Efficient ASIC Architectures for Low Latency Niederreiter Decryption

Daniel Fallnich[1,2], Shutao Zhang[1] and Tobias Gemmeke[1]

[1] RWTH Aachen University, Aachen, Germany
{zhang,gemmeke}@ids.rwth-aachen.de
[2] now with IBM, Böblingen, Germany
fallnich@ibm.com

Abstract.
Post-quantum cryptography addresses the increasing threat that quantum computing poses to modern communication systems. Among the available "quantum-resistant" systems, the Niederreiter cryptosystem is positioned as a conservative choice with strong security guarantees. As a code-based cryptosystem, the Niederreiter system enables high performance operations and is thus ideally suited for applications such as the acceleration of server workloads. However, until now, no ASIC architecture is available for low latency computation of Niederreiter operations. Therefore, the present work targets the design, implementation and optimization of tailored architectures for low latency Niederreiter decryption. Two architectures utilizing different decoding algorithms are proposed and implemented using a 22nm FDSOI CMOS technology node. One of these optimized architectures improves the decryption latency by 27% compared to a state-of-the-art reference and requires at the same time only 25% of the area.

Keywords: Application-Specific Architecture · Post-Quantum Cryptography · Niederreiter Cryptosystem · Hardware Implementation

1 Introduction

Advances in the development of quantum computers are raising concerns that large-scale quantum computers could threaten the confidentiality of modern communications systems, provided by cryptographic algorithms. In order to maintain secure communications, post-quantum cryptography (PQC) aims to defend against attacks from quantum computers by the introduction of so-called quantum-resistant cryptosystems. To assess the suitability of these cryptosystems with respect to diverse applications, the National Institute of Standards and Technology (NIST) is currently evaluating post-quantum key encapsulation mechanisms (KEM) and digital signature algorithms, with the goal to standardize at least one system from each category. For key encapsulation, one of the second round finalists is a scheme called *Classic McEliece*, which is based on the Niederreiter cryptosystem [AASA+20]. This code-based Niederreiter cryptosystem, whose associated characteristics allow for high-speed operations, represents a conservative choice among quantum-resistant systems. Confidence in the security of this cryptosystem follows from an extensive history of cryptanalysis.

Despite its conservative and well-researched security guarantees, the Niederreiter cryptosystem never experienced wide-spread adoption, due to relatively large key sizes. Nevertheless, the accomplishment of high-speed operations as well as strong security levels suggest the suitability of this cryptosystem for applications in data centers and other application fields, where security and performance are critical. These fields are expected to

rank among the early adopters of post-quantum cryptography, where hardware-accelerated high-speed operations are desirable. However, up to now, no ASIC architecture has been proposed for the Niederreiter cryptosystem. Therefore, the present work targets the design and implementation of ASIC architectures for the code-based Niederreiter cryptosystem. In consistency with the application scenarios described above, the focus thereby lies on facilitating a low latency decryption operation of the Niederreiter system with high area efficiency.

The development of highly efficient hardware architectures for the Niedereiter cryptosystem is aided by employing suitable decoding algorithms for low latency processing. Additionally, an appropriate polynomial evaluation approach which also takes remaining steps of Niederreiter decryption into account allows for advantageous operational characteristics. Where possible, algorithmic optimizations are introduced, in order to improve either the decryption latency or area footprint of an associated architecture. Corresponding ASIC architectures are derived and subsequently optimized, thus allowing for low latency computation of the Niederreiter decryption operation.

The remainder of this paper is structured as follows: Section 2 gives a brief background of the Niederreiter cryptosystem as well as binary Goppa codes and their associated decoding algorithms. Section 3 provides an overview of previous hardware implementation approaches of this cryptosystem, while the proposed ASIC architectures are detailed in Section 4. Some implementation aspects of these architectures are described in Section 5. Section 6 discusses results of the proposed architectures and compares them to previous approaches. Finally, Section 7 summarizes the findings and results of this paper.

2 Code-Based Cryptography

The use of error-correcting codes in the design of cryptosystems was already proposed in 1978 by Robert McEliece [McE78]. The code-based Classic McEliece KEM builds upon the *Niederreiter* cryptosystem, which is a "dual" variant of the McEliece cryptosystem [ABC+20]. However, the aforementioned KEM bears the name of the original proposal by Robert McEliece, which used binary Goppa codes and remains unbroken, apart from parameter modifications. Niederreiter's variant of this system allows for an increase in performance, when considering key encapsulation [WSN17]. However, the original publication also proposed the use of Reed-Solomon codes, which led to successful attacks of this system [SS92]. Therefore, this work considers the Niederreiter cryptosystem using binary Goppa codes.

2.1 Binary Goppa Codes

Binary Goppa codes are a class of linear error-correcting codes, which, due to their structure, exhibit certain characteristics, that are advantageous for applications in code-based cryptography. As a sub-class of Goppa codes, binary Goppa Codes operate in $GF(2^m)$ and possess a minimum distance of $d_{\min} \geq 2t + 1$, where t is the number of correctable errors [Ber73].

A binary Goppa code is defined by a monic *generator polynomial* $g(x)$ and a *support vector* of field elements L, described by

$$g(x) = x^t + \sum_{i=0}^{t-1} g_i x^i, \quad g_i \in GF(2^m) \tag{1}$$

and

$$L = (L_0, ..., L_{n-1}), \quad L_i \in GF(2^m), \tag{2}$$

where n is the code length [HP03]. When the generator polynomial $g(x)$ is an irreducible polynomial, the resulting code is called an irreducible Goppa code and in the following this property is assumed for all discussed Goppa codes.

2.2 Decoding Binary Goppa Codes

In order to decode binary Goppa codes and recover a transmitted codeword $c \in GF(2^n)$ from an erroneous codeword $\tilde{c} = c + e$ with error vector e, a decoding procedure is applied, which comprises three major steps: *Syndrome computation*, *solving the key equation* and *determining the roots of the error locator polynomial* [McE02].

The first step in the decoding process is the computation of the syndrome polynomial S. This syndrome is obtained from a received word $\tilde{c}$ as the product $S = H \times \tilde{c}$, where the $t \times n$ matrix H is called a parity check matrix, with $H_{ij} = L_{j-1}^{i-1}/g(L_{j-1})$ [LC87].

Subsequently, a *decoding algorithm* is utilized in conjunction with the computed syndrome in order to construct an *error locator polynomial* λ, whose roots correspond to the error positions. The process of computing the aforementioned error locator polynomial is also referred to as solving the key equation [Ber15] given by $S(x)\lambda(x) \equiv \omega(x) \bmod g(x)$, where the error evaluator polynomial ω can be omitted in the present case of binary codes [Hey13]. Since the error locations are represented by the roots of the error locator polynomial, these locations can subsequently be determined by evaluating the error locator polynomial λ for all support elements L_i, where the indices of support elements that are roots of λ indicate an erroneous position. By obtaining the error vector from the roots of λ, the initial codeword can be reconstructed as $c = \tilde{c} - e$, which equals $c = \tilde{c} + e$ in the binary case.

Various decoding algorithms are available for solving the key equation of binary Goppa codes. Since Goppa codes are a sub-class of alternant codes, alternant decoding algorithms can be utilized in the decoding process for Goppa codes [MBR15]. However, when applying *general decoding algorithms*, which were not specifically designed for Goppa codes, only $t/2$ errors can be corrected directly, while the Niederreiter cryptosystem requires a correction capability of t errors. This limitation is overcome by computing a *double-sized syndrome* $S^{(2)} = H^{(2)} \times (S|0)$ [HG13], with the double-sized parity check matrix

$$H^{(2)} = \begin{bmatrix} \frac{1}{g^2(L_0)} & \frac{1}{g^2(L_1)} & \cdots & \frac{1}{g^2(L_{n-1})} \\ \frac{L_0}{g^2(L_0)} & \frac{L_1}{g^2(L_1)} & \cdots & \frac{L_{n-1}}{g^2(L_{n-1})} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{L_0^{2t-1}}{g^2(L_0)} & \frac{L_1^{2t-1}}{g^2(L_1)} & \cdots & \frac{L_{n-1}^{2t-1}}{g^2(L_{n-1})} \end{bmatrix} . \tag{3}$$

By using $S^{(2)}$ as an input for a general decoding algorithm instead of S, up to t errors are correctable.

Even though the approach stated above allows for the selection of a decoding algorithm from a broad spectrum, this work focuses on variants of the two most commonly employed algorithms that allow for efficient hardware implementations. These Algorithms, *Patterson's algorithm* as well as the *inversionless Berlekamp-Massey algorithm*, will be described in the following.

2.2.1 Patterson's Algorithm

In 1975, Patterson proposed an algorithm that was specifically designed for the purpose of decoding binary Goppa codes. This algorithm, whose pseudocode is shown in Algorithm 1, builds upon the ideas of the decoding algorithms by Berlekamp and Massey as well as Sugiyama et al. [Pat75]. After syndrome inversion, which corresponds to Sugiyama's approach, Patterson's decoding algorithm employs a decomposition of the inverted syndrome

T. Patterson's decoding procedure thereby exploits unique properties of irreducible binary Goppa codes [EOS06]. Due to this tailored approach, Patterson's algorithm is capable of correcting t errors directly, without the use of a double-sized syndrome.

Algorithm 1 Patterson's algorithm for decoding binary Goppa codes

function PATTERSON'S ALGORITHM$\big(S, g(x)\big)$
  $T(x) \leftarrow S^{-1} \bmod g(x)$
  if $T(x) = x$ **then**
    $\lambda(x) \leftarrow x$
    return $\lambda(x)$
  end if
  $R(x) \leftarrow \sqrt{T(x) + x}$
  $(a(x), b(x)) \leftarrow \mathrm{EEA}(R(x), g(x))$ with $\deg(a) \leq \frac{t-1}{2}$, $\deg(b) \leq \frac{t}{2}$
  $\lambda(x) \leftarrow a(x)^2 + x \cdot b(x)^2$
  return $\lambda(x)$
end function

2.2.2 Inversionless Berlekamp-Massey Algorithm

Another commonly employed decoding algorithm for binary Goppa codes is the *Berlekamp-Massey algorithm* (BM). Even though this algorithm relies on mostly simple field operations, it also requires field inversions in an iterative loop. In order to facilitate an efficient hardware implementation without repeated inversions, a variant of the BM algorithm is used for one of the proposed ASIC architectures, called the *inversionless Berlekamp-Massey algorithm* (iBM) [Bur71].

The pseudocode of the iBM algorithm is shown in Algorithm 2. It can be seen, that during the coefficient update step (line 5 of Algorithm 2), the coefficients of the error-locator polynomial are multiplied by a scalar field element γ, which causes the inversion of the discrepancy in the subsequent steps to vanish. Due to this scalar multiplication of the coefficients of $\lambda(x)$, the resulting error-locator polynomial is a scalar multiple of the polynomial determined by the original BM algorithm [SS01]. Since only the roots of $\lambda(x)$, which remain unaffected by scalar multiplication, are of interest for decoding Goppa codes, this property of the iBM algorithm does not impact the final solution.

Algorithm 2 Inversionless Berlekamp-Massey algorithm

1: **function** INVERSIONLESS BERLEKAMP-MASSEY$\big(S\big)$
2: $\lambda(x) \leftarrow 1$, $b(x) \leftarrow 1$, $l \leftarrow 0$, $\gamma \leftarrow 1, \delta \leftarrow 0$
3: **for** k **from** 0 **to** $2t - 1$ **do**
4: $d \leftarrow \delta$, $\delta \leftarrow \sum_{i=0}^{t} \lambda_i \cdot S_{k-i}$
5: $\lambda(x) \leftarrow \gamma \cdot \lambda(x) - \delta \cdot b(x) \cdot x$
6: **if** $\delta = 0$ or $l < 0$ **then**
7: $b(x) \leftarrow x \cdot b(x)$, $l \leftarrow l + 1$, $\gamma \leftarrow \gamma$
8: **else**
9: $b(x) \leftarrow \lambda(x)$, $l \leftarrow -1 - l$, $\gamma \leftarrow \delta$
10: **end if**
11: **end for**
12: **return** $\lambda(x)$
13: **end function**

2.3 Niederreiter Cryptosystem

In 1986, Niederreiter proposed an asymmetric code-based cryptosystem [Nie86], which is considered a variant of the McEliece cryptosystem [BLP08]. Instead of encoding a plaintext message in a codeword, Niederreiter's approach employs the error vector e as the plaintext and consequentially the syndrome S as the ciphertext, thus the codeword is thereby negligible. The resulting cryptosystem exhibits a smaller ciphertext than McEliece's system and requires no CCA2-conversion [HG13], thus this system is favorable for key exchange applications. Nevertheless, the Niederreiter cryptosystem is equivalent to the McEliece cryptosystem in terms of security [LDW94]. In the following, a "modern" variant[1] of the Niederreiter cryptosystem will be employed. The operations of the Niederreiter cryptosystem are summarized in Algorithm 3, with a notation similar to the work of Wang et al. [WSN18].

Key generation of the Niederreiter cryptosystem allows for the selection of system parameters m (field size), n (code size), t (maximum error number) and $\bar{k}$ (code dimension). With these parameters, a random permutation $L = (L_0, ..., L_{n-1})$, with $L_i \in GF(2^m)$ of n distinct field elements is selected, which is called the *support vector* [WSN18]. By storing a permutation implicitly in the support vector, the use of a permutation matrix $\bar{P}$, as it is employed in the McEliece cryptosystem, can be avoided [HG13]. Thereafter, a random irreducible *generator polynomial* $g(x)$ of degree t is chosen. The support vector and generator polynomial subsequently allow for the computation of the $t \times n$ parity check matrix H. In the "modern" variant of the Niederreiter cryptosystem, this parity check matrix is then transformed into its systematic form $H = [I_{mt}|K_{\text{pub}}]$, which reduces the size of the public key to $mt \times (n - mt)$ [WSN18]. Afterwards, the public key is given by the non-systematic part of H, i.e. K_{pub}, while the private key K_{priv} comprises the generator polynomial $g(x)$ as well as the support vector L.

[1] The distinction between "classic" and "modern" variants follows the nomenclature of [HG13]. Other authors, e.g. [WSN17], apply the modern variant without further differentiation.

Algorithm 3 Operations of the Niederreiter Cryptosystem

System parameters $m, n, \bar{k}, t$
function KEY GENERATION$(m, n, \bar{k}, t)$
    Select a random permutation $L = (L_0, ..., L_{n-1})$ of n field elements in $GF(2^m)$.
    Select a random irreducible polynomial $g(x)$ of degree t.
    Determine the associated parity check matrix H.
    Find the systematic form of H using Gaussian elimination as $H = [I_{mt}|K_{\text{pub}}]$.
    return the public key K_{pub} and the private key $K_{\text{priv}} = (g(x), L)$.
end function
function ENCRYPTION(K_{pub}, e)
    $S \leftarrow [I_{mt}|K_{\text{pub}}] \times e$
    return the ciphertext S.
end function
function DECRYPTION$(K_{\text{priv}} = (g(x), L), S)$
    Determine the double-sized parity check matrix $H^{(2)}$ according to Equation 3.
    Determine the double-sized syndrome $S^{(2)} \leftarrow H^{(2)} \times (S|0)$
    Retrieve the error-locator polynomial $\lambda(x)$ from $S^{(2)}$ by use of a decoding algorithm.
    Retrieve the roots of $\lambda(x)$ as e, i.e. $e_i = 1 \iff \lambda(L_i) = 0$.
    return the plaintext e.
end function

Table 1: System parameters, security levels and key sizes for the Niederreiter cryptosystem.

Security Level	Parameters			Size $g(x)$ [kbit]	Size L [kbit]	Size H [kbit]	Size K_{priv} [kbit]	Size K_{pub} [kbit]
	n	m	t					
80 bit	1632	11	33	0.374	18	592	18	461
128 bit	2960	12	56	0.684	36	1989	37	1538
256 bit	6624	13	115	1.508	86	9903	88	7668
266 bit	6960	13	119	1.560	90	10767	92	8374

Encryption in the Niederreiter cryptosystem requires the representation of a plaintext message as an error vector e of Hamming weight t. Various *constant-weight encoding* algorithms were proposed for this task, e.g. [HCG17], although they are not further considered here. Encryption in the Niederreiter cryptosystem is then equivalent to syndrome computation of binary Goppa codes, given by the product of the plaintext e and the parity check matrix $[I_{mt}|K_{\text{pub}}]$, yielding the ciphertext, i.e. the syndrome $S = [I_{mt}|K_{\text{pub}}] \times e$.

Decryption of Niederreiter ciphertexts, which is the subsequent focus of the present work, is accomplished by decoding the error vector e (the plaintext) from the syndrome S (the ciphertext). Assuming the application of a general decoding algorithm, the first step in the decryption process is the computation of the *double-sized syndrome* as the product $S^{(2)} = H^{(2)} \times (S|0)$, where $H^{(2)}$ denotes the $2t \times n$ *double-sized parity check matrix* given by Equation 3 and $(S|0)$ denotes the syndrome, right-padded with zeros to n bit. Afterwards, an error-locator polynomial $\lambda(x)$ of degree t is constructed from $S^{(2)}$ by means of a decoding algorithm. By evaluating the error-locator polynomial for each element L_i of the secret support L, its roots can be found, where indices of support elements that are roots of $\lambda(x)$ correspond to indices of bits in the error-vector e for which $e_i = 1$.

2.3.1 Parameter Selection

The system parameters n, m and t of the Niederreiter cryptosystem allow for a tradeoff between security level and performance. A selection of parameter sets considering effects from attacks on the Niederreiter system [BLP08] are listed in Table 1. Due to its high-speed operations and confidence in its security guarantees, the Niederreiter cryptosystem is inherently well suited for applications in critical environments, such as data centers. The proposed architectures target this scenario with its high-speed and low-area objectives. Therefore, architectures supporting long-term security for critical data are appropriate, where the selected parameter set should also comply with the Classic McEliece KEM proposal. Due to this reason, a parameter set resulting in a security level of 266 bit was selected, with the associated parameters $n = 6960$, $m = 13$ and $t = 119$. This parameter set still provides a 128 bit "quantum-resistant" security level when considering attacks using quantum computers executing Grover's algorithm [WSN18] and follows the recommendations for PQC given in [ABB+15].

3 Previous Work

Due to its history and associated position as a reliable conservative choice, the McEliece cryptosystem has received significantly more attention than the variant proposed by Niederreiter. Nevertheless, several implementations of the Niederreiter cryptosystem do exist, which are listed in Table 2, in addition to McEliece implementations for comparison. "Low-reiter", for instance, is a Niederreiter software implementation, which targets 8-bit AVR microcontrollers and provides a security level of 80 bit, while utilizing Patterson's

Table 2: Overview of code-based PQC hardware implementations.

Design[a]		Device	Security Level [Bit]	Decoding Algorithm	f_{clk} [GHz]	Slices[b]	Latency[c] [µs]		
							Key.	Enc.	Dec.
[SWM+10]	(M)	Xilinx LX110T	103	Patterson	0.163	14537	9000	500	1290
[GV14]	(M)	Xilinx XC6VHX255T	128	Patterson	0.254	5357	-	4.74	920
[MBR15][d]	(M)	Xilinx 3AN-1400	80 — 256	Arguello	0.123	2108	-	-	601
[CCKA21]	(M)	Xilinx XC7K70T	266	Patterson	0.050	n.d.	-	n.d.	n.d.
[HG13]	(N)	Xilinx LX240	80	Sugiyama	0.220	2474	-	0.91	49.72
[HDYC18]	(N)	Xilinx XC6VLX240T	76.5	Patterson	0.250	4252	-	1.41	798.57
[WSN18]	(N)	Altera 5SGXEA7N	256	BM	0.248	121806	3896.52	21.83	68.77

[a] (M) = McEliece, (N) = Niederreiter

[b] Results are given for the total slices of an implementation.

[c] Key. = key generation, Enc. = encryption, Dec. = decryption

[d] Results are listed for a 128 bit parameter set.

algorithm for decoding [Hey10]. Considering FPGA implementations, architectures for Niederreiter encryption and decryption with 80 bit security were proposed in [HG12] and [HG13]. This led to two designs employing Patterson's algorithm and the BM algorithm, respectively, although the results are not directly transferable[2] to Niederreiter implementations conforming to the Classic McEliece KEM submission, which was proposed later. In 2018, Hu et al. presented an ASIP design implemented on an FPGA, which supports Niederreiter encryption as well as decryption. This architecture is furthermore capable of generating signatures using the Niederreiter cryptosystem [HDYC18]. Lastly, an FPGA implementation of the complete Niederreiter system providing long-term security with a security level of 266 bit is given in [WSN17] and [WSN18]. The aforementioned implementation is scalable with respect to different parameter sets and employs the BM algorithm for constant-time decryption. To the best of our knowledge, no ASIC architecture apart from a HLS-based comparison of PQC KEM algorithms [BSNK19] exists for the Niederreiter cryptosystem.

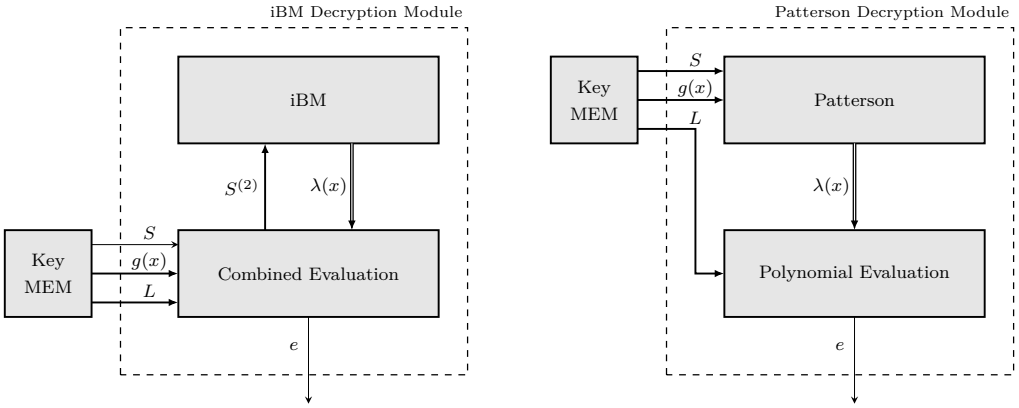
4 Architecture Design

Two application-specific hardware architectures were designed in the context of the present work, an iBM-based as well as a Patterson-based architecture. An overview of these architectures is shown in Figure 1. For both decryption modules, ciphertext and private keys are assumed to be located in external key memories, which facilitates a fair comparison of architectures without the influence of a constant large area contribution of the key memory.

The iBM-based decryption module comprises two sub-modules: A *combined evaluation module* for computation of double-sized syndromes and polynomial evaluation as well as an *iBM module* for computation of the error-locator polynomial. A decryption operation is executed by first computing the double-sized syndrome using the combined evaluation module. This double-sized syndrome is then employed by the iBM module to construct an error-locator polynomial $\lambda(x)$ from the syndrome. This error-locator polynomial is fed back into the combined evaluation module, which evaluates the polynomial at all points corresponding to support vector elements L_i and returns the plaintext represented by an error-vector e, thus completing the decryption.

The Patterson-based decryption module exhibits less complex sequencing, as utilization of Patterson's algorithm for decryption renders the use of a double-sized syndrome unnecessary. Therefore, the ciphertext syndrome and generator polynomial are

[2] This is due to the fact, that the implementation in [HG13] assumes the double-sized parity check matrix $H^{(2)}$ as a part of the private key.



(a) iBM Decryption (b) Patterson Decryption

Figure 1: Overview of the proposed (a) iBM-based and (b) Patterson-based Niederreiter decryption architectures.

applied directly in the *Patterson decoding module*, in order to obtain the error locator polynomial. Using the secret support vector, the error-locator polynomial is then evaluated by the *polynomial evaluation module*, which subsequently provides the plaintext bits.

4.1 Finite Field Arithmetic

Since arithmetic modules for finite field arithmetic represent the fundamental components of the aforementioned modules of Niederreiter decryption architectures, they should be carefully designed, such that low latency architectures with reasonable area efficiency are facilitated. In the following, irreducible polynomials required for operations in $GF(2^m)$ as well as $GF(2^m)[x]/f$ are assumed to match the polynomials given in the Classic McEliece KEM proposal [ABC+20]. For all arithmetic modules a standard basis representation, i.e. a representation as coefficients of a polynomial, is assumed, thus allowing for fast multiplier implementations [DInS09]. Efficient design points for these arithmetic modules will be detailed in the following.

4.1.1 Operations in $GF(2^m)$

Addition in a finite field $GF(2^m)$ with elements represented as polynomials equals the addition of polynomials. $GF(2^m)$ addition is straightforward and performed by bit-wise XOR of field elements, because polynomial addition is achieved by addition of coefficients, which in $GF(2)$ is equivalent to the logical XOR operation.

Multiplication in a finite field can be implemented by using a multitude of approaches. For fast multiplication algorithms, such as Montgomery or Karatsuba-Ofman multiplication, it is assumed that these algorithms do not allow for efficient implementations for the choice of $m = 13$, which is congruent with the findings of Wang et al. [WSN17]. Hence, Mastrovito multiplication is employed in the proposed decryption architectures, as an approach featuring low latency multiplication with moderate area footprint. Low latency operations are thereby achieved by combining the partial product computation with the reduction steps [Mas89]. A finite field multiplier can thereby be designed as a combinatorial function with low latency. Although optimizations for Mastrovito's approach exist (see e.g. [PDCS07b] or [PDCS07a]), improvements for the present case of an irreducible pentanomial are marginal compared to the additional design effort, hence the original approach by Mastrovito is used here.

Squaring in $GF(2^m)$ is implemented using a field multiplier, when an idle field multiplier is available, e.g. in the last step of Patterson's algorithm. However, if this is not the case, squaring using multipliers is relatively costly in terms of area footprint. Exploiting the observation that in binary fields, squaring can be expressed as

$$c = a^2 \bmod f(x) \equiv a_{m-1}^2 x^{2(m-1)} + a_{m-2}^2 x^{2(m-2)} + ... + a_1^2 x^2 + a_0^2 \bmod f(x)\,, \quad (4)$$

less complex implementations are possible [DInS09]. By applying a reduction to the aforementioned squared polynomial, squaring is also implemented by a combinatorial low latency approach similar to the Mastrovito multiplication.

Square root computations for field elements may exploit the fact, that due to Fermat's little theorem

$$a^{2^m} \equiv a \bmod f(x)\,, \quad (5)$$

where $f(x)$ is a binary irreducible polynomial with $\deg(f(x)) = m$. Therefore, $\sqrt{a} = a^{2^{m-1}}$, with $\sqrt{a}^2 = (a^{2^{m-1}})^2 = a^{2^m} \equiv a \bmod f(x)$. The exponentiation required to determine the square root is thereby computed using a square-and-multiply approach. Because square roots only have to be computed in the context of Patterson's algorithm and idle multipliers are always available, the square-and-multiply scheme is mapped onto a field multiplier, similar to the approach of Ghosh and Verbauwhede [GV14].

Inversion is an expensive operation in a finite field $GF(2^m)$. Available inversion approaches, such as application of the EEA, exponentiation or table lookup, differ in the attainable latencies and area costs [WM15]. In order to balance area footprint and decryption latency, two approaches were selected, while considering the respective operation characteristics: For inversion operations that appear in an iterative loop and are therefore repeatedly executed, a very low latency lookup approach was used, in order to avoid a significant impact on the total decryption latency. Inversion operations that are only performed once and require a high throughput instead of low latency processing were implemented using a smaller array of multipliers and squaring modules. By using Fermat's little theorem, as described above, inversion can thereby be evaluated by computing a^{2^m-2} in a square-and-multiply scheme [DPBM00]. As the squared inverses of field elements are required for the computation of a double-sized syndrome, these squared inverses can be directly obtained by computing the power $a^{2^m-3} \equiv a^{-2} \bmod f(x)$ instead of $a^{2^m-2} \equiv a^{-1} \bmod f(x)$.

4.1.2 Polynomial Operations in $GF(2^m)[x]/g$

Addition in a polynomial extension field $GF(2^m)[x]/g$ is achieved by addition of polynomial coefficients, similar to addition in $GF(2^m)$. Since the addition of polynomial coefficients in $GF(2^m)$ corresponds to the bit-wise XOR operation, addition in a binary polynomial extension field is also implemented as a bit-wise XOR operation. Due to the relatively low cost of $m = 13$ XOR gates per coefficient, polynomial adders are assumed to operate completely in parallel with $m(t + 1) = 1560$ XOR gates per adder.

Multiplication in $GF(2^m)[x]/g$ is considerably more complex than multiplication in $GF(2^m)$. It was shown that especially Karatsuba-Ofman multiplication is suited to achieve efficient multiplier architectures in large binary fields [AESI10]. However, polynomial multiplication is only required for two multiplication operations in Patterson's algorithm, where the multiplicants in each case possess a maximum degree of $t + 1/2 = 60$ and during each operation an available $GF(2^m)$ multiplier array is idle. Therefore, in order to avoid overhead and to limit the design effort, polynomial multiplication is implemented in the present work as an interleaved operation with alternating scalar multiplication with partial product accumulation and modular reduction controlled by a finite state machine (FSM).

Scalar multiplication, i.e. the multiplication of a polynomial in $GF(2^m)[x]/g$ with a multiplier element from $GF(2^m)$ is easily achieved by multiplying each coefficient of the

multiplicand, which constitutes an element in $GF(2^m)$, with the multiplier element. This operation is implemented by $t + 1 = 120$ parallel $GF(2^m)$ multiplier modules.

Squaring in a polynomial extension field can be realized by multipliers or by dedicated squaring modules. Since no dedicated polynomial multiplier was employed, squaring has to be implemented differently. As an alternative, Equation 4 can be applied for polynomial squaring in $GF(2^m)[x]/g$. Simplifications are possible, since polynomial squaring is only necessary for polynomials exhibiting a maximum degree of $t + 1/2 = 60$. Therefore, no reduction is required and polynomial squaring is implemented by squaring of individual coefficients as well as a hard-wired coefficient reordering.

Inversion approaches in polynomial extension fields are theoretically equivalent to approaches in the underlying extension field $GF(2^m)$. Nevertheless, different latency-area design points have to be considered for the case of larger polynomial extension fields. Due to exponential scaling behaviour, a lookup-table approach becomes infeasible for $GF(2^m)[x]/g$. Additionally, inversion by exponentiation would require $mt = 1547$ square-and-multiply iterations, thus rendering this approach also unsuitable. Therefore, as a balanced design point, inversion using the EEA was selected for field inversions in $GF(2^m)[x]/g$.

4.2 Decoding Algorithms

4.2.1 Patterson Decoding

The Patterson decoding module, which is depicted in Figure 2, comprises two major modules, namely a *polynomial EEA* module, used for polynomial inversion and decomposition, and a *polynomial square root* module, used for determining polynomial square roots in $GF(2^m)[x]/g$. Both modules execute all field multiplications on a shared array of $t+1 = 120$ $GF(2^m)$ multipliers, which allows for significant area reduction of the associated decoding module.

A Patterson decoding operation starts by reading generator and syndrome polynomial coefficients into shift registers acting as serial-to-parallel converters, thus limiting the required memory bandwidth. Subsequently, the polynomial EEA module is enabled in order to obtain the inverse syndrome $T(x) = S^{-1}$. The following case distinction in the original Patterson algorithm can be neglected for the Niederreiter cryptosystem, because the error-locator polynomial will always have t distinct roots. Therefore, the polynomial square root of $T(x) + x$ is then computed using the polynomial square root module. This square root is decomposed by the polynomial EEA module into polynomials $a(x)$ and $b(x)$. In order to construct the error-locator polynomial as $\lambda(x) = a(x)^2 + x \cdot b(x)^2$, the polynomials $a(x)$ and $b(x)$ are reordered as even and odd parts of the error-locator polynomial, respectively. Thereafter, coefficient-wise squaring of the reordered polynomial using the multiplier array results in the desired error-locator polynomial $\lambda(x)$.

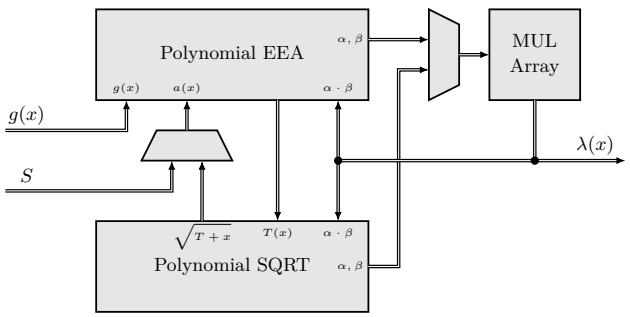


Figure 2: Block diagram of the Patterson decoding module with a shared multiplier array.

4.2.2 Polynomial EEA

The proposed architecture for polynomial EEA computations achieves latency and area advantages over standard EEA approaches by utilizing various optimizations. For instance, the computation of the second Bézout coefficient is omitted and polynomial divisions are combined with the coefficient update operation of scratch polynomials. These measures were adapted from the McEliece software implementation given in [BS08].

The update procedure of scratch polynomials in the EEA usually comprises the computation of a quotient polynomial $q(x)$, followed by a polynomial multiplication of the quotient polynomial with the scratch polynomials of the previous iteration. Since polynomial division and multiplication are costly operations, the EEA architecture presented here combines the division and multiplication steps. Exemplified by the update of the scratch polynomial $u^{(i)}$, the update procedure can thus be formulated as follows:

$$u^{(i+1)} = u^{(i-1)} + \frac{r_{\Delta+j}^{(i-1)}}{r_{\Delta}^{(i)}} u^{(i)} x^j \,, \tag{6}$$

where $\Delta = \deg(r^{(i)})$ and the update step given by the equation above is repeated for all j from $\delta^{(i)} = \deg(r^{(i-1)}) - \deg(r^{(i)})$ to zero. Furthermore, polynomial degree computation is realized as a separate module based on a priority encoding approach.

In addition to the operations described above, the EEA module supports two operational modes, selected by a mode input signal. The first mode corresponds to the computation of an inverse polynomial, while the second mode allows for polynomial decomposition.

4.2.3 Polynomial Square Root

Square root computation by multiplying a polynomial by a suitable matrix, as proposed by Patterson [Pat75], does not reduce the required multiplicative complexity to an adequate level. Therefore, a method described by Huber is used here for efficient square root computation, as it facilitates a balance between square root computation latency and area footprint.

Huber's method computes the square root of a polynomial $z(x)$ as

$$\sqrt{z(x)} \equiv \sqrt{\hat{z}(x)} + \sqrt{x}\sqrt{\tilde{z}(x)} \bmod g(x) \,, \tag{7}$$

where $\hat{z}(x)$ and $\tilde{z}(x)$ represent the even and odd parts of $z(x)$, respectively, such that $z(x) \equiv \hat{z}(x) + x \cdot \tilde{z}(x)$ [Hub96]. Since $\hat{z}(x)$ and $\tilde{z}(x)$ only have even non-zero coefficients, it follows from Equation 4 that square roots of these polynomials are easily obtained as $(\sqrt{\hat{z}})_i = \sqrt{\hat{z}_{2i}}$ and $(\sqrt{\tilde{z}})_i = \sqrt{\tilde{z}_{2i}}$. For the generator polynomial $g(x)$ a similar decomposition into even and odd parts $\hat{g}(x)$ and $\tilde{g}(x)$ can be applied. With these polynomials, the root $\sqrt{x}$ can be determined with respect to the generator $g(x)$ as [Hub96]

$$\sqrt{x} \equiv \sqrt{\hat{g}(x)} \cdot \left(\sqrt{\tilde{g}(x)}\right)^{-1} \bmod g(x) \,. \tag{8}$$

The proposed polynomial square root module computes the roots $\sqrt{\hat{z}(x)}$ and $\sqrt{\tilde{z}(x)}$ by determining the square roots of each coefficient of $z(x) = T(x) + x$ in parallel using the shared multiplier array. The reordering into even and odd parts is realized without additional overhead by appropriate wiring of the multiplier result to the associated registers. Since the operations for the decomposition of $z(x)$ and $g(x)$ are identical, the same resources are used for these operations, where only the root $\sqrt{x}$ has to be stored separately. For the inversion of $\sqrt{\tilde{g}(x)}$ the EEA module described above is utilized in inversion mode.

4.2.4 iBM Decoding

Inversionless Berlekamp-Massey decoding allows for constant time decoding while employing simple finite field operations. The iBM decoding module presented below aims to reduce the decoding latency compared to previous implementations while at the same time maintaining a balanced design point with high area efficiency. Latency reduction without adverse effects on the area footprint is achieved by various novel approaches, which are described below.

The architecture of the proposed iBM decoding module (shown in Figure 3) comprises two sub-modules, associated with the primary steps of the iBM algorithm, *discrepancy computation* and *coefficient update*. In order to avoid a large and thus inefficient decoding module, fully parallelized operation on all $t + 1 = 120$ coefficients of the error-locator polynomial should be avoided. Therefore, the iBM decoding module operates on subsets of 20 coefficients in parallel. This block-wise computation allows for the introduction of two primary measures for latency reduction: *pipelined operation* and *coefficient update bypass*.

Pipelined operation thereby implies that as soon as the first block of coefficients is updated during an iteration, this block is fed into the discrepancy computation module, in order to start discrepancy computation of the next iteration. With this scheme the cycles for a single iBM operation are reduced from $2(t + 1)/20 + 1 = 13$ to $t + 1/20 + 2 = 8$ with a constant number of 2 cycles for the final accumulation step and the update of the first coefficient block. Thus, the total decoding latency is reduced by approximately 38%.

It was shown that for the iBM algorithm the upper $t - k$ coefficients of the error-locator polynomial are 0 in iteration k [SS01]. Therefore, considering architectures with block-wise operations on these coefficients, it is possible to bypass updates of coefficient blocks that only contain zero coefficients. This measure reduces the total amount of cycles for pipelined decoding from $2t \cdot (t + 1/20 + 2) = 1904$ to $t \cdot (t + 1/20 + 2) + \sum_{i=1}^{(t+1)/20} 20 \cdot (i + 2) = 1612$, corresponding to a relative latency reduction of approximately 15%.

4.2.5 Discrepancy Computation

Discrepancy computation constitutes the first step in an iBM iteration. As described before, the discrepancy $\delta^{(k+1)}$ in an iteration k is computed as

$$\delta^{(k+1)} = \sum_{i=0}^{t} \lambda_i^{(k)} \cdot S_{k-i}^{(2)} \,. \tag{9}$$

Due to the relative shift of syndrome against error-locator coefficients in each iteration, implementation of this structure can be performed using a shift register. The proposed discrepancy computation module features a shift register that shifts only once per iteration and selects blocks of coefficients via multiplexers, in order to reduce switching activity.

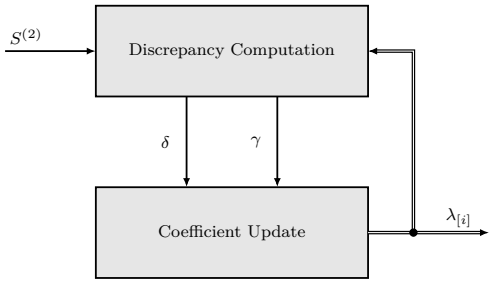


Figure 3: Block diagram of the pipelined iBM decoding module, operating on blocks $\lambda_{[i]}$ of the error-locator polynomial.

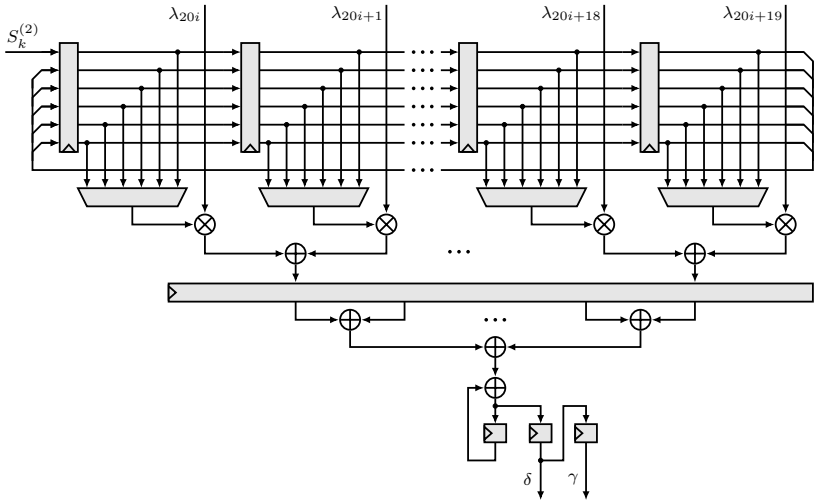


Figure 4: Block diagram of the iBM discrepancy computation module.

This approach, which is shown in Figure 4, furthermore facilitates the coefficient update bypass described before.

It can be observed in Figure 4 that syndrome-error-locator products are pairwise added followed by a register stage. The remaining additions after the register stage are realized as an adder tree with an additional accumulator register. This split, involving an intermediate register stage, ensures a short critical path of the whole module.

4.2.6 Coefficient Update

Following the discrepancy computation, coefficients of the error-locator polynomial are updated in the coefficient update module according to

$$\lambda(x)^{(k+1)} = \gamma^{(k)} \cdot \lambda(x)^{(k)} - \delta^{(k+1)} \cdot b(x)^{(k)} \cdot x\,. \tag{10}$$

Since the formulation of the coefficient update procedure exhibits a regular structure, this procedure is ideally suited to be implemented using a systolic array. Therefore, the coefficient update module relies on such a systolic array, which is depicted in Figure 5. In the proposed systolic array, blocks of error-locator polynomial coefficients remain stationary in an associated processing element (PE), while the auxiliary coefficients b_i are shifted between the PEs, which corresponds to the multiplication by x. Furthermore, the currently updated subset of coefficients is also stored in dedicated registers, which makes these coefficients accessible for the discrepancy computation module. The use of multiplexed registers in this coefficient update module allows for the coefficient update bypass.

4.3 Polynomial Evaluation

Polynomial evaluation in finite fields is an essential operation for the decryption of Niederreiter ciphertexts, as it is necessary for computation of the double-sized syndrome as well as root searching of the error-locator polynomial. Assuming the availability of a low latency decoding module, polynomial evaluation might account for the majority of the resulting decryption latency and area footprint of a Niederreiter decryption architecture, e.g. as seen in [WSN18]. Therefore, careful design of a polynomial evaluation architecture is mandatory for efficient implementation of the whole decryption process.

Even though sophisticated polynomial evaluation approaches, e.g. FFT-based schemes, are available, the large number of intermediate results that have to be stored when

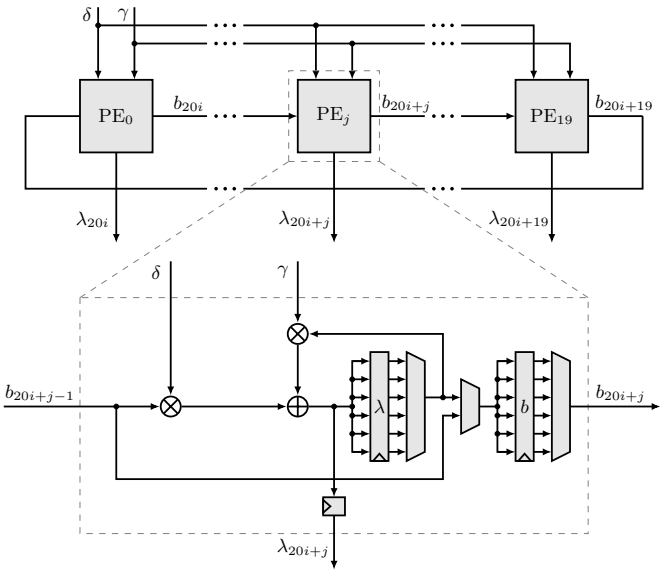


Figure 5: Block diagram of the iBM coefficient update module's systolic array (top) with detail view of a processing element (PE) at the bottom.

using these schemes results in large memories, thus negatively impacting area efficiency. Furthermore, evaluating a polynomial in a specific order necessitates a subsequent root sorting, which deteriorates decryption latency. Therefore, Horner's method is employed in this work for polynomial evaluation. This approach, which will be described in the following, brings the additional advantage that the resulting architecture can be reused for double-sized syndrome computation.

4.3.1 Horner's Method

Horner's rule allows for the computation of the polynomial point $g(L_i)$ as

$$g(L_i) = (((g_t L_i + g_{t-1})\, L_i + g_{t-2})\, ...)\, L_i + g_0\,, \tag{11}$$

which eliminates exponentiations and thus allows for the evaluation of a polynomial using solely finite field multiplication and addition.

The proposed polynomial evaluation module considers *coefficient stationary* processing. Hereby $t+1$ coefficients are stored in t PEs and field elements are fed into a systolic array, while partial sums are transferred between PEs of such an array. Even though a polynomial of degree t possesses $t+1$ coefficients, the coefficient of x^t can be treated as the first partial sum and thus only t PEs are required. A systolic array was derived from this approach, which features reduced fanout and memory bandwidth requirements.

The aforementioned systolic array for polynomial evaluation is suited to directly evaluate the error-locator polynomial and is therefore employed in the Patterson decryption module. For iBM-based decryption, however, in addition to polynomial evaluation, computation of a double-sized syndrome is necessary, which is described in the following.

4.3.2 Double-sized Syndrome

The double-sized syndrome, which is required for correcting t errors using Berlekamp-Massey decoding, can be computed as the product of the zero-padded syndrome S and a double-sized parity check matrix $H^{(2)}$ as $H^{(2)} \times (S|0)$. With the definition of each element

of the double-sized parity check matrix as $H^{(2)}_{i,j} = L^i_j / g^2(L_j)$, with $i \in [0, 2t-1]$ and $j \in [0, n-1]$, several observations facilitate optimizations of the double-sized syndrome computation. First, the computation of the double-sized parity check matrix $H^{(2)}$ can be merged with the following vector-matrix multiplication by multiplying each value of $g^2(L_j)$ by the corresponding syndrome bit before constructing the double-sized parity check matrix and accumulating its columns [WSN18]. Since a field element thereby gets multiplied by a single syndrome bit, this operation is easily realized by the AND operation of each element bit and the syndrome bit. Furthermore, due to zero-padding of the syndrome polynomial, the last columns of $H^{(2)}$ have no influence on the final vector-matrix product and can thus be omitted from computation entirely, where only the first $mt = 1547$ columns have to be computed. Lastly, it can be observed that each row of the double-sized parity check matrix can be obtained from the previous row by element-wise multiplication of a row by the truncated support vector $(L_0, ..., L_{mt-1})$. This allows for the iterative computation of the double-sized parity check matrix by a single multiplication per matrix element.

Using above observations, a systolic array can be designed for double-sized syndrome computation. The double-sized syndrome systolic array operates in a *partial sum stationary* scheme, where generator polynomial values and support vector elements are transferred between PEs. Instead of focusing on a single row or column of the double-sized parity check matrix, the array computes entries of multiple rows and columns in parallel, i.e. the entries $H^{(2)}_{i,0}, H^{(2)}_{i-1,1}, H^{(2)}_{i-2,2}, ..., H^{(2)}_{i-mt,mt}$ are calculated in parallel in iteration i of this scheme. This approach exhibits the advantage of reduced storage and memory bandwidth requirements, with additional improvements for fanout of support vector elements. In the present case, an array consisting of t PEs computes t coefficients of the double-sized syndrome concurrently.

4.3.3 Combined Evaluation Module

Even though the aforementioned optimizations for polynomial evaluation and double-sized syndrome systolic arrays enable a significant circuit size reduction compared to unoptimized designs, these arrays would still occupy a majority of the area of an associated Niederreiter decryption architecture. Hence, it is desirable to further decrease the area footprint of the modules for these operations. When analyzing the systolic arrays for polynomial evaluation and double-sized syndrome computation, it becomes apparent that these arrays resemble each other. Instead of instantiating two distinct arrays for evaluation and syndrome computation, it is therefore advantageous to employ a single combined systolic array, which executes both operations. This *combined evaluation* module[3] will be described in the following in greater detail.

The systolic array of the combined evaluation module employed for iBM-based decryption is illustrated in Figure 6. Apart from the combined systolic array, this module comprises $m = 13$ parallel AND gates for multiplying generator polynomial values by syndrome bits and a constant delay FIFO used to buffer generator polynomial values as well as values of the double-sized parity check matrix. Furthermore, a shift register for parallel-to-serial and serial-to-parallel conversion, a comparator for determining roots of the error-locator polynomial, and a squared inversion module are used in the combined evaluation module.

Double-sized syndrome computation is initiated on the combined evaluation module by sequentially loading coefficients of the generator polynomial $g(x)$ into the shift-register. Subsequently, these coefficients are loaded into the combined systolic array and support vector elements are read from memory to obtain generator polynomial values $g(L_i)$. The

[3]Note, that the combined double-sized syndrome computation and polynomial evaluation approach was developed independently from another implementation, which also combines these two operations [MBR15]. However, the implementation of Massolino et al. employs a different dataflow and does not consider interleaving of syndrome computation and decoding.

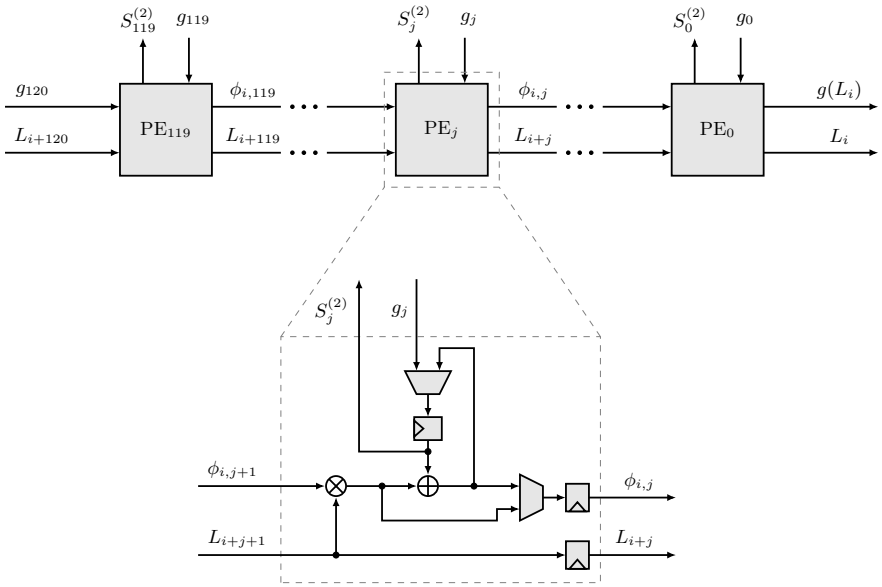


Figure 6: Block diagram of the combined systolic array (top) with detail view of the jth PE (bottom).

squared inverses of these values are then stored in the constant delay FIFO. Afterwards, these polynomial values are multiplied by the corresponding syndrome bits and are fed back into the systolic array[4], in order to compute the first half of a double-sized syndrome, which is loaded into the shift register, hence allowing to immediately resume double-sized syndrome computation. The product of support element powers and the squared inverses of generator polynomial values that exit the systolic array in this first iteration are required to resume computation of the double-sized syndrome and are thus stored in the constant delay FIFO. After completing the computation of the first half of the double-sized syndrome, the respective products and support vector elements are read from memory and from the FIFO to obtain the second half of the double-sized syndrome. Using this sequence, the complete double-sized syndrome computation is completed within

$$N_{\mathrm{cyc},t,S^{(2)}} = t + m - 1 + mt + 1 + t + mt + 1 + t + mt = 5012 \tag{12}$$

cycles, where t or $t + m - 1$ cycles are required to fill the array, mt cycles are required to iterate over support vector elements and single cycles are required for multiplying the generator values by syndrome bits and to store the first half of the double-sized syndrome.

For root search of the error-locator polynomial the sequence of operations is considerably simpler: After block-wise loading of error-locator coefficients into the combined systolic array (not shown in Figure 6), the error-locator polynomial is evaluated for all support vector elements and the downstream comparator module converts polynomial values to bits of the error vector, where a root of this polynomial correspond to a 1 in the error vector.

[4]Note, that the first PE of the combined systolic array is slightly modified, since it is possible to omit the multiplication by a support element for the first row of $H^{(2)}$.

5 Implementation

All modules described in Section 4 were implemented in Verilog and SystemVerilog, respectively. Two ASIC designs, corresponding to the iBM- and Patterson-based decryption modules, were developed using a digital design flow relying on Cadence Genus and Innovus software systems for the 22nm FDSOI CMOS technology node from GlobalFoundries (GF).

The combined evaluation module's FIFO in the iBM-based design was implemented using a memory macro from the GF 22nm FDSOI memory portfolio. While the iBM- and Patterson-based decryption ASIC architectures operate at a clock frequency of 1 GHz, higher clock frequencies are achievable, since the clock frequency is limited by the delay of memory macros and not by the decryption logic itself. Therefore, the layout design adopts multicycle paths across the key memory, which allows for clock frequencies of up to 2 GHz. The aforementioned ASIC designs were verified using testbench driven simulations as well as formal verification methods.

6 Evaluation

In order to evaluate the designed Niederreiter decryption ASIC architectures and to assess their suitability for efficient low latency decryption, these architectures should be compared to previous state-of-the-art approaches. However, no previous ASIC designs are available for comparison. The FPGA design introduced by Wang et al. is the only available Niederreiter decryption architecture that supports the parameter set selected in our research. The authors of that architecture state that the majority of their FPGA design can be re-used for the development of an ASIC design [WSN17]. Therefore, such an ASIC design was created from the given Verilog source using the given speed-optimized design point for comparison purposes, where only the block RAM modules, instantiated for polynomial evaluation, were exchanged with appropriate memory macros. The resulting ASIC design will subsequently be referred to as the *reference design*.

6.1 Decoding Algorithms

Results for the proposed iBM and Patterson decoding modules as well as the BM decoding module from the reference design are shown in Table 3. When juxtaposing the iBM module and the reference Berlekamp-Massey module, it can be seen, that while the proposed iBM decoding module requires significantly fewer arithmetic modules and registers, it still allows for a decoding speedup of approximately 91%. This can be explained by the systolic array approach of the iBM module that specifically considers operations on coefficient blocks and introduces various optimizations, such as the pipelined discrepancy and coefficient update computations. Comparison of these modules to the proposed Patterson decoding module shows a significantly increased utilization of arithmetic modules and registers as well as an increased cycle count in the latter, which results from the complex and specialized operations of Patterson decoding. However, Patterson decoding eliminates the need for double-sized syndrome computation. Therefore, when complete decryption architectures are compared below, the apparent disadvantage of Patterson decoding is put into perspective.

Table 3: Arithmetic module and cycle counts for different decoding designs.

Design	Adders	Multipliers	Inversions	Registers	Cycles
iBM	40	60	0	5109	1619
Patterson	241	121	1	17966	5739
Reference	240	80	1	13079	3095

6.2 Error-Locator Polynomial Evaluation

Both of the two proposed iBM-based and Patterson-based ASIC architectures employ systolic arrays derived from Horner's method for polynomial evaluation, while the reference design uses an additive FFT approach, which allows for a reduction of the multiplicative complexity [GM10]. With the latency of Horner's method for polynomial evaluation depending on the number of evaluated elements the scenario of error-locator polynomial evaluation for $n = 6960$ elements is assumed here for comparison.

Arithmetic module as well as cycle counts for both approaches are shown in Table 4. The FFT approach of the reference design is thereby advantageous both in the number of required arithmetic modules and the resulting cycle count for polynomial evaluation. However, using this approach, a large number of intermediate results have to be accessed in parallel, which requires large memory macros and a high register count, thus reducing the area efficiency of the FFT polynomial evaluation module. Furthermore, since FFT-based polynomial evaluation requires a specific evaluation order, evaluation has to be followed by a root sorting operation, which ultimately eliminates the latency advantage of the polynomial evaluation in the reference design.

Table 4: Module and cycle counts for error-locator polynomial root search approaches.

Design	Adders	Multipliers	Registers	Memory Size	Cycles[a]	
					Eval.	Sort
Horner	119	119	6240	-	7086	-
FFT	128	40	20069	$2 \cdot (768 \times 70)$	1082	6972

[a] Eval. = evaluate error-locator polynomial, Sort = sort roots

6.3 Double-sized Syndrome Computation

Double-sized syndrome computation in the iBM-based decryption architecture is executed on the combined evaluation module, while in the reference design this operation is performed by the FFT polynomial evaluation module and a dedicated double-sized syndrome module. For the reference design this double-sized syndrome module comprises 40 multipliers and a fast field inversion module. The proposed combined evaluation module, on the other hand, computes squared inverses using a square-and-multiply approach.

Results of both approaches are shown in Table 5. Syndrome computation in the reference design requires fewer multipliers than in the proposed iBM architecture, resulting from the use of FFT-based polynomial evaluation, which leads to an increased area requirement from the associated memory macros. While the polynomial evaluation portion exhibits a longer latency in the proposed iBM architecture, the actual syndrome construction is faster than in the reference design, due to the use of the combined evaluation and syndrome computation approach using a higher number of field multipliers.

Table 5: Module and cycle counts for double-sized syndrome computation approaches.

Design	Adders	Multipliers / Squarers	Inversions	Registers	Memory Size	Cycles[a]	
						Eval.	Synd.
iBM	119	130 / 12	0	6552	1568×13	1800	3338
Reference	170	80 / 1	1	25757	$2 \cdot (768 \times 70)$	1082	4658

[a] Eval. = evaluate generator polynomial, Synd. = compute double-sized syndrome

6.4 Design Space Comparison

In addition to the assessment of decryption architectures on a module-wise basis, as described above, these architectures should also be evaluated in their entirety, in order to consider dataflow dependencies and interactions across module boundaries. Therefore, the two proposed ASIC designs are compared to the reference design with respect to the key performance indicators decryption latency, area footprint and power dissipation, which allow to determine different points in the design space. All designs are compared at a clock frequency of $f_{\text{clk}} = 1$ GHz.

Results of the aforementioned designs after PNR are summarized in Table 6, where additional area efficiency figures are given as the reciprocal of the product of decryption latency and area requirements. Table 6 also includes results of a 2 GHz iBM design, as an example of a decryption design operating at a higher clock frequency. The aforementioned results thereby prove the effectiveness of the introduced optimization measures for the design of efficient Niederreiter decryption architectures.

In terms of latency, the proposed Patterson-based decryption architecture achieves the lowest latency of the compared designs, due to the elimination of the double-sized syndrome computation step. The iBM-based decryption architecture exhibits only a slightly longer decryption latency, which follows from the fast decoding module in conjunction with optimized polynomial evaluation and double-sized syndrome computation. While the reference design features the fastest polynomial evaluation of all compared designs, the longer latency of remaining decryption steps in this architecture outweigh this advantage. Therefore, it follows that the total decryption latency for the reference design is approximately 28% and 32% higher than the latencies of the proposed iBM-based and Patterson-based decryption architectures, respectively.

The differences of the compared approaches also manifests in the attainable area of the associated implementations. By combining multiple decryption steps into the same module and balancing the number of parallel units, the iBM-based architecture achieves a very low area requirements, which positively impacts the area efficiency. The proposed Patterson-based design results in a significantly larger footprint, due to the complex operations of Patterson decoding. Even though the memory-intensive FFT polynomial evaluation approach of the reference design might prove advantageous for FPGA implementations, in the derived ASIC architecture, this approach leads to a considerable larger area footprint compared to the proposed designs, with a 297% and 145% area increase relative to the iBM and Patterson decryption designs. The impact of large memory macros in the reference design furthermore becomes apparent for power dissipation figures, were the iBM- and Patterson-based architectures achieve approximately four times lower power dissipation compared to the reference design.

It should also be mentioned, that while the iBM decryption architecture as well as the reference design only employ constant-time operations, the EEA implementation of the Patterson decryption architecture may result in input dependent latency variations.

Table 6: Summary of results of the compared ASIC designs after PNR[a].

Design	f_{clk}	Latency		Area	Power	Energy/Op.[b]	AT-Efficiency
	[GHz]	[Cycles]	[µs]	[mm^2]	[mW]	[nJ]	[(µs · mm^2)$^{-1}$]
iBM	2	13271	6.64	0.0750	107.91	716.0	2.0080
iBM	1	13185	13.19	0.0744	56.80	748.9	1.0190
Patterson	1	12825	12.83	0.1205	60.71	778.6	0.6468
Reference	1	16889	16.89	0.2955	248.11	4190.0	0.2004

[a] All designs were implemented using the GF 22nm FDSOI CMOS node.

[b] Energy per decryption operation.

These variations render the aforementioned architecture susceptible to timing side-channel attacks. The iBM-based design and the reference design, on the other hand, are not vulnerable to timing side-channel attacks.

7 Conclusion

The presented research aims to facilitate low latency decryption for the Niederreiter cryptosystem with high area efficiency. This objective is achieved by the design, implementation as well as the optimization of two ASIC architectures for Niederreiter decryption, which target the GF 22nm FDSOI CMOS technology node. Furthermore, optimizations considering the memory bottleneck allow to place-and-route the inversionless Berlekamp-Massey (iBM) decryption architecture at 2 GHz.

The proposed iBM-based and Patterson-based decryption architectures enable an unprecedented decryption latency, area footprint and power dissipation. Compared to previous solutions, the improved performance in this work is achieved due to the proposed novel dataflow optimizations, especially for inversionless decoding, which allows for a 27% speedup in terms of cycle count compared to previous state-of-the-art approaches. At the same time, the occupied area of the iBM ASIC design is reduced to approximately 25% of the area of previous approaches, through optimization techniques, such as a combined systolic array for double-sized syndrome computation and polynomial evaluation. Therefore, the aforementioned design exhibits an area efficiency that is over an order of magnitude higher than the efficiency of prior approaches. By selecting a large parameter set, the implemented designs were shown to support decryption of long-term secure Niederreiter ciphertexts. With the constant-time operations of the iBM-based decryption design, this architecture is efficiently hardened against timing side-channel attacks. Hence, it can be assumed that the proposed iBM design is ideally suited for applications in high security and high performance environments.

References

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second round of the nist post-quantum cryptography standardization process, 2020.

[ABB+15] Daniel Augot, Lejla Batina, Daniel J. Bernstein, Joppe Bos, Johannes Buchman, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsin, Tanja Lange, Mohamed S. E. Mohamed, Christian Rechberger, Peter Schwab, Nicolas Sendrier, Frederik Vercauteren, and Bo-Yin Yang. Initial recommendations of long-term secure post-quantum systems, 2015.

[ABC+20] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic mceliece: conservative code-based cryptography. In *NIST Post-Quantum Cryptography Standardization Round 3 Submission*, 2020.

[AESI10] Sameh Abdulah, Ashraf El-Sisi, and Nabil Ismail. Hardware implementation of efficient modified karatsuba multiplier used in elliptic curves. *International Journal of Network Security*, Vol.11:pp.155–162, 11 2010.

[Ber73] E. Berlekamp. Goppa codes. *IEEE Transactions on Information Theory*, 19(5):590–592, 1973.

[Ber15] Elwyn R. Berlekamp. *Algebraic Coding Theory - Revised Edition.* World Scientific Publishing Co., Inc., 2015.

[BLP08] Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the mceliece cryptosystem. In *Post-Quantum Cryptography*, pages 31–46, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BS08] Bhaskar Biswas and Nicolas Sendrier. Mceliece cryptosystem implementation: Theory and practice. In *Post-Quantum Cryptography*, pages 47–62, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BSNK19] Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri. Nist post-quantum cryptography- a hardware evaluation study. Cryptology ePrint Archive, Report 2019/047, 2019. https://ia.cr/2019/047.

[Bur71] H. Burton. Inversionless decoding of binary bch codes. *IEEE Transactions on Information Theory*, 17(4):464–466, 1971.

[CCKA21] Alvaro Cintas Canto, Mehran Mozaffari Kermani, and Reza Azarderakhsh. Reliable architectures for composite-field-oriented constructions of mceliece post-quantum cryptography on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(5):999–1003, 2021.

[DInS09] Jean-Pierre Deschamps, Jose Luis Imaña, and Gustavo D. Sutter. *Hardware Implementation of Finite-Field Arithmetic.* McGraw-Hill, 2009.

[DPBM00] A.V. Dinh, R.J. Palmer, R.J. Bolton, and R. Mason. A low latency architecture for computing multiplicative inverses and divisions in $gf(2^m)$. In *2000 Canadian Conference on Electrical and Computer Engineering. Conference Proceedings. Navigating to a New Era (Cat. No.00TH8492)*, volume 1, pages 43–47 vol.1, 2000.

[EOS06] D. Engelbert, R. Overbeck, and A. Schmidt. A summary of mceliece-type cryptosystems and their security. Cryptology ePrint Archive, Report 2006/162, 2006. https://ia.cr/2006/162.

[GM10] Shuhong Gao and Todd Mateer. Additive fast fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010.

[GV14] Santosh Ghosh and Ingrid Verbauwhede. Blake-512-based 128-bit cca2 secure timing attack resistant mceliece cryptoprocessor. *IEEE Transactions on Computers*, 63:1–1, 05 2014.

[HCG17] Jingwei Hu, Ray C. C. Cheung, and Tim Güneysu. Compact constant weight coding engines for the code-based cryptography. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(9):1092–1096, 2017.

[HDYC18] Jingwei Hu, Wangchen Dai, Liu Yao, and Ray C.C Cheung. An application specific instruction set processor (asip) for the niederreiter cryptosystem. In *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*, pages 1–6, 2018.

[Hey10] Stefan Heyse. Low-reiter: Niederreiter encryption scheme for embedded microcontrollers. In *Post-Quantum Cryptography*, pages 165–181, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[Hey13] Stefan Heyse. *Post Quantum Cryptography: Implementing Alternative Public Key Schemes On Embedded Devices - Preparing for the Rise of Quantum Computers*. PhD thesis, Ruhr-University Bochum, 2013.

[HG12] Stefan Heyse and Tim Güneysu. Towards one cycle per bit asymmetric encryption: Code-based cryptography on reconfigurable hardware. In *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 340–355, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[HG13] Stefan Heyse and Tim Güneysu. Code-based cryptography on reconfigurable hardware: tweaking niederreiter encryption for performance. *Journal of Cryptographic Engineering*, 3(1):29–43, 2013.

[HP03] W. Cary Huffman and Vera Pless. Fundamentals of error-correcting codes. 06 2003.

[Hub96] K. Huber. Note on decoding binary goppa codes. *Electronics Letters*, 32:102–103, 1996.

[LC87] Mansour Loeloeian and Jean Conan. A transform approach to goppa codes. *IEEE Trans. Inf. Theor.*, 33(1):105–115, January 1987.

[LDW94] Yuan Xing Li, R.H. Deng, and Xin Mei Wang. On the equivalence of mceliece's and niederreiter's public-key cryptosystems. *IEEE Transactions on Information Theory*, 40(1):271–273, 1994.

[Mas89] Edoardo D. Mastrovito. Vlsi designs for multiplication over finite fields $gf(2^m)$. In Teo Mora, editor, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[MBR15] Pedro Maat C. Massolino, Paulo S. L. M. Barreto, and Wilson V. Ruggiero. Optimized and scalable co-processor for mceliece with binary goppa codes. *ACM Trans. Embed. Comput. Syst.*, 14(3), 2015.

[McE78] R. McEliece. A public key cryptosystem based on algebraic coding theory, 1978.

[McE02] R. McEliece. *The Theory of Information and Coding*. Cambridge University Press, 2002.

[Nie86] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. In *Problems of Control and Information Theory*, 1986.

[Pat75] N. Patterson. The algebraic decoding of goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[PDCS07a] Nicola Petra, Davide De Caro, and Antonio G.M. Strollo. High speed galois fields $gf(2^m)$ multipliers. In *2007 18th European Conference on Circuit Theory and Design*, pages 468–471, 2007.

[PDCS07b] Nicola Petra, Davide De Caro, and Antonio G.M. Strollo. A novel architecture for galois fields $gf(2^m)$ multipliers based on mastrovito scheme. *IEEE Transactions on Computers*, 56(11):1470–1483, 2007.

[SS92] V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. 2(4):439–444, 1992.

[SS01] D.V. Sarwate and N.R. Shanbhag. High-speed architectures for reed-solomon decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(5):641–655, 2001.

[SWM+10] Abdulhadi Shoufan, Thorsten Wink, H. Gregor Molter, Sorin A. Huss, and Eike Kohnert. A novel cryptoprocessor architecture for the mceliece public-key cryptosystem. *IEEE Transactions on Computers*, 59(11):1533–1546, 2010.

[WM15] R. W. Ward and T. C. A. Molteno. Efficient hardware calculation of inverses in $gf(2^8)$. 2015.

[WSN17] Wen Wang, Jakub Szefer, and Ruben Niederhagen. Fpga-based key generator for the niederreiter cryptosystem using binary goppa codes. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 253–274. Springer International Publishing, 2017.

[WSN18] Wen Wang, Jakub Szefer, and Ruben Niederhagen. Fpga-based niederreiter cryptosystem using binary goppa codes. In *Post-Quantum Cryptography*, pages 77–98. Springer International Publishing, 2018.