Address Translation

&

IRAM Cache

מגישים:

אורי יפת – Ori.yeffet@mail.huji.ac.il

עומרי דסה – omrids@gmail.com

Group 327

מנחה:

אמיר פרן – amir.paran@intel.com

מהי הבעיה?

הפרויקט עוסק בשני רכיבים משמעותיים בכרטיס רשת חדשני מסוג Ethernet.

כרטיס רשת – רכיב חומרתי הנמצא במחשבים/שרתים ומהווה את שער הגישה שלהם אל רשת האינטרנט.

בכרטיסי רשת טיפוסיים כיום יש רכיבי עיבוד (CPU Controllers) אשר תפקידם לעבד נתונים, לנתח את תכולתם ולנתב אותם בהתאם לפרוטוקולי התקשורות הסטנדרטיים. רכיבי העיבוד הנ"ל נשלטים באמצעות סט פקודות Firmware הנשלפות מזכרון ההוראות – (Instruction Ram) IRAM בעת עבודת המעבד. על מנת לשנות/להוסיף פונקציונאליות לרכיב העיבוד ניתן לטעון סט פקודות שונה/משודרג אל תוך זיכרון ה- IRAM ובאופן זה לאפשר ורסטיליות של הרכיב והגדרת פיצ'רים מותאמים לדרישות של כל מחשב/שרת בנפרד.

לעיתים, גודל זיכרון ה- IRAM מגביל את כמות הפיצ'רים שיחידת העיבוד בשבב מסוגלת לספק לקליינט של כרטיס תקשורת. במילים פשוטות – אין מספיק מקום לכל הפקודות לכל הפיצ'רים. לפיכך נרצה להגדיל את מרחב הכתובות ל- IRAM ובכך לאפשר יותר פקודות שיתארו כמות גדולה יותר של פיצ'רים.

בנוסף, כאשר מעוניינים לשדרג את ביצועי הכרטיס ונדרשים לכוח מחשוב רב יותר, יש צורך לבצע מספר חישובים במקביל ולהעביר נתונים ממספר ערוצי תקשורת דרך ניתוב יחיד. אם כך יהיו מספר מעבדים בשבב אך עדיין ישנם מרחבי זכרון ציבוריים בשבב שלכל ערוץ יש צורך לגשת אליהם.

כשעובדים בחישוב מקבילי המתודולוגיה גורסת שעל כל מעבד לעבוד באופן נפרד וללא תלות בשאר המעבדים. כלומר, כל יחידת עיבוד עובדת אל מול מרחב הזיכרון הפומבי כמו הייתה היחידה.

למשל אם מעבד מספר 1 ירצה לגשת לכתובת 0x10 בזיכרון ומעבד מספר 2 ירצה לגשת לכתובת 0x10 בזיכרון, לאו דווקא מדובר באותה הכתובת. יש צורך להבין האם כתובת 0x10 הנה פומבית או אישית ורק לאחר מכן להמשיך את הגישה לזיכרון המתאים לאחר שנמפה את הכתובת 0x10 הוירטואלית לכתובת הפיסית אליה התכוון לגשת כל אחד מן המעבדים השונים.

לסיכום, הבעיות במעבר לדרישות כרטיסי הרשת החדשים הן:

מרחב הכתובות ל IRAM אינו מספק את הקליינטים של כרטיסי התקשורת.

שימוש במחשוב מקבילי הכולל מספר מעבדים ואזורים פומביים בצ'יפ דורש מערכת לתרגום הכתובות הפומביות והאישיות לכתובות פיזיות יחידות, ביעילות ובמהירות.

למה זה מעניין וחשוב?

הפרויקט שלנו הוא מתן פתרון בשלבי הפיתוח של כרטיסי הרשת החדשניים לשתי בעיות שנוצרו כתופעת לוואי להעלאת הדרישות מכרטיס הרשת. בתהליך שנבצע לצורך מתן הפתרונות אנו נחקור פתרונות דומים הניתנים בעולם החומרה, נציע פתרונות בעצמנו היכולים לפתור את הבעיה ונחפש את הפתרון הטוב ביותר, כאשר מספר אתגרים עומדים בפנינו:

- עלינו למצוא פתרונות שאינם מתפוצצים בקום.
- עלינו למצוא פתרונות שעומדים ב Performance.
- הפתרון צריך להיות ורסטילי ופרמטרי על מנת שיוכל לשמש לפרוייקטים עתידיים.

פתרון זה חשוב משום שהוא מהווה אבן דרך בפיתוח וקידום היכולות של כרטיסי הרשת: ללא רכיב טוב לתרגום ומיפוי הכתובות לא ניתן יהיה לבצע חישוב מקבילי עם זיכרון פומבי. הרחבת מרחב כתובות ה- IRAM הכרחי על מנת לספק לקליינטים של כרטיסי הרשת פיצ'רים נוספים.

פרויקט זה יוצר בנו עניין אישי רב שכן אנו לומדים מקרוב את עולם החומרה שנסקר באופן מצומצם באוניברסיטה אל מול עולם התוכנה. אנו זוכים באופן זה לעבוד במטודולגיות עבודה לתכנון דיגיטלי ולהבין את המהלך הכללי (FLOW) בשבב הכולל מבנים חומרתיים בסיסיים.

מעבר לעניין והפיתוח האישי שאנו מקבלים מפרויקט זה, יש עניין רב בעולם פיתוח החומרה והבנת הכלים החדשים שברשותינו שכן הטכנולוגיה בתחום התקדמה לעומת פרויקטים קודמים. פתרונות שלא היו רלוונטים לפיתוחי עבר יכולים כעת להיות רלוונטיים משום שהטכנולוגיה השתפרה והתקדמה. לכן אנו נדרשים להכיר פיתוחים רבים בעולם החומרה אשר עשויים להיות רלוונטיים לפרויקט ולבצע את ההתאמות הנדרשות, דבר זה מוסיף עניין רב לפרויקט.

איפה החדשנות?

ראשית פרויקט זה נמצא בחזית טכנולוגיית כרטיס הרשת המתוכנן לעבוד בקצבים שטרם נראו בשוק. עולם כרטיסי הרשת והעברת המידע מזנק בשנים האחרונות בדרישות מיצרני כרטיסי התקשורת. זינוק בדרישות המקום והזמן מייצרים אתגרים חדשים עליהם יש להתגבר.

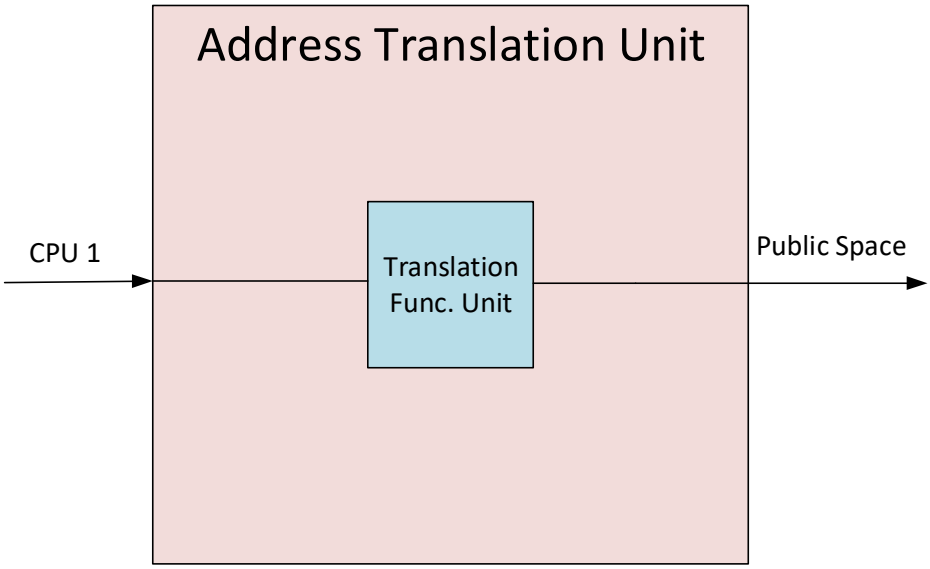
אחד מהאתגרים הנ"ל הוא העברת המידע בתדרים גבוהים יותר ללא איבוד או עיוות מידע ועל מנת לעשות זאת נוצר הצורך בחישוב המקבילי. בתכנון כרטיסי רשת קודמים, לא נדרש החישוב המקבילי ולכן פונקציית התרגום הייתה עובדת ללא מיפוי עבור יחידת עיבוד בודדת. בפרוייקט הזה יש צורך לחלק את משאבי זיכרון השבב למספר יחידות עיבוד ועל כן למפות את הכתובות של בקשות ממספר מעבדים למרחב הזיכרון הציבורי.

אתגר נוסף אשר פוקד את יצרני כרטיסי התקשורת הוא סיפוק מרחב זכרון הוראות גדול יותר ליחידות העיבוד מבלי לשלם במקום פיזי על גבי השבב על מנת שקליינטים של כרטיסי התקשורת יוכלו להוסיף בכל שדרוג תוכנה של כרטיסי התקשורת עוד ועוד פיצ'רים ליחידת העיבוד שלהם. Caching הנו פתרון מוכר לבעיה, אולם הפתרון שלנו טומן בחובו את היכולת לשמר את חלק ממרחב הזכרון Uncached בעוד חלקו הנוסף Cached ובאופן הזה לאפשר לקליינטים להחליט על פיצ'רים שאינם מוכנים לספוג ירידה בביצועים עקב הפיכת מרחב זכרון ההוראות ל Cached.

מהי האסטרטגיה שלנו והגישה לפתרון?

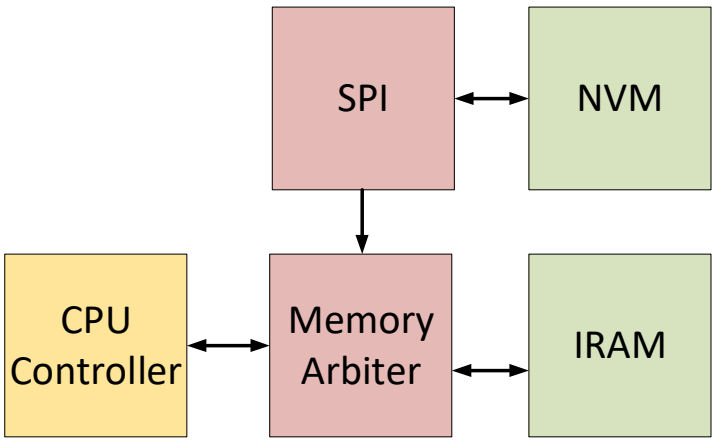
ביוון שהמודולים שאנו מעוניינים ליצור לא היו קיימים בפרויקטים קודמים בצורה שיענו על הדרישות הקיימות בפרויקט הזה, ראשית נבדוק ונראה כיצד השתמשו ברכיבים דומים בפרויקטים קודמים/אחרים.

חקירת Address Translation:



נשים לב שכאשר יש מעבד יחיד המשתמש באזורים הציבוריים בשבב ניתן להתייחס לאזורים אלו כאל "פרטיים" ועל כן ניתן ליצור פונקציית תרגום חומרתית חד-חד ערכית אשר תקשר בין בקשות המעבד המגיעות עם כתובות וירטואליות לכתובות הפיזיות של רכיבי הזיכרון הנמצאים באזורים הפומביים. באופן זה ניתן לבצע את התרגום בין כתובות וירטואליות לכתובות פיזיות בצורה מהירה ונוחה.

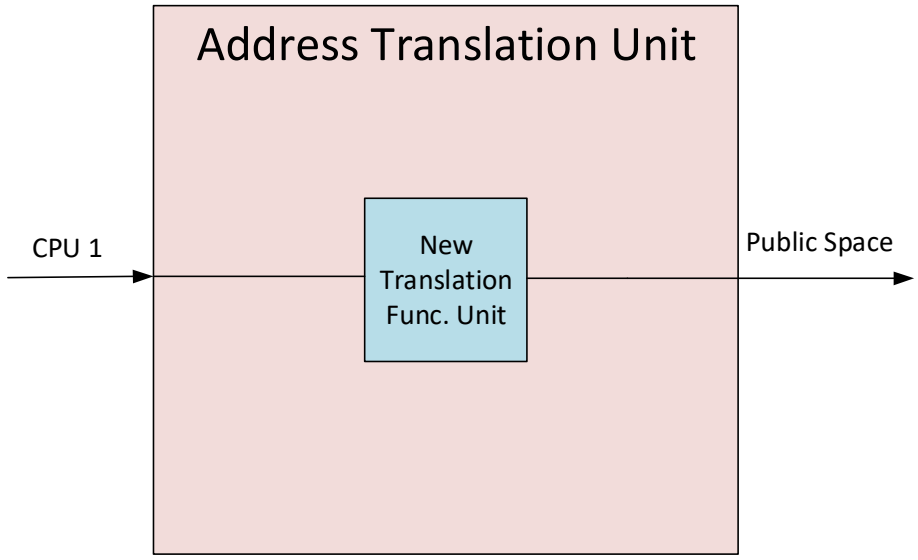
חקירת IRAM Cache:



כידוע זיכרון מסוג RAM אינו נשמר כאשר אין מתח ברכיב, ועל כן יש לטעון אותו מחדש בכל הפעלה. אולם יתרון משמעותי של זכרון מסוג זה לעומת NVM (Non Violated Memory) הינו מהירות שליפת מידע, על כן יש צורך כי הפקודות המפעילות את המעבד יהיו שמורות בזיכרון מסוג זה. בפרויקטים קודמים כשהיו מעלים את המתח לרכיב כחלק מתהליך תחילת הפעולה של הרכיב (boot sequence) זיכרון ה-IRAM היה נטען בפקודות שהיו שמורות בזיכרון איטי וגדול יותר – NVM.

לאחר שהבנו את צורות הפעולה שהייתה מקובלת בפרויקטים קודמים נרצה לתכנן פתרון שיהיה ניתן לייצר אותו בשלבים (הסקטבורד והמכונית) כך שנוכל לעקוב ולראות שהוא עובד בכל שלב בדרך ורק אז לשכלל אותו.

פתרון ביניים ל- Address Translation (P0):



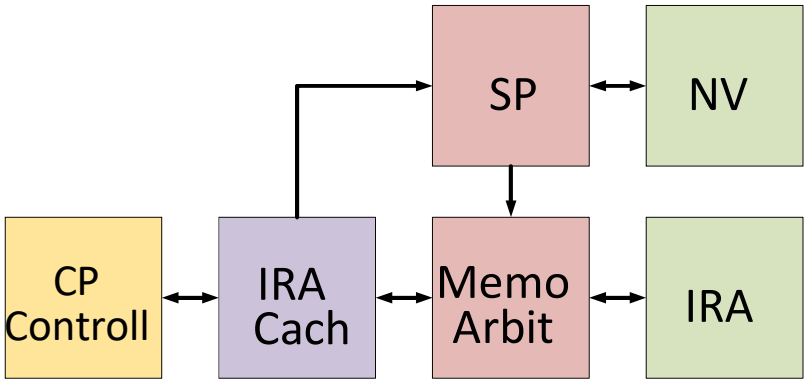
כיוון שהפתרון שלנו לתרגום הכתובות מצריך שני חלקים עיקריים ארביטרציה בין המעבדים השונים, ויצירת פונקציית תרגום חדשה, נוכל לפרק את הפתרון בצורה הבאה:

נחליף את הפונקציה החומרתית הישנה בפונקציה החומרתית החדשה ונבדוק שהיא מקיימת את הנדרש כלומר מתרגמת כתובות מהמרחב הווירטואלי למרחב הפיזי בצורה נכונה.

ולאחר מכן נוסיף את הארביטרציה בין המעבדים ונבדוק שהפונקציה עובדת עבור כולם.

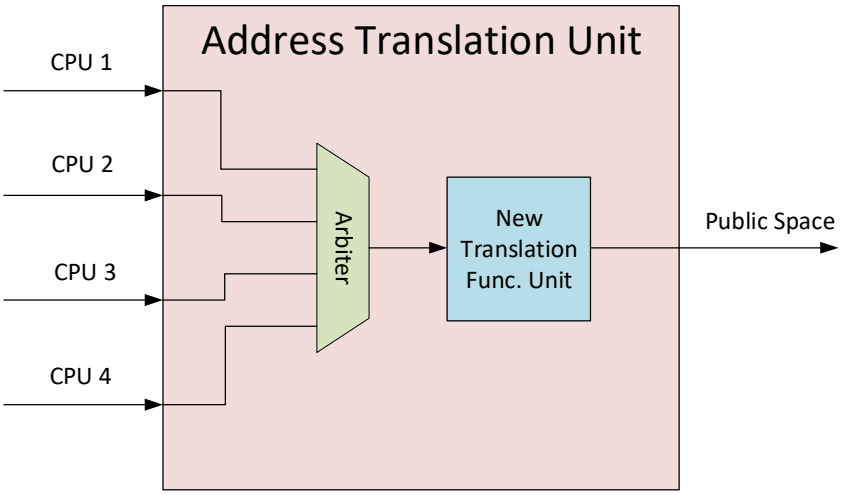
(נשים לב שאם היינו משנים את הסדר של פתרון הביניים כלומר קודם יוצרים ארביטרציה ומשתמשים בפונקציה הישנה ואז מחליפים את הפונקציה קל להבין שלא היה ניתן לבדוק את הרכיב בצורה כזו, משום שהפונקציה הישנה לא מיועדת לעבוד עבור מספר מעבדים.)

פתרון ביניים ל- IRAM Cache (P0):



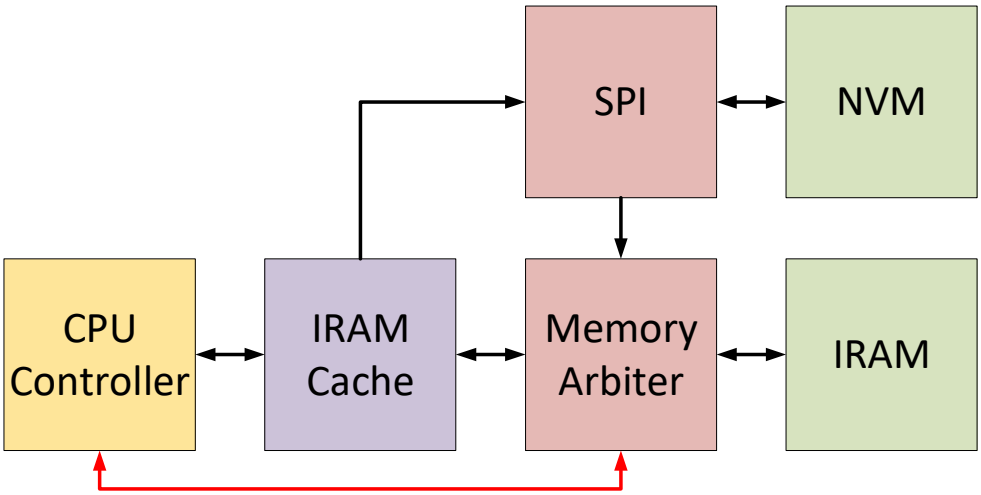
על מנת להרחיב את מרחב הזכרון של זכרון ה- IRAM נשתמש במטודולוגיית Caching ונהפוך את זכרון ה- IRAM כולו להיות Cached Memory. המשמעות הנה שיחידת העיבוד מייצרת בקשות להוראות באמצעות כתובות למרחב הזכרון של ה- NVM אשר יהיה ממופה למרחב הזכרון של ה- IRAM ע"י מנגנון ה- Cache. באופן זה ניתן להחזיק יותר הוראות למעבד בזכרון ה- NVM ולשלוף אותם מה- IRAM בצורה מהירה אם וכאשר הן אכן יושבות ב IRAM ואין צורך לכתוב אותן מה- NVM.

פתרון סופי ל- Address Translation (P1):



נרצה לשלב מנגנון היוצר ארביטרציה הוגנת בין בקשות של המעבדים השונים ובכל פעם שמעבד מסוים יזכה באביטרציה נתרגם את הפקודה שלו ונעביר אותה לאחר התרגום להמשך העבודה. נשים לב שכעת, לאחר שבדקנו את פונקציית התרגום שלנו, נותר לנו רק לבדוק את תהליך הארביטרציה. תהליך הארביטרציה גורם לנו לעבוד בכל פעם אל מול מעבד יחיד, והפונקציה החדשה שניצור תדע להתמודד מול בקשה של מעבד יחיד ולתרגם אותה בצורה טובה להמשך התהליך בזיכרונות. בכך נפתור את המגבלה שנוצרה במעבר לחישוב מקבילי ונוכל לעבוד מול מספר מעבדים במקביל.

פתרון סופי ל- IRAM Cache (P1):



נרצה להוסיף למנגון ה- Cache פיצ'ר המאפשר לנעול חלק ממרחב הזכרון של ה- IRAM ובכך לדאוג שתמיד שליפת הוראות מתוך מרחב זה יקרה בזמן אופטימלי. שאר מרחב הכתובות שנותר בIRAM יוותר Cached למרחב הכתובות של ה- NVM כמו בפתרון הביניים כלומר יכיל את ההוראות שהיו בשימוש לאחרונה וינוהל ע"י מנגנון ה- Cache ישירות.

בעזרת פתרון זה נצליח ליצור מרחק כתובות גדול ככל שנרצה (כגודל ה- NVM) ובנוסף נותיר לקליינטים של יחידת העיבוד את החופש לבחור בפיצ'רים מסויימים שלא יפגעו מבחינת Performance.

כיצד נבדוק את הפרויקט ונעריך את ביצועיו?

על מנת לבדוק את הרכיבים החדשים שנתכנן נעזר בשני כלים עיקריים- AV, Formal.

AV – בעזרת וריפיקציה (Verification) בסיסית נכסה מקרי קצה היכולים לקרות בפתרונות שלנו ובנוסף נבצע בדיקה עבור כמות גדולה של ערכים רנדומליים בכניסות למודול ונבדוק שאנו מקבלים את הערכים להם אנו מצפים ביציאה מן המודול.

בכדי לבצע את הבדיקות הנ"ל נדרש לבנות Test Bench כלומר מודולים נוספים שידעו להכניס את הערכים המתאימים בכניסות ולצפות לערכים המצופים ביציאות.

Formal –טכניקה לבדיקת נכונות. באמצעות Jasper שהנו כלי אשר בהנתן הנחות על הקלט מריץ את כל האפשרויות על גבי התכנון הדיגיטלי על מנת להוכיח התכנות של אירועים בקוד. באמצעות כלי זה ננסה להוכיח כי המודולים שבנינו מקיימים קבוצת חוקים אותם נקבע מראש. Formal יאפשר לנו להוכיח כי לא נגיע למצבים אסורים או שלא מתקיימת התנהגות לא מצופה ברכיב, ובכך בעצם להוכיח כי המודול תקין תחת ההנחות והחוקים שקבענו.

כשרוצים להעריך את הביצועים של חומרה מסוימת מבחינת יעילות מדד טוב הוא להסתכל על מספר מחזורי השעון הנדרשים בכדי לבצע פעולה מסוימת.

בשני הרכיבים שנבנה, כיוון שאנו מוסיפים לוגיקה על גבי פתרונות קיימים או מחליפים את הלוגיקה לאחת מסובכת יותר ניתן לצפות שלכל היותר נגיע ליעילות של הפתרון הקודם. כלומר נבצע את הפעולה המסובכת יותר באותה כמות של מחזורי שעון בהם בצענו את הפעולה הפשוטה יותר.

בנוסף על הפתרונות שלנו לא "להתפוצץ" במקום על גבי השבב. כלומר פתרון חומרתי טוב ככל שיהיה בבחינה פונקציונלית לא יכול להתקיים אם הוא חורג ממגבלות המקום הפיסי על גבי השבב.

הן את מגבלת המקום והן את מגבלת זמן הלוגיקה בתוך מחזורי השעון נוכל לבדוק ע"י כלי הסינטזה עליהן נפרט בהמשך.

איזה ידע רכשנו ועוד נרכוש?

במהלך הסמסטר רכשנו ידע הכרחי לתחילת עבודה על הפרוייקט:

- קראנו את הפרקים הראשונים של:

John L. Hennessy and David A. Patterson (2003). Computer Architecture: A Quantitative Approach, Third Edition, Morgan Kaufmann Publishers, Inc. ISBN 1558605967.

אלו הם הפרקים הרלוונטים הפורשים את הבסיס לארכיטקטורות מחשבים מרמת הטרנזיסטור הבודד כמתג ועד לרגיסטר.

- קראנו את הפרקים הנוגעים ל- Synthesis של:

Logic Synthesis and Verification. Editors: Soha Hassoun - Tufts University, Tsutomu Sasao - Kyushu Institute Of Technology.

התעמקנו בעיקר במימוש פיסי של Latches, Flops, Mux, Synchronizer שכן מסתמן שאלו הם הרכיבים העיקריים שחשוב שנבין לעומק את פרטי המימוש הפיסי שלהם. הן ב – Address Translation והן ב- IRAM Cache השימוש ברכיבים אלו אינטנסטיבי.
על הפרקים ב Verification דילגנו לעת עתה שכן נרשמנו לקורס Verification For Logic Designers של Intel (פירוט בהמשך)

- במהלך השנתיים האחרונות לקחנו קורסים הנוגעים לפרוייקט שלנו:

83328 INTRODUCTION TO ANALOG ELECTRONICS – transistor usage

83413 INTRODUCTION TO VLSI – components pros/cons

מבוא לאלקטרוניקה אנלוגית ומבוא ל VLSI הניחו את הקרקע להבנת תהליך ה Synthesis של רכיבי החומרה שאנו מתכננים.

67200 COMPUTER ARCHITECTURE – base components and usages

מבנה המחשב הפגיש אותנו לראשונה עם עולם החומרה – אופן פעולת המעבד והעברת נתונים.

83414 ADVANCED VLSI

בסמסטר האחרון למדנו בקורס ב VLSI מתקדם על ממשקים מהירים. צוואר הבקבוק בתעשייה ודרכי פתרון.

בשנה הבאה ישנם שני קורסים שהיינו מאד רוצים לקחת על מנת לבסס את האוריינטציה שלנו סביב תחום התקשורת:

67594 INTRODUCTION TO COMMUNICATION NETWORKS

67804 (INTER) NETWORK PROTOCOL AND ARCHITECTURES: THEORY AND PRACTICE

במהלך הסמסטר קראנו חומרים נוספים והשתתפנו בהדרכות וקורסים הנוגעים לתחום התקשורת והתכנון הדיגיטלי על מנת לפתח את האוריינטציה שלנו סביב הפרוייקט שלנו:

- אמיר פרן, המנחה שלנו 'העביר לנו הרצאה בנושא ASIC product flow – מטודולוגיית העבודה בפיתוח שבב תקשורת.
- קראנו את CompTIA Network+ Certification 1'st Chapter of Network fundamentals, וכן את Fujitsu's TCP/IP protocols tutorial על מנת ללמוד את המבואות לפרוטוקלי תקשורת עד שנזכה לקבל סקירה רחבה יותר בקורסים של מגמת התקשורת והסייבר של האוניברסיטה.
- קראנו את פרקים 1-6 של Frank Vahid's digital design הסוקר באופן שיטתי וכללי את התבניות העיצוב הבסיסיות של תכנון דיגיטלי.

- בקיץ ישנם שני קורסים של Intel שבכוונתנו לקחת:
  - Advance Logic Design Concepts – הקורס חופף במעט לפרקים של Frank Vahid's digital design, אך יותר מוכוון לעבודה בשפת ה Verilog וכן מקיף תבניות עיצוב מתקדמות יותר בתכנון דיגיטלי. בנוסף אחד ממרכיבי הקורס החשובים הינו נושא ה- Synchronization.
  - Verification For Logic Designers – מטרת ההדרכה היא לחשוף את המתכננים הדיגיטלים לשיטות ומטודולוגיות ה- Verification & Validation של רכיב דיגיטלי. בין היתר יועבר נושא ה- Formal Verification שהיו כלי חזק מאד להוכחת נכונות לפונקציונליות של רכיב.

אילו כלים למדנו ועוד נלמד?

- על מנת ללמוד פקודות בסיסיות לעבודה בסביבת Unix, מצאנו קורס אינטרנטי ב YouTube אשר השיעורים הראשונים שלו הביאו אותנו לרמת ידע מספקת:
Unix/Linux Tutorial Videos | Mr. Subba Raju

- בנוסף להדרכה שנערכה מטעם ביה"ס להנדסה ומדעי המחשב, קיבלנו הדרכה בהיקף של שעה וחצי בקמפוס אינטל על מערכת ה GIT הפנימית. בנוסף תרגלנו את העבודה עם מערכת ה GIT במשימת "Hello World" שלנו.

- קיבלנו הדרכה בהיקף של שעה בקמפוס אינטל על כלי ה- Eclipse DVT ואת משימת "Hello World" כתבנו ב DVT.

- השתתפנו בקורס מבוא ל Verilog שזו שפת התכנון הלוגי בה נשתמש למימוש. הקורס סקר את הנושאים הבאים: Lexical rules , Module structure , Data types, Structural & behavioral modeling, Generates, Hardware elements, TB, Design guidelines. את משימת "Hello World" כתבנו כמובן בשפת Verliog.

- השתתפנו בהדרכה בהיקף 4 שעות על כלי ה- Verdi שהינו כלי סימולציה ו- Debug לתכנון דיגיטלי. את משימת ה- "Hello World" דיבגנו בכלי זה וכן צפינו בגלים של סימולציה.

- טרם הנחנו את ידינו על כלי ה- Visio ליצירת שרטוטים. כאשר ניגש למשימת התכנון הראשוני וכתיבת מסמך האפיון, ניאלץ להשתמש ב Visio וללמוד אותו.

- על מנת לבדוק שכלי הסינטזה, אשר מבצע Synthesis לתכנון הדיגיטלי שלנו בפרוייקט, אכן בונה את הרכיבים החשמליים באופן שבו תכננו (כל זאת על מנת לוודא תזמון ונדל"ן), ניאלץ ללמוד לקרוא Netlist. Netlist היא השפה המתארת את הרכיבים הלוגים וחיבוריהם לאחר סינטזה. כמו כן, לכלי הסינטזה יכולת למדוד Performance ולכן חשוב שנכיר את הדו"חות שביכולתו להוציא ולקרוא אותם. זהו כלי שלא ידענו בתחילת הסמסטר שנצטרך ללמוד אך כעת אנו יודעים. בשלבים יותר מתקדמים בפרוייקט נוכל להריץ אותו וללמוד אותו.

- בדיקת הרכיבים בפרוייקט מבחינה פונקציונאלית הנם חלק בלתי נפרד מהפרוייקט ועל מנת לעשות זאת נשתמש בכלי בשם Jasper ל- Formal Verification. טרם פגשנו בכלי זה אך בהמשך אנו צפויים לכתוב משימה חדשה – Asynchronous -FIFO אשר בה נתנסה בכלי.

- משימת ה- "Hello World": כתבתנו Full Adder במספר וריאציות. התרגיל אפשר לנו לשים את ידנו לראשונה על שפת ה – Verilog ולהתנסות במערכת ה- GIT של Intel. בנוסף לכתיבה התנסנו בכלי הסימולציה והבדיקה- Verdi, שבאמצעותו הרמנו דיאגרמות גלים של ה- FA ולמדנו על שאר היכולות של הכלי. פירוט מלא של המשימה מופיע ב Mini-Report.

- בקיץ, לאחר הקורס ב Advance Logic Design Concepts, אנו צפויים להתחיל בתרגיל נוסף ומעט מתקדם יותר, שמעבר להתנסות בכלי המימוש הבסיסיים, יפתח אותנו בתחום התכנון וידרוש מאיתנו לאפיין רכיב. התרגיל הוא לתכנן ולכתוב Asynchronous -FIFO. רכיב חומרה שמטרתו לסנכרן שני רכיבים המעבירים מידע דיגיטלי בינהם ועובדים בתדרי שעון שונים. בתרגיל הנ"ל תחילה נרשום מסמך אפיון שבו נפרט את תכנון הרכיב, ולאחר מכן ניגש למימוש. פרט להתנסות במטודולוגיית העבודה, נעשה שימוש בשלושה כלים חדשים: WaveDrom- שבאמצעותו נשרטט דיאגרמות גלים, Visio- שבאמצעותו נשרטט דיאגרמות בלוקים לתכנון, ו- Jasper שבאמצעותו נבצע Formal Verification על הרכיב.

מה הלו"ז להמשך?

תאריך	אורי יפת		עומרי דסה	
	משימה	תעדוף	משימה	תעדוף
יולי 18	קורס בוריפיקציה למתכננים דיגיטלים	P0	קורס בוריפיקציה למתכננים דיגיטלים	P0
אוגוסט 18	קורס בעקרונות תכנון דיגיטלי מתקדמים + משימת Asynchronous-FIFO	P0	קורס בעקרונות תכנון דיגיטלי מתקדמים + משימת Asynchronous - FIFO	P0
ספטמבר 18	חקירת מטודולוגיות Caching ומטודולוגיות Cache evictions + חקירת הבלוקים הסמוכים למעבד.	P0	חקירת מטודולוגיות Translation ומטודולוגיות ארביטרציה + חקירת הבלוקים הסמוכים ליחידת תרגום הכתובות.	P0
אוקטובר 18	כתיבת מסמך אפיון ל-IRAM Cache + התאמת הבלוקים הסמוכים להכנסתו.	P0	כתיבת מסמך אפיון ל Address Translation + התאמת הבלוקים הסמוכים להכנסתו.	P0
דצמבר 18	קידוד Full Mapping – מימוש	P0	קידוד פונקציית תירגום חדשה לבקשה ממעבד יחיד.	P0
ינואר 19	Debug + בדיקת מקום ותזמון ע"י כלי הסינטזה	P0	Debug + בדיקת מקום ותזמון ע"י כלי הסינטזה	P0
פברואר 19	קידוד Partial Mapping - מימוש	P1	קידוד ארביטרציה ושילוב בקשות ממספר מעבדים לפונקציית התרגום.	P1
מרץ 19	Debug	P1	Debug + בדיקת מקום ותזמון ע"י כלי הסינטזה	P1
אפריל 19	Formal Verification	P2	Formal Verification	P2