Efficient Hardware Implementation of the Stream Cipher WG-16 with Composite Field Arithmetic

Xinxin Fan, Nusa Zidaric, Mark Aagaard, and Guang Gong
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, N2L 3G1, CANADA
{x5fan,nzidaric,maagaard,ggong}@uwaterloo.ca

ABSTRACT

The Welch-Gong (WG) stream cipher family was designed based on the WG transformation and is able to generate keystreams with mathematically proven randomness properties such as long period, balance, ideal tuple distribution, ideal two-level autocorrelation and high and exact linear complexity. In this paper, we present a compact hardware architecture and its pipelined implementation of the stream cipher WG-16, an efficient instance of the WG stream cipher family, using composite field arithmetic and a newly proposed property of the trace function in tower field representation. Instead of using the original binary field $\mathbb{F}_{2^{16}}$, we demonstrate that its isomorphic tower field $\mathbb{F}_{(((2^2)^2)^2)^2}$ can lead to a more efficient hardware implementation. Efficient conversion matrices connecting the binary field $\mathbb{F}_{2^{16}}$ and the tower field $\mathbb{F}_{(((2^2)^2)^2)^2}$ are also derived. Our implementation results show that the pipelined WG-16 hardware core can achieve the throughput of 124 MHz at the cost of 478 slices in an FPGA and 552 MHz at the cost of $12,031$ GEs in a 65 nm ASIC, respectively.

Keywords: Stream cipher, WG-16 transformation, tower field, pipeline design, hardware implementations.

1. INTRODUCTION

With the advent of ubiquitous computing, communication security has moved more and more to the forefront of attention. Security is mandatory to ensure that the communication system is properly functioning and to prevent misuse. Stream ciphers are fast cryptographic primitives that provide confidentiality of electronically transmitted data. When compared to other cryptographic primitives, stream ciphers are competitive in software applications with exceptionally high throughput, and in hardware applications with exceptionally small footprint. Their major applications, though by no means restricted to, are 4G telecommunication systems [21, 22], IEEE 802.11 wireless networks [9], Bluetooth [1], digital video broadcasting systems like pay-TV and RFID tags [19, 23].

The WG ciphers [17] refer to a family of synchronous and hardware oriented stream ciphers built from Linear Feedback Shift Registers (LFSRs) and Boolean functions with compact Algebraic Normal Forms (ANFs), which can be regarded as nonlinear filter generators over an extension field. One instance of the WG stream cipher family called WG-29 [16] was submitted to the ECRYPT Stream Cipher (eSTREAM) project [5] and entered the second phase in 2005. Among more than 20 submissions, the WG-29 is the only candidate that has mathematically proven randomness properties such as ideal two-level autocorrelation, balance, long period, ideal tuple distribution, and exact linear complexity [3]. Those randomness properties are paramount for protecting communication systems from cryptanalysis by hackers. Moreover, the ideal two-level correlation sequences are very effective to combat channel noise since the autocorrelation will reach the maximum value after one period, thereby facilitating the synchronization between a transmitter and a receiver. Besides the stream cipher WG-29, other instances of the WG stream cipher family have been proposed to secure RFID systems [13], resource-constrained smart devices [7] and 4G-LTE networks [6].

Thanks to the attractive randomness properties of the WG stream cipher family, its hardware implementations have also attracted a lot of attention. Nawaz and Gong [16] described a hardware architecture for implementing WG-29 using normal basis, which requires seven multipliers and one inversion over $\mathbb{F}_{2^{29}}$. Their hardware design has been further improved in [17] by eliminating one multiplier through signal reuse and replacing the inversion with an exponentiation. In [11], Krengel proposed an interleaved approach with precomputation which can achieve an 8-fold speed-up at the cost of 2^{29} bits of ROM in hardware. Lam *et al.* [12] presented the hardware design of the MOWG, a multi-bit output variant of the original WG cipher. The authors optimized the proposed hardware architecture through extensive signal reuse as well as pipelining with reuse techniques. Recently, El-Razouk *et al.* [4] proposed a novel hardware design for WG-29 that is based on the efficient computation of the trace of a product of two finite field elements with type-II optimal normal basis (ONB) representations. Their ASIC implementations can achieve an improvement of 40% in area, 39% in dynamic power consumption and 17% in speed, when compared to previous results in the literature.

Motivated by El-Razouk *et al.*'s work in [4], we address compact hardware implementation of WG-16 [6], an efficient instance of the WG stream cipher family, in this paper. Due to the lack of Gaussian normal bases [15] over $\mathbb{F}_{2^{16}}$, the method for computing the trace of a product of two field elements proposed in [4] cannot be directly applied to WG-16. However, we demonstrate that using the isomorphic tower field $\mathbb{F}_{(((2^2)^2)^2)^2}$ of $\mathbb{F}_{2^{16}}$ as well as normal bases for all towerings the nice property of the trace function in [4] can be recovered. By combining efficient tower field arithmetic and low-cost basis conversion matrices, we propose a compact hardware architecture as well as its pipelined design for WG-16. Our implementations on FPGA (i.e., Spartan-6) and ASIC (i.e., 65nm CMOS technology) show that the pipelined WG-16 hardware core can run at the maximum frequency of 124 MHz and 578 MHz, at the cost 478 slices and $12,031$ gate equivalents (GEs) as well as 138 mW and 25.5 mW of dynamic power consumption on the target platform, respectively.

The rest of this paper is organized as follows. Section 2 gives some notations and a brief description of the stream cipher WG-16. In Section 3, we describe efficient tower field arithmetic in $\mathbb{F}_{(((2^2)^2)^2)^2}$ and derive low-cost basis conversion matrices. Section 4 proposes a compact hardware architecture for WG-16 based on a newly proposed property of the trace function. In Section 5, we describe a pipelined design for the proposed compact hardware architecture of WG-16. The F-PGA and ASIC implementations of the pipelined design are discussed and compared to previous work in Section 6. Finally, Section 7 concludes this contribution.

2. PRELIMINARIES

This section defines some notations that will be used to describe the stream cipher WG-16 and its hardware architecture throughout this paper.

- $\mathbb{F}_2 = \{0, 1\}$, the Galois field with two elements 0 and 1.
- $p(X) = X^{16} + X^5 + X^3 + X^2 + 1$, a primitive polynomial of degree 16 over $\mathbb{F}_2$.
- $\mathbb{F}_{2^{16}}$, the extension field of $\mathbb{F}_2$ defined by the primitive polynomial $p(X)$ with 2^{16} elements. Let ω be a primitive element of $\mathbb{F}_{2^{16}}$ such that $p(\omega) = 0$.
- $\mathbb{F}_{(((2^2)^2)^2)^2}$, the isomorphic tower construction of $\mathbb{F}_{2^{16}}$.
- $\mathrm{Tr}_{\mathbb{F}_{2^m}}^{\mathbb{F}_{2^{mn}}}(X) = X + X^2 + X^{2^2} + \cdots + X^{(2^m)^{n-1}}$, the relative trace function from $\mathbb{F}_{2^{mn}} \mapsto \mathbb{F}_{2^m}$. If m is equal to 1 then $\mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^n}}(\cdot)$ is just called the trace and denoted by $\mathrm{Tr}(\cdot)$.
- $l(X) = X^{32} + X^{25} + X^{16} + X^7 + \omega^{2743}$, a primitive polynomial of degree 32 over $\mathbb{F}_{2^{16}}$ which is used as the feedback polynomial of LFSR.
- $q(X) = X + X^{2^{11}+1} + X^{2^{11}+2^6+1} + X^{2^6-2^{11}+1} + X^{2^{11}+2^6-1}$, a permutation polynomial over $\mathbb{F}_{2^{16}}$.
- WGP-16$(X^d) = q(X^d + 1) + 1$, the WG-16 permutation with decimation d from $\mathbb{F}_{2^{16}} \mapsto \mathbb{F}_{2^{16}}$, where $d = 1057$ is coprime to $2^{16} - 1$.
- WGT-16$(X^d) = \mathrm{Tr}(\text{WGP-16}(X^d))$, the WG-16 transformation with decimation d from $\mathbb{F}_{2^{16}} \to \mathbb{F}_2$, where $d = 1057$ is coprime to $2^{16} - 1$.
- Normal basis (NB) of $\mathbb{F}_{2^{16}}$: A normal basis of $\mathbb{F}_{2^{16}}$ over $\mathbb{F}_2$ is a basis of the form $\{\theta, \theta^2, \cdots, \theta^{2^{15}}\}$, where $\theta = \omega^{1091}$ (i.e., a normal element) is used in this work.
- $\oplus_m$, the bitwise XOR operator for two operands of m bits.
- $\odot_m$, the bitwise AND operator for two operands of m bits.
- $\otimes_m$, the multiplication operator for two operands over $\mathbb{F}_{2^m}$ or its isomorphic fields.

2.1 Overview of the Stream Cipher WG-16

The stream cipher WG-16 [6] is a hardware-oriented keystream generator that consists of three main components as illustrated in Figure 1:

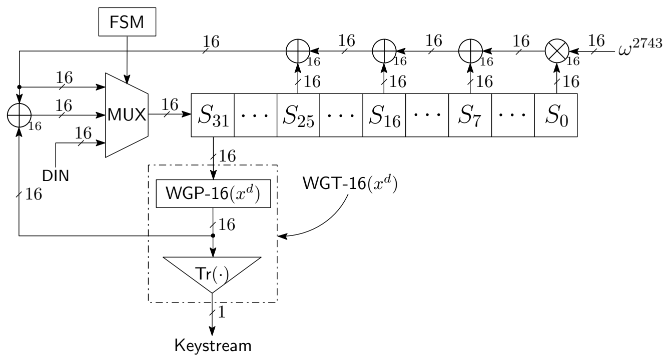


Figure 1: The High-Level Hardware Architecture of the Stream Cipher WG-16

I. A 32-stage LFSR over finite field $\mathbb{F}_{2^{16}}$ with the internal states S_k ($k = 0, \ldots, 31$) and the feedback polynomial $l(X)$.

II. A WG-16 transformation module WGT-16(S_{k+31}^d), which takes as an input the state S_{k+31} ($k \geq 0$) of the LFSR and outputs one bit keystream. For accommodating the initialization phase, WGT-16(S_{k+31}^d) is further split into two sub-modules:

- A WG-16 permutation module WGP-16(S_{k+31}^d), which takes as an input the state S_{k+31} ($k \geq 0$) of the LFSR and outputs a 16-bit intermediate value.
- A trace computation module $\mathrm{Tr}(\cdot)$ that compresses the 16-bit intermediate value from WGP-16(S_{k+31}^d) to one bit keystream.

III. A finite state machine (FSM) that controls the operation of the WG-16 cipher.

The stream cipher WG-16 operates in three phases, namely key/IV loading phase, initialization phase, and running phase, under the control of the FSM. During the key/IV loading phase, a 128-bit key and a 128-bit initialization vector (IV) will be first loaded into the LFSR within 32 clock cycles through the pin DIN. After loading the required key and IV, the initialization phase will be performed in the next 64 steps without any output. In the initialization phase, the input to the LFSR is the bitwise XOR of the linear feedback from the LFSR and the 16-bit intermediate value (i.e., a nonlinear feedback) from the WG-16 permutation module. The running phase starts from the 97-th step and one bit keystream will be generated in each clock cycle. In the running phase, the only input to the LFSR is the linear feedback within the LFSR. The recurrence relations for updating the

LFSR in the initialization and running phases are summarized below:

$$S_{k+32} = \begin{cases} (\omega^{2743} \otimes_{16} S_k) \oplus_{16} S_{k+7} \oplus_{16} S_{k+16} \oplus_{16} \\ S_{k+25} \oplus_{16} \mathsf{WGP\text{-}16}(S_{k+31}^d), \quad 0 \le k < 64 \\ (\omega^{2743} \otimes_{16} S_k) \oplus_{16} S_{k+7} \oplus_{16} S_{k+16} \oplus_{16} \\ S_{k+25}, \quad k \ge 64 \end{cases}.$$

3. TOWER CONSTRUCTIONS OF $\mathbb{F}_{2^{16}}$ AND BASIS CONVERSION MATRICES

The hardware implementations of the stream cipher WG-16 involve finite field arithmetic (i.e., addition, multiplication, exponentiation and inversion) over $\mathbb{F}_{2^{16}}$. Since several exponentiations of the form X^{2^ι} ($\iota > 1$) are computed during the evaluation of the WG-16 transformation, it is natural to utilize a normal basis of $\mathbb{F}_{2^{16}}$ over $\mathbb{F}_2$ for efficient implementation. Although we can directly implement multiplications and inversions over $\mathbb{F}_{2^{16}}$ using the normal basis, the hardware and time complexities of the resulting implementation are high due to the lack of Gaussian normal bases [15] over $\mathbb{F}_{2^{16}}$. To address the aforementioned issues, we employ the isomorphic tower construction of $\mathbb{F}_{2^{16}}$ to achieve better performance. In order to describe our choice for polynomial and normal bases at each level of tower unambiguously, we use the field representations given in Table 1 as a reference.

Table 1: Finite Field Representations Used for Unique Reference

Finite Field	Defining Polynomial
$\mathbb{F}_{2^{16}} \cong \mathbb{F}_2[\omega]$	$X^{16}+X^5+X^3+X^2+1$
$\mathbb{F}_{2^8} \cong \mathbb{F}_2[z]$	$X^8+X^4+X^3+X^2+1$
$\mathbb{F}_{2^4} \cong \mathbb{F}_2[y]$	X^4+X+1
$\mathbb{F}_{2^2} \cong \mathbb{F}_2[x]$	X^2+X+1

For obtaining the tower construction of $\mathbb{F}_{2^{16}}$, we first construct $\mathbb{F}_{2^2}$ by using the irreducible polynomial $e(X)$ over $\mathbb{F}_2$, then construct $\mathbb{F}_{(2^2)^2}$ by using a certain irreducible polynomial $f(X)$ of degree 2 over $\mathbb{F}_{2^2}$, and then construct $\mathbb{F}_{((2^2)^2)^2}$ by using a certain irreducible polynomial $g(X)$ of degree 2 over $\mathbb{F}_{(2^2)^2}$. Finally, we construct $\mathbb{F}_{(((2^2)^2)^2)^2}$ by using a certain irreducible polynomial $h(X)$ of degree 2 over $\mathbb{F}_{((2^2)^2)^2}$. Note that all individual field extensions have degree two, as illustrated below:

$$\mathbb{F}_2 \xrightarrow{e(X)} \mathbb{F}_{2^2} \xrightarrow{f(X)} \mathbb{F}_{(2^2)^2} \xrightarrow{g(X)} \mathbb{F}_{((2^2)^2)^2} \xrightarrow{h(X)} \mathbb{F}_{(((2^2)^2)^2)^2}.$$

3.1 Tower Construction with Normal Bases

Since the efficiency of the arithmetic over $\mathbb{F}_{(((2^2)^2)^2)^2}$ is closely related to the selection of the irreducible polynomials as well as the bases for the towerings, we consider using normal bases for all towerings here, as illustrated in Table 2.

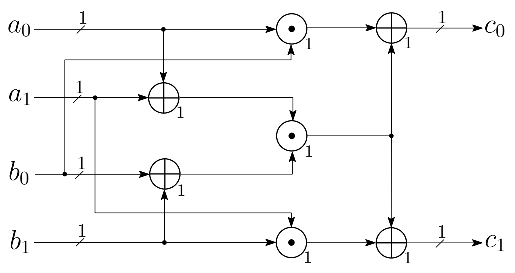
3.1.1 Arithmetic operations in $\mathbb{F}_{2^2}$.

Let $A = a_0\alpha + a_1\alpha^2$ and $B = b_0\alpha + b_1\alpha^2$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_2$. A multiplication $C = AB$ is computed as follows (see Figure 2(a)):

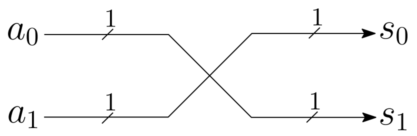
$$\begin{aligned} AB &= (a_0\alpha + a_1\alpha^2)(b_0\alpha + b_1\alpha^2) \\ &= a_0b_0\alpha^2 + (a_0b_1 + a_1b_0)(\alpha+\alpha^2) + a_1b_1\alpha \\ &= [(a_0+a_1)(b_0+b_1) + a_0b_0]\alpha + \\ &\quad [(a_0+a_1)(b_0+b_1) + a_1b_1]\alpha^2 = c_0\alpha + c_1\alpha^2 = C. \end{aligned}$$

For a non-zero element $A \in \mathbb{F}_{2^2}$, the square (i.e., the Frobenius mapping with respect to $\mathbb{F}_2$) of A is calculated as follows (see Figure 2(b)):

$$\begin{aligned} A^2 &= A^{-1} = (a_0\alpha + a_1\alpha^2)^2 = a_0\alpha^2 + a_1\alpha^4 \\ &= a_1\alpha + a_0\alpha^2 = s_0\alpha + s_1\alpha^2 = S. \end{aligned}$$



(a) Multiplication Unit M_2

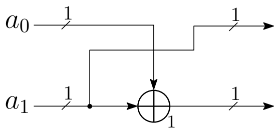


(b) Squaring (Inversion) Unit S_2 (I_2)

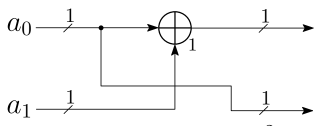
Figure 2: Multiplication, Squaring and Inversion in $\mathbb{F}_{2^2}$ with Normal Bases

Note that the inverse of $A \in \mathbb{F}_{2^2}$ is equivalent to the square. Moreover, the multiplications of $A \in \mathbb{F}_{2^2}$ by α and α^2 are carried out as follows (see Figure 3):

$$\begin{aligned} \alpha A &= a_0\alpha^2 + a_1(\alpha+\alpha^2) = a_1\alpha + (a_0+a_1)\alpha^2, \\ \alpha^2 A &= a_0(\alpha+\alpha^2) + a_1\alpha = (a_0+a_1)\alpha + a_0\alpha^2. \end{aligned}$$



(a) Multiplication by α Unit M_α



(b) Multiplication by α^2 Unit M_{α^2}

Figure 3: Multiplication by α and α^2 in $\mathbb{F}_{2^2}$ with Normal Bases

3.1.2 Arithmetic operations in $\mathbb{F}_{(2^2)^2}$.

Let $A = a_0\beta + a_1\beta^4$ and $B = b_0\beta + b_1\beta^4$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{2^2}$. A multiplication $C = AB$ in $\mathbb{F}_{(2^2)^2}$ is computed as follows (see Figure 4(a)):

$$\begin{aligned} AB &= (a_0\beta + a_1\beta^4)(b_0\beta + b_1\beta^4) \\ &= a_0b_0\beta^2 + (a_0b_1 + a_1b_0)\beta^5 + a_1b_1\beta^8 \\ &= [(a_0+a_1)(b_0+b_1)\alpha + a_0b_0]\beta + \\ &\quad [(a_0+a_1)(b_0+b_1)\alpha + a_1b_1]\beta^4 = c_0\beta + c_1\beta^4 = C. \end{aligned}$$

For a non-zero element $A \in \mathbb{F}_{(2^2)^2}$, the square of A is calculated as follows (see Figure 4(b)):

$$\begin{aligned} A^2 &= (a_0\beta + a_1\beta^4)^2 = a_0^2\beta^2 + a_1^2\beta^8 \\ &= a_0^2[(\alpha+1)\beta + \alpha\beta^4] + a_1^2[\alpha\beta + (\alpha+1)\beta^4] \\ &= [(a_0^2+a_1^2)\alpha + a_0^2]\beta + [(a_0^2+a_1^2)\alpha + a_1^2]\beta^4 \\ &= s_0\beta + s_1\beta^4 = S. \end{aligned}$$

Table 2: Tower Construction with Normal Bases

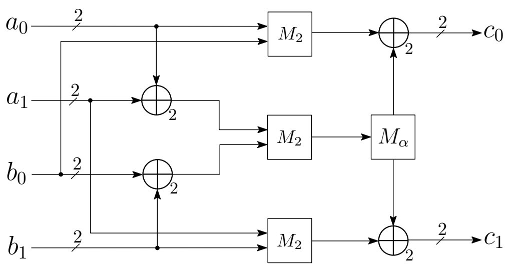
Finite Field	Normal Basis	Defining Polynomial
$\mathbb{F}_{2^{16}} \cong \mathbb{F}_{(((2^2)^2)^2)^2}$	$\{\delta, \delta^{256}\}$, where $\delta = \omega^{45049}$	$h(X) = X^2 + X + \mu$, where $\mu = \beta + \lambda\gamma$
$\mathbb{F}_{2^8} \cong \mathbb{F}_{((2^2)^2)^2}$	$\{\gamma, \gamma^{16}\}$, where $\gamma = z^7$	$g(X) = X^2 + X + \lambda$, where $\lambda = \alpha^2\beta$
$\mathbb{F}_{2^4} \cong \mathbb{F}_{(2^2)^2}$	$\{\beta, \beta^4\}$, where $\beta = y$	$f(X) = X^2 + X + \alpha$
$\mathbb{F}_{2^2} \cong \mathbb{F}_{(2)^2}$	$\{\alpha, \alpha^2\}$, where $\alpha = x$	$e(X) = X^2 + X + 1$

The Frobenius mapping of A with respect to $\mathbb{F}_{2^2}$, which is the 4th power operation, is computed as follows:

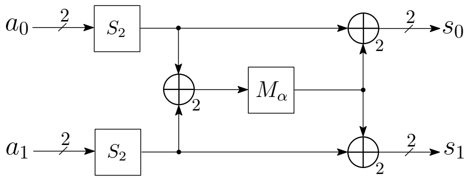
$$A^{2^2} = (a_0\beta + a_1\beta^4)^4 = a_0\beta^4 + a_1\beta^{16} = a_1\beta + a_0\beta^4.$$

Letting A be a non-zero element in $\mathbb{F}_{(2^2)^2}$, the inverse of A, denoted by I, can be calculated by the Itoh-Tsujii algorithm (ITA) [10] as follows (see Figure 4(c)):

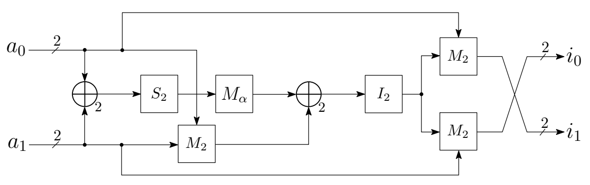
$$\begin{aligned} A^{-1} &= (AA^4)^{-1}A^4 \\ &= [(a_0\beta + a_1\beta^4)(a_1\beta + a_0\beta^4)]^{-1}(a_1\beta + a_0\beta^4) \\ &= [(a_0 + a_1)^2\alpha + a_0a_1]^{-1}(a_1\beta + a_0\beta^4) \\ &= i_0\beta + i_1\beta^4 = I. \end{aligned}$$



(a) Multiplication Unit M_4



(b) Squaring Unit S_4

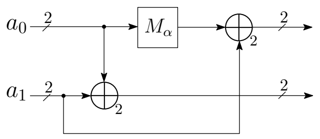


(c) Inversion Unit I_4

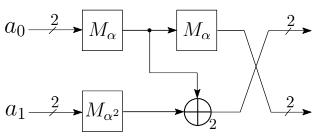
Figure 4: Multiplication, Squaring, and Inversion in $\mathbb{F}_{(2^2)^2}$ with Normal Bases

The multiplications of $A \in \mathbb{F}_{(2^2)^2}$ by $\lambda, \lambda^2, \beta$ and $\alpha\beta$ are carried out as follows (see Figure 5):

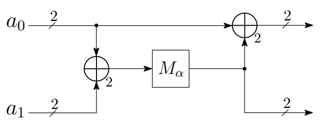
$$\begin{aligned} \lambda A &= \alpha^2\beta A \\ &= a_0\alpha^2[(\alpha+1)\beta + \alpha\beta^4] + a_1\alpha^2(\alpha\beta + \alpha\beta^4) \\ &= (a_0\alpha + a_1)\beta + (a_0 + a_1)\beta^4, \\ \lambda^2 A &= \alpha\beta^2 A \\ &= a_0\alpha\beta^3 + a_1\alpha\beta^6 = a_0\alpha(\beta + \alpha\beta^4) + a_1\alpha^2\beta \\ &= (a_0\alpha + a_1\alpha^2)\beta + (a_0\alpha^2)\beta^4, \\ \beta A &= a_0[(\alpha+1)\beta + \alpha\beta^4] + a_1(\alpha\beta + \alpha\beta^4) \\ &= [a_0 + (a_0 + a_1)\alpha]\beta + [(a_0 + a_1)\alpha]\beta^4, \\ \alpha\beta A &= a_0(\beta + \alpha^2\beta^4) + a_1(\alpha^2\beta + \alpha^2\beta^4) \\ &= (a_0 + a_1\alpha^2)\beta + (a_0\alpha^2 + a_1\alpha^2)\beta^4. \end{aligned}$$



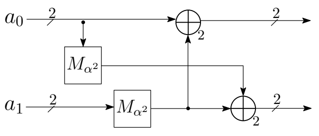
(a) Multiplication by λ Unit M_λ



(b) Multiplication by λ^2 Unit M_{λ^2}



(c) Multiplication by β Unit M_β



(d) Multiplication by $\alpha\beta$ Unit $M_{\alpha\beta}$

Figure 5: Multiplication by $\lambda, \lambda^2, \beta$ and $\alpha\beta$ in $\mathbb{F}_{(2^2)^2}$ with Normal Bases

3.1.3 Arithmetic operations in $\mathbb{F}_{((2^2)^2)^2}$.

Let $A = a_0\gamma + a_1\gamma^{16}$ and $B = b_0\gamma + b_1\gamma^{16}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{(2^2)^2}$. A multiplication $C = AB$ in $\mathbb{F}_{((2^2)^2)^2}$ is carried out as follows (see Figure 6(a)):

$$\begin{aligned} AB &= (a_0\gamma + a_1\gamma^{16})(b_0\gamma + b_1\gamma^{16}) \\ &= a_0b_0\gamma^2 + (a_0b_1 + a_1b_0)\gamma^{17} + a_1b_1\gamma^{32} \\ &= [(a_0 + a_1)(b_0 + b_1)\lambda + a_0b_0]\gamma + \\ &\quad [(a_0 + a_1)(b_0 + b_1)\lambda + a_1b_1]\gamma^{16} \\ &= c_0\gamma + c_1\gamma^{16} = C. \end{aligned}$$

For a non-zero element $A \in \mathbb{F}_{((2^2)^2)^2}$, the square of A is calculated as follows (see Figure 6(b)):

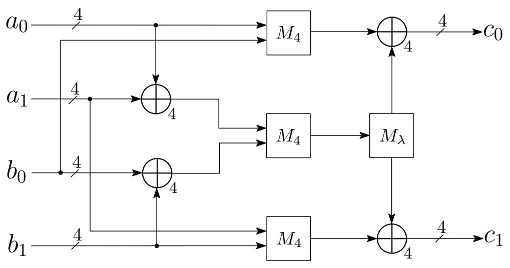
$$\begin{aligned} A^2 &= (a_0\gamma + a_1\gamma^{16})^2 = a_0^2\gamma^2 + a_1^2\gamma^{32} \\ &= a_0^2[(\lambda+1)\gamma + \lambda\gamma^{16}] + a_1^2[\lambda\gamma + (\lambda+1)\gamma^{16}] \\ &= [(a_0^2 + a_1^2)\lambda + a_0^2]\gamma + [(a_0^2 + a_1^2)\lambda + a_1^2]\gamma^{16} \\ &= s_0\gamma + s_1\gamma^{16} = S. \end{aligned}$$

The Frobenius mapping of A with respect to $\mathbb{F}_{2^4}$, which is the 16th power operation, is computed as follows:

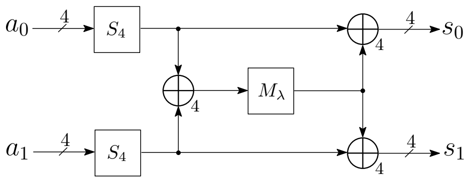
$$A^{2^4} = (a_0\gamma + a_1\gamma^{16})^{16} = a_0\gamma^{16} + a_1\gamma^{256} = a_1\gamma + a_0\gamma^{16}.$$

Letting A be a non-zero element in $\mathbb{F}_{((2^2)^2)^2}$, the inverse of A, denoted by I, can be calculated by the ITA as follows (see Figure 6(c)):

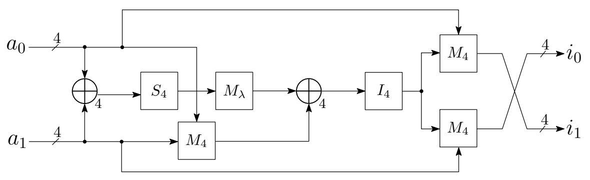
$$\begin{aligned} A^{-1} &= (AA^{16})^{-1}A^{16} \\ &= [(a_0\gamma + a_1\gamma^{16})(a_1\gamma + a_0\gamma^{16})]^{-1}(a_1\gamma + a_0\gamma^{16}) \\ &= [(a_0+a_1)^2\lambda + a_0a_1]^{-1}(a_1\gamma + a_0\gamma^{16}) \\ &= i_0\gamma + i_1\gamma^{16} = I. \end{aligned}$$



(a) Multiplication Unit M_8



(b) Squaring Unit S_8



(c) Inversion Unit I_8

Figure 6: Multiplication, Squaring, and Inversion in $\mathbb{F}_{((2^2)^2)^2}$ with Normal Bases

The multiplication of $A \in \mathbb{F}_{((2^2)^2)^2}$ by μ is carried out as follows (see Figure 7):

$$\begin{aligned} \mu A &= (\beta + \lambda\gamma)(a_0\gamma + a_1\gamma^{16}) \\ &= a_0\beta\gamma + a_1\beta\gamma^{16} + a_0\lambda\gamma^2 + a_1\lambda\gamma^{17} \\ &= [a_0(\alpha\beta) + (a_0+a_1)\lambda^2]\gamma + [a_1\beta + (a_0+a_1)\lambda^2]\gamma^{16} \end{aligned}$$

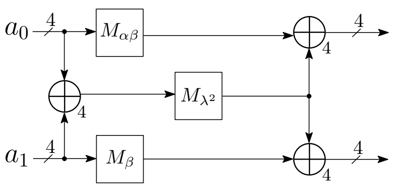


Figure 7: Multiplication by μ Unit M_μ in $\mathbb{F}_{((2^2)^2)^2}$ with Normal Bases

3.1.4 *Arithmetic operations in $\mathbb{F}_{(((2^2)^2)^2)^2}$.*

Let $A = a_0\delta + a_1\delta^{256}$ and $B = b_0\delta + b_1\delta^{256}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{((2^2)^2)^2}$. A multiplication $C = AB$ in $\mathbb{F}_{(((2^2)^2)^2)^2}$ is computed as follows (see Figure 8(a)):

$$\begin{aligned} AB &= (a_0\delta + a_1\delta^{256})(b_0\delta + b_1\delta^{256}) \\ &= a_0b_0\delta^2 + (a_0b_1 + a_1b_0)\delta^{257} + a_1b_1\delta^{512} \\ &= [(a_0+a_1)(b_0+b_1)\mu + a_0b_0]\delta + \\ &\quad [(a_0+a_1)(b_0+b_1)\mu + a_1b_1]\delta^{256} \\ &= c_0\delta + c_1\delta^{256} = C. \end{aligned}$$

For a non-zero element $A \in \mathbb{F}_{(((2^2)^2)^2)^2}$, the square of A is calculated as follows (see Figure 8(b)):

$$\begin{aligned} A^2 &= (a_0\delta + a_1\delta^{256})^2 = a_0^2\delta^2 + a_1^2\delta^{512} \\ &= a_0^2[(\mu+1)\delta + \mu\delta^{256}] + a_1^2[\mu\delta + (\mu+1)\delta^{256}] \\ &= [(a_0^2 + a_1^2)\mu + a_0^2]\delta + [(a_0^2 + a_1^2)\mu + a_1^2]\delta^{256} \\ &= s_0\delta + s_1\delta^{256} = S. \end{aligned}$$

The Frobenius mapping of A with respect to $\mathbb{F}_{2^8}$, which is the 256th power operation, is computed as follows:

$$A^{2^8} = (a_0\delta + a_1\delta^{256})^{256} = a_0\delta^{256} + a_1\delta^{65536} = a_1\delta + a_0\delta^{256}.$$

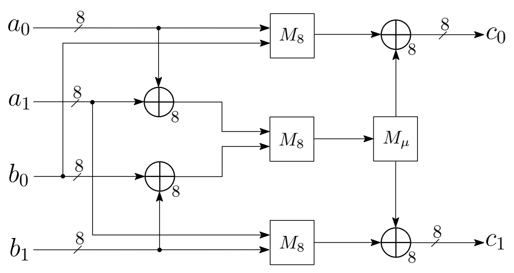
Letting A be a non-zero element in $\mathbb{F}_{(((2^2)^2)^2)^2}$, the inverse of A, denoted by I, can be calculated by the ITA as follows (see Figure 8(c)):

$$\begin{aligned} A^{-1} &= (AA^{256})^{-1}A^{256} \\ &= [(a_0\delta + a_1\delta^{256})(a_1\delta + a_0\delta^{256})]^{-1}(a_1\delta + a_0\delta^{256}) \\ &= [(a_0+a_1)^2\mu + a_0a_1]^{-1}(a_1\delta + a_0\delta^{256}) \\ &= i_0\delta + i_1\delta^{256} = I. \end{aligned}$$

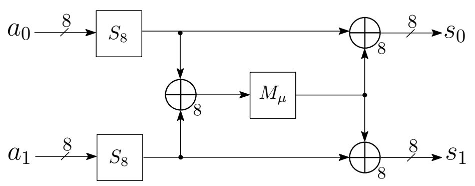
3.1.5 *Efficient Conversion Matrices.*

Two matrices $\mathbf{M}_{NT}$ and $\mathbf{M}_{TN}$ are needed for converting elements between normal basis and tower field representations. As noticed by Nogami *et al.* in [18], these conversion matrices are easily found but they are not uniquely determined because the modular polynomials $e(X), f(X), g(X)$ and $h(X)$ have conjugate elements as zeros. In particular, efficient conversion matrices that lead to small critical path delay are rare. We conduct an exhaustive search with 16 conjugate variants of conversion matrix and the best pair of $\mathbf{M}_{NT}$ and $\mathbf{M}_{TN}$ is shown below:

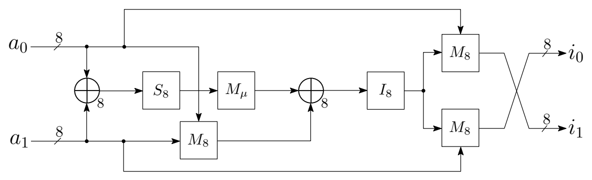
$$\mathbf{M}_{NT} = \begin{bmatrix} 1&1&0&1&0&0&0&1&1&0&1&1&0&0&0&0 \\ 0&0&0&0&0&0&1&1&0&1&0&1&0&0&1&0 \\ 1&0&0&0&1&0&0&1&0&1&0&0&1&0&0&0 \\ 0&1&1&1&0&0&0&1&0&0&0&0&0&1&1&1 \\ 1&0&0&1&0&1&0&0&1&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0&0&0&0&1&0&1&1&1 \\ 0&0&1&0&0&0&1&0&0&0&1&1&1&1&1&0 \\ 0&0&1&0&0&0&1&0&0&1&0&0&0&1&0&1 \\ 1&0&1&1&0&0&0&0&1&1&0&1&0&0&0&1 \\ 0&1&0&1&0&0&1&0&0&0&0&0&0&0&1&1 \\ 0&1&0&0&1&0&0&0&1&0&0&0&1&0&0&1 \\ 0&0&0&0&0&1&1&1&0&1&1&1&0&0&0&1 \\ 1&0&0&0&0&0&1&0&1&0&0&1&0&1&0&0 \\ 0&0&0&1&0&1&1&1&0&0&0&0&0&0&1&0 \\ 0&0&1&1&1&1&1&0&0&0&1&0&0&0&1&0 \\ 0&1&0&0&0&1&0&1&0&0&1&0&0&0&1&0 \end{bmatrix}$$



(a) Multiplication Unit M_{16}



(b) Squaring Unit S_{16}



(c) Inversion Unit I_{16}

Figure 8: Multiplication, Squaring, and Inversion in $\mathbb{F}_{(((2^2)^2)^2)^2}$ with Normal Bases

and

$$\mathbf{M}_{NT} = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1
\end{bmatrix}.$$

Note that the pair of $\mathbf{M}_{NT}$ and $\mathbf{M}_{TN}$ achieves $\text{WT}(\mathbf{M}_{NT}) + \text{WT}(\mathbf{M}_{TN}) = 92+100 = 192$, where $\text{WT}(\cdot)$ counts the number of 1's in a matrix (i.e., the weight of a binary matrix). Moreover, the Hamming weight of row vectors of $\mathbf{M}_{NT}$ and $\mathbf{M}_{TN}$ is less than or equal to 7 and 9, respectively. As a result, the critical path delays of implementing $\mathbf{M}_{NT}$ and $\mathbf{M}_{TN}$ using the tree structure [2] are $3T_X$ and $4T_X$, respectively, where T_X denotes the delay of a XOR gate.

3.2 Hardware and Time Complexities of Tower Construction

We summarize the hardware and time complexities of the building blocks in tower construction with normal bases in Table 3, where N_X (resp. T_X) and N_A (resp. T_A) denote the number (resp. the delay) of XOR gates and AND gates, respectively.

The tower construction described in Section 3 allows a hardware architecture with a highly regular structure, having almost identical basic building blocks for each layer. This high level of regularity allows accurate prediction of area complexities for basic building blocks on higher lever of the tower field, based on results obtained in the base field. If we refer to Table 3 and compare area complexities for multipliers M_2 and M_4, we can observer that M_4 will contain three M_2 blocks (so 12 XOR gates and 2 AND gates), a M_α block (one XOR gate) and four 2-bit XOR gates, adding up to a total of 21 XOR gates and 9 AND gates in multiplier M_4.

Table 3: Hardware and Time Complexities of Building Blocks in Section 3

Tower Field	Building Block	N_X	N_A	Critical Path Delay
$\mathbb{F}_{2^2}$	Multiplication (M_2)	4	3	$2T_X + T_A$
	Squaring(S_2)/ Inversion(I_2)	0	0	0
$\mathbb{F}_{(2^2)^2}$	Multiplication (M_4)	21	9	$5T_X + T_A$
	Squaring (S_4)	7	0	$3T_X$
	Inversion (I_4)	17	9	$5T_X + 2T_A$
$\mathbb{F}_{((2^2)^2)^2}$	Multiplication (M_8)	84	27	$9T_X + T_A$
	Squaring (S_8)	31	0	$7T_X$
	Inversion (I_8)	100	36	$17T_X + 3T_A$
$\mathbb{F}_{(((2^2)^2)^2)^2}$	Multiplication (M_{16})	312	81	$15T_X + T_A$
	Squaring (S_{16})	114	0	$13T_X$
	Inversion (I_{16})	427	117	$39T_X + 4T_A$
	Conversion $\mathbf{M}_{NT}$	76	0	$3T_X$
	Conversion $\mathbf{M}_{TN}$	84	0	$4T_X$

4. A COMPACT HARDWARE ARCHITECTURE OF THE WG-16 STREAM CIPHER

In this section, we describe a compact hardware architecture for the WG-16 stream cipher.

4.1 Properties of the Trace Function in Tower Field Representation

In [4], El-Razouk *et al.* proved that when the elements in $\mathbb{F}_{2^m}$ are represented in a type-II ONB [8] the trace of the product of any two elements can be efficiently computed as the modulo-2 sum of coordinates of the bitwise ANDing of the two elements. Despite the lack of Gaussian normal bases over $\mathbb{F}_{2^{16}}$, we show that using the isomorphic tower field $\mathbb{F}_{(((2^2)^2)^2)^2}$ as constructed in Section 3.1, the above nice property still holds.

LEMMA 1. *Given the tower construction in Section 3.1, we have the following basic facts:*

- *For any two elements $A = a_0\delta + a_1\delta^{256}$ and $B = b_0\delta + b_1\delta^{256}$ in $\mathbb{F}_{(((2^2)^2)^2)^2}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{((2^2)^2)^2}$, we have $Tr^{\mathbb{F}_{(((2^2)^2)^2)^2}}_{\mathbb{F}_{((2^2)^2)^2}}(AB) = (a_0 \otimes_8 b_0) \oplus_8 (a_1 \otimes_8 b_1)$.*
- *For any two elements $A = a_0\gamma + a_1\gamma^{16}$ and $B = b_0\gamma + b_1\gamma^{16}$ in $\mathbb{F}_{((2^2)^2)^2}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{(2^2)^2}$, we have $Tr^{\mathbb{F}_{((2^2)^2)^2}}_{\mathbb{F}_{(2^2)^2}}(AB) = (a_0 \otimes_4 b_0) \oplus_4 (a_1 \otimes_4 b_1)$.*
- *For any two elements $A = a_0\beta + a_1\beta^4$ and $B = b_0\beta + b_1\beta^4$ in $\mathbb{F}_{(2^2)^2}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_{2^2}$, we have $Tr^{\mathbb{F}_{(2^2)^2}}_{\mathbb{F}_{2^2}}(AB) = (a_0 \otimes_2 b_0) \oplus_2 (a_1 \otimes_2 b_1)$.*
- *For any two elements $A = a_0\alpha + a_1\alpha^2$ and $B = b_0\alpha + b_1\alpha^2$ in $\mathbb{F}_{2^2}$, where $a_0, a_1, b_0, b_1 \in \mathbb{F}_2$, we have $Tr^{\mathbb{F}_{2^2}}_{\mathbb{F}_2}(AB) = (a_0 \odot_1 b_0) \oplus_1 (a_1 \odot_1 b_1)$.*

Proof. Noting that
$\mathrm{Tr}_{\mathbb{F}_{((2^2)^2)^2}}^{\mathbb{F}_{(((2^2)^2)^2)^2}}(\delta) = \mathrm{Tr}_{\mathbb{F}_{((2^2)^2)^2}}^{\mathbb{F}_{(((2^2)^2)^2)^2}}(\delta^{256}) = 1$,
$\mathrm{Tr}_{\mathbb{F}_{(2^2)^2}}^{\mathbb{F}_{((2^2)^2)^2}}(\gamma) = \mathrm{Tr}_{\mathbb{F}_{(2^2)^2}}^{\mathbb{F}_{((2^2)^2)^2}}(\gamma^{16}) = 1$,
$\mathrm{Tr}_{\mathbb{F}_{2^2}}^{\mathbb{F}_{(2^2)^2}}(\beta) = \mathrm{Tr}_{\mathbb{F}_{2^2}}^{\mathbb{F}_{(2^2)^2}}(\beta^4) = 1$,
$\mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^2}}(\alpha) = \mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^2}}(\alpha^2) = 1$
and using the multiplication formulae derived in Section 3.1, the results follow. □

Proposition 1. *Given the tower construction in Section 3.1, the trace of the product of any elements $U = (u_0, u_1, \ldots, u_{15})$ and $V = (v_0, v_1, \ldots, v_{15})$ in $\mathbb{F}_{(((2^2)^2)^2)^2}$ can be computed as the modulo-2 sum of the coordinates of the bitwise ANDing operation of U and V, i.e.,*

$$Tr(UV) = \bigoplus_{i=0}^{15}(u_i \odot_1 v_i).$$

Proof. With the tree field representation, we have

$$\begin{aligned}
U &= u_{0..7}\delta + u_{8..15}\delta^{256} \\
&= (u_{0..3}\gamma + u_{4..7}\gamma^{16})\delta + (u_{8..11}\gamma + u_{12..15}\gamma^{16})\delta^{256} \\
&= ((u_{0..1}\beta + u_{2..3}\beta^4)\gamma + (u_{4..5}\beta + u_{6..7}\beta^4)\gamma^{16})\delta + \\
&\quad ((u_{8..9}\beta + u_{10..11}\beta^4)\gamma + (u_{12..13}\beta + u_{14..15}\beta^4)\gamma^{16})\delta^{256} \\
&= \Big(\big((u_0\alpha + u_1\alpha^2)\beta + (u_2\alpha + u_3\alpha^2)\beta^4\big)\gamma + \\
&\quad \big((u_4\alpha + u_5\alpha^2)\beta + (u_6\alpha + u_7\alpha^2)\beta^4\big)\gamma^{16}\Big)\delta + \\
&\quad \Big(\big((u_8\alpha + u_9\alpha^2)\beta + (u_{10}\alpha + u_{11}\alpha^2)\beta^4\big)\gamma + \\
&\quad \big((u_{12}\alpha + u_{13}\alpha^2)\beta + (u_{14}\alpha + u_{15}\alpha^2)\beta^4\big)\gamma^{16}\Big)\delta^{256},
\end{aligned}$$

where $u_{j..j+7} \in \mathbb{F}_{((2^2)^2)^2}$ for $j = 0, 8$, $u_{j..j+3} \in \mathbb{F}_{(2^2)^2}$ for $j = 0, 4, 8, 12$, $u_{j..j+1} \in \mathbb{F}_{2^2}$ for $j = 0, 2, 4, \ldots, 14$ and $u_j \in \mathbb{F}_2$ for $j = 0, \ldots, 15$ and V has a similar representation. The trace of the product of U and V then can be computed as follows:

$$\begin{aligned}
\mathrm{Tr}(UV) &= \mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^2}} \circ \mathrm{Tr}_{\mathbb{F}_{2^2}}^{\mathbb{F}_{(2^2)^2}} \circ \mathrm{Tr}_{\mathbb{F}_{(2^2)^2}}^{\mathbb{F}_{((2^2)^2)^2}} \circ \\
&\quad \mathrm{Tr}_{\mathbb{F}_{((2^2)^2)^2}}^{\mathbb{F}_{(((2^2)^2)^2)^2}}\big((u_{0..7}\delta + u_{8..15}\delta^{256}) \\
&\quad (v_{0..7}\delta + v_{8..15}\delta^{256})\big) \\
&= \mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^2}} \circ \mathrm{Tr}_{\mathbb{F}_{2^2}}^{\mathbb{F}_{(2^2)^2}} \circ \mathrm{Tr}_{\mathbb{F}_{(2^2)^2}}^{\mathbb{F}_{((2^2)^2)^2}}((u_{0..7} \otimes_8 v_{0..7}) \oplus_8 \\
&\quad (u_{8..15} \otimes_8 v_{8..15})) \\
&= \bigoplus_{i=0,8}\Big(\mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^2}} \circ \mathrm{Tr}_{\mathbb{F}_{2^2}}^{\mathbb{F}_{(2^2)^2}}((u_{i..i+3} \otimes_4 v_{i..i+3}) \oplus_4 \\
&\quad (u_{i+4..i+7} \otimes_4 v_{4..7}))) \\
&= \bigoplus_{i=0,4,8,12}\Big(\mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^2}}((u_{i..i+1} \otimes_2 v_{i..i+1}) \oplus_2 \\
&\quad (u_{i+2..i+3} \otimes_2 v_{i+2..i+3}))) \\
&= \bigoplus_{i=0,2,4,\ldots,14}\Big(\mathrm{Tr}_{\mathbb{F}_2}^{\mathbb{F}_{2^2}}(u_{i..i+1} \otimes_2 v_{i..i+1})\Big) \\
&= \bigoplus_{i=0}^{15}(u_i \odot_1 v_i),
\end{aligned}$$

where each equation follows from one basic fact in Lemma 1. □

Corollary 1. *Given the tower construction in Section 3.1, for any elements $X = (x_0, x_1, \ldots, x_{15})$, $U = (u_0, u_1, \ldots, u_{15})$ and $V = (v_0, v_1, \ldots, v_{15})$ in $\mathbb{F}_{(((2^2)^2)^2)^2}$ we have $Tr(X^{2^w}) = Tr(X) = \bigoplus_{i=0}^{15} x_i$ and $Tr(UV) = Tr\left(U^{2^w} \odot_{16} V^{2^w}\right)$, where w is an integer.*

Proof. Given the tower construction in Section 3.1, the element $1 \in \mathbb{F}_{2^{16}}$ can be denoted by $(1, 1, \ldots, 1)$. Therefore, we immediately obtain $\mathrm{Tr}(X) = \bigoplus_{i=0}^{15} x_i$ by setting $U = X$ and $V = 1$ in Proposition 1. Noting that $\mathrm{Tr}(X^{2^w}) = \mathrm{Tr}(X)$ for any $X \in \mathbb{F}_{(((2^2)^2)^2)^2}$, we obtain the following result by setting $X = UV$ and using the Proposition 1:

$$\mathrm{Tr}(UV) = \mathrm{Tr}\left((UV)^{2^w}\right) = \mathrm{Tr}\left(U^{2^w}V^{2^w}\right) = \mathrm{Tr}(U^{2^w} \odot_{16} V^{2^w}).$$

□

Corollary 2. *Given the tower construction in Section 3.1, for any elements U, V and W in $\mathbb{F}_{(((2^2)^2)^2)^2}$ we have*

$$Tr(U \odot_{16} W) \oplus_1 Tr(V \odot_{16} W) = Tr((U \oplus_{16} V) \odot_{16} W).$$

Proof. The proof is the same as the Corollary 2 in [4]. □

4.2 An Optimized Hardware Architecture of the WGT-16(X^d) Module

Based on the Proposition and two Corollaries in Section 4.1, the WG-16 transformation WGT-16(X^d) can be computed as follows:

$$\begin{aligned}
& \mathsf{WGT\text{-}16}(X^d) = \mathrm{Tr}\left(\mathsf{WGP\text{-}16}(X^d)\right) \\
&= \mathrm{Tr}\left(q(X^{1057} \oplus_{16} 1) \oplus_{16} 1\right) = \mathrm{Tr}\left(q(X^{1057} \oplus_{16} 1)\right) \\
&= \mathrm{Tr}\Big(Y \oplus_{16} Y^{2^{11}+1} \oplus_{16} Y^{2^{11}(2^{11}-1)+1} \oplus_{16} \\
&\quad Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)\Big) \\
&= \mathrm{Tr}\left(Y \oplus_{16} Y^{2^{11}+1}\right) \oplus_1 \mathrm{Tr}\left(YY^{2^{11}(2^{11}-1)}\right) \oplus_1 \\
&\quad \mathrm{Tr}\left(Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)\right) \\
&= \mathrm{Tr}\left(Y \oplus_{16} Y^{2^{11}+1}\right) \oplus_1 \mathrm{Tr}\left(Y^{2^6} \odot_{16} Y^{2(2^{11}-1)}\right) \oplus_1 \\
&\quad \mathrm{Tr}\left(Y^{2^6} \odot_{16} \left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)\right) \\
&= \mathrm{Tr}\left(Y \oplus_{16} Y^{2^{11}+1}\right) \oplus_1 \\
&\quad \mathrm{Tr}\left(Y^{2^6} \odot_{16} \left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1} \oplus_{16} Y^{2(2^{11}-1)}\right)\right),
\end{aligned}$$

where $Y = X^{1057} \oplus_{16} 1 = X^{2^{10}+2^5+1} \oplus_{16} 1$ and the last three equations follow from the Proposition 1 and Corollaries 1 and 2, respectively. Note that $Y^{2^{11}-1}$ can be computed by one of the following two methods:

1. $Y^{2^{11}-1} = Y^{\left((1+2)(1+2^2)+2^4\right)(1+2^5)+2^{10}}$, which requires five multipliers M_{16} with a hardware complexity of $N_X = 1,560$ and $N_A = 405$ and a critical path delay of $75T_X + 5T_A$.
2. $Y^{2^{11}-1} = Y^{2^{11}}Y^{-1}$, which requires one multiplier M_{16} and one inverter I_{16} with a hardware complexity of $N_X = 739$ and $N_A = 198$ as well as a critical path delay of $54T_X + 5T_A$.

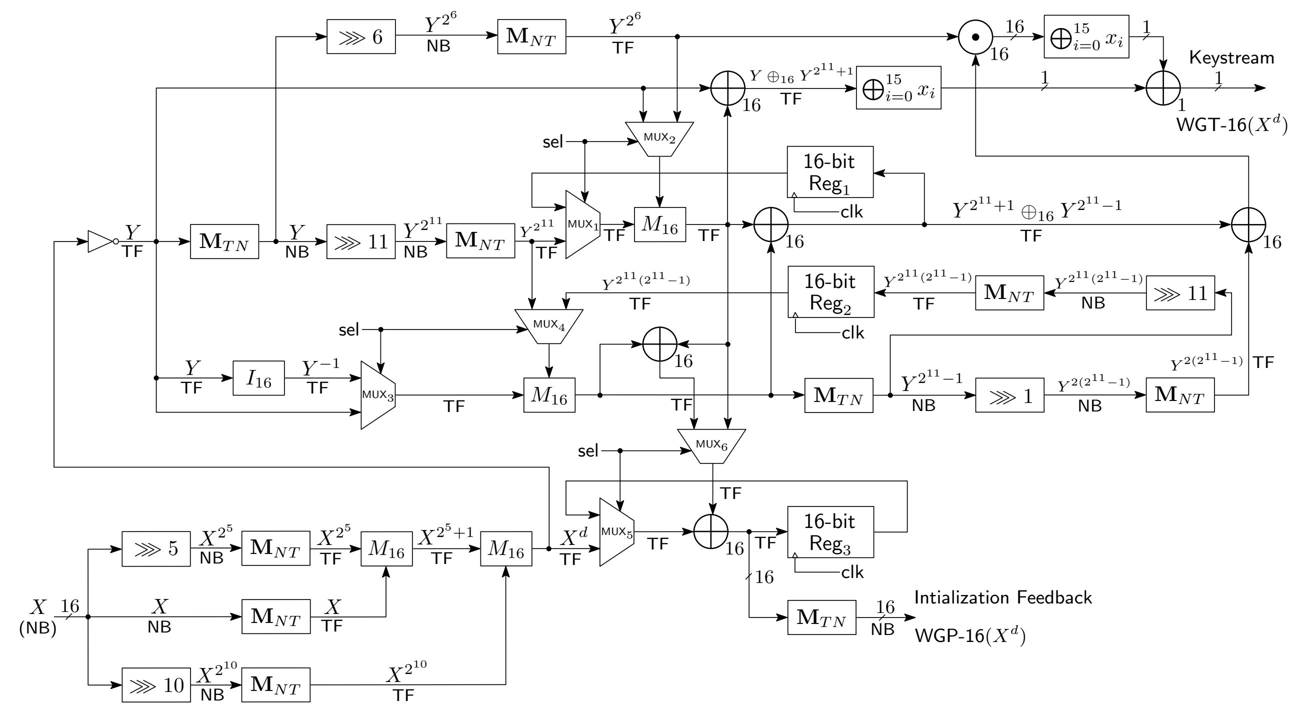


Figure 9: The Integrated Hardware Architecture WGP_T for Computing WGP-16(X^d) and WGT-16(X^d)

It is not difficult to find that the second method is more efficient in terms of both hardware and time complexities. Therefore, the WG-16 transformation WGT-16(X^d) can be computed using four multipliers in total, where two multipliers are utilized for generating Y, one for computing $Y^{2^{11}+1}$ and one for calculating $Y^{2^{11}-1}$. In particular, thanks to the nice property for computing the trace of the product of any two elements with tower field representations in $\mathbb{F}_{(((2^2)^2)^2)^2}$, the two multiplications $Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)$ and $YY^{2^{11}(2^{11}-1)}$ inside the trace function have been replaced by bitwise AND and XOR operations, which reduces both hardware and time complexities significantly.

4.3 An Integrated Hardware Architecture of the WGP-16(X^d) Module

In the key/IV initialization phase, a 16-bit output from WGP-16(X^d) needs to be used as a nonlinear feedback to the LFSR, where the WG-16 permutation WGP-16(X^d) can be computed as follows:

$$\begin{aligned}
&\text{WGP-16}(X^d) = q(X^{1057} \oplus_{16} 1) \oplus_{16} 1 \\
=\ &\left(1 \oplus_{16} Y \oplus_{16} Y^{2^{11}+1}\right) \oplus_{16} \\
&\left(YY^{2^{11}(2^{11}-1)}\right) \oplus_{16} \left(Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)\right) \\
=\ &\left(X^{1057} \oplus_{16} Y^{2^{11}+1}\right) \oplus_{16} \\
&\left(YY^{2^{11}(2^{11}-1)}\right) \oplus_{16} \left(Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)\right),
\end{aligned}$$

where $Y = X^{1057} \oplus_{16} 1$. Note that $Y^{2^{11}+1}$ is computed in the WGT-16(X^d) module, whereas the two intermediate values $YY^{2^{11}(2^{11}-1)}$ and $Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)$ are missing in the WGT-16(X^d) module due to the application of the trace function. Considering the existence of four multipliers in the WGT-16(X^d) module, we are able to reuse two of them and serially compute WGP-16(X^d) over two consecutive clock cycles. To conduct the serial computation of WGP-16(X^d) within two clock cycles, we have introduced additional six multiplexers MUX$_i$ ($i = 1, 2, \ldots, 6$) and three 16-bit registers Reg$_i$ ($i = 1, 2, 3$) into the hardware architecture of the WGT-16(X^d) module. A complete and compact hardware architecture that integrates WGP-16(X^d) and WGT-16(X^d) modules, denoted by WGP_T, is illustrated in Figure 9 and a more detailed description of the pipelined design will be presented in Section 5.

4.4 Hardware and Time Complexities

The WG-16 hardware core is composed of three components: a) a FSM; b) a 32-stage LFSR[1]; and c) an integrated WGT-16(X^d)/WGP-16(X^d) module as illustrated in Figure 9. Let N_R, N_A, N_X, N_O, and N_I denote the number of Registers, AND gates, XOR gates, OR gates, and Inverters, respectively. We summarize the hardware complexity for implementing the WG-16 stream cipher in Table 4.

Let T_R, T_A, T_X, T_O, and T_I denote the delay of a Register, an AND gate, a XOR gate, an OR gate, and an Inverter, respectively. We first notice that the delay through the LFSR is significantly smaller than that through the integrated WGT-16(X^d)/WGP-16(X^d) module, due to the less number of multipliers when compared to the WG transformation.

[1]For evaluating the hardware complexity of the LFSR, we include the cost of two 2-to-1 16-bit multiplexers associated with the LFSR (see Figure 1). Moreover, the multiplication by the constant ω^{2743} can be implemented by multiplying with a 16×16 matrix $\mathbf{M}_{\omega^{2743}}$ such that $\mathrm{WT}(\mathbf{M}_{\omega^{2743}}) = 110$.

Table 4: Hardware Complexity of the WG-16 Stream Cipher

Component	N_R	N_A	N_X	N_O	N_I
LFSR	512	64	158	32	32
WGT-16(X^d)/WGP-16(X^d)	48	649	$2,555$	96	112

The following two lemmas characterize the critical path delays of the initialization and running phases of the WG-16 hardware core. The longest paths of the initialization and running phases of the WG-16 hardware core are respectively given by

$$T_{Init} = 88T_X + 8T_A + T_R + T_O + 2T_I \text{ and}$$
$$T_{Run} = 94T_X + 9T_A + T_R + T_O + 2T_I.$$

The lemma can be easily obtained by adding the delays of the components on the longest pathes during the initialization and running phases. From the lemma, we can see that the maximum delay through the WG-16 hardware core is $T_{Run} = 94T_X + 9T_A + T_R + T_O + 2T_I$. Noting that the critical path delay in the above WG-16 hardware core is long, we present a pipelined design in the next section.

5. A PIPELINED DESIGN OF THE WG-16 STREAM CIPHER

In this section, we describe a pipelined design of the WG-16 stream cipher in order to achieve a higher throughput.

5.1 A Pipelined Hardware Architecture

For creating a pipelined design, the integrated hardware architecture WGP_T in Figure 9 has been decomposed into two submodules module_A and module_B, as shown in Figure 10. While the module_A contains the common computational components that are shared by the initialization and running phase and outputs the values $Y^{2^6}, Y^{2^{11}}, Y^{-1}$ and X^d, the module_B performs the rest of computations. Both submodules are implemented as pipelines in order to increase the throughput for the entire architecture.

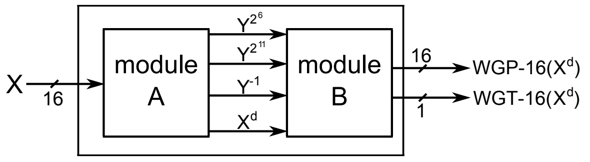


Figure 10: The Decomposed WGP_T Module as Submodules module_A and module_B

5.1.1 *Analysis of Basic Building Blocks*

In order to determine appropriate pipeline stages, all the basic building blocks (see Section 3) for performing the tower field arithmetic have been implemented as combinatorial circuits on the target FPGA and ASIC platforms. The area and delay of the basic building blocks on FPGA and AISC platforms are summarized in Table 5.

The FPGA device (i.e., Spartan-6 XC6SLLX9) used in our implementation features 6-input and 2-output look-up tables (LUTs), which can implement any 6-input Boolean functions. Recall from Section 3 that the two output bits c_0 and c_1 of M_2, namely

$$c_0 = (a_0 + a_1)(b_0 + b_1) + a_0 b_0$$
$$c_1 = (a_0 + a_1)(b_0 + b_1) + a_1 b_1,$$

are 4-input Boolean functions, computed on the same values of inputs a_0, a_1, b_0 and b_1. Hence, the M_2 multiplication can be realized on one LUT, using both outputs. In M_4 block, we expect to find four LUTs connected to the four output bits (the product) and the three LUTs for three M_2 blocks, which gives the minimum of 7 LUTs. The remaining LUTs are inferred to implement the XOR gates at the inputs. Similarly, going to M_8 level, we expect 8 LUTs on the outputs, together with the 33 LUTs for the three M_4 multipliers. Note that the lower level blocks M_4's are not integrated into M_8 directly. Instead they are broken down and their signals are rerouted without altering the functionality. Moreover, parts of M_4 blocks are combined with the last XORs, merged with computations from M_λ block, and realized in the LUTs connected directly to the M_8 outputs. Nevertheless, an approximate area complexity estimation still can be made and we can expect to find at least 16 output LUTs and 120 LUTs for M_8 multiplications in M_{16} block.

Table 5: FPGA and ASIC Implementation Results of Basic Building Blocks for Tower Field Arithmetic

Basic Building Block	FPGA Results			ASIC Results	
	# of LUTs	# of Slices	Block Delay [ns]	Area [GE]	Block Delay [ns]
M_2	1	1	6.669	22.9	0.17
S_2/I_2	0	0	5.512	0.0	0.00
M_4	11	11	8.517	10.3	0.57
S_4	2	2	6.984	25.0	0.28
I_4	2	2	6.984	75.4	0.56
M_8	40	14	10.613	401	1.13
S_8	6	3	7.118	116	0.73
I_8	41	15	12.915	400	2.13
M_{16}	148	52	13.925	1440	2.09
S_{16}	24	10	8.322	442	1.49
I_{16}	147	60	22.826	1684	5.02
$\mathbf{M}_{NT}$	17	7	7.800	219	0.33
$\mathbf{M}_{TN}$	18	8	7.963	210	0.36

5.1.2 *Design of Pipeline Architecture*

Based on the implementation of basic building blocks (see Table 5), it is obvious that using the inversion module I_{16} inside a pipeline stage is not the best option. Hence, the inversion module I_{16} (see Figure 8(c)) has been implemented using four pipeline stages: 1) the initial multiplication module M_8 and squaring module S_8 in parallel; 2) the $M\mu$ module; 3) the inversion I_8 module; and 4) the last two multiplication modules M_8 in parallel. We shall refer to this approach as pipelining at the I_8 level. Similarly, we refer to pipelining at M_8 level for M_{16} module (see Figure 8(a)) implemented with inter-stage registers inserted between three parallel M_8 modules and the $M\mu$ module.

By inspecting the integrated hardware architecture WGP_T in Figure 9, we identify the critical path for module_A to be the data-path from X to Y^{-1}. Based on implementation results of this data-path and previous discussion, the design of `module_A` pipeline narrows down to two options: pipelining the submodule at M_{16}/I_8 level, which results in a 7-stage

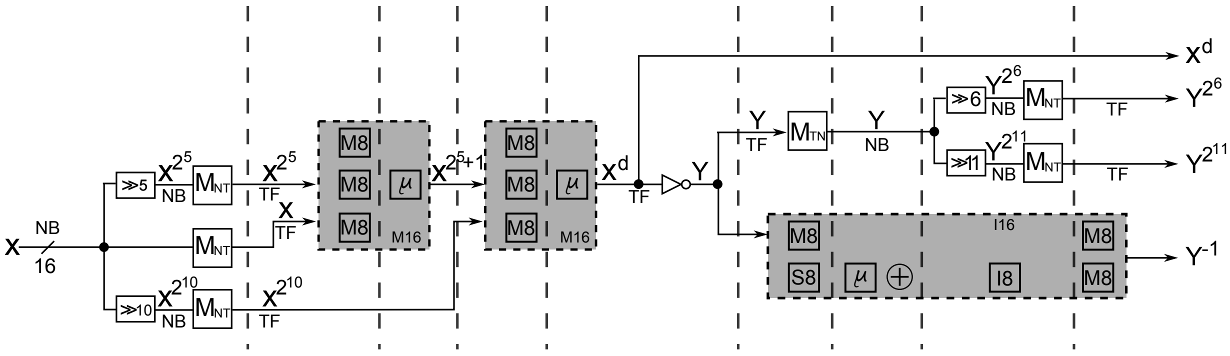


Figure 11: Submodule module_A with Vertical Dashed Lines Representing Pipeline-Stage Borders

pipeline, and pipelining at M_8/I_8 level, which requires two additional pipeline stages. Note that the initial exponentiations X^{2^5} and $X^{2^{10}}$, together with basis conversion, are carried out in the first stage of the pipeline. For the first design option, the two multiplications XX^{2^5} and $X^{2^{10}}X^{2^5}$ are computed atomically, on two M_{16} modules placed in two consecutive pipeline stages. In the latter case of M_8/I_8 pipelining the two multiplications are carried out over four pipeline stages. In both cases, the computation of Y^{-1} is implemented at I_8 level and requires four pipeline stages. The resulting pipelined design of module_A at M_8/I_8 level is illustrated in Figure 11, where the vertical lines represent pipeline-stage borders. The three grey blocks demonstrate the pipelined design for M_{16} and I_{16} modules (see Figures 8(a) and 8(c)) as explained before. For the M_{16}/I_8 level pipelining, one can omit two pipeline boarders inside two M_{16} modules in Figure 11.

Regarding to the pipeline design of module_B, we notice that two of the four multipliers belong to module_B and can be reused to serially compute WGP-16(X^d). To conduct the serial computation of WGP-16(X^d), six additional multiplexers MUX$_i$ ($i = 1, 2, \ldots, 6$) and three 16-bit registers Reg$_i$ ($i = 1, 2, 3$) are introduced into the hardware architecture of the WGT-16(X^d) module (see Figure 9). The outputs of six multiplexers and the updated states of three registers are summarized in Table 6.

module_B is designed by employing a two-stage pipeline as shown in Figure 12. While the solid lines represent the part of circuit used for the common computations of WGT-16(X^d) and WGT-16(X^d), the dotted lines represent the components exclusively used for computing WGP-16(X^d). During the first clock cycle, the control signal sel is at low logic level and thus MUX$_i$ ($i = 1, 2, \ldots, 6$) generate the signals $Y^{2^{11}}, Y, Y^{-1}, Y^{2^{11}}$, X^{1057} and $Y^{2^{11}+1}$ at their outputs, respectively. In the first pipeline-stage, intermediate products $YY^{2^{11}}$ and $Y^{-1}Y^{2^{11}}$ and the value $X^{1057} \oplus Y^{2^{11}+1}$ are computed. The value $Y^{2^{11}(2^{11}-1)}$ is obtained in the second pipeline-stage, which is carried out as a right cyclic shift for 11 bits and requires conversion between tower field and normal basis representations. In the third clock cycle, the control signal sel is pulled up, which enables six multiplexers to feed correct operands to multipliers and XOR gates and the two multipliers are reused to obtain values $YY^{2^{11}(2^{11}-1)}$

Table 6: Multiplexers Outputs and States of Registers During the Two-Stage Computation of WGP-16(X^d)

Component	Control Signal sel	
	0	1
MUX$_1$	$Y^{2^{11}}$	$Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}$
MUX$_2$	Y	Y^{2^6}
MUX$_3$	Y^{-1}	Y
MUX$_4$	$Y^{2^{11}}$	$Y^{2^{11}(2^{11}-1)}$
MUX$_5$	X^{1057}	$X^{1057} \oplus_{16} Y^{2^{11}+1}$
MUX$_6$	$Y^{2^{11}+1}$	$YY^{2^{11}(2^{11}-1)} \oplus_{16}$ $Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)$
Reg$_1$	$Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}$	Not Relevant
Reg$_2$	$Y^{2^{11}(2^{11}-1)}$	Not Relevant
Reg$_3$	$X^{1057} \oplus_{16} Y^{2^{11}+1}$	WGP-16(X^d)

and $Y^{2^6}\left(Y^{2^{11}+1} \oplus_{16} Y^{2^{11}-1}\right)$. Finally, in the fourth clock cycle, the transformation to normal basis representation is done and the feedback signal WGP-16(X^d) is ready for use. When the fifth clock signal starts, the LFSR is clocked and latched with the result of the bitwise XOR of WGP-16(X^d) and the linear feedback within the LFSR.

5.2 Finite State Machine (FSM)

The FSM takes as inputs the 1-bit signals clk and rst and generates four control signals, denoted by lfsr_en, init, load, and sel. The control signals init and load determine one of the three phases at which the WG-16 hardware core stays, namely the key/IV loading phase, the initialization phase, and the running phase. The control signal lfsr_en is used to clock the LFSR to accommodate the serial computation of WGP-16(X^d). In the initialization phase, a new WGP-16(X^d) value is available once every $P+4$ clock cycles, where P equals the number of pipeline stages in moduleA. Since every WGP-16(X^d) value (except the first one) is computed from the previous WGP-16(X^d) value XORed with previous LFSR feedback, the full potential of the pipeline can not be used during the initialization phase, which results in a throughput of $\frac{16}{P+4}$ bits/clock in this phase (i.e., There are P idle cycles between two consecutive WGP-16(X^d) computations.). Two binary counters are used to keep track of

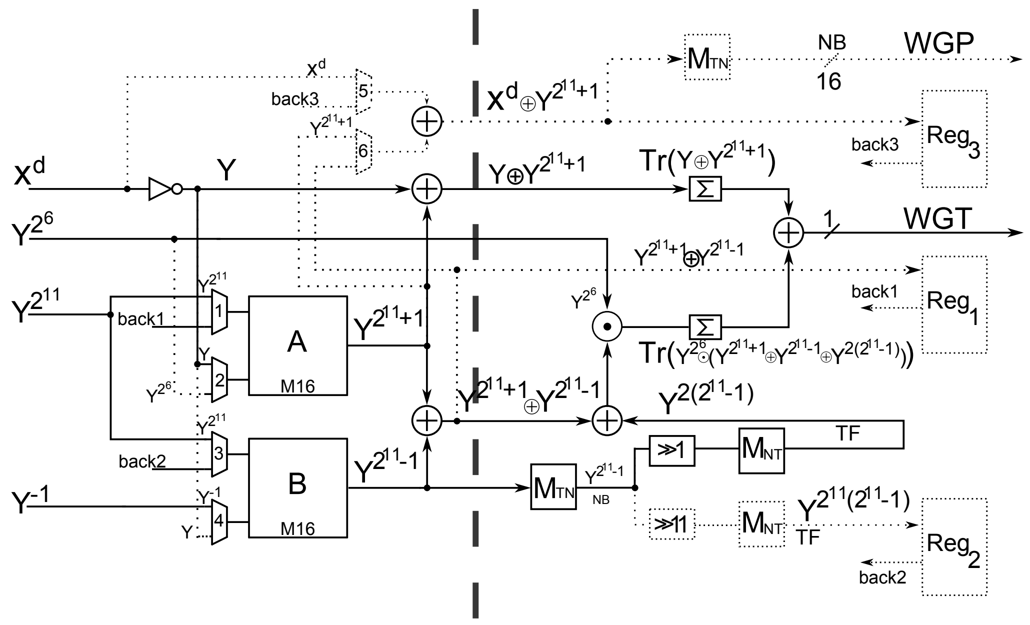


Figure 12: Submodule `module_B` with Vertical Dashed Lines Separating Two Pipeline Stages

Table 7: Implementation Results and Comparisons of Various Instances of the WG Stream Cipher Family on FPGA and ASIC Platforms

Hardware Platform	WG Cipher Instance	Area: # of FFs	Area: # of LUTs	Area: # of Slices	Speed [MHz]	Dynamic Power [mW]
FPGA	WG-29 [17]	–	6,449	–	30	380
	WG-29 [4]	–	4,044	–	34	187
	MOWG-29 [4]	–	5,512	–	35	342
	WG-16 (pipeline level M_{16}/I_8)	666	1,558	478	124	138
	WG-16 (pipeline level M_8/I_8)	725	1,478	491	116	90
		Area [GE]			Speed [MHz]	Total Power [mW]
ASIC	WG-29 [17]	33,180			144	7.28
	WG-29 [4]	19,892			169	4.45
	MOWG-29 [4]	26,261			151	5.89
	WG-16 (pipeline level M_{16}/I_8)	12,031			552	25.5
	WG-16 (pipeline level M_8/I_8)	12,352			558	25.8

the number of times LFSR has been clocked and the number idle clock cycles. The aforementioned 1-bit control signal sel is used to choose correct operands for the computation of WGP-16(X^d) (see Table 6). The running phase begins after $32 + 64(P + 4)$ clock cycles. Note that there are $P + 2$ additional idle cycles before the first bit of the keystream is available. Beyond this point, the core outputs a new keystream bit every clock cycle.

The integrated hardware architecture WGP_T, which implements both WGP-16(X^d) and WGT-16(X^d) (see Figure 9), is basically a $P+4$ stage pipeline. To reuse the two multipliers in `module_B` to compute WGP-16(X^d), we simply feed the pipeline with the same input X three times, each with appropriate value of the control signal sel, as described in the previous paragraph. In our pipelined design, we chose to pipeline at the level of M_8 since this gave us the best trade-off between clock speed, length of initialization phase, and area. Pipelining at a finer granularity (e.g., M_4) will double the length of the initialization phase with only a small increase in clock speed. Another issue with the module_B pipelining is that pipelining at a lower level does not only increase the number of pipeline stages (and hence the number of idle clock cycles), but also complicates the design of the FSM.

6. IMPLEMENTATION RESULTS AND COMPARISONS

In this section, we report the FPGA and ASIC implementation results of the proposed pipelined hardware architecture of the WG-16 stream cipher and compare our result with previous implementations of other instances of the WG stream cipher family. Our FPGA area and speed results are for Xilinx Spartan-6 FPGA device XC6SLX9 using Xilinx Synthesis Tool (XST) for synthesis and ISE for implementation [24]. All implementation results, including flip-flops, look-up tables, area (slices), speed (maximum frequency), and dynamic power consumption are obtained after post place-and-route phase and the dynamic power consumption is recorded at a frequency of 124 MHz. Moreover, we use the "-Power" option in the Mapping and Place-and-Route phases to further reduce the power consumption.

Our ASIC implementations provide area (GEs), speed (maximum frequency), and power consumption results for the 65 nm CMOS technology using Synopsys Design Compiler for synthesis [20] and Cadence SoC Encounter to complete the Place-and-Route phase. The area and speed results are obtained from the SoC Encounter's accurate area and speed reports after Place-and-Route phase. Furthermore, the dynamic power consumption is evaluated under the optimal frequency for the pipelined design. Table 7 presents the speed, area, and dynamic power consumption results for both FPGA and ASIC implementations.

The relative performance results of the two pipelined designs are similar for both FPGA and ASIC platforms as shown in Table 7. While pipelining at M_{16}/I_8 level has a shorter pipeline and needs a smaller number of clock cycles for the initialization phase, the dynamic power consumption is higher, when compared to the design pipelined at M_8/I_8 level. The actual choice of pipeline design highly depends on the requirements of practical applications.

In comparison with the large instance WG-29 [17], the stream cipher WG-16 is more efficient in terms of throughout, area, and dynamic power consumption, without decreasing the security level [14]. Therefore, the stream cipher WG-16 makes a good trade-off between security and performance among the instances of the WG stream cipher family.

7. CONCLUSION

In this paper, we propose a compact hardware architecture and its pipelined version for the stream cipher WG-16 based on the combination of efficient tower field arithmetic over $\mathbb{F}_{(((2^2)^2)^2)^2}$ as well as a newly discovered property of the trace function in tower field representation. Various formulae for performing efficient arithmetic over $\mathbb{F}_{(((2^2)^2)^2)^2}$ have been derived and low-cost basis conversion matrices have been found to conduct fast conversion of a finite field element between $\mathbb{F}_{2^{16}}$ and its isomorphic tower field $\mathbb{F}_{(((2^2)^2)^2)^2}$. Our FPGA and ASIC implementation results show that the WG-16 hardware core can achieve a throughput of 124 Mbit/s and 552 Mbit/s, at the cost of 478 slices and $12,031$ GEs in hardware and 138 mW and 25.5 mW dynamic power consumption, respectively. Based on these results, the stream cipher WG-16 is a competitive candidate for securing pervasive digital communication and computing systems.

8. REFERENCES

[1] Bluetooth Specification (core version 4.0), Bluetooth Special Interests Group, June 2010, available at `https://www.bluetooth.org/Technical/Specifications/adopted.htm`.

[2] C. Canright, "A Very Compact S-Box for AES", *The 7th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2005*, LNCS 3659, J. R. Rao and B. Sunar (Eds.), Berlin, Germany: Springer-Verlag, pp. 441-455, 2005.

[3] L. Chen and G. Gong, *Communication System Security*, Boca Raton, Florida, USA: Chapman & Hall/CRC, 2012.

[4] H. El-Razouk, A. Reyhani-Masoleh and G. Gong, "New Implementations of the WG Stream Cipher", *Centre for Applied Cryptographic Research (CACR) Technical Reports*, CACR 2012-31, available at `http://cacr.uwaterloo.ca/techreports/2012/cacr2012-31.pdf`.

[5] eSTREAM – The ECRYPT Stream Cipher Project, 2005, available at `http://www.ecrypt.eu.org/stream/`.

[6] X. Fan, T. Wu, and G. Gong, "An Efficient Stream Cipher WG-16 and Its Application for Securing 4G-LTE Networks", to appear in the proceedings of *The 3rd International Conference on Communication and Network Security - ICCNS 2013*, London, The United Kingdom, November 16 - 17, 2013.

[7] X. Fan, K. Mandal, and G. Gong, "WG-8: A Lightweight Stream Cipher for Resource-Constrained Smart Devices", *The 9th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness - Qshine 2013*, LNICTS 115, K. Singh et al. (Eds.), Berlin, Germany: Springer-Verlag, pp. 617-632, 2013.

[8] S. Gao and H. W. Lenstra Jr., "Optimal Normal Bases", *Design, Codes and Cryptography*, Vol. 2, No.

4, pp. 315-323, 1992.

[9] IEEE 802.11^{TM}-2007: IEEE Standard for Information Technology – Telecommunications and Information Exchange between Systems – Local and Metropolitan Area Networks – Specific Requirements Part11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Computer Society, June 12, 2007.

[10] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases", *Information and Computation*, 78:171-177, 1988.

[11] E. Krengel, "Fast WG Stream Cipher", *IEEE Region 8 International Conference on Computational Technologies in Electrical and Electronics Engineering - SIBIRCON 2008*, pp. 31-35, 2008.

[12] C.-H. Lam, M. D. Aagaard, and G. Gong, "Hardware Implementations of Multi-Output Welch-Gong Ciphers", *Centre for Applied Cryptographic Research (CACR) Technical Reports*, CACR 2011-01, available at http://cacr.uwaterloo.ca/techreports/2011/cacr2011-01.pdf.

[13] Y. Luo, Q. Chai, G. Gong and X. Lai, "A Lightweight Stream Cipher WG-7 for RFID Encryption and Authentication", *The Proceedings of Global Telecommunications Conference (GLOBECOM 2010)*, pp. 1-6, 2010.

[14] K. Mandal, G. Gong, X. Fan, and M. Aagaard, "On Selection of Optimal Parameters for the WG Cipher Family", *The 13th Canadian Workshop on Information Theory (CWIT'13)*, pp. 17-21, 2013.

[15] A. Menezes, *Application of Finite Fields*, Norwell, Massachusetts, USA: Kluwer Academic Publisher, 1993.

[16] Y. Nawaz and G. Gong, "The WG Stream Cipher", *eSTREAM PHASE 2 Achive*, available at http://www.ecrypt.eu.org/stream/p2ciphers/wg/wg_p2.pdf, 2005.

[17] Y. Nawaz and G. Gong, "WG: A Family of Stream Ciphers with Designed Randomness Properties", *Information Science*, vol. 178, no. 7, pp. 1903-1916, 2008.

[18] Y. Nogami, K. Nekado, T. Toyota, N. Hongo, and Y. Morikawa, "Mixed Bases for Efficient Inversion in $\mathbb{F}((2^2)^2)^2$ and Conversion Matrices of SubBytes of AES", *The 12th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2010*, LNCS 6225, S. Mangard and F.-X. Standaert (Eds.), Berlin, Germany: Springer-Verlag, pp. 234-247, 2010.

[19] K. Nohl and H. Plötz, "Mifare – Little Security Despite Obscurity", *The 24th Chaos Communication Congress - 24C3*, available at http://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html, 2007.

[20] Synopsys Design Compiler, http://www.synopsys.com/Tools/Implementation/RTLSynthesis/.

[21] The 3rd Generation Partnership Project (3GPP), "TS 35.216: Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2; Document 2: SNOW 3G specification (V10.0.0)", available at http://www.etsi.org/deliver/etsi_ts/135200_135299/135216/10.00.00_60/ts_135216v100000p.pdf, April 2011.

[22] The 3rd Generation Partnership Project (3GPP), "TS 35.222: Specification of the 3GPP Confidentiality and Integrity Algorithms EEA3 & EIA3; Document 2: ZUC specification (V11.0.1)", available at http://www.etsi.org/deliver/etsi_ts/135200_135299/135222/11.00.01_60/ts_135222v110001p.pdf, May 2012.

[23] R. Verdult, F. D. Garcia, and J. Balasch, "Gone in 360 Seconds: Hijacking with Hitag2", *The 21st USENIX Security Symposium - USENIX Security 2012*, USENIX Association, pp. 237-252, 2012.

[24] Xilinx, http://www.xilinx.com/.