



מגמת הנדסת אלקטרוניקה ומחשבים

ספר התמחות:

מיקרו מעבדים

VHDL

תוכן עניינים

2.....	דבר המפמ"ר
3.....	מיקרו מעבדים
304	VHDL

דבר המפמ"ר

אנו עומדים בפתחה של תקופה של חידושים טכנולוגיים, אתגרים חשיבתיים והתנסויות מרתקות. במגמת הנדסת אלקטרוניקה ומחשבים אתם תחשפו לסביבת למידה מתקדמת ומשמעותית. זכרו, כי אתם חלק חשוב מהמחר ובידכם הכוח והידע לקחת חלק בפיתוח דור העתיד.

הפקדתי בידי צוות מורי המגמה, אחריות רבה, לחשוף אתכם לידע חדשני, להנחות אתכם בפרויקטים מאתגרים ולהוביל אתכם להצלחה.

שלומי אדמונד אחנין

מפמ"ר מגמות הנדסת אלקטרוניקה ומחשבים ומערכות בקרה ואנרגיה

מיקרו מעבדים

פרק 1: תכונות Embedded C

מערכת משובצת מחשב, Embedded System, היא מערכת מחשב ייעודית המתפקדת בתוך מכלול רחב יותר של מערכת מכנית או חשמלית. לעיתים קרובות למערכת יש אילוץ תכנות בזמן אמת (Real Time Computing). המערכת מכונה משובצת משום שהיא חלק ממכלול התקנים מכניים או חשמליים.

מערכות משובצות מודרניות מבוססות על מיקרו-בקרים מסוגים שונים, ישנם מיקרו-בקרים לשימוש כללי, מיקרו-בקרים המתמחים בחישובים מיוחדים ואפילו מערכות המותאמות באופן מיוחד ליישום הנדרש. מעבדי DSP (Digital Signal Processor) הם דוגמה למיקרו-בקרים מתמחים בחישובים.

יש מערכות משובצות מחשב בשעונים דיגיטליים, נגני MP3 ובקרים תעשייתיים. ישנן מערכות פשוטות בעלות מיקרו-בקר אחד וישנן מערכות מורכבות בעלות מספר רב של מיקרו-בקרים והתקנים היקפיים. ברכב מודרני יש מספר רב של מערכת מחשב: מחשב מנוע, מחשב תיבת ההילוכים מחשב בקרת שיוט ועוד.

Embedded C 1.1

שפת Embedded C היא סדרה של הרחבות לשפת C שנועדה לתת מענה למערכות משובצות מחשב. מבחינה היסטורית נדרשו לשפת C תוספות מיוחדות התומכות במרחבי זיכרון מרובים, חלוקת זיכרון לבנקים (Memory Banks), פעולות חשבוניות עם נקודה עשרונית קבועה ופנייה להתקני קלט פלט בחומרה.

בשנת 2008 הורחב התקן של שפת C כדי לתמוך בדרישות אלה. השפה המורחבת משתמשת בכל התחביר הרגיל של שפת C.

השוואה בין שפת C לבין שפת Embedded C

שפת Embedded C	שפת C
יישומים מבוססי מיקרו-בקר.	יישומי מחשב שולחני.
שימוש במשאבים המוגבלים של המיקרו בקר כמו זיכרונות RAM ו-ROM והדקי קלט פלט.	שימוש במשאבי המחשב כמו זיכרון ומערכת ההפעלה.
תכונות ייחודיות כמו טיפוסים משתנים בעלי נקודה קבועה, מרחבי זיכרון והדקי קלט פלט.	תכונות סטנדרטיות כמו משתנים.
המהדר מייצר קובץ שאפשר להוריד למיקרו-בקר. שם תתבצע הריצה.	מהדרי C מייצרים קובץ הרצה המתואם למערכת ההפעלה.

מרחבי הזיכרון במיקרו-בקר ATmega329P

למיקרו-בקר ATmega329P הקיים בלוח UNO יש שלושה מרחבי זיכרון שונים:

- זיכרון FLASH בגודל 32kB המשמש לתכנית ולערכים קבועים. הערכים ישמרו בזמן ההורדה (Upload) לזיכרון.
- זיכרון SRAM בגודל 2kB המשמש למשתני התכנית.
- זיכרון EEPROM בגודל 1kB המשמש לשמירת נתונים לטווח ארוך. אפשר לשנות את הנתונים בזמן הרצה תוך שימוש בספרייה EEPROM.

כדי להשתמש בזיכרון FLASH לערכים קבועים נוסף את מילת המפתח PROGMEM לצד ההגדרה של משתנים או באמצעות פעולת המאקרו F המציינת שמירת טקסט בזיכרון ה-FLASH. דוגמאות לכך נראה בהמשך.

כמו כן אפשר להנחות את המהדר להקצות מקום בזיכרון ה-SRAM באמצעות מילת המפתח volatile לצד שם המשתנה. הנחיה זו מונעת מהמהדר להקצות שטח זמני למשתנה. השימוש בהנחיה זו, כפי שכבר ראינו במקצוע המוביל, משמשת עבור שגרות פסיקה. בפרק 2, חומרת הבקר, נתייחס בהרחבה למרחבי זיכרון אלה.

1.2 חזרה מבני בקרה בשפת C

ביטוי לוגי פשוט

ביטוי לוגי הוא ביטוי תנאי המחזיר את אחת משתי אפשרויות הבאות: אמת (true) או שקר (false). ביטוי המחזיר ערך שונה מאפס נחשב לביטוי אמת, וביטוי המחזיר ערך השווה לאפס נחשב לביטוי שקר.

כאשר ביטוי התנאי כולל אופרטורים לוגיים של יחס, תוצאת התנאי תהיה 1 לערך אמת ו-0 לערך שקר.

להלן תחביר הביטוי הלוגי, הנקרא משמאל לימין:

Expression1 op Expression2

Expression1 ו-Expression2 הם ביטויים חשבוניים, ו-op מייצג אופרטור של יחס: גדול, גדול או שווה, קטן, קטן או שווה, שווה או שונה.

בטבלה 1.1 מופיעים אופרטורי היחס:

אופרטור	תיאור	דוגמאות לערכי אמת (1)	דוגמאות לערכי שקר (0)
<	קטן	$3 < 8$	$8 < 3$
<=	קטן שווה	$3 <= 8, 2 <= 2$	$8 <= 3$
>	גדול	$8 > 3$	$3 > 8$
>=	גדול שווה	$8 >= 3, 2 >= 2$	$3 >= 8$
==	שווה	$5 == 5$	$4 == 5$
!=	שונה	$4 != 5$	$5 != 5$

טבלה 1.1 – אופרטורי היחס

הערות:

- שימו לב יש לקרוא את הטבלה משמאל לימין.
- שימו לב להבדל בין אופרטור ההשמה = לאופרטור היחס ==.

ביטוי לוגי מורכב

ביטוי לוגי מורכב הוא אוסף של ביטויים לוגיים פשוטים הקשורים זה לזה באמצעות מילות קישור. מילות הקישור הן: וגם, ו/או, לא.

בטבלה 2.2 מופיעות את מילות הקישור.

אופרטור	תיאור	דוגמה	הסבר
&&	מילת הקישור וגם (AND)	$a > 9 \ \&\& \ a < 100$	אמת רק כאשר a הוא מספר דו-ספרתי.
	מילת הקישור ו/או (OR)	$a < 0 \ \ a > 100$	אמת רק כאשר a הוא מספר שלילי או גדול מ-100.
!	מילת קישור לשלילה (NOT)	$!(a < 0 \ \ a > 100)$	אמת רק כאשר a הוא מספר בין 0 לבין 100, כולל ערכי הקצה.

טבלה 2.2 – אופרטורי יחס לוגיים מורכבים

סימון אופרטור לוגי שונה מסימון אופרטור בוליאני. הכפלת סימן האופרטורים הבוליאני AND ו-OR משנה את משמעותם לאופרטורים לוגיים. להלן הדרכים לסימון האופרטורים הלוגיים והאופרטורים הבוליאניים:

- AND לוגי מסומן ב-&&, ואילו AND בוליאני מסומן ב-&
- OR לוגי מסומן ב-||, ואילו OR בוליאני מסומן ב-|
- NOT לוגי מסומן ב-!, ואילו NOT בוליאני מסומן ב-~

הקדימות של האופרטורים הלוגיים && ו-|| נמוכה מהקדימות של הפעולות החשבוניות והבוליאניות של אופרטורי היחס. הקדימות של אופרטור השלילה ! גבוהה, ומסמנים קדימות זאת באמצעות שימוש בסוגריים.

האופרטורים הלוגיים && ו-|| הם בינאריים, כלומר פועלים על שני ביטויים לוגיים. כלומר, בפעולת AND לוגי, אם הביטוי הלוגי הראשון הוא שקר, לא נבדק התנאי השני; ובפעולת OR לוגי, אם הביטוי הלוגי הראשון הוא אמת, לא נבדק התנאי השני. לעומתם, האופרטור הלוגי ! הוא אונרי, כלומר פועל על ביטוי לוגי יחיד.

משפט תנאי if

משפט תנאי בסיסי עשוי לכלול ביטוי לוגי פשוט או מורכב וקטע תכנית המתבצע רק כאשר תוצאת הביטוי הלוגי היא אמת. כאשר תוצאת הביטוי הלוגי היא שקר לא מתבצע דבר וביצוע

```
if (logical expression)
{
    הוראות
}
```

התכנית ממשיך אל השורה הבאה בתכנית. להלן תחביר של משפט תנאי (logical expression משמעו ביטוי לוגי):

כאשר קטע התכנית מורכב מהוראה אחת אפשר להשתמש בתחביר הזה:

```
if (logical expression)
    הוראה יחידה
```

למרות זאת, מומלץ להשתמש בתחביר הקודם, הכולל {}, המאפשר הרחבה של קטע התכנית בצורה פשוטה.

חשוב לזכור ⓘ

אסור לסיים את שורת התנאי בסימן <.>.

תרגול

שאלה 1.1

כתבו ביטוי לוגי הבודק אם תלמיד קיבל ציון עובר בבחינה. ציון הבחינה מאוחסן במשתנה mark וציון עובר הוא ציון הגבוה מ-54.

שאלה 1.2

כתבו תכנית הבודקת אם מספר הנמצא במשתנה number הוא זוגי. אם המספר זוגי הדליקו את נורת הלד בהדק 13 בערכת הפיתוח.

משפט תנאי if-else

משפט התנאי עשוי לכלול ביטוי לוגי פשוט או מורכב ושני קטעי תכנית, האחד מתבצע אם תוצאת הביטוי הלוגי היא אמת, והאחר – אם תוצאת הביטוי הלוגי היא שקר. משפט תנאי זה נקרא גם משפט ברירה, משום שאפשר לברור בין שתי אפשרויות ביצוע של קטעי התכנית. משתמשים במשפט תנאי זה כאשר יש שני מקרים המצריכים תגובה שונה. לאחר ביצוע הקטע המתאים התכנית ממשיכה בהוראה המופיעה לאחר משפט התנאי. להלן תחביר משפט הברירה:

```
if (logical expression)
{
    הוראות כאשר התנאי הוא אמת
}
else
{
    הוראות כאשר התנאי הוא שקר
}
```

חשוב לזכור

אסור לסיים את שורת התנאי בסימן <.>.

משפט תנאי else-if

במשפט תנאי else-if מופיעים כמה משפטי תנאי ברצף. משתמשים במשפט זה כאשר צריך לבדוק מספר רב של תנאים הקשורים זה לזה.

משפט התנאי כולל ביטוי לוגי פשוט או מורכב וקטע תכנית המתבצע כאשר תוצאת הביטוי הלוגי היא אמת. לאחר ביצוע קטע התכנית מסתיים משפט התנאי, והתכנית ממשיכה בהוראה הבאה אחריו. כאשר תוצאת הביטוי הלוגי היא שקר מתבצע משפט תנאי אחר הכולל ביטוי לוגי. אם תוצאת הביטוי הלוגי של משפט התנאי החדש היא אמת יתבצע קטע קוד מתאים. אם תוצאת הביטוי הלוגי היא שקר יתבצע משפט תנאי נוסף. רצף משפטי התנאי ימשיך כל עוד התנאי הוא שקר. אם לא מתקיים אף לא אחד מהתנאים יתבצע קטע השייך ל-else האחרון.

משפט התנאי נקרא גם משפט רב-ברירה משום שאפשר לברור ביצוע של קטע ייעודי אחד בתכנית מתוך מספר קטעי ברירה אפשריים.

לאחר ביצוע הקטע המתאים ביצוע התכנית ממשיך בהוראה המופיעה לאחר משפט התנאי. להלן תחביר של משפט רב-ברירה לדוגמה:

```
if (logical expression 1)
{
    הוראות כאשר תנאי 1 הוא אמת
}
else if (logical expression 2)
{
    הוראות כאשר תנאי 2 הוא אמת
}
...
else
{
    הוראות כאשר אף לא אחד מהתנאים מתקיים
}
```

משפט תנאי טרינארי

לעיתים יש צורך לכתוב משפט תנאי הקובע את ערכו של משתנה על פי התנאי.

לדוגמה:

משפט תנאי אופרטור טרינארי	משפט תנאי רגיל
$\text{res} = a > 10 ? x : 2 * x;$	<pre> if (a > 10) res = x; else res = 2 * x; </pre>

הכתיבה באמצעות האופרטור הטרינארי קומפקטית:

logic expression ? value if true : value if false

תרגול 

שאלה 1.3

להדקים 6, 7 ו-8 מחוברים שלושה מפסקים: sw0, sw1 ו-sw2 בהתאמה. אם המפסק סגור, מתקבל "1" במוצאו.

להדקים 9 ו-10 מחוברים שתי נורות: L1 ו-L2 בהתאמה. נורה דולקת אם מספקים לה "1".

אם sw0 פתוח שתי הנורות כבויים. אם sw0 במצב סגור ו-sw1 במצב סגור דולקת נורה L1. נורה L2 דולקת אם sw2 סגור.

כתבו תכנית למימוש הבעיה הנ"ל.

משפט switch-case

לעיתים צריך לבדוק את ערכו של ביטוי מספרי ולבצע הוראות שונות עבור ערכים שונים של הביטוי. תפריטי תוכניות, לדוגמה, מציגים מספר אפשרויות. המשתמש יכול לבחור בכל אחת מהאפשרויות וכל בחירה גורמת להפעלת קטע קוד שונה. להלן התחביר של משפט מסוג זה:

```
switch (integral expression)
{
    case integral constant:
        statements
    case integral constant:
        statements
    ...
    default:
        statements
}
```

להלן הסבר של הביטויים בתחביר של משפט switch-case:

- **expression** – הוא ביטוי מספרי שלם בלבד. ביטוי זה מחושב ולאחר מכן מתבצעת התאמה של תוצאת החישוב לאחד מהערכים הקבועים המופיעים ליד משפט case.
- **integral constant** – ערך מספרי קבוע. ערך זה חייב להיות ייחודי בבולוק ההגדרה של משפט הברירה switch. בסופו מופיע הסימן <:>.
- **default** – ערך המציין את כל האפשרויות שאין להן התאמה לאף לא אחד מהערכים המופיעים במשפט ה-case. ערך זה חייב להיות מוגדר בבולוק משפט רב-הברירה switch.
- **statements** – הוראות המתבצעות לאחר התאמת הערך הקבוע עם ערך הביטוי השלם. הוראות אלה עשויות להכיל את ההוראה break המאפשרת הפסקת ביצוע הוראות בבולוק הנוכחי. השימוש בהוראה זו יגרום לסיום משפט רב-הברירה. אי-שימוש בהוראת break יגרום לכך שכל ההוראות, ממשפט ה-case המתאים ועד לסוף משפט רב-הברירה, יבוצעו בזו אחר זו.

לולאת while

בלולאה זו קודם נבדק התנאי (בקרת כניסה), ואם הוא מתקיים (אמת) גוף הלולאה מתבצע. לאחר ביצוע גוף הלולאה חוזרים לבדוק את התנאי. אם התנאי לא מתקיים (שקר) מסתיים משפט החזרה וממשיכים אל ההוראה הבאה בתכנית. אם התנאי לא מתקיים בזמן ריצת התכנית, גוף הלולאה לא יתבצע. להלן התחביר של לולאת while:

```
while (logic expression)
{
    // גוף הלולאה
}
```

אם גוף הלולאה הוא הוראה אחת בלבד, אפשר להשמיט את הסוגריים המסולסלות {}. עם זאת, מומלץ מאוד לשמור על התחביר עם הסוגריים.

לולאת do-while

בלולאה זו קודם מתבצע גוף הלולאה ולאחר מכן נבדק התנאי (בקרת יציאה). אם התנאי מתקיים (אמת) חוזרים לבצע את גוף הלולאה. אם התנאי אינו מתקיים (שקר) מסתיים משפט החזרה וממשיכים אל ההוראה הבאה בתכנית. לפי כך גוף הלולאה מתבצע לפחות פעם אחת. להלן תחביר של לולאת do-while:

```
do {
    // גוף הלולאה
} while (logic expression);
```

 חשוב לזכור

בלולאת do-while חייבים לשמור על הסוגריים המסולסלות {}.

לולאת for

בתבניות לולאה משתמשים במשתנה הנקרא מונה הלולאה. למשתנה זה יש ערך התחלתי מסוים. ערכו של המשתנה מתעדכן במהלך התכנית עד שהוא מגיע לערך סופי. ערכו של המונה יגדל במהלך התכנית, אם הערך הסופי קטן מהערך ההתחלתי, ויפחת במהלך התכנית, אם הערך הסופי קטן מהערך ההתחלתי. כאשר נדרש ביצוע של לולאות כאלה נוח להשתמש בלולאת for הכוללת את כל המרכיבים הדרושים: אתחול, תנאי, עדכון וגוף הלולאה. להלן תחביר של לולאת for:

```
(שדה עדכון ; שדה תנאי ; שדה אתחול ) for
{
    גוף הלולאה
}
```

להלן הסבר הביטויים המופיעים בתחביר לולאת for:

שדה אתחול – בשדה זה נתונות הוראות המתבצעות לפני שהלולאה מתבצעת בפעם הראשונה. ההוראות בשדה זה מופרדות האחת מהשנייה באמצעות סימן פסיק <,>.

שדה תנאי – שדה זה מכיל ביטוי לוגי המשמש לבקרת ביצוע הלולאה. אם התנאי מתקיים (אמת) הלולאה תתבצע, אם התנאי אינו מתקיים (שקר) הלולאה מסתיימת – תנאי עצירה.

שדה עדכון – שדה זה מכיל הוראות המתבצעות לאחר ביצוע גוף הלולאה. ההוראות בשדה זה מופרדות זו מזו באמצעות סימן פסיק <,>.

הערות:

- בין שדה לשדה חייב להופיע סימן נקודה-פסיק <;>
- אין חובה להשתמש בכל השדות
- שימוש בהוראה for(;;) חוקי ומציין לולאה אינסופית

סדר ביצוע פעולות הלולאה

1. ביצוע שדה האתחול
2. אם התנאי מתקיים, יבוצעו הפעולות האלה:
 - 2.1. יבוצע גוף הלולאה
 - 2.2. יבוצע שדה עדכון
 - 2.3. ביצוע התכנית יחזור לסעיף 2
3. אם התנאי אינו מתקיים מסתיים ביצוע הלולאה, והביצוע עובר לפקודה הבאה בתכנית

שאלה 1.4

כתבו תכנית לחישוב מכפלת המספרים מ-1 עד 10.

ממשו באמצעות שלוש לולאות.

שאלה 1.5

כתבו תכנית לחישוב סכום המספרים: $1+7+14+\dots+7k$. כאשר נתון $k=10$.

ממשו באמצעות שלוש לולאות.

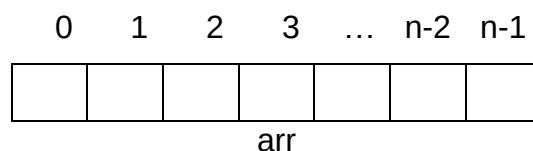
שאלה 1.6

כתבו תכנית לחישוב סכום המספרים: $1-2+3-4+5-6+\dots+21$.

ממשו באמצעות שלוש לולאות.

1.3 מערכים חד ממדיים

מערך הוא אוסף של משתנים מאותו טיפוס. אוסף המשתנים מקבל שם המייצג את המערך. לכל אחד מתאי המערך אפשר מספר סידורי רציף מאפס ואילך. להלן איור 1.1, שבו תיאור עקרוני של מערך:



איור 1.1 – תיאור עקרוני של מערך

באיור 1.1 מוצג מערך בגודל n בשם `arr`. האינדקס של התא הראשון הוא 0, האינדקס של התא השני הוא 1, וכן הלאה עד לתא האחרון שהאינדקס שלו הוא $n-1$. הפנייה אל תאי המערך במתבצעת באמצעות השימוש באינדקס כך: `arr[index]`. אופן פנייה זה, `arr[index]`, שקול לפנייה למשתנה בודד, כלומר בדרך זו אפשר לבצע כל פעולה שאפשר לבצע על משתנה בודד.

להלן דוגמה להכרזה על מערך:

```
type arr_name[size] = {value,...};
```

להלן הסבר הביטויים במערך לדוגמה:

- **type** – הוא סוג המשתנה (char, int, long, float)
- **arr_name** – שם המערך. שם זה חייב לקיים את כללי קביעת השמות בתכנית.
- **size** – קבוע מספרי שלם המייצג את גודל המערך.
- **value** – ערכי אתחול של המערך.

לפניכם חמישה כללים להכרזה על מערך:

- בהכרזה על מערך חייבים לקבוע את טיפוס המערך ואת שמו.
- גודל המערך חייב להיות ערך מספרי קבוע. למשל, ההגדרה `int a[10]` – חוקית.
- גודל מערך לא יכול להיות משתנה. למשל, ההגדרה `int b[x]` אינה חוקית, אם `x` משתנה.
- אפשר להגדיר מערך באמצעות ערכי אתחול בלי לציין את גודלו. למשל, `int arr[] = {1,1,2,3};` הוא מערך בגודל 4
- מספר הערכים שבאמצעותם מאתחלים את המערך חייב להיות זהה לגודלו של המערך. למשל, ההגדרה `int arr[3] = {1,2};` אינה חוקית.

תרגול 

שאלה 1.7

הגדירו מערך של מספרים שלמים. ערכי המספרים הם:

100, -9, 90, 4, -10.

שאלה 1.8

הגדירו מערך המכיל את התווים האלה:

'U', 'N', 'O', 'B', 'O', 'O', 'K'

הגדרת מערך במרחב זיכרון ה-FLASH

הגדרת מערך במרחב זיכרון FLASH (זיכרון תכנית) משמשת להגדרת ערכים קבועים בלבד. טבלאות המרה הן דוגמה להגדרת מערך במרחב זיכרון זה משום שהערכים בטבלה אינם משתנים. הגדרות לדוגמה:

```
const PROGMEM int arr[] = {25, 89, -93};
const char str[] PROGMEM = "THIS IS A STRING";
```

הגישה למערכים מתבצעת כמו במערכים אחרים.

אפשר לגשת למערכים גם באמצעות פעולות המאקרו האלה:

```
pgm_read_byte_near(array_name + index)
pgm_read_word_near(array_name + index)
```

שימוש במאקרו F()

פעולת המאקרו F() מאפשרת להגדיר מחרוזות במרחב הזיכרון FLASH כדי מנת לחסוך בזיכרון SRAM (זיכרון המשתנים). לדוגמה:

```
Serial.println(F("Text is located in Flash Memory"));
```

פעולות print או println של המחלקה Serial משתמשות בזיכרון SRAM כדי להציג את המחרוזת המועברת כפרמטר. אם התכנית עמוסה בטקסט מומלץ להפעיל את המאקרו F() כדי לחסוך במקום.

דוגמה 1.1 – הפעלת תצוגה מסוג אנודה משותפת (CA) במערך

בדוגמה זו מוצגת תכנית המציגה את המספרים 0 עד 9 באופן מחזורי. בדוגמה השתמשנו בפנייה למפתח ובמעריך חד-ממדי.

```
//          0          1          2          3          4
byte table7Seg[] = {B1000000, B1111001, B0100100, B0110000, B0011001,
//          5          6          7          8          9
    B0010010, B0000010, B1111000, B0000000, B0010000};

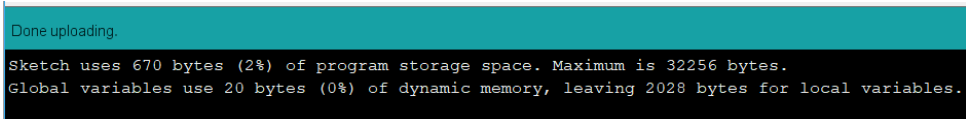
byte index;

void setup() {
    DDRD = B11111110;    // הגדרת 7-1 כיציאות 0 ככניסה
    index = 0;           // אתחול משתנה האינדקס למערך
}

void loop() {
    PORTD = table7Seg[index]; // הפעלת יחידת התצוגה עם הערך מהמערך
    delay(1000);             // השהייה של 1 שנייה
    index++;                  // קידום אינדקס המערך
    if (index == 10)         // בדיקה אם סיימתם את כל האיברים במערך
        index = 0;          // אם סיימתם, חזרו לתחילת המערך למחזור הצגה נוסף
}
```

המערך החד-ממדי table7Seg מכיל 10 ערכים המייצגים את הספרות 0-9. במקום 0 מופיע הערך הבינארי של הספרה 0, ובמקום 1 מופיע הערך של הספרה 1 וכן הלאה. המערך החד-ממדי מוגדר במרחב הזיכרון SRAM.

לאחר פעולת ההידור תתקבל ההודעה הבאה:



```
Done uploading.
Sketch uses 670 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 20 bytes (0%) of dynamic memory, leaving 2028 bytes for local variables.
```

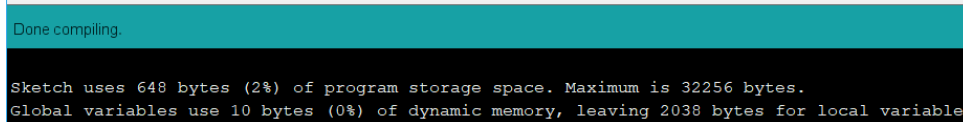
התכנית משתמשת ב-670 בתים במרחב זיכרון התכנית (FLASH) וב-20 בתים ממרחב זיכרון המשתנים (SRAM).

שינוי ההכרזה של המערך כקבוע במרחב זיכרון התכנית:

```
1 //          0          1          2          3          4
2 const byte table7Seg[] PROGMEM = {B1000000, B1111001, B0100100, B0110000, B0011001,
3 //          5          6          7          8          9
4          B0010010, B0000010, B1111000, B0000000, B0010000};
```

השינוי אינו משפיע על התכנית, אך הוא משנה את הדרך שהזיכרון מוקצה בה.

לאחר פעולת ההידור נקבל את ההודעה הבאה:



Done compiling.
Sketch uses 648 bytes (2%) of program storage space. Maximum is 32256 bytes.
Global variables use 10 bytes (0%) of dynamic memory, leaving 2038 bytes for local variables.

התכנית משתמשת ב-648 בתים במרחב זיכרון התכנית (FLASH) וב-10 בתים ממרחב זיכרון המשתנים (SRAM).

קל להבחין בחיסכון במשאבי זיכרון כשמשתמשים במערכים בעלי ערכים קבועים במרחב זיכרון התכנית.

1.4 פעולות

פעולה (או פונקציה) היא רצף הוראות המאגדות יחד כדי לבצע מטלה מוגדרת. השימוש בפעולות משפר את מבנה התכנית, את קריאות הקוד ואת הגמישות של התכנית לבצע שינויים. שיפור זה מאפשר להוריד את עלויות הפיתוח והתחזוקה של התכנית.

להלן יתרונות השימוש בפעולות:

- מודולאריות – הפעולה מתמקדת לרוב בבעיה אחת מסוימת.
- שימוש חוזר – אפשר להשתמש בה מספר רב של פעמים ללא צורך בכתיבת הקוד מחדש – השימוש בפעולות חוסך במקום ובזמן פיתוח.
- תחזוקה – השימוש בפעולות מאפשר לזהות קטעי קוד בעייתיים ולתקן אותם בקלות.
- קריאות – התכנית הראשית מניעה את הפעולות, ולכן קל יותר לקרוא את התכנית ולהבין את אופן פעולתה.

הגדרת הפעולה

הגדרת הפעולה כוללת את גוף (body) הפעולה ואת סופה (return).

```
returned_type function_name(parameters list)
{
    local variables
    body
    return
}
```

חתימת הפעולה כוללת את שלושת המרכיבים האלה:

- **returned_type** – מרכיב זה מציין את הטיפוס המוחזר של הפעולה. כאשר הפעולה אינה צריכה להחזיר ערך, יוגדר הטיפוס המוחזר כ-void.
- **function_name** – מרכיב זה הוא שם ייחודי (identifier) המזהה את הפעולה. השם נבחר על פי כללי בחירת השמות בתוכניות בשפת C – מקובל לתת שם בעל משמעות המרמז על תפקידה של הפעולה.
- **parameters list** – רשימת הפרמטרים הפורמליים של הפעולה. רשימת הפרמטרים היא אוסף של טיפוסים משתנים המופרד בפסיק <,>. עם זאת הרשימה גם יכולה להיות ריקה. בשלב הפעלת הפעולה יקבלו הפרמטרים הפורמליים ערכים. חובה לספק ערכים בזמן ההפעלה לכל אחד ואחד מהפרמטרים הפורמליים.

בפעולה אפשר להגדיר משתנים מקומיים שתחום הקיום שלהם הוא בה בלבד. שמות המשתנים המקומיים חייבים להיות מיוחדים ושונים משמות הפרמטרים הפורמליים.

גוף הפעולה יכול לכלול כל אחת מהוראות שפת C, כולל משפטי תנאי ולולאות.

הוראת return

פעולה חייבת להסתיים בהוראה return המציינת את סוף הפעולה ומחזירה את ערך תוצאת פעולתה. הערך המוחזר חייב להיות תואם לסוג ה-returned_type. אפשר להשמיט את הוראת return אם טיפוס הערך המוחזר הוא void. אפשר להשתמש בהוראה כמה פעמים הפעולה. בכל פעם שההוראה מתבצעת הפעולה מסתיימת.

משתנה קבוע בפעולה

אפשר להגדיר בפעולה משתנה במרחב זיכרון התכנית, אבל חייבים להגדיר אותו כמשתנה קבוע (סטטי). דוגמה להגדרה בפעולה:

```
const static char str[] PROGMEM = "Text in function";
```

משתנה קבוע מוגדר תמיד בזיכרון ונגיש רק מתוך הפונקציה. בדוגמה לעיל מוגדר המשתנה הקבוע במרחב התכנית. אם נשמיט את המילה PROGMEM מהדוגמה, יוגדר המשתנה הקבוע במרחב הזיכרון הדינאמי SRAM.

 תרגול

שאלה 1.9

כתבו פונקציה המקבלת מספר שלם ומחזירה 1 אם הוא זוגי ו-0 אם הוא אי-זוגי.

שאלה 1.10

כתבו פונקציה המקבלת שני מספרים שלמים ומציגה את כל המספרים בתחום מהקטן לגדול.

שאלה 1.11

כתוב פונקציה המקבלת מספר שלם ובודקת אם הוא ראשוני. אם הוא ראשוני היא תחזיר 1, ואם לא תחזיר 0.

דוגמה 1.2 פעולה לחישוב סכום ערכי מערך

```

int arr[10];
void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(0)); // starts the random number generator
  fill_random(arr, 10);      // fill array with random numbers
  print_arr(arr, 10);        // print the array
  Serial.print("The sum of the array is ");
  Serial.println(sum_arr(arr, 10)); // calculate the sum and print
}

void loop() {
}

/*
 * This function fills an array with random numbers
 * input: a[] array of integer
 *        arr_size the size of the array
 * output: None
 */
void fill_random(int a[], int arr_size){
  for(int i=0; i<arr_size; i++)
    a[i] = random(200); // generate a random number
}

/*
 * This function displays the array in the serial monitor
 * input: a[] array of integer
 *        arr_size the size of the array
 * output: None
 */
void print_arr(int a[], int arr_size){
  for(int i=0; i<arr_size-1; i++) {
    Serial.print(a[i]);
    Serial.print(", ");
  }
  Serial.println(a[arr_size-1]);
}
/*

```

המשך הקוד בעמוד הבא <<

```

* This function calculate the sum of an array
* input: a[] array of integer
*      arr_size the size of the array
* output: the sum of the values in the array
*/
int sum_arr(int a[], int arr_size){
    int sum = 0;
    for(int i=0; i<arr_size; i++)
        sum += a[i];
    return sum;
}

```

הסבר

בתכנית זו השתמשנו בשלוש פעולות על מערכים. כל הפעולות זומנו מתוך הפעולה `setup`. להלן פירוט הפעולות:

1. פעולת `setup`:

- התכנית תקבע באמצעות ההוראה `Serial.begin` את קצב השידור ל-9600bps ותפתח את הערוץ לתקשורת.
- הפעלת מחולל המספרים האקראיים.
- מילוי המערך `arr` במספרים אקראיים.
- הדפסת המערך במוניטור.
- חישוב סכום איברי המערך והדפסתו במוניטור.

2. פעולת `loop`: פעולה ריקה

3. פעולת `fill_random`: הפעולה מקבלת את המערך `arr`, מערך של מספרים שלמים, ואת גודלו `arr_size` ומציבה בתוכו מספרים אקראיים מ-0 עד 199.
4. פעולת `print_arr`: הפעולה מקבלת את המערך `arr`, מערך של מספרים שלמים, ואת גודלו `arr_size` ומדפיסה בשורה אחת של המוניטור את המספרים לפי הסדר.
5. פעולת `sum_arr`: הפעולה מקבלת את המערך `arr`, מערך של מספרים שלמים, ואת גודלו `arr_size`, מחשבת את סכום המספרים במערך ומחזירה אותו.

דוגמה 1.3 חישוב ממוצע של 10 דגימות

```

const int ARR_SIZE = 10;    // קבוע המגדיר את גודל המערך
int arr[ARR_SIZE];          // הגדרת המערך
void setup() {
    Serial.begin(9600);
    clear_arr(arr, ARR_SIZE);
}
void loop() {
    add_arr(arr, ARR_SIZE, analogRead(A0));
    delay(10);
    Serial.println(avg_arr(arr, ARR_SIZE));
}
/*
 * This function fills an array with zero
 * input: a[] array of integer
 *        arr_size the size of the array
 * output: None
 */
void clear_arr(int a[], int arr_size){
    for(int i=0; i<arr_size; i++)
        a[i] = 0;
}
/*
 * This function calculate the average of an array
 * input: a[] array of integer
 *        arr_size the size of the array
 * output: the average of the values in the array
 */
float avg_arr(int a[], int arr_size){
    long sum = 0;
    for(int i=0; i<arr_size; i++)
        sum += a[i];
    return (float)sum / arr_size;
}

```

המשך הקוד בעמוד הבא <<

```

/*
 * This function add a vlaue to cell 0 in an array
 * all other values are shifted upwards
 * input: a[] array of integer
 *        arr_size the size of the array
 *        value to insert
 * output: None
 */
void add_arr(int a[], int arr_size, int value){
    for(int i=arr_size-1; i>0; i--){
        a[i] = a[i-1];
    }
    a[0] = value;
}

```

הסבר

1. פעולת setup:

- התכנית תקבע באמצעות ההוראה Serial.begin את קצב התקשורת ל-9600bps ותפתח את הערוץ לתקשורת.
- איפוס המערך arr.

2. פעולת loop:

- הוספת ערך דגימה חדש למערך באמצעות הפעולה add_arr.
- השהייה לביצוע המרת האנלוגי בדיגיטלי.
- הצגת ממוצע הקריאות.

3. פעולת clear_arr:

- הפעולה מקבלת את המערך arr, מערך של מספרים שלמים, ואת גודלו arr_size, ומאפסת אותו.

4. פעולת avg_arr:

- הפעולה מקבלת את המערך arr, מערך של מספרים שלמים, ואת גודלו arr_size, ומחזירה את הממוצע של הערכים במערך.

5. פעולת add_arr:

- הפעולה מקבלת את המערך arr, מערך של מספרים שלמים, את גודלו arr_size וערך של דגימה. הפעולה מזיזה כלפי מעלה את הערכים במערך: ערכו של תא 8 יוזז לתא 9, ערכו של תא 7 יוזז לתא 8 וכן הלאה. ההזזה האחרונה תעביר את ערכו של תא 0 לתא 1. ערך הדגישה החדש יוכנס לתא 0 במערך.

תרגול **שאלה 1.12**

כתבו פעולה המקבלת מערך a של מספרים שלמים, מספר שלם s המייצג את גודלו של המערך ומספר שלם נוסף x . הפעולה תחזיר את מספר הפעמים שהערך x הופיע במערך a .

שאלה 1.13

נתונה הפעולה `what` המקבלת מערך של מספרים שלמים ואת גודלו של המערך.

```
int what(int a[], int arr_size) {  
    int b = 0;  
    for(int i=1; i < arr_size; i++) {  
        if (a[i] > a[b])  
            b = i;  
    }  
    return b;  
}
```

מה מחזירה הפעולה?

1.5 עבודה עם קבצים רבים

לעיתים בפיתוח של פרויקט יש לכתוב פעולות רבות. מצד אחד הן מקילות על פיתוח הפרויקט, מצד שני הן גורמות לתכנית להיות עמוסה. לכן אפשר לחלק את מסמך התכנית לקטעים האחראיים לטיפול בקטגוריות שונות. עם זאת, האיתור בתוך המסמך עשוי להיות מסורבל. כדי למנוע סרבול ישנה אפשרות ליצור במקום קטעים כמה קבצים שונים בתכנית. כך תחזוקת התכנית וניהולה יהיו פשוטים יותר. אפשר ליצור קבצים מסוגים שונים:

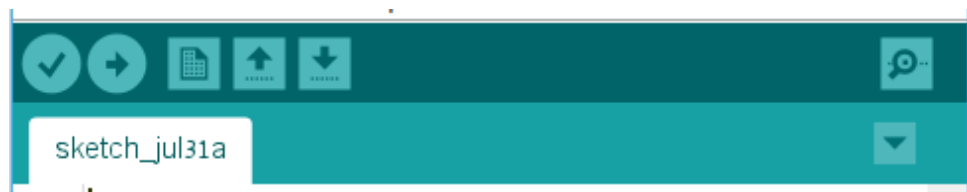
- קובץ חסר סיומת – קובץ של סביבת הפיתוח.
- קובץ בעל סיומת h – קובץ כותרת של שפת C או C++.
- קובץ בעל סיומת c – קובץ קוד של שפת C.
- קובץ בעל סיומת cpp – קובץ קוד של שפת C++.

בפרק זה נתייחס רק לקבצים חסרי סיומת, משום הקבצים האחרים דורשים ידע קודם בשפת C או בשפת C++.

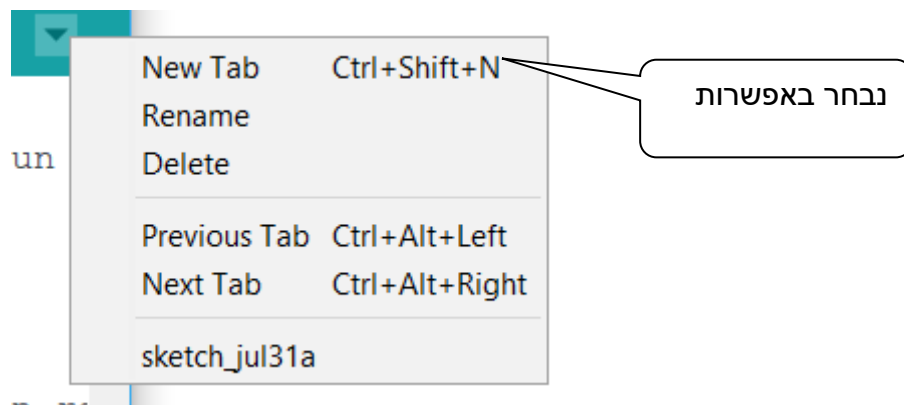
סביבת הפיתוח משרשרת את הקבצים חסרי הסיומת לקובץ הראשי המכיל את הפעולות setup ו-loop. כך שבפועל החלוקה לקבצים היא לוגית בלבד, ואין צורך לציין שימוש בהם. בניגוד לספריות שחייבים לשלב.

הוספת קובץ חדש

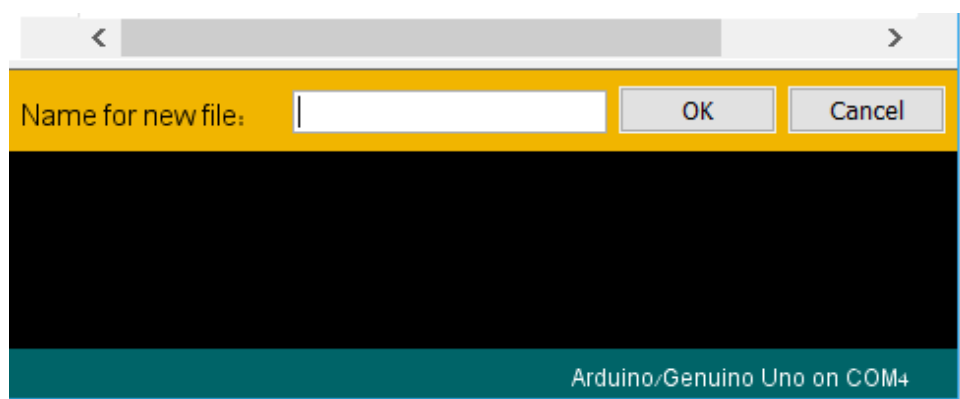
לאחר יצירת פרויקט בסביבת הארדוינו נוסף קובץ חדש בלחיצה על תפריט הכרטיסיות (Tab) המסומן באיור להלן.



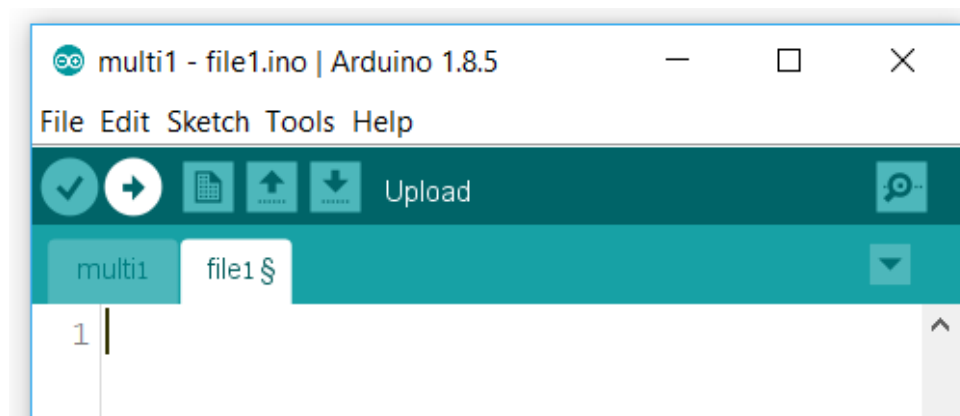
לאחר הלחיצה יתקבל התפריט הזה:



בחלק התחתון של סביבת הפיתוח נקבל את תיבת טקסט הזו:



נזין בה את שם הקובץ ללא סיומת, נאשר בלחיצה על OK, ונקבל כרטיסייה חדשה עם השם שבחרנו.

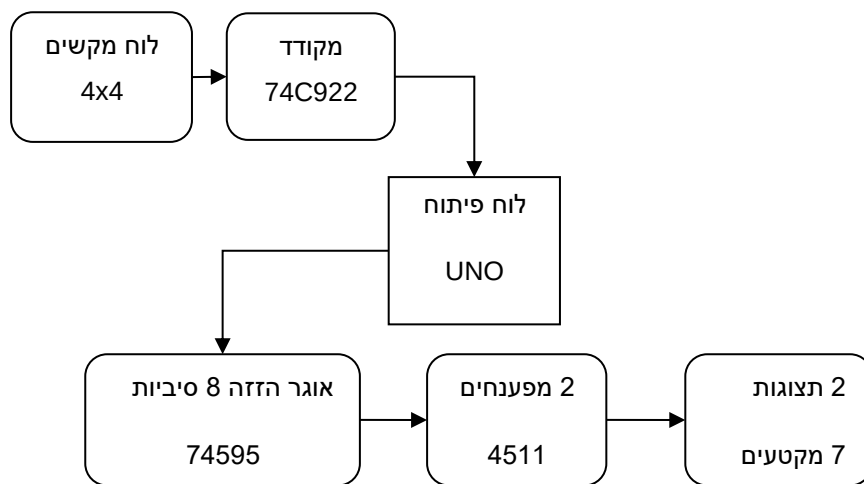


1.6 פיתוח פרויקטים

בחלק זה נציג פתוח של פרויקט הממחיש את השימוש בכמה קבצים.

פרויקט: לוח מקשים ותצוגת 7 מקטעים

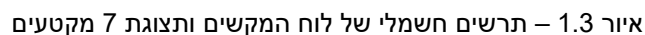
באיור 1.2 מוצג תרשים המלבנים של המערכת. בתרשים – התקן הקלט, לוח מקשים, המכיל 16 מקשים ומקודד שתפקידו לזהות לחיצה ולספק מידע ללוח הפיתוח. התקן הפלט הוא תצוגה נומרית (מספרית) בת שתי ספרות. לכל יחידת תצוגה חובר מפענח הכולל דוחף זרם. המידע לתצוגה משודר באופן טורי מלוח הפיתוח אל אוגר הזזה ברוחב שמונה סיביות. כל ארבע סיביות משמשות להצגת מידע ביחידת תצוגה אחת.



איור 1.2 – תרשים מלבנים

התרשים החשמלי של הפרויקט

באיור 1.3 מוצג התרשים מפורט של הרכיבים המחוברים אל לוח הפתוח. התרשים החשמלי לא כולל את לוח הפתוח. בתרשים מופיעים רק שמות ההדקים בלוח הפתוח שהרכיבים מחוברים אליהם.



מקודד לוח המקשים 74C922

מקודד לוח המקשים פועל על עיקרון של סריקה של מצב המקשים. כאשר הוא מזהה לחיצה על מקש כלשהו, ההדק DA במקודד עובר למצב גבוה למשך כל זמן הלחיצה. בזמן זה, ועד ללחיצה הבאה, בהדקים DOA-DOB יופיע הצירוף בינארי המייצג את המקש שנלחץ.

אוגר הזזה 74HC595

אוגר ההזזה מקבל מידע טורי מהלוח הפיתוח ומעביר אותו למידע מקבילי. ההעברה מתבצעת באמצעות שלושה הדקים:

- SDI – כניסה טורית לאוגר.
- RCLK – אות השעון עבור פעולת ההזזה.
- SRCLK – אות המשמש לנעילת המידע במוצא אוגר ההזזה.

העברת המידע הטורי לאוגר ההזזה אינה משפיעה על מוצאי אוגר ההזזה מפני שהרכיב כולל נעילה המפרידה בין אוגר ההזזה לבין המוצאים. לאחר העברת שמונה הסיביות מועבר המידע אל המוצאים באמצעות הדק SRCLK.

מפענח 4511

המפענח מקבל צירוף בינארי לפי קוד BCD במבואות A–D ומייצר אותות הפעלה ליחידת התצוגה. הרכיב כולל דוחפי זרם המוגבלים בנגדים חיצוניים.

מיפוי הדקים לוח פתוח UNO

7	6	5	4	3	2	1	0	מפתח/סיבית
C	C	13	12	11	10	9	8	PB
						SRCLK	RCLK	הקצאה
7	6	5	4	3	2	1	0	PD
SDI	DOD	DOC	DOB	DOA	DA			הקצאה

תכנית של הפרויקט

התכנית בנויה משלושה קבצים:

- Project1 – התכנית הראשית הכוללת את הפעולות setup ו-loop.
- Keyboard – הגדרות ופעולות הקשורות ללוח המקשים בלבד.
- Disp7Seg – הגדרות ופעולות הקשורות ליחידת התצוגה בלבד.

להלן פירוט על הקבצים.

קובץ Project1

```

unsigned int value;
void setup() {
  InitKbd();
  InitShiftReg();
  value = 0;
}
void loop() {
  char key;
  if (keyAvailable()>0){
    key = getKey();
    if (isDigit(key)){
      value = value * 10 + (key - '0');
      value = (value>100)? value % 100 : value;
      show(value);
    }
  }
}

```

פעולת setup – הפעולה מפעילה את הפעולות InitKbd ו-InitShiftReg המגדירות את הדקי החמרה.

פעולת loop – בפעולה מתבצעת בדיקה אם נלחץ מקש (keyAvailable). אם כן, ערכו של המקש נבדק כדי לוודא שערך זה מייצג ספרה (isDigit) ומייצרים באמצעותו מספר דו-ספרתי. המספר הדו-ספרתי נשלח ליחידת התצוגה באמצעות הפעולה show.

תרגול 

שאלה 1.14

השתמשו במנוע חיפוש ומצאו את ההסבר לפעולה isDigit.

מהן הפעולות המזהות אותיות בכלל? ומהן הפעולות המתייחסות לגודל האותיות?

קובץ Keyboard

```

const int DOA = 3;
const int DOB = 4;
const int DOC = 5;
const int DOD = 6;
const int DA = 2;    // INT0
const int KBDPINS[] = {DOA, DOB, DOC, DOD, DA};

const PROGMEM char table[16] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '0',
'A', 'B', 'C', 'D', '#', '*'};

const int BUFSIZE = 5;
char buf[BUFSIZE];
byte index;
/*
 * This function initialize the MCU pins
 * and attach the interrupt pin
 * INPUT: None
 * OUTPUT: None
 */
void InitKbd(){
    for(int i=0; i<5; i++) {
        pinMode(KBDPINS[i], INPUT);
    }
    attachInterrupt(digitalPinToInterrupt(DA), kbdISR, FALLING);
    index = 0;
}
/*
 * This is an ISR function
 * whenever DA is falling the value is read from the encoder
 * the raw value is translated to an appropriate char
 * and stored in a queue implemented with an array
 * INPUT: None
 * OUTPUT: None
 */

```

המשך הקוד בעמוד הבא <<

```

void kbdISR(){
    byte tmp = (PIND & B01111000) >> 3;
    if (index < BUFSIZE){
        buf[index] = table[tmp];
        index++;
    }
}
/*
 * This function check if a key is available
 * INPUT: None
 * OUTPUT: ture  if exist
 *          false if not exist
 */
bool keyAvailiable(){
    return index;
}
/*
 * This function return a key if exist.
 * The values is kept in a queue
 * INPUT: None
 * OUTPUT: key  if exist
 *          -1 if not exist
 */
char getKey(){
    char tmp;
    if (keyAvailiable()>0) {
        tmp = buf[0];
        for(int i = index-1; i>0; i--)
            buf[i-1] = buf[i];
        index--;
        return tmp;
    }
    return -1;
}

```

טבלת ההמרה table – המערך table המוגדר במרחב הזיכרון של התכנית משמש כטבלת המרה של ערך גלמי, המתקבל ממוצא המקודד, בערך תווי המופיע על לוח המקשים. הערך הגולמי משמש כאינדקס למערך והערך שבתוך המערך מציין את התו. הערכים שהוגדרו במערך table שבתכנית מתאימים לחיבור מסוים של לוח המקשים למקודד. אם בשלב יישום הפרויקט לוח המקשים יחובר באופן שונה מהמתואר, תידרשו להתאים את הערכים בטבלת ההמרה.

פעולת InitKbd – פעולה זו מגדירה באמצעות לולאה את ההדקים המחוברים למקודד לוח המקשים ומקשרת את ההדק DA לשגרת הפסיקה kbdISR. שגרת הפסיקה תופעל בירידת האות בהדק DA.

פעולת kbdISR – הפעולה פונה למפתח PD וקוראת ממנו 4 סיביות. הסיביות מוזזות כך שמתקבל מספר מ-0 עד 15. מספר זה משמש לפנייה למערך table המוגדר במרחב התכנית ומכיל את ערך התווי של המקש. ערך זה מוכנס למערך המשמש כתור. גודל התור הוא 5.

פעולת keyAvailable – הפעולה מחזירה את ערכו של המשתנה index. כאשר המשתנה מכיל 0, התור ריק ומחוזר הערך שקר. בכל מקרה אחר מוחזר הערך אמת.

פעולת getKey – הפעולה בודקת אם התור מלא. אם התור מלא יוחזר הערך שנמצא בראש התור (תא אפס במערך). לאחר הוצאת הערך מהמערך מוזזים הערכים במערך מקום אחד לכיוון תא 0. אם התור ריק יוחזר ערך -1.

קובץ Disp7Seg

```

const int SDI = 7;
const int RCLK = 8;
const int SRCLK = 9;
const int SHIFTRGPINS[] = {SDI, RCLK, SRCLK};
/*
 * This function initialize the MCU pins
 * INPUT: None
 * OUTPUT: None
 */
void InitShiftReg(){
    for(int i=0; i<3; i++) {
        pinMode(SHIFTRGPINS[i], OUTPUT);
    }
    digitalWrite(SRCLK, LOW);
}
/*
 * This function display a 2 digits value
 * INPUT: value to display
 * OUTPUT: None
 */
void show(int value) {
    byte unit = value % 10;
    byte ten = value / 10;
    digitalWrite(SRCLK, LOW);
    shiftOut(SDI, RCLK, MSBFIRST, (ten << 4) | unit);
    digitalWrite(SRCLK, HIGH);
}

```

פעולת InitShiftReg – פעולה זו מגדירה באמצעות לולאה את ההדקים המחוברים אל אוגר ההזזה כפלט. כמו כן היא קובעת כי הדק נעילת המידע יהיה ברמה נמוכה. מטרת הבחירה להגדיר ההדקים בלולאה היא להציג את השימוש במערכים. במקרה זה, כמובן, אפשר להגדיר כל אחד מההדקים בנפרד.

פעולת show – הפעולה מקבלת ערך מספרי דו-ספרתי, מפרקת אותו לספרותיו (יחידות ועשרות) ומשתמשת בפעולה קיימת shiftOut כדי לשדר את המידע באופן טורי. לאחר השידור הטורי, הדק SRCLK עולה לרמה גבוהה כדי להעביר את המידע אל הדקי המוצא באוגר ההזזה.

תרגול

שאלה 1.15

השתמשו במנוע חיפוש ומצא הסבר לפעולה shiftOut. מה משמעות המילה MSBFIRST ומהן האפשרויות הנוספות של הפעולה, אם וקיימות? כיצד תשפיע בחירה אחרת של פרמטר זה על פעולת התוכנית?

שאלה 1.16

בפרויקט הנ"ל השתמשנו בלוח מקשים ותצוגה. ישנה דרישה להרחיב את הפרויקט כך שיכלול מנוע חשמלי. לוח המקשים יקבע את מהירות המנוע, והיא תהיה בתחום 0 עד 99. כאשר ערך של המהירות 0 המנוע במנוחה, וכאשר הערך הוא 99 המנוע יפעל במהירות הגבוהה ביותר.

- א. השלימו את תרשים המלבנים על פי הדרישה.
- ב. השלימו את תרשים החשמלי לחיבור המנוע.
- ג. הוסיפו קובץ נפרד לטיפול במנוע.
- ד. שנו את התכנית הראשית כך שהמערכת תפעל על פי הדרישה.

סיכום פרק 1

בפרק 1 למדתם את הנושאים האלה:

1. תכונות של שפת Embedded C
2. ההבדלים בין שפת C לבין שפת Embedded C
3. חזרה על משפטי תנאי
4. חזרה על switch-case
5. חזרה על לולאת while
6. חזרה על לולאת do-while
7. חזרה על לולאת for
8. חזרה על מערך חד-ממדי
9. הגדרת מערך במרחב זיכרון ה-FLASH
10. חזרה על פעולות
11. ריבוי קבצים
12. עקרונות פיתוח פרויקטים וריבוי קבצים

פרק 2: חומרת הבקר

ביחידה הקודמת, מערכות משובצות מחשב, הכרנו את המונח מיקרו-בקר באופן כללי, והתמקדנו בעיקר במיקרו-בקר ATmega32P. בפרק זה נכיר מונחים אלה לעומק על מונחים אלה וכן את תכונותיהם.

2.1 מיקרו-בקר

בקר (Controller) הוא מחשב שנועד לבקר תהליכים או פעולה של התקן כלשהו. הבקר אחראי, למשל, לבקרת הטמפרטורה במערכת מיזוג אוויר. מיקרו-בקר הוא בקר הדחוס לתוך שבב אחד.

תכונות המיקרו-בקר:

- לרוב, בקר מטפל במשימה אחת שכמות הנתונים בה מעטה.
- מרחב הזיכרון שלו אינו גדול.
- מהירות הפעולה אינה גבוהה.
- יש בבקר רכיבי עזר מובנים המסייעים בזמן פעולתו.

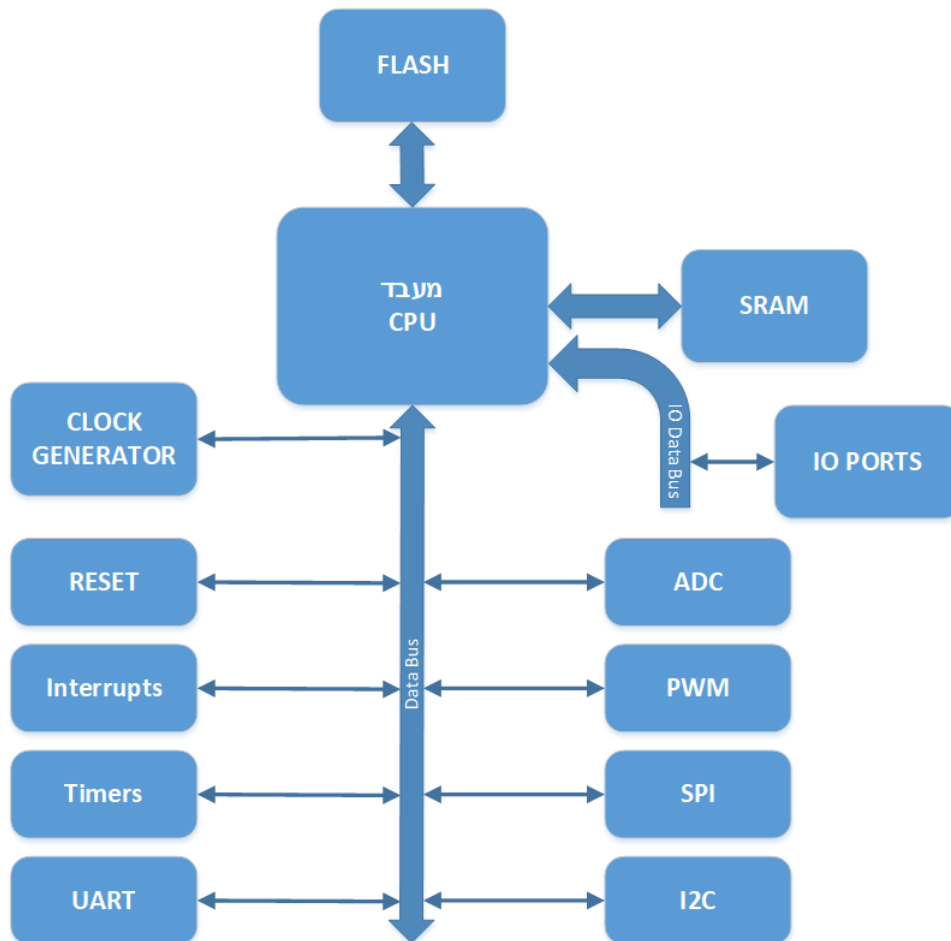
רכיבי העזר בבקר:

- **קוצבי זמן** – משמשים למדידת זמן.
- **מונים** – משמשים למניין אירועים.
- **בקרי תקשורת טורית** – משמשים ליצירת תקשורת בין שני בקרים או בין הבקר להתקנים אחרים.
- **בקר פסיקות** – משמש לניהול בקשות שירות של מעגלים פנימיים וחיצוניים המתריעים על אירועים שונים.
- **אפנון רוחב הדופק PWM** – משמש ליצירת דופק בעל רוחב משתנה.
- **ממירים A/D ו- D/A** – משמשים לקריאה ולכתיבה של אותות אנלוגיים.
- **משווי אנלוגיים** – משמשים לזיהוי התחום של ערכים אנלוגיים.

באיור 2.1 להלן מופיעות היחידות הבסיסיות שמהן בנוי מיקרו-בקר. לב המערכת הוא **המעבד** המחובר אל שני מרחבי זיכרון עיקריים: זיכרון הבזק (Flash) המשמש לרוב לאחסון התכנית, הערכים הקבועים ותכנית הטעינה (Loader), וזיכרון בגישה אקראית סטטי (SRAM – Static)

(RAM) המשמש לאחסון הנתונים. במיקרו-בקרים רבים יש גם זיכרון מסוג EEPROM שלא מופיע באיור.

פס נתוני קלט-פלט (IO Data Bus) מחבר את הדקי המיקרו-בקר מפתחי הקלט-פלט אל המעבד. חלק מההדקים משמשים למטרות נוספות פרט להיותם הדקי IO.



איור 2.1 – מבנה עקרוני של מיקרו בקר

Clock Generator – מחולל אות שעון של המיקרו-בקר. אפשר לקבוע את קצב עבודתו באמצעות חיבור גביש חיצוני. במיקרו-בקרים מסוימים אפשר לקבוע את קצב העבודה באמצעות שעון פנימי מובנה.

Reset – מערכת המאתחלת את המיקרו-בקר. מקורות האיפוס מגוונים, החל בכניסה מיוחדת המחוברת בדרך כלל למפסק לחצן, דרך המנגנון Watchdog ועד לאיפוס עקב נפילת מתח.

ADC – מערכת הממירה אותות אנלוגיים לאותות ספרתיים.

PWM – מערכת המייצרת אותות ריבועיים בעלי רוחב משתנה.

Interrupts – מנגנון לניהול בקשות לטיפול באירועים חיצוניים (הדק המחובר למעגל חשמלי) או באירועים פנימיים (קוצבי זמן, מונים, תקשורת טורית).

Timers – מונים הסופרים אירועים. מונים אלה יכולים גם לשמש כקוצבי זמן.

UART – מערכת המקשרת בין שני מיקרו-בקרים או בין מיקרו-בקר להתקן אחר. המערכת אחראית לשידור ולקליטה של מידע בצורה טורית.

I2C – מערכת המחברת בצורה טורית התקנים במרחקים קצרים אל המיקרו-בקר. התקשורת היא בשיטה דו-כיוונית למחצה (Half Duplex).

SPI – מערכת המחברת בצורה טורית התקנים במרחקים קצרים אל המיקרו-בקר. התקשורת היא בשיטה דו-כיוונית מלאה (Full Duplex).

בפרק זה נחזור על הנושאים האלה:

- כניסות ויציאות ספרתיות.
- כניסה אנלוגית המתחברת אל ממיר ADC.
- יציאה אנלוגית המשתמשת בהדק ספרתי ליצירת אותות PWM.
- פסיקות חיצוניות.
- קוצבי זמן.

בפרק 3 נעמיק בתקשורת הטורית UART המחברת בין מחשב ולוח UNO וכן בין שני לוחות UNO.

בפרקים 4 ו-5 נלמד על התקני התקשורת I2C ו-SPI ונציג יישומים שלהם.

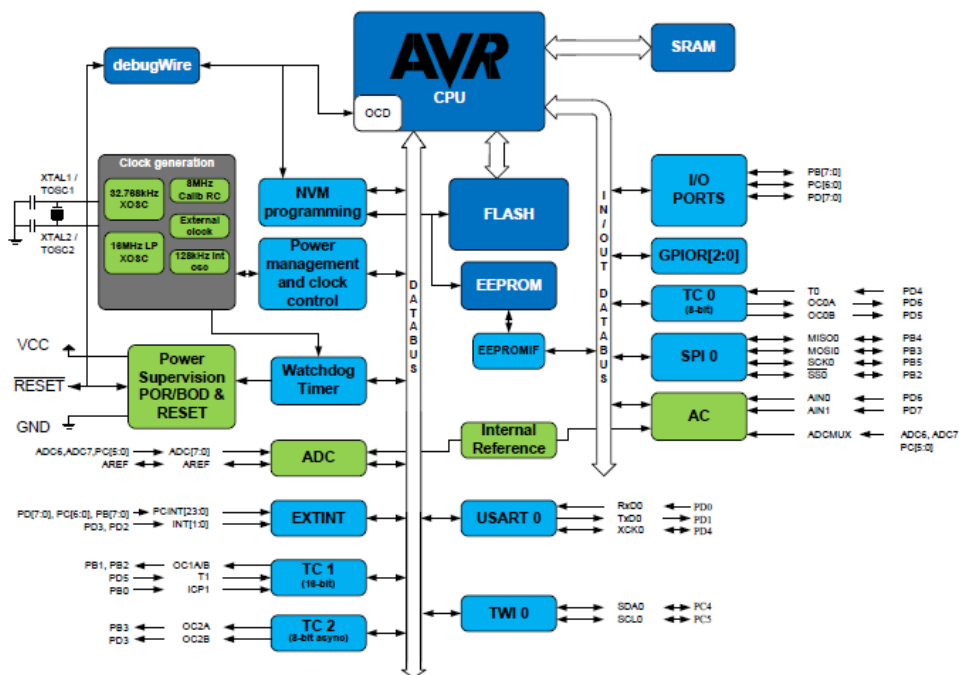
2.2 מבנה הפנימי של מיקרו-בקר ATmega328P

המיקרו-בקר ATmega328P של היצרנית Atmel® מבוסס על ארכיטקטורת AVR® enhanced RISC ומיוצר בטכנולוגיית CMOS שצריכת ההספק שלה נמוכה. המיקרו-בקר פועל עם מידע ברוחב 8 סיביות (8-bit). ההוראות בו מתבצעות במחזור פעולה אחד. כלומר, עבור קצב של 16MHz יתבצעו כ-16 מיליון פעולות בשנייה.

תכונות של מיקרו-בקר ATmega328P

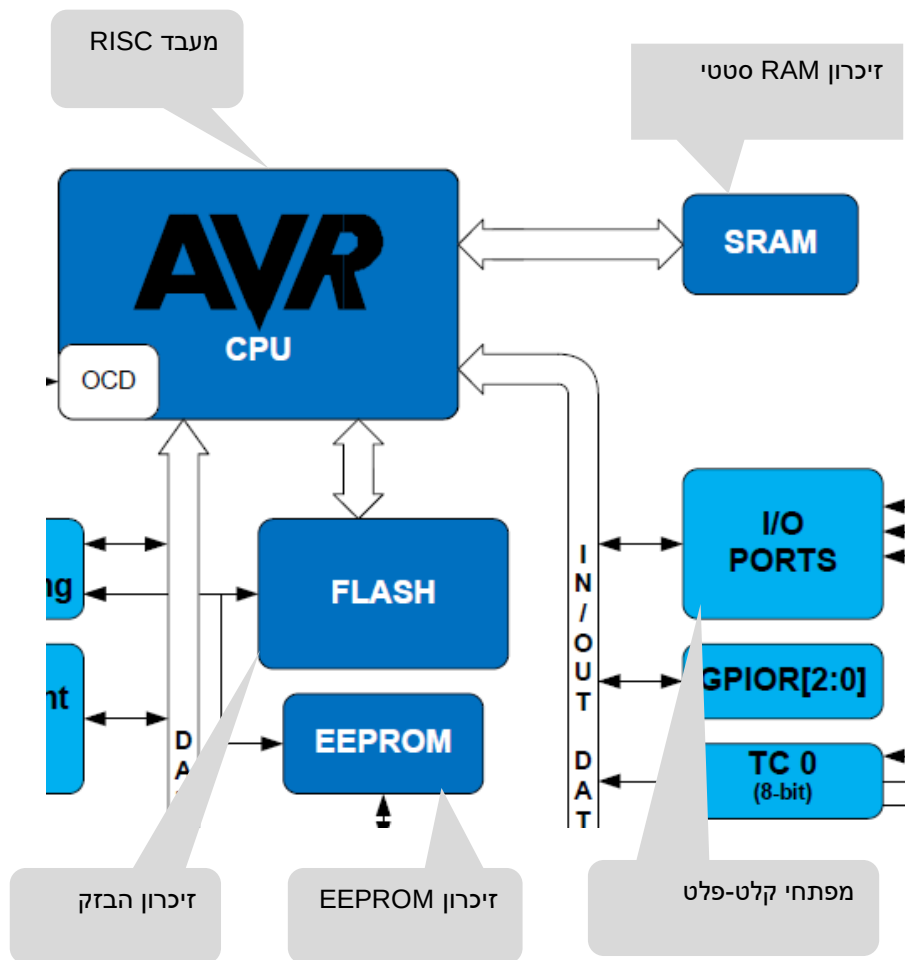
- ארכיטקטורת RISC 8 סיביות
 - 131 הוראות
 - רוב ההוראות מתבצעות במחזור אחד
 - 32 אוגרים לשימוש כללי
- מרחבי זיכרון
 - זיכרון Flash בגודל 32KB
 - זיכרון SRAM בגודל 2KB
 - זיכרון EEPROM בגודל 1KB
- רכיבים היקפיים
 - 2 מונים או קוצבי זמן של 8 סיביות
 - מונה או קוצב זמן של 16 סיביות
 - מונה זמן אמת עם מתנד נפרד (מונה 16 סיביות)
 - 6 ערוצי PWM
 - 6 ערוצי המרה מאנלוגי לדיגיטלי ברוחב 10 סיביות
 - 2 ממשקים טוריים SPI (ממשק רגיל וממשק USART)
 - ממשק טורי I2C (TWI)
 - תקשורת טורית USART
 - קוצב זמן Watchdog עם מתנד מקומי נפרד
 - משווה אנלוגי
- מאפיינים מיוחדים
 - איפוס בזמן הפעלה
 - מתנד פנימי
 - מקורות פסיקה פנימיים וחיצוניים
- קלט פלט I/O
 - 23 קווי I/O שאפשר לתכנת
- מתחי פעולה
 - 1.8–5.5V

באיור 2.2 מופיעה דיאגרמת הבלוקים של המיקרו-בקר.



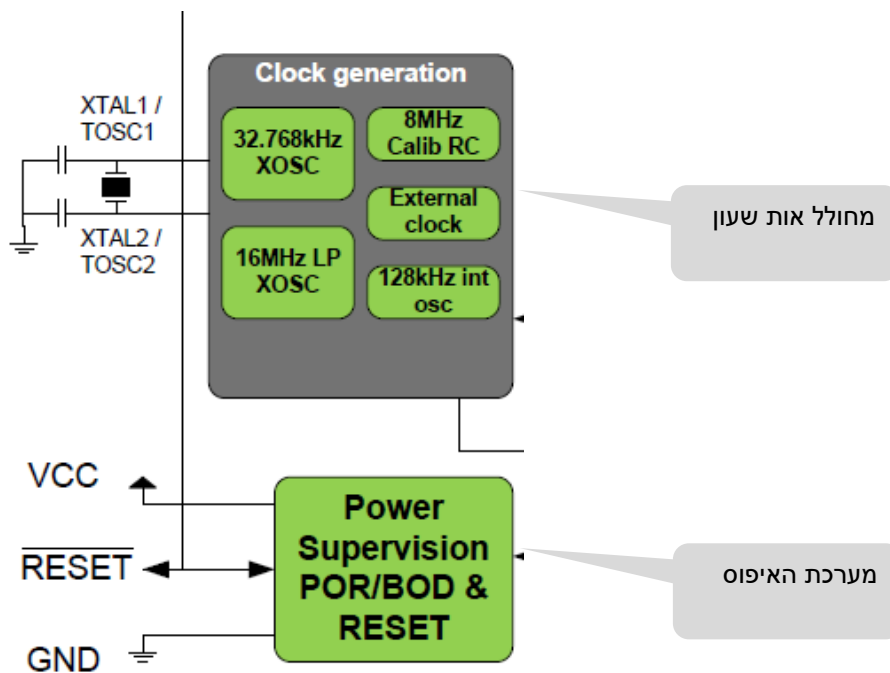
איור 2.2 – דיאגרמת בלוקים של ATmega328P

איור 2.3 מתאר את מרכיבי המחשב היסודיים: מעבד, זיכרון וקלט-פלט.



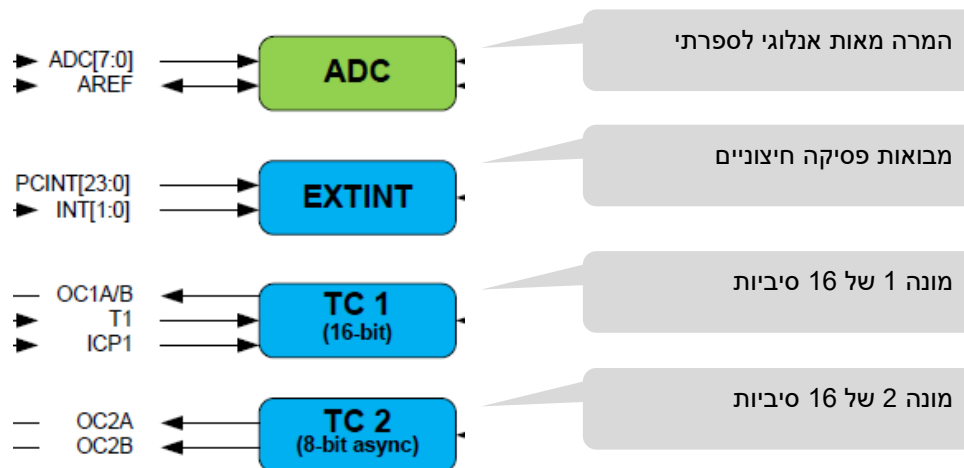
איור 2.3 – המעבד, יחידות הזיכרון ומפתחי קלט-פלט

איור 2.4 מתאר את מערכת מחולל אות השעון ואת מערכת האיפוס.



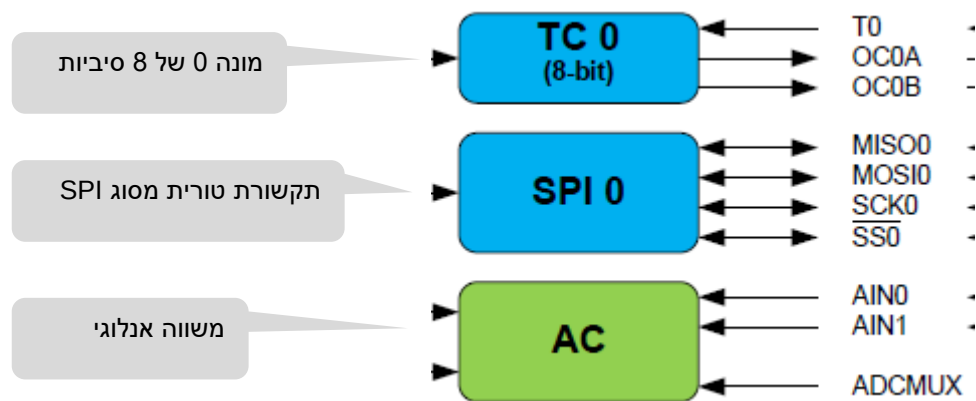
איור 2.4 – מחולל השעון ומערכת האיפוס

איור 2.5 מתאר רכיבי העזר במיקרו-בקר.



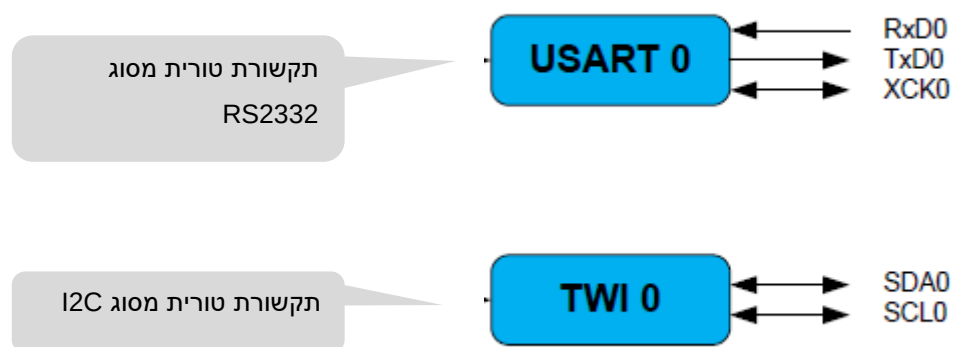
איור 2.5 – רכיבי עזר: פסיקה חיצונית, ממיר ADC ומוני 16 סיביות

איור 2.6 מתאר קבוצה נוספת של רכיבי עזר.



איור 2.6 – מונה 8 סיביות, משוואה אנלוגי ותקשורת טורית SPI

איור 2.7 מתאר עוד קבוצה של רכיבי עזר.



איור 2.7 – בקר תקשורת טורית RS232 ותקשורת טורית I2C

2.3 המפרט הטכני של Arduino UNO

הלוח Arduino UNO מבוסס על המיקרו-בקר ATmega328P. ללוח 14 הדקים המשמשים לכניסה או ליציאה ספרתית, 6 כניסות תקביליות, מתנד גבישי בתדר 16MHz, חיבור USB, שקע אספקת מתח, לחצן איפוס וממשק חיבור ICSP. נוסף על כך, הלוח כולל 4 נוריות LED: נורת חיווי פעולה ON; נורת LED לשימוש כללי המחוברת להדק 13; ושתי נוריות LED לחיווי פעולה של תקשורת בין המיקרו-בקר למחשב.

מפרט טכני

מיקרו-בקר	ATmega328P
תדירות שעון	16MHz
זיכרונות	
זיכרון FLASH, תכנית	32KB
זיכרון RAM סטטי לנתונים	2KB
זיכרון EEPROM	1kB
מאפיינים חשמליים	
מתח פעולה	5V
מתח מבוא בשקע המתח (מומלץ)	7–12V
זרם עבור הדק 3.3V	50mA
זרם לכל הדק I/O	20mA
הדקים ספרתיים (דיגיטליים)	
הדקי קלט-פלט ספרתיים (דיגיטליים) I/O	14 ממוספרים מ-0 עד 13 (אפשר להשתמש גם בהדקים A0–A5)
נורת LED מובנית LED_BUILTIN	הדק 13 בלוח
הדקים תקביליים (אנלוגיים)	
ADC	ממיר מתקבילי לספרתי ברוחב 10 סיביות ובקצב של 15ks/s (דגימות לשנייה)
כניסות תקביליות (אנלוגיות)	6 ערוצים מסומנים A0–A5
AREF כניסת מתח ייחוס	
הדקי I/O PWM (יציאות אנלוגיות)	6 מתוך 14 הדקי I/O ומסומנים ב-~>, <~> והם: 3, 5, 6, 9, 10, 11
תקשורת טורית	
הדקי I/O תקשורת UART	TX – 1 (מתוך 14 הדקי I/O), RX – 0
הדקי I/O תקשורת I2C	SDA – A4, SCL – A5 (מתוך 6 הדקים תקביליים)
הדקי I/O תקשורת SPI	SS – 10, MOSI – 11, MISO – 12, SCK – 13 (מתוך 14 הדקים)

2.4 מאפיינים חשמליים של הדקים ספרתיים

הדק תלוי בכיוון שהוגדר לו – כניסה או יציאה. להלן מאפיינים הדק כניסה והדק יציאה עבור $V_{CC} = 5V$.

מאפייני הדק כניסה:

סמל	תיאור	ערך מזערי	ערך מרבי	יחידה
V_{IL}	מתח כניסה לרמה נמוכה	-0.5	1.5	V
V_{IH}	מתח כניסה לרמה גבוהה	3	5.5	V
I_{IL}	זרם כניסה ברמה נמוכה		1	μA
I_{IH}	זרם כניסה ברמה גבוהה		1	μA

טבלה 2.1 – מאפיינים חשמליים הדק כניסה

מאפייני הדק יציאה:

סמל	תיאור	ערך מזערי	ערך טיפוסי	ערך מרבי	יחידה
V_{OL}	מתח יציאה ברמה נמוכה			0.9	V
V_{OH}	מתח יציאה ברמה גבוהה	4.1			V
I_{OL}	זרם יציאה ברמה נמוכה		20	40	mA
I_{OH}	זרם יציאה ברמה גבוהה		20-	-40	mA

טבלה 2.2 – מאפיינים חשמליים הדק יציאה

על אף שכל הדק יכול להעביר זרם של 20mA לצרכן, קיימת מגבלה על כמות הזרם הכוללת שמספר צרכנים יכולים לצרוך בו זמנית מהדקים שונים. צריכת הזרם מוגבלת ל-100mA על פי קבוצות הדקים ולפי הרמה הלוגית במוצא ההדק. להלן חלוקת הקבוצות על פי הרמות הלוגיות:

I_{OH}

קבוצה 1: PC0-PC5, PD0-PD4, ADC7, RESET

קבוצה 2: PB0-PB5, PD5-PD7, ADC6, XTAL1, XTAL2

I_{OL}

קבוצה 1: PC0-PC5, ADC7, ADC6

קבוצה 2: PB0-PB5, PD5-PD7, XTAL1, XTAL2

קבוצה 3: PD0-PD4, RESET

2.5 פעולות הדקים ספרתיים

הגדרת הדק ספרתי

לפני שימוש בהדק ספרתי יש להגדיר את אופן פעולתו: כניסה או יציאה. הגדרת ההדק מתבצעת באמצעות הפעולה `pinMode`:

`pinMode(pin, mode)`

הפעולה מקבלת שני פרמטרים ומגדירה את אופן פעולת ההדק.

`pin` מספר ההדק הספרתי מ-0 עד 13

`mode` הגדרת אופן הפעולה של ההדק:

OUTPUT – יציאה

INPUT – כניסה

INPUT_PULLUP – כניסה עם נגד Pull Up

כתיבת הדק יציאה ספרתי

כתיבת להדק יציאה ספרתי מתבצעת באמצעות הפעולה `digitalWrite`:

`digitalWrite(pin, value)`

הפעולה מקבלת שני פרמטרים וקובעת את ערך הדק היציאה.

`pin` מספר ההדק הספרתי מ-0 עד 13

`value` הערך המסופק להדק:

LOW – ערך נמוך, מתח 0V

HIGH – ערך גבוה, מתח 5V

הערה:

כאשר מגדירים הדק ספרתי כקלט (INPUT) ולאחר מכן כותבים לאותו הדק ערך לוגי HIGH, נגד ה-Pull Up של אותו הדק, ומתקבלת אותה תוצאה כמו שימוש ב-INPUT_PULLUP.

קריאת הדק יציאה ספרתי

קריאת הדק יציאה ספרתי מתבצעת באמצעות ההוראה digitalRead:

digitalRead(pin)

pin מספר ההדק הספרתי מ-0 עד 13

הערך שההוראה מחזירה הוא:

LOW – ערך נמוך, מתח 0V

HIGH – ערך גבוה, מתח 5V

2.6 מאפיינים חשמליים של הדקים תקביליים

1. מאפיינים של הדק כניסה אנלוגי:

כדי להשתמש בהדקים A0-A5 כהדקי כניסה אנלוגיים נודא שהם מוגדרים ככניסות ספרתיות רגילות (INPUT; אין צורך להגדיר באופן מיוחד מכיוון שזו ברירת המחדל). להלן המאפיינים:

תחום המתחים – אפשר לחבר לכל אחד מהדקי הכניסה מתח בתחום 0V עד 5V.

תחום הערכים הספרתיים – כל מתח בתחום מ-0V עד 5V מתורגם לערך מספרי בתחום מ-0 עד 1023 בהתאמה.

הרזולוציה – כושר ההבחנה של הממיר הוא $\Delta V = 5V/1024 = 4.88mV$.

זמן ההמרה – הזמן הדרוש לתרגום אות מתח לערך מספרי הוא $t_c = 100\mu Sec$.

$0V \leq V_{in} \leq 5V$	תחום מתחים
$0 \leq D \leq 1023$	תחום ערכים ספרתיים
$\Delta V = 4.88mV$	רזולוציה (כושר הבחנה)
$t_c = 100\mu Sec$	זמן המרה

טבלה 2.3 – מאפיינים של הדק כניסה אנלוגי

הנוסחה לחישוב הערך הספרתי (D) לפי מתח היא:

$$D = \left[V_{in} / \Delta V \right]$$

הערך השלם של היחס בין המתח האנלוגי בכניסה ובין כושר ההבחנה.

הנוסחה לחישוב מתח הכניסה לפי הערך הבינארי היא:

$$V_{in} = \Delta V \cdot D$$

שינוי תחום מתחי הכניסה

הערכים שתוארו לעיל הם ערכי ברירת המחדל של המיקרו-בקר. אפשר לשנות ערכים אלה באמצעות הפעולה **analogReference(type)** המאפשרת לקבוע אחת משלוש אפשרויות (type):

- **DEFAULT** – מתח ייחוס של 5V (ערכי מחדל); מתחי כניסה 0V עד 5V כפי שתואר לעיל.
- **INTERNAL** – מתח ייחוס מונה של 1.1V; המרה של מתחי כניסה מ-0V עד 1.1V.
- **EXTERNAL** – מתח ייחוס חיצוני בתחום 0V עד 5V; מתחי הכניסה נקבעים על פי מתח המסופק להדק AREF הסמוך להדק 13 במחבר הספרתי.

בחירת מתח ייחוס – Vref – משפיעה על תחום מתחי הכניסה:

$$V \leq V_{in} \leq V_{ref}$$

כושר ההבחנה משתנה בהתאמה:

$$\Delta V = V_{ref} / 1024$$

2. מאפיינים של הדק יציאה אנלוגי:

מכיוון שבפועל אין הדק יציאה אנלוגי, משתמשים ביציאה ספרתית הפועלת על עיקרון אפנון רוחב הדופק (PWM (Pulse Width Modulation.

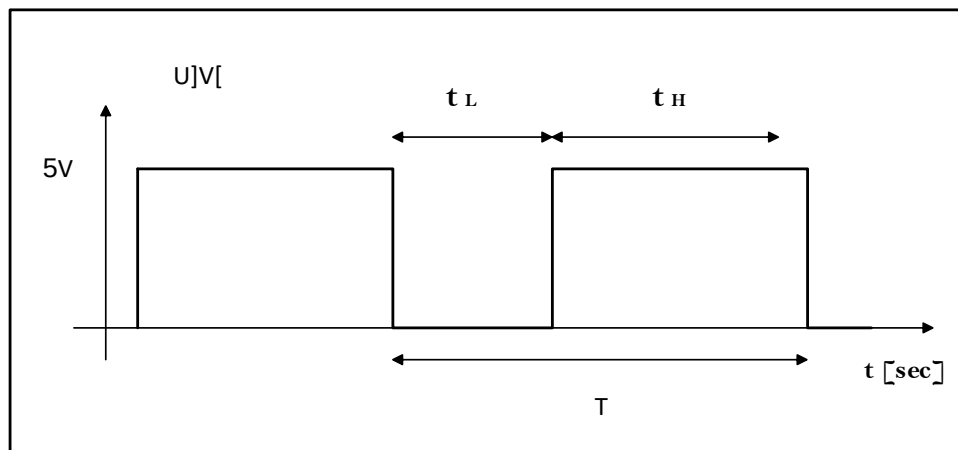
מאפייני גל ריבועי

- עוצמת הגל – מתח 0V או 5V.
- תדירות הגל – $f [Hz]$
- זמן המחזור – $T [sec]$
- גורם המחזור – DC (Duty Cycle) (מקובל לציין באחוזים). גורם המחזור הוא היחס בין פרק הזמן שבו הגל הריבועי נמצא בעוצמה 5V ובין זמן המחזור:

$$DC = \frac{t_H}{T}$$

$$T = t_H + t_L \quad \text{חשוב לציין כי:}$$

באיור 2.8 להלן מתואר גל ריבועי:



איור 2.8 – גל ריבועי

שיטת PWM מבוססת על העיקרון שהתדר נשמר קבוע, אך גורם המחזור משתנה לפי הצורך. עקב שינוי גורם המחזור משתנה המתח הממוצע של האות. המתח הממוצע הוא גודל המתח הישר השקול לגל הריבועי. המתח הממוצע למחזור אחד מחושב בנוסחה הזאת:

$$V_{AV} = \frac{5V \times t_H}{T} = 5V \times \frac{t_H}{T} = 5V \times DC$$

על פי הנוסחה, המתח הממוצע תלוי בגורם המחזור DC בלבד.

ערכו של גורם המחזור הוא בתחום: $0\% \leq DC \leq 100\%$. מכאן נובע שהמתח הממוצע יהיה בתחום $0 \leq V_{AV} \leq 5V$.

מסקנה: בממוצע אפשר להפיק אות מתח בתחום 0V עד 5V בתלות בגורם המחזור.

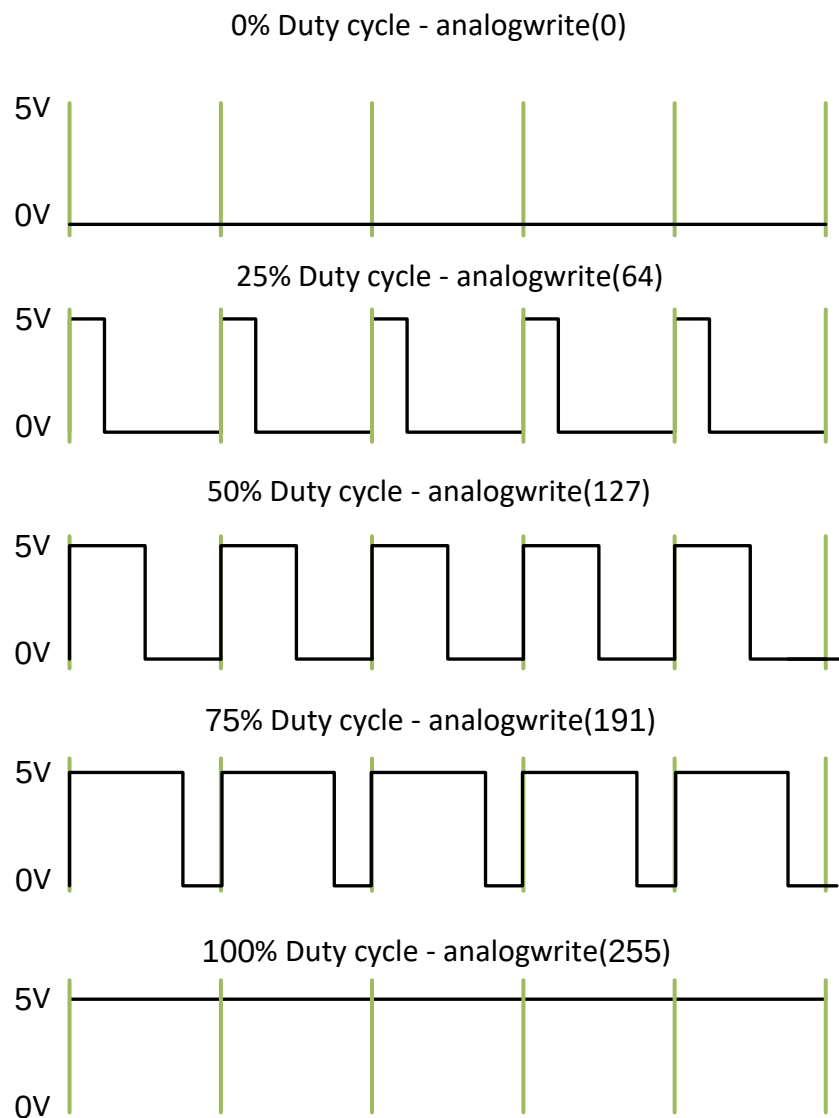
כדי לייצר אות PWM בתוכנה משתמשים במספר המייצג את גורם המחזור. המספר הוא בתחום 0 עד 255. ההתאמה בין גורם המחזור, המתחים והערך המספרי נתונה להלן:

$$0\% \leq DC \leq 100\%$$

$$0 \leq V_{AV} \leq 5V$$

$$0 \leq Value \leq 255$$

להלן איור 2.9 המתאר את הקשר בין הערך המספרי הנכתב ליציאה האנלוגית לבין גורם המחזור המתקבל.



איור 2.9 – הקשר בין הערך המספרי לבין מחזור גורם המחזור

2.7 פעולות הדקים אנלוגיים (תקביליים)

קריאת הערך האנלוגי

הקריאה מהדק כניסה אנלוגי מתבצעת באמצעות ההוראה `analogRead`:

`analogRead(pin)`

pin מספר ההדק האנלוגי בתחום A0 עד A5 (או 0 עד 5).

ההוראה מחזירה ערך מספרי בין 0 לבין 1023.

כתיבת הערך האנלוגי

כתיבת הערך האנלוגי מתבצעת באמצעות ההוראה `analogWrite`:

`analogWrite(pin, value)`

pin מספר ההדק האנלוגי 3, 5, 6, 9, 10, 11

value ערך מספרי בין 0 לבין 255

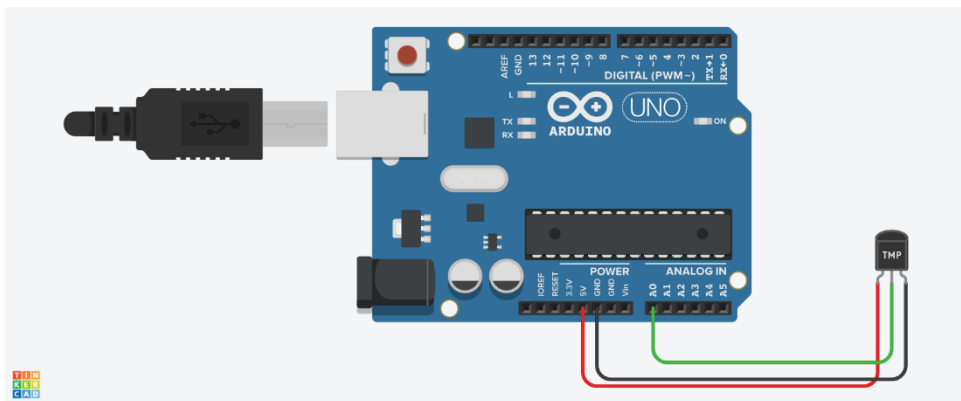
ההוראה גורמת ליצירת האות הריבועי בהדק המתאים עד להוצאת ערך אנלוגי חדש של אותו הדק או לקריאה או לכתיבה של ערך ספרתי להדק זה.

בהדקים 3, 9, 10 ו-11 נוצר אות בתדר של 490Hz בקירוב ואילו בהדקים 5 ו-6 נוצר אות בתדר 980Hz בקירוב (בגלל שימוש נוסף המתבצע ביחידות הקשורות להדקים אלה).

דוגמה 2.1 – חיבור LM36 לכניסה אנלוגית A0 בסימולציה¹ TinkerCad

בדוגמה זו תוצג תכנית להצגת הטמפרטורה בחלון ה-Serial Monitor. התכנית תציג את הערך הבינארי, את עוצמת המתח הנמדד ואת הטמפרטורה.

הרכיב LM36 דומה לרכיב LM35, שהכרנו ביחידה מערכות משובצות מחשב. תחום הטמפרטורה של LM36 הוא בין -40°C ל- 125°C , ואילו תחום הטמפרטורה של LM35 הוא בין 10°C ל- 125°C . בטמפרטורה 25°C מתח המוצא של הרכיב LM36 הוא 750mV , ואילו מתח המוצא של LM35 הוא 500mV התואם ל- 50°C .



איור 2.11 – דיאגרמת חיבורים של LM36 למיקרו בקר

הנוסחה לחישוב המתח היא:

$$V = \Delta V \cdot D = 4.88\text{mV} \cdot D$$

הנוסחה לחישוב הטמפרטורה היא:

$$T = \frac{V}{10\text{mV}} - 50[^{\circ}\text{C}]$$

הערך 50 המופיע בנוסחה הוא ערך לתיקון חישוב הטמפרטורה ב- 25°C , כפי שהוסבר לעיל.

¹ ראה נספח א: שימוש בסביבת הסימולציה של TinkerCad

להלן התכנית:

```

void setup() {
  Serial.begin(9600);
}

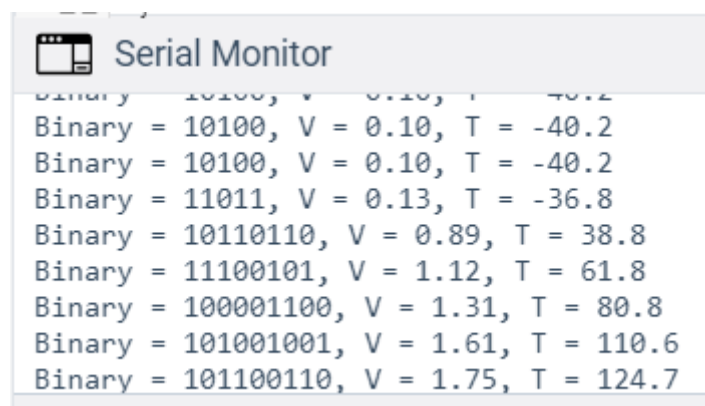
void loop() {
  int binValue;           // הערך הבינארי של הדגימה
  float voltage;          // ערך המתח של הדגימה
  float temprature;       // ערך הטמפרטורה של הדגימה
  binValue = analogRead(A0); // קריאת הערך האנלוגי
  voltage = 0.00488 * binValue; // חישוב המתח לפי הערך הבינארי וכושר ההבחנה
  temprature = voltage / 0.01 - 50; // חישוב ערך הטמפרטורה לפי המתח
                                // הצגת ערכי המשתנים עם כתוביות מתאימות

  Serial.print("Binary = ");
  Serial.print(binValue, BIN);
  Serial.print(", V = ");
  Serial.print(voltage);
  Serial.print(", T = ");
  Serial.println(temprature,1);
}

```

הפלט:

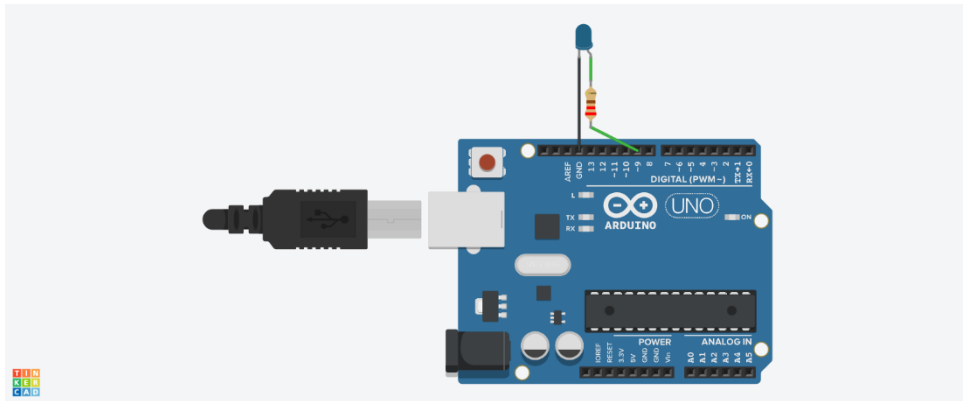
באיור 2.12 מתוארת תוצאת ההרצה בסימולציה של התכנית.



איור 2.12 – סימולציית הרצה של התכנית בטמפרטורות שונות

דוגמה 2.2 – עמעום LED בסימולציה TinkerCad

בדוגמה זו תוצג תכנית לשינוי הדרגתי של עוצמת LED ממצב כבוי לעוצמה מלאה, ולהפך.



איור 2.13 – תרשים חיבורים של LED ללוח

```
void setup(){
    pinMode(9, OUTPUT);          // הגדרת ההדק כפלט
}

void loop() {
    int bright;                  // משתנה לעוצמת ההארה

                                // לולאה המפעילה את הLED בעוצמה הולכת וגדלה
    for (bright = 0; bright <= 255; bright += 5) {
        analogWrite(9, bright); // PWM אות יצירת
        delay(50);
    }

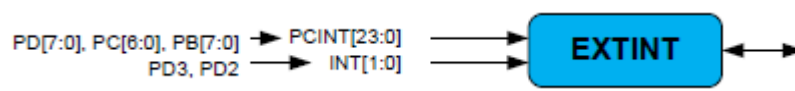
                                // לולאה המפעילה את הLED בעוצמה הולכת וקטנה
    for (bright = 255; bright >= 0; bright -= 5) {
        analogWrite(9, bright);
        delay(50);
    }
}
```

2.8 בקר הפסיקות במיקרו-בקר ATmega328P

איור 2.14 מציג את יחידת הבקרה של פסיקות החומרה החיצוניות במיקרו-בקר.

כפי שרואים יש שתי קבוצות של הדקי כניסה הגורמות לפסיקה:

- INT[1:0] – שני הדקים ייעודיים המדווחים על אירוע המחייב תגובה מיידי. כל הדק מפעיל שגרת פסיקה משלו.
- PCINT[23:0] – 24 הדקים המדווחים על שינוי ברמה הלוגית. קבוצת הדקים זו מחולקת לשלוש תתי קבוצות הדקים. כל תת קבוצה מפעילה שגרת פסיקה אחרת המשותפת לכל ההדקים.



איור 2.14 – בקר הפסיקות במיקרו בקר

בקר הפסיקות מסוגל לזהות בקשות פסיקה לפי שלוש אפשרויות:

- **עלייה** (Rising) – האות החשמלי משנה את ערכו מרמה לוגית 0 ל-1.
- **ירידה** (Falling) – האות החשמלי משנה את ערכו מרמה 1 לרמה 0.
- **רמה נמוכה** (Low Level) – האות החשמלי מוחזק ברמה הנמוכה 0.

אם הדק מוגדר כפלט (OUTPUT), שינוי במצבו יכול לעורר בקשת פסיקה. אפשרות זו פותחת פתח להפעלת שגרת פסיקה גם באמצעות הוראה בתוכנה.

בפרק 2.9 נדון רק בקבוצת הפסיקה הראשונה INT[1:0].

שם פסיקה	מספר הדק בלוח	הדק במיקרו-בקר
INT0	2	PD2
INT1	3	PD3

טבלה 2.4 – מיפוי הדקי הפסיקה

2.9 פעולות פסיקה בסביבת הפיתוח

כאמור, כאשר מתרחש אירוע פסיקה חומרתי באחד מההדקים 2 או 3 בלוח הפיתוח מופעלת שגרת פסיקה (ISR – Interrupt Service Routine) המתאימה להדק. להלן פירוט על פעולות הפסיקה השונות:

פעולת attachInterrupt

כותרת הפעולה:

```
void attachInterrupt(pin, ISR, mode)
```

הפעולה קובעת את אופן קבלת הפסיקה בהדק הנקוב וכן איזו שגרת פסיקה תופעל כאשר יתרחש אירוע פסיקה. לפעולה זו יש שלושה פרמטרים:

- pin – מספר הדק הפסיקה:

מומלץ, מטעמי תאימות, להשתמש במאקרו digitalPinToInterrupt המאפשר להמיר את מספר ההדק בלוח הפיתוח למספרו במיקרו-בקר.

- ISR – שם שגרת הפסיקה:

פעולה חסרת פרמטרים שאינה מחזירה ערך.

- mode – האופן שבו תתקבל שגרת הפסיקה:

אופן	תיאור
LOW	כאשר ההדק ברמה נמוכה
RISING	כאשר מתבצע שינוי של עלייה
FALLING	כאשר מתבצע שינוי של ירידה
CHANGE	כאשר מתבצע שינוי

לפניהם זימון לדוגמה:

```
attachInterrupt(digitalPinToInterrupt(2), myF, RISING);
```

זימון זה קובע שכאשר תתרחש עלייה של אות מתח בהדק 2 תופעל בלוח הפיתוח שגרת פסיקה myF.

פעולת detachInterrupt

כותרת הפעולה:

```
void detachInterrupt(pin)
```

הפעולה חוסמת את קבלת הפסיקה הנתונה. לפעולה זו יש פרמטר יחיד:

- pin – מספר הדק הפסיקה:

מומלץ, מטעמי תאימות, להשתמש במאקרו digitalPinToInterrupt המאפשר להמיר את מספר ההדק בלוח הפיתוח למספרו במיקרו-בקר.

פעולת noInterrupts

כותרת הפעולה:

```
void noInterrupts()
```

הפעולה מונעת את קבלת כל הפסיקות.

שימו לב! שימוש בפסיקה יפסיק את פעילותן התקינה של פעולות התלויות במנגנון הפסיקה, כגון: delay.

פעולת interrupts

כותרת הפעולה:

```
void interrupts()
```

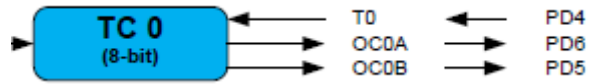
הפעולה מאפשרת את כל הפסיקות.

volatile

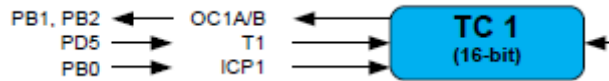
מילת מפתח זו מופיעה בעת הגדרת משתנה. היא מאלצת את המהדר להגדיר את המשתנה במרחב זיכרון ה-RAM ולא בזיכרון זמני (אוגרים). אם נעשה שימוש במשתנים הגלובליים בשגרת פסיקה, חשוב להגדיר אותם בתכנית באמצעות מילת המפתח.

2.10 מונים

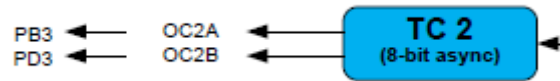
בפרק 2.2 תיארנו את המבנה הפנימי של המיקרו-בקר. באיור 2.15 (א, ב ו-ג) מתוארים שלושת המונים המשמשים גם כקוצבי זמן:



איור 2.15 א – מונה קוצב זמן 0



איור 2.15 ב – מונה קוצב זמן 1



איור 2.15 ג – מונה קוצב זמן 2

מונים 0 ו-2 הם מונים של 8 סיביות, ומונה 1 – של 16 סיביות.

לפניכם מפת הדקי המונים שתוארו באיורים בלוח הפיתוח:

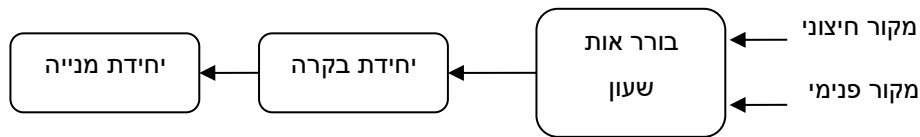
מפתח/סיבית	0	1	2	3	4	5	6	7
PB	8	9	10	11	12	13	C	C
	ICP1	OC1A	OC1B	OC2A				
		PWM	PWM	PWM				
PD	0	1	2	3	4	5	6	7
				OC2B	T0	OC0B/T1	OC0A	
				PWM		PWM	PWM	

טבלה 2.5 – מפת הדקי המונים בלוח הפיתוח

יש לציין כי לחלק מהדקי המיקרו-בקר יש תפקידים רבים. למשל, הדק PD5 יכול לשמש כהדק ספרתי, או כהדק יציאה אנלוגי וגם ככניסה למונה 1. המשמעות של כמה תפקידים היא שהגדרה מסוימת עלולה לבטל הגדרות אפשריות אחרות.

מבנה יחידת מנייה

באיור 2.16 מתואר המבנה העקרוני של מונה במיקרו-בקר:



איור 2.16 – מבנה יחידת מנייה

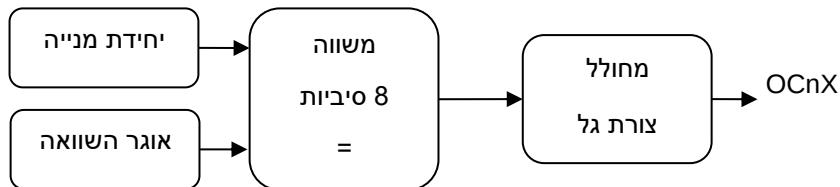
לכל מונה יש יחידת מנייה פנימית בגודל 8 סיביות או 16 סיביות. יחידת המנייה מחוברת אל יחידת בקרה הקובעת את איפוס המונה; את כיוון המנייה: מעלה או מטה; ואת ביצוע פעולת השינוי: קידום המונה באחד או החסרה באחד.

ההבדל בין מונה ובין קוצב זמן הוא במקור אות השעון:

- מקור חיצוני – התייחסות כאל יחידת מנייה של אירועים אקראיים או סדורים.
- מקור פנימי – אות שעון בתדר שהוא כפולה של אות השעון של המיקרו-בקר.

יחידת ההשוואה

באיור 2.17 מתוארת יחידת ההשוואה. בכל מונה קיימות שתי יחידות השוואה A ו-B. כל יחידה היא עצמאית. באמצעות כל אחת מהיחידות אפשר לייצר את אותות PWM.



איור 2.17 – מבנה יחידת ההשוואה

OCnX הוא מציג אחת מ-6 יציאות של יחידות ההשוואה. להלן המקרא:

n – מייצג את מספר המונה 0, 1 או 2.

X – מייצג את הערוץ A או B.

OC (Output Compare) – הדק מוצא של יחידת ההשוואה.

שלושת המונים 0, 1 ו-2 משמשים את סביבת הפיתוח IDE כדי לספק פעולות כמו השהייה ויצירת אותות PWM המשמשים כיציאה אנלוגית.

- מונה 0 משמש לפעולות השהייה כמו `delay`, `millis`, ו-`micros`. גישה ישירה אל אוגרי המיקרו-בקר תשפיע על פעולות קוצב הזמן.
- מונה 1 משמש את ספריית `Servo`.
- מונה 2 משמש ליצירת צליל באמצעות הפעולה `tone`.

שימוש באחד המונים למטרות אחרות יגרום להפסקת העבודה הרגילה של סביבת הפיתוח. למשל, שימוש בקוצב זמן 1 עלול לגרום להפסקת האפשרות להשתמש ביציאת PWM בהדקים 9 ו-10 בלוח הפיתוח כיציאות אנלוגיות (`analogWrite`).

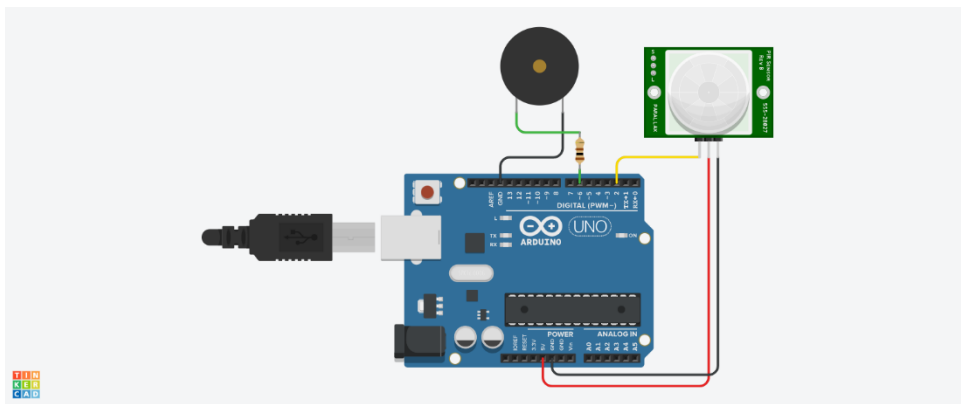
דוגמה 2.3 – יצירת צליל התראה לאחר שחיישן PIR מזהה תנועה בסימולציה

TinkerCad

בדוגמה זו נציג תכנית המשמיעה צליל בתדר 1kHz ומדליקה נורה כאשר חיישן PIR מזהה תנועה.

באיור 2.18 מתואר תרשים חיבורים של חיישן PIR ורמקול Piezo. כאשר החיישן מזהה תנועה הוא מייצר דופק חשמלי למשך זמן קצוב. כל עוד יש דופק יושמע צליל בתדר 1kHz.

גלאי התנועה, PIR, פועל על עיקרון של זיהוי גוף חם בתנועה. אם גוף חם, חסר תנועה, נמצא באזור הגילוי גלאי התנועה לא יזהה אותו. הגוף יזוהה רק אם הוא בטמפרטורה גבוהה מסביבתו ונמצא בתנועה.



איור 2.18 – תרשים חיבורים של חיישן PIR ורמקול Piezo

```
void setup() {
  pinMode(2, INPUT);           // הגדרת הדק מבוא לחיישן התנועה
  pinMode(13, OUTPUT);         // הגדרת הדק מוצא לנורת הלד
  pinMode(6, OUTPUT);          // הגדרת הדק מוצא לרמקול
}

void loop() {
  if (digitalRead(2) == HIGH) { // בדיקה האם זוהתה תנועה
    digitalWrite(13, HIGH);     // הדלקת הלד
    tone(6, 1000);              // השמעת צליל בתדר 1000 הרץ
    while (digitalRead(2) == HIGH); // המתנה לסיום דופק זיהוי התנועה
    digitalWrite(13, LOW);      // כיבוי הלד
    noTone(6);                  // הפסקת הצליל
  }
}
```

בפעולת ה-loop התבצעה קריאה של הדק הכניסה אליו מחובר חיישן ה-PIR. אם ערכו של חיישן התנועה היה גבוה (HIGH) הושמע צליל ונורת הled הודלקה.

לאחר מכן התבצעה לולאת המתנה עד שערכו של החיישן עבר למצב נמוך (LOW) ואז הופסק הצליל ונורת הled כובתה.

הגישה התכנותית שננקטה, הוצגה לראשונה במבוא למערכות משובצות מחשב, נקראת שיטת התשאול (polling). על פי גישה זו מתבצעת לולאת המתנה עד לקיום תנאי סיום. בזמן לולאת ההמתנה לא מתבצעות פעולות אחרות. החיסרון העיקרי של שיטה זו הוא בהתייחסות לאירוע אחד והזנחת אירועים אחרים במערכת במידה וקיימים. כלומר, אם צריך לטפל בשני אירועים שונים ואירוע אחד נמצא כעת בטיפול, לא נוכל לטפל באירוע האחר עד שלא נסיים את הטיפול באירוע הנוכחי.

דוגמה 2.4 – יצירת צליל התראה באמצעות פסיקה לאחר שחיישן PIR מזהה**תנועה בסימולציה TinkerCad**

דוגמה זאת חוזרת על דוגמה 2.3 ומוסיפה לה פסיקה.

זו נציג תכנית המשמיעה צליל בתדר 1kHz ומדליקה נורה כאשר חיישן PIR מזהה תנועה.

בתכנית להלן, במקום להשתמש בשיטת התשאול (Polling) בפעולה loop, נשתמש במנגנון הפסיקות.

```
void setup() {
  pinMode(2, INPUT);
  pinMode(13, OUTPUT);
  pinMode(6, OUTPUT);

  // קישור הדק 2 לשגרת פסיקה המגיבה לשינוי במצב ההדק
  attachInterrupt(digitalPinToInterrupt(2), myISR, CHANGE);
}

// שגרת פסיקה

void myISR(){
  if (digitalRead(2) == HIGH) {      // אם זוהתה תנועה
    digitalWrite(13, HIGH);          // הדלק נורה
    tone(6, 1000);                   // השמע צליל
  }
  else {                             // אם אין תנועה
    digitalWrite(13, LOW);           // כבה נורה
    noTone(6);                       // הפסק צליל
  }
}

// פעולה ריקה

void loop() {
}
```

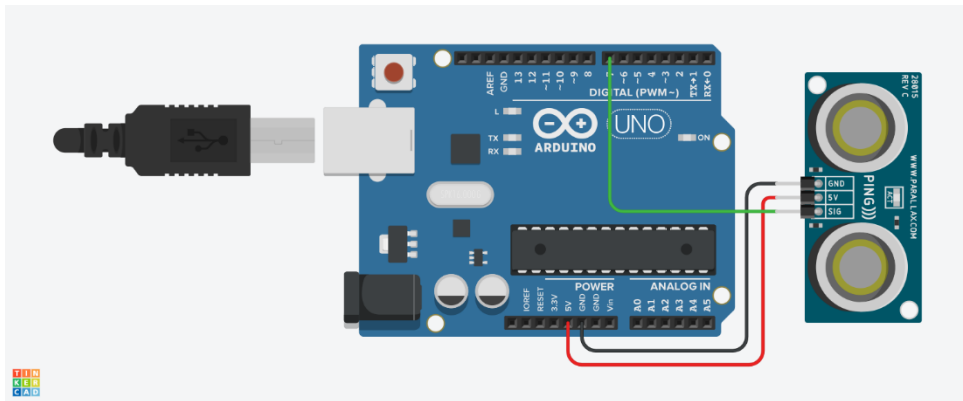
דוגמה 2.5 – גלאי מרחק חיישן אולטרה סוני בסימולציה TinkerCad

בדוגמה זו נראה תכנית המציגה את המרחק של גוף כלשהו מהחיישן. המרחק יוצג ביחידות של ס"מ.

איור 2.19 מציג תרשים חשמלי של מד מרחק אולטרה סוני. לרכיב 3 הדקים:

- Vcc, Gnd – מתח אספקה.
- Signal – הדק דו-כיווני המשמש לכניסה וליציאה.

כדי למדוד מרחק, מספקים להדק Signal דופק הפעלה ברוחב זמן של 10 מיקרו-שניות. לאחר מכן מודדים את משך הזמן של הדופק המתקבל בחזרה בהדק זה. משך הזמן הוא מדד למרחק של הגוף מהחיישן. חיישן המרחק משדר צליל בתדר על-קולי ומחשב את הזמן שלוקח לגל לחזור אליו. ולכן הזמן המתקבל הוא של מרחק כפול (הלך וחזור).



איור 2.9 – תרשים חיסורים של חיישן מרחק אולטרה סוני

הפעולה `pulseIn`

`pulseIn(pin, value)`

הפעולה מקבלת שני פרמטרים ומחזירה את משך הזמן שבו ההדק נמצא בערך המוגדר בפרמטר.

pin – מספר ההדק הספרתי מ-0 עד 13

value – HIGH או LOW.

למשל: אם הערך הוא HIGH, הפעולה תמתין למעבר מ-LOW ל-HIGH ותתחיל למדוד את הזמן עד לירידה בחזרה לערך LOW. הפעולה תציג את הזמן במיקרו-שניות שבו הדופק היה ב-HIGH.

הפעולה יכולה למדוד זמנים מ-10 מיקרו שניות עד ל-3 דקות.

לפעולה יש גרסת הפעלה עם פרמטר שלישי של `timeout`.

`pulseIn(pin, value, timeout)`

timeout – הזמן הקצוב במיקרו-שניות.

אם הדופק בהדק אינו משתנה בפרק זמן קצוב של 1 שנייה (`timeout`), הפעולה מחזירה 0.

התכנית

```

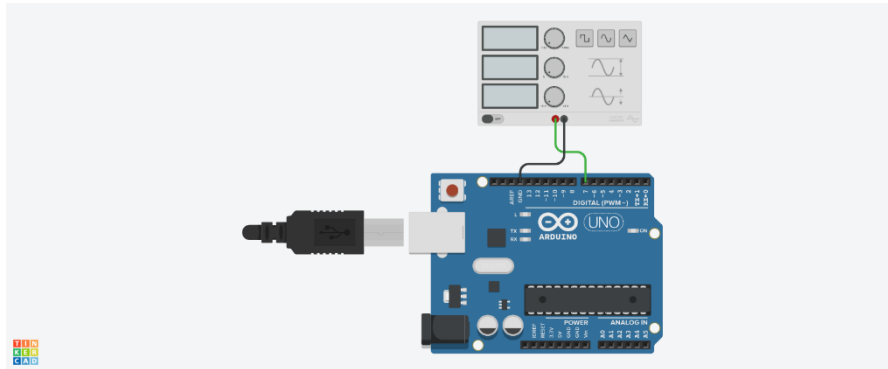
int cm = 0;           // מרחק בס"מ
void setup() {
  pinMode(7, INPUT);
  Serial.begin(9600);
}
void loop() {
  // שליחת דרבון להפעלת מערכת המדידה
  pinMode(7, OUTPUT); // הגדרה כהדק יציאה
  digitalWrite(7, LOW); // הוצאת ערך נמוך
  delayMicroseconds(2);
  digitalWrite(7, HIGH); // הוצאת ערך גבוה – דרבון
  delayMicroseconds(10); // משך הדרבון 10 מיקרו-שניות
  digitalWrite(7, LOW); // סיום אות הדרבון
  // מדידת משך זמן הדופק
  pinMode(7, INPUT); // הגדרה כהדק כניסה
  // זימון פעולה למדידת משך הזמן בגבוה
  // חישוב המרחק בס"מ
  cm = 0.01723 * pulseIn(7, HIGH);
  Serial.print(cm);
  Serial.println("cm");
  delay(100);
}

```


דוגמה 2.6 – מד תדר בסימולציה Tinkercad

כתבו תכנית המציגה את התדר של גל ריבועי.

איור 2.10 מציג את תרשים החיבורים של מחולל גל אל לוח UNO.



איור 2.10 – תרשים החיבורים של מחולל גל

```

long th = 0; // משתנה לזמן שבו האות בגבוה
long tl = 0; // משתנה לזמן שבו האות בנמוך
int freq; // משתנה לתדר

void setup() {
    pinMode(7, INPUT); // הגדרת הדק כניסה
    Serial.begin(9600);
}

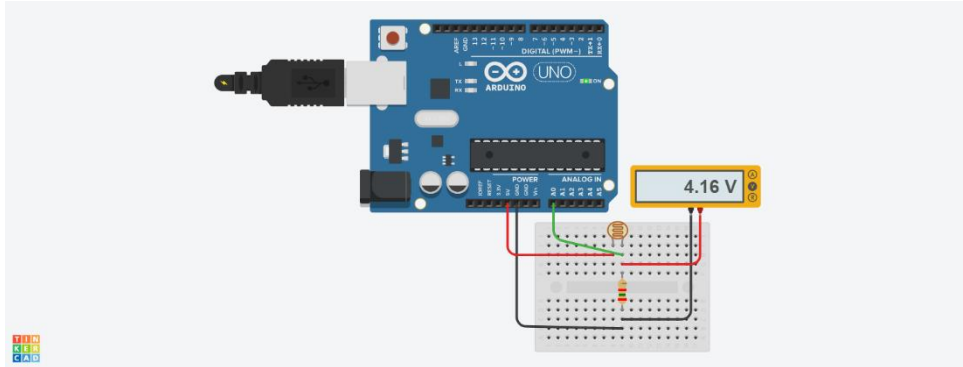
void loop() {
    th = pulseIn(7, HIGH); // מדידת הזמן שבו האות הגבוה
    tl = pulseIn(7, LOW); // מדידת הזמן שבו האות בנמוך
    freq = 1000000/(th+tl); // חישוב התדירות f = 1/T = 1/(th + tl)
    Serial.print("Freq = ");
    Serial.println(freq);
}

```

שאלות חזרה

שאלה 2.1

מחברים את המעגל שלפניכם בסימולציה:



ערך הנגד הוא $2.5k\Omega$.

- א. בנו את המעגל בסימולטור.
- ב. כתבו תכנית המציגה את הערך הבינארי ואת המתח של המעגל המתואר.

שאלה 2.2

מחברים מעגל כמו בדוגמה 2.1, אך הוחלט לשנות את מתח הייחוס. כדי לשנות את מתח הייחוס חיברו את ההדק AREF לנקודת המתח של 3.3V בלוח הפיתוח.

- א. מהי ההוראה שצריך להוסיף כדי שמתח הייחוס ישתנה לפי הדרישה?
- ב. חשבו את כושר ההבחנה החדש.
- ג. שנו את התכנית ובדקו אותה בסימולטור.

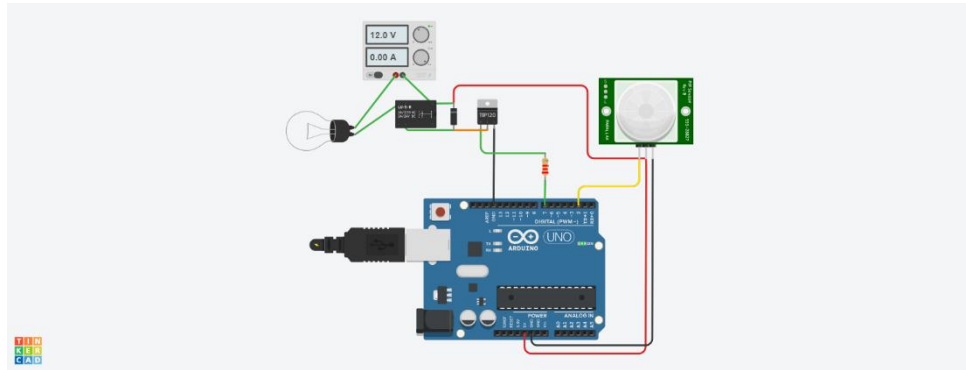
שאלה 2.3

דרוש מעגל הכולל דיודת RGB LED ו-3 פוטנציומטרים של $10k\Omega$ כל אחד. הפוטנציומטרים מחוברים לכניסות אנלוגיות ואילו הדקי RGB LED מחוברים ליציאות אנלוגיות.

- א. תכננו ובנו מעגל בסימולטור התואם לדרישות לעיל.
- ב. כתבו תכנית הפעלה.

שאלה 2.4

נתון התרשים החשמלי שלפניכם:



התרשים מתאר מערכת להפעלת תאורה אוטומטית. כאשר מזוהה תנועה, מופעלת התאורה למשך דקה אחת. אם מזוהה תנועה תוך כדי ההפעלה של התאורה, יחודש זמן ההפעלה לדקה אחת. אם כעבור דקה, לא מזוהה תנועה התאורה תופסק.

א. מה תפקידו של הטרנזיסטור הביפולרי TIP120?

ב. מדוע נחוץ ממסר?

ג. כתבו תכנית למימוש המערכת.

שאלה 2.5

מוספים לדוגמה 2.5 לעיל רמקול Piezo על פי התרשים מדוגמה 2.3.

כתבו תכנית המשמיעה צליל בתדר הולך וגדל ככל שהמרחק מתקצר, וצליל נמוך יותר ככל שהמרחק ארוך יותר. הצליל יופסק, אם אין זיהוי של עצם מול החיישן האולטרה סוני.

שאלה 2.6

כדי לשפר את הדיוק של מד התדר מדוגמה 2.6 הוחלט למדוד 10 מחזורים ולהציג את ממוצע המדידות.

כתבו תכנית לשיפור המדידה.

סיכום פרק 2

בפרק 2 למדתם את הנושאים האלה:

1. מבנה עקרוני של מיקרו-בקר
2. המבנה הפנימי של המיקרו-בקר ATmega328P
3. תכונות המיקרו-בקר ATmega328P
4. הדקי כניסה ויציאה ספרתיים
5. ממיר ADC במיקרו-בקר
6. יצירת אותות PWM
7. הדקי כניסה תקביליים
8. הדקי יציאה תקביליים
9. שליטה בעצמת הארה של LED
10. יצירת צלילים
11. מדידת זמן דופק
12. מדידת תדר
13. פסיקות חומרה

פרק 3: תקשורת טורית תקן RS232

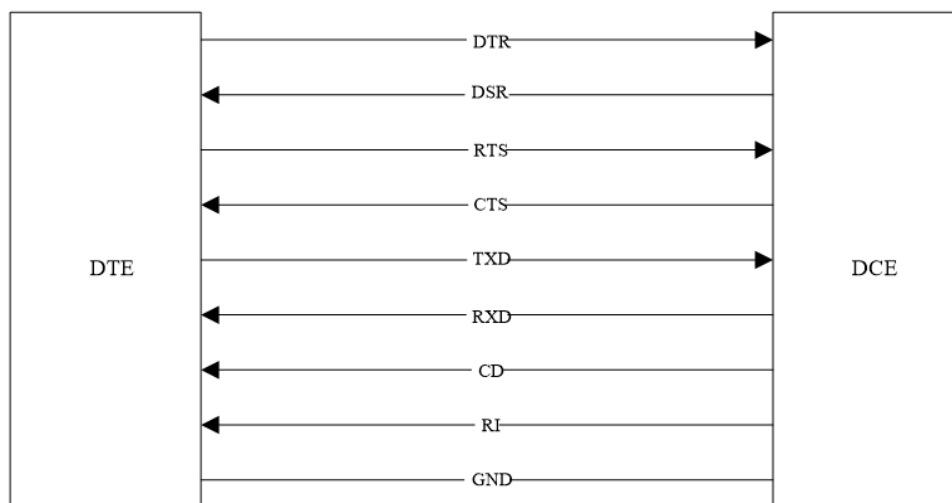
כפי שלמדנו ביחידה מערכות משובצות מחשב, המונח תקשורת טורית מתאר תהליך של העברת נתון סיבית אחת בכל פעם, באופן סדרתי דרך ערוץ תקשורת אחד.

3.1 תקן TIA232

אחד מתקני התקשורת הטורית הוותיקים הוא תקן RS232 שהוצג לראשונה בשנת 1962. כיום יש לו גרסת עדכון F, והתקן נקרא TIA232. במקור נועד תקן זה לחבר בין ציוד תקשורת (מודם) ובין ציוד קצה (מחשב, מסוף). בפועל השתמשו בתקן זה גם כדי לחבר שתי יחידות קצה זו לזו (חיבור בין שתי מערכות ממוחשבות). מרבית המחשבים האישיים אינם כוללים חיבורי RS232, וכדי להשתמש בתקן זה יש להשתמש במתאם USB-to-RS232.

קווי החיבור

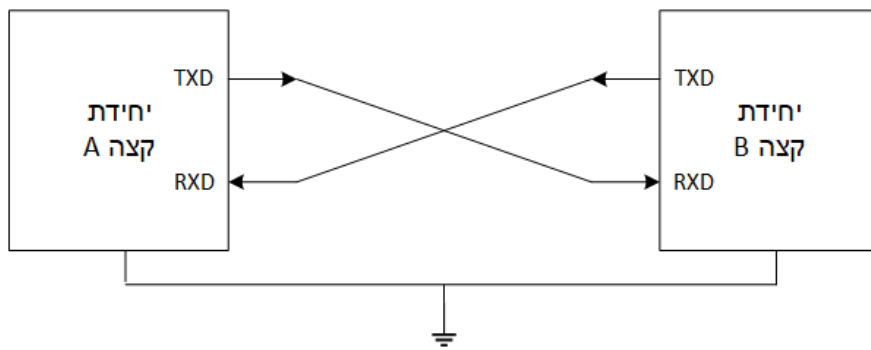
תקן TIA232 מגדיר את קווי החיבור בין יחידות הקצה. קווי החיבור, על פי התקן, כוללים התייחסות לממשק בין ציוד התקשורת – DCE (מודם) ובין ציוד הקצה – DTE (מחשב).



איור 3.1 – החיבור בין יחידת קצה לציוד תקשורת.

הקווים TXD ו-RXD משמשים לצורכי שידור וקליטה של הנתונים. הקווים DTR, DSR, CTS, ו-RTS משמשים לבקרת הזרימה של הנתונים. הקווים CD ו-RI משמשים לציון מצבו של קו התקשורת.

כאשר מחברים שתי יחידות קצה זו לזו ללא ציוד תקשורת מבטלים את השימוש בהדקים האחרים המשמשים לציון מצב המודם ובקרת הזרימה. בפועל נשארים שלושה קווי חיבור בלבד. איור 3.2 מציג את אופן החיבור של שתי יחידות קצה באמצעות שלושה קווים בלבד.



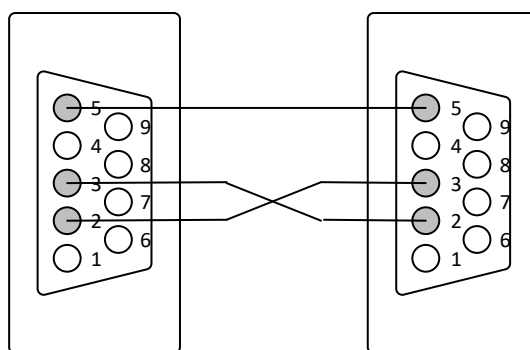
איור 3.2 – חיבור בין שתי יחידות קצה

בטבלה להלן מתוארים תפקידי ההדקים בחיבור של שתי יחידות קצה:

קו שיחידת הקצה משדרת	Transmit Data	TXD
דרכו		
קו שיחידת הקצה קולטת דרכו	Receive Data	RXD

המידע שמשדרת יחידת קצה אחת בקו TXD נקלט ביחידת הקצה השנייה בקו RXD. חיבור כזה נקרא חיבור מוצלב. כיוון שקיים ערוץ נפרד לשידור TXD וערוץ נפרד לקליטה RXD אפשר לקיים שידור דו-כיווני במקביל. שיטת שידור זו נקראת Full Duplex.

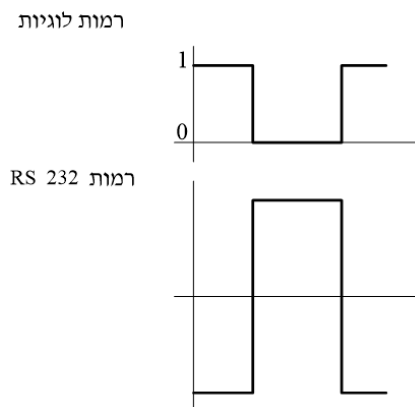
החיבור הפיזי בין יחידות הקצה נעשה באמצעות כבל חשמלי. בקצה כל כבל קיים מחבר (connector) המשמש לחיבור אל יחידת הקצה. באיור 3.3 מופיע מבנה מחבר טיפוסי של 9 הדקים, ובפועל משתמשים ב-3 מתוך ה-9 בלבד.



איור 3.3 – חיבור הדקים

רמות המתח

תקן TIA232 מגדיר את רמות המתחים לרמות הלוגיות. ברמה לוגית 0 טווח המתחים במשדר הוא בין 5V ל-15V, וברמה לוגית 1 תחום המתחים במשדר הוא בין 5V ל-15V. ברמה לוגית 0 תחום המתחים במקלט הוא בין 3V ל-15V ועבור הרמה הלוגית 1 תחום המתחים במקלט הוא בין 3V ל-15V. טווח המתחים בין 3V ל-3V אינו מוגדר בתקן ויש לדאוג שהמעבר דרכו יהיה מהיר ככל האפשר ביחס לקצב השידור. מתחים אלה נמדדים ביחס לקו הייחוס GND (Ground) המייצג את רמת המתח 0V. קווים המיוחסים לקו GND נקראים קווים לא מאוזנים (Unbalanced Line). באיור 3.4 משורטטות באופן עקרוני, זו מתחת לזו, דיאגרמות המתארות את רמות המתחים הלוגיות ואת רמות המתחים המתאימות בתקן RS232.



איור 3.4 – רמות לוגיות ורמות מתחים בתקן RS232

המתח המרבי המוגדר בתקן RS232 הוא $\pm 25V$ בקו פתוח. אותות המתח הנפוצים הם: $\pm 5V, \pm 10V, \pm 12V$ ו- $\pm 15V$, והם נקבעים באמצעות מעגל דחיפה (Line Driver). ישנם מעגלי דחיפה המספקים את רמות המתחים בהתאם לתקן וניזונים מספק אחד של 3V או 5V. מעגלי הדחיפה צריכים לעמוד בתנאים קיצוניים של קצר בקווים ובמתח שיא של $\pm 25V$.

על מעגלי הדחיפה לעמוד ב-Slew Rate, מהירות מעבר בין רמות, של $30V/\mu S$. שינויים מהירים יותר עלולים לגרום להפרעות אלקטרומגנטיות בין המוליכים (Crosstalk).

העומס המחובר לכל קו לא יהיה נמוך מ- $3k\Omega$ ולא גבוה מ- $7k\Omega$.

סיכום מאפייני מתח

- מתחי כניסה בתחומים -3V עד -15V, 3V עד 15V
- מתח שיא בכניסה $\pm 25V$
- תחום מתחים -3V עד 3V אינו מוגדר בתקן
- קווי השידור מוגנים מקצר

אורך כבל

אורכו של הכבל המרבי הוא בין 15 ל-20 מטר. האורך תלוי בקיבול של הכבל.

קצב מרבי

הקצב המרבי הוא 120kbit/s.

סוגי תקשורת

התקן EIA232 תומך בשתי שיטות שידור: סינכרונית וא-סינכרונית. בשיטת השידור הסינכרונית למשדר ולמקלט יש קווי אות שעון נפרדים. אותות השעון נועדו לתאם בין הצד המשדר ובין הצד הקולט. בתקשורת א-סינכרונית לא מסופק אות שעון, ולכן נדרשת שיטה אחרת כדי לתאם בין המשדר למקלט. בשיטה הא-סינכרונית קו השידור הוא במצב לא פעיל, ועובר באופן זמני למצב המציין את תחילת השידור. לאחר ציון תחילת השידור המידע מועבר. בסוף השידור הקו חוזר למצב לא פעיל. תקשורת א-סינכרונית מאופיינת בשידור במבנה של מסגרות – Frames. מרבית מערכות המחשבים, ובכללם המחשב האישי, תומכים בתקשורת א-סינכרונית בלבד. בפרק זה נדון בתקשורת א-סינכרונית בלבד.

מסגרת שידור

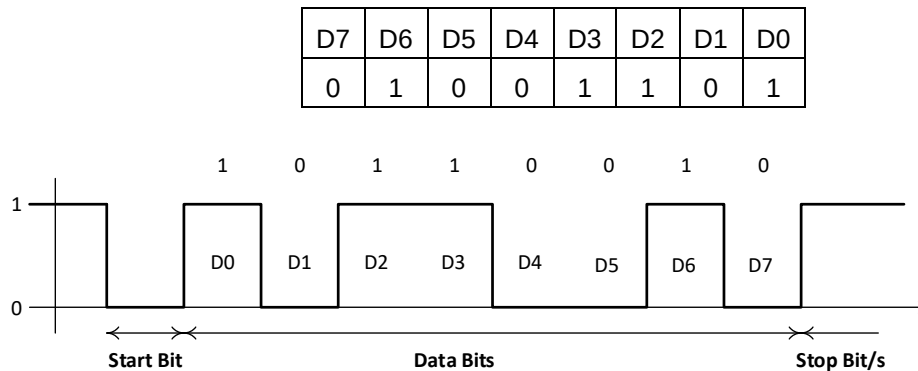
בתקשורת א-סינכרונית המידע מועבר בין יחידות הקצה לפי הצורך. כלומר, רק כאשר יש מידע להעברה מתחיל תהליך של שידור. כאשר אין מידע לשידור, קו השידור במצב לא פעיל. מצב קו לא פעיל מאופיין ברמה לוגית 1.

המידע שנועד להעברה מפוצל, לפי הצורך, ליחידות מידע. כל יחידת מידע משודרת בנפרד במבנה הנקרא מסגרת מידע. תקן RS232 תומך בהעברת יחידות מידע בגודל של 5–9 סיביות. כיום, במרבית היישומים נהוג להשתמש ב-8 סיביות מידע. תקן RS232 אינו מתייחס לתקינות העברת המידע כולו, אלא נותן מענה בסיסי בלבד לזיהוי תקינות העברת יחידת מידע. תקינות העברת המידע נבדקות באמצעות השיטה בדיקת זוגיות (Parity Check).

כל מסגרת שידור מתחילה במעבר ממצב לא פעיל למצב פעיל, מ-1 ל-0. מצב זה נשמר במשך זמן קצוב המאפיין את זמן השידור של סיבית בודדת. סיבית זו נקראת סיבית התחלה (Start bit). לאחר סיבית ההתחלה משודרות בזו אחר זו הסיביות של יחידת המידע. תחילה משודרת הסיבית המשמעותית פחות (LSB), ובסוף משודרת הסיבית המשמעותית יותר (MSB). לאחר סיביות המידע אפשר להוסיף סיבית ביקורת. שידור המסגרת מסתיים לאחר החזרת הקו למצב לא פעיל למשך זמן של סיבית אחת לפחות. הסיבית האחרונה נקראת סיבית הסיום – אות סיבית העצירה (Stop bit). יכולות להיות מספר סיביות עצירה.

דוגמה 3.1 – שידור התו 'M'

איור 3.5 מציג את העברת התו 'M' תוך שימוש ב-8 סיביות מידע וללא סיבית ביקורת. לתו 'M' קוד ASCII שערכו $77=4DH=01001101B$.



איור 3.5 – שידור התו 'M' ביחס לזמן

בדיקת זוגיות

תקן RS232 מאפשר הוספת סיבית נוספת למסגרת לשם בדיקת מהימנות המידע המועבר. השיטה מבוססת על מניין מספר הסיביות שערכן 1 ביחידת המידע. בפועל אין חשיבות לתוצאת המניין מבחינה כמותית, אלא לתכונת הזוגיות שלה. למשל, לכל המספרים הבינאריים התלת-ספרתיים האלה: 011, 101, 110 וכן 000 יש מספר זוגי של 1. ולכל המספרים האלה: 001, 010, 100, 111 יש מספר אי-זוגי של 1.

בדיקת זוגיות מסוג זוגי – EVEN PARITY

בשיטה זו מונים את מספר הסיביות שערכן 1 ביחידת המידע. אם התוצאה זוגית **מוסיפים** סיבית ביקורת 0. אם התוצאה אי-זוגית **מוסיפים** סיבית ביקורת 1.

בדיקת זוגיות מסוג אי-זוגי – ODD PARITY

בשיטה זו מונים את מספר הסיביות שערכן 1 ביחידת המידע. אם התוצאה זוגית **מוסיפים** סיבית ביקורת 1. אם התוצאה אי-זוגית **מוסיפים** סיבית ביקורת 0.

סיבית סיום

כאמור, כל מסגרת מידע מסתיימת במצב לא פעיל, שהוא 1 לוגי. מצב זה מציין סוף שידור של מסגרת והכנה לקראת השידור הבא אחריה. יחידת הזמן שבה המערכת במצב לא פעיל מוגדרת באמצעות משך הזמן של שידור סיבית. התקן קובע אפשרות של 1, $1\frac{1}{2}$ או 2 יחידות זמן של שידור סיבית שהמשדר ימצא בה במצב לא פעיל. השימוש במשך זמן גדול מזמן שידור סיבית נועד ליחידות קצה איטיות הדורשות משכי עיבוד גדולים יותר משל המשדר. במרבית המערכות סיבית עצירה אחת מספיקה.

קצב תקשורת Baud Rate

אחד ממאפייני התקשורת החשובים הוא מהירות העברת המידע בין יחידות הקצה. מאפיין זה קובע את כמות המידע שאפשר להעביר בשנייה אחת. כיוון שהמידע מועבר במבנה של מסגרות שאורכן לא אחיד, מודדים את הקצב במספר הסיביות המשודרות בשנייה אחת. יחידות הקצב נקראות סל"ש (סיביות לשנייה) או bps (bit per second). בתקן RS232 מקובלות היחידות baud לתיאור קצב הסיביות. להלן קצב התקשורת המקובלים:

110 ; 150 ; 300 ; 1,200 ; 2,400 ; 4,800 ; 9,600 ; 19,200 ; 38,400 ; 57,600 ; 115,200 ; 230,400 ; 460,800 ; 921,600.

זמן סיבית מחושב לפי הנוסחה הזו:

$$t_{BIT} = \frac{1}{baud} \text{ [Sec]}$$

זמן מסגרת מחושב לפי הנוסחה הזו (n הוא אורך המסגרת בסיביות):

$$t_{FRAME} = n \times t_{BIT} = n \times \frac{1}{baud} = \frac{n}{baud} \text{ [Sec]}$$

קצב העברה מרבי הוא מספר המסגרות המשודרות במשך שנייה אחת ברצף, בזו אחר זו

ללא הפסקה. קצב העברת המסגרות נמדד ביחידות של $\frac{Frames}{Sec}$, או ביחידות $\frac{Char}{Sec} = cps$.

קצב ההעברה המרבי מחושב בעזרת הביטוי הזה:

$$Rate_{MAX} = \frac{1}{t_{FRAME}} = \frac{baud}{n} \text{ [cps]}$$

דוגמה 3.2 – חשובי זמנים וקצב

ערוץ תקשורת משדר בתקן RS232 בקצב baud=4800bps, 8 סיביות מידע וסיבית עצירה אחת. משדרים את המידע ללא סיבית ביקורת.

- א. חשבו את זמן הסיבית.
- ב. חשבו את מספר הסיביות במסגרת מידע.
- ג. חשבו את זמן המסגרת.
- ד. חשבו את הקצה המרבי של העברת המידע.

פתרון

- א. $t_{BIT} = \frac{1}{baud} = \frac{1}{4800} = 208.3 \mu Sec$
- ב. $10 = 1 \text{ סיבית עצירה} + 8 \text{ סיביות מידע} + n \text{ סיבית התחלה}$
- ג. $t_{FRAME} = \frac{n}{baud} = \frac{10}{4800} = 2.083 mSec$
- ד. $Rate_{MAX} = \frac{baud}{n} = \frac{4800}{10} = 480 \text{ cps}$

סיכום מאפייני תקשורת:

- המידע המשודר מחולק ליחידות מידע בסיסיות.
- כל יחידת מידע היא בעלת מספר סיביות מידע (DATA BITS) הנע בין 5 ל-9.
- כל יחידת מידע משודרת במבנה של מסגרת (FRAME).
- מסגרת מתחילה בסיבית התחלה 0.
- שידור מסגרת מסתיים בסיבית עצירה אחת או יותר.
- אפשר להשתמש בסיבית בקרות לאימות הנתון המתקבל.
- קצב העברת המידע נקרא baud rate.

יתרונות של שיטת התקשורת

- נתמך בהתקנים מסורתיים משום ששיטת התקשורת פשוטה.
- תומך במרחק גדול, 15–20 מטר, עם אפשרות גילוי שגיאות.
- חסינות גבוהה לרעש מתחי (השידור המינימלי $\pm 5V$).
- מחיר נמוך.
- קיימים גם מתאמי USB, מתאמי רשת ומתאמי RS485 במחירים נמוכים.

חסרונות של שיטת התקשורת

- מתאים לתקשורת בין מערכות, אך אינו מתאים לתקשורת בין שבבים או חיישנים.
- הקצב תלוי במרחק – במרחקים ארוכים הקצב נמוך ובמרחקים קצרים בקצב גבוה.
- נדרש מתאם רמות לוגיות נפרד, הגורם לעלויות גבוהות יותר.
- משמש במערכות בעלות שליט אחד ועבד אחד, ואינו מתאים לשמש במערכות שליט והתקני עבד.
- קווי התמסורת אינם מאוזנים (מתחים מיוחדים ל-GND).

תרגול 

שאלה 3.1

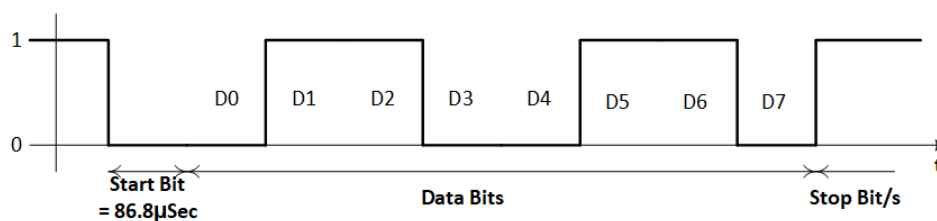
משדרים את התו 'Q' לפי פרמטרי התקשורת האלה:

8 סיביות מידע, סיבית עצירה אחת, ללא סיבית ביקורת, קצב של 9600bps.

- מהו הקוד של התו 'Q' לפי בסיס עשרוני, הקסדצימלי ובינארי?
- שרטטו דיאגרמת שידור של התו ביחס לזמן. ציינו בדיאגרמה את ערכי הרמות הלוגיות ואת ערכי הזמנים.
- מהו זמן המסגרת?
- כמה תווים אפשר לשדר ברציפות על פי פרמטרי התקשורת?

שאלה 3.2

נתונה דיאגרמת השידור שלפניכם:



- מהו הערך הבינארי המשודר? הציגו גם בבסיס 10 וגם בבסיס 16.
- מהו התו המשודר? היעזרו במנוע חיפוש.
- חשבו את קצב השידור.
- חשבו את זמן המסגרת.
- כמה תווים אפשר לשדר ברציפות על פי פרמטרי התקשורת?

3.2 תקשורת טורית UNO

קווי התקשורת בלוח



קווי
התקשורת

איור 3.6 – קווי תקשורת בלוח UNO

כפי שניתן לראות באיור 3.6 לוח הפיתוח כולל שני הדקי תקשורת RS232. הדקים אלה משמשים להפעלת תוכנת ה-monitor ולכן לא נוכל לחבר להדקים אלה יחידת תקשורת נוספת. אפשרות אחרת היא לחבר את התקן החומרה להדקים אחרים בלוח ולהשתמש במחלקה אחרת המדמה תקשורת טורית, הנקראת SoftwareSerial.

3.3 המחלקות Serial ו-SoftwareSerial

ביחידה מערכות משובצות מחשב הכרנו את המחלקה Serial ואת המחלקה SoftwareSerial. בטבלה 3.1 להלן מתוארות חלק מהפעולות של המחלקה Serial, ובטבלה 3.2 מתוארות חלק מהפעולות של המחלקה SoftwareSerial.

תיאור של חלק הפעולות של מחלקת Serial

הפעולה	תיאור הפעולה
void begin(long speed) void begin(long speed, int config)	הפעולה מאתחלת את התקשורת למהירות המוגדרת בפרמטר speed ואת פרמטרי התקשורת המוגדרים בפרמטר config . ברירת המחדל היא SERIAL_81 – 8 סיביות מידע, ללא סיבית בקורת וסיבית סיום אחת.
void end()	הפעולה מפסיקה את השימוש בהדקים TX ו-RX למטרות תקשורת. לאחר מכן אפשר להשתמש בהדקים למטרות קלט פלט ספרתי.
size_t print(val) size_t print(val, format)	הפעולה שולחת את הפרמטר val כמידע טקסטואלי דרך ערוץ התקשורת. כך יוצגו הערכים שיתקבלו: • ערך שלם יוצג כמספר שלם. • ערך ממשי יוצג בדיוק של שתי ספרות אחרי הנקודה העשרונית. • תו או מספר כבית יוצג כתו • מחרוזת תוצג כמחרוזת הפרמטר format קובע את אופן ההצגה של ערכים מספריים לפי בסיסים שונים או את אופן קביעת מספר הספרות אחרי הנקודה. הפעולה print מציגה את המידע בהמשך השורה והפעולה println מציגה את המידע ועוברת לשורה חדשה. הפעולה מחזירה את מספר הבתים שנכתבו. size_t מוגדר כמספר שלם וחיובי. הערה: כדי לשלוח נתון מספרי באופן בינארי השתמשו בפעולה write.
size_t write(byte val) size_t write(string str) size_t write(byte buf[], int len)	הפעולה משדרת את המידע באופן בינארי. הפרמטר val הוא נתון מסוג בית. הפרמטר str הוא נתון מסוג מחרוזת תווים. הפרמטר buf הוא מערך של בתים באורך בפרמטר len. הפעולה מחזירה את מספר הבתים שנכתבו.

size_t מוגדר כמספר שלם וחיובי.	
הפעולה קוראת ומחזירה את הערך שנקלט בערוץ התקשורת הטורית. אם אין מידע זמין, הפעולה תחזיר -1.	int read()
הפעולה מחזירה את מספר הבתים שנקלטו בערוץ התקשורת הטורית. יש לערוץ התקשורת הטורית חוצץ (buffer) בגודל 64 בתים.	int available()
הפעולה מחפשת מספר שלם תקין ומחזירה אותו. אפשר לציין תו המשמש להתעלמות מסימנים במספר שלם, למשל הסימן <, > המציין הפרדה בין קבוצת ספרות המציינות אלפים: 1,000,000. כמו כן הפעולה מדלגת על סימנים בהתחלה שאינם סימן מינוס או ספרה. אם נקלטה ספרה ראשונה וחלף זמן קצוב או הופיע תו שאינו מספר, ההמרה מסתיימת. אם לא נקלטה ספרה בזמן קצוב, יוחזר ערך 0.	long parseInt() long parseInt(char skipChar)
הפעולה קוראת מחוצץ הקלט אל מערך תווים char[] או מערך בתים byte[]. הפעולה מסתיימת אם נקלטו len בתים או אם עבר זמן קצוב. כשמשמשים בגרסה Until הקלט יסתיים אם זוהה התו c או שנקלטו len בתים או שעבר זמן קצוב. אם הקלט לא תקין יוחזר הערך 0.	size_t readBytes(buffer, int len) size_t readBytesUntil(char c, buffer, int len)

טבלה 3.1 – תיאור חלקי של פעולות המחלקה Serial

תיאור של חלק הפעולות של מחלקת SoftwareSerial

הפעולה	תיאור הפעולה
void begin(long speed)	הפעולה מאתחלת את התקשורת למהירות המוגדרת בפרמטר speed.
size_t print(val) size_t print(val, format)	הפעולה שולחת את הפרמטר val כמידע טקסטואלי דרך ערוץ התקשורת. כך יוצגו הערכים שיתקבלו: - ערך שלם יוצג כמספר שלם. - ערך ממשי יוצג בדיוק של שתי ספרות אחרי הנקודה העשרונית. - תו או מספר כבית יוצג כתו - מחרוזת תוצג כמחרוזת

הפרמטר format קובע את אופן ההצגה של ערכים מספריים לפי בסיסים שונים או את אופן קביעת מספר הספרות אחרי הנקודה. הפעולה print מציגה את המידע בהמשך השורה והפעולה println מציגה את המידע ועוברת לשורה חדשה. הפעולה מחזירה את מספר הבתים שנכתבו. size_t מוגדר כמספר שלם וחיובי. הערה: כדי לשלוח נתון מספרי באופן בינארי השתמשו בפעולה write.	
הפעולה משדרת את המידע באופן בינארי. הפרמטר val הוא נתון מסוג בית. הפרמטר str הוא נתון מסוג מחרוזת תווים. הפרמטר buf הוא מערך של בתים באורך בפרמטר len. הפעולה מחזירה את מספר הבתים שנכתבו. size_t מוגדר כמספר שלם וחיובי.	<pre>size_t write(byte val) size_t write(string str) size_t write(byte buf[], int len)</pre>
הפעולה קוראת ומחזירה את הערך שנקלט בערוץ התקשורת הטורית. אם אין מידע זמין, הפעולה תחזיר -1.	<pre>int read()</pre>
הפעולה מחזירה את מספר הבתים שנקלטו בערוץ התקשורת הטורית. יש לערוץ התקשורת הטורית חוצץ (buffer) בגודל 64 בתים.	<pre>int available()</pre>
הפעולה קוראת מחוצץ הקלט אל מערך תווים char[] או מערך בתים byte[]. הפעולה מסתיימת אם נקלטו len בתים או אם עבר זמן קצוב. כשמשתמשים בגרסה Until הקלט יסתיים אם זוהה התו c או שנקלטו len בתים או שעבר זמן קצוב. אם הקלט לא תקין יוחזר הערך 0.	<pre>size_t readBytes(buffer, int len) size_t readBytesUntil(char c, buffer, int len)</pre>

טבלה 3.2 – תיאור חלקי של פעולות המחלקה SoftwareSerial

מגבלת השימוש במחלקה

אם משתמשים בכמה התקני תקשורת של מחלקה זו, אפשר לקלוט רק מידע אחד בזמן נתון.

שילוב ספריית SoftwareSerial

כדי להשתמש במחלקה SoftwareSerial חייבים לשלב את הספרייה באמצעות ההוראה:

```
#include <SoftwareSerial.h>
```

הוראה זו תשולב בראש קובץ התכנית.

יצירת עצם של המחלקה

בניגוד להתקן התקשורת הקיים במיקרו-בקר, אין במחלקה SoftwareSerial התקן תקשורת. לכן, חייבים ליצור עצם של המחלקה שיהיה התקן תקשורת טורית. כפי שלמדנו בשפת C# כדי להשתמש בעצם חייבים ליצור אותו באמצעות פעולת בנייה. יש שתי פעולות בנייה, וכאן נלמד אחת מהן.

כותרת פעולת הבנייה היא:

```
SoftwareSerial(int rxPin, int txPin);
```

הפעולה מקבלת את מספרי ההדקים בלוח הפיתוח המוקצים לקליטה ושידור. הפרמטרים של הפעולה הם:

- rxPin – מספר הדק הקליטה.
- txPin – מספר הדק השידור.

3.4 תקשורת טורית בשפת C#

כפי שלמדנו ביחידה מערכות משובצות מחשב, בשפת C# יש מחלקה בשם `SerialPort` הנמצאת במרחב השמות `System.IO.Ports`. כדי להשתמש במחלקה נוסף לתכנית שלנו את מרחבי השמות האלה:

```
using System.IO.Ports;
```

המחלקה `SerialPort`

טבלה 3.3 מציגה חלק ממאפייני המחלקה `SerialPort`:

שם	תיאור
BaudRate	קצב ערוץ התקשורת הטורית
Databits	מספר הסיביות ביחידת מידע
Parity	סיבית ביקורת
PortName	שם מפתח התקשורת
StopBits	מספר סיביות סיום (עצירה)
ReadTimeout	זמן קצוב לקריאה
WriteTimeout	זמן קצוב לכתיבה
IsOpen	ערך המציין אם הערוץ פתוח

טבלה 3.3 – חלק ממאפייני המחלקה `SerialPort`

טבלה 3.4 מציגה ממשק חלקי של פעולות המחלקה `SerialPort`:

שם פעולה	תיאור
<code>SerialPort()</code>	פעולת בנייה מחדל
<code>SerialPort(String, Int32, Parity, Int32, StopBits)</code>	פעולת בנייה המקבלת את שם המפתח הטורי, קצב, סיבית ביקורת, סיביות מידע וסיביות סיום.
<code>void Open()</code>	פעולה הפותחת את ערוץ התקשורת. הפעולה קובעת <code>IsOpen = true</code>
<code>void Close()</code>	פעולה הסוגרת ערוץ התקשורת. הפעולה קובעת <code>IsOpen = false</code>
<code>string ReadLine()</code>	הפעולה קוראת מחרוזת מחוצץ הקלט עד לסימן שורה חדשה <code><\n></code> .
<code>void WriteLine(string text)</code>	הפעולה כותבת את המחרוזת <code>text</code> אל חוצץ הפלט.

טבלה 3.4 – ממשק חלקי של פעולות המחלקה `SerialPort`

דוגמה 3.3 – הגדרת מפתח טורי במחשב האישי בשפת C#

```

1. using System.IO.Ports;
2. class Program
3. {
4.     static SerialPort serial;
5.     static void Main(string[] args)
6.     {
7.         serial = new SerialPort("COM4",19200,Parity.None,8,StopBits.One);
8.         serial.ReadTimeout = 1000;
9.         serial.WriteTimeout = 1000;
10.        serial.Open();
11.        // ToDo
12.    }
13. }

```

להלן הסבר התכנית:

שורה 4: הגדרת עצם מהמחלקה SerialPort

שורה 7: בנייה של עצם עם פרמטרי התקשורת האלה:

PortName = "COM4"	כשם המפתח שלוח הארדואינו מחובר אליו
BauRate = 19200	
Parity = Parity.None	ללא סיבית ביקורת
DataBits = 8	
StopBits = StopBits.One	סיבית סיום אחת

שורה 8: קביעת זמן קצוב של שנייה אחת. אם כעבור שנייה אין קלט הפעולה מסתיימת.

שורה 9: קביעת זמן קצוב של שנייה אחת. אם כעבור שנייה אי אפשר לשים את הפלט הפעולה מסתיימת.

שורה 10: פתיחת ערוץ התקשורת לשידור ולקליטה. אם ערוץ התקשורת תפוס בידי תוכנת המוניטור, אי אפשר לפתוח את הערוץ.

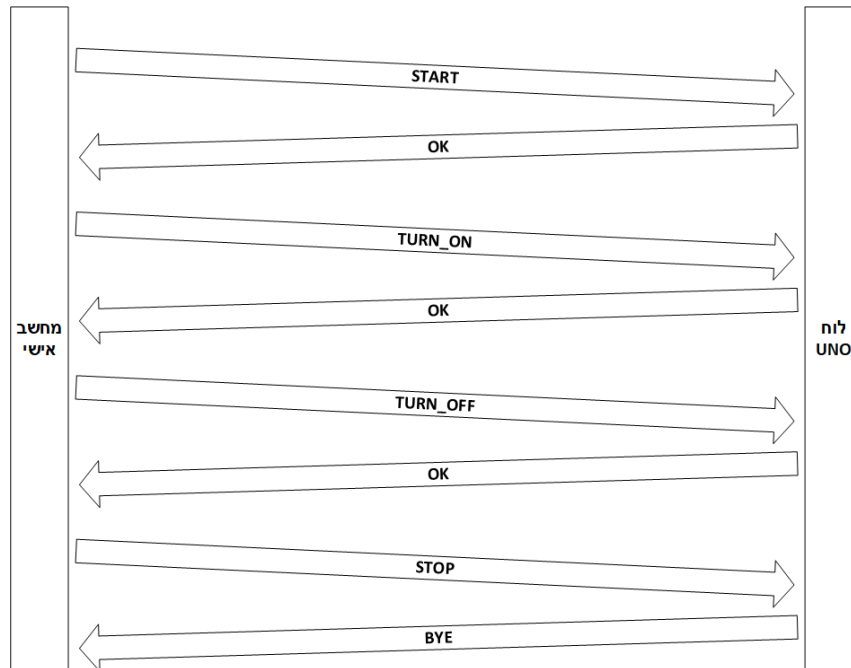
דוגמה 3.4 – הפעלת LED בלוח UNO באמצעות פקודה מתוכנה בשפת C#

בדוגמה זו נכתוב שתי תוכניות: הראשונה למחשב האישי בשפת C# והשנייה ללוח ה-UNO. תקשורת בין המחשב ללוח תתבצע בתקשורת טורית בקצב 19,200. מטרת התכנית במחשב האישי היא להציע למשתמש את אפשרויות הפעולה אלה:

1. התחברות ללוח UNO.
2. הדלקת LED.
3. כיבוי LED.
4. התנתקות מלוח ה-UNO וסגירת התכנית.

מטרת התכנית בלוח ה-UNO היא להמתין לקבלת הוראות מהמחשב האישי ולבצע אותן. לאחר ביצוע ההוראה תישלח הודעה חוזרת ללוח המאשרת לבצע אותה.

איור 3.7 מציג את פרוטוקול ההעברה בין המחשב ללוח:



איור 3.7 – פרוטוקול ההעברה בין המחשב ללוח

צד לוח UNO

לוח ה-UNO נמצא במצב המתנה לקבלת ההודעה START. עם קבלת ההודעה עוברת התוכנה למצב שהיא מצפה לקבל אחת משלוש הודעות אפשריות: TURNON, TURNOFF, או STOP. לאחר המעבר נשלחת הודעת אישור אל המחשב האישי. עם קבלת ההודעה, מתבצע פענוח שלה:

הודעת TURNON – התוכנה מדליקה את ה-LED ושולחת הודעת אישור ביצוע (OK).

הודעת TURNOFF – התוכנה מכבה את ה-LED ושולחת הודעת אישור ביצוע (OK).

הודעת STOP – התוכנה חוזרת למצב המתנה לקבלת הודעת START ושולחת אישור התנתקות (BYE).

```
// הגדרת קבועים

#define ARRAYSIZE 25
#define START "START"
#define STOP "STOP"
#define TURNON "TURNON"
#define TURNOFF "TURNOFF"
#define OK "OK\n"
#define BYE "BYE\n"

// הגדרת משתני התכנית

char val[ARRAYSIZE]; // מערך תווים לקליטת מחרוזת הודעה
int s; // אורך המחרוזת שנקלטה
bool isWaiting; // משתנה להגדרת מצב המתנה לתקשורת
int ledPin = 13; // הדק הLED

void setup () {
    Serial.begin(19200); // אתחול תקשורת טורית
    isWaiting = true; // אתחול למצב המתנה
    pinMode(ledPin, OUTPUT); // הגדרת הדק ספרתי כפלט
    digitalWrite(ledPin, LOW); // כיבוי הLED
}

void loop () {
    while (Serial.available() > 0) { // המתנה לקבלת מידע
        s = Serial.readBytesUntil('#', val, ARRAYSIZE); // קליטה עד קבלת תו #
        val[s] = 0; // הוספת ערך 0 – סוף מחרוזת
        if (isWaiting && strcmp(val, START) == 0) { // המתנה לקבלת הודעת התחברות
            isWaiting = false; // מעבר למצב הוראות
            Serial.print(OK); // אישור מעבר לקבלת הוראות
        }
    }
}
```

המשך הקוד בעמוד הבא <<

```
else if (!isWaiting) {
    if (strcmp(val, STOP)== 0){ // קבלת הודעת ניתוק
        isWaiting = true;
        Serial.print(BYE);           // שליחת אישור ניתוק
    }
    else if (strcmp(val, TURNON)== 0){ // קבלת הודעת הדלקה
        digitalWrite(ledPin, HIGH);
        Serial.print(OK);
    }
    else if (strcmp(val, TURNOFF)== 0) { // קבלת הודעת כיבוי
        digitalWrite(ledPin, LOW);
        Serial.print(OK);
    }
}
}
```

צד מחשב אישי

```

using System;
using System.IO.Ports;           // מרחב שמות של מפתח תקשורת
namespace Serial2
{
    class Program
    {
        static SerialPort serial; // עצם מסוג תקשורת טורית
        static bool keepContinue;  // משתנה בוליאני לביצוע הלולאה הראשית
        static int choice;         // משתנה הבחירה של אפשרויות התפריט
        static bool isConnected;  // משתנה בוליאני לציון מצב התחברות ראשוני
        static void Main(string[] args)
        {
            isConnected = false;   // קביעת מצב לא מחובר
                                   // פעולת בנייה לעצם של תקשורת טורית עם אתחול הפרמטרים
            serial = new SerialPort("COM4", 19200, Parity.None, 8, StopBits.One);
            serial.ReadTimeout = 1000; // קוצב זמן לביצוע קריאה
            serial.Open();           // פתיחת ערוץ התקשורת
            keepContinue = true;    // אתחול משתנה הלולאה
                                   // הצגת תפריט למשתמש

            Console.WriteLine("\nLED CONTROL UNO");
            Console.WriteLine("1. CONNECT");
            Console.WriteLine("2. TURN ON");
            Console.WriteLine("3. TURN OFF");
            Console.WriteLine("4. DISCONNECT");
            Console.Write("Enter your choice <1..4> ");
            while (keepContinue)
            {
                choice = Console.ReadKey().KeyChar; // קליטת תו
                switch (choice)                     // משפט ברירה רב-מצבי
                {

```

המשך הקוד בעמוד הבא <<

```

case '1':                                // התחברות ראשונית
    if (!isConnected)                    // אם עדיין לא מחובר
    {
        serial.WriteLine("START#"); // שליחת הודעת התחברות
    }
    break;
case '2':                                // דרישה להדלקת הלד
    serial.WriteLine("TURNON#");        // שליחת הודעת הדלקה
    break;
case '3':                                // דרישה לכיבוי הלד
    serial.WriteLine("TURNOFF#");        // שליחת הודעת כיבוי
    break;
case '4':                                // דרישה להתנתקות
    serial.WriteLine("STOP#");            // שליחת הודעת התנתקות
    break;
}
try                                     // ניסיון לבצע את הקריאה בקטע הבא
{
    string message = serial.ReadLine(); // קלט מחרוזת
    if (message == "BYE")                // אישור התנתקות
    {
        Console.WriteLine("Good Bye\nPress any key...");
        isConnected = false;            // עדכון משתנים בוליאניים לסיום
        keepContinue = false;
    }
    else if (message == "OK")            // קבלת אישור להודעות
    {
        switch (choice)
        {
            case '1':                    // אישור התחברות
                Console.WriteLine("\nYou are CONNECTED");
                isConnected = true;
                break;

```

המשך הקוד בעמוד הבא <<

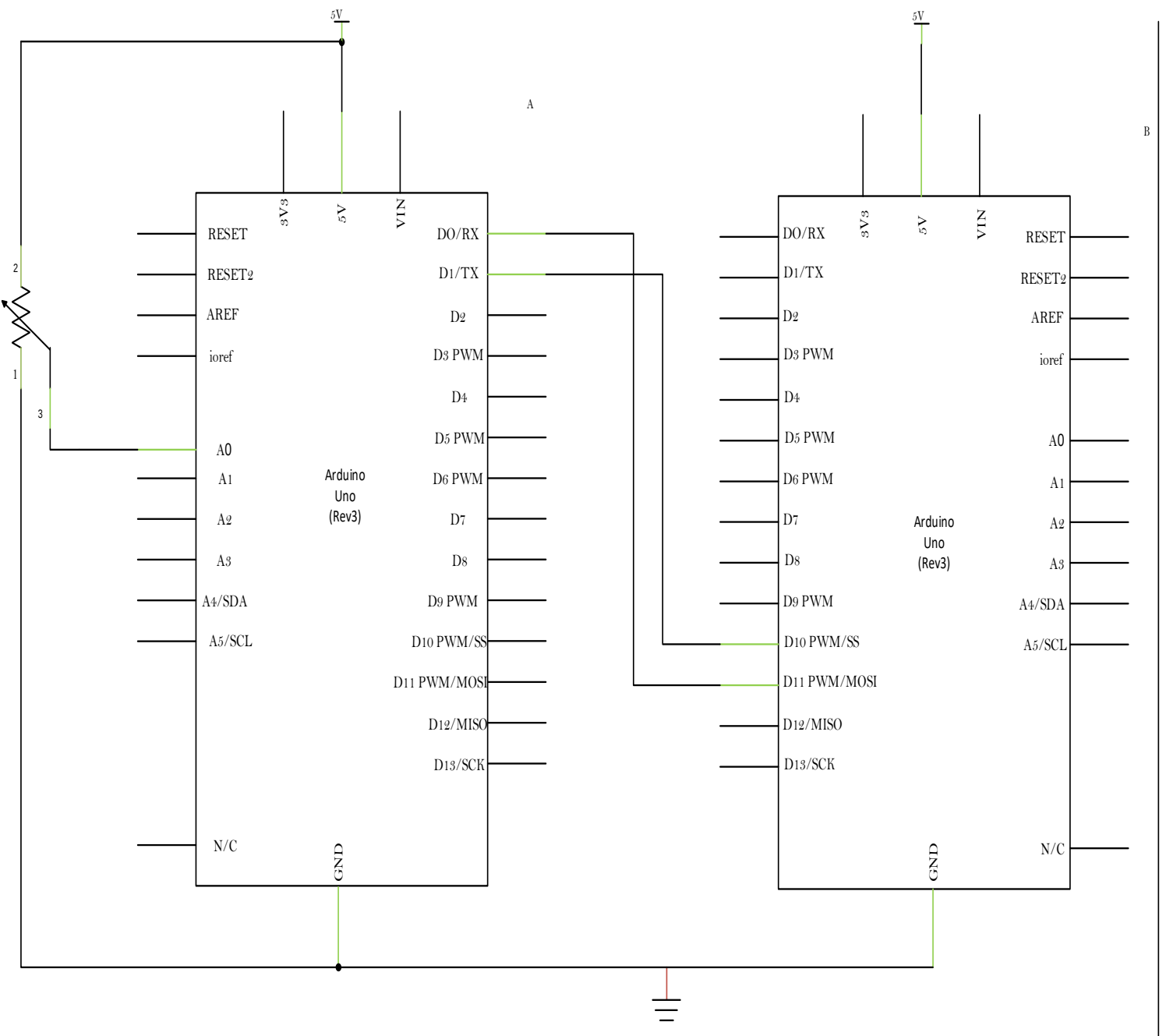

```
        case '2':                // אישור הפעלה
            Console.WriteLine("\nLED IS ON");
            break;
        case '3':                // אישור כיבוי
            Console.WriteLine("\nLED IS OFF");
            break;
    }                            // הצגה חוזרת של התפריט
    Console.WriteLine("\nLED CONTROL UNO");
    Console.WriteLine("1. CONNECT");
    Console.WriteLine("2. TURN ON");
    Console.WriteLine("3. TURN OFF");
    Console.WriteLine("4. DISCONNECT");
    Console.WriteLine("Enter your choice <1..4> ");
    }
}
catch (TimeoutException) { }    // אם ניסיון הקלט לא הצליח
}
serial.Close();                // סגירת מפתח התקשורת בסיום התכנית
}
}
```

דוגמה 3.5 – קריאת ערך אנלוגי ושידורו בתקשורת טורית ללוח אחר

כתבו תכנית בלוח A הדוגמת ערך אנלוגי מהדק A0 ומשדרת אותו לערוץ התקשורת הטורית.

כתבו תכנית בלוח B הקולט את המידע בערוץ תקשורת הממומש באמצעות SoftwareSerial המוגדר בהדקים 10 ו-11 בלוח הפיתוח. המידע הנקלט ישודר ל-Serial Monitor.

באיור 3.8 מוצג תרשים החיבורים של לוחות הפיתוח:



איור 3.8 – תרשים החיבורים של לוחות הפיתוח

תוכנה לוח A

```

void setup() {
  Serial.begin(9600);          // אתחול וקביעת קצב השידור
}
void loop() {
  Serial.println(analogRead(A0)); // קריאת הערך האנלוגי ושידורו בערוץ התקשורת
  delay(500);                  // השהייה
}

```

תוכנה לוח B

```

#include <SoftwareSerial.h>      // שילוב הספרייה
SoftwareSerial mySerial(10,11); // יצירת עצם של התקן תקשורת
int val;                        // משתנה לקליטת הערך האנלוגי
void setup() {
  mySerial.begin(9600);         // אתחול וקביעת קצב התקשורת לערוץ
  Serial.begin(9600);          // אתחול וקביעת קצב התקשורת לערוץ
}
void loop() {
  value = mySerial.parseInt();  // קריאת ערך שלם מערוץ התקשורת בתוכנה
  Serial.println(value);        // הצגת הערך במוניטור
  delay(10);
}

```

להלן הסבר התכנית:

התוכנה בלוח A דוגמת ערך ומשדרת אותו דרך ערוץ התקשורת המובנה. אפשר לצפות בערכים אלה בלוח A באמצעות תוכנת המוניטור.

התוכנה בלוח B מאתחלת שני ערוצי תקשורת: האחד מובנה והאחר ממומש בתוכנה. המידע מלוח A נקלט מהערוץ הממומש בתוכנה ונשלח לתצוגה דרך ערוץ התקשורת המובנה.

תרגול

שאלה 3.3

כתבו תכנית ללוח UNO המפעילה LED, המחובר ליציאה אנלוגית 11, באמצעות ערך מספרי שיתקבל דרך ערוץ התקשורת הטורית באמצעות ה-Serial Monitor. הערך המספרי שיישלח יהיה בין 0 ל-255.

שאלה 3.4

כתבו תכנית ללוח UNO המפעילה LED, המחובר ליציאה אנלוגית 11, באמצעות ערך מספרי שיתקבל דרך ערוץ התקשורת הטורית באמצעות תוכנה בשפת C# שנכתבה למחשב האישי. הערך המספרי שיישלח יהיה בין 0 ל-255.

בצד המחשב האישי ייקלט ערך מספרי מהמשתמש, ולאחר מכן יישלח דרך ערוץ התקשורת הטורית אל לוח UNO.

בצד הלוח תהיה המתנה לקבלת ערך מספרי. לאחר קבלת הערך תעודכן היציאה האנלוגית.

שאלה 3.5

כתבו תכנית ללוח UNO המפעילה LED, המחובר ליציאה אנלוגית 11, באמצעות ערך מספרי שיתקבל דרך ערוץ התקשורת הטורית המחובר ללוח פיתוח אחר. הערך המספרי שיישלח יהיה בין 0 ל-255.

בצד האחר ייקלט ערך מספרי מהמשתמש, ולאחר מכן יישלח דרך ערוץ התקשורת הטורית אל לוח ה-UNO הראשון.

סיכום פרק 3

בפרק 3 למדתם את הנושאים האלה:

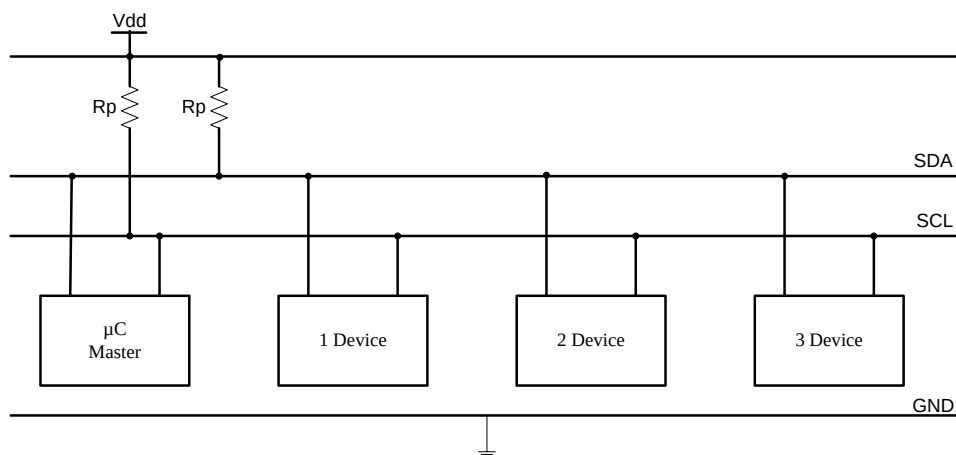
1. תקן תקשורת טורית RS232
2. קווי התקשורת TX ו-RX
3. מאפיינים חשמליים של קו תקשורת
4. שיטת שידור א-סינכרונית
5. מבנה מסגרת שידור: סביבת התחלה, מידע, סיבית ביקורת וסיביות סיום
6. זמן סיבית וקצב שידור
7. זמן מסגרת וקצב שידור מסגרות
8. פעולות המחלקה Serial
9. התקן UART בתוכנה SoftwareSerial
10. תקשורת טורית במחשב האישי
11. תקשורת טורית בשפת C# והמחלקה SerialPort
12. מאפיינים ופעולות המחלקה SerialPort
13. יצירת פרוטוקול תקשורת בין מערכות
14. יישום פרוטוקול התקשורת בין המחשב ללוח UNO
15. כתיבת תכנית לדוגמה להפעלה ולכיבוי של לד מרחוק בתקשורת טורית.
16. תקשורת בין שני לוחות UNO

פרק 4: פרוטוקול I²C

פרוטוקול I²C – Inter-Integrated Circuit (איי סקוורד סי) – הומצא בראשית שנות השמונים בידי חברת Philips (כיום NXP) כדי לאפשר תקשורת סינכרונית פשוטה בין רכיבים הנמצאים על אותו מעגל. טכנולוגיה זו ידועה גם בשם Two Wire Interface המאפשרת חיבור של כמה רכיבים תוך שימוש בשני קווים בלבד (קו שלישי הוא קו ייחוס אדמה, Ground). החל מאמצע שנות התשעים החלו חברות נוספות לשווק מוצרים התואמים לחלוטין לתקן זה.

4.1 ארכיטקטורה

איור 4.1 להלן מתאר את אופן החיבור בין ההתקנים שונים. המערכת מאפשרת תקשורת סינכרונית² בין התקנים שונים במעגל או בין התקנים במעגלים שונים תוך שימוש בשני קווי תקשורת בלבד: SCL – Serial Clock, SDA – Serial Data. התקשורת היא דו-כיוונית למחצה. כלומר, התקשורת יכולה להתבצע בשני הכיוונים, אך לא בו זמנית.



איור 4.1 – ארכיטקטורת I²C

קווי התקשורת הם דו-כיווניים ומחוברים במקביל לכל ההתקנים. הדקי התקן המתחבר לקווים הם בטכנולוגיית Open Drain³. לכל קו מחובר נגד R_p, Pull-Up, המחובר למתח 5V. התקן יכול להשפיע על הקו רק כאשר הוא גורם לו להיות במצב נמוך (Low). מספיק שאחד מההתקנים מוריד את הקו לרמה נמוכה כדי שהוא יישאר ברמה זו. התקנים נוספים יכולים

² תקשורת סינכרונית היא תקשורת הנוצרת בין היחידות לפי הצורך. הצד היוזם שולח איתות התחלה, והצד האחר מאזין לקווי התקשורת ומגיב לפי פרוטוקול ידוע. בסוף התהליך התקשורת מסתיימת באיתות סיום.

³ Open Drain מתייחס למצב בו הדק השפך (Drain) בטרנזיסטור מסוג FET נשאר פתוח ללא חיבור. כדי שהטרנזיסטור יפעל יש לחבר נגד חיצוני אל מקור מתח.

להוריד במקביל את הקו לרמה נמוכה וכך הם יכולים להשפיע על פעילותם של ההתקנים האחרים.

Master-Slave

ההתקנים במערכת יכולים להיות מסוג התקן שליט (Master) או התקן עבד (Slave). המערכת כוללת שליט אחד וכמה התקני עבד. לפי פרוטוקול מתאים, אפשר שהתקנים יחליפו תפקידים במהלך העבודה. המערכת מסוגלת לתמוך גם במצב שיש כמה התקנים המוגדרים כשליטים (Multi-Master-Bus).

התקשורת בין ההתקנים מתחילה ביוזמת ההתקן השליט. השליט פונה אל ההתקנים באמצעות כתובת המוגדרת לכל התקן עבד. הכתובת מועברת דרך הקו SDA המתוזמן באמצעות קו השעון SCL.

קו השעון SCL נוצר בידי ההתקן השליט בלבד ומספק להתקני העבד בקרת זרימה. עם זאת, התקני עבד איטיים יכולים להשפיע, במידת הצורך, על משך הזמן שהקו ברמה נמצא לוגית נמוכה.

קו הנתונים SDA מופעל בידי השליט או בידי העבד לפי כיוון זרימת המידע, אך השליט בלבד אחראי לתזמון.

4.2 אותות בערוץ

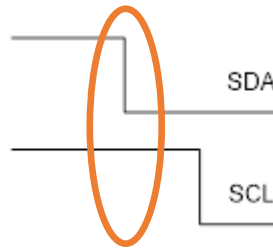
הקווים SDA ו-SCL משמשים להעברת אותות מידע בין ההתקנים השונים. נבחין בין שני סוגי איתות: בקרה ומידע.

איתותי בקרה

כאשר המערכת מופעלת לראשונה הקווים נמצאים במצב גבוה (HIGH), משום שכל ההתקנים במצב לא פעיל. ההתקן השליט מתחיל תהליך פנייה אל התקני העבד באמצעות האיתות Start ומסיים את התהליך באמצעות האיתות Stop. האיתותים Start ו-Stop משנים את מצב קו הנתונים SDA כאשר קו השעון SCL נמצא במצב גבוה, ואילו בשאר המקרים קו הנתונים SDA משתנה רק כאשר קו השעון SCL נמצא במצב נמוך. האיתותים שומרים על ערכו של SDA קבוע כאשר קו השעון SCL נמצא במצב גבוה.

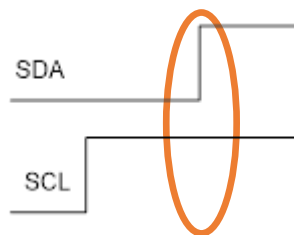
להלן איורים 4.2 ו-4.3 המציגים את איתות Start ואת איתות Stop:

1. איתות Start: קו SDA בירידה כאשר קו SCL ברמה גבוהה.



איור 4.2 – איתות Start

2. איתות Stop: קו SDA בעלייה כאשר קו SCL ברמה גבוהה.

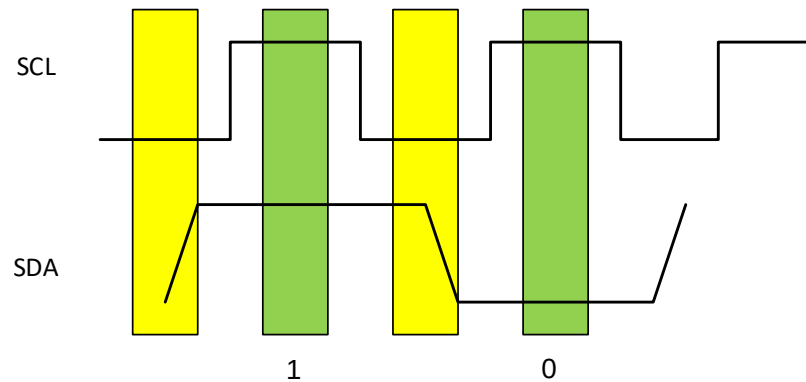


איור 4.3 – איתות Stop

העברת סיבית

איור 4.4 מראה את אופן השידור של שתי סיביות עוקבות 1 ו-0. כאשר קו השעון SCL נמצא ברמה נמוכה קובעים את ערך הסיבית לשידור בקו הנתונים SDA (מסומן במלבן הצהוב). שומרים על ערך הסיבית בקו הנתונים ומעלים את אות השעון לרמה גבוהה. כאשר אות השעון

ברמה גבוהה שומרים על קו הנתונים קבוע (מסומן במלבן הירוק). עם ירידת אות השעון מושלם מחזור העברת הסיבית.



איור 4.4 – העברת סיביות

4.3 מסגרות שידור

לאחר שידור איתות Start מתחיל תהליך של העברת מידע מהשליט אל העבד, וחזרה. העברת המידע מסתיימת כאשר השליט מאותת איתות Stop.

המידע המועבר הוא במנה של 8 סיביות וסיבית נוספת ACK (Acknowledge), המשמשת לאישור. כלומר, בכל מסגרת שידור מועברות 9 סיביות. הסיבית הראשונה המועברת במסגרת השידור היא הסיבית המשמעותית ביותר MSB.

לאחר 8 סיביות מידע המשודרות בכיוון אחד משודרת סיבית אישור אחת בכיוון ההפוך. המשדר והמקלט מחליפים תפקידים למשך סיבית אחת. היחידה שקלטה משדרת סיבית אישור (ACK) ברמה נמוכה (L) בקו ה-SDA. אם היחידה ששידרה מזהה רמה גבוהה (H) היא מניחה שאחת מהאפשרויות התרחשו:

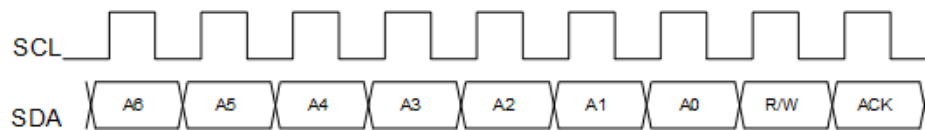
1. כאשר השליט משדר את המידע לעבד, הוא מניח שהעבד אינו מסוגל לקבל את המידע מאחת הסיבות האלה: (1) אין התקן עבד מתאים; (2) ההוראה לא הובנה; (3) לעבד אין יכולת לקבל כרגע מידע.
2. כאשר העבד משדר את המידע לשליט, הוא מבין שהשליט מבקש לסיים את העברת המידע.

נבחין בין שתי מסגרות שידור:

- מסגרת מנת כתובת (Address Packet Format)
- מסגרת מנת נתונים (Data Packet Format)

מסגרת מנת כתובת

מנת כתובת היא באורך 9 סיביות: 7 סיביות משמשות לכתובת, סיבית READ/WRITE, סיבית ACK לבקרה וסיבית לאישור. אם הסיבית READ/WRITE ברמה לוגית גבוהה (H), תתבצע פעולת קריאה. אם לא, תתבצע פעולת כתיבה. כאשר עבד מזהה את כתובתו, הוא צריך לאשר (ACK) על ידי הורדת קו SDA לרמה נמוכה באות השעון התשיעי. אם העבד עסוק או אינו מסוגל להיענות לבקשה מכל סיבה אחרת, קו ה-SDA צריך להישאר ברמה לוגית גבוהה. השליט יכול לשלוח איתות Stop כדי לסיים את התהליך או לנסות לשלוח שוב איתות התחלה Start כדי להתחיל מחזור העברה חדש.



איור 4.5 – מסגרת כתובת

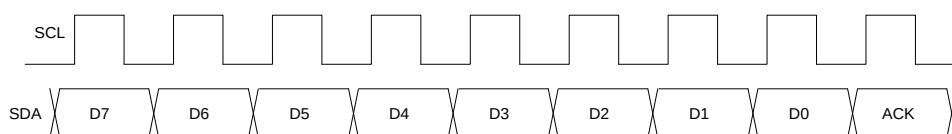
מתכנן המערכת יכול לקבוע את כתובות התקני העבד, אבל כתובת 0000000 שמורה לקריאה כללית.

כאשר מתבצעת פנייה לכתובת כללית, כל התקני העבד מגיבים בקביעת קו ה-SDA לרמה נמוכה במחזור האישור (ACK). השליט ישדר כתובת כללית כאשר הוא רוצה להעביר הודעה זהה לכל התקני העבד. כאשר השליט יוזם פנייה לכתובת כללית לשם כתיבה, כל התקני העבד יאשרו במחזור האישור (ACK). לאחר מסגרת הכתובת כל מנות הנתונים ייקלטו על ידי התקני העבד. שידור של כתובת כללית לצורך קריאה היא חסרת משמעות – פעולה כזו תגרום להתנגשות כי מספר התקני עבד ישדרו נתונים שונים.

מבנה מסגרת מנת נתונים

מנת הנתונים היא באורך 9 סיביות, ובהן נתון בגודל (data byte) וסיבית אישור (ACK). במהלך העברת הנתונים, ההתקן השליט מייצר את אות השעון, את איתות ההתחלה Start ואת איתות הסיום Stop. ההתקן הקולט אחראי לאישור הקליטה. אישור ACK מתבצע על ידי הורדת קו ה-SDA לרמה נמוכה במחזור התשיעי של אות השעון. כאשר היחידה הקולטת מקבלת את הבית האחרון – או אם היחידה הקולטת אינה מסוגלת לקלוט מסיבה כלשהי בתיים נוספים – היא תיידע את יחידת השידור על ידי שליחת NACK, קו SDA ברמה נמוכה במהלך המחזור התשיעי.

סיבית ה-MSB של בית הנתונים משודרת תחילה.

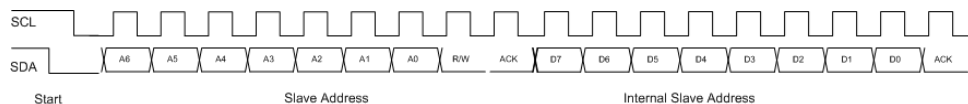


איור 4.6 – מסגרת נתונים

4.4 פרוטוקול תוכנה

בתחילה ההתקן השליט שולח רצף Start, המציין את תחילת ההעברה. בעקבות כך התקני העבד נכנסים למצב הקשבה כדי לבדוק אם התקשורת מיועדת להם. מיד לאחר מכן שולח השליט את מנת הכתובת להתקן עבד. העבד שמזהה את כתובתו ממשיך בתהליך התקשורת, ואילו האחרים מתעלמים מהתהליך וממתינים לרצף התחלה חדש.

בעקבות האישור (ACK) המתקבל מהעבד, שולח השליט כתובת פנימית של העבד שאליה השליט רוצה לפנות למטרת כתיבה או קריאה. ערך הכתובת הפנימית נקבע על פי מרחב הכתובות של העבד בלבד. להתקנים פשוטים אין כתובת פנימית, אולם למרבית ההתקנים יש כמה כתובות פנימיות. חשוב לציין כי הפנייה הראשונה אל העבד היא לשם כתיבת הכתובת הפנימית, ולכן הסיבית READ/WRITE (R/W) תהיה ברמה 0 כדי לציין תהליך כתיבה לעבד.

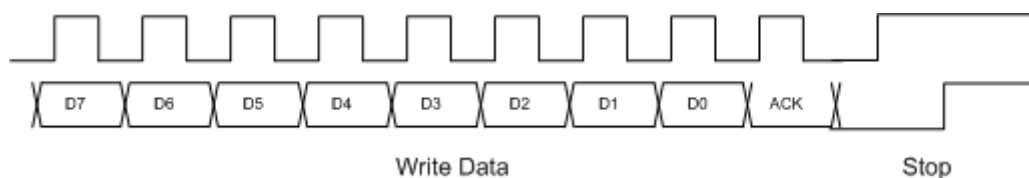


איור 4.7 – שידור כתובת התקן

תהליך הכתיבה

לאחר שליחת כתובת ההתקן ומספר הכתובת הפנימית השליט שולח בזה אחר זה נתונים אל העבד. הנתונים נשמרים בעבד בכתובות פנימיות עוקבות לפי סדר קבלתם. להלן רצף הפעולות:

1. שדר איתות Start.
2. שדר כתובת התקן עבד עם סיבית R/W ברמה 0.
3. שדר כתובת פנימית בהתקן העבד.
4. שדר את הנתון להתקן העבד.
5. חזר על שידור הנתונים להתקן עבד – אופציונאלי.
6. שדר רצף Stop.

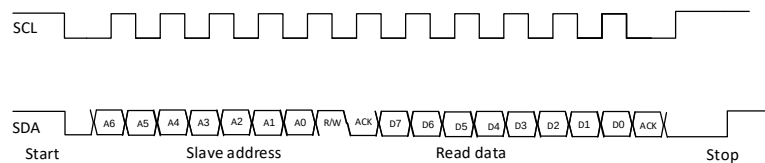


איור 4.8 – שידור נתון לעבד

תהליך הקריאה

לאחר שליחת כתובת ההתקן ומספר הכתובת הפנימית, השליט שולח רצף התחלה חדש עם כתובתו של התקן העבד. אולם בתהליך הקריאה נשלחת סיבית R/W ברמה 1. לאחר האישור השליט קורא מידע מהתקן העבד לפי הצורך. אחרי כל קריאת נתון השליט מאשר את הקריאה באמצעות ACK. בקריאת הנתון האחרון שולח השליט רצף Stop. להלן רצף הפעולות:

1. שדר רצף Start.
2. שדר כתובת התקן עבד עם סיבית R/W ברמה 0.
3. שדר כתובת פנימית בהתקן העבד.
4. שדר רצף Start.
5. שדר כתובת התקן עבד עם סיבית R/W ברמה 1.
6. קלוט נתון מהתקן העבד.
7. חזור על קליטת נתונים מהתקן העבד – אופציונאלי.
8. שדר רצף Stop.



איור 4.9 – יחידת התקשורת במיקרו-בקר

4.5 מתיחת שיעון – Clock stretching

כאמור, אות השיעון מיוצר על ידי ההתקן השליט ומועבר אל התקני העבד כדי לקבוע את קצב העברת המידע. כאשר ההתקן השליט מעביר מידע הוא קובע את קצב ההעברה. אולם כאשר ההתקן השליט קורא מידע, התקן העבד מבצע את ההעברה וקובע את הקצב. לעיתים התקן העבד אינו מסוגל לעמוד בקצב שהשליט מכתוב, ולכן כדי לאפשר תקשורת בקצב המתאים להתקן העבד, הוא יכול להשאיר את קו השיעון SCL ברמה לוגית נמוכה למשך הזמן הדרוש לו.

במהלך העברת המידע התקן העבד מוריד את הקו מטה כאשר ההתקן השליט מוריד את הקו (פעולה זאת אפשרית הודות לשימוש בנגדי pull up). כאשר ההתקן השליט מעלה את קו השיעון לרמה גבוהה, וקו זה עדיין ברמה נמוכה בצד של התקן העבד, השליט ממתיין עד שגם התקן העבד יעלה את הקו לרמה גבוהה.

4.6 קצב תקשורת

קצב התקשורת התקני הוא 100kbit/s. אפשר לשדר גם בערכי קצב אחרים, והם מפורטים בטבלה:

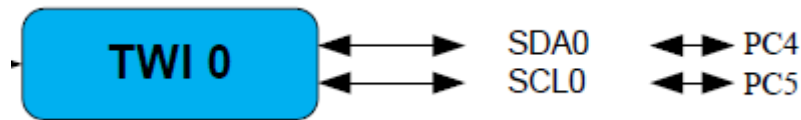
קצב תקשורת	אופני תקשורת
10kbit/s	Low Speed Mode – נמוכה
100kbit/s	Standard speed Mode – תקנית
400kbit/s	Fast mode – מהירה
1Mbit/s	Fast mode plus – מהירה פלוס
3.4Mbit/s	High speed mode – גבוהה

קצב העברת הנתונים בין השליט לעבד נמוך מקצב התקשורת מכיוון שקיימת תקורה של שליחת כתובת, והעברת אישורים (ACK/NACK).

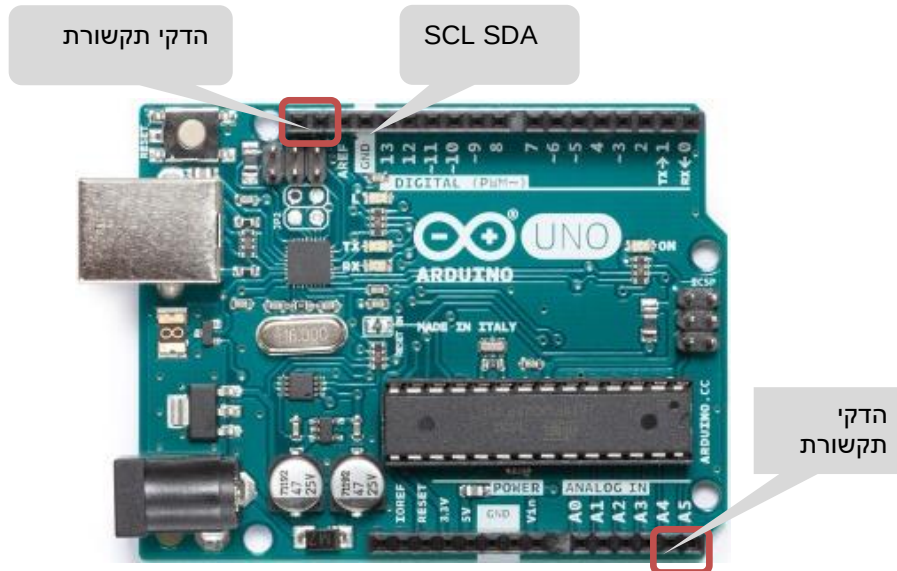
מספר ההתקנים שניתן לחבר לשליט אחד מוגבל על ידי מרחב הכתובות (7 קווי כתובת–128 קווי כתובות) והקיבול הכולל של הערוץ. אסור שהקיבול הכולל יהיה גבוה מ-400pF, וזאת כדי שלא לקצר את המרחק בין היחידות השונות.

4.7 קווי התקשורת בלוח Arduino UNO

המיקרו-בקרי ATmega328P כולל תמיכה מובנית בתקשורת I²C:



כזכור, הדקי מפתח PC[5:0] משמשים גם כהדקי כניסה אנלוגיים A5–A0 בהתאמה.



איור 4.10 – הדקי התקשורת בלוח הפיתוח

כפי שרואים באיור 4.10 אפשר לחבר התקנים לשקעי החיבור המיועדים לכך בלוח מצד שמאלה למעלה או בשקעים המסומנים ב-A4 (תפקידו SDA) וב-A5 (תפקידו SCL). המיקרו-בקרי תומך בתקשורת בקצב מרבי של עד 400kbit/s.

הערה: לוח הפיתוח לא כולל נגדי Pull-Up הדרושים לקיום התקשורת ולכן בכל תכנון חייבים להוסיף אותם.

4.8 מחלקת Wire

המחלקה Wire היא ספרייה הכוללת אוסף פעולות המאפשר שימוש בתקשורת I²C. הספרייה תומכת בכתובות של 7 סיביות, כלומר בתחום 0 עד 127. כתובות 0 עד 7 שמורות ולכן הכתובת הראשונה הפנויה לשימוש היא 8.

בספרייה ישנו חוצץ נתונים בגודל 32 בתים, ולכן חשוב לשים לב שלא תתבצע חריגה מגבול זה. אם בשידור יחיד תתבצע חריגה מגודל 32 בתים כל הבתים העודפים יושלכו.

בטבלה 4.1 מתוארות את פעולות המחלקה (UML).

תיאור הפעולה	הפעולה
הפעולה מאתחלת את התקשורת לקווי I²C. address: כתובת עבד של 7 סיביות. אם לא מוגדרת כתובת, ההתקן יהיה שליט.	<pre>void begin() void begin(byte address) void begin(int address)</pre>
הפעולה משמשת את ההתקן השליט לבקש בתיים מהתקן העבד. אפשר לקרוא את המידע באמצעות הפעולות read-ו available. address: כתובת עבד של 7 סיביות. quantity: מספר הבתים המבוקשים. stop: ערך true – השליט ישלח איתות Stop לאחר הבקשה, הגורם לשחרור הקווים. זוהי ברירת המחדל. ערך false – השליט ישלח שוב איתות Stop לאחר הבקשה. במצב זה הקווים עדיין תפוסים. הפעולה מחזירה את מספר הבתים שהתקבלו מהתקן העבד.	<pre>byte requestForm(byte address, byte quantity) byte requestForm(byte address, byte quantity, bool stop)</pre>
הפעולה מתחילה שידור להתקן עבד על פי הכתובת הנתונה. address: כתובת עבד של 7 סיביות. הפעולה write תאגור בתור את הבתים המשודרים לעבד. הפעולה endTransmission תשדר את הבתים שנאגרו בתור.	<pre>void beginTransmission (byte address) void beginTransmission (int address)</pre>
הפעולה מסיימת את תהליך השידור לעבד שאותחל באמצעות ההוראה beginTransmission ומשדרת את הבתים שנאגרו בתור בפעולת write. stop: ערך true – השליט ישלח איתות Stop לאחר השידור, הגורם לשחרור הקווים. זוהי ברירת המחדל. ערך false – השליט ישלח שוב איתות Stop לאחר השידור. במצב זה הקווים עדיין תפוסים, דבר המונע מהתקן שליט אחר לשדר בין ההודעות. הפעולה מחזירה בית המציין את מצב השידור: 0 – הצלחה. 1 – הנתונים ארוכים עבור חוצץ השידור (32). 2 – התקבל NACK לאחר שידור הכתובת. 3 – התקבל NACK לאחר שידור הנתונים. 4 – שגיאה אחרת.	<pre>byte endTransmission() byte endTransmission(bool stop)</pre>
הפעולה משדרת נתון מהתקן עבד בתגובה לבקשה של התקן שליט או אוגרת בתור לשידור בתיים המתקבלים מהשליט להעברה לעבד. val: נתון מסוג בית. str: נתון מסוג מחרוזת תווים. buf: מערך של בתיים באורך הפרמטר len. הפעולה מחזירה את מספר הבתים שנכתבו. size_t: מוגדר כמספר שלם וחיובי.	<pre>size_t write(byte val) size_t write(string str) size_t write(byte buf[], int len)</pre>

int available()	הפעולה מחזירה את מספר הבתים הזמינים לקריאה באמצעות ההוראה read. הקריאה לפעולה תתבצע בהתקן שליט לאחר הקריאה להוראה requestForm או בהתקן עבד לאחר פעולת מנהל onReceive.
int read()	הפעולה קוראת ומחזירה את הערך ששידר התקן עבד לאחר קריאה לפעולה requestForm או את הערך ששידר תקן שליט לעבד.
void setClock(long freq)	הפעולה משנה את תדירות אות השעון לתקשורת. קצב התקשורת הבסיסי הוא 100KHz. freq : תדירות רצויה ביחידות של הרץ – Hz.
void onReceive (myHandler)	הפעולה מקבלת, כפרמטר, שם של פעולה ורושמת אותה כפעולה שתופעל כאשר תקן עבד קולט מידע מתקן שליט. כותרת הפעולה המועברת כפרמטר: void myHandler(int numBytes)
void onRequest (myHandler)	הפעולה מקבלת, כפרמטר, שם של פעולה ורושמת אותה כפעולה שתופעל כאשר תקן שליט מבקש נתונים מתקן עבד. כותרת הפעולה המועברת כפרמטר: void myHandler()

הגדרת לוח הפיתוח כשליט

הפעלת לוח הפיתוח כהתקן שליט תתבצע תוך שימוש בפעולה begin() ללא פרמטר. פעולה זו תאתחל את קווי התקשורת לשימוש ב-I²C.

להלן ההוראה להגדרת לוח הפיתוח כשליט: Wire.begin();

הגדרת לוח הפיתוח כעבד

הפעלת לוח הפיתוח כהתקן עבד תתבצע תוך שימוש בפעולה begin(address) עם פרמטר כתובת. פעולה זו תאתחל את קווי התקשורת לשימוש ב-I²C ותקצה עבורו כתובת. כאמור, ניתן להקצות כתובות החל מ-8.

להלן ההוראה להגדרת לוח הפיתוח כעבד: Wire.begin(8);

כתיבה על ידי התקן שליט

כל תקשורת מתחילה בהתקן השליט באמצעות איתות Start ומיד לאחר מכן משודרת מנת כתובת עם סיבית READ/WRITE המתאימה לכתיבה (ST+W). לאחר העברת הכתובת אפשר להעביר בתי נתונים בזה אחר זה המסתיימים באיתות Stop.

להלן ההוראות לכתיבה על ידי התקן שליט:

Wire.beginTransaction(address);	התחלת שידור לעבד בעל הכתובת (address).
Wire.write("string");	כתיבת מחרוזת לתור השידור.
Wire.write(val);	כתיבת ערך מספרי לתור השידור.
Wire.endTransmission();	שידור הבתים הנמצאים בתור וסיום השידור.

הפעלת ההוראה האחרונה כפי שמופיע להלן מאפשרת לבדוק את תוכן המשתנה result ולאבחן תקלות תקשורת:

```
result = Wire.endTransmission();
```

קריאה על ידי התקן שליט

כל תקשורת מתחילה בהתקן השליט באמצעות איתות Start ומיד לאחר מכן משודרת מנת כתובת עם סיבית READ/WRITE המתאימה לקריאה (ST+R). לאחר העברת הכתובת ההתקן השליט ממתין לקליטת הנתונים מהתקן העבד.

להלן ההוראות לקריאה על ידי התקן שליט:

requestForm(address, quantity)	ההתקן השליט מבקש כמות (quantity) נתונים מהעבד לפי כתובתו (address).
while (Wire.available() > 0) { ... Wire.read(); }	כל עוד יש נתונים בחוצץ הקלט ייקראו הנתונים מהחוצץ.

הפעלת ההוראה הראשונה כפי שמופיע להלן מאפשרת לבדוק את תוכן המשתנה amount ולדעת כמה בתים התקבלו בפועל:

```
amount = Wire.requestForm(address, quantity);
```

כתיבה על ידי התקן עבד

התקן עבד יגיב לבקשת קריאת נתונים שיוזם התקן שליט. מכיוון שהתקשורת היא א-סינכרונית אי אפשר לקבוע מתי תגיע בקשה כזו. כדי לטפל באירועים א-סינכרוניים מגדירים פעולה שתופעל כאשר מתקבלת בקשה להעברת נתונים להתקן השליט. הפעולה תירשם כפעולה שתופעל כאשר בקשת מידע מתקבלת.

להלן הוראת רישום: Wire.onRequest(myHandler);

להלן הגדרת פעולת הכתיבה על ידי התקן עבד:

```
void myHandler() {
    Wire.write("string");
    Wire.write(val);
}
```

קריאה על ידי התקן עבד

התקן עבד יגיב לבקשת כתיבת נתונים שיוזם התקן שליט. מכיוון שהתקשורת היא אסינכרונית אי אפשר לקבוע מתי תגיע בקשה כזו. כדי לטפל באירועים א-סינכרוניים מגדירים פעולה שתופעל כאשר מתקבלת בקשה לקבלת נתונים מהתקן השליט. הפעולה תירשם כפעולה שתופעל כאשר בקשת קריאת מידע מתקבלת.

הוראת רישום: `Wire.onReceive(myHandler);`

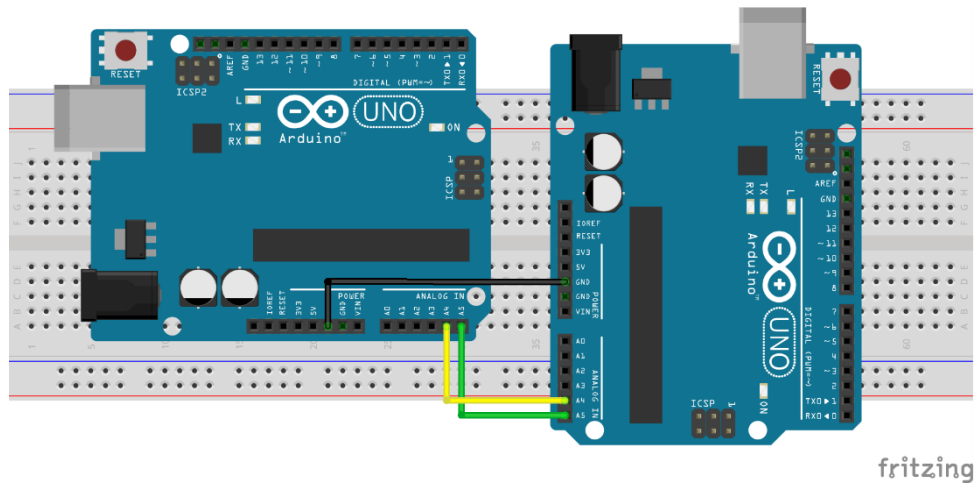
הגדרת פעולת הקריאה על ידי התקן עבד:

```
void myHandler(int numBytes) {
    while (Wire.available() > 0) {
        Wire.read();
        ...
    }
}
```

דוגמה 4.1 – תקשורת בין שני לוחות פתוח UNO שליט כותב/עבד קורא

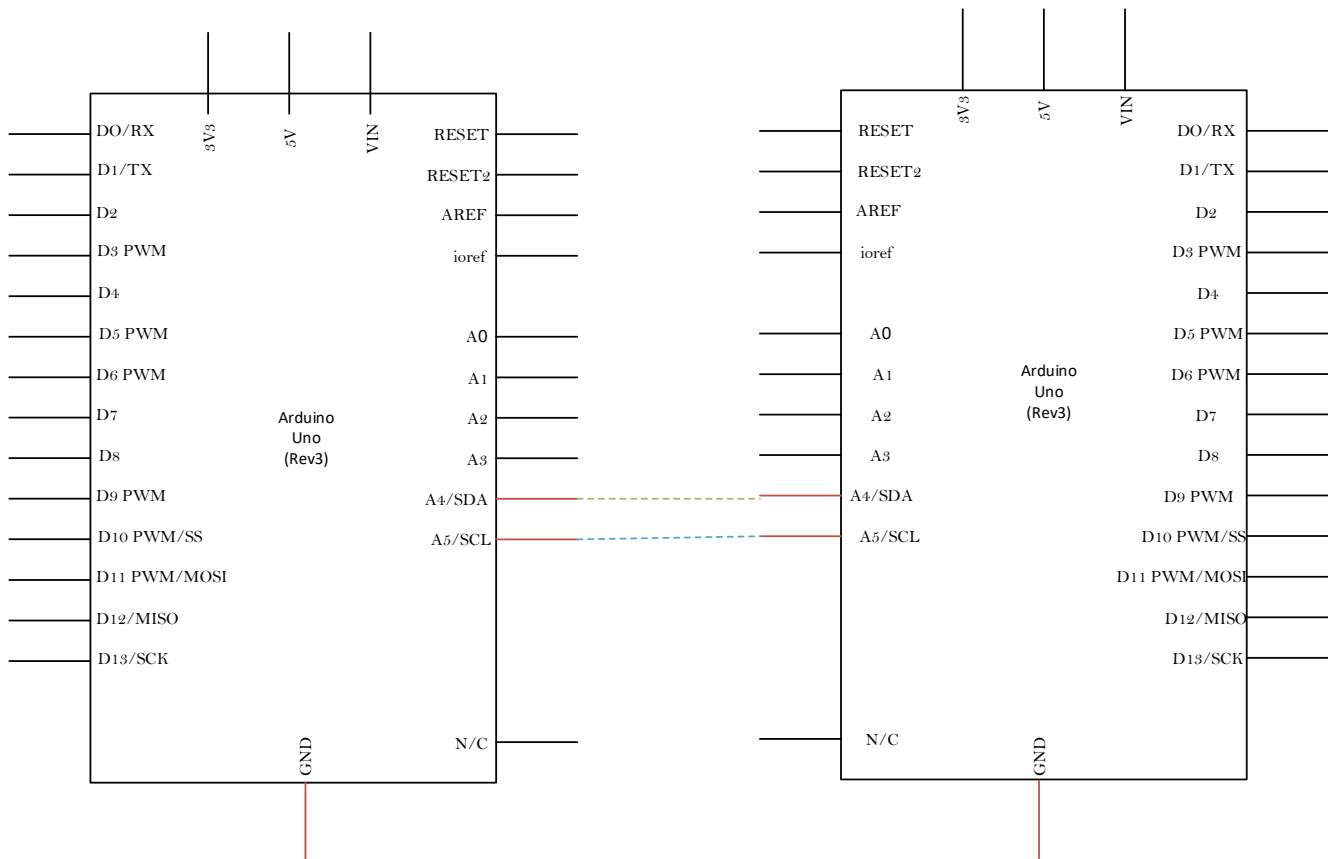
בדוגמה זו ניצור תקשורת בין שני לוחות פתוח. האחד ישמש כהתקן שליט, והאחר כהתקן עבד. ההתקן השליט ישלח מידע להתקן העבד ובו בזמן יציג את המידע ב- Serial Monitor. ההתקן העבד יקלוט את המידע ויציג אותו ב- Serial Monitor.

איור 4.11 מציג את אופן החיבור בין שני הלוחות:



איור 4.11 – חיבור לוחות בתקשורת I²C

איור 4.12 מציג את התרשים החשמלי לחיבור בין הלוחות:



איור 4.12 – תרשים חשמלי לחיבור בין הלוחות

קוד לוח שליט

```

include <Wire.h>                                // שילוב ספריית התקשורת
                                                // הגדרת קבועים
const char MSG[] = "From Master to slave";      // מחרוזת לשידור
const byte MSG_SIZE = sizeof(MSG);              // חישוב אורך המחרוזת
const byte SLAVE_ADDRESS = 8;                  // מספר התקן העבד

void setup () {
  Wire.begin();                                // I2C אתחול תקשורת
  Serial.begin(19200);                          // אתחול תקשורת טורית
  Serial.println("Master");                      // הצגת הודעה במוניטור
}

void loop () {
                                                // הצגת מידע במוניטור

  Serial.print("Sending ");
  Serial.print(MSG_SIZE-1);
  Serial.println(" chars");
  Serial.print(MSG);
  Serial.print("#");
  Serial.println(SLAVE_ADDRESS);

                                                // שידור מידע לעבד
  Wire.beginTransmission(SLAVE_ADDRESS);        // אתחול שידור לעבד
  Wire.write(MSG_SIZE-1);                       // כתיבת מספר המציין את גודל מחרוזת
  Wire.write(MSG);                              // כתיבת מחרוזת לתור התקשורת
  Wire.write(SLAVE_ADDRESS);                    // כתיבת מספר לתו המציין את מספר העבד
  Wire.endTransmission();                       // שידור חוצץ השידור ואיתות סיום השידור
  delay(5000);
}

```

שימו לב: קצב התקשורת שנקבע הוא 19,200, ואילו ברירת המחדל בתוכנת המוניטור היא 9,600.

קוד לוח עבד

```

#include <Wire.h>                // שילוב ספריית התקשורת
                                // הגדרת קבועים
const byte SLAVE_ADDRESS = 8;   // מספר התקן העבד
                                // משתני התכנית
char c;                          // משתנה לקליטת תו
int i;                           // משתנה המציין את מספר התו הנקלט
byte val;                        // משתנה הקולט את מספר העבד

void setup() {
  Wire.begin(SLAVE_ADDRESS); // I2C אתחול תקשורת
  Wire.onReceive(myHandler);  // רישום פעולת קליטה
  Serial.begin(19200);        // אתחול תקשורת טורית
  Serial.println("Slave");     // הצגת הודעה במוניטור
}

void loop() {
}

void myHandler(int numberOfBytes) {
                                // קליטת אורך המחרוזת
  if (Wire.available()) {      // בדיקה אם ישנו מידע בחוצץ הקליטה
    i = Wire.read();           // אחסון אורך המחרוזת
    Serial.print("Messgae length is ");
    Serial.println(i);
  }

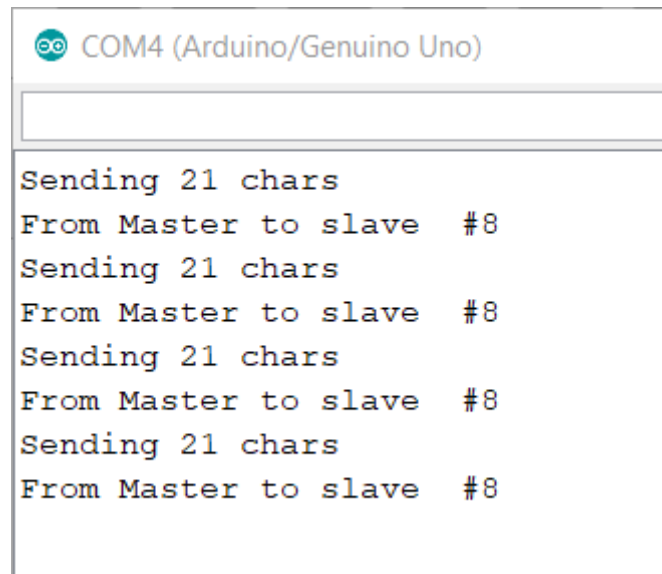
                                // קליטת המחרוזת ומספר העבד
  while(Wire.available()) {    // לולאת בדיקה אם ישנו מידע בחוצץ הקליטה
    if (i>0) {
      c = Wire.read();         // קריאת תו מחוצץ הקליטה
      Serial.print(c);         // הצגת התו במוניטור
      i--;
    }
    else {
      val = Wire.read();       // קריאת תו מספר העבד
      Serial.println(val);     // הצגת המספר במוניטור
    }
  }
}

```

שימו לב: קצב התקשורת שנקבע הוא 19,200, ואילו ברירת המחדל בתוכנת המוניטור היא

9,600

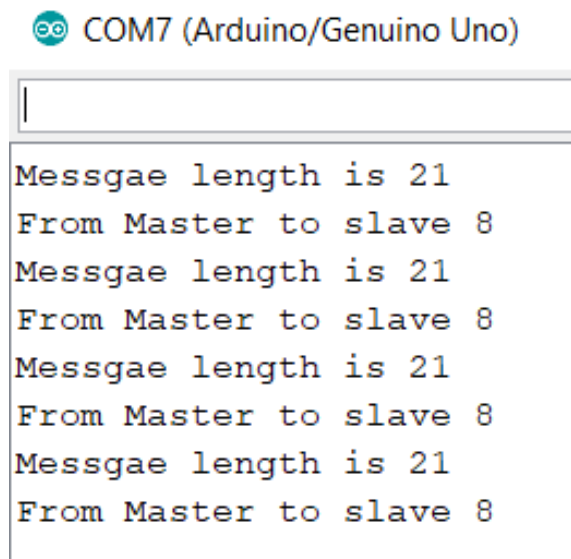
לאחר הרצת התכנית בשני הלוחות יתקבל את הפלט הזה:



```
COM4 (Arduino/Genuino Uno)

Sending 21 chars
From Master to slave #8
Sending 21 chars
From Master to slave #8
Sending 21 chars
From Master to slave #8
Sending 21 chars
From Master to slave #8
```

איור 4.13 – קטע מחלון המוניטור בלוח השליט



```
COM7 (Arduino/Genuino Uno)

Messgae length is 21
From Master to slave 8
Messgae length is 21
From Master to slave 8
Messgae length is 21
From Master to slave 8
Messgae length is 21
From Master to slave 8
```

איור 4.14 – קטע מחלון המוניטור בלוח העבד

הערה: אם מפעילים את שני הלוחות מאותו מחשב חייבים לפתוח את סביבת הפיתוח פעמיים (לא באמצעות File → New).

תרגול

שאלה 4.1

כמה בתים ממלאים את חוצץ השידור בכל פעם שהפעולה loop מתבצעת?

שאלה 4.2

בקוד לוח שליט בדוגמה 1 מחליפים את השורה:

```
const char MSG[] = "From Master to slave"; // מחרוזת לשידור
```

בשורה הזאת:

```
const char MSG[] = "##abcdefghijklmnopqrstuvwxyzz##"; // מחרוזת לשידור
```

כמה בתים ימלאו את חוצץ השידור בכל פעם שהפעולה loop מתבצעת?

שאלה 4.3

מה יקרה אם במקום השורה החדשה שמופיעה בשאלה 4.2 תבוא השורה הזאת?

```
const char MSG[] = "!##abcdefghijklmnopqrstuvwxyzz##"; // מחרוזת לשידור
```

שאלה 4.4

נתון הקטע שלפניכם מתכנית מסוימת:

```
Wire.beginTransmission(0x30);
Wire.write(100);
Wire.write(200);
Wire.write(300);
Wire.endTransmission();
```

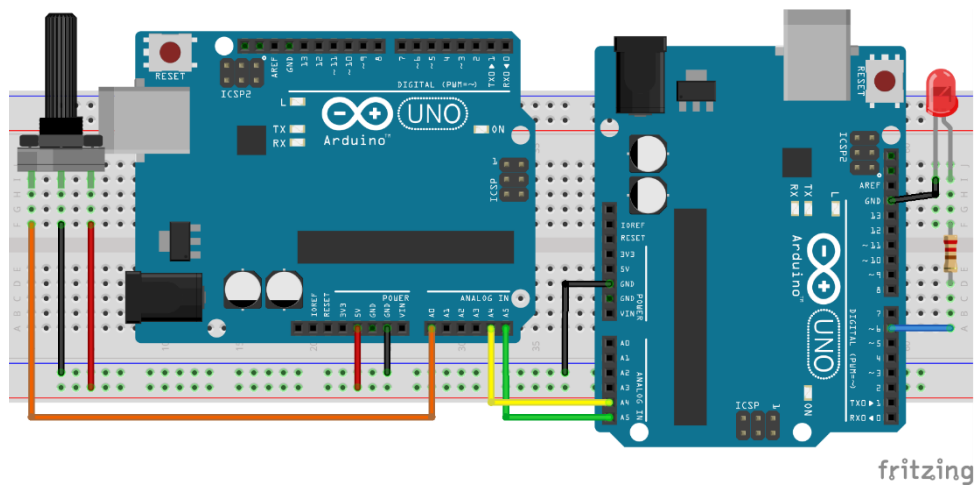
- א. קבעו לאיזה התקן נקבע הקוד – עבד או שליט? הסבירו.
- ב. מה משמעות הערך 0x30 בקוד?
- ג. מה משמעות הערכים 100, 200, 300 בקוד?
- ד. מהו התהליך המתרחש בקטע הקוד?
- ה. בהנחה שההתקן האחר הוא לוח פיתוח UNO, כתבו קטע קוד המטפל במידע שעובר בתהליך התקשורת.

דוגמה 4.2 – תקשורת בין שני לוחות פיתוח UNO: שליט קורא ועבד כותב

בדוגמה זו ניצור תקשורת בין שני לוחות פיתוח. האחד ישמש כהתקן שליט ואחר כהתקן עבד. ההתקן השליט ישלח בקשת מידע מהתקן העבד ויציג את המידע ב-Serial Monitor. התקן העבד ישדר את המידע ויציג אותו ב-Serial Monitor.

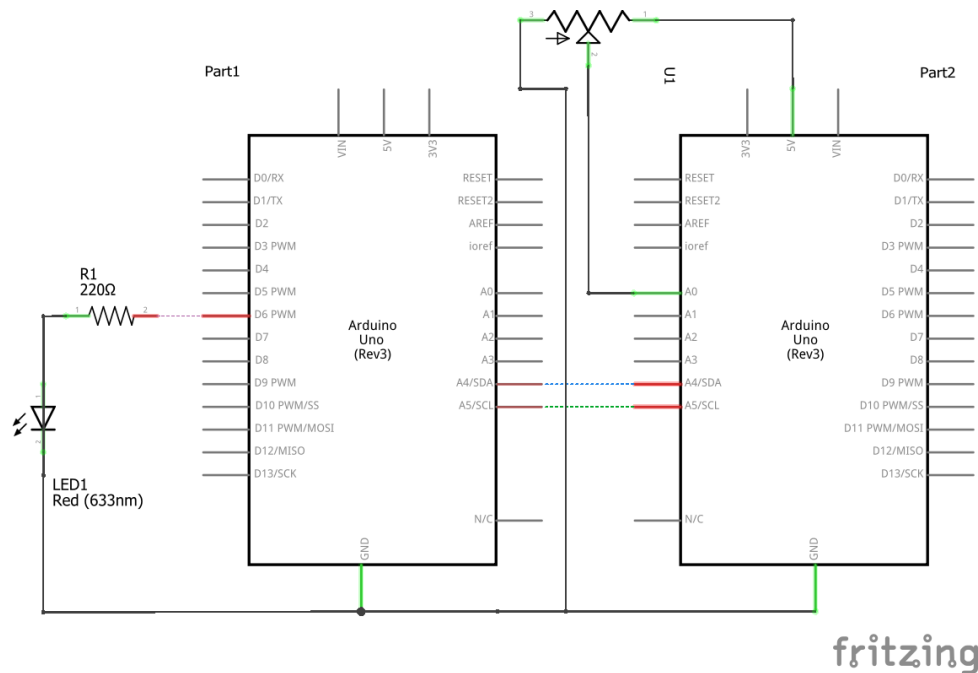
להתקן העבד נחבר לכניסה האנלוגית A0 פוטנציומטר ולהתקן השליט נחבר נורת LED. עוצמת ההפעלה של הנורה תהיה יחסית לערך המתח בפוטנציומטר כפי שקורא אותו התקן העבד.

איור 4.15 מתאר את החיבורים בלוחות וביניהם.



איור 4.15 – תרשים חיבורים בלוחות וביניהם.

איור 4.16 מתאר את הסכימה החשמלית של הלוחות.



איור 4.16 – תרשים חיבורים החשמליים של הלוחות

קוד לוח שליט

```

#include <Wire.h>                // שילוב ספריית התקשורת
                                // הגדרת קבועים

const byte SLAVE_ADDRESS = 8;    // מספר התקן העבד
const int pwmPin = 6;            // הדק יציאה אנלוגית
                                // משתנה התכנית

int adcValue;                    // ערך האות האנלוגי

void setup() {
  Wire.begin();                  // I2C אתחול תקשורת
  Serial.begin(19200);           // אתחול תקשורת טורית
  Serial.println("Master");      // הצגת הודעה במוניטור
}

void loop(){
                                // בקשה מהעבד לשלוח נתונים

  int amount = Wire.requestFrom(SLAVE_ADDRESS, 2);
  if (amount == 2){              // בדיקה אם מספר הנתונים שהתקבל תקין
    adcValue = Wire.read();       // קריאת הבית הנמוך
    adcValue = adcValue + Wire.read() * 256; // קריאת הבית הגבוה וחישוב כולל
    analogWrite(pwmPin, adcValue / 4); // יצירת אות אנלוגי
                                // הצגת מידע במוניטור

    Serial.print("received ");
    Serial.println(adcValue);
  }
  delay(5000);
}

```

שימו לב: קצב התקשורת שנקבע הוא 19,200, ואילו ברירת המחדל בתוכנת המוניטור היא 9,600.

קוד לוח עבד

```

#include <Wire.h> // שילוב ספריית התקשורת
                // הגדרת קבועים
const byte SLAVE_ADDRESS = 8; // מספר התקן העבד
                // משתנה התכנית
int adcValue; // משתנה לערך אנלוגי

void setup() {
  Wire.begin(SLAVE_ADDRESS); // I2C אתחול תקשורת
  Wire.onRequest (myHandler); // רשום פעולת בקשה
  Serial.begin(19200); // אתחול תקשורת טורית
  Serial.println("Slave"); // הצגת הודעה במוניטור
}
void loop() {
}
void myHandler(int numberOfBytes) {
  adcValue = analogRead(A0); // קריאת ערך אנלוגי
                             // הצגת הערך האנלוגי במוניטור

  Serial.print("Sending ");
  Serial.println(adcValue);

  byte lowAdc = adcValue % 256; // חישוב הבית הנמוך של הערך האנלוגי
  byte highAdc = adcValue / 256; // חישוב הבית הגבוה של הערך האנלוגי
  Wire.write(lowAdc); // שידור הבית הנמוך
  Wire.write(highAdc); // שידור הבית הגבוה
}

```

שימו לב: קצב התקשורת שנקבע הוא 19,200, ואילו ברירת המחדל בתוכנת המוניטור היא 9,600.

שאלה 4.5

בקוד לוח העבד בדוגמה 4.2 מופיעות שורות הקוד האלה:

```
byte lowAdc = adcValue % 256;    // חישוב הבית הנמוך של הערך האנלוגי  
byte highAdc = adcValue / 256;    // חישוב הבית הגבוה של הערך האנלוגי
```

- א. מדוע משתמשים בשתי הוראות כדי להעביר את הערך האנלוגי?
- ב. כתבו הוראות אחרות המבצעות את אותה הפעולה.

שאלה 4.6

בקוד הלוח שליט בדוגמה 4.2 בודקים אם התקבלו בדיוק 2 בתיים.

מה יקרה אם התקן עבד ישלח כמות בתיים שונה מ-2 בתיים?

דוגמה 4.3 – מדידת מרחק באמצעות חיישן אולטרה-סוני SRF08

בדוגמה זו ניצור תקשורת בין לוח פיתוח המשמש כהתקן שליט ובין חיישן SRF08 המשמש כהתקן עבד. ההתקן השליט ישלח בקשת מידע מהתקן העבד ויציג את המידע ב- Serial Monitor.

כתובת הבסיס, ברירת המחדל, של החיישן האולטרה-סוני היא E0 לפי בסיס 16 (0xE0). ספריית Wire מצפה לקבל כתובת בת 7 סיביות ולכן נדרש לשנות את ערך הכתובת ל-70 (0x70) לפי בסיס 16. הפעולה מתבצעת באמצעות הזזת הסיביות מקום אחד ימינה.

HEX	7	6	5	4	3	2	1	0
E0	1	1	1	0	0	0	0	R/W
הזזת הסיביות ימינה פעם אחת								
70	0	1	1	1	0	0	0	0

זמן המדידה של החיישן הוא כ-65mSec.

לחיישן האולטרה-סוני יש 36 אוגרים פנימיים. מתוך האוגרים נשתמש באוגר ההוראה COMMAND שכתובתו 0 ובשני אוגרים נוספים: 2 ו-3, המאחסנים את ערך הקריאה האחרונה של החיישן. כל האוגרים הנוספים מאחסנים את הקריאות הנוספות של החיישן.

להלן מוצגים 4 האוגרים הראשונים של החיישן:

כתובת אוגר	קריאה	כתיבה
0	גרסת התוכנה של ההתקן	אופן פעולה
1	מצב חיישן אור	קביעת עוצמת ההגבר האנלוגי
2	הערך הגבוה של אות ההד	קביעת טווח המדידה
3	הערך הנמוך של אות ההד	אין משמעות

אוגר ההוראה בכתובת 0 קובע את אופן פעולת הרכיב.

להלן חלק מהוראות הפעולה של החיישן:

קוד הוראה HEX	משמעות ההוראה
0x50	מדידת מרחק ביחידות אינטש
0x51	מדידת מרחק ביחידות סנטימטר
0x52	מדידת מרחק לפי זמן ביחידות מיקרו-שניות

תרגול

שאלה 4.7

חפשו באינטרנט דפי נתונים של הרכיב SRF08, וענו על השאלות להלן:

- א. מהו ערך ברירת המחדל של עוצמת ההגבר האנלוגי בחיישן?
- ב. מעוניינים בהגבר אנלוגי של כ-200. מהו הערך שיש לשלוח לאוגר 1?
- ג. מהו ערך ברירת המחדל של טווח המדידה?
- ד. מהו הערך שיש להעביר לאוגר 2 כדי שטווח המדידה יהיה עד טווח של מטר אחד?

להלן קוד התכנית מדידת מרחק באמצעות חיישן אולטרה-סוני SRF08

```

#include <Wire.h> // שילוב ספריית התקשורת
const byte SLAVE_ADDRESS = 0x70; // מספר התקן העבד
// משתנה התכנית

int reading; // ערך הקריאה

void setup() {
  Wire.begin(); // I2C אתחול תקשורת
  Serial.begin(9600); // אתחול תקשורת טורית
}

void loop(){
  // שידור הודעה למדידת טווח

  Wire.beginTransmission(SLAVE_ADDRESS); // שידור להתקן התקשורת
  Wire.write(byte(0x00)); // בחירת אוגר פנימי 0
  Wire.write(byte(0x51)); // הוראת מדידה ביחידות סנטימטר
  Wire.endTransmission(); // שידור התור וסיום המנה
  // המתנה לסיום ביצוע המדידה

  delay(70);

  // קריאת הטווח מאוגר 2

  Wire.beginTransmission(SLAVE_ADDRESS); // שידור להתקן התקשורת לצורך קריאה
  Wire.write(byte(0x02)); // בחירת אוגר פנימי 2
  Wire.endTransmission(); // שידור התור וסיום המנה

  // בקשת קריאה של 2 בתים

  Wire.requestFrom(SLAVE_ADDRESS, 2);

  // קריאת ערך החיישן
  // בדיקת מספר הבתים שנקלטו
  if (2 <= Wire.available()) {
    reading = Wire.read(); // קריאת הערך הגבוה
    reading = reading << 8; // הזזה שמאלה 8 פעמים
    reading |= Wire.read(); // קריאת הערך הנמוך ושילוב עם הערך הגבוה
    Serial.println(reading); // הצגת המידע
  }
  delay(500);
}

```

שאלה 4.8

רוצים לקבוע לחיישן כתובת חדשה 0xF8 במקום ברירת המחדל 0xE0.

- א. על פי דפי המפרט של החיישן, מהו רצף הפעולות הדרוש?
- ב. כתבו פעולה בשם `changeAddress` לקביעת הכתובת החדשה.

שאלה 4.9

רוצים להוסיף לתכנית את קטע הקוד להלן:

```
Wire.beginTransaction(SLAVE_ADDRESS);
Wire.write(byte(0x02));
Wire.write(byte(0x46));
Wire.endTransmission();
```

- א. לאיזה אוגר מתבצעת פנייה?
- ב. מה משמעות הערך המספרי 0x46?
- ג. היכן צריך למקם את קטע הקוד? הסבירו.

שאלה 4.10

מערכת התראת מרחק כוללת חיישן מרחק SRF08 ורמקול המחובר להדק 6 בלוח הפיתוח. המרחק המרבי של המדידה יהיה 6 מטרים.

כתבו תכנית המשמיעה צליל בתדר משתנה על פי המרחק. אם המרחק 6 מטרים ומעלה, יושמע צליל בתדר 200Hz. ככל שהמרחק קצר יותר, יושמע תדר גבוה יותר. כשהמרחק 10 ס"מ, יושמע צליל בתדר 3150Hz.

הנחיות:

- השתמשו בפעולה `tone` כדי להשמיע את הצליל.
- השתמשו בפעולה `map` כדי לקבוע את התדר על פי המרחק.

4.9 הרחבת הדקים ספרתיים

לוח הפיתוח UNO המבוסס על המיקרו-בקר ATmega329P מאפשר שימוש ב-20 הדקים לשימושים שונים. אפשר להשתמש בהדקים לכניסות וליציאות ספרתיות, לכניסות ויציאות אנלוגיות, לקווי תקשורת ועוד.

מרחב האפשרויות אינו גדול, ולעיתים בפרויקטים דרושים הדקים בכמות גדולה יותר. אפשרות אחת היא לבחור לוח פיתוח בעל כמות הדקים גדולה יותר, כמו לוח הפיתוח ATMEGA 2560. אפשרות אחר היא להוסיף רכיב נוסף המרחיב את כמות ההדקים, כמו הרכיב PCF8574.

4.10 רכיב PCF8574 מרחיב I/O של 8 סיביות עם פס I2C

להלן המאפיינים העיקריים של הרכיב PCF8574:

- צריכת זרם נמוכה במצב המתנה.
- ממשק תקשורת טורית מסוג I²C.
- הדקי יציאה המסוגלים לספק זרם גבוה ללדים.

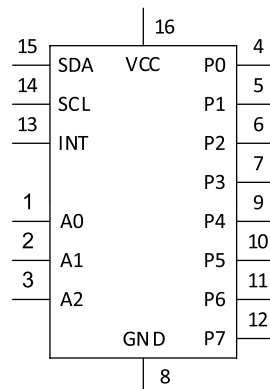
תיאור הרכיב

הרכיב תוכנן לפעול בתחום המתחים 2.5v עד 6v, וכן לספק הרחבה של הדקים ספרתיים לשימוש כולל (GPIO) עבור מרבית המיקרו-בקרים באמצעות ממשק I²C.

לרכיב 8 הדקים (P0-P7) דו-כיווניים המסוגלים לספק זרם הפעלה ללדים. כל אחד מההדקים יכול לשמש לקלט או לפלט ללא צורך בבקרת כיוון של המידע. בזמן ההפעלה כל יציאות הרכיב במצב גבוה, והמידע בכל הדק נשמר באמצעות נועל (LATCH).

תיאור הדקים

באיור 4.17 להלן מתוארים הדקי הרכיב:



איור 4.17 – הדקי הרכיב

הדקים P7–P0 משמשים לפי הצורך לקלט או לפלט.

הדקים A2–A0 הם הדקי כניסה המשמשים לקביעת הכתובת של הרכיב.

הדקי SDA ו-SCL משמשים לתקשורת I²C.

הדק INT הוא הדק יצאה המספק מידע על התהליך. אפשר להשתמש בו למטרות פסיקה חיצונית במיקרו-בקרי.

הדקים VCC ו-GND משמשים כהדקי אספקת המתח.

קביעת הכתובת

HEX	7	6	5	4	3	2	1	0
4X	0	1	0	0	A2	A1	A0	R/W
הזזת הסיביות ימינה פעם אחת								
2X	0	0	1	0	0	A2	A1	A0

במקרה שכניסות A2, A1, A0 מחוברות לאדמה נקבל את הכתובת 0x20.

 תרגול

שאלה 4.11

- כמה רכיבים מסוג PCF8574 אפשר לחבר בערוץ התקשורת מסוג I²C?
- מהו תחום הכתובות של הרכיבים?

ערך המפתח

כאשר קולטים או משדרים לרכיב את המידע אגור באופן הזה:

7	6	5	4	3	2	1	0
P7	P6	P5	P4	P3	P2	P1	P0

למשל, אם נשדר את הצירוף הבינארי 01010101, היציאות P0, P2, P4 ו-P6 יהיו ברמה לוגית 1, והיציאות האחרות יהיו ברמה לוגית 0.

קביעת P2=0

קוד לכתיבת ערך 0 למפתח P2 לרכיב בעל כתובת בסיס 0x40 (כתובת 0x20 כאשר משתמשים בסביבת הפיתוח).

```
Wire.beginTransmission(0x20);  
Wire.write(byte(0xFB));          // 1111 1011  
Wire.endTransmission();
```

קריאת ערך P3

```
Wire.requestFrom(0x20, 1);  
  
if (Wire.available()) {          // האם נקלט  
    reading = Wire.read();        // קריאת ערך  
    reading = reading & 0x08;     // פעולת מסיכה  
}
```

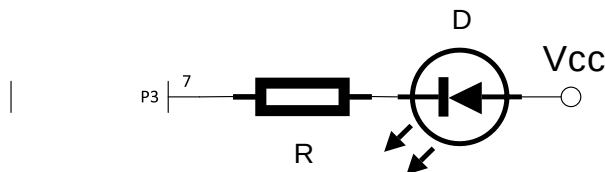
4.11 ניסוי PCF8574

4.11.1 מטרת הניסוי:

- א. הכרת המאפיינים החשמליים של הרכיב PCF8574.
- ב. הכרת פרוטוקול I^2C של הרכיב.
- ג. כתיבת תכנית לשימוש ברכיב.

4.11.2 שאלות הכנה

1. חפשו באינטרנט דפי מפרט של הרכיב PCF8574.
2. על פי דפי המפרט מהו הזרם המרבי בהדק יציאה נמוך?
3. מהו מתח היציאה V_{OL} במתח הפעלה $V_{CC} = 5V$ ובזרם של $10mA$?
4. מהו ערכו של הנגד R (המופיע באיור 4.18 להלן) שיש לחבר לדiodת LED הפועלת במתח של $1.8V$ ובזרם של $10mA$. נתון כי $V_{CC} = 5V$. היעזרו בתשובה לשאלה הקודמת.



איור 4.18 חיבור לד למפתח P3

5. שרטטו מעגל הכולל: לוח פיתוח UNO, רכיב PCF8574, דiodת LED המחוברת למפתח P3 ומפסק לחיצה המחובר למפתח P6. הוסיפו נגדים לפי הצורך. את השרטוט מומלץ לבצע בתוכנה fritzing.

4.11.3 ציוד נדרש

1. מחשב שמותקנת בו סביבת הפתוח של Arduino.
2. לוח פיתוח UNO וכבל תקשורת.
3. מטריצה וחוטי חיבור.
4. נורת LED ונגד $2k\Omega$.
5. מפסק לחצן ונגד $10k\Omega$.
6. רכיב PCF8574 ושני נגדים $10k\Omega$.

4.11.4 מהלך ניסוי

1. בנו את המערכת שתכנתם בשאלת ההכנה 5.
2. כתבו תכנית להפעלת דיודת ה-LED המחוברת למפתח P3 כך שתבהב בקצב של 0.1 שנייה.
3. הריצו את התכנית.
4. כתבו תכנית הגורמת להבהוב ה-LED כל עוד לוחצים על המפסק.
5. הריצו את התכנית.

תרגול 

4.12 שאלה

תלמיד כתב את קטע הקוד שלפניכם:

```
Wire.beginTransmission(0x20);
Wire.write(byte(0xF0));
Wire.endTransmission();

Wire.requestFrom(0x20, 1);
if (Wire.available()) {           // האם נקלט
  reading = Wire.read();          // קריאת ערך
  reading = reading & 0x08;       // פעולת מסיכה
}
```

א. מה צפוי להיות ערכו של המשתנה `reading`?

ב. מהי המסקנה מהתוצאה?

4.12 שעות זמן אמת

שעות זמן אמת (RTC – Real Time Clock) הוא שעות שמסוגל לספק מידע לגבי התאריך והשעה. רוב המיקרו-בקרים אינם כוללים בתוכם שעות זמן אמת. אפשר לממש שעות בתוכנה תוך שימוש בקוצבי הזמן הפנימיים במיקרו-בקר. חסרנו של השעות הוא בצורך לכוון אותו בכל פעם מחדש מכיוון שאינו שומר על ערך התאריך והשעה לאחר הפעלה מחדש של המיקרו-בקר (הפעלה קרה). לשם מעקב אחרי התאריך והשעה קיימת מחלקה בשם Time שתפקידה לספק את המידע. השימוש במחלקה מצריך שימוש במשאבי המיקרו-בקר: קוצבי זמן וקוד תכנית.

דרך יעילה יותר לממש שעות זמן אמת הוא שעות הממומש בחומרה, כמו הרכיב DS1307. אפשר לחבר לרכיב סוללת גיבוי כדי לשמור על רציפות הזמן. ואכן, מרבית המודולים כוללים את הרכיב עם סוללת גיבוי וממשק I²C לחיבור למיקרו-בקר.

4.13 רכיב DS1307 שעות זמן אמת

להלן מאפיינים עיקריים של רכיב DS1307:

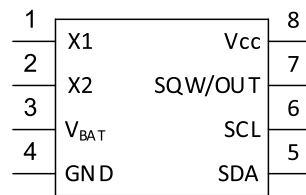
- שעות זמן אמת הסופר שניות, דקות, שעות, יום בחודש, חודש, יום בשבוע, ושנה עם תיקון לשנים מעוברות עד שנת 2100.
- 56 בתים מסוג RAM המגובים באמצעות סוללה.
- ממשק תקשורת טורית מסוג I²C.
- מוצא גל ריבועי בר תכנות.
- מערכת גילוי ומיתוג בעקבות נפילת מתח.

תיאור הרכיב

הרכיב, שעות זמן אמת, מאחסן את מידע השעה והתאריך לפי קוד BCD וכולל גם 56 בתים המאוחסנים בזיכרון RAM סטטי. התקשורת עם הרכיב מתבצעת באמצעות זוג חוטים בתקשורת I²C. הרכיב מטפל בחודשים השונים ובשנה מעוברת לפי הצורך. השעות יכול לפעול במחזור של 12 שעות (AM/PM) או 24 שעות. כמו כן הרכיב כולל גלאי נפילת מתח וממתג את אספקת המתח אוטומטית לסוללת הגיבוי.

תיאור הדקים

באיור 4.19 להלן מתוארים הדקי הרכיב:



איור 4.19 – הדקי הרכיב

הדקים X1 ו-X2 משמשים לחיבור גביש בתדר 32768Hz.

הדק V_{BAT} משמש לחיבור סוללת ליתיום בצורת כפתור במתח של 3V. הסוללה יכולה להחזיק מעמד עד 10 שנים.

הדקי SDA ו-SCL משמשים לתקשורת I²C.

הדק SQW/OUT הוא הדק יציאה המספק אות ריבועי באחד מארבע תדרים (1Hz, 4kHz, 8kHz, 32kHz) אם מאפשרים לו באמצעות הוראה מתאימה.

הדקים VCC ו-GND משמשים כהדקי אספקת המתח.

קביעת הכתובת

HEX	7	6	5	4	3	2	1	0
D0	1	1	0	1	0	0	0	R/W
הזזת הסיביות ימינה פעם אחת								
68	0	1	1	0	1	0	0	0

מפת הכתובות

הבתים 0 עד 7 ברכיב מתייחסים לשעה ותאריך, ואילו הבתים מ-8 עד 63 הם תאי זיכרון מסוג RAM.

כתובת	0	1	2	3	4	5	6	7	משמעות
0	שניות			10 שניות			CH		00-59
1	דקות			10 דקות			0		00-59
2	שעות			10 שעות			1	0	01-12
2	שעות			10 שעות			0	0	00-23
3	יום			0	0	0	0	0	1-7
4	תאריך			10 תאריך			0	0	01-28/29 01-30 01-31
5	חודש			10 חודש			0	0	01-12
6	שנה			10 שנים			00-99		
7	RS0	RS1	0	0	SQWE	0	0	OUT	

סיבית 7 בכתובת CH 0 (Clock Halt)

כאשר CH=1, השעון אינו פועל; כאשר CH=0, השעון פועל.

סיבית 5 בכתובת A/P 2

AM – 0, PM – 1.

סיבית 4 בכתובת SQWE 7

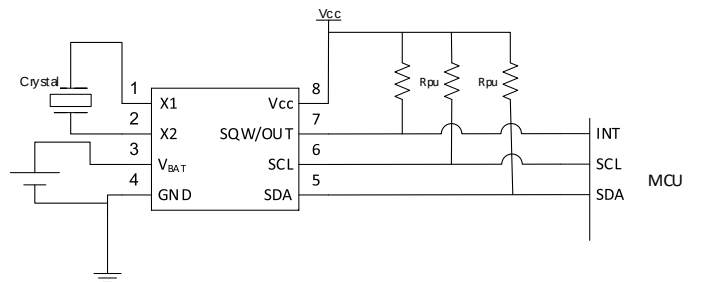
כאשר $SQWE = 1$ הגל הריבועי מאפשר. בהדק ה-OUT ברכיב יופק גל ריבועי הנקבע על ידי הסיביות 0 ו-1 בכתובת 7.

כאשר $SQWE = 0$, הגל הריבועי אינו מאפשר. הדק ה-OUT ברכיב ייקבע על פי הערך של סיבית 7 בכתובת 7 – OUT.

הטבלה להלן מציגה את ערכי התדר האפשריים בהדק OUT על פי ערכי סיביות 0 ו-1 בכתובת 7:

תדירות	RS1	RS0
1Hz	0	0
4.096kHz	0	1
8.192kHz	1	0
32.768kHz	1	1

האיור 4.20 להלן מתאר את החיבור בין מיקרו-בקר לרכיב. החיבור להדק INT במיקרו-בקר הוא אופציונלי. אפשר לרכוש את הרכיב כמודול הכולל סוללה, גביש, נגדים והדקי חיבור למיקרו-בקר.



איור 4.20 – תרשים חשמלי DS1307 למיקרו בקר

קוד לאתחול הזמן 14:20 – ללא שניות

```
Wire.beginTransmission(0x68);           // פנייה לשעון
Wire.write(byte(0x01));                  // כתובת פנימית 1
Wire.write(byte(0x20));                  // כתובת פנימית 1 דקות
Wire.write(byte(0x14));                  // כתובת פנימית 2 שעות תבנית 24
Wire.endTransmission();                  // שידור הבתים וסיום המנה
```

הערכים נשלחים לפי קוד בבסיס 16 התואם לערכים לפי קוד BCD.

קוד לקריאת הזמן – שעה בלבד והצגתה

```
Wire.beginTransmission(0x68);           // פנייה לשעון
Wire.write(byte(0x02));                  // כתובת פנימית 1
Wire.endTransmission();                  // שידור הבתים וסיום המנה

Wire.requestFrom(0x68, 1);
if (Wire.available()) {                  // האם נקלט ערך חדש
    hour = Wire.read();                  // קריאת ערך
    hour_1 = hour & 0x0F;                // פעולת מסיכה ליחידות השעה
    hour_10 = (hour & 0xF0) >> 4;       // מסיכה לחלק העשרות של השעה
    Serial.print("Hour is: ");
    Serial.print(hour_10);
    Serial.print(hour_1);
}
```


תרגול 

שאלה 4.13

כתבו תכנית לאתחול השעון לשעה 13:33:55. לאחר האתחול התכנית תציג את השניות בלבד. השניות יוצגו פעם אחת בשנייה בלבד.

4.14 ניסוי DS1307

4.14.1 מטרות הניסוי:

- א. הכרת המאפיינים החשמליים של הרכיב DS1307.
- ב. הכרת פרוטוקול I^2C של הרכיב.
- ג. כתיבת תכנית לשימוש ברכיב.

4.14.2 שאלות הכנה

1. חפשו באינטרנט דפי מפרט של הרכיב DS1307.
 2. על פי דפי המפרט מהו מתח ההפעלה של הרכיב?
 3. על פי דפי המפרט מהו תחום מתחי הפעולה של הסוללה?
 4. על פי דפי המפרט מהו התדר המרבי של קו התקשורת SCL ?
 5. שרטטו מעגל הכולל את לוח הפיתוח UNO ואת הרכיב DS1307 (או מודול).
- את השרטוט מומלץ לבצע בתוכנה fritzing.

4.14.3 הציוד הדרוש:

1. מחשב שמותקנת בו סביבת הפיתוח של Arduino.
2. לוח פיתוח UNO וכבל תקשורת.
3. מטריצה וחוטי חיבור.
4. רכיב DS1307 (או מודול מתאים).

4.14.4 מהלך הניסוי:

1. בנו את המערכת שתכננתם בשאלת ההכנה 5.
2. כתבו תכנית לאתחול הרכיב לשעה והתאריך הנוכחי. לאחר מכן הציגו את שעה כל שנייה.
3. הריצו את התכנית.

שאלה 4.14

תלמיד כתב את קטע הקוד שלפניכם:

```
Wire.beginTransaction(0x68);  
Wire.write(byte(0x03));  
Wire.endTransmission();  
  
Wire.requestFrom(0x68, 1);  
if (Wire.available()) {           // האם נקלט  
    x = Wire.read();               // קריאת ערך  
}
```

א. הסבירו מה מתבצע בקוד לעיל?

ב. בהנחה שהשעון מכוון לשעה ולתאריך הנוכחיים. מה הערך במשתנה x?

סיכום פרק 4

בפרק 4 למדתם את הנושאים האלה:

1. ארכיטקטורה של I^2C .
2. תקני שליט Master ועבד Slave.
3. אותות בערוץ Strat ו-Stop.
4. העברת סיבית בודדת
5. העברת כתובת רכיב וקביעת כיוון העברת המידע
6. פרוטוקול קריאה וכתיבה
7. מחלקת Wire ופעולות המחלקה
8. תכנית התקן שליט בארדואינו
9. תכנית התקן עבד בארדואינו
10. חיבור התקן חיישן אולטרה-סוני SRF08
11. חיבור יחידת הרחבה PCF8574
12. חיבור שעון זמן אמת DS1307

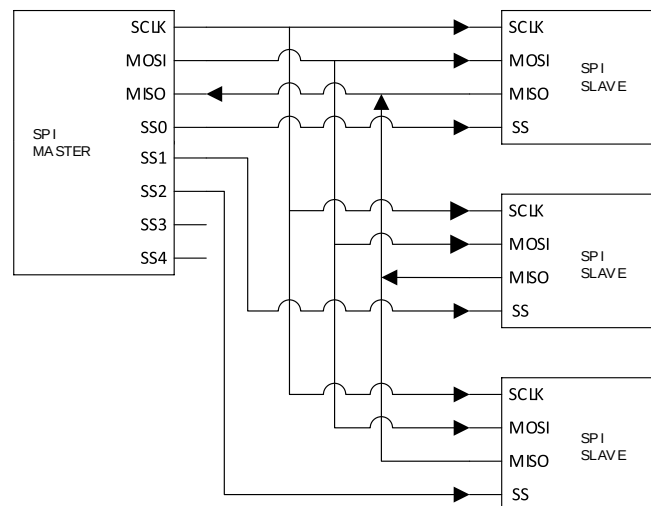
פרק 5: פרוטוקול SPI

פרוטוקול SPI (Serial Peripheral Interface) פותח באמצע שנות השמונים על ידי חברת מוטורלה (Motorola) כדי לאפשר תקשורת סינכרונית למרחקים קצרים, בעיקר במערכות משובצות. הפרוטוקול הפך לתקן ומשמש ביישומים כגון תצוגות LCD.

5.1 ארכיטקטורה

איור 5.1 מתאר את אופן החיבור בין ההתקנים שונים. המערכת מאפשרת תקשורת סינכרונית דו-כיוונית מלאה בין התקן שליט יחיד לבין התקני העבד במערכת תוך שימוש בהדקי התקשורת האלה:

שם	תיאור
SCLK – Serial Clock	אות שעון התקשורת
MOSI – Master Output Slave Input	אות מידע היוצא מהשליט אל העבד
MISO – Master Input Slave Output	אות מידע היוצא מהעבד אל השליט
SS – Slave Select	בחירת העבד, פעיל בנמוך



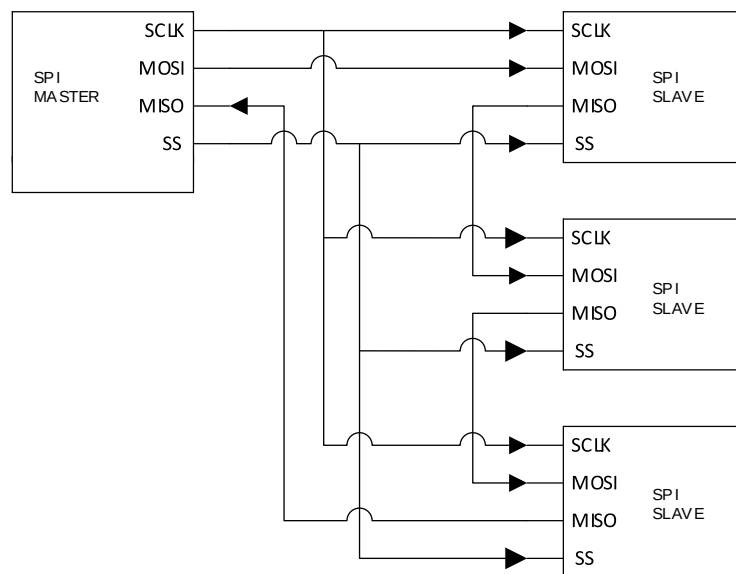
איור 5.1: ארכיטקטורת SPI התקני עבד עצמאיים

התקן שליט והתקני עבד

המערכת כוללת שליט אחד וכמה התקני עבד. התקשורת בין ההתקנים מתחילה ביוזמת ההתקן השליט. השליט פונה אל ההתקנים באמצעות הדק בחירה SS. התקן שלא נבחר על ידי השליט מעביר את הדק MISO למצב שלישי (עכבה גבוהה) כדי למנוע התנגשות על הקו.

האות בקו השעון SCLK נוצר על ידי ההתקן השליט בלבד ומספק להתקני העבד בקרת זרימה. המידע יוצא מהדק MOSI לכיוון העבד, ובו בזמן מוציא העבד מידע מהדק MISO לכיוון השליט, כך שבכל פעם שנשלחת סיבית לעבד מתקבלת סיבית בצד השליט.

איור 5.2 מתאר חיבור של התקן שליט ל-fnch התקני עבד עם קו בחירה יחיד בתצורה הנקראת Daisy Chain.



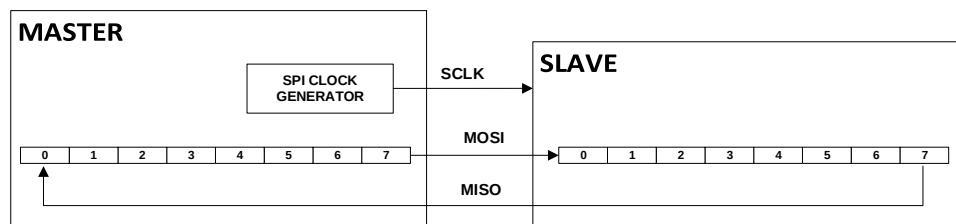
איור 5.2: ארכיטקטורת SPI בתצורת Daisy Chain

בתצורה זו יש קו בחירה אחד, והמידע עובר דרך כל רכיבי העבד.

5.2 אותות בערוץ התקשורת

התקשורת מתחילה ביוזמת ההתקן השליט. לאחר בחירת התקן העבד באמצעות קו SS, מתחיל ההתקן השליט לייצר אות שעון בתדר המתאים לעבד. בכל מחזור שעון ההתקן השליט שולח סיבית מידע אחת הנקלטת על ידי העבד, ובאותו מחזור העבד משדר לשליט סיבית.

איור 5.3 מתאר את המבנה הרעיוני של התקשורת בין יחידת שליט לעבד. בכל התקן יש אוגר הזזה. כאשר ההתקן השליט מוציא ערך לקו ה-MOSI העבד דוגם את הערך באמצעות קו השעון ומכניס את הנתון לאוגר ההזזה. התקן השליט קולט את הערך המתקבל בקו ה-MISO ומכניס אותו לאוגר ההזזה. אחרי שהשעון דופק שמונה פעמים, שני הצדדים מחליפים מידע הנמצא באוגרי ההזזה.



איור 5.3: העברת המידע בין ההתקנים

לרוב הסיבית המשמעותית ביותר (MSB) משודרת תחילה, כפי שניתן לראות באיור 5.3. אולם קיימת גם אפשרות לשדר את הסיבית הפחות משמעותית (LSB) תחילה.

מספר הסיביות המשודר הוא לרוב 8 סיביות, אולם קיימים התקנים המשדרים 16 סיביות.

פרט לתדירות אות השעון, ההתקן השליט קובע עוד שני מאפיינים של אות השעון: קוטביות (POLARITY) ומופע (PHASE). שני מאפיינים אלה מסומנים בהתקן ב-CPOL ו-CPHA בהתאמה.

מאפיין CPOL

קובע את הקוטביות של אות השעון.

$$\underline{CPOL = 0}$$

מציין מצב שבו בזמן ההמתנה אות השעון נמצא ברמה לוגית נמוכה (L), ובכל מחזור הוא עולה לרמה גבוהה (H) למשך זמן קצוב. השינוי המוביל (Leading Edge) הוא עלייה (Rising Edge) והשינוי המסיים (Trailing Edge) הוא ירידה (Falling Edge).

$$\underline{CPOL = 1}$$

מציין מצב שבו בזמן ההמתנה אות השעון נמצא ברמה לוגית גבוהה (H), ובכל מחזור הוא יורד לרמה נמוכה (L) למשך זמן קצוב. השינוי המוביל (Leading Edge) הוא ירידה (Falling Edge) והשינוי המסיים (Trailing Edge) הוא עלייה (Rising Edge).

מאפיין CPHA

קובע את תזמון הסיביות ביחס לאות השעון.

$$\underline{CPHA = 0}$$

הצד המשדר משנה את קו השידור (MOSI בשליט ו-MISO בצד העבד) בשינוי המסיים (Trailing Edge) במחזור הקודם, ואילו הצד הקולט דוגם את הסיבית בשינוי המוביל (Leading Edge) של אות השעון. במילים אחרות צד המשדר מוציא את המידע בזמן ההמתנה ודוגם את האות בזמן הופעת הדופק.

$$\underline{CPHA = 1}$$

הצד המשדר משנה את קו השידור (MOSI בשליט ו-MISO בצד העבד) בשינוי המוביל (Leading Edge) במחזור הקודם ואילו הצד הקולט דוגם את הסיבית בשינוי המסיים (Trailing Edge) של אות השעון. במילים אחרות הצד המשדר מוציא את המידע בזמן הופעת הדופק ודוגם את האות בזמן ההמתנה.

אופני שידור DATA MODES

CPOL	CPHA	Mode
0	0	0
0	1	1
1	0	2
1	1	3

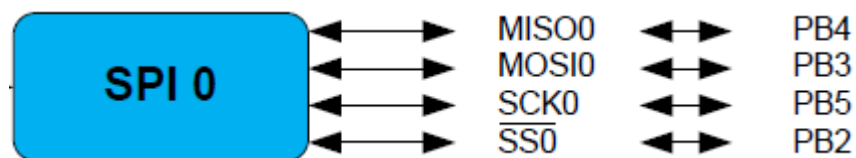
5.3 מפתח התקשורת במיקרו-בקר ATmega329P

מאפייני מפתח התקשורת

- תקשורת סינכרונית דו-כיוונית מלאה באמצעות 3 קווים
- ניתן להפעיל את מפתח התקשורת כשליט (Master) או כעבד (Slave)
- 7 תדרי שידור שונים הניתנים לבחירה
- קצב תקשורת מהיר – מחצית אות השעון של המיקרו-בקר
- מילת מידע של 8 סיביות
- תומך ב-4 אופני שידור
- תומך בשידור MSB תחילה או LSB תחילה.

כאמור המיקרו-בקר מאפשר שידור ב-7 קצבי תקשורת שונים. קצב התקשורת הוא חלוקה של תדר הגביש המחובר למיקרו-בקר. חלוקות אפשריות הן: 2, 4, 8, 16, 32, 64 ו-128. עבור תדר גביש של 16MHz ניתן לשדר בקצב מרבי של 8MHz ומזערי של 125kHz.

הדקי המיקרו-בקר



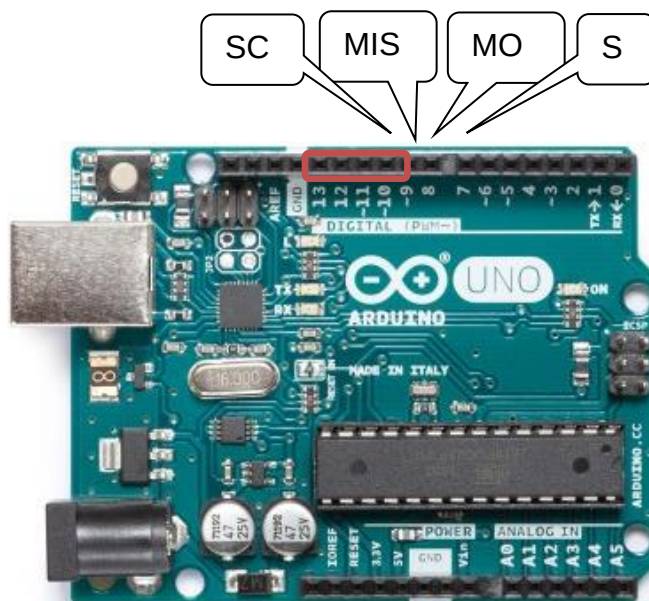
הגדרות כיוון ההדקים

הדק	כיוון בהתקן עבד	כיוון בהתקן שליט
MOSI	קלט	לפי הגדרת משתמש
MISO	לפי הגדרת משתמש	קלט
SCK	קלט	לפי הגדרת משתמש
SS	קלט	לפי הגדרת משתמש

5.4 הדקי התקשורת בלוח Arduino UNO

בלוח הפיתוח UNO מפתח PB ממופה להדקים הבאים:

מפתח/סיבית	0	1	2	3	4	5	6	7
PB	8	9	10	11	12	13	C	C
			SS	MOSI	MISO	SCK		



איור 5.4: הדקי התקשורת בלוח הפיתוח

5.5 מחלקת SPISettings

המחלקה SPISetting היא מחלקה חברה⁴ (friend) של מחלקת SPI. משמעות החברות היא שלמחלקה SPI יש גישה מלאה לכל מרכיבי המחלקה SPISetting בצורה ישירה.

המחלקה נועדה לקבוע את התצורה של מפתח התקשורת SPI. למחלקה 2 פעולות בנייה. פעולות הבנייה מופיעות בטבלה 5.1:

הפעולה	תיאור הפעולה
SPISettings()	פעולת בנייה מחדל המאתחלת את התצורה של מפתח התקשורת SPI לערכים קבועים: clock = 4000000 bitOrder = MSBFIRST dataMode = SPI_MODE0
SPISettings(unsigned int clock, byte bitOrder, byte dataMode)	הפעולה מאתחלת את התצורה של מפתח התקשורת SPI על פי הפרמטרים האלה: clock – קביעת תדר אות שעון התקשורת. כפי שצוין בסעיף הקודם ניתן לקבוע 7 תדרים בלבד. הפעולה בודקת את הערך המועבר, וקובעת את התדר לאחד משבעת התדרים האפשריים. bitOrder – קביעת סדר שידור המידע הטורי: LSBFIRST שידור סיבית משמעותית פחות תחילה. MSBFIRST

⁴ המקבילה לחברות בשפת C# היא הרשאת גישה מסוג protected.

שידור הסיבית המשמעותית ביותר תחילה. dataMode – קביעת שיטת השידור. ישנן ארבע אפשרויות: SPI_MODE0, SPI_MODE1, SPI_MODE2, SPI_MODE3	
---	--

טבלה 5.1: פעולות המחלקה

השימוש בפעולה זו מתבצע בפעולה `beginTransaction` של המחלקה SPI כפי שנראה בסעיף 5.6.

דוגמאות

- פעולת בנייה ללא פרמטרים:

```
SPISettings setting();
```

פרמטרי התקשורת יקבעו לפי ברירת המחדל.

- פעולת בנייה עם פרמטרים:

```
SPISettings setting(10000000, MSBFIRST, SPI_MODE0);
```

התדר הנקוב הוא $10\text{MHz} = 10000000\text{Hz}$, אבל עבור תדר גביש של 16MHz תדר השידור המרבי הוא $8\text{MHz} = 16\text{MHz}/2$ (לפי יחס חלוקה של 2) ולכן התדר שיקבע בפועל הוא 8MHz .

ערכו של התדר תלוי ברכיב המחובר אל מפתח התקשורת. הערך שיועבר כפרמטר לאות השעון יהיה של הרכיב המחובר. פעולת הבנייה תתאים את דרישת השעון של הרכיב לתדר האפשרי במיקרו-בקר.

5.6 מחלקת SPI

המחלקה SPI היא ספרייה הכוללת אוסף פעולות המאפשר שימוש בתקשורת. לוח הפיתוח יישמש כשליט (Master) וההתקן המחובר אליו כעבד (Slave). לא ניתן להגדיר את לוח הפיתוח כעבד.

בטבלה 5.2 מתוארות חלק מפעולות המחלקה (UML).

הפעולה	תיאור הפעולה
void begin()	הפעולה מאתחלת את התקשורת. הדקי התקשורת SCK, MOSI ו-SS מוגדרים כיציאות (OUTPUT). הדקים SCK ו-MOSI ברמה נמוכה, והדק SS ברמה גבוהה (מכיוון שפעיל ברמה נמוכה).
void end()	הפעולה מבטלת את השימוש בהדקים בתקשורת SPI. הגדרות ההדקים אינן משתנות.
void beginTransaction (SPISetting setting)	הפעולה מאתחלת את מפתח התקשורת על פי ההגדרות בפרמטר setting. setting – עצם מהמחלקה SPISetting המגדיר את 3 פרמטרי התקשורת: תדר, סדר העברת הנתונים ואופן הפעולה של התקן התקשורת. לפירוט, ראו מחלקת SPISetting.
byte endTransaction ()	הפעולה מסיימת את השימוש בקו התקשורת. בדרך כלל, נקרא לפעולה לאחר העלאת קו הבחירה של התקן העבד לרמה לוגית גבוהה, כדי

לאפשר שימוש בקו התקשורת עם התקנים אחרים.	
הפעולה משדרת וקולטת בו-זמנית. • הפעולה transfer מקבלת בית כפרמטר ומחזירה בית. • הפעולה transfer16 מקבלת נתון בגודל 16 סיביות (ב- UNO משתנה שלם int) ומחזירה נתון באורך 16 סיביות. • הפעולה מקבלת מערך של בתים ומשתנה המציין את גודלו של המערך. הפעולה משדרת את המערך כולו, ומחזירה את המידע באותו המערך.	byte transfer(byte val) unsigned int transfer16(unsigned int val16) void transfer(byte buf[], int count)

טבלה 5.2: פעולות המחלקה

5.7 דוחף תצוגה MAX7219

תיאור הרכיב

MAX7219 הוא דוחף תצוגת LED מסוג קתודה משותפת CC. הרכיב תומך בתצוגה ספרתית מרובבת של עד 8 יחידות 7 מקטעים (7-Segment), תצוגת פס גרפי (Bar-Graph) או 64 יחידות LED עצמאיות.

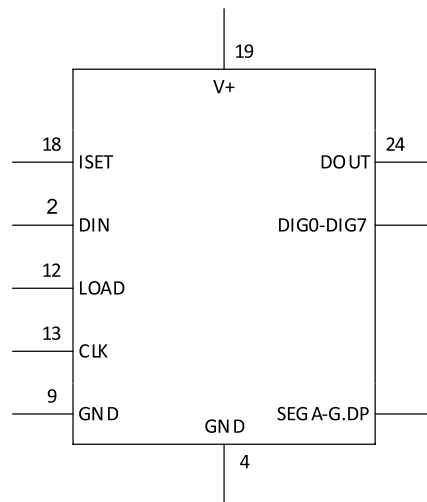
לרכיב ממשק תקשורת טורית הכולל כניסה טורית עבור הנתונים, כניסת אות שעון וכן כניסה לטעינת הערך ברכיב. בגרסה MAX7221 קיימת תמיכה מלאה בפרוטוקול SPI. כמו כן, רכיב הדק מוצא מאפשר שרשרת של מספר רכיבים בתצורת Daisy Chained.

הרכיב כולל זיכרון סטטי של 8 בתים לאחסון ערכי התצוגה, יחידת פענוח, יחידת ריבוב תצוגה, ומעגלי דחיפה ליחידות ה-LED. עוצמת הזרם נקבעת באמצעות נגד יחיד.

ניתן לגשת לכל אחת מיחידות התצוגה לשם עדכון הערך בה, וכמו כן לקבוע אילו יחידות תצוגה יופעלו. ניתן לשלוט עצמת ההארה בצורה אנלוגית ובצורה ספרתית.

תיאור הדקים

באיור 5.5 מתוארים הדקי הרכיב.



איור 5.5: הדקי הרכיב

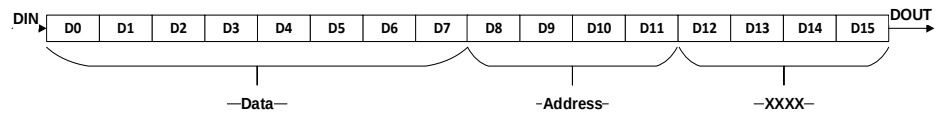
- ההדקים DIG0-DIG7 משמשים לבחירת יחידת התצוגה.
- ההדקים SEG A – G, DP משמשים להפעלת המקטעים ביחידת התצוגה.
- ההדקים DIN, LOAD ו-CLK משמשים לתקשורת הטורית.
- ההדק ISET קובע את עוצמת הזרם ליחידות התצוגה.
- ההדק DO UT משמש כיציאה טורית לשרשור רכיבים.
- ההדקים VCC ו-GND משמשים כהדקי אספקת המתח.

שידור הנתונים

הנתונים משודרים כאשר הסיבית המשמעות ביותר (MSB) תחילה. מספר הסיביות המשודרות הוא 16. להלן החלוקה לסוגים:

- סיביות D0-D7 סיביות הנתונים
- סיביות D8-D11 סיביות הכתובת של הרכיב
- סיביות D12-D15 אינן בשימוש

איור 5.6 מתאר את מבנה המידע המשודר לרכיב.



איור 5.6: מבנה מילת המידע המשודרת

המידע באוגר ההזזה ברכיב ננעל בזמן עליית דופק השעון (Rising Edge) ללא קשר למצבו של הדק המבוא LOAD (ברכיב MAX7221 ההדק CS חייב להיות במצב נמוך על מנת שההזזה תבצע). לאחר 16 דפקים המידע באוגר ההזזה נטען לאוגרים ברכיב עם עליית האות בהדק ה-LOAD (CS ברכיב MAX7221).

אופן הפעולה של השידור מתאים ל-MODE 0: CPOL = 0, CPHA = 0.

מפת הכתובות

HEX	כתובת					אוגר
	D12-15	D11	D10	D9	D8	
0xX0	X	0	0	0	0	NOP
0xX1	X	0	0	0	1	DIG 0
0xX2	X	0	0	1	0	DIG 1
0xX3	X	0	0	1	1	DIG 2
0xX4	X	0	1	0	0	DIG 3
0xX5	X	0	1	0	1	DIG 4
0xX6	X	0	1	1	0	DIG 5
0xX7	X	0	1	1	1	DIG 6
0xX8	X	1	0	0	0	DIG 7
0xX9	X	1	0	0	1	Decode Mode
0xXA	X	1	0	1	0	Intensity
0xXB	X	1	0	1	1	Scan Limit
0xXC	X	1	1	0	0	Shutdown
0xFF	X	1	1	1	1	Display Test

X – אין חשיבות לערך

טבלה 5.3: מפת הכתובות

מצבי פעולה

מצב כבוי – Shutdown Mode

במצב זה פעולת ריבוב התצוגה מופסקת, מעגלי דחיפת הזרם ליחידות התצוגה מופסקים, ועקב כך כל יחידות התצוגה מוחשקות. המידע באוגרי התצוגה נשאר ללא שינוי. מצב זה יכול לשמש למטרת חיסכון או להציג הבהוב המתריע על מצב "אזעקה".

אוגר בכתובת 0xCX								מצב
D7	D6	D5	D4	D3	D2	D1	D0	
X	X	X	X	X	X	X	0	כבוי
X	X	X	X	X	X	X	1	פועל

X – אין חשיבות לערך

טבלה 5.4: אוגר Shutdown

מצב פענוח – Decode Mode

באמצעות אוגר זה ניתן לקבוע אם יתבצע פענוח לפי קוד BCD לספרה ביחידת התצוגה. כל סיבית באוגר שולטת בספרת תצוגה. הצבת 0 בסיבית מציינת שלא יתבצע פענוח, והצבת 1 בסיבית מציינת שיתבצע פענוח.

אוגר בכתובת 0x9X								מצב
D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	0	0	0	אין פענוח
0	0	0	0	0	0	0	1	פענוח D0 בלבד
0	0	0	0	0	0	1	1	פענוח D0 ו-D1 בלבד
1	1	1	1	1	1	1	1	פענוח כל יחידות התצוגה

טבלה 5.5: אוגר Decode Mode

קוד BCD

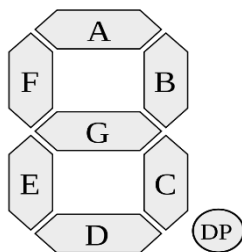
קוד BCD הוא קוד בן 4 סיביות המייצגות ערך עשרוני באופן בינארי. כיוון שכדי לרשום מספר מ-0 עד 9 חייבים 4 סיביות, יש משמעות לקוד רק ל-10 ערכים מ-0000B עד 1001B. יתר המצבים חסרי משמעות. ביחידת הפענוח ברכיב מנצלים את המצבים האחרים כדי להציג סימנים נוספים.

בשורה הראשונה בטבלה 5.6 מוצג קוד BCD מורחב. הקוד מוצג לפי בסיס הקסה-דצימאלי (16) ומתחתיו הערך שיוצג ביחידת התצוגה. שימו לב, עבור הקוד 0FH התצוגה תוחשך.

קוד	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
תצוגה	0	1	2	3	4	5	6	7	8	9	E	H	L	P	-	

טבלה 5.6: קוד BCD מורחב

כאשר בוחרים להשתמש בפעולת הפענוח, מעבירים לאוגר התצוגה המתאימה ערך מספרי לפי הקוד המופיע בטבלה 5.6. כאשר בוחרים שלא להשתמש בפעולת הפענוח, מעבירים לאוגר התצוגה המתאימה ערך מספרי לפי הטבלה 5.7



D7	D6	D5	D4	D3	D2	D1	D0
DP	A	B	C	D	E	F	G

טבלה 5.7: מיפוי מקטעים באוגר הנתונים ליחידת התצוגה

קביעת עוצמה – Intensity

עוצמת ההארה ברכיב נקבעת על ידי נגד חיצוני המחובר בין הדק ISET לבין מתח ההפעלה VCC. הזרם המסופק ליחידת התצוגה הוא פי 100 מערך הזרם שקובע הנגד. ניתן לחבר נגד קבוע או נגד משתנה לצורך ויסות עוצמת ההארה. ערכו המינימלי של הנגד צריך להיות $9.53k\Omega$. ערך זה קובע זרם של 40mA ליחידת התצוגה. עצמת ההארה ניתנת לקביעה גם באמצעות אוגר עוצמת ההארה.

בקרת ההארה מתבצעת על ידי הצבת ערך מספרי בין 0 ל-15 באוגר עוצמת ההארה. כאשר ברכיב 7219, הערך 0 מייצג עצמה מינימלית של 1/32, 1 מייצג עוצמה של 3/32, והערך 15 המייצג עוצמה של 31/32.

אוגר בכתובת 0xA0								מצב
D7	D6	D5	D4	D3	D2	D1	D0	
X	X	X	X	0	0	0	0	1/32
X	X	X	X	0	0	0	1	3/32
X	X	X	X	1	1	1	1	31/32

X – אין חשיבות לערך

טבלה 5.8: קביעת עוצמת ההארה באמצעות אוגר העצמה

מגבלת סריקה – Scan Limit

אוגר מגבלת הסריקה קובע אילו יחידות תצוגה מ-1 עד 8 יוצגו באופן מרובב. ערכים מספריים מ-0 עד 7 קובעים את מספר יחידות התצוגה שיופעלו. 0 מייצג יחידת תצוגה 1, 1 מייצג יחידות תצוגה 1 ו-2, 2 מייצג יחידות תצוגה 1, 2 ו-3 עד לערך 7 המייצג הפעלה של כל יחידות התצוגה.

טבלה 5.9 מציגה את אופן הגדרת יחידות התצוגה שיופעלו.

אוגר בכתובת 0xB0								מצב
D7	D6	D5	D4	D3	D2	D1	D0	
X	X	X	X	X	0	0	0	תצוגה 1
X	X	X	X	X	0	0	1	תצוגות 1 ו-2
X	X	X	X	X	0	1	0	תצוגות 1, 2 ו-3
X	X	X	X	X	1	1	1	כל התצוגות מ-1 עד 8

X – אין חשיבות לערך טבלה 5.9: אוגר מגבלת הסריקה

שאלה 5.1

חפשו באינטרנט דפי נתונים של הרכיב 7219, וכתבו מה מטרת אוגר NOP המוגדר בכתובת 0?

שאלה 5.2

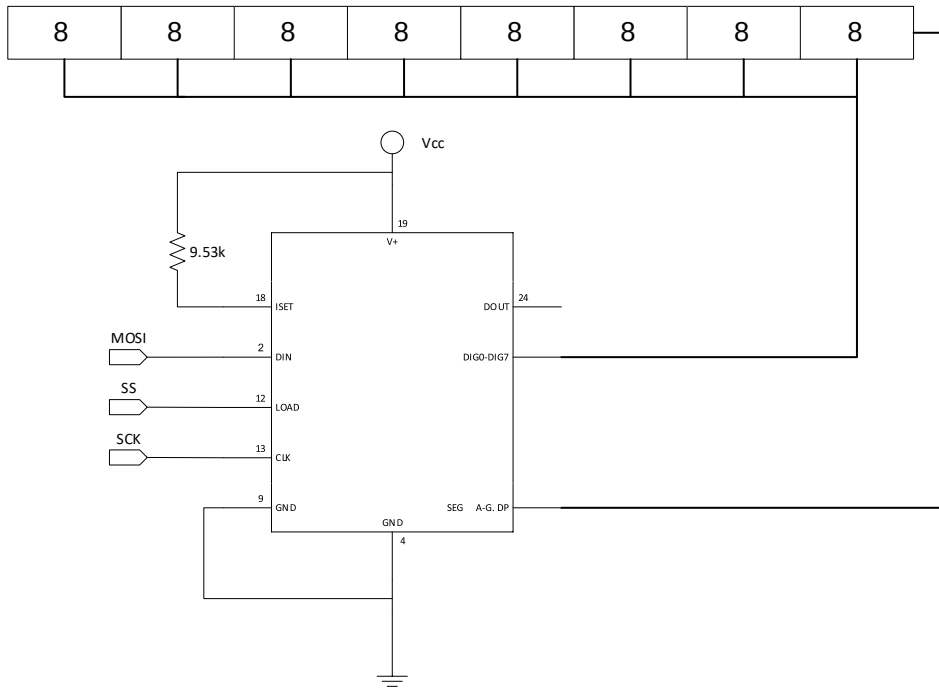
חפשו באינטרנט דפי נתונים של הרכיב 7219, וכתבו מה מטרת אוגר Display Test המוגדר בכתובת 0xXF?

שאלה 5.3

חפשו באינטרנט דפי נתונים של מארז של תצוגת 7 מקטעים מסוג קתודה משותפת (CC). למארז 4 יחידות תצוגה.

א. זהו את הדקי הרכיב: A, B, C, D, E, F, G, DP, COM1, COM2, COM3, ו-COM4.
 ב. על פי דפי המפרט קבעו לאיזה הדקים ברכיב 7219 יחוברו ההדקים המשותפים: COM1, COM2, COM3, ו-COM4. הסבירו.

איור 5.7 מתאר את אופן החיבור של הרכיב ליחידת התצוגה. כפי שניתן לראות באיור הדקים SEGA-G של הרכיב 7219 מחוברים אל הדקי ההפעלה של מקטעי ה-LED בתצוגה המרובבת, והדקים DIG1-DIG7 ברכיב 7219 מחוברים אל הדק ה-COM של כל אחת מתצוגת 7-Segment. הנגד בתרשים החשמלי קובע את עוצמת הארה של התצוגה.



איור 5.7: תרשים חיבורים חשמלי של הרכיב

דוגמה 5.1 – הצגת מספר דו ספרתי עם פענוח

בדוגמה זו ניצור תקשורת בין שני לוחות הפיתוח לבין הרכיב ונציג את המספרים מ-0 עד 99 באופן מחזורי תוך השתייה של 0.1 שנייה בין מספר אחד לאחר. הדקי התקשורת של הרכיב יחוברו להדקים המתאימים בלוח הפיתוח על פי איור 5.7 ואיור 5.4.

בתוכנית נעשה שימוש בשתי פעולות עזר:

- `Init7219` – הפעולה מאתחלת את הרכיב כך שיופעלו רק שתי יחידות תצוגה על ידי הגבלת הסריקה לשתי היחידות בלבד. שתי יחידות התצוגה יפוענחו לפי קוד BCD, בעוצמת הארה מרבית. לבסוף תופעל יחידת התצוגה.
- `writeNumber` – הפעולה מקבלת מספר שלם בין 0 לבין 99, כולל. הפעולה מפרקת את המספר לספרת היחידות ולספרת העשרות ומעבירה אותן לרכיב: היחידות ליחידת תצוגה 1, והעשרות ליחידת התצוגה 2.

הפעולה `setup` מזמנת את הפעולה `Init7219` ואילו הפעולה `loop` מזמנת את הפעולה `writeNumber` עם ערך המשתנה בין 0 ל-99, וחוזר חלילה.

שילוב ספריית תמיכה בתקשורת //

```
#include <SPI.h>
```

הגדרת משתנים קבועים המייצגים את הכתובות הפנימיות ברכיב //

7219

```
const unsigned int DIG0 = 0x0100; // כתובת תצוגה 1
```

```
const unsigned int DIG1 = 0x0200; // כתובת תצוגה 2
```

```
const unsigned int DECODE_REG = 0x0900; // כתובת אוגר הפענוח
```

```
const unsigned int INTENSITY_REG = 0x0A00; // כתובת אוגר עצמת
```

ההארה

```
const unsigned int SCAN_LIMIT_REG = 0x0B00; // כתבת אוגר מגבלת
```

הסריקה

```
const unsigned int SHUTDOWN_REG = 0x0C00; // כתובת אוגר הכיבוי
```

הגדרת פרמטרי התקשורת: תדר, סדר שידור סיביות ואופן פעולה //

```
SPISettings setting(10000000, MSBFIRST, SPI_MODE0);
```

```
byte number; // משתנה מונה מספרים
```

פעולת אתחול //

```
void setup() {
```

```
  Serial.begin(9600);
```

```
  SPI.begin(); // אתחול תקשורת SPI
```

```
  number = 0; // קביעת ערך התחלתי למשתנה
```

```
  Init7219(); // קריאה לפעולת אתחול הרכיב
```

```
}
```

```

// פעולת הלולאה
void loop() {
    writeNumber(number);           // קריאה לפעולה המשדרת מספר לרכיב 7219
    Serial.println(number);
    number++;                       // קידום המספר
    number = number % 100;          // הגבלת ערך המספר מ-0 עד 99
    delay(100);                     // השהייה
}

// הפעולה מאתחלת את הרכיב 7219
void Init7219(){
    SPI.beginTransaction(setting); // קריאה לפעולה המאתחלת את התקשורת
    digitalWrite(SS, LOW);         // בחירת הרכיב – פעיל בנמוך
    SPI.transfer16(DECODE_REG | 0x0003); // הפעלת פענוח לשתי יחידות תצוגה
    digitalWrite(SS, HIGH);        // נעילת המידע ברכיב

    digitalWrite(SS, LOW);         // בחירת הרכיב – פעיל בנמוך
    SPI.transfer16(INTENSITY_REG | 0x000F); // קביעת עצמת הראה מרבית
    digitalWrite(SS, HIGH);        // נעילת המידע ברכיב

    digitalWrite(SS, LOW);         // בחירת הרכיב – פעיל בנמוך
    SPI.transfer16(SCAN_LIMIT_REG | 0x0001); // הגבלת סריקה לשתי יחידות
    digitalWrite(SS, HIGH);        // נעילת המידע ברכיב

```

```

digitalWrite(SS, LOW);                // פעיל בנמוך – בחירת הרכיב
SPI.transfer16(SHUTDOWN_REG | 0x0001); // הפעלת הרכיב
digitalWrite(SS, HIGH);                // נעילת המידע ברכיב
}

// הפעולה מקבלת מספר בין 0 ל-99 ומעבירה את הערך שלו לרכיב
התצוגה

void writeNumber(byte num){
    digitalWrite(SS, LOW);                // פעיל בנמוך – בחירת הרכיב
    SPI.transfer16(DIG0 | (num % 10));    // עדכון יחידת תצוגה 1 בערך היחידות
    digitalWrite(SS, HIGH);                // נעילת המידע ברכיב
    digitalWrite(SS, LOW);                // פעיל בנמוך – בחירת הרכיב
    SPI.transfer16(DIG1 | (num / 10));    // עדכון יחידת תצוגה 2 בערך
    העשרות
    digitalWrite(SS, HIGH);                // נעילת המידע ברכיב
}

```

שאלה 5.4

תלמיד החליט לשכתב את הקוד של הפעולה `writeNumber`.

```

// הפעולה מקבלת מספר בין 0 ל-99 ומעבירה את הערך שלו לרכיב
התצוגה

void writeNumber(byte num){
    digitalWrite(SS, LOW);
    SPI.transfer16(DIG0 | (num & 0x0F));
    digitalWrite(SS, HIGH);
}

```



```
digitalWrite(SS, LOW);

SPI.transfer16(DIG1 | (num & 0xF0));

digitalWrite(SS, HIGH);

}
```

- א. האם התיקון שהציע התלמיד תקין?
- ב. מה יוצג בתצוגה?
- ג. הציעו תיקון, אם נדרש תיקון לפעולה.

שאלה 5.5

תלמיד אחר הציע קוד אחר לפעולה `writeNumber`. לטענתו הקוד יעיל יותר.

הפעולה מקבלת מספר בין 0 ל-99 ומעבירה את הערך שלו לרכיב // התצוגה

```
void writeNumber(byte num){

    digitalWrite(SS, LOW);

    SPI.transfer16(DIG0 | (num % 10));

    digitalWrite(SS, HIGH);

    if (num % 10 == 0) {

        digitalWrite(SS, LOW);

        SPI.transfer16(DIG1 | (num & 0xF0));

        digitalWrite(SS, HIGH);

    }

}
```

- א. האם התיקון שהציע התלמיד תקין?
- ב. מה יוצג בתצוגה?
- ג. הציעו תיקון אם נדרש תיקון לפעולה.

שאלה 5.6

כתבו מחדש את התוכנית מדוגמה 5.1 כך שיוצגו המספרים מ-0 עד 999.

שאלה 5.7

כתבו מחדש את התוכנית מדוגמה 5.1 כך שאם המספר הוא חד-ספרתי מ-0 עד 9 תוצג ספרה אחת בלבד ואם המספר הוא דו-ספרתי, הוא יוצג בשתי יחידות התצוגה.
רמז: היעזרו בטבלה 5.6.

שאלה 5.8

כתבו תוכנית המציגה תאריך בתבנית הבאה: dd-mm-yy. הניחו כי התצוגה מורכבת מ-8 יחידות Segment-7.
לדוגמה 11-07-18.
רמז: היעזרו בטבלאות 5.6 ו-5.7.

שאלה 5.9

כתבו תוכנית המציגה את השעה בתבנית HHMM. למשל, 1245. עוצמת התאורה של השעון תהיה תלויה בעוצמת האור של הסביבה. כך שאם עוצמת האור של הסביבה גבוהה, אז עוצמת האור של השעון תהיה גבוהה ולהיפך, אם עוצמת האור של הסביבה נמוכה עוצמת האור של השעון תהיה נמוכה.

- א. שרטטו מעגל חשמלי הכולל חיישן אור המשמש לזיהוי עוצמת אור הסביבה. קבעו לאיזה הדק בלוח הפיתוח החיישן מחובר. היעזרו בפרק 3 של מערכות משובצות מחשב.
- ב. כתבו תוכנית עבור השעון.

5.8 מסך גרפי 12864B

המסך הגרפי מבוסס על הבקר ST7920 המציג טקסט אלפאנומרי (א-ב לועזי וספרות), גופנים סיניים ותווים המוגדרים על ידי המשתמש.

הרכיב כולל זיכרון מסוג ROM התומך בגופנים סיניים בגודל 16x16 נקודות וכן 126 תווים בגודל 16x8 נקודות. הרכיב תומך גם בתצוגה גרפית בשטח של 64x256 נקודות באמצעות זיכרון גרפי GDRAM. ניתן להציג בו זמנית גם תווים וגם גרפיקה. יש ברכיב תמיכה בארבעה מרחבי זיכרון CGRAM ליצירת תווים בגודל 16x16.

טבלה 5.10 מציגה את הדקי רכיב התצוגה.

שם	מספר	I/O	חיבור	תיאור
RST	17	I		כניסת איפוס
PSB	15	I		בחירת ממשק 0 – טורי 1 – 4/8 מקבילי
RS(CS*)	4	I	MPU	פעולה מקבילית: בחירת אוגר 0 – אוגר ההוראות 1 – אוגר הנתונים פעולה טורית: קו בחירה 1 – רכיב מאופשר 0 – רכיב לא מאופשר
RW(SID*)	5	I	MPU	פעולה מקבילית: בקרת כתיבה/קריאה 0 – כתיבה 1 – קריאה פעולה טורית: כניסת נתונים טורית
E(SCLK*)	6	I	MPU	פעולה מקבילית: קו אפשר דרבון פעולה טורית: שעון טורי
D4 ÷ D7	11 ÷ 14	I/O	MPU	4 הסיביות הגבוהות לממשק 8 סיביות 4 הסיביות לממשק 4 סיביות
D0 ÷ D3	7 ÷ 10	I/O	MPU	4 הסיביות הנמוכות לממשק 8 סיביות
VDD	2	I	Power	מתח הפעלה
VSS	1	I	Power	אדמה
VOUT	18	O		יציאת מכפל המתח LCD
BLA	19	I		תאורת LED אחורית
BLK	20	I		תאורת LED אחורית
V0	3	I		בקרת ניגודיות תצוגה

* שמות ההדקים כאשר משמשים בתקשורת טורית

טבלה 5.10: הדקי רכיב התצוגה

תיאור הדקים

רכיב התצוגה מותאם לעבודה במתח אספקה של 5V. מתח האספקה VDD יכול להיות בתחום בין 4.5V לבין 5V.

ההדק V0 משמש לבקרת הניגודיות של התצוגה. הדק זה מבוקר באמצעות מתח המתקבל מפוטנציומטר כדי להתאים את התצוגה לפי דרישת המשתמש.

הדקים BLK ו-BLA הם הדקי אספקת מתח לתאורה אחורית מבוססת LED.

לרכיב שני ממשקי חיבור: מקבילי או טורי. הבחירה בין שתי האפשרויות מתבצעת באמצעות הדק PSB המתואר בטבלה 5.10.

ממשק מקבילי $PSB = 1$

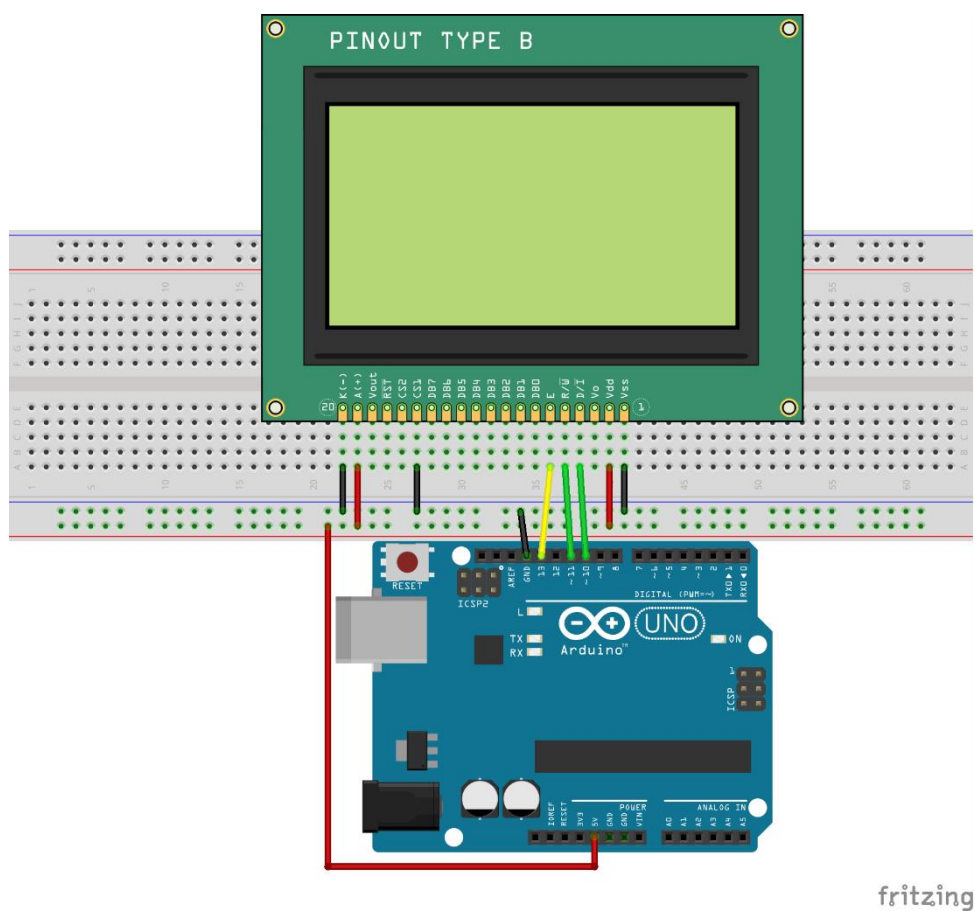
בחיבור המקבילי ניתן לבחור בין ממשק של 4 או 8 סיביות נתונים. הבחירה בין שני הממשקים מתבצעת בזמן אתחול רכיב התצוגה. כאשר בוחרים בממשק מקבילי של 8 סיביות נשתמש בהדקי נתונים D0 עד D7. כאשר בוחרים בממשק של 4 סיביות נשתמש בהדקי נתונים D4 עד D7. בממשק המקבילי נשתמש בהדקי בקרה RS, RW ו-E. בפרק זה לא נדון בממשק המקבילי.

ממשק טורי $PSB = 0$

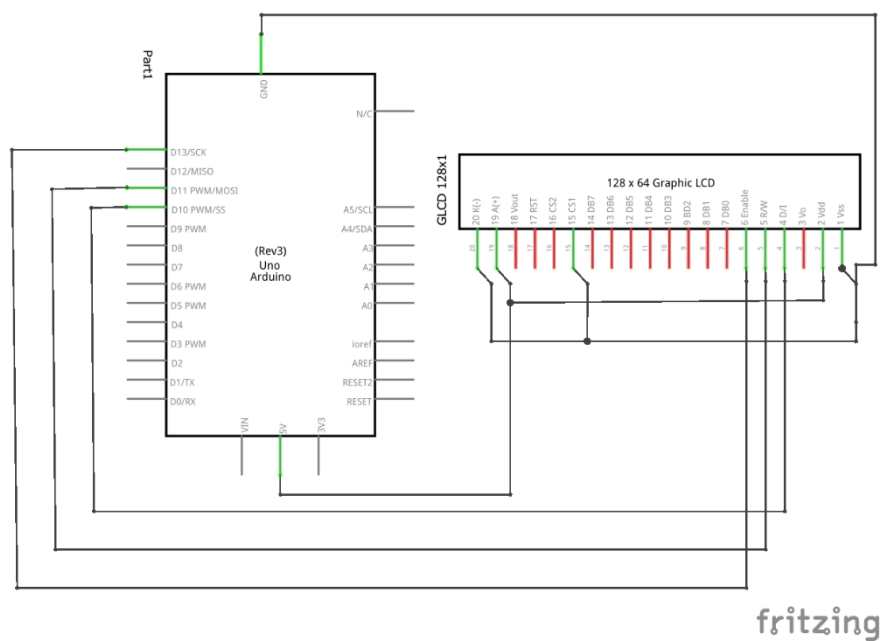
בממשק הטורי לא נעשה שימוש בהדקי הנתונים D0 עד D7. הדקי בקרה לממשק הטורי הם CS, SID ו-SCLK. הדקים אלה תואמים לחיבור בממשק SPI.

תרשימי חיבורים

באיורים להלן מוצג תרשימי החיבור של לוח התצוגה ללוח הפיתוח.



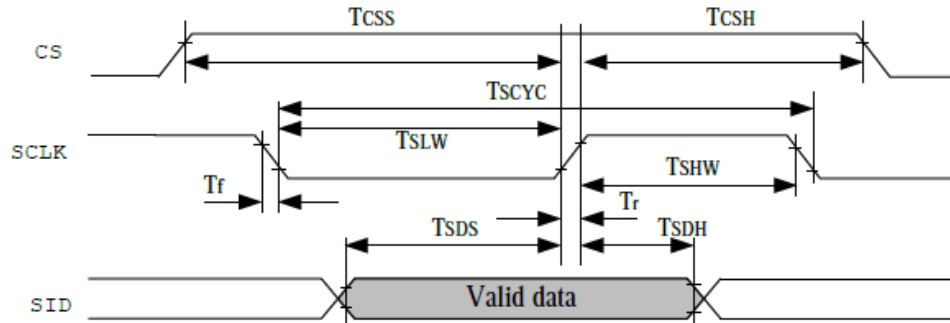
איור 5.8: חיבור רכיב התצוגה ללוח הפתוח



איור 5.9: תרשים חשמלי של חיבור לוח הפיתוח לרכיב התצוגה

תזמון תקשורת טורית

איור 5.10 מתאר את ממשק התזמון הטורי לסיבית אחת.



איור 5.10: תרשים זמנים ממשק טורי

מהתרשים רואים כי בשלב ראשון מעבירים את הדק הבחירה CS לערך גבוה (פעיל). לאחר מכן מעבירים את הדק אות השעון הטורי לערך נמוך. בשלב זה מוציאים להדק הטורי את המידע. המידע הטורי נדגם בזמן עליית האות בהדק השעון.

על פי תבנית זו קוטביות אות השעון היא מסוג $CPOL = 1$. ההתחלה היא בירידת שעון, וסיום ההעברה הוא בעליית שעון. על פי התבנית $CPHA = 1$, כלומר הצד המשדר מוציא את המידע בזמן הופעת הדופק. על פי התבנית אופן הפעולה הוא 3.

יש לשים לב כי בניגוד לתקן SPI קו ה-CS פועל ברמה גבוהה במקום ברמה נמוכה.

בממשק טורי אפשר רק לכתוב ואין אפשרות לקרוא.

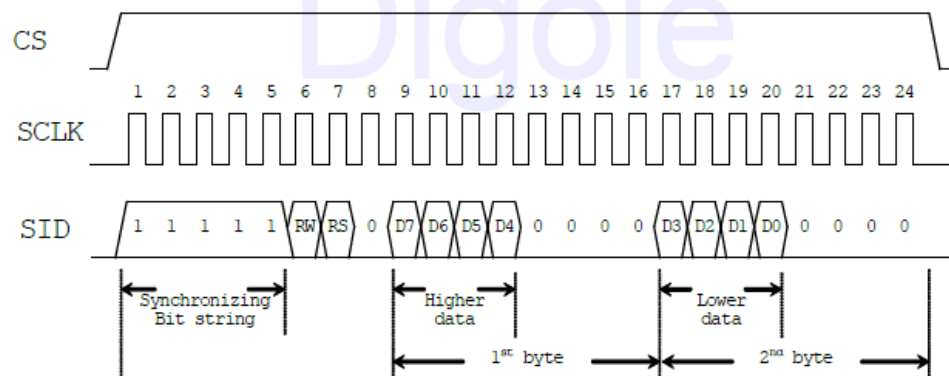
כאשר משתמשים ברכיב תצוגה יחיד, ניתן לחבר את קו הבחירה באופן קבוע ברמה גבוהה $CS=1$.

שידור נתונים

איור 5.11 מתאר שידור מידע אל רכיב התצוגה. שידור נתונים מתחיל עם העברת קו הבחירה למצב פעיל $CS=1$.

הבית המשודר ראשון מתחיל ב-5 סיביות של '1' המשמשות לסינכרון. שתי הסיביות שאחריהן מציינות קריאה או כתיבה (RW) ובחירת אוגר (RS). הסיבית האחרונה היא 0.

לאחר הבית הראשון המהווה סינכרון, משודרים זוגות בתים המציינים את מילת המידע בגודל 8 סיביות. הבית הראשון מבין הזוגות כולל את 4 הסיביות הגבוהות ו-4 אפסים והבית השני את 4 הסיביות הנמוכות ו-4 אפסים.



איור 5.11: שידור מידע לרכיב התצוגה

אתחול ערוץ התקשורת

אתחול ערוץ התקשורת מתבצע באמצעות סדרת ההוראות המוצגות להלן:

1. אתחול ערוץ התקשורת וקביעת רמת אות בחירת העבד $SS=0$

```
SPI.begin();
digitalWrite(SS, LOW);
```

2. יצירת עצם setting מסוג SPISettings

```
SPISettings setting(8000000L, MSBFIRST, SPI_MODE3);
```

פרמטרי התקשורת:

```
clock = 8000000Hz
bitOrder = MSBFIRST
```



```
dataMode = SPI_MODE3
```

3. הפעלת תהליך העברת המידע לפי פרמטרי התקשורת

```
SPI.beginTransaction(setting);
```

לאחר אתחול ערוץ התקשורת אפשר להעביר מידע אל רכיב התצוגה. על פי איור 5.11, כל העברה כוללת לפחות 3 בטים רצופים.

פעולת LcdWrite

הפעולה LcdWrite מעבירה ערך אחד בגודל בית אל אוגר ההוראות (Command) או אל אוגר הנתונים (Data). משמעות ההעברה אל אוגר ההוראות או הנתונים תובהר בהמשך. הוראות הפעולה מיישמות את הרצף המוצג באיור 5.11.

```
const bool LCD_COMMAND = true;

const bool LCD_DATA = false;

/*
 *                               // הפעולה מעבירה בית אחד אל אוגר ההוראות או אוגר
הנתונים
 * INPUT:
 *   command - הבית - ערך יופנה ערך הבית -
 *   LCD_COMMAND העברה לאוגר ההוראות
 *   LCD_DATA העברה לאוגר הנתונים
 *   value - הערך המועבר לאוגר -
 */

void LcdWrite(bool command, byte value){
    digitalWrite(SS, HIGH);      // set SS
```

```
// sync = B111110RS0 : RS=1 (נתון) or RS=0 (הוראה)

byte sync = command ? B11111000 : B11111010; // הסנכרון בשלוב בחירת
האוגר

SPI.transfer(sync); // שליחת הבית הראשון

SPI.transfer(value & 0xF0); // שליחת הניבל5 הגבוה בבית

SPI.transfer(value << 4); // שליחת הניבל הנמוך בבית

digitalWrite(SS, LOW); // clear SS

delayMicroseconds(100); // המתנה לסיום ביצוע פעול הכתיבה

בתצוגה
}
```

זמן ביצוע הפעולה ברכיב התצוגה, לפי נתוני היצרן, הוא 72 מיקרו שניות. כדי לאפשר לרכיב להשלים את פעולתו נקבעה השהייה של 100 מיקרו שניות הגדולה מעט מהערך הנדרש.

הוראות הגדרה (commands) בסיסיות

בהוראות הבאות מתבצעת כתיבה (RW=0) לאוגר ההוראות (RS=0).

Display Clear

ההוראה מוחקת את התוכן המופיע על מסך התצוגה.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

קוד ההוראה $0x01^{6}$ (DB7–DB0).

⁵ ניבל (nibble) – מתאר יחידת מידע בת 4 סיביות

⁶ הערך 0x01 מציין ערך מספרי בבסיס 16 (Hexa-Decimal).

ההוראה תציב בכל תאי הזיכרון במרחב ה-DDRAM את הערך 0x20 (רווח), לאחר מכן תאתחל את מונה הכתובת ל-0 (AC=0), ותקבע את כיוון הסמן (cursor) לתנועה בכיוון ימין. ההוראה תגרום למונה הכתובות לגדול ב-1 בכל פעולת כתיבה לזיכרון או קריאה מהזיכרון.

Return Home

ההוראה תמקם את הסמן בתחילת השורה הראשונה.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	X

X – אין חשיבות לערך

קוד ההוראה 0x02 או 0x03 (DB7–DB0).

ההוראה תקבע את מונה הכתובת ל-0 (AC=0). הסמן ימוקם בתחילת השורה הראשונה. תוכן זיכרון DDRAM לא משתנה.

Entry Mode Set

ההוראה קובעת את האופן שפעולות כתיבה או קריאה משפיעות על מונה הכתובות לאחר הפנייה למרחב זיכרון DDRAM.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

קודי ההוראה 0x04 עד 0x07 (DB7–DB0).

הערך בזמן האתחול הוא: 0x06.

I/D: בקרת מונה הכתובות

1 – מונה הכתובות גדל ב-1

0 – מונה הכתובות קטן ב-1

S: בקרת תזוזת התצוגה

תיאור	I/D	S
תוכן התצוגה נע מקום אחד שמאלה בכל כתיבה.	H	H
תוכן התצוגה נע מקום אחד ימינה בכל כתיבה.	L	H
תוכן התצוגה לא נע.	X	L

X – אין חשיבות לערך

כאשר S=H מתקבל אפקט של תנועת טקסט כאשר הסמן נשאר במקומו.

Display Control

ההוראה קובעת אם התצוגה פועלת או כבויה ואת מאפייני הסמן. הפעולה לא משפיעה על תוכן מרחב זיכרון ה-DDRAM.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	1	D	C	B

קודי ההוראה 0x08 עד 0x0F (DB7–DB0).

הערך בזמן האתחול הוא: 0x08.

D: בקרת תצוגה

1 – התצוגה פעילה

0 – התצוגה כבויה.

C: בקרת הצגת סמן

1 – הסמן מוצג

0 – הסמן לא מוצג

B: בקרת הבהוב הסמן

1 – הסמן מהבהב. התו במיקום הסמן מהבהב.

0 – הסמן לא מהבהב.

Cursor/Display Shift Control

ההוראה קובעת את הכיוון שהסמן ינוע בו או את כיוון התזוזה של התצוגה. תוכן זיכרון DDRAM אינו משתנה.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	X	X

X – אין חשיבות לערך

קודי ההוראה 0x10 עד 0x1F (DB7–DB0).

הערך בזמן האתחול לא מוגדר.

S/C: בחירת מאפיין לשליטה

כאשר הסיבית היא H אפשר לשלוט בהזזה (S – shift), וכאשר הסיבית היא L אפשר לשלוט בסמן (C – cursor).

ערוך AC	תיאור	S/C	R/L
$AC = AC - 1$	הסמן נע מקום אחד שמאלה	L	L
$AC = AC + 1$	הסמן נע מקום אחד ימינה	L	H
ללא שינוי	התצוגה זזה מקום אחד שמאלה, הסמן עוקב אחרי ההזזה	H	L
ללא שינוי	התצוגה זזה מקום אחד ימינה, הסמן עוקב אחרי ההזזה	H	H

Function Set

ההוראה קובעת את אופן פעולת ממשק הנתונים המקבילי ואת השליטה בפנייה לסדרת ההוראות המורחבות.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	X	RE	X	X

X – אין חשיבות לערך

קודי ההוראה 0x20 עד 0x3F (DB7–DB0).

הערך בזמן האתחול הוא: 0x30.

DL: בקרת ממשק 4/8 סיביות

1 – ממשק 8 סיביות

0 – ממשק 4 סיביות

RE: בקרת סדרת הוראות מורחבות

1 – הפעלת סדרת הוראות מורחבת

0 – הפעלת סדרת הוראות בסיסיות (הערך בזמן ההפעלה הראשונית)

אי אפשר לשנות בו-זמנית את DL ו-RE. יש לקבוע קודם את DL ואחר כך את RE בשתי פניות שונות של ההוראה.

מכיוון שבזמן ההפעלה הראשונית של יחידת התצוגה ערכה של הסיבית המבקרת את הפעלת סדרת הוראות ההרחבה היא אפס ($RE=0$), יופעלו רק ההוראות הבסיסיות. באמצעות ההוראה *Function Set* יתאפשר שימוש בהוראות ההרחבה ($RE=1$). מרגע שנבחר להשתמש בהוראות תינתן להן משמעות חדשה. בסעיף הוראות הרחבה להלן נציין את המשמעות החדשה של ההוראות.

Set CGRAM Address

ההוראה קובעת כתובת במרחב הזיכרון של התווים (CGRAM) לתוך מונה הכתובות AC.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

הכתובות האפשריות נקבעות באמצעות הערכים המוצבים בסיביות AC5–AC0 כלומר בתחום הכתובות 0x00 עד 0x3F. ניתן לפנות ל- 64 כתובות שונות במרחב זיכרון התווים.

קודי ההוראה 0x40 עד 0x7F (DB7–DB0).

Set DDRAM Address

ההוראה קובעת כתובת במרחב הזיכרון של הנתונים (DDRAM) לתוך מונה הכתובות AC.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0

הכתובות האפשריות נקבעות באמצעות הערכים המוצבים בסיביות AC6–AC0 כלומר בתחום הכתובות 0x00 עד 0x7F. ניתן לפנות ל-128 כתובות שונות במרחב זיכרון הנתונים.

קודי ההוראה 0x80 עד 0x7F (DB7–DB0).

הוראת מצב (status) בסיסית

בהוראה להלן מתבצעת קריאה (RW=1) של אוגר ההוראות (RS=0).

Read Busy Flag

ההוראה מספקת מידע המציין את מצב רכיב התצוגה: עסוק או פנוי, ואת ערך מונה הכתובת.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

BF: דגל מצב

1 – התצוגה במצב עסוק. לא ניתן לשלוח מידע נוסף.

0 – התצוגה פנויה.

הכתובות המתקבלות מהערכים המוצבים בסיביות AC6–AC0 הן בתחום 0x00 עד 0x7F.

באופן פעולה של תקשורת טורית לא ניתן לקרוא מרכיב התצוגה.

הוראת נתון (data) בסיסית

בהוראות להלן מתבצעת פנייה אל אוגר הנתונים (RS=1).

Write Data to RAM

ההוראה כותבת מידע (RW=0) ל-RAM הפנימי ומעדכנת את מונה הכתובות (AC) על פי

הוראת הגדרה Entry Mode Set (קידום ב-1 או הפחתה של 1).

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D7	D6	D5	D4	D3	D2	D1	D0

Read RAM Data

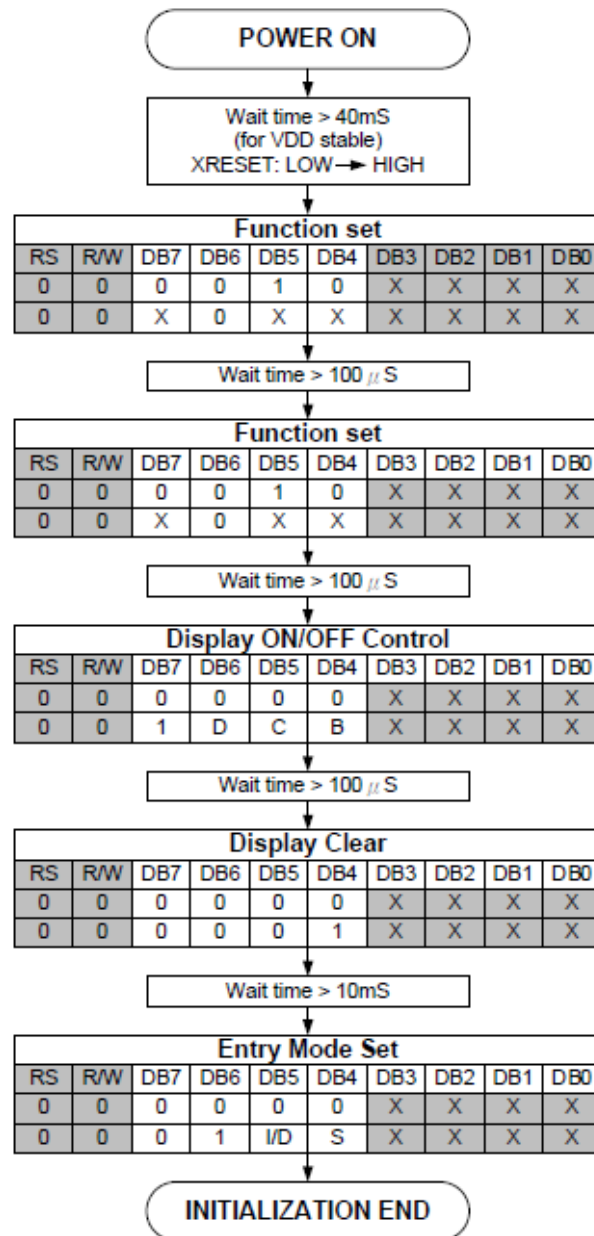
ההוראה קוראת (RW=1) מידע מה-RAM הפנימי ומעדכנת את מונה הכתובות (AC) על פי הוראת הגדרה Entry Mode Set (קידום ב-1 או הפחתה של 1).

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D7	D6	D5	D4	D3	D2	D1	D0

באופן פעולה של תקשורת טורית לא ניתן לקרוא מרכיב התצוגה.

אתחול רכיב התצוגה

בתרשים הזרימה שבאיור 5.12 מתואר תהליך האתחול של רכיב התצוגה מרגע הפעלתו או מרגע איפוס הרכיב. כאשר מחברים את יחידת התצוגה בתקשורת טורית מסוג SPI העברת המידע מתבצעת בתבנית של חיבור של 4 סיביות נתונים.



איור 5.12: תרשים זרימה של אתחול הרכיב

קבועי תוכנית

```

const byte SYNCFLAG = 0x0F8;           // סיביות 11111 המשמשות לסינכרון 5
const byte RS = 0x02;                   // ערך הסיבית הקובעת פנייה לאוגר
הנתונים
const byte DISPLAY_CLEAR = 0x01; // קוד ההוראה לאיפוס התצוגה
  
```

```

const byte ENTRY_MODE_SET = 0x04;      // קוד ההוראה לקביעת אופן
השינוי של מונה הכתובות

const byte I_D = 0x02;                  // מונה הכתובות גדל באחד

const byte DISPLAY_CONTROL = 0x08;      // קוד ההוראה לקביעת מאפייני
הסמן והתצוגה

const byte D = 0x04;                    // קוד להפעלת התצוגה

const byte FUNCTION_SET = 0x20;         // קוד ההוראה להגדרת שיטת חיבור
התצוגה למיקרו-בקר

const bool LCD_COMMAND = true;           // פנייה לאוגר ההוראות

const bool LCD_DATA = false;             // פנייה לאוגר הנתונים

```

פעולת InitLcd

הפעולה מאתחלת את רכיב התצוגה לפי תרשים הזרימה שבאיור 5.12.

```

void InitLCD() {

    delay(40);                          // המתנה 40mS

    SPI.beginTransaction(setting);       // אתחול תהליך העברה

    LcdWrite(LCD_COMMAND, FUNCTION_SET);

    LcdWrite(LCD_COMMAND, FUNCTION_SET);

    LcdWrite(LCD_COMMAND, DISPLAY_CONTROL | D);

    LcdWrite(LCD_COMMAND, DISPLAY_CLEAR);

    delay(10);                          // המתנה 10mS

    LcdWrite(LCD_COMMAND, ENTRY_MODE_SET | I_D);

}

```

הצגת מידע ברכיב התצוגה

ניתן להציג מידע טקסטואלי ב-4 שורות שונות. הכתובות להצגת מידע ב-4 שורות מוגדרות בתוכנית כערכים קבועים. להלן הגדרת קבועי הכתובות:

```
const byte LINE1 = 0x80;           // כתובת תחילת שורה ראשונה בתצוגה
const byte LINE2 = 0x90;           // כתובת תחילת שורה שנייה בתצוגה
const byte LINE3 = 0x88;           // כתובת תחילת שורה שלישית בתצוגה
const byte LINE4 = 0x98;           // כתובת תחילת שורה רביעית בתצוגה
```

פעולת LcdPrint

הפעולה מציגה טקסט בשורה לפי בחירה.

```
/*
 * The function line address and text and displays it.
 * INPUT: line - line address
 *      text - array of char '\0' terminated
 * OUTPUT: None
 */
void LcdPrint(byte line, const char text[]){
    byte i = 0;
    LcdWrite(LCD_COMMAND, line);           // קביעת כתובת השורה
    while (i < 16 && text[i] != '\0'){    // לולאה להעברת התווים עד לערך המציין סוף
        LcdWrite(LCD_DATA, text[i]);      // כתיבת התו לאוגר הנתונים
        i++;
    }
}
```

דוגמה 5.2 – הצגת טקסט ושעון זמן ברכיב התצוגה

בדוגמה זו מוצג טקסט ב-3 שורות הראשונות, ובשורה הרביעית מוצג ערך של שעון זמן. התוכנית כוללת את הפעולות והקבועים שתוארו לעיל. כדי להקל על קריאות התוכנית וארגונה נכתבו הפעולות והקבועים בקבצים נפרדים.

להלן קובץ **LcdLib.h** המכיל את קבועי הכתובות:

```
const byte LINE1 = 0x80;           // כתובת תחילת שורה ראשונה בתצוגה
const byte LINE2 = 0x90;           // כתובת תחילת שורה שנייה בתצוגה
const byte LINE3 = 0x88;           // כתובת תחילת שורה שלישית בתצוגה
const byte LINE4 = 0x98;           // כתובת תחילת שורה רביעית בתצוגה
```

להלן קובץ **LcdLib** המכיל את הקבועים והפעולות:

```
const byte SYNCFLAG = 0x0F8;       // סיביות 11111 המשמשות לסינכרון 5
const byte RS = 0x02;              // ערך הסיבית הקובע פנייה לאוגר הנתונים
const byte DISPLAY_CLEAR = 0x01;   // קוד ההוראה לאיפוס התצוגה
const byte ENTRY_MODE_SET = 0x04;  // קוד ההוראה לקביעת אופן
// השינוי של מונה הכתובות
const byte I_D = 0x02;             // מונה הכתובות גדל באחד
const byte DISPLAY_CONTROL = 0x08; // קוד ההוראה לקביעת מאפייני
// הסמן והתצוגה
const byte D = 0x04;               // קוד להפעלת התצוגה
const byte FUNCTION_SET = 0x20;    // קוד ההוראה להגדרת שיטת חיבור
// התצוגה למיקרו-בקר
const bool LCD_COMMAND = true;     // פנייה לאוגר ההוראות
```

```

const bool LCD_DATA = false;           // פנייה לאוגר הנתונים

/*
 * פעולה לאתחול התצוגה
 * בהתאם לתרשים הזרימה
 */

void InitLCD(){
    delay(40);
    SPI.beginTransaction(setting);
    LcdWrite(LCD_COMMAND, FUNCTION_SET);
    LcdWrite(LCD_COMMAND, FUNCTION_SET);
    LcdWrite(LCD_COMMAND, DISPLAY_CONTROL | D);
    LcdWrite(LCD_COMMAND, DISPLAY_CLEAR);
    delay(10);
    LcdWrite(LCD_COMMAND, ENTRY_MODE_SET | I_D);
}

/*
 * הפעולה מעבירה בית לאוגר הנתונים או לאוגר ההוראות
 * INPUT: command
 *     if commnad = true transfer to instruction register
 *     if command = false transfer to data register
 *     value a byte to transfer
 */

void LcdWrite(bool command, byte value){

```

```
digitalWrite(SS, HIGH);

SPI.transfer(command ? SYNCFLAG : SYNCFLAG | RS);

SPI.transfer(value & 0xF0);

SPI.transfer(value << 4);

digitalWrite(SS, LOW);

delayMicroseconds(100);

}

/*
 * הפעולה מציגה טקסט בשורה הנקבעת על פי פרמטר מועבר
 * INPUT: line - line address
 *      text - array of char '\0' terminated
 * OUTPUT: None
 */

void LcdPrint(byte line, const char text[]){

    byte i = 0;

    LcdWrite(LCD_COMMAND, line);

    while (i < 16 && text[i] != '\0'){

        LcdWrite(LCD_DATA, text[i]);

        i++;

    }

}
```

להלן התוכנית הראשית:

```

#include <SPI.h> // שלוב מחלקת התקשורת
#include "LcdLib.h" // שלוב קובץ הקבועים

// הגדרת משתני התוכנית

SPISettings setting(8000000L, MSBFIRST, SPI_MODE3);

char timer[10]; // מערך לאחסון מחרוזת השעה
long sec = 45213; // time = 12:33:33

// פעולת אתחול

void setup() {
    Serial.begin(9600); // אתחול תקשורת טורית UART
    SPI.begin(); // אתחול תקשורת טורית SPI
    digitalWrite(SS, LOW); // CS=0
    InitLCD(); // זימון הפעולה לאתחול רכיב התצוגה
    LcdPrint(LINE1+1, " LCD Example 1"); // טקסט מוזח בשורה הראשונה
    LcdPrint(LINE2, " 2 Hello, world "); // טקסט מתחיל שורה 2
    LcdPrint(LINE3, " 3 Hello, world "); // טקסט מתחיל שורה 3
    LcdPrint(LINE4, " Time: "); // טקסט מתחיל שורה 4
}

// הפעולה הראשית

void loop() {
    byte h = sec / 3600; // חישוב שעה
    byte m = sec % 3600 / 60 ; // חישוב דקות
    byte s = sec % 3600 % 60; // חישוב שניות
    sprintf(timer, "%02d:%02d:%02d", h, m, s); // יצירת מחרוזת של שעון זמן

```

```

LcdPrint(LINE4 + 3, timer);    // זימון פעולה להצגת מחרוזת הזמן במקום
המתאים
Serial.println(timer);          // הצגת השעה במוניטור
sec++;                          // קידום מונה השניות
if (sec == 3600*24){            // האם עברו 24 שעות
    sec = 0;                    // איפוס מונה השניות
}
delay(10);                      // השהייה (גודל ההשהיה צריך להיות 1 שנייה)
}

```

ביאור התוכנית:

לצורך השימוש בתקשורת SPI שולבה בתוכנית המחלקה SPI.

פעולת setup – בפעולה הוגדרו פרמטרי התקשורת ואותחל רכיב התצוגה. בהמשך הוצגו ארבע שורות של טקסט. בשורה הראשונה הטקסט אינו מתחיל בתחילת השורה מכיוון שנוסף הערך 1 לכתובת השורה הראשונה: `LcdPrint(LINE1+1, " LCD Example ");`

פעולת loop – בפעולה מומשה הדמייה של שעון זמן של 24 שעות. השעון לא מראה את השעה האמיתית, אלא יחידת זמן קצרה יותר. זאת כדי לבחון את הזמן החולף בתקופה קצרה יותר. ערך השעה מוצג במקביל בתוכנת המוניטור וביחידת התצוגה. הפעולה מציגה רק את השורה הרביעית מכיוון שהשעה מוצגת בשורה זו. ההוראה להצגת השעה היא:

`LcdPrint(LINE4 + 3, timer);`

הערך 3 מציין דילוג על 6 תווים בשורה מכיוון שלכל כתובות משדרים נתון של 16 סיביות המהווים 2 תווים.

תוצאת ההרצה מוצגת באיור 5.13 להלן:



איור 5.13: תמונת מסך יחידת התצוגה

תרגול 

שאלה 5.10

כתבו תוכנית המציגה המידע הזה:

שורה 1: שם בית הספר

שורה 2: כיתה

שורה 3: שם התלמיד

שורה 4: שנת הלידה

שאלה 5.11

חזרו על שאלה 5.10, אבל הפעם כתבו תוכנית שבה שנת הלידה תופיע במרכז השורה.

אופן פעולה גרפי

באופן פעולה זה מתייחסים להצגת תמונה על פני המסך כולו ללא התייחסות למבנה השורה שהוצג באופן הפעולה הרגיל (טקסט).

תמונה

תמונה ממוחשבת בנויה ומאוחסנת מאוסף של נקודות המאורגנות במבנה מלבני. לכל נקודה בתמונה יש ערך של צבע, המקודד באופן מספרי, והמתקבל ממגוון צבעים קיים. מגוון הצבעים הוא אחד הפרמטרים הקובעים את איכות התמונה. ככל שמספר הצבעים גדול יותר כך התמונה עשירה יותר.

כאשר מציגים את התמונה המאוחסנת, כל נקודה בה הנקראת גם פיקסל (pixel), מתאימה לנקודה במסך (גם התמונה וגם המסך הם מלבניים). המרחק בין שתי נקודות סמוכות במסך הוא הפרמטר הנוסף הקובע את איכות התמונה. ככל שהמרחק בין הנקודות קטן יותר, כך איכות התמונה טובה יותר.

תצוגה חד-צבעית (מונוכרומטית) היא יחידת תצוגה המציגה צבע אחד בלבד או שאינה מציגה צבע כלל. הצבע המוצג מקודד באמצעות הערך 1. הערך 0 מייצג מצב בו צבע לא מוצג. הצבע שיוצג תלוי בצבעה של יחידת התצוגה.

ביחידת התצוגה המוצגת באיור 5.13, הערך 1 מייצג את הצבע הלבן ואילו הערך 0 מייצג את צבע הרקע – כחול. גודל המסך המוצג באיור זה הוא 128×64 פיקסלים: 64 שורות ו-128 פיקסלים בכל שורה. המסך כולל 8192 פיקסלים. גודל הזיכרון הדרוש לאחסון פיקסלים אלה הוא 1KB.

כל תמונה שתוצג ביחידת התצוגה חייבת להיות מאוחסנת בזיכרון. כל תמונה כזו משתמשת ב-1KB מתוך מרחבי הזיכרון.

כדי להציג תמונה ביחידת התצוגה, עלינו לבחור תמונה בגודל המתאים. ליחידת התצוגה שאנו משתמשים בה נדרשת תמונה מונוכרומטית בגודל 128×64 . באמצעות כלי עזר בשם LCDAssistant, המתואר בנספח ב, ניתן ליצור מהתמונה מערך חד-ממדי בגודל 1KB.

הוראות הגדרה (commands) – הרחבה

הוראות ההרחבה נכנסות לתוקף לאחר שינוי ערכה של הסיבית $RE=1$ במילת הבקרה Function Set.

בהוראות להלן מתבצעת כתיבה ($RW=0$) לאוגר ההוראות ($RS=0$).

Standby

ההוראה תעביר את כרטיס הזיכרון למצב המתנה. כל הוראה לאחר הוראה זו תוציא את הכרטיס ממצב המתנה. תוכן זיכרון DDRAM נשאר ללא שינוי.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	1

קוד ההוראה $0x01$ (DB7–DB0).

Vertical Scroll or RAM Address Select

ההוראה מאפשרת את פעולת הגלילה האנכית אליה נתייחס בהמשך או את האפשרות לקבוע כתובת במרחב זיכרון – CGRAM שתואר בסעיף הוראות בסיסיות.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	SR

קודי ההוראה $0x02$ או $0x03$ (DB7–DB0).

הערך בזמן האתחול הוא: $0x02$.

SR: בחירת גלילה

1 – הפעלת הגלילה האנכית

0 – אפשרור ההוראה הבסיסית Set CGRAM Address

Reverse

ההוראה הופכת את כיוון ההצגה של הטקסט באחת מהשורות 1 עד 4. בכל שימוש בהוראת היפוך ישתנה כיוון ההצגה של הטקסט המופיע בשורה.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	R0	R1

קודי ההוראה 0x04 עד 0x07 (DB7–DB0).

הערך בזמן האתחול הוא: 0x04.

תיאור	R1	R0
השורה הראשונה במצב רגיל או מהופך	L	L
השורה השנייה במצב רגיל או מהופך	L	H
השורה השלישית במצב רגיל או מהופך	H	L
השורה הרביעית במצב רגיל או מהופך	H	H

שימו לב כי ניתן להציג רק שתי שורות מתוך הארבע בבקר תצוגה יחיד.

Extended Function Set

ההוראה קובעת את אופן פעולת ממשק הנתונים המקבילי, את השליטה בפנייה לסדרת ההוראות המורחבות ובהפעלה או כיבוי של יחידת התצוגה.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	x	RE	G	x

x – אין חשיבות לערך

קודי ההוראה 0x20 עד 0x3F (DB7–DB0).

בזמן הפעלת הוראות ההרחבה ערכי הסיביות הן RE=1, G=0.

DL: בקרת ממשק 4/8 סיביות

1 – ממשק 8 סיביות

0 – ממשק 4 סיביות

RE: בקרת סדרת הוראות מורחבות

1 – הפעלת סדרת הוראות מורחבת

0 – הפעלת סדרת הוראות בסיסיות

G: בקרת תצוגה גרפית

1 – הפעלת תצוגה גרפית

0 – כיבוי תצוגה גרפית

אי אפשר לשנות בו-זמנית את DL, RE ו-G. יש לקבוע קודם את DL או G ורק אחר כך את RE.

Set Scroll Address

הוראה הקובעת את כתובת הגלילה האנכית במונה הכתובות כאשר $SR=1$ בהוראה Vertical Scroll or RAM Address Select.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

קודי ההוראה 0x40 עד 0x7F (DB7–DB0).
כתובת הגלילה האנכית היא בעצם כתובת של שורה בטבלה.

Set Graphic RAM Address

ההוראה קובעת את מונה הכתובות (AC) של מרחב הזיכרון GDRAM. משתמשים בהוראה זו פעמיים:

- ההוראה הראשונה קובעת את הכתובת האנכית (vertical).
- ההוראה השנייה קובעת את הכתובת האופקית (horizontal).

כותבים שתי ההוראות ברצף כדי להשלים את קביעת הכתובת.

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	AC5	AC4	AC3	AC2	AC1	AC0

RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	0	0	0	AC3	AC2	AC1	AC0

מרחב הכתובות האנכיות AC5..AC0

מרחב הכתובות האופקיות AC3..AC0

כתובת אנכית היא כתובת של שורה בטבלה.

כתובת אופקית היא כתובת שמאוחסנים בה שני בתים בזה אחר זה בשורה אחת.

מונה הכתבות (AC) של הזיכרון הגרפי גדל באופן אוטומטי לאחר קביעת הכתובות האופקיות והאנכיות. קידום של הכתובת האופקית מתבצע עד לכתובת 0x0F, לאחר מכן הכתובת חוזרת ל-0x00 באופן אוטומטי.

בכל אופן, הכתובת האנכית אינה מתעדכנת כתוצאה מאיפוס של הכתובת האופקית.

קבועי התוכנית

```

const byte SYNCFLAG = 0x0F8;          // סיביות 11111 המשמשות לסינכרון 5
const byte RS = 0x02;                  // ערך הסיבית הקובעת פנייה לאוגר
הנתונים

                                     // הוראות בסיסיות

// Basic COMMANDS RS=0 RW=0 RE=0

const byte DISPLAY_CLEAR = 0x01;      // קוד ההוראה לאיפוס התצוגה
const byte ENTRY_MODE_SET = 0x04;     // קוד ההוראה לקביעת אופן
השינוי של מונה הכתבות

const byte I_D = 0x02;                 // מונה הכתובות גדל באחד
const byte DISPLAY_CONTROL = 0x08;    // קוד ההוראה לקביעת מאפייני
הסמן והתצוגה

const byte D = 0x04                    // קוד להפעלת התצוגה
const byte FUNCTION_SET = 0x20;       // קוד ההוראה להגדרת שיטת חיבור
התצוגה למיקרו-בקר

                                     // RS=0 RW=0 RE=1 הוראות
ההרחבה

const byte EXTENDED_FUNCTION_SET = FUNCTION_SET;
const byte G = 0x02;                  // קוד להפעלת אופן העבודה הגרפי
const byte RE = 0x04;                 // קוד להפעלת הוראות ההרחבה
const byte SET_GRDISRAM_ADDR = 0x80;  // כתובת הבסיס של הזיכרון הגרפי

const bool LCD_COMMAND = true;        // פנייה לאוגר ההוראות
const bool LCD_DATA = false;          // פנייה לאוגר הנתונים

```


מעברים בין מצבי טקסט וגרפיקה

LCD_EnableGraphics

הפעולה מפעילה את אופן העבודה הגרפי ואת השימוש בהוראות ההרחבה.

```
/*  
 *                               // הפעולה מפעילה את המצב הגרפי  
 */  
void LCD_EnableGraphics(){  
    LcdWrite(LCD_COMMAND, EXTENDED_FUNCTION_SET | RE | G);  
}
```

LCD_DisableGraphics

הפעולה מבטלת את אופן העבודה הגרפי ומפעילה את אופן עבודה רגיל (טקסט).

```
/*  
 *                               // הפעולה מכבה את המצב הגרפי  
 */  
void LCD_DisableGraphics(){  
    LcdWrite(LCD_COMMAND, FUNCTION_SET);  
}
```

כתיבה לזיכרון הגרפי

כתובת הזיכרון הגרפי בנויה משתי תתי-כתובות: הכתובת האנכית והכתובת האופקית. ערך בן שני בתים (16 סיביות: D15..D0) קובע נתון בכתובת אופקית אחת. הכתובת האופקית גדלה באחת באופן אוטומטי ומתאפסת לאחר שהגיעה לכתובת 0x0F. הכתובת האנכית אינה משתנה.

שלבי התהליך:

1. קבע את הכתובת האנכית (Y) של הזיכרון הגרפי (מספר השורה)
2. קבע את הכתובת האופקית (X) של הזיכרון הגרפי.
3. כתוב את ערך הבית הראשון D15..D8 לזיכרון הגרפי.
4. כתוב את ערך הבית השני D7..D0 לזיכרון הגרפי.

בשלבים 1 ו-2 מתבצעת פנייה לאוגר ההוראות (LCD_COMMAND), ובשלבים 3 ו-4 מתבצעת פנייה לאוגר הנתונים (LCD_DATA).

ערך הכתובת האנכית הוא מספר בתחום 0 עד 63 וערך הכתובת האופקית הוא מספר בתחום 0 עד 15. ערכי הכתובות משולבים בתוך ההוראה המורחבת הקובעת את הכתובת במרחב הזיכרון הגרפי Set Graphic RAM Address. בכל כתובת אופקית מאוחסנים שני בתים ברצף. להלן קוד התהליך:

```
LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR); // כתובת אנכית
LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR); // כתובת אופקית
LcdWrite(LCD_DATA, value);           // נתון לבית ראשון
LcdWrite(LCD_DATA, value);           // נתון לבית שני
```

מיפוי הזיכרון למסך

מסך התצוגה מחולק לשני חצאים: תצוגה עליונה ותצוגה תחתונה.

הצגה בתצוגה העליונה

כדי להציג בתצוגה העליונה נפנה ל-32 הכתובות האנכיות ונעביר 8 זוגות ערכים בגודל בית באופן רציף החל מכתובת אופקית 0.

הצגה בתצוגה תחתונה

כדי להציג בתצוגה התחתונה נפנה ל-32 הכתובות האנכיות ונעביר 8 זוגות ערכים בגודל בית באופן רציף החל מכתובת אופקית 0x08.

LCD_FillGraphics

הפעולה מקבלת ערך מספרי (value) כפרמטר, ומאתחלת את מרחב הזיכרון הגרפי בערך זה. להלן קוד הפעולה:

```
/*
 * הפעולה מאתחלת את כל מרחב הזיכרון הגרפי בערך המתקבל כפרמטר
 * INPUT: value – תוכן הזיכרון – ערך הקובע את תוכן הזיכרון
 */
void LCD_FillGraphics(byte value){
    byte x, y;
    for(y=0; y<32; y++){          // 32 כתובות בתצוגה
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR | y); // שורה Y
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR);    // עמודה 0
        for(x=0; x<16; x++){      // העברת 16 מלים לשורה
            LcdWrite(LCD_DATA, value); // מונה אופקי משתנה באופן אוטומטי
            LcdWrite(LCD_DATA, value);
        }
    }
}
```

ביאור הפעולה:

בפעולה פונים ל-32 כתובות באמצעות המשתנה y. הכתובות מציינות את השורות בתצוגה. לכל אחת מהכתובות מעבירים 16 זוגות של ערכים ברצף בזה אחר זה. ערכים אלה ממלאים את מרחב הזיכרון של אותה כתובת.

בפעולה מועברים 32 בתיים לכל אחת מ-32 השורות, שהם $1024B = 1KB$.

בפעולה לא התייחסנו למיפוי המסך לזיכרון מכיוון שהעברנו ערך אחד בלבד לכל מרחב הזיכרון הגרפי.

דוגמה 5.3 – הצגת טקסט ושינוי התצוגה הגרפית

בדוגמה זו נראה את השימוש ביחידת התצוגה ואת המעבר מהמצב הרגיל למצב הגרפי באמצעות הפעולה `LCD_EnableGraphics`, וחזרה למצב הרגיל באמצעות הפעולה `LCD_DisableGraphics`. נתחיל באופן עבודה רגיל (הצגת טקסט), ובו נציג שתי שורות של טקסט. לאחר מכן נעבור למצב גרפי, ובו נמלא את המסך בשלושה ערכים שונים: `0xAA`, `0x55` ו-`0x00` באמצעות הפעולה `LCD_FillGraphics`. שני הערכים `0xAA` ו-`0x55` יגרמו למילוי המסך בקווים אנכיים ואילו הערך `0x00` ינקה את המסך. בסיום נחזור למצב הרגיל. המעבר בין מצב עבודה גרפי למצב רגיל אינו משפיע על המידע הטקסטואלי המוצג.

התוכנית כוללת את הפעולות והקבועים שתוארו עד כה. כדי להקל על קריאות התוכנית וארגונה נכתבו הפעולות והקבועים בקבצים נפרדים.

להלן קובץ `LcdLib.h` המכיל את קבועי הכתובות:

```
const byte LINE1 = 0x80;           // כתובת תחילת השורה הראשונה בתצוגה
const byte LINE2 = 0x90;           // כתובת תחילת השורה השנייה בתצוגה
const byte LINE3 = 0x88;           // כתובת תחילת השורה השלישית בתצוגה
const byte LINE4 = 0x98;           // כתובת תחילת השורה הרביעית בתצוגה
```

להלן קובץ `LcdLib` המכיל את הקבועים והפעולות:

```
const byte SYNCFLAG = 0x0F8;       // סיביות 11111 המשמשות לסינכרון 5
const byte RS = 0x02;              // ערך הסיבית הקובע פנייה לאוגר הנתונים

                                   // הוראות בסיסיות
                                   // Basic COMMANDS RS=0 RW=0
RE=0
```

```

const byte DISPLAY_CLEAR = 0x01; // קוד ההוראה לאיפוס התצוגה

const byte ENTRY_MODE_SET = 0x04; // קוד ההוראה לקביעת אופן השינוי של
מונה הכתבות

const byte I_D = 0x02; // מונה הכתובות גדל באחד

const byte DISPLAY_CONTROL = 0x08; // קוד ההוראה לקביעת מאפייני הסמן
והתצוגה

const byte D = 0x04; // קוד להפעלת התצוגה

const byte FUNCTION_SET = 0x20; // קוד ההוראה להגדרת שיטת חיבור
התצוגה למיקרו-בקר

// RS=0 RW=0 RE=1 הוראות הרחבה

const byte EXTENDED_FUNCTION_SET = FUNCTION_SET;

const byte G = 0x02; // קוד להפעלת אופן העבודה הגרפי

const byte RE = 0x04; // קוד להפעלת הוראות ההרחבה

const byte SET_GRDISRAM_ADDR = 0x80; // כתובת הבסיס של הזיכרון הגרפי

const bool LCD_COMMAND = true; // פנייה לאוגר ההוראות

const bool LCD_DATA = false; // פנייה לאוגר הנתונים

/*
פעולה המשנה את העבודה לאופן גרפי *
*/

void LCD_EnableGraphics(){
    LcdWrite(LCD_COMMAND, EXTENDED_FUNCTION_SET | RE | G);
}

```

```
/*  
 * פעולה המשנה את העבודה לאופן רגיל *  
 */  
void LCD_DisableGraphics(){  
    LcdWrite(LCD_COMMAND, FUNCTION_SET);  
}  
/*  
 * מילוי הזיכרון בערך המתקבל כפרמטר *  
 * INPUT: value - a value to set the graphic memory  
 */  
void LCD_FillGraphics(byte value){  
    byte x, y;  
    for(y=0; y<32; y++){  
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR | y);  
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR);  
        for(x=0; x<16; x++){  
            LcdWrite(LCD_DATA, value);  
            LcdWrite(LCD_DATA, value);  
        }  
    }  
}  
/*  
 * פעולה לאתחול התצוגה *  
 * בהתאם לתרשים הזרימה *
```



```

*/

void InitLCD(){

    delay(40);

    SPI.beginTransaction(setting);

    LcdWrite(LCD_COMMAND, FUNCTION_SET);

    LcdWrite(LCD_COMMAND, FUNCTION_SET);

    LcdWrite(LCD_COMMAND, DISPLAY_CONTROL | D);

    LcdWrite(LCD_COMMAND, DISPLAY_CLEAR);

    delay(10);

    LcdWrite(LCD_COMMAND, ENTRY_MODE_SET | I_D);
}

/*
* הפעולה מעבירה בית לאוגר הנתונים או לאוגר ההוראות
* INPUT: command
*     if commnad = true transfer to instruction register
*     if command = false transfer to data register
*     value a byte to transfer
*/

void LcdWrite(bool command, byte value){

    digitalWrite(SS, HIGH);

    SPI.transfer(command ? SYNCFLAG : SYNCFLAG | RS);

    SPI.transfer(value & 0xF0);

    SPI.transfer(value << 4);

```

```
digitalWrite(SS, LOW);

delayMicroseconds(100);

}

/*
 * הפעולה מציגה טקסט בשורה הנקבעת על פי פרמטר מועבר
 * INPUT: line - line address
 *      text - array of char '\0' terminated
 * OUTPUT: None
 */

void LcdPrint(byte line, const char text[]){
    byte i = 0;

    LcdWrite(LCD_COMMAND, line);

    while (i < 16 && text[i] != '\0'){
        LcdWrite(LCD_DATA, text[i]);

        i++;
    }
}
```

להלן התוכנית הראשית:

```

#include <SPI.h> // שילוב מחלקת התקשורת

#include "LcdLib.h" // שילוב קובץ הקבועים

// הגדרת משתנה התוכנית

SPISettings setting(8000000L, MSBFIRST, SPI_MODE3);

// פעולת אתחול התוכנית

void setup() {

  Serial.begin(9600); // אתחול תקשורת טורית UART

  SPI.begin(); // אתחול תקשורת טורית SPI

  digitalWrite(SS, LOW); // CS=0

  InitLCD(); // זימון הפעולה לאתחול רכיב התצוגה

  LcdPrint(LINE1+1, "LCD Example 2"); // טקסט מוזח בשורה הראשונה

  LcdPrint(LINE3+1, "Any Text"); // טקסט מתחיל שורה 3

}

// הפעולה הראשית

void loop() {

  delay(1000); // השהייה להצגת מסך באופן פעולה רגיל

  LCD_EnableGraphics(); // מעבר לאופן פעולה גרפי

  LCD_FillGraphics(0x55); // הצגת מסך בעל פסים אנכיים

  delay(1000);

  LCD_FillGraphics(0xAA); // הצגת מסך בעל פסים אנכיים משלימים

  delay(1000);

  LCD_FillGraphics(0x00); // ניקוי המסך הגרפי

```

```
LCD_DisableGraphics();      // חזרה למצב רגיל
}
```

ביאור התוכנית:

לצורך השימוש בתקשורת SPI שולבה בתוכנית המחלקה SPI.

פעולת setup – בפעולה הוגדרו פרמטרי התקשורת ואותחל רכיב התצוגה. בהמשך הוצגו שתי שורות של טקסט. בשתי השורות הטקסט אינו מתחיל בתחילת השורה מכיוון שנוסף הערך 1 לכתובת. לדוגמה בשורה הראשונה: `LcdPrint(LINE1+1, " LCD Example`;".

פעולת loop – בפעולה נעשה שימוש בהשהייה של שנייה אחת בין המעברים השונים כדי לאפשר לצופה זמן כדי להבחין בין המעברים. בתחילה עוברים למצב פעולה גרפי ומציגים בזה אחר זה עם השהיות את שלושת הערכים 0xAA, 0x55 ו-0x00. לאחר מכן חוזרים למצב טקסט. התהליך חוזר בלולאה אינסופית.

LCD_FullScreenGraphics

הפעולה מקבלת מערך בשם graphics של מספרים שלמים מטיפוס בית. גודלו של המערך הוא 1024 בתים, כגודל הזיכרון הגרפי ביחידת התצוגה. הערכים במערך מייצגים תמונה חד-צבעית בגודל 128x64 פיקסלים. הפעולה מעתיקה את איברי המערך אל מרחב הזיכרון הגרפי. להלן קוד הפעולה:

```
/*  
 * הפעולה מציבה במרחב הזיכרון הגרפי את ערכי המערך  
 * INPUT: graphics is an array of size 128x64=1024 bytes  
 */  
void LCD_FullScreenGraphics(const byte graphics[]){  
    byte x, y;  
  
    // מחצית עליונה של המסך  
    for(y=0; y<32; y++){  
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR | y);  
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR);  
        for(x=0; x<8; x++){  
            LcdWrite(LCD_DATA, graphics[2*x + 16*y]);  
            LcdWrite(LCD_DATA, graphics[2*x+1 + 16*y]);  
        }  
    }  
  
    // מחצית תחתונה של המסך  
    for(; y<64; y++){  
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR | (y-32));  
        LcdWrite(LCD_COMMAND, SET_GRDISRAM_ADDR | 0x08);
```

```

for(x=0; x<8; x++){
    LcdWrite(LCD_DATA, graphics[2*x + 16*y]);
    LcdWrite(LCD_DATA, graphics[2*x+1 + 16*y]);
}
}
}

```

ביאור הפעולה:

המשתנה y המקבל ערכים מאפס עד 63 משמש לשתי מטרות: כתובת אנכית וחישוב מיקום במערך.

כתובת אנכית

המשתנה y משמש לפנייה לשורות ביחידת התצוגה. כפי שהוסבר, בחלק מיפוי הזיכרון למסך, פונים בפועל לכתובות אפס עד 31. ולכן כאשר מתבצעת פנייה למחצית העליונה של המסך משתמשים בערכו של המשתנה בתחום אפס עד 31, וכאשר מתבצעת פנייה למחצית התחתונה של המסך משתמשים בערכו של המשתנה בתחום 32 עד 63. כדי לתקן את תחום כתובות השורה לתחום אפס עד 31, משתמשים בפעולה החשבונית: $y-32$.

מיקום במערך

המערך החד-ממדי `graphics` מכיל 1024 בתים המייצגים את התמונה. המחצית הראשונה של המערך מכיל את המחצית הראשונה של התמונה, והמחצית השנייה שלו את המחצית השנייה של התמונה.

מכיוון שהמערך הוא חד-ממדי ואילו המסך הוא דו-ממדי (מלבני), משתמשים בפעולה חשבונית כדי למפות כל ערך במערך החד-ממדי לערך המייצג שורה ועמודה במסך. לשם כך משתמשים במשתנה y המקבל ערכים בתחום מאפס עד 63, ובמשתנה x המקבל ערכים בתחום מאפס עד 7.

המסך מיוצג באמצעות זוג ערכים x ו- y . גודל המסך על פי קואורדינטות אלה הוא 64×8 . כל מקום המוגדר באופן זה מייצג ערך בגודל של שני בתים. המעבר מערך דו-ממדי המיוצג באמצעות המשתנים x ו- y מוצג בנוסחאות האלה:

כתובת זוגית במערך: $16y + 2 \cdot x$

כתובת אי-זוגית במערך: $16y + 2 \cdot x + 1$

לדוגמה, כאשר רוצים להעביר ערך מהמערך החד-ממדי לשורה 11 במסך ולעמודה 2, נציב בביטויי הכתובת $y=11$ ו- $x=2$. הערך המחושב של הכתובות המתאימות במערך לאחר ההצבה הן: 180 ו-181.

דוגמה 5.4 – הצגת תמונה חד-צבעית

בדוגמה זו נציג את התמונה החד-צבעית בגודל 128x64 פיקסלים הזו:



מתמונה זו נוצר מערך חד-ממדי באמצעות כלי העזר LCDAssistant בגודל 1kB בשם images. המערך הוגדר בתוך קובץ בשם andro.h.

להלן קטע מהמערך:

```
1 //-----
2 // File generated by LCD Assistant
3 // http://en.radzio.dxp.pl/bitmap\_converter
4 //-----
5
6 const unsigned char images [] = {
7 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
8 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
```

להלן התוכנית הראשית:

```
#include <SPI.h> // שילוב מחלקת התקשורת
#include "LcdLib.h" // שילוב קובץ הקבועים
#include "andro.h" // שילוב קובץ התמונה

SPISettings setting(8000000L, MSBFIRST, SPI_MODE3);

void setup() {
    Serial.begin(9600); // אתחול תקשורת טורית UART
    SPI.begin(); // אתחול תקשורת טורית SPI
    digitalWrite(SS, LOW); // CS=0
    InitLCD(); // זימון הפעולה לאתחול רכיב התצוגה
```



```

}

void loop() {

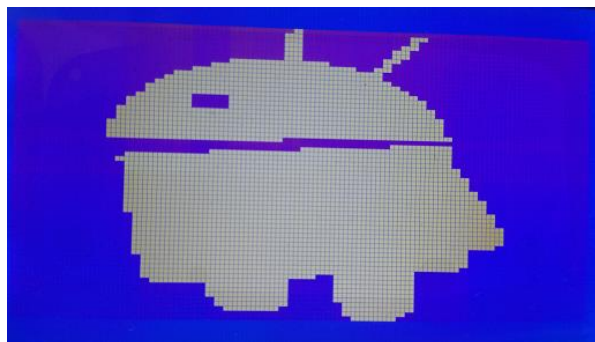
  delay(2000);

  LCD_EnableGraphics();           // אפשרור הגרפיקה
  LCD_FullScreenGraphics(images); // הצגת התמונה במסך
  delay(2000);

  LCD_FillGraphics(0x00);         // ניקוי התצוגה
  LCD_DisableGraphics();         // יציאה ממצב גרפי
}

```

להלן התמונה שהתקבלה בתצוגה:



תרגול 

שאלה 5.12

כתבו פעולה בשם rect המציירת במסך מלבן מלא בממדים 80x40.

שאלה 5.13

חזרו על שאלה 5.12 ומרכזו את המלבן למרכז המסך.

5.14 שאלה

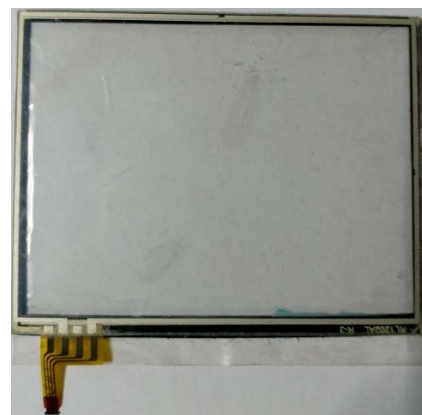
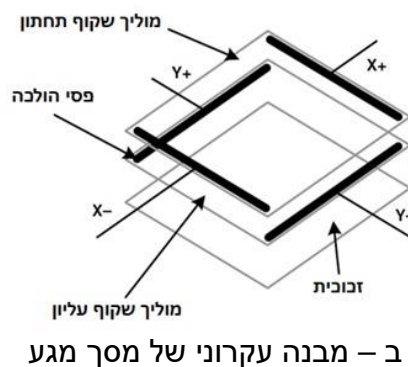
חפשו באינטרנט תמונה חד-צבעית בגודל 128x64. השתמשו בכלי LCDAssistant, וצרו קובץ המכיל את התמונה.

הריצו את התוכנית ובדקו אם אכן מתקבלת התמונה.

5.9 מסך מגע

עיקרון הפעולה

מסך מגע התנגדותי בנוי משתי שכבות שקופות המצופות בחומר מוליך המונחות אחת על השנייה. כאשר נוצר לחץ על המסך ממגע של אצבע או עט, השכבה העליונה יוצרת קשר עם השכבה התחתונה. באיור 5.14-א מוצג מסך מגע הנפוץ במסכי LCD גרפיים, ובאיור 5.14-ב מתואר מבנהו העקרוני של מסך המגע.

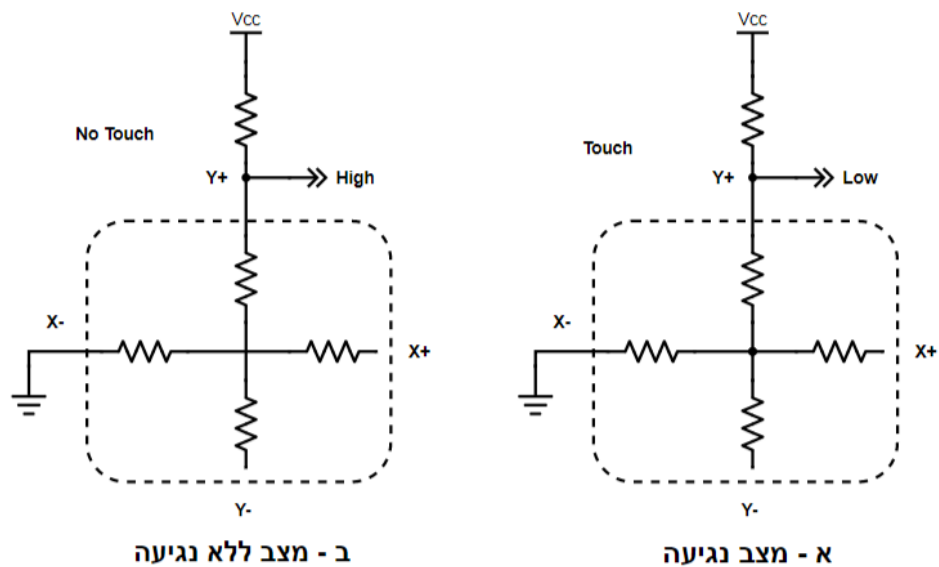


איור 5.14: מבנה מסך מגע

כאשר מחברים מתח לאחת מהשכבות, נוצר מחלק מתח בנקודת הלחיצה. ערך הקואורדינטה של נקודת המגע בשכבה שבה המתח סופק נקבע על פי קריאת המתח המתקבל באחד מפסי ההולכה של השכבה האחרת.

זיהוי מגע

כדי לדעת אם נוצר מגע במסך, מחברים מתח חיובי (V_{cc}) דרך נגד pullup להדק $Y+$ וחיבור הדק $X-$ לאדמה. התנגדות ל-pullup חייבת להיות גדולה משמעותית מהתנגדות המוחלטת של מסך המגע, שהיא בדרך כלל כמה מאות אוהמים. כאשר אין מגע, המתח בהדק $Y+$ שווה למתח החיובי כפי שניתן לראות באיור 5.15-ב. כאשר יש מגע נוצר מחלק מתח בין נגד ה-pullup לבין מסך המגע, והמתח בהדק $Y+$ יהיה נמוך כפי שמוצג באיור 5.15-א.



איור 5.15: זיהוי לחיצה במסך מגע

קריאת קואורדינטות מסך מגע 4-Wire

ערכי הקואורדינטות x ו- y של מסך מגע בעל 4 חוטים נקראים בשני שלבים באופן בלתי תלוי. בשלב אחד קוראים את הערך x , ובשלב האחר קוראים את הערך y או סליהפך.

קריאת ערך y

כפי שניתן לראות באיור 15.16-א ההדק $Y+$ מחובר למתח גבוה V_{cc} , וההדק $Y-$ מחובר לאדמה (Gnd). קריאת ערך המתח מתבצעת בהדק $X+$ (או בהדק $X-$). ערך המתח שנקרא הוא ביחס ישר לערך קואורדינטת y בנקודת הלחיצה. ערך ה- y יחושב על פי הנוסחה הזו:

$$y = \frac{V_{X+}}{V_{CC}} \times height$$

height – גובה המסך

V_{CC} – המתח הגבוה המס

ופק

V_{X+} – המתח הנמדד בהדק X+.

קריאת ערך X

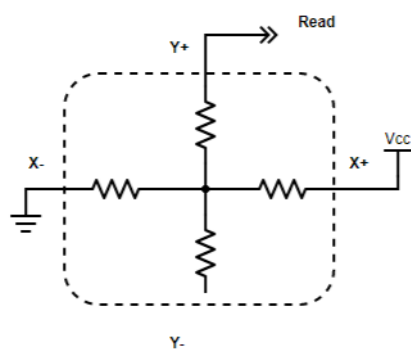
כפי שניתן לראות באיור 15.16-ב הדק X+ מחובר למתח גבוה V_{CC} , והדק X- מחובר לאדמה (Gnd). קריאת ערך המתח מתבצעת בהדק Y+ (או Y-). ערך המתח שנקרא הוא ביחס ישר לערך קואורדינטת X בנקודת הלחיצה. ערך ה-X יחושב על פי הנוסחה הזו:

$$x = \frac{V_{Y+}}{V_{CC}} \times width$$

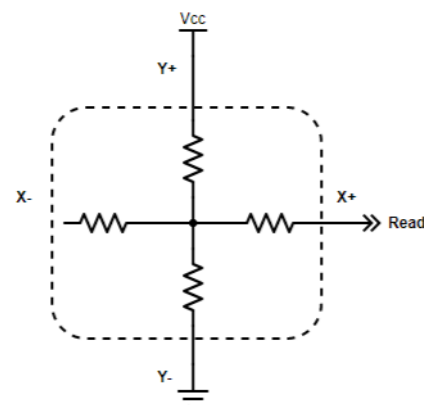
width – רוחב המסך

V_{CC} – המתח הגבוה המסופק

V_{Y+} – המתח הנמדד בהדק Y+.



ב - קריאת ערך X



א - קריאת ערך Y

איור

5.16: אופן החיבור לקריאת ערכי קואורדינטות

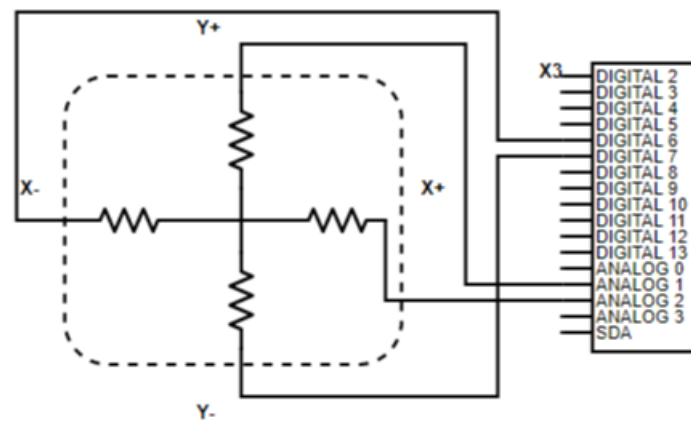
אופן החיבור ללוח הפיתוח

מסך מגע יחובר לארבעה הדקים בלוח הפיתוח:

- שני הדקים ספרתיים שיוגדרו כקלט או כפלט לפי הצורך
- שני הדקי מבוא אנלוגיים

באיור 15.17 מתואר חיבור של הדקי מסך המגע אל לוח הפיתוח. הדקי לוח המגע מחוברים כך:

הדקי מסך מגע	Y-	Y+	X-	X+
הדקי לוח פתוח	7	A1	6	A2



איור 5.17: חיבור מסך מגע אל הדקי לוח הפיתוח

תוכנית לזיהוי לחיצה

בתוכנית זו נשתמש בהגדרות המוצגות באיור 5.15. להלן קוד התוכנית:

```
// מיפוי הדקי מסך המגע ללוח הפיתוח

#define XM 6
#define XP A2
#define YM 7
#define YP A1

void setup() {
    pinMode(XM, OUTPUT);           // הגדרת ההדק כפלט
    digitalWrite(XM, LOW);         // קביעת ערך ההדק לרמה נמוכה
    pinMode(YP, INPUT_PULLUP);     // הגדרת ההדק כקלט עם נגד pullup
    Serial.begin(9600);
}

void loop() {
    if (digitalRead(YP) == HIGH){   // בדיקת מצב ההדק – אם לא מזוהה
        לחיצה
        Serial.println("No touch");
    }
    else{
        Serial.println("Touched");
    }
    delay(200);
}
```

התוכנית פועלת על פי שיטת התשאול (polling). לפי שיטה זו מתבצעת קריאה מתמדת של הדק Y+ ועל פי תוצאת הקריאה מתקבלת החלטה.

תרגול 

שאלה 5.15

- א. מהו השינוי שצריך לבצע בתוכנית שהוצגה לעיל, אם משתמשים בהדק Y- במקום בהדק Y+ ובהדק X+ במקום X-?
- ב. מהם הצירופים הנוספים המשמשים לזיהוי לחיצה?

שאלה 5.16

השיטה החלופית לשיטת התשאול לזיהוי מצבו של הדק היא שיטת הפסיקה (interrupt). שיטה זו נלמדה ביחידת הלימוד "מבוא למערכות משובצות מחשב", בפרק 5 "פסיקות חומרה ומונים".

- א. הציעו תרשים חיבורים של מסך המגע אל לוח הפיתוח.
- ב. כתבו תוכנית לזיהוי לחיצה לפי שיטה זו.

תוכנית לקריאת קואורדינטות של נקודת הלחיצה

בתוכנית זו נשתמש בהגדרות המוצגות באיור 5.17. להלן קוד התכנית:

```
// מיפוי הדקי מסך המגע ללוח הפיתוח

#define XM 6
#define XP A2
#define YM 7
#define YP A1

// הערכים כפי שנקראו בפינות של מסך המגע
const int left = 125, right = 965, top = 85, bottom = 905;

// ממדי המסך ביחידות פיקסלים
```

```

const int touchHeight = 240, touchWidth = 320;

int x, y;                                // משתנים עבור הערך הגולמי
int xpos, ypos; // משתנים עבור הערך לאחר מיפוי הערך הגולמי לממדי המסך

void setup() {
  Serial.begin(9600);
}

void loop() {
  // הגדרת הדקי לוח הפיתוח כדי לזהות לחיצה על פי תוכנית קודמת
  pinMode(XM, OUTPUT);
  digitalWrite(XM, LOW);
  pinMode(YP, INPUT_PULLUP);
  if (digitalRead(YP) == LOW){ // בדיקה אם קיימת לחיצה
    readTouch();              // פעולה לקריאת קואורדינטת של הלחיצה
    // הצגת ערכים גולמיים של קואורדינטות נקודת הלחיצה

    Serial.println("Raw values of x and y");
    Serial.print("x = ");
    Serial.print(x);
    Serial.print(", y = ");
    Serial.println(y);

    // מיפוי ערכי קואורדינטות גולמיים לקואורדינטות מסך בפיקסלים
    xpos = map(x, left, right, 0, touchWidth);
  }
}

```



```

ypos = map(y, top, bottom, 0, touchHeight);

        // הצגת ערכים של קואורדינטות נקודת הלחיצה לאחר המיפוי

Serial.println("Mapped values of x and y");

Serial.print("x = ");

Serial.print(xpos);

Serial.print(", y = ");

Serial.println(ypos);

}

delay(200);

}

// הפעולה מגדירה את ההדקים לקריאת הקואורדינטות של נקודת הלחיצה

void readTouch(){

                                // קריאת קואורדינטת x

pinMode(XM, OUTPUT);           // XM מוגדר כפלט ברמה נמוכה

digitalWrite(XM, LOW);

pinMode(XP, OUTPUT);           // XP מוגדר כפלט ברמה גבוהה

digitalWrite(XP, HIGH);

pinMode(YP, INPUT);            // YP מוגדר כקלט אנלוגי

x = analogRead(YP);            // התעלמות מקריאה ראשונה

x = analogRead(YP);

                                // קריאת קואורדינטת y

pinMode(YM, OUTPUT);           // YM מוגדר כפלט ברמה נמוכה

```

```
digitalWrite(YM, LOW);

pinMode(YP, OUTPUT);           // מוגדר כפלט ברמה גבוהה

digitalWrite(YP, HIGH);

pinMode(XP, INPUT);           // מוגדר כקלט אנלוגי

y = analogRead(XP);           // התעלמות מקריאה ראשונה

y = analogRead(XP);

}
```

פעולת loop

הפעולה loop פועלת על פי שיטת התשאול (polling). לפי שיטה זו מתבצעת קריאה מתמדת של הדק Y+. מצבו של הדק זה יציין אם קיימת לחיצה. לחיצה מאופיינת ברמה לוגית נמוכה. אם זוהתה לחיצה, תופעל פעולה שמטרתה לקרוא את ערכי הקואורדינטות x ו-y של נקודת הלחיצה. לאחר קריאת ערכי הקואורדינטות הן מתורגמות לערכי x ו-y המתאימים למסך שהרזולוציה שלו היא 240x320 פיקסלים.

פעולת readTouch

הפעולה מזומנת רק לאחר שזוהתה לחיצה במסך המגע. עם זימונה הפעולה מגדירה את ההדקים של ציר x על פי איור 5.16-ב ומבצעת קריאה של ערך ה-x המתאים לנקודת הלחיצה. ערך זה נשמר במשתנה x. לאחר מכן הפעולה מגדירה מחדש את ההדקים של ציר y על פי איור 5.16-א ומבצעת קריאה של ערך ה-y המתאים לנקודת הלחיצה. ערך זה נשמר במשתנה y.

דוגמה 5.5 – זיהוי לחיצה ברביע במסך התצוגה

בדוגמה זו נזהה לחיצה באחד מארבעת רביעי מסך המגע כפי שמתואר באיור 5.18. המסך מוצג ברזולוציה של 240x320. לאחר כל לחיצה תוצג הודעה מתאימה בתוכנת המוניטור.



איור 5.18: רביעי מסך המגע

בטבלה להלן מתוארים תחומי הערכים המזהים את הרביעים השונים.

רביע	Xmin	Xmax	Ymin	Ymax
1	0	160	0	120
2	160	320	0	120
3	160	320	120	240
4	0	160	120	240

להלן קוד התוכנית:

```
// מיפוי הדקי מסך המגע ללוח הפיתוח

#define XM 6
#define XP A2
#define YM 7
#define YP A1

// הערכים כפי שנקראו בפינות של מסך המגע
const int left = 125, right = 965, top = 85, bottom = 905;

// ממדי המסך ביחידות פיקסלים
const int touchHeight = 240, touchWidth = 320;

int x, y; // משתנים עבור הערך הגולמי
```

```
int xpos, ypos; // משתנים עבור הערך לאחר מיפוי הערך הגולמי לממדי המסך

void setup() {
    Serial.begin(9600);
}

void loop() {
    // הגדרת הדקי לוח הפיתוח כדי לזהות לחיצה על פי תוכנית קודמת

    pinMode(XM, OUTPUT);
    digitalWrite(XM, LOW);
    pinMode(YP, INPUT_PULLUP);
    if (digitalRead(YP) == LOW){ // בדיקה אם קיימת לחיצה
        readTouch();           // פעולה לקריאת הקואורדינטה של הלחיצה
        // מיפוי ערכי קואורדינטות גולמיים לקואורדינטות מסך בפיקסלים

        xpos = map(x, left, right, 0, touchWidth);
        ypos = map(y, top, bottom, 0, touchHeight);

        int id = identify();

        if (id != -1){
            Serial.print("quarter - ");
            Serial.println(id);
        }
        else
            Serial.println("Error");
    }
}
```

```

delay(200);
}

// הפעולה מזהה את תחום נקודת הלחיצה
int identify(){
    if (xpos >= 0 && xpos < 160 && ypos >= 0 && ypos < 120)
        return 1;
    if (xpos >= 160 && xpos < 320 && ypos >= 0 && ypos < 120)
        return 2;
    if (xpos >= 160 && xpos < 320 && ypos >= 120 && ypos < 240)
        return 3;
    if (xpos >= 0 && xpos < 160 && ypos >= 120 && ypos < 240)
        return 4;
    return -1; // error
}

// הפעולה מגדירה את ההדקים לקריאת הקואורדינטות של נקודת הלחיצה
void readTouch(){
    // read for x coordinate
    pinMode(XM, OUTPUT); // מוגדר כפלט ברמה נמוכה
    digitalWrite(XM, LOW);
    pinMode(XP, OUTPUT); // מוגדר כפלט ברמה גבוהה
    digitalWrite(XP, HIGH);
    pinMode(YP, INPUT); // מוגדר כקלט אנלוגי
    x = analogRead(YP); // התעלמות מקריאה ראשונה
    x = analogRead(YP);

```

```

// read for y coordinate
pinMode(YM, OUTPUT);           // מוגדר כפלט ברמה נמוכה
digitalWrite(YM, LOW);
pinMode(YP, OUTPUT);           // מוגדר כפלט ברמה גבוהה
digitalWrite(YP, HIGH);
pinMode(XP, INPUT);            // מוגדר כקלט אנלוגי
y = analogRead(XP);             // התעלמות מקריאה ראשונה
y = analogRead(XP);
}

```

הסבר התוכנית

התוכנית היא שיפור של התוכנית הקודמת. לפעולה loop נוסף קטע המזמן את הפעולה identify המחזירה את מספר הרביע שבו זוהתה הלחיצה. הערך שהוחזר מוצג בתוכנת המוניטור אם הוא מייצג קריאה נכונה של קואורדינטות הלחיצה.

פעולת identify

הפעולה בודקת את תחומי ערכי x ו-y בהתאם לערכים המוגדרים בטבלה. אם התקבלו ערכים שגויים תחזיר הפעולה ערך שלילי המציין שגיאה. למשל ההוראה:

```

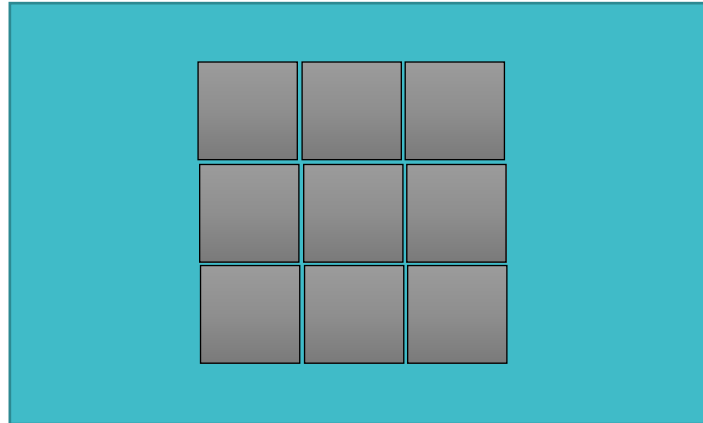
if (xpos > 0 && xpos < 160 && ypos > 0 && ypos < 120)
    return 1;

```

מזהה את רביע 1

שאלה 5.17

תלמיד תכנן ליצור משחק איקס-עיגול. הוא ביקש לחלק את המסך כך שבמרכז יוצגו 9 ריבועים. איור 5.19 מתאר את מבנה המסך הנדרש. גודלו של המסך הוא 240×320 .



איור 5.19: מבנה חלוקת המסך

- א. קבעו את ערכי הגבולות של כל אחת מהמשבצות המופיעות במרכז מסך המגע. הציגו באמצעות טבלה.
- ב. כתבו פעולה המזהה את המשבצת שבה זוהתה הלחיצה. הפעולה תחזיר ערך מספרי בין 1 לבין 9 אם זיהוי תקין, וערך שלילי אם הזיהוי שגוי.
- ג. שכתבו את התוכנית מדוגמה 5.5 כך שתציג את מספר המשבצת בה בוצעה לחיצה.

סיכום פרק 5

בפרק 5 למדתם את הנושאים האלה:

1. ארכיטקטורה של SPI
2. התקני שליט (Master) והתקני עבד (Slave)
3. אותות בערוץ התקשורת
4. מפתח התקשורת במיקרו-בקר
5. מחלקת SPISettings
6. מחלקת SPI
7. דוחף תצוגה MAX7219
8. מסך גרפי B12864
9. אופן פעולה להצגת טקסט
10. אופן פעולה להצגת גרפיקה
11. הצגת תמונה במסך גרפי
12. שימוש בתוכנת עזר LCDAssistant לקבלת קוד בינרי של תמונה
13. מסך מגע התנגדותי
14. זיהוי לחיצה במסך מגע
15. זיהוי מיקום הלחיצה במסך מגע

פרק 6: רכיבי חומרה

בפרקים הקודמים הכרנו את יסודות השימוש בלוח הפיתוח:

- חיבור חיישנים ספרתיים, כגון חיישן PIR, וחיישנים תקביליים, כגון חיישן LM35.
- הפעלה של התקני הפעלה ספרתיים, כגון רכיבי תצוגה, והתקני הפעלה תקביליים.
- חיבור של התקנים באמצעות תקשורת טורית מסוגים שונים:
 - תקשורת טורית UART (חיבור בין מחשב ללוח הפיתוח)
 - תקשורת טורית I2C (חיבור התקנים ללוח הפיתוח)
 - תקשורת טורית SPI (חיבור התקנים ללוח הפיתוח)

בפרק זה נציג חיבור של התקנים מתקדמים אל לוח פתוח המאפשרים חיבור אל העולם באמצעות רשת מקומית או רשת האינטרנט, וכן עיבוד מידע ויזואלי המתקבל ממצלמה. החיבור אל ההתקנים המתקדמים מתבצע באמצעות אחת משיטת התקשורת הטורית SPI או I2C.

6.1 תקשורת מחשבים

תקשורת מחשבים או תקשורת נתונים, הוא תחום העוסק בהעברה של נתונים בין שני מחשבים או יותר. בכל מערכת תקשורת צד אחד שולח מידע לצד אחר המקבל את המידע. השולח נקרא גם מקור המידע, והמקבל – יעד המידע. לרוב, התקשורת היא דו-כיוונית – כל צד גם שולח מידע וגם מקבל מידע.

כאשר הצד השולח רוצה להעביר מידע עליו לתרגם את המידע לאות המסוגל לנוע או להתפשט בתווך. תרגום המידע לאות נעשה באמצעות משדר הממיר את המידע לאות ומשדר אותו לתווך. האות מתפשט בתווך ומגיע ליעד. המקלט קולט את האות בהגיעו ליעד וממיר אותו למידע.

רשת מחשבים היא אוסף של מחשבים עצמאיים המקושרים ביניהם. רשת התקשורת מספקת למשתמשים שירותי העברת מידע מקצה אחד של הרשת לקצה השני. התוכנה של רשת התקשורת מורכבת ביותר. נהוג לחלק אותה לשכבות אחדות שכל אחת מהן מספקת שירותי תקשורת לשכבה שמעליה ומתקשרת עם השכבה העמיתה במחשב המרוחק. יישומי התקשורת שייכים לשכבה העליונה שנקראת שכבת היישום. תוכנת התקשורת ממומשת לפי פרוטוקול תקשורת.

פרוטוקול הוא אוסף של כללים שיחד יוצרים שפה משותפת או נוהל התקשרות בין הצדדים המעורבים בתקשורת. תפקידו להסדיר את העברת המידע בין הצדדים.

רשתות התקשורת נחלקות לשני סוגים עיקריים:

- **רשתות מקומיות – LAN: Local Area Networks**
 - רשתות אלה משתייכות לארגון אחד ומוגבלות לקילומטרים ספורים.
- **רשתות ארוכות טווח – WAN: Wide Area Networks**
 - רשתות אלה מתפרשות על פני אזורים נרחבים, למשל מדינות או יבשות.

כדי שמחשב יוכל להתחבר לרשת מקומית יש להוסיף לו כרטיס מיוחד שנקרא **כרטיס רשת**. לכל סוג רשת קיים סוג אחר של כרטיס רשת. הכרטיס מתחבר ללוח של המחשב ומכיל מחבר המאפשר חיבור לכבל התקשורת. כרטיס התקשורת מספק ממשק בין המחשב לרשת. כל המידע שזורם ברשת מועבר דרך **ערוץ תקשורת**. הערוץ יכול להיות זוג חוטי נחושת, כבל, סיב אופטי וכדומה, ואף ערוץ אלחוטי.

ציוד תקשורת דרוש להגברת אותות ולקישור בין רכיבים של הרשת או בין הרשת לבין רשת חיצונית אחרת.

נהוג להבחין בין שתי טכנולוגיות שידור:

- **רשתות הפצה**
 - לרשת זו יש ערוץ הפצה אחד המשותף לכל המחשבים ברשת. כאשר מחשב ברשת שולח הודעה היא מגיעה לכל המחשבים ברשת. רק המחשב שההודעה מיועדת אליו יקרא אותה ויטפל בה. רשת אתרנט Ethernet היא דוגמה לרשת מקומית המבוססת על ערוץ הפצה.

- **רשתות מיתוג**
 - רשת זו מבוססת על ערוצי נקודה לנקודה – נל"ן. ערוץ נל"ן מחבר ישירות בין שני מחשבים בלבד.

רשתות מקומיות כוללות מרכזיות מיתוג, מכשור אלקטרוני המאפשר העברת מידע בין היחידות באמצעות מתגים אלקטרוניים המחברים אותן. המרכזיות הופכות את הרשתות המקומיות לרשתות מיתוג הנעשה באופן ריכוזי.

כדי להקטין את הסיבוך בבנייה ובתחזוקה של מערכות תקשורת, נהוג לחלק אותן לשכבות. לכל שכבה תפקיד מסוים במערכת התקשורת, והיא מספקת שירותי תקשורת לשכבות.

מעליה. כל מערכת קצה צריכה לממש את כל השכבות שמהן מורכבת מערכת התקשורת. כל שכבה במחשב מנהלת שיחה עם השכבה העמיתה במחשב האחר בפרוטוקול המתאים.

אלק

במודל של המערכת יש 5 שכבות תקשורת:

- שכבת היישום
 - שכבת התובלה
 - שכבת הרשת
 - שכבת הערוץ
 - השכבה הפיזית
- מטפלת בתקשורת בין תהליכי היישום
- מטפלת בתקשורת בין מערכות הקצה
- מטפלת בהעברת מנות ברשת
- מטפלת בהעברת נתונים בערוץ
- מטפלת בשידור וקליטה של אותות פיזיים

רשת האינטרנט מחברת קבוצה גדולה של רשתות פרטיות או ציבוריות לרשת עולמית אחת. כל אחת מהרשתות עשויה להיות שונה מהרשתות האחרות מבחינת התקני החומרה והתוכנה. התוכנה של האינטרנט צריכה לאפשר העברת הודעות בין הרשתות הללו גם כאשר אינן תואמות.

רשת האינטרנט היא רשת מיתוג ובנויה מרשת של נתבים. נתב הוא צומת מיתוג שיכול לחבר בין כמה רשתות שאינן בהכרח תואמות. הנתב הוא מחשב המסוגל לטפל בהעברה גדולה של נתונים. הנתב בוחן כל מנת נתונים כדי להחליט לאיזה מערוצי היציאה לשלוח אותה.

פרוטוקול האינטרנט IP (Internet Protocol) מחבר רשתות שונות לרשת אחת. בפרוטוקול זה נקבע כיצד מחליטים על כתובות למחשבים ברשת. לכל חיבור ברשת יש כתובת ייחודית שנקראת כתובת IP. יחידות הנתונים של הפרוטוקול נקראות IP-datagram.

הפרוטוקול מכניס את הנתונים שהתקבלו ליחידה IP-datagram הכוללת גם כתובת IP ושולח את הנתונים ליעדם. הפרוטוקול שייך לשכבת הרשת (שכבה 3).

כתובת IP

כתובת IP היא מספר מזהה בן 32 סיביות. כל מנה הנשלחת באינטרנט מכילה שדה באורך 32 סיביות של כתובת המקור ושדה באורך 32 סיביות של כתובת היעד. המחשב שמסדר את הנתונים חייב לדעת את כתובת ה-IP של המחשב שאליו מיועדים הנתונים.

כדי להקל על המשתמשים בהכנסת הכתובת משתמשים בייצוג עשרוני. בייצוג זה מייצגים כל בית (8 סיביות) באמצעות ערך עשרוני מ-0 עד 255, ומשתמשים בנקודות כדי להפריד בין 4 הבתים (סה"כ 32 סיביות).

דוגמה לכתובת

ייצוג עשרוני	ייצוג בינארי			
192.168.1.100	11000000	10101000	00000001	01100100

פרוטוקול IP אינו מבטיח אמינות. ייתכן שמנות יאבדו ברשת מסיבות שונות. לעומת זאת הפרוטוקול **TCP** (Transmission Control Protocol) מבטיח אמינות. הפרוטוקול מבטיח שכל מנה תגיע ליעדה. הפרוטוקול בודק את המנות שמתקבלות, מסדר אותן לפי הסדר ומבטיח שכל פנה תגיע בדיוק פעם אחת ללא כפילות. פרוטוקול זה שייך לשכבת התעבורה (שכבה 4).

מתייחסים לשני פרוטוקולים אלו בשם הכולל **TCP/IP**.

יישומי רשת מורכבים משני תהליכים הרצים בשני מחשבים שונים. למשל היישום Web מורכב מתהליך לקוח (דפדפן) הרץ במחשב המשתמש ומתהליך שרת שרץ במחשב אחר. התהליכים רצים בו-זמנית ומתקשרים ביניהם על ידי החלפת הודעות, לפי פרוטוקול ציבורי ידוע, באמצעות רשת התקשורת.

פרוטוקולים פרטיים מפותחים על ידי חברות או על ידי אנשים פרטיים. במקרה כזה מי שמפתחים את היישום קובעים את הפרוטוקול לפי ראות עיניהם. לאחר מכן, הם מתכננים את השרת והלקוח כך שיפעלו בהתאם לפרוטוקול.

התהליכים שמרכיבים יישום רשת שולחים ומקבלים הודעות דרך **שקעים** (Sockets). שקע זה נקרא גם **API** (Application Programmer Interface), כלומר ממשק למתכנתי יישומים. תהליך של שכבת היישום שולח מידע, מעביר את ההודעה לשקע ונסמך על קיומה של מערכת תובלה. כאשר הודעה מגיעה למערכת התובלה, המערכת מעבירה את ההודעה לשקע של תהליך היישום, שהוא יעד המידע. מערכת התובלה הזו נתמכת בתשתית רשת התקשורת, כלומר בשכבות הנמוכות יותר של הרשת.

השקע מאפשר למתכנתי היישומים קבוצה של פעולות שבאמצעותן ניתן לגשת לשירותי הרשת באופן כללי, ולשירותי שכבת התובלה באופן ספציפי.

פְּתָחָה (Port)

פתחה הוא תהליך מסוים שדרכו יכולות תוכנות להעביר נתונים. התהליך מתבצע בפרוטוקולים של שכבת התעבורה: TCP ו-UDP. לכל פתחה יש מספר מזהה בן 16 סיביות. בפרוטוקול TCP ניתן להשתמש ב-65535 כתובות שונות. הכתובת נקראת מספר הפורט.

מרחב הכתובות של הפורטים מחולק ל-3 קבוצות:

- 1023–0 פורטים מוכרים
- 49151–1024 פורטים רשומים
- 65535–49152 פורטים פרטיים או דינמיים

דוגמאות לפורטים של פרוטוקולים נפוצים

21 – פרוטוקול העברת קבצים (FTP)

25 – פרוטוקול דואר יוצא (SMTP)

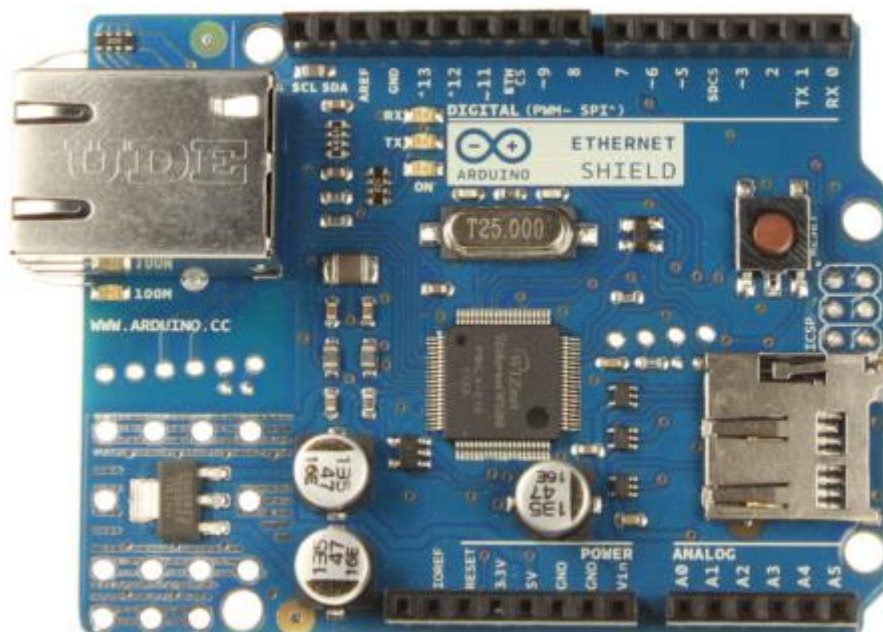
53 – פרוטוקול DNS

80 – פרוטוקול העברת דפי אינטרנט HTTP

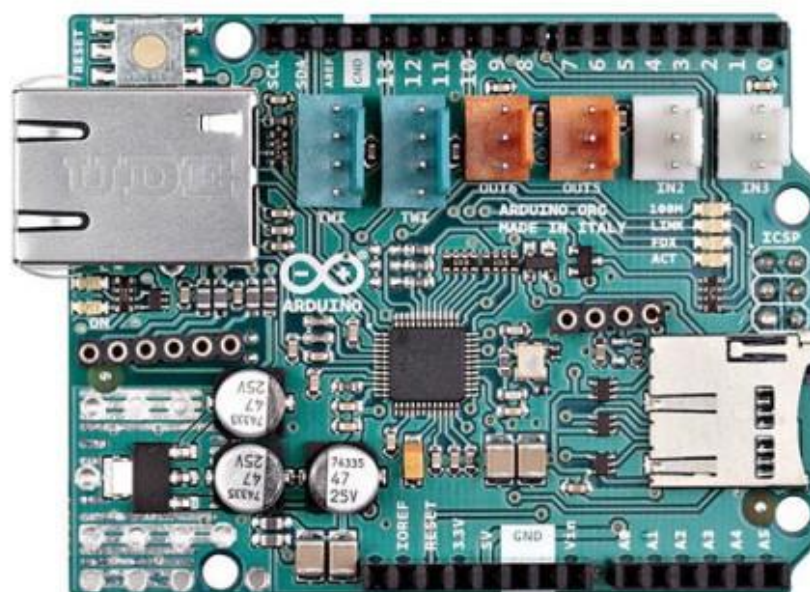
443 – פרוטוקול העברת דפי אינטרנט מאובטחים HTTPS

Ethernet Shield 6.2

כרטיס הרחבה Ethernet Shield הוא כרטיס רשת המחובר אל המיקרו בקר ואחראי על התקשורת. יש שתי גרסאות של כרטיס ההרחבה Ethernet Shield: גרסה 1 וגרסה 2. שתי הגרסאות עדיין קיימות למכירה, אבל Arduino הגדירו אותן ככרטיסים שיצאו מהמחזור (Retired) ומציעים במקומן כרטיסים אחרים.



איור 6.1: כרטיס הרחבה גרסה 1



איור 6.2: כרטיס הרחבה גרסה 2

לשתי הגרסאות של כרטיס ההרחבה יש תכונות דומות:

- נדרש שימוש בלוח הפיתוח
- מתח ההפעלה הוא 5V, והוא מסופק על ידי לוח הפיתוח
- מהירויות החיבור היא 10/100Mb
- החיבור הוא באמצעות תקשורת SPI

גרסה 1 מבוססת על בקר תקשורת W5100 עם חוצץ זיכרון בגודל 16KB ואילו גרסה 2 מבוססת על בקר תקשורת W5500 עם חוצץ זיכרון בגודל 32KB.

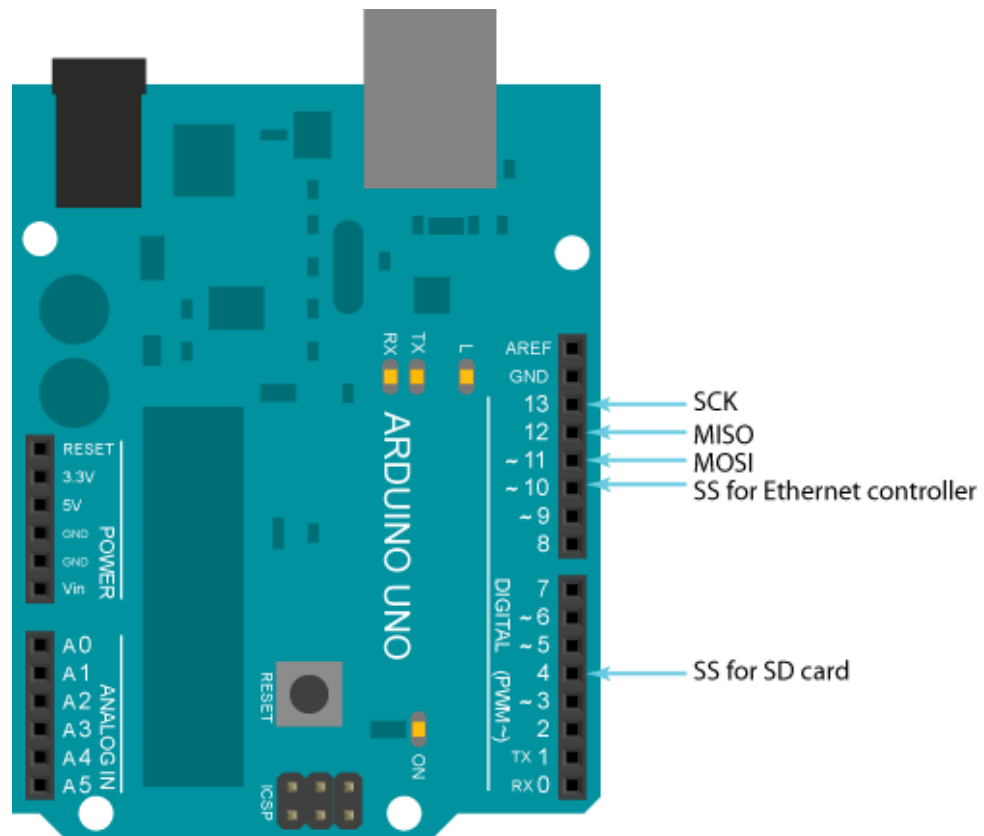
שני בקרי התקשורת מספקים יכולות תמיכה בפרוטוקול IP עבור שני פרוטוקולי תעבורה: TCP ו-UDP. גרסה 1 תומכת בחיבור של עד 4 שקעים (Sockets) בו-זמנית וגרסה 2 תומכת בחיבור של עד 8 שקעים בו-זמנית. לשני כרטיסי ההרחבה יש מחבר (Connector) תקני RJ-45.

שני כרטיסי ההרחבה כוללים חריץ הרחבה לכרטיסי micro-SD היכולים לשמש מקום אחסון לקבצים עבור התקשורת ברשת. הגישה לקורא כרטיסי ה-SD מתבצעת באמצעות מחלקת (ספרייה) SD. בעת שימוש במחלקה זו SS ממופה להדק 4 בלוח הפיתוח.

לשני כרטיסי ההרחבה קיים מנגנון איפוס המבטיח פעולה תקינה של בקר התקשורת.

בלוח הפיתוח משתמשים בתקשורת SPI בשני התקנים: כרטיס התקשורת W5500 (או W5100) וכרטיס הזיכרון micro-SD. ההדקים 10, 11, 12 ו-13 בלוח הפיתוח משמשים את תקשורת ה-SPI. הדק 10 משמש כהדק הבחירה SS עבור בקר התקשורת, והדק 4 משמש כהדק הבחירה SS עבור רכיב הזיכרון.

מכיוון שגם רכיב הזיכרון וגם בקר התקשורת חולקים את ערוץ התקשורת SPI, רק אחד מהם יוכל להשתמש בערוץ התקשורת בכל פעם. אם משתמשים בשני ההתקנים, המחלקות המתאימות ישלטו בהדקי הבחירה לפי הצורך. אם לא משתמשים בהתקן מסוים, חייבים לקבוע את ערכו של ההדק למצב לא פעיל. במקרה זה עבור התקן התקשורת יש לקבוע את הדק 10 כיציאה ברמה גבוהה, ועבור התקן הזיכרון יש לקבוע את הדק 4 כיצירה ברמה גבוהה.



איור 6.3: הדקי לוח הפיתוח UNO המשמשים את כרטיס הרחבה

מאפייני כרטיס ההרחבה

- לחצן האיפוס מאפס גם את לוח הפיתוח וגם את בקר התקשורת.
- נוריות חיווי
 - ON – ציון מצב אספקת מתח לכרטיס ההרחבה
 - LINK – מציין חיבור לרשת. הבהוב מציין קיום תקשורת
 - FDX – תקשורת במצב דו-כיווני מלא (Full Duplex)
 - 100M – תקשורת בקצב של 100Mb/s (אחרת בקצב 10Mb/s)
 - ACT – בהבהוב מציין פעילות בקווי RX ו-TX

6.3 מחלקות תמיכה בתקשורת מחשבים

מחלקות הליבה של סביבת הפיתוח של Arduino כוללות מחלקות התומכות בתקשורת מחשבים, בזרימת מידע (Stream) ובמודל של תכנות לחישוב מבוזר שרת-לקוח (Client-Server).

מחלקת IPAddress

המחלקה נועדה להקל על אחסון והעברת כתובת IP לפי הצורך כיחידה אחת. מידע הכתובת מאוחסן במערך בגודל 4 בתים שכל אחד מהם מכיל מספר חיובי בין 0 לבין 255 או כמספר ארוך חיובי בגודל 32 סיביות.

ממשק חלקי של המחלקה

פעולה	תיאור הפעולה
<code>IPAddress()</code>	פעולת הבנייה הקובעת את הכתובת ל-0.0.0.0
<code>IPAddress(byte1, byte2, byte3, byte4)</code>	פעולת הבנייה הקובעת את הכתובת ל-byte1.byte2.byte3.byte4
<code>IPAddress(bytes[])</code>	פעולת בנייה המקבלת כתובת כמערך של בתים. הנחה: הכתובת תקינה.
<code>bool fromString(char[] address)</code>	הפעולה מקבלת מחרוזת של הכתובת. הפעולה תמיר את המחרוזת למערך בן 4 בתים. הפעולה תחזיר true אם ההמרה הצליחה, ו-false אם ההמרה לא הצליחה.

המחלקה כוללת גם פעולת השוואה לוגית (==) בין שתי כתובת של המחלקה IPAddress, וכן פעולה פרטית `raw_address()` הזמינה לחברי המחלקה בלבד ומחזירה הפנייה למערך של הכתובת. המחלקה מוגדרת כמחלקת ליבה ועל כן אין צורך להוסיף שורת הכללה של מחלקה זו.

שתי מחלקות ליבה נוספות Client.h ו-Server.h מגדירות את אוסף הפעולות⁷ שחייבים לממש אם רוצים להשתמש במודל שרת-לקוח.

המחלקה Stream היא מחלקת ליבה הכוללת בתוכה אוסף פעולות של מידע הנקלט או משודר באופן תווי. למשל המחלקה Serial שהכרנו בפרקים הקודמים משתמשת בפעולות של מחלקה זו. השימוש במחלקה מאפשר אחידות בהתייחסות לזרימת המידע ללא קשר לסוג התקשורת.

מחלקת Ethernet / Ethernet2

המחלקות מוגדרות בקובץ בעל סיומת h. הן נועדו לעבוד עם כרטיסי ההרחבה Ethernet Shield V1 (Ethernet.h) ו-Ethernet Shield V2 (Ethernet2.h). המחלקות מאפשרות חיבור אל רשת האינטרנט. לוח הפיתוח יכול לשמש כשרת (Server) המקבל חיבורים נכנסים (מאזין) או כלקוח (Client) המתחבר אל שרת. ממשק שתי המחלקות זהה לחלוטין פרט לכך שהמחלקה Ethernet.h תומכת בבקר התקשורת W5100 ואילו המחלקה Ethernet2.h תומכת בבקר התקשורת W5500. כל מחלקה תומכת ב-4 חיבורים בו-זמנית בלבד.

ממשק חלקי של המחלקה

פעולת begin
<pre>void begin (mac, ip, dns, gateway, subnet) void begin (mac, ip, dns, gateway) void begin (mac, ip, dns) void begin (mac, ip) int begin (mac)</pre>
תיאור הפעולה
הפעולה מאתחלת את המחלקה וקובעת את פרמטרי התקשורת. • mac – מערך בגודל 6 בתים המשמש לציון כתובת ייחודית של כרטיס הרשת. הכתובת נקבעת על ידי יצרן הכרטיס. • ip – כתובת לוגית של כרטיס הרשת. • dns – כתובת ip של שרת dns האחראי לתרגום שמות מתחם לכתובות ip. • gateway – כתובת ip של השער דרכו מתבצעת העברת המידע אל הרשת.

⁷ הפעולות במחלקה מוגדרות ברמת הכותרת בלבד. מחלקות אחרות המשתמשות באותה המחלקה (היורשות שלה) חייבות לממש את הפעולות. ההגדרה המקבילה בשפת C# נקראת ממשק Interface.

- **subnet** – מספר המציין את כתובת תת-הרשת. למספר תבנית דומה לכתובת ip.

הערות:

כאשר מספקים את הפרמטר mac בלבד, כל שאר הפרמטרים מתקבלים משרת DHCP מקומי.⁸ השימוש מגדיל באופן משמעותי את נפח התוכנית. הפעולה מחזירה ערך מספרי שלם 1 המציין הצלחה או 0 כישלון בהתחברות. כאשר מספקים שני פרמטרים mac ו-ip הפרמטרים האחרים נגזרים על פי כתובת ה-ip שסופקה. אם ip=192.168.1.100 אז gateway = 192.168.1.1 dns = וכתובת תת-הרשת sub = 255.255.255.0.

פעולות נוספות

פעולה	תיאור הפעולה
IPAddress localIP()	הפעולה מחזירה עצם מסוג כתובת IP של הכתובת המוגדרת בכרטיס הרשת.
IPAddress subnetMask()	הפעולה מחזירה עצם מסוג כתובת IP של כתובת מסכת הרשת.
IPAddress gatewayIP ()	הפעולה מחזירה עצם מסוג כתובת IP של השער.
IPAddress dnsServerIP ()	הפעולה מחזירה עצם מסוג כתובת IP של שרת dns.
int maintain()	הפעולה מנהלת את הקצאת פרמטרי התקשורת באופן אוטומטי. אם משתמשים בהקצאה אוטומטית, נדרש להשתמש בפעולה זו באופן מחזורי כדי להבטיח חידוש הקצאת הפרמטרים.

⁸ DHCP (Dynamic Host Configuration Protocol) יחידה המספקת כתובות באופן אוטומטי.

דוגמה 6.1 – יצירת חיבור תקשורת באמצעות שרת DHCP

בדוגמה זו נתחבר לרשת מקומית ונקבל את פרמטרי התקשורת באופן אוטומטי.

```
#include <SPI.h>                // SPI תמיכה בתקשורת
#include <Ethernet.h>           // מחלקת תמיכה ברשתות תקשורת
byte mac[] = {                  // הגדרת מספר מזהה פיזי לכרטיס הרשת
  0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
void setup() {
  Serial.begin(9600);           // אתחול תקשורת טורית למוניטור
                                // הגדרות פרמטרי התקשורת באופן אוטומטי
  if (Ethernet.begin(mac) == 0) { // האם החיבור נכשל
    Serial.println("Failed to configure Ethernet using DHCP");
    // no point in carrying on, so do nothing forevermore:
    for(;;)                     // לולאה אינסופית במקרה של כישלון
      ;
  }
  Serial.println(Ethernet.localIP()); // הצגת הכתובת שהוקצתה באופן אוטומטי
}
void loop() {
}
```

לאחר ביצוע הידור נקבל את הסיכום הזה:

```
Done uploading.
Sketch uses 11754 bytes (36%) of program storage space.
Global variables use 589 bytes (28%) of dynamic memory,
```

שימו לב לכמות הזיכרון שבשימוש!

לאחר הרצה נקבל את ההודעה הזאת:

```
address printed on a sta
COM7 (Arduino/Genuino Uno)
192.168.0.101
```

דוגמה 6.2 – יצירת חיבור תקשורת ללא שרת DHCP

בדוגמה זו נתחבר לרשת מקומית ונקבע את פרמטרי התקשורת באופן ידני.

```
#include <SPI.h> // מחלקת תמיכה בתקשורת SPI
#include <Ethernet.h> // מחלקת תמיכה ברשתות תקשורת
byte mac[] = { // הגדרת מספר מזהה פיזי לכרטיס הרשת
  0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
IPAddress ip(192,168,0,201); // קביעת כתובת IP
void setup() {
  Serial.begin(9600);
  Ethernet.begin(mac,ip); // קביעת כתובת באופן מפורש
  Serial.print("IP = ");
  Serial.println(Ethernet.localIP());
  Serial.print("Subnet Mask = ");
  Serial.println(Ethernet.subnetMask());
```

```

Serial.print("Gateway = ");

Serial.println(Ethernet.gatewayIP());

Serial.print("DNS Server = ");

Serial.println(Ethernet.dnsServerIP ());

}

void loop() {

}

```

לאחר ביצוע הידור נקבל את הסיכום הזה:

Done uploading.
 Sketch uses 4760 bytes (14%) of program storage space.
 Global variables use 331 bytes (16%) of dynamic memory,

שימו לב שכמות הזיכרון בשימוש פחתה באופן משמעותי.

לאחר הרצה נקבל את ההודעה הזאת:

COM7 (Arduino/Genuino Uno)

IP = 192.168.0.201
 Subnet Mask = 255.255.255.0
 Gateway = 192.168.0.1
 DNS Server = 192.168.0.1

מחלקת EthernetClient

מחלקה זו מטפלת בכל תהליכי ההקמה והניהול של לקוח אינטרנט.

ממשק חלקי של המחלקה

פעולה	תיאור הפעולה
EthernetClient()	פעולת בנייה היוצרת עצם של לקוח.
int connect(ip, port)	הפעולה גורמת לחיבור לשרת בכתובת ip ובפורט נתונים. ip – כתובת ip המוגדרת כעצם מסוג IPAddress.
int connect(URL, port)	URL – מחרוזת המייצגת את שם המתחם של השרת שהלקוח רוצה להתחבר אליו. port – כתובת המפתח של השרת שהלקוח רוצה להתחבר אליו. הפעולה מחזירה קוד המייצג את תוצאת הפעולה: (1) הפעולה הצליחה (0) הפעולה לא הצליחה
bool connected()	הפעולה מחזירה true אם הלקוח מחובר, ו-false אם הלקוח אינו מחובר. הפעולה תחזיר true גם אם הלקוח כבר אינו מחובר אולם יש נתונים שעוד לא נקראו.
void stop()	הפעולה מנתקת את הלקוח מהשרת.
int available()	הפעולה מחזירה את מספר הבתים הזמינים לקריאה.
int read()	הפעולה מחזירה את הבית הבא שנקלט מהשרת. אם אין מידע זמין יוחזר הערך -1.
int write(val) int write(buf, len)	הפעולה כותבת נתונים לשרת שהלקוח מחובר אליו. הנתונים משודרים כרצף בתים. val – שידור נתון כבית או כתו. buf – מערך של בתים או תווים המיועדים לשידור. len – אורך הבתים במערך. הפעולה מחזירה את מספר הבתים ששודרו.
int print(val) int println(val, format)	שולחת את val כמידע טקסטואלי דרך ערוץ התקשורת: - ערך שלם יוצג כמספר שלם. - ערך ממשי יוצג בדיוק של שתי ספרות אחרי הנקודה. - תו או מספר כבית יוצג כתו. - מחרוזת תוצג כמחרוזת.

הפרמטר **format** קובע את אופן ההצגה של ערכים מספריים לפי בסיסים שונים או קביעת מספר הספרות אחרי הנקודה.
הפעולה **print** מציגה את המידע בהמשך השורה וההוראה
println מציגה את המידע ועוברת לשורה חדשה.
הפעולה מחזירה את מספר הבתים שנכתבו.

מחלקת EthernetServer

מחלקה זו מטפלת בכל תהליכי ההקמה והניהול של שרת אינטרנט. המחלקה מאזינה להתקשרויות ממחשבי לקוח ומסוגלת לטפל ב-4 חיבורים בו-זמנית.

ממשק חלקי של המחלקה

פעולה	תיאור הפעולה
EthernetServer(port)	פעולת בנייה היוצרת עצם של שרת המאזין לפורט המועבר כפרמטר לפעולה. port – כתובת הפורט בו השרת מאזין.
void begin()	הפעולה גורמת לשרת להתחיל להאזין לחיבורים נכנסים.
EthernetClient available()	הפעולה מחזירה עצם EthernetClient המייצג לקוח המחובר לשרת ויש לו מידע זמין לקריאה. החיבור עם הלקוח נשמר עד לסגירת החיבור.
int write(val) int write(buf, len)	הפעולה כותבת נתונים אל הלקוח המחובר לשרת. הנתונים משודרים כרצף בתים. val – שידור נתון כבית או כתו. buf – מערך של בתים אות תווים המיועדים לשידור. len – אורך הבתים במערך. הפעולה מחזירה את מספר הבתים ששודרו.

פרוטוקול HTTP

פרוטוקול HTTP (Hypertext Transfer Protocol) נועד לאפשר תקשורת בין לקוח לשרת. הפרוטוקול פועל בשיטה של בקשה ותגובה (request-response) בין לקוח ושרת. דפדפן רשת (web browser) משמש כלקוח, ויישום במחשב המארח את האתר (web site) משמש כשרת.

תבנית בקשה – Request

- שורת הבקשה
- כותרות נוספות, אם יש צורך בהן, המסתיימות ב-CRLF ("r\n")
- שורה ריקה – שורה המסתיימת ב-CRLF המציינת את סוף הכותרות
- גוף ההודעה (אופציונאלי)

בקשת GET

זוהי אחת מהפעולות הנפוצות בפרוטוקול, והיא משמשת לבקשת מידע ממשאב (resource) מסוים באתר.

תבנית הפעולה:

GET /name?value pairs HTTP/ver

name – שם המשאב

value pairs – זוג של שם וערך

? – הפרדה בין שם המשאב לזוגות הערכים

HTTP/ver – ציון סוף הבקשה וכן הגרסה.

דוגמה

/index.html?a=1&b=2

המשאב הוא דף אינטרנט בשם index.html. שני זוגות הערכים הם (a, 1) ו-(b, 2). זוגות הערכים מופרדים באמצעות הסימן &.

בעת השימוש בדפדפן, בקשת ה-GET נוצרת באפן אוטומטי, ותוכן הבקשה מתקבל משורת ה-URL. למשל `http://www.mySite.co.il/index.html?a=1&a=2` ייצור את בקשת ה-GET שהוצגה בדוגמה.

תבנית תגובה – Response

לאחר קבלת הבקשה שולח השרת תגובה הכוללת:

- שורת המצב
- כתורות נוספות, אם יש צורך בהן, המסתיימות ב-CRLF ("\\n\\r")
- שורה ריקה – שורה המסתיימת ב-CRLF המציינת את סוף הכותרות
- גוף ההודעה (אופציונאלי)

שורת המצב כוללת את גרסת הפרוטוקול, מספר המציין קוד של המצב וביטויי טקסט קשורים. כל חלק מופרד ברווח.

תיאור חלקי של קודי מצב

קוד	תיאור
200 OK	הבקשה תקינה.
400 Bad Request	הבקשה לא יכולה להתקבל בגלל תחביר שגוי.
404 Not Found	הדף שהתבקש לא נמצא, ייתכן שיהיה זמין בעתיד.

לדוגמה

HTTP/1.1 200 OK	- שורת מצב
Content-Type: text/html	- כותרת סוג המסמך
	- שורה ריקה
<html> Hello, World </html>	- דף אינטרנט

יתרון השימוש בפרוטוקול

היתרון בשימוש בפרוטוקול זה בפרויקטים מבוססי לוח פיתוח UNO ודומיו המשמשים כשרתי אינטרנט הוא באוניברסליות של חיבור לקוחות באמצעות דפדפנים הזמינים במחשבים אישיים,

מחשבי לוח (טאבלטים) ובטלפונים חכמים (סמארטפונים). הגישה לשרת באמצעות דפדפנים מייתרת את הצורך בכתיבת קוד ללקוחות.

חסרון השימוש בפרוטוקול

החסרון הבולט של הפרוטוקול הוא אבטחת המידע. הגישה לשרת פתוחה ואינה כוללת מגבלות גישה והצפנת מידע. נושא אבטחת המידע חורג מתחום הידע הנדרש בתוכנית הלימודים, ולכן לא נעסוק בו.

דוגמה 6.3 – מדידת טמפרטורה מרחוק

בדוגמה זו ניצור שרת אינטרנט HTTP המאזין לחיבור של לקוחות דפדפן אינטרנט (Internet Browser) בפורט 80. עם התקשרות לקוח יישלח ערך של חיישן טמפרטורה LM36 המחובר לכניסה האנלוגית A0 בלוח הפתוח.

בדוגמה זו נתבסס על הדוגמה 2.1 מפרק 2.

```
#include <SPI.h>

#include <Ethernet.h>

byte mac[] = {                                // כתובת פיזית של כרטיס הרשת
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};

IPAddress ip(192, 168, 0, 201); // קביעת כתובת סטטית לשרת

EthernetServer server(80); // יצירת עצם שרת עם האזנה לפורט 80

void setup() {
  // פתיחת ערוץ תקשורת טורית ותקשורת אתרנט. אם לא קיים מתאם מתאם תקשורת
  אתרנט, התוכנית נעצרת

  Serial.begin(9600);

  Ethernet.begin(mac, ip); // אתחול כרטיס הרשת עם הכתובת הפיזית
  והלוגית
```

```

// בדיקת הימצאות כרטיס ההרחבה של הרשת
if (Ethernet.hardwareStatus() == EthernetNoHardware) {
    Serial.println("Ethernet shield was not found.\nSorry, can't run without
hardware. :(");
    while (true) {
        // לולאה אינסופית לעצירת התוכנית
    }
}

server.begin();
// אתחול השרת בפורט 80

Serial.print("server is at ");
Serial.println(Ethernet.localIP());
}

void loop() {
    // האזנה לחיבורים נכנסים

    EthernetClient client = server.available(); // קבלת עצם מסוג לקוח אם הוא
זמין

    if (client) {
        // אם הלקוח זמין

        Serial.println("new client");

        // כל בקשה בפרוטוקול זה מסתיימת בשורה ריקה

        boolean currentLineIsBlank = true;

        while (client.connected()) {
            // לולאת בדיקה אם לקוח מחובר

            if (client.available()) {
                // בדיקה אם קיים נתון זמין

                char c = client.read();
                // קריאת תו בודד

                Serial.write(c);
                // הצגת התו שנקלט

                // בדיקה אם התקבל תו סוף שורה והשורה הייתה ריקה

                if (c == '\n' && currentLineIsBlank) {

```

```

// שידור כותרת תגובה סטנדרטית
client.println("HTTP/1.1 200 OK");

client.println("Content-Type: text/html");

client.println(); // שידור שורה ריקה – סוף תגובה

client.println("<!DOCTYPE HTML>");

client.println("<html>"); // תגית המגדירה את תחילת דף האינטרנט

// חישוב ערך הטמפרטורה הנמדדת

int binValue = analogRead(A0);

float voltage = 0.00488 * binValue;

float temperature = voltage / 0.01 - 50;

// שידור ערך הטמפרטורה ללקוח

client.print("The temperature is ");

client.print(temperature);

client.println("<br />");

client.println("</html>"); // תגית סיום דף האינטרנט

break; // יציאה מתוך לולאת בדיקת לקוח מחובר
}

if (c == '\n') {

// תחילת שורה חדשה

currentLineIsBlank = true;

} else if (c != '\r') {

// אם התו שונה מחזרה לתחילת שורה אז השורה אינה ריקה

currentLineIsBlank = false;

}

```

```

    }
}

delay(1);                // המתנה זמן לשליחה

client.stop();           // סגירת התקשורת מול הלקוח

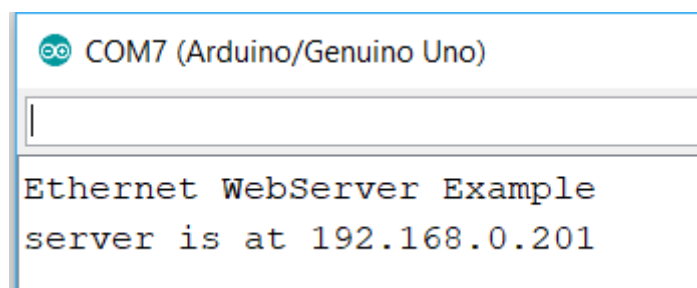
Serial.println("client disconnected");

}

}

```

לאחר הרצת התוכנית נקבל את ההודעה הזאת:



```

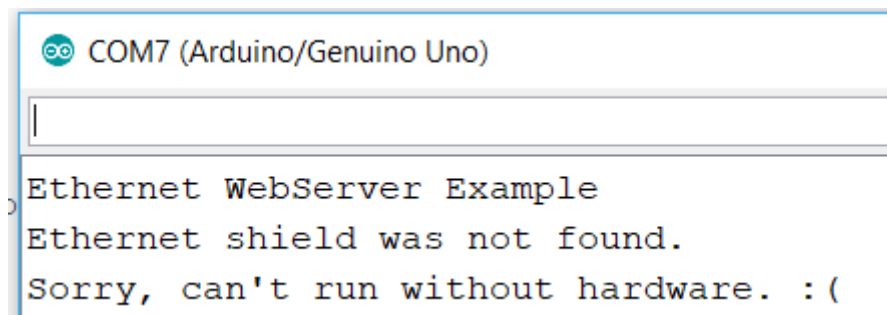
COM7 (Arduino/Genuino Uno)

Ethernet WebServer Example
server is at 192.168.0.201

```

שימו לב! הכתובת קבועה ונקבעה לפי כללי הרשת המקומיים. השרת מאזין בפורט 80 המתאים לפרוטוקול HTTP.

אם כרטיס ההרחבה אינו מחובר ללוח הפיתוח נקבל הודעת שגיאה:



```

COM7 (Arduino/Genuino Uno)

Ethernet WebServer Example
Ethernet shield was not found.
Sorry, can't run without hardware. :(

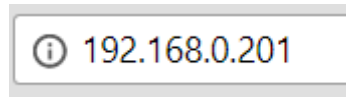
```

לאחר הגדרות החומרה, מאזינים לחיבור נכנס. עם קבלה של חיבור חדש קולטים את ההודעה המתקבלת מהלקוח ומזהים את סוף ההודעה. בדוגמה זו לא בודקים את סוג הבקשה אלא,

רק את סופה. עם זיהוי סוף הודעת בקשה נשלחת הודעת תגובה עם קוד הצלחה 200 ובהמשכו, כחלק ממסמך HTML מצורף ערך הקריאה של הטמפרטורה.

הפעלה

כדי לקבל מידע משרת האינטרנט שיצרנו נפתח דפדפן כלשהו (Chrome למשל) ונקליד את כתובת ה-IP שלו:



אם המחשב לא נמצא באותה הרשת כמו של השרת נקבל תגובה:

This site can't be reached

192.168.0.201 took too long to respond.

במקרה כזה נחבר את המחשב לאותה הרשת ונוודא שהכתובת שקבלנו שייכת למרחב הכתובות כמו זה של השרת.

אם החיבור תקין נקבל את התגובה הזו:

The temprature is 25.6

(או כל ערך אחר של טמפרטורה).

היתרון של נוחות הגישה לשרת מדפדפנים דורש ידע בשפת התגיות HTML שחורג מהידע הנדרש ביחידה זו.

לקורא המעוניין לפתח מערכת מבוססת שרת אינטרנט מומלץ להעמיק את הידע בתחום שפת התגיות וכן בפרוטוקול HTTP. לצורך כך ניתן להשתמש באתר:

<http://www.w3schools.com>

דוגמה 6.4 – מדידת ערך אנלוגי ושידורו לשרת

בדוגמה זו ניצור שרת מקומי המאזין לחיבורים נכנסים של לקוחות בפורט 5000. השרת יוכל לטפל ב-4 לקוחות לכל היותר. כל לקוח ישדר לשרת מספר מזהה ייחודי וערך של דגימה אנלוגית. השרת יאחסן לכל לקוח את 5 הדגימות האחרונות שקלט. השרת וכל לקוח מבוססים על לוח פיתוח UNO בעל כרטיס הרחבה EthernetShield.

תוכנת צד לקוח

```
#include <Ethernet.h>

#include <SPI.h>

// פרמטרים של כרטיס התקשורת
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED }; // כתובת פיזית
IPAddress ip(192, 168, 0, 202 ); // כתובת לוגית
// פרמטרים של השרת
unsigned int ServerPort = 5000; // פורט שהשרת מאזין בו
char stringServerIp[] = "192.168.0.105"; // מחרוזת כתובת השרת
IPAddress serverIp; // עצם כתובת שרת
int id = 1; // מזהה ייחודי של הלקוח
int analogPin = A0; // הדק הערוץ האנלוגי
EthernetClient client; // עצם תקשורת לקוח

void setup() {
  serverIp.fromString(stringServerIp); // המרת מחרוזת הכתובת ל-4 מספרים
  Ethernet.begin(mac, ip); // אתחול כרטיס הרשת
  Serial.begin(9600);
}
```

```

void loop() {
    trans();                                // שידור הערך האנלוגי
    delay(20000);                           // המתנה
}

// פעולה המשדרת את הערך האנלוגי לשרת
void trans() {
    Serial.print("connecting Server ");
    Serial.print(stringServerIp);
    Serial.print(":");
    Serial.println(ServerPort);

    // בדיקה אם הלקוח מחובר לשרת

    if (client.connect(serverIp, ServerPort)) { // חיבור הצליח

        // המידע משודר לכרטיס הרשת ובמקביל למוניטור בערוץ
        הטורי

        Serial.println("connected");
        client.print("ID=");
        Serial.print("ID=");
        client.println(id);
        Serial.println(id);
        client.print("VAL=");
        Serial.print("VAL=");

        int val = analogRead(analogPin);      // קריאת הערך האנלוגי
        client.println(val);
    }
}

```

```
Serial.println(val);

while (client.connected()){           // כל עוד קיים חיבור
    if (client.available()) {         // האם קיים נתון
        char c = client.read();       // קרא את הנתון
        Serial.print(c);              // הצגת הנתון
    }
}

Serial.println();

Serial.println("disconnecting.");

client.stop();                       // התנתקות מהשרת

} else {

    Serial.println("connection failed");

}

}
```

תוכנת צד שרת

בדוגמה זו נתחבר לרשת מקומית ונקבע את פרמטרי התקשורת באופן ידני.

```
#include <SPI.h>

#include <Ethernet.h>

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED}; // כתובת פיזית
IPAddress ip(192, 168, 0, 201); // כתובת לוגית
unsigned int ServerPort = 5000; ; // פורט שהשרת מאזין בו
EthernetServer server(ServerPort); // עצם תקשורת שרת
int samples[4][5]; // מערך דו-ממדי לאחסון מידע של 4 לקוחות (5 דגימות)

void setup() {
  Serial.begin(9600);
  Ethernet.begin(mac, ip); // אתחול השרת
  // בדיקה אם החומרה מחוברת
  if (Ethernet.hardwareStatus() == EthernetNoHardware) {
    Serial.println("Ethernet shield was not found.\nSorry, can't run without hardware. :(");
    while (true) {
      delay(1); // לולאה אינסופית
    }
  }
  server.begin(); // הפעלת השרת
  Serial.println("Server started");
  Serial.println(Ethernet.localIP());
  memset(samples, 0, sizeof(samples)); // איפוס ערכי המערך
```

```

}

void loop() {

    // האזנה לחיבורים נכנסים

    EthernetClient client = server.available();

    if (!client) // אם אין חיבור לסיים את הפעולה

        return;

    Serial.println("new client");

    while (!client.available()) { // המתנה לקבלת נתונים

        delay(1);

    }

    String req = client.readStringUntil('\r'); // קריאת מחרוזת טקסט לעצם מסוג
מחרוזת

    Serial.println(req);

    int id; // משתנה לזיהוי הנקלט

    int val; // משתנה לערך הנקלט

    if (req.indexOf("ID=") == -1) { // האם השורה הראשונה היא לא זיהוי
"יחודי"

        Serial.println("invalid data");

        client.stop(); // סגירת התקשורת ויציאה
מהפעולה

        return;

    } else {

        int i = req.indexOf("=");

        id = (int)(req.substring(i+1)).toInt(); // שלוף תת-מחרוזת והמר לערך מספרי

```

```

}

client.read();

req = client.readStringUntil('\r');           // קריאת מחרוזת של השורה
השנייה
Serial.println(req);

if (req.indexOf("VAL=") == -1) {             // האם השורה היא לא של ערך דגימה
    Serial.println("invalid data");

    client.stop();                           // סגירת התקשורת ויציאה
מהפעולה
    return;
} else {

    int i = req.indexOf("=");

    val = (int)(req.substring(i+1)).toInt(); // שלוף תת-מחרוזת והמר לערך מספרי
    if (id>0 && id<6){                       // בדיקת חוקיות של מזהה ייחודי
        id--;

        for(int i=0; i<4; i++)               // הזזת ערכי המערך כלפי מטה
            samples[id][i] = samples[id][i+1];

        samples[id][4] = val;                // הכנסת נתון לסוף המערך
    }
}

client.flush();                             // ריקון חוצץ הקלט

                                           // יצירת מחרוזת תגובה

String s = "Response: ID = ";

s += id+1;

```

```

s += " VAL = ";

s += val;

Serial.println(s);

// Send the response to the client

client.print(s);                                     // שידור מחרוזת תגובה

delay(1);

Serial.println("Client disconnected");

// אחרי היציאה מהפעולה העצם של הלקוח ישוחרר והתקשורת
תסתיים
}

```

לאחר הפעלת צד השרת וצד הלקוח נקבל את התגובות האלה:

חלון המוניטור של הלקוח

```

connecting Server 192.168.0.105:5000
connected
ID=1
VAL=246
Response: ID = 1 VAL = 246
disconnecting.

```

☐ Autoscroll

חלון המוניטור של השרת

```
new client
ID=1
VAL=246
Response: ID = 1 VAL = 246
Client disconnected
```

☐ Autoscroll

6.2 מודול מצלמה

בליבה של כל מצלמה דיגיטלית קיים התקן מצב מוצק⁹ הלוכד את האור המגיע דרך עדשה כדי ליצור תמונה. התקן זה נקרא חיישן.

החיישן בנוי מאוסף נקודות זעירות הרגישות לאור. כל נקודת חישה מייצרת אות יחסי לעוצמת האור הפוגע בה. מאוסף כל הנקודות ביחד ניתן לשחזר את התמונה כפי שנקלטה באמצעות החיישן. נקודת החישה, כחיישן אנלוגי, אינה מבחינה בין הצבעים השונים ולכן ממקמים מעליה מסננים צבעוניים המעבירים את הצבעים ירוק, כחול ואדום בלבד (RGB – Red Green Blue). כל נקודת חישה עוברת תהליך של הגברת עוצמה והמרת אות אנלוגי לדיגיטלי ולאחר מכן מיוצגת באמצעות ערך מספרי. כל נקודת חישה כזו נקראת פיקסל (pixel).

באופן עקרוני, ככל שמספר הפיקסלים בחיישן גדול יותר כך נוכל להבחין ביותר פרטים בתמונה. אולם לא די במספר הפיקסלים המוגדרים בחיישן, אלא גם בגודלו של החיישן. הגדלת מספר הפיקסלים בחיישן בעל ממדים מסוימים יקטין את ממדי נקודת הפיקסל מחד ויקטין את היעילות של הפיקסל מאידך. פיקסל קטן יקלוט פחות אור ויהיה רגיש יותר לרעש הנובע מאופן פעולתו. פיקסל גדול, לעומת זאת, יקלוט אור רב יותר ויהיה פחות רגיש לרעש הנובע מאופן פעולתו. פיקסל קטן מגיע לערכו המרבי בעוצמת הארה נמוכה יותר ביחס לפיקסל גדול יותר. כאשר פיקסל מגיע לערכו המרבי הוא עלול להשפיע על ערכי פיקסלים סמוכים ולשבש את התמונה המתקבלת.

לסיכום, אם נשווה בין שני חיישנים בעלי כמות פיקסלים שווה וממדים פיזיים שונים, הרי שבחיישן בעל הממדים הפיזיים הגדולים הפיקסל יהיה גדול יותר, ואילו בחיישן בעל הממדים הקטנים יותר הפיקסל יהיה קטן יותר. התמונה שתתקבל מהחיישן בעל הממדים הפיזיים הגדולים יותר תהיה באיכות טובה יותר.

כרטיסי הרחבה למצלמה ArduCAM 2MP

המודול המבוסס על החיישן OV2640 מיוצר ומשווק על ידי חברת ArduCAM בשתי תצורות:

⁹ התקן מצב מוצק, solid-state device, הוא התקן המבוסס על חומרים מוליכים למחצה. דיודה וטרנזיסטורים הם דוגמאות להתקנים כאלה.



תצורת מיני



תצורה רגילה

איור 6.4: תצורות של מודול OV2640

שתי התצורות באיור 6.4 נבדלות בשיטת התקשורת אל לוח הפיתוח. ביחידה זו נתמקד בתצורת המיני בלבד.

כרטיס הרחבה למצלמה ArduCAM-M-2MP

כרטיס ההרחבה בתצורת מיני הוא גרסה משופרת של הגרסה הרגילה ובעל מצלמה באיכות גבוהה 2MP ובעל ממשק תקשורת מסוג SPI המקטין את סיבוכי ההפעלה של המצלמה ביחס לגרסה הרגילה. הממשק SPI מאפשר חיבור של מספר כרטיסי הרחבה של מצלמה למיקרו-בקר אחד.

תכונות עיקריות

- חיישן תמונה 2MP מדגם OV2640
- ממשק תקשורת I2C להגדרת חיישן התמונה
- ממשק תקשורת SPI להוראות הפעלה של המצלמה ולזרימת הנתונים
- תמיכה בדחיסת תמונה JPEG

מאפיינים עיקריים

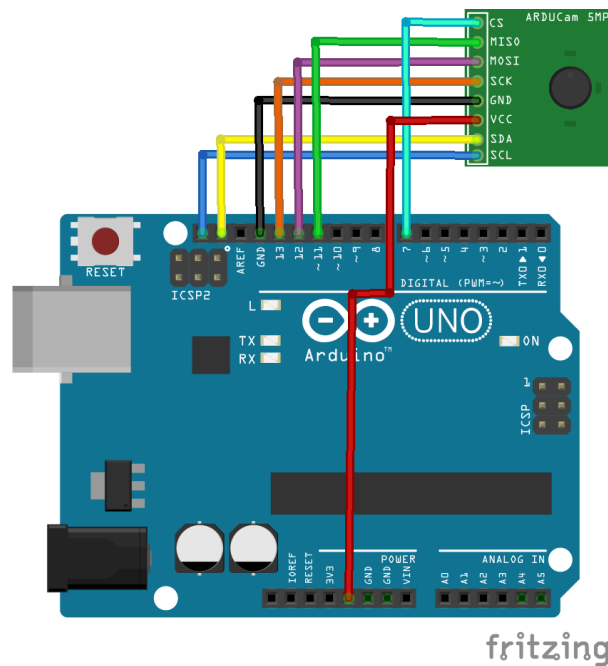
- מתח הפעלה 5V
- צריכת זרם
 - מצב רגיל: 70mA
 - מצב צריכה נמוכה: 20mA
- מהירות תקשורת SPI: 8MHz
- גודל מערך פעיל של החיישן: 1600x1200 פיקסלים

הדקי כרטיס ההרחבה

מספר	שם	סוג	תיאור
1	CS	קלט	קו בחירה של ההתקן לממשק תקשורת SPI
2	MOSI	קלט	קלט של ההתקן מממשק תקשורת SPI
3	MISO	פלט	פלט של ההתקן לממשק תקשורת SPI
4	SCLK	קלט	אות השעון של ממשק תקשורת SPI
5	GND	אדמה	אדמה
6	+5V	מתח	מתח הפעלה בתחום 3.3V עד 5V
7	SDA	דו-כיווני	קו נתונים דו-כיווני לתקשורת I2C
8	SCL	קלט	אות השעון של ממשק תקשורת I2C

חיבור המצלמה ללוח הפיתוח UNO

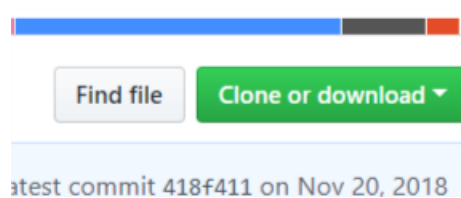
איור 6.5 מתאר את תרשימי החיבורים של כרטיס ההרחבה ללוח הפיתוח.



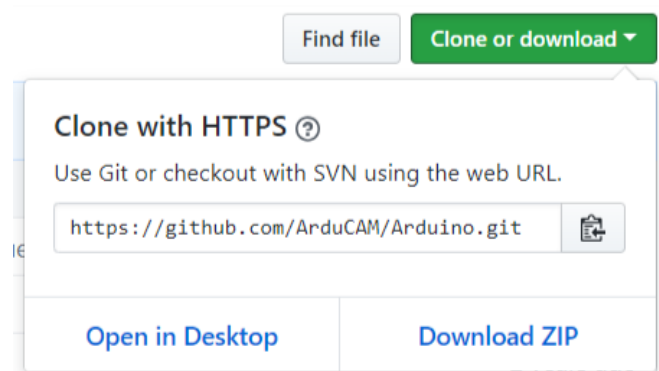
איור 6.5: תרשים חיבורים של כרטיס ההרחבה ללוח הפיתוח

התקנת ספריית תמיכה

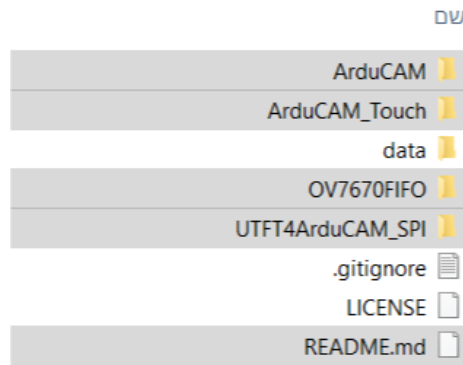
1. גלשו לאתר <https://github.com/ArduCAM/Arduino>
2. הורידו את קבצי הספרייה מהאתר – Clone or download



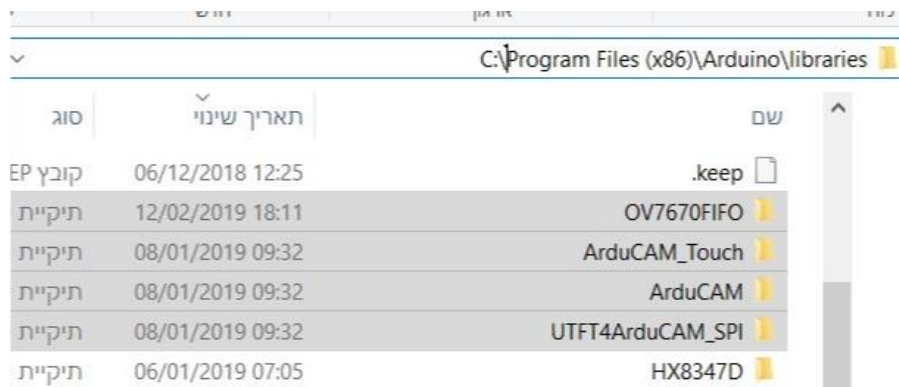
3. בחרו באפשרות Download ZIP



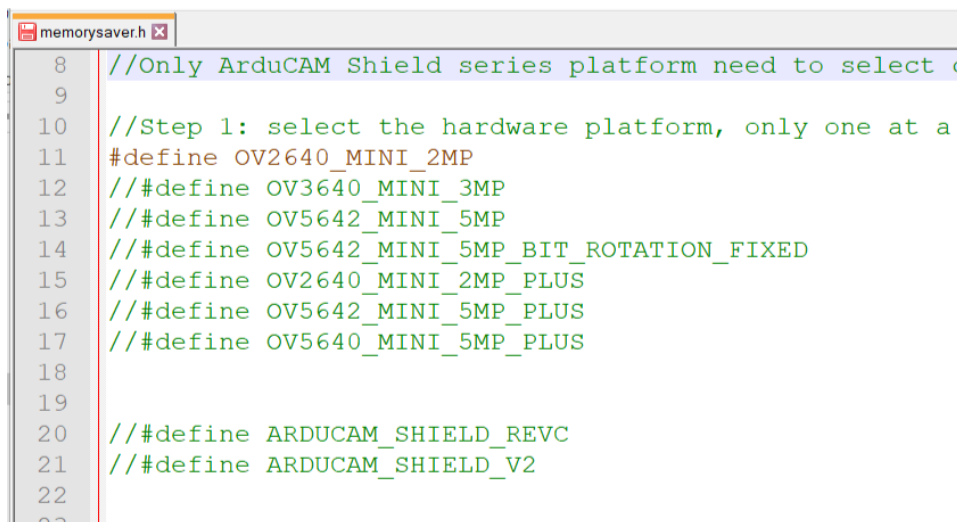
4. שמרו את הקובץ. שימו לב לתיקייה שבה הקובץ נשמר.
5. חלצו את הקובץ והעתיקו את הקבצים המסומנים אל תיקיית הספרייה של Arduino.



6. שמרו את הקבצים המסומנים בתיקייה.

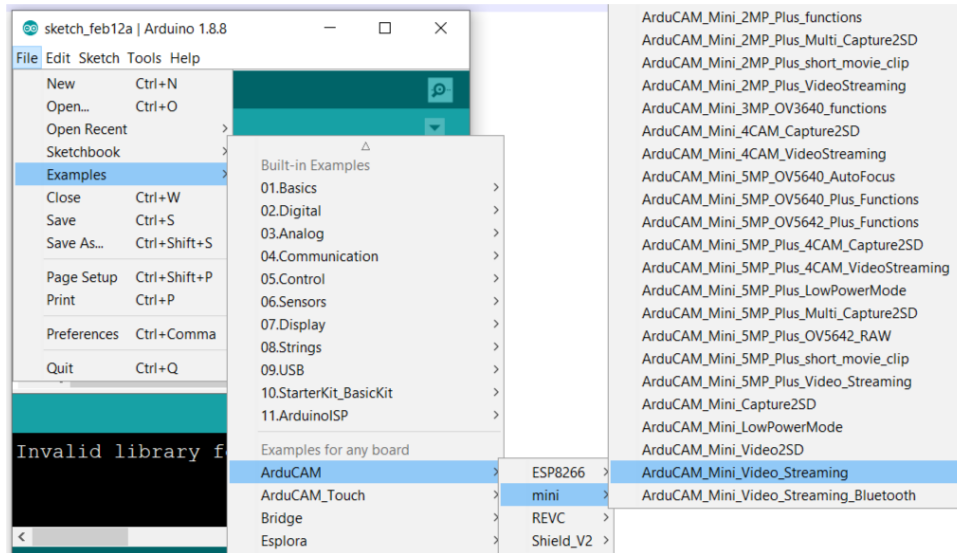


7. פתחו את התיקייה ArduCAM וערכו את הקובץ memorysaver.h



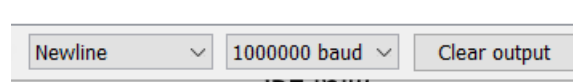
8. פתחו את סביבת הפיתוח והגדירו את לוח הפיתוח ל- UNO.

9. בתפריט הקובץ בחרו את הדוגמה הזו:

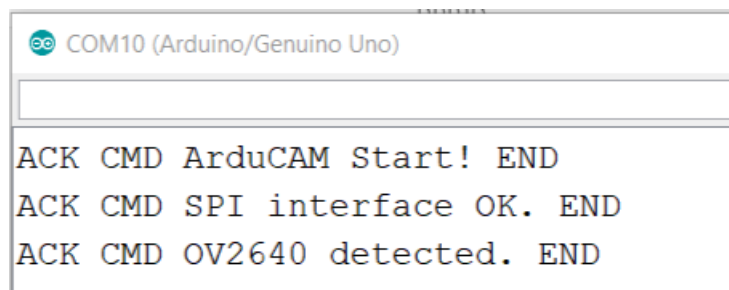


10. העלו את התוכנית ללוח הפיתוח.

11. פתחו את המוניטור ושנו את קצב התקשורת ל-1000000.



אם הכרטיס זוהה יתקבל את הדיווח הזה:



12. כדי לצפות בתמונה המתקבלת מהמצלמה הפעילו את התוכנית:

ArduCAM_Host_V2.exe

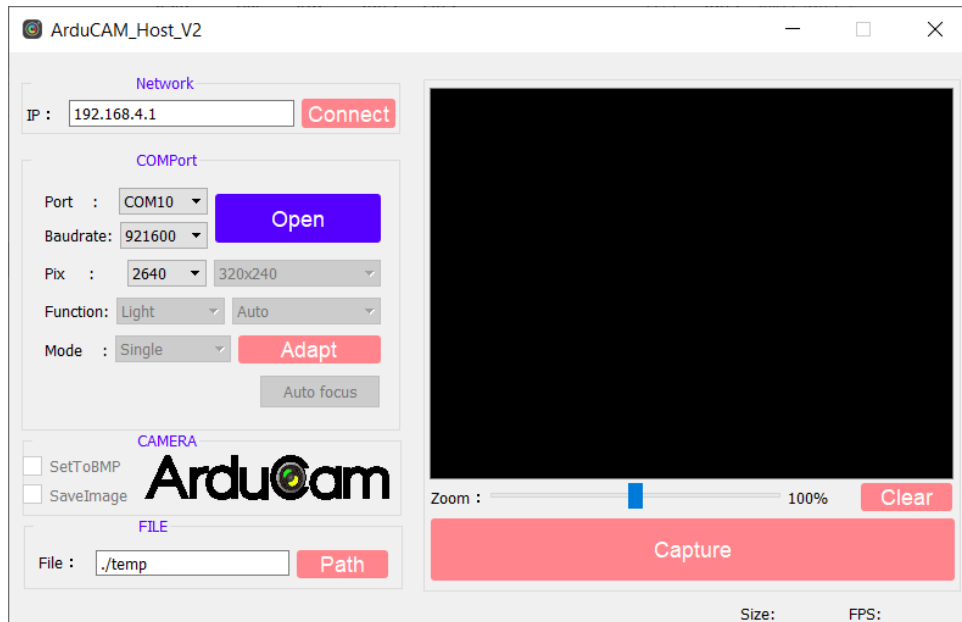
הנמצאת בתיקייה המשנה:

examples\host_app\ArduCAM_Host_V2.0_Windows

הנמצאת בתיקיה:

C:\Program Files (x86)\Arduino\libraries\ArduCAM\

(או במקום אחר המוגדר במחשב)



13. ודאו כי מפתח התקשורת מוגדר כראוי ולחצו על הכפתור Open.



14. לחיצה על כפתור Capture תלכוד תמונה.



סיכום פרק 6

בפרק 6 למדתם את הנושאים האלה:

1. תקשורת מחשבים
2. סוגי רשתות
3. מודל רשת 5 שכבות
4. פרוטוקול IP
5. כרטיס הרחבה רשת קווית
6. מחלקת IPAddress
7. מחלקת Ethernet
8. מחלקת EthernetClient
9. מחלקת EthernetServer
10. פרוטוקול HTTP
11. מודול מצלמה
12. לכידת תמונה למחשב
13. לכידת סרטון

פרק 7: עקרונות במערכות בקרה ממוחשבות

כיום מערכות בקרה ממוחשבות נפוצות מאוד ומשמשות לשליטה ולבקרה על פעולתם של תהליכים שונים.

מערכת הבקרה הממוחשבת מכילה שני רכיבים עיקריים:

- **יחידת בקרה ממוחשבת**, שתפקידה להפעיל את התהליך המבוקר ולבקר את אופן פעולתו, באמצעות תוכנת פיקוד ובקרה, המאוחסנת בזיכרון יחידת הבקרה.
- **יחידת ממשק אדם-מכונה**, שתפקידה לאפשר לאדם שמפעיל את "המכונה" (המתקן או התהליך), לקבוע את אופן פעולתה על ידי הזנת נתונים ומתן הוראות הפעלה ולעקוב אחר אופן פעולתה על ידי צפייה בתמונה של התהליך או בגרפים המופקים ממנו כפי שהם מוגדרים ביחידת הממשק אדם-מכונה.

שימוש בממשק אדם-מכונה, מקנה יתרון נוסף למערכת הבקרה הממוחשבת, והוא איסוף נתונים רציף מהתהליך המבוקר, אחסונם בזיכרון ועיבודם בהמשך, כדי לתת לאדם, המפעיל את התהליך, תמונה כוללת על תפקוד התהליך.

7.1 מושגי יסוד במערכות בקרה

מערכת לבקרת תהליך היא מערכת המאפשרת להפעיל בצורה מיטבית תהליך ייצור או תהליך המבצע פעולה נדרשת ולבקר את אופן פעולתו של התהליך.

המערכת לבקרת תהליך כוללת למעשה שני מרכיבים עיקריים הנובעים משמה – **מערכת ויחידת בקרה**.

המערכת היא צירוף כל המרכיבים של התהליך, הקשורים ביניהם באופן כזה שניתן להפעילם יחד ולהשתמש בהם להשגת המטרה הרצויה.

הבקרה היא פעולה של פיקוח ושליטה אוטומטית על משתנים פיזיקליים שונים (גדלים הניתנים למדידה ולחישוב) בתוך המערכת או פעולה של ויסות אנרגיה או חומר בתוך המערכת כדי לאפשר את תפקודה התקין של המערכת.

אופן ביצוע הבקרה, הדרישות מהמערכת ותוכנית ההפעלה של מערכת הבקרה נקבעות בידי האדם המפעיל את המערכת.

בכל מערכת בקרה נהוג להגדיר שני אותות עיקריים: אות המשתנה המבוקר ואות הערך הרצוי.

אות המשתנה המבוקר, המכונה גם משתנה התהליך, הוא אות פיזיקלי המתקבל בדרך כלל במוצא המערכת. אפשר למדוד אות זה במדידה ישירה או עקיפה. אות פיזיקלי זה נמדד ומומר לאות חשמלי (על ידי חיישן ומתמר), ולאחר מכן הוא משודר ליחידת הבקרה.

אות הערך הרצוי, המכונה גם אות הייחוס, הוא אות המבוא למערכת הבקרה, הקובע את נקודת הייחוס שהמערכת תופעל לפיה. בדרך כלל, אות זה מוזן על ידי האדם המפעיל את המערכת.

למעשה, תפקידה העיקרי של מערכת הבקרה הוא לווסת את המשתנה המבוקר כך שערכו יהיה שווה (או לפחות קרוב ככל שניתן) לערך הרצוי במערכת.

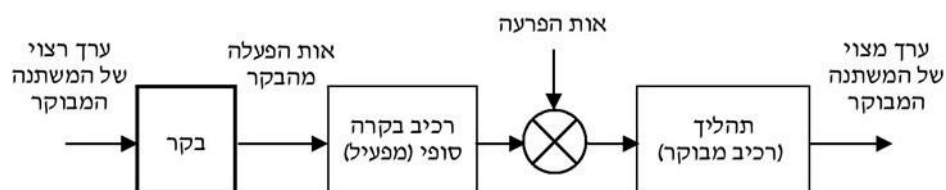
7.2 סיווג מערכות בקרה

מבחינים בין שני סוגים עיקריים של מערכות בקרה, הנבדלות באופן הפעלתן: מערכות הפועלות בחוג פתוח ומערכות הפועלות בחוג סגור. מערכות בקרה בחוג סגור נקראות גם מערכות בקרה בעלות משוב.

מערכת בקרה בחוג פתוח

מערכות בקרה הפועלות בחוג פתוח הן מערכות שלא מתקיים בהן קשר גומלין בין ערכו של המשתנה המבוקר, שהוא אות המוצא במערכת, לבין ערכו של הערך הרצוי, שהוא אות המבוא למערכת. למעשה, זו מערכת שאינה יכולה לווסת את עצמה.

מערכת בקרה בחוג פתוח מתוארת באמצעות תרשים המלבנים המוצג באיור 7.1.



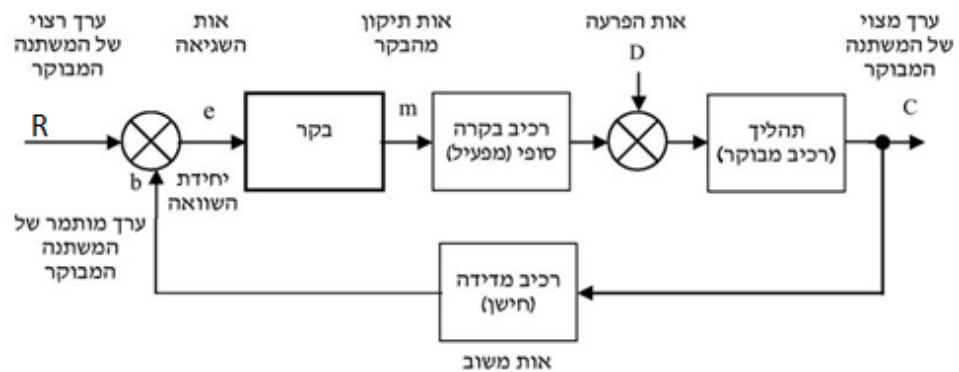
איור 7.1: תרשים מלבנים מערכת בקרה בחוג פתוח

מערכת בקרה בחוג סגור

מערכת בקרה בחוג סגור מקבלת שני אותות: אות המשתנה המבוקר ואות הערך הרצוי. תפקידה העיקרי של מערכת זו הוא לווסת את ערך האות הראשון כדי להשוותו, או לפחות לקרב אותו ככל שניתן, לערך של האות השני.

לצורך פעולת הוויסות, מוסיפים למערכת רכיב מדידה, המודד את המשתנה המבוקר ובודק מה קורה בפועל בתהליך. המידע המתקבל מרכיב המדידה משודר לבקר המבקר את פעולת התהליך ומאפשר לבקר לבצע תיקון בערכו של המשתנה המבוקר. רכיב המדידה מהווה למעשה אות משוב לבקר והוא מייחד את מערכת הבקרה בחוג סגור, לעומת מערכת בקרה בחוג פתוח, שאין בה אות משוב.

מערכת בקרה בחוג סגור מתוארת באמצעות תרשים המלבנים המוצג באיור 7.2.



איור 7.2: תרשים מלבנים מערכת בקרה בחוג סגור

כפי שאפשר לראות בתרשים המלבנים, במערכת בקרה בחוג סגור ישנם חמישה מרכיבים עיקריים:

יחידת השוואה (משוואה) – למשווה שני מבואות ומוצא אחד. מבוא אחד קולט את הערך הרצוי של המשתנה המבוקר (R), ומבוא שני קולט את הערך הנדגם (נמדד) מהתהליך עצמו (b). המשווה מחשב את השגיאה הנוצרת במערכת על ידי הפחתת הערך הנמדד מהערך הרצוי, וכך יוצר את אות השגיאה (e) המשודר לבקר (אות השגיאה המשודר לבקר הוא $e = R - b$).

הבקר – הבקר הוא "המוח" של מערכת הבקרה. הוא קולט במבואו את אות השגיאה המתקבל מהמשוואה, ומפיק במוצאו אות תיקון לתהליך. אות התיקון הוא אות חשמלי בדרך כלל, והוא משודר לרכיב הבקרה הסופי בתהליך, הנקרא גם מפעיל (אות התיקון תלוי בשגיאה: $m = f(e)$).

רכיב הבקרה הסופי (מפעיל) – תפקידו של רכיב זה הוא להפוך את אות התיקון החשמלי, שמפיק הבקר (m), לאות המאפשר ביצוע שינויים בתהליך עצמו. שינויים בתהליך דורשים בדרך כלל שימוש באנרגיה גדולה יחסית, לכן המפעיל כולל בתוכו גם אלמנט של הגברה.

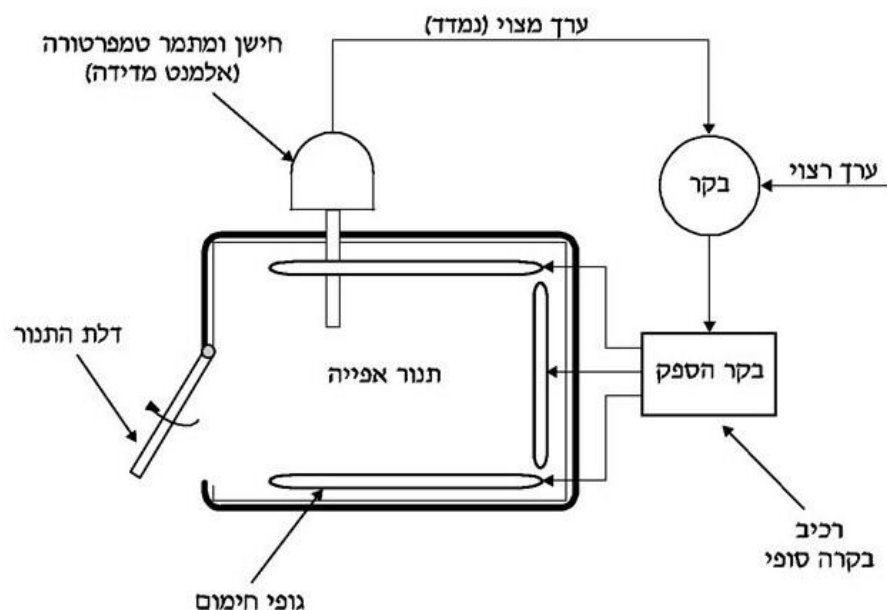
התהליך המבוקר – התהליך הוא המקום הפיזיקלי במערכת שבו מתבצעת פעולת הבקרה ובו מתקיים ונמדד המשתנה המבוקר.

רכיב המדידה – רכיב זה הוא החיישן הממוקם בתהליך, שתפקידו לדגום (למדוד) את המשתנה הפיזיקלי המבוקר. החיישן נעזר במתמר הממיר את הגודל הפיזיקלי הנמדד (כמו טמפרטורה, לחץ, מפלס וכדומה) לאות חשמלי המשודר ליחידת המשווה של מערכת הבקרה.

האות הנמדד בעזרת החיישן משמש כאות משוב במערכת הבקרה. אות זה מאפשר לבקר לבצע את פעולת התיקון הנדרשת במערכת הבקרה, בהתאם לשגיאה המחושבת.

דוגמה 7.1 – מערכת בקרה בחוג סגור של תנור אפייה

באיור 7.3 מתוארים מרכיבי תנור אפייה תעשייתי שבו מתבצעת בקרת טמפרטורה.



איור 7.3: מרכיבי תנור אפייה

המערכת כוללת חיישן ומתמר טמפרטורה, המשמשים כרכיב מדידה הממוקם בתהליך. תפקידו של הרכיב הוא למדוד את הטמפרטורה בתנור. הטמפרטורה היא המשתנה המבוקר במערכת, וגופי החימום משמשים כאמצעי החימום בתנור. בקר הספק משמש כמפעיל או כרכיב הבקרה הסופי במערכת. תפקידו של הבקר הוא לשלוט על עוצמת ההספק המסופק לגופי החימום. הטמפרטורה בתנור מווסתת בחוג סגור, בעזרת הבקר, לערכה הרצוי. הערך הרצוי, הדרוש בתנור, נקבע על ידי האדם המפעיל את המערכת. חיישן טמפרטורה המותקן בתנור מודד את הטמפרטורה (הערך המצוי) ומשדר אותה אל בקר הטמפרטורה. הבקר מבצע השוואה בין ערך הטמפרטורה הנמדד לבין ערך הטמפרטורה הרצוי שהוזן אליו. ההפרש בין שני האותות (השגיאה) מומר לאות מוצא מהבקר (אות תיקון) משודר אל בקר ההספק. בקר ההספק שולט על עוצמת ההספק המסופק לגופי החימום, ועל הטמפרטורה בתוך התנור, ומביא את טמפרטורת התנור לערכה הרצוי.

במערכת הבקרה של תנור האפייה, ההפרעה החיצונית מתבטאת בפתיחת הדלת בזמן הכנסת מוצרים לתנור, ובסגירתה לאחר מכן.

פתיחת הדלת גורמת ל"בריחת חום" מהתנור אל הסביבה, וכתוצאה מכך הטמפרטורה בתנור יורדת. חיישן הטמפרטורה מזהה את ירידת הטמפרטורה ומשדר לבקר אות משוב קטן יותר. הבקר מחשב את השגיאה בין הערך הרצוי (הקבוע) לבין הערך הנמדד (שירד), ומפיק אות תיקון להפעלת גופי החימום בהספק גבוה יותר, כדי להתגבר על ירידת הטמפרטורה.

תהליך דומה קורה לאחר סגירת הדלת של התנור. החיישן מזהה כי הטמפרטורה בתנור עלתה (בגלל ההספק הגבוה יותר שנשלח קודם לכן אל גופי החימום) וגורם לבקר להפיק אות תיקון קטן יותר, שיקטין את ההספק של גופי חימום, עד לייצוב הטמפרטורה בתוך התנור סביב הערך הרצוי שנקבע.

7.3 מערכת בקרה ממוחשבת

בתהליכים התעשייתיים המודרניים משולבים חוגי בקרה רבים. בין המבואות והמוצאים של החוגים השונים קיים קשר מורכב. המורכבות הנדרשת מחוגי הבקרה מחייבת שימוש במערכות בקרה ממוחשבות המבוססות על מיקרו-מעבד או על מיקרו-בקר.

בהשוואה למערכות בקרה רגילות (המבוססות על רכיבים אלקטרוניים) יש למערכת בקרה ממוחשבת כמה יתרונות בולטים:

- יכולת לבצע פעולות מורכבות יותר ומהירות יותר על נתונים
- יכולת לטפל בכמות רבה של משתנים מבוקרים בו-זמנית
- יכולת ליצור אותות תיקון לתהליך המבוקר באמצעות תוכנה, ולא רק על ידי חומרה
- יכולת להציג את נתוני התהליך בזמן אמת, על גבי צג מחשב, בצורה נוחה וגמישה: בצורת גרפיקה, בצורת אנימציה או בצורת דוחות
- יכולת לאגור נתונים לצורכי תיעוד, היסטוריה וניתוח תהליכים, חקירת מצבים חריגים ותקלות

המבנה והמאפיינים של מערכת בקרה ממוחשבת

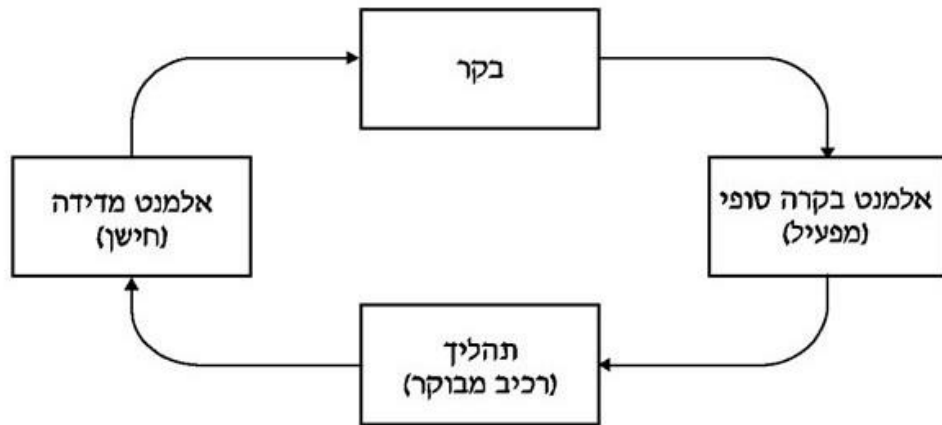
במערכות בקרה ממוחשבות המיקרו-בקר הוא רכיב מרכזי בפעולת המערכת. בדומה לכל מערכת בקרה הפועלת בחוג סגור, גם המערכת הממוחשבת כוללת כמה רכיבים המאפיינים את פעולתה:

- בקר (הכולל בתוכו משווא או פונקציית השוואה)
- מפעיל המהווה רכיב בקרה סופי
- תהליך או רכיב מבוקר
- חיישן – רכיב מדידה הממיר ומשדר את אות המשתנה המבוקר

איור 7.4 מציג את רכיבי מערכת הבקרה הממוחשבת בחוג סגור, ואת קשרי הגומלין ביניהם: הבקר מפיק אות לרכיב הבקרה הסופי (המפעיל) המותקן בתהליך, ורכיב המדידה (החיישן) מודד את פיזיקלי בתהליך ומשדר אותו אל הבקר.

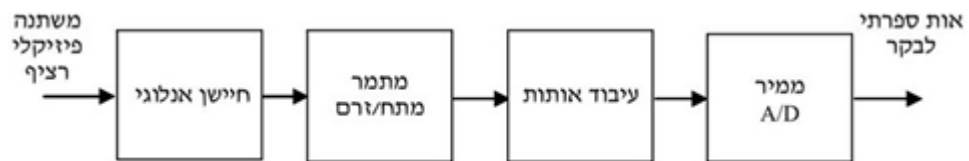
מערכת בקרה ממוחשבת היא מערכת ספרתית המבוססת על מיקרו-בקר, ורוב האותות שהיא קולטת, מעבדת ומשדרת הם ספרתיים. עיבוד הנתונים מבוסס על תוכנה מתאימה הטעונה במיקרו-מעבד.

מערכות הבקרה הממוחשבות כוללות כמה רכיבים אלקטרוניים שתפקידם להמיר את האותות הרציפים במערכת באותות הספרתיים בבקר. רכיבים אלה נקראים ממירי אות.



איור 7.4: קשרי הגומלין בין רכיבי מערכת הבקרה הממוחשבת בחוג סגור

מערכת לדגימת אותות משמשת להמרת אות פיזיקלי רציף (אות אנלוגי) לאות ספרתי. באיור 7.5 מוצג תרשים מלבנים של המערכת.



איור 7.5: תרשים מלבנים של מערכת לדגימת אותות מהתהליך

המשתנה הפיזיקלי שהמערכת דוגמת יכול להיות גודל חשמלי (מתח, התנגדות) גודל אופטי (עוצמת הארה, גודל מכני – תזוזה או לחץ) או כל גודל פיזיקלי אחר. החיישן נבחר בהתאם לסוג המשתנה הפיזיקלי שהמערכת צריכה למדוד. המתמר קולט את המידע המשודר מהחיישן האנלוגי וממיר אותו באות מתח חשמלי או לזרם חשמלי בערכים הנהוגים במערכות הבקרה. מעגל עיבוד האותות מסנן את האות המשודר מהחיישן. הממיר A/D ממיר את אות המתח או את הזרם האנלוגי באות ספרתי המשודר לבקר המערכת.

מערכת שמפעילה את רכיבי הבקרה הסופיים בתהליך, משמשת להמרת אות ספרתי באות אנלוגי. באיור 7.6 מוצג תרשים מלבנים של המערכת.



איור 7.6: תרשים מלבנים של מערכת להפעלת רכיבים אנלוגיים

במערכת בקרה ממוחשבת הבקר מפיק אות תיקון ספרתי המומר באות אנלוגי, ומשמש להפעלת המפעיל (רכיב הבקרה הסופי) האנלוגי. המפעיל ממיר את האות החשמלי שהוא מקבל לגודל פיזיקלי תואם.

הבקר הספרתי במערכת, מחשב ומפיק אות תיקון ספרתי לתהליך, הנשלח לממיר ה-D/A. הממיר ממיר את זה לערך אנלוגי של מתח או זרם המשודרים לרכיב הבקרה הסופי (מפעיל). אות זה הוא אות תקני המתקבל במערכות בקרה. לעיתים יש צורך להגבירו או להמירו לאות המאפשר את הפעלת רכיב הבקרה הסופי במערכת. המפעיל מאפשר לשנות את ערכו של המשתנה הפיזיקלי המבוקר בתהליך.

אביזרי קצה

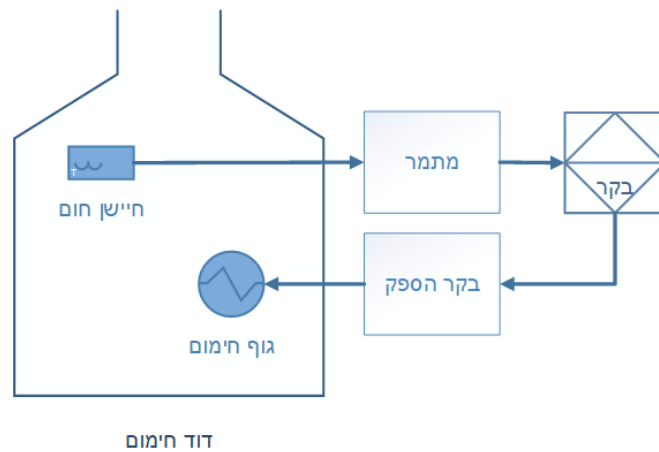
המונח אביזרי קצה הוא שם כולל לכל האביזרים המותקנים בתהליך "בקצוות" של יחידת הבקרה, במבוא היחידה ובמוצאה. ללא אביזרי הקצה לא היה מתאפשר תפקוד מלא של התהליך.

בין אביזרי הקצה נכללים החיישנים, רכיבים המותקנים בתהליך או בסביבתו, ודוגמים את המתרחש בו, והמפעילים, רכיבים המותקנים בתהליך ומאפשרים את הפעלתו בצורה הרצויה.

7.4 מערכות בקרה מבוססות מיקרו-בקר

דוגמה 7.2 – מערכת לבקרת טמפרטורה דו-מצבית עם חיישן ספרתי

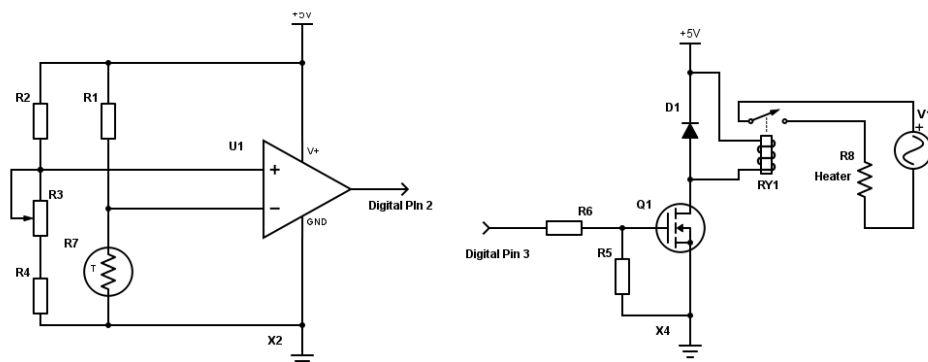
באיור 7.7 מתוארים רכיבי דוד חימום שבו מתבצעת בקרת טמפרטורה.



איור 7.7: רכיבי דוד החימום

המערכת כוללת חיישן ומתמר טמפרטורה, המשמשים כיחידת מדידה הממוקמת בתהליך. תפקידה למדוד את הטמפרטורה בדוד. הטמפרטורה היא המשתנה המבוקר במערכת. גוף החימום משמש כאמצעי החימום בדוד. בקר הספק משמש כמפעיל או כרכיב הבקרה הסופי במערכת, שתפקידו לשלוט על עוצמת ההספק המסופק לגוף החימום. הטמפרטורה בדוד מווסתת בחוג סגור, בעזרת הבקר, לערכה הרצוי. הערך הרצוי הנדרש בדוד, נקבע בידי האדם המפעיל את המערכת. חיישן טמפרטורה המותקן בתנור, מודד את הטמפרטורה (הערך המצוי) ומשדר אותה אל הבקר. הבקר משווה בין ערך הטמפרטורה הנמדד לבין ערך הטמפרטורה הרצוי. ההפרש בין שני האותות (השגיאה) מומר לאות מוצא מהבקר (אות תיקון) משודר אל בקר ההספק. בקר ההספק שולט על עוצמת ההספק המסופק לגוף החימום ועל הטמפרטורה בתוך הדוד ומביא את טמפרטורת הדוד לערכה הרצוי.

באיור 7.8 מתואר המעגל החשמלי המבקר את הטמפרטורה בדוד החימום.



איור 7.8: רכיבי המעגל החשמלי

חיישן הטמפרטורה T מחובר בתצורת גשר נגדים. יציאת הגשר מחוברת אל משווא אנלוגי. כאשר המתח המתפתח על חיישן הטמפרטורה נמוך יותר מהמתח המתפתח על הנגדים R4 ו-R3 יתקבל במוצא מגבר השרת מתח גבוה. ולהיפך, כאשר המתח על החיישן יהיה גבוה יותר יתקבל מתח מוצא נמוך. מוצא מגבר השרת מחובר אל הדק 2 בלוח הפיתוח UNO, המוגדר כהדק מבוא.

קביעת הערך הרצוי של הטמפרטורה נקבע באמצעות קביעת ערכי הנגדים של הגשר. כפי שניתן לראות בתוכנית בהמשך, הערך של האות בהדק 2 נדגם, ומתבצעת החלטה על הפעלת גוף החימום. אם הטמפרטורה נמוכה מהערך הדרוש מסופק ערך גבוה (HIGH) להדק 3 המוגדר כהדק מוצא. אם הטמפרטורה נמוכה מהערך הרצוי מסופק ערך נמוך (LOW). אות המוצא מהדק 3 מסופק לטרנזיסטור ממשפחת MOS-FET המשמש כמתג. כאשר מתקבל מתח גבוה בהדק 3, הטרנזיסטור עובר למצב הולכה, עובר דרכו זרם הגורם לממסר המחובר אליו להוליך ולהעביר את המגעון (קונקטור) למצב סגור (ON). כאשר המתח בהדק 3 נמוך הטרנזיסטור עובר למצב קטעון ומפסיק את הזרם בממסר. עקב כך המגעון עובר למצב פתוח (OFF).

כאשר המגעון סגור מתפתח על גוף החימום מתח הגורם לפיזור חום.

בקרה דו-מצבית: ON-OFF

בדוגמה זו התהליך המבוקר הוא מכל המים של דוד החימום. המשתנה המבוקר הוא טמפרטורת המים. המיקרו-בקר משמש כמשווא וכבקר עצמו המספק אותות דו-מצביים HIGH ו-LOW המפעילים מתג אלקטרוני הגורם להפעלה של גוף החימום באמצעות ממסר. המתג והממסר הם אלמנט ההפעלה הסופי. במצב ON, שבו האות הוא HIGH, גוף החימום מופעל ומעלה את הטמפרטורה של הנוזל. במצב OFF, שבו האות הוא LOW, גוף החימום מפסיק את פעולתו והטמפרטורה יורדת באופן טבעי.

בתהליך חימום נוזל, כפי שמתואר בדוגמה, קבוע זמן החימום הוא ארוך ולכן יש להמתין זמן ארוך בין הדגימות (קריאות) של הטמפרטורה. המתנה ארוכה יחסית תאפשר לחימום או להפסקתו להשפיע על התהליך.

להלן תוכנית הבקרה

```

const int Sensor = 2;
const int Heater = 3;

void setup() {
  pinMode(Sensor, INPUT);
  pinMode(Heater, OUTPUT);
}

void loop() {
  if (digitalRead(Sensor) == HIGH)
    digitalWrite(Heater, HIGH);
  else
    digitalWrite(Heater, LOW);
  delay(1000);
}

```

הערה: התוכנית לעיל מתאימה לחיישן טמפרטורה מסוג PTC. התנגדות של חיישן זה גדלה ככל שהטמפרטורה עולה.

תרגול 

שאלה 7.1

כתבו הוראה אחת המחליפה את משפט התנאי בתוכנית לעיל.

שאלה 7.2

מדוע נדרשת הוראת השהייה בתוכנית?

שאלה 7.3

משנים את חיישן הטמפרטורה לסוג NTC. התנגדות החיישן קטנה ככל שהטמפרטורה עולה. שנו את התוכנית כך שהיא תתאים לחיישן זה.

שאלה 7.4

מחליפים את חיישן הטמפרטורה בחיישן התנגדותי הרגיש לאור – LDR.

א. הסבירו את פעולת המעגל החשמלי.

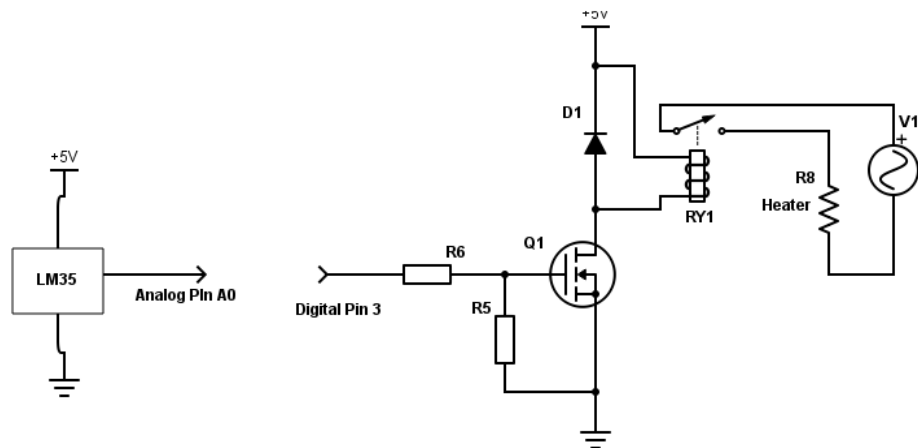
ב. האם צריך לשנות את התוכנית? אם כן הציעו תוכנית מתאימה.

דוגמה 7.3 – מערכת לבקרת טמפרטורה דו-מצבית עם חיישן תקבילי

דוגמה זו מבוססת על איור 7.7. בדוגמה זו נחליף את החיישן והמתמר הספרתי בחיישן תקבילי

מסוג LM35 [d1].

איור 7.9 מתאר את המעגל החשמלי.



איור 7.9: רכיבי המעגל החשמלי

בשונה מהדוגמה הקודמת, לחיישן הטמפרטורה תקבילי המאפיינים האלה:

- מסוגל למדוד טמפרטורות מ-55- מעלות ועד ל-150 מעלות
- הרכיב מסוגל לפעול במתחי עבודה מ-4V ועד 20V
- מקדם טמפרטורה לינארי – $10\text{mV}/^{\circ}\text{C}$
- תמיכה במדידת טמפרטורה חיובית או שלילית לפי תצורת חיבור מתאים

החיבור המתואר באיור 7.9 מאפשר מדידת טמפרטורות בין 2 ל-150 מעלות.

בשיטת חיבור זו ערך הטמפרטורה הרצויה נקבע בתוכנית עצמה כערך קבוע. ניתן לשנות שיטה זו באמצעות חיבור ממשק מתאים של לוח מקשים ותצוגה או שימוש בתקשורת אינטרנט.

להלן תוכנית הבקרה

```
const int Sensor = A0;

const int Heater = 3;

const int refTemp = 40; // temprature referance value

int val = 0; // ערך ספרתי
float voltage; // ערך מתח מחושב
float temp; // ערך טמפרטורה מחושב

void setup() {
  pinMode(Heater, OUTPUT);
}

void loop() {
  val = analogRead(Sensor); // קריאת ערך אנלוגי
  voltage = 0.00488 * val; // חישוב ערך המתח
  temp = voltage / 0.01; // חישוב ערך הטמפרטורה

  if (temp < refTemp)
    digitalWrite(Heater, HIGH);
  else
    digitalWrite(Heater, LOW);

  delay(1000);
}
```

שאלה 7.5

א. הציעו תוספת לבחירתכם הכוללת ממשק משתמש: לוח מקשים ותצוגה או תקשורת אינטרנט.

ב. שכתבו את התוכנית מהדוגמה כך שניתן יהיה לקבוע את הערך הרצוי.

שאלה 7.6

תלמיד הציע לבטל את ההשהייה בתוכנית.

א. כיצד ישפיע שינוי זה על תפקוד התוכנית?

ב. הציעו דרך לבטל את התופעה המתקבלת מביטול זה.

בקרה רציפה – יחס

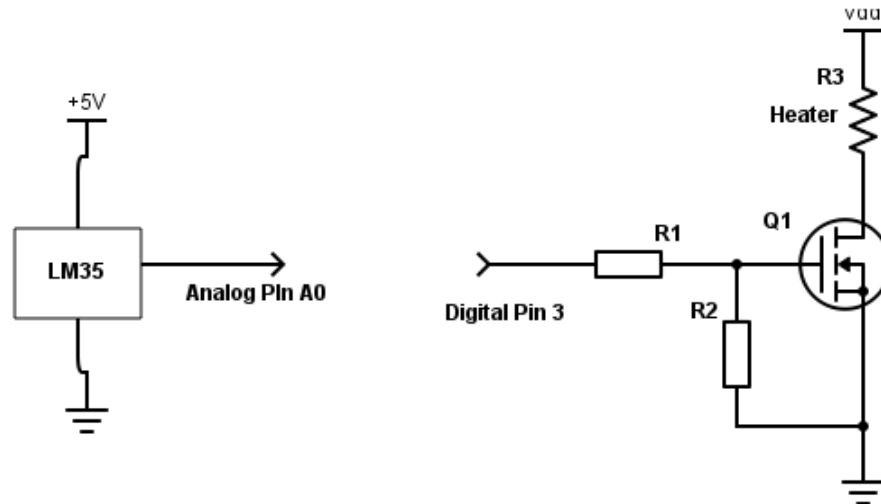
בדוגמה לפניכם נתאר מערכת בקרה שאות התיקון שלה הוא אות אנלוגי. אות התיקון הוא ערך מספרי רציף מ-0 עד 255 המייצג את רוחב הדופק בשיטת PWM. ככל שהערך המספרי גדול יותר עוצמת המתח הישר הממוצע גבוה יותר.

שיטת PWM נדונה בהרחבה ביחידה "מבוא למערכות משובצות מחשב", פרק 3 – "מבואות ומוצאים תקביליים" בסעיף 3.5 מוצא אנלוגי. [d2]

דוגמה 7.4 – מערכת לבקרת טמפרטורה בשיטת PWM עם חיישן**תקבילי**

באיור 7.10 מתואר מעגל חשמלי לבקרת טמפרטורה. במעגל זה הטרנזיסטור ממתג באופן ישיר את גוף החימום המחובר אל מקור מתח ישר. לעומת זאת, בדוגמאות הקודמות אין חשיבות לסוג אות המתח מכיוון שבקרת ההספק מתבצעת באמצעות מגעון מכני.

השימוש במגעון מכני אינו ישים בשיטת PWM בעיקר בגלל זמן המיתוג הארוך יחסית והבלאי המכני שהיה נגרם לו. ניתן כמובן למתג מתח חילופין באמצעות רכיבים אלקטרוניים כדוגמת TRIAC-SCR.



איור 7.10:

רכיבי המעגל החשמלי

גם כאן, בשיטת חיבור זו ערך הטמפרטורה הרצויה נקבע בתוכנית עצמה כערך קבוע. ניתן לשנות שיטה זו באמצעות חיבור ממשק מתאים של לוח מקשים ותצוגה או שימוש בתקשורת אינטרנט.

להלן תוכנית הבקרה:

```
const int Sensor = A0;
const int Heater = 3;
const int refTemp = 40;           // ערך טמפרטורת הייחוס
int val = 0;                      // ערך ספרתי
float voltage;                    // ערך מתח מחושב
float temp;                       // ערך טמפרטורה מחושב
int heaterValue = 128;            // הפעלה בעצמה של 50%

void setup() {
  pinMode(Heater, OUTPUT);
  analogWrite(Heater, heaterValue); // התחלה בעוצמה של 50%
}
```



```

void loop() {

    val = analogRead(Sensor);           // קריאת ערך אנלוגי
    voltage = 0.00488 * val;             // חישוב ערך המתח
    temp = voltage / 0.01;                // חישוב ערך הטמפרטורה

    // change pwm value

    if (temp < refTemp)

        heaterValue += 5;

    else

        heaterValue -= 5;

    // check limits

    if (heaterValue < 0)

        heaterValue = 0;

    if (heaterValue > 255)

        heaterValue = 255;

    analogWrite(Heater, heaterValue);

    delay(5000);

}

```

בתוכנית נדגם הערך הגולמי של הטמפרטורה כל 5 שניות. ערך זה מומר לערך מתח ולאחר מכן לערך טמפרטורה. ערך הטמפרטורה משווה לערך הרצוי. אם הטמפרטורה לא מגיעה לערך הרצוי מסופק אות PWM בעצמה גבוהה יותר. לו עברנו את הערך הרצוי, מסופק אות PWM בעצמה נמוכה יותר.

בכל מחזור פעולה ערך אות PWM מקודם או מופחת ב-5. לאחר כל שינוי בערכו של אות PWM נבדקים גבולות האות. אם קיימת חריגה, ערכי אות PWM מתוקנים לערכי הגבולות המתאימים.

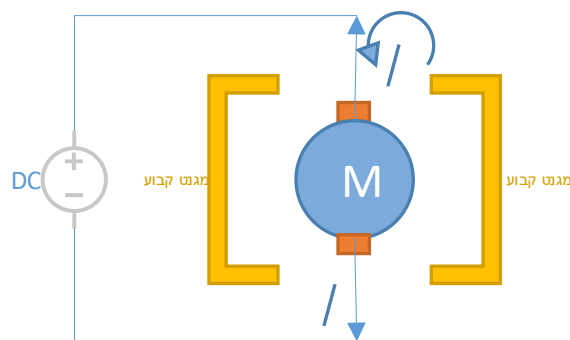
תרגול

שאלה 7.7

מה יקרה אם נקצר את זמן ההשהייה מ-5 שניות ל-0.1 שנייה?

7.4.1 מנוע זרם ישר (DC motor)

מנוע חשמלי הוא התקן הממיר אנרגיה חשמלית לאנרגיית תנועה סיבובית. בכל מנוע יש שני חלקים: סטטור ורוטור. הסטטור הוא בית המנוע שאינו נע, כלומר החלק הנייח, ואילו הרוטור הוא החלק המותקן בתוך בית המנוע והנע בתנועה סיבובית. תנועת המנוע מתרחשת בעקבות הכוח הנוצר על תיל מוליך זרם השרוי בשדה מגנטי. ביחידה זו נתמקד במנועים הפועלים בזרם ישר בלבד. השדה המגנטי נוצר בסטטור באמצעות סלילים חשמליים או באמצעות מגנטים קבועים. לשם פישוט נתייחס למנועי זרם ישר בעלי מגנטים קבועים. הרוטור בנוי מציר מרכזי ומסביבו מלופפים סלילים חשמליים. מכיוון שהרוטור מסתובב נדרשים מגעונים, הנקראים גם מברשות, שתפקידם לחבר בין מקור המתח החשמלי המחובר למנוע לבין סלילי הרוטור. המברשות במנוע מתבלות עם השימוש, ולכן במנועים המשמשים רבות נהוג לאפשר החלפה של המברשות.



איור 7.11: מבנה עקרוני של מנוע זרם ישר

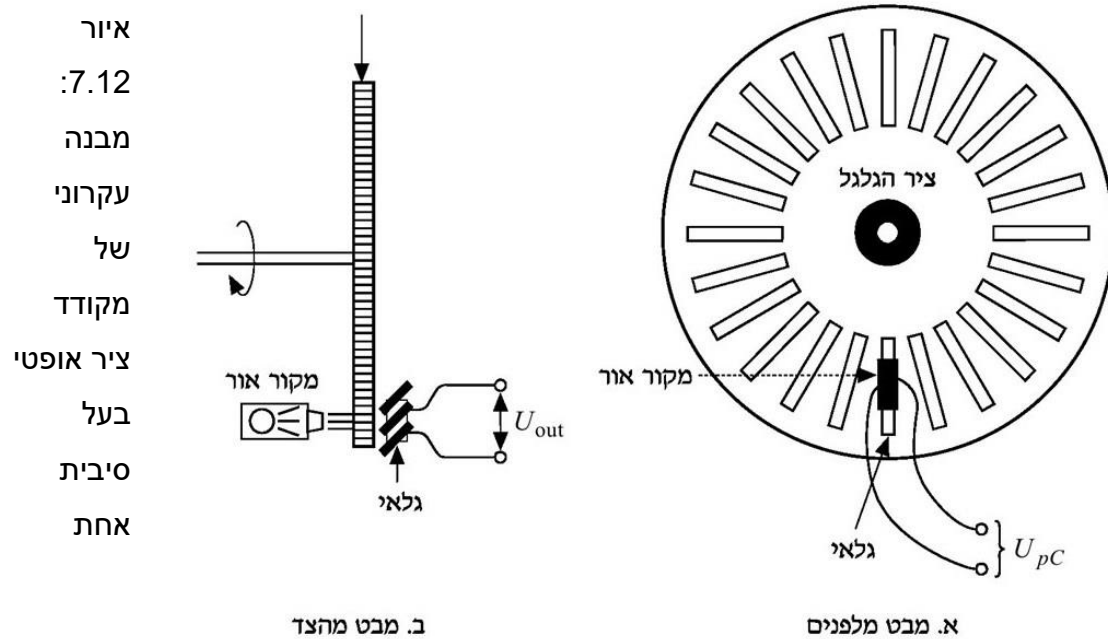
מנוע לזרם ישר יכול לפעול בשני כיוונים על פי קוטביות המתח המחובר למברשות.

לצפייה בסרטון אנימציה המתאר את אופן פעולתו של המנוע לחץ כאן.

7.4.2 מקודד ציר אופטי

מקודד ציר אופטי הוא התקן המחובר על ציר המנוע ומספק אותות ספרתיים המציינים את כיוון הסיבוב ואת הזווית והמהירות של המנוע. המדידה נעשית באמצעות ספירה של הדפקים החשמליים שיוצר גלאי אור.

איור 7.12 מתאר את המבנה העקרוני של מקודד ציר אופטי של סיבית אחת הבנוי מגלגל ובו 24 גזרות זהות, שבכל אחת מהן יש חריץ. בכל מעבר של חריץ מול מקור האור נוצר דופק חשמלי U_{out} .



זווית הסיבוב וכושר הבחנה

בגלגל שבו יש K חריצים, בכל סיבוב שלם של הגלגל (סיבוב שלם של ציר המנוע) נוצרים K דפקים חשמליים. כל דופק חשמלי מציין תנועה של $1/K$ של הסיבוב השלם.

כושר ההבחנה R במעלות של המקודד הוא:

$$R = \frac{360}{K}$$

כושר ההבחנה של המקודד שבאיור 7.11 הוא:

$$R = \frac{360}{K} = \frac{360}{24} = 15^\circ$$

תרגול 

שאלה 7.8

מהי הזווית שתימדד אם נספרים 18 דפקים חשמליים?

מהירות המנוע

נהוג לציין מהירות המנוע ביחידות של סיבובים לדקה – סל"ד (RPM – Round Per Minute). כדי לחשב את מהירות ההמנוע סופרים את מספר הדפקים החשמליים (P) ביחידת זמן נתונה (T). לאחר הספירה מחשבים את מספר הדפקים החשמליים (f_p) הנוצרים במשך שנייה אחת:

$$f_p = \frac{P}{T} \text{ [pulses per second or Hz]}$$

הכפלת המספר ב-60 מאפשרת לחשב את מספר הדפקים בדקה אחת (f_{pm}):

$$f_{pm} = f_p \cdot 60 \text{ [pulses per minute]}$$

במנוע שבו קיימים K חריצים, מהירותו (n) תהיה:

$$n = \frac{f_{pm}}{K} = \frac{P}{KT} \cdot 60 [\text{round per minute}]$$

למשל, ספירה של 96 דפקים חשמליים בשנייה אחת עבור הדוגמה שבאיור 7.11 תציין מהירות:

$$n = \frac{P}{KT} \cdot 60 = \frac{96}{24 \cdot 1} \cdot 60 = 240RPM$$

תרגול 

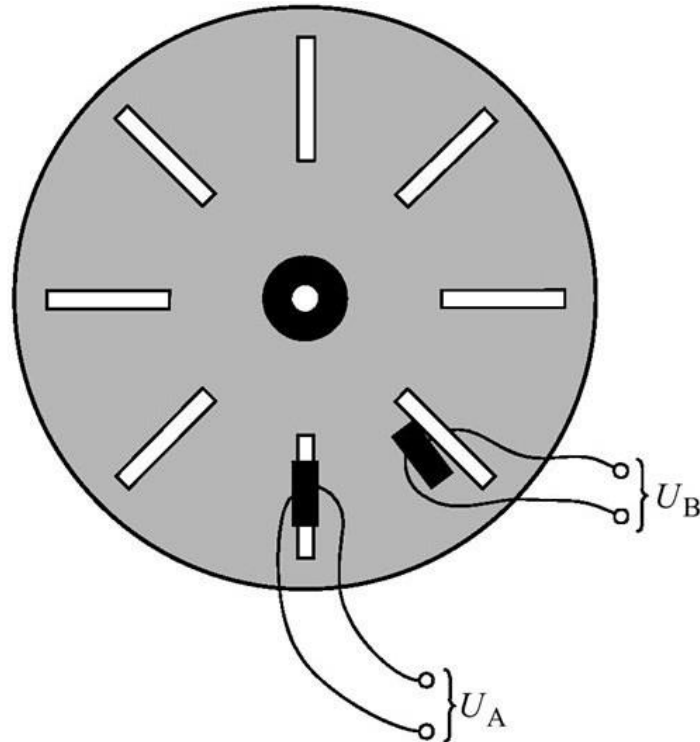
שאלה 7.9

מהי מהירותו של המנוע מדוגמה 7.11 אם נספרו 10 דפקים במשך 25msec?

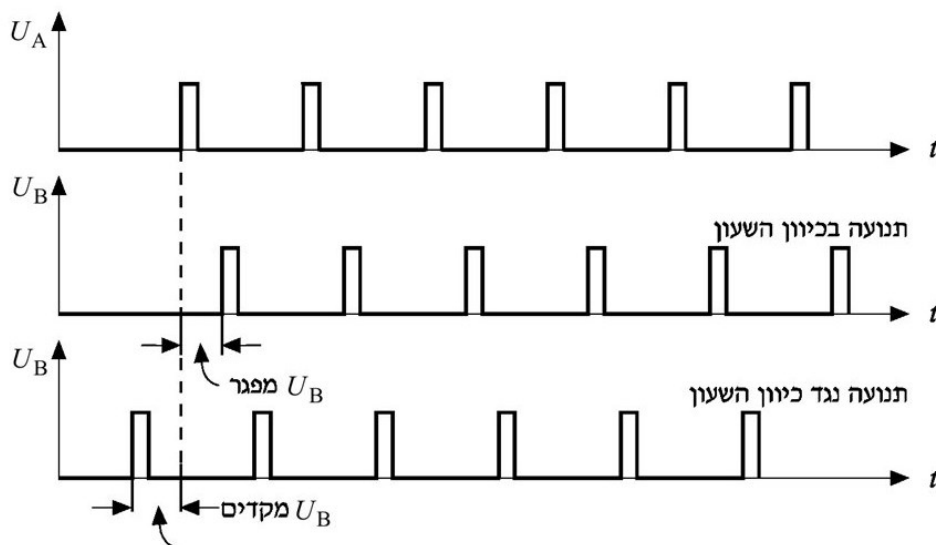
החיסרון בשיטת מדידה זו הוא שלא ניתן לקבוע את כיוון הסיבוב של המנוע שאת מהירותו הזוויתית (או את מצבו) רוצים למדוד. ניתן להתגבר על חיסרון זה על ידי הוספת גלאים. שימוש בשני גלאים A ו-B, כמתואר באיור 7.12, מאפשר לדעת את כיוון התנועה על פי זמן ההופעה של כל אחת משתי סדרות הדפקים שהגלאים יוצרים.

כאשר הגלגל מסתובב בכיוון השעון סדרת הדפקים של הגלאי B (U_B) מפגרת אחר סדרת הדפקים של הגלאי A (U_A). כאשר הגלגל מסתובב נגד כיוון השעון, הסדרה U_B מקדימה את הסדרה U_A . הדפקים המתקבלים מתוארים באיור 7.14.

איור 7.13: מקודד אופטי של סיבית אחת עם שני גלאים



איור 7.14: אותות מוצא של המקודד המתואר באיור 7.13

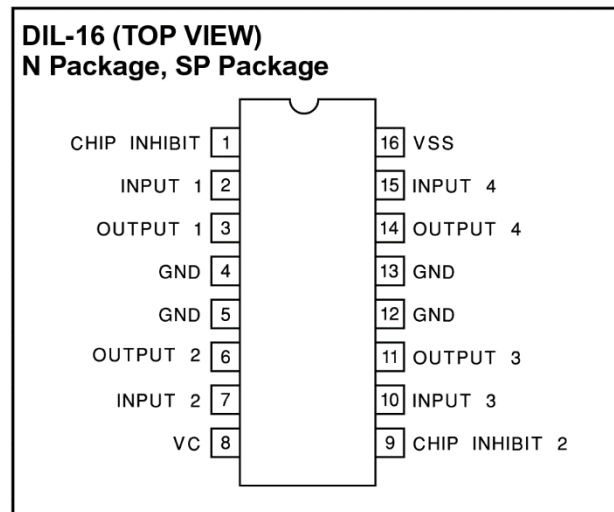


7.4.3 דוחף מנוע L293D

הרכיב המוכלל L293D מספק מערכת בתצורה הנקראת H-Bridge התומכת בהפעלה של שני מנועי DC בשתי מגמות סיבוב או הפעלה של מנוע צעד. הרכיב מסוגל לספק זרם הפעלה של עד 600mA וכולל דיודות שיכוך פנימיות לתופעת המעבר הקיימת בעומס השראי (סלילי המנוע).

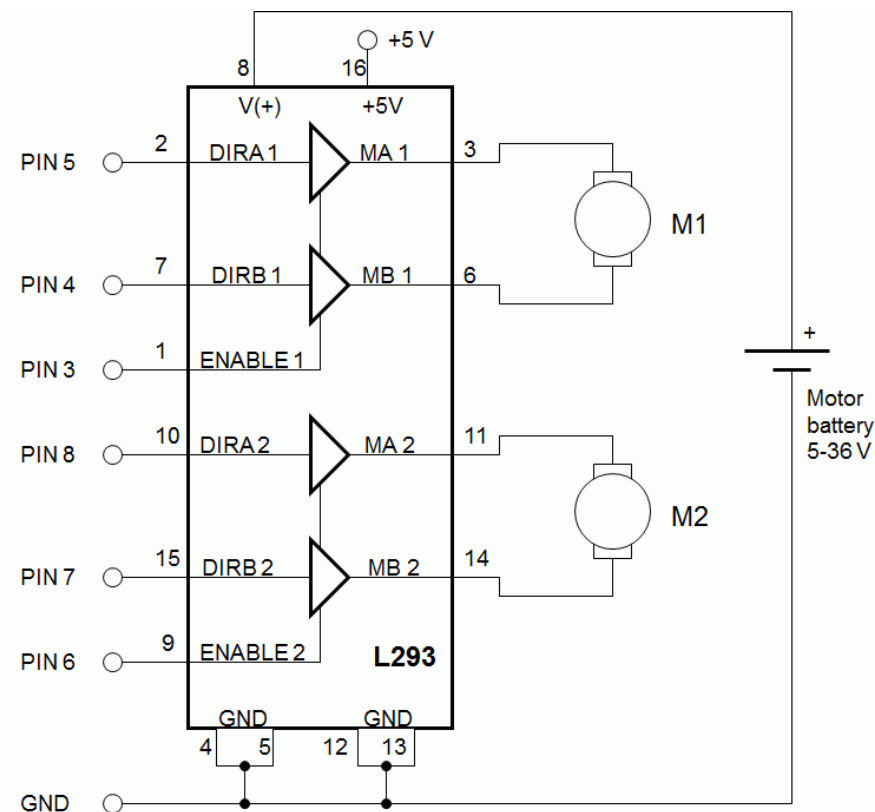
איור 7.15 מציג את דיאגרמת ההדקים של הרכיב.

CONNECTION DIAGRAMS



איור 7.15: דיאגרמת ההדקים של הרכיב

איור 7.16 מתאר את אופן החיבור של שני מנועי DC אל הרכיב. כפי שניתן לראות באיור כל מנוע מחובר לזוג הדקי מוצא: MA1, MB2, MA2, MB2 בהתאמה. מתח ההפעלה למנוע מסופק באמצעות הדק $V+$ ומתח הבקרה הלוגית מסופק להדק 5V. הדקי המבוא DIRA1, DIRB2 שולטים בכיוון ההפעלה של המנוע המחובר להדקים MA1 ו-MB1 והדקי מבוא DIRA2, DIRB2 שולטים בכיוון ההפעלה של המנוע המחובר להדקים MA1 ו-MA2. הפעלה של כל מנוע מתבצעת באמצעות הדקים ENABLE1 ו-ENABLE2).



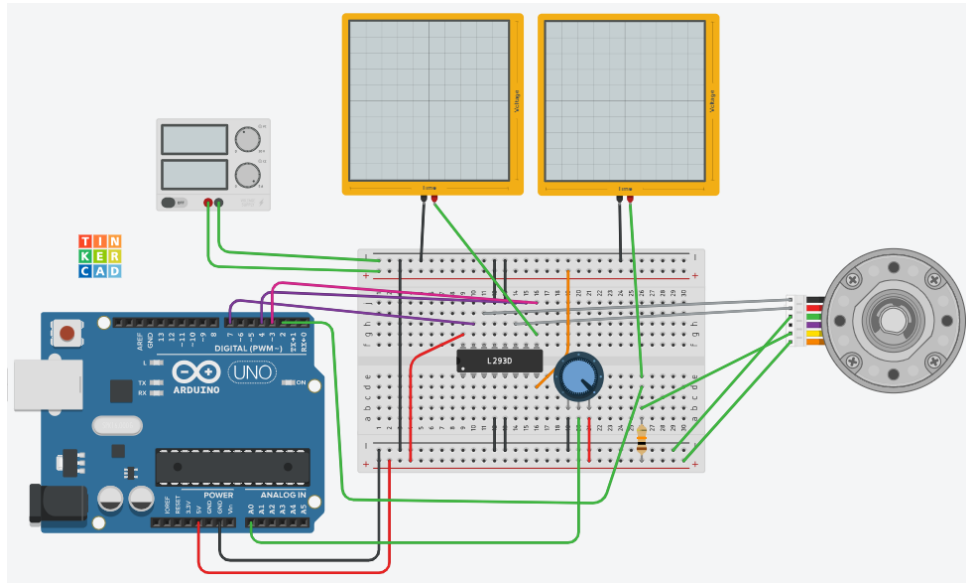
איור 7.16: סכמת חיבורים של שני מנועי DC

הטבלה שלפניכם מתארת אופני ההפעלה של מנוע ביחס למבואות הבקרה.

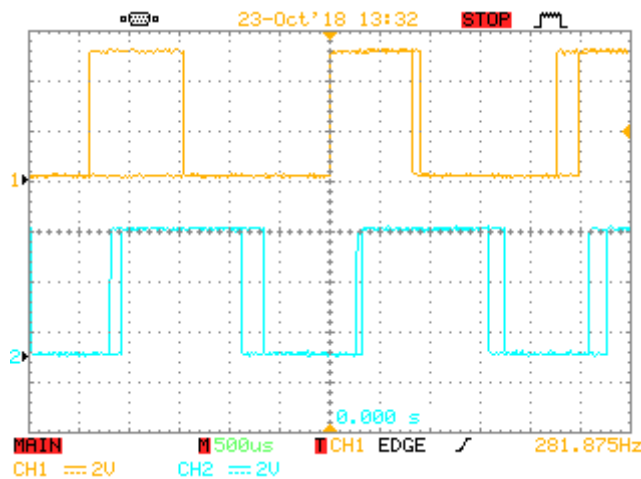
פעולה	DIRB	DIRA	ENABLE
סיבוב בכיוון השעון CW	High	Low	High
סיבוב נגד כיוון השעון CCW	Low	High	High
עצירה מהירה	Low	Low	High
עצירה מהירה	High	High	High
עצירה בתנועה חופשית	x	x	Low

דוגמה 7.5 – מערכת לבקרת מהירות מנוע DC בשיטת PWM באמצעות מקודד (encoder) אופטי

איור 7.17 מתאר רכיבי בקרת מהירות של מנוע לזרם ישר (DC), המנוע כולל מקודד אופטי דו-ערוצי, ואיור 7.18 מתאר את אותות המוצא של המקודד האופטי כפי שנמדדו במשקף תנודות.



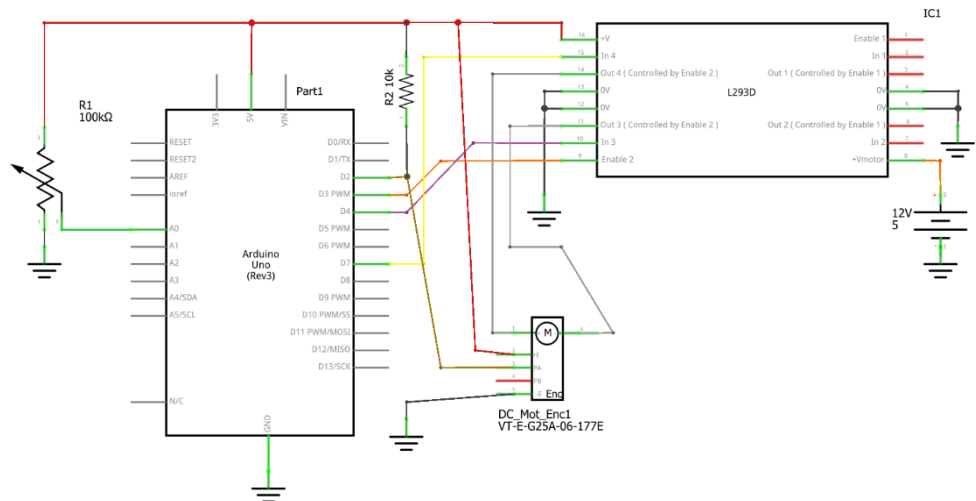
איור 7.17: רכיבי מערכת בקרת המהירות של מנוע זרם ישר עם מקודד אופטי



איור 7.18: אותות המוצא של המקודד האופטי

איור 7.18 מציג את אותות המוצא של הערוצים A (ערוץ CH1) ו-B (ערוץ CH2). ניתן להבחין כי בין אותות הערוצים קיים הפרש מופע המאפשר לזהות את כיוון הסיבוב של המנוע. תדירות האות מאפשרת לחשב את מהירות המנוע.

איור 7.19 מתאר את הסכימה החשמלית של המעגל. המנוע מחובר אל הרכיב L293 משמש להפעלת המנוע בכיוון ובמהירות הרצויה. הדקי הרכיב IN3 ו-IN4 משמשים לבחירת הכיוון והעצירה, והדק Enable 2 משמש לקביעת המהירות באמצעות בקרה לפי PWM. המנוע מוזן ממקור מתח חיצוני של 12V. קביעת המהירות הרצויה נקבעת באמצעות פוטנציומטר המחובר אל הדק A0 המשמש ככניסה אנלוגית.



fritzing

איור 7.19: סכימה חשמלית של המעגל

בדוגמה התייחסנו לבקרת מהירות בכיוון אחד בלבד, ולכן הסתפקנו במדידת אות מוצא יחיד של המקודד האופטי.

להלן תוכנית הבקרה:

```

// L293 מבואות בקרה של דוחף המנוע
const int IN3 = 4;
const int IN4 = 7;
const int EN2 = 3;

// הדקי מוצא של מקודד אופטי
const int ENCODEA = 2;

// משתני התוכנית
volatile long count = 0; // מונה מספר דפקים של המקודד
const int DELAY = 200; // זמן השהייה למדידת מהירות מצויה
int frequency; // משתנה המייצג את תדירות המהירות המצויה
int ref; // משתנה המייצג את תדירות המהירות הרצויה
int Speed = 128; // משתנה הקובע את ערך אות ההפעלה של המנוע:
50%

void setup() {
  pinMode(ENCODEA, INPUT);
  pinMode(IN4, OUTPUT); // input 4
  pinMode(EN2, OUTPUT); // enable 2
  pinMode(IN3, OUTPUT); // input 3
  analogWrite(EN2, Speed);
  // digitalWrite(EN2, HIGH); // משמש למדידת תדר מהירות מרבית
  digitalWrite(IN4, LOW); // בחירת כיוון פעולה של המנוע
  digitalWrite(IN3, HIGH);
  delay(1000)

```

```

}

void loop()
{
    count = 0;                // איפוס מונה הדפקים המתקבלים מהמקודד האופטי
    attachInterrupt(digitalPinToInterrupt(2), encoder, FALLING); // אפשרור פסיקה
    חיצונית

    delay(DELAY);             // השהייה לקריאת מצב הפסיקות
    detachInterrupt(digitalPinToInterrupt(2)); // חסימת פסיקה חיצונית
    frequency = count * 1000 / DELAY; // חישוב תדירות המהירות המצויה
    ref = analogRead(A0);      // קריאת מצב הפוטנציומטר
                                // משפט תנאי להשוואת הערך הרצוי לערך המצוי

    if (ref/1023.0 <= frequency/1415.0) { // אם הערך המצוי גבוה מהערך הרצוי
        Speed -= 5;                // הקטנת ערך ההפעלה למנוע
        if (Speed < 0)             // בדיקת גבול תחתון – מגבלה לערך מזערי אפס
            Speed = 0;
    }

    else {                       // הערך המצוי נמוך מהערך הרצוי
        Speed += 5;                // הגברת ערך ההפעלה למנוע
        if (Speed > 255)           // בדיקת גבול עליון – מגבלת ערך מרבי
            Speed = 255 ;
    }

    analogWrite(EN2, Speed) ;     // הפעלת המנוע עם ערך מתוקן
}

// שגרת (פעולת) פסיקה

```

```
void encoder () {
    count++;
}
```

הערות:

- הערך 1415 המופיע במשפט התנאי הוא תדר המתקבל בהדקי המקודד האופטי בעת הפעלת המנוע במהירות מרבית כפי שנבדקה באמצעות ההוראה הזו:
 - digitalWrite(EN2, HIGH);
- הערך 1023 מייצג את הערך המרבי המתקבל מתהליך המרת האות האנלוגי לספרתי (10 סיביות).

תרגול 

שאלה 7.10

זמן המדידה של הדפקים המתקבלים ממוצא המקודד האופטי הוא 200msec ונקבע באמצעות הקבוע DELAY.

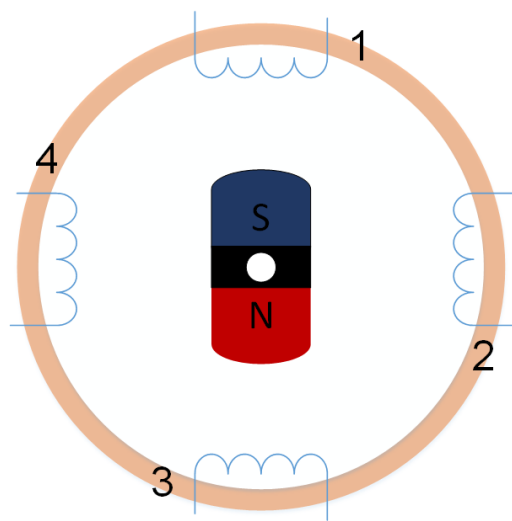
```
delay(DELAY);
```

תלמיד הציע להגדיל את הזמן לשנייה אחת כדי שמדידת התדר תהיה מדויקת.

- האם התלמיד צודק?
- כיצד שינוי זה ישפיע על פעולת בקרת המהירות מבחינת זמן התגובה הכולל מרגע שינוי דרישת המהירות עד לרגע שבו נקבל את המהירות הרצויה?

7.4.4 מנוע צעד

בסעיף 7.4.1 ניתנה סקירה של מנוע לזרם ישר. בסעיף נתייחס למנוע נוסף לזרם ישר הנקרא מנוע צעד (stepper motor). כאמור במנוע לזרם ישר רגיל הסטטור מכיל מגנט קבוע ליצירת שדה מגנטי ורוטור הכולל סלילים חשמליים המלופפים סביבו. לעומת זאת במנוע הצעד הסטטור כולל ארבעה סלילים חשמליים המלופפים בהיקף של המנוע והרוטור עשוי ממגנט קבוע. מכיוון שהרוטור עשוי ממגנט קבוע אין צורך בחיבור מתח, ולכן גם לא נדרשות מברשות. המתח מחובר לסלילים החשמליים הממוקמים בסטטור. מכיוון שמנוע הצעד אינו מכיל מברשות – שלא כמו במנוע רגיל – אין בלאי הנובע מהשימוש בו.



איור 7.20: מבנה עקרוני של מנוע צעד

עקרון הפעולה

כאשר מחברים מתח לסליל 1 בסטטור נוצר שדה מגנטי המושך אליו את אחד הקטבים של המגנט הקבוע ברוטור. כל עוד המתח נשמר מקובע הרוטור לשדה שיוצר סליל זה. הפעלת סליל 2 תגרום לתנועת המגנט לכיוונו. עקב כך תתקבל תנועה של 90 מעלות בכיוון השעון. הפעלת סליל 3 תגרום לתנועה נוספת של 90 מעלות בכיוון השעון. אם נפעיל את סליל 4 ולאחר מכן את סליל 1 נחזור אל נקודת ההתחלה, כלומר יושלם סיבוב בכיוון השעון.

הפעלת הסלילים בסדר הפוך, כלומר סליל 1, סליל 4, סליל 3, סליל 2, סליל 1 תגרום לסיבוב שלם בתנועה נגד כיוון השעון.

ניתן להפעיל את הסלילים באופן שונה. ניתן לקבוע תנועה בת 45 מעלות בכל פעם במקום 90 מעלות, ובכך להקטין את מרווח הצעד ולאפשר תנועה מדויקת יותר. הפעלה כזו נקראת

הפעלה בחצאי צעדים. לדוגמה, כדי להפעיל את הסלילים בכיוון השעון בחצאי צעדים נפעיל בכל פעם סליל אחד או שני סלילים סמוכים:

סליל 1 – 0 מעלות; סלילים 1 ו-2 (הרוטור יינעל בין שני הסלילים) – 45 מעלות; סליל 2 – 90 מעלות; סלילים 2 ו-3 (הרוטור יינעל בין שני הסלילים) – 135 מעלות; וכן הלאה.

הטבלה מתארת את סדר הפעלת סלילי מנוע הצעד. כאשר הערך 1 הסליל מופעל, וכאשר הערך 0 סליל מנותק.

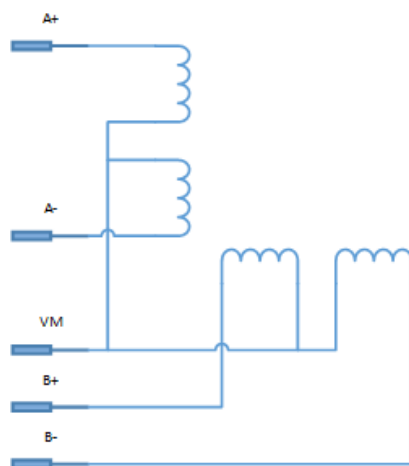
צעד	סליל 1	סליל 2	סליל 3	סליל 4
1	1	0	1	0
2	0	1	1	0
3	0	1	0	1
4	1	0	0	1

בפועל, הרוטור במנוע אינו מורכב ממגנט אחד, אלא מקטבים מגנטיים רבים. מספר הקטבים יקבע את הפסיעה של מנוע הצעד. לדוגמה, במנוע מסוים מותקנים קטבים מגנטיים כך שהמנוע יפעל ב-100 צעדים. כל פסיעה של המנוע תהיה 3.6 מעלות.

לצפייה בסרטון אנימציה המתאר את אופן פעולתו של המנוע לחצו [כאן](#).

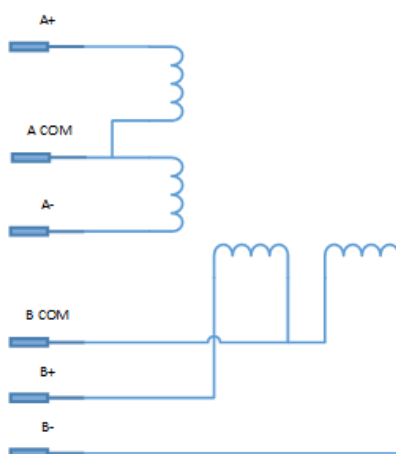
קיימות צורות שונות של חיבורי הסלילים במנוע צעד.

איור 7.21 מציג את אופן חיבור הסלילים במנוע בעל 5 חוטים.



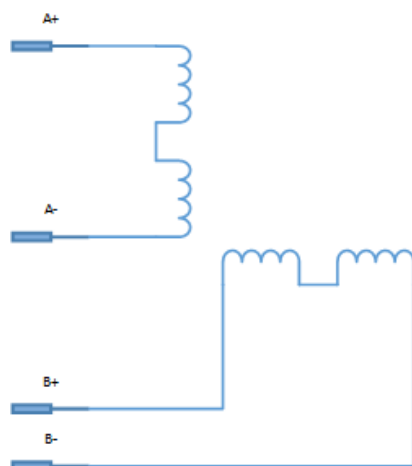
איור 7.21: מנוע צעד חד-קוטבי במבנה של 5 חוטים

איור 7.22 מציג את אופן חיבור הסלילים במנוע בעל 6 חוטים.



איור 7.22: מנוע צעד חד-קוטבי במבנה של 6 חוטים

איור 7.23 מציג את אופן חיבור הסלילים במנוע בעל 4 חוטים.



איור 7.23: מנוע צעד דו-קוטבי במבנה של 4 חוטים

7.4.5 מחלקת Stepper

המחלקה Stepper היא ספרייה הכוללת אוסף פעולות המאפשר להפעיל את המנוע צעד חד-קוטבי או צעד דו-קוטבי באמצעות ארבעה חוטים. כמו כן היא מאפשרת הפעלת מנוע צעד באמצעות שני קווי בקרה בלבד. לאפשרות זו נדרש מעגל חשמלי המרחיב את אפשרויות ההפעלה משני חוטים לארבעה חוטים. ביחידה זו נעסוק בחיבור של ארבעה חוטים בלבד.

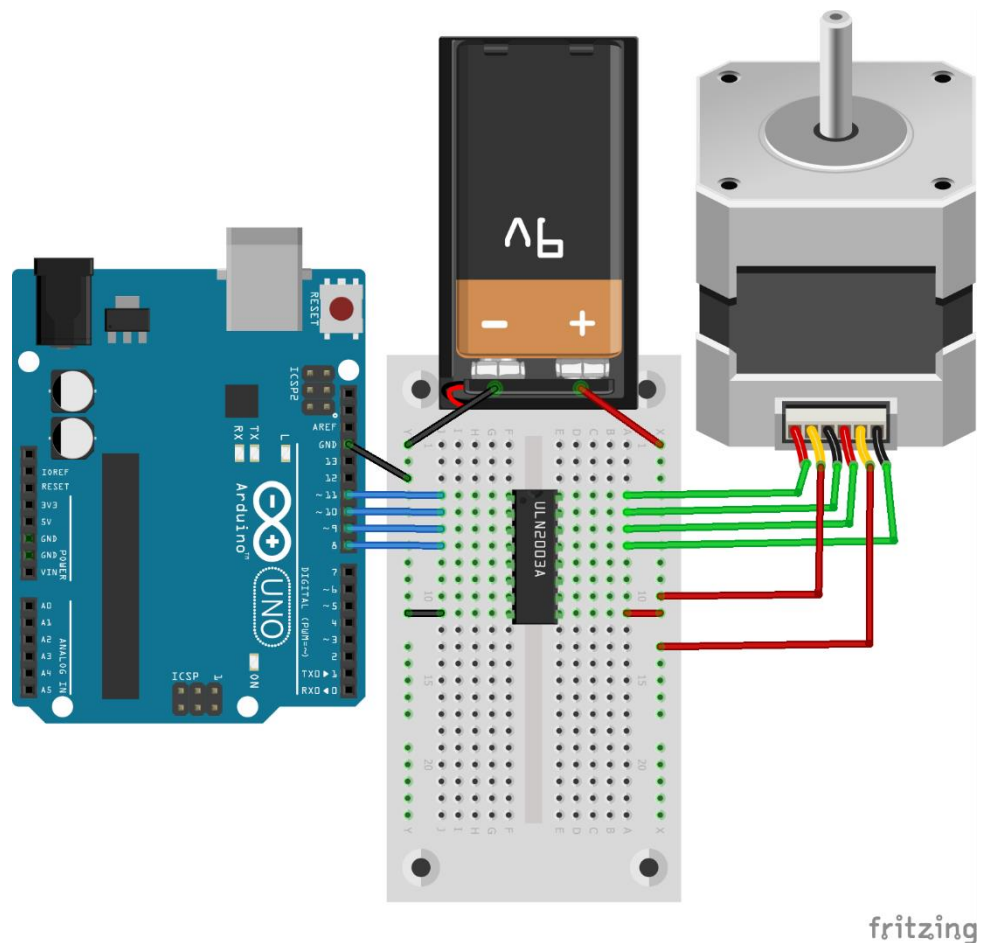
בטבלה מתוארות חלק מפעולות המחלקה (UML):

הפעולה	תיאור הפעולה
void Stepper(int, n, int pin1, int pin2, int pin 3, int pin4)	פעולה בונה המייצרת עצם מסוג מנוע צעד ומאתחלת את תכונות המחלקה: n – מספר הצעדים של המנוע הנדרשים להשלמת סיבוב שלם. pin1 עד pin4 – מספרי ההדקים בלוח הפתוח המבקרים את הסלילים 1 עד 4 בהתאמה.
void setSpeed(long speed)	פעולת קביעה של התכונה הקובעת את המהירות שבה המנוע יפעל ביחידות סיבובים לדקה (סל"ד – rpm).
void step(int n)	הפעולה מפעילה את המנוע במספר צעדים הנקבע על ידי הפרמטר n. n – מספר צעדי ההפעלה. אם ערכו של n חיובי יופעל המנוע בכיוון אחד, ואם ערכו של n שלילי יופעל המנוע בכיוון ההפוך. הנחה: מהירות ההפעלה נקבעת באמצעות הפעולה setSpeed.
int version ()	הפעולה מחזירה מספר שלם המייצג את גרסת המחלקה.

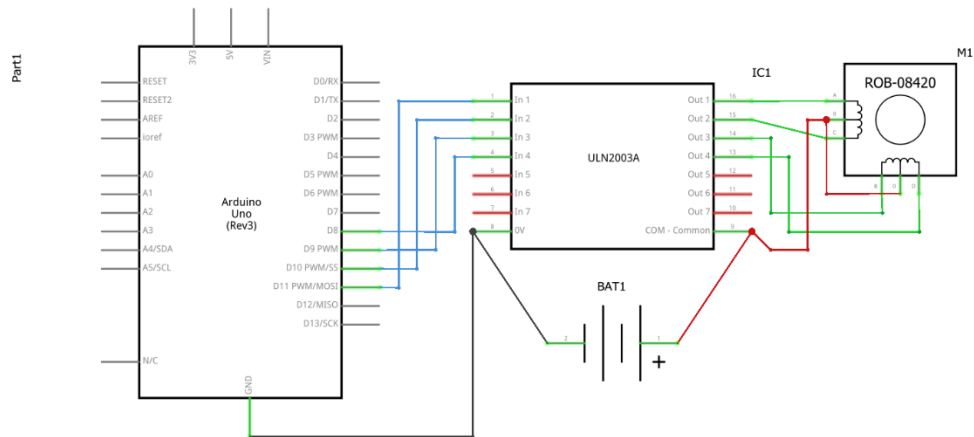
דוגמה 7.6 – הפעלת מנוע צעד בסיבוב מלא

בדוגמה זו נחבר מנוע צעד בעל שישה חוטים אל לוח הבקר באמצעות דוחף זרם

המנוע מחובר אל לוח הפיתוח לפי תרשים חיבורים המתואר באיור 7.22. סלילי המנוע ממותגים באמצעות דוחף זרם ULN2003 המכיל 7 דוחפי זרם. כל דוחף זרם מיושם באמצעות שני טרנזיסטורים המחוברים בצורה הנקראת צמד דרלינגטון. תרשים החיבורים מתואר באיור 7.24 והתרשים החשמלי מתואר באיור 7.25.



איור 7.24: תרשים חיבורים של מנוע צעד חד-קטבי ללוח הפיתוח



fritzing

איור 7.25: תרשים חשמלי של מנוע הצעד ללוח הפיתוח

להלן נתון קוד התוכנית

```
#include <Stepper.h>

// יצירת עצם למנוע צעד

// המנוע מחובר להדקים 8, 9, 10, 11

// מספר הצעדים של המנוע הוא 200

Stepper myStepper(200, 8, 9, 10, 11);

void setup() {

    // קביעת המנוע לשישים סיבובים לשנייה

    myStepper.setSpeed(60);

}

void loop() {

    // ביצוע 200 צעדים – סבוב שלם בכיוון אחד

    myStepper.step(200);

    delay(500);

    // ביצוע 200 צעדים – סבוב שלם בכיוון הפוך

    myStepper.step(-200);
}
```

```
delay(500);
}
```

תרגול 

שאלה 7.11

שנו את התוכנית מדוגמה 7.6 כך שהמנוע יפעל במהירות של 180 סל"ד ויבצע תנועה של 90 מעלות בכל כיוון עם השהייה של רבע שנייה בכל החלפת כיוון פעולה.

7.4.5 חיישן זווית

חיישן הזווית הוא התקן המחובר אל ציר הסיבוב של המנוע ומספק מידע על התנועה הזוויתית של המנוע. מקודד הציר האופטי, שהכרנו בסעיף 7.4, יכול לשמש למטרה זו על ידי ספירת מספר החריצים. בסעיף זה נציג דרך נוספת למדוד זווית באמצעות פוטנציומטר. כפתור הפוטנציומטר (knob) מחובר אל ציר המנוע באופן ישיר או באמצעות תמסורת של גלגלי שניים. לפוטנציומטר קיימת מגבלת תנועה ועל כן נגביל את תנועת מנוע הצעד בהתאם למגבלת הפוטנציומטר (קיימים פוטנציומטרים בעלי מספר סיבובים). איור 7.26 מתאר את מבנהו העקרוני של פוטנציומטר.



איור 7.26: מבנה עקרוני של פוטנציומטר

המתח המתקבל מהדק הזחלן של הפוטנציומטר הוא יחסי לתנועה הזוויתית של ציר המנוע. להלן ביטוי המתאר את המתח בין הדק הזחלן בפוטנציומטר לינארי ביחס לזווית המרבית של תנועת הכפתור:

$$V_{knob} = V_{CC} \frac{\theta_{knob}}{\theta_{max}}$$

כאשר

V_{CC}	המתח המסופק לפוטנציומטר
θ_{max}	הזווית המרבית שבה הפוטנציומטר מסוגל לנוע
θ_{knob}	זווית התנועה של הכפתור ביחס לאחד מהקצוות שלו
V_{knob}	המתח הקיים בזחלן

בפוטנציומטרים רגילים הזווית המרבית היא 290 מעלות.

כדי ליצור תנועה זוויתית דו-כיוונית נמקם תחילה את הפוטנציומטר במחצית מהלך התנועה שלו. במקרה כזה המתח המתקבל בזחלן יהיה מחצית המתח המסופק לפוטנציומטר. מתחים הנמוכים ממחצית המתח המסופק יצינו זווית נגד כיוון השעון ואילו מתחים גבוהים ממחצית המתח המסופק יצינו זווית בכיוון השעון.

 תרגול

שאלה 7.12

נתון פוטנציומטר בעל זווית מרבית של 300 מעלות המחובר למקרו-מתח של 5v.

- מהו תחום המתח המתקבל בזחלן הפוטנציומטר?
- מהו מתח הזחלן בזווית של 100 מעלות?
- באיזו זווית ממוקם הזחלן אם התקבל מתח של 3v?

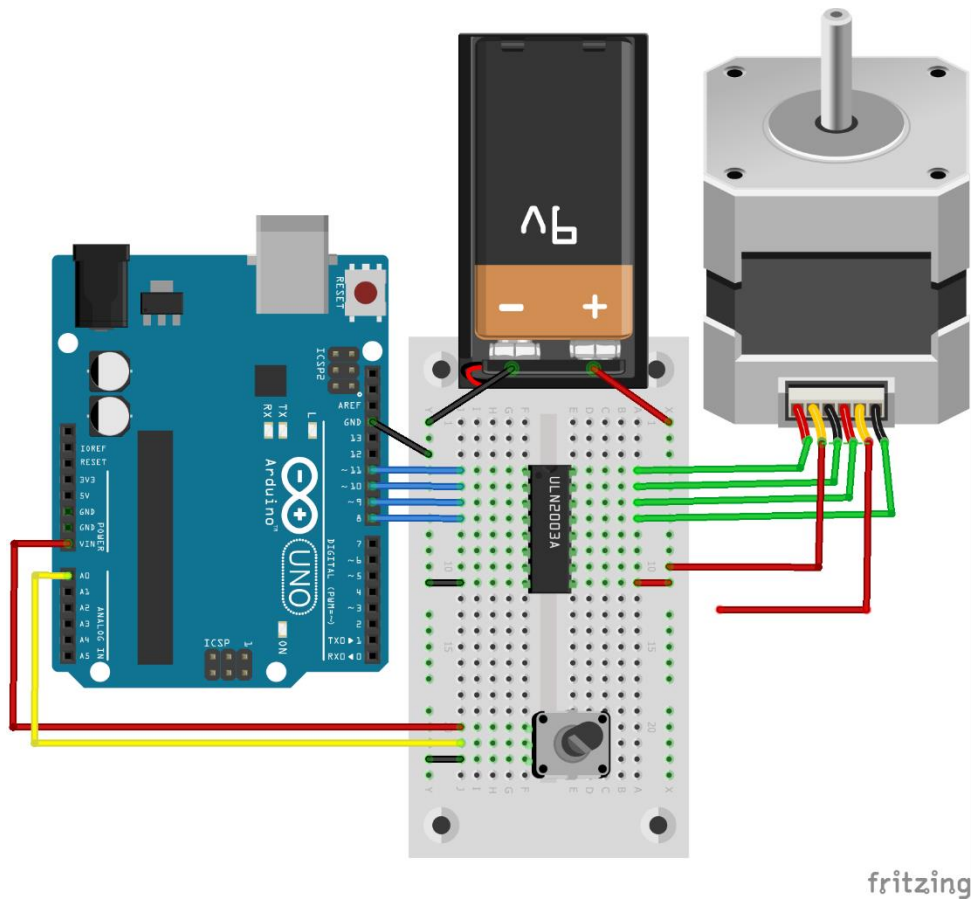
שאלה 7.13

נתון פוטנציומטר בעל זווית מרבית של 300 מעלות המחובר למקרו-מתח של 5v. ממקמים את הפוטנציומטר במרכז התחום ומגדירים את המתח המתקבל כזווית 0 מעלות.

- מהו תחום הזווית שהפוטנציומטר מודד?
- מהי הזווית אם נמדד מתח של 1.25v?
- מהי הזווית אם נמדד מתח של 3.75v?
- מהי הזווית אם נמדד מתח של 1v?

דוגמה 7.7 – בקרת זווית במנוע צעד

בדוגמה זו נציג חיבור של מנוע צעד שהציר שלו מחובר לכפתור הפוטנציומטר. איור 7.27 מציג את תרשים החיבורים (באיור לא מופיע החיבור בין ציר המנוע לבין כפתור הפוטנציומטר) ובאיור 7.28 מוצג התרשים החשמלי המתאים.



איור 7.27: תרשים חיבורי מנוע הצעד והפוטנציומטר ללוח הפיתוח


```

void loop() {
    rawVal = analogRead(A0);           // קריאת ערך הפונציומטר
    presentVal = map(rawVal,0,1024,0,300); // מיפוי הערך הגולמי לזווית
                                           // בדיקה אם היעד הושג
    if (presentVal>refVal)               // בדיקה אם הערך הנוכחי גבוה מהיעד
        stepper.step(2);                 // הזז 2 צעדים בכיוון אחד במגמה לצמצם את הפער
    if (presentVal<refVal)               // בדיקה אם הערך הנוכחי נמוך מהיעד
        stepper.step(-2);                // הזז 2 צעדים בכיוון הפוך במגמה לצמצם את הפער
}

```

ביאור התוכנית

בתוכנית נקרא הערך של הפוטנציומטר כערך בינארי בתחום מ-0 עד 1023 ומאוחסן במשתנה rawVal. ערך זה מייצג את המיקום היחסי של הפוטנציומטר ביחס לאחד מהקצוות שלו. קצה אחד מוגדר כאפס מעלות והקצה האחר מוגדר כ-300 מעלות. מתבצעת העתקה לינארית של תחום הערכים מ-0 עד 1024 לתחום הערכים מ-0 עד 300 באמצעות הפעולה map. הפעולה map מקבלת כפרמטרים את המשתנה rawVal, שבעבורו היא מבצעת את ההעתקה של הערכים מהתחום 0 עד 1024 לתחום 0 עד 300. תוצאת הפעולה מאוחסנת במשתנה presentVal.

פעולת הבקרה משווה את הערך הנוכחי presentVal לערך הרצוי refVal. אם הערך הנוכחי גדול או קטן מהערך הרצוי, מנוע הצעד נע 2 צעדים בכיוון המצמצם את הפער.

חסרונה של התוכנית המתוארת הוא בתנודות של המנוע המתקיימות כאשר הוא מגיע ליעד ונמצא סביב נקודת היעד. זאת משום שמשפטי התנאי גורמים להפעלה מתמדת של המנוע.

שאלה 7.14

- א. מהו המיפוי הדרוש כדי שהזווית שתחושב תהיה 150- מעלות בקצה אחד ו-150 מעלות בקצה השני?
- ב. לאחר המיפוי מחדש, מריצים את התוכנית עם הערך הרצוי 50 מעלות. מהי הזווית ביחס להגדרה הקודמת, שלפיה קצה אחד היה אפס והקצה השני היה 300 מעלות?

שאלה 7.15

- מה תהיה מידת ההשפעה על התנודות של המנוע אם נשנה את מספר צעדי התנועה מ-2 ל-5?

שאלה 7.16

- הציעו שינוי בתוכנית כך שהתנודות ייפסקו עם ההגעה ליעד.

סיכום פרק 7

בפרק 7 למדתם את הנושאים האלה:

1. מושגי יסוד במערכת בקרה
2. סיווג מערכות בקרה חוג פתוח וחוג סגור
3. תרשים מלבנים של מערכת בקרה בחוג סגור
4. מערכת בקרה ממוחשבת
5. מערכת בקרה מבוססת מיקרו-בקר
6. מערכת בקרה דו-מצבית עם חיישן ספרתי
7. מערכת בקרה דו-מצבית עם חיישן אנלוגי
8. מערכת בקרה רציפה עם חיישן אנלוגי
9. מנוע זרם ישר
10. מקודד ציר אופטי
11. דוחף מנוע L293D
12. בקרת מהירות מנוע DC בשיטת PWM באמצעות מקודד אופטי
13. מנוע צעד
14. מחלקת Stepper
15. חיישן זווית

VHDL

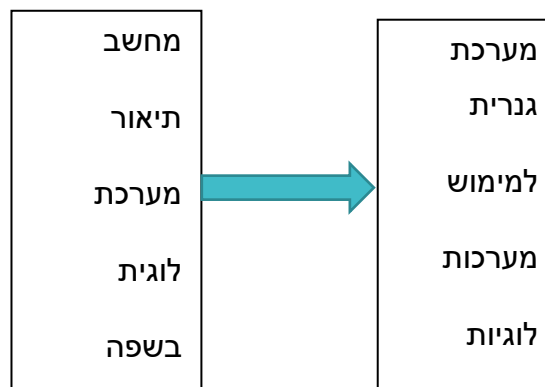
הקדמה

הערה לקורא: בהקדמה הזאת יש נושאים רבים שכדי להבין אותם צריך ניסיון וידע מוקדם בכמה תחומים מגוונים. אף-על-פי-כן אל תירתע מלקרוא אותה, מפני שיש בה את היסודות הראשונים שנחוצים להבנת התהליך.

ננסה להבין את הנושא של הרכיבים המתכנתים באמצעות שאלה **מאתגרת**, בהנחה שהתנסיתם בתכנות בשפה עילית כגון C או C# או כל שפה אחרת.

אנחנו רוצים לבנות מערכת לוגית שמבוקרת בעזרת מחשב. מהמערכת נדרשת **גנריות**, שמאפשרת תכנון של מערכת לוגית כלשהי על-פי דרישה. לדוגמה: יישום ערכים של xor ושל and או כל מערכת לוגית מורכבת אחרת.

מהר מאוד תוכלו לגלות שרכיבי ה-F.P.G.A. ושפת ה-V.H.D.L. (שאותם נתאר בהמשך) מעניקים גמישות מחשבית רבה ביישומים של מערכות לוגיות מורכבות מאוד.



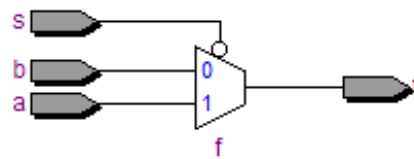
איור 1.1 מערכת חומרה של תוכנה

לצורך הדיון נניח שיש לנו את הדברים הבאים:

- א. מחשב עם יכולת של תקשורת.
- ב. יכולת לתכנת בשפה עילית.
- ג. מערכת חומרה עם יכולת של קליטת נתונים.
- ד. מערכת של זיכרון.

לפנינו ניצבים שני אתגרים: כיצד לתאר את המערכת הלוגית במחשב בשפה עילית, וכיצד לבנות מערכת **גנרית** לשם תכנון של כל מערכת **צירופית** שנרצה.

בשביל זה ניזכר במערכת רֶבֶב (multiplexer).



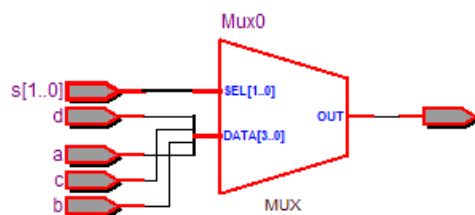
איור 1.2 רֶבֶב לשני מצבים

s	f
0	b
1	a

רֶבֶב הוא הבסיס למערכת גנרית. נדגים את זה בדוגמאות הבאות. למשל, באמצעות הגדרה של ערכים קבועים במשתנים ($a=0$; $b=1$), יתקבל שער NOT המהופך מהערך של s ($f=\text{NOT } s$).

דוגמה נוספת: מימוש השער XOR באמצעות multiplexer עם 4 כניסות.

לקביעה ($a=0$, $b=1$, $c=1$, $d=0$) תתקבל תוצאה על-פי טבלת האמת הבאה:



איור 1.3 רֶבֶב ל-4 מצבים

a	b	F
0	0	0
0	1	1

1	0	1
1	1	0

המבנה הזה יכול להיות הבסיס לכל שער לוגי, הוא תלוי כמובן בערכים הקבועים של טבלת האמת.

תארו לעצמכם מבנה של מערכת זיכרון, שאפשר "לשתול" בה את הערכים של הכניסות a,b,c,d.

ולא רק זה, אלא שאת הערכים של הכניסות הללו אפשר "לשתול" ממחשב אחר שנמצא בתקשורת עם המערכת.

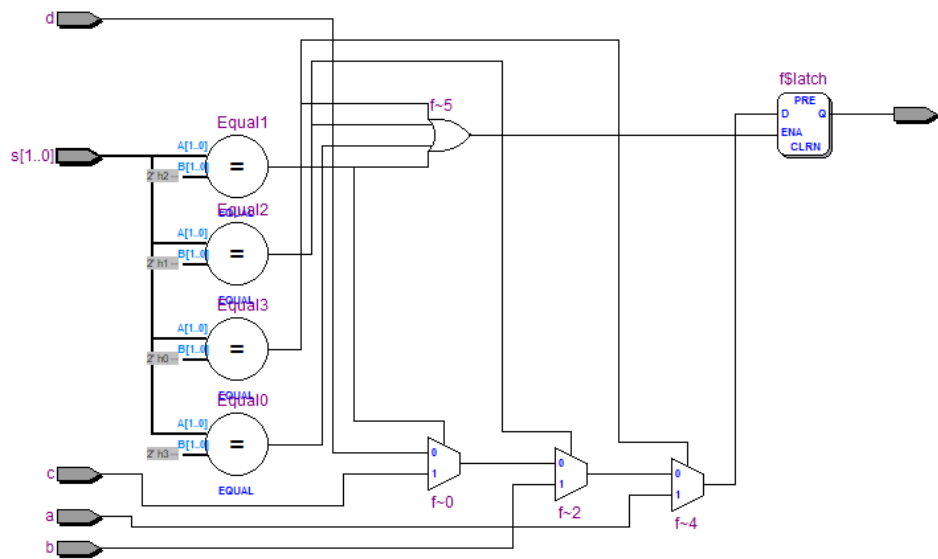
מערכת הזיכרון שאפשר "לשתול" בה ערכים נקראת Look Up Table (L.U.T.).

המערכת הזאת היא צירופית, כלומר ללא מערכת זיכרון.

דוגמה נוספת: בעבור השער and "נשתול" את הצירוף הזה: (a=0', b=0, c=0, d=1).

a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

נתאר לעצמנו מערכת זיכרון שקווי הכתובת שלה הם a,b והתוכן בכל כתובת מתאים לתוכן של f, אבל בכל מקרה לגופו. מבנה כזה נקרא Look Up Table (L.U.T.). מובן שאפשר להרחיב מערכות כאלה עד 32 קווי address. כמו שצוין קודם זוהי מערכת צירופית בלבד, ללא משובים פנימיים.



איור 1.4 מערכת לא צירופית. בסופה יש מערכת נעילה, שממומשת באמצעות תתי-רֶבָּבִים ומשווים.

עד עכשיו תיארנו את הצד החומרתי L.U.T., שבאמצעותו אפשר לממש מערכות לוגיות. אמנם תיארנו רק מערכות שפשוט מאוד להמחיש אותן, אך הקורא בוודאי מבין שלא הייתה זכות קיום לנושא הזה אם כל העשייה שלו הייתה מסתכמת בזאת בלבד.

נרחיב מעט את היריעה ונניח, שיש כמה מערכות צירופיות כאלה וצריך לחבר אותן ביחד. כל המערך הזה של תכנון מערכות לוגיות ושל הצירוף שלהן ביחד למערכת אחת נקרא:

שדות בני תכנות. (F.P.G.A.) Field Programmable Gate Array. התרגום של זה בעברית הוא: מערך של שדות בני תכנות.

נסכם את החלק החומרתי. יש לנו רכיב שאפשר לממש אתו מערכות לוגיות באמצעות מערכת הזיכרון L.U.T. אם מדובר במערכת לא צירופית אפשר להוסיף לה נועל (LATCH).

נטפל כעת בחלק של התוכנה. זהו אתגר לתאר מערכת xor או מערכת and בשפה עילית (שפת C או שפה אחרת). הקורא צריך להבין שלא פשוט לתאר חומרה בשפה עילית.

דוגמה:

```
If s=0
    f=1
else
    f=0
```

זה תיאור של שער not.

דוגמה נוספת:

```
If a=b
    f=0
Else
    f=1
```

זה תיאור של שער xor.

יש שתי שפות עיקריות שמתארות מבנים כאלה: Verilog, V.H.D.L.

Very high speed logic

Hardware

Description

Languish

המילים Very high speed logic רומזות על רכיבים מהירים מאוד. החשיבות של המהירות מתבטאת בעיקר בזמן אמת (real time). אבל המילים hardware description מלמדות על תיאור של חומרה ששונה מהלוגיקה הקלאסית, כדוגמת xor סינכרוני במידת הצורך.

נכון לשלב זה כבר למדנו על שפה שמתארת חומרה (V.H.D.L.) ואפילו על העיקרון של רכיב ה-F.P.G.A. נשארה לנו עוד משימה אחת בתהליך: התרגום של השפה העילית לטבלת האמת בעזרת ה-L.U.T.

ה-synthesizer מתרגם את ההתנהגות הדרושה ב-VHDL לערכי ה-L.U.T. שבתוך ה-F.P.G.A.

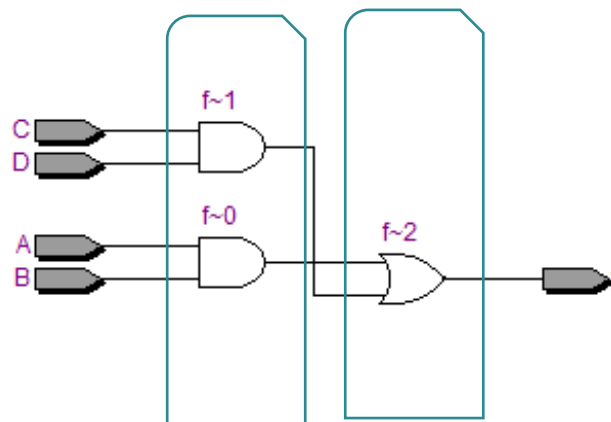
קיימות שתי צורות כדי לצרוב את הרכיב. באחת מהן הוא נצרב ישירות אל ה-F.P.G.A. – SRAM Object File (.sof). בשנייה נתוני הקובץ נשלחים באמצעות המחשב דרך חיבור של JTAG (נרחיב על זה בהמשך).

יתרון חשוב מאוד של רכיבי ה-F.P.G.A. הוא יכולת העיבוד שלהם במקביל (parallel processing).

נתאר כעת פונקציה: $f = A \text{ and } B \text{ or } C \text{ and } D$. המחשב יבצע את זה בצורה טורית: "decoding execute", "fetch", "and" – שלוש הפעולות שנדרשות בכל הוראת מחשב. אפשר לשפר זאת באמצעות pipe line, כלומר פעולת ההבאה (הפעולה השנייה – fetch) ופעולת הפענוח (הפעולה הראשונה – decoding) יתקיימו בעת ובעונה אחת.

עניין אחר הוא כל נושא ה-thread – הפעלת כמה מעבדים באותו הזמן.

ב-F.P.G.A. אפשר לבצע את $(A \text{ and } B)$ ואת $(C \text{ and } D)$ ביחד לאחר ביצוע פעולת OR.

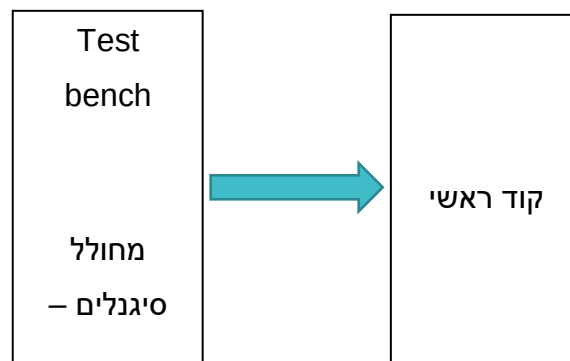


איור 1.5 ביצוע מקבילי בשני שלבים

נושא הסימולציה הוא יתרון חשוב של סביבת ה-V.H.D.L. היכולת לבדוק את התכנון לפני הצריבה שלו על הרכיב נובעת מסביבת פיתוח מהירה עם פוטנציאל גבוה מאוד באיתור תקלות אפשריות.

כללי השפה שחלים על VHDL שונים מהכללים שנדרשים בשביל צריבה.

בעיקרון מדובר בקוד שבודק את הקוד הראשי.



איתור תקלות

לפעמים הקושי להתחבר לסביבה הוא סיבה מספקת כדי לבחור לנו סביבת עבודה מתאימה יותר במידה של צדק. יודעי דבר יסכימו שלכל פרויקט מתלווים מדי פעם קשיים לא מעטים. השאלה שעומדת לפנינו היא מהם האמצעים לבדיקת הפרויקט או לאיתור תקלות בתוכו. לא מעט פרויקטים מתעכבים בפיתוח בגלל מחסור באמצעי בדיקה או באמצעים לאיתור תקלות. ברכיבי F.P.G.A. קיימים אמצעים כאלה שמשמשים לצורכי איתור, כגון logic analyzer.

בפרקים של **Hardware תכנות חומרה וסימולציה** נשים דגש רב על ההכרה של סביבת העבודה, ובייחוד על עניין רכישת הביטחון.

פרק 2: הכרת כלי סימולציה

תהליך של סימולציה נועד כדי לדמות מציאות במחשב. לפעמים בנייה של מערכת מורכבת ויקרה כל-כך לא מבטיחה תוצאות רצויות במיוחד, שכן מלכתחילה תמיד קיים הבדל בין התכנון ליישום. במשך שנים פותחו כלי סימולציה שמדמים כל מיני מציאויות.

סימולציה היא ניסיון לתאר מציאות, ובאמצעותה אפשר לבבא תוצאה אפשרית. לפיכך חשוב מאוד להכיר את המטרה שנודרש מאתנו לדמות, ובנוסף יש לדעת איך נזהה תוצאה בלתי-רצויה.

מצביאים נהגו להשתמש בעבר בסימולציות כדי לתכנן קרבות או כדי לבחון תרחישים אפשריים, וזה נהוג אף כיום.

סימולציה טובה מכסה את כל האפשרויות – אם לא, האויב יכול להפתיע. ובכל זאת יש למציאות דמיון רב, יותר מכל דמיון אנושי. במטרה ידועה או בסימולציה אפשר להתקרב למציאות רק במידה מסוימת, לעולם לא יהיה אפשר להגיע למציאות שלמה.

סימולציה שקרובה מאוד למציאות תתרום רבות ללא ספק לקיצור זמן הפיתוח של הפרויקט, ולכן נראה שיש בסביבה של FPGA יתרון על-פני סביבות אחרות מקבילות.

לצורך המחשה נניח שאנחנו רוצים לפתח מד-מרחק על-פי חיישן מרחק כדוגמת SRF04 (החיישן הזה מספק אות ריבועי לפי הפונקציה של מרחק העצם מהחיישן). אפשר לדמות בקלות רבה למדי את פונקציית המרחק ולבדוק את התכנון של מה שנועד להיות צרוב על רכיב ה-FPGA.

מכל האמור אפשר להבין שסביבת הסימולציה היא במחשב PC בלי שום קשר לרכיב הנצרב FPGA. ליתר דיוק אפשר להתחשב גם בסביבה הזאת ובהשפעותיה על התכנון, אולם כרגע מוקדם מדי לטפל בנושאים מורכבים שכאלה.

ראשית יש להבחין בין סימולציה להדמיה.

סימולציה: קוד מחשב שמדמה נתונים של המציאות הנבדקת, ובאמצעותו נבדק קוד התכנון שאותו צריך ליישם.

הדמיה: בתהליך הזה משוחזרת מציאות לא ידועה על-סמך חישובים מתמטיים, כמו במכשיר M.R.I. האתגר בהדמיה הוא לגלות את המציאות – לעומת הסימולציה, שמנסה לשחזר את המציאות.

נראה דוגמה נוספת מתחום הבקרה, שתמחיש לנו את ההדמיה.

ניתוח פונקציית תמסורת בלי לבנות אפילו מרכיב אחד מהמערכת, ואף יותר מזה – הרצת הסימולציה בתוכנה כגון MATLAB, חוסכים לנו בבניית המערכת ויותר מכול מקצרים את זמן הפיתוח.

מכיוון שהסימולציה פועלת על מכשיר ה-PC ולא צריך לצרוב אותה על הרכיב, שפת VHDL ושפת VRILOG מעניקות לה תחביר גמיש מאוד, והוא שונה בתכלית משפת VHDL שאכן זקוקה לצריבה.

הכרה של סביבת העבודה היא תנאי הכרחי כדי לרכוש ביטחון ברכיבים מתוכננים (F.P.G.A.). לכן יש לתרגל כמה שאפשר.

בפרק הזה נכיר את ה-test bench. יש בו איורים ברורים, שתפקידם הוא להקטין במידת האפשר את האפשרות לחוסר הצלחה.

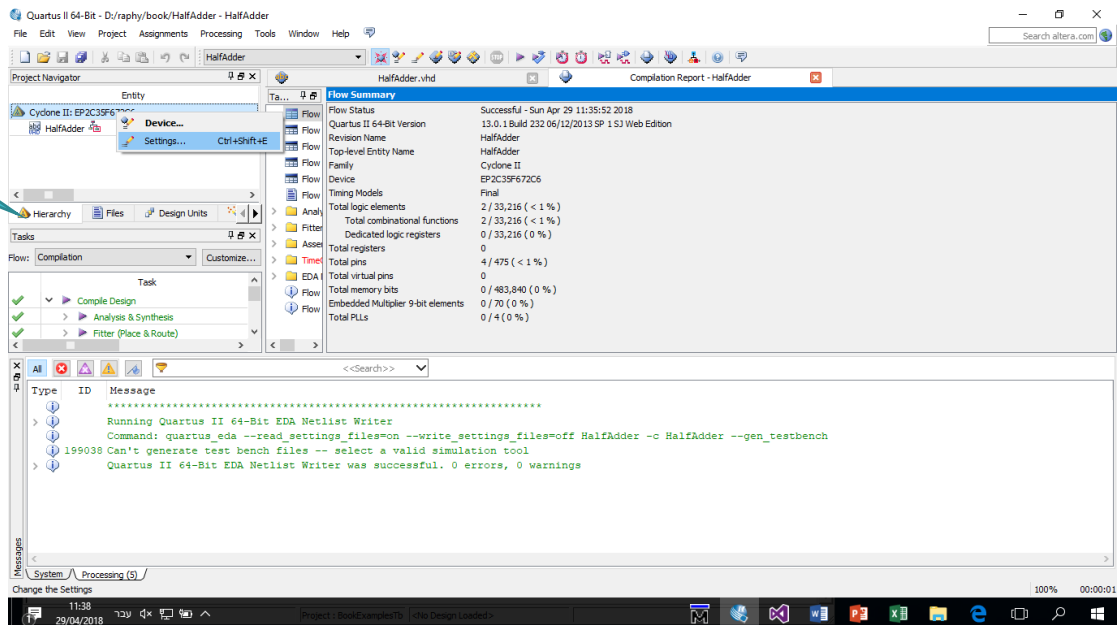
ברוכים הבאים לעולם יפה ומרתק!

בשורת החיפוש הקלידו Modelsim.

יצירת קובץ סימולציה בסביבת quartus:

1. הקישו על הלשונית hierarchy.

2. הקליקו על העכבר בצד ימין ובחרו ב-setting.

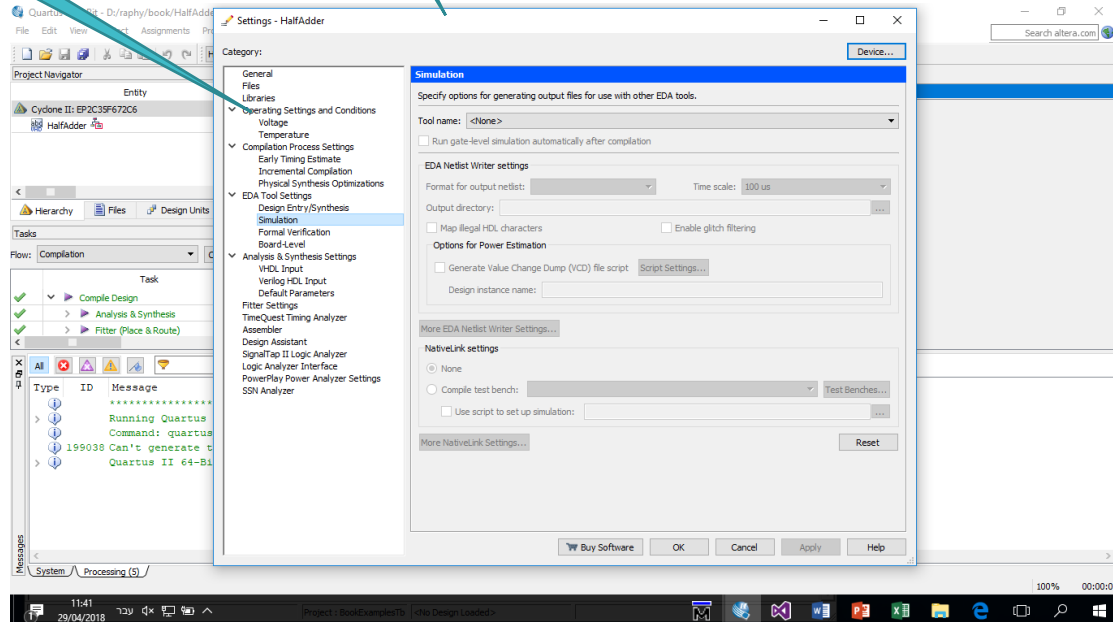


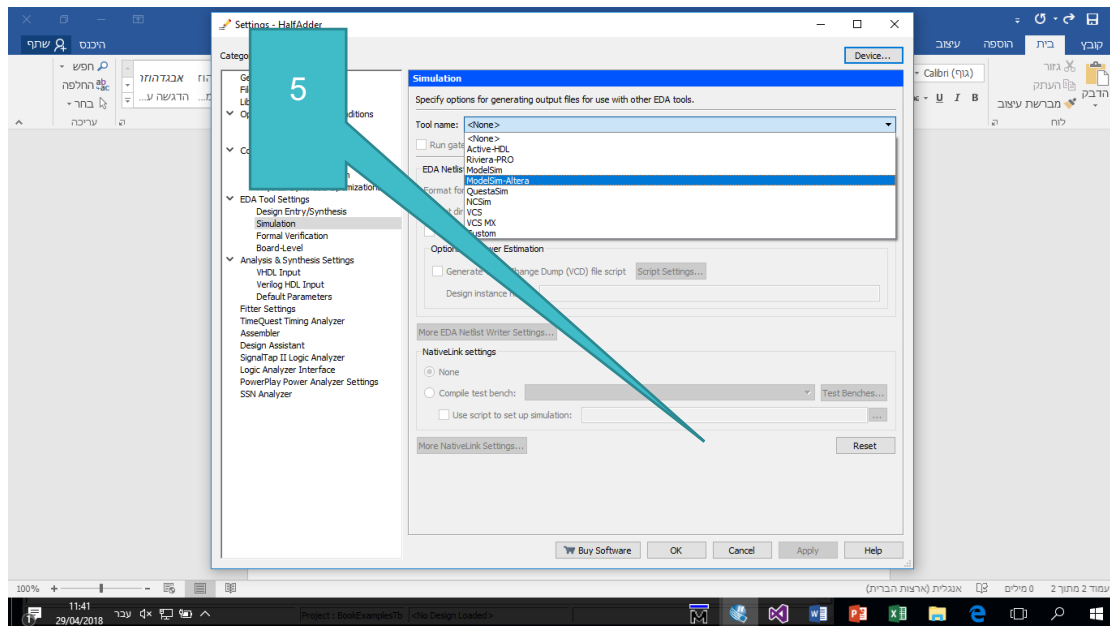
בחרו ב-simulation

4

3

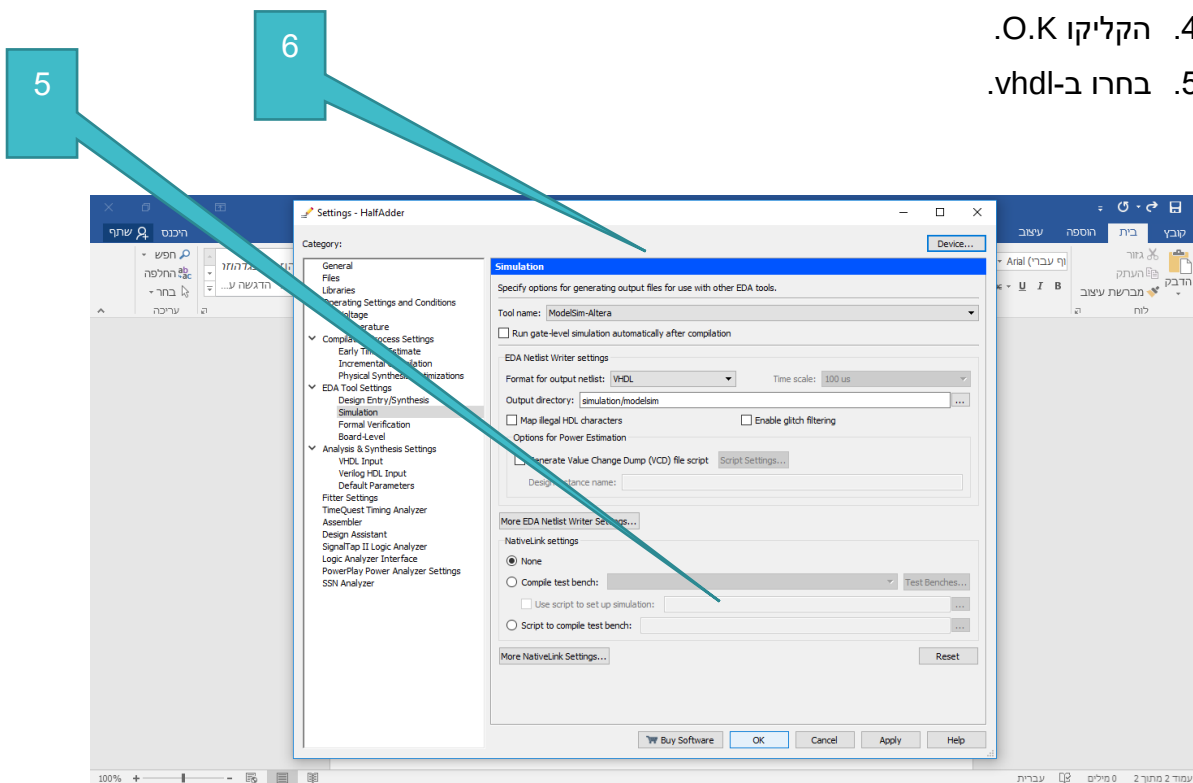
3. הקליקו ב-tool name על האפשרות altera-modelsim



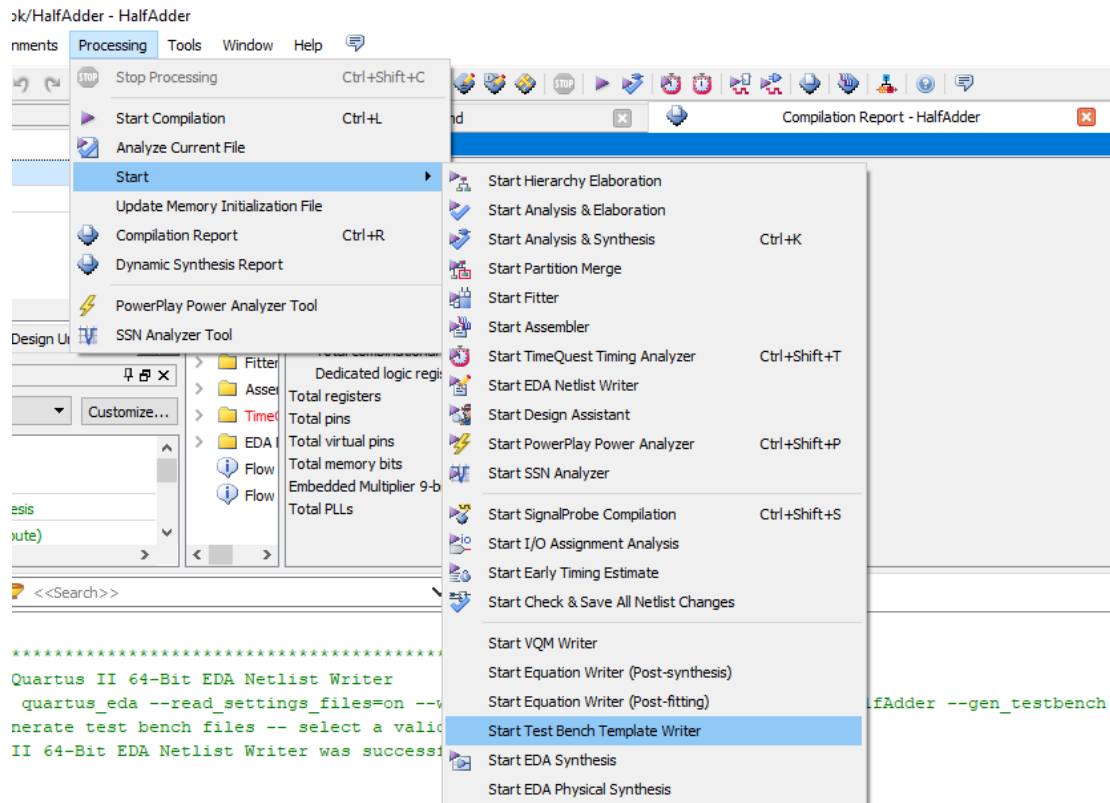


4. הקליקו O.K.

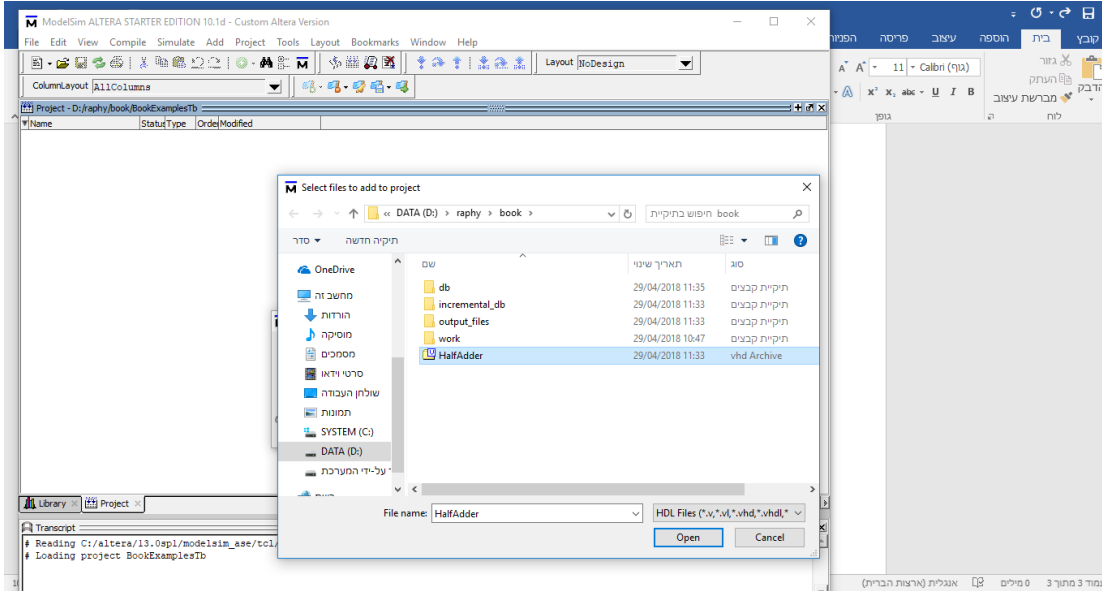
5. בחרו ב-vhdl.



6. מתוך האפשרויות בלשונית processing שבתפריט בחרו ב-start, לאחר מכן בחרו ב-.start test bench template writer

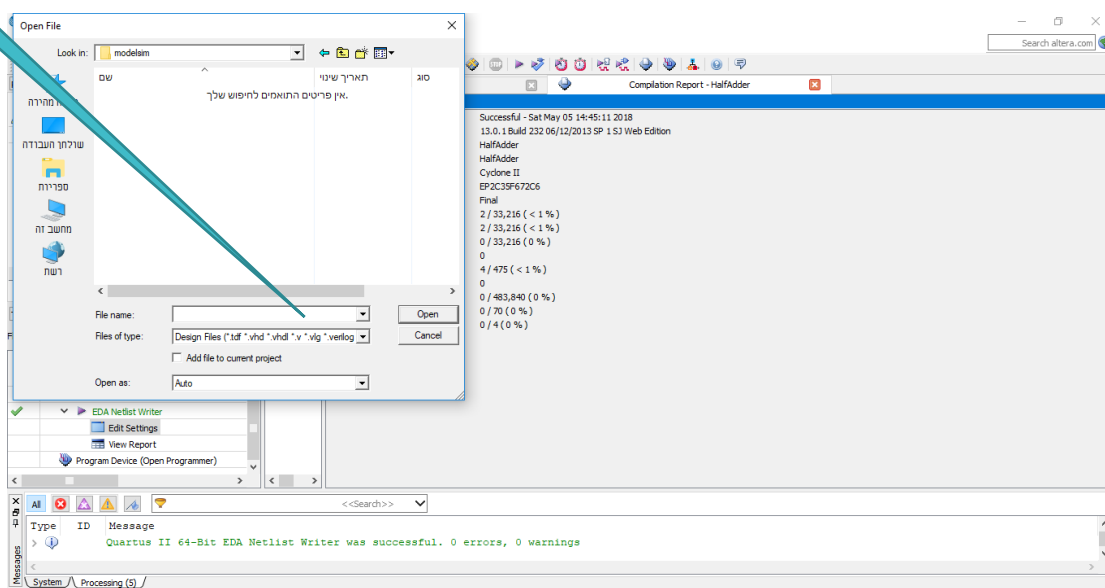


7. בתום ההרצה בחרו מהתפריט file את open, ולאחר מכן פתחו מספריית simulation את ספריית modelsim.
- File=>open=>simulation=>modelsim



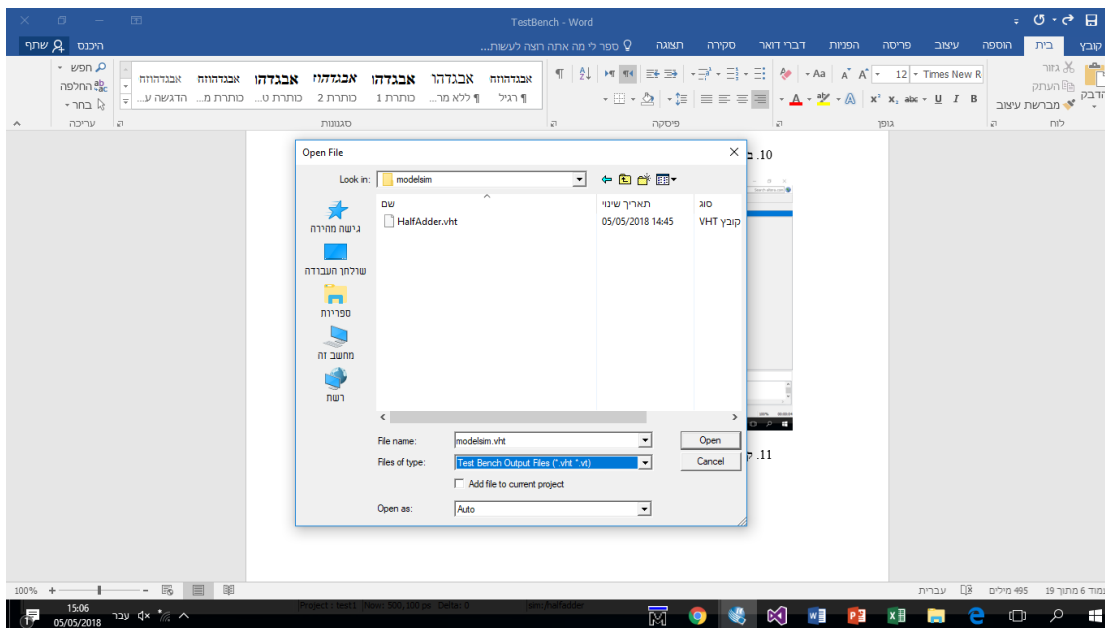
- בספרייה modelsim לא תמצאו כלום, לא לדאוג.
8. הקליקו על files of types ובחרו באפשרות .vht.

9



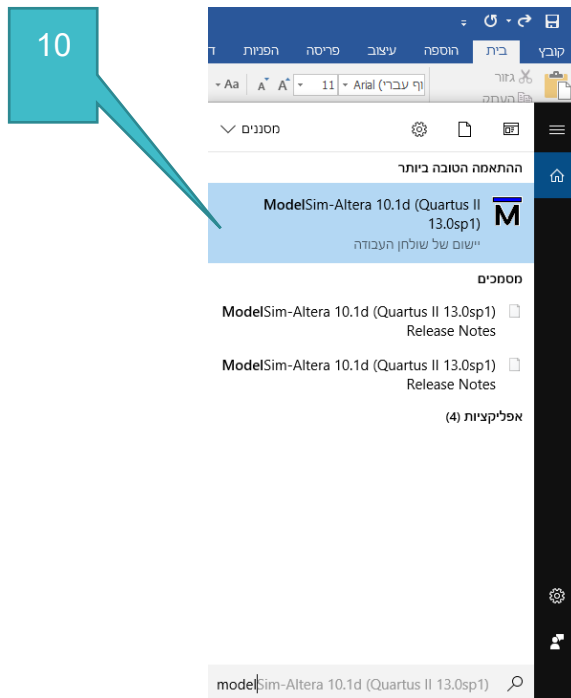
10. קובץ עם סיומת של vht אמור להופיע.

HalfAdder.vht

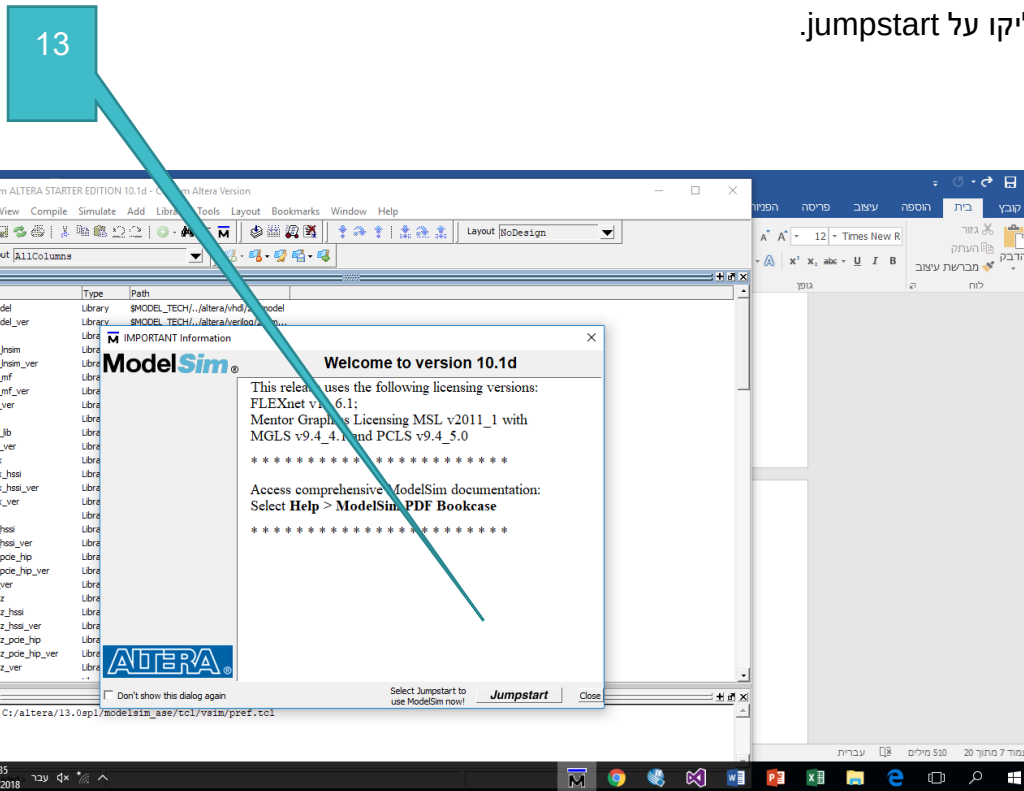


סביבת הסימולציה:

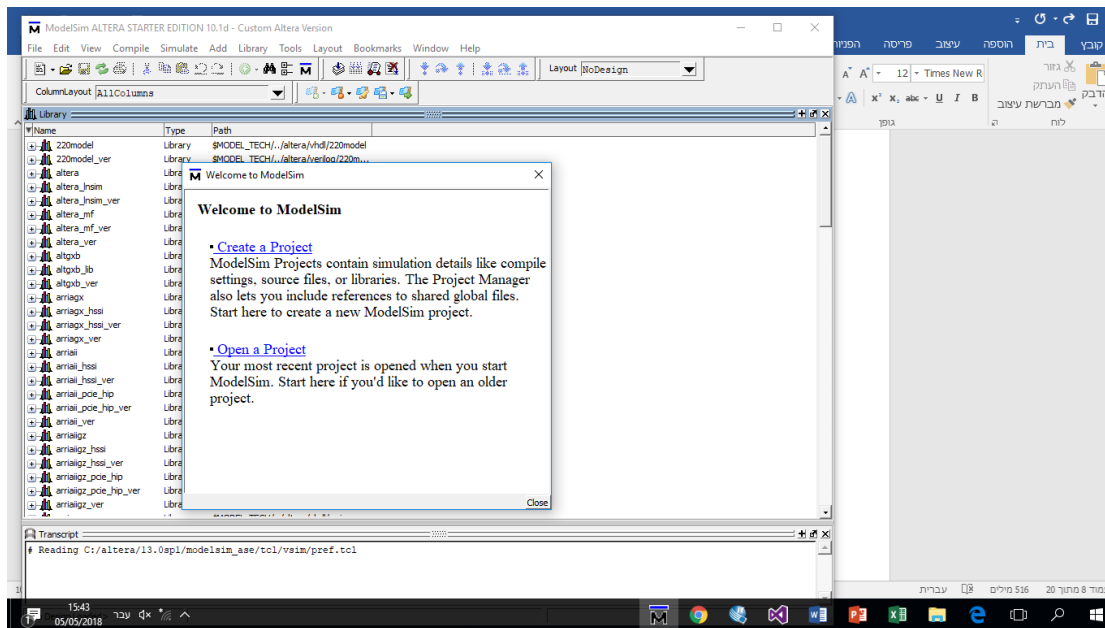
11. באמצעות מנוע החיפוש (search) חפשו את modelsim והקליקו על הצלמית שלו.



12. הקליקו על jumpstart.



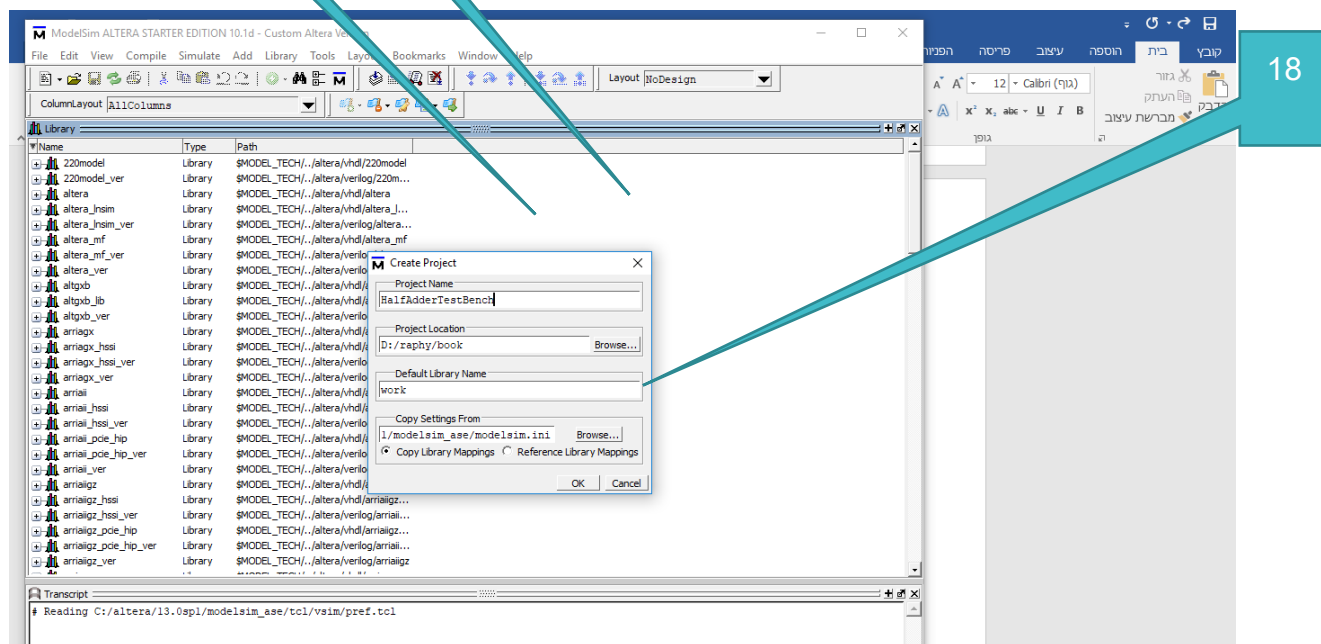
13. הקליקו על Create a project.



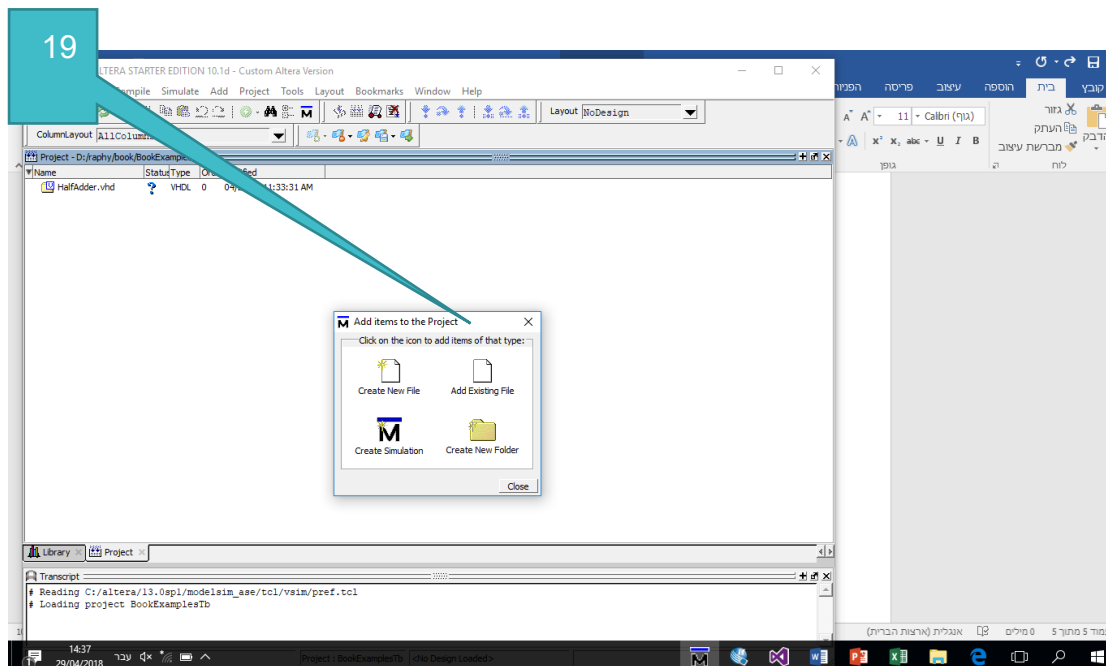
14. בחרו את המסלול שבו פתחתם את הפרויקט ב-quartus.

15. לצורך מעקב נקבע את שם הפרויקט: HalfAdderTB.

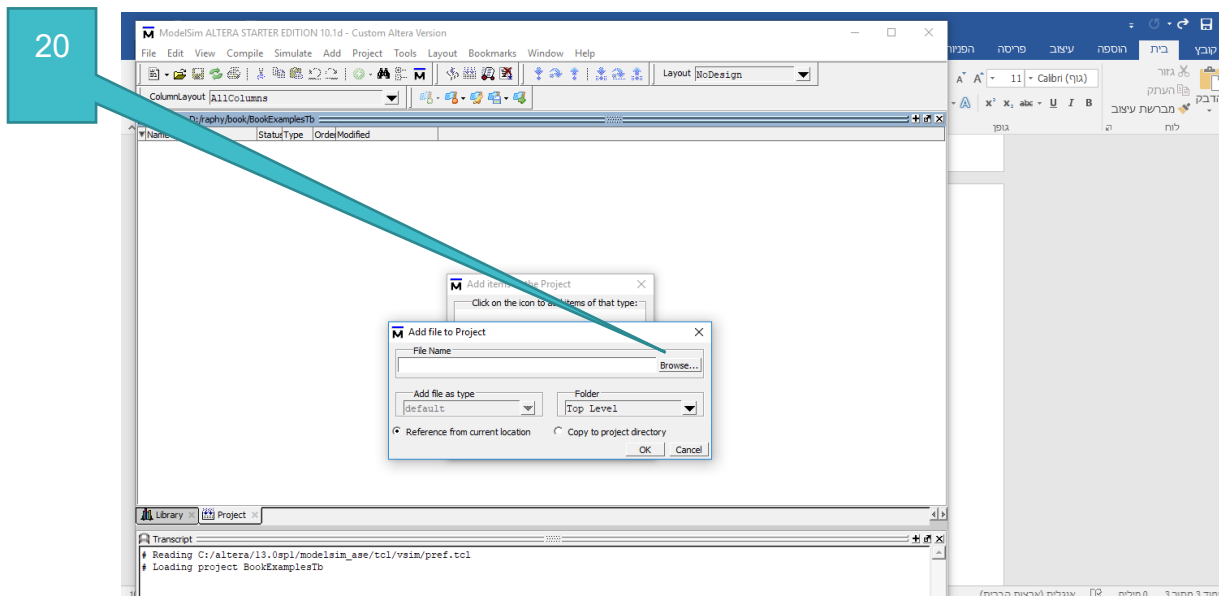
16. ספריית work נפתחת בתור ברירת מחדל, אין לשנות אותה.

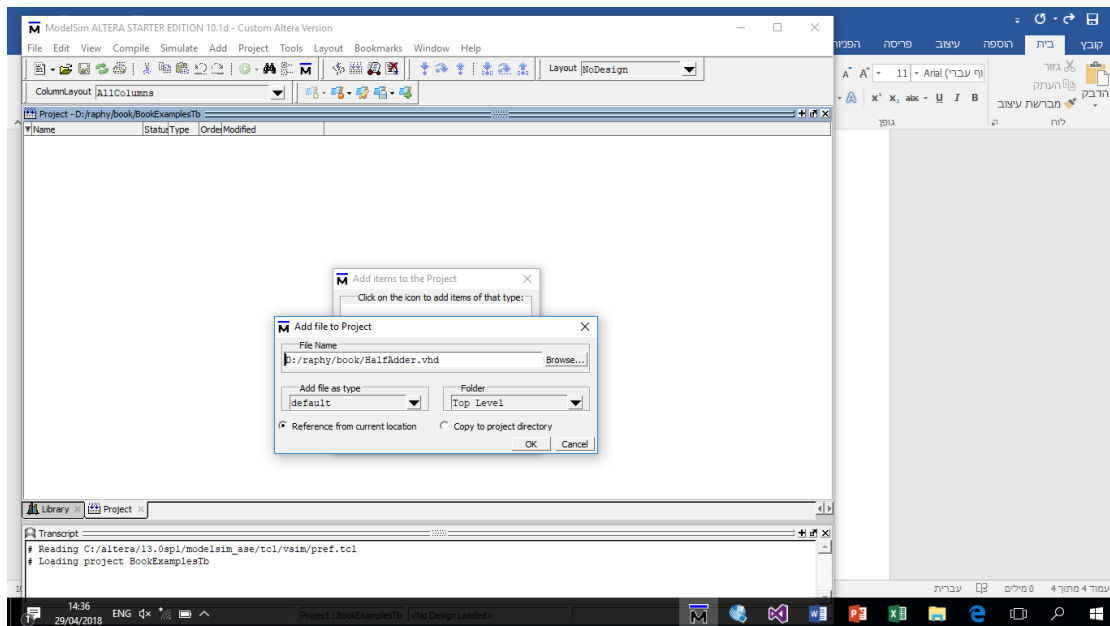


17. הקליקו על Add Existing Files והגיעו ל-HalfAdder.vhd וגם ל-HalfAdder.vht לפי הסעיפים שלעיל (מסעיף 10 והלאה).



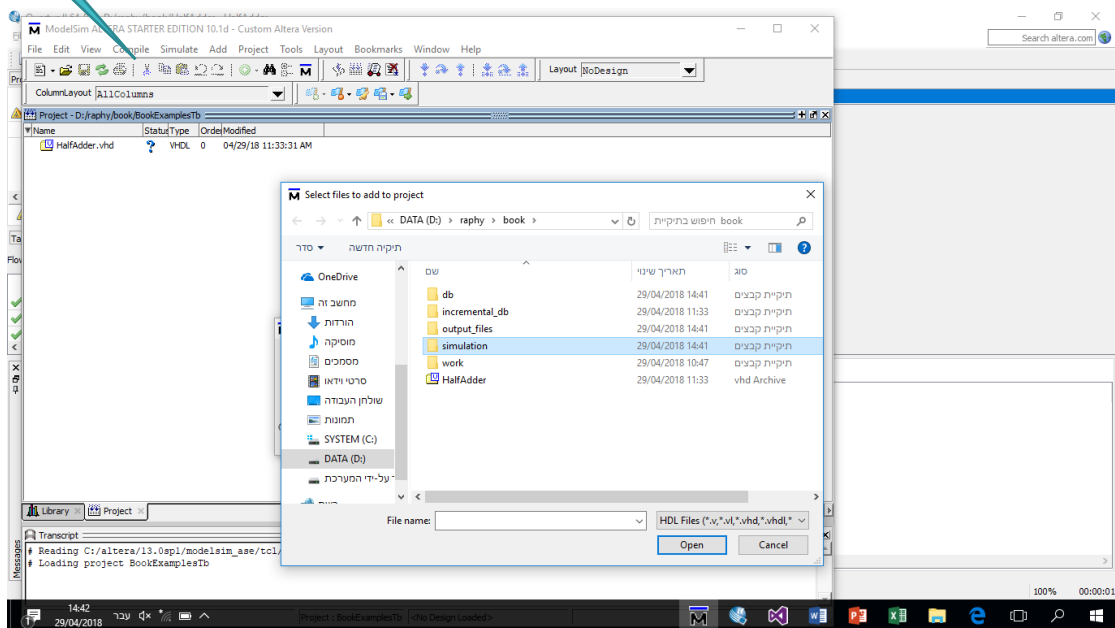
18. דפדפו למיקום הנכון של הקבצים באמצעות browse.



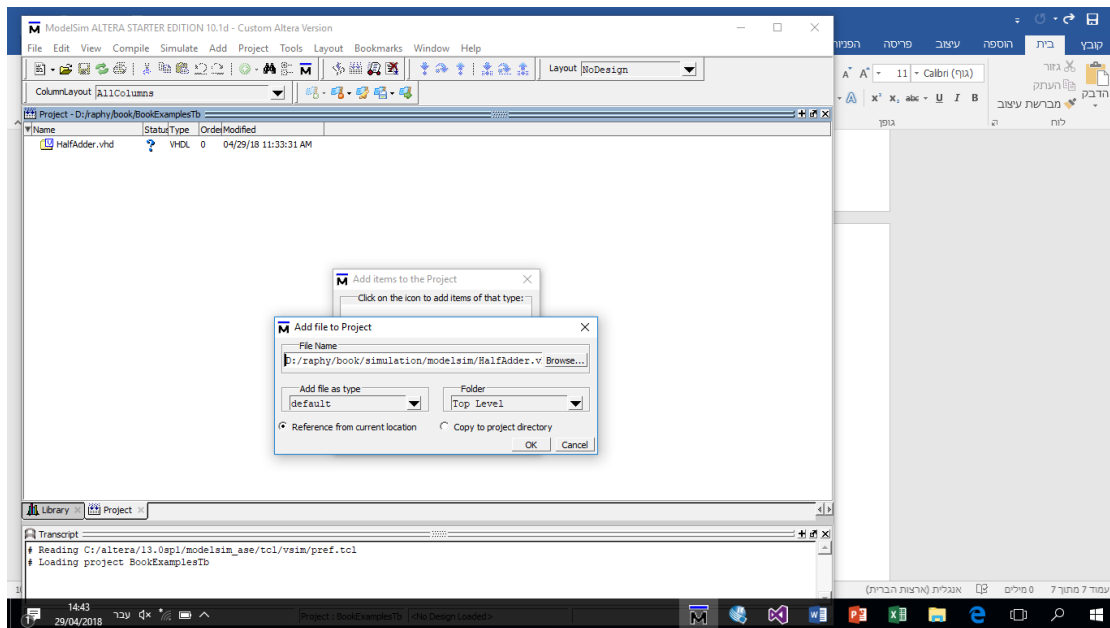


19. הוספה של קובץ vht מספריית simulation.

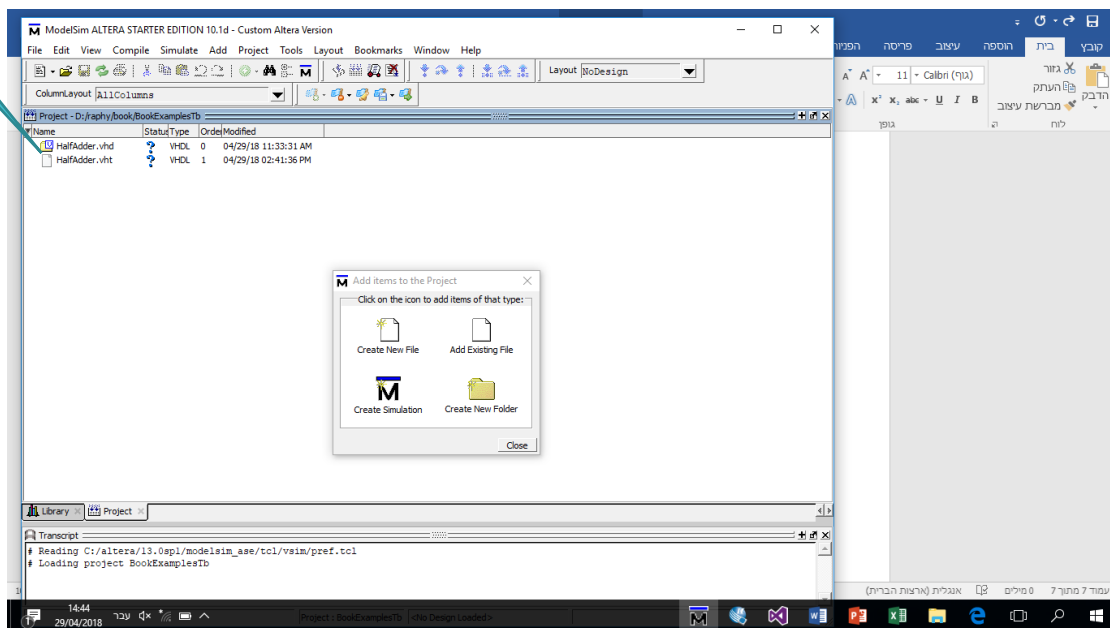
21



20. הוספה של קובץ vhd.

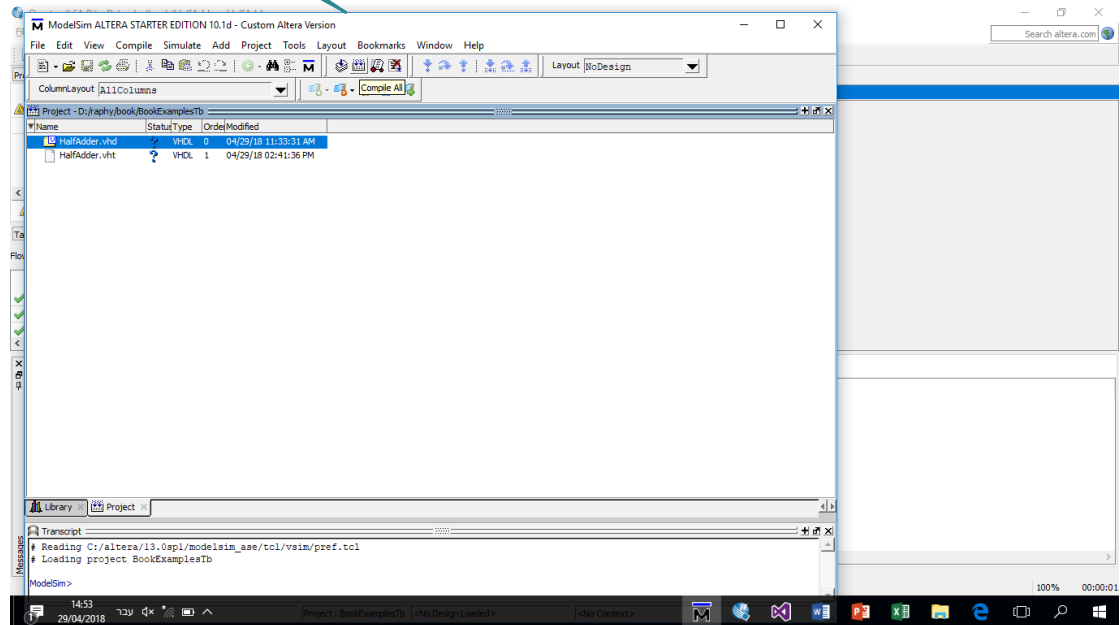


23



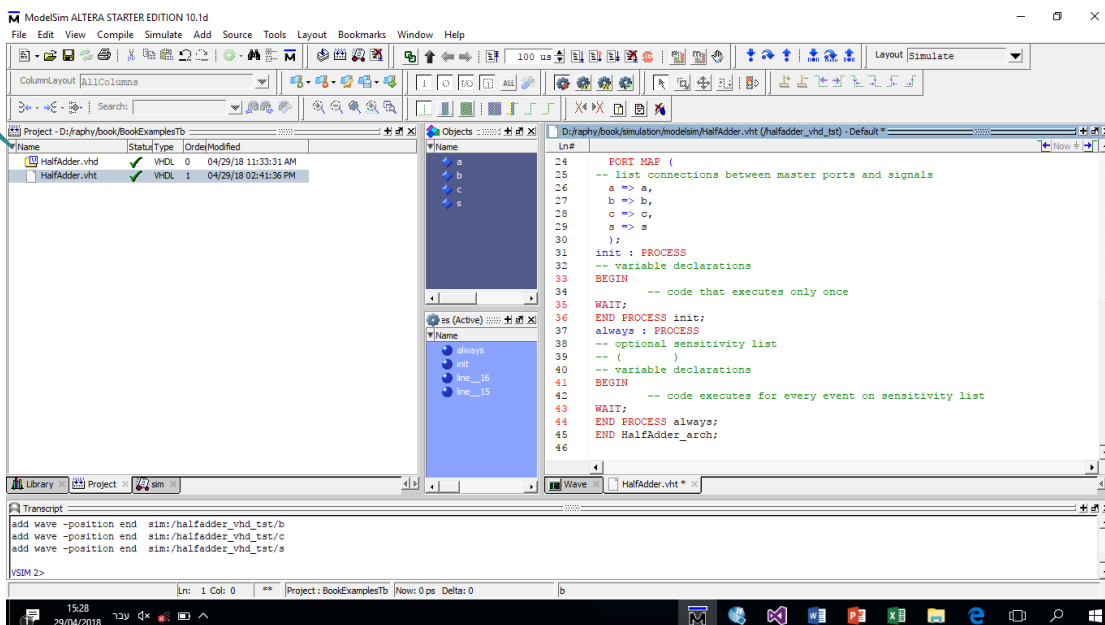
21. סמנו אחד מהקבצים באמצעות קליק.

22. הקליקו על compile all. העמידו את העכבר על הצלמית כדי לזהות את הצלמית המתאימה.

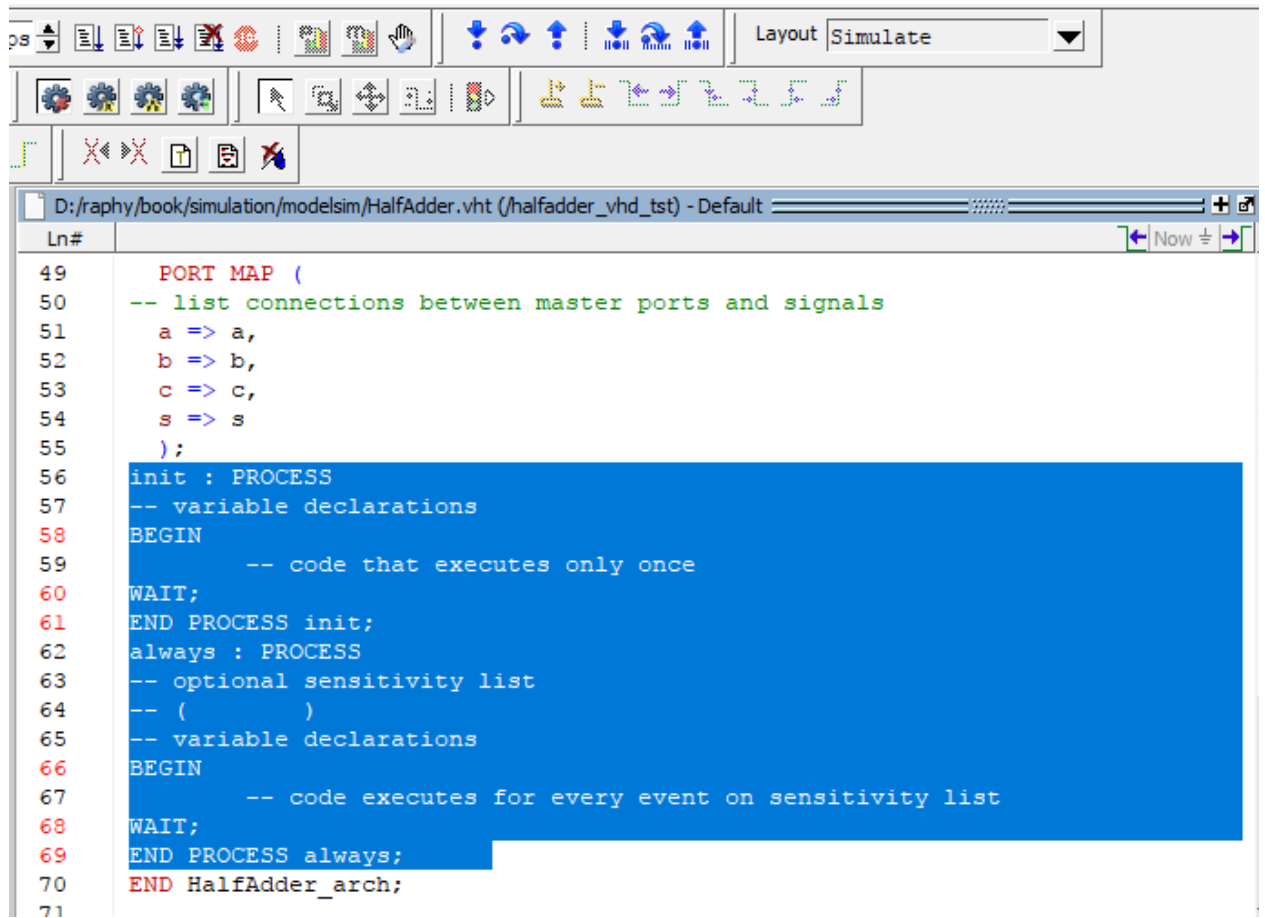


23. לאחר הקומפילציה סימני השאלה ישתנו לסימן של V.

24. הקליקו פעמיים על הקובץ השני.



25. מסיבות שנסביר מאוחר יותר, נא מחקו את השורות הבאות: מ-init:process ועד end .process always



```

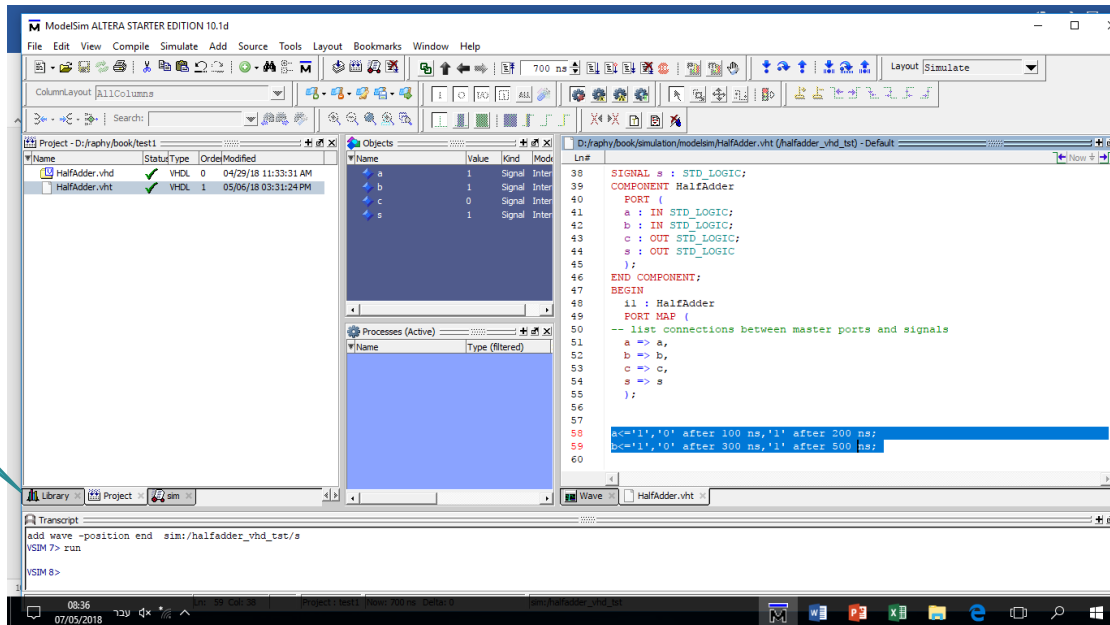
49  PORT MAP (
50  -- list connections between master ports and signals
51  a => a,
52  b => b,
53  c => c,
54  s => s
55  );
56  init : PROCESS
57  -- variable declarations
58  BEGIN
59      -- code that executes only once
60  WAIT;
61  END PROCESS init;
62  always : PROCESS
63  -- optional sensitivity list
64  -- ( )
65  -- variable declarations
66  BEGIN
67      -- code executes for every event on sensitivity list
68  WAIT;
69  END PROCESS always;
70  END HalfAdder_arch;
71
  
```

26. הוסיפו את שתי השורות הבאות, נא הקפידו להקליד אותן במדויק.

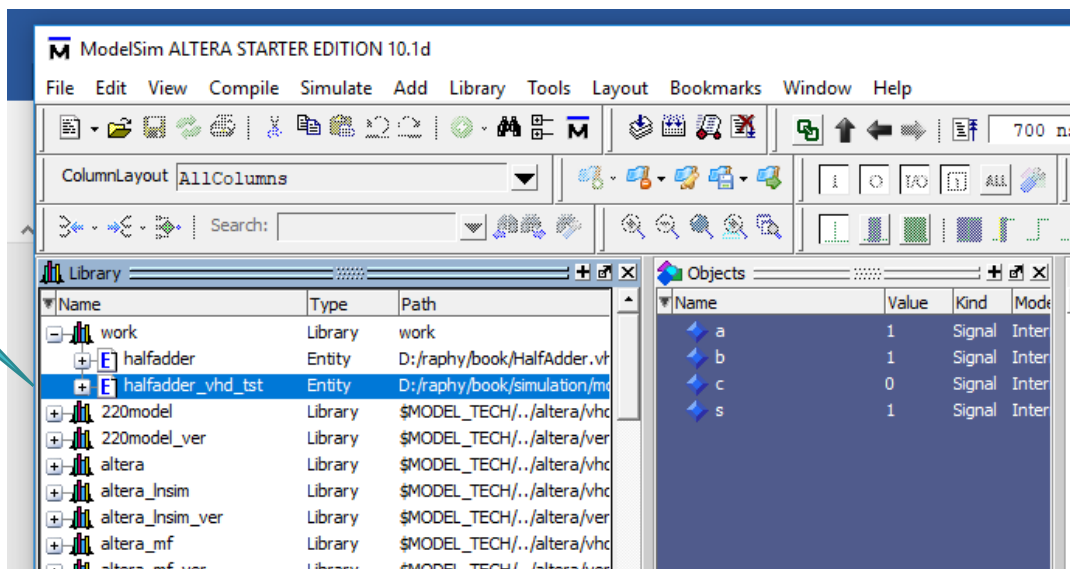
;a<='1','0' after 100 ns,'1' after 200 ns

;b<='1','0' after 300 ns,'1' after 500 ns

27. הקליקו על צלמית ה-library.

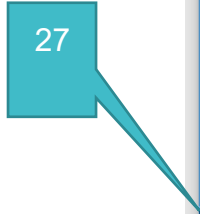


28. הקליקו פעמיים על קובץ ה-test bench בספריית work.

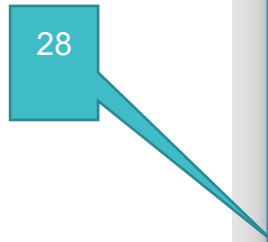


29. הקליקו על library.

30. כעת תופיע ספריית work כפי שהוגדרה באחד מהשלבים הקודמים.

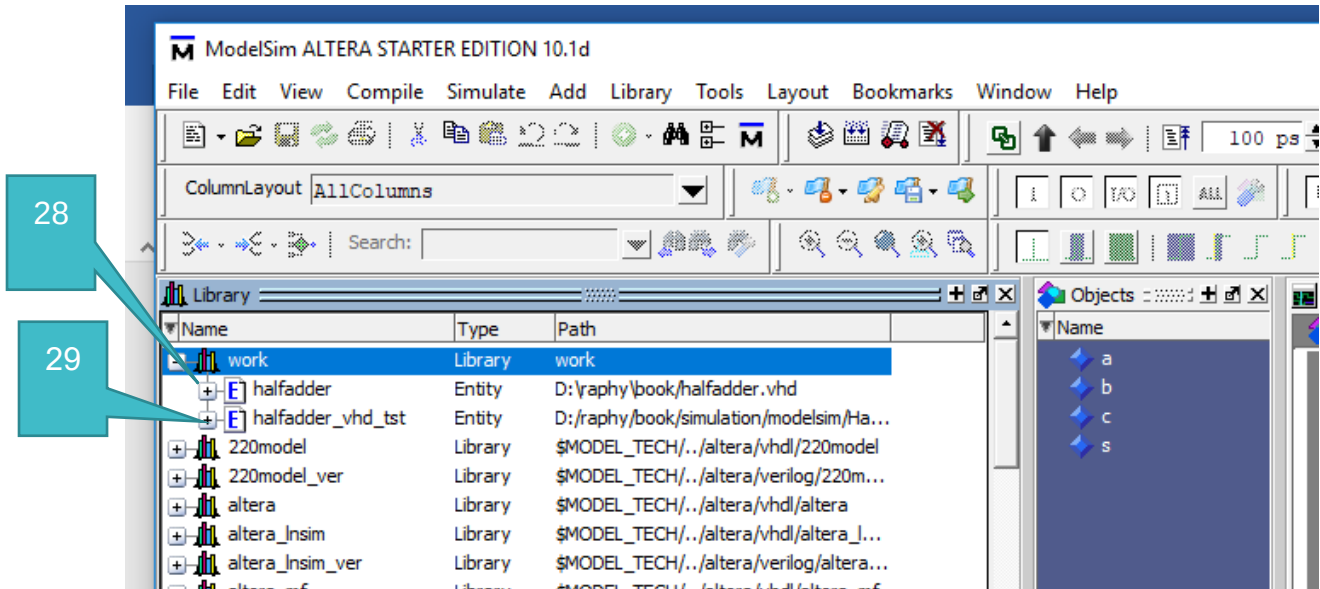


31. הקליקו על סימן הפלוס (+) שבספריית work.

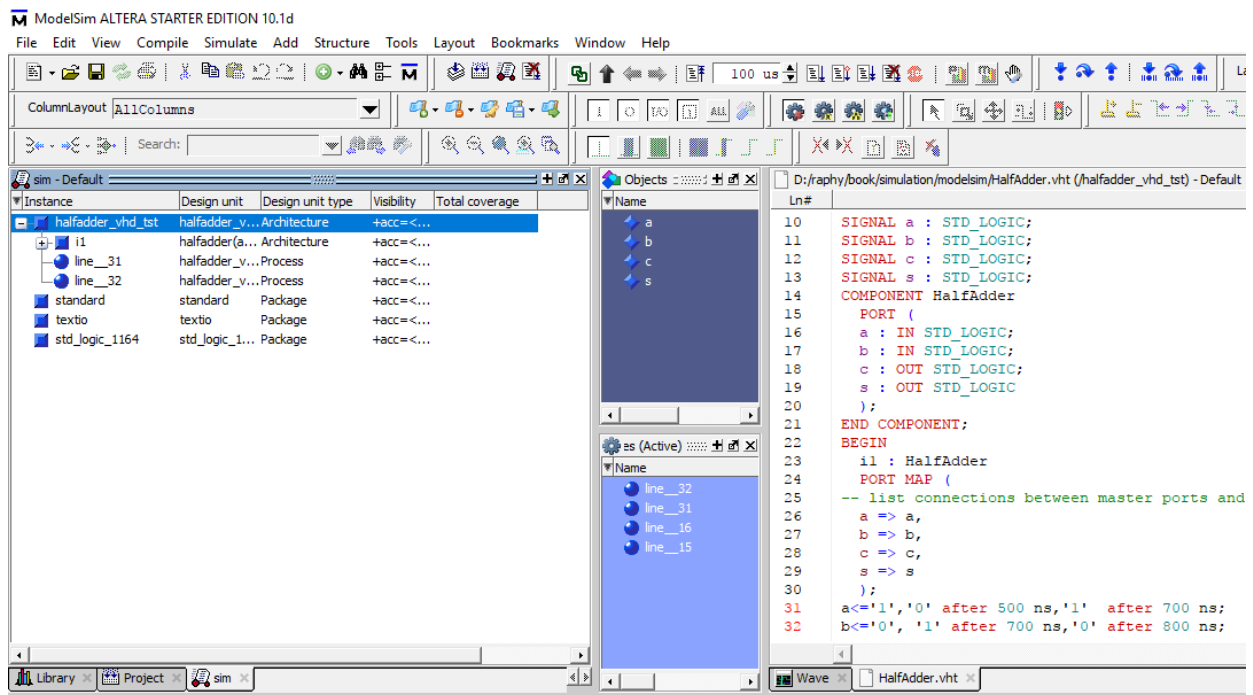


32. יופיעו שני קבצים – האחד הוא: HalfAdder.vhd.

33. הקובץ השני הוא: HalfAdder_vhd_tst.



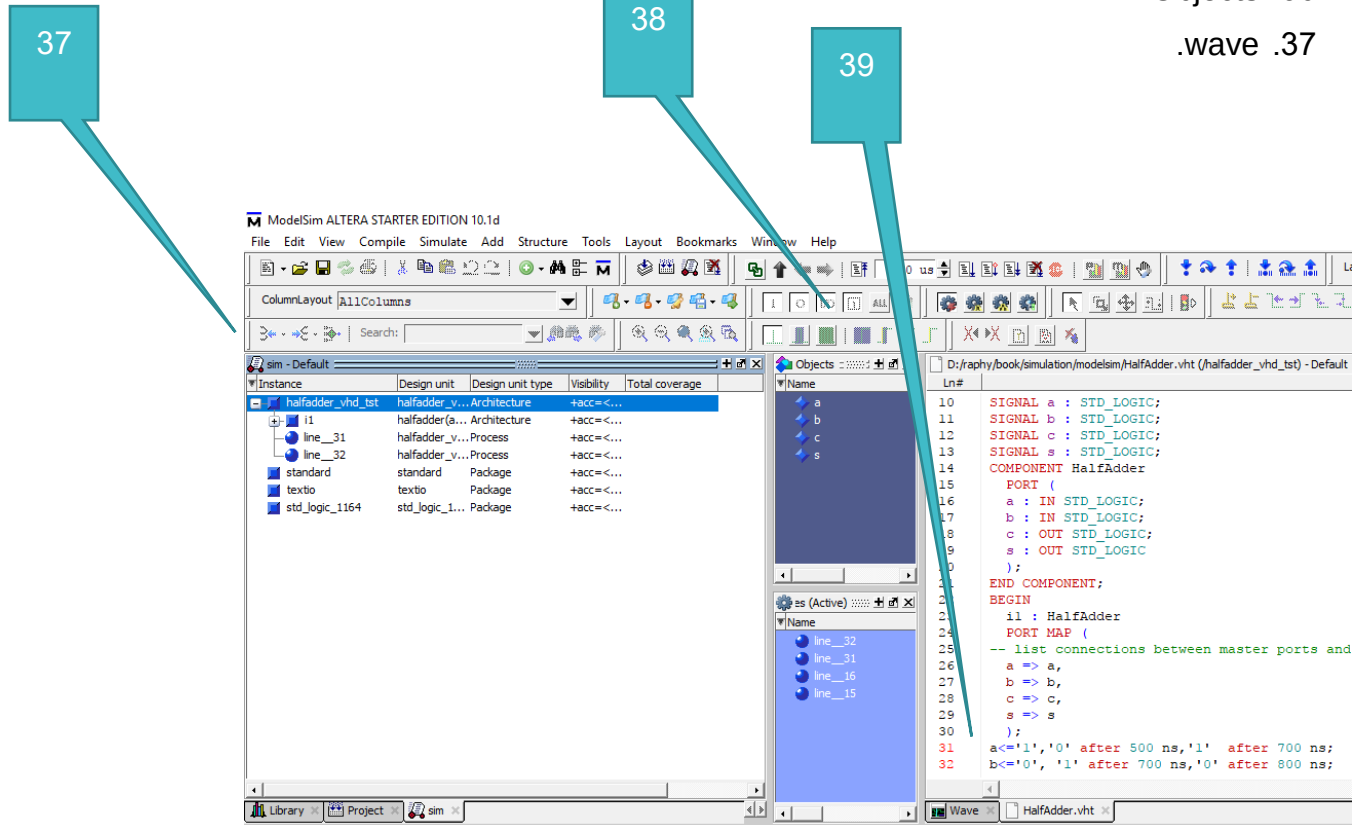
34. יפתחו החלונות האלה, נפרט אותם ונציג כל אחד מהם בנפרד.



.Sim .35

.Objects .36

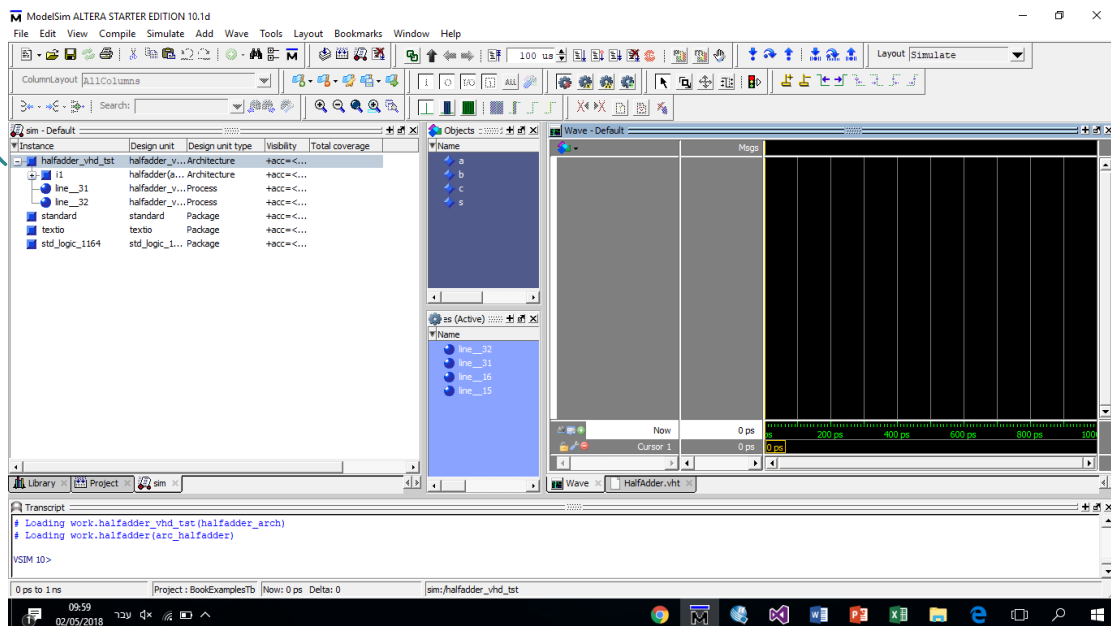
.wave .37



38. הקליקו על הצלמית של wave.

39. הקליקו עם הלחצן הימני של העכבר בשרת HalfAdder_vhd_tst.

41

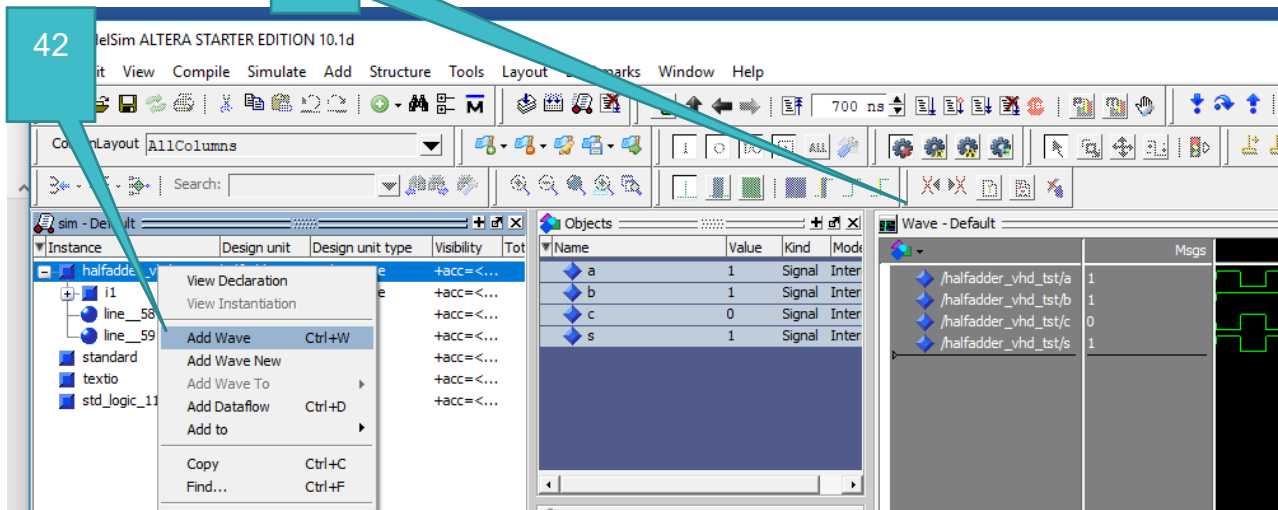


40. לחצו על add wave.

41. בחלון object תופיע לפניהם רשימה של הכניסות ושל היציאות.

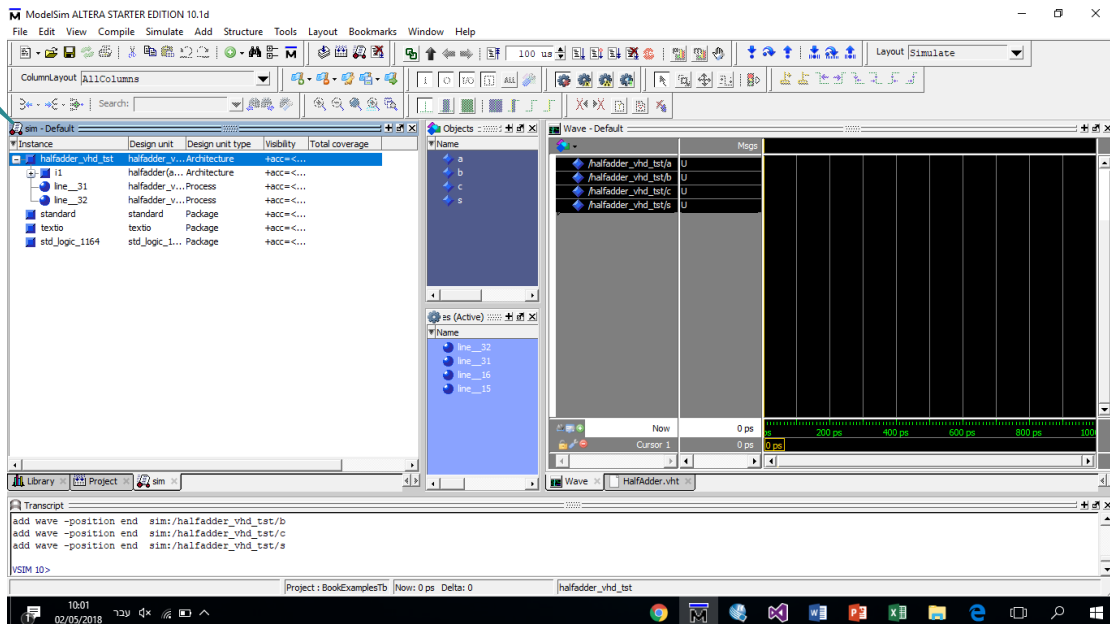
43

42



42. אפשר גם לגרור את כל הרשימה לחלון ה-wave באמצעות הלחצן השמאלי של העכבר.

44



43. נכיר את פונקציות ההרצה.

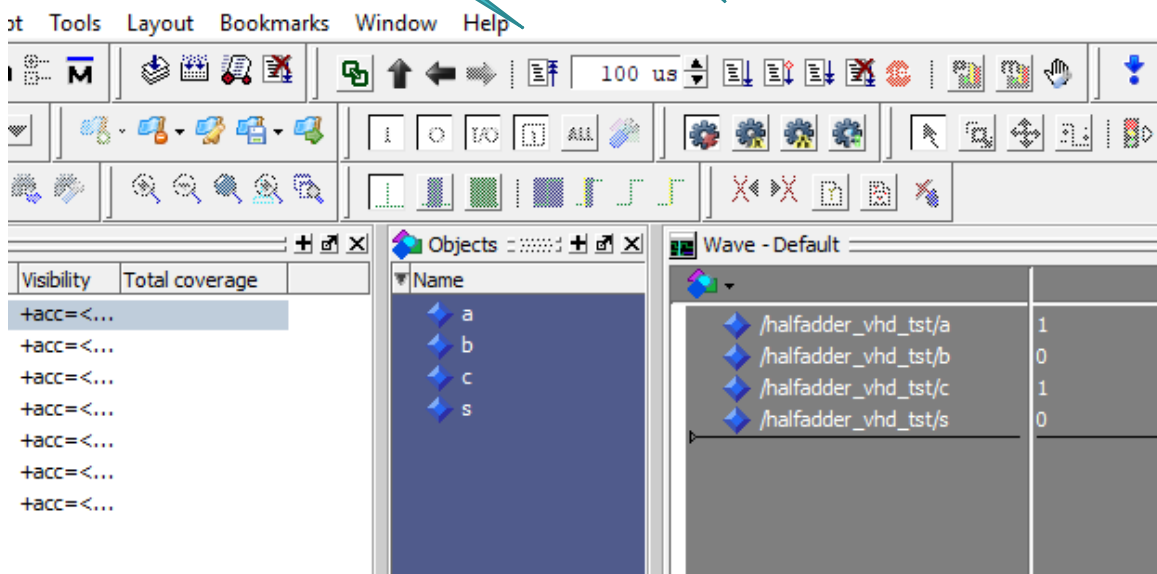
44. Run.

45. Restart.

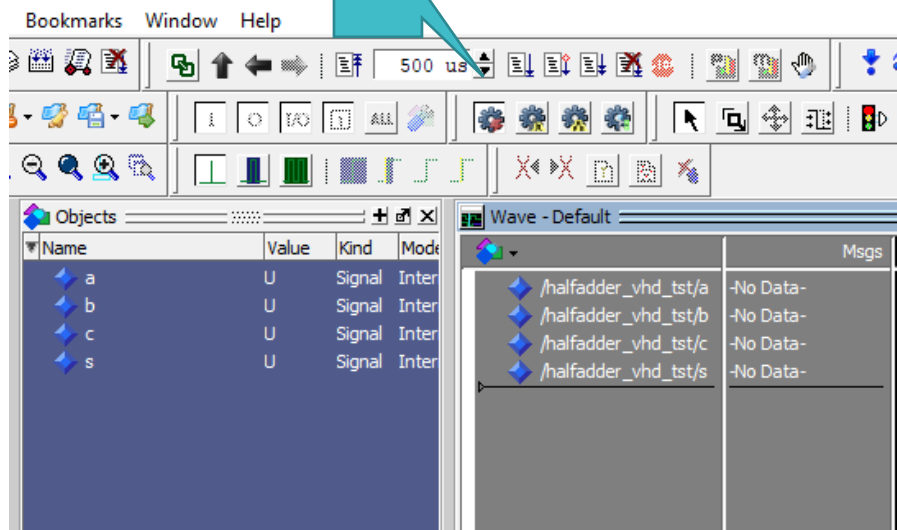
48

46

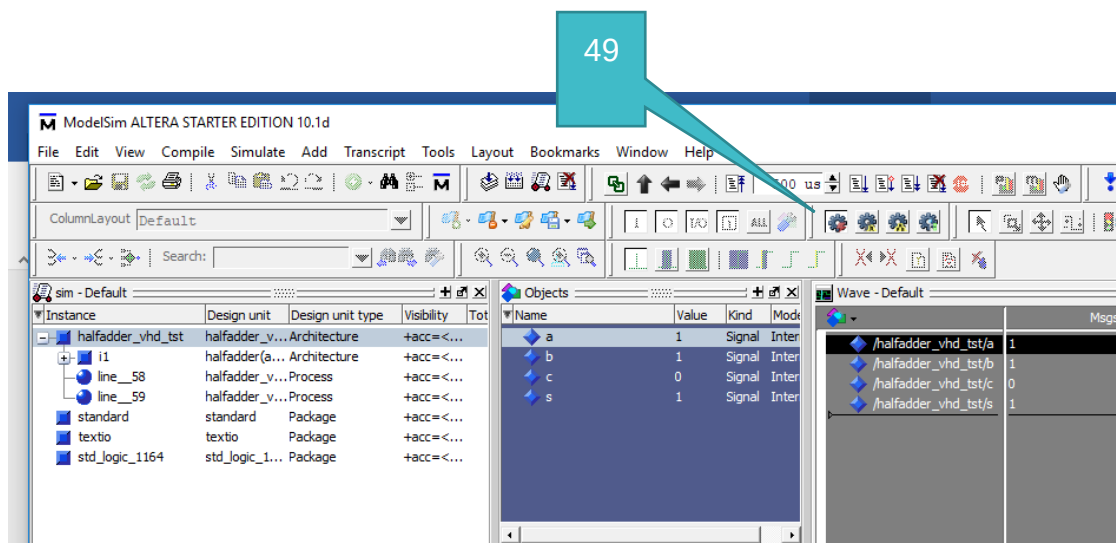
47



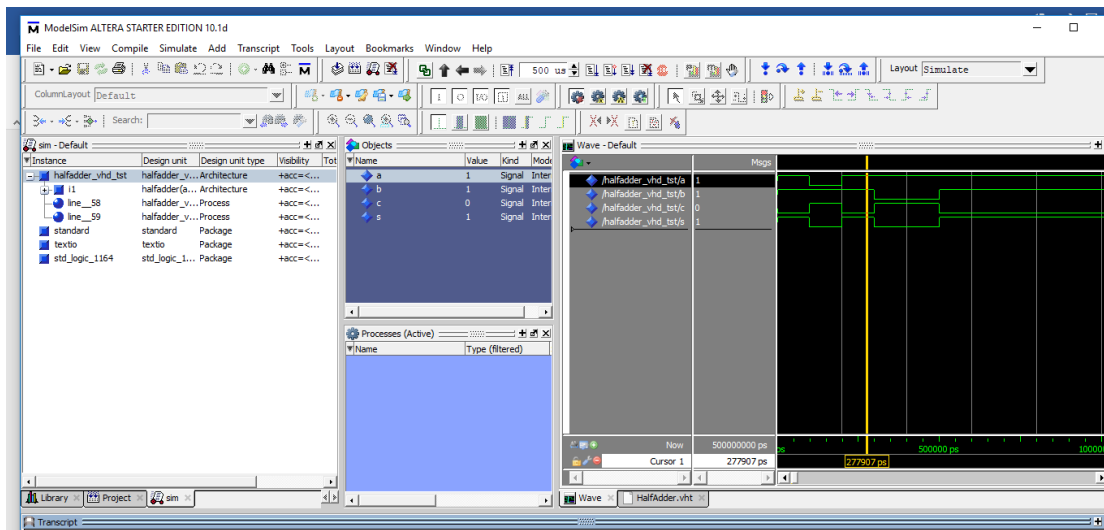
46. בחלון הזה אפשר לקבוע את זמן ההרצה. שנו את 100 ל-500 ואת ns (n = ננו) ל-us.
(u = מיקרו).



47. הקליקו על run.
בחלון שתקבל תוצאה של גלים.

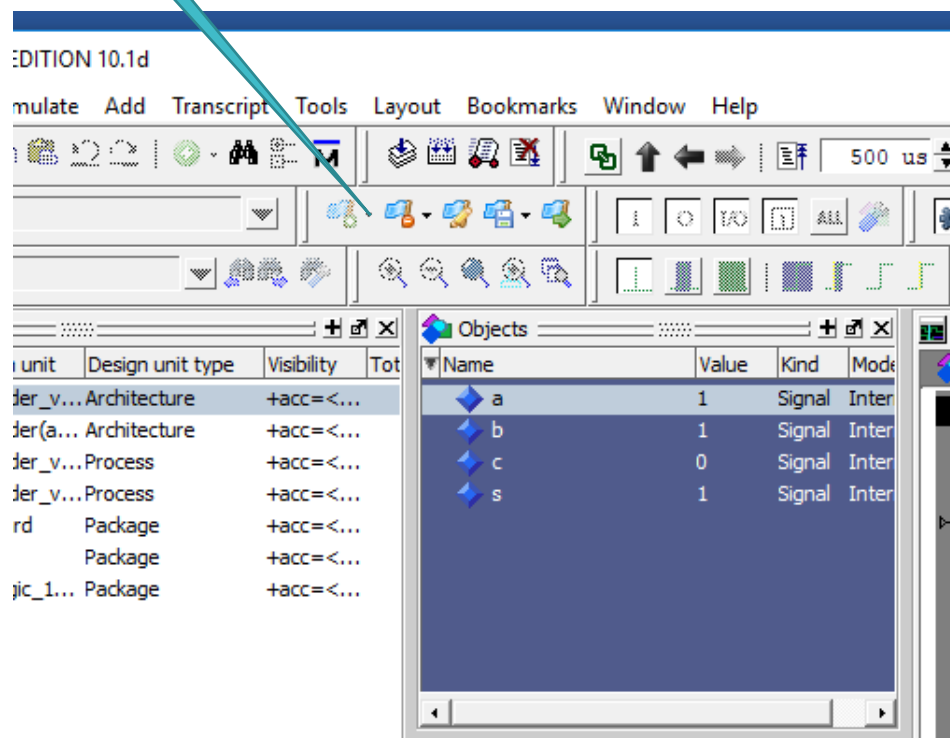


48. תוצאה אפשרית:



51

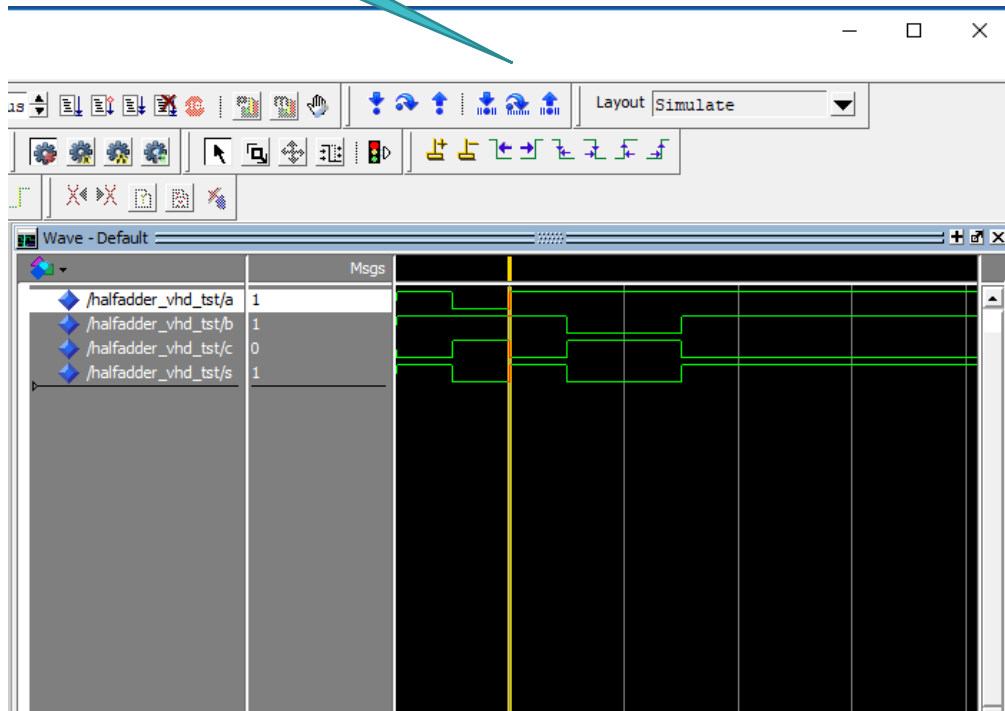
49. אפשר לעשות zoom, הקישו על המסך השחור ב-wave.



50. הקישו על אחד מהמשתנים בחלון wave.

כאן הזרקור ממוקד על התוצאה של משתנה a. אם נקיש על החצים שמסמלים את העליות ואת הירידות נוכל להגיע לשינויים הבאים או לשינויים הקודמים.

52



פרק 3: אבני בנייה לקומפילציה

אחרי שהכרנו בפרקים הקודמים את סביבות העבודה של quartus ושל modelsim, חשוב להדגיש שתרגול בסביבת העבודה הוא בסיס חיוני להתקדמות של הלומד בנבכי השפה.

בפרק הזה נלמד את המבנה הבסיסי של שפת ה-VHDL, שבה משתמשים ביישומים של מעגלים לוגיים בסיסיים.

שפת ה-vhdl מחולקת לשני חלקים עיקריים: entity ו-architecture.

Entity: בחלק הזה נגדיר את הכניסות ואת היציאות של המעגל המתוכנן.

Architecture: בחלק הזה נתאר את הלוגיקה שאנחנו רוצים ליישם.

בפרק הזה נכיר דרך נוספת כדי ליצור פרויקט, נוסף על הדרך הראשונה שהכרנו בפרק על סביבת העבודה של ה-quartus. דרך העבודה הזאת לא הובאה קודם כי מניסיונו של הכותב הדברים מובנים טוב יותר דרך המקלדת. האמצעים האוטומטיים הם טובים יותר כשהניסיון של המפעיל אותם הוא רב יותר.

3.1 יצירת קובץ חדש בעזרת אמצעים גרפיים

לפני כן נזכיר שהשלב הראשון הוא entity – הגדרת הכניסות והיציאות של המעגל או

של הפרויקט שאנחנו מתכננים.

נחזור לדוגמה של half adder.

$$S = a \text{ XOR } b.$$

$$C = a \text{ AND } b.$$

טבלת האמת שלו היא:

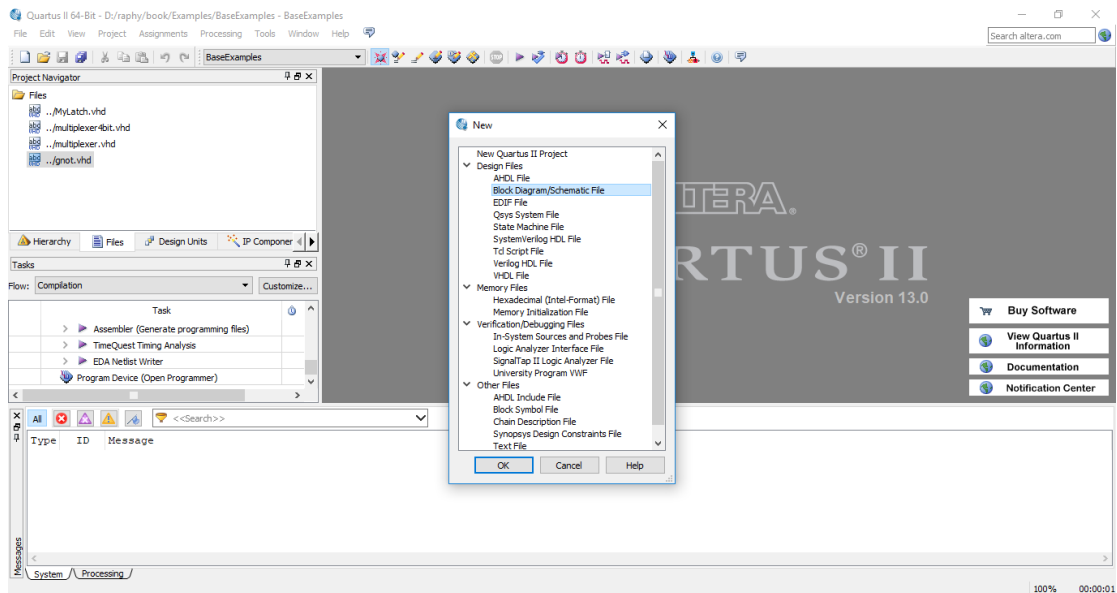
a	b	s	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

כניסות המערכת הן: a, b.

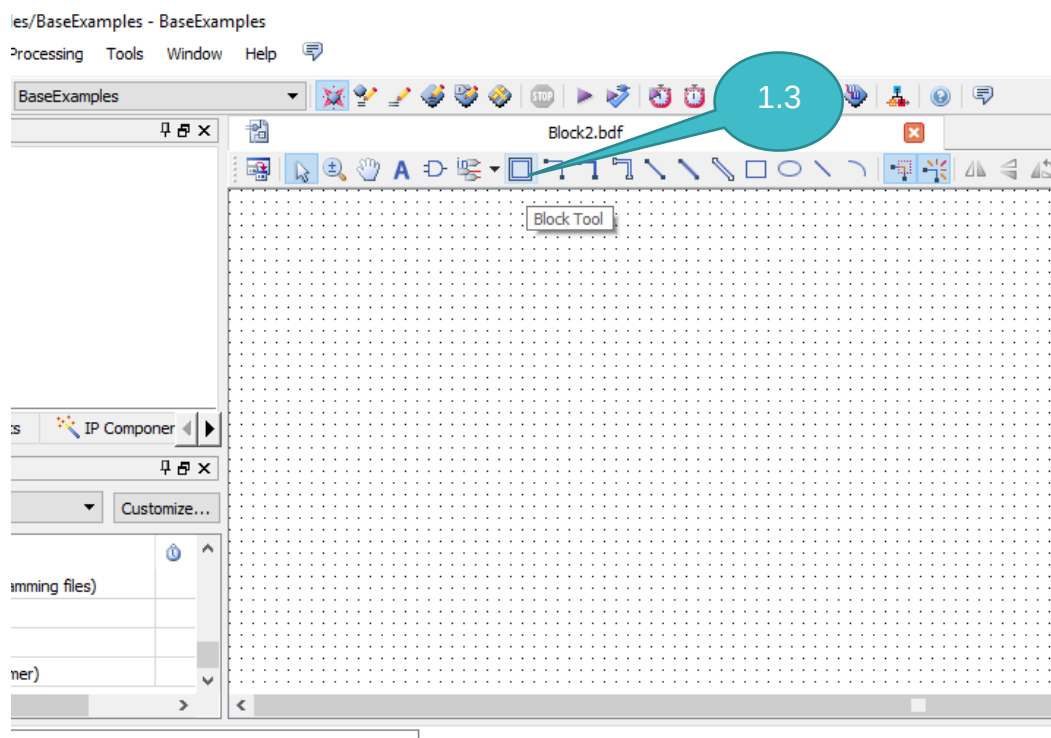
יציאות המערכת הן: c, s.

בחרו מהתפריט את דרך עבודתכם לפי הצעדים הבאים:

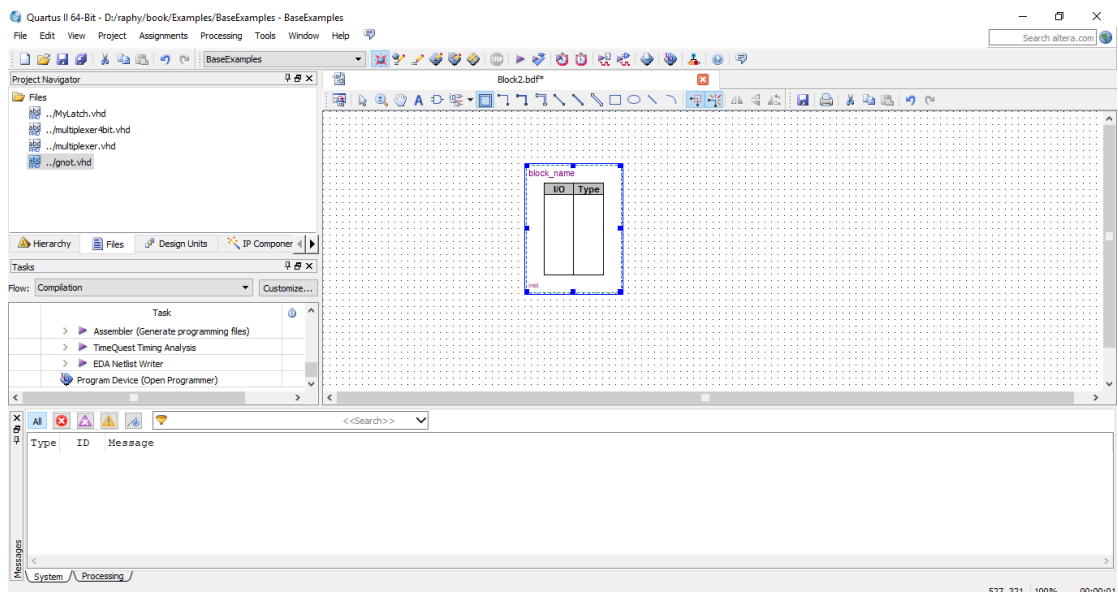
1. file => new
2. block diagram /schematic file
3. הקישו o.k



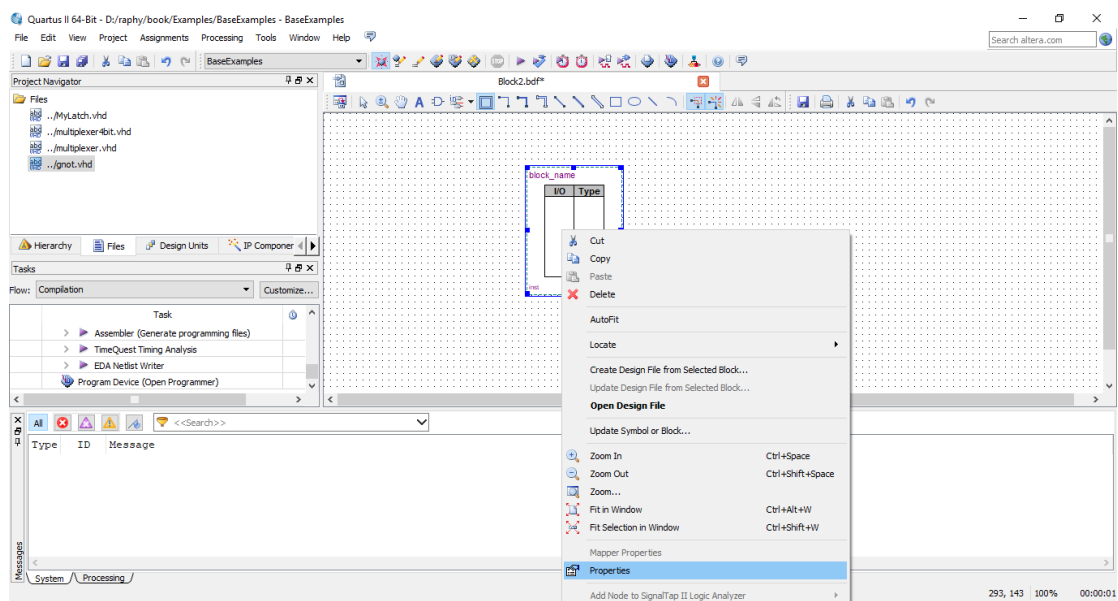
4. ממו את block tool באמצעות לחיצה על העכבר.



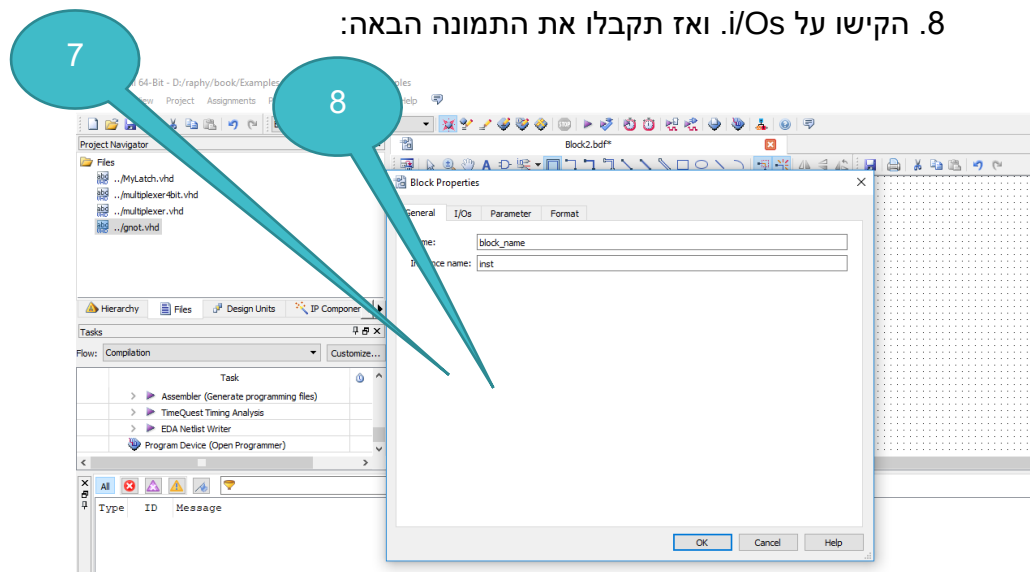
5. כשהחץ של העכבר ייהפך לצלב, ציירו בעזרתו מלבן.



6. העמידו את המלבן על הריבוע והקישו על הלחצן הימני על העכבר. בחרו את האפשרות Properties.

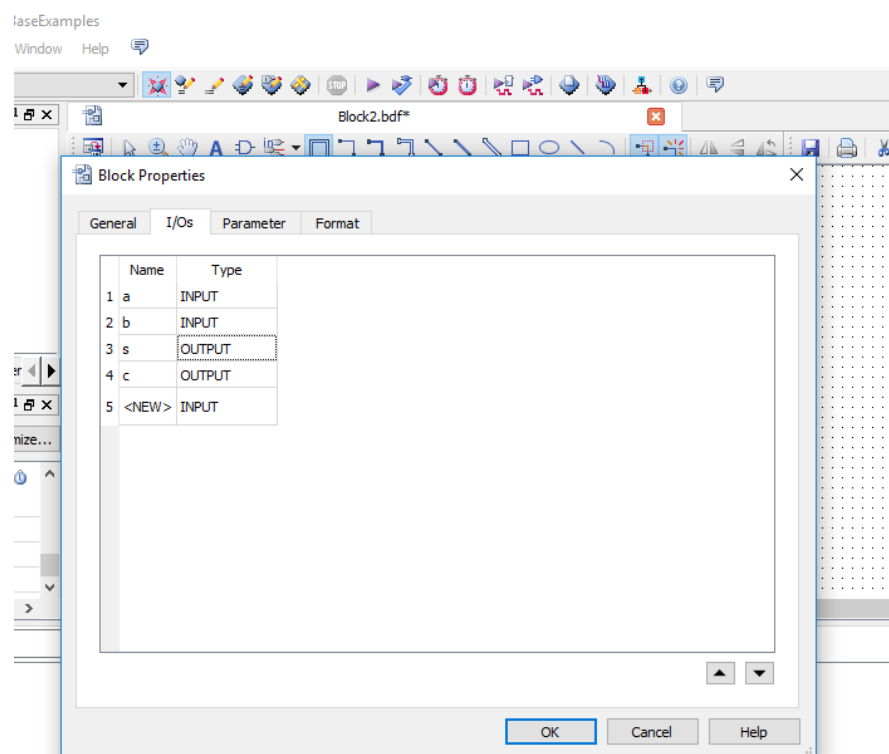


7. שנו את השם block name ל-HalfAdder.

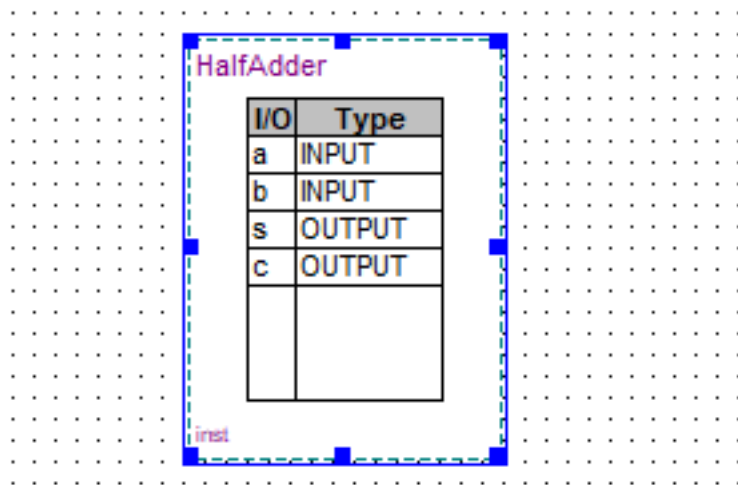


9. הקישו פעמיים על new, כתבו a במקומו והקישו enter. חזרו שוב לעשות ככה גם בנוגע לפרמטרים האחרים (b,c,s), עד שתקבלו את התמונה הבאה. שימו לב ל-output ביציאות s,c. תוכלו לעשות זאת אם תשנו את ה-input ל-output.

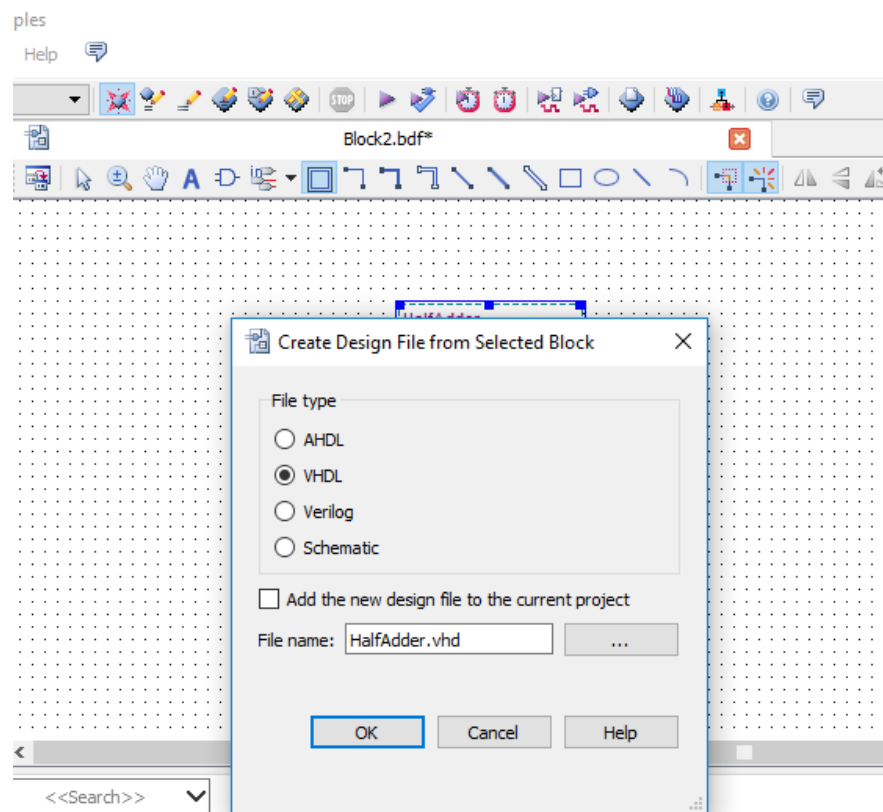
10. הקישו o.k.



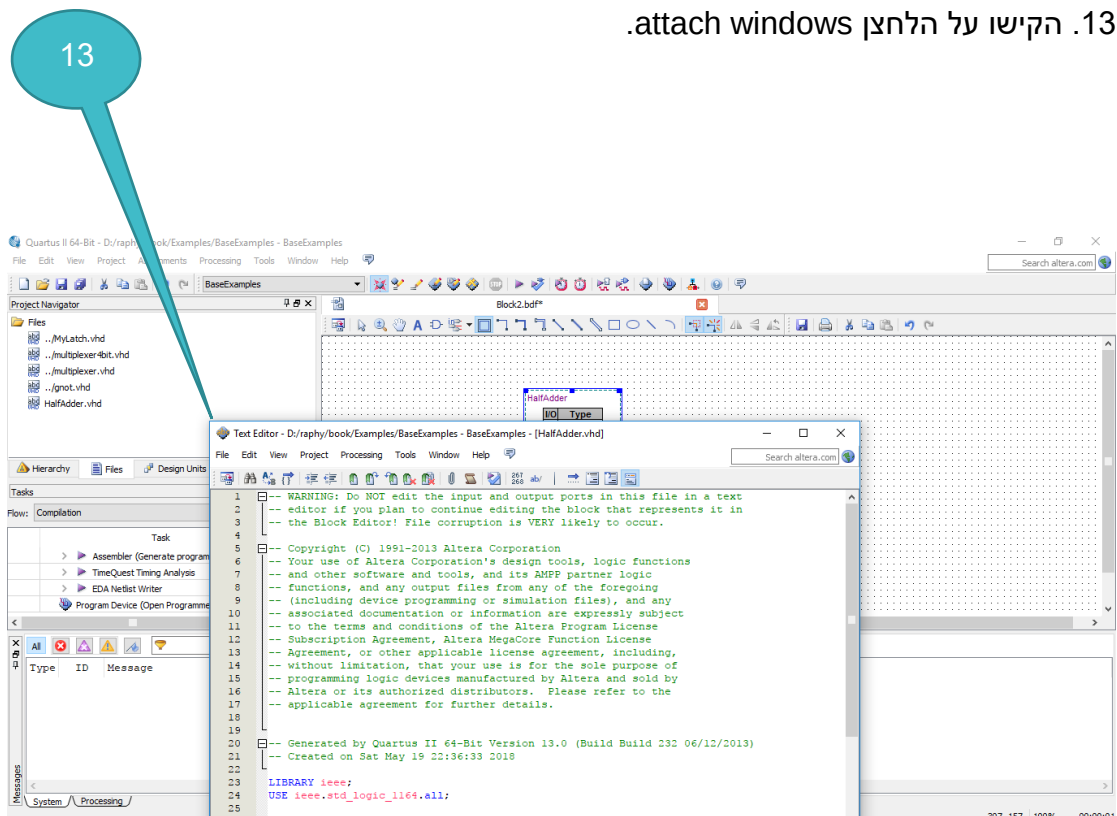
11. אם פעלתם נכון תתקבל התמונה הבאה:



12. עמדו על המלבן. כשהחץ יתחלף לצלב הקישו על הלחצן הימני של העכבר ואז על
 Creat design file from selected block. בחרו את האפשרות vhd, לחצו o.k ושוב o.k.



13. הקישו על הלחצן attach windows.



לנחיותכם, מחקו את כל ההערות הידועות.

1. מבנים בסיסיים של vhd.

```

1
2
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.all;
5
6  ENTITY HalfAdder IS
7  PORT
8  (
9    a : IN STD_LOGIC;
10   b : IN STD_LOGIC;
11   s : OUT STD_LOGIC;
12   c : OUT STD_LOGIC;
13 );
14
15 END HalfAdder;
16
17

```

כל תכנון מורכב משלושה רכיבים: ספריות, architecture, entity.

שורות 2,3 הן ספריות, ונקדיש להן פרק בהמשך. כמו כל ספרייה התפקיד שלהן הוא לתרגם את פעולות השפה.

```
Entity <שם התכנון> is
Port (
כניסות ויציאות
);
End;
```

באופן מפורט יותר:

ENTITY (שורה 6) – הגדרת התכנון, שם התכנון: HalfAdder.

PORT (שורה 8) – הגדרת הכניסות והיציאות.

IN STD_LOGIC (שורות 10 עד 13) – הגדרת הכניסות לתכנון והיציאות ממנו.

בשורה 9 פותחים את הסוגר.

בשורה 14 סוגרים את הסוגר בצירוף סימן של נקודה־פסיק.

בשורה 16 מסתיים קטע ה-ENTITY.

עכשיו נפנה לחלק שבו מוגדרים ביצועי התכנון – architecture. החלק הזה מורכב מהקטעים הבאים:

Architecture

Begin

End

```

6  ENTITY HalfAdder IS
7
8  PORT
9  (
10 a : IN STD_LOGIC;
11 b : IN STD_LOGIC;
12 s : OUT STD_LOGIC;
13 c : OUT STD_LOGIC
14 );
15
16 END HalfAdder;
17
18 ARCHITECTURE HalfAdder_architecture OF HalfAdder IS
19
20 BEGIN
21
22 s<= a xor b ;
23 c<= a and b;
24
25 END HalfAdder_architecture;
26

```

הערה: הוראת הסיום end יכולה להופיע לבד או להתלוות לשם התכנון, כמו שאפשר לראות בשורה 16 ובשורה 25.

את הלוגיקה של התכנון נכתוב בין הפקודה begin לפקודה end.

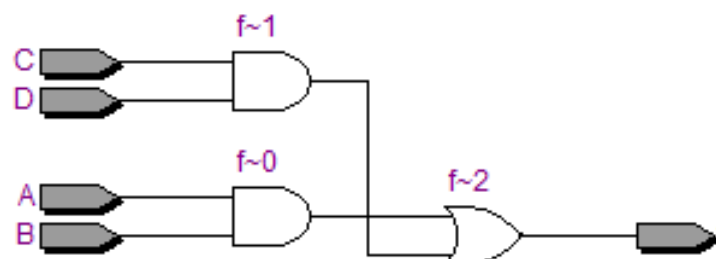
דוגמה נוספת:

```

2
3  LIBRARY ieee;
4  USE ieee.std_logic_1164.all;
5
6  ENTITY Gnot IS
7  |  PORT
8  |  (
9  |    A,B,C,D : IN STD_LOGIC;
10 |    f : OUT STD_LOGIC
11 |  );
12 |
13 |  END ;
14 |
15 |  ARCHITECTURE arc_Gnot OF Gnot IS
16 |  |
17 |  |  BEGIN
18 |  |    f<= (A and B) OR (C and D) ;
19 |  |  END ;
20

```

אפשר להמחיש את התכנון גם בעזרת RTL tools.



תרגיל תכנון בסיסיים:

חשוב מאוד שתתרגלו בסביבת העבודה ותשלטו בה בלי הנטל של הלוגיקה. אמנם השלב הזה מייגע, אך הוא די חיוני.

1. תכנונו מעגלי יסוד כגון: or, and, xor, nand.

3.2 אבני בנייה למערכות צירופיות

כדי להבין נושא חדש נדרש תמיד להבין את היסודות שהוא מורכב מהם. אני מזמין את הקורא למלאכת מחשבת, ובה נזהה את הדרישות הבסיסיות שמהן תורכב השפה:

1. תרגום שפה של טקסט לרכיבים לוגיים.
 2. טיפול בסוגים של מספרים.
 3. תיאורים סינכרוניים ואסינכרוניים.
 4. מערכות מורכבות כגון מכונת מצבים.
- בפרקים הבאים ננסה להמחיש כל אחד מהנושאים האלה.

אופרטור – operator: שם כולל לפעולה בתחום הדעת.

מערכות לוגיות הן ביטוי פיזיקלי לנושא "קבוצות" במתמטיקה.

אופרטור החיתוך – $\cap$ במתמטיקה.

דוגמה: $x = \{1,2,3,4,5,6\}$; $y = \{1,2,3,6\}$; $z = x \cap y$; $z = \{1,2,3,6\}$.

z הוא תוצאה של פעולת חיתוך בין שתי קבוצות.

בבסיס בינארי פעולת החיתוך היא AND. אפשר לתאר את פעולת OR או את פעולת NOT וכיוצא בזה.

ההגדרה של מערכת צירופית: מערכת לוגית שהתוצאות ביציאות שלה תלויות אך ורק במצב שבכניסות שלה.

סימן ההיכר של מערכת לוגית צירופית הוא מערכת ללא משוב.

$C \leftarrow a \text{ and } b;$

היא מערכת צירופית.

$C \leftarrow a \text{ and } C;$

אינה מערכת צירופית, מכיוון שהיא מכילה את זיכרון המצב הקודם של C , ועליו מבוצעת הפעולה and עם הפרמטר a .

לאחר שהכרנו את המושגים "אופרטור" ו"מערכת צירופית", נכיר כמה אופרטורים בסיסיים במערכות צירופיות: and , or , nand , nor , xor , xnor .

$C \leftarrow a \text{ and } b;$

$C \leftarrow a \text{ or } b;$

$C \leftarrow \text{not } a;$

$C \leftarrow a \text{ xor } b;$

$C \leftarrow a \text{ xnor } b;$

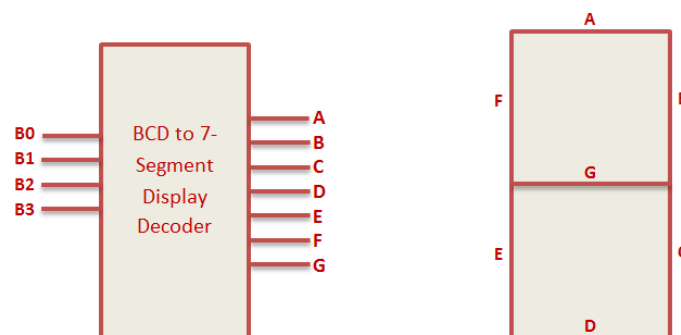
$C \leftarrow a \text{ nand } b;$

$C \leftarrow a \text{ nor } b;$

תרגול לדוגמה 

bcd to seven segment

אפיון פרויקט:



B3 B2 B1 B0	A B C D E F G
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0000100

The boolean expression for the logic circuit is

$$A = B_0 + B_2 + B_1B_3 + B_1'B_3'$$

$$B = B_1' + B_2'B_3' + B_2B_3$$

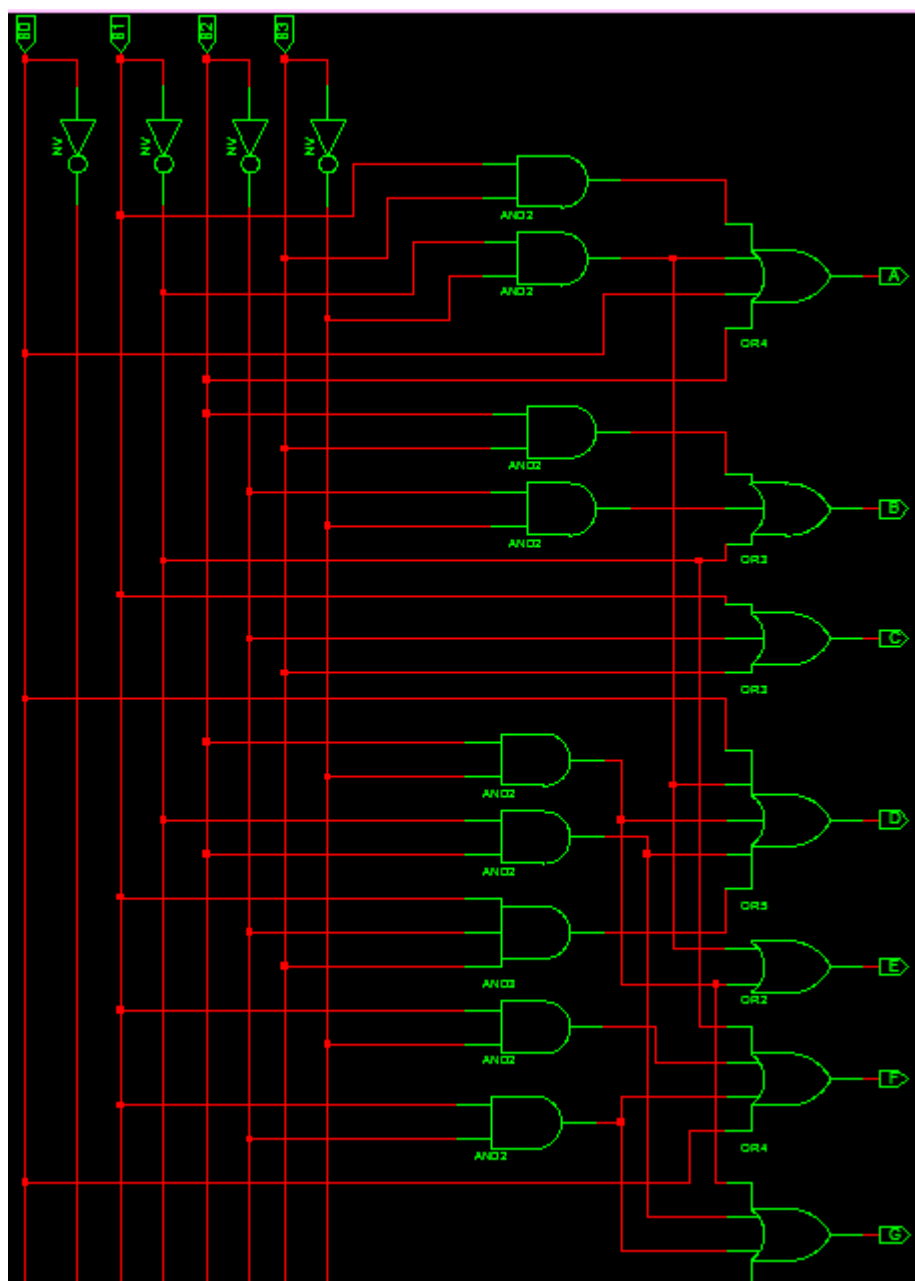
$$C = B_1 + B_2' + B_3$$

$$D = B_1'B_3' + B_2B_3' + B_1B_2'B_3 + B_1'B_2 + B_0$$

$$E = B_1'B_3' + B_2B_3'$$

$$F = B_0 + B_2'B_3' + B_1B_2' + B_1B_3'$$

$$G = B_0 + B_1B_2' + B_1'B_2 + B_2B_3'$$



```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity bcd_7seg is  
Port (B0,B1,B2,B3 : in STD_LOGIC;  
A,B,C,D,E,F,G : out STD_LOGIC);  
end bcd_7seg;
```

לא מקובל בכתיבה כזאת

```
architecture Behavioral of bcd_7seg is
```

```
begin
```

```
A <= B0 OR B2 OR (B1 AND B3) OR (NOT B1 AND NOT B3);  
B <= (NOT B1) OR (NOT B2 AND NOT B3) OR (B2 AND B3);  
C <= B1 OR NOT B2 OR B3;  
D <= (NOT B1 AND NOT B3) OR (B2 AND NOT B3) OR (B1 AND NOT B2  
AND B3) OR (NOT B1 AND B2) OR B0;  
E <= (NOT B1 AND NOT B3) OR (B2 AND NOT B3);  
F <= B0 OR (NOT B2 AND NOT B3) OR (B1 AND NOT B2) OR (B1 AND NOT  
B3);  
G <= B0 OR (B1 AND NOT B2) OR (NOT B1 AND B2) OR (B2 AND NOT B3);
```

```
end Behavioral;
```

השמה מותנית

כל הפעולות לא רגישות לאותיות קטנות או לאותיות גדולות.

No sensitive case

הקורא מוזמן לחזור על הפרק שעוסק בסינתזה ובסימולציה אם יצטרך.

מובן שאפשר לבנות מערכות לוגיות על-פי התכנים שנלמדו במערכות ספרתיות.

עכשיו נראה מבנים מורכבים יותר.

ניזכר בערך של xor.

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

אופרטור when

מבנה האופרטור:

Output <= <output value> when <condition> else

```
c <= '0' when ab = "00" else
```

```
'1' when ab = "01" else
```

```
'1' when ab = "10" else
```

```
'0' when ab = "11";
```

c (יציאה) יקבל את הערכים שמשמאל ל-when לפי התנאים שמימין ל-when.

אפשר לכתוב את זה יותר בקיצור.

נתאר את התנהגות הערך של `xor` בתור משוואה. במקרה של שוויון בין הכניסות `a=b` יתקבל '0' ביציאה, אי־שוויון יניב תוצאה של '1'.

```
c <= '0' when a=b else '1';
```

אופרטור `with select`

מבנה האופרטור:

```
with <output value> select
```

```
<output> <= <condition>;
```

```
with c select
```

```
c <= '0' when "00",
```

```
    '1' when "01",
```

```
    '1' when "10",
```

```
    '0' when "11";
```

אפשר גם לממש את הערך של `xor` בעזרת מבנה שנקרא **process** (תהליך). נלמד אותו בפרק `process`.

עיקרון מנחה בתכנון חומרה:

- אל תשאיר לעולם אפשרויות בלתי־מוגדרות.
- אין הדגש על היתרונות של כל אחת מההתניות בכתיבה ובסינתזה.

דוגמה: מימוש מפענח (decoder) בעזרת האופרטורים when ו-select.

a	b	Q0	Q1	Q2	Q3
X	X	0	0	0	0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

הקטע הקוד עליפי הטבלה:

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY whenDecoder IS

PORT(

ab : IN STD_LOGIC_VECTOR(1 downto 0); STD למה דווקא

q : out STD_LOGIC_VECTOR(3 downto 0)

);

END whenDecoder;

ARCHITECTURE whenDecoder_architecture OF whenDecoder IS

BEGIN

q<="0001" when ab="00" else

    "0010" when ab="01" else

    "0100" when ab="10" else

    "1000" when ab="11";

END whenDecoder_architecture;
    
```

הקוד הבא מתאר את ה-test bench של ה-decoder עם האופרטור when:

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY whenDecoder_vhd_tst IS

END whenDecoder_vhd_tst;

ARCHITECTURE whenDecoder_arch OF whenDecoder_vhd_tst IS

SIGNAL ab : STD_LOGIC_VECTOR(1 DOWNTO 0):="00";

SIGNAL q : STD_LOGIC_VECTOR(3 DOWNTO 0);

COMPONENT whenDecoder

    PORT (

        ab : IN STD_LOGIC_VECTOR(1 DOWNTO 0);

        q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)

    );

END COMPONENT;

BEGIN

    i1 : whenDecoder

    PORT MAP (

        ab => ab,

        q => q

    );

    ab<="00" after 200 ns , "01" after 400 ns , "10" after 600 ns ,

    "11" after 1000 ns;

END whenDecoder_arch;

```

Msgs							
/ab	-No Data-	00	01	10	11		
/q	-No Data-	0001	0010	0100	1000		

אפשר לראות את תגובת היציאה Q של המפענח (decoder) בהתאמה עם הכניסות ab.

מימוש המפענח הזה לפי האופרטור with:

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY withDecoder IS

PORT

(
  ab : IN STD_LOGIC_VECTOR(1 downto 0);
  q : OUT STD_LOGIC_VECTOR(3 downto 0)
);

END withDecoder;

with ab select
    q<=  "0001" when "00",
        "0010" when "01",
        "0100" when "10",
        "1000" when "11";

END withDecoder_architecture;

```

תיאור חלקי של התנהגות או של צירופים לא מוגדרים

עכשיו נדון באחת התופעות המסוכנות של תיאור חומרה.

המפענח תוכן בעצם כדי לכסות את כל האפשרויות באותות הכניסה של ab.

לפעמים לא נצליח לחשוב בדעתנו על כל האפשרויות, בגלל טעויות תכנון או מפני ריבוי האפשרויות והקושי להיות מודע אליהן.

כלל מנחה בתכנון חומרה: במקום שדעתך לא תגיע אליו קלי הסינתזה "יפעיל את דעתו", ודעתו לא תתיישב תמיד עם דעתך. במילים פשוטות, חיסכון במחשבה יוביל לתכנון לקוי.

אולם קלי הסינתזה לא מאפשרים לנו תמיד לטעות.

בדוגמה של האופרטור with select שלא מכסה את כל האפשרויות, נתקבל שגיאת קומפילציה:

Error (10313): VHDL Case Statement error at withDecoder.vhd(27): **Case Statement choices must cover all possible values of expression**

האופרטור others

כיסוי כל המקרים לא תמיד מעניין אותנו באמת. במקרה כזה נפעל עם האופרטור others.

האופרטור others יאפשר לנו את החופש לתת את הדעת על שאר המצבים.

דוגמה:

```
with ab select
```

```
    q<= "0001" when "00",
```

```
        "0010" when "01",
```

```
        "0100" when others;
```

	Msgs								
/ab	11	00	01	10	11				
/q	0100	0001	0010	0100					

השינוי מתקבל במצבים $ab=00$, $ab=01$ – אבל במצבים $ab=10$, $ab=11$ אין שינוי ביציאה q . בשני המצבים האלה הערך ביציאה הוא $q=0100$.

אופרטורים לוגיים ב-with וב-when

אפשר לצרף לפקודה אופרטורים לוגיים כדוגמת or , שמבוטא באמצעות הסימון $|$ או and .

```
with ab select
```

```
  q<="0001" when "00" | "01",
    "0010" when others;
```

	Msgs							
/ab	-No Data-	00	01	10	11			
/q	-No Data-	0001		0010				

לפי הסימולציה המצורפת הערך ביציאה הוא $q = 0001$ כאשר $ab = 00$ ו- $ab = 01$, ובשאר המצבים הערך הוא $q = 0010$.

נחזור לתרגיל הדוגמה "bcd to seven segment", אולם הפעם היישום יתבצע בעזרת when select-ו.

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity bcd_7segment is
Port (BCDin : in STD_LOGIC_VECTOR (3 downto 0);
Seven_Segment : out STD_LOGIC_VECTOR (6 downto 0));
end bcd_7segment;

architecture Behavioral of bcd_7segment is
begin
with BCDin select
    Seven_Segment <=
        "0000001"    when "0000", ---0
        "1001111"    when "0001", ---1
        "0010010"    when "0010", ---2
        "0000110"    when "0011", ---3
        "1001100"    when "0100", ---4
        "0100100"    when "0101", ---5
        "0100000"    when "0110", ---6
        "0001111"    when "0111", ---7
        "0000000"    when "1000", ---8
        "0000100"    when "1001", ---9
        "1111111"    when others;

end Behavioral;
```

אופרטור When

```

Seven_Segment <= "0000001"  when BCDin="0000" else---0
                                "1001111"  when BCDin="0001" else---1
                                "0010010"  when BCDin="0010" else--2
                                "0000110"  when BCDin="0011" else--3
                                "1001100"  when BCDin="0100" else--4
                                "0100100"  when BCDin="0101" else--5
                                "0100000"  when BCDin="0110" else--6
                                "0001111"  when BCDin="0111" else--7
                                "0000000"  when BCDin="1000" else--8
                                "0000100"  when BCDin="1001" else--9
                                "1111111"  when BCDin="1010";

```

תהליך סדרתי ומקבילי

על-פי הלוגיקה הקלאסית האופרטור process (=תהליך) מכיל רכיבים או רעיונות שמוכרים לנו דווקא מהתוכנה ומהחומרה. אפשר לומר שהאופרטור process הוא בעצם הרחבה של הלוגיקה הקלאסית.

בעיקרון, כל תהליך הוא מעגל לוגי בפני עצמו.

לכל תהליך יש כניסות (input) ויציאות (output).

3.3 תרגום של קוד לרכיבים באמצעות RTL

חשוב מאוד להבין תרגום של קוד שנכתב בשביל רכיבים לוגיים.

אפשר לעשות את זה ב-quartos בעזרת RTL.

אלה הלשוניות שצריך לבחור מהתפריט: RTL => netlist viewer => tools.

שפת ה-Vhdl היא תרגום של שפת קוד לרכיבים לוגיים. הסינטיסייזר (synthesizer) מסוגל לתרגם לרכיבים ידועים קוד שנכתב על-פי מבנים מוכרים מראש. כל השערים הלוגיים הם בעצם רכיבים פרימיטיביים של מערכות לוגיות. ודאי ידוע לכם שרכיבים כאלה מורכבים מטרנזיסטורים, מנגדים ומדיודות. במערכות לוגיות כמעט שלא צריך לנתח את המעגלים ברמה כזאת. הפונקציות הלוגיות (and, xor וכדומה) הן כל מה שמעניין אותנו. על כן יש לכל סביבה רכיבים יסודיים שמהם היא מורכבת, ואלה נקראים רכיבים פרימיטיביים.

בין רכיבי ה-FPGA יש גם רכיבים פרימיטיביים שמהם הסביבה מממשת את המערכות הלוגיות.

המטרה של הפרק הזה היא להכיר כמה שאפשר את הרכיבים הפרימיטיביים שבהם הסביבה quartos משתמשת.

את ההתנהגות של המעגלים נבחן כמובן בסימולציה ולא כאן. במקרים מסוימים שאין בהם קורלציה, מבט על המעגלים האלה יתרום בהחלט להבנת התהליך.

רכיבים פרימיטיביים שהסביבה נעזרת בהם: flip-flop, multiplexer ועוד.

Multiplexer – רבב

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY mymux IS

PORT

(

a,b,s : IN STD_LOGIC;

q : OUT STD_LOGIC

);

END mymux;

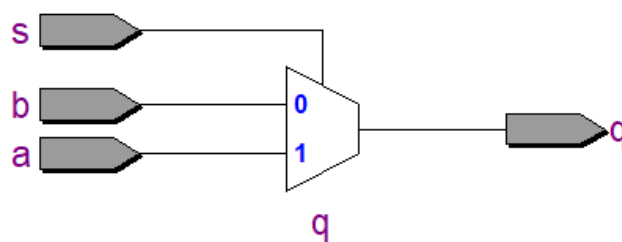
ARCHITECTURE mymux_architecture OF mymux IS

BEGIN

q<=a when s='1' else b;

END mymux_architecture;

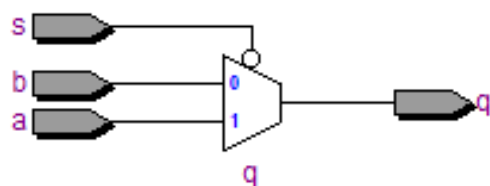
```



```
q<=a when s='1' else b;
```

בציור העליון היציאה q תקבל את הערך של הכניסה a כאשר $s=1$.

לעומת זאת בציור התחתון היציאה q תקבל את הערך של הכניסה a כאשר $s=0$.



$q \leftarrow a$ when $s = '0'$ else b ;

נממש שער של xor באמצעות רגב.

דוגמא נוספת:

הדוגמא הזאת אולי תיראה לכם קצת מוזרה, אבל היא חשובה מאוד.

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY myxor IS

PORT

(

q : out std_logic;

ab : in std_logic_vector(1 downto 0)

);

END myxor;

ARCHITECTURE whenDecoder_architecture OF myxor IS

BEGIN

q<= '0' when ab="00" else

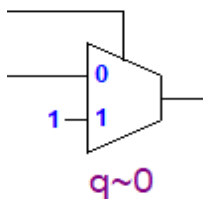
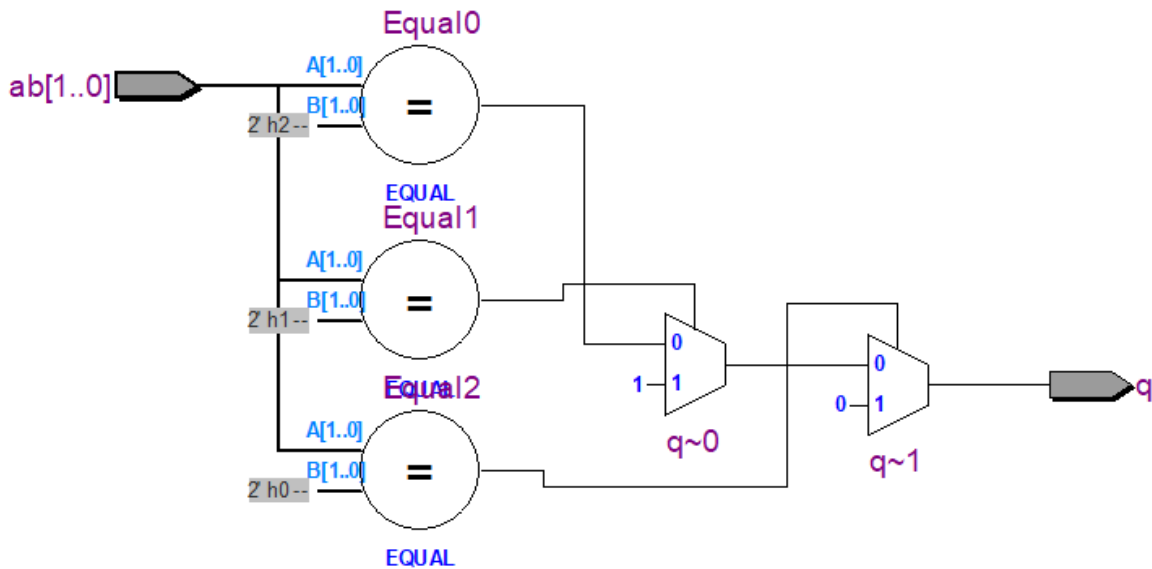
    '1' when ab="01" else

    '1' when ab="10" else

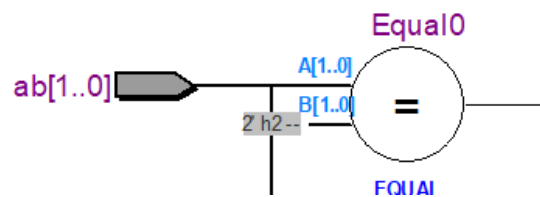
    '0' when ab="11" else '0';

END;
```

לכל סינטיסייזר יש דרך מימוש משלו, ולכן חשוב להכיר את הכלי שבו אנחנו עובדים.



רֶבֶב (multiplexer)



מִשְׁוֶן (comparator)

למשוון יש שתי כניסות: האחת היא כניסת ab והשנייה היא ערך של $2h0$. המשמעות של זה היא הערך 00 בבסיס הקסדצימלי ($16=$), ואילו המספר 2 מציין שני ביטים. ב- $2h1$ מסומל הערך 01 בבסיס 16 , והמספר 2 יורה שוב על שני ביטים.

הכניסה ab משותפת לכל המשוונים ומסונכרנת עם התכנים של כל אחד מהם.

הערך ביציאה של המשוון הוא '1' במקרה של שוויון בין הכניסות.

הרבב בורר בין שתי הכניסות.

הבה נוכיח שהמעגל הלוגי הנ"ל מתנהג כמו שער xor.

בשביל זה נבנה טבלת אמת.

a	b	Equal 0	Equal 1	Equal 2	$q \sim 0$	$q \sim 1$
0	0	0	0	1	0	0
0	1	0	1	0	1	1
1	0	0	0	1	1	1
1	1	0	0	0	0	0

אם נעקוב אחרי הטבלה לפי השרטוט נגיע לתוצאה הנ"ל.

נתמקד כרגע במצב האחרון: $ab=11$. במצב הזה לכל היציאות של המשוונים יש ערך של '0',

וגם הערך ביציאה של הרבב $q \sim 1$ הוא '0'.

חשוב לשים לב שהמערכת היא צירופית כמובן, בלי משובים או דלגלג (flip-flop circuit).

Gated latch

הקוד של שער ה-xor:

```
q<= '0' when ab="00" else
    '1' when ab="01" else
    '1' when ab="10" else
    '0' when ab="11" else '0';
```

הקריאה של האופרטור when צריכה להיות בדרך הבאה:

$q='0'$ אם הכניסה $ab="00"$,

אחרת '1' כאשר $ab="01"$,

אחרת '1' כאשר $ab="10"$,

אחרת '0' כאשר $ab="11"$,

אחרת '0'.

המצב האחרון נראה לכאורה מיותר, מכיוון שלשני ביטים יש רק ארבע אפשרויות.

אם כן, פקודת ה-else האחרונה היא למעשה מצב חמישי.

צריך להכיר טוב כל סביבת עבודה שאנחנו עובדים בה. quartus היא סביבת העבודה שלנו.

הדוגמא הבאה תבהיר מדוע ראוי וכדאי להכיר את סביבת העבודה.

כפי שכבר צוין יש חמישה מצבים, ועל כן המצב האחרון הוא מיותר.

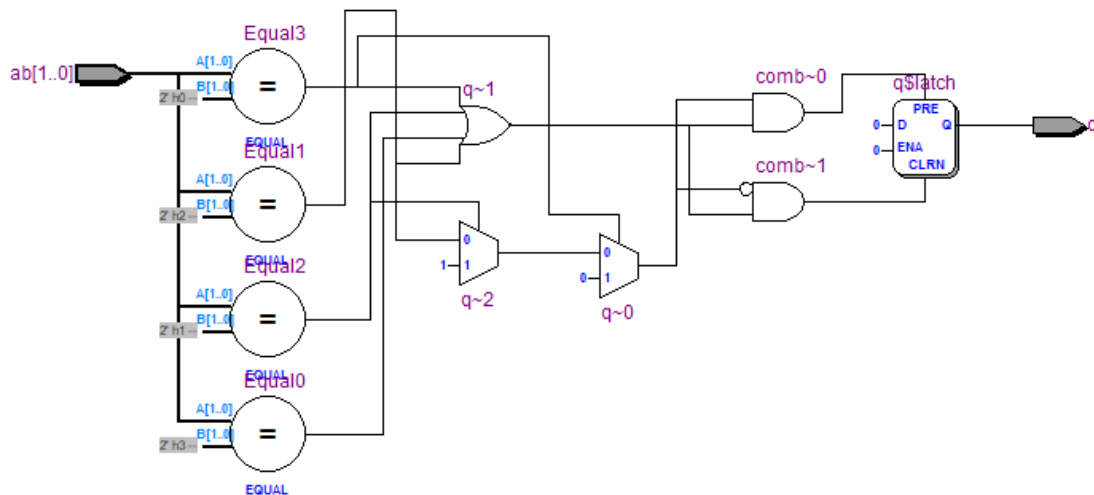
אם נשנה את הקוד הוא יעבור בלא כל ספק את הקומפילציה בלי שגיאות.

```
q<= '0' when ab="00" else
```

```
'1' when ab="01" else
```

```
'1' when ab="10" else
```

```
'0' when ab="11";
```



מהתרשמות ראשונה רואים מיד שמספר הרכיבים הוא גדול יותר.

יתר על כן, יש כאן דלגלג (flip-flop circuit) ומערכת זיכרון. התכלית של מערכות זיכרון שונה

מזו של מערכות צירופיות.

אם יש כאן דלגלג זה מרמז על מספר אפשרויות שונה ממה שנראה לעין.

נדון בזה בהמשך.

Process – תהליך

בתיאור של שפת ה-VHDL נשתמש בשני מרחבים: sequential לאירועים סדרתיים ו-concurrent לאירועים מקבילים. אירוע סדרתי מיוצג באמצעות תהליך (process), שכולל סדרה של פעולות המבוצעות לפי סדר ההופעה שלהן. את האירוע המקבילי תייצג קריאה מקבילה.

אתחול של משתנים הוא דוגמה לאירוע מקבילי. אפשר לצרף ביחד כמה רכיבים, למשל: **רֶבֶב (multiplexer), פֶּעֶנָח (decoder), דִּלְגָּלָג (flip-flop), מוֹנֶה (counter).**

כמו בכל אופרטור לוגי אחר, קשה מאוד להגדיר תהליך. לפיכך ננסה לבטא את התכונות שלו במקום לתת הגדרה אחת כוללת.

אחת מהתכונות הבסיסיות של התהליך היא סדרה של פעולות רציפות שבנויה על-פי סדר ההופעה שלהן.

התכונה השנייה, thread, היא תכונה מקבילית – שלא כמו במערכת CPU. מערכת כזו היא סדרתית בלבד, אלא אם כן משתמשים בכמה מעבדים.

אפשר לראות בכל תהליך מעגל לוגי בפני עצמו ולהפעיל תהליכים אחדים במקביל.

המבנה של התהליך הוא:

```
Process name : process (sensitivity list)

(

begin

)

end process;
```

process name – קביעת השם של התהליך.

sensitivity list – רשימה של הכניסות לתהליך, נרחיב את הדברים בהמשך.

begin – לכל תהליך מוסיפים פקודת begin, נוסף על ה-begin של ה-architecture.

(
)

הסוגריים קובעים את המסגרת של התהליך.

end process – בנוסף לסוגריים, הפקודה הזאת מעידה על הסיום של התהליך.

חוץ מהרשימה של הרגישויות, נראה שכל ההגדרות עד כאן ברורות.

sensitivity list

ובכן, הרשימה של הרגישויות מבטאת את אותות הכניסה שהתהליך יגיב עליהם. לשם הבהרה, אם יש במעגל לוגי 3 כניסות אנחנו חייבים לציין את כולן ברשימה, אחרת התהליך לא יגיב רק בגלל כניסה אחת שלא נרשמה.

דוגמה:

```
library ieee;

use ieee.std_logic_1164.all;

entity processXor is
port (
a,b :in std_logic;
    f :out std_logic);
end;

architecture arc_processXor of processXor is
begin
xorprocess:      process(a,b)
                    begin
                        f<=a xor b;
                    end process;
end arc_processXor;
```

אם נשמיט כניסה אחת (a או b) מרשימת הרגישויות, כל שינוי שיקרה בה לא ישפיע על התהליך עצמו.

לדוגמה: אם נשמיט את כניסה a, כל שינוי שיקרה בה לא ישפיע על f.

נבחן זאת בעזרת הסימולציה וניצור test bench.

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;


ENTITY processXor_vhd_tst IS

END processXor_vhd_tst;

ARCHITECTURE processXor_arch OF processXor_vhd_tst IS

-- constants

-- signals

SIGNAL a : STD_LOGIC:='0';

SIGNAL b : STD_LOGIC:='0';

SIGNAL f : STD_LOGIC;

COMPONENT processXor

    PORT (

        a : IN STD_LOGIC;

        b : IN STD_LOGIC;

        f : OUT STD_LOGIC

    );

END COMPONENT;

BEGIN

    i1 : processXor

    PORT MAP (

-- list connections between master ports and signals

        a => a,

        b => b,

        f => f

    );

    a<='0' after 200 ns,'1' after 400 ns,'0' after 600 ns,'1' after 800 ns;

    b<='0' after 200 ns,'0' after 400 ns,'1' after 600 ns,'1' after 800 ns;

END processXor_arch;
```

	Msgs						
processxor_vhd_tst/i1/a	0						
processxor_vhd_tst/i1/b	0						
processxor_vhd_tst/i1/f	0						

זאת הסימולציה הרגילה, אפשר לעקוב אחרי ההתנהגות של f לפי הכניסות a ו-b.
 כעת נבטל את הכניסה a מרשימת הרגישויות (sensitivity list).

```
xorprocess:process(b)

begin

f<=a xor b;

end process;
```

	Msgs						
/a	0						
/b	0						
/f	1						

	a=0	a=0	a=1	a=0	a=1	a=1	a=0
	b=0	b=0	b=0	b=1	b=1	b=0	b=0
	f=0	f=0	f=0	f=1	f=1	f=1	f=1
			שינוי ב-	שינוי ב-	שינוי ב-	שינוי ב-	שינוי ב-
			a ללא	b ושינוי	a ללא	b ללא	a ללא
			שינוי ב-	ב-	שינוי ב-	שינוי ב-	שינוי ב-
			f	f	f	f	f
						כצפוי	

הסינטיסיזר כותב אזהרה רצינית שיש לשים לב אליה ולהביא אותה בחשבון:

Warning (10492): VHDL Process Statement warning at processXor.vhd(16):
signal "a" is read inside the Process Statement but isn't in the Process Statement's sensitivity list.

הסימן a נמצא בתהליך אבל לא ברשימת הרגישויות שלו.

המשמעות של התוצאה היא מעניינת מאוד. לשינוי ב-a בלי שינוי ביציאה f – יש רק מובן אחד: קיים מנגנון שפשוט שומר על הערך. זה מזכיר לכם לבטח רק דבר אחד – **מעגל דלגלג** flip-flop (flop circuit), המנגנון היחיד שידוע כיצד לשמור. התופעה הזאת נקראת latch – נעילה. התוצאה שתיגרם בגלל הנעילה תהיה שונה מההתנהגות הרגילה.

כלל גדול בתכנון של מערכות לוגיות עם תהליך: לא להחסיר לעולם את הכניסות מרשימת הרגישויות.

Sequence and signals

מערכת צירופית-סדרתית ומשתנים.

הנושא הבא הוא אחד מהנושאים הרגישים של שפת ה-vhdl.

Sequence היא מערכת סדרתית, כלומר רצף של פעולות על-פי סדר מבני.

אולם זו גם מערכת צירופית, שבה היציאה מגיבה רק על אותות מהכניסות וללא משוברים.

דוגמה למערכת סדרתית: בניית מסכם מלא ממסכם למחצה בתהליך אחד בלבד.

המשוואות של המסכם למחצה הן:

```
S<=a xor b xor c;
Cout<=a and b or Cin(a or b);
```

אפשר להחיל פעולת xor על שני משתנים בלבד. כלומר צריך "לאחסן" את תוצאת הביניים של הפעולה הראשונה במשתנה זמני ואז להשלים את שאר הפעולות.

```
Z<=a xor b;
S<=z xor c;
```

קיבלנו מבנה סדרתי – סדרה של פעולות. פרויקט נכון הוא כזה שמאופיין כראוי על-פי הפונקציונליות שלו. זה נושא רחב מאוד ודרוש בו ניסיון רב, ובוודאי בכל הנוגע לכתיבה של פעולות סדרתיות.

למבנה כזה בתהליך process נדרשת הגדרה של משתנה ביניים: signal או variable.

ההבדל ביניהם בולט מאוד – signal מוגדר מחוץ לתהליך, ואילו variable מוגדר בתוכו.

נבדוק כמו כן את כלל הברזל של רשימת הרגישויות. יש סתירה כפי שכבר אפשר לראות: משתנה ביניים כדוגמת signal לא מתפקד בתור כניסה למעגל, אבל תהליך ה-process עדיין צריך לדבוק ברשימת הרגישויות. את הסתירה הזאת נפתור באמצעות המשתנה variable, שמוגדר בתוך התהליך בלבד.

נסכם את מה שלמדנו עד כאן: עלינו לבחון מבנה סדרתי שמוגדר בעזרת משתנה ביניים.

Signal: זהו משתנה ביניים שיש לו ממד של זמן. לכל signal יש מנגנון השהיה שנקרא delta.

דוגמא 2

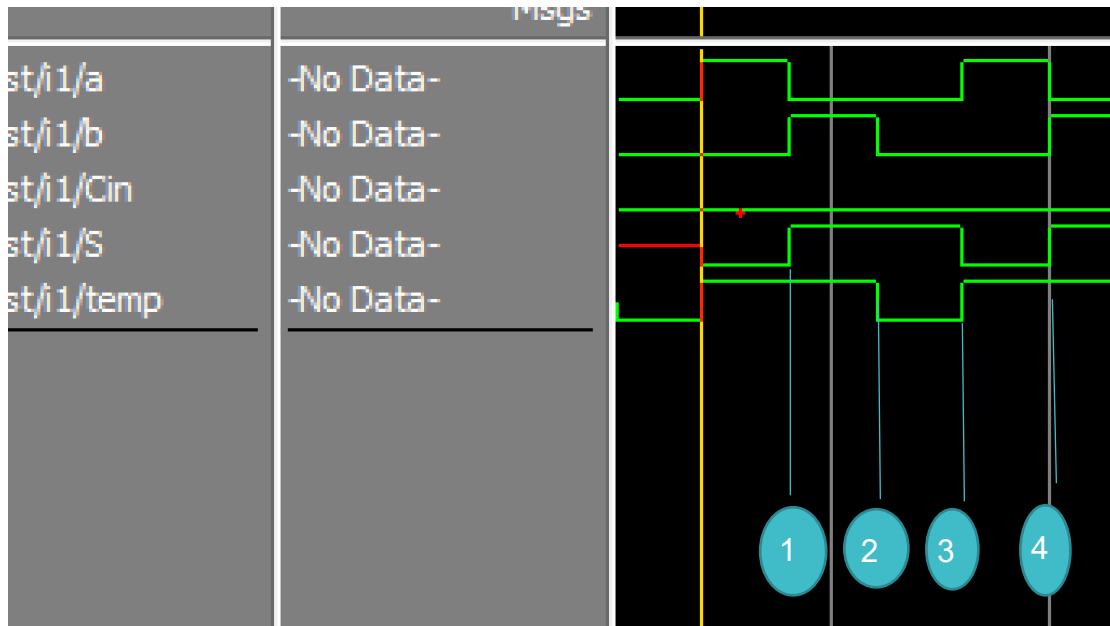
מקום ההגדרה של ה-signal נמצא אחרי ה-architecture ולפני ה-begin.

```
library ieee;

use ieee.std_logic_1164.all;

entity fullAdder is
port (
a,b,Cin :in std_logic;
S,Cout :out std_logic
);
end;

architecture arc_processXor of fullAdder is
signal temp :std_logic;
begin
FullAdderpro:                                process(b,a,Cin)
begin
temp<=a xor b;
S<=temp xor Cin;
end process;
end arc_processXor;
```



יש לשים לב היטב לשינויים ב-temp, הקווים הכחולים מציינים את פרקי הזמן. נזכור שה-temp לא נמצא ברשימת הרגישויות, ולכן אצלו השינויים לא מתבטאים ביציאה. משמאל לבלון 1 ה-temp משתנה, אבל היציאה S לא משתנה. בין בלון 1 לבלון 2 היציאה S משתנה בעקבות שינוי ב-temp. בין בלון 2 לבלון 3 נקבל שוב מצב זהה: ה-temp משתנה, אבל היציאה S לא משתנה. ראינו כאן מבנה סדרתי רציף שלא מתנהג כמו מעגל לוגי על-פי הלוגיקה הבסיסית. הפתרון הוא להוסיף משתנה מסוג אחר. אנחנו זקוקים למשתנה פנימי בתהליך, משתנה שלא תלוי ברשימה של הרגישויות. משתנה מסוג כזה נקרא variable. Variable לא נחוץ ברשימת הרגישויות, וכל שינוי שקורה בו מתעדכן מיד בלי תלות ביחידה של זמן.

```
library ieee;

use ieee.std_logic_1164.all;

entity fullAdderVariable is
port (
a,b,Cin :in std_logic;
S,Cout :out std_logic
);
end;

architecture arc_processXor of fullAdderVariable is
begin
FullAdderpro:process(b,a,Cin)
variable temp : std_logic:='0';
begin
temp:=a xor b;
S<=temp xor Cin;
end process;

end arc_processXor;
```

		Msgs			
/a	1				
/b	1				
/Cin	0				
/S	0				

אפשר לראות בקלות שהשגיאות אכן תוקנו. עקבו אחרי הכניסות ואחרי היציאות.

האופרטורים if ו-case בתהליך ה-process

מעגלי דלגלג – flip-flop circuits

מעגלי דלגלג הם מאבני הבנייה החשובות של המערכות הלוגיות. אפשר למצוא אותם במונים, במערכות סינכרוניות ולא סינכרוניות ובמכונות מצבים. הם גם רכיב חשוב מאוד בתוך שיטה להאצת תהליכים שנקראת pipe line. התפקיד שלה הוא לקצר את משך הזמן של התהליך.

```
library ieee;

use ieee.std_logic_1164.all;

entity srffn is
port (
s,r :in std_logic;
q :out std_logic);
end;

architecture arc_srff of srffn is
signal sq: std_logic;
begin
srffq:process(s,r)

begin
if s='0' and r='0' then
sq<=sq;
elsif s='0' and r='1' then
sq<='0';
elsif s='1' and r='0' then
sq<='1';
else
sq<='0';
end if;
end process;

q<=sq;

end;
```

בתהליך הזה בא לידי ביטוי האופרטור if, כל אפשרות אחרת מבטאת באמצעות elsif. כדי לכסות את שאר האפשרויות חשוב מאוד לסיים את התהליך.

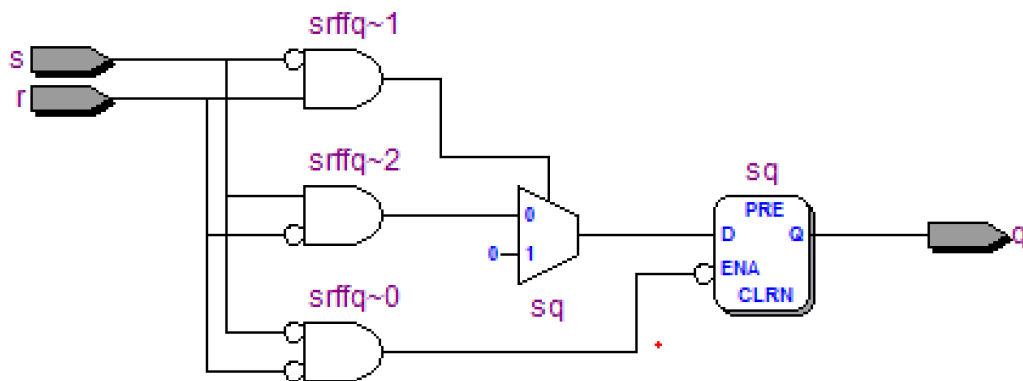
ביתר פירוט: האפשרות הראשונה היא הערכים $r=0$; $s=0$ עם האופרטור if.

האפשרות השנייה היא הערכים $r=1$; $s=0$ עם האופרטור elsif.

האפשרות השלישית דומה לאפשרות השנייה מבחינת התחביר.

באפשרות הרביעית קיים האופרטור else, כדי לכסות את כל האפשרויות אם לא הגדירו אותן.

כלל ברזל: אסור להשאיר אפשרויות בלתי גמורות שלא טיפלנו בהן.



Srffq~0

Srffq~0 מזהה את מצב הכניסות ($r=0$; $s=0$). הערך ביציאה של השער and יהיה 1. הדלגלג sq יהיה חסום, משום שהערך ב-ena שלו הוא 1 בזמן שהוא פעיל במצב של 0 active (low – פעיל בתדר נמוך).

כאשר $s=0$; $r=1$ הערך ביציאה של Srffq~0 יהיה 0 בגלל המהפכים שבכניסה. לפיכך הדלגלג יהיה פעיל.

הערך ביציאה של Srffq~1 יהיה 1, ועל כן הערך ביציאת הרבב של sq יהיה 0. מפני הפעילות של הדלגלג, היציאה שלו (q) תקבל את הערך של הכניסה (d).

$r=0$; $s=0$ הוא מצב מעניין. במצב הזה היציאה q חסומה בפני שינויים, ולכן הדלגלג נמצא במצב נעילה. $ena=1$ היא למעשה המשמעות של $sq \leq sq$.

מצב הנעילה נקרא latch.

כבר ראינו שמצב נעילה לא נחוץ גורם לשגיאה לוגית. עשינו זאת כשהשתמשנו ב-signal ולא ציינו אותו ברשימה של הרגישויות.

Srff באמצעות האופרטור case

```

library ieee;

use ieee.std_logic_1164.all;

entity srffcase is
port (
sr :in std_logic_vector(1 downto 0);
q :out std_logic);
end;

architecture arc_srff of srffcase is
signal sq: std_logic;
begin
srffq:process(sr)
begin
case sr is
when "00" =>
sq<=sq;
when "01" =>
sq<='0';
when "10" =>
sq<='1';
when "11" =>
sq<='0';
end case;
end process;
q<=sq;
end;

```

בשני המקרים של האופרטורים if ו-case משתמשים ב-signal ולא ב-variable, מכיוון שהמערכת לא רציפה. אין כאן תלות בתוצאת ביניים, אבל נוצרת בעקבות זאת בעיה אחרת.

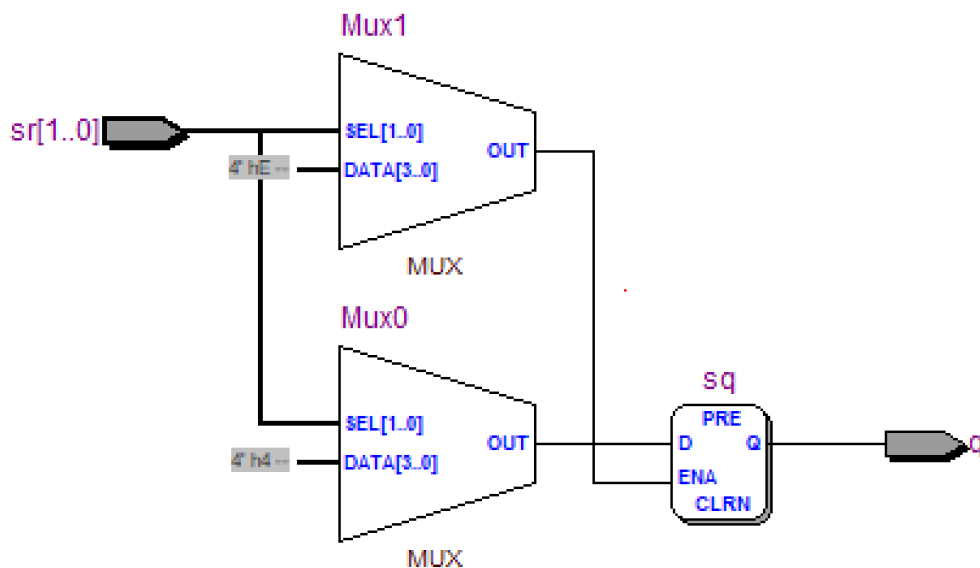
מדוע נשתמש ב-signal?

היציאה port מוגדרת בתור out ואי-אפשר לקרוא ממנה, לעומת inout שהוא כניסה וגם יציאה. מפני שאין לנו אפשרות לקרוא ערכי port מתוך יציאה מסוג out, ניעזר ב-signal. המערכת אינה רציפה אז אפשר לעדכן את היציאה מיד, וככה לא תהיה לנו שגיאה בסימולציה.

Sq הוא משתנה מסוג signal במקרה הזה, אבל הוא לא קשור ליציאה q.

לכן השורה $q \leq sq$ היא חיבור פיזי בין המשתנה sq ליציאה q.

מעניין שהיה אפשר לכתוב את השורה הזאת כבר בתוך תהליך ה-process, אלא שחיבור מבחוץ לא תלוי בהתעוררות התהליך. רק חיבור מבפנים יהיה תלוי בהתעוררות של תהליך ה-process.



כניסת ה-data של הרבב הראשון (mux1) היא 1110 בבסיס בינארי, כלומר E בבסיס הקסדצימלי (בסיס 16). כניסת ה-data של הרבב השני (mux0) היא 0100 בבסיס בינארי, כלומר 4 בבסיס הקסדצימלי.

הרבב mux1 שולט על ה-ena, הרבב mux0 שולט על ה-data.

דוגמה: במצב של $sr=00$ הערך ביציאה out של הרבב mux1 הוא 0 – כלומר יש ena של 0 בדלגלג sq, ולכן היציאה q נשארת בלי שינוי.

מערכות סינכרוניות

במערכות צירופיות (combinatorial systems) לא מוגדר לנו הזמן של הביצוע, כי מספר הרכיבים בכל מסלול לא ידוע מראש והוא משתנה מתכנון לתכנון. החשיבות של זמן הביצוע היא, שבלעדיו לעולם לא נדע מתי לאסוף את הנתונים הנחוצים לנו. תארו לעצמכם שאתם מתכננים להגיע למקום מסוים, אבל בלי היכולת להעריך את זמן ההגעה של האוטובוס.

מערכות סינכרוניות הן מערכות שתלויות בזמן, והן יכולות לדעת מראש את זמן הביצוע. תשאלו בוודאי את השאלה הזאת: אם זמן הביצוע לא ברור כיצד אפשר לשלוט בו באמצעות שעון?



המערכת עטופה במעגלי דלגלג. גם אם זמן הביצוע לא ידוע, התוצאה ביציאה של הדלגלג השני תהיה תלויה בשעון. אם יש כמה מסלולים כאלה אפשר להתחשב במסלול הארוך ביותר.

המתכנן מסוגל לשלוט במערכות סינכרוניות

קיימות שיטות כדי להאיץ תהליכים. נוכל לעשות זאת למשל אם נחצה את הענן ונוסיף עוד דלגלג – לשיטה הזאת קוראים pipe line.

אם כן, מערכת סינכרונית תלויה בשעון ופועלת באמצעות תהליך. ברשימה של הרגישויות שלה נמצא לא רק שעון, אלא גם מערכת אתחול – reset. הפקודה reset מאפסת בדרך כלל את מעגלי הדלגלג. מאוחר יותר נראה שבמכונת מצבים ה-reset מאתחל במפורש את המשתנים.

במערכות לא סינכרוניות (asynchronous systems) נשים ברשימת הרגישויות (sensitivity list) את כל המשתנים שבשבילם נדרש התהליך (process).

במערכות סינכרוניות (synchronous systems) נציין רק את המשתנים clock ו-reset.

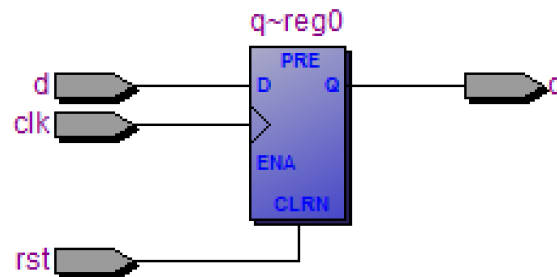
דוגמה של דלגלג סינכרוני (synchronous flip-flop):

```
library ieee;

use ieee.std_logic_1164.all;

entity dffq is
port (
    clk :in std_logic;
    rst :in std_logic;
    d :in std_logic;
    q :out std_logic);
end;

architecture arc_dff of dffq is
signal sq: std_logic;
begin
    process(clk,rst)
    begin
        if rst='1' then
            q<='0';
        elsif clk'event and clk='1' then
            q<=d;
        end if;
    end process;
end;
```



הציור הגרפי הזה מתאר את התוצאה של התכנון.

נא לשים לב לדלגלג שמגיב על העלייה של השעון ולאيفוס שנעשה בעזרת clear.

תכונות של תהליך סינכרוני (synchronous process) בשביל שעון

אתחול באמצעות reset, במקרה הזה מדובר באיפוס.

```
clk'event and clk='1' then if
```

כרגע נציין ש-event הוא אירוע, כלומר שינוי ב-clk במקרה הזה.

לקטע '1' יש רק משמעות אחת: השינוי הוא מ-0 ל-1. זה בדיוק המובן של "עליית שעון". לכן העניין בכל השורה הזאת (if clk'event and clk='1' then) הוא זיהוי של עליית שעון.

ואילו הביטוי elsif clk'event and clk='1' then מזהה את הירידה של השעון.

שימו לב לאופרטור גרש '<' >', כפי שמתואר בפרק המבוא לשפת ה-vhdl.

לא נשתמש ב-if כפי שראינו כאן, אלא ב-elsif.

הציור הגרפי לעיל מתאר רכיב בסיסי בסביבת העבודה של altera, רכיב שקראנו לו פרימיטיבי (primitive).

המבנה הוא **מדרג** (hierarchical) – יש סדר עדיפויות למופע של התנאי. התנאי הראשון הוא בעדיפות הגבוהה ביותר, התנאי השני בעדיפות שנייה וכן הלאה.
ל-rst העדיפות הגבוהה ביותר, ואחריו נמצא ה-elsif בעדיפות משנית.

כניסה סינכרונית וכניסה לא סינכרונית

כניסה סינכרונית לא תלויה בשעון – בכל פעם שמתקיים התנאי של ה-rst, תהליך ה-process מיישם אותו מיד. הסברנו כבר את הסיבה: התהליך מתעורר רק כשיש שינוי ברשימה של הרגישויות. אם קיים שינוי ב-clk וגם ב-rst ולא רק ה-rst משתנה – הביצוע יהיה לטובת ה-rst.
clk הוא השעון, כל מה שנמצא תחת הגדרתו (כמו $q \leq d$) יהיה בהתאם אליו.

Clock enable

מטעמי תזמון נדרשת גם כניסה סינכרונית בשביל תכנון נכון. במקרה כזה אפשר להוסיף עוד תנאי, גם זה תלוי כמובן בשעון.

נתבונן בשלבים של התכנון בדוגמה הבאה:

```
library ieee;

use ieee.std_logic_1164.all;

entity dffq is
port (
    clk :in std_logic;
    rst :in std_logic;
    d,enable :in std_logic;
    q :out std_logic);
end;

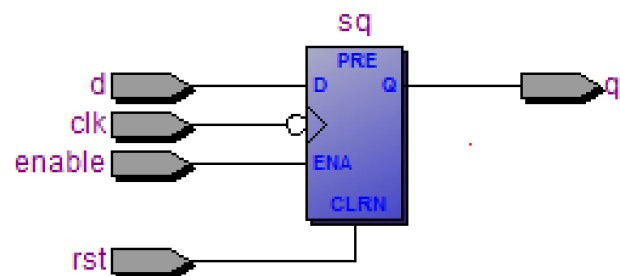
architecture arc_dff of dffq is
signal sq: std_logic;
begin

    process(clk,rst)
    begin
        if rst='1' then
            sq<='0';
        elsif clk'event and clk='0' then
            if enable='1' then
                sq<=d;
            else
                sq<=sq;
            end if;
        end if;

    end process;

    q<=sq;

end;
```



כאן השתמשנו שוב ב-signal ולא ב-variable כדי להגדיר את היציאה.
אפשר לעשות את זה כמובן גם באמצעות variable לפי הדוגמה הבאה:


```
library ieee;

use ieee.std_logic_1164.all;

entity dffq is
port (

    clk :in std_logic;

    rst :in std_logic;

    d,enable :in std_logic;

    q :out std_logic);

end;

architecture arc_dff of dffq is
--signal sq: std_logic;

begin

    process(clk,rst)

    variable sq: std_logic;

    begin

    if rst='1' then

        sq:='0';

    elsif clk'event and clk='0' then

        if enable='1' then

            sq:=d;

        else

            sq:=sq;

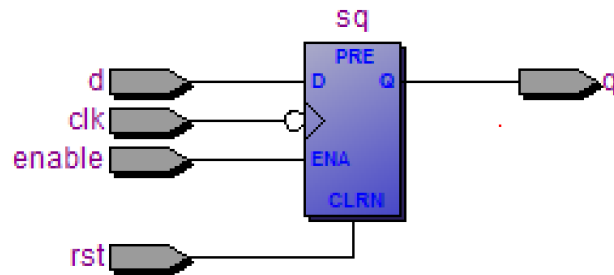
        end if;

    end if;

    q<=sq;

    end process;

end;
```



לצורך ההמחשה ה-signal הושאר, אבל הוא נמחק אחר כך על-ידי הערה של שני מינוסים – במקומו שמנו את ה-variable. במקרה הזה הסינטיסייזר (synthesizer) מתרגם את שתי ההשמות באותו האופן.

גם מעגל הדלגלג הזה מגיב על ירידה של שעון.

דבר נוסף: מהלימודים על מערכות ספרתיות בוודאי זכור לכם שיש שתי יציאות הופכיות זו מזו, לדוגמה q ו- $\bar{q}$. בשפת vhdl לא חייבים להוסיף את זה, כי המקרים שבהם צריך את שתי היציאות הללו הם מעטים, ומניסיון כמעט שלא נזקקים לשתייהן.

דלגלג מסוג Tff

```
library ieee;

use ieee.std_logic_1164.all;

entity tffq is
port (
        clk :in std_logic;
        rst :in std_logic;
        q :out std_logic);
end;

architecture arc_tff of tffq is
begin
        process(clk,rst)
        variable sq :std_logic;
        begin
        if rst='1' then
                sq:='0';
        elsif clk'event and clk='0' then
                sq:=not sq;
        end if;
        end process;
end;
```

הקוד של הסימולציה בעבור דלגלג ה-tff:

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;


ENTITY tffq_vhd_tst IS

END tffq_vhd_tst;

ARCHITECTURE tffq_arch OF tffq_vhd_tst IS

  -- constants

  -- signals

  SIGNAL clk : STD_LOGIC:='0';

  SIGNAL q : STD_LOGIC;

  SIGNAL rst : STD_LOGIC;

  COMPONENT tffq

    PORT (

      clk : IN STD_LOGIC;

      q : OUT STD_LOGIC;

      rst : IN STD_LOGIC

    );

  END COMPONENT;

BEGIN

  i1 : tffq

    PORT MAP (

-- list connections between master ports and signals

      clk => clk,

      q => q,

      rst => rst

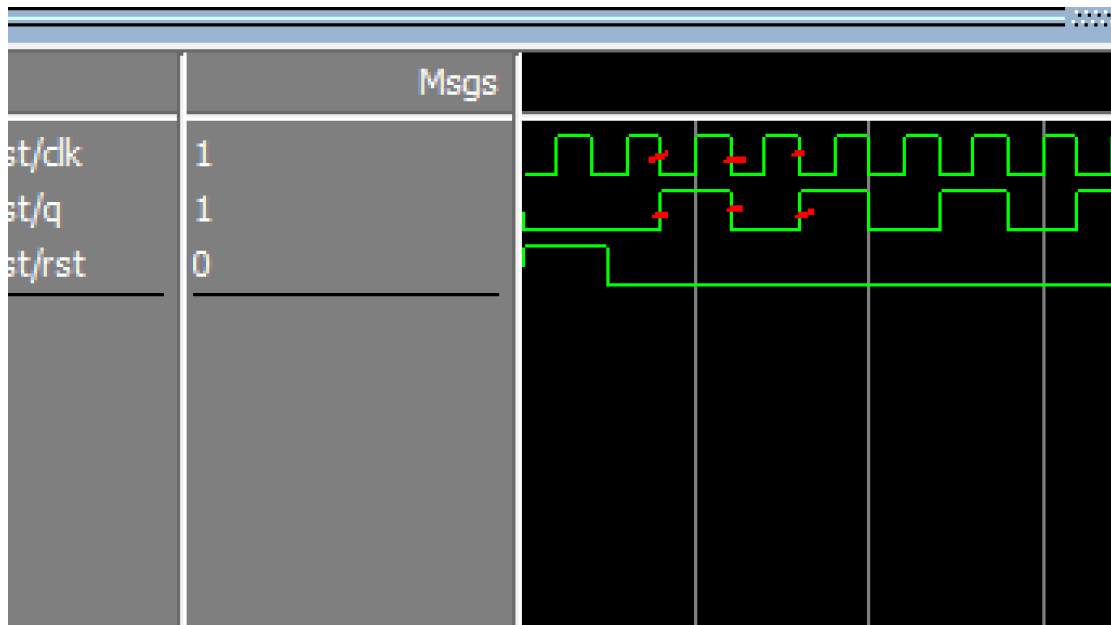
    );

  clk<= not clk after 20 ns;

```

```
rst<='1','0' after 50 ns;
```

```
END tffq_arch;
```



אפשר לראות על-פי הסימולציה שהיא מגיבה על ירידת שעון.

הסימנים האדומים מסמלים את התגובה של היציאה q על הירידה של השעון (clk).

ה-rst נמצא במצב של 1 (50 ns) ויורד לכיוון מצב של 0.

התחביר של הסימולציה הוא די קריא.

הדוגמה הבאה שניתן היא מעניינת ביותר.

אוגר הזזה – Shift Register

נתבונן באופן שבו הדוגמה הזאת בנויה על-פי הקוד הבא:

```
library ieee;

use ieee.std_logic_1164.all;

entity shiftRegister is
port (
    clk :in std_logic;
    rst :in std_logic;
    d :in std_logic;
    q :out std_logic);
end;

architecture arc_shiftRegister of shiftRegister is
-----signal sq: std_logic;
```

המשך הקוד בעמוד הבא <<

```

begin

    process(clk,rst)

        variable sq,sq1,sq2 :std_logic:='0';

        begin

            if rst='1' then

                q<='0';

            elsif clk'event and clk='0' then

                sq:=d;

                sq1:=sq;

                q<=sq1;

            end if;

        end process;

    .

```

המימוש מעניין מאוד, אבל לרצף של ההוראות האלה חסר ממד של זמן:

```

sq:=d;

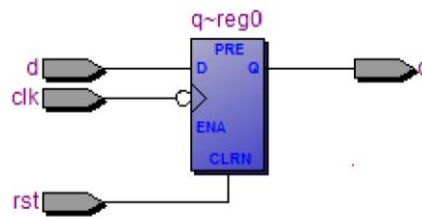
sq1:=sq;

```

אוגר הזזה הוא ביסודו מערכת של זיכרון, אבל להוראה `sq1:=sq` חסר ממד של זמן. לפיכך היא לא יכולה ליצור דלגלג (flip-flop). אין בחיבורים האלה ממד של זמן, ובאמת ה-synthesizer מחשיב אותם לחיבור אחד.

החיבור הזה זהה כמעט לחלוטין לסוג החיבור שלמדנו עליו בלימודי החשמל.

מעניין שבחיבור החשמלי ההוא יש חיבור בין variable ל-signal. אז יש גם ממד של זמן, וה-synthesizer מתרגם את החיבור הזה לדלגלג.



Shift register עם signal

אפשר לכתוב את הקוד הזה כשורה רק באמצעות signal. מפני שלכל דרגה דרוש ממד אחר, רק signal יהיה מתאים למשימה זו.

```
library ieee;

use ieee.std_logic_1164.all;

entity shiftRegisterSignal is
port (

    clk :in std_logic;

    rst :in std_logic;

    d :in std_logic;

    q :out std_logic);

end;

architecture arc_shiftRegister of shiftRegisterSignal is

signal sq,sq1: std_logic;

begin
```

המשך הקוד בעמוד הבא <<


```

process(clk,rst)
begin
  if rst='1' then
    q<='0';
    sq<='0';
    sq1<='0';

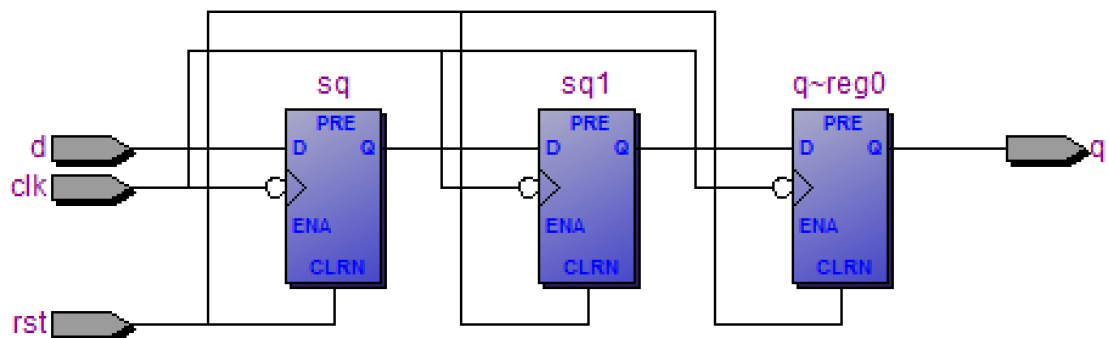
  elsif clk'event and clk='0' then

    sq<=d;
    sq1<=sq;
    q<=sq1;

  end if;
end process;

end;

```



התוצאה המיוחלת: שלושה מעגלי דלגלג שמזינים זה את זה.

הקורא ודאי ישאל את עצמו ובצדק: מה בדבר הסימולציה? הרי בדוגמה של srff הייתה בעיה בסימולציה וה-variable פתר אותה, וכאן אנחנו משתמשים ב-signal ואילו ה-variable הוא הבעייתי.

זה באמת נכון, אבל הקורא צריך להתחשב ברשימה של הרגישויות. במבנה סינכרוני היא מכילה את clock ואת reset בלבד.

כל דלגלג מתעורר בעקבות שינוי כלשהו בשני משתנים אלו (clock ו-reset).

להלן הקוד של הסימולציה:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY shiftRegisterSignal_vhd_tst IS
END shiftRegisterSignal_vhd_tst;

ARCHITECTURE shiftRegisterSignal_arch OF shiftRegisterSignal_vhd_tst IS

SIGNAL clk : STD_LOGIC:='0';

SIGNAL d : STD_LOGIC:='0';

SIGNAL q : STD_LOGIC;

SIGNAL rst : STD_LOGIC:='0';

COMPONENT shiftRegisterSignal

PORT (

clk : IN STD_LOGIC;

d : IN STD_LOGIC;

q : OUT STD_LOGIC;

rst : IN STD_LOGIC

);

END COMPONENT;
```

המשך הקוד בעמוד הבא <<

```

BEGIN

    i1 : shiftRegisterSignal

    PORT MAP (

        clk => clk,

        d => d,

        q => q,

        rst => rst

    );

    clk<= not clk after 20 ns;

    rst<='1','0' after 200 ns;

    d<='1' after 250 ns,'0' after 500 ns,'1' after 750 ns,'0' after 1000 ns;

END shiftRegisterSignal_arch;

```

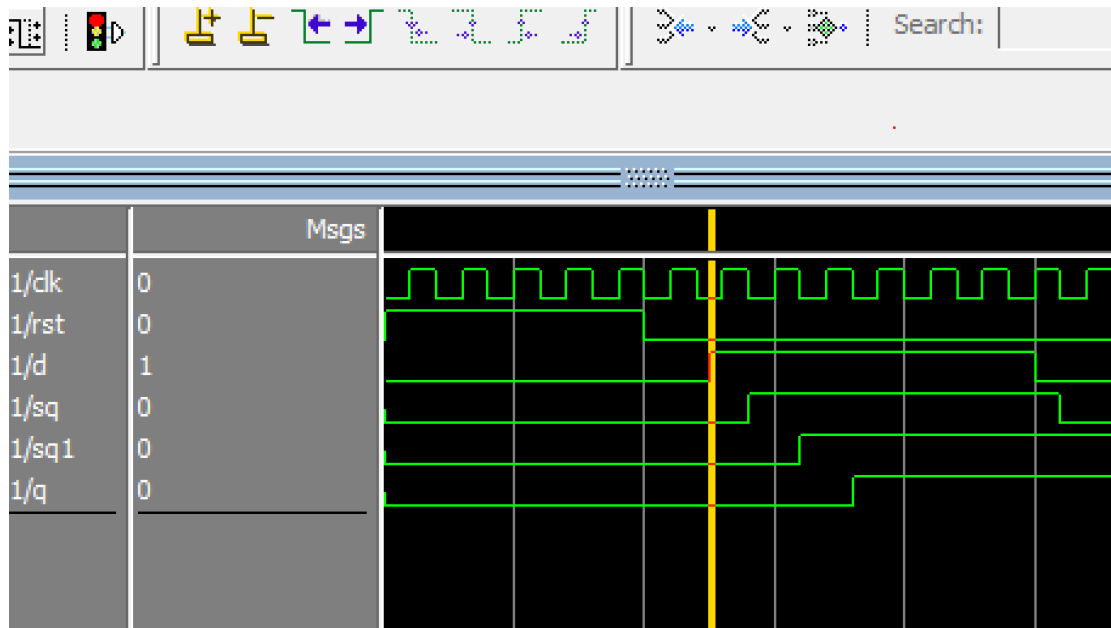
הסימולציה מחוללת את d בגל ריבועי בכל 250 ns (ננו־שניות).

פעולת not מתקיימת בשעון (clk) בכל 20 ns (ננו־שניות).

הערך ההתחלתי ב-rst הוא 1 לאחר 200 ns (ננו־שניות), ואז הוא יורד ל-0.

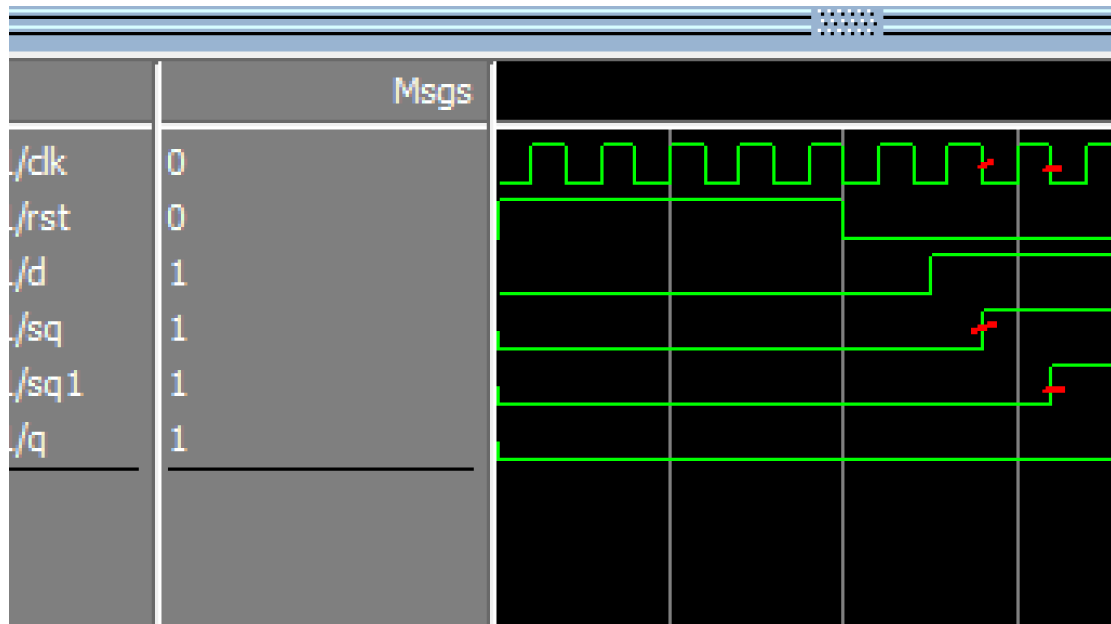
שימו לב שהכניסה d מתקדמת על ציר הזמן בהפרשים של 250 ns (ננו־שניות).

התוצאה של הסימולציה נתונה לפניהם בציור הבא:



ברגע שמקבלים את הערך $d=1$, כל דלגלג משתנה על-פי הירידה של השעון.

הסימנים האדומים על הקווים הירוקים ב- clk , ב- sq וב- $sq1$ מציינים את התגובה של מעגלי הדלגלג על הירידה של השעון.



Counter – מונה

מונים הם נדבך חשוב מאוד בתכנון של חומרה. הם נחוצים לא רק בשביל ספירה, אלא אף כדי לבקר תהליכים.

זכור לכם בוודאי עד כמה קשה ליישם מונים סינכרוניים במערכות לוגיות, בייחוד אם אמצעי התכנון הוא מכונת מצבים.

והנה יש לנו שפה שאפשר ליצור בה מונה באמצעות תיאור של חומרה.

תכנון xor סינכרוני בתור פתרון

בדוגמה 2 (שער של מסכם מלא full adder) השתמשנו ב-signal אבל השמטנו את temp מרשימת הרגישויות, ולכן קבלנו שגיאה.

נסתכל על דוגמה דומה, אבל הפעם היא תהיה סינכרונית.

```
library ieee;

use ieee.std_logic_1164.all;

entity fullAddersync is

port (

clk,rst :std_logic;

a,b,Cin :in std_logic;

S,Cout :out std_logic

);

end;

architecture arc_processXor of fullAddersync is

signal temp :std_logic;

begin

FullAdderpro:process(clk,rst)

begin

if rst='1' then

s<='0';

temp<='0';

elsif clk'event and clk='1' then

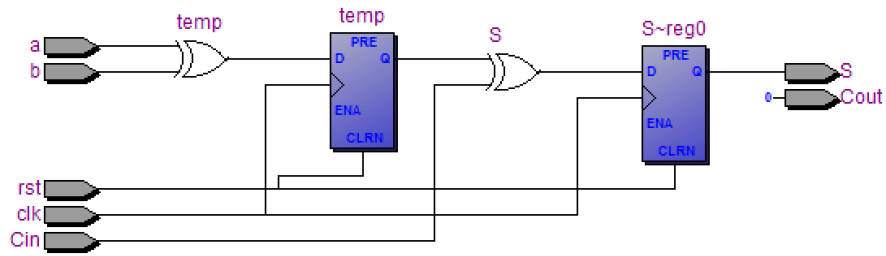
temp<=a xor b;

S<=temp xor Cin;

end if;

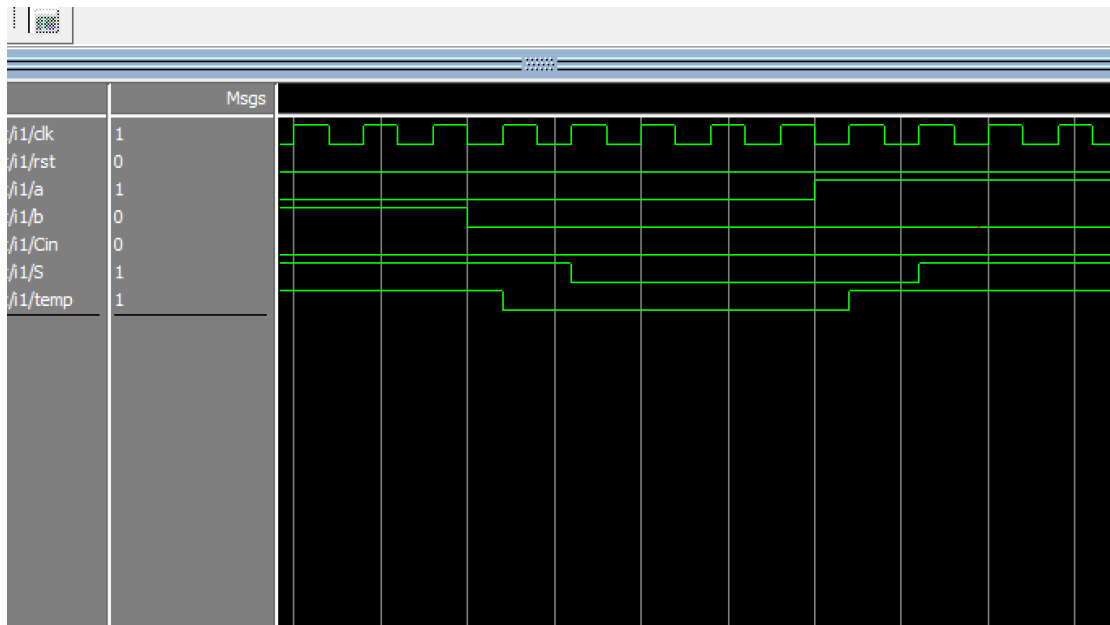
end process;

end arc_processXor;
```



לפי התרגום של הקוד רואים בציור שני מעגלי דלגלג בין שני מעגלים צירופיים. לפתרון מהסוג הזה קוראים **pipe line**.

הסימולציה מראה שלא צריך לרשום את המשתנה temp ברשימה של הרגישויות.



הקורא מוזמן לעקוב אחרי השינויים בכניסות וביציאה S.

דוגמה: מונה עד המספר 7

יש לתכנן מונה טבעתי שסופר עד המספר 7 בלבד.

המונה מגיב על ירידה של שעון, ואפשרות האיפוס שלו אינה סינכרונית.

```
library ieee;

use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity count7 is
port (
    clk :in std_logic;
    rst :in std_logic;
    q :out std_logic_vector(2 downto 0));
end;

architecture arc_tff of count7 is
--signal sq: std_logic;
begin
    process(clk,rst)
        variable sq :std_logic_vector(2 downto 0);
        begin
            if rst='1' then
                sq:="000";
            elsif clk'event and clk='0' then
                sq:=sq +'1';
            end if;
            q<=sq;
        end process;
    end;
```


לפי הביטוי $2^3 - 1$, למונה טבעתי עד המספר 7 נדרשים שלושה ביטים. בספירה הזאת קיימות שמונה אפשרויות, אבל המונה יספור רק שבע מהן. המשתנה שנשתמש בו הוא סטטי, ולכן אפשר לעשות בעזרתו אקומולציה ולהגיע לתוצאה מצטברת.

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;


entity count7 is

port (

        clk :in std_logic;

        rst :in std_logic;

        q :out std_logic_vector(2 downto 0));

end;

architecture arc_tff of count7 is

begin

        process(clk,rst)

        variable sq :std_logic_vector(2 downto 0);

        begin

        if rst='1' then

                sq:="000";

        elsif clk'event and clk='0' then

                sq:=sq + '1';

        end if;

        q<=sq;

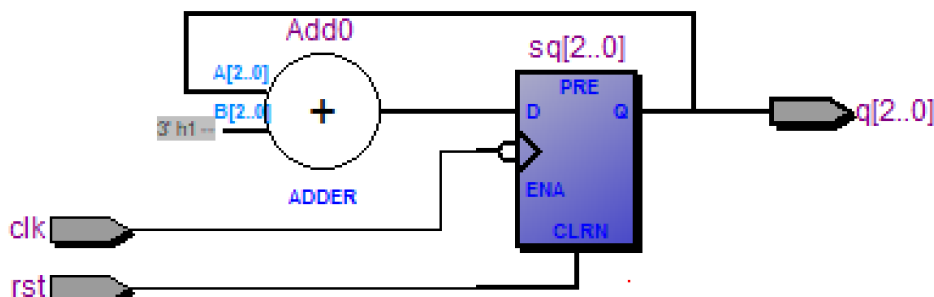
        end process;

end;
```

Sq הוא משתנה סטטי, וכל ירידה של שעון מקדמת אותו ב-1.

בתהליך לא סינכרוני חשוב להקפיד שהרשימה של הרגישויות תהיה מלאה. בתהליך סינכרוני הרשימה אכן מצטמצמת ל-clock ול-reset בלבד, אבל אנחנו חייבים לאתחל את כל המשתנים ב-reset. אחרת נקבל התנהגויות שלא מתאימות לציפיות שלנו.

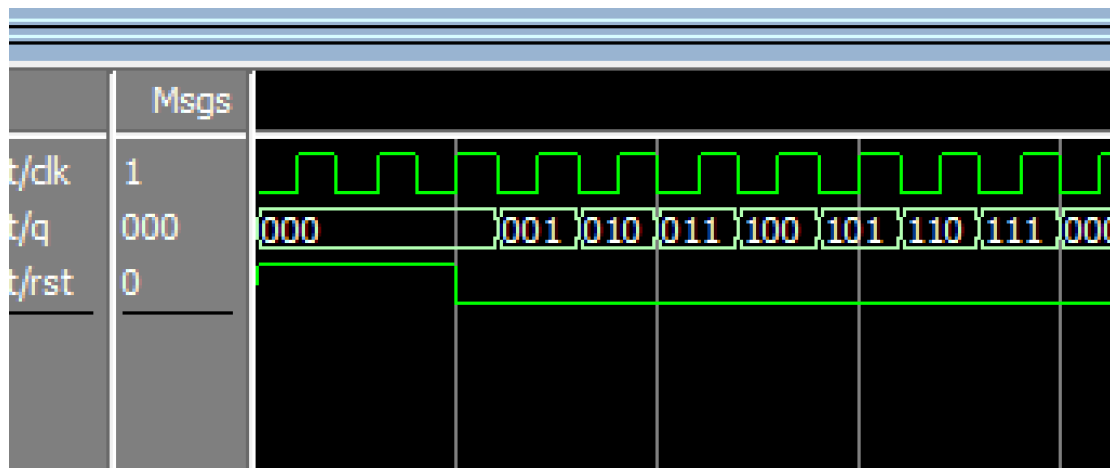
כזכור, כל שינוי מעורר את התהליך. במקרה שלנו זה גם מקדם את sq בתור משתנה סטטי. התכלית של המשתנה הסטטי אינה אחרת מתכליתו במערכת זיכרון.



בתוצאה של השרטוט הזה יש דברים מעניינים, והיא שונה לגמרי מתוצאות שרטוט של מערכות לוגיות.

הכניסה הראשונה למסכם היא בעצם היציאה של הדלגלג q, ובכניסה השנייה שלו הערך הוא 1. הדלגלג מתעדכן בכל פעם שמתרחשת ירידה של שעון. משתנה סטטי שומר על הערך שלו גם אם המערכת לא פעילה. את זה בדיוק עושה הדלגלג, שנקרא כאן Latch. תהליך ההצטברות (אקומולציה) מתקיים באמצעות המסכם (Adder) והנועל (Latch).

ה-quartus יכול לממש את המערכת בעזרת מערכות בסיסיות ורכיבים פרימיטיביים (כמו מסכם ודלגלג).



אפשר לראות כאן ספירה של מחזור אחד מ-0 ועד 7.

איפוס סינכרוני

דוגמה: מונה עשרוני – מונה שסופר עד 10.

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;

entity count10 is
port (

    clk :in std_logic;

    rst :in std_logic;

    q :out std_logic_vector(3 downto 0));

end;
```

המשך הקוד בעמוד הבא <<

```

architecture arc_tff of count10 is
begin
    process(clk,rst)

        variable sq :std_logic_vector(3 downto 0);

    begin
        if rst='1' then
            sq:="0000";

        elsif clk'event and clk='0' then

            if sq<= 10 then
                sq:=sq +'1';

            else
                sq:="0000";

            end if;

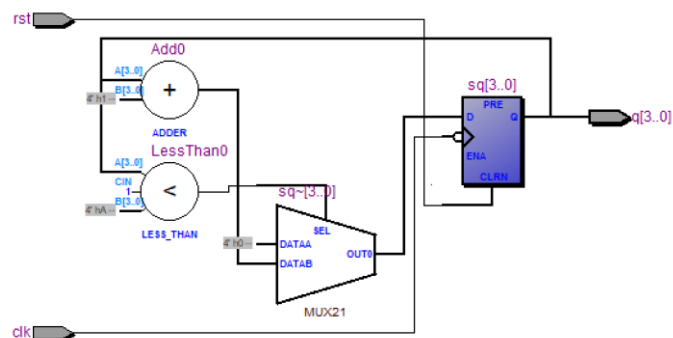
        end if;

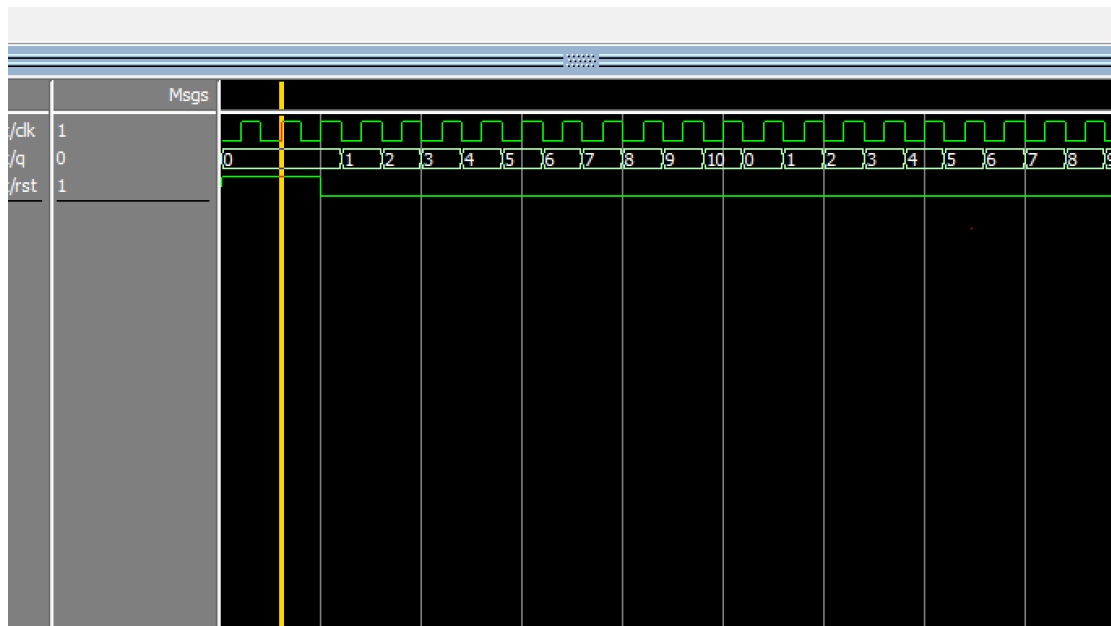
        q<=sq;

    end process;

end;

```





מלבד המסכם, שמגדיל את ערך היציאה של הנועל בכל ירידה של שעון, יש בתכנון הזה רכיב שתפקידו להשוות ל-10.

הרכב (multiplexer) שולט למעשה על המצב בין כניסה data_a לכניסה data_b.

הנועל (Latch) מגיב על הירידה של השעון. ערך הכניסה של הרבב הוא data_a=0, אז כשהמונה מגיע למספר 10 הרבב קורא את המצב בכניסה data_a ומעביר אותו אל הנועל.

איפוס שמתוזמן עם השעון נקרא איפוס סינכרוני, אולם איפוס מסוג reset הוא לא סינכרוני.

פרק 4: מבוא בסיסי לסימולציה

לאחר המבוא בפרק "הכרת כלי הסימולציה" נזכיר את התפקיד של הסימולציה, שהוא לשערך תוצאה של קוד במצב נתון. הסימולציה מחליפה כמובן את המציאות כמה שאפשר, ולכן יש להחשיב אותה למחוללת האירועים בקוד הנדרש.

תפקידה של הסימולציה הוא להזין נתונים בקוד המקורי כדי לבדוק את התפקודים שלו.

סימולציה היא מחולל (generator), ולכן אין לה כניסות או יציאות. בכל זאת צריך לתת לה שם, ואת ההגדרה של התכנון ב-vhdl יעשה האופרטור entity. תכנון הסימולציה יוגדר ללא כניסות ויציאות כמו תכנון רגיל, כפי שראינו במבוא לסינתזה.

```
Entity <simulation name> is
```

```
End;
```

דוגמה:

```
ENTITY count10_vhd_tst IS
```

```
END count10_vhd_tst
```

synthesize able and non-synthesize able

שפת ה-vhdl בנויה משני חלקים, וחשוב מאוד להבין אותם. אי-אפשר להפוך כל קוד לשערים לוגיים.

הקוד שאפשר להפוך לשערים לוגיים נקרא synthesize able, ניתן לסינתזה. זאת סימולציה באמצעות מחולל הבדיקה test bench – סביבה שמחוללת אותות ובודקת את הביצועים של האובייקט הנבדק.

סימולציה פועלת במחשב ולא על ה-FPGA – שתי הסביבות הללו שונות זו מזו במהות שלהן. דוגמה טובה שתבהיר לנו את הנושא היא השהיית זמן. במחשב יש מנגנון שמגדיר יחידת זמן בשפות (כגון שפת C), והוא נקרא delay.

ב-FPGA אפשר ליצור השהיית זמן רק באמצעות תכנון של מונה שסופר בתדר מסוים, כפי שנראה בהמשך הספר. ראוי לאמץ את הגישה שאומרת ששפת vhdl לצורכי סימולציה היא שפה שונה משפת vhdl לצורכי סינתזה. מכאן אפשר להבין שספריות שתומכות בסימולציה לא תומכות בסינתזה.

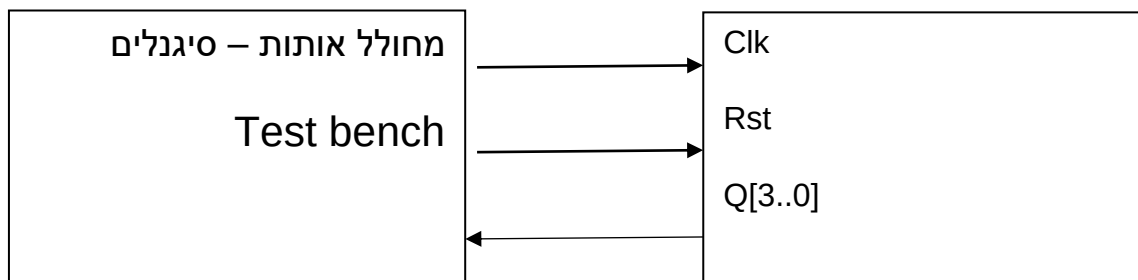
לעולם אל תערבבו בין השתיים.

עם זאת, שפת vhdل לצורכי סינתזה יכולה לפעול בסביבה של סימולציה.

לפי מה שנכתב כאן, ספריות לצורכי סימולציה יכללו ספריות לצורכי סינתזה ואפילו יותר מזה.

הגדרת משתנים – signal

מכיוון שאין לסימולציה כניסות ויציאות, יש לחולל אותות ולחבר אותם לאובייקט הנבדק.



לפנינו שתי משימות:

1. לחולל אותות ב-test bench אל הסיגנלים הנכונים.

2. לחבר אותם לאובייקט הנדרש.

אפשר לחולל אותות רק בעזרת משתנה מסוג signal, משום שיש לו ממד של זיכרון.

לפיכך נגדיר את כל הכניסות ואת כל היציאות של האובייקט בתור signal.

דוגמה:

```

SIGNAL clk : STD_LOGIC:='0';

SIGNAL q : STD_LOGIC_VECTOR(3 DOWNT0 0);

SIGNAL rst : STD_LOGIC;
  
```

בשלב הבא נגדיר את האובייקט הנבדק.

דבר זה נעשה בעזרת האופרטור component.

חשוב לזכור שבשעת השימוש באופרטור הזה צריך להעתיק את הכניסות ואת היציאות בדיוק כמו שעשינו בזמן התכנון.

דוגמה:

```

COMPONENT count10

  PORT (

    clk : IN STD_LOGIC;

    q : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);

    rst : IN STD_LOGIC

  );

END COMPONENT;

```

בעצם המילה entity התחלפה, ובמקומה באה המילה component.

השלב הבא:

Begin – תחילת קוד

למעשה קיימות כאן שתי סביבות:

1. סביבת ה-component, שעליה אנחנו מבצעים את הבדיקות.
2. הסביבה שמחוללת את האותות ב-test bench.

נדרש מאתנו לחבר בין שתי הסביבות. נוכל לעשות זאת עם האופרטור port map ובעזרת ציון שם ה-component.

```

uut : count10
PORT MAP (
  clk => clk,
  q => q,
  rst => rst
);

```

uut – המילה הזאת לא מציינת את היחידה הנבדקת ולא מורה עליה, אפשר לציין במקומה כל שם אחר. אולם לאחר הנקודתיים יש לציין את היחידה הנבדקת, במקרה הזה: count10.

Port map הוא החיבור בין ה-test bench ליחידה הנבדקת (count10). למעשה הוא מחווט בעזרת האופרטור => את החלקים של היחידה הנבדקת ל-signal של ה-test bench. הכלל

הוא, שכל החלקים ששייכים ל-component נמצאים משמאל לאופרטור => והחלקים ששייכים ל-test bench נמצאים מימין לאופרטור הזה.

בשלב הזה נחולל את האותות הרצויים.

יצירת אותות – מחוללים

אופרטורים שמשמשים ליצירת אותות ולא מומלצים לצורכי סינתזה הם פסוקי wait.

קיימים שלושה סוגים של פסוקי wait:

```
wait for<time expression>

wait on <signal name>

wait until <Boolean expression>
```

הפסוק wait for

קיימת חבילה (package) של הגדרות שמאפשרת לנו להגדיר פרקי זמן בתור חלק מהשפה. בחבילה הזאת אפשר להגדיר זמנים לפי הגדלים האלה: פיקו, ננו, מיקרו, מילי ושניות.

מבנה הפסוק:

```
Wait for <delay number> <unit time>
```

דוגמה:

```
Rst<='1','0', after 200 ns;
```

הערך rst נמצא במצב 1 ויורד למצב אפס לאחר 200 ns = מאתיים ננו-שניות.

יצירת אות ריבועי:

```
Clk<= not clk after 20 ns;
```

בכל עשרים ננו־שניות יתקיים היפוך במצב של הערך.

שרשרת של אותות:

```
X<='1','0' after 200 ns, '1' after 400 ns,'0' after 600 ns;
```

בדוגמה האחרונה נוצר אות בכל 200 ננו־שניות. במקרה הזה הזמן מחושב ממש מהרגע הראשון של ההתחלה.

```
X<='1','0' after 200 ns, '1' after 200 ns,'0' after 200 ns;
```

כתיבה כזאת היא לא חוקית, מפני שההפרש בין הזמנים של האותות נדרש כמובן להיות חיובי.

Process בתור מחולל אותות ובשילוב עם wait for

מעניין לראות כאן את ההבדל בין test bench לכתיבת קוד של synthesizable.

ל-process אין רשימה של רגישויות, ולא צריך לחשב שם את הפרשי הזמנים בין האותות – כמו באופרטור for.

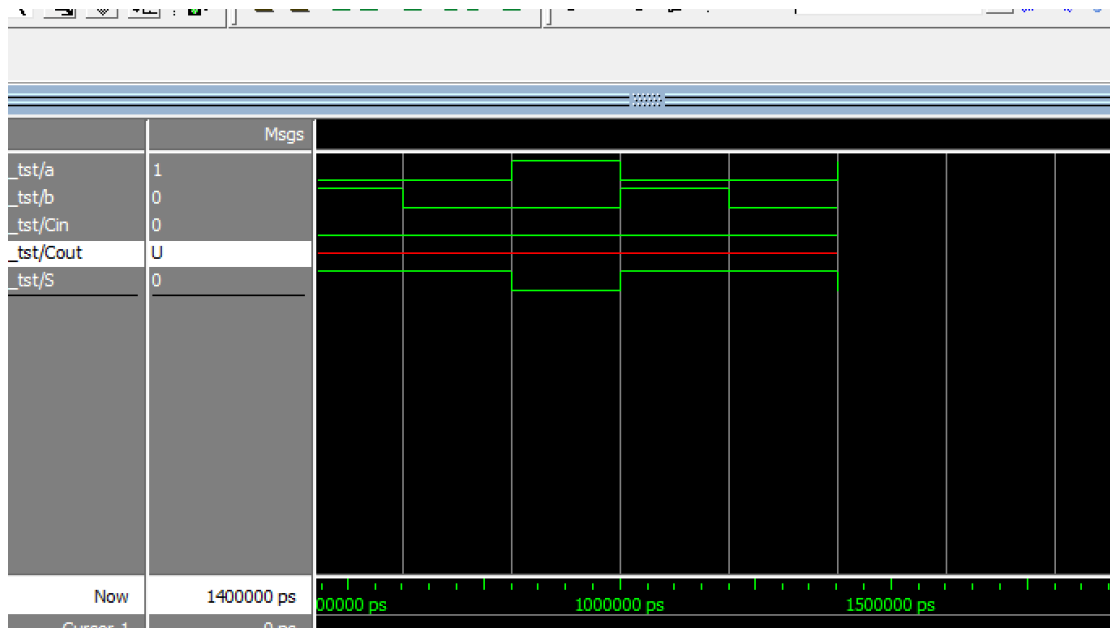
האופרטור שהתווסף כאן הוא wait.

האופרטור הזה עוצר את הסימולציה, ובלעדיו ה-process ימשיך לפעול בלי סוף באופן תאורטי.

למעשה לסביבה של סימולציה, כגון ה-modelsim, יש זמן הפעלה מוגבל.

דוגמה ממסכם מלא:

```
process  
  
begin  
  
a<='0';  
  
b<='0';  
  
cin<='0';  
  
wait for 200 ns;  
  
a<='1';  
  
b<='0';  
  
cin<='0';  
  
wait for 200 ns;  
  
a<='0';  
  
b<='1';  
  
cin<='0';  
  
wait for 200 ns;  
  
a<='1';  
  
b<='1';  
  
wait;  
  
end process;
```



בסימולציה אפשר לראות את התגובות של המסכם המלא. הלוגיקה של Cout לא מוגדרת, ועל כן הוא נראה בצבע אדום.

האופרטור wait until

האופרטור הזה ימתין לאירוע שהתשובה עליו היא בוליאנית.

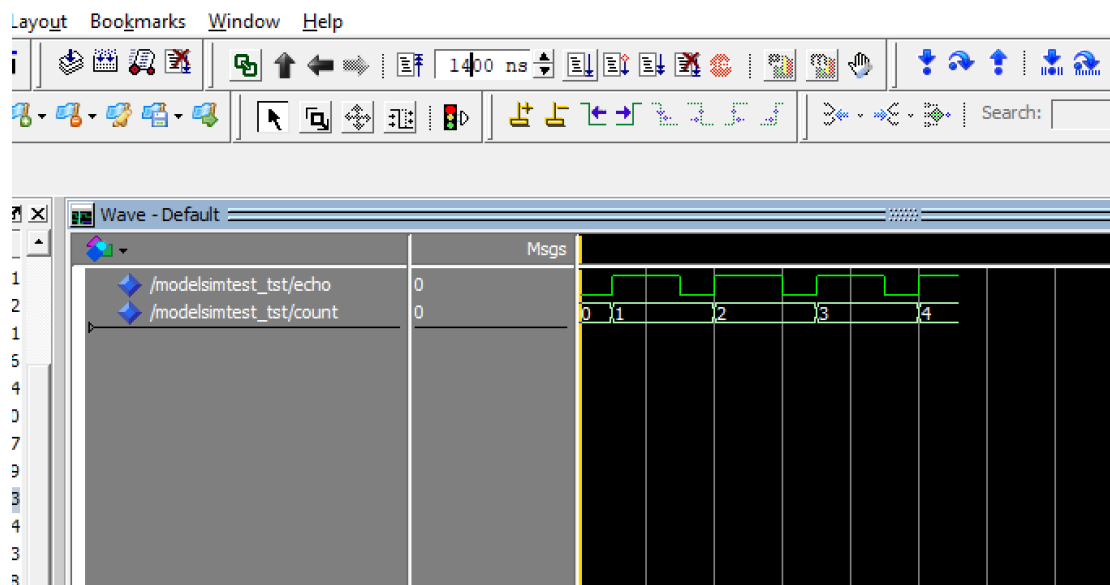
המבנה שלו הוא:

```
wait until<Boolean expression>
```

דוגמה:

```
wait until x=1;
```

בדוגמה הבאה נוצר גל ריבועי בעזרת wait for ומזומנת התגובה אליו באמצעות wait until. הפקודה wait until מתקיימת בתהליך של process.



```
USE ieee.std_logic_arith.all;

USE ieee.std_logic_unsigned.all;

ENTITY modelsimTest_tst IS

END;

ARCHITECTURE count10_arch OF modelsimTest_tst IS

-- constants

-- signals

SIGNAL echo : STD_LOGIC:='0';

signal count :integer range 1023 downto 0:=0;


BEGIN


process

begin

echo<='0';

wait for 250 ns;

echo<='1';

wait for 500 ns;

echo<='0';

end process;

process

begin

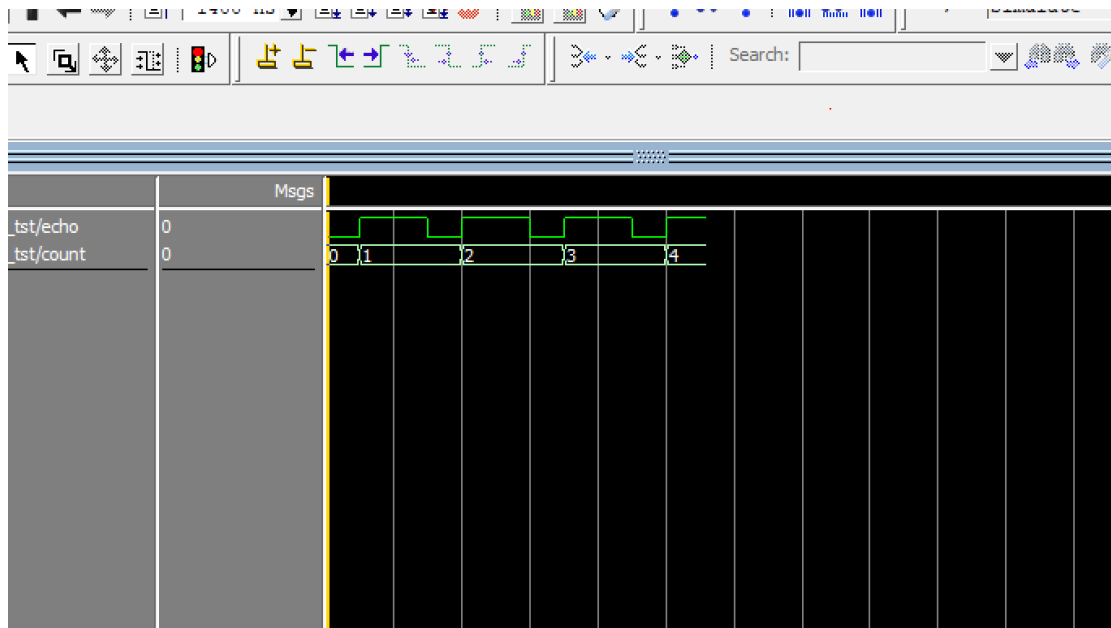
wait until echo='1';

count<=count+1;

end process;

END;
```

ממתין לאירוע ב-echo



איך צריך לקרוא את הסימולציה?

התהליך ממתין ל-echo=1, ולכן ה-counter מתקדם ב-1. מכאן תהליך ה-process ימתין עד לשחרור ה-echo ממעבד של 1 ואז ימשיך את הפעולה שלו.

התוצאה שתקבל בסימולציה בצורה של מניית אירועים.

ה-process הראשון ישמש בתור מחולל, והשני – בתור תגובה.

האופרטור wait on

זה אופרטור שימתין לכל שינוי ללא הבחנה בין 0, 1, עלייה או ירידה.

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_arith.all;

USE ieee.std_logic_unsigned.all;

ENTITY modelsimTest_tst IS

END;

ARCHITECTURE count10_arch OF modelsimTest_tst IS

SIGNAL echo : STD_LOGIC:='0';

signal count :integer range 1023 downto 0:=0;

BEGIN

process

begin

echo<='0';

wait for 250 ns;

echo<='1';

wait for 500 ns;

echo<='0';

end process;


process

begin

wait on echo;

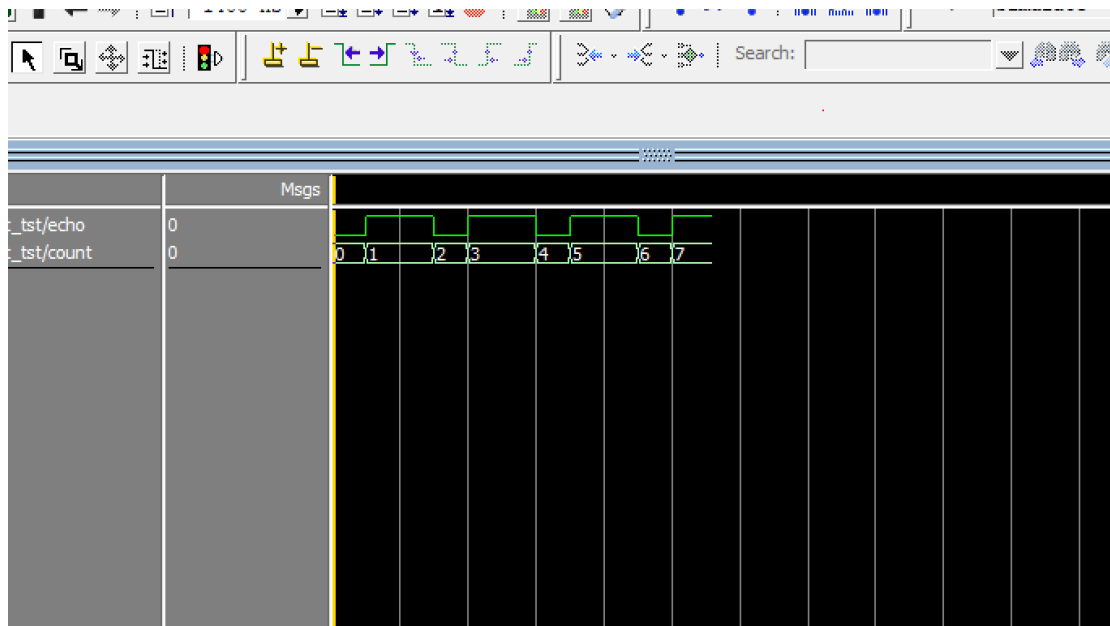
count<=count+1;

end process;

END;

```

ממתין לאירוע ב-echo



שתי הסימולציות מסבירות את ההבדל בין wait until ובין wait on.

הפקודה wait on מונה כל שינוי שמתרחש, אבל הפקודה wait until מחכה במקרה הזה לאירוע שנדרש ממנה להמתין אליו.

בכל אחת מהפקודות האלה צריך לקרות שינוי של אירוע מסוים כדי שהן יוכלו להשתחרר ממצב של המתנה.

לא מומלץ להשתמש בפקודת wait until לצורכי סינתזה, אך אפשר להסביר בעזרתה איך נזהה עלייה או ירידה של שעון.

זיהוי עלייה או ירידה של שעון באמצעות wait until

אפשר לזהות עליית שעון או ירידת שעון אם מצרפים את הפקודה clock'event עם האופרטור גרש '<'.>.

ראו את הקוד הבא:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_arith.all;

USE ieee.std_logic_unsigned.all;

ENTITY modelsimTest_tst IS

END;

ARCHITECTURE count10_arch OF modelsimTest_tst IS

-- constants

-- signals

SIGNAL echo : STD_LOGIC:='0';

signal count :integer range 1023 downto 0:=0;


BEGIN

process

begin

echo<='0';

wait for 250 ns;

echo<='1';

wait for 500 ns;

echo<='0';

end process;


process

begin
```

לא מומלץ למתכנתים מתחילים ב-vhdl להשתמש בקוד כזה לצורכי סינתזה, אבל זה בכל זאת אפשרי. הערה זו נוגעת כמובן לתהליך הזה ולא לתהליך הקודם שמחולל את האותות, שם אכן ברור שזה לא לסינתזה כי אין השהיות של for לצורכי סינתזה.

```
wait until echo'event and echo='1';

count<=count+1;

end process;

END;
```

רשימת הרגישויות של התהליך

כפי שכבר הוסבר, לתהליך הזה חייבת להיות רשימה של רגישויות.

אפשר להחליף את רשימת הרגישויות הזאת באמצעות שימוש ב-wait on.

נעסוק בקוד שמתאר srff:

```
library ieee;

use ieee.std_logic_1164.all;

entity srffWaitOn is
port (
s,r :in std_logic;
q :out std_logic
);
end;

architecture arc_srff of srffWaitOn is
signal sq: std_logic;

begin

srffq:process is

begin
if s='0' and r='0' then
sq<=sq;
elsif s='0' and r='1' then
sq<='0';
elsif s='1' and r='0' then
sq<='1';
else
sq<='0';
end if;
```

המשך הקוד בעמוד הבא <<

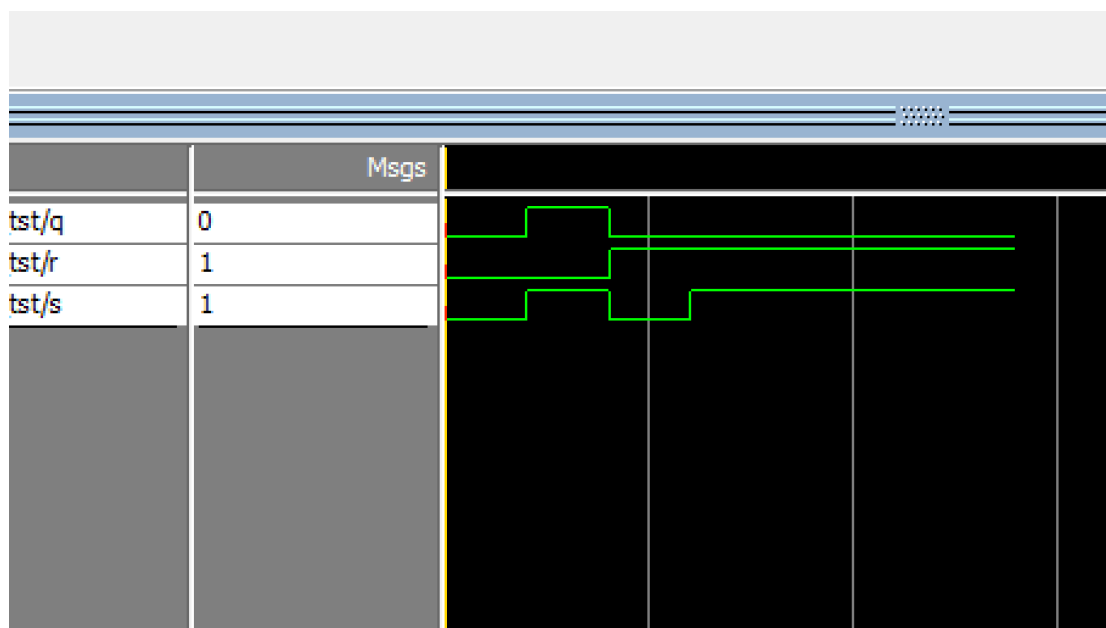
תחליף לרשימה של רגישויות באמצעות שימוש ב-`wait on`.

```
wait on s,r;

end process;

q<=sq;

end;
```



והנה לפניכם התוצאה הצפויה. שוב, זה לא מומלץ לצורכי סינתזה.

מכונת מצבים סופית

קיימים שני סוגים של מערכות דיגיטליות. הסוג הראשון הוא מערכת צירופית (combinatorial logic), מערכת שהיציאה שלה תלויה אך ורק בכניסות. כבר פגשנו מערכות כאלה: מסכם מלא, מפענחים, רבבים.

הסוג השני הוא מערכת סדרתית (sequential logic) – מערכת שהיציאה שלה תלויה בכניסה אבל גם במשתנה פנימי או כמו שנראה כאן, במצב העכשווי שבו היא נמצאת (Current state). במערכות כאלה אפשר למצוא מגוון רחב: ממונים פשוטים ועד למיקרו-פרוססורים.

במקרה הזה נעסוק במערכות סדרתיות שבהן מספר המצבים הוא סופי, מצב זה נקרא finite state machine ובקיצור F.S.M. האתגר הוא להראות כיצד שפת ה-vhdl תומכת בתפיסת התכנות הזאת.

למערכות סדרתיות מהסוג F.S.M. יש כמה מצבים של כניסות ויציאות, בתוספת חוקים אחדים שמאפשרים להן לעבור ממצב אחד למצב אחר.

מקובל שכדי לאפיין מערכת דיגיטלית מתחילים בהערכת מספר המצבים הסופי. זה מאפשר למתכנת לחשוב בכלליות על המערכת, בלי להיכנס יותר מדי לפרטים על האופן שבו צריך לממש זאת. כפי שנראה בהמשך, נצטרך לחשוב ככה בדוגמת החניון או במערכת הפסיקות.

עיצוב התוכנה או אפיון מדויק יותר של הדרישות יאפשרו לנו מעבר מידי לקוד של vhd.

בתהליך הבא נתאר את הפרויקט או את הדרישות של התכנון ברמת black box. טווח הסקירה של החשיבה הזאת יהיה מהרמה הכללית וכלפי מטה עד לרמת הביצוע.

(Top to down view design)

אחרי כן נתאר את המערכת בתור bubbles chart – הדיאגרמה הגרפית המקובלת כדי לתאר מכונת מצבים.

נדרש מאתנו לתכנן מונה לספירת המכונות שנכנסות לחניון ציבורי, לצורך הפשטות לא נתחשב במכונות שיוצאות ממנו.

המבנה הזה מציג תיאור ציורי של המערכת:

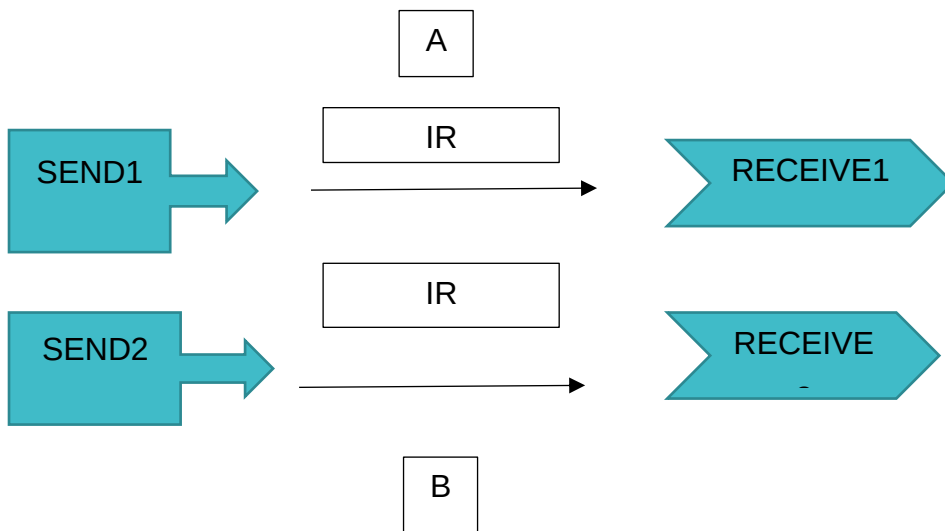
נתונים שני חיישנים, וכל אחד מהם שולח קרן – לצורך העניין: IR, send1 ו-send2.

שני מקלטים (receive 1 ו-receive 2) קולטים את הקרניים מהחיישנים.

נניח עוד, שמכונית מגיעה ממקום A למקום B.

בשעת המעבר ממקום A למקום B המכונית הזאת חותכת את קרני ה-IR.

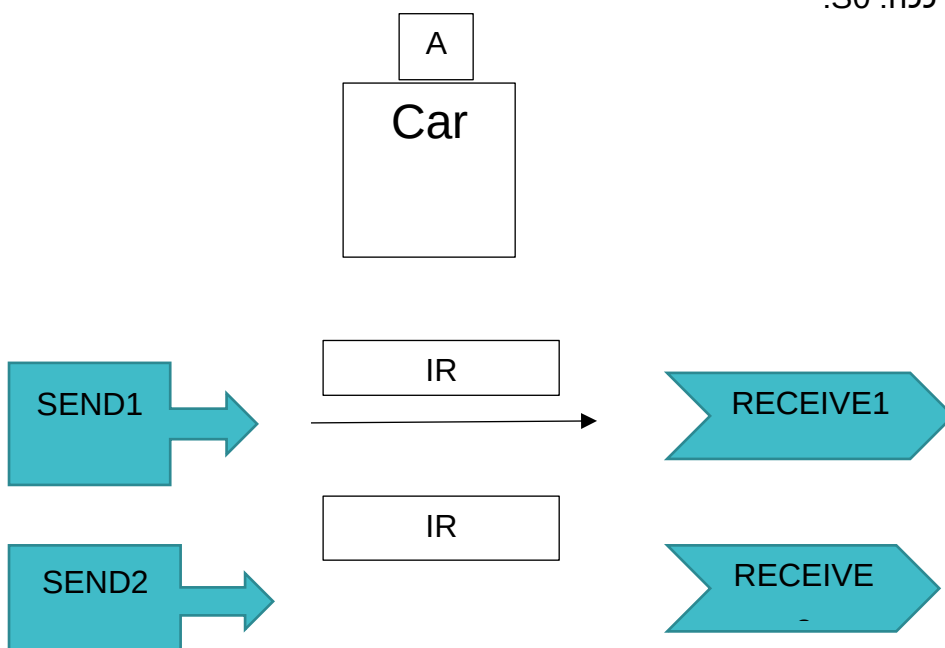
ההנחה האחרונה שלנו היא: קרן שפוגעת ב-receiver מתורגמת לרמה לוגית של 1, ואם קורה ההפך – לרמה לוגית של 0.

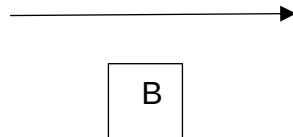


המכונית לפני שהיא חוצה את הקרניים IR.

הרמות הלוגיות R1, R2 מתואמות עם המקלטים receive1, receive2.

R1=1 ; R2=0 – נסמן זאת ככה: S0.

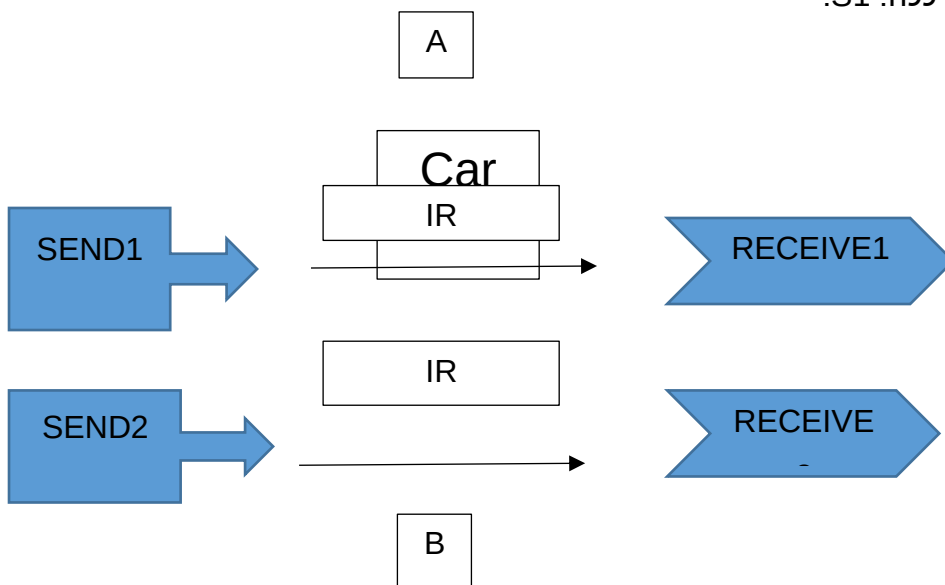




המכונית חוצה את קרן A.

הרמות הלוגיות R1, R2 מתואמות עם המקלטים receive1, receive2.

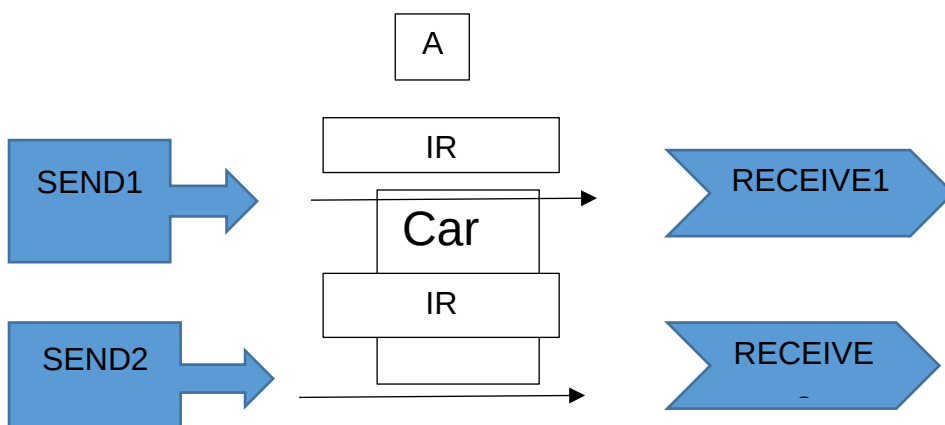
S1: נסמן זאת ככה: $R1=1$; $R2=0$



המכונית חוצה את קרן A.

הרמות הלוגיות R1, R2 מתואמות עם המקלטים receive1, receive2.

S2: נסמן זאת ככה: $R2=1$; $R1=1$

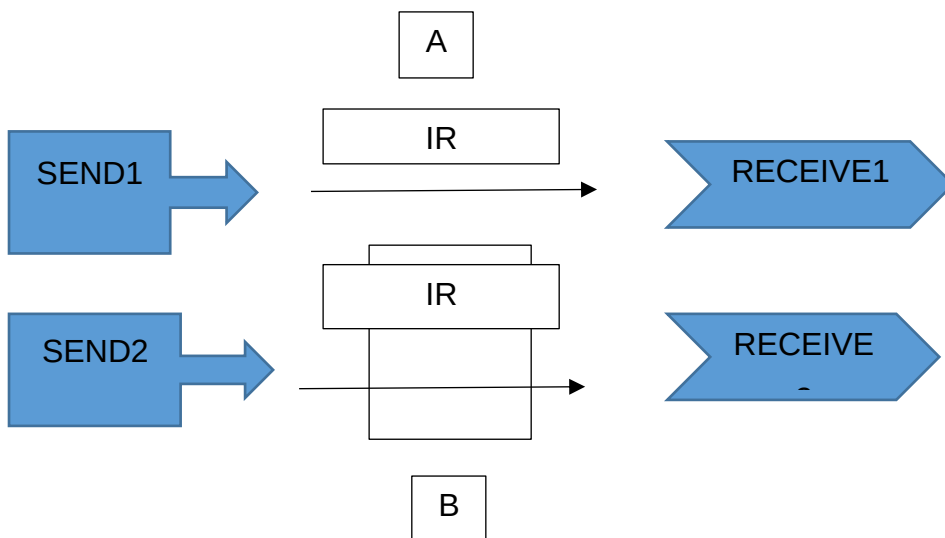


B

המכונת חוצה את קרן A.

הרמות הלוגיות R1, R2 מתואמות עם המקלטים receive1, receive2.

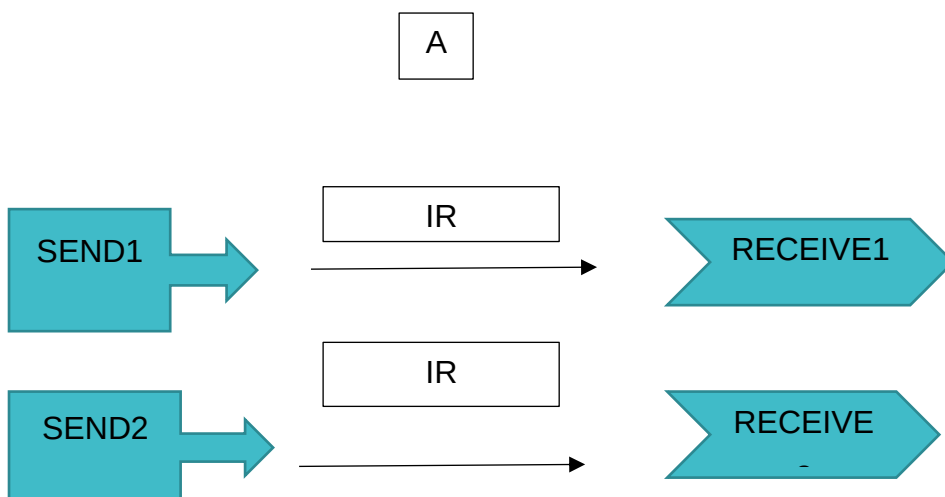
R1=0 ; R2=1 – נסמן זאת ככה: S3.

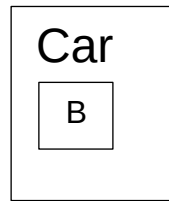


המכונת חוצה את קרן A.

הרמות הלוגיות R1, R2 מתואמות עם המקלטים receive1, receive2.

R1=0 ; R2=0 – נסמן מצב זה בתור read.



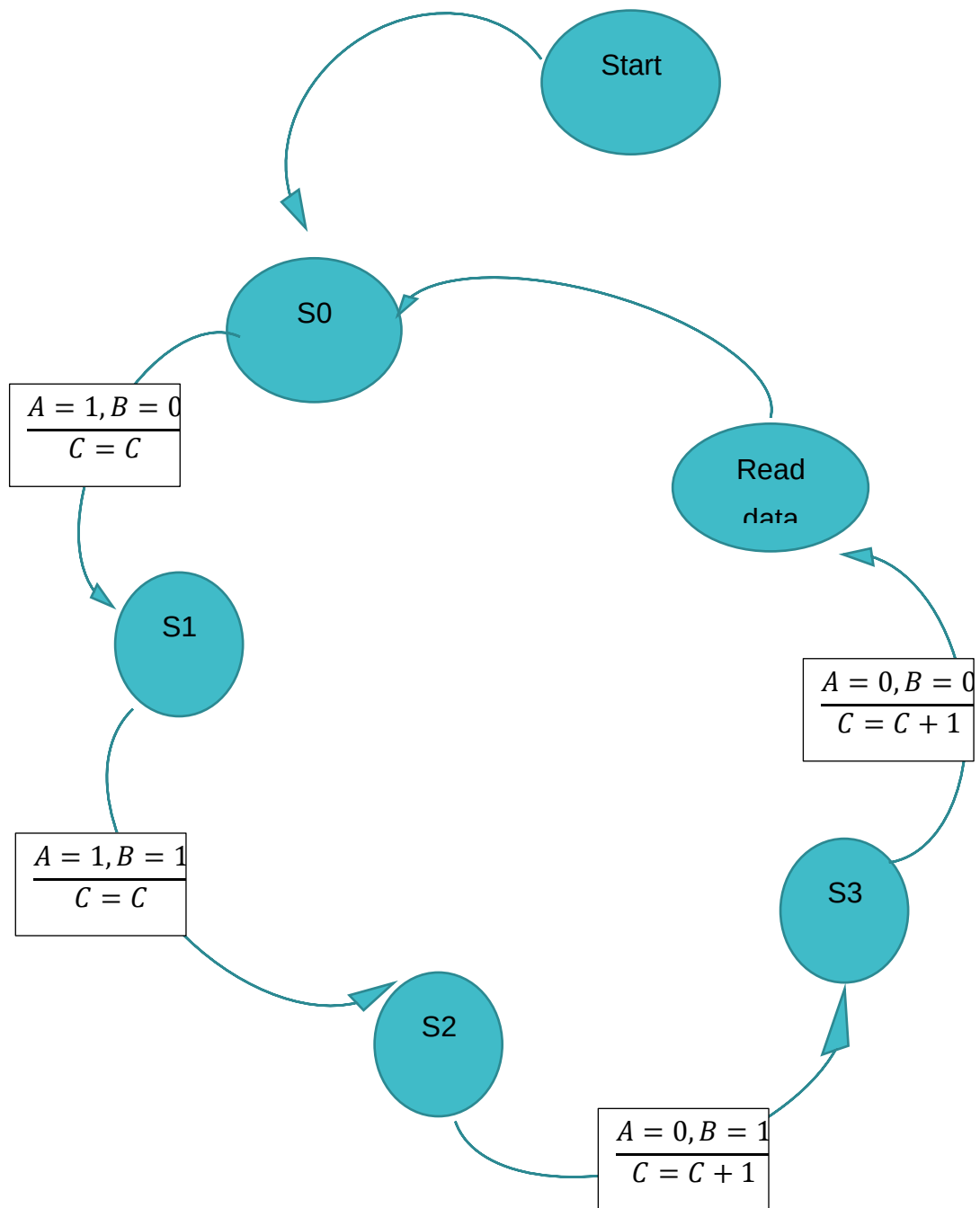


נהוג ונוח מאוד לתאר את השתלשלות הדברים בעזרת bubbles chart – דיאגרמת בועות. לא נהוג לתאר זאת בתרשים זרימה עם כל מיני צורות מרובעות, כפי שנעשה לעיל.

בדיאגרמת הבועות (bubbles chart) נהוג לסמן בעזרת קוֹשֶׁבֶר את התנאים: במונה מצוין

המצב בכניסות, ובמכנה – התוצאה של תנאי הכניסה הרגעית $\frac{\text{מצב הכניסות}}{\text{תוצאה}}$.

Bubbles Chart – דיאגרמת בועות



נוח מאוד להשתמש בדיאגרמת הבועות בשביל תיאור ויזואלי, אולם מובן ששיטה זו די מוגבלת אם לפנינו מקרה עם מספר רב ביותר של מצבים. את זה דווקא כדאי לתאר עם טבלה. בטבלה

תתואר מכונת המצבים כך: current הוא המצב העכשווי שמכונת המצבים נמצאת בו, next הוא המצב שהמכונה תימצא בו בשלב הבא.

Current	next	output
start	S0	c=c
S0	S1	C=c
S1	S2	C=c
S2	S3	C=c+1
S3	read	C=c
read	start	C=c

Type

לפני שנמשיך, נכיר את האופרטור type.

אפשר להגדיר קבוצת של עצמים בתור קבוצה אחת בעזרת האופרטור הזה.

Type <names of group> is (list of group);

בקשר לדוגמה שלנו זה ייראה כך:

Type state_vector is(start,s0,s1,s2,s3,read);

State_vector הוא שם הקבוצה.

האופרטור Type הוא לא סוג של משתנה שאפשר ליישם כמו signal או variable, לכן צריך להפוך את state_vector ל-signal.

מבנה ההמרה הוא:

Signal <name of signal> <:> <name of type>

ובקשר לנושא שלנו:

Signal state:state_vector;

ה-signal state יכול עכשיו לקבל כל ערך מקבוצת ה-state_vector.

דוגמה:

State <=start;

Start <=s0;

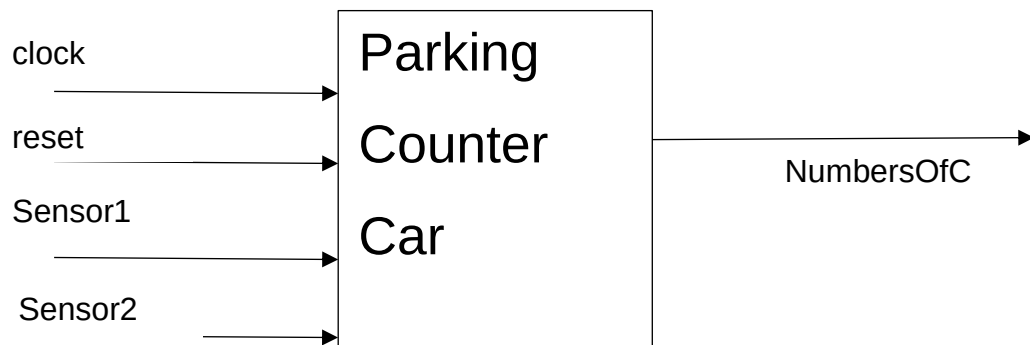
וכן הלאה.

תיאור הפרויקט "קופסה שחורה"

בשלב הראשון נאפיין את הכניסות ואת היציאות של הפרויקט.

המערכת תהיה סינכרונית, נציג את הסיבות לזה מאוחר יותר.

כזכור מערכת סינכרונית היא מערכת שמתואמת לשעון.



הקוד על-פי התרשים:

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_arith.all;
```

```
use ieee.std_logic_unsigned.all;
```

```
entity ParkingCounterCar is
```

```
port (
```

```
        clock :in std_logic;

        reset :in std_logic;

        sensor1 :in std_logic;

        sensor2 :in std_logic;

        NumbersOfCar :out std_logic_vector(3 downto 0)

    );

end;

architecture arc_ParkingCounterCar of ParkingCounterCar is

    type state_vector is(start,s0,s1,s2,s3,readData);

    signal state:state_vector;

    signal count :std_logic_vector(3 downto 0);

begin

    process(clock,reset)

    begin

        if reset='1' then

            count<=(others=>'0');

            state<=start;

        elsif clock'event and clock='1' then

            case state is

                when start =>

                    state<=s0;

                    when s0 =>

                        count<=count;
```

```
if (sensor1='1') and (sensor2 ='0') then
```

```
    state<=s1;
```

```
else
```

```
    state<=s0;
```

```
end if;
```

```
when s1 =>
```

```
    count<=count;
```

```
    if sensor1='1' and sensor2 ='1' then
```

```
        state<=s2;
```

```
    else
```

```
        state<=s1;
```

```
    end if;
```

```
when s2 =>
```

```
    count<=count;
```

```
    if sensor1 = '0' and sensor2 = '1' then
```

```
        state<=s3;
```

```
    else
```

```
        state<=s2;
```

```
    end if;
```

```
when s3 =>
```

```

        count<=count+'1';

    if sensor1='0' and sensor2 ='0' then

        state<=readData;

    else

        state<=s3;

    end if;

when readData =>

    count<=count;

    state<=s0;

when others=>

    state<=s0;

end case;

end if;

end process;

NumbersOfCar<=count;

end;
```

הערות אחדות בקשר לקוד ולדיאגרמת הבועות:

1.

```

type state_vector is(start,s0,s1,s2,s3,readData);

signal state:state_vector;
```

השורות האלה מגדירות את הקבוצה שנקראת state_vector.

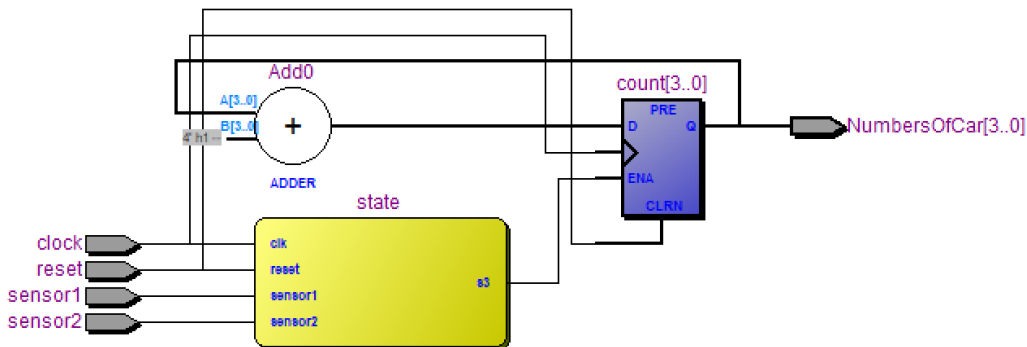
ה-signal state מאפשר לנו לגשת אל הקבוצה הזאת, כי המשתנה ב-vhdl הוא מסוג signal או מסוג variable.

2. במצב של reset מוגדר המצב ההתחלתי של מכונת המצבים.

מצב התחלתי של מערכת הוא חשוב ביותר, מפני שלתהליך החיבור של המתח למערכת לוגית תמיד מתלווה אקראיות. היא יכולה להופיע באחת מהאפשרויות הניתנות, ואפילו גרוע מזה – באחת מהאפשרויות שעדיין לא הוגדרו. חייבים לצמצם מצבים בלתי מוגדרים באמצעות הפסוק $\text{when others} \Rightarrow$, וחשוב מאוד תמיד לסיים את האפשרויות הבלתי מוגדרות באופן שכל המצבים יהיו בסוף מוגדרים.

3. מספר המצבים במכונת המצבים הוא 6. בסופו של דבר כל האפשרויות מתורגמות למספרים בינאריים – התהליך הזה נקרא קידוד. יש כל מיני סוגים של קידוד. כדי לקודד בעבור 6 אפשרויות נדרשות 3 סיביות לפחות. ל-3 סיביות יש 8 אפשרויות. במכונת המצבים מוגדרות רק 6 אפשרויות, ועל כן את 2 האפשרויות שנשארו לנו נגדיר בעצמנו באמצעות when others .

התרשים הלוגי שנוצר לפי הקוד:



המלבן הצהוב מתאר מערכת לוגית סינכרונית, ולעומתו המחבר add הוא מערכת לא סינכרונית.

היציאה s3 מאפשרת היווצרות של מעגל דלגלג (flip-flop circuit). משום שהמחבר מובנה בתהליך עם רשימת רגישויות של clock ו-reset, לפיכך הוא מערכת קומבינטורית שמסתיימת במעגל דלגלג ונועלת את התוצאה העכשווית עד לעדכון הבא.

אפשר לכתוב את הקוד הזה גם עם משתנה מסוג variable.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity ParkingCounterCarVar is
port (
    clock :in std_logic;
    reset :in std_logic;
    sensor1 :in std_logic;
    sensor2 :in std_logic;
    CurrenStateVector :out integer range 15 downto 0;
    NumbersOfCars :out std_logic_vector(3 downto 0)
);
end;

architecture Arc_ParkingCounterCarVar of ParkingCounterCarVar is
begin
    process(clock,reset)
    type state_vector is(start,s0,s1,s2,s3,readData);
    variable state:state_vector;
    variable count :std_logic_vector(3 downto 0):="0000";
    begin
        if reset='1' then
            count:=(others=>'0');
            state:=start;
        elsif clock'event and clock='1' then
```

case state is

when start =>

state:=s0;

CurrenStateVector<= 0;

when s0 =>

CurrenStateVector<= 1;

count:=count;

if (sensor1='1') and (sensor2 ='0') then

state:=s1;

else

state:=s0;

end if;

when s1 =>

CurrenStateVector<= 2;

count:=count;

if sensor1='1' and sensor2 ='1' then

state:=s2;

else

state:=s1;

end if;

when s2 =>

CurrenStateVector<= 3;

count:=count;

if sensor1 = '0' and sensor2 = '1' then

state:=s3;

else

state:=s2;

end if;

when s3 =>

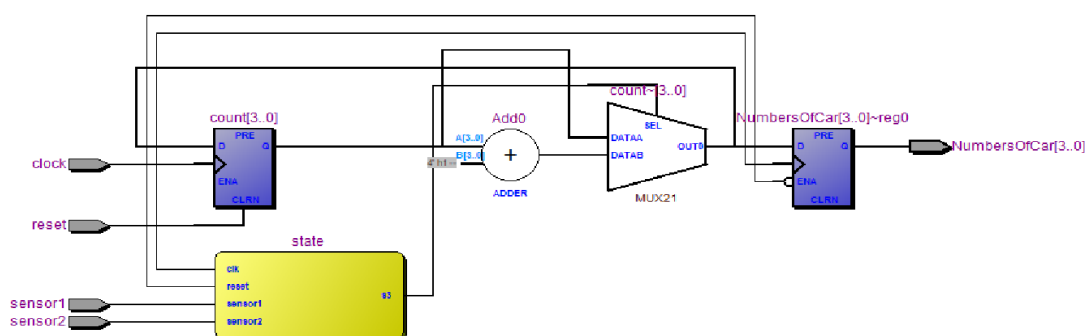
```

CurrentStateVector<= 4;

    count:=count+'1';
    if sensor1='0' and sensor2 ='0' then
        state:=readData;
    else
        state:=s3;
    end if;
when readData =>
    CurrentStateVector<= 5;
    count:=count;
    state:=s0;
when others=>
    CurrentStateVector<= 6;
    state:=s0;
end case;
NumbersOfCars<=count;
end if;
end process;

end;

```



כאן יש "עטיפה" עם מעגלי דלגלג של הרפב ושל המחבר (connector), שהם בעצם מערכות קומבינטוריות.

בשורות הבאות נמצאים ההבדלים בין כתיבה עם variable לכתיבה עם signal.
ב-variable הדברים מוגדרים בתוך התהליך:

```
process(clock,reset)
type state_vector is(start,s0,s1,s2,s3,readData);
variable state:state_vector;
variable count :std_logic_vector(3 downto 0);
```

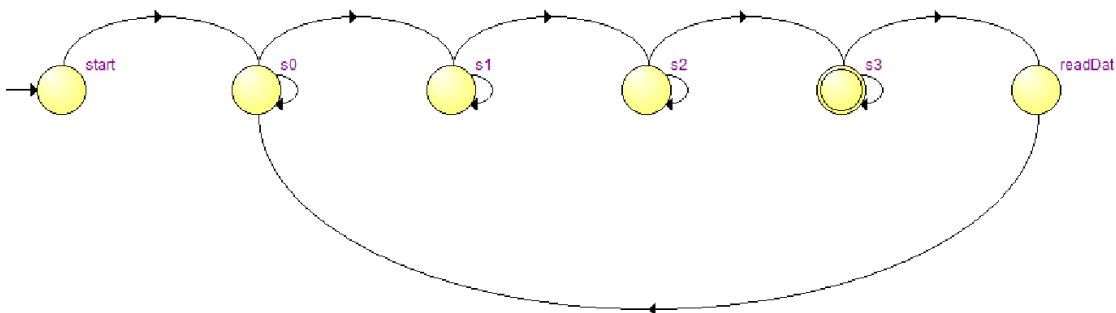
ולעומת זאת כך הם פני הדברים ב-signal:

```
architecture arc_ParkingCounterCar of ParkingCounterCar is
```

```
type state_vector is(start,s0,s1,s2,s3,readData);
signal state:state_vector;
```

```
signal count :std_logic_vector(3 downto 0);
```

בתרשים הבא אפשר לראות את דיאגרמת הבועות שנוצרה על-פי הקוד המתורגם.



לכל בועה, חוץ מהראשונה ומהאחרונה, יש שני מצבים.

שלא כמו בשני המצבים האלה, בקוד מתוארים התנאים שנחוצים לנו כדי לעבור למצב האחר או כדי להישאר במצב הנוכחי. ראו את החצים, הם מורים על מעבר או על הישארות באותו המצב.

שימו לב שאפשר להגיע למצב ההתחלתי רק בעזרת reset. ניתן לראות זאת בקוד, אבל לפי דיאגרמת הבועות הלולאה לא כוללת אותו.

הכוח של מכונת מצבים הוא שהיא יודעת לברור די בקלות בין מצבים זהים. שימו לב למשל להשוואה בין start ובין s3 – יש להם תנאי מעבר זהים.

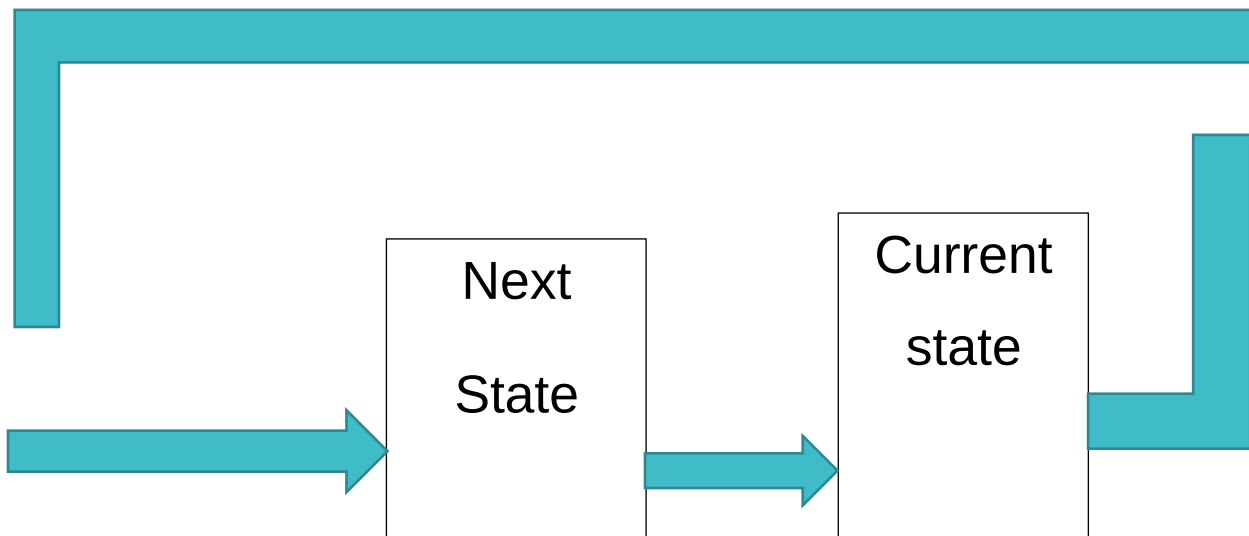
מכונות מצבים נוספות

החיסרון של מכונת מצבים הוא משאבים רבים. ציבור הקוראים מתבקש שלא להקל ראש בעניין הזה בימים שמשאבי חומרה הם זולים ועם זאת צפופים כל-כך. הרבה משאבים מאטים את הביצועים שלהם וגם צורכים הספק גבוה.

כולנו רוצים שהמכשירים שלנו יזללו אנרגיה מאיכות נמוכה ויהיו כמובן מהירים יותר. אפילו שהספר הזה לא דן בעקרונות של תוכן לוגי, אופן המימוש שלו חשוב מאוד. על הקורא לדעת שתוכן לוגי הוא הבסיס לתכנון עם אתגרים של real time ולחיסכון באנרגיה.

מהי מכונת מצבים?

במערכת שבה השלב הבא נקבע לפי השלב הנוכחי, מובן שנדרש משוב. מכאן נסיק שמכונת מצבים היא צירופית, כלומר קומבינטורית. המבנה הבסיסי של מכונת מצבים הוא משוב.



אם שתי המערכות הן צירופיות, זמני ההשהיה של כל אחת מהן הם בלתי צפויים. התוצאה תהיה מערכת ללא שליטה. אבל אם ה-current state יהיה מצב סינכרוני, המערכת תגיב על-

פי הקצב של המערכת הסינכרונית. במערכת לא סינכרונית זמני ההשהיה נקבעים לפי השרשרת של הרכיבים מהכניסה למוצא. אם יש כמה מסלולים בעת ובעונה אחת, זמן התגובה של המערכת ייקבע לפי המסלול העכשווי.

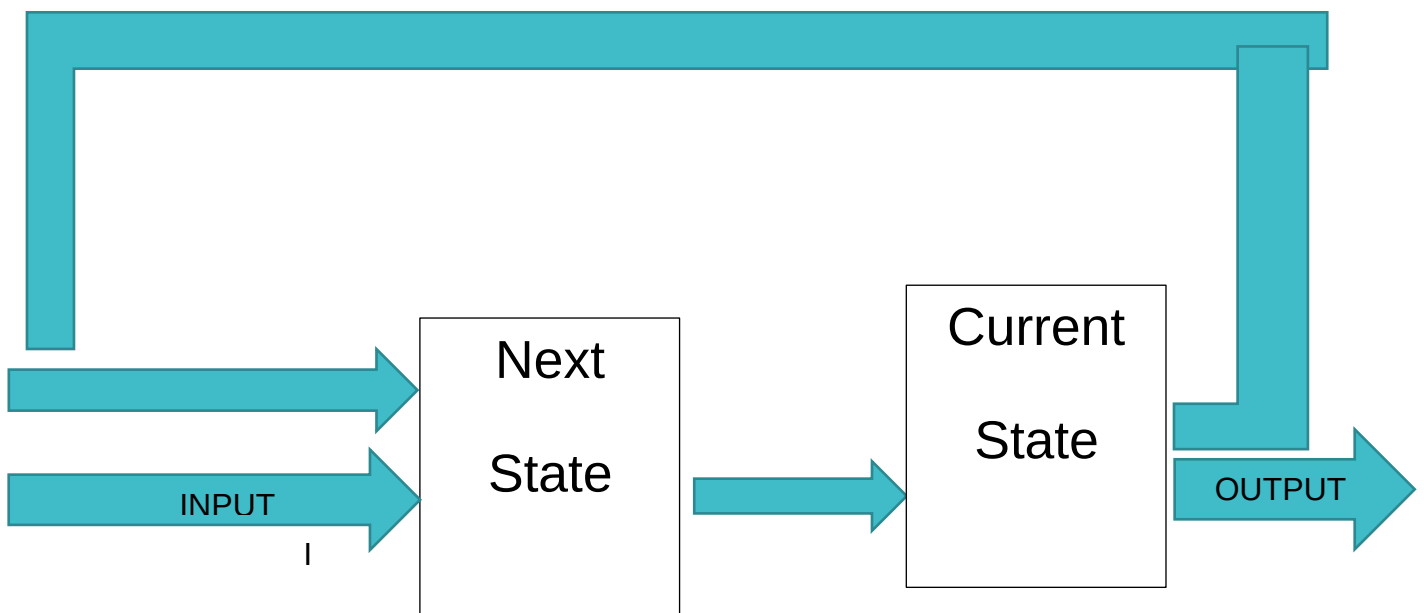
התנאים של בחירת המסלול לא תמיד ידועים מראש, ולכן אי-אפשר לצפות מראש את זמן התגובה.

מערכת קומבינטורית מהירה ללא זמן תגובה ידוע.

שילוב מערכת צירופית עם מערכת סינכרונית.

Full Moore State Machine

מערכת מסוג כזה נקראת full moore state machine. היציאה של המערכת נקבעת אך ורק על-פי המצב העכשווי שלה. next state הוא מצב צירופי, ואילו ה-current state הוא מצב סינכרוני. במצב כזה המערכת הסינכרונית קובעת את קצב העבודה של המערכת, זאת אומרת את התדר שלה. זמן התגובה של המערכת הצירופית מחושב תמיד לפי שרשרת הרכיבים הארוכה שבתוך המערכת, ועל-פי זה נקבע תדר המערכת. לפיכך לא ייתכן שהתדר של המערכת יהיה מהיר יותר מזמן התגובה הלא סינכרוני.

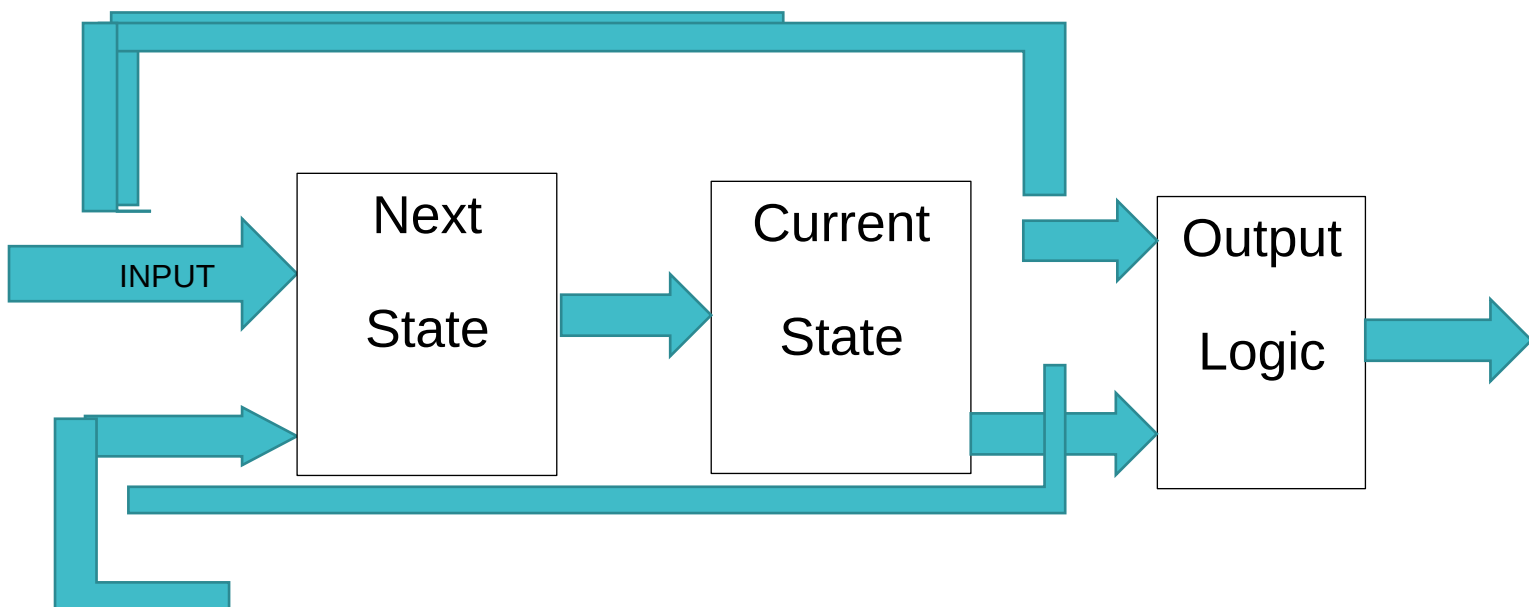


לפי הנאמר לעיל, התוצאה היא מערכת קומבינטורית מסוג next state שמשולבת עם מערכת סינכרונית. המצב הבא במערכת זו תלוי בכניסה (input) וב-current state גם יחד.

היציאות במערכות כאלה תלויות ב-next state, אבל לא תמיד זה באמת הצורך. נניח שהמערכת הסינכרונית היא מונה סינכרוני, ושצריך להראות את המספרים על שבעה מקטעים (seven segment) – אז נזדקק למעגל לוגי נוסף, שייקרא output.

נציג את המעגל הזה במערכת הבאה.

Mealy State Machine



```
LIBRARY ieee;

USE ieee.std_logic_1164.all;


ENTITY ParkingCounterCarVar_vhd_tst IS

END ParkingCounterCarVar_vhd_tst;

ARCHITECTURE ParkingCounterCarVar_arch OF
ParkingCounterCarVar_vhd_tst IS

    SIGNAL clock : STD_LOGIC:= '0';

    SIGNAL NumbersOfCar : STD_LOGIC_VECTOR(3 DOWNTO 0);

    SIGNAL reset : STD_LOGIC;

    SIGNAL sensor1 : STD_LOGIC;

    SIGNAL sensor2 : STD_LOGIC;

    SIGNAL CurrenStateVector : integer range 15 downto 0;

    COMPONENT ParkingCounterCarVar

        PORT (

            clock : IN STD_LOGIC;

            NumbersOfCar : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);

            CurrenStateVector :out integer range 15 downto 0;

            reset : IN STD_LOGIC;

            sensor1 : IN STD_LOGIC;

            sensor2 : IN STD_LOGIC

        );

    END COMPONENT;

BEGIN
```

```
i1 : ParkingCounterCarVar
```

```
PORT MAP (
```

```
-- list connections between master ports and signals
```

```
clock => clock,
```

```
NumbersOfCar => NumbersOfCar,
```

```
CurrenStateVector => CurrenStateVector,
```

```
reset => reset,
```

```
sensor1 => sensor1,
```

```
sensor2 => sensor2
```

```
);
```

יצירת גל ריבועי של השעון בתדר כזה: 50Mz.

```
clock<=not clock after 20 ns;
```

יצירת גל מדרגה של reset לפרק זמן של 100ns.

```
reset<='1','0' after 100 ns;
```

יצירת גלים שמותאמים לשלבי הכניסה. בכל שלב צריך לדמות את החיישן לפי המצב שבו המכונת נמצאת בזמן הכניסה לחניון.

כל ערך של 1 בחיישנים sensor1 או sensor2 מציין שהמכונת קטעה את קרן הכניסה.

```
process
```

```
begin
```

```
sensor1<='0';
```

```
sensor2<='0';
```

```
wait for 300 ns;
```

```
sensor1<='1';
```

```
sensor2<='0';
```

```
wait for 300 ns;
```

```
sensor1<='1';
```

```
sensor2<='1';
```

```
wait for 300 ns;
```

```
sensor1<='0';
```

```
sensor2<='1';
```

```
wait for 300 ns;
```

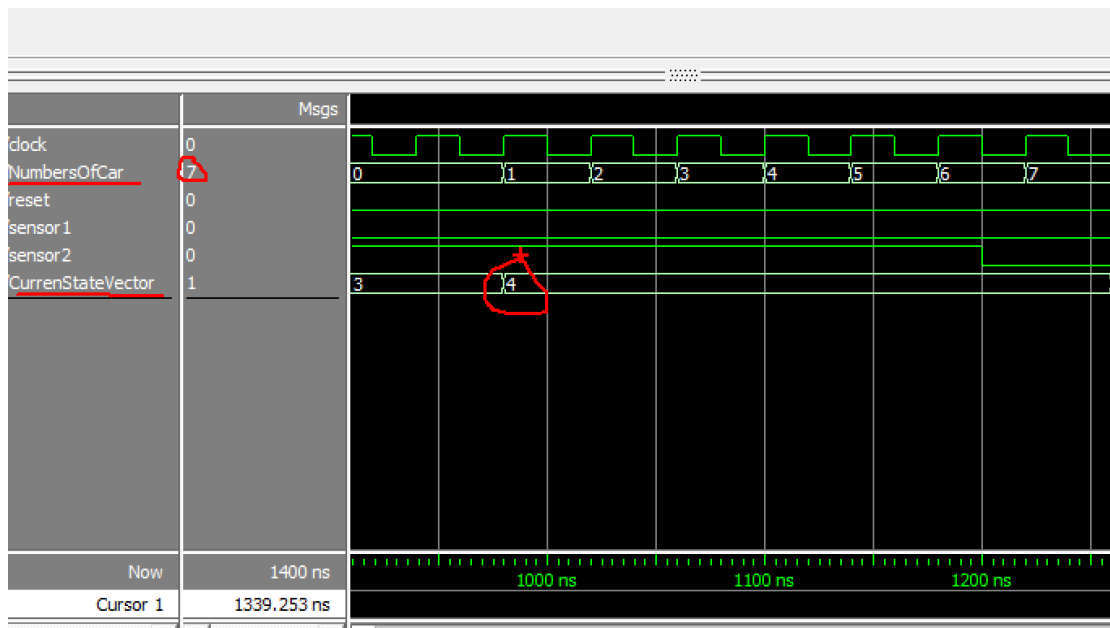
```
sensor1<='0';
```

```
sensor2<='0';
```

```
wait for 300 ns;
```

```
end process;
```

```
END ParkingCounterCarVar_arch;
```



אפשר לראות בסימולציה את הנתונים הבאים:

CurrenStateVector – מציין את המצב שמכונת המצבים נמצאת בו.

NumbersOfCar – מציין את מספר המכונות שנכנסו לחניון.

על-פי דיאגרמת הבועות אפשר להבין, שספירת מכונות נוספת באמצעות המונה תתאפשר רק כשמכונת המצבים תשלים עד תום סיבוב מלא. על-פי הסימולציה מספר המכונות הוא 7 (מוקף באדום).

במצב כלשהו של s3, המונה יגדל ב-1 כל עוד התנאי מתקיים. איך בדיוק? ראו את הקוד של s3 להלן:

when s3 =>

CurrenStateVector<= 4;

count:=count+'1';

if sensor1='0' and sensor2 ='0' then

state:=readData;

else

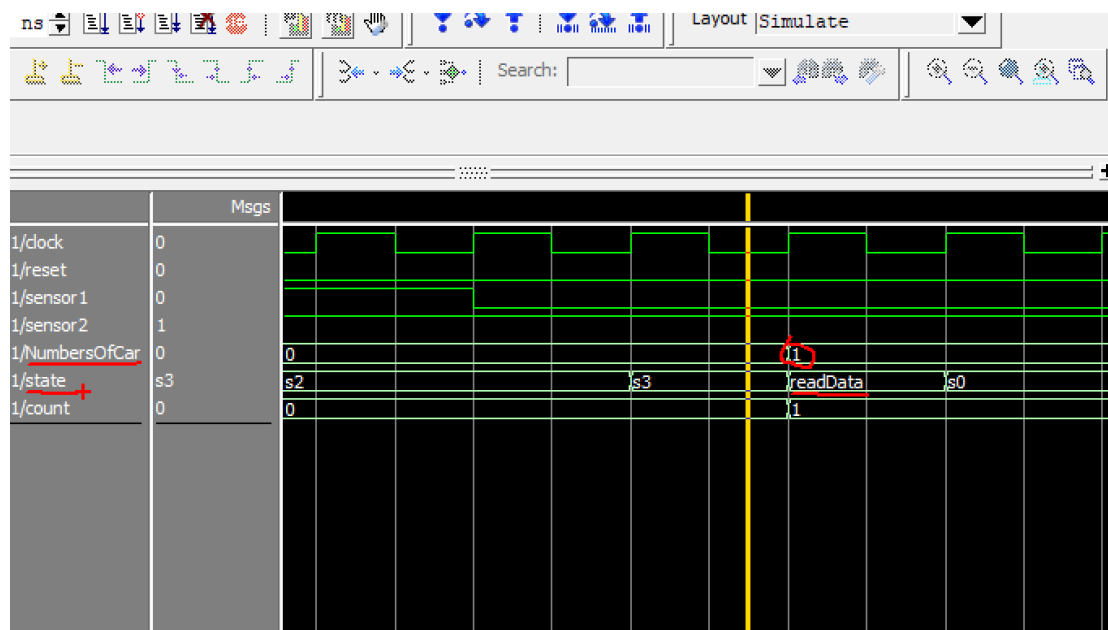
end if;

מראה את s3.

זיהוי המצב של החיישנים (sensor1=, sensor2=) נעשה בעצם בשלב הקודם – s2.

על הקריאה של אותות הכניסה ביציבות וללא שגיאות נלמד בהמשך.

בכל אופן הסימולציה הזאת נכונה יותר, לעומת הקודמת.



ב-NumbersOfCar אפשר לראות את מספר המכוניות.

כמו כן אפשר לראות בסימולציה את המצב `readData`, שבו מספר המכוניות שהמונה (`NumbersOfCar`) כבר קרא הוא 1. ראו את הסימנים המנחים בצבע אדום.

במשתנה type מוגדרים כל המצבים של מכונת המצבים. כפי שכבר הזכרנו זו מערכת לוגית בסופו של דבר, ועל כן יש לתרגם את כל המצבים של המכונה בעזרת קידוד בינארי. במקרה

שלנו יש שישה מצבים, ולכן צריך שלוש סיביות כדי לקודד אותם. המשמעות של המונח "קידוד" היא היכולת להבדיל בין המצבים בשביל הפעולה הנדרשת.

קידוד וקטור במכונת מצבים

קידוד של מכונת מצבים הוא אפשרי על-פי טבלת האמת הזאת:

A0	A1	A2	OUTPUT
0	0	0	START
0	0	1	S0
0	1	0	S1
0	1	1	S2
1	0	0	S3
1	0	1	DON'T CARE
1	1	0	DON'T CARE
1	1	1	DON'T CARE

one hot קידוד – One Hot Decoding

שיטה נפוצה של קידוד היא "one hot". השיטה הזאת יעילה בפשטות שלה, אבל היא יקרה מבחינת מעגלי הדלגלג.

A0	A1	A2	OUTPUT
0	0	0	START=0000_0001
0	0	1	S0=0000_0010
0	1	0	S1=0000_0100
0	1	1	S2=0000_1000
1	0	0	S3=0001_0000
1	0	1	S3=0010_0000
1	1	0	S3=0100_1000
1	1	1	S3=1000_1000

בקידוד "one hot" המספר 1 מוזז כלפי שמאל בכל מצב.

היתרון של "one hot": מספר המצבים לקידוד נקבע לפי ²ⁿ, במקרה שלנו שלוש סיביות לשמונה מצבים.

אפשר לממש את "one hot" בתור מערכת זיכרון, נחזור לזה כשנלמד על מערכות זיכרון.

אפשר גם לתת למכונת המצבים הנחיה שתגרום לה לממש את הקידוד בתור "one hot".

בשביל זה "נגייס" את האופרטורים constant ו-subtype.

האופרטור subtype מאפשר את ההגדרה של קבוצה חדשה מתוך קבוצה קיימת. בתורת הקבוצות זה נקרא "קבוצה חלקית".

מערכת פסיקות לבקרים עם עדיפות Interrupt priority צריך לתכנן מערכת פסיקות לבקרים.

בקרים הם רכיבים (לדוגמה מחשב) שיש להם יכולות מצומצמות יותר מיכולות של (central processing unit) c.p.u. אחד מהמנגנונים של הרכיבים האלה הוא מערכות פסיקה, כלומר בקר שנמצא במצב של הרצת תוכנה ונדרש לבצע תכנות אחר שהוא בסדר עדיפות דחוף יותר. הדוגמה השגרתית והנפוצה בנושא הזה היא פריצת פורץ לבית. הבקר אמור להפסיק את התוכנית העכשווית ולהריץ במקומה תוכנת אזעקה, אבל ייתכן ששני דברים יקרו ביחד באותו הזמן: התלקחות אש בבית וגם כניסה של פורץ אליו.

לפי ההיגיון נראה שתסכימו עם סדר העדיפויות הזה:

1. הפעלת מכשירים של כיבוי אש.
2. הפעלת אזעקה בגלל הפורץ.
3. הפעלת תוכנת ההשקיה של הגינה.
4. הפעלת בקרה של מיזוג אוויר.

מכונת מצבים היא כלי שייתן לנו פתרון יעיל במצב כזה.

על-פי סדר העדיפויות נקודד את המצבים הנדרשים.

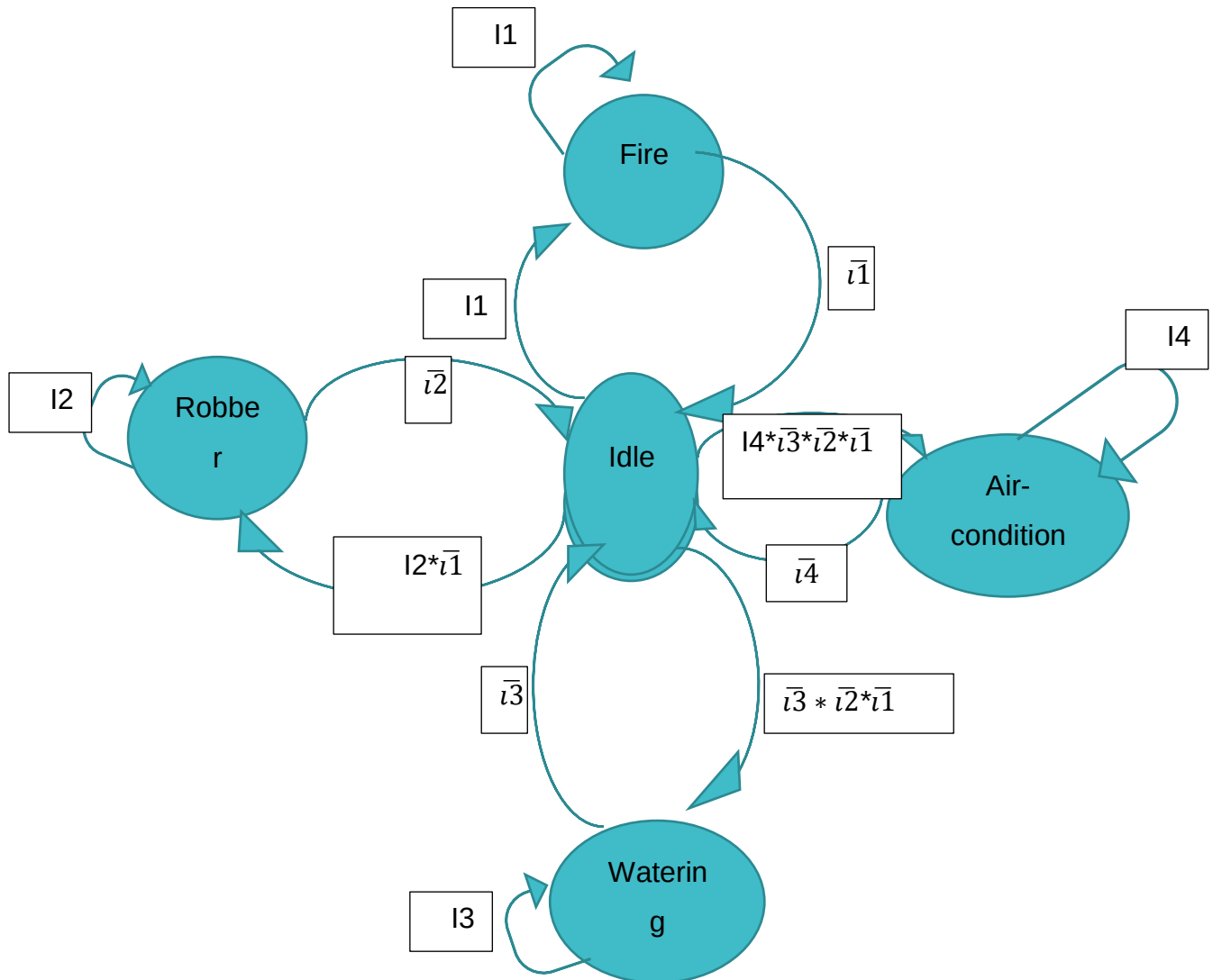
לתוכנית שגרתית נהוג לקרוא idle, מצב המתנה או מצב מנוחה.

נסמן ב-i1,i2,i3,i4 את סדר העדיפויות של הרשימה.

i1 היא העדיפות הגבוהה ביותר, ולעומתה i4 היא העדיפות הנמוכה ביותר.

נניח לצורך הפשטות של התכנון, שהבקר יתפנה כדי לטפל בהפרעה (interrupt) מעדיפות נמוכה רק בסיום ההתעסקות עם הפרעה בעדיפות גבוהה ממנה. זאת אומרת שכל עוד לא כבתה האש בבית לגמרי, הבקר לא יודיע על הפורץ – התעסקות כזו בשלב כזה היא עדיין מוקדמת מדי.

נתבונן בדיאגרמת הבאות:



קביעת העדיפות בטבלת העירור באמצעות משוואות עירור

קביעת העירור במכונת מצבים נעשית באופן הזה:

משוואת המעבר ממצב idle למצב fire נקבעת עם הביטוי $i_1=1$, ובאופן הזה התנאי למעבר אל כל אחד מהמצבים האחרים הוא $i_1=0$. המעבר למצב robber מתקבל אם $i_2=1$ ואילו

$i1=0$. המשוואה שמתקבלת בעקבות זאת היא: $i1 \cdot i2$. למשוואות הבאות נקרא משוואות עירור, הן מסכמות את התנאים שגורמים למכונת המצבים להימצא בכל מצב נתון.

Fire	$i1$
Robber	$i1 \cdot i2$
Watering	$i1 \cdot i2 \cdot i3$
Air condition	$i1 \cdot i2 \cdot i3 \cdot i4$
Idle	$i1 \cdot Q_{fire} + i2 \cdot +$

מכונת המצבים הבאה מיושמת בתור מכונת Moore, שבה מופרד החלק הסינכרוני שתלוי בשעון מהחלק הצירופי ומהחלק הביצועי. כמו שכבר צוין, למכונת מצבים יש שני חלקים עיקריים: משוואות העירור שמתקיימות בשלב ה-next state ולעומת זאת החלק הסינכרוני שמתקיים בחלק ה-current state. בחלק הביצועי נעשה התרגום של חלק ה-current state והמרתו אל ביצועי המכונה בפועל, במילים אחרות התפוקה של המכונה.

המכונה הבאה ממומשת באמצעות "one hot".

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_arith.all;
```

```
use ieee.std_logic_unsigned.all;
```

```
entity InterruptSystem is
```

```
port (
```

```
    clk          :in std_logic;
```

```
    rst          :in std_logic;
```

```
    i1           :in std_logic;
```

```
    i2           :in std_logic;
```

```
    i3           :in std_logic;
```

```

        i4          :in std_logic;

        interrupt :out std_logic_vector(3 downto 0)

    );

end;
```

architecture arc_InterruptSystem of InterruptSystem is

```

    signal Nstate,Cstate          :std_logic_vector(3 downto 0);

    constant idle                 :std_logic_vector(3 downto 0):="0000";
    constant fire                 :std_logic_vector(3 downto 0):="0001";
    constant robber               :std_logic_vector(3 downto 0):="0010";
    constant watering             :std_logic_vector(3 downto 0):="0100";
    constant aircondition         :std_logic_vector(3 downto 0):="1000";

begin

    current_state:process(clk,rst)

    begin

        if rst='1' then

            Cstate<=idle;

        elsif clk'event and clk='1' then

            Cstate<=Nstate;

        end if;

    end process;
```

```
next_state: process(i1,i2,i3,i4)

    begin

        if i1='1' then

            Nstate<=fire;

        elsif i1='0' and i2='1' then

            Nstate<=robber;

        elsif i1='0' and i2='0' and i3='1' then

            Nstate<=watering;

        elsif i1='0' and i2='0' and i3='0' and i4='1' then

            Nstate<=aircondition;

        else

            Nstate<=idle;

        end if;

    end process;

output:process(Cstate)

    begin

        case Cstate is

            when fire =>

                interrupt<="0001";

            when robber =>

                interrupt<="0010";

            when watering =>
```

```

interrupt<="0100";

when aircondition =>

    interrupt<="1000";

when others=>

    interrupt<="0000";

end case;

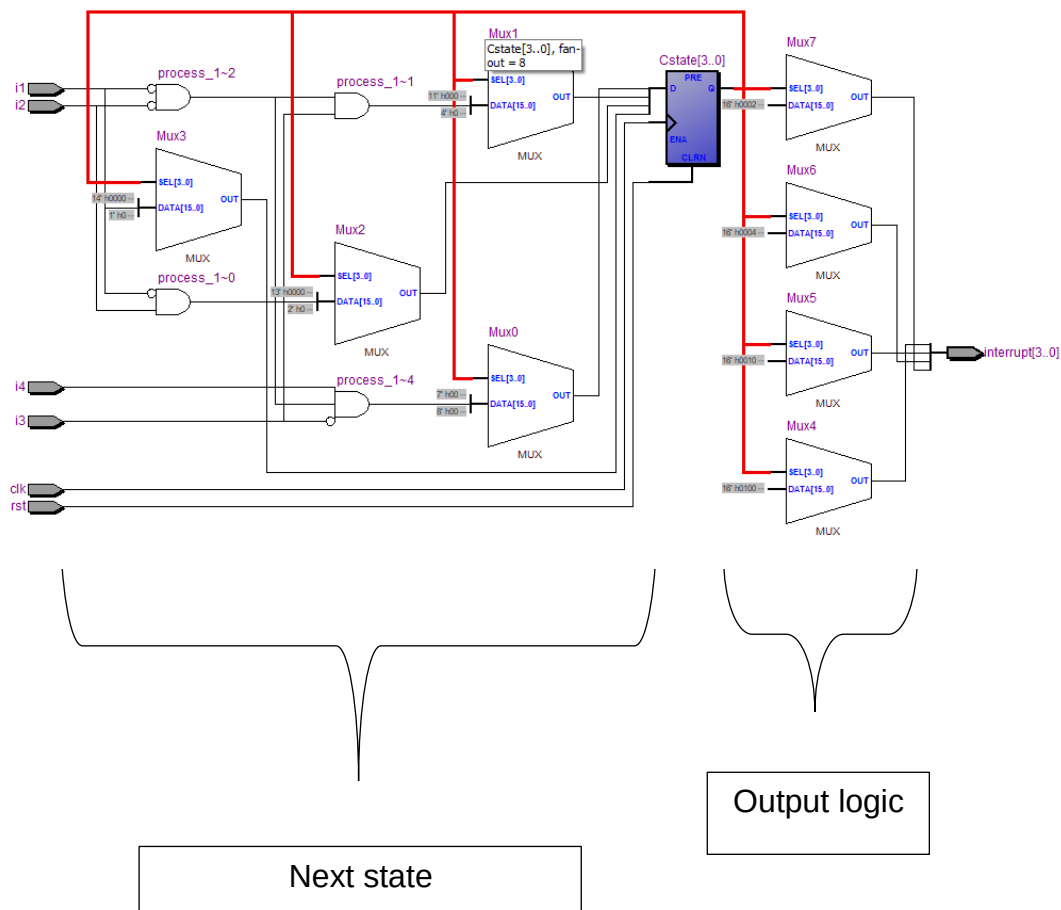
end process;

end;

```

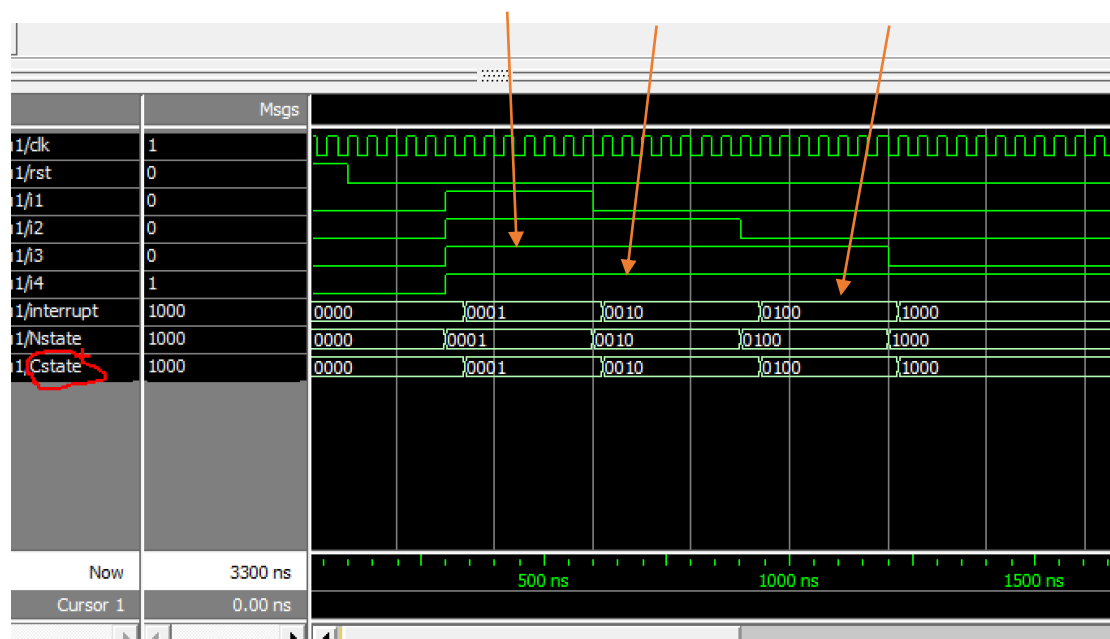
בתרגום של הקוד למערכת לוגית אפשר לזהות את החלקים של מכונת המצבים.

באיור הבא אפשר להבחין בחלקים של מכונת המצבים באופן מרשים. חלק ה-next state ועמו הכניסות i1–i4 מרכיבים ביחד את הכניסות ל-current state. המשוב של המערכת צבוע באדום לכל אורך הדרך: מה-current state ועד ל-next state.



התמקדנו בסימולציה שמתרחשות בה כמה פסיקות בעת ובעונה אחת. תמיד נבחר סימולציה בעלת פסיקה עם עדיפות גבוהה. ראו את הסימולציה, שבה הכניסות מתפקדות באופן הבא:

I1=1	I1=0	I1=1
State=fire	State=robber	State=air condition



פרק 5: מבוא לסינתזה

תכנון בשפת vhdל מכיל רכיבים הכרחיים שחוזרים בכל תכנון, ולצדם רכיבים שהם אמנם לא הכרחיים מבחינת השפה אך חיוניים כדי לתכנן כל קוד באופן נכון וקריא.

הרכיב הראשון הוא ספריות תמיכה.

כמו בכל שפה, יש ספריות שההוראות שלהן מפורשות באמצעות הספרייה הבסיסית:.

```
library ieee;

use ieee.std_logic_1164.all;
```

הספרייה הראשונה – 1164

יש כמובן עוד ספריות. באופן כללי אפשר לומר ששלוש הספריות הבאות עושות את רוב הדברים.

אין טעם להיכנס בשלב המוקדם הזה לתוכן של הספריות האלה.

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_unsigned.all;

use ieee.std_logic_arith.all;
```

התרגום של entity הוא "ישות", זה שם התכנון בצירוף הסיומת is.

Port()

Port (עם סוגריים) – כאן נגדיר את הכניסות ואת היציאות של התכנון.

In,out

In,out – מציינים את הכיוונים של זרימת המידע אל התכנון או החוצה ממנו.

Std_logic

Bit

סיבית היא ספֶּרָה בינארית. ב-vhdl זה מוגדר בדרך כלל בתור std_logic או בתור bit. בספר הזה נאמץ את הגישה של std_logic. על ההבדלים ביניהם נלמד בהמשך.

Std_logic_vector (3 downto 0)

ב-vhdl יותר מביט אחד מצוין בעזרת המילה vector. בדוגמה הזאת מצוינים 4 ביטים.

End<entity name>;

End מסיים את פקודת ה-entity.

Architecture <architecture name> of <entity name> is

ב-**Entity** הגדרנו את הכניסות ואת היציאות של התכנון. לעומת זאת ב-architecture אנחנו מגדירים את התכנון עצמו, כלומר את הלוגיקה שמתלווה אליו.

Architecture name – כאן מוגדר השם של מבנה התכנון. שימו לב, קיים שם שמוגדר ב-entity – זהו שם התכנון. לעומתו יש את השם של מבנה התכנון. החלק שמקשר ביניהם הוא entity name, והוא נמצא בהגדרה של ה-architecture.

Begin

Begin: מציין את תחילת התכנון, התהליך יוסבר בהמשך.

End <architecture name>

End: מציין את סיום הארכיטקטורה בצירוף שם הארכיטקטורה.

```
Entity counter is
Port (
Clock :in std_logic;
Reset :in std_logic;
Q :out std_logic_vector(3 downto 0)
);
End counter;
```


Counter, שם התכנון עם חלקיו האחרים כפי שמצוין.

```
Architecture arc_counter of counter
```

```
Begin
```

```
Body logic
```

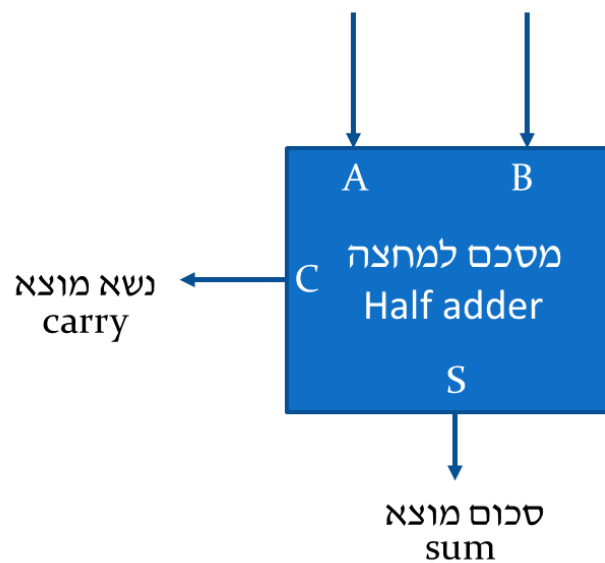
```
גוף התכנון
```

```
End arc_counter;
```

נסביר עוד כמה מושגים באמצעות תרגיל תכנון.

נתכנן מסכם של ארבע סיביות בעזרת Half Adder וגם בעזרת full adder.

מלימודי מערכות ספרתיות אנחנו יודעים שמסכם למחצה יכול לסכם שני ביטים. התוצאה של המסכם מתבטאת ב-S, ואם יש במעגל נשא (carry) התוצאה שלו מתבטאת ב-C. נתוני טבלת האמת מופיעים בטבלה הבאה:



כניסות				יציאות	
ספרות				סכום	נשא
A		B		sum	carry
0		0		0	0
0		1		1	0
1		0		1	0
1	1	0		1	

$S=A \text{ xor } B$

$C=A \text{ AND } B$

נממש את התכנון:

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY HalfAdder IS

PORT

(

a : IN STD_LOGIC;

b : IN STD_LOGIC;

s : OUT STD_LOGIC;

c : OUT STD_LOGIC

);

END HalfAdder;

ARCHITECTURE HalfAdder_architecture OF HalfAdder IS

BEGIN

s<= a xor b;

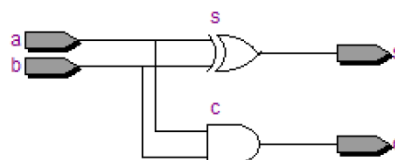
c<= a and b;

END HalfAdder_architecture;

```

תרגול 

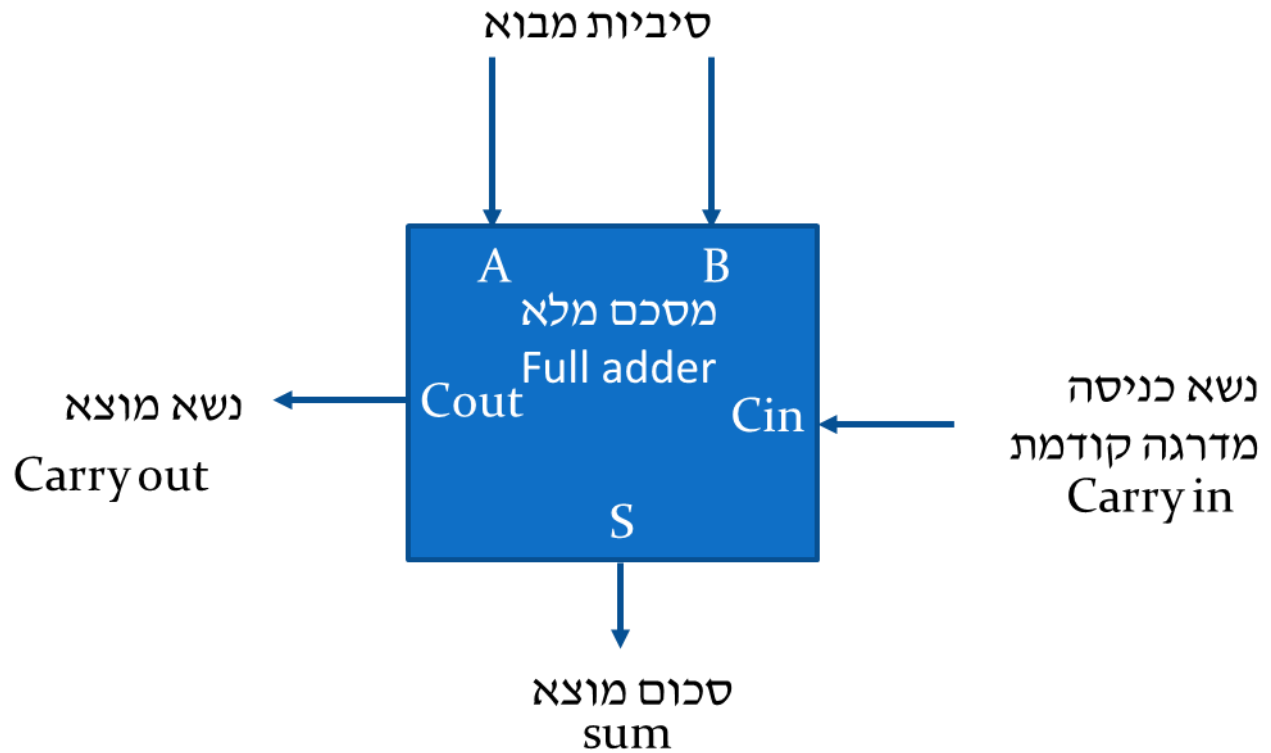
עליכם לזהות את כל החלקים של התכנון על-פי ההגדרות.



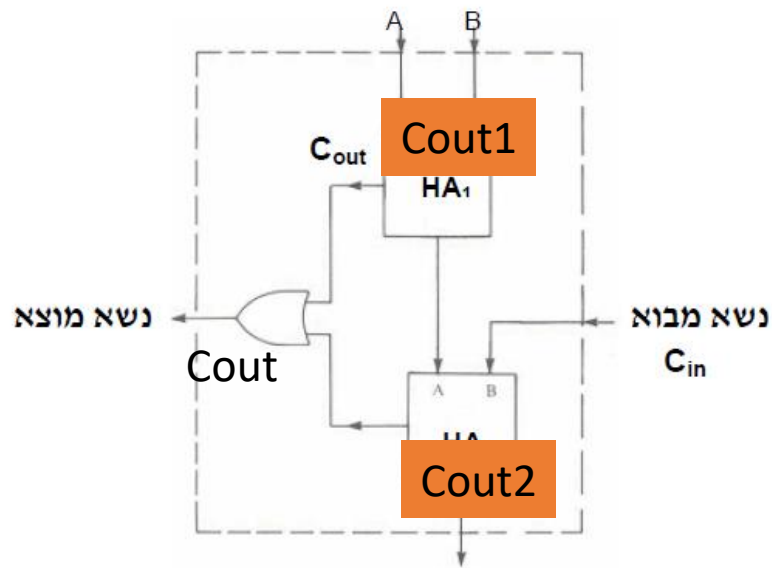
הפקת הציור על-ידי ה-RTL.

מסכם מלא על-ידי שני מסכמים למחצה.

הסבר על: component, signal, port map



מצב	מחובר A	מחובר B	נשא מבוא Cin	סכום S	נשא מוצא Cout
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1



מימוש המערכת באמצעות שני מסכמים למחצה

נצא לקראת אתגר קטן וחשוב. ידוע לנו מהלימודים שבכל שפה עילית יש פונקציות או פעולות, אמנם בעלות כינויים שונים אך עם מטרות דומות.

יש להן שתי מטרות עיקריות: 1. מניעת חזרה מיותרת על דברים שכבר נעשו. 2. תכנון נכון של הפרויקט.

אם נפרק פרויקט לחלקיו נוכל לשלוט בו. כל שינוי שיידרש באחת הפונקציות יצריך בדרך כלל שינוי של כל הפרויקט, לכן לא נצטרך לשנות פרויקט אם נתכנן אותו בלי פונקציות.

ב-vhdl קיימות גישות דומות, אולם היישום שלהן שונה. ההשוואה היא רעיונית בלבד, ודרך היישום שונה לחלוטין.

כמו בפונקציה, יש להכריז על הגישה שנקטנו ולאחר מכן ליישם אותה.

הפרויקט הוא מסכם מלא (full adder) על-ידי מסכמים למחצה (half adder).

מסכם למחצה הוא תת-תוכנית של המסכם המלא.

לתת-תוכניות נקרא components, הם מרכיבים את התוכנית השלמה.

במקרה שלנו המסכם למחצה יהיה component.

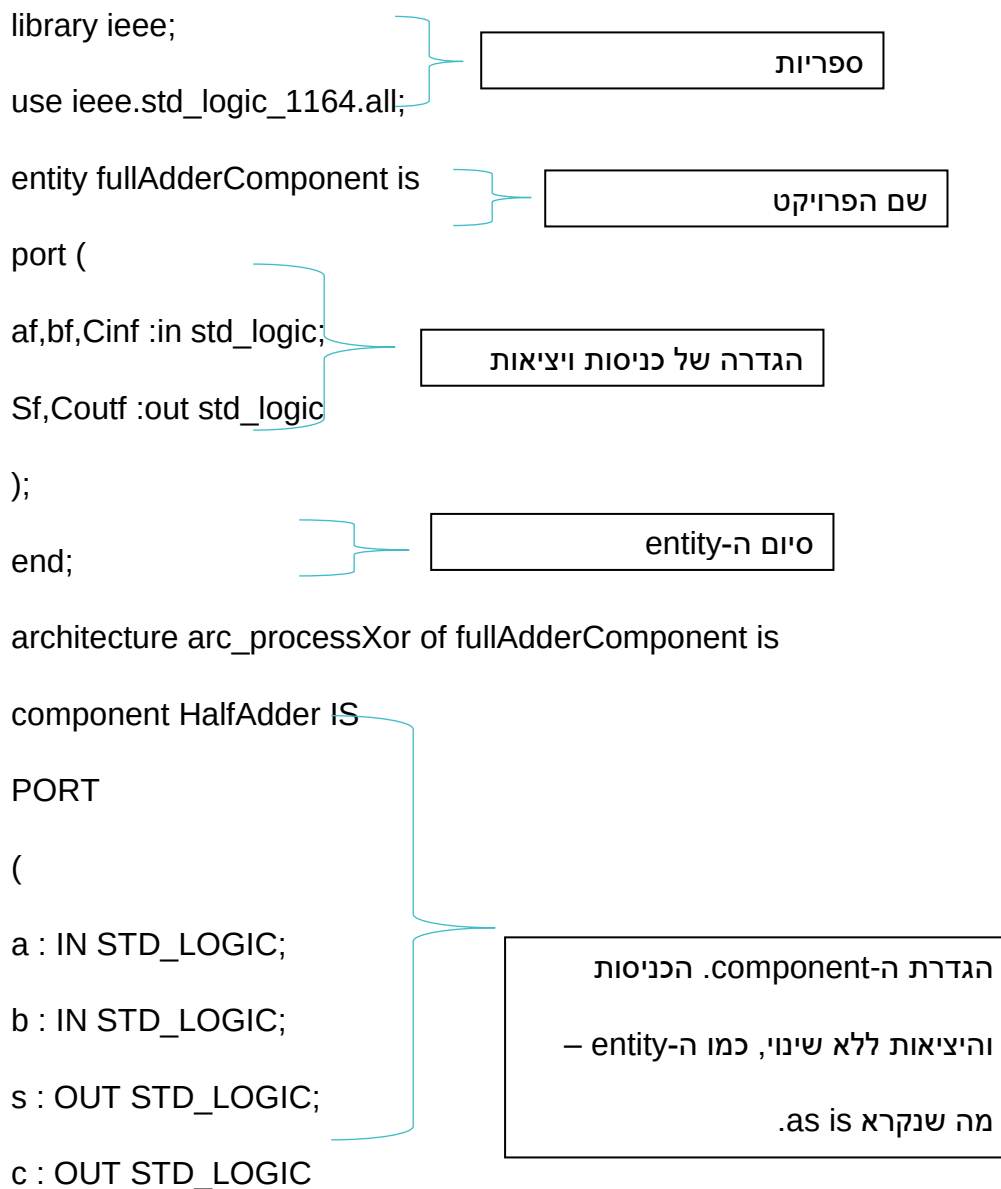
הכניסות של המסכם המלא הן (A,B,Cin) והיציאות שלו נקראות (S,Cout), אבל יש עוד חיבורים פנימיים: (Cout1,Cout2). לחיבורים הפנימיים נקרא signals.

Signal: אמצעי של חיווט וגם משתנה ששומר ערכים זמניים.

המושגים קצת מורכבים, אבל לא לדאוג – נבהיר אותם בהמשך.

נסכם את מה שלמדנו עד כאן. אנחנו רוצים לתכנן מסכם מלא באמצעות מסכמים למחצה שהוכנו מראש.

ניעזר במונחים component ו-signal.



```
);
```

END component;

```
signal Cout1,Cout2,ssf :std_logic;
```

```
begin
```

component-ה סיום

הגדרת המשתנים בעזרת signal

כל תכנון מתחיל ב-begin.

```
u1:HalfAdder
```

```
port map(
```

```
    a=>af,
```

```
    b=>bf,
```

```
    s=>ssf,
```

```
    c=>Cout1);
```

בניית המסכם למחצה הראשון,

```
u2:HalfAdder
```

```
port map(
```

```
    a=>Cinf,
```

```
    b=>ssf,
```

```
    s=>sf,
```

```
    c=>Cout2);
```

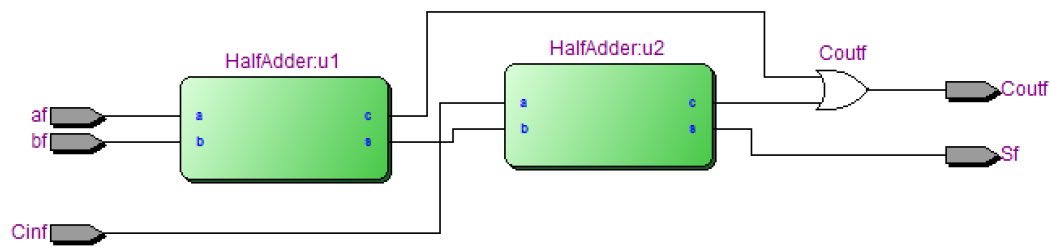
בניית המסכם למחצה השני, חשוב להקפיד על סדר

```
Coutf<=cout1 or cout2;
```

```
end arc_processXor;
```

אפשר לבצע פעולות לוגיות.

סיום הארכיטקטורה



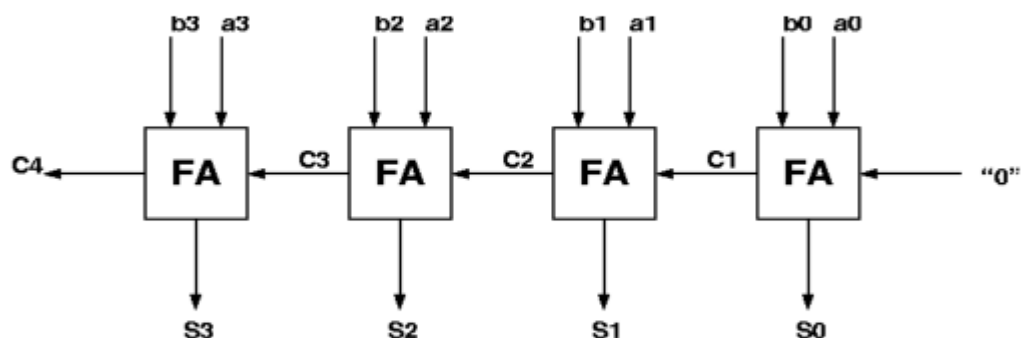
בניית רכיב על-פי component

בבניית רכיב קיים האופרטור $(=)$. כל החלקים מצד שמאל שלו שייכים לרכיב המקורי על-פי ה-component, כלומר המסכם למחצה (half adder). אולם החלקים מצד ימין שייכים לתכנון הנוכחי, כלומר למסכם המלא (full adder).

כל רכיב נקרא בשם כלשהו בתכנון הנוכחי (מסכם מלא), בצירוף נקודתיים ושם הרכיב החדש (component). כל פסוק בשורה מסתיים בפסיק $<, >$ חוץ מהאחרון, שנסגר בעזרת סוגר ובתוספת נקודה-פסיק $<.>$.

מסכם מלא

Ripple Carry Adder



מסכם של 4 סיביות מסוגל לחבר כל זוג של שתי סיביות. התוצאה מתקבלת ב-S, ואם קיים נשא (carry) הדרגה הבאה מתקבלת ב-C.

בתכנון הזה לא נגדיר את הכניסות (a,b) בתור המכפלה של ביט אחד בארבע, אלא בתור וקטור של ארבעה ביט.

```
library ieee;

use ieee.std_logic_1164.all;

entity fourbitAdder is

port (

afbit,bfbit :in std_logic_vector(3 downto 0);

Cinf :in std_logic;

Sfbit :out std_logic_vector(3 downto 0);

Coutf :out std_logic);

end;

architecture arc_fourbitAdder of fourbitAdder is

component fullAdderComponent is

port (

af,bf,Cinf :in std_logic;

Sf,Coutf :out std_logic

);

end component;
```

המשך הקוד בעמוד הבא <<

```
signal carry :std_logic_vector(3 downto 0);

begin

u1:fullAdderComponent
port map(

    af=>afbit(0),

    bf=>bfbits(0),

    sf=>sfbit(0),

    Cinf=>'0',

    Coutf=>carry(0));

u2:fullAdderComponent
port map(

    af=>afbit(1),

    bf=>bfbits(1),

    sf=>sfbit(1),

    Cinf=>carry(0),

    Coutf=>carry(1));

u3:fullAdderComponent
port map(

    af=>afbit(2),

    bf=>bfbits(2),

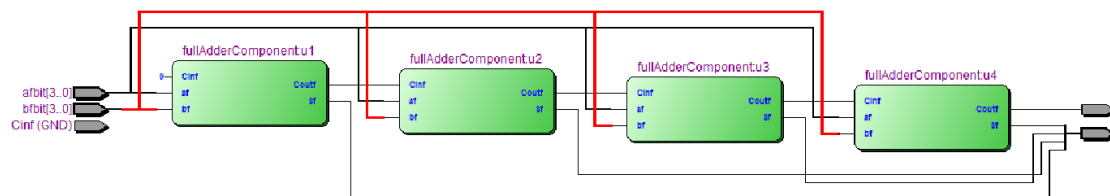
    sf=>sfbit(2),

    Cinf=>carry(1),

    Coutf=>carry(2));
```

המשך הקוד בעמוד הבא <<

```
u4:fullAdderComponent
port map(
    af=>afbit(3),
    bf=>bfbit(3),
    sf=>sfbit(3),
    Cinf=>carry(2),
    Coutf=>carry(3));
    coutf<=carry(3);
end arc_fourbitAdder;
```



וקטור בתור מערך

כמו שכבר צוין קודם וקטור הוא צירוף של כמה ביטים, אבל אפשר לפנות לכל ביט שבתוכו באמצעות שימוש באופרטור "סוגריים".

Carry(0) היא פנייה לביט הראשון, carry(1) היא פנייה לביט השני וכן הלאה.

בפרק הזה הובאו החלקים ההכרחיים לנו כדי ליצור פרויקט בסיסי.

8. מערכים וזיכרונות

לפני שנעסוק במערכים ובזיכרונות נכיר את האופרטור type וגם את האופרטור subtype.

Type מאפשר לנו להגדיר קבוצה של עצמים "תחת קורת גג אחת".

קיימים שני סוגים של type:

Pre-defined type

User defined type

הסוג הראשון – pre define type, מטפל בכל ההגדרות ששייכות לשפה. לדוגמה:

```
type Boolean (false,true)
type bit ('0','1')
```

הסוג השני – user defined type, מטפל בהגדרות שיצר המשתמש. לדוגמה:

```
Type money is (shekel,dollar,mark,dinar)
```

בשני הסוגים מחשיבים את הרכיב השמאלי לעצם הראשון.

המבנה של type:

```
Type <name of type > is < kind of type>
```

דוגמאות:

Color מגדיר קבוצה של צבעים.

```
Type color Is (red,green,blue,yellow);
```

הגדרת ערך אחד מהקבוצה בתור ערך קבוע (constant) ששמו הוא his_choice וערכו blue.

```
Constants his_choice:color :=blue;
```

הדוגמאות האלה מוגדרות בתור user defined type.

Subtype

Subtype היא תת-קבוצה של type, שמאפשרת לנו להגדיר כמה מאיברי הקבוצה של type בתור קבוצה חדשה. לדוגמה:

```
Subtype subcolor is (red,yellow);
```

אפשר גם להגדיר את subtype בפני עצמו ללא קשר ל-type.

```
Subtype small_int is integer range 0 to 255;
```

בעצם זו תת-קבוצה של integer.

```
Subtype byte is bit_vector(7 downto 0);
```

```
Subtype word is bit vector(15 downto 0);
```

קבוצות ה-subtype הוגדרו כאן בתור pre-defined user.

Type as enumerated

אפשר לפשט מבנים מורכבים אם נשתמש באמצעים מוחשיים. אחת הדרכים היא באמצעות קבוצה של שמות.

כדי להגדיר קבוצה של שמות דרוש אמצעי זיהוי מקודד. בקבוצה color הרכיב השמאלי ביותר הוא הראשון, אבל בלתי אפשרי ליישם איבר כזה בלי אמצעי זיהוי כדוגמת "תעודת זהות". יש דרכים אחדות כדי לקודד ערכים בקבוצה, השיטה הזאת נקראת enumerated.

דוגמה: מהרכיבים של color.

Color	binary	hot	gray
Red	00	0001	00
Green	01	0010	01
Blue	10	0100	10
yellow	11	1000	11

בדרך כלל כלי הסינתזה מאפשרים לנו לבחור את צורת הקידוד, או שנוכל לעשות את זה באמצעות קוד.

Binary: הקידוד הוא על-פי ספירה בינארית.

One hot: הקידוד הוא על-פי שינוי המיקום של הספרה אחת.

Gray: בכל איבר מתקיים שינוי אחד.

קידוד של שלושה ביט ייראה כך:

000,001,011,010,110,111,101,100

דוגמה:

```
Type opcode is (add,sub,mul,div);
Signal code:opcode;
Process(code)
Begin
Case code is
When add =>
    Operand <=add code;
    When sub =>
    Operand <=sub code;

When mul =>
Operand <=mul code;

When div =>
```

כל הסוגים של ה-operand מייצגים פעולה סמלית בלבד.

Physical type

קבוצה של ערכים פיזיקליים היא די שימושית בתחום הסימולציה. ראו את הדוגמאות הבאות:

```
type< Physical type definition > is < kind of type >
```

```
Units
```

```
-----
```

```
--
```

Physical type יאפשר לצורבים של הסימולציה להגדיר יחידות.

דוגמה:

```
Type resistance is range 0 to 1e9
```

```
Units
```

```
Ohm:
```

```
Kohm=1000 ohm;
```

```
Mohm=1000 kohm;
```

```
Type time is range -2_147_483_647 TO 2_147_483_647
```

```
Units
```

```
Fs:
```

```
Ps=1000 fs;
```

```
Ns=1000 ps;
```

```
Us=1000 ns;
```

```
Ms=1000 us;
```

Character type

כל הסימנים במקלדת ועוד מקודדים קוד של ascii:

TYPE character **IS** (

nul, soh, stx, etx, eot, enq, ack, bel,
bs, ht, lf, vt, ff, cr, so, si,
dle, dc1, dc2, dc3, dc4, nak, syn, etb,
can, em, sub, esc, fsp, gsp, rsp, usp,

' ', '!', '"', '#', '\$', '%', '&', "'",
'(', ')', '*', '+', ',', '-', '.', '/',
'0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', ':', ';', '<', '=', '>', '?',

'@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z', '[', '\', ']', '^', '_',

`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
'x', 'y', 'z', '{', '|', '}', '~', del,

c128, c129, c130, c131, c132, c133, c134, c135,
c136, c137, c138, c139, c140, c141, c142, c143,
c144, c145, c146, c147, c148, c149, c150, c151,
c152, c153, c154, c155, c156, c157, c158, c159,

-- the character code for 160 is there (NBSP),

-- but prints as no char

'ı', 'İ', 'ϕ', '£', '¤', '¥', 'İ', 'Š',
'ı', '©', 'ª', '«', '¬', '®', '¯',
'±', '²', '³', '´', 'µ', '¶', '·',
'₁', '²', '³', '⁴', '⁵', '⁶', '⁷', '⁸', '⁹',

'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç',
'È', 'É', 'Ê', 'Ë', 'Ì', 'Í', 'Î', 'Ï',
'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×',
'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ', 'ß',

'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç',
'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷',
'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ');

Attribute of scalar types

בשפות של תכנות השאיפה היא להגיע לכתיבה גמישה וקריאה, כתיבה שבה לא נצטרך לכתוב מחדש את הקוד בגלל כל שינוי קטן. בשביל זה השפות האלה מעניקות למתכנת אפשרויות נרחבות כדי להשיג את המטרה הזאת. אחד מאמצעי העזר ב-vhdl הוא האופרטור attribute , המכיל בתוכו גרש לצד המשתנה הנבחר. בדוגמה זו המשתנה הוא T.

נתונה קבוצה של איברים שהוגדרה בתור type.

T'left – הערך השמאלי ביותר.

T'right – הערך הימני ביותר.

T'low – הערך הנמוך ביותר.

T'high – הערך הגבוה ביותר.

T'ascending – הערכים במגמת עלייה, המערכת מחזירה true במקרה שזה נכון.

T'image(x) – המחזורות X שמיוצגת ב-T.

דוגמה: נתונות לפנינו שתי קבוצות – set_index_range ; logic_level.

Type set_index_range is range 21 downto 11;

Type logic_level is (unknown,low,underdrive,high);

מציאת הערך השמאלי:

```
Set_index_range'left=21
```

מציאת הערך הימני:

```
Set_index_range'right=11
```

מציאת הערך הנמוך:

```
Set_index_range'low=11
```

מציאת הערך הגבוה:

```
Set_index_range'left=21
```

מגמת עלייה:

```
Set_index_range'left=false
```

שער שווה-ערך שנמצא במקום ה-14:

```
Set_index_range'left=14
```

Aggregated type

לפעמים צריך לאתחל את המערכים כבר בהתחלה. אפשר לאתחל את המערך מיד או בחלקים.

Aggregated פירושו על-ידי הצטברות.

Positional association – הערכים מקושרים עם אלמנטים משמאל לימין. האיבר הראשון הוא השמאלי ביותר, מימינו האיבר השני וכן הלאה.

Named association – התאמה במפורש בין מיקום האיברים לאיברים עצמם באופן סלקטיבי, הסדר שלהם לא חשוב כאן.

דוגמאות:

```
Type point is array (1 to 3) of real;
```

הפיכת ה-type למשתנה בלי אתחול:

```
Signal view_point :point;
```

Positional association

הפיכת ה-type למשתנה באמצעות אתחול מידי:

```
Signal view_point :point:=(10.0, 20.0 0.0);
```

התאמה בין האיבר למקום שלו בקבוצה, אפשר לתאם אחרת אם התוכנה מאפשרת את זה.

```
Signal view_point:point(1=>10.0,2=>20.0,3=>30.0);
```

Array type

Array type הוא למעשה מערך של type.

המבנה של array type הוא:

```
Type< name of array> is array of < kind of array >
```

דוגמה:

מערך של 8 bit:

```
Type BitArray is(7 downto 0) of bit;
```

מערך של 8 bit:

```
Type arrayofbyte is (15 downto 0) of bit_vector(7 downto 0);
```

הדוגמא הבאה תדגים כתיבה בהירה של קוד.

```
subtype byte is integer range 7 downto 0;
```

```
type sword is array (0 to 15) of byte;
```

```
signal word :sword;
```

ה-subtype מגדיר את byte בתור 8 bit.

המערך הוא בגודל של 16 כתובות, כלומר בגודל של byte.

הזכרנו לא מעט, שהמשתנים היחידים ב-vhdl הם signal ו-variable.

לכן מתקיימת ההמרה מה-type ל-signal.

אפשר לכתוב זאת בדרך ברורה יותר.

הגדרות ראשונות של byte ושל address:

```
subtype byte is integer range 7 downto 0;  
subtype address is integer range 15 downto 0;
```

להלן דוגמה לכתיבה קריאה יותר. אם יידרש שינוי בטווח של ה-address לא נצטרך לשנות את כל המקומות שבהם השתמשנו בפקודה הזאת, אלא רק את ההגדרה הראשית.

```
type mem is array (address) of byte;  
signal RamMem : mem;
```

זיכרון שממומש באמצעות type

זו יצירת זיכרון של 4 כתובות ברוחב של 8 bit, כלומר byte.

הזיכרון נוצר בעזרת האופרטור type.

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_arith.all;  
USE ieee.std_logic_unsigned.all;
```

```
ENTITY ram IS
```

```
PORT
```

```
(
```

```
  clk    : IN STD_LOGIC;
```

```
rst    : IN STD_LOGIC;

re     : IN std_LOGIC;

add    : IN integer range 15 downto 0;

data   : IN integer range 255 downto 0;

Q      : OUT integer range 255 downto 0

);

END;

ARCHITECTURE arc_ram OF ram IS

subtype byte is integer range 255 downto 0;

subtype address is integer range 3 downto 0;

type mem is array (address) of byte;

signal RamMem :mem;

BEGIN

process(clk,rst)

begin

    if rst='1' then

        q<=0;

    elsif clk'event and clk='1' then

        if re='1' then

            RamMem(add)<=data;

        else

            Q<=RamMem(add);

        end if;

    end if;

end if;
```

```
end process;
```

```
END;
```

נתבונן בהוראות הבאות:

RamMem(add) הוא זיכרון שאפשר לכתוב אליו רק אם $r=1$.

לכן: $RamMem(add) \leq data$.

```
if re='1' then
```

```
    RamMem(add) <= data;
```

```
else
```

```
    Q <= RamMem(add);
```

```
end if;
```

החלק re שולט על יכולת הכתיבה והקריאה מתוך הזיכרון.

$RamMem(add) \leq data$ – ההוראה הזאת מכניסה לזיכרון את הנתון (data) לפי הכתובת (add).

פעולה כזו של הזנת נתון נקראת write.

$Q \leq RamMem(add)$ – התפקיד של ההוראה הזאת לעומתה הוא להוציא נתון מהזיכרון דרך היציאה Q.

הקוד של הסימולציה הוא:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;


ENTITY ram_vhd_tst IS

END ram_vhd_tst;

ARCHITECTURE ram_arch OF ram_vhd_tst IS

-- constants

-- signals

SIGNAL add : integer range 15 downto 0;

SIGNAL clk : STD_LOGIC:= '0';

SIGNAL data : integer range 255 downto 0;

SIGNAL Q : integer range 15 downto 0;

SIGNAL re : STD_LOGIC;

SIGNAL rst : STD_LOGIC;

COMPONENT ram

    PORT (

        add : IN integer range 15 downto 0;

        clk : IN STD_LOGIC;

        data : IN integer range 255 downto 0;

        Q : OUT integer range 15 downto 0;

        re : IN STD_LOGIC;

        rst : IN STD_LOGIC

    );
```

```
END COMPONENT;

BEGIN

    i1 : ram

    PORT MAP (

-- list connections between master ports and signals

        add => add,

        clk => clk,

        data => data,

        Q => Q,

        re => re,

        rst => rst

    );

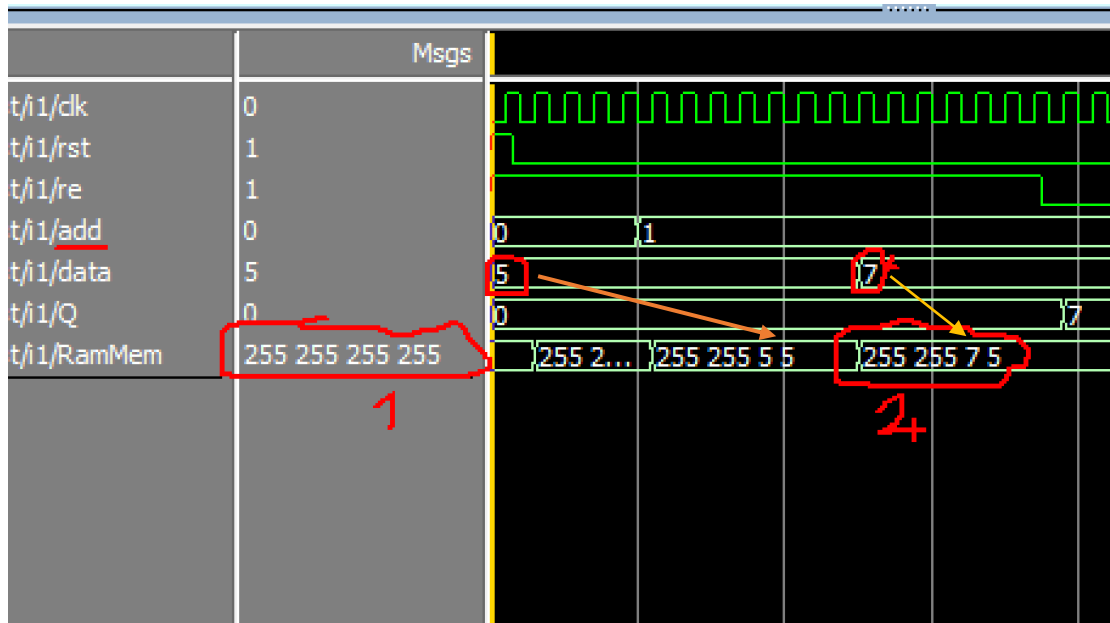
    clk<= not clk after 20 ns;

    rst<= '1','0' after 50 ns;

    re<= '1','0' after 500 ns;

    data<= 5,7 after 600 ns;

END ram_arch;
```

אפשר לראות בסימולציה כתיבה אל הכתובת $add=0$ בצירוף $re=1$ והנתון $data=5$.

כאשר $re=0$ ה- $data$ מופיע ביציאה Q .

שימו לב לסימונים האדומים. בהתחלה של התהליך כל הנתונים בזיכרון הם 255 (לפי מה שמוקף באדום בתרשים של הסימולציה ומסומן במספר 1).

אבל בסוף של התהליך הנתונים 5 ו-7 "השתבצו" בזיכרון (מוקף באדום ומסומן במספר 2). עקבו אחרי כתובות ה- add של הנתונים כדי להבין את סיבת ההתמקמות שלהם בזיכרון. היעזרו גם בחצים הכתומים.

Tri state

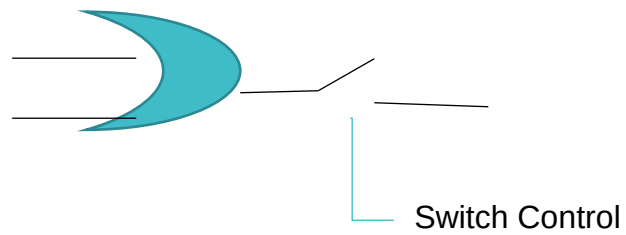
על-פי הדוגמה הקודמת רכיב הזיכרון פעיל כל הזמן, נמצא במצב קריאה $read$ או נמצא במצב כתיבה $write$.

הגישה אל הזיכרון לא נחוצה תמיד. הפתרון למצב כזה הוא ניתוק קווי ה- $data$ מהמעגל.

זה בעצם עוד מצב לוגי, נוסף על המצב הבינארי.

בעגה המקצועית הוא נקרא $high z$ או $tri state$.

$High impedance$ – לרכיבים כאלה יש מתג מבוקר שמנתק או מחבר את חלקי היציאה.



גם רכיבי FPGA בנויים כמו ה-tri state. שפת ה-vhdl תומכת כמובן בתכונה הזאת.
מצב שלישי מוגדר מתוך ה-type character בתור 'Z'. ה-'Z' משמש ב-vhdl כמו tri state.

קוד יישום של tri state:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_arith.all;

USE ieee.std_logic_unsigned.all;

ENTITY triState IS

PORT

(

sel :in std_logic;

a,b :in std_logic;

f :out std_logic

);

END;

ARCHITECTURE arc_triState OF triState IS

BEGIN

process(sel,a,b)

begin

    if sel='1' then

        f<=a and b;

    else

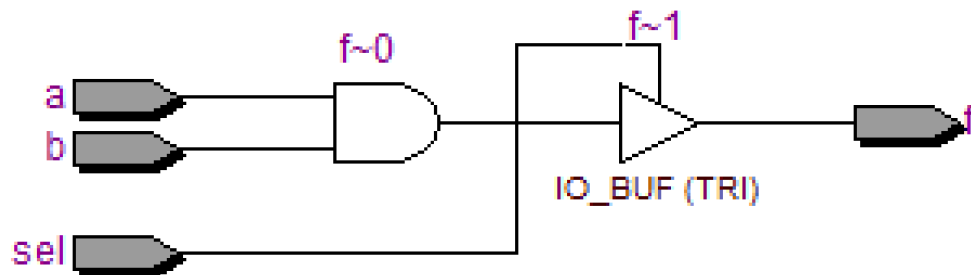
        f<= 'Z';

    end if;

end process;

END;
```

התוצאה של הקוד:



הרכיב buf הוא בעצם מתג מבוקר, גם הוא מסומן בתור tri.

	Msgs	
_vhd_tst/a	0	
_vhd_tst/b	0	
_vhd_tst/f	Z	
_vhd_tst/sel	0	

בסימולציה אפשר להבחין שכאשר $sel=0$ המוצא f משתנה ל- $high\ z$. המצב הלוגי הזה הוא למעשה מצב של נתק. אפשר לראות את זה על-פי הסימנים האדומים.

עכשיו אפשר לעשות את השינוי בזיכרון.

מימוש של זיכרון עם רכיבי tri state

בקוד המקורי נצטרך לשנות את אופן ההגדרה של data מ-integer range ל-std_logic_vector, מפני שהוא לא תומך בפקודה 'Z' high.

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_arith.all;

USE ieee.std_logic_unsigned.all;


ENTITY ramHighZ IS

PORT

(
    clk    : IN STD_LOGIC;
    rst    : IN STD_LOGIC;
    sel    : IN STD_LOGIC;
    re     : IN STD_LOGIC;
    add    : IN integer range 15 downto 0;
    data   : IN std_logic_vector(7 downto 0);
    Q      : OUT std_logic_vector(7 downto 0)
);

END;


ARCHITECTURE arc_ramHighZ OF ramHighZ IS

    subtype byte is std_logic_vector(7 downto 0);
```

```
subtype address is integer range 15 downto 0;

type mem is array (address) of byte;

signal RamMem : mem;

signal SQ : std_logic_vector(7 downto 0);

BEGIN

process(clk,rst)

begin

    if rst='1' then

        SQ<=(others=>'0');

    elsif clk'event and clk='1' then

        if re='1' then

            RamMem(add)<=data;

        else

            SQ<=RamMem(add);

        end if;

    end if;

end process;

process(sel,SQ)

begin

    if sel='1' then

        Q<=SQ;

    else
```

```

Q<= "ZZZZZZZZ";

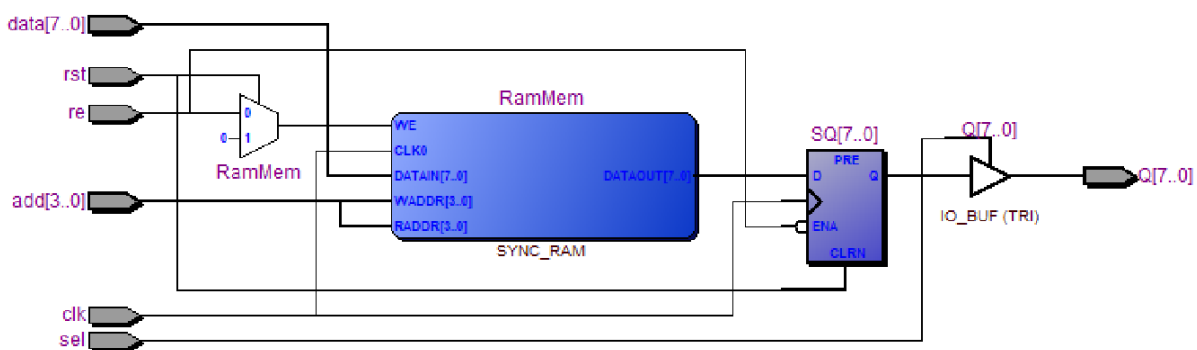
end if;

end process;

END;

```

המימוש של הקוד באמצעות התוכנה:



לא נדון בשלב הזה באופן המימוש. הכתיבה כאן משתמשת ברכיבים מוכנים של סביבת העבודה quartus. אפשר לראות שה-buffer, רכיב ה-tri state, משוכפל כמו היציאה Q[7..0]. כלומר קיימים 8 רכיבי tri state – אחד לכל יציאה. התרגום של ה-buffer מתפקד בעצם בתור חוצץ.

הקוד של הסימולציה הוא:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY ramHighZ_vhd_tst IS

END ramHighZ_vhd_tst;

ARCHITECTURE ramHighZ_arch OF ramHighZ_vhd_tst IS

    SIGNAL add : integer range 3 downto 0;

    SIGNAL clk : STD_LOGIC:= '0';

    SIGNAL data : STD_LOGIC_VECTOR(7 DOWNTO 0);

    SIGNAL Q : STD_LOGIC_VECTOR(7 DOWNTO 0);

    SIGNAL re : STD_LOGIC;

    SIGNAL rst : STD_LOGIC;

    SIGNAL sel : STD_LOGIC;

    COMPONENT ramHighZ

        PORT (

            add : IN integer range 3 downto 0;

            clk : IN STD_LOGIC;

            data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);

            Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);

            re : IN STD_LOGIC;

            rst : IN STD_LOGIC;

            sel : IN STD_LOGIC

        );

    END COMPONENT;
```



```
BEGIN

    i1 : ramHighZ

    PORT MAP (

-- list connections between master ports and signals

        add => add,

        clk => clk,

        data => data,

        Q => Q,

        re => re,

        rst => rst,

        sel => sel

    );

    clk<= not clk after 20 ns;

    rst<='1','0' after 50 ns;

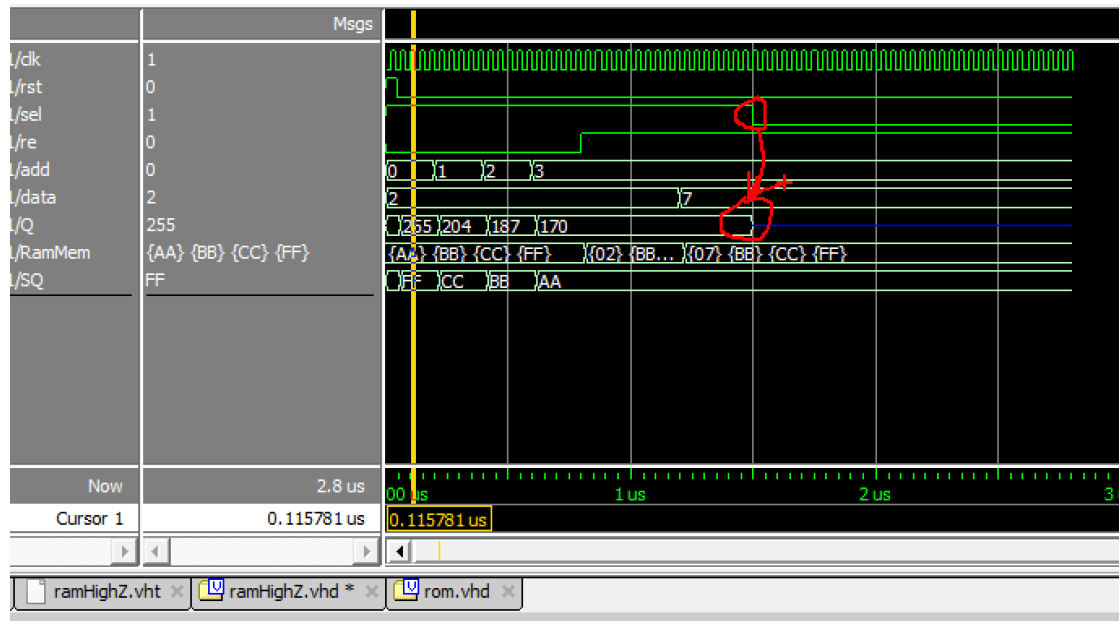
    re<='0','1' after 800 ns;

    data<=x"02",x"07" after 1200 ns;

    add<=0,1 after 200 ns,2 after 400 ns,3 after 600 ns;

    sel<='1','0' after 1500ns;

END ramHighZ_arch;
```



הפעלת זיכרון R.O.M. באמצעות אתחול

אפשר לנצל את type array בתור רכיבי זיכרון מסוג R.O.M.

נדגים המרה ממצב בינארי למצב של שבעה מקטעים (bcd to 7 segment).

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

USE ieee.std_logic_arith.all;

USE ieee.std_logic_unsigned.all;


ENTITY rom IS

PORT

(
    clk    : IN STD_LOGIC;
    rst    : IN STD_LOGIC;
    re     : IN STD_LOGIC;
    add    : IN integer range 3 downto 0;
    Q      : OUT integer range 255 downto 0
);


END;
```

```

ARCHITECTURE arc_rom OF rom IS
  subtype byte is integer range 255 downto 0;
  subtype address is integer range 3 downto 0;
  type mem is array (address) of byte;

```

קביעת התוכן של זיכרון R.O.M. באמצעות אתחול של המערך

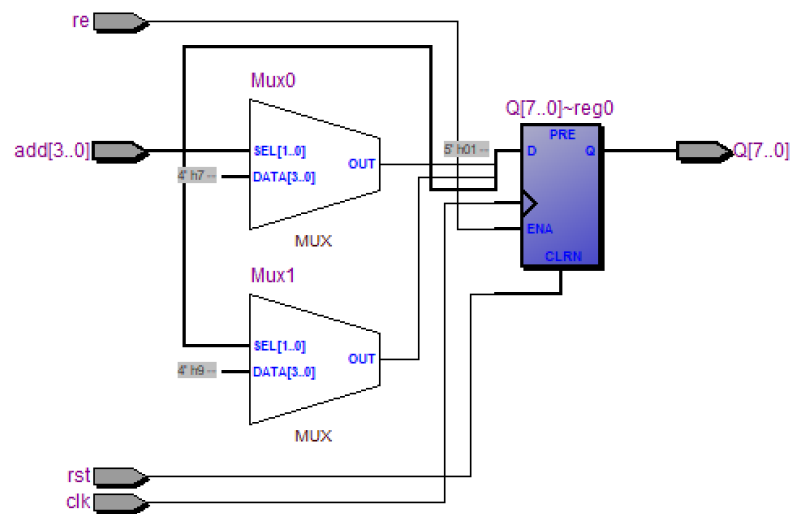
```

signal RamMem : mem := (12,13,14,15);

BEGIN
  process(clk,rst)
  begin
    if rst='1' then
      q<=0;
    elsif clk'event and clk='1' then
      if re='1' then
        Q<=RamMem(add);
      end if;
    end if;
  end process;

END;

```



הקוד של הסימולציה הוא:

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY rom_vhd_tst IS

END rom_vhd_tst;

ARCHITECTURE rom_arch OF rom_vhd_tst IS

    SIGNAL add : integer range 3 downto 0;

    SIGNAL clk : STD_LOGIC:= '0';

    SIGNAL Q : integer range 255 downto 0;

    SIGNAL re : STD_LOGIC;

    SIGNAL rst : STD_LOGIC;
    
```

```

COMPONENT rom

    PORT (

        add : IN integer range 3 downto 0;

        clk : IN STD_LOGIC;

        Q   : OUT integer range 255 downto 0;

        re  : IN STD_LOGIC;

        rst : IN STD_LOGIC

    );

END COMPONENT;

BEGIN

    i1 : rom

    PORT MAP (

        add => add,

        clk => clk,

        Q => Q,

        re => re,

        rst => rst

    );

    clk<= not clk after 20 ns;

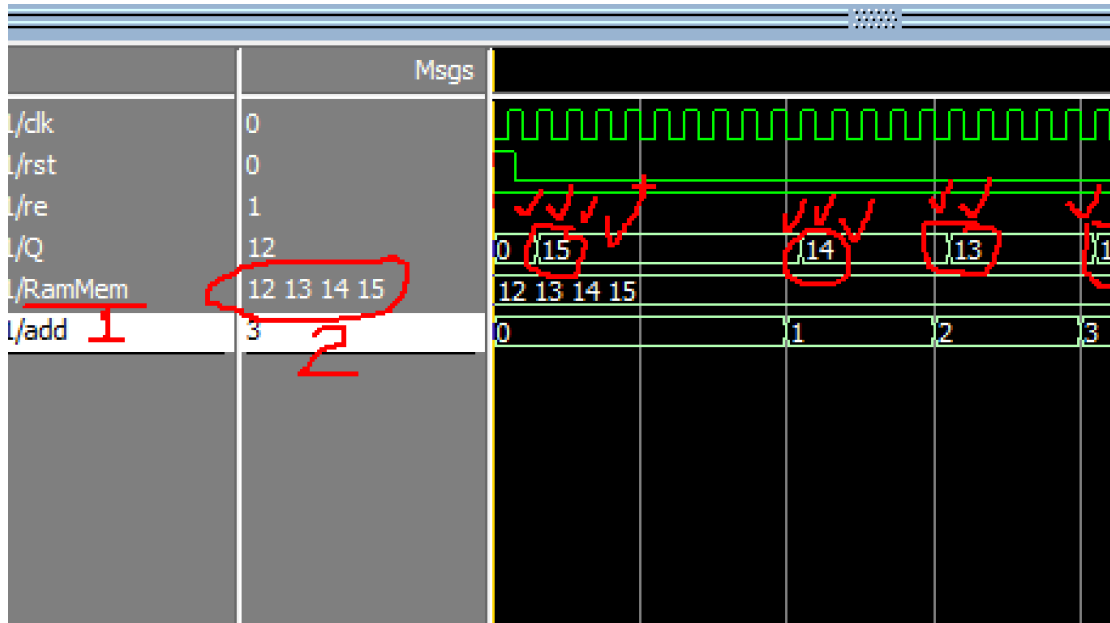
    rst<='1','0' after 30 ns;

    add<=0,0 after 200 ns,1 after 400 ns,2 after 600 ns,3 after 800 ns;

    re<='1','0' after 1500 ns;

END rom_arch;

```



הסבר על הסימולציה בהשוואה לכתיבה

ה-RamMem הוא התוכן של הזיכרון על-פי האתחול של ה-signal:

```
signal RamMem :mem:=(11,12,13,14);
```

כמו שצוין, הנתון הראשון הוא 12 – ראו את יציאה Q בסימולציה. הוא באמת מופיע בה ראשון, ויש סימן של v מעליו. מספר סימני ה-v מעל הנתונים מלמד על סדר ההופעה שלהם. שימו לב להבדל בין מה שרשום ב-signal ובין מה שרשום בסימולציה. לכאורה הנתונים הפוכים אבל זה לא המצב באמת, משום שלפי הסימולציה ההתקדמות בזמן היא מימין לשמאל.

מערך לא סופי – Unconstrained array type

עד כאן הכרנו מערכים סופיים בעזרת קביעה של התחום שלהם. הגודל של מערכים לא סופיים נקבע על-פי התוכן שלהם. כדי לתכנן חומרה אנחנו נדרשים קודם לחשוב היטב – באופן כללי ובמערכות זיכרון בפרט.

זיכרון מהסוג הזה הוא שימושי בעיקר במחרוזות – כשלא נוח לספור את מספר האותיות.

הדבר דומה לאתחול של מערך מהסוג string.

האופרטור שתומך במערך הזה ללא הגבלה הוא <> – פעמיים אי-שוויון.

דוגמה:

Type type_name **array is** (type mark range <>)

דוגמה:

Type sample is array (natural range <>) of integer;

Variable short_sample_buf : sample(0 to 30);

String

לעניין המחרוזות קיימת ב-Vhdl הגדרה בסיסית. כמו שאר ההגדרות ששייכות לשפה גם String הוא חלק ממנה. זה מערך של positive עם character, ולפיכך אפשר להגדיר אותו בתור subtype.

Type **string** is array (positive range <>) of character;

זכור מוגדר כאן type מסוג תו (=character).

בדרך כלל ה-type יכול להיות בסדר עולה או בסדר יורד (ascending או descending), במחרוזות נהוג רק סדר עולה (ascending).

הכתובת הראשונה היא 1 ולא 0, ראו בדוגמה של char. נקודה נוספת: חשוב שיהיה כאן סדר עולה – ascending.

יש רכיבים שכדי להפעיל אותם נדרשים תווים של ascii. זיכרון של מערך לא סופי בהחלט יכול לסייע בשביל זה.

שימו לב לאופן ההגדרה של ה-signal, חלק ממנו מוגדר בתור aggregated.

דוגמה:

subtype char is string (1 to 4);

signal RamMemChar:char:=('a','b','c',others=>'d');

דוגמא מעשית:**שידור רצוף של סדרת תווים**

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY romOfChar IS

PORT

(

    clk    : IN STD_LOGIC;

    rst    : IN STD_LOGIC;

    re     : IN STD_LOGIC;

    add    : IN integer range 3 downto 0;

    Q      : OUT character

);

END;

ARCHITECTURE arc_romOfChar OF romOfChar IS

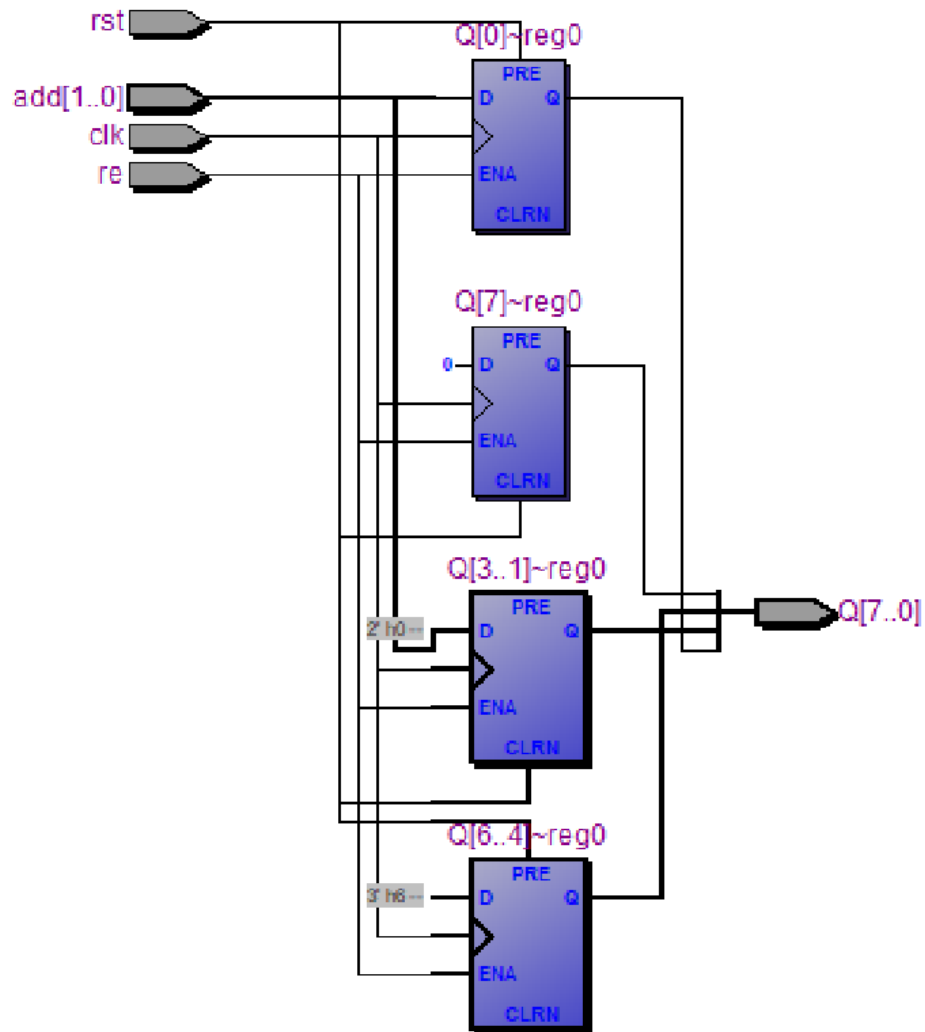
    subtype char is string (1 to 4);

    signal RamMemChar:char:=('a','b','c',others=>'d');

BEGIN
```

```
process(clk,rst)
begin
    if rst= '1' then
        q<='q';
    elsif clk'event and clk='1' then
        if re='1' then
            Q<=RamMemChar(add);
        end if;
    end if;
end process;

END;
```



הקוד של הסימולציה הוא:

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

ENTITY romOfChar_vhd_tst IS
END romOfChar_vhd_tst;

ARCHITECTURE romOfChar_arch OF romOfChar_vhd_tst IS

SIGNAL add : integer range 3 downto 0;

SIGNAL clk : STD_LOGIC:= '0';
  
```

```

SIGNAL Q : character;

SIGNAL re : STD_LOGIC;

SIGNAL rst : STD_LOGIC;

COMPONENT romOfChar

    PORT (

        add : IN integer range 3 downto 0;

        clk : IN STD_LOGIC;

        Q : OUT character;

        re : IN STD_LOGIC;

        rst : IN STD_LOGIC

    );

END COMPONENT;

BEGIN

    i1 : romOfChar

        PORT MAP (

-- list connections between master ports and signals

            add => add,

            clk => clk,

            Q => Q,

            re => re,

            rst => rst

        );

    clk<= not clk after 20 ns;

    rst<='1','0' after 30 ns;

```

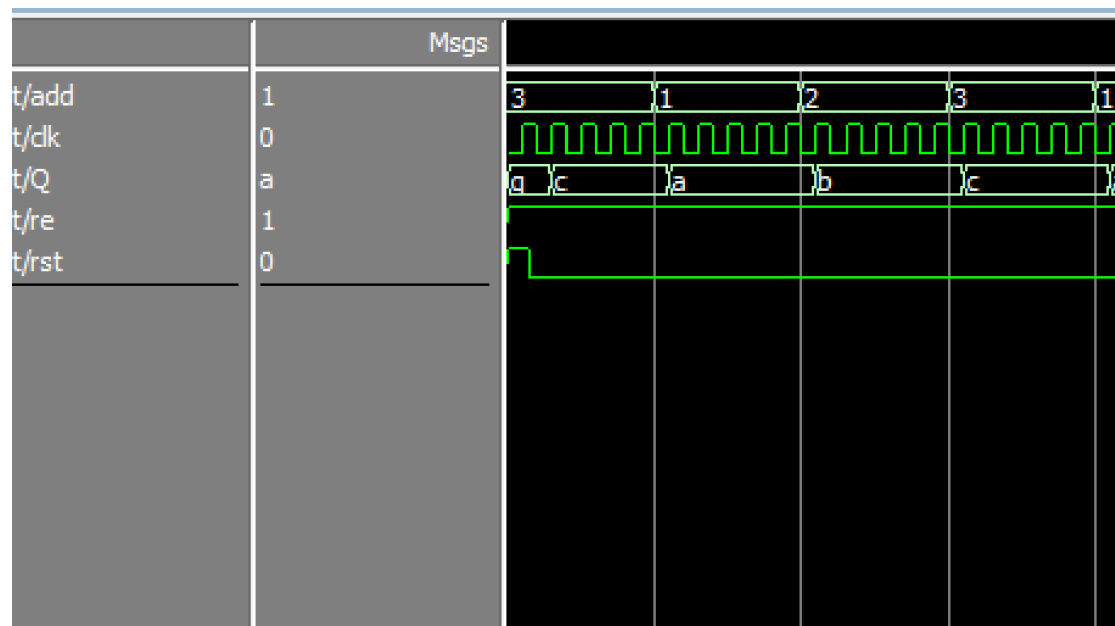
add<=1 after 200 ns,2 after 400 ns,3 after 600 ns,1 after 800 ns;

re<='1','0' after 1200 ns;

END romOfChar_arch;

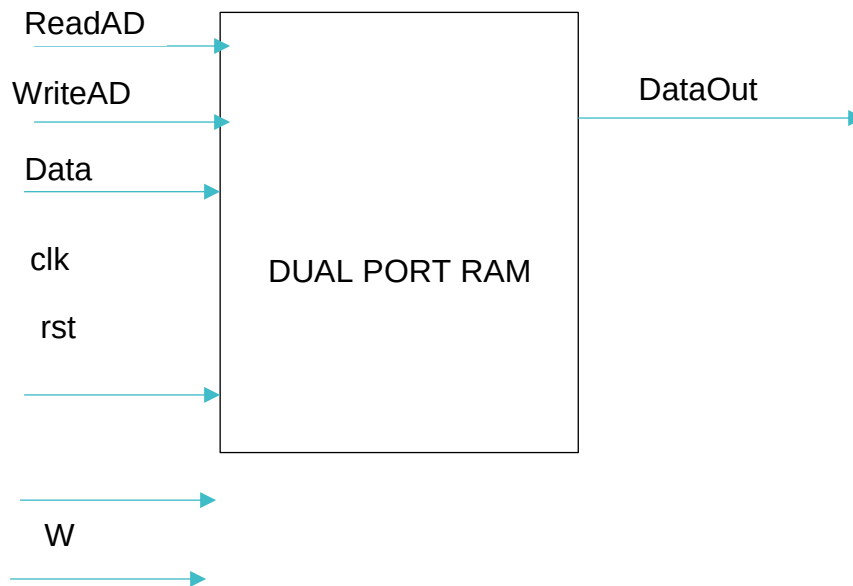
הסימולציה מתארת שידור דרך Q של התווים שנרשמו באופן הבא:

signal RamMemChar:char:=('a','b','c',others=>'d');



DUAL PORT RAM

זהו רכיב זיכרון שאפשר לכתוב אותו ולקרוא בו בעת ובעונה אחת.



בהזדמנות הזאת נזכיר שכדאי לאפיין כל פרויקט בעזרת diagram block – "קופסה שחורה".

בתור תמונה ראשונה זהו בהחלט תיאור התחלתי ונוח כדי לאפיין בו את הפרויקט.

נתאר עכשיו את הכניסות ואת היציאות.

ReadADD: מצביע על הכתובת שממנה קוראים.

WriteADD: מצביע על הכתובת שאליה כותבים.

Data: מאפשר כתיבה אל תוך הזיכרון.

WE: מאפשר כתיבה כלשהי.

DataOut: יציאה מן הזיכרון.

הכתיבה אל תוך הזיכרון היא רק "באישור" של WE, ולעומת זאת הקריאה מתקיימת תמיד.

Package

נצעד צעד נוסף לעבר כתיבה מקצועית יותר. פרויקטים גדולים מורכבים מחלקים, ואחד מהם הוא top – כמו החלק main ב-cpp. חלק כזה אמור להכיל רק חיבור של כל תתי-התכניות ולא את ההגדרות.

בשפת ה-vhdl אפשר להגדיר את כל ההגדרות ב-package.

יתרון נוסף לקיבוץ ההגדרות ב-package הוא החיסכון בהגדרות כפולות ובתתי-פרויקטים, כדוגמת ה-byte שהוגדר בפרויקט הזה.

מבנה – package

Package identifier is

Body of package

End package identifier

ובעברית:

Package is מציין

גוף

End package מציין

אמנם שומרים את ה-package בתור קובץ נפרד, אך גם מכלילים אותו בקובץ של הפרויקט הגדול. ההכללה של הקובץ נעשית באמצעות הפקודה הבאה:

```
USE work.name of package.all;
```

בדוגמה הבאה כל ההגדרות מתקיימות ב-package.

המבנה הבא מכיל דוגמה לקובץ ה-package.

```
package DualPortRamPackage is
```

```

subtype byte is integer range 255 downto 0;

subtype nibble is integer range 3 downto 0;

type DPRAM is array (nibble) of byte;

signal ram:DPRAM;

end package DualPortRamPackage;

```

הקוד עם ההגדרה של package.

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;

```

ההגדרה של package.

```

use work.DualPortRamPackage.all;

```

```

ENTITY DualPortRam IS

```

```

PORT

```

```

(

```

```

    clk    : IN STD_LOGIC;

```

```

    rst    : IN STD_LOGIC;

```

```

    WE     : IN STD_LOGIC;

```

```

    ReadAdd : IN nibble;

```

```

    WriteAdd : IN nibble;

```

```

    Data    : IN byte 0;

```

```

    Dataout  : OUT byte

```



```
);  
  
END;  
  
ARCHITECTURE arc_DualPortRam OF DualPortRam IS  
  
BEGIN  
  
process(clk,rst)  
  
begin  
  
    if rst='1' then  
  
        Dataout<=0;  
  
    elsif clk'event and clk='1' then  
  
        if WE='1' then  
  
            ram(writeAdd)<=data;  
  
        end if;  
  
        Dataout<=ram(ReadAdd);  
  
    end if;  
  
end process;  
  
END;
```

הקוד של הסימולציה הוא:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;


ENTITY DualPortRam_vhd_tst IS

END DualPortRam_vhd_tst;

ARCHITECTURE DualPortRam_arch OF DualPortRam_vhd_tst IS

    SIGNAL clk : STD_LOGIC:='0';

    SIGNAL Data byte;

    SIGNAL Dataout : integer range 255 downto 0;

    SIGNAL ReadAdd : integer range 3 downto 0;

    SIGNAL rst : STD_LOGIC;

    SIGNAL WE : STD_LOGIC;

    SIGNAL WriteAdd : integer range 3 downto 0;

    COMPONENT DualPortRam

        PORT (

            clk          : IN STD_LOGIC:='0';

            rst          : IN STD_LOGIC;

            WE           : IN STD_LOGIC;

            ReadAdd      : IN integer range 3 downto 0;

            WriteAdd     : IN integer range 3 downto 0;

            Data         : IN integer range 255 downto 0;

            Dataout      : OUT integer range 255 downto 0

        );
```

```
END COMPONENT;

BEGIN

    i1 : DualPortRam

    PORT MAP (

        clk => clk,

        Data => Data,

        Dataout => Dataout,

        ReadAdd => ReadAdd,

        rst => rst,

        WE => WE,

        WriteAdd => WriteAdd

    );

    clk<=not clk after 20 ns;

    rst<='1','0' after 40 ns;


    process

    begin

        data<=2;

        WriteAdd<=0;

        we<='1';

        wait for 200 ns;

        data<=4;

        WriteAdd<=1;

        we<='1';
```

```

wait for 200 ns;

data<=6;

WriteAdd<=2;

we<='1';

wait for 200 ns;

we<='0';

ReadAdd<=0;

wait for 200 ns;

we<='0';

ReadAdd<=1;

wait for 200 ns;

we<='0';

ReadAdd<=2;

end process;

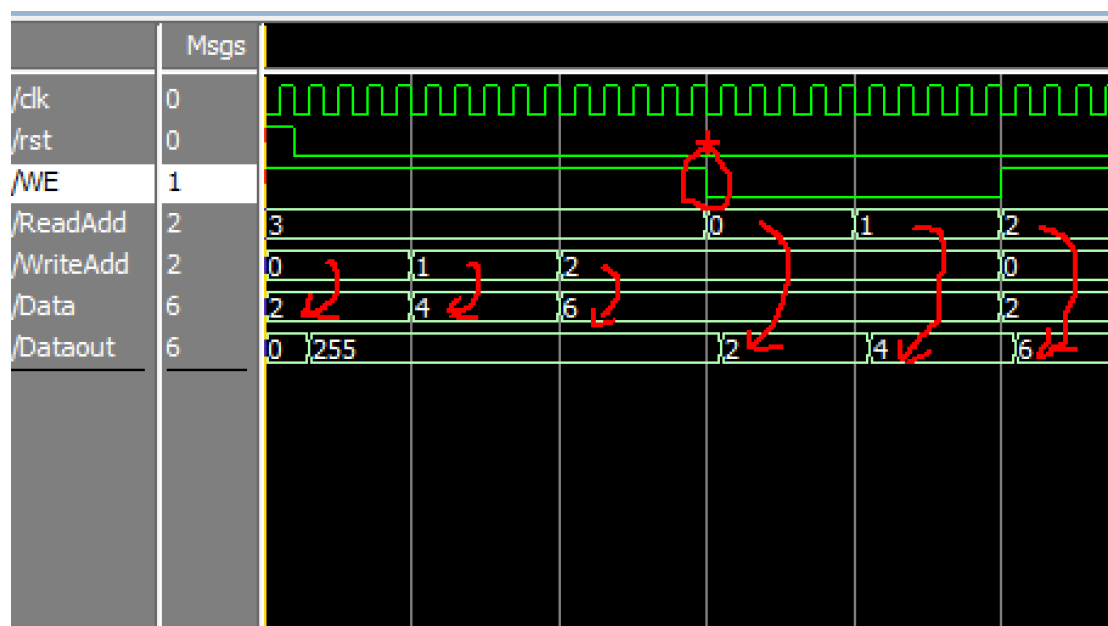
END DualPortRam_arch;

```

אם תתבוננו בקוד של הסימולציה תוכלו להבחין, שצורת ה-process היא אפשרות נוחה כדי לקבוע בכל זמן את המשתנים על-פי הצורך של הרגע.

שלושת החצים האדומים שמופיעים על הסימולציה בצד שמאל, מסמנים כתיבה אל תוך הזיכרון על-פי הכתובות 0,1,2 ובהתאמה עם הנתונים 2,4,6 – כל עוד $WE=1$.

שלושת החצים האדומים בצד ימין של הסימולציה מציינים קריאה מהזיכרון כאשר $WE=0$.



9. פונקציות ופרוצדורות

הפרק הזה מעניין מאוד, מפני שהגישה שלו משלבת בין תכנון חומרה לתכנון תוכנה.

כמו שכבר ציינו, פרויקט נכון הוא פרויקט שמנוהל בצורה נכונה.

מחשבה רבה מושקעת בעניין הזה, כלומר בדרך שבה אפשר לנהל פרויקטים בסדר גודל של אלפי שורות בלי שגיאות.

פונקציות ופרוצדורות מסייעות מאוד בנושאים האלה.

אולם מדובר בנושאים קשים שהם מעין יצור כלאיים, כלומר הם מצריכים חשיבה של תוכנה וחומרה.

בפרק "מבנים מיוחדים" נפרט יותר את הנושא.

יש פונקציות שמתארות מבנה חומרת, ובפונקציות אחרות משתמש המתכנת לצרכיו או לצורך העמסת פונקציות.

הנושא של העמסת פונקציות עוסק בהחלפת התפקיד של האופרטורים. לדוגמה: באופרטור + משתמשים כדי לחבר שני מספרים. אפשר להמיר אותו ולהשתמש בו כדי לחבר בין שני תכנים של פונקציות.

פונקציות לצרכים חומרתיים הן פונקציות מהסוג **synthesizable**, ואפשר להמיר אותן לרכיבים לוגיים. לעומת זאת בפונקציות לצורכי המרה, כגון מבסיס בינארי לבסיס עשרוני, אין סינתזה – **non-synthesizable**.

Procedure היא פקודה עם פונקציה, שנועדה כדי לבצע רצף של פעולות בדומה לאובייקטים משפת `cpp`.

פונקציות ופרוצדורות נקראות תתי-תוכניות (**subtype**).

הרחבה של package

לפני שנגדיר מהן פונקציות ופרוצדורות, נרחיב את נושא ה-`package`.

בפרק על מערכים וזיכרונות הכרנו את תרומתו של `package` לצורכי ייעול הכתיבה.

אפשר לפתח את הנושא הזה עוד יותר בעזרת פונקציות ופרוצדורות.

שפת `vhdl` מאפשרת לנו לכתוב קוד קומבינטורי ב-`package`.

כמו ב-entity וב-architecture גם כאן אפשר להגדיר את המשתנים ואת גוף ה-package על המבנה הבא:

נוסף על החלק ההצהרתי **package declaration** נצרך גם את **package body**.

```
Package identifier is
```

```
Package declarative
```

```
End package identifier
```

```
Package body identifier is
```

```
Package body declarative
```

```
End package body identifier
```

ראו את הדוגמה הבאה לפונקציה, נעסוק בה בהמשך.

```
library ieee;
```

```
use ieee.std_logic_1164.all;
```

הכרזה על ה-package בשם `BaseFunctionPackage`:

```
package BaseFunctionPackage is
```

הצהרה על הפונקציה:

```
function compare(a,b :integer)return integer;
```

סיום הצהרת ה-package:

```
end package;
```

לגוף ה-package body יש שם זהה כמו להצהרה:

```
package body BaseFunctionPackage is
```

גוף הפונקציה:

```
function compare(a,b :integer) return integer is
begin
if a>b then
    return a;
else
    return b;
end if;
```

סיום הפונקציה:

```
end function compare;
```

סיום ה-package body:

```
end package body;
```

function פונקציות

הפונקציות האלה דומות במהות שלהן לפונקציה ב-cpp או בכל שפה עילית אחרת.

ההגדרה שלהן כוללת כניסות ויציאות, וכמובן את שם הפונקציה.

הן עוסקות בעיקרן בתוצאת חישוב אחת בלבד.

ההגדרה באנגלית מבטאת יפה את הכוונה:

The function encapsulates a collection of statements that compute result.

מבנה הפונקציה:

```
Function identifier
Begin
Sequential statement
End function
```


כמו בשפה עילית צריך להכריז על הפונקציה, אבל יש גם את גוף הפונקציה. גם כאן ההכרזה מתבצעת באמצעות package declarative, אך גוף הפונקציה נמצא ב-package body. ראו את הכיתוב הנלווה לדוגמה הבאה.

הכרזה על הפונקציה:

```
function compare(a,b :integer)return integer;
```

הפונקציה כוללת את הכניסות a,b מהסוג integer ואת ה-return – הערך המוחזר מהפונקציה.

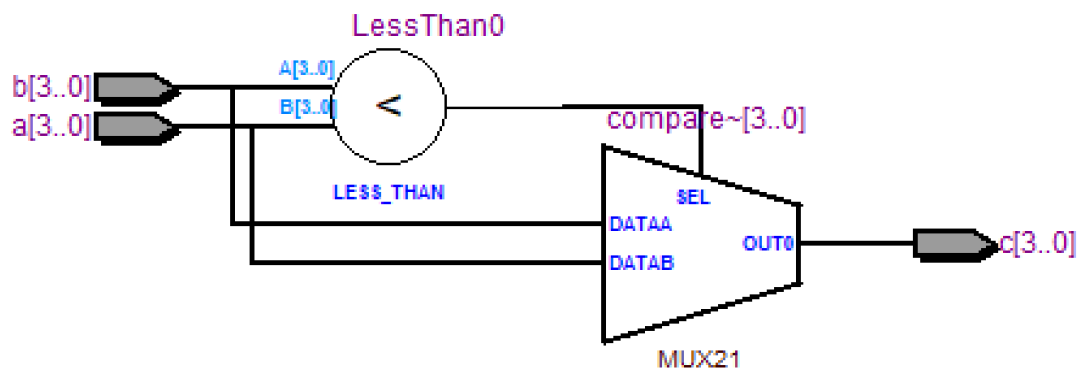
אם מטרת הפונקציה היא סינתזה הערך המוחזר הוא יציאת המעגל, ואם לא הערך מוחזר כמו בפונקציה רגילה.

הפונקציה מוצאת את המספר המקסימלי בין שני ערכים a,b:

```
function compare(a,b :integer)return integer is
begin
if a>b then
    return a;
else
    return b;
end if;
```

זה תיאור חומרתי, ברירה בין שני מצבים (רֶבֶב – multiplexer).

כפי שאפשר לראות בציור הבא:



ה-lessThan הוא משווה שהתוצאה שלו בוררת בין הכניסות a,b.

כל הקוד "מסתתר" ב- package, ולתוכנית הראשית entity נשאר להציג רק פעולה אחת. ראו את הקוד הבא:

```
use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;

use ieee.std_logic_unsigned.all;
```

הכרזה על package בקוד הראשי:

```
use work.baseFunctionPackage.all;
```

```

entity BaseFunction is
port (
    a :in integer range 15 downto 0;
    b :in integer range 15 downto 0;
    c :out integer range 15 downto 0
);
end;

architecture arc_BaseFunction of BaseFunction is
begin
process(a,b)
begin
c<=compare(a,b);
end process;
end;

```

אמנם זה קוד די פשוט, אבל הרעיון הוא יצירה של אופרטור חדש.

כל הקוד מסתכם בהוראה אחת בלבד, שנקראת compare, ותפקידה להשוות בין שני גדלים.

נניח שאנחנו רוצים להוסיף עוד פונקציה על הפונקציה הקיימת.

הדבר פשוט מאוד. כל מה שצריך לעשות זה להוסיף ל-package את הפונקציה ולייצג אותה בקוד הראשי.

שינויי הקוד הנדרשים:

```
library ieee;  
  
use ieee.std_logic_1164.all;  
  
  
package BaseFunctionPackage is  
  
  
function compare(a,b :integer)return integer;
```

הפונקציה החדשה:

```

function CmpBool(a,b :integer) return boolean;

end package;

package body BaseFunctionPackage is

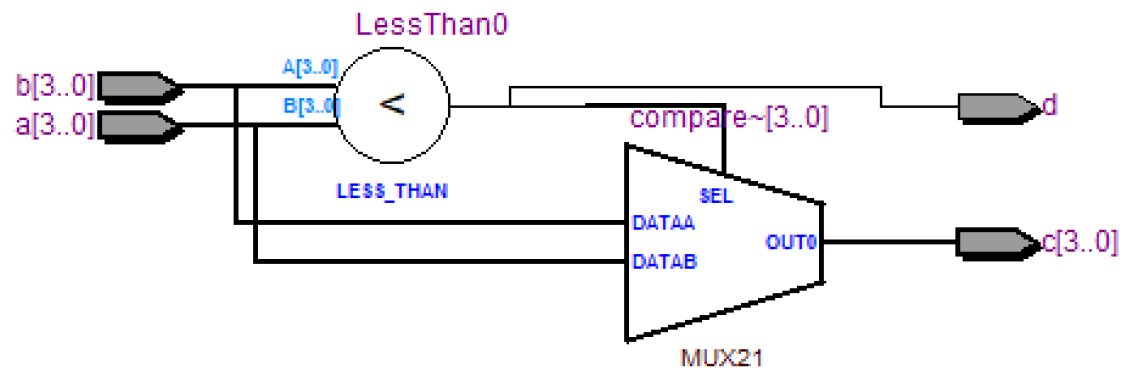
function compare(a,b :integer)return integer is
begin
    if a>b then
        return a;
    else
        return b;
    end if;
end function compare;

----compare boolean

function CmpBool(a,b :integer) return boolean is
begin
    if a>b then
        return true;
    else
        return false;
    end if;
end function CmpBool;

end package body;
```

פעולת ההחזר היא בוליאנית:



השינוי בשרטוט החשמלי כל-כך הגיוני. נכון היציאה d נובע מהמשוואה $LessThan$.

קוד הסימולציה הוא:

```
LIBRARY ieee;

USE ieee.std_logic_1164.all;


ENTITY BaseFunction_vhd_tst IS

END BaseFunction_vhd_tst;

ARCHITECTURE BaseFunction_arch OF BaseFunction_vhd_tst IS

    SIGNAL a : integer range 15 downto 0;

    SIGNAL b : integer range 15 downto 0;

    SIGNAL c : integer range 15 downto 0;

    SIGNAL d : boolean;

    COMPONENT BaseFunction

        PORT (

            a : IN integer range 15 downto 0;

            b : IN integer range 15 downto 0;

            c : OUT integer range 15 downto 0;

            d : OUT boolean

        );

    END COMPONENT;

BEGIN

    i1 : BaseFunction

    PORT MAP (

-- list connections between master ports and signals

        a => a,

        b => b,

        c => c,

        d => d

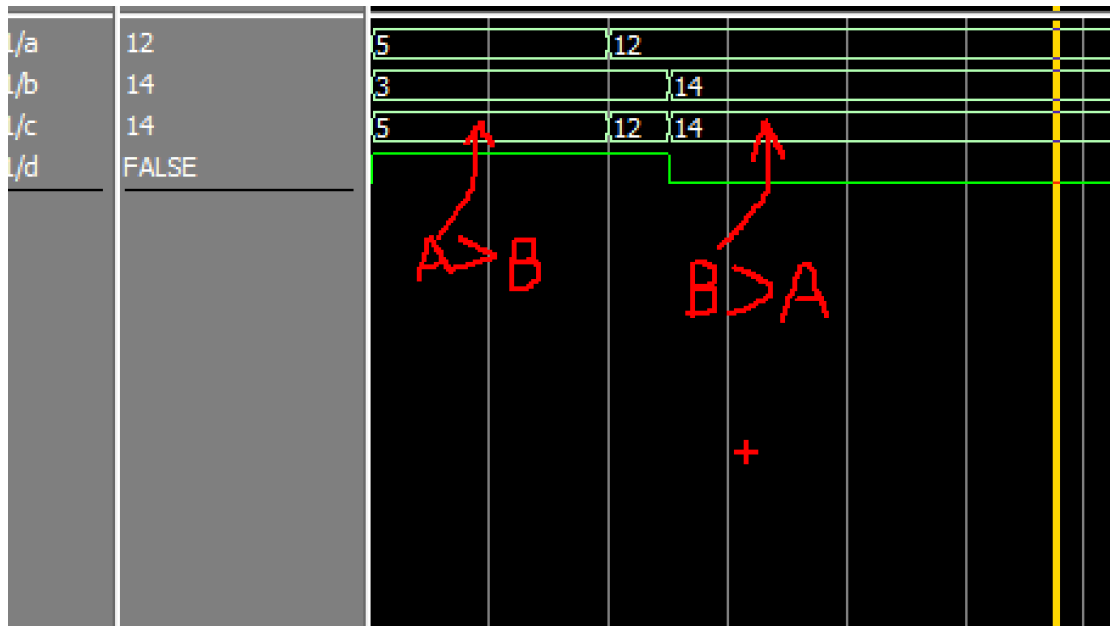
    );
```

```
a<= 5,12 after 200 ns;

b<= 3,14 after 250 ns;

END BaseFunction_arch;
```

התוצאה של הסימולציה:



Procedure פרוצדורה

הפרוצדורה היא כמוסה ובה אוסף של הצהרות רציפות, והתפוקה שהיא מניבה (פלט) משפיעה על היציאות.

The procedure encapsulates a collection of sequential statements that are executed for their effect.

מבנה הפרוצדורה:

```
Procedure identifier(parameter interface list) is
Begin
End procedure;
```

לפרוצדורה יש כניסות ויציאות. בדומה לפונקציות ב-cpp, אפשר ליישם את הפרוצדורה בצירוף ההכרזה או להפריד את ההכרזה ממקום הביצוע.

Procedure parameter

הכללת פרמטרים מאפשרת לנו לבצע אלגוריתמים על נתונים בכל מיני זמנים.

כמו אצל הפונקציות, בקריאה לפרוצדורה נכללים הפרמטרים שאותם היא נדרשת לבצע. הפרוצדורה מכילה כניסות ויציאות.

הכניסות הן מסוג in, inout, והיציאות הן מסוג out.

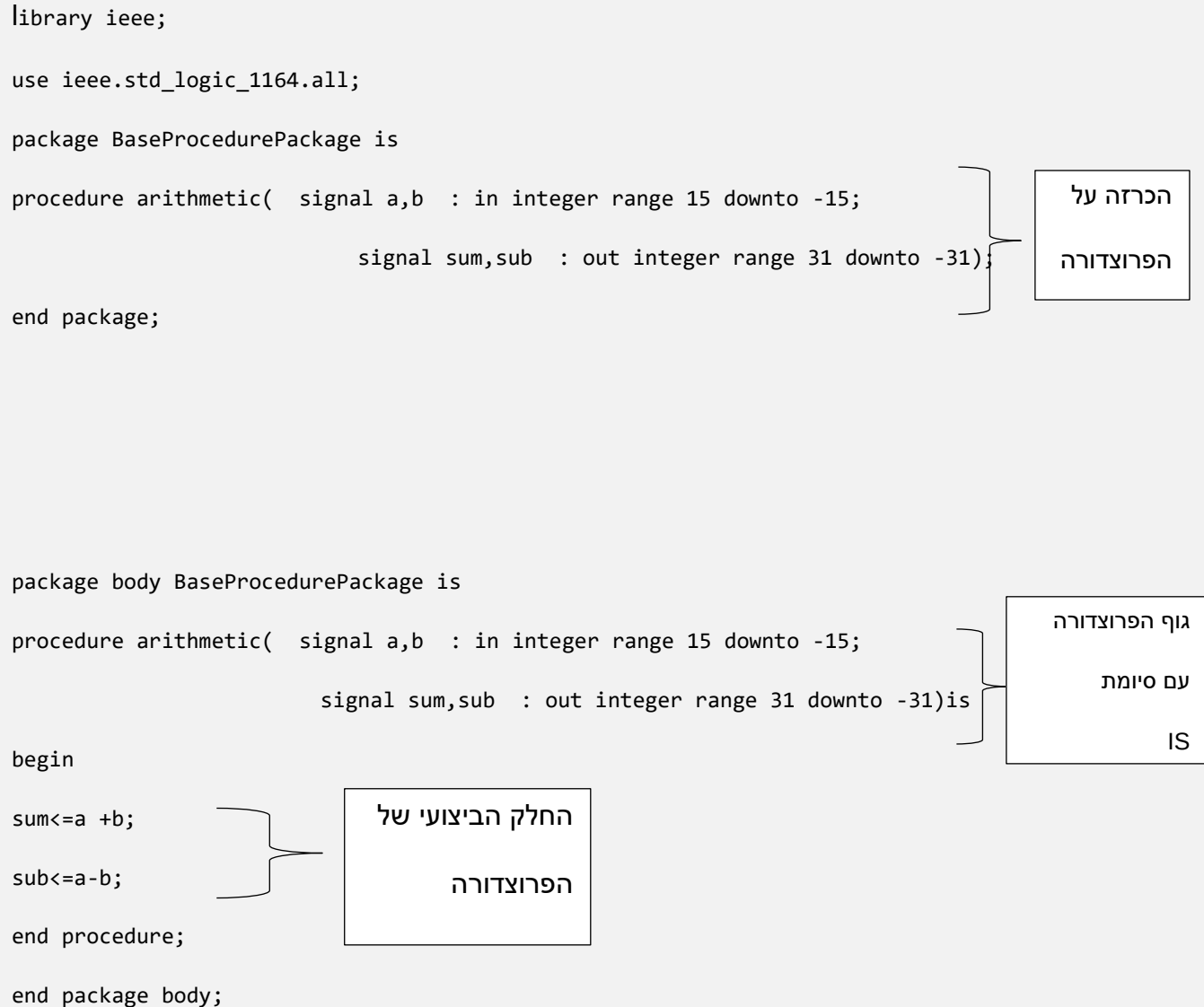
כפי שנראה מאוחר יותר ההגדרות האלה דומות למבנה של פרויקט entity.

בחלק ההצהרתי מוגדרות הכניסות והיציאות, חלק זה נקרא port.

החלק האחר הוא החלק הביצועי, כמו ב-architecture.

הדוגמה הבאה תמחיש את ההבדל בין הפרוצדורה לפונקציה:

הגדרת הפרוצדורה על-ידי package



כפי שכבר הזכרנו, מקום ההכרזה של הפרוצדורה שונה ממקום הביצוע – בדומה לפונקציות בשפת c++ או בשפות אחרות. תוכלו לראות את זה אם תשוו את ההגדרות של הכניסות ושל היציאות לאובייקטים שמאפשרים להחזיר יותר מערך אחד בודד. מצד אחד פרוצדורה מאפשרת לנו לתפוס חומרה שיש לה כניסות וגם יציאות, ומהצד האחר אפשר לחשוב גם בסגנון של תוכנה. במקום ביצוע חוזר באמצעות מונים, שהם מבנה חומרתי מובהק, אפשר להשתמש בלולאות כדי לחזור שוב על פעולות.

את הנושא הזה נרחיב בהמשך הספר כשנלמד על מבנים מיוחדים.

עכשיו נתמקד בקריאה לפרוצדורה.

קריאה לפרוצדורה מקוד ראשי:

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.std_logic_arith.all;

use ieee.std_logic_unsigned.all;

use work.BaseProcedurePackage.all;
```

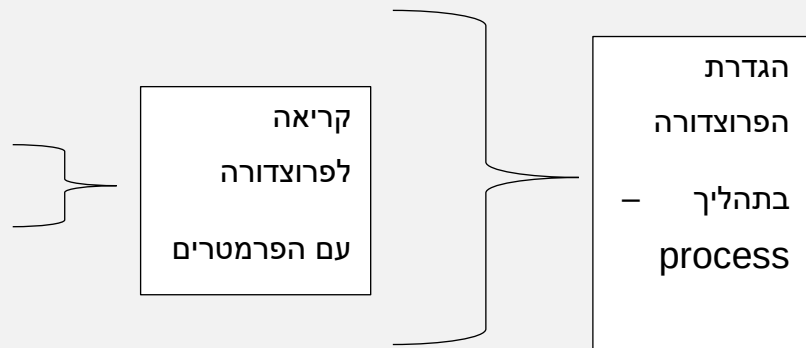


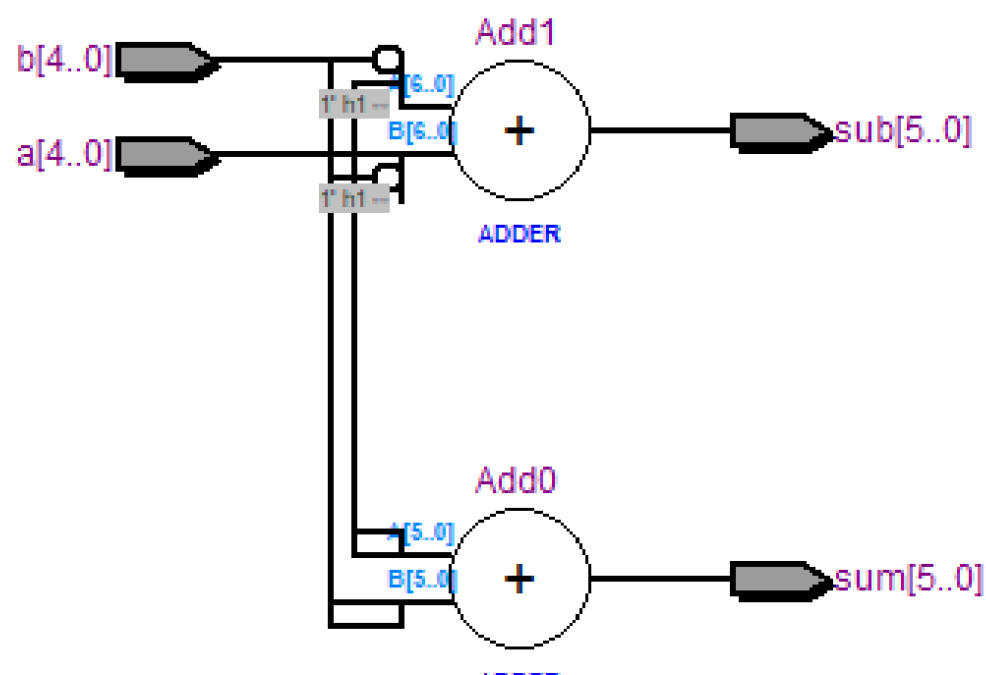
```
entity BaseProcedure is
port (
    a :in integer range 15 downto -15;
    b :in integer range 15 downto -15;
    sum :out integer range 31 downto -31;
    sub :out integer range 31 downto -31
);
end;
```

```
architecture arc_BaseProcedure of BaseProcedure is
begin
process(a,b)
```

```
begin
arithmetic(a,b,sum,sub);
end process;
```

```
end;
```





בשרטוט מתוארים המחברים באמצעות add1, add2. מובן שהחיסור נעשה בעזרת הערך שמשלים למספר שתיים.

סימולציה:

Msgs						
/a	7	3	7			
/b	2	5	2			
/sub	5	4	5			
/sum	9	8	9			

בסימולציה הזאת אפשר לראות את פעולות החשבון עם התוצאה השלילית.

10. Simulation – סימולציה

תהליך של סימולציה נועד כדי לדמות מציאות במחשב. לפעמים בנייה של מערכת מורכבת ויקרה כל-כך לא מבטיחה תוצאות רצויות במיוחד, שכן מלכתחילה תמיד קיים הבדל בין התכנון ליישום. במשך שנים פותחו כלי סימולציה שמדמים כל מיני מציאויות.

סימולציה היא ניסיון לתאר מציאות, ובאמצעותה אפשר לנבא תוצאה אפשרית. לפיכך חשוב מאוד להכיר את המטרה שגדרש מאתנו לדמות, ובנוסף יש לדעת איך נזהה תוצאה בלתי-רצויה.

מצביאים נהגו להשתמש בעבר בסימולציות כדי לתכנן קרבות או כדי לבחון תרחישים אפשריים, וזה נהוג אף כיום.

סימולציה טובה מכסה את כל האפשרויות – אם לא, האויב יכול להפתיע. ובכל זאת יש למציאות דמיון רב, יותר מכל דמיון אנושי. במטרה ידועה או בסימולציה אפשר להתקרב למציאות רק במידה מסוימת, לעולם לא יהיה אפשר להגיע למציאות שלמה.

סימולציה שקרובה מאוד למציאות תתרום רבות ללא ספק לקיצור זמן הפיתוח של הפרויקט, ולכן נראה שיש בסביבה של FPGA יתרון על-פני סביבות אחרות מקבילות.

לצורך המחשה נניח שאנחנו רוצים לפתח מד-מרחק על-פי חיישן מרחק כדוגמת SRF04 (החיישן הזה מספק אות ריבועי לפי הפונקציה של מרחק העצם מהחיישן). אפשר לדמות בקלות רבה למדי את פונקציית המרחק ולבדוק את התכנון של מה שנועד להיות צרוב על רכיב ה-FPGA.

מכל האמור אפשר להבין שסביבת הסימולציה היא במחשב PC בלי שום קשר לרכיב הנצרב FPGA. ליתר דיוק אפשר להתחשב גם בסביבה הזאת ובהשפעותיה על התכנון, אולם כרגע מוקדם מדי לטפל בנושאים מורכבים שכאלה.

ראשית יש להבחין בין סימולציה להדמיה.

סימולציה: קוד מחשב שמדמה נתונים של המציאות הנבדקת, ובאמצעותו נבדק קוד התכנון שאותו צריך ליישם.

הדמיה: בתהליך הזה משוחזרת מציאות לא ידועה על-סמך חישובים מתמטיים, כמו במכשיר M.R.I. האתגר בהדמיה הוא לגלות את המציאות – לעומת הסימולציה, שמנסה לשחזר את המציאות.

נראה דוגמה נוספת מתחום הבקרה, שתמחיש לנו את ההדמיה.

ניתוח פונקציית תמסורת בלי לבנות אפילו מרכיב אחד מהמערכת, ואף יותר מזה – הרצת הסימולציה בתוכנה כגון MATLAB, חוסכים לנו בבניית המערכת ויותר מכול מקצרים את זמן הפיתוח.

מכיוון שהסימולציה פועלת על מכשיר ה-PC ולא צריך לצרוב אותה על הרכיב, שפת VHDL ושפת VRILOG מעניקות לה תחביר גמיש מאוד, והוא שונה בתכלית משפת VHDL שאכן זקוקה לצריבה.

הכרה של סביבת העבודה היא תנאי הכרחי כדי לרכוש ביטחון ברכיבים מתוכננים (F.P.G.A.). לכן יש לתרגל כמה שאפשר.

בפרק הזה נכיר את ה-test bench. יש בו איורים ברורים, שתפקידם הוא להקטין במידת האפשר את האפשרות לחוסר הצלחה.

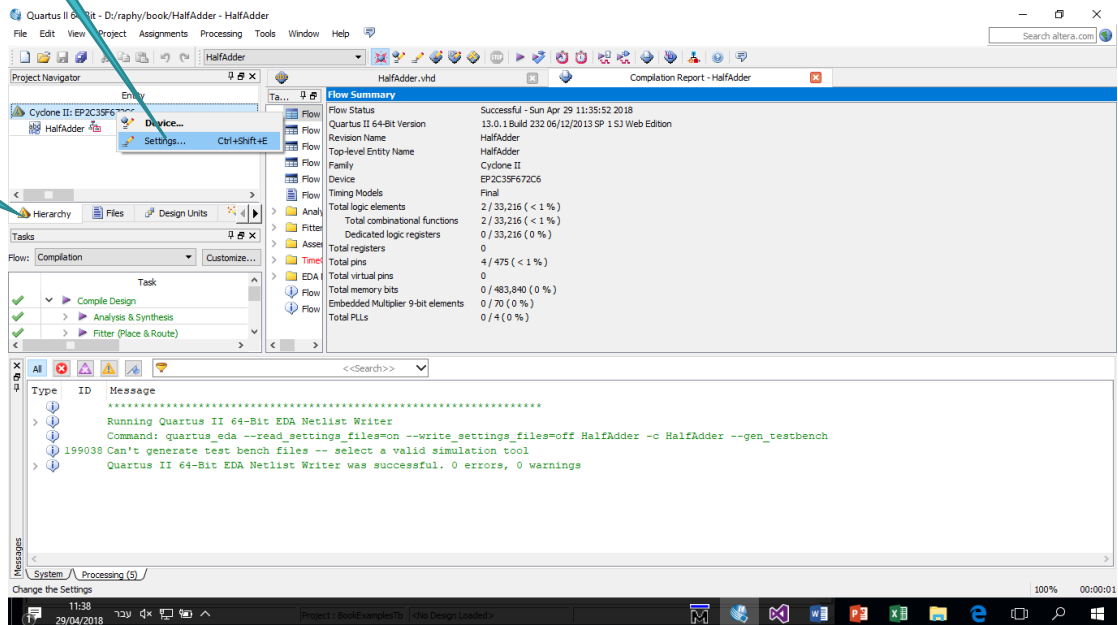
ברוכים הבאים לעולם יפה ומרתק!

1. בשורת החיפוש הקלידו Modelsim.

יצירת קובץ סימולציה בסביבת quartus:

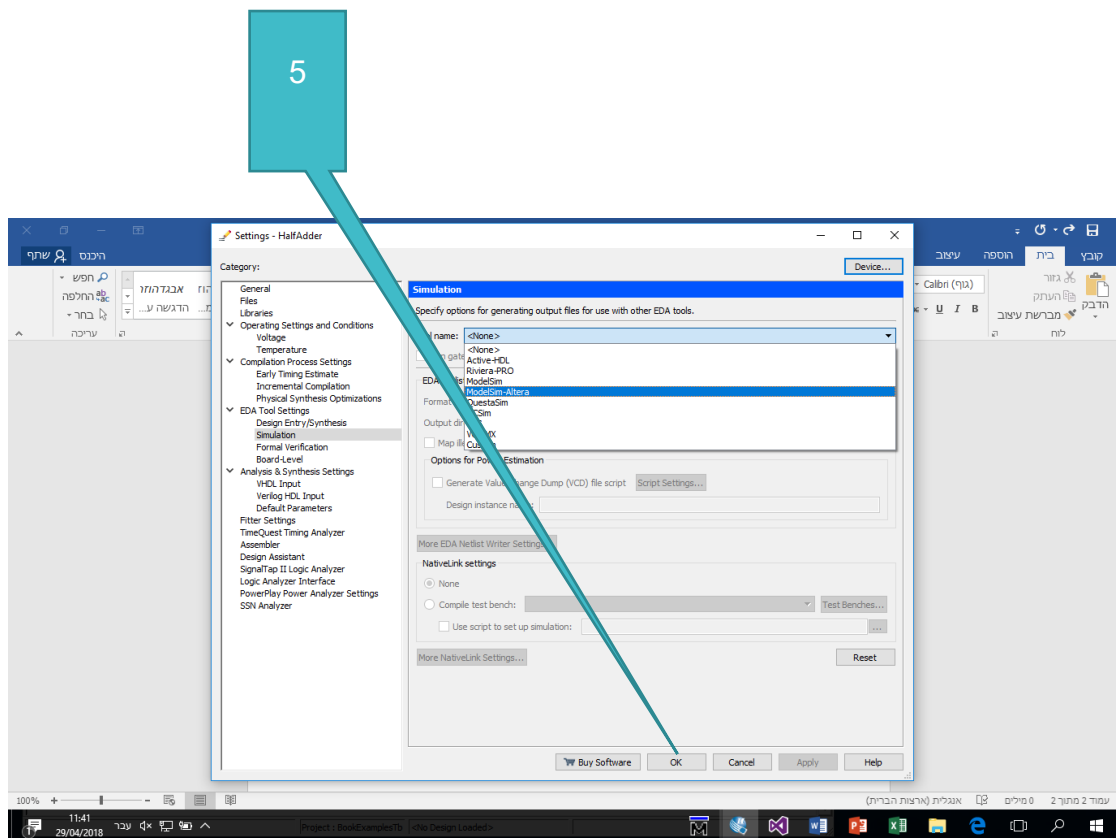
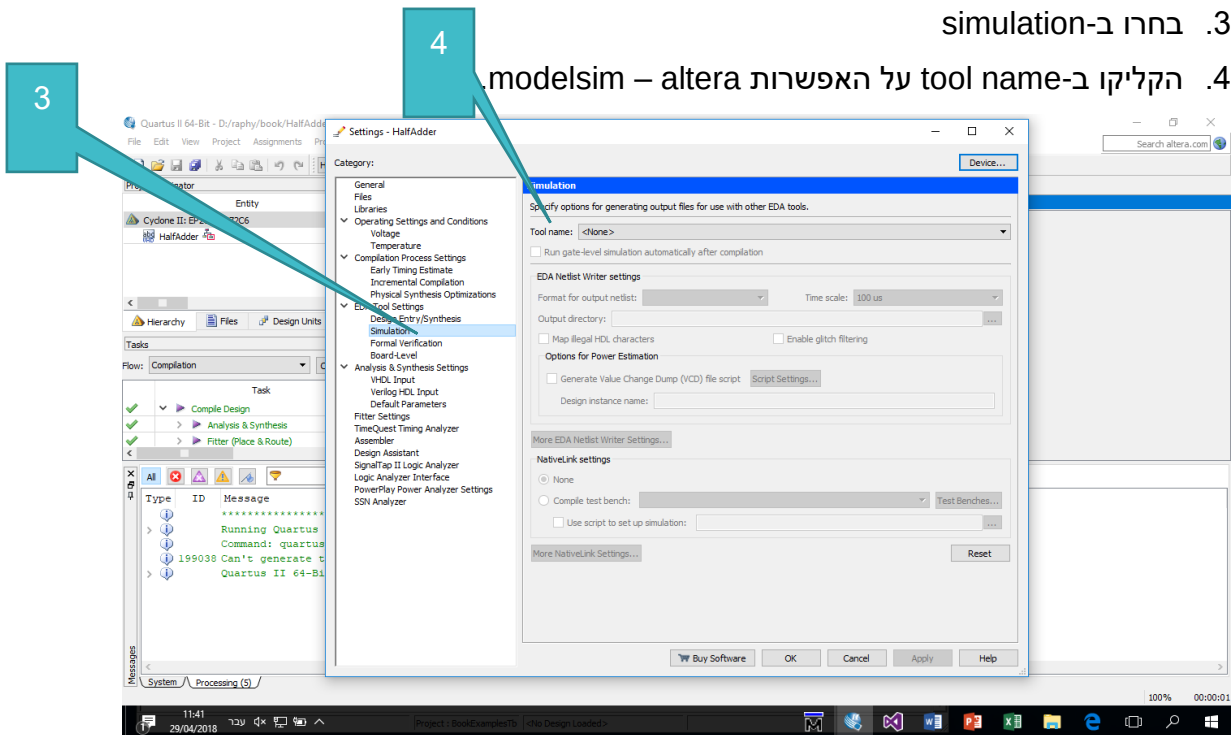
1. הקישו על הלשונית hierarchy.

2. הקליקו על העכבר בצד ימין ובחרו ב-setting.



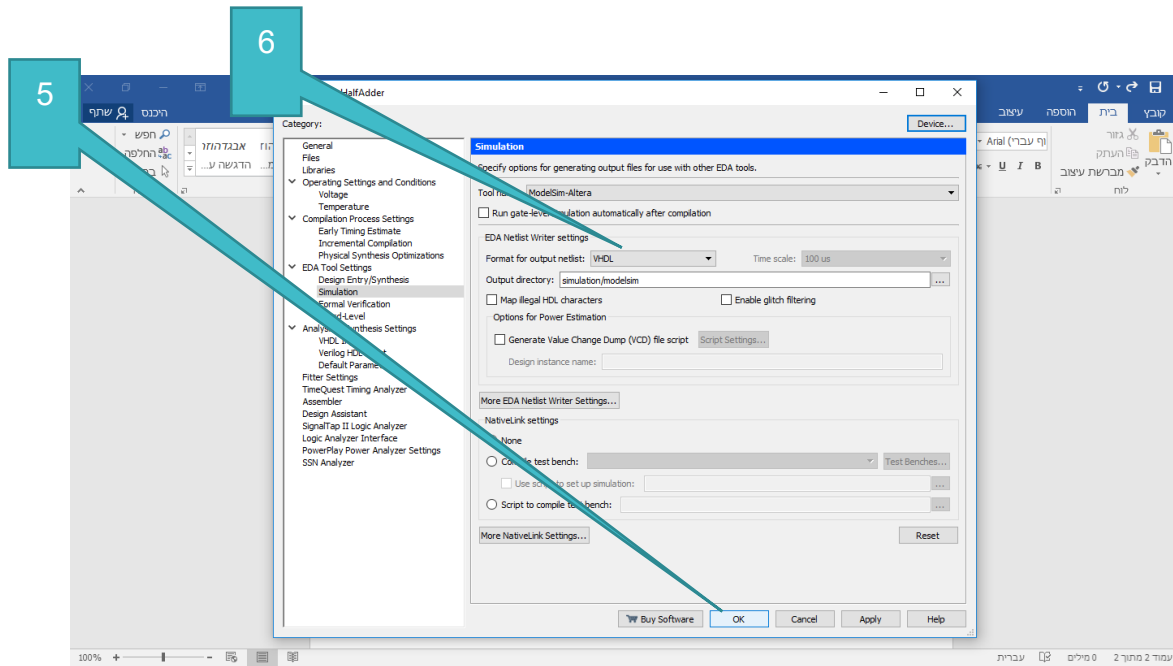
3. בחרו ב-simulation

4. הקליקו ב-tool name על האפשרות altera – modelsim

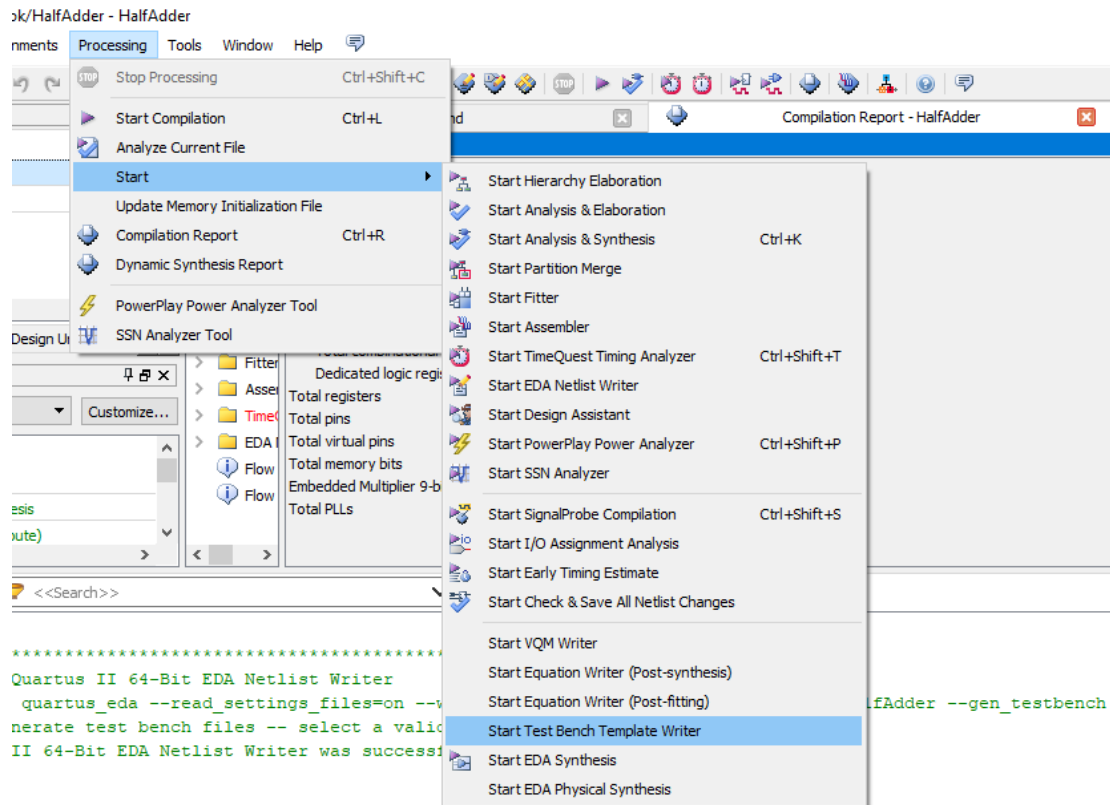


5. הקליקו O.K.

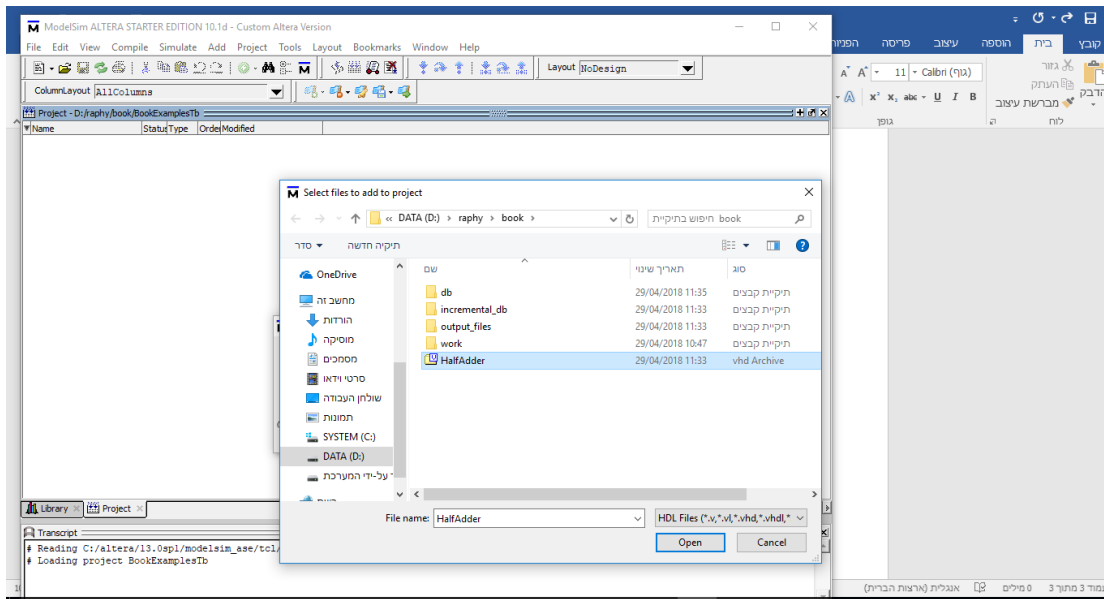
6. בחרו ב-vhdl.



7. מתוך האפשרויות בלשונית processing שבתפריט בחרו ב-start, לאחר מכן בחרו ב-.start test bench template writer



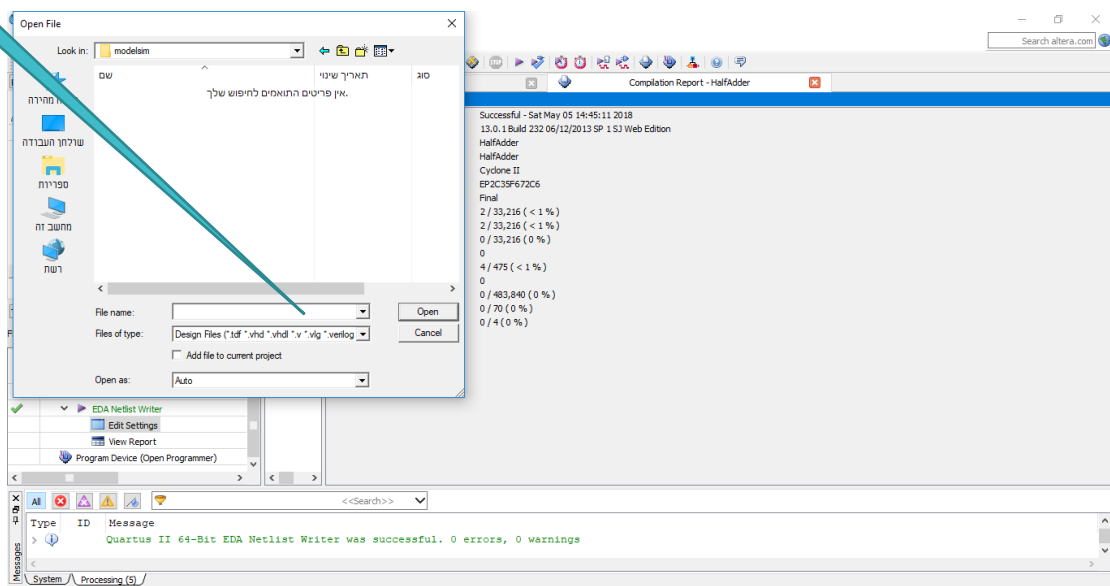
8. בתום ההרצה בחרו מהתפריט file את open, ולאחר מכן פתחו מספריית simulation את ספריית modelsim.
- File=>open=>simulation=>modelsim



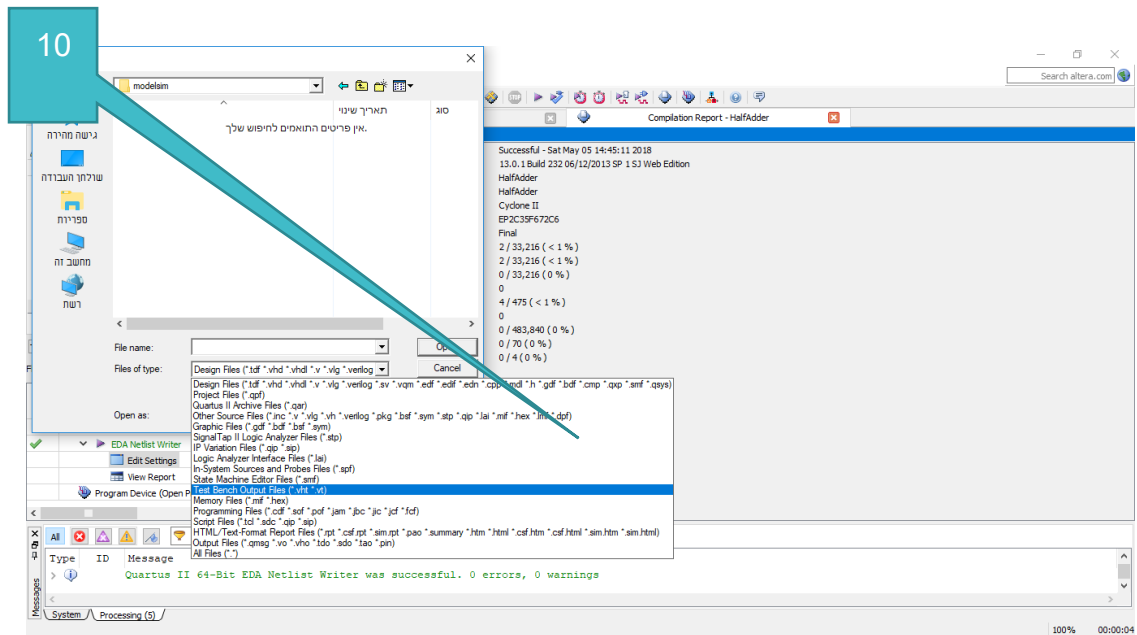
בספרייה modelsim לא תמצאו כלום, לא לדאוג.

9. הקליקו על files of types ובחרו באפשרות vht.

9

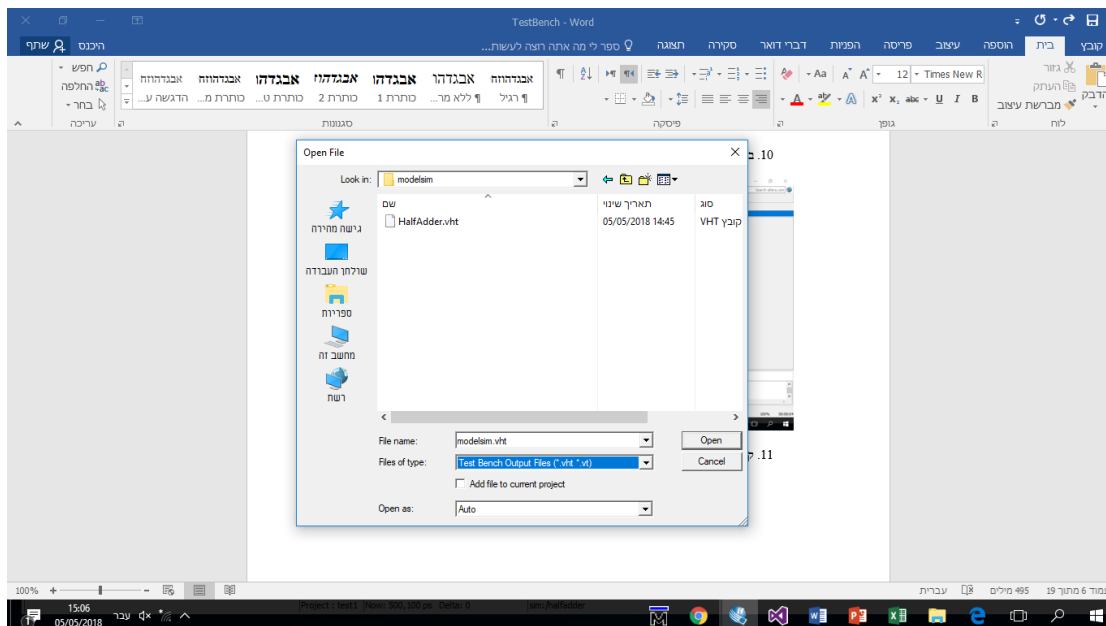


10. בחירת הסוג של קובץ ה-vht מתוך התפריט.



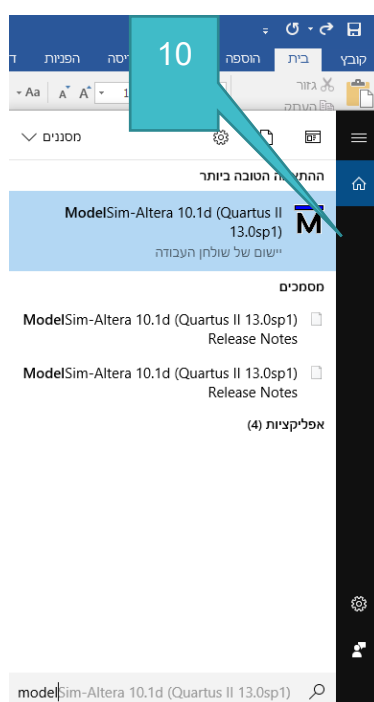
11. קובץ עם סיומת של vht אמור להופיע.

HalfAdder.vht

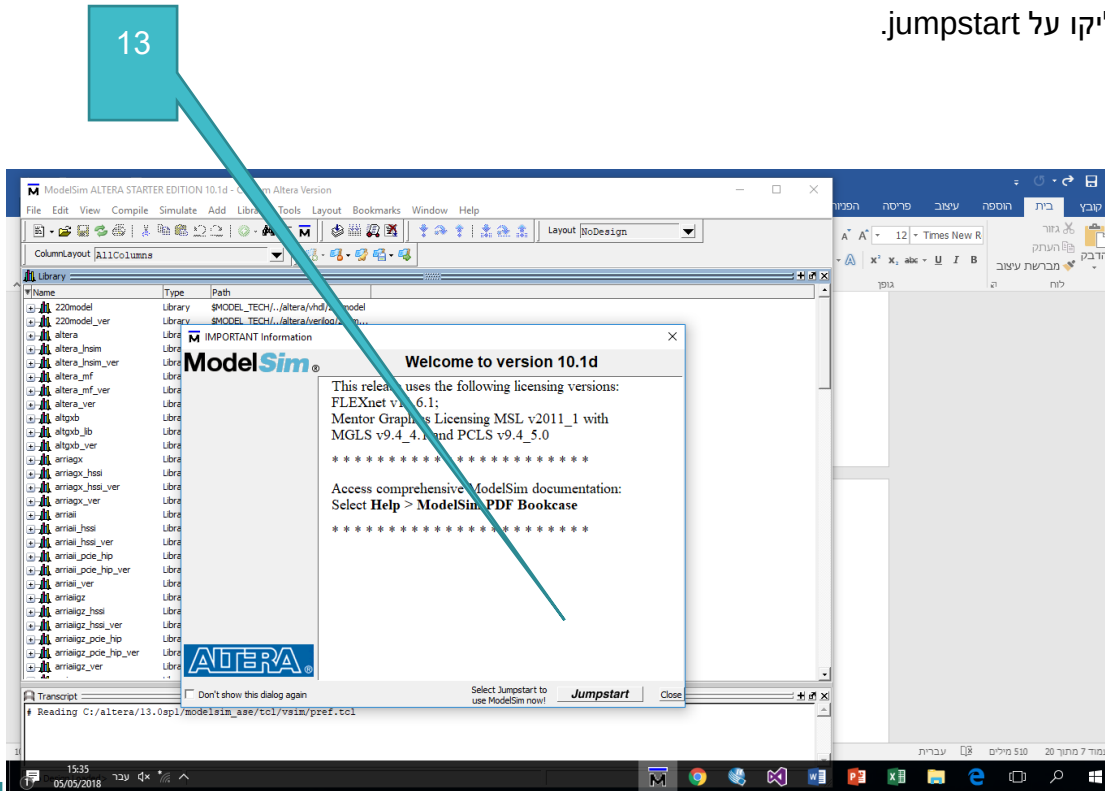


סביבת הסימולציה:

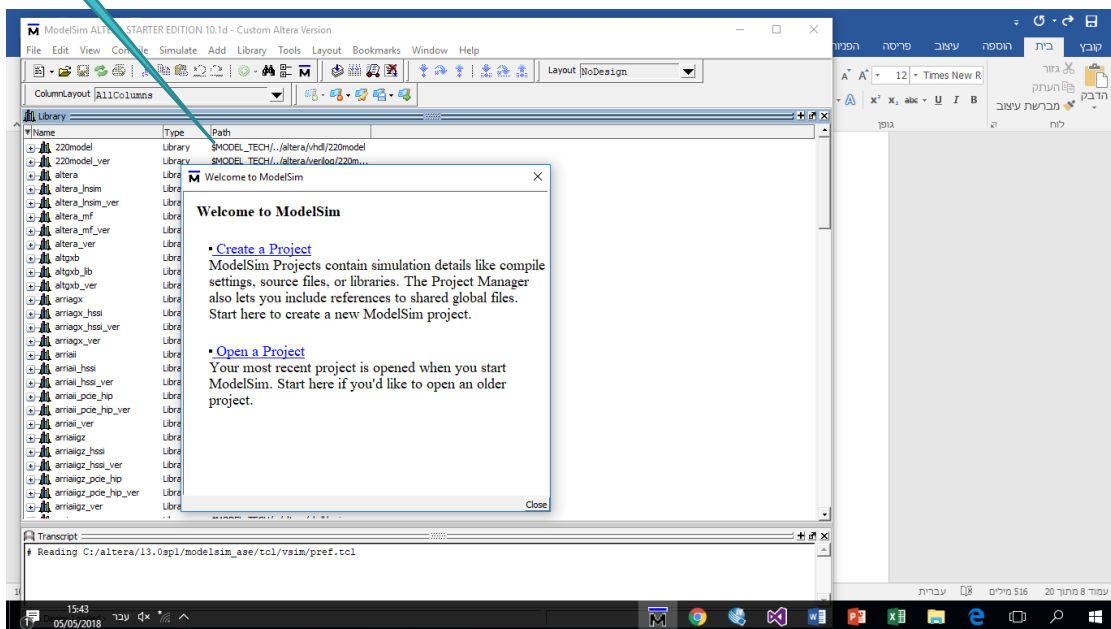
12. באמצעות מנוע החיפוש (search) חפשו את modelsim והקליקו על הצלמית שלו.



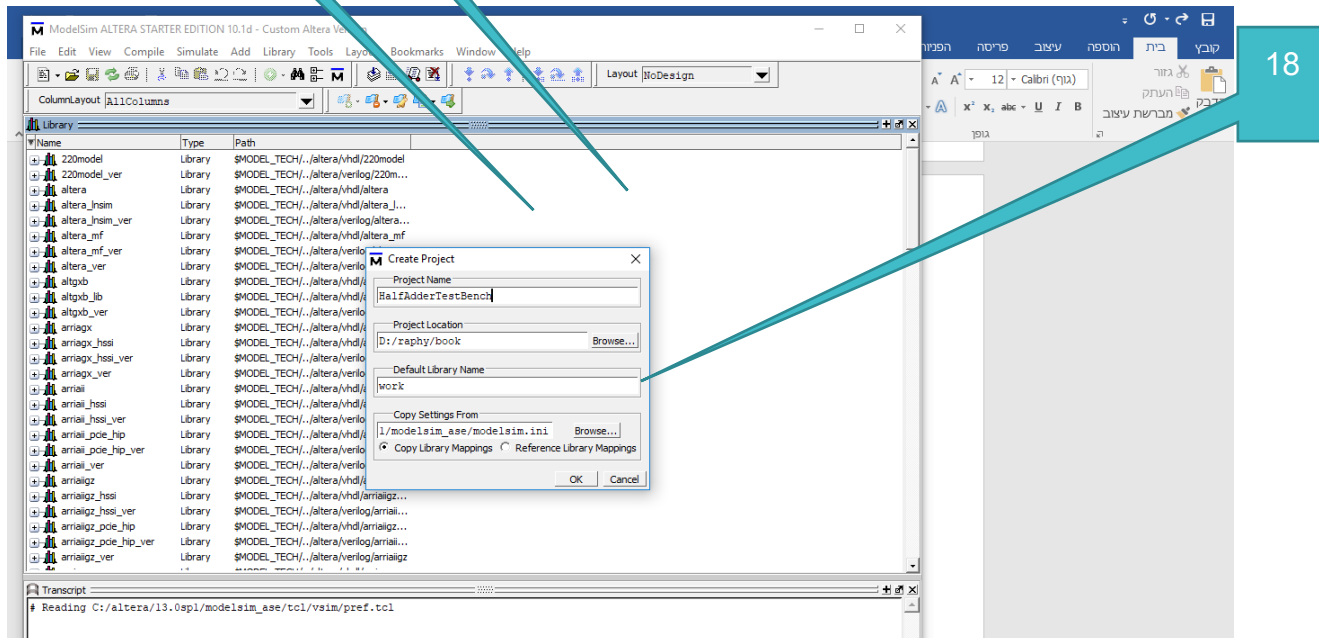
13. הקליקו על jumpstart.



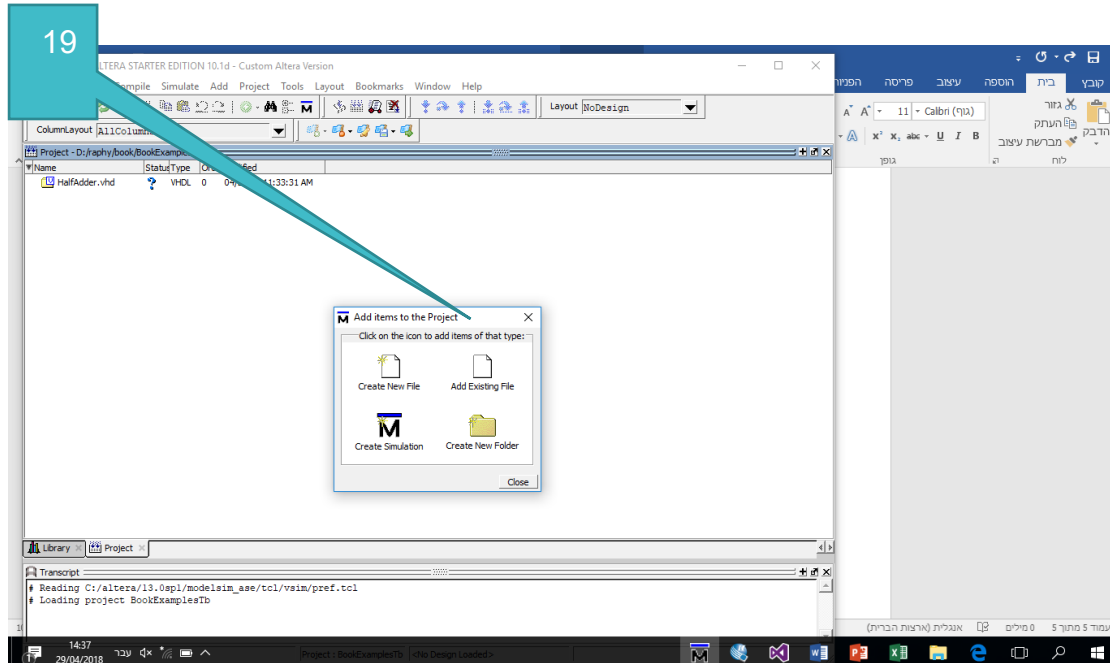
14. Create a project על הקליקו.



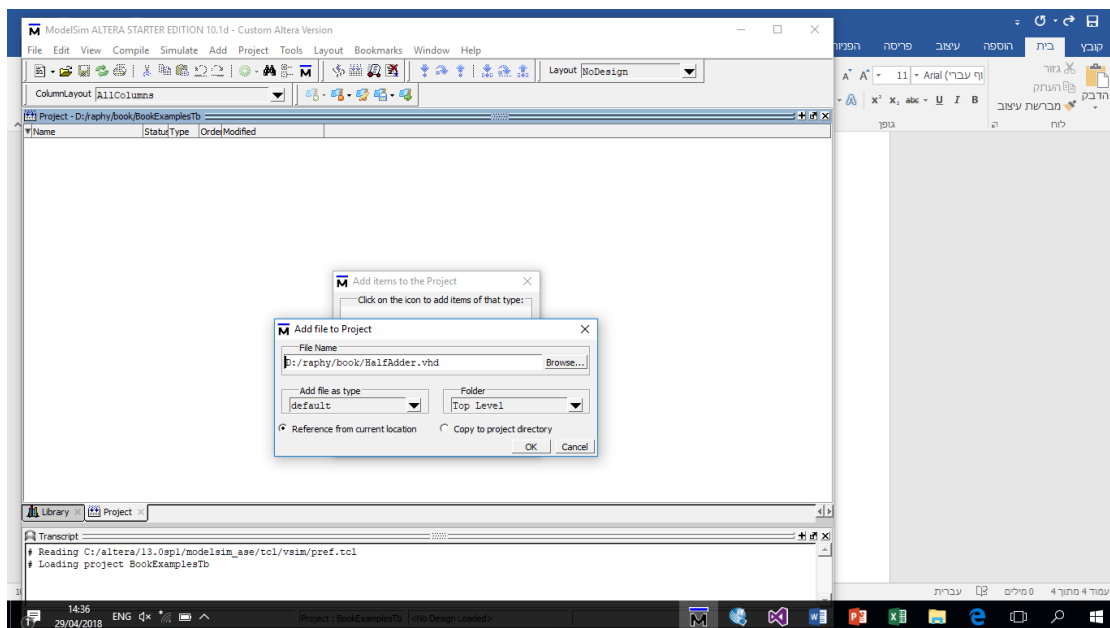
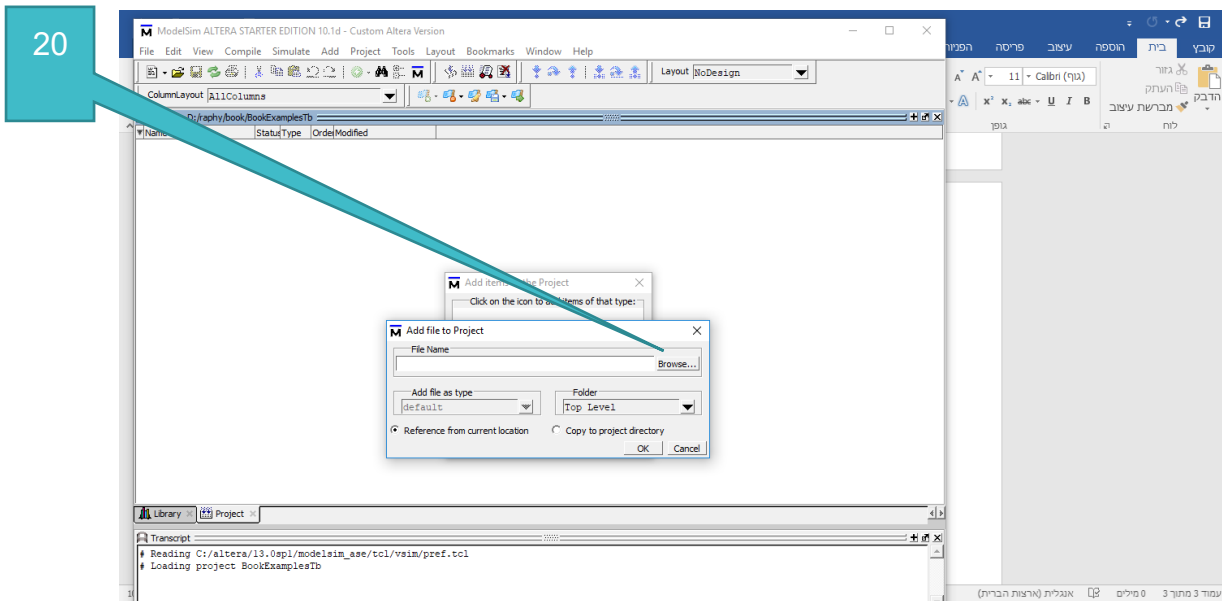
16. בחרו את המסלול שבו פתחתם את הפרויקט ב-quartus.
17. לצורך מעקב נקבע את שם הפרויקט: HalfAdderTB.
18. ספריית work נפתחת בתור ברירת מחדל, אין לשנות אותה.



19. הקליקו על Add Existing Files והגיעו ל-HalfAdder.vhd וגם ל-HalfAdder.vht לפי הסעיפים שלעיל (מסעיף 10 והלאה).

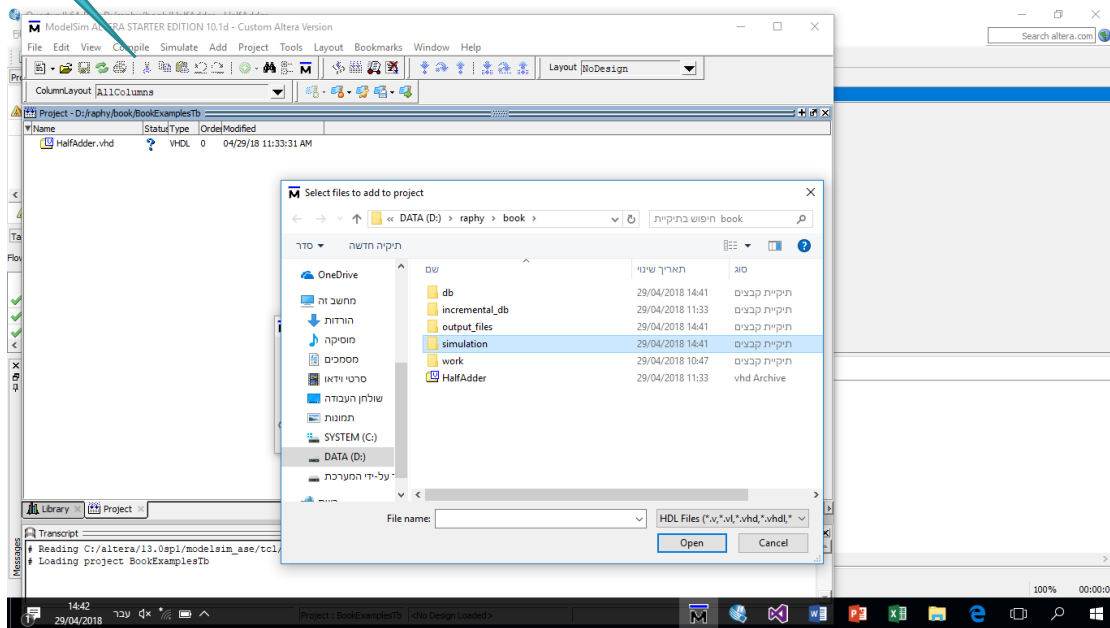


20. דפדפו למיקום הנכון של הקבצים באמצעות .browse.

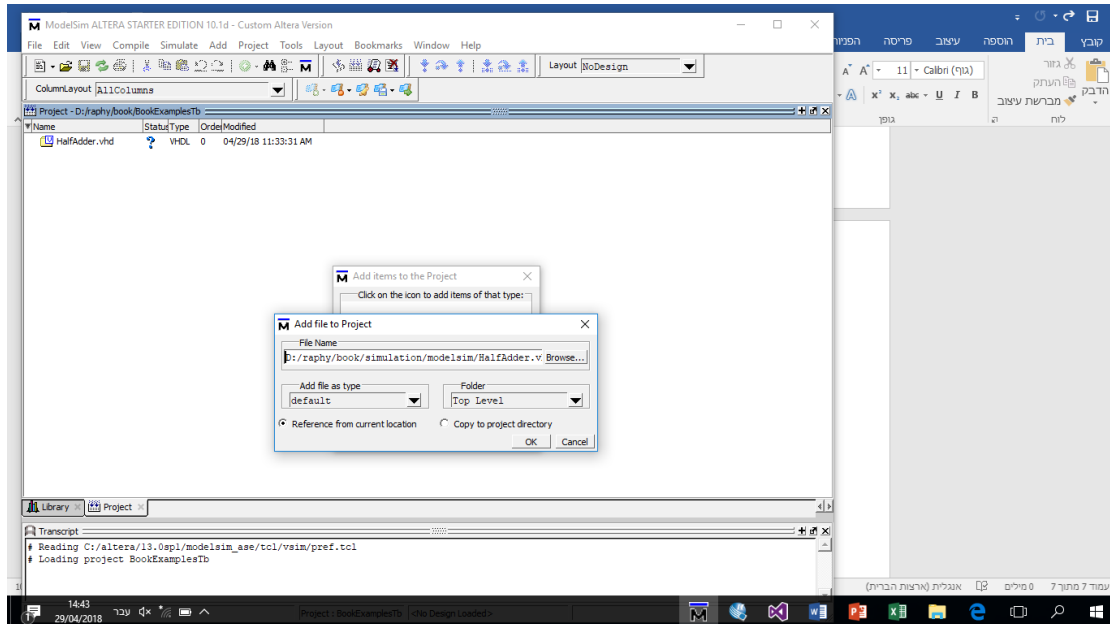


21

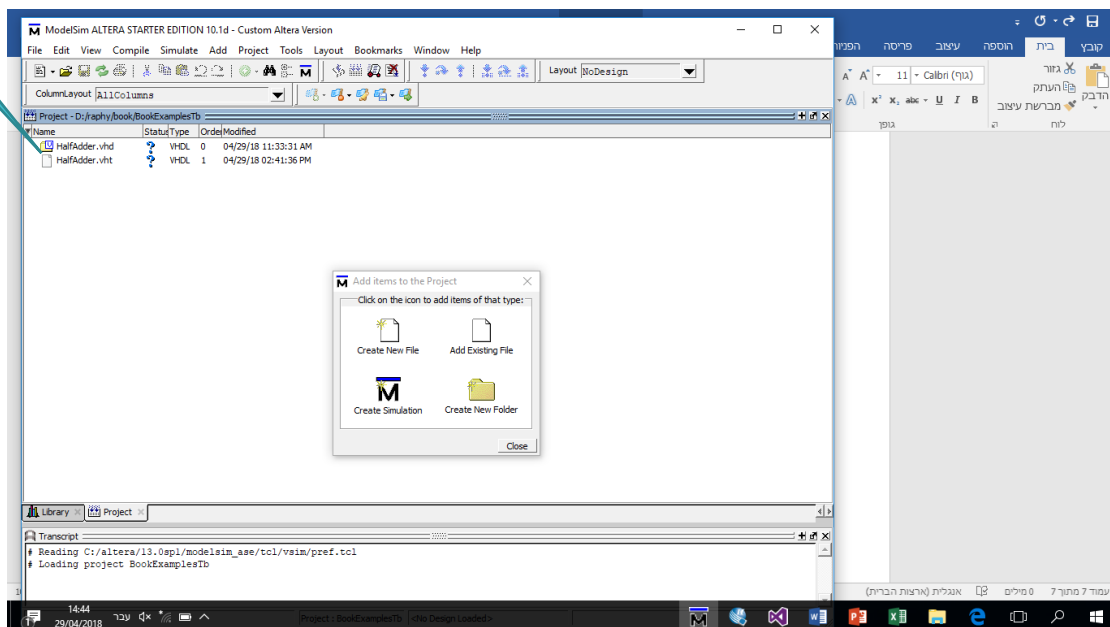
21. הוספה של קובץ vht מספריית simulation.



22. הוספה של קובץ vhd.



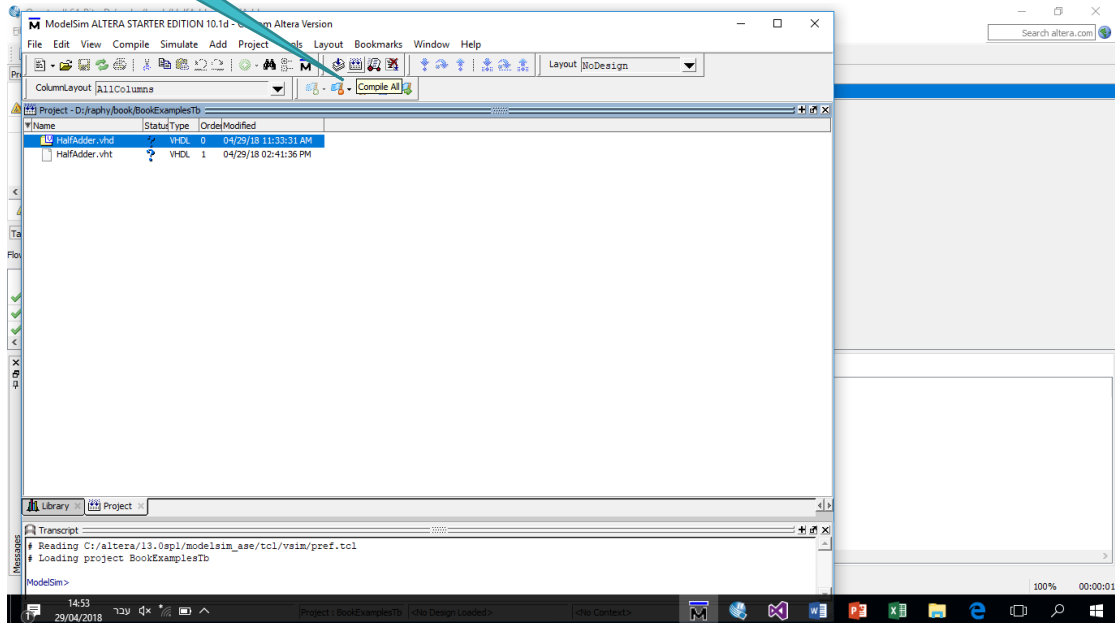
23



23. סמנו אחד מהקבצים באמצעות קליק.

24. הקליקו על compile all. העמידו את העכבר על הצלמית כדי לזהות את הצלמית המתאימה.

24

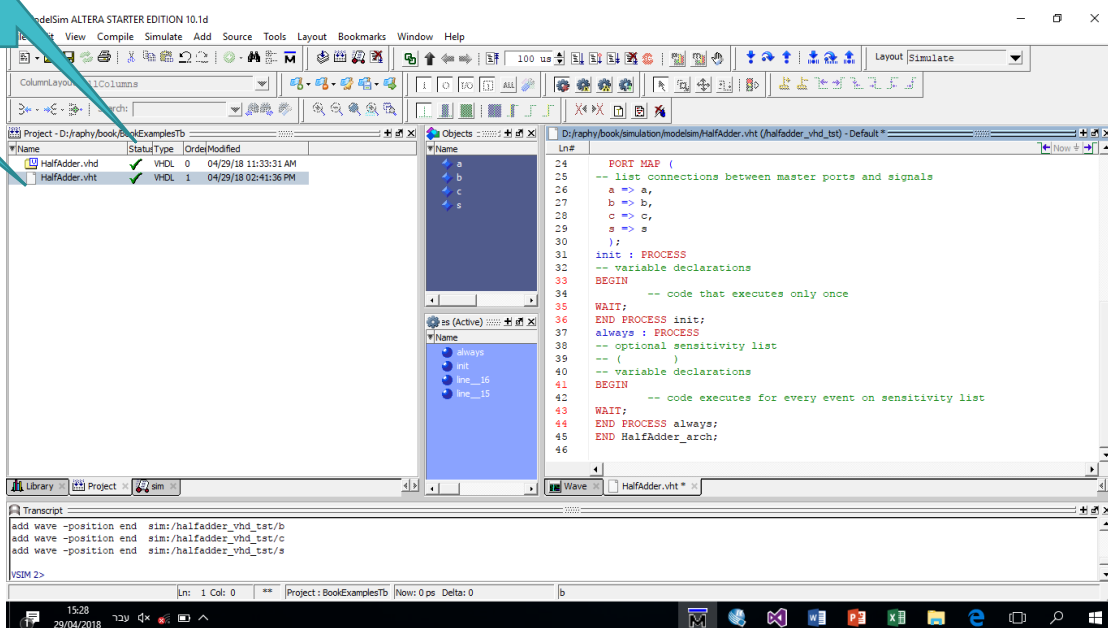


25. לאחר הקומפילציה סימני השאלה ישתנו לסימן של V.

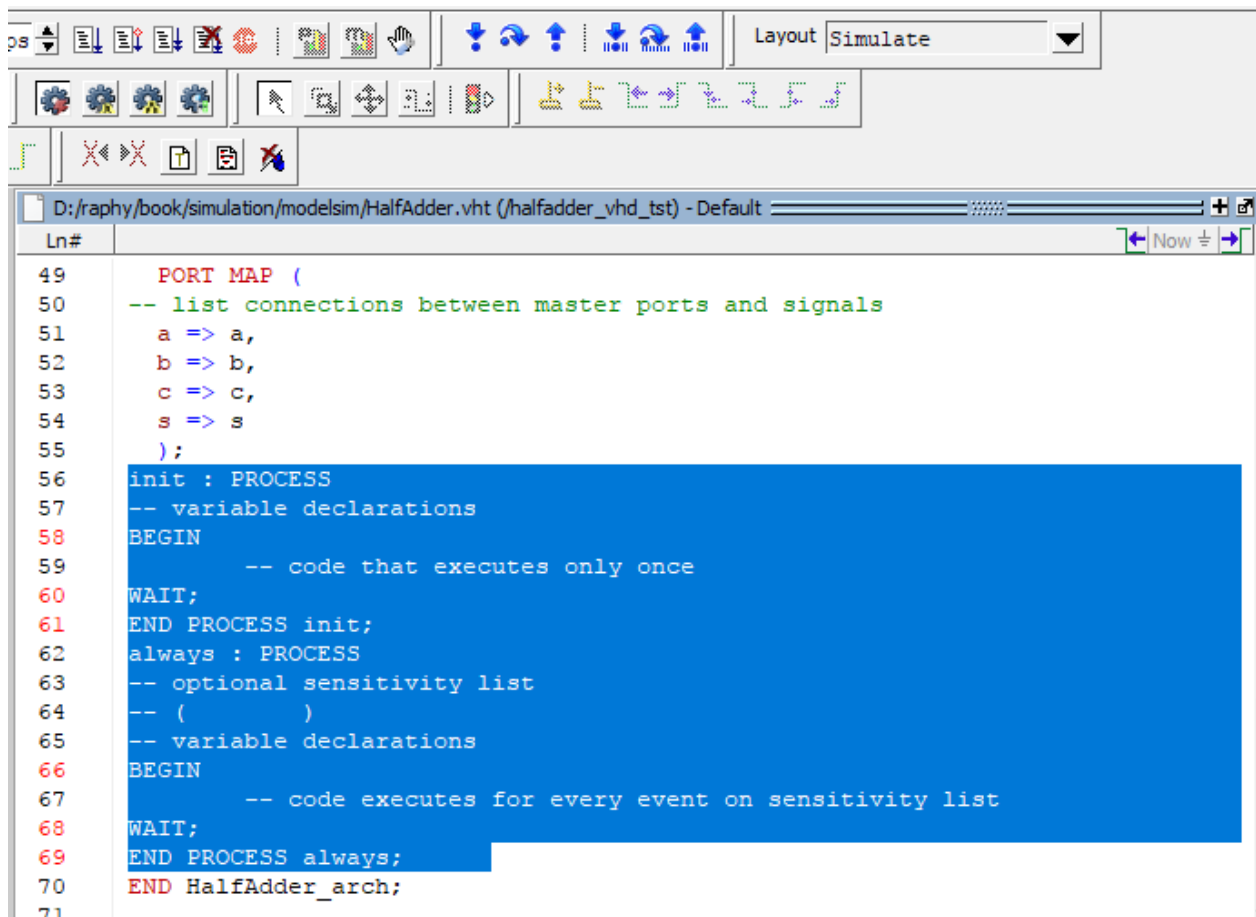
26. הקליקו פעמיים על הקובץ השני.

25

26



27. מסיבות שנסביר מאוחר יותר, נא מחקו את השורות הבאות: מ-init:process ועד end .process always



```

49  PORT MAP (
50  -- list connections between master ports and signals
51  a => a,
52  b => b,
53  c => c,
54  s => s
55  );
56  init : PROCESS
57  -- variable declarations
58  BEGIN
59  -- code that executes only once
60  WAIT;
61  END PROCESS init;
62  always : PROCESS
63  -- optional sensitivity list
64  -- ( )
65  -- variable declarations
66  BEGIN
67  -- code executes for every event on sensitivity list
68  WAIT;
69  END PROCESS always;
70  END HalfAdder_arch;
71

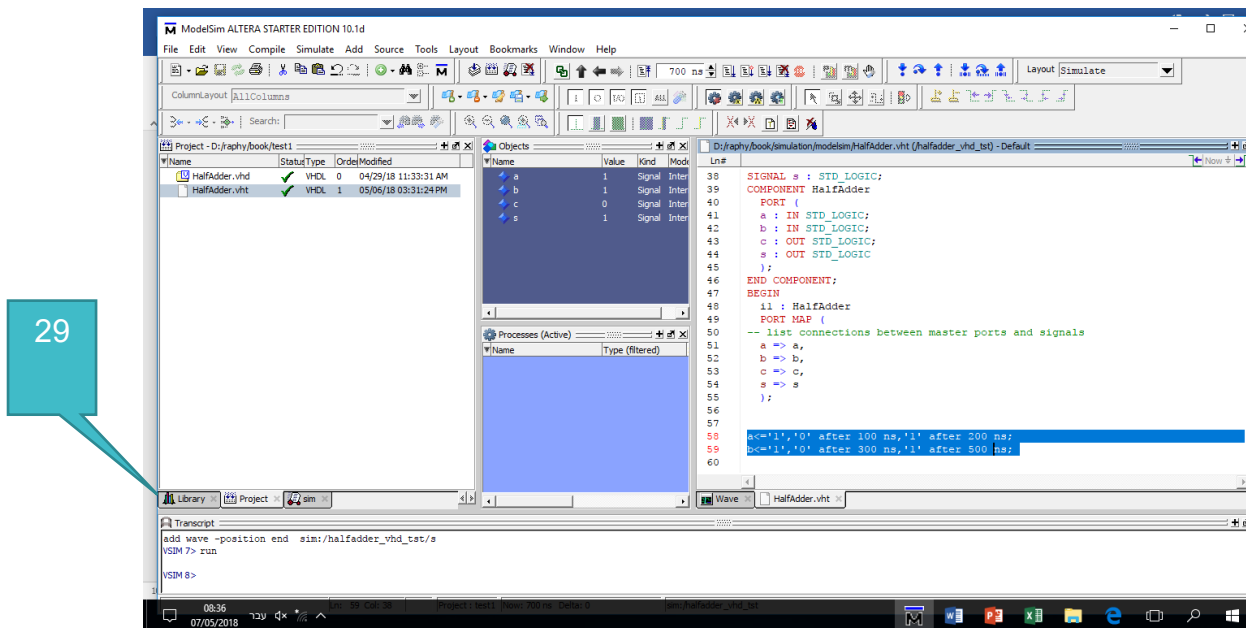
```

28. הוסיפו את שתי השורות הבאות, נא הקפידו להקליד אותן במדויק.

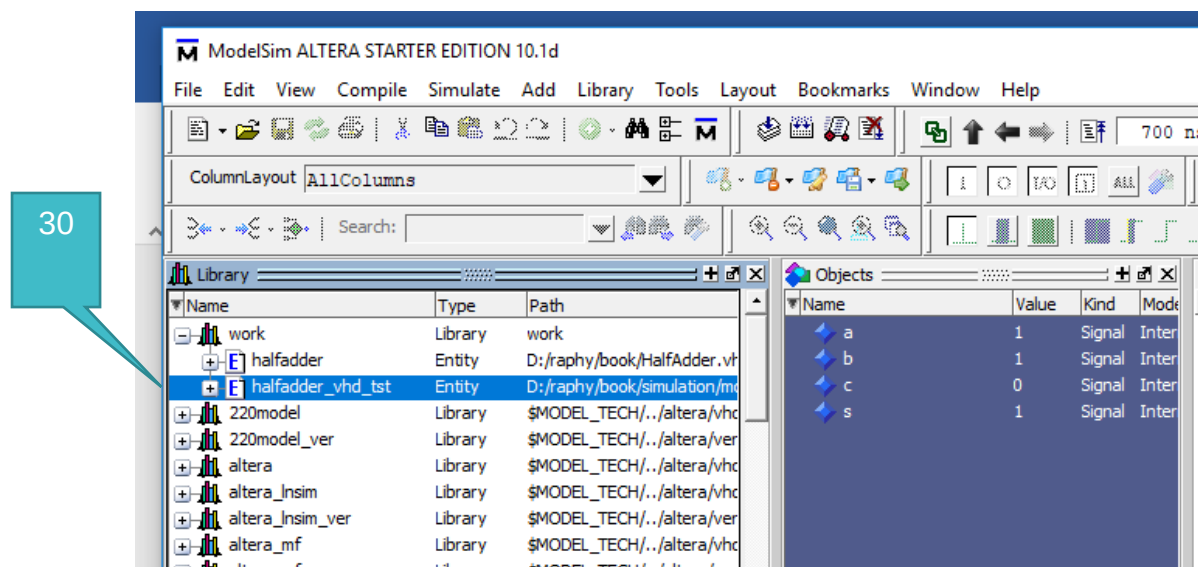
;a<='1','0' after 100 ns,'1' after 200 ns

;b<='1','0' after 300 ns,'1' after 500 ns

29. הקליקו על צלמית ה-library.

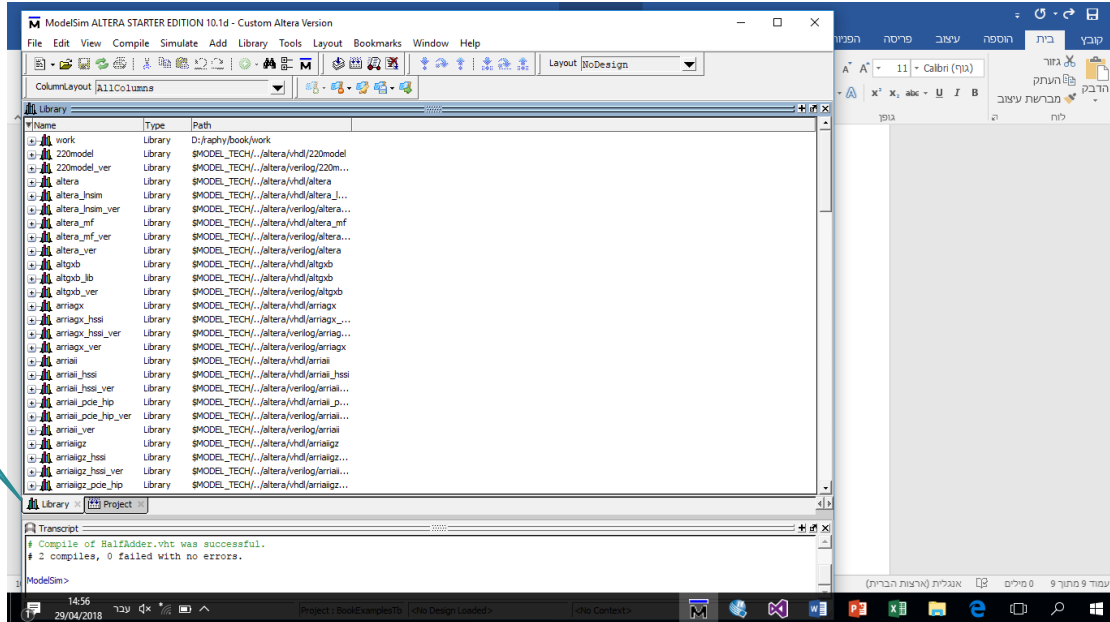


30. הקליקו פעמיים על קובץ ה-test bench בספריית work.

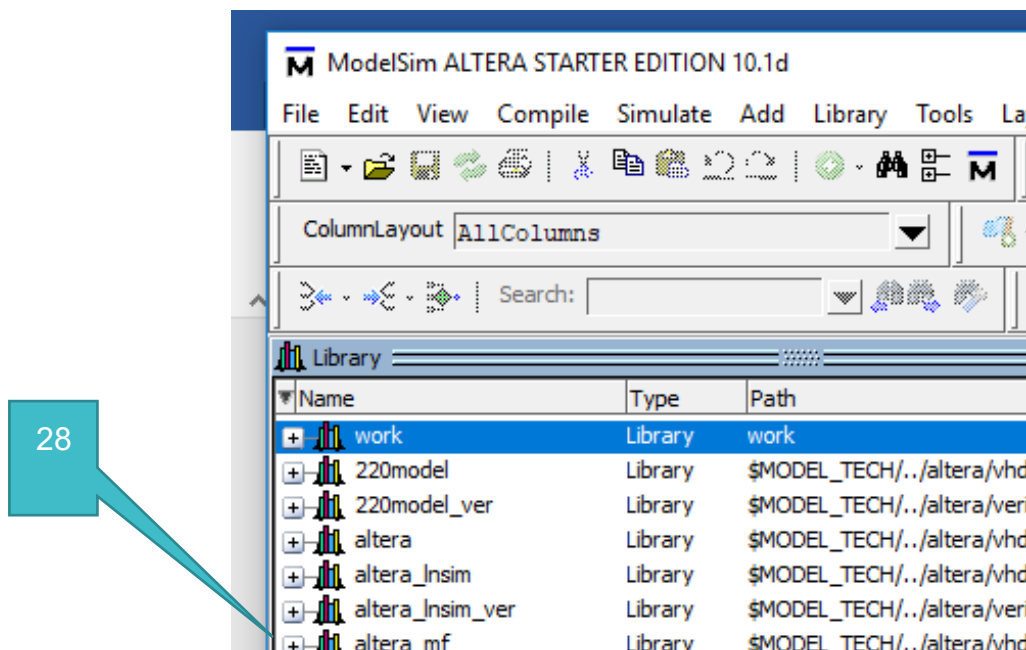


31. הקליקו על library.

32. כעת תופיע ספריית work כפי שהוגדרה באחד מהשלבים הקודמים.

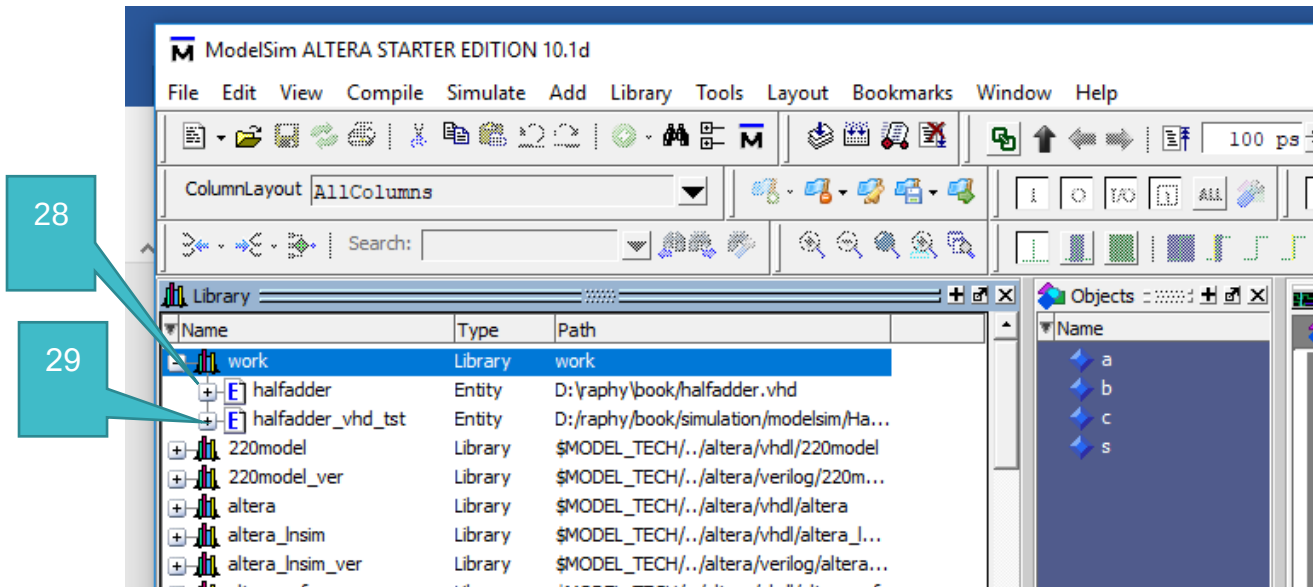


33. הקליקו על סימן הפלוס (+) שבספריית work.

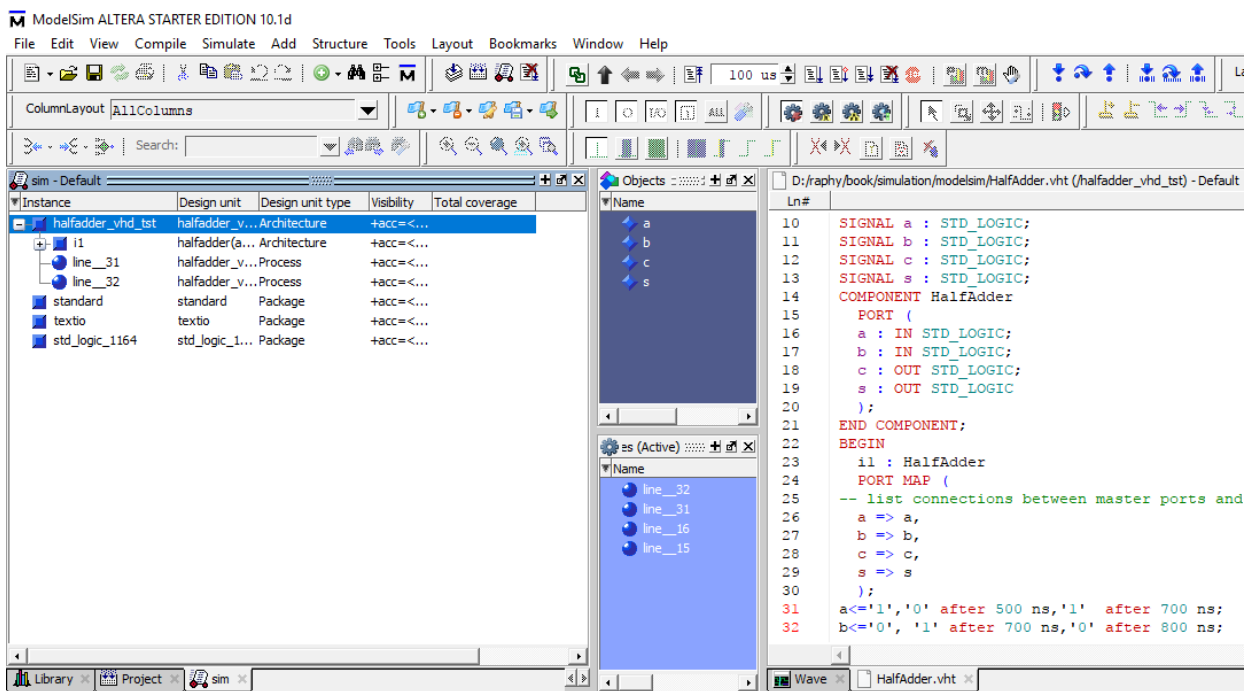


34. יופיעו שני קבצים – האחד הוא: HalfAdder.vhd.

35. הקובץ השני הוא: HalfAdder_vhd_tst.



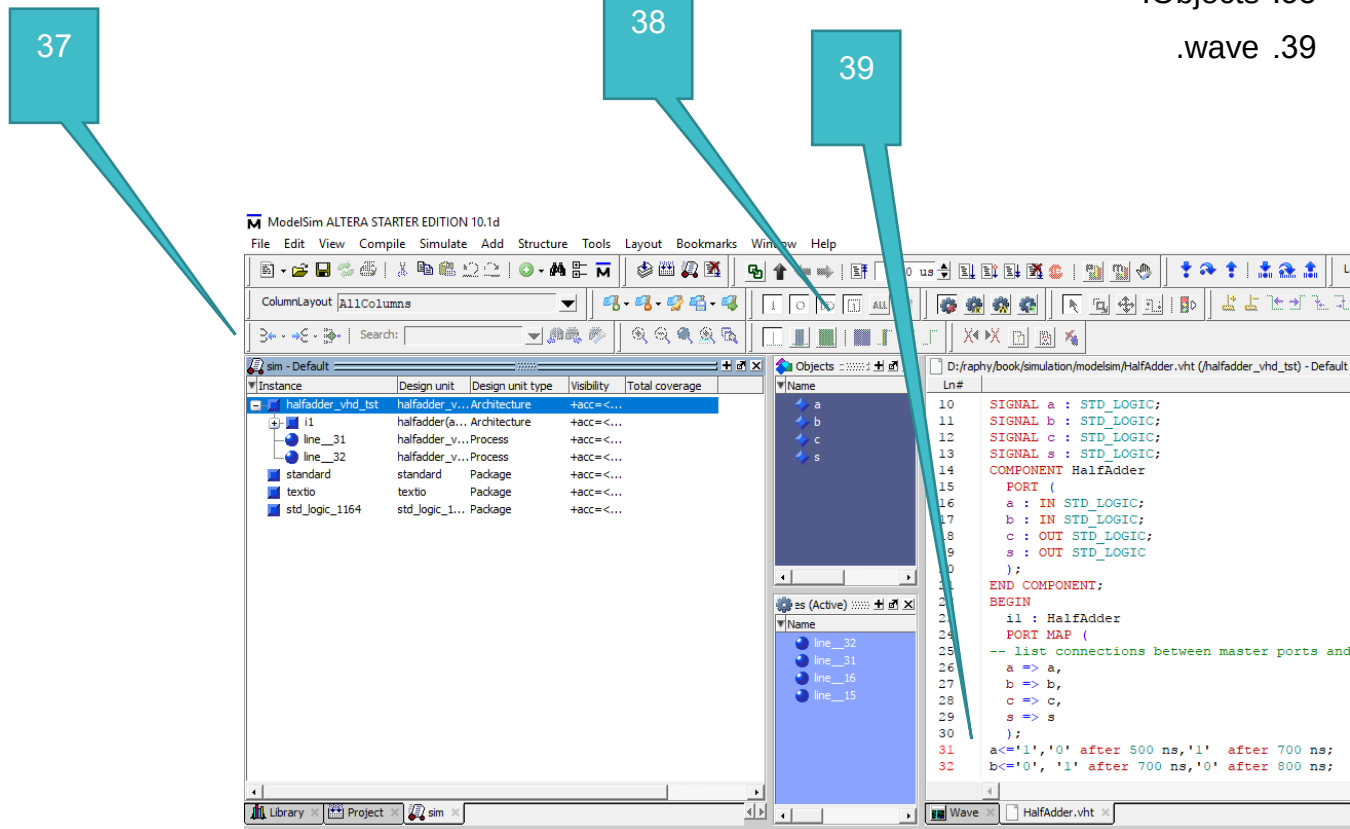
36. יפתחו החלונות האלה, נפרט אותם ונציג כל אחד מהם בנפרד.



.Sim 37

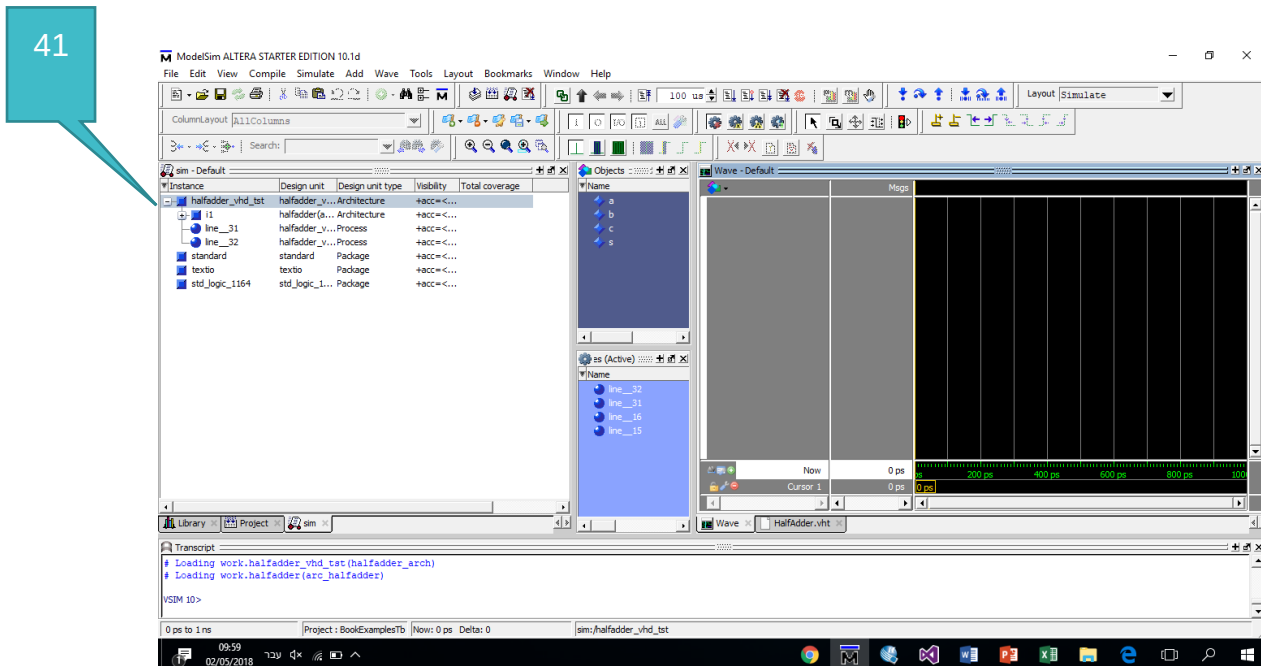
.Objects 38

.wave 39



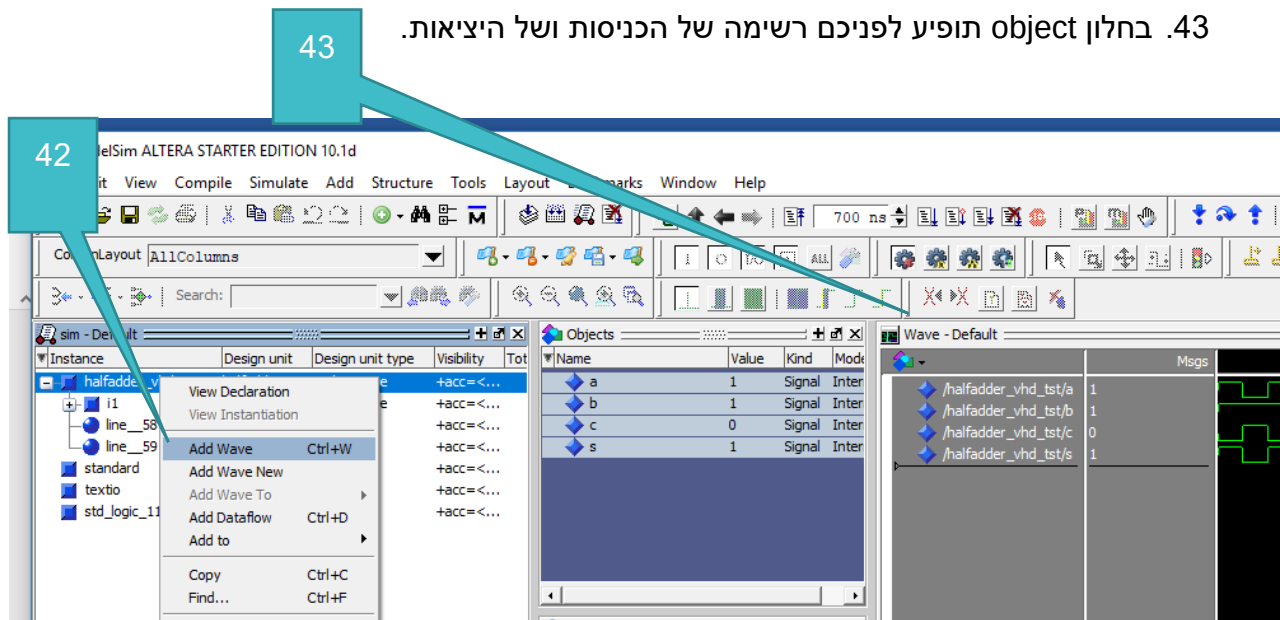
40. הקליקו על הצלמית של wave.

41. הקליקו עם הלחצן הימני של העכבר בשרת HalfAdder_vhd_tst.



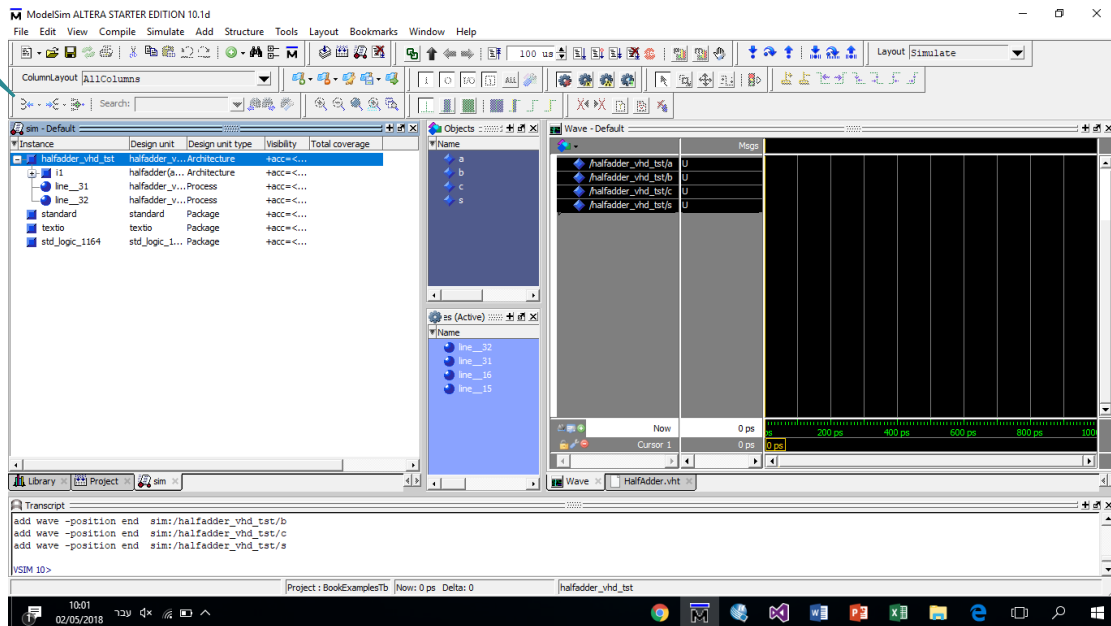
42. לחצו על add wave.

43. בחלון object תופיע לפניהם רשימה של הכניסות ושל היציאות.



44. אפשר גם לגרור את כל הרשימה לחלון ה-wave באמצעות הלחצן השמאלי של העכבר.

44



45. נכיר את פונקציות ההרצה.

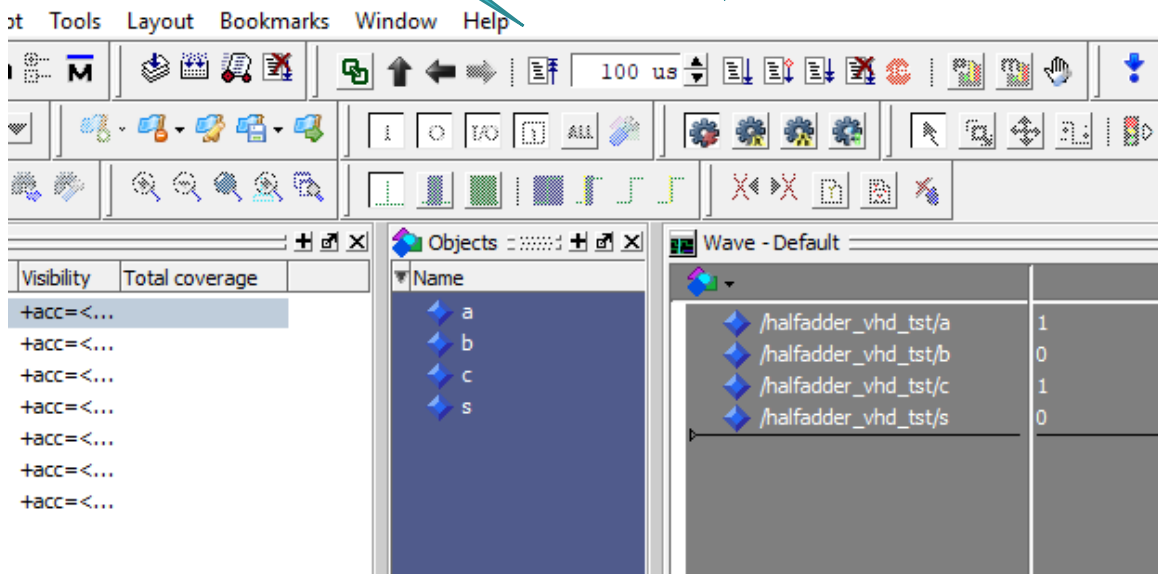
46. Run.

47. Restart.

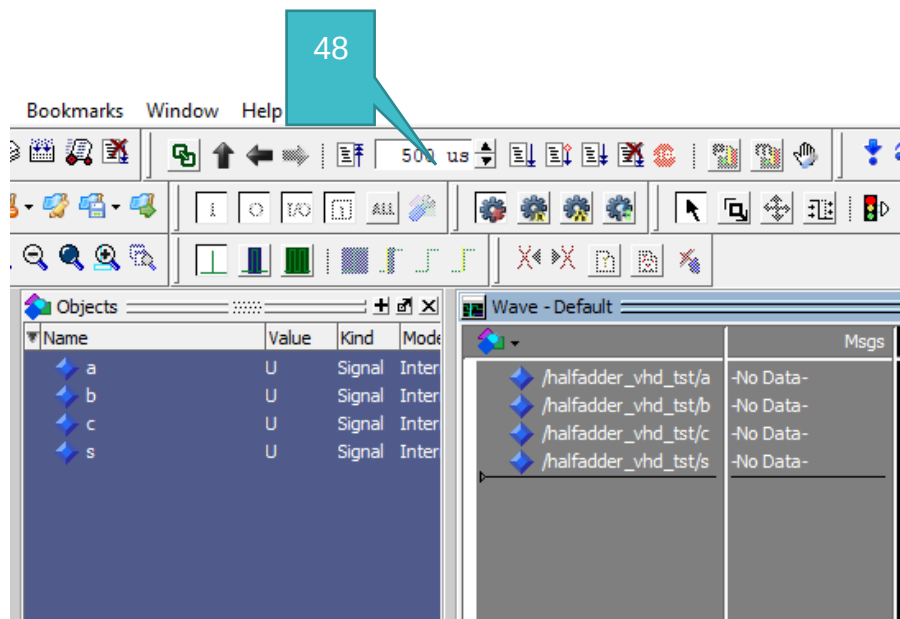
47

48

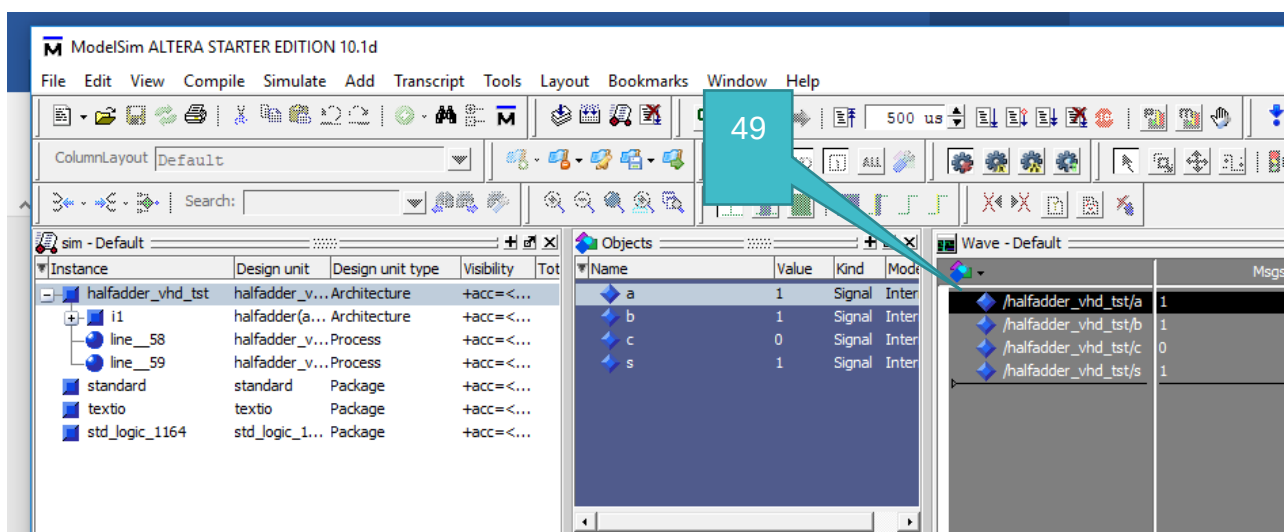
46



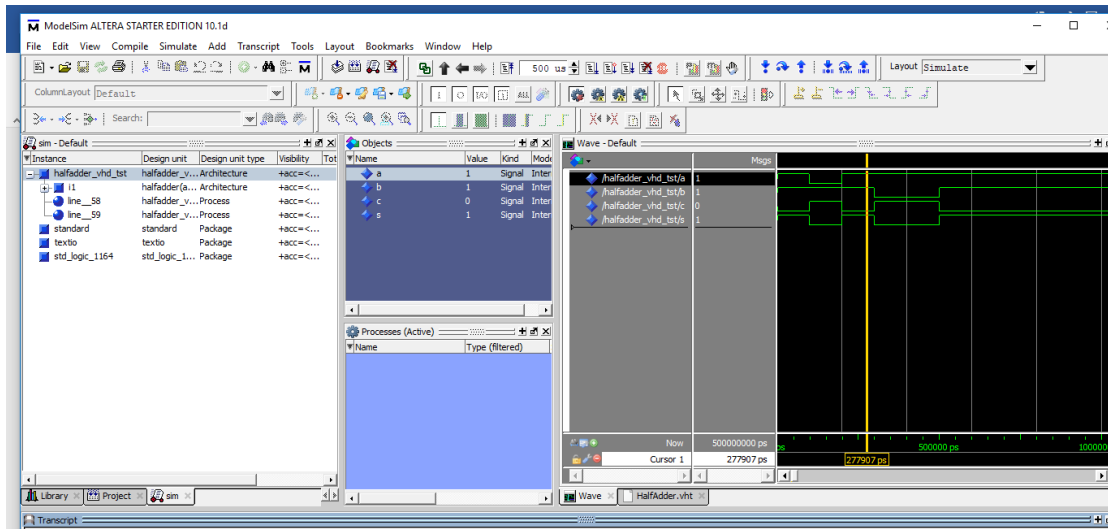
48. בחלון הזה אפשר לקבוע את זמן ההרצה. שנו את 100 ל-500 ואת ns (n = ננו) ל-us.
(u = מיקרו).



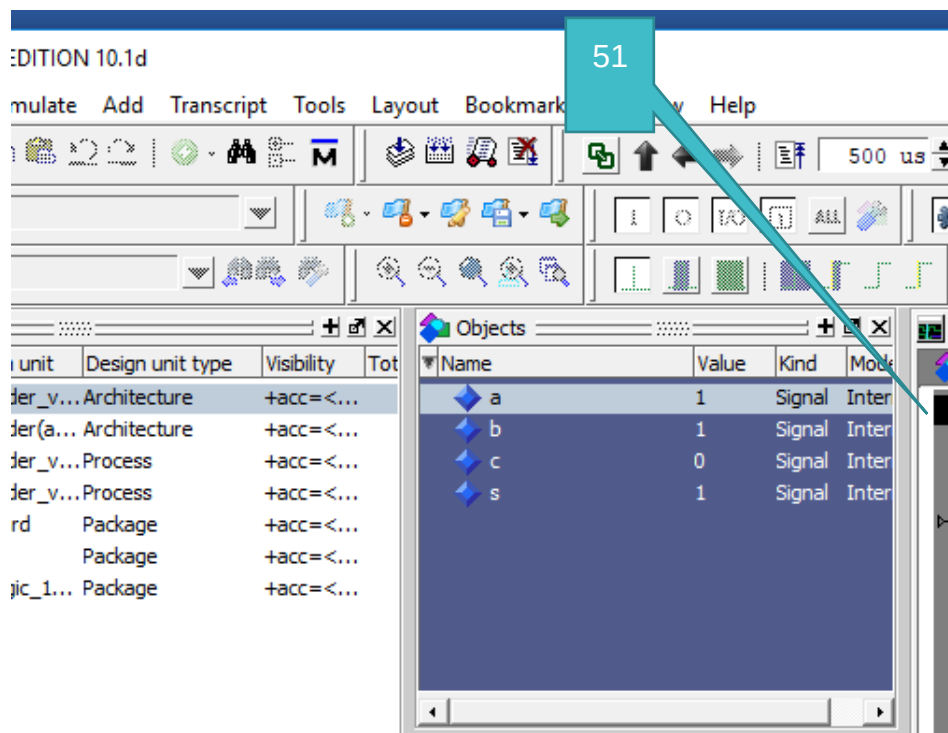
49. הקליקו על run.
בחלון תתקבל תוצאה של גלים.



50. תוצאה אפשרית:



51. אפשר לעשות zoom, הקישו על המסך השחור ב-wave.



52. הקישו על אחד מהמשתנים בחלון wave.

כאן הזרקור ממוקד על התוצאה של משתנה a. אם נקיש על החצים הכחולים שמופנים כלפי מעלה וכלפי מטה נוכל להגיע לשינויים הבאים או הקודמים.

52

