ANALYSIS AND INNER-ROUND PIPELINED IMPLEMENTATION OF SELECTED PARALLELIZABLE CAESAR COMPETITION CANDIDATES

by

Sanjay Deshpande

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

_____ Dr. Kris Gaj, Thesis Director

_____ Dr. Jens Peter Kaps, Committee Member

_____ Dr. Xiang Chen, Committee Member

_____ Dr. Monson Hayes, Department Chair

_____ Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: _____ Fall Semester 2016

George Mason University

Fairfax, VA.

Analysis and Inner-Round Pipelined Implementation of Selected Parallelizable CAESAR Competition Candidates

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

by

Sanjay Deshpande
Bachelor of Technology
Jawaharlal Nehru Technological University, 2014

Director: Kris Gaj, Associate Professor
Electrical and Computer Engineering

Fall Semester 2017
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to Shri Kesari Hanuman, my grandfather Shri Nand Kumar Deshpande, my parents Megha Deshpande & Vinay Deshpande, my advisor Dr. Kris Gaj, and Ankitha Prabhu and my beloved friends.

Acknowledgement

I would like to express my heartfelt gratitude to my advisor Dr. Kris Gaj for his patience, motivation and guidance through the research and thesis documentation. I would also take this opportunity to thank Dr. Jens Peter Kaps, Ekawat Homsirikamol a.k.a “Ice”, William Diehl, Farnoud Farahmand, Panasayya Yalla, Ahmed Ferozpuri, Malik Umar Sharif and Rabia Shahid for their immense help.

Table of Contents

Page

List of Tables

Table Page

List of Figures

Figure Page

Abstract

ANALYSIS AND INNER-ROUND PIPELINED IMPLEMENTATION OF SELECTED PARALLELIZABLE CAESAR COMPETITION CANDIDATES

Sanjay Deshpande, M.S.

George Mason University, 2016

Thesis Director: Dr. Kris Gaj

In this thesis, we have first characterized candidates of the Competition for Authenticated Encryption, Security, Applicability, and Robustness (CAESAR). Then, we have chosen five candidates from the Round 2 and Round 3 submissions, namely SCREAM, AES-COPA, Minalpher, OCB, and AES-OTR. We first obtained the initial estimates of the Maximum Clock Frequency, Throughput, Area, and Critical path from the Basic Iterative High Speed Architecture. Then, we implemented the inner-round pipelining for all the selected algorithms to improve the Frequency and Throughput by reducing Critical path and processing multiple blocks of data simultaneously. We targeted the largest available FPGA in the student version of Xilinx ISE, i.e., Xilinx Virtex 6 XC6VLX75T-3FF784. Our results have demonstrated the improvement in the Clock Frequency by a factor varying from x1.28 for OCB to x1.84 for SCREAM, and the improvement in the Throughput to Area ratio (with Area expressed using LUTs) by a factor varying from x0.96 for Minalpher to x1.70 for SCREAM.

1. Introduction

Encryption is the most effective way to achieve data security. The primary purpose of encryption is to protect the confidentiality of digital data stored on computer systems or transmitted via the Internet or other computer networks. Modern encryption algorithms play a vital role in the security assurance of IT systems and communications as they can provide not only confidentiality, but also the following key elements of security

Authentication: The origin of a message can be verified.

Integrity: Proof that the contents of a message have not been changed since the message was sent.

Non-repudiation: The sender of a message cannot deny sending the message.

1.1. Authenticated Encryption:

1.1.1. What is Authenticated Encryption?

Authenticated Encryption (AE) or Authenticated Encryption with Associated Data (AEAD) is a cryptographic algorithm that simultaneously provides confidentiality, integrity, and authentication of message; decryption is combined in single step with integrity verification. The Authenticated ciphers takes plaintext message, associated data AD, a public message number Npub, and an optional secret message number Nsec as an

input and provide resulting ciphertext C, tag T, and optional encrypted Nsec. The ciphertext C, and optional encrypted Nsec are computed as a function of Npub, Nsec, AD, message, and key and this transformation ensures the confidentiality of the transaction. At the end of plaintext encryption a tag T is produced which is a keyed-hash function computed from all blocks of the AD and plaintext, as well as Npub, Nsec, and key. The tag is appended to the end of the ciphertext to assure and verify the integrity and authenticity of the transaction as shown in the Figure *1*. Decryption of the ciphertext and optional encrypted Nsec is conducted in a similar fashion. Identical parameters, including AD, key, and message numbers, are required for validation. Tag' is then computed as above, and verified against the concatenated Tag. If Tag = Tag' then authentication and integrity of the transaction are assured; otherwise the decrypted ciphertext is not released. If authenticity and integrity are verified, the outputs of the transaction are the AD, plaintext, and optional decrypted Nsec.

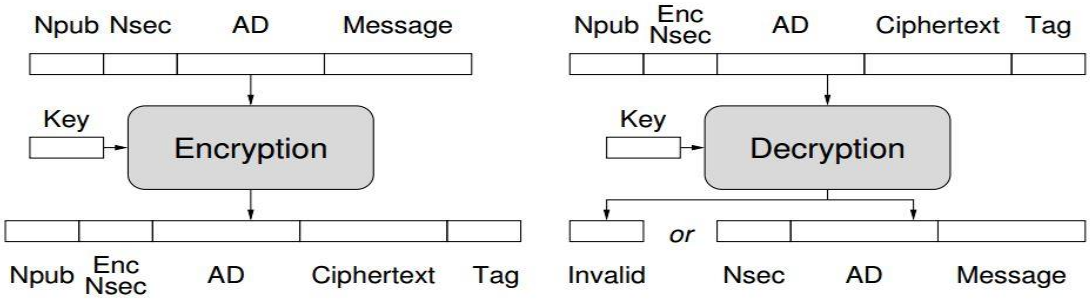

Figure 1: Input and Output of an Authenticated Cipher. Notation: Npub - Public Message Number, Nsec - Secret Message Number, Enc Nsec - Encrypted Secret Message Number, AD - Associated Data

A typical programming interface for Authenticated Encryption provides the following functions.

Encryption:

Input: plaintext, key, and optionally a header in plaintext that will not be encrypted, but will be covered by authenticity protection.

Output: ciphertext and authentication tag (Message Authentication Code).

Decryption:

Input: ciphertext, key, authentication tag, and optionally a header.

Output: plaintext, or an error if the authentication tag does not match the supplied ciphertext or header.

1.1.2. Applications and Advantages of Authenticated Encryption

Authenticated encryption can provide plaintext awareness and security against chosen ciphertext attacks. In these attacks, an adversary attempts to gain an advantage against a cryptosystem (e.g., information about the secret decryption key) by submitting carefully chosen ciphertexts to some decryption oracle and analyzing the decrypted results. Authenticated encryption schemes can recognize improperly-constructed ciphertexts and refuse to decrypt them. This in turn prevents the attacker from requesting the decryption of any ciphertext unless he generated it correctly using the encryption algorithm, which would imply that he already knows the plaintext. Implemented correctly, this removes the usefulness of the decryption oracle, by preventing an attacker from gaining useful information that he does not already possess.

1.2. CAESAR Contest

1.2.1. Organization and Schedule

Cryptographic competitions have become common way of developing the cryptographic standard. This process has worked really well in case of Advanced Encryption Standard (AES), developed in the period 1997-2001, and then SHA 3 competition (Secure Hash Algorithm 3), developed in the period 2007-2012. In 2013, a new contest, called CAESAR - Competition for Authenticated Encryption: Security, Applicability, and Robustness - has been announced. This contest started off with 57 candidates in Round 1, and then reached Round 2 with 29 candidates and Round 3 with 15 candidates remaining.

1.2.2. Evaluation Criteria

Performance of candidates in hardware has always been a very important evaluation factor, when all remaining algorithms have been found to have adequate security strength. Hardware evaluation has become possible in CAESAR because of the two novel approaches. First, the design teams have been asked to submit their own Verilog/VHDL code before the end of Round 2. Secondly, High-Level Synthesis, based on the newly developed Xilinx Vivado HLS tool, has been applied to transform reference C implementations of CAESAR candidates to the corresponding efficient Register Transfer Level (RTL), hardware description language (HDL).

1.2.3. Importance of Hardware Benchmarking

In CAESAR competition, an attempt has been made to conduct hardware benchmarking of each candidate at early stages of the contest, when the number of competing algorithms was still very large, namely there were still 29 authenticated cipher families remaining, with multiple variants for some of them (such as PRIMATEs, Deoxys, Keyak).

2. Classification of the CAESAR Candidates

2.1. Introduction

Secret-key Cryptography helps in protecting the confidentiality and integrity of the messages against all possible misbehavior by the attacker. Even if the Public Key Cryptography introduced new ways to protect and share the messages, the Secret-key cryptography has always proven to outperform it in terms of speed and speed to area ratio. Usually the data is protected either by secret-key cryptography alone or by a hybrid of public-key and secret-key cryptography.

2.2. Design Classification

2.2.1. Type

Block cipher: A Block cipher encrypts one block of a message at a time, independently from other blocks. A block cipher consists of two paired algorithms, one for encryption and the other one for decryption. The decryption algorithm is said to be the inverse function of encryption.

The encryption function can be specified as follows

$$E_K(P) := E(K,P) : \{0,1\}^k \, X \, \{0,1\}^n \rightarrow \{0,1\}^n,$$

The inverse function, i.e., the decryption function can be specified as follows:

$$E^{-1}_K(C) := D_K(C) = D(K, C) : \{0, 1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$$

From the equations we can see that both encryption and decryption accept two inputs: an input block of size n bits and a key of size k bits, both yielding an n-bit output block.

For example, AES is a block cipher that encrypts a 128-bit block using a 128bit, 192-bit, and 256-bit keys.

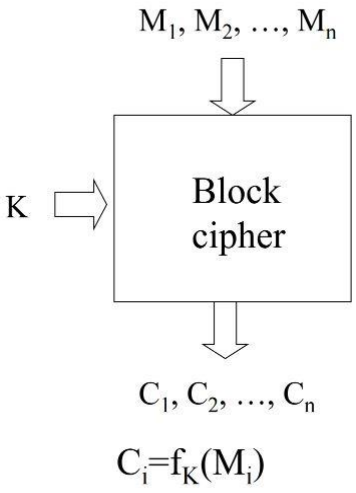

Figure 2: Block Cipher

Stream Cipher:

In a Stream cipher every block of ciphertext is a function of the current block of plaintext and the current internal state of the cipher. A stream cipher encrypts a variable length message using a public nonce and a secret key.

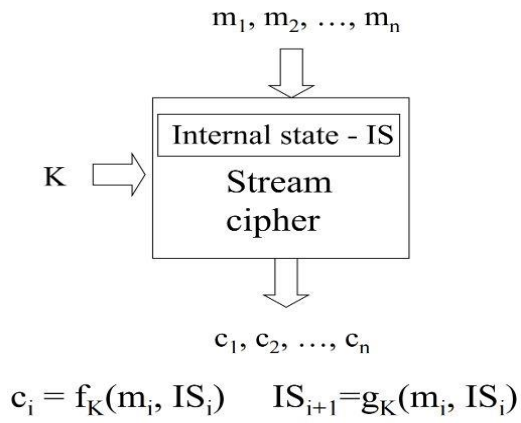

Figure 3: Stream Cipher

Message Authentication Code (MAC): MAC is used to authenticate the message, it is a short piece of information used to confirm that the message has been received from the stated sender and has not been changed in transit. The Authenticator is also sent with the encrypted message which protects message against corruption. MACs are often built from block ciphers or from cryptographic hash functions such as SHA-3.

Authenticated Encryption with Associated Data (AEAD): AEAD is a block cipher mode of operation, which provides all three security services, i.e., confidentiality, integrity and authentication. Decryption involves integrity and authenticity verification.

From the Round 2 of the CAESAR competition, data of all the 29 candidates was extracted from the specification of each cipher and analysis of all candidates from the point of view of capability for parallel processing of blocks belonging to the same associated data, message, and ciphertext was done. All the data was tabulated as shown in Table *1*.

Then, from the ciphers which had the capability for parallel processing of blocks, 5 candidates were selected based on their maximum clock frequency in Basic High-Speed Architecture. The ciphers which had lowest maximum frequency were selected.

Table 1: Analysis of ALL Round 2 candidates from the point of view of capability for parallel processing of blocks belonging to the same AD/message/ciphertext.

Candidate	Type	Tag Size	Key Size	Nonce Size	Block Size	State size	Max AD/ M Size	#Round	Parallelizable	NMRM
ACORN	SC	128	128	128	1	293	$<2^{64}$	10	yes	
AEGIS-128L	BC	128	128	128	256	1024	2^{64}	1	no	
AEGIS-128	BC	128	128	128	128	640	2^{64}	1	no	
AEGIS-256	BC	128	256	256	256	768	2^{64}	1	no	
AES-COPA	BC	128	128	128	128	128	2^{71}	10	yes	yes
AES-JAMBU	BC	48	96	48	48	144	$<2^{64}$	52	no	
AES-JAMBU	BC	32	96	32	32	96	$<2^{64}$	42	no	
AES-JAMBU	BC	64	128	64	64	192	$<2^{64}$	68	no	
AES-JAMBU	BC	64	128	64	64	192	$<2^{64}$	68	no	
AES-OTR	BC	128	128	96	128	128	2^{64}	10	yes	
AES-OTR	BC	128	128	96	128	128	2^{64}	10	yes	
AES-OTR	BC	128	256	96	128	128	2^{64}	10	yes	
AES-OTR	BC	128	256	96	128	128	2^{64}	10	yes	
AEZ	TBC	128	128	128	128	128	2^{51}	$8 < R < 24$	yes	yes
Ascon-128	P	128	128	128	64	320	$<2^{67}$	6	no	
Ascon-128a	P	128	128	128	128	320	$<2^{68}$	8	no	
CLOC-AES-12	BC	64	128	96	128	256	$<2^{67}$	10	no/yes	
CLOC-AES-8	BC	64	128	64	128	256	$<2^{67}$	10	no/yes	
CLOC-TWINE	BC	32	80	48	128	256	$<2^{67}$	10	no/yes	
SILC-AES	BC	64	128	96	128	256	$<2^{67}$	10	no/yes	
SILC-AES	BC	64	128	64	128	256	$<2^{67}$	10	no/yes	
SILC-PRESENT	BC	32	80	48	128	256	$<2^{67}$	31	no/yes	
SILC-LED	BC	32	80	48	128	256	$<2^{67}$	48	no/yes	

Table 1(2): Analysis of ALL Round 2 candidates from the point of view of capability for parallel processing of blocks belonging to the same AD/message/ciphertext.

Candidate	Type	Tag Size	Key Size	Nonce Size	Block Size	State size	Max AD/ M Size	#Round	Parallelizable	NMRM
Deoxys ≠ -128 - 128	BC	128	128	64	128	128	2^{67}	14	yes	
Deoxys ≠ -256 - 128	BC	128	256	64	128	128	2^{67}	16	yes	
Deoxys = -128-128	BC	128	128	128	128	128	2^{67}	14	yes	yes
Deoxys= -256-128	BC	128	256	128	128	128	2^{67}	16	yes	yes
ELmD	BC	128	128	64	128	128	2^{64}	12	yes	yes
HS1-SIV-lo	HS1	64	256	96	256	512	2^{67}	8	no	yes
HS1-SIV	HS1	128	256	96	256	512	2^{67}	12	no	yes
HS1-SIV-hi	HS1	256	256	96	256	512	2^{67}	20	no	yes
ICEPOLE 128	P	128	128	128	1024	1280	2^{64}	6	no	
ICEPOLE 128a	P	128	128	96	1024	1280	2^{64}	6	no	
Joltik≠ -64 - 64	TBC	64	64	64	128	128	$<2^{35}$	24	yes	
Joltik≠-80 - 112	TBC	112	80	64	128	128	$<2^{35}$	32	yes	
Joltik≠ -96 - 96	TBC	96	96	64	128	128	$<2^{35}$	32	yes	
Joltik≠ -128 - 64	TBC	64	128	64	128	128	$<2^{35}$	32	yes	
Joltik= -64-64	TBC	64	64	64	128	128	$<2^{66}$	24	yes	yes
Joltik= -80-112	TBC	112	80	64	128	128	$<2^{66}$	32	yes	yes
Joltik= -96-96	TBC	96	96	64	128	128	$<2^{66}$	32	yes	yes
Joltik= -96-96	TBC	96	96	64	128	128	$<2^{66}$	32	yes	yes
Joltik= -128-64	TBC	64	128	64	128	128	$<2^{66}$	32	yes	yes
Ketje Jr	P	64	96	86	16	200	$<2^{96}$	22	no	
Keyak	P	128	128	128	1600	1536	$<2^{123}$	12	no	

Table 1(3): Analysis of ALL Round 2 candidates from the point of view of capability for parallel processing of blocks belonging to the same AD/message/ciphertext.

Candidate	Type	Tag Size	Key Size	Nonce Size	Block Size	State size	Max AD/ M Size	#Round	Parallelizable	NMRM
Minalpher	TBC	128	128	104	256	64	$<2^{104}-1$	17	yes	yes
MORUS-1280-128	SC	128	128	128	128	1280	$<2^{64}$	5	no	
MORUS-640-128	SC	128	128	128	128	640	$<2^{64}$	5	no	
MORUS-1280-256	SC	128	256	128	256	1280	$<2^{64}$	5	no	
NORX32	P	128	128	64	128	512	$<2^{128}$	6	no	
NORX64	P	256	256	128	128	1024	$<2^{128}$	4	no	
OCB	BC	128	128	120	128	128	$<2^{128}$	10	yes	
OMD	CF	32-256	256	256	256	256	$<2^{64}$	64	no	
PAEQ64	P	64	64	64	432	512	$<2^{99}$	20	yes	
PAEQ80	P	80	80	80	416	512	$<2^{99}$	20	yes	
PAEQ128	P	128	128	128	368	512	$<2^{99}$	20	yes	
PAEQ64-T	P	512	64	64	432	512	$<2^{99}$	20	yes	
PAEQ64-TNM	P	512	64	128	432	512	$<2^{99}$	20	yes	
PAEQ128-T	P	512	128	128	368	512	$<2^{99}$	20	yes	
PAEQ128-TNM	P	512	256	256	240	512	$<2^{99}$	20	yes	yes
PAEQ192	P	128	192	128	304	512	$<2^{99}$	20	yes	
PAEQ160	P	160	160	128	336	512	$<2^{99}$	20	yes	
PAEQ256	P	128	256	128	240	512	$<2^{99}$	20	yes	
Π-Cipher096	P	128	96	32	256	128	$<2^{67}-8$	3	no	
Π-Cipher128	P	256	128	128	512	256	$<2^{67}-8$	3	no	
Π-Cipher128	P	512	128	128	1024	512	$<2^{67}-8$	3	no	
Π-Cipher256	P	512	256	128	1024	512	$<2^{67}-8$	3	no	

Table 1(4): Analysis of ALL Round 2 candidates from the point of view of capability for parallel processing of blocks belonging to the same AD/message/ciphertext.

Candidate	Type	Tag Size	Key Size	Nonce Size	Block Size	State size	Max AD/ M Size	#Round	Parallelizable	NMRM
POET-AES10-AES4	BC	128	128	128	128	128	$<2^{128}$	10	yes	yes
POET-AES10-AES10	BC	128	128	128	128	128	$<2^{128}$	10	yes	yes
PRIMATEs-HANUMAN-120	P	120	120	120	40	280	$<2^{120}$	12	no	
PRIMATEs-GIBBON-120	P	120	120	120	40	280	$<2^{120}$	6	no	
PRIMATEs-APE-120	P	240	240	120	80	280	$<2^{120}$	12	no	
PRIMATEs-HANUMAN-80	P	80	80	80	40	200	$<2^{80}$	12	no	
PRIMATEs-GIBBON-80	P	80	80	80	40	200	$<2^{80}$	6	no	
PRIMATEs-APE-80	P	160	160	80	80	200	$<2^{80}$	12	no	
SCREAM	TBC	128	128	96	128	128	$<2^{128}$	10	yes	
SHELL	BC	128	128	64	128	256	$<2^{70}$	10	yes	
STRIBOB	P	128	192	128	64	128	$2^{127}/2^{64}$	12	yes	
Tiaoxin-346	BC	128	128	128	128	768	$<2^{128}-1$	35	yes	
TriviA-ck	SC	128	128	64	64	384	$<2^{64}/<2^{128}$	64	no	

Legend:

R - Number of Rounds

NMRM - Nonce Misuse Resistant Mode

Max AD - Maximum Associated Data

M size - Message size in bits

In Parallelizable column **no/yes** means that Encryption is not parallelizable and Decryption is parallelizable.

Types of Cipher:

SC- Stream Cipher based

BC -Block Cipher based

P - Permutation based

TBC- Tweakable Block Cipher based

CF - Compression Function based

HS1- Hash Stream 1 based

3. General Methodology

Pipelining is one of the well-known techniques used to increase the speed of any digital design. In this project we have implemented inner-round pipelining of 5 different ciphers. The inner-round pipelining provides substantial amount of increase in the speed of the cipher with small increase in the circuit area. In this method the pipeline registers are inserted inside the round function of the cipher and then path is balanced accordingly for the dataflow [15].

The Basic Iterative architecture, as shown in *Figure 4*, is implemented first, and its frequency, area and critical path are determined. Based on this information, we insert a pipeline register to reduce the critical path. The location of the pipeline register is chosen in such a way that the critical path between two adjacent registers is reduced and balanced. In this project we have implemented a two-stage inner-round pipelining, as shown in Figure *5*.

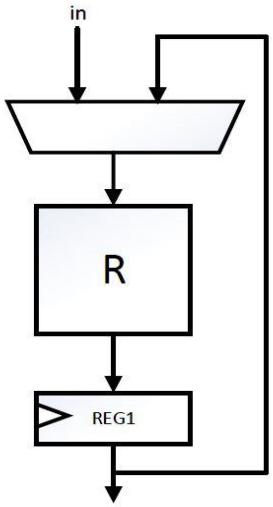

Figure 4: Basic Architecture

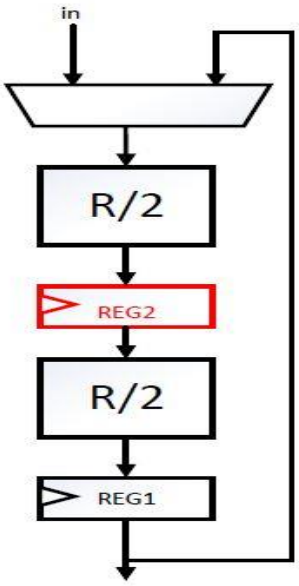

Figure 5: 2-stage inner-round pipelined architecture

The implementation strategy of a pipelined architecture is as shown in the form of a timing diagram in Figure *6* and Figure *7*.

For N Rounds:

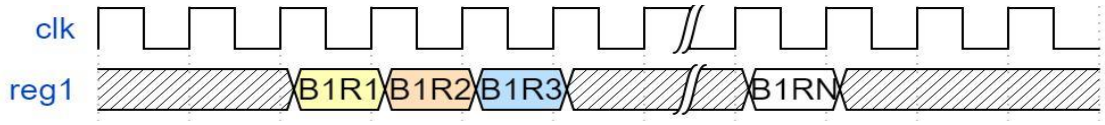

Figure 6: Basic iterative architecture: B-Block, R- Round, B1R1- Block1 Round1, and so on.

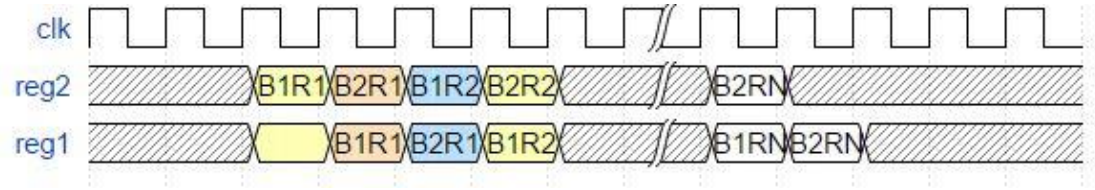

Figure 7: 2-stage inner-round pipelined architecture: B-Block, R- Round, B1R1- Block1 Round1, B2R1 Block2 Round1, and so on.

As shown in *Figure 6*, in the basic iterative architecture single block of data is processed through N rounds in N clock cycles, and then the result is sent to the output. In the 2-state inner-round pipelined architecture, two blocks of data are read in two consecutive clock cycles, and the output is collected after 2N+1 clock cycles. Additionally, two consecutive pairs of input blocks can be processed every 2N clock cycles.

We have implemented five selected ciphers targeting the largest available FPGA in the student version of Xilinx ISE, i.e., Xilinx Virtex 6 XC6VLX75T-3FF784. The inner-round pipeline implementations of all ciphers are shown in the following chapters.

4. SCREAM – Side Channel Resistant Authenticated Encryption with Masking

4.1. Introduction and Major Features

SCREAM [6] is a CAESAR candidate submitted by Grosso et al. It is based on Liskov, Rivest and Wagner's Tweakable Block Cipher (TBC). It is a simple and regular design allowing excellent performance using a wide range of architectures. The specification of the cipher claims the ease of masking used as a side-channel countermeasure. The SCREAM's resistance against conventional attacks is inherited from TAE, providing security beyond the birthday bound. The other important advantage of SCREAM is that it provides fully parallelizable authenticated encryption.

4.2. Recommended Parameters

There are totally four sets of Parameters for SCREAM, based on the security level (with 6, 8, 10 and 12 steps).

Lightweight security. 80-bit security, with a protocol avoiding related keys

Tight parameters: 6 steps, Safe parameters: 8 steps.

Single-key security. 128-bit security, with a protocol avoiding related keys

Tight parameters: 8 Steps, Safe parameters: 10 steps.

Related-key security. 128-bit security with possible related keys

Tight parameters: 10 Steps, Safe parameters: 12 Steps.

The recommended sets of parameters are as follows:

- First set of recommended parameters: SCREAM with 10 steps, single-key security.
- Second set of recommended parameters: SCREAM with 12 steps, related-key security.

4.3. Encryption and Decryption

The SCREAM works in the TAE mode as proposed in [1].

There are 3 main steps in the encryption process:

First, the associated data is processed by dividing it into 128-bit blocks. Each block is encrypted through the tweakable block cipher and then the output values are XORed, and the final output of this step is stored as Auth.

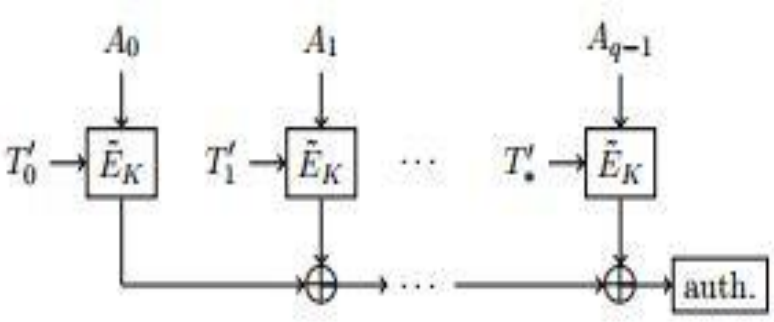

Figure 8: TAE: associated data processing.

Second, plaintext is encrypted using the tweakable block cipher in order to generate the ciphertext values. If the last block is partial, its bitlength is encrypted to generate a mask and the result of encryption is truncated to the partial block. This block is then XORed with the partial plaintext block, such that the ciphertext length is same as the plaintext length.

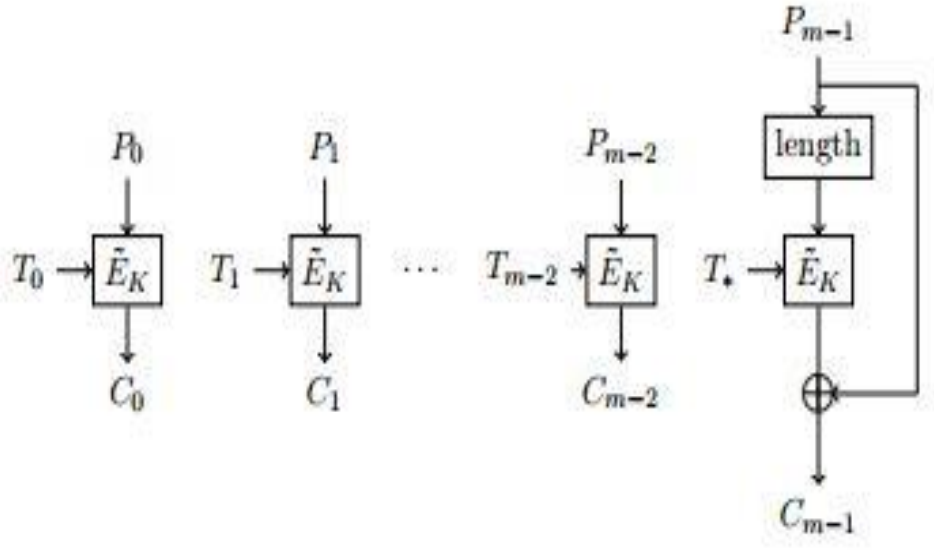

Figure 9: TAE: encryption of the plaintext blocks.

Finally, the tag is generated as shown below in Figure *10*. The checksum (i.e. the XOR of all plaintext blocks) is encrypted and then XORed with auth.

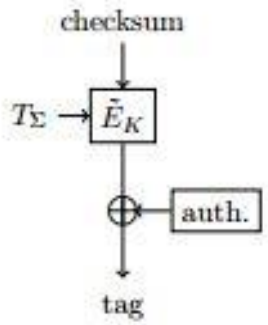

Figure 10: TAE Tag processing

In TAE mode, for security reasons we always use distinct tweak values. The algorithm uses some special values for domain separation, and tweaks of the form (N||c||control byte), where N is the Nonce and c is a block counter.

Decryption is similar to encryption, with slight changes, which will be discussed in Section 4.4. During decryption, the values of ciphertext C, tag T, and associated data A are used to recover the plaintext. If the tag is incorrect, the algorithm returns a null output.

4.4. Basic High-Speed Architecture

The implementation of SCREAM has been divided into a Datapath and Controller. This design processes one 128-bit block at a time.

4.4.1. Datapath Design

From the recommended set of parameters, SCREAM with 10 steps, single-key security was implemented. The values of Key, public message number (Npub), checksum, Tag and Tweak input are registered as shown in Figure *11*. The inputs are given to the Ek_bidir, which computes values of ciphertext/message from the given message/ciphertext respectively.

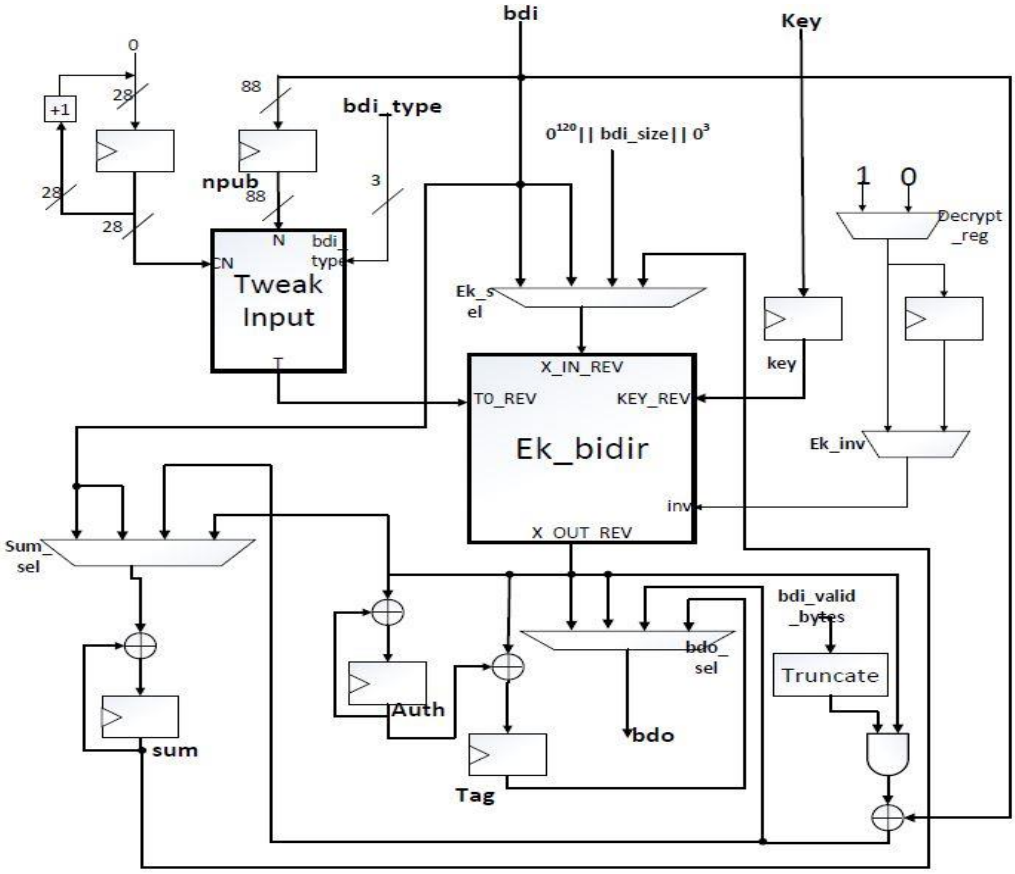

Figure 11: SCREAM Datapath

The entity Ek_bidir is involved in performing both Encryption and Decryption. The Ek_bidir has a step function called Ek_step, which is built with the combination of S-Box and L-Box. The values of Sigma and Tweak are registered and are provided as an input to Ek_step. All the inputs from the top module are converted into Reverse Endian and given as an input to Ek_step. The Ek_step function acts like a round function from AES. The algorithm includes 10 rounds known as steps. The count of each step value is dependent upon the value of Sigma. The sigma value is initiated with zero in encryption mode and incremented by one per each clock cycle until ten (the number of rounds) whereas in the decryption mode the value of sigma is initialized with ten and is decremented by one in each clock cycle until it reaches zero. The Tweak value is dependent on the value of sigma, so with each new value of sigma the tweak value changes as well. Processing of one block of data takes ten clock cycles.

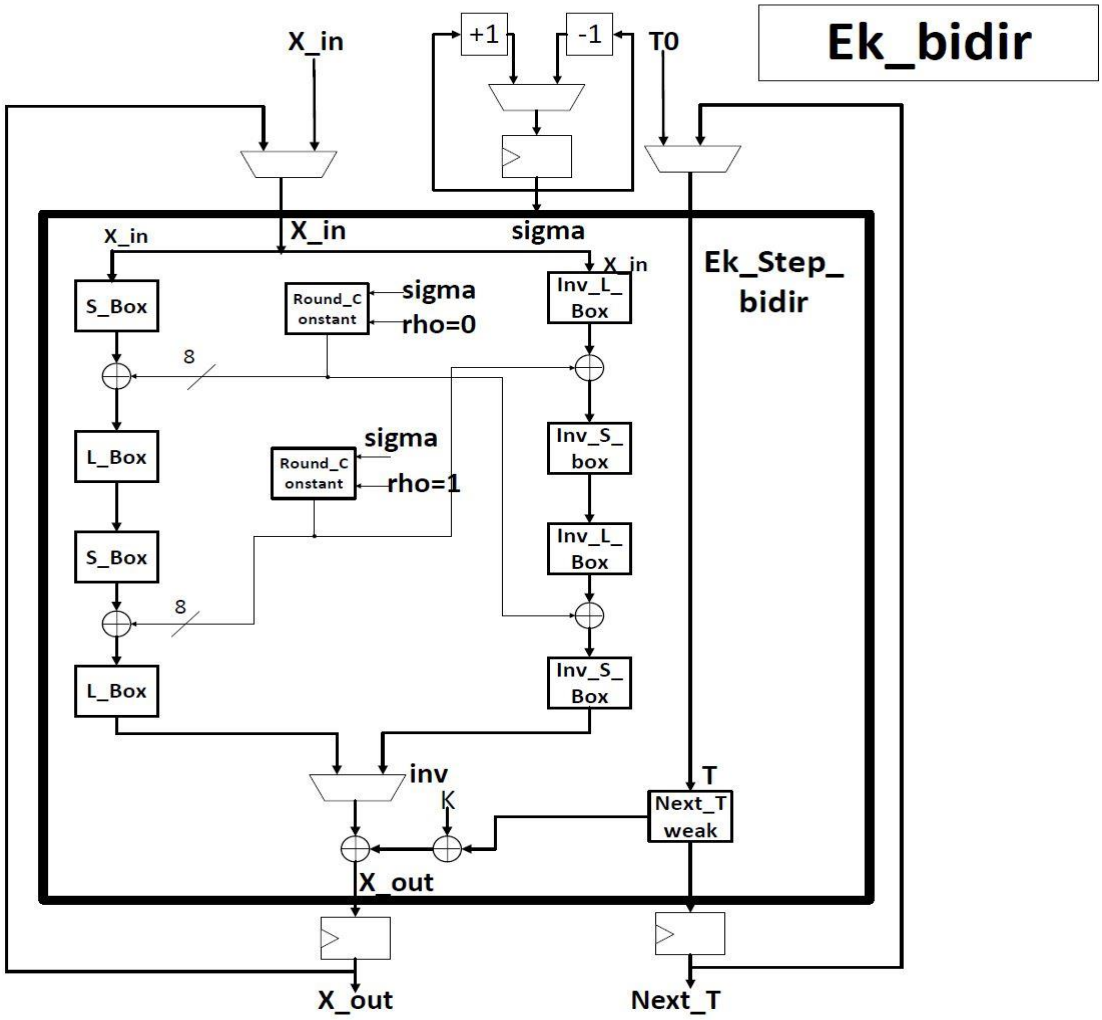

Figure 12: Ek_bidir

The Ek_Step function consists of two round functions, round0 and round1. The round function is formed with the combination of S-box and L-box. After the data is processed through the S-box, it is XORed with the Round constant, and then sent through the L-box. The order of round0 and round1 processing is based on

mode of operation. In encryption mode the data is processed through round0 first, and then it is passed through round1, whereas in decryption mode, data is processed through round1 first and then through round 2. After processing these rounds, the resulting value is XORed with the Tweak value.

4.4.2. Controller Design:

Upon the reset the controller enters the initialization state as shown in the Figure *13*. Then, the next state is Key update. In this state, if bdi_valid and key_update are high, then the key is updated, else the control moves to the next state, that is Load public message number. In this state Public message number (Npub) is loaded. Then, the associated data processing starts. Every time the Ek_done signal goes high, next block of associated data is loaded in for processing. When the end of type signal (bdi_eot) goes high, then the last block of associated data is processed. Next state is Encryption/Decryption of Message. Based on the value of the decrypt signal, the processing is started with Ek_start going high. Every time Ek_done signal goes high, the next block of message is loaded for Encryption/Decryption. Once the last block of message is encrypted/decrypted next state is Tag generation. In this state, if the decrypt is low, then tag is generated, done signal goes high, and control shifts to the initialization state. If the decrypt signal is high, then after Tag generation state, the next state is Tag verification state, where the Tag is verified and then the control shifts to the initialization state.

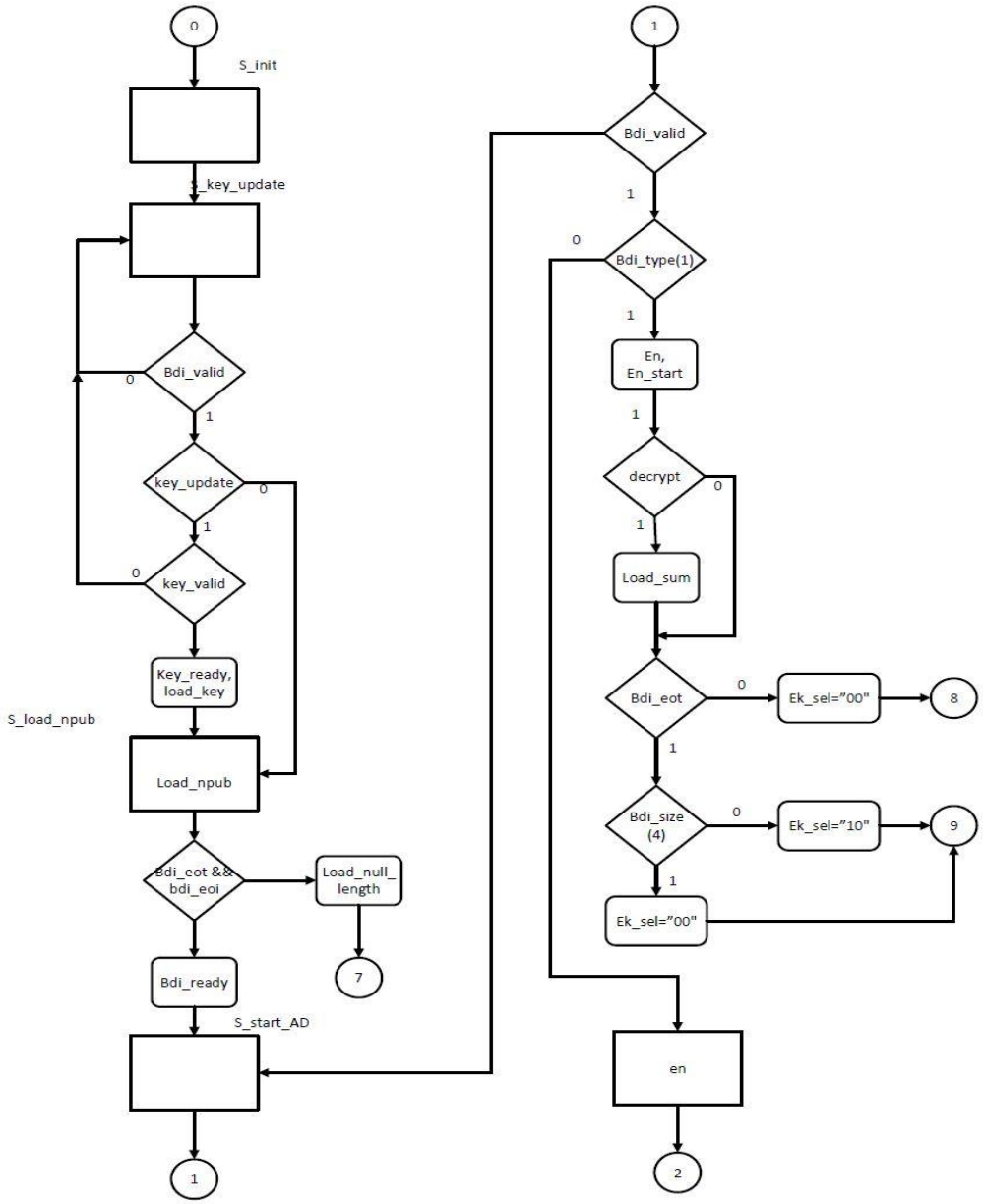

Figure 13: SCREAM Cipher Controller ASM (1)

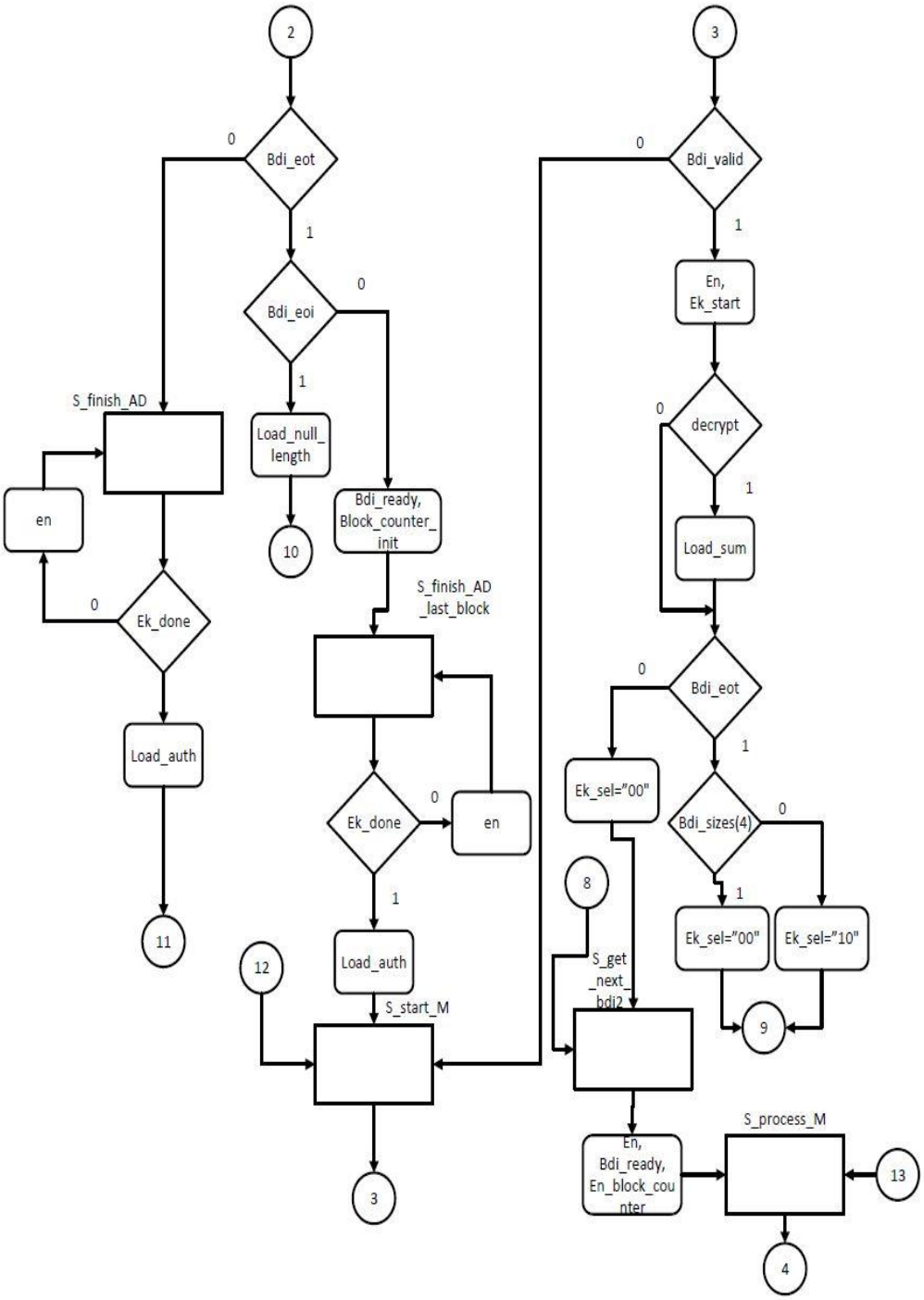

Figure 14: SCREAM Cipher Controller ASM (2)

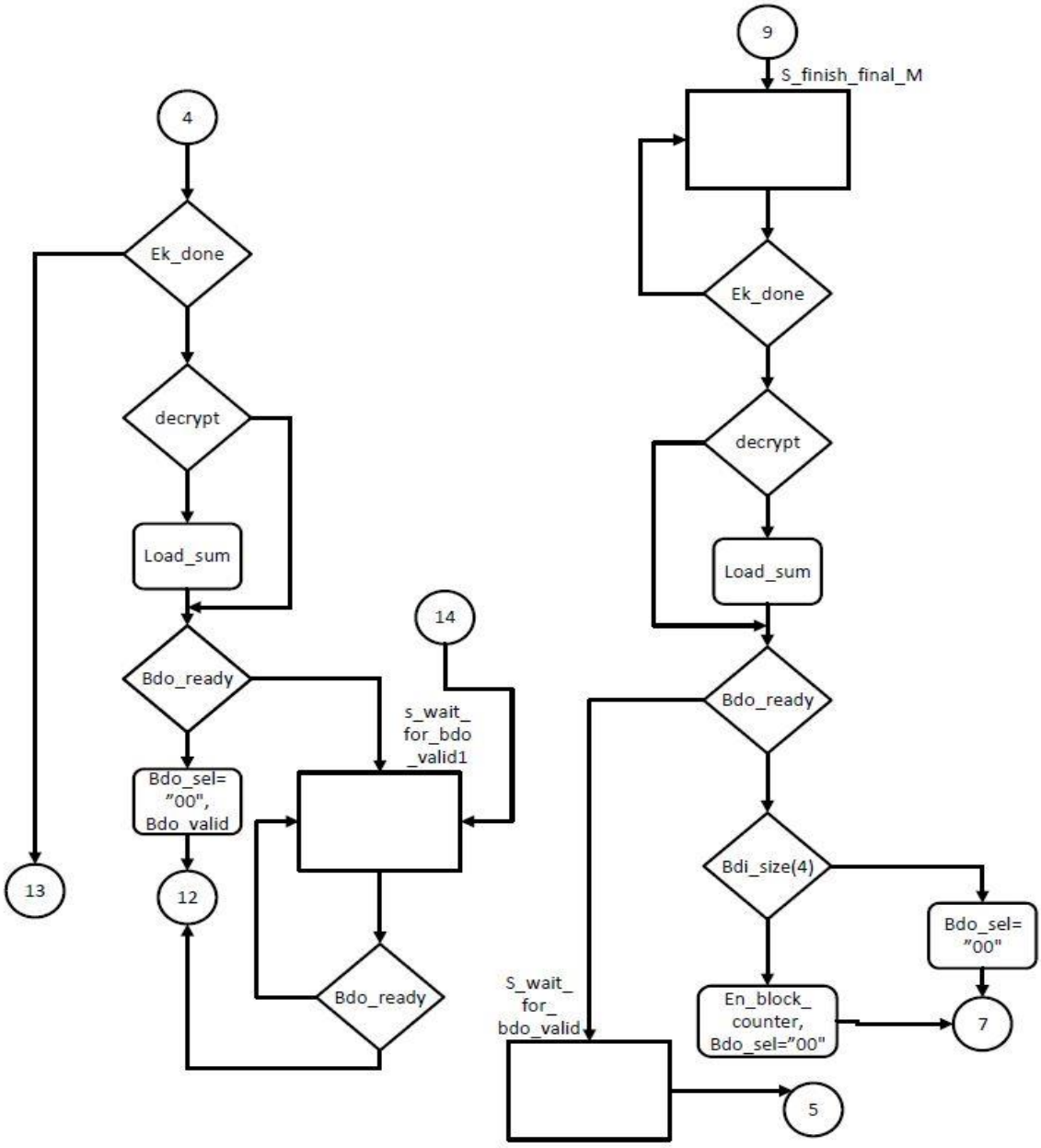

Figure 15: SCREAM Cipher Controller ASM (3)

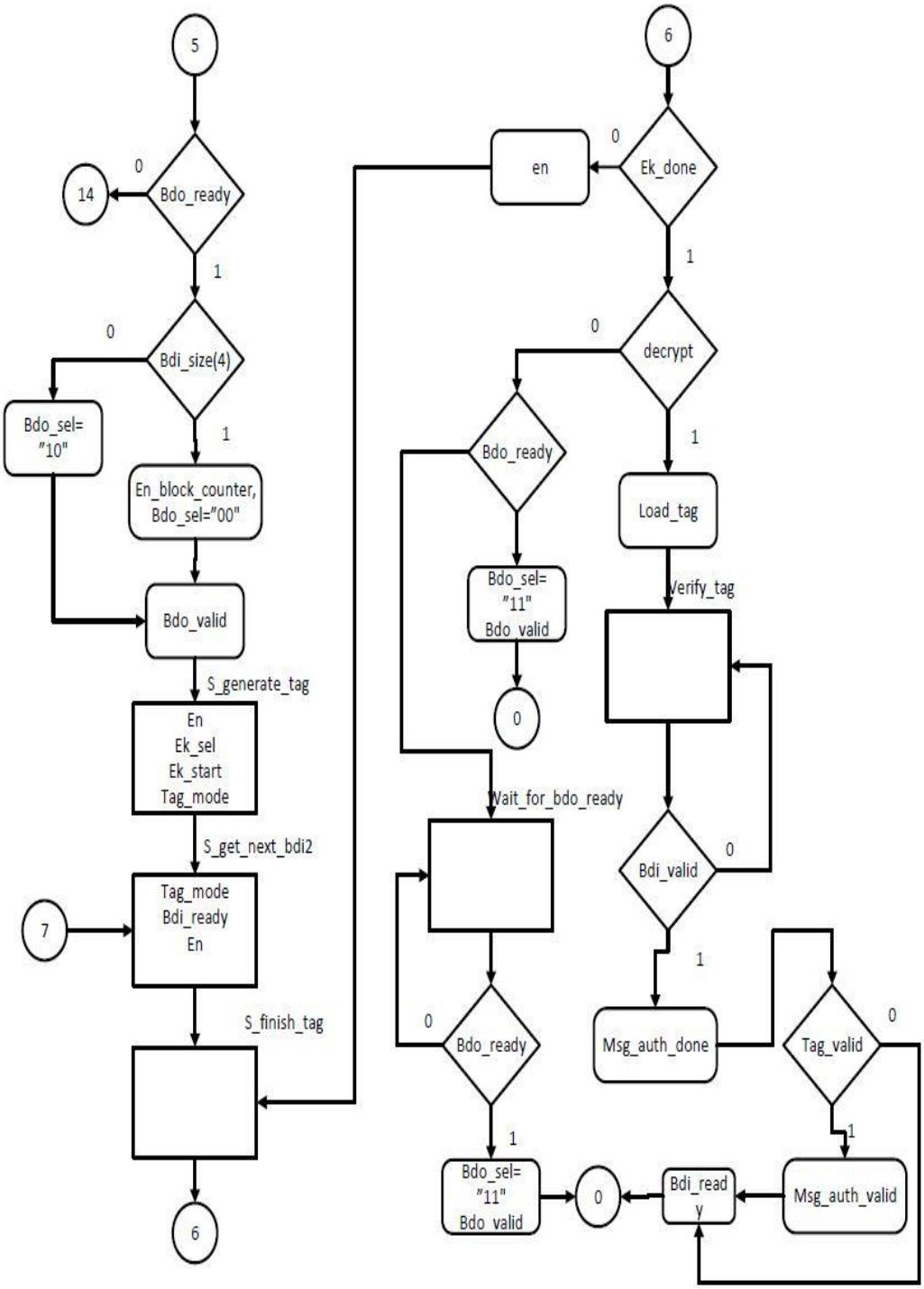

Figure 16: SCREAM Cipher Controller ASM (4)

4.5. Optimized Pipelined Architecture

In the pipelined architecture, the Datapath and Controller design from Basic Iterative Architecture were used as a starting point. The optimized pipelined design can process two 128-bit data blocks at a time. The Datapath design is the same as the Basic Iterative Architecture with a few modifications in bus widths and addition of components. To reduce the critical path, Registers were inserted in the Datapath design of Basic Architecture.

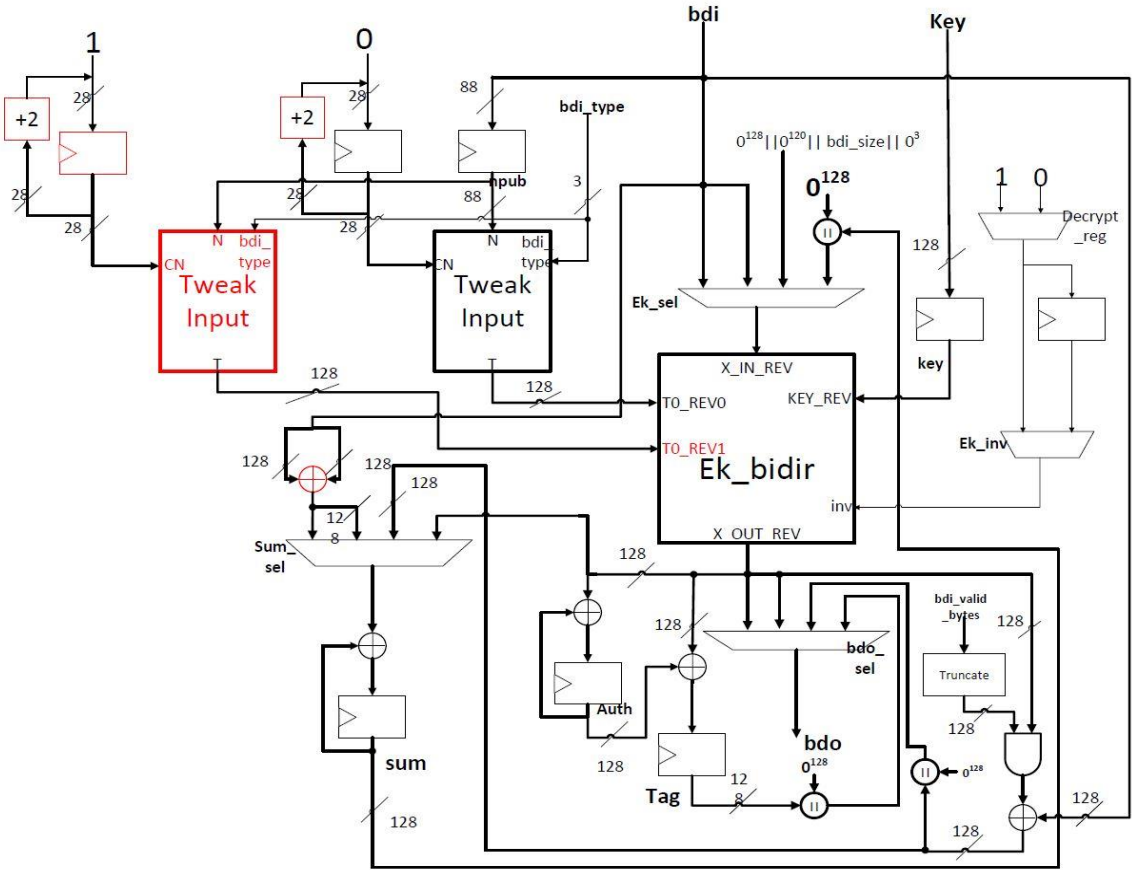

Figure 17: Scream Datapath Pipelined

The critical path of the design lies through the round function of the cipher as it is the longest path. By placing register experimentally at different locations, critical path was tested by calculating Maximum clock frequency.

4.5.1. Register Insertion

Two registers were inserted in the Round function of the cipher, i.e., Ek_Step, one for encryption and one for decryption, as shown in Figure *18*. These registers also help processing two blocks of data in parallel, with modifications in controller shown in Section 3.5.2. They also help with reducing the critical path of the circuit. To process two blocks of data in parallel and to maintain the timing and synchronization, datapath design was also modified as shown in Figure *18*.

4.5.2. Path Balancing

Changes made with respect to the basic iterative architecture are as follows

- One additional Tweak calculator was added, as shown in Figure *17*, which helps in achieving the synchronization, while processing the data.
- The sigma value increases once per two clock cycles, both for encryption and decryption.
- An additional Reverse endian module was added to process second block of data.

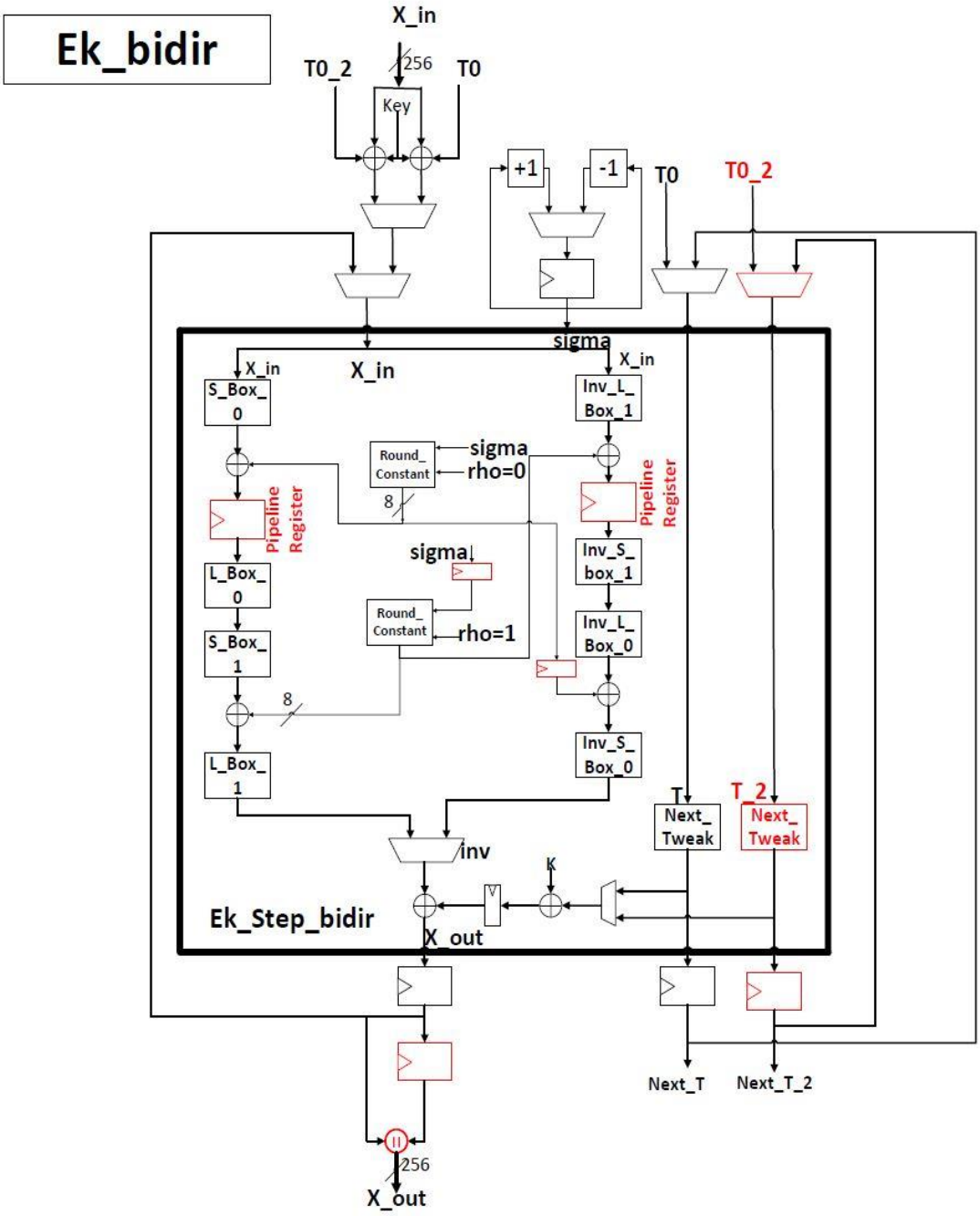

Figure 18: Ek_bidir Pipelined

4.5.3. Controller Modifications

The controller from the basic architecture is modified to meet the timing and synchronize the design. Extra states and three additional signals ‘en1’, ‘en2’, and ‘msb’ were added to the control logic to handle the new block of data as shown in the figure. This design takes 21 clock cycles to complete the processing of two blocks of data. The modified controller is as shown in the Figure 19.

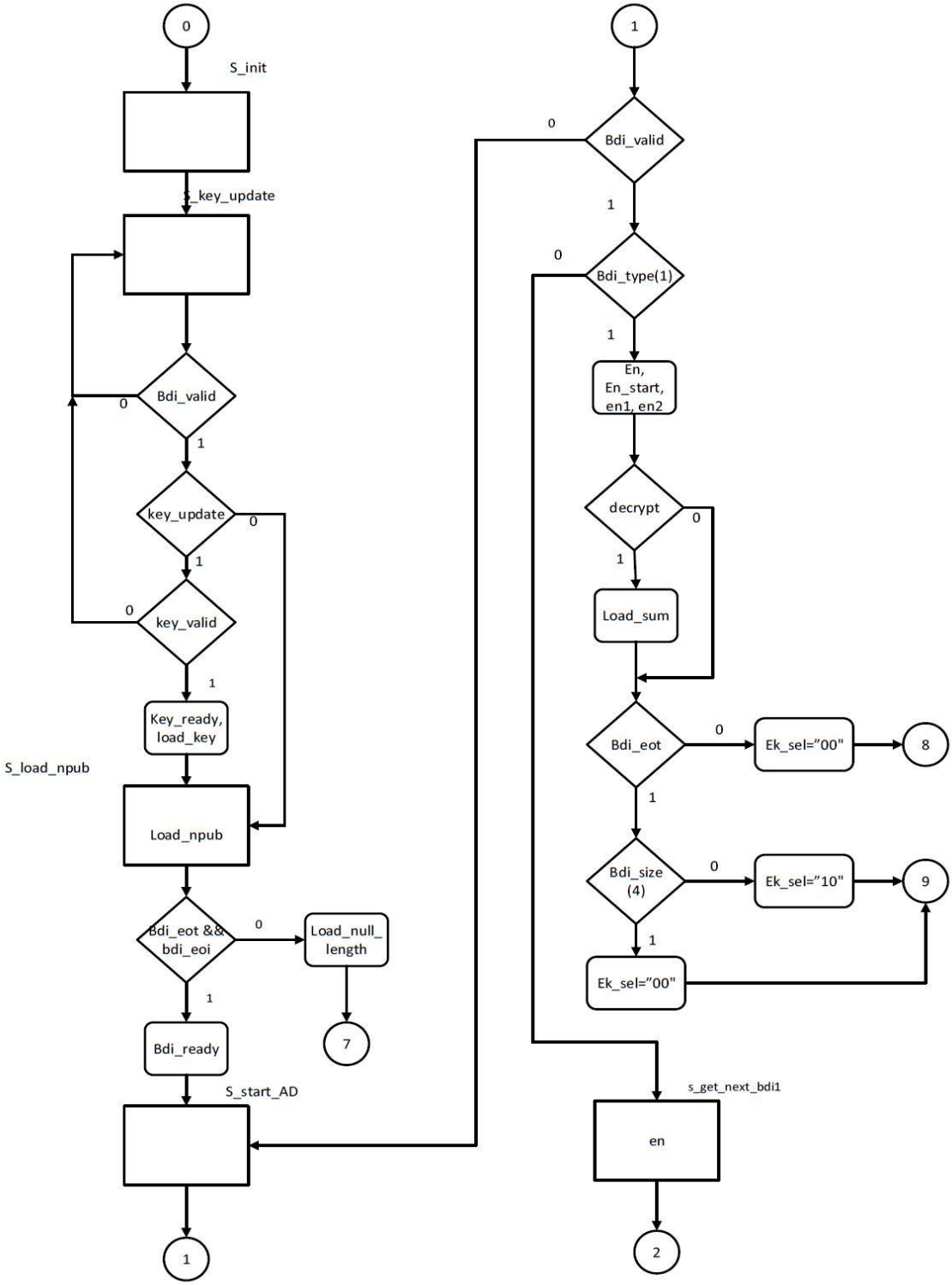

Figure 19: SCREAM Controller Pipelined (1)

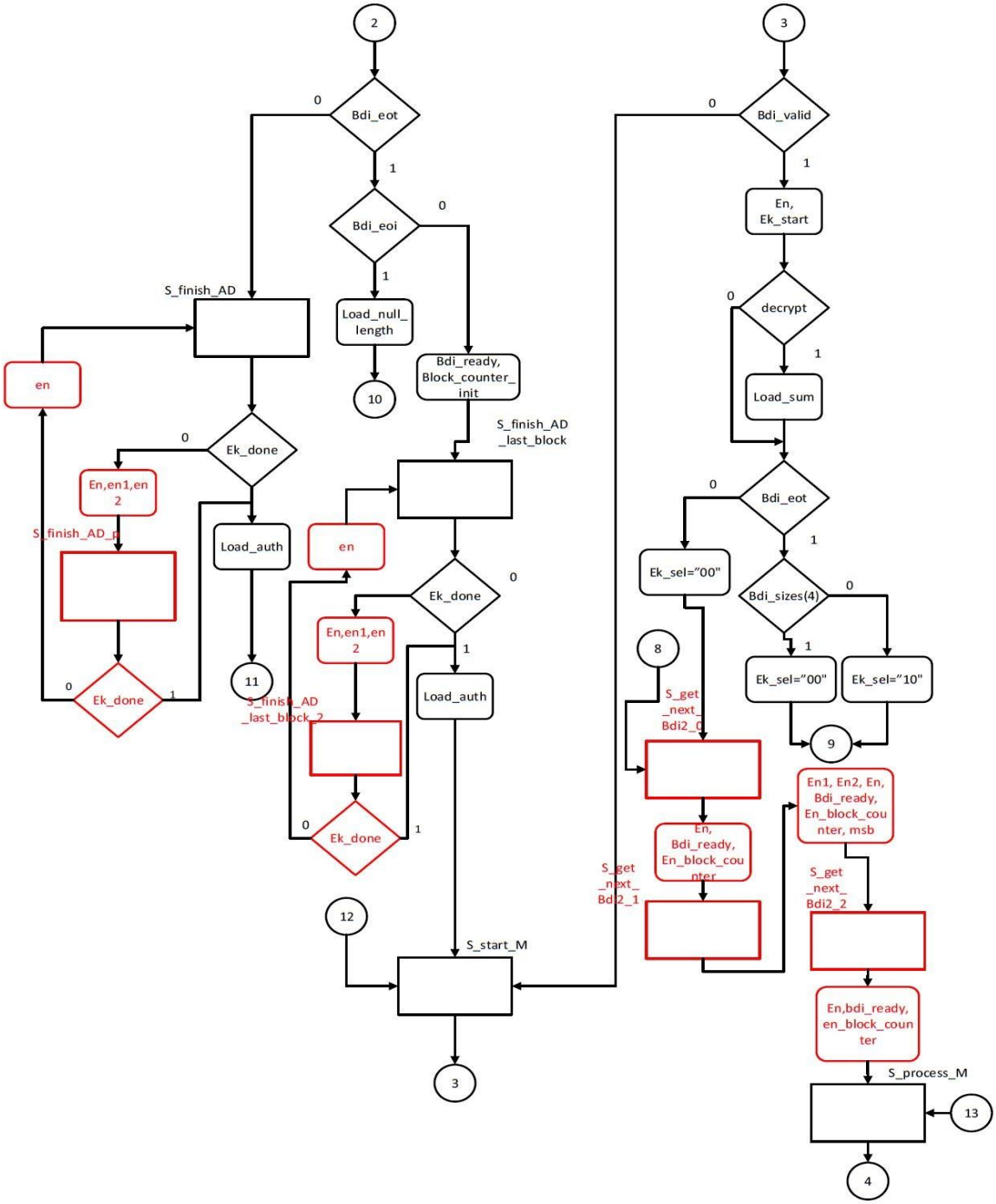

Figure 20: SCREAM Controller Pipelined (2)

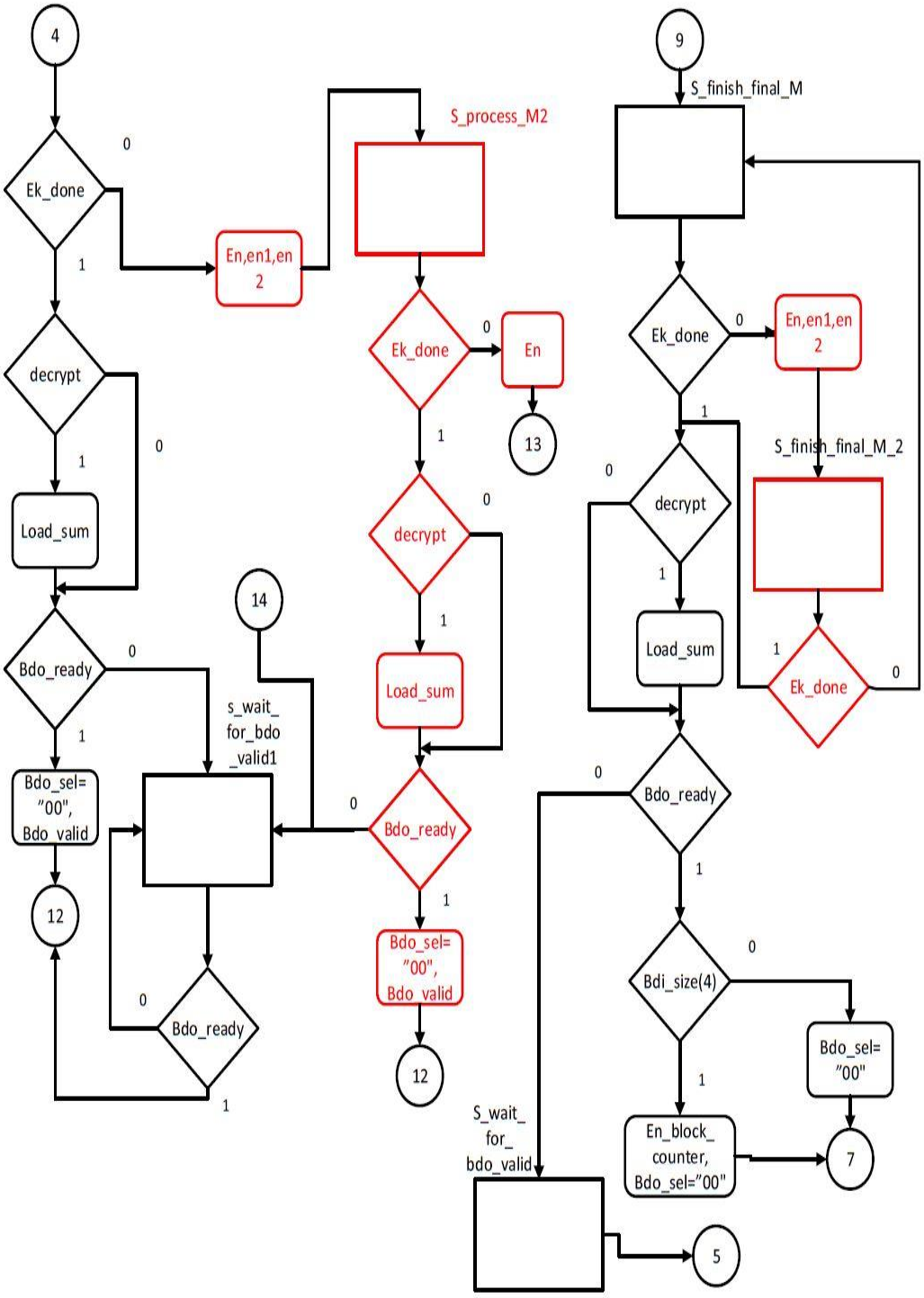

Figure 21: SCREAM Controller Pipelined (3)

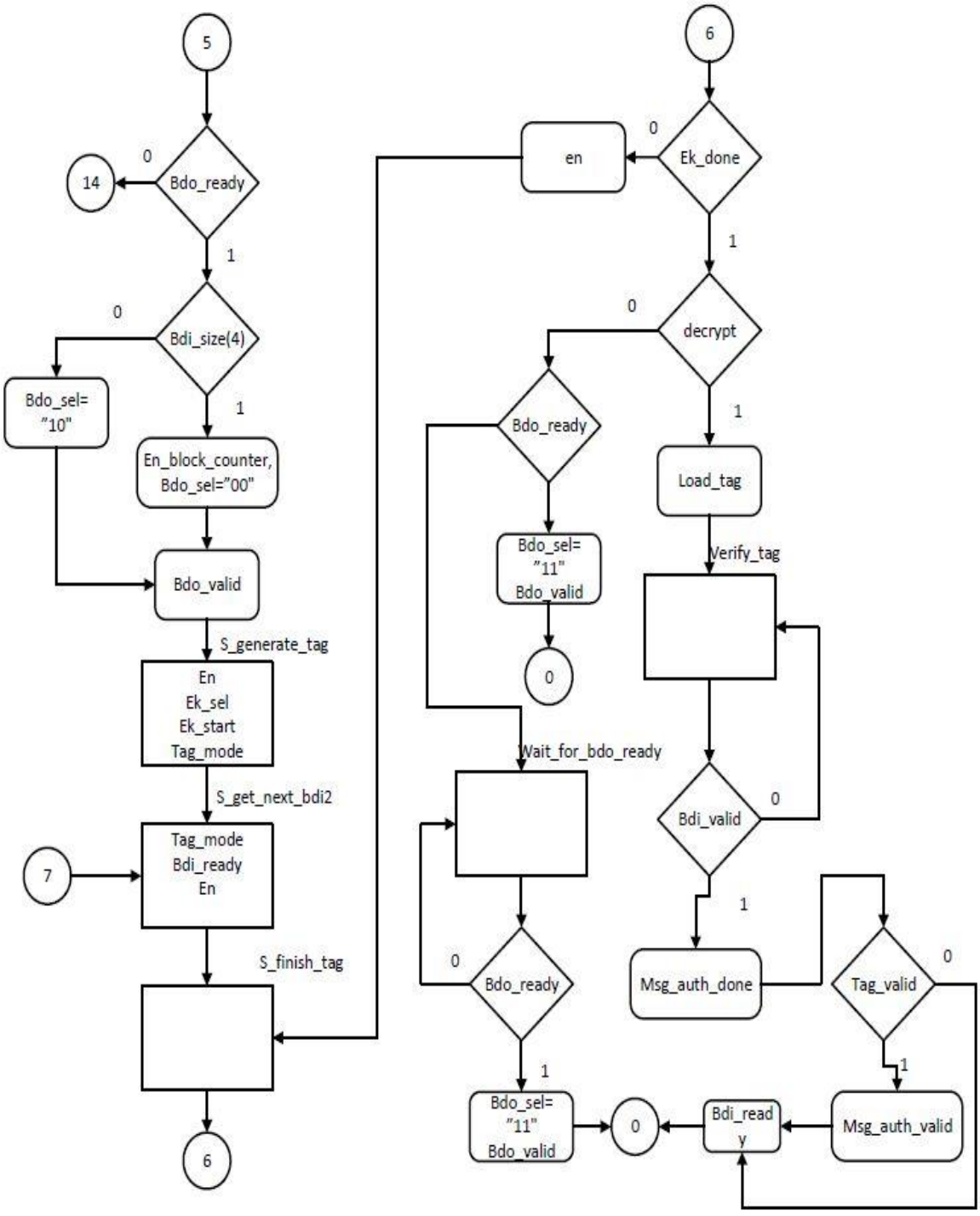

Figure 22: SCREAM Controller Pipelined (4)

5. AES-COPA

5.1. Introduction and Major features

AES-COPA [10] is CAESAR candidate which was designed by Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser and Kan Yasuda. The two important parameters of AES-COPA are key length 'k' which can be either 16 bytes (128 bits), 24 bytes (192 bits), or 32 bytes (256 bits) and tag length 'τ' which lies between bytes (64 bits) and 16 bytes (128 bits). It also has nonce a.k.a public message number of length 16 bytes (128 bits). The key size of AES-COPA is same as the key size of the AES. AES-COPA does not support secret message number, It supports variable length associated data and plaintexts.

5.2. Recommended Parameters:

There is only one recommended parameter set given in the specification.

- key length: 16 byte (128 bits) and tag length: 16 byte (128 bits).

5.3. Encryption and Decryption

The encryption takes public message number 'N', associated data 'A', and message 'M' as input and returns ciphertext 'C' and tag 'T'. The decryption takes public message number 'N', associated data 'A', ciphertext 'C' and tag 'T' as input and returns output 'M' if the tag is correct and otherwise null. Bit 'P' is also supplied as an input in both

encryption and decryption process indicating whether the last block of M was incomplete and thus padding is applied.

First the associated data is processed to get a ‘V’ value as shown in the Figure *23*.

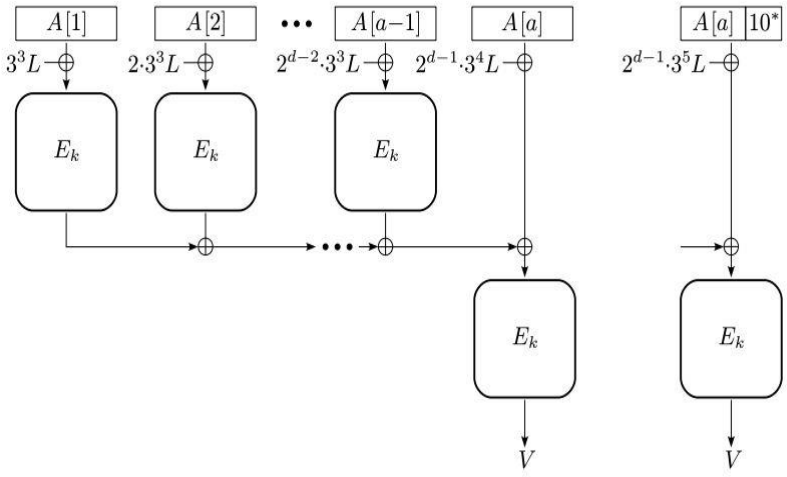

Figure 23: Associated data processing

The encryption and decryption procedures of AES-COPA on such a (possibly padded) message M [1].….M[d] of d 128-bit blocks and on a ciphertext C[1].….C[d] is as shown in the Figure *24*. Each 128 bit block of message/ciphertext is XORed with ‘$2^{(d-1)}$3L’ and then processed through the AES the result is now XORed with ‘V’ and ‘L’ values and then they are processed in AES and the result is then XORed with ‘2^{d}L’ and stored as ciphertext/message. All the intermediate values after first AES block are XORed and stored as ‘S’ value as shown in the Figure *24*, this value is used in the Tag generation.

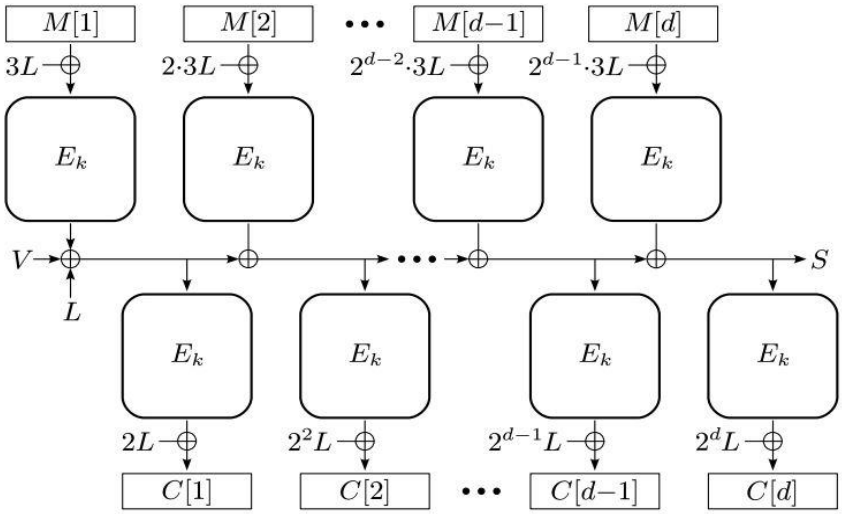

Figure 24: Message/ Ciphertext processing

After encryption/decryption then next step is the tag calculation as shown in the Figure *25*. In this step all the message blocks are XORed with each other and L value and then are processed through the AES block and then the result is then XORed with S value and processed through the AES block and the result is XORed with final L value and then stored as Tag.

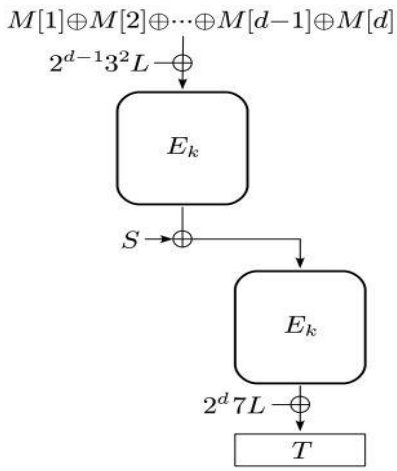

Figure 25: Tag Generation

5.4. Basic High-Speed Architecture

The Basic High-speed Architecture of AES-COPA has been divided in to a Datapath and Controller as shown in the figure below.

5.4.1. Datapath Design

The associated data processing, plain text processing and tag processing requires two Ek(AES) blocks as shown in Figure *27*. So, in Basic High-Speed Architecture design two AES encryption/decryption blocks AES_EncDec1 and AES_EncDec2 (as shown in Figure *30*) were used. The The 'L' value is processed by encrypting a block of zeros through AES_EncDec1 and is stored in a register. Then using the Galois field multiplication blocks, with different multipliers like multi2, multi3 and multi7 the 'L' value is processed as shown in the Figure *26*. The value of L is XORed with the associated

data or plaintext/ciphertext before sending through the AES block. After processing all the associated data blocks through AES_EncDec1 then ‘V’ value is stored in a register. After that, when first block of plaintext/ciphertext enters the AES_EncDec1 and completely processed, AES_EncDec2 sits idle. After the first block is processed through the AES_EncDec1 the result is sent for the further processing through AES_EncDec2. At the same time second block of data is loaded in to the AES_EncDec1. This way parallel processing is done through all the blocks. After processing all the message/ciphertext blocks the tag is calculated using checksum (XOR of all input blocks) value.

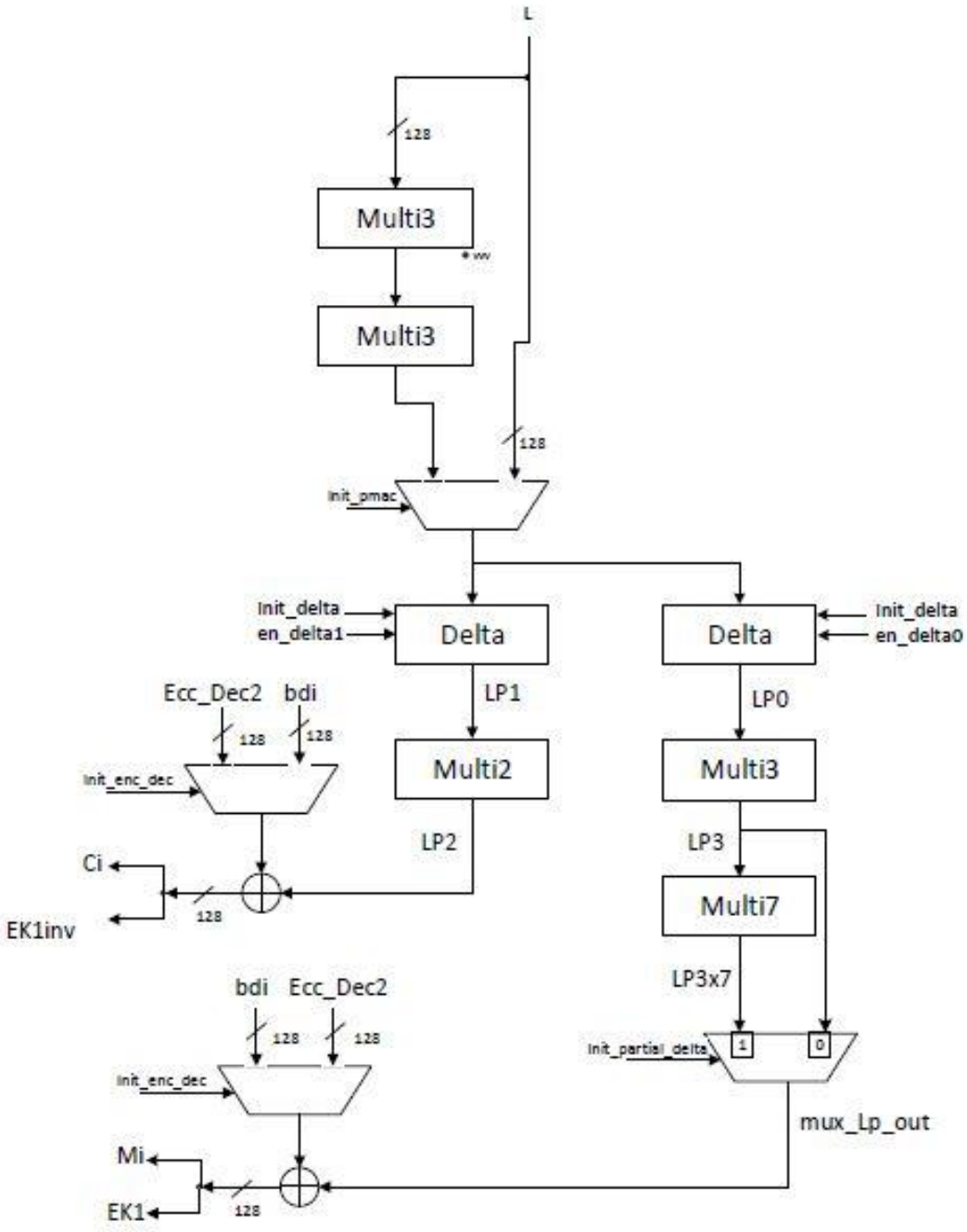

Figure 26: Delta value Calculation

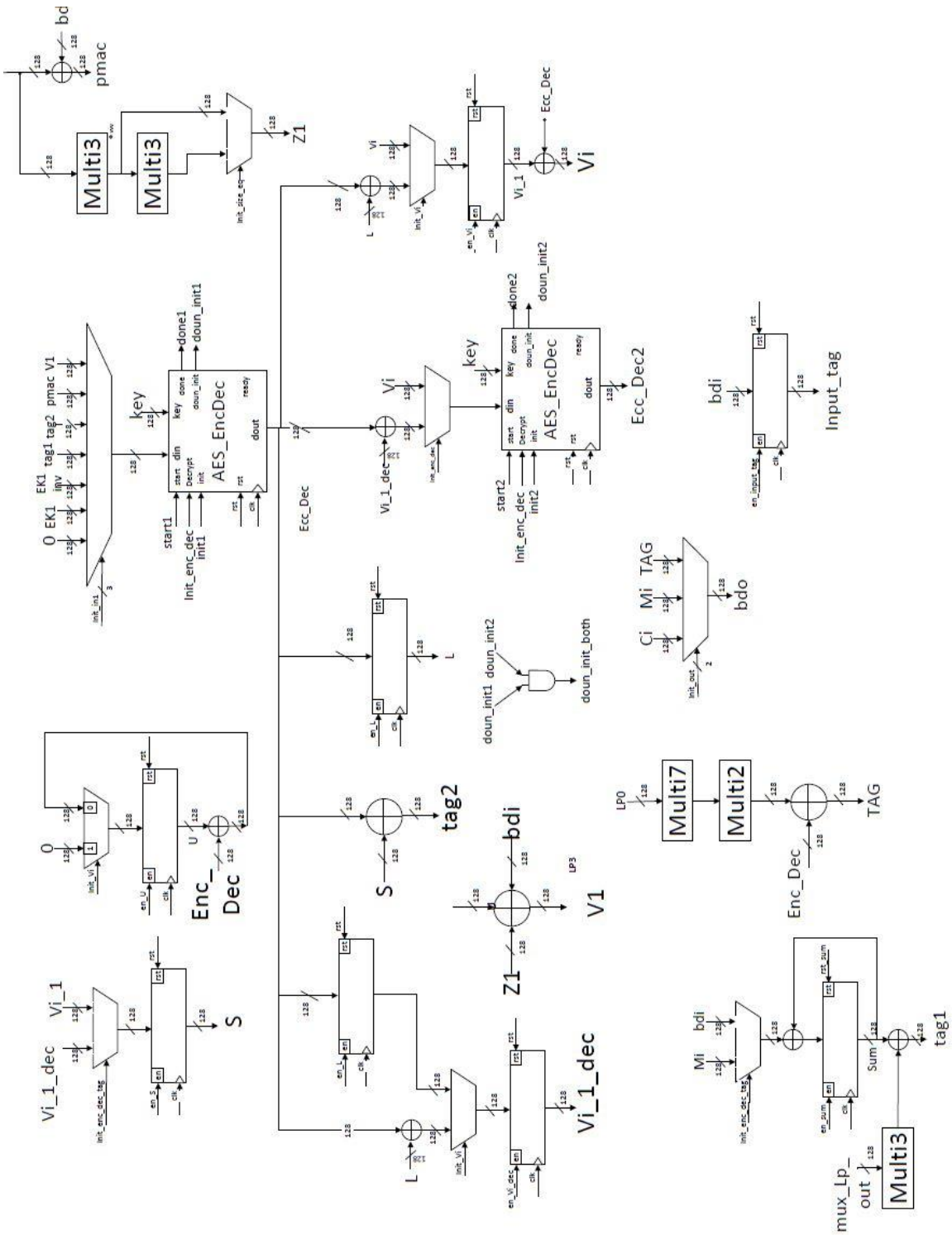

Figure 27: AES-COPA Datapath

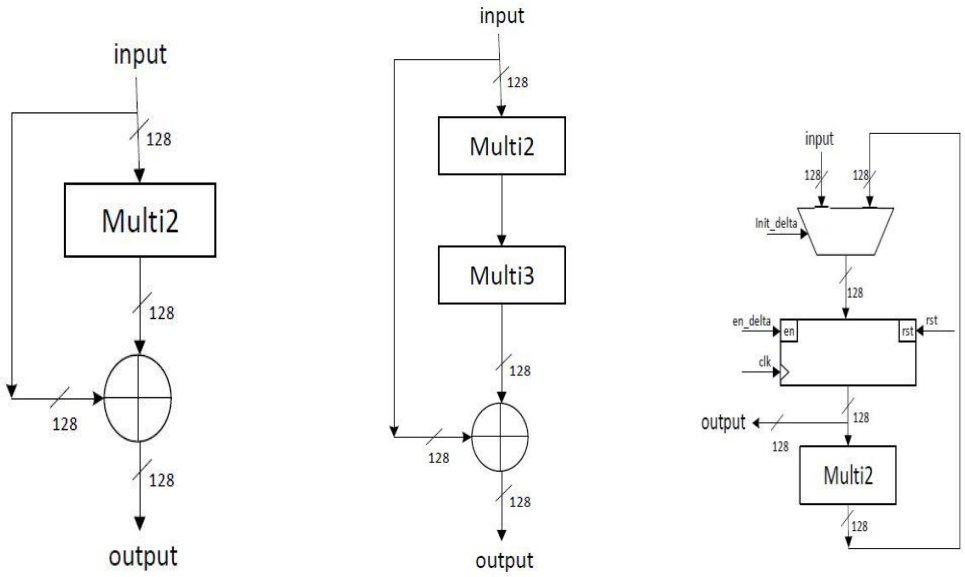

Figure 28: Multi 3, Multi 7, Delta

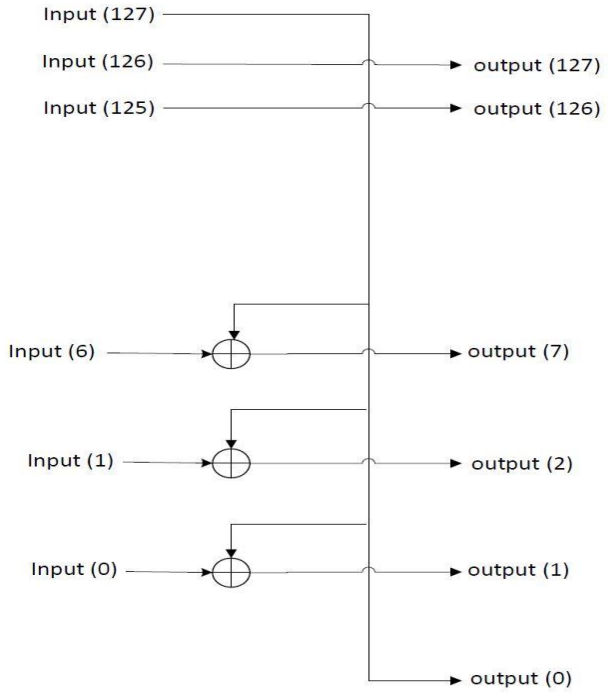

Figure 29: Multi 2

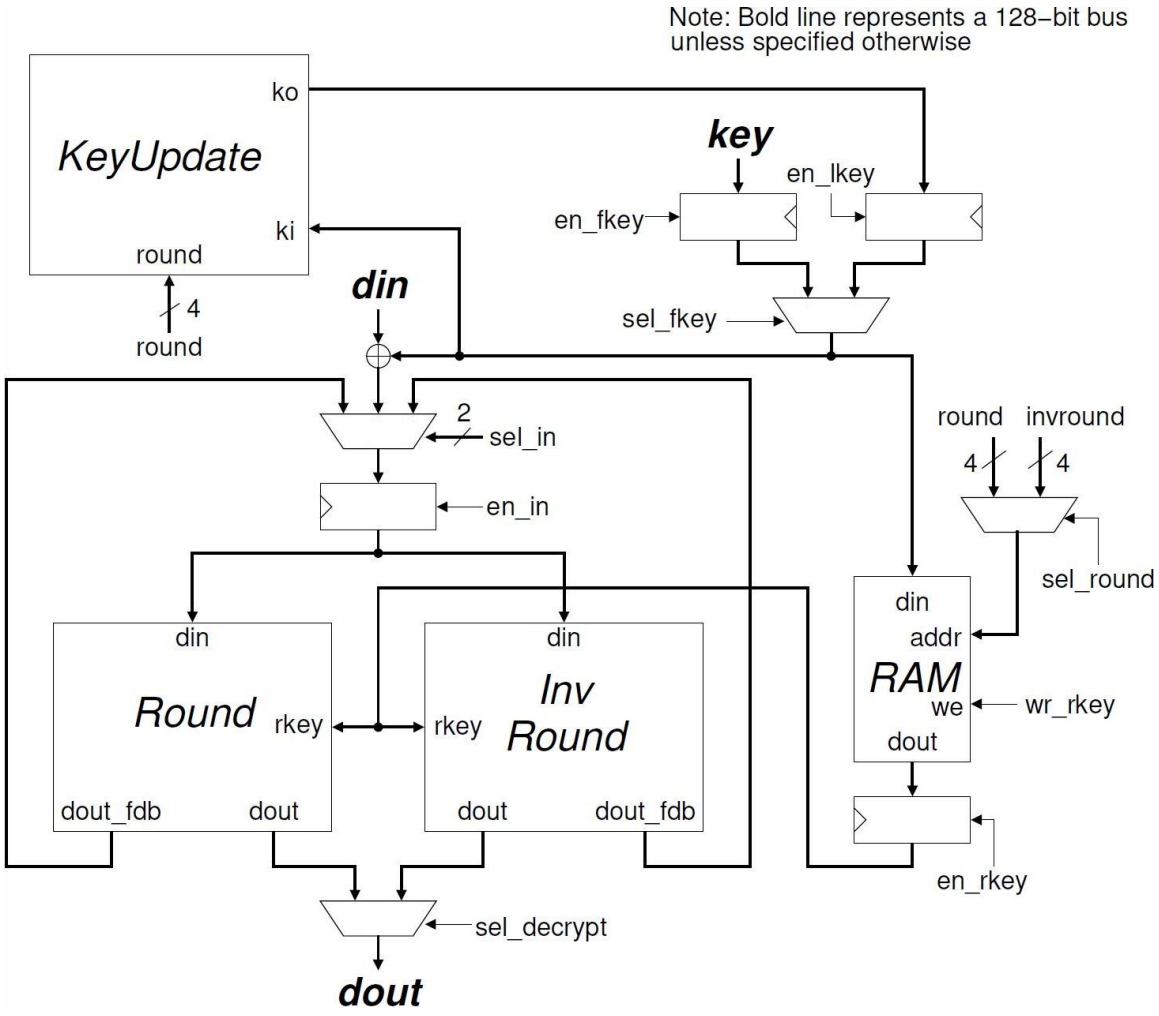

Figure 30: AES Datapath

5.4.2. Controller Design

Upon the reset the controller enters reset state and then if the key needs to be updated then the next state is key_read and then based of key_valid signal Key scheduling is done. After that L value is calculated and delta is initialized. If the key update is not required then the controller enters the associated data processing initialization state where if the bdi_valid is high then associated data processing starts, every time the

done1 signal goes high new block of associated data is processed till it reaches the last block of associated data(i.e. when the bdi_eot goes high). After the associated data processing, the controller enters in to encryption or decryption based on the value of bdi_decrypt. In case of first block of encryption/decryption only the first AES core is started. From the second block parallel processing is done in first AES core and second core as described in AES-COPA datapath design. Every time the done signal goes high new block of message/ciphertext is encrypted/decrypted till it reaches the last block of message/ciphertext after which the controller waits for bdo_ready signal to write the message to output. Once the encryption/decryption state is completed next state is tag generation. In case of encryption the once the tag generation is completed controller waits for bdo_ready to write the message to the output and then enters the reset state whereas tag generation in decryption is slightly different than encryption because we require all the messages for tag generation. So the controller waits for extra 10 clock cycles to complete the processing of last block of data in AES_EncDec2 and then tag generation is started and then enters the tag comparison state where tag is matched with received tag and then controller goes to the reset state.

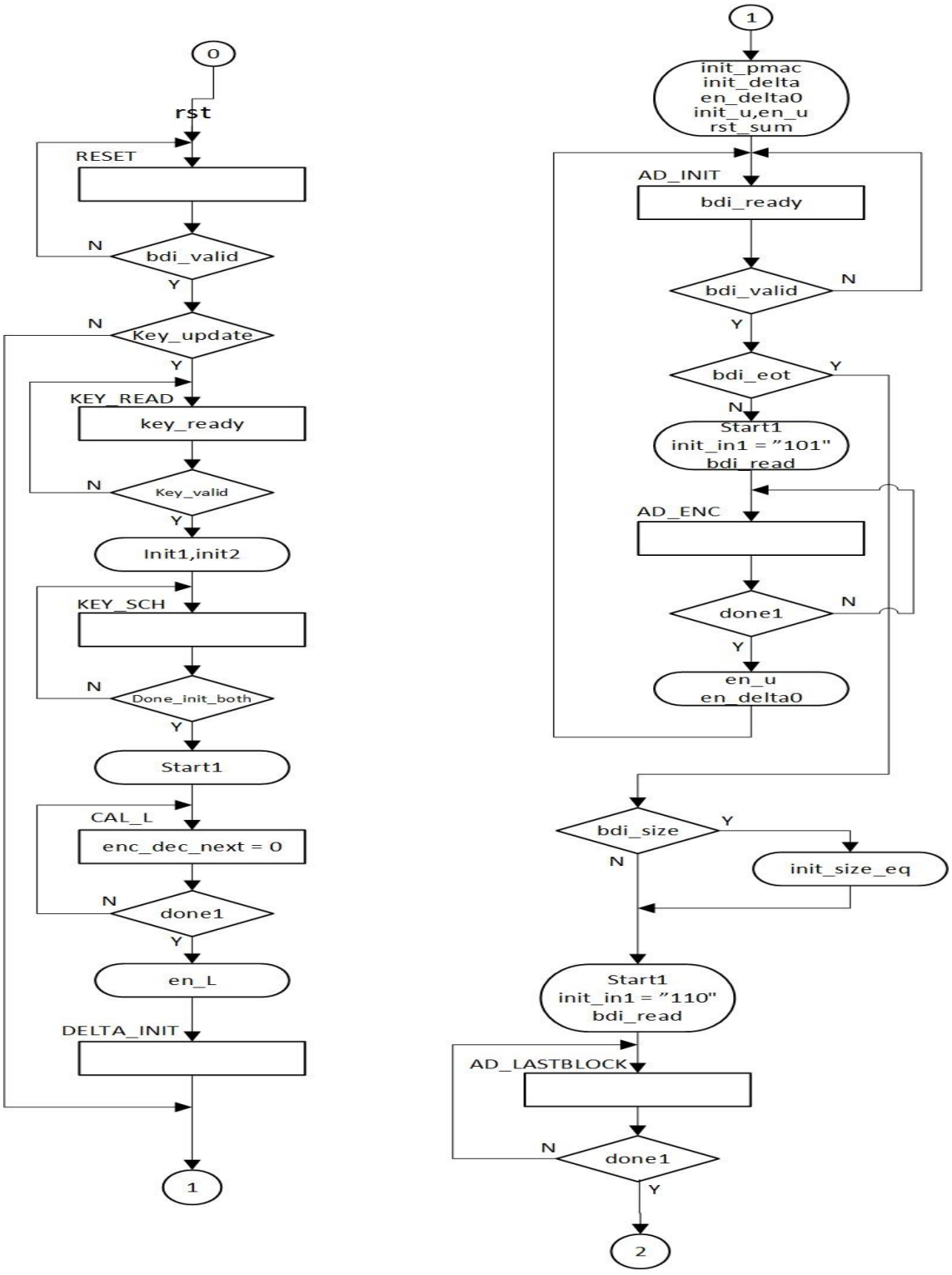

Figure 31: AES-COPA Controller (1)

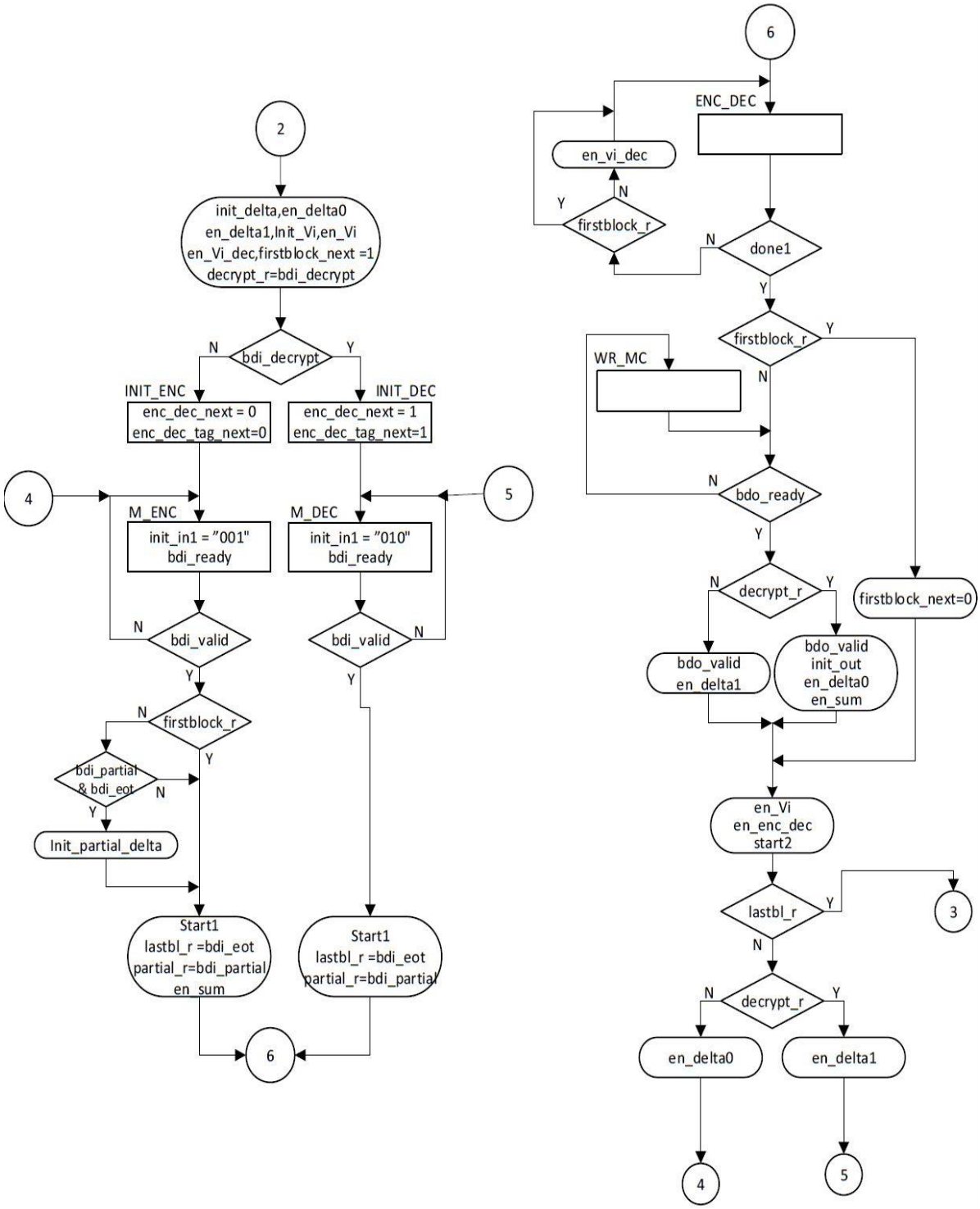

Figure 32: AES-COPA Controller (2)

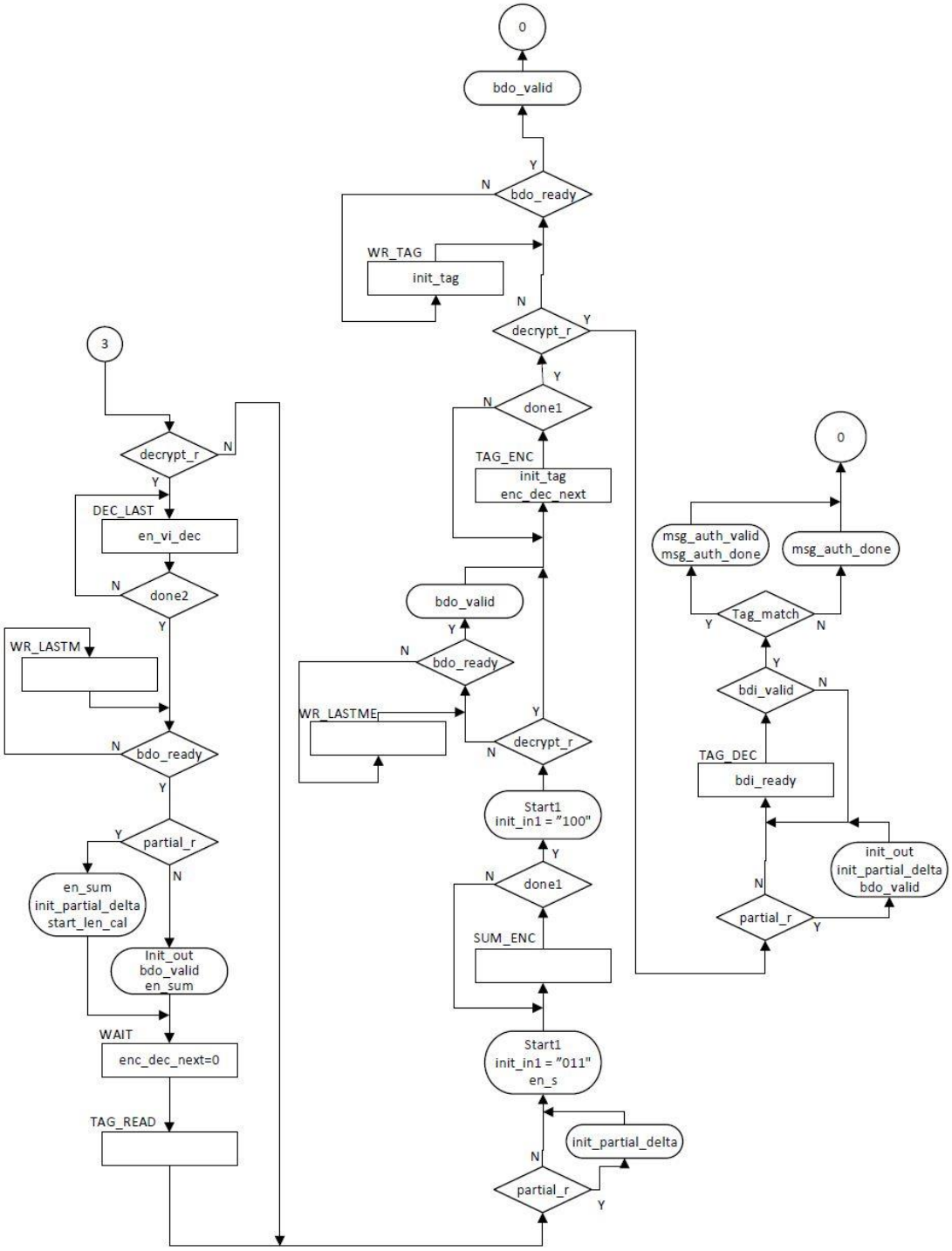

Figure 33: AES-COPA Controller (3)

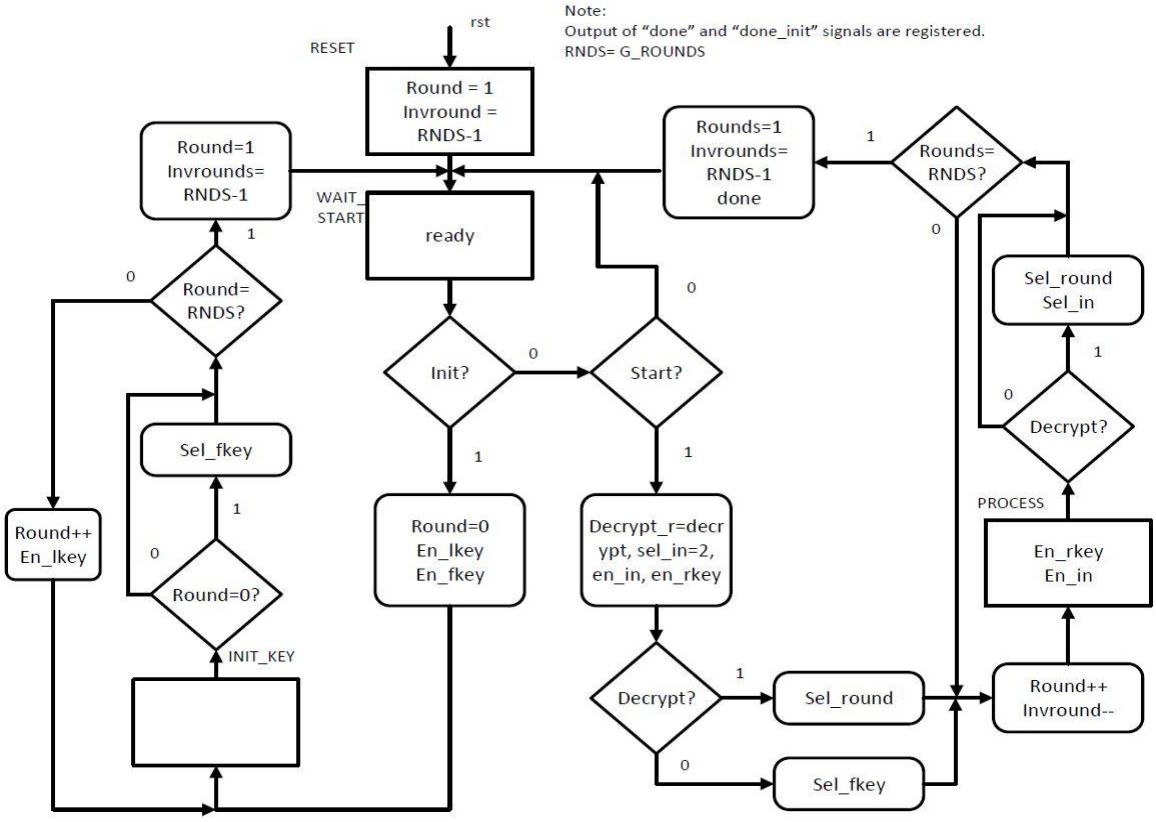

Figure 34: AES Controller

5.5. Optimized Pipelined Architecture

In the pipelined architecture, the Datapath and Controller Design from Basic Iterative Architecture were used as a starting point. The optimized pipelined design can process two 128-bit data blocks at a time. The Datapath design is same as the Basic Iterative Architecture with few modification in bus widths and addition of components as shown in the Figure *35*. To reduce the critical path, Registers were inserted in the Datapath design of Basic Architecture. The critical path of the design lies in the round function of the AES as it is the longest path.

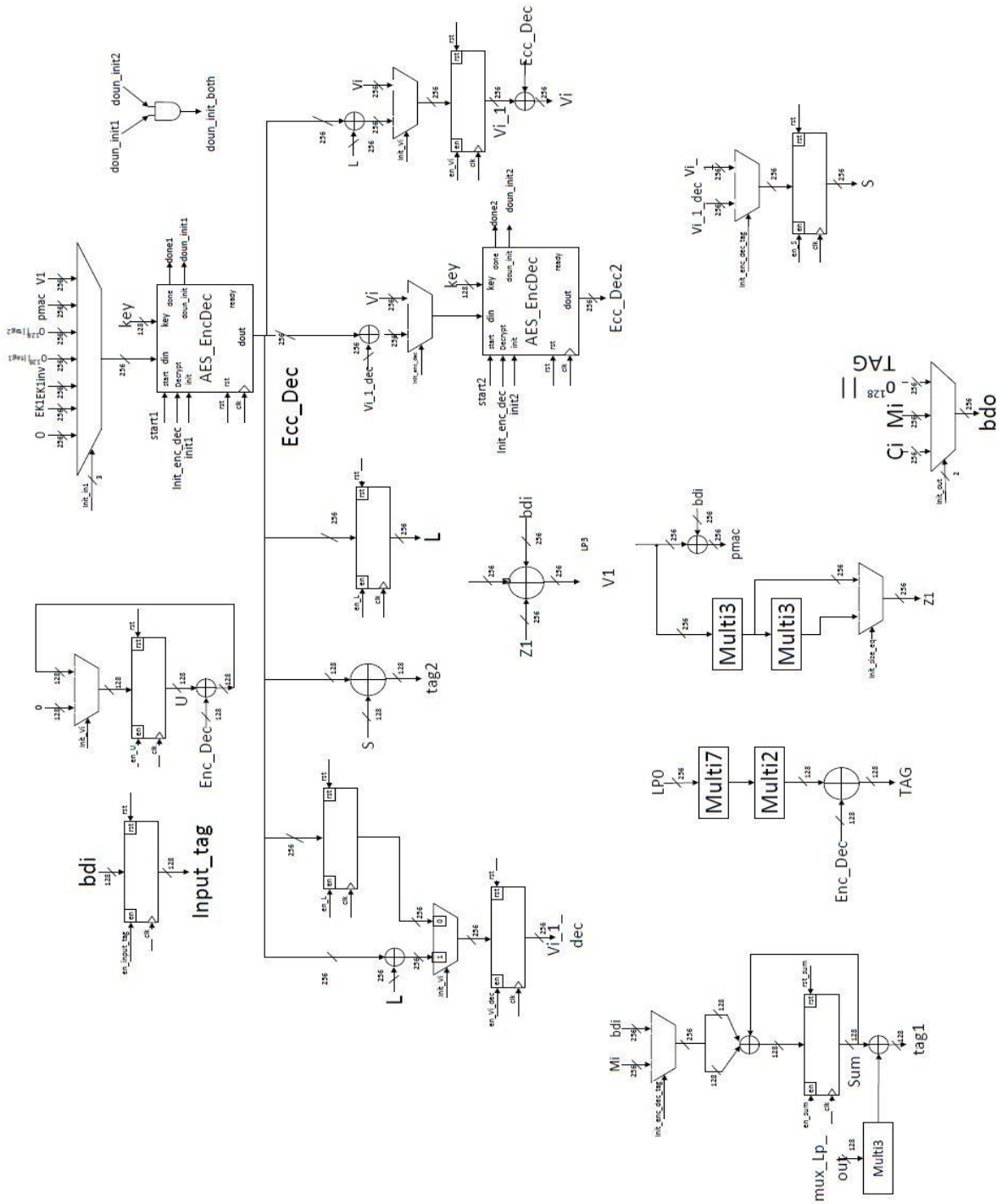

Figure 35: AES-COPA Datapath Pipelined

5.5.1. Register Insertion

Two registers were inserted in the round function based on the critical path from basic high-speed architecture, one register in round function and the other register in the invround function of AES_EncDec in the *Figure 36*. The register insertion reduced the critical path and increased the Maximum clock frequency.

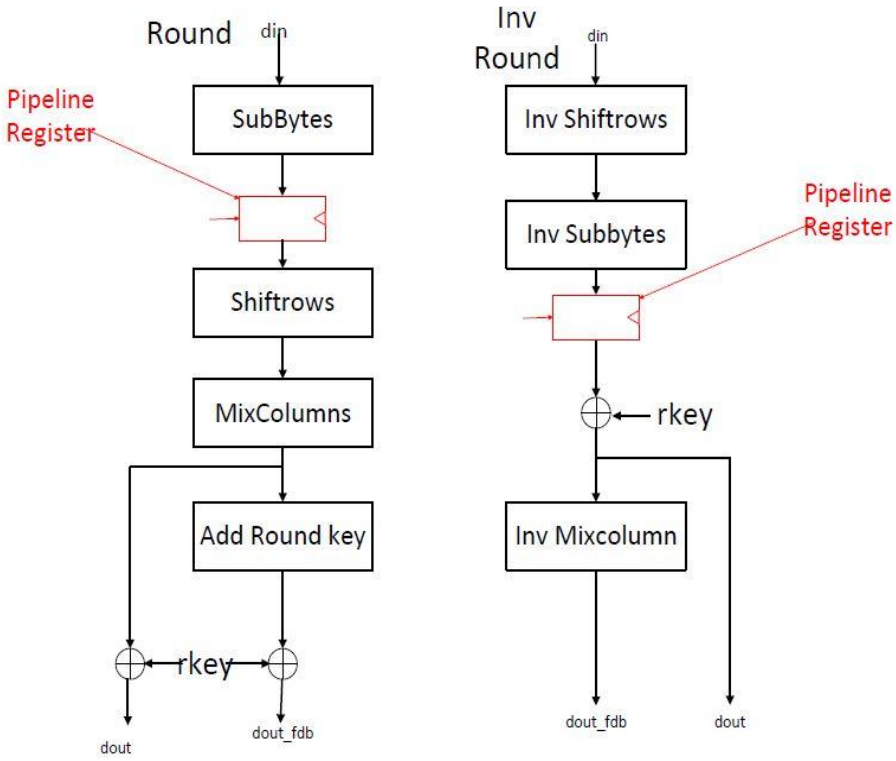

Figure 36: AES Round Pipelined

5.5.2. Path Balancing

The path was balanced by changing the widths of the buses to 256 bits as shown in the Figure 35 *and additional Galois field multipliers were added in the datapath to generate the '$2^{(d-1)}3L$' value for the second block of data as shown* Figure 29*. The AES_EncDec datapath was also modified to support two blocks of data as shown in the*

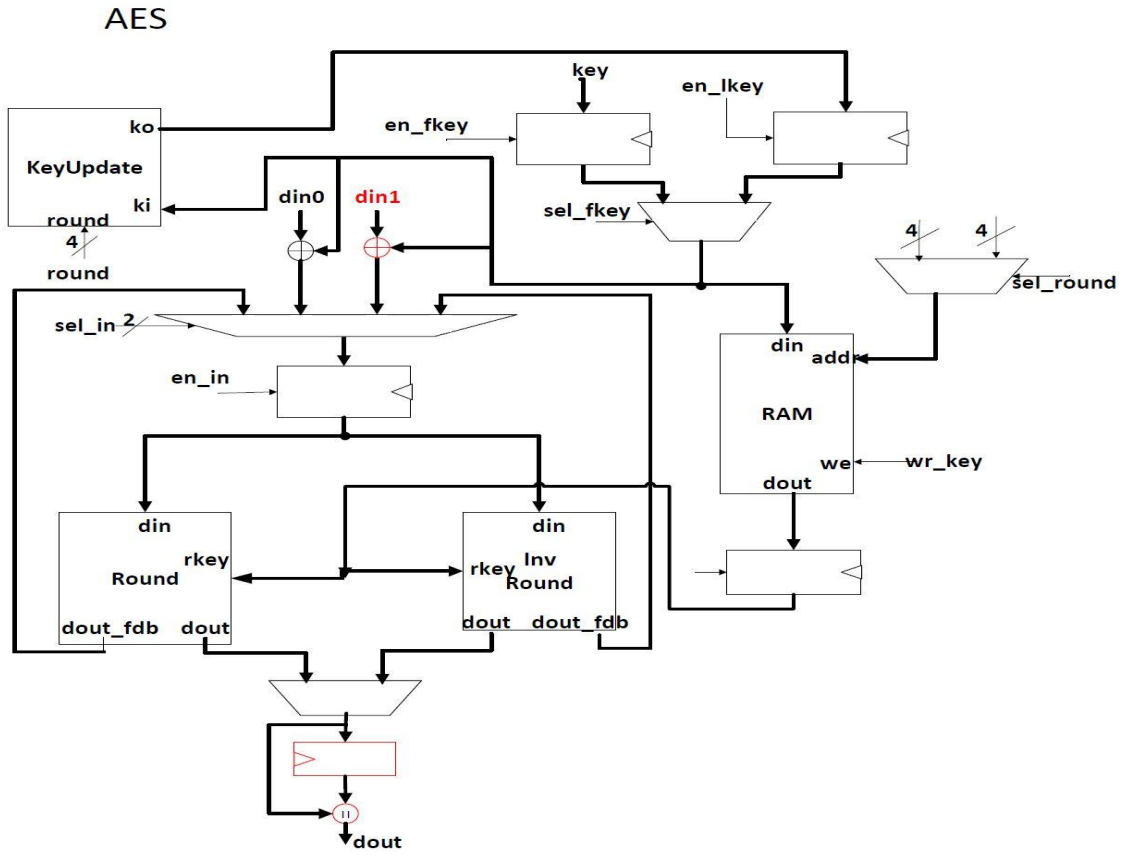

Figure *37*.

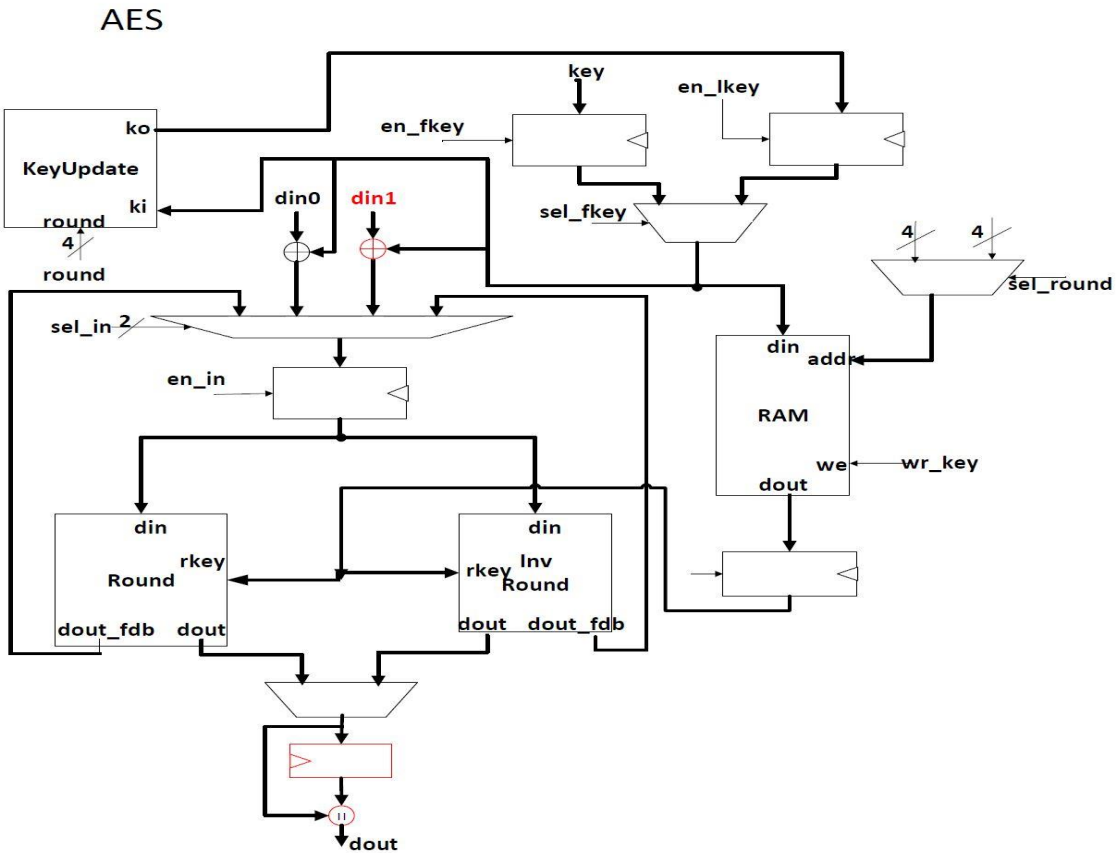

Figure 37: AES Datapath Pipelined

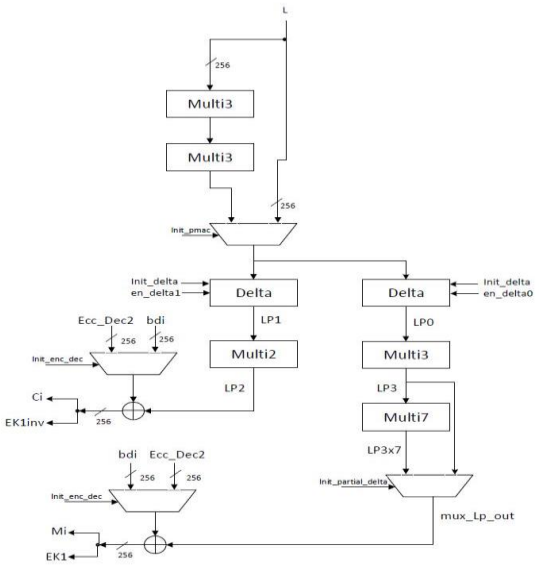

Figure 38: Delta Value calculation Pipelined

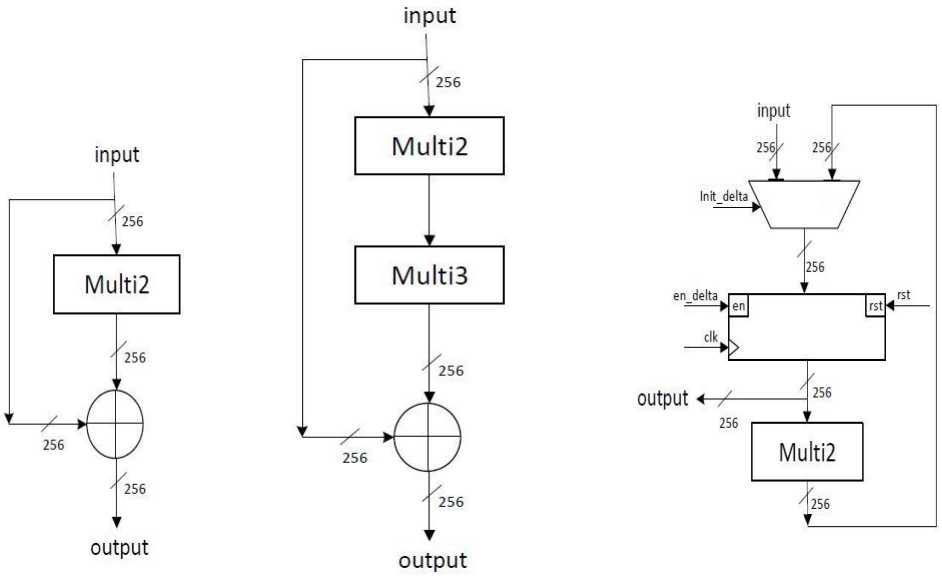

Figure 39: Multi 3, Multi 7, Delta –Pipelined.

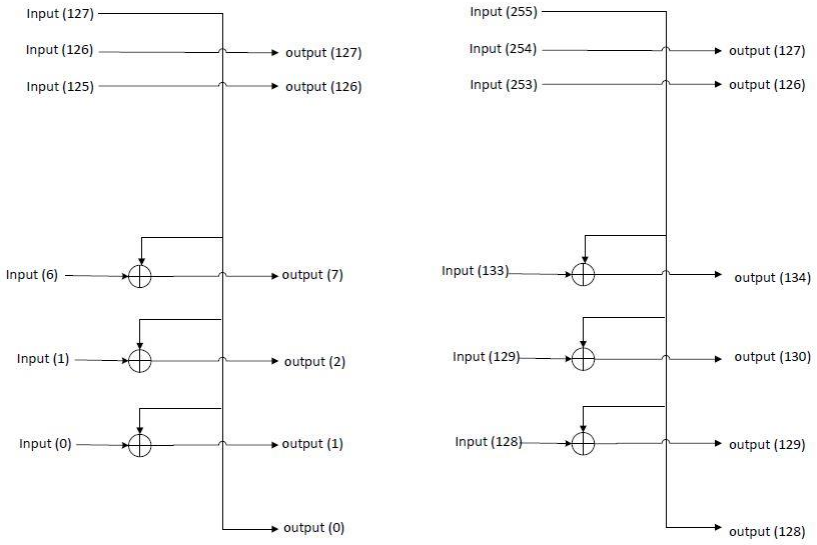

Figure 40: Multi 2 pipelined

5.5.3. Controller Modifications

The top-level controller was slightly modified to support the $(d-1)^{th}$ block of associated data(from Figure *23*) as shown in the figure. The flow of the main controller is mainly dependent on the done signals from two AES cores done1 and done2 for processing data. So the changes in the AES_EncDec controller were made accordingly to balance the path.

Changes in the controller of AES_EncDec:

The Key scheduling in AES_EncDec remained the same but the data processing was changed by adding four additional states. Three states were added to get the data ready entering the round function and an additional state was added to process two blocks of data at the same time. The round value increases by one per two clock cycles and same round key is available for two clock cycles. The modified controller ASM chart is as shown in the Figure *41*.

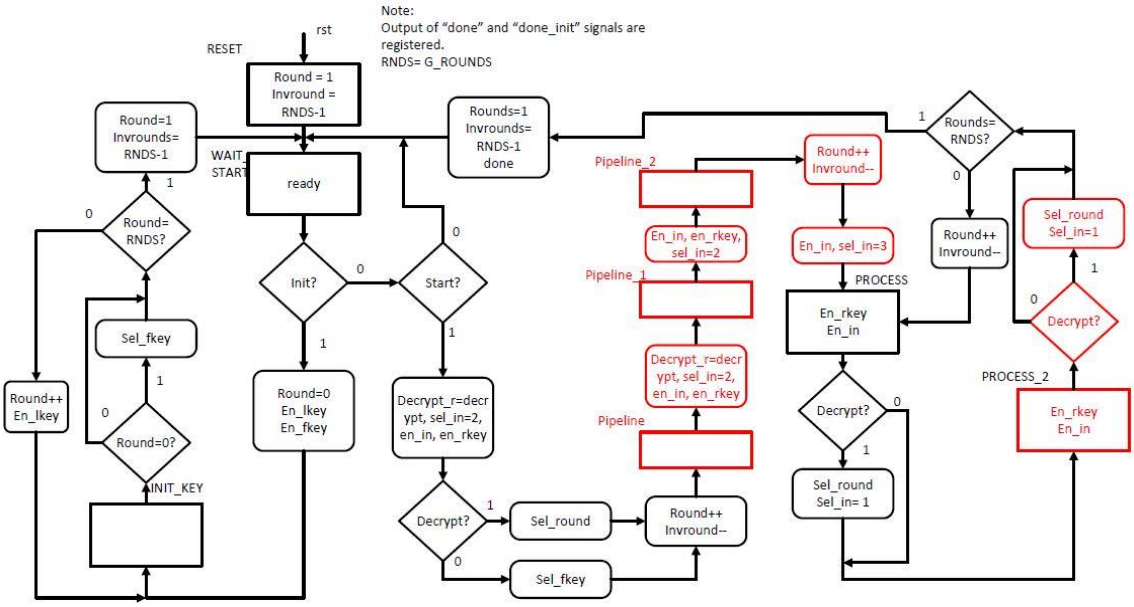

Figure 41: AES Controller Pipelined

6. Minalpher

6.1. Introduction and Major Features

Minalpher [11] was submitted by NTT Secure Platform Laboratories, Mitsubishi Electric Corporation, University of Fukui in CAESAR competition. It was designed by Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, Shoichi Hirose.

Minalpher supports two functionalities

i. Authenticated Encryption and Associated Data (AEAD)

ii. Message Authentication Code (MAC)

The construction is based on Kurosawa's Tweakable block-cipher [3][4] from a permutation based block-cipher by Even and Mansour[2].

Tweakable Even-Mansour

The Tweakable Even-Mansour is a Tweakable block cipher which is based on permutation denoted by P. It operates on n bits such that n mod 2 = 0. Inverse of P is denote by P^{-1}. Modes of operation of Minalpher are based on Tweakable Even-Mansour. The encryption and decryption algorithms are denoted by TEM_ENC and TEM_DEC respectively. These algorithms are also useful in tag generation.

6.2. Recommended Parameters:

Recommended parameter set given in the specification.

- key length: 16 byte (128 bits) and tag length: 16 byte (128 bits).

6.3. Encryption and Decryption

The functionality followed for the encryption and decryption is AEAD. It consists of two algorithms i.e. AEAD_ENC which is encryption algorithm and AEAD_DEC which is decryption algorithm.

AEAD Encryption: The inputs to AEAD_ENC are a secret key ‘K’, nonce ‘N’, associated data ‘A’ and message ‘M’ in return it gives the ciphertext ‘C’, tag or a reject symbol as the output.

AEAD Decryption: The input to AEAD_DEC are a secret key ‘K’, nonce ‘N’, associated data ‘A’ ciphertext ‘C’ and tag in return it gives message ‘M’ or the reject symbol as the output.

Both AEAD_ENC and AEAD_DEC does not accept inputs that do not satisfy the following conditions and return reject symbol as output.

- a secret key, denoted by ‘K’, such that $K \in \{0, 1\}^{n/2}$,
- a nonce, denoted by ‘N’, such that $N \in \{0, 1\}^{n/2-s}$,
- associated data, denoted by ‘A’, such that $A \in \{0, 1\}^{*}$,
- a message, denoted by ‘M’, such that $M \in \{0, 1\}^{*}$,
- a ciphertext, denoted by ‘C’, such that $|C| \bmod n = 0$ and $|C| \geq n$, and
- a tag, denoted by tag, such that $|tag| = l$,

Where n=256 and s= 24.

Minalpher-P:

Minalpher-P is a primitive permutation function which maps 256-bit input value to a 256-bit output value. The P block in Figure *43* is Minalpher P. Minalpher P has round function R which consists of a four permutation functions as shown.

- S-function
- T-function
- M-function
- E-function

Number of rounds is denoted by ‘r’ where r = 17(recommended). Let the number of rounds be r + 0.5. The following are performed from i= 1 to r. The input to the function is as shown

$$Xi <= R(X_{i-1}, E(i-1))$$

R(X, E) is the round function and is denoted by

$$R(X, E(i)) <= M \circ T \circ S(X) \oplus E(i)$$

Last round(half round) is as follows

$$X_{r+1} <= T \circ S(X_r)$$

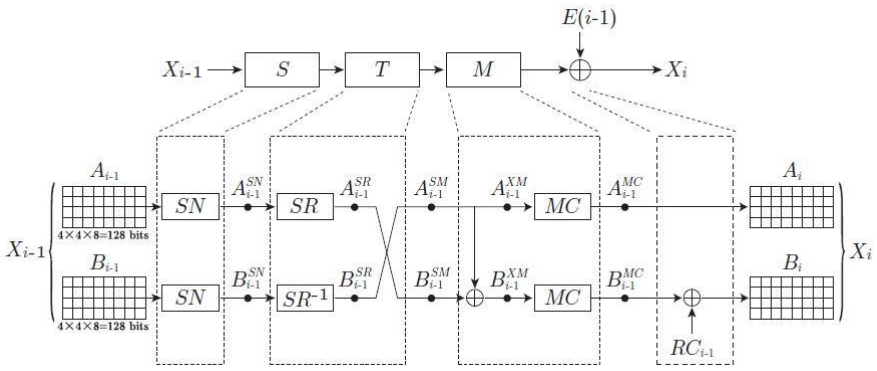

Figure 42: Round Function

SN in the Figure *42* stands for subnibbles where each nibble in the state into another value by using 4-bit S-box where s is the permutation. SR is the shufflerows which shuffles the nibble positions within each row. SR consists of 2 different shuffle functions SR1 and SR2 with different set of permutations. SM is swap matrices which swaps the matrix Ai for the matrix Bi as shown in Figure *42*. XM is XOR matrix where matrix A is XORed with matrix B and assigned to B_{xm}. MC is mixcolumns which is a linear function which can be expressed as multiplication by the following matrix.

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Then there is addition of round constant RC_{i-1} calculated from the round number i and its matrix.

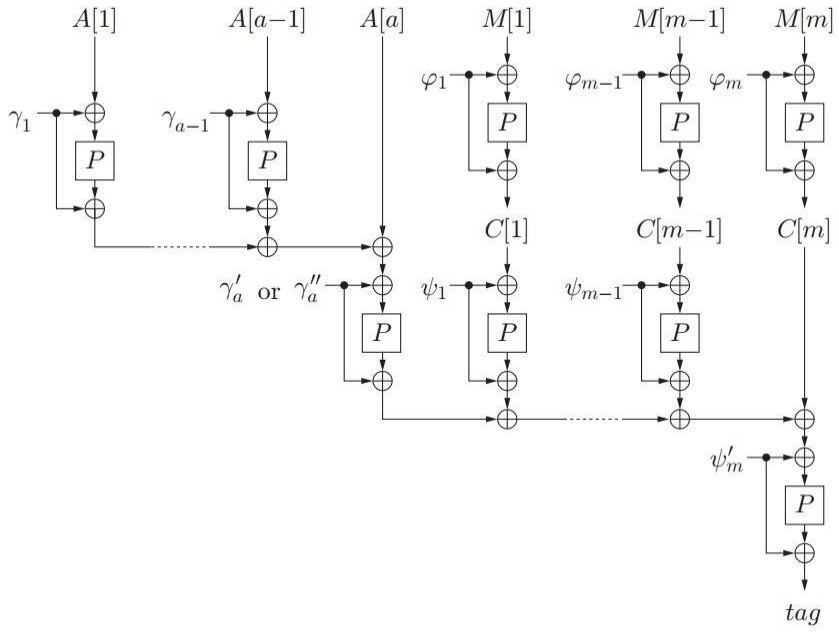

Figure 43: Associated data and plaintext processing.

6.4. Basic High-Speed Architecture

6.4.1. Datapath Design:

The Datapath design of Minalpher consists of two TEM cores TEM_M and TEM_AUX which acts as the permutation block as shown in the Figure *44*. The values of message, ciphertext, associated data, public message number and key are registered before processing them. Firstly the value of L is calculated by concatenating the key, flag and nonce value and then encrypting it and then the result is processed through galois field multiplication and the result is stored as ‘t’. The tweak value is calculated based on the values of ‘i’ and ‘j’ per each block of data. The associated data blocks are processed

through the TEM_AUX block and stored. After that message blocks are processed through the TEM_M blocks. After each message block is processed through the TEM_M, then the resulting value is sent through the TEM_AUX for the calculation of tag. After all the message blocks are processed then the value of tag is calculated.

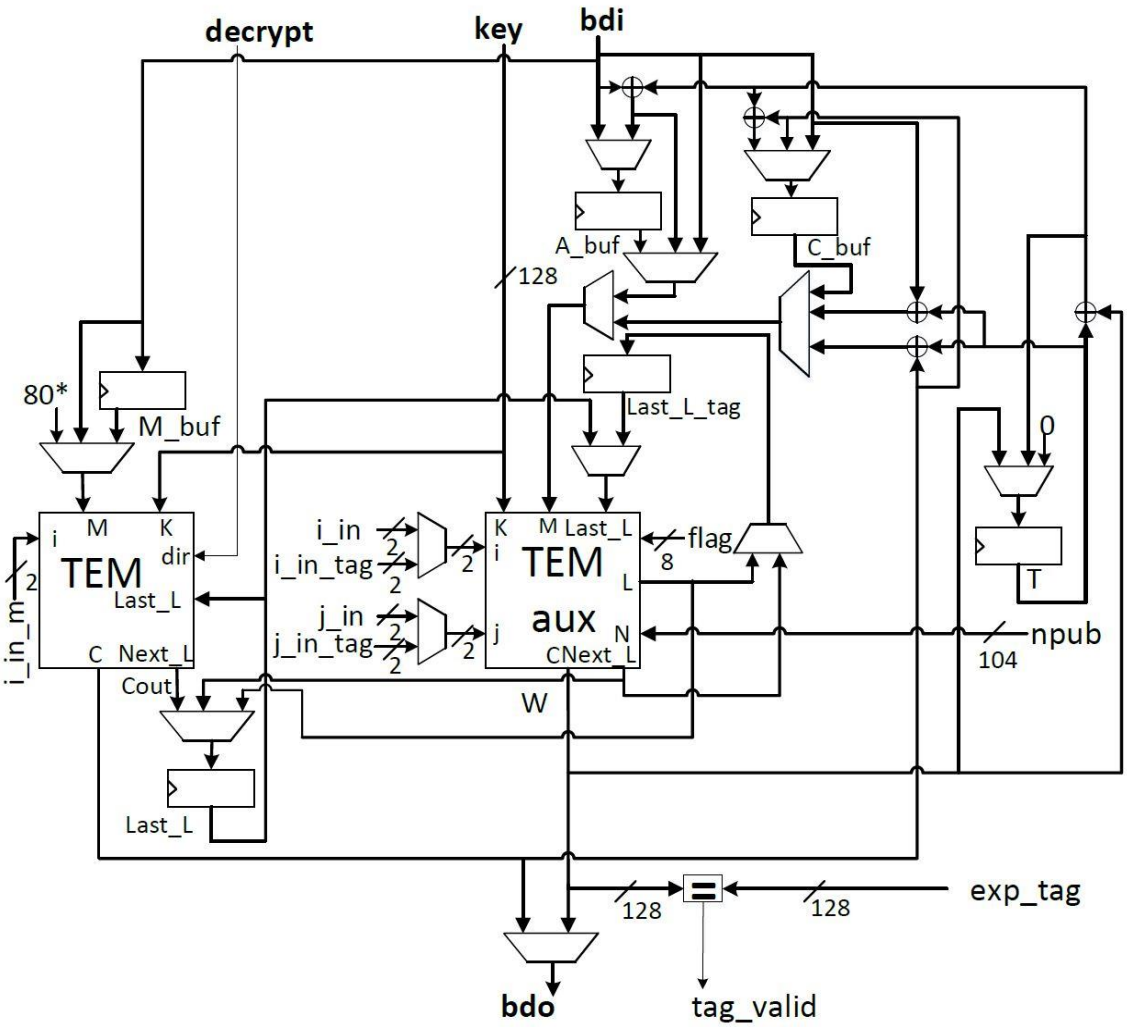

Figure 44: Minalpher Datapath

TEM_AUX and TEM_M blocks are shown clearly in Figure *45* and Figure *46*. Both consists of tweak calculator which takes the values ‘i’, ‘j’ and previous ‘L’ value as an input and process the next L. Minalpher_P block is the permutation block which consists of subnibbles, shufflerows, swap-matrix, mixcolumn and add round constant as shown in Figure *47* and Figure *48*. Each block is processed 17 rounds through the permutation function i.e. each block of data takes 17 clock cycles to give the output.

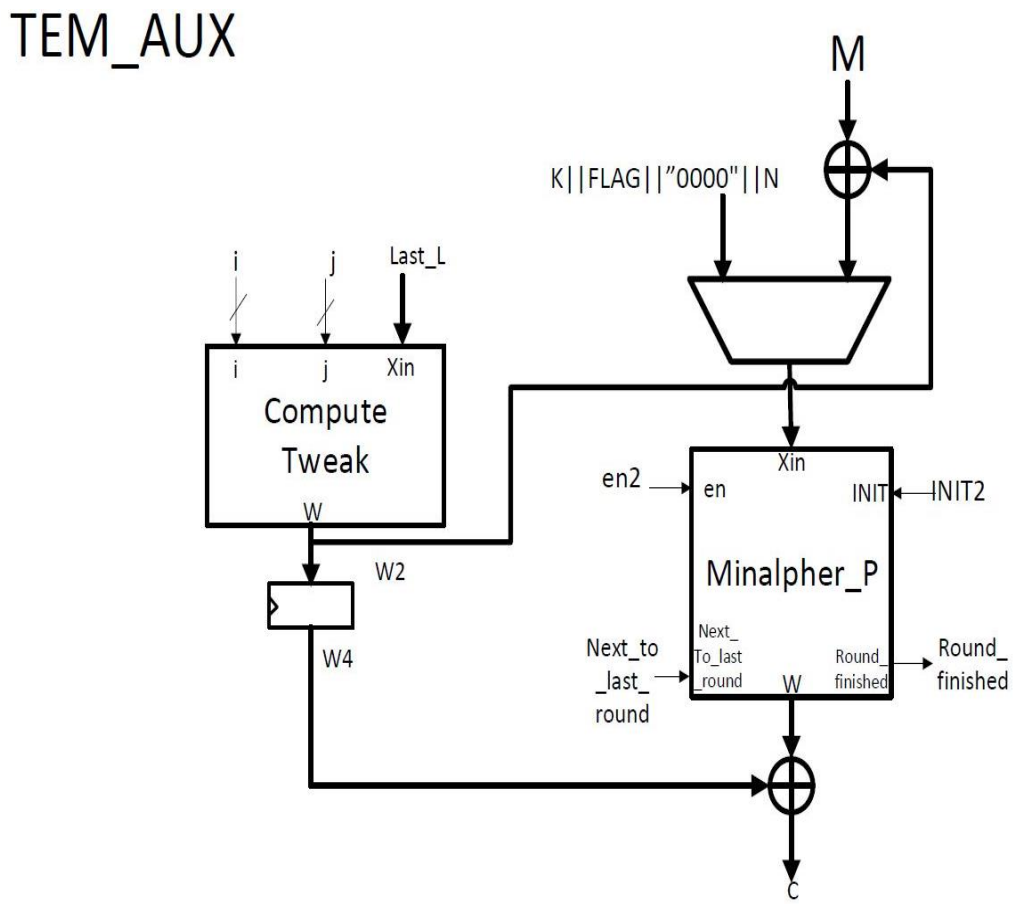

Figure 45: TEM_AUX

TEM

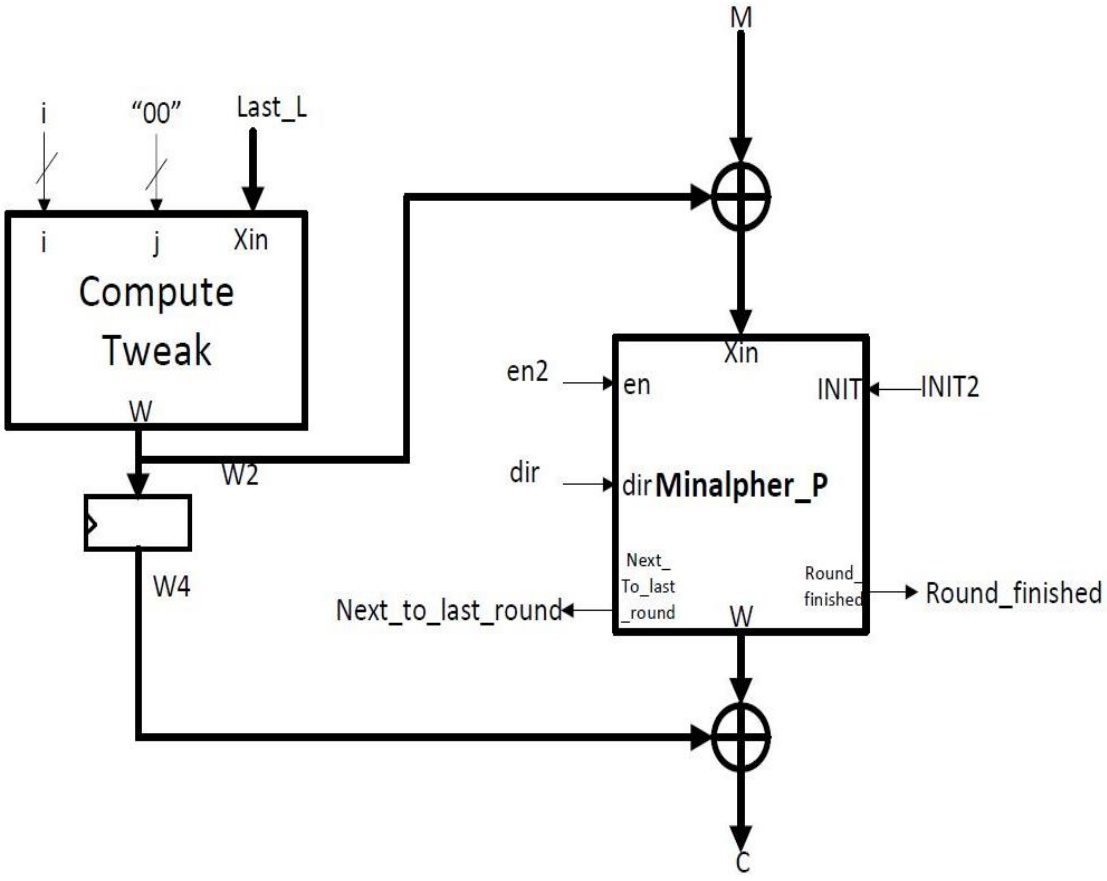

Figure 46: TEM

MINALPHER_P

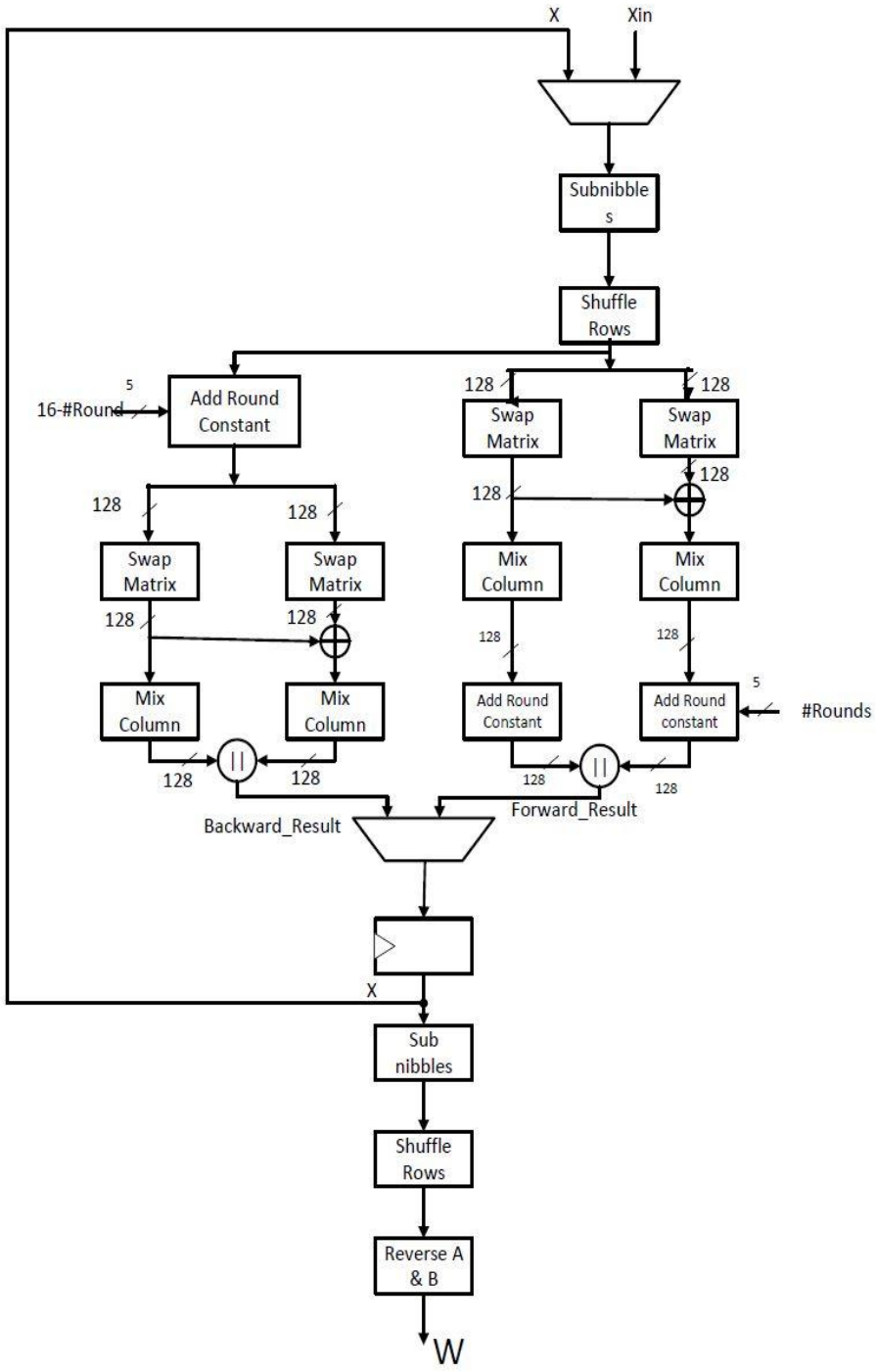

Figure 47: Minalpher_P

MINALPHER_P_FWD

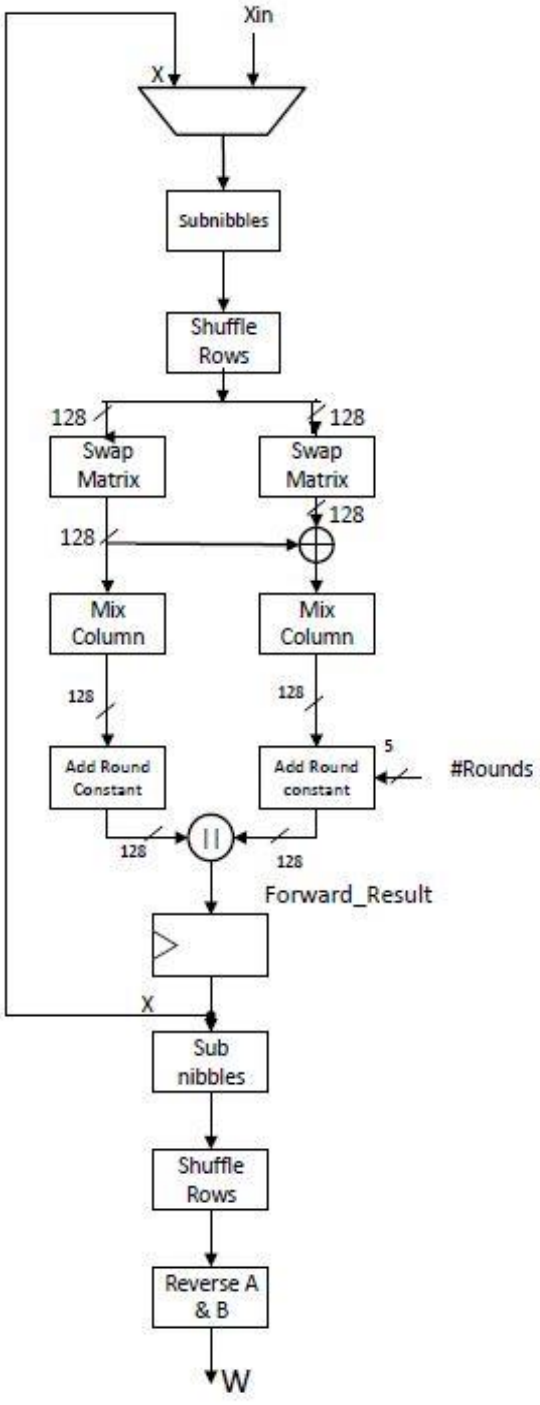

Figure 48: Minalpher_P forward

6.4.2. Controller Design:

Upon the reset the controller enters the initialization state and if key need to be updated the new key is loaded and then the associated data initialization is started and then the 'γ_a' value i.e., the subkey is calculated and registered. Then, if the bdi_valid is high the associated data is loaded and processed through the Minalpher_P block. After each value of associated data is processed, the value is XORed with previously processed associated data and stored in the register. After processing all the blocks of associated data, if the bdi_ready and bdi_valid are high, then the plaintext is loaded and XORed with 'Ψ_m' then it processed through Minalpher_P and then the ciphertext is collected. This Ciphertext is again processed through Minalpher_P block and stored in a register 'T'. Similarly after each ciphertext is processed through the Minalpher_P blocks and XORed with previous block and stored in Register 'T'. This value helps in the calculation of Tag. After all the plaintext processing is done then the value of Tag is calculated. The 'T' value is processed through the Minalpher_P block to calculate the Tag.

TEM Controller:

Separate controller was also developed to handle the TEM blocks which starts processing of data on the start signals from main controller the and at the end of processing of each block will provide a signal TEM_finished as shown in the Figure *53*.

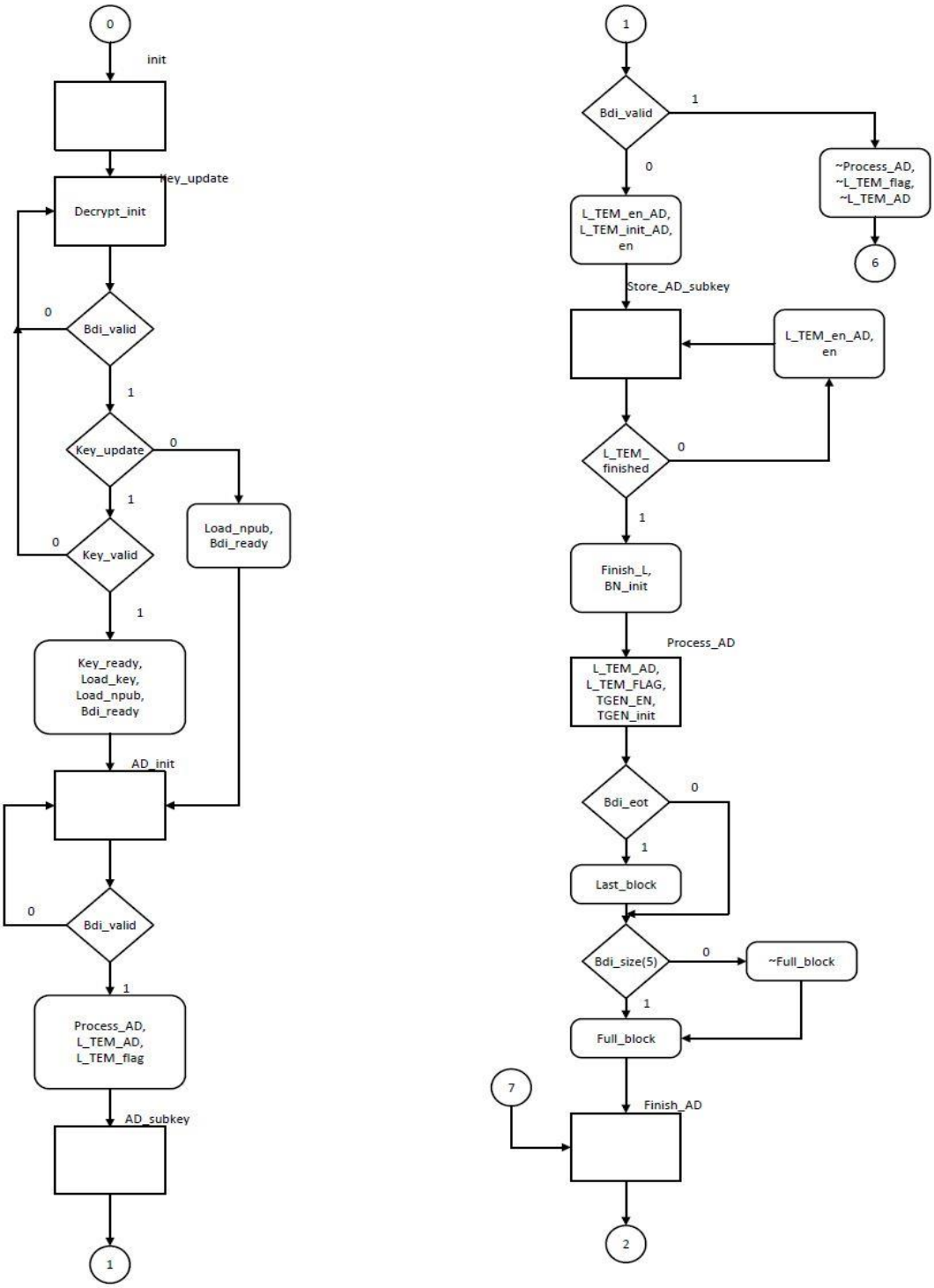

Figure 49: Minalpher Controller (1)

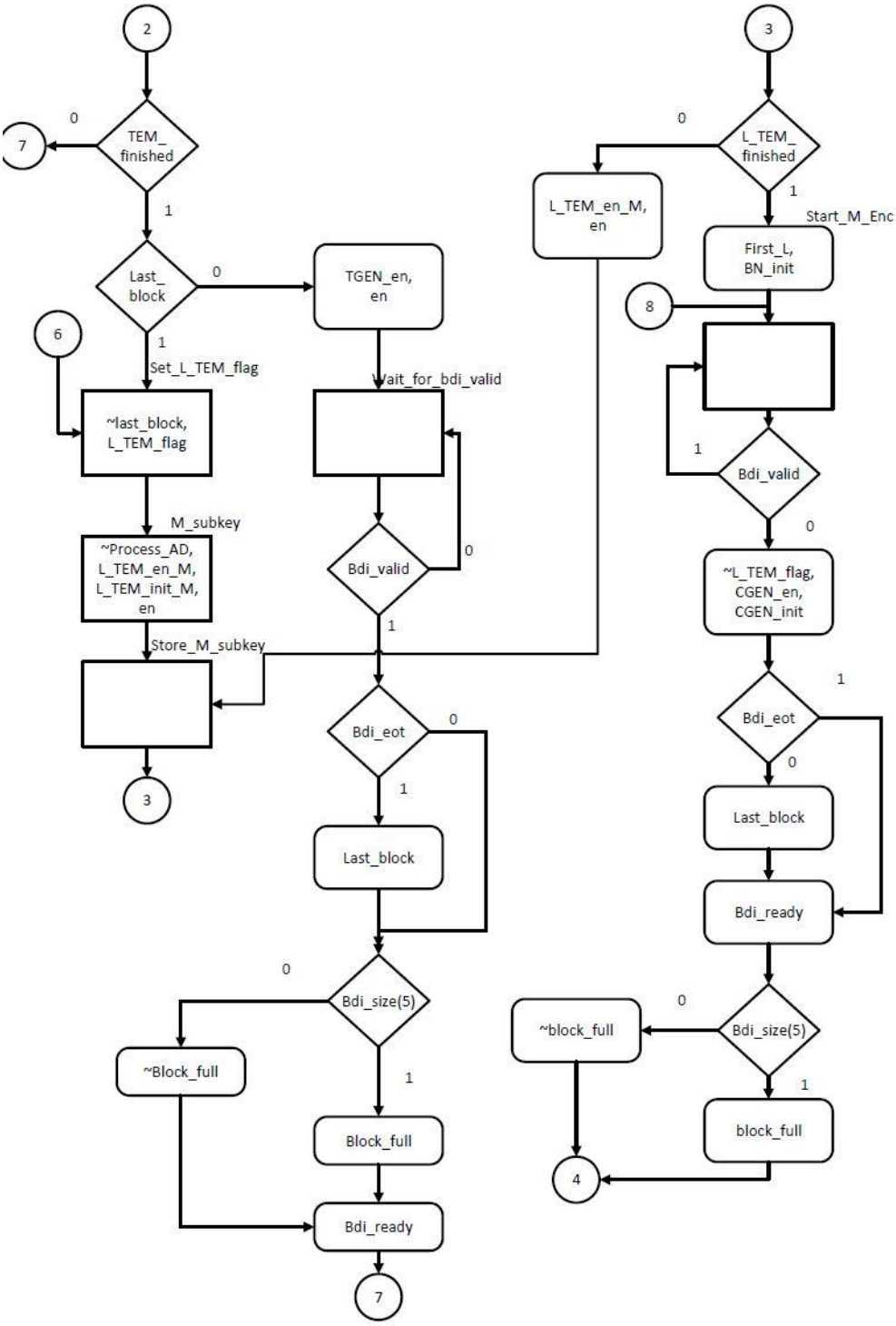

Figure 50: Minalpher Controller (2)

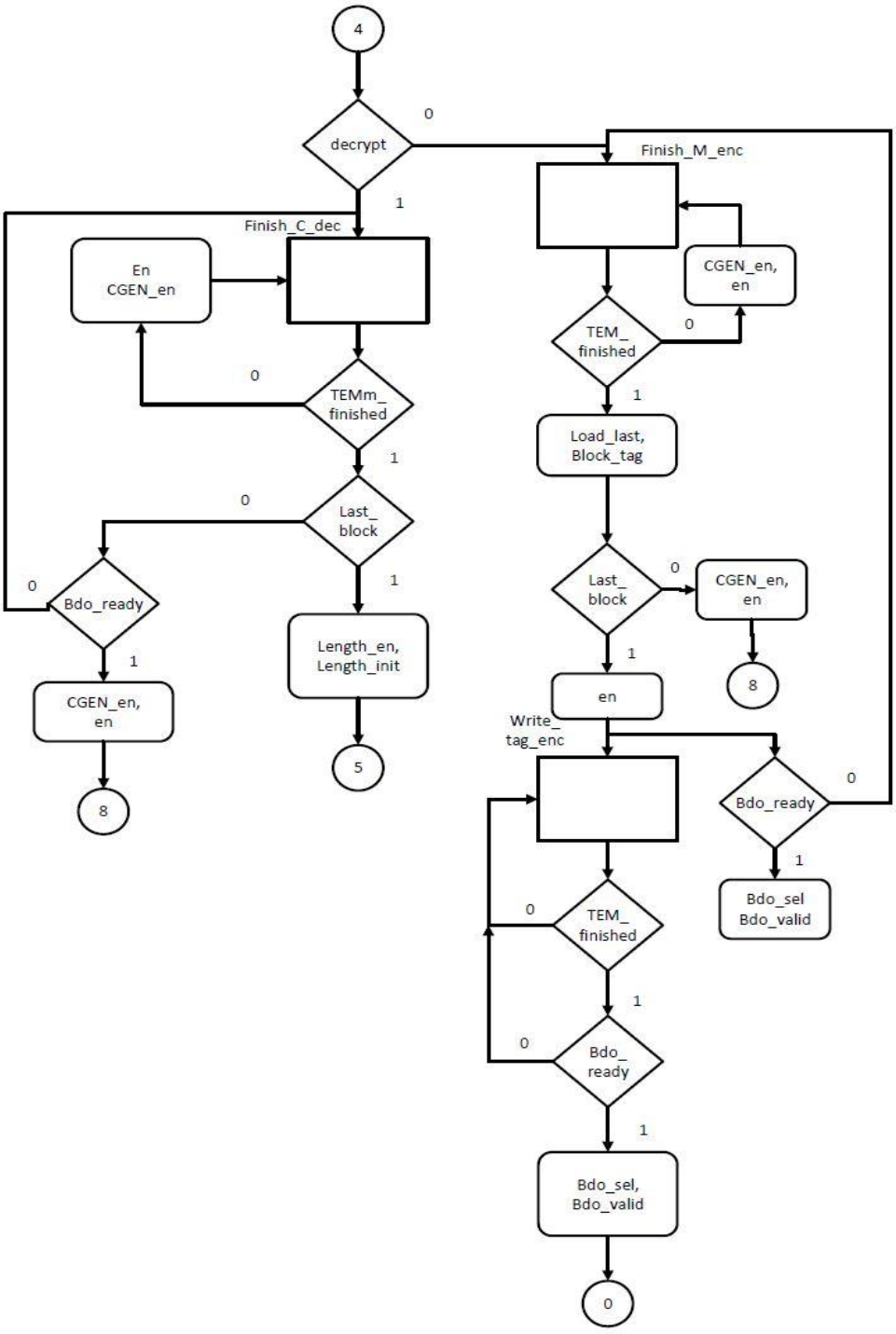

Figure 51: Minalpher Controller (3)

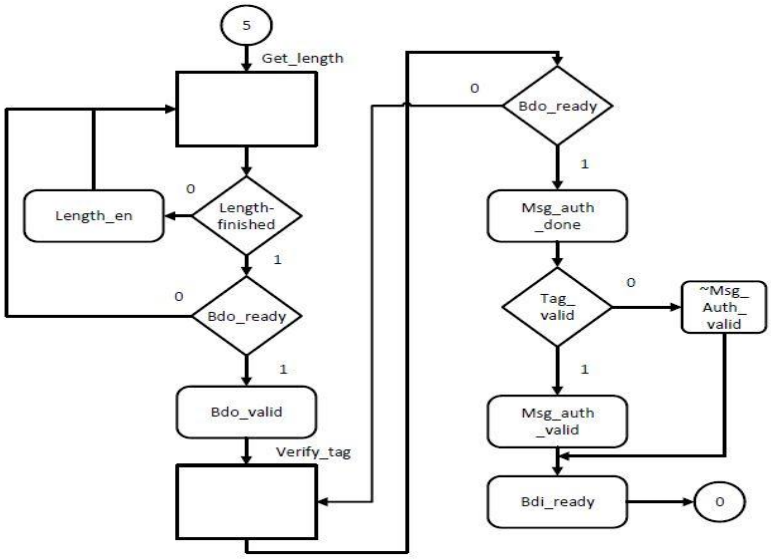

Figure 52: Minalpher Controller (4)

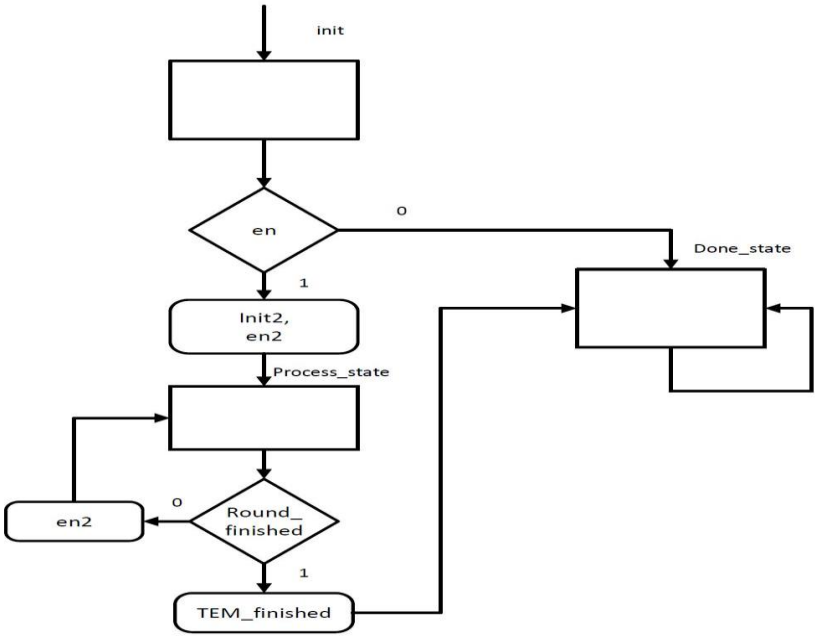

Figure 53: TEM Controller

6.5. Optimized Pipelined Architecture:

In the pipelined architecture, the Datapath and Controller Design from Basic Iterative Architecture were used as a starting point. The optimized pipelined design can process two 256-bit data blocks at a time. The Datapath design is same the Basic Architecture with few modification in bus widths and addition of components. To reduce the critical path, registers were inserted in the Datapath design of Basic Architecture. The critical path of the design lies in the round function of the TEM_M and TEM_AUX as it is the longest path.

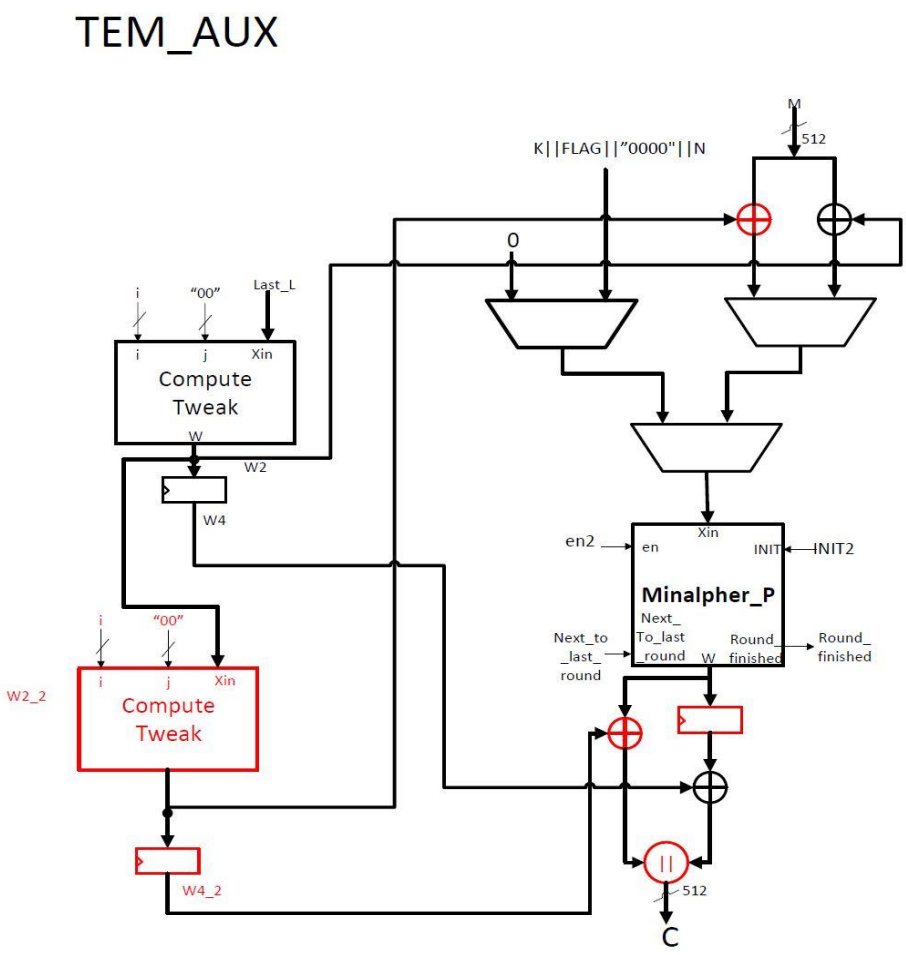

Figure 54: TEM_AUX Pipelined

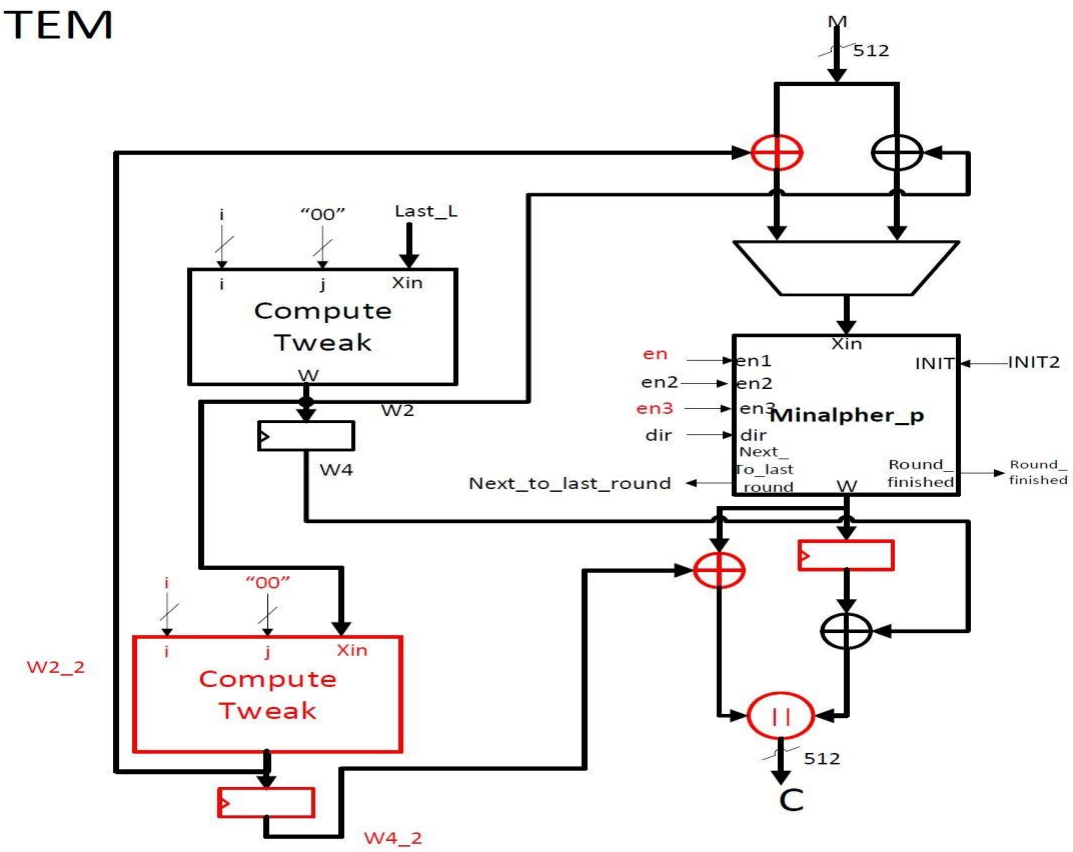

Figure 55: TEM Pipelined

6.5.1. **Register Insertion**:

Two registers were inserted in the Minalpher_P block, one in the encryption path and other in the decryption path as shown in the Figure *56* and one register was inserted in the Minalpher_P_FWD block as shown in Figure *57*. The register insertion has reduced critical path and increased the maximum clock frequency.

MINALPHER_P

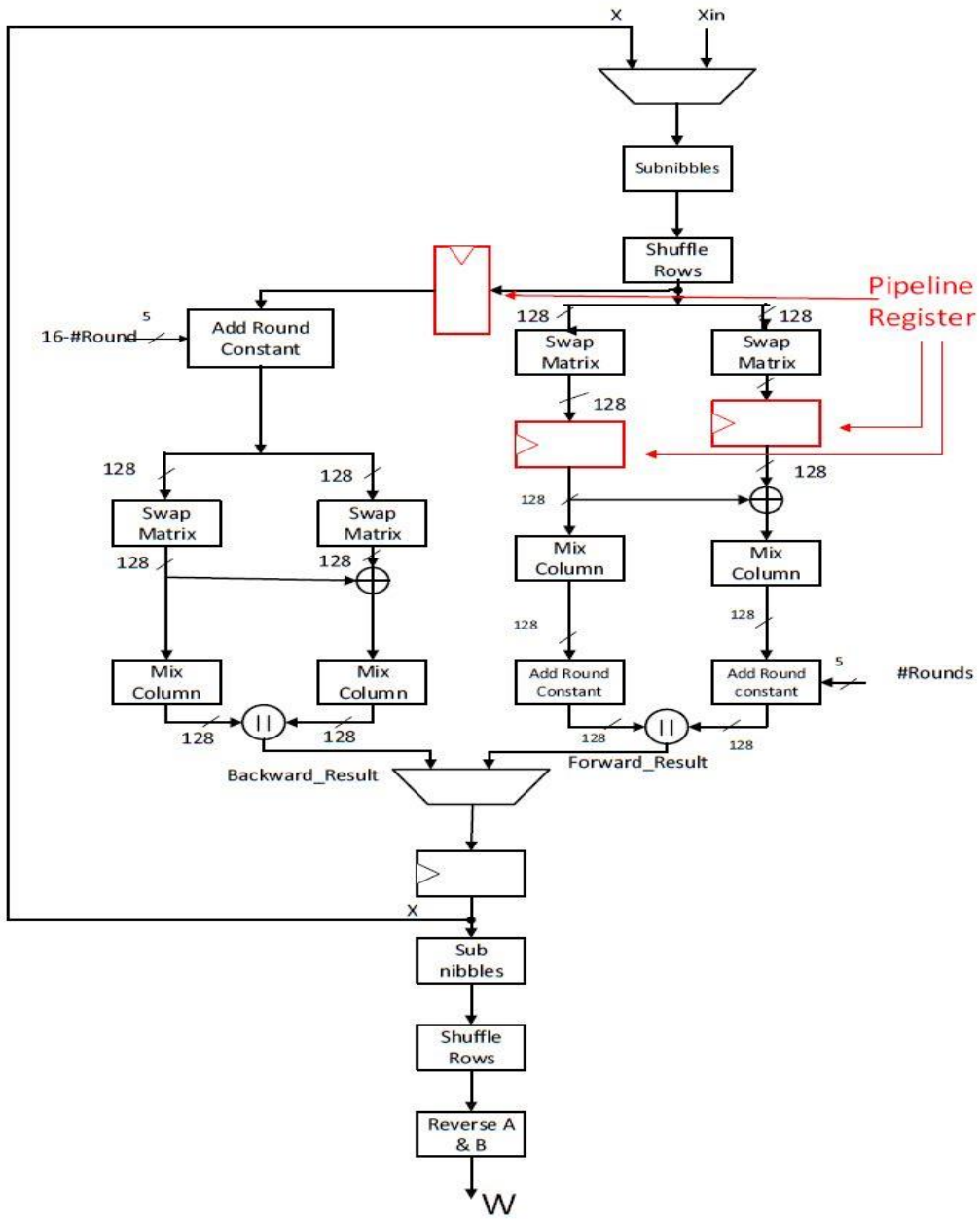

Figure 56: Minalpher_P Pipelined

MINALPHER_P_FWD

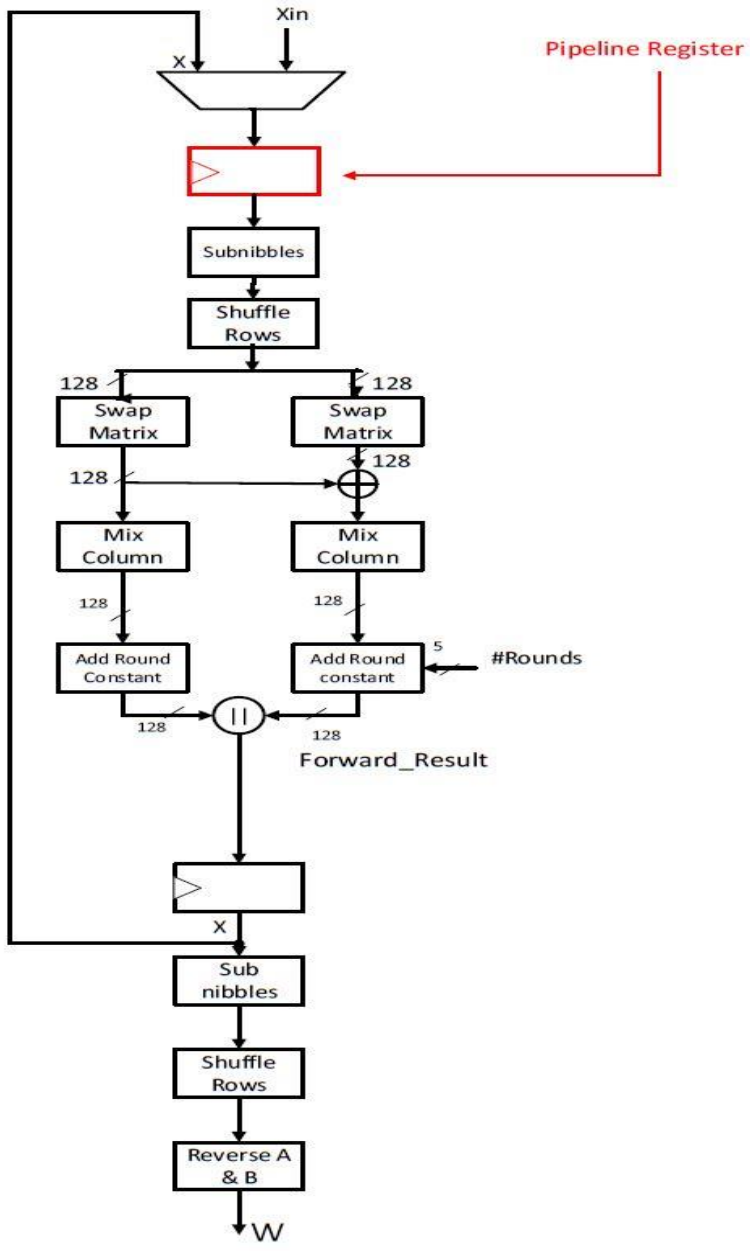

Figure 57: Minalpher_P forward Pipelined

6.5.2. Path Balancing

Path balancing was achieved by changing the widths of the buses to 256 as shown in Figure *54* and Figure *55*. The Tweak generator in TEM_AUX and TEM_M was duplicated for the calculation of the tweak for the second block of data. The output from the first tweak generator was given to the second tweak generator to generate the second tweak. An additional register was placed at the output of the Minalpher_P to hold the first block of data till the last round of second block of data was processed as shown in Figure *54* and Figure *55*.

6.5.3. Controller Modifications

There were no modifications in the top level controller. As the control was totally dependent of TEM_finished signal. Changes were made in the lower level TEM_M and TEM_AUX controllers. Four additional signals en1, en3, en4 and msb and two additional states were added to handle the processing of the second block of data as shown in the Figure *58*. One state for initialize the second block and second state to control the round for two blocks of data.

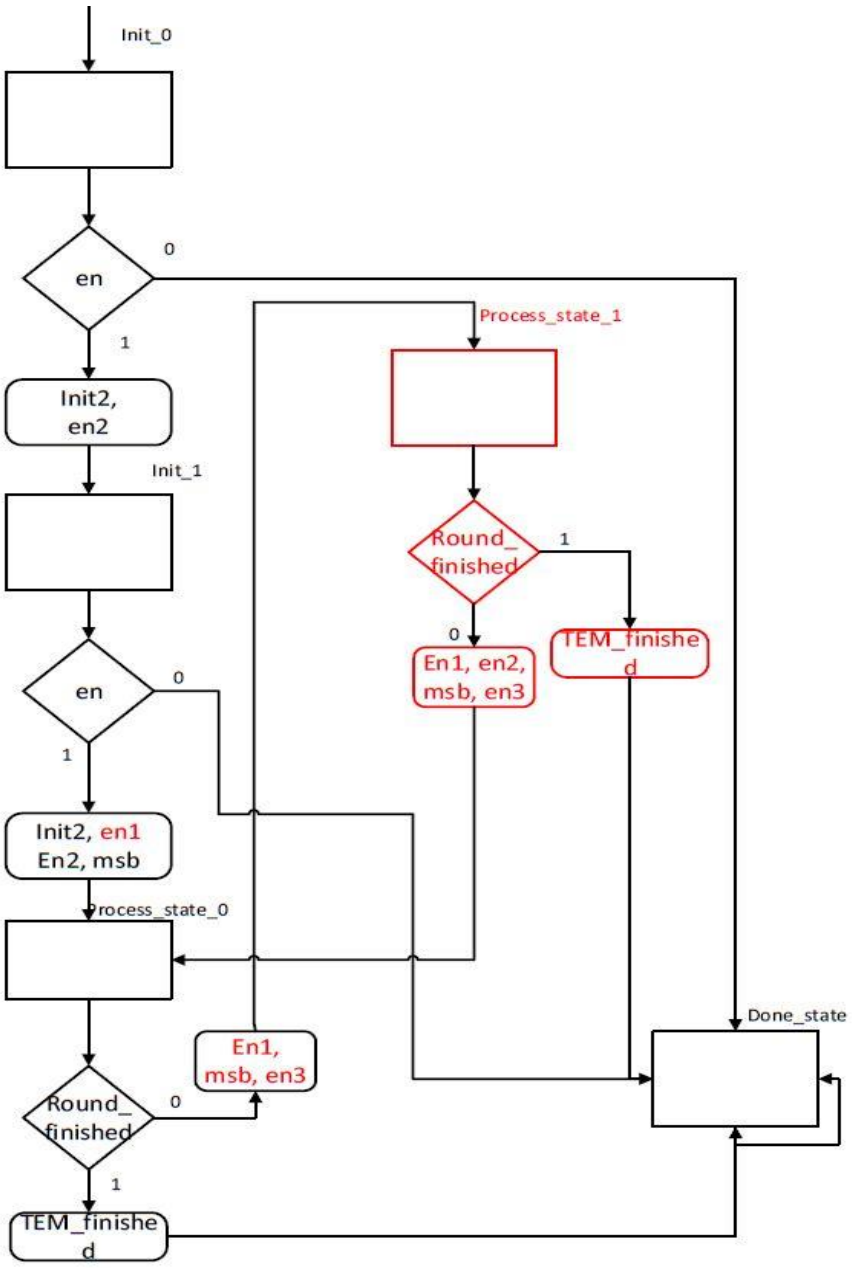

Figure 58: TEM Controller Pipelined

7. OCB

7.1. Introduction and Major features

OCB [8] was submitted and designed by Ted Krovetz and Phillip Rogaway in CAESAR competition. OCB was designed to have following features fast, provably secure, parallel, timing-attack resistant, online and Static AD. OCB is an AEAD scheme that depends on a block cipher. The block cipher must have 128 bit block size. In this design we made a choice of AES 128 as a block cipher. OCB uses nonce 'N' encryption and decryption which should be distinct for each encryption operation. After encrypting plaintext 'P' in the presence of associated data 'A' the cipher outputs the ciphertext 'C' of same length as 'P' and additionally authentication tag.

7.2. Recommended Parameters:

Recommended parameter set given in the specification.

- key length: 16 byte (128 bits) and tag length: 16 byte (128 bits).

7.3. Encryption and Decryption

As mentioned earlier the encryption and decryption in OCB are dependent on the blockcipher.

Encryption: OCB encrypt function accepts key, nonce, associated data and plaintext as input. Initialization is done by calculation of key dependent variable i.e. L, double L. After the initialization the plaintext is divided in to sequence of 128 bit blocks. Next step is to calculate nonce dependent variables and per-encryption variables i.e bottom, Ktop, stretch, offset_0, checksum_0. Then the full block encryption takes place following which partial block is padded and encrypted. After all the plaintext blocks are encrypted then the tag calculation takes place. Finally ciphertext and tag are assembled together and sent as the output.

Decryption: OCB Decrypt function works similar to OCB Encrypt with few modifications. The blockcipher in OCB decrypt works in decrypt mode. OCB decrypt function accepts key, nonce, associated data, ciphertext and tag. Initialization is done by calculation of key dependent variable i.e. L, double L. After the initialization the ciphertext is divided in to sequence of 128 bit blocks. Next step is to calculate nonce dependent variables and per-encryption variables i.e bottom, Ktop, stretch, offset_0, checksum_0. Then the full block decryption takes place following which partial block is padded and decrypted checksum is also calculated in parallel. After all the ciphertext blocks are decrypted then the tag calculation takes place. The calculated tag and received tag are compared and validated. If the tag is valid then the plaintext blocks are assembled and sent to the output else the invalid flag is set high.

7.4. Basic High-Speed Architecture

7.4.1. Datapath Design:

The Datapath design of OCB consists of AES 128 core as shown in the Figure *59* which is used as a block cipher. The 128 bit key is loaded in four clock cycles i.e. 32 bits in each clock cycle. After which the L value is calculated by encrypting block of zeros using the key. Then, the key dependent variable L and double_L are calculated and are stored in a RAM. After which the associated data is processed using the AES_EncDec. The processed associated data is XORed with the previously processed associated and stored as Sum. This sum is useful later during the tag calculation. Before processing the message the nonce dependent variables called bottom, ktop, and stretch are calculated. Then, the plaintext processing is started by loading 128 bits of plaintext for processing each time. After processing all the blocks of data, the tag value is calculated by XORing checksum_m(XOR of all input values), offset_m and L value and then processing it through the AES_EncDec. If we are running cipher in the decryption mode then the tag verification is done after this step.

The AES 128 block that we used in this design consists of a mixed round which helps in calculation of both round as well as inverse round and the datapath is as shown in Figure *60*.

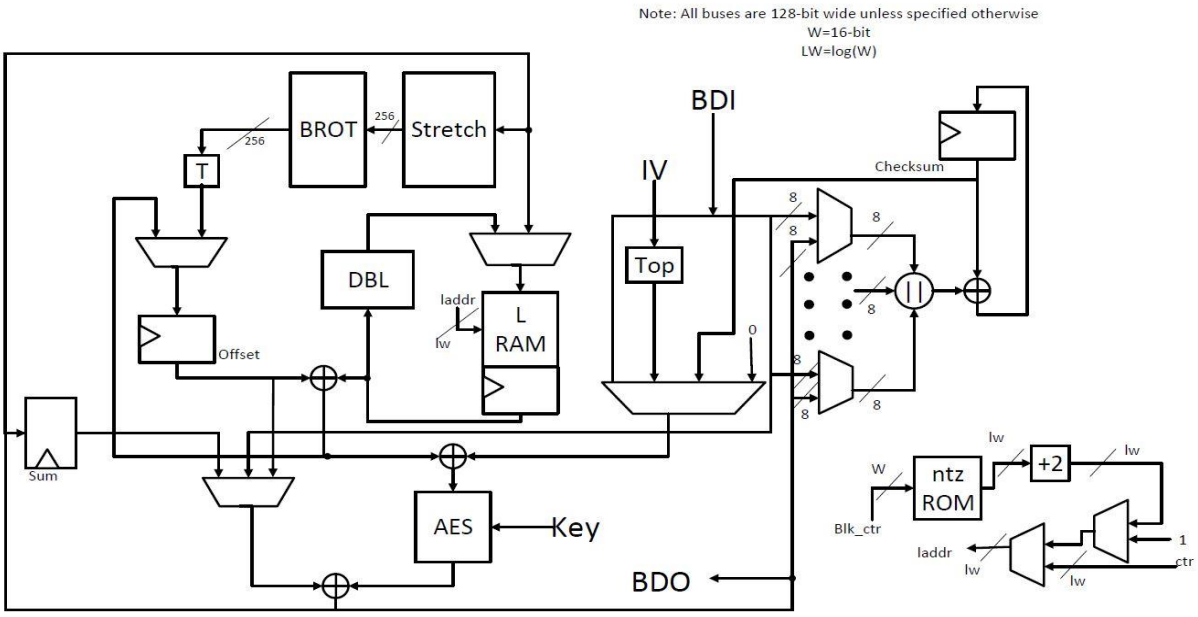

Figure 59: OCB Datapath

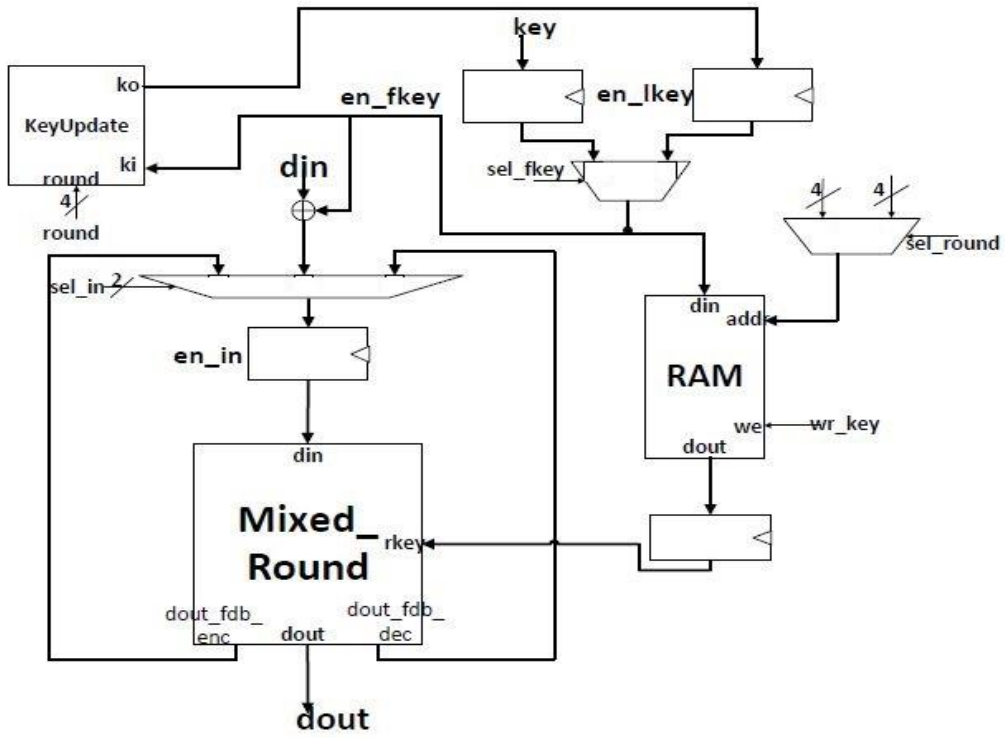

Figure 60: AES Datapath

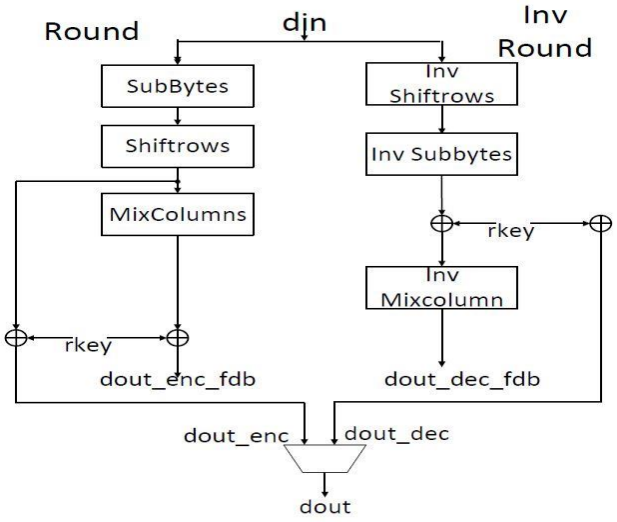

Figure 61: Mixed Round

7.4.2. Controller Design:

Upon the reset the controller enter the initialization state as shown in the Figure *62*. Then, next step is Key check where it checks whether the key update is required or not. If key_update is high then controller moves to the load keys state, it takes 4 clock cycles to load the key completely (i.e. 32 bits in each clock cycle). Then, the next state is key initialization where the value for L (key dependent variable) value is calculated and stored. Then, in the next state the controller waits for the data, when the bdi_type is nonce then Ktop is calculated and if the bdi_type is message/ciphertext then the encryption/decryption starts based on decrypt equals to low/high respectively. Once the data is completely processed (i.e. bdi_eoi is high) then the next state is Tag calculation. After Tag calculation the controller goes back to the initialization state.

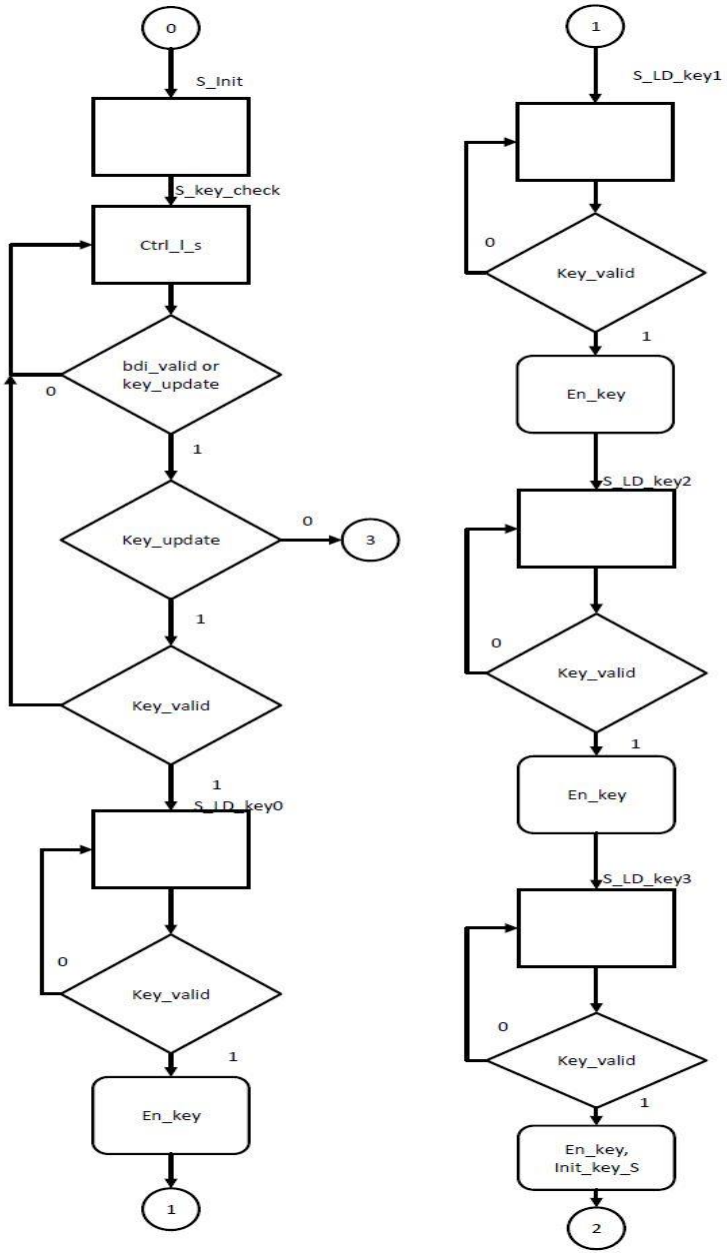

Figure 62: OCB Controller (1)

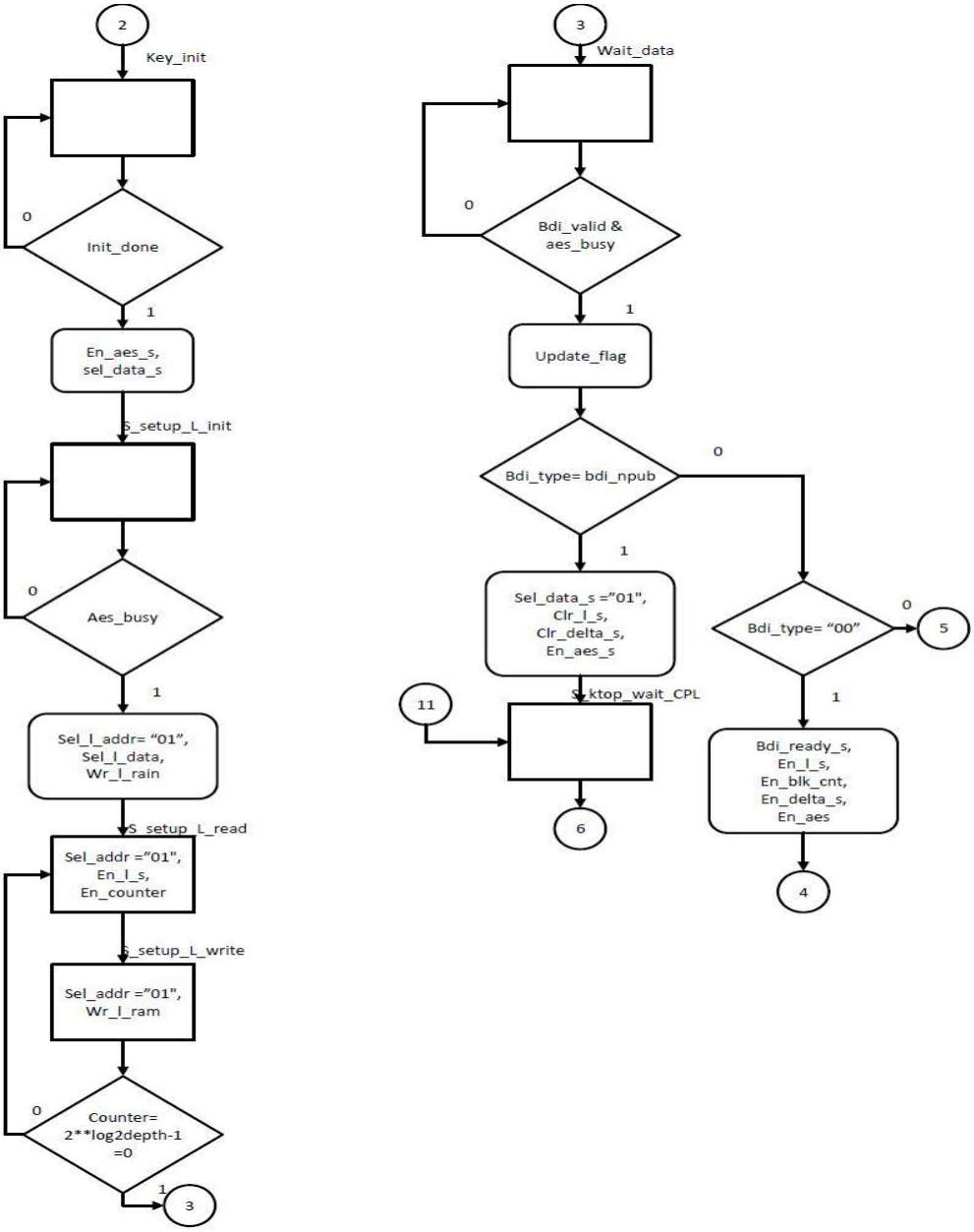

Figure 63: OCB Controller (2).

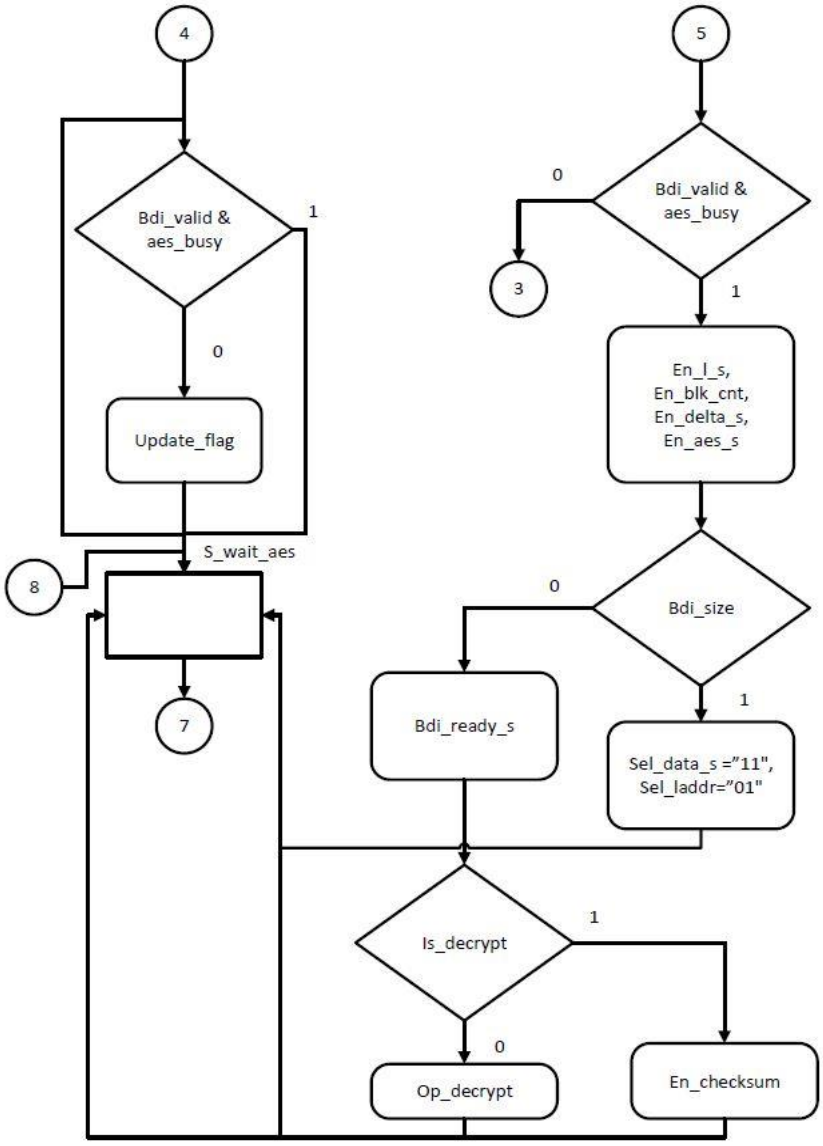

Figure 64: OCB Controller (3)

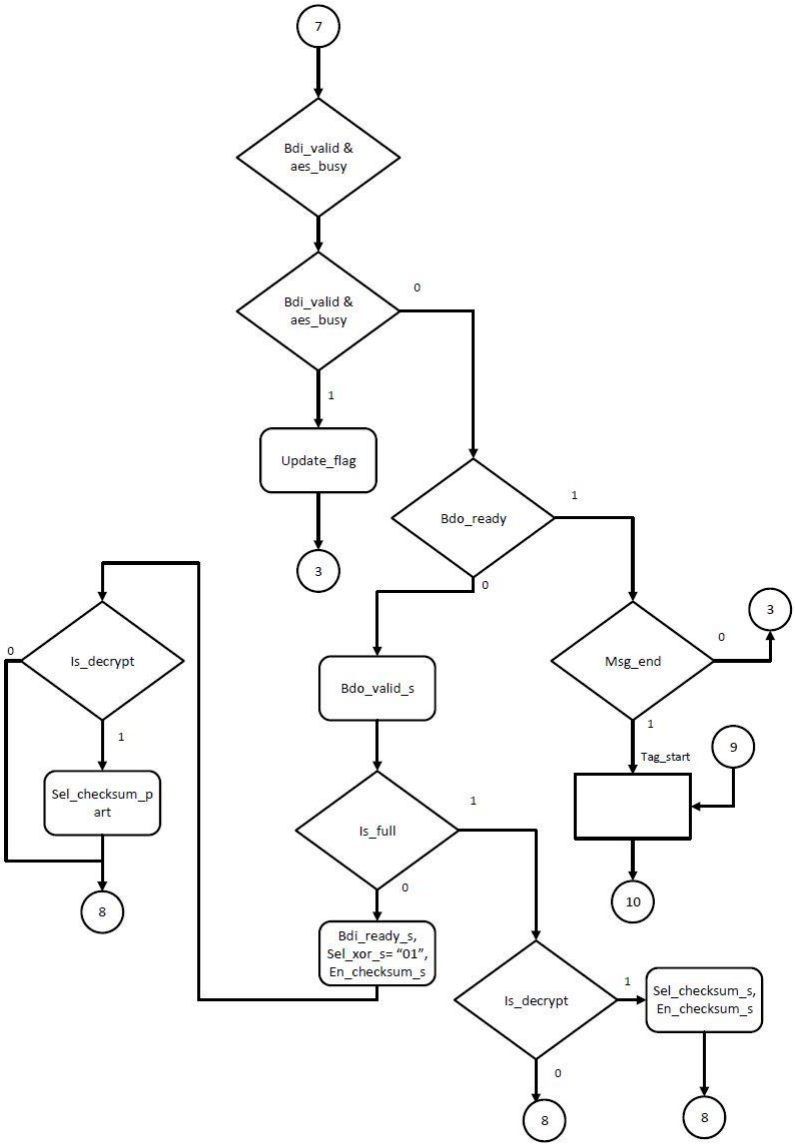

Figure 65: OCB Controller (4)

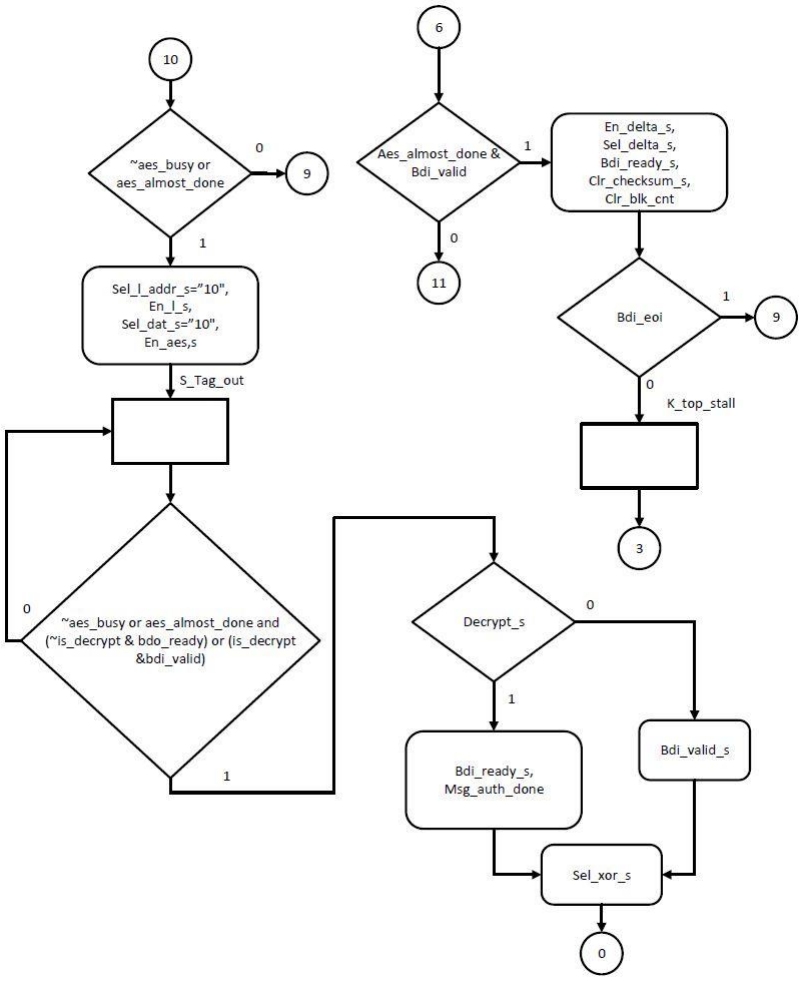

Figure 66: OCB controller (5)

7.5. **Optimized Pipelined Architecture**:

In the pipelined architecture, the Datapath and Controller Design from Basic Iterative Architecture were used as a starting point. The optimized pipelined design can process two 128-bit data blocks at a time. The Datapath design is same the Basic Architecture with few modification in bus widths and addition of components. To reduce the critical

path, registers were inserted in the Datapath design of Basic Architecture. The critical path of the design lies in the round function of the AES as it is the longest path.

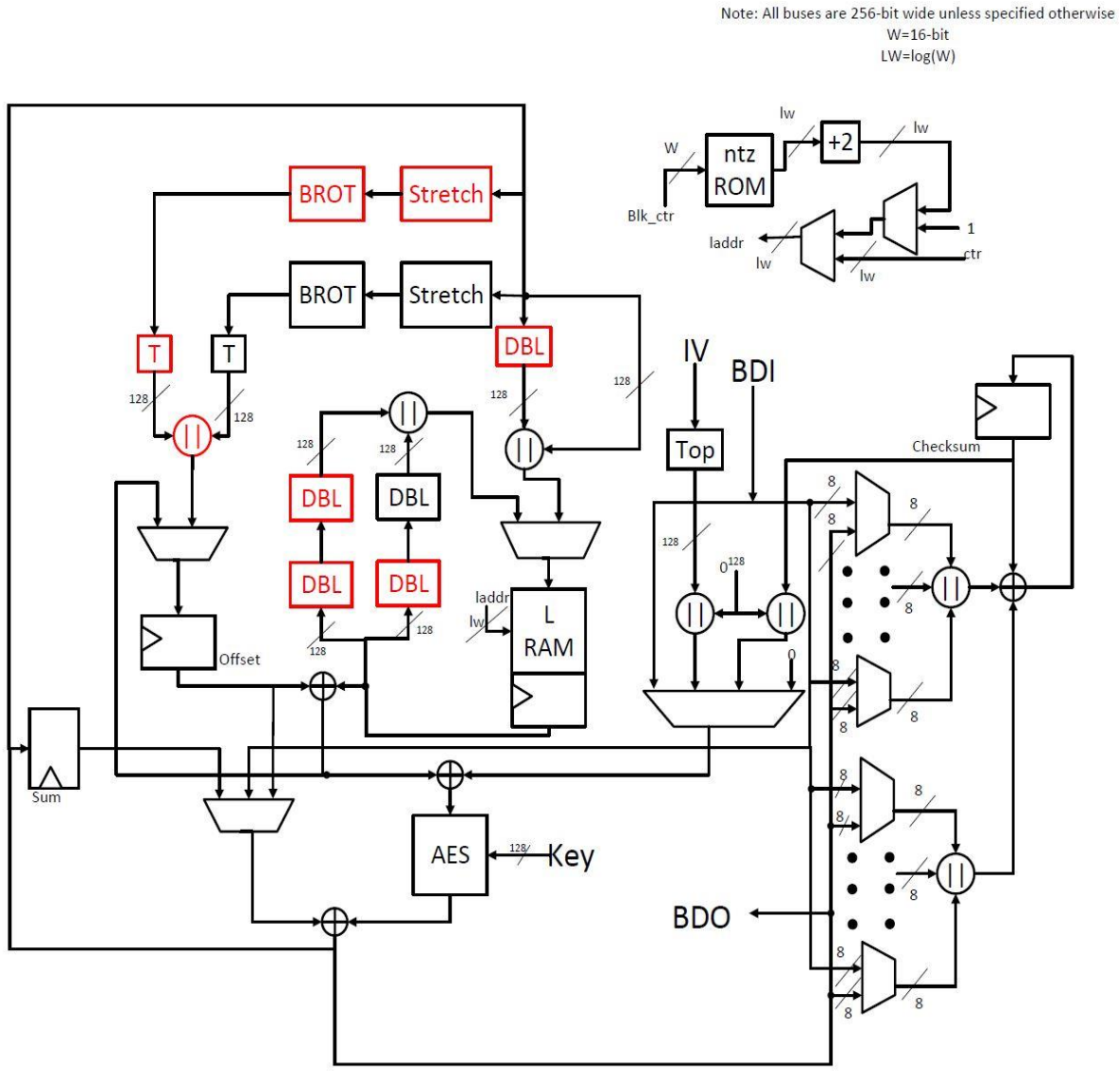

Figure 67: OCB Datapath Pipelined

7.5.1. Register Insertion

Two registers were inserted in the mixed round function of AES as shown in the Figure *68* to reduce the critical path. This register helps in processing two blocks of data in parallel. Due to the insertion of this register the maximum frequency was increased.

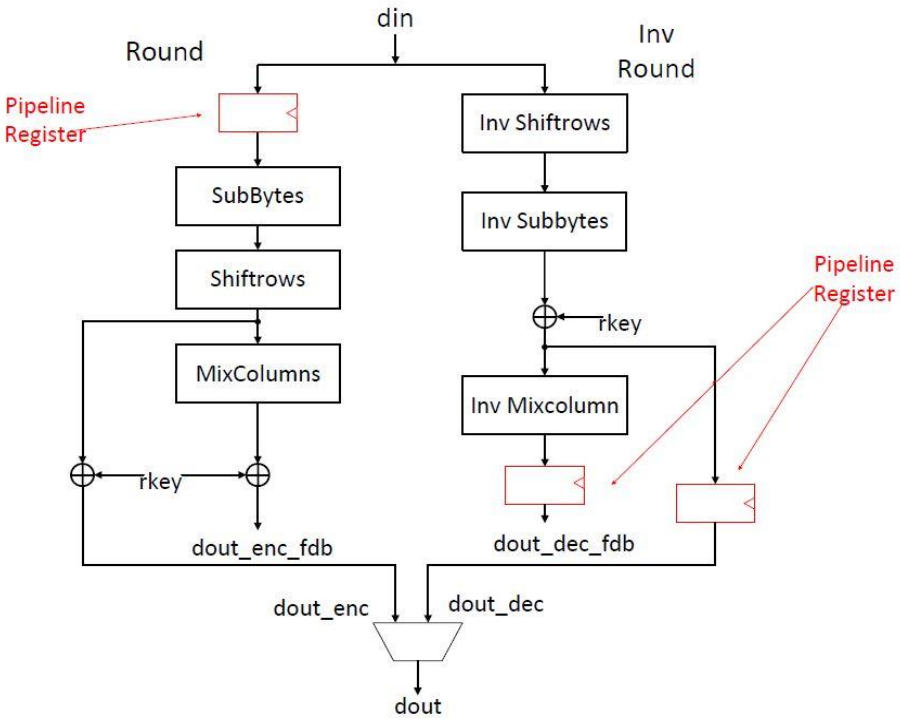

Figure 68: AES Mixed Round Pipelined

7.5.2. Path Balancing

The width of bdi and bdo were changed to 256 bits. In the new design, additional set of key dependent variables i.e. L_2, double_L_2 are pre-calculated and stored in RAM and

nonce dependent variables i.e. bottom_2, ktop_2 and stretch_2 were calculated to help the processing of second block of data as shown in the

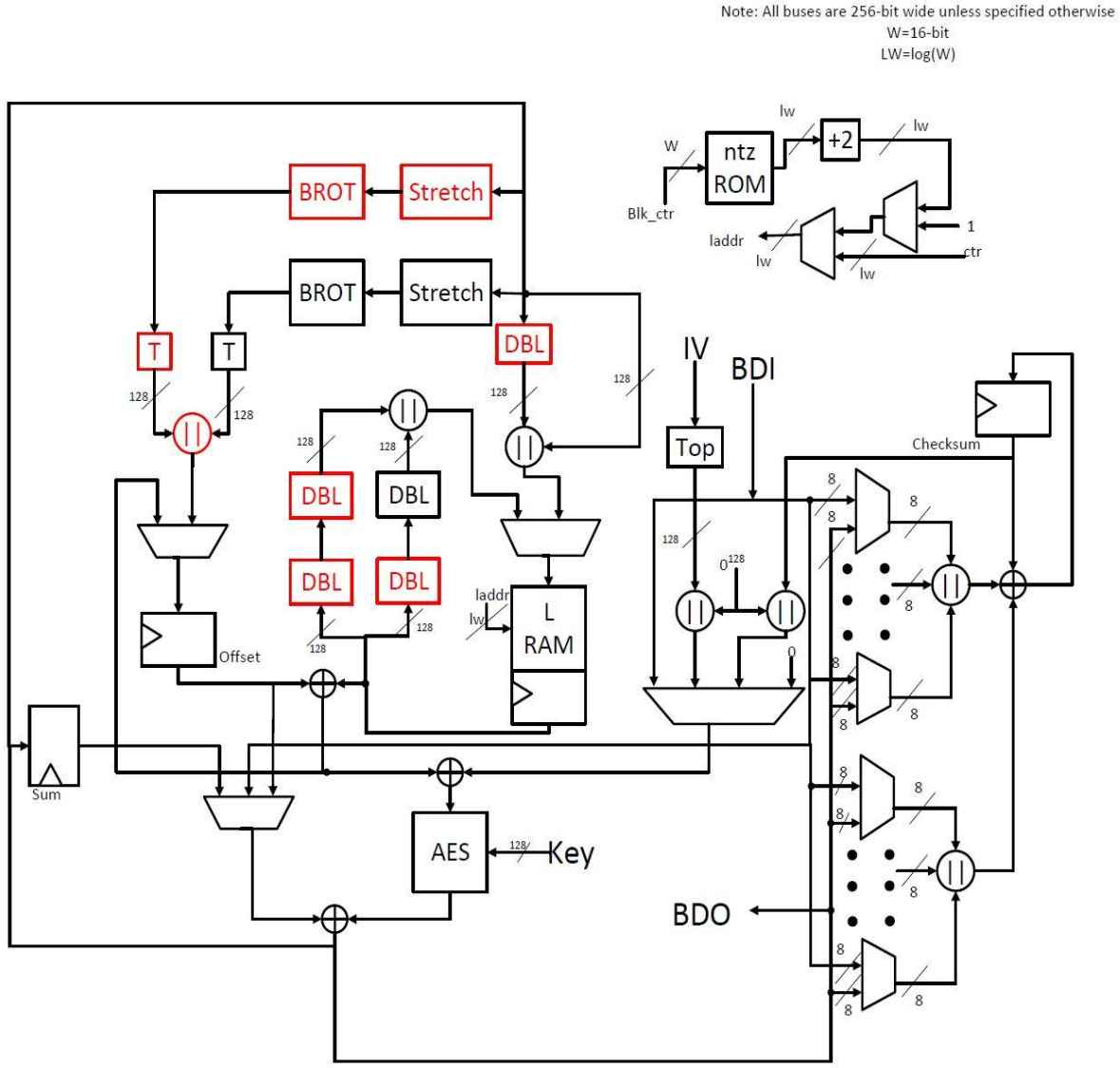

Figure *67*. Changes were made in the AES datapath as shown in the Figure *69*. Additional register was added at the output of the mixed round function to hold the value of first block after processing until the second block was processed.

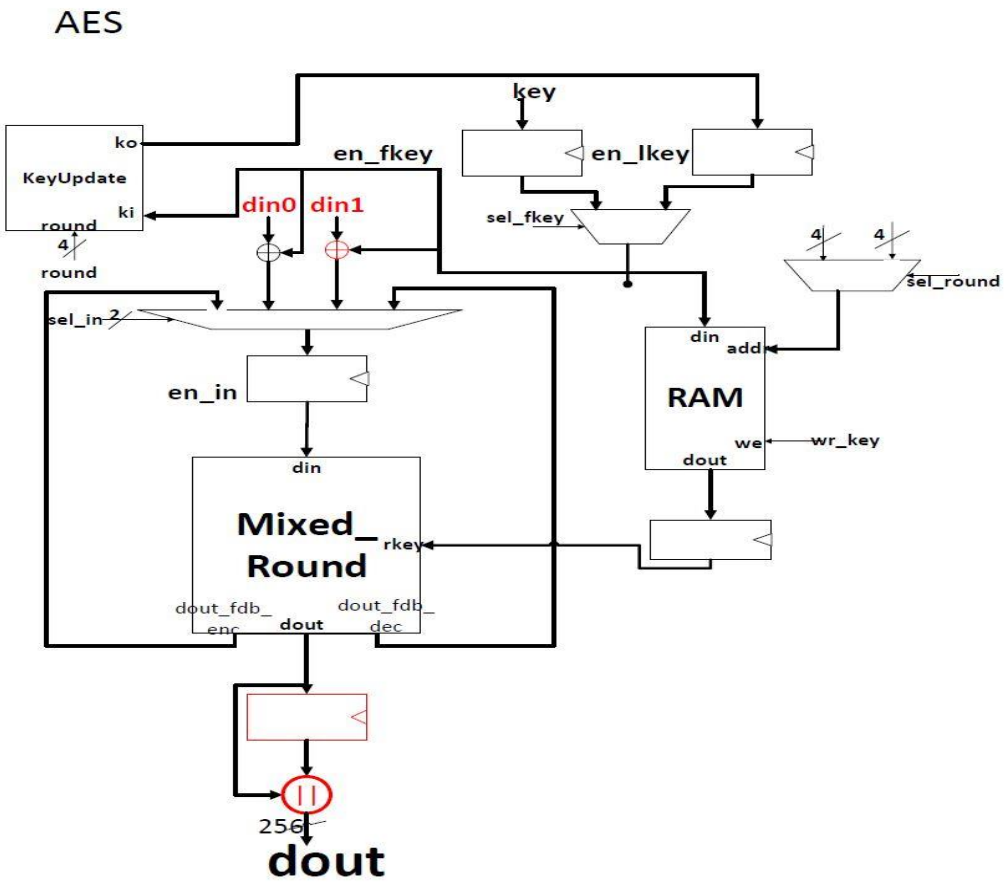

Figure 69: AES Datapath pipelined

7.5.3. Controller Modification

There were no changes in the top level controller. Changes in the AES controller were made.

Changes in the controller of AES:

The Key scheduling in AES_EncDec remained the same but the data processing was changed by adding four additional states. Three states were added to get the data ready entering the round function and an additional state was added to process two

blocks of data at the same time. The round value increases by one per two clock cycles and same round key is available for two clock cycles. The modified controller ASM chart is as shown in the Figure *70*.

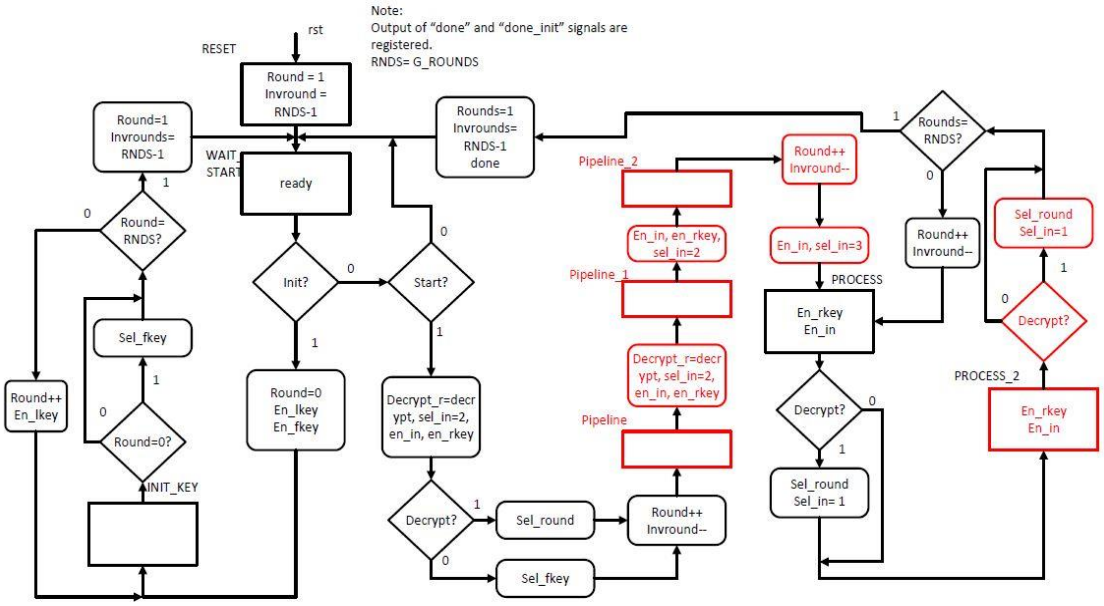

Figure 70: AES Controller Pipelined

8. AES-OTR

8.1. Introduction and Major features

AES-OTR [9] was submitted by NEC Corporation, Japan which was designed by Kazuhiko Minematsu. It is a blockcipher mode of operation to perform an encryption with associated data. OTR stands for Offset Two-Round. The encryption/decryption algorithm uses AES as the block cipher. The encryption function accepts key, nonce, associated data and plaintext as the input and provides ciphertext and tag as the output. The decryption function accepts the key, nonce, associated data and ciphertext as the input and provides plaintext as output when tag matches (|M| = |C|) otherwise rejection symbol is sent as an output. For processing the associated data there are two distinct types of associated data processing (ADP) techniques Serial ADP and Parallel ADP. In this project we used parallel ADP.

8.2. Recommended Parameters:

There are four different parameter sets recommended in specification.

- Primary recommendation: 16-byte (128 bits) key, 12-byte nonce (102 bits), 16-byte tag (128 bits), and parallel ADP.
- Secondary recommendation: 16-byte (128 bits) key, 12-byte (102 bits) nonce, 16-byte (128 bits) tag, and serial ADP.

- Third recommendation: 32-byte (256 bits) key, 12-byte (102 bits) nonce, 16-byte (128 bits) tag, and parallel ADP.
- Fourth recommendation: 32-byte (256 bits) key for AES-256, 12-byte (102 bits) nonce, 16-byte (128 bits) tag, and serial ADP.

We used primary recommendation in this project.

8.3. Encryption and Decryption

First step in encryption process is partition the plaintext in to 128 bit blocks i.e. (M[1],M[2]...,M[m])←M. Let us assume 256 bit blocks of plaintext be to be (M[1], . . . ,M[ℓ])←M. For every i < ℓ the i^{th} chunk M[i] = M[2i − 1],M[2i]) is encrypted with the help of two-round feistel permutation with masks as shown in the following equations

$$C[2i - 1] = EK(2^{i-1}L \oplus M[2i - 1]) \oplus M[2i]$$

$$C[2i] = EK(2^{i-1}3L \oplus C[2i - 1]) \oplus M[2i - 1]$$

L in the above mentioned equation is obtained by encrypting the nonce with tag-length. For the last chunk of plaintext the encryption process is slightly different. If the number of bits in last chunk is greater than 128 bits then a variant of two-round feistel permutation is is applied for encryption of this plaintext and if number of bits in last chunk is less than 128 bits then variant of CTR mode is applied as shown in the Figure *71*.

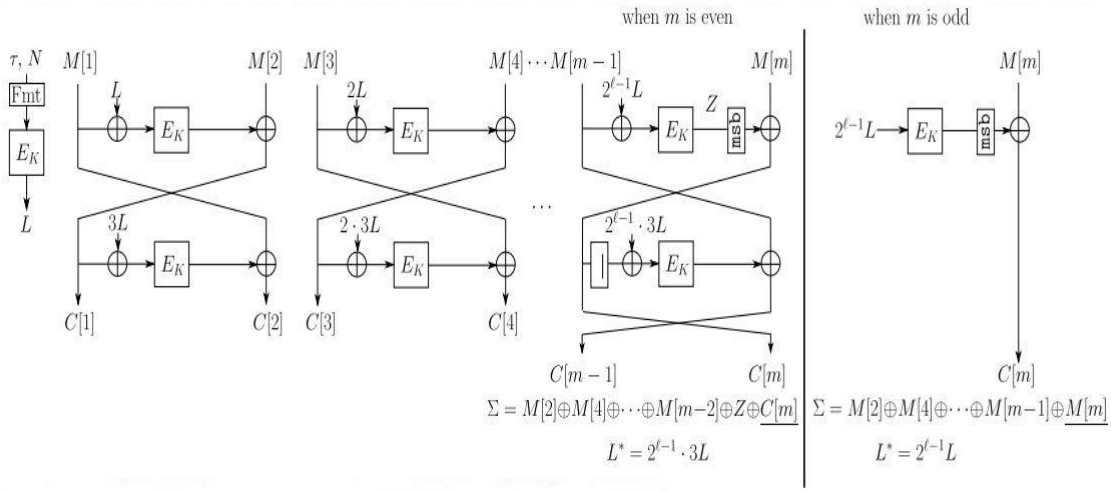

Figure 71: AES-OTR Encryption

As mentioned earlier there are two variants for processing associated data, serial ADP and parallel ADP. As per the primary recommendation we used parallel ADP in this implementation. In Parallel ADP it uses a parallelizable pseudorandom function which is a variant of PMAC. Untrancated data tag TA is calculated by processing associated data as shown in Figure *72*.

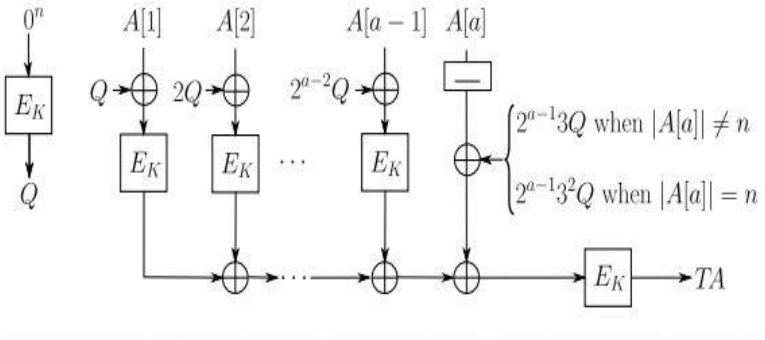

Figure 72: Parallel ADP

For tag calculation the Σ value and L value from Figure *71* and TA value from the Figure *72* are used. The Σ value is XORed with L value and Encrypted then the result is XORed with the TA value to obtain tag as shown in the *Figure 73*.

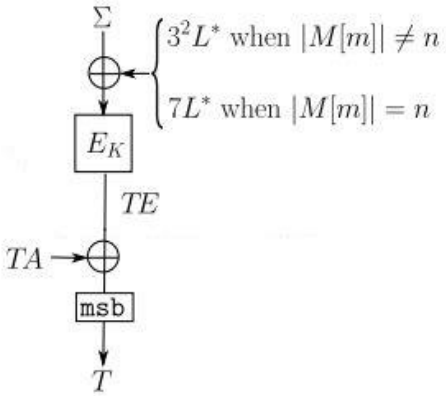

Figure 73: Tag calculation.

8.4. Basic High-Speed Architecture

8.4.1. Datapath Design:

The Datapath design of AES-OTR consists of AES key on the fly function used for the Ek block as shown in the Figure *74*. It consists of galois field multiplication by 4 and multiplication by 2 fields. It also consists of padding block for the incomplete blocks. The value of key, nonce, and public message number are registered. The data flow in the datapath is as follows: Initially the key is loaded and then the 'Q' value and gamma value are calculated from the nonce. After that the associated data processing is started, each processed data block is XORed with next block and then that value is registered. Same procedure is repeated until last block and then the resulting value is again processed through the AES block and then stored as TA shown in Figure *72*. Then, the plaintext processing is started. The encryption/decryption process in AES-OTR is quite different, when compared to the other algorithms. It consists of fiestel structure where the even plaintext/ciphertext block depends on the values odd plaintext/ciphertext as shown in the Figure *71*. After the encryption then next step is tag calculation where the checksum value is encrypted and XORed with the TA value to get the tag value.

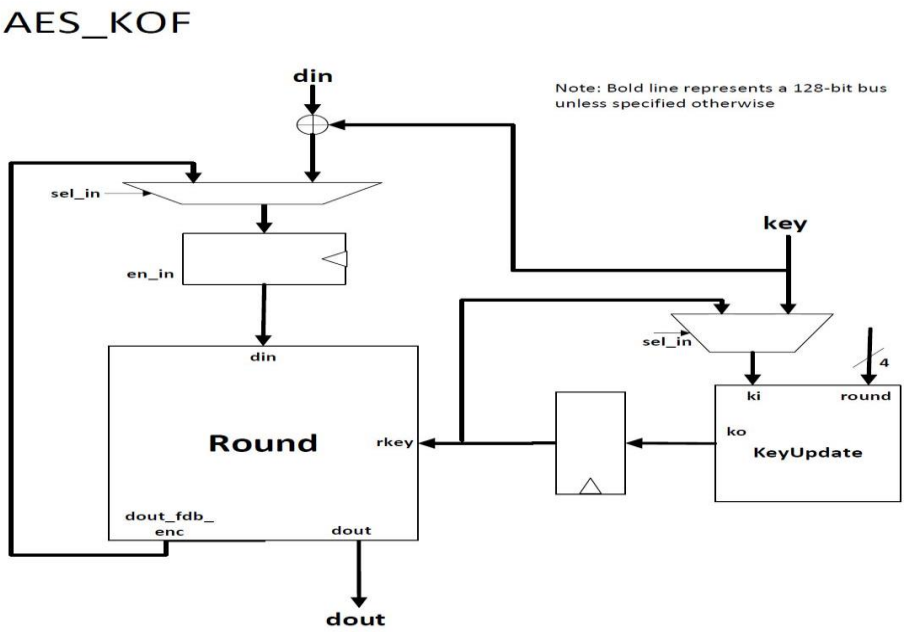

Figure 74: AES KOF Datapath

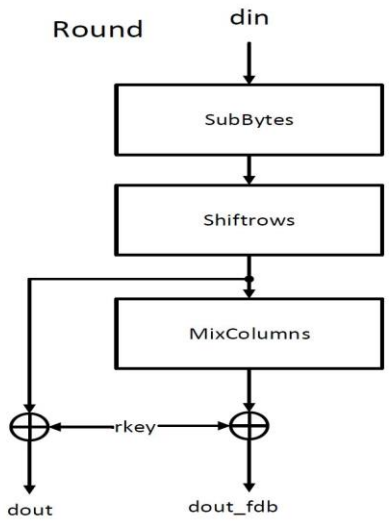

Figure 75: AES KOF Round

8.4.2. Controller Design:

Upon the reset the controller enters the initialization state. Then, the next state is wait state where the controller checks whether a new input is available first then it checks for the key update. If the key update is high next state is wait for key and whenever key is ready, key is initialized. After key update next state is wait for public message number, after loading public message number next state is initialize message where controller waits for message initialization delay then the controller moves to the next state i.e. wait for message, in this state, based on the value of bdi_type controller decides the further processing. If the bdi_type is associated Data then next state will be associated data processing, if the bdi_type is message/ciphertext then next state will be process data state where message/ciphertext will be encrypted/decrypted. Once all the blocks are processed next state is tag generation and if its decryption then next step is tag authentication where the tag is checked with the tag received. After this controller goes back to the initialization state.

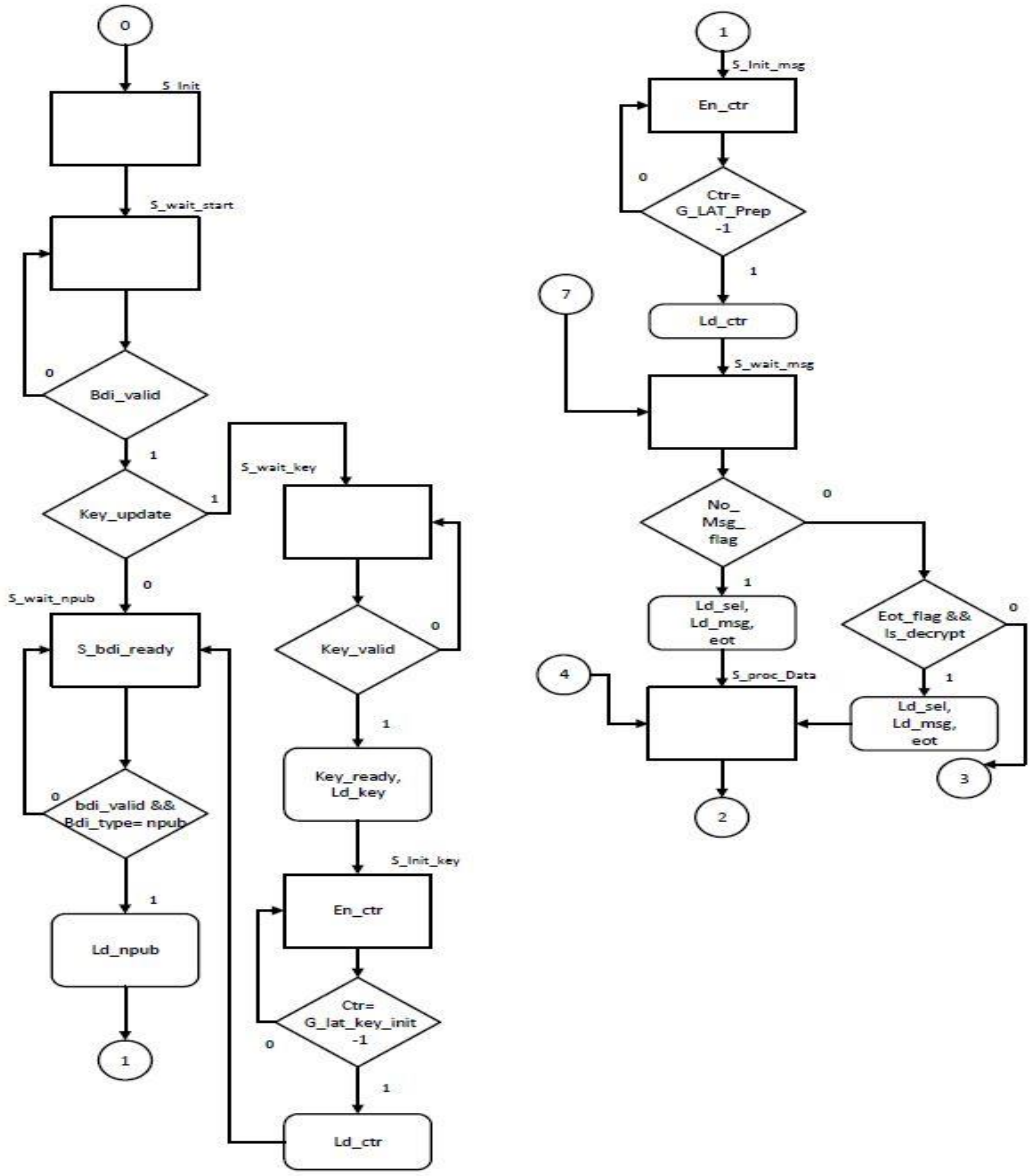

Figure 76: AES OTR Controller (1)

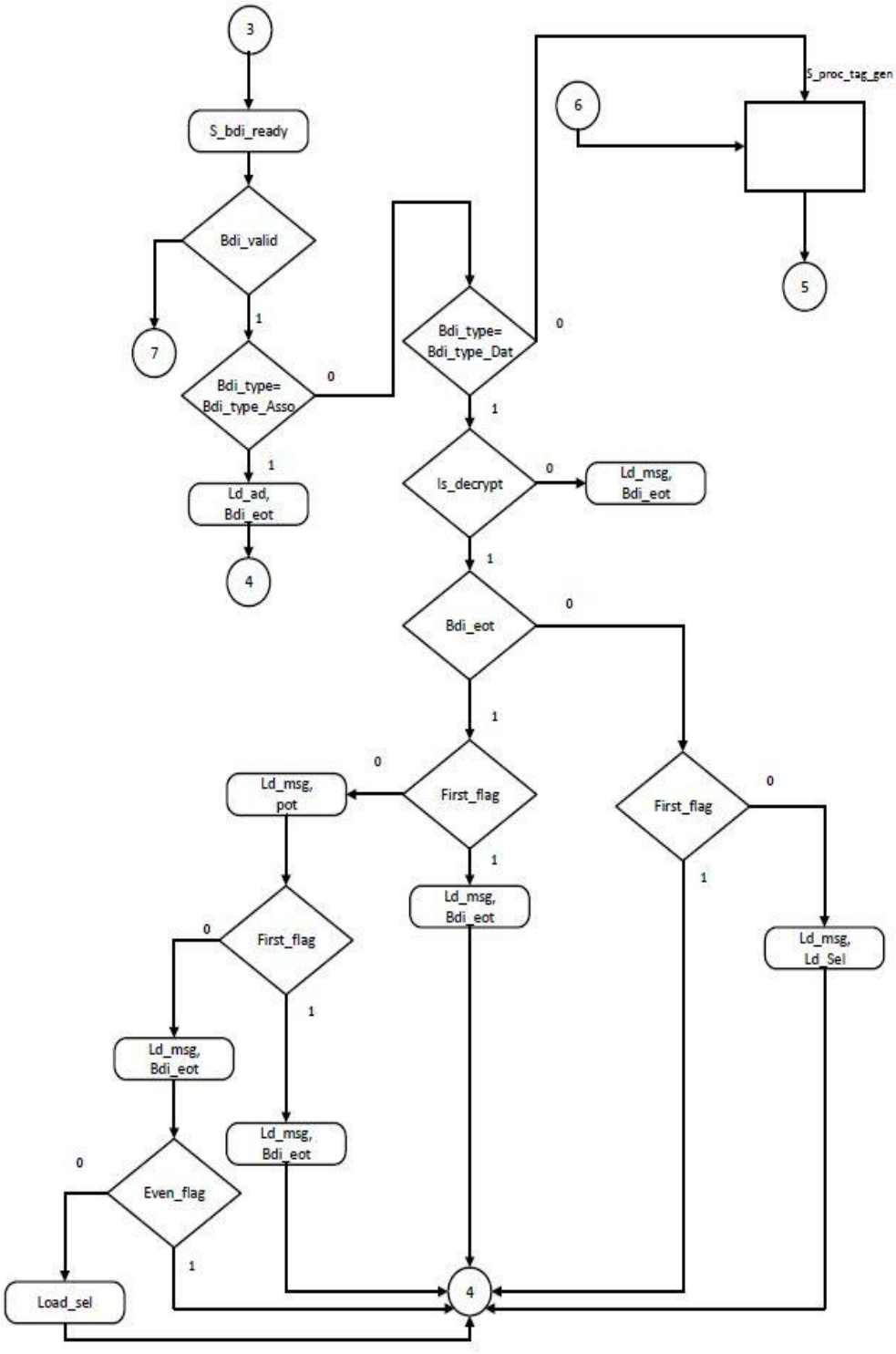

Figure 77: AES OTR Controller (2)

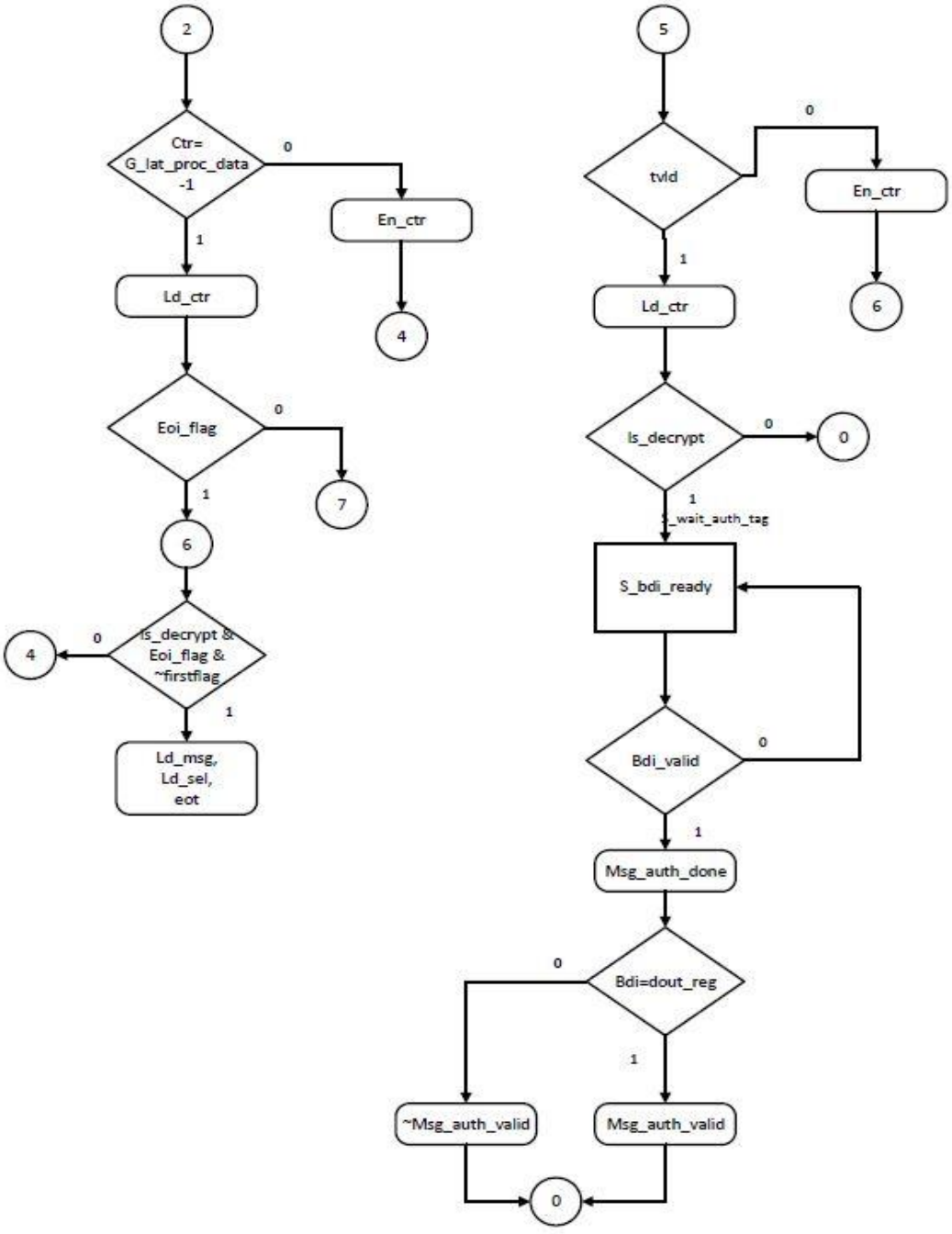

Figure 78: AES OTR Controller (3)

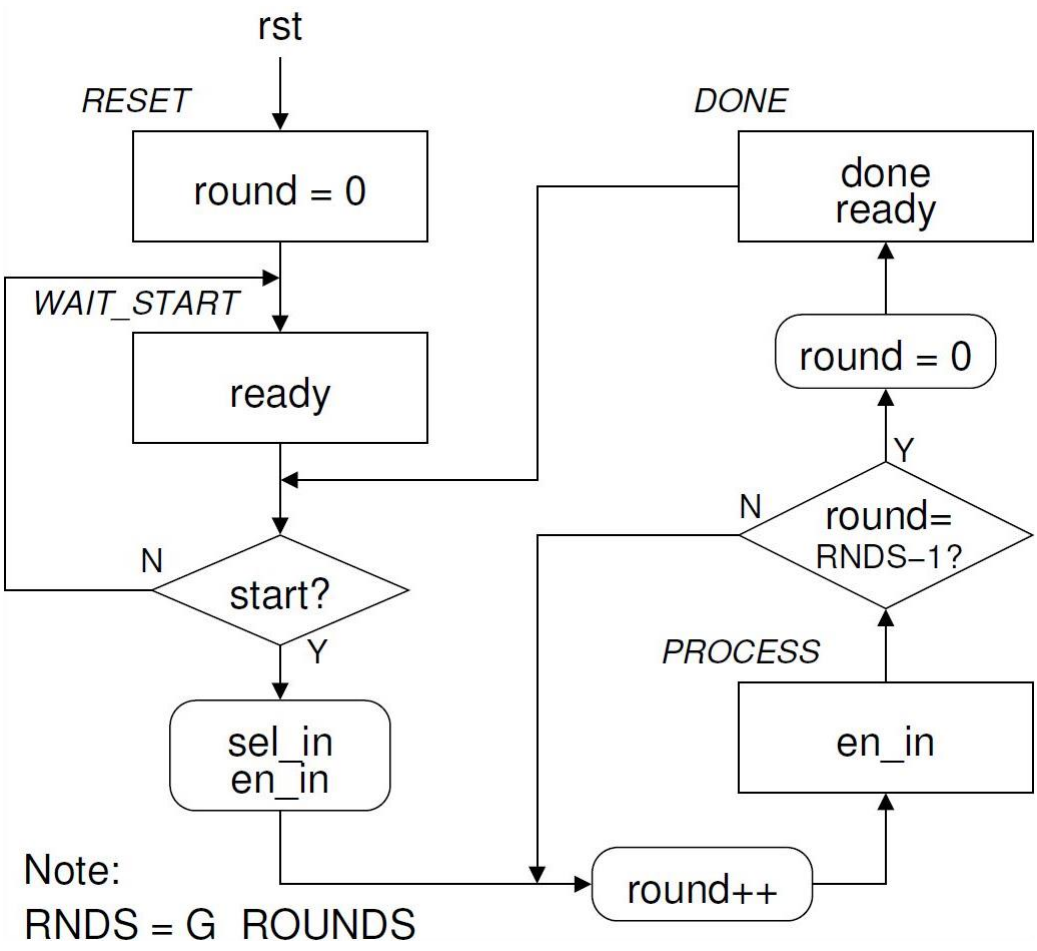

Figure 79: AES KOF Controller

8.5. Optimized Pipelined Architecture:

In the pipelined architecture, the Datapath and Controller Design from Basic Iterative Architecture were used as a starting point. The optimized pipelined design can process two 128-bit data blocks at a time. The Datapath design is same as the Basic Architecture with few modification in bus widths and addition of components. To reduce the critical path, registers were inserted in the Datapath design of Basic Architecture. The critical

path of the design lies in the round function of the AES as it is the longest path. This design was different when compared to the other designs due to the feistel structure and the data dependency between odd and even blocks. The pipelined architecture works in the following way: The value of first block and third Block are simultaneously loaded and processed then the second and fourth blocks are loaded and they are processed with the help of first and third blocks.

8.5.1. Register Insertion

Register was inserted in the round function of AES_KOF function as shown in the Figure *80*. Due to the insertion of this register the Maximum clock frequency from basic architecture was increased.

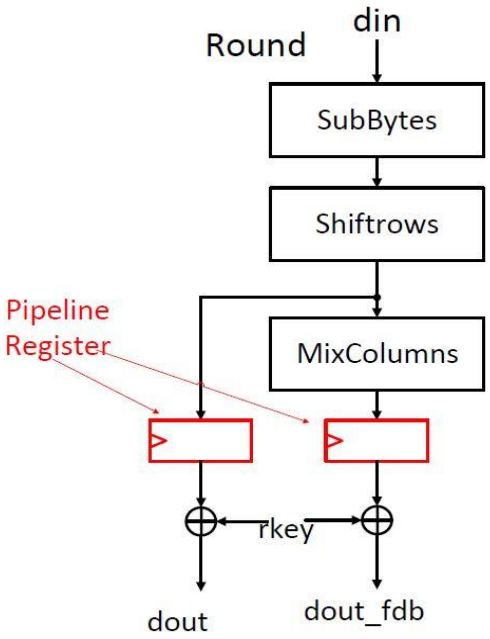

Figure 80: AES KOF Round Pipelined

8.5.2. Path Balancing

The Path balancing was done by pre-calculating the values of gamma for second block of associated data processing and pre-calculating the L value for the for second block of plaintext/ciphertext.

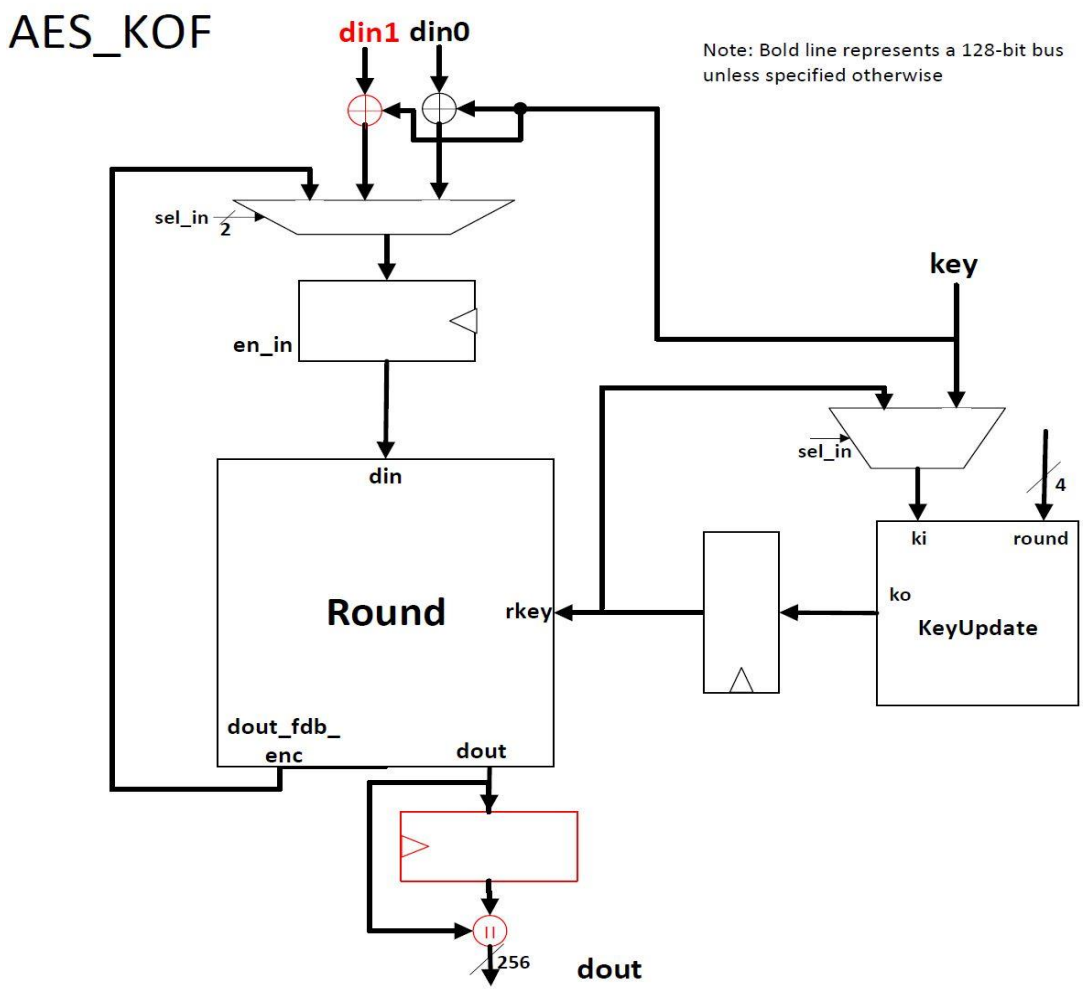

Figure 81: AES-OTR Pipelined

8.5.3. Controller Modifications

The Top level controller was slightly modified because after the critical path was shifted to the following signals after the register insertion in AES_KOF so, the signals ‘ctr’ and no_msg_flag were registered to reduce the critical path.

Changes in the controller of AES_KOF: The data processing was changed by adding four additional states. Three states were added to get the data ready entering the round function, one additional state was added to process two blocks of data at the same time and the additional. The round value increases by one per two clock cycles and same round key is available for two clock cycles. The modified controller ASM chart is as shown in the Figure *82*.

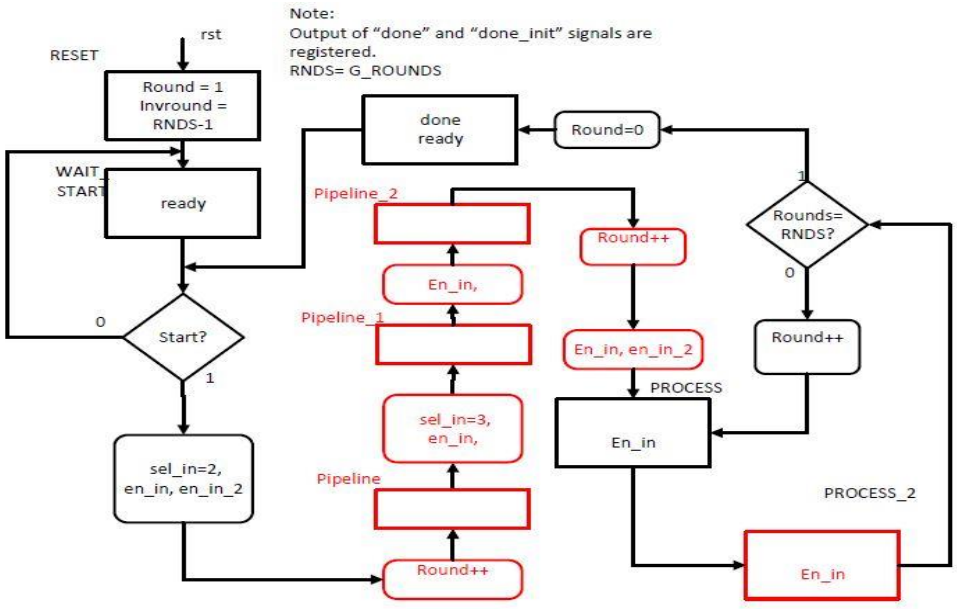

Figure 82: AES KOF Controller Pipelined

9. Performance Evaluation

Performance of each candidate was evaluated based on the area occupied by the design and improvement in the maximum clock frequency and throughput. All the following results were generated targeting the largest available FPGA in the student version of Xilinx ISE, i.e., Xilinx Virtex 6 XC6VLX75T-3FF784.

Formula for Throughput Calculation of each candidate:

Basic Architecture:

Table 2: Throughput Calculation Formula: Basic Architecture

Candidate	Throughput Formula
SCREAM	(128/11)*fclk ≈ 11.63*fclk
AES-COPA	(128/11)*fclk ≈ 11.63*fclk
AES-OTR	(128/12)*fclk ≈ 10.66*fclk
MINALPHER	(256/19)*fclk ≈ 13.47*fclk
OCB	(128/12)*fclk ≈ 10.66*fclk

fclk = Clock Frequency

Pipelined Architecture:

Table 3: Throughput Calculation Formula: Pipelined Architecture

Candidate	**Throughput Formula**
SCREAM	(256/23)*fclk ≈ 11.13*fclk
AES-COPA	(256/23)*fclk ≈ 11.13*fclk
AES-OTR	(256/24)*fclk ≈ 10.66*fclk
MINALPHER	(512/38)*fclk ≈ 13.47*fclk
OCB	(256/24)*fclk ≈ 10.66*fclk

fclk = Clock Frequency

9.1. Implementation Results

The following tables are the implementation results of Basic Architecture Design versus the Pipelined Design. Table *4*4 shows the comparison of maximum clock frequency in Basic Architecture and Pipelined Architecture.

Table 4: Maximum Clock Frequency comparison.

Candidate	Maximum Clock Frequency (MHz)	
	Basic Architecture	Pipelined Architecture
SCREAM	92	170
AES-COPA	120	210
AES-OTR	150	235
MINALPHER	168	222
OCB	172	221

Table 5: Throughput Comparison.

Candidate	Throughput (Mbits/sec)	
	Basic Architecture	Pipelined Architecture
SCREAM	1071	1892
AES-COPA	1396	2337
AES-OTR	1600	2507
MINALPHER	2264	2991
OCB	1835	2357

Table 5 shows the comparison of Throughput in Basic Architecture and Pipelined Architecture.

Table 6 shows the comparison of Area expressed in LUTs in Basic Architecture and Pipelined Architecture.

Table 6: Area Comparison (expressed in LUTs).

Candidate	Area (LUTs)	
	Basic Architecture	Pipelined Architecture
SCREAM	3644	3968
AES-COPA	4902	6484
MINALPHER	7836	11285
OCB	3312	3673
AES-OTR	5058	7443

Error! Not a valid bookmark self-reference. shows the comparison of Area expressed in Slices in Basic Architecture and Pipelined Architecture.

Table 8 shows the comparison of Throughput to Area ratio (Area expressed in LUTs) in Basic Architecture and Pipelined Architecture.

Table 7: Area Comparison (expressed in Slices)

Candidate	Area (Slices)	
	Basic Architecture	Pipelined Architecture
SCREAM	1546	2442
AES-COPA	2216	4431
MINALPHER	3974	5915
OCB	1742	2905
AES-OTR	2219	3637

Table 8: Throughput to Area ratio (Area expressed in LUTs)

Candidate	Throughput/Area (Mbits/sec LUTs)	
	Basic Architecture	Pipelined Architecture
SCREAM	0.29	0.48
AES-COPA	0.28	0.36
AES-OTR	0.32	0.34
MINALPHER	0.29	0.27
OCB	0.55	0.64

Table 9: Throughput to Area ratio (Area expressed in Slices)

Candidate	Throughput/Area (Mbits/sec Slices)	
	Basic Architecture	Pipelined Architecture
SCREAM	0.69	0.77
AES-COPA	0.63	0.53
AES-OTR	0.72	0.69
MINALPHER	0.57	0.51
OCB	1.05	0.81

Table *9*9 shows the comparison of Throughput to Area ratio (Area expressed in LUTs) in Basic Architecture and Pipelined Architecture.

9.2. Analysis of Results

From the tabulated data the graphs have been visualized as shown in Figure *83*, Figure *84*, Figure *85*, Figure *86*, Figure *87*, and Figure *88*. We can see that the candidate SCREAM has gained 84% increase in its maximum clock frequency. AES-COPA has gained 75% increase in its maximum clock frequency. MINALPHER has gained 39.88% increase in its maximum clock frequency. OCB has gained 28.49% increase in its maximum clock frequency. AES-OTR has gained 56% increase in its maximum clock frequency.

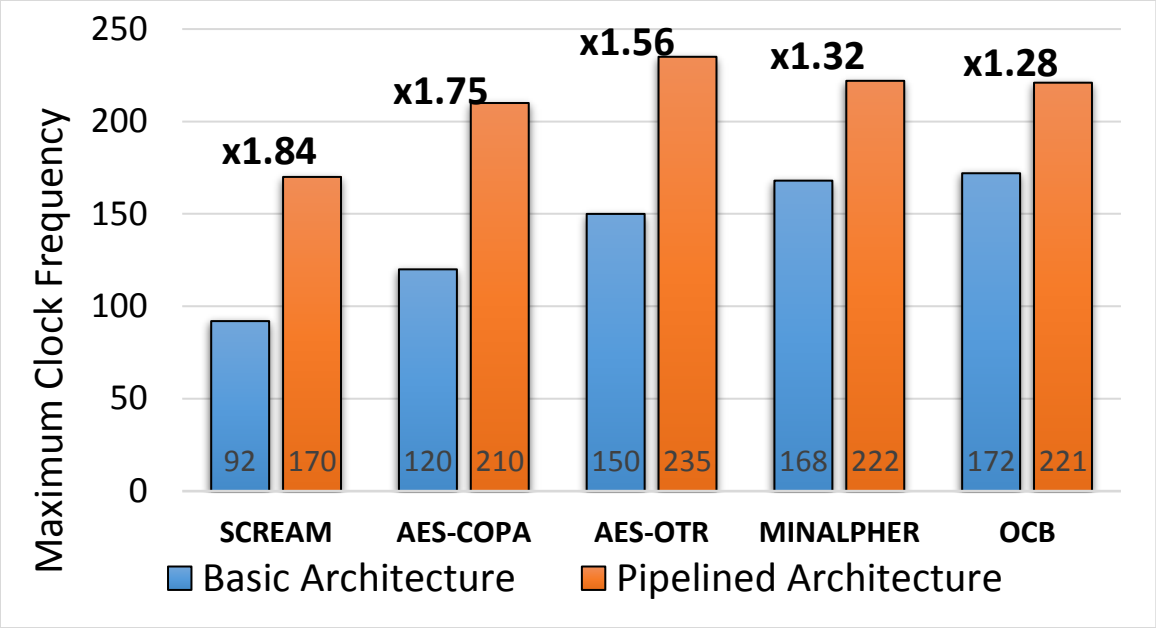

Figure 83: Plot - Maximum Clock Frequency- Basic Architecture vs Pipelined Architecture

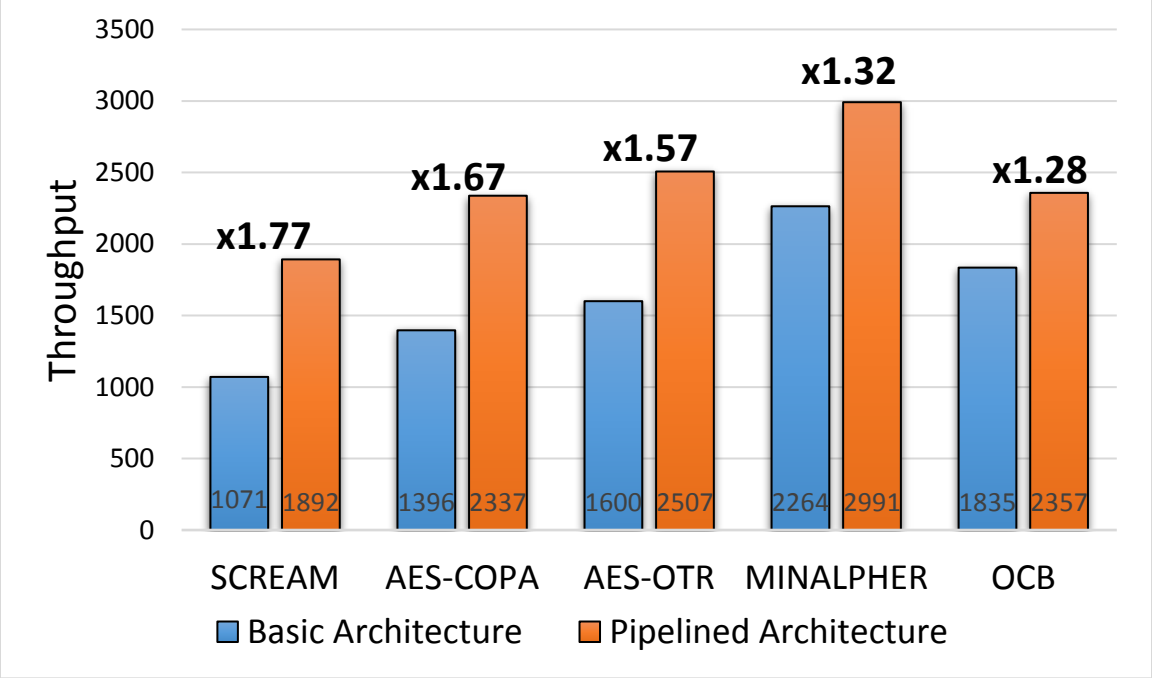

Figure 84: Plot - Throughput- Basic Architecture vs Pipelined Architecture

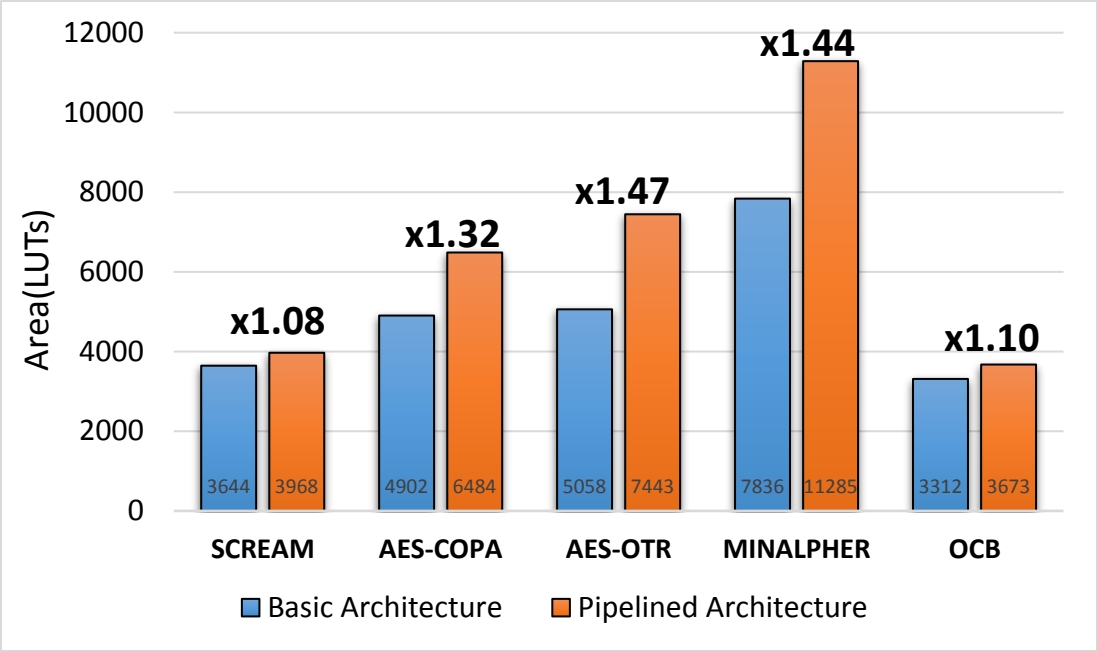

Figure 85: Plot – Area Comparison (expressed in LUTs)

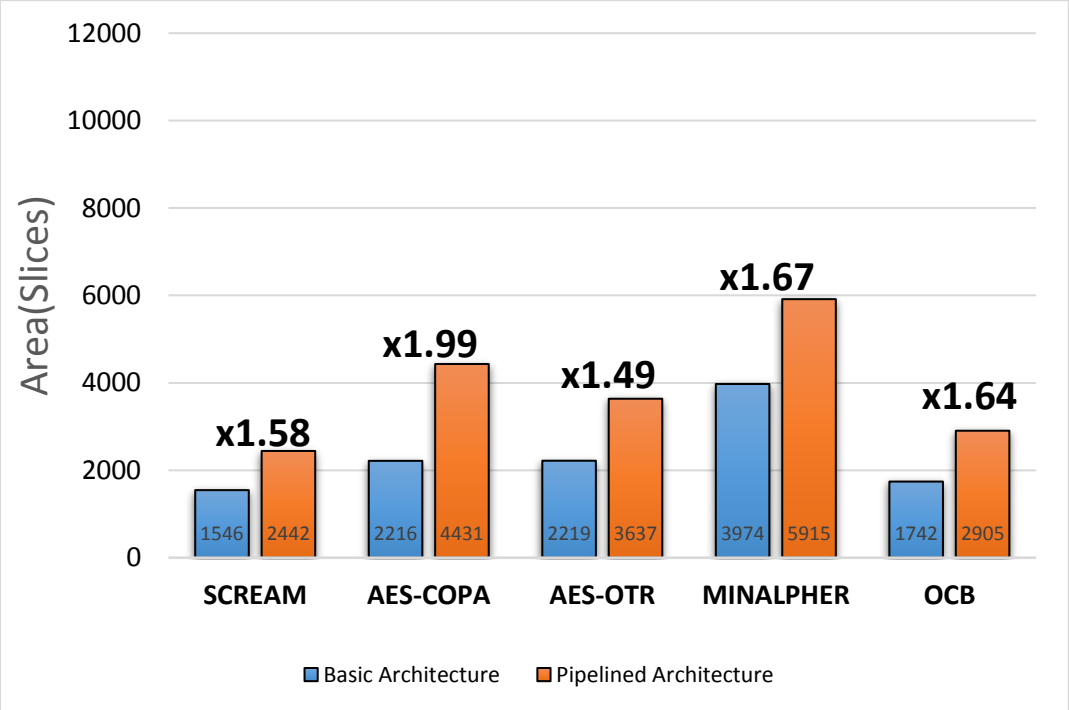

Figure 86: Plot - Area Comparison (expressed in Slices)

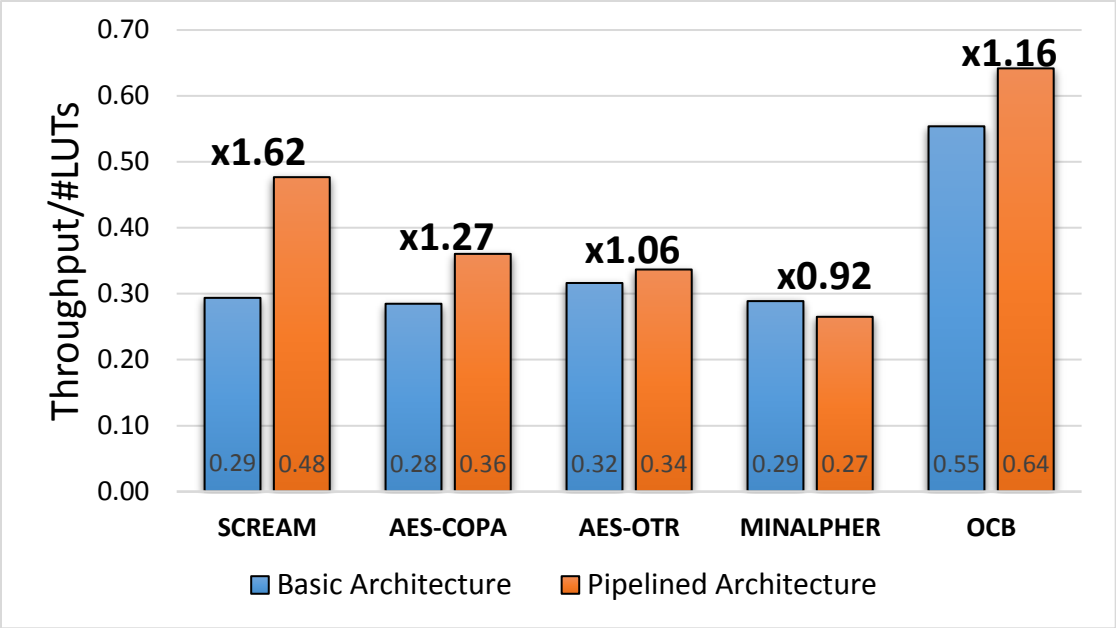

Figure 87: Plot - Throughput to Area Ratio (Area expressed in LUTs)

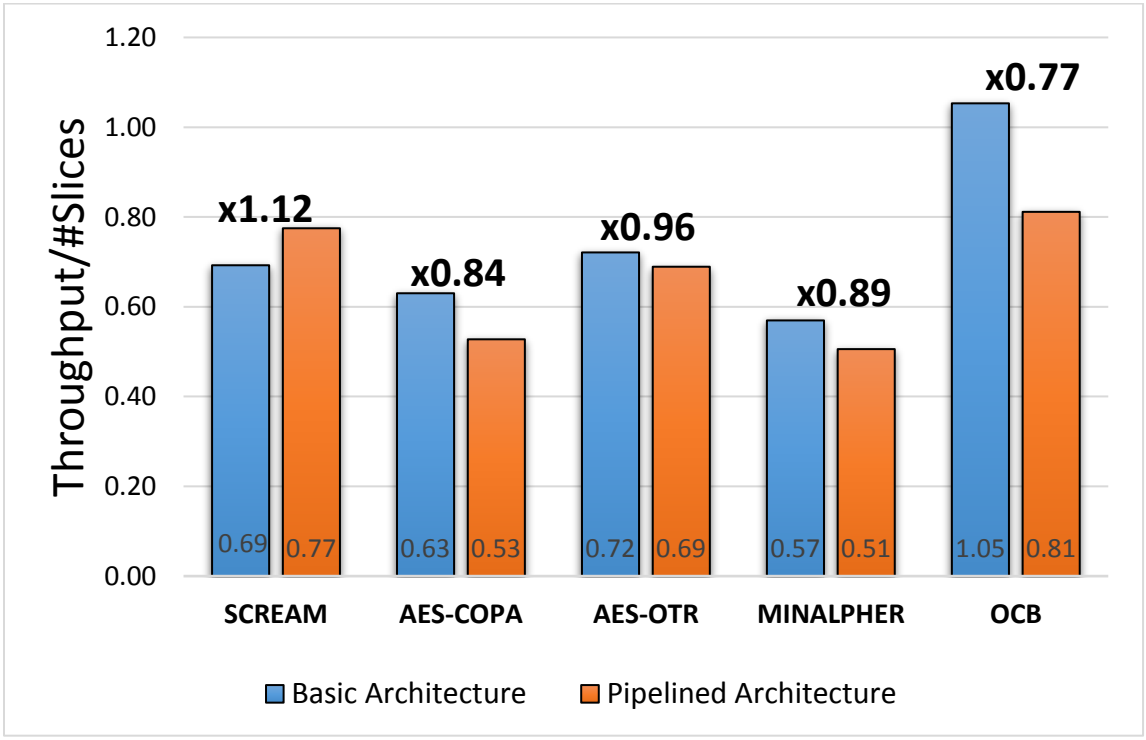

Figure 88: Plot - Throughput to Area Ratio (Area expressed in Slices)

10. Conclusions

The improvement in Maximum clock frequency and Throughput depends on the algorithm and its critical path. We can say that the candidate with the lowest value in the Basic Architecture has achieved the highest amount of gain in the percentage increase of maximum clock frequency and throughput in Pipelined Architecture. Our results have demonstrated the improvement in the Clock Frequency by a factor varying from x1.28 for OCB to x1.84 for SCREAM, the improvement in the Throughput by a factor varying from x1.28 for OCB to x1.77 for SCREAM, and the improvement in the Throughput to Area ratio (with Area expressed using LUTs) by a factor varying from x0.92 for Minalpher to x1.62 for SCREAM. Improvement in Throughput/#LUTs was observed in four candidates except Minalpher. Improvement in Throughput/#Slices was observed in one candidate i.e. SCREAM. Pipelined implementation performance wise, the top 3 candidates are SCREAM, AES-COPA, and AES-OTR. The candidates MINALPHER and OCB fall at the bottom as there was not much improvement in their maximum clock frequency and throughput after pipelining.

Bibliography

[1] M. Liskov, R. L. Rivest, and D. Wagner, Tweakable block ciphers, J. Cryptology, 24 (2011), pp. 588–613.

[2] Shimon Even and Yishay Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. Journal of Cryptology, 10(3):151{162, 1997.

[3] Kaoru Kurosawa. Power of a public random permutation and its application to authenticated-encryption. IACR Cryptology ePrint Archive, 2002:127, 2002.

[4] Kaoru Kurosawa. Power of a public random permutation and its application to authenticated encryption. IEEE Transactions on Information Theory, 56(10):5366-5374, 2010.

[5] ATHENa results database. http://cryptography.gmu.edu/athenadb/. Automated Tool for Hardware EvaluatioN project.

[6] Vincent Grosso, Gaetan Leurent, Francois-Xavier Standaert, Kerem Varici, Francois Durvaux, Lubos Gaspar, and Stephanie Kerckhof. SCREAM and iSCREAM. Submission to CAESAR, March 2014.

[7] Ekawat Homsirikamol, William Diehl, Ahmed Ferozpuri, Farnoud Farahmand, Malik Umar Sharif, and Kris Gaj. Gmu hardware api for authenticated ciphers. Cryptology ePrint Archive, Report 2015/669, 2015. http://eprint.iacr.org/.

[8] Ted Krovetz and Phillip Rogaway. The OCB Authenticated-Encryption Algorithm. Submission to CAESAR, May 2014.

[9] Kazuhiko Minematsu. AES-OTR v3.1. NEC Corporation, Japan. Submission to CAESAR, September 2016.

[10] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Elmar Tischhauser, and Kan Yasuda. AES-COPA v.2. Dept. Electrical Engineering, ESAT/COSIC, KU Leuven, Belgium, iMinds, Belgium, DTU Compute, Technical University of Denmark, Denmark, NTT Secure Platform Laboratories, Japan. Submission to CAESAR.

[11] Yu Sasaki, Yosuke Todo, Kazumaro Aoki, Yusuke Naito, Takeshi Sugawara, Yumiko Murakami, Mitsuru Matsui, Shoichi Hirose. Minalpher v1.1. NTT Secure Platform Laboratories, Mitsubishi Electric Corporation, University of Fukui, Submission to CAESAR, August 2015.

[12] K. Gaj and P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," Proc. 3rd Advanced Encryption Standard Conference, New York, April 2000, pp. 40-54.

[13] P. Chodowiec, P. Khuon, and K. Gaj, "Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer-Round Pipelining," ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays, Monterey, CA, Feb. 2001, pp. 94-102.

[14] K. Gaj and P. Chodowiec, "Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard using Field Programmable Gate Arrays," LNCS 2020, Progress in Cryptology - CT-RSA 2001, Ed. D. Naccache, RSA Conference 2001 - Cryptographers' Track, San Francisco, Apr. 2001, pp. 84-99.

[15] E. Homsirikamol, M. Rogawski, and K. Gaj, "Throughput vs. Area Trade-offs in High-Speed Architectures of Five Round 3 SHA-3 Candidates Implemented Using Xilinx and Altera FPGAs," in LNCS 6917, Cryptographic Hardware and Embedded Systems - CHES 2011, Nara, Japan, Sep. 28-Oct. 1, pp. 491-506.

[16] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M.U. Sharif, and K. Gaj, "A Universal Hardware API for Authenticated Ciphers," 2015 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2015, Mayan Riviera, Mexico, Dec. 7-9, 2015.

[17] E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, M.U. Sharif, and K. Gaj "GMU Hardware API for Authenticated Ciphers," Cryptology ePrint Archive: Report 2015/669, first version - July 2015.

[18] W. Diehl and K. Gaj, "RTL Implementations and FPGA Benchmarking of Three Authenticated Ciphers Competing in CAESAR Round Two," 19th Euromicro Conference on Digital System Design - DSD 2016, Limassol, Cyprus, Aug. 31-Sep. 2, 2016.

Curriculum Vitae

Sanjay Deshpande received his Bachelors of Technology degree from the Jawaharlal Nehru Technological University, India in 2014. He was ranked among the top 3 students in the university. He has been involved in teaching various undergraduate courses at George Mason University. He was also involved as a Research Assistant in Cryptographic Engineer Research Group (CERG) with interest in High Speed implementation of CAESAR candidates.