הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

בי"ס להנדסת חשמל

פרויקט מס': 22-1-1-2494

דוח מסכם

שם הפרויקט: OCB

מבצעים:

שם: באסל מנצור ת.ז.: 208204859

שם: עדן ח'אלד ת.ז.: 206525396

מנחים:

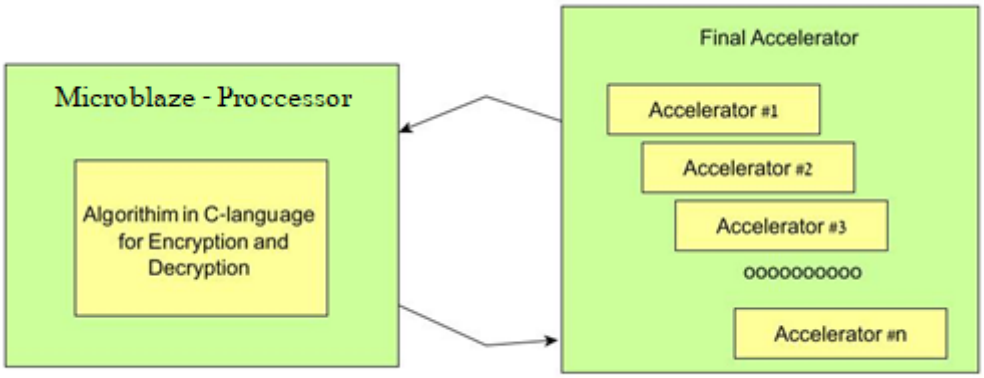
מר. אורן גנון

מקום ביצוע הפרויקט: אוניברסיטת תל אביב.

תוכן עניינים

תקציר:

פרויקט זה הינו בתחום החומרה למערכות משובצות מחשב, בו נממש מצפין חומרה על מעבד MicroBlaze מתקדם של חברת Xilinx, כפי שמתואר בדיאגרמה הבאה :



איור 1 : דיאגרמת בלוקים

כך שהמאיץ בנוי בשפת Verilog שבעזרתה נוכל לבנות רכיבים חדשים שיכולים לבצע כמה פעולות באותו מחזור שעון למשל רכיב AddShift שיכול לבצע גם הזזה וגם הוספה במחזור שעון אחד במקום 2,בניית רכיבים כאלה שיכולים לבצע יותר מפעולה אחת באותו מחזור שעון יעזור לנו בלקצר משמעותית את זמן ריצת האלגוריתם, ולבסוף נבדוק את הנכונות שלו בסימולציות, הסינתזה וכמובן נבדוק האם מה שבנינו עובד חלק ולפי הדרישות על שבב Kintex-7 של חברת Xilinx.

<u>הערה חשובה להמשך:</u>

האלגוריתם שלנו נעזר תקן ההצפנה הלאומי AES כדי להצפין/לפענח שמשתמש בפונקציות Rijndael שהן קבוצה של פונקציות סימטריות שעוזרות לנו לערבל שורות/עמודות המטריצות שמהוות את המילה שרוצים להצפין/לפענח או להחליף חלקים מהמטריצה באופן סימטרי על ידי שימוש במטריצות כמו s_box של Rijndael שמקבילה לחסור ו- Inv_s_box שמקבילה ל"הופכי של חיסור".

על ידי שימוש בקבוצת הפונקציות האלו אנו דואגים שההצפנה/פיענוח יהיו כמה שיותר "בטוחים"

פונקציות Rijndael מחולקות לשתי קבוצות פונקציות "כפל" ופונקציות "חיסור" כך ש- :

- פונקציות הכפל הינן mul2, mul3, mul9, mul11, mul13 and mul14 כך ש- mul2 למשל מקבילה ל- "כפל ב2" וכך הלאה.
- פונקציית "חיסור" וההופכית שלה sub and isub .

1 הקדמה

1.1 מוטיבציה:

למה בכלל להריץ אלגוריתם הצפנה? ולמה כדאי להאיץ?

שאלות חשובות שהתשובה שלהן פשוטה ... ביטחון.

הצפנה הינה השיטה הטובה ביותר לשמור על אבטחת הנתונים כך שהיא מגינה על תוכן הקבצים ואף אחד שאין לו את ה"כלי" לבצע את הפיענוח המתאים לא יוכל להשתמש בהם, ותמיד עדיף להשתמש במאיצי חומרה לסיבה שאפילו יותר פשוטה ... מהירות, הגענו ל-2023 ולהשתמש בהצפנה של קבצים שתיקח שבוע , כמה ימים או אפילו כמה שעות לפעמים תפגע במיקום של החברה בשוק ותפסיד מול מתחרים.

להשתמש בתוכנה או חומרה לבד כדי לבצע את ההצפנה נותנת לנו או הצפנה "טובה" אבל בזמן מאוד ארוך שיכול להגיע לשבוע, או הצפנה גרועה אבל בזמן הצפנה מצוין, ומכאן הגיע הצורך לפתרון ביניים שמשלב פתרון חומרתי עם פתרון תוכנתי.

עוד שאלה מעניינת, האם יש מי שכבר עשה את זה?

כן יש כמה חלופות למשל CBC and EBC methods .

<u>אז למה OCB?</u>

ECB הינה שיטת הצפנה על ידי חלוקה לבלוקים נפרדים אבל על ידי מפתח הצפנה אחד משותף לכל הבלוקים ,דבר שפוגע ברמת ההצפנה למשל נוכל להצפין תמונה כלשהיא והתמונה החדשה נבחין בצורה המקורית של התמונה למרות שלא רצינו, מצד שני יש לנו את הCBC שגם הוא משתמש בחלוקה לבלוקים אבל לכל בלוק יש לו מפתח הצפנה אחר ,החיסרון הוא שווקטור ההתחלה משודר יחד עם הקובץ המוצפן מה שיכול לגרום בסיבה העיקרית להצפנה... בטיחות.

1.2 גישה לפתרון הבעיה:

<u>לפני שנתחיל בעבודה נצרך:</u>

1. לחקור לעומק את אלגוריתם ההצפנה OCB ותקן ההצפנה שמבוסס עליו שהינו תקן ההצפנה המתקדם AES.
2. ללמוד איך עובד המעבד MicroBlaze ומה המאפיינים שלו.
3. ללמוד איך בונים Platform ואיך משתמשים בתוכנת Vitis כדי להריץ ולחבר בין האלגוריתם והמאיץ החומרתי.

<u>שלבי עבודה:</u>

אחרי שעברנו, חקרנו ולמדנו את החומר הנדרש :

1. לכתוב קוד בשפת C ,יעיל ומסודר כמה שאפשר.
2. לבצע profiling , כדי לראות איזה חלק/ים צריך להחליף במאיץ חומרתי.
3. נכתוב תוכן חומרתי למאיץ בשפת Verilog ונבדוק אותם על ידי סימולציות (Test benches).
4. נחבר את המאיץ ל- platform ונבצע את השינויים הנדרשים באלגוריתם .
5. נריץ פעם בלי המאיץ/ים ופעם עם וננתח את התוצאות.

1.3 מטרות הפרויקט :

אחרי שנקבל את התוצאות ונוציא את כל הדוחות שצריך להוציא נעבור על הכל ובמיוחד 3 דוחות הכי משמעותיות בעולם של VLSI ונדרוש עבור כל אחר דרישה שבסוף נשאף להגיע אליה והן :

- Timing report- בדוח הזה נבדוק שעמדנו בכל דרישות הזמן של המערכת ושאין לנו שגיאות setup and hold.
- Power report- בדוח הזה נבדוק כמה הספק צורך הרכיב וכמה הספק מבזבז ונדרוש שההספק שמתבזבז יהיה מינימלי כמה שאפשר , ובמקרה הכי גרוע לא תעבור את ה25% מההספק שמתבזבז על ידי המעבד עצמו.
- Utilization report - פה מילת המפתח הינה "כסף", ובעולם הVLSI שטח שווה כסף, ולכן נשאף לממש את הרכיב שלנו בשטח הכי קטן שאפשר ,ובמקרה הכי גרוע שלא יעבור את 40% משטח המעבד עצמו.

דרישה נוספת הינה זמן (במחזורי שעון) נבדוק כמה לקח לרכיב שלנו לרוץ ולבצע את הכל ונדרוש שזמן הריצה שלנו יהיה כמה שפחות, ובמקרה הכי גרוע נקבל שיפור של 10% בין הרצה עם מאיץ להרצה בלי.

כל הבדיקות האלו ועוד יבוצעו בהתחלה בעזרת סימולציות של תוכנת Vivado של חברת Xilinx ואז בסוף נוודא אם אכן כל הדרישות הנ"ל מתקיימות בזמן אמת על השבב בעזרת Vitis.

2 רקע תאורטי

2.1 תיאור הפרויקט:

אלגוריתם ההצפנה OCB מתואר עי ידי ה pseudo code, ודיאגרמת הבלוקים הבאים :

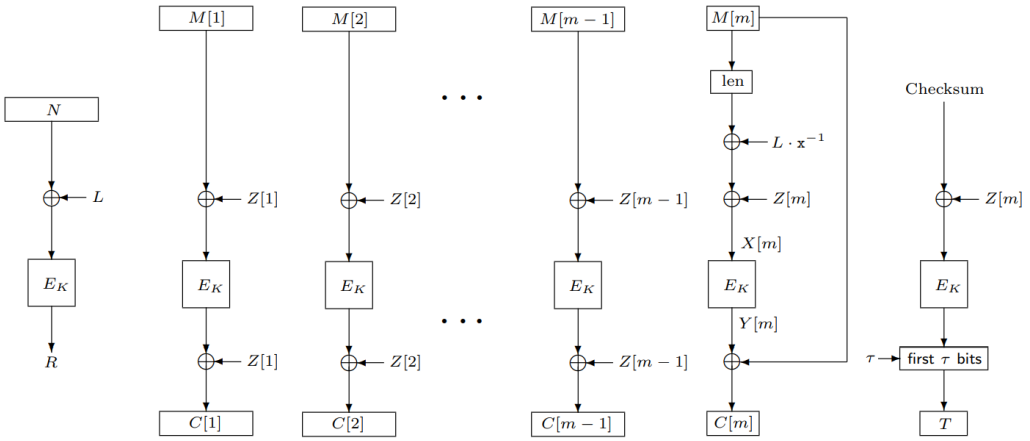
Algorithm $\mathrm{OCB.Enc}_K\ (N, M)$	**Algorithm** $\mathrm{OCB.Dec}_K\ (N, \mathcal{C})$
Partition M into $M[1]\cdots M[m]$	Partition $\mathcal{C}$ into $C[1]\cdots C[m]\ T$
$L \leftarrow E_K(0^n)$	$L \leftarrow E_K(0^n)$
$R \leftarrow E_K(N \oplus L)$	$R \leftarrow E_K(N \oplus L)$
for $i \leftarrow 1$ **to** m **do** $Z[i] = \gamma_i \cdot L \oplus R$	**for** $i \leftarrow 1$ **to** m **do** $Z[i] = \gamma_i \cdot L \oplus R$
for $i \leftarrow 1$ **to** $m-1$ **do**	**for** $i \leftarrow 1$ **to** $m-1$ **do**
$\quad C[i] \leftarrow E_K(M[i] \oplus Z[i]) \ \oplus\ Z[i]$	$\quad M[i] \leftarrow E_K^{-1}(C[i] \oplus Z[i]) \ \oplus\ Z[i]$
$X[m] \leftarrow \mathrm{len}(M[m]) \oplus L \cdot \mathtt{x}^{-1} \oplus Z[m]$	$X[m] \leftarrow \mathrm{len}(C[m]) \oplus L \cdot \mathtt{x}^{-1} \oplus Z[m]$
$Y[m] \leftarrow E_K(X[m])$	$Y[m] \leftarrow E_K(X[m])$
$C[m] \leftarrow Y[m] \oplus M[m]$	$M[m] \leftarrow Y[m] \oplus C[m]$
$C \leftarrow C[1]\cdots C[m]$	$M \leftarrow M[1]\cdots M[m]$
Checksum $\leftarrow$	Checksum $\leftarrow$
$\quad M[1] \oplus \cdots \oplus M[m-1] \oplus C[m]\,0^* \oplus Y[m]$	$\quad M[1] \oplus \cdots \oplus M[m-1] \oplus C[m]\,0^* \oplus Y[m]$
$T \leftarrow E_K(\mathrm{Checksum} \oplus Z[m])$ [first τ bits]	$T' \leftarrow E_K(\mathrm{Checksum} \oplus Z[m])$ [first τ bits]
return $\mathcal{C} \leftarrow C \parallel T$	**if** $T = T'$ **then return** M
	else return INVALID

איור 2 : תיאור האלגוריתם

כך ש :

1. M- הינו מערך של בלוקים שכל בלוק מכיל חלק מהמילה.
2. C- המילה המוצפנת
3. N- מהווה את ה- Nonce.
4. T- מהווה את ה- Tag

5. L,Y,Z,R הינם מערכי עזר.
6. E_k מתאר הרצת תקן ההצפנה המתקדם AES, לפי מפתח k.



איור 3 : דיאגרמת בלוקים לאלגוריתם

האלגוריתם מתחיל בלייצר מהמפתח סט של מפתחות שבמקרה שלנו היות וגודל הבלוק הינו 128 ביט אז מייצרים 10 מפתחות, ואז מייצרים סט של מערכים שבעזרתם מצפינים או מפענחים את המילה ואז מייצרים לפי התיאור התכנתי מילה שהיא תוצאת ההצפנה/פיענוח .

בנוסף מייצרים מילה חדשה בשם tag שמשרשרים לתוצאה במידה ומדובר בהצפנה או מורידים אותו לפני הפענוח ואז מיצרים אחת אחרת ודרכה בודקים אם אכן הגענו למילה המקורית אם שתיהן זהות או שהייתה איזה שהיא בעיה והפלט שגוי במידה והן שונות.

במילים פשוטות האלגוריתם שלנו מקבל מילה שצריך להצפין או לפענח ,מפתח להצפנה/פיענוח, ו- Nonce, אחרי שמבצעים הצפנה מקבלים מילה מוצפנת שמשורשר אליה tag שבעזרתו אפשר לבדוק בזמן הפיענוח אם נקבל את המילה הנכונה בחזרה או לא.

3 מימוש:

3.1 תיאור תוכנה:

הקוד נכתב בהתחלה בתוכנת Visual Studio, חולק לחלקים כך שכל חלק מתואר על ידי פונקציה מתאימה וכל סט של פונקציות שמתאר פרק כלשהו באלגוריתם נכתב ב- module מתאים, בקוד יש 4 modules, שני קבצי header לצורך הצהרה על פונקציות חיצוניות וקריאה לספריות חשובות והגדרת Enums, וקובץ C ראשי main.

3.1.1 מחלקות:

1. Rijndael_functions : שמכיל את כל הפונקציות של Rijndael שנעזרים בהן להצפנה הפנימית AES.
2. Encrypt_functions : שמכיל כל הפונקציות הדרושות לצורך ההצפנה לפי תקן AES.
3. Decrypt_functions : שמכיל כל הפונקציות הדרושות לצורך הפיענוח לפי תקן AES.
4. OCB_functions : שמכיל כל הפונקציות הדרושות להצפנת/פיענוח מילה שלמה.

3.1.2 **קבצי הגדרות:**

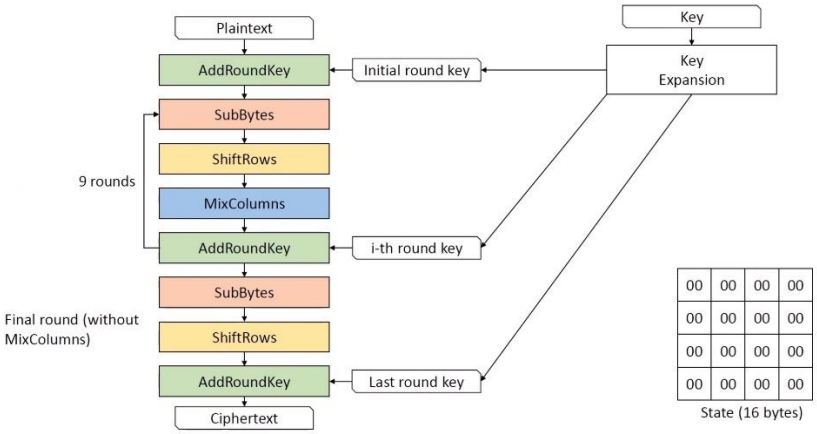
1. AES_defines : שמכיל קריאות לספריות חשובות לצורך הצפנת בלוק אחד והגדרת enums שמתארים שמות מטריצות Rijndael ותיאור שגיאות.
2. OCB_defines : שמכיל קריאות לספריות חשובות לצורך הצפנה מילה שלמה והגדרת Enums שמתארים מצבי פעולה אם זה הצפנה או פענוח ותיאור שגיאות.

3.1.3 **קובץ ראשי:**

1. Main : שמחבר את הכל יחד ומצפין או מפענח את הקלט.

3.1.4 **תיאור אלגוריתם העזר AES-128B :**

תקן AES ידוע ב3 קונפיגורציות 128 ביטים 192 ביטים או 256 ביטים, החלטנו לממש אלגוריתם הצפנה של 128 ביטים ולכן הצפנת כל בלוק ייקח 11 סיבוב שפי שמתואר בדיאגרמה הבאה :



איור 4 : דיאגרמת בלוקים של תקן AES-128B.

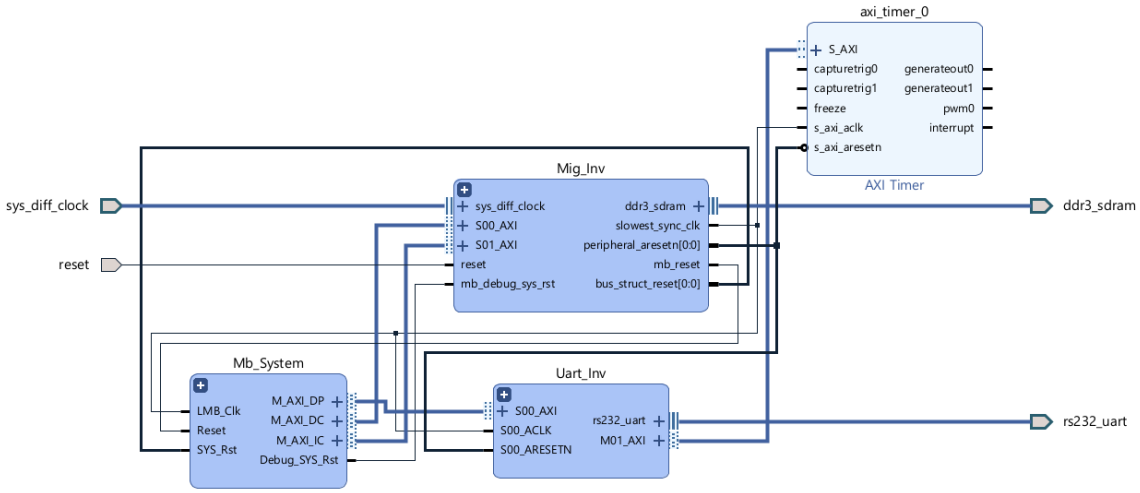
חשוב לציין שכל בלוק ממופה למטריצה מצב דו ממדית M_{4X4} שנקראת State.
בנוסף כל שלב מהשלבים Add_RoundKey, SubBytes, ShiftRows and MixColumns משתמש ולפחות אחת מפונקציות Rijndael.

3.1.5 **תיאור אלגוריתם OCB שנכתב :**

האלגוריתם שלנו מייצר כמה מערכי עזר שהם L_Arrays, Z_Arrays and L_Inv_Array.
אחרי האתחול של המערכים האלו נחלק את המילה המבוקשת למערך של בלוקים ובמידה שאורך המילה אינו מכפלה שלמה של 128 ביטים מרפדים באפסים ואז מבצעים כמה פעולות של xor והצפנה לפי תקן AES-128B שתיארנו בפרק הקודם כך ש- מיוצרים עוד שני מערכי עזר שהינם X_Array, Y_Array שבעזרתם מייצרים את המילה המוצפנת/מפוענחת ואת ה- Tag במידה ומדובר בהצפנה כמו שתואר בפרק תיאור הפרויקט.

3.2 תיאור סיבית עבודה:

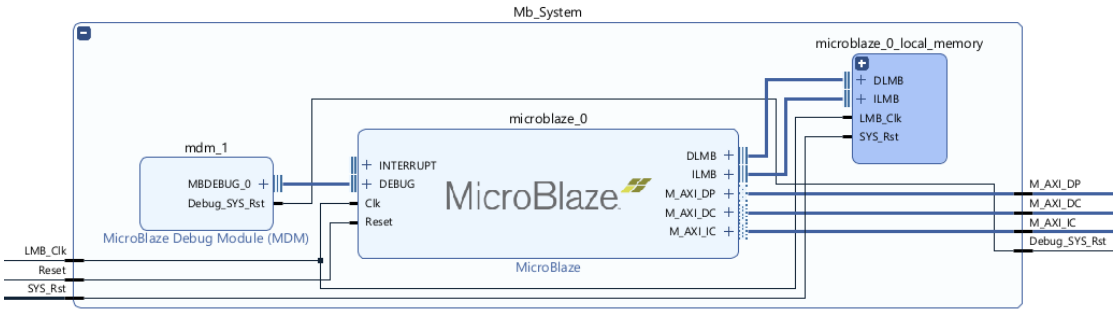
כדי שנריץ את האלגוריתם על שבב Kintex-7 KC705 בנינו את ה- platform הבאה:



איור 5: פלטפורמה כללית ללא מאיץ

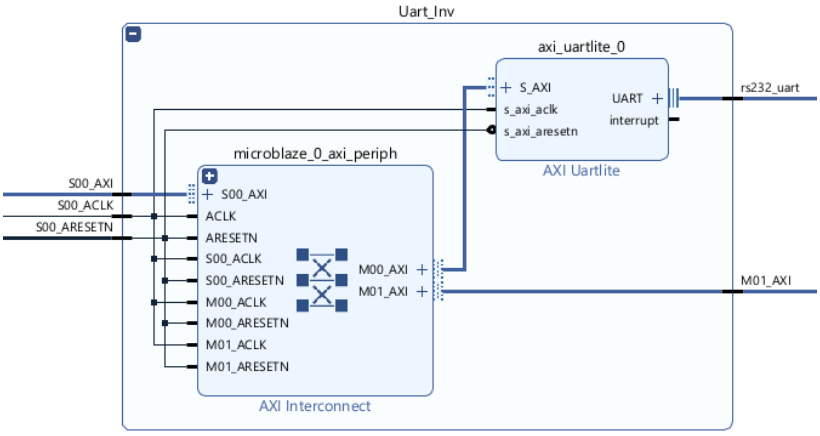
3.2.1 תיאור היררכיות:

1. MB_System: שמכיל את מעבד MicroBlaze עם הזיכרון הפנימי שלו MicroBlaze local Memory וגם MicroBlaze debug Module (MDM).



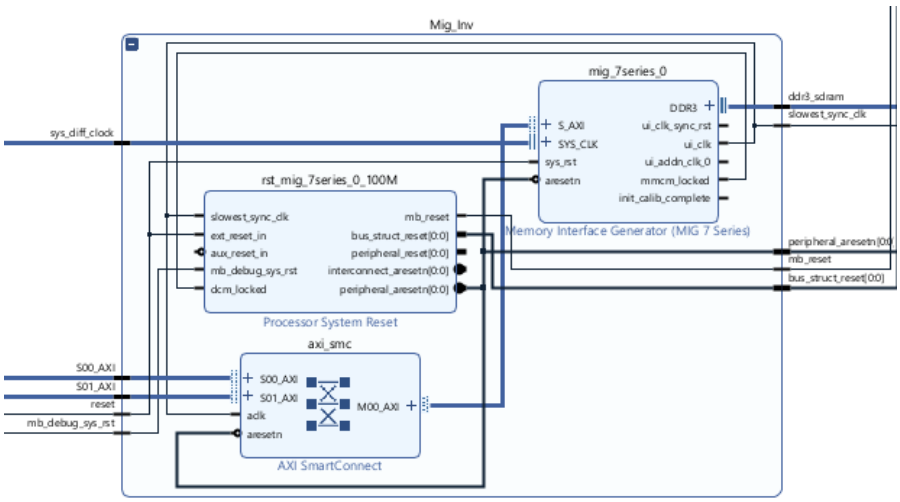
איור 6: היררכיית המעבד

2. Uart_Inv: שמכיל את uartlite שבעזרתו מוציאים את הפלט בצורה טורית וה- interconnect שבעזרתו מחברים בין המעבד למוצא הטורי.



איור 7: היררכיית Uart

3. Mig_Inv : שמכיל את הזיכרון החיצוני שנמצא על השבב ויחידת ה reset שלו ורכיב interconnect שמחבר בין המעבד לזיכרון החיצוני של השבב, בנוסף לכך הרכיב מייצר את השעון שמריץ את המעבד.



איור 8 : היררכיית זיכרון השבב

בנוסף חיברנו timer שבעזרתו נוכל למדוד את זמן הריצה במחזורי שעון.

אופן שימוש :

1. כדי שנוכל להשתמש בפקודות הטיימר עשינו include לספריה xtmrctr.h.
2. הגדרנו 3 משתנים start_cycles,end_cycles,elapsed_cycles שמהווים את מספר מחזורי השעון ההתחלתי, הסופי וההפרש בינם בהתאם.
3. השתמשנו בפקודה asm volatile("mfs %0, rmsr" : "=r" (start_cycles)) כדי למדוד את מספר מחזורי השעון ההתחלתי לפני שמתחילים את ההצפנה.
4. אחרי סיום העבודה השתמשנו בפקודה asm volatile("mfs %0, rmsr" : "=r" (end_cycles)) כדי למדוד את מספר מחזורי השעון הסופי.
5. ולבסוף חישבנו את ההפרש בין שני הערכים האלו כדי לדעת את מספר המחזורים הנדרש.

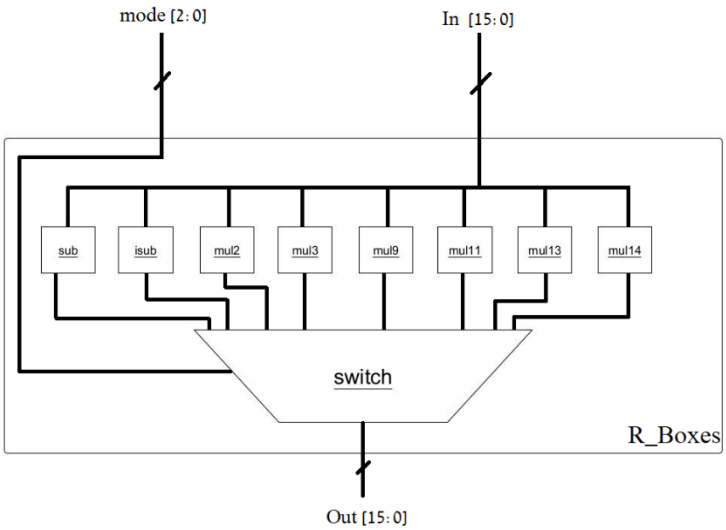
3.3 <u>ביצוע profiling :</u>

בעזרת visual studio ביצענו profiling לקוד שלנו וקיבלנו את דוח של כמה קריאות מתבצעות לעבר כל פונקציה וסיכמנו בטבלה מספר () 10 הפונקציות שמספר הקריאות להן הכי גבוה והסתכלנו על תרגום קוד הסי לשפת מכונה גם דרך visual studio כדי להחליט מי הפונקציה/ות שצורכות הכי הרבה מחזורי שעון ובסוף אחרי כמה סדיקות החלטנו להחליף את הפונקציה R_boxes שמכילה את מטריצות Rijndael והפונקציות שבעזרתן מבצעים ערבוב שורות/עמודות וחיסור לפי Rijndael וגם הפעולות ההופכיות שלהן.

3.4 תיאור חומרה – מאיץ :

לפי ההערה מהתקציר שכתבנו בהתחלה בנינו פונקציה שנקראת R_Boxes שהיא אחראית לביצוע פונקציות Rijndael שהזכרנו מקודם, לפי טבלה מספר (9.1) אנו יכולים לראות שהפונקציה הזאת היא החומרה ביותר מבחינת זמן ריצה ולכן החלטנו להחליף אותה בחומרה מתאימה.

בנינו את החומרה המתאימה לפי דיאגרמת הבלוקים הבאה :

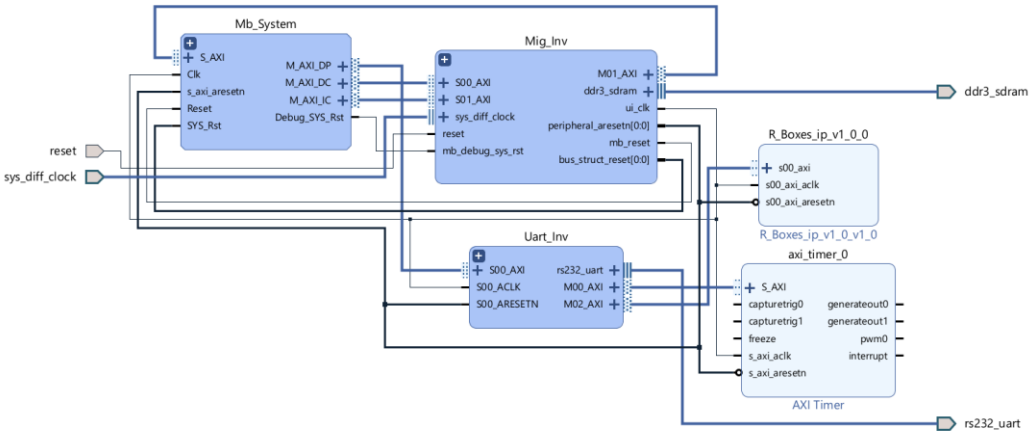


איור 9 : דיאגרמת בלוקים של המאיץ

וכמובן כתבנו לה testbench ואחרי שווידינו שהפלט כינו אכן מתאים לפלט של פונקציות Rijndael שילבנו אותה לתוך IP חדש שחיברנו אותו לפלטפורמה שנראה בעמוד הבא.

3.5 תיאור סביבת עבודה עם מאיצים :

אחרי שהוספנו את ה- IP החדש שלנו קיבלנו את ה- platform הבאה :



איור 10 : פלטפורמה כללית כוללת מאיץ

אחרי זה עברנו לתוכנת Vitis שוב, והסתכלנו בקובץ xparameters שהוא הקובץ שמכיל מידע עבור כל הרכיבים ב-platform שכולל כתובות שלהם, טווח כתובות של הזיכרון המוקצה עבור כל IP שצריך זיכרון, והחלפנו את שורות הקוד שלנו על ידי כתיבה לזיכרון של הכניסות של ה- IP החדש ואחר כך קריאה מהזיכרון שמתאים למוצא שלו.

4 ניתוח תוצאות:

4.1 השוואה בין דרישות התכנון לתוצאות בזמן אמת:

תוצאות הרצת האלגוריתם ללא מאיץ:

without accelerator	Input length [Bytes]	run time [Clock Cycles]	power consumption [Watt]	size [Slices]
1	0	36772588	2.085	21747
2	3	32145660	2.085	21747
3	16	40190664	2.085	21747
4	20	46123856	2.085	21747
5	32	53774479	2.085	21747
6	34	58667103	2.085	21747
7	1000	449061896	2.085	21747

טבלה 1 : תוצאות הרצת האלגוריתם ללא מאיץ

תוצאות הרצת האלגוריתם עם מאיץ:

with accelerator	Input length [Bytes]	run time [Clock Cycles]	power consumption [Watt]	size [Slices]
1	0	27187805	2.119	21925
2	3	22399810	2.119	21925
3	16	30529948	2.119	21925
4	20	33142411	2.119	21925
5	32	40634504	2.119	21925
6	34	42067212	2.119	21925
7	1000	338491355	2.119	21925

טבלה 2 : תוצאות הרצת האלגוריתם עם מאיץ

כלומר, בממוצע קיבלנו ש-:

In average	
without accelerator	run time [Clock Cycles]
	102390892.3
	power consumption [Watt]
	2.085
	size [Slices]
	21747
with accelerator	run time [Clock Cycles]
	76350435
	power consumption [Watt]
	2.119
	size [Slices]
	21925
run time after Vs before [Clock Cycles]	
25.43%	
power on chip after VS before [Watt]	
1.63%	
size of new IP VS size of MicroBlaze[Slices]	
22.25%	

טבלה 3 : סיכום תוצאות

השוואה בין הדרישות לתוצאות:

	real time	requirements
run time improvement [%]	25.43	>10
power consumption increment [%]	1.63	<25
utilization increment [%]	22.25	<40

טבלה 4: השוואה ביון דרישות לתוצאה

כלומר הצלחנו להגיע לתוצאות מאוד טובות ולעמוד בכל הדרישות מבלי שיהיה לנו שגיאות setup and hold .

5 סיכום, מסקנות והצעות להמשך:

5.1 סיכום ומסקנות:

כמו שראינו בפרקים הקודמים הצלחנו לסיים את הפרויקט עם תוצאות הרבה יותר טובות מהדרישות מה שמעיד על שיפור משמעותי בזמן ריצה עם תוספת מנמלית של שטח והספק ולכן נוכל להסיק שלפעמים צריך להסתכל על קוד בשפת תכנות כלשהי ולשקול אם עדיף/צריך להחליף חלק ממנה בקוד חומרתי כדי לשפר את זמן הריצה שלה .

5.2 הצעות להמשך:

בפרויקט שלנו הראנו איך אפשר לשפר את הביצועים על ידי החלפת חלק מהקוד שכתוב בשפת סי בקוד חומרתי, אבל עדיין יש שאלות שלא נוכל לענות עליהן רק על ידי הניסוי שעשינו למשל:

1. האם תמיד צריך לשלב קוד בשפת תכנות לקוד חומרתי?
   כדי לקבל תשובה לשאלה הזאת צריך לכתוב מאיצים עבור כמה אלגוריתמים ולהשוות ביניהם.
2. האם צריך להחליף את כל הקוד שכתוב בשפת תכנות לקוד חומרתי ?
   כדי לקבל תשובה לשאלה הזאת צריך קוד חומרתי שקול לכל האלגוריתם ולנתח את התוצאות.
3. האם אלגוריתם OCB הינו הכי טוב להצפין או שזה נכון במקרים מסוימים ?
   כדי לקבל תשובה לשאלה הזאת צריך לנתח להריץ ולהשוות את הביצועים של אלגוריתמי הצפנה קיימים ולחשוב אולי אם יש עוד רעיון לכתיבה שלא קיים עדיין.

6 פרק נוסף – שילוב זיכרון דינמי:

בזמן העבודה על הפרויקט היה מאוד מסקרן לדעת מה ישתנה אם נחליף את הזיכרון הסטטי בזיכרון דינמי רצינו בעצם לראות האם בכך נוכל לחסוך בשימוש בזיכרון מיותר ולייעל יותר את האלגוריתם, והאם זה ישפר את זמן הריצה או לא.

ולכן שינינו את האלגוריתם כך שישתמש ברוב חלקיו בהקצאות זיכרון דינמי ובסיום השימוש ישחרר אותם והרצנו שוב על השבב פעם בלי מאיץ ופעם עם מאיץ, וסיכמנו את התוצאות בטבלה מספר (9.2) וכהשוואה בין התוצאות של עם זיכרון דינמי לבלי קיבלנו את הטבלה הבאה :

	runtime_average	power_average	size_average
without dynaimc and without accerelator	102390892.3	2.085	21747
without dynaimc and with accerelator	76350435	2.119	21925
with dynaimc and without accerelator	107386881.7	2.085	21747
with dynaimc and with accerelator	76410641.86	2.119	21925
differences in average	run time [clock cycles]	power [Watt]	size [slices]
without dynamic memory	26040457.29	0.034	178
with dynamic memory	31036446.71	0.034	178
differences in average in percentage	run time [clock cycles]	power [Watt]	size [slices]
without dynamic memory	25.432	1.631	0.819
with dynamic memory	28.902	1.631	0.819

טבלה 5 : השוואה בין התוצאות עם שימוש בזיכרון דינמי ובלי

כפי שאפשר לראות מהטבלה מספר מחזורי השעון בלי הוספת המריץ אכן גדל כתוצאה מפקודות ההקצאה ושחרור הזיכרון אבל אחרי הוספת המאיץ קיבלנו שיפור של כמעט 29% כלומר כמעט 4% יותר מהניסוי בלי שימוש בזיכרון דינמי ולכן נסיק ששימוש בזיכרון דינמי שיפר לנו את הביצועים יותר ואם יש אפשרות להשתמש בו אז כדאי.

7 תיעוד הפרויקט:

לינק לפרויקט באתר GitHub : https://github.com/BaselDOS/OCB

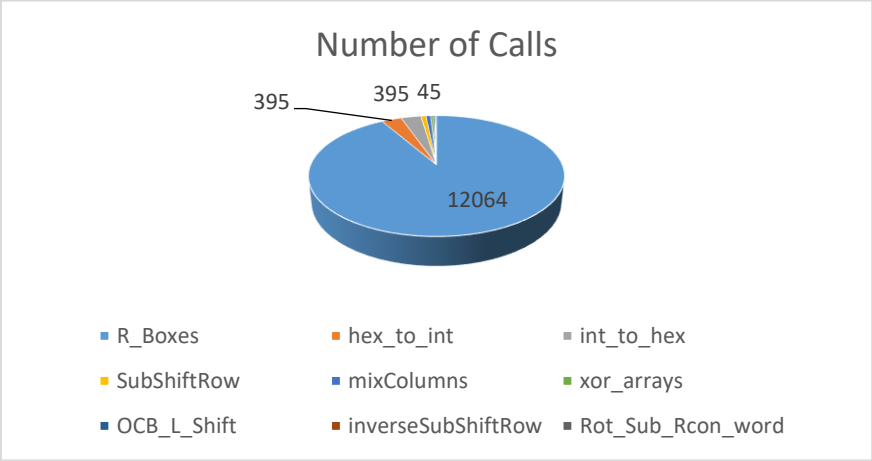
תיעוד הפרויקט:

באתר הפרויקט אפשר לראות תיקיה מרכזית בשם OCB וקובץ טקט בשם Readme.txt שכולל הסבר דומה להסבר שיש כאן, בתוך התיקייה הראשית ישנם כמה תיקיות :

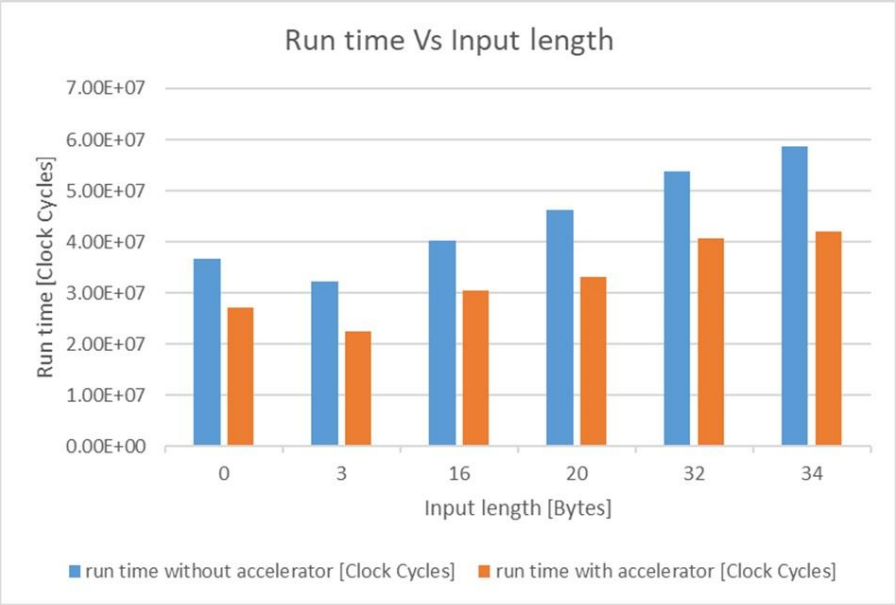
1. Pdf : בתוך תיקיה זו אפשר למצוא את תכנית העבודה מצגת האמצע ואת דוח ההתקדמות בנוסף לכל חומר קריאה שנעזרנו בו שהינו מטיפוס PDF .
2. Src : שמכיל תת תיקיות שמכילות קבצי ה source לפי שפות :
   a. C_Code שמכיל שתי תת תיקיות שמכילות את האלגוריתם אחת עם הקצאות דינמיות ואחת בלי שבכל אחת יש שתי גרסאות של האלגוריתם שבלי מאיץ ועם.
   b. RTL. תיקיה זו מכילה את החומרה שנבנתה בשפת Verilog וה- test benches.
   c. Platforms שמכיל את שתי הפלטפורמות עם מאיץ ובלי.
3. Test_vectors שמכיל שני קבצי טקסט שבכל אחד יש מספר קלטה בדיקה.
4. Images שמכיל תמונות JPG של הפלטפורמות תוצאות הפלט במעבדה והשבב שהרצנו עליו, גרפים ועוד.
5. Reports : שמכיל כל הדוחות שהוצאנו .

8 רשימת איורים:

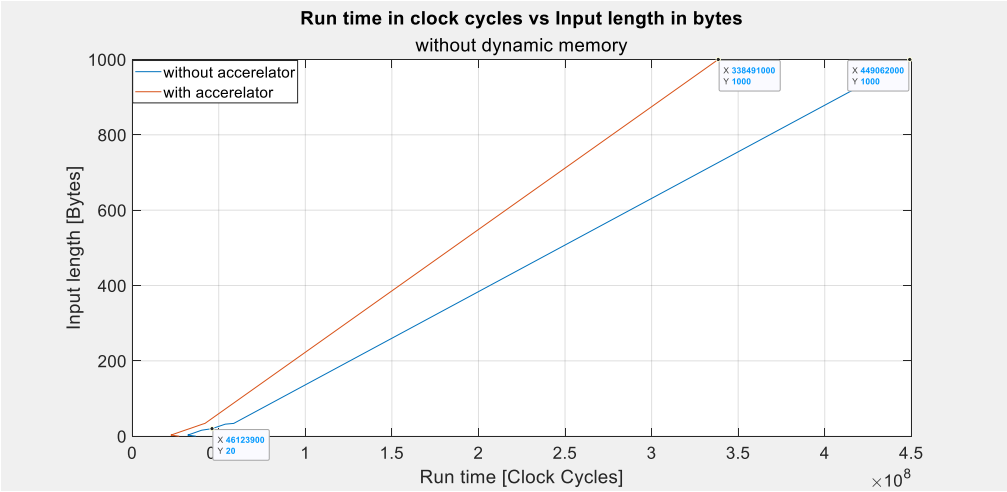
8.1 תיאור מספר הקריאות ל- 9 הפונקציות המשמעותיות



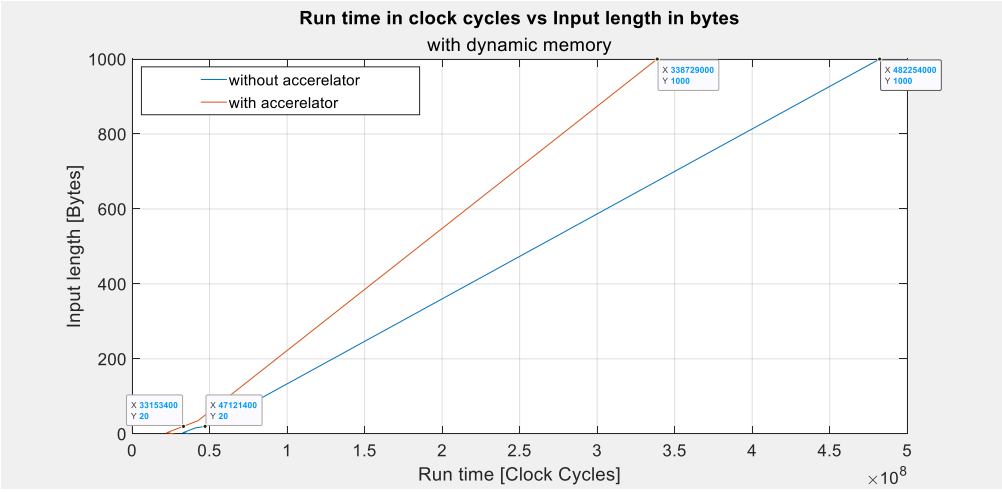
8.2 זמן הריצה יחסית לאורך הקלט כהשוואה למעגל עם מאיץ ובלי



8.3 השוואה - זמן הריצה כפונקציה של אורך הקלט בלי להשתמש בזיכרון דינמי



8.4 השוואה - זמן הריצה כפונקציה של אורך הקלט כולל שימוש בזיכרון דינמי



8.5 השבב שהשתמשנו בו

9 רשימת טבלאות :

9.1 רשימת הפונקציות המשמעותיות מבחינת מספר קריאות וזמן ריצתם באחוזים

Function Name	Number of Calls	Time %
R_Boxes	12064	37.38
hex_to_int	395	0.02
int_to_hex	395	0.01
SubShiftRow	100	0.9
mixColumns	90	2.22
xor_arrays	45	0.05
OCB_L_Shift	34	0.01
inverseSubShiftRow	20	0.28
Rot_Sub_Rcon_word	20	0.05
inverseMixedColumn	18	0.71
Array_1d_to_2d	12	0.02
Array_2d_to_1d	12	0.02
AESEncryption	10	0.26
str_to_Uchar_array	8	0.24
array_1d_to_str	8	0.05
printf	7	0.08
OCB_enc_dyc	2	0.45
OCB_init	2	0.29
log2	2	0.12
plainntext_to_blocks	2	0.11
ceil	2	0.07
strncpy	2	0.07
ciphertext_blocks_to_str	2	0.07
round_keys_generator	2	0.03
zero_padding	2	0.02
main	1	0.13
Tag_remover	1	0.01

9.2 סיכום תוצאות - זמן ריצה, הספק ושטח

without dynamic memory				
without accelerator	Input length [Bytes]	run time [Clock Cycles]	power consumption [Watt]	size [Slices]
1	0	36772588	2.085	21747
2	3	32145660	2.085	21747
3	16	40190664	2.085	21747
4	20	46123856	2.085	21747
5	32	53774479	2.085	21747
6	34	58667103	2.085	21747
7	1000	449061896	2.085	21747
with accelerator	Input length [Bytes]	run time [Clock Cycles]	power consumption [Watt]	size [Slices]
1	0	27187805	2.119	21925
2	3	22399810	2.119	21925
3	16	30529948	2.119	21925
4	20	33142411	2.119	21925
5	32	40634504	2.119	21925
6	34	42067212	2.119	21925
7	1000	338491355	2.119	21925
with dynamic memory				
without accelerator	Input length [Bytes]	run time [Clock Cycles]	power consumption [Watt]	size [Slices]
1	0	37656169	2.085	21747
2	3	32860533	2.085	21747
3	16	40996864	2.085	21747
4	20	47121387	2.085	21747
5	32	54614437	2.085	21747
6	34	56205009	2.085	21747
7	1000	482253773	2.085	21747
with accelerator	Input length [Bytes]	run time [Clock Cycles]	power consumption [Watt]	size [Slices]
1	0	27201093	2.119	21925
2	3	22408559	2.119	21925
3	16	30528330	2.119	21925
4	20	33153373	2.119	21925
5	32	40645812	2.119	21925
6	34	42208419	2.119	21925
7	1000	338728907	2.119	21925

9.3 השוואה בין מעגלים עם זיכרון דינמי/מאיצים לבין בלי

	runtime_average	power_average	size_average
without dynaimc and without accerelator	102390892.3	2.085	21747
without dynaimc and with accerelator	76350435	2.119	21925
with dynaimc and without accerelator	107386881.7	2.085	21747
with dynaimc and with accerelator	76410641.86	2.119	21925
differences in average	run time [clock cycles]	power [Watt]	size [slices]
without dynamic memory	26040457.29	0.034	178
with dynamic memory	31036446.71	0.034	178
differences in average in percentage	run time [clock cycles]	power [Watt]	size [slices]
without dynamic memory	25.432	1.631	0.819
with dynamic memory	28.902	1.631	0.819

9.4 השוואה בין שטח המאיץ לשטח המעבד

size of our IP	178	
size of our IP VS size of platform/MicroBlaze		diffirence in percentage
Minimum size of Microblaze in Kintex-7	800	22.25
Maximum size of Microblaze in kintex-7	1000	17.8
size of platform without IP	21747	0.819

9.5 סיכום תוצאות

In average	
without accelerator	run time [Clock Cycles]
	102390892.3
	power consumption [Watt]
	2.085
	size [Slices]
	21747
with accelerator	run time [Clock Cycles]
	76350435
	power consumption [Watt]
	2.119
	size [Slices]
	21925
run time after Vs before [Clock Cycles]	
25.43%	
power on chip after VS before [Watt]	
1.63%	
size of new IP VS size of MicroBlaze[Slices]	
22.25%	

10 רשימת מקורות:

(1) https://web.cs.ucdavis.edu/~rogaway/ocb/ocb-faq.htm#describe-ocb

(2) https://en.wikipedia.org/wiki/Rijndael_MixColumns

(3) https://en.wikipedia.org/wiki/Rijndael_S-box

(4) https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

(5) https://datatracker.ietf.org/doc/html/draft-moskowitz-aes128-ocb-00

והכי חשוב מבין כולם:

(6) https://www.linkedin.com/in/glenn-kirilow-255640102/

והפוסט שכתב במיוחד שלנו:

(7)https://www.linkedin.com/safety/go?url=https%3A%2F%2Fwww.theeeview.com%2Fhow-does-memory-allocation-for-a-microblaze-work%2F&trk=flagship-messaging-web&messageThreadUrn=urn%3Ali%3AmessagingThread%3A2-MDg4ZmE4ZTctNzUxNy00OWExLWE1YzAtMDBhMjQxODI4MjYyXzAxMw%3D%3D&lipi=urn%3Ali%3Apage%3Ad_flagship3_profile_view_base%3B1aqjMBT0Q3ew2SPvjR6gVw%3D%3D

(8) https://github.com/ahegazy/aes/tree/master/src