BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF ELECTRICAL AND ELECTRONIC TECHNOLOGY

ÚSTAV ELEKTROTECHNOLOGIE

RS-FEC LAYER IMPLEMENTATION FOR 400GB/S ETHERNET

RS-FEC LAYER IMPLEMENTATION FOR 400GB/S ETHERNET

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Patrik Zahálka**
AUTOR PRÁCE

SUPERVISOR — **Ing. Petr Vyroubal, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2020

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Electrical Manufacturing and Materials Engineering**

Department of Electrical and Electronic Technology

Student: Bc. Patrik Zahálka ***ID:*** 173783

Year of study: 2 ***Academic year:*** 2019/20

TITLE OF THESIS:

RS-FEC layer implementation for 400Gb/s ethernet

INSTRUCTION:

Learn about the Intel Stratix 10 DX FPGA chips and describe the principles of Reed-Solomon correction coding and its use in the Ethernet protocol. Design an RS-FEC layer architecture for 400 Gb / S Ethernet on an Intel Stratix 10 DX FPGA chip. Implement the proposed architecture and verify its functionality in simulations. Evaluate the results and discuss the possibilities for further expansion.

RECOMMENDED LITERATURE:

Podle pokynů vedoucího diplomové práce.

Date of project specification: 3.2.2020 ***Deadline for submission:*** 19.5.2020

Supervisor: Ing. Petr Vyroubal, Ph.D.

doc. Ing. Petr Bača, Ph.D.
Subject Council chairman

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Elektrotechnická výroba a materiálové inženýrství**

Ústav elektrotechnologie

Student: Bc. Patrik Zahálka ***ID:*** 173783
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Implementace vrstvy RS-FEC pro 400 Gb/s Ethernet

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s FPGA čipy Intel Stratix 10 DX a popište principy opravného kódování Reed-Solomon a jeho použitím v rámci protokolu Ethernet. Navrhněte architekturu vrstvy RS-FEC pro 400 Gb/S Ethernet na FPGA čipu Intel Stratix 10 DX. Navrženou architekturu implementujte a ověřte její funkčnost v simulacích. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího rozšíření.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího diplomové práce.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 3.6.2020

Vedoucí práce: Ing. Petr Vyroubal, Ph.D.

doc. Ing. Petr Bača, Ph.D.
předseda oborové rady

ABSTRACT

This Master's thesis deals with RS-FEC layer implementation using VLSI hardware description for 400 GE (Gigabit Ethernet) in the FPGA Intel® Stratix® 10 DX 2100. In the theoretical part of this work, current state of Ethernet speeds and context of RS-FEC layer within Ethernet protocol is described including PLD fabrication process and mathematical aspects of RS-FEC self-correction algorithm. In the practical part, parametrizable RS-FEC system is described including evaluation of the first results achieved and future scope of this project is discussed.

KEYWORDS

Reed-Solomon Error Correction Codes, Forward Error Correction, 400 Gbps Ethernet, FPGA

ABSTRAKT

Tato diplomová práce se věnuje problematice VLSI návrhu a implementaci vrstvy RS-FEC pro 400 Gb/s Ethernet do FPGA Intel® Stratix® 10 DX 2100. V práci je charakterizován současný stav rychlostí Ethernetu, význam a kontext samoopravných kódů v rámci protokolu Ethernet. Dále je popsána výroba PLD čipů i matematická podstata RS samoopravných kódů. V části praktické je představen návrh řešení systému RS-FEC, který byl realizován genericky pomocí jazyka VHDL. Zároveň byly jeho komponenty implementovány a v závěrečné diskusi je popsáno jeho řešení, dosažené výsledky včetně jeho budoucího rozšíření.

KLÍČOVÁ SLOVA

Reed-Solomonovy samoopravné kódy, 400 Gb/s Ethernet, FPGA

ZAHÁLKA, Patrik. *RS-FEC layer implementation for 400 Gb/s Ethernet*. Brno, 2020, 77 p. Master's Thesis. Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav elektrotechnologie. Advised by Ing. Petr Vyroubal, Ph.D.

ROZŠÍŘENÝ ABSTRAKT

V dnešní době nám možná připadá přirozené, že poskytovatelé datových služeb přichází čas od času s novými produkty i vyššími rychlostmi datových přenosů. Dříve bylo standardní praxí počkat si na to, až se nám s připojením přes telefonní linku načte obrázek a netrpělivě jsme sledovali každou jeho načtenou část. Dnes však na tuto skutečnost pomalu zapomínáme. Navíc i vzhledem k tomu, že se do sítě připojuje stále více nových zařízení, stávají se internetové služby populárním a nepostradatelným médiem pro dnešní společnost [1]. Nároky uživatelů internetu ruku v ruce s technologickým pokrokem však dovedly maximální přenosové rychlosti od jednotek kilobitů po stovky gigabitů za sekundu, a meta zde rozhodně nekončí. V současné době existují experimantální vysokorychlostní ethernetové platformy, které operují bezmála na jednotkách terabitů za sekundu. Rychlejší přenos dat neznamená pouze nějaké vyšší číslo či kratší dobu čekání na odeslání souboru. Stojí za tím obrovské úsilí celého elektrotechnického průmsylu od inovací v kabeláži, elektrotechnické i elektronické výrobě po inovace v informačních a síťových technologiích. Každý takový pokrok umožní posun v jiných odvětvích, například ve zdravotním průmyslu pro vývoj nových léků a vakcín, předpovědích klimatu, finančních službách, genetickém inženýrství atp.

Vzhledem k tomu, že jsou obecně přenosová média vystavena faktorům rušení, stává se běžně, že se někdy přenesená zpráva poškodí a přijímač ji proto poté chybně vyhodnotí (či vůbec). Toto se stává tím častěji, čím rychlejší platforma je a čím méně je ošetřena proti rušení. Tomu lze však předejít několika způsoby. Jedním z nich je např. použití kroucené dvojlinky, která je však tématem pro nižší přenosové rychlosti, až 1 Gbit/s [2]. Jiný způsob, než použití kabeláže, je implementace samoopravných algoritmů pro rekonstrukci přenášené zprávy navzdory tomu, že je z části poškozena. Jedná se o algoritmy, které lze implementovat do hradlových polí FPGA jako firmware. V této diplomové práci je řešen aspekt implementace Reed-Solomonových kódu pomocí návrhu VLSI pro hradlová pole Intel® Stratix® 10 DX pro projekt NDK - Netcope Development Kit.

Reed-Solomonovy kódy byly doporučeny jako vhodné řešení pro dosažení rychlostí 100 Gb/s ve standardu IEEE Std. 802.3bm^{TM} z roku 2015. Důvodem je hlavně to, že jsou celkově dobře prostudovány a jejich vývoj sice stojí úsilí, ale dodá potřebnou kapacitu pro opravování chyb a lze je v rozumném měřítku implementovat do hradlových polí FPGA. Základní koncept pro opravu chyb pomocí Reed-Solomonových kódů spočívá v tom, že se připojí redundantní část o určitém počtu symbolů ke zprávě, kterou chceme poslat, tzv. parita. Tím se vytvoří unikátní a validní datový „balík“, tzv. kódové slovo, které je výstupem vysílací strany systému RS-FEC. Pokud se kódové slovo poškodí, přijímací strana chybu detekuje a vytvoří polynom, který se dále dešifruje následujícími komponentami, které dokáží

najít pozice chyb a jejich velikosti na těchto pozicích. Toto „poškození“ lze chápat tak, že se k původnímu nepoškozenému kódovému slovu přičetl určitý chybový polynom, který právě reprezentuje vzniklý šum. Systém tedy hledá přesně tento polynom, který způsobil chybu při přenosu, tzv. chybový polynom $E(x)$. Nalezením tohoto polynomu a jeho přičtením k přijatému „poškozenému“ kódovému slovu se docílí rekonstrukce, ideálně původního validního kódového slova. Reed-Solomonovy kódy se obecně značí $RS(n, k)$, kde n specifikuje počet prvků celého kódového slova a k počet prvků zprávy.

Tato diplomová práce se tedy zabývá touto problematikou, implementací RS(544, 514) pro 400 Gb/s Ethernet. Předpokládá se, že čtenář nemá předchozí zkušenosti s touto problematikou, a proto je v práci Reed-Solomonův samoopravný algoritmus detailně shrnut. Teoretická část dále obsahuje kapitolu o síťových technologiích, výrobě mikročipů a informace o použité technologii. V praktické části je uveden návrh řešení systému pro 400GE (Gigabit Ethernet), podrobný rozbor navržených komponent a prozatimní dosažené výsledky úspěšné implementace. Dále jsou v tomto textu shrnuty výsledky a také je konstruktivně okomentováno další rozšíření tohoto systému. Samotný proces Reed-Solomonových samoopravných kódů je detailně popsán v [3] včetně jeho hardwarové podoby, a proto se tato diplomová práce z velké části odkazuje právě na tento zdroj.

Reed-Solomonovy kódy operují nad tzv. Galoisově konečným tělesem. To ve výsledku znamená, že celý samoopravný algoritmus provádí výpočty s $(m-1)$-bitovými symboly, přičemž každý takový symbol je jedním z celkem jeho $2^m - 1$ uspořádaných členů. V rámci systému RS-FEC pro 400GE se výpočet provádí s desetibitovými symboly, proto $m = 10$. Prvky Galoisova tělesa jsou určeny i uspořádány podle toho, jaký tzv. primitivní polynom $p(x)$ řádu m (v binární podobě) toto těleso vytváří. Zajímavou vlastností konečných těles je to, že pokud se budeme snažit vytvořit $(2^m + 1)$ prvek, výsledkem bude znovu prvek první, $(2^m + 2)$ prvek je znovu prvek druhý atd. V Galoisově tělese platí také odlišná algebra, a to taková, že při provádění algebraických operací nedochází ke změně řádů (přetékání bitů). Operace sčítání a odčítání je jedna a ta samá operace, která se provádí pomocí hradla XOR. Systém tedy nevyžaduje použití žádných sčítaček ani odčítaček, a tudíž nelze využít DSP bloky v čipu FPGA. V práci byla tedy zvolena varianta implementace systému do Look-up tabulek hradlového pole včetně jejich použití jako paměti ROM. Jak je obecně známo, logické operátory XOR jsou sobě komutativní a výsledný návrh lze optimalizovat, což provádí syntézní nástroje, v této práci Intel Quartus Prime Pro [4]. Operace násobení se také liší a provádí se ve dvou fázích. Vstupy do násobičky nad $GF(2^m)$ chápeme jako polynomy řádu $m-1$ a provádíme klasické násobení mnohočlenu mnohočlenem. Výsledek tohoto násobení však překračuje řád prvků $GF(2^m)$ a je tedy nutné jej do tohoto řádu “vrátit” pomocí operace modulo

m vydělením tím samým primitivním polynomem $p(x)$, který generuje celé Galoisovo těleso $GF(2^m)$. Implementace dělení v Galoisově poli je komplikovanější a vyžaduje využití velkého množství logických zdrojů. Vzhledem k tomu, že se v systému RS-FEC násobičky vyskytují ve velkém množství, jedná se o komponentu velmi náročnou na využití zdrojů v FPGA, zejména Look-up tabulek. Dělení bylo provedeno tak, že po nalezení inverzního prvku dělitele (jmenovatel) se tento prvek vynásobil s dělencem (čitatel). V komponentách, kde se provádí dělení nad $GF(2^m)$, se proto musely nejdříve najít tyto inverzní prvky, seřadit vzestupně dle jejich základní hodnoty a pro dosažení vysoké datové propustnosti implementovat do Look-up tabulek charakteru ROM. Toto provádí navržený VHDL podprogram, který dokáže vytvořit GF inverze pro kterákoliv požadovaná $GF(2^m)$ bez nutnosti spouštět externí skripty. Tyto vektory však zabírají velké množství logických zdrojů a bylo potřeba jich v systému generovat co nejméně. I když je systém syntetizovatelný, výsledný návrh vyžaduje pro finální implementaci z důvodu přílišného využití dostupných zdrojů v FPGA odpovídající optimalizace. To se také projevuje dlouhou dobou fáze hledání vhodného propojení logických buněk (Route). V návrhu bylo tedy nutné vhodně určit, kdy použít konstanty a kdy násobičky. Pro samoopravný algoritmus bylo také potřeba najít čtverce a vyšší řády Galoisova tělesa. Vhodným řešením bylo použití operace násobení nežli generování konstanty navzdory tomu, že prvotní návrh počítal s pravým opakem, a to z důvodu využití co nejmenšího množství násobiček v GF pro zkrácení kritické cesty ze vstupu na výstup komponent.

Vrstva RS-FEC je součástí podvrstvy PCS (Physical Coding Sublayer) fyzické vrstvy Ethernetu PHY. Komponenta systému RS-FEC, která přijímá data z PCS vrstvy na vysílací straně, se nazývá RS-FEC Enkodér, který provádí tzv. skramblování. To je takový rekurzivní proces, který provádí dělení mnohočlenu mnohočlenem, tedy celé zprávy $M(x)$ tzv. Generačním polynomem $G(x)$, který definuje norma IEEE Std 802.3bs™ 2017. Jeho úkolem je vytvořit tzv. *paritu*, což je zbytek po tomto rekurzivním procesu dělení. V praxi se při implementaci RS-FEC Enkodéru ukázalo, že syntézní nástroje nedokáží plně optimalizovat komplexní zapojení této funkce pro potřebnou datovou propustnost, což je z velké části dáno návrhem násobiček. Zjistilo se, že dva různé parametrizovatelné návrhy násobiček v RS-FEC Enkodéru vyústily ve dva různé, avšak zanedbatelně rozdílné, výsledky ve smyslu využití zdrojů, a proto je potřeba hledět na procesy skramblování a deskramblování jinak, než udává obecné schéma pro provádění této operace.

Vysílací strana dále také provádí distribuci jednotlivých symbolů zprávy mezi jednotlivé linky PCS. Vzhledem ke standardu IEEE Std 802.3 bs™ 2017 distribuci provádí MUX mezi 16 PCS linek, přičemž navržený systém nabízí funkci volby počtu linek pomocí generických patametrů, a tudíž lze navržený systém použít i pro 200GE

či budoucí rychlosti Ethernetu, pro které v současnosti ještě neexistuje normovaná verze.

Druhou částí systému RS-FEC je část přijímací, která dekóduje dvě kódová slova. Pro provedení opravy zprávy musí přijímací strana nejprve vrátit proložené symboly zprávy do stavu před proložením. Poté se provede tzv. deskramblování, a to každého přijatého kódového slova. To je obdobný proces jako skramblování, avšak se v tomto případě dělení provádí pouze jedním elementem Galoisova tělesa, resp. kořenem generačního polynomu $G(x)$, poté jeho následujícím elementem, atp. Dělení se provede tedy celkem $2t$-krát, přičemž v FPGA lze toto dělení provádět paralelně. Nutno podotknout, že opravnou kapacitu Reed-Solomonova samoopravného algoritmu t určuje jeho tzv. *Hamingova vzdálenost*, která je dána počtem paritních symbolů, v tomto případě algoritmus opraví až 15 symbolů. Deskramblováním lze zjistit, zda jsou přijatá kódová slova dělitelná beze zbytku kořeny Generačního polynomu $G(x)$. Tím se ověří, zda při přenosu k nějaké chybě došlo, čili zda je na vstupu dekodéru validní kódové slovo či nikoliv. Pokud je výsledek nulový, systém RS-FEC opravu provádět nebude. Pokud ne, deskrambler vytvořil takový polynom, který je nutno dále dešifrovat za účelem zpětného vytvoření validního kódového slova. Takový vytvořený polynom se nazývá syndrom $S(x)$ a přímo charakterizuje chybový vektor $E(x)$. Operace pro výpočet syndromů $S(x)$ lze také chápat jako provádění rychlé Fourierovy transformace nad konečným Galoisovo tělesem, CFFT (Cyclotomic Fast Fourier Transform).

Díky nalezeným syndromům $S(x)$ lze najít pozice a velikosti chyb na těchto pozicích ve třech komponentách. Děšifrování tedy pokračuje hledáním dvou polynomů, a to polynomu pro lokalizaci chyb $\Lambda(x)$, tzv. lokátoru chyb, a vektoru vyhodnocujícího velikosti chyb $\Omega(x)$. Pro jejich nalezení byl použit rozšířený Euklidův algoritmus, který obecně provádí hledání největšího společného dělitele (NSD) dvou polynomů $S(x)$, x^{2t}. V procesu dekódování je Euklidův algoritmus použit pro řešení tzv. klíčové rovnice (anglicky Key equation), která řeší polynom $\Omega(x)$. Tuto tzv. Klíčovou rovnici lze upravit do tvaru pro Euklidův procesor, čili $S(x) \times \Lambda(x) + F(x) \times x^{2t} = \Omega(x)$. Cílem Euklidova procesoru však není nalezení NSD, nýbrž přímo polynomů $\Omega(x)$ a $\Lambda(x)$, což je jeho hlavní výhoda. Euklidův procesor se sestává z celkem t vrstev, přičemž každá vrstva se skládá ze strany pro polynomiální dělení a násobení. Při výpočtu se v jeho každé následující vrstvě snižuje stupeň mezivýsledku polynomu $\Omega(x)$, zatímco se zvyšuje řád $\Lambda(x)$, což reflektuje počet vzniklých chyb. Euklidův algoritmus se „zastaví“, jakmile splní podmínku stupně mezivýsledku $\Omega(x) < t$. V tomto bodě lze polynom $\Lambda(x)$ vyřešit v kompoentě Chien search a signál $\Omega(x)$ přivést na vstup komponenty Forneyho algoritmu.

Komponenta Chien search řeší výpočet lokátoru chyb hrubou silou. A to tak, že dosazuje jednotlivé primitivní elementy Galoisova tělesa reprezentující jednotlivé

pozice chybového polynomu $E(x)$ a tím hledá, na kterých se bude $\Lambda(x)$ rovnat nule. Výsledný polynom má tedy stejnou šířku, jako kódové slovo. Komponenta dále počítá derivaci tohoto polynomu $\Lambda'(x)$.

Vstupy pro Forneyho algoritmus jsou tyto dva polynomy z Chien search včetně polynomu $\Omega(x)$ z Euklidova procesoru. Úkolem této komponenty je výpočet zlomku, přičemž v čitateli je $\Omega(x)$ a ve jmenovateli $\Lambda'(x)$, což vyžaduje pro polynom stupně $n-1$ využití velkého množství logických zdrojů v FPGA. A dále součin s náležitým inverzním prvkem Galoisova tělesa. V první implementaci funkční verze Forneyho algoritmu se ukázalo, že uložení různých stupňů inverzních prvků Galoisova tělesa do ROM vyústilo v dlouhou dobu fáze propojování (anglicky fáze Route), a proto bylo nutné provádět výpočet těchto mocnin přímo v hardwaru.

Jelikož navržené komponenty splňují požadavky pro časování, avšak využívají velké množství logiky, práce se bude v budoucnu orientovat směrem ke snížení využití logických zdrojů v FPGA, zejména Look-up tabulek plnící funkci paměti ROM, které jsou kritickým bodem v současného návrhu. Řešení by mohl přinést přístup sdílení jak konstantních elementů ROM paměti mezi komponentami systému RS-FEC, tak registrů, zejména v Euklidově procesoru. Vhodným řešením pro minimalizaci využití zdrojů ve Forneyho algoritmu může také přinést přístup selekce jednotlivých t chybových pozic v komponentě Chien search, čímž se Forneyho vzorec použije místo n-krát pouze t-krát. Toto však bude vyžadovat vytvoření odpovídající logiky a zpoždění výpočtu o jeden hodinový takt.

DECLARATION

I declare that I have written the Master's Thesis titled "RS-FEC layer implementation for 400 Gb/s Ethernet" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENTS

I would like to express my gratitude to the thesis supervisor *Ing. Petr Vyroubal, Ph.D.* at the Faculty of Electrical Engineering and Communication at the Brno University of Technology (BUT) for his availability and leadership in the right direction. The key contribution to the existence of this thesis is thanks to *Ing. Jiří Matoušek, Ph.D.* at the Faculty of Information Technology BUT for his trust and assigning this topic to me which evoked a constant enthusiasm in me. I also want to thank to my thesis advisor *Ing. Štěpán Friedl* at the academical institution Cesnet s. z. p. o. for his professional guidance and his availability.

Finally, I am also very grateful for the support of my dear family and my dear friends, especially to *Bc. David Houška*, for providing me support and encouragement through the process of development, research and writing this paper. Thank you for such a wonderful contribution.

Contents

List of Figures

List of Tables

Introduction

In recent decades, since the communication technologies have become widespread, all the industry sectors and businesses require increasing amount of data to remain agile and innovative. Technologies ensuring real-time data processing and fast data transferring became an essential part of the world today. On the other hand, while talking about large data transferring, such changing dynamics of public demands requires also high stability of data transfer. Therefore, one of the the main purposes of Reed Solomon Forward Error Correction (RS-FEC) algorithm is to ensure error-free digital data transfer. Thanks to the RS-FEC layer, balance between high performance, efficiency and reliability of digital data transfer can be achieved to reduce noise effects not only in high-speed Ethernet platforms [18, 19]. In particular, this system will be employed within a new NDK platform (Netcope Development Kit) which is currently in development at the academic organisation Cesnet, z. s. p. o.

Error correction methods are used on daily basis by all of us. For instance, by putting emphasis on whether an information sent has been successfully received and processed, simply by repeating the message over and over again. This repetition is a form of the error correction encoding. The principle is similar in RS-FEC, based on attaching redundant parity-check symbols to the message sent to the encoder part of the error correction system. Therefore, the system uses this redundancy for erroneous data correction at the error correction decoder. The purpose of adding the decoder part (the error correction capability) is also to avoid decoding some other message [19]. The principle of error correction coding within the RS-FEC layer is based on attaching parity-check symbols as a redundant part to the message received instead of repeating the whole message again. Subsequently, the decoder, the unit ensuring the confidence level of the system, correctly extracts the original source signal out of the corrupted data on the input of the decoder [19].

Employing RS-FEC layer seemed to be an effective solution for reaching 100 Gbps rates, however, its full error correction capability has still not been fully exploited yet, even for the new 400GE. In terms of the RS-FEC system, the main difference between these two standards is that the new 200/400 Gbps rates operate with two codewords in parallel compared to only single one for 100 Gbps rates. Therefore, the most challenging aspect for successful implementation of this system is to balance and minimize logic resources utilization. The main reason for RS-FEC system implementation into an FPGA as firmware is clearly the capability of conducting hardware operations in an FPGA concurrently enabling high-speed computations, possibly at 400 Gbps rate which is the main topic of this paper [19, 11]. In addition, VLSI design enables its development in a generic form for faster development of future platforms and creating different variants of the system.

1 Theoretical Part

In the theoretical part of this work, FPGA technology is briefly described, PLD (Programmable Logic Devices) manufacturing and RS error correction codes is characterized. Also, one chapter dealing with Galois field algebra is included and RS error correction flow from the mathematical point of view is described.

1.1 Brief History of Ethernet

Since its introduction in the early 1980s, Ethernet has become a dominant and popular protocol for Local-Area-Networks (LANs), used mostly in offices. Over the following years, demands for higher data-rates of Ethernet began to rise enormously. The first experimental 2,94 Mbps shared bus-based system was able to transmit with only one station at a time. So called Medium Access Control (MAC) protocol detecting collisions controls the use of the shared bus. Each station is free to transmit MAC frames but if a collision occurs during transmission, it stops for certain amount of time and tries again if the channel is free for transmitting. The first commercially available standards were bus-based systems capable of 10 Mbps operation. There were no changes to the MAC protocol or MAC frame format. But, the innovation was that Ethernet was configured in a star topology which enabled traffic to go through a central hub, however, again with transmission limited through the hub. The need for faster data-rates resulted in the central hub replacement by a switch allowing full-duplex operation. Thanks to this, with the switch and MAC format protocol unchanged, collision detection is no longer needed. Further enhancements to the MAC layer were added through time to improve data rate requirements, such as provision for larger frame size. Ethernet quickly achieved widespread attention and acceptance and became a dominant technology. Not only for LANs, but also Metropolitan-Area Networks(MANs) and spread also to a wide range of applications and environments due to its extraordinary adaptability[20].

The same MAC protocol and frame format are used at all data rates. The main differences among various standardizations for different data transfer speeds are at physical layer in the definition of signal transmission medium of Ethernet [20].

Historical perspective of the first Ethernet platforms did help in the initiation higher speeds of Ethernet development beyond 100GE. The bandwidth explosion was (and still is) driven by increasing number of users, increased access methodologies, access rates and increased number of services (such as social media, video on demand, etc.). From 2000 to 2019 around 3.1 billion individual users were connected to the Internet. [1].

Nowadays, the most data-intensive sectors with the most significant growth rates of data traffic are financial, data-intensive science and peering. Slower growth rates have been estimated for cable users and end-stations, such as IP traffic and servers I/O. [21].

1.2 OSI Reference Model

The very first primary definition of modern networking was approved by the International Organisation for Standardisation in 1984. The OSI (Open Systems Interconnection) Reference model can be perceived as a core of serial networking technologies, including industrial Ethernet. It is a layered description of data transfer among devices within a network [22].

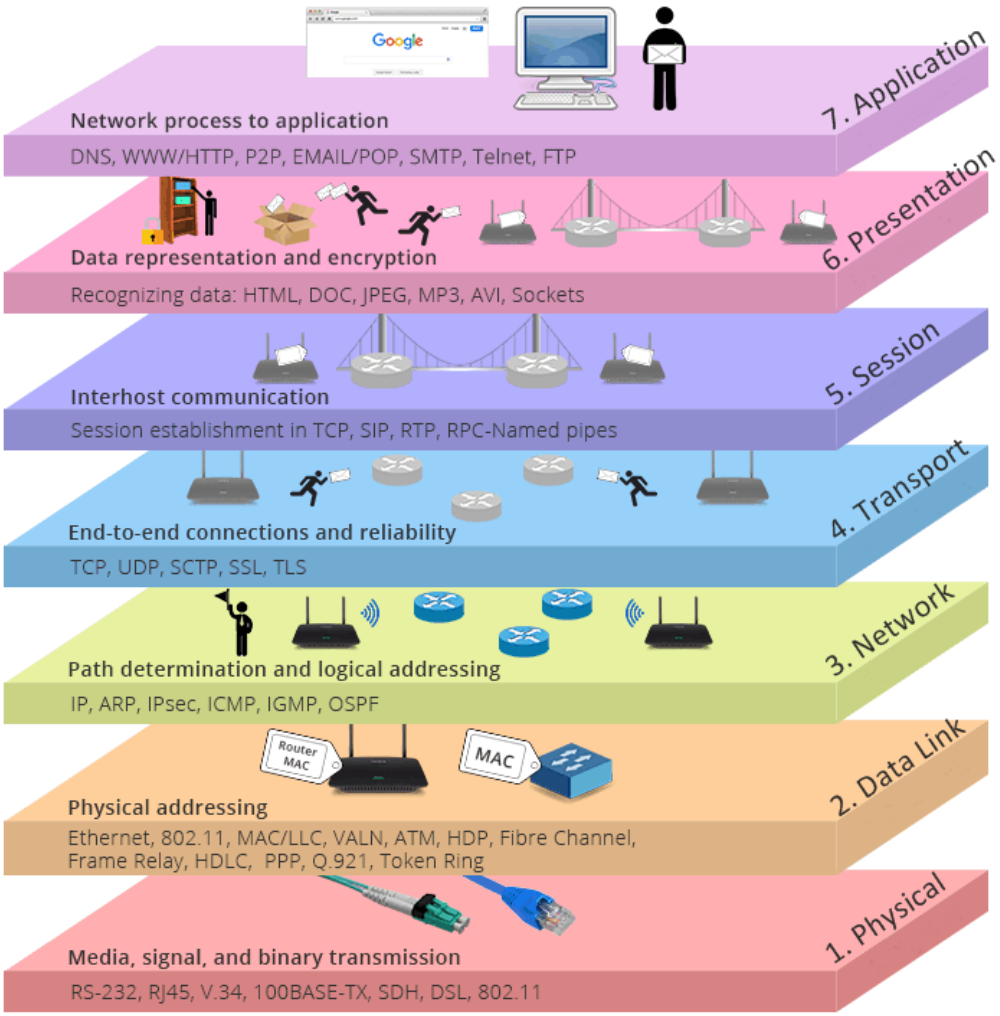

Fig. 1.1: Seven layers of OSI Reference model (taken from [5])

The uppermost layer, number 7, is called *Application Layer* and is the closest to end users. It directly interacts with users' software applications to provide desired communication functions [22].

Layer 6 is called *Presentation Layer* which provides end user data translation to network format so that lower layers can accept the data [22].

Session Layer is the fifth layer which manages connections between respective remote and local computers and also terminates connections between them. It also conducts data verification procedures if data have been delivered correctly or not [22].

Layer 4 is called *Transport Layer* ensuring complete delivery of data usually by using error correction functions or by other means. Sequences of data from are being transferred from a source to a destination host via network [22].

Layer 3, *Network Layer*, creates logical paths using switching functions for data transmission from node to node so that network can be formed from the node of the transmitter side to the address node of the desired destination. [22].

Layer 2, so called *Data Link Layer* (DLL) allows direct node-to-node data transfer. On this level, data are packed into frames based on Point-to-Point Protocol and encoded into single bits and further unpacked. The layer is divided into two sublayers: $MediaAccessControl$ (MAC) controls which device in a network will be permitted to transmit data to a media. The second sublayer is $LogicalLinkControl$ (LLC) conducting frame synchronization, network layer protocols identification. LLC is also used to control data flow and checks for errors [22].

The very bottom layer is *Physical* (PHY) which is responsible for conveying unstructured raw data bitstream at the electrical level and defines the physical specifications of the data connection, such as optical fibre specifications, operation voltages of an electrical fibre, layout pins of the connector etc. PHY and MAC layers are interconnected via MII interface [22].

Conventional Ethernet PHY consists of three additional sublayers: Physical Coding Sublayer (PCS), Physical Medium Attachment (PMA) and Physical Media Dependent (PMD) sublayer. PCS is responsible for interfacing to the higher layer MAC through MMI (Media-Independent Interface) interface. PMD sublayer specifies optoelectronic components and, if required, implements digital signal processing on the transmittted and/or received signal. PMA sublayer conducts multiplexing of n physical lanes to x PCS lanes and backwards [17, 23].

1.3 Forward Error Correction and Current State of Ethernet Speeds

The very first implementations of RS (Reed-Solomon) error correction algorithms were first available since the Voyager deep space communication system in the 1977. [19, 11]. Nowadays, RS codes can be found in various applications, such as radio and television transmissions, disk storage, high-speed computer memory I/O and data communication technologies. For instance, high-speed Ethernet. FEC was first introduced for backplane and then, to deliver more economical optical transceivers and cable technologies, for a few front-panel use, such as copper cabling for 100GE, 50GE and 25GE for the purpose of error-free data transmissions, however, for the penalty of carrying additional bits for the FEC mechanism to encode, transmit, decode and correct the data packet re-transmissions [14].

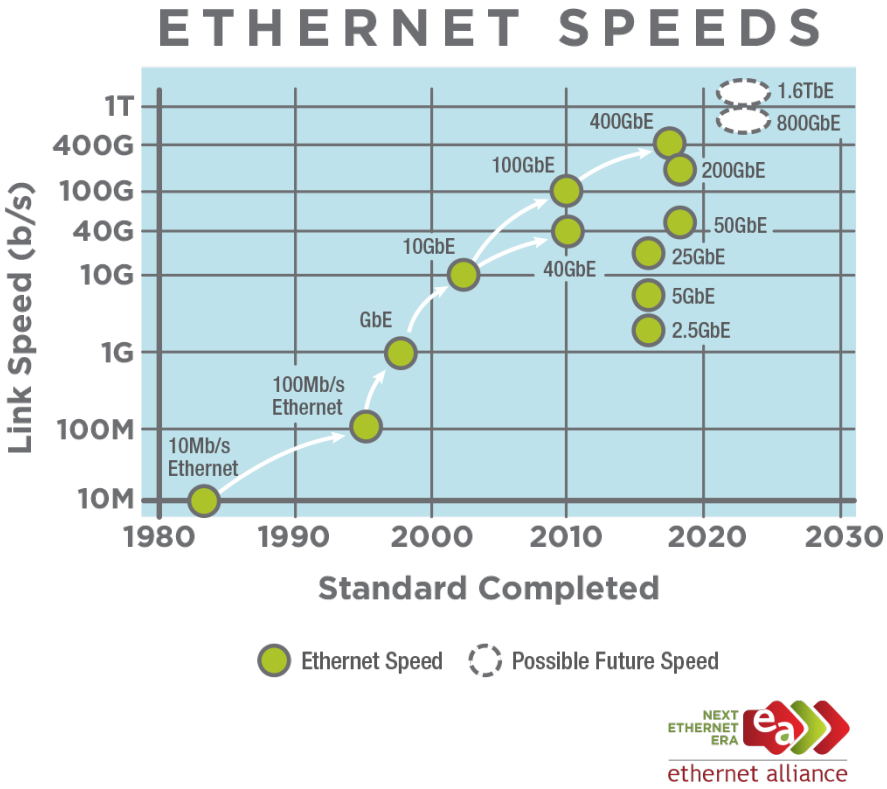

Fig. 1.2: Graph of current state and future projections of Ethernet speeds (taken from [6])

In recent years, Ethernet protocol is undergoing significant development. It is capable of achieving 100GE, 200GE and 400GE speeds. Trends of current speeds of Ethernet are shown in figures 1.2 and 1.3 [14, 6].

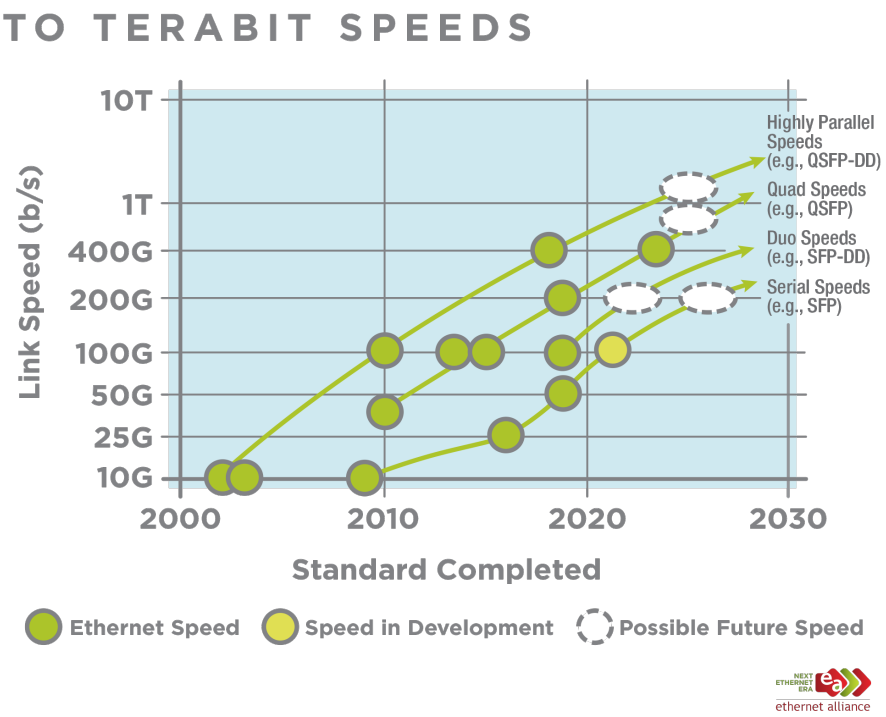

Fig. 1.3: Current state and future projections of standards completion for various forms of Ethernet (taken from [6])

The FEC system is required for such high throughput because optical transceivers, electrical interfaces and cables are noisy signalling environments. So the bit error rate (BER) which these electronics generate itself require an algorithm-based error-correction method. RS-FEC uses an approach of finite-sized block of bits known as a block code. In Ethernet it is called a message. RS-FEC is a cyclic type of FEC despite it works as a linear code, meaning with fixed block of bits where a FEC symbol is 10 bits in size. It is known also as RS 544 FEC. In the following table 1.1, details of RS-FEC for Ethernet implementations are summarized [14]:

Tab. 1.1: RS-FEC (544, 514) for 200GE/400GE Specifications (taken from [14])

Symbols	Explanation
514	Total number of symbols in a codeword
$514 \cdot 10 = 5{,}440$	Bits in the codeword or block
544 - 514 = 30	Number of check symbols per codeword or block
$514 \cdot 10 = 5{,}140$	Number of bits referring to the size of the information bits per block
$\frac{(544-514)}{2}$	Maximum number of symbols which can be corrected in a codeword or in a block

1.4 Programmable Logic Devices Fabrication

FPGA chips belong to the family of active semiconductor devices. Such devices require extremely pure silicon and germanium in lesser extent. Intrinsic silicon is much more difficult to prepare than intrinsic germanium. The pure form of silicon needs about $1.5 \cdot 10^{10}$ of intrinsic carriers per cm^3 and $2.4 \cdot 10^{13}$ per cm^3 for germanium. Silicon is obtained from silicon dioxide or silicon tetrachloride by normal metallurgical processes and needs to be further purified until the number of foreign atoms is less than 1 in $1 \cdot 10^{10}$ per cm^3 to create silicon pure enough for semiconductor devices. The most frequent method is the Czochralski crystal pulling, shown in figure 1.4 [7]. This method is based on seed crystal insertion into a bath of molten silicon.

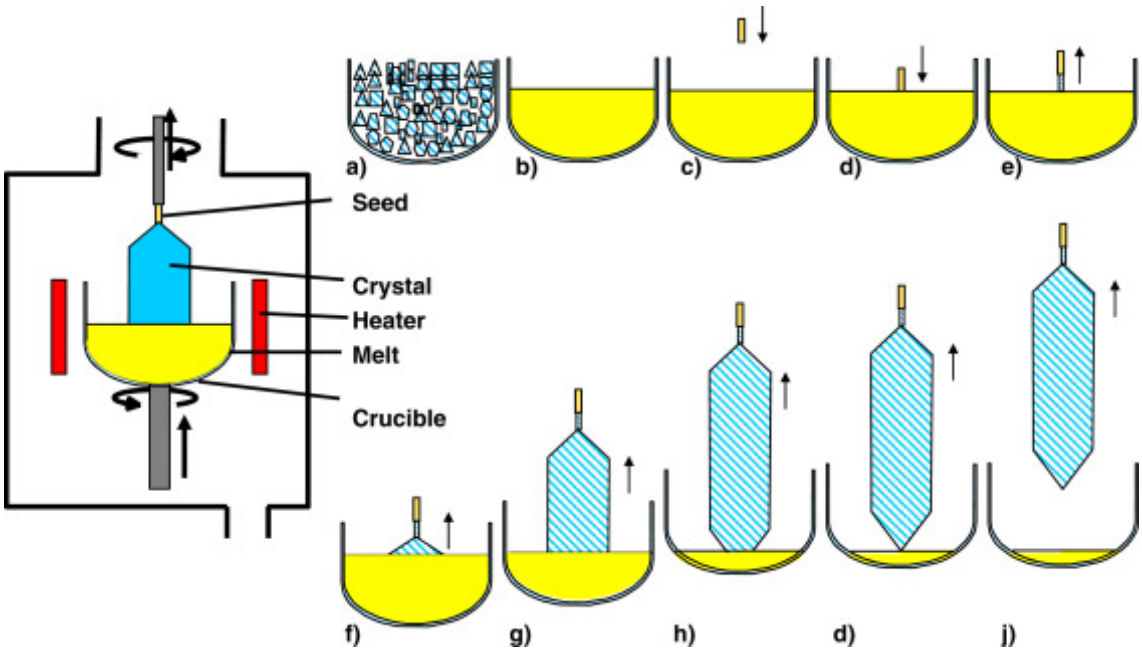

Fig. 1.4: Process of Czochralski crystal pulling: (a) melted polycrystalline silicon (b) in a crucible. (c, d) Seeding procedure: The seed crystal dipped into the melt, followed by Dash necking (e), shouldering (f), cylindrical growth (g), growth of end cone (h), lift off (i), cooling and removing the crystal (j) (taken from [7])

A single silicon crystal of 10 to 15 cm with required impurities can be obtained by being withdrawn from the molten bath. In the next step the cooled crystal is being divided into 1 mm thick slices along the crystallographic direction of the crystal in order to avoid internal structure disruption of the crystal. Resulting slice is approximately 0,2 mm thick after the etching and polishing procedures removing surface damage after the step of slicing. This is the key step for further epitaxial layer growth with the same orientation on to the underlying sliced silicon surface [24].

In the next step, ion implantation to the pure silicon crystal lattice with impurities is performed. In this step, the energy of accelerated ion implants up to 300 keV, determining the depth of implanting, are bombarded into the silicon substrate [24].

Next and the most important stage of the integrated circuit fabrication process is epitaxial deposition. It involves an epitaxial layer growth on the slice of the silicon dioxide. The layer is grown in the atmosphere of silicon tetrachloride and hydrogen respectively with strictly controlled conditions. A perfect crystal is the key requirement for correct outputs of subsequent stages [24].

The first stage after the epitaxial deposition is oxidization which is used two or three times in order to create a mask for the impurity atoms diffusion after selective etching. One method of oxidization used for oxide layer creation of high quality physical properties is based on passing oxygen over the surface of silicon slice at a temperature of 1200 °C, as illustrated in 1.5. As the result, passivizing layer with uniform thickness of $5 \cdot 10^{-7}$ m is created. Oxide layers have been used also as elements in active and passive devices and silicon functional blocks. Oxide layer becomes important especially during planar and epitaxial planar transistors manufacturing. The reason is very low leakage currents currents due to the junctions formed under A layer of silicon oxide. In this way, 10 silicon slices with 200 monolithic circuits in each would be produced [24]. Next step is photo-engraving process

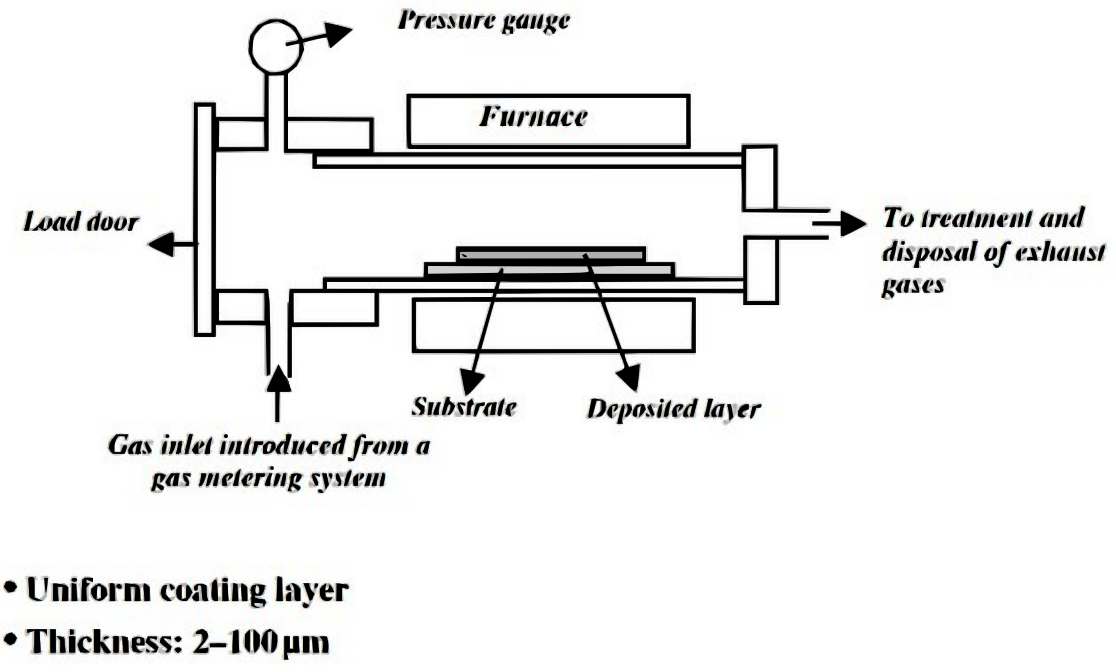

Fig. 1.5: Illustration of the process of Chemical Vapour Deposition (CVD) (taken from [8])

which involves two operations: photographic mask preparation and the etching of the

silicon dioxide. The purpose of this step is to cut off windows allowing the diffusion of subsequent stages to take place. Next step is the photographic mask production. Each one comprises of large number of identical elements, each of which is the original mask layout. Due to the photographic equipment limitations, sequence of stages for reduction are required. A typical sequence involves artwork originals preparation, photographic reduction, step-by-step contact printing and rephotographing. Subsequently, when the photographic mask has been prepared, with photo etching the processing of the slice may be started, illustrated in figure 1.6. The former part, the centrifugal force spreads the liquid photoresist dropped on a rotating surface of the slice at 800 rpm and subsequently is let to dry in an oven. In the latter part, as the photoresist is placed in the exact position required and subsequently exposed to UV (Ultraviolet) light. The UV light causes the photoresist to polymerize with the opaque layer. The rest of the photoresist unexposed to the UV light is then removed. The polymerized photoresist forms a layer resistant to hydrofluoric acid used to etch the silicon base away [24]. The further step of of integrated circuit

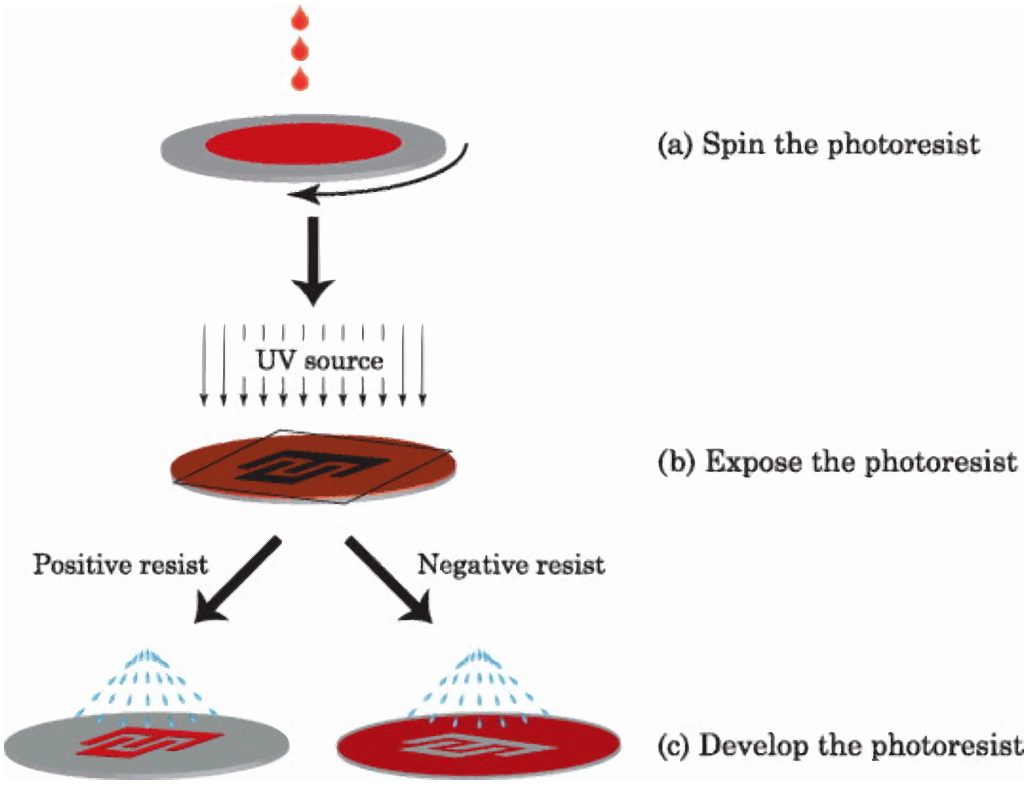

Fig. 1.6: Process of transferring a pattern onto a substrate. (a) Coating the substrate with a photosensitive material; (b) alignment of the mask and exposure to the UV light source; (c) spraying the photoresist to remove the extra photoresist defined by the mask patterns (taken from [9]).

fabrication is diffusion which consists of combination of epitaxial deposition and

diffusion. The diffusion process takes place within the holes etched in the silicon dioxide. During the process of diffusion the time and concentration of the impurities must be accurately controlled in order to obtain specific diffusion depths according to the required transistor design. The choice of an element of diffusant must meet the requirement for easy diffusion into the intrinsic silicon but not into the silicon dioxide. For instance, boron and phosphorus are the usual diffusants. The next step, the process of evaporation is to be conducted which is important for ohmic contact production and interconnections realization. This process takes place in vacuum with golden, nickel or also aluminium rods being evaporated. As the result, a thin layer over the entire surface is produced. The main issue of the process is to avoid changing the desired nature of the semiconductors when alumina is added. Therefore, with masking and etching only the desired alumina configuration remains to form the contacts and interconnections [24].

As a slice of semiconductor has been developed, the very last step is cutting into individual circuits and packaging. Cutting can be performed by using a diamond-tipped tool by drawing it across the edge of the surface of the slice. Subsequently, by breaking each part separate chips are produced. Individual chips are ready for mounting and encapsulation [24].

1.5 Field Programmable Gate Arrays

Typical hierarchical structure of modern FPGA chips consists of programmable logic blocks further containing pool of combinatorial logic blocks and flip-flops to be used in an intended design. These logic elements are often combined with memory, typically with various amount of SRAM (Static Random Access Memory) inside an FPGA chip. This typical architecture is shown in the figure 1.7 which contains so called CLB (Configurable Logic Block) units interconnected within a matrix-like grid and surrounded by programmable interconnect. Each CLB typically consists of a set of so called BLE (Basic Logic Element) units. Inside a single BLE there is an element allowing logical function implementation called LUT (Look-up table) [11, 10, 25].

More detailed view on a typical high-level hierarchical structure of an FPGA is shown in figure 1.8. CLBs form a large array including BRAMs (Block RAMs) and DSP (Digital Signal Processing) blocks, similar to arithmetic logic units (ALU) of a processor which can be programmed accordingly to perform arithmetic logic operations, such as add, multiply, subtract, compare, etc. Depending on the type of operators required, both CLBs and DSPs can perform integer, floating point or/and bitwise operations. Results of these operations are stored in registers present in

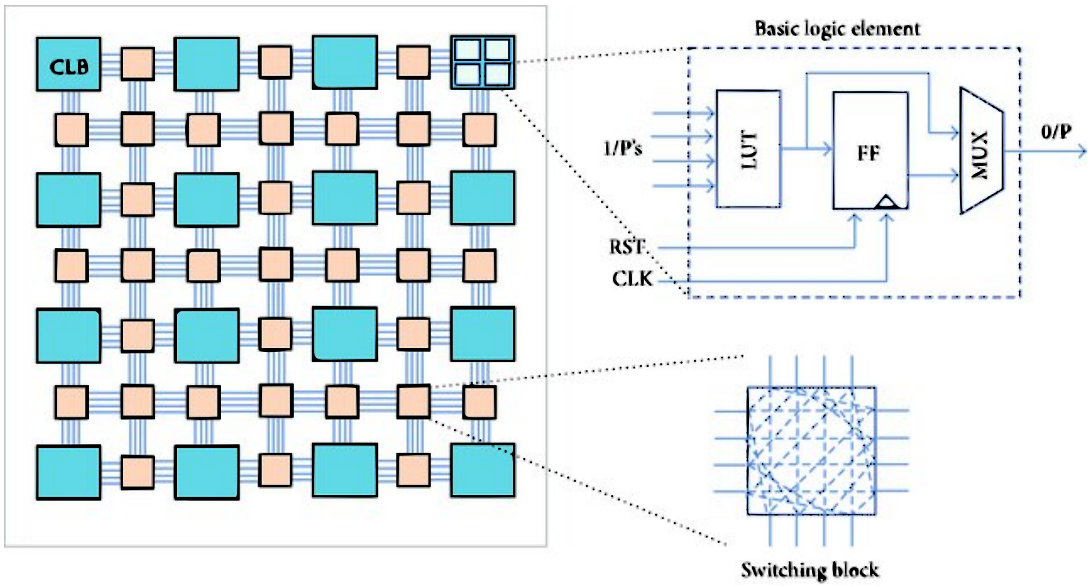

Fig. 1.7: Structure of the typical SRAM-based FPGA (taken from [10]).

CLBs, DSPs or/and BRAMs. These blocks can be connected via flexible configurable interconnects which are based on user design. The output of one operator can directly flow into the input of the next operator [16].

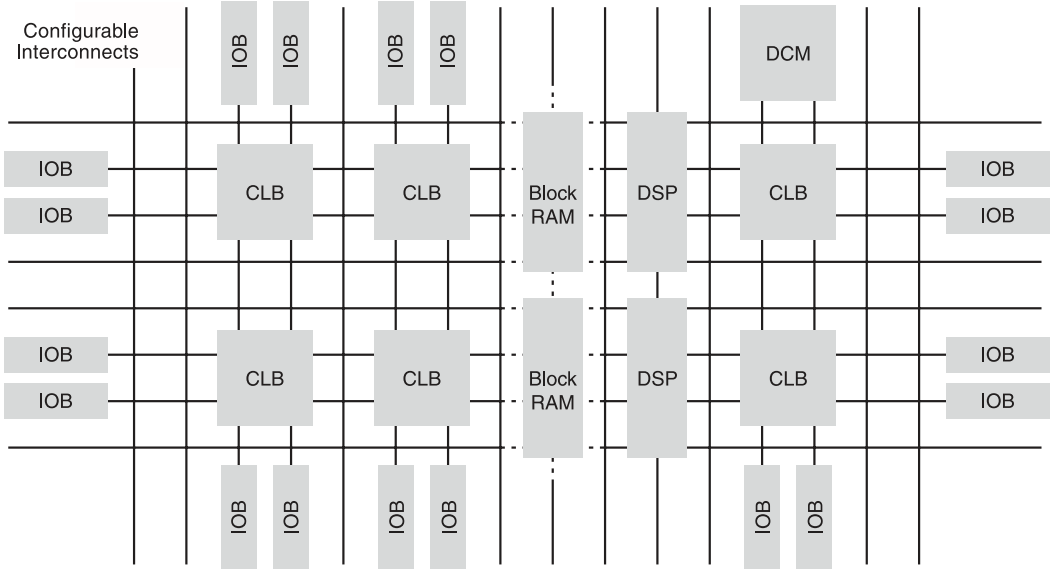

Fig. 1.8: High-level structure of the typical SRAM-based FPGA (taken from [10]).

The architecture then enables to create a massive array of application-specific ALUs which allow both instruction and data-level parallelism. Compared to processor units, there are no inefficiencies, such as processor cache, but data within an FPGA can be directly streamed between operators. These operators can be

configured to have point-to-point dedicated interconnects, thereby setting them to pipelined configuration. For instance, throughput on integer operations are in order of Tera-operations per second, on floating point operations in order of gigaflops per second [16].

Another great advantage of FPGAs is that they can be easily interfaced to other chips or external signals by so called input/output blocks (IOBs) (see figure 1.8) behind the chip pads. So that each pad can serve as an input or an output or both. In particular, IOBs are designed to support various memory and processor-interface standards, such as support of multiple DDR3 (Double Data Rate Type 3), DDR4 (Double Data Rate Type 4) and more memory controllers, various generations of PCI Express® (Peripheral Component Interface), Intel's Front Side Bus (FSB), Quick Path Interconnect (QPI) protocols. Support for these processor interfaces and protocols enables computing applications running on FPGAs to interact with processor and accelerate the desired applications [16].

FPGA vendors also include hardwired IP (Intellectual Property) nonprogrammable cores inside the chips supporting commons recurrent functions in many designs[11]. These include general-purpose processors, high-speed serial interfaces, arithmetic blocks and Ethernet MAC (Medium Access Control) [11].

1.5.1 Pipelining

Clock conditioning has become also a common feature in FPGAs. Digital circuits can be also supplied by a clock signal which is, ideally, a simple square wave oscillating at a certain fixed frequency. The most basic concept of a sequential system in an FPGA chip contains number of combinatorial logic blocks in between arrays of clock-sensitive components called flip-flops where current state of outputs of combinatorial blocks depends on current state of inputs to these combinatorial blocks. These are generally made of all logic functions with any level of intended complexity realized by logic gates including interconnection among them. This involves also multiplexors, encoders and decoders [11].

For instance, a D-type flip-flop (DFF). Every time there is a rising edge of the clock signal, it allows desired signals to propagate from its input D to its output Q. In this particular moment the input D is connected to the output Q in a single DFF. However, apart from this specific time D is disconnected from Q [11].

An important requirement for the resultant sequential system is that signals between the output Q of the first flip-flop and the input D of the next one must remain stable by the time the next clock cycle rising edge. It is therefore required to ensure the worst case propagation delay between these delay elements of the design. This also applies for all subsequent DFFs. Nowadays, this timing check is

automated so that the designer needs to be concerned with the specification of the logic behaviour of the circuit [11].

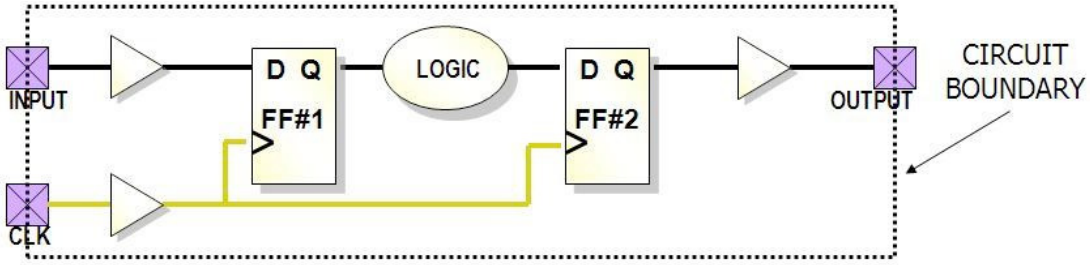

Fig. 1.9: Basic concept of a sequential circuit (taken from [11])

This technique is generally known as pipelining. Data throughput is one of the most important parameters not only in this work. The purpose of pipelining is to satisfy throughput requirements also with minimum resource penalty. Algorithms which can be performed in parallel with sequential delay elements (such as DFFs) result in higher throughput, such as in digital processing (DSP) applications, multi-core CPU (Central Processing Unit) parallel platforms, many-core GPU (Graphics Processing Unit) and FPGAs compared to traditional single-core systems. Among all these parallel platforms, FPGA-based systems allow the highest flexibility for programming parallel cores. This can be achieved thanks to the high-level synthesis which significantly increases productivity, reduces time-to-market window and helps to implement efficient parallel hardware of complex register transfer level (RTL) designs. The objective of the synthesis is to find a suitable performance solution for a design with given available resources [26].

Pipeline optimization strategy is based on partitioning this large scale designs into smaller data processing elements connected in series while each element (combinatorial block) executes its operation in parallel in a time-sliced mode. This requires some buffer storage (pipeline registers). The registered output of one element becomes the input of the next one. The time between each clock signal is set to be greater than the longest delay between pipeline stages, so that when the registers are clocked data written to the following registers are results of the previous stage of the pipeline. Pipelined systems also requires more resources then the combinatorial logic elements because each pipeline stage cannot reuse resources of other stages [26].

The key pipeline parameters are number of pipeline stages, latency, clock cycle time, delay, throughput and turnaround time. A pipeline synthesis can be constrained by resources or time, or combination of both. A resource-constraint synthesis pipeline limits the area on chip or available number of functional units

on target device. A time-constraint pipeline synthesis puts more effort to required throughput and turnaround time. Finding a solution which consumes minimum resources is the task for the so called scheduler [26]. Reasons why FPGAs are chosen as promising platforms for high-performance data-intensive applications are summarized in table 1.2 including their drawbacks [15, 16]. Compared to GPUs and other multicores which consume power in hundreds of watts, FPGAs power consumption lies in the range of tens of watts [27] [16].

Tab. 1.2: Positives and downsides of FPGAs (taken from [15])

Advantages	Disadvantages
Massive parallelism of compute operations which can be put to more optimal configurations	Processing data with constrained cost and resources.
Flexibility in terms of involvement of different kinds of components (hard cores, IP, memory, LUT structure of the programmable fabric). Second, ability to field reprogram parts or the entire FPGA chip.	Not clear what should stay as software part and what hardware part of a desired complex system. Interfacing between these two approaches requires additional development.
Flexibility in terms of an ability to field reprogram parts or the entire FPGA chip.	Low efficiency of data movement around the chip.
Small amount of distributed memory incorporated into the fabric which brings the memory closer to the processing	IP library is required for FPGA-based systems development.
Low power solution enabling more processing than GPUs for quarter of power required	Design entry methodology is lacking more restricted approach to harness the flexibility of the hardware.
Scalability in terms of creating a chain of FPGA chips together while the algorithm is larger than the single one.	Strict rules of what is synthetizable and what is not.
Nowadays, custom chips deliver more data throughput per dollar.	Innovation is required in the area of high-performance interfacing to get large amount of data onto the chip.

The major reason for lower power consumption in FPGAs is that these devices operate in range of 100-300 MHz compared to processors executing operations usually between 2-3 GHz. Recently, in terms of high-end FPGA devices such as Intel® Stratix® DX, programmable clock-tree performance reaches around 1GHz [27] [16].

1.5.2 Typical Applications

In the mid-2000s the high performance computing industry (HPC) demand caused course of General-purpose CPU vendors to shift from single-core CPU-based systems orientation to multicore architectures to meet high-performance demands of the industry. The reason for this is that if frequency of single-core processors increases, power dissipation rises to impractical levels. The result of this is that it enables to exploit CPU performance by adopting parallel designs enabling previously unattainable performance levels [16].

There is a broad spectrum of applications where FPGAs embedded inside equipment or forming a massive compute server farms play major role. In table 1.3 applications for High Performance Servers are shown. These applications are in constant need of compute power. The greater the computation power, the more complex algorithms can be implemented to produce more accurate results [16]. In table 1.4

Tab. 1.3: Applications of FPGAs for High Performance Servers (taken from [16])

Industry	Sample Applications
Government labs	Climate modelling, nuclear waste simulation, warfare modelling, disease modelling and research, aircraft and spacecraft modelling
Defense	Video, audio, data mining, analysis for threat monitoring, pattern matching, image analysis for target recognition
Financial services	Options valuation, risk analysis of assets
Geo-sciences and engineering	Seismic modelling and analysis, reservoir simulation
Life sciences	Gene encoding and matching, drug modelling and discovery

sample applications for High-Performance Embedded Computers are shown. All industries mentioned require certain specific equipment for compute-intensive and data-intensive tasks. In the past, these systems were based on custom integrated circuits designed for high memory rates and to handle data-intensive processing [16].

Tab. 1.4: Applications of FPGAs for High Performance Embedded Computers (taken from [16])

Industry	Sample Applications
Defense	Beam forming in radar
Airborne Electronics	Image compression and analysis in payload
Communications	Encryption in network routers
Medical Imaging	Image rendering
Financial Services	Low latency and high throughput data processing in trading solutions

1.6 Intel® Stratix® 10

Since the task of this work is to implement the system RS-FEC into FPGA Intel® Stratix® 10 DX 2100, this chapter therefore deals with the technology present in Intel® Stratix® 10 family.

SoC (System on Chip) devices of this family dispose of the Intel HyperFlex FPGA Architecture combined with 14 nm Tri-Gate (3D) process technology (see the structure in the figure 1.10) which replaced the conventional 2D planar MOSFET transistors so that geometries have been reduced below 20 nm. It contains so called Hyper-Registers present all over the functional blocks within the chip. The advantage of this technology is that all the conventional blocks such as Adaptable Logic Modules ALMs, embedded memory (M20K) and digital signal processing allow to select the optimal register location automatically after place-and-route to maximize core performance without additional changes or added complexity after the place-and-route step of the design process. The next advantage is that such registers reduce routing congestion [25]. Another useful feature of the chip is the

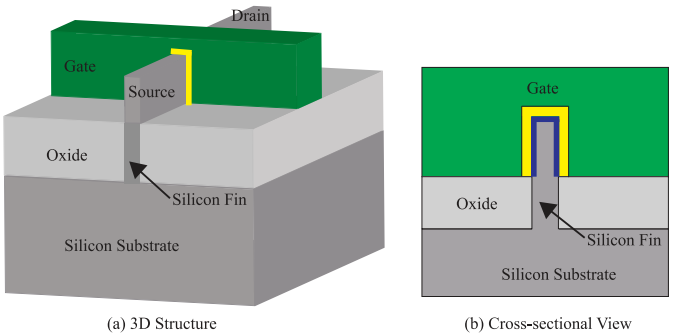

Fig. 1.10: Structure of Tri-gate 3D Fin-Fet by Intel® (taken from [[12]])

programmable clock tree synthesis. It reduces timing and skew uncertainty to reach the maximum core clock performance. This feature enables its entire architecture to

double its performance compared to its predecessors Stratix V FPGAs. Core clocking also uses intelligent branch which allows to reduce dynamic power dissipation in the clock networks [25].

The Hyper-Aware design flow includes a Fast Forward Compile tool which enables performance exploration and guides the designer to the maximum performance of his solution. A Hyper-Retimer step near the end of the design offers further optimization after place-and-route step. An enhanced synthesis and place-and-route algorithms which use the Hyper-Registers. In the end, it uses 70 % less power than Stratix V FPGAs predcessors. There is an embedded quad-core 64-bit ARM Cortex-A53 processor system included and also components DRAM (Dynamic Random Access Memory), SRAM (Static Random Access Memory) and ASICs in a single package [25].

1.7 Reed-Solomon Error Correction Codes

Reed-Solomon codes belong to the category of block codes. This means that a message to be transmitted to the divider of n symbols is divided into separate block of data called codeword. The former part of a single codeword is an original message consisting of k information symbols in a message to be transmitted.

In the latter part a parity protection of $(n - k) = 2t$ symbols is added to the original message. The error-correction capability of RS codes is determined by its Hamming distance which is determined by number of parities. For RS codes, its Hamming distance is $2t + 1$ and the overall error-correcting capability is [13]:

$$error\text{–}correcting\ capability = \frac{Hamming\ distance}{2} \tag{1.1}$$

The variable t specifies the number of symbols the algorithm is able to correct in a block of n symbols of the codeword. This is illustrated in the figure 1.11 [3]. So that each block of information symbols has its own parity protection added as a separate part of the codeword.

In addition, RS code is also a linear code. This means that sum of any two codewords is still a valid codeword [13]. It is also a cyclic code meaning that cyclically shifting the symbols of a codeword produces another one. RS code can be therefore described as an (n, k) code. Different parameters for a code provides different levels of protection and complexity of the implementation changes respectively [3].

There is a significant advantage of RS code. It enables having all bits of a symbol of m in error and it counts as only one symbol error in terms of the correction capacity of the code [3].

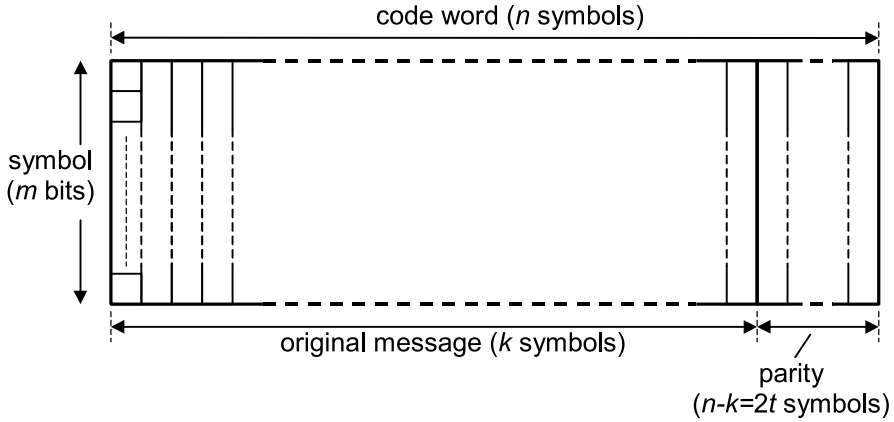

Fig. 1.11: RS code definitions (taken from [3])

1.7.1 Galois Field

Galois field belongs to the family of finite fields and named after the French mathematician Évariste Galois. A Galois field consists of a set of elements based on a primitive element α which takes values $\alpha, \alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{N-1}$ to form a set of 2^m elements where $N = 2^m - 1$. Galois fields are then marked as $GF(2^m)$. Each field element can be also represented by polynomial expression [3]:

$$a_{m-1}x^{m-1} + \ldots + a_1x^0, \tag{1.2}$$

where coefficients α_{m-1} to α_0 take values of 0 or 1. Therefore, it is possible to describe a single field element by the binary number $\alpha_{m-1}, \ldots, \alpha_1, \alpha_0$. There is in total 2^m combinations of the $m-$ bit number [3].

1.7.2 Galois Field Mathematics

Arithmetic operations with finite field elements differ from conventional mathematics with normal integers, especially while multiplying in a Galois field. Galois field arithmetical operations are addition, subtraction, multiplication and division. The difference is that any arithmetical operation of two field elements always produces another field element [3].

Addition and Subtraction

While adding two Galois field elements, two polynomials are added in this form [3]:

$$(a_{m-1}x^{m-1} + \ldots + a_1x^0) + (b_{m-1}x^{m-1} + \ldots + b_1x^0) = c_{m-1}x^{m-1} + \ldots + c_1x^0 \tag{1.3}$$

This operation $a_i + b_i = c_i$ applies for degrees $0 \leq i \leq m-1$. The coefficients can only take the values 0 and 1. If $a_i = b_i$, then c_i produces 0. Respectively, if

$a_i \neq b_i$, then c_i produces 1. This signifies that addition of two Galois field elements is accomplished by modulo-two addition of their respective coefficients. In binary form, addition is realized by the exclusive-OR function of two binary numbers. Therefore, addition of two identical Galois field elements produces zero. It also implies that any result of subtraction of two Galois field elements from each other is exactly the same as addition. In the end, minus sign can be replaced with plus sign. In other words, if a positive element in an equation is needed to be expressed on the other side of the equation, the sign stands the same [3].

Multiplication and Division

There is a significant difference between multiplication with standard integers and multiplication in a Galois field. The main difference is that if polynomials of degree $m-1$ are multiplied, the result is a product polynomial of degree $2m-2$ which is not a valid element of GF(2^m). With the same approach as with subtraction, for a result of multiplication product modulo is required. In Galois field, the valid field element is obtained by dividing the product product polynomial by a field generator polynomial $p(x)$ in order to "return" the value of the straightforward multiplication of the polynomial back to a valid Galois field element [3].

Division of two field elements of the Galois field is accomplished by multiplying the inverse of the divisor. The inverse element is defined as when the element value is multiplied by the inverse field element, value of 1 is produced [3].

The Field Generator Polynomial

The field generator polynomial or primitive polynomial $p(x)$ of degree m defines a specific finite field bound to it. When a different generator polynomial or primitive polynomial is selected, it produces different results. Therefore, one generator polynomial or primitive polynomial must be selected for a single Galois field. The next requirement for a generator polynomial or a primitive polynomial is that it must be irreducible (with no factors of the GF) [3].

Based on the primitive element α as a root of the field generator polynomial the all non-zero values of Galois field can be generated. So that to obtain the complete field, it means that [3]:

$$p(\alpha) = 0 \tag{1.4}$$

To determine the repeating sequence of the field elements, it is needed to express the highest degree of the primitive polynomial. First, since a GF element is written in the index form, for instance α^0, it is possible to get next value α^1 by multiplying its entire polynomial α^0 in its polynomial form by α. Then, the next Galois field value is obtained. It is also important to mention that if the highest degree of the

generator polynomial $p(x)$ has been reached this way, it is needed for any member of the polynomial form to substitute this member by the expression of the highest degree of the generator polynomial. In this manner, all 2^m Galois field elements can be obtained by starting with an element with index 0 up to the element $(2^m - 2)$. It is also important to highlight that the decimal form of an element is a representation of the respective polynomial form, meaning binary form. For instance, polynomial $\alpha^2 + 1$ is equal to the representation of bits x^2 and x^0 in log. 1 giving 0101 in the 4-bit binary form. Next important characteristics of the Galois field is that if the maximum is exceeded, it can be found that the index form, for instance for the first element beyond the last one which is $\alpha^{(2^m-1)}$, is equal to element α^0, the next one equals to α^1 and so on and all these values remain valid within the desired Galois field [3].

1.7.3 The Code Generator Polynomial

While constructing an RS code, the values of the message parity symbols must be elements of a Galois field. An (n, k) RS code is constructed by the code generator polynomial $G(x)$ involving $(n - k = 2t)$ factors, the roots of consecutive elements of the Galois field. For a code based on m-bit symbols, the Galois field consists of 2^m elements [3]:

$$g(x) = (x + \alpha^b)(x + \alpha^{b+1}) \dots (x + \alpha^{b+2t-1}), \tag{1.5}$$

where b specifies at which degree the Galois field roots begin.

1.8 Reed-Solomon Encoder

The output of the encoding process comprises of two data blocks. The first block is formed by k information symbols and is represented by the message polynomial $M(x)$ of order $k - 1$. This polynomial of information symbols to be encoded can be written as follows [3]:

$$M(x) = M_{k-1}x^{k-1} + \ldots + M_1x + M_0, \tag{1.6}$$

where k is the number of symbols in a message to be transmitted, M_{k-1} is the first symbol of the message and each one $M_{k-1}, \ldots, M_1, M_0$ is an m-bit message symbol, an element of $GF(2^m)$ [3].

The key purpose of the RS encoder is to add the parity polynomial to the message polynomial $M(x)$ in order to form a valid codeword $T(x)$ to be transmitted. To encode the message polynomial $M(x)$, it has to be first multiplied by x^{n-k}. In this step, its resultant degree is extended by $(n - k)$ symbols and $M(x)$ is shifted

to the part of the higher bit significance which ensures that there will be enough number of free bits for the parity polynomial symbols in the less significant part of the resulting polynomial $T(x)$. In the subsequent part, the result must be divisible by the generator polynomial $G(x)$. After the division, a quotient $Q(x)$ is produced including a remainder $r(x)$ of degree up to $n-k-1$ [3]:

$$\frac{M(x) \times x^{n-k}}{g(x)} = q(x) + \frac{r(x)}{g(x)} \tag{1.7}$$

As described in the section 1.7.2 Multiplication and division, any result of a division operation in a Galois field is the remainder $r(x)$ as a valid element of the Galois field. An important property of the transmitted codeword is that it is always divisible by the generator polynomial without remainder which also applies to the individual roots of the generator polynomial $G(x)$. In the end, the transmitted code word is formed by combining $M(x)$ and $r(x)$ [3]:

$$T(x) = M(x) \times x^{n-k} + r(x) \tag{1.8}$$

which gives the following polynomial in a systematic form [3]:

$$T(x) = M_{k-1}x^{n-1} + \ldots + M_0 x^{n-k} + r_{n-k-1}x^{n-k-1} + \ldots + r_0 \tag{1.9}$$

Process of Encoding can be also perceived as conducting so called Galois field Fourier transform (further studied in [13]). It is a generalized view of discrete Fourier Transform to finite fields. In this point of view, polynomial $V(x) = V_0 + V_1 x + \ldots + V_{n-1}x^{n-1}$, where $\alpha^n = 1$, represents the spectrum of the transmitted codeword $T(x)$ over $GF(2^m)$. Polynomials $V(x)$ and $T(x)$ form a Fourier transform pair. Fourier Transform and inverse transform in Galois Field are polynomial evaluations by replacing x with α^i. From the point of view of frequency domain, process of encoding is making 2t spectral components as zero $V_j = 0$ for $j = 0, 1, \ldots, 2t-1$.

1.9 Reed-Solomon Decoder

In the RS decoder part, the transmitted polynomial $T(x)$ becomes the first part of a received polynomial $R(x)$. The last part of the received polynomial $R(x)$ of $n = k+2t$ members is the error polynomial $E(x)$. Each of the coefficients $E_{n-1} \ldots E_0$ is an m-bit error value and a valid element of $GF(2^m)$. Therefore, the $R(x)$ is [3]:

$$R(x) = T(x) + E(x) \tag{1.10}$$

where the error polynomial $E(x)$ can be written in a polynomial form as:

$$E(x) = E_{n-1}x^{n-1} + \ldots + E_1 x + E_0 \tag{1.11}$$

The positions of the errors errors in the entire code word by the degree of x for this term. If more than $t = \frac{(n-k)}{2}$ of the E values are non-zero, the correction capacity of the code is exceeded and the errors are not correctable [3].

1.9.1 The Syndromes

The first step of the decoding process is to divide the received polynomial $R(x)$ by each of the factors $(x + \alpha^i)$ (see equation 1.5) which form the generator polynomial $G(x)$ thanks to the property of the transmitted codeword which is always divisible by the generator polynomial without remainder. It applies, obviously, when the transmitted codeword $T(x)$ has been received without any of its bits in error. Remainders of these divisions are known as syndromes S_i [3]:

$$\frac{R(x)}{(x+\alpha^i)} = Q_i(x) + \frac{S_i}{(x+\alpha^i)}, \tag{1.12}$$

which applies for $b \leq i \leq b+2t-1$; where b is chosen to match the set of consecutive factors in the equation 1.5. In this work, $b = 0$ is chosen so the remainders can be written as $S_i, \ldots, S_{2t-1}$. The following rearrangement gives a single syndrome value S_i [3]:

$$S_i = Q_i(x) \times (x + \alpha^i) + R(x), \tag{1.13}$$

so that after expressing $x = \alpha^i$ the equation is reduced to [3]:

$$S_i = R(\alpha^i) \tag{1.14}$$

$$S_i = R_{n-1}(\alpha^i)^{n-1} + R_{n-2}(\alpha^i)^{n-2} + \ldots + R_1\alpha^i + R_0 \tag{1.15}$$

where $R_{n-1}, \ldots, R_0$ are the symbols of the received codeword. Therefore, a remainder S_i by the substitution $x = \alpha^i$ in the received polynomial $R(x)$ in each of the syndrome values can also be formed as an alternative to the division in the equation 1.12 [3].

If the substitution $x = \alpha^i$ is applied to the equation 1.10, it can be found that [3]:

$$R(\alpha^i) = T(\alpha^i) + E(\alpha^i) \tag{1.16}$$

and because of the fact that $T(\alpha^i)$ is a factor of $g(x)$, then $T(\alpha^i) = 0$. This gives the resulting property of the syndrome values which are not affected by the data values [3]:

$$R(\alpha^i) = E(\alpha^i) = S_i \tag{1.17}$$

These syndrome values are therefore directly dependent on the error pattern. It implies that if no errors have occurred, all syndrome values are in zero. It is important to emphasize that based on the equation 1.14 it is possible to calculate the syndrome values directly from the received codeword $R(x)$ [3].

1.9.2 The Set of Syndrome Equations

For the syndrome value determination and its location, reformulation of the error polynomial is required in a way in which only the error values are included. This can be achieved in further steps. Therefore, if assumed that v errors have occurred in a single transmission, then [3]:

$$E(x) = Y_1 x^{e_1} + Y_2 x^{e_2} + \ldots + Y_v x^{e_v} \tag{1.18}$$

while $v \leq t$; $e_1, \ldots, e_v$ are identificators of the specific error locations in a codeword which, in a form of powers of x, represent the corresponding degrees of $R(x)$ in error. And, $Y_1, \ldots, Y_v$ represent *error values* occurred in these specific error positions of the received codeword $R(x)$. It is now known that $R(\alpha^i) = E(\alpha^i)$ and if 1.18 is substituted in 1.17, then a single syndrome S_i can be written as [3]:

$$S_i = E(\alpha^i) = Y_1 \alpha^{ie_1} + Y_2 \alpha^{ie_2} + \ldots + Y_v \alpha^{ie_v} \tag{1.19}$$

where if $\alpha^{ie_1}, \ldots, \alpha^{ie_v}$ is substituted by $X_1^i, \ldots, X_v^i$ known as *error locators*, then the generic equation for $2t$ syndromes in a single transmission is [3]:

$$S_i = Y_1(X_1^i) + Y_2(X_2^i), \ldots, Y_v(X_v^i) \tag{1.20}$$

The number of syndrome equations is restricted by the correction capacity of the RS code used. They are generally denoted as $S_0, \ldots, S_{2t-1}$ in order to correspond to the roots $\alpha^0, \ldots, \alpha^{2t-1}$ of the generator polynomial $G(x)$. The powers of X_v^i depend on these roots chosen for the generator polynomial so that the powers of syndrome locators X_v^i in a single equation 1.20 are the same for the respective syndrome equations while the v index denotes the index where the error occurred in the received codeword $R(x)$ [3].

1.9.3 The Error Locator Polynomial

The following step of the RS decoding procedure is the process of error locator polynomial determination. A form of the error locator polynomial denoted as $\Lambda(x)$ has v factors constructed as $(1 + X_j x)$. It has its error locators X_j as inverses $X_1^{-1}, \ldots, X_v^{-1}$ as its roots of the v values when $j = 1$ [3]:

$$\Lambda(x) = (1 + X_1 x) + (1 + X_2 x) + \ldots + (1 + X_v x) \tag{1.21}$$

Its expanded version has degree of v, as follows [3]:

$$\Lambda(x) = 1 + \Lambda_1 x + \ldots + \Lambda_{v-1} x^{v-1} + \Lambda_v x^v \tag{1.22}$$

Now, the task is to find these coefficients of the error locator polynomial.

1.9.4 The Euclidean Algorithm

In this step, coefficients of the error location polynomial are obtained. The Euclidean algorithm is based on finding the the highest common factor of two numbers. It uses the relationship between the errors and the syndromes expressed in a form of an equation based on polynomials. It requires two new polynomials: the *syndrome polynomial* $S(x)$ and a *error magnitude polynomial* $\Omega(x)$. These two polynomials will be used in a so called *Key equation* [3]. Origin of this equation is described in detail in [28]. In this point, it is important to mention that all the requirements for each of the syndromes still apply (see equation 1.19). Now, the set of syndromes (the syndrome polynomial) $S(x)$ can be written in a polynomial form as [3]:

$$S(x) = S_{b+2t-1}x^{2t-1} + \ldots + S_{b+1}x + S_b, \tag{1.23}$$

where the coefficients are $2t$ syndrome values already calculated from the received codeword (see equation 1.15) [3]. The error magnitude polynomial is defined as [3]:

$$\Omega(x) = \Omega_{v-1}x^{v-1} + \ldots + \Omega_1 x + \Omega_0 \tag{1.24}$$

The Key equation can be written as [3]:

$$\Omega(x) = [S(x)\Lambda(x)] mod(x^{2t}) \tag{1.25}$$

where $S(x)$ is the syndrome polynomial and $\Lambda(x)$ is the error polynomial. Terms of degree x^{2t} or higher are ignored. Then, the key equation is [3]:

$$\Omega_0 = S_b \tag{1.26}$$

$$\Omega_1 = S_{b+1} + S_b\Lambda_1 \tag{1.27}$$

$$\vdots$$

$$\Omega_{v-1} = S_{b+v-1} + S_{b+v-2}\Lambda_1 + \ldots + S_b\Lambda_{v-1} \tag{1.28}$$

In general, the extended Euclidean algorithm finds the highest common factor d of two elements a and b [3]:

$$ua + vb = d \tag{1.29}$$

where u and v are coefficients produced by the algorithm. The product of $S(x)$ of degree $2t-1$ and $\Lambda(x)$ of degree v gives the resultant product of degree $2t+v-1$ (see equation 1.28) [3]:

$$S(x) \times \Lambda(x) = F(x) \times x^{2t} + \Omega(x) \tag{1.30}$$

where the single terms of $1 \times x^{2t}$ are represented by the terms of $F(x)$ and the remaining part by $\Omega(x)$. If rearranged to calculate the $\Omega(x)$, then [3]:

$$\Omega(x) = \Lambda(x) \times S(x) + F(x) \times x^{2t} \tag{1.31}$$

now, the $S(x)$ and x^{2t} correspond to the a and b terms of the equation 1.29 [3]. The Euclidean algorithm continues as follows. The purpose of the algorithm is to find $\Omega(x)$ with degree less than t (see equation 1.1). First, the algorithm consists of dividing x^{2t} by $S(x)$ and a remainder is produced. In the next step, $S(x)$ becomes the new dividend while the previous remainder is the new divisor. This process is continued until the degree of remainder (representing $\Omega(x)$) becomes less than t and the multiplying factor $\Lambda(x)$ will be also found [3].

The Euclidean algorithm can be can be also applied to polynomials. Originally, its task is to find the greatest common divisor (GCD) of two polynomials $a(x)$ and $b(x)$. Their GCD can be written as [13]:

$$u(x)a(x) + v(x)b(x) = gcd(a(x), b(x)) \tag{1.32}$$

By setting $b(x)$ as primitive polynomial $p(x)$, the $gcd(a(x), p(x))$ equals 1 since $p(x)$ is relative prime to $b(x)$. Then, if rewritten in this way, we have [13]:

$$u(x)a(x) + v(x)p(x) = gcd(a(x), p(x)) \tag{1.33}$$

$$u(x)a(x) + v(x)p(x) = 1 \tag{1.34}$$

Modulo $p(x)$ is applied at both sides, then [13]:

$$1 = a(x)u(x)mod(p(x)) \tag{1.35}$$

$$a^{-1}(x) = u(x)mod(p(x)) \tag{1.36}$$

Now, $u(x)$ mod $p(x)$ is the inverse of $a(x)$ and therefore the algorithm requires polynomial divisions [13].

1.9.5 Chien Search

Chien search serves for roots determination of the coefficient values $\Lambda_1, \ldots, \Lambda_v$ (see equation 1.22) in order to solve the error locator polynomial. If the polynomial is written in this form [3]:

$$\Lambda(x) = X_1(x + X_1^{-1})X_2(x + X_2^{-1}) \ldots \tag{1.37}$$

then, the result of the function will be zero if $x = X_1^{-1}, X_2^{-1}, \ldots$ are found, where x is [3]:

$$x = \alpha^{-e_1}, \alpha^{-e_2}, \ldots . \tag{1.38}$$

In the Chien search all possible field values of the GF roots α^i, where $0 \leq i \leq (n-1)$ are substituted into equation 1.22. The values $X_1, \ldots, X_v$ of the error locator polynomial are then found by trial and error. If the expression $\Lambda(x) = 0$, then the value x is a root of this function which has been found by the Chien search and the error position $X_1, \ldots, X_v$ has been identificated. The search begins with $\alpha^{-(n-1)}$ $(= \alpha^1)$, then $\alpha^{-(n-2)}$ $(= \alpha^2)$ and continues to α^0 corresponding to the last symbol of the word while the first symbol of the codeword corresponds to the x^{n-1} term [3].

1.9.6 Forney's Equation

Forney's equation allows to calculate error values $Y_1, \ldots, Y_v$ (see equation 1.18) and therefore the error polynomial $E(x)$ formation. An error value is calculated accordingly [3]:

$$Y_j = X_j^{1-b} \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})} \tag{1.39}$$

where $\Lambda'(X_j^{-1})$ is the derivative of $\Lambda(X)$ for $x = X_j^{-1}$. If $b = 1$, the X_j^{1-b} term disappears. Therefore, the formula is often quoted in the literature as Ω/Λ', which gives wrong results for $b = 0$ and other values defined in equation 1.5 where the code generator polynomial has been determined. The equation 1.39 gives valid results only for symbol positions in error. If the calculation is made at other positions, the result is generally non-zero and invalid. Therefore, the Chien search is needed in order to identify these error positions [3].

1.9.7 RS-FEC Error Correction Capability

Beside the already introduced notation $RS(n, k)$ it essentially defines a vector space of k dimensions and the every non-zero codeword differs at least $2t + 1$ coordinates. The received codeword $R(x)$ is also called RS frame.

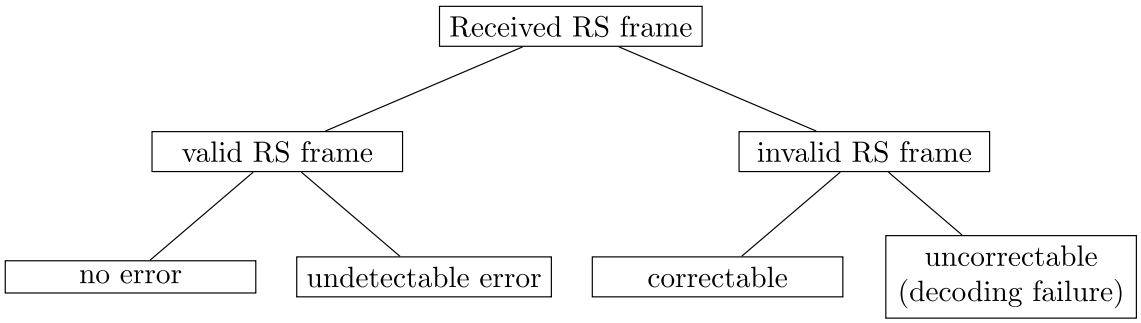

Fig. 1.12: RS-FEC error correction capability breakdown (taken from [13])

While receiving an invalid codeword, the decoder will map the codeword to the closest one in the vector space defined by (n, k) RS code. In case of more than t

errors occurred during transmission, the received codeword may be closer to another valid codeword and therefore the decoder will map it to this valid codeword. In this situation, the error is undetectable because the Euclidean processor has found solution for a valid codeword reconstruction. If an erroneous codeword differs from all the codewords in $t + 1$ or more coordinates, then the codeword after decoding is still invalid. In this case, the error is detectable and the decoder can report the error [13]. A breakdown of all possible types of errors which may occur during transmission are shown in figure 1.12.

2 Practical Part

Practical part of the work deals with hardware aspect of Reed-Solomon error correction algorithm for 400 Gigabit Ethernet and its implementation in Intel® Stratix® 10 DX FPGA chip.

Brief digital circuit block diagrams for both encoder and decoder components are shown and explained in following chapters. It is also important to highlight that since the beginning of this work there has been a strong focus on parametrizability of the system, required data throughput and the overall functionality covering the main points of the scope of this project.

2.1 Motivation

During writing this thesis, the Physical Coding Sublayer of Ethernet PHY has been realized and implemented. Hence, the further step was to develop an IEEE Std. 802.3$bs^{\mathrm{TM}}-2017$ compliant RS-FEC layer for 400 Gigabit Ethernet for project Netcope Development Kit within the academical organisation Cesnet z. s. p. o.

Because of the fact that many modern technologies, such as constantly emerging cloud services providers, financial network organizations, large scale enterprise data centers are dependent on digital data, RS error correction system is a sufficient solution for ensuring reliability of current data rates in the PHY layer [14].

In terms of future Ethernet speeds, there are prototypes nowadays reaching 800 Gbps speeds employing 2x RS-FEC based on Clause 119 of IEEE Std. 802.3 [29]. Hence, it is therefore highly likely that RS-FEC will be present in future Ethernet IEEE standards and it is therefore worth its parametrizable VLSI (Very-Large-Scale Integration) description.

2.2 RS-FEC Layer Concept and Galois Field Construction

General concept of RS-FEC system follows procedure shown in figure 2.1. Ensuring reliability of digital data transmission begins in the part of transmitter. Incoming message $M(x)$ to be corrected by the decoder part of the RS-FEC algorithm has to be encrypted first to form a parity polynomial which is adjusted to the message itself and then sent together in a form of a codeword $T(x)$ to the receiver (decoder).

On the way between data transmitter and receiver, noise of the environment causes a quantifiable error $E(x)$ which can be expressed in a form of a polynomial of the same degree as the transmitted codeword $T(x)$. The idea is that the Error

polynomial $E(x)$ which has been added to the transmitted codeword $T(x)$ during transmission.

The main task of the decoder is to find the error pattern $E(x)$ so that the pattern can be subtracted/added to the received message $R(x)$ at the side of the RS decoder.

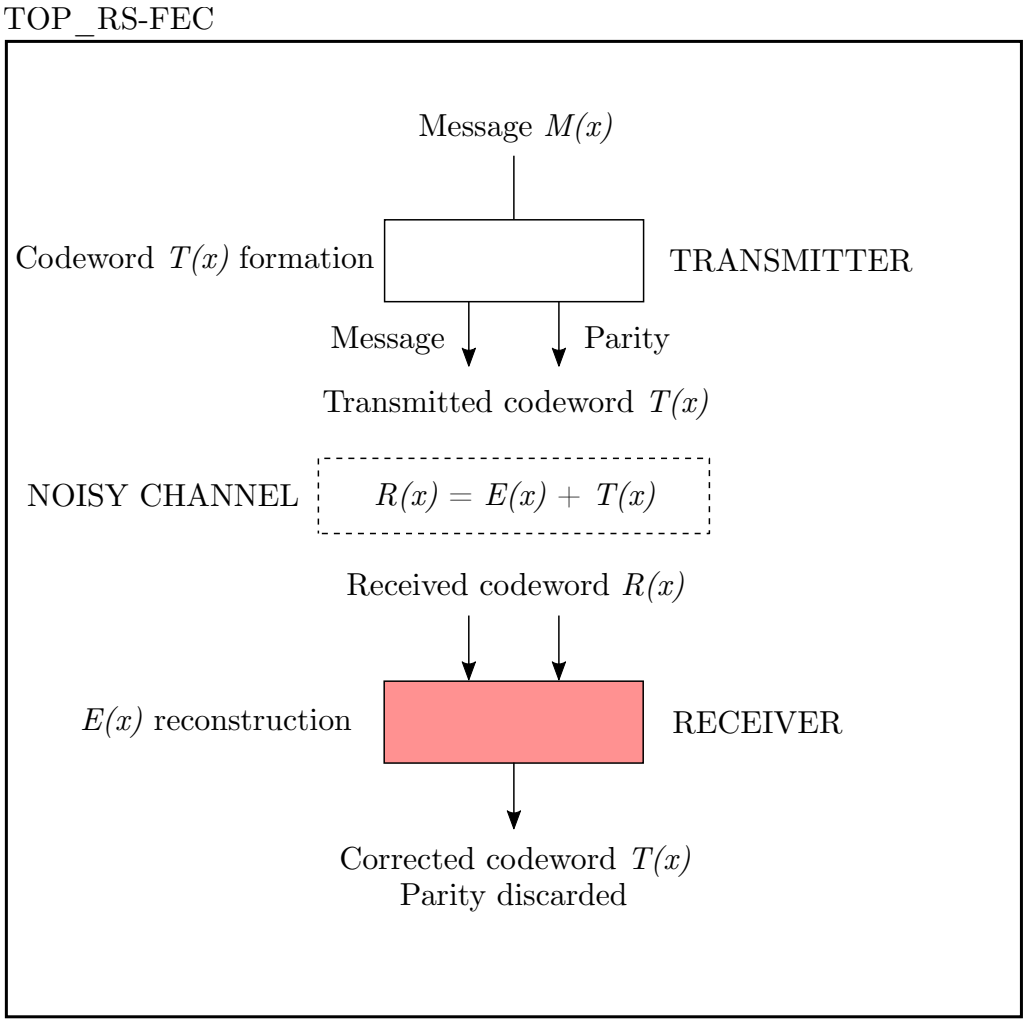

Fig. 2.1: Diagram of RS-FEC concept

First of all, since the entire system operates over GF(2^m) where m=10, therefore with 10-bit symbols, it is needed the Galois field to be formed $GF(2^{10})$. It is a repeating sequence of 2^{10} 10-bit primitive elements α where each of them has its own position label α^i starting with α^0=1 and ending with $\alpha^{2^{10}-2}$ while following elements, such as $\alpha^{2^{10}-1}$ represent again α^0 and so on.

Based on the chapter 1.7.2 a list of primitive GF elements α^i where i=0, 1, 2, ... $(2^{10}-2)$ can be constructed in a way shown in table 2.1.

With reference to the chapter 1.7.2 an inverse GF element is a valid element of the GF to be multiplied with another one while 1 (α^0) is produced. Inverse elements of Galois field can be found using logarithmic method, however, excluding

Tab. 2.1: Example of primitive elements and inverses of $GF(2^m)$ where m=4 (taken from [3])

Index form	Polynomial form	Binary form	Decimal form	Inverses (decimal)
0	0	0000	0	0
α^0	1	0001	1	1
α^1	α	0010	2	9
α^2	α^2	0100	4	13
α^3	α^3	1000	8	15
α^4	α+1	0011	3	14
α^5	α^2+α	0110	6	7
α^6	α^3+α^2	1100	12	10
α^7	α^3+α+1	1011	11	5
α^8	α^2+1	0101	5	11
α^9	α^3+α	1010	10	12
α^{10}	α^2+α+1	0111	7	6
α^{11}	α^3+α^2+α	1110	14	3
α^{12}	α^3+α^2+α+1	1111	15	8
α^{13}	α^3+α^2+1	1101	13	4
α^{14}	α^3+1	1001	9	2

field element 0 which does not have multiplicative inverse. Following example shows calculation of inverse of ($\alpha^{14} = 9$) in GF where $m = 4$: [3]

$$\alpha^{-imod(2^m-1)} = \alpha^{-14mod15} = \alpha^1 = 2 \tag{2.1}$$

For inverse of $\alpha^{-13} = \alpha^2 = 4$, $\alpha^{-12} = \alpha^3 = 8$ etc. Inverse Galois field is therefore a "mirrored" form of Galois field which applies for elements α^1, α^1, ..., α^{2^m-2}.

Each component of the RS-FEC system requires two main functions for its operation. First, addition/subtraction in Galois field which is realized by XOR function. Second, multiplication in GF which is realized by two main components, as shown in 2.2, multiplier and divider. Result of the multiplication operation over GF(2^m) produces another valid element of this field which is ensured by the modulo operation.

If we want to multiply two binary polynomials, we will get a product of degree $2m - 1$ which is not an element of GF(2^m). Then the product becomes a dividend in the following division operation reducing the degree to a valid element of GF(2^m) while divisor is primitive polynomial $p(x)$ and has degree m. As a result, remainder after the division is a valid element of GF(2^m) and quotient is discarded.[3]

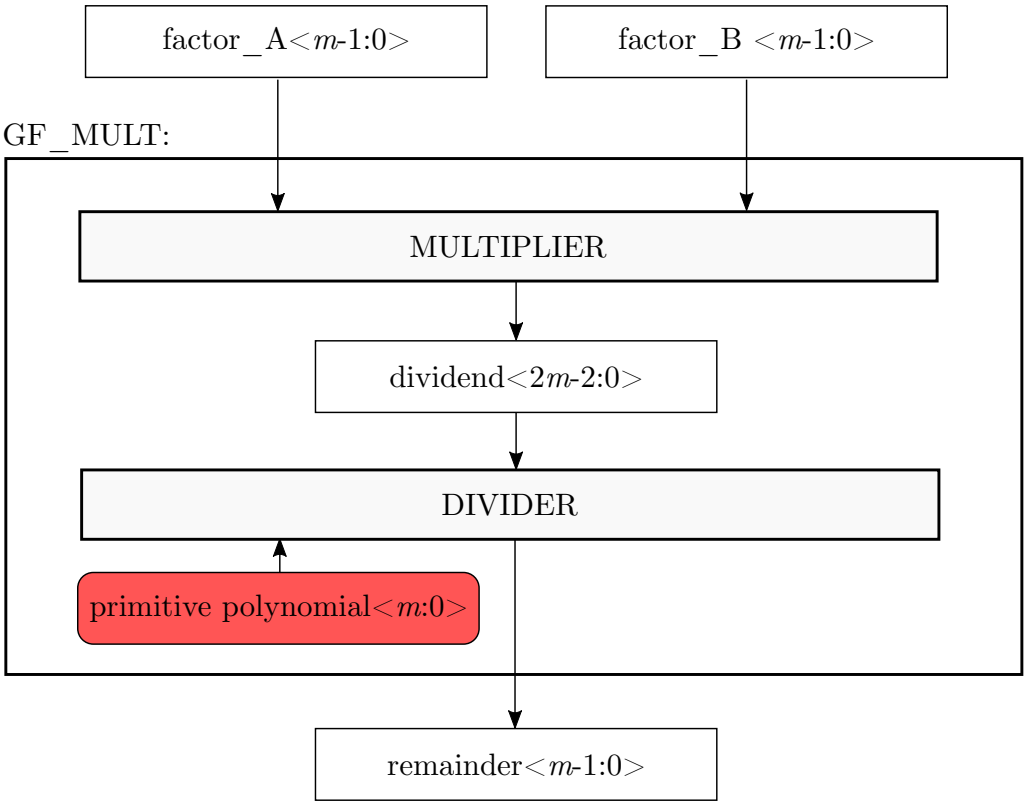

Fig. 2.2: Galois field multiplier

GF multipliers are critical component for resultant hardware requirements because they are used in every component in a large extent. In this work, two types of parametrizable GF multipliers were synthetized and implemented. It turned out that RS-FEC Encoder with behavioral VLSI description of shift-and-add full GF multipliers (based on [3], page 28) requires less logic than its RTL-based description. But when implemented within a more complex component, the resultant difference in hardware utilization was not significant. Therefore, future optimizations should focus on optimal operation with 10-bit symbols rather than its full parametrizability.

IEEE Std 802.3$bs^{\text{TM}}-2017$ also defines Primitive polynomial $p(x)$ in subchapter 119.2.4.6 for $GF(2^{10})$: [17]

$$\alpha = x^{10} + x^3 + 1 \tag{2.2}$$

and Generator polynomial for parity polynomial calculation $G(x)$, shown in table 2.2 [17].

The circuit needed for parity calculation conducts pipelined form of long polynomial division where the dividend is incoming message $M(x)$ multiplied by x^{2t} which serves as a room for parity polynomial $r(x)$ and the divisor is constant generator polynomial $G(x)$. The divider is based on adding already multiplied generator polynomial by a specific GF element with already pre-registered values of this op-

eration. Initially, this number can be found when the most significant degree of the $M(x)\times x^{2t}$ is added to the most significant degree of initialized register in 0. Then, the generator polynomial $G(x)$ is multiplied with this number and is added to remaining degrees of the register. In the next cycle, registered values replace the previous ones and the operation continues in the same way.

Tab. 2.2: Coefficients of the generator polynomial G_i (taken from [17])

i	G_i	i	G_i	i	G_i
0	523	11	883	22	565
1	834	12	503	23	108
2	128	13	942	24	1
3	158	14	385	25	552
4	185	15	495	26	230
5	127	16	720	27	187
6	392	17	94	28	552
7	193	18	132	29	575
8	610	19	593	30	1
9	788	20	249		
10	361	21	282		

It requires k cycles to calculate the parity polynomial $r(x)$ [3]. Such operation has been implemented in a similar way which is shown in figure 2.5 as a hardware representation of this recursive function.

2.2.1 Reed-Solomon Encoder

The process of remainder polynomial $r(x)$ calculation [3] follows IEEE Std. 802.3bs$^{\text{TM}}$-2017 for 200GE and 400GE. [17] The RS-FEC Encoder follows procedure shown in figure 2.3.

RS-FEC Encoder has been successfully verified using ModelSim and compared with Annex 119A IEEE Std. 802.3$bs^{\text{TM}}-2017$. The encoder part has been created first for $m = 4$. Results of this simulation and intermediate calculations correspond to the values available in [3] on page 11. Full parametrizability of this component has been therefore confirmed. An advantage of the code is the possibility of larger or smaller variants creation for various data lengths (shortened codes creation) simply by changing generic variables in the code. However, digital circuits introduced for RS-FEC Encoder and Descrambler provide general functional models, for resultant implementation, larger parallel combinatorial functions with registers were to be taken in account for the desired sequential RS-FEC system within the PCS layer.

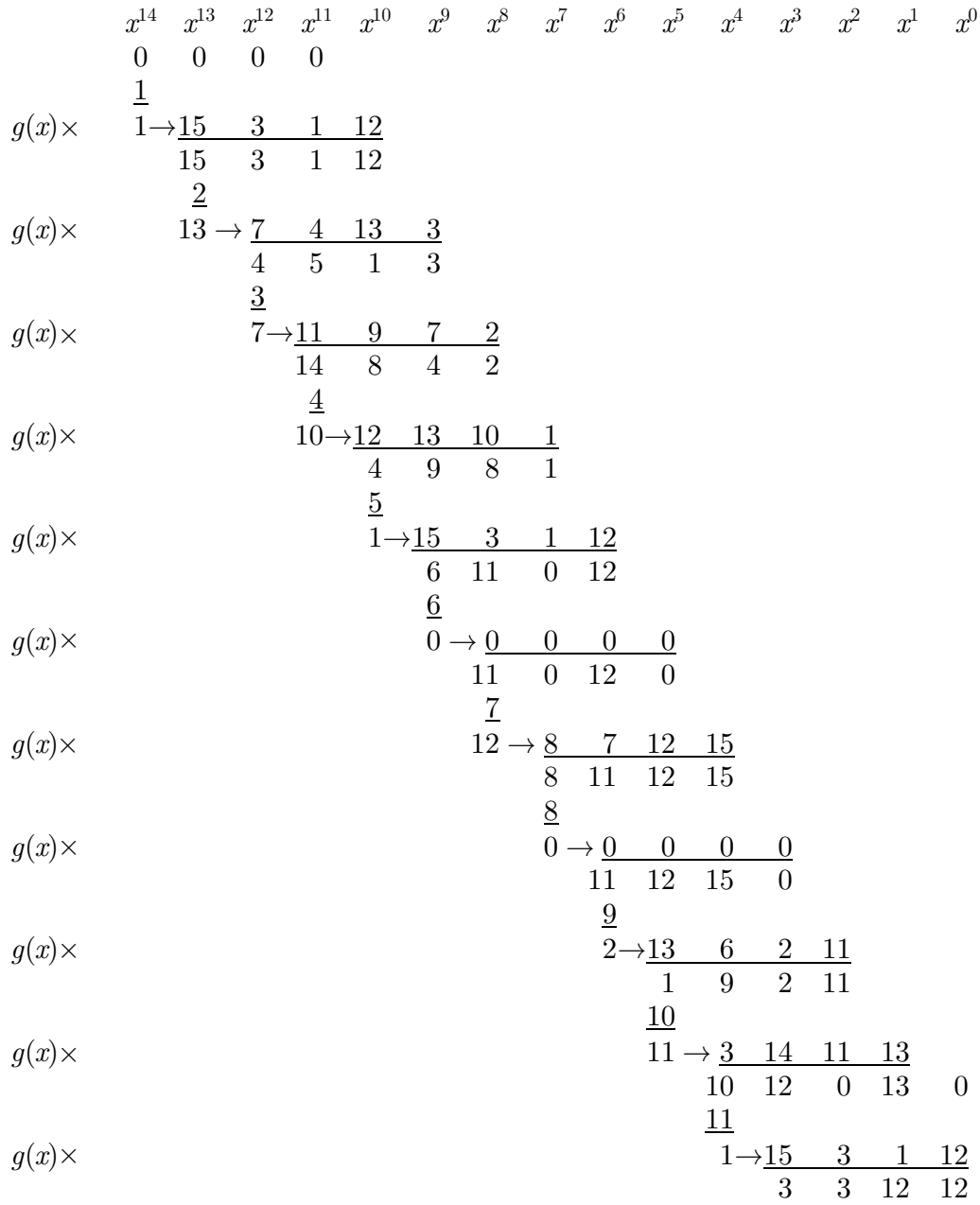

Fig. 2.3: Numerical representation of RS-FEC Encoder operation (taken from [3])

General diagram of RS-FEC Encoder operation for 400GE is shown in figure 2.5. It consists of two main components: `RS_400_FEC_ENC` consisting of two RS-FEC Encoders for `codeword_A` and `codeword_B` generation and MUX for symbols distribution.

2.2.2 Symbols Distribution

According to the IEEE Std. 802.3*bs* ™ −2017 subchapter 119.2.4.7 for 400GE the two codewords `codeword_A` and `codeword_B` of 544 symbols are further interleaved

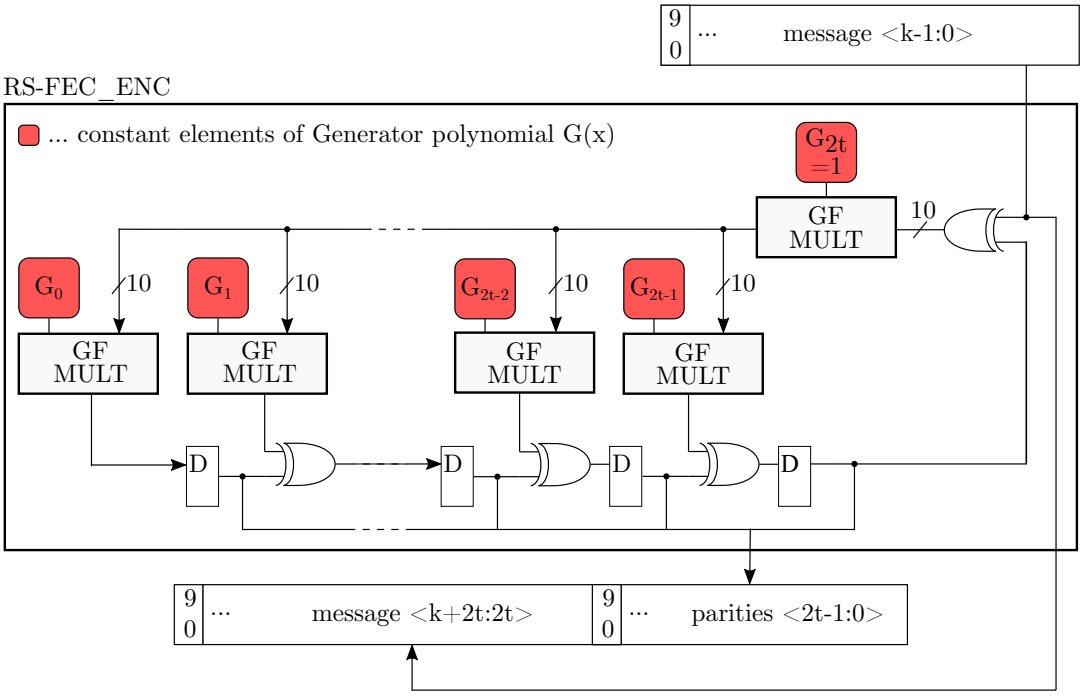

Fig. 2.4: RS-FEC Encoder hardware diagram for polynomial division

in the component `RS_ENC_INT` and distributed among 16 PCS lanes. This component has also been designed generically. Therefore, by changing number of generic parameters it is possible to switch between 200 Gbps or 400 Gbps platforms or even IEEE Standards for future Ethernet speeds, if needed.

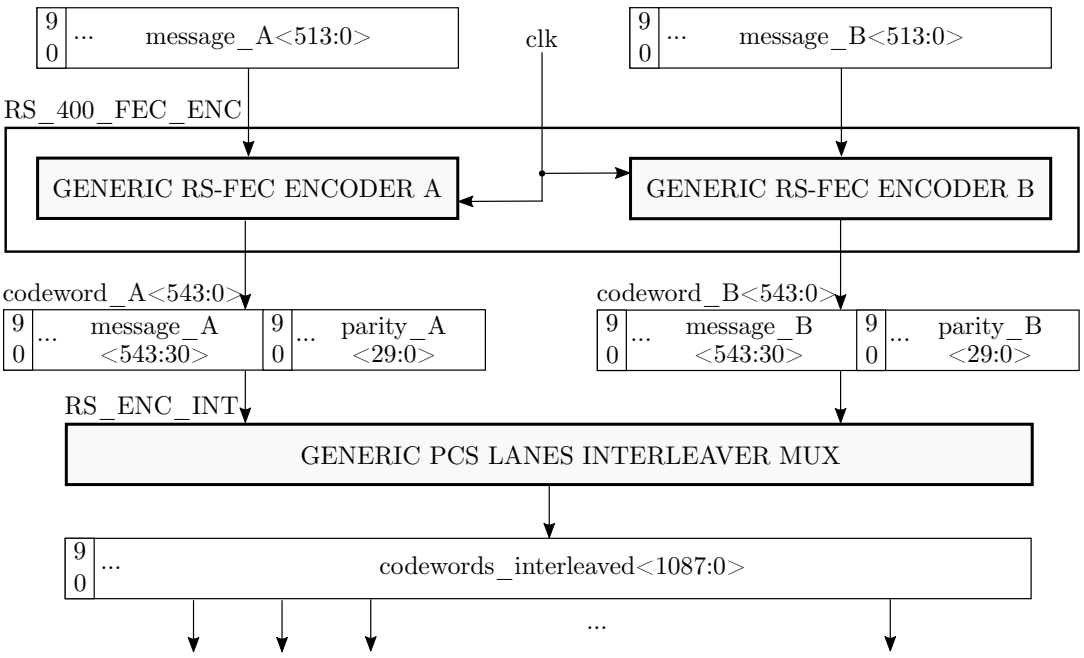

Fig. 2.5: RS-FEC Transmitter diagram for 400GBASE

The interleaving and deinterleaving of two codewords has been realized at symbol-level and follows the procedure shown in the pseudocode below [17]:

```
const int MSGS = 2          //Messages to interleave
const int PCSL = 16         //PCS lanes
const int SYM = 514-1       //Symbols of a message to be encoded
const int GENPOL = 31-1     //Symbols of the Generator polynomial G(x)
```

```
if (PCSL/MSGS) mod(MSGS) = 0
then
A = PCSL/MSGS

if ((SYM+1 + GENPOL)/J) mod(J) = 0
then
B = (SYM+1 + GENPOL)/J

const int CWSYM = SYM + GENPOL
```

```
for all K=0 to (A-1)
for all J=0 to (B-1)

if even(K)

tx_out<PCSL×K+MSGS ×J> = codeword_A<CWSYM-A×K-J>
tx_out<PCSL×K+2J+1> = codeword_B<CWSYM-A×K-J>

else

tx_out<16K+2J> = codeword_B<CWSYM-A×K-J>
tx_out<16K+2J+1> = codeword_A<CWSYM-A×K-J>
```

2.2.3 Reed-Solomon Decoder

Reed-Solomon decoder constitutes the most complicated part of the Reed-Solomon error correction system. Its block diagram is shown in figure . Block diagrams of each component or RS-FEC Decoder are shown and described in further subchapters.

Before the input vectors `err_codeword_A` and `err_codeword_B` enter top-level RS-FEC Decoder entity `RS_FEC_DEC`, input interleaved signal is deinterleaved back to separate codewords A and B. This is because two RS-FEC Decoders reconstruct error polynomial `err_pol_A` and `err_pol_B` from these two input codewords separately while each one operates with respective RS frame. Deinterleaving has been also designed generically and is therefore possible to switch among various platforms based on number of PCS (Physical Coding Sublayer) lanes of the platform as determined in IEEE Std. $802.3bs^{\mathrm{TM}}-2017$. In terms of 400GE operation 16 PCS lanes.

Since the input signal propagates through RS-FEC Decoder, first, the system checks if input codewords are divisible by consequent roots of Generator polynomial $G(x)$ without remainder. In this work, $b = 0$ and therefore the division starts with α^0 for respective syndrome S_1 and so on. This operation exploits the fact that Generator polynomial $G(x)$ has been constructed from such GF elements beginning with α^b, as determined in equation 1.5), and therefore made the incoming codewords divisible by these elements without remainder one after another forming a set of syndromes. An entity conducting this operation is `FEC_DESC` and has been realized generically. Remainders of subsequent divisions by Galois field primitives form factors of respective degrees of Syndrome polynomial $S(x)$ (see 1.19) `syndrome_A` and `syndrome_B`. Also, desired number of Galois field elements were calculated (see Tab. 2.1 using a function generating a constant vector implemented in ROM-based Look-up tables. This function can generate any $GF(2^m)$, in this case $GF(2^4)$ for prototype and $GF(2^{10})$ for resultant data widths.

The set of syndromes is further decoded by two Euclidean processors for each codeword separately used for mapping the received codeword based on its syndromes to the closest valid codeword by further components. There are two outputs from each Euclidean processor. First, magnitude polynomial $\gamma\Omega(x)$ and second error locator polynomial $\gamma\Lambda(x)$. These polynomials are used to give information about locations and magnitudes of errors in received codewords based on their syndromes. Since the Key equation (see equation 1.26) has been introduced in [28] on page 2 and Euclid's method applied to the Key equation, equation 1.31 can be found. The Euclidean processor consists of t layers in total which determines the maximal degree of output vectors lambda and omega.

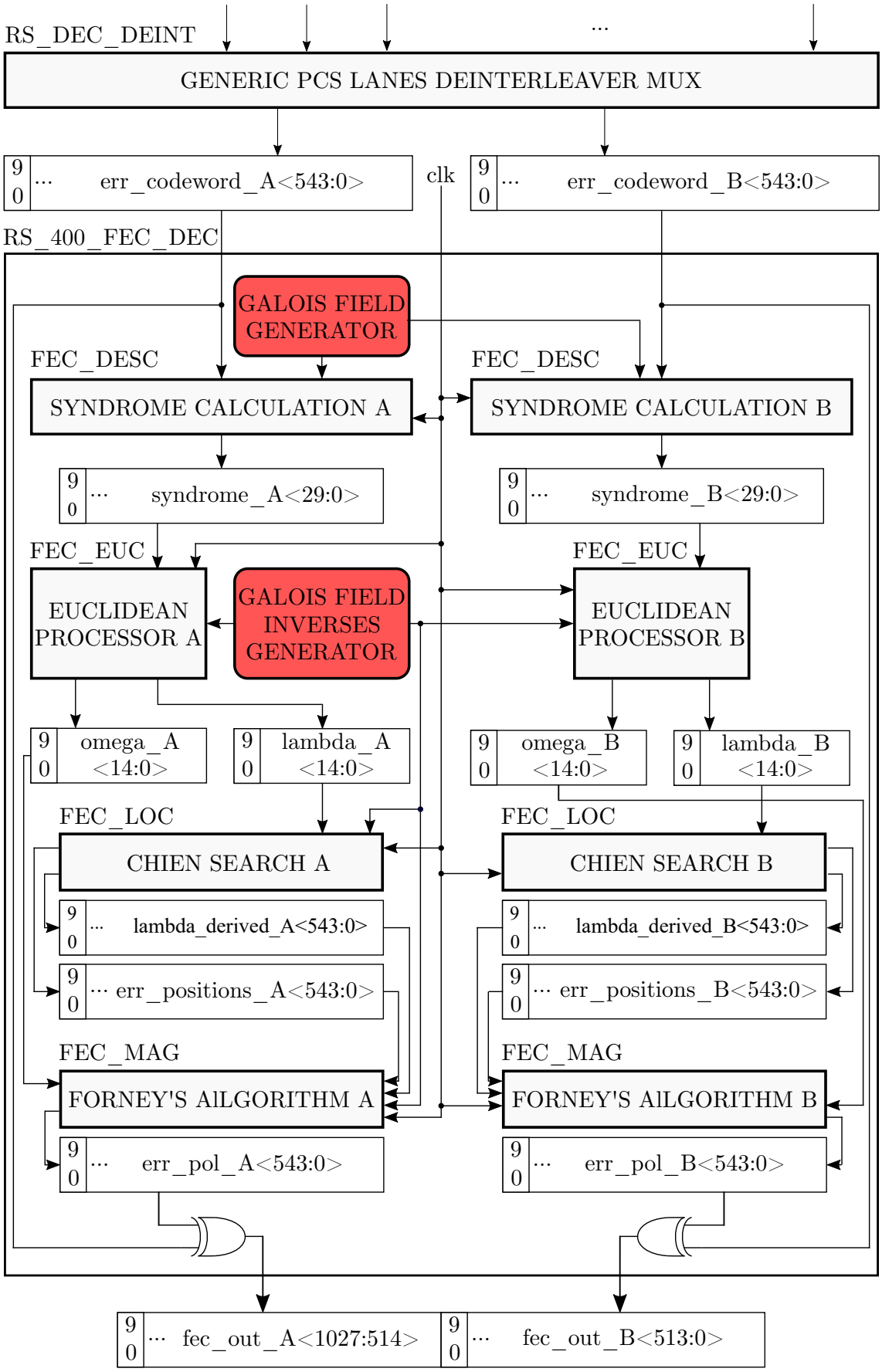

Fig. 2.6: Diagram of the RS-FEC Decoder

An interesting property of this algorithm is that if $\gamma\Omega(x)$ has been successfully found, its degree determines in which layer the calculation has finished. In other words, if the received codeword contains single error, the result of $\gamma\Omega(x)$ computation is in the first layer. If the received codeword contains two errors, the result of computation $\gamma\Omega(x)$ lies in the second layer etc. Respectively, $\gamma\Lambda(x)$ has been also found in these layers, as well. In addition to the polynomial division part, GF inverses have to be used in this process which were found using logarithms as shown in equation 2.1. This process and its implementation is described in more detail in subchapter 2.2.3.

Last two entities for solving the Error polynomial $E(x)$ `fec_out_A` and `fec_out_B` are Chien search and Forney's algorithm. For these components, multiple degrees of inverses were found using a function which generates inverses as a constant stored in ROM-based LUTs. It has also been designed generically and can be therefore used for any GF(2^m). Chien search solves equation 1.37 and derivation described in 1.9.6. Forney's algorithm conducts more straightforward operation, and this is the fraction in the equation 1.39.

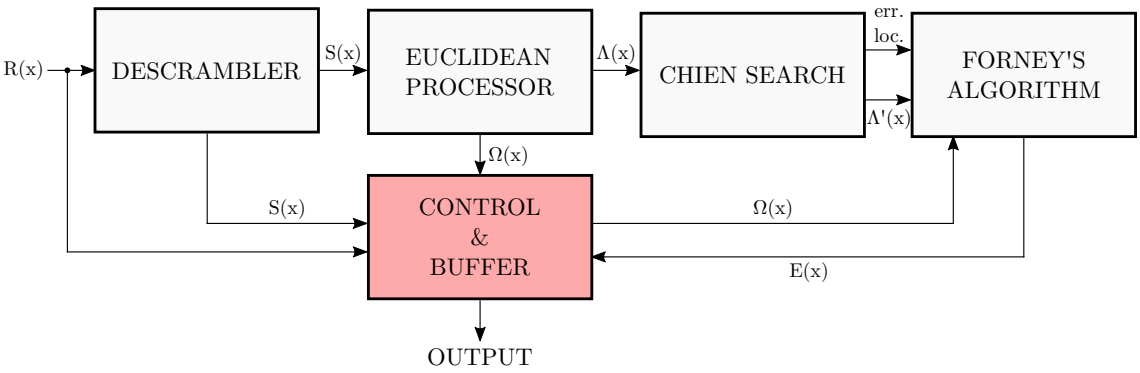

Fig. 2.7: Sequential solution for a single RS-FEC decoder

The resultant RS-FEC is a sequential component which further requires control unit which also serves for error correction status reporting to the PCS layer. Its operation is summarized in 1.9.7. In addition, buffers are needed to synchronize signals with the sequential error correction flow. This is shown in figure 2.7.

Syndromes computation

Hardware for syndrome calculation follows procedure shown in figure 2.9. The most significant degree of the input received message $R(x)$ is added with registered value initialized in 0 and then multiplied with a constant primitive element α^b. In the next cycle, XOR gate reads one degree lower factor of the input message $R(x)$ and

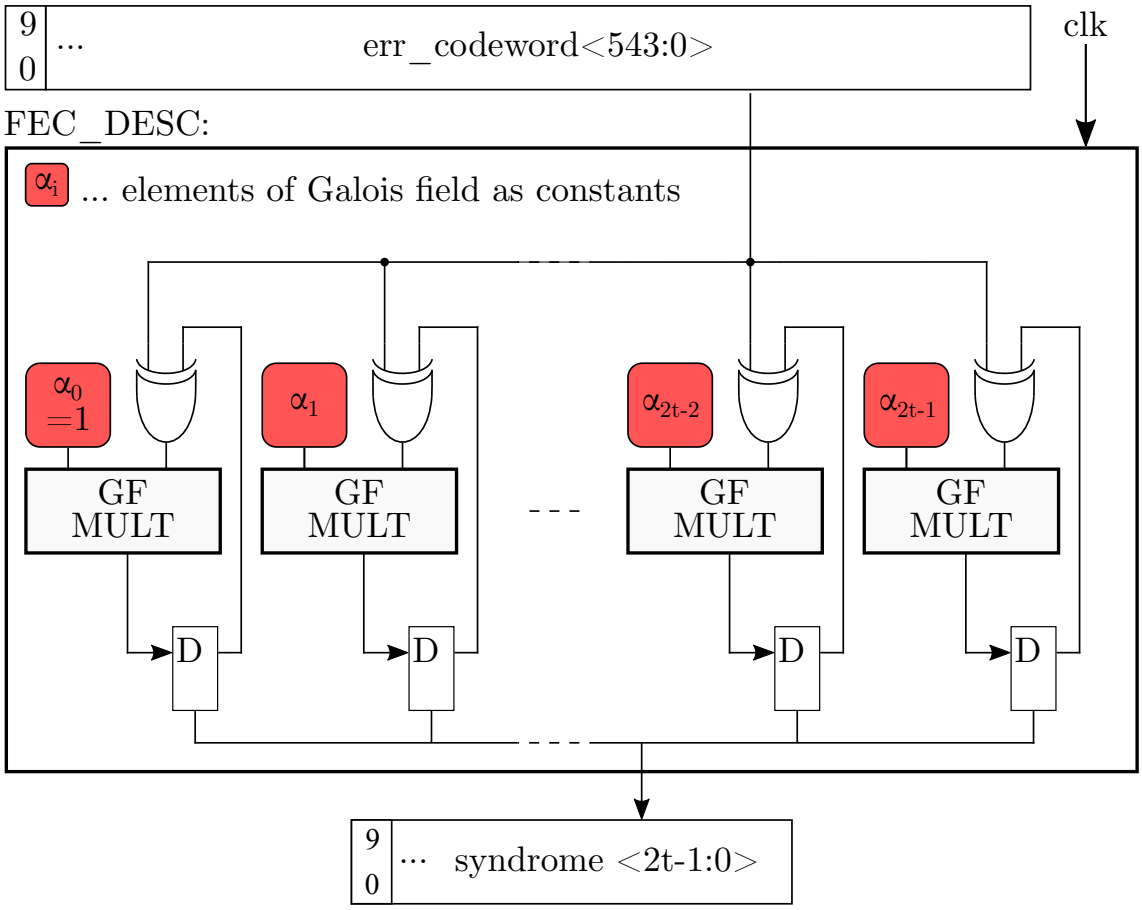

Fig. 2.8: Hardware of descrambler for syndromes calculation

the remaining process remains the same. Here, $b = 0$. Then, this value rewrites the previous initial value in the register etc.

In parallel, this operation shown in figure 2.9 runs with other primitive elements of GF α^{b+1}, ... $\alpha^{b+k+2t-1}$. Process with multiplying by α^0 forms S_0, α^1 forms S_0 etc. This general operation is represented in a diagram shown in 2.8.

Euclidean processor design

The main process present in a single layer of the Euclidean processor has been divided into two parts: left-hand process for polynomial division and right-hand process conducting polynomial multiplication and addition, as described in [3]. Figure 2.10 shows calculation of the first layer of the Euclidean processor for $\gamma\Omega(x)$ and $\gamma\Lambda(x)$ calculation and figure 2.11 the second layer.

Suppose that input to the FEC_EUC from RS_DESC is this polynomial $S(x)$:

$$S(x) = 12x^3 + 4x^2 + 3x + 15 \tag{2.3}$$

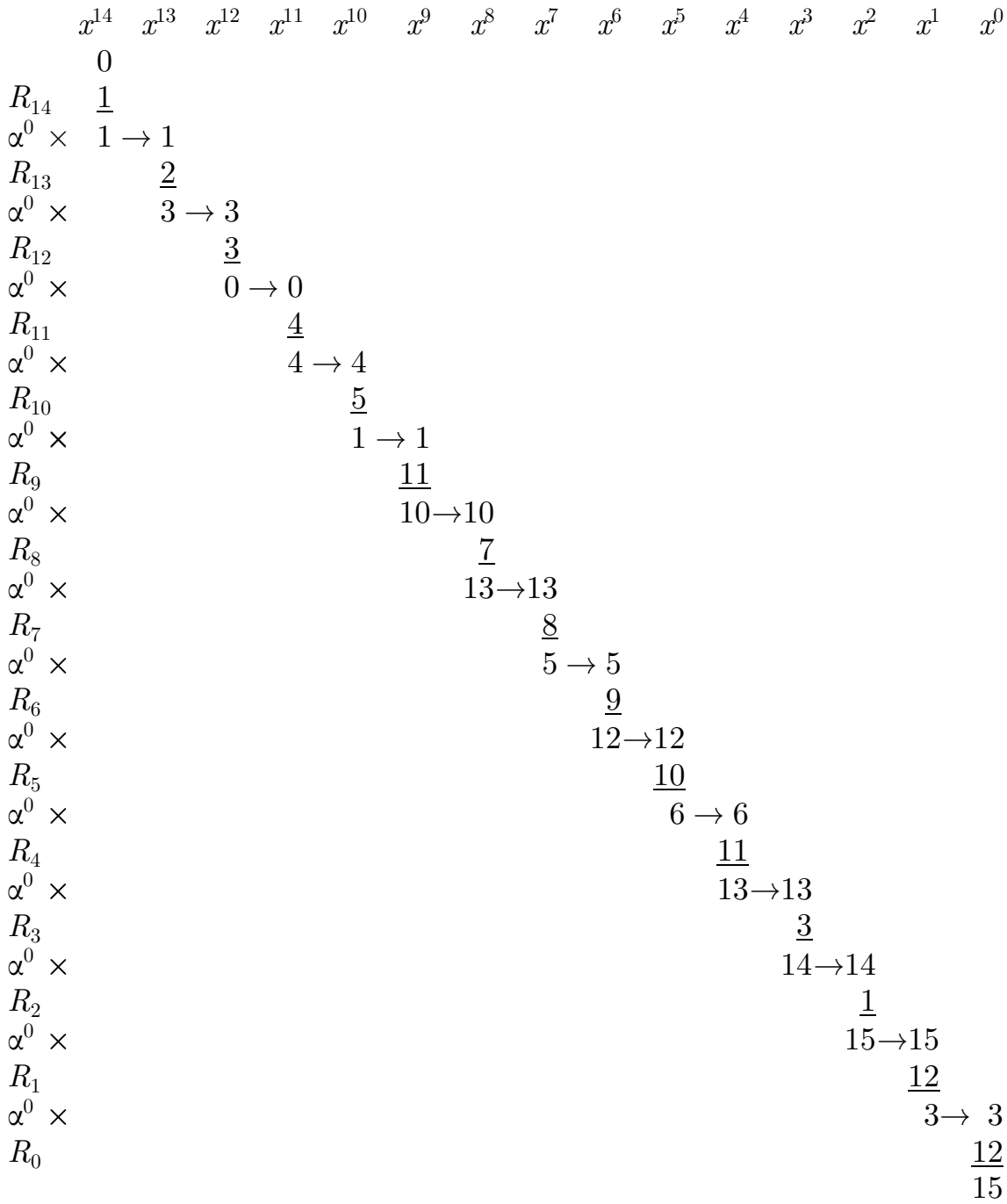

Fig. 2.9: Numerical representation of syndrome calculation operation

	x^4	x^3	x^2	x^1	x^0		x^2	x^1	x^0
dividend:	1	0	0	0	0			0	0
divisor$\times 10x$*:*	1	14	13	12				10	0
		14	13	12	0			10	0
divisor$\times 6$*:*		14	11	10	4			0	6
remainder:			6	6	4			10	6

Fig. 2.10: Eample of first layer of Euclidean processor operation [3]

In the first layer, register with dividend polynomial initialized in x^{2t} (here, x^4) will be divided by polynomial $S(x)$ as dividend, however, multiplied by certain value causing the most significant degree of the dividend (x^4) to be equal to the most significant degree of the divisor which results in cancelling the most significant degrees after addition of contents in these two registers. This number can be found by dividing the most significant degree of the dividend by divisor's most significant factor. Divisor polynomial is constant for the entire layer based on its degree, number of division steps differs. Then, resulting quotient of the division is:

$$\frac{1x^4}{12x^3} = \alpha^0 x^4 \times \alpha^{-6} x^{-3} = \alpha^0 x^4 \times \alpha^9 x^{-3} = 1x^4 \times 10x^{-3} = 10x \tag{2.4}$$

This number (10x) is the quotient which multiplies both the divisor $S(x)$ and also register with polynomial $F(x)$ in the right-hand side of the process. On the left, multiplied divisor factors are added to respective factors of the dividend x^4 while the most significant degree is cancelled. Then, another factor with degree lower by one is adjusted to the dividend in the next cycle and register of the dividend is rewritten. Multiplication on the right-hand side with polynomial $F(x)$ initialized in 1 is trivial (it is important to mention that multiplication is performed over $GF(2^4)$:

$$1 \times (10x) = 10x \tag{2.5}$$

However, there is no previous layer and therefore all remaining registers are set to 0. At the same time as left-hand preforms addition, on the right-hand side is performed as well and then rewrites current register for addition:

$$0 + 10x = 10x \tag{2.6}$$

This process continues once again since maximal number of errors t occured. If the number of errors is lower, based on the divisor's degree the number of cycles can be determined.

$$cycles = 1 + (degree\_dividend - degree\_divisor) \tag{2.7}$$

If the degree of dividend in this example is 4 and degree of divisor is 3, then 2 cycles of division have to be performed to obtain remainder. On this basis, first implementation of Euclidean processor is based to secure the Euclidean processor operation.

The second and following layers operate on similar basis as the first one, however, inputs to the layer change. First, on the left-hand side, divisor from the previous layer becomes current layer dividend and remainder from the previous layer is divisor of the current layer. This applies also for next layers. On the right-hand side, two registers read data from the previous layer. First, current register $F(x)$ is loaded with data from previous result $\gamma\Lambda(x)$ and initial sum to the process (first row) reads data of $F(x)$ from previous cycle.

	x^4	x^3	x^2	x^1	x^0		x^2	x^1	x^0
dividend:		12	4	3	15			0	1
divisor$\times 2x$*:*		12	12	8			7	12	0
			8	11	15		7	12	1
divisor$\times 13$*:*			8	8	1			11	8
remainder:				3	14		7	7	9

Fig. 2.11: Example of remaining layers of Euclidean processor operation [3]

Now it is important to select the right output. If the degree of remainder on the left-hand side of the euclidean algorithm is lower than t (in this example 2) the $\gamma\Lambda(x)$ and $\gamma\Omega(x)$ has been found. In the realization of euclidean processor, output logic selects correct output based on registered degrees of dividends and divisors of each layer of the euclidean. There is also room for improvement because the degree can be estimated only in the first layer and based on this, other layers can be synchronized accordingly. This has to be tested first.

The diagram of the euclidean processor is shown in figure 2.12. Similarly as the algorithm shown above, the main operation is divided into three main parts. Based on incoming syndrome polynomial degree every layer of the euclidean processor is configured. For the division part, inverses of GF are loaded in LUTs. Intermediate remainders of each layer are used by control logic to set up next layer and output logic selects the right output. In addition, intermediate product from the right-hand process is also sent to control logic in order to shift products after multiplication accordingly.

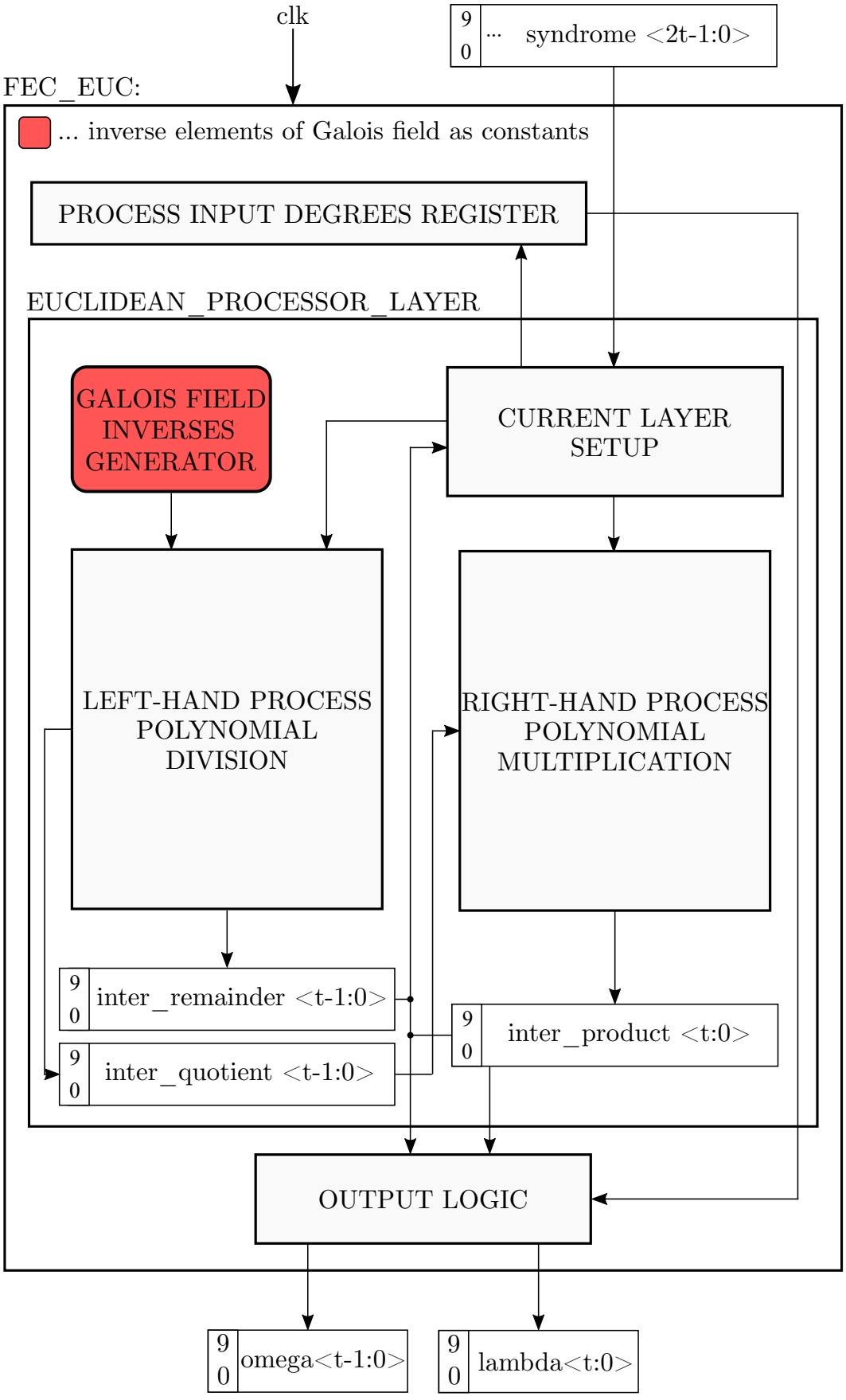

Fig. 2.12: Diagram of Euclidean processor unit for RS-FEC

Hardware for error locations

Chien search component performs two main operations: solves Error locator polynomial $\gamma\Lambda(x)$ (see equation 1.37) and derivation of polynomial $\gamma\Lambda'(x)$ for each of α^b, $\alpha^{b+1}...\alpha^{b+k+2t-1}$ described in chapter1.9.5. Based on these two outputs, Error polynomial can be solved using Forney's equation. There are therefore $k + 2t - 1$ solutions for each of α^b, α^{b+1} ... $\alpha^{b+k+2t-1}$ elements for both output vectors which correspond to the width of Error polynomial $E(x)$. With reference to equation 1.37 the Error locator polynomial can be solved by following procedure. The result of the previous example in the right-hand side process of the Euclidean processor is [3]:

$$\gamma\Lambda(x) = \gamma\Lambda_2 + \gamma\Lambda_1 + \gamma = 7x^2 + 7x + 9 \tag{2.8}$$

Divided by γ the Error locator polynomial equals to:

$$\frac{\gamma\Lambda(x)}{\gamma} = \Lambda(x) = 14x^2 + 14x + 1 \tag{2.9}$$

The Chien search algorithm can be performed at positions $x = \alpha^b$, α^{b+1} ... $\alpha^{b+k+2t-1}$, in this example for $e_{14} = \alpha^{14}$, as follows:

$$\Lambda(x) = 14x^2 + 14x + 1 = 14(\alpha^{-e_{14}})^2 + 14\alpha^{-e_{14}} + 1 = 14(\alpha^{-14})^2 + 14\alpha^{-14} + 1 \tag{2.10}$$

In this point it is possible to identify respective inverse elements stored in ROM-based Look-up table shown in 2.1 realized by a constant vector generated by a generic VHDL function:

$$\Lambda(x) = 14(\alpha^1)^2 + 14\alpha^1 + 1 \tag{2.11}$$

Implemented hardware of Chien Search then follows this calculation of this line:

$$\Lambda(\alpha^{-14}) = \alpha^{11}(\alpha^1)^2 + \alpha^{11}\alpha^1 + \alpha^0 \tag{2.12}$$

$$\Lambda(\alpha^{-14}) = \alpha^{11}\alpha^2 + \alpha^{11}\alpha^1 + \alpha^0 = \alpha^{13} + \alpha^2 + \alpha^0 = 13 + 15 + 1 = 3 \tag{2.13}$$

The result of $\Lambda(\alpha^{-14})$ is 3 which signifies that on position e_{14} an error did not occur. This computation runs for all error positions e_b ... $e_{b+k+2t-1}$ in parallel and therefore the benefit of parallelism in an FPGA has been exploited.

Since the first derivative of $\Lambda(x)$ has been found, as shown in the following example where even degrees of x are set to zero [3]:

$$\Lambda(X_j)' = \Lambda_1 + \Lambda_3 \times X_j^{-2} + \Lambda_5 \times X_j^{-4} + \dots \tag{2.14}$$

$$\Lambda(X_j)' = \frac{14X_j}{X_j} = 14 \tag{2.15}$$

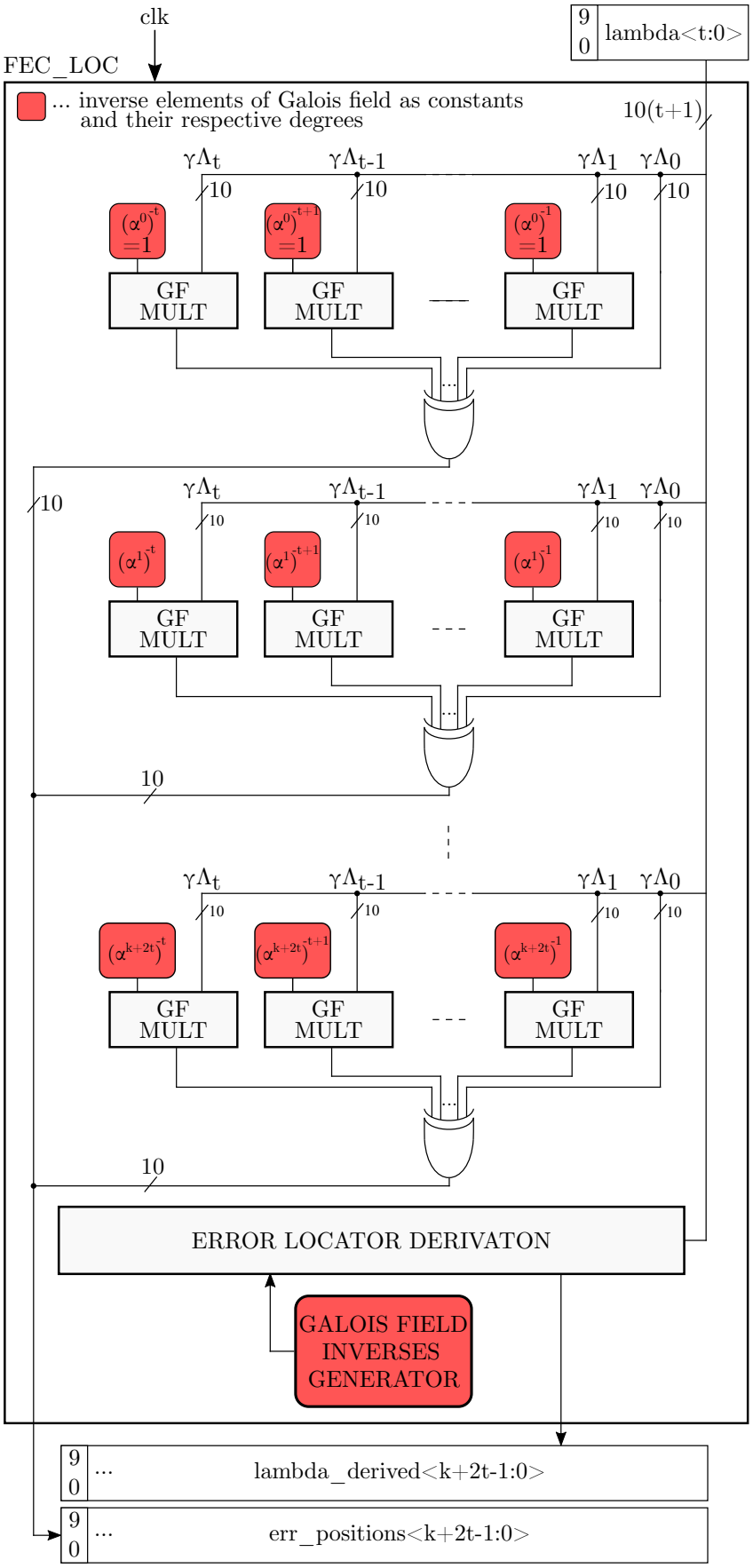

Fig. 2.13: Diagram of Chien search as the unit for error positions determination

Hardware for $\Lambda(x)$ derivation calculation differs only in terms of number of factors to be added by XOR gates as represented in 2.13. It terms of hardware requirements reduction, it is not needed to divide the Error locator polynomial by γ thanks to the fact that γ will be cancelled out in further division in Forney's algorithm, it is convenient to omit the division part in 2.17.

It is also important to note that $\gamma\Lambda(x)$ at each error position has its own derivative $\gamma\Lambda(x)'$ and therefore derivatives of higher degrees of the Magnitude polynomial $\gamma\Lambda(x)$ (degrees of 3 and more) will differ compared to the example above. It is therefore possible to form a vector of the same length as the error polynomial with derivatives of $\gamma\Lambda(x)'$ for each error position which is convenient for parallel computations omitting redundant control logic prolonging the critical path from input to output.

Hardware for Error Magnitudes Calculation

Since the results of Error locator polynomial $\gamma\Lambda(x)$ and derivation of $\gamma\Lambda'(x)$ are found, Forney's algorithm solves the Error polynomial $E(x)$ by calculating fraction for each of α^b, α^{b+1}, ..., $\alpha^{b+k+2t-1}$. Therefore, this component has been divided into three main parts. First part solves Magnitude polynomial $\gamma\Omega(x)$ and second part solves fraction with already calculated $\gamma\Lambda'(x)$. If an error position is found, multiplication of the solved fraction with respective primitive element of GF is performed. This 2-way MUX causes that if the received codeword has no errors, the entire Error polynomial $E(x)$ is set to 0 otherwise calculates result only for selected zero error positions. Since the result of the left-hand side process of the Euclidean processor is [3]:

$$\gamma\Omega(x) = \gamma\Omega_1 + \gamma\Lambda_0 = 3x + 14 \tag{2.16}$$

Divided by γ the Magnitude polynomial $\Omega(x)$ equals to:

$$\frac{\gamma\Omega(x)}{\gamma} = \Omega(x) = 6x + 15 \tag{2.17}$$

Forney's algorithm follows equation 1.39 as shown in this example [3]:

$$Y_j = X_j^{1-b}\frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})} \tag{2.18}$$

Since error positions X_j have been found in the Chien search part (here, at positions $X_j = \alpha^9$ and $X_j = \alpha^2$), error magnitudes Y_j can be calculated for these positions where $b = 1$ [3]:

$$Y_j = X_j^{1-b}\frac{6(X_j^{-1}) + 15}{\Lambda'(X_j^{-1})} = \alpha^9\frac{6\alpha^{-9} + 15}{14} = 13 \tag{2.19}$$

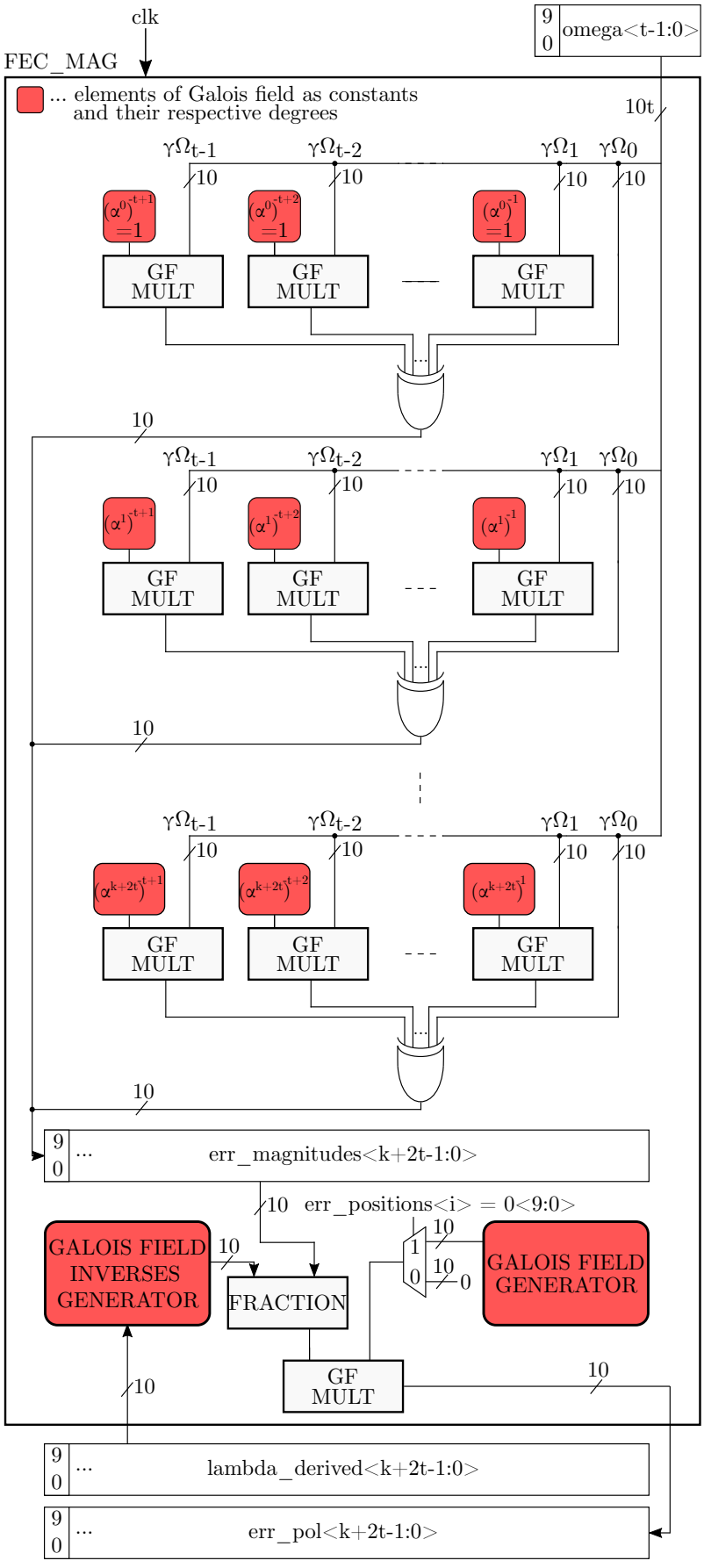

Fig. 2.14: Diagram of Forney's algorithm as a unit for error polynomial calculation

And here for α^2 [3]:

$$Y_j = X_j^{1-b}\frac{6(X_j^{-1})+15}{\Lambda'(X_j^{-1})} = \alpha^2\frac{6\alpha^{-2}+15}{14} = 2 \tag{2.20}$$

Result of this operation is then added to the input `err_codeword` and then the $R(x)$ can be corrected as shown in figure 2.15:

	x^{14}	x^{13}	x^{12}	x^{11}	x^{10}	x^9	x^8	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
$R(x) =$	1	2	3	4	5	11	7	8	9	10	11	3	1	12	12
$E(x) =$	0	0	0	0	0	*13*	0	0	0	0	0	0	*2*	0	0
$T(x) =$	1	2	3	4	5	6	7	8	9	10	11	3	3	12	12

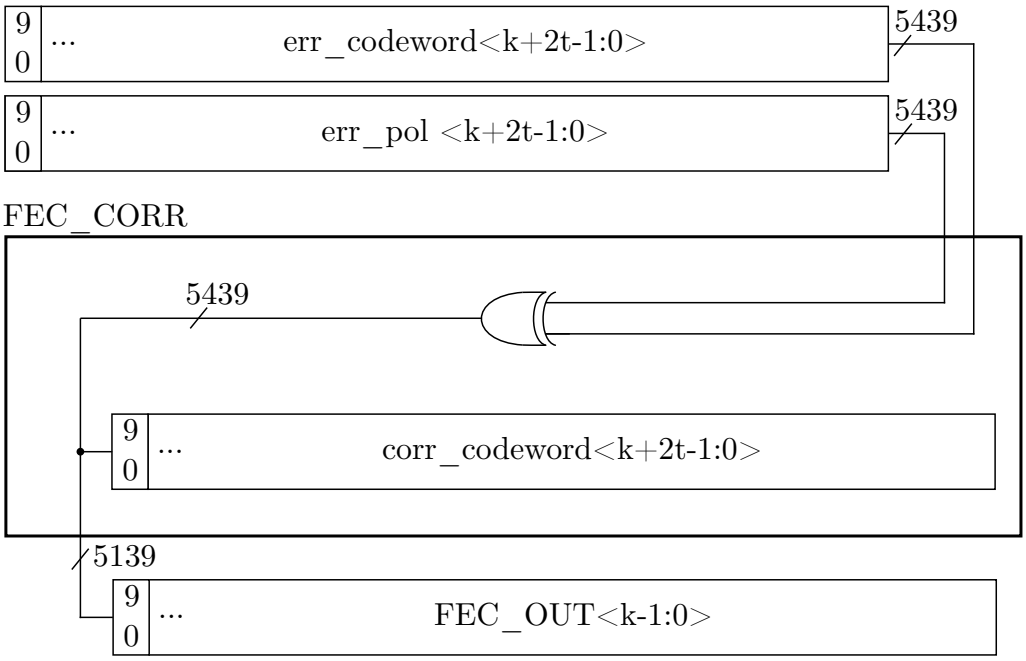

Fig. 2.15: Hardware for error correction

2.2.4 Testing

Testing of the system RS-FEC follows a straightforward procedure illustrated in figure 2.16. The main idea is to generate an error pattern in the form of polynomial representing $R(x)$. The calculated error pattern should be the same as the generated one. In addition, inner signals are also being traced for effective debugging of the code. The functionality of the algorithm was tested for shortened RS(544, 514) and $RS(256, 226)$ over $GF(2^{10})$ using $G(x)$ from Clause 119; RS(29, 15) over $GF(2^{10})$ with $G(x)$ from Clause 91 and shortened $RS(9, 5)$ over $GF(2^4)$ with reference to [3] where all possible error positions and error values were examined.

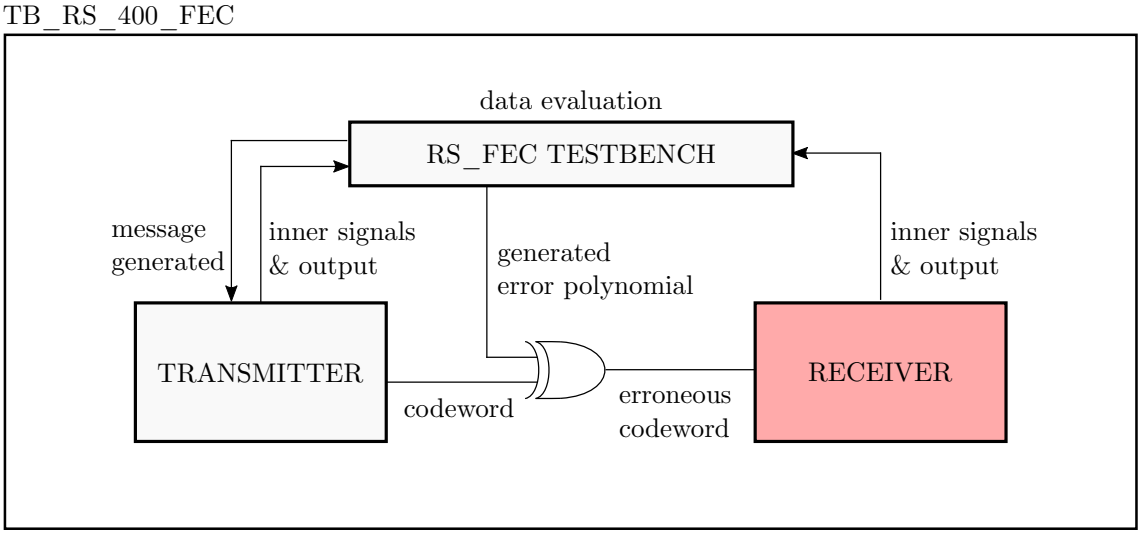

Fig. 2.16: Testing of the RS-FEC system

2.2.5 Implementation

This section summarizes the implementation part of the time-constrained system. During implementation, timing constraints for clock signals were set to 5 ns. Discussion, reflection and future improvements can be found in chapter 2.3.

Implementation Compilation flow

Development of RS-FEC for 400GE was conducted using Intel® Quartus® Prime Pro Edition Design Software [4] which supports VHDL 1987 (IEEE Standard 1076-1987), VHDL 1993 (IEEE Standard 1076-1993) and VHDL 2008 (IEEE Standard 1076-2008). Compilation flow of imported design consists of 8 main stages [30]:

- IP Generation - identifies IP components used in the project, their status and version.
- Analysis & Synthesis - performs synthesis, optimization, minimization and maps design logic to device resources, checks for design file and project errors. Results of this stage are preliminary and preserved for next stages. Design synthesis translates design source files into a form of netlist for mapping to device resources. It examines the logical completeness and consistency of the design, checks for boundary connectivity, syntax errors and also minimizes and optimizes design logic and may change or remove redundant user logic to ensure efficient use of device resources. In the end of synthesis, the Compiler generates a database of the most basic (atom) elements which design synthesis requires to implement the design in silicon. Atoms include logic cells organised into look-up tables, D flip flops I/O pins, block memory, DSP block

and connections between atoms. This can be graphically represented in RTL Viewer.

- Fitter (Place & Route) - placement and routing to specific target device is performed while respecting timing and placement constraints and any Fitter settings specified. Fitter determines the best placement and routing of logic in the target FPGA device. By default, fitter selects appropriate resources, interconnection paths and pin locations. If design logic is assigned to specific device resources, the Fitter attempts to match those requirements and optimize any other remaining unconstrained design logic. If the Fitter cannot fit the design in the current device, the compilation is terminated and reports an error message. This stage consists of 6 substages [30]:
  - Plan - places all periphery elements (I/Os and PLLs, etc.) and determines legal clock plan. Core placement or routing has not yet been performed.
  - Early Place - this is an optional stage. It places all core elements in an approximate location. This facilitates further design planning and finalizes clock planning for Intel® Stratix® 10 family and Intel® Agilex™ designs.
  - Place - places all core elements in a legal location.
  - Route - creates all routing between elements in the design.
  - Retime - moves (retimes) existing registers into Hyper-Registers for fine-grained performance improvement.
  - Fitter (Finalize) - For Intel® Stratix® 10 family devices preforms post-Route fix-up after retime stage. Also, generates Technology Map Viewer to view internal structure of the design netlist after Analysis & Synthesis, for instance, for high fanout nets examination.
- Fast Forward Timing Closure Recommendations - generates reports which estimate performance gains by making specific RTL modifications.
- Timing Analysis - analyzes and validates the timing performance of all design logic.
- Power Analysis - this is an optional stage for device power consumption estimation.
- Assembler - converts the placement and routing by Fitter into a progrmamming image for the FPGA device.
- EDA Netlist Writer - generates output files of the project for use in other EDA tools.

Galois Field Multipliers

GF multipliers appear in all the components of RS-FEC and were designed generically in two parts: part modulo and multiplication part with reference to [3], page 28 where circuit of a full 4-bit shift-and-add multiplier is described, generic version of such function has been created on gate level. Single GF multiplier consumes 39 ALMs. But, this number can be reduced during optimization process in Fitter stage of the design compilation process.

CRC Implementation

Further progress of the RS-FEC development with already designed GF multipliers could proceed in Scrambler and Descrambler synthesis and implementation. These components were realised based on a general procedure for data scrambling and descrambling (see Figure 2.4 and Figure 2.8). As assumed, some level of optimization of GF multipliers has been reached. On the other hand, results of synthesis of these components show enormous use of resources. This is because synthesis tools, in general, struggle with full optimization of large and complex functions over Galois fields, especially RS codes $RS(544, 514)$. Despite the fact that there was some level optimization, the more cycles is demanded for the recursive algorithm to be conducted, the optimization process gets less effective. In addition, it takes a long time to synthesize and implement these components. Achieved results and times of synthesis and implementation phases are summarized in following tables.

Tab. 2.3: Duration of compilation stages of RS-FEC Encoder for $RS(544, 514)$

Stage	Duration [hh:mm:ss]
Synthesis	08:47:31
Fitter	02:48:36
Timing Analyzer	00:02:17

Tab. 2.4: Duration of compilation stages of RS-FEC Descrambler for $RS(544, 514)$

Stage	Duration [hh:mm:ss]
Synthesis	00:44:02
Fitter	05:00:01
Timing Analyzer	00:03:13

Designed parametrizable system is implementable, however, timing requirements of the system with general model for scrambling and descrambling have not been fulfilled. The design therefore has to be further optimized. First, optimization

of GF multipliers towards critical path reduction might help the design to reduce resource utilization and increase maximal operation frequency for the cost of its parametrizability reduction in the future. Based on previous practice of RS-FEC designs within the academical organisation Cesnet s. z. p. o., implementation of so called "xor network", a fully optimized net of exclusive-OR gates which also might significantly reduce critical path and resource utilization, as studied in [31].

Tab. 2.5: Results of synthesis and implementation of time-constrained sequential RS-FEC Encoder for 400GE

Phase	ALUTs	Dedicated Logic Registers	Maximal Frequency
Synthesis	359516	1800	-
Implementation	388305	2420	$\tilde{6}$.81 MHz

Tab. 2.6: Results of synthesis and implementation of time-constrained sequential RS-FEC Descrambler for 400GE

Phase	ALUTs	Dedicated Logic Registers	Maximal Frequency
Synthesis	719583	1800	-
Implementation	507043	786533	$\tilde{6}$.81 MHz

Euclidean Processor Implementation

The first attempt to implement hardware of the designed and tested Euclidean processor into Intel Stratix DX FPGA resulted in exceeding available resources. This happened mainly because each generated layer of the Euclidean processor contained its own Look-up table of GF inverses of 2^m which resulted in such high hardware consumption. In addition, used tool for synthesis did not estimate correctly which registers and ALMs of certain layer will not be used in a given layer of the Euclidean processor.

Tab. 2.7: Duration of single stages of compilation of both components of Euclidean processor

Stage	Duration [hh:mm:ss]
Synthesis	00:13:37
Fitter	00:09:23
Timing Analyzer	00:00:16

In this paper, for timing estimation of the current design, single layer of the euclidean processor has been implemented. Further improvements include Euclidean

processor re-design, generally in terms of sharing registers of both sides of RS-FEC, however, functional core will remain the same. Further efforts for resources utilization minimization will be focused on implementing additional logic to single euclidean layers selecting correct results based on its current layer index which indirectly determines incoming polynomial degrees and therefore discard unnecessary logic which is never used in the current layer of a given index. Result of implementation of one layer is shown below.

Tab. 2.8: Results of synthesis and implementation of a single sequential component RS-EUC for 400GE

Phase	ALUTs	Dedicated Logic Registers	Maximal Frequency
Synthesis	11833	340	-
Implementation	13322	14568	$\tilde{2}$37.64 MHz

Chien Search Component Implementation

Synthesis and implementation of hardware for Chien Search components show that resources utilization do not exceed critical level for its implementation and timing requirements have been fulfilled. Results of these stages are shown in Tab. 2.9 and Tab. 2.12 below.

Tab. 2.9: Duration of single stages of compilation of both components for finding error locations

Stage	Duration [hh:mm:ss]
Synthesis	00:03:27
Fitter	00:36:06
Timing Analyzer	00:00:37

Tab. 2.10: Results of synthesis and implementation of both components RS-CHS for 400GE

Phase	ALUTs	Dedicated Logic Registers	Maximal Frequency
Synthesis	57506	28753	-
Implementation	66685	107813	$\tilde{2}$23.76 MHz

Forney's Algorithm implementation

In the component for Forney's algorithm computation, the most important part to focus on is the use of a constant containing Galois field inverses. This constant

contains 2^m bits and is therefore the largest in the design. This constant is used for division by respective GF derivative calculated in Chien Search component for each error position. There is actually $n = 544$ error positions but only $t = 30$ positions can be corrected. The hardware in Forney's algorithm calculates error magnitude for each n. This means that the 2^m elements has to be generated 544 times. This is actually the cause of such a large hardware utilization which has to be reduced to only, ideally, t-times 2^m. This approach might be fulfilled by a sorting algorithm which will shift error positions in a given direction of an array and accumulate them. Then, with t-times for loop, pointer will find respective inverse elements in the constant array of GF inverses based on the given positions.

Tab. 2.11: Duration of compilation stages of a single entity of RS-FOR conducting Forney's algorithm

Stage	Duration [hh:mm:ss]
Synthesis	00:46:07
Fitter	04:31:17
Timing Analyzer	00:03:05

Current design expects the result of the Forney's algorithm to be calculated in three clock cycles. Further extensions will probably require one more clock cycle delay.

Tab. 2.12: Results of synthesis and implementation of a single time-constrained sequential entity RS-FOR for 400GE

Phase	ALUTs	Dedicated Logic Registers	Maximal Frequency
Synthesis	521565	14897	-
Implementation	539392	977902	$\tilde{2}$45.16 MHz

2.3 Discussion and Reflection

In this work, main goals of the RS-FEC development have been reached. Since the very beginning of the development, the design was focused mainly on maximal data throughput, exploiting benefits of the FPGA technology, its parametrizability and resource usage, however, in lesser extent. Full parametrizability of the design has been reached which enables its reusability and possible creation of various forms of the system for future uses or its implementation in different areas. In addition, including Galois fields generation, its inverses and various degrees of inverses using VHDL subprograms without using additional scripts might noticeably reduce development speed of future Ethernet platforms.

In terms of the overall functionality, the system has been tested and full parametrizability has been successfully verified. It is therefore possible to modify this system for various shortened forms of RS-FEC by changing these main generic variables: m, n, k, $2t$ and number of PCS lanes for codewords distribution. Thanks to generic PCS lanes interleaving and deinterleaving the system can be implemented also for 200GE or other projects.

On the other hand, current state of art requires further optimization, especially CRC and Euclidean processor units, and control logic, which is the plan for future work. In the end, the focus of the development changed from the effort for maximal data throughput and timing requirements to balance between resources utilization and data throughput where resources play crucial role for implementation of such a large system. Based on unbiased user experience from this development work, three key points should ensure its further safe progress: identification and restriction of resource-intensive look-up tables generation, sharing resources between two parts of the RS-FEC and use of best practices from previous RS-FEC implementations to combine the best approaches to this topic.

In terms of the resources minimization, significant reduction of ALUTs might be reached by selecting error positions only at the Chien search component and form the error polynomial at the Forney's algorithm in the last stage of the pipeline of the component. This will require an extra logic at the Chien search and one clock cycle delay but significantly reduce number of ROM-based Look-up tables in the Forney's algorithm, meaning using Forney's equation t-times only compared to generating these inverses of the $GF(2^m)$ n-times.

Summary

In this Master's Thesis, fully parametrizable Reed-Solomon self-correcting algorithm for 400 GE has been successfully designed, its function verified in simulations, optimized and implemented. In the theoretical part of this work, Reed-Solomon error correcting algorithm is described including FPGA technology and Galois finite field algebra including a general overview of modern networking and implications to current high-speed Ethernet. Based on the theoretical part of the work it turned out that due to the Finite field algebra it will not be possible to use dedicated DSP blocks of the FPGA chip. Hence, the entire algorithm was stored in LUTs.

In the practical part, design and testing of RS-FEC system is discussed, used hardware for its realization and future challenges including best practices are summarized. The very first attempt to implement the system resulted in exceedingly large resource utilization and therefore optimizations were needed to be conducted. The main cause of this were complicated and hardware-intensive inverse circuits of Euclidean processor layers. It turned out that it is not feasible to implement fully parallel Euclidean processor unit including Forney's algorithm. Based on user experience from this work of VLSI design of Forney's algorithm and Chien search it was found that it is better to calculate single degrees of inverses of Galois field elements on chip than storing them in ROM-based Look-up tables. However, in the current state of art, all components of the RS-FEC are implementable and further optimization is required. Major challenge of this project and key for its implementation was to balance hardware resources utilization from the previous focus on timing fulfilment, which is mainly the task for further CRC unit optimization.

This work suggests using ROM-based Look-up tables in lesser extent. It was found that implementation of ROM-based Look-up tables at the smallest scale possible is the key for successful implementation of this system in Intel® Stratix® 10 DX FPGA. Hence, employing Euclidean processor units for solving Key equations is discussed. Since there are two encoders and decoders employed for each codeword respectively, it is therefore better to consider shared resources between these two parts of RS-FEC system and using algorithms based on minimal utilization of ROM-based Look-up tables.

Future orientation of this work will be focused mainly on ALUTs usage minimization and increasing throughput for CRC unit. In particular, significant benefit of this system is its parametrizability which enables faster further optimization process which applies especially for Euclidean processor. Another benefit is its simple implementability for various scales of the code underlining its variability also for various future uses.

Bibliography

[1] David J. Law. Ieee 802.3 industry connections bandwidth assessment part ii. page 57, April 2020. URL: http://www.ieee802.org/3/ad_hoc/bwa2/BWA2_Report.pdf.

[2] Ieee standard for ethernet amendment 4: Physical layer specifications and management parameters for 1 gb/s operation over a single twisted-pair copper cable. *IEEE Std 802.3bp-2016 (Amendment to IEEE Std 802.3-2015 as amended by IEEE Std 802.3bw-2015, IEEE Std 802.3by-2016, and IEEE Std 802.3bq-2016)*, pages 1–211, 2016.

[3] C.K.P. Clarke. Bbc r & d white paper whp 031, Jan 2002. URL: https://downloads.bbc.co.uk/rd/pubs/whp/whp-pdf-files/WHP031.pdf.

[4] Intel® quartus® prime software suite. URL: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html.

[5] Tcp/ip vs. osi: What's the difference between the two models?, Nov 2017. URL: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=2820&lang=en.

[6] 2019 roadmap, 2019. URL: https://ethernetalliance.org/technology/2019-roadmap/.

[7] J. Friedrich. Methods for bulk growth of inorganic crystals: Crystal growth. In *Reference Module in Materials Science and Materials Engineering*. Elsevier, 2016. doi:10.1016/b978-0-12-803581-8.01010-9.

[8] Sh.K. Amin, Mai Hassan, S El-Sherbiny, and H. Abdallah. An overview of production and development of ceramic membranes. 11:7708–7721, 01 2016.

[9] Behraad Bahreyni. *Fabrication and design of resonant microdevices*. W. Andrew Inc, Norwich, NY, 2008.

[10] David H. K. Hoe, L. P. Deepthi Bollepalli, and Chris D. Martinez. FPGA fault tolerant arithmetic logic: A case study using parallel-prefix adders. *VLSI Design*, 2013:1–10, November 2013. doi:10.1155/2013/382682.

[11] J Serrano. Introduction to FPGA design. *CERN*, 2008. URL: https://cds.cern.ch/record/1100537, doi:10.5170/CERN-2008-003.231.

[12] Venkata P. Yanambaka, Saraju P. Mohanty, Elias Kougianos, Dhruva Ghai, and Garima Ghai. Process variation analysis and optimization of a FinFET-based

VCO. *IEEE Transactions on Semiconductor Manufacturing*, 30(2):126–134, May 2017. doi:10.1109/tsm.2017.2669314.

[13] B Xue. Vlsi design of a reed-solomon decoder for gigabit automotive ethernet, 2016. Thesis Supervisor: C.H. (Kees) van Berkel (Supervisor 1), Marc C.W. Geilen (Supervisor 2), Özgün Paker (External coach).

[14] Test solutions to validate 400ge devices and networks, 2017. URL: https://ethernetalliance.org/wp-content/uploads/2018/02/Ixia-400GE-Test-Solutions.pdf.

[15] B. BAILEY. Using fpgas for ai, Dec 2019. URL: https://www.edn.com/design/integrated-circuit-design/4314765/Taking-a-bite-out-of-power-techniques-for-low-power-ASIC-design.

[16] P. Sundararajan. High performance computing using fpgas, Sep 2010. URL: http://www.bdtic.com/download/Xilinx/WP375.pdf.

[17] IEEE standard for ethernet - amendment 10: Media access control parameters, physical layers, and management parameters for 200 gb/s and 400 gb/s operation. doi:10.1109/ieeestd.2017.8207825.

[18] Why is big data so important in today's world?, Feb 2019. URL: https://datafloq.com/read/why-is-big-data-so-important-in-todays-world/2674.

[19] William A Geisel. Tutorial on reed-solomon error correction coding. *NASA*, 1990. URL: https://ntrs.nasa.gov/search.jsp?R=19900019023.

[20] William Stallings. Gigabit ethernet: From 1 to 100 gbps and beyond. *The Internet Protocol Journal*, pages 20–32, 2015. URL: https://pdfs.semanticscholar.org/0c43/eb117949f3eb6d5bd361b97f5a90d9f4266e.pdf.

[21] David J. Law. Ieee 802.3 industry connections bandwidth assessment part ii. page 40, April 2012.

[22] Information technology – open systems interconnection – basic reference model: The basic model, Jul 1994. URL: https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=2820&lang=en.

[23] et al. Savio. A physical coding sublayer for gigabit ethernet over pof, Oct 2010. URL: https://www.researchgate.net/publication/273445103_A_PHYSICAL_CODING_SUBLAYER_FOR_GIGABIT_ETHERNET_OVER_POF.

[24] R. C. Seals and G. F. Whapshott. *Programmable Logic: PLDs and FPGAs*. Macmillan Education UK, 1997. doi:10.1007/978-1-349-14003-9.

[25] M. HUTTON. Understanding how the new intel® hyperflex™ fpga architecture enables nextgeneration high-performance systems. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01231-understanding-how-hyperflex-architecture-enables-high-performance-systems.pdf.

[26] Ab Al-Hadi Ab Rahman, Anatoly Prihozhy, and Marco Mattavelli. Pipeline synthesis and optimization of FPGA-based video processing applications with CAL. *EURASIP Journal on Image and Video Processing*, 2011(1), November 2011. doi:10.1186/1687-5281-2011-19.

[27] Intel® stratix® 10 device datasheet, Mar 2020. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10_datasheet.pdf.

[28] M. Morii and M. Kasahara. Generalized key-equation of remainder decoding algorithm for reed-solomon codes. *IEEE Transactions on Information Theory*, 38(6):1801–1807, 1992.

[29] 800g specification. URL: https://ethernettechnologyconsortium.org/wp-content/uploads/2020/03/800G-Specification_r1.0.pdf.

[30] Intel® quartus® prime pro edition user guide: Design compilation, Apr 2020. URL: www.intel.com/content/www/us/en/programmable/documentation/zpr1513988353912.html#jbr1443197641054.

[31] Jan Velecký. Implementation of self-correcting codes for 100 gb/s ethernet, 2017. Thesis Supervisor: Ing. Lukáš Kekely.

List of symbols, physical constants and abbreviations

RS	Reed-Solomon
FPGA	Field Programmable Gate Array
GE	Gigabit Ethernet
VLSI	Very-Large-Scale Integration
FEC	Forward Error Correction
PHY	Ethernet Physical Layer
LAN	Local Area Network
MAC	Medium Access Control
OSI	Open Systems Interconnection
DLL	Data Link Layer
LLC	Local Link Control
PCS	Physical Coding Sublayer
PMD	Physical Media Dependent
PMA	Physical Medium Attachment
BER	Bit Error Rate
DDR	Double Data Rate
PCI	Peripheral Component Interface
FSB	Front Side Bus
QPI	Quick Path Interconnect
DFF	D-type Flip Flop
DSP	Digital Signal Processing
RTL	Register Transfer Level
CPU	Central Processing Unit
GPU	Graphics Processing Unit
I/O	Input/Output
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
ALUT	Adaptive Look-up Table
UV	Ultraviolet
HPC	High Performance Computing
CLB	Configurable Logic Block
RAM	Random Access Memory
BLE	Basic Logic Element
LUT	Look-Up-Table
DLLs	Delay Locked Loops
PLLs	Phase Locked Loops

ASICs	Application Specific Integrated Circuits
IP	Intellectual Property
MAC	Medium Access Control
SoC	System on Chip
ALMs	Adaptable Logic Modules
DRAM	Dynamic Random Access Memory
SRAM	Static Random Access Memory
GPUs	Graphics Compressing Units
CRC	Cyclic Redundancy Check