Packet Telemetry Encoder (PTME) VHDL Model

Data Sheet

Prepared by Sandi Habinc

PTME-001-01
Version 0.7 rev 2
September 2005

Först Långgatan 19
413 27Göteborg
Sweden

tel +46 31 7758650
fax +46 31 421407
www.gaisler.com

Table of contents

1 INTRODUCTION

1.1 Scope

This document describes the *Packet Telemetry Encoder* (PTME) VHDL model. The objective is to describe the VHDL model at a level of detail allowing its integration into an overall system. It is not the objective to describe the VHDL model to a level of detail allowing modifications or usage of individual modules in the model hierarchy.

1.2 Introduction

The purpose of the *Packet Telemetry Encoder* (PTME) synthesizable VHDL model is to provide the user with a single module implementing the *Consultative Committee for Space Data Systems* (CCSDS) recommendations for telemetry and channel coding.

1.3 Applicable standards and limitations

The PTME model is based on the *European Space Agency* (ESA) *Procedures, Standards and Specifications* (PSS) and the CCSDS recommendations. The model has been specified to support both sets of standards as far as possible. Any discrepancies due to conflicts between the two sets have been explained in the text. At the time of writing there were no relevant documents available from the *European Cooperation for Space Standardization* (ECSS).

1.4 Commissioned and non-commissioned functions

The PTME model has been partitioned in commissioned and in non-commissioned functionalities. The non-commissioned functionalities have been classified as such because not properly verified or validated, because the interfaces of the functions are not user oriented, or because a function might be removed in a future revision of the model. Non-commissioned functions are not described in detail in this document.

The main non-commissioned functionalities are listed hereafter and are repeated in the text:
- Telemetry test interface on the boundary of the PTME
- Virtual Channel Interface (VCI) on the boundary of the PTME
- Dynamic memory allocation interface on the boundary of the PTME
- Physical addressing on the PTME Internal Bus (PIB)
- Memory test interface on the boundary of the PTME, based on the PTME Internal Bus (PIB)

1.5 Configuration at compile time and during operation

The PTME VHDL model can be configured at compile or synthesis time to <u>include</u> various functions in a design. The resulting instantiation of the design can then be configured during operation to <u>enable</u> the use of implemented functions. It is important to recognise the difference.

1.6 Not implemented

The PTME VHDL model does not implement the following:
- Telemetry Encoder does not support Reed-Solomon encoding interleave depths 3 and 8.
- The length of received Source/Telemetry Packets must be greater than the on-chip input buffers of the Telemetry Encoder (see *gLength* generic).

1.7 Applicable documents

AD1	Packet Telemetry Standard, ESA PSS-4-106, Issue 1, January 1988
AD2	Packet Telemetry, CCSDS 102.0-B-5, Issue 5, November 2000
AD3	Telemetry Channel Coding Standard, ESA PSS-04-103, Issue 1, September 1989
AD4	Telemetry Channel Coding. CCSDS 101.0-B-5, Issue 6, October 2002
AD5	Radio Frequency and Modulation Standard, ESA PSS-04-105, Issue 1, Dec. 1989
AD6	Packet Telecommand Standard, ESA PSS-04-107, Issue 2, April 1992
AD7	Telecommand: Part 2 - Data Routing Service, CCSDS 202.0-B-3, June 2001
AD8	Telecommand Decoder Specification, ESA PSS-04-151, Issue 1, September 1993
AD9	AMBA^{TM} Specification, Rev 2.0, ARM IHI 0011A, 13 May 1999, Issue A, first release, ARM Limited
AD10	RS-232 EIA/TIA Standard

1.8 Applicable VHDL source code

AD11	Packet Telemetry Encoder (PTME) synthesizable VHDL model, version 0.8c, February 2004, *ptme_lib.vhd*
AD12	AMBA synthesizable VHDL package, version 0.5, February 2002, *amba.vhd*

1.9 Reference documents

RD1	Space Data Communication, ESA PSS-04-0, March 1991
RD2	Telemetry Summary Concept and Rationale, CCSDS 100.0.G-1, December 1997
RD3	Packet Telemetry Service Specification, CCSDS 103.0-B-2, Issue 2, June 2001
RD4	ESA VHDL Modelling Guidelines, ASIC/001, Issue 1, September 1994
RD5	IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993
RD6	IEEE Standard Multivalue Logic System for VHDL Model Interoperability (*Std_Logic_1164*), IEEE Std 1164-1993
RD7	IEEE Standards Interpretations: IEEE Standard VHDL Language Reference Manual, IEEE Std 1076/INT-1991
RD8	Bit Serial Encoder, E. Berlekamp, European Patent Specification, publication number 0 066 618, 24 September 1986
RD9	US4410989: Bit serial encoder, Elwyn R. Berlekamp, Cyclotomics, Inc., 11 December 1980
RD10	US5446747: Error-correction coding method with at least two systematic convolutional codings in parallel, corresponding iterative decoding method, decoding module and decoder, Claude Berrou, France Télécom, 16 April 1992

1.10 Acronyms and abbreviations

AHB	Advanced High-performance Bus (AMBA interface)
AMBA	Advanced Microcontroller Bus Architecture TM
APB	Advanced Peripheral Bus (AMBA interface)
AOS	Advanced Orbiting Systems
ASIC	Application Specific Integrated Circuit
ASM	Attached Synchronisation Marker
BAT	Bandwidth Allocation Table
CCSDS	Consultative Committee for Space Data Systems
CD	Clock Divider
CE	Convolutional Encoder
CLCW	Command Link Control Word
CRC	Cyclic Redundancy Code
DMA	Direct Memory Access
ECSS	European Cooperation for Space Standardization
EDAC	Error Detection and Correct
ESA	European Space Agency
FECW	Frame Error Control Word
FHP	First Header Pointer
FPGA	Field Programmable Gate Array
GF	Galois Field
LFSR	Linear Feedback Shift Register
NRZ	Non Return to Zero
OPCF	Operational Control Field
PA	PacketAsynchronous
PAPB	PacketAPB
PIB	PTME Internal Bus (internal PTME interface)
PP	PacketParallel
PSR	Pseudo Randomiser
PSS	Procedures, Standards and Specifications
PW	PacketWire
RSE	Reed-Solomon Encoder
SEU	Single Event Upset
SP	Split-Phase
TE	Turbo Encoder
TM	Telemetry
TME	Telemetry Encoder
VC	Virtual Channel
VCA	Virtual Channel Assembler
VCB	Virtual Channel Buffer
VCE	Virtual Channel Encoder
VCI	Virtual Channel Interface (internal PTME interface)
VCM	Virtual Channel Multiplexer
VCR	Virtual Channel Request (internal PTME interface)

1.11 Reed-Solomon code patent

The Reed-Solomon encoder implementations previously used by the European Space Agency (ESA) have been based on the bit serial encoder patented by Berlekamp under US Patent 4,410,989 (RD9) and its counterparts in other countries (RD8). That architecture exploits maximum resource sharing by implementing all Galois field operations bit serially. The core of the encoder is the bit serial multiplier which can perform the Galois field multiplication using a serial shift register and some combinatorial logic. The patent relies on the usage of a Galois field representation that has a dual basis, which enables bit serial multiplication. The ESA and CCSDS telemetry channel coding standard Galois Field representation has a dual basis, since it has been selected for envisaged implementations using the Berlekamp encoder.

The benefits of the bit serial multiplier are insignificant when implementing a monolithic encoder. A bit serial encoder approach using a parallel multiplier only adds a marginal area overhead. The benefit of such an approach is that the encoder can be used without restrictions since the Berlekamp patent is not infringed. The Reed-Solomon Encoder (RSE) described in this document is therefore based on a bit serial implementation with a parallel multiplier.

1.12 Turbo code patent

Implementers should be aware that a wide class of turbo codes is covered by a patent by France Télécom and Télédiffusion de France under US Patent 5,446,747 (RD10) and its counterparts in other countries. Potential user agencies should direct their requests for licenses to:

Mr. Christian Hamon
CCETT GIE/CVP
4 rue du Clos Courtel, BP59,
35512 CESSON SEVIGNE Cedex, France
Tel: +33 2 99 12 48 05, Fax: +33 2 99 12 40 98

1.13 Change history

In version 0.4 the ClockDivider only operates on ***BitClk***, which can be divided to generate an output bit rate frequency. A separate input has been added, ***IdleSegmentLen***, to set the Segment Length Identifier bits in Idle Transfer Frames when output on a separate Virtual Channel in order to avoid conflicts between CCSDS and ESA PSS recommendations. The Reed-Solomon E=8 (255, 239) code is fully integrated. Flexible Virtual Channel Identifier allocation has been added. Support for 16 and/or 32 bit OPCF data transfer has been added.

In version 0.5 the packet abort function is fully integrated, but the packet length check function has been removed completely. The CLCW interface functionality has been extended. The latency requirements for the Turbo encoder has been reduced. The relation between telemetry bit rate and system clock has been improved.

In version 0.6 the telemetry channel encoders are controlled by means of generics instead of constants defined in a package. For the overall PTME however, the same constants are still used to define what values are to be assigned to said generics. Improved usage of the *Conv_UnSigned* functions has been implemented in the definition package to avoid synthesis problems.

In version 0.7 a buffer empty indicator was introduced for each VCA.

2 CONVENTIONS

2.1 Advanced Microcontroller Bus Architecture

Convention according to the Advanced Microcontroller Bus Architecture (AMBA) Specification, AD9, applying to the AHB and APB interfaces:

- Signal names are in upper case, except for the following:
- A lower case '*n*' in the name indicates that the signal is active low.
- Constant names are in upper case.
- The *least* significant bit of an array is located to the *right*, carrying index number zero.

AMBA n-bit field		
most significant		least significant
n-1	n-2 down to 1	0

Table 1: *AMBA n-bit field definition*

2.2 Consultative Committee for Space Data Systems

Convention according to the Consultative Committee for Space Data Systems (CCSDS) recommendations, applying to all relevant structures:

- The *most* significant bit of an array is located to the *left*, carrying index number zero, and is transmitted first.
- An octet comprises eight bits.

General convention, applying to signals and interfaces:

- Signal names are in mixed case.
- An upper case '*_N*' suffix in the name indicates that the signal is active low.

CCSDS n-bit field		
most significant		least significant
0	1 to n-2	n-1

Table 2: *CCSDS n-bit field definition*

2.3 Galois Field

Convention according to the Consultative Committee for Space Data Systems (CCSDS) recommendations, applying to all Galois Field GF(2^8) symbols:

- A Galois Field GF(2^8) symbol comprises eight bits.
- The *least* significant bit of a symbol is located to the *left*, carrying index number zero, and is transmitted first.

Galois Field GF(2^8) symbol		
least significant		most significant
0	1 to 6	7

Table 3: *Galois Field GF(2^8) symbol definition*

2.4 Telemetry Transfer Frame format

The telemetry Transfer Frame specified in AD1 and AD2 is composed of a Primary Header, a Secondary Header, a Data Field and a Trailer with the following structures.

Transfer Frame			
Transfer Frame Header		Transfer Frame Data Field	Transfer Frame Trailer
Primary	Secondary (optional)	ket | Packet | Pa	OPCF / FECW (optional)
6 octets	0 / 4 octets	variable	0 / 2 /4 / 6 octets
223 / 446 / 892 / 1115 / 239 / 478 / 956 / 1195 octets			

Table 4: *Transfer Frame format*

Transfer Frame Primary Header						
Frame Identification				Master Channel Frame Count	Virtual Channel Frame Count	Frame Data Field Status
Version	S/C Id	VC Id	OPCF Flag			
2 bits 0:1	10 bits 2:11	3 bits 12:14	1 bit 15	8 bits	8 bits	16 bits
2 octets				1 octet	1 octet	2 octets

Table 5: *Transfer Frame Primary Header format*

Frame Data Field Status				
Secondary Header Flag	Sync Flag	Packet Order Flag	Segment Length Id	First Header Pointer
1 bit 0	1 bit 1	1 bit 2	2 bits 3:4	11 bits 5:15
2 octets				

Table 6: *Part of Transfer Frame Primary Header format*

Transfer Frame Secondary Header (optional)		
Secondary Header Identification		Secondary Header Data
Secondary Header Version	Secondary Header Length	Virtual Channel Frame Count
2 bits 0:1	6 bits 2:7	24 additional bits 0:23
1 octet		3 octets

Table 7: *Transfer Frame Secondary Header format*

Transfer Frame Trailer (optional)	
Operational Control Field (optional)	Frame Error Control Word (optional)
0 / 4 octets	0 / 2 octets

Table 8: *Transfer Frame Trailer format*

2.5 Reed-Solomon encoder data format

The applicable standards AD3 and AD4 specify a Reed-Solomon E=16 (255, 223) code resulting in the frame lengths and codeblock sizes listed in table 9.

Interleave depth	Attached Synchronisation Marker	Transfer Frame	Reed-Solomon Check Symbols
1	4 octets	223 octets	32 octets
2		446 octets	64 octets
3		669 octets	96 octets
4		892 octets	128 octets
5		1115 octets	160 octets
8		1784 octets	256 octets

Table 9: *Reed-Solomon E=16 codeblocks with Attached Synchronisation Marker*

The applicable standards AD4 also specifies a Reed-Solomon E=8 (255, 239) code resulting in the frame lengths and codeblock sizes listed in table 10.

Interleave depth	Attached Synchronisation Marker	Transfer Frame	Reed-Solomon Check Symbols
1	4 octets	239 octets	16 octets
2		478 octets	32 octets
3		717 octets	48 octets
4		956 octets	64 octets
5		1195 octets	80 octets
8		1912 octets	128 octets

Table 10: *Reed-Solomon E=8 codeblocks with Attached Synchronisation Marker*

2.6 Turbo encoder data format

The applicable standard AD4 specifies four Turbo code information block lengths *k*. It also specifies four code rates *r*. The resulting Turbo encoded block size is dependent on both the information block length and the code rate, $n=32/r + (k+4)/r$, as shown in table 13.

Information block length [bits]	Information block length [octets]	Reed-Solomon interleave depth
1784	223	1
3568	446	2
7136	892	4
8920	1115	5

Table 11: *Turbo encoder information block length as per AD4*

Information block length k [bits]	Codeblock length n [bits]			
	rate 1/2	rate 1/3	rate 1/4	rate 1/6
1784	3576	5364	7152	10728
3568	7144	10716	14288	21432
7136	14280	21420	28560	42840
8920	17848	26772	35696	53544

Table 12: *Turbo codeblock lengths as per AD4*

Attached Synchronisation Marker	Turbo Codeblock	
	Encoded Transfer Frame	Trellis Termination
32/r bits	k/r bits	4/r bits

Table 13: *Structure of a Turbo encoded block as per AD4*

2.7 Attached Synchronisation Marker

The Attached Synchronisation Marker pattern depends on the encoding scheme in use, as specified in AD4 and shown in table 14.

Mode	Hexadecimal stream (left to right)
Nominal	$1ACFFC1D_h$
Alternative	$352EF853_h$
Rate 1/2 turbo encoded	$034776C7272895B0_h$
Rate 1/3 turbo encoded	$25D5C0CE8990F6C9461BF79C_h$
Rate 1/4 turbo encoded	034776C7272895B0 $FCB88938D8D76A4F_h$
Rate 1/6 turbo encoded	25D5C0CE8990F6C9461BF79C $DA2A3F31766F0936B9E40863_h$

Table 14: *Attached Synchronization Marker hexadecimal pattern*

2.8 Command Link Control Word

The Command Link Control Word (CLCW) can be transmitted as part of the Operation Control Field (OPCF) in a Transfer Frame Trailer. The CLCW is specified in AD6 and AD7 and is listed in table 15, table 16 and table 17.

Command Link Control Word	
Static part	Dynamic part
Generated by the Telemetry Encoder (TME), or from external data interface	From external data interface
0:15	16:31
16 bits	16 bits

Table 15: *Command Link Control Word overview*

Static part of the Command Link Control Word					
Control Word Type	Version Number	Status Field	COP in Effect	Virtual Channel Identifier	Reserved Field
0	1:2	3:5	6:7	8:13	14:15
1 bit	2 bits	3 bits	2 bits	6 bits	2 bits

Table 16: *Command Link Control Word static part*

Dynamic part of the Command Link Control Word							
No RF Available	No Bit Lock	Lock Out	Wait	Retransmit	FARM B Counter	Report Type	Report Value
16	17	18	19	20	21:22	23	24:31
1 bit	1 bit	1 bit	1 bit	1 bit	2 bits	1 bit	8 bits

Table 17: *Command Link Control Word dynamic part*

2.9 Source Packet

The Source Packet defined in the ESA PSS AD1 standard and the CCSDS AD2 recommendation is used in this document and is listed in table 18, although the Segmentation Flags are interpreted differently in the two referenced documents. The Telemetry Packet defined in ESA PSS AD1 standard has also this same format, although the Packet Length and Data Field are interpreted differently. The differences have no effect on the commissioned part of the Telemetry Encoder (TME).

Source Packet / Telemetry Packet									
Packet Header							Packet Data Field		
Packet Identification				Packet Sequence Control		Packet Length	Data Field Header (optional)	Source Data	Packet Error Control (optional)
Version Number	Type	Data Field Header Flag	Application Process Id	Segmentation Flags	Source Sequence Count				
0:2	3	4	5:15	16:17	18:31	32:47			
3 bits	1 bit	1 bit	11 bits	2 bits	14 bits	16 bits	variable	variable	variable

Table 18: *Source Packet and Telemetry Packet format*

2.10 Asynchronous bit serial data format

The asynchronous bit serial interface complies to the data format defined in AD10. It also complies to the data format and waveform shown in table 19 and figure 1. The interface is independent of the transmitted data contents. Positive logic is considered for the data bits. The number of stop bits can optionally be either one or two. The parity bit can be optionally included, although the value of the bit is not used in the design described this document.

Asynchronous bit serial format	start	D0	D1	D2	D3	D4	D5	D6	D7	parity	stop	stop
	first	*lsb*							*msb*			*last*
General data format ***i*** **= {0,** ***n*****}**		8**i*+7	8**i*+6	8**i*+5	8**i*+4	8**i*+3	8**i*+2	8**i*+1	8**i*			
		last							*first*			

Table 19: *Asynchronous bit serial data format*

2.11 Waveform formats

The Packet Telemetry Encoder (PTME) accepts and generates the waveform formats shown in the following figures.

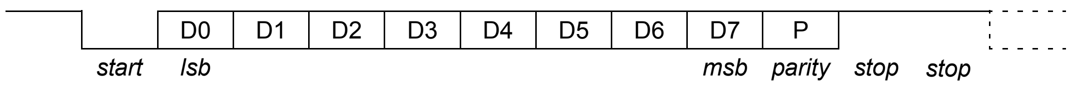

Figure 1: *Asynchronous bit serial waveform*

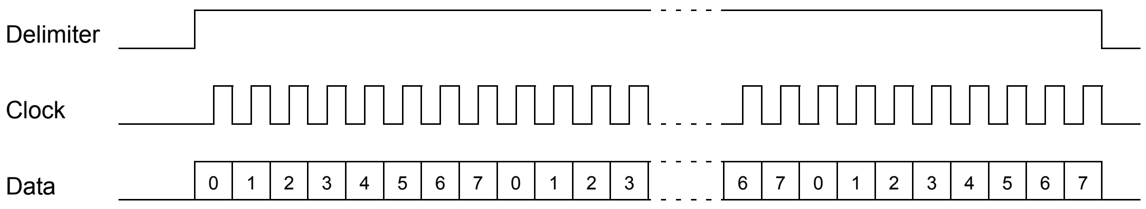

Figure 2: *Synchronous bit serial waveform*

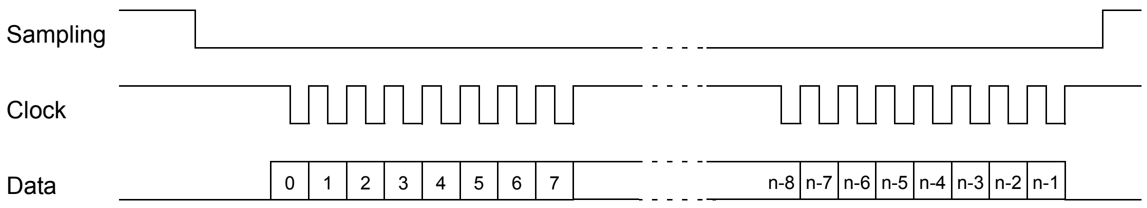

Figure 3: *Synchronous bit serial waveform, Telemetry Interface (AD8)*

3 OVERVIEW

The Packet Telemetry Encoder (PTME) VHDL model comprises several encoders and modulators implementing the Consultative Committee for Space Data Systems (CCSDS) recommendations and the European Space Agency (ESA) Procedures, Standards and Specifications (PSS) for telemetry and channel coding. The Packet Telemetry Encoder (PTME) VHDL model comprises the following:

- Telemetry Encoder (TME)
- Reed-Solomon Encoder (RSE)
- Turbo Encoder (TE)
- Pseudo-Randomiser (PSR)
- Non-Return-to-Zero Mark encoder (NRZ)
- Convolutional Encoder (CE)
- Split-Phase Level modulator (SP)
- Clock Divider (CD)

The Telemetry Encoder (TME) implements the AD1 and AD2 telemetry standards. The VHDL model implementing the TME is possible to configure to support from 1 to 8 Virtual Channels (VCs). The Reed-Solomon Encoder (RSE) implements encoding according to the AD3 and AD4 channel coding standards. The Turbo Encoder (TE) implements encoding according to the AD4 channel coding recommendation. The Pseudo-Randomiser (PSR) implements encoding according to the AD3 and AD4 channel coding standards. The Convolutional Encoder (CE) implements encoding according to the AD3 and AD4 channel coding standards. The Non-Return-to-Zero Mark encoder (NRZ) and Split-Phase Level modulator (SP) implements signal modulation according to the AD5 standard. The Clock Divider (CD) generates the bit rates and clock enable signals for the different encoders and modulators above.

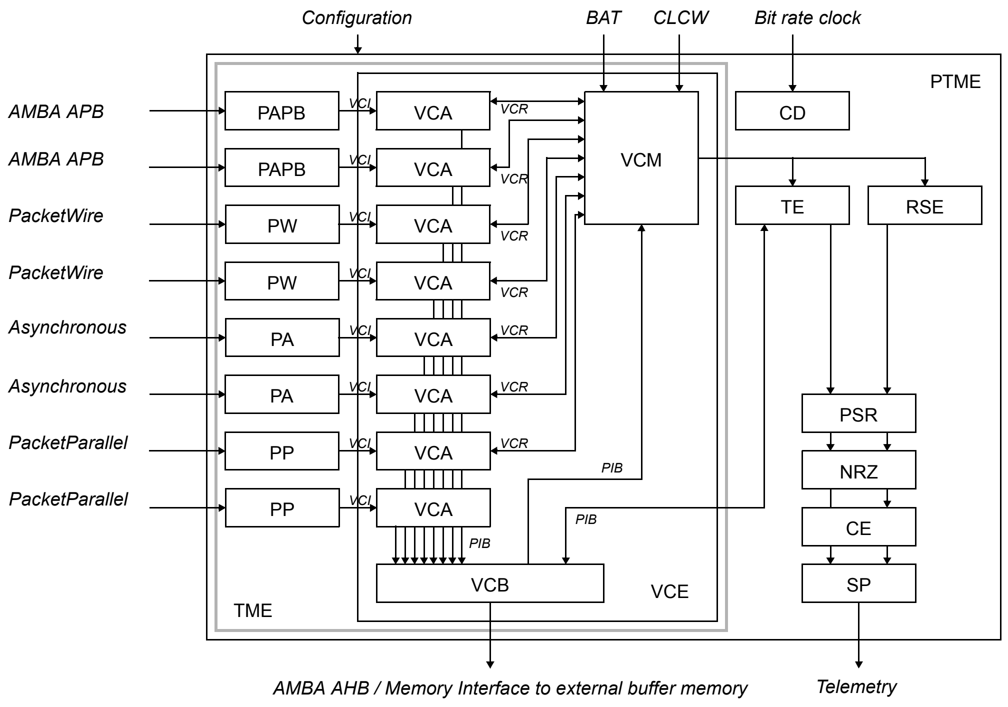

Figure 4: *PTME block diagram (example with eight Virtual Channels)*

4 MODULE SPECIFICATIONS

The Packet Telemetry Encoder (PTME) VHDL model comprises a set of modules implementing the embedded encoders and modulators. The overall specification of each module will be discussed in this section. It should be noted that it is not always possible to utilise all the specified capabilities of each module in the PTME due to architectural constraints.

4.1 Packet Telemetry Encoder (PTME)

The Packet Telemetry Encoder (PTME) module always instantiates the following modules:
- Telemetry Encoder (TME)
- Clock Divider (CD)

The PTME module can be configured at compile time to instantiate separately each of the following modules:
- Reed-Solomon Encoder (RSE)
- Turbo Encoder (TE)
- Pseudo-Randomiser (PSR)
- Non-Return-to-Zero Mark encoder (NRZ)
- Convolutional Encoder (CE)
- Split-Phase Level modulator (SP)

The PTME module provides the interconnection between the different encoders and supports all permissible coding chains. It facilitates an external telemetry test interface, although not commissioned, to allow telemetry insertion to the Reed-Solomon Encoder (RSE) and Turbo Encoder (TE), bypassing the Telemetry Encoder (TME). The symbol output of the last encoder in the coding chain is routed to the output interface of the PTME for which a re-synchronisation of the signal is performed to avoid glitch generation.

The PTME module combines the internal unidirectional data buses of the memory interface to a single bidirectional data bus interface. This is also done for the Bandwidth Allocation Table (BAT) interface data bus.

Is should be noted that the Telemetry Encoder (TME) module is not a separate entity in the PTME VHDL model. It is merely a constellation of modules related to the telemetry encoding process. The description given in this document is principally oriented around the functional structure rather than the implementation oriented structure of the PTME VHDL model. Details will be provided in section 5.1. For time being it is sufficient to state that the PTME module instantiates the components constituting the TME.

4.2 Telemetry Encoder (TME)

The Telemetry Encoder (TME) generates telemetry data according to ESA PSS standard AD1 and CCSDS recommendation AD2. The TME incorporates the user interfaces to the telemetry encoder, the control of the temporary buffering of user data, the generation of the telemetry Transfer Frame, the downlink bandwidth arbitration between Virtual Channels, and the control of subsequent encoders and modulators.

4.2.1 Implementation options selected at compile time

The TME can be configured at compile time to provide a variety of services, each option being selected individually. All configuration options are listed in section 7. The most important options are explained in this section, having a substantial impact on the implemented design.

The number of Virtual Channels to be implemented is selectable between one and eight. The Virtual Channel Identifiers are assigned automatically sequentially beginning at 0, or via external input when flexible allocation is selected. It is possible to generate Idle telemetry Transfer Frames on any of the implemented channels or on a separate Virtual Channel on which no user data can be transmitted.

The different Virtual Channels share a common external buffer memory in which data received on the user interfaces are temporarily stored. The sizing and splitting of the external buffer memory between the Virtual Channels is fixed to some extent at compile time. The following parameters are set: the overall memory size, the number of areas into which the memory is split, the number of areas that can be grouped and allocated to any Virtual Channel.

Support for alternative Attached Synchronisation Marker (ASM) generation, as per AD1, can be selected. Support for Secondary Header generation, Version-1 as per AD1, can be selected. Support for Operation Control Field (OPCF) generation can be selected. It is also possible to select non-standard 32 bit wide OPCF insertion. Selection between a serial synchronous interface or a parallel implementation for the OPCF data input is supported. Support for Frame Error Control Word (FECW) generation can be selected. Support for time strobe generation can be selected. The Bandwidth Allocation Table (BAT) is used for the arbitration of the downlink bandwidth and the following interfaces can be selected: no interface, asynchronous memory type interface, or the BAT can be implemented outside the TME (and PTME). Support for implementing 223 octet based and/or 239 octet based frame lengths is provided.

Several parameters control the implementation of each individual Virtual Channel and their interfaces. The following parameters are configurable at compile time. Support for packet handling can be selected, generating the First Header Pointer dynamically. Support for Idle Source Packet insertion can be selected. Support for external buffer memory availability indication can be selected. Support for user abort of packet input can be selected. The input interface to the Virtual Channel can be selected from one of the following options: PacketWire (PW), PacketAsynchronous (PA), PacketParallel (PP), PacketAPB (PAPB) with associated data width selection, or direct connection via the Virtual Channel Input (VCI) which is an internal interface format and is not commissioned. The number of memory areas allocated to the Virtual Channels can either be done statically at compile time or dynamically via an external interface, the latter not being commissioned.

4.2.2 Transfer Frame generation

The Telemetry Encoder (TME) generates the telemetry Transfer Frame according to AD1 and AD2 as described in the subsequent sections. Some of the fields in the Transfer Frame are optional, of which some are indicated by means of flags in the Primary Header and the inclusion of other fields is handled by management rather than by the communication protocol.

4.2.2.1 Operational configuration

The encoder allows the following fields to be enabled/disabled or varied during operation, provided that the corresponding implementation option has been selected at compile time. The encoder should be reset after a configuration change.

- Nominal or alternative Attached Synchronisation Marker
- Time strobe periodicity
- Transfer Frame length
- Spacecraft Identifier
- Virtual Channel Identifier allocation
- Transfer Frame Secondary Header (selected individually per Virtual Channel)
- Operation Control Field (selected collectively for all Virtual Channels)
- Frame Error Control Field
- Blank space generation for Reed-Solomon check symbols
- Blank space generation for Turbo code trellis termination

4.2.2.2 Attached Synchronisation Marker

The encoder always generates an Attached Synchronisation Marker (ASM) in front of the Transfer Frame. The encoder generates a 32 bit long ASM. The other ASM lengths listed in table 14, are produced by subsequent encoders. The encoder is always capable of generating the nominal ASM as listed in table 14. If selected at compile time, the encoder can also produce the alternative ASM listed in table 14. The alternative ASM can be used when a stream of Transfer Frames is embedded as data in another stream using the nominal ASM. The encoder implements one output stream for which either the nominal or alternative ASM will be produced at a time.

4.2.2.3 Transfer Frame lengths

The TME supports the Transfer Frame lengths listed in table 20.

Transfer Frame length choice	Transfer Frame lengths
000_b	223 octets
001_b	446 octets
010_b	892 octets
011_b	1115 octets
100_b	239 octets
101_b	478 octets
110_b	956 octets
111_b	1195 octets

Table 20: *Transfer Frame length alternatives*

4.2.2.4 Transfer Frame Primary Header

The Transfer Frame Primary Header format is listed in table 6.

4.2.2.4.1 Frame Identification

The Transfer Frame Version Number is permanently set to 00_b (Version-1). The Version Number can be only modified by changing a constant in the source code, which is not recommended.

The Spacecraft Identifier is configurable during operation.

The Virtual Channel Identifier is dynamically generated, depending on what Virtual Channel has been chosen for transmission in the Transfer Frame being composed. The Virtual Channel selection is described in section 4.2.3. If the TME has been configured at compile time to support flexible Virtual Channel Identifier allocation, it is possible to configure the individual Virtual Channel Identifier fields via an external input, provided that the encoder is reset in between changes. The Virtual Channel Identifier used for Idle Transfer Frames is set at compile time when flexible allocation is not selected.

The Operational Control Field Flag is configurable during operation and indicates whether or not the Operation Control Field (OPCF) is included in the Transfer Frame, provided the option is selected at compile time. The OPCF is transferred in all Transfer Frames when enabled.

4.2.2.4.2 Master Frame Counter

The Master Frame Counter is an 8 bit wide modulo 256 sequential binary counter, incremented for each transmitted Transfer Frame.

4.2.2.4.3 Virtual Channel Frame Counter

There is one Virtual Channel Frame Counter for each Virtual Channel implemented. The Virtual Channel Frame Counter is an 8 bit wide modulo 256 sequential binary counter, incremented for each Transfer Frame in which the associated Virtual Channel is transmitted. The Virtual Channel Frame Counter width can be extended to 32 bits as described in section 4.2.2.5.

4.2.2.4.4 Frame Data Field Status

The Frame Data Field Status is associated with the Virtual Channel being transmitted in the Transfer Frame.

The Secondary Header Flag indicates whether or not a Transfer Frame Secondary Header is transmitted in the Transfer Frame. It is possible to enable the transmission of the Secondary Header on a per Virtual Channel basis, as declared valid in the CCSDS AD2 recommendation.

The Data Field Synchronisation Flag indicates whether packets are synchronously inserted, when 0_b, or whether data are asynchronously inserted, when 1_b, in the Data Field of the Transfer Frame. Refer to the ESA telemetry standard AD1 for a detailed definition

The Packet Order Flag indicates in which order packets have been inserted in the Data Field. The flag is only used as a data bit by the TME and has no other effect on its operation.

The Segment Length Identifier selects the maximum field length of the standard Telemetry Packets inserted in the Transfer Frame. The identifier is only used as data bits by the TME and has no other effect on its operation.

The First Header Pointer is required for the packet chaining process during packet de-multiplexing on the ground. The encoder generates Transfer Frames in the active or idle mode as defined in AD1. The data contained in an active Transfer Frame can be either packets or non-packet data. Packets are referred to as synchronous data in AD1, whereas non-packet data are referred to as asynchronous data. Since packets may be of varying lengths, it is unlikely that an integer number of packets will be exactly contained within a Transfer Frame Data Field, some packets will consequently be split between two Transfer Frames. The encoder calculates the position of the first octet of the first packet to be placed in each Transfer Frame, in accordance with AD1, and then inserts this value into the First Header Pointer (FHP) field in the Transfer Frame. The FHP is initiated to *all ones* at reset. If a packet is longer than the data field of the Transfer Frame, the FHP is set to *all ones*. The encoder fully supports the FHP formation and does not require any alignment of the data with the Transfer Frame boundary.

Part of the Frame Data Field Status field for the Idle Transfer Frames described in section 4.2.2.6.3 is fixed in the encoder. The Data Field Synchronisation Flag is 0_b, since idle data is inserted as per AD2. The Packet Order Flag is 0_b. The First Header Pointer is set to the predefined Idle First Header Pointer code. This code is the same both for packet and non-packet data: 11111111110_b.

4.2.2.5 Transfer Frame Secondary Header

The generation of the Secondary Header specified in the ESA PSS AD1 standard is supported by the TME. If this option is selected, the support for Secondary Header generation will be implemented for all Virtual Channels. The TME allows the Secondary Header to be included or omitted individually for each Virtual Channel, as permitted in the CCSDS AD2 recommendation. The ESA PSS AD1 standard only allows the inclusion or omission to be done collectively for all Virtual Channels.

4.2.2.5.1 Secondary Header Identifier

The Secondary Header Version Number is permanently set to 00_b (Version-1). The Secondary Header Length field is permanently set to 000011_b, indicating a length for the entire Secondary Header of four octets.

4.2.2.5.2 Secondary Header Data

The Secondary Header Data field contains an additional 24 bits of the Virtual Channel Frame Counters associated with the transmitted Transfer Frame. These bits are the 24 most significant bits of the resulting extended Virtual Channel Frame Counter of 32 bits. The 8 least significant bits of the counter are transmitted in the Primary Header Virtual Channel Counter field.

MSB	32 bit Virtual Channel Frame Counter				LSB
0		23	24		31
MSB	Secondary Header Data	LSB	MSB	Virtual Channel Frame Count	LSB
0		23	0		7

Table 21: *Virtual Channel Frame Counter definition*

4.2.2.6 Transfer Frame Data Field

The data transmitted in the Transfer Frame Data Field is either user data, defined as the active state, or when no user data is available, a pseudo-random bit sequence as specified in section 4.2.2.6.3, defined as the idle state.

The information stored in the Transfer Frame Data Field in the active state can either be packets, defined as synchronous insertion, or any other data, defined as asynchronous insertion or bit streaming. The main difference between the two insertion types is the generation of the First Header Pointer, which is always dynamic for the former and normally fixed for the latter.

4.2.2.6.1 Supported formats

For the synchronous insertion, the encoder supports all packet formats defined in the CCSDS AD2 recommendation:

- Source Packet (Version Number 000_b - Version-1)
- CCSDS Network Protocol (NP) Datagram
- Internet Protocol Datagram (IPv4)
- Encapsulation Packet

For the synchronous insertion, the encoder additionally supports the packet formats defined in the ESA PSS AD1 standard:

- Source Packet (Version Number 100_b - Version-2)
- Telemetry Packet (Version Number 100_b - Version-2)

The encoder actually supports any privately defined or future data format for synchronous data insertion since being largely independent of the data contents.

The encoder supports any privately defined data format for asynchronous data insertion into the Transfer Frame Data Field. It also provides the possibility for dynamic First Header Pointer generation for asynchronous insertion, although not endorsed by the ESA PSS and CCSDS telemetry standards and recommendations.

For the Idle Source Packet insertion function described in section 4.2.5.3, both the Source Packet and the Telemetry Packet formats are supported. The Segment Length Flags are set to

11_b, which means that the Source Packet is identical to the Telemetry Packet since no segmentation is performed (Version-2) or no packet grouping is performed (Version-1). The Version Number can be set during operation.

4.2.2.6.2 Data fetch

The Transfer Frame Data Field is fetched from the external buffer memory area allocated to the Virtual Channel selected for transmission in the Transfer Frame. The Transfer Frame generation process is not concerned with the content of the data being fetched. The data reception process in the interface of each individual Virtual Channel is however concerned with the data delimiting, and in some cases the contents, as described in section 4.2.5.

4.2.2.6.3 Idle Transfer Frame generation

When there is insufficient data available from the implemented Virtual Channels to complete a Data Field, the encoder generates an Idle Transfer Frame on the fly. The Idle Transfer Frame Data Field need not to contain any useful data and is filled with a pseudo-random bit sequence. The pseudo-random generator polynomial is $h(x) = x^9+x^4+1$ and is implement as a many-to-one implementation, which is a Fibonacci version of a Linear Feed-back Shift Register (LFSR), with x^0 as output. The pseudo-random generator is free-running, but is only shifted once for each bit generated for the Data Field. The generator is initialized to *all ones* at reset and each time the associated Virtual Channel Frame Counter reaches zero. In this way the contents of the Data Field are deterministic and are possible to reproduce on ground provided that the Virtual Channel Frame Counter value is known for the Transfer Frame being analysed.

The encoder can implement a separate Virtual Channel for Idle Transfer Frame generation or use one of the Virtual Channels implemented for user data transfers. This is selectable at compile time at which the Virtual Channel Identifier is set for the Idle Transfer Frame creation, when flexible Virtual Channel Identifier allocation is not selected.

4.2.2.7 Transfer Frame Trailer

The Transfer Frame Trailer comprises two optional fields; the Operation Control Field and the Frame Error Control Word.

4.2.2.7.1 Operation Control Field

The encoder supports the generation of the Operation Control Field (OPCF) if this option is selected at compile time. It is possible to enable or disable the OPCF insertion during a mission. The OPCF is transmitted in each Transfer Frame if enabled, as specified in the ESA PSS AD1 standard and the CCSDS AD2 recommendation. It is not possible to transmit the OPCF in Transfer Frames associated with only a specific Virtual Channel, which is additionally permitted in AD2.

It is possible to input 16 and/or 32 bits of the OPCF via an external interface, which is selectable at compile time. If both options are implemented, 32 bit insertion can be explicitly enabled via an input. When 32 bit insertion is used, the format of the OPCF is under user responsibility. The format of the OPCF for 16 bit insertion is described hereafter. It is possible to overwrite bits 16 and 17 with discrete input values instead of using the information available in the CLCW. This option is enabled by means of a configuration input.

The first leading bit of the OPCF is the Type Flag. Only the Type-1-Report (Type Flag being 0_b) is supported by the encoder. The Type-1-Report contains the Command Link Control Word (CLCW) specified in AD6 and AD7 and listed in table 16 and table 17. The first static part of the CLCW is assembled by the encoder, as listed in table 22, and the second dynamic part of the CLCW is received via an external interface, as listed in table 23. The encoder supports interfacing of two CLCW sources, by providing two Virtual Channel Identifier inputs that can be associated with two different Packet Telecommand Decoders.

Dynamic/Static part of the Command Link Control Word					
Control Word Type	Version Number	Status Field	COP in Effect	Virtual Channel Identifier	Reserved Field
0	1:2	3:5	6:7	8:13	14:15
0_b	00_b	000_b	01_b	mission configurable	00_b

Table 22: *Command Link Control Word dynamic/static part assignment*

Dynamic part of the Command Link Control Word							
No RF Available	No Bit Lock	Lock Out	Wait	Retransmit	FARM B Counter	Report Type	Report Value
16	17	18	19	20	21:22	23	24:31
all bits retrieved from external interface							

Table 23: *Command Link Control Word dynamic part*

The encoder implements two interface types for the retrieval of the dynamic part of the CLCW, which are selectable at compile time. To be compatible with existing Packet Telecommand Decoders, the *Telemetry Interface* specified in AD8 is implemented. The interface is also know as the *TTC-B-01 Serial 16-Bit Digital Channel* from an older ESA document. This interface provides two synchronous bit serial channels. For future use, the encoder features a direct parallel input for providing the CLCW data, for which the interfacing to multiple sources should be implemented outside the encoder. Both interfaces support 16 and 32 bit data transfer.

4.2.2.7.2 Frame Error Control Word

The encoder supports the generation of the Frame Error Control Word (FECW) if the option is selected at compile time. It is possible to enable or disable the FECW insertion during a mission. The FECW is inserted in all Transfer Frames when enabled. The FECW contains a Cyclic Redundant Code (CRC) calculated over the Transfer Frame. The polynomial is $g(x) = x^{16}+x^{12}+x^5+1$ and is a one-to-many implementation, which is a Galois version of a Linear Feed-back Shift Register (LFSR) with x^0 as output. The LFSR is initialised to *all ones* between Transfer Frames. The field is known as the Frame Error Control Field (FECF) in AD2.

4.2.2.8 Coding space

The encoder can insert blank space between the end of the Transfer Frame and the beginning of the subsequent ASM to accommodate Reed-Solomon and Turbo coding, if individually selected at compile time. The encoder supports Transfer Frame lengths directly compatible with the E=16 (255, 223) Reed-Solomon code, as specified AD3 and AD4. The appropriate number of zero-value octets are inserted for each information block length as listed in table 9. The encoder also supports insertion of zero-value octets in amounts specified for the E=8 (255, 239) Reed-Solomon code as listed in table 10. The encoder supports insertion of four zero-value bits for the Turbo coding as specified in AD4.

4.2.3 Bandwidth allocation and selection

The encoder provides two built-in algorithms for selecting which Virtual Channel to output in the next Transfer Frame. It is not possible to perform the Virtual Channel selection externally to the encoder. The Bandwidth Allocation algorithm guarantees a minimum bandwidth for each Virtual Channel. In this algorithm the encoder uses adaptive channel ordering in order to utilize the available bandwidth as efficiently as possible. In the Priority Selection algorithm the encoder selects the highest priority Virtual Channel that has a Data Field ready for transmission.

The Bandwidth Allocation Table (BAT) is used for the arbitration of the downlink bandwidth and is used for both algorithms. The following options can be selected at compile time: internal BAT with no interface, internal BAT with asynchronous memory type interface, or the complete BAT is located outside the encoder. For the latter option, all modifications of the BAT contents must be done synchronously with the encoder clock or when a write indicator output is asserted by the encoder. The size of the BAT is selectable at compile time.

When the BAT is located in the encoder, it is initialised at reset. It will be initialised to contain an incremental sequence of values in the range of the implemented Virtual Channel Identifiers. All Virtual Channels will therefore have a nearly equal bandwidth allocated when the Bandwidth Allocation algorithm is selected. If the Priority Selection algorithm is enabled, this initialisation sequence will mean that Virtual Channel with identifier zero will have the highest priority and the Virtual Channel with the highest identifier value will have the lowest priority. For example, in an implementation with four Virtual Channels and 32 entries, the BAT will contain the following sequence after reset: 0, 1, 2, 3, 0, 1, 2, 3 and so on.

4.2.4 Time strobe

A time strobe is generated according to AD1, being asserted simultaneously with the first bit of the Attached Synchronisation Marker output. The timing of the time strobe is not valid if Turbo encoding is applied because data buffering takes place. The accuracy of the time strobe is reduced by the usage of other encoders and modulator. Usually the time strobe is being produced one bit clock period earlier for each simple encoder in use. The time strobe is asserted for 128 bit clock periods. The periodicity of the time strobe is configurable during a mission, being linked to the eight least significant bits of the Virtual Channel Frame Counter of Virtual Channel number 0, as listed in table 24.

Time strobe periodicity setting	Virtual Channel 0 Frame Count values (8 bits)
0000_b	0, 1, 2, 3, 4... 253, 254, 255
0001_b	0, 2, 4, 6, 8... 250, 252, 254
0010_b	0, 4, 8, 12, 16... 244, 248, 252
0011_b	0, 8, 16, 24, 32... 232, 240, 248
0100_b	0, 16, 32, 48, 64... 208, 224, 240
0101_b	0, 32, 64, 96, 128, 160, 192, 224
0110_b	0, 64, 128, 192
0111_b	0, 128
1---_b	0

Table 24: *Time strobe periodicity*

4.2.5 Virtual Channels

The Virtual Channels in the Telemetry Encoder (TME) are treated as separate entities, each being individually configured during operation.

4.2.5.1 Operational configuration

The encoder allows the following functions or protocol fields to be varied during operation for each Virtual Channel, provided that the corresponding design option is selected at compile time.

The following fields in the Frame Data Field Status can be configured:
- Secondary Header Flag (used as data bit and Secondary Header insertion qualifier)
- Data Field Synchronisation Flag (use as data bit and Idle Source Packet generation qualifier)
- Packet Order Flag (used as data bit only)
- Segment Length Identifier (used as data bits only) as listed in table 25

Segment Length Identifier	Maximum Data Field length of Telemetry Packets (AD1)
00_b	256 octets (only in AD1)
01_b	512 octets (only in AD1)
10_b	1024 octets (only in AD1)
11_b	65536 octets (no segmentation) (both AD1 and AD2)

Table 25: *Segment Length Identifier interpretation*

The following functions can be configured or enabled for the data reception process:
- External buffer memory allocation to the Virtual Channel (non-commissioned)
- Threshold for external buffer memory availability indication, as listed in table 26
- Dynamic or static First Header Pointer calculation, where dynamic calculation must be used with synchronous data insertion and can be used with asynchronous data insertion (although not endorsed in telemetry standards), and where static calculation must only be used with asynchronous data insertion (independent of Data Field Synchronisation Flag setting above)

Threshold selection	Available unused octets in external buffer memory
00_b	≥ 262 (256 + 6)
01_b	≥ 518 (512 + 6)
10_b	≥ 1030 (1024 + 6)
11_b	≥ length of Transfer Frame Data Field

Table 26: *Threshold setting for external buffer memory availability indication*

For the Idle Source Packet generation the following can be configured during operation:
- Version Number (most significant bit, 000_b and 100_b supported) (used as data bit only)
- Threshold for number of polls before generation commences, as listed in table 27 and explained in section 4.2.5.3

Poll threshold setting	Number of polls
000_b	Idle Source Packets always inserted (no polls required)
001_b	1
010_b	4
011_b	16
100_b	64
101_b	256
110_b	1024
111_b	Idle Source Packets never inserted

Table 27: *Poll threshold setting for Idle Source Packet insertion*

4.2.5.2 Data buffering

The Virtual Channels receive data, via the interfaces described later, which are stored in the external buffer memory. Each Virtual Channel keeps track of the number of octets received and the packet boundaries in order to calculate the First Header Pointer discussed in section 4.2.2.4.4. Data are stored in pre-allocated external buffer memory slots comprising Transfer Frame Data Fields. The Frame Data Field Status for each Transfer Frame, including the First Header Pointer, is also stored in the external buffer memory.

Since the external buffer memory is shared between Virtual Channels, each Virtual Channel is allocated a portion of the memory space. The external buffer memory allocated to each Virtual Channel is treated as a circular buffer of Transfer Frame Data Fields. The encoder provides two options for partitioning the memory space in pre-allocated slots. There is an optimised scheme and a simplified scheme, trading buffering capability versus implementation complexity, as listed in table 28. The allocation selection is nominally done at compile time, but also be done dynamically during operation via an non-commissioned interface, as described in section 5.2.3.

Transfer Frame length setting	Transfer Frame length [octets]	Allocated memory space [octets]	
		optimised	simplified
000_b	223	256 - 32 = 224	256
001_b	446	512 - 64 = 448	512
010_b	892	1024 - 128 = 896	1024
011_b	1115	1024 + 128 = 1152	2048
100_b	239	256 - 16 = 240	256
101_b	478	512 - 32 = 480	512
110_b	956	1024 - 64 = 960	1024
111_b	1195	1024 + 256 = 1280	2048

Table 28: *Allocated external buffer memory space per Transfer Frame Data Field*

The access to the external buffer memory is divided in two paths; one for storing data received from the user and one for storing auxiliary data such as the Frame Data Field Status. The auxiliary path is also used for storing Idle Source Packets as described in section 4.2.5.3. By isolating the path for user data from the path for auxiliary data it is possible to provide a guaranteed user data throughput for the user interface.

The Virtual Channels do not perform segmentation of Source Packets (Version-2), as specified in AD1, which should be done by the data source. The Virtual Channels can handle packets and data blocks with data fields of up to 65536 octets. Note that only a part of the maximum size packet will reside in the external buffer memory at a time if the memory allocated to a Virtual Channel is smaller than the 65536+6 octets.

The encoder does not check the contents or the format of the packets provided by the source. The Virtual Channel is thus independent of the packet format and any data structure can be used.

4.2.5.3 Idle Source Packet generation

The encoder can optionally generate and insert Idle Source Packets for each Virtual Channel to fill up an incomplete Transfer Frame Data Field. This ensures that user data do not remain inaccessible due to an incomplete Transfer Frame being resident in the external buffer memory. This function is only available when the Virtual Channel is operating with packets, referred to as synchronous data insertion in AD1. Idle Source Packets should not be confused with Idle Transfer Frames.

The Idle Source Packet insertion process starts when all of the following conditions are met:

- the Virtual Channel operates with packets (Data Field Synchronisation Flag is 0_b)
- the source is not sending data
- the Virtual Channel has no Data Field completed and available for transmission
- the Virtual Channel contains an incomplete Data Field (containing non-Idle Source Packets or part of a non-Idle Source Packet) that is resident in the external buffer memory
- the poll threshold is not set to 111_b
- the appropriate number of polls have been counted as listed in table 27

When the Idle Source Packet insertion process has filled the Data Field, the Virtual Channel will indicate to the encoder that it has a Data Field completed and available for transmission. The Idle Source Packet insertion process stops after the last Idle Source Packet has been stored in the Data Field in the external buffer memory. In the case that the last Idle Source Packet spilled over, it will be written to the next Data Field. The time required to complete a Data Field with Idle Source Packets depends on how much space there is left to be filled. Should the source begin to send data after the Idle Source Packet insertion process has started, the current Idle Source Packet will be completed in parallel with the storing of received data and then the Idle Source Packet insertion process will be suspended. As soon as the source stops sending data again, indicating the end of the normal packet, the Idle Source Packet insertion process resumes and continues until the Data Field has been completed (it will not stop due to a new packet being received) unless the Data Field was already completely filled with normal packets.

When a Transfer Frame Data Field contains no user packets or when it contains only part of an Idle Source Packet, the Idle Source Packet insertion process will not start. The situation where a Data Field contains only part of an Idle Source Packet occurs when an Idle Source Packet spilled over from the previous Transfer Frame and it is the only data in the new Transfer Frame.

The structure of the Idle Source Packet is listed in table 29. Note that the CCSDS AD2 standard permits simultaneous transmission of Source Packets and the other packet formats defined in the standard. This makes it possible to use Idle Source Packet insertion also with other packet formats than the Source Packets format alone. The Data Field of the Idle Source Packet is generated with a pseudo-random generator. The pseudo-random generator polynomial is $h(x) = x^9+x^4+1$ and is implement as a many-to-one implementation, which is a Fibonacci version of a Linear Feed-back Shift Register (LFSR), with x^0 as output.

Packet Identification:	
Version Number (bit 0):	configurable during operation
Version Number (bit 1 to 2):	00_b
Type (bit 3):	0_b
Data Field Header Flag (bit 4):	0_b
Application Process Identifier (bit 5 to 15):	11111111111_b
Packet Sequence Control:	
Segmentation Flags (bit 0 to 1):	11_b
Source Sequence Count (bit 2 to 15):	00000000000000_b
Packet Length:	
Packet Length (bit 0 to 15):	0000000000000111_b
Idle Source Packet Data Field:	
Idle Source Packet Data Field (bit 0 to 63):	pseudo-random data

Table 29: *Idle Source Packet structure*

4.2.5.4 Input interfaces

The Telemetry Encoder (TME) provides an internal interface format between the Virtual Channels and the user interfaces. The available user interface types are described in the subsequent sections. They all provide the same basic functionality for interfacing the Virtual Channels as will be described hereafter. The main purpose of the user interfaces is to receive data from a given communication protocol and forward the data to the Virtual Channel. Octet and packet boundaries are extracted in the process and signalled to the Virtual Channel. The data structure used by the Virtual Channel is octet based and the adaptation of any differences in data size with the input protocol is performed by the user interface modules. The user interfaces provide feed back from the Virtual Channel, signalling when there is space left in the external buffer memory for a data block of a predefined length (buffer availability indication) and when the Virtual Channel input is busy or not ready to receive any data (busy signalling).

The Telemetry Encoder (TME) provides the possibility to abort the insertion of a packet. This option is selected at compile time. A Transfer Frame is normally released for transmission as soon as its Data Field is completely filled. This can lead to parts of a packet being sent before the complete packet has been received by the encoder, making it impossible to abort the packet insertion without violating the communication protocol. The encoder provides optional support to avoid this problem by holding any Transfer Frame which contains part of a packet that is not completely received by the encoder. As long as the packet is not completely received by the encoder, signalled by the user via the packet delimiter input, the packet can be retracted by the user and the complete packet is removed from the external buffer memory. There is a discrete input provided for this function. The abort input should be de-asserted before the insertion of a packet begins, i.e. before the packet delimiter is asserted. If asserted before the end of the packet insertion, i.e. before the packet delimiter is de-asserted, the packet will be aborted and retracted. No data will be received until the packet delimiter is de-asserted and re-asserted for the insertion of a new packet. If the abort input is asserted before the insertion of a packet begins, an abort will not be possible. Transfer Frames containing part of that packet will be released for transmission as soon as they are completed. There is a possibility for a deadlock situation that has to be handled by the user. If a large packet is received which is stored in all possible Transfer Frame Data Fields in the external buffer memory space, it will not be possible for the encoder to release the corresponding frame, neither will it be possible for the user to insert the complete packet. It is possible to avoid this situation by never transferring packets for which the size is close the number of Transfer Frame Data Fields that can be stored in the external buffer memory space, or not to use the abort function for such large packets. If the situation should still occur, the only remedy is for the packet to be aborted by the user. A potential deadlock situation can be observed via the busy output that will be asserted for an infinite period of time.

All interfaces provide an indicator output signal which is asserted when no user packet data is located in the input buffers of the interfaces or in the external buffer memory. It is also asserted if only idle packet data (non-user data) is located in the external buffer memory. In this way the user can know if all user data has been transmitted.

4.2.5.4.1 PacketWire interface (PW)

The PacketWire (PW) interface to the Virtual Channel is a simple bit synchronous protocol. The data can either be packets or a bit stream. The interface comprises three input signals; bit data, bit clock and packet delimiter. There are additional discrete signals provided for busy signalling and external buffer memory availability indication.

Data should consist of multiples of eight bits otherwise the last bits will be lost. The input packet delimiter signal is used to delimit packets. It should be asserted while a packet is being input, and de-asserted in between. In addition, the input packet delimiter signal should define the octet boundaries in the input data stream, the first octet explicitly and the following octets each subsequent eight bit clock cycles. The packet delimiter can be permanently asserted when non-packet data are input, provided that the First Header Pointer calculation is static for the Virtual Channel and the corresponding Data Field Synchronisation Flag is set to 1_b.

4.2.5.4.2 PacketParallel interface (PP)

The PacketParallel (PP) interface to the Virtual Channel is a simple parallel asynchronous memory type interface. Data are written in by means of a write strobe. There are additional discrete signals provided for packet delimiting, busy signalling and external buffer memory availability indication. The interface supports 8 bit data input. The packet delimiter signal is used to delimit packets. It should be asserted while a packet is being input and de-asserted in between. The packet delimiter can be permanently asserted when non-packet data are input, provided that the First Header Pointer calculation is static for the Virtual Channel and the corresponding Data Field Synchronisation Flag is set to 1_b.

4.2.5.4.3 PacketAsynchronous interface (PA)

The PacketAsynchronous interface (PA) to the Virtual Channel is a simple bit asynchronous protocol used for receiving data only. There are no provision in the data protocol itself for packet delimiting or handshake. Instead there are some additional discrete signals provided for packet delimiting, busy signalling and external buffer memory availability indication. The packet delimiter signal is used to delimit packets. It should be asserted while a packet is being input and de-asserted in between. The packet delimiter can be permanently asserted when non-packet data are input, provided that the First Header Pointer calculation is static for the Virtual Channel and the corresponding Data Field Synchronisation Flag should is set to 1_b. The interface supports 8 bit data with 1 or 2 stop bits and a parity bit that can be masked if required although the parity value is unused. The protocol is according to AD10 and is listed in table 19 and shown in figure 1. There are four different baud rates supported which can be chosen during operation as listed in table 30. The baud rates are determined at compile time towards a fixed system clock frequency. It is thus not possible to change the system clock frequency without affecting the baud rates.

Baud rate choice	baud rate
00_b	9600 baud
01_b	19200 baud
10_b	38400 baud
11_b	57600 baud

Table 30: *Baud rates*

4.2.5.4.4 PacketAPB interface (PAPB)

The PacketAPB interface (PABP) to the Virtual Channel is compliant with the AMBA APB interface specification defined in AD9. Multiple PAPB slave interfaces can be located on the same AMBA APB bus, provided that the corresponding select signals are generated separately. The interface supports a variable width data input, although the data are output to the Virtual Channel as 8 bit wide octets. The maximum input data width is configurable for each interface at compile time and can be set to 8, 16, 24 or 32 bits. The interface allows simultaneous input of data up to the corresponding maximum input data width above. It is possible to set dynamically the data input width for each access, supporting 8, 16, 24 or 32 bits transfers, as long as being shorter than the aforementioned maximum.

Packet delimiting and handshake is performed through the PAPB Configuration Register. There are additional discrete signals provided for busy signalling and external buffer memory availability indication to reduce the number of accesses to the interface. The packet delimiter bit in the PAPB Configuration Register is used to delimit packets. It should be set while a packet is being input and cleared in between. The packet delimiter can be permanently asserted when non-packet data are input, provided that the First Header Pointer calculation is static for the Virtual Channel and the corresponding Data Field Synchronisation Flag is set to 1_b.

The input address is interpreted as a byte address, as per AD12. Since each access is a word access, the two least significant address bits are assumed always to be zero. Only address bit 10 is decoded during write accesses to allow a maximum burst of 1024 words to the PAPB Data Input Register. Misaligned addressing is not supported. The address is not decoded during read accesses, neither are the select, enable and write strobes. Only the PAPB Configuration Register can be read. The unused data bits 31:7 are always driven to zero. The interface is designed to work in a multiplexed unidirectional bus scheme. Re-mapping between the opposing numbering conventions in the CCSDS and AMBA documentation is performed.

Bit number	Mode	Default	Name	Remarks	
31:7	r	all zeros	unused	all zero during read	
6	r	0	Ready	Interface ready to receive a segment	
5	r	0	Busy	Interface busy	
4	r	0	unused	zero during read	
3	r/w	0	Valid	Packet delimiter/enable when asserted	
2	r/w	0	Abort	Abort packet	
1:0	r/w	00	Size	mode	Input data size and transmit order
				00_b	8 bit: transmit 7:0
				01_b	16 bit: transmit 15:8, 7:0
				10_b	24 bit: transmit 23:16, 15:8, 7:0
				11_b	32 bit: transmit 31:24, 23:16, 15:8, 7:0

Table 31: *PacketAPB Configuration Register bit definition*

Bit number	Mode	Default	Name	Remarks
31:24	w	n/a	data	only transmitted in 32 bit mode
23:16				only transmitted in 32 and 24 bit modes
15:8				only transmitted in 32, 24 and 16 bit modes
7:0				transmitted in all modes

Table 32: *PacketAPB Data Input Register bit definition*

Register name	Address	Read/Write	Remark	Reference
Configuration Register	---- -0--_h	r/w	configuration and status	table 31
Data Input Register	---- -4--_h	w	data transfer	table 32

Table 33: *Address mapping for PacketAPB AMBA APB slave interface*

4.3 Reed-Solomon Encoder (RSE)

The CCSDS recommendation AD4 specifies two similar Reed-Solomon codes, one (255, 223) code and one (255, 239) code. The ESA PSS standard AD3 only specifies the former code. Although the definition style differs between the AD3 and AD4, the (255, 223) code is the same in both documents. The definition used in this document is based on the ESA standard AD3.

The Reed-Solomon Encoder (RSE) implements both codes, which can be used directly in the Packet Telemetry Encoder (PTME) module.

The Reed-Solomon encoder also supports other interleave depths than those specified in AD3 and AD4.

The Reed-Solomon encoder is compliant with the two coding algorithms in AD4:

- there are 8 bits per symbol;
- there are 255 symbols per codeword;
- the encoding is systematic:
  - for E=8 or (255, 239), the first 239 symbols transmitted are information symbols, and the last 16 symbols transmitted are check symbols;
  - for E=16 or (255, 223), the first 223 symbols transmitted are information symbols, and the last 32 symbols transmitted are check symbols;
- the E=8 code can correct up to 8 symbol errors per codeword;
- the E=16 code can correct up to 16 symbol errors per codeword;
- the field polynomial is

$$f_{esa}(x) = x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$$

- the code generator polynomial for E=8 is

$$g_{esa}(x) = \prod_{i=120}^{135} (x + \alpha^i) = \sum_{j=0}^{16} g_j \cdot x^j$$

  for which the highest power of x is transmitted first;
- the code generator polynomial for E=16 is

$$g_{esa}(x) = \prod_{i=112}^{143} (x + \alpha^i) = \sum_{j=0}^{32} g_j \cdot x^j$$

  for which the highest power of x is transmitted first;
- interleaving is supported for depth $I = \{1 \text{ to } 8\}$, where information symbols are encoded as I codewords with symbol numbers $i + j*I$ belonging to codeword i {where $0 \leq i < I$ and $0 \leq j < 255$};
- shortened codeword lengths are supported;

- the input and output data from the encoder are in the representation specified by the following transformation matrix T_{esa}, where ι_0 is transferred first

$$\begin{bmatrix}\iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7\end{bmatrix} = \begin{bmatrix}\alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0\end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

- the following matrix T^{-1}_{esa} specifying the reverse transformation

$$\begin{bmatrix}\alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0\end{bmatrix} = \begin{bmatrix}\iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7\end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The encoder is slaved to the telemetry Transfer Frame generation process. The Reed-Solomon output is non-return-to-zero level encoded.

4.4 Turbo Encoder (TE)

The Turbo Encoder (TE) encodes data from the preceding Telemetry Encoder according to CCSDS recommendation AD4. It replaces the nominal Attached Synchronisation Marker (ASM) with a specific ASM as defined for each coding rate.

The Turbo encoder is compliant with the coding algorithm in AD4:

- Code type: systematic parallel concatenated turbo code.
- Number of component codes: 2 (plus an uncoded component to make the code systematic).
- Type of component codes: recursive convolutional codes.
- Number of states of each convolutional component code: 16.
- The nominal code rate is selectable with *r = 1/2, 1/3, 1/4* or *1/6* bit per symbol.
- The specified information block lengths *k* are shown in table 11. The corresponding codeblock lengths in bits, *n=(k+4)/r*, for the specified code rates are shown in table 12.
- The permutation for each block length *k* is given by a particular reordering of the integers *1, 2, . ., k* as generated by the algorithm in figure 5. The constant *k* is expressed as $k=k_1k_2$. The parameters k_1 and k_2 for the specified block sizes are given in table 34. The operations in the algorithm are performed for *s=1* to *s=k* to obtain the permutation numbers *π(s)*. In the equation below, *x* denotes the largest integer less than or equal to *x*, and p_q denotes one of the prime integers listed in table 35. The interpretation of the permutation numbers is such that the *s*th bit read out on line “in b” in figure 6 is the *π(s)*th bit of the input information block.

Information block length *k*, bits	k_1	k_2
1784	8	223 x 1
3568	8	223 x 2
7136	8	223 x 4
8920	8	223 x 5

Table 34: *Standard k_1 and k_2 parameters*

$p_1 = 31$	$p_2 = 37$	$p_3 = 43$	$p_4 = 47$	$p_5 = 53$	$p_6 = 59$	$p_7 = 61$	$p_8 = 67$

Table 35: *Prime integers*

$$i = \left\lfloor \frac{s-1}{2 \cdot k_2} \right\rfloor$$

$$j = \left\lfloor \frac{s-1}{2} \right\rfloor - ik_2$$

$$t = (19i + 1) \bmod \frac{k_1}{2}$$

$$q = t \bmod 8 + 1$$

$$c = (p_q j + 21m) \bmod k_2$$

$$m = (s-1) \bmod 2$$

$$\pi(s) = 2\left(t + c\frac{k_1}{2} + 1\right) - m$$

Figure 5: *Permuter algorithm*

- Backward and forward connection vectors (see figure 6):
  - Backward connection vector for both component codes and all code rates: G0 = 10011_b.
  - Forward connection vector for both component codes and rates 1/2 and 1/3: G1 = 11011_b.
    - Puncturing of every other symbol from each component code is necessary for rate 1/2.
    - No puncturing is done for rate 1/3.
  - Forward connection vectors for rate 1/4: G2 = 10101_b, G3 = 11111_b (1st component code); G1 = 11011_b (2nd component code). No puncturing is done for rate 1/4.
  - Forward connection vectors for rate 1/6: G1 = 11011, G2 = 10101_b, G3 = 11111_b (1st component code); G1 = 11011_b, G3 = 11111_b (2nd component code). No puncturing is done for rate 1/6.
- The Attached Synchronisation Marker (ASM) differs for the four code rates. The number of bits are proportional to the code rate, with the marker length being 32/*r*-bit for *r* = 1/2, 1/3, 1/4 and 1/6, e.g. *r* = 1/6 gives a sequence of 192 bits. As for Reed-Solomon coding, the ASM is used for frame synchronisation. The different ASMs are listed in table 14.

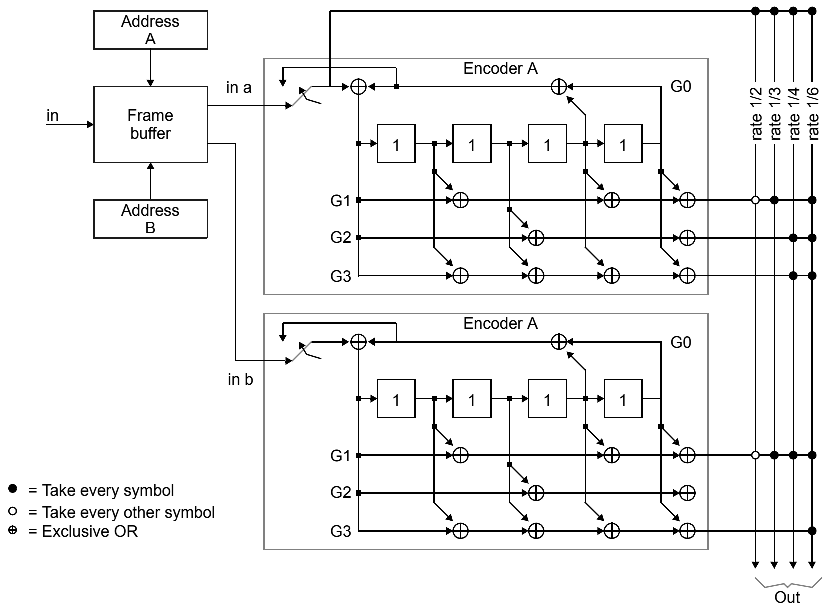

Figure 6: *Turbo Encoder functional block diagram*

The Turbo encoder output is non-return-to-zero level encoded. The encoder increases the output bit rate with a factor of two to six, depending on the selected coding rate. It is not allowed to simultaneously engage both Turbo and Reed-Solomon and/or Convolutional encoding. The Turbo encoding introduces a latency corresponding to one Transfer Frame transmission duration, since a complete Transfer Frame needs to be available and buffered before encoding can commence. The encoder is slaved to the telemetry Transfer Frame generation process.

4.5 Pseudo-Randomiser (PSR)

The Pseudo-Randomiser (PSR) generates a bit sequence according to AD3 and AD4 which is xor-ed with the data output of preceding encoders. This function allows the required bit transition density to be obtained on a channel in order to permit the receiver on ground to maintain bit synchronisation.

The polynomial for the Pseudo-Randomiser is $h(x) = x^8+x^7+x^5+x^3+1$ and is implemented as a Fibonacci version (many-to-one implementation) of a Linear Feedback Shift Register (LFSR). The registers of the LFSR are initialised to *all ones* between Transfer Frames. The Attached Synchronisation Marker (ASM) is not effected by the encoding.

4.6 Non-Return-to-Zero Mark encoder (NRZ)

The Non-Return-to-Zero Mark encoder (NRZ) encodes differentially a bit stream from preceding encoders according to AD5. The waveform is shown in figure 11.

Both data and the Attached Synchronisation Marker (ASM) are affected by the coding. When the encoder is not enabled, the bit stream is by default non-return-to-zero level encoded.

4.7 Convolutional Encoder (CE)

The Convolutional Encoder (CE) implements two convolutional encoding schemes. The ESA PSS standard AD3 specifies a basic convolutional code without puncturing. This basic convolutional code is also specified in the CCSDS recommendation AD4, which in addition specifies a punctured convolutional code.

The basic convolutional code has a code rate of 1/2, a constraint length of 7, and the connection vectors $G1 = 1111001_b$ (171 octal) and $G2 = 1011011_b$ (133 octal) with symbol inversion on output path, where G1 is associated with the first symbol output.

The punctured convolutional code has a code rate of 1/2 which is punctured to 2/3, 3/4, 5/6 or 7/8, a constraint length of 7, and the connection vectors $G1 = 1111001_b$ (171 octal) and $G2 = 1011011_b$ (133 octal) without any symbol inversion. The puncturing and output sequences are listed in table 36. The encoder also supports rate 1/2 unpunctured coding with aforementioned connection vectors and no symbol inversion.

Puncturing pattern	Code rate	Output sequence
1 = *transmitted symbol* 0 = *non-transmitted symbol*		*$C_1(t)$, $C_2(t)$ denote values at bit time* t
C_1: 1 0 C_2: 1 1	2/3	$C_1(1)$ $C_2(1)$ $C_2(2)$...
C_1: 1 0 1 C_2: 1 1 0	3/4	$C_1(1)$ $C_2(1)$ $C_2(2)$ $C_1(3)$...
C_1: 1 0 1 0 1 C_2: 1 1 0 1 0	5/6	$C_1(1)$ C_2 (1) $C_2(2)$ $C_1(3)$ $C_2(4)$ $C_1(5)$...
C_1: 1 0 0 0 1 0 1 C_2: 1 1 1 1 0 1 0	7/8	$C_1(1)$ $C_2(1)$ $C_2(2)$ $C_2(3)$ $C_2(4)$ $C_1(5)$ $C_2(6)$ $C_1(7)$...

Table 36: *Puncture code sequence for convolutional code rates*

4.8 Split-Phase Level modulator (SP)

The Split-Phase Level modulator (SP) modulates a bit stream from preceding encoders according to AD5. The waveform is shown in figure 14.

Both data and the Attached Synchronisation Marker (ASM) are effected by the modulator. The modulator will increase the output bit rate with a factor of two.

4.9 Clock Divider (CD)

The Clock Divider (CD) provides clock enable signals for the telemetry and channel encoding chain in Packet Telemetry Encoder (PTME). The clock enable signals are used for controlling the bit rates of the different encoder and modulators. The source for the bit rate frequency is always the dedicated bit rate clock input ***BitClk***.

The bit rate clock input ***BitClk*** can be divided to a degree 2^n which is set at compile time. The divider can be configured during operation to divide the bit rate clock frequency from 1/1 to $1/2^n$, where n is the width of the clock divider.

The bit rate frequency can either be based on the output frequency of the last encoder in a coding chain or as the input frequency for all non-rate-increasing encoders, being selectable at compile time. No actual clock division is performed, since clock enable signals are used. No clock multiplexing is performed in the PTME.

The Clock Divider (CD) supports clock rate increases for the following encoders and rates:
- Turbo Encoder (TE), rate 1/2, 1/3, 1/4 and 1/6;
- Convolutional Encoder (CE), 1/2, 2/3, 3/4, 5/6 and 7/8;
- Split-Phase Level modulator (SP), rate 1/2.

The asynchronous reset input is synchronised towards the ***BitClk*** and ***Clk*** clock inputs and the resulting synchronised reset outputs are used for the reset of the encoders and modulators in the PTME.

It is possible to optimise the PTME to operate only on the system clock ***Clk***, which will remove any synchronisation logic otherwise required between the Clk and BitClk clock domains. The ***Clk*** and ***BitClk*** inputs should then be tied together outside the PTME.

5 MODULE DESCRIPTIONS

Each module used in the Packet Telemetry Encoder (PTME) can be used in a stand alone mode, but since the objective of this document is to describe the PTME VHDL model as a top level element, each module will not be discussed in detail. Only modules requiring further discussion other than what has been specified in preceding sections will be described hereafter.

5.1 Telemetry Encoder (TME)

Is should be noted that the Telemetry Encoder (TME) module is not a separate module in the PTME VHDL model. It is only a constellation of individual modules related to the telemetry encoding, i.e. the Virtual Channel Encoder (VCE) and the input interface modules PacketWire (PW), PacketAsynchronous (PA), PacketParallel (PP) and PacketAPB (PAPB). The description given in this document is more oriented around the functional structure rather than the implementation oriented structure of the PTME VHDL model. The discrepancies are however unlikely to cause any misunderstandings for most user since entirely internal to the PTME.

The internal Telemetry Encoder (TME) architecture is based around an proprietary bus structure, the PTME Internal Bus (PIB or PI-bus). The PIB is also shared by the Turbo Encoder (TE) in configurations when it is included in the PTME. The PIB could additionally be shared by the Reed-Solomon Encoder (RSE), but would require some adaptations of the VHDL model. The main purpose of the PIB is to allow data buffering for the different Virtual Channels to a common external buffer memory. This external buffer memory can also be used by the TE. The external buffer memory is nominally located outside the device implementing the PTME, but can be located on-chip if technology permits it. The PTME accesses the external buffer memory via the Virtual Channel Buffer (VCB) module. The VCB acts as an arbiter on the PIB and allocates the appropriate memory bandwidth between the different clients. The VCB can be connected to the external buffer memory directly via an asynchronous interface or via the AMBA Advanced High-performance Bus (AHB) master interface.

For each Virtual Channel there is a Virtual Channel Assembler (VCA) which receives data from various types of input interface and stores it in the external buffer memory via the VCB. The VCA keeps track of the number of octets received and the packet boundaries.

The Virtual Channel Multiplexer (VCM) creates telemetry Transfer Frames, taking the data and the packet boundary information from the external buffer memory as previously stored by the VCA. The VCAs inform the VCM when there is data available for transmission via the Virtual Channel Request interface (VCR) which adheres to a proprietary protocol.

There are several types of input interfaces that can be used for the Virtual Channels. The interface between such an input interface module and the corresponding VCA is named the Virtual Channel Interface (VCI) which adheres to a proprietary protocol. The interface types supported are PacketWire interface (PW), PacketAsynchronous (PA), PacketParallel (PP) and PacketAPB (PAPB). The interfaces are situated outside the VCE.

The number of Virtual Channels to be implemented is set with the ***gNumberOfVCs*** constant at compile time. If the ***gIdleFrameVC*** constant is set outside the number of implemented Virtual Channels, a separate Virtual Channel will be implemented for Idle Transfer Frames. Support for implementing 223 octet based and/or 239 octet based frame lengths is set with ***gFrameLength***.

5.2 Virtual Channel Encoder (VCE)

The Virtual Channel Encoder (VCE) is mainly a hierarchical structure in which the VCAs, the VCM and the VCB are instantiated.

The VCE implements the partitioning of the external buffer memory between the different Virtual Channel, allocating each module its share of the memory taking into account the selected Transfer Frame length etc. This is preformed when the ***gGroupInterface*** constant in the PTME_Configuration package is cleared, see section 7. The allocation process will be described in detail in section 6.3. The following parameters are generated dynamically by the VCE for each Virtual Channel:

- identifier of the memory base area allocated to a Virtual Channel
- number of Transfer Frame Data Fields allocated to a Virtual Channel
- number of octets allocated per Data Field for a given Virtual Channel.

If the ***gGroupInterface*** constant is set, then the allocation is performed outside the VCE and the PTME. This design option is however not commissioned.

5.2.1 Virtual Channel Assembler (VCA)

Each Virtual Channel Assembler (VCA) module can be configured by means of generics as described in section 4.2.1 and listed in table 58. It should be noted that the value of the ***gLength*** generic should be interpreted as an increase of the internal input buffer size (in octets) from the default size of one octet. The ***gIdle*** generic (selecting Idle Source Packet insertion) is only meaningful when the ***gPacket*** generic is selected (selecting packet handling). The ***gAbort*** generic (selecting packet insertion abort possibility) is only is only meaningful when the ***gPacket*** generic is selected.

The VCA receives data over the Virtual Channel Interface (VCI) which are transferred over the PTME Internal Bus (PIB) to the Virtual Channel Buffer (VCB) which will store them in the external buffer memory. The data are initially buffered locally in the VCA. The sizing of the internal input buffer has been mentioned above and will also be discussed later. The purpose of the internal input buffer is to allow for latency and jitter on the PIB. The VCI is described in section 6.2.

The VCA keeps track of the number of octets received and the packet boundaries in order to calculated the First Header Pointer (FHP) discussed hereafter. The data are stored in pre-allocated slots in the external buffer memory comprising Transfer Frame Data Fields. The FHP is also stored in the slot for each Transfer Frame, as part of the Frame Data Field Status (FDFS). The VCA fully supports the FHP generation and does not require any alignment of the packets with the Transfer Frame Data Field boundary. The FHP can either be generated dynamically (when the ***gPacket*** generic is selected and the ***DynamicFHP*** input port is asserted) based on the aforementioned packet boundary information, or it can be fixed to *all ones* (when the ***gPacket*** generic is not selected or the ***DynamicFHP*** input port is de-asserted). Since the generation of the FHP is not directly linked to the ***Sync*** input port, it is possible to generate a dynamic FHP even if the Sync flag is set to one in the FDFS (i.e. asynchronous non-packet data insertion).

A too large internal input buffer size could cause problems for the FHP generation if the received packets are shorter than the internal input buffer size. If a short packet is the first packet in a Transfer Frame and which also spills over into the next Transfer Frame, only the first end of packet would be taken into account and the FHP for the second Transfer Frame would not be correct. A work around is not to size the internal input buffer larger than the smallest possible packet, which is seven octets for CCSDS Source Packets but could be as small as one octet for the CCSDS Encapsulation Packet.

The insertion of the Transfer Frame Secondary Header is performed outside the VCA. The ***SecHeader*** input port is only used as a data bit for the FDFS and does not effect the sizing of the Transfer Frame Data Field in any sense. The ***Sync*** input port is used as a data bit for the FDFS and is also used as one of the qualifiers for the Idle Source Packet insertion. The ***PktOrder*** input port is only used as a data bit for the FDFS. The ***SegmentLen*** input port is only used as data bits for the FDFS and does not effect the external buffer memory availability signalling described hereafter.

The external buffer memory space allocated to each VCA is treated as a circular buffer. The VCA manages this circular buffer by means of frame and octet pointers. The write frame and octet pointer pair is used for handling the incoming data and are communicated to the VCB over the PIB (nominal data). The FDFS frame and octet pointer pair is used for writing the FDFS to the external buffer memory. The Idle Source Packet frame and octet pointer pair is used for inserting said packets in the external buffer memory. The FDFS and Idle Source Packet pointer pairs are combined in common auxiliary frame and octet pointer pairs for accesses to the VCB over the PIB (auxiliary data). A read frame pointer is used for tracking the read out of the data and is only communicated to the VCM over the Virtual Channel Request (VCR) interface.

The nominal write and read frame pointers are used for calculating whether the external buffer memory is full, whether there is space available for incoming data and whether there is at least one complete Data Field in the external buffer memory. The VCA will indicate via the VCI when it is not prepared to receive another octet (busy signalling) due to the internal input buffer being full. An additional cause is that the external buffer memory allocated to the VCA is filled.

The VCA will indicates over the VCI when there is space left in the external buffer memory for a predefined amount of octets. The threshold choice is done with the ***RdyThreshold*** input port and is defined in table 26. The implementation of the threshold can either be exact or approximated when the ***gFPGA*** constant is set in the PTME configuration package (see table 57). In the latter case the threshold is fixed to three frames in memory independently of the threshold choice, which guarantees a that there are 518 octets available independently of frame size (for a minimum frame length of 223 or 239).

All VCAs in a PTME are sized according to the same external buffer memory size, the number of memory areas and the number of areas that can be grouped together for a single VCA. Each VCA can then be allocated an individual number of frames that can be stored in memory (via the ***MaxFramePtr*** input port) and the number of octets that can be stored in one such frame (via the ***MaxOctetPtr*** input port). Note that there must be at least three frames allocated to each VCA. Note that it is not a complete Transfer Frame that is stored in memory, only the Data Field and the Frame Data Field Status field. It is possible to change the allocation during operation, provided that the module is reset after a change. It should be noted that it is the responsibility of the VCA user to ensure that the number of octets set for a frame correspond to the Transfer Frame Data Field. This value is affected by the lengths of the Transfer Frame, Secondary Header and Trailer and are calculated outside the VCA. The FDFS information is stored as two octets just after the Data Field in the associated memory slot.

As mentioned earlier, the VCA implements two PIB write ports: nominal and auxiliary. The PIB can be configured at compile time for logical pointer addressing or physical addressing. The aforementioned pointers are directly used for the logical pointer addressing option (when ***gPhysicalAddress*** is cleared), while the pointers are merged to a physical address for the latter which is not commissioned. The VCA needs to know the size of the Transfer Frame (via the ***FrameLen*** input port) in order to calculate the physical offset address. The base address for physical addressing is copied directly from the ***BaseAddress*** input port. The logical addressing is independent of the Transfer Frame length and the base pointer used is copied directly from the ***BaseAreaPtr*** input port. The PIB is described in section 6.3. The memory address generation is described in section 5.2.3. Physical addressing is a non-commissioned option.

The VCA communicates with the VCM module over the VCR interface. The VCA indicates when there is at least one frame available for transfer. The interface is described in section 6.4. The base pointer for the VCR interface is copied directly from the ***BaseAreaPtr*** input port or the ***BaseAddress*** input port when the ***gPhysicalAddress*** constant is set. There are no latency requirements between the VCA and VCM with respect to the external memory buffer, since no frame will be released by the VCA for read out from the VCM before it is completely written to the memory.

Idle Source Packets are generated by the VCA and inserted into the external buffer memory as described in section 4.2.5.3. The poll threshold is set with the ***PollThreshold*** input port for which the interpretation is listed in table 27. The most significant bit of the Version number of the Source Packet is set with the ***PktVersion*** input port. The insertion process has been designed to constrain the insertion rate to one octet every eight clock cycles.

The VCA module is in the system clock domain clocked by the ***Clk*** clock input port and is reset by the ***Reset_rise_N*** input port. The reset of the module can either be performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The data, control and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising input port clock edge. The VCA should be reset after a change of configuration port values.

5.2.2 Virtual Channel Multiplexer (VCM)

The Virtual Channel Multiplexer (VCM) creates Transfer Frames, taking the Frame Data Field Status information and the Data Field from the external buffer memory, as previously stored by the VCA. The VCM module has no generics, instead it is configured at compile time through the PTME_Configuration package described in section 7. The VCM produces the rest of the Transfer Frame Primary Header, including the Master Channel Frame Counter. The VCM holds a Virtual Channel Frame Counter for each Virtual Channel. It is incremented after the transmission of a telemetry Transfer Frame corresponding to the relevant Virtual Channel.

The VCM will produce the alternative Attached Synchronisation Marker (ASM) when the ***AltASM*** input port is asserted and the ***gAltASM*** constant is set at compile time. Otherwise only the nominal ASM will be generated.

The VCM will generate the time strobe as specified in section 4.2.4, with the periodicity being set with the ***TimeMode*** input port as described in table 24.

The Transfer Frame length is chosen with the ***FrameLen*** input port as described in table 28. This input is also used, together with the ***SecHeader***, ***IdleSecHeader***, ***OPCF*** and ***FECW*** input ports, to calculate the Data Field size for each Virtual Channel. The calculation is performed locally in the VCM. The same calculation is also performed in the VCE where the result is distributed to each individual VCA.

The Spacecraft Identifier is set with the ***SCId*** input port. The individual Virtual Channel Identifiers can be set via the ***FlexVCId*** input port in designs where flexible allocation is implemented by setting the ***gFlexVCId*** constant.

The generation of the Secondary Header is enabled individually for each Virtual Channel with the ***SecHeader*** input port.

The following applies when Idle Transfer Frames are generated on a separate Virtual Channel: the generation of a Secondary Header is enabled with the ***IdleSecHeader*** input port, the ***IdleSegmentLen*** input port is used as data bits for the FDFS, and the ***IdleFlexVCId*** input port is used as data bits for the Virtual Channel Identifier.

For active Transfer Frames, i.e. non-idle, the VCM fetches data from the external buffer memory via the PIB. Firstly the two octets corresponding to the FDFS are fetched, since being transmitted first. Secondly the Data Field is fetched. The VCM features an internal prefetch buffer which is one octet deep but can be increased by means of the ***gPreLength*** constant at compile time. The prefetching will commence as soon as the corresponding Transfer Frame is being output. The purpose of the internal prefetch buffer is to allow for latency and jitter on the PIB. The PIB can either be implemented as logical pointers or physical addressing as previously discussed in section 5.2.1 and in section 6.3. The corresponding memory area information is take from either the ***AreaPtr*** or ***BaseAddr*** input port of the VCA being selected for transfer. The frame pointer information is always taken from the ***FramePtr*** input port. Physical addressing is not commissioned.

An Idle Transfer Frame is produced on a predefined Virtual Channel when no VCA has stored a complete frame in the external buffer memory. Idle Transfer Frames are not stored in the external buffer memory but are generated while being output and their length is identical to a normal Transfer Frame. Idle Transfer Frames should not be confused with Idle Source Packets. The VCM can implement a specific Virtual Channel Frame Counter if the Idle Transfer Frames are transmitted on a separate Virtual Channel on which it is not possible to transfer user data. The design of the pseudo-random generator ensures that a Single Event Upset (SEU) or any other spurious event does not cause the generator to jump to an invalid state and incorrectly produce a static output.

The VCM produces the Transfer Frame Trailer, with the Operation Control Field (OPCF) being generated when the ***OPCF*** input port is asserted and the ***gOPCF*** constant is set, and with the Frame Error Control Word (FECW) generated when the ***FECW*** input port is asserted and the ***gFECW*** constant is set.

The VCM implements the *Telemetry Memory* interface according to AD8 when the ***gOPCFInterface*** constant is cleared. It is possible to generate the interface signals from the ***CLCWClk*** input port or from the ***BitClk*** input port when the ***gOPCFBitClock*** constant is set. If the ***gOPCFInterface*** constant is cleared, the dynamic part of the CLCW will be taken from the parallel input port ***CLCWData*** and the ***CLCWWrite*** output port will be de-asserted when this occurs. The data width of the part of the CLCW (or OPCF) to be transferred from the external input is selected by the ***gOPCFLength*** constant. When set to zero, nominal 16 bit transfer is implemented as described in section 4.2.2.7.1. When set to one, non-standard 32 bit transfer is implemented, transferring the complete CLCW (or OPCF). When set to two, both 16 and 32 bit transfers are implemented, the 32 bit transfer being enabled by setting the ***CLCWLength*** input. When the ***CLCWOverWrite*** input is asserted, bit 16 of the CLCW will be overwritten with the ***CLCWNoRFAvail*** input value, and bit 17 with the ***CLCWNoBitLock*** input value.

The VCM implements the Cyclic Redundancy Code (CRC) for the Frame Error Control Word. The CRC is calculated over all preceding fields of the Transfer Frame. The Attached Synchronisation Marker (ASM) is not included in the calculation, neither is the Frame Error Control Word field itself.

The VCM can generate blank space for insertion of Reed-Solomon check symbols by the Reed-Solomon Encoder (RSE) when the ***gReedSolomon*** constant is selected at compile time. The ***ReedSolomon*** input port enables the insertion and the ***FrameLen*** input port decides the number of blank octets to be inserted as per table 9 and table 10.

The VCM can generate four blank bits for the trellis termination process in the Turbo Encoder (TE) when the ***gTurbo*** constant is selected at compile time. The blank insertion is enabled with the ***Turbo*** input port. The VCM provides special control signals for the TE.

The VCM scans all VCR interfaces for the implemented VCAs in order to determine which Virtual Channel to output in the next Transfer Frame. The interface is described in section 6.4. The Bandwidth Allocation Table (BAT) can be accessed via an asynchronous memory type interface when the ***gBatInterface*** constant equals one. The BAT can also be located outside the VCM when the ***gBatInterface*** constant equals two, being read via the ***BatRegister*** input port with the ***BatWrite*** output port being de-asserted when this occurs. The Priority Selection scheme is enabled by means of the ***BatPriority*** input port. The depth of the BAT is set with the ***gBatDepth*** constant at compile time.

The part of the VCM module implementing the PIB and VCR interfaces is in the system clock domain clocked by the ***Clk*** clock input port and reset by the ***Reset_rise_N*** input port. The rest of the VCM module is in the bit rate clock domain clocked by the ***BitClk*** clock input port and reset by the ***Reset_bit_N*** input port. The clock enable port ***BitTick*** is used for defining the bit rate. The VCM can also include the ***CLCWClk*** clock domain, see ***gOPCFBitClock*** constant above. The ***BatCS_N*** and ***BatRW_N*** input ports also form a local clock domain, see ***gBatInterface*** constant above.

Synchronisation is performed for signals going between clock domains, provided that the ***gCommonClock*** constant is not set at compile time. The reset of the module can either be performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The data, control, clock enable and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising input port clock edge. The VCM should be reset after a change of configuration port values.

5.2.3 Virtual Channel Buffer (VCB)

The Virtual Channel Buffer (VCB) multiplexes the different VCA write accesses and the VCM read accesses on to a common external buffer memory interface, either directly or via the AMBA AHB interface. It supports read and write accesses from the TE when included in a design. The implemented mechanism uses a time slot and a sub-time slot concept.

5.2.3.1 Memory bandwidth allocation

The access bandwidth to the external buffer memory is dynamically allocated in the VCB, giving an equal amount of guaranteed bandwidth to the different modules. The internal arbiter scans requests from all modules in parallel to decide which module will be granted an access the next time slot. The arbiter follows a simple round robin scheme where each VCA is assigned one write slot, the VCM is assigned one read slot, the Turbo Encoder is assigned one read and one write slot, and finally all VCAs are assigned one common slot for auxiliary write accesses. This extra auxiliary write access slot is also arbitrated in a round robin fashion for the different VCAs, but only takes one slot in the overall time slot arbitration. Auxiliary write accesses are used for writing Frame Data Field Status information and Idle Telemetry Packet data without absorbing any of the bandwidth that is guaranteed for normal data transfers from the VCAs. The Virtual Channel dedicated to Idle Transfer Frame generation does not require any memory area or access bandwidth when implemented separately for its purpose.

Since an unused slot is assigned to the next entry in the fixed round robin table, no bandwidth is wasted. This allows sharing of the bandwidth between different modules. A module with high input data rate can use temporarily some of the overall bandwidth overhead that is yielded by a module with a lower input data rate. The latency and jitter is compensated by means of internal buffers in the modules. The arbiter can sustain full utilisation of the bus without idle periods. The analysis of bandwidth allocation on the memory or AMBA AHB bus must be done in conjunction with known system and downlink frequencies and data rates of the input sources.

Module entry	Requirement
VCA 0 write	1 access for every 8 bits received
VCA 1 write	
VCA 2 write	
VCA 3 write	
VCA 4 write	
VCA 5 write	
VCA 6 write	
VCA 7 write	
Auxiliary VCA write	overall low requirement
VCM read	1 access for every 8 telemetry bits transmitted
Turbo read	9 accesses for every 8 telemetry bits transmitted
Turbo write	1 access for every 8 telemetry bits transmitted
Test read	n/a
Test write	n/a

Table 37: *External buffer memory bandwidth allocation*

5.2.3.2 Memory area allocation

The external buffer memory size is configured at compile time by means of the ***gMemoryDepth*** constant in the PTME_Configuration package, described in section 7. It should be interpreted as a memory of two to the power of ***gMemoryDepth*** octets, each octet being eight bits wide. The memory is divided in memory areas, as defined by the ***gAreaDepth*** constant. It should be interpreted as two to the power of ***gAreaDepth*** areas. The size of such a memory area is determined at compile time and cannot be changed during operation. Each Virtual Channel can be allocated one or more memory areas.

The number of areas that can be allocated to a single Virtual Channel is defined by the ***gGroupDepth*** constant. It should be interpreted as two to the power of ***gGroupDepth*** memory areas that can be grouped together as a maximum. Each memory area is then divided in frames, each having a size depending on the length of the Transfer Frame as listed in table 28. The simplified implementation option is selected at compile time by setting the ***gFPGA*** constant.

Since the internal structure of the TME is based on pointers rather than memory addresses, the sizing of these pointers will be explained in detail here. The octet pointer is used for pointing to an octet in a frame. The frame in external buffer memory contains both the Transfer Frame Data Field and the Frame Data Field Status information. The octet pointer is always sized to eleven bits to cover all possible frame sizes. For small Transfer Frame sizes, down to only eight bits are required.

The area pointer is used for pointing out the area in which a frame is located. Due to the design of the TME, it is not necessary to point to the individual areas allocated to a Virtual Channel. It is sufficient to point to the lowest numbered area allocated to a Virtual Channel, also called the base area. The frame pointer is thus used for pointing to a frame in a group of areas and is thus normally oversized. Note that for different Transfer Frame lengths, a different number of bits are used of the frame pointer to form a memory address. Please refer to table 38 to for the pointer sizing and definition.

Area pointer	Frame pointer	Octet pointer
0 to gAreaDepth-1	0 to gMemoryDepth-8-gAreaDepth+gGroupDepth-1	0 to 10

Table 38: *Pointer sizing and definition*

The pointers are formed into a memory address either in the VCB when the ***gPhysicalAddress*** constant is cleared or in each module connected to the PIB when ***gPhysicalAddress*** is set. The principle is the same and is shown in table 39 and table 40, although the latter is an non-commissioned option. The constants have been abbreviated to *gM* for ***gMemoryDepth***, *gA* for ***gAreaDepth***, *gG* for ***gGroupDepth***, *gF* for the width of the frame pointer (equals ***gMemoryDepth***-8-***gAreaDepth***+ ***gGroupDepth***), and finally *gO* for the eleven bit octet pointer maximum width.

Transfer Frame length	Pointer		or-ed		
223 or 239	area	0 to gA-1-gG	gA-gG to gA-1		
	frame		0 to gG-1	gG to gF-1	
	octet				3 to gO-1
	memory	0 to gM -1			
446 or 478	area	0 to gA-1-gG	gA-gG to gA-1		
	frame		1 to gG-1+1	gG+1 to gF-1	
	octet				2 to gO-1
	memory	0 to gM -1			
892 or 956	area	0 to gA-1-gG	gA-gG to gA-1		
	frame		2 to gG-1+2	gG+2 to gF-1	
	octet				1 to gO-1
	memory	0 to gM -1			
1115 or 1195	area	0 to gA-1-gG	gA-gG to gA-1		
	frame		3 to gG-1+3	gG+3 to gF-1	
	octet				0 to gO-1
	memory	0 to gM -1			

Table 39: *Memory addressing for Transfer Frame lengths, without sequential EDAC*

It is possible to configure the TME to support Error Detection and Correction (EDAC) by means of the ***gEdacSupport*** constant. One of the possibilities is to locate the EDAC check bytes in the same memory as the telemetry data. This effectively reduced the available memory area for telemetry by a half. This is also reflected in the memory address generation as shown in table 40.

Transfer Frame length	Pointer	EDAC		or-ed		
223 or 239	area		0 to gA-1-gG	gA-gG to gA-1		
	frame			1 to gG-1+1	gG+1 to gF-1	
	octet					3 to gO-1
	memory	0	1 to gM -1			
446 or 478	area		0 to gA-1-gG	gA-gG to gA-1		
	frame			2 to gG-1+2	gG+2 to gF-1	
	octet					2 to gO-1
	memory	0	1 to gM -1			
892 or 956	area		0 to gA-1-gG	gA-gG to gA-1		
	frame			3 to gG-1+3	gG+3 to gF-1	
	octet					1 to gO-1
	memory	0	1 to gM -1			
1115 or 1195	area		0 to gA-1-gG	gA-gG to gA-1		
	frame			4 to gG-1+4	gG+4 to gF-1	
	octet					0 to gO-1
	memory	0	1 to gM -1			

Table 40: *Memory addressing for Transfer Frame lengths, with sequential EDAC*

Each Virtual Channel Assembler (VCA) is assigned an area identifier, a maximum number of frames and a size of the Data Field of each frame. This allocation is either performed by the VCE when the ***gGroupInterface*** constant is cleared, or it is performed from outside the PTME via the memory allocation interface although this option is not-commissioned. The former will be explained in detail hereafter.

Each Virtual Channel is assigned a fixed number of areas by means of the ***gGroupSize*** constant, which should be interpreted as two to the power of ***gGroupSize*** memory areas are being assigned. The PTME will issue a warning if the sum of areas assigned to the Virtual Channels is more or less than available in the buffer external memory. The area identifiers for the different Virtual Channels are allocated as follows. The Virtual Channel with the largest number of areas allocated will be located at the bottom of the memory space, receiving the lowest area value. The Virtual Channel with the second largest number of areas will receive the next area value, and so on till the Virtual Channel with the smallest number of allocated areas receives the largest area value. For Virtual Channels with equal amount of allocated areas, the Virtual Channel with the highest Virtual Channel Identifier value will receive the lowest area value. The Virtual Channel located in the highest area will also have its memory reduced by 4096 octets in case Turbo encoding is engaged. This is done by means of the assignment of the maximum number of frames allowed to be stored by the relevant Virtual Channel.

The formula for calculating the number of frames allowed for a Virtual Channel is shown in table 41 and is dependent on the ***gFPGA*** constant. The number of octets allowed for each frame is defined by the Data Field size and can be found in table 42. An example of how an external memory buffer is partitioned and how the Data Fields of different Virtual Channels are stored will be shown in section 5.2.3.5.

Since the turbo encoder does not fully implement the PIB addressing, this is performed inside the VCB. For the physical addressing implementation, the base address output is a direct combinatorial copy of the base address input, which is performed in the encoder. The twelve right most bits of the offset addresses are used by the encoder and the remaining bits to the left are permanently set to logical zeros. For the logical pointer implementation, the twelve bits of the turbo addresses are used by the encoder. To offset the turbo address range with respect to the other modules in the PTME, two options are possible. If the ***gGroupInterface*** constant is set in the PTME configuration package, see section 7, then the additional bits required to fill the left most bits of the memory address are taken from the PTME configuration input port ***TurboFill***. Otherwise the corresponding bits are set to *all one*s.

Transfer Frame length	**optimised**	**simplified**
223	gMemoryDepth / gAreaDepth / 224	gMemoryDepth / gAreaDepth / 256
446	gMemoryDepth / gAreaDepth / 448	gMemoryDepth / gAreaDepth / 512
896	gMemoryDepth / gAreaDepth / 896	gMemoryDepth / gAreaDepth / 1024
1115	gMemoryDepth / gAreaDepth / 1152	gMemoryDepth / gAreaDepth / 2048
239	gMemoryDepth / gAreaDepth / 240	gMemoryDepth / gAreaDepth / 256
478	gMemoryDepth / gAreaDepth / 480	gMemoryDepth / gAreaDepth / 512
956	gMemoryDepth / gAreaDepth / 960	gMemoryDepth / gAreaDepth / 1024
1195	gMemoryDepth / gAreaDepth / 1280	gMemoryDepth / gAreaDepth / 2048

Table 41: *Number of frames allocated to a Virtual Channel for each allocated area*

Transfer Frame length {octets}	Data Field length {octets]							
					Secondary Header			
			OPCF				OPCF	
		FECW		FECW		FECW		FECW
223	217	215	213	211	213	211	209	207
446	440	438	436	434	436	434	432	430
892	886	884	882	880	882	880	878	876
1115	1109	1107	1105	1103	1105	1103	1101	1099
239	233	231	229	227	229	227	225	223
478	472	470	468	466	468	466	464	462
956	950	948	946	944	946	944	942	940
1195	1189	1187	1185	1183	1185	1183	1181	1179

Table 42: *Octets in a Transfer Frame Data Field*

5.2.3.3 Memory interface

An asynchronous memory interface is implemented when the ***gMemoryInterface*** constant is selected at compile time. The interface features a single chip select signal and can only support a single external memory device for data storage. It features separate write and read strobes. The VCB implements two data buses for input and output data, together with a data enable strobe. The multiplexing of the two unidirectional buses has to be performed outside the VCB. The memory interface is based on two simple cycles, the read and the write cycle.

The read cycle only takes one clock period to perform, where chip select and read strobes are asserted simultaneously with the address output on the rising clock edge. Data are sampled on the subsequent rising clock edge. The chip select and read strobes are de-asserted on the subsequent rising clock edge. The hold time for the data read is assumed to be met due to stray capacitance since the data bus is only to be used by the VCB. This is also the reason why only a single memory device can be attached to this bus. The write cycle takes two clock cycles, where the chip select strobe is asserted simultaneously with the address and data output on the rising clock edge. The write strobe and the data enable strobe are asserted simultaneously on the subsequent falling clock edge and are again de-asserted on the subsequent falling clock edge. The write strobe is de-asserted on the subsequent rising clock edge. Also in this case is it assumed that the write hold time will be met due to stray capacitance since the data bus is only to be used by the VCB. The half period before and after the write strobe generation and data enable is sufficient to avoid data bus contention. The read and write cycles can be mixed without limitations and no idle cycles in between.

The memory interface supports wait state generation, selectable at compile time with the ***gWaitStates*** constant. The number wait states that can be inserted are selectable at compile time with the ***gWaitStateDepth*** constant. The number of wait states to be supported can be selected for read and write access with the ***WaitStateRd*** and ***WaitStateWr*** input ports, respectively. The memory interface supports Error Detection And Correction (EDAC) as described in section 5.2.3.3.1. The EDAC work on eight bit data and store eight bit check symbols. Check symbols can either read and write in parallel with the data from a separate memory device or in sequence from the same memory device. Wait state generation with EDAC is supported.

5.2.3.3.1 Error Correction and Detection (EDAC)

The EDAC implementation can either a classical Hamming code with Single Error Correction (SEC) and Double Error Detection (DED) capability or a quasi-cyclic (16, 8) code with Double Error Correction (DEC) capability, selectable by means of the ***gEdacType*** constant. Corresponding parity generation tables are shown in table 43 and table 44.

Parity		**Data**							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	not	x	x	x					
1		x	x		x				
2	not	x		x	x				
3			x	x	x				
4	not					x	x	x	
5						x	x		x
6	not					x		x	x
7							x	x	x

Table 43: *Hamming (8,4,4) parity generation*

Parity		**Data**							
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	not	x		x			x		x
1	not	x	x	x		x	x	x	
2			x	x	x	x			
3		x	x	x	x				x
4			x				x	x	x
5		x				x	x	x	x
6		x		x	x	x		x	x
7		x	x		x			x	

Table 44: *Quasi-cyclic code parity generation*

The first alternative provides a simple implementation with DED only if the errors occur in the two different nibbles of the byte. The second alternative provides a complex implementation but provides DEC for the full byte. The implementation requires a look up table for the decoding procedure. Both encoders are implement with separate encode and decode logic to facilitate back-to-back read and write accesses. Errors detected during read are signalled via the ***EdacErr*** output port as listed in.

no error	**single error**	**double error**	**multiple errors**
00_b	01_b	10_b	11_b

Table 45: *EDAC error codes*

5.2.3.4 AMBA AHB master interface

The AMBA AHB master interface has been reduced in function to support only what is required for the PTME. The following AMBA AHB features are constrained:

- does not support ***HRESP*** = *ERROR*, *SPLIT* or *RETRY*
- assumed that accesses will always be completed with ***HRESP*** = *OKAY*
- ***HSIZE***=*BYTE,* ***HTRANS***=*NONSEQ* or *IDLE,* ***HBURST***=*SINGLE,* ***HPROT***=0000_b
- never asserts ***HLOCK***
- only big-endianness supported

Since only byte accesses are performed, the byte data to be written is copied to all four byte positions on ***HWDATA*** during write accesses. The addressed byte is extracted from ***HRDATA*** for read accesses. Each access type, read and write, takes one ***HCLK*** clock period to complete, provided there are no wait states on the AMBA AHB bus. The latency and the jitter is compensated by means of input buffers in the interfacing modules. The arbiter and the AHB master can under optimal conditions sustain full utilisation of the bus without idle clock periods. The VCB can act as the default AHB master generating idle accesses when required.

The memory address described in section 5.2.3.2 is mapped on to the ***HADDR*** bus as shown in table 46. The ***HAMAX*** constant is defined in the AMBA VHDL package AD12.

AHB address	HAMAX-1 downto gMemoryDepth	gMemoryDepth-1 downto 0
memory address	*all zeros*	0 to gMemoryDepth-1

Table 46: *AMBA AHB address mapping*

5.2.3.5 Configuration example

An example of how the memory mapping is achieved for a PTME design with seven Virtual Channel and a Turbo encoder is shown in table 47 and table 41. The example design has a an overall memory of 512 kBytes, divided into 16 areas with 32678 bytes each. Each Virtual Channel can be assigned 1, 2, or 4 areas. Frame lengths based on 223 octets are implemented. The simplified addressing option has been selected. The actual placing of the Data Field in the external memory buffer is shown in table 49 for Virtual Channel 2, assuming no Secondary Header, no OPCF and no FECW. The guaranteed bandwidth allocation shown in table 50 is proportional to the system frequency, assuming that the memory interface is used.

Module	Address range	Areas	Remark
Virtual Channel 0	70000_h - $7EFFF_h$	2	reduced buffer size
Virtual Channel 1	60000_h - $6FFFF_h$	2	
Virtual Channel 2	00000_h- $1FFFF_h$	4	increased buffer size
Virtual Channel 3	50000_h- $5FFFF_h$	2	
Virtual Channel 4	40000_h- $4FFFF_h$	2	
Virtual Channel 5	30000_h- $3FFFF_h$	2	
Virtual Channel 6	20000_h- $2FFFF_h$	2	
Turbo Encoder	$7EFFF_h$- $7FFFF_h$	n/a	Turbo Encoder memory

Table 47: *Example of memory mapping*

Module	Areas	Frame size				Remark
		223	446	892	1115	
		Number of frames in memory				
Virtual Channel 0	2	240	120	60	30	reduced due to turbo encoder
Virtual Channel 1	2	256	128	64	32	
Virtual Channel 2	4	512	256	128	64	increased size
Virtual Channel 3	2	256	128	64	32	
Virtual Channel 4	2	256	128	64	32	
Virtual Channel 5	2	256	128	64	32	
Virtual Channel 6	2	256	128	64	32	

Table 48: *Example of stored Transfer Frames in external buffer memory*

Transfer Frame length							
223		446		892		1115	
Data	Address	Data	Address	Data	Address	Data	Address
Data Field 0	00000:000D8	Data Field 0	00000:001B7	Data Field 0	00000:00375	Data Field 0	00000:00454
FSFS 0	000DA:000DB	FSFS 0	001B8:001B9	FSFS 0	00376:00377	FSFS 0	00456:00457
Data Field 1	00100:001D8	Data Field 1	00200:003B7	Data Field 1	00400:00775	Data Field 1	00800:00C54
FSFS 1	001DA:001DB	FSFS 1	003B8:003B9	FSFS 1	00776:00777	FSFS 1	00C56:00C57
...	...	...	...	...	...	...	...
Data Field 511	1FF00:1FFD8	Data Field 255	1FE00:1FFB7	Data Field 127	1FC00:1FC75	Data Field 63	1F800:1FC54
FSFS 511	1FFDA:1FFDB	FSFS 255	1FFB8:1FFB9	FSFS 127	1FC76:1FC77	FSFS 63	1FC56:1FC57

Table 49: *Example of memory usage for Virtual Channel 2 (hexadecimal addresses)*

Unit	Requirement	Clocks	Guaranteed bandwidth
VCA 0 write	1 access for every 8 bits received	2 clock periods	0,4 bit per Hz
VCA 1 write			
VCA 2 write			
VCA 3 write			
VCA 4 write			
VCA 5 write			
VCA 6 write			
auxiliary VCA accesses	overall low requirement	2 clock periods	
VCM read	1 access for every 8 telemetry bits transmitted	1 clock period	
Turbo read	9 accesses for every 8 telemetry bits transmitted	1 clock period	
Turbo write	1 access for every 8 telemetry bits transmitted	2 clock periods	

Table 50: *Example of external buffer memory bandwidth allocation (fixed)*

5.3 Reed-Solomon Encoder (RSE)

The Reed-Solomon Encoder (RSE) encodes a serial bit stream from preceding encoders according to AD4 and the resulting symbol stream is output bit serially. The encoder generates codeblocks by receiving information symbols from the preceding encoders which are transmitted unmodified while calculating the corresponding check symbols which in turn are transmitted after the information symbols. The check symbol calculation is disabled during reception and transmission of unmodified data not related to the encoding. The calculation is independent of any previous codeblock and is perform correctly on the reception of the first information symbol after a reset. A functional diagram of the encoder is shown in figure 7.

Each information symbol is received serially and assembled into an 8 bit symbol. The symbol is fed to a binary network in which parallel multiplication with the coefficients of a generator polynomial is performed. The products are added to the values contained in the check symbol memory and the sum is then fed back to the check symbol memory while shifted one step. This addition is performed octet wise requiring eight clock cycles per input symbol. The most significant check symbol is implemented in a shift register where it is serially added to the next information symbol being received. This cycle is repeated until all information symbols have been received. The contents of the check symbol memory are then serially output from the encoder. The encoder is based on a bit serial architecture, except for the parallel multiplier.

The encoder can be configured at compile time to support only the E=16 (255, 223) code, only the E=8 (255, 239) code, or both. This is done with the ***gRate*** generic. Only the selected coding schemes are implemented. The choice between the two is performed during operation with the ***Rate*** input port, where a logical zero corresponds to E=16 and a logical one corresponds to E=8.

The maximum number of supported interleave depths I_{max} is selected at compile time with the ***IDepth*** generic, the range being 1 to 8. For a specific instantiation of the encoder, the choice of any interleave depth ranging from 1 to the chosen I_{max} is supported during operation. It is possible to skip interleave depth 3 support by means of the ***SkipIDepth3*** generic, which is useful for the PTME implementation. The area of the encoder is minimised, i.e. logic required for a greater interleave depth than I_{max} is not unnecessarily included. It is also possible to support uncontiguous data structures, i.e. space between the end of the check symbols and the next ASM, by means of the ***gUncontiguous*** generic (normally not used in the PTME).

The interleave depth is chosen during operation by means of the ***Interleave(0:2)*** input port, as listed in table 51.

The encoder also implements non-return-to-zero level encoding and pseudo-randomisation controlled by means of the ***gMark*** and ***gPseudo*** generics, respectively. This is not used in the PTME where there are separate modules for such encoding to provide a flexible coding chain. The are four compile time options for the implementation of the check symbol memory in the encoder, selected by the ***Style*** generic. The first option is based on flip-flops. The second on latches. The third option provides an interface to an external 32 bit wide memory (not used in the PTME). The fourth option is an equal split between flip-flops and latches.

InterLeave(0:2)	Interleave depth	E=8, RS(255, 239)		E=16, RS(255, 223)	
		information	**check**	**information**	**check**
000b	1	239	16	223	32
001b	2	478	32	446	64
010b	3	717	48	669	96
011b	4	956	64	892	128
100b	5	1195	80	1115	160
101b	6	1434	96	1338	192
110b	7	1673	112	1561	224
111b	8	1912	128	1784	256

Table 51: *Interleave depth alternatives for Reed-Solomon Encoder (RSE)*

One clock enable port is used for defining the bit rate. Two control signals, attached synchronisation marker and transfer frame delimiters, are clocked through the encoder to keep them in sync with the encoded data bit stream. The transfer frame delimiter is used for determining the input information symbols. The attached synchronisation marker delimiter is used for bypassing and disabling the encoder.

The reset of the encoder can either be performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The data, control, clock enable and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising clock edge of the input clock port. The encoder should be reset after a change of configuration port values.

The internal architecture of the RSE is shown in figure 8 and each sub-module is described in the subsequent sections.

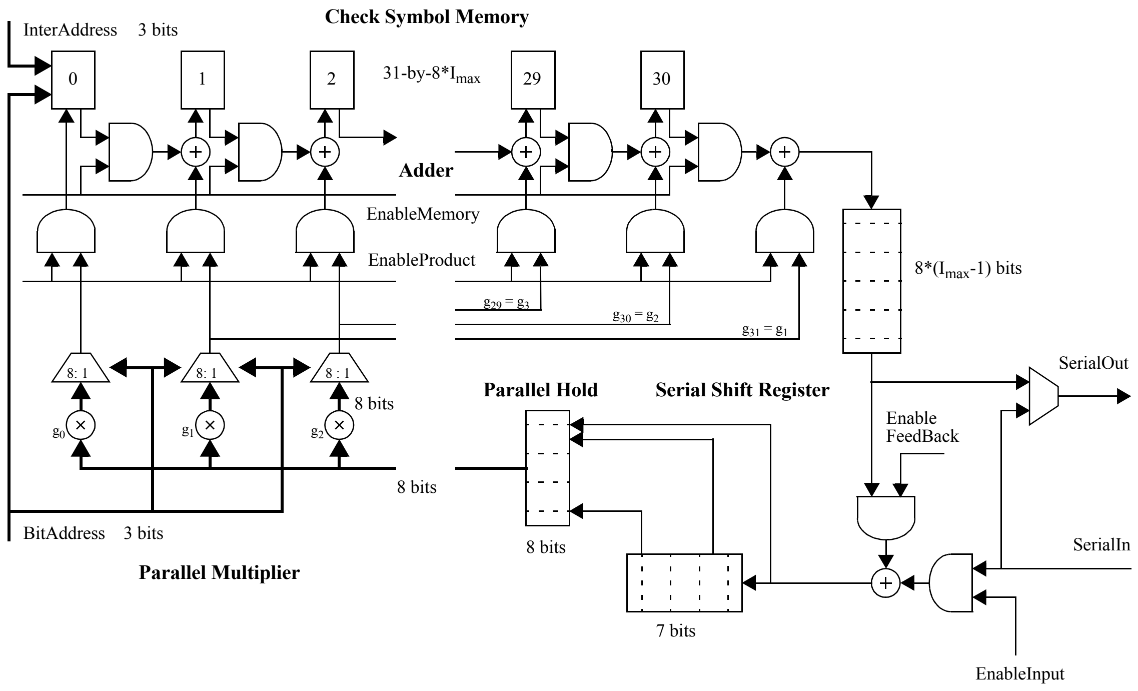

Figure 7: *Functional diagram of Reed-Solomon Encoder (RSE)*

5.3.1 Adder

The adder performs a bit wise exclusive or operation between the outputs of the check symbol memory and the multiplier. The operators stemming from the memory can be suppressed to logical zero by means of a control signal. The operators stemming from the multiplier can be suppressed to logical zero by means of a second control signal. Data from the memory or the multiplier must only be enabled when containing data related to the ongoing codeword.

5.3.2 Check symbol memory

The check symbol memory holds I_{max}*8-by-31 bits. It is not reset, instead the feedback from it can be suppressed in the adder when zero data is expected, as during the reception of the first information symbol of each interleave depth.

5.3.3 Parallel multiplier

The multiplier performs parallel multiplication directly in the dual basis representation. The individual bits of the 32 symbols produced are selected by means of a bit address provided by the control block.

5.3.4 Serial shift and parallel hold registers

The serial shift register contains the most significant check symbol, performing the addition between it and the incoming information symbols on the input. The resulting symbol is fed back to the multiplier by means of a parallel holding register. The addition only takes operands related to the ongoing codeword generation, else the operand values are suppressed to logical zero by means of control signals. The data input is suppressed during read out of check symbols.

5.3.5 Control

The control logic implements all required control signals and two address buses. It also performs the selection of source for the data output: the input data stream or the check symbol memory contents.

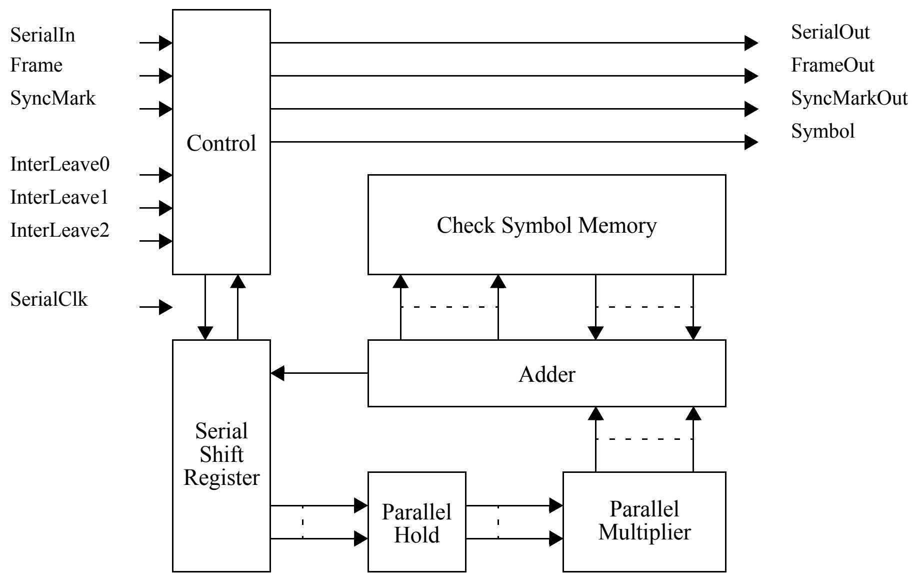

Figure 8: *Block diagram of the bit serial Reed-Solomon Encoder (RSE)*

5.4 Turbo Encoder (TE)

The Turbo Encoder (TE) encodes a serial bit stream from preceding encoders according to AD4 and the resulting symbol stream is output bit serially. The input data corresponding to the telemetry transfer frame are encoded. The encoder replaces the nominal Attached Synchronisation Marker (ASM) with a specific ASM defined for each coding rate as specified in AD4 and repeated in table 14. It is possible to disable the encoder with the ***Turbo*** input port, but the uncoded bit stream should be passed by the encoder. Only the switching of the internal control signal is disabled.

One clock enable port is used for defining the input bit rate and another clock enable port is used for defining the output bit rate. The encoder should increase the output bit rate with a factor depending on the coding rate. It is the responsibility of the user to ensure that the correct ratio is maintained between the two frequencies of the input and output clock enable ports. The encoding is done in the bit rate domain clocked by the ***BitClk*** clock input port. Two control signals, attached synchronisation marker and transfer frame delimiters, are clocked through the encoder to keep them in sync with the encoded data bit stream. Three additional input ports are used for the control of the encoder.

The encoder supports the recommended information block lengths specified in AD4 and listed in table 11. The selection between the information block lengths is performed with the ***FrameLen(0:2)*** input port as listed in table 52. The encoder does not support the information block length of 16384 bits that is mentioned as under investigation in AD4.

FrameLength(0:2)	000b	223 octets	supported
	001b	446 octets	
	010b	892 octets	
	011b	1115 octets	
	100b	239 octets	unsupported
	101b	478 octets	
	110b	956 octets	
	111b	1195 octets	

Table 52: *Information block length selection for Turbo Encoder (TE)*

The encoder supports the code rates specified in AD4 and listed in table 12. The selection between the code rates is performed with the ***TurboRate(0:1)*** input port as listed in table 53.

TurboRate(0:1)	00_b	rate 1/2
	01_b	rate 1/3
	10_b	rate 1/4
	11_b	rate 1/6

Table 53: *Code rate selection for Turbo Encoder (TE)*

The encoder uses an external buffer memory to temporarily store the information block during the encoding procedure. The external buffer memory is organised in two areas, the first is used for the code information block that is being received and the second is used for the information block that is being coded and transmitted. The interpretation of the two memory areas is shifted between frames to avoid data copying.

The memory interfacing is performed over the PTME Internal Bus (PIB), see section 6.3 for details. The encoder both writes and reads data over this bus. The word format is always eight bits and 4096 words are required, 2048 for each of the two areas. The PIB is implemented in the system clock domain clocked by the ***Clk*** input port.

The PIB interface supports two implementation possibilities, either based on logical pointers or on physical addresses. This is selected at compile time by the ***gPhysicalAddress*** constant in the PTME configuration package, see section 7. The difference for the TE implementation is not that significant. The twelve address bits used by the encoder are byte oriented, with the left most bit being used to distinguish between the two memory areas used for the received and transmitted information block. Physical addressing is not commissioned. Note that the TE does not implement the PIB interface addressing in detail, it only provides a twelve bit address. The base address and the base area pointers are created in the Virtual Channel Buffer (VCB), see section 5.2.3.

A low latency option can be selected at compile time by the ***gTurboLatency*** constant the PTME configuration package, see section 7. The implementation of the encoder will then include two separate read and write buffers, configurable with ***gTurboLengthRd*** and ***gTurboLengthWr***.

No output is generated by the encoder while the first information block is received, and the corresponding frame and attached synchronisation marker delimiters are not asserted during this time period. Each information block will be delayed by the time corresponding to the transmission of one information block. This will cause a corresponding static timing offset for the time strobe generated by the Telemetry Encoder (TME).

The reset of the encoder can either be performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The data, control, clock enable and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising clock edge of the input clock ports. The encoder should be reset after a change of configuration port values.

The internal architecture of the Turbo Encoder (TE) is shown in figure 9 and each sub-module is described in the subsequent sections.

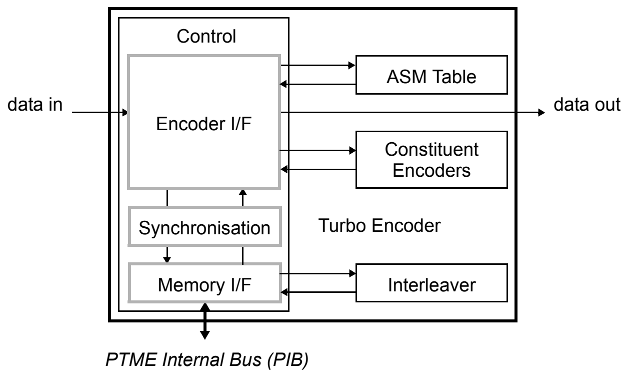

Figure 9: *Turbo Encoder (TE) block diagram*

5.4.1 Interleaver

The interleaver (or permuter) implements the address generation that is required to fetch the correct bit from the buffered information block. The address generation unique for each information block length. The generation is controlled by the controller, it can be reset or stepped forward by means of control signals. A sequential address is produced together with a pseudo-random address on the output ports of the Interleaver. The addresses are 14 bits wide to cover the maximum information block length in bits.

The interleaver is not designed straight after the algorithmic specification in AD4 since inefficient to implement on silicon. The corresponding pseudo-random address sequence is preformed by means of simpler operators and counters. The two sequences are however identical. The interleaver module is in the system clock domain, ***Clk***.

5.4.2 Constituent encoders

The constituent encoders (or component codes) in the TE are simple recursive convolutional encoders shown in figure 6 and are implemented as a single module in the TE. The constituent encoders are controlled by the controller. It is possible to reset the encoders, to shift the encoders, to enable the feed back path of the encoders, and to step through the output multiplexing sequence.

The encoders take two bit serial streams as input, one sequential and one pseudo-random stream. A single multiplexed serial output stream is produced, with the appropriate code rate increase. The constituent encoder module is in the bit rate clock domain, ***BitClk***.

5.4.3 Table

The Attached Synchronisation Marker (ASM) differs for the various code rates as specified in AD4. The Table implements the different ASMs as a combinatorial look up table that has been coded to optimise the implementation for area. The similarities between the four ASMs have been utilised in the optimisation of the design. The data is addressed with the coding rate choice input and an address input pointing to the addressed bit in the ASM.

5.4.4 Control

The Control module of the TE implements the overall control of the encoder and the PIB interface. The module is divided in two clock domains, one for the PIB interface which also interfaces the Interleaver, and one for the bit encoding which interfaces constituent encoders and the attached synchronisation marker table. Synchronisation between the ***BitClk*** and ***Clk*** clock domains is performed in the Control module if the ***gCommonClock*** constant is not set in the PTME configuration package, see section 7.

5.5 Pseudo-Randomiser (PSR)

The Pseudo-Randomiser (PSR) generates a bit sequence according to AD3 and AD4 which is xor-ed with the incoming serial bit stream from preceding encoders. The resulting bit stream is output serially. It is not possible to disable the encoder, the uncoded bit stream should instead be passed by the encoder.

One clock enable port is used for defining the bit rate. Two control signals, attached synchronisation marker and transfer frame delimiters, are clocked through the encoder to keep them in sync with the encoded data bit stream. The attached synchronisation marker delimiter is also used for defining what data are not to be encoded, initialising the encoder LFSR to *all ones* while being asserted.

The reset of the encoder can be either performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The the data, control, clock enable and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising clock edge of the input clock port.

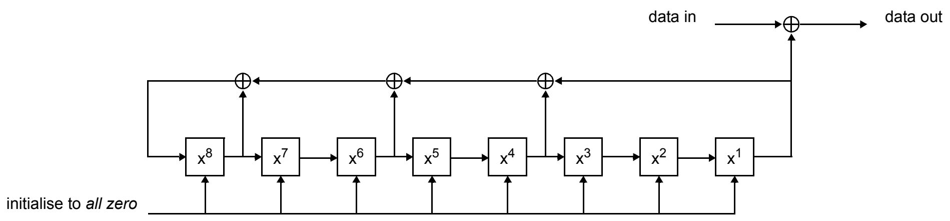

Figure 10: *Pseudo-Randomiser (PSR) functional block diagram*

5.6 Non-Return-to-Zero Mark encoder (NRZ)

The Non-Return-to-Zero Mark encoder (NRZ) encodes differentially a bit stream from preceding encoders according to AD5 and the resulting bit stream is output serially. All input data are encoded. It is not possible to disable the encoder, the uncoded bit stream should instead be passed by the encoder, which is by default non-return-to-zero level encoded. The output waveform is shown in figure 11.

One clock enable port is used for defining the bit rate. Two control signals, attached synchronisation marker and transfer frame delimiters, are clocked through the encoder to keep them in sync with the encoded data bit stream. The control signals are not used for any other purpose inside the encoder.

The reset of the encoder can be either performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The the data, control, clock enable and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising clock edge of the input clock port.

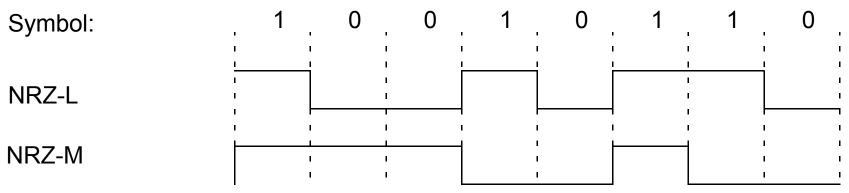

Figure 11: *Non-Return-to-Zero - Level and Non-Return-to-Zero - Mark waveforms*

5.7 Convolutional Encoder (CE)

The Convolutional Encoder (CE) encodes a serial bit stream from preceding encoders according to AD4 and the resulting symbol stream is output both bit serially and two-bit in parallel. All input data are encoded. It is not possible to disable the encoder, the uncoded bit stream should instead be passed by the encoder.

One clock enable port is used for defining the input bit rate and another clock enable port is used for defining the output bit rate. The encoder should increase the output bit rate with a factor depending on the coding rate. It is the responsibility of the user to ensure that the correct ratio is maintained between the two frequencies of the input and output clock enable ports.

The results of the two connection vectors are multiplexed on the bit serial output port as per AD4. For the rate 1/2 coding, the results of the two connection vectors G1 and G2 are also output in parallel for each input bit on the corresponding output ports, with symbol inversion on G2 when specified. The latter can be used to implement an external multiplexer and puncture logic, without the need for an code rate increase in the encoder, i.e. the input and output clock enable ports are tied together outside the encoder. This is supported in the PTME.

The encoder can be configured to support only basic unpunctured encoding, see figure 12, only punctured encoding, see figure 13, or both coding schemes. This is done with the ***gConvolute*** generic. Only the selected coding schemes are implemented. The different code rates are chosen with the ***Rate(0:2)*** input port, as shown in table 54. Note that the rate 1/2 code without symbol inversion is not specified in AD4. Only six bit registers are implemented for the constraint length seven code, since the first input delay is not necessary to generate the specified convolutional code correctly.

Rate(0:2)	00-_b	rate 1/2, with symbol inversion
	01-_b	rate 1/2, no symbol inversion
	100_b	rate 2/3, punctured
	101_b	rate 3/4, punctured
	110_b	rate 5/6, punctured
	111_b	rate 7/8, punctured

Table 54: *Code rate alternatives for Convolutional Encoder (CE)*

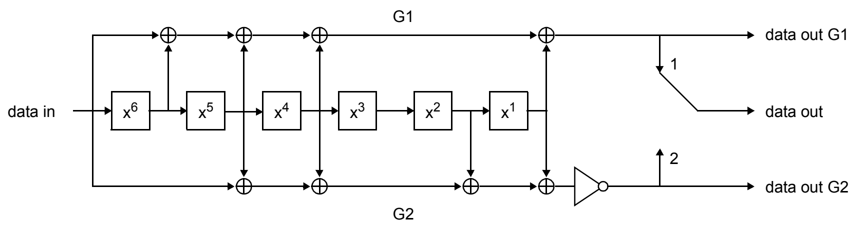

Figure 12: *Basic convolutional encoder functional block diagram*

The reset of the encoder can be either performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The the data, clock enable and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising clock edge of the input clock port. The encoder should be reset after a change of configuration port values.

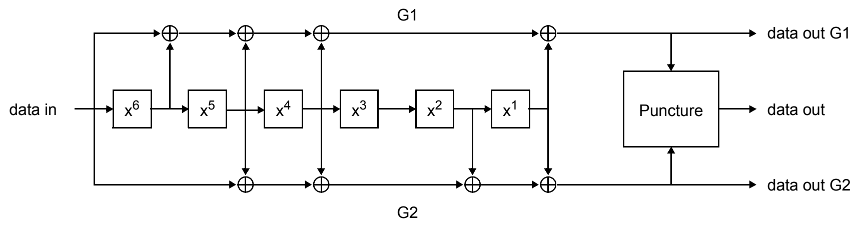

Figure 13: *Punctured convolutional encoder functional block diagram*

5.8 Split-Phase Level modulator (SP)

The Split-Phase Level modulator (SP) modulates a serial bit stream from preceding encoders according to AD5 and the resulting bit stream is output serially. All input data are modulated. It is not possible to disable the modulator, the unmodulated bit stream should instead be passed by the encoder. The output waveform is shown in figure 14.

One clock enable port is used for defining the input bit rate and another clock enable port is used for defining the output bit rate. The modulator should increase the output bit rate with a factor of two. It is the responsibility of the user to ensure that this ratio is maintained between the two frequencies of the input and output clock enable ports. The reset of the modulator can be either performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. The the data, clock enable and reset input ports are assumed to be synchronised with respect to the input clock port. All clocking is performed on the rising clock edge of the input clock port.

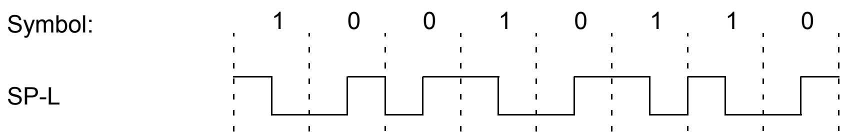

Figure 14: *Split-Phase - Level waveform*

5.9 Clock Divider (CD)

Packet Telemetry Encoder (PTME) has a clocking scheme that is partitioned in multiple clock domains. The two main domains are the system clock domain and the bit clock domain. The system clock domain is predominantly associated with all functions that communicate with the external buffer memory. The bit clock domain is associated with all function directly associated with the telemetry and channel coding chain. There are several encoders that are part of both the system clock domain and the bit clock domain. There are also several local clock domains associated with the different input interfaces, which are kept local to each input. Every such input interface is also part of the system clock domain.

Since all telemetry and channel coding is performed in the bit clock domain, using the same PTME input clock, it is necessary to generate clock enable signals to cope with bit rate differences between the channel encoders. Each encoder has therefore at least one clock enable input. Those encoders or modulators that increase the bit rate have two clock enable inputs, one for the input bit rate and one for the output bit rate. The clock enable inputs are also used for regulating the overall bit rate as will be described later.

The ***BitClk*** clock input is used for the bit clock domain. This is an input separated from the ***Clk*** clock input used for the system clock domain. The PTME can be configured either to operate with two separate and independent clock frequencies for the system clock domain and the bit clock domain, or to assume that the two clock inputs are always tied to the same external clock source without any clock skew difference, using the ***gCommonClock*** parameter, see section 7. This is configured at compile time. The former option allows the two clock inputs to be either generated from independent sources or to be tied together, since synchronisation logic between the two clock domains will be permanently generated at compile time. The latter option will not produce any synchronisation logic and the two inputs will be considered as a common clock.

The bit clock input ***BitClk*** can be used either for defining the input clock rate of the encoders, or for defining the output clock rate of the encoders. This is configured at compile time using the ***gClockStyle*** parameter, see section 7. The former option is only useful for encoders that do not increase the bit rate on their outputs. This includes the Telemetry Encoder (TME), the Reed-Solomon Encoder (RSE), the Pseudo-Randomiser (PSR) and the Non-Return-to-Zero Mark encoder (NRZ). It also includes the Convolutional Encoder (CE) when used with the basic convolutional code which features two dedicated unmultiplexed outputs that need to be multiplexed outside the PTME. The latter option can be used with all encoders, additionally including the Convolutional Encoder (CE) with both basic and punctured convolutional coding, the Turbo Encoder (TE) and the Split-Phase Level modulator (SP). Note that the output clock rate is that of the output of the last encoder or modulator in a coding chain.

The main task of the Clock Divider (CD) is to generate the different clock enable signals that are used by the different encoders and modulators. To support an arbitrary choice of encoders and modulators in an encoding scheme, the CD generates four different enable signals in the PTME. The first one is used for the basic telemetry bit rate and is used by the TME and the RSE. The second one is used by the CE for the output frequency and can have one of the following ratios with respect to the first enable signal frequency 2, 3/2, 4/3, 6/5 and 8/7. The third one is used by the TE for the output frequency and can have one of the following ratios with respect to the first enable signal frequency: 2, 3/2, 4/3, 6/5 and 8/7. The fourth one is only used by the SP for the output frequency and is twice the frequency of the modulator input. The PSR and

NRZ can either use the third one or the first one, depending on whether TE is engaged or not. The enable signal used for the SP input frequency can be the second one if CE is engaged, or the third one if TE is engaged, or else the first one is used. Support is only provided in a given configuration for those encoders that are implemented in the PTME, see section 7.

The CD is also able to divide the bit rate clock frequency by means of the different clock enable signals. The clock divider can be sized at compile time to accommodate different degrees of division. The divider can be configured during operation to divide the bit rate clock frequency from 1/1 to $1/2^n$, where n is the width of the clock divider, using the ***gClockDepth*** parameter, see section 7. An example of clock division is shown in table 55, where the clock rate selection is made with the ***OutputBitRate*** input. The resulting frequency is used as the output bit rate of the last encoder or modulator in the encoding chain. Note that the bit rate clock is not divided as such, only the different clock enable signal frequencies are divided.

OutputBitRate(0:7)	00_h	1/1
	01_h	1/2
	...	...
	FE_h	1/255
	FF_h	1/256

Table 55: *Example of bit rate clock division for an* n=*8 bit wide divider*

The bit rate relationship between the encoders and modulators in the PTME is shown in table 56. The output symbol rate, f_o, can be obtained by dividing the ***BitClk*** input, actually generating clock enable pulses. Since bit rate generation is based on the output symbol rate, f_o, of the last encoder or modulator used in the encoding chain, the telemetry bit rate, f_{tme}, can vary for the same f_o frequency depending on the used encoding scheme as shown in table 56.

There are two ways of implementing a designing that has to operate on different frequencies: either to use a common clock and to distribute separate enable signals; or to use separate clock signals with different frequencies derived from a common clock. The former approach is easier to manage since only a single clock is required in the design but it can limit the overall performance of the system. It is also more power consuming than the latter approach.

The latter approach will give a faster implementation if the slowest part of the design also happens to be the one with the critical timing path. However, the latter approach requires accurate clock tree design and balancing in order to work properly. The PTME VHDL model has been based on the former approach to simplify its use, especially in Field Programmable Gate Array (FPGA) devices.

	TME															
					RSE				TE							
			CE				CE		1/2		1/3		1/4		1/6	
		SP		SP		SP		SP		SP		SP		SP		SP
TME, f_{tme}	f_o	$f_o/2$	$f_o/2$	$f_o/4$	f_o	$f_o/2$	$f_o/2$	$f_o/4$	$f_o/2$	$f_o/4$	$f_o/3$	$f_o/6$	$f_o/4$	$f_o/8$	$f_o/6$	$f_o/12$
RSE, f_{rse}	-	-	-	-	f_o		$f_o/2$	$f_o/4$	-	-	-	-	-	-	-	-
TE, f_{te}	-	-	-	-	-	-	-	-	f_o	$f_o/2$	f_o	$f_o/2$	f_o	$f_o/2$	f_o	$f_o/2$
PSR, f_{te}	f_o	$f_o/2$	$f_o/2$	$f_o/4$	f_o	$f_o/2$	$f_o/2$	$f_o/4$	f_o	$f_o/2$	f_o	$f_o/2$	f_o	$f_o/2$	f_o	$f_o/2$
NRZ, f_{nrz}	f_o	$f_o/2$	$f_o/2$	$f_o/4$	f_o	$f_o/2$	$f_o/2$	$f_o/4$	f_o	$f_o/2$	f_o	$f_o/2$	f_o	$f_o/2$	f_o	$f_o/2$
CE, f_{ce}	-	-	f_o	$f_o/2$	-	-	f_o	$f_o/2$	-	-	-	-	-	-	-	-
SP, f_{sp}	-	f_o	-	f_o	-	f_o	-	f_o	-	f_o	-	f_o	-	f_o	-	f_o

Table 56: *Bit rates for different encoders based on an obtained output frequency*

As it can be seen from table 56, there will be multicycle paths in some of the modules, due to the use of clock enable signals, when not operated at the maximum possible frequency. It is rather simple to define multicycle paths for a module since each module has one or two clock enable signals. By analysing the clock enable periodicity with respect to the clock frequency one can set a single multicycle path constraint for each module. However, the multicycle paths will only be valid for some of the possible encoder and modulator combinations. For example, if the TME is operated at the maximum bit rate there will not be any multicycle paths in that module as a result of the code rate differences with respect to other encoders such as the CE, and there will be no multicycle paths due to output bit rate regulation as done by the CD. As a conclusion, the multicycle paths can only be optimised for in implementations where it is known at compile time that a particular encoder or modulator will not be operated at the highest possible bit rate.

For advanced user there is always the possibility to modify the PTME hierarchy and build a new clocking scheme based on separate clock signals for the different encoders and modulators. This approach will most certainly be required for ASIC designs in order to produce an optimal result.

When an AMBA AHB interface is implemented in the Telemetry Encoder (TME), the clock domain of this interface will be the same as the system clock domain and will use the ***Clk*** clock input. There is thus no dedicated AMBA AHB clock and reset interface in the PTME.

The CD synchronises the external asynchronous reset input towards the system clock domain and the bit clock domain. The AMBA APB input interfaces are reset by a separate reset signal that is assumed to be synchronous with corresponding interface clock. The CD should be reset whenever a configuration input value is changed. The reset of the two clock enable generators and clock divider can be either performed synchronously or asynchronously, depending on the ***gSyncReset*** constant in the PTME configuration package, see section 7. All clocking is performed on the rising clock edge of the clock ports.

The PTME supports a any clock frequency ration between ***BitClk*** and ***Clk***, provided that the BitClk frequency is <u>never</u> larger than twice the Clk frequency.

5.10 Packet Telemetry Encoder (PTME)

The Packet Telemetry Encoder (PTME) module instantiates all encoder and modulator modules previously described. The PTME module propagates all interface signals to the relevant embedded modules, with few signal modifications, allowing a high degree of control and observability of the embedded encoders and modulators. The PTME module combines the internal unidirectional data buses of the memory interface to a single bidirectional data bus interface. This is also done for the Bandwidth Allocation Table (BAT) interface data bus.

5.10.1 Connectivity

The PTME module provides the interconnection between the different encoders and supports all permissible coding chains. The symbol output of the last encoder in the coding chain is routed to the output interface of the PTME for which a re-synchronisation of the signal is performed to avoid glitch generation. It facilitates an external telemetry test interface, although not commissioned, to allow telemetry insertion to the Reed-Solomon Encoder (RSE) and Turbo Encoder (TE) while bypassing the Telemetry Encoder (TME). The PTME module distributes the four clock enable signals from the Clock Divider (CD) to the encoders and modulators to obtain the desired bit rate and the correct bit rate increase. It implements the connectivity between the enclosed encoders and modulators as described hereafter.

The output from the Telemetry Encoder (TME) can be connected to:
- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encoder
- Convolutional encoder
- Split-Phase Level modulator

The input to the Reed-Solomon Encoder (TME) can be connected to:
- Telemetry encoder

The output from the Reed-Solomon Encoder (TME) can be connected to:
- Pseudo-Randomiser
- Non-Return-to-Zero Mark modulator
- Convolutional encoder
- Split-Phase Level modulator

The input to the Turbo Encoder (TE) can be connected to:
- Telemetry encoder

The output from the Turbo Encoder (TE) can be connected to:
- Pseud-Randomiser
- Split-Phase Level modulator

The input to the Pseudo-Randomiser (PSR) can be connected to:
- Telemetry encoder
- Reed-Solomon encoder

The output from the Pseud-Randomiser (PSR) can be connected to:
- Non-Return-to-Zero Mark modulator
- Convolutional encoder
- Split-Phase Level modulator

The input to the Non-Return-to-Zero Mark encoder (NRZ) can be connected to:
- Telemetry encoder
- Reed-Solomon encoder
- Pseudo-Randomiser

The output from the Non-Return-to-Zero Mark encoder (NRZ) can be connected to:
- Convolutional encoder
- Split-Phase Level modulator

The input to the Convolutional Encoder (CE) can be connected to:
- Telemetry encoder
- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encoder

The output from the Convolutional Encoder (CE) can be connected to:
- Split-Phase Level modulator

The input to the Split-Phase Level modulator (SP) can be connected to:
- Telemetry encoder
- Reed-Solomon encoder
- Pseudo-Randomiser
- Non-Return-to-Zero Mark encoder

Due to the flexibility of the coding chain, it is possible to enable illegal encoder combinations. It is the responsibility of the user not to enable the illegal combinations as listed hereafter:
- TE and CE
- TE and NRZ
- TE and RSE

5.10.2 Limitations

The PTME limits to some extent the capabilities of the encoders and modulators included in the module, which can be understood from the preceding module descriptions. The implemented clocking scheme in the PTME also limits the overall telemetry bit rate performance of the design since all encoder and modulators are clocked by the same bit rate clock. In the worst case the telemetry encoder needs to be clocked up to twelve times faster than required for the telemetry rate in order to be compatible with the turbo encoder and the split-phase modulator. This limitation can be overcome if the TME, the TE and SP are all clocked on different clocks derived from the same source maintaining clock edge synchronisation, which is however not implemented in the PTME. This issue is discussed in detail in section 5.9.

6 MODULE INTERFACES

The Packet Telemetry Encoder (PTME) is based on internal interface structures for which the use outside the PTME is not considered. The interfaces are therefore described only briefly.

6.1 Multiple input interfaces

The PTME module implements four different type of input interfaces for packet and data transfer to the Virtual Channels via the Virtual Channel Interface (VCI) defined in section 6.2. Since it is possible to select any type interface for up to eight possible Virtual Channels, each interface type has therefore been copied up to eight times in the PTME port declaration. It should be noted that only some of these interfaces will be used, which is configured at compile time as per section 7.

6.2 Virtual Channel Interface (VCI) definition

The Virtual Channel Interface (VCI) resides between each Virtual Channel Assembler (VCA) and its input interface modules. The actual user input interfaces are placed outside the VCA since there can be several different types of interfaces and protocols to be implemented. The VCI is designed to provide all the information required by the VCA for proper data insertion into Transfer Frames. The VCI is synchronous and independent of the preceding input interface implementation.

The VCI provides the VCA with the following information:
- Packet start detected
- Packet in progress
- Packet end detected
- Packet to be held
- Packet to be aborted
- Data word available
- Data word (octet)

The VCI provides the input interface module with the following information:
- Interface is busy processing the input data
- Space available in external buffer memory for a packet (or segment)

The VCI is only capable of processing one data word at a time. There is no other buffering capability on the VCI. Buffering can either be made in the input interface module or in the VCA internal input buffer.

6.3 PTME Internal Bus (PIB or PI-bus) definition

The communication between Virtual Channel Buffer (VCB) and the different clients: Virtual Channel Assembler (VCA), Virtual Channel Multiplexer (VCM) and Turbo Encoder (TE); is done over the PTME Internal Bus (PIB or PI-bus).

Each client requests an access to the external buffer memory by asserting a request signal to the VCB. The VCB will acknowledge the request by a grant signal after which the request from the client can be released. The request input from the client just granted access will be masked in the next clock period to allow the client to remove its request not to interpret a single request as two. At the moment the client asserts its request signal it also drives the corresponding frame and octet pointers to inform the VCB of where in the external buffer memory the relevant data resides, and whether it is a read or a write access. Note that the VCA can only make write accesses and the VCM can only make read accesses. These frame and octet pointer values are allowed to change after the request has been granted by the VCB, allowing the client to generate a new request immediately after the previous one has been granted. The VCB will assert a ready signal when the data transfer has been completed. At the moment the VCB asserts its ready signal it will also drive the data read from the external buffer memory on the PIB, provided that it is a read access. The write frame and octet pointers are used by the VCB to form the absolute memory address or AHB address.

The VCA has two PBI interfaces, one for nominal write accesses and one for auxiliary accesses for Frame Data Field Status storage or Idle Source Packet insertion. Nominal and auxiliary access requests can be made in parallel. The VCM can only make read accesses. The TE can only make a write or a read access at a time.

The PIB can also be implemented with a base address and offset address pair, called physical addressing, but this is a non-commissioned compile time option.

6.4 Virtual Channel Request (VCR) definition

The communication between the VCAs and the VCM is done over the Virtual Channel Request (VCR) interface. Each VCA informs the VCM when there is enough data in the external buffer memory to fill the complete Data Field of the Transfer Frame. This is done by a request signal to the VCM which will acknowledge the request by a grant signal after which the request from the VCA can be released. At the moment the VCA asserts its request signal it also drives the corresponding frame pointer value to inform the VCM of where in its external buffer memory the relevant data resides. This frame pointer value is allowed to change after the request has been granted by the VCM, allowing the VCA to generate a new request immediately after the previous one has been granted.

The VCM will assert a ready signal when the data contents for a frame has been read out from the memory after which the VCA is allowed to write over the memory location with new incoming data. The VCM asserts a poll signal each time a VCA is checked for available data, which can be used for initiating Idle Source Packet insertion in the Virtual Channel, see section 4.2.5.3.

7 PTME DESIGN OPTIONS

The Packet Telemetry Encoder (PTME) VHDL model is largely configurable at compile time by constants defined in a configuration package, PTME_Configuration, allowing the number of supported Virtual Channels to be selected, allocation of memory space, enabling or disabling of encoders etc. The constants from the package are then either propagated to the generics of the sub-modules or used directly in the sub-modules as constants.

The PTME model supports a variety of configuration options which will produce a design tailored to user's needs. The options listed in table 57 and table 58 can be configured before synthesis or compilation of the model.

Parameter	Type	Interpretation	Description
Constants related to the Virtual Channels of the Telemetry Encoder (TME)			
gNumberOfVCs	Natural	range 1 to 8	Number of VCs
gIdleFrameVC	Natural	range 0 to 7	Identification of VC to use for Idle Transfer Frames
gFlexVCId	Natural	range 0 to 1	Flexible VC Id allocation
Constants related to the memory size and partitioning			
gMemoryDepth	Positive	2^n memory bytes	Amount of memory to be shared by all VCs
gAreaDepth	Positive	2^n memory areas	Number of areas into which memory is partitioned
gGroupDepth	Positive	2^n areas grouped	Maximum number of memory areas allowed for any VC
gGroupInterface	Natural	range 0 to 1	automatic / manual external memory area assignment
Bandwidth Allocation Table configuration			
gBatDepth	Positive	2^n entire	Number of BAT entries
Configuration of PTME capabilities			
gFrameLength	Natural	range 0 to 2	223 based, 239 based, both
gAltASM	Natural	range 0 to 1	Alternate Attached Synchronisation Marker support
gTime	Natural	range 0 to 1	Time Strobe support
gSecHeader	Natural	range 0 to 1	Secondary Header support
gOPCF	Natural	range 0 to 1	OPCF/CLCW support
gOPCFLength	Natural	range 0 to 2	CLCW data transfer length 16, 32 or both
gOPCFInterface	Natural	range 0 to 1	TTC-B-01 / synchronous-parallel
gFECW	Natural	range 0 to 1	FECW/CRC support
gBatInterface	Natural	range 0 to 2	no interface / asynchronous / synchronous-parallel
gPreLength	Natural	+1 default	Virtual Channel Multiplexer internal prefetch buffer size
gReedSolomon	Natural	range 0 to 1	Reed-Solomon encoder support
gRSStyle	Natural	range 0 to 3	flip-flops, latches, external memory, 50/50
gUnContiguous	Natural	range 0 to 1	Uncontiguous CADU support
gTurbo	Natural	range 0 to 1	Turbo encoder support
gTurboLengthRd	Natural		Turbo buffer size
gTurboLengthWr	Natural		Turbo buffer size
gTurboLatency	Natural	range 0 to 1	Turbo latency optimisation

Table 57: *PTME configuration parameters*

Parameter	Type	Interpretation	Description
gPseudo	Natural	range 0 to 1	Pseudo-random support
gMark	Natural	range 0 to 1	NRZ-Mark support
gConvolute	Natural	range 0 to 3	no support, basic, punctured, both basic and punctured
gSplit	Natural	range 0 to 1	Split Phase support
Clock divider and clocking style configuration			
gClockDepth	Positive	2^n divider	Clock divider width
gCommonClock	Natural	range 0 to 1	Separate / Common - BitClk and Clk
gClockStyle	Natural	range 0 to 1	Input BitClk / Output BitClk
gClkFrequency	Natural		Clock frequency (Hz) for PacketAsynchronous module
gSyncReset	Natural	range 0 to 1	Asynchronous reset / Synchronous reset
gOPCFBitClock	Natural	range 0 to 1	OPCF using CLCWClk, OPCF using BitClk
Memory addressing style			
gPhysicalAddress	Natural	range 0 to 1	Logical pointer support / Physical address support
gPhysicalDepth	Positive		Physical address width
gMemoryInterface	Natural	range 0 to 1	AMBA AHB interface, asynchronous memory interface
gWaitStates	Natural	range 0 to 1	Wait State support
gWaitStateDepth	Positive	2^n -1 wait states	Maximum number of wait states
gEdacSupport	Natural	range 0 to 3	no / sequential / parallel implementation / both
gEdacType	Natural	range 0 to 1	Hamming Code / Cyclic Code
gMemoryTest	Natural	range 0 to 1	Memory test support
Design optimisation and simplification			
gFPGA	Natural	range 0 to 1	FPGA targeted
gSlowVCAExtra	Natural	range 0 to 1	Slow access support for VCA auxiliary write
gSlowVCAWrite	Natural	range 0 to 1	Slow access support for VCA write
gAcknowledgeVCB	Natural	range 0 to 1	Acknowledge support in VCB
gFrameCheck	Natural	range 0 to 1	Check frame status in VCM

Table 57: *PTME configuration parameters*

Parameter	Type	Interpretation	Description
Constants related to the individual Virtual Channels of the Telemetry Encoder (TME)			
gPacket	Natural	range 0 to 1	Packet support
gIdle	Natural	range 0 to 1	Idle Source Packet generation support
gReady	Natural	range 0 to 1	Ready-for-segment signalling support
gEmpty	Natural	range 0 to 1	Buffer empty signalling support
gAbort	Natural	range 0 to 1	Abort packet insertion support
gLength	Natural	+1 default	Internal input buffer size for Virtual Channel Assembler
gInterface	Natural	range 0 to 4	PacketWire, PacketAsynchronous, PacketParallel, AMBA APB (PAPB), no internal interface module
gPAPBDataSize	Natural	range 1 to 4	8 bits, 16 bits, 24 bits, 32 bits
gGroupSize	Natural		2^n memory areas allocated to Virtual Channel

Table 58: *PTME configuration parameters (for individual Virtual Channels)*

8 PTME INTERFACES

The Packet Telemetry Encoder (PTME) VHDL model interfaces are listed in table 59 and described in the subsequent sections.

Name	Type	Mode	Description	Remark
System interface				
Reset_N	Std_ULogic	in	Asynchronous reset	
Clk	Std_ULogic	in	System clock	
Bit rate clock interface				
BitClk	Std_ULogic	in	Bit clock	
Memory allocation interface (non-commissioned)				
MaxOctetPtrs	OctetPtrMatrixType	in	Max number of octets	
MaxFramePtrs	FramePtrMatrixType	in	Max number of frames	
BaseAreaPtrs	AreaPtrMatrixType	in	Base area pointers	
BaseAddresses	PhysicAddrMatrixType	in	Base addresses	
TurboBase	PhysicAddrType	in	Turbo base address	
TurboFill	TurboOffSetType	in	Turbo logical fill	
General configuration interface				
AltASM	Std_ULogic	in	Enable alternate ASM	
TimeMode	Std_Logic_Vector(0 to 3)	in	Selects time rate	
FrameLen	Std_Logic_Vector(0 to 2)	in	Selects frame length	
SCId	Std_Logic_Vector(0 to 9)	in	Spacecraft Id	
IdleFlexVCId	VCPtrType	in	VC Id	separate Idle Transfer Frame channel
IdleSecHeader	Std_ULogic	in	Secondary Header	
IdleSegmentLen	Std_Logic_Vector(0 to 1)	in	Segment Length Id	
OPCF	Std_ULogic	in	OPCF/CLCW when '1'	
FECW	Std_ULogic	in	FECW/CRC when '1'	
ReedSolomon	Std_ULogic	in	Reed-Solomon coding	
Turbo	Std_ULogic	in	Turbo coding	
TurboRate	Std_Logic_Vector(0 to 1)	in	Turbo code rate	
Pseudo	Std_ULogic	in	Pseudo-random coding	
Mark	Std_ULogic	in	NRZ-M coding	
Convolute	Std_ULogic	in	Convolutional coding	
ConvoluteRate	Std_Logic_Vector(0 to 2)	in	Convolutional rate	
Split	Std_ULogic	in	Split Phase-L coding	
EdacEnable	Std_ULogic	in	Enables EDAC	
EdacParallel	Std_ULogic	in	Select parallel EDAC	
WaitStateRd	WaitStateType	in	Read wait states	
WaitStateWr	WaitStateType	in	Write wait states	
OutputBitRate	ClockPtrType	in	Output bit rate	

Table 59: *PTME interfaces*

Name	Type	Mode	Description	Remark
Virtual Channel Assembler configuration interfaces				
FlexVCId	VCPtrMatrixType	in	Flexible VC Id	
PollThreshold	ThreeMatrixType	in	Poll count threshold	
RdyThreshold	TwoMatrixType	in	VC ready threshold	
DynamicFHP	OneMatrixType	in	Enable FHP insertion	
SecHeader	OneMatrixType	in	Secondary Header	
Sync	OneMatrixType	in	Packets / bit stream	
PktOrder	OneMatrixType	in	Packet order flag	
SegmentLen	TwoMatrixType	in	Segment Length Id	
PktVersion	OneMatrixType	in	Idle Packet Version	
BaudRate	TwoMatrixType	in	Baud rate selector	
IgnorParity	OneMatrixType	in	Ignore parity bit	
TwoStopBits	OneMatrixType	in	Two stop bits	
Virtual Channel Interface (VCI) (non-commissioned)				
VCIBegin	OneMatrixType	in	Packet start detect	
VCIActive	OneMatrixType	in	Packet in progress	
VCIComplete	OneMatrixType	in	Packet end detect	
VCIHold	OneMatrixType	in	Packet to be held	
VCIAbort	OneMatrixType	in	Packet aborted	
VCIAvailable	OneMatrixType	in	Data available	
VCIData	OctetMatrixType	in	Data input	
VCIBusy	OneMatrixType	out	Not ready for octet	
VCIRdy	OneMatrixType	out	Ready for paket	
VCIEmpty	OneMatrixType	out	No packet in buffer	
PacketWire input interface				
PWValid	OneMatrixType	in	Packet delimiter	
PWClk	OneMatrixType	in	Bit clock	
PWData	OneMatrixType	in	Data	
PWAbort	OneMatrixType	in	Abort packet	
PWBusy_N	OneMatrixType	out	Not ready for octet	
PWRdy	OneMatrixType	out	Ready for paket	
PWEmpty	OneMatrixType	out	No packet in buffer	
PacketAsynchronous input interface				
PAValid_N	OneMatrixType	in	Packet delimiter	
PAData	OneMatrixType	in	Asynchronous bit	
PAAbort	OneMatrixType	in	Abort packet	
PABusy_N	OneMatrixType	out	Not ready for octet	
PARdy	OneMatrixType	out	Ready for paket	
PAEmpty	OneMatrixType	out	No packet in buffer	

Table 59: *PTME interfaces*

Name	Type	Mode	Description	Remark
PacketParallel input interface				
PPValid_N	OneMatrixType	in	Packet delimiter	
PPWr_N	OneMatrixType	in	Octet write strobe	
PPData	OctetMatrixType	in	Octet data	
PPAbort	OneMatrixType	in	Abort packet	
PPBusy_N	OneMatrixType	out	Not ready for octet	
PPRdy	OneMatrixType	out	Ready for paket	
PPEmpty	OneMatrixType	out	No packet in buffer	
PacketAPB input interface				
PCLK	Std_ULogic	in	Interface clock	
PRESETn	Std_ULogic	in	Synchronised reset	
PAPBIn	APB_Slv_In_Vector(7 downto 0)	in	Interface input	
PAPBOut	APB_Slv_Out_Vector(7 downto 0)	out	Interface output	
PAPBBusy_N	OneMatrixType	out	Not ready for octet	
PAPBRdy	OneMatrixType	out	Ready for paket	
PAPBEmpty	OneMatrixType	out	No packet in buffer	
Memory test interface (non-commissioned)				
TestReq	Std_ULogic	in	Request	
TestRead	Std_ULogic	in	Read access	
TestWrite	Std_ULogic	in	Write access	
TestGnt	Std_ULogic	out	Access granted	
TestRdy	Std_ULogic	out	Access completed	
TestAddr	StdAddrType	in	Address	
TestRdData	Octet	out	Data output	
TestWrData	Octet	in	Data input	
TestBaseAddr	PhysicAddrType	in	Base address	
TestOffset	PhysicAddrType	in	Offset address	
Memory interface				
CS_N	Std_ULogic	out	Chip select	
Rd_N	Std_ULogic	out	Enable data read	
Wr_N	Std_ULogic	out	Enable data write	
Address	StdAddrType	out	Address	
Data	Octet	inout	Data	
EDAC interface				
Edac	Octet	inout	Check bits	
EdacErr	Std_Logic_Vector(0 to 1)	out	Error code	
AMBA AHB Master interface				
AHBMasterIn	AHB_Mst_In_Type	in		
AHBMasterOut	AHB_Mst_Out_Type	out		
Telemetry test interface (non-commissioned)				
Test	Std_ULogic	in	test enable	

Table 59: *PTME interfaces*

Name	Type	Mode	Description	Remark
TestIn	Std_ULogic	in	telemetry data	
TestSyncIn	Std_ULogic	in	ASM delimiter	
TestFrameIn	Std_ULogic	in	Frame delimiter	
TestTurboSync	Std_ULogic	in	Turbo delimiter	
TestTurboFrame	Std_ULogic	in	Turbo delimiter	
TestTurboTerm	Std_ULogic	in	Turbo delimiter	
TestOut	Std_ULogic	out	telemetry data	
TestSyncOut	Std_ULogic	out	ASM delimiter	
TestFrameOut	Std_ULogic	out	Frame delimiter	
TestBitTick	Std_ULogic	out	bit delimiter	
Channel Access Data Unit output interface				
TimeStrobe	Std_ULogic	out	Time strobe	
CADUSyncMark	Std_ULogic	out	ASM delimiter	
CADUFrameMark	Std_ULogic	out	Frame delimiter	
CADUOddFrame	Std_ULogic	out	Odd numbered frame	
CADUClk	Std_ULogic	out	CADU clock	
CADUOut	Std_ULogic	out	CADU data	
CADUG1Out	Std_ULogic	out	Convoluted CADU data	
CADUG2Out	Std_ULogic	out	Convoluted CADU data	
Bandwidth Allocation Table interface				
BatPriority	Std_ULogic	in	Priority VC select	
BatRegister	BatFileType	in	Register file	
BatWrite	Std_ULogic	out	Allowed to write	
BatCS_N	Std_ULogic	in	Chip select	
BatRW_N	Std_ULogic	in	Read/write	
BatA	BatAddressType	in	Address	
BatD	BatDataType	inout	Data input	
Operation Control Field / CLCW / TTC-B-01 interface				
CLCWLength	Std_ULogic	in	CLCW length select	
CLCWData	Std_Logic_Vector(0 to 31)	in	CLCW parallel data	
CLCWWrite	Std_ULogic	out	Allowed to write	
CLCWOverWrite	Std_ULogic	in	Overwrite bit 16 and 17	
CLCWNoRFAvail	Std_ULogic	in	Bit 16	
CLCWNoBitLock	Std_ULogic	in	Bit 17	
TCId0	Std_Logic_Vector(0 to 5)	in	TC VC Id no. 0	
TCId1	Std_Logic_Vector(0 to 5)	in	TC VC Id no. 1	
CLCWClk	Std_ULogic	in	CLCW clock	
CLCWSel	Std_ULogic	in	Selects TTCD0/TTCD1	
TTCSample	Std_ULogic	out	TTC-B-01 sample	
TTCClk	Std_ULogic	out	TTC-B-01 clock	
TTCD0	Std_ULogic	in	TTC-B-01 input no. 0	
TTCD1	Std_ULogic	in	TTC-B-01 input no. 1	

Table 59: *PTME interfaces*

8.1 System interface

8.1.1 ***Reset_N*: Synchronised reset:** Std_ULogic **(I)**

This active low input signal asynchronously resets the PTME VHDL model. The signal is assumed to be asynchronous.

8.1.2 ***Clk*: System clock:** Std_ULogic **(I)**

This input signal is the system clock signal for the PTME VHDL model. Most registers are clocked on the rising ***Clk*** edge.

8.2 Bit rate interface

8.2.1 ***BitClk*: Bit clock:** Std_ULogic **(I)**

This is the bit clock used by the various encoders in the PTME. All registers are clocked on the rising edge.

8.3 Memory allocation interface

Non-commissioned interface.

8.4 General configuration interface

8.4.1 ***AltASM*: Enable alternate ASM:** Std_ULogic **(I)**

Enables the output of the alternative ASM. Not effective with Turbo Encoding. The PTME should be reset after each change.

8.4.2 ***TimeMode*: Time rate selection:** Std_Logic_Vector(0 to 3) **(I)**

Selects the rate of the time strobe periodicity as per table 24. The PTME should be reset after each change.

8.4.3 ***FrameLen*: Transfer Frame length selection:** Std_Logic_Vector(0 to 2) **(I)**

Selects the Transfer Frame length as per table 20. The PTME should be reset after each change.

8.4.4 ***SCId*: Spacecraft Identifier:** Std_Logic_Vector(0 to 9) **(I)**

Sets the Spacecraft Identifier. The PTME should be reset after each change.

8.4.5 ***IdleFlexVCId*: Flexible VC Id for Idle Transfer Frames:** VCPTRType **(I)**

Sets the VC Id for Idle Transfer Frames when output on a separate Virtual Channel and in configuration flexible allocation is implemented. The PTME should be reset after each change.

8.4.6 ***IdleSecHeader*: Secondary Header for Idle Transfer Frames:** Std_ULogic **(I)**

Enables Secondary Header generation for Idle Transfer Frames when output on a separate Virtual Channel. The PTME should be reset after each change.

8.4.7 *IdleSegmentLen*: Segment Length Identifier for Idle Transfer Frames: Std_Logic_Vector(0 to 1) **(I)**

Data bits in Frame Data Field Status of Idle Transfer Frames when output on a separate Virtual Channel. The PTME should be reset after each change.

8.4.8 *OPCF*: Operational Control Field enable: Std_ULogic **(I)**

Enables OPCF generation when asserted. The PTME should be reset after each change.

8.4.9 *FECW*: Frame Error Control Word enable: Std_ULogic **(I)**

Enables FECW generation when asserted. The PTME should be reset after each change.

8.4.10 *ReedSolomon*: Reed-Solomon coding enable: Std_ULogic **(I)**

Enables Reed-Solomon encoding when asserted. The PTME should be reset after each change.

8.4.11 *Turbo*: Turbo coding enable: Std_ULogic **(I)**

Enables Turbo encoding when asserted. The PTME should be reset after each change.

8.4.12 *TurboRate*: Reed-Solomon coding rate: Std_Logic_Vector(0 to 1) **(I)**

Selects Turbo code rate as per table 53. The PTME should be reset after each change.

8.4.13 *Pseudo*: Pseudo-Randomiser enable: Std_ULogic **(I)**

Enables Pseudo-Randomisation when asserted. The PTME should be reset after each change.

8.4.14 *Mark*: Non-Return-to-Zero - Mark enable: Std_ULogic **(I)**

Enables non-return-to-zero - mark encoding when asserted. The PTME should be reset after each change.

8.4.15 *Convolute*: Convolutional encoding enable: Std_ULogic **(I)**

Enables Convolutional encoding when asserted. The PTME should be reset after each change.

8.4.16 *ConvoluteRate*: Convolutional coding rate: Std_Logic_Vector(0 to 1) **(I)**

Selects Convolutional code rate as per table 54. The PTME should be reset after each change.

8.4.17 *Split*: Split-Phase - Level enable: Std_ULogic **(I)**

Enables split-phase - level modulation when asserted. The PTME should be reset after each change.

8.4.18 *EdacEnable*: EDAC enable: Std_ULogic **(I)**

Enables EDAC protection of memory when asserted. Only effective when memory interface and EDAC are implemented. The PTME should be reset after each change.

8.4.19 *EdacParallel*: Parallel EDAC selection: Std_ULogic (I)

Selects parallel EDAC operation when asserted, else sequential EDAC operation is selected. Only effective when memory interface and EDAC with both parallel and sequential operation are implemented. The PTME should be reset after each change.

8.4.20 *WaitStateRd*: Wait states for read access: WaitStateType (I)

Sets the number of wait states for a read access. Only effective when memory interface is implemented with wait state support. The PTME should be reset after each change.

8.4.21 *WaitStateWr*: Wait states for write access: WaitStateType (I)

Sets the number of wait states for a write access. Only effective when memory interface is implemented with wait state support. The PTME should be reset after each change.

8.4.22 *OutputBitRate*: Output bit rate selection: ClockPtrType (I)

Sets the bit rate derived from the system clock for the encoding chain. Only effective when the system clock is enabled for symbol rate generation. The PTME should be reset after each change.

8.5 Virtual Channel Assembler configuration interfaces

The different Virtual Channels Assemblers are individually configured via this interface. An input array of entire is provided for each parameters, being indexed by the Virtual Channel number. The description hereafter is based on the type of the array entries rather than the array itself.

8.5.1 *FlexVCId*: Virtual Channel Identifier selection: VCPtrMatrixType (I)

Sets the Virtual Channel Identifier in configuration flexible allocation is implemented. The PTME should be reset after each change.

8.5.2 *PollThreshold*: Poll count threshold: Std_Logic_Vector(0 to 2) (I)

Sets the poll threshold for Idle Source Packet insertion, as per table 27. The PTME should be reset after each change.

8.5.3 *RdyThreshold*: Memory availability threshold: Std_Logic_Vector(0 to 1) (I)

Sets the external buffer memory availability threshold, as per table 26. The PTME should be reset after each change.

8.5.4 *DynamicFHP*: Dynamic FHP enable: Std_ULogic (I)

Enables dynamic calculation of First Header Pointer when asserted. The PTME should be reset after each change.

8.5.5 *SecHeader*: Secondary Header Flag: Std_ULogic (I)

Enables Secondary Header generation when asserted. The PTME should be reset after each change.

8.5.6 *Sync*: Data Field Synchronisation Flag: Std_ULogic (I)

Indicates whether packets are synchronously inserted, when 0_b, or asynchronously inserted, when 1_b. Acts as a qualifier for Idle Source Packet generation. The PTME should be reset after each change.

8.5.7 *PktOrder*: Packet Order Flag: Std_ULogic (I)

Data bit in Frame Data Field Status. The PTME should be reset after each change.

8.5.8 *SegmentLen*: Segment Length Identifier: Std_Logic_Vector(0 to 1) (I)

Data bits in Frame Data Field Status. The PTME should be reset after each change.

8.5.9 *PktVersion*: Packet Order Flag: Std_ULogic (I)

Data bit in Idle Source Packet, corresponding to bit 0 of the Version Number. The PTME should be reset after each change.

8.5.10 *BaudRate*: Baud rate selection: Std_Logic_Vector(0 to 1) (I)

Sets the baud rate for the PA interface, as per table 30. The PTME should be reset after each change.

8.5.11 *IgnorParity*: Ignore parity bit: Std_ULogic (I)

When asserted, a parity is included in the received data on the PA interface and should be ignored. The PTME should be reset after each change.

8.5.12 *TwoStopBits*: Two stop bits: Std_ULogic (I)

When asserted, two stop bits received on the PA interface and should be checked for validity. The PTME should be reset after each change.

8.6 Virtual Channel Interface (VCI)

Non-commissioned interface.

8.7 PacketWire (PW) input interface

The different interfaces are individually interfaced via this interface. An array of entire is provided for each port, being indexed by the Virtual Channel number. The description hereafter is based on the type of the array entries rather than the array itself. Note that only some of the entries in the array might actually be used in a design.

8.7.1 *PWValid*: Packet delimiter: Std_ULogic (I)

This input signal is the packet delimiter for the interface. It should be de-asserted between packets.

8.7.2 *PWClk*: Bit clock: Std_ULogic (I)

This input signal is the PackeWire bit clock. The receiver registers are clocked on the rising ***PWClk*** edge.

8.7.3 *PWData*: Data: Std_ULogic (I)

This input signal is the serial data input for the interface. Data are sampled on the rising ***PWClk*** edge when ***PWValid*** is asserted.

8.7.4 *PWRdy*: Ready for paket: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one segment. The output is clocked out on the rising ***Clk*** edge.

8.7.5 *PWBusy_N*: Not ready for data: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one octet. The output is clocked out on the rising ***Clk*** edge.

8.7.6 *PWAbort*: Abort paket: Std_ULogic (I)

This signal indicates that a packet should be aborted.

8.7.7 *PWEmpty*: No packet in buffer: Std_ULogic (O)

This signal indicates whether the Virtual Channel does not contain any user packet data. It can however contain idle packet data. The output is clocked out on the rising ***Clk*** edge.

8.8 PacketAsynchronous (PA) input interface

The different interfaces are individually interfaced via this interface. An array of entire is provided for each port, being indexed by the Virtual Channel number. The description hereafter is based on the type of the array entries rather than the array itself. Note that only some of the entries in the array might actually be used in a design.

8.8.1 *PAValid_N*: Packet delimiter: Std_ULogic (I)

This input signal is the packet delimiter for the interface. It should be de-asserted between packets. It can also be tied to logic zero if non-packet data is to be transferred.

8.8.2 *PAData*: Data: Std_ULogic (I)

This input signal is the bit serial asynchronous data input for the interface.

8.8.3 *PARdy*: Ready for paket: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one segment. The output is clocked out on the rising ***Clk*** edge.

8.8.4 *PABusy_N*: Not ready for data: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one octet. The output is clocked out on the rising ***Clk*** edge.

8.8.5 *PAAbort*: Abort paket: Std_ULogic (I)

This signal indicates that a packet should be aborted.

8.8.6 *PAEmpty*: No packet in buffer: Std_ULogic (O)

This signal indicates whether the Virtual Channel does not contain any user packet data. It can however contain idle packet data. The output is clocked out on the rising ***Clk*** edge.

8.9 PacketParallel (PP) input interface

The different interfaces are individually interfaced via this interface. An array of entire is provided for each port, being indexed by the Virtual Channel number. The description hereafter is based on the type of the array entries rather than the array itself. Note that only some of the entries in the array might actually be used in a design.

8.9.1 *PPValid_N*: Packet delimiter: Std_ULogic (I)

This input signal is the packet delimiter for the interface. It should be de-asserted between packets. It can also be tied to logic zero if non-packet data is to be transferred.

8.9.2 *PPData*: Data: Octet (I)

This input signal is the 8 bit parallel data input for the interface. It is sampled on the rising ***PPWr_N*** edge.

8.9.3 *PPWr_N*: Octet write strobe: Std_ULogic (I)

This input signal is the octet write strobe. The data is sampled on the rising ***PPWr_N*** edge.

8.9.4 *PPRdy*: Ready for paket: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one segment. The output is clocked out on the rising ***Clk*** edge.

8.9.5 *PPBusy_N*: Not ready for data: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one octet. The output is clocked out on the rising ***Clk*** edge.

8.9.6 *PPAbort*: Abort paket: Std_ULogic (I)

This signal indicates that a packet should be aborted.

8.9.7 *PPEmpty*: No packet in buffer: Std_ULogic (O)

This signal indicates whether the Virtual Channel does not contain any user packet data. It can however contain idle packet data. The output is clocked out on the rising ***Clk*** edge.

8.10 PacketAPB (PAPB) input interface

The different interfaces are individually interfaced via this interface. An array of entire is provided for each port, being indexed by the Virtual Channel number. Note that only some of the entries in the array might actually be used in a design. The description hereafter is based on the type of the array entries rather than the array itself. For detailed information on the records used for the APB interface see AD9 and AD12. The clock and reset signals are common for all interfaces.

8.10.1 *PRESETn*: Synchronised reset: Std_ULogic (I)

This active low input signal asynchronously resets the AMBA APB interfaces in the PTME VHDL core. The signal is assumed to be synchronous with the AMBA APB clock ***PCLK*** rising edge. The input is used on registers that are all clocked on the rising ***PCLK*** edge.

8.10.2 *PCLK*: Interface clock: Std_ULogic (I)

This input signal is the AMBA APB clock which is the clock for the AMBA APB interfaces in the PTME VHDL core. All registers are clocked on the rising ***PCLK*** edge.

8.10.3 *PAPBIn*: Interface input: APB_Slv_In_Type (I)

This signal record is the slave interface input.

8.10.3.1 *PSEL*: Slave select: Std_ULogic (I)

This signal indicates that the APB slave device is selected and a data transfer is required. The input is sampled on the rising ***PCLK*** edge for write accesses.

8.10.3.2 *PENABLE*: Enable strobe: Std_ULogic (I)

This strobe signal is used to time all accesses on the peripheral bus. The enable signal is used to indicate the second cycle of an APB transfer. The rising edge of ***PENABLE*** occurs in the middle of the APB transfer. The input is sampled on the rising ***PCLK*** edge for write accesses.

8.10.3.3 *PADDR*: Address bus: Std_Logic_Vector(*PAMAX*-1 downto 0) (I)

This is the APB address bus can be up to 32 bits wide. The input is sampled on the rising ***PCLK*** edge for write accesses. *PAMAX* is defined in the AMBA VHDL package, AD12.

8.10.3.4 *PWRITE*: Write strobe: Std_ULogic (I)

This signal indicates the APB transfer direction When asserted this signal indicates an APB write access and when de-asserted a read access. The input is sampled on the rising ***PCLK*** edge.

8.10.3.5 *PWDATA*: Write data bus: Std_Logic_Vector(*PDMAX*-1 downto 0) (I)

The APB write data bus is driven by the peripheral bus master during write cycles (when ***PWRITE*** is asserted). The data is sampled on the rising ***PCLK*** edge for write accesses. *PDMAX* is defined in the AMBA VHDL package, AD12.

8.10.4 *PAPBOut*: Interface output: APB_Slv_Out_Type (O)

This signal record is the slave interface output.

8.10.4.1 *PRDATA*: Read data bus: Std_Logic_Vector(*PDMAX*-1 downto 0) (O)

The APB read data bus is driven by the selected slave during read cycles (when ***PWRITE*** is de-asserted). *PDMAX* is defined in the AMBA VHDL package, AD12.

8.10.5 *PAPBRdy*: Ready for paket: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one segment. The output is clocked out on the rising ***Clk*** edge.

8.10.6 *PAPBBusy_N*: Not ready for data: Std_ULogic (O)

This signal indicates whether the Virtual Channel is ready to receive one octet. The output is clocked out on the rising ***Clk*** edge.

8.10.7 *PAPBEmpty*: No packet in buffer: Std_ULogic (O)

This signal indicates whether the Virtual Channel does not contain any user packet data. It can however contain idle packet data. The output is clocked out on the rising ***Clk*** edge.

8.11 Memory test interface

Non-commissioned interface.

8.12 Memory interface

8.12.1 *CS_N*: Memory chip select: Std_ULogic (O)

This active low output is the chip select for the external memory. ***CS_N*** changes state on the rising ***Clk*** edge.

8.12.2 *Wr_N*: Memory write strobe: Std_ULogic (O)

This active low output is the write strobe for the external memory. ***Wr_N*** changes state on the falling ***Clk*** edge.

8.12.3 *Rd_N*: Memory read strobe: Std_ULogic (O)

This active low output is the read strobe for the external memory. ***Rd_N*** changes state on the rising ***Clk*** edge.

8.12.4 *Address* Memory address: StdAddrType (O)

These outputs are the addresses generated by the PTME for reading and writing to the external memory. They change state on the rising ***Clk*** edge.

8.12.5 *Data*: Memory data: Octet (I/O)

These signals are used to read and write data to the external memory. They are driven by the PTME on the rising ***Clk*** edge and are tristated on the falling ***Clk*** edge.

8.13 EDAC interface

8.13.1 *Edac*: EDAC check bit data: Octet (I/O)

These signals are used to read and write EDAC check bits to the external memory. They are driven by the PTME on the rising ***Clk*** edge and are tristated on the falling ***Clk*** edge.

8.13.2 *EdacErr*: EDAC error flags: Std_Logic_Vector (0 to 1) (I/O)

These signals are used signal errors detected by the EDAC. They are output on the rising ***Clk*** edge.

8.14 AMBA AHB master interface

For detailed information on the records used for the AMBA AHB interface see AD10 and AD12.

8.14.1 *AHBMasterIn*: Interface input: AHB_Mst_In_Type (I)

This signal record is the general AHB master interface input.

8.14.1.1 *HGRANT*: Bus grant: Std_ULogic (I)

This signal indicates that the AHB master device is selected and a data transfer is required. The input is sampled on the rising ***HCLK*** edge.

8.14.1.2 *HREADY*: Transfer done: Std_ULogic (I)

This signal indicates to the AHB master device that an access is completed. The input is sampled on the rising ***HCLK*** edge.

8.14.1.3 *HRESP*: Response type: Std_Logic_Vector(1 downto 0) (I)

This signal indicates to the AHB master device with what a result an access has been completed. The input is sampled on the rising ***HCLK*** edge.

8.14.1.4 *HRDATA*: Read data bus: Std_Logic_Vector(*HDMAX*-1 downto 0) (I)

This signal provides the AHB master device read data at the end of the access. The input is sampled on the rising ***HCLK*** edge. *HDMAX* is defined in the AMBA VHDL package, AD12. *HDMAX* is assumed to be the default 32.

8.14.2 *AHBMasterOut*: Interface output: AHB_Mst_Out_Type (O)

This signal record is the general AHB master interface output.

8.14.2.1 *HBUSREQ*: Bus request: Std_ULogic (O)

This signal indicates that the AHB master device is requesting the bus. The output is clocked out on the rising ***HCLK*** edge.

8.14.2.2 *HLOCK*: Lock request: Std_ULogic (O)

This signal indicates whether the AHB master device is requesting a locked access. The output is permanently driven to logical zero.

8.14.2.3 *HTRANS*: Transfer type: Std_Logic_Vector(1 downto 0) (O)

This signal indicates the type of the transfer that the AHB master device is issuing. The output is clocked out on the rising ***HCLK*** edge.

8.14.2.4 ***HADDR*: Transfer type:** Std_Logic_Vector(*HAMAX*-1 downto 0) **(O)**

This signal carries the address of the transfer that the AHB master device is issuing. The output is clocked out on the rising ***HCLK*** edge. *HAMAX* is defined in the AMBA VHDL package, AD12. *HAMAX* is assumed to be the default 32.

8.14.2.5 ***HWRITE*: Read / Write:** Std_ULogic **(O)**

This signal indicates whether it is a read or a write access that the AHB master device is issuing. The output is clocked out on the rising ***HCLK*** edge.

8.14.2.6 ***HSIZE*: Transfer size:** Std_Logic_Vector(2 downto 0) **(O)**

This signal indicates the size of the access that the AHB master device is issuing. The output is permanently driven to logical zeros, indicating a byte access.

8.14.2.7 ***HBURST*: Burst type:** Std_Logic_Vector(2 downto 0) **(O)**

This signal indicates the burst type of access that the AHB master device is issuing. The output is permanently driven to logical zeros, indicating a single access.

8.14.2.8 ***HPROT*: Protection control:** Std_Logic_Vector(32 downto 0) **(O)**

This signal indicates the protection type of access that the AHB master device is issuing. The output is permanently driven to logical zeros.

8.14.2.9 ***HWDATA*: Write data bus:** Std_Logic_Vector(*HDMAX*-1 downto 0) **(O)**

This signal carries the data of the write transfer that the AHB master device is issuing. The output is clocked out on the rising ***HCLK*** edge. *HDMAX* is defined in the AMBA VHDL package, AD12. *HDMAX* is assumed to be the default 32. Since only byte accesses are performed, the byte data to be written is copied to all four byte positions on ***HWDATA***.

8.15 Telemetry test interface

Non-commissioned interface.

8.16 Channel Access Data Unit output interface

8.16.1 *TimeStrobe*: Time strobe: Std_ULogic (O)

This signal is asserted when a time strobe is to be generated as specified for Virtual Channel 0. The output is clocked out on the rising ***BitClk*** edge.

8.16.2 *CADUSyncMark*: ASM delimiter: Std_ULogic (O)

This signal is asserted when the Attached Synchronisation Marker is being output. The output is clocked out on the rising ***BitClk*** edge.

8.16.3 *CADUFrameMark*: Transfer frame delimiter: Std_ULogic (O)

This signal is asserted when the Transfer Frame is being output. The output is clocked out on the rising ***BitClk*** edge.

8.16.4 *CADUClk*: CADU clock: Std_ULogic (O)

This signal is the bit delimiter for the CADU output. The output is clocked out on the rising ***BitClk*** edge.

8.16.5 *CADUOut*: CADU data: Std_ULogic (O)

This signal is the bit serial CADU data output. The output is clocked out on the rising ***BitClk*** edge.

8.16.6 *CADUOddFrame*: Odd numbered Transfer Frame: Std_ULogic (O)

This signal is asserted when a Transfer Frame for which the Master Channel Frame Count value is odd. The output is clocked out on the rising ***BitClk*** edge.

8.16.7 *CADUG1Out*: Convoluted CADU data: Std_ULogic (O)

This output carries the G1 data vector from the Convolutional encoder. The output is clocked out on the rising ***BitClk*** edge.

8.16.8 *CADUG2Out*: Convoluted CADU data: Std_ULogic (O)

This output carries the G2 data vector from the Convolutional encoder. The output is clocked out on the rising ***BitClk*** edge.

8.17 Bandwidth Allocation Table interface

8.17.1 *BatPriority*: Priority mode: Std_ULogic (I)

Enables priority mode when asserted. The PTME should be reset after each change.

8.17.2 *BatRegister*: Register file: BatFileType (I)

Input of the full BAT register file when the external synchronous interface option is selected. The PTME should be reset if value changes out side the ***BatWrite*** window.

8.17.3 *BatWrite*: Register file write permission: Std_ULogic (O)

Indicates when the register file ***BatRegister*** can be modified without the need for a PTME reset when the external synchronous interface option is selected.

8.17.4 *BatCS_N*: Chip select: Std_ULogic (I)

Active low chip select for BAT access when asynchronous interface option is selected.

8.17.5 *BatRW_N*: Read/Write indicator: Std_ULogic (I)

Active read/write indicator for BAT access when asynchronous interface option is selected.

8.17.6 *BatA*: Address: BatAddressType (I)

Address for the BAT access when asynchronous interface option is selected.

8.17.7 *BatD*: Data: BatDataType (I/O)

Data for BAT read and write access when asynchronous interface option is selected.

8.18 Operation Control Field / CLCW / TTC-B-01 interface

8.18.1 *CLCWLength*: CLCW data transfer length: Std_ULogic (I)

Indicates the size of the CLCW data word to be transferred. Only used when both 16 and 32 bit CLCW data width support is implemented. The PTME should be reset after each change.

8.18.2 *CLCWData*: CLCW parallel data: Std_Logic_Vector(0 to 31) (I)

Synchronous parallel CLCW data input when the external synchronous interface option is selected. Should not change out side the ***CLCWWrite*** window.

8.18.3 *CLCWWrite*: CLCW write permission: Std_ULogic (O)

Indicates when ***CLCWData*** can be modified when the external synchronous interface is used.

8.18.4 *CLCWOverWrite*: Overwrite bits 16 and 17: Std_ULogic (I)

Indicates that bit 16 and 17 should be overwritten with the contents of the two following inputs. The PTME should be reset after each change.

8.18.5 *CLCWNoRFAvail*: No RF Available, bit 16: Std_ULogic (I)

8.18.6 *CLCWNoBitLock*: No Bit Lock, bit 17: Std_ULogic (I)

8.18.7 *TCId0*: Virtual Channel Identifier setting: Std_Logic_Vector(0 to 5) (I)

Virtual Channel Identifier setting for CLCW transmitted while ***CLCWSel*** is asserted.

8.18.8 *TCId1*: Virtual Channel Identifier setting: Std_Logic_Vector(0 to 5) (I)

Virtual Channel Identifier setting for CLCW transmitted while ***CLCWSel*** is de-asserted.

8.18.9 *CLCWClk*: Clock: Std_ULogic (I)

Clock input used for generation of TTC-B-01 protocol.

8.18.10 *CLCWSel*: CLCW selection: Std_ULogic (I)

Selection between ***TCId0/TTCD0*** or ***TCId1/TTCD1*** transmission as part of the CLCW.

8.18.11 *TTCSample*: Packet delimiter: Std_ULogic (O)

This signal is the packet delimiter for the interface.

8.18.12 *TTCClk*: Bit clock: Std_ULogic (O)

This signal is the bit clock for the interface. The data are sampled on the rising ***TTCClk*** edge.

8.18.13 *TTCD0*: Data: Std_ULogic (I)

Data sampled on the rising ***TTCClk*** edge when ***CLCWSel*** is de-asserted.

8.18.14 *TTCD1*: Data: Std_ULogic (I)

Data sampled on the rising ***TTCClk*** edge when ***CLCWSel*** is asserted.

9 PTME VHDL SOURCE CODE DESCRIPTION

The Packet Telemetry Encoder (PTME) model is written in synthesizable VHDL, currently targeted towards the Synplify synthesis tool from Synplicity, but is also compatible with the Synopsys Design Compiler. The PTME is an almost fully synchronous design based on a single system clock strategy. The asynchronous part is related to the different interfaces of the core, such as the PacketWire (PW) input interface. The model and test benches are written according to RD4 as far as applicable. The VHDL code complies to VHDL'93, RD5.

9.1 Packages and libraries, interface port and generic types

The following VHDL packages are used in the PTME VHDL model:
- Std.Standard,
- IEEE.Std_Logic_1164, IEEE.Std_Logic_Arith
- AMBA_Lib.AMBA
- PTME_Lib.PTME_Configuration, PTME_Lib.PTME_Definition

The PTME VHDL model interfaces does not comply to the normally required *Std_ULogic* and *Std_Logic_Vector* types, RD4, since it often uses the *UnSigned* array type.

The recommended target library for the PTME VHDL model is PTME_Lib. Note that the AMBA interface can be located in another library than AMBA_Lib, but this will require a modification of the PTME VHDL code.

There are no generics used for the top entity in the PTME VHDL model.

9.2 Compilation order

The compilation order for the PTME is as follows:

• AMBA_Lib:	amba.vhd
• PTME_Lib:	ptme_lib.vhd

The compilation order for the PTME test bench is as follows:

• CCSDS_Lib:	ccsds_lib.vhd
• PTME_TB_Lib:	ptme_tb.vhd

9.3 Simulation

A testbench is provided with the PTME VHDL model which can be used to set up verification runs. The output from the simulation is in ASCII text format and can be processed off line to verify the Reed-Solomon encoder, the Turbo encoder and the telemetry encoder. The testbench is configured with the same configuration package as the PTME VHDL model. The testbench will adapt itself to the selected PTME configuration.

9.4 Model hierarchy

The PTME VHDL model hierarchy is listed in table 60.

PTME_Configuration (package)				
PTME_Definition (package)				
PacketTelemetryEncoder (PTME)				
	VirtualChannelEncoder (VCE)			*Telemetry Encoder (TME)*
		VirtualChannelAssembler (VCA)		
		VirtualChannelMultiplexer (VCM)		
		VirtualChannelBuffer (VCB)		
	PacketAsynchronous (PA)			
	PacketAPB (PAPB)			
	PacketParallel (PP)			
	PacketWire (PW)			
	TurboEncoder (TE)			
		Turbo_Control		
		Turbo_ConstituentEncoders		
		Turbo_Interleaver		
		Turbo_Table		
	RS_Package (package)			
	ReedSolomonEncoder (RSE)			
		RS_Adder		
		RS_Ctrl		
		RS_Multiplier		
		RS_SerialShift		
		RS_Memory		
			RS_MemoryBlock	
				RS_MemoryLatch
	PseudoRandomiser (PSR)			
	ConvolutionalEncoder (CE)			
	NonReturnZero (NRZ)			
	SplitPhase (SP)			
	ClockDivider (CD)			

Table 60: *PTME VHDL model hierarchy*

APPENDIX A: THEORETICAL BACKGROUND ON REED-SOLOMON CODING

This appendix is based on the textbooks RD11 and RD12, and on the ESA and CCSDS standards AD3 and AD4, respectively.

A.1 Reference documents

RD11	Error-Control Coding for Computer Systems, T. Rao et al., Prentice-Hall International, 1989, USA
RD12	Coded-Modulation Techniques for Fading Channels, S. Jamil et al., Kluwer Academic Publishers, 1994, USA

A.2 Reed-Solomon encoding

Cyclic codes are linear codes having the property that for a given code any cyclic shift of a codeword is also a codeword. Cyclic codes are characterised by using a polynomial representation for codewords and manipulation of polynomials for encoding and decoding procedures.

An (*n, k*) cyclic code has the length *n* of which *k* symbols are information symbols.

A codeword polynomial of such a code can be represented as

$$c(x) = a(x) \cdot g(x)$$

where

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_1x + c_0$$

$$a(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \ldots + a_1x + a_0$$

$$g(x) = g_rx^r + g_{r-1}x^{r-1} + \ldots + g_1x + g_0$$

Polynomial $a(x)$ is called the information polynomial and is of degree k-1 or less corresponding to the k information symbols $a_{k\text{-}1}$, $a_{k\text{-}2}$, ..., a_1, a_0, to be encoded.

Polynomial $c(x)$ is called the codeword polynomial and is of degree n-1, corresponding to the encoded information polynomial.

The variable x is a dummy variable only used for indicating the strength or the position of the corresponding coefficients.

Polynomial $g(x)$ is called the generator polynomial and is of degree $r = n$-1. The generator polynomial has two properties: its coefficients g_0 and g_r are always non zero; and it divides x^n+1. Using a polynomial of degree n-k which is a factor of x^n+1 we can generate an (*n, k*) cyclic code.

Bose-Chaudhuri-Hocquenghem (BCH) codes are a subclass of cyclic codes, where the symbols are chosen from a Galois field $GF(q)$, with the generator polynomial having roots in $GF(q^{m*})$, an extension field of $GF(q)$.

Reed-Solomon codes are a subclass of BCH codes, where $GF(q)$ and $GF(q^{m^*})$ are chosen to be the same. The rest of the discussion will be restricted to Reed-Solomon codes with symbols from $GF(2^m)$. More on operations in $GF(2^m)$ can be found in appendix A.3.

An (n, k) primitive Reed-Solomon code defined over $GF(2^m)$ has a code length of $n = 2^m$-1 and a minimum Hamming distance of n-k+1. The number of symbols is n and the number of check symbols is n-k. The minimum Hamming distance is $d_{min} = n$-k+1. A code can correct any combination of t errors and detect up to d errors ($d \geq t$) if and only if $d_{min} \geq t$+d+1. Each symbol is an element of $GF(2^m)$.

The generator polynomial of a Reed-Solomon code can be constructed as the least common multiple of the minimal polynomials of n-k consecutive powers of the primitive elements of $GF(2^m)$, i.e. α^h, α^{h+1}, ..., $\alpha^{h+n-k+1}$, where h is an integer. This means that the generator polynomial of a Reed-Solomon code can be written as

$$g(x) = \prod_{i=0}^{n-k-1} (x + \alpha^{h+1}) = \sum_{j=0}^{n-k} g_j x^j$$

With different values of h, h different generator polynomials can be formed. For

$$h = (k+1)/2$$

with odd k, $g(x)$ is a self-reciprocal polynomial, i.e.

$$g(x) = x^{n-k} g\left(\frac{1}{x}\right)$$

A self-reciprocal generator polynomial has its coefficients satisfying the following relation

$$g_j = g_{n-k-j}$$

Using self-reciprocal generator polynomials will reduce the number of calculations to be performed in a Reed-Solomon encoder.

The previous definition of the codeword polynomial calculation for an information polynomial forms a non-systematic codeword, i.e. the information and check symbols are mixed, making it difficult to extract the information symbols. One can however construct a codeword where the first part contains the information symbols and the last part contains the check symbols, according to the following transformations. It shall be noted that a codeword must contain the generator polynomial $g(x)$ as a factor.

Let us multiply the information polynomial $a(x)$ with x^{n-k}. A division of $a(x)x^{n-k}$ with $g(x)$ results in the quotient $q(x)$ and the reminder $r(x)$

$$\frac{a(x)x^{n-k}}{g(x)} = q(x) + \frac{r(x)}{g(x)}$$

which can be rewritten as

$$a(x)x^{n-k} = q(x)g(x) + r(x)$$

and finally rewritten as

$$a(x)x^{n-k} + r(x) = q(x)g(x)$$

where $q(x)$ is of degree k-1, and $r(x)$ is per definition of one degree less than $g(x)$ which is r-1 = n-k-1. The reminder $r(x)$ can be represented as

$$r(x) = a(x)x^{n-k}\operatorname{mod} g(x) = \sum_{i=0}^{r-1} r_i x^i = \sum_{i=0}^{n-k-1} r_i x^i$$

The codeword of a dummy information polynomial $q(x)$ is according to previous definitions

$$c(x) = q(x)g(x) = a(x)x^{n-k} + r(x)$$

where $c(x)$ is indeed a factor of $g(x)$. It can also be seen that the first part of the codeword contains the information symbols and the last part the check symbols

$$c(x) = a(x)x^{n-k} + r(x) = \sum_{i=n-k}^{n-1} c_i x^i + \sum_{i=0}^{n-k-1} c_i x^i = \sum_{i=0}^{k-1} a_i x^{n-k+i} + \sum_{i=0}^{r-1} r_i x^i$$

The remainder polynomial $r(x)$ of the division $a(x)/g(x)$ can be calculated using the Linear Feedback Shift Register in figure 15, characterised by the equation

$$x^r \operatorname{mod} g(x) = g_{r-1}x^{r-1} + g_{r-2}x^{r-2} + \ldots + g_1 x + x_0$$

which with $r = n$-k implements the desired encoder. Note that all additions and multiplications of symbols are made in $GF(2^m)$.

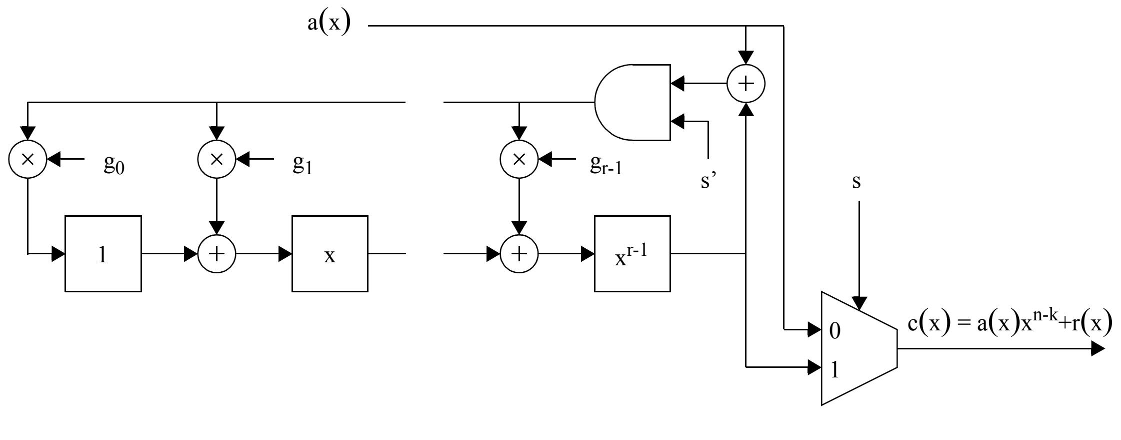

Figure 15: *Codeword polynomial generator*

A.3 Galois fields

A finite Galois field *GF*(2) contains two elements *0* and *1*. The two operations addition and multiplication are defined as:

Addition		
+	*0*	*1*
0	*0*	*1*
1	*1*	*0*

Multiplication		
*	*0*	*1*
0	*0*	*0*
1	*0*	*1*

The element *0* and *1* are the additive identity and the multiplicative identity elements of *GF*(2), respectively. A polynomial of degree *m* defined over *GF*(2) is represented as

$$f(x) = \sum_{i=0}^{m} f_i x^i$$

where f_i's are elements in *GF*(2), i.e., $f_i = 0$ or *1*. A polynomial *f*(*x*) of degree m with coefficients in *GF*(2) is said to be irreducible if it is not divisible by any polynomial with coefficients in *GF*(2) of degree less than *m* and greater than zero.

The finite field $GF(2^m)$ is an extension of *GF*(2) generated from and irreducible polynomial *f*(*x*) of degree m over *GF*(2). Equivalently, *GF*(2) is a subfield of $GF(2^m)$, i.e., the elements of *GF*(2) are also elements of $GF(2^m)$. The finite field $GF(2^m)$ also has its primitive element α which is the (2^m-1)th root of unity, that is

$$\alpha^{(2^m - 1)} = 1$$

The finite field $GF(2^m)$ contains 2^m elements: *0*, *1*, α, α^2, ..., $\alpha^{(2**m-2)}$. The irreducible polynomial *f*(*x*) of degree *m* over *GF*(2) has no roots in *GF*(2) but its roots lie in some extension field of *GF*(2). If *f*(*x*) has the primitive element α of $GF(2^m)$ as a root, then it is called a primitive irreducible polynomial. In this case, α^i with $i \le m$ can be represented in term of *1*, α, α^2, ..., α^{m-1}, by using the fact that α^i is also a root of *f*(*x*), that is

$$f(\alpha^i) = 0$$

An arbitrary element β of $GF(2^m)$ can be represented as a binary *m*-tuple

$$\beta = (\beta_{m-1}, \beta_{m-2}, ..., \beta_1, \beta_0)$$

In this case, the zero element is represented as $0 = (0, 0,..., 0)$. For $0 \le i \le (m-1)$, the binary components of the m-tuple representing $\beta = \alpha^i$ satisfy the following relation

$$\beta_j = \begin{cases} 1, \text{if} j = i \\ 0, \text{elsewhere} \end{cases} \qquad \text{for } j = \{0, 1, ..., (m-1)\}$$

Let β and γ be two arbitrary elements of $GF(2^m)$ represented in terms of the primitive element α and binary m-tuples as

$$\beta = \alpha^i = (\beta_{m-1}, \beta_{m-2}, ..., \beta_1, \beta_0)$$

$$\gamma = \alpha^j = (\gamma_{m-1}, \gamma_{m-2}, ..., \gamma_1, \gamma_0)$$

Addition in $GF(2^m)$ is defined as

$$\beta + \gamma = \theta = (\theta_{m-1}, \theta_{m-2}, ..., \theta_1, \theta_0)$$

where

$$\theta_i = \beta_i + \gamma_i \qquad \text{for } i = \{0, 1, ..., (m-1)\}$$

Multiplication in $GF(2^m)$ is defined as

$$\beta \cdot 0 = 0$$
$$\beta \cdot 1 = \beta$$
$$\beta \cdot \gamma = \theta = \alpha^i \cdot \alpha^j = \alpha^{(i+j)\text{mod}(2^m - 1)}$$

The logical implementation of the addition is straightforward, since the bit wise + operator can be implemented as an xor-gate. Note that subtraction is identical to addition. The logical implementation of the multiplication is somewhat more complicated. Lets start with a look at the polynomial representation of the multiplication

$$\beta \cdot \gamma = \theta = (\beta_{m-1}x^{m-1} + ,..., + \beta_1 x + \beta_0) \cdot (\gamma_{m-1}x^{m-1} + ,..., + \gamma_1 x + \gamma_0)$$

The result of the expansion of this expression for degree $m = 8$ can be represented as a matrix

$$\beta \cdot \gamma \Leftrightarrow \begin{bmatrix}
\beta_7\gamma_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\beta_7\gamma_6 & \beta_6\gamma_7 & 0 & 0 & 0 & 0 & 0 & 0 \\
\beta_7\gamma_5 & \beta_6\gamma_6 & \beta_5\gamma_7 & 0 & 0 & 0 & 0 & 0 \\
\beta_7\gamma_4 & \beta_6\gamma_5 & \beta_5\gamma_6 & \beta_4\gamma_7 & 0 & 0 & 0 & 0 \\
\beta_7\gamma_3 & \beta_6\gamma_4 & \beta_5\gamma_5 & \beta_4\gamma_6 & \beta_3\gamma_7 & 0 & 0 & 0 \\
\beta_7\gamma_2 & \beta_6\gamma_3 & \beta_5\gamma_4 & \beta_4\gamma_5 & \beta_3\gamma_6 & \beta_2\gamma_7 & 0 & 0 \\
\beta_7\gamma_1 & \beta_6\gamma_2 & \beta_5\gamma_3 & \beta_4\gamma_4 & \beta_3\gamma_5 & \beta_2\gamma_6 & \beta_1\gamma_7 & 0 \\
\beta_7\gamma_0 & \beta_6\gamma_1 & \beta_5\gamma_2 & \beta_4\gamma_3 & \beta_3\gamma_4 & \beta_2\gamma_5 & \beta_1\gamma_6 & \beta_0\gamma_7 \\
0 & \beta_6\gamma_0 & \beta_5\gamma_1 & \beta_4\gamma_2 & \beta_3\gamma_3 & \beta_2\gamma_4 & \beta_1\gamma_5 & \beta_0\gamma_6 \\
0 & 0 & \beta_5\gamma_0 & \beta_4\gamma_1 & \beta_3\gamma_2 & \beta_2\gamma_3 & \beta_1\gamma_4 & \beta_0\gamma_5 \\
0 & 0 & 0 & \beta_4\gamma_0 & \beta_3\gamma_1 & \beta_2\gamma_2 & \beta_1\gamma_3 & \beta_0\gamma_4 \\
0 & 0 & 0 & 0 & \beta_3\gamma_0 & \beta_2\gamma_1 & \beta_1\gamma_2 & \beta_0\gamma_3 \\
0 & 0 & 0 & 0 & 0 & \beta_2\gamma_0 & \beta_1\gamma_1 & \beta_0\gamma_2 \\
0 & 0 & 0 & 0 & 0 & 0 & \beta_1\gamma_0 & \beta_0\gamma_1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_0\gamma_0
\end{bmatrix}$$

where the top row contains the coefficient of x^{14} being $c_{14} = \beta_7\gamma_7$, the next row contains the coefficient of x^{13} being $c_{13} = (\beta_7\gamma_6+\beta_6\gamma_7)$, and so on down to the coefficient of x^0 being $c_0 = \beta_0\gamma_0$. The powers from 8 to 14 of x (i.e. m to m-1+m-1) can be expressed in powers of x ranging from 0 to 7 (i.e. 0 to m-1) according to a mapping as defined by the field polynomial $f(x)$ of degree m defining the Galois field. This can be achieved by simple matrix manipulation. First we collapse the upper part of the matrix to a vector for which each element is the sum s_i of the elements in the corresponding row of the matrix

$$\begin{bmatrix}
(\beta_7\gamma_7) \\
(\beta_7\gamma_6 + \beta_6\gamma_7) \\
(\beta_7\gamma_5 + \beta_6\gamma_6 + \beta_5\gamma_7) \\
(\beta_7\gamma_4 + \beta_6\gamma_5 + \beta_5\gamma_6 + \beta_4\gamma_7) \\
(\beta_7\gamma_3 + \beta_6\gamma_4 + \beta_5\gamma_5 + \beta_4\gamma_6 + \beta_3\gamma_7) \\
(\beta_7\gamma_2 + \beta_6\gamma_3 + \beta_5\gamma_4 + \beta_4\gamma_5 + \beta_3\gamma_6 + \beta_2\gamma_7) \\
(\beta_7\gamma_1 + \beta_6\gamma_2 + \beta_5\gamma_3 + \beta_4\gamma_4 + \beta_3\gamma_5 + \beta_2\gamma_6 + \beta_1\gamma_7)
\end{bmatrix} = \begin{bmatrix} s_{14} \\ s_{13} \\ s_{12} \\ s_{11} \\ s_{10} \\ s_9 \\ s_8 \end{bmatrix}$$

A matrix corresponding to the coefficients of α^8 to α^{14} (i.e. α^m to α^{2m-2}) for the field polynomial $f(x)$ is shown here

$$\{f(\alpha^{14}), ..., f(\alpha^8)\} \Leftrightarrow \begin{bmatrix} f_{14_7} & f_{14_6} & f_{14_5} & f_{14_4} & f_{14_3} & f_{14_2} & f_{14_1} & f_{14_0} \\ f_{13_7} & f_{13_6} & f_{13_5} & f_{13_4} & f_{13_3} & f_{13_2} & f_{13_1} & f_{13_0} \\ f_{12_7} & f_{12_6} & f_{12_5} & f_{12_4} & f_{12_3} & f_{12_2} & f_{12_1} & f_{12_0} \\ f_{11_7} & f_{11_6} & f_{11_5} & f_{11_4} & f_{11_3} & f_{11_2} & f_{11_1} & f_{11_0} \\ f_{10_7} & f_{10_6} & f_{10_5} & f_{10_4} & f_{10_3} & f_{10_2} & f_{10_1} & f_{10_0} \\ f_{9_7} & f_{9_6} & f_{9_5} & f_{9_4} & f_{9_3} & f_{9_2} & f_{9_1} & f_{9_0} \\ f_{8_7} & f_{8_6} & f_{8_5} & f_{8_4} & f_{8_3} & f_{8_2} & f_{8_1} & f_{8_0} \end{bmatrix}$$

By multiplying the vector containing the sums s_i as a scalar to the corresponding row elements of the matrix containing the coefficients α^8 to α^{14} of the field polynomial $f(x)$, the following matrix is obtained

$$\begin{bmatrix} s_{14} \\ s_{13} \\ s_{12} \\ s_{11} \\ s_{10} \\ s_9 \\ s_8 \end{bmatrix} \text{»} \begin{bmatrix} f_{14_7} & f_{13_7} & f_{12_7} & f_{11_7} & f_{10_7} & f_{9_7} & f_{8_7} \\ f_{14_6} & f_{13_6} & f_{12_6} & f_{11_6} & f_{10_6} & f_{9_6} & f_{8_6} \\ f_{14_5} & f_{13_5} & f_{12_5} & f_{11_5} & f_{10_5} & f_{9_5} & f_{8_5} \\ f_{14_4} & f_{13_4} & f_{12_4} & f_{11_4} & f_{10_4} & f_{9_4} & f_{8_4} \\ f_{14_3} & f_{13_3} & f_{12_3} & f_{11_3} & f_{10_3} & f_{9_3} & f_{8_3} \\ f_{14_2} & f_{13_2} & f_{12_2} & f_{11_2} & f_{10_2} & f_{9_2} & f_{8_2} \\ f_{14_1} & f_{13_1} & f_{12_1} & f_{11_1} & f_{10_1} & f_{9_1} & f_{8_1} \\ f_{14_0} & f_{13_0} & f_{12_0} & f_{11_0} & f_{10_0} & f_{9_0} & f_{8_0} \end{bmatrix} \Rightarrow \begin{bmatrix} f_{14_7}s_{14} & f_{13_7}s_{13} & f_{12_7}s_{12} & f_{11_7}s_{11} & f_{10_7}s_{10} & f_{9_7}s_9 & f_{8_7}s_8 \\ f_{14_6}s_{14} & f_{13_6}s_{13} & f_{12_6}s_{12} & f_{11_6}s_{11} & f_{10_6}s_{10} & f_{9_6}s_9 & f_{8_6}s_8 \\ f_{14_5}s_{14} & f_{13_5}s_{13} & f_{12_5}s_{12} & f_{11_5}s_{11} & f_{10_5}s_{10} & f_{9_5}s_9 & f_{8_5}s_8 \\ f_{14_4}s_{14} & f_{13_4}s_{13} & f_{12_4}s_{12} & f_{11_4}s_{11} & f_{10_4}s_{10} & f_{9_4}s_9 & f_{8_4}s_8 \\ f_{14_3}s_{14} & f_{13_3}s_{13} & f_{12_3}s_{12} & f_{11_3}s_{11} & f_{10_3}s_{10} & f_{9_3}s_9 & f_{8_3}s_8 \\ f_{14_2}s_{14} & f_{13_2}s_{13} & f_{12_2}s_{12} & f_{11_2}s_{11} & f_{10_2}s_{10} & f_{9_2}s_9 & f_{8_2}s_8 \\ f_{14_1}s_{14} & f_{13_1}s_{13} & f_{12_1}s_{12} & f_{11_1}s_{11} & f_{10_1}s_{10} & f_{9_1}s_9 & f_{8_1}s_8 \\ f_{14_0}s_{14} & f_{13_0}s_{13} & f_{12_0}s_{12} & f_{11_0}s_{11} & f_{10_0}s_{10} & f_{9_0}s_9 & f_{8_0}s_8 \end{bmatrix}$$

This matrix is then concatenated with the lower part of the matrix containing the expansion of the multiplication of the two polynomials β and γ, which results in a final matrix where the sum of each row corresponds to the coefficients (θ_{m-1}, ..., θ_0) of the product between β and γ

$$\begin{bmatrix} f_{14_7}s_{14} & f_{13_7}s_{13} & f_{12_7}s_{12} & f_{11_7}s_{11} & f_{10_7}s_{10} & f_{9_7}s_9 & f_{8_7}s_8 & \beta_7\gamma_0 & \beta_6\gamma_1 & \beta_5\gamma_2 & \beta_4\gamma_3 & \beta_3\gamma_4 & \beta_2\gamma_5 & \beta_1\gamma_6 & \beta_0\gamma_7 \\ f_{14_6}s_{14} & f_{13_6}s_{13} & f_{12_6}s_{12} & f_{11_6}s_{11} & f_{10_6}s_{10} & f_{9_6}s_9 & f_{8_6}s_8 & 0 & \beta_6\gamma_0 & \beta_5\gamma_1 & \beta_4\gamma_2 & \beta_3\gamma_3 & \beta_2\gamma_4 & \beta_1\gamma_5 & \beta_0\gamma_6 \\ f_{14_5}s_{14} & f_{13_5}s_{13} & f_{12_5}s_{12} & f_{11_5}s_{11} & f_{10_5}s_{10} & f_{9_5}s_9 & f_{8_5}s_8 & 0 & 0 & \beta_5\gamma_0 & \beta_4\gamma_1 & \beta_3\gamma_2 & \beta_2\gamma_3 & \beta_1\gamma_4 & \beta_0\gamma_5 \\ f_{14_4}s_{14} & f_{13_4}s_{13} & f_{12_4}s_{12} & f_{11_4}s_{11} & f_{10_4}s_{10} & f_{9_4}s_9 & f_{8_4}s_8 & 0 & 0 & 0 & \beta_4\gamma_0 & \beta_3\gamma_1 & \beta_2\gamma_2 & \beta_1\gamma_3 & \beta_0\gamma_4 \\ f_{14_3}s_{14} & f_{13_3}s_{13} & f_{12_3}s_{12} & f_{11_3}s_{11} & f_{10_3}s_{10} & f_{9_3}s_9 & f_{8_3}s_8 & 0 & 0 & 0 & 0 & \beta_3\gamma_0 & \beta_2\gamma_1 & \beta_1\gamma_2 & \beta_0\gamma_3 \\ f_{14_2}s_{14} & f_{13_2}s_{13} & f_{12_2}s_{12} & f_{11_2}s_{11} & f_{10_2}s_{10} & f_{9_2}s_9 & f_{8_2}s_8 & 0 & 0 & 0 & 0 & 0 & \beta_2\gamma_0 & \beta_1\gamma_1 & \beta_0\gamma_2 \\ f_{14_1}s_{14} & f_{13_1}s_{13} & f_{12_1}s_{12} & f_{11_1}s_{11} & f_{10_1}s_{10} & f_{9_1}s_9 & f_{8_1}s_8 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_1\gamma_0 & \beta_0\gamma_1 \\ f_{14_0}s_{14} & f_{13_0}s_{13} & f_{12_0}s_{12} & f_{11_0}s_{11} & f_{10_0}s_{10} & f_{9_0}s_9 & f_{8_0}s_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \beta_0\gamma_0 \end{bmatrix} \Rightarrow \begin{bmatrix} \theta_7 \\ \theta_6 \\ \theta_5 \\ \theta_4 \\ \theta_3 \\ \theta_2 \\ \theta_1 \\ \theta_0 \end{bmatrix}$$

Multiplication with a constant in a fixed Galois field is somewhat more simple and is used in the calculation of the reminder as explained for Reed-Solomon codes. Assume that β is the variable and γ is the constant to be multiplied with. By collapsing the matrix above and factoring out β, the following matrix can be obtained. The vector containing β is multiplied as a scalar to the corresponding row elements of the matrix, which results in a final matrix where the sum of each row corresponds to the coefficients (θ_{m-1}, ..., θ_0) of the product between β and γ

$$\begin{bmatrix}\beta_7\\\beta_6\\\beta_5\\\beta_4\\\beta_3\\\beta_2\\\beta_1\\\beta_0\end{bmatrix} \text{»} \begin{bmatrix}
f(\gamma\alpha^7)_7 & f(\gamma\alpha^7)_6 & f(\gamma\alpha^7)_5 & f(\gamma\alpha^7)_4 & f(\gamma\alpha^7)_3 & f(\gamma\alpha^7)_2 & f(\gamma\alpha^7)_1 & f(\gamma\alpha^7)_0\\
f(\gamma\alpha^6)_7 & f(\gamma\alpha^6)_6 & f(\gamma\alpha^6)_5 & f(\gamma\alpha^6)_4 & f(\gamma\alpha^6)_3 & f(\gamma\alpha^6)_2 & f(\gamma\alpha^6)_1 & f(\gamma\alpha^6)_0\\
f(\gamma\alpha^5)_7 & f(\gamma\alpha^5)_6 & f(\gamma\alpha^5)_5 & f(\gamma\alpha^5)_4 & f(\gamma\alpha^5)_3 & f(\gamma\alpha^5)_2 & f(\gamma\alpha^5)_1 & f(\gamma\alpha^5)_0\\
f(\gamma\alpha^4)_7 & f(\gamma\alpha^4)_6 & f(\gamma\alpha^4)_5 & f(\gamma\alpha^4)_4 & f(\gamma\alpha^4)_3 & f(\gamma\alpha^4)_2 & f(\gamma\alpha^4)_1 & f(\gamma\alpha^4)_0\\
f(\gamma\alpha^3)_7 & f(\gamma\alpha^3)_6 & f(\gamma\alpha^3)_5 & f(\gamma\alpha^3)_4 & f(\gamma\alpha^3)_3 & f(\gamma\alpha^3)_2 & f(\gamma\alpha^3)_1 & f(\gamma\alpha^3)_0\\
f(\gamma\alpha^2)_7 & f(\gamma\alpha^2)_6 & f(\gamma\alpha^2)_5 & f(\gamma\alpha^2)_4 & f(\gamma\alpha^2)_3 & f(\gamma\alpha^2)_2 & f(\gamma\alpha^2)_1 & f(\gamma\alpha^2)_0\\
f(\gamma\alpha)_7 & f(\gamma\alpha)_6 & f(\gamma\alpha)_5 & f(\gamma\alpha)_4 & f(\gamma\alpha)_3 & f(\gamma\alpha)_2 & f(\gamma\alpha)_1 & f(\gamma\alpha)_0\\
f(\gamma)_7 & f(\gamma)_6 & f(\gamma)_5 & f(\gamma)_4 & f(\gamma)_3 & f(\gamma)_2 & f(\gamma)_1 & f(\gamma)_0
\end{bmatrix} \Rightarrow \begin{bmatrix}\theta_7\\\theta_6\\\theta_5\\\theta_4\\\theta_3\\\theta_2\\\theta_1\\\theta_0\end{bmatrix}$$

A.4 Derivation of ESA standard from CCSDS recommendation

Since the definition of the (255, 223) Reed-Solomon code differs between the ESA standard, AD3, and the CCSDS recommendation, AD4, a theoretical derivation of the ESA standard from the CCSDS recommendation is performed hereafter, showing the required equivalence.

A.4.1 CCSDS standard

The field polynomial in AD4 is specified as

$$f_{ccsds}(x) = x^8 + x^7 + x^2 + x + 1$$

The generator polynomial, in the case of E=16, in AD4 is specified as

$$g_{ccsds}(x) = \prod_{i=112}^{143} (x - \alpha^{11j}) = \sum_{j=0}^{32} g_j \cdot x^j$$

Where $n = 255$, $k = 223$ and $h = 112$ results in a self-reciprocal generator polynomial

$$h = (k+1)/2 \Rightarrow 112 = (223+1)/2$$

By expanding the generator polynomial the following coefficients are obtained:

Coefficients of $g_{ccsds}(x)$	**Polynomial in α_{ccsds}**
$g_0 = g_{32}$	α^0
$g_1 = g_{31}$	α^{249}
$g_2 = g_{30}$	α^{59}
$g_3 = g_{29} = g_{13} = g_{19}$	α^{66}
$g_4 = g_{28}$	α^4
$g_5 = g_{27}$	α^{43}
$g_6 = g_{26}$	α^{126}
$g_7 = g_{25}$	α^{251}
$g_8 = g_{24}$	α^{97}
$g_9 = g_{23}$	α^{30}
$g_{10} = g_{22}$	α^3
$g_{11} = g_{21}$	α^{213}
$g_{12} = g_{20}$	α^{50}
$g_{14} = g_{18}$	α^{170}
$g_{15} = g_{17}$	α^5
g_{16}	α^{24}

Table 61: *Coefficients of the generator polynomial $g_{ccsds}(x)$ as per* AD4

The input and output data from the decoder shall be in the dual basis representation as per AD4 where ι_0 is transmitted first. Note that the order has been reversed compared to the GF representation otherwise used in this text. Note also that this new representation is the dual basis to a basis β_{ccsds}, with the relation $\beta_{ccsds} = \alpha_{ccsds}{}^{117}$. The following transformation matrix T_{ccsds} specifies the new basis

$$\begin{bmatrix} \iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7 \end{bmatrix} = \begin{bmatrix} \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \end{bmatrix} \times \begin{bmatrix} 1&0&0&0&1&1&0&1\\ 1&1&1&0&1&1&1&1\\ 1&1&1&0&1&1&0&0\\ 1&0&0&0&0&1&1&0\\ 1&1&1&1&1&0&1&0\\ 1&0&0&1&1&0&0&1\\ 1&0&1&0&1&1&1&1\\ 0&1&1&1&1&0&1&1 \end{bmatrix}$$

The following matrix $T^{-1}{}_{ccsds}$ specifies the revers transformation

$$\begin{bmatrix} \alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0 \end{bmatrix} = \begin{bmatrix} \iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7 \end{bmatrix} \times \begin{bmatrix} 1&1&0&0&0&1&0&1\\ 0&1&0&0&0&0&1&0\\ 0&0&1&0&1&1&1&0\\ 1&1&1&1&1&1&0&1\\ 1&1&1&1&0&0&0&0\\ 0&1&1&1&1&0&0&1\\ 1&0&1&0&1&1&0&0\\ 1&1&0&0&1&1&0&0 \end{bmatrix}$$

A.4.2 ESA standard

The field polynomial in AD3 is specified as

$$f_{esa}(x) = x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$$

The generator polynomial in AD3 is specified as

$$g_{esa}(x) = \prod_{i=112}^{143} (x + \alpha^i) = \sum_{j=0}^{32} g_j \cdot x^j$$

Powers of the primitive element α are roots in the ESA generator polynomial $g_{esa}(x)$, whereas powers of the eleventh power of the primitive element α, i.e. α^{11}, is used in CCSDS generator polynomial $g_{ccsds}(x)$. To obtain the same generator polynomial for the CCSDS and ESA standards the relationship between the field polynomials $f_{esa}(x)$ and $f_{ccsds}(x)$ should be $\alpha_{esa} = \alpha_{ccsds}{}^{11}$.

It can be shown that this relation is indeed valid for the two polynomials. By assuming for a moment that the ESA polynomial $f_{esa}(x)$ is unknown, the following can be deduced from the above relation. For the powers 0 to 7 of α_{esa} we know the corresponding 8-tuple by the definition of a Galois field, as shown in table 62.

α_{esa}	**powers of α_{esa}** $\alpha^7\alpha^6\alpha^5\alpha^4\alpha^3\alpha^2\alpha^1\alpha^0$	α_{ccsds}	**powers of α_{ccsds}** $\alpha^7\alpha^6\alpha^5\alpha^4\alpha^3\alpha^2\alpha^1\alpha^0$
α^0	*00000001*	α^0	*00000001*
α^1	*00000010*	α^{11}	*10101101*
α^2	*00000100*	α^{22}	*10111110*
α^3	*00001000*	α^{33}	*00111010*
α^4	*00010000*	α^{44}	*00111100*
α^5	*00100000*	α^{55}	*11011100*
α^6	*01000000*	α^{66}	*01010110*
α^7	*10000000*	α^{77}	*11001010*

Table 62: *Relationship between α_{esa} and α_{ccsds}*

We also know the 8-tuples for the powers 0, 11, 22, .., 77 of α_{ccsds} since we know the field polynomial $f_{ccsds}(x)$. From table 62 it can be deduced that an ESA 8-tuple can be formed from a CCSDS 8-tuple according to the equations in table 63. Since the coefficients of α^8 in $GF(2^8)$ define the corresponding field polynomial, the polynomial $f_{esa}(x)$ can be obtained by inserting $\alpha_{ccsds}{}^{88} = (0, 1, 0, 0, 0, 0, 1, 0)$ (which corresponds to $\alpha_{esa}{}^{8}$) in to table 63 and solve the resulting equation system, as done in the ten steps shown in table 64 and table 65.

Polynomial in $f_{ccsds}(x)$	**Coefficients of polynomial in $f_{esa}(x)$**
$x^0 =$	x^1+x^0
$x^1 =$	$x^7+x^6+x^3+x^2$
$x^2 =$	$x^6+x^5+x^4+x^2+x^1$
$x^3 =$	$x^7+x^{5+}x^4+x^3+x^2+x^1$
$x^4 =$	$x^6+x^5+x^4+x^3+x^2+x^1$
$x^5 =$	$x^4+x^3+x^2+x^1$
$x^6 =$	$x^7+x^6+x^5$
$x^7 =$	$x^7+x^5+x^2+x^1$

Table 63: *Forming an $f_{esa}(x)$ 8-tuple from an $f_{ccsds}(x)$ 8-tuple*

Polynomial in f_{ccsds}	Coefficients of polynomial in $f_{esa}(x)$	Step 1	Step 2	Step 3	Step 4
$0 = x^0$	x^1+x^0	$x^0=x^1$			
$1= x^1$	$x^7+x^6+x^3+x^2$				$1=x^5+1+x^3+x^0$
$0 = x^2$	$x^6+x^5+x^4+x^2+x^1$				$0=1+x^5+x^3$
$0 = x^3$	$x^7+x^5+(x^4+x^3+x^2+x^1)$	$x^5=x^7$			
$0 = x^4$	$x^6+x^5+x^4+x^3+x^2$				$0=1+x^5+x^0$
$0 = x^5$	$x^4+x^3+x^2+x^1$			$x^3=x^4$	
$1 = x^6$	$x^7+x^6+x^5$		$x^6 = 1$		
$0 = x^7$	$x^7+x^5+x^2+x^1$		$x^0=x^1=x^2$		

Table 64: *Derivation of $f_{esa}(x)$ by solving the equation system for $f_{esa}(\alpha^8)$*

Step 5	Step 6	Step 7	Step 8	Step 9	$f_{esa}(\alpha^8)$
$1=x^5+1+x^3+x^0$	$0=x^5+x^3+x^0$	$0=x^5+x^5+1+x^5+1$	$x^5=0$	$x^7=0$	$x^0=1$
$0=1+x^5+x^3$	$x^3=x^5+1$		$x^3=1$	$x^4=1$	$x^1=1$
$0=1+x^5+x^0$	$x^0=x^5+1$		$x^0=1$	$x^1=x^2=1$	$x^2=1$
					$x^3=1$
					$x^4=1$
					$x^5=0$
					$x^6=1$
					$x^7=0$

Table 65: *Continued derivation of $f_{esa}(x)$*

The resulting polynomial fitting the equations in table 63 is

$$f(x) = x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$$

which is indeed equivalent to the field polynomial $f_{esa}(x)$ as defined in AD3.

By expanding the generator polynomial previously defined the following constants are obtained

Coefficients of $g_{esa}(x)$	**Polynomial in α_{esa}**	**Polynomial in α_{ccsds}**
$g_0 = g_{32}$	α^0	α^0
$g_1 = g_{31}$	α^{69}	α^{249}
$g_2 = g_{30}$	α^{214}	α^{59}
$g_3 = g_{29} = g_{13} = g_{19}$	α^6	α^{66}
$g_4 = g_{28}$	α^{209}	α^4
$g_5 = g_{27}$	α^{143}	α^{43}
$g_6 = g_{26}$	α^{81}	α^{126}
$g_7 = g_{25}$	α^{46}	α^{251}
$g_8 = g_{24}$	α^{32}	α^{97}
$g_9 = g_{23}$	α^{165}	α^{30}
$g_{10} = g_{22}$	α^{93}	α^3
$g_{11} = g_{21}$	α^{228}	α^{213}
$g_{12} = g_{20}$	α^{190}	α^{50}
$g_{14} = g_{18}$	α^{85}	α^{170}
$g_{15} = g_{17}$	α^{70}	α^5
g_{16}	α^{234}	α^{24}

Table 66: *Coefficients of the generator polynomial $g_{esa}(x)$ as per* AD3

Each coefficient of the generator polynomial $g_{esa}(x)$ is expressed in powers of α_{esa} and also in powers of α_{ccsds} following the relations $\alpha_{esa} = \alpha_{ccsds}{}^{11}$. Note that the coefficients expressed in α_{ccsds} are the same as for $g_{ccsds}(x)$.

The input and output data from the decoder shall be in the basis representation as per AD3 where ι_0 is transmitted first. Note that the order has been reversed compared to the $GF(2^8)$ representation otherwise used in this text. The following transformation matrix T_{esa} specifies the new basis

$$\begin{bmatrix}\iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7\end{bmatrix} = \begin{bmatrix}\alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0\end{bmatrix} \times \begin{bmatrix}0&0&1&1&0&1&1&1\\0&1&0&1&1&1&1&1\\1&0&0&0&0&1&1&1\\0&0&0&0&1&0&0&1\\0&0&1&1&1&1&1&1\\0&0&1&0&1&0&1&1\\0&1&1&1&1&0&0&1\\0&1&1&1&1&0&1&1\end{bmatrix}$$

The following matrix T^{-1}_{esa} specifies the revers transformation

$$\begin{bmatrix}\alpha_7 & \alpha_6 & \alpha_5 & \alpha_4 & \alpha_3 & \alpha_2 & \alpha_1 & \alpha_0\end{bmatrix} = \begin{bmatrix}\iota_0 & \iota_1 & \iota_2 & \iota_3 & \iota_4 & \iota_5 & \iota_6 & \iota_7\end{bmatrix} \times \begin{bmatrix}1&1&1&0&1&1&0&1\\0&1&0&1&1&1&1&1\\0&0&0&1&0&1&1&1\\0&1&0&1&1&0&1&0\\1&0&0&0&1&0&0&0\\0&1&0&1&0&1&1&0\\0&0&0&0&0&0&1&1\\1&0&0&1&1&0&0&0\end{bmatrix}$$

By applying all elements of $GF(2^8)$ over $f_{esa}(x)$ to the above transformation matrix, and then do the same for all elements of $GF(2^8)$ over $f_{ccsds}(x)$ to the previously defined transformation matrix, one can easily observe that the two standards map input and output data to the elements of $GF(2^8)_{esa}$ and $GF(2^8)_{ccsds}$ consistently with the $\alpha_{esa} = \alpha_{ccsds}{}^{11}$ relation.

Finally, both standards form a generator polynomial $g(x)$ with equivalent coefficients, both standards map to the same input and output data. It is therefore reasonable to believe that both standards specify the same Reed-Solomon encoder.

The specified Reed-Solomon code can correct up to 16 symbol errors (= t) and detect up to 16 errors (= d), since $d_{min} = n\text{-}k+1$, where $d_{min} = 33$, $n = 255$ and $k = 223$, and the two relations $d_{min} \geq t+d+1$ and $d \geq t$ both hold.

A.5 Parallel multiplication in dual basis

The reason for employing the conversion to the dual basis as has been shown earlier is that a bit serial multiplier which uses few discrete components can be used, as per AD4. However, this bit serial multiplier is protected by the patent RD8 which would pose restrictions on any implementation using it. Instead, a parallel multiplier can be developed which is suitable for VLSI design having a modest gate count.

Based on the circuitry for generating the reminder polynomial shown in figure 15, a straightforward implementation of a Reed-Solomon encoder can be achieved. Since the generator polynomial is specified in the normal basis, as opposed to the dual basis in which input and output data is represented, a transformation is necessary on the input and output of the encoder. But observing that for all representations of $GF(2^8)$ the addition operator is the same, and that only the multiplication operator is related to the field polynomial $f(x)$, one can perform the multiplication and the input and output conversions in a single matrix operation. This will be shown hereafter.

Data is input to the encoder in the dual basis representation and is transformed to normal basis representation by multiplication with the reverse transformation matrix T^{-1}. The resulting data is then multiplied with a coefficient of $g(x)$ which is constant and can be done as previously shown in section A.3. Thereafter there are only additions performed which are independent of $GF(2^8)$ representation. The data is then fed back to the multiplier again. When the data is output, it is first converted to the dual basis representation by multiplication with the transformation matrix *T*. The data flow and conversions can be described as

$$\begin{bmatrix}\iota_0 & .. & \iota_7\end{bmatrix} \times \begin{bmatrix}T^{-1}\end{bmatrix} = \begin{bmatrix}\alpha_7 & .. & \alpha_0\end{bmatrix} \Rightarrow \begin{bmatrix}\alpha_7 \\ .. \\ \alpha_0\end{bmatrix} \gg \begin{bmatrix} f_\alpha(\gamma\alpha^7)_7 & .. & f_\alpha(\gamma\alpha^7)_0 \\ .. & .. & .. \\ f_\alpha(\gamma)_7 & .. & f_\alpha(\gamma)_0 \end{bmatrix} \Rightarrow \begin{bmatrix}\alpha'_7 \\ .. \\ \alpha'_0\end{bmatrix} \Rightarrow \begin{bmatrix}\alpha'_7 & .. & \alpha'_0\end{bmatrix} \times \begin{bmatrix}T\end{bmatrix} = \begin{bmatrix}\iota'_0 & .. & \iota'_7\end{bmatrix}$$

Since all three matrixes are of size 8-by-8, and the only manipulations of the data between the matrix operations is by addition, the three matrixes can be merged into one by regular matrix multiplication after being transposed to take into account the different bit ordering between the two representations

$$\begin{bmatrix}T_{trans}^{-1}\end{bmatrix} \times \begin{bmatrix} f_\alpha(\gamma\alpha^7)_7 & .. & f_\alpha(\gamma\alpha^7)_0 \\ .. & .. & .. \\ f_\alpha(\gamma)_7 & .. & f_\alpha(\gamma)_0 \end{bmatrix} \times \begin{bmatrix}T_{trans}\end{bmatrix} = \begin{bmatrix} m_{7_7} & .. & m_{7_0} \\ .. & .. & .. \\ m_{0_7} & .. & m_{0_0} \end{bmatrix}$$

For example, the following matrix multiplications produces the matrix used for multiplication with the constant α^{143} in $GF(2^8)$ over $f_{esa}(x)$ where the variable is in dual basis representation. The constant α^{143} corresponds to the coefficient g_5 of $g_{esa}(x)$, which is also equivalent to g_5 of $g_{ccsds}(x)$

$$\begin{bmatrix} 1&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&1&1 \\ 0&1&0&1&0&1&1&0 \\ 1&0&0&0&1&0&0&0 \\ 0&1&0&1&1&0&1&0 \\ 0&0&0&1&0&1&1&1 \\ 0&1&0&1&1&1&1&1 \\ 1&1&1&0&1&1&0&1 \end{bmatrix} \times \begin{bmatrix} 0&0&1&1&0&0&0&1 \\ 1&0&1&1&0&1&1&1 \\ 1&1&1&1&0&1&0&0 \\ 0&1&1&1&1&0&1&0 \\ 0&0&1&1&1&1&0&1 \\ 1&0&1&1&0&0&0&1 \\ 1&1&1&1&0&1&1&1 \\ 1&1&0&1&0&1&0&0 \end{bmatrix} \times \begin{bmatrix} 1&1&1&0&1&1&0&0 \\ 1&1&1&1&1&0&1&0 \\ 1&1&1&0&0&0&0&1 \\ 1&0&0&1&0&0&0&0 \\ 1&1&1&1&1&1&0&0 \\ 1&1&0&1&0&1&0&0 \\ 1&0&0&1&1&1&1&0 \\ 1&1&0&1&1&1&1&0 \end{bmatrix} = \begin{bmatrix} 1&1&0&0&0&0&0&1 \\ 1&0&1&0&0&0&0&1 \\ 0&1&0&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0 \\ 1&0&0&1&0&1&0&0 \\ 0&0&0&0&1&0&1&1 \\ 1&0&0&0&0&1&0&1 \\ 1&0&0&0&0&0&1&1 \end{bmatrix}$$

This matrix can now be used for multiplication with g_5 as shown below, where each element of the input vector is multiplied as a scalar to each element in the corresponding row. Each row is then summed to form the corresponding element in the output vector.

$$f(\iota) \cdot g_5 = f(\iota') \Leftrightarrow \begin{bmatrix}\iota_7 \\ \iota_6 \\ \iota_5 \\ \iota_4 \\ \iota_3 \\ \iota_2 \\ \iota_1 \\ \iota_0\end{bmatrix} \gg \begin{bmatrix} 1&1&0&0&0&0&0&1 \\ 1&0&1&0&0&0&0&1 \\ 0&1&0&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0 \\ 1&0&0&1&0&1&0&0 \\ 0&0&0&0&1&0&1&1 \\ 1&0&0&0&0&1&0&1 \\ 1&0&0&0&0&0&1&1 \end{bmatrix} \Rightarrow \begin{bmatrix}\iota'_7 \\ \iota'_6 \\ \iota'_5 \\ \iota'_4 \\ \iota'_3 \\ \iota'_2 \\ \iota'_1 \\ \iota'_0\end{bmatrix}$$