Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Electrical Engineering Department



|  |
| --- |
| **Computer Architecture (2)**  **Encryption Algorithm Using Pipelining** |

|  |  |  |
| --- | --- | --- |
| *Authors:* |  | *Supervisor:* |
| Tala Ibraheem  Tala Hashem  Ibrahim Enaya | 20190891  20190406  20191117 | Dr. Awos Kanan |

*May 29th, 2022*

***Abstract***

*A pipeline design in general is a continuous overlapped movement used in many platforms for the purpose of increasing the throughput of sort of instructions. In this report we will implement the pipeline design to improve an encryption algorithm that provides high security standards in short time.*

Table of Contents

[1 Introduction 2](#_Toc104746143)

[1.1 Proposed design 2](#_Toc104746144)

[2 Implementation 3](#_Toc104746145)

[2.1 block diagram 3](#_Toc104746146)

[2.2 Design and analysis 9](#_Toc104746147)

[2.3 Test runs and discussion 11](#_Toc104746148)

[2.4 performance analysis 12](#_Toc104746149)

[speedup & throughput 12](#_Toc104746150)

[Conclusion and Future Work 12](#_Toc104746151)

# Introduction

In this project we will discuss how we can implement a pipeline design using Verilog to achieve the highest possible throughput.

Our design will mainly be talking about encryption algorithm. Encryption is one of the cryptographic algorithms which tries to protect the system from any attacks. As known, there are many types of encryption algorithms, therefore, we will focus on one famous algorithm called AES, refers to Advance Encryption Standard.

However, the design includes so many complicated operations that goes deeply in cryptography major, so we’ll be using a simplified design with simple operations that are enough to define and illustrate the concept of pipelining.

Our model consists of five stages each stage is responsible for some sort of operations. The effectiveness of our pipeline model occurs by overlapping the operations of our algorithm without waiting for each text to execute separately.

## Proposed design

Our Verilog code consists of one main module includes five stages of our pipeline design. This module takes three inputs, text block, key block and the clock. And it ends up with one output which is cipher text. The first stage of our model starts by storing all inputs in memory then at every positive edge, a partial text of 16 char and partial key of 5 char are loaded from memory to start executing Round1 of our encryption algorithm. Also a pc counter was defined which is responsible for loading a new text and new key every cycle. Each round maintains same sort of operations, starting with Caesar cipher followed by XOR, shifting to the right and swapping the text.

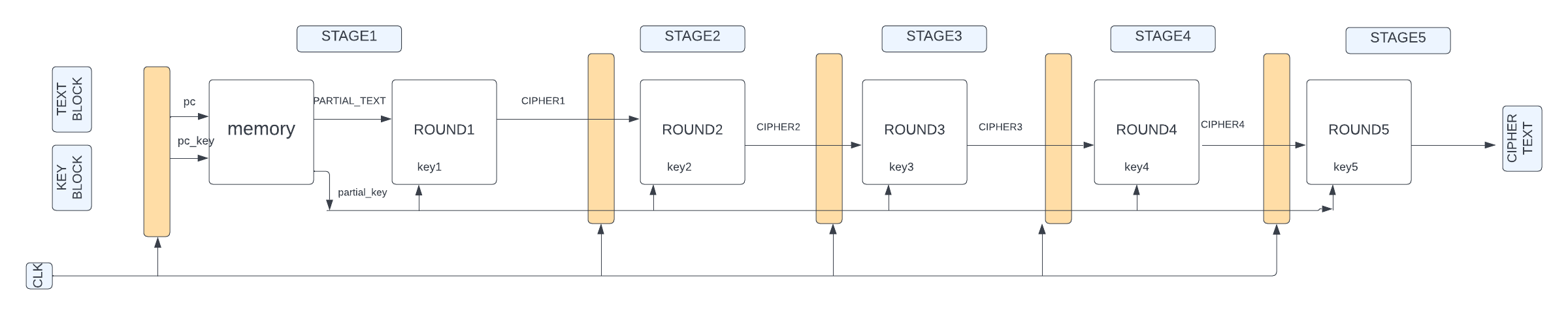
Moreover, registers were added between every two stages to save values of previous stage and prepare them to the next stage. So the registers were connected to the clock and at the negative edge of each clock cycle the data resulted from current stage is written on flipflops and at the next positive edge, data is read from flipflop and ready to proceed to next stage.

The next four stages present four rounds with same operations and same analysis, however the input of each round varies depending on the output of previous round. The last stage outcome is the encrypted text or what is called the cipher text.

# Implementation

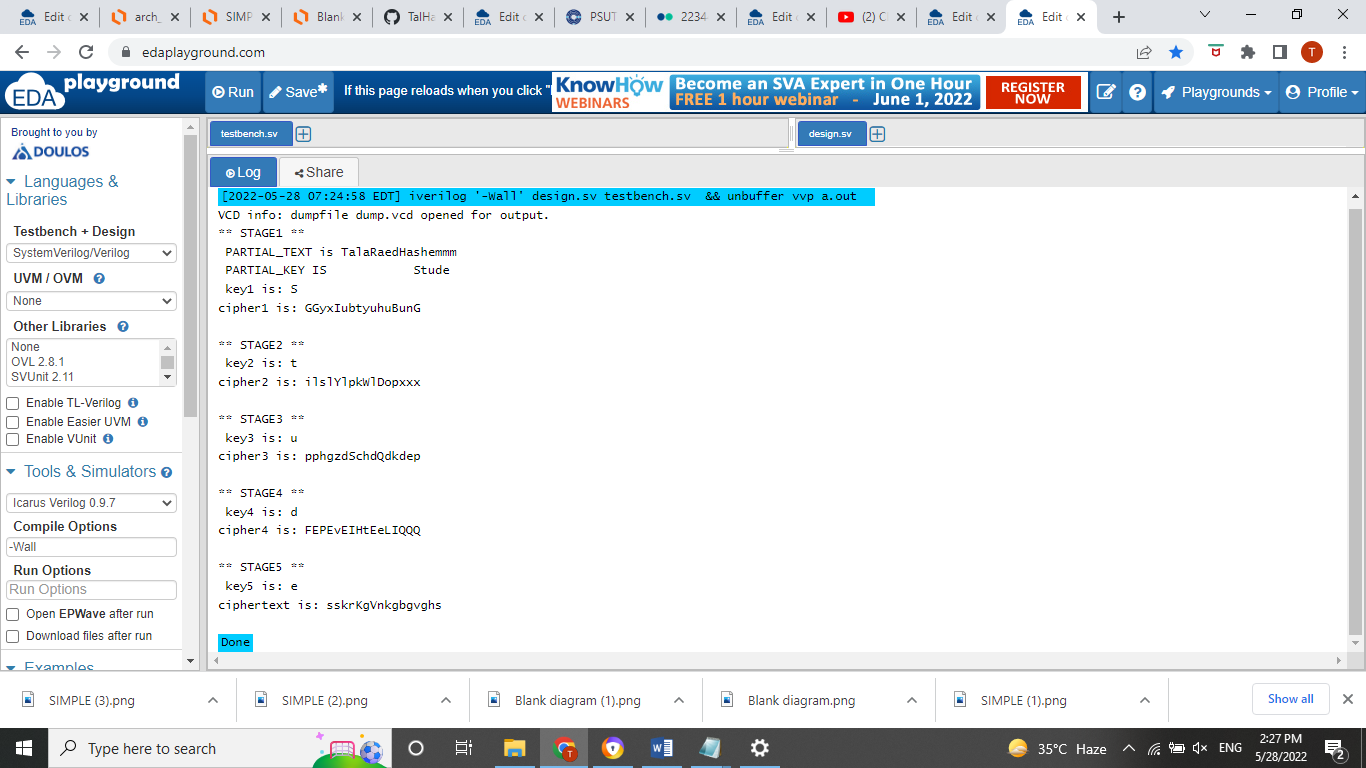
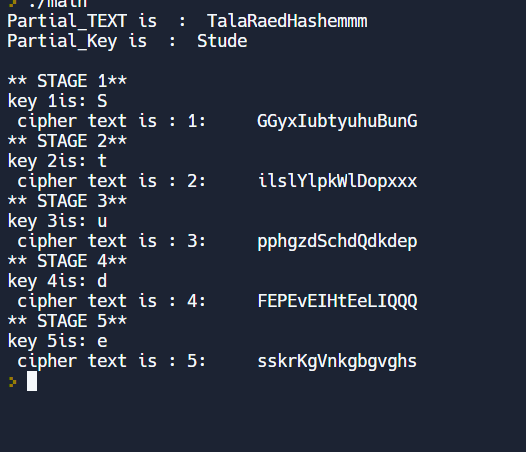
## block diagram

**Low level details:**

This block diagram shows a very simple level details. It ulistrates the tracing of **one** text only through five stages.

Low level detailed block

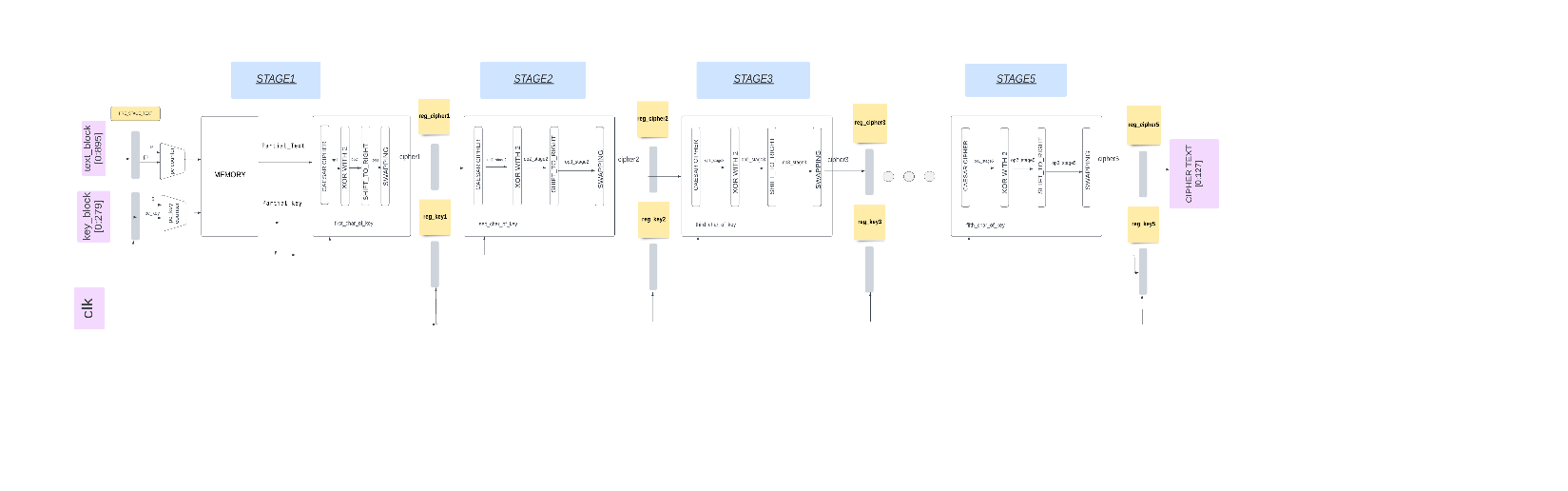
* Verilog screenshot equivalent to the simple block diagram showing tracing of **one** text through five stages && C++ code equivalent to the simple block diagram:



**High level detail:**

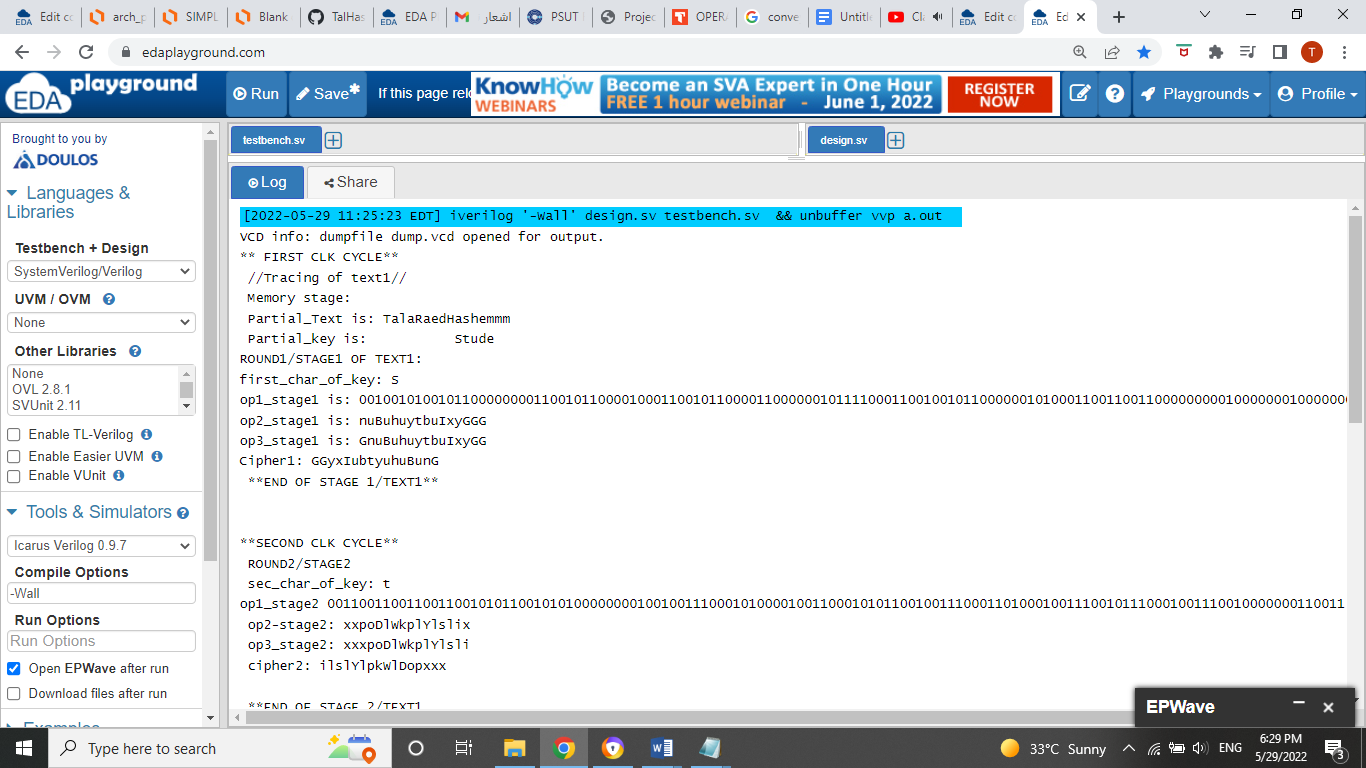
* This block diagram shows the operations of each round.

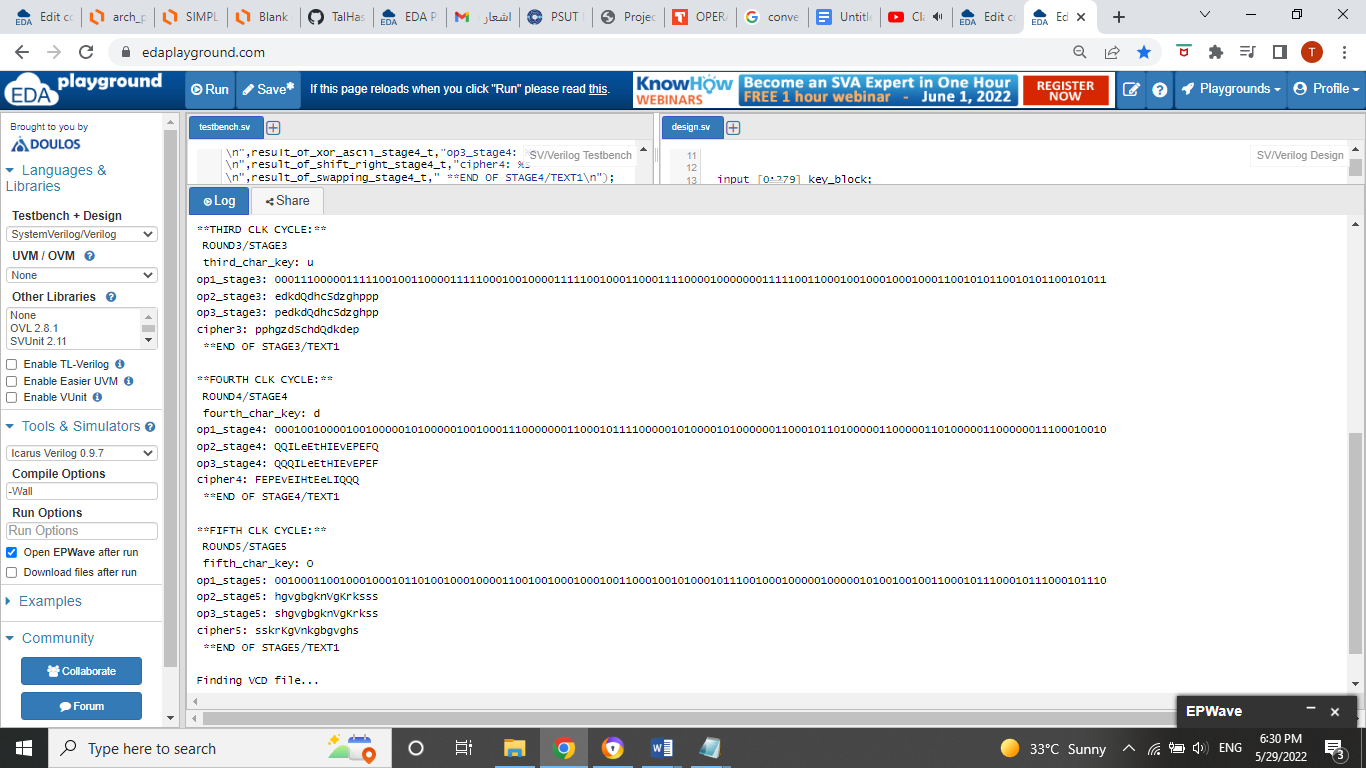
Note: the diagram is implemented as PNG image with higher quality.



High level detailed block

* Screenshot of Verilog code showing tracing of **one** text with details:



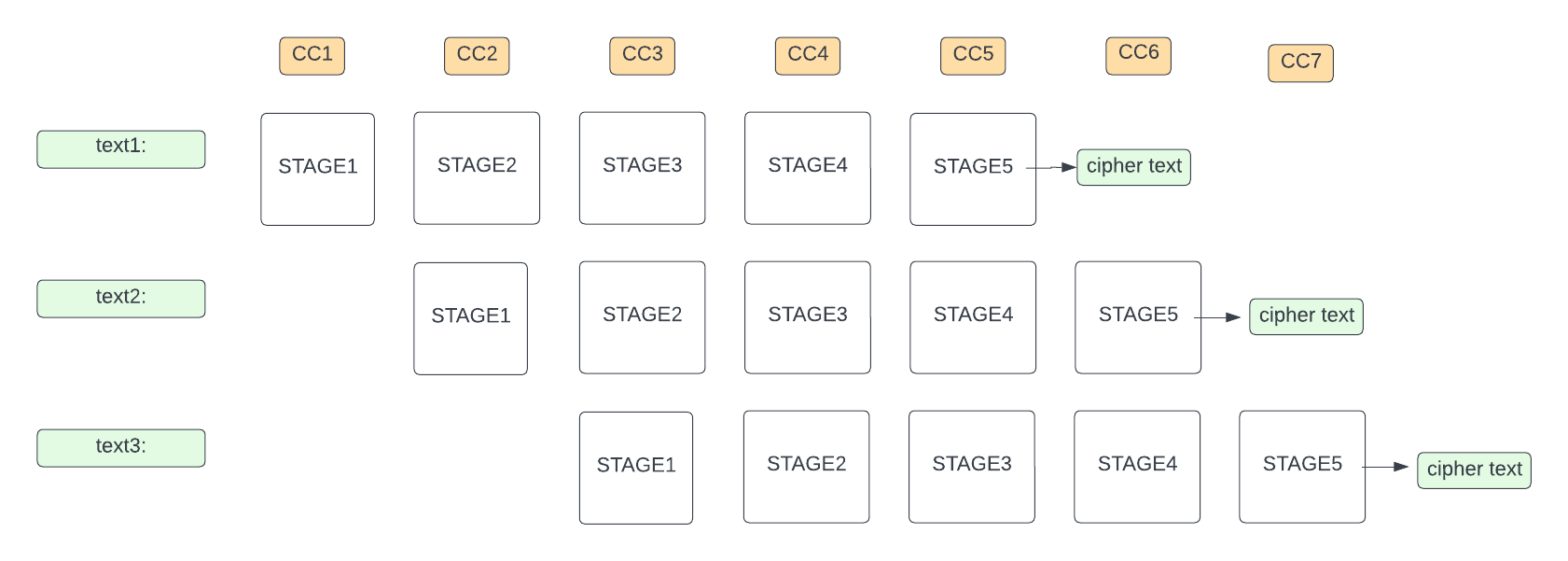


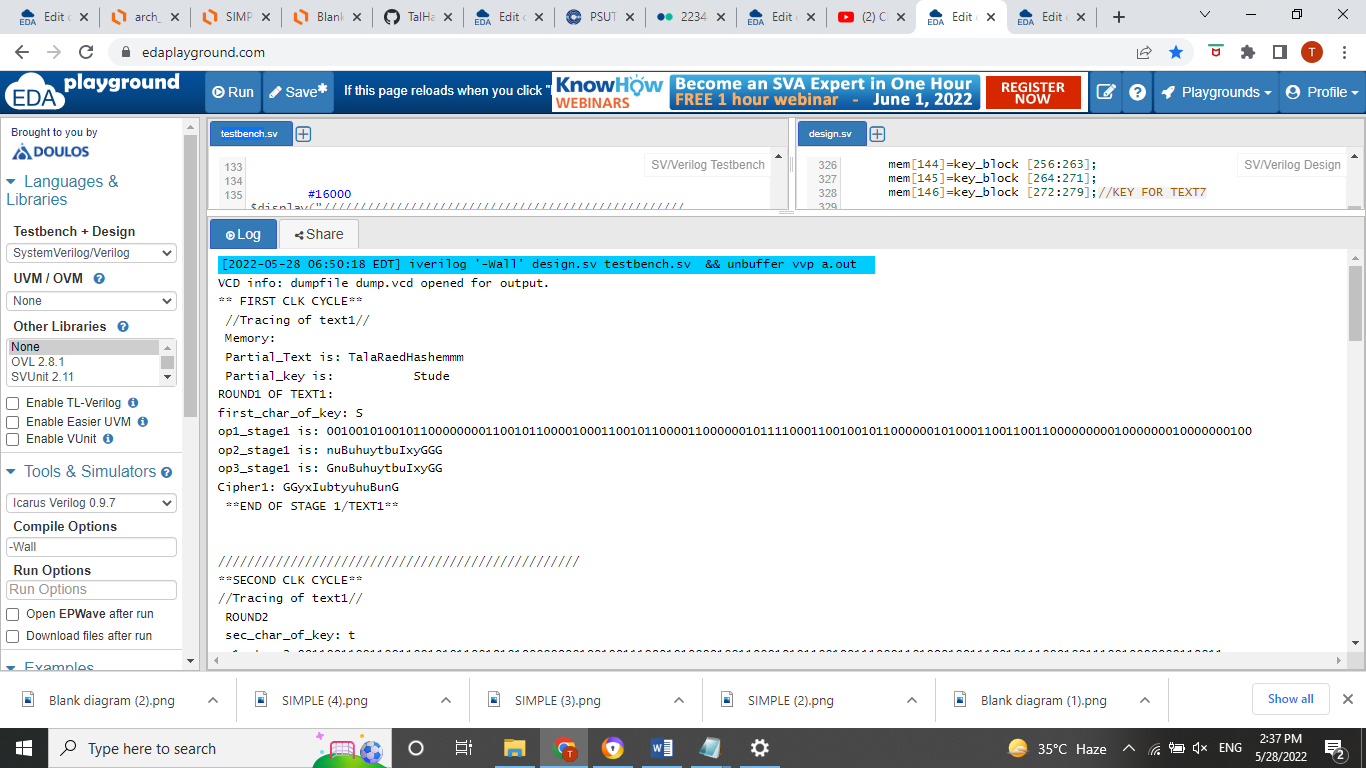
**Pipelining details:**

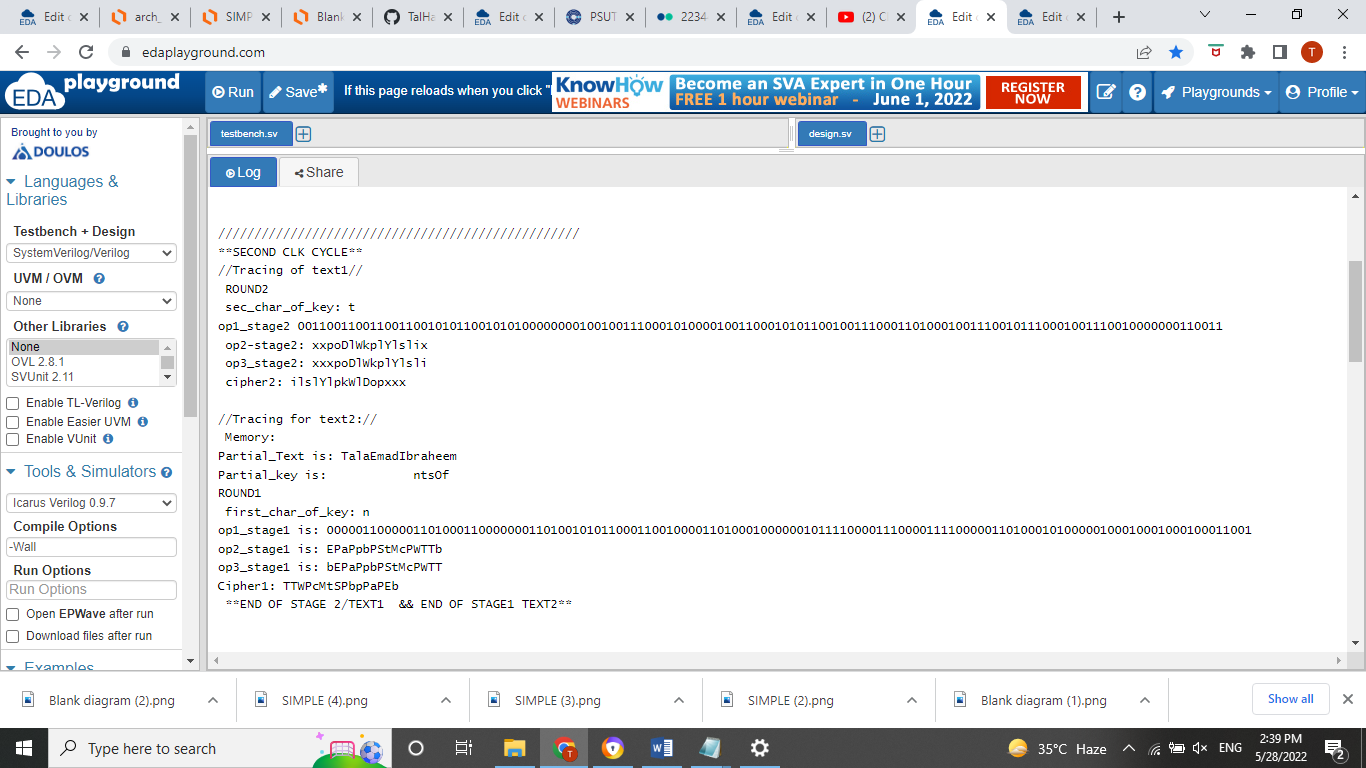
* Block diagram showing **pipeline** tracing up to 3-texts:

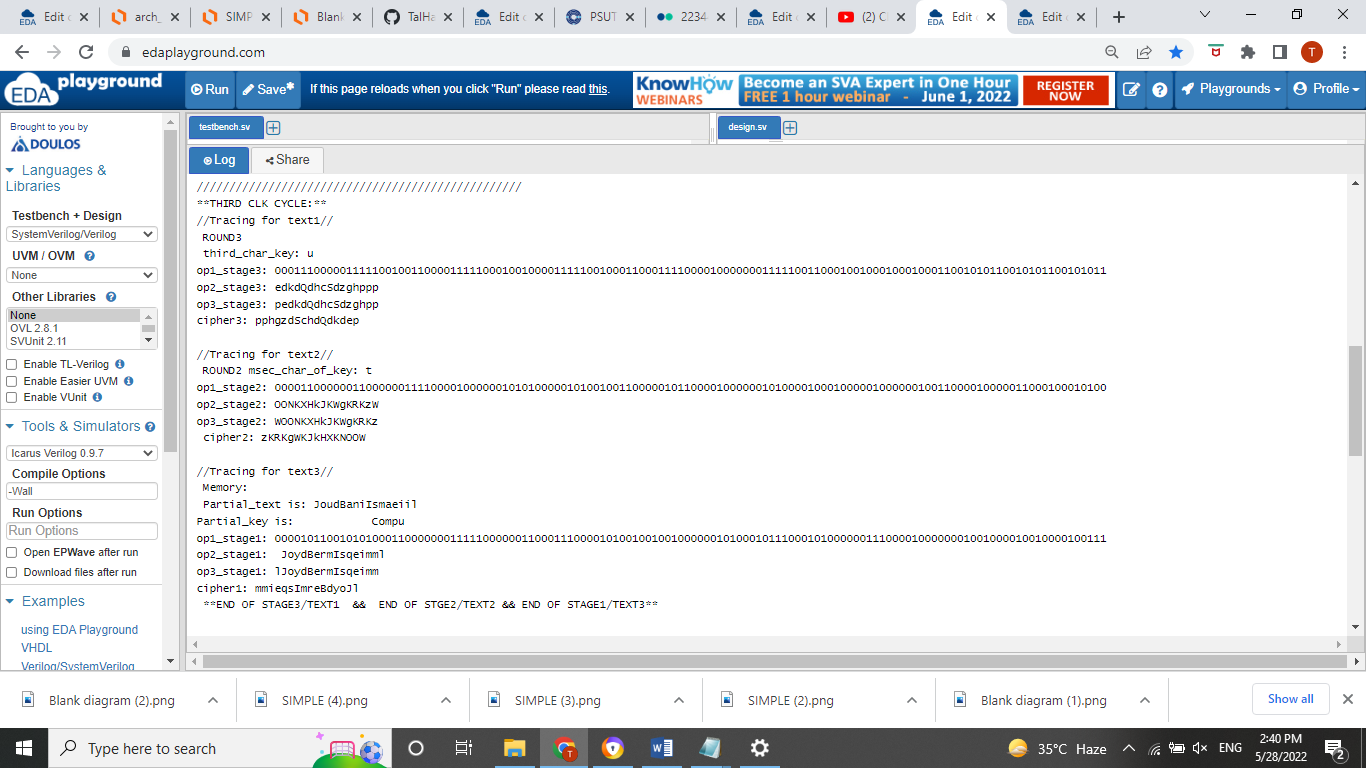
NOTE: #of cycles=#of stages+(n-1), such that n is number of texts

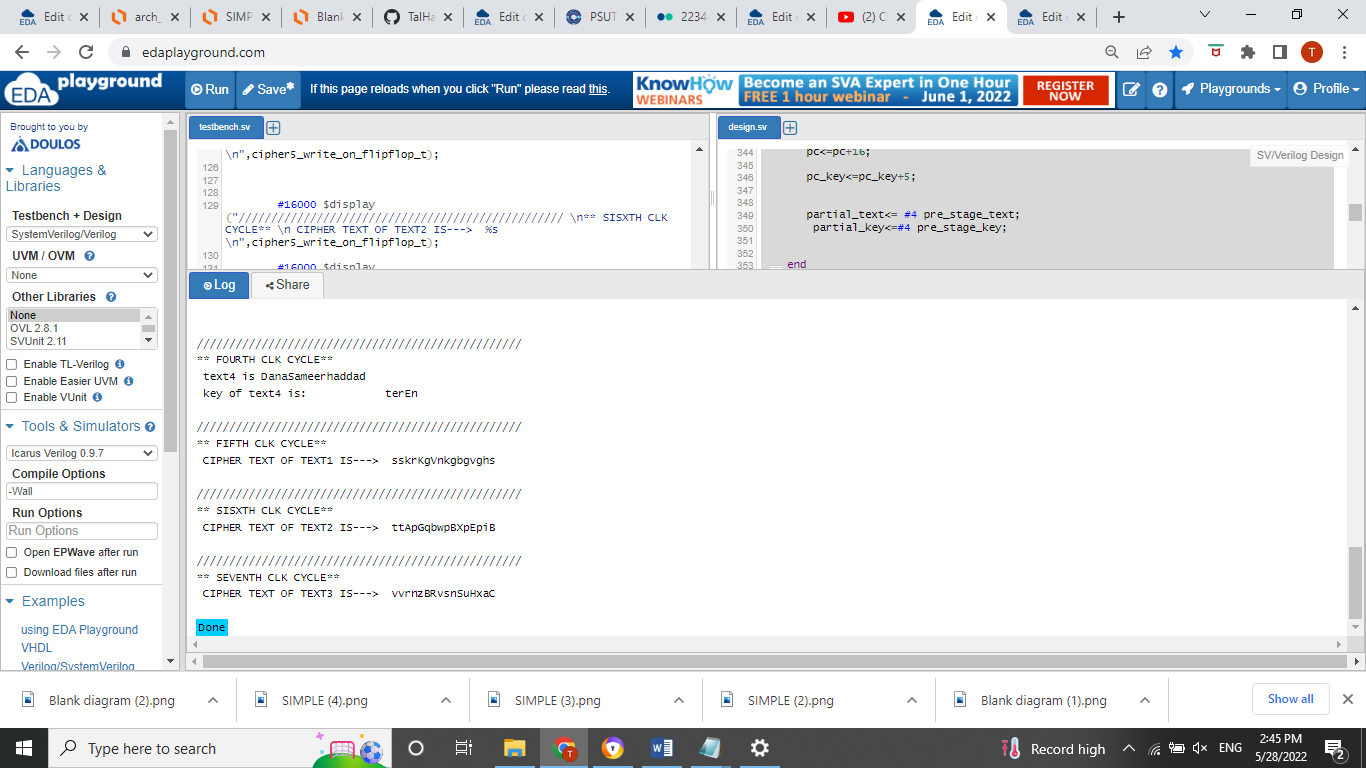
#of cycles=5+(3-1)=7 clk cycles



* Verilog screenshot showing **pipeline** tracing up to 3-cycles with the **execution of each operation** in each stage:

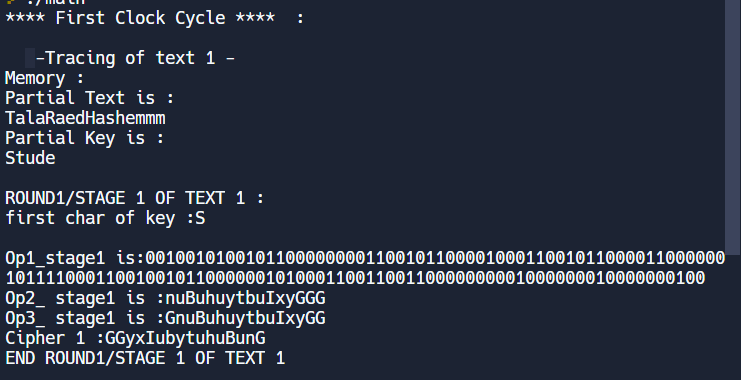




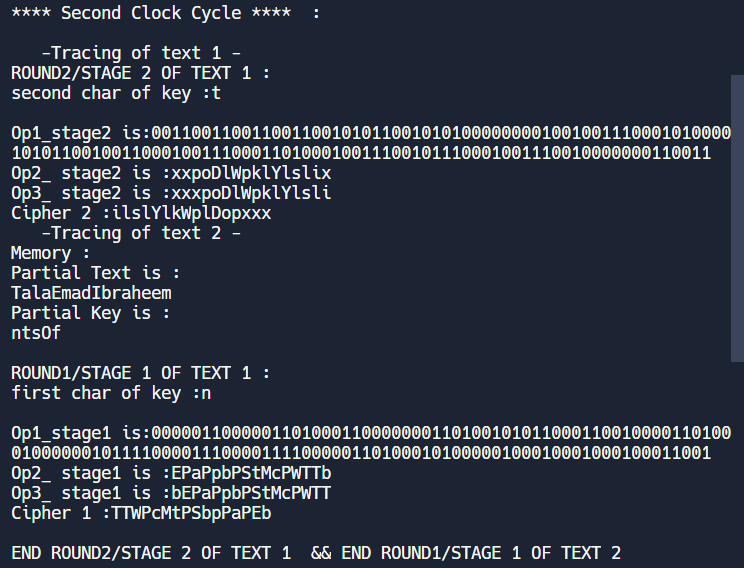
****

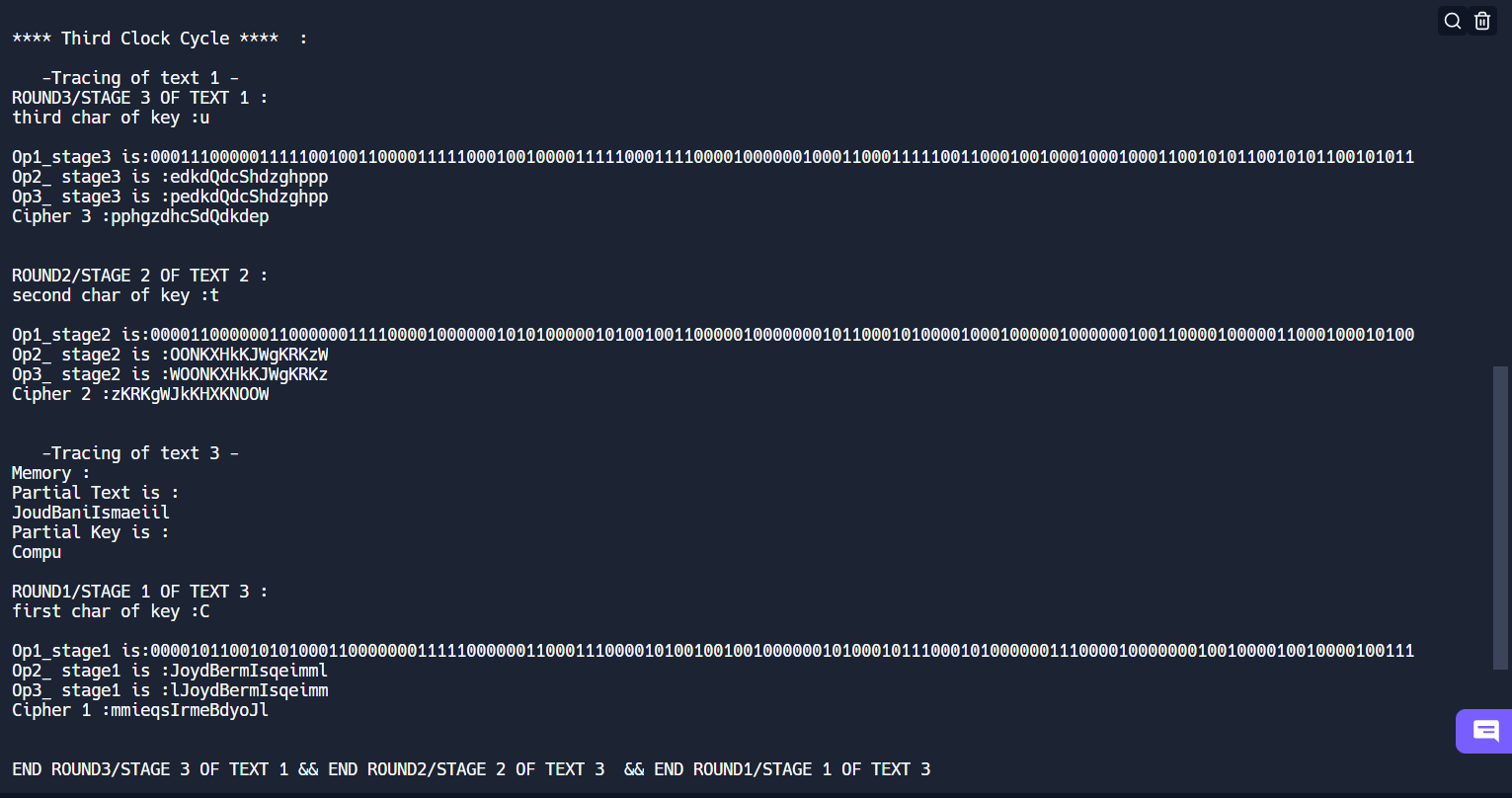
C++ screenshot showing our design up to 3-cycles:

**First cycle:**

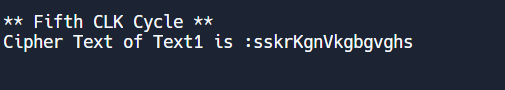


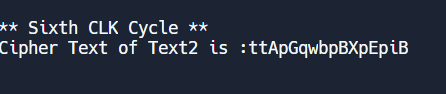
**Second cycle:**

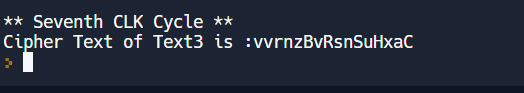


****  
  
**Third cycle:**

**Fifth, sixth,seventh cycles :**

****





## Design and analysis

Inputs:

* Text\_block[896 bits]: in our design, the text\_block consists of seven plain texts, each text is 128bits (16 char/text). These seven texts are supposed to enter our pipeline design respectively every one clock cycle.
* Key\_block[280 bits]: each plain text has a unique key of 40bits (5 char/key), and since we have seven plain texts, then seven keys are required where they get into our design as a block of 280 bits.
* Clk: the clock synchronizes the flow of our pipeline design.

Output:

* Cipher\_text[128 bits]: the encrypted text.

Registers:

* Pre\_stage\_text: store the text\_block in memory
* Pre\_stage\_key: stores the key\_block in memory
* Pc: counter of memory locations for plain texts.
* Pc\_key: counter of memory locations for keys.
* Partial\_text: loads each text (128bits /16 char) from memory.
* Partial\_key: loads the key of each plain text (40 bits/5 char) from memory.
* Reg\_cipher1: saves the output of stage1.
* Reg\_key1: saves the key of the text in stage1.
* Reg\_cipher2: saves the output of stage2.
* Reg\_key2: saves the key of the text in stage2.
* Reg\_cipher3: saves the output of stage3.
* Reg\_key3: saves the key of the text in stage3.
* Reg\_cipher4: saves the output of stage4.
* Reg\_key4: saves the key of the text in stage4.
* Reg\_cipher5: saves the CIPHER\_TEXT.

Wires (used for tracing issues):

* Op1\_stage1: result of Caesar cipher in stage1
* Op2\_stage1: result of XOR operation in stage1
* Op3\_stage1: result of shift\_to\_the\_right operation in stage1
* Cipher1: result of stage1
* Op1\_stage2: result of Caesar cipher in stage2
* Op2\_stage2: result of XOR operation in stage2
* Op3\_stage2: result of shift\_to\_the\_right operation in stage2
* Cipher2: result of stage2
* Op1\_stage3: result of Caesar cipher in stage3
* Op2\_stage3: result of XOR operation in stage3
* Op3\_stage3: result of shift\_to\_the\_right operation in stage3
* Cipher3: result of stage3
* Op1\_stage4: result of Caesar cipher in stage4
* Op2\_stage4: result of XOR operation in stage4
* Op3\_stage4: result of shift\_to\_the\_right operation in stage4
* Cipher4: result of stage4
* Op1\_stage5: result of Caesar cipher in stage5
* Op2\_stage5: result of XOR operation in stage5
* Op3\_stage5: result of shift\_to\_the\_right operation in stage5
* Cipher5: result of stage5

Stage 1:

* First: load partial\_text and partial\_key from memory, then increment pc counter/ pc\_key counter
* Second: start Round1 of our encryption algorithm which consists of four operations.
* OP1: Caesar cipher

Op1= (plaintext+ key1) %52

* Op2: XOR

Op2= op1 xor 2

* Op3: shift one bit to the right

Op3= op3>>1

* Op4: swapping

Swap first char with last char & swap second char with before last char, so on.

Stage 2:

* Start Round2 of our encryption algorithm which consists of four operations.
* Op1
* Op2
* Op3
* Op4

Stage 3:

* Start Round3 of our encryption algorithm which consists of four operations.
* Op1
* Op2
* Op3
* Op4

Stage 4:

* Start Round4 of our encryption algorithm which consists of four operations.
* Op1
* Op2
* Op3
* Op4

Stage 5:

* Start Round5 of our encryption algorithm which consists of four operations.
* Op1
* Op2
* Op3
* Op4

## Test runs and discussion

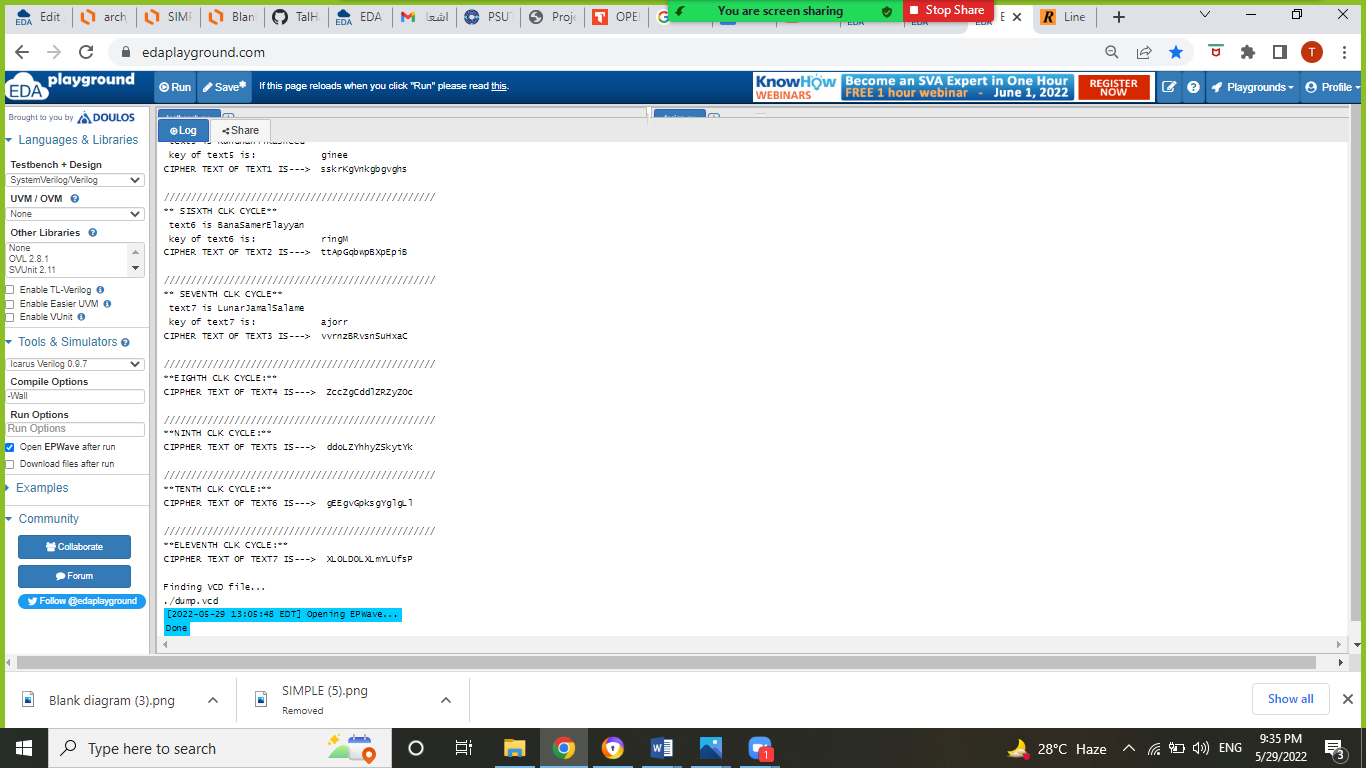
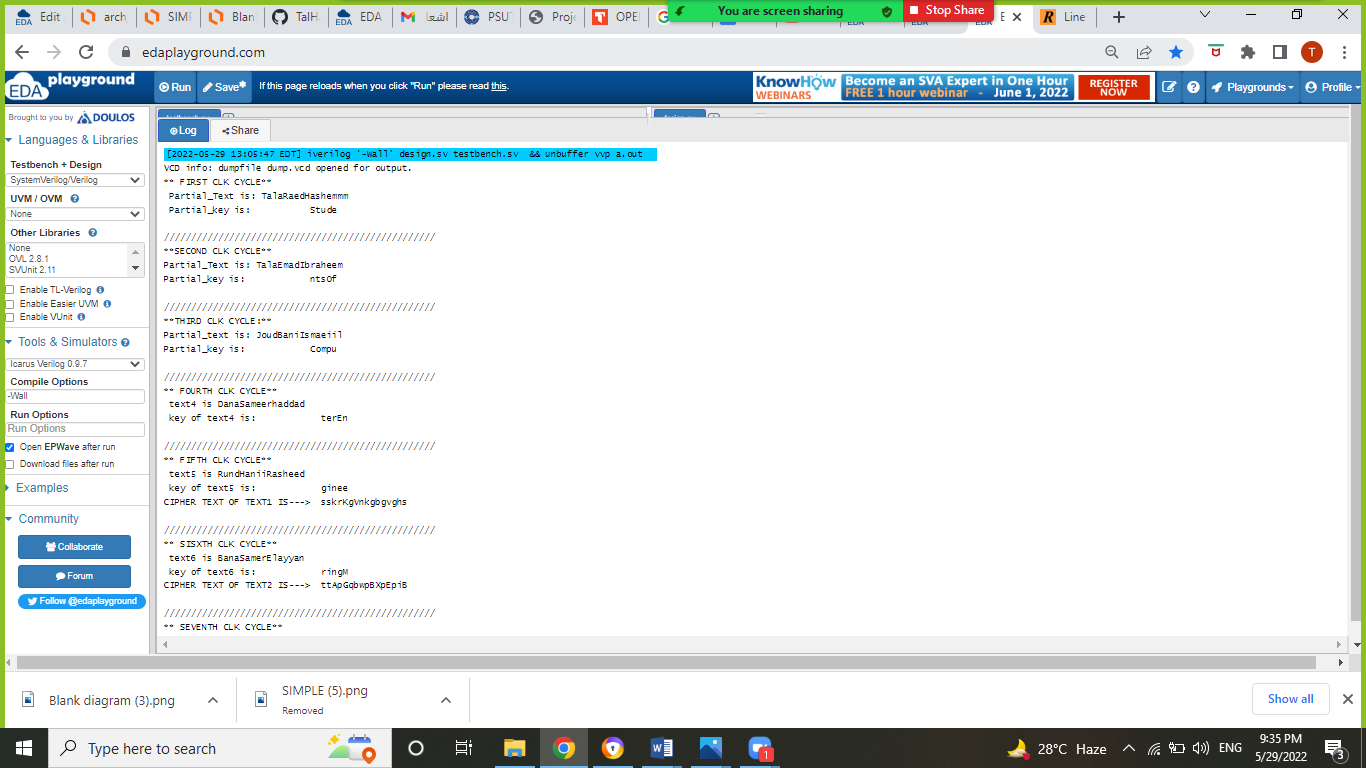
At every clock cycle, a new text of 16 char and a key of 5 char are loaded from memory, pass through the five stages. The first text will be executed as a cipher text after five clk cycles, then followed by the next cipher at the sixth clk cycle and so one for the remaining ciphers.

So we conclude that number of cycles needed to execute seven cipher texts of seven plain text is:

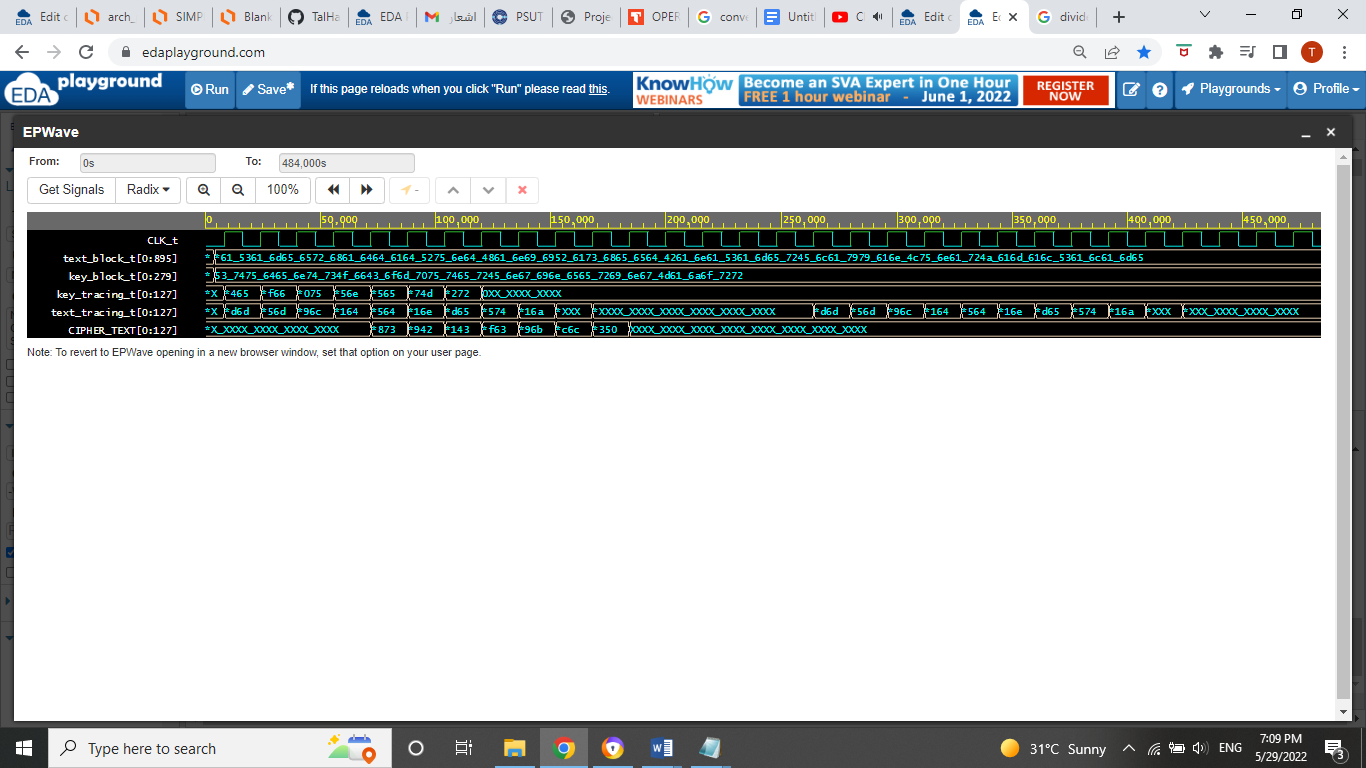
# of cycles=5+(n-1), such that n is # of texts

# of cycles=5+(7-1)

#of cycles=11 cycle



* The following screenshot illustrates the wave form of our design synchronized with a clk of period=16000



Waveform

## performance analysis:

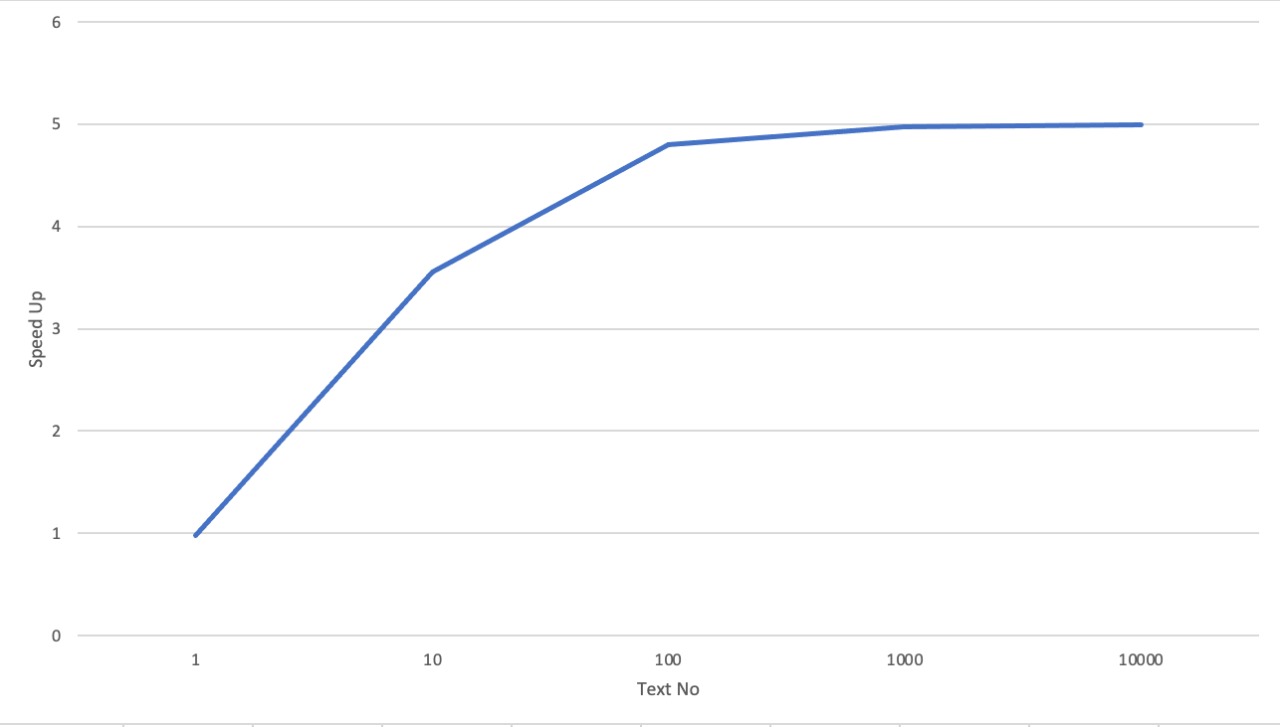
* Speedup

Assuming the period of our clock is: 16000, and the delay of buffers is: 1000. We made the analysis of speedup on multiple texts:

For one text: SINGLE CYCLE: 16000 \*5 =80,000 sec PIPELINE: 16000 \* 5 +(delay of buffers) = 81,000 sec SPEED UP = 0.98For 10 texts: SINGLE CYCLE: 16000 \*5\*10 =800,000 sec PIPELINE: 16000 \* (5 +9) +(delay of buffers) = 225,000 sec SPEED UP = 3.556For 100 texts: SINGLE CYCLE: 16000 \*5\*100 =8,000,000 sec PIPELINE: 16000 \* (5 +99) +(delay of buffers) = 1,665,000sec SPEED UP = 4.804For 1000 texts: SINGLE CYCLE: 16000\*5\*1000 =80,000,000 PIPELINE: 16000 \* (5 +999) + (delay of buffers) = 16,065,000 sec SPEED UP = 4.97For 10000 texts: SINGLE CYCLE: 16000 \*5\*10000 =800,000,000 sec PIPELINE: 16000 \* (5 +9999) +(delay of buffers) = 160,065,000 sec SPEED UP = 4.997

* throughput:

As shown in our calculations, the time for one text in pipeline is worse than time in single cycle due to the extra delay of buffers, however the throughput of pipeline will increase. So let’s consider 100 text in pipeline design, it will take 1,665,000 sec to execute all texts, however if we consider this time for a single cycle, only 20 texts will execute within this period of time.



Speedup/# of text chart

# Conclusion and Future Work

After the countless hours put into the work and checking the speed up and throughput, we finally concluded that a pipeline version of this would be ideal for our encryption algorithm, having a cipher text after each cycle. So this design achieves a trustworthy security and a high level of throughput.

This model can be useful in VoIP system (voice over internet protocol). This system describes the method to place and receive phone calls over the internet, where encryption of both voice and data takes place and where the speed of encryption is very critical.