Master thesis

Forward Error Correction and failure rates on Aurora high-speed links

Marc Brakels, s1105000

University of Twente, Thales Hengelo

28 November 2017

Commission:
dr. ir. A.B.J. Kokkeler (UT, CAES)
dr. ir. H. Schurer (Thales Hengelo)
ir. J. Scholten (UT, PS)
ir. E. Molenkamp (UT, CAES)
ing. J. Flierman (Thales Hengelo)

ABSTRACT

Data interconnects require more and more bandwidth increasing the data rate per lane. The result is that the signal quality is less than ideal and one has to be careful in how to handle bit errors. This report will look into the handling of bit errors using Forward Error Correction (FEC). For the Forward Error Correction, a Reed-Solomon algorithm is used because it has a hard bound on error correction strength. As the FEC is not placed next to the transceiver but at a small distance from it, all the logic in between needs to be analysed. This logic in between is the primary source of unrecoverable errors as Reed-Solomon does not protect this. The calculated Mean Time Between Failures (MTBF) is around 40,000 years at BER of 10^{-12}. The system is also simulated but at a higher BER. For BER of $4.88 \cdot 10^{-4}$ a failure rate of 8.86% is found for a block of 170 kb. It is hard to proof the design because of the low BER, a pessimistic estimation with 95% confidence results in an MTBF of 6766 years or more. This is a lot better than no FEC at all. It can be improved by moving the FEC closer to the transceiver.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEFINITIONS

BINARY PREFIX CONVENTION

To prevent mistakes the ISO/IEC 80000 [1] norm is used. This means prefix k, M, G, T, P, E are power of 10 prefixes and ki, Mi, Gi, Ti, Pi, Ei are power of 2 prefixes.

LIST OF ACRONYMS

ADC	Analog to Digital Convertor
BER	Bit Error Rate
BRAM	FPGA Block RAM resource
CC	Clock Correction
DAC	Digital to Analog Convertor
FEC	Forward Error Correction (algorithm)
FF	FPGA Flip Flop resource
FIFO	First In First Out buffer component
FPGA	Field Programmable Gate Array
GCD	Greatest Common Divider
LFSR	Linear-Feedback Shift Register
LUT(6)	FPGA Look Up Table resource
MDBF	Mean Data Between Failures
MTBF	Mean Time Between Failures
PAM	Pulse Amplitude Modulation (PAM-2 is equal to NRZ modulation)
RS	Reed-Solomon

LIST OF VARIABLES

n	Number of symbols per RS block
k	Number of data symbols per RS block
t	Number of correctable errors (t=(n-k)/2)
ch	Number of parallel RS channels
m	Number of bits(digits)

GLOSSARY

Word level synchronisation	The alignment of a 66 bits word in a stream of bits
Block level synchronisation	The alignment of start word in a stream of words
module	Is the Verilog name for VHDL design entity
wireline	All no wireless communication.
GTH	High-performance differential transmitter up to 13.1 Gbps from Xilinx
Coregen	Xilinx IP generation wizard/tool

BINOMIAL DISTRIBUTION

The binomial distribution is often used in this report. It is introduced in appendix B. The probability mass function of binomial distribution is given by

$$f(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Where k is number of successful experiments, n the total number of experiments and p the probability of success for one experiment. $\binom{n}{k}$ is the binomial coefficient.

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

NOTES ON NOTATIONS

For readable of tables, the units are normally not written down. The standard SI unit can be assumed. The following default units are used:

MTBF	s	seconds
MDBF	b	bits
BER		errors per bit

Colour is often used in the illustration to make them more readable; it is recommended having a colour print or having a digital version of figures.

In this report Little-endian can be assumed.

a	Referred to an error from section 2.6 / appendix C.
IDLE	Indicates that it is Aurora word
`Sym_gen`	Indicates that it is the name of a module

ACKNOWLEDGEMENTS

As part of the Master program at the University of Twente, I have to do a final thesis. This thesis consists out of a literature study [2] of 10 EC, and a final part of 30 EC, this report. I have chosen to do this at Thales Hengelo. Thales has provided the assignment and a working place at their building in Hengelo. I will like to thank Jeroen Flierman for providing supervising on a daily basis for things like hardware/software use and other general questions. I also want to thank Hans Schurer who is present at the bi-weekly meetings at Thales to discuss the progress and provide feedback, where also Jeroen was present. I also want to thank Bert Molenkamp and André Kokkeler who provided me with feedback and guidance from the university. For the hardware support I would like to thank Jaap Mol. Also, I want to thank Marijn Ufkes for arrange the contact within Thales.

As this is a follow up on literature study [2], it is recommended to have read it.

1. INTRODUCTION

This report is written with an aim to towards Thales products, mainly focusing on the communication in their RADAR systems. In these systems, there is the need to communicate between the front end and back end but also within the front/back end. The front end has parts such as an Analog to Digital Converter (ADC) / Digital to Analog Converter (DAC), and basic signal processing and the back end has the main processing part. The front end produces/consumes a lot of data (in the 10 Gbps order) and this is not reducing with newer designs as ADC/DAC resolutions increase and higher sample rates are used. Also, the trend is to move more of the front end processing to the back end, to be more flexible. The result is that the data rate increases rapidly.

This report focuses on designing a multi-Gbps no-rf links such as Ethernet [3] and Aurora [4] as shown in figure 1. These gigabit rates are needed because of the growing demand for bandwidth. This increase of bandwidth can be accomplished by adding more lanes, but this takes up more space on circuit boards and adds more cables/fibres, so most of the time the choice is made for a higher data rate per lane. At the moment of writing 10 Gbps is widely available, 25 Gbps is available on the most recent hardware, and 56 Gbps is in the final phase of research and should be available next year. For this research, the Xilinx Kintex UltraScale FPGA series [5] is used with data rates in the order of 10 Gbps. The test board used is the *Xilinx Kintex UltraScale KCU105 Evaluation board* [6].

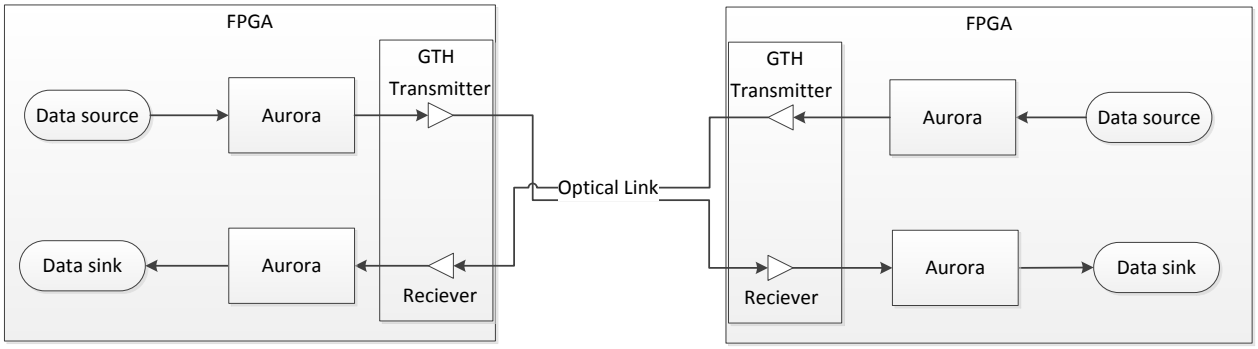

Figure 1 An Aurora link connecting two FPGA with each other via optical cable.

For these systems, it is desired that the communication is fault free or the errors mark the data as corrupt. The goal is to design a system which has a Mean Time Between Failures (MTBF) in the order of 1 million years. The systems used, operate most of the time in hard real-time environments, so it is not desired to have to wait when an error occurs until the error is corrected. Meaning retransmit is not a desired option, as it quickly results in delays in the order of 100 microseconds to a couple of milliseconds when an error occurs. That is why this report is focussing on Forward Error Correction (FEC) where errors are corrected by means of adding extra data to be able to correct errors at runtime with minimum extra latency.

In this report the focus will be at correcting the occasional errors using FEC and making a prediction on the failure rates. This report is a follow up of the literate study [2] on this topic. The 1 million years statement comes from the hypothesis in the literate study which is "Is it possible to design a protocol that enables a very reliable link (MTBF +1 million years) at high data rates (10 Gbps) for BER of 10^{-8} on top of or mixed with an existing protocol." Instead of

using a BER of 10^{-8} the real BER is considered which is 10^{-12}. This value for BER is used because it is the value of IEEE 802.3 10GBASE-R [7, p. 389].

Thales already uses the Aurora [4] protocol, so the system is designed for the Aurora IP [8]. The Aurora specification itself does not supply a clear recommendation for the BER. New standards such as IEEE 100GBASE-R with RS-FEC [9, p. 370] have an even lower BER of 10^{-5} [9, p. 403]. The recommendation given in the literate study [2] to use FEC close to the transceivers is followed.

In chapter 2 the system is thoroughly analysed for the error generation and propagation. In the next chapter the design is explained including the design decisions. In chapter 4 the design is reviewed and validated. Finally the report is concluded with conclusions and recommendations.

2. THEORY

From the literature study, the choice was made to use a Forward Error Correction (FEC) as close as possible to the transmitter and receiver components. The preferred solution is Reed-Solomon [10] (RS) interleaved. Simple putting an FEC IP in the data stream is not working because first of all the IP needs to be synchronised and also the data rate on both sides of the IP are different because of the overhead. So a more advance design is needed. In the figures below, the basic building blocks for encoder and decoder are given. In appendix D the complete transmitter and receiver are given. The first step is taking in the data words after which a bus width conversion is needed as the RS is operating on a different data width. After the RS part the data is again bus width converted and encapsulated in a data word to be transmitted. On the receiver side this process is done in reverse order with of course an RS decoder instead of an encoder. The way data is interleaved is illustrated in figure 4.

The choices for the design are explain in chapter 3. For this chapter it is enough to know the general outline as shown in figure 2 and 3. The parameters used are given in table 4, section 3.1. But still the parameters can be chosen differently to some extent.

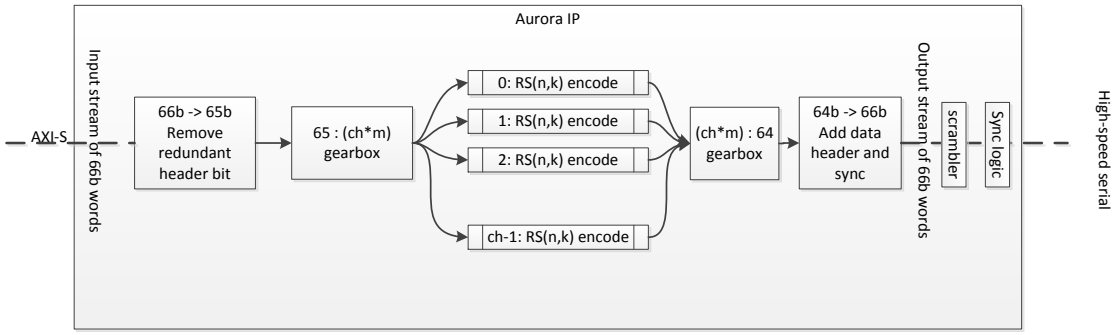

Figure 2 RS encoder flow note that the FIFO are not shown. They are between all the steps.

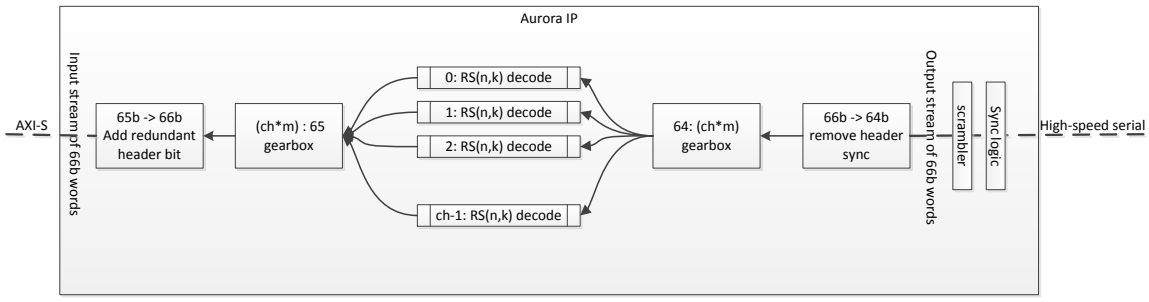

Figure 3 RS decoder flow, again the FIFO are not shown.

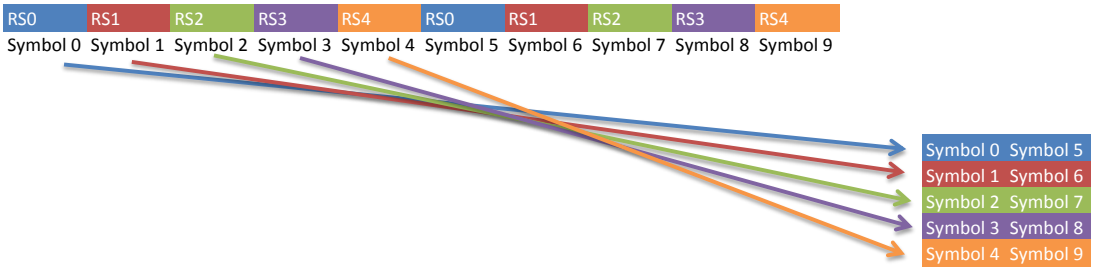

Figure 4 Example interleave with 5 channels

2.1. AURORA

For the communication Aurora [4] is used. Aurora is a high-speed serial protocol designed by Xilinx and the protocol specification is open. It is a low-level protocol and does not support routing nor retransmit. It is designed to do point-to-point communication with limited flow control. Also it shares a lot with IEEE 802.3 [7] such as word-level synchronisation, gearbox and scrambler. The protocol works on word-level, transmitting 64 bits data as a 66 bits word or 8 bits data as 10 bits word. The first one is used in this report, see also figure 5. These two extra bits are used for two things: word-level synchronisation and making differences between control and data words.

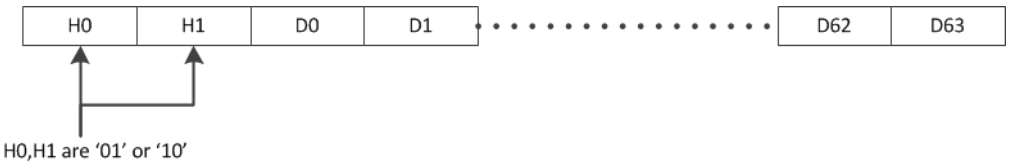

Figure 5 The structure of a 66 bits Aurora word

This word-level synchronisation is needed as the data is transferred over a single differential pair. So the receiving party does not know the word alignment in the bit stream. The header has two values '01' and '10'; the receiver just starts receiving and counting the number of '01' and '10' at the chosen header location. If the header is aligned then it will only see '01' and '10'. If the header is not aligned then it will see the data which is scrambled. So, it will also detect '00' and '11'. If the number of these wrong headers is high, then it will move its alignment to the next location and start testing if this is the correct position. This will continue until the header has been found at the receiver side, then it marks the data as valid but keeps checking the alignment. This is the same process as used in IEEE 802.3 [7, p. 415] and visualised in figure 6 and in more detail figure 7.

This alignment process is sensitive for header errors. However, this is not a real problem as this will start to play only when a very high BER occurs, as explained next. The module responsible for the word level synchronisation is `BLOCK_SYNC_SM` (the internal hardware is equal to IEEE version). It has two counters one for the total number of headers (sh_cnt) and one for the invalid headers (sh_invalid_cnt). The behaviour of these counters is given in figure 7. It is a continues checking process counting the total number of headers and invalid headers with two separate counters. When it reaches 64 headers (`RESET_CNT` in figure 7) it resets both counters. If in the time to count to 64 headers it counts 16 invalid headers it will mark the system as out of sync (`SLIP` in figure 7). The probability on 16 corrupted headers because of errors in 64 headers is very small. Every header as two header bit that can go corrupt, making the case on an error

2BER per header. So the probability of 16 errors in 64 headers is $f(16,64,2BER) = 3.2 \cdot 10^{-29}$ @ BER of 10^{-3} (please see appendix B for an introduction on probability). It can be safely said this will not be the limiting factor.

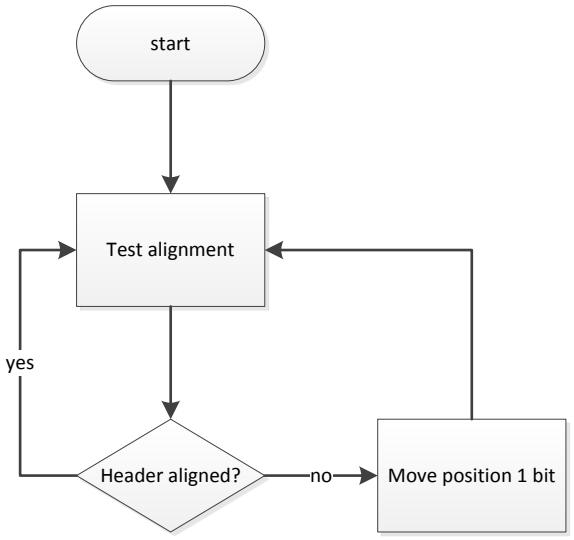

Figure 6 Word level alignment process, simplification of figure 7

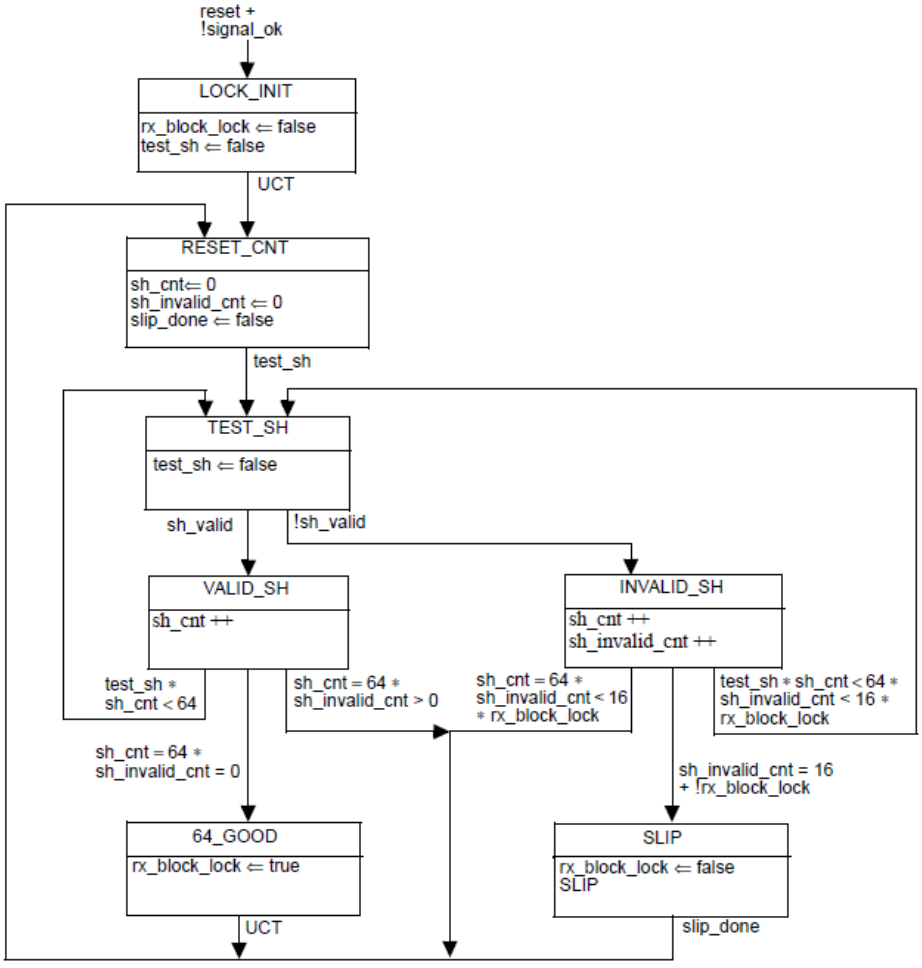

Figure 7 Lock state diagram for word alignment [7, p. 415] [4, p. 35]

To compensate for the difference in clock speeds between the receiver and transmitter, clock correction (CC) control words are used. These words do not contain any information except that they are CC words. The transmitter inserts at least 3 CC words per 10,000 cycles [4, p. 45]. In the case that the receiver runs too slow, the buffer of the receiver will slowly fill. To prevent overflows, the receiver may throw away the clock correction words, to make space again in the buffer. The other case is when the receiver runs too fast; to prevent underflow the receiver may insert clock correction words. This buffer is the elastic buffer which has two clock domains. One side is clocked at the recovered clock from the lane (tx clock) and the other clock is clocked on the derived clock of the system clock, see figure 8.

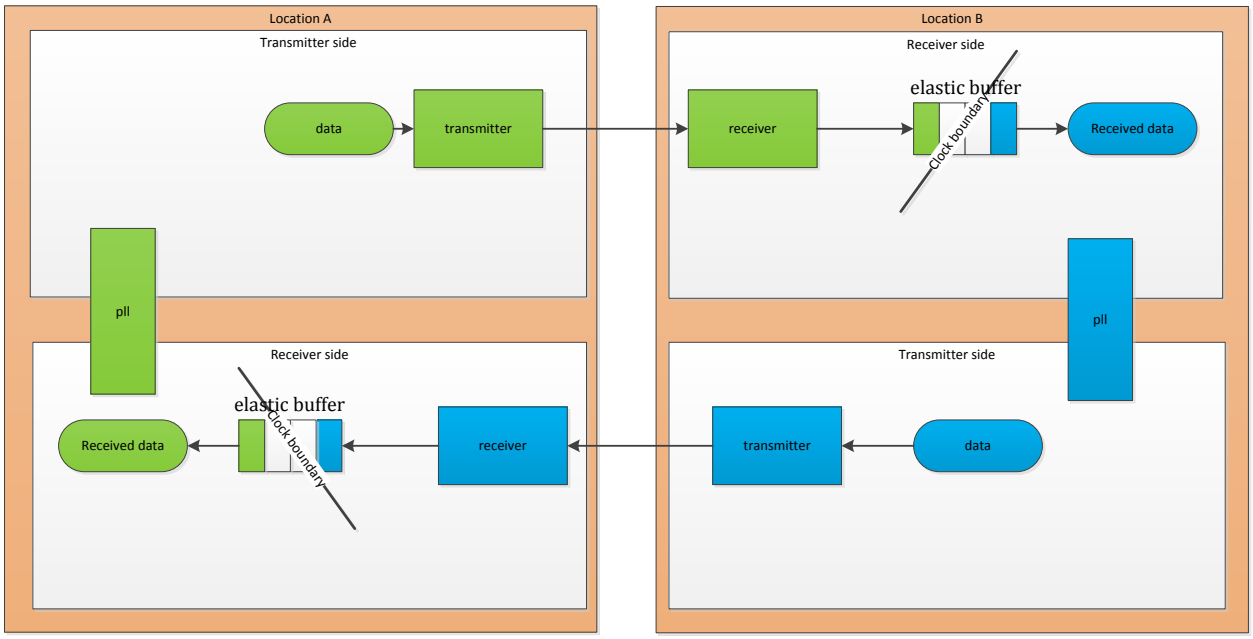

Figure 8 The different clock domains of two connected systems

The data or control might be detected as CC because of bit errors, which might thrown away a complete word. Because the buffer is overflowing. Aurora has some other control words. Most can be ignored as they will pass-through the new FEC. However, some words are not pass through the FEC because they are critical for the link. One of them is already discussed: CC. The others are GEN_CH_BOND, GEN_NA_IDLE and IDLE. The first two are always passed straight through without FEC. The IDLE is only passed straight through if the output buffer is empty. This can only happen when the system is starting. As these words are not protected by FEC they need a separate analysis what will happen if there is an error in them. This is explained in section 2.6.

2.2. Reed-Solomon

Reed-Solomon [10] (RS) is the chosen error correction algorithm. Reed-Solomon is a widely used FEC algorithm, used in CD [11, p. 8], space communication [11, p. 9] and several other applications. RS works by adding extra parity symbols to allow for detection and correction of errors in the transmission. The syntax is RS(n,k) where n is the number of symbols and k is the number of data symbols. t is the error correction strength in symbols, and can be calculated as follow:

$$t = \frac{n-k}{2}$$

So, RS(63,57) means that the RS-block is 63 symbols long, has 57 data symbols and can correct 3 errors. When it is known where the errors are, then it can correct double the amount of errors, 6 errors in this case. However, this is not used in this research, as there is no knowledge over where the error will happen.

Reed-Solomon makes use of finite field arithmetic; this results in that the number of possible values for a symbol is given by $q = p^m$. Where p is the radix and m the number of digits in a symbol. The number of symbols (n) is always 1 less then number of values.

$$n = p^m - 1$$

Instead of using all elements, some elements can be fixed to zero, allowing for any n smaller then p^m. For a complete explanation please read [11]. If radix (p) is set to 2 then m needs to be larger than or equal to 6 bits for RS(63,57).

Note that it is possible to have a non-binary Reed-Solomon code ($p \neq 2$). These non-binary RS codes can be interesting when used in combination with PAM-3/5 [12, pp. 198-202,245] which is used in Ethernet 100BASE-T4 [3, p. 101]/1000BASE-T [13, p. 17]. This way symbols can be directly mapped on a FEC instead of doing first a conversion to the binary system. For example, PAM-3 with radix $p = 3$ and a symbols size of $m = 5$ digits, has $q = 3^5 = 253$ values while the closest binary versions are $2^7 = 128$ and $2^8 = 256$.

As the system makes use of a 2-PAM for modulation plus the Reed-Solomon [14] and Aurora IPs are both an entirely binary system. A non-binary system will only result in more problems and no advantages for this design. The choice is made to stick to a binary system.

The data is also interleaved primarily to allow for more straightforward parallelisation of the RS hardware, but it also makes the RS handle burst errors better. As the errors are distributed on more RS blocks. However, this comes at the cost of a higher latency.

2.3. SCRAMBLER ERROR PROPAGATION

The scrambler is present in Aurora protocol to do DC-balancing. In the literature study [2] it was shown that the scrambler introduces extra errors. The scrambler used is the same as IEEE standard for Ethernet [7, p. 401]. The scrambler makes use of following polynomial (0,39,58). So if there is an error at position 2 in a word. Then after the scrambler there are errors at 2+0, 2+39 and 2+58. As long as m is smaller than 19 then all errors are in different symbols. If $ch * m \geq 58 + m$ holds, meaning that RS total channel width is more than or equal the max spacing between the errors plus one symbol, then it holds that it is not possible to have one pre-scrambler error occurs twice in a RS block. Given these two constraints 1 error pre-scrambler will result in 3 errors post-scrambler in 3 different RS blocks, see figure 9. As the 3 error are spaced at a fixed distance between each other all errors are different RS channels. In section 2.5 the error propagation is discussed further.

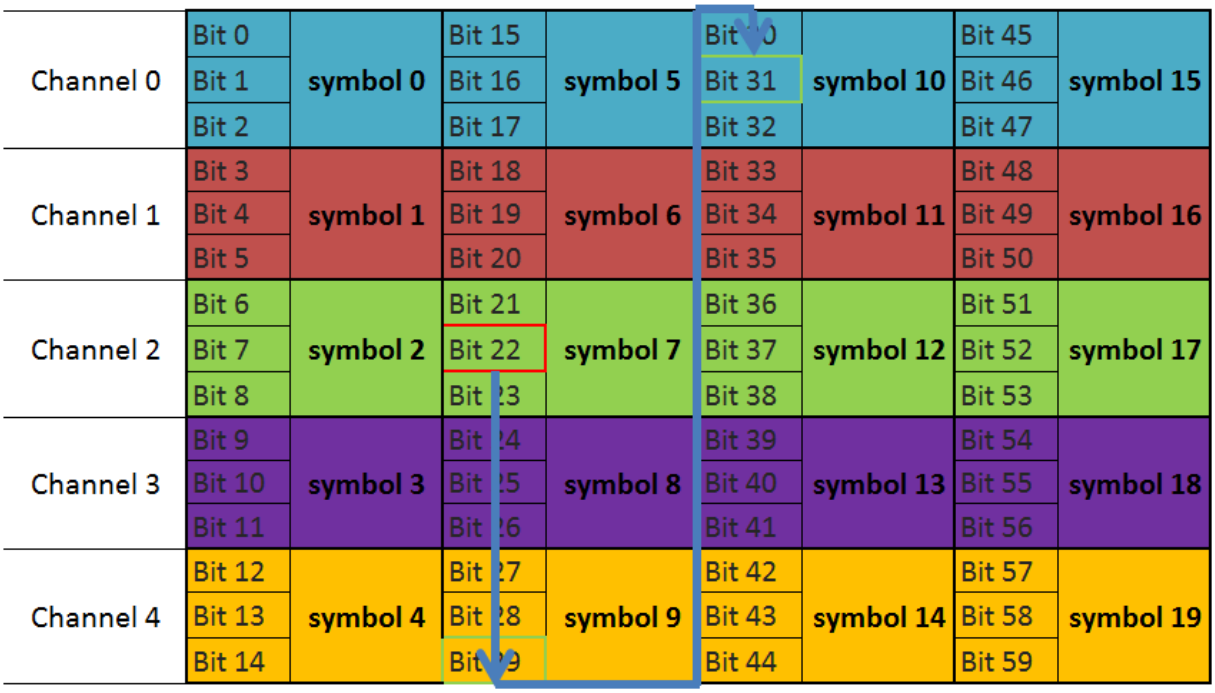

Figure 9 Error position of scrambler error within RS channels. Bit 22 is corrupt, post scrambler also bit 29 and 31 are corrupt. This example is with m=3, ch=5 and scrambler polynomial (0,7,9)

2.4. BLOCK SYNCHRONIZING

The RS blocks need to be synchronised to be able to successfully decode the RS-blocks. The header is still two bits, '01' is a data word and '10' is a control word. A new control code [4, pp. 39-40] is introduced called synchronisation (0xFF) with all the remaining bits '1', replacing the `reserved` control code. A synchronisation control word indicates the start of a new block. The synchronisation period can be found by means of searching for the smallest period for which the system aligns again with the beginning offset. GCD stands for Greatest Common Divider, which is a mathematical operation.

$$period = \frac{m * ch * n}{GCD(m * ch * n, 64)} + 1$$

The +1 is there to allow for a synchronisation word. So, for this case (table 4), this will be 2458 output cycles. See figure 10 for the construction of the packet. From bottom to top. First there are 2112 66 bits words of Aurora in one packet which can be rewritten as 65 bits. These words are storage in 1760 78 bits words by means of gearbox in the data fragment of 416 6 bits width RS blocks. The RS encoder will add the correction symbols. These RS-blocks are put 13 parallel and then after each other to form one packet. This stream is data width converted to 2457 64 bits words to enable encapsulation in a 66 bits Aurora data word. Also the packet is started with a sync word to clearly mark the beginning of packet.

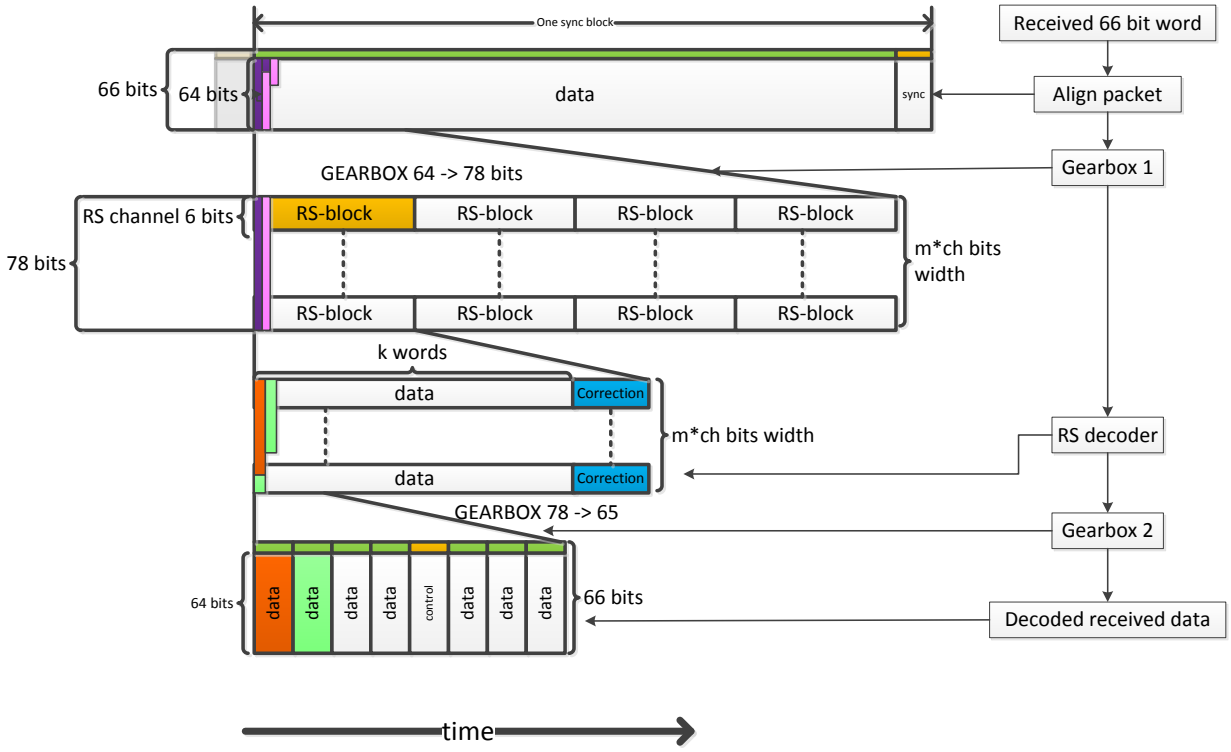

Figure 10 The structure of a packet with on the right the step in the receive process

2.5. ERROR PROPAGATION SCRAMBLER

Ideally all communication is protected by means of FEC but this is not possible as it has been chosen to do it inside Aurora. Some Aurora commands, such as Clock Correction (CC), are not allowed to be modified and are passed through without any protection. The result is that these commands need to have a separate analysis for error propagation.

> The explanation uses a smaller word, which has two header bits and six data bits. The scrambler polynomial is (0, 3, 5) versus the real one (0, 39, 58) to allow for understandable/readable illustrations.

Looking at a word pre-scrambler, the error distribution is equal for every bit. After the scrambler this probability. The header bits bypass the scrambler, so they still have the same error properties as pre-scrambler. However the data bits will pass through the scrambler; this will introduce extra errors at fixed spacing. Making the bit error distribution no longer **independent** of each other.

The probability on a single error pre-scrambler does not change; this is still about $64BER$ (for the explanation case: $6BER \approx f(1{,}6, BER)$). Only 58 (for the explanation case: 5) bits will result in also an error in the next word, these are marked blue in figure 11.

Figure 11 Blue bits are errors that propagate to the next word

This error propagation is displayed in figure 12, part B. Part A shows when an error is not propagated.

Figure 12 Error propagation to the next word. In A the error in stays within one word; in B it crosses the word boundary.

2.6. DIFFERENT TYPES OF WORD ERRORS

There is a difference between how many times a type of word is transmitted. This is weight into the error calculation by means of fixed weight per type. Note that FEC protected control is seen as data. Then the following values are found for occurrence:

- IDLE (GEN_CC, GEN_CH_BOND, GEN_NA_IDLE, IDLE) – about 1/1000 of time 0.1%
- Synchronization – ones every block: 0.08%
- Data – 100% minus above so 99.82%

To make the analysis manageable only the errors pre-scrambler are considered and the worst-case propagation is assumed. This will mean that the calculated MTBF is worse than in real life.

The MTBF is calculated by means of tracing the different types of errors and data cases. Up to two errors per word are checked. Above this it is simply assumed that the error is fatal, meaning that the data cannot be recovered. The following words can be transmitted, the weights given are based on the likelihood of occurring. In table 1 a summary is given of all error types analysed with their MTBF. See also figures 13, 14 and 15 for the traces where green means no errors, blue an error but correctable and red means non correctable.

> For referring to the errors explained in Appendix C, underlining is used. For example, e refers to error e.

Instead of only looking at BER of 10^{-12}, it is plotted in figure 16. Where the square are the calculation, which can be split in two major parts called e and m.

Figure 13 When a Data word is transmitted the following can happen based on the number of errors and the location.

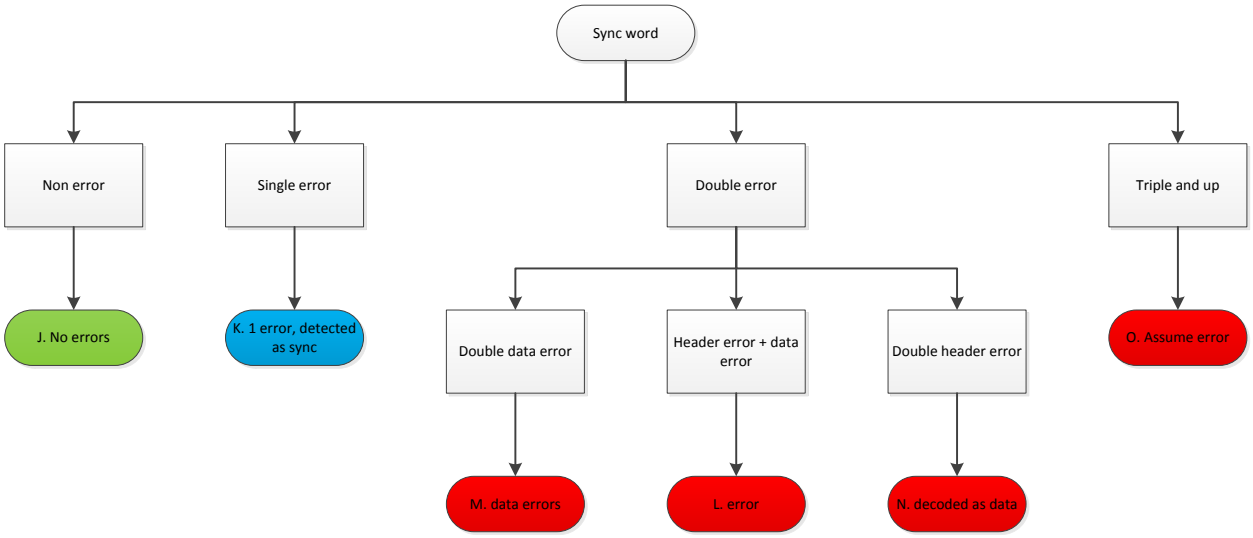

Figure 14 When a Synchronization word is transmitted the following can happen based on the number of errors and the location.

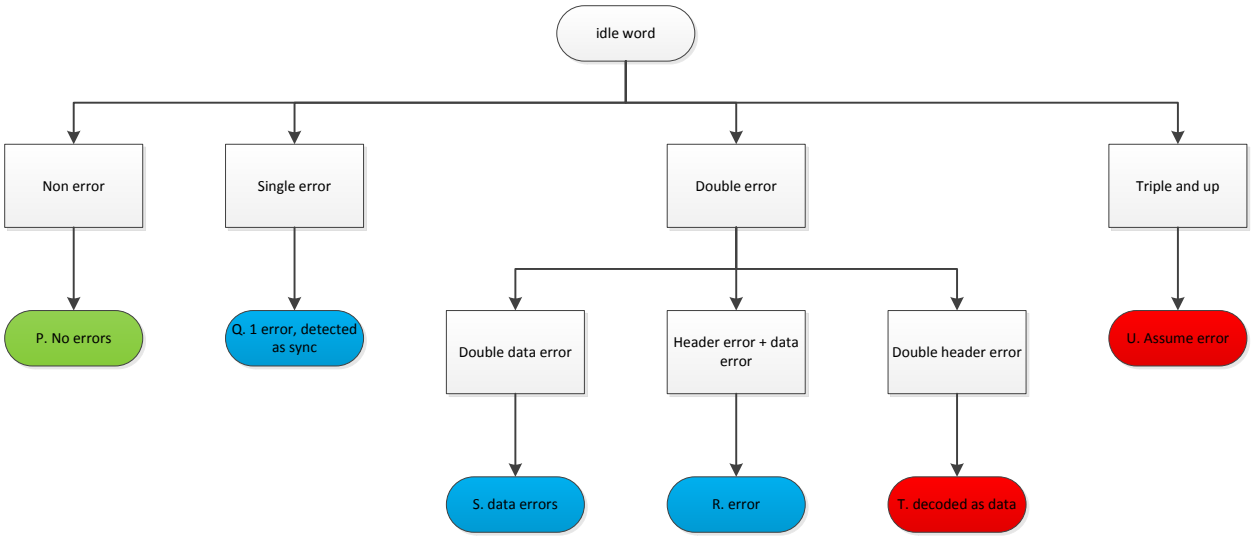

Figure 15 When an Idle word is transmitted the following can happen based on the number of errors and the location.

Table 1 Overview of all error paths. The calculation are given in appendix C.

error name	Weight	Transmitted word type	What happens	Received as	This is	percent of total errors	MTBF
a	99.82%	data	no error	data	not a problem	0.000%	
b	99.82%	data	single data error	data	correctable by RS	0.000%	
c	99.82%	data	double data error	data	correctable by RS	0.000%	
d	99.82%	data	triple error	Error	unrecoverable	0.000%	2.14E+22
e	99.82%	data	header error 1 bit	idle	unrecoverable	99.925%	1.37E+12
f	99.82%	data	header error 1 bit	sync	unrecoverable	0.000%	1.83E+19
g	99.82%	data	double error header	idle	unrecoverable	0.000%	4.72E+22
h	99.82%	data	double error header	sync	unrecoverable	0.000%	5.91E+29
i	99.82%	data	double error header	ignore	unrecoverable	0.017%	8.01E+15
j	0.08%	sync	no error	sync	not a problem	0.000%	
k	0.08%	sync	1 error	sync	not a problem	0.000%	
l	0.08%	sync	1 header + 1 data error	error	unrecoverable	0.002%	7.68E+16
m	0.08%	sync	2 data errors	error	unrecoverable	0.056%	2.44E+15
n	0.08%	sync	2 header errors	data	unrecoverable	0.000%	9.83E+18
o	0.08%	sync	triple error	error	unrecoverable	0.000%	2.62E+25
p	0.10%	idle	no error	idle	not a problem	0.000%	
q	0.10%	idle	1 error	idle	not a problem	0.000%	
r	0.10%	idle	1 header + 1 data error	idle	not a problem	0.000%	
s	0.10%	idle	2 data errors	idle	not a problem	0.000%	
t	0.10%	idle	2 header errors	data	unrecoverable	0.000%	8.00E+18
u	0.10%	idle	triple error	Error	unrecoverable	0.000%	2.14E+25

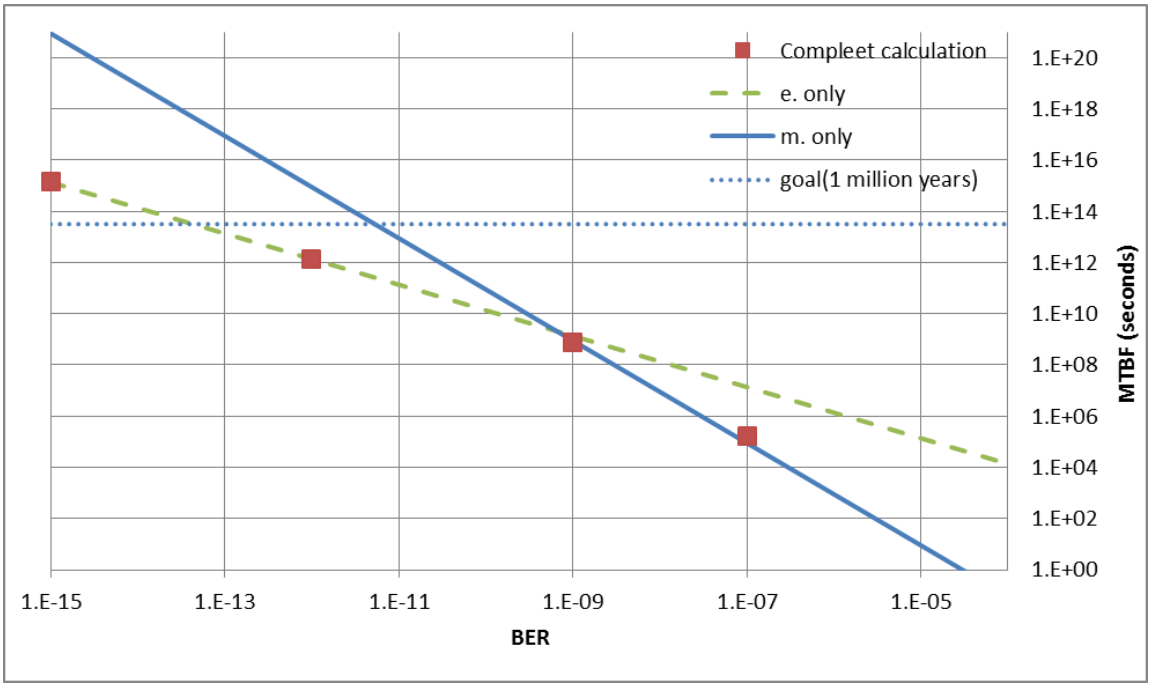

Figure 16 Visualization of approximation of MTBF for given BER

2.7. REED-SOLOMON CORRECTION STRENGTH

For the correction of the data errors, Reed-Solomon is used. These are only the errors b (single) and c (double) from table 1. The analysis for failure is done on a synchronisation block size. It is assumed that all errors occur in the same RS block in the synchronisation block (see figure 17, all errors are in the yellow block), while in the real world they are distributed over all blocks. However, this allows for a more straightforward calculation at the cost that the result is very conservative. The scrambler amplifies the errors but because the way the errors are distributed over the RS blocks it is not possible to have more than 1 error post scrambler in an RS block because of 1 error pre-scrambler, as shown in section 2.3. In the next table all possible options are given. Only when the total number of error is greater than $\frac{(n-k)}{2}$, will it result in a decode error.

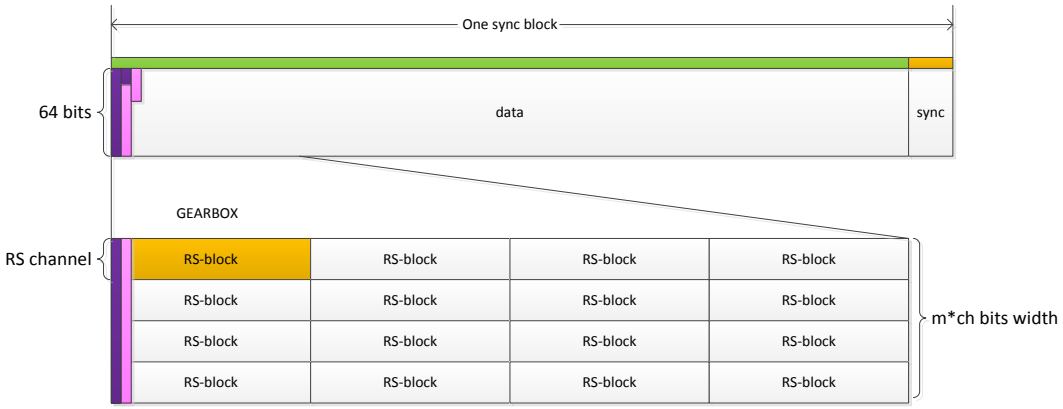

Figure 17 Sync block packet, assuming that all RS are in one block (yellow)

In table 2 the different options are given, red are the cases when the number of errors is greater than what one RS block can correct. Because the probability on failure because of none protected logic is much large then probability on not being able to correct with FEC is small. It can be simple said that error correction strength is always enough to correct. Note that this is only the case for a low BER.

Table 2 MTBF for given combination of single and double errors. Red is when the error correction streght needed is more than maximum (4).

Type of error		number of	Probability on	
single	double	errors	error in a block	MTBF
0	0	0	1.00E+00	
0	1	2	4.95E-18	
0	2	4	1.23E-35	
0	3	**6**	2.02E-53	9.72E+47
1	0	1	1.57E-07	
1	1	3	7.79E-25	
1	2	**5**	1.93E-42	1.02E+37
1	3	**7**	3.18E-60	6.18E+54
2	0	2	1.24E-14	
2	1	4	6.12E-32	
2	2	**6**	1.52E-49	1.30E+44
2	3	**8**	2.50E-67	7.86E+61
3	0	3	6.47E-22	
3	1	**5**	3.21E-39	6.13E+33
3	2	**7**	7.94E-57	2.48E+51
3	3	**9**	1.31E-74	1.50E+69
4	0	4	2.54E-29	
4	1	**6**	1.26E-46	1.56E+41
4	2	**8**	3.12E-64	6.31E+58
4	3	**10**	5.14E-82	3.82E+76
5	0	**5**	7.98E-37	2.46E+31
5	1	**7**	3.95E-54	4.97E+48
5	2	**9**	9.78E-72	2.01E+66
5	3	**11**	1.61E-89	1.22E+84

2.8. SUMMARIZING

Instead of considering all the calculations only the major contributions are considered. Then for a BER of 10^{-9} and lower e is the major component above this m will be the major contribution. In figure 16 this is visualised. Note that the calculation is made on the assumption that $BER \ll 1$, so when using BER of 10^{-5} or even higher may not give the correct answer.

As can be seen from the figure 16, the MTBF of 1 million years at a BER of 10^{-12} is not reached. The best that can be reached at a BER of 10^{-12} is an MTBF about 40,000 years. Remember that this is the MTBF that is unrecoverable meaning that the next layer needs to handle this. The data will be marked as corrupted in these cases as the FEC cannot successful decode the stream.

However, in a very rare case it can be marked as correct [15]. For this at least one error more than it can correct needs to happen so t+1.

In the literature report [2] it was concluded that it is possible to design a link with an MTBF of 1 million years at a BER of 10^{-8} however here it is shown that it is not possible. At a BER of 10^{-8} only an MTBF of 14 weeks is reached. This difference comes from the fact that the design is much more thoroughly analysed compared to the literature study.

3. DESIGN AND BUILD

The Aurora IP [14] needs to be modified to allow the FEC to be integrated. The goal is to have a good reusable IP and minimal changes in the Aurora IP. The choice was made to modify the encoding and decoding modules as these are close to the transmitter/receiver, on the other hand there is no need to care about the synchronising on word-level

In the next sections, the design is discussed starting with the selection processes for the RS and channel width. After which the different part are discussed such as gearbox, FIFO and scrambler. The chapter is ended with resources used and some suggestion for improvement of the design.

3.1. DESIGN PARAMETERS

Two linked excel sheets are available which contain the design parameter checks and MTBF calculations of the previous chapter. In the previous chapter, final design parameters are assumed. Table 3 gives a list of design checks to make sure the design can work. The design checks can be defined in a number of categories. Reed-Solomon requirements make sure that it is a valid RS code. RS IP requirements are requirements that are related to the IP used such as limited symbol size and there are some recommendations which should help with making a synthesizable design which meets the timing constraints.

Table 3 Design checks

	Requirement type	**Note**
$\boldsymbol{n < 2^m}$	Reed-Solomon requirement	
$\boldsymbol{m =< 12}$	RS IP requirement	Limit symbol size to 12 bits
$\boldsymbol{(n-k)\%2 = 0}$	Reed-Solomon requirement	
$\boldsymbol{k < n}$	Reed-Solomon requirement	
$\boldsymbol{n < 4096}$	RS IP requirement	
Processing power	System requirement	The RS-processing power is enough to keep up with data stream.
Overflow check	System requirement	Input data rate is less than the maximum data rate
$\boldsymbol{m \cdot ch < 100}$	Channel width recommendation	Wide channel can result in problems with timing constraints withing the gearboxes.
$\boldsymbol{t < 5}$	Correction strength recommendation	High correction strength is not needed.

As the unprotected words are the primary source of failure to decode, it does not help to increase the error correction strength to high values. Good choices for t are 3 and 4. Also, a large t results in more hardware use as the error correction becomes more complicated. For the choice of the number of symbols one should remember that the overhead is given by $\frac{(n-k)}{k} = \frac{2t}{n-2t}$. So, a small n (7 or 15) will mean a lot of overhead for given t, while a large n will result in longer latency. This latency [14, p. 25] is given by

$$latency = 2t^2 + 9t + n + 14$$

This longer latency is not a problem. That is why the choice of n is mainly based on choosing a practical value for m and limiting the overhead. Remember that in section 2.2 following equation was given:

$$n = p^m - 1$$

For example, m equal to 8 it will result in a byte width symbol size, which fits precisely 8 times in a 64 bits channel. In this research m is set to 6, to have not a too large n. Later on, it was shown that a better choice is m is 8 as then only the 65:64 input gearboxes is needed. In section 3.2 the gearbox is explained.

The number of parallel channels is chosen based on the throughput needed. As the limiting factor is the decoding process of Reed-Solomon, it is enough to only check at the decoding side for throughput. The Reed-Solomon IP documentation [14, p. 24] gives the following formula for calculating throughput. For this first the processing delay (PD) (cycle time) need to be calculated.

$$PD = 2t^2 + 9t + 3$$

Note that the cycle time only depends on t. So, for t is 4, PD is 71. If $PD \leq n$ then throughput is $TP = f * m$ where f is clock frequency. If $PD > n$ then the throughput is limited to $TP = \frac{n}{PD} * f * m$. This is the throughput of a single RS decoder. The n is chosen as the maximum n for $m = 6$ which is 63 then the decoder can process $TP_o = \frac{Number\ of\ output\ symbols}{Time\ it\ takes\ to\ process\ it} = \frac{n}{PD} = \frac{63}{71} \approx 0.89$ output symbols per clock. The maximum n is chosen because it will result in the lowest overhead for the given m.

The Aurora part will produce words of 66 bits width as it is known that the header can only have two states: '01' and '10'. It can be reduced to one bit. Resulting in a total word size of 65 bits or 10.83 6-bit data symbols. The RS IP can handle $TP_i = \frac{Number\ of\ input\ symbols}{Time\ it\ takes\ to\ process\ it} = \frac{k}{PD} = \frac{55}{71} = 0.77$ data symbols per clock. Because of the overhead it is impossible to process all input words because the chain adds overhead. So, to calculate the number of parallel channels needed, the output will be leading. The output needs to produce one word per clock. An output word has 64 bits of data or 10.67 symbols per clock. Or in other words, the RS encoder needs to produce at least 10.67 symbols per clock. With this the number of channels to keep up with the output can be calculated.

$$\frac{output\ requires}{channel\ capacity(TP_o)} = \frac{64/m}{n/PD} = \frac{64/6}{63/71} = 12.02 \qquad \frac{n}{PD} < 1$$

$$\frac{output\ requires}{channel\ capacity} = \frac{64/m}{1} \qquad \frac{n}{PD} \geq 1$$

Because partial channels are not allowed at least 13 channels of m = 6 with n = 63 and t = 4 are needed. The number of channels can be lower because the output is not required every cycle because the GTH gearbox will pause the signal for one cycle every 33 cycles. Then the calculation becomes:

$$\frac{64/6}{63/71} * \frac{32}{33} = 11.66 \qquad \frac{n}{PD} < 1$$

$$\frac{64/m}{1} * \frac{32}{33} \qquad \frac{n}{PD} \geq 1$$

Or 12 full channels, as this was later discovered it is not used in this design.

The data rate (DR) is set to 8.25 Gbps as this is the rate used in the example project earlier by Thales. So, in summary, the following design parameters are used.

Table 4 Configuration parameters

t	Maximum number of symbols one RS-block can correct	4
m	Symbol width	6 bits
n	Number of symbols in a RS-block	63
ch	Number of parallel RS-channels	13
k	Number of data symbols in a RS-block	55
DR	Data rate	8.25 Gbps

In the next section, the different steps of the encoding and decoding processes are described.

3.2. GEARBOX

As the input width does not match with the output width and the Reed-Solomon width there is the need to convert the bus width. The following gearboxes are needed. The input stream is 66 bits of which 64 bits are data and 2 header bits. As the header bits are XOR of each other only one bit is left meaning 64 bits data and 1 header bit. These 65 bits go to the RS encoder which has an input width of $ch * m = 78$ bits. The resulting bit stream needs to be encoded in 64 data bits of the data word. This means that the bus has two times a bus width conversion at the encoder: 65:78 and 78:64. On the decoder side exactly the opposite happens: 64:78 and 78:65.

Ideal is to reuse an existing IP for this. The options are limited as most gearbox IPs found have fixed ratios which do not match with this design. Because of this, a custom gearbox module is written. The module consists out of a large input shift register which shifts in an input word. By means of a large mux the output word is selected. This module is fully parameterized for any gearbox ratio. See figure 18 for example configuration of design.

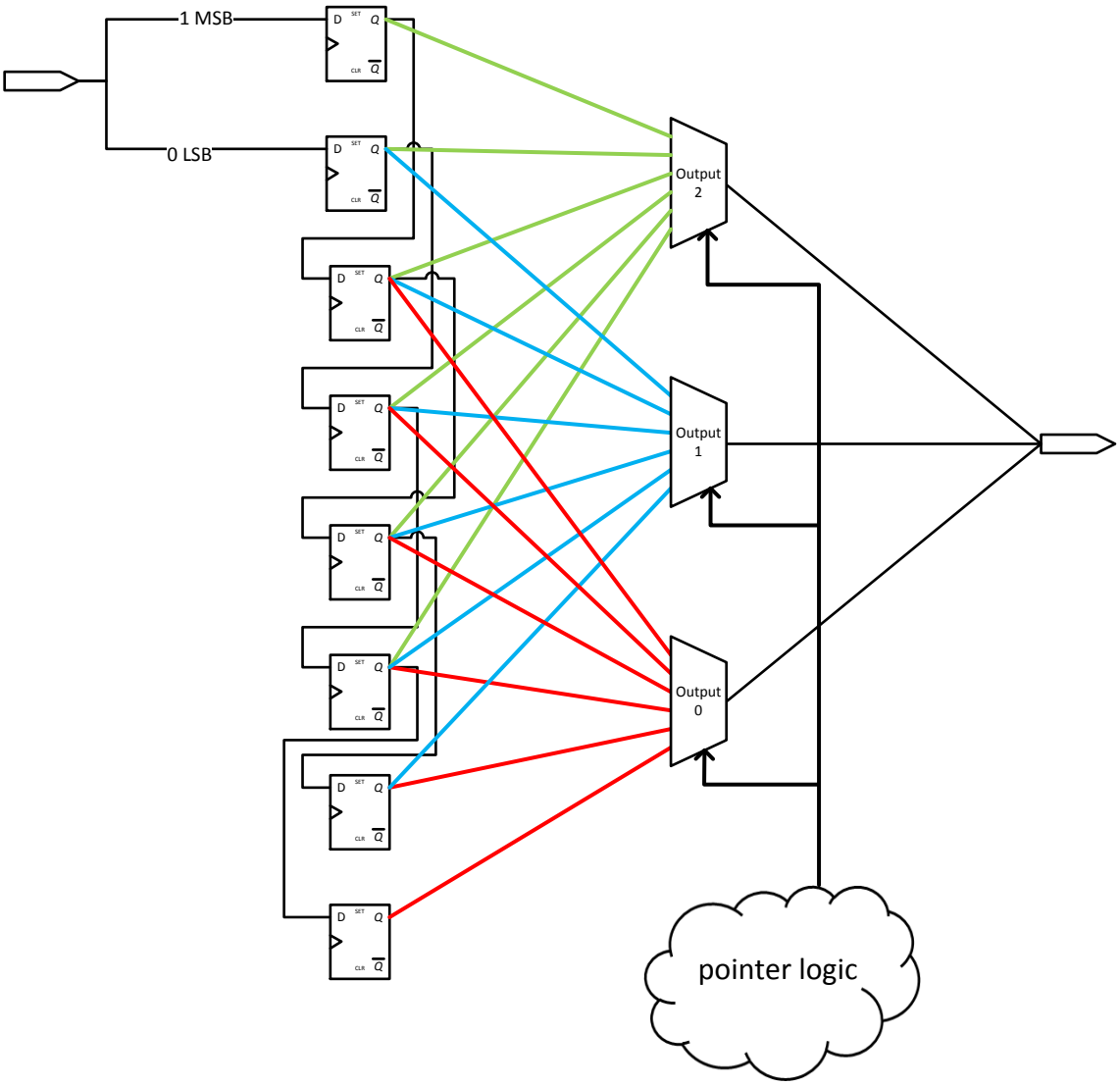

Figure 18 : Gearbox design for 2:3 with 4 input words width shift register. Not all control logic is shown.

These shift registers usually are not a problem because there is dedicated logic for [16, p. 47]. The MUXs will result in a lot of hardware as they connect almost any bit from the shift register to the output. As the shift register has a size of four times the input it is about 256 bit of which about ¾ can be selected. The mux has then a ratio of 192:1 using this [17, p. 19] the estimate LUT6 count is 17*3+1=52. Times the number of output bits it will be around 3000-4000 LUT6's depending on the chosen gearbox ratio.

One part which is not taken into account is that the pointer can only have a limited number of values. In the case of 65:78 the pointer is always adding or subtracting by a multiple of 13 (5*13=65 and 6*13=78). This means that for the pointer holds that

$$offset \,\%\, 13 = 260 \,\%\, 13 = 0$$

260 is the initial offset. Reducing the number of mux inputs by factor 13 to a merely 14. Then estimated 4 LUT6 per output is needed or 312 LUT total. The pointer beheavor is not include in the design as it was later discovered.

3.3.FIFO

The consumption and production is match to each other but the rate of production is not constant per clock cycle but is constant in the long run. So buffers are needed to handle this varies in consumption and production. To keep it simple a FIFO is used. The whole system has to clock out 1 word per cycle and if there is a CC or synchronisation then it will pause the whole encoding/decoding system. Defining the system happens on output rate. r_{in} is defined as the number of input words per clock and w_{in} is the input width (65 bits). After the first gearbox the r_{gb1} is

$$r_{gb1} = r_{in}\frac{65}{78}$$

Then it will pass through RS encoder. The rate past the RS is

$$r_{rs} = r_{gb1}\frac{63}{55}$$

Then it will be pushed to the second gearbox.

$$r_{final} = r_{rs}\frac{78}{64}$$

It is given that r_{final} needs to be 1 so from this the other rates can be calculated. These rates of course also hold at the decoding side but then in reverse order.

Table 5 Word rate

	rate	width	bit/cycle	Symbols/cycle
r_{in}	0.86	65	55.87	9.31
r_{gb1}	0.72	78	55.87	9.31
r_{rs}	0.82	78	64.00	10.67
r_{final}	1.00	64	64.00	10.67

The first version used a fixed firing moment however this introduces problems at the moment that the system is paused. As generation becomes out of sync and because the fire rate ratios must be exact large counters are needed. Also it makes the design inflexible as the fire rate need to programmed. So instead the system is free running allowing to fill the output buffer as soon as possible. Also instead of using an own designed FIFO the Xilinx coregen FIFO [18] is used. With it the busses are changed to AXI-S [19] such that backpressure can be applied. Now the system consists out of following different parts: Gearbox, FIFOs, Reed-Solomon and edge logic.

In figure 19 the system is given. This system also includes a scrambler which will be introduced in section 3.5. The edge logic will be discussed in the next section.

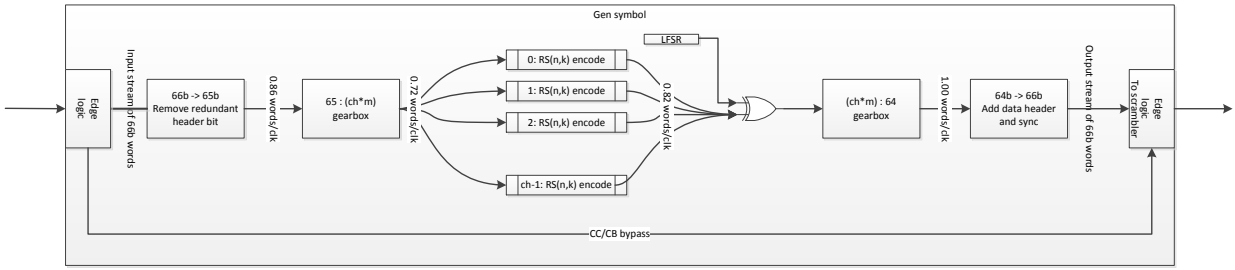

Figure 19 Encoding FEC system with edge logic

3.4. EDGE LOGIC AND FLOW CONTROL

The Aurora IP supplies a 66 bits word it wants to transmit. In the edge logic, the second header bit is removed as it does not contain useful information. Then the word is conditional shifted into the FIFO. When the buffer is half full only non-idle words are accepted and Idle words are not. If the buffer is not half full then it will accept all words. As the FIFO is read at a rate lower than the input rate it will never underflow. The overflow is prevented by means of limiting the time the IP can take in AXI-S data. This is done in `TX_LL_CONTROL_SM` module only allowing 50% of the time to accept an AXI-S data.

The module `encoding_timing` will generate the timing and synchronisation words. In `sym_gen` the synchronisation word is injected also the CC words will bypass here the FEC.

At the receiving side `sync_dec` will synchronise the system providing the synchronisation words. The decoded data from the FEC is always put on the output of the `sym_dec` block, if no data is available it will put out an idle.

3.4.1. MULTILANE DATA SYNCHRONISATION

Without the FEC system in place Aurora will take care of the multi-lane synchronisation. However, because the way the FEC works this no longer works. Consider the case with four channels/lanes which are connected via a 32 bytes width AXI-S bus and data is a multiple 8 of bytes. Consider the case where first 8 bytes are transmitted and then 32 bytes. Before this data has been transmitted continues, so FIFOs are almost full. First 8 bytes will be transmitted on one channel with data word (channel 0) and all other channels (1/2/3) have idle words. The FEC system will take in the data word on channel 0. On the other channels the idle words are ignored as the input buffer is full enough. Next comes the 32 bytes packet, meaning that all channels will have data words which will be shifted in. At the receiver ends the data becomes corrupted. As the idle symbols are not transmitted the data on channels 1/2/3 get shifted, see figure 20. The solution for this is to use placeholders, this is done by means of transmitting an Aurora SEP words with no data. These SEP words are always clocked in, in the same way as data. This modification is done in `TX_LL_CONTROL_SM` and `RX_LL_DATAPATH`.

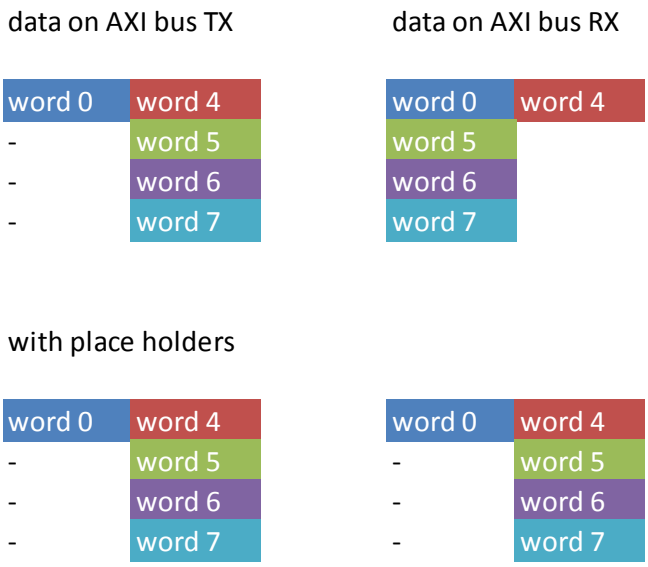

Figure 20 Multi-lane synchronisation problems

3.5. SCRAMBLER

During the design phase it was discovered that the data cannot be seen as uniform random distributed as it contains a lot of idle words. But in the calculation in chapter 2 it is assumed that it is uniform. To make the data uniform it is xor-ed with a pseudo seeded random data stream. The data is again xor-ed with the same pseudo seeded random data at the receiver side resulting in the original data.

The generation of pseudo seeded random is done with a Linear-Feedback Shift Register (LFSR) of 13 bits width. Six LFSR are placed in parallel with different starting values and the same construct/feedback polynomial. LFSR needs to have a period of at least 2457, so 12 bits maximal length LFSR is enough. As our channel is 78 bits width, 13 bits is more practical. The feedback polynomial chosen is 0x1C80[1] as illustrated in figure 21. The choice for the feedback polynomial is arbitrary; any 13 bits maximal length feedback polynomial is possible even 12 bits is fine. Smaller will also work but then the system has a repeating pattern.

Compare to the scrambler that is already present this one has not the problem with error amplification. This difference comes from the fact that the new scrambler is not a self-synchronising scrambler.

[1] The Maximal Length LFSR Feedback Terms are from Philip Koopman. https://users.ece.cmu.edu/~koopman/lfsr/index.html

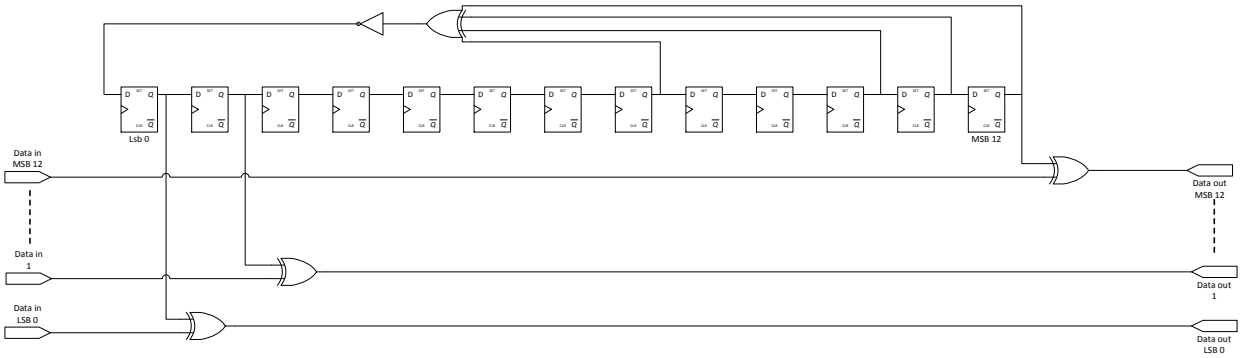

Figure 21 LFSR used for data scrambler. Synchronization and pause logic is not shown. Be aware that the registers are aligned from LSB at left to MSB at right.

3.6. LATENCY

Adding the FEC introduces some delay in the pipeline. The extra delay consists out of gearboxes, RS and FIFO's. The gearboxes buffers 4 samples, there are 2 gearboxes at both sides. The RS encoder has a delay of 3 cycles and the FIFO also has a couple of cycles delay because data is buffered. The first TX FIFO is controlled to be half full. The gearboxes fire at fixed moments in time. So after start-up phase: FIFO 1 half full, FIFO 2 is nearly empty because the RS encoder consumes as soon as possible. The second gearbox starts firing precisely 48 cycles after the first. Moreover, the output 5 cycle later. So the total number of bits between output and first gearbox is 53 cycles at 55.87 bits/cycle or 2.96 kb latency. On RX side data is moved to output as soon as possible so only a few cycles are wasted on FIFO and gearbox and it is estimated to be 10 cycles. The primary delay comes from RS decoder which is 145 cycles [14, p. 25]. In table 6 the latency is given, the total is 16 kb or 2 μs.

Table 6 Latency

FIFO 1 ~16 sample		16*55.87=894 bits	
Gearbox 1 to output		2.96 kb	
	TX latency		3.9 kb
RS latency		145*78=11.3 kb	
Other latency		0.8 kb	
	RX latency		12.1 kb
Total latency			16.0 kb

3.7. RESOURCES USE

The complete system is built on a Kintex UltraScale FPGA KCU105 Evaluation Kit. The design consists out of 4 Aurora lanes at 8.25 Gbps with a test data generator plus checker (this is default example design for Aurora from Xilinx). The system can be configured in a number of loopback configurations to allow for external or internal loopback. In table 7 the system resource usage is given without FEC and chipscope[2]. In table 8 the hardware used with FEC and chipscope is given.

[2] Xilinx debug logic allows to look into the chip at runtime

Table 7 Hardware use for 4 full duplex aurora channels without FEC and chipscope

Resource	Utilization	Available	Utilization %
LUT	2973	242400	1.23
LUTRAM	398	112800	0.35
FF	7559	484800	1.56
BRAM	4	600	0.67
IO	19	520	3.65
GT	4	20	20.00
BUFG	4	480	0.83

Table 8 Hardware use for 4 full duplex aurora channels with FEC and chipscope

Resource	Utilization	Available	Utilization %	Increase
LUT	66719	242400	27.52	+2237%
LUTRAM	5520	112800	4.89	+1397%
FF	85735	484800	17.68	+1133%
BRAM	61	600	10.17	+1517%
IO	3	520	0.58	
GT	4	20	20.00	
BUFG	5	480	1.04	

The major change is in `aurora_lane_*_i` from around 270 LUT and 250 FF to 15204 LUT and 18098 FF. The increase is meanly because of RS. In table 9 and table 10 the hardware use is given for 1 Aurora lane for encoder and decoder site.

Table 9 RS encoder hardware use

Resource	Utilization(1 encoder)	Utilization(13 encoders)	Utilization(13) of KCU105
LUT	108	1404	0.58%
FF	131	1703	0.35%

Table 10 RS decoder hardware use

Resource	Utilization(1 decoder)	Utilization(13 decoders)	Utilization(13) of KCU105
LUT	441	5733	2.4%
FF	521	6773	1.4%
BRAM	0.5	6.5	1.1%

The remaining hardware use is mainly for the FIFO and the GEARBOX. So per channel there is about 15,000 LUT (6.2%), 18,000 FF (3.7%) and 6.5 BRAM (1.1%) of extra hardware. The remaining difference in BRAM use is because of chipscope. For comparison the logic use for one channel would roughly fit inside Xilinx Artix 35 (XC7A35T) or a Spartan 6 45 (XC6SLX45). Or it will be about 10% in the smallest device in the Ultrascale+ Kintex serie (XCKU3P).

Hardware use can be reduced by choosing optimal channel width ($ch * m$) preferred 64 or 65 bit, reducing t from 4 to 3 or 2 and have the logic run on its own clock domain at maximal speed reducing the need for parallelization.

3.8. ALTERNATIVES AND IMPROVEMENTS

As with all research some choices have to be made on limited knowledge, while later on it shows that it is not the best case. So also, in this case, there are better design choices. It was already shown in section 2.8 that this design does not meet the design requirements. The main reason for this is the unprotected control. A solution could be to move the FEC as close as possible to the transceiver. Desirable just after the synchronising of 66 bits words clocked at transmitter clocked. This means that also CC and CB words are protected. The only part that still can fail is the synchronising however the probability on this is very small.

Also, the hardware use can be improved by choosing the optimum design parameters by means of extensive search for parameters. For example m=8, ch=8, t=3 will remove move half of gearboxes and quarter of FIFOs. In table 11 number of different configuration are given with quick estimate for size. Note that the estimate does not take in account the optimisations op synthesis tooling. Current design has an estimate of 23,269 LUTs while synthesis results are 15,204 LUTs.

Table 11 Quick estimate without FIFO for different configuration

ch	n	t	Data overhead	total LUT6	
13	63	4	14.55%	23,269	Current
13	63	3	10.53%	22,645	
13	31	3	24.00%	12,489	
8	255	3	2.41%	12,417	
6	4095	3	0.15%	21,587	
9	127	3	4.96%	19,951	

4. TESTING

As explained in the previous chapters the calculation of MTBF is not that trivial, meaning that testing is needed to validate these calculation methods. Consider the error e, there is one error in the header, which has a probability of $2BER$ plus the data needs to match with the idle symbols, there are $64 * 63 * 62 * 61 * 60 * 59 = 5.398 \cdot 10^{10}$ data words that match enough with idle word of total $2^{64} = 1.845 \cdot 10^{19}$. So, the probability that this error occurs is $5.85 \cdot 10^{-21}$ per word at BER of 10^{-12}. Testing with a higher BER of 10^{-4} will increase this to $5.85 \cdot 10^{-13}$ per word. Translating this to data, 113 Tb need to be processed to have on average one error of this kind.

Instead of working with MTBF, Mean Data Between Failure (MDBF) is used as it is independent of data rate (DR).

$$MDBF = MTBF \cdot DR = \frac{1}{\frac{failure\ probability}{runlength}}$$

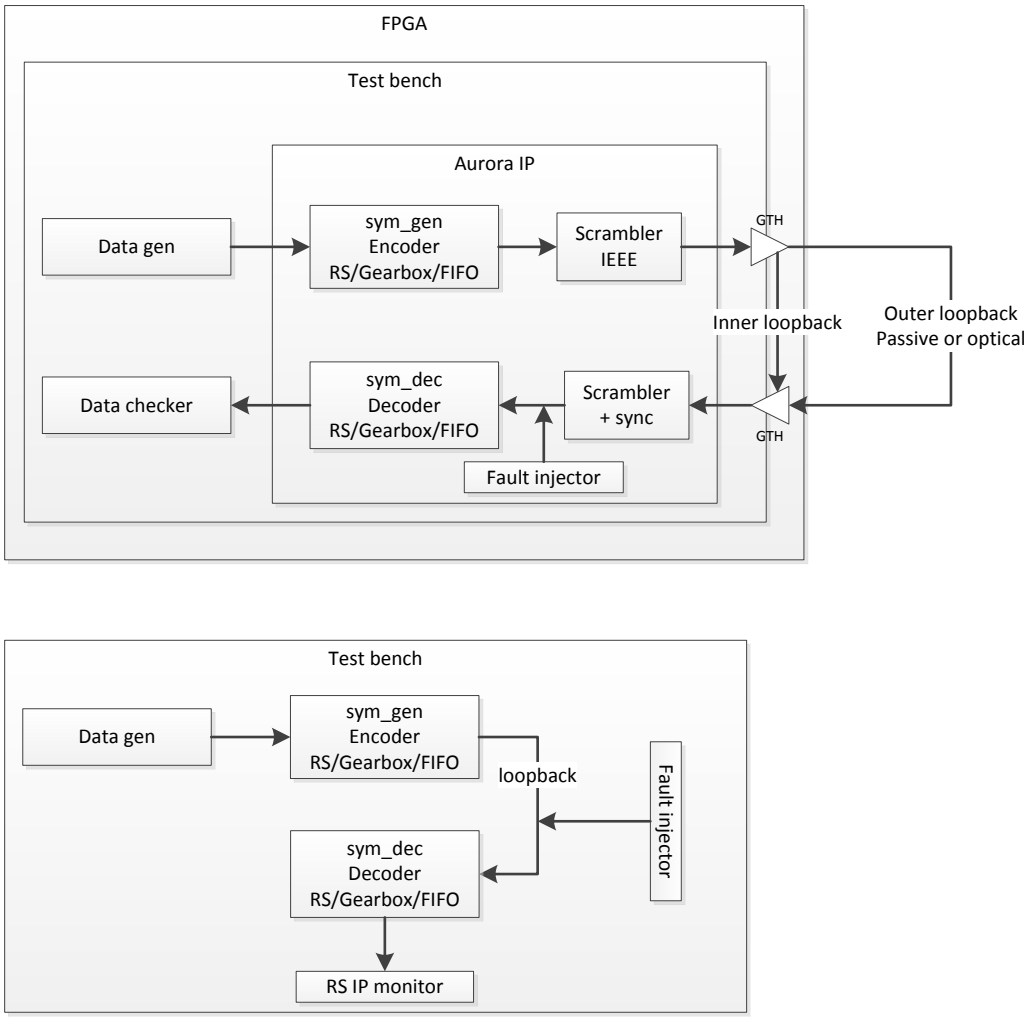

Figure 22 above hardware test environment and below the simulation setup

4.1. SIMULATION BER TESTING

A testbench which contains `sym_gen` and `sym_dec` is used to run a test for BER, see figure 22. The components are connected via a BER injector. The injector uses the Verilog function random:

```
output_word2[i] = output_word[i] ^ (($random % iBER==0 && error_mode == 1));
```

Where `iBER` is the mean data between failures (MDBF): $\frac{1}{BER}$ and `error_mode` is equal to 1 when BER testing. The Verilog standard states that `$random` is uniform distributed [20, p. 313]. As long as the remainder operator is a power of 2 it will keep the uniform distribution. Otherwise there will be a small non-uniformity. The tests are run on different BERs see the result in figure 23 and Appendix A. There is a very sharp edge from decodable to un-decodable. This can be explained by the fact that the RS can correct up to 4 words meaning that it cannot correct 5 errors. $f(5{,}63 * 6, BER) = \cdots BER^5$ which is 5th order behaviour. This is shown as the best case in figure 23. This case is solely calculated on RS decoder assuming all errors are in the data. The worst-case calculation m and e are from chapter 2.

Because a very high BERs (10^{-4} and up) are used for simulating, the assumption of low BER ($BER \ll 1$) is no longer valid meaning that the calculation may show a calculation error. The worst case (m and e) predict a lower error rate then simulated. Starting from 10^{-5} it should match again, but there are no simulations at this location. Because the probability on failure is very low in the order of 10^{-9} at BER of 10^{-5}. This cannot be tested using simulation because of the long simulation time. Only look at the best case, it matches very well. If the case with and without scrambler are compared then a 3 times lower BER is needed to keep the same failure probability with scrambler compared to without. This matches with the prediction made in the literature report [2].

There is a problem with calculation of the m part as it assumes that it always will fail if there is a synchronisadtion error while the system can handle a number of sequel synchronisation errors. So to cover these errors the run length is increased to 1.72 Mb compared to the standard length of 172 kb. It is expected, for these runlengths, that the long run has the following failure rate.

$$f_{long} = 1 - (1 - f_{short})^{10}$$

This is the probability that at least one run out of 10 has a failure. For readability the formula is rewritten the other way around and it is added to figure 23, to see straight what the difference is (long run is with scrambler and short run without scrambler).

$$\left(1 - f_{long}\right)^{\frac{1}{10}} = 1 - f_{short}$$

$$1 - \left(1 - f_{long}\right)^{\frac{1}{10}} = f_{short}$$

Using the Wilson score interval [21] with 95% confidence level, an approximation can be made for the failure rate for a complete synchronisation block of 172 kb. This gives an upper and lower bound, within these bound the real failure rate should lay with a 95% confidence. These results are given in table 15, appendix A. Note that the spread is not the same for all points. This is because the number of runs is not the same for all measurements. The Wilson method is preferred over Normal approximation interval [21] as it also works when the p is close to 0 or 1.

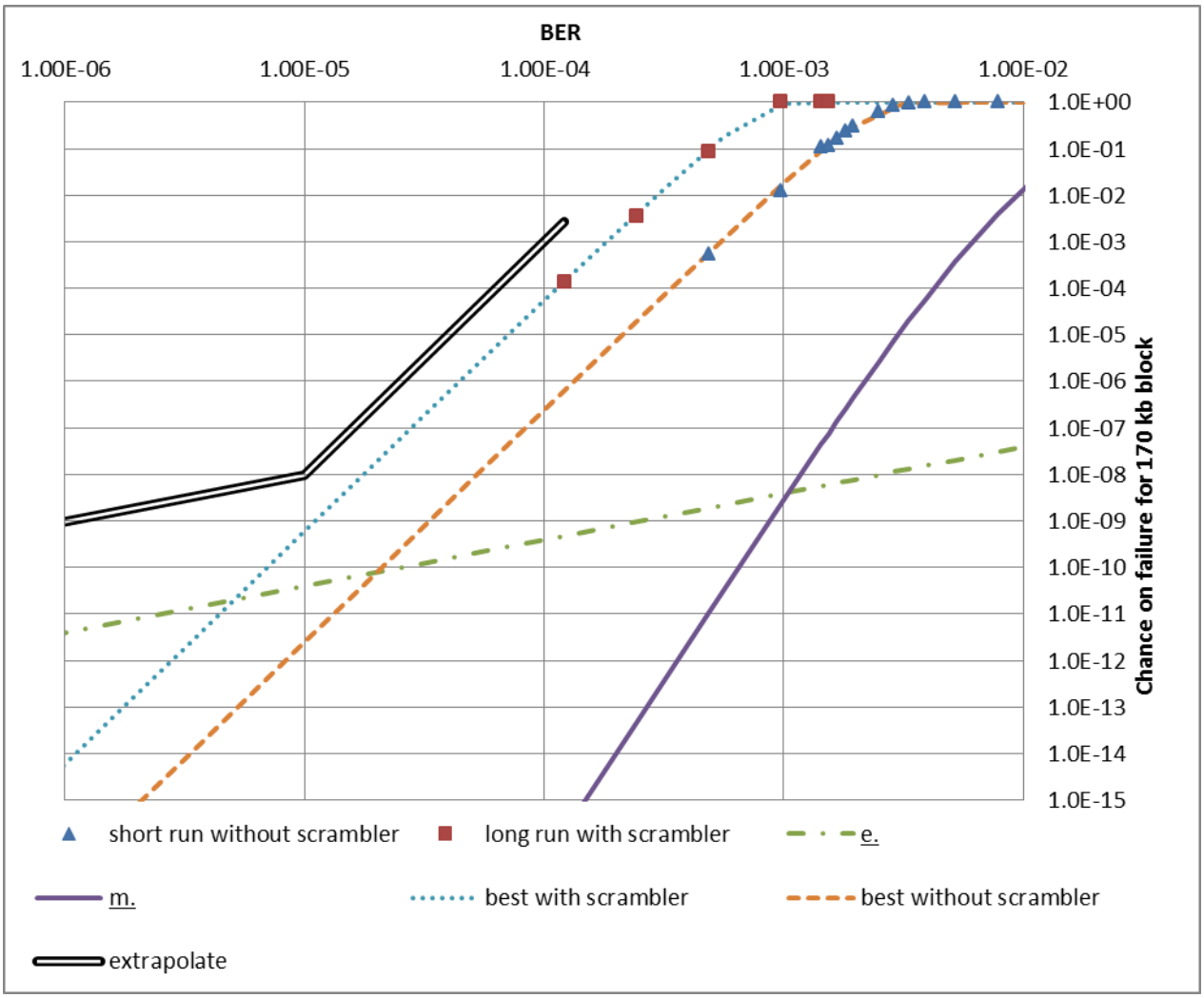

Figure 23 Simulation results with calculated estimates. Note that m is draw with assumption there need to be 4 sync errors.

4.2. HARDWARE TESTING

Instead of using simulation a hardware setup can be used. See figure 22 for the hardware setup used for testing. If the hardware can make BER of 10^{-4} then there is about 1 error in 4 hours. The design used has 4 channels, so about every 57 minutes there is a fatal error because of e. The hardware error injector can generate a BER of $\frac{1}{256*66} = 5.92 \cdot 10^{-5}$ or one e error average every 96 minutes for 4 channels at 8.25 Gbps.

Because the system runs at a much higher BER, other errors are playing a more significant role. However, the error injector that is used is not true uniform as the injector only injects one or zero errors per 66 bits word. As a result, all errors because of a double error in a 66 bits word do not happen. Meaning that error e stays the primary error for this test case.

For testing this time can be measured, this time has a Poisson binomial distribution. As the testing is done with a Hardware Evaluation license for the Reed-Solomon IP it will stop working after 2-8 hours. This could display the same error. So, to prevent this the system is run for 1.5 hours, after which the result is checked. It is calculated that in about 24.0% of the time the error

needs to have happened. Because of the way the test is run which requires manual resetting after 90 minutes, only about 3-5 tests a day can be run. Nevertheless, these tests are **useless** as every time the system is reset it also reset the error injectors; resulting in the same test every time.

Two tests have been run, both tests ended after about 3 hours, because of the RS IP time limit. So at least it is shown that the system can be run for three hours straight with a high single error injection rate.

In the end a hardware test was run with optical attenuators. Values tested where 3, 6, 9 and 12 dB, this attenuation is without loss with in the connecters. 3 and 6 dB is not enough to generate errors. At 9 dB the BER is around 10^{-12} to 10^{-10}. The system was run for an hour and all errors were correct. At 12 dB the system was not enable to setup a link.

4.3. BLOCK SYNCHRONISATION ERRORS

To check for the synchronisation errors, a run with only errors in the synchronisation part is done with the scrambler enabled. At a BER of $\frac{1}{64} = 0.016$, the failure rate is ~0.34 with $n = 35$. If a synchronisation word has more than 3 errors post scrambler it is marked as corrupt. The probability for this is given in theory and calculated in table 12. The failure chance for m is corrected for the fact that 2 or 3 error post scrambler will not lead to a failure. Only 4 and up will result in a failure. This makes the calculation for m more realistic and less pessimistic.

Table 12 Error probability at BER of 0.016.

		Pre scrambler	At least pre-scrambler	At most post-scrambler
l	0.27%	1 header + 1 data	1+1	1+3
m	28.16%	2 data	2	6
n	0.02%	2 header error	2(always failure)	2(always failure)
o	17.29%	3 error	3	At least 3
total	45.74%			

The transmission has a length of 1,705,158 bits which are processed by the RS decoder or expressed as 347*13 RS blocks (13*63*6 bit per RS-block). This means there are 10 synchronisation words. Every word has 45.74% probability on be corrupt. The simulated results give 19 out of 35 first synchronisation words are corrupt. This is a failure rate of 54.3%. As above table assumed that at max triple error occurs, while 4 errors is still 8.16%, 5 errors is 3.06% and 6 errors is 0.95%. If these errors are summed then it adds up to 57.91%. As the number of runs is low, it is not clear how well this matches nevertheless at least it is not a contradiction.

4.4. SIMULATION BURST ERROR TESTING

The system should be able to handle burst errors with a length of $ch * t$ symbols. Note that handling of burst error is not a design requirement but the outcome of the design decisions. A burst error does not always stop at the symbol boundary. This result in a shorter maximum burst length of $(ch * t - 1) * m + 1$, in this design $(13 * 4 - 1) * 6 + 1 = 307$ bits. It is not possible to say that it is always possible to correct at least a burst of 307 bits. As the bits are random they can become any signal including signals that are decoded as true. This means that it

is not possible to claim that a burst of 307 bits can always be corrected. For example, the case that a data symbol and error symbol are both changed such that the data is right again. The probability that all bits in a burst have a correct value for this is very small and can be safely ignored. A simulation is run where a burst error is simulated by replace *i* succeeding bits with random data for different lengths. It is expected that there are only a few failures to decode up to 307 bits of errors. The results are given in table 13.

This test does not cover is the fact that the header for 64b66b word synchronisation is also corrupt and only 15 header errors may happen before the realignment process starts again. In 307 bits there are maximum 5 headers. So even if all headers are corrupt it is still less than 15. This is not a limiting factor for error correction.

If there is a synchronisation word in the burst error then it will fail to decode because of the missing of the synchronisation word. There is one synchronisation word per 2458 words. There is a maximum of 5 error-words in one burst. The probability on a synchronisation word in a burst is $\frac{5}{2458} = 2.03 \cdot 10^{-3}$. The same story when there is a CC word. Because CC words normally is not seen as data words however in the case when it is corrupted, it may be seen as a data word. It is given that there are at least 3 CC words per 10,000 words but testbench has one per run. Results for the burst error simulations are given in table 13.

Table 13 Simulation runs of burst error

burst error length	runs	Failures	failure chance
150	9093	21	0.23%
200	9119	26	0.29%
250	9101	30	0.33%
300	9106	37	0.41%
350	9110	8598	94.38%
400	9096	8667	95.28%

It is expected that a small amount of burst errors is not recoverable until the burst error length become too long, that it will almost always fail. This is also shown in table 13. It can correct most of the error up to length of 300 bits, after which it drops quickly.

Symbol 0	Symbol 5	Symbol 10	**Symbol 15**	**Symbol 20**	Symbol 25	Symbol 30
Symbol 1	Symbol 6	Symbol 11	**Symbol 16**	**Symbol 21**	Symbol 26	Symbol 31
Symbol 2	Symbol 7	**Symbol 12**	**Symbol 17**	Symbol 22	Symbol 27	Symbol 32
Symbol 3	Symbol 8	**Symbol 13**	**Symbol 18**	Symbol 23	Symbol 28	Symbol 33
Symbol 4	Symbol 9	**Symbol 14**	**Symbol 19**	Symbol 24	Symbol 29	Symbol 34

Figure 24 Example of maximal burst correction (10 symbols), for t=2 and 5 channels

4.5. SEPARATE TESTING

Doing only BER testing is not enough to get a picture of the working. So instead a limited subset of errors is tested. Synchronisation words are already tested, see section 4.3. For the e errors this is very hard because even if there is always a header error then there is still only a

probability of about $7.32 \cdot 10^{-10}$ or about once every 90.21 Gb. This might be possible to test with a hardware co-simulation setup. This would require some modifications as the current hardware has not a true uniform error injector. This is not done because of limited time.

4.6. Conclusion testing

As already shown in this chapter, it is very hard to prove that the design is correct because failure to decode only happens very rarely. Test, where simply a BER is applied, will only indicate if the system works correctly. Using Wilson interval a couple of claims can be made.

Claim 1 with 95% (99%) confidence: If the BER is $6.10 \cdot 10^{-5}$ then the failure rate **without** scrambler is **less** than 0.0431 (0.0744%) for a block of 172 kb. Or an MDBF of at least 399 Mb (231 Mb).

Claim 2 with 95 (99%) confidence: If the BER is $1.22 \cdot 10^{-4}$ then the failure rate **with** scrambler is **less** than 0.262% (0.321%) for a block of 1.7 Mb. Or an MDBF of at least 650 Mb (531 Mb).

Claim 3 with 95% (99%) confidence: If the BER is $1.22 \cdot 10^{-4}$ then the failure rate **with** scrambler is at **least** than 0.0674% (0.0551%) for a block of 1.7 Mb. Or an MDBF at most 2.5 Gb (3.1 Gb).

If the most pessimistic approach to extrapolate to a lower BER is used then a first order approximation can be assumed resulting in an MTBF of at least 79.3 Pb or 111 days for 95%-confidence with a scrambler at a BER of 10^{-12}. Note that this is not a proof, as it assumes the worst-case order of errors is first order see section 2.6. A more optimistic approach could be to assume fifth order until a BER of 10^{-5} and for lower values first order. This extrapolation is also shown in figure 23 showing that it is still a safe bet. Then the MTBF is at least 6766 years. Calculating an upper bound on the MTBF is useless as it is fifth order resulting in a value many orders bigger than the age of the universe.

The Wilson score interval can be rewritten if no errors are detected, then the maximum failure rate is given by

$$e_m \approx \frac{1}{n} z^2$$

Where z is the z-score and n the number of tests. Doubling the amount of test reduces the error marge by half. If 95% is used and first-order approximation is applied. Then to get an MTBF of 1,000,000 years at BER of 10^{-12}, the failure rate is given by

$$MTBF = \frac{1}{\frac{DR}{run\ length} \cdot chance} = \frac{1}{\frac{DR}{1.7 \cdot 10^6} f_{long}}$$

$$f_{long@1E-12} = \frac{1.7 \cdot 10^6}{DR \cdot MTBF} = 6.5 \cdot 10^{-18}$$

And for BER of $1.22 \cdot 10^{-4}$.

$$f_{long@1.22E-4} = f_{long@1E-12} \frac{1.22 \cdot 10^{-4}}{10^{-12}} = 8.0 \cdot 10^{-10}$$

From which n can be found for BER of $1.22 \cdot 10^{-4}$.

$$n = \frac{z^2}{e_m} = \frac{1.96^2}{8.0 \cdot 10^{-10}} = 4.8 \cdot 10^9$$

Given that a run is about 80 seconds then the total runtime is about 120,000 years. In software this is not possible and hardware co-simulation needs to be used as then a run is only about 200 μs meaning that the test only takes 111 days. On top of this a single board can easily hold many parallel systems. 16 systems in one KCU105 is doable then only 7 days are needed for testing. If a higher level of confidence is needed then only marginal number of extra runs is needed. For example, if 99.9% is desired then instead 7 days it needs 21 days.

If it is desired to test it at real BER rates fully then the FPGA cloud service is the way to go. This is a service where you can rent FPGA for a short time. If the run length of a block is still 1.7 Mb and a proof is needed for MTBF of 1,000 years then a 3,800 years of simulation time is needed. For example the f1.2xlarge[3] holds one VU9P, which can hold about 64 parallel simulations. Meaning it needs about 60 f1.2xlarge years. Meaning that it is possible to test the system at low BER, however it will require a lot of hardware co-simulation time.

In the literature report the self-synchronising scrambler is approximated by means of saying the BER is 3 times higher. Inspecting the simulation results it is found that for same failure rate about a factor of 3 in BER between with and without scrambler is present, see figure 23. However, this is at a high BER. At a lower BER there is no data and because at lower BER different errors play a role it cannot simply be said that the assumption also holds at a lower BER rates.

All tests are done with uniform random distributed bit errors which are not correlated. If the error distribution is not independent then the result might change. All tests are done without considering clocking differences also the hardware test have the same clock for RX and TX.

[3] Amazon AWS EC2 F1 https://aws.amazon.com/ec2/instance-types/f1/

5. CONCLUSIONS

As has been shown it is possible to build a system that can handle errors using FEC. However, the desired MTBF of 1 million years is not met according to the worst case analysis. The main reason for this is that the control cannot be protected by means of FEC. However, it is not unthinkable that the real-life system performance is better than the MTBF of 1 million years. This is because of the pessimistic analysis of the system. To prove this a technique is suggested in section 4.6 but will require a lot of simulation time even when hardware co-simulation is used.

One of the major problems with FEC is that when it is moved further away from the transceivers more logic is introduced in-between them. The logic in-between also needs be analysed for error propagation. In this case this logic is the main source of unrecoverable errors.

A few claims are made in combination with pessimistic extrapolation. The system performance at BER of 10^{-12} can be estimated to be at least about a couple of thousand years. This is still a lot lower compared to the goal. However, compared to the system without FEC it is still a lot better. A hardware test was run at BER of 10^{-12} - 10^{-10} which shows that it works with real errors. However the test was only run for a short time.

It has been shown that a scrambler can be effectively be used to make the data uniform. The presented design only consumes limited resources and does not amplify the errors. The already present self-synchronising scrambler has the property that it amplifies errors in a predictable way. This means that the error correction is more loaded then when a no-self-synchronising scrambler is used.

As the IEEE Ethernet standard already shows with 100 GbE (25 Gbps lane rate) that wireline BER is no longer less than 10^{-12} and it is not expected that the newer standard will restore the BER again. Meaning that it is likely that FEC will become standard practice for wireline communication systems. When the system is used in the new 100 GbE architecture with a BER of 10^{-5} the system only accomplish an MTBF of around 1-2 hours. Ethernet has a mean time between packet loss of 40 seconds. Also, remember that it was stated in literature report [2] that a loss of 1 dB increases the BER by a factor of 100. At 5 GHz / 10 Gbps the loss for PCB trace on FR-4 is already around 2 dB for 200 mm or ~0.01 dB/mm [22]. A 5 cm PCB trace is equal to increase of factor 10 in BER.

6. RECOMMENDATIONS

Currently the header is not protected; a solution for this could be to move a data bit to the header of a control word. So 3 header bits and 63 bits data. These 3 header bits encode the two states, when using hamming(3,1) it is always possible to decode the state if there is one error or less. Another approach could be to have 1 header bit, 64 bits data and 1 parity bit. Alternatively, find an error correction algorithm that has an overhead of less than 1/65, for example RS(1023,1017). However, with these approaches, the clock recovering needs to also be adjusted.

However, the best approach is to redesigning a complete new encoding. So not standard 64b66b as it is not designed with error correction in mind. For example an encoding could be built around 128 bit word plus 4 header bits. One of these header bits could be a parity bit to make it possible to detect always a single error. Also, the self-synchronising scrambler could be removed and traded in for xor-scrambler with a limited set of vectors. Future research could look into design of a fault tolerate 128b132b code with RS(255,251) or 192b195b with RS(4095,4087).

The gearboxes used in this design are very large and could build a lot simpler. Currently, all shift register bits can go to the output; this is most of the time not needed as the offset has a limited number of possible values. By better selection of the gearbox ratios by means of changing $ch * m$, one should be able to reduce the resource use. Also reducing the input shift register size should help a lot. Also FIFO size can be reduced on many places when a detail analysis is done on them.

The present design can be upgraded easily to higher bitrates, if the system cannot keep up simply increase the number of parallel RS blocks and the throughput is increased. Another option could be to give the FEC logic its own clock domain and use the FIFOs for clock domain crossing.

Currently, the IDLE words are not protected and if there is a bit error it will propagate. Idle words contain 2 bytes of useful information of which 4 bits are fixed to '0', the other 6 bytes are don't care. These 6 don't care bytes could contain the same data as the first 2 bytes and use major voting to choose which one is right. When doing this one has to be aware of the self-synchronising scrambler.

The analysis becomes quickly complicated, to overcome this, it is desired to use an FEC algorithm even closer to the transceiver. Preferable run it at the recovered transmitter clock just after the word level synchronisation. This allows for more straightforward analysis and results in a better error correction strength. This might introduce problems with integration in the current design.

Another could be to use formal methods to validate the design. This will allow for better proof of the design. With these methods one has to be very careful not to run in a state-space explosion.

The new scrambler introduced could also be used for encrypting with a shared key instead of the pseudo-random data. This is known as XOR-cipher this method is not known for its strong encryption [23]. However, might be better than no encryption. Other encryption algorithms may be used as long as the resulting distribution is close to uniform.

As the data is uniform distributed it might be possible to disable the self-synchronizing scrambler after the setup phase. This removes the problem with error amplification of the

scrambler. Alternatively, use the properties of the scrambler to predict where the other errors are.

Other options could be to do data compression on control words as there is only a very limited set of control words.

6.1. TAKING IT IN PRODUCTION

Before putting the system in a production environment, it needs to be tested thoroughly. As explained in the testing chapter only a few tests have been run. Which where all positive but because of the nature of the design this does not proof that the design is always working. It is recommended to run the system with a real high BER in the order of 10^{-10}, for example by using an optical attenuator. Another approach is to use an external error injector.

Also, it is highly recommended to do a redesign and move the FEC closer to the transceivers as explained in section 3.8. This will make analysing a lot simpler. A monitor interface needs to be integrated.

7. BIBLIOGRAPHY

[1] International Organization for Standardization, IEC 80000-13:2008, Quantities and units -- Part 13: Information science and technology, International Organization for Standardization, 2008.

[2] M. Brakels, "Literature study on Forward Error Correction and failure rates," Enschede, 2017.

[3] IEEE, 802.3-2015 - IEEE Standard for Ethernet - Section 2, IEEE, 2015.

[4] Xilinx, Aurora 64B/66B v1.3 SP011, Xilinx, 2014.

[5] Xilinx, DS890 UltraScale Architecture and Product Data Sheet: Overview v3.0, Xilinx, 2017.

[6] Xilinx, "KCU105 Board User Guide," [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/kcu105/ug917-kcu105-eval-bd.pdf.

[7] IEEE, 802.3-2015 - IEEE Standard for Ethernet - Section 4, IEEE, 2015.

[8] Xilinx, PG074 Aurora 64B/66B v11.1 LogiCORE IP Product Guide, Xilinx, 2016.

[9] IEEE, 802.3-2015 - IEEE Standard for Ethernet - Section 6, IEEE, 2015.

[10] S. B. W. a. V. K. Bhargava, "An Introduction to Reed-Solomon Codes," 1994.

[11] W. a. Bhargave, An introduction to Reed-Solomon Codes, Wiley-IEEE press, 1994.

[12] S. Haykin, Analog and Digital communications, John Wiley & Sons, 2007.

[13] IEEE, 802.3-2015 - IEEE Standard for Ethernet - Section 3, IEEE, 2015.

[14] Xilinx, PG107 Reed-Solomon Decoder v9.0, Xilinx, 2015.

[15] K.-M. a. M. R. J. Cheung, "The undetected error probability for Reed-Solomon codes," *Military Communications Conference, 1988. MILCOM 88,* 1988.

[16] Xilinx, UG574 UltraScale Architecture Configurable Logic Block User Guide v1.5, Xilinx, 2017.

[17] K. Chapman, XAPP522 v1.2 Multiplexer Design Techniques for Datapath Performance with Minimized Routing Resources, Xilinx, 2014.

[18] Xilinx, PG057 FIFO Generator LogiCORE IP Product Guide v13.1, Xilinx, 2017.

[19] ARM, AMBA4 AXI4-Stream Protocol v1.0.

[20] IEEE, IEEE-1364-2005 Standard for Verilog Hardware Description Language, IEEE, 2005.

[21] L. D. Brown, T. T. Cai and A. DasGupta, Interval Estimation for a Binomial Proportion, The Institute of Mathematical Statistics, 2001.

[22] T. Okubo, T. Sudo, T. Hosio, H. Tsuyoshi and F. Kuwako, Signal transmission loss on printed circuit board in GHz frequency region, IEEE, 2013.

[23] T. Johansson and F. Jonsson, Theoretical analysis of a correlation attack based on convolutional codes, IEEE Transactions on Information Theory, 2002.

APPENDIX A SIMULATION RESULTS

Table 14 Simulation results. The long run is normalised to length of short run

	short run without scrambler		long run with scrambler		estimate			
BER	Failure probability	n	Failure probability	n	e.	m.	best with scrambler	best without scrambler
1.00E-12					3.93E-18	4.25E-81	6.04E-45	2.49E-47
1.53E-05	0.00E+00	966			6.00E-11	1.24E-23	4.93E-09	2.05E-11
3.05E-05	0.00E+00	958			1.20E-10	3.16E-21	1.56E-07	6.52E-10
6.10E-05	0.00E+00	8910			2.40E-10	8.00E-19	4.84E-06	2.07E-08
1.22E-04	0.00E+00	943	1.33E-04	6012	4.80E-10	2.00E-16	1.47E-04	6.50E-07
2.44E-04	0.00E+00	8901	3.59E-03	311	9.60E-10	4.91E-14	4.19E-03	2.00E-05
4.88E-04	5.56E-04	28758	8.86E-02	306	1.92E-09	1.15E-11	1.02E-01	5.95E-04
9.77E-04	1.23E-02	891	1.00E+00	315	3.84E-09	2.47E-09	8.94E-01	1.63E-02
1.43E-03	1.09E-01	883	1.00E+00	308	5.61E-09	4.41E-08	1.00E+00	9.18E-02
1.54E-03	1.19E-01	906	1.00E+00	314	6.04E-09	7.67E-08	1.00E+00	1.26E-01
1.67E-03	1.70E-01	887			6.54E-09	1.39E-07	1.00E+00	1.76E-01
1.82E-03	2.39E-01	894			7.14E-09	2.64E-07	1.00E+00	2.49E-01
1.95E-03	3.12E-01	905			7.66E-09	4.47E-07	1.00E+00	3.25E-01
2.50E-03	6.37E-01	886			9.81E-09	2.65E-06	1.00E+00	6.84E-01
2.86E-03	8.57E-01	886			1.12E-08	6.80E-06	1.00E+00	8.67E-01
3.33E-03	9.63E-01	900			1.31E-08	1.98E-05	1.00E+00	9.78E-01
3.91E-03	9.99E-01	885			1.53E-08	5.75E-05	1.00E+00	9.99E-01
5.21E-03	1.00E+00	892			2.04E-08	3.66E-04	1.00E+00	1.00E+00
7.81E-03	1.00E+00	962			3.05E-08	3.86E-03	1.00E+00	1.00E+00
1.04E-02	1.00E+00	952			4.05E-08	1.62E-02	1.00E+00	1.00E+00

Table 15 Wilson score interval 95% for simulation results

	Short run without scrambler		long run with scrambler	
BER	Lower	Upper	Lower	Upper
1.00E-12				
1.53E-05	4.32E-19[4]	3.96E-03		
3.05E-05	-4.32E-19[4]	3.99E-03		
6.10E-05	2.71E-20[4]	4.31E-04		
1.22E-04	-4.32E-19[4]	4.06E-03	2.00E-05	8.85E-04
2.44E-04	0.00E+00	4.31E-04	6.86E-04	1.86E-02
4.88E-04	3.43E-04	9.04E-04	6.17E-02	1.26E-01
9.77E-04	6.91E-03	2.20E-02	9.88E-01	1.00E+00
1.43E-03	8.99E-02	1.31E-01	9.88E-01	1.00E+00
1.54E-03	9.97E-02	1.42E-01	9.88E-01	1.00E+00
1.67E-03	1.47E-01	1.96E-01		
1.82E-03	2.13E-01	2.68E-01		
1.95E-03	2.82E-01	3.43E-01		
2.50E-03	6.04E-01	6.68E-01		
2.86E-03	8.32E-01	8.78E-01		
3.33E-03	9.49E-01	9.74E-01		
3.91E-03	9.94E-01	1.00E+00		
5.21E-03	9.96E-01	1.00E+00		
7.81E-03	9.96E-01	1.00E+00		
1.04E-02	9.96E-01	1.00E+00		

[4] Rounding error because of limit precision of double word, should be zero

APPENDIX B A SHORT INTRODUCTION TO THEORY OF PROBABILITY

In this Appendix a short introduction to the relevant theory of probability is given. For more detailed explanation, please read [12].

Consider the following case where there is a collection of coins. A coin has a probability of having heads up of 0.5(50%). Now consider the case where two coins are thrown, then there are four possible outcomes: HH, HT, TH, TT. Where H is head and T is tail. Assume that throwing one coin does not affect the other coin, this is called independence. Then there are two separated throws (experiments). Consider the case where the outcome is HH. The first coin toss have a probability of 0.5 for H, the same for the second. So, because the events do not influence each other (independent) the probabilities can be multiplied. So, the probability on HH is 0.25, this is of course also the case for TT. For HT, this is 0.25 but because it cannot be distinguished between HT and TH, this will sum to 0.5.

Consider that heads is defined as success, this kind of experiments are binomial distributed experiments. For this, the following formula holds for probability.

$$f(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Where k is the number of successful experiments (heads), n is the number of experiments and p is the probability on success for a single experiment. So, considering the case with coins then for HH $(2,2,0.5)$, TT $f(0,2,0.5)$ and HT/TH $f(1,2,0.5)$.

Going back to the case with bit errors, the probability on a bit error is expressed as Bit Error Rate (BER), which is the average number of bit errors per bit. These rates are in the order of 10^{-12} , so average one error per 1,000,000,000,000 bits. The assumption is that these errors are independent. Then p is equal to BER, so p gives the probability on a bit error. Consider the case where there is an error in a 64 bits word, how likely is this is given by $f(1,64,BER)$. But normally also 2+ errors need to be taken into account. But because BER is much smaller then 1, the other terms can be neglected ($BER \gg BER^2$). So, the probability on a bit error in a 64 bits word can be approximated by $64BER$.

Consider have a random byte which has a uniform distribution, meaning that the probability on the value 0x00 is the same as the value 0x01 and the value 0x02 etc. This means that the probability on 0x00 is $\frac{1}{256}$ because there are 2^8 options which are all even likely. Going back to tossing coins, see the value 0x00 as 8 times TAIL and 0x01 as first 7 times TAIL and then 1 times HEAD. So, the probability that all 8 tosses are equal to the prediction is given by $f(8,8,0.5) = \frac{1}{256}$.

The MTBF can be calculated with

$$MTBF = \frac{1}{n \cdot p}$$

Where n is the number of events per second and p is the probability on a failure. So, when the link is running at 1 Gbps and word size of 64 bit. Then the probability on failure in a word is $p = f(1,64,BER) \approx 64\,BER$. There are $n = \frac{1 \cdot 10^9}{64} = 1.5625 \cdot 10^7$ words per second so MTBF is $\frac{1}{10^9 BER}$, for BER of 10^{-12} this is MTBF of 1000 seconds.

APPENDIX C ERROR PROPAGATION

Below the different cases are calculated. When calculating these cases, it is often assumed that $BER \ll 1$. This assumption will help with two things: it will remove the need to do error analyse on many errors in one word and it helps to simplify the equations.

a. Considering the case where a data word is transmitted. The probability that of 66 bits non bits are corrupted is given by
$f(0,2,BER) * f(0,64,BER) = (1-BER)^2(1-BER)^{64} \approx 1$
Note that it is split in 2 bits header part which has error rate of BER and 64 bits data part, with error rate of BER. As the data is pushed through the scrambler, the number of error triple but does not affect the probability on error, only the fact that single error in data is always a triple error in data. But because the previous word could also have errors it also needs to be make sure that the previous 58 bits do not have an error . It becomes then
$f(0,2,BER) * f(0,64,BER) * f(0,58,BER) = (1-BER)^2(1-BER)^{64}(1-BER)^{57} = 1 - 1.23 \cdot 10^{-10} \approx 1$

b. A transmitted data word has 1 error in data part. This can be corrected by FEC.
Current: $f(0,2,BER) * f(1,64,BER) = (1-BER)^2\ 64\ (1-BER)^{63}(BER)^1$
There is also the probability that a previous error propagates. In this case it can be simple assume the worse that 1 error will result in 3 errors in next word. This is impossible because of the fact that at most 2 errors propagate. But this makes the calculation only more conservative.
$f(1,64,BER) * \frac{58}{64} = \frac{58}{64} 64BER = 58BER$

c. A transmitted data word has 2 errors in data part. This can be corrected by FEC.
$f(0,2,BER) * f(2,64,BER) = (1-BER)^2\ 64 \cdot\ 63/2\ (1-BER)^{62}(BER)^2$
$f(0,2,BER) * (f(1,64,BER) * \frac{58}{64} * f(1,64,BER) + f(2,58,BER))$

d. A transmitted data word has 3 errors anywhere. If this happens it is assumed that it will result in failed decode. But most of the time FEC can correct it. Two words are considered because the scrambler can cross words.
$f(3,132,BER) = (1-BER)^{129} * 132 * 131 * 130/6 * (BER)^3$

e. A transmitted data word has an error in the header. It now not clear if it is a control or data word. The decoder has preferred for control so it checks how well it match with idle control word, if no more than 7 bit different it accept it as idle control. One bit different is already in the header.
$f(1,2,BER) * \frac{64*63*62*61*60*59}{2^{64}} = \frac{64!\cdot 2BER}{58!\cdot 2^{64}}$

f. As e, but only 3 bit different is accepted from synchronization word.
$f(1,2,BER) * \frac{64*63}{2^{64}} = \frac{64!\cdot 2BER}{62!\cdot 2^{64}}$

g. A transmitted data word with two header errors will become control word. It can be ignored(i), seen as idle(g) or synchronization(h). The probability that header has 2 errors is $f(2,2,BER) = BER^2$. For idle it may be 7 bits away, so 64*63*62*61*60*59*58 possible words that are 7 bits away
$\frac{BER^2 64!}{57!2^{64}}$

h. Synchronization, same as g but now 3 bits away
$\frac{BER^2 64!}{61!2^{64}}$

i. Ignored, remaining options if 2 header errors
$BER^2(1 - \frac{64!}{57!2^{64}} - \frac{64!}{61!2^{64}})$

j. A transmitted control synchronization word has no error(s):
$f(0,2,BER) * f(0,64,BER) = (1 - BER)^2(1 - BER)^{64}$

k. 1 error in control synchronization word is always decoded as a synchronization.
Meaning not resulting in error
$f(0,2,BER) * f(1,64,BER) + f(1,2,BER) * f(0,64,BER)$
And propagation of error from previous word
$f(1,87,BER)$

l. 1 error header and 1 error in data when a synchronization word is send.
$f(1,2,BER) * f(1,64,BER)$
And progation of error from previous word
$f(1,58,BER)f(1,2,BER)$

m. 2 data errors (effectieve 6 maximum), is not always decode as a synchronization so seen as failure
$f(0,2,BER) * f(2,64,BER)$
And progation of error from previos word
$f(2,58,BER) + f(1,58,BER)f(1,64,BER)$

n. 2 header errors is seen as dataword, so failure.
$f(2,2,BER) * f(0,64,BER)$

o. Triple errors are assumed to result in failure
$f(3,132,BER)$

p. A transmitted control idle word has no error(s):
$f(0,2,BER) * f(0,64,BER) = (1 - BER)^2(1 - BER)^{64}$

q. 1 error in control idle word is always decode as an idle but may contain a corrupted idle
$f(0,2,BER) * f(1,64,BER) + f(1,2,BER) * f(0,64,BER)$
And progation of error from previous word
$f(1,58,BER)$

r. 1 error header and 1 error in data, is always decode as an idle but may contain a corrupted idle
$f(1,2,BER) * f(1,64,BER)$
And progation of error from previous word
$f(1,58,BER)f(1,2,BER)$

s. 2 data errors, this may result in corrupt IDLE word
$f(0,2,BER) * f(2,64,BER)$
And progation of error from previous word
$f(2,58,BER) + f(1,58,BER)f(1,64,BER)$

t. 2 header errors, this will transform it in data word
$f(2,2,BER)$

u. Triple error are assumed to result in failure
$f(3,132,BER)$

APPENDIX D COMPLETE CHAIN