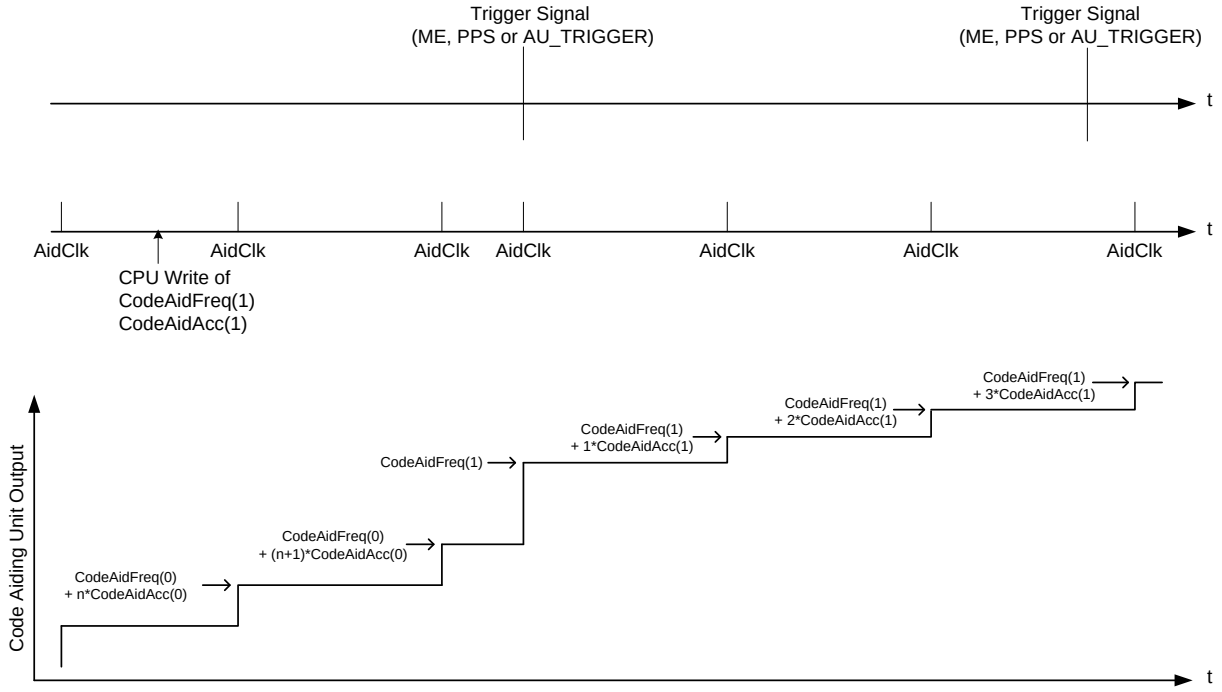


Figure 3-52: Aiding Unit Behaviour Example 1

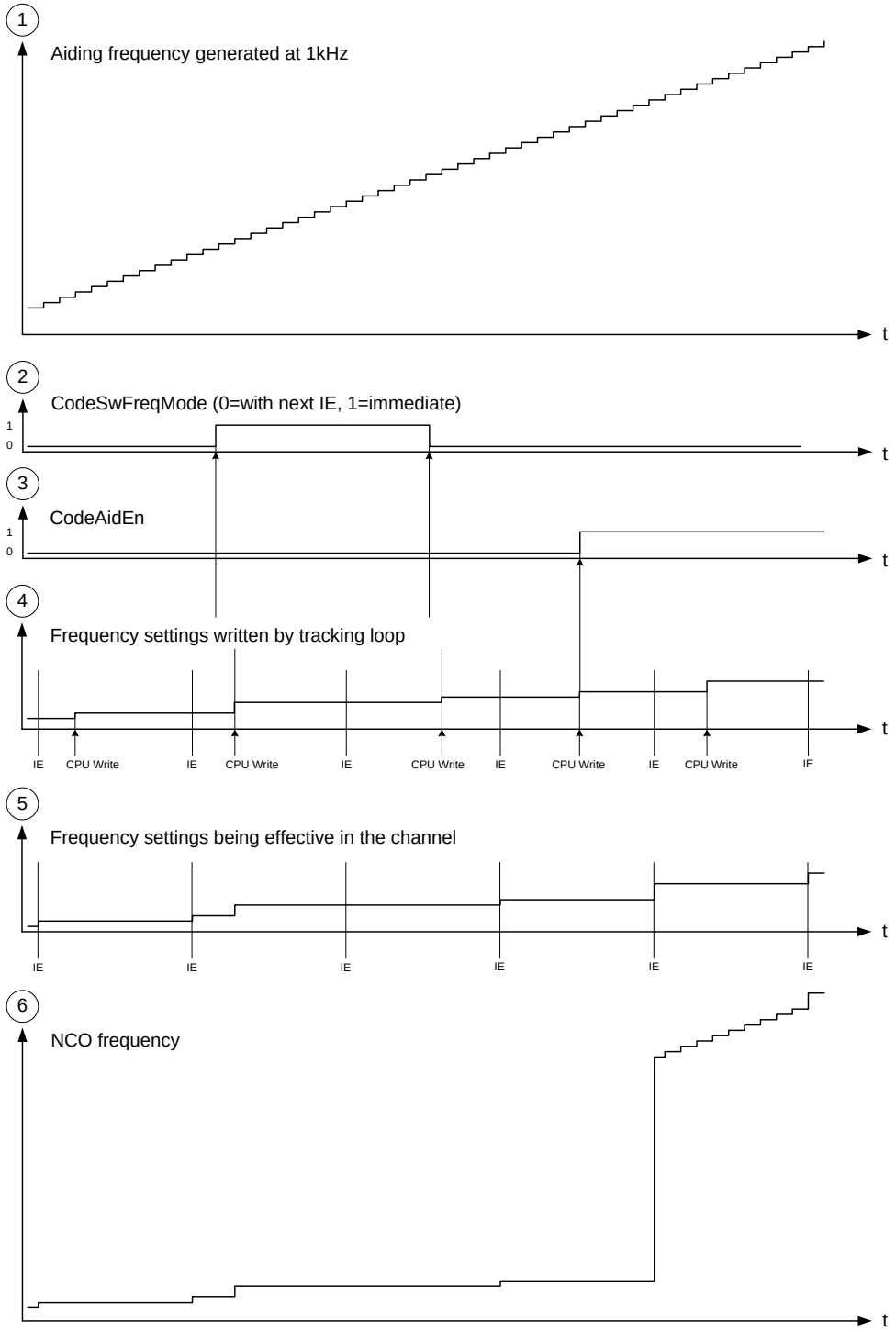


Figure 3-53: Aiding Unit Behaviour Example 2

3.4.7 Observables

The channel observables consist of the Integration Epoch (IE) observables and the Measurement Epoch (ME) observables. The Integration Epoch observables mainly consist of the correlation values and the environmental settings which caused these correlation values (e.g. NCO settings) used for the acquisition and tracking loops. The Measurement Epoch observables are mainly used to get channel measurements (code/carrier phase) which are taken at the same time instance for all channels.

3.4.7.1 Integration Epoch Observables

Register Name U = Unsigned S = Signed	Description
LoopState (U)	This register is writable and can therefore be used by the software to write data in it, which is then transferred (together with the other observables) via DMA into the external RAM. This allows keeping the context (e.g. loop settings) together with the correlation values.
IE_IMT_LSW (U)	The lower 32bits of the Instrument Measurement Time (IMT) Counter at the time instance of Integration Epoch.
IE_ValueEE_I (S)	The Early Early Inphase correlation value. See also Note-1 below this table.
IE_ValueEE_Q (S)	The Early Early Quadrature correlation value. See also Note-1 below this table.
IE_ValueE_I (S)	The Early Inphase correlation value. See also Note-1 and Note-2 below this table.
IE_ValueE_Q (S)	The Early Quadrature correlation value. See also Note-1 and Note-2 below this table.
IE_ValueP_I (S)	The Punctual Inphase correlation value. See also Note-1 and Note-2 below this table.
IE_ValueP_Q (S)	The Punctual Quadrature correlation value. See also Note-1 and Note-2 below this table.
IE_ValueL_I (S)	The Late Inphase correlation value. Note: See also Note-1 and Note-2 below this table.
IE_ValueL_Q (S)	The Late Quadrature correlation value. See also Note-1 and Note-2 below this table.
IE_ValueLL_I (S)	The Late Late Inphase correlation value. See also Note-1 below this table.
IE_ValueLL_Q (S)	The Late Late Quadrature correlation value. See also Note-1 below this table.
DataCollect (U)	This register contains the data bits which are collected with every Integration Epoch.
IE_CodeFreq (U)	If the *CodeFreqSel* bit in the *CorrUnitCtrl* register is one, the observable will reflect only the Code NCO increment caused by the programming of *CodeSwFreq* (see chapter 3.4.6.1), not including the aiding component caused by *CodeAidFreq*. If the *CodeFreqSel* bit in the *CorrUnitCtrl* register is zero, the observable will reflect the Code NCO increment caused by the summation of *CodeSwFreq* and the aiding component *CodeAidFreq*.
IE_CarrFreq (S)	If the *CarrFreqSel* bit in the *CorrUnitCtrl* register is one, the observable will reflect only the Carrier NCO increment caused by the programming of *CarrSwFreq* (see chapter 3.4.6.2), not including the aiding component caused by *CarrAidFreq*. If the *CarrFreqSel* bit in the *CorrUnitCtrl* register is zero, the observable will reflect the Carrier NCO increment caused by the summation of *CarrSwFreq* and the aiding component *CarrAidFreq*.
IE_CarrObsPhase (S)	By default this register contains the 12MSB's of the carrier phase and the 20 bit cycle count (see chapter 3.4.2). If the bit *IE_CarrObsSel* in the *CorrUnitCtrl* register is set, the register will contain the whole 32bit of the carrier phase. This allows software slaving of channels, since the NCO's of the channels can be phase aligned by software.
IE_ContCount (U)	This register contains the 32bit Continuous Count latched at Integration Epoch.
IE_CodePhase (U)	This register contains the 32 bit code NCO phase latched at Integration Epoch.

Table 3-27: Integration Epoch (IE) Observables

Note-1: The sign bit of the correlation values is copied in the upper bits. Therefore the software does not need do no further shifting of the correlation values in order to correctly reconstruct the two's complement values.

Note-2: If Antenna Switch Mode is enabled, this observable has a special function (see chapter 3.6.3).

3.4.7.2 Measurement Epoch Observables

Register Name U = Unsigned S= Signed	Description
ME_IMT_LSW (U)	The time stamp of the lower 32bits of the Instrument Measurement Time (IMT) count at the time instance "Measurement Epoch". Note that this time stamp is typically identical with the time stamp ME_IMT_LSW hosted in the Channel Matrix. However if channels are slaved, there is one *CoreClk* cycle delay between the channels for everything. Then also the ME_IMT_LSW time stamps in the slaved channels will differ for this delay.
ME_CarrObsPhase (S)	By default this register contains the 12MSB's of the carrier phase and the 20 bit cycle count (see chapter 3.4.2). If the bit *ME_CarrObsSel* in the *CorrUnitCtrl* register is set, the register will contain the whole 32bit of the carrier phase. This allows software slaving of channels, since the NCO's of the channels can be phase aligned by software.
ME_IntCount (U)	This register contains the value of the 21bit Integration Count at the time instance "Measurement Epoch".
ME_ContCount (U)	This register contains the value of the 32bit Continuous Count at the time instance "Measurement Epoch".
ME_CodePhase (U)	This register contains the 32bit code NCO phase at the time instance "Measurement Epoch".

Table 3-28: Measurement Epoch (ME) Observables

3.4.7.3 DMA Transfer of GNSS Observables

All channel observables can be transferred via DMA to any destination address. These observables are referred to as "Raw Samples" in some applications. The *GNSS_DMACtrl* register allows to select which observable shall be transmitted via DMA and which not. Each bit represents one observable. The exact matching can be found in the address table in chapter 7.4.5.47. In order to set up a DMA operation, the following order is recommended:

- Clear *GNSS_DMACtrl* register in order to be sure that the DMA is not working
- Configure the *GNSS_DMAStartAddr*
- Configure the *GNSS_DMACurAddr* (typically set to the same value as *GNSS_DMAStartAddr*)
- Configure the *GNSS_DMAEndAddr*
- Configure the *GNSS_DMACtrl* register in order to enable the observables transfer

Note that the GNSS DMA area must be seen as a circular buffer. Once the GNSS DMA transfer reaches the end address of the buffer it wraps around and starts again from the start address. Therefore by setting the *GNSS_DMAStartAddr* and the *GNSS_DMAEndAddr* only the borders of the circular buffer are defined. The address in the defined buffer where the DMA shall start to write its observables to is defined by writing to the *GNSS_DMACurAddr*. Note that typically a DMA current address pointer is read only. In this case (GNSS DMA) it is also writeable. Therefore the DMA transfer can be started at any address which is within the buffer. The DMA transfer is always 32bit wide. By reading the *GNSS_DMACurAddr* register it returns the address of the next free 32bit data element.

If DMA is enabled, the Integration Epoch strobe is not directly routed from the GNSS core to the Interrupt Controller, but is delayed until the DMA transfer is finished to make sure that the DMA is ready if the software gets invoked via interrupt. Note that only the notification to the software is delayed, not the Integration Epoch itself. The Integration Epoch observables contain of course the data belonging to the time instance "Integration Epoch".

Also note that the DMA is intended to transfer Integration Epoch Observables. Therefore the GNSS DMA is triggered solely by the Integration Epoch. However since the Measurement Epoch Observables are just behind the Integration Epoch Observables (from an address point of view) it is also possible to transfer the Measurement Epoch Observables by means of DMA at the Integration Epoch time instance.

The timing of the DMA process is as follow:

If any bit in the *GNSS_DMACtrl* register is switched on, it takes 11 *CoreClk* cycles to intitialize the DMA. Then the hardware steps through the *GNSS_DMACtrl* register. It starts with Bit0 (IntLoopState) and it takes one *CoreClk* cycle to check whether a DMA transfer of this observable is requested or not. If it is requested it takes 5 *CoreClk* Cycles to transfer each observable into the DMA FIFO of depth 32. If it is not requested it will step to the next bit (*IE_MT_LSW*) and decide once more (one *CoreClk* cycle) if this observable has to be transferred or not.

Once there is a value in the FIFO, the AMBA bus is requested and it takes minimum 9 *SysClk* cycles until the observable arrives at the destination (e.g. external RAM). Note that the AMBA bus can be blocked by instances with higher priority than the DMA. In this case the 9 *SysClk* cycles are exceeded. After the DMA transfer of a particular channel is finished it takes 3 *CoreClk* cycles to end the DMA process.

The DMA timing behaviour is depicted in Figure 3-54.

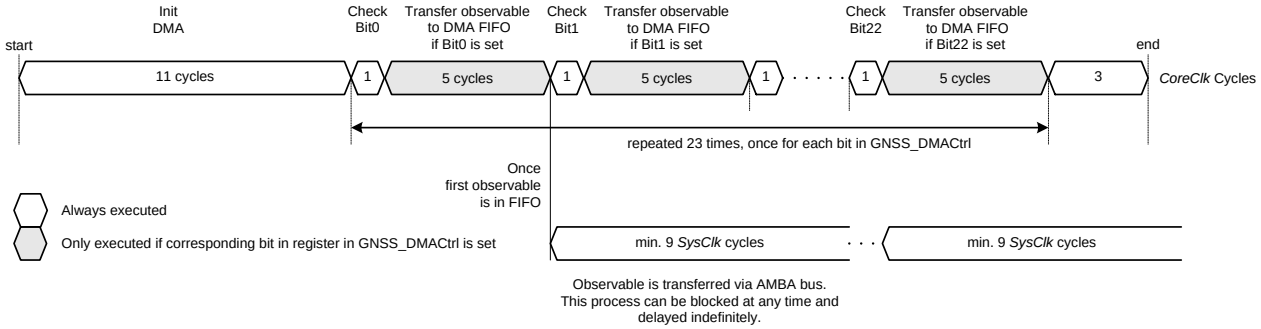


Figure 3-54: DMA Timing Behaviour

Example: In order to transfer all IE correlator values together with the time stamp via DMA in a system with 40 MHz *CoreClk* and *SysClk* at DMA rate of 1kHz, the *GNSS_DMACtrl* register would have to be configured to 0x00000FFE. Then it would take:

11 *CoreClk* cycles to initialize the DMA

23 x 1 *CoreClk* cycles to step through the register *GNSS_DMACtrl*

11 x 5 *CoreClk*cycles to transfer the observables into the FIFO

11 x 9 *SysClk* cycles to transfer the observables out of the FIFO to the destination (e.g. external RAM)

Note: For timing calculations it has to be taken into account that the 11x5 transfer and the 11x9 transfer are done in parallel, hence the spacing between two successive transfers of one channel is at least 5 CoreClk or 9 SysClk cycles (whichever is longer), possibly extended by wait states or other activity on the AHB bus.

In the case of DMA transfers at the same time, these are going to be processed sequentially, starting from channel-0 and finishing with channel-35.

Note: There is no risk that one processes the IE interrupt while the DMA is still running, since the IE interrupt is notified to the GIC only if the DMA for that channel is finished.

The data amount would be 11 x 32bit per millisecond, which is 352 kBit/s.

3.4.7.4 Timing Notes about GNSS Observables

Whenever the Code NCO 32bit phase has a wrap around, the Integration Epoch counter is increased by one. If it reaches the programmed level, the Integration Epoch Counter will signal the *Integration Epoch* (*IE*) and the IE observables are latched. However from the point in time when the code NCO signals the wrap around until the latching of the IE observables it takes exactly 3 *CoreClk* cycles.

The Measurement Epoch latching is artificially delayed by 3 *CoreClk* cycles so that both time lines ME and IE are consistent.

Example: If in one channel the IE would coincide with the ME than the channel observables would be:

ME_IMT_LSW = IE_IMT_LSW and

ME_CodePhase = IE_CodePhase

3.4.8 Channel Slaving Concepts

Up to 36 channels can be slaved together. There are two ways of slaving channels:

- Hardware Slaving (known also from AGGA-2)
- Software Slaving (new)

Besides this there are different applications where slaving is useful. Some of them will be treated in this section in order to get familiar with the slaving concepts. The cases which are treated as examples are:

- Attitude Determination
- Fast Acquisition / Multipath Detection
- Correlation Function Probing

3.4.8.1 Hardware/Software Slaving

It has to be noted that Hardware and Software Slaving are not fully equivalent to each other. Both methods have advantages and disadvantages.

Hardware Slaving means that the signal (e.g. Carrier Frequency, Code Frequency, PRN Code, etc) is slaved to the next channel via hardware. The software configures the slaving once and then the information from the master channel is spread to the slave channels without interaction from software. The restriction is that only subsequent channels can be slaved. That means it is for example possible to slave channels 1,2,3,4 but not 1,3,4,5, since the chain is interrupted at channel 2.

Software slaving means that the software is copying the relevant information (Carrier NCO word, Code NCO word) to the slave channels. The advantage is that any channel can be slaved, also if the channels are not subsequent. Another advantage is that the software can apply modifications to the slaved values and by thus e.g. apply an offset to the copied NCO word. The disadvantage however is an increase in CPU load and that not everything can be slaved via software.

If Hardware Slaving is used it has to be noted that there is one *CoreClk* delay in the slaved signals. This can also be seen in Figure 3-30. This "one *CoreClk* delay" feature is necessary in order to make sure that the signals from the first channel can be slaved up to the last channel without problems in signal propagation time.

The following table gives an overview of the signals which can be slaved from channel to channel:

Signal to slave	Hardware	Software
Input Data	InputSel = 12	N/A
ME	TimeBaseSel = 1	N/A
PPS	TimeBaseSel = 1	N/A
Carrier	CarrSel = 1	Yes (copy)
Delay Clock	CodeSel = 1	Yes (config), but Restriction
Code Out 1	CodeSel = 1	Yes (config)
Code Out 2	CodeSel = 1	Yes (config)
Primary Rate	CodeSel = 1	Yes (copy)
BOC Rate (cos)	CodeSel = 1	Yes (copy)
BOC Rate (sin)	CodeSel = 1	Yes (copy)
LE	IntEpochSel = 1	Yes (config)
IE	IntEpochSel = 1	Yes (config)
ASE	TimeBaseSel = 1	N/A

Table 3-29: Slaving possibilities

The table has to be read as follow: If Hardware slaving is chosen, the Input Data has to be delayed by one *CoreClk* cycle, since also the local signals (e.g. Carrier, Code) are delayed by one *CoreClk* cycle. This is also true for timing signals like PPS, ME and ASE. However in case of software slaving this is not applicable, since there is no delay in the signals (e.g. Carrier, Code) between the channels. Therefore also the input data and timing signals must not be delayed.

The Carrier for example can be slaved via Hardware if the bit *CarrSel* in the *ChannelCtrl* register is set to 1. Alternatively it can be slaved via software. Then the software has to adjust the carrier NCO phase once and from there on just copy the carrier NCO word (frequency) from the master channel to the slave channel. The carrier phase of two channels can be synchronized with the help of the *ME_CarrObsPhase* register. It can be configured such that all 32bits of the Carrier NCO phase are accessible. From that observable the phase difference of two channels can be read out. This phase difference can then be adjusted with the help of the *CarrSwShift* register.

The Delay Clock can be slaved via Hardware Slaving by setting the *CodeSel* bit in the *ChannelCtrl* register to 1. Alternatively it can be slaved via software in that sense that the Delay Line is configured the same in the slave channel than in the master channel. Note that it is not possible with software slaving to build up a correlator chain, meaning that one delay line is chained with a delay line from another channel.

The PRN code can be slaved via Hardware Slaving by setting the *CodeSel* bit in the *ChannelCtrl* register to 1. Alternatively it can be slaved via software in that sense that the Code Generator is configured the same in the slave channel than in the master channel.

The PrimaryRate, the BOCRate(cos) and BOCRate(sin) are all derived from the Code NCO word. The Code NCO word for example can be slaved via Hardware if the bit *CodeSel* in the *ChannelCtrl* register is set to 1. Alternatively it can be slaved via software. Then the software has to adjust the code NCO phase once and from there on just copy the code NCO word (frequency) from the master channel to the slave channel. The code phase of two channels can be synchronized with the help of the *ME_CodePhase* register. From that observable the phase difference of two channels can be read out. This phase difference can then be adjusted with the help of the *CodeSwShift* register.

The Integration Epoch and Long Epoch can be slaved via Hardware Slaving by setting the *IntEpochSel* bit in the *ChannelCtrl* register to 1. Alternatively the Integration and Long Epoch can be configured via software in the slave channel such that they are equal or controllable delayed to the one in the master channel.

3.4.8.2 Examples

For **Attitude Determination** typically two or more channels are slaved together. The signal is tracked on the master channel and all slave channels should behave the same like the master channel (Carrier NCO, Code NCO, Code). Then it is possible to read out the phase difference from the I/Q correlation values of the slave(d) channel(s).

In case of Hardware Slaving the following bits would have to be in the *ChannelCtrl* register:

- *CarrSel* = 1
- *CodeSel* = 1
- *IntEpochSel* = 1
- *TimeBaseSel* = 1
- *InputSel* = 12

Additionally the Delay Line must be configured such that the correlators in the slave channel(s) see the same PRN code phase than the correlators in the master channel (see 3.4.4 Code Delay Line Unit (CDLU)). Also in the slave channels the correlators can be configures special if Hybrid Attitude Mode is used (see 3.6.3 Antenna Switch Correlation Results).

Since there is always one *CoreClk* delay between the slaved signals (e.g. Carrier, Code), also the incoming data has to be delayed by one *CoreClk* cycle (*InputSel* = 12) in order to maintain the phase relation between local data and input data.

For **Fast Acquisition** and **Multipath Detection** it might be of interest to have many correlators slaved together. In this case, the same configuration of the *ChannelCtrl* register should be used than in the Attitude Determination Case. However the Delay Line would have to be configures such that the correlators are not working parallel (like in the Attitude Determination Case), but are chained (see see 3.4.4 Code Delay Line Unit (CDLU)).

3.5 Time Base Generator

The Time Base Generator anticipated in Figure 3-1 produces the Epoch Clock (EC), the Measurement Epoch (ME), and the Pulse-Per-Second (PPS). Also it provides the Instrument Measurement Time (IMT) as well as an external clock Interface. The Time Base Generator is depicted in Figure 3-55.

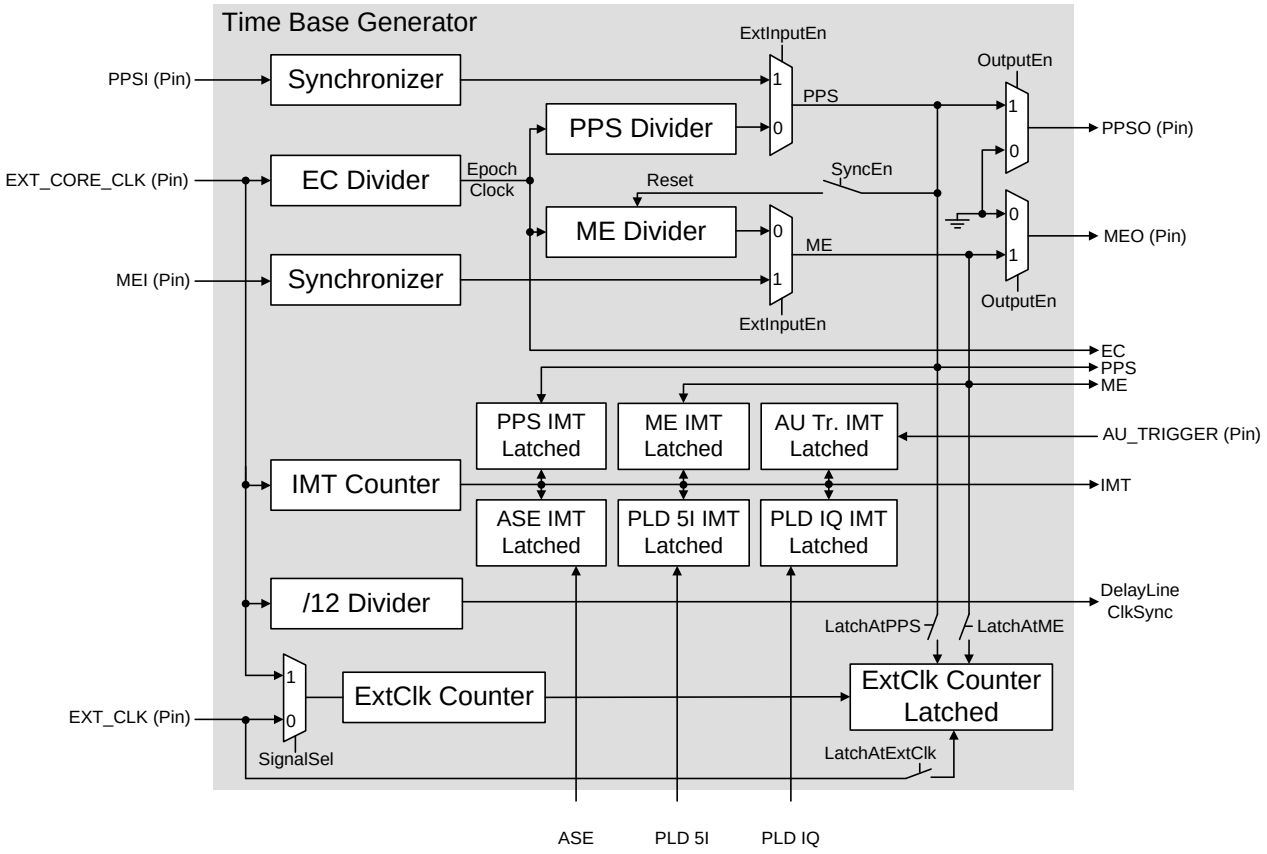


Figure 3-55: Time Base Generator Block Diagram

3.5.1 Epoch Clock (EC)

The EC Divider is used to divide the *CoreClk* and to output the Epoch Clock. This is a divider with a 17 bit integer part and a 10 bit fractional part and can be programmed via the *EpochClkDiv* register. Note that the integer part has to be programmed with the desired value -1. New settings written to the Epoch Clock Divider become effective with the next Epoch Clock. The Epoch Clock is used for the generation of the ME, the PPS and the Antenna Switch Epoch (ASE) strobe.

Example: Assuming a *CoreClk* of 30,456285 MHz and a desired Epoch Clock of 1 kHz, the division ratio would be 30456,285. The value which would have to be programmed is 30456 – 1 = 30455 for the integer part and 0.285 * 1024 = 291.84 => 292 for the fractional part.

3.5.2 Measurement Epoch (ME)

The Measurement Epoch is used to sample the observables of all channels and to synchronize the measurements of multiple AGGA's if more than one AGGA is used.

The ME Divider is used to divide the Epoch Clock and to output the Measurement Epoch. This is a programmable 14 bit divider. It is programmed via the *DivRatio* field of the *MESettings* register. Note that the divider has to be programmed with the desired value – 1 and that new settings become effective with the next ME event.

The Measurement Epoch can be selected from an internally generated Measurement Epoch or an external input (MEI: Measurement Epoch Input). This can be selected by the *ExtInputEn* bit in the *MESettings* register. If the external input

is used a synchronizer will synchronize the external *MEI* signal with the internal *CoreClk*. Internally the ME signal is routed to the instances (e.g. channel), but it is also available as an output port (MEO: Measurement Epoch Output). The internal ME signal is asserted for one *CoreClk* cycle, while the external *MEO* signal is asserted for 128 *CoreClk* cycles. Other than in AGGA-2, the *MEI* has not to be connected externally to the *MEO* if only one AGGA-4 is used. In receivers with multiple AGGA-4s all *MEI* should be connected to the *MEO* from one AGGA-4, referred to as the master AGGA-4 in order to ensure that all channels in the receiver use the same Measurement Epoch (ME).

The *OutputEn* field of the *MESettings* register controls whether the *MEO* signal is enabled or constantly de-asserted.

It is also possible to generate an interrupt for the Measurement Epoch (see chapter 4.12.2).

The ME time instance is latched to the registers *ME_IMT_MSW* and ME*_IMT_LSW*.

Note: If MEI is used it takes 4 *CoreClk* cycles from the rising edge of the MEI input signal until the latching of the corresponding IMT value.

Note: If MEI is used then the difference between MEI and MEO is 3 to 5 *CoreClk* cycles.

3.5.3 Pulse Per Second (PPS)

The *PPS* output is intended for synchronizing external equipment to the receiver time.

The PPS Divider is used to divide the Epoch Clock and to output the Pulse Per Second. This is a programmable 14 bit divider. It is programmed via the *DivRatio* field of the *PPSSettings* register. Note that the divider has to be programmed with the desired value – 1 and that new settings become effective with the next PPS event.

The Pulse Per Second can be selected from an internally generated Pulse Per Second or an external input (PPSI: Pulse Per Second Input). This can be selected by the *ExtInputEn* bit in the *PPSSettings* register. If the external input is used a synchronizer will synchronize the external PPSI signal with the internal *CoreClk*. Internally the PPS signal is routed to the instances as PPS signal, but it is also available as an output port (PPSO: Pulse Per Second Output). The internal PPS signal is asserted for one *CoreClk* cycle, while the external *PPSO* signal is asserted for 128 *CoreClk* cycles. Other than in AGGA-2, the *PPSI* does not need to be connected externally to the *PPSO* if only one AGGA-4 is used. In receivers with multiple AGGA-4 all *PPSI* should be connected to the *PPSO* from one AGGA-4, referred to as the master AGGA-4, to ensure that all channels in the receiver use the same Pulse Per Second.

The *OutputEn* field of the *PPSSettings* register controls whether the *PPSO* is enabled or constantly de-asserted.

It is also possible to generate an interrupt for the Pulse Per Second (see chapter 4.12.2).

The PPS time instance is latched to the registers *PPS_IMT_MSW* and *PPS_IMT_LSW*.

Note: If PPSI is used then the difference between PPSI and PPSO is 4 to 6 *CoreClk* cycles.

3.5.4 Synchronisation of ME and PPS

By setting the bit *SyncEn* in the *PPSSettings* register, the Measurement Epoch (ME) is phase aligned with the PPS once. After writing the *SyncEn* bit, the hardware detects and clears it immediately. In case that the PPS is generated inside the AGGA-4, only the ME Divider is reloaded to its nominal value. In case that the PPS is coming from external, the ME Divider and the EC Divider (integer and fractional part) are reloaded with their nominal values. By this it can be assured that the PPS strobe and the ME strobe are phase aligned at the time of alignment.

3.5.5 Instrument Measurement Time (IMT)

The 64 bit IMT Counter counts *CoreClk* cycles. It can be read via the *IMT_MSW* and *IMT_LSW* registers at any time and it will return the current IMT count. The *IMT_MSW* register contains the Most Significant Word (upper 32 bits) of the IMT Counter, while the *IMT_LSW* register contains the Lowest Significant Word (lower 32bits) of the IMT Counter. By writing to these registers, the IMT Counter can be preset with any value. The written value becomes effective with the next Measurement Epoch. Note that always both parts (*IMT_MSW* and *IMT_LSW*) have to be written (the order does not matter), before the hardware executes the IMT preset. If this protection mechanism would not be there it could happen that an ME occurs while only one part of the IMT counter (either *IMT_MSW* or *IMT_LSW*) has already been written. Then the ME Observables would indicate an inconsistent time stamp.

When reading the current (unlatched) IMT count (not the IMT observables) the user is advised to first read the the *IMT_MSW*, second the *IMT_LSW* and third again the IMT_MSW. That way a wrap around in the IMT counter can be detected and corrected.

At the following GNSS events (PPS, ME, ASE, PLD 5I, PLD IQ, AU Trigger) the counter is latched and the values are stored to the corresponding IMT latch registers (see also Figure 3-55). Therefore these events can be precisely time tagged.

The lower 32 bits of the IMT Count (= *IMT_LSW*) are distributed within the GNSS core as time stamp. Each channel for example uses it to precisely time tag the Integration Epoch.

IMT_MSW and *IMT_LSW* are reset by the Power On or AGGA4 reset pins (*PWR_ON_RESET_N*, *GNSS_RESET_N*), but they are not cleared by an AGGA4 Reset (see section 6.5.3).

3.5.6 External Clock (ExtClk) Interface

The External Clock Interface can be used for various purposes. The receiver can for example calculate clock corrections for its own oscillator even without a Position Velocity Time (PVT) if the External Clock input is connected to a stable external reference.

The External Clock Counter can either be clocked by the signal coming from the EXT_CLK input pin or with the signal coming from the EXT_CORE_CLK input pin. This can be programmed via the *SignalSel* bit in the *ExtClkSettings* register. The External Clock Counter is 32bit wide and has a natural wrap around. It is detecting the rising edge of the counting signal. The External Clock Counter can be read at any time by reading the register *ExtClkCnt*. It will return the current value. The External Clock Counter can count frequencies up to 50 MHz.

Additionally the counter can be latched at the following events (ME, PPS, *ExtClk*). The latched value can be read by reading the register *ExtClkCntLatched*. Depending on the switch bits, the latched counter shall do the following actions:

Switch Bits			Actions		
extClkLatchAtExtClk	*extClkLatchAtME*	*extClkLatchAtPPS*	*ExtClkCnt* is latched with trigger from EXT_CLK input	*ExtClkCnt* is latched with ME	*ExtClkCnt* is latched with PPS
0	0	0	No	No	No
0	0	1	No	No	Yes
0	1	0	No	Yes	No
0	1	1	No	Yes	Yes
1	0	0	Yes	No	No
1	0	1	Yes	No	Yes
1	1	0	Yes	Yes	No
1	1	1	Yes	Yes	Yes

Table 3-30: Action Table for *ExtClkCntLatched*

Note: If the External Clock Counter is running with *CoreClk* all latch triggers are valid (ME, PPS, EXT_CLK input pin). If the External Clock Counter is running with the signal from the EXT_CLK input pin only the ME and PPS latch triggers are valid.

3.5.7 Delay Line Clock Sync

In order to make sure that the delay line clock divider counter state is phase coherent over all channels, a synchronization signal DelayLineClkSync is generated every 12 *CoreClk* cycles in the Time Base module. This signal simultaneously clears all clock dividers, resulting in a phase-alignment of all delay line clocks regardless of their division ratio.

3.6 Antenna Switch Controller (ASC)

This section describes the Antenna Switch Controller (ASC). As shown in Figure 3-56, it consists of the Antenna Switch Epoch Output (*ASEO*) Divider, a Synchroniser for the Antenna Switch Epoch Input (*ASEI)* and the Antenna Switch Sequencer.

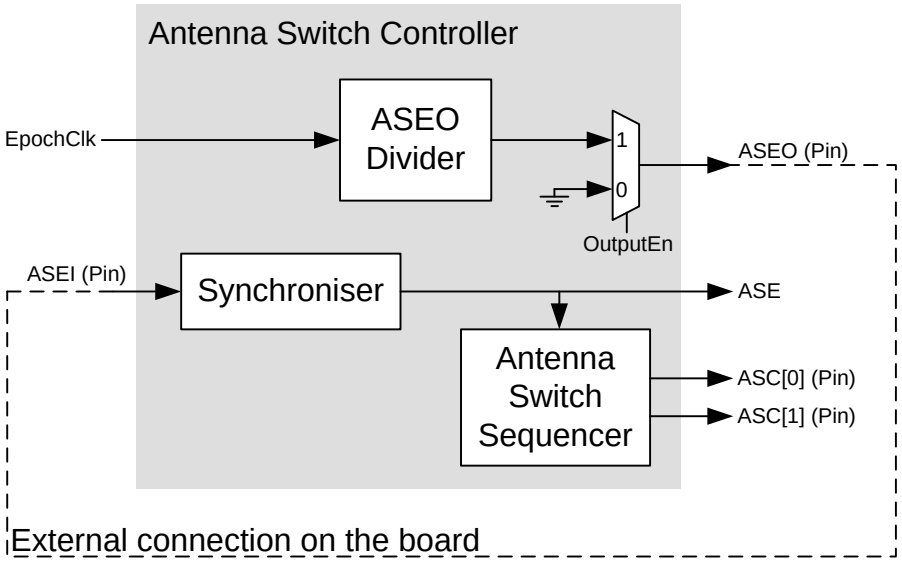


Figure 3-56: Antenna Switch Controller (ASC)

The Antenna Switch Controller supports the Hybrid Parallel-Multiplex attitude determination scheme (see [RD-05]). The two lines Antenna Switch Control *ASC* outputs are intended for controlling external antenna switching of up to four antennas (see also Figure 3-57). Also the Antenna Switch Epoch *ASE* strobe is generated, which signals antenna switch events to the channels and which can be used to control the updating of the correlation values (see chapter 3.6.3). An interrupt can be generated whenever *ASE* occurs (see chapter 4.12.2). Both the *ASE* strobe and the *ASC* outputs are generated from the Antenna Switch Epoch Input *ASEI*.

In receivers with one AGGA-4 *ASEI* should be externally connected to its own *ASEO* output. In receivers with multiple AGGA-4 all *ASEI* inputs should be connected to the *ASEO* signal from one AGGA-4, referred to as the master AGGA-4, to ensure that all channels in the receiver use the same *ASE*.

3.6.1 Antenna Switch Epoch (ASE)

The *ASE* strobe signals the end of each Antenna Switch Epoch. It is taken from the *ASEI* input after synchronization and detection of the rising edge. An interrupt can be generated whenever *ASE* occurs.

The *ASEO* signal can have frequencies from *EpochClk* to *EpochClk/1024* depending on the programming of the ASEO Divider. This divider can be programmed by the *DivRatio* field in the *AntSwitchCtrl* register. A typical value for *ASEO* would be 200 Hz.

The *OutputEn* bit of the *AntSwitchCtrl* register controls whether the *ASEO* output is enabled or constantly de-asserted. When it is disabled the ASEO Divider is also disabled. If enabled, the *ASEO* output is asserted for 128 *CoreClk* cycles once it occurs. The internal *ASE* signal is asserted for one *CoreClk* cycle.

In order to reduce switching noise the integrators are inhibited at the start of each Antenna Switch Epoch for 128 *CoreClk* cycles.

The time instance of the Antenna Switch Epoch is latched in the registers *ASE_IMT_MSW* and *ASE_IMT_LSW*

3.6.2 Antenna Switch Sequencer

The Antenna Switch Sequencer outputs on the two pins *ASC[0]* and *ASC[1]* which antenna is currently asserted to control the external antenna switching of up to four antennas to a common RF/IF section as depicted in Figure 3-57. To be clear; this antenna multiplexing scheme is only needed for the Hybrid Parallel-Multiplex attitude determination

scheme (see [RD-05]), not for a parallel attitude determination scheme, where each AGGA Input Module is connected to one dedicated antenna or for normal positioning applications.

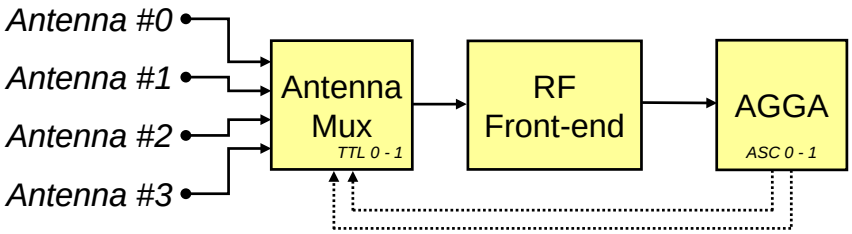


Figure 3-57: AGGA-4 Antenna Multiplexing

The mapping is as follow:

Antenna	ASC[0]	ASC[1]
#0	0	0
#1	0	1
#2	1	0
#3	1	1

Table 3-31: ASC Output Mapping

The antennas can be switched in a sequence. The sequence can be programmed by the *SwitchSequencer* field in the *AntSwitchCtrl* register. At the start of each Antenna Switch Epoch the Antenna Switch Sequencer asserts the next higher *ASC* output that has been enabled by the *SwitchSequencer* field. If no bit has been set in the *SwitchSequencer* field the *ASC* output is static. After PowerOn Reset the ASC pins have the assignment ASC[1:0] = 0, while after previous ASC start/stop, the ASC[1:0] assignment must not necessarily be zero, but remains static at it's last state.

At the end of each Antenna Switch Epoch (ASE) the number of the antenna that was asserted during the last period is stored in the read-only *SwitchID* field of the *AntSwitchCtrl* register. By reading this value the firmware can reconstruct to which antenna the correlation values correspond.

3.6.3 Antenna Switch Correlation Results

If the *ASEEn* bit in the *CorrUnitCtrl* register is enabled, then the correlators of a channel behave differently. The following table depicts this (see also Table 3-20):

	ASEEn = 0			ASEEn = 1		
	Multiplication	**reset @**	**latch @**	**Multiplication**	**reset @**	**latch @**
EE	Incoming Signal x EE	IE or LE	IE	Incoming Signal x EE	IE or LE	IE
E	Incoming Signal x E	IE or LE	IE	Incoming Signal x P	(IE or LE) and ASE	ASE
P	Incoming Signal x P	IE or LE	IE	Incoming Signal x P	(IE or LE) and ASE	IE
L	Incoming Signal x L	IE or LE	IE	Incoming Signal x P	ASE	ASE
LL	Incoming Signal x LL	IE or LE	IE	Incoming Signal x LL	IE or LE	IE

Table 3-32: Action Table for Correlators

The Early and Punctual correlator together result in the same functionality than AGGA-2/3 was giving. However if the ASC mechanism is applied for signals where no inversion due to data is expected (e.g. pilot signals) then the AGGA-4 offers an additional mode which simplifies the handling of the correlation values for the software. The late correlator is resetting and latching with ASE only. Therefore no reconstruction of the ASE correlation value is needed. It can directly be read out.

Figure 3-58 shows graphically how the Early, Late and Punctual correlator work together with the Antenna Switch Controller.

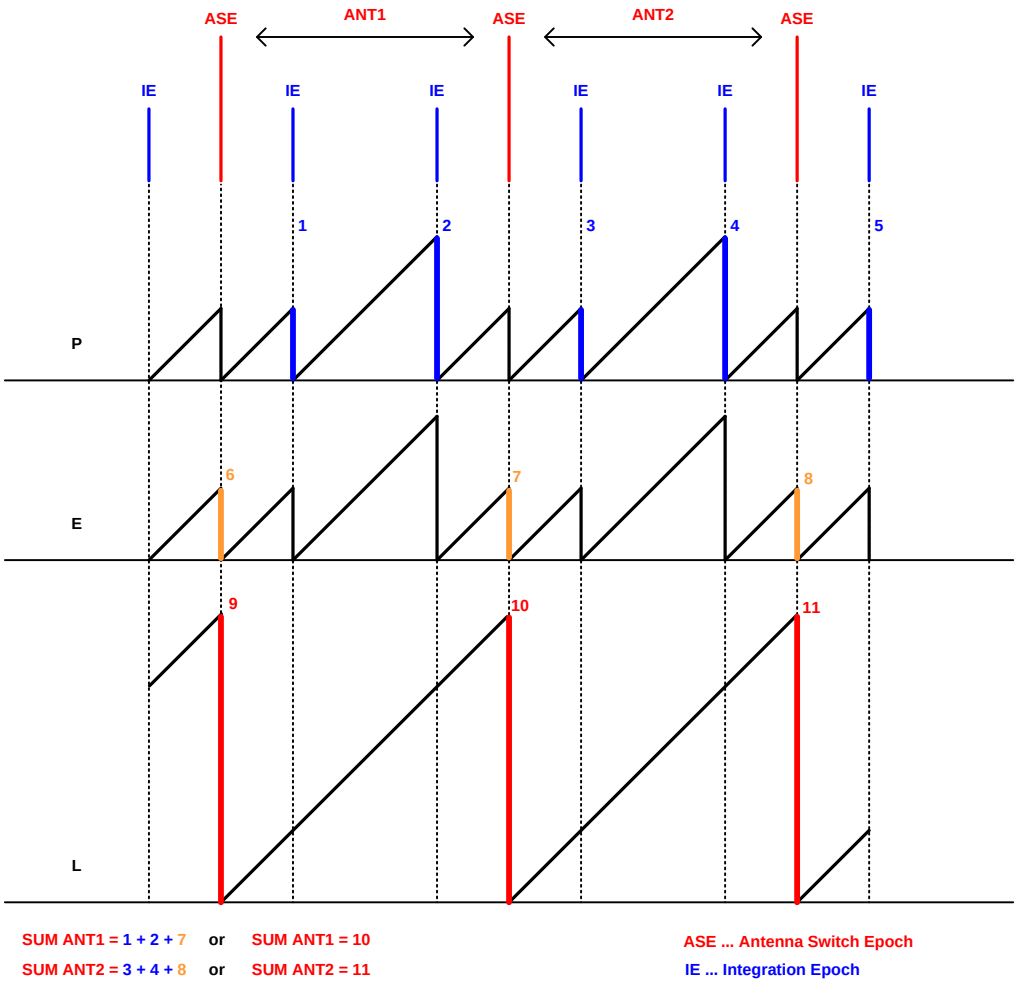


Figure 3-58: ASC Behaviour on Correlators

3.7 AMBA High Performance Bus (AHB) Interface

As anticipated in Figure 2-1, the GNSS module is connected to the AMBA High Performance BUS (AHB) via two interfaces: the AHB master and the AHB slave interface. The AHB master interface is required to implement DMA capability to the GNSS module and the AHB slave interface is used to access all module internal registers (read and write).

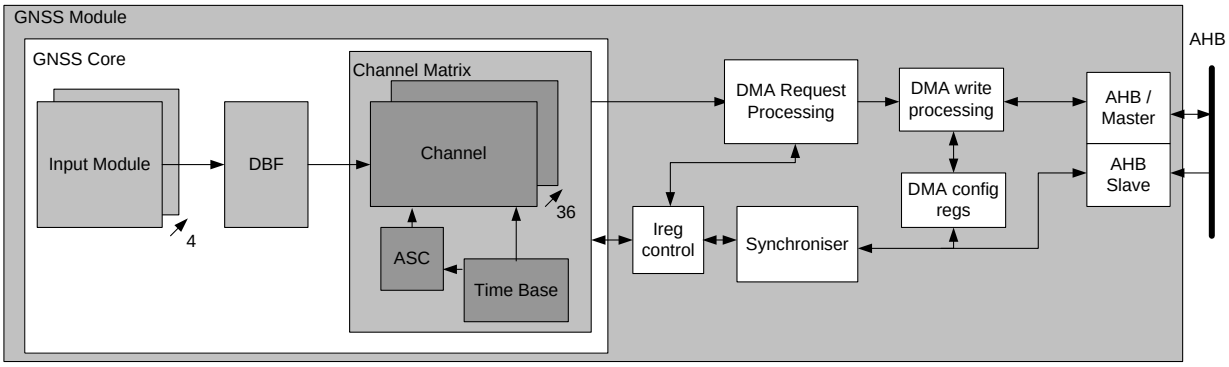


Figure 3-59: GNSS Module

4 Processor Module

AGGA-4 uses the LEON-2 Fault Tolerant Processor together with the Cobham Gaisler Floating Point Unit (GRFPU). An overview to the Processor Module and its corresponding periphery can be seen in Figure 4-1 (the solid blocks, which are not greyed out).

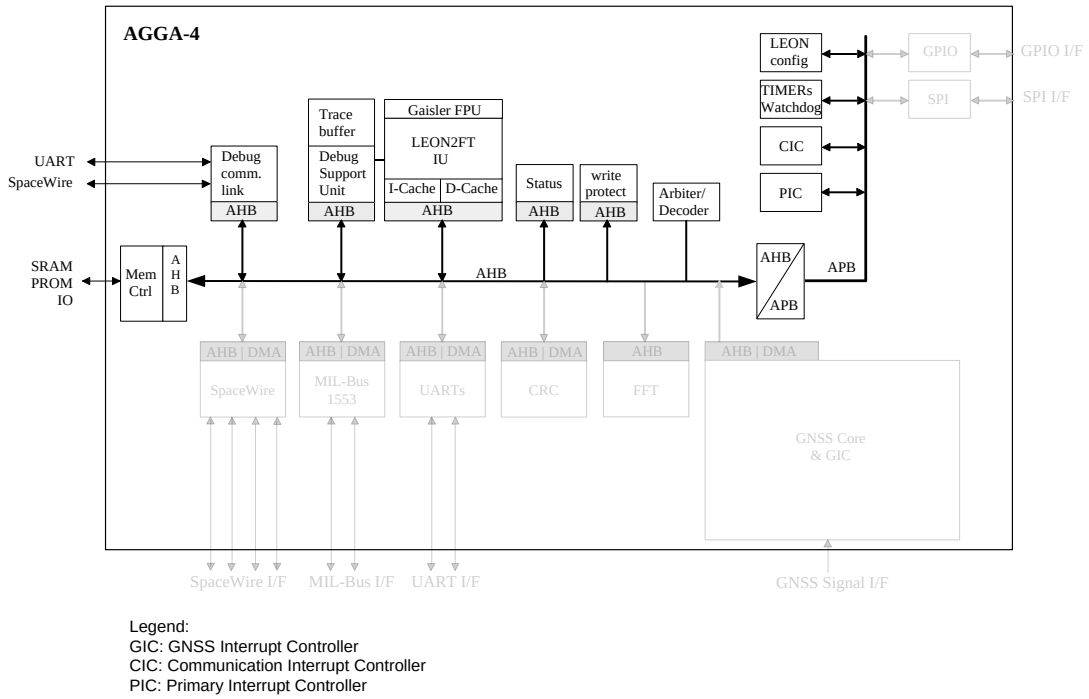


Figure 4-1: Overview of Processor Modules

The LEON-2 processor implements the SPARC V8 standard as defined in the SPARC V8 architectural manual (see [AD-01], [RD-06]). The LEON-2-FT version implemented is 1.0.9.16.2.

The following chapters give an overview to the Leon-2 modules.

4.1 Integer Unit

4.1.1 Instruction Timing

Table 4-1 lists the cycles per instruction (assuming cache hit and no load interlock):

Instruction	Cycles
JMPL	2
Double Load	2
Single Store	2
Double Store	3
SMUL/UMUL	5
SDIV/UDIV	35
Taken Trap	4
Atomic load/store	3
All other intructions	1

Table 4-1 Instruction timing

4.2 Floating Point Unit (FPU)

The AGGA-4 uses the Cobham Gaisler Floating Point Unit (GRFPU). It is a high-performance FPU implementing floating-point operations as defined in IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754) and SPARC V8 standard (IEEE-1754). Supported formats are single and double precision floating-point numbers. The advanced design combines two execution units, a fully pipelined unit for execution of the most common FP operations and a non-blocking unit for execution of divide and square-root operations. The logical view of the GRFPU is shown in Figure 4-2.

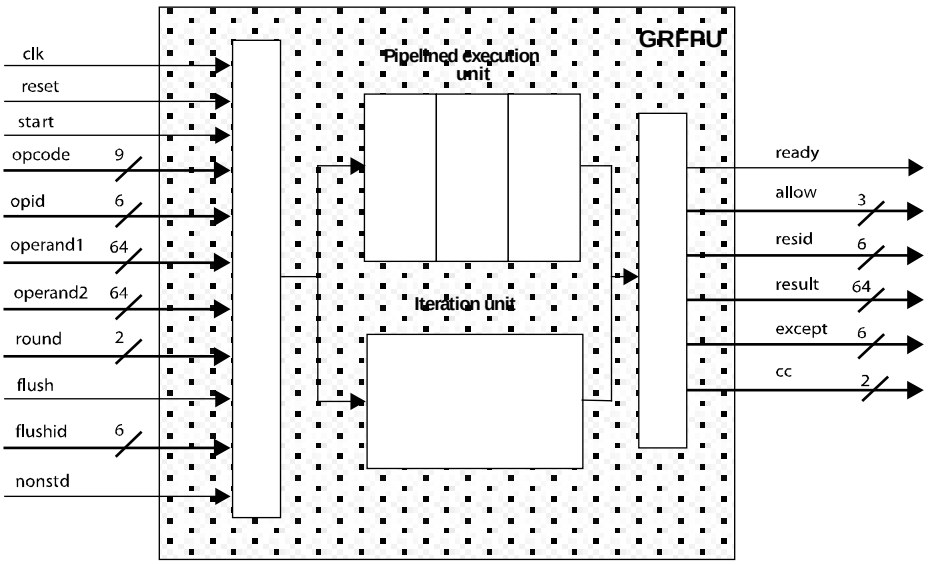


Figure 4-2: Cobham Gaisler Floating Point Unit (GRFPU)

The following chapters describe the GRFPU from functional point of view. Chapter "Functional description" gives details about GRFPU's implementation of the IEEE-754 standard including FP formats, operations, opcodes, operation timing, rounding and exceptions. "Signals and timing" describes the GRFPU interface and its signals. "GRFPU Control Unit" describes the software aspects of the GRFPU integration into a LEON processor through the GRFPU Control Unit - GRFPC. For implementation details refer to the white paper, "GRFPU - High Performance IEEE-754 Floating-Point Unit" (available at *www.gaisler.com*).

4.2.1 Floating-point number formats

GRFPU handles floating-point numbers in single or double precision format as defined in the IEEE-754 standard with exception for denormalized numbers. See section Denormalized numbers for more information on denormalized numbers.

4.2.2 FP operations

GRFPU supports four types of floating-point operations: arithmetic, compare, convert and move. The operations implement all FP instructions specified by SPARC V8 instruction set, and most of the operations defined in IEEE-754. All operations are summarized in Table 4-2, with their opcodes, operands, results and exception codes. Throughputs and latencies and are shown in Table 4-3.

Operation	OpCode[8:0]	Op1	Op2	Result	Exceptions	Description
Arithmetic operations						
FADDS FADDD	1000001 1000010	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Addition
FSUBS FSUBD	1000101 1000110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Subtraction
FMULS FMULD FSMULD	1001001 1001010 1101001	SP DP SP	SP DP SP	SP DP DP	UNF, NV, OF, UF, NX UNF, NV, OF, UF, NX UNF, NV, OF, UF	Multiplication, FSMULD gives exact double-precision product of two single-precision
FDIVS FDIVD	1001101 1001110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX, DZ	Division
FSQRTS FSQRTD	101001 101010	- -	SP DP	SP DP	UNF, NV, NX	Square-root
Conversion operations						
FITOS FITOD	11000100 11001000	-	INT	SP DP	NX -	Integer to floating-point conversion
FSTOI FDTOI	11010001 11010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. The result is rounded in round-to-zero mode.
FSTOI_RND FDTOI_RND	111010001 111010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. Rounding according to RND input.
FSTOD FDTOS	11001001 11000110	-	SP DP	DP SP	UNF, NV UNF, NV, OF, UF, NX	Conversion between floating-point formats
Comparison operations						
FCMPS FCMPD	1010001 1010010	SP DP	SP DP	CC	NV	Floating-point compare. Invalid exception is generated if either operand is a signaling NaN.
FCMPES FCMPED	1010101 1010110	SP DP	SP DP	CC	NV	Floating point compare. Invalid exception is generated if either operand is a NaN (quiet or signaling).
Negate, Absolute value and Move						
FABSS	1001	-	SP	SP	-	Absolute value.
FNEGS	101	-	SP	SP	-	Negate.
FMOVS	1		SP	SP	-	Move. Copies operand to result output.

SP - single precision floating-point number
DP - double precision floating-point number
INT - 32 bit integer
CC - condition codes, see table : Signal descriptions
UNF, NV, OF, UF, NX - floating-point exceptions, see section Exceptions

Table 4-2: GRFPU Operations

Arithmetic operations include addition, subtraction, multiplication, division and square-root. Each arithmetic operation can be performed in single or double precision formats. Arithmetic operations have one clock cycle throughput and a latency of four clock cycles, except for divide and square-root operations, which have a throughput of 16 - 25 clock cycles and latency of 16 - 25 clock cycles (see Table 4-3). Add, sub and multiply can be started on every clock cycle, providing high throughput for these common operations. Divide and square-root operations have lower throughput and higher latency due to complexity of the algorithms, but are executed in parallel with all other FP operations in a non-blocking iteration unit. Out-of-order execution of operations with different latencies is easily handled through the GRFPU interface by assigning an id to every operation which appears with the result on the output once the operation is completed (see section4.2.8).

Operation	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD, FMULS, FMULD, FSMULD	2	5
FITOS, FITOD, FSTOI, FSTOI_RND, FDTOI, FDTOI_RND, FSTOD, FDTOS	2	5
FCMPS, FCMPD, FCMPES, FCMPED	2	5
FDIVS	17	17
FDIVD	17.5 (16/19)*	17.5 (16/19)*
FSQRTS	25	25
FSQRTD	25.5 (24/27)*	25.5 (24/27)*
* Throughput and latency are data dependant with two possible cases with equal statistical possibility.		

Table 4-3: Throughput and Latency

Conversion operations execute in a pipelined execution unit and have throughput of one clock cycle and latency of four clock cycles. Conversion operations provide conversion between different floating-point numbers and between floating-point numbers and integers.

Comparison functions offering two different types of quiet Not-a-Numbers (QNaNs) handling are provided. Move, negate and absolute value are also provided. These operations do not ever generate unfinished exception (unfinished exception is never signaled since compare, negate, absolute value and move handle denormalized numbers).

4.2.3 Exceptions

GRFPU detects all exceptions defined by the IEEE-754 standard. This includes detection of Invalid Operation (NV), Overflow (OF), Underflow (UF), Division-by-Zero (DZ) and Inexact (NX) exception conditions. Generation of special results such as NaNs and infinity is also implemented. Overflow (OF) and underflow (UF) are detected before rounding. If an operation underflows the result is flushed to zero (GRFPU does not support denormalized numbers or gradual underflow). A special Unfinished exception (UNF) is signaled when one of the operands is a denormalized number which is not handled by the arithmetic and conversion operations.

4.2.4 Rounding

All four rounding modes defined in the IEEE-754 standard are supported: round-to-nearest, round-to-+inf, round-to--inf and round-to-zero.

4.2.5 Denormalized numbers

Denormalized numbers are not handled by the GRFPU arithmetic and conversion operations. A system (microprocessor) with the GRFPU could emulate rare cases of operations on denormals in software using non-FPU operations. A special Unfinished exception (UNF) is used to signal an arithmetic or conversion operation on the denormalized numbers. Compare, move, negate and absolute value operations can handle denormalized numbers and do not raise the unfinished exception. GRFPU does not generate any denormalized numbers during arithmetic and conversion operations on normalized numbers. If the infinitely precise result of an operation is a tiny number (smaller than minimum value representable in normal format) the result is flushed to zero (with underflow and inexact flags set).

4.2.6 Non-standard Mode

GRFPU can operate in a non-standard mode where all denormalized operands to arithmetic and conversion operations are treated as (correctly signed) zeroes. Calculations are performed on zero operands instead of the denormalized numbers obeying all rules of the floating-point arithmetics including rounding of the results and detecting exceptions.

4.2.7 NaNs

GRFPU supports handling of Not-a-Numbers (NaNs) as defined in the IEEE-754 standard. Operations on signaling NaNs (SNaNs) and invalid operations (e.g. inf/inf) generate the Invalid exception and deliver QNaN_GEN as result. Operations on Quiet NaNs (QNaNs), except for FCMPES and FCMPED, do not raise any exceptions and propagate QNaNs through the FP operations by delivering NaN-results according to Table 4-4. QNaN_GEN is 0x7fffe00000000000 for double precision results and 0x7fff0000 for single precision results.

		Operand 2		
		FP	QNaN2	SNaN2
Operand 1	none	FP	QNaN2	QNaN_GEN
	FP	FP	QNaN2	QNaN_GEN
	QNaN1	QNaN1	QNaN2	QNaN_GEN
	SNaN1	QNaN_GEN	QNaN_GEN	QNaN_GEN

Table 4-4: Operations on NaNs

4.2.8 Timing

The FPU is fully pipelined and a new operation can be started every clock cycle. The only exceptions are divide and square-root operations which require 16 to 26 clock cycles to complete, and which are not pipelined. Division and square-root are implemented through iterative series expansion algorithm. Since the algorithms basic step is multiplication the floating-point multiplier is shared between multiplication, division and square-root. Division and square-root do not occupy the multiplier during the whole operation and allow multiplication to be interleaved and executed parallelly with division or square-root.

4.2.9 GRFPC - GRFPU Control Unit

The GRFPU Control Unit (GRFPC) is used to attach the GRFPU to the LEON integer unit (IU). GRFPC performs scheduling, decoding and dispatching of the FP operations to the GRFPU as well as managing the floating-point register file, the floating-point state register (FSR) and the floating-point deferred-trap queue (FQ). Floating-point operations are executed in parallel with other integer instructions, the LEON integer pipeline is only stalled in case of operand or resource conflicts.

4.2.10 Floating-Point register file

The GRFPU floating-point register file contains 32 32-bit floating-point registers (%f0-%f31). The register file is accessed by floating-point load and store instructions (LDF, LDDF, STD, STDF) and floating-point operate instructions (FPop).

4.2.11 Floating-Point State Register (FSR)

The GRFPC manages the floating-point state register (FSR) containing FPU mode and status information. All fields of the FSR register as defined in SPARC V8 specification are implemented and managed by the GRFPU conforming to the SPARC V8 specification and the IEEE-754 standard. Implementation-specific parts of the FSR managing are the NS (non-standard) bit and *ftt* field.

If the NS (non-standard) bit of the FSR register is set, all floating-point operations will be performed in non-standard mode as described in section 4.2.6. When the NS bit is cleared all operations are performed in standard IEEE-compliant mode.

Following floating-point trap types never occur and are therefore never set in the ftt field:

- unimplemented_FPop: all FPop operations are implemented
- hardware_error: non-resumable hardware error
- invalid_fp_register: no check that double-precision register is 0 mod 2 is performed

GRFPU implements the *qne* bit of the FSR register which reads 0 if the floating-point deferred-queue (FQ) is empty and 1 otherwise.

The FSR is accessed using LDFSR and STFSR instructions.

4.2.12 Floating-Point Exceptions and Floating-Point Deferred-Queue

GRFPU implements the SPARC deferred trap model for floating-point exceptions (fp_exception). A floating-point exception is caused by a floating-point instruction performing an operation resulting in one of following conditions:

- an operation raises IEEE floating-point exception (ftt = IEEE_754_exception) e.g. executing invalid operation such as 0/0 while the NVM bit of the TEM field id set (invalid exception enabled).
- an operation on denormalized floating-point numbers (in standard IEEE-mode) raises unfinished_FPop floating-point exception
- sequence error: abnormal error condition in the FPU due to the erroneous use of the floating-point instructions in the supervisor software.

The trap is deferred to one of the floating-point instructions (FPop, FP load/store, FP branch) following the trap-inducing instruction (note that this may not be next floating-point instruction in the program order due to exception-detecting mechanism and out-of-order instruction execution in the GRFPC). When the trap is taken the floating-point deferred-queue (FQ) contains the trap-inducing instruction and up to seven FPop instructions that were dispatched in the GRFPC but did not complete.

After the trap is taken the *qne* bit of the FSR is set and remains set until the FQ is emptied. The STDFQ instruction reads a double-word from the floating-point deferred queue, the first word is the address of the instruction and the second word is the instruction code. All instructions in the FQ are FPop type instructions. The first access to the FQ gives a double-word with the trap-inducing instruction, following double-words contain pending floating-point instructions. Supervisor software should emulate FPops from the FQ in the same order as they were read from the FQ.

Note that instructions in the FQ may not appear in the same order as the program order since GRFPU executes floating-point instructions out-of-order. A floating-point trap is never deferred past an instruction specifying source registers, destination registers or condition codes that could be modified by the trap-inducing instruction. Execution or emulation of instructions in the FQ by the supervisor software gives therefore the same FPU state as if the instructions were executed in the program order.

4.3 Exceptions

The LEON core inside AGGA-4 adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority (TT=Trap Type, Pri=Priority).

Trap	TT	Pri	Description
reset	0x00	1	Power-on reset
write error	0x2b	2	write buffer error (illegal/write protected address access)
instruction_access_exception	0x01	3	Error during instruction fetch
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction
privileged_instruction	0x03	4	Execution of privileged instruction in user mode
fp_disabled	0x04	6	FP instruction while FPU disabled
cp_disabled	0x24	6	CP instruction while Co-processor disabled
watchpoint_detected	0x0B	7	Instruction or data watchpoint match
window_overflow	0x05	8	SAVE into invalid window
window_underflow	0x06	8	RESTORE into invalid window
mem_address_not_aligned	0x07	10	Memory access to un-aligned address
fp_exception	0x08	11	FPU exception
data_access_exception	0x09	13	Access error during load or store instruction (memory EDAC error)
tag_overflow	0x0A	14	Tagged arithmetic overflow
divide_exception	0x2A	15	Divide by zero
interrupt_level_1	0x11	31	Asynchronous interrupt 1 (AHB Error)
interrupt_level_2	0x12	30	Asynchronous interrupt 2 (Parallel I/O 0)
interrupt_level_3	0x13	29	Asynchronous interrupt 3 (Timer 1)
interrupt_level_4	0x14	28	Asynchronous interrupt 4 (FFT Done)
interrupt_level_5	0x15	27	Asynchronous interrupt 5 (GNSS low)
interrupt_level_6	0x16	26	Asynchronous interrupt 6 (Parallel I/O 1)
interrupt_level_7	0x17	25	Asynchronous interrupt 7 (Timer 2)
interrupt_level_8	0x18	24	Asynchronous interrupt 8 (Communication)
interrupt_level_9	0x19	23	Asynchronous interrupt 9 (Parallel I/O 2)
interrupt_level_10	0x1A	22	Asynchronous interrupt 10 (Timer 3)
interrupt_level_11	0x1B	21	Asynchronous interrupt 11 (GNSS high)
interrupt_level_12	0x1C	20	Asynchronous interrupt 12 (Parallel I/O 3)
interrupt_level_13	0x1D	19	Asynchronous interrupt 13 (Timer 4)
interrupt_level_14	0x1E	18	Asynchronous interrupt 14 (DSU Trace Buffer)
interrupt_level_15	0x1F	17	Asynchronous interrupt 15 (NMI)
trap_instruction	0x80 - 0xff	16	Software trap instruction (TA)

Table 4-5 Trap allocation and priority

4.4 Watch-points

The integer unit contains four hardware watch-points. Each watch-point consists of a pair of application-specific registers (%asr24/25, %asr26/27, %asr28/29 and %asr30/31) registers; one with the break address and one with a mask:

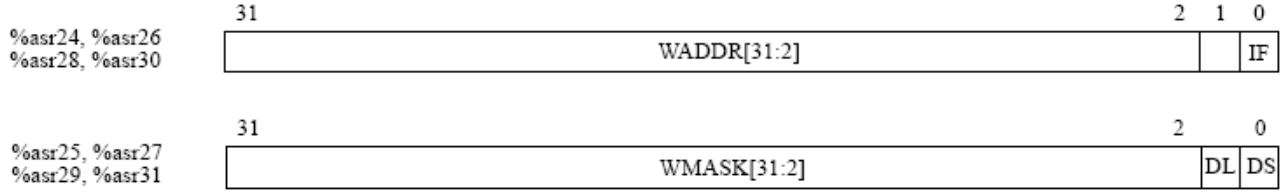


Figure 4-3 Watch Point Registers

Any binary aligned address range can be watched - the range is defined by the WADDR field, masked by the WMASK field (WMASK[x] = 1 enables comparison). On a watch-point hit, trap 0x0B is generated. By setting the IF, DL and DS bits, a hit can be generated on instruction fetch, data load or data store. Clearing these three bits will effectively disable the watch-point function. After reset all three bits (IF, DL, DS) are zero.

4.5 Power Down Register

The processor can be powered-down by writing an arbitrary value to the *PowerDown* register. Power-down mode will be entered on the next load or store instruction. To enter power-down mode immediately, a store to the *PowerDown*

register should be performed immediately followed by a 'dummy' load. During power-down mode, the integer unit will effectively be halted. The power-down mode will be terminated (and the integer unit re-enabled) when an unmasked interrupt with higher level than the current processor interrupt level (PIL) becomes pending. All other functions and peripherals operate as nominal during the power-down mode. A suitable power-down routine could be:

```
struct pwd_reg_type { volatile int pwd; };
power_down()
{
struct pwd_reg_type *lreg = (struct pwd_reg_type *) 0x80000018;
while (1) lreg->pwd = lreg->pwd;
}
```

In assembly, a suitable sequence could be:

```
power_down:
set 0x80000000, %l3
st %g0, [%l3 + 0x18]
ba power_down
ld [%l3 + 0x18], %g0
```

Note: The LEON2 error „Power-down causes lock-up of processor", documented in the AT697 errata sheet is corrected in this LEON2 version.

4.6 Leon Configuration Register

Since LEON is synthesized from an extensively configurable VHDL model, the *LeonConfig* register (read-only) is used to indicate which options were enabled during synthesis.

4.7 Fault tolerant Features

4.7.1 Register file protection

To prevent erroneous operations from SEU errors, the main register file is implemented with hard flip flops.

4.7.2 External memory EDAC

The on-chip memory EDAC can correct one error and detect two errors in a 32-bit word. For each word, a 7-bit checksum is generated according to the equations below. Correction is done on-the-fly and no timing penalty occurs during correction. If an un-correctable error (double-error) is detected, an error response is generated on the AHB bus, which can lead to a memory exception in the processor, and interrupt 1 is generated. If a correctable error occurs, no error response is generated but the event is registered in the AHB failing address and memory status register, and interrupt 1 is generated. This interrupt can be used to scrub the failing memory location. The EDAC can be used during access to PROM or SRAM (0x4000_0000 - 0x5FFF_FFFF) areas by setting the corresponding EDAC enable bits in the *MCFG3* register. The equations below show how the EDAC check bits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
Not(CB2) = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
Not(CB3) = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

If the memory is configured in 32-bit mode, the EDAC check bit bus (CB[6:0]) is used, the bits are stored in a separate memory space, which is not seen by the user and which does not reduce the available memory in the 32-bit space.

If the memory is configured in 8-bit mode, the EDAC check bit bus (CB[6:0]) is not used but it is still possible to use EDAC protection. Data is always accessed as words (4 bytes at a time) and the corresponding check bits are located at the address acquired by inverting the word address (ADDRESS[27:2]) and using it as a byte address. The same chip-select is kept active. A word written as four bytes to addresses 0, 1, 2, 3 will have its check bits at address 0x0FFFFFFF, addresses 4, 5, 6, 7 at 0x0FFFFFFE and so on. All the bits up to the maximum bank size will be inverted while the same chip-select is always asserted. This way all the bank size can be supported and no memory will be unused (except for a maximum of 4 Byte in the gap between the data and check bit area). The 8-bit mode applies to RAM and PROM. Only byte-writes should be performed to ROM with EDAC enabled. In this case, only the corresponding byte will be written.

The operation of the EDAC can be tested through the *MCFG3* register. If the *WB* (write bypass) bit is set, the value in the *TCB* field will replace the normal check bits during memory write cycles. If the *RB* (read bypass) is set, the memory check bits of the loaded data will be stored in the *TCB* field during memory read cycles.

Note: when the EDAC is enabled, the *ReadModifyWriteEn* bit in *MCFG2* register must be set.

Access to 32-bit wide EDAC protected memory occurs with no performance degradation. Accesses to 8-bit wide EDAC protected memory needs 25% more time (The checksum has to be written sequentially).

Note: when EDAC is enabled and 8bit PROM mode is used, the EDAC is only supported for read accesses.

When writing into 8bit PROM the EDAC checksum has to be calculated and has to be written in the correct address by hand.

4.7.3 Cache memory protection

Error detection of cache tags and data is implemented using two parity bits per tag and per 4-byte data sub-block. The tag parity is generated from the tag value and the valid bits. The data sub-block parity is derived from the sub-block data. The parity bits are written simultaneously with the associated tag or sub-block and checked on each access. The two parity bits correspond to the parity of odd and even data (tag) bits.

If a tag parity error is detected during a cache access, a cache miss will be generated and the tag (and data) will be automatically updated. All valid bits except the one corresponding to the newly loaded data will be cleared. If a data sub-block parity error occurs, a miss will also be generated but only the failed sub-block will be updated with data from main memory.

4.8 Multiplication/Division Instructions

Full support for SPARC V8 divide instructions is provided (SDIV/UDIV/SDIVCC/UDIVCC). The divide instructions perform a 64-by-32 bit divide and produce a 32-bit result. Rounding and overflow detection is performed as defined in the SPARC V8 standard.

The LEON processor supports the SPARC integer multiply instructions UMUL, SMUL, UMULCC and SMULCC. These instructions perform a 32x32-bit integer multiply, producing a 64-bit result. SMUL and SMULCC perform signed multiply while UMUL and UMULCC perform unsigned multiply. UMULCC and SMULCC also set the condition codes to reflect the result.

4.9 Cache Sub-System

Separate instruction (32 KB) and data (16 KB) caches are provided. The instruction cache uses streaming during line-refill to minimize refill latency. The data cache uses write through policy and implements a double-word write-buffer. The data cache also performs bus-snooping on the AHB bus.

Cache configuration

Instruction Cache	32 Kbyte
Associativity	4
Cache Line Size	32 Byte (8 dwords)
Replacement Algorithm	Pseudo Random
Locking	disabled
Data Cache	16 Kbyte
Associativity	2
Cache Line Size	16 Byte (4 dwords)
Replacement Algorithm	Pseudo Random
Locking	disabled

Table 4-6 Cache configuration

The LEON core inside AGGA-4 implements Harvard architecture with separate instruction and data buses, connected to two independent cache controllers. In addition to the address, a SPARC processor also generates an 8-bit address space identifier (ASI), providing up to 256 separate, 32-bit address spaces. During normal operation, the LEON core accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC standard. Using the LDA/STA instructions, alternative address spaces can be accessed. The table shows the ASI usage for LEON. Only ASI[3:0] is used for the mapping, ASI[7:4] has no influence on operation.

ASI	Usage
0x0, 0x1, 0x2, 0x3	Forced cache miss (replace if cacheable)
0x4, 0x7	Forced cache miss (update on hit)
0x5	Flush instruction cache
0x6	Flush data cache
0x8, 0x9, 0xA, 0xB	Normal cached access (replace if cacheable)
0xC	Instruction cache tags
0xD	Instruction cache data
0xE	Data cache tags
0xF	Data cache data

Figure 4-4 ASI usage

Access to ASI 4 and 7 will force a cache miss, and update the cache if the data was previously cached. Access with ASI 0 - 3 will force a cache miss, update the cache if the data was previously cached, or allocated a new line if the data was not in the cache and the address refers to a cacheable location. The cacheable areas are by default the prom and ram areas:

Address range	Mapping	Module
0x0000_0000 - 0x1FFF_FFFF	PROM	cacheable
0x2000_0000 - 0x3FFF_FFFF	IO	Non-cacheable
0x4000_0000 - 0x5FFF_FFFF	SRAM	cacheable
0x6000_0000 - 0x7FFF_FFFF	RAMS_N[4] memory	Non-cacheable
0x8000_0000 - 0xFFFF_FFFF	Internal (AHB)	Non-cacheable

Table 4-7 Default cache table

The memory behind chip select RAMS_N[4] is not cacheable. This offers the possibility to support shared memory or a dual-port memory that is connected to that chip select.

The operation of the instruction and data caches is controlled through a common Cache Control Register (*CacheCtrl*). Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

If the Data Cache Freeze or the Instruction Cache Freeze bit is set, the corresponding cache will be frozen when an asynchronous interrupt is taken. The device is affected by the ESA Alert EA-2012-EEE-17-A. Before using cache freezing the Alert has to be read and understood. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt. If a cache has been frozen by an interrupt, it can only be enabled again by enabling it in the Cache Control Register. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

Note: Always flush the data/instruction cache before enabling.

4.9.1 Instruction Cache

4.9.1.1 Operation

If instruction burst fetch is enabled in the cache control register (*CacheCtrl*) the cache line is filled from main memory starting at the missed address and until the end of the line. At the same time, the instructions are forwarded to the IU (streaming). If the IU cannot accept the streamed instructions due to internal dependencies or multi-cycle instruction, the IU is halted until the line fill is completed. If the IU executes a control transfer instruction (branch/CALL/JMPL/RETT/TRAP) during the line fill, the line fill will be terminated on the next fetch. If instruction burst fetch is enabled, incremental AHB bursts will be used on consecutive instruction fetches, even when the cache is disabled. In this case, the fetched instructions are only forwarded to the IU, the cache is not updated and a staggered burst will be observed.

If a memory access error occurs during a line fill with the IU halted, the corresponding valid bit in the cache tag will not be set. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address. If the error remains, an instruction access error trap (tt=0x1) will be generated.

4.9.1.2 Instruction cache tag

A instruction cache tag entry consists of several fields as shown in Figure 4-5.

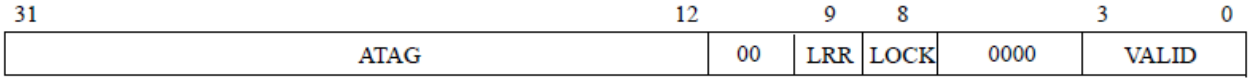


Figure 4-5: Intruction Cache Tag

Field Definitions:

- [31:10]: Address Tag (ATAG) - Contains the tag address of the cache line.
- [9]: LRR - Used by LRR algorithm to store replacement history. '0' as PRR is used.
- [8]: LOCK - Locks a cache line when set. 0 as instruction cache locking is not enabled.
- [7:0]: Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits is set when a sub-block is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset. A FLUSH instruction will clear all valid bits. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and so on.

4.9.2 Data Cache

4.9.2.1 Operation

Each line has a cache tag associated with it consisting of a tag field, valid field with one valid bit for each 4-byte sub-block and optional lock and LRR bits. On a data cache read-miss to a cachable location 4 bytes of data are loaded into the cache from main memory. The write policy for stores is write-through with no-allocate on write-miss. In a multi-set configuration a line to be replaced on read-miss is chosen according to the replacement policy. If a memory access error occurs during a data load, the corresponding valid bit in the cache tag will not be set and a data access error trap (tt=0x9) will be generated

4.9.2.2 Write buffer

The data cache contains a write buffer able to hold a single 8,16,32, or 64-bit write transaction. For half-word or byte stores, the stored data is replicated into proper byte alignment for writing to a word-addressed device. The write is processed in the background so the processor pipeline can keep executing while the write is being processed. However, any following instruction that requires bus access (hence, NOT a load cache-hit) will cause the write buffer to be emptied prior fetching new data to avoid generation of any stale data.

Since the processor executes parallel to the write buffer, a write error will not cause an exception synchronous to the excecution of the store instruction. Depending on memory and cache activity, the write cycle may not occur until several clock cycles after the store instructions has completed. If a write error occurs, the currently executing instruction will take trap 0x2b.

Note: the 0x2b trap handler should flush the data cache, since a write hit would update the cache while the memory would keep the old value due the write error.

Due to the write buffer latency, depending on memory and cache activity, a write operation may not become effective until several cycles after the store instruction has completed in the processor. For APB register accesses where the exact time of writing is relevant, it may be advised to force a flush of the write buffer.

This can be done by adding a dummy memory access, e.g. a load or store (usually faster) to a dummy address in a non-cacheable area. Since dummy instructions may have a performance impact and the next store will always flush out the previous store, it is sufficient to do this at the end of a sequence of stores (e.g. configuring a GNSS function or an interface).

Note that in any case the correct order of store and load transactions and the coherency of data in the cache is guaranteed by the hardware.

4.9.2.3 Data Cache Snooping

The data cache controller will monitor write accesses to the AHB bus performed by other AHB masters (DMA). When a write access is performed to a cacheable memory location, the corresponding cacheline will be invalidated in the data cache if present.

Note: See also Known Issues chapter 8.4.11

4.9.2.4 Data Cache Tag

A data cache tag entry consists of several fields as shown in **Figure 4-6**.

Figure 4-6: Data Cache Tag

Field Definitions:

- [31:10]: Address Tag (ATAG) - Contains the address of the data held in the cache line.
- [9]: LRR - Used by LRR algorithm to store replacement history. '0' as PRR is used.
- [8]: LOCK - Locks a cache line when set. '0' as instruction cache locking is not enabled.
- [3:0]: Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits is set when a sub-block is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and V[3] to address 3.

4.9.3 Cache Flushing

The instruction and data cache is flushed by executing the FLUSH instruction, setting the FI bit in the cache control register (*CacheCtrl*), or by writing to any location with ASI=0x5. The flushing will take one cycle per cache line and set during which the IU will not be halted, but during which the instruction cache will be disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register. Selective flushing of individual cache lines is not supported.

4.9.4 Diagnostic Cache access

Tags and data in the instruction and data cache can be accessed through ASI address space 0xC, 0xD, 0xE and 0xF by executing LDA and STA instructions. Address bits making up the cache offset will be used to index the tag to be accessed while the least significant bits of the bits making up the address tag will be used to index the cache set.

Diagnostic read of tags is possible by executing an LDA instruction with ASI=0xC for instruction cache tags and ASI=0xE for data cache tags. A cache line and set are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. Similarly, the data sub-blocks may be read by executing an LDA instruction with ASI=0xD for instruction cache data and ASI=0xF for data cache data. The sub-block to be read in the indexed cache line and set is selected by A[4:2].

The tags can be directly written by executing a STA instruction with ASI=0xC for the instruction cache tags and ASI=0xE for the data cache tags. The cache line and set are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. D[31:10] is written into the ATAG field (see above) and the valid bits are written with the D[7:0] of the write data. Bit D[9] is written into the LRR bit (if enabled) and D[8] is written into the lock bit (if enabled). The data sub-blocks can be directly written by executing a STA instruction with ASI=0xD for the instruction cache data and ASI=0xF for the data cache data. The sub-block to be read in the indexed cache line and set is selected by A[4:2].

Note that diagnostic access to the cache is not possible during a FLUSH operation and will cause a data exception (trap=0x09) if attempted.

4.9.5 Cache Freeze Alert

ESA Alert EA-2012-EEE-17-A details the anomaly in the Leon2FT ipcore that also is applicable for the AGGA-4. This alert also shows a workaround for this anomaly.

Note: ESA Alerts can be found at https://alerts.esa.int

4.10 Byte/Half Word Accesses

Single bytes and half words are written to memory with AHB HSIZE = '000'/'001' indicating byte/half word size transactions. Addresses for theses transactions can be byte or half word aligned. (Byte = 0x1, 0x2, 0x3, … half word = 0x0, 0x2, 0x4, …). Bytes/half words are duplicated during the write process.

Example:

```
stb 0x23, %address
```

The example above would generate the word 0x23232323 on the AHB Bus.

```
sth 0x1234, %address
```

The example above would generate the word 0x12341234 on the AHB Bus.

If cacheable memory is read using a single byte/half word access, a whole word is transferred over the AHB bus. Byte accessing is done inside the processor module. Subsequent reads of bytes/half words in the fetched word are served from the cache.

Byte ordering in memory is big endian. D(31:24) stores address[0x0]; D(23:16) stores address[0x1] and so on (see Figure 4-7 Big Endian).

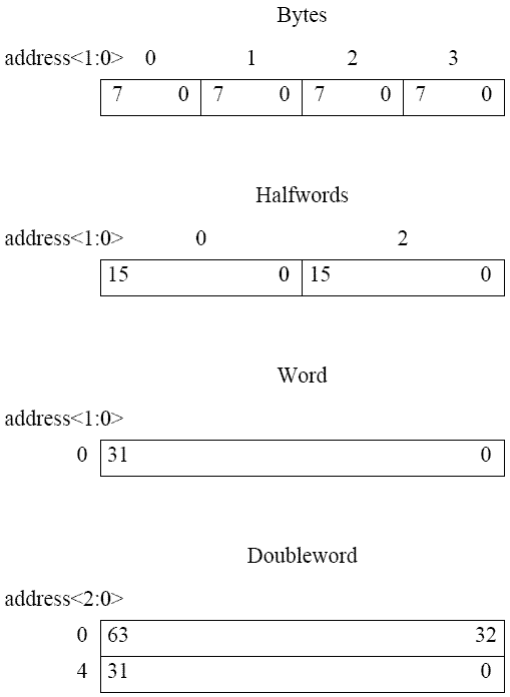


Figure 4-7 Big Endian

Single byte/half word accesses to an EDAC protected memory should not be used in applications with high memory bandwidth demands. Byte/half word writes to an un-initialized memory will return an EDAC error. Single byte writes to a 32 bit wide RAM interface needs 5 clock cycles (compared to 3 when writing a whole word). Single byte writes to an 8 bit wide RAM interface need 25 clock cycles.

Single byte/half word accesses **are not allowed for APB Slaves**. The APB Bridge does not take the HSIZE signal into account. Therefore single byte writes like:

```
stb 0x12, 0x8000_000A0
```

The example above would store the value 0x12121212 into that register and therefore unexpectedly overwrite the other three bytes of this register.

If single bytes/half words are read from an APB device the bridge reads a word from the APB device but will only transfer the addressed byte to the AHB Master. Therefore information may be lost (i.e. if a read to clear mechanism or a FIFO is used).

Note that the memory accesses to the APB area are not cached. Subsequent reads of single bytes will therefore generate a series of accesses to the APB device. However, compilers may try to optimise accesses to hardware (APB) registers, merge them, change their order, or even remove accesses whose read values is never used by the SW. This can have unwanted side effects (e.g. removing a 'clear-on-read'), and therefore in C-code, every access to these hardware registers should use variables / pointers declared as 'volatile'.

4.11 Memory Controller

The external memory bus is controlled by a programmable memory controller. The controller acts as a slave on the AHB bus. The function of the memory controller is programmed through memory configuration registers 1, 2 & 3 (*MCFG1*, *MCFG2* & *MCFG3*) through the APB bus. The memory bus supports three different areas: PROM, SRAM and local I/O. The PROM area can be equipped with PROM/EEPROM or Flash ROM devices. The memory bus can also be configured in 8-bit mode for applications with low memory and performance demands.

Note about the *RWE_N* and *WRITE_N* pins. *WRITE_N* is asserted 0 on all write transactions in all areas (PROM, SRAM, IO) and all bus-width / data width combinations. *RWE_N* are complementary byte write signals, asserted at same cycle, *RWE_N*[0] for *DATA*[31:24], *RWE_N*[1] for *DATA*[23:16], *RWE_N*[2] for *DATA*[15:8], *RWE_N*[3] for *DATA*[7:0]. They are asserted as follows:

- With 8-bit bus width (*PROMWidth* or *RAMWidth* = 00)., they are always "1110", indicating that DATA[31:24] holds the valid data
- With 32-bit SRAM (*RAMWidth* = "10") and *ReadModifyWriteEn* = 1, they are always "0000"
- In all other situations (writes to 32-bit PROM, IO or write to 32-bit RAM with *ReadModifyWriteEn* = 0, they are asserted according to the store width to select 1, 2 or 4 byte lanes

4.11.1 Attaching an external memory controller

To attach an external memory controller, RAMS_N[4] should be used since it allows the cycle time to vary through the use of BRDY_N. In this way, delays can be inserted as shown in [AD-04].

4.11.2 8-bit PROM/EEPROM and SRAM access

To support applications with low memory and performance requirements efficiently, it is not necessary to always have full 32-bit memory banks. The SRAM and PROM areas can be individually configured for 8-bit operation by programming the ROM and RAM size fields in the memory configuration registers. 32-bit accesses to 8-bit memory will be transformed in a burst of four or five, with EDAC enabled, read cycles. During writes, only the necessary bytes will be written.

Note: If EDAC is enabled, all byte writes will generate read-modify-write transactions for a complete word and its EDAC bits. So a single byte write on 8-bit EDAC protected memory will generate 10 memory transactions (5 bytes to read, 5 bytes to write).

Note: EEPROM/Flash based memories operating in 8 bit mode shall only be written using single byte assembler commands (stb), and parity bytes must be written explicitly by SW to the appropriate address (see section 4.7.2). Using half-word/word/d-word assembler commands (sth, st, std) leads to malfunction.

4.11.3 8-bit IO access

Similar to the PROM/RAM areas, the I/O area can also be configured to 8-bits mode. However, the I/O device will NOT be accessed by multiple 8 bits accesses as the memory areas, but only with one single access just as in 32-bit mode. To accesses an I/O device on an 8-bit bus, LDUB/STB should be used.

Note: It is not possible to access the IO space with the DSU or the SpaceWire modules if 8bit IO access is chosen. These masters are not capable of byte/half word transactions. They support 32-bit accesses only.

4.12 Interrupt Controller

4.12.1 Primary Interrupt Controller (PIC)

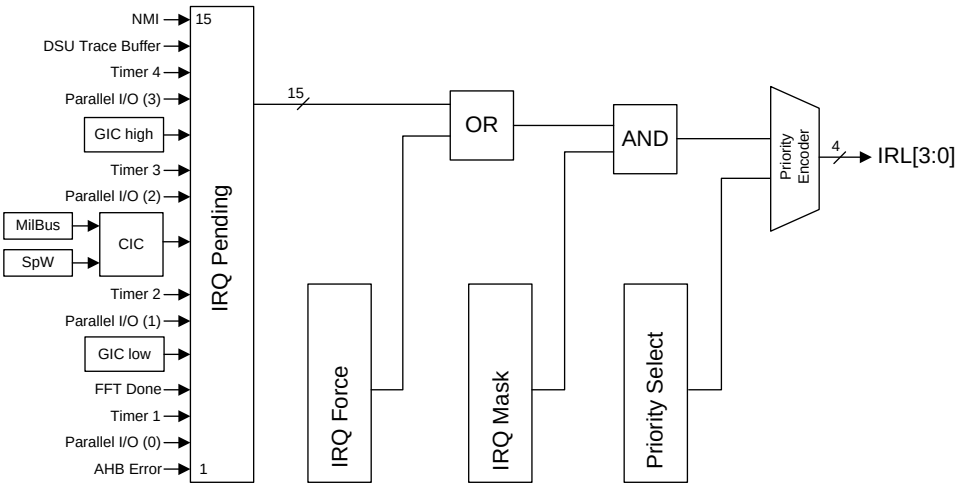


Figure 4-8: Primary Interrupt Controller (PIC)

When an interrupt is generated, the corresponding bit is set in the *PrimIntPending* register. The pending bits are ANDed with the interrupt mask which is contained in the *PrimIntMaskAndPrio* register and then forwarded to the priority selector. Each interrupt can be assigned to one of two levels as programmed in the *PrimIntMaskAndPrio* register. Level 1 has higher priority than level 0. The interrupts are prioritized within each level, with interrupt 15 having the highest priority and interrupt 1 the lowest. The highest interrupt from level 1 will be forwarded to the IU - if no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded.

The NMI can not be masked through the mask register; it also has always priority level 1. When the IU acknowledges the interrupt, the corresponding pending bit will automatically be cleared.

Interrupts can also be forced by setting a bit in the *PrimIntForce* register. In this case, the IU acknowledgement will clear the force bit rather than the pending bit.

By setting the corresponding clear bit in the *PrimIntClear* register the respective pending bit in the PIC will be cleared. The IU acknowledgement will not be able to clear the interrupts that are not stored inside the primary interrupt controller (GIC high, CIC, GIC low). The interrupt handling routine has to make sure that the corresponding interrupt is cleared on its source. Also if a secondary interrupt controller interrupt shall be forced, it has to be forced at the source.

The GIC low/high pending bits in the primary interrupt controller are directly connected to the interrupt vector queues in the GIC. That means if the GIC interrupt vector queue is not empty, the corresponding GIC low/high pending bit in the PIC will be on. If the GIC interrupt vector queue is empty the corresponding GIC low/high pending bit in the PIC is not active.

The pending bits inside the CIC are OR'ed together and are directly connected to the CIC pending bit in the PIC. Therefore if any pending bit inside the CIC will be active, the CIC pending bit in the PIC will be on.

4.12.1.1 Pending Interrupt Cleared by Ticc

AGGA4 is affected by the following issue (ESA IPN #384):

An interrupt (tt = 0x11 – 0x1F), pending in the *PrimIntPending* or the *PrimIntForce* registers is systematically cleared by a software trap (Ticc) which shares the same lower 6 bit [5:0] in its trap type. As a side effect, if cache-freeze on interrupt is activated by setting *CacheCtrl.DF* and/or *CacheCtrl.IF*, the cache will be frozen erroneously. The anomaly only happens when the trap indicated in the pending or force irq register shares the lower 6bits with the software trap number used in the Ticc instruction.

The following table summarises which software traps would clear which pending interrupt:

Pending interrupt (value of 'tt')	SW Trap that clears interrupt (value of 'tt')
0x11	0x91, 0xD1
0x12	0x92, 0xD2
0x13	0x93, 0xD3
0x14	0x94, 0xD4
0x15	0x95, 0xD5
0x16	0x96, 0xD6
0x17	0x97, 0xD7
0x18	0x98, 0xD8
0x19	0x99, 0xD9
0x1A	0x9A, 0xDA
0x1B	0x9B, 0xDB
0x1C	0x9C, 0xDC
0x1D	0x9D, 0xDD
0x1E	0x9E, 0xDE
0x1F	0x9F, 0xDF

Table 4-8: Pending cleared by Ticc

The following **software workaround** can be adopted:

No Ticc instruction shall use trap numbers which result in a tt value of 0x9- or 0xD

4.12.2 GNSS Interrupt Controller (GIC)

The first secondary interrupt controller called *GNSS Interrupt Controller (GIC)* is located inside the GNSS clock domain. Two interrupt signals (transporting the priority) are provided to the primary interrupt controller.

The GNSS interrupt controller provides priorisation of the trigger events. The following events are served by the GNSS interrupt controller:

Trigger Event	Number
GNSS Channel Integration Epoch elapsed	36
GNSS Pulse Per Second Epoch elapsed	1
GNSS Antenna Switch Epoch elapsed	1
GNSS Measurement Epoch elapsed	1
GNSS Power Level Detector I/Q elapsed	1
GNSS Power Level Detector 5I elapsed	1
GNSS DMA error occurred	1

Table 4-9: GNSS Interrupts

The implementation of the GNSS Interrupt Controller is similar to the secondary interrupt controller of the LEON-IP. An interrupt priority register is added to provide the possibility to assign each GNSS interrupt to one of two interrupt levels. Also a queue mechanism is implemented, which can be used to store the interrupts in the order of occurrence. The timestamp (IMT value) of an interrupt is stored inside the GNSS registers. The GNSS Interrupt Controller is depicted in Figure 3-4.

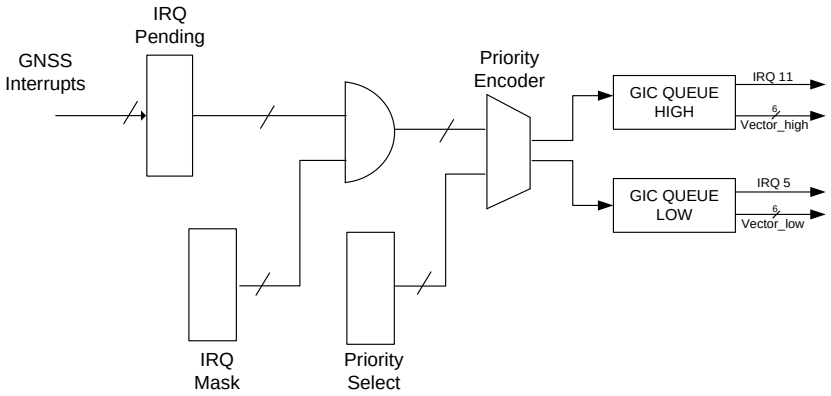


Figure 4-9: GNSS Interrupt Controller

If an interrupt is enabled trough the mask (*GIC_Mask0/1*), the interrupt is priorized via the *GIC_Prio_0/1* registers and queued in either the *GIC_QueueHigh* or *GIC_QueueLow* queue according to their priority. The 6bit queue entry identifies the interrupt source (e.g. channel, ME, PPS, etc). Also the queue maintains the timely order (first in first out). If an interrupt is registered in the queue it is automatically cleared by the hardware in the pending register (*GIC_Pend0/1*) and can not be cleared by the software. If the user reads the queue entry, also the queue entry will be cleared automatically by the hardware and therefore must not be cleared by the software. The queue has a depth of 64 entries. If a queue is empty a read on it will return the value 0x3F. If a queue is full, the corresponding pending bits will remain in the pending register (*GIC_Pend0/1*) until the queue has a free entry. Once there is a free entry, the pending bit is registered in the queue and automatically cleared in the pending register (*GIC_Pend0/1*). However in this case (full queue) the user has to take into account that the interrupts may not be entered in the correct time order once the queue gets free again. If the PIC signals a GNSS low/high interrupt it is recommended to read the corresponding GIC queue in the interrupt service routine until the queue is empty and thus do a batch processing of interrupt events.

If an interrupt is disabled through the mask (*GIC_Mask0/1*), the pending bit is not cleared automatically. The user has to clear it by setting the corresponding bit in the interrupt clear (*GIC_Clear0/1*) register.

This allows treating some events by polling (by masking the interrupt and reading the pending bit), while other events can be handled by an interrupt handler (by unmasking the interrupt and reading the queue).

By reading the register *GIC_QueueStatus* it is also possible to get the current fill status of the *GIC_QueueHigh* and *GIC_QueueLow* as well as the information whether one of the queues had an overflow somewhere in the past.

Note: The internal Pending register in the AGGA-4 has 42bits. Since it is not possible for the software to read all 42bits at a time, a special mechanism has been implemented. If the software reads the *GIC_Pend0* register it reads the lower 32bits of the 42bit Pending register. The upper 10bits of the 42bit Pending register are stored by the hardware into a shadow register. If the software now reads the *GIC_Pend1* register, it will read the content of the shadow register and gets the residual 10bits of the latched 42bit Pending register. Therefore in order to read the current status of the upper 10bits of the 42bit Pending register, the software always has to read first *GIC_Pend0* and then *GIC_Pend1*.

Note: If a GNSS channel has enabled GNSS DMA, the Integration Epoch (IE) is not directly signalled from the channel to the GIC, but is delayed by the DMA machine until the DMA transfer of that channel is finished. That way it is ensured that the user can be sure, once he gets an IE interrupt, that the DMA is already done. If a GNSS channel has disabled GNSS DMA the Integration Epoch (IE) is directly signalled to the GIC. Note that only the notification to the user is delayed, not the IE mechanism itself. Therefore the IE observables do of course carry the correct content.

Note: After reset of the GNSS core the GIC queue low and the GIC queue high can have one arbitrary entry. Therefore if the user wants to make use of these queues, it is recommended to read the GIC queues after a GNSS reset until 0x3F (queue empty) is readback.

4.12.3 Communication Interrupt Controller (CIC)

The second secondary interrupt controller is the Communication Interrupt Controller (CIC).

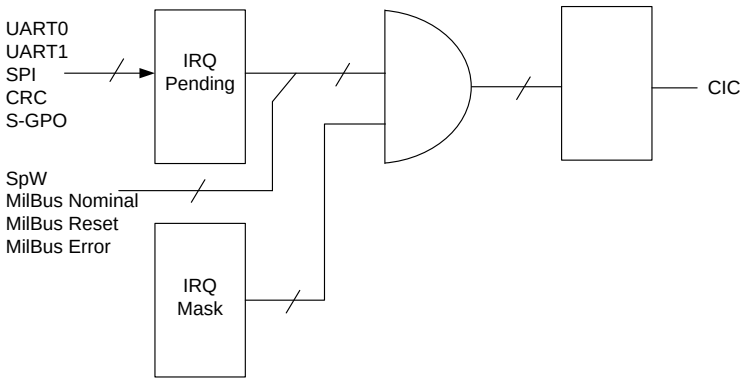


Figure 4-10: Communication Interrupt Controller

The implementation of the Communication Interrupt Controller is similar to the secondary interrupt controller of the LEON-IP. When a communication interrupt is generated, the corresponding bit is set in the *CIC_Pending* register. The pending bits are ANDed with the *CIC_Mask* register and then forwarded to the PIC.

By setting the corresponding clear bit in the *CIC_Clear* register the respective pending bit in the CIC will be cleared. However the SpaceWire Module and the Mil-Bus IP Core do have their own interrupt controllers. Therefore the interrupts of those modules are just mirrored in the CIC pending register, but they are not stored there. They are stored in their own interrupt controller. Therefore the SpaceWire and Mil-Bus interrupts have to be cleared in the SpaceWire respectively Mil-Bus interrupt controller.

The following events are served by the Communication Interrupt Controller:

Trigger Event	Number
Mil-Bus Nominal (C53It)	1
Mil-Bus Reset (C53Rst)	1
Mil-Bus Error (C53Err)	1
Spacewire	1
UART-1 (RxDone, TxDone, Error)	3
UART-0 (RxDone, TxDone, Error)	3
SPI	1
CRC Ready	1
SGPO Overrun	1

Table 4-10: CIC Interrupts

4.13 Timer Unit

The timer unit implements four 32-bit timers and one 10-bit shared prescaler (see Figure 4-11). The prescaler is clocked by the Leon clock and decremented on each clock cycle. When the prescaler underflows, it is reloaded from the prescaler reload register (*TimerPrescaleReload*) and a timer tick is generated for the four timers. The effective division rate is equal to prescaler reload register value + 1. The current value of the prescaler can always be accessed by reading the *TimerPrescaleCounter* register.

The operation of the timers is controlled through the timer control register (*TimerN_Ctrl*). A timer is enabled by setting the *EN* bit in the *TimerN_Ctrl* register. Note that the prescaler and timer reload values must be set to a legal value before enabling the timers. The timer value is then decremented each time the prescaler generates a timer tick. When a timer underflows, it will automatically be reloaded with the value of the timer reload register (*TimerN_Reload*) if the *RL*

bit in the *TimerN_Ctrl* is set, otherwise it will stop at 0xffffffff and reset the *EN* bit. The effective division rate is equal to timer reload register value + 1. An interrupt will be generated after each underflow.

The timer can be reloaded with the value in the reload register at any time by writing a "1" to the *LD* bit in the *TimerN_Ctrl*.

The current value of any of the four timers can be accessed by reading the *TimerN_Counter* register.

To minimize complexity, the four timers share the same prescaler and decrementer as shown in Figure 4-11. The minimum allowed prescaler division factor is 8 (*TimerPrescaleReload* needs to be set to 7 at least before enabling any of the timers). There is no restriction to the value programmed in the *TimerN_Reload* registers (0..0xFFFF).

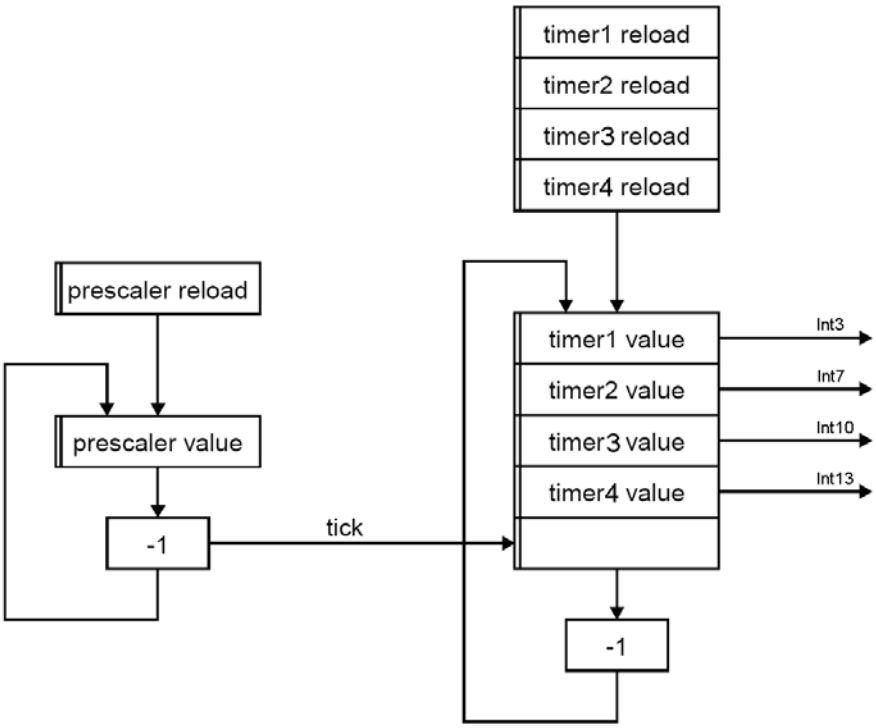


Figure 4-11 Timer Unit

4.14 Write Protection Unit

4.14.1 Overview

Write protection is provided to protect the RAM area against accidental over-writing. It is implemented with two methods: the address/mask method as implemented in the original LEON-2 model, and an extended version using start/end addressing. Both methods can be used simultaneously. Therefore in total AGGA-4 has 4 Write Protect Units. After reset write protection is disabled for all units.

Note: The APB registers (0x8…) and the DSU area (0x9…) can not be protected with the Write Protection Unit. They are always accessible in user and superuser mode.

The I/O area can be protected by setting the *IOEnable* bit in the *MCFG1* register to zero.

The PROM area can be protected by setting the *PROMWriteEn* bit in the *MCFG1* register to zero.

4.14.2 Address/Mask Write Protection

The address/mask write protection is implemented with two block protect units capable of disabling or enabling write access to a binary aligned memory block in the range of 32 Kbyte - 1 Gbyte. Block protect unit 1 is controlled by *WriteProtect1*, block protect unit 2 is controlled by *WriteProtect2*. The units operate as follows: on each write access to

RAM, address bits (29:15) are XORed with the *TAG* field in the control register, and ANDed with the *MASK* field. A write protection hit is generated if the result is equal to zero, and the corresponding unit is enabled in block protect mode (*BP* = 1) or if the results is not zero and the unit is enabled in segment mode (*BP* = 0).

Note: The ROM area can be write protected by clearing the *PROMWriteEn* bit in the *MCFG1* register.

4.14.3 Start/End Address Write Protection

The start/end address write protect scheme contains two identical units that compare the AHB write address against a start and an end address. They can be controlled via the registers *WriteProtectStartAddressN* and *WriteProtectEndAddressN*, where N can be either 3 for Write Protect Unit 3 and 4 for Write Protect Unit 4. If operated in block protect mode (*BP* = 1) in the *WriteProtectStartAddressN* register and the AHB write address is equal or higher than the start address and lower or equal to the end address, a write protect hit is generated. If operated in segment mode (*BP* = 0) in the *WriteProtectStartAddressN* register, a write protect hit is generated when the write address is lower than the start address, or higher than the end address.

The write protection can be enabled by setting the *UM* bit to "1" in user mode or by setting the *SU* bit to "1" in superuser mode.

Note: All AHB Masters, except for the CPU and the Mil-Bus use the HPROT value "0011" meaning that they are "superuser". The Mil-Bus uses the HPROT value "0000" meaning that it is "user". The CPU can be both.

4.14.4 Generation of Write Protection

The results from the two write protection schemes are combined together according to the following scheme:

- If all enabled units operate in block protect mode, then a write protect error will be generated if any of the enabled units signal a write protection hit.
- If at least one of the enabled units operates in segment mode, then a write protect error will be generated only if all units operating in segment mode signal a write protection hit.

A write protection error will result in that the AHB write cycle is ended with an AHB error response and the data is not written to the memory. The ROM area can be write protected by clearing the *PROMWriteEn* bit in the *MCFG1* register.

"Enabled" in this context means enabled either in user or in supervisor mode or both.

4.15 AHB Status Register

An access triggering an error response on the AHB bus will be registered in two registers; *AHBFailingAddress* and *AHBStatus*. The failing address register will store the address of the access while the status register will store the access and error types. The registers are updated when an error occurs. Once the *AHBFailingAddress* register and the *AHBStatus* register have been updated by the hardware, the *NE* (new error) bit in the *AHBStatus* register is set by the hardware. The two registers are then locked, no new error will be recorded as long as *NE* is set, however the associated interrupts are still raised. After the software has read the *AHBFailingAddress* and the *AHBStatus* it must clear the *NE* bit to re-arm the registers.

The AHB status register is also triggered by correctable one-bit errors in the external SRAM. So if the memory controller identifies a correctable error in the attached memory, the *CE* (correctable) and *NE* (new error) bits are set.

When the *NE* bit is set, an interrupt is generated to inform the processor about the error.

For un-correctable errors the *NE* bit and the corresponding interrupt in the primary interrupt controller has to be cleared.

For correctable errors the failed memory location has to be refreshed by a read-write cycle to the failed address, the *NE* and *CE* bit in the *AHBStatus* register has to be cleared, and finally the corresponding interrupt in the primary interrupt controller has to be cleared.

4.16 Hardware Debug Support Unit

4.16.1 Overview

The LEON processor includes hardware debug support to aid software debugging on target hardware. The support is provided through two modules: a debug support unit (DSU) and a debug communication link (DCL). The DSU can put the processor in debug mode, allowing read/write access to all processor registers and cache memories. The DSU also contains a trace buffer which stores executed instructions or data transfers on the AMBA AHB bus. The debug communications link (UART or SpaceWire) implements a simple read/write protocol.

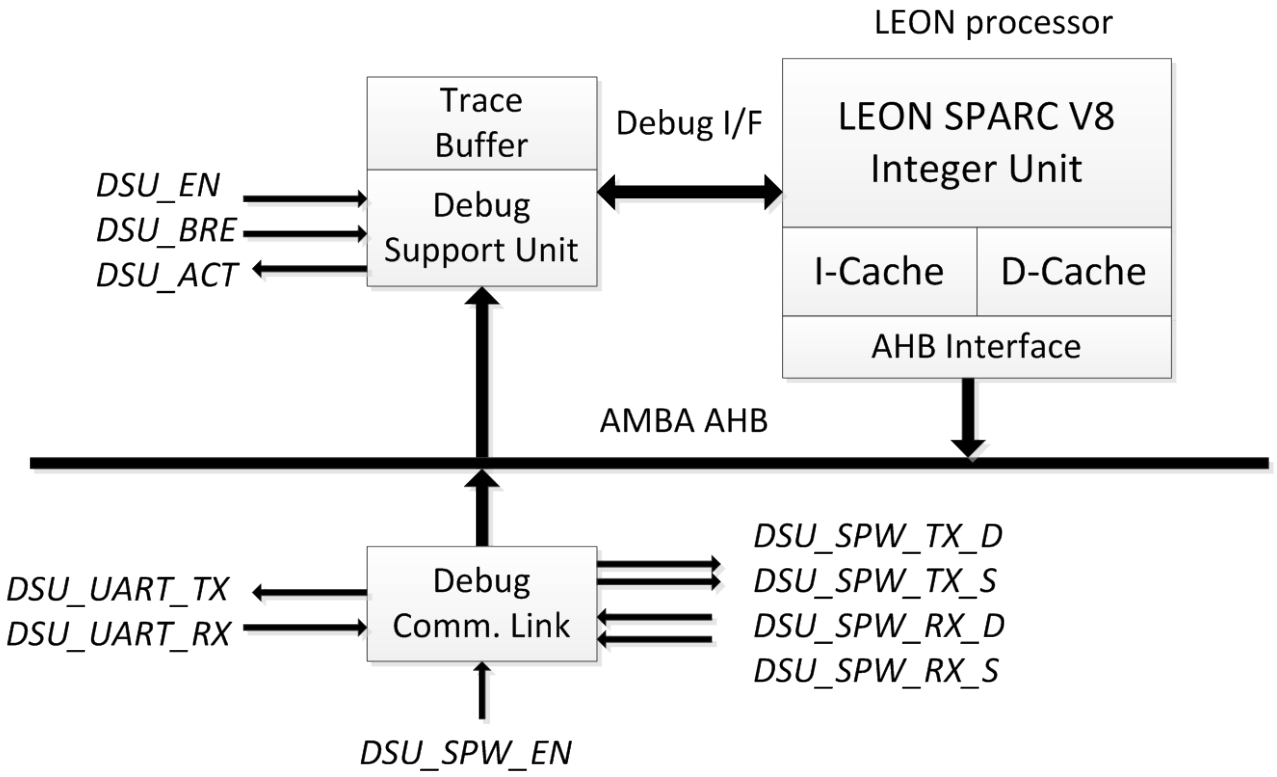


Figure 4-12 DSU Overview

The debug support unit is used control the trace buffer and the processor debug mode. The DSU is attached to the AHB bus as slave, occupying a 2 Mbyte address space. Through this address space, any AHB master can access the processor registers and the contents of the trace buffer. The DSU control registers can be accessed at any time, while the processor registers and caches can only be accessed when the processor has entered debug mode. The trace buffer can be accessed only when tracing is disabled/completed. In debug mode, the processor pipeline is held and the processor is controlled by the DSU. Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1)
- integer unit hardware breakpoint/watch point hit (trap 0xb)
- rising edge of the external break signal (*DSU_BRE* pin)
- setting the break-now (*BN*) bit in the *DSU_Ctrl* register
- a trap that would cause the processor to enter error mode
- occurrence of any, or a selection of traps as defined in the DSU control register
- after a single-step operation
- DSU breakpoint hit

The debug mode can only be entered when the debug support unit is enabled through an external pin (*DSU_EN*). When the debug mode is entered, the following actions are taken:

- PC and nPC are saved in temporary registers (accessible by the debug unit)
- an output signal (*DSU_ACT* pin) is asserted to indicate the debug state
- the timer unit is stopped to freeze the LEON timers

Note: The watchdog continues to run, if enabled.

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the *BN* bit in the *DSU*_Ctrl register or by de-asserting *DSU_EN*. The timer unit will be re-enabled and execution will continue from the saved PC and nPC. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be reset and the processor restarted at any address.

4.16.2 DSU Memory Map

The DSU memory map can be seen in the programming section (chapter 7.3)

Note: The DSU memory addresses from 0x9000_0000 to 0x9FFF_FFFC can only be accessed if DSU is enabled via DSU_EN pin set to '1'. This is done to avoid accidental (in-flight) activation of the debug functionality (break points etc.).

Note: The trace buffer starts at 0x90010000 and ends at 0x90011FFC and is mirrored 8 times in the memory range 0x90010000 – 0x90020000. If tracing is enabled, the trace buffer cannot be accessed by software.

The IU/FPU registers can be accessed at the following addresses (window = psr.cwp):

- `%on : 0x90020000 + (((window * 64) + 32 + 4*n) mod 512)`
- `%ln : 0x90020000 + (((window * 64) + 64 + 4*n) mod 512)`
- `%in : 0x90020000 + (((window * 64) + 96 + 4*n) mod 512)`
- `%gn : 0x90020200 + 4*n`
- `%fn : 0x90030000 + 4*n`

4.16.3 Trace Buffer

The trace buffer consists of a circular buffer that stores executed instructions or AHB data transfers. A 30-bit TimeTagCounter is also provided and stored in the trace as time tag. The trace buffer operation is controlled through the *DSU_Ctrl* register and the *TraceBufferCtrl* register. When the processor enters debug mode, tracing is suspended. The size of the trace buffer is 512 lines (depth) x 128bit (width) corresponding to 8kbyte.

The trace buffer is 128 bits wide, the information stored is indicated in Table 4-11 and Table 4-12 below:

Bits	Name	Definition
127	AHB breakpoint hit	Set to '1' if a DSU AHB breakpoint hit occurred
126	-	Unused
125:96	TimeTagCounter	The value of the TimeTagCounter
95:92	IRL	Processor interrupt request input
91:88	PIL	Processor interrupt level (psr.pil)
87:80	Trap type	Processor trap type (psr.tt)
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA or HWDATA
31:0	Load/Store address	AHB HADDR

Table 4-11 Trace Buffer Line Definition (AHB Tracing Mode)

Note: See chapter 4.15 for AHB HMASTER and HSIZE explanation. The other AHB signals behave like stated in the AHB Specification (see [RD-06]).

Bits	Name	Definition
127	Instruction breakpoint hit	Set to '1' if a DSU instruction breakpoint hit occurred.
126	Multi-cycle instruction	Set to '1' on the second and third instance of a multi-cycle instruction (LDD, ST or FPOP)
125:96	Time tag	The value of the time tag counter
95:64	Load/Store parameters	Instruction result, Store address or Store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to '1' if traced instruction trapped
32	Processor error mode	Set to '1' if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

Table 4-12 Trace Buffer Line Definition (Instruction Tracing Mode)

During instruction tracing, one instruction is stored per line in the trace buffer with the exception of multi-cycle instructions. Multi-cycle instructions are entered two or three times in the trace buffer. For store instructions, bits [63:32] correspond to the store address on the first entry and to the stored data on the second entry (and third in case of STD). Bit 126 is set on the second and third entry to indicate this. A double load (LDD) is entered twice in the trace buffer, with bits [63:32] containing the loaded data. Multiply and divide instructions are entered twice, but only the last entry contains the result. Bit 126 is set for the second entry. For FPU operation producing a double-precision result, the first entry puts the MSB 32 bits of the results in bit [63:32] while the second entry puts the LSB 32 bits in this field. When a trace is frozen, interrupt 14 (DSU Trace Buffer) is generated.

The DSU time tag counter is incremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode, and restarted when execution is resumed.

The trace buffer can trace executed instructions, transfers on AHB or both (mixed-mode). The trace buffer control register contains two counters that store the address of which location of the trace buffer will be written on next trace. Since the buffer is circular, it actually points to the oldest entry in the buffer. The indexes are automatically incremented after each stored trace entry.

When both instructions and AHB transfers are traced ('mixed mode tracing'), the buffer is divided on two halves. Instructions are stored in the lower half (0x90010000) and AHB transfers in the upper half (0x90011000) of the buffer. The MSB bit of the AHB index counter is then automatically kept high, while the MSB of the instruction index counter is kept low. When the *AF* bit in the TraceBufferCtrl register is set, AHB tracing is stopped when the processor is in debug mode. When *AF* is cleared, tracing continues until the AHB trace enable bits are cleared.

Additionally there is the possibility to apply trace buffer filtering. AHB trace buffer filtering reduces the amount of AHB transactions dumped into the trace buffer and helps debugging the access from specific masters or to specific address areas on the AHB bus. Please note that master and slave filtering is subtractive, an access will be traced only if

it passes both filters. With a setting of *MFILT* = 2 and *SFILT* = 1 for example, only accesses from AHB master #2 to addresses starting with 0x8…. (APB registers) will be traced.

Programming a 0 in both fields disables the trace filtering, all AHB accesses are traced (LEON2 default functionality). Note that regardless of the trace filter settings, AHB tracing also needs to be enabled with the usual configuration bits (bit 0 of the *DSU_Ctrl* register and bit 25 of the *TraceBufferCtrl* register).

Note: The *TraceBufferCtrl* register is not cleared by any reset.

4.16.4 DSU Control Register

The *DSU_Ctrl* register gives many possibilities to influence the DSU behaviour. The details can be found in the programming section (chapter 7.3.1.1)

4.16.5 DSU (Hardware) Breakpoints

The DSU contains two breakpoint registers for matching either AHB addresses or executed processor instructions. A breakpoint hit is typically used to freeze the trace buffer, but can also put the processor in debug mode. Freezing can be delayed by programming the delay counter *DCNT* field in the *DSU_Ctrl* register to a non-zero value. In this case, the *DCNT* value will be decremented (after breakpoint condition) for each additional trace until it reaches zero. Then the trace buffer is frozen. If the *BT* bit in the *DSU_Ctrl* register is set, the DSU will force the processor into debug mode when the trace buffer is frozen. Note that due to pipeline delays, up to 4 additional instructions can be executed before the processor is placed in debug mode.

A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to '1' are compared during breakpoint detection. To break on executed instructions, the *EX* bit in the *AHB_BreakAddressN* register should be set. To break on AHB load or store accesses, the *LD* and/or *ST* bits of the *AHB_MaskN* register should be set.

4.16.6 Instruction (Software) Breakpoints

To insert instruction breakpoints, the breakpoint instruction (ta 1) should be used. This will leave the four IU hardware breakpoints free to be used as data watch points. Since cache snooping is only done on the data cache, the instruction cache must be flushed after the insertion or removal of breakpoints. To minimize the influence on execution, it is enough to clear the corresponding instruction cache tag (which is accessible through the DSU). The DSU hardware breakpoints should only be used to freeze the trace buffer, and not for software debugging since there is a 4-cycle delay from the breakpoint hit before the processor enters the debug mode.

4.16.7 DSU Trap Register

The *DSU_Trap* register is a read-only register that indicates which SPARC trap type that caused the processor to enter debug mode. When debug mode is force by setting the *BN* bit in the *DSU_Ctrl* register, the trap type will be 0xb (hardware watch point trap).

4.16.8 Single Stepping

By writing the *SS* bit and resetting the *BN* bit in the *DSU_Ctrl* register, the processor will resume execution for one instruction and then automatically enter debug mode.

4.16.9 Booting from DSU

By asserting *DSU_EN* and *DSU_BRE* at reset time, the processor will directly enter debug mode without executing any instructions. The system can then be initialized via the DSU communication link, and applications can be downloaded and debugged.

5 Interface Modules

5.1 UART

Two identical DMA capable UARTs are provided for serial communications. The UARTs support data frames with 8 data bits, one optional parity bit and one stop bit. To generate the bit-rate, each UART has a programmable 12-bit clock divider programmable via the *UARTn_Scaler* register. Hardware flow-control is supported through the RTSN/CTSN hand-shake signals.

5.1.1 Transmitter Operation

The transmitter is enabled through the *TXEN* bit in the *UARTn_Ctrl* register.

Communication over UART is done by transferring data packets using DMA. To transmit data the CPU writes the relevant data into the memory and sets the DMA Start Address Pointer *UARTn_Tx_SAP* to the start address of the data. The Start Address Pointer (SAP) will contain the address of the first memory cell to be used. Then the DMA End Address Pointer *UARTn_Tx_EAP* has to be set to the end address of the data. The End Address Pointer (EAP) will contain the address of the last memory cell to be used. Writing EAP starts the I/O activity and the DMA transfer of the data is started. Note that the Start Address Pointer always has to be 32bit aligned. Therefore bit 1:0 of the *UARTn_Tx_SAP* are always zero. The End Address Pointer can be byte aligned. The Current Address Pointer *UARTn_Tx_CAP* always points to the next 32bit address which will be transmitted. By reading the Current Address Pointer during the transfer the software can determine the progress of the transmission if needed.

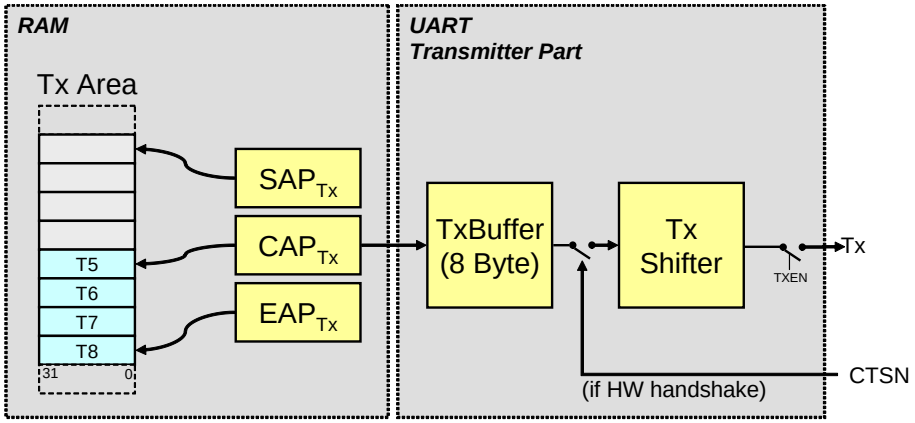


Figure 5-1: UART Transmit Path

The data in the RAM which has to be transmitted is then transferred into the TxBuffer. From there on it is converted to a serial stream for the transmitter serial output pin (*UARTn_TX*). It automatically sends a start bit followed by eight data bits, an optional parity bit, and one stop bit (Figure 5-2).

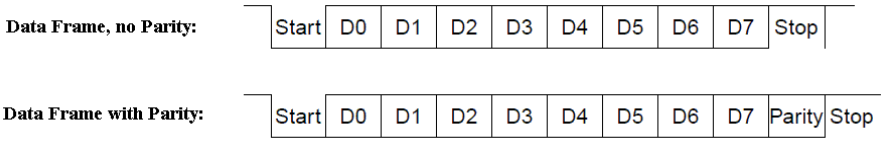


Figure 5-2 UART data frames

Following the transmission of the stop bit, if no new character is available in the TxShifter, the transmitter serial data output remains high and the transmitter shift register empty bit (*TSE*) will be set in the *UARTn_Status* register. Transmission resumes and the *TSE* bit is cleared when a new character is loaded into the TxShifter. When the TxBuffer is empty the *TE* bit in the *UARTn_Status* register is set. If there is data in the TxBuffer the *TE* bit is cleared. If the transmitter is disabled by setting the *TXEN* bit in the *UARTn_Ctrl* register to zero the transmitter will immediately stop

any active transmission including the character currently being shifted out from the transmitter shift register. A transmission can be considered completed when the *TE* and *TSE* bits are both asserted.

If flow control is enabled, the *CTSN* input must be low in order for the character to be transmitted. If it is de-asserted in the middle of a transmission, the character in the shift register is transmitted and the transmitter serial output then remains inactive until *CTSN* is asserted again. If the *CTSN* is connected to receivers *RTSN*, overrun can effectively be prevented.

If all data of the defined transmit area has been transmitted an interrupt (*UARTn_TxDone*) is generated which is visible and controllable in the Communication Interrupt Controller (CIC).

5.1.2 Receiver Operation

The receiver is enabled for data reception through the receiver enable *RXEN* bit in the *UARTn_Ctrl* register.

The receiver looks for a high to low transition of a start bit on the receiver serial data input pin. If a transition is detected, the state of the serial input is sampled half bit clocks later. If the serial input is sampled high the start bit is invalid and the search for a valid start bit continues. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals (at the theoretical centre of the bit) until the proper number of data bits and the parity bit have been assembled and one stop bit has been detected. The serial input is shifted through an 8-bit shift register where all bits have to have the same value before the new value is taken into account, effectively forming a low-pass filter with a cut-off frequency of 1/8 *SysClk*.

During reception, the least significant bit is received first. The data is then transferred from the RxShifter to the RxBuffer and the data ready (*DR*) bit is set in the *UARTn_Status* register as soon as the RxBuffer contains at least one data frame. The parity, framing and overrun error bits are set at the received byte boundary, at the same time as the *DR* bit is set. The data frame is not stored in the RxBuffer if an error is detected. Also, the new error status bits are ORed with the old values before they are stored into the *UARTn_Status* register. Thus, they are not cleared until written to with zeros from the APB bus. If both the RxBuffer and the RxShifter are full when a new start bit is detected, then the character held in the RxShifter will be lost and the overrun bit (*OVR*) will be set in the *UARTn_Status* register. If flow control is enabled, then the *RTSN* will be negated (high) when a valid start bit is detected and the RxBuffer is full. When the RxBuffer is read, the *RTSN* will automatically be reasserted again.

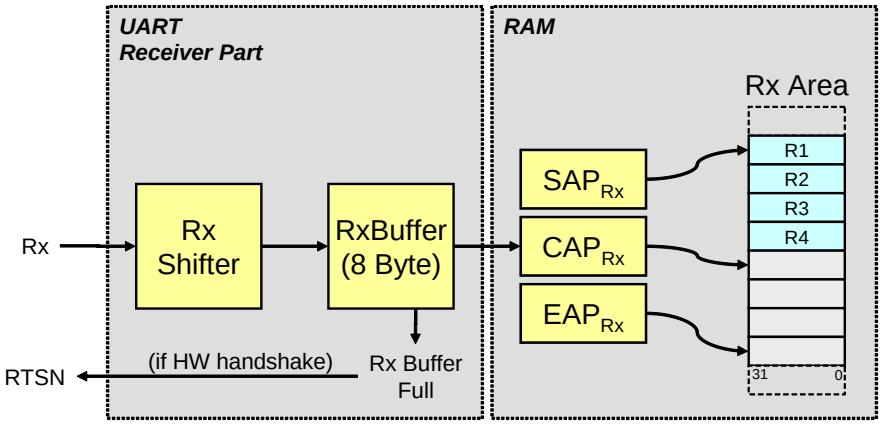


Figure 5-3: UART Receive Path

To receive data the CPU has to configure the DMA Start Address Pointer *UARTn_Rx_SAP* to the start address of the reception RAM area. The Start Address Pointer (SAP) will contain the address of the first memory cell to be used. Then the DMA End Address Pointer *UARTn_Rx_EAP* has to be set to the end address of the receive area. The End Address Pointer (EAP) will contain the address of the last memory cell to be used. Writing EAP starts the I/O activity and the contents of the RxBuffer are written via DMA to the memory. The Current Address Pointer (CAP) will be maintained by the ASIC. It will be set to SAP after EAP has been written. The Current Address Pointer can be used to read the number of bytes which have been received so far (*UARTn_Rx_CAP – UARTn_Rx_SAP*).

While the *UARTn_Rx_CAP* and *UARTn_Rx_EAP* Pointer can address byte wise, the *UARTn_Rx_SAP* has to be 32bit aligned. Therefore bit 1:0 of the written *UARTn_Rx_SAP* are always zero.

Note: The UART DMA always does 32bit writes into memory. Having a look to Figure 5-3 and assuming SAP would be set to `0x40000000` this would mean that the DMA would start writing the following data upon reception on the following addresses:

- After reception of the first byte: `R1 00 00 00` on address `0x40000000`
- After reception of the second byte: `R1 R2 00 00` on address `0x40000000`
- After reception of the thrird byte: `R1 R2 R3 00` on address `0x40000000`
- After reception of the fourth byte: `R1 R2 R3 R4` on address `0x40000000`
- After reception of the fifth byte: `R5 00 00 00` on address `0x40000004`
- ...

Therefore, if the *UARTn_Rx_EAP* Pointer is not 32bit aligned, the user has to make sure that the residual 1..3 bytes after the *UARTn_Rx_EAP* pointer are not used by the software, since they contain invalid data.

If all data of the defined receive area has been received (*UARTn_Rx_CAP* > *UARTn_Rx_EAP*) an interrupt (*UARTn_RxDone*) is generated which is visible and controllable in the Communication Interrupt Controller (CIC).

5.1.3 Baud Rate Generation

Each UART contains a 12-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the *SysClk* and generates a UART tick each time it underflows. It is reloaded with the value of the *ReloadValue* field in the *UARTn_Scaler* register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate minus one. Therefore the correct *ReloadValue* can be determined with the following formula:

$$ReloadValue = \frac{SysClk}{BaudRate * 8} - 1$$

Note: If the baud rate is changed during operation up to 16 bytes can be corrupted until the new baud rate setting will work.

5.1.4 General UART Rules

Here below some additional behaviour which applies for the whole UART:

- A "1" written to *UART_Reset* register will stop all I/O activity on the memory.
- It will be sole responsibility of the user to specify meaningful addresses. The ASIC will not perform any plausibility checks. It is assumed, that SAP ≤ EAP always.

5.1.5 Error Handling

Each UART has five error conditions which can trigger the *UARTn_Error* interrupt in the Communication Interrupt Controller (CIC). The error interrupt can be masked/forced/cleared as a whole in the CIC.

5.1.5.1 Framing Error

If a framing error is detected, the *FER* bit in the *UARTn_Status* register of the corresponding UART is set and the *UARTn_Error* interrupt is generated (if not masked in the CIC).

The *FER* bit can be cleared by writing a "0" to that bit.

5.1.5.2 Parity Error

If a parity error is detected, the *PER* bit in the *UARTn_Status* register of the corresponding UART is set and the *UARTn_Error* interrupt is generated (if not masked in the CIC).

The *PER* bit can be cleared by writing a "0" to that bit.

5.1.5.3 Overrun in Receiver

If the RxShifter and the RxBuffer are full and new data is incoming, an overrun is detected in the UART receiver and the *OVR* bit in the *UARTn_Status* register of the corresponding UART is set and a *UARTn_Error* interrupt is generated (if not masked in the CIC).

The *OVR* bit can be cleared by writing a "0" to that bit.

5.1.5.4 Break Received

If a Break Received is detected, the *BR* bit in the *UARTn_Status* register of the corresponding UART is set and the *UARTn_Error* interrupt is generated (if not masked in the CIC).

The *BR* bit can be cleared by writing a "0" to that bit.

5.1.5.5 Timeout

If data is available in the RxBuffer and is not read within a certain amount of time a timeout signal can be generated, which switches the *RTO* bit in the *UARTn_Status* register to "1" and which generates the *UARTn_Error* interrupt (if not masked in the CIC).

The *RTO* bit is cleared if the data in the RxBuffer is read, or if the timeout feature is disabled or if the UART receiver is disabled.

The timeout feature can be switched on/off by the programming of the Timeout Enable (TOE) bit in the *UARTn_Ctrl* register.

The timeout period can be programmed by the *TimeoutCnt* field in the *UARTn_Scaler* register. The timeout scaler is clocked with the BaudRate which can also be expressed as *SysClk*/(*Scaler*+1)/8. The *TimeoutCnt* field has therefore to be programmed with the desired amount of BaudTicks – 1.

If the timeout feature is enabled, the timeout counter is reloaded with the *TimeoutCnt* whenever the RxBuffer was empty and now has data. The counter then starts running downwards towards zero and either stops if the RxBuffer gets empty before the counter elapses or the counter reaches zero and generates the timeout.

5.1.6 Internal Loopback

If the *LB* bit in *UARTn_Ctrl* register is set, the UART will be in the internal loop back mode. In this mode, the transmitter output is internally connected to the receiver input and the *RTSN* is connected to the *CTSN*. It is then possible to perform loop back tests to verify operation of receiver, transmitter and associated software routines. In this mode, the outputs remain in the inactive state, in order to avoid sending out data.

5.2 SpaceWire

5.2.1 SpaceWire Module

The AGGA-4 provides a SpaceWire module (see also Figure 5-4) with four identical SpaceWire interfaces (or channels) for communication purposes.

The four communication SpaceWire links share a common AHB interface and a common DMA controller. Data can be transferred via DMA from a SpaceWire link to the external memory and vice versa. The DMA function requires that the SpaceWire-AHB interface has the capability to become AHB master.

The SpaceWire module includes an interrupt controller (see also Figure 5-5). A common SpaceWire interrupt signal is generated to signal a request of service to the processor via the Communication Interrupt Controller (CIC).

The operation of the SpaceWire interfaces is controlled by registers. The registers used to control the SpaceWire interface are split into two groups. One group contains all registers commonly used by the SpaceWire module (*SpW_ModuleConfig, SpW_ModuleTimeCtrl, SpW_ModuleTimeCode, SpW_ModuleIntMask, SpW_ModuleIntStatus, SpW_ModuleIntClear*). The other group contains all registers which are individual to each SpaceWire channel (*SpW**n**_StatusAndCtrl, SpW**n**_Tx_SAP, SpW**n**_Tx_EAP, SpW**n**_Tx_CAP, SpW**n**_Tx_Rx_Config, SpW**n**_Rx_SAP, SpWn_Rx_EAP, SpWn_Rx_CAP*) and are available 4 times (***n*** = 0..3). The SpaceWire module control and status registers are accessible via the SpaceWire-APB interface.

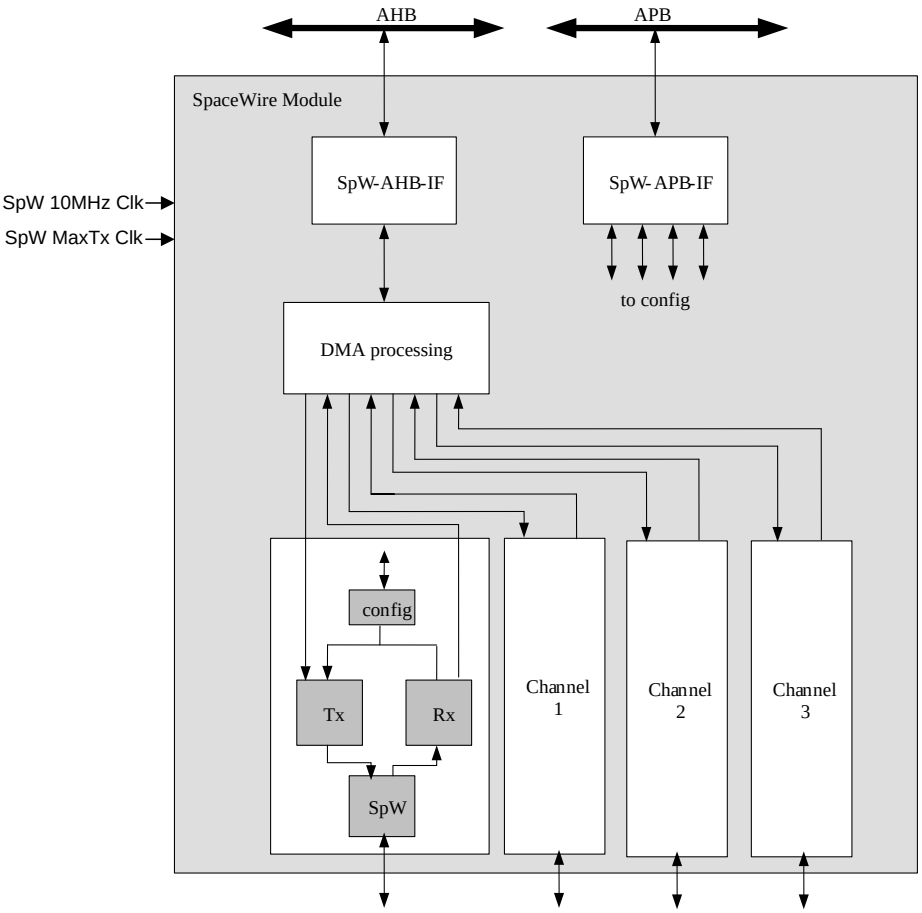


Figure 5-4: SpaceWire Module

5.2.2 Data Transfer

Communication over SpaceWire is done by transferring data packets using DMA. To transmit data the CPU writes the relevant data into the memory and sets the registers in the SpaceWire interface (*SpWn_Tx_SAP* and *SpWn_Tx_EAP*) that points to this data. Further transfer activities are applied by the appropriated DMA without any CPU intervention. To receive data the CPU sets the register in the SpaceWire interface (*SpWn_Rx_SAP* and *SpWn_Rx_EAP*) that points to the receive area in the external memory. For each SpaceWire link 8 registers are provided. Four registers to assign an

area (transmit/receive) in the external memory, two registers which carry the current address value and two configuration registers.

Transmit and receive registers of each of the four SPW links are programmable independently to use separate memory buffers, however all data streams are sharing a common DMA and AHB interface.

When setting up a DMA channel, the DMA pointers must be written in the order *SpWn_R|Tx_SAP*, then *SpWn_R|Tx_EAP*, because it is the write to the *EAP* register which will start the DMA transfer. Moreover, writing to the *SAP or EAP* register while a DMA transfer is in progress shall be avoided, as it will interrupt the ongoing transfer and most likely lead to loss of data.

By reading the current address register (*SpWn_Tx_CAP, SpWn_Rx_CAP*) the progress of data transfer can be observed.

The End Of Packet (EOP), or the occurrence of an error (Error EOP, EEP) on the SpaceWire link is indicated through the interrupt bit *EOP_EEP* (if enabled/unmasked). In addition, in case of EEP, *EEP_Received* will be set in *SpW**n**_Tx_Rx_Config*. The DMA transfer is stopped, and the number of received characters (bytes) will then be given by the difference *SpWn_Rx_CAP - SpWn_Rx_SAP*.

When setting the *NoStopOnEOP* field in *SpW**n**_Tx_Rx_Config*, processing of EOP / EEP in the RX datastream is disabled. All incoming EOP and EEP are discarded, all received SPW packets are concatenated into the same DMA buffer until the buffer is full (CAP has reached EAP), which will be indicated by the *RxAreaFull* interrupt bit.

The maximum transmit (Tx) data rate is x MBit/s, where x is the used AGGA-4 *SysClk* in MHz. Lower Tx data rates can be configured in the *SpWn_StatusAndCtrl* register. The Rx data rate is twice the *SpWRxClk* clock (see **Table 6-2**).

5.2.3 SpW Interrupts

Various SPW events are notified by setting bits in the *SpW_ModuleIntStatus* register. For each of the events, an interrupt to the processor can be raised if unmasked in the *SpW_ModuleIntMask* register. The interrupt bits can be selectively cleared in the *SpW_ModuleIntClear* register.

For each of the four SPW links, the following events are reported:

- SpaceWire link error (*SPWn_LinkError)*
- SpaceWire link connected (*SPWn_LinkConnected)*
- Packet transmission completed (*SPWn_TxDone)*
- Receive DMA buffer (in external memory) full (*SPWn_RxAreaFull)*
- Packet reception completed (EOP/EEP received) (*SPWn)EOP_EEP)*

Figure 5-5 shows the SpaceWire interrupt controller.

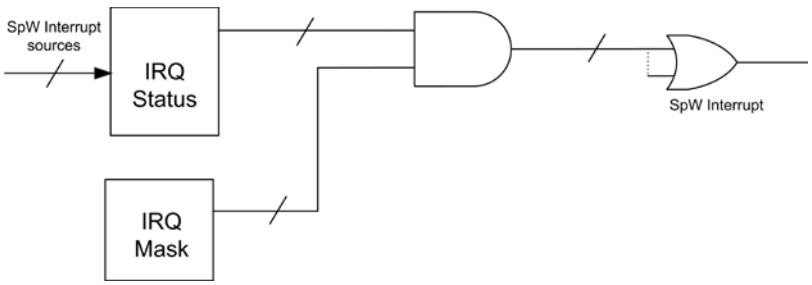


Figure 5-5: SpW Interrupt Controller

5.3 MIL-Bus

5.3.1 General description

The MIL-STD-1553B bus is a serial bus whose application is foreseen worldwide in many Space Applications. It is a multiplex data bus that operates asynchronously in a command/response mode. The transmission on the bus occurs in a half-duplex way. Every data transfer occurring on the bus is initialized and managed by the Bus Controller. More information can be found in [RD-07]and [RD-08].

The data exchanges between the terminals connected to the bus involve 16 bits words. The data can be driven through the bus at a speed of 10^6 bits/s.

The IP1553 provides all facilities for efficient coupling between the Application and the 1553B buses.

The IP1553 has a dual bus capability and operates as a Remote Terminal (RT).

As a Remote Terminal, the IP1553 accepts all options and mode commands specified by the MIL-STD-1553B protocol. In addition, the transfer commands (including mode commands), may optionally be illegalized through dedicated Characterization words.

In the Remote Terminal mode, the IP1553 offers a high level of autonomy. 1553B transfers are controlled by the IP1553 on a Sub Address based splitting scheme. The Application can allow multiple buffers which sizes are defined by Sub Address. The IP1553 provides a Control Word that ensures the completion and the validity of the message and an Information Word giving dating and the address where data were stored.

The IP1553 is fully compatible with MIL-STD-1553B protocol. The next lines present the main features provided by the IP1553:

- Full MIL-STD-1553B compliance
- Remote Terminal capability
- Dual redundant bus capability
- Fully compatible with an AMBA bus interface
- Programmable non response time-out (14 µs or 31 µs)
- Loop test for transmission checking
- Illegal commands management (in RT mode)

5.3.2 Architectural description

5.3.2.1 1553B bus coupling module architecture

Figure 5-6 describes a typical IP1553 based on a 1553B bus coupling module. Very little additional hardware is required to ensure a full dual redundant 1553B bus coupling to an Application.

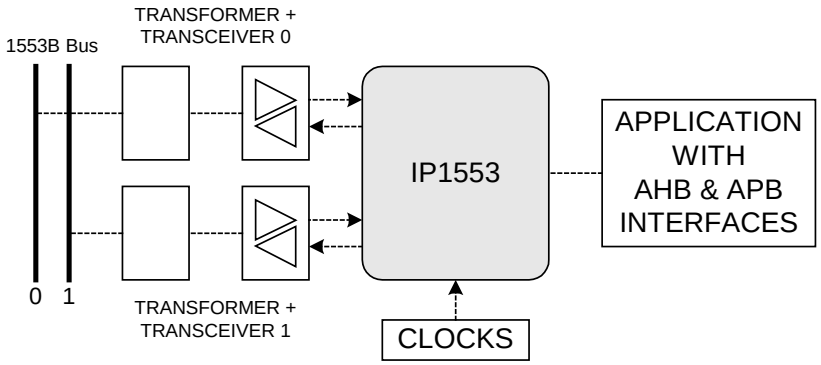


Figure 5-6: IP1553 environment

5.3.2.2 IP1553 internal architecture

The internal architecture of IP1553 is shown on Figure 5-7. It is composed of the five major following elements: Manchester Encoder/Decoder blocks, Internal registers, Commands management & Executing unit, Application Interface block including an AMBA AHB interface and an APB interface.

5.3.2.2.1 Manchester Encoder/Decoder blocks

The Encoder converts the parallel binary data to a serial Manchester encoded bit stream. The serial bit stream is then driven to the active 1553B bus transmitter. Two Encoders are included in the IP1535 chip.

The Decoder converts the 1553B serial Manchester data from one receiver to parallel binary data. Thanks to two independent Decoders the IP1553 is able to listen simultaneously on both buses.

5.3.2.2.2 Internal registers

They contain necessary information for IP1553 chip's configuration or operation.

5.3.2.2.3 Commands management & executing unit

This block manages the 1553B protocol. Its function is defined by the Internal registers which control the way the IP1553 initiates responds to commands in the Remote Terminal mode.

5.3.2.2.4 Application Interface block

The Application Interface provides an APB interface for configuration and status registers. It provides also an AMBA AHB interface for memory access.

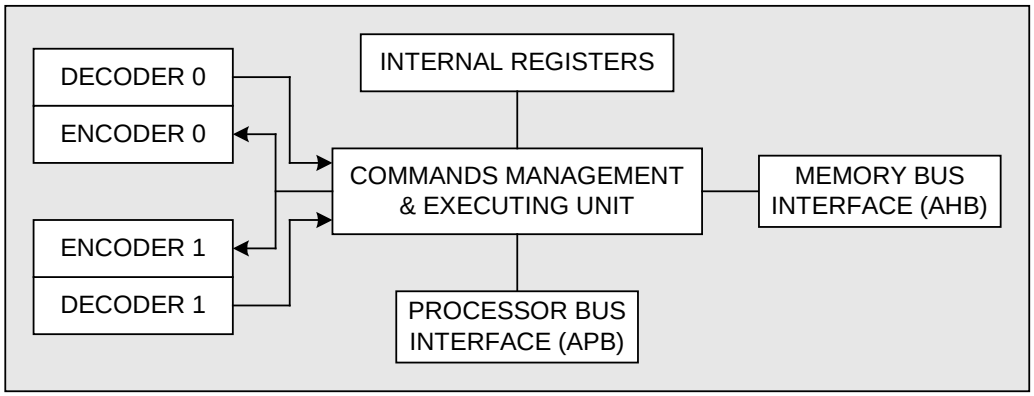


Figure 5-7: IP1553 internal block diagram

5.3.3 IP1553 Initialization

The IP1553 acts as Remote Terminal (RT).

After a *AGGA4_RESET_N* or *PWR_ON_RESET_N* activation, the IP1553 enters the initialization sequence. After a Rst1553 bit activation, the IP1553 enters the reset sequence.

In order to prevent any hazardous functioning of the chip, the following inputs or bits in registers must be stable out of the "Reset state": *Mode*, *TimeOut*, *DbcEn*, *BrCstEn*, *WdSize*, *LockEn*, *DW16En*, *ExtArea* bits. Any modification on these bits must be followed by the activation of *Rst1553* bit (*C53CDST*).

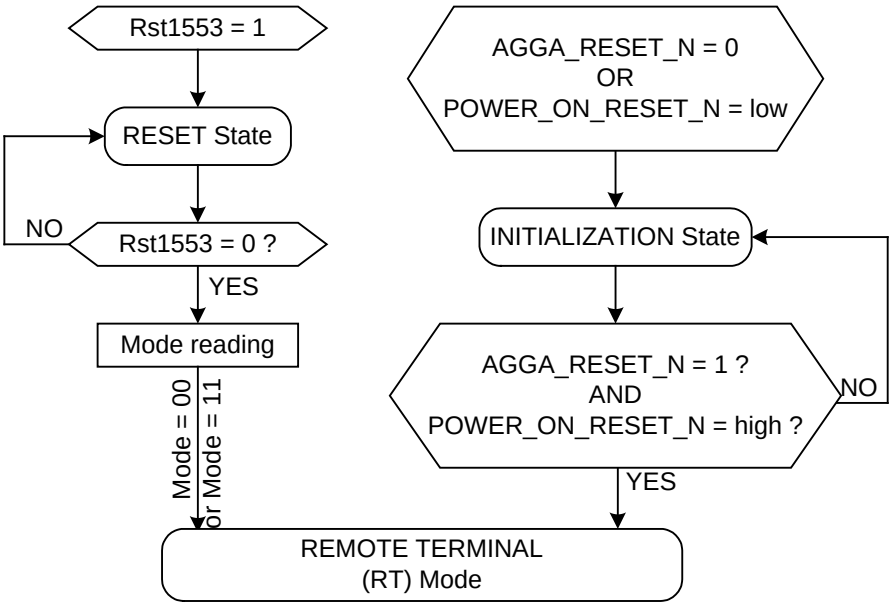


Figure 5-8: IP1553 initialization and reset sequence

The Reset state corresponds to a reset of the IP1553 except its Bus Processor Interface in order to allow the Application to write in its Configuration register and its Command register.

The Initialization state corresponds to a reset of the whole IP1553. Therefore, the IP1553 cannot be configured during Initialization state as its Bus Processor Interface is not active. The IP1553 leaves its Initialization state 5 µs after the rising edge of the *AGGA4_RESET_N* or *PWR_ON_RESET_N* input pin.

5.3.3.1 APB Interface

The detailed register description can be found in the programming section (chapter 7.2.14)

The Application should configure these registers after the Initialization state ends.

When the Application set the *Rst1553* bit to "0", the IP1553 leaves its reset state. The Application can then set the *GoStop* bit to "1".

Upon *AGGA4_RESET_N* or *PWR_ON_RESET_N* input activation the IP1553 enters Initialization state and reset its eight registers. Upon *Rst1553* bit activation the IP1553 enters Reset state and reset the bits that are in read mode in the *C53CDST*, *C53RTI* and the *C53TTI* register.

5.3.3.2 Remote Terminal (RT) mode definition

5.3.3.2.1 RT mode working scheme

The Remote Terminal sequence is as shown on Figure 5-9.

During "Initialization state" the IP1553 initializes its registers. During this period the IP1553 does not respond to any request from the 1553B bus. The *RstSt* bit (*C53CDST*) is set to "1" by the IP1553 to indicate that it is in "Initialization state", otherwise this bit is set to "0".

The IP1553 enters its "Stand-by state" after the end of the initialization. If *GoStop* bit (*C53CDST*) is set to "1", the IP1553 enters its "Active state". The *BusyFlag* bit (*C53CDST*) indicates when it is set to "1" that the IP1553 is in "Active state".

Upon receipt of a valid command, the IP1553 performs an RT address parity control. In case of mismatch between hard-wired parity and computed parity, the IP1553 does not process the command and set the *ErrParAd* bit in *C53CDST* and generate C53Err interrupt through activation of the *ErrRTAd* bit in *C53EIT*.

Upon receipt of "Reset remote terminal" Mode command, the IP1553 generates the C53Rst interrupt through activation of *RstCom* bit in *C53RIT* register and drives a 0 pulse on the open-drain output *MIL_RESET_OUT_N*.

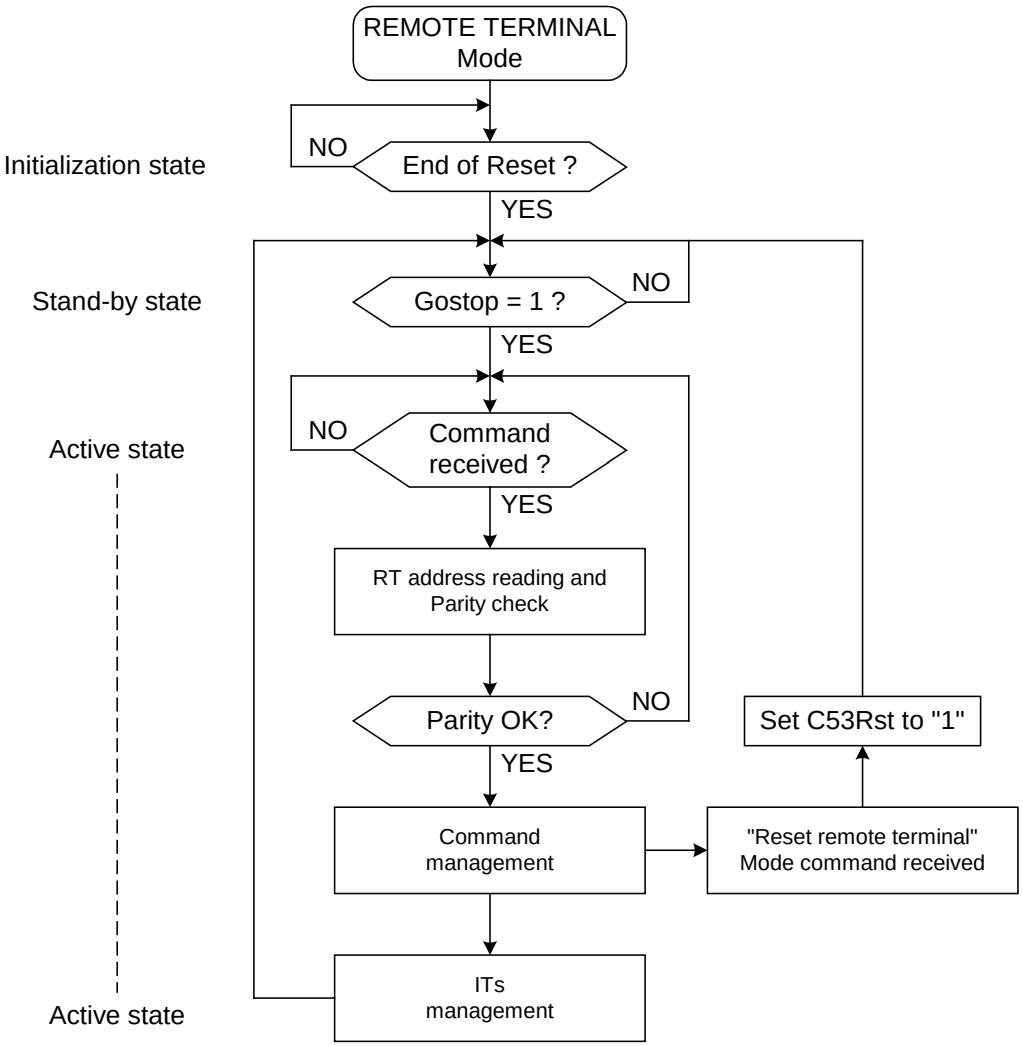


Figure 5-9: RT mode working scheme

5.3.3.2.2 RT mode internal registers definition

In the Remote Terminal mode, the following registers are used: the Status word register, the Last Command register and the BIT register.

5.3.3.2.2.1 Status word register

This register contains the last Status word sent in response to Bus Controller interrogation.

The Status word register is as described hereafter:

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
RTAD(4:0)					ME	NULL	SREQ	NULL	NULL	NULL	BRX	BUSY	SSF	DBC	TF

Figure 5-10: Status word structure

RTAD(4:0) Remote Terminal address

ME
Message Error

SREQ
Service Request bit

BRX
Broadcast Command received

BUSY Busy bit

SSF Subsystem Flag bit

DBC Dynamic Bus Control acceptance

TF Terminal Flag Bit = TTOFLG + MEMERR + BUSYAP

NULL
Unused

During Transmit BIT word TF bit is forced to "0". When TF bit is high, the bits B10 to B1 are irrelevant.

The bit BUSY is always set to "0" as none busy case can occur in the IP1553. It is possible for the Application to update SSF and SREQ bits of the Status word thanks to the Characterization word contained in memory. The Application is also updated the DBC bit of the Status word thanks to the *DbcEn* bit of the Configuration register (*C53CF*), see § 5.3.3.2.3.5 and § 5.3.3.2.4.2.

At initialization, the register is reset at 0000 (H) except Remote Terminal address.

5.3.3.2.2.2 Command word register

This register contains the last valid received command word addressing the RT. At initialization, the register is reset at 0000 (H)

The Command word register is as described hereafter:

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
RTAD(4:0)					T/R	SA(4:0)					WC(4:0)				

Figure 5-11: Command word structure

RTAD(4:0) Remote Terminal address

T/R Transmit/Receive

SA(4:0) Sub-address/Mode

CWC(4:0) Data word count/Mode code

5.3.3.2.2.3 Built-In-Test (BIT) word register

This register informs the Bus controller about possible 1553B transfer errors.

The BIT word register is as described hereafter:

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
NULL	MEMERR	TTO0	TTO1	TFINH	NULL	NULL	NULL	NULL	HIWRD	LOWRD	UNDCMD	T/R_ILL	LP	BUSYAP	TTOFLG

Figure 5-12: BIT word structure

MEMERR Hresp input = Error

TTO0	Transmission time-out activation on bus 0
TTO1	Transmission time-out activation on bus 1
TFINH	Inhibition of Terminal Flag bit
HIWRD	Number of words received higher than expected
LOWRD	Number of words received lower than expected
UNDCMD	Undefined Command received
T/R_ILL	Bit T/R illegal in the Mode command received
LP	*Reserved*
BUSYAP	Time-out access
TTOFLG	Transmission time-out activation on active bus
NULL	Unused

At initialization, the register is reset at 0000 (H).

Note:

- For transmission time-out description refer to § 5.3.3.2.7.2.
- Undefined Command bit is always set to "0" as Undefined commands are treated as illegal commands (see § 5.3.3.2.4.1)

5.3.3.2.2.4 Internal registers management

The internal registers are updated for each valid command addressing the RT (except for some Mode commands) in the following way:

TYPE OF THE COMMAND	COMMAND Register	STATUS Register	BIT Register
Mode command: "Transmit Last Command"	Not modified	Not modified	Not modified
Mode command: "Transmit Status word"	Updated	Not modified	Not modified
Mode command: "Transmit BIT word"	Updated	Updated	Not modified
Other valid commands addressing the RT	Updated	Updated	Updated
Invalid commands or commands addressing another RT	Not modified	Not modified	Not modified

5.3.3.2.3 RT mode special functions

5.3.3.2.3.1 GoStop bit (C53CDST)

The 1553 activity can be controlled with this bit: a low to high transition or a high level allows the 1553 to start/continue its Command processing, otherwise it stays/returns to "Stand-by state" (after the completion of the current Command processing).

5.3.3.2.3.2 RstSt bit (C53CDST)

This signal is set to "0" by the 1553 block when it is in "Initialization state", otherwise it is set to "1" during "Active state". So it follows the activation of AGGA4_RESET_N input pin, *1553Rst* bit or C53Rst interrupt.

5.3.3.2.3.3 BrCstEn bit (C53CF)

It is possible to control the authorization of Broadcast commands with *BrCstEn* bit.

If *BrCstEn* = "1" then Broadcast commands is allowed, otherwise Broadcast commands is rejected (is equivalent to commands with a different RT address).

5.3.3.2.3.4 Memory access error

When the IP1553 encounters a Memory access error, it sets MEMERR flag to "1" in the BIT word if Hresp input = Error during the access, or set BUSYAP bit to "1" in the BIT word if a Time-out has occurred during the access.

If the error occurs before transmission of the Status word, the IP1553 sends its Status word with ME bit set to "1" then stops the Command processing. Otherwise, it completes the emission of the current word then stops the Command processing. In all cases, the IP1553 generates the C53Err interrupt.

5.3.3.2.3.5 SSF and SREQ bits

During the Command processing, the IP1553 reads a Characterization word that contains information about SSF and SREQ bits. If these bits are set to "1" in the Characterization word, the IP1553 sets them to "1" in the Status word, otherwise the IP1553 sets them to "0".

5.3.3.2.3.6 RTAD(4:0) and RTParity

The RTAD(4:0) define the Remote Terminal address according to the following table:

Address	RTAD(4)	RTAD(3)	RTAD(2)	RTAD(1)	RTAD(0)
0	0	0	0	0	0
1	0	0	0	0	1
…					
31	1	1	1	1	1

Note: Address 31 is reserved for broadcast messages (refer to AD1)

The RTParity allows the detection of a possible address parity error. This input is wired in order to obtain an odd parity on the Remote Terminal address as stated hereafter:

RTParity = 1 ⊕ RTAD(4) ⊕ RTAD(3) ⊕ RTAD(2) ⊕ RTAD(1) ⊕ RTAD(0)

The RTParity input value is checked for each valid command received. If the value reflects a parity error, the IP1553 does nottreat the command and sets *ErrParAd* bit to "1" in *C53CDST* and generate C53Err interrupt through activation of *ErrRTAd* bit in *C53EIT*.

5.3.3.2.4 Commands management

5.3.3.2.4.1 Commands legalization/illegalization

The IP1553 is able to legalize/illegalize any incoming command (including Mode commands, except "Transmit Last Command", "Transmit Status" and "Transmit BIT"), using Characterization words included into the Characterization area of the memory. One Characterization word is linked to each command (the format is as defined § 5.3.3.2.8.4).

The IP1553 accesses to the Characterization memory at the beginning of each command processing to check the legality of the command (except for "Transmit Last Command", "Transmit Status" and "Transmit BIT" Mode commands which are always legal).

To declare legal an incoming command the following condition is fulfilled:

- LEG bit of the Characterization word is set to "1"

For any illegal command the IP1553 sends back its Status word with ME bit set to "1".

All unused commands are illegalized in the Characterization memory.

5.3.3.2.4.2 Mode commands management

The IP1553 behaviour is as described hereafter for the following Mode commands:

- Dynamic bus control:

The IP1553 is able to accept or not the control of the 1553B bus.

Application authorizes its acceptance by setting to "1" a dedicated bit *DbcEn* (*C53CF*).

Upon receipt of this command, if legalized and if *DbcEn* bit is high level, the IP1553 sends its Status word with DBC bit set to "1" and generate the C53It interrupt through the activation of the ItDbc bit (*C53NIT*).

Otherwise it sends only its Status word with DBC bit set to "0" and also ME bit set to "1" if the command is illegalized.

The RT to BC mode switching will be effective only after modification of Mode bits (*C53CF*) and AGGA4_RESET_N, PWR_ON_RESET_N or Rst1553 activation.

- Synchronize without data word:

Upon receipt of this command, if legalized, the IP1553 generates the interrupt C53It through the activation of the *ItSync* bit (*C53NIT*) and send its Status word. Otherwise it sends only its Status word with ME bit set to "1".

- Synchronize with data word:

Upon receipt of this command, if legalized, the IP1553 stores the "Synchronization word" into the Buffer associated to Receive Mode commands, generate the interrupt C53It through the activation of the *ItSync* bit (*C53NIT*) and send its Status word. Otherwise it sends only its Status word with ME bit set to "1".

- Transmit status word:

Upon receipt of this command, if legalized, the IP1553 sends the content of the "Status word" register.

- Initiate self test:

Upon receipt of this command, if legalized, the IP1553 sends its Status word. Otherwise it sends only its Status word with ME bit set to "1".

- Transmit BIT word:

Upon receipt of this command, if legalized, the IP1553 sends its Status word followed by the content of the "BIT word" register. Otherwise it sends only its Status word with ME bit set to "1".

- Transmitter shutdown:

Upon receipt of this command, if legalized, the IP1553 inhibits the opposite bus transceiver by forcing to high level the corresponding pins (*MIL_TX0*, *MIL_TX0B*, *MIL_TX0INH* or *MIL_TX1*, *MIL_TX1B*, *MIL_TX1INH*). Otherwise it sends only its Status word with ME bit set to "1".

- Override transmitter shutdown:

Upon receipt of this command, if legalized, the IP1553 cancels the inhibition of the opposite bus transceiver. Otherwise it sends only its Status word with ME bit set to "1".

- Inhibit terminal flag:

Upon receipt of this command, if legalized, the IP1553 forces to "0" the TF bit of the Status word and send its Status word. Otherwise it sends only its Status word with ME bit set to "1".

- Override inhibit terminal flag:

Upon receipt of this command, if legalized, the IP1553 cancels the inhibition of the TF bit of the Status word send its Status word. Otherwise it sends only its Status word with ME bit set to "1".

- Reset remote terminal:

Upon receipt of this command, if legalized, the IP1553 sends its Status word and generate the *C53Rst* interrupt (*C53RIT*) and drives a 0 pulse on the open-drain output *MIL_RESET_OUT_N*. Otherwise it sends only its Status word with ME bit set to "1".

The Reset state corresponds to a reset of the IP1553 except its Bus Processor Interface (APB) in order to allow the Application to write in its Configuration register and its Command register. In this state, it resets the bits that are in read mode in the C53CDST, C53RTI and C53TTI registers.

- Transmit vector word:

Upon receipt of this command, if legalized, the IP1553 sends its Status word, read the Vector word from the Buffer associated to Transmit Mode commands send it. Otherwise it sends only its Status word with ME bit set to "1".

- Transmit last command word:

Upon receipt of this command, the IP1553 sends its Status word followed by the content of "Command word" register.

- Selected transmitter shutdown:

Upon receipt of this command, if legalized, the IP1553 stores the "data word" into the Buffer associated to Receive Mode commands and send its Status word. Otherwise it sends only its Status word with ME bit set to "1".

- Override selected transmitter shutdown:

Upon receipt of this command, if legalized, the IP1553 stores the "data word" into the Buffer associated to Receive Mode commands and send its Status word. Otherwise it sends only its Status word with ME bit set to "1".

- Reserved Mode commands:

These Mode commands should be illegalized into the Characterization words. Then upon receipt of these commands, the IP1553 would only send its Status word with ME bit set to "1".
The mode codes available are taken from the following table:

Transmit-receive bit	Mode code	Function	Associated data word	Broadcast command allowed	Command implemented
1	00000	Dynamic Bus Control	No	No	Yes
1	00001	Synchronize	No	Yes	Yes
1	00010	Transmit Status Word	No	No	Yes
1	00011	Initiate self-test	No	Yes	No(1)
1	00100	Transmitter shutdown	No	Yes	Yes
1	00101	Override transmitter shutdown	No	Yes	Yes
1	00110	Inhibit terminal flag bit	No	Yes	Yes
1	00111	Override inhibit terminal flag bit	No	Yes	Yes
1	01000	Reset remote terminal	No	Yes	Yes
1	01001	Reserved	No	No	No
1	01111	Reserved	No	No	No
1	10000	Transmit vector word	Yes	No	Yes
0	10001	Synchronize	Yes	Yes	Yes
1	10010	Transmit last command	Yes	No	Yes
1	10011	Transmit bit word	Yes	No	Yes
0	10100	Selected transmitter shutdown	Yes	Yes	Yes
0	10101	Override selected transmitter shutdown	Yes	Yes	Yes
1 or 0	10110	Reserved	Yes	No	No
1 or 0	11111	Reserved	Yes	No	No

(1) The response for this command depends only on the legalization bit in the Characterization word. There is no special treatment associated with this command.

5.3.3.2.5 No response time-out management

The IP1553 is able to handle programmable no response time-out. The time-out value is taken into account to detect a no-response from a Remote Terminal in a RT to RT transfer. The *TimeOut* bit (*C53CF*) determines the chosen value for no response time-out. The smallest value is 14 µs (*TimeOut* = "0"), and the highest value is 31 µs (*TimeOut* = "1").

5.3.3.2.6 Redundancy management

The IP1553 is fully compliant to § 4.6 of document AD1.

The IP1553 is able to detect any incoming command on either the nominal or the redundant bus and respond on the bus it has been activated by.

The content of the internal registers (Status word register, Last Command register and BIT register) always corresponds to a transfer performed on the active bus.

If a transfer is interrupted on a 1553B bus by a valid command incoming on the other bus and addressing the RT, the IP1553 stops the processing of the previous command and start the processing of the new command.

5.3.3.2.7 Test support & Error report

5.3.3.2.7.1 Built-In-Test (BIT) word

This 16 bit word provides error information for each 1553B transfer.

The BIT word corresponds to the bit description given in § 5.3.3.2.2.3.

5.3.3.2.7.2 Transmission time-out

The IP1553 contains a time-out to preclude a signal transmission of greater than 800 µs. If this time-out triggers, then TTO0 or TTO1 and TTOFLG is set to "1" in the BIT word (see § 5.3.3.2.2.3).

This function does not preclude a correct transmission in response to a command.

Reset of this time-out function is performed by the reception of a valid command addressing the RT on the bus on which the time-out has occurred.

5.3.3.2.7.3 "Loop Test" and "Self test"

The 1553 function in AGGA4 neither implements a "Self Test" nor a "Loop Test".

5.3.3.2.8 Memory management

The data management function ensures the following points:

- The transfer of received data from the 1553B bus to the memory,
- The availability of these data for the Application,
- The transfer of data to transmit on the 1553B bus from the memory.

The IP1553 realizes the memory management and communicate with the memory controller. It uses a shared area in memory. This area is defined by the Application thanks to *MemArea* bits (*C53CF*). *MemArea* defines an area for the IP1553 of 128Kwords. This area can be placed in memory at each multiple of 128K word address in the whole memory, i.e. when MemArea is increased by 1 the IP1553 area will be moved of 128 K word address.

There are three different areas in this shared memory:

- A command area which size is 4 Kwords including the received command words, control words associated to these commands, associated data to Mode commands, characterization words and two double indirection tables of 32 addresses;
- A data received area which size is 32 Kwords including data words received on 1553B bus;
- A data to transmit area which size is 32 Kwords including data words to transmit on 1553B bus.

The 1553B and Identification words are stored in memory. The bits WdSize and DW16En (C53CF) determine if the IP1553 stores the words in 32 bit words or 16 bit words. Wdsize= 1 is only possible if DW16En is set to 1.

16 MSB	**16 LSB**
1st 1553B word	unused

16 MSB	16 LSB
1st Identification word	
2nd 1553B word	
2nd identification word	

1553B and Identification words mapping with 16 bit Word mode

16 MSB	16 LSB
1st 1553B word	1st Identification word
2nd 1553B word	2nd Identification word

1553B and Identification words mapping with 32 bit Word mode

5.3.3.2.8.1 Command area

There are the following parts in the command area (Figure 5-16):

- A circular buffer, which contains the received command words on the 1553B bus. When a command is received and legalized, the IP1553 writes in this buffer at the end of the exchange the command word received, a control word giving information about the exchange and a 32 bit word containing a 21 bit dating field and the block number where the data of the command have been stored.

  This buffer is in write access mode by the IP1553 and in read access mode by the Application, the *LastCmdAd* bits in the *C53CDST* gives the address in the command area of the last command treated.

  At the end of the buffer, the access pointer takes the beginning address of the area.

- Two 32 word tables including the data associated to each Mode Command.

  The first table contains the data associated to receive Mode command. It is in write access mode by the IP1553 and in read access mode by the Application. The Mode Code of the command gives the address in the table.

  The second table contains the data associated to transmit Mode command. It is in read access mode by the IP1553 and in write access mode by the Application. The Mode Code of the command gives the address in the table.

- A 32 word table including characterisation words. Each word contains information about the legalization of the command (LEG bit), the enable of the *ItTrok* bit in the *C53NIT* (TROK bit) and the Status response thanks to SSFB and SREQ bits. The Application initializes this table in order to allow the IP1553 to use it.

  When the IP1553 receives a command, it accesses to this table to read the characterization word associated to the command word received. The Sub Address or the Mode Code gives the address in the table.

- A double 32 word table giving the beginning block number for receive data, the maximum size of the buffer in blocks and the next block to be used by the IP1553 in the data area for each 32 Sub Address. The Application initializes this table in order to allow the IP1553 to use it.

  The table word is as described Figure 5-13 (detailed description § 5.3.3.2.8.4):

B31	B30.....B21	B20.....B11	B10.....B0
ADUPD	Current block	Maximum buffer size	Beginning buffer reception block

Figure 5-13: Indirection table word for reception

When a Receive Command is received, the IP1553 reads the Indirection table word at the address given by the Sub Address. Then the IP1553 writes the received data in the data area starting at the Beginning buffer reception block (BRBlk) + Current block value (CurBlk). The value of the CurBlk field is zero at the beginning. The size of a block is 16 (32 bit words) when 32 bit Word mode is activated, otherwise the size is 32 (32 bit words, but only MSB are used). Once this task is finished, the IP1553 compares the Maximum buffer size (MaxBS) with (CurBlk+1):

- If the buffer is not full (CurBlk + 1 < MaxBS), the IP1553 updates the CurBlk if the bit Address Update (ADUPD) is set to "1" by writing CurBlk+1. If the ADUPD bit is not set, the CurBlk does not be updated and the next data received at this same Sub Address is written at the CurBlk + BRBlk. Meanwhile, the BRBlk may have been modified by the Application.
- If the buffer is full (CurBlk + 1 = MaxBS), the IP1553 changes its internal table indicator for this Sub Address in order to point on the complementary table at the time of the next Receive command at this Sub Address. The IP1553 updates the CurBlk field with zero and generate the C53It interrupt through the *ItSwitch* bit (*C53NIT*) activation.

The IP1553 maintains a table, which contains the table indicator for each Sub Address. This indicator is reported in the Control Word associated to the received command.

At initialization, the IP1553 uses the table 0, which starts at "MemArea+10F80H" and ends at "MemArea+10F9FH". When the buffer available will be full and if the ADUPD bit is set to "1", the IP1553 uses the table 1 which starts at "MemArea+10FA0H" and ends at "MemArea+10FBFH" for the Sub Address considered. It warns the Application thanks to C53It. When the buffer defined by the table 1 will have be full and if the ADUPD bit is set to "1", the IP1553 uses the table 0 again, and so on…

- A double 32 word table giving the beginning block number for transmit data, the maximum size of the buffer in blocks and the next block to be used by the IP1553 in the data area for each 32 Sub Address. The Application initializes this table in order to allow the IP1553 to use it

The table word is as described in Figure 5-14 (detailed description § 5.3.3.2.8.4)

B31	B30.....B21	B20.....B11	B10.....B0
ADUPD	Current block	Maximum buffer size	Beginning buffer transmission block

Figure 5-14: Indirection table word for transmission

When a Transmit Command is received, the IP1553 reads the Indirection table word at the address given by the Sub Address. Then the IP1553 reads the data to transmit in the data area starting at the Beginning buffer transmission block (BTBlk) + Current block value (CurBlk). The value of the CurBlk field is zero at the beginning. The size of a block is 16 (32 bit words) when 32 bit Word mode is activated, otherwise the size is 32 (32 bit words, but only MSB are used). Once this task is finished, the IP1553 compares the Maximum buffer size (MaxBS) with (CurBlk+1):

- If the buffer is not entirely read (CurBlk + 1 < MaxBS), the IP1553 updates the CurBlk if the bit Address Update (ADUPD) is set to "1" by writing CurBlk+1. If the ADUPD bit is not set, the CurBlk is not updated and the next data to transmit from this same Sub Address is read at the CurBlk + BTBlk. Meanwhile, the BTBlk may have been modified by the Application.
- If the buffer is entirely read (CurBlk +1 = MaxBS), the IP1553 changes its internal table indicator for this Sub Address in order to point on the complementary table at the time of the next Transmit

command at this Sub Address. The IP1553 updates the CurBlk field with zero and generate the C53It interrupt through the *ItSwitch* bit (*C53NIT*) activation.

The IP1553 maintains a table, which contains the table indicator for each Sub Address. This indicator is reported in the Control Word associated to the received command.

At initialization, the IP1553 uses the table 0, which starts at "MemArea+10FC0H" and ends at "MemArea+10FDFH". When the buffer available has been entirely read and if the ADUPD bit is set to "1", the IP1553 uses the table 1 which starts at "MemArea+10FE0H" and ends at "MemArea+10FFFH" for the Sub Address considered. It warns the Application thanks to C53It. When the buffer defined by the table 1 has been entirely read and if the ADUPD bit is set to "1", the IP1553 uses the table 0 again, and so on…

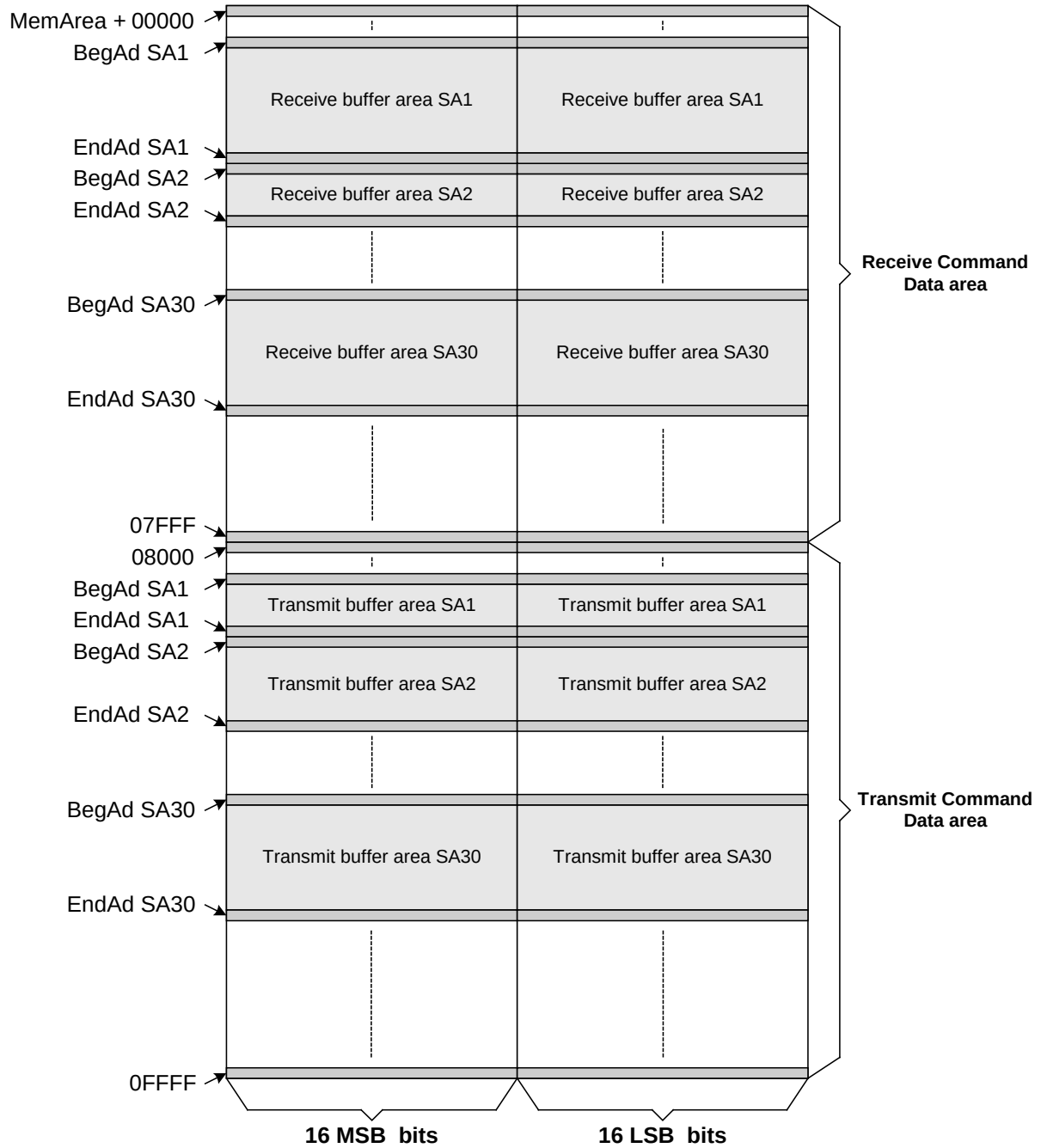


Figure 5-15: Data area mapping in RT mode

Note: The addresses in Figure 5-15 are address words.

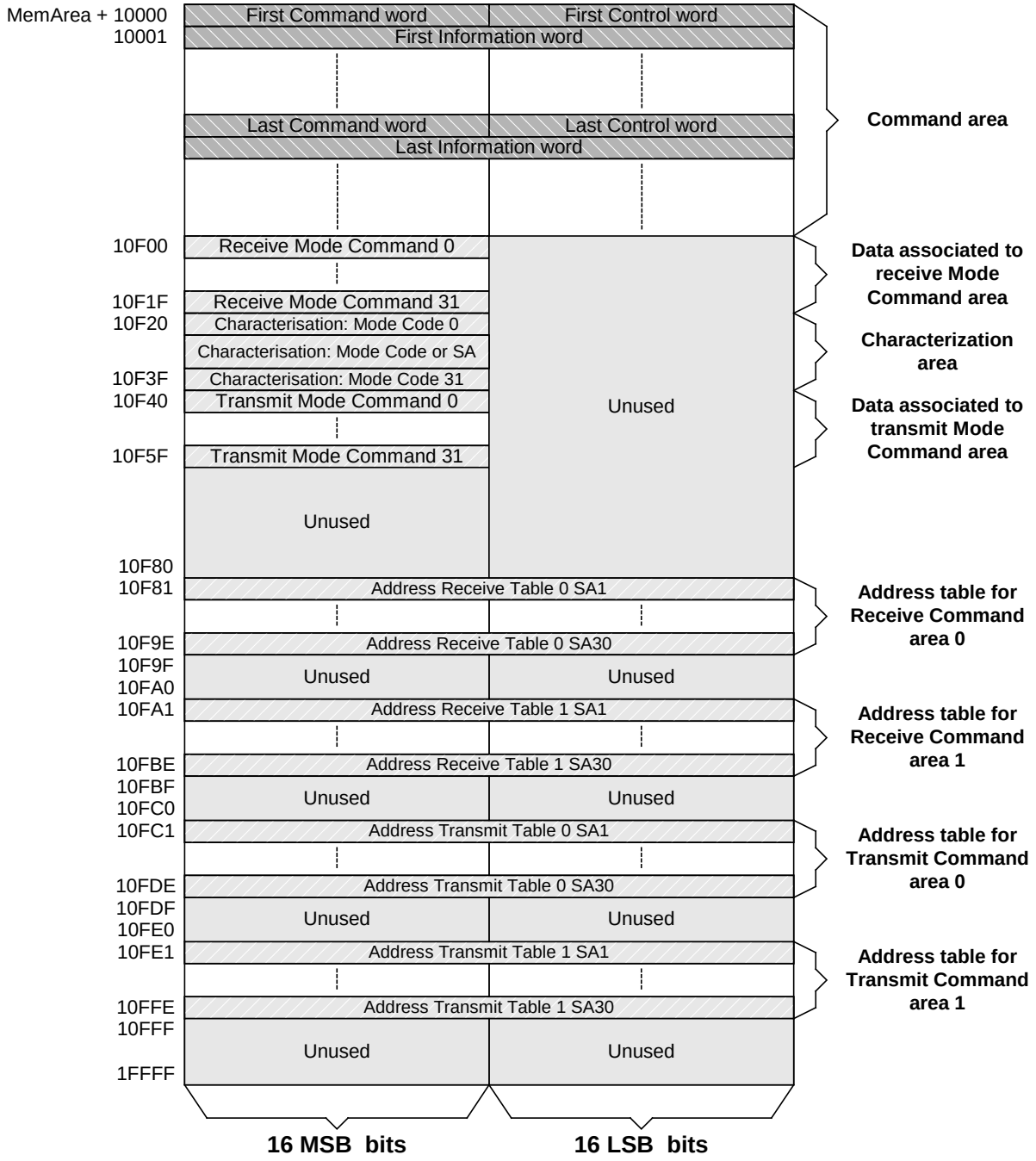


Figure 5-16: Command area mapping in RT mode

Note: The addresses in Figure 5-16 are address words.

5.3.3.2.8.2 Data reception area

The data reception area contains the data received on the 1553B bus. This area starts at "MemArea+00000H" and ends at "MemArea+08000H", that is to say 32 Kwords. It is virtually divided in 30 sub areas corresponding to the 30 Sub Address. Each sub area is identified by its BRBlk and MaxBS, written in the Indirection table word at the address related to the Sub Address of the received command. Each sub area contains whether one or several 16 word wide buffers if the 32 bit Word mode is activated, else one or several 32 word wide buffers to memorize the received data. The Figure 5-17 & Figure 5-18 show how the data are mapped:

	16 Most Significant Bits	16 Least Significant Bits
BRBlk+CurBlk & 0	1st 1553 Data received	2nd 1553 Data received
BRBlk+CurBlk & 1	3rd 1553 Data received	4th 1553 Data received
	⋮	⋮
BRBlk+CurBlk & E	29th 1553 Data received	30th 1553 Data received
BRBlk+CurBlk & F	31th 1553 Data received	32nd 1553 Data received

Figure 5-17: Data reception buffer mapping when 32 bit Word mode

	16 Most Significant Bits	16 Least Significant Bits
BRBlk+CurBlk & 00	1st 1553 Data received	Unused
BRBlk+CurBlk & 01	2nd 1553 Data received	
	⋮	
BRBlk+CurBlk & 1E	31th 1553 Data received	
BRBlk+CurBlk & 1F	32nd 1553 Data received	

Figure 5-18: Data reception buffer mapping when 16 bit Word mode

5.3.3.2.8.3 Data transmission area

The data transmission area contains the data to transmit on the 1553B bus. This area starts at "MemArea+08000H" and ends at "MemArea+0FFFFH", that is to say 32 Kwords. It is virtually divided in 30 sub areas corresponding to the 30 Sub Address. Each sub area is identified by its BTBlk and CurBlk, written in the Indirection table word at the address related to the Sub Address of the received command. Each sub area contains whether one or several 16 word wide buffers if the 32 bit Word mode is activated, else one or several 32 word wide buffers to memorize the data to transmit. The Figure 5-19 and Figure 5-20 show how the data are mapped:

	16 Most Significant Bits	16 Least Significant Bits
BTBlk+CurBlk & 0	1st 1553 Data to transmit	2nd 1553 Data to transmit
BTBlk+CurBlk & 1	3rd 1553 Data to transmit	4th 1553 Data to transmit
	⋮	⋮
BTBlk+CurBlk & E	29th 1553 Data to transmit	30th 1553 Data to transmit
BTBlk+CurBlk & F	31th 1553 Data to transmit	32nd 1553 Data to transmit

Figure 5-19: Data transmission buffer mapping when 32 bit Word mode

	16 Most Significant Bits	16 Least Significant Bits
BTBlk+CurBlk & 00	1st 1553 Data to transmit	Unused
BTBlk+CurBlk & 01	2nd 1553 Data to transmit	
	⋮	
BTBlk+CurBlk & 1E	31th 1553 Data to transmit	
BTBlk+CurBlk & 1F	32nd 1553 Data to transmit	

Figure 5-20: Data transmission buffer mapping when 16 bit Word mode

5.3.3.2.8.4 Memory words structure

There are 6 different words used by the IP1553 in the memory. The data words, the command words, the Control Word, the Characterization Word, the Indirection Word and the Information Word are described hereafter:

- Control Word:

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
DV	T	RT-RT	BUSID	NULL	MEMERR	TTO0	TTO1	TFINH	HIWRD	LOWRD	UNDCMD	T/R_ILL	LP	BUSYAP	TTOFLG

Figure 5-21: Control Word structure

DV	Data Valid
T	Indirection table number used for the command treatment
RT-RT	RT-RT command received (when IP1553 is the receiving RT)
BUSID	Command received on the nominal ("0") or redundant ("1") bus
MEMERR	HResp input = Error for memory access
TTO0	Transmission time-out activation on bus 0
TTO1	Transmission time-out activation on bus 1

TFINH	Inhibition of Terminal Flag bit
HIWRD	Number of words received higher than expected
LOWRD	Number of words received lower than expected
UNDCMD	Undefined Command received
T/R_ILL	Bit T/R illegal in the Mode command received
LP	*Reserved*
BUSYAP	Application Interface is busy (Time-out error for memory access)
	TTOFLG Transmission time-out activation on active bus
NULL	Unused (“0”)

During a nominal exchange bits B11 to B0 is set to “0”. For “Transmit Last Command”, “Transmit Status” and “Transmit BIT”, the control word written represents the state of the previous command.

- Characterization word:

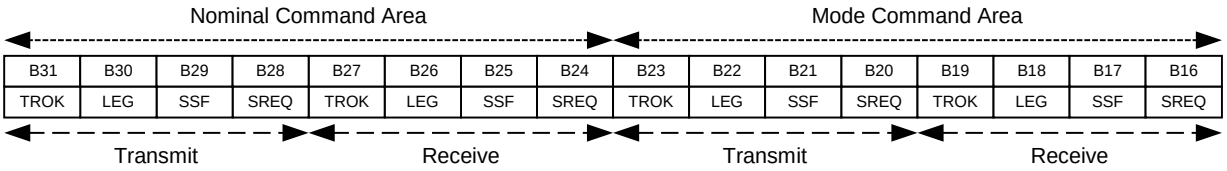


Figure 5-22: Characterization word structure

TROK	Enable of the ItTrok bit (C53NIT)
LEG	Legalization bit: “0” Illegal / “1” Legal
SSF	Sub System Flag bit
SREQ	Service Request bit

For the Sub Address 31 and 32 (“00000b”), the bits B31 to B24 are irrelevant, as the command is a Mode command.

- Information words:

These words are written by the IP1553 after the command and control word in the command area for each valid and legal command treated.

When 32 bit Word mode, the fields in the indirection table are defined hereafter:

B31	B30.....B21	B20.....B11	B10.....B0
ADUPD	Current block	Maximum buffer size	Beginning buffer transmission block

B31	B30.....B21	B20.....B11	B10.....B0
ADUPD	Current block	Maximum buffer size	Beginning buffer reception block

Figure 5-23: Indirection table word for reception when 32 bit Word mode

Beginning buffer transmission or reception block field is 11 bits wide. We can access every block of 16 words of 32 bits (a full 1553 data buffer) in the data reception or transmit area, which size is 32 Kwords.

Maximum buffer size field is 10 bits wide. We cannot allocate more than 16 Kwords to a single Sub Address.

(Maximum buffer size value + Beginning buffer block value) must be lower than 7FFH, which is the address of the last block in the transmission or reception area.

B31.....B21	B20.....B0
Data block number	Dating

Figure 5-24: Information word structure when 32 bit Word mode

The Data block number field is 11 bits wide. Its value corresponds to the sum of Beginning buffer block and Current block. In case of Mode Command, Extended Memory mode or 1553 exchange error (Data Valid bit = “0”) this field has no significance and corresponds to the value of the last valid and legal command, which is not a Mode command. The Data block number for the first block of the memory is zero.

The Dating field is always zero.

When 16 bit Word mode, the fields in the indirection table are defined hereafter:

B31	B30.....B21	B20.....B11	B10.....B0
ADUPD	Current block	Maximum buffer size	Beginning buffer transmission block

B31	B30.....B21	B20.....B11	B10.....B0
ADUPD	Current block	Maximum buffer size	Beginning buffer reception block

Figure 5-25: Indirection table word for reception when 16 bit Word mode

Beginning buffer transmission or reception block field is 11 bits wide. But only the 10 LSB is used by the IP1553. We can access every block of 32 words of 16 most significant bits of the 32 bit words (a full 1553 data buffer) in the data reception or transmit area, which size is 32 Kwords.

Maximum buffer size field is 10 bits wide. We can allocate up to 32 Kwords to a single Sub Address.

(Maximum buffer size value + Beginning buffer block value) must be lower than 3FFH, which is the address of the last block in the transmission or reception area.

B31	B30.....B21	B20.....B0
0	Data block number	Dating

Figure 5-26: Information word structure when 16 bit Word mode

The Data block number field is 10 bits wide. Its value corresponds to the sum of Beginning buffer block and Current block. In case of Mode Command, Extended Memory mode or 1553 exchange error (Data Valid bit = "0") this field has no significance and corresponds to the value of the last valid and legal command, which is not a Mode command. The Data block number for the first block of the memory is zero.

The Dating field is always zero.

5.3.3.2.8.5 Extended Area access to the memory

The IP1553 provides 5 bits (*ExtSubAd* bits in *C53CF*) which are written by the Application to define a Sub Address to be used with the Extended Area mode. This mode is activated thanks to the *ExtArea* bit (*C53CF*). If this bit is set to "0", the ExtSubad is treated as the others.

The Extended Area mode allows the IP1553 to access the whole memory in write or read mode. The beginning address of these accesses is provided by the Extended Memory Base Address register (*C53EMBA*). The IP1553 is then able to access memory areas outside of the 128 Kwords normally provided thanks to *MemArea* bits (*C53CF*). In this peculiar case, there is not an end area address: the *C53EMBA* is incremented for each double 1553 word read or write in the memory, when the Sub Address defined is used and the Extended Area mode allowed. The words count associated to each 1553 command for this Sub Address is even.

The *C53EMBA* is programmable by the 1553B bus thanks to a Receive command with only two words sent to (ExtSubAd + 1). The 10 LSB of the first 1553 data is stored in the bits 25 to 16 of the *C53EMBA*(25 downto 0). The second data word is stored in the 16 LSB bits of this register.

5.3.3.3 IP1553 System interface

5.3.3.3.1 Transceivers interface

The IP1553 is able to support a redundant 1553B bus.

Each connection to a 1553B bus is done through a Transceiver and a Transformer as shown in Figure 5-27.

Each Transceiver – IP1553 interface is composed of at least 5 signals: RX & RXB input signals for Receiver section, TX & TXB & TXINH output signals for Transmitter section. The waveform of these signals is in accordance with Figure 5-28 & Figure 5-29.

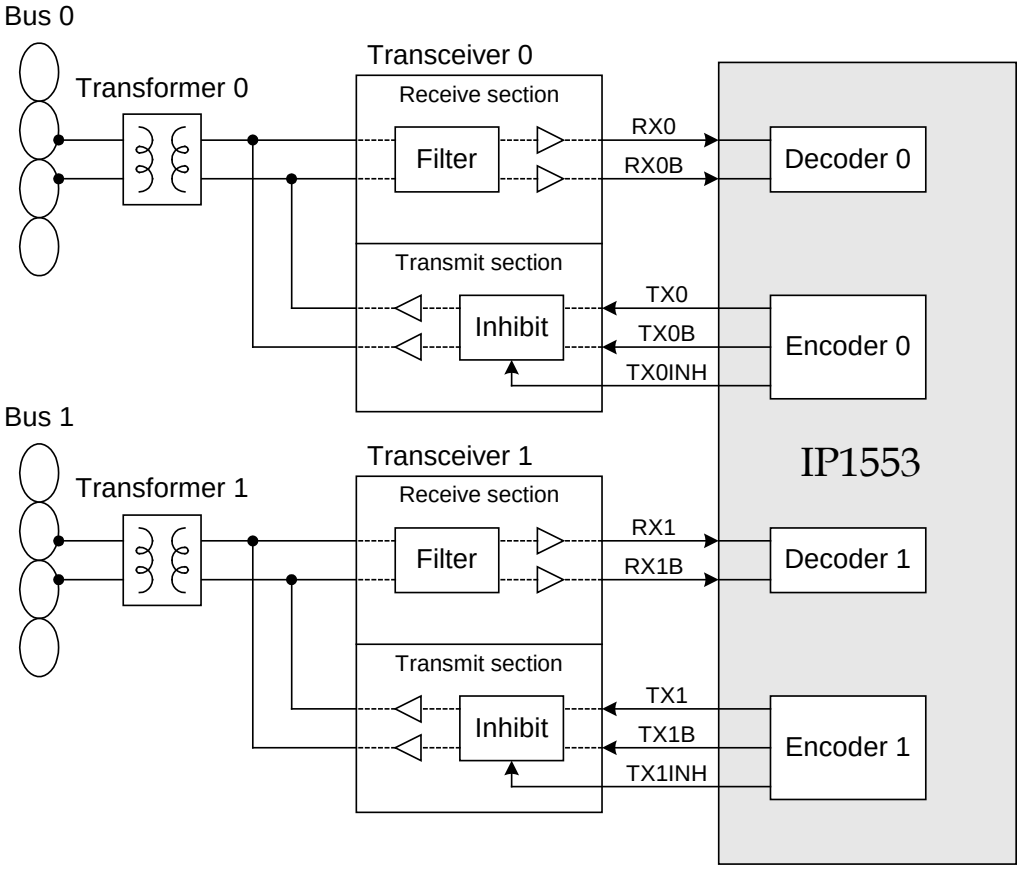


Figure 5-27: Transceiver interface

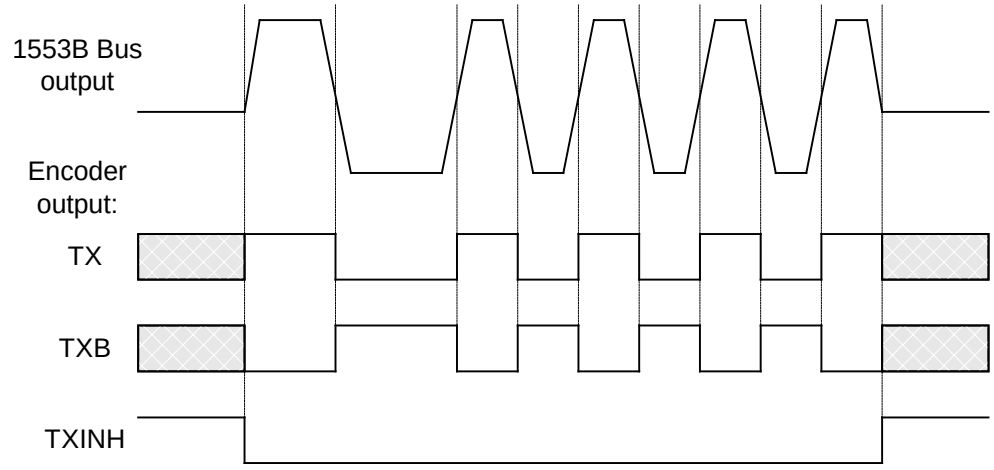


Figure 5-28: Transmitter signals waveform

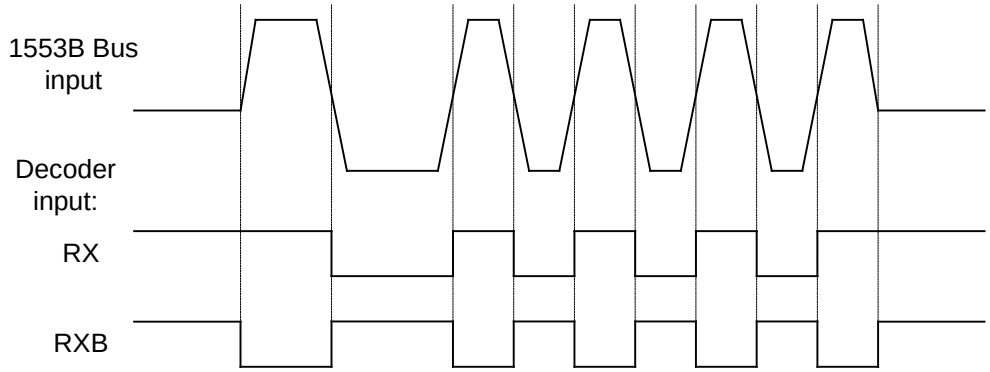


Figure 5-29: Receiver signals waveform

As shown on Figure 5-29, the expected idle level for RX and RXB signals is logic "1". This means that the IP core is compatible with the Smiths type transceiver.

5.3.3.3.2 IP1553 Interrupts support

The IP1553 is able to inform the Application about special events, which are likely to happen. When these events occur, an output interrupt is generated. There are 3 different interrupts: a nominal, a reset and an error interrupt, which the causes are sometimes multiple. They are described hereafter:

- **C53It**: Nominal interrupt. It is generated thanks to the following bits activation: *ItTrok*, *ItSync*, *ItDbc* and *ItSwitch*. To generate the interrupt, the IP1553 combines these sources with a logical OR. These bits are in the Nominal Interrupt register (*C53NIT*).
    - *ItTrok*: Maskable bit (*ItTrokMask* in *C53CF*). In RT mode, this bit is set to "1" at the end of each valid exchange. It is reset to "0" upon *C53NIT* read access.
    - *ItSync*: Maskable bit (*ItSyncMask* in *C53CF*). In RT mode, this bit is set to "1" after "Synchronize" Mode command reception if this command is legalized. In BC mode, this bit is set to "1" after the execution of an Instruction block with SYNC bit set in the first Instruction word. In BM mode, this bit is set to "1" when the IP1553 toggles from one buffer to the other. It is reset to "0" upon *C53NIT* read access.
    - *ItDbc*: Non maskable bit. In RT mode, this bit is set to "1" after "Dynamic bus control" Mode command reception if this command is legalized. It is reset to "0" upon *C53NIT* read access.
    - *ItSwitch*: Non maskable bit. In RT mode, this bit is set to "1" after change of data address table for a sub-address. It is reset to "0" upon *C53NIT* read access.
- **C53Rst**: Reset interrupt. In RT mode, it is generated after "Reset remote terminal" Mode command reception if this command is legalized. The bit *RstCom* is reset to "0" upon *C53RIT* read access.
- **C53Err**: Error interrupt. It is generated thanks to the following bits activation: *ErrMem*, *Err1553*, *ErrInst* and *CwError*. To generate the interrupt, the IP1553 combines these sources with a logical OR. These bits are in the Error Interrupt register (*C53EIT*).
    - *ErrMem*: Non maskable bit. In all modes, this bit is set to "1" in case of Memory access error: time-out access or Hresp input = Error. It is reset to "0" upon *C53EIT* read access.
    - *Err1553*: Maskable bit (*Err1553Mask* in *C53CF*). In RT mode, this bit is set to "1" in case of 1553 error. In BC mode, this bit is set to "1" if EXCERR flag is set in the Control word without additional retry allowed. It is reset to "0" upon *C53EIT* read access.

- *CwError*: Non maskable bit. In BC mode, this bit is set to "1" when the bit CWERR is set in the control word written in memory.
- *ErrInst*: Non maskable bit. In BC mode, this bit is set to "1" in case of Illegal Instruction detection by the BC. It is reset to "0" upon *C53EIT* read access.
- *ErrRTAd*: Non maskable bit. In all modes, this bit is set to "1" in case of RTAD parity error. It is reset to "0" upon *C53EIT* read access.

5.4 DSU Communication Link

DSU communicates either with SpaceWire or with an UART. Selection is made by the *DSU_SPW_EN* pin.

- *DSU_SPW_EN* = low → DSU via UART
- *DSU_SPW_EN* = high → DSU via SpaceWire

5.4.1 DSU UART Operation

The DSU communication link consists of a UART connected to the AHB bus as a master (Figure 5-30). A simple communication protocol is supported to transmit access parameters and data. A link command consists of a control byte, followed by a 32-bit address, followed by optional write data. If the LR bit in the DSU control register is set, a response byte will be sent after each AHB transfer. If the LR bit is not set, a write access does not return any response, while a read access only returns the read data. Data is sent on 8-bit basis as shown in Figure 5-32. Through the communication link, a read or write transfer can be generated to any address on the AHB bus.

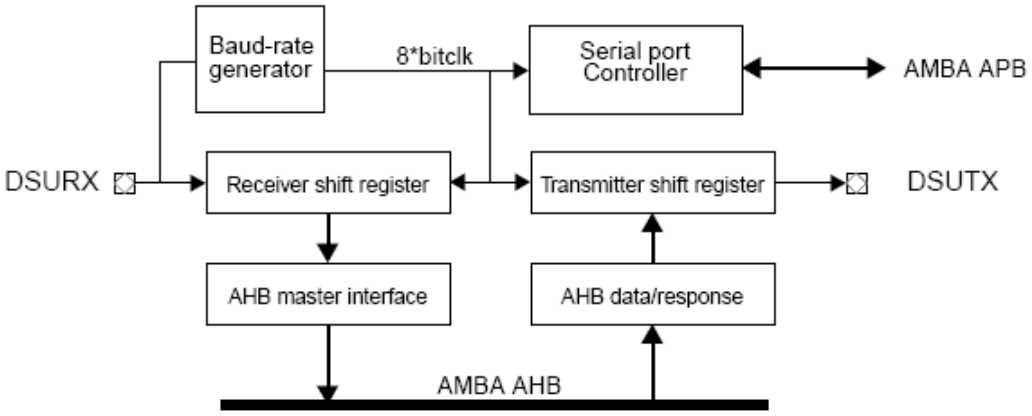


Figure 5-30 DSU communication link block diagram

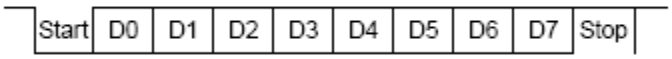


Figure 5-31 DSU UART Frame

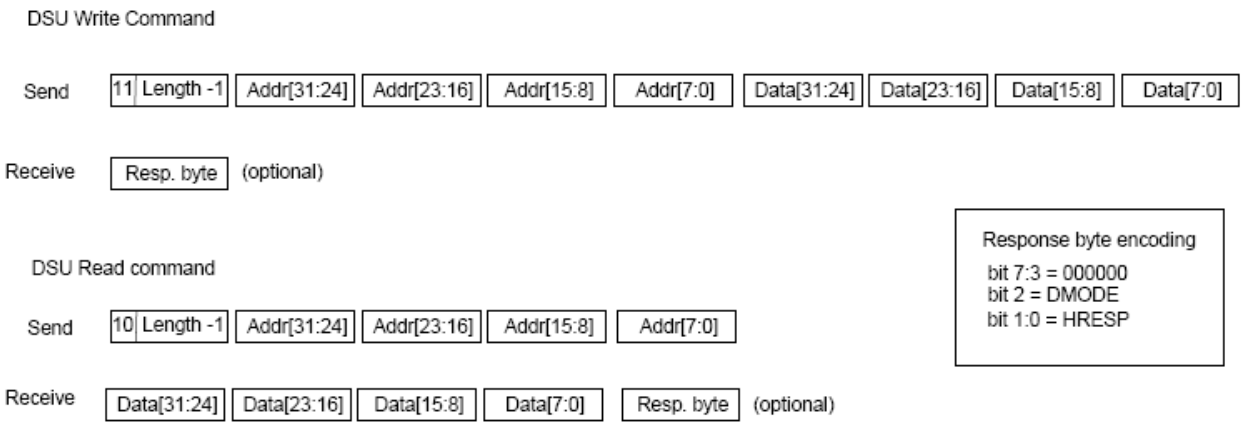


Figure 5-32 DSU Communication link commands

Note: For DSU Write operation the maximum word length is 64!

Note: Response Byte is enabled through *DSU_Ctrl* register

A response byte can optionally be sent when the processor goes from execution mode to debug mode. Block transfers can be performed be setting the length field to *n*-1, where *n* denotes the number of transferred words. For write accesses, the control byte and address is sent once, followed by the number of data words to be written. The address is automatically incremented after each data word. For read accesses, the control byte and address is sent once and the corresponding number of data words is returned.

Details about the *DSU_UART_Status* register can be found in section 7.2.7.1.

5.4.2 DSU UART Baud rate generation (not available in DSU SpaceWire operation)

The UART contains a 14-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the *SysClk* and generates a UART tick each time it underflows. It can be programmed via the register *DSU_UART_Scaler*. The scaler is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate. If not programmed by software, the baud rate will be automatically discovered. This is done by searching for the shortest period between two falling edges of the received data (corresponding to two bit periods). When three identical two-bit periods have been found, the corresponding scaler reload value is latched into the reload register, and the BL bit is set in the UART control register. If the *BL* bit in *DSU_UART_SpW_Ctrl* register is reset by software, the baud rate discovery process is restarted. The baud-rate discovery is also restarted when a 'break' is received by the receiver, allowing to change the baud rate from the external transmitter. For proper baud rate detection, the value 0x55 should be transmitted to the receiver after reset or after sending break.

Please note that it is not recommended to change the scaler setting while the DSU UART link is connected. This will likely cause restart of the link, baud rate discovery leading again to the old baud rate. The DSU UART baud rate should be changed in the external UART device before connecting it to AGGA-4.

The best scaler value for manually programming the baud rate can be calculated as follows:

$$Scaler = SysClk / (baudrate * 8) - 1$$

5.4.3 DSU SpaceWire Operation

DSU SpaceWire does not support RMAP commands, it uses the same protocol as the DSU UART (see **Figure 5-32**). A command always starts with a MSB = 1 to indicate command (read/write and length). Prepended characters with MSB = 0 are ignored. The transmit speed can be adjusted via the *DSU_UART_SpW_Ctrl* register and can be set to either 10 MBit/s or to *SysClk*.

Note: In DSU SpaceWire operation the link is hardcoded to autostart mode and therefore automatically starts the link after NULL token have been received.

5.5 Serial Peripheral Interface (SPI)

The SPI module works as SPI master, but it is prepared to operate either as Clock Master or as Clock Slave (see chapter 6.4.5 for further details).

In order to initiate a transfer, the transmitter has to be enabled first by setting the *TxEn* bit in the *SPI_StatusAndCtrl* register. Then the data to be transferred has to be written to the *SPI_Tx* register and is loaded by the hardware into the TxShifter. If the data has been transferred from the *SPI_Tx* register into the TXShifter, the *XmtEmpty* flag signals that the *SPI_Tx* register is now empty and that new data can be loaded into the *SPI_Tx* register if needed (e.g. consecutive data tranfers).

Transfer in the above sense means that the data from the *SPI_Tx* register is transmitted on the *MOSI* pin, while at the same time data is received on the *MISO* pin. The received data is collected in the RxShifter and transferred into the *SPI_Rx* register after *DataLength* bits have been received.

If a transfer of one register content has been completed the user gets notified via the SPI interrupt which is visible and controllable in the Communication Interrupt Controller (CIC).

With the bitfield *DataLength* in the *SPI_StatusAndCtrl* register the number of bits to be transferred/received can be programmed. Note that the minimum number of bits to transfer/receive is 8bits, while the maximum is 16bits. Note also that the *DataLength* field always applies for transmit and receive at the same time.

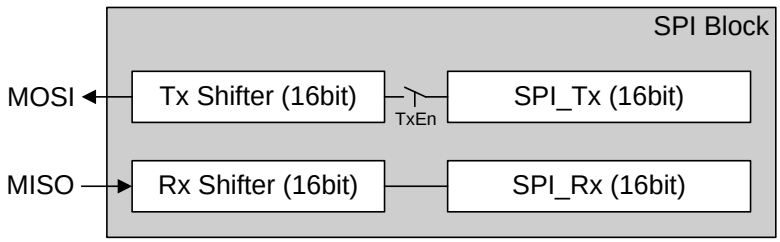


Figure 5-33: SPI Block

The *XmtDone* flag of *SPI_StatusAndCtrl* register can be used as a status flag in order to see when a transfer is ongoing and when it is completed. The *XmtDone* flag is "0" as long as the transfer is ongoing and "1" if the transfer is completed. Note that the *XmtDone* flag is cleared if the *TxEn* bit is set to "0".

After a SPI transmission, the content of the RxShifter is transferred to the *SPI_Rx* register. The *RxAvailable* flag in *SPI_StatusAndCtrl* register indicates that the received data is available in the *SPI_Rx* register. It is set back to zero by reading it. Note: If the *SPI_Rx* register is not read, then the new receive process will overwrite the content of the *SPI_Rx* register.

Data transfer order: If *MsbFirst*=0 then the LSB of the data (or *SPI_Tx*) is transmitted first (meaning Bit0 of the *SPI_Tx*). If *MsbFirst*=1 then the MSB of the the data (not the *SPI_Tx*) is transmitted first (depending on what is set in the *DataLength* field).

By setting the *Ss_n_afterTx* bit in the *SPI_StatusAndCtrl* register to "1" it is possible to do consecutive data transfers in a way that the corresponding slave is not de-asserted between the transfers. However in this case the following procedure has to be applied:

- Set the *TxEn* bit in the *SPI_StatusAndCtrl* register
- Write *Data* to the *SPI_Tx* register
- Wait until the bit *XmtEmpty* in the *SPI_StatusAndCtrl* register is "1" and then write new data in the *SPI_Tx* register. Note that it can take a few *SysClk* cycles until the *XmtEmpty* flag is being raised after the first write transfer has been initiated. This is due to the fact that the SPI and system clock domain can be different (*SPIClk* is typically slower than the *SysClk*).
- Method A (interrupt driven): In case the SPI interrupt occurs, read the *XmtEmpty* flag. If it is "1" write new data to the *SPI_Tx* register.

- Method B (polling): Always poll the *XmtEmpty* flag. If it is "1" write new data to the *SPI_Tx* register.

The last bullet (either Method-A or Method-B) shall be repeated until there is no more data to transfer for the consecutive data transfer. The user has to decide in his application which method is the most effective for him.

Timing diagrams for SPI operation can be found in [AD-04].

5.6 16-bit I/O Port

To access the I/O ports the *PIO_IO* register has to be read/written. Note that there is only one register for input and output. The *PIODirection* register defines the direction of the corresponding I/O port.

With the *PIOIntConfig* register it is also possible to feed up to four external interrupts into the AGGA-4.

See programming section for more details.

Note: After changing the *PIOIntConfig* register, the corresponding interrupt should be cleared in the Primary Interrupt Controller. A change of the configuration may generate a single interrupt.

To save pins, I/O pins are shared with other functions:

- *PIO*(0) and *PIO*(1) define the PROM width at boot time (see also *MCFG1* register), note that *PIO*(0) has to be tied to 0 during reset
- *PIO*(2) defines the EDAC usage for PROM at boot time (see also *MCFG3* register)
- *PIO*(8) is used as UART-1 CTSN signal if flow control is enabled in the *UART1_Ctrl*
- *PIO*(9) is used as UART-1 RTSN signal if flow control is enabled in the *UART1_Ctrl*
- *PIO(11)* is used as UART-1 TX signal (in addition to the dedicated *UART1_TX* pin) if the bit *TxEn* is enabled in the *UART1_Ctrl* register and provided that *PIO(11)* is configured as output.
- *PIO*(12) is used as UART-0 CTSN signal if flow control is enabled in the *UART0_Ctrl*
- *PIO*(13) is used as UART-0 RTSN signal if flow control is enabled in the *UART0_Ctrl*
- *PIO(15)* is used as UART-0 TX signal (in addition to the dedicated *UART0_TX* pin) if the bit *TxEn* is enabled in the *UART0_Ctrl* register and provided that *PIO(15)* is configured as output.

Reading the *PIO_IO* register always gives the actual value on the corresponding PIO pins. Writing the *PIO_IO* register always sets the value to be used when the corresponding PIO pin is in output mode.

Note: If flow control is used for the UART, then the PIO direction of the RTSN signal has to set as output and the CTSN signals has to be set as input in the *PIODirection* register.

5.7 GPIO

In addition to the 16 bit I/O port provided by the LEON on-chip peripherals further 16 general purpose input/output ports are implemented. All 16 GPIOs are implemented as bi-directional ports. This gives the user the possibility to configure the number of inputs or outputs according to his needs.

The *GPIO_Status* register can be used to get the current level on the particular *GPIO* pin, no matter if it is configured as input or output. A "1" represents a high level on the *GPIO* pin, while a "0" represents a low level on the *GPIO* input pin.

The *GPIO_Direction* register can be used to configure a particular *GPIO* pin as input or output. A "1" in the corresponding bit position of the *GPIO_Direction* register configures the pin as an output, while a "0" in the corresponding bit position of the *GPIO_Direction* register configures the pin as an input.

The *GPIO_Output* register can be used to assign a certain level to a particular GPIO pin. Note that the settings written to this register have only an effect if the *GPIO* pin is configured as output. Otherwise the programmed output value has no effect. A "1" written to the corresponding bit position of the *GPIO_Output* register assigns the corresponding *GPIO* pin to Vcc, while a "0" in the corresponding bit position of the *GPIO_Output* register assigns the corresponding *GPIO* pin to Ground. If the *GPIO_Output* register is read it returns the programmed values, not the current status of the *GPIO* pins.

5.8 Serial General Purpose Output (SGPO)

Beyond the standard UART ports described in section 5.1, a modified UART transmitter with a 16 byte FIFO is available, using the AGGA-4 pin *SGPO*. When the application software writes into *SGPO_Tx*, the hardware automatically splits the 32bit word into bytes and feeds 1, 2, 3 or 4 of them into the UART FIFO (least significant byte first). The *ByteSel* field in the *SGPO_Ctrl* register determines how many bytes of the 32bit word are transferred into the FIFO.

The application software has to take care that the FIFO has no overrun. The FIFO status can be read by reading the *SGPO_Status* register. An interrupt *SGPOOverrun* is generated, indicating that data has been lost. The interrupt is visible and controllable in bit 0 of the Communication Interrupt Controller (CIC).

It is also possible to disable the transmitter by setting the *TxEn* bit in the *SGPO_Ctrl* register to zero. With the configuration bits *PS* in the *SGPO_Ctrl* register the parity can be switched between even and odd. With the configuration bits *PE* in the *SGPO_Ctrl* register the parity can be enabled or disabled.

The baud rate can be programmed by writing to the *SGPO_Scaler* register. This 12bit divider is used to generate the baud rate from the LEON clock.

$$Scaler = \frac{LeonClk}{BaudRate * 8} - 1$$

6 System Support Functions

6.1 FFT Module

The FFT-Module performs the Fast Fourier Transformation using the radix-2 algorithm developed by Cooley and Tukey. The implemented FFT uses 128 points.

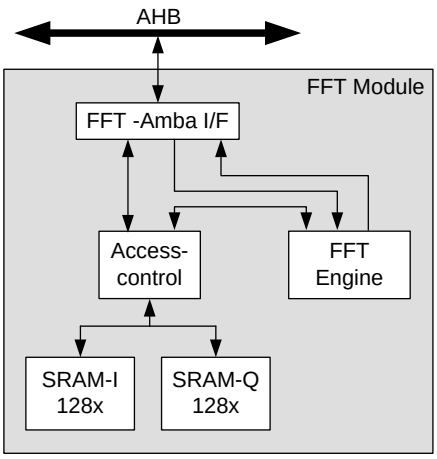


Figure 6-1: FFT Module

6.1.1 FFT-AHB interface

In order to guarantee high data transfer rates to the FFT module it is integrated as an AMBA module. The data transfer is realized via the AMBA bus. The FFT module is connected as a slave module to the AHB.

6.1.2 FFT Handling

The data written by the processor into the RAM is composed of real and imaginary data. The expected format is 32bit two's complement integer numbers. Since the FFT handles complex data, two RAMs are implemented. All data written at even address (0, 8, 10, ..), is written into the SRAM-I, which contains the real part of the value. Data written at odd address (4, C, 14, ..), is written into the SRAM-Q, which contains the imaginary part of the value. Note that the bit reversal of the sample indexes is done in hardware, so that the software can write the sequential input values onto sequential FFT RAM addresses. This sample re-ordering process is executed automatically as the values are written to the FFT RAM. This means that when a value is written to the address that should hold the *n*-th (sequentially ordered) sample, the value is actually written to the address that corresponds to the index *m*, where *m* is the value of the reversed binary representation of *n*. In other words, when the SW wants to write the *n*-th (zero-based indexing) input sample of the input it does so by issuing a write command to the FFT Module's base address (0xB0000000) plus an offset of 2*n**4 bytes, for the real part, or (2*n*+1)*4, for the imaginary part. However, this value is instead (transparently) written to the offset 2*m**4, for the real part, or (2*m*+1)*4, for the imaginary part, with *m* being the value of the bit-reversed *n*.

For example, the value written to the *FFTValue_1_real* position of the FFT RAM (*n*=1, '0000001' in binary) is actually written by the hardware to the *FFTValue_64_real position*. (*m*=64, '1000000' in binary).

After the write transfer is completed, the user should set the *StartFFT* bit in the *FFTCtrl* register. If this bit is set, the FFT engine starts and takes over the control of the access to the FFT RAMs. If the FFT has been started the FFT RAM must not be accessed by the processor or any other party then the FFT until the FFT is done. This is because the FFT module is using the FFT RAM also for intermediate calculation steps.

If the processor tries to access the FFT RAMs during the time they are under control of the FFT engine, an error response will be generated on the AHB. Reading the FFT control register is always possible.

After the FFT processing is finished, the FFT module generates an interrupt FFTdone in the Primary Interrupt Controller and the processor can fetch the processed values which are located in the same FFT RAM area where the input data has be written to. If the Interrupt is masked it can be used as a status flag. FFTdone is also visible in FFTCtrl registers (cleared when writing to FFTCtrl).

Note that there is no need from the software side to scale the inputs, in order to avoid an overflow within the FFT.

The outputs of the FFT are IEEE 754 single precision Floating Point Numbers. However, it should be noted that the internal implementation is a 55-bit (39 + 16) fixed point format, stripping the lower 16 decimal bits after each radix-2 butterfly. As a consequence, a data dependent rounding error is introduced which can lead to significant (>> 100%) relative errors for individual samples when comparing with a floating-point implementation. However, when normalising the relative error by the maximum absolute value, hence dividing by max [abs (FFT(i)], the error is bounded to a few percent, allowing the correct detection of the peak value to determine the Doppler shift during acquisition.

A Fast Fourier Transformation is executed in 5126 *SysClk* cycles.

6.2 CRC Module

The architecture of the CRC block is depicted in Figure 6-2. The core consists of a 32bit LFSR register and a 32bit polynomial register. Therefore up to 32bit CRC's can be calculated.

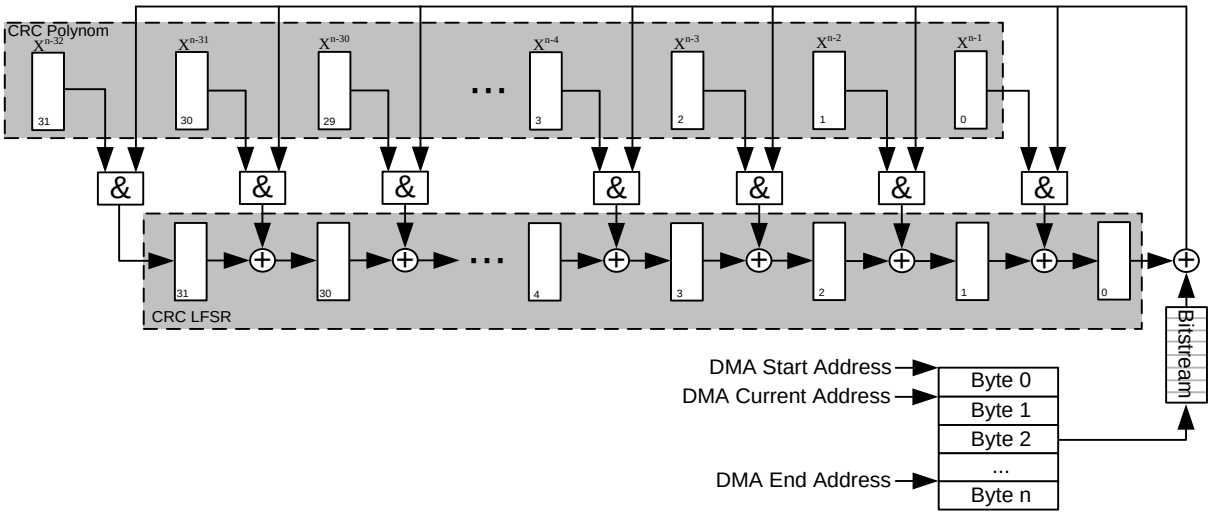


Figure 6-2: CRC Hardware Architecture

In order to set up the CRC, the polynomial must be programmed to the desired CRC type via the register *CRCPolynom*. Note that the highest order term has to be discarded. Example: In order to program the Polynomial $1 + x^5 + x^{12} + x^{16}$ into the register, the taps for x^0, x^5 and x^{12} have to be set to „1", while x^{16} is discarded. The polynomial must be right aligned and in reversed format. That means for a 16 bit CRC the bit 0 (rightmost bit) of the *CRCPolynom* register corresponds to x^{15} ,while for a 32 bit CRC the bit 0 of the *CRCPolynom* register corresponds to x^{31}. Table 6-1 gives some examples of the values which have to be programmed in order to get the desired CRC function.

The init value for the CRC calculation has to be written to the *CRCLFSR* register prior to every CRC start. When the CRC calculation has been done, the *CRCLFSR* register (containing the final CRC checksum) is XOR'ed with the value of the *CRCFinalXOR* register. After this, the software gets notified via interrupt. Additionally the readiness can be polled by masking the interrupt and polling the interrupt pending register.Note that the readiness can not be polled by observing the current pointer.

It has to be noted that not all CRC types require a final XOR multiplication. In that case the *CRCFinalXOR* register has to be programmed to zero. Note that the final XOR value has to be written only once in order to configure the type of CRC. It has not to be programmed every time a CRC is triggered.

By default, the CRC result in *CRCLFSR* is provided in normal format, i.e. bit 0 will contain the LSB of the CRC (= the x^0 term). For some CRC types it is required to reverse the CRC result after the final XOR multiplication (MSB<->LSB swap). This can be programmed by setting the bit *ReverseResult* in the *CRCCtrl* register to 1. In this case the content of the *CRCLFSR* register is bit-reversed and automatically right aligned by the hardware. E.g. a CRC of length 16 would be reversed and right shifted by 16, bit 0 will contain the x^{15} term.

In order to define the data which shall be processed the start and end address can be set with the *CRCStartAddress* respectively *CRCEndAddress* register. The *CRCStartAddress* has to be word (32bit) aligned, while the *CRCEndAddress* can be byte aligned. After a write to *CRCEndAddress* register, the CRC calculation is started. Therefore this register should be configured as the last one.

As soon as the *CRCEndAddress* register is written, the DMA machine will start and fetch the first byte and hand it over to the CRC calculation. The byte is feed bitwise into the CRC. With the bit *ReverseInData* in the *CRCCtrl* register it can be determined whether a byte is put from MSB to LSB or vice versa into the Bitstream FIFO. By reading the *CRCCurAddress* register it returns the address of the byte which is currently processed. Note that during CRC runtime the user shall not write any CRC register. Otherwise the CRC processing gets corrupted.

Example: In order to calculate the CRC over 6 bytes which are placed starting at address 0x40000000, the *CRCStartAddress* would have to be programmed to 0x40000000, the *CRCEndAddress* to 0x40000005.

CRC Type	Polynomial	CRCPolynom	CRCLFSR (Init)	CRCFinalXOR	Reverse InData	Reverse Result
CRC16-CCITT	$1 + x^5 + x^{12} + x^{16}$	0x00008408	0x0000FFFF	0x00000000	0	0
IBM-CRC-16	$1 + x^2 + x^{15} + x^{16}$	0x0000A001	0x00000000	0x00000000	1	1
CRC-32 (IEEE 802.3)	$1+ x^1+ x^2+ x^4+ x^5+ x^7+ x^8+ x^{10}+ x^{11}+ x^{12}+ x^{16}+ x^{22}+ x^{23}+ x^{26} x^{32}$	0xEDB88320	0xFFFFFFFF	0xFFFFFFFF	1	1

Table 6-1: CRC Examples

CRC16-CCITT refers to the implementation specified in the ITU-T Recommendation V.41 (1988) and in ECSS-E-70-41A (2003) or ECSS-E-ST-70-41C (2016). AGGA4 correctly meets the test cases specified in these standards.

Please note that other implmentations may exist, also labeled 'CRC-CCITT', producing different results, even though using the same polynomial.

6.3 EEPROM Support Function

In space hardware it is required to switch-off the EEPROM devices when they are not in use, because of the data retention issues. Therefore the AGGA-4 offers an EEPROM switch functionality as depicted in Figure 6-3.

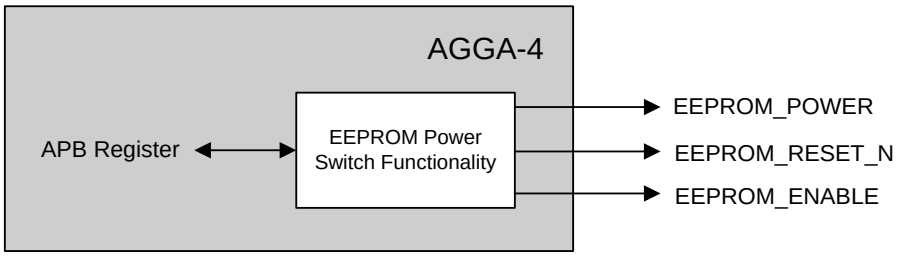


Figure 6-3: EEPROM Power Switch Functionality

By writing 0xAFFEDEAx (x= 0..F) in the *EEPROM_Switch* bitfield in the *EEPROM_StatusAndCtrl* register the switch on sequence is initiated. After switching on, the EEPROM_On bit indicates this as "1" and the EEPROM is powered.

By writing 0xDEADAFFx (x= 0..F) in the *EEPROM_Switch* bitfield in the *EEPROM_StatusAndCtrl* register the switch off sequence is initiated. After switching off, the EEPROM_On bit indicates this as "0" and the EEPROM is powered down after t4.

By reading the 4 LSB's of the *EEPROM_StatusAndCtrl* register the current status of the EEPROM control signals (EEPROM_On, *EEPROM_POWER*, *EEPROM_RESET_N* and *EEPROM_ENABLE*) can be read.

Note that all bits in EEPROM_StatusAndCtrl are 0 after reset, the EEPROM is disabled. The EEPROM Support Function can therefore not be used for a boot EEPROM.

The switching of the pins is performed in a particular sequence as depicted in Figure 6-4.

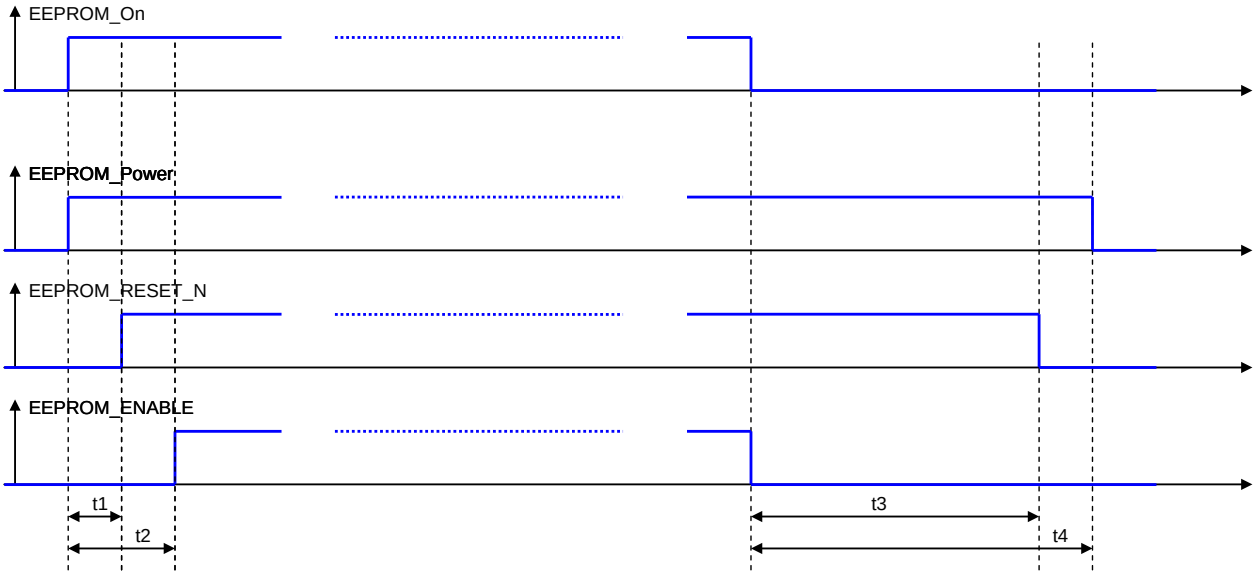


Figure 6-4: EEPROM Switching Sequence

The individual times ar as follow:

- T1 = 1ms
- T2 = 2ms
- T3 = 100ms
- T4 = 101ms

Note that the above specified times T1 to T4 are only true if the pin configurable *SYS_CLK_DIV* divider is set such that the output of the divider equals to 10 MHz.

6.4 Clock Distribution

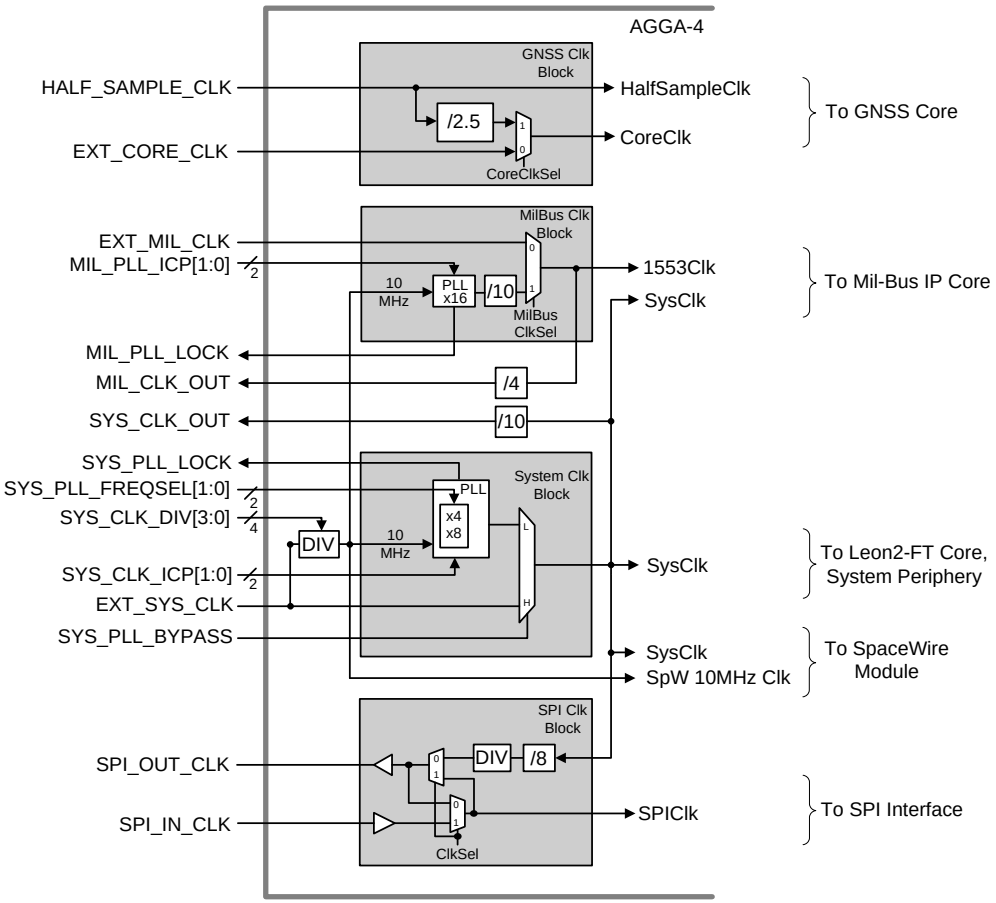


Figure 6-5: AGGA-4 Clock Distribution

The clock generation and distribution is shown in **Figure 6-5.** The minimum periods for the clock domains shown at the right side of the figure as well as other clocking constraints and recommendations. These minimum periods have to be respected from cycle-to-cycle, the jitter of the clock sources and the internal PLL's has to be considered, possibly leading to a reduced maximum frequency. Further details of the clock generation are provided in the following sub-sections.

Clock Domain	Constraint / Comment
CoreClk	Recommended frequency: 40 MHz
HalfSampleClk	Frequency shall be exactly 2.5 x CoreClk only in DDC Mode
SysClk	Frequency shall be >= CoreClk Note: SpaceWire transmission clock is related to SysClk
1553Clk	Recommended 16 MHz, to be compliant with MIL standard
SpW10MHzClk	Recommended 10 MHz, to be compliant with SpaceWire standard
SpWRxClk	Clock domain recovered from Data-Strobe input signals in SpaceWire
SPIClk	

Table 6-2: Clock domains

6.4.1 System Clock (*SysClk*) Divider

It is possible to divide the clock signal applied to *EXT_SYS_CLK* by an integer value. The division is programmable via the 4bit *SYS_CLK_DIV* external pin configuration. If all *SYS_CLK_DIV* pins are all logical low, the division factor is 1 (output clk = input clk). If the pin *SYS_CLK_DIV [0]* is high and *SYS_CLK_DIV [1:3]* are low, the division factor would be 2 and so on.

If any of the following functions is used, the divider should be programmed such that the output frequency is 10 MHz.

- System PLL
- MilBus PLL
- EEPROM support functions
- SpW interface

6.4.2 GNSS Core Clock (*CoreClk*) Generation

The GNSS Core can be provided with two clock inputs. The *HALF_SAMPLE_CLK* and the *EXT_CORE_CLK*. While the *HALF_SAMPLE_CLK* is directly fed into the GNSS core, the *CoreClk* can be derived from two sources. By the *CoreClkSel* bit in the *GNSSCoreClkCtrl* register it can be selected whether the user wants to generate the *CoreClk* from a 2.5 division of the *HALF_SAMPLE_CLK* or if he wants to select the *CoreClk* directly from the *EXT_CORE_CLK* input pin (default).

Note: Besides of being radiation hardened, the *CoreClkSel* bit is also protected by triple redundancy against SEU.

6.4.3 Mil-Bus Clock Generation

The MilBus IP Core is supplied with the two clocks: *1553Clk* and *SysClk*. The *1553Clk* has to be 16MHz in order to comply with the MIL-1553 data rate requirements. *1553Clk* can be taken directly from the input pin *EXT_MIL_CLK* or it can be generated with an internal PLL. This can be selected by the *MilBusClkSel* bit in the *MilBusClkCtrl* register. For monitoring purposes, *1553Clk* divided by 4 is provided at the output pin *MIL_CLK_OUT*.

If the PLL is selected, the PLL takes its input clock from the divided *EXT_SYS_CLK* pin. The PLL then multiplies the 10 MHz PLL input frequency by 16, thus generating 160 MHz. The 160 MHz is then divided by 10 in order to arrive with 16MHz for the *1553Clk*.

$$f_{1553Clk} = f_{EXT\_SYS\_CLK} / (\text{DivRatio} + 1) * 16 / 10$$

6.4.4 System Clock (*SysClk*) Generation

The Leon2-FT as well as the UARTs and the periphery are supplied with the *SysClk*. The *SysClk* can either be taken directly from the *EXT_SYS_CLK* pin or it can be taken from the PLL. The selection is done by the *SYS_PLL_BYPASS* pin. If the *SYS_PLL_BYPASS* is asserted logical high, the PLL is bypassed and the *SysClk* is taken from the *EXT_SYS_CLK* pin. If the *SYS_PLL_BYPASS* is asserted logical low, the PLL is enabled and the *SysClk* is driven by the PLL output. For monitoring purposes, *SysClk* divided by 10 is provided at the output pin *SYS_CLK_OUT*. With *SYS_PLL_FREQSEL* = 0, the multiplication factor of the system clock PLL is 4, with *SYS_PLL_FREQSEL* = 1, the multiplication factor is 8.

Note: The overall processing performance does not increase linearly with *SysClk*. External factors (e.g. SRAM Wait States) are also relevant, and therefore the overall performance needs to be analysed at system level.

Note: The *SysClk* frequency always has to be equal or greater than the GNSS *CoreClk* frequency. Otherwise a stable operation of the AGGA-4 system can not be guaranteed.

6.4.5 SPI Clock (*SPIClk*) Generation

Although the AGGA-4 is the SPI Master, it can also accept a clock input from a SPI slave. This is needed in order to properly communicate with Saphyrion RF Front End chips which use a modified SPI interface (clock is provided by SPI slave).

Therefore the AGGA-4 has a bit *ClkSel* in the *SPI_StatusAndCtrl* register by which selects between internal and external *SPIClk*.

- *ClkSel* = 1: *SPIClk* is taken directly from the pin *SPI_IN_CLK*. The incoming clock synchronized internally. A change of mode won't lead to glitches in the internal AGGA-4 logic, but the current transfer may be corrupted. Therefore it is recommended to change the clock direction only if no transfer is ongoing.

- *ClkSel* = 0: *SPIClk* is derived from *SysClk* as shown in **Figure 6-5**, pre-divided by 8 and further divided by the 8bit programmable *DivRatio* bitfield in the *SPI_ClkDivider* register. The divider ratio has to be programmed according to the following formula:

$$DivRatio = \frac{SystemClk}{8 \cdot SPIClk} - 1$$

6.4.6 SpW Clock Generation

The SpaceWire modules require two clock inputs. A *SpW10MHzClk* for start-up and the *SysClk* for fast transmission. The *SpW10MHzClk* is derived by the divided *EXT_SYS_CLK*.

6.4.7 PLL Lock Status

The Lock Status of the two PLLs (System and MilBus) can be monitored either via register or via the external pins *MIL_PLL_LOCK* and *SYS_PLL_LOCK*.

If read via register, the status of the Leon PLL can be read by reading the *LeonPLLStatus* bit in the *PLLStatus* register. The status of the MilBus PLL can be read by reading the *MilBusPLLStatus* bit in the *PLLStatus* register.

Note that after power on it takes 2400 *EXT_SYS_CLK* cycles until the *MilBusPLLStatus* is set.

6.5 Reset Mechanisms

The AGGA-4 has several reset possibilities which are depicted in Figure 6-6. Besides a global "Power On Reset" at pin *PWR_ON_RESET_N*, there are other reset possibilities which only reset parts of the AGGA-4. Each occurrence of such a partial reset is registered in the *ResetStatus* register. This allows the software to determine the reset source after a reset has occurred. Moreover, the upper bits of the *ResetStatus* register can be used by the software to store information which is not cleared by any reset, except power on reset.

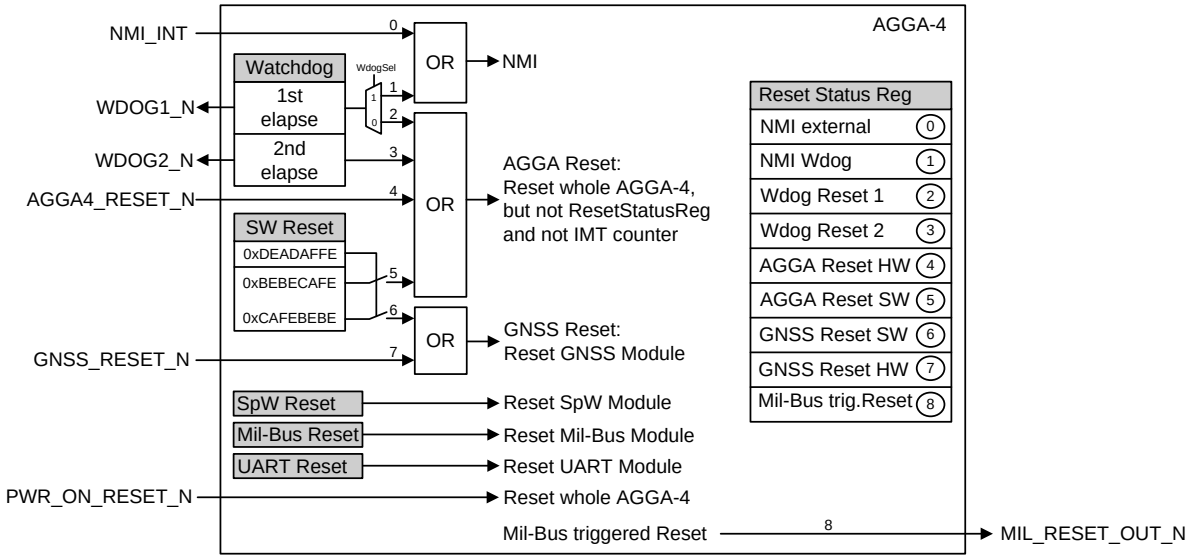


Figure 6-6: Reset Possibilities within AGGA-4

Most of the registers in AGGA-4 use a synchronous reset. Therefore it is generally required that all clocks are active during reset (see also section 8.4.4). After assertion of any of the global AGGA-4-reset sources (PWR_ON_RESET_N, AGGA4_RESET_N, SW Reset, Watchdog, but not the GNSS_RESET_N), the internal reset is kept asserted until the SysClk PLL has asserted its lock signal (if that PLL is used at all), plus an additional 128 SysClk cycles. When the SysClk PLL is disabled (bypassed), the delay is always 128 SysClk cycles. When the PLL loses lock (= de-asserts its lock signal) during operation, no reset will be generated. The 1553Clk PLL lock is not taken into account during reset generation. The *PrimaryRAM1/2* and *SecondaryRAM1/2* memories are not reset.

6.5.1 Power On Reset

Asserting *PWR_ON_RESET_N* to low level will restore the default values of all registers as specified in chapter 7, including the *ResetStatus* and IMT counter (*IMT_MSW, IMT_LSW*) registers, provided that the respective clocks are active. The registers which have no default value are marked as "undef" in the "Default" column in the programming section (chapter 7). The content of these registers is undefined after the Power On Reset. After deassert of the external power on reset, the AGGA-4 internal reset is de-asserted after 9 *EXT_SYS_CLK* cycles if SYS_PLL_BYPASS = 1 and 2209 *EXT_SYS_CLK* cycles if SYS_PLL_BYPASS = 0. After the system part is ready, the GNSS Core takes an additional 4 *CoreClk* cycles to reset.

6.5.2 Watchdog Reset

The watchdog consists of an 18 bit *WdogPrescaler* register and a 24 bit counter which is reloaded with the *WdogReload* value. The prescaler is clocked with the *SysClk*. With every time tick that is generated from the prescaler the pre loadable watchdog counter is decremented. The watchdog can be configured with a two-stage "bark and bite" function: If the counter elapses (= reaches zero) for the first and for the second time, depending on the setting of the *WdogSel* register, the following actions take place:

Action	1st elapse	2nd elapse
WdogSel = 0	AGGA-4 Reset is executed (bite)	Not applicable
WdogSel = 1	NMI interrupt is generated (bark)	AGGA-4 Reset is executed (bite)

Regardless of the *WdogSel* register, the two elapse's (bark and bite) are signalled by driving the open-drain output pins *WDOG1_N* and *WDOG2_N* to 0.

Writing to the *ReloadValue* register returns the state of the watchdog to the initial state (before the '1st elapse').

The watchdog can be disabled by setting the *ReloadValue* in the *WdogReload* register to zero (default state after reset).

Note that bit (15:0) of the *WdogPrescaler* register (the *PrescaleValue*) *are* mapped to bits (17:2) of the internal prescaler register which is used by the watchdog function, hence the prescaler division ratio is given as *PrescaleValue* *4 + 2. The default value of the *PrescaleValue* fieldafter reset is all "1".

There is a dedicated write enable register (*WdogWriteEnable*) which inhibits an accidental write to the prescaler or reload value of the watchdog. The user has to write the pattern "DEADAFFE" in the *WdogWriteEnable* register, before the write access to either the reload or the prescaler value is granted for one time. The write enable register gives write access to one register (reload/prescaler) for one time.

The Watchdog Elapse Time can be calculated with the following formula:

Watchdog Elapse Time = *SysClk*Period * ((*PrescaleValue* * 4) + 2) * (*ReloadValue* + 1)

6.5.3 AGGA-4 Reset

The AGGA-4 reset can be triggered by the Watchdog, the *AGGA4_RESET_N* input pin or by the software (Software Reset registers). The AGGA-4 reset will restore the default values of all registers as specified in chapter 7, except for the *ResetStatus* and IMT counter registers. The *ResetStatus* register allows the software to reconstruct the reset source.

With the IMT counter the software can reconstruct the blackout time after a reset. This can be used for a fast reacquisition of the GNSS signals after an AGGA-4 reset has occurred.

The duration of the AGGA-4 Reset is 1 *SysClk* (internal) cycle + 5 *EXT_SYS_CLK* cycles. After the system part is ready, the GNSS Core takes additional 4 *CoreClk* cycles to reset.

6.5.4 GNSS Reset

The GNSS reset can be triggered by the *GNSS_RESET_N* input pin or by the software (Software Reset registers SwResetEnable and SwResetExecute). If one of these sources triggers the GNSS reset, only the GNSS module is reset (including the IMT counter).

Note that during a GNSS reset the Leon shall not access any GNSS register. Otherwise the LEON will hang for the time of the GNSS reset.

The duration of the GNSS reset is 1 *SysClk* (internal) cycle + 4 *CoreClk* cycles

6.5.5 SW Reset

It is possible to trigger an AGGA-4 reset or GNSS reset by software. Therefore two registers are foreseen. With the first one (*SwResetEnable*) the reset can be armed by writing the pattern "0xDEADAFFE" into this register. Any other value in this register will inhibit a possible reset signal triggered by software. Note that the "Enable" is only valid for one write access to the *SwResetExecute* register.

The second register (*SwResetExecute*) executes the reset. It has to be noted that the AGGA-4 Reset is triggered by writing the pattern "0xBEBECAFE" into this register, while the GNSS Reset is triggered by writing "0xCAFEBEBE" into the register.

Any existing content or writes to the *SwResetExecute* register are ignored unless the register has been armed by previously writing "0xDEADAFFE" into the *SwResetEnable* register.

6.5.6 SpW Reset

The SpaceWire module has a dedicated bit to reset the SpaceWire module. This can be done by writing "1" to the *Reset* bit in the *SpW_ModuleConfig* register. There is no dedicated HW pin to reset only the Space Wire module, but the SpaceWire module is also reset by the above mentioned global AGGA-4 resets (except GNSS Reset).

6.5.7 Mil-Bus Reset

The Mil-Bus module has a dedicated bit to reset the Mil-Bus module. This can be done by writing "1" to the *Rst1553* bit in the *C53CDST*. There is no dedicated HW pin to reset only the Mil-Bus module, but the Mil-Bus module is also reset by the above mentioned global AGGA-4 resets (except GNSS Reset).

6.5.8 UART Reset

The UART module can be reset by writing "1" to the *Reset* bit in the *UART_Reset*. There is no dedicated HW pin to reset only the UART module, but the UART module is also reset by the above mentioned global AGGA-4 resets (except GNSS Reset)..

6.5.9 Mil-Bus triggered Reset

By sending the Mil-Bus mode command 8-"Reset Remote Terminal" to the AGGA-4, the AGGA-4 will drive the open-drain output pin *MIL_RESET_OUT_N* to level low for one *1553Clk* cycle. By connecting the *MIL_RESET_OUT_N* pin to one of the AGGA-4 reset input pins it is possible to force an AGGA-4 reset from the Mil-Bus bus controller to which the AGGA-4 is connected to.

6.5.10 Reset Status Register

The default value (after Power On Reset) of the *ResetStatusRegister* is all "1". If any 'warm' reset (= except for the Power On Reset) occurs, this register is not cleared, only the corresponding bit in the *ResetStatusRegister* is set to zero. This allows SW to distinguish after a reset cycle which reset was executed. The software should set back the all bits to "1" after a reading this register.

6.5.11 Processor reset

The processor is reset by the *PWR_ON_RESET_N* or by any of the sources triggering the AGGA-4 reset. After reset, the processor will start fetching from address 0. In the Processor Status Register (PSR), traps are disabled (ET = 0), supervisor mode is enabled (S = 1), all other fields are undefined or keep their previous value. As indicated in the *CacheCtrl* register, caches / burst fetch and snoop are also disabled. Caches need to be flushed before enabling.

6.6 AMBA BUS

6.6.1 AHB priorities

There are 7 bus masters on the AGGA-4 AMBA AHB Bus with the following priorities:

Priority		Participant
highest	6	DSU
	5	Milbus 1553 Module
	4	GNSS Module
	3	UART Module
	2	SpW Module
	1	CRC Module
lowest	0	Processor

Table 6-3: AMBA Priorities

Note that an AHB burst transaction, once started, can not be interrupted by a higher priority master. This means that a even a low priority master can lock the bus for considerable time. Extreme case would be a CPU cache-line fetch (burst of 8), to be multiplied with the duration of a single access (see section 6.6.2 below) from 8-bit PROM with maximum (30) wait-states, EDAC on: The latency would be 8*5*(30+2) = 1280 cycles. Even longer latencies are possible if accesses conditioned *BRDY_N* are used.

The following transaction types and bursts are used by the different masters:

- The processor uses 8x burst for cache line fetches and 2x burst for load / store double instructions
- The SpW RX DMA and SpW TX DMA uses 2x burst whenever possible (however, see the issue described in section 8.4.6). The SPW DMA performs one 32-bit AHB access per 4 SPW characters
- The UART DMA uses single 32-bit accesses, no burst. Note that the UART-RX and UART-TX performs one 32-bit AHB access per received byte .
- The CRC Module uses single 32-bit accesses, no burst.
- GNSS Module does not perform burst. The spacing between two transfers. See also description in 3.4.7.4
- Mil1553 Module perform 32-bit accesses, no burst

6.6.2 AHB/APB bus access duration

The duration of an AHB access depends on the area of the address map (see section 7.1) which is accessed, width of the AHB access (8, 16 or 32 bit), on the width of the external memory (8 or 32 bit), the configuration (wait-states) and on the ratio of *SysClk* / *CoreClk* frequencies. The duration – in number of *SysClk* cycles – can be calculated as follows:

- For accesses to PROM, RAM and memory-mapped IO, the duration of a single external access is two (2) plus the wait states which are either configured in *MCFG1* and *MCFG2*, or the input pin *BRDY_N* can be used to impose wait states. However, as explained in the following bullets, depending on access width, memory bus width one single AHB access, depending on access width, memory width, and settings may result in several external accesses.
- For PROM, up to 30 wait states can be configured in *MCFG1*, hence a single acess takes up to 32 cycles. One 32-bit AHB read from an 8-bit wide PROM with EDAC enabled results in 5 accesses (4 data and 1 parity byte), hence up to 160 wait states can be generated.
- For memory mapped I/O area, up to 15 wait states can be configured in *MCFG1*. The IO area shall always be accessed with AHB accesses matching the IO bus width (hence using LDUB/STB instruction for 8-bit IO and LD/ST for 32-bit IO), and one AHB transaction always results in one single I/O access.
- For RAM, up to 4 wait states can be configured in *MCFG2*, hence a single access takes up to 6 cycles. One 32-bit AHB access to 8-bit RAM with EDAC generates 5 memory accesses. Sub-word (8 / 16-bit) writes to RAM with EDAC are executed as Read-Modify-Write transactions, resulting in two accesses with 32-bit RAM, and 10 memory accesses with 8-bit RAM (read 4 bytes + 1 parity, write 4 bytes + 1 parity).
- For APB registers, the duration is always 2 cycles.

- Accesses to the registers located in the GNSS module are crossing the clock domain, and their duration depends on the *SysClk* / *CoreClk* frequency ratio and phase. The user can check the timing with the Trace Buffer.
- Accesses to the FFT module takes 2 cycles for the data memory and 1 cycle for the *FFTCtrl* register.

6.7 Test Support Pins

In order to support the application debugging, some internal signals can be accessed via test pins. These points are shown in **Figure 3-30**, highlighted in red.

Note: **These signals are not for flight use.** They are combinatorial outputs, and output delay on these signals may vary largely over temperature, supply, from test point to test point, and from channel to channel between 7 – 27 ns (compared to *EXT_CORE_CLK* in PLL Bypass mode). Even at constant temperature / voltage, the skew between these signals can be up to 5 ns. This can lead to a cycle slip which should be considered when interpreting the signal / code outputs.

6.7.1 I/Q Data

The 3bit I and 3bit Q data can be accessed just before and just after the Final Down Converter of a particular channel. The I/Q data is visible on the pins *SIGNAL_OUT_I[2:0]* and *SIGNAL_OUT_Q[2:0]*.

The selection of the channel to monitor is done by the programming of the *SignalOutChanSel* bitfield in the *TestSettings* register.

The selection of monitoring before or after the Final Down Converter is done by the programming of the *SignalOutPosSel* bitfield in the *TestSettings* register.

Note: If a certain channel shall be monitored the user has to make sure that the channel is activated in the *ChActivation0 or ChActivation1* register.

6.7.2 IntEpoch and Code Out

It is possible to monitor the Integration Epoch and the punctual delay line output of two independent channels.

The Integration Epoch of the first selected channel is visible on the pin *INT_EPOCH1*, the Integration Epoch of the second selected channel is visible on the pin *INT_EPOCH2*. The selection of the Integration Epoch or Long Epoch is done by programming the bit *IntSourceSel* of *CorrUnitCtrl*.

The punctual delay line output of the first selected channel is visible on the pins *CODE_OUT1[1:0]*, the punctual delay line output of the second selected channel is visible on the pins *CODE_OUT2[1:0]*.

The selection of the channel to monitor is done by the programming of the *CodeOut1Sel* bitfield and the *CodeOut2Sel* bitfield in the *TestSettings* register.

Note: If a certain channel shall be monitored the user has to make sure that the channel is activated in the *ChActivation0 or ChActivation1* register.

6.7.3 IMT_12

The IMT_12 pin connects to the bit 12 of the IMT count (defined from 0 to 63). The IMT count itself is clocked by the *CoreClk*. Therefore this pin gives a mean to check if the AGGA sees the *CoreClk* and if it the *CoreClk* has the desired frequency.

Example: A 40 MHz clock at the *EXT_CORE_CLK* will result frequency of 40 MHz / 2^{13} =~ 4.88kHz on *IMT_12*.

Note: If the *IMT_12* is observed as a frequency the divisor is 2^{13} rather than 2^{12} as one might think. This can be explained by the following example: Imagine the LSB would be observed. Then it changes its value with every *CoreClk* but its frequency is *CoreClk/2*.

7 Programming

This section lists the registers needed to program and control the AGGA-4.

This section uses the following conventions:

- 'undef' means not affected by any of the resets and content is undefined at power-up.
- R means Read-only , writing has not effect unless explicitly mentioned
- W means Write-only, the Read value has no meaning with no side effect unless explicitly mentioned
- RW means Read-Write
- 'reserved' means writing has no effect, and reading value has no meaning

7.1 Overall Address Map

Address Range	Size [Mbytes]	Mapping	Module
0x00000000 - 0x1FFFFFFF	512	PROM	Memory Controller
0x20000000 - 0x3FFFFFFF	512	I/O	
0x40000000 - 0x7FFFFFFF	1024	SRAM	
0x80000000 - 0x8FFFFFFF	256	Processor, Interface and System Support registers (see section 7.2)	AHB-APB bridge
0x90000000 - 0x9FFFFFFF	256	DSU (see section 7.3)	DSU Module
0xA0000000 - 0xAFFFFFFF	256	GNSS (see section 7.4)	GNSS Module
0xB0000000 - 0xBFFFFFFF	256	FFT (see section 7.5)	FFT Module
0xC0000000 - 0xFFFFFFFF	1024	Unused (see Note below)	

Table 7-1: Overall Address Map

Note: An access to an empty/unused AHB/APB address location triggers the AHB default slave (see AHB status register, chapter 4.15)

7.2 APB Address Map (Processor, Interface and System Support)

7.2.1 Overview

Address	Size [bits]	Name
0x80000000	31	MCFG1
0x80000004	19	MCFG2
0x80000008	32	MCFG3
0x8000000C	28	AHBFailingAddress
0x80000010	10	AHBStatus
0x80000014	32	CacheCtrl
0x80000018	32	PowerDown
0x8000001C	32	WriteProtect1
0x80000020	32	WriteProtect2
0x80000024	31	LeonConfig
0x80000028	N/A	Reserved
0x8000002C	N/A	Reserved
0x80000030	N/A	Reserved
0x80000034	N/A	Reserved

Address	Size [bits]	Name
0x80000038	N/A	Reserved
0x8000003C	N/A	Reserved
0x80000040	32	Timer1_Counter
0x80000044	32	Timer1_Reload
0x80000048	3	Timer1_Ctrl
0x8000004C	N/A	reserved
0x80000050	32	Timer2_Counter
0x80000054	32	Timer2_Reload
0x80000058	3	Timer2_Ctrl
0x8000005C	N/A	Reserved
0x80000060	10	TimerPrescaleCounter
0x80000064	10	TimerPrescaleReload
0x80000068	N/A	Reserved
0x8000006C	N/A	Reserved
0x80000070	32	Timer3_Counter
0x80000074	32	Timer3_Reload
0x80000078	3	Timer3_Ctrl
0x8000007C	N/A	Reserved
0x80000080	32	Timer4_Counter
0x80000084	32	Timer4_Reload
0x80000088	3	Timer4_Ctrl
0x8000008C	N/A	Reserved
0x80000090	32	PrimIntMaskAndPrio
0x80000094	16	PrimIntPending
0x80000098	16	PrimIntForce
0x8000009C	16	PrimIntClear
0x800000A0	16	PIO_IO
0x800000A4	16	PIODirection
0x800000A8	32	PIOIntConfig
0x800000AC	N/A	Reserved
0x800000B0	18	WdogPrescale
0x800000B4	24	WdogReload
0x800000B8	1	WdogSel
0x800000BC	32	WdogWriteEnable
0x800000C0	N/A	Reserved
0x800000C4	7	DSU_UART_Status
0x800000C8	2	DSU_UART_SpW_Ctrl
0x800000CC	14	DSU_UART_Scaler
0x800000D0	30	WriteProtectStartAddress3
0x800000D4	30	WriteProtectEndAddress3
0x800000D8	30	WriteProtectStartAddress4
0x800000DC	30	WriteProtectEndAddress4
0x800000E0	16	GPIO_Status
0x800000E4	16	GPIO_Output
0x800000E8	16	GPIO_Direction
0x800000EC	9	ResetStatus
0x800000F0	32	SwResetEnable
0x800000F4	32	SwResetExecute
0x800000F8	6	MilBusRTAddress
0x800000FC	12	AGGA4Version

Address	Size [bits]	Name
0x80000100	13	CIC_Mask
0x80000104	13	CIC_Pending
0x80000108	9	CIC_Clear
0x8000010C	N/A	Reserved
0x80000110	19	SPI_StatusAndCtrl
0x80000114	8	SPI_ClkDivider
0x80000118	16	SPI_Tx
0x8000011C	16	SPI_Rx
0x80000120	32	CRCLFSR
0x80000124	32	CRCPolynom
0x80000128	32	CRCFinalXOR
0x8000012C	2	CRCCtrl
0x80000130	32	CRCStartAddress
0x80000134	32	CRCEndAddress
0x80000138	32	CRCCurAddress
0x8000013C	N/A	reserved
0x80000140	27	C53CF
0x80000144	26	C53EMBA
0x80000148	32	C53CDST
0x8000014C	9	C53NIT
0x80000150	5	C53EIT
0x80000154	1	C53RIT
0x80000158	N/A	reserved
0x8000015C	32	C53RTI
0x80000160	32	C53TTI
0x80000164	N/A	reserved
0x80000168	N/A	reserved
0x8000016C	N/A	reserved
0x80000170	32	SGPO_Tx
0x80000174	6	SGPO_Status
0x80000178	5	SGPO_Ctrl
0x8000017C	12	SGPO_Scaler
0x80000180	32	UART0_Tx_SAP
0x80000184	32	UART0_Tx_EAP
0x80000188	32	UART0_Tx_CAP
0x8000018C	32	UART0_Rx_SAP
0x80000190	32	UART0_Rx_EAP
0x80000194	32	UART0_Rx_CAP
0x80000198	8	UART0_Status
0x8000019C	10	UART0_Ctrl
0x800001A0	29	UART0_Scaler
0x800001A4	32	UART1_Tx_SAP
0x800001A8	32	UART1_Tx_EAP
0x800001AC	32	UART1_Tx_CAP
0x800001B0	32	UART1_Rx_SAP
0x800001B4	32	UART1_Rx_EAP
0x800001B8	32	UART1_Rx_CAP
0x800001BC	8	UART1_Status
0x800001C0	10	UART1_Ctrl
0x800001C4	29	UART1_Scaler

Address	Size [bits]	Name
0x800001C8	1	UART_Reset
0x800001CC	N/A	reserved
0x800001D0	N/A	reserved
0x800001D4	N/A	reserved
0x800001D8	N/A	reserved
0x800001DC	N/A	reserved
0x800001E0	1	GNSSCoreClkCtrl
0x800001E4	N/A	reserved
0x800001E8	1	MilBusClkCtrl
0x800001EC	2	PLLStatus
0x800001F0	N/A	reserved
0x800001F4	N/A	reserved
0x800001F8	32	EEPROM_StatusAndCtrl
0x800001FC	N/A	Reserved
0x80000200	13	SpW0_StatusAndCtrl
0x80000204	N/A	Reserved
0x80000208	32	SpW0_Tx_SAP
0x8000020C	32	SpW0_Tx_EAP
0x80000210	32	SpW0_Tx_CAP
0x80000214	9	SpW0_Tx_Rx_Config
0x80000218	32	SpW0_Rx_SAP
0x8000021C	32	SpW0_Rx_EAP
0x80000220	32	SpW0_Rx_CAP
0x80000224	N/A	Reserved
0x80000228	N/A	Reserved
0x8000022C	N/A	Reserved
0x80000230	N/A	Reserved
0x80000234	N/A	Reserved
0x80000238	N/A	Reserved
0x8000023C	N/A	Reserved
0x80000240	13	SpW1_StatusAndCtrl
0x80000244	N/A	Reserved
0x80000248	32	SpW1_Tx_SAP
0x8000024C	32	SpW1_Tx_EAP
0x80000250	32	SpW1_Tx_CAP
0x80000254	9	SpW1_Tx_Rx_Config
0x80000258	32	SpW1_Rx_SAP
0x8000025C	32	SpW1_Rx_EAP
0x80000260	32	SpW1_Rx_CAP
0x80000264	N/A	Reserved
0x80000268	N/A	Reserved
0x8000026C	N/A	Reserved
0x80000270	N/A	Reserved
0x80000274	N/A	Reserved
0x80000278	N/A	Reserved
0x8000027C	N/A	Reserved
0x80000280	13	SpW2_StatusAndCtrl
0x80000284	N/A	Reserved
0x80000288	32	SpW2_Tx_SAP
0x8000028C	32	SpW2_Tx_EAP

Address	Size [bits]	Name
0x80000290	32	SpW2_Tx_CAP
0x80000294	9	SpW2_Tx_Rx_Config
0x80000298	32	SpW2_Rx_SAP
0x8000029C	32	SpW2_Rx_EAP
0x800002A0	32	SpW2_Rx_CAP
0x800002A4	N/A	Reserved
0x800002A8	N/A	Reserved
0x800002AC	N/A	Reserved
0x800002B0	N/A	Reserved
0x800002B4	N/A	Reserved
0x800002B8	N/A	Reserved
0x800002BC	N/A	Reserved
0x800002C0	13	SpW3_StatusAndCtrl
0x800002C4	N/A	Reserved
0x800002C8	32	SpW3_Tx_SAP
0x800002CC	32	SpW3_Tx_EAP
0x800002D0	32	SpW3_Tx_CAP
0x800002D4	9	SpW3_Tx_Rx_Config
0x800002D8	32	SpW3_Rx_SAP
0x800002DC	32	SpW3_Rx_EAP
0x800002E0	32	SpW3_Rx_CAP
0x800002E4	N/A	reserved
0x800002E8	N/A	reserved
0x800002EC	N/A	reserved
0x800002F0	N/A	reserved
0x800002F4	N/A	reserved
0x800002F8	N/A	reserved
0x800002FC	N/A	reserved
0x80000300	2	SpW_ModuleConfig
0x80000304	2	SpW_ModuleTimeCtrl
0x80000308	8	SpW_ModuleTimeCode
0x8000030C	32	SpW_ModuleIntMask
0x80000310	32	SpW_ModuleIntStatus
0x80000314	32	SpW_ModuleIntClear

7.2.2 LEON and Memory Interface Registers

7.2.2.1 MCFG1 (Memory Configuration Register 1)

MCFG1				
Bit	**Field**	**Default**	**R/W**	**Description**
31	reserved	0	R	reserved
30	PBRDYN	0	R/W	PROM area bus ready enable (PBRDYN). If set, a PROM access will be extended until *BRDY_N* is asserted
29	ABRDYN	0	R/W	Asynchronous bus ready (ABRDYN). If set, the BRDY_N input can be asserted without relation to the *SysClk*
28-27	IOBusWidth	undef	R/W	Defines the data with of the I/O area 00 = 8 bit 10 = 32 bit 01, 11 = reserved (bit 27 shall always be written 0)

MCFG1				
Bit	**Field**	**Default**	**R/W**	**Description**
26	BusReadyEn	0	R/W	1 = Bus ready (*BRDY_N*) enable for IO accesses 0 = Bus ready (*BRDY_N*) disable for IO accesses
25	BusErrorEn	0	R/W	1 = Bus error (*BEXC_N*) enable for IO accesses 0 = Bus error (*BEXC_N*) disable for IO accesses
24	reserved	0	R	Reserved
23-20	IOWaitStates	undef	R/W	0 = 0 waitstates 1 = 1 waitstate … 15 = 15 waitstates
19	IOEnable	0	R/W	0 = the access to the memory bus I/O area is disabled 1 = the access to the memory bus I/O area is enabled
18-12	reserved	0	R	Reserved
11	PROMWriteEn	0	R/W	0 = disables write access to the PROM area 1 = enables write access to the PROM area
10	reserved	0	R	Reserved
9-8	PROMWidth	PIO[1:0]	R/W	Defines the data with of the PROM area 00 = 8 bit width 10 = 32 bit width 01, 11 = reserved (bit 8 shall always be written 0) At reset PROMWidth ist set with the values on the PIO input pins. Bit 9 is set with the value on the PIO[1] pin. Bit 8 is set with the values on the PIO[0] pin. PIO[0] has to be tied to 0.
7-4	PROMWriteWs	15	R/W	Defines the number of wait states during PROM write cycles 0 = 0 waitstates 1 = 2 waitsates … 15 = 30 waitstates
3-0	PROMReadWs	15	R/W	Defines the number of wait states during PROM read cycles 0 = 0 waitstates 1 = 2 waitsates … 15 = 30 waitstates

7.2.2.2 MCFG2 (Memory Configuration Register 2)

MCFG2				
Bit	**Field**	**Default**	**R/W**	**Description**
31-13	reserved	0	R	Reserved
12-9	RAMBankSize	undef	R/W	0 = 2^0 x 8 kByte (8 kByte) 1 = 2^1 x 8 kByte (16 kByte) 2 = 2^2 x 8 kByte (32 kByte) … 14 = 2^14 x 8 kbyte (128 Mbyte) 15 = 2^15 x 8 kbyte (256 Mbyte) The 4 banks decoded by *RAMS_N[0..3]* are always contiguous, hence their address range depends on this field Field not initialised at reset
8	reserved	0	R	Reserved
7	BusReadyEn	undef	R/W	0 = disable *BRDY_N* for *RAMS_N*[4]

MCFG2				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = enable *BRDY_N* for *RAMS_N*[4] bitfield is not configured after reset
6	ReadModifyWriteEn	undef	R/W	Enable read-modify-write cycles on sub-word writes to 32-bit areas with common write strobe *(RWEN_N will always be "0000")* bitfield is not configured after reset
5-4	RAMWidth	undef	R/W	00 = 8 bit width 10 = 32 bit width 01, 11 = reserved (bit 4 shall always be written 0) bitfield is not configured after reset
3-2	RAMWriteWaitstates	undef	R/W	RAM Write Waitstates (0..3)
1-0	RAMReadWaitstates	undef	R/W	RAM Read Waitstates (0..3)

7.2.2.3 MCFG3 (Memory Configuration Register 3)

MCFG3				
Bit	**Field**	**Default**	**R/W**	**Description**
31-30	RFC	0	R	Register file check bits (RFC). Zero check bits are used, sibce register file is implemented with hard flip flops
29-28	reserved	0	R	Reserved
27	ME	1	R	0 = memory EDAC is not present 1 = memory EDAC is present
26-12	reserved	0	R	Reserved
11	WB	0	R/W	0 = no write bypass for EDAC diagnostic 1 = write bypass for EDAC diagnostic
10	RB	0	R/W	0 = no read bypass for EDAC diagnostic 1 = read bypass for EDAC diagnostic
9	RE	undef	R/W	0 = disable EDAC checking of the RAM area 1 = enable EDAC checking of the RAM area
8	PE	PIO[2]	R/W	0 = disable EDAC checking of the PROM area 1 = enable EDAC checking of the PROM area At reset this bit is initialized with the value of PIO[2]
7-0	TCB	undef	R/W	Test Check Bits. This field replaces the normal check bits during store cycles when WB is set. TCB is also loaded with the memory check bits during load cycles when RB is set

7.2.2.4 AHBFailingAddress

AHBFailingAddress				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	Address	undef	R	AHB failing address

7.2.2.5 AHBStatus

AHBStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	Reserved
9	CE	undef	R/W	EDAC correctable error. This bit is set when a correctbale EDAC error is detected

AHBStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
				This bit has to be cleared by the software
8	NE	0	R/W	New Error. This bit is set when a new error occurred. *AHBStatus* and *AHBFailingAddress* remain locked (no new error is recorded) until this bit is cleared by SW.
7	RW	undef	R	Read/Write. This bit is set if the failed access was a read cycle, otherwise it is zero
6-3	HMASTER	undef	R	AHB master. This field contains the master that generated the failed access 0 = Processor 1 = CRC Module 2 = SpW Module 3 = UART_DMA 4 = GNSS Module 5 = Milbus 6 = DSU 7-15 = reserved
2-0	HSIZE	undef	R	This field contains the size of the failed transfer 0 = 1 Byte 1 = 2 Byte 2 = 4 Byte 3 = 8 Byte 4-7 = reserved

7.2.2.6 CacheCtrl

CacheCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-30	DREPL	1	R	Data cache replacement policy (DREPL) 1 = Pseudo Random
29-28	IREPL	1	R	Instruction cache replacement policy (IREPL) 1 = Pseudo Random
27-26	ISETS	3	R	3 = 4-way associative
25-24	DSETS	1	R	1 = 2-way associative
23	DS	0	R/W	0 = disable data cache snooping 1 = enable data cache snooping
22	FD	0	W	0 = data cache is not flushed 1 = data cache is flushed
21	FI	0	W	0 = instruction cache is not flushed 1 = instruction cache is flushed
20-19	CPC	2	R	2 = 2 cache parity bits (CPC) are used to protect the caches
18-17	CPTE	undef	R/W	Cache parity test bits. (CPTE). These bits are XOR'ed to the data and tag parity bits during diagnostic writes.
16	IB	0	R/W	0 = burst fill during instruction fetch disabled 1 = burst fill during instruction fetch enabled
15	IP	0	R	Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress
14	DP	0	R	Data cache flush pending (DP). This bit is set when a data cache flush operation is in progress

CacheCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
13-12	ITE	undef	R/W	Instruction cache tag error counter (ITE). This filed is incremented every time an instruction cache tag parity error is detected
11-10	IDE	undef	R/W	Instruction cache data error counter (IDE) - This field is incremented each time an instruction cache data sub-block parity error is detected
9-8	DTE	undef	R/W	Data cache tag error counter (DTE) - This field is incremented every time a data cache tag parity error is detected. Does not overflow, stops at '11'. Can be cleared by writing zeroes into the corresponding bits
7-6	DDE	undef	R/W	Data cache data error counter (DDE) - This field is incremented each time an instruction cache data sub-block parity error is detected. Does not overflow, stops at '11'. Can be cleared by writing zeroes into the corresponding bits
5	DF	undef	R/W	Data Cache Freeze on Interrupt (DF) - If set, the data cache will automatically be frozen when an asynchronous interrupt is taken
4	IF	undef	R/W	Instruction Cache Freeze on Interrupt (IF) - If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken
3-2	DCS	0	R/W	Data Cache state. Indicates the current data cache state Note: Always flush the data cache before enabling 0 = disabled 1 = frozen 2 = disabled 3 = enabled
1-0	ICS	0	R/W	Instruction Cache state. Indicates the current data cache state Note: Always flush the instruction cache before enabling 0 = disabled 1 = frozen 2 = disabled 3 = enabled

7.2.2.7 PowerDown

PowerDown				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	value	0	R/W	write of arbitrary value causes power down of Integer Unit

7.2.2.8 LeonConfig

LeonConfig				
Bit	**Field**	**Default**	**R/W**	**Description**
31	reserved	0	R	Reserved
30	DSUPresent	1	R	DSU is present in AGGA-4
29	SDRAMPresent	0	R	SDRAM Controller is not present in AGGA-4
28-26	NrOfWatchpoints	4	R	Four Watchpoints are implemented in the AGGA-4
25	UMACPresent	0	R	UMAC/SMAC instruction is not implemented in the AGGA-4
24-20	NWINDOWS	7	R	There are 8 SPARC register windows implemented in AGGA-4

LeonConfig				
Bit	**Field**	**Default**	**R/W**	**Description**
19-17	ICSZ	3	R	Instruction cache set size. The size (in Kbytes) of the instruction cache set. Cache set size = $2^{ICSZ} = 2^3 = 8$ kByte AGGA4 has 4 instruction cache sets (*CacheCtrlReg.ISETS*), hence the total data cache size is 32 kByte
16-15	ILSZ	3	R	Instruction cache line size (in 32-bit words) = $2^{ILSZ} = 2^3 = 8$
14-12	DCSZ	3	R	Data cache set size. The size (in Kbytes) of the data cache set. Cache set size = $2^{DCSZ} = 2^3 = 8$ kByte AGGA4 has 2 data cache sets (*CacheCtrlReg.DSETS*), hence the total data cache size is 16 kByte
11-10	DLSZ	2	R	Data cache line size (in 32-bit words) = $2^{DLSZ} = 2^4 = 4$
9	UDIVPresent	1	R	UDIV/SDIV instruction is implemented in AGGA-4
8	UMULPresent	1	R	UMUL/SMUL instruction is implemented in AGGA-4
7	LeonWdgPresent	0	R	Leon standard watchdog is not implemented in AGGA-4 AGGA-4 has an advanced watchdog implemented
6	MemStatReg	1	R	Memory status and failing address register present
5-4	FPUType	2	R	Gaisler FPU is implemented in AGGA-4
3-2	PCICoreType	0	R	There is no PCI interface in the AGGA-4
1-0	WriteProtectType	1	R	Standard Write protection type is implemented in AGGA-4

7.2.3 Timer Registers

7.2.3.1 TimerN_Counter

The register description is valid for the following registers: *Timer1_Counter, Timer2_Counter, Timer3_Counter,* and *Timer4_Counter.*

TimerN_Counter				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	CounterValue	undef	R/W	read: current counter value write: counter value is modified once

7.2.3.2 TimerN_Reload

The register description is valid for the following registers: *Timer1_Reload, Timer2_Reload, Timer3_Reload,* and *Timer4_Reload.*

TimerN_Reload				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	ReloadValue	undef	R/W	Timer division ratio = ReloadValue + 1

7.2.3.3 TimerN_Ctrl

The register description is valid for the following registers: *Timer1_Ctrl, Timer2_Ctrl, Timer3_Ctrl,* and *Timer4_Ctrl.*

TimerN_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-3	reserved	0	R	Reserved
2	LD	undef	W	0 = do nothing 1 = load the reload value into the counter register
1	RL	undef	R/W	0 = counter will not be reloaded automatically after underflow 1 = counter will be reloaded automatically after underflow
0	EN	0	R/W	0 = disable timer

TimerN_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = enable timer

7.2.3.4 TimerPrescaleCounter

TimerPrescaleCounter				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	reserved
9-0	CounterValue	0	R/W	read: current counter value write: counter value is modified once

7.2.3.5 TimerPrescaleReload

TimerPrescaleReload				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	Reserved
9-0	ReloadValue	0	R/W	Prescaler division ratio = ReloadValue + 1 Note that this value must be set to at least 7 before enabling the timers

7.2.4 Primary Interrupt Controller Registers

7.2.4.1 PrimIntMaskAndPrio

PrimIntMaskAndPrio				
Bit	**Field**	**Default**	**R/W**	**Description**
31	reserved	1	R	reserved (NMI is always priority 1)
30	DSUTraceBufPrio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
29	Timer4Prio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
28	PIO3Prio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
27	GICHighPrio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
26	Timer3Prio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
25	PIO2Prio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
24	CICPrio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
23	Timer2Prio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
22	PIO1Prio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
21	GICLowPrio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
20	FFTDonePrio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
19	Timer1Prio	0	R/W	0 = Interrupt is low priority

PrimIntMaskAndPrio				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = Interrupt is high priority
18	PIO0Prio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
17	AHBErrorPrio	0	R/W	0 = Interrupt is low priority 1 = Interrupt is high priority
16	reserved	0	R	Reserved
15	reserved	0	R	reserved (NMI is not maskable)
14	DSUTraceBufMask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
13	Timer4Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
12	PIO3Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
11	GICHighMask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
10	Timer3Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
9	PIO2Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
8	CICMask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
7	Timer2Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
6	PIO1Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
5	GICLowMask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
4	FFTDoneMask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
3	Timer1Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
2	PIO0Mask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
1	AHBErrorMask	0	R/W	0 = Interrupt disabled 1 = Interrupt enabled
0	reserved	0	R	Reserved

7.2.4.2 PrimIntPending

PrimIntPending				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	Reserved
15	NMI	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
14	DSUTraceBuf	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
13	Timer4	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
12	PIO3	0	R	0 = Interrupt is not pending 1 = Interrupt is pending

PrimIntPending				
Bit	**Field**	**Default**	**R/W**	**Description**
11	GICHigh	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
10	Timer3	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
9	PIO2	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
8	CIC	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
7	Timer2	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
6	PIO1	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
5	GICLow	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
4	FFTDone	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
3	Timer1	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
2	PIO0	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
1	AHBError	0	R	0 = Interrupt is not pending 1 = Interrupt is pending
0	reserved	0	R	Reserved

7.2.4.3 PrimIntForce

PrimIntForce				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	Reserved
15	NMI	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
14	DSUTraceBuf	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
13	Timer4	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
12	PIO3	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
11	reserved	0	R	Reserved
10	Timer3	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
9	PIO2	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
8	reserved	0	R	Reserved
7	Timer2	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
6	PIO1	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
5	reserved	0	R	Reserved
4	FFTDone	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced

PrimIntForce				
Bit	**Field**	**Default**	**R/W**	**Description**
3	Timer1	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
2	PIO0	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
1	AHBError	0	R/W	0 = Interrupt is not forced 1 = Interrupt is forced
0	reserved	0	R	Reserved

7.2.4.4 PrimIntClear

PrimIntClear				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	reserved
15	NMI	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
14	DSUTraceBuf	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
13	Timer4	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
12	PIO3	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
11	reserved	0	R	reserved
10	Timer3	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
9	PIO2	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
8	reserved	0	R	reserved
7	Timer2	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
6	PIO1	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
5	reserved	0	R	Reserved
4	FFTDone	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
3	Timer1	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
2	PIO0	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
1	AHBError	0	W	0 = Interrupt is not cleared 1 = Interrupt is cleared
0	reserved	0	R	Reserved

7.2.5 PIO Registers

7.2.5.1 PIO_IO

PIO_IO				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	Reserved

PIO_IO				
Bit	**Field**	**Default**	**R/W**	**Description**
15	PIO15Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
14	PIO14Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
13	PIO13Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
12	PIO12Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
11	PIO11Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
10	PIO10Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
9	PIO9Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
8	PIO8Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
7	PIO7Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
6	PIO6Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
5	PIO5Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
4	PIO4Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
3	PIO3Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
2	PIO2Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
1	PIO1Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)
0	PIO0Data	undef	R/W	read: current level of IO (regardless of *PIODirection*) write: value driven on PIO output (if *PIODirection* is set to output)

7.2.5.2 PIODirection

PIODirection				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	Reserved
15	PIO15OutEn	0	R/W	0 = IO is input 1 = IO is output
14	PIO14OutEn	0	R/W	0 = IO is input 1 = IO is output
13	PIO13OutEn	0	R/W	0 = IO is input 1 = IO is output
12	PIO12OutEn	0	R/W	0 = IO is input 1 = IO is output
11	PIO11OutEn	0	R/W	0 = IO is input 1 = IO is output
10	PIO10OutEn	0	R/W	0 = IO is input 1 = IO is output

PIODirection				
Bit	**Field**	**Default**	**R/W**	**Description**
9	PIO9OutEn	0	R/W	0 = IO is input 1 = IO is output
8	PIO8OutEn	0	R/W	0 = IO is input 1 = IO is output
7	PIO7OutEn	0	R/W	0 = IO is input 1 = IO is output
6	PIO6OutEn	0	R/W	0 = IO is input 1 = IO is output
5	PIO5OutEn	0	R/W	0 = IO is input 1 = IO is output
4	PIO4OutEn	0	R/W	0 = IO is input 1 = IO is output
3	PIO3OutEn	0	R/W	0 = IO is input 1 = IO is output
2	PIO2OutEn	0	R/W	0 = IO is input 1 = IO is output
1	PIO1OutEn	0	R/W	0 = IO is input 1 = IO is output
0	PIO0OutEn	0	R/W	0 = IO is input 1 = IO is output

7.2.5.3 PIOIntConfig

PIOIntConfig				
Bit	**Field**	**Default**	**R/W**	**Description**
31	EN3	0	R/W	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be disabled
30	LE3	undef	R/W	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive
29	PL3	undef	R/W	Polarity. If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge)
28-24	ISEL3	undef	R/W	I/O port select. The value of this field defines which I/O port (0 - 15) should generate parallel I/O port interrupt n. Note that I/O 16-31 are not valid.
23	EN2	0	R/W	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be disabled
22	LE2	undef	R/W	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive
21	PL2	undef	R/W	Polarity. If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge)
20-16	ISEL2	undef	R/W	I/O port select. The value of this field defines which I/O port (0 - 15) should generate parallel I/O port interrupt n. Note that I/O 16-31 are not valid.
15	EN1	0	R/W	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be disabled
14	LE1	undef	R/W	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive

PIOIntConfig				
Bit	**Field**	**Default**	**R/W**	**Description**
13	PL1	undef	R/W	Polarity. If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge)
12-8	ISEL1	undef	R/W	I/O port select. The value of this field defines which I/O port (0 - 15) should generate parallel I/O port interrupt n. Note that I/O 16-31 are not valid.
7	EN0	0	R/W	Enable. If set, the corresponding interrupt will be enabled, otherwise it will be disabled
6	LE0	undef	R/W	Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive
5	PL0	undef	R/W	Polarity. If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge)
4-0	ISEL0	undef	R/W	I/O port select. The value of this field defines which I/O port (0 - 15) should generate parallel I/O port interrupt n. Note that I/O 16-31 are not valid.

7.2.6 Watchdog Registers

7.2.6.1 WdogPrescale

WdogPrescale				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	Reserved
15-0	PrescaleValue	65536	R/W	Prescaler = PrescaleValue * 4 + 2

7.2.6.2 WdogReload

WdogReload				
Bit	**Field**	**Default**	**R/W**	**Description**
31-24	reserved	0	R	Reserved
23-0	ReloadValue	0	R/W	Division Ratio = ReloadValue + 1 (a ReloadValue of 0 disables the watchdog)

7.2.6.3 WdogSel

WdogSel				
Bit	**Field**	**Default**	**R/W**	**Description**
31-1	reserved	0	R	Reserved
0	WdogSel	0	W	0 = AGGA reset is executed after 1st elapse of Wdog 1 = NMI interrupt is generated after 1st elapse of Wdog and AGGA reset is executed after 2nd elapse of Wdog Note: This bit is not readable. The application should cache the value in memory if needed.

7.2.6.4 WdogWriteEnable

WdogWriteEnable				
Bit	**Field**	**Default**	**R/W**	**Description**

WdogWriteEnable				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	W	write of "0xDEADAFFE" enables a one time write access to either the WdogPrescale or the WdogReload register

7.2.7 Debug link (UART and SPW) Registers

7.2.7.1 DSU_UART_Status

Note: The following register description is only valid if the DSU is operated via UART.

DSU_UART_Status (valid if DSU_SPW_EN pin = low)				
Bit	**Field**	**Default**	**R/W**	**Description**
31-7	reserved	0	R	Reserved
6	FE	0	R	Framing error (FE) -indicates that a framing error was detected
5	reserved	0	R	Reserved
4	OV	0	R	(OV) - indicates that one or more character have been lost due to overrun
3	reserved	0	R	Reserved
2	TH	1	R	Transmitter hold register empty (TH) - indicates that the transmitter hold register is empty
1	TS	1	R	Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty
0	DR	0	R	Data ready (DR) - indicates that new data is available in the receiver holding register

7.2.7.2 DSU_UART_SpW_Ctrl

DSU_UART_SpW_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-3	reserved	0	R	Reserved
2	LinkSpeed	0	R/W	0 = SpW link speed = 10MBits/s 1 = SpW link speed = *SysClk* Note: this bit is only valid if DSU SpW is enabled (pin *DSU_SPW_EN* = 1)
1	BL	0	R/W	Baud rate locked (BL) - is automatically set when the baud rate is locked Note: this bit is only valid if DSU UART is enabled (pin *DSU_SPW_EN* = 0)
0	RE	0	R/W	Receiver enable (RE) - if set, enables both the transmitter and receiver Note: this bit is only valid if DSU UART is enabled (pin *DSU_SPW_EN* = 0)

7.2.7.3 DSU_UART_Scaler

Note: The following register description is only valid if the DSU is operated via UART.

DSU_UART_Scaler (valid if DSU_SPW_EN pin = low)				
Bit	**Field**	**Default**	**R/W**	**Description**
31-14	reserved	0	R	Reserved
13-0	ReloadValue	8191	R/W	Baud Rate = *SysClk* / [8 * (ReloadValue + 1)] or ReloadValue = *SysClk* / (BaudRate * 8) - 1

7.2.8 Write Protect Registers

7.2.8.1 WriteProtectN

The register description is valid for the *WriteProtect1* and the *WriteProtect2* register (Address/Mask protection)

WriteProtectN				
Bit	**Field**	**Default**	**R/W**	**Description**
31	EN	0	R/W	0 = disable write protect unit 1 = enable write protect unit
30	BP	undef	R/W	0 = segement protect mode 1 = block protect mode
29-15	TAG	undef	R/W	Address Tag. This field is compared against address(29:15)
14-0	MASK	undef	R/W	Address Mask. This field contains the address mask

7.2.8.2 WriteProtectStartAddressN

The register description is valid for the following registers: *WriteProtectStartAddress3, WriteProtectStartAddress4*

WriteProtectStartAddressN				
Bit	**Field**	**Default**	**R/W**	**Description**
31-30	reserved	0	R	reserved
29-2	Start	undef	R/W	contains the first address in the protected block
1	BP	undef	R/W	Block Protect. If set, selects block protect mode
0	reserved	0	R	reserved

7.2.8.3 WriteProtectEndAddressN

The register description is valid for the following registers: *WriteProtectEndAddress3, WriteProtectEndAddress4*

WriteProtectEndAddressN				
Bit	**Field**	**Default**	**R/W**	**Description**
31-30	reserved	0	R	reserved
29-2	End	undef	R/W	contains the last address in the protected block
1	UM	0	R/W	User Mode. If set, write protection is enabled for user mode accesses
0	SU	0	R/W	Superuser Mode. If set, write protection is enabled for superuser mode accesses

7.2.9 GPIO Registers

7.2.9.1 GPIO_Status

GPIO_Status				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	reserved
15	GPIO_15	N/A	R	0 = GPIO[15] is level low 1 = GPIO[15] is level high
…				
1	GPIO_1	N/A	R	0 = GPIO[1] is level low 1 = GPIO[1] is level high
0	GPIO_0	N/A	R	0 = GPIO[0] is level low 1 = GPIO[0] is level high

7.2.9.2 GPIO_Output

GPIO_Output				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	reserved
15	GPIO_15	0	R/W	0 = GPIO[15] is assigned to level low 1 = GPIO[15] is assigned to level high
…				
1	GPIO_1	0	R/W	0 = GPIO[1] is assigned to level low 1 = GPIO[1] is assigned to level high
0	GPIO_0	0	R/W	0 = GPIO[0] is assigned to level low 1 = GPIO[0] is assigned to level high

7.2.9.3 GPIO_Direction

GPIO_Direction				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	reserved
15	GPIO_15	0	R/W	0 = GPIO[15] is an input 1 = GPIO[14] is an output
…				
1	GPIO_1	0	R/W	0 = GPIO[1] is an input 1 = GPIO[1] is an output
0	GPIO_0	0	R/W	0 = GPIO[0] is an input 1 = GPIO[0] is an output

7.2.10 Reset and Miscellaneous Registers

7.2.10.1 ResetStatus

ResetStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
31-9	DataContainer	all "1"	R/W	these bits can be used by the SW to store information over a reset (except power on reset). The bits are not changed by any reset, except power on reset. The bits are also not touched by any HW functionality.
8	MilBusTrigReset	1	R/W	If a reset occurs, the bit is set to "0" by the hardware.
7	GNSS_ResetHw	1	R/W	If a reset occurs, the bit is set to "0" by the hardware.
6	GNSS_ResetSw	1	R/W	If a reset occurs, the bit is set to "0" by the hardware.
5	AGGAResetSw	1	R/W	If a reset occurs, the bit is set to "0" by the hardware.
4	AGGAResetHw	1	R/W	If a reset occurs, the bit is set to "0" by the hardware.
3	WdogReset2	1	R/W	If a reset occurs, the bit is set to "0" by the hardware.
2	WdogReset1	1	R/W	If a reset occurs, the bit is set to "0" by the hardware.
1	NMI_WDOG	1	R/W	If the Watchdog triggered the NMI, the bit is set to "0" by the hardware.
0	NMI_External	1	R/W	When *NMI_INT* pin is raised for >= 1 *SysClk* cycle, the bit is set to "0" by the hardware. NMI (IRQ15) assertion requires BOTH edges (rise-fall or fall-rise) to occur with > 1 *SysClk* cycle distance.

7.2.10.2 SWResetEnable

SwResetEnable				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	W	write of 0xDEADAFFE arms the software reset any other value will inhibit a software reset arming is granted for one write of SwResetExecute

7.2.10.3 SWResetExecute

SwResetExecute				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	W	write of 0xBEBECAFE executes an AGGA reset write of 0xCAFEBEBE executes a GNSS reset register has to be armed first by SwResetEnable register

7.2.10.4 MilBusRTAddress

MilBusRTAddress				
Bit	**Field**	**Default**	**R/W**	**Description**
31-6	reserved	0	R	reserved
5	Parity	0	R/W	Parity for Mil Bus Remote Terminal Address
4-0	RTAddress	0	R/W	Mil Bus Remote Terminal Address

7.2.10.5 AGGA4Version

AGGA4Version				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	Version	1	R	“1” for the AGGA4 ASIC

7.2.11 Communication Interrupt Controller Registers

7.2.11.1 CIC_Mask

CIC_Mask				
Bit	**Field**	**Default**	**R/W**	**Description**
31-13	reserved	0	R	reserved
12	MilBusNominal (C53It)	0	R/W	0 = Interrupt for Mil Bus Nominal is disabled 1 = Interrupt for Mil Bus Nominal is enabled
11	MilBusReset (C53Rst)	0	R/W	0 = Interrupt for Mil Bus Reset is disabled 1 = Interrupt for Mil Bus Reset is enabled
10	MilBusError (C53Err)	0	R/W	0 = Interrupt for Mil Bus Error is disabled 1 = Interrupt for Mil Bus Error is enabled
9	SpaceWire	0	R/W	0 = Interrupt for SpaceWire is disabled 1 = Interrupt for SpaceWire is enabled
8	UART1_Error	0	R/W	0 = Interrupt for UART1 Error is disabled 1 = Interrupt for UART1 Error is enabled
7	UART1_TxDone	0	R/W	0 = Interrupt for UART1 Transmission Done is disabled 1 = Interrupt for UART1 Transmission Done is enabled
6	UART1_RxDone	0	R/W	0 = Interrupt for UART1 Receive Done is disabled 1 = Interrupt for UART1 Receive Done is enabled
5	UART0_Error	0	R/W	0 = Interrupt for UART0 Error is disabled 1 = Interrupt for UART0 Error is enabled
4	UART0_TxDone	0	R/W	0 = Interrupt for UART0 Transmission Done is disabled

CIC_Mask				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = Interrupt for UART0 Transmission Done is enabled
3	UART0_RxDone	0	R/W	0 = Interrupt for UART0 Receive Done is disabled 1 = Interrupt for UART0 Receive Done is enabled
2	SPI	0	R/W	0 = Interrupt for SPI is disabled 1 = Interrupt for SPI is enabled
1	CRCReady	0	R/W	0 = Interrupt for CRC Ready is disabled 1 = Interrupt for CRC Ready is enabled
0	SGPOOverrun	0	R/W	0 = Interrupt for Serial GPO Overrun is disabled 1 = Interrupt for Serial GPO Overrun is enabled

7.2.11.2 CIC_Pending

CIC_Pending				
Bit	**Field**	**Default**	**R/W**	**Description**
31-13	reserved	0	R	Reserved
12	MILBusNominal (C53It)	0	R	0 = Interrupt for Mil Bus Nominal is not pending 1 = Interrupt for Mil Bus Nominal is pending
11	MILBusReset (C53Rst)	0	R	0 = Interrupt for Mil Bus Reset is not pending 1 = Interrupt for Mil Bus Reset is enabled
10	MILBusError (C53Err)	0	R	0 = Interrupt for Mil Bus Error is pending 1 = Interrupt for Mil Bus Error is enabled
9	SpaceWire	0	R	0 = Interrupt for SpaceWire is not pending 1 = Interrupt for SpaceWire is pending
8	UART1_Error	0	R	0 = Interrupt for UART1 Error is not pending 1 = Interrupt for UART1 Error is pending
7	UART1_TxDone	0	R	0 = Interrupt for UART1 Transmission Done is not pending 1 = Interrupt for UART1 Transmission Done is pending
6	UART1_RxDone	0	R	0 = Interrupt for UART1 Receive Done is not pending 1 = Interrupt for UART1 Receive Done is pending
5	UART0_Error	0	R	0 = Interrupt for UART0 Error is not pending 1 = Interrupt for UART0 Error is pending
4	UART0_TxDone	0	R	0 = Interrupt for UART0 Transmission Done is not pending 1 = Interrupt for UART0 Transmission Done is pending
3	UART0_RxDone	0	R	0 = Interrupt for UART0 Receive Done is not pending 1 = Interrupt for UART0 Receive Done is pending
2	SPI	0	R	0 = Interrupt for SPI is not pending 1 = Interrupt for SPI is pending
1	CRCReady	0	R	0 = Interrupt for CRC Ready is not pending 1 = Interrupt for CRC Ready is pending
0	SGPOOverrun	0	R	0 = Interrupt for Serial GPO Overrun is not pending 1 = Interrupt for Serial GPO Overrun is pending

7.2.11.3 CIC_Clear

CIC_Clear				
Bit	**Field**	**Default**	**R/W**	**Description**
31-13	reserved	0	R	Reserved
12	reserved	0	R	reserved (MilBus Interrupt has to be cleared in MilBus Module)
11	reserved	0	R	reserved (MilBus Interrupt has to be cleared in MilBus Module)
10	reserved	0	R	reserved (MilBus Interrupt has to be cleared in MilBus Module)
9	reserved	0	R	reserved (SpW Interrupt has to be cleared in SpW Module)
8	UART1_Error	0	W	0 = Interrupt for UART1 Error is not cleared

CIC_Clear				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = Interrupt for UART1 Error is cleared
7	UART1_TxDone	0	W	0 = Interrupt for UART1 Transmission Done is not cleared 1 = Interrupt for UART1 Transmission Done is cleared
6	UART1_RxDone	0	W	0 = Interrupt for UART1 Receive Done is not cleared 1 = Interrupt for UART1 Receive Done is cleared
5	UART0_Error	0	W	0 = Interrupt for UART0 Error is not cleared 1 = Interrupt for UART0 Error is cleared
4	UART0_TxDone	0	W	0 = Interrupt for UART0 Transmission Done is not cleared 1 = Interrupt for UART0 Transmission Done is cleared
3	UART0_RxDone	0	W	0 = Interrupt for UART0 Receive Done is not cleared 1 = Interrupt for UART0 Receive Done is cleared
2	SPI	0	W	0 = Interrupt for SPI is not cleared 1 = Interrupt for SPI is cleared
1	CRCReady	0	W	0 = Interrupt for CRC Ready is not cleared 1 = Interrupt for CRC Ready is cleared
0	SGPOOverrun	0	W	0 = Interrupt for Serial GPO Overrun is not cleared 1 = Interrupt for Serial GPO Overrun is cleared

7.2.12 SPI Registers

7.2.12.1 SPI_StatusAndCtrl

SPI_StatusAndCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-20	reserved	0	R	Reserved
19-16	DataLength	0	R/W	Number of bits to be sent/received – 1 The value must be between 8bits and 16bits
15-11	SlaveSelect	0	R/W	Each bit enables the corresponding SELN line Bit 11 corresponds to SELN0 Bit 15 corresponds to SELN4 0 = corresponding SELN line is high (deactivated) 1 = corresponding SELN line is low (activated)
10	XmtDone	1	R	0 = transmission ongoing 1 = transmission completed
9	XmtEmpty	1	R	0 = content of SPI_Tx is not transfered to the Tx Shifter yet 1 = content of SPI_Tx is already loaded into the TxShifter and therefore the SPI_Tx is empty and ready for new value
8	RxAvailable	0	R	0 = SPI_Rx is not ready for read 1 = SPI_Rx has received data and is ready for read. This bit is cleared by reading SPI_Rx register.
7	Ss_n_afterTx	0	R/W	If set, the *SPI_SEL_N* signal is not deasserted between two consecutive transmissions to the same slave
6	reserved	0	R	Reserved
5	TxEn	0	R/W	0 = SPI transmit disable 1 = SPI transmit enable
4	MsbFirst	0	R/W	Set order of the data transmission/reception 0 = LSB first 1 = MSB first
3	ClkSel	1	R/W	0 = Interface generates *SPIClk* on pin *SPI_OUT_CLK* 1 = Interface receives *SPIClk* on pin *SPI_IN_CLK*
2	ClkPhase	0	R/W	Clock phase of *SPIClk* in relationsihp to serial data.

SPI_StatusAndCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
				0 = data valid on the 1st *SPIClk* after slave select asserted 1 = data valid on the 2nd *SPIClk* after slave select asserted
1	ClkPol	0	R/W	Master clock polarity when idle 0 = *SPIClk* is low when idle 1 = *SPIClk* is high when idle
0	RcvClkPol	0	R/W	Received clock polarity. 0 = *MISO* data sampled on the falling edge of *SPIClk* 1 = *MISO* data sampled on the rising edge of *SPIClk*

7.2.12.2 SPI_ClkDivider

SPI_ClkDivider				
Bit	**Field**	**Default**	**R/W**	**Description**
31-8	reserved	0	R	Reserved
7-0	DivRatio	0	R/W	DivRatio = *SysClk* / (8 * *SPIClk*) - 1

7.2.12.3 SPI_Tx

SPI_Tx				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	Reserved
15-0	Data	0	R/W	Data to be transmitted

7.2.12.4 SPI_Rx

SPI_Rx				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	Reserved
15-0	Data	0	R	Data received

7.2.13 CRC Unit Registers

7.2.13.1 CRCLFSR

CRCLFSR				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	before CRC start: write init value to this register after CRC completion: read final CRC value from this register Note that the written init value can not be read back. When reading the register it always reflects the CRC result. Assuming a polynomial of order n, Bit_0 corresponds to the initial values that matches x^{n-1} Bit_1 corresponds to the initial values that matches x^{n-2}, ... Unless n=32, Bits n to 31 shall be zero.

7.2.13.2 CRCPolynom

CRCPolynom				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	defines the feedback taps. A set bit means feedback tap active Assuming a polynomial of order n,

CRCPolynom				
Bit	**Field**	**Default**	**R/W**	**Description**
				Bit_0 corresponds to x^1, Bit_1 corresponds to x^2, ... Unless n=32, Bits n to 31 shall be zero.

7.2.13.3 CRCFinalXOR

CRCFinalXOR				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	at the end of the CRC processing, the result is XOR'd with CRCFinalXOR Assuming a polynomial of order n, Bit_0 is xORed with Bit 0 of CRCLFSR, Bit_1 is xORed with bit 1 of CRCLFSR, etc. Unless n=32, Bits n to 31 shall be 0

7.2.13.4 CRCCtrl

CRCCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	reserved	0	R	Reserved
1	ReverseResult	0	R/W	0 = CRC result is not reversed 1 = CRC result is reversed
0	ReverseInData	0	R/W	0 = bytes are read from MSB to LSB (left to right) 1 = bytes are read from LSB to MSB (right to left)

7.2.13.5 CRCStartAddress

CRCStartAddress				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	start address of data to be processed start address has to be 32bit aligned (bit 1:0 always zero)

7.2.13.6 CRCEndAddress

CRCEndAddress				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	end address of data to be processed end address can be byte aligned DMA fetching and CRC processor will start when writing to this register

7.2.13.7 CRCCurrentAddress

CRCCurAddress				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	current 8-bit address of data byte to be processed at the end of a CRC block: CRCCurAddress = CRCEndAddress

7.2.14 MILBUS Registers

7.2.14.1 C53CF

C53CF				
Bit	**Field**	**Default**	**R/W**	**Description**
31-28	reserved	0	R/W	Reserved
27	DW16En	0	R/W	0 = 16-bit Word Size disabled 1 = 16-bit Word Size enabled
26-25	reserved	0	R/W	Reserved
24-16	MemArea	0	R/W	Memory area MSB: code the base address (19 LSB of Haddr at 0) of the 64Kwords of data memory, followed by the 64Kwords of BC instruction or RT command memory
15	Err1553Mask	0	R/W	Err1553 Mask: must be at "0" to allow C53Err interrupt on 1553 error
14	ItSyncMask	0	R/W	ItSync Mask: must be at "0" to allow C53It interrupt on reception of synchronise order
13	ItTrokMask	0	R/W	ItTrok Mask: must be at "0" to allow C53It interrupt on end of a RT valid exchange
12-8	ExtSubAd	0	R/W	Sub-address number for Extended Memory Area mode
7	WdSize	0	R/W	0 = 32bit Word Size 1 = 16bit Word Size (if DW16En is set to 1)
6	ExtArea	0	R/W	0 = extended memory area mode inhibited 1 = extended memory area mode activated
5	reserved	0	R/W	Reserved
4	BrCstEn	0	R/W	0 = broadcast mode disable 1 = broadcast mode enable
3	DbcEn	0	R/W	0 = dynamic bus control disable 1 = dynamic bus control enable
2	TimeOut	0	R/W	0 = 14us time out selection 1 = 31us time out selection
1-0	Mode	0	R/W	0 = Remote Terminal (RT) Mode 1 = reserved 2 = reserved 3 = Remote Terminal (RT) Mode

7.2.14.2 C53EMBA

C53EMBA				
Bit	**Field**	**Default**	**R/W**	**Description**
31-26	reserved	0	R	reserved
25-0	ExtMemAd	0	R	Base Address of the extended memory area (written by 1553 link) (Word addressing)

7.2.14.3 C53CDST

C53CDST				
Bit	**Field**	**Default**	**R/W**	**Description**
31-15	LastCmdAd	0x10FFE	R	Address in the 1553 memory area of the last command treated. Before any command, the value is irrelevant so the reset value is set to a value out of the command area. (Word addressing)

14	ErrParAd	0	R	Set to “1” in case of RT address parity error
13-7	reserved	0	R	Reserved
6	BusyFlag	0	R	0 = stand by state 1 = active state
5	RstSt	0	R	0 = reset state or initialization state 1 = active state given by BusyFlag
4-3	reserved	0	R	Reserved
2	reserved	0	R/W	Reserved
1	GoStop	0	R/W	1553 Start Stop Command: set to “1” to activate the 1553 function
0	Rst1553	0	R/W	1553 function reset: set to “1” to reset the 1553 part

7.2.14.4 C53NIT

C53NIT				
Bit	**Field**	**Default**	**R/W**	**Description**
31-9	reserved	0	R	reserved
8-4	SwitchSubAd	0	R	Last Sub-address number which activated ItSwitch
3	ItSwitch	0	R	Not Maskable Interrupt: Set to “1” after change of data address table for a Sub-address; reset upon register read access
2	ItDbc	0	R	Not Maskable Interrupt: Dynamic bus control Command reception: set to “1” after Dynamic bus control Mode command reception in RT mode; reset upon register read access
1	ItSync	0	R	Maskable Interrupt: Set to “1” after reception of synchronise order, if SYNC bit set in the first instruction word; reset upon register read access
0	ItTrok	0	R	Maskable Interrupt: Set to “1” at the end of a RT valid exchange; reset upon register read access

7.2.14.5 C53EIT

C53EIT				
Bit	**Field**	**Default**	**R/W**	**Description**
31-5	reserved	0	R	reserved
4	ErrRTAd	0	R	Not Maskable Interrupt : Set to 1 in case of RT address parity error; reset upon register read access
3	reserved	0	R/W	reserved
2	reserved	0	R	reserved
1	Err1553	0	R	Maskable Interrupt: Set to “1” in case of 1553 error; reset upon register read access
0	ErrMem	0	R	Not Maskable Interrupt: Set to “1” in case of DRAM access error; reset upon register read access

7.2.14.6 C53RIT

C53RIT				
Bit	**Field**	**Default**	**R/W**	**Description**
31-1	reserved	0	R	reserved

C53RIT				
Bit	**Field**	**Default**	**R/W**	**Description**
0	RstCom	0	R	Not maskable Interrupt: Set to "1" after reception of a valid "Reset Remote Terminal"; reset upon register read access

7.2.14.7 C53RTI

C53RTI				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	RTIndex	0	R	Reception Table Index Value

7.2.14.8 C53TTI

C53TTI				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	TTIndex	0	R	Transmission Table Index Value

7.2.15 Serial General Purpose Output (SGPO) Registers

7.2.15.1 SGPO_Tx

SGPO_Tx				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	write: written data is fed into the Tx FIFO read: written data is read back

7.2.15.2 SGPO_Status

SGPO_Status				
Bit	**Field**	**Default**	**R/W**	**Description**
31-6	reserved	0	R	reserved
5-0	FIFOCnt	0	R	Transmitter FIFO Count (0=empty, 16=full)

7.2.15.3 SGPO_Ctrl

SGPO_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-5	reserved	0	R	Reserved
4	TXEn	0	R/W	0 = transmitter disabled 1 = transmitter enabled
3	PS	0	R/W	0 = even parity 1 = odd parity
2	PE		R/W	0 = parity is disabled 1 = parity is enabled
1-0	ByteSel	0	R/W	0 = 1byte (Bit 7..0) of S-GPO-TX is transfered to the FIFO 1 = 2bytes (Bit 15..0) of S-GPO-TX is transfered to the FIFO 2 = 3bytes (Bit 23..0) of S-GPO-TX is transfered to the FIFO 3 = 4bytes (Bit 31..0) of S-GPO-TX is transfered to the FIFO

7.2.15.4 SGPO_Scaler

SGPO_Scaler				
Bit	**Field**	**Default**	**R/W**	**Description**
31-12	reserved	0	R	reserved
11-0	Scaler	0	R/W	Scaler = *SysClk* / (BaudRate * 8) - 1

7.2.16 UART Registers

7.2.16.1 UARTn_Tx_SAP

The register description is valid for the following registers: *UART0_Tx_SAP* and *UART1_Tx_SAP*.

UARTn_Tx_SAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	start address for data transmission, 32-bit aligned Bit 1:0 ignored at write, "00" at read

7.2.16.2 UARTn_Tx_EAP

The register description is valid for the following registers: *UART0_Tx_EAP* and *UART1_Tx_EAP*.

UARTn_Tx_EAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	end address for data transmission end address is a byte address (8bit aligned) After write access to this register the UART starts to read data from external memory

7.2.16.3 UARTn_Tx_CAP

The register description is valid for the following registers: *UART0_Tx_CAP* and *UART1_Tx_CAP*.

UARTn_Tx_CAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	current address for data transmission (bit 1:0 = always "00") Points to the first 32bit-word not yet read from memory

7.2.16.4 UARTn_Rx_SAP

The register description is valid for the following registers: *UART0_Rx_SAP* and *UART1_Rx_SAP*.

UARTn_Rx_SAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	start address for data reception, 32-bit aligned (bit 1:0 ignored) writing to this register stops an ongoing DMA transfer

7.2.16.5 UARTn_Rx_EAP

The register description is valid for the following registers: *UART0_Rx_EAP* and *UART1_Rx_EAP*.

UARTn_Rx_EAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	end address for data reception end address is a byte address (8bit aligned)

UARTn_Rx_EAP				
Bit	**Field**	**Default**	**R/W**	**Description**
				Note: If the EAP is not 32bit aligned, the user has to make sure that the residual 1..3 bytes after the EAP are not used by the software, since the DMA writes zeros on these bytes.

7.2.16.6 UARTn_Rx_CAP

The register description is valid for the following registers: *UART0_Rx_CAP* and *UART1_Rx_CAP*.

UARTn_Rx_CAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	current address for data reception shows next free byte address for data reception At the end of a transfer: CAP - SAP = nr. of received bytes

7.2.16.7 UARTn_Status

The register description is valid for the following registers: *UART0_Status* and *UART1_Status*.

UARTn_Status				
Bit	**Field**	**Default**	**R/W**	**Description**
31-8	reserved	0	R	reserved
7	RTO	0	R	Receiver time out (when TOE = '1') RTO is cleared when RxBuffer is read or TOE=0 or RXEN=0
6	FER	0	R/W	Framing Error (write '0' to clear)
5	PER	0	R/W	Parity Error (write '0' to clear)
4	OVR	0	R/W	Received overrrun (write '0' to clear)
3	BR	0	R/W	Break Received (write '0' to clear)
2	TE	1	R	Transmitter buffer empty
1	TSE	1	R	Transmitter shift register empty
0	DR	0	R	Data Ready. This bit is set when data is in the RxBuffer

7.2.16.8 UARTn_Ctrl

The register description is valid for the following registers: *UART0_Ctrl* and *UART1_Ctrl*.

UARTn_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	reserved
9	LEND	0	R/W	0 = memory data in big endian order 1 = memory data in little endian order
8	TOE	0	R/W	0 = Timeout Function Disable 1 = Timeout Function Enable
7	LB	0	R/W	0 = internal loopback disabled 1 = internal loopback enabled
6	FL	0	R/W	0 = data flow controlled only with TX and RX signals 1 = data flow using CTS/RTS signals
5	PE	0	R/W	0 = parity is disabled 1 = parity is enabled
4	PS	0	R/W	0 = even parity 1 = odd parity
3	reserved	0	R	reserved

UARTn_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
2	reserved	0	R	reserved
1	TXEn	0	R/W	0 = Tx Disable 1 = Tx Enable
0	RXEn	0	R/W	0 = Rx Disable 1 = Rx Enable

7.2.16.9 UARTn_Scaler

The register description is valid for the following registers: *UART0_Scaler* and *UART1_Scaler*.

UARTn_Scaler				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	0	R	Reserved
28-16	TimeoutCnt	8191	R/W	Timeout Counter Reload Value. Counts in units of BaudTicks. TiemoutCnt = DesiredBaudTicks – 1
15-12	Reserved	0	R	Reserved
11-0	ReloadValue	0	R/W	Baud Rate = *SysClk* / [8 * (ReloadValue + 1)] or ReloadValue = *SysClk* / (BaudRate * 8) - 1

7.2.16.10 UART_Reset

UART_Reset				
Bit	**Field**	**Default**	**R/W**	**Description**
31-1	reserved	0	R	Reserved
0	Reset	0	W	If set, the UART Module (UART0 and UART1) is reset.

7.2.17 Clock, PLL and EEPROM Control Registers

7.2.17.1 GNSSCoreClkCtrl

GNSSCoreClkCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-1	reserved	0	R	Reserved
0	CoreClkSel	0	R/W	0 = *CoreClk* (internal) is taken from input pin *EXT_CORE_CLK* 1 = *CoreClk* (internal) is taken from pin *HALF_SAMPLE_CLK* / 2.5

7.2.17.2 MilBusClkCtrl

MilBusClkCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-1	reserved	0	R	Reserved
0	MilBusClkSel	0	R/W	0 = *1553Clk* is taken from input pin *EXT_MIL_CLK* 1 = *1553Clk* is taken from MilBus PLL output

7.2.17.3 PLLStatus

PLLStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	reserved	0	R	Reserved
5-2	ClkDiv	*	R	Value set in *SYS_CLK_DIV* input pins

PLLStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
1	MilBusPLLStatus	0	R	0 = not locked 1 = locked
0	LeonPLLStatus	0	R	0 = not locked 1 = locked

7.2.17.4 EEPROM_StatusAndCtrl

EEPROM_StatusAndCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-4	EEPROM_Switch	0	W	0xAFFEDEAx to switch EEPROM on (x = 0 .. F) 0xDEADAFFx to switch EEPROM off (x = 0 .. F)
3	EEPROM_Enable	0	R	the status of the EEPROM_Enable signal
2	EEPROM_Reset	0	R	the status of the EEPROM_Reset signal
1	EEPROM_Power	0	R	the status of the EEPROM_Power signal
0	EEPROM_On	0	R	the status of the EEPROM_On signal

7.2.18 Spacewire Registers

7.2.18.1 SpWn_StatusAndCtrl

SpWn_StatusAndCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-13	reserved	0	R	Reserved
12	FCT_Err	0	R	FCT Error status (reset after read)
11	ESC_Err	0	R	ESC Error status (reset after read)
10	Parity_Err	0	R	Parity Error status (reset after read)
9	Disconnect_Err	0	R	Disconnect Error status (reset after read)
8	LinkOk	0	R	Link OK status. (this bit is reset if link error occurs or if bits 1-0 are set to "00")
7-5	reserved	0	R	
4-3	TxBitRate	0	R/W	Transmit bit rate selection (if bit 2 is set) 00: max. transmit bit rate, given by *SysClk* 01: max. transmit bit rate x 1/2 10: max. transmit bit rate x 1/4 11: max. transmit bit rate x 1/8
2	DefaultTxRate	0	R/W	0 = default transmit bit rate (10 Mbit/s, given by *SpW10MHzClk*) 1= transmit bit rate selected by "TxBitRate"
1	AutoStart	1	R/W	0 = no Auto Start 1 = Auto start link after a NULL token is received
0	StartStopLink	0	R/W	0 = Stop Link 1 = Start Link

7.2.18.2 SpWn_Tx_SAP

SpWn_Tx_SAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	start address for data transmission, 32-bit aligned Bit 1:0 ignored at write, "00" at read

7.2.18.3 SpW_Tx_EAP

SpWn_Tx_EAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	end address for data transmission After write access to this register SPWn starts to read data from external memory

7.2.18.4 SpWn_Tx_CAP

SpWn_Tx_CAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	current address for data transmission (in 32-bit steps) Points to the first 32bit word not yet read from memory

7.2.18.5 SpWn_Tx_Rx_Config

SpWn_Tx_Rx_Config				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	reserved
9	TxLengthError	0	R	Asserted when HeaderField mode is enabled (see below) and the packet is shorter than the specified header length. (read-only, but cleared upon reading)
8	EEP_Received	0	R	It is set if a packet was terminated with EEP (read-only, but cleared upon reading)
7-5	reserved	0	R	
4	LittleEndianEnable	0	R/W	0 = big endian (byte 0 transported on D31-D24) 1 = little endian (byte 0 transported on D7-D0)
3	reserved	0	R	
2	NoStopOnEOP	0	R/W	if set, RX operation will not stop when an EOP or EEP is received, no EOP/EEP interrupt is generated, successive packets are concatenated.
1	HeaderField	0	R/W	Header Field Control bit: If set, SpW TX interface uses the first byte of a packet as number of bytes which are transmitted as header bytes. The range for the header field is from minimum 2 bytes (first byte + header itself) to a maximum 16 bytes. The size of header field can be 4, 8, 12 or 16 bytes. This means, that the data field starts at the next modulo 4 bytes. The rest of a 4-byte block which is not covered by the number of header bytes will not be transmitted.
0	reserved	0	R	

7.2.18.6 SpWn_Rx_SAP

SpWn_Rx_SAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	start address for data reception, 32-bit aligned (bit 1:0 ignored) Writing to this register stops an ongoing DMA transfer

7.2.18.7 SpWn_Rx_EAP

SpWn_Rx_EAP

Bit	Field	Default	R/W	Description
31-0		3	R/W	end address for data reception bit 1:0 are always "11"

7.2.18.8 SpWn_Rx_CAP

SpWn_Rx_CAP				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	During transfer: current 32-bit address for data reception (bit 1:0 always 0) At the end of a transfer: CAP – SAP = nr. of received bytes Bit 1:0 ≠ 00 indicates a non 32-bit aligned packet was received

7.2.18.9 SpW_ModuleConfig

SpW_ModuleConfig				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	reserved	0	R	reserved
1	Reset	0	W	If set, the Spw Module is reset.
0	reserved	0	R	

7.2.18.10 SpW_ModuleTimeCtrl

SpW_ModuleTimeCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	reserved	0	R	reserved
1	IRQ_En_All	0	R/W	Enable TICK_IN interrupt for all time codes
0	IRQ_En_Valid	0	R/W	Enable TICK_IN interrupt for valid time codes

7.2.18.11 SpW_ModuleTimeCode

SpW_ModuleTimeCode				
Bit	**Field**	**Default**	**R/W**	**Description**
31-8	reserved	0	R	reserved
7-6	Time CodeFlags	0	R/W	time code flags (must be set to "00")
5-0	Time CodeValue	0	R/W	

7.2.18.12 SpW_ModuleIntMask

SpW_ModuleIntMask				
Bit	**Field**	**Default**	**R/W**	**Description**
31	SPW_TickIn	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
30-29	reserved	0	R	reserved
28	SPW3_RxAreaFull	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
27	SPW3_EOP_EEP	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
26	SPW3_TxDone	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
25	SPW3_LinkConnected	0	R/W	0 = Interrupt is masked

SpW_ModuleIntMask				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = Interrupt is enabled
24	SPW3_LinkError	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
23-21	reserved	0	R	reserved
20	SPW2_RxAreaFull	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
19	SPW2_EOP_EEP	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
18	SPW2_TxDone	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
17	SPW2_LinkConnected	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
16	SPW2_LinkError	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
15-13	reserved	0	R	reserved
12	SPW1_RxAreaFull	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
11	SPW1_EOP_EEP	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
10	SPW1_TxDone	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
9	SPW1_LinkConnected	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
8	SPW1_LinkError	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
7-5	reserved	0	R	reserved
4	SPW0_RxAreaFull	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
3	SPW0_EOP_EEP	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
2	SPW0_TxDone	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
1	SPW0_LinkConnected	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled
0	SPW0_LinkError	0	R/W	0 = Interrupt is masked 1 = Interrupt is enabled

7.2.18.13 SpW_ModuleIntStatus

SpW_ModuleIntStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
31	SPW_TickIn	0	R	SpW Module TICK_IN
30-29	reserved	0	R	
28	SPW3_RxAreaFull	0	R	SPW3 Receive area full
27	SPW3_EOP_EEP	0	R	SPW3 EOP/EEP Received
26	SPW3_TxDone	0	R	SPW3 packet is transmitted
25	SPW3_LinkConnected	0	R	SPW3 Link is running
24	SPW3_LinkError	0	R	SPW3 Link Error
23-21	reserved	0	R	
20	SPW2_RxAreaFull	0	R	SPW2 Receive area full

SpW_ModuleIntStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
19	SPW2_EOP_EEP	0	R	SPW2 EOP/EEP Received
18	SPW2_TxDone	0	R	SPW2 packet is transmitted
17	SPW2_LinkConnected	0	R	SPW2 Link is running
16	SPW2_LinkError	0	R	SPW2 Link Error
15-13	reserved	0	R	
12	SPW1_RxAreaFull	0	R	SPW1 Receive area full
11	SPW1_EOP_EEP	0	R	SPW1 EOP/EEP Received
10	SPW1_TxDone	0	R	SPW1 packet is transmitted
9	SPW1_LinkConnected	0	R	SPW1 Link is running
8	SPW1_LinkError	0	R	SPW1 Link Error
7-5	reserved	0	R	
4	SPW0_RxAreaFull	0	R	SPW0 Receive area full
3	SPW0_EOP_EEP	0	R	SPW0 EOP/EEP Received
2	SPW0_TxDone	0	R	SPW0 packet is transmitted
1	SPW0_LinkConnected	0	R	SPW0 Link is running
0	SPW0_LinkError	0	R	SPW0 Link Error

7.2.18.14 SpW_ModuleIntClear

SpW_ModuleIntClear				
Bit	**Field**	**Default**	**R/W**	**Description**
31	SPW_TickIn	0	W	0 = do nothing 1 = Interrupt is cleared
30-29	reserved	0	R	reserved
28	SPW3_RxAreaFull	0	W	0 = do nothing 1 = Interrupt is cleared
27	SPW3_EOP_EEP	0	W	0 = do nothing 1 = Interrupt is cleared
26	SPW3_TxDone	0	W	0 = do nothing 1 = Interrupt is cleared
25	SPW3_LinkConnected	0	W	0 = do nothing 1 = Interrupt is cleared
24	SPW3_LinkError	0	W	0 = do nothing 1 = Interrupt is cleared
23-21	reserved	0	R	Reserved
20	SPW2_RxAreaFull	0	W	0 = do nothing 1 = Interrupt is cleared
19	SPW2_EOP_EEP	0	W	0 = do nothing 1 = Interrupt is cleared
18	SPW2_TxDone	0	W	0 = do nothing 1 = Interrupt is cleared
17	SPW2_LinkConnected	0	W	0 = do nothing 1 = Interrupt is cleared
16	SPW2_LinkError	0	W	0 = do nothing 1 = Interrupt is cleared
15-13	reserved	0	R	Reserved
12	SPW1_RxAreaFull	0	W	0 = do nothing 1 = Interrupt is cleared
11	SPW1_EOP_EEP	0	W	0 = do nothing 1 = Interrupt is cleared

SpW_ModuleIntClear				
Bit	**Field**	**Default**	**R/W**	**Description**
10	SPW1_TxDone	0	W	0 = do nothing 1 = Interrupt is cleared
9	SPW1_LinkConnected	0	W	0 = do nothing 1 = Interrupt is cleared
8	SPW1_LinkError	0	W	0 = do nothing 1 = Interrupt is cleared
7-5	reserved	0	R	Reserved
4	SPW0_RxAreaFull	0	W	0 = do nothing 1 = Interrupt is cleared
3	SPW0_EOP_EEP	0	W	0 = do nothing 1 = Interrupt is cleared
2	SPW0_TxDone	0	W	0 = do nothing 1 = Interrupt is cleared
1	SPW0_LinkConnected	0	W	0 = do nothing 1 = Interrupt is cleared
0	SPW0_LinkError	0	W	0 = do nothing 1 = Interrupt is cleared

7.3 DSU Address Map

7.3.1 DSU Overview

Address	Register
0x90000000	DSU_Ctrl
0x90000004	TraceBufferCtrl
0x90000008	TimeTagCounter
0x90000010	AHB_BreakAddress1
0x90000014	AHB_Mask1
0x90000018	AHB_BreakAddress2
0x9000001C	AHB_Mask2
0x90010000 - 0x9001FFFC	Trace buffer
…0	Trace bits 127 - 96
…4	Trace bits 95 - 64
…8	Trace bits 63 - 32
…C	Trace bits 31 - 0
0x90020000 - 0x9003FFFC	IU/FPU register file
0x90080000 - 0x900FFFFC	IU special purpose registers
0x90080000	Y register
0x90080004	PSR register
0x90080008	WIM register
0x9008000C	TBR register
0x90080010	PC register
0x90080014	NPC register
0x90080018	FSR register
0x9008001C	DSU_Trap
0x90080040 - 0x9008005C	ASR16 - 23 (not implemented)
0x90080060 - 0x9008007C	ASR24 - 31 (watchpoints see 4.5)
0x90100000 - 0x9013FFFC	Instruction cache tags
0x90140000 - 0x9017FFFC	Instruction cache data

0x90180000 - 0x901BFFFC	Data cache tags
0x901C0000 - 0x901FFFFC	Data cache data

7.3.1.1 DSU_Ctrl

DSU_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	0	R	reserved
28-20	DCNT	undef	R/W	Trace buffer delay counter (DCNT)
19	RE	0	W	Reset error mode (RE) - if set, it will clear the error mode in the processor
18	DR	0	R/W	Debug mode response (DR) - if set, the DSU communication link will send a response word when the processor enters debug mode
17	LR	0	R/W	Link response (LR) - if set, the DSU communication link will send a response word after AHB transfer
16	SS	0	R/W	Single step (SS) - if set, the processor will execute one instruction and then return to debug mode
15	PE	0	R	Processor error mode (PE) - returns '1' on read when processor is in error mode, else '0'
14	EE	DSUEN	R	value of the external *DSU_EN* pin
13	EB	DSUBRE	R	value of the external *DSU_BRE* pin
12	DM	0	R	0 = processor is not in debug mode 1 = processor is in debug mode
11	DE	0	R/W	Delay counter enable (DE) - if set, the trace buffer delay counter will decrement for each stored trace. This bit is set automatically when a DSU breakpoint is hit and the delay counter is not equal to zero
10	BZ	DSUBRE	R/W	Break on error traps (BZ) - if set, it will force the processor into debug mode on all except the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap
9	BX	DSUBRE	R/W	Break on trap (BX) - if set, it will force the processor into debug mode when any trap occurs
8	BD	0	R/W	Break on DSU breakpoint (BD) - if set, it will force the processor to debug mode when a DSU breakpoint is hit
7	BN	DSUBRE	R/W	Break now (BN) -If set, it will force processor into debug mode. If cleared, the processor will resume execution
6	BS	0	R/W	Break on S/W breakpoint (BS) - if set, debug mode will be forced when a breakpoint instruction (ta 1) is executed
5	BW	DSUBRE	R/W	Break on IU watch point - if set, debug mode will be forced on an IU watch point (trap 0xb)
4	BE	DSUBRE	R/W	Break on error (BE) - if set, it will force the processor to debug mode when the processor would have entered error condition (trap in trap)
3	FT	0	R/W	Freeze timers (FT) - if set, the scaler in the LEON timer unit will be stopped during debug mode to preserve the time for the software application
2	BT	0	R/W	Break on trace (BT) - if set, it will generate a DSU break condition on trace freeze

DSU_Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
1	DM	undef	R/W	Delay counter mode (DM). In mixed tracing mode, setting this bit will cause the delay counter to decrement on AHB traces. If reset, the delay counter will decrement on instruction traces
0	TE	0	R/W	0 = Trace Buffer disabled 1 = Trace Buffer enabled

7.3.1.2 TraceBufferCtrl

TraceBufferCtrl (!!Note that this register is not cleared during reset!!)				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	MFILT	undef	R/W	trace only accesses from AHB masters with a particular master index 0 = trace accesses from all masters 1 = trace only CRC module 2 = trace only SpW module 3 = trace only UART-DMA 4 = trace only GNSS DMA 5 = trace only Mil-Bus module 6 = trace only the DSU-UART 7 = trace only the CPU
28-27	SFILT	undef	R/W	trace only addresses with a certain prefix (bits 31:28) 0 = trace all slave addresses 1 = trace only APB (0x8) registers 2 = trace only GNNS (0xA) registers 3 = trace only FFT (0xB) registers
26	AF	undef	R/W	0 = AHB trace buffer will not be frozen if processor enters debug mode 1 = AHB trace buffer will be frozen if processor enters debug mode
25	TA	undef	R/W	0 = Trace AHB (TA) data transfer disabled 1 = Trace AHB (TA) data transfer enabled
24	TI	undef	R/W	0 = Trace Instruction (TI) disabled 1 = Trace Instruction (TI) enabled
23-21	reserved	undef	R	reserved
20-12	AHBIndex	undef	R	AHB trace index counter
11-9	reserved	undef	R	reserved
8-0	InstIndex	undef	R	Instruction trace index counter

7.3.1.3 TimeTagCounter

TimeTagCounter				
Bit	**Field**	**Default**	**R/W**	**Description**
31-30	reserved	0	R	reserved
29-0	TimeTagValue	n/a	R/W	The DSU time tag counter is incremented each *SysClk* cycle as long as the processor is running. The counter is stopped when the processor enters debug mode, and restarted when execution is resumed.

7.3.1.4 AHB_BreakAddressN

The register description is valid for the following registers: *AHB_BreakAddress1, AHB_BreakAddress2*

AHB_BreakAddressN				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	BADDR	0	R/W	break address
1	reserved	0	R	reserved
0	EX	0	R/W	0 = no break on executed instructions 1 = break on executed instuctions

7.3.1.5 AHB_MaskN

The register description is valid for the following registers: *AHB_Mask1, AHB_Mask2*

AHB_MaskN				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	BMASK	0	R/W	mask address for break
1	LD	0	R/W	0 = no break on load accesses 1 = break on load accesses
0	ST	0	R/W	0 = no break on store accesses 1 = break on store accesses

7.3.1.6 DSU_Trap

DSU_Trap				
Bit	**Field**	**Default**	**R/W**	**Description**
31-13	reserved	0	R	reserved
12	EM	0	R	Error mode (EM). Set if the trap would have cause the processor to enter error mode
11-4	TrapType	0	R	8bit SPARC Trap Type
3-0	reserved	0	R	reserved

7.4 GNSS Address Map

7.4.1 GNSS Base Addresses

Since the GNSS module is split into several channels and input modules, the address map is split into base and offset addresses. The AHB base addresses for the respective channels, input modules, RAM blocks and other functions are given in the following table. The address offsets provided in the other subsections of this chapter must be added to the base addresses.

Address	Mapping
0xA0000000	Channel 0
0xA0010000	Channel 1
0xA0020000	Channel 2
0xA0030000	Channel 3
0xA0040000	Channel 4
0xA0050000	Channel 5
0xA0060000	Channel 6
0xA0070000	Channel 7
0xA0080000	Channel 8

Address	Mapping
0xA1000000	RAM Channel 0
0xA1010000	RAM Channel 1
0xA1020000	RAM Channel 2
0xA1030000	RAM Channel 3
0xA1040000	RAM Channel 4
0xA1050000	RAM Channel 5
0xA1060000	RAM Channel 6
0xA1070000	RAM Channel 7
0xA1080000	RAM Channel 8

Address	Mapping
0xA0090000	Channel 9
0xA00A0000	Channel 10
0xA00B0000	Channel 11
0xA00C0000	Channel 12
0xA00D0000	Channel 13
0xA00E0000	Channel 14
0xA00F0000	Channel 15
0xA0100000	Channel 16
0xA0110000	Channel 17
0xA0120000	Channel 18
0xA0130000	Channel 19
0xA0140000	Channel 20
0xA0150000	Channel 21
0xA0160000	Channel 22
0xA0170000	Channel 23
0xA0180000	Channel 24
0xA0190000	Channel 25
0xA01A0000	Channel 26
0xA01B0000	Channel 27
0xA01C0000	Channel 28
0xA01D0000	Channel 29
0xA01E0000	Channel 30
0xA01F0000	Channel 31
0xA0200000	Channel 32
0xA0210000	Channel 33
0xA0220000	Channel 34
0xA0230000	Channel 35
0xA0240000	ChannelMatrix
0xA0250000	Input Module 0
0xA0260000	Input Module 1
0xA0270000	Input Module 2
0xA0280000	Input Module 3
0xA0290000	reserved
0xA02A0000	reserved
0xA02B0000	Power Level Detector
0xA02C0000	Interrupt Registers

Address	Mapping
0xA1090000	RAM Channel 9
0xA10A0000	RAM Channel 10
0xA10B0000	RAM Channel 11
0xA10C0000	RAM Channel 12
0xA10D0000	RAM Channel 13
0xA10E0000	RAM Channel 14
0xA10F0000	RAM Channel 15
0xA1100000	RAM Channel 16
0xA1110000	RAM Channel 17
0xA1120000	RAM Channel 18
0xA1130000	RAM Channel 19
0xA1140000	RAM Channel 20
0xA1150000	RAM Channel 21
0xA1160000	RAM Channel 22
0xA1170000	RAM Channel 23
0xA1180000	RAM Channel 24
0xA1190000	RAM Channel 25
0xA11A0000	RAM Channel 26
0xA11B0000	RAM Channel 27
0xA11C0000	RAM Channel 28
0xA11D0000	RAM Channel 29
0xA11E0000	RAM Channel 30
0xA11F0000	RAM Channel 31
0xA1200000	RAM Channel 32
0xA1210000	RAM Channel 33
0xA1220000	RAM Channel 34
0xA1230000	RAM Channel 35
Other addr: 0xA1xxxxxx	Reserved (access will lead to system lockup)

7.4.2 Input Module Registers

Address Offset	Size [bits]	Name
+ 0x00	4	InputModuleCtrl
+ 0x04	7	DDCMainPhaseInc
+ 0x08	30	DDCMainFIRQuantThres
+ 0x0C	7	DDCAuxPhaseInc
+ 0x10	30	DDCAuxFIRQuantThres
+ 0x14	12	DACtrl

7.4.2.1 InputModuleCtrl

InputModuleCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-6	reserved	0	R	Reserved
5	DDCAuxInSel	0	R/W	0 = data from local Input Module is taken 1 = data from previous Input Module is taken
4	DDCMainInSel	0	R/W	0 = data from local Input Module is taken 1 = data from previous Input Module is taken
3-2	IFCFormat	0	R/W	0 = Sign/Magnitude 1 = Unsigned 2 = Two's Complement 3 = reserved
1-0	Mode	0	R/W	0 = Input Format Converter Mode (IFC) 1 = Real to complex Converter Mode (R2C) 2 = Digital Down Converter Simple Mode (DDC simple) 3 = reserved

7.4.2.2 DDCMainPhaseInc

DDCMainPhaseInc				
Bit	**Field**	**Default**	**R/W**	**Description**
31-7	reserved	0	R	Reserved
6-0	Inc	0	R/W	Signed Increment for DDC I/Q Mixer NCO. New settings become effective with the next ME. The phase of the NCO is reset to zero with the next ME in order to be able to synchronize different DDC NCO's.

7.4.2.3 DDCMainFIRQuantThres

DDCMainFIRQuantThres				
Bit	**Field**	**Default**	**R/W**	**Description**
31-30	reserved	0	R	Reserved
29-20	Thres2	0	R/W	Output = +7 for Input > +Thres2 Output = -7 for Input < -Thres2 Output = +5 for "+Thres1 < Input <= +Thres2" Output = -5 for "-Thres2 <= Value < -Thres1"
19-10	Thres1	0	R/W	Output = +3 for "+Thres0 < Input <= +Thres1" Output = -3 for "-Thres1 <= Value < -Thres0"
9-0	Thres0	0	R/W	Output = +1 for "0 <= Value <= +Thres0" Output = -1 for "-Thres0 <= Input < 0"

7.4.2.4 DDCAuxPhaseInc

DDCAuxPhaseInc				
Bit	**Field**	**Default**	**R/W**	**Description**
31-7	reserved	0	R	Reserved
6-0	Inc	0	R/W	Signed Increment for DDC I/Q Mixer NCO. New settings become effective with the next ME. The phase of the NCO is reset to zero with the next ME in order to be able to synchronize different DDC NCO's.

7.4.2.5 DDCAuxFIRQuantThres

DDCAuxFIRQuantThres				
Bit	**Field**	**Default**	**R/W**	**Description**
31-30	reserved	0	R	Reserved
29-20	Thres2	0	R/W	Output = +7 for Input > +Thres2 Output = -7 for Input < -Thres2 Output = +5 for "+Thres1 < Input <= +Thres2" Output = -5 for "-Thres2 <= Value < -Thres1"
19-10	Thres1	0	R/W	Output = +3 for "+Thres0 < Input <= +Thres1" Output = -3 for "-Thres1 <= Value < -Thres0"
9-0	Thres0	0	R/W	Output = +1 for "0 <= Value <= +Thres0" Output = -1 for "-Thres0 <= Input < 0"

7.4.2.6 DACtrl

DACtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-12	reserved	0	R	Reserved
11-0	Value	0	R/W	Control Word for DA Converter

7.4.3 Power Level Detector Registers

Address Offset	Size [bits]	Name
+ 0x00	2	PLD5IInputSel
+ 0x04	25	PLD5ICtrl
+ 0x08	8	PLDIQInputSel
+ 0x0C	14	PLDIQPreAccCtrl
+ 0x10	25	PLDIQCtrl
+ 0x14	24	Acc5IPlusSeven
+ 0x18	24	Acc5IPlusFive
+ 0x1C	24	Acc5IPlusThree
+ 0x20	24	Acc5IPlusOne
+ 0x24	24	Acc5IMinusOne
+ 0x28	24	Acc5IMinusThree
+ 0x2C	24	Acc5IMinusFive
+ 0x30	24	Acc5IMinusSeven
+ 0x34	24	AccIPlusSeven
+ 0x38	24	AccIPlusFive
+ 0x3C	24	AccIPlusThree
+ 0x40	24	AccIPlusOne
+ 0x44	24	AccIMinusOne
+ 0x48	24	AccIMinusThree
+ 0x4C	24	AccIMinusFive
+ 0x50	24	AccIMinusSeven
+ 0x54	24	AccQPlusSeven
+ 0x58	24	AccQPlusFive
+ 0x5C	24	AccQPlusThree

Address Offset	Size [bits]	Name
+ 0x60	24	AccQPlusOne
+ 0x64	24	AccQMinusOne
+ 0x68	24	AccQMinusThree
+ 0x6C	24	AccQMinusFive
+ 0x70	24	AccQMinusSeven

7.4.3.1 PLD5IInputSel

PLD5IInputSel				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	reserved	0	R	Reserved
1-0	InputSel	0	R/W	0 = Levels are measured from IM0 1 = Levels are measured from IM1 2 = Levels are measured from IM2 3 = Levels are measured from IM3 new values written in in InputSel become effective with the next trigger (ME or acc. Samples)

7.4.3.2 PLD5ICtrl

PLD5ICtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-25	reserved	0	R	Reserved
24	SelectTrigger	0	R/W	0 = Accumulation Registers are latched after AccTime Samples 1 = Accumulation Registers are latched with MEO
23-0	AccTime	0	R/W	Accumulation Time in Samples

7.4.3.3 PLDIQInputSel

PLDIQInputSel				
Bit	**Field**	**Default**	**R/W**	**Description**
31-8	reserved	0	R	Reserved
7-6	InputSel	0	R/W	0 = Input Modules after Input Format Converter 1 = Channels before Final Down Converter 2 = Channels after Final Down Converter 3 = reserved
5-0	SignalSel	0	R/W	n = Input Module n or Channel n would be selected selections which do not make sense (e.g. Input Module 20) are reserved new values written in in InputSel and SignalSel become effective with the next trigger (ME or acc. Samples)

7.4.3.4 PLDIQPreAccCtrl

PLDIQPreAccCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-14	reserved	0	R	reserved
13-4	Pause	0	R/W	number of cycles the preaccumulation will be paused
3	ReQuantFactor	0	R/W	0 = pre accumulator is requantised with 2^*ExpFactor*

PLDIQPreAccCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = pre accumulator is requantised with 2^(2*ExpFactor)
2-0	ExpFactor	0	R/W	the valid range for *ExpFactor* is 0..5 an *ExpFactor* of 6..7 is reserved

7.4.3.5 PLDIQCtrl

PLDIQCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-25	reserved	0	R	reserved
24	SelectTrigger	0	R/W	0 = Accumulation Registers are latched after AccTime Samples 1 = Accumulation Registers are latched with MEO
23-0	AccTime	0	R/W	Accumulation Time in Samples. If pre-accumulation is enabled, AccTime is given as the number of pre-accumulated samples.

7.4.3.6 PLD Accumulation Registers

The register description is valid for the following registers: Acc5IPlusSeven, Acc5IPlusFive, Acc5IPlusThree, Acc5IPlusOne, Acc5IMinusOne, Acc5IMinusThree, Acc5IMinusFive, Acc5IMinusSeven, AccIPlusSeven, AccIPlusFive, AccIPlusThree, AccIPlusOne, AccIMinusOne, AccIMinusThree, AccIMinusFive, AccIMinusSeven, AccQPlusSeven, AccQPlusFive, AccQPlusThree, AccQPlusOne, AccQMinusOne, AccQMinusThree, AccQMinusFive, AccQMinusSeven.

PLD Accumulation Registers				
Bit	**Field**	**Default**	**R/W**	**Description**
31-24	reserved	0	R	reserved
23-0	Value	N/A	R	Accumulated value over the measuring interval

7.4.4 Channel Matrix Registers

Address Offset	Size [bits]	Name
+ 0x00	32	ChActivation0
+ 0x04	4	ChActivation1
+ 0x08	2	DBFInputSel
+ 0x0C	27	EpochClkDiv
+ 0x10	16	MESettings
+ 0x14	17	PPSSettings
+ 0x18	17	AntSwitchCtrl
+ 0x1C	4	ExtClkSettings
+ 0x20	32	ExtClkCnt
+ 0x24	32	ExtClkCntLatched
+ 0x28	32	IMT_LSW
+ 0x2C	32	IMT_MSW
+ 0x30	32	ME_IMT_LSW
+ 0x34	32	ME_IMT_MSW
+ 0x38	32	PPS_IMT_LSW
+ 0x3C	32	PPS_IMT_MSW
+ 0x40	32	ASE_IMT_LSW
+ 0x44	32	ASE_IMT_MSW
+ 0x48	32	PLD_5I_IMT_LSW
+ 0x4C	32	PLD_5I_IMT_MSW

+ 0x50	32	PLD_IQ_IMT_LSW
+ 0x54	32	PLD_IQ_IMT_MSW
+ 0x58	32	AUT_IMT_LSW
+ 0x5C	32	AUT_IMT_MSW
+ 0x60	19	TestSettings

7.4.4.1 ChActivation0

ChActivation0				
Bit	**Field**	**Default**	**R/W**	**Description**
31		0	R/W	0 = Channel #31 is disabled 1 = Channel #31 is enabled
…				**…**
0		0	R/W	0 = Channel #0 is disabled 1 = Channel #0 is enabled

7.4.4.2 ChActivation1

ChActivation1				
Bit	**Field**	**Default**	**R/W**	**Description**
31-4	reserved	0	R	reserved
3		0	R/W	0 = Channel #35 is disabled 1 = Channel #35 is enabled
…				**…**
0		0	R/W	0 = Channel #32 is disabled 1 = Channel #32 is enabled

7.4.4.3 DBFInputSel

DBFInputSel				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	reserved	0	R	reserved
1	DBF1Sel	0	R/W	0 = IM3 and IM2 are processed 1 = IM3 and IM1 are processed
0	DBF0Sel	0	R/W	0 = IM0 and IM1 are processed 1 = IM0 and IM2 are processed

7.4.4.4 EpochClkDiv

EpochClkDiv				
Bit	**Field**	**Default**	**R/W**	**Description**
31-27	reserved	0	R	reserved
26-10	IntPart	0	R/W	Integer Part of Epoch Clock Divider Epoch Clock = *CoreClk* / (DivRatio + 1) new settings become effective with the next Epoch Clock
9-0	FracPart	0	R/W	fractional part of Epoch Clk Divider new settings become effective with the next Epoch Clock

7.4.4.5 MESettings

MESettings				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	reserved
15	ExtInputEn	0	R/W	0 = ME is generated from local ME Divider 1 = ME is generated from MEI
14	OutputEn	0	R/W	0 = MEO generation is disabled 1 = MEO generation is enabled
13-0	DivRatio	0	R/W	ME Clock = Epoch Clock / (DivRatio + 1) a new setting becomes effective with the next ME

7.4.4.6 PPSSettings

PPSSettings				
Bit	**Field**	**Default**	**R/W**	**Description**
31-17	reserved	0	R	reserved
16	SyncEn	0	W	0 = do nothing 1 = PPS strobe is synchronized to ME strobe once
15	ExtInputEn	0	R/W	0 = PPS is generated from local PPS Divider 1 = PPS is generated from PPSI
14	OutputEn	0	R/W	0 = PPSO generation is disabled 1 = PPSO generation is enabled
13-0	DivRatio	0	R/W	PPS Clock = Epoch Clock / (DivRatio + 1) a new setting becomes effective with the next PPS

7.4.4.7 AntSwitchCtrl

AntSwitchCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-17	reserved	0	R	reserved
16-15	SwitchID	0	R	ID of Ant. which was active during the last AntEpoch 0 = Ant #1 1 = Ant #2 2 = Ant #3 3 = Ant #4
14-11	SwitchSequencer	0	R/W	each bit represents one antenna. Bit0 = Ant0, Bit1 = Ant1, …
10	OutputEn	0	R/W	0 = ASEO generation is disabled 1 = ASEO generation is enabled
9-0	DivRatio	0	R/W	Ant Switch Clock = Epoch Clock / (DivRatio + 1)

7.4.4.8 ExtClkSettings

ExtClkSettings				
Bit	**Field**	**Default**	**R/W**	**Description**
31-4	reserved	0	R	reserved
3	LatchAtPPS	0	R/W	0 = Counter is not latched with PPS 1 = Counter is latched with PPS
2	LatchAtME	0	R/W	0 = Counter is not latched with ME 1 = Counter is latched with ME

ExtClkSettings				
Bit	**Field**	**Default**	**R/W**	**Description**
1	LatchAtExtClk	0	R/W	0 = Counter is not latched with trigger from *EXT_CLK* pin 1 = Counter is latched with trigger from *EXT_CORE_CLK pin*
0	SignalSel	0	R/W	0 = Counter is running with *EXT_CLK* pin 1 = Counter is running with *EXT_CORE_CLK pin*

7.4.4.9 ExtClkCnt

ExtClkCnt				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	current value of External Clock Counter

7.4.4.10 ExtClkCntLatched

ExtClkCntLatched				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	latched value of External Clock Counter

7.4.4.11 IMT_LSW

IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	Read: Current value of IMT low Write: Preset value becoming effective with the next ME

7.4.4.12 IMT_MSW

IMT_MSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	Read: Current value of IMT high Write: Preset value becoming effective with the next ME

7.4.4.13 ME_IMT_LSW

ME_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT low at ME time instance

7.4.4.14 ME_IMT_MSW

ME_IMT_MSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT high at ME time instance

7.4.4.15 PPS_IMT_LSW

PPS_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT low at PPS time instance

7.4.4.16 PPS_IMT_MSW

PPS_IMT_MSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT high at PPS time instance

7.4.4.17 ASE_IMT_LSW

ASE_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT low at ASE time instance

7.4.4.18 ASE_IMT_MSW

ASE_IMT_MSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT high at ASE time instance

7.4.4.19 PLD_5I_IMT_LSW

PLD_5I_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT low at PLD 5I time instance

7.4.4.20 PLD_5I_IMT_MSW

PLD_5I_IMT_MSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT high at PLD 5I time instance

7.4.4.21 PLD_IQ_IMT_LSW

PLD_IQ_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT low at PLD IQ time instance

7.4.4.22 PLD_IQ_IMT_MSW

PLD_IQ_IMT_MSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT high at PLD IQ time instance

7.4.4.23 AUT_IMT_LSW

AUT_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT low at AUT (Aiding Unit Trigger) time instance

7.4.4.24 AUT_IMT_MSW

AUT_IMT_MSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT high at AUT (Aiding Unit Trigger) time instance

7.4.4.25 TestSettings

TestSettings				
Bit	**Field**	**Default**	**R/W**	**Description**
31-19	reserved	0	R	reserved
18-13	CodeOut2Sel	63	R/W	0 = PRN Code and IntEpoch of channel 0 is output 1 = PRN Code and IntEpoch of channel 1 is output … 35 = PRN Code and IntEpoch of channel 35 is output 36..63 switches CODE_OUT2[1:0] to ground (saves power, since pins are not toggling)
12-7	CodeOut1Sel	63	R/W	0 = PRN Code and IntEpoch of channel 0 is output 1 = PRN Code and IntEpoch of channel 1 is output … 35 = PRN Code and IntEpoch of channel 35 is output 36..63 switches CODE_OUT1[1:0] to ground (saves power, since pins are not toggling)
6	SignalOutPosSel	1	R/W	0 = Signal before FDC is output 1 = Signal after FDC is output
5-0	SignalOutChanSel	63	R/W	0 = Input Data of channel 0 before/after FDC is output 1 = Input Data of channel 1 before/after FDC is output … 35 = Input Data of channel 35 before/after FDC is output 36..63 switches SIGNAL_OUT_I[2:0] and SIGNAL_OUT_Q[2:0] to ground (saves power, since pins are not toggling)

7.4.5 Channel Registers

Address Offset	Size [bits]	Name
+ 0x000	9	ChannelCtrl
+ 0x004	32	CarrSwFreq
+ 0x008	32	CarrSwShift
+ 0x00C	32	CodeSwFreq
+ 0x010	32	CodeSwShift
+ 0x014	27	NCOSettings
+ 0x018	20	CodeGenUnitCtrl
+ 0x01C	28	PrimCodeRam1Ctrl
+ 0x020	26	PrimCodeRam2Ctrl
+ 0x024	28	SecCodeRam1Ctrl
+ 0x028	28	SecCodeRam2Ctrl
+ 0x02C	28	VFCGExtTaps
+ 0x030	28	VFCGInit
+ 0x034	28	VFCGLength

+ 0x038	31	DelayLineCtrl
+ 0x03C	19	CorrUnitCtrl
+ 0x040	29	IntCountCtrl
+ 0x044	32	ContCntOffset
+ 0x048	22	AidingUnitCtrl
+ 0x04C	32	CarrAidFreq
+ 0x050	24	CarrAidAcc
+ 0x054	25	CodeAidFreq
+ 0x058	17	CodeAidAcc
+ 0x05C	16	LoopState
+ 0x060	32	IE_IMT_LSW
+ 0x064	29	IE_ValueEE_I
+ 0x068	29	IE_ValueEE_Q
+ 0x06C	29	IE_ValueE_I
+ 0x070	29	IE_ValueE_Q
+ 0x074	29	IE_ValueP_I
+ 0x078	29	IE_ValueP_Q
+ 0x07C	29	IE_ValueL_I
+ 0x080	29	IE_ValueL_Q
+ 0x084	29	IE_ValueLL_I
+ 0x088	29	IE_ValueLL_Q
+ 0x08C	32	DataCollect
+ 0x090	32	IE_CodeFreq
+ 0x094	32	IE_CarrFreq
+ 0x098	32	IE_CarrObsPhase
+ 0x09C	32	IE_ContCount
+ 0x0A0	32	IE_CodePhase
+ 0x0A4	32	ME_IMT_LSW
+ 0x0A8	32	ME_CarrObsPhase
+ 0x0AC	21	ME_IntCount
+ 0x0B0	32	ME_ContCount
+ 0x0B4	32	ME_CodePhase
...		
+ 0x800	23	GNSS_DMACtrl
+ 0x804	32	GNSS_DMAStartAddr
+ 0x808	32	GNSS_DMAEndAddr
+ 0x80C	32	GNSS_DMACurAddr

7.4.5.1 ChannelCtrl

ChannelCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-9	reserved	0	R	reserved
8	TimeBaseSel	0	R/W	0 = PPS, ME and ASE from local channel 1 = PPS, ME and ASE from previous channel (slaving)
7	IntEpochSel	0	R/W	0 = IE and LE from local channel 1 = IE and LE from previous channel (slaving)
6	CodeSel	0	R/W	0 = Code NCO signal from local channel 1 = Code NCO signal from previous channel (slaving)
5	CarrSel	0	R/W	0 = Carrier NCO signal from local channel 1 = Carrier NCO signal from previous channel (slaving)

ChannelCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
4-0	InputSel	0	R/W	0 = first bypass of DBF0 (IM 0) 1 = second bypass of DBF0 (IM 1/2) 2 = first bypass of DBF1 (IM 2/1) 3 = second bypass of DBF1 (IM 3) 4 = reserved 5 = reserved 6 = IM0 DDCaux 7 = IM1 DDCaux 8 = IM2 DDCaux 9 = IM3 DDCaux 10 = reserved 11 = reserved 12 = data from previous channel (slaving) 13 = data from DBF0 (-135°) 14 = data from DBF0 (-90°) 15 = data from DBF0 (-45°) 16 = data from DBF0 (-0°) 17 = data from DBF0 (+45°) 18 = data from DBF0 (+90°) 19 = data from DBF0 (+135°) 20 = data from DBF1 (-135°) 21 = data from DBF1 (-90°) 22 = data from DBF1 (-45°) 23 = data from DBF1 (-0°) 24 = data from DBF1 (+45°) 25 = data from DBF1 (+90°) 26 = data from DBF1 (+135°) 27 - 31 = reserved

7.4.5.2 CarrSwFreq

CarrSwFreq				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	Signed Carrier NCO Phase Increment (software part, not including the aiding part)

7.4.5.3 CarrSwShift

CarrSwShift				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	Write: Signed Carrier NCO Phase Increment which is added once to the Carrier NCO Read: Current Carrier NCO Phase (signed value)

7.4.5.4 CodeSwFreq

CodeSwFreq				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	Unsigned Code NCO Phase Increment (software part, not including the aiding part)

7.4.5.5 CodeSwShift

CodeSwShift				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	Write: Signed Code NCO Phase Increment which is added to the Code NCO once or repeatedly (software part, not including the aiding part) Read: Unsigned Current Code NCO Phase

7.4.5.6 NCOSettings

NCOSettings				
Bit	**Field**	**Default**	**R/W**	**Description**
31-27	reserved	0	R	reserved
26-5	CodeSw ShiftSteps	0	R/W	number of shift steps
4-3	CodeSw ShiftMode	0	R/W	0 = new value becomes effective with next IE 1 = new value becomes effective immediately 2 = new value becomes effective repeated with every IE 3 = reserved
2	CodeSw FreqMode	0	R/W	0 = new value becomes effective with next IE 1 = new value becomes effective immediately Note: In order to get the channel running from its reset state or if the Code NCO Frequency is set to zero during operation it is necessary to set this bit to immediate mode until the Code NCO is running. Otherwise there is a deadlock per definition. If the Code NCO frequency is zero there will never be an integration epoch and therefore also no new Code NCO values would become efefctive, since no integration epoch would occur.
1	CarrSw ShiftMode	0	R/W	0 = new value becomes effective with next IE 1 = new value becomes effective immediately
0	CarrSw FreqMode	0	R/W	0 = new value becomes effective with next IE 1 = new value becomes effective immediately

7.4.5.7 CodeGenUnitCtrl

CodeGenUnitCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-20	reserved	0	R	reserved
19	CodeRamSel	0	R/W	0 = CM1 is routed to output1, CM2 is routed to output2. If coupled output is routed to output 1 1 = CM1 is routed to output2, CM2 is routed to output1. If coupled output is routed to output 2
18	BOCCosEn	0	R/W	0 = BOC sine is selected 1 = BOC cosine is selected
17	BOCEffective	0	R/W	0 = BOCDivider + BOCPattern becomes effective with next IE 1 = BOC Divider + BOCPattern becomes effective immediately
16-13	BOCDivider	0	R/W	Primary Rate = NCO Rate / (BOCDivider + 1)
12-11	BOCPattern	0	R/W	BOC Pattern or L2C Enable/Disable Control
10	GPSL2CEn	0	R/W	0 = do nothing

CodeGenUnitCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
				1 = BOC register output is routed to Output 2
9	GPSL5En	0	R/W	0 = shift reg.1 is reloaded when length counter1 has elapsed, shift reg2 is reloaded when length counter2 has elapsed 1 = shift reg.1 is reloaded when length counter1 has elapsed, shift reg1 and reg2 are reloaded when length counter2 has elapsed
8	VFCGLongEn	0	R/W	0 = VFCG uses 2x14 bit shift registers 1 = VFCG uses 1x28 bit shift register
7	ResetSCM2	0	W	0 = do nothing (write only) 1 = SCM2 starts reading from offset with next StartOfEpoch
6	ResetSCM1	0	W	0 = do nothing (write only) 1 = SCM1 starts reading from offset with next StartOfEpoch
5	StartCM2	0	W	0 = do nothing (write only) 1 = start CM2 with next Trigger
4	StartCM1	0	W	0 = do nothing (write only) 1 = start CM1 with next Trigger
3	StartVFCG	0	W	0 = do nothing (write only) 1 = start VFCG with next Trigger (executed only once)
2-1	StartOfEpochSel	0	R/W	0 = shift register 1 of VFCG 1 = shift register 2 of VFCG 2 = primary code memory 1 3 = primary code memory 2
0	Trigger	0	R/W	0 = next Integration Epoch 1 = next Measurement Epoch

7.4.5.8 PrimCodeRam1Ctrl

PrimCodeRam1Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-28	reserved	0	R	reserved
27-14	Offset	0	R/W	at next Trigger, the primary RAM is read from Offset The Offset value is only applied once if the StartCM bit is set E.g. a programmed offset value of 1 causes the CG to start with Chip Nr.2. Note that the offset value always has to be smaller or equal than the length value. Otherwise the output of the Code Generator is invalid.
13-0	Length	0	R/W	length of the primary code - 1

7.4.5.9 PrimCodeRam2Ctrl

PrimCodeRam2Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-26	reserved	0	R	reserved
25-13	Offset	0	R/W	at next Trigger, the primary RAM is read from Offset The Offset value is only applied once if the StartCM bit is set

				E.g. a programmed offset value of 1 causes the CG to start with Chip Nr.2. Note that the offset value always has to be smaller or equal than the length value. Otherwise the output of the Code Generator is invalid.
12-0	Length	0	R/W	length of the primary code - 1

7.4.5.10 SecCodeRam1Ctrl

SecCodeRam1Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-28	reserved	0	R	reserved
27-14	Divider	0	R/W	secondary code rate = primary code rate / (Divider +1)
13-7	Offset	0	R/W	write: at next StartofEpoch the sec. RAM is read from Offset E.g. a programmed offset value of 1 causes the Secondary Code to start with Chip Nr.2. Note that the offset value always has to be smaller or equal than the length value. Otherwise the output of the Secondary Code is invalid. read: returns the current position of the secondary code pointer
6-0	Length	0	R/W	length of the secondary code - 1 A value of 0 deactivates the secondary code memory.

7.4.5.11 SecCodeRam2Ctrl

SecCodeRam2Ctrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-28	reserved	0	R	reserved
27-14	Divider	0	R/W	secondary code rate = primary code rate / (Divider +1)
13-7	Offset	0	R/W	write: at next StartOfEpoch the sec. RAM is read from Offset E.g. a programmed offset value of 1 causes the Secondary Code to start with Chip Nr.2. Note that the offset value always has to be smaller or equal than the length value. Otherwise the output of the Secondary Code is invalid. read: returns the current position of the secondary code pointer
6-0	Length	0	R/W	length of the secondary code - 1 A value of 0 deactivates the secondary code memory.

7.4.5.12 VFCGExtTaps

VFCGExtTaps				
Bit	**Field**	**Default**	**R/W**	**Description**
31-28	reserved	0	R	reserved
27-14	HighPart	0	R/W	upper 14 feedback taps (Polynom-2)
13-0	Low Part	0	R/W	lower 14 feedback taps (Polynom-1)

7.4.5.13 VFCGInit

VFCGInit				
Bit	**Field**	**Default**	**R/W**	**Description**
31-28	reserved	0	R	reserved
27-14	HighPart	0	R/W	init of upper 14 taps (init LFSR-2)

VFCGInit				
Bit	**Field**	**Default**	**R/W**	**Description**
13-0	LowPart	0	R/W	init of lower 14 taps (init LFSR-1)

7.4.5.14 VFCGLength

VFCGLength				
Bit	**Field**	**Default**	**R/W**	**Description**
31-28	reserved	0	R	reserved
27-14	HighPart	0	R/W	length of upper PRN sequence (LFSR-2 length)
13-0	LowPart	0	R/W	length of lower PRN sequence (LFSR-1 Length)

7.4.5.15 DelayLineCtrl

DelayLineCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31	reserved	0	R	reserved
30-26	SpacingLL	0	R/W	SpacingLL = Delay Line Spacing from L to LL in taps-1 SpacingLL 0 = 1 tap delay SpacingLL 1 = 2 taps delay … SpacingLL 21 = 22 taps delay SpacingLL 22..31 = reserved The Spacing becomes effective with the next integration epoch
25-21	SpacingL	0	R/W	SpacingL = Delay Line Spacing from P to L in taps-1
20-16	SpacingP	0	R/W	SpacingP = Delay Line Spacing from E to P in taps-1
15-11	SpacingE	0	R/W	SpacingE = Delay Line Spacing from EE to E in taps-1
10-6	SpacingEE	0	R/W	SpacingEE = Delay Line Spacing from Code In to EE in taps-1
5	CodeSwap	0	R/W	0 = Code1 goes through DL1, Code2 goes through DL2 1 = Code1 goes through DL2, Code2 goes through DL1
4	ClkSel	0	R/W	0 = local delay line clock is used 1 = delay line clock from previous channel is used
3	CodeOutSel	0	R/W	0 = CodeOut equals CodeIn (bypass mode) 1 = Code Out equals Output LL from local Delay Line
2	CodeInSel	0	R/W	0 = Code from local Code Generator Unit is used 1 = Code from previous channel is used
1-0	DivRatio	0	R/W	Delay Line Clock = *CoreClk* / (DivRatio + 1) new settings become effective with the next IE

7.4.5.16 CorrUnitCtrl

CorrUnitCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-19	reserved	0	R	reserved
18	IntSourceSel	0	R/W	0 = DMA and GIC is triggered by IE 1 = DMA and GIC is triggered by LE new settings become effective immediately
17	LongEpochReady	0	R	if the bit is set new LE values are available. It is cleared with the next IE.
16	DataCollectReady	0	R	if the bit is set, new data is available in the DataCollect register the bit is cleared if the register DataCollect is read

CorrUnitCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
15	IE_CarrObsSel	0	R/W	0 = IE_CarrObs register contains phase MSB's and cycle cnt 1 = IE_CarrObs register contains only phase
14	ME_CarrObsSel	0	R/W	0 = ME_CarrObs register contains phase MSB's and cycle cnt 1 = ME_CarrObs register contains only phase
13	CodeFreqSel	0	R/W	0 = IE_CodeFreq Observable does include aiding component 1 = IE_CodeFreq Observable does not include aiding component
12	CarrFreqSel	0	R/W	0 = IE_CarrFreq Observable does include aiding component 1 = IE_CarrFreq Observable does not include aiding component
11	ResetDataCollect	0	W	0 = do nothing 1 = the Data Collect Register&Counter is cleared at next IE
10-6	DataCollectLength	0	R/W	DataCollectLength = Bits to collect - 1
5-4	CDLSel	0	R/W	0 = LL-Q correlates with the LL replica of the Delay Line 1 1 = LL-Q correlates with the punctual replica of Delay Line 1 2 = LL-Q correlates with the punctual replica of Delay Line 2 3 = LL-Q correlates with P of DL1 and P-inverse of DL2
3	IQSel	0	R/W	0 = LL-Q correlates with the incoming Q-path 1 = LL-Q correlates with the incoming I-path
2	ASEEn	0	R/W	0 = do nothing 1 = accumulators are working with ASE instead of IE
1	LongEpochLLQ	0	R/W	0 = all correlators are integrating over the same time period (IE) 1 = LL-Q integrates with IE, all other correlators with LongEpoch
0	ResetAtStartofEpoch	0	W	0 = do nothing 1 = Reset IntCount and Accumulators with next Start of Epoch (executed only once)

7.4.5.17 IntCountCtrl

IntCountCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	0	R	reserved
28-21	LongEpochDiv	0	R/W	Long Epoch = IntEpoch / (LongEpochDiv + 1)
20-0	AccLength	0	R/W	Accumulation length in BOC Rate cycles - 1. New written value becomes effective with the next IE

7.4.5.18 ContCntOffset

ContCntOffset				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	offset is added once with the next integration epoch

7.4.5.19 AidingUnitCtrl

AidingUnitCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-23	reserved	0	R	reserved
22-21	TriggerSel	0	R/W	0 = Written parameters become effective with the next MEO 1 = Written parameters become effective with the next PPS 2 = Written parameters become effective with the next rising edge of the AU_TRIGGER pin

AidingUnitCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
				3 = reserved
20	CodeAidEn	0	R/W	0 = Code Aiding Unit disabled (becomes effective at next IE) 1 = Code Aiding Unit enabled (becomes effective at next IE)
19	CarrAidEn	0	R/W	0 = Carrier Aiding Unit disabled (becomes effective at next IE) 1 = Carrier Aiding Unit enabled (becomes effective at next IE)
18-0	DivRatio	0	R/W	AidClk = CoreClk / (DivRatio + 1)

7.4.5.20 CarrAidFreq

CarrAidFreq				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	Carrier Aiding Frequency (Doppler) Note that the CarrAidFreq register has 38bit internally. Therefore the 6 LSB's are cleared if the upper 32bit are written by the software. Read this register returns the 32-MSB of current value of the Frequency generated in the Aiding Unit.

7.4.5.21 CarrAidAcc

CarrAidAcc				
Bit	**Field**	**Default**	**R/W**	**Description**
31-24	reserved	0	R	reserved
23-0	Value	0	R/W	Carrier Aiding Acceleration (Doppler Rate) Read this register returns the current effective value

7.4.5.22 CodeAidFreq

CodeAidFreq				
Bit	**Field**	**Default**	**R/W**	**Description**
31-25	reserved	0	R	reserved
24-0	Value	0	R/W	Code Aiding Frequency (Doppler) Note that the CodeAidFreq register has 31bit internally. Therefore the 6 LSB's are cleared if the upper 25bit are written by the software Read this register returns the 25-MSB of current value of the Frequency generated in the Aiding Unit.

7.4.5.23 CodeAidAcc

CodeAidAcc				
Bit	**Field**	**Default**	**R/W**	**Description**
31-17	reserved	0	R	reserved
16-0	Value	0	R/W	Code Aiding Acceleration (Doppler Rate) Read this register returns the current effective value

7.4.5.24 LoopState

LoopState

Bit	Field	Default	R/W	Description
31-16	reserved	0	R	reserved
15-0	Value	0	R/W	This register has no influence on the hardware and can be freely programmed

7.4.5.25 IE_IMT_LSW

IE_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		N/A	R	value of IMT low at Int Epoch time instance

7.4.5.26 IE_ValueEE_I

IE_ValueEE_I				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.27 IE_ValueEE_Q

IE_ValueEE_Q				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.28 IE_ValueE_I

IE_ValueE_I				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.29 IE_ValueE_Q

IE_ValueE_Q				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.30 IE_ValueP_I

IE_ValueP_I				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension

28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.31 IE_ValueP_Q

IE_ValueP_Q				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.32 IE_ValueL_I

IE_ValueL_I				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.33 IE_ValueL_Q

IE_ValueL_Q				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.34 IE_ValueLL_I

IE_ValueLL_I				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.35 IE_ValueLL_Q

IE_ValueLL_Q				
Bit	**Field**	**Default**	**R/W**	**Description**
31-29	reserved	N/A	R	Sign Extension
28-0		N/A	R	Note that the sign bit (Bit28) is also copied to Bit 31-29. Therefore the software doesn't have to shift after reading.

7.4.5.36 DataCollect

DataCollect				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R (*)	contains the raw symbols of the navigation message (*) writing to this register only clears the *DataCollectReady* bit in the CorrUnitCtrl register

7.4.5.37 IE_CodeFreq

IE_CodeFreq				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	If CodeFreqSel=0 then IE_CodeFreq=CodeSwFreq+CodeAidFreq If CodeFreqSel=1 then IE_CodeFreq=CodeSwFreq

7.4.5.38 IE_CarrFreq

IE_CarrFreq				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	If CarrFreqSel=0 then IE_CarrFreq=CarrSwFreq+CarrAidFreq If CarrFreqSel=1 then IE_CarrFreq=CarrSwFreq

7.4.5.39 IE_CarrObsPhase

IE_CarrObsPhase				
Bit	**Field**	**Default**	**R/W**	**Description**
31-12	CycleCnt/Phase	0	R	Carrier Cycle Count (if IE_CarrObsSel = 0) 20 MSB's of Carrier Phase (if IE_CarrObsSel = 1)
11-0	Phase	0	R	12 MSB's of Carrier Phase (if IE_CarrObsSel = 0) 12 LSB's of Carrier Phase (if IE_CarrObsSel = 1)

7.4.5.40 IE_ContCount

IE_ContCount				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	32bit continuous counter at Integration Epoch

7.4.5.41 IE_CodePhase

IE_CodePhase				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	Code Phase at Integration Epoch

7.4.5.42 ME_IMT_LSW

ME_IMT_LSW				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	value of IMT low at ME time instance

7.4.5.43 ME_CarrObsPhase

ME_CarrObsPhase				
Bit	**Field**	**Default**	**R/W**	**Description**
31-12	CycleCnt/Phase	0	R	Carrier Cycle Count (if ME_CarrObsSel = 0) 20 MSB's of Carrier Phase (if ME_CarrObsSel = 1)
11-0	Phase	0	R	12 MSB's of Carrier Phase (if ME_CarrObsSel = 0) 12 LSB's of Carrier Phase (if ME_CarrObsSel = 1)

7.4.5.44 ME_IntCount

ME_IntCount				
Bit	**Field**	**Default**	**R/W**	**Description**
31-21	reserved	0	R	reserved
20-0	Value	0	R	21bit Integration Count at Measurement Epoch

7.4.5.45 ME_ContCount

ME_ContCount				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	32bit Continuous Count at Measurement Epoch

7.4.5.46 ME_CodePhase

ME_CodePhase				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R	32bit Code Phase at Measurement Epoch

7.4.5.47 GNSS DMA Ctrl

GNSS_DMACtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-23	reserved	0	R	reserved
22	ME_CodePhase	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
21	ME_ContCount	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
20	ME_IntCount	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
19	ME_CarrObs/Phase	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
18	ME_IMT_LSW	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
17	IE_CodePhase	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
16	IE_ContCount	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
15	IE_CarrierObs/Phase	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
14	IE_CarrFreq	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
13	IE_CodeFreq	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
12	DataCollect	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
11	IE_ValueLL_Q	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
10	IE_ValueLL_I	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
9	IE_ValueL_Q	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled

GNSS_DMACtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
8	IE_ValueL_I	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
7	IE_ValueP_Q	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
6	IE_ValueP_I	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
5	IE_ValueE_Q	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
4	IE_ValueE_I	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
3	IE_ValueEE_Q	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
2	IE_ValueEE_I	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
1	IE_IMT_LSW	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled
0	LoopState	0	R/W	0 = transfer of this observable disabled 1 = transfer of this observable enabled

7.4.5.48 GNSS_DMAStartAddr

GNSS_DMAStartAddr				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	start address of the destination buffer for that channel. Has to be 32bit aligned.

7.4.5.49 GNSS_DMAEndAddr

GNSS_DMAEndAddr				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	end address of the destination buffer for that channel. Has to be 32bit aligned.

7.4.5.50 GNSS_DMACurAddr

GNSS_DMACurAddr				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0		0	R/W	current address of the DMA write pointer for that channel **Note that the current address pointer has to be set to a value which is equal or greater than the start address but smaller than the end address before the DMA is started. Also not that the Current Address Pointer has to be 32bit alinged.**

7.4.6 Channel RAM Address Map

The *PrimaryRAM1/2* and *SecondaryRAM1/2* memories are not reset.

Address Offset	Size [bits]	Name
+ 0x0000	5120	PrimaryRAM1
+ 0x0A00	5120	Primary RAM2
+ 0x1400	100	SecondaryRAM1
+ 0x1440	100	SecondaryRAM2

7.4.6.1 PrimaryRAM1

PrimaryRAM1			
Address Offset	**Bit**	**Default**	**Description**
+ 0x00	7-0	undef	Data 0 Bit 7 = Chip 1 … Bit 0 = Chip 8
	31-8	undef	reserved
+ 0x04	7-0	undef	Data 1
	31-8	undef	reserved
…	…		…
+ 0x9FC	7-0	undef	Data 639
	31-8	undef	reserved

7.4.6.2 PrimaryRAM2

PrimaryRAM2			
Address Offset	**Bit**	**Default**	**Description**
+ 0x00	7-0	undef	Data 0 Bit 7 = Chip 1 … Bit 0 = Chip 8
	31-8	undef	reserved
+ 0x04	7-0	undef	Data 1
	31-8	undef	reserved
…	…		…
+ 0x9FC	7-0	undef	Data 639
	31-8	undef	reserved

7.4.6.3 SecondaryRAM1

SecondaryRAM1			
Address Offset	**Bit**	**Default**	**Description**
+ 0x00	7-0	undef	Data 0 Bit 7 = Chip 1 … Bit 0 = Chip 8
	31-8	undef	reserved
+ 0x04	7-0	undef	Data 1
	31-8	undef	reserved
…	…		…

SecondaryRAM1			
Address Offset	**Bit**	**Default**	**Description**
+ 0x2C	7-0	undef	Data 11
	31-8	undef	reserved
+ 0x30	0-3	undef	reserved
	7-4	undef	Data 12
	31-5	undef	reserved
+ 0x34	31-0	undef	reserved
+ 0x38	31-0	undef	reserved
+ 0x3C	31-0	undef	reserved

7.4.6.4 SecondaryRAM2

SecondaryRAM2			
Address Offset	**Bit**	**Default**	**Description**
+ 0x00	7-0	undef	Data 0 Bit 7 = Chip 1 … Bit 0 = Chip 8
	31-8	undef	reserved
+ 0x04	7-0	undef	Data 1
	31-8	undef	reserved
…	**…**		**…**
+ 0x2C	7-0	undef	Data 11
	31-8	undef	reserved
+ 0x30	0-3	undef	reserved
	7-4	undef	Data 12
	31-5	undef	reserved
+ 0x34	31-0	undef	reserved
+ 0x38	31-0	undef	reserved
+ 0x3C	31-0	undef	reserved

7.4.7 GNSS Interrupt Controller Registers

Address Offset	Size [bits]	Name
+ 0x00	32	GIC_Mask0
+ 0x04	10	GIC_Mask1
+ 0x08	32	GIC_Prio0
+ 0x0C	10	GIC_Prio1
+ 0x10	32	GIC_Pend0
+ 0x14	10	GIC_Pend1
+ 0x18	32	GIC_Clear0
+ 0x1C	10	GIC_Clear1
+ 0x20	6	GIC_QueueLow
+ 0x24	6	GIC_QueueHigh
+ 0x28	16	GIC_QueueStatus

7.4.7.1 GIC_Mask0

GIC_Mask0				
Bit	**Field**	**Default**	**R/W**	**Description**
31	Ch31	0	R/W	0 = Interrupt for channel #31 is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for channel #31 is stored in Interrupt Queue and causes PIC interrupt
…				…
0	Ch0	0	R/W	0 = Interrupt for channel #0 is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for channel #0 is stored in Interrupt Queue and causes PIC interrupt

7.4.7.2 GIC_Mask1

GIC_Mask1				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	reserved
9	DMAErr	0	R/W	0 = Interrupt for DMAErr is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for DMAErr is stored in Interrupt Queue and causes PIC interrupt
8	PLD_IQ	0	R/W	0 = Interrupt for PLD_IQ is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for PLD_IQ is stored in Interrupt Queue and causes PIC interrupt
7	PLD_5I	0	R/W	0 = Interrupt for PLD_5I is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for PLD_5I is stored in Interrupt Queue and causes PIC interrupt
6	ME	0	R/W	0 = Interrupt for ME is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for ME is stored in Interrupt Queue and causes PIC interrupt
5	ASE	0	R/W	0 = Interrupt for ASE is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for ASE is stored in Interrupt Queue and causes PIC interrupt
4	PPS	0	R/W	0 = Interrupt for PPS is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for PPS is stored in Interrupt Queue and causes PIC interrupt
3	Ch35	0	R/W	0 = Interrupt for channel #35 is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for channel #35 is stored in Interrupt Queue and causes PIC interrupt
…				…
0	Ch32	0	R/W	0 = Interrupt for channel #32 is stored in Pending Register, but does not cause PIC interrupt 1 = Interrupt for channel #32 is stored in Interrupt Queue and causes PIC interrupt

7.4.7.3 GIC_Prio0

GIC_Prio0				
Bit	**Field**	**Default**	**R/W**	**Description**
31	Ch31	0	R/W	0 = Interrupt for channel #31 is assigned to level low 1 = Interrupt for channel #31 is assigned to level high
…				…
0	Ch0	0	R/W	0 = Interrupt for channel #0 is assigned to level low 1 = Interrupt for channel #0 is assigned to level high

7.4.7.4 GIC_Prio1

GIC_Prio1				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	reserved
9	DMAErr	0	R/W	0 = Interrupt for DMA error is assigned to level low 1 = Interrupt for DMA error is assigned to level high
8	PLD_IQ	0	R/W	0 = Interrupt for power level detector IQ is assigned to level low 1 = Interrupt for power level detector IQ is assigned to level high
7	PLD_5I	0	R/W	0 = Interrupt for power level detector 5I is assigned to level low 1 = Interrupt for power level detector 5I is assigned to level high
6	ME	0	R/W	0 = Interrupt for measurement epoch is assigned to level low 1 = Interrupt for measurement epoch is assigned to level high
5	ASE	0	R/W	0 = Interrupt for antenna switch epoch is assigned to level low 1 = Interrupt for antenna switch epoch is assigned to level high
4	PPS	0	R/W	0 = Interrupt for pulse per second is assigned to level low 1 = Interrupt for pulse per second is assigned to level high
3	Ch35	0	R/W	0 = Interrupt for channel #35 is assigned to level low 1 = Interrupt for channel #35 is assigned to level high
…				…
0	Ch32	0	R/W	0 = Interrupt for channel #32 is assigned to level low 1 = Interrupt for channel #32 is assigned to level high

7.4.7.5 GIC_Pend0

GIC_Pend0				
Bit	**Field**	**Default**	**R/W**	**Description**
31	Ch31	0	R	0 = Interrupt for channel #31 is not pending 1 = Interrupt for channel #31 is pending
…				…
0	Ch0	0	R	0 = Interrupt for channel #0 is not pending 1 = Interrupt for channel #0 is pending

7.4.7.6 GIC_Pend1

GIC_Pend1				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	reserved
9	DMAErr	0	R	0 = Interrupt for DMA error is not pending 1 = Interrupt for DMA error is pending
8	PLD_IQ	0	R	0 = Interrupt for power level detector IQ is not pending 1 = Interrupt for power level detector IQ is pending

GIC_Pend1				
Bit	**Field**	**Default**	**R/W**	**Description**
7	PLD_5I	0	R	0 = Interrupt for power level detector 5I is not pending 1 = Interrupt for power level detector 5I is pending
6	ME	0	R	0 = Interrupt for measurement epoch is not pending 1 = Interrupt for measurement epoch is pending
5	ASE	0	R	0 = Interrupt for antenna switch epoch is not pending 1 = Interrupt for antenna switch epoch is pending
4	PPS	0	R	0 = Interrupt for pulse per second is not pending 1 = Interrupt for pulse per second is pending
3	Ch35	0	R	0 = Interrupt for channel #35 is pending 1 = Interrupt for channel #35 is pending
…				…
0	Ch32	0	R	0 = Interrupt for channel #32 is pending 1 = Interrupt for channel #32 is pending
				Note: In order to get the latest state of IntPend1, IntPend0 has to be read before

7.4.7.7 GIC_Clear0

GIC_Clear0				
Bit	**Field**	**Default**	**R/W**	**Description**
31	Ch31	0	W	0 = Pending Bit for channel #31 is left unchanged 1 = Pending Bit for channel #31 is cleared
…				…
0	Ch0	0	W	0 = Pending Bit for channel #0 is left unchanged 1 = Pending Bit for channel #0 is cleared

7.4.7.8 GIC_Clear1

GIC_Clear1				
Bit	**Field**	**Default**	**R/W**	**Description**
31-10	reserved	0	R	reserved
9	DMAErr	0	W	0 = Pending Bit for DMA error is left unchanged 1 = Pending Bit for DMA error is cleared
8	PLD_IQ	0	W	0 = Pending Bit for power level detector IQ is left unchanged 1 = Pending Bit for power level detector IQ is cleared
7	PLD_5I	0	W	0 = Pending Bit for power level detector 5I is left unchanged 1 = Pending Bit for power level detector 5I is cleared
6	ME	0	W	0 = Pending Bit for measurement epoch is left unchanged 1 = Pending Bit for measurement epoch is cleared
5	ASE	0	W	0 = Pending Bit for antenna switch epoch is left unchanged 1 = Pending Bit for antenna switch epoch is cleared
4	PPS	0	W	0 = Pending Bit for pulse per second is left unchanged 1 = Pending Bit for pulse per second is cleared
3	Ch35	0	W	0 = Interrupt for channel #35 is left unchanged 1 = Interrupt for channel #35 is cleared
…				…
0	Ch32	0	W	0 = Interrupt for channel #32 is left unchanged 1 = Interrupt for channel #32 is cleared

7.4.7.9 GIC_QueueLow

GIC_QueueLow				
Bit	**Field**	**Default**	**R/W**	**Description**
31-6	reserved	0	R	reserved
5-0	Data	63	R	0 = channel #0 had interrupt 1 = channel #1 had interrupt **...** 34 = channel #34 had interrupt 35 = channel #35 had interrupt 36 = PPS interrupt occured 37 = ASE interrupt occured 38 = ME interrupt occured 39 = PLD_5I interrupt occured 40 = PLD_IQ interrupt occured 41 = DMA error occured 42 = reserved **...** 62 = reserved 63 = interrupt queue empty **Note:** After GNSS reset this register should be read until 0x3F (queue empty) is readback. See datasheet for more details.

7.4.7.10 GIC_QueueHigh

GIC_QueueHigh				
Bit	**Field**	**Default**	**R/W**	**Description**
31-6	reserved	0	R	reserved
5-0	Data	63	R	0 = channel #0 had interrupt 1 = channel #0 had interrupt **...** 34 = channel #34 had interrupt 35 = channel #35 had interrupt 36 = PPS interrupt occured 37 = ASE interrupt occured 38 = ME interrupt occured 39 = PLD_5I interrupt occured 40 = PLD_IQ interrupt occured 41 = DMA error occured 42 = reserved **...** 62 = reserved 63 = interrupt queue empty **Note:** After GNSS reset this register should be read until 0x3F (queue empty) is readback. See datasheet for more details.

7.4.7.11 GIC_QueueStatus

GIC_QueueStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
31-16	reserved	0	R	reserved

GIC_QueueStatus				
Bit	**Field**	**Default**	**R/W**	**Description**
15	QueueLowOverflow	0	R/W	If set, this bit indicates that the IntQueueLow was full and another low queue enabled interrupt could not be registered in the low queue anymore. The bit can be set back to zero by writing a "1" in this bit.
14	QueueHighOverflow	0	R/W	If set, this bit indicates that the IntQueueHigh was full and another high queue enabled interrupt could not be registered in the high queue anymore. The bit can be set back to zero by writing a "1" in this bit.
13-7	FifoLevelQueueLow	N/A	R	FIFO level of IntQueueLow. 0 = empty, 64 = full
6-0	FifoLevelQueueHigh	N/A	R	FIFO level of IntQueueHigh. 0 = empty, 64 = full

7.5 FFT Address Map

7.5.1 FFT RAM and Registers

Address	Size [bits]	Name
0xB0000000	32	FFTValue_0_real
0x B0000004	32	FFTValue_0_imag
0x B0000008	32	FFTValue_1_real
0x B000000C	32	FFTValue_1_imag
...		...
0x B00003F0	32	FFTValue_126_real
0x B00003F4	32	FFTValue_126_imag
0x B00003F8	32	FFTValue_127_real
0x B00003FC	32	FFTValue_127_imag
0x B0000400	1	FFTCtrl (see below)

7.5.1.1 FFTCtrl

FFTCtrl				
Bit	**Field**	**Default**	**R/W**	**Description**
31-2	reserved	0	R	Reserved
1	FFTDone	0	R	Asserted when FFT done, cleared when writing to FFTCtrl
0	startFFT	0	R/W	0 = do nothing 1 = start FFT engine. Bit is cleared to 0 when FFT is done.

7.5.1.2 FFTValue

FFTValue				
Bit	**Field**	**Default**	**R/W**	**Description**
31-0	Value	undef	R/W	Write: Input data for FFT, in two's complement format Read : Output data of FFT calculation in IEEE 754 single precision Floating Point format

8 Miscellaneous

8.1 Implementation Losses

For the AGGA-4 the following implementation losses have been found by analysis and simulation.

Code	BPSK(1) [dB]	BPSK(10) [dB]	Remark
Input Module			
DDC Part			
- IQ Mixer	0,15	0,15	
- FIR Filter	0,05	0,05	
- Aliasing (downsampling)	0,05	0,05	
- Quantization	0,16	0,16	with optimal thresholds
R2C Part			
- FIR Filter	0,02	0,02	
- Aliasing (downsampling)	0,02	0,02	
- Quantization	0,16	0,16	
Digital Beam Forming			
- 2 Quantizations steps	0,33	0,33	
Channel			
- Final Down Converter	0,11	0,11	5 bits rounded phase quantization
Correlation Losses			
- Bandwidth Limited Code	0,04	0,44	
- Jittering Code NCO	0,03	0,29	for *CoreClk* frequency = 40 MHz

For further details see [AD-03].

8.2 Radiation Mitigation

8.2.1 Flip Flops

The AGGA-4 uses radiation hardened flip flops except for the following functions which are implemented with soft flip flops:

- Delay Lines of GNSS Core (total 36*20 = 7920 flip-flops). Bit errors in the code will be filtered out by the correlation process.
- Accumulators for EE,E,P,L,LL in the GNSS core (total 36*581 = 20916 flip-flops). Errors in the correlation values, if on the MSB, can be large. However, the impact can be mitigated by plausibility checks (detection of outliers), appropriate discriminators and filters in the loop processing.
- Leon Memory interface data and check-bits buses (input and output) flip flops, total = 2* (32 + 7) = 78 flip-flops. It is therefore recommended to always enable the external memory EDAC.

8.2.2 RAM's

The following RAM's are used inside the AGGA ASIC with the following protection mechanisms:

- GNSS primary code RAM, not protected
- FFT RAM, not protected
- DSU RAM (hosting the DSU trace buffer), not protected
- Cache Memory, parity protected, see 4.7.3 for further details

In case of the GNSS primary code RAM a bit flip would not disturb the GNSS receiver operation.

Note: If the primary code RAM is switched off, a radiation hard constant "1" comes out of the memories, so that the residual code generator is not disturbed.

The GNSS secondary code RAM and the Leon register file (typically also RAM) are implemented as hard flip flops and are therefore not listed in the RAM section.

For external RAM the AGGA offers a hardware EDAC protection. See 4.7.2 for further details.

8.2.3 TMR

The Triple Module Redundant implementation is used to protect critical register against SEU (Single Even Upset). Due to the fact that the AGGA-4 design uses radiation hard technology, the use of TMR is limited to very critical registers.

8.2.3.1 Watchdog TMR

The internal Watchdog is capable to force a reset in the whole ASIC. Therefore, the mechanism that directly generates both watchdog level signals is protected with TMR. This protection consists of the triplication of the internal state machine and generating via triple voting of their results the watchdog signals.

8.2.3.2 AGGA-4 Reset TMR

The external AGGA-4 reset is used to control a counter to release the reset only when the AGGA-4 clock PLL is locked. This allows keeping the design in reset during the stabilization of the PLL. A change in one of the registers of this counter can lead to a global reset. Therefore, this counter is implemented only once but if this counter reaches the 5 wait states, it sets to high a TMR register.

8.2.3.3 GNSS CoreClkSel TMR

CoreClkSel determines whether the *CoreClk* is the external *EXT_CORE_CLK* or the result of a division by 2.5 of the *HALF_SAMPLE_CLK* pin. Due to the possibility that one of these clocks is connected to ground, a accidental change in this register whould deactivate the whole GNSS circuit. Therefore, for this single bit register, an additional TMR protection is implemented.

8.2.3.4 Secure Lock TMR

The internal lock signal shall indicate when the clock is stable and the system is ready to operate. This lock signal generated by a counter that keeps the system in reset until a given time is elapsed. An SEU in one of this counter registers could force to a general reset in the system. Therefore, additional TMR protection is implemented in this counter.

Note: A more detailed analysis on the radiation testing of AGGA-4 can be found in

8.3 Synchronization of multiple AGGA devices

In order to slave multiple AGGA-4's together in order to increase the amount of GNSS channels, the following arrangement for ME, PPS and IMT is recommended:

The intended way to have the same PPS in all AGGA-4's is:

- Provide the same GNSS Core Clock to all AGGA-4's
- Define one AGGA-4 as PPS Master.
- Connect the PPSO from the Master to the PPSI pins of all AGGA-4's
- Configure all AGGA-4's to use the external PPS from the PPSI pin (also the master)
- From now on all AGGA-4's should see the same PPS event

The intended way to have the same ME and IMT in all AGGA-4's is:

- Provide the same GNSS Core Clock to all AGGA-4's
- Define one AGGA-4 as Measurement Epoch (ME) Master.
- Connect the MEO from the Master to the MEI pins of all AGGA-4's
- Configure all AGGA-4's to use the external ME from the MEI pin (also the master)
- At time of initialization (after the AGGA's have been configured to use MEI), write "0" in the registers *IMT_LSW* and *IMT_MSW*
- The "0" IMT value will become effective with the next ME
- From now on all AGGA-4's run fully synchronized (also their IMT counters)

8.4 Known Implementation Issues and Suggested Work Arounds

8.4.1 GNSS Interrupt Queue Initialization

After reset of the GNSS core the GIC queue low and the GIC queue high can have one arbitrary entry. Therefore if the user wants to make use of these queues, it is recommended to clean the queues by reading the registers *GIC_QueueLow* and *GIC_QueueHigh* after a GNSS reset until 0x3F (queue empty) is readback.

8.4.2 Cache Freeze Alert

See section 4.9.5

8.4.3 Pending Interrupt Cleared by Ticc

See section 4.12.1.1

8.4.4 Reset and Clock Issue

If the AGGA-4 is being reset (see chapter 6.5 for further details on the reset logic) without having the two clock domains CoreClk and SysClk active at time of reset, uncontrolled GNSS DMA transfers to random addresses can occur.

The receiver board design shall therefore make sure that both clock inputs *EXT_SYS_CLK* and *EXT_CORE_CLK* are available to the chip while *POWER_ON_RESET_N* is asserted. This is true for all possible AGGA-4 input modes.

If the user wants to make use of the AGGA-4 DDC mode some additional remarks have to be taken into account. In this case the GNSS operation needs two clock domains, the *CoreClk* and the *HalfSampleClk*.

The user has two possibilities in this case:

1. *CoreClk* and *HalfSampleClk* are generated outside the AGGA-4 and assigned to the pins *EXT_CORE_CLK* and *HALF_SAMPLE_CLK*. In this case no further considerations have to be taken into account, provided that all three external clocks are available during power-on reset.
2. *HalfSampleClk* is generated externally outside the AGGA-4 and is assigned to the pin *HALF_SAMPLE_CLK*. CoreClk is generated internally by setting the *CoreClkSel* bit in the *GNSSCoreClkCtrl* register (see also chapter 6.4.2 for more details). In this case *CoreClk* is not present during power-on reset and the following startup sequence must be executed before actually making use of the GNSS core and its DMA function:

Action	Comment
Reset the AGGA-4 (e.g. power on or AGGA-4 reset)	*EXT_SYS_CLK*, *EXT_CORE_CLK* and *HALF_SAMPLE_CLK* must be available at time of reset. Note that in this particular case *EXT_CORE_CLK* can be any frequency from 1MHz to the maximum, there is no need to ensure a factor 2.5 between *HalfSampleClk* and *CoreClk*.
Program *CoreClkSel* = 1 in the *GNSSCoreClkCtrl* register	Now *CoreClk* is generated internally by dividing the *HALF_SAMPLE_CLK* input.
Execute a GNSS Reset by writing "0xDEADAFFE" into the *SwResetEnable* register, followed by a write of "0xCAFEBEBE" into the *SwResetExecute* register.	Now the GNSS core is properly initialized and ready to use.

8.4.5 Start of Code Generators

Each Code Generator (VFCG, Primary Code Memory 1, Primary Code Memory 2) has the ability to (re-)start its code generation (by setting the bits *StartVFCG*, *StartCM1*, *StartCM2* in the *CodeGenUnitCtrl* register) with either the IE or the ME depending on the *Trigger* bit in the *CodeGenUnitCtrl* register.

However the Code NCO phase is not reset at the moment of the configured trigger after arming one of the bits *StartVFCG*, *StartCM1*, *StartCM2*. Therefore the user has to take into account that the edges of the code chips generated by the (re-)started code generator can be phase shifted relative to the IE or ME by a fraction of one chip.

If the user wants to implement a particular code phase relative to the ME he is advised to either take this effect into account or to shift the code phase with the *CodeSwShift* register to the desired phase in relation.

Typically the user wants to have the Code Epoch aligned with IE. In this case the user is advised to set the *ResetAtStartOfEpoch* bit in the *CorrUnitCtrl* register once the Code has started with the trigger IE. After setting the *ResetAtStartOfEpoch* bit the IE is aligned to the Code Epoch and the sub-chip phase misalignment vanishes. Note that in this case one IE can occur with a very small delta time to its predecessor (depending on the sub-chip phase misalignement).

8.4.6 SpaceWire RX DMA loss of EOP and data when packets arrive very close

When two SPW packets arrive close to each other on the same SPW link, EOP characters may be discarded. This happens if 8 or more characters of the next packet have been received at the time the DMA controller is transfering the last character (and process the EOP) of the previous packet. In this case the EOP is not reported in the interrupt register *SpW_ModuleIntStatus* (bit *SPWx_EOP_EEP)*, the DMA is not stopped and the DMA continues with the next packet instead. It is neither possible to give a minimum latency between the packets to trigger this condition, nor is it possible to calculate the probability of the occurence of this problem, because the latency of the DMA transfer is largely application dependent. The DMA transfer latencies depend on the load of the internal AHB bus which is a shared resource of the CPU and all the DMA units in the AGGA-4.

In addition if this "loss-of-EOP" condition is triggered and the size of the old packet is NOT a multiple of 4 bytes (N*4, not including the EOP), then the first 4 bytes of the new packet are discarded.

For example, two 9-character SPW packets with data (in hex)…

- 00, 01, 02, 03, 04, 05, 06, 07, 08
- 10, 11, 12, 13, 14, 15, 16, 17, 18

… would be merged into the following 32-bit words found in the DMA area:

- 0x 00 01 02 03
- 0x 04 05 06 07
- 0x 08 *05 06 07 (last 3 bytes are dummy)*
- 0x 14 15 16 17
- 0x 18 *15 16 17 (last 3 bytes are dummy)*

The following measures are suggested as workaround:

Solution A: The error will not occur if the following conditions are fulfilled:

each SpW packet received by AGGA4 has a length of modulo 8

AND {

the receive area, defined by the two registers *SpWn_Rx_SAP* and *SpWn_Rx_EAP* is large enough to receive a complete SpW packet

OR

in the case the receive area is NOT large enough to receive a complete SpW packet OR the length of the SpW packet is unknown, then the length of the receive area has to be modulo 8

} AND

the receive area is double word aligned, i.e. *SpWn_Rx_SAP* register bits (2 to 0) have to be zero (0b000).

Solution B: The error will not occur if the defined DMA receive area is always exactly 4 bytes. As a consequence, the DMA area has to be configured and re-started after each 4 bytes received. To avoid the overhead of frequent interrupts, this could be done by a background polling routine, copying the 4-byte word to a larger SW buffer and restarting the DMA. Processing a packet could then still use the EOP interrupt, but retrieving the packet from this SW buffer.

Solution C: The error will not occur if the next SpW packet is only received, after an EOP interrupt was generated from the previous SpW packet - this means no SpW packet is in the "pipeline". This could typically be ensured by a high level flow control (e.g. the receiver acknowledges receipt with a return packet, and only then the sender(s) will be allowed to send the next packet).

8.4.7 Data Loss when utilizing both UART's in parallel.

In certain scenarios of high congestion with both UARTs enabled for RX and TX, a loss of data on one of the receivers has been observed. The problem could never be fully analysed.

The user is therefore currently advised to make not use of both UART's at the same time until the issue is clarified.

8.4.8 GNSS Code Shift error by changing mode

The GNSS code generator described in chapter 3.4.3.5 allows to perform shifting of the code phase in different modes. However when writing the bitfield *CodeSwShiftSteps* in the *NCOSettings* register to zero, in very rare cases the hardware will shift the maximum number of 4194304 steps rather than 0 steps.

Although on first hand this issue seems minor one has to be aware that it not only can lead to an unsucessfull acquisition of a particular channel, but also can have an influence on the length of the integration time (depending on the value in the register *CodeSwShift*) and thus on the CPU load in order to process the channel. The user is therefore advised to never write "0" in the bitfield *CodeSwShiftSteps*.

If the user wants to stop the automatic slewing of the code generator he is advised to set the value in the register *CodeSwShift* to zero and the value of *CodeSwShiftSteps* to "1".

8.4.9 GNSS AHB Slave initilization

Due to an undeterministic initialization of the GNSS AHB Slave after power-on, it is possible that any access to a GNSS core register can freeze the complete AMBA bus of the AGGA-4. Once this state has been reached, a system reset is required.

The following options are suggested as workaround:

- Program and enable the watchdog as shown in chapter 6.5.2. In the case the watchdog will generate a reset if this problem occurs. After a triggered watchdog reset the initialization of the GNSS AHB Slave is in a consistent state.
- After power up, just execute a software reset as shown in chapter 6.5.5. After this reset the initialization of the GNSS AHB Slave is in save state.
- Writing to GNSS Code Memory (example: address 0xA100:0000) before the first access to any GNSS register. This dummy write generates the required ready signal to resolve the dead-lock in AHB slave interface.

8.4.10 Access to non-exisiting GNSS code memories

The 36 GNSS code memories are implemented in the address range 0xA1000000 to 0xA123FFFF. Accesses to the addresses from 0xA1240000 to 0xA13FFFFF will be interpreted by the AGGA-4 as accesses to code memories,

although they are not implemented. In such a case the AHB Slave interface of the GNSS core will be frozen and the AGGA-4 is no longer accessible. Only an AGGA-4 reset will break this situation.

Any access to address 0xA1240000 to 0xA13FFFFF must therefore be avoided.

8.4.11 Cache Snooping Problem

Problem description

An anomaly has been detected in the LEON2FT data cache controller, in a corner case in conjunction with partially filled cache lines, i.e. lines allocated in the cache, of which not all valid bits are set. If the processor makes a read access to a not yet cached word of such cache line (valid bit not set), causing a read miss, and that access closely coincides with a DMA write to that same cache line causing a snoop hit, the cache may become incoherent. The cache line should be cleared by the snooper, but the DMA write address may erroneously be set valid again, even though data in memory and data in cache are different.

Problem impact

The data cache may be incoherent, and as a consequence, data or commands coming in via DMA could be missed by the processor, for example when the processor is polling a set of DMA locations, and DMA updates one of them.

The probability of occurrence seems to be generally very low, being conditioned by the coincidence of several events:

a) Existence of a partially filled cache line

b) CPU read to a not yet cached location of this cache line

c) DMA write to an already cached location of the same cache line

d) That CPU read and DMA write coincide in a narrow time window

Corrective/Preventive actions to users

Several countermeasures may be considered:

1. Do not use the cache for DMA data. This can be achieved by bypassing the cache when reading DMA locations with ASI 0x4 or 0x7. Flushing the cache before accessing any DMA locations would work as well, however will likely have a negative impact in performance.
2. Ensure cache coherency by an appropriate software protocol. For example, simultaneous read/write access is usually avoided by the common double buffering scheme for DMA data, where one buffer is written to by DMA until full, then flip (swap) writing to the other buffer, while the processor reads and processes only the buffer which is not currently attached to DMA.
3. Since the probability of occurrence seems to be very low, a use 'as-is' may be considered along with a suitable higher level fault detection and recovery scheme. For example, for commands, a timeout mechanism could be used, and data packets be protected by parity or CRC.

See also [RD-10] for further information.