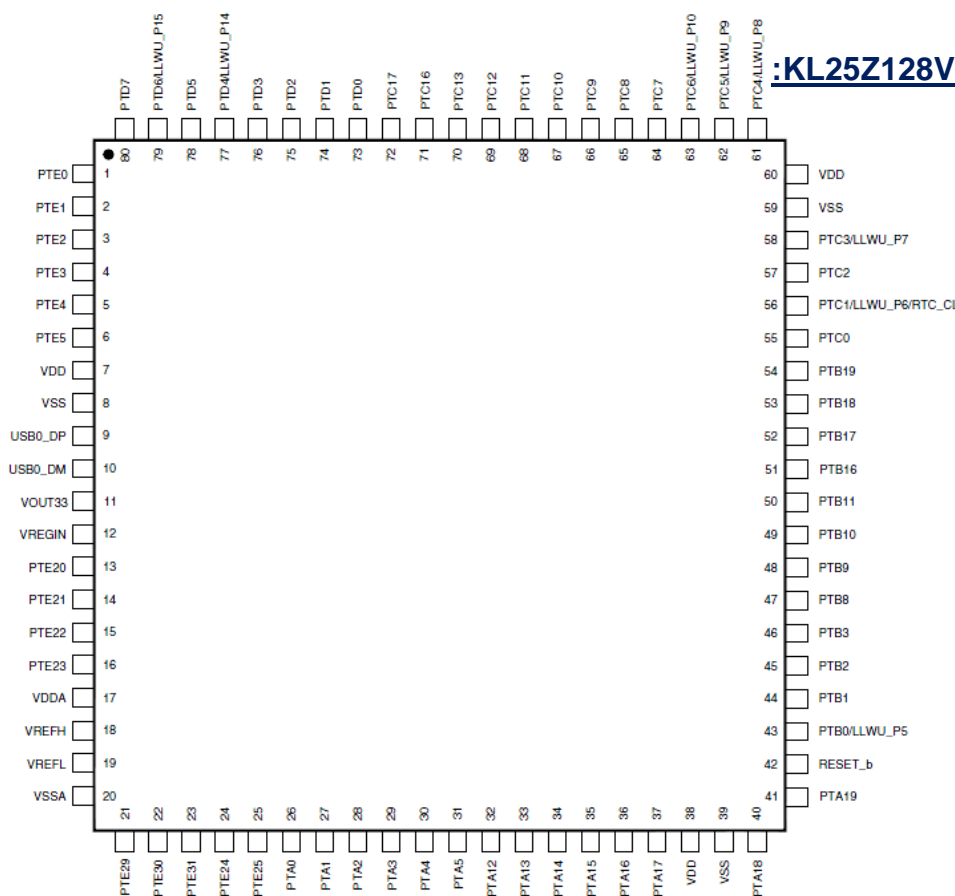


תוכן עניינים

| | |
|----|---|
| 2 | A. דיאגרמת רגלי הבקר KL25Z128VLK4: |
| 4 | B. צירוף קבצים לפרויקט: |
| 6 | C. הידור ובניית הפרויקט (Building Project): |
| 6 | D. מצב Debug: |
| 8 | E. מצב Active Application: |
| 9 | F. הכנה לניסוי: |
| 9 | G. שאלות הכנה תיאורטיות: |
| 10 | H. הקדמה ודרישות מקצועיות מחייבות לקראת ביצוע שלב החלק המעשי: |
| 11 | I. סיווג ארכיטקטורת תכנות FSM לשני סוגים: |
| 12 | J. חלק מעשי נדרש לביצוע – כתיבת קוד מערכת בשפת C : |
| 13 | K. צורת הגשה דוח מכין: |
| 13 | L. צורת הגשה דוח מסכם: |

DEBUG and PROGRAMMING – CodeWarrior IDE

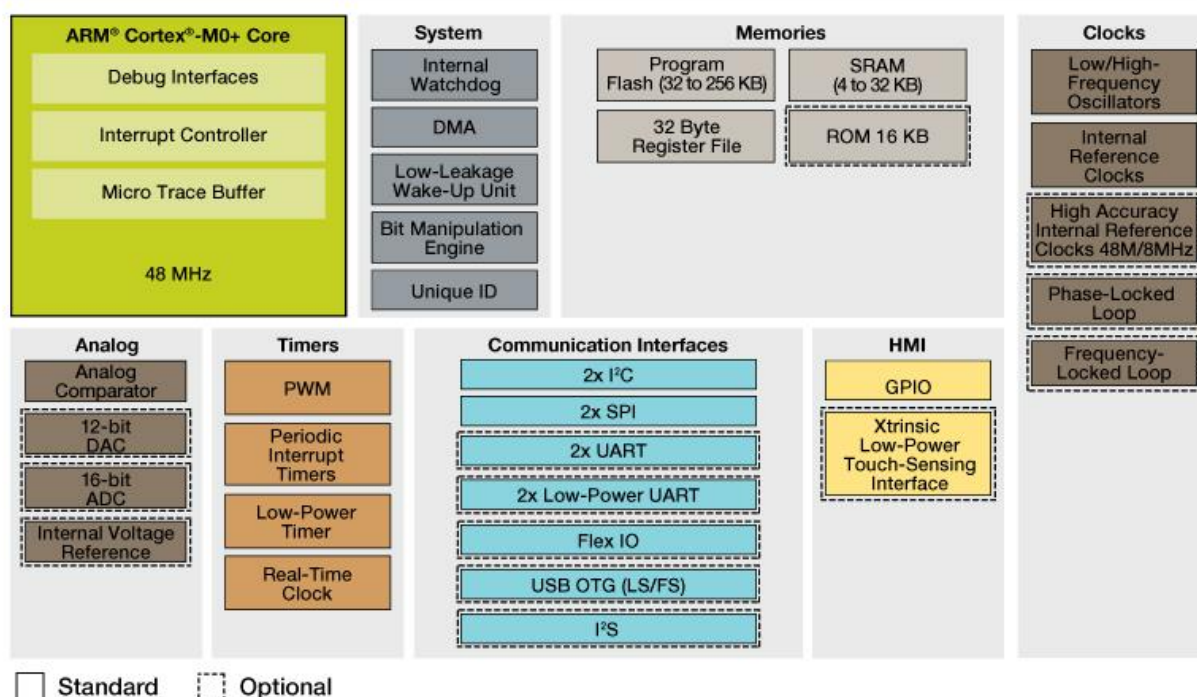
A. דיאגרמת רגלי הבקר **KL25Z128VLK4**:



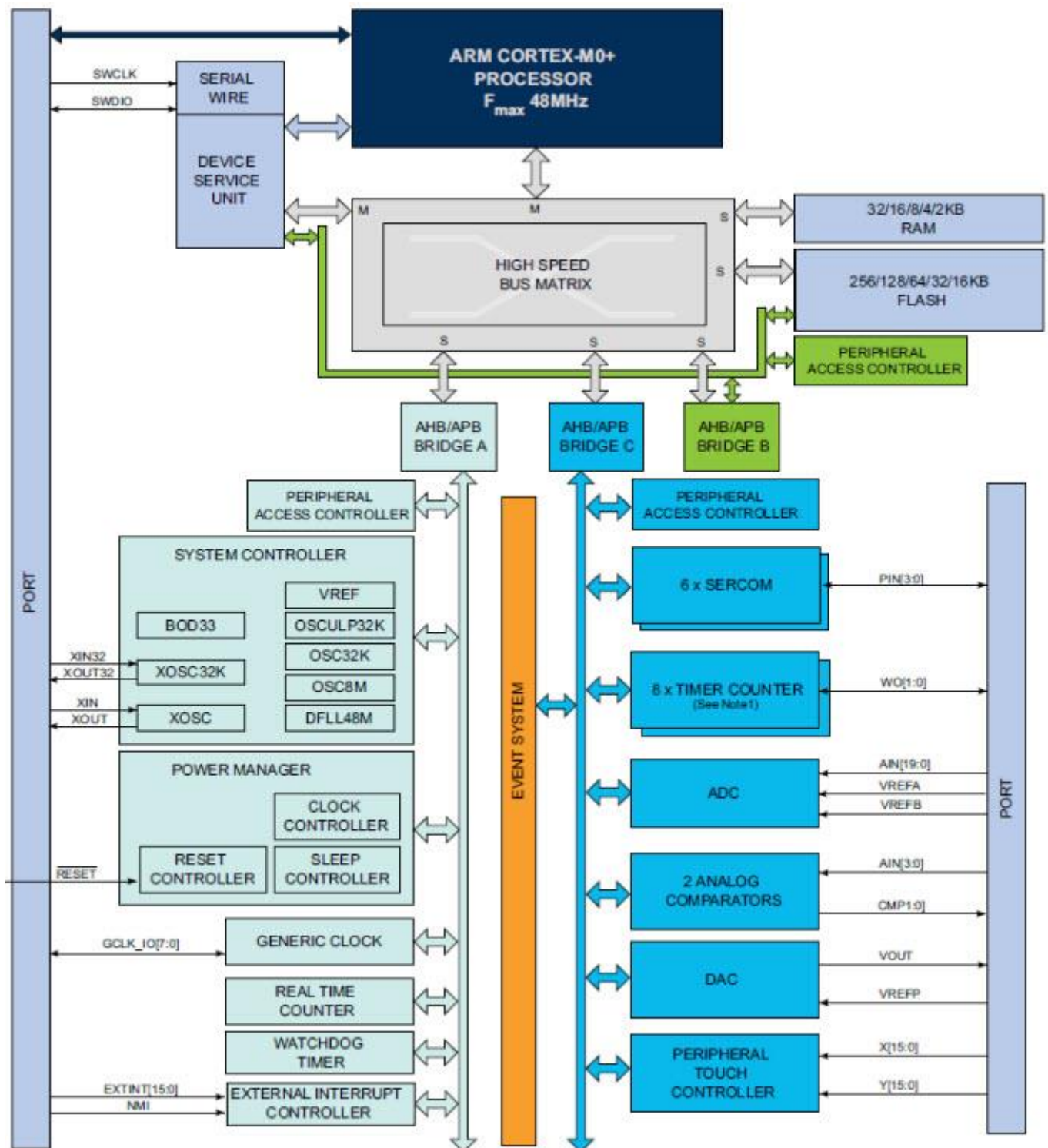
KL25 80-pin LQFP pinout diagram

Signal Multiplexing and Pin Assignments find in “KL25 Sub-Family Data Sheet” page 46-53 ♦

Kinetis KL2x MCU Family Block Diagram



:Cortex M0+ Block Diagram ◆



B. צירוף קבצים לפרויקט:

לאחר שפתחנו פרויקט (כמתואר בניסוי Lab2preface) נרצה לצרף קבצים נוספים (התומכים במשימה לדוגמה המתוארת בסעיף H) מעבר לקבצים הבסיסיים הנוצרים בשלב פתיחת הפרויקט. להלן 3 השלבים:

1. בכל תיקיית פרויקט שנפתחה (במקום שייעדנו לה מבעוד מועד) ישנן 4 תיקיות. תיקייה אחת בשם הפרויקט שבחרנו ו-3 תיקיות עזר נוספות של סביבת הפיתוח.

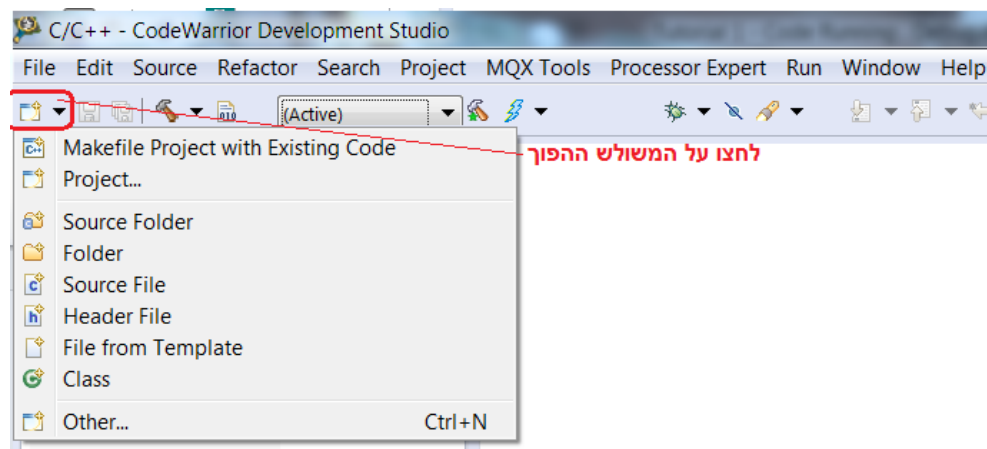
| Name | Date modified |
|-------------------------------------|------------------|
| 3 תיקיות עזר של סביבת הפיתוח | |
| .metadata | 10/02/2015 10:49 |
| RemoteLaunch | 10/02/2015 10:49 |
| RemoteSystemsTempFiles | 10/02/2015 10:49 |
| TEST | 10/02/2015 11:26 |
| תיקייה בשם הפרויקט שפתחנו | |

בתוך תיקיית **TEST** (שם הפרויקט) ישנן תיקיות וקבצים הנבנים ע"י סביבת הפיתוח באופן אוטומטי ומכילות קבצי מקור (Source Files) וקבצי כותר (Header Files) **בסיסיים** המתאימים לשלב הבקר שבחרנו.

- ♦ את קובצי המקור שנרצה להוסיף לביצוע הפרויקט שלנו נוסיף לתיקיית **Sources**
- ♦ את קובצי המקור שנרצה להוסיף לביצוע הפרויקט שלנו נוסיף לתיקיית **Project_Headers**

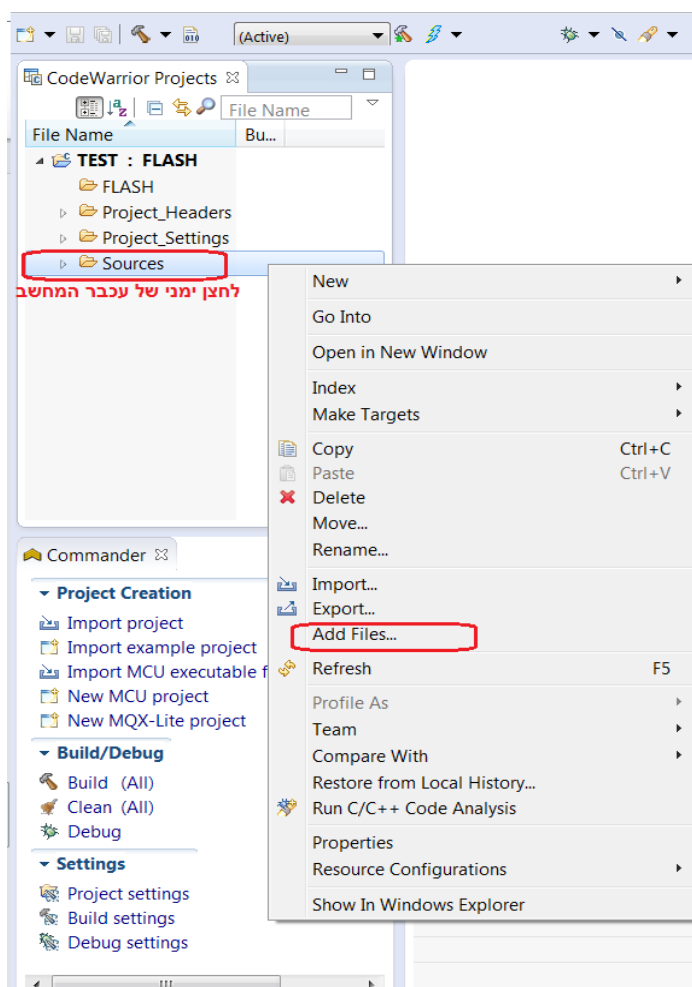
| | |
|--------------------------|------------------|
| .settings | 10/02/2015 11:26 |
| FLASH | 10/02/2015 11:26 |
| Project_Headers | 10/02/2015 11:26 |
| Project_Settings | 10/02/2015 11:26 |
| Sources | 10/02/2015 11:26 |
| .cproject | 10/02/2015 11:26 |
| .cwGeneratedFileSetLog | 10/02/2015 11:26 |
| .project | 10/02/2015 11:26 |
| ReferencedRSESystems.xml | 10/02/2015 11:26 |

2. קובצי source , header (ודברים נוספים) חדשים נוכל לפתוח בצורה הבאה.

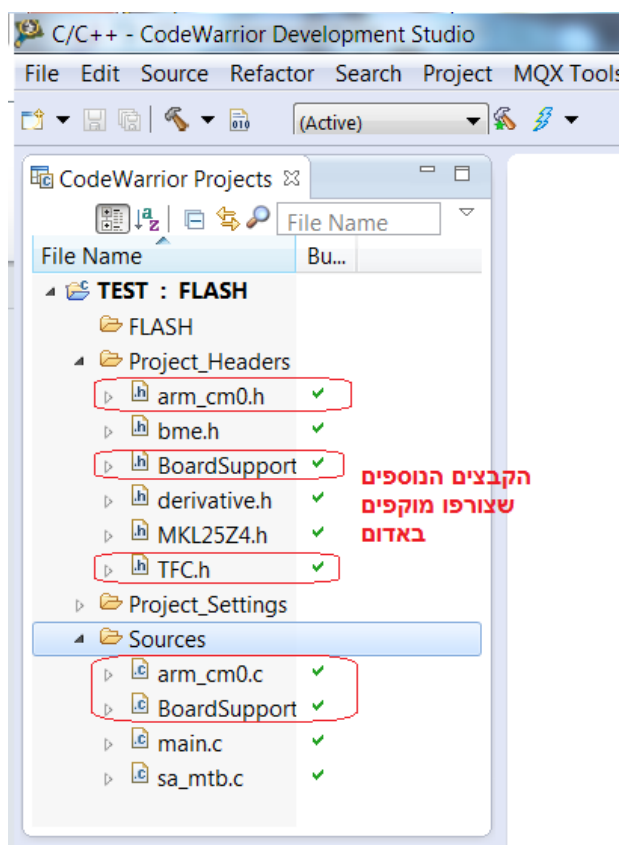


כמובן שיש לצרפם לאחר מכן לתיקיות המתאימות שבסעיף קודם **באחת** מ-2 הצורות הבאות.

- ♦ לפי ההסבר בסעיף קודם. לאחר מכן יש ללחוץ **File → Refresh** (או F5) כדי לצרפם לפרויקט.
- ♦ לחצן ימני על התיקייה המתאימה תחת שם הפרויקט בחלון סביבת הפיתוח (ראה תצלום הבא).

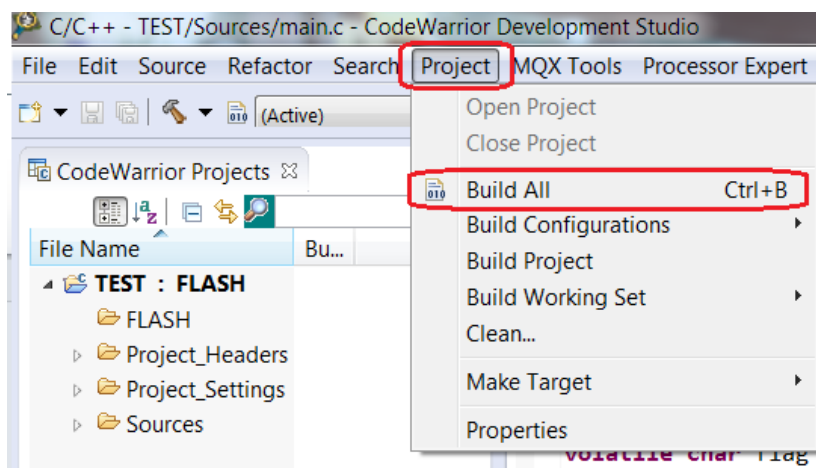


3. בקובץ זה נתרגל שימוש בשפת C. לשם כך יש לצרף לפרויקט שפתחתם בשלב זה את הקבצים המיועדים לתיקיות Sources ו-Project_Headers. יש לבדוק שהקבצים צורפו לפרויקט.



C. הידור ובניית הפרויקט (Building Project):

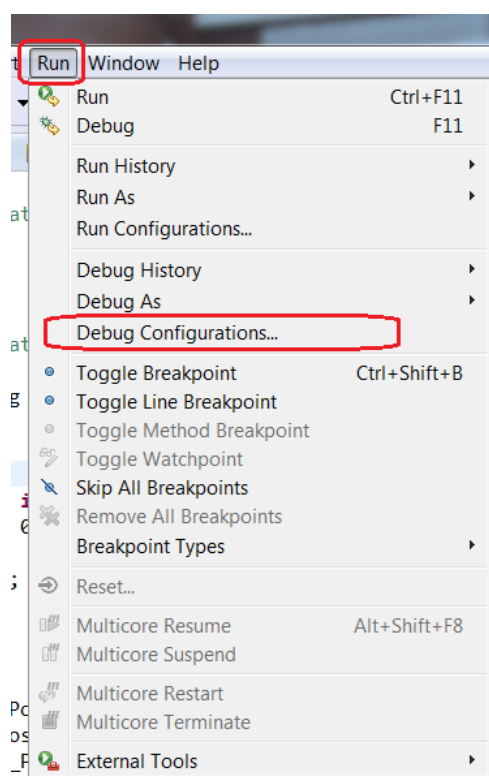
לאחר שפתחתם פרויקט וצירפתם את הקבצים המתאימים לתוך תיקיות **Sources** ו- **Project_Headers**, הקבצים נמצאים בתיקיית Lab2 (הסבר לכך נמצא בקובץ פתיחת פרויקט). בשלב זה נבצע הידור ובניית הפרויקט.



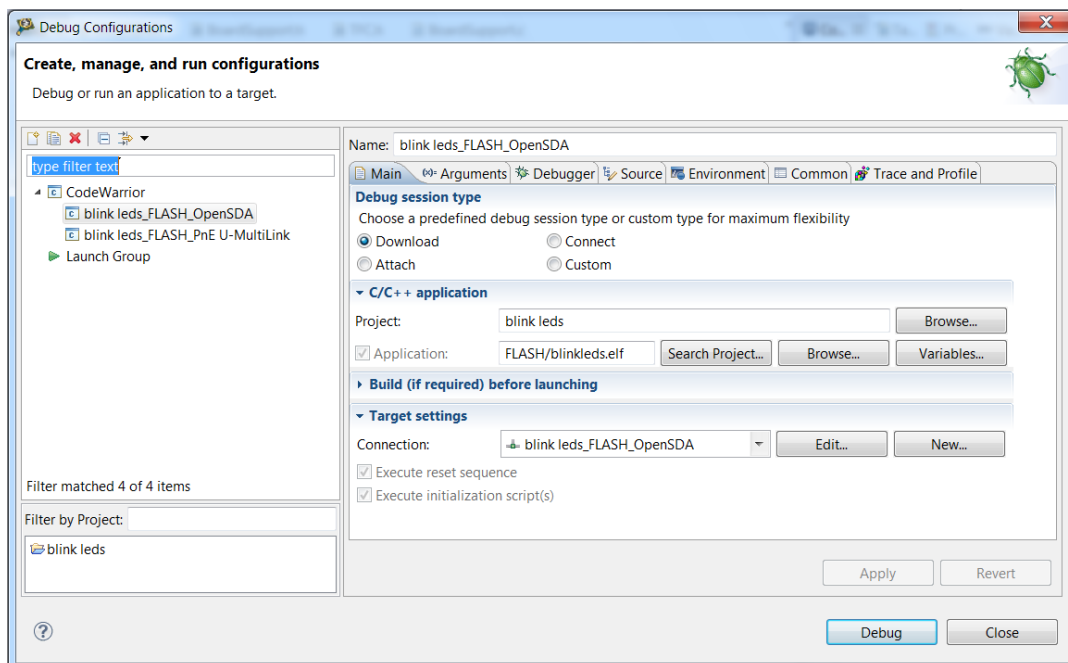
D. מצב Debug:

במצב DEBUG (צריבת קוד לזיכרון ה-FLASH של הבקר והרצתו בבקר בלבד עם יכולות DEBUG דרך ה-PC) באפשרותנו להריץ את הקוד במצב אמת כאשר הקוד רץ בבקר ולא ב-PC, אולם השליטה על הרצת הקוד (נקודות עצירה, הרצה בצעדים וכו') היא דרך ה-PC.


1. לשינוי הגדרות ברירת מחדל של ה-Debugger:

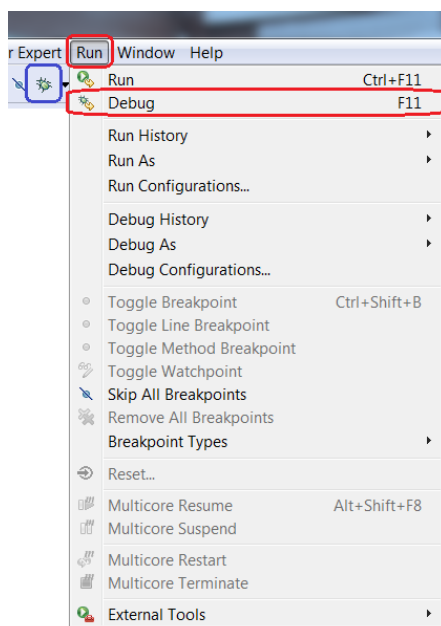


זוהי הגדרת ברירת המחדל.

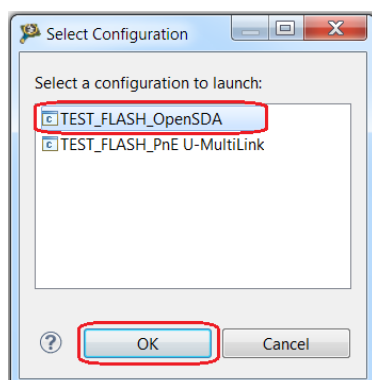


2. מצב DEBUG:

Run → Debug **OR** F11 **OR** push on 



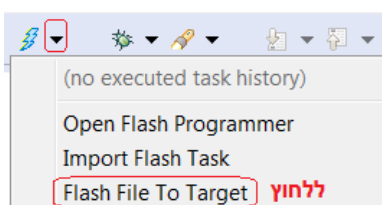
3. בחלון שנפתח יש לפעול כמו בצילום הבא.



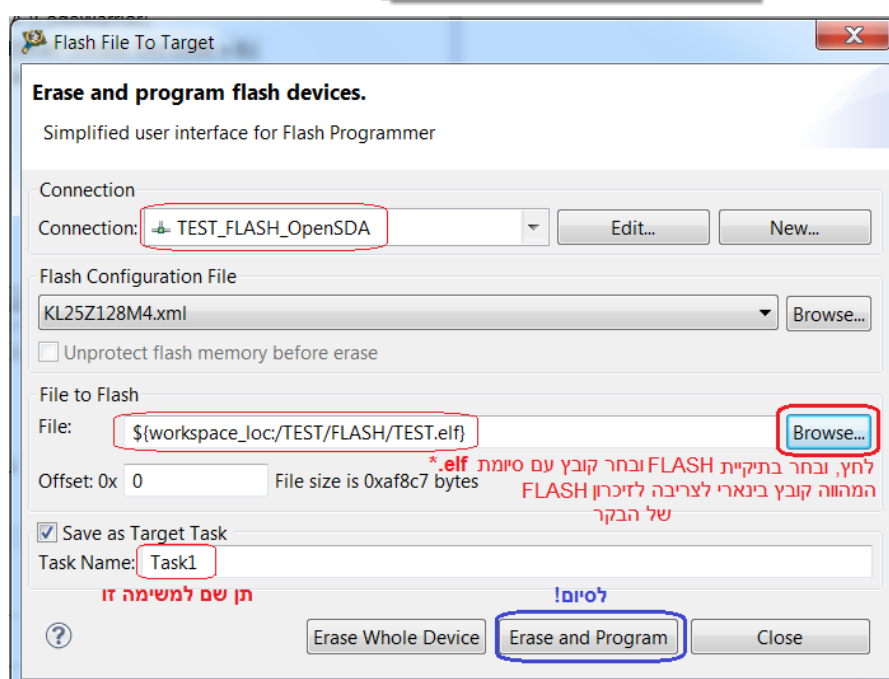
4. נכנסתם למצב DEBUG, אתם יכולים לבצע הרצה של הקוד ולבחון את פעולת הבקר (כמו שלמדתם במעבדת "מבוא למחשבים").


E. מצב Active Application:

בפעם הראשונה שתבצעו מצב זה תוכלו להגדיר משימה בשם המקשרת בין הקובץ הבינארי המהווה את התוכנית אשר ברצונכם לצרוב לזיכרון הבקר, כך בכל לחיצה על סמל הברק תתבצע צריבה. יש ללחוץ על המשולש הפוך מימין לברק ויפתח לכם החלון הבא:



נפתח החלון הבא:



מעתה ואילך בכל פעם שתלחצו על סמל  תתבצע צריבה של התוכנית לזיכרון FLASH של הבקר. במצב זה הקוד ייצרב על הבקר ויפעל במנותק מה- Host. לאחר צריבה יש ללחוץ על כפתור RESET כדי שרגיסטר PC ייטען בכתובת תחילת התוכנית הראשית.

F. הכנה לניסוי:

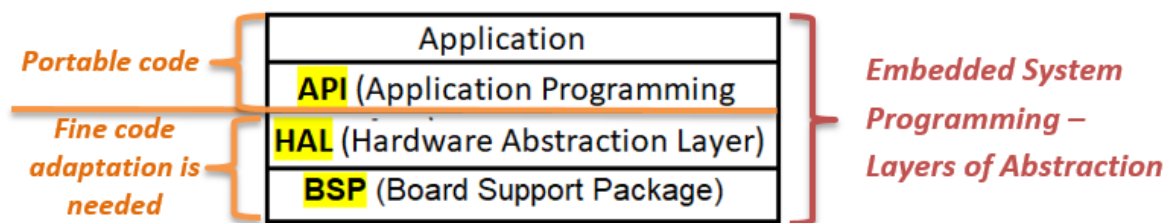
- במשימת מעבדה זו נתרגל את הנושאים **GPIO, I/O Interrupt, LPM**.
- ♦ בקובץ כרטיס הבקר "**KL25 Sub-Family User Manual**" לקרוא את הפרקים 10, 13, 3 (pages 51-54), 11, 41 הרלוונטיים בספר
 - ♦ לקרוא עמודים 35-36 בספר "**Cortex M0+ Generic User Guide**"
 - ♦ להפעיל את הקוד לדוגמא במצב **DEBUG** ולאחר מכן במצב **Active Application** ולהבין אותו.

G. שאלות הכנה תיאורטיות:

1. הסבר מהו **RGB led** ומה יתרונו על לד רגיל.
2. רשום טבלה המקשרת בין מס' בינארי בין 0x0 ל- 0x7 לבין הצבע המתאים.
3. הסבירו מדוע ישנו צורך ברגיסטרים **GPIOx_PSOR, GPIOx_PCOR, GPIOx_PTOR** כאשר ניתן לבצע את אותן הפעולות ע"י שימוש ברגיסטר **GPIOx_PDOR**.
4. מנה והסבר את ההבדלים בין הפונקציה **RGB_Serial_Show(void)** ולבין **RGB_LED_OFF** (רמז: חלק מהתשובות טמונות ב **disassembly**).
5. הסבר את השימוש במשתנים מסוג **volatile, const, static, global**
6. הסבר כיצד המעבד יודע להבדיל על חזרה מפונקציה רגילה ולבין חזרה מ- **handler** של פסיקה (רמז: הבחינו בהתנהגות מרגיסטר **LR**).
7. כיצד מתבצע **debouncing** עבור לחצן המחובר ל- **PTD7** בחומרה? בתוכנה? האם לולאת ה **debounce** ב- **handler** של **PORTD** מיותרת?
8. הסבר את הצורך והשימוש בבקר הפסיקות **NVIC** (Nested Vectored Interrupt Controller) ורשום דוגמה לשימוש בקוד לדוגמה.

H. הקדמה ודרישות מקצועיות מחייבות לקראת ביצוע שלב החלק המעשי:

1. הקוד למימוש המערכת נדרש להיות בארכיטקטורת תוכנה FSM ומבוסס Interrupt Driven, כלומר טריגר למעבר בין מצבים במערכת נעשה כתוצאה מבקשות פסיקה (ראו Tutorial 4 pages 11-13) **ולא תחת מעטפת של לולאה אינסופית (הגורמת לבזבז הספק ובנוסף מוגבלת מבחינת ארכיטקטורת התוכנה בהכללה ותחזוקה של קוד התוכנית) אלא בשימוש מצבי שינה של המעבד.**
הערה: גישה זו במערכת הכתובה על גבי מערכת הפעלה נקראת Event Driven.
2. **נדרש לארגן את הקוד בצורה מסודרת בקבצים נפרדים לצורך חלוקת קוד המערכת לשכבות הבאות:**
(תזכורת: הסבר טכני כיצד לבצע חלוקת קוד לקבצים נפרדים מופיעה במודל תחת לשונית LAB2)
 ✓ שכבת (Board Support Package) **BSP** מכילה קוד לקנפוג רגיסטרים של רכיבים פריפריאליים של הבקר (בניסוי מעבדה זו מדובר על קינפוג לדים, מתגים ולחצנים). **שם הקבצים בשכבה זו יהיו עם קידומת bsp, למשל bsp_example.c**
 ✓ שכבת ה- (Hardware Abstraction Layer) **HAL** מכילה רוטינות הדרייברים של המערכת המנהלות את הממשק עם הרכיבים הפריפריאליים של המערכת באופן ישיר (בניסוי מעבדה זו מדובר על רוטינה לכתיבת ערך כארגומנט למערך הלדים, רוטינה המחזירה ערך קריאה ממערך המתגים, רוטינת ISR של בקשת פסיקה ממערך הלחצנים). **שם הקבצים בשכבה זו יהיו עם קידומת hal, למשל hal_example.c**
 ✓ שכבת ה- (Application Programming Interface) **API** מכילה רוטינות על בסיסן אנו כותבים את האפליקציה של המערכת ב High Level תוך גישה לרכיבים פריפריאליים דרך API בלבד כאשר המימוש של השכבות מטה "שקוף" לשכבה זו, קוד זה צריך להיות portable כך שהוא יהיה תקף גם במידה וה-MCU של המערכת יתחלף באחר. **שם הקבצים בשכבה זו יהיו עם קידומת api, למשל api_example.c**
 ✓ שכבת ה- Application מכילה רוטינות שירות High level כגון חיפוש איבר במערך, מיון וכו' ומכילה את קוד ה- main היא שכבת קוד הגבוהה ביותר בה מתקיים הממשק עם המשתמש (מכילה את קוד מעטפת ה- FSM של המערכת). **שם הקבצים של שכבה זו הם main.c, app_func.c**



הערה:

בכתיבת קוד גנרי המחולק לשכבות נוכל לבצע העברה קלה בין מערכת הכתובה עבור משפחה MSP430x4xx למערכת הכתובה עבור משפחה MSP430x2xx ולהיפך, ניהול המעבר מתחלק לשני מקרים:

- i. במקרה של מעבר בין משפחות של אותו שבב הבקר כאשר שתיהן מכילות את המודולים הפריפריאליים בשימוש המערכת אזי נדרש רק לעדכן את קובץ ה-BSP.
- ii. במקרה שמשפחה אחת חסרה לפחות מודול פריפריאלי אחד הנמצא בשימוש המערכת אזי נדרש לעדכן את קובצי שכבות ה-BSP וה-HAL.
3. העיקרון המרכזי בארכיטקטורת תוכנה FSM המבוססת Interrupt Driven – בקוד ה-ISR של מקור בקשת הפסיקה (במעבדה זו בקשת פסיקה קוראת בלחיצה על אחד מהלחצנים) אנו מקבלים החלטה לעדכון ערך משתנה המצב **state** בלבד, בצורה זו אנו מעבירים מידע משכבת ה-HAL (המגיע אליה מהשכבה הפיזית = שכבת החומרה) היישר לשכבת ה-Application.
4. **הארה חשובה:** החל מניסוי 2 ואילך, השהיות בקוד המערכת יהיו בשימוש טיימרים בלבד (חוץ מהשהיות נקודתיות שיוגדרו כיוצאי דופן) ולא כפי שנעשה בניסוי מעבדה 1 בשימוש לולאות for ל"שריפת" מחזורי שעון מעבד (הנקראת תשאול, polling).

1. סיווג ארכיטקטורת תכנות FSM לשני סוגים:

ארכיטקטורת תכנות FSM מחולקת לשני סוגים הבאים:

1. מערכת Simple FSM :

מעבר ממצב נוכחי (current_state) למצב הבא (next_state) אפשרי רק לאחר סיום קטע הקוד (המשימה) של המצב הנוכחי. כדי לתמוך בכך יש צורך להגדיר את קטע הקוד הנדרש להיות אטומי במצב הנוכחי כ critical section כדי שאף פסיקה לא תוכל "לחתוך" את קטע הקוד הזה, כלומר בתחילת קטע הקוד של המצב למסך גלובאלית את הפסיקות (GIE=0) ובסיום לאפשר אותן (GIE=1).

2. מערכת Advanced FSM :

מעבר ממצב נוכחי (current_state) למצב הבא (next_state) אפשרי גם במהלך ביצוע המצב הנוכחי במידה והמצב הבא הוא ברמת עדיפות גבוהה יותר.
כדי לתמוך בכך יש צורך בהגדרת מבני הנתונים הבאים:

- i. משיקולי ביצועים תחת משטר Real Time את רמת העדיפות של המצבים נגדיר ע"י קידוד המצבים ברמת עדיפות עולה (מצב idle=0) ככל שערך קידוד עולה כך רמת העדיפות גבוהה יותר. המשמעות, בכניסה ל-ISR עקב בקשת פסיקה, מעבר למצב הבא יתבצע רק אם ערך המצב הבא גדול מערך המצב הנוכחי.
- ii. במקרה שבו המצב הבא "חותך" את המצב הנוכחי במהלך ביצוע המצב הנוכחי עלינו לנהל זאת כך שבסיום, ביצוע המצב הנוכחי ימשיך מהמקום (ערך PC) וערכי ה-context (ערך data החיוני) בו "נחתך".
לצורך כך נגדיר שני מערכי נתונים:
 - ✓ למצב i נגדיר את מערך $context_i$ בגודל מוגדר מראש המוקצים עבור שמירת ה-context של הרגיסטרים הפריפריאליים הנדרש בכל אחד מהמצבים. למשל, בעבודה הכוללת הדפסה למערך הLEDים נצטרך לשמור את ה context במבנה המכיל את $LEDs\ value, loop\ delay\ value$.

במצב זה תוכן מבנה $context_i$ מכיל שלוש שדות ונכתוב לתוכו א נקרא מתוכו את הערכים בכל $context\ switch$.

✓ נגדיר מערך למימוש מבנה נתונים של מחסנית למימוש תור של ביצוע מצבים $exeQue$. מצב "שנחתך" יכנס לתור בשיטת LIFO לצורך המשך ביצוע.

להלן סיכום סדר הפעולות לביצוע:

- i. בכניסה ל ISR עקב בקשת פסיקה, מעבר למצב הבא j יתבצע רק אם ערך המצב הבא j גדול מערך המצב הנוכחי i. במקרה זה, נבצע שמירת ה context של המצב הנוכחי (כתיבה למערך $context_i$), ביצוע push (לא להתבלבל עם פקודת אסמבלי push של ה stack) של ערך המצב הנוכחי לתור $exeQue$ ולבסוף עדכון משתנה המצב $state$ לערך המצב הבא j.
- ii. בסיום ביצוע מצב j ביצוע מצב i צריך להמשיך מהיכן "שנחתך" לכן נבצע את השלבים בצורה הפוכה, ביצוע pop (לא להתבלבל עם פקודת אסמבלי pop של ה stack) מהתור $exeQue$ לתוך משתנה המצב $state$ וטעינת תוכן $context_i$ ל Register File.
- iii. פעולת שלב ii תימשך עד לריקון התור $exeQue$.

ג. חלק מעשי נדרש לביצוע – כתיבת קוד מערכת בשפת C :

ארכיטקטורת התוכנה של המערכת נדרשת להיות מבוססת *Simple FSM* (ראה הסבר בסעיף I) המבצעת אחת מתוך ארבע פעולות בהינתן בקשת פסיקה חיצונית של לחיצת לחצן מתוך ארבעת הלחצנים PB0, PB1, PB2, PB3 המחוברים לארבעת רגלי הבקר P2.0 – P2.3 (P2 is connected to PTD0 - PTD7), את מערך הLEDs נחבר ל- P1 (P1 is connected to PTB0 – PTB3, PTB8 – PTB11). בתחילת התוכנית, הבקר נמצא במצב שינה. קוד התוכנית נדרש להיות מחולק לשכבות (כמתואר בסעיף H). טרם שלב כתיבת הקוד נדרש לשרטט גרף דיאגרמת FSM מפורטת של ארכיטקטורת התוכנה של המערכת ולצרפה לדו"ח מכין. המצבים אלו הצמתים והקשתות אלו המעברים ממצב למצב בגין בקשות פסיקה.

- בלחיצה על לחצן PB0 (state=1):

בהגדרה מראש של מערך ב main באורך 9 המכיל ספרות ת"ז של סטודנט. יש להציג על גבי 8-bit LEDs array את ספרות ת"ז ספרה אחר ספרה עם השהיית ביניים של 0.5sec.

הערה: מצב אחר אינו ראוי "לחתוך" מצב זה טרם השלמת הביצוע המוגדר של המצב

- בלחיצה על לחצן PB1 (state=2):

נדרש להדליק לד בודד בדילוגים מימין לשמאל עם השהיה בין ערכי הספירה של 0.5sec. משך זמן הפעולה יהיה 7 שניות (תוך שמירת ערך הכתיבה ללדים בחלוף הזמן, כך שבביצוע הבא של המצב הלד ימשיך לדלג מהיכן שהפסיק).

הערה: מצב אחר אינו ראוי "לחתוך" מצב זה טרם השלמת הביצוע המוגדר של המצב

- בלחיצה על לחצן PB2 (state=3):

התוכנית מפיקה אות PWM במוצא רגל P2.7 בתדר 4kHz עם DutyCycle=75% (ברזולוציה מקסימאלית – ודאו זאת בעזרת שימוש ב-scope).

הערה: מצב אחר רשאי "לחתוך" מצב זה (מאחר ופעולתו היא אינסופית) ובך לסיים אותו

- (state=idle=0):

הבקר מכבה את הלדים וחוזר למצב שינה (Sleep Mode).

K. צורת הגשה דוח מכין:

- הגשת מטלת דוח מכין תיעשה ע"י העלאה למודל של תיקיית zip מהצורה **id1_id2.zip** (כאשר $id1 < id2$), רק הסטודנט עם הת"ז id1 מעלה את הקבצים למודל.
- התיקייה תכיל את שני הפרטים הבאים בלבד:
 - ✓ קובץ **Preface_lab2.pdf** – מכיל תשובות לחלק ההקדמה של ניסוי LAB2
 - ✓ קובץ **Preparation_lab2.pdf** – מכיל תשובות לחלק תיאורטי דו"ח מכין LAB2
 - ✓ תיקייה בשם **CW** - מכילה שתי תיקיות, אחת בשם **Sources** של קובצי source (קבצים עם סיומת *.c), והשנייה בשם **Project_Headers** של קובצי header (קבצים עם סיומת *.h).

L. צורת הגשה דוח מסכם:

- הגשת מטלת דוח מכין תיעשה ע"י העלאה למודל של תיקיית zip מהצורה **id1_id2.zip** (כאשר $id1 < id2$), רק הסטודנט עם הת"ז id1 מעלה את הקבצים למודל.
- התיקייה תכיל את שני הפרטים הבאים בלבד:
 - ✓ קובץ **final_labx.pdf** – מכיל תיאור והסבר לדרך הפתרון של מטלת זמן אמת.
 - ✓ תיקייה בשם **CW** - מכילה שתי תיקיות, אחת בשם **Sources** של קובצי source (קבצים עם סיומת *.c), והשנייה בשם **Project_Headers** של קובצי header (קבצים עם סיומת *.h).

בהצלחה.