“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

Abstract

Fully homomorphic encryption (FHE) allows secure operation of signal processing in the cloud. The encrypted signals that we send to the cloud can be processed without decrypting the signals. Recently, many studies have been done on different variants for the FHE construct. These variants include complex and compute-heavy computational algorithms. As an example of these algorithms; bootstrapping, forward, and reverse FFT transformations can be given. In this study, the Torus (TFHE) fast full homomorphic encryption library was analyzed on an embedded system. NTT has been added to the THFE library, which is thought to be an advantage for embedded systems. With the added NTT, the system was tested Ultra96-V2 and the results were compared. TFHE, which we propose with NTT application, performs better in multiplying ring polynomials than FFT versions of TFHE on embedded platform. Also, Thus, aims to speed up homomorphic NAND gate operation with efficient polynomial multiplication system.

1. Introduction

IoT, cloud computing and data analytics can make it possible to collect data effectively and leverage the high computing capacity of cloud computing for applications. However, in IoT applications, the data to be processed by its nature are shared with third parties. In applications with sensitive data, this can cause security concerns. Therefore, the development of these systems has been delayed due to the violation of personal private rights and the disclosure of data that should be kept confidential. As a solution to these privacy concerns, a fully homomorphic encryption scheme can be proposed. Fully Homomorphic encryption (HE) allows one to perform operations on the encrypted data without decrypting the ciphertext. This is a powerful security feature since the system does not need to share the secret key with no one else, e.g., the cloud provider. However, modern HE schemes requires computationally complex operations and these operations need to be accelerated on embedded software systems.

Homomorphic encryption systems can be used in many applications involving sensitive data. For instance, The long-term cardiac health monitoring system designed by Kocabas and Soyata can be given as an example of the application in which contain sensitive data analyzing [1]. In this system, security gaps have been prevented by using the fully homomorphic scheme. Furthermore, fully homomorphic encryption structures can be extended to many applications of sensitive data analyzing. Such as financial services, genomics, secure image analysis. Sathya et al. emphasize about these issues and their usage in real life [2]. As a result of Sathya's work the fully homomorphic

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

scheme has been described as an expensive and time-consuming process and has to be accelerated and improved for real-life application.

Fast implementations of homomorphic encryption schemes depend on efficient polynomial arithmetic; The most time-consuming operation in polynomial arithmetic is high-order polynomial multiplication. Although the potential applications of FHE are very important for sensitive data processing systems, since the vulnerability cannot be detected, it is not currently applicable in practical applications due to its algorithmic complexity and long computational processes. Arithmetic operations on ring polynomials in FHE systems are the most time-consuming operations of these systems. For this reason, there are different methods for ring polynomial multiplication operations in the literature. Although some homomorphic encryption systems based on high-order polynomials are suitable for NTT and FFT, some applications may not be suitable for NTT and FFT, at this point polynomial multiplication systems such as Toom-Cook [3] or Karatsuba [4] may be preferred. But Lattice-Based Fully Homomorphic Encryption systems are suitable for NTT and FFT-based polynomial multiplication. The TFHE full homomorphic encryption library used in this study is also based on the lattice-based encryption structure. Therefore, operations can be performed on FFT or NTT for high-order polynomial computation operations for the TFHE library. FFT is preferred in TFHE and the time complex of polynomial multiplication operation decreases from O(n^2) to O(n.logn) thanks to FFT.

Although the performance of polynomial multiplication using FFT increases, the FHE structure cannot operate in acceptable times due to its high complexity for use in applications. Therefore, there are different FFT-based polynomial multiplication applications proposed for efficient and practical lattice-based encryption systems in different platforms in the literature. These applications aimed to accelerate the system based on software [5,6,7] and hardware [8,9,10].

In this study, we present NTT-based polynomial multiplication for TFHE polynomial computation on embedded software system instead of using FFT-based polynomial multiplication on TFHE.

In addition, many studies can be found in the literature for the acceleration of NTT systems used for polynomial multiplication in lattice-based encryption algorithms in terms of hardware [11, 12, 13] and software [14, 15].

This study aims to observe the efficiency of polynomial multiplications on embedded systems and thus speed up the homomorphic NAND gate operation. Using the NTT-based polynomial multiplication method, the

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

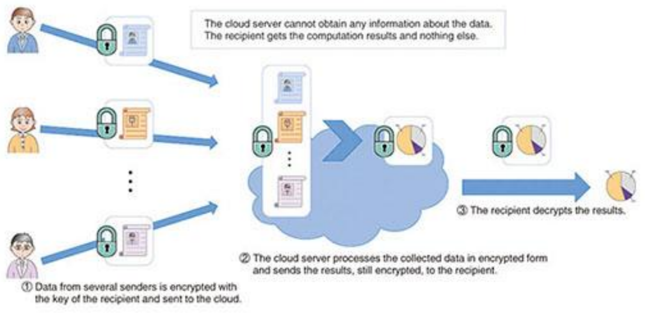
homomorphic NAND gate operation in the TFHE system on the Ultra96-V2 Board has been accelerated with software and hardware-based improvements.

2. Theoretical Background

In this section, information that forms the basis of the studies carried out within the scope of the project and that will facilitate the understanding of the processes is given.

2.1 Homomorphic Encryption (HE)

Homomorphic encryption was first introduced by Rivest, Shamir, and Adleman in 1978 as the RSA encryption system [16]. This is a well-known public key cryptography that coined the term "homomorphic feature" and provides data security by factoring the factors of two large prime numbers. The homomorphic encryption system is basically defined in 4 stages: Key generation (KeyGen), Encryption(Enc), Decryption(Dec), Evaluaiton (Eval). Basically; KeyGen is for creates a secret key sk or a keypair (a public key pk and a secret key sk). Enc is a function which encrypts a plaintext m, and outputs a ciphertext c. Dec is a function which decrypts a ciphertext c, and outputs a plaintext m. The KeyGen; Enc; Dec steps up to this stage are common across traditional encryption algorithms. However, the Eval function, another step of homomorphic encryption, is specific to the homomorphic encryption algorithm.


The cloud server cannot obtain any information about the data.
The recipient gets the computation results and nothing else.
③ The recipient decrypts the results.
② The cloud server processes the collected data in encrypted form and sends the results, still encrypted, to the recipient.
① Data from several senders is encrypted with the key of the recipient and sent to the cloud.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

Figure 1: Fully Homomorphic Encryption Schematic

An evaluation function which is Eval takes encrypted versions of two plaintexts m1 and m2 and an mathematical operation as the input and outputs an evaluated ciphertext.

$$
\begin{aligned}
Enc(m1) . Enc(m2) &= (\llbracket m1 \rrbracket ^{e} (\bmod N)) . (\llbracket m2 \rrbracket ^{e} (\bmod N)) \\
&= \llbracket (m1 .m2) \rrbracket ^{e} (\bmod N) \\
&= Enc(m1 . m2)
\end{aligned}
$$

Homomprhic Encryption can be divided into three types according to the number of operation types allowed on the ciphertext ; Partial HE, Somewhat HE, Fully HE.

Partial HE: A form of encryption which allows one specific type of computations (addition or multiplication) to be carried out on ciphertext and generate an encrypted result which, when decrypted, matches the result of operations performed on the plaintext. Although the type of transaction that can be used is limited, there is no limit to the number of transactions. The well-know Partial HE scheme is RSA , it allows only multiplication operation and does not support addition operation.

Somewhat HE: Somewhat Homomorphic Encryption (SHE) that allows both addition and multiplication operations on chiphertext, though the number of times to use operation is limited. A well-known Somewhat HE scheme is BGN which allows unlimited number of times to use addition operation, while limits by one the number of times to use multiplication operation and the size of ciphertext is constant.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

Fully HE: Fully homomorphic encryption allows any operations (essentially both addition and multiplication), and the number of times to use operation any is unlimited. It constructed based on ideal lattices and has a big computation load in memory. Fully homomorphic encryption allows any operations (essentially both addition and multiplication), and the number of times to use operation any is unlimited. It constructed based on ideal lattices and has a big computation load in memory. However, the security of FHE is create strong security guarantees. FHE has a lot of effective applications, especially in cloud environments, and therefore a variety of FHE schemes have been proposed nowadays.

Traditional encryption systems are not fully secure in a 3rd user service such as cloud servers, as the confidentiality of sensitive data must be protected. The homomorphic encryption is a special kind of encryption mechanism that can resolve the security and privacy issues. Unlike the public key encryption, which has three security procedures, i.e., key generation, encryption and decryption; there are four procedures in HE scheme, including the evaluation algorithm as shown in Fig. 1 The HE allows the third party service providers to perform certain type of operations on the user's encrypted data without decrypting the encrypted data, while maintaining the privacy of the users' encrypted data. In homomorphic encryption, if the user wants to query some information on the cloud server, he first encrypts the data and stores the encrypted data in the cloud. Then, after some time, the user sends query information to the cloud server. The cloud server runs a prediction algorithm on encrypted data using HE without knowing the contents of the encrypted data. Then, the cloud returns the encrypted prediction back to the user and the user decrypts the received encrypted data using the user's secret key, while preserving the privacy of his data as show in Fig. 1. In HE, mathematical operation on the plaintext during encryption is equivalent to another operation performed on the cipher text [17]. Let us consider a simple homomorphic operation on the plaintext with the corresponding cipher text operation.

2.2 Ring Learning with Errors (Lattice -Based)

Ring-Learning with Errors (RLWE) was first introduced by Lyubashevsky et al. [18]. This is a computationally intensive problem involving many addition and multiplication operations that is believed to be very difficult to solve even with the advancing quantum computer technology. Generally, it was called Learning with Errors over Rings because it uses polynomial rings over finite fields.

The idea behind RLWE is the vectors can be seen as polynomials modulo the nth polynomial which n is a power of 2.

Lyubashevsky et al.showed the security proof of RLWE problem is reducible to worst-case problems on ideal lattices, which is hard for polynomial-time quantum algorithms.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

With RLWE we use the coefficients of polynomials and which can be added and multiplied within a finite field (Fq) and where all the coefficients will be less than q.

Initially they decided agree on a complexity value of n, and which is the highest coefficient power to be used. Then they create q which is 2^n−1. All the polynomial operations will then be conducted with a module of q and where the largest coefficient value will be q−1.

$$\mathbf{A} = a_{n-1}x^{n-1} + \ldots + a_1 x + a_1 x^2 + a_0 \quad (1.1)$$

Next it divided by Φ(*x*), which is *x^n*+1:

$$\mathbf{A} = (a_{n-1}x^{n-1} + \cdots + a_1 x + a_1 x^2 + a_0) \div (x^n + 1) \quad (1.2)$$

Then generated an error polynomial (**e**) and a secret polynomial (**s**):

$$\mathbf{e\_A} = e_{n-1}x^{n-1} + \ldots e_2 x^2 + e_1 x^+ e_0 \pmod{q}$$
$$\mathbf{s\_A} = s_{n-1}x^{n-1} + \ldots s_2 x^2 + s_1 x^+ s_0 \pmod{q} \quad (1.3)$$

Created b_A which is a polynomial created from A, e_A and s_A:

$$\mathbf{b\_A} = A \times s_A + e_A \quad (1.4)$$

Generated his own error polynomial e’ and a secret polynomial s’:

$$\mathbf{e\_B} = e'_{n-1}x^{n-1} + \ldots e'_2 x^2 + e'_1 x^+ e'_0 \pmod{q}$$
$$\mathbf{s\_B} = s'_{n-1}x^{n-1} + \ldots s'_2 x^2 + s'_1 x^+ s'_0 \pmod{q}$$

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: **https://github.com/DHLSan/FastHomNAND**
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

2.3 TFHE

This library is a C/C++ library which implements the fastest gate-by-gate bootstrapping operations. The library allows operations to be performed on encrypted private data without providing any access. The library includes 10 different binary logic gates with negation and the Mux gate that can operate on ciphertexts. Each binary gate takes about 13 milliseconds time to homomorphic evaluation [tfhe]. Unlike other libraries, the TFHE library has no limits on the number of logic gates used or the headlamps made. As a result of the universal logic gates it contains, it can be compatible with many operations and algorithms. Also with using this universal gates it can be produce all mathematical operations theoretically.

TFHE implements a dedicated Fast Fourier Transformation and uses AVX, AVX2 and FMA assembly vectorization instructions.

FFTW3 processor is default processor in TFHE. TFHE also include the Spqlios-fma library which makes faster than the performance of the FFT. Lastly it has Project Nayuki, that includes two different applications of the fast Fourier transform - one in portable C, and the other using the AVX assembly instructions.

In this study, among the different FFT processors in the TFHE library, the Nayuki Portable processor, which was written in pure C and applicable to every platform, was selected and the result of our study is compared with this processor.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

2.3 Number Theoretic Transform (NTT)

The NTT is basically a form of Discrete Fast Fourier transformation. The NTT is a generalization of the DFT. With a lot of work, it basically lets one perform fast convolutions on integer sequences without any round-off errors. Convolutions are useful for multiplying long numbers or long polynomials, and the NTT is faster than other known methods. The objective of NTT is to multiply 2 polynomials such that the coefficient of the resultant polynomials are calculated under a particular modulo. The benefit of NTT is that all the calculations are done in integers.

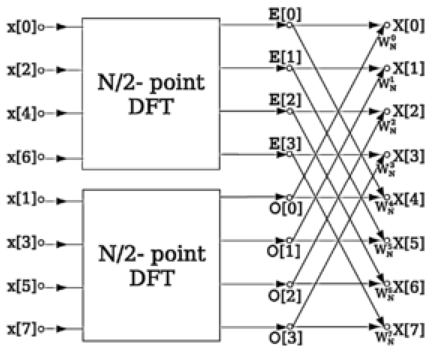
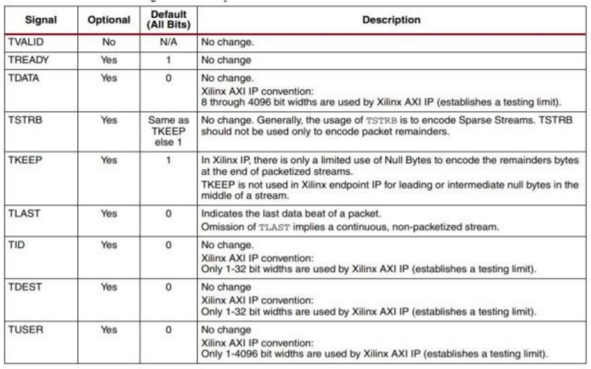


Figure 1: NTT Schematic

2.4. AXI Stream

AXI Stream is used for high speed transmissions and allows unlimited data burst size. Also, it is not memory mapped. It has single channel for transmission of data. Each AXI4-Stream act as a single unidirectional channel for a handshake data flow. In this communication protocol there are 9 signals, these are shown in Table 1.

Table 1: AXI Stream Signals and Descriptions



Signal	Optional	Default (All Bits)	Description
TVALID	No	N/A	No change.
TREADY	Yes	1	No change
TDATA	Yes	0	No change. Xilinx AXI IP convention: 8 through 4096 bit widths are used by Xilinx AXI IP (establishes a testing limit).
TSTRB	Yes	Same as TKEEP else 1	No change. Generally, the usage of TSTRB is to encode Sparse Streams. TSTRB should not be used only to encode packet remainders.
TKEEP	Yes	1	In Xilinx IP, there is only a limited use of Null Bytes to encode the remainders bytes at the end of packetized streams. TKEEP is not used in Xilinx endpoint IP for leading or intermediate null bytes in the middle of a stream.
TLAST	Yes	0	Indicates the last data beat of a packet. Omission of TLAST implies a continuous, non-packetized stream.
TID	Yes	0	No change. Xilinx AXI IP convention: Only 1-32 bit widths are used by Xilinx AXI IP (establishes a testing limit).
TDEST	Yes	0	No change Xilinx AXI IP convention: Only 1-32 bit widths are used by Xilinx AXI IP (establishes a testing limit).
TUSER	Yes	0	No change Xilinx AXI IP convention: Only 1-4096 bit widths are used by Xilinx AXI IP (establishes a testing limit).

It is seen that some of them are required and some of them are optional. Since some of them are optional we used the TVALID, TREADY, TSTRB and TLAST, TDATA channels. TVALID is a master sourced signal, it indicates that the master is driving a valid transfer. TREADY indicates that the slave can accept a transfer in the current cycle. TDATA includes the data to be transmit or received. TLAST signals the last byte of the stream. TSTRB is the byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte [19].

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

There is an handshaking process in AXI Stream. The following figure shows the basics of it;

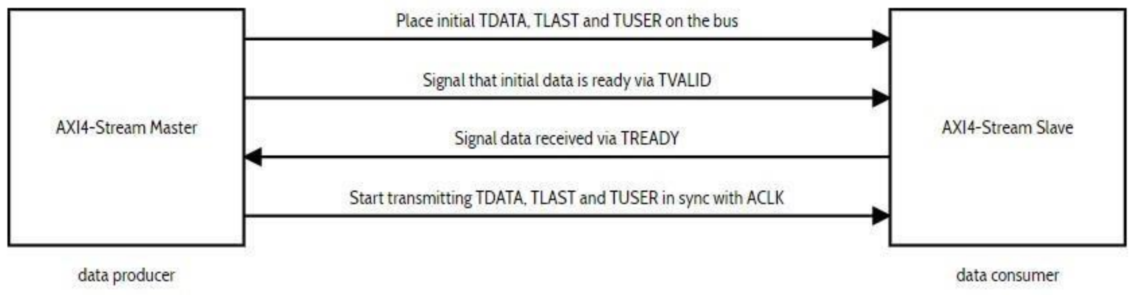

Place initial TDATA, TLAST and TUSER on the bus
Signal that initial data is ready via TVALID
AXI4-Stream Master
Signal data received via TREADY
AXI4-Stream Slave
Start transmitting TDATA, TLAST and TUSER in sync with ACLK
data producer
data consumer

Figure 2: AXI Stream Handshaking Process

For a transfer to take place both TVALID and TREADY must have asserted. The transfer starts when master sends TVALID and gets response with TREADY from slave. Once TVALID is asserted, it must remain asserted until the handshake occurs. But if slave asserts TREADY before TVALID is asserted, it can reassert the signal.

3. Method

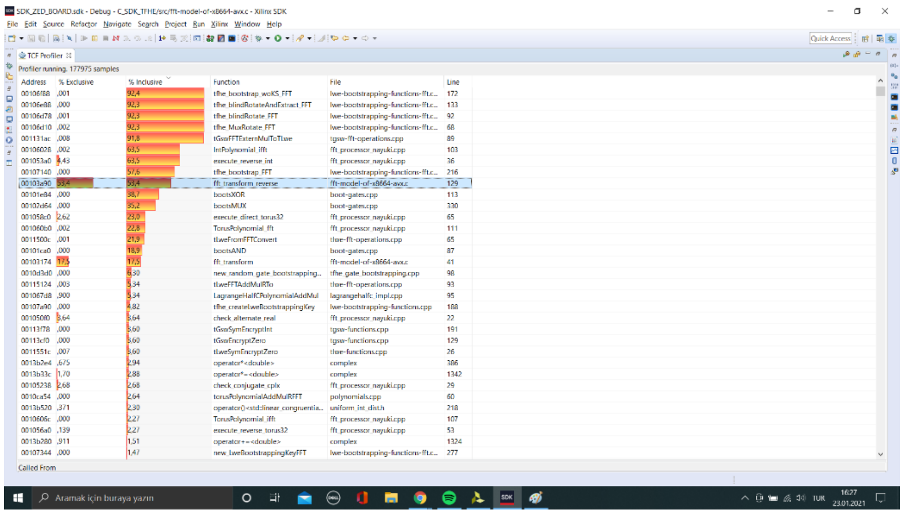
3.1 Analyzing TFHE with TCF Profiling

At the out set, although we know that the most time-consuming function in the Tfhe system is fft transformations, we have identified the most time-consuming function of the TFHE system by utilizing the software profiling system which name is TCF Profiling process provided by Xilinx. As seen in Figure 3, the fft_transform_reverse and fft_transform operations under the Exclusive title are the most time-consuming functions when applying the homomorphic NAND gate operation in the TFHE system while operating on ULTRA96-V2 (Arm). As seen in Figure 3, the fft_transform_reverse and fft_transform operations under the

Exclusive title are the most time-consuming functions when applying the homomorphic NAND gate operation in the TFHE system. Homomorphic NAND gate operation was implemented on the ARM Cortex A5, the processor of the ULTRA96-V2 development board, and it was determined that the most time-consuming functions are FFT based without any development.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s


SDK_ZED_BOARD.sdk - Debug - C_SDK_TFHE/src/fft-model-of-x8664-avx.c - Xilinx SDK

TCF Profiler

Profiler running. 177975 samples

Address	% Exclusive	% Inclusive	Function	File	Line
00106f88	,001	92,4	tfhe_bootstrap_woKS_FFT	lwe-bootstrapping-functions-fft.c...	172
00106e88	,000	92,3	tfhe_blindRotateAndExtract_FFT	lwe-bootstrapping-functions-fft.c...	133
00106d78	,001	92,3	tfhe_blindRotate_FFT	lwe-bootstrapping-functions-fft.c...	92
00106d10	,002	92,3	tfhe_MuxRotate_FFT	lwe-bootstrapping-functions-fft.c...	68
001131ac	,008	91,8	tGswFFTExternMulToTLwe	tgsw-fft-operations.cpp	89
00106028	,002	63,5	IntPolynomial_ifft	fft_processor_nayuki.cpp	103
001053a0	4,43	63,5	execute_reverse_int	fft_processor_nayuki.cpp	36
00107140	,000	57,6	tfhe_bootstrap_FFT	lwe-bootstrapping-functions-fft.c...	216
00103a90	53,4	53,4	fft_transform_reverse	fft-model-of-x8664-avx.c	129
00101e84	,000	38,7	bootsXOR	boot-gates.cpp	113
00102d64	,000	35,2	bootsMUX	boot-gates.cpp	330
001058c0	2,62	23,0	execute_direct_torus32	fft_processor_nayuki.cpp	65
001060b0	,002	22,8	TorusPolynomial_fft	fft_processor_nayuki.cpp	111
0011500c	,001	21,9	tLweFromFFTConvert	tlwe-fft-operations.cpp	65
00101ca0	,000	18,9	bootsAND	boot-gates.cpp	87
00103174	17,5	17,5	fft_transform	fft-model-of-x8664-avx.c	41
0010d3d0	,000	6,30	new_random_gate_bootstrapping...	tfhe_gate_bootstrapping.cpp	98
00115124	,003	5,34	tLweFFTAddMulRTo	tlwe-fft-operations.cpp	93
001067d8	,900	5,34	LagrangeHalfCPolynomialAddMul	lagrangehalfc_impl.cpp	95
00107a90	,000	4,82	tfhe_createLweBootstrappingKey	lwe-bootstrapping-functions.cpp	188
001050f0	3,64	3,64	check_alternate_real	fft_processor_nayuki.cpp	22
00113f78	,000	3,60	tGswSymEncryptInt	tgsw-functions.cpp	191
00113cf0	,000	3,60	tGswEncryptZero	tgsw-functions.cpp	129
0011551c	,007	3,60	tLweSymEncryptZero	tlwe-functions.cpp	26
0013b2e4	,675	2,94	operator*<double>	complex	386
0013b33c	1,70	2,88	operator*=<double>	complex	1342
00105238	2,68	2,68	check_conjugate_cplx	fft_processor_nayuki.cpp	29
0010ca54	,000	2,64	torusPolynomialAddMulRFFT	polynomials.cpp	60
0013b520	,371	2,30	operator()<std::linear_congruentia...	uniform_int_dist.h	218
0010606c	,000	2,27	TorusPolynomial_ifft	fft_processor_nayuki.cpp	107
001056a0	,139	2,27	execute_reverse_torus32	fft_processor_nayuki.cpp	53
0013b280	,911	1,51	operator+=<double>	complex	1324
00107344	,000	1,47	new_LweBootstrappingKeyFFT	lwe-bootstrapping-functions-fft.c...	277

Called From

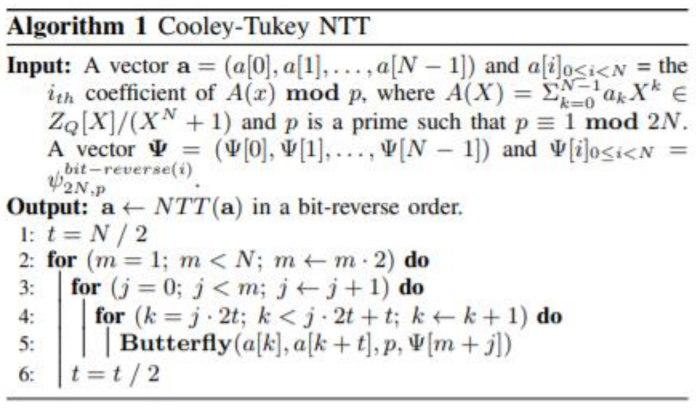
Figure 3: Software TCF Profiling for Homomorphic NAND gate on ULTRA96-V2

3.2 NTT implementation TFHE

Encrypted data in TFHE is stored in a ring polynomial ring ∈ ZQ[X]/(XN + 1). While the TFHE library uses the FFT system to multiply these polynomials, NTT is integrated instead of this system in our project. In this way, it is aimed to operate in the integer domain by getting rid of the burden of processing in the complex domain. Since the data is stored as 64-bit unsigned integer in TFHE, the modulus value selected for the NTT system is selected as 0xffffffff00000001, which is one of the special Mersenne primes that will help speed up the 64-bit long system.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

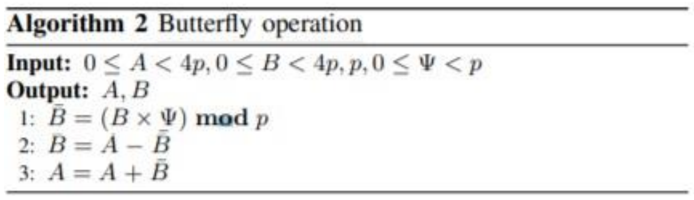
University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s



```
Algorithm 1 Cooley-Tukey NTT
Input: A vector $\mathbf{a} = (a[0], a[1], \ldots, a[N-1])$ and $a[i]_{0 \le i < N}$ = the
    $i_{th}$ coefficient of $A(x)$ mod $p$, where $A(X) = \Sigma_{k=0}^{N-1} a_k X^k \in$
    $Z_Q[X]/(X^N + 1)$ and $p$ is a prime such that $p \equiv 1$ mod $2N$.
    A vector $\mathbf{\Psi} = (\Psi[0], \Psi[1], \ldots, \Psi[N-1])$ and $\Psi[i]_{0 \le i < N} =$
    $\psi_{2N,p}^{bit-reverse(i)}$.
Output: $\mathbf{a} \leftarrow NTT(\mathbf{a})$ in a bit-reverse order.
1: $t = N / 2$
2: for $(m = 1;\ m < N;\ m \leftarrow m \cdot 2)$ do
3:     for $(j = 0;\ j < m;\ j \leftarrow j + 1)$ do
4:         for $(k = j \cdot 2t;\ k < j \cdot 2t + t;\ k \leftarrow k + 1)$ do
5:             Butterfly$(a[k], a[k+t], p, \Psi[m+j])$
6:     $t = t / 2$
```

Figure 4: Pseudo-Code of NTT Cooley-Tukey Algorithm

The Cooley Tukey algorithm, which can be examined in Figure 4, is integrated into the TFHE system. The algorithm iteratively divides an N-point DFT into k-interspersed N/k-points NTT. Depending on the divisor k, the algorithm is called radix-k NTT, and each division is called a step; the number of stages is logk(N). In TFHE, the encrypted texts is represent with polynomials and the polynomials are defines with 1024 coefficient. For this reason, a 10-stage NTT transformation process is applied in the TFHE system.



```
Algorithm 2 Butterfly operation
Input: $0 \le A < 4p, 0 \le B < 4p, p, 0 \le \Psi < p$
Output: $A, B$
1: $\bar{B} = (B \times \Psi)$ mod $p$
2: $B = A - \bar{B}$
3: $A = A + \bar{B}$
```

Figure 5: Pseudo-Code of butterfly function in NTT Cooley-Tukey Algorithm

The butterfly operation in line 5 of algorithm 1 can be examined in figure 4. As can be seen, although the Cooley-Tukey algorithm is one of the most effective methods for polynomial multiplication, it is based on many modular arithmetic operations. At this point, 0xffffffff00000001 Mersenne Prime I, which is specific to the data type of your system, has been selected.

3.2.1 Mersenne Prime for ARM Instruction Set

ARM prime works using the 64-bit prime number 2^64-2^32+1 arithmetic module. With this special number it is possible to calculate x mod p more easily. By getting rid of divide with this special prime, since there is no splitting in the arm instruction set. It becomes possible to design a customized modulation system for the arm processor. The special form p means we can make a mod p routine with just a few scrolling and arithmetic operations.

2^64 == 2^32 -1 mod p

2^96 == -1 mod p

2^128 == -2^32 mod p

2^192 == 1 mod p

2^n * 2^(192-n) = 1 mod p

Thus :

x3 * 2^96 + x2 * 2^64 + x1 * 2^32 + x0 mod p

= x2 * 2^64 + x1*2^32 + x0-x3) [using 2^96 mod p = -1]

= (x1+x2) * 2^32 + x0 - x3 - x2 [using 2^64 mod p = 2^32 -1]

3.2.2 Special Modular Multiplication Designed with Mersenne Prime for ARM Processors

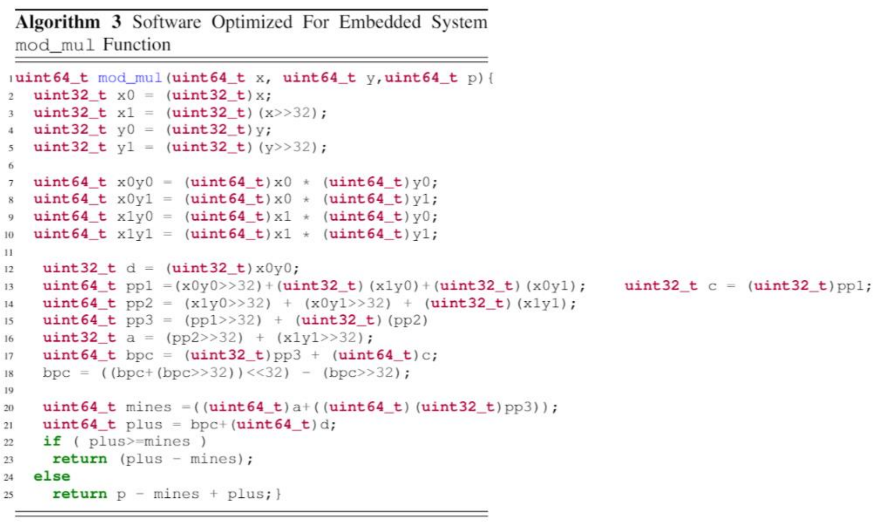
“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

As can be seen in Figure 4, the Cooley-Tukey algorithm is based on modular multiplication. Instead of the modular addition and subtraction in Figure 5, modular multiplication is the most time-consuming function compared to other operations. This is the most time-consuming function of modular multiplication in the butterfly algorithm repeated in NTT.

Thanks to the system-specific prime, the modular multiplication process is avoided by dividing instruction which is much more time-consuming instruction then others. We designed it in Figure 6 and integrated it into the system. In the modular multiplication we designed, we performed the modulation process by applying simple shifting, addition and subtraction operations without applying the division operation.

The mod_mul function in Figure 6 is designed for 32-bit processor systems, as it runs on the Zed Board Development Board in the early stages of the project. In the ongoing phases of the project, work continued on the Ultra96-V2 and a faster mod_mul function was obtained by replacing lines 1-10 with a 64-bit multiplication. This algorithm has been shared with you, as applicability to all kinds of platforms is our first goal.



Algorithm 3 Software Optimized For Embedded System `mod_mul` Function

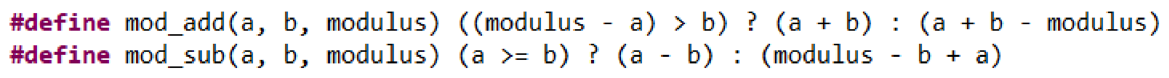
```
1  uint64_t mod_mul(uint64_t x, uint64_t y,uint64_t p){
2   uint32_t x0 = (uint32_t)x;
3   uint32_t x1 = (uint32_t)(x>>32);
4   uint32_t y0 = (uint32_t)y;
5   uint32_t y1 = (uint32_t)(y>>32);
6
7   uint64_t x0y0 = (uint64_t)x0 * (uint64_t)y0;
8   uint64_t x0y1 = (uint64_t)x0 * (uint64_t)y1;
9   uint64_t x1y0 = (uint64_t)x1 * (uint64_t)y0;
10  uint64_t x1y1 = (uint64_t)x1 * (uint64_t)y1;
11
12   uint32_t d = (uint32_t)x0y0;
13   uint64_t pp1 =(x0y0>>32)+(uint32_t)(x1y0)+(uint32_t)(x0y1);     uint32_t c = (uint32_t)pp1;
14   uint64_t pp2 = (x1y0>>32) + (x0y1>>32) + (uint32_t)(x1y1);
15   uint64_t pp3 = (pp1>>32) + (uint32_t)(pp2)
16   uint32_t a = (pp2>>32) + (x1y1>>32);
17   uint64_t bpc = (uint32_t)pp3 + (uint64_t)c;
18   bpc = ((bpc+(bpc>>32))<<32) - (bpc>>32);
19
20   uint64_t mines =((uint64_t)a+((uint64_t)(uint32_t)pp3));
21   uint64_t plus = bpc+(uint64_t)d;
22   if ( plus>=mines )
23    return (plus - mines);
24  else
25    return p - mines + plus;}
```

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

Figure 6: Software Optimized For Embedded System mod_mul Function

3.2.3 Little trick with burden of function calling mod add and mod mull

Macro directives are defined for mod_add and mod_sub operations in the Butterfly algorithm. These directives can be viewed in figure 7.



```
#define mod_add(a, b, modulus) ((modulus - a) > b) ? (a + b) : (a + b - modulus)
#define mod_sub(a, b, modulus) (a >= b) ? (a - b) : (modulus - b + a)
```

Figure 7: Macro directives for Mod_add and Mod_sub

Thanks to the function definitions mentioned above, the if condition used in the C language has been defined in the most effective way, and thanks to the macro directives, the system has been relieved of the burden of calling the function and acceleration has been achieved.

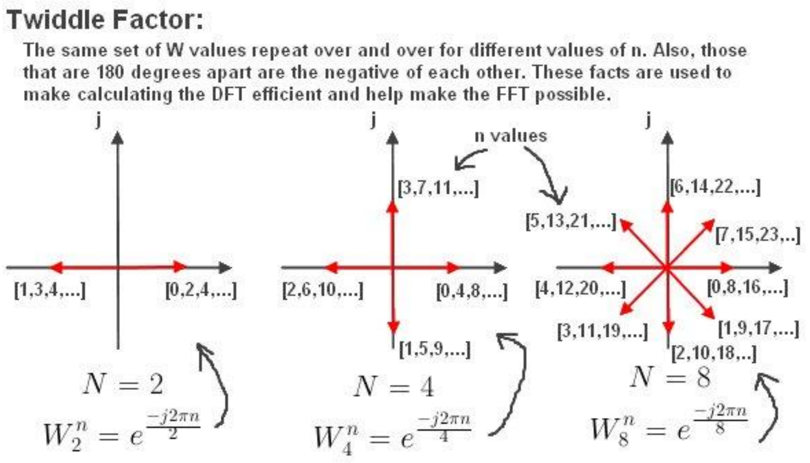
3.2.4 Off-load computation of Twilled Factors

The twiddle factors used during the butterfly operations are stored in a precomputed table, Ψ. The number of twiddle factors required doubles at each stage. Merging the powers of the ψ2N,p term in the negacyclic convolution using NTT/INTT with the powers of ψN,p in NTT is also possible in the Cooley-Tukey algorithm [27]. The difference from NTT without negacyclic convolution is that it uses the powers of the 2Nth root of unity for twiddle factors (Ψ in Algo. 1), which are stored in a bit-reverse order: Ψ[j] = ψ bit−reverse(j) 2N,p Because the Cooley-Tukey algorithm produces output data in a bit-reverse order, the input data to the algorithm requires bit-reversal permutation (bit-reversing) prior to or after NTT to reorder the output values.

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

In this study, twiddle factors that are calculated specifically for the selected prime and data length and polynomial coefficient are stored in local arrays, thus avoiding the burden of calculating these values and reducing the time spent on NTT and INTT.

By examining the twiddle factor schematic exemplified in Figure 5, it can be understood that the calculation method changes according to the polynomial coefficient.



Twiddle Factor:

The same set of W values repeat over and over for different values of n. Also, those that are 180 degrees apart are the negative of each other. These facts are used to make calculating the DFT efficient and help make the FFT possible.

Figure 8: Schematic for Twiddle Factor

3.3 Hardware Implementation NTT for TFHE

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

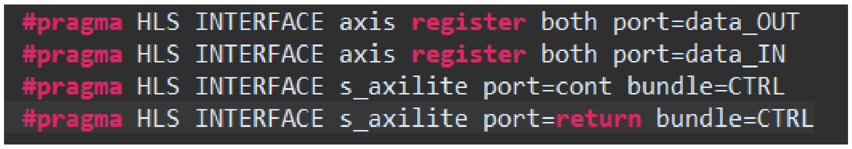
University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

In this section, the NTT application created based on HLS is explained.

3.3.1 HLS Implementation

Experimental Setup Implementation of experiment part of this study can be separated into three main parts: (1) HLS configuration that is used to generate all reconfigurable IP cores and control unit, (2) Vivado configuration that is used to create SoC hardware designs and (3) Vitis configuration that is used to verification of software part of SoC design and to communicate data transfer between hardware and software.

In this study, the HLS implementation is used to create and package the custom. The NTT algorithm is implement on the Vitis HLS 2020.2 tool.



```
#pragma HLS INTERFACE axis register both port=data_OUT
#pragma HLS INTERFACE axis register both port=data_IN
#pragma HLS INTERFACE s_axilite port=cont bundle=CTRL
#pragma HLS INTERFACE s_axilite port=return bundle=CTRL
```

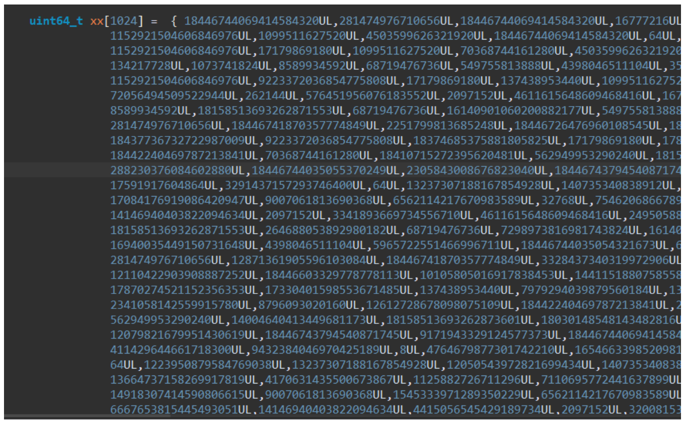
Figure 9: HLS Pragmas for Comminication

The communication protocol of the input and output ports of the custom IP produced with the first two pragmas that can be examined in the figure 9 is defined. The reason why Custom IP is defined as communication protocol AXI Stream is due to the high data transfer load of NTT operating on polynomial arrays with 1024 coefficients. Arrays containing big data, which are transferred only thanks to the data transfer logic between the memories without creating a load on the processor provided by AXI-Stream, have been successfully transmitted to AXI-DMA thanks to these first two pragms.

Thanks to the last two pragmas indicated in the figure 9, a control mechanism has been produced that communicates with the AXI-Lite communication protocol, which ensures the controllability of the custom IP produced by the processor.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s



```
uint64_t xx[1024] =  { 18446744069414584320UL,281474976710656UL,18446744069414584320UL,16777216UL
            1152921504606846976UL,1099511627520UL,4503599626321920UL,18446744069414584320UL,64UL,
            1152921504606846976UL,17179869180UL,1099511627520UL,70368744161280UL,450359962632192
            134217728UL,1073741824UL,8589934592UL,68719476736UL,549755813888UL,4398046511104UL,35
            1152921504606846976UL,9223372036854775808UL,17179869180UL,137438953440UL,10995116275
            72056494509522944UL,262144UL,576451956076183552UL,2097152UL,4611615648609468416UL,167
            8589934592UL,18158513693262871553UL,68719476736UL,16140901060200882177UL,54975581388
            281474976710656UL,18446741870357774849UL,2251799813685248UL,18446726476960108545UL,18
            18437736732722987009UL,9223372036854775808UL,18374685375881805825UL,17179869180UL,178
            18442240469787213841UL,70368744161280UL,18410715272395620481UL,562949953290240UL,1815
            2882303760846028880UL,18446744035055370249UL,2305843008676823040UL,1844674379454087174
            17591917604864UL,3291437157293746400UL,64UL,13237307188167854928UL,140735340838912UL,
            17084176919086420947UL,9007061813690368UL,6562114217670983589UL,32768UL,754620686678
            14146940403822094634UL,2097152UL,3341893669734556710UL,4611615648609468416UL,24950588
            18158513693262871553UL,2646880538929801820UL,68719476736UL,7298973816981743824UL,16140
            16940035449150731648UL,4398046511104UL,5965722551466996711UL,18446744035054321673UL,6
            281474976710656UL,12871361905596103084UL,18446741870357774849UL,3328437340319972906UL
            12110422903908887252UL,18446603329778778113UL,10105805016917838453UL,144115188075855
            17870274521152356353UL,17330401598553671485UL,137438953440UL,7979294039879560184UL,13
            2341058142559915780UL,8796093020160UL,12612728678098075109UL,18442240469787213841UL,2
            562949953290240UL,14004640413449681173UL,18158513693262873601UL,18030148548143482816U
            12079821679951430619UL,18446743794540871745UL,9171943329124577373UL,1844674406941458
            4114296446617183000UL,9432384046970425189UL,8UL,4764679877301742210UL,165466339852098
            64UL,12239508795847690380UL,13237307188167854928UL,12050543972821699434UL,1407353408
            13664737158269917819UL,4170631435500673867UL,1125882726711296UL,7110695772441637899UL
            14918307414590806615UL,9007061813690368UL,15453339712893502290UL,6562114217670983589UL
            6667653815445493051UL,14146940403822094634UL,4415056545429189734UL,2097152UL,32008153
```

Figure 10: Off load Calculated Twiddle NTT factors

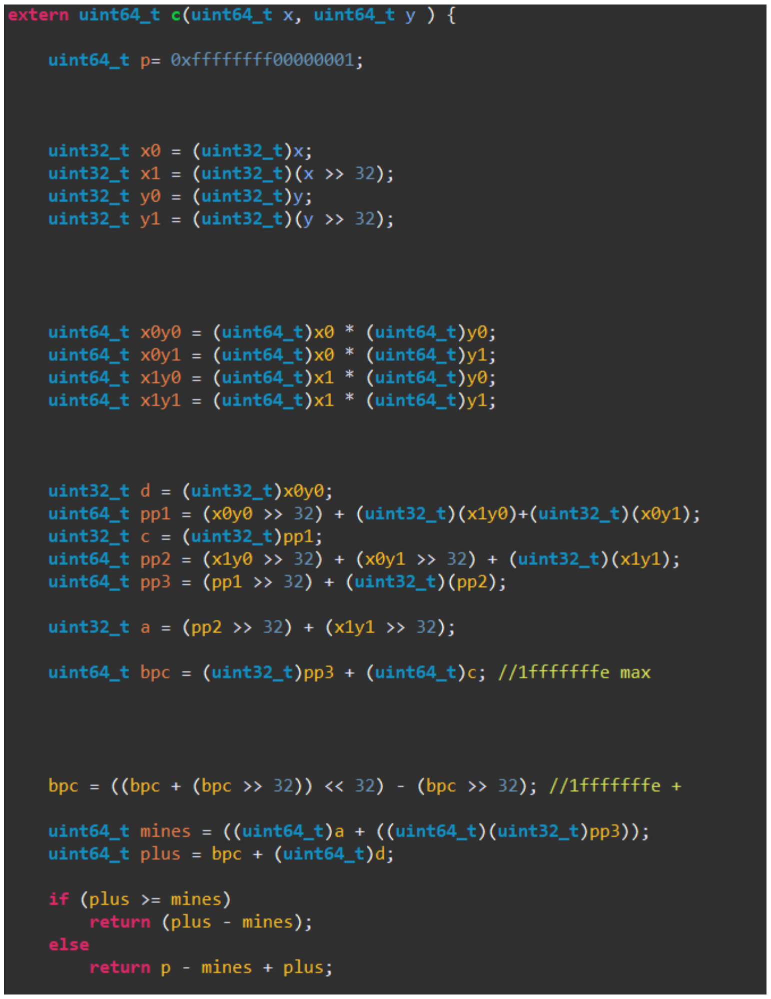
The uint64_t xx [1024] array specified in Figure 10 specifies the Twiddle factors for the previously calculated NTT. This large array is embedded in the IP so that the transfer load of this array is also blocked. By dividing the array into smaller pieces such as #pragma HLS array_partition variable=array_x cyclic/block factor=4 for this array, the memory space can be reduced and thus more effective results can be produced. However, this pragma was not used in this project due to resource throttling in ULTRA96-V2.

3.3.2 NTT Which Is Successfully Performed on HLS

The modular multiplication operation described in Section 3.2.2 involves many shifting and addition operations. Shift operations can produce more effective results at the hardware level. Therefore, the mod_mul function defined on HLS is designed as a sub-function for ntt_CT with the name C. NTT operation to this sub-function specified in the source file in Vitis HLS goes to the child. The mathematical approach of this function is detailed in section 3.2.1.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

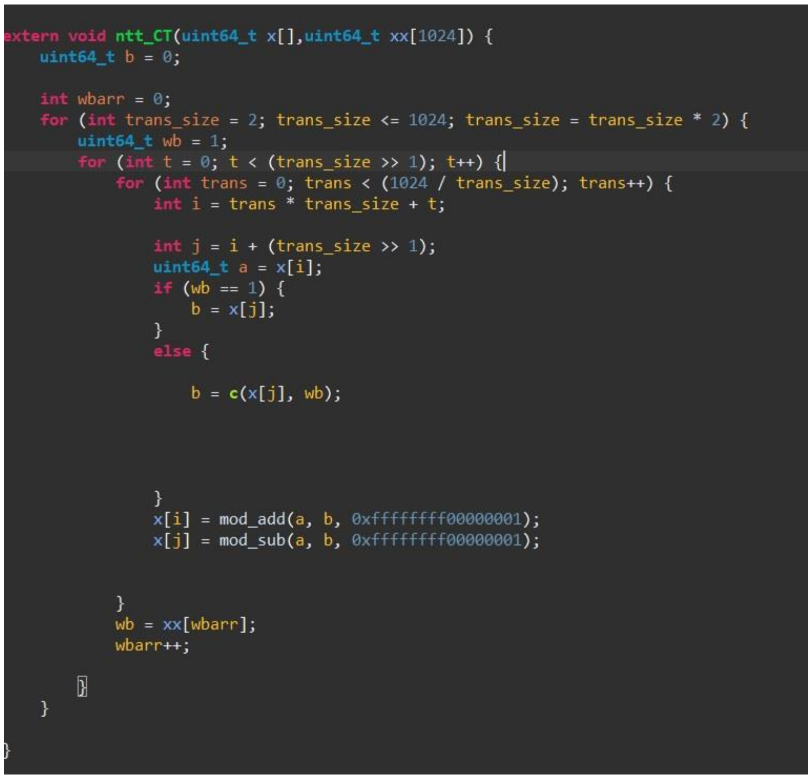
University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s



```
extern uint64_t c(uint64_t x, uint64_t y ) {

    uint64_t p= 0xffffffff00000001;



    uint32_t x0 = (uint32_t)x;
    uint32_t x1 = (uint32_t)(x >> 32);
    uint32_t y0 = (uint32_t)y;
    uint32_t y1 = (uint32_t)(y >> 32);




    uint64_t x0y0 = (uint64_t)x0 * (uint64_t)y0;
    uint64_t x0y1 = (uint64_t)x0 * (uint64_t)y1;
    uint64_t x1y0 = (uint64_t)x1 * (uint64_t)y0;
    uint64_t x1y1 = (uint64_t)x1 * (uint64_t)y1;



    uint32_t d = (uint32_t)x0y0;
    uint64_t pp1 = (x0y0 >> 32) + (uint32_t)(x1y0)+(uint32_t)(x0y1);
    uint32_t c = (uint32_t)pp1;
    uint64_t pp2 = (x1y0 >> 32) + (x0y1 >> 32) + (uint32_t)(x1y1);
    uint64_t pp3 = (pp1 >> 32) + (uint32_t)(pp2);

    uint32_t a = (pp2 >> 32) + (x1y1 >> 32);

    uint64_t bpc = (uint32_t)pp3 + (uint64_t)c; //1fffffffe max



    bpc = ((bpc + (bpc >> 32)) << 32) - (bpc >> 32); //1fffffffe +

    uint64_t mines = ((uint64_t)a + ((uint64_t)(uint32_t)pp3));
    uint64_t plus = bpc + (uint64_t)d;

    if (plus >= mines)
        return (plus - mines);
    else
        return p - mines + plus;
```

Figure 11: Mod_Mul (C) in Vitis HLS

The loop boundaries of the ntt_CT function indicated in Figure 12 could not be determined by vivado HLS. Therefore, hardware advantage is provided only at the modular multiply level in this cycle. Since the polynomials found in TFHE have a coefficient of 1024, there is a 10-stage NTT application. Trans_size value takes 2,4,8,16,32,64,128,256,512,1024 values and performs 10-stage operation. The remaining two loops inside take variable values depending on the number of stages, but in overall the inner two loops run 512 times.



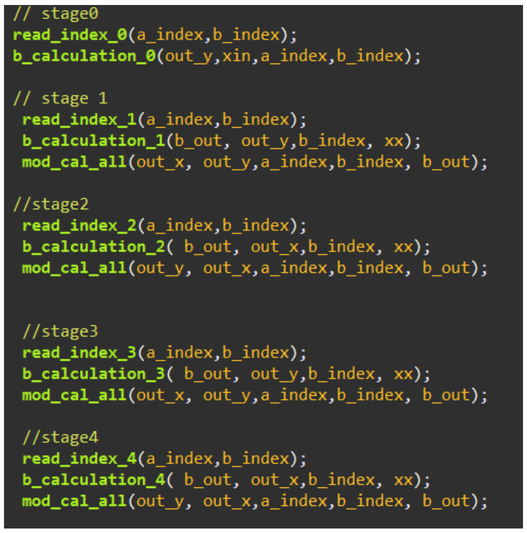
```
extern void ntt_CT(uint64_t x[],uint64_t xx[1024]) {
    uint64_t b = 0;

    int wbarr = 0;
    for (int trans_size = 2; trans_size <= 1024; trans_size = trans_size * 2) {
        uint64_t wb = 1;
        for (int t = 0; t < (trans_size >> 1); t++) {
            for (int trans = 0; trans < (1024 / trans_size); trans++) {
                int i = trans * trans_size + t;

                int j = i + (trans_size >> 1);
                uint64_t a = x[i];
                if (wb == 1) {
                    b = x[j];
                }
                else {

                    b = c(x[j], wb);




                }
                x[i] = mod_add(a, b, 0xffffffff00000001);
                x[j] = mod_sub(a, b, 0xffffffff00000001);


            }
            wb = xx[wbarr];
            wbarr++;

        }
    }

}
```

Figure 12: The main Sub Function Ntt_CT

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

3.3.3 NTT Failed Due to Ultra96's Lack of Resources

The most important part of our project that is promising and open to improvement is explained in this section.



```
// stage0
read_index_0(a_index,b_index);
b_calculation_0(out_y,xin,a_index,b_index);

// stage 1
 read_index_1(a_index,b_index);
 b_calculation_1(b_out, out_y,b_index, xx);
 mod_cal_all(out_x, out_y,a_index,b_index, b_out);

//stage2
 read_index_2(a_index,b_index);
 b_calculation_2( b_out, out_x,b_index, xx);
 mod_cal_all(out_y, out_x,a_index,b_index, b_out);


 //stage3
 read_index_3(a_index,b_index);
 b_calculation_3( b_out, out_y,b_index, xx);
 mod_cal_all(out_x, out_y,a_index,b_index, b_out);

 //stage4
 read_index_4(a_index,b_index);
 b_calculation_4( b_out, out_x,b_index, xx);
 mod_cal_all(out_y, out_x,a_index,b_index, b_out);
```

Figure 11: Splitting NTT Stage

NTT stages indicated in Figure 11 are reserved. The reason for this process is that HLS vitis application needs to know the limits of these loops in order to parallelize the for loops and apply the default PIPLINE operation. The NTT designed in this application has 10 stages and each stage has its own unique number of loop variable. For this reason, stage special cycle numbers belonging to 10 stages are determined for 10 different functions, and each stage is divided into 3 separate stages as reading index, calculation, and modular calculation.

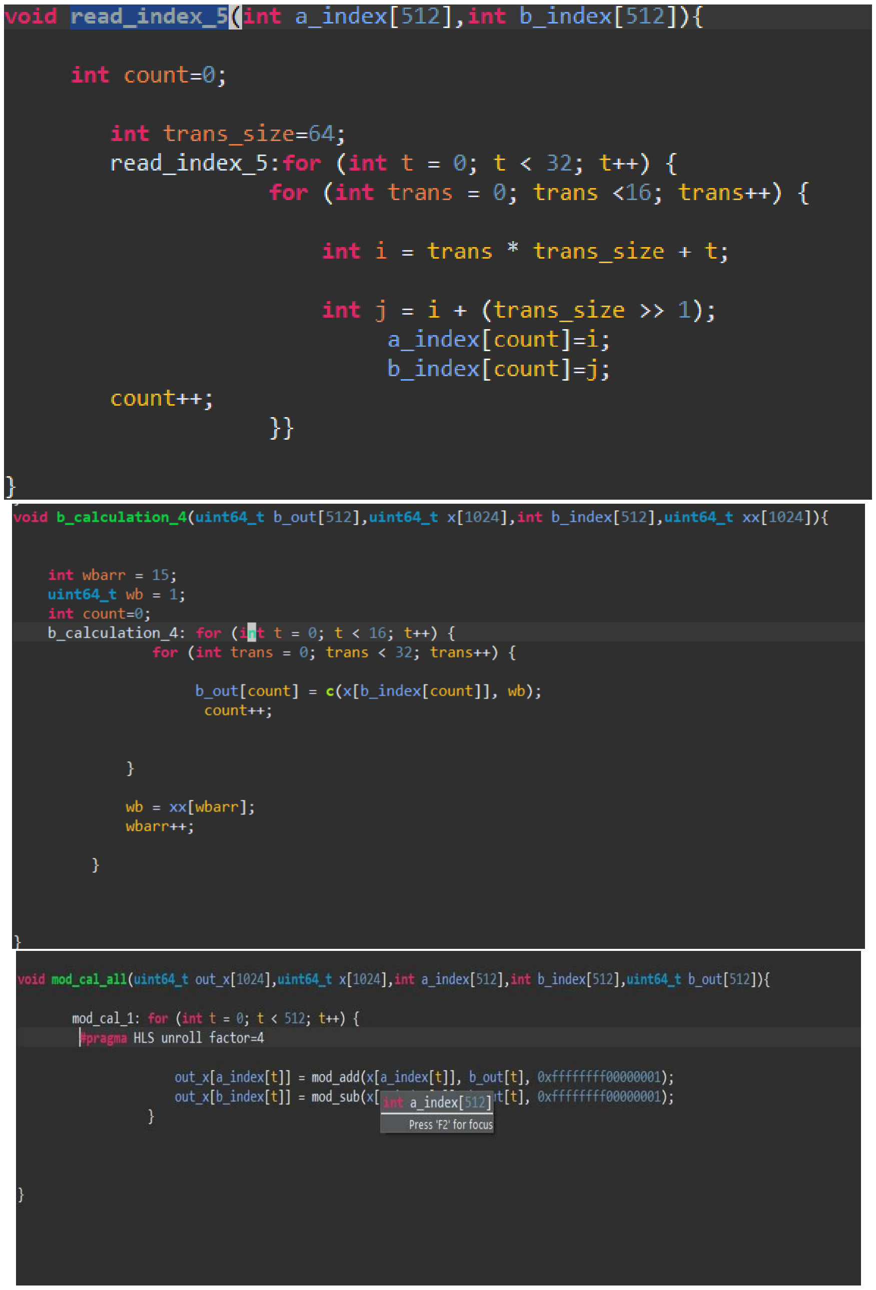
In this way, the processing times of the functions can be reduced simply by defining the necessary pragmas in these functions on vitis HLS.

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

By applying the #pragma HLS UNROLL function to the read index function in Figure 12, the loop can be split into different parts at the same time. This process has become applicable by calculating the loop values in NTT stages. This process can also be repeated in the modular computation and computation phases. In this way, it can be predicted that the NTT processing time will decrease significantly. In the ongoing stages of our project, it has been decided to work on development cards with higher computational capacity. In this context, the laboratory in the school will be utilized or work will continue on the Development Cards rented on the Cloud with the support received.

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s



```
void read_index_5(int a_index[512],int b_index[512]){

    int count=0;

      int trans_size=64;
      read_index_5:for (int t = 0; t < 32; t++) {
                for (int trans = 0; trans <16; trans++) {

                   int i = trans * trans_size + t;

                   int j = i + (trans_size >> 1);
                       a_index[count]=i;
                       b_index[count]=j;
      count++;
                }}

}
```

```
void b_calculation_4(uint64_t b_out[512],uint64_t x[1024],int b_index[512],uint64_t xx[1024]){


    int wbarr = 15;
    uint64_t wb = 1;
    int count=0;
    b_calculation_4: for (int t = 0; t < 16; t++) {
                for (int trans = 0; trans < 32; trans++) {

                    b_out[count] = c(x[b_index[count]], wb);
                     count++;


            }

            wb = xx[wbarr];
            wbarr++;

        }


}
```

```
void mod_cal_all(uint64_t out_x[1024],uint64_t x[1024],int a_index[512],int b_index[512],uint64_t b_out[512]){

        mod_cal_1: for (int t = 0; t < 512; t++) {
         #pragma HLS unroll factor=4

                    out_x[a_index[t]] = mod_add(x[a_index[t]], b_out[t], 0xffffffff00000001);
                    out_x[b_index[t]] = mod_sub(x[                 t[t], 0xffffffff00000001);
                }



}
```

Figure 12: Splitting NTT Stage inner Funcitons

“**FastHNAND: Hardware Acceleration of Homomorphic NAND Gate**

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

3.3.2 Vivado 2020.2 Implementation

The custom IP developed on Vitis HLS was integrated into the system in the Vivado Design indicated in Figure 13. The data_IN and data_Out ports specify uint64_t type polynomial coefficients entering and exiting the IP. The communication type of these ports is AXI-Stream. In this way, it has become operable with the AXI DMA IP provided by Xilinx. The data_in port communicates with the memory to stream port, while the data_out port communicates with the stream to memory port. Polynomial coefficients are successfully obtained from the memory on the development board.

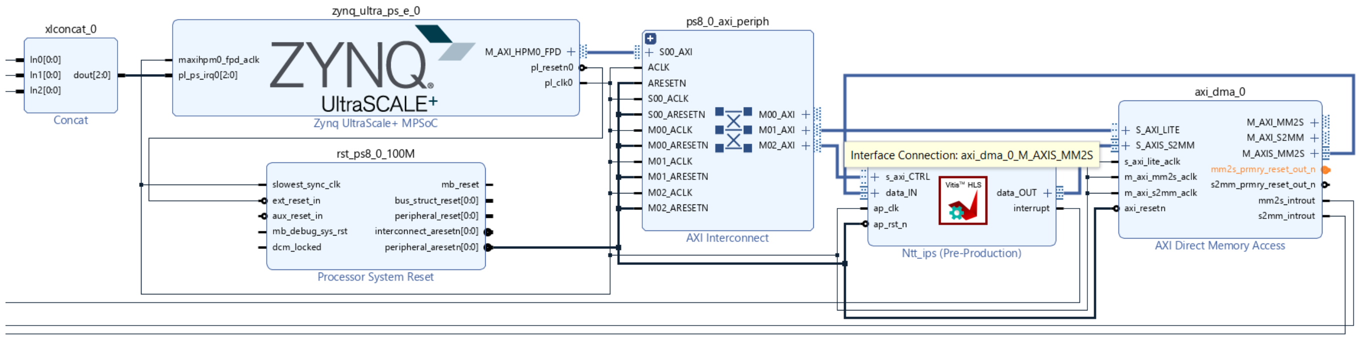
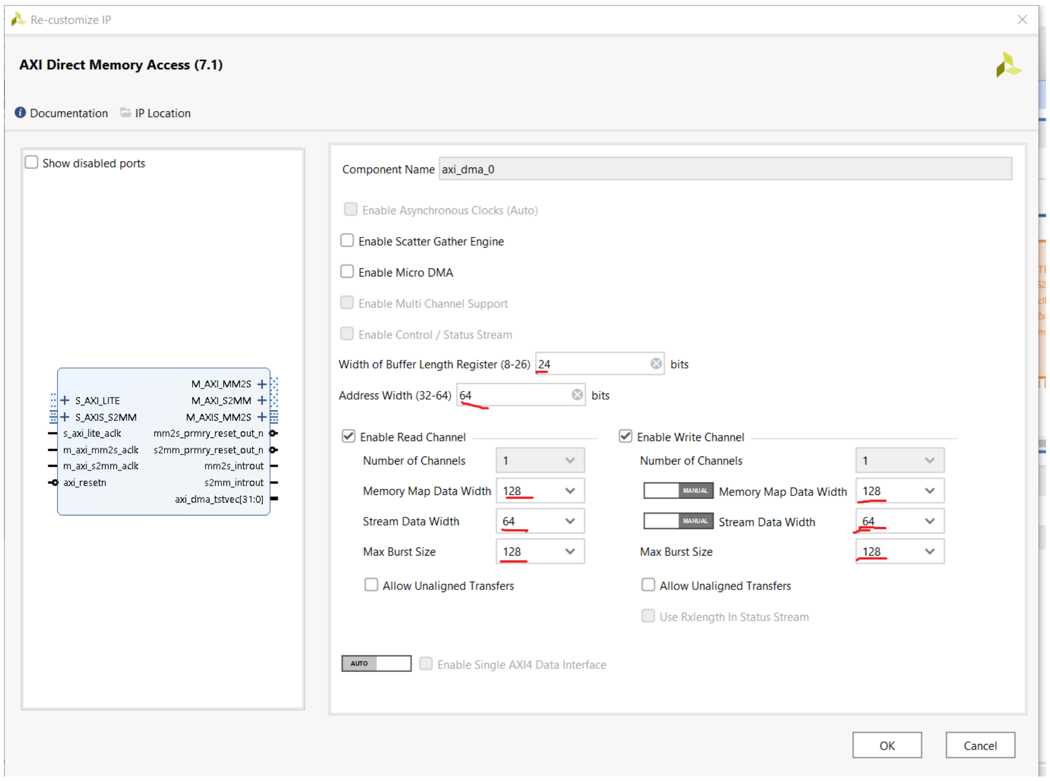


Figure 13: Vivado Design for NTT IP

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

The default values of Xilinx axi_dma IP are not suitable for the system since the system data type is 64 bits. In this context, the sections marked with red ensured that the DMA IP works in accordance with the system. In this section, the data length that DMA communicates is determined as 64 bits and the maximum transfer size is selected.



AXI Direct Memory Access (7.1)

[ ] Show disabled ports

Component Name: axi_dma_0

[ ] Enable Asynchronous Clocks (Auto)
[ ] Enable Scatter Gather Engine
[ ] Enable Micro DMA
[ ] Enable Multi Channel Support
[ ] Enable Control / Status Stream

Width of Buffer Length Register (8-26): 24 bits
Address Width (32-64): 64 bits

[✓] Enable Read Channel		[✓] Enable Write Channel	
Number of Channels	1	Number of Channels	1
Memory Map Data Width	128	Memory Map Data Width	128
Stream Data Width	64	Stream Data Width	64
Max Burst Size	128	Max Burst Size	128
[ ] Allow Unaligned Transfers		[ ] Allow Unaligned Transfers	
		[ ] Use Rxlength In Status Stream	

[ ] Enable Single AXI4 Data Interface

Figure 14: AXI-DMA Configuration

“FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: **https://github.com/DHLSan/FastHomNAND**
Video Link: https://www.youtube.com/watch?app=desktop&v=Yz9s4SajXfA&t=5s

3.3.3 Vitis 2020.2 Implementation

The TFHE library works based on C++11 standards. Therefore, in order for the library to work on the development board, complier flags must be set in accordance with C++11. This operation is provided by the -std=c+11 directive.

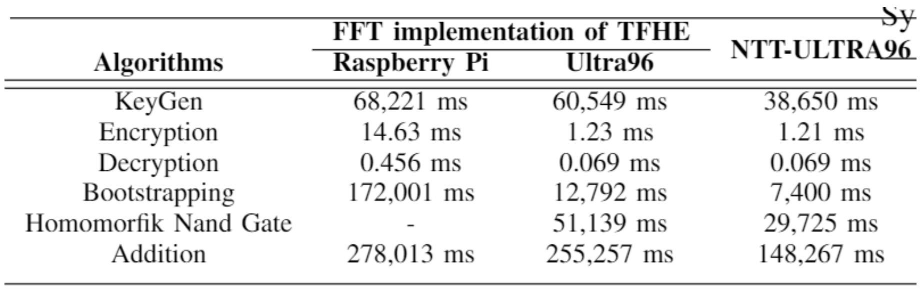
In addition, the TFHE library performs homomorphic NAND gate operation, and the memory it uses increases to about 300 MB. Since the default stack size determined in the complier is 1KB, this stack size has been increased by applying the necessary operations. After these processes were applied, the required codes from the TFHE library were fragmented and entered into the Vitis platform and run on the ULTRA96-V2.

FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?v=Yz9s4SajXfA&t=5s

4. Empirical Results

In Figure 15, there are time outputs of TFHE of a recently published article in the literature performed on raspberry pi.

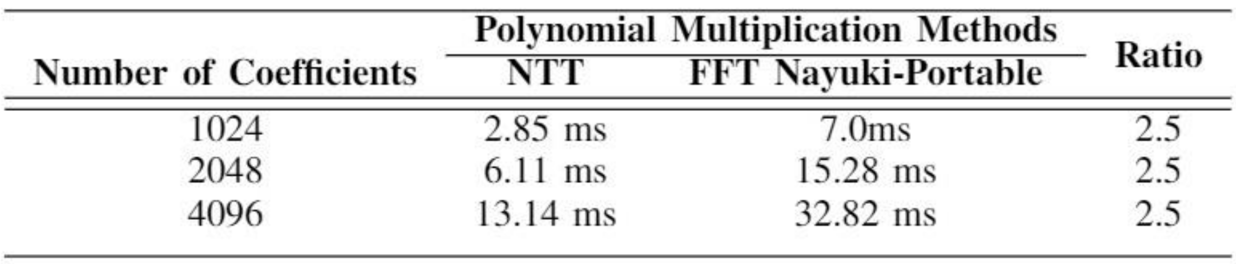


Algorithms	FFT implementation of TFHE		NTT-ULTRA96
	Raspberry Pi	Ultra96	
KeyGen	68,221 ms	60,549 ms	38,650 ms
Encryption	14.63 ms	1.23 ms	1.21 ms
Decryption	0.456 ms	0.069 ms	0.069 ms
Bootstrapping	172,001 ms	12,792 ms	7,400 ms
Homomorfik Nand Gate	-	51,139 ms	29,725 ms
Addition	278,013 ms	255,257 ms	148,267 ms

Figure 15: Raspberry Pi

As seen in figure 15, ULTRA96-V2 produces faster results compared to other development cards even in FFT application.

4.1 Software Based Acceleration Results



Number of Coefficients	Polynomial Multiplication Methods		Ratio
	NTT	FFT Nayuki-Portable	
1024	2.85 ms	7.0ms	2.5
2048	6.11 ms	15.28 ms	2.5
4096	13.14 ms	32.82 ms	2.5

Figure 16: Polynomial Multiplication on Ultra96-V2

FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?v=Yz9s4SajXfA&t=5s

Thanks to the implemented software-based customized modular multiplication and similar improvements, the polynomial multiplication process has been accelerated 2.5 times on the ULTRA96-V2. In this way, the homomorphic NAND gate operation also accelerated by 2.5 times, decreasing from 51139 ms to 29725 ms. Since polynomials with coefficients of 1024 are used in the TFHE system, the acceleration obtained when switching from FFT to NTT integration on the polynomial multiplication in Figure 16 can be observed.

x2.5

Faster

FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?v=Yz9s4SajXfA&t=5s

4.2 Hardware Based Acceleration Results

Thanks to the hardware specific IP produced with Vitis HLS, the ntt operation produced results 2.8 times faster.

Algorithms	**FFT -Ultra96**	**NTT-Ultra96**	**NTT-Ultra96 Hardware Implementation**
Homomorphic NAND Gate	51,139 ms	29,725 ms	25,26625 ms

x2.8

Faster

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: https://github.com/DHLSan/FastHomNAND
Video Link: https://www.youtube.com/watch?v=Yz9s4SajXfA&t=5s

6. Conclusion

As a result of our studies, it was decided that homomorphic encryption systems should be accelerated to be applicable for practical applications. It has been observed that fully homomorphic encryption systems are extremely suitable for sensitive data processing applications and can be used in applications where transactions will be made on finance, private health data, and data that should be kept confidential in the future. In summary, the Homomorphic Nand gate operation was chosen because it is a universal logic gate where all mathematical operations can be designed in theory. As a result of the improvements in the software part, this process has been accelerated 2.5 times on the Ultra96-V2. As a result of hardware-based developments, the homomorphic NAND gate process has been accelerated 2.8 times in general. Studies will continue on this system, which is expected to obtain much more acceleration coefficients in future studies. On TFHE, there is a repeated NTT operation. Parallelization may also be possible for this operation in an upper function. My work that will continue next month will be to accelerate this system more effectively.

FastHNAND: Hardware Acceleration of Homomorphic NAND Gate

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: **https://github.com/DHLSan/FastHomNAND**
Video Link: https://www.youtube.com/watch?v=Yz9s4SajXfA&t=5s

References

Ovunc Kocabas and Tolga Soyata. Towards privacy-preserving medicalcloud computing using homomorphic encryption. InVirtual and MobileHealthcare: Breakthroughs in Research and Practice, pages 93–125.IGI Global, 2020.

[1] Sathya, S.S., Vepakomma, P., Raskar, R., Ramachandra, R. and Bhattacharya, S., 2018. A review of homomorphic encryption libraries for secure computation. *arXiv preprint arXiv:1812.02428*.

[2] Toom, A.L.: The complexity of a scheme of functional elements realizing the multiplication of integers. In: Soviet Mathematics Doklady, vol. 3, pp. 714–716 (1963)

[3] Karatsuba, A.A., Ofman, Y.P.: Multiplication of many-digital numbers by automatic computers. In: Doklady Akademii Nauk, vol. 145, pp. 293–294. Russian Academy of Sciences (1962)

[4] Thanh-Hoang, T., Shambayati, A., Deutschbein, C., Hoffmann, H. and Chien, A.A., 2014, September. Performance and energy limits of a processor-integrated fft accelerator. In *2014 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1-6). IEEE.

[5] De Clercq, R., Roy, S.S., Vercauteren, F. and Verbauwhede, I., 2015, March. Efficient software implementation of ring-LWE encryption. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (pp. 339-344). IEEE.

[6] Liu, Z., Azarderakhsh, R., Kim, H. and Seo, H., 2017. Efficient software implementation of ring-LWE encryption on IoT processors. *IEEE Transactions on Computers*

[7] Thanawala, N., 2021. *Hardware Acceleration of Polynomial Multiplication using Pipelined FFT* (Doctoral dissertation, UC Irvine).

[8] Millar, K., Łukowiak, M. and Radziszowski, S., 2019, December. Design of a flexible Schönhage-Strassen FFT polynomial multiplier with high-level synthesis to accelerate HE in the cloud. In *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)* (pp. 1-5). IEEE.

[9] Pöppelmann, T., Naehrig, M., Putnam, A. and Macias, A., 2015, September. Accelerating homomorphic evaluation on reconfigurable hardware. In *International workshop on cryptographic hardware and embedded systems* (pp. 143-163). Springer, Berlin, Heidelberg.

[10] Roy, S.S., Turan, F., Jarvinen, K., Vercauteren, F., Verbauwhede, I.: Fpga-based highperformance parallel architecture for homomorphic computing on encrypted data. Cryptology ePrint Archive, Report 2019/160 (2019)

University name: Eskisehir Technical University
List of group members: Batuhan BULUT
Supervisor: Assist. Prof. Dr. Ismail San
Email of students and Supervisor: Batuhanbulut9196@gmail.com
Which board you used: Ultra96-V2
GitHub Link: **https://github.com/DHLSan/FastHomNAND**
Video Link: https://www.youtube.com/watch?v=Yz9s4SajXfA&t=5s

[11] Sinha Roy, S., J¨arvinen, K., Vliegen, J., Vercauteren, F., Verbauwhede, I.: Hepcloud: An fpga-based multicore processor for fv somewhat homomorphic function evaluation. IEEE Transactions on Computers 67(11), 1637–1650 (2018). DOI 10.1109/TC.2018.2816640

[12] 6. Riazi, M.S., Laine, K., Pelton, B., Dai, W.: Heax: An architecture for computing on encrypted data. In: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '20, p. 1295–1309. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3373376.3378523. URL https://doi.org/10.1145/3373376.3378523

[13] Al Badawi, A., Veeravalli, B., Aung, K.M.M.: Faster number theoretic transform on graphics processors for ring learning with errors based cryptography. In: 2018 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), pp. 26–31. IEEE (2018)

[14] 6. Al Badawi, A., Veeravalli, B., Mun, C.F., Aung, K.M.M.: High-performance fv somewhat homomorphic encryption on gpus: An implementation using cuda. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 70–95 (2018)

[15] Rivest, R.L., Shamir, A. and Adleman, L., 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), pp.120-126.

[16] Fontaine, C. and Galand, F., 2007. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, *2007*, pp.1-10.

[17] Bos, J.W., Costello, C., Naehrig, M. and Stebila, D., 2015, May. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *2015 IEEE Symposium on Security and Privacy* (pp. 553-570). IEEE.

[18] https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf