

כתיבת ספריה לחיישנים ורכיבי קלט / פלט

ציוד נדרש:

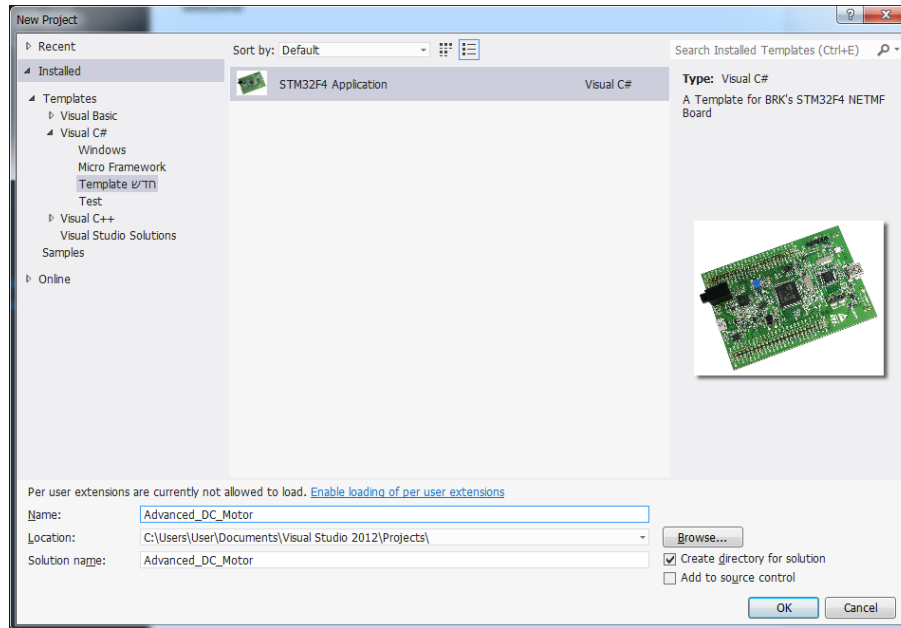
- ערכת פיתוח
- חיישן / רכיב קלט/פלט אליו רוצים לכתוב את הספרייה
- חוטי חיבור תואמים בין בקר לרכיב

רקע עיוני

- שיטת PWM – אפנון רוחב פולס
- הבנה מעמיקה בנישא תכנות מונחה עצמים (OOP)
- מרכיבים בסיסיים של אובייקטים וסוגיהם:
 - constructor
 - שדה (field)
 - מאפיינים (Properties)
 - מטודות / פונקציות (Method / function)
- ירושה | overwrite

מהלך הניסוי

1. בניסוי זה אנו ניצור ספריה להפעלת מנוע DC במהירות משתנה.
2. ניצור פרויקט חדש לעבודה עם הבקר, כפי שנלמד בשיעורים הקודמים. ניתן לו שם `Advanced_DC_Motor`. כמובן, ניתן לבחור כל שם אחר, אך מומלץ שהוא יתאר את הרכיב בו נעסוק בפרויקט.



3. במאפייני הפרויקט, כרגיל, נשנה את ה Transport ל USB

4. נפתח את קובץ Program.cs שבפרויקט.

5. נחבר את הבקר למחשב ע"י שני חיבורי ה USB שלו ונצרום את הפרויקט לבקר בכדי לוודא תקשורת תקינה בין המחשב לבקר. בכך נפתור בעיות עתידיות – במידה ויהיו בעיות צריבה בהמשך, נדע בוודאות שהבקר תקין.

6. נעצור את התקשורת מול הבקר ע"י הלחיצה על הפקד stop שבסרגל הכלים של Visual Studio.

7. כרגע התוכנה של הפרויקט (התוכן של הקובץ Program.cs) נראית כך:

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Stm32;
using System.Threading;
using System.Text;

namespace Advanced_DC_Motor
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print("Hello World!");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

```
    }  
  
    }  
}
```

8. מחוץ ל namespace הקיים (רצוי אחריו), ניצור אחד חדש בשם DC_Motor_Lib, למשל. ובתוכו מחלקה (class) בשם DC_Motor. אזי התוכנה תראה כך:

```
using System;  
using Microsoft.SPOT;  
using Microsoft.SPOT.Hardware;  
using Stm32;  
using System.Threading;  
using System.Text;  
  
namespace Advanced_DC_Motor  
{  
    public class Program  
    {  
        public static void Main()  
        {  
            Debug.Print("Hello World!");  
            Thread.Sleep(Timeout.Infinite);  
        }  
    }  
}
```

```
namespace DC_Motor_Lib  
{  
    public class DC_Motor  
    {  
  
    }  
}
```

9. בתוך המחלקה החדשה נגדיר את המשתנים שיהיו שימושיים בה, אך לא יהיו חשופים למשתמשים חיצוניים למחלקה:

```
namespace DC_Motor_Lib  
{  
    public class DC_Motor
```

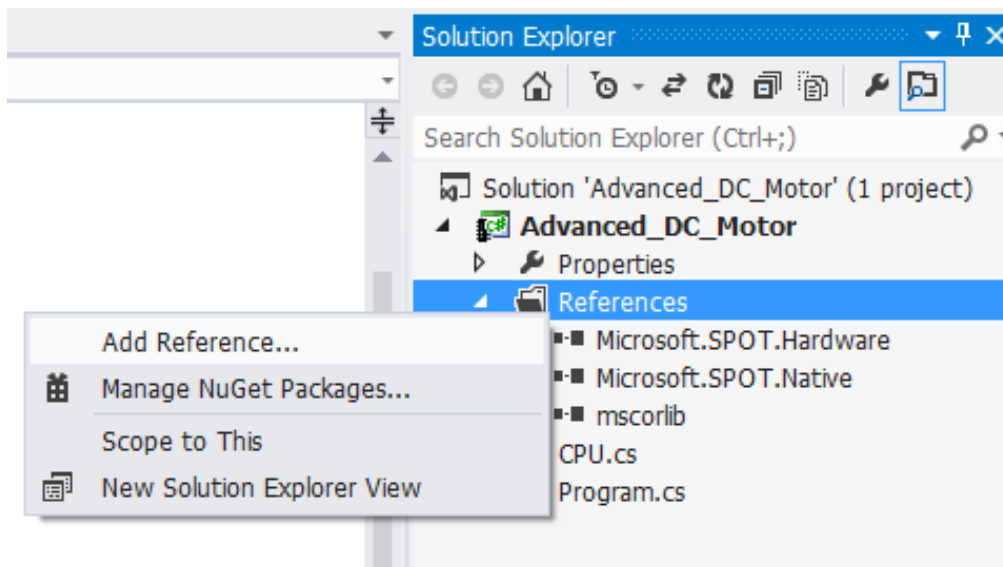
© BRK כל הזכויות שמורות. אין להעביר לצד שלישי ללא אישור בכתב מהחברה.

```

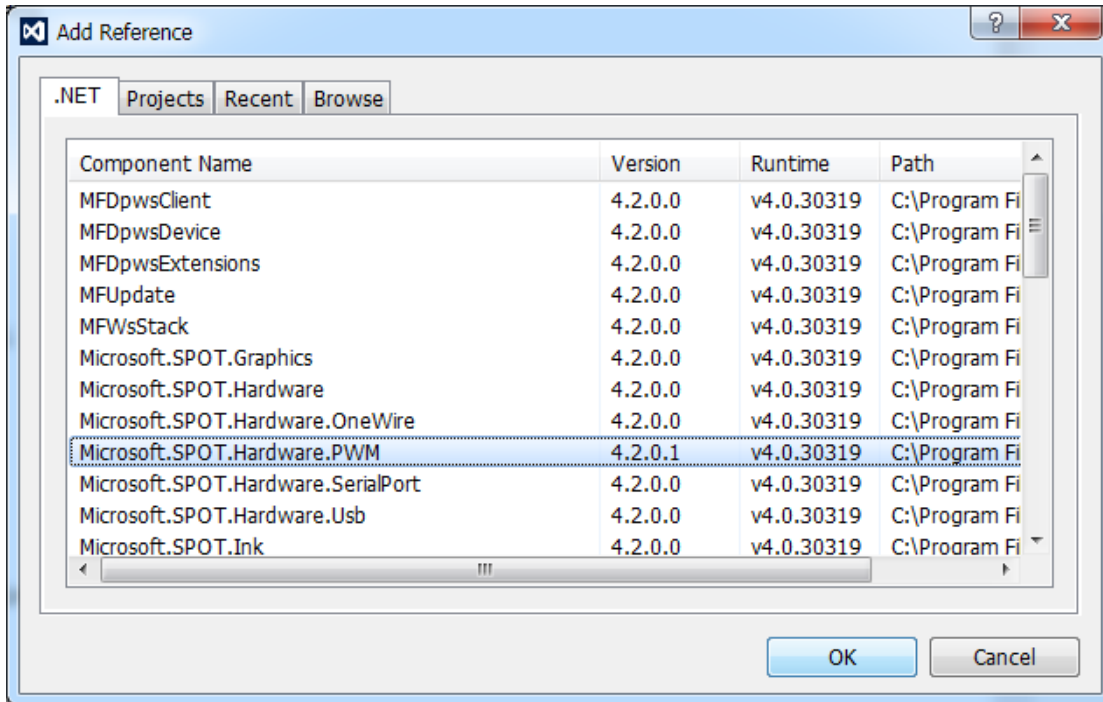
{
    private string name;           שם המנוע. לדוגמא: "מנוע שמאלי" או "מנוע ימני"
    private uint stopTime, startTime; זמן ההתנעה וזמן העצירה של המנוע
    private uint speed;           מהירות סיבוב רצויה באחוזים
    private bool direction;      כיוון הסיבוב (עם כיוון השעון או נגדו)
    private Cpu.PWMChannel pwm_ch; הפין הבקר אליו נחבר את המנוע דרך דוחף.
}
}

```

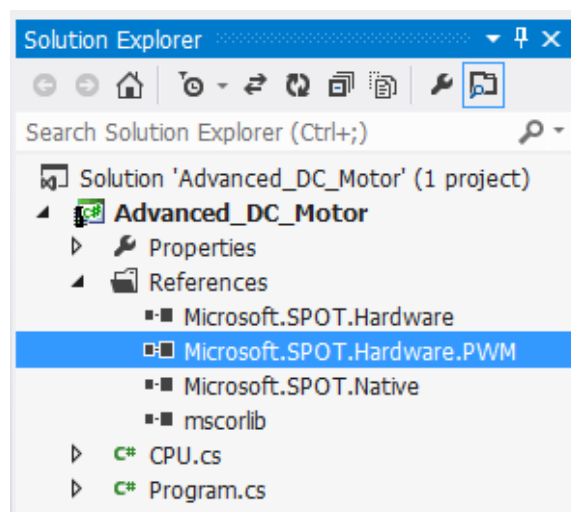
10. מאחר ובפרויקט יהיה שימוש ב PWM, יש להוסיף את ה reference שלו. כזכור, נלחץ לחצו ימני על ה References של הפרויקט המופיעים בחלונית Solution Explorer שבצדו הימני של המסך ובתפריט שתפתח נבחר את האופציה העליונה: Add Reference



11. בחלונית שתפתח, בלשונית .Net. (השמאלית ביותר) נבחר את הרכיב: Microsoft.SPOT.Hardware.PWM כמתואר באיור הבא:



12. נלחץ על הלחצן OK בחלקה השמאלי של החלונית ונוודא בחלונית ה Solution Explorer כי ה Reference התוסף בהצלחה:



13. נגדיר שני הדקים אליהם יתחבר המנוע. אחד מהם חייב להיות ערוץ PWM (אפנון רוחב פולס) והשני הדק מוצא רגיל:

```
private PWM pwm1;
private OutputPort out1;
```

14. שימו לב: רק הגדרנו את עצם קיומם של הדקים אלו, אך לא הגדרנו את מאפייניהם כגון לאיזה הדק אם מתייחסים.

15. לאחר הגדרת השדות (fields) ניתן לעבור להגדרת ה Constructor – הבונה של המחלקה.

16. בהמשך להגדרת השדות, נגדיר תחילה את הבונה (Constructor) המצומצם יותר:

```
public DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin)
{
    name = "Motor";
    stopTime = 5000;
    startTime = 5000;
    speed = 100;
    pwm_ch = FirstPin;
    direction = false;

    out1 = new OutputPort(SecondPin, direction);
    pwm1 = new PWM(pwm_ch, 100, 0.0, false);
    pwm1.Start();
}
```

הסבר קצר: ה Constructor מקבל ממפעיל את ההדקים אליהם מחובר המנוע ע"ג הכרטיס וקובע את כל הערכים כברירת מחדל:

- שם המנוע – Motor
- זמן עצירה – 5 שנית (5000 מילי שניות)
- זמן התנעה – 5 שנית (5000 מילי שניות)
- מהירות – 100%
- כיוון סיבוב – עם כיוון השעון.

לאחר מכן, הבונה מגדיר את המאפיינים של ההדקים שהוגדרו בשדות ע"פ אלו אליהם מחובר המנוע בפועל. בסיום ההגדרות, הבונה מפעיל את ה pwm.

17. אין זה חובה, אך מעוניינים יכולים להוסיף (ולא למחוק את הקודם) גם בונה מלא יותר כדוגמת זה:

```
public DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin,
               uint StopTime, uint StartTime, uint Speed,
               bool CW_Direction )
{
    name = "Motor";
    stopTime = StopTime;
    startTime = StartTime;
    speed = Speed;
```

© BRK כל הזכויות שמורות. אין להעביר לצד שלישי ללא אישור בכתב מהחברה.

```
pwm_ch = FirstPin;
direction = CW_Direction;

out1 = new OutputPort(SecondPin, direction);
pwm1 = new PWM(pwm_ch, 100, 0.0, direction);
pwm1.Start();
}
```

18. בשלב זה, ה namespace החדש יראה כך:

```
namespace DC_Motor_Lib
{
    public class DC_Motor
    {
        private string name;
        private uint stopTime, startTime;
        private uint speed;
        private bool direction;
        private Cpu.PWMChannel pwm_ch;

        private PWM pwm1;
        private OutputPort out1;

        public DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin)
        {
            name = "Motor";
            stopTime = 5000;
            startTime = 5000;
            speed = 100;
            pwm_ch = FirstPin;
            direction = false;

            out1 = new OutputPort(SecondPin, direction);
            pwm1 = new PWM(pwm_ch, 100, 0.0, false);
            pwm1.Start();
        }

        public DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin,
                       uint StopTime, uint StartTime, uint Speed,
                       bool CW_Direction)
        {
            name = "Motor";
            stopTime = StopTime;
            startTime = StartTime;
            speed = Speed;
            pwm_ch = FirstPin;
            direction = CW_Direction;

            out1 = new OutputPort(SecondPin, direction);
        }
    }
}

© BRK כל הזכויות שמורות. אין להעביר לצד שלישי ללא אישור בכתב מהחברה.
```



```

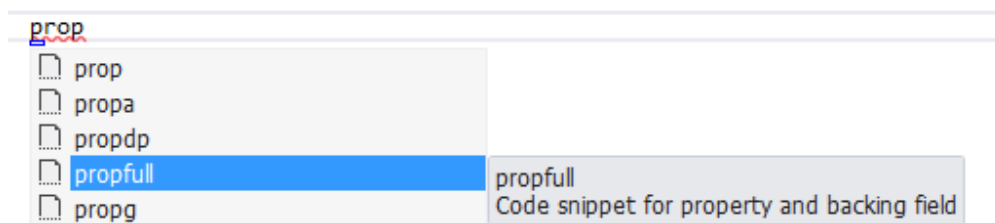
        pwm1 = new PWM(pwm_ch, 100, 0.0, direction);
        pwm1.Start();
    }

}
}

```

19. נעבור להגדרת המאפיינים (ה Properties) של המחלקה שיצרנו (DC_Motor).

20. ב Visual studio קיימים מספר "קיצורי דרך" לשם יצירת ה Properties הרצויים. נעמוד עם הסמן מחוץ לבונה שיצרנו (אחריו), אך עדיין בתוך המחלקה, כמובן. נקליד במקלדת את האותיות prop שהן תחילת המילה properties ונראה את האפשרויות ש Visual Studio נותן להשלמת הקוד:



21. עם העכבר או המקלדת נבחר את האפשרות propfull ונלחץ על הלחצן Tab שבמקלדת.

22. נראה שסביבת הפיתוח השלימה את הקוד הבא:

```

private int myVar;

public int MyProperty
{
    get { return myVar; }
    set { myVar = value; }
}

```

23. בשלב זה, מבלי שנגענו במשהו, ניתן לעבור בין השדות המודגשים ולשנות אותם. למעשה, כבר הגדרנו את השדות בראש המחלקה וכאן יהיה עלינו רק לקשר את ה property לשדה.

24. נשנה את השם של ה property שיצרנו מ MyProperty ל Name (שימו לב ש נהוג להתחיל את שמות המאפיינים באותיות גדולות, בשונה משדות שאת שמם נהוג להתחיל באותיות קטנות).

25. מאחר ובמחלקה שלנו name הוא שדה מסוג string (מחרוזת), נשנה את הטיפוס שלו מ int ל string.

26. כפי שאמרנו, היות והשדות הוגדרו בראש המחלקה, אין צורך בהגדרה חוזרת כאן וניתן למחוק את השורה הראשונה.

27. נשנה את שמות השדות מ myVar ל name (באותיות קטנות!)

28. בסה"כ קיבלנו את ה property הבא:

```
public string Name
{
    get { return name; }
    set { name = value; }
}
```

29. אין זה חובה, אך ניתן להגדיר לכל property את התיאור שלו שיוצג למשתמש על פיו ידע מה עליו להזין בproperty זה, כפי שיודגם בהמשך. בשלב זה רק ניצור אותו ע"י עמידה עם הסמן בשורה שלפני הproperty והקשה על מקש / **שלוש** פעמים.

```
/// <summary>
/// |
/// </summary>
public string Name
{
    get { return name; }
    set { name = value; }
}
```

30. סביבת הפיתוח תשלם את הקוד ותציג את הדבר הבא:

31. הסמן עומד בין שתי תגי ה <summary>. נקליד שם את הטקסט המתאר את ה property שלנו, למשל:

```
/// <summary>
/// motors name (as left, right, up, etc...)
/// </summary>
public string Name
{
    get { return name; }
    set { name = value; }
}
```

32. באופן דומה ניתן להגדיר גם את המאפיינים הנוותרים של המחלקה: StartTime, StopTime, Speed, כפי שנעשה בקוד הבא:

```
/// <summary>
/// Stop time is a time that takes motor to stop since SlowStop() command.
/// </summary>
```

```
public uint StopTime
{
    get { return stopTime; }
    set { stopTime = value; }
}

/// <summary>
/// Start time is a time that takes motor to reach 100% power since SlowStart()
/// command.
/// </summary>
public uint StartTime
{
    get { return startTime; }
    set { startTime = value;}
}

/// <summary>
/// set or get speed of this motor. Speed needs to be between 0 and 100
/// </summary>
public uint Speed
{
    get { return speed; }
    set { speed = value; }
}
```

33. ב property של Speed נדרשים מספר תוספות: ויודוי שערך השדה לא יעלה על המספר 100 היות ואין יותר מ 100% תפוקת המנוע, והדבר הנוסף הוא התחשבות בכיוון הסיבוב לשם קביעת גורם המחזור המשנה בפועל את מהירות המנוע. לשם תיקונים אלו נוסף למאפיין המהירות את הקוד הבא:

```
/// <summary>
/// set or get speed of this motor. Speed needs to be between 0 and 100
/// </summary>
public uint Speed
{
    get { return speed; }
    set
    {
        if (value > 100) speed = 100;
        else speed = value;
    }
}
```

© BRK כל הזכויות שמורות. אין להעביר לצד שלישי ללא אישור בכתב מהחברה.

```

        if (direction) pwm1.DutyCycle = 1 - (double)speed / 100;
        else pwm1.DutyCycle = (double)speed / 100;
    }
}

```

34. נגדיר Property שיהיה מקושר לשדה כיוון הסיבוב של המנוע ויקבע את כיוון הסיבוב של המנוע בפועל ע"י מתן "1" לוגי או "0" לוגי ברגל השנייה (לא ה PWM) של המנוע:

```

/// <summary>
/// set or get the direction of motors rotation
/// </summary>
public bool CW_Direction
{
    get { return direction; }
    set
    {
        direction = value;
        out1.Write(direction);
        pwm1.DutyCycle = 1 - pwm1.DutyCycle;
    }
}

```

35. בשלב זה המחלקה מוגדרת היטב, בעלת כל השדות והמאפיינים הנדרשים, אך אינה יודעת לבצע דבר בפועל. היא לא מסובבת ולא עוצרת את המנוע כנדרש!

36. לאחר כל המאפיינים, נוסיף פונקציות / מטודות של עצירה והנעה מהירה של המנוע:

```

/// <summary>
/// fast start of motor to set speed
/// </summary>
public void FastStart()
{
    Debug.Print("fast start");
    out1.Write(direction);
    if (direction) pwm1.DutyCycle = 1 - speed / 100;
    else pwm1.DutyCycle = speed / 100;
    Debug.Print("DC: "+(speed / 100).ToString("f2")+" dir: " + direction);
}

```

```

/// <summary>
/// fast stop of motor

```

```

/// </summary>
public void FastStop()
{
    Debug.Print("fast stop");
    if (direction) out1.Write(false);
    else pwm1.DutyCycle = 0;

    Debug.Print("DC: "+pwm1.DutyCycle.ToString("f2")+ " dir: " + direction);
}

```

יש להקפיד על כך שכל המטודות יהיו מוגדרות כ `public` ובשום אופן לא `private` !

37. נוסף גם את המטודה המתחילה את הסיבוב של המנוע באופן הדרגתי. ניתן להיעזר הקוד הבא:

```

/// <summary>
/// slowly start the motor according to StartTime value
/// </summary>
public void SlowStart(uint StartTime)
{
    Debug.Print("slow start");
    out1.Write(direction);

    if (direction)
    {
        for (double dc = 1; dc > 0; dc -= 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
            Thread.Sleep((int)StartTime / 100);
        }
    }
    else
    {
        for (double dc = 0; dc < speed / 100; dc += 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
            Thread.Sleep((int)StartTime / 100);
        }
    }
}

```

38. ניתן להוסיף גם מטודה באותו השם, אך שאינה מקבלת פרמטר של `StartTime`, אלא לוקחת אותו מתוך השדה במלקה. המשתמש יוכל לבחור באיזה מבין שתי המטודות להשתמש.

```

/// <summary>
/// slowly start the motor according to StartTime value
/// </summary>
public void SlowStart()
{
    Debug.Print("slow start");
    out1.Write(direction);

    if (direction)
    {
        for (double dc = 1; dc > 0; dc -= 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
            Thread.Sleep((int)stopTime / 100);
        }
    }
    else
    {
        for (double dc = 0; dc < speed / 100; dc += 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
            Thread.Sleep((int)startTime / 100);
        }
    }
}

```

39. באופן דומה, נוסיף גם את מטודות העצירה המתונה:

```

/// <summary>
/// slowly stop the motor according to StopTime value
/// </summary>
public void SlowStop()
{
    Debug.Print("slow stop");

    if (direction)

```

```
{
    for (double dc = 0; dc < speed / 100; dc += 0.01)
    {
        pwm1.DutyCycle = dc;
        Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
        Thread.Sleep((int)stopTime / 100);
    }
}
else
{
    for (double dc = 1; dc > 0; dc -= 0.01)
    {
        pwm1.DutyCycle = dc;
        Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
        Thread.Sleep((int)stopTime / 100);
    }
}
}
```

```
/// <summary>
/// slowly stop the motor according to StopTime value
/// </summary>
public void SlowStop(uint StopTime)
{
    Debug.Print("slow stop");

    if (direction)
    {
        for (double dc = 0; dc < speed / 100; dc += 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
            Thread.Sleep((int)StopTime / 100);
        }
    }
    else
    {
        for (double dc = 1; dc > 0; dc -= 0.01)
```

```

        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: " + dc.ToString("f2") + " dir: " + direction);
            Thread.Sleep((int)StopTime / 100);
        }
    }
}

```

40. בשלב זה המחלקה מוכנה לעבודה!

41. כרגע כל תוכנת הפרויקט (Program.cs) נראית כך:

```

using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Stm32;
using System.Threading;
using System.Text;

namespace Advanced_DC_Motor
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print("Hello World!");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}

namespace DC_Motor_Lib
{
    public class DC_Motor
    {
        private string name;
        private uint stopTime, startTime;
        private uint speed;
        private bool direction;
        private Cpu.PWMChannel pwm_ch;

        private PWM pwm1;
        private OutputPort out1;
    }
}

```



```
public DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin)
{
    name = "Motor";
    stopTime = 5000;
    startTime = 5000;
    speed = 100;
    pwm_ch = FirstPin;
    direction = false;

    out1 = new OutputPort(SecondPin, direction);
    pwm1 = new PWM(pwm_ch, 100, 0.0, false);
    pwm1.Start();
}
```

```
public DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin,
                uint StopTime, uint StartTime, uint Speed,
                bool CW_Direction)
{
    name = "Motor";
    stopTime = StopTime;
    startTime = StartTime;
    speed = Speed;
    pwm_ch = FirstPin;
    direction = CW_Direction;

    out1 = new OutputPort(SecondPin, direction);
    pwm1 = new PWM(pwm_ch, 100, 0.0, direction);
    pwm1.Start();
}
```

```
/// <summary>
///
```

```
/// </summary>
public string Name
{
    get { return name; }
    set { name = value; }
}

/// <summary>
/// Stop time is a time that takes motor to stop since SlowStop()
command.
/// </summary>
public uint StopTime
{
    get { return stopTime; }
    set { stopTime = value; }
}

/// <summary>
/// Start time is a time that takes motor to reach 100% power since
SlowStart() command.
/// </summary>
public uint StartTime
{
    get { return startTime; }
    set { startTime = value;}
}

/// <summary>
/// set or get speed of this motor. Speed needs to be between 0 and 100
/// </summary>
public uint Speed
{
    get { return speed; }
    set
    {
        if (value > 100) speed = 100;
        else speed = value;

        if (direction) pwm1.DutyCycle = 1 - (double)speed / 100;
        else pwm1.DutyCycle = (double)speed / 100;
    }
}
```

```
    }
}

/// <summary>
/// set or get the direction of motors rotation
/// </summary>
public bool CW_Direction
{
    get { return direction; }
    set
    {
        direction = value;
        out1.Write(direction);
        pwm1.DutyCycle = 1 - pwm1.DutyCycle;
    }
}

//////////////////////////////////// Methods //////////////////////////////////////

/// <summary>
/// fast start of motor to set speed
/// </summary>
public void FastStart()
{
    Debug.Print("fast start");

    out1.Write(direction);
    if (direction) pwm1.DutyCycle = 1 - speed / 100;
    else pwm1.DutyCycle = speed / 100;

    Debug.Print("DC: "+(speed / 100).ToString("f2")+ " dir: "+direction);
}

/// <summary>
/// fast stop of motor
/// </summary>
public void FastStop()
{
    Debug.Print("fast stop");
    /*
    if (direction) pwm1.DutyCycle = 1;
    else pwm1.DutyCycle = 0;
    */
}
```

```
*/
if (direction) out1.Write(false);
else pwm1.DutyCycle = 0;

Debug.Print("DC: "+pwm1.DutyCycle.ToString("f2")+ " dir: "+direction);
}

/// <summary>
/// slowly start the motor according to StartTime value
/// </summary>
public void SlowStart()
{
    Debug.Print("slow start");

    out1.Write(direction);
    if (direction)
    {
        for (double dc = 1; dc > 0; dc -= 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
            Thread.Sleep((int)stopTime / 100);
        }
    }
    else
    {
        for (double dc = 0; dc < speed / 100; dc += 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
            Thread.Sleep((int)startTime / 100);
        }
    }
}

/// <summary>
/// slowly start the motor according to StartTime value
/// </summary>
public void SlowStart(uint StartTime)
{
```

```
Debug.Print("slow start");
out1.Write(direction);

if (direction)
{
    for (double dc = 1; dc > 0; dc -= 0.01)
    {
        pwm1.DutyCycle = dc;
        Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
        Thread.Sleep((int)StartTime / 100);
    }
}
else
{
    for (double dc = 0; dc < speed / 100; dc += 0.01)
    {
        pwm1.DutyCycle = dc;
        Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
        Thread.Sleep((int)StartTime / 100);
    }
}
}
```

```
/// <summary>
/// slowly stop the motor according to StopTime value
/// </summary>
public void SlowStop()
{
    Debug.Print("slow stop");

    if (direction)
    {
        for (double dc = 0; dc < speed / 100; dc += 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
        }
    }
}
```

```

        Thread.Sleep((int)stopTime / 100);
    }
}
else
{
    for (double dc = 1; dc > 0; dc -= 0.01)
    {
        pwm1.DutyCycle = dc;
        Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
        Thread.Sleep((int)stopTime / 100);
    }
}

}

/// <summary>
/// slowly stop the motor according to StopTime value
/// </summary>
public void SlowStop(uint StopTime)
{
    Debug.Print("slow stop");

    if (direction)
    {
        for (double dc = 0; dc < speed / 100; dc += 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
            Thread.Sleep((int)StopTime / 100);
        }
    }
    else
    {
        for (double dc = 1; dc > 0; dc -= 0.01)
        {
            pwm1.DutyCycle = dc;
            Debug.Print("DC: "+dc.ToString("f2")+ " dir: "+direction);
            Thread.Sleep((int)StopTime / 100);
        }
    }
}
}

```

```
}
}
```

42. נבדוק את הפעילות התקינה של המנוע.

43. ננתק את הבקר מהמחשב ונחבר אליו את המנוע DC דרך דוחף הזרם, כפי שזה נעשה בניסוי בו עסקנו בהפעלת מנוע DC, אלא שהפעם עלינו לחבר את הדקי הפיקוד של דוחף הזרם לערוץ PWM ופין כללי פנוי. לדוגמא, נחבר את המנוע להדקי הבקר PD_11 ו PD_12 הממוקמים בקירוב זה לזה.

44. כעת, נרצה להפעיל את המנוע מתוך ה Main(), אך ראשית עלינו להוסיף את ה using:

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Stm32;
using System.Threading;
using System.Text;
using dc

namespace DC_Motor_Lib
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print("Hello World!");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

namespace DC_Motor_Lib

שימו לב שיש להוסיף את ה using של שם המחלקה שכתבתם!

45. ניצור לפני ה Main() את האובייקט החדש של המנוע. שימו לב ל"פרגון" של סביבת הפיתוח במופיע בבלון אפור.

```
public class Program
{
    static DC_Motor Motor1 = new DC_Motor(
    public static void Main()
    {
        Debug.Print("Hello World!");
        Thread.Sleep(Timeout.Infinite);
    }
}
```

DC_Motor.DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin)

- const
- ConstraintException
- continue
- Cpu
- Cpu.PWMChannel
- CW_Direction:
- DateTime
- DateTimeKind
- DayOfWeek

© BRK כל הזכויות שמו. שור בכתב מהחברה.

46. מאחר וכתבנו שני בונים (constructor), נוכל לבחור עם העכבר את הבונה המתאים ע"י לחיצה על החצים למעלה / למטה. נבחר באפשרות השנייה שהוא הבונה המגדיר מאפיינים רבים יותר של האובייקט אוטו אנו יוצרים.

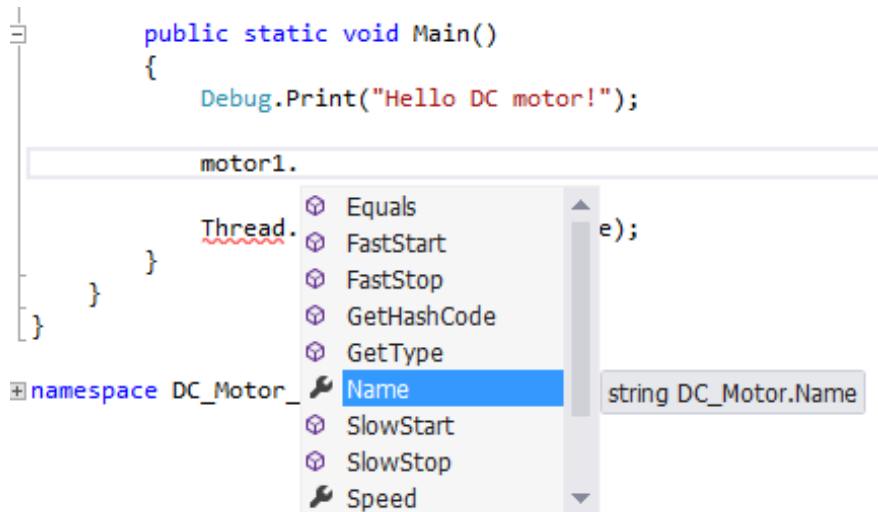
```
static DC_Motor Motor1 = new DC_Motor(
public static void Main()
{
    Debug.Print("Hello World!");
    Thread.Sleep(Timeout.Infinite);
}
ce DC_Motor_Lib[...]
```

47. נמלא את הפרמטרים המועברים לבונה בפעולה בונה ונקבל את הקוד הבא:

```
namespace Advanced_DC_Motor
{
    public class Program
    {
        static DC_Motor motor1 = new DC_Motor(PWM_Channels.PD_12,
                                                Pins.GPIO_PIN_D_11,
                                                5000, 5000, 100, false);

        public static void Main()
        {
            Debug.Print("Hello World!");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

48. נשנה את הכיתוב המודפס על המסך ב `Debug.Print` לטקסט התואם את הפרויקט ונרשום מתחתיו את שם אובייקט המנוע (`motor1`) שכרגע יצרנו. כאשר נקיש על מקש הנקודה במקלדת (.) נראה מיד את רשימת ה"רכוש" של האובייקט עליו עבדנו עד כה.



49. נבחר, למשל את המאפיין Name של המנוע ונוכל להכניס אליו שם או תיאור מילולי של המנוע המדובר:

```
motor1.Name = "left";
```

50. לאחר מכן, בתוך ה Main() נחליף את ההשהיה האינסופית בלולאה אינסופית הבודקת את התפקוד של המנוע. ניתן להיעזר בקוד הבא או בדומה לו:

```
while (true)
{
    motor1.SlowStart(5000);
    Thread.Sleep(3000);

    motor1.Speed = 70;
    Debug.Print(" speed: 70");
    Thread.Sleep(3000);
    motor1.Speed = 40;
    Debug.Print(" speed: 40");
    Thread.Sleep(3000);

    motor1.SlowStop(5000);
    Thread.Sleep(3000);

    motor1.FastStart();
    Thread.Sleep(3000);

    motor1.FastStop();
    Thread.Sleep(3000);

    motor1.CW_Direction = !motor1.CW_Direction;
```

© BRK כל הזכויות שמורות. אין להעביר לצד שלישי ללא אישור בכתב מהחברה.

}

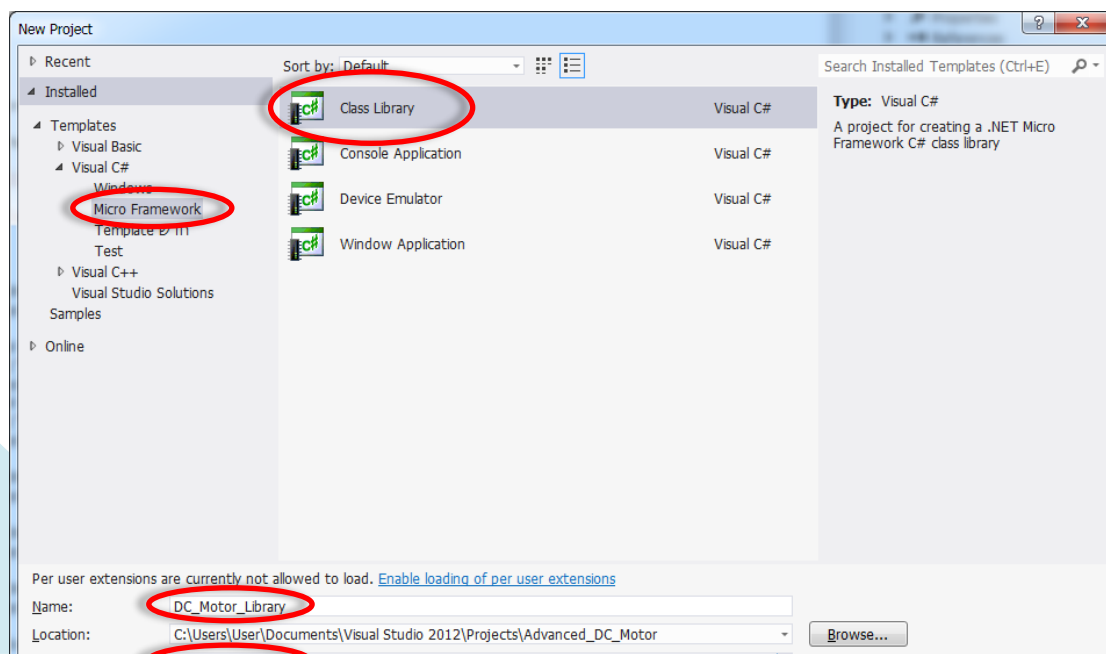
51. נצרוב את התוכנה לבקר ונוודא את הפעילות של המנוע ע"פ התוכנה, כלומר:

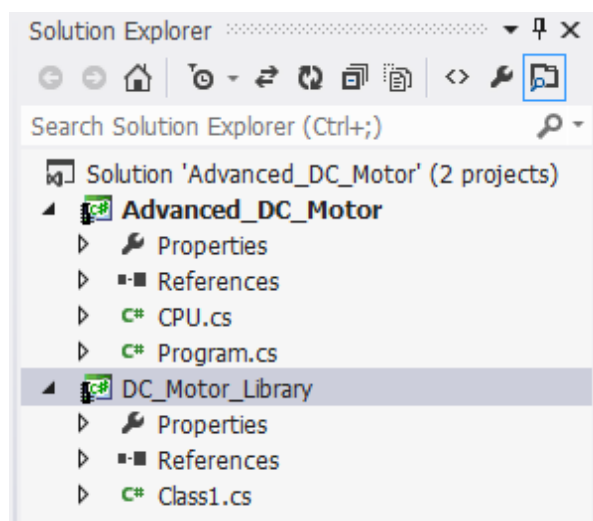
- a. מפעילה את המנוע במהירות הולכת וגדלה במשך 5 שניות
- b. מפעילה את המנוע במהירות סיבוב מרבית במשך 3 שנית
- c. מפעילה את המנוע ב 70% מהמהירות סיבוב המרבית שלו במשך 3 שנית
- d. מפעילה את המנוע ב 40% מהמהירות סיבוב המרבית שלו במשך 3 שנית
- e. עוצרת את המנוע בהדרגתיות במשך 5 שניות
- f. ממתנה עם מנוע כבוי 3 שניות
- g. מפעילה את המנוע בהאצה מרבית ומשאירה אותו פועל במשך 3 שניות
- h. עוצרת את המנוע במהירות האפשרית (ללא "בלמים") וממתנה 3 שניות
- i. מחליפה את כיוון הסיבוב של המנוע וחוזרת על כל הסעיפים מהתחלה עבור כיוון סיבוב חדש.

52. לאחר שווידאנו כי התוכנה פועלת באופן תקין, ניתן ליצור את הקובץ DLL עבור המחלקה של המנוע.

53. נלחץ בתפריט המשימות על תפריט ה File ונבחר את אפשרות New project המאפשרת פיתח פרויקט חדש, אך בחלון שיפתח נבצע מספר שינויים:

- a. נבחר את סוג הפרויקט Class Library שבתוך ה Micro Framework
- b. ניתן לפרויקט שם של ספריה
- c. נוסיף את הפרויקט החדש אותו אנו יוצרים אל ה Solution הקיים.





54. נוכל לראות שהפרויקט החדש התוסף לsolution הקיים בחלונת Solution Explorer:

55. נפתח את הקובץ Class1.cs המופיע בפרויקט החדש שפתחנו.

56. כרגע הקובץ מכיל רק את:

```
using System;
using Microsoft.SPOT;

namespace DC_Motor_Library
{
    public class Class1
    {
    }
}
```

57. נחליף (העתק + הדבק) את המחלקה הקיימת בו (מודגשת במרקר צהוב) במחלקה שכתבנו בפרויקט הקודם שנמצא באותו ה solution.

58. אם כן, נעתיק (נסמן את כל הקוד של המחלקה ונלחץ בו זמנית על המקשים Ctrl+c) את המחלקה `DC_Motor` שכתבנו בקובץ `Program.cs` שבפרויקט `Advanced_DC_Motor`

59. נדביק (הקשה בו זמנית על מקשי המקלדת Ctrl+v) אותה **במקום** המחלקה הקיימת כרגע בקובץ `Class1.cs` שבפרויקט `DC_Motor_Library` והמודגשת **בצהוב** בספר.

60. מיד נראה כמה שגיאות עליהם מצביע ה `Visual studio`:

```
using System;
using Microsoft.SPOT;

namespace DC_Motor_Library
{
    public class DC_Motor
    {
        private string name;
        private uint stopTime, startTime;
        private uint speed;
        private bool direction;
        private Cpu.PWMChannel pwm_ch;

        private PWM pwm1;
        private OutputPort out1;

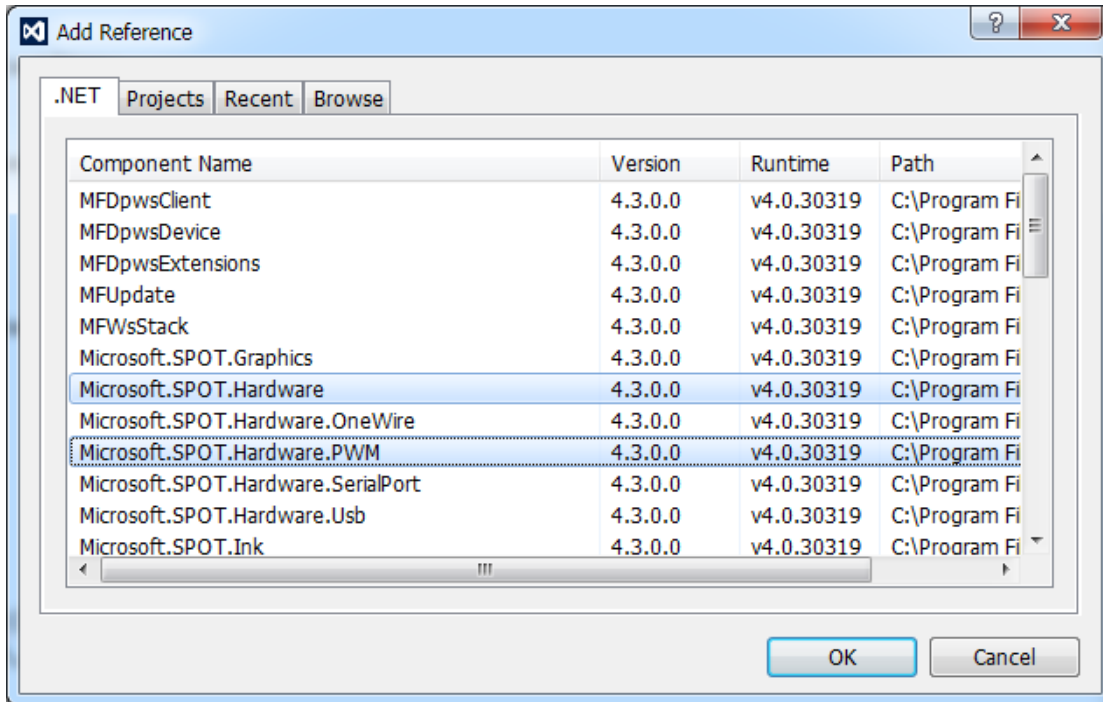
        public DC_Motor(Cpu.PWMChannel FirstPin, Cpu.Pin SecondPin)
        {
            name = "Motor";
            stopTime = 5000;
            startTime = 5000;
            speed = 100;
            pwm_ch = FirstPin;
            direction = false;

            out1 = new OutputPort(SecondPin, direction);
            pwm1 = new PWM(pwm_ch, 100, 0.0, false);
            pwm1.Start();
        }
    }
}
```

הערה: התצלום מסך שבסעיף זה אינו מכיל את כל המחלקה, כמובן, אלא מהווה המחשה בלבד של השגיאות עליהם מצביעה סביבת הפיתוח.

61. הסיבה לכך הם ה `References` וה `using` אותם יש להוסיף לפרויקט זה.

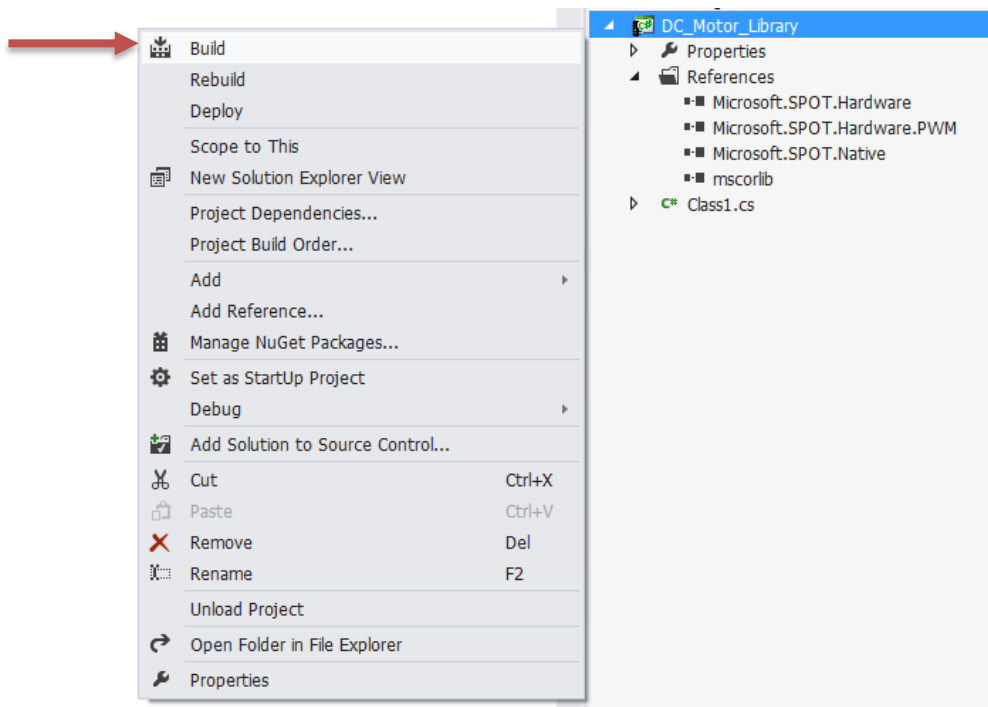
62. נוסיף את ה `Reference` שהיו בפרויקט `Advanced_DC_Motor`:



63. נוסף את ה `using` הנדרשים בקובץ `Class1.cs`:

```
using Microsoft.SPOT.Hardware;
using System.Threading;
```

64. נלחץ מקש ימני על הפרויקט `DC_Motor_Library` ונבחר את האפשרות `Build` שתצור את הקובץ `DLL` עבור אובייקט המנוע.



65. בחלונת ה `Output` שתוצג ניתן יהיה לראות את הסיומ המוצלח של הפעולה:

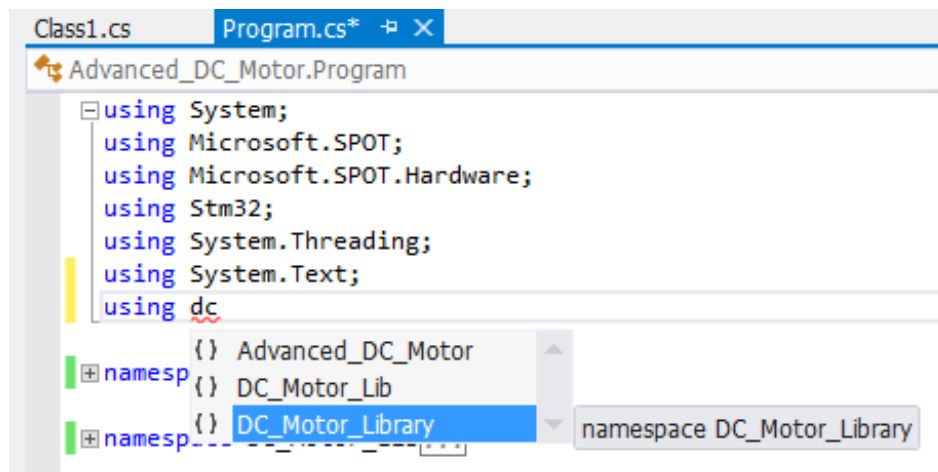
```
==== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

© BRK כל הזכויות שמורות. אין להעביר לצד שלישי ללא אישור בכתב מהחברה.

66. בשלב זה יש להוסיף קובץ DLL זה ל References של הפרויקט Advanced_DC_Motor.
 67. נלחץ מקש ימני על References של Advanced_DC_Motor (ולא של DC_Motor_Library!),
 נבחר את האופציה Add reference ובלשונית Browse נגיע לקובץ DLL שנוצר שהנתיב אליו
 מופיע בחלונית ה Output בשורה שלפני ההודעה על הסיום המוצלח של פעולת ה Build.
 לדוגמא:

```
DC_Motor_Library -> C:\Users\User\Documents\Visual Studio2012\Projects\  
Advanced_DC_Motor\DC_Motor_Library\bin\Debug\DC_Motor_Library.dll
```

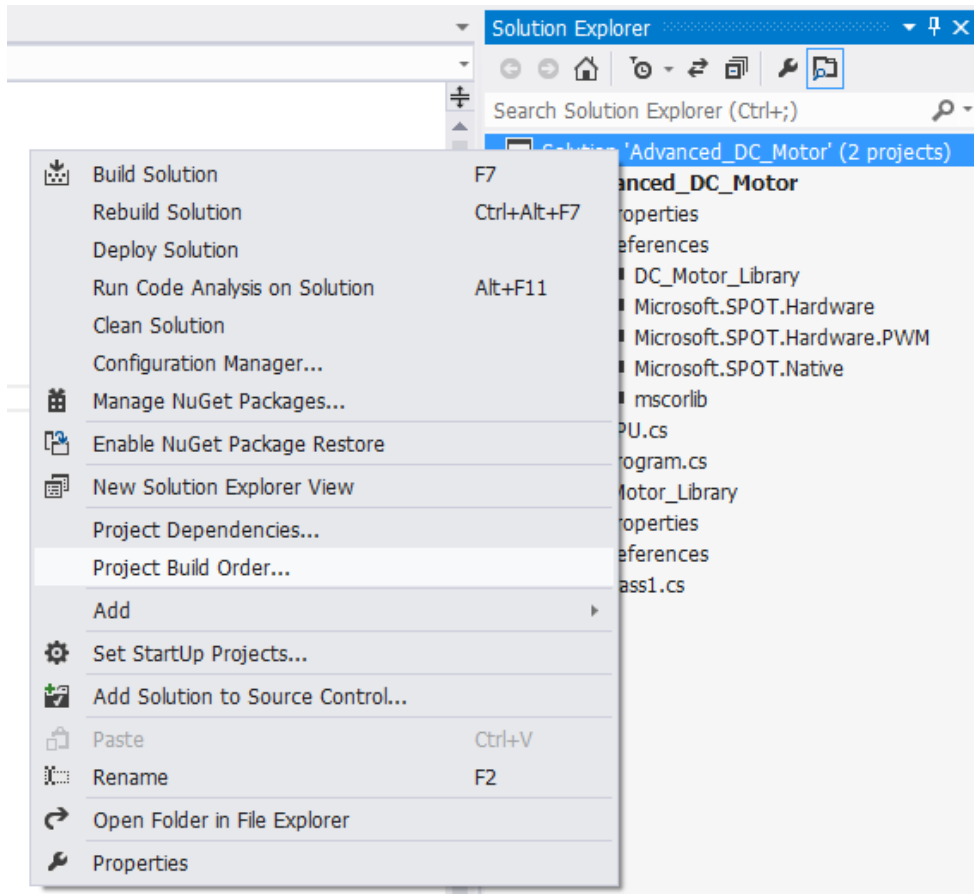
68. נשנה את ה using בתוך הקובץ Program.cs המפנה לאובייקט המנוע בתוך הפרויקט
 Advanced_DC_Motor (using DC_Motor_Lib;) ל using המפנה לאובייקט המנוע
 הנלקח מתוך קובץ DLL שה reference אליו צורף לפרויקט. שימו לב שבזמן ההקלדה
 סביבת הפיתוח עוזרת ומשלימה את השם המתאים.



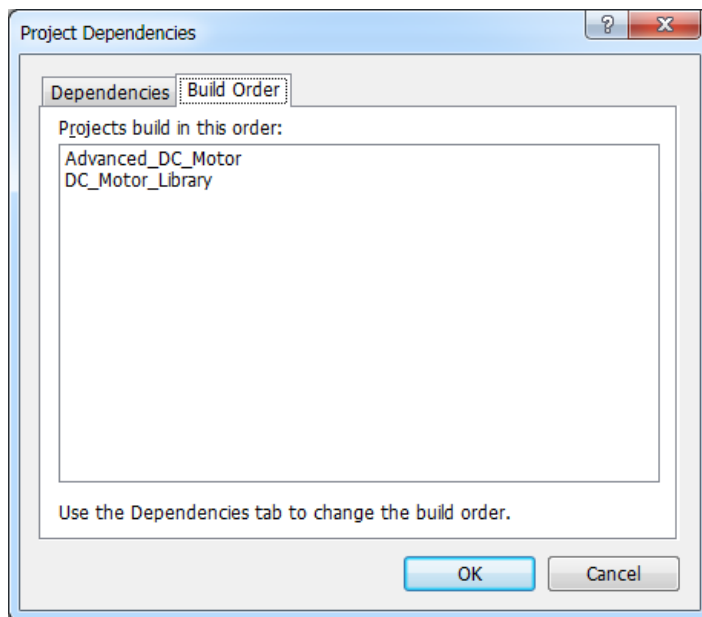
```
Class1.cs Program.cs*
Advanced_DC_Motor.Program
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using Stm32;
using System.Threading;
using System.Text;
using dc
{ } Advanced_DC_Motor
+ namespace { } DC_Motor_Lib
+ namespace { } DC_Motor_Library namespace DC_Motor_Library
```

69. נשנה את סדרי פעולת ה Build של ה solution בכדי שקודם הפעולה תתבצע על הפרויקט
 המכיל את אובייקט המנוע והיוצר את קובץ ה DLL ורק לאחר מכן הפעולה תתבצע על
 הפרויקט המשתמש בקובץ זה.

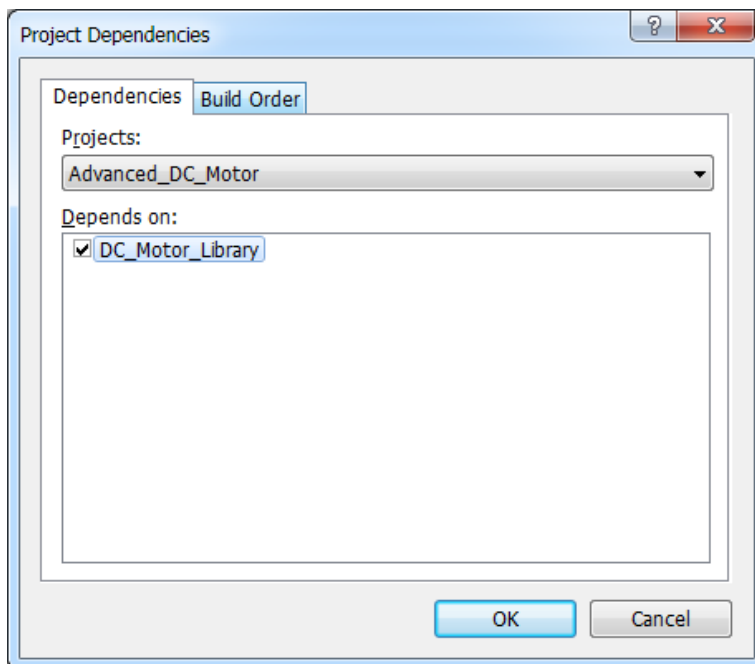
70. לשם כך, נלחץ עם המקש הימני של העכבר על ה solution ובתפריט שתפתח נבחר את
 האפשרות Project Build Order...



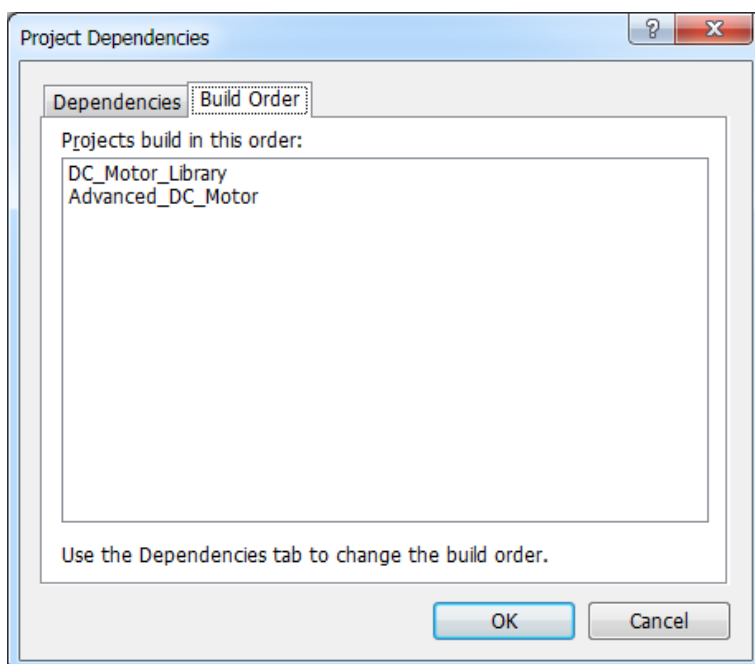
71. יפתח החלון המופיע בתצלום, בו ניתן לראות שעד כה פעולת ה Build התבצעה קודם על הפרויקט Advanced_DC_Motor ורק לאחר מכן על הפרויקט DC_Motor_Library.



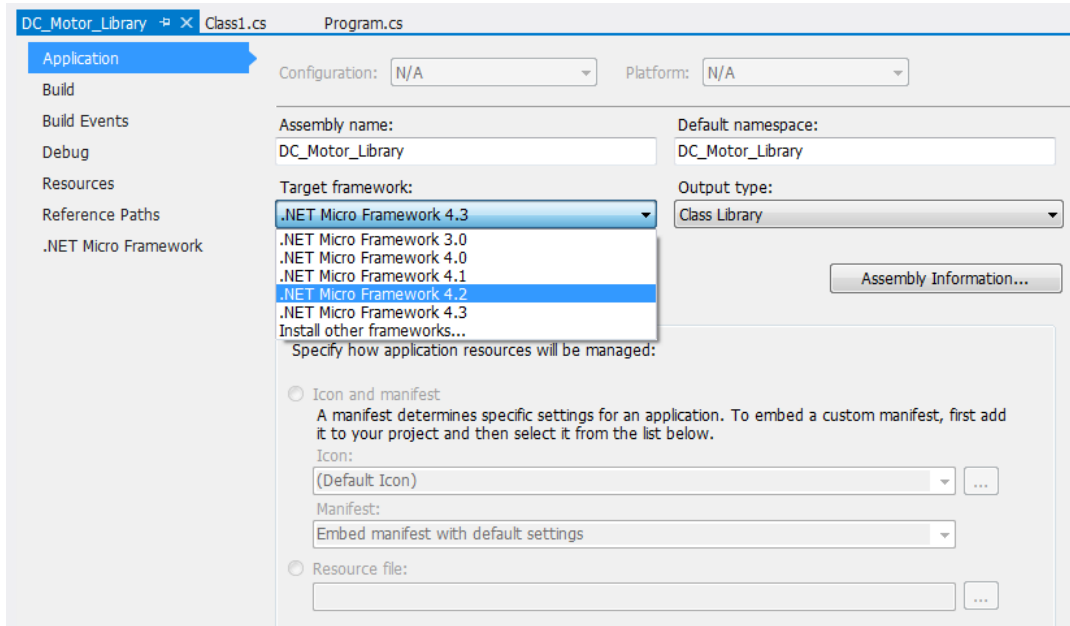
72. מאחר ואנו רוצים לשנות מצב זה, נלחץ עם העכבר על הלשונית Dependencies ונסמן V בתוך התיבה של DC_Motor_Library כמתואר בתצלום הבא:



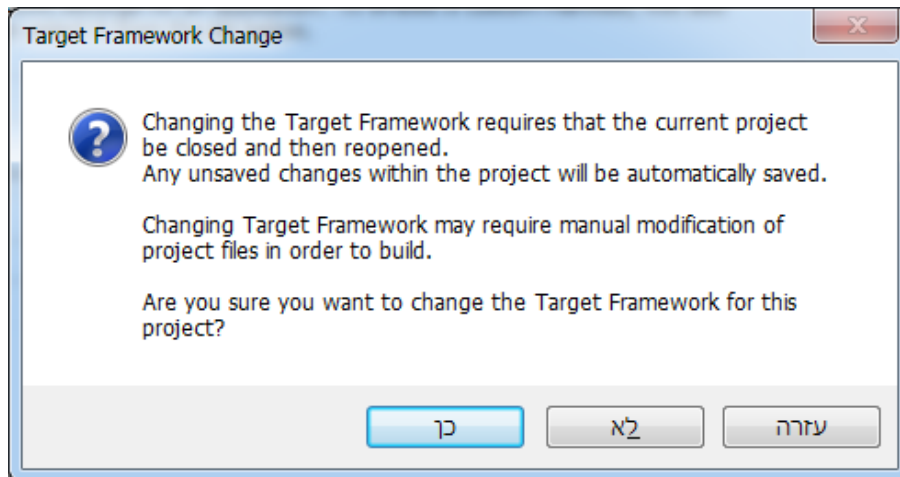
73. נחזור בחזרה ללשונית Build Order ונראה שסדר הפרויקטים השתנה.



74. לבסוף נחליף את הגרסה של .Net Micro Framework של הפרויקט היוצר את ה DLL (DC_Motor_Library) מגרסה 4.2 ל 4.3 הנתמכת ע"י ערכת הפיתוח. נעשה זאת ע"י כניסה למאפייני הפרויקט (DC_Motor_Library) וביצוע השינוי המדובר בלשונית ה Application, כמתואר בתצלום הבא:



75. יפתח חלון הבא המתריע על כך שהשינויים שלא שמורים יעבדו. יש ללחוץ על לחצן "כן".



76. יש לצרוב את התוכנה לבקר ולוודא שהיא פועלת בדיוק באותו אופן כפי שזה היה עם אובייקט המנוע הממוקם בתוך הפרויקט ולא מחוצה לו.

77. לאחר ביצוע האימות וכל הבדיקות, ניתן למחוק את כל ה namespace של DC_Motor_Lib שבתוך קובץ ה Program.cs שבפרויקט Advanced_DC_Motor מאחר ויותר ולא מעשה בו שימוש כלל.

78. ניתן ואף מומלץ להתנסות בלהוסיף constructors, properties, methods ולהרחיב את המחלקה כראות עינכם.

79. על דרך יצירת ספרייה זו, ניתן ליצור ספריות נוספות עבור כל רכיב הקיים בשוק.

80. בהצלחה ושתהיה עבודה פורייה!