**Lab1 Preparation Report**

***יהונתן ארמא 207938903***

***יובל סעיד 206921892***

**PART B:**

1. PxDIR – קובע את הכיווניות של הרכיב – Input או Output:
2. Input
3. Output

PxSEL – קובע את פונקציונליות הרגל של הבקר – IO או חומרה אחרת שאפשר לקנפג לרגל:

1. GPIO(Default)
2. Other Hardware

PxIN – כאשר אנחנו במצב של I/O במצב של input אז את ערך ה-input אנו מקבלי מרגיסטר זה:

1. Low Value
2. High Value

PxOUT – כאשר אנחנו במצב של I/O במצב של output אז לרגיסטר זה נכתוב את הערך שאותו נרצה להוציא לרכיב המחובר:

1. Low Value
2. High Value
3. כאשר מבצעים reset:
   1. ראשית ה-PC נטען בכתובת הראשנה של התוכנית.
   2. הרגיסטרים נטענים באופן הבא:
      1. PxSEL -reset with PUC
      2. PxDIR – reset with PUC
      3. PxOUT – Untouched (כלומר נשאר מה שהיה הערך לפני ה-reset).
      4. PxIN – אינו נקבע על ידי הבקר ולכן לא רלוונטי.

נבחין ש- PUC – Power Up Clear.

כלומר reset with PUC אומר שבזמן reset הערך מתאפס ל-0.

1. נרצה לכתוב ל- P9SEL את הערך כדי שכל הכניסות יהיו במצב I/O, ובנוסף נרצה ב-P9DIR שהכניסות הזוגיות יהיו ב-‘1’ (Output) והאי זוגיות ב-'0' (Input), לכן נכתוב:

BIC.B #0xFF, &P9SEL

MOV.B #0xAA, &P9DIR

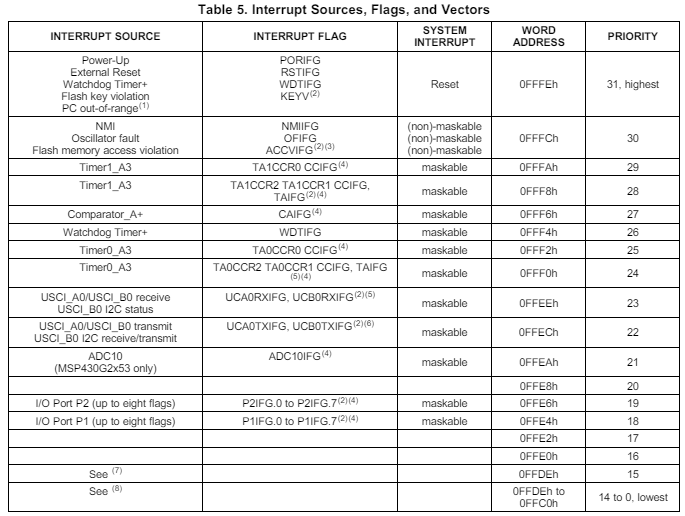
1. אם ה-DudyCycle הוא 50% אז נרצה שהערך יהיה '1' למשך 0.5ms, בנוסף תדר ה-MCLK הוא 1MHz ולכן:
2. פסיקה היא אות חשמלי המתקבל במעבד על מנת להודיע לו על פעולות אסינכרויות שהתבצעו.

כלומר, ברגע שהמעבד שלנו מקבל – Interrupt מרגיסטר שאנחנו מתייחסים אליו (כלומר מקונפג כמו שצריך לאפשרות לפסיקה ואנחנו לא מתעלמים ממנו), אז המעבד שלנו 'קופץ' לפונקציה ייעודית לפסיקה ומבצע אותה, לאחר ביצוע הפונקציה, המעבד יחזור לשורה ממנה הוא קפץ. אנו צריכים פסיקות מכיוון שיש לנו חומרות או אינפוטים לבקר שלנו שמבצעים פעולות במקביל לבקר, ואנחנו נרצה להודיע לעדכן את המעבד שלנו בצורה אסינכרונית על דברים אלו, וזו הדרך לעשות זאת. בנוסף, לעיתים יש צורך לתעדף פעולות חשובות יותר או פחות, ואנו נעזרים בפסיקות על מנת לבצע את התיעדוף הזה. לדוגמה לחיצה על מקלדת או עכבר, או אפילו לחיצה על כפתור ה-reset, דורשים מאיתנו לבצע -interrupt כי אלו פעולות חשובות שאנו רוצים שהמעבד יטפל בהם עכשיו, ו'יעזוב' את מה שהוא עושה.

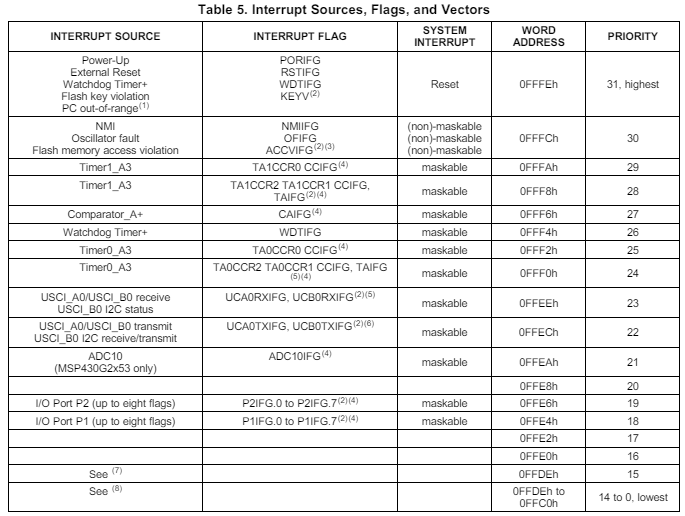
1. נזכור שבזמן תשאול, המעבד 'שורף' אנרגיה וזמן על בדיקה אם תנאי מסוים התקיים ובזמן זה אינו יכול לבצע שום דבר במקביל.

אם אנחנו נשתמש בפסיקה ונבקש באופן אקטיבי מהרכיב ש'יודיע' לנו בעזרת פסיקה שהתנאי מתקיים, אז המעבד יוכל לבצע מה שהוא רוצה בזמן הזה (ואולי אפילו לישון ולחסוך אנרגיה), וכאשר התנאי מתקיים אז הוא יבצע את הפעולה שהוא נדרש, מבלי לבזבז זמן ואנרגיה על בדיקת התנאי כל הזמן.

שילוב בין השניים יתקיים לדוגמה כאשר קבלנו כמה פסיקות ואנחנו נבצע תשאול לגבי איזה תנאים התקיימו.

1. ראשית נבחין בשני סוגים של פסיקות:
   1. לא ניתנות למיסוך – פסיקות בעדיפות גבוהה שלא ניתן לקנפג אותן להיות ממוסכות:

פסיקות אלו הן הפסיקות בעדיפות הכי גבוהה ולכן המעבד עוזב הכל ומטפל בהן קודם. דוגמה – Reset.

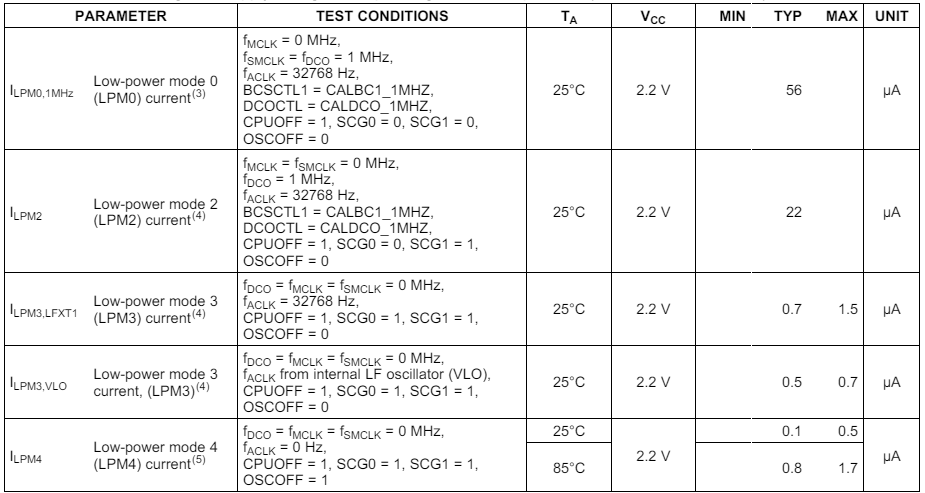
* 1. ניתנות למיסוך – פסיקות שנגרמות מגורמים חיצוניים, וניתן למסך אותן באופן לוקלי או גלובלי.

כאלו יש שלושה סוגים:

* + 1. פסיקה חיצונית - פסיקה הנגרמת על ידי רכיב חומרה חיצוני כמו מקלדת או עכבר.
    2. פסיקה פנימית – פסיקה הנוצרת על ידי החומרה או התוכנה בעקבות אירוע פנימי של הבקר.
    3. פסיקת תוכנה – פסיקה המתעוררת בעקבות הדלקת דגל מהתוכנה.

1. נבחין שבמצב הרגיל המעבד שלנו מבצע את הפקודות ב-flash באופן סריאלי מצב זה נקרא , המצב האקטיבי של הבקר.

לעיתים התוכנית שלנו מאופיינת בכך שיש בה חלקים שבה נרצה שהבקר 'יחכה' או יחסוך באנרגיה. לטובת זה יש דגלים תחת השמות – CPUOFF, OSCOFF, SCG0, SCG1, שבעזרתם נוכל לקבוע באיזה מצב חיסכון באנרגיה נהיה:

תיאור:

1. נרצה להגדיר את הדברים הבאים:
   1. נרצה שהרגל תהיה מסוג I/O ולכן נקבע את רגל 0 של P2SEL ל-0.
   2. נרצה שהרגל תיתן לנו פסיקה בירידת מתח ולכן נגדיר אותה כ-Input , ולכן נקבע את רגל 0 של P2DIR ל-0.
   3. נרצה שהפסיקה תתקבל מ-PullUp, ולכן נקבע את רגל 0 של P2IES ל-1.
   4. נרצה לאפשר פסיקה מרגל זו, ולכן נקבע את רגל 0 של P2IE ל-1.
   5. נוריד את דגל הפסיקה (נאפס את הפסיקה), ולכן נאפס את רגל 0 של P2IFG.

ולכן אלו הפקודות שנבצע:

BIC.B #0x01, &P2SEL

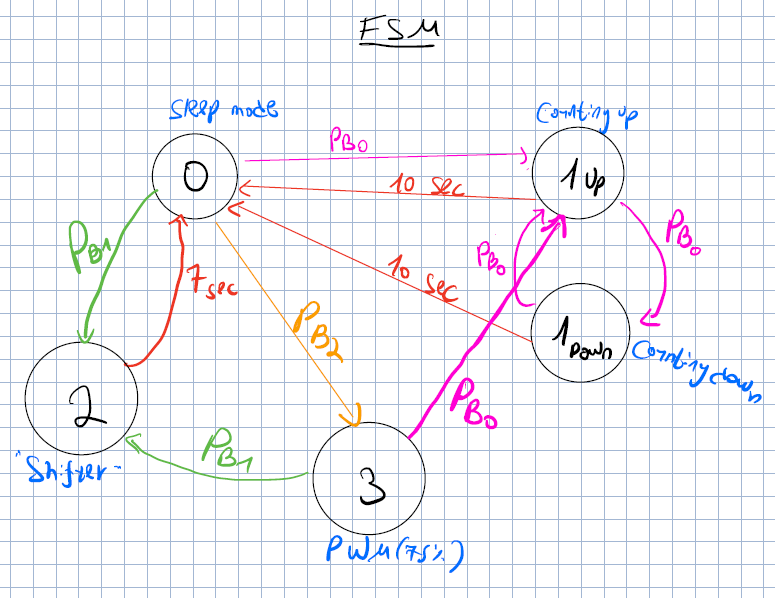
BIC.B #0x01, &P2DIR

BIS.B #0x01, &P2IES

BIS.B #0x01, &P2IE

BIC.B #0x01, &P2IFG

1. נעמוד על ההבדלים:
   1. Event Driven – תכנות המבוסס פרדיגמה מסוג זה הוא תכנות שמבסס את התקשורת בין רכיבים שונים, בינם או מול המעבד מתבצעת ברמת התוכנה שהיא רמה גבוהה. כלומר, הטיפול בפעולות אסינכרוניות מתבצע בעיקר בעזרת Events, שנשמרים בבאפרים או בתורים.
   2. Interrupt Driven – תכנות המבוסס פרדיכמה מסוג זה הוא תכנות המבסס את התקשורת בין רכיבים שונים, בינם או מול המעבד בעיקר בעזרת Interrupts, שזה כפי שאנחנו יודעים מתבצע ברמת החומרה. תכנות זה לדוגמה מתאים כאשר אנחנו מתכנתים מערכת שמתוארת בעזרת מכונה מצבים וה-Interrupts מגיעים ו-'משנים' את מצב התוכנית.
   3. Blocking – פרדיגמה זו היא פרדיגמת תכנות בה הקוד מתבצע בצורה טורית ולא ניתן למקבל את התוכנית כלומר שורת הקוד הקודמת 'חוסמת' את שורת הקוד הבאה, ואם צריך לעמוד בתנאי מסויים בשביל להמשיך בתוכנית את המצב הזה 'חוסם' את המשך התוכנית.
   4. Non-Blocking – פרדיגמה זו היא פרדיגמת תכנות שבה הקוד כתוב בצורה כזו שהוא לא 'חוסם' את המשך התוכנית, כלומר אם הפעולה לא מתבצעת עד הסוף, זה לא מונע מהמשך התוכנית להתבצע, בצורה זו ניתן 'למקבל' את התוכנית.

FSM