**บทคัดย่อ**

การฝึกงานในครั้งนี้กระผมได้ทำการฝึกงานในทั้ง 2 สายงาน ทั้ง Analog และ Digital โดยงาน Analog นั้น กระผมได้ทำ 2 Stage Op Amp ส่วนสายงาน Digital กระผมได้เขียน Code VHDL เพื่อสร้างวงจร Encryption แบบ AES128 และ AES128/256 และโครงงานในฉบับนี้มีการแบ่งงานออกเป็น 2 Part อย่างชัดเจน คือ Two Staged Op Amp และ AES128 and AES128/256 Encryption

**Part 1 : Two Staged Op Amp**

**บทนำ**

Op Amp นั้นเป็นอุปกรณ์ที่มีประโยชน์มากในแง่ของความสะดวก ดังนั้นการออกแบบ Op Ampนั้นจึงเป็นที่แพร่หลายเป็นอย่างมาก มีรูปแบบอยู่มากมายและหลากหลายวิธีซึ่งจะมีข้อดีข้อเสียแต่ละอย่างที่ต่างกันไป ดังนั้นเราจึงจำเป็นที่จะมีความรู้พื้นฐานเพื่อใช้ในการ trade off ตัวแปรของ Op Amp เพื่อให้ได้ Spec ตามที่เราพึงปราถนา หรือในบางครั้งหากการปรับค่านั้นเป็นไปได้อย่างมาก เราก็จำเป็นที่จะต้องเปลี่ยนเทคนิกหรือผสมผสานเทคนิก เพื่อให้ง่ายต่อการปรับค่า

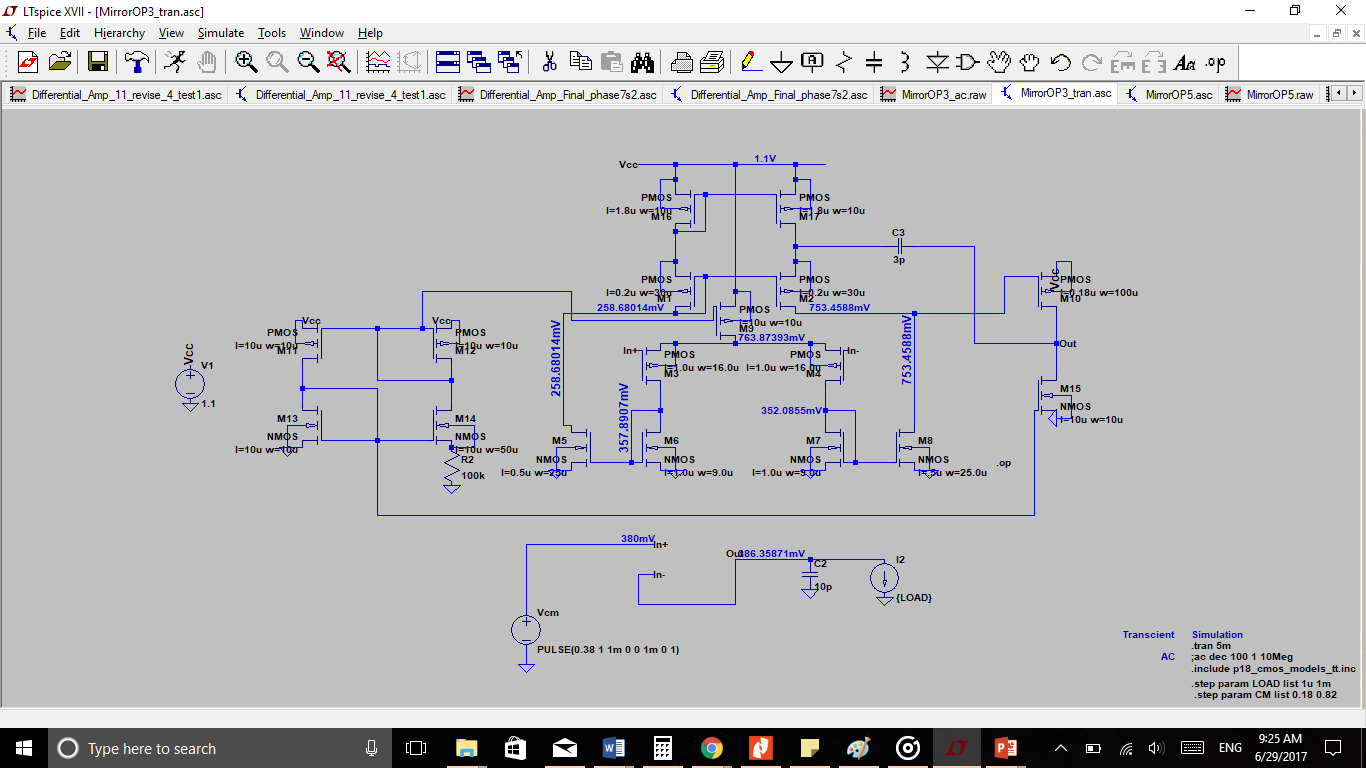


Figure 1

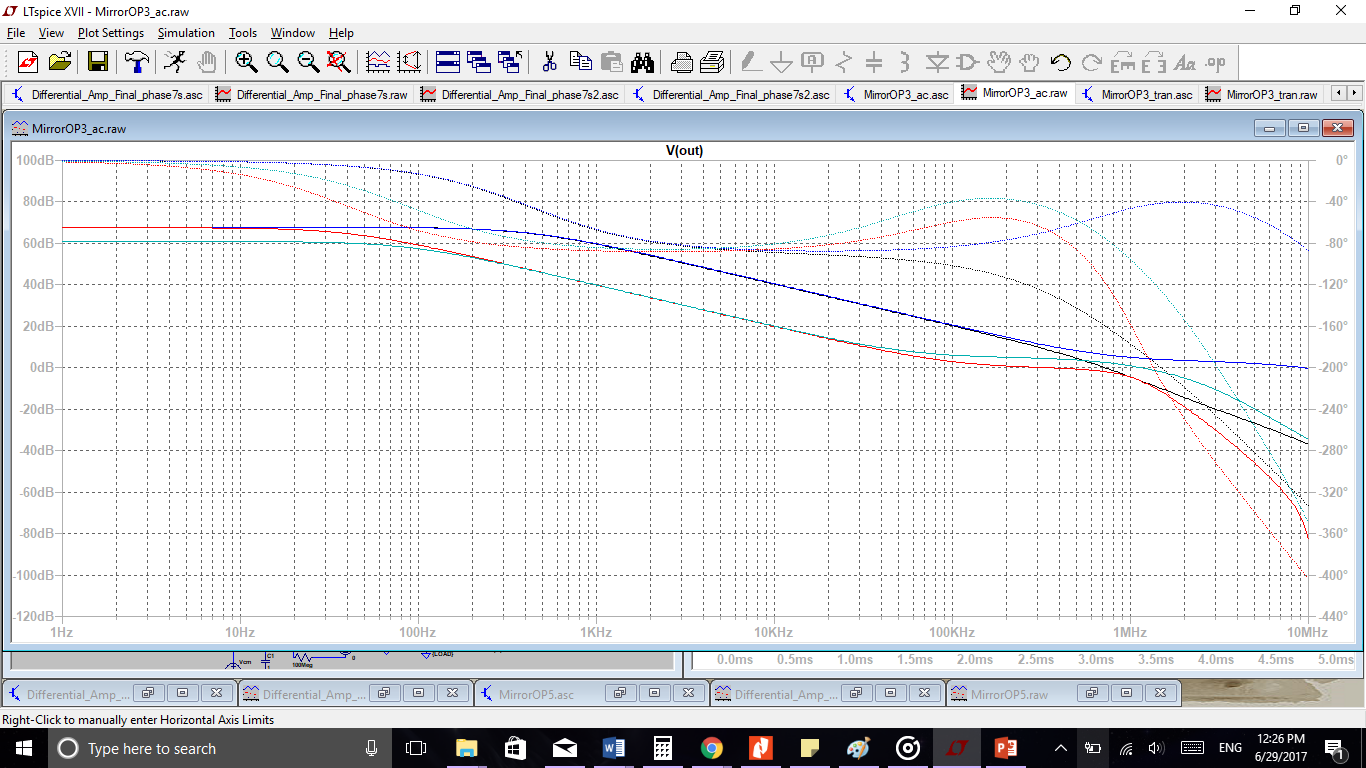


Figure 2

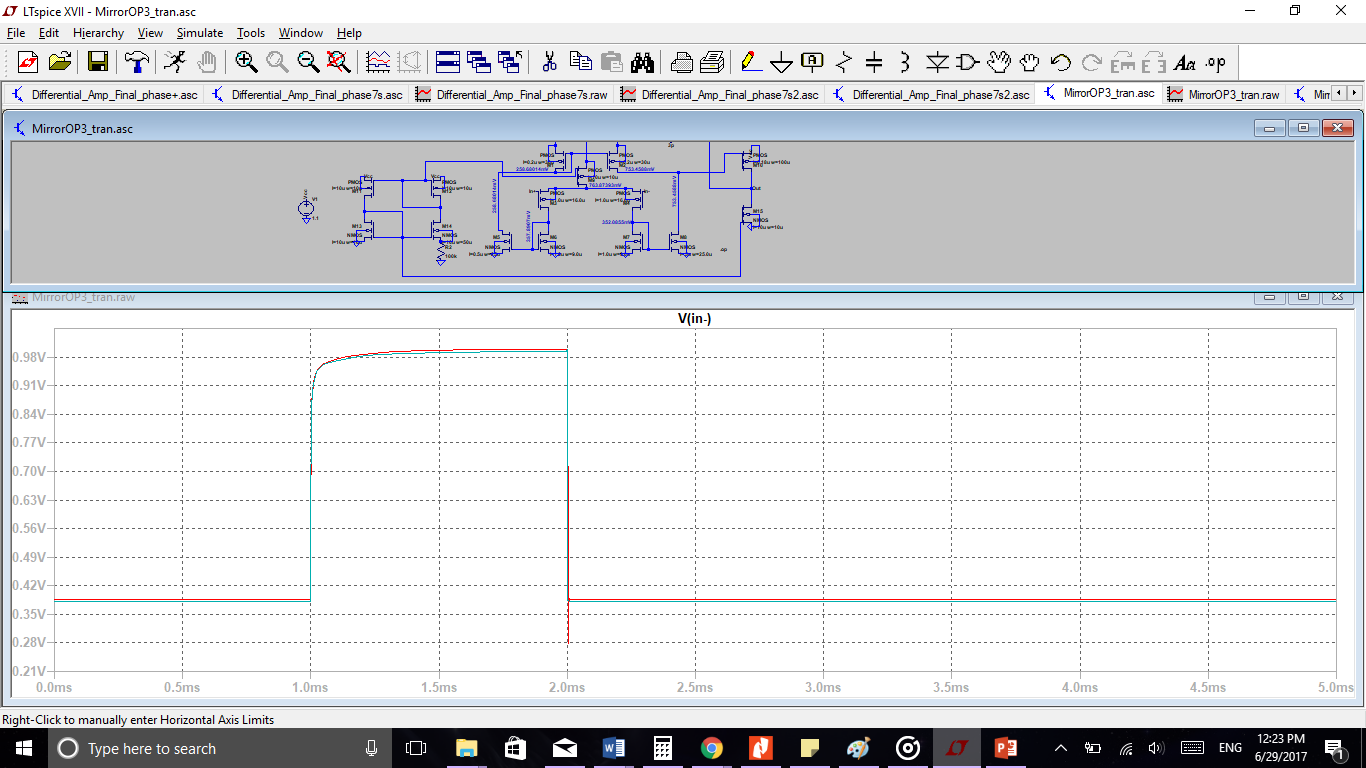


Figure 3

วงจรที่ทำคือการผสมผสานระหว่าง INDIRECT COMPENSATION OPAMP กับ MIRROR OPAMP หรือเรียกได้ว่าเป็นวงจร INDIRECT COMPENSATION MIRROR OPAMP ตาม Figure 1 และลองปรับค่าหลายครั้ง ทำให้ได้ผลลัพธ์เป็นที่ค่อนข้างหน้าพอใจ เพราะว่า Bode Plot ที่ได้เกือบจะผ่าน Spec ที่ได้รับมอบหมาย ในแง่ของ Gain และ Bandwidth ณ ICMR ตาม Figure 2 ที่ต่างไป รวมทั้ง Pulse Response (Figure 3) ด้วย

หลักการที่ถูกใช้ในการปรับนั้นคือ

***สรุป***

*การทำ* Analog *ในครั้งนี้ ทำให้ทราบว่าในวงการ* Analog *ความยากอยู่ที่การปรับ เพราะว่า Analog นั้นมีแง่ของ* Qualitative *ซึ่งต่างจาก* Digital *ที่มีเพียง* Logic *0 หรือ 1 เท่านั้น อย่างไรก็ตามการปรับค่ามันมี Trend ของมันอยู่ หากเราสามารถรู้ว่าการปรับค่าอะไร จะส่งผลต่ออะไรก็ จะทำให้เราสร้างวงจร Analog ได้อย่างไม่ยากลำบาก*

***อ้างอิง***

***SIC\_Lecture.pdf***

**Part 2 : AES-128, AES-128/256 Comparison**

**บทนำ**

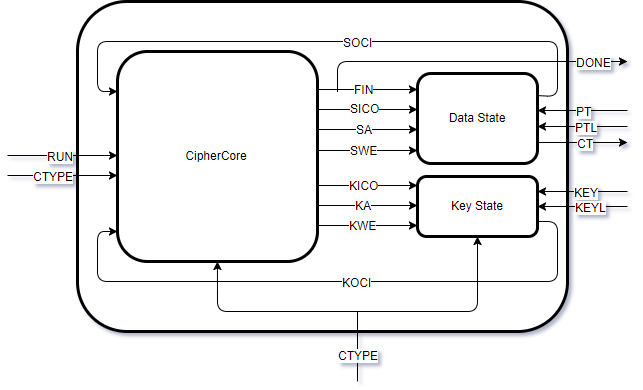
ปัจจุบันนี้มี Super Computer ซึ่งสามารถแกะรหัสอะไรก็ได้ ดังนั้นความปลอดภัยจึงเป็นสิ่งที่ต้องระแวง และระวังเป็นอย่างยิ่ง ดังนั้นรหัสที่มีความซับซ้อนในเชิงโครงสร้างจึงเป็นสิ่งที่จำเป็นอย่างมาก สำหรับ Part นี้จะทำการอธิบายวงจร AES-128 และวงจร AES-128/256 ในรูปแบบ 8 bit แล้วนำมาเปรียบเทียบกันว่า ขนาดและพื้นที่ที่มีเพิ่มเติม มันมากขึ้นเพียงใด

**Overall Structure**

**A screenshot of a cell phone

Description generated with very high confidence**

**Top Module Wiring**

****

**CipherCore**

**Top Module Signal**

CLK : in std\_logic : Input Clock

RST : in std\_logic : Input Reset

RUN : in std\_logic : Perform Encrypt after load

CTYPE : in std\_logic : Cipher Type

SOCI : in std\_logic\_vector (7 downto 0) : Data State Out Cipher In

SICO : out std\_logic\_vector (7 downto 0) : Data State In Cipher Out

SA : out std\_logic\_vector (3 downto 0) : State Address

SWE : out std\_logic : State Write Enable

KOCI : in std\_logic\_vector(7 downto 0) : Key Out Cipher In

KICO : out std\_logic\_vector(7 downto 0) : Key In Cipher Out

KA : out std\_logic\_vector(4 downto 0) : Key Address

KWE : out std\_logic : Key Write Enable

FIN : out std\_logic : Cipher Finished

**ตัวแปรเกี่ยวกับ State**

**CipherState** : enum : เพื่อชี้ว่า FSM นี้กำลังทำงาน Operation ใดอยู่ โดย State ที่เป็นไปได้คือ (IDLE, SUB\_BYTE, SHIFT\_ROW, MIX\_COLUMN, ADD\_ROUND\_KEY)

IDLE : State สำหรับการพักและไม่ทำงาน

SUB\_BYTE : State ที่ทำการแทนค่าข้อมูลแต่ละตัวลงใน S-BOX

SHIFT\_ROW : State ที่ทำการสลับตำแหน่งใน Row

MIX\_COLUMN : State ที่ทำการคูณ Matrix ใน Column

ADD\_ROUND\_KEY : State ที่ทำการ XOR State(Data) ด้วย Key

**KeyState** : enum : เพื่อชี้ว่าใน Sub-FSM นี้กำลังทำงาน Operation ใดอยู่ ซึ่ง Sub-FSM นี้จะทำงานเฉพาะตอนที่อยู่ในสถานะ ADD\_ROUND\_KEY เท่านั้น โดย State ที่เป็นไปได้คือ (IDLE, READ\_LAST, READ\_MIDDLE, CHAIN)

IDLE : State สำหรับการพักและไม่ทำงาน

READ\_LAST : State ที่อ่าน Key จาก Column สุดท้าย

READ\_MIDDLE : State ที่อ่าน Key จาก Column ที่ 3 (เริ่มต้นจาก 0)

CHAIN : State ที่ทำการ XOR และ Save ข้อมูลทั้ง Key และ State

rwState : enum : เพื่อชี้ว่า FSM นี้อยู่ในกระบวนการไหน READ หรือ WRITE แต่ตัวแปรนี้จะใช้ในเฉพาะบาง CipherState

**ตัวแปรเกี่ยวกับ Address ของ State และ Key**

StateAddress : unsigned(3) : สำหรับเก็บตัวแปร Address ของ State(Text)

KeyAddress : unsigned(3) : สำหรับเก็บตัวแปร Address ของ Key

StateAddressN : std\_logic\_vector(3 downto 0) : เป็น Combinational Logic ที่มาจาก RegRow และ RegCol

**ตัวแปร Register ที่สำหรับการเก็บข้อมูลชั่วคราว**

RegZ : std\_logic : ใช้ในการป้องกันการย้าย State เมื่อ StateAddress เป็น 0 ในครั้งแรก

RegA : std\_logic\_vector(7 downto 0)

RegB : std\_logic\_vector(7 downto 0)

RegC : std\_logic\_vector(7 downto 0)

RegD : std\_logic\_vector(7 downto 0)

**ตัวแปรพิเศษสำหรับการทำ Mix Column**

SigAx : std\_logic\_vector(7 downto 0);

SigBx : std\_logic\_vector(7 downto 0);

SigCx : std\_logic\_vector(7 downto 0);

SigDx : std\_logic\_vector(7 downto 0);

SigSOCIx : std\_logic\_vector(7 downto 0);

**ตัวแปรสำหรับนับจำนวนรอบ**

Round : unsigned(3 downto 0)

LRound : unsigned(3 downto 0) : ย่อมาจาก Last Round เป็นสัญญาณที่ขึ้นอยู่กับ CT(CipherType)

**Substitution Box**

SBox : สำหรับใช้ในการแทนค่าตอนทำ Operation Sub Byte และ Add Round Key

**Round Constant**

RCon : เป็น List ของ Round Constant ที่ใช้ในการ XOR ในแต่ละรอบตอนช่วง Add Round Key

**Main State Diagram**

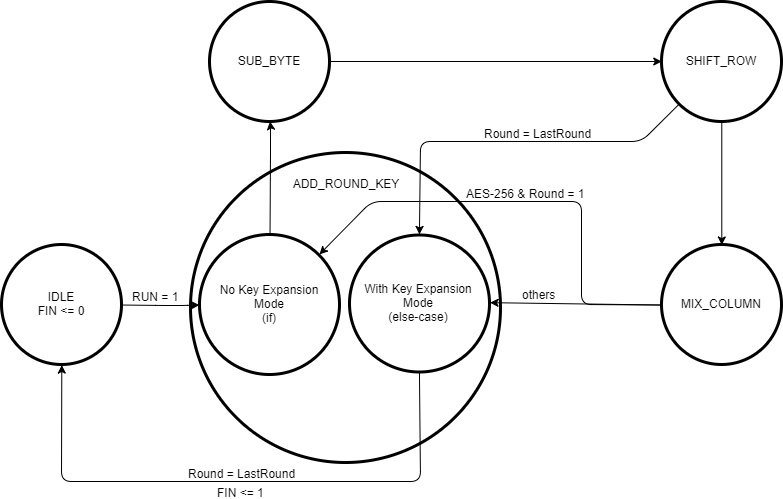
****

Figure 1

ภาพดังรูป Figure 1 เป็น State Diagram ที่แสดงหลักการทำงาน และจะเริ่มทำงานเมื่อมีการป้อนสั่งคำสั่ง RUN หลังจากนั้น FSM นี้จะวนจาก Add Round Key แล้วต่อด้วย Sub Byte, Shift Row และ Mix Column สุดท้ายกลับไปที่ Add Round Key อีกครั้ง เมื่อทำถึงรอบสุดท้าย FSM นี้จะ Skip คำสั่ง Mix Column และไปทำ Add Round Key หลังจากนั้นก็ส่งสัญญาณ FIN(Finish) เพื่อบอกให้ Data State ทำการป้อนข้อมูล Cipher Text ออกจากตัวมาเป็นอันสิ้นสุดกระบวนการเข้ารหัส AES

**Add Round Key(No Key Expansion) State Diagram**

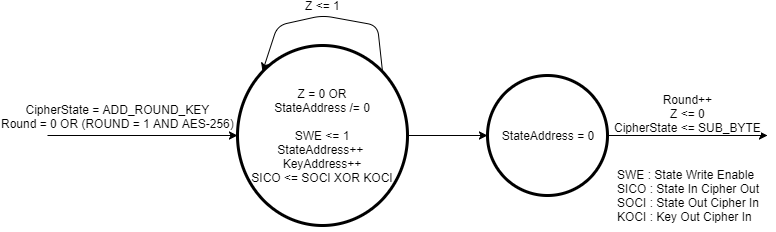


Figure 2

Diagram(Figure 1) นี้แสดงให้เป็นถึง การทำงานของ Add Round Key ซึ่ง Add Round Key ในรอบนี้จะไม่ทำ Key Expansion และจะมีเพียงแค่การนำ Key มา XOR กับ Data State เท่านั้น

**Add Round Key (With Key Expansion) State Diagram**

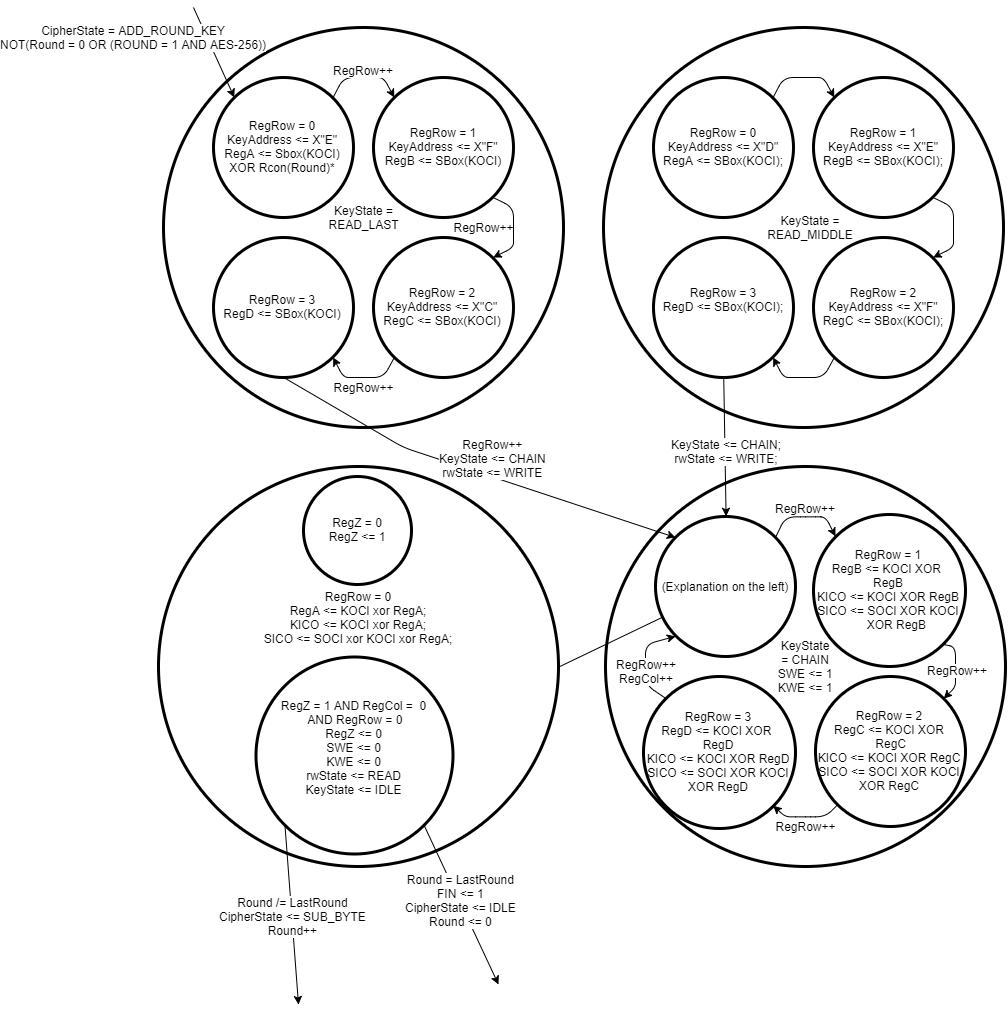


Figure 3

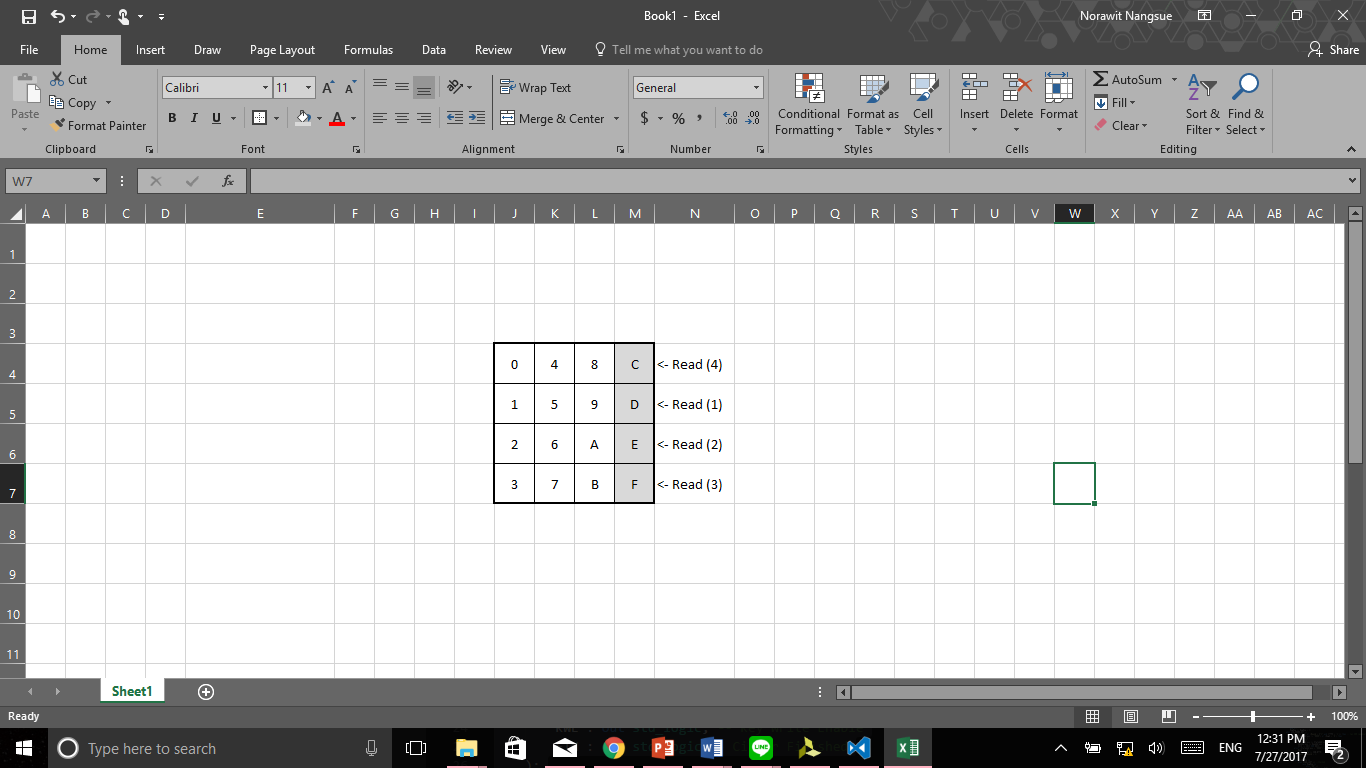


Figure 4

Diagram ตาม Figure 3 แสดงให้เป็นถึง การทำงานของ Add Round Key โดยที่จะเป็นกรณีทั่วไปโดยจะมีการทำ Key Expansion โดย เริ่มแรกจาก READ LAST คือการอ่านค่าหลักสุดท้ายของ KeyState โดยเลือก Address D, E, F, C (Figure 4) พร้อมแทนค่าลงใน S-BOX และ XOR ด้วย Round Constant แล้วเก็บเข้า RegA, RegB, RegC, RegD ตามลำดับ ซึ่งก็เป็นการสิ้นสุดการทำงานของ READ LAST และพร้อมที่จะทำงานในส่วนของ Key Expansion(CHAIN)

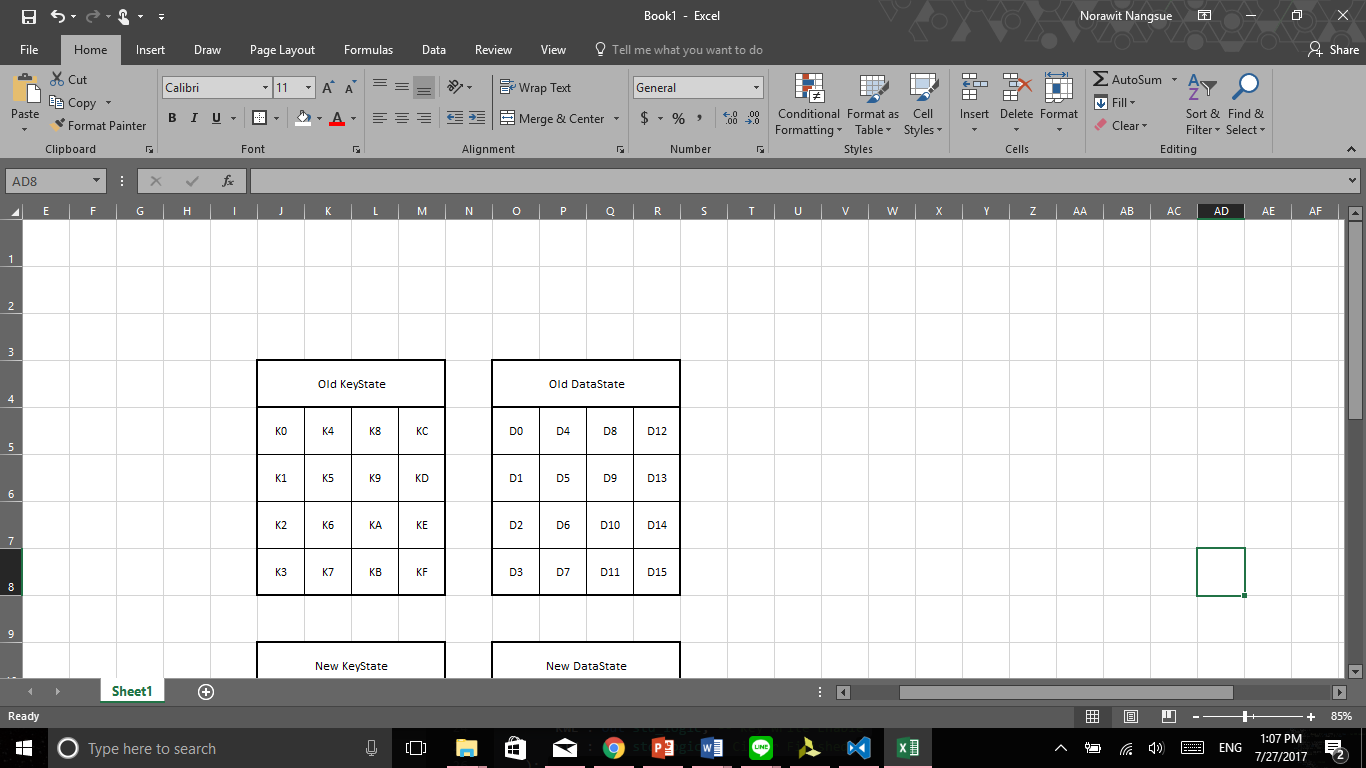


Figure 5

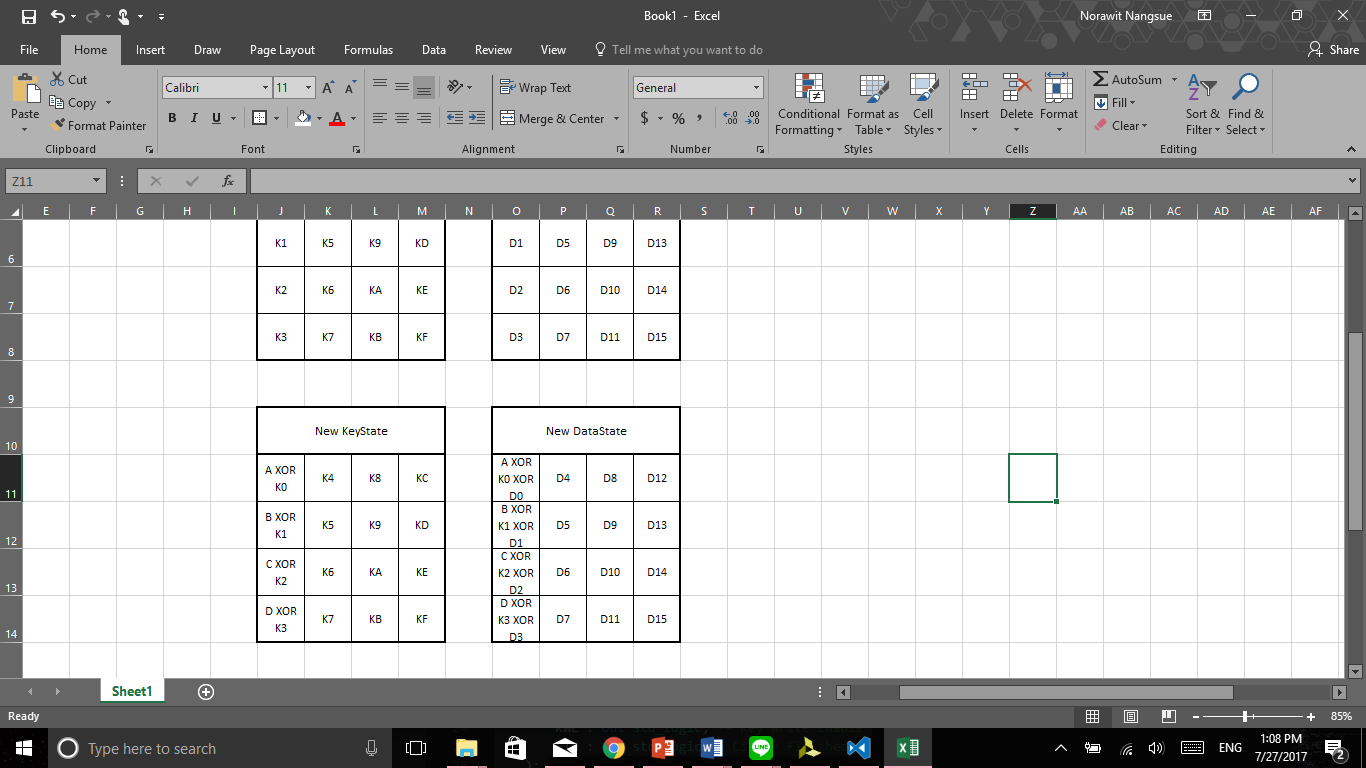


Figure 6

หลังจากที่ได้อ่านค่าหลักสุดท้ายและผ่านกระบวนการต่างๆ ก็มาถึงกระบวนการ Chain คือการนำค่าใน Reg ต่างๆมา XOR กับ KeyState เพื่อสร้าง Key ใหม่สำหรับการ ทำ Key Expansion ครั้งถัดไป และจะเห็นได้ว่า การทำ Key Expansion จะทำควบคู่กับการ Add Round Key หรือก็คือการนำ Key ที่ได้มานั้นมา XOR กับ Data State ไปด้วย(Figure 5, Figure 6)

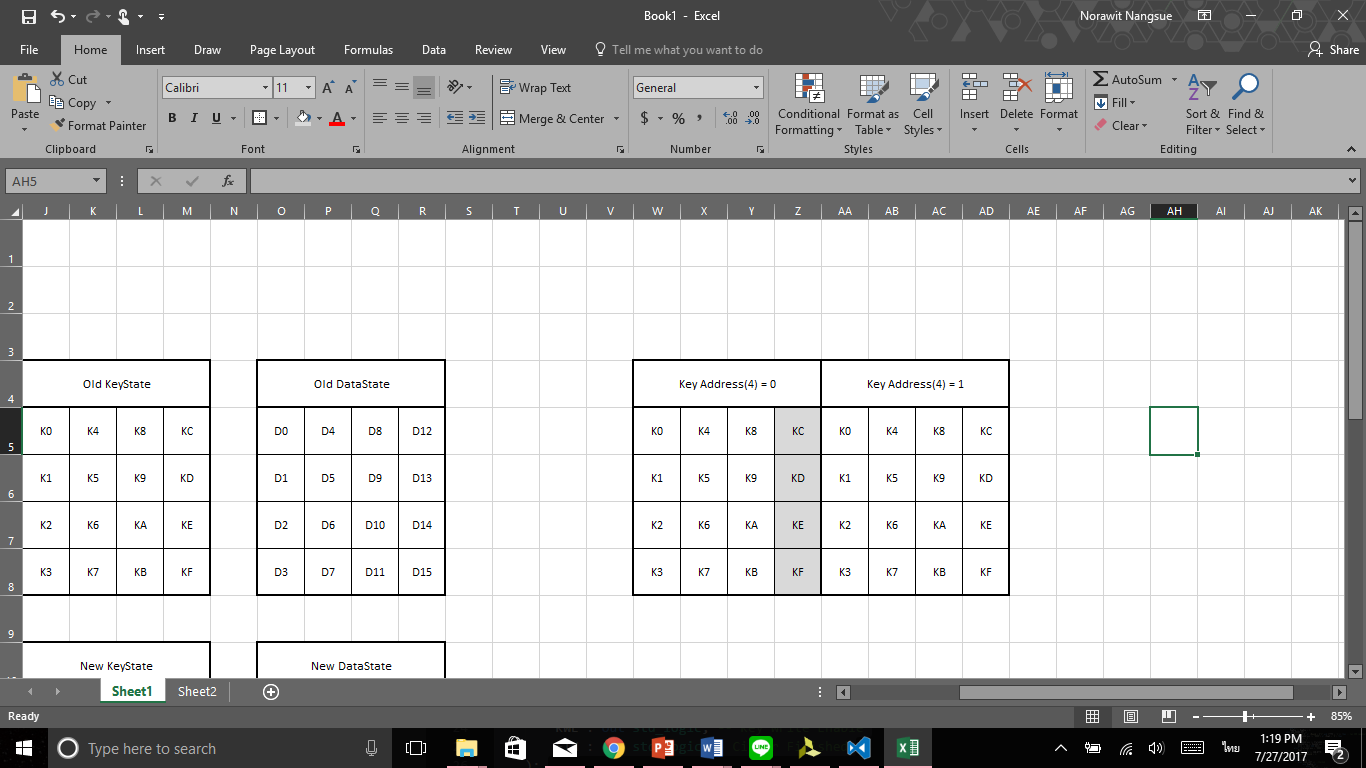


Figure 7

ในกรณี AES-256 Key มีหลักว่าจะต้องนำไปเข้า SBOX ก่อนที่ที่จะเข้า Key Expansion ต่อไป ซึ่งจากการสังเกตุนั้น รอบที่เป็นรอบคู่ (Round = 0,2,4,6) จะใช้ Key Address ชุดซ้าย ส่วนรอบคี่จะใช้ Key Address ชุดขวา จึงจำเป็นที่จะต้องมีกระบวนการ Read Middle เพื่อทำการแทนค่าลง S-BOX และนำใส่เข้า RegA, RegB, RegC, RegD หลังจากนั้นก็เข้ากระบวนการ CHAIN ต่อไป

**Sub Byte State Diagram**

****

Figure 8

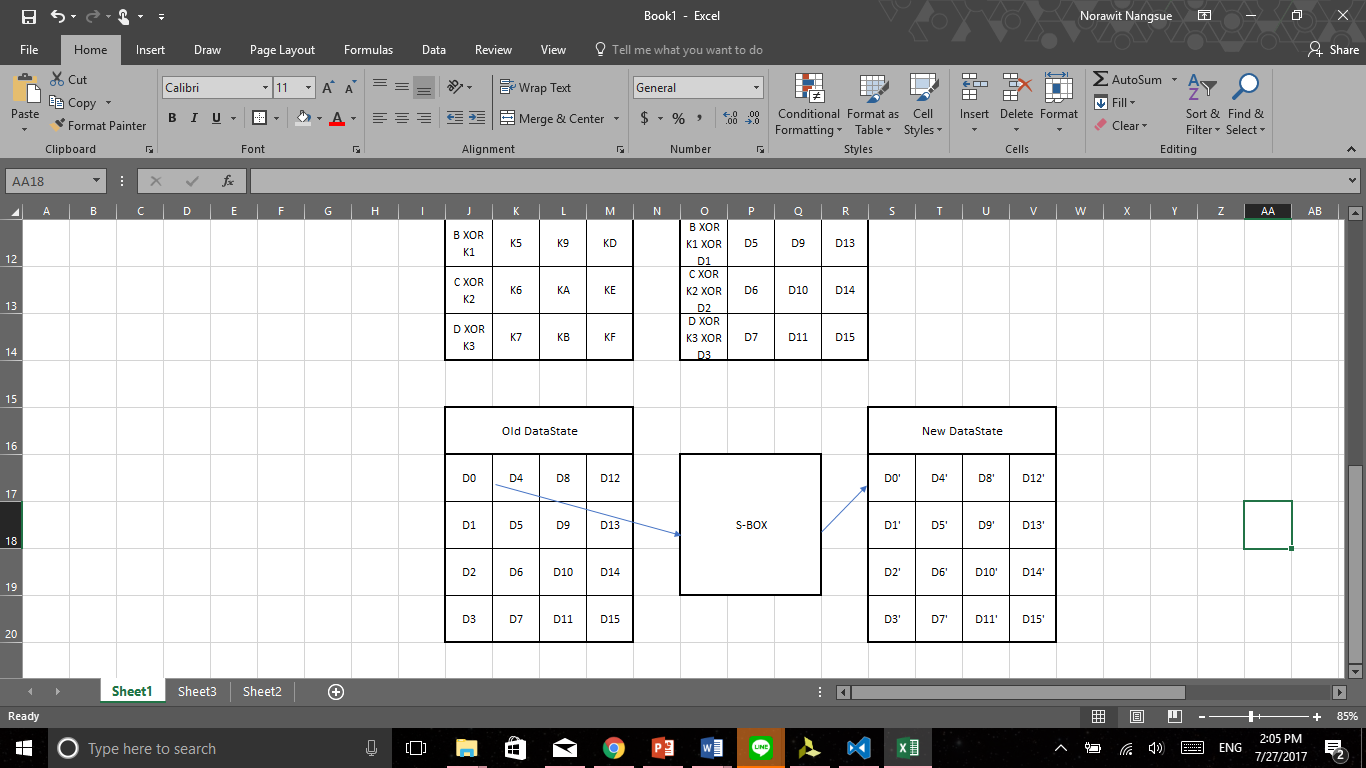


Figure 9

State Sub Byte คือการแทนค่า Data แต่ละตัวลงใน S-Box ดัง Figure 9 แล้วนำบันทึกใส่ Data State อีกที การกระทำนี้จะใช้เพียง 1 Clock ต่อ 1 Bit ตามที่ได้อธิบายโครงสร้างของ Data State การบันทึกจะบันทึกใส่ที่ Address – 1 ทำให้การ Read และการ Write พร้อมกันนั้นเป็นไปได้ FSM นี้จะเปลี่ยน State เมื่อ State นั้นได้แทนค่าครงทุกตัวแล้ว หรือเรียกได้ว่า State Address นั้นกลับไปที่จุดเริ่มตั้น ดังนั้นมันจึงมีความจำเป็นที่จะต้องใช้ Register Z เพื่อให้ไม่มีการย้าย State เมื่อ State Address = 0 ในครั้งแรก

**Shift Row State Diagram**

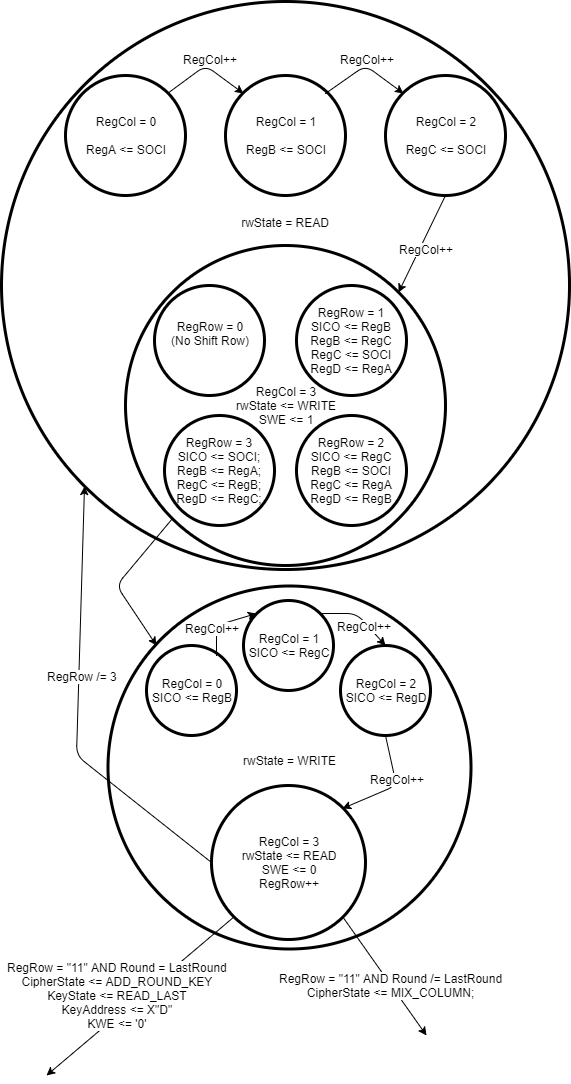


Figure 10

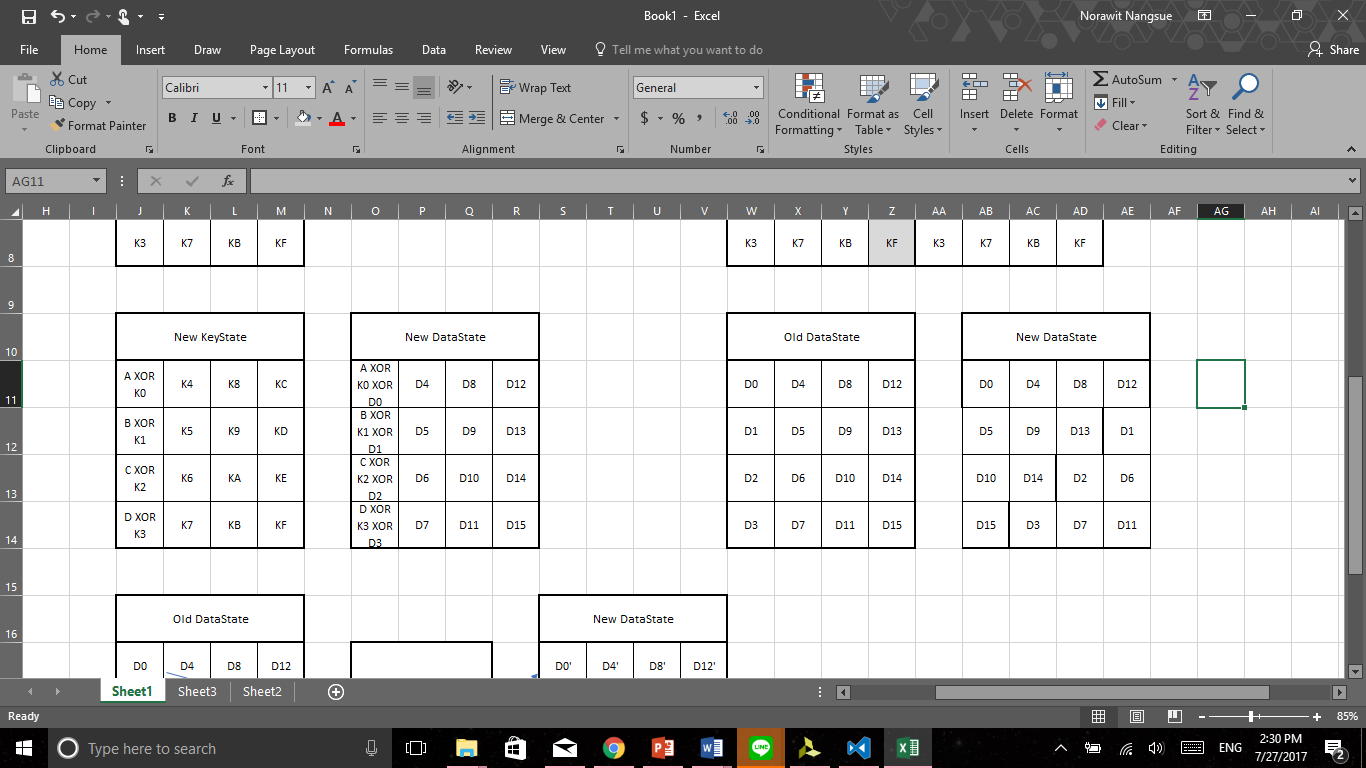


Figure 11

State Shift Row นั้นเป็นการสลับตำแหน่งในแถวเดียวกันตาม Figure 11 ซึ่งแต่ละแถวจะมีลักษณะที่สลับในรูปแบบที่ต่างกัน เนื่องจากมีการ Operate แบบ 8 bit ดังนั้นขั้นตอนที่ FSM ดัง Figure 10 นี้ทำคือ เปลี่ยน Column (การเขียนหรืออ่าน Address ใน State นี้จะเกิดจาก Combinational Logic ของ Row และ Column) ในขั้นเริ่มต้นจะทำการอ่านค่าทีละ Address ใส่ใน Register ต่างๆ หลังจากนั้นก็ทำการสลับตำแหน่งใน Register พร้อมทำการส่งค่าออก ไปเรื่อยๆ เมื่อส่งเรียบร้อยก็จะเพิ่มค่า Row ขึ้นอีก 1 แล้วกลับไปอ่านแต่ละ Column อีกที และเมื่อ Row มีค่าเป็น 3 แล้วก็ทำการเปลี่ยน Operation ซึ่ง Operation ที่จะทำต่อนั้นขึ้นอยู่กับว่า รอบของ FSM หลักนั้นถึงรอบสุดท้ายแล้วหรือไม่ หากใช่ก็ต้องไปทำ Add Round Key ต่อไป ในทางกลับกัน หากไม่ใช่ก็ FSM ก็จะไปทำในส่วนของ Mix Column

**Mix Column State Diagram**

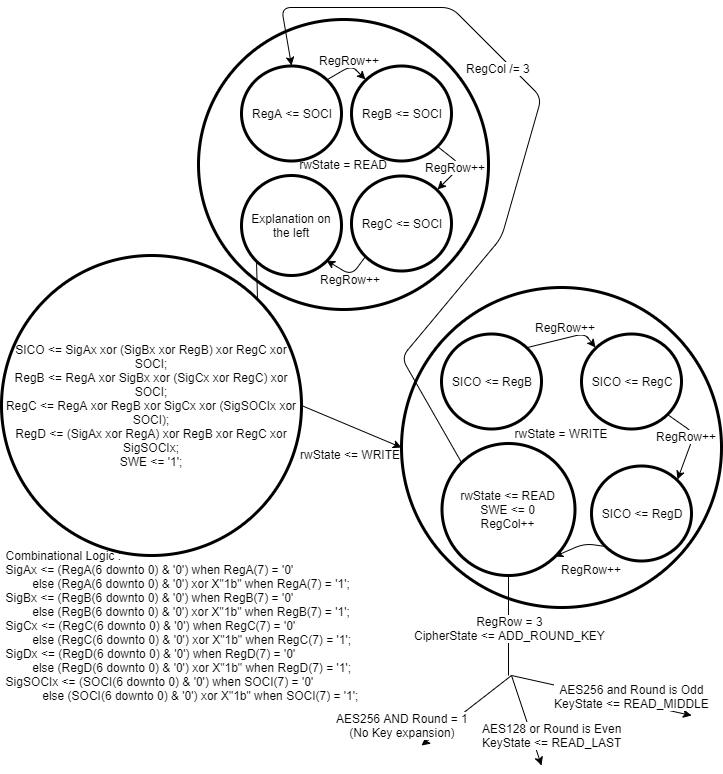
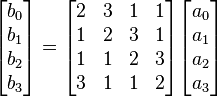
****

Figure 12



Mix Column นั้นเป็นการผสมข้อมูลทาง Column โดยตามหลักจะเป็นคูณ Matrix 4x4 ตาม สมการด้านบน แต่ใน Code VHDL จะเป็นการทำ Combination Logic และผลลัพธ์ที่ได้จะเป็นผลของการ Shift และพิจารณาในรูปแบบของ GF(8) ซึ่งมีหลักการดังนี้ ยกตัวอย่างเช่นการหาผลลัพธ์ของ จะได้ว่า

เป็นการคูณ 2 ทาง GF(8) ถ้าหากว่า

MSB ของ จะมีค่าเป็น ที่ Left Shift ไป 1 ครั้ง เช่น

ตอนแรก ตอนหลัง

ในทางกลับกัน MSB ของ จะต้องทำการ XOR กับ หลังจากที่ Left Shift ไปแล้ว 1 ครั้ง เช่น

ตอนแรก หลัง Left Shift ได้

และ จะได้

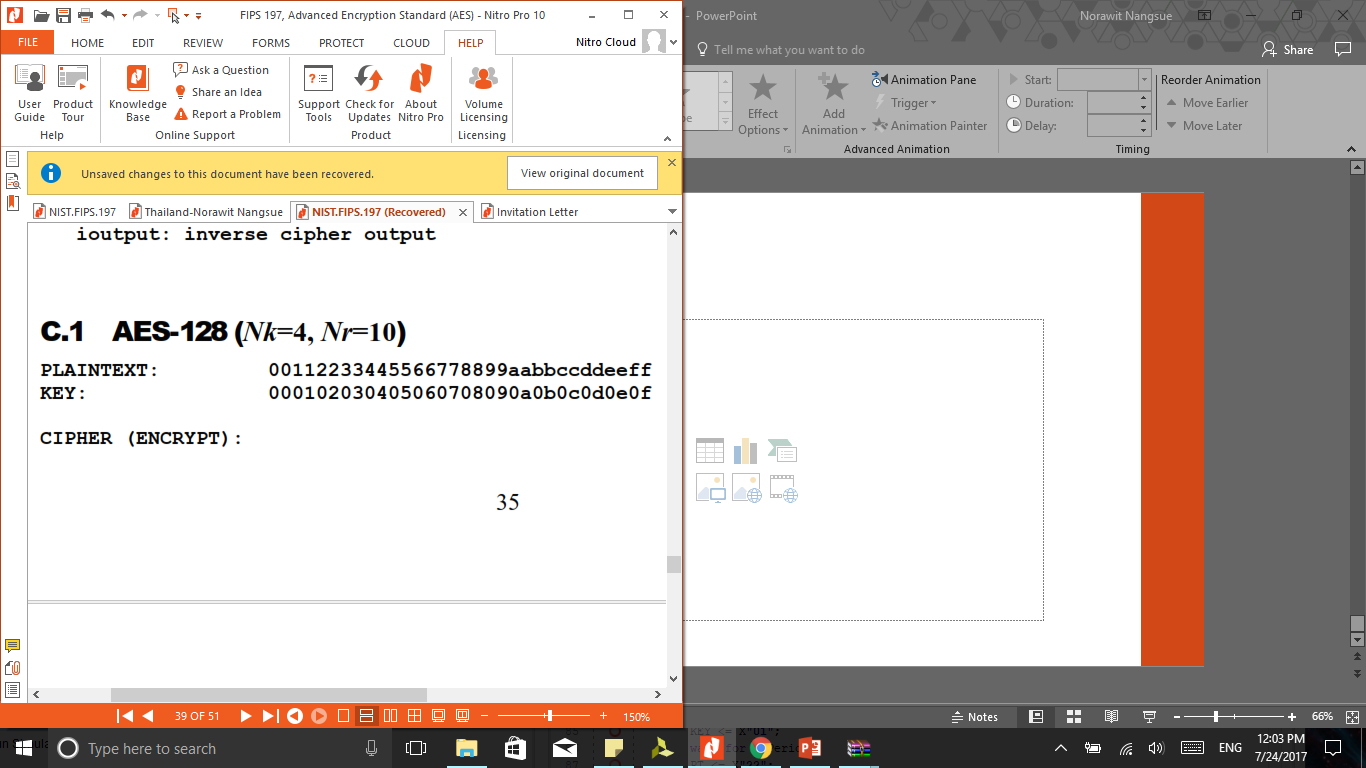
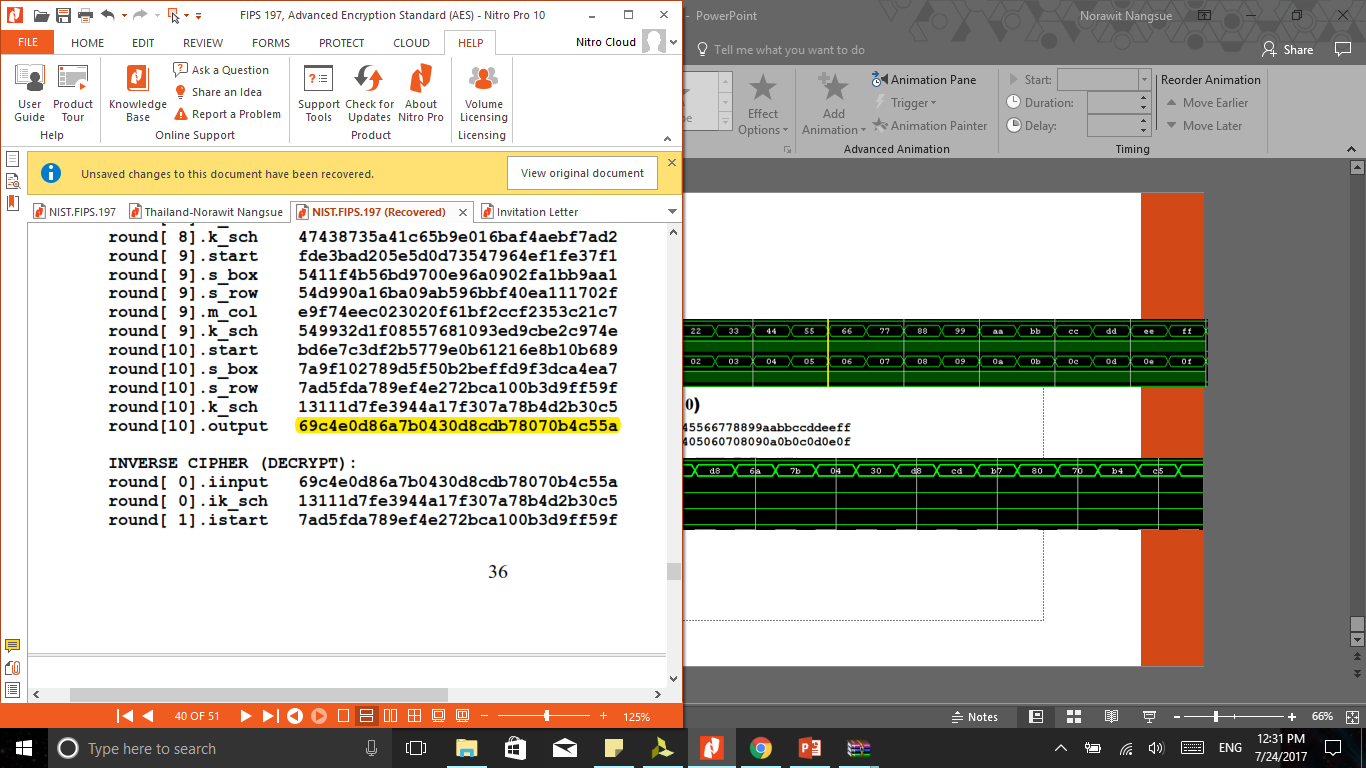
เป็นการคูณ 3 ทาง GF(8) วิธีการคูณจะเป็นดังนี้

จะเห็นได้ว่ามันคือการทำ และ

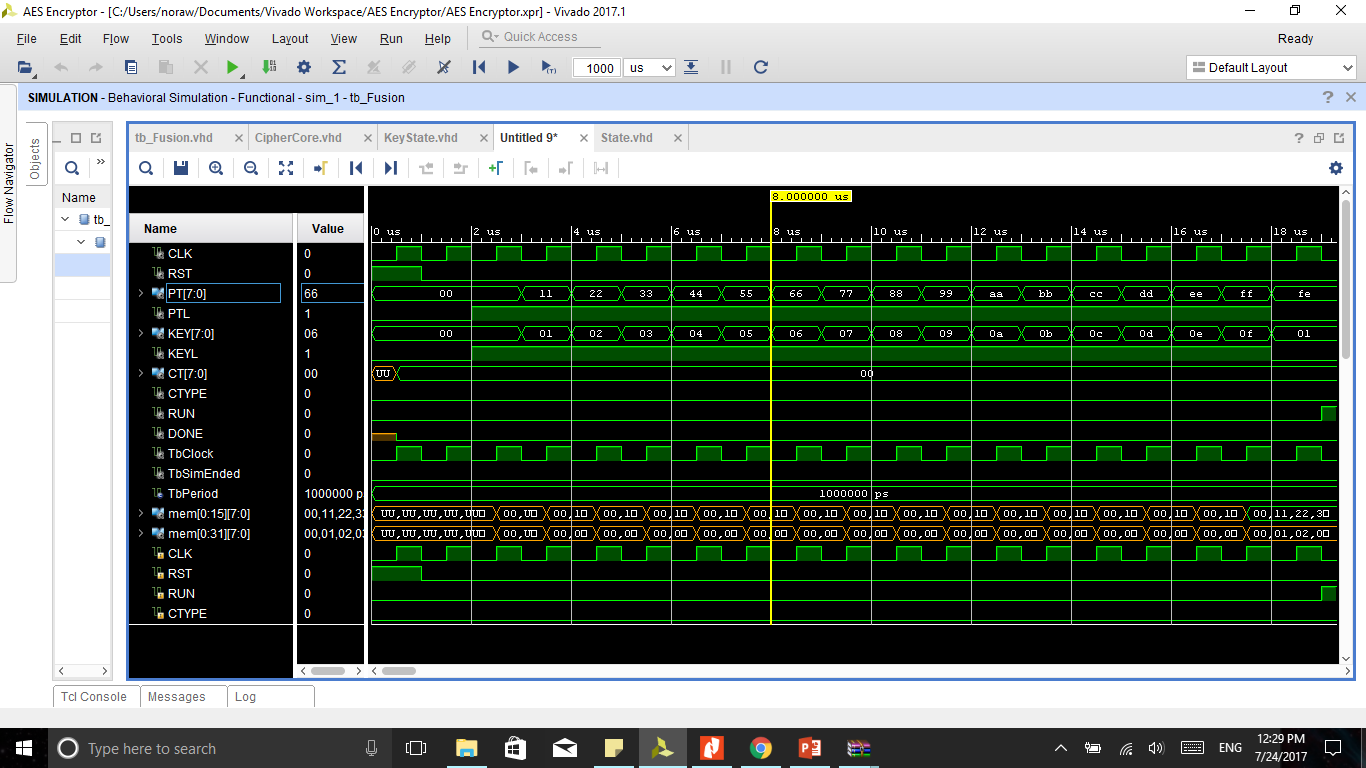
ในบรรทัด SigAx <= (RegA(6 downto 0) & '0') when RegA(7) = '0' else (RegA(6 downto 0) & '0') xor X"1b" when RegA(7) = '1'; เป็น Combinational Logic ที่ใช้ในการอธิบายการกระทำเช่นนี้

**Simulation AES-128**

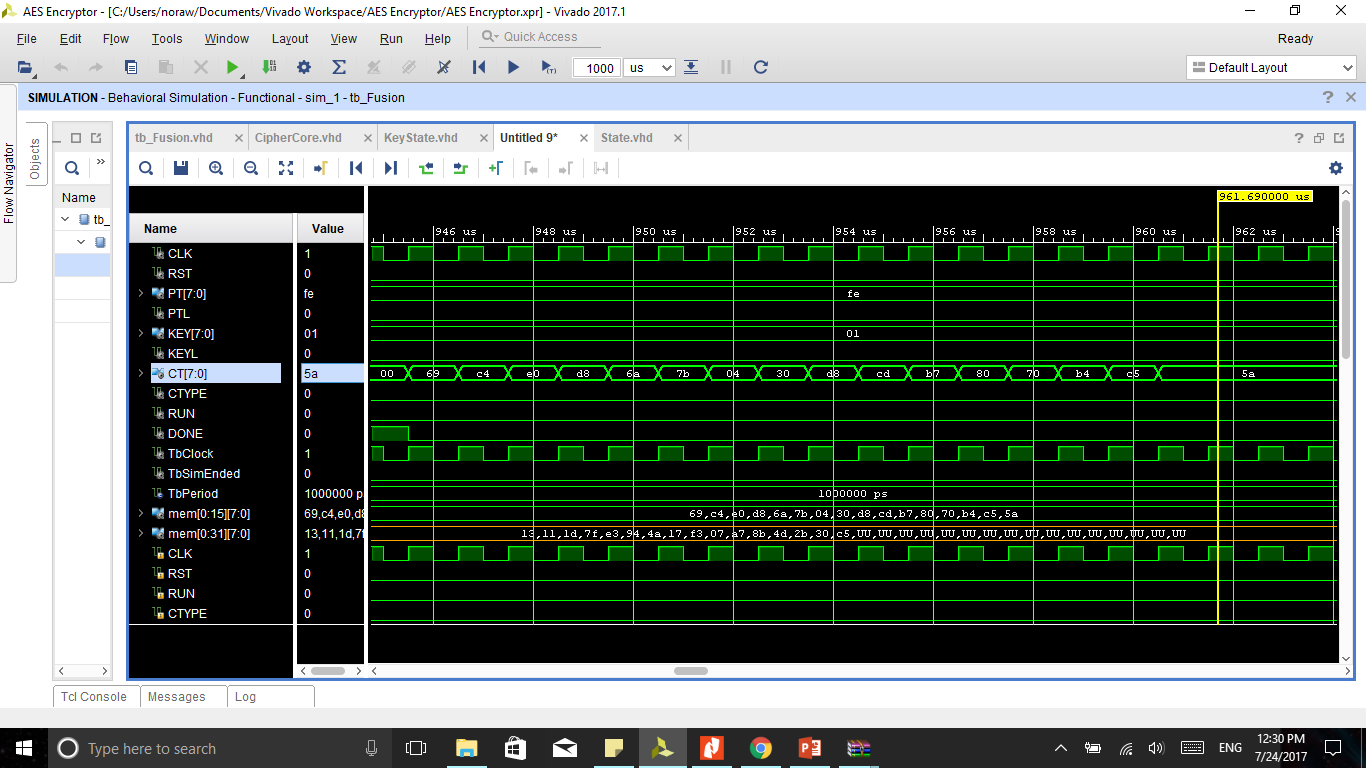
**From Reference (Plain Text, Key)**

****

**Input (Plain Text, Key)**

****

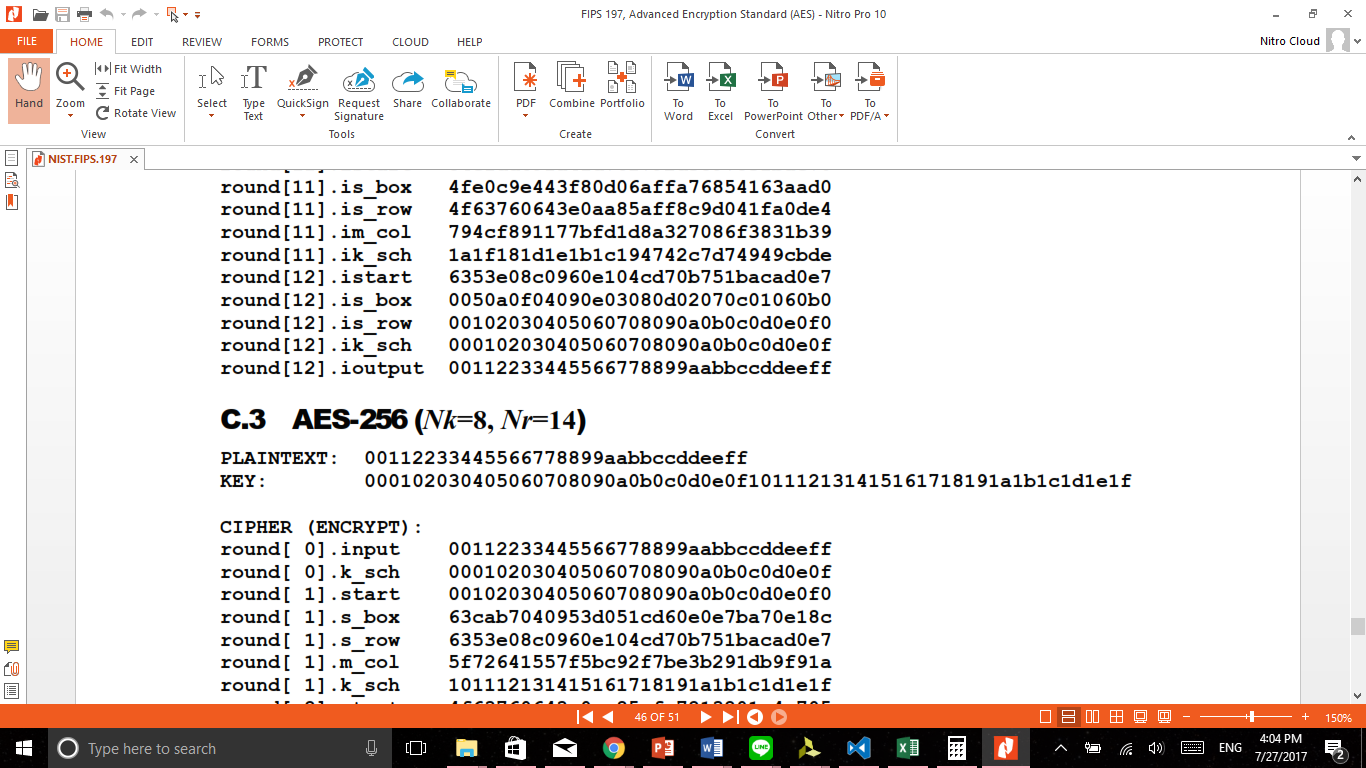
**Output**

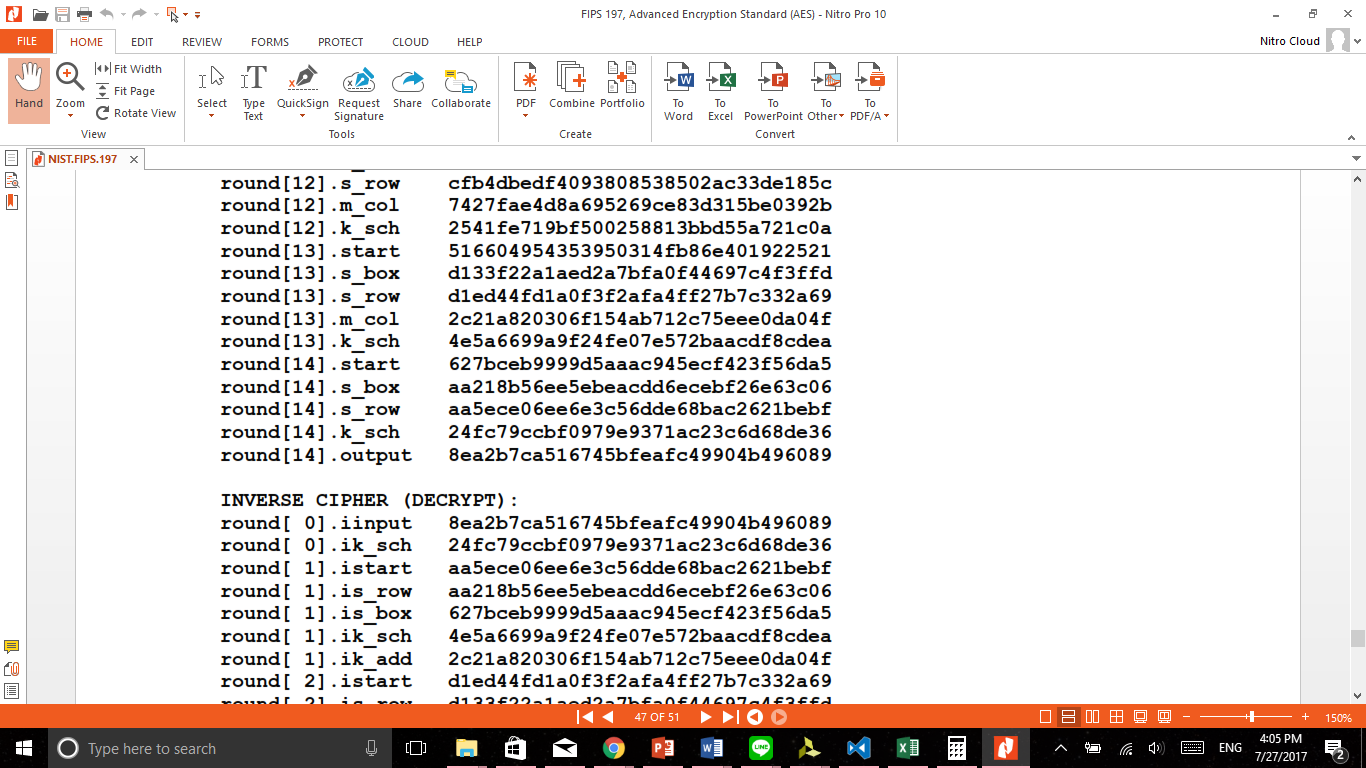
****

**ซึ่งได้ผลลัพธ์ที่ตรงกับความคาดหวัง**

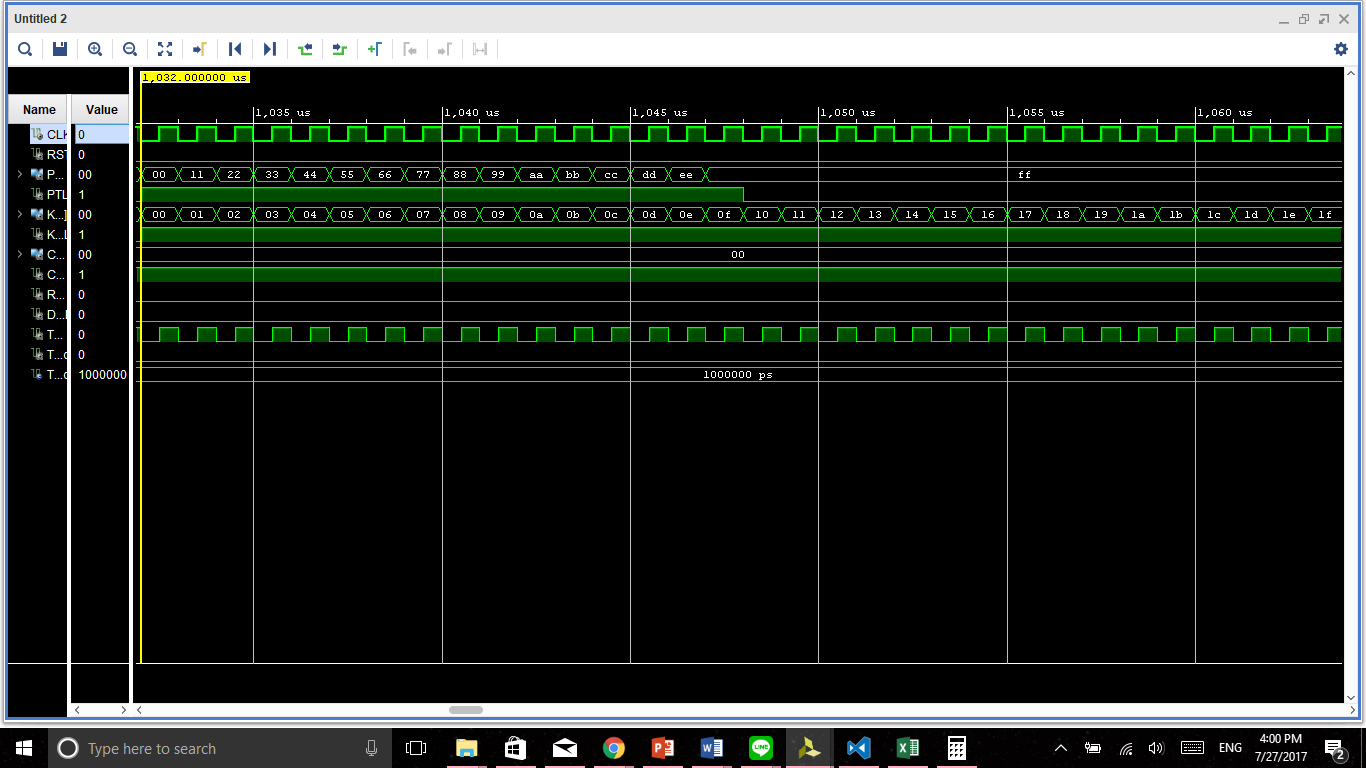
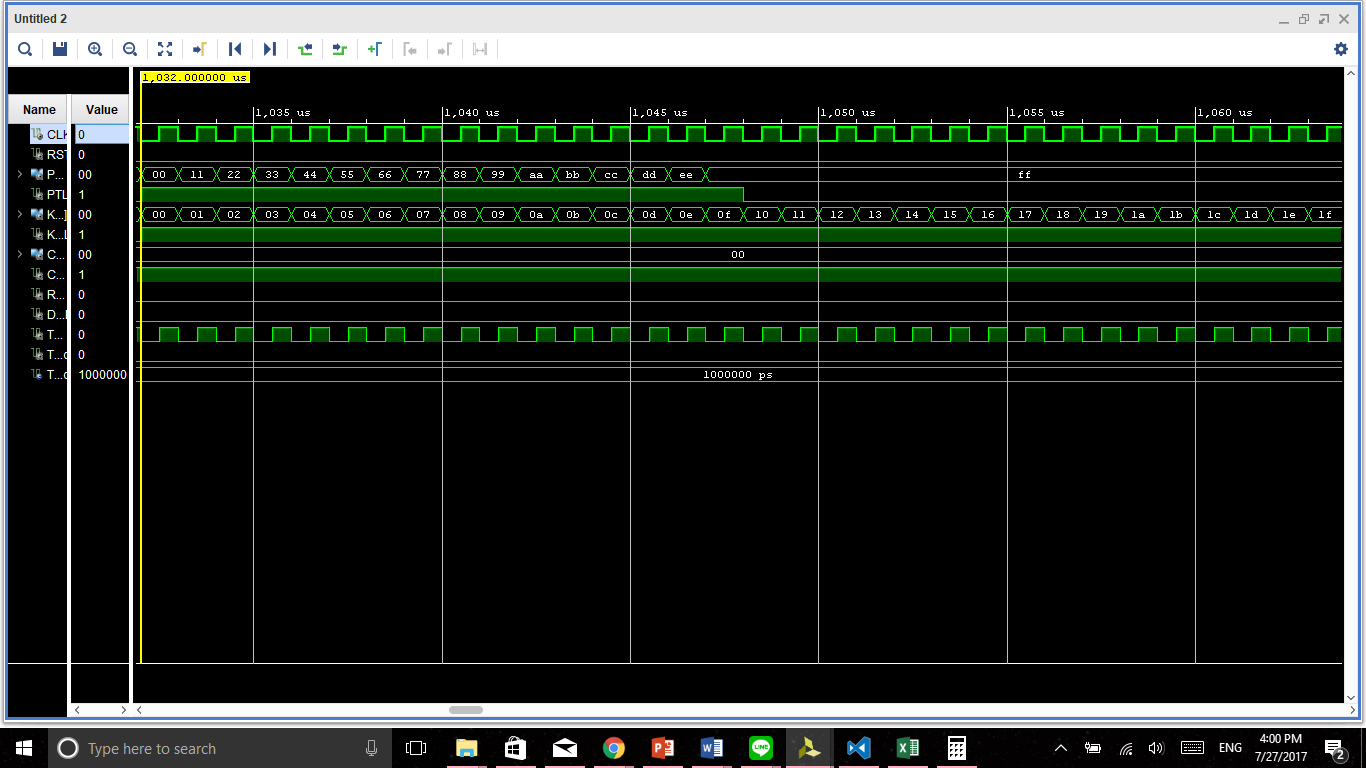
**Simulation AES-256**

**From Reference (Plain Text, Key)**

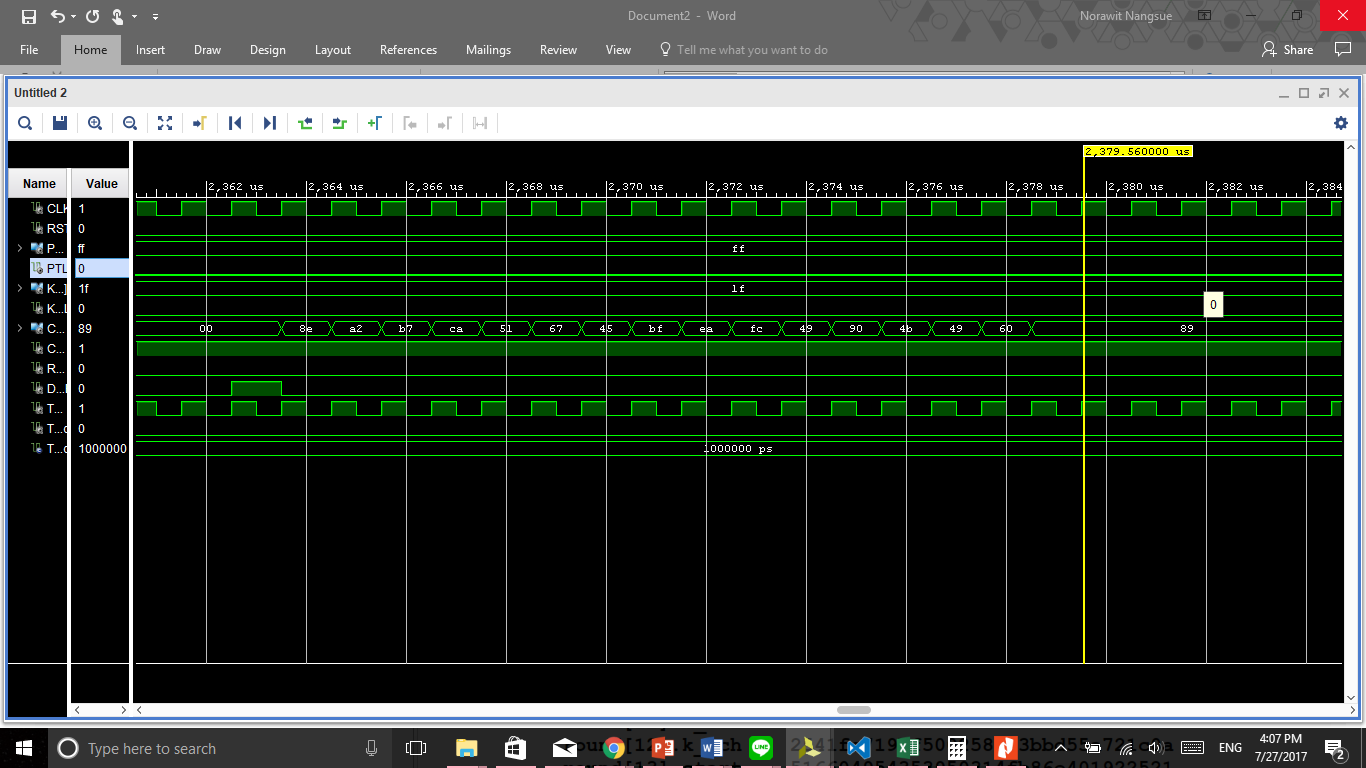




**Input (Plain Text, Key)**



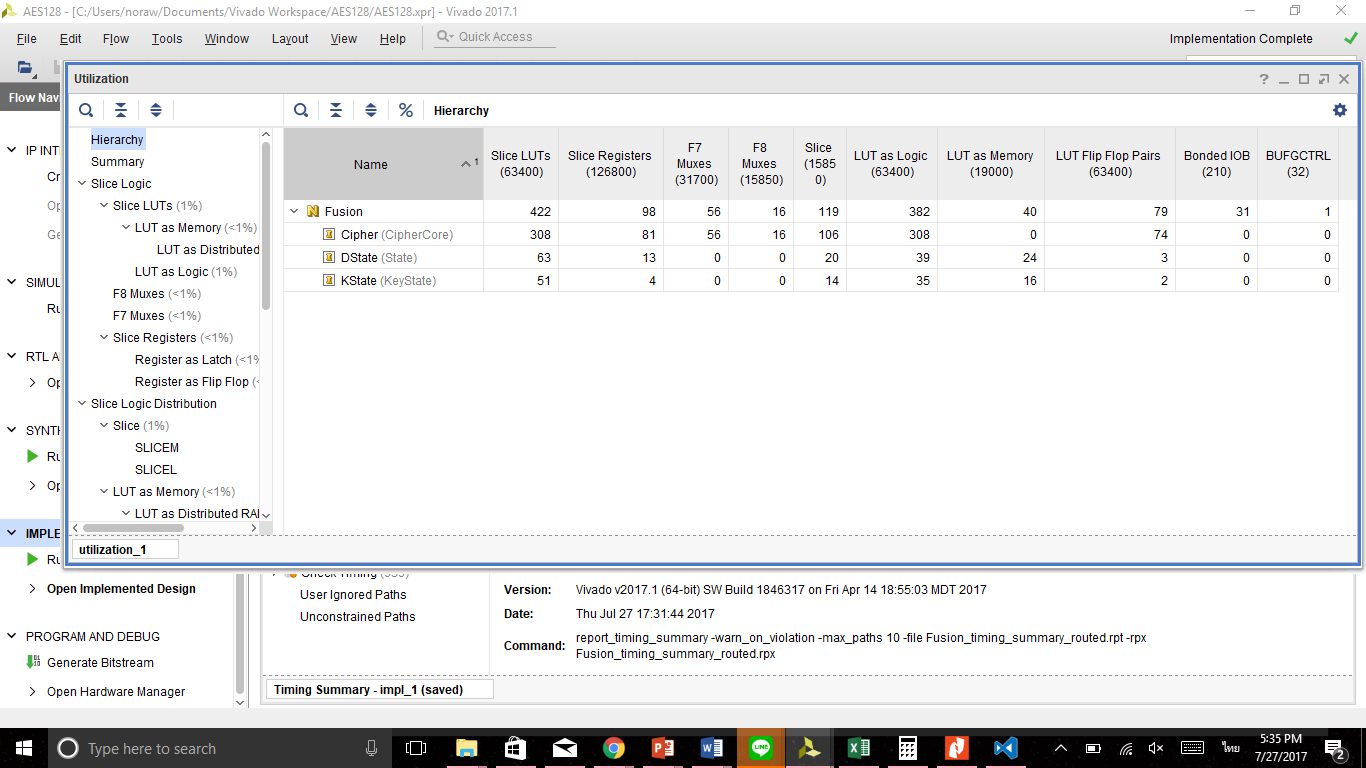
**Output**



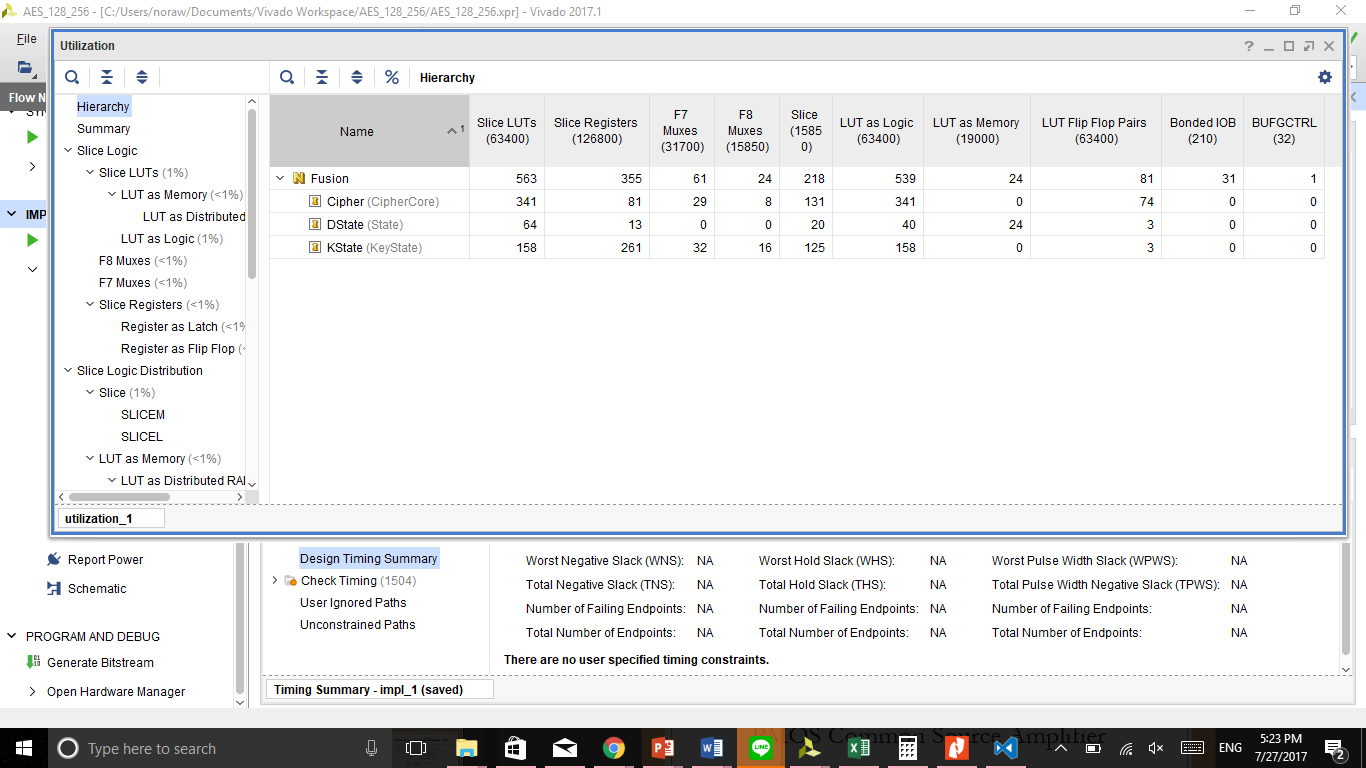
**ซึ่งได้ผลลัพธ์ที่ตรงกับความคาดหวัง**

**AES-128 vs AES-128/256 Comparison**

**AES-128**



**AES-128/256**



จะเห็นได้ว่าสิ่งที่เพิ่มเติมมาอย่างชัดเจนคือ KeyState นั้นมี slice เพิ่มจำนวนมาเป็น 3 เท่า

**สรุป**

วงจร AES Encrypt นั้นเป็นวงจรที่ไม่ซับซ้อน มีหลายขั้นตอน ในการออกแบบนั้นการออกแบบนั้นวิธีคิดของ State Diagram นั้นจะมีความคล้ายคลึงกับการ Concept ของการเข้ารหัสเป็นอย่างมาก เพียงแต่เพิ่มความเป็น Digital เข้าไปเท่านั้น และในความเป็นจริงแล้ววงจร Digital ทั่วไปสามารถอนุมานเป็นรูปแบบนี้ได้ทั้งนั้น

เอกสารอ้างอิง

<https://crypto.stackexchange.com/questions/2402/how-to-solve-mixcolumns>

<http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html>

**NIST.FIPS.197 -** nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf