Texas Instruments MSP430 Programlama

MSP430 Mikrodenetleciyi Donanımları

Tüm Yönleriyle Code Composer Studio

IAR ile MSP430 Programlamak

Port Giriş/Çıkış, Buton ve Döngü İşlemleri

Kesmeler

TimerA1 Sayıcı ve PWM İşlemleri

MSP430 ile LCD Uygulaması

MSP430 ve 74HC595

SD16 ADC Modülü

ΣΔ Modülasyon Tanım ve Detayları

MSP430 Dahili Sıcaklık Sensörü Okuma

EZ430-F2013 Donanımı

Fırat Deveci

Ön Söz

2008'de PIC ve CCS C ile başladığım mikrodenetleyici programlama macerama 2009 yılınca C'nin gücünü görerek Hi-Tech ve PIC konusunda, daha sonrasında ise 2009 yılında WinAVR ile ilgili bir kitap yazarak devam ettim.

Bu süre zarfı boyunca Microchip PIC16, PIC18, PIC24, dsPIC30 ve dsPIC33F serilerinin yanında, NXP firmasının LPC serisi ARM mikrodenetleyicilerle sıkça çalışma fırsatı buldum. Özellikle mikrodenetleyiciler ile uğraşan herkesin son durağı sayılabilecek RTOS kullanımının yanında C dilinin gücünü bir kez daha gördüm.

Son dönemlerde ise güç elektroniği konusunda kendimi geliştirirken, mikrodenetleyicilerin oynadığı kritik görevlere bizzat şahit oldum.

Bunların yanında özellikle analog ve sinyal işleme konusunda uzman olan Texas Ins. firmasının dünyada kullanımının yaygınlaşmasını istediği ve çok düşük güç tüketimiyle yola çıkan MSP430'un çeşitli vesilelerle kullanıcılara dağıtılması ve 16bit mikrodenetleyicilere yeni adım atmak isteyenlere bir fırsat doğması benim de bu konuda çalışma yapmama vesile oldu.

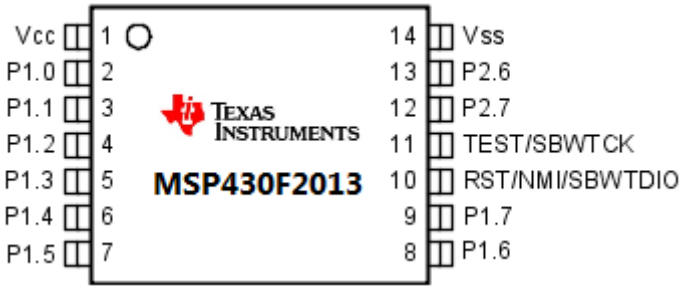
Yukarıdaki düşünceler ışığında hazırlanan kitap, sizlere MSP430 ile ilk adımları atmayı öğretirken, diğer mikrodenetleyicilerde bulunmayan bazı özellikler hakkında bilgi sahibi yapmanın yanı sıra, Code Composer Studio'nun kullanımını da sizlere aktaracaktır.

Fırat Deveci
Ağustos 2011
fxdev@fxdev.org

Elektrik ve elektroniğe gönül veren herkese…

BÖLÜM 1 – MSP430 GENEL ÖZELLİKLERİ VE OSC AYARLARI

1.1) MSP430F2013'e Genel Bakış



Şekil 1 - MSP430F2013 Genel Yapı

Hazırlanan not boyunca kullanılacak olan MSP430F2013'ün genel özellikleri tablo 1'den görülebilir.

Tablo 1- MSP430F2013 Özellikleri

Frequency (MHz)	16
Flash (Kb)	2
SRAM (byte)	128
GPIO	10
Timers – 16bit	1
Watchdog	Var
Brown Out Reset	Var
USI	Var
Comparators	Yok
Temp Sensor	Var
ADC	16bit Sigma Delta
ADC Channel	4 adet

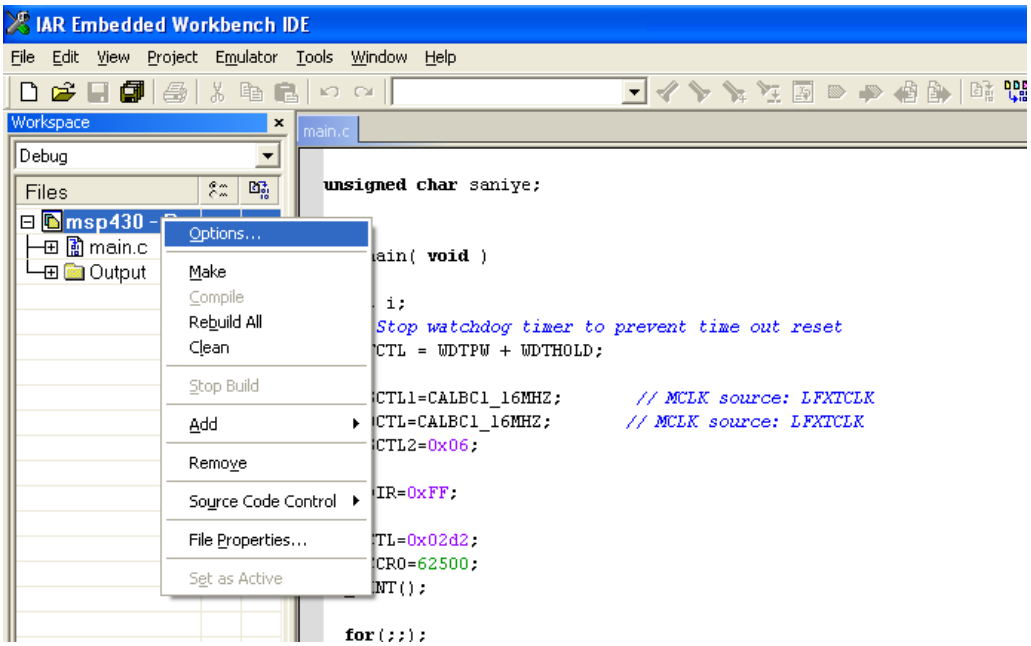
1.2) MSP430 Hakkında Genel Bilgi ve Dikkat Edilmesi Gereken Hususlar

MSP430 Texas Instrument firmasının ürettiği çok düşük güç tüketimiyle ön plana çıkan bir mikrodenetleyicidir. 16-bit RISC mimariye sahiptir, içerisinde I2C, SPI, USART, ADC gibi klasik haline gelmiş bir çok modül bulundurmaktadır.

MSP430 programlamak için ***IAR Embedded Workbench*** üzerine kurulu MSP430 derleyicisi ya da ***Code Composer Studio*** kullanılmaktadır. Bu iki program da TI sitesinden ücretsiz indirilerek kullanılabilmektedir.

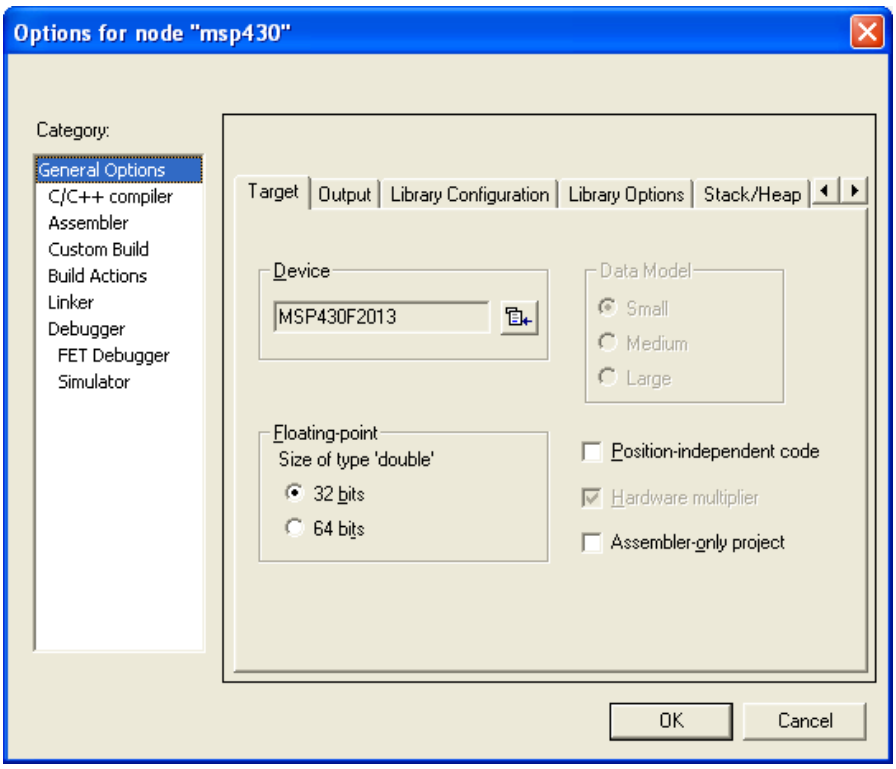
Code Composer Studio'nun kullanımı basit olduğundan, proje oluşturma ve yaratma ayarlarına değinilmeyecektir.

IAR Embedded Workbench programı ile debug işlemleri yapılacağında özellikle aşağıdaki noktalara dikkat etmek gerekmektedir.



Şekil 2 - MSP430 Option ayarı

Öncelikle şekil 2'de görülen kısımdan options sekmesine girmek gerekir.



Şekil 3 - Program Ayarları

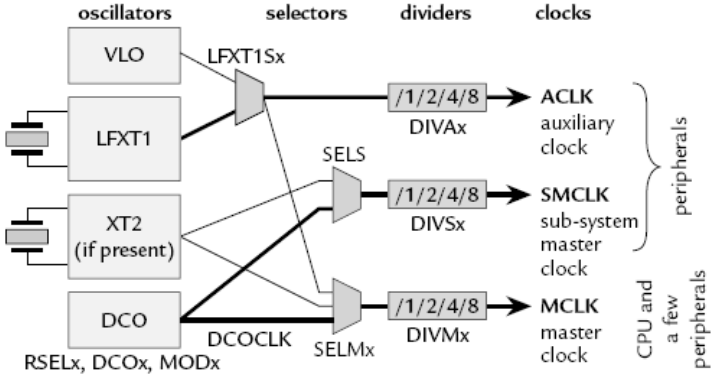
Şekil 3'te görülen **Device** kısmından öncelikle programlayacağımız işlemciyi seçmeli, daha sonra **Linker** kısmından **Other->Output Format->Intel Standart** seçilmelidir. Bu kısım önemlidir, çünkü bu ayar yapılmazsa MSP430 çalışmamaktadır. Daha sonra Debugger kısmından FET Debugger'ı, FET debugger kısmından da **TI USB-IF**'ı seçip, **Automatic** kısmının yanındaki boş kareye basıp MSP430'un takılı olduğu **COM** portu seçilmeli ve **OK** denmelidir.

1.3) MSP430 OSC Ayarları

MSP430 kendi içerisinde kullanacağı 3 adet saat kaynağına sahiptir. Bunlar;

Master Clock (MCLK) : CPU ve birkaç çevresel birim tarafından kullanılır.
Sun-system Master Clock (SMCLK) : Çevresel birimler için kullanılır.
Auxiliary Clock (ACLK) : Çevresel birimler için kullanılır.

Bu birimlerin nasıl bağlandıkları ve çıkışa nasıl yansıdıkları şekil 4'te görülebilir.



Şekil 4 - Osilatör Kaynakları

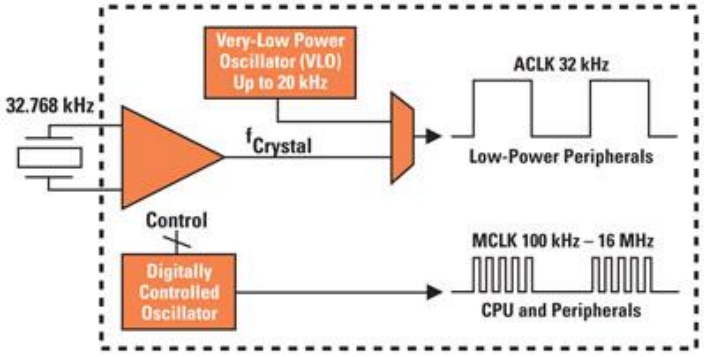
Şekil 4'te görülen MCLK veya SMCLK genellikle MHz düzeylerinde iken, ACLK birimi ise genellikle kHz mertebesindedir. MSP430'a saat sinyali üreten 3 birim bulunur:

Low-or-high-freq-crystal-osc (LFXT1) ***:*** Genelde 32kHz civarındadır. Eğer dıştan kristal kullanılacaksa MSP430 ile senkronize yapılması gerekmektedir.

Internal Very-Low-Power-Low-Freq-Osc (VLO): Genellikle MSP430f2xx modellerinde yaygındır. Kristal kullanılmayacaksa LFXT1'in alternatifidir. Msp430f2013 için hızı 12kHz'dir.

Digitally Controlled Osc (DCO) ***:*** Tüm MSP430 entegrelerinde mevcuttur. 1us içinde çalışmaya başlayan, yazılımsal olara kontrol edilebilen, RC osilatördür.

Frekans ayarlama kısmı DCOCTL, BCSCTL1-3 registerleri ile IFG1 ve IE2 bitleri ile kontrol edilir.



Şekil 5 - L or H Freq. Crystal Osc

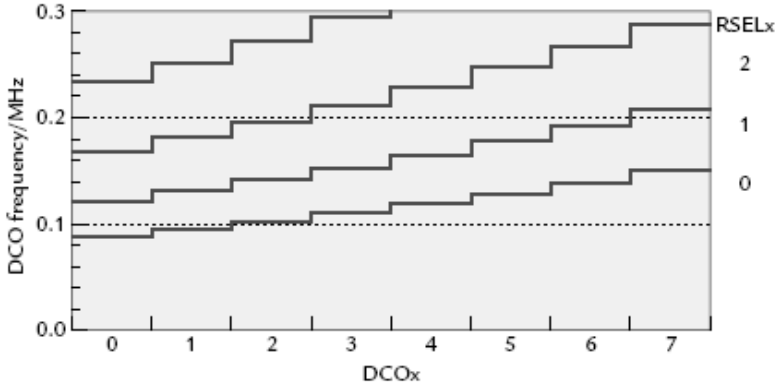
1.3.1) VLO

VLO, 12kHz hızında çalışan internal osilatördür ve LFXT1 yerine yeni modellerde kullanılır. MSP430F2013 datasheetinde LFXT1 modülü çalışırken **0.8uA**, VLO çalışırken ise **0.5uA** çekildiği söylenmiştir. VLO, dış etkenlere bağlı olarak 4kHz'den 20kHz'e kadar değişkenlik gösterebilmektedir. **Bunun için zamanın önem kazandığı devrelerde kullanılmaması gerekir.** Ayrıca VLO kullanıldığında LFXT1 modülü etkisiz kalacağından P2.6 ve P2.7 pinleri dijital I/O olurlar. Aşağıdaki örnekte ise VLO saat kaynağı olarak seçilmiş, P2.6 ve P2.7 dijital I/O yapılmıştır.

```
#include "msp430.h"                              // MSP430 başlık dosyası

void main( void )
{
      int i;

      WDTCTL = WDTPW + WDTHOLD;                  // Watchdog Timer'ı durdur.

      BCSCTL3|=LFXT1S_2;                         // ACLK, VLO'dan gelecek
      IFG1 &= ~OFIFG;                            // OSC hata bayrağı siliniyor

      __bis_SR_register(SCG1 + SCG0);            // DCO durduruluyor

      BCSCTL2 |= SELM_3 | DIVM_0;                // MCLK = LFXT1

      P1DIR=0xFF;                  // P1 portu çıkış olarak yönlendiriliyor

      for(;;)
      {
            P1OUT^=0xFF;
            // P1 çıkışı belirli bir gecikmeden sonra terslenecek
            i=6000;while(i--); // Belirli bir gecikme sağlanıyor.
      }
}
```

1.3.2) Digitally Controlled Oscillator (DCO)

DCO birimi mikrodenetleyici enerjilendikten 1-2us içinde çalışmaya başlar.
RSELx ile 0.09-20MHz aralığında range belirlenirken,
DCOx ile de RSELx ile belirlenen aralıktan %8 oranında frekans seçer.



Şekil 3 – RSELx ve DCOx

RSELx ve DCOx'in seçtiği frekans değeri ise aşağıdaki formülden hesaplanabilir.

$$f_{\text{average}} = \frac{32 f_{\text{DCO}} f_{\text{DCO+1}}}{\text{MOD} \times f_{\text{DCO}} + (32 - \text{MOD}) \times f_{\text{DCO+1}}}.$$

$$T_{\text{average}} = \frac{\text{MOD} \times T_{\text{DCO+1}} + (32 - \text{MOD}) \times T_{\text{DCO}}}{32}.$$

Bu hesapla uğraşılmaması için derleyicinin içinde aşağıdaki tanımlamalar yapılmıştır.

```
BCSCTL1=CALBC1_1MHZ      // Set range
DCOCTL =CALBC1_1MHZ      // Set DCO step and modulation
```

CALBC1 kodu 1MHZ, 8MHZ, 12MHZ ve 16MHZ için kullanılabilir.

Tüm bunları ayarladıktan sonra saat kaynak seçimi, div işlemlerinin ayarları gibi işlemler için yine BCSCTL1-3 ayarlanmalıdır.

Eğer yukarıdaki kalıplar kullanılmak istenmiyorsa aşağıdaki rakamlar kullanılarak yine istenilen frekans değeri elde edilebilir.

freq	RSELx	MOD	DOCx
4MHz	1010	10111	101
8MHz	1101	00000	100
20MHz	1111	11010	110

1.4) Status Register Kullanarak Saat Modülünü Kontrol Etmek

Daha çok düşük güç operasyonları ile ilgili kısımdır.
CPUOFF : MCLK'yi deaktif eder, CPU ve MCLK'yi kullanan dış elemanlar durur.
SCG1 : SMCLK'yi deaktif eder.
SCG0 : DC generatoru kapatır.
OSCOFF : VLO ve LFXT1'i kapatır.

1.5) Düşük Güç Tüketimi

MSP403 işlemcisinin çekeceği güç yazılımsal olarak ayarlanabilmektedir.

Active Mode : MCLK'yi deaktif eder, CPU ve MCLK'yi kullanan dış elemanlar durur.
LPM0 : SMCLK'yi deaktif eder.
LPM3 : DC generatoru kapatır.
LPM4 : VLO ve LFXT1'i kapatır.

instrinsics.h kütüphanesi tanımlanarak,

__low_power_mode_3() şeklinde o istenilen moda girilebilir ve
__low_power_mode_off_on_exit() ile tekrar aktif moda dönülebilir.

Ayrıca;

__bis_SR_register(LPM3_bits) ile ayarlamalar yapılacağı gibi
__bir_SR_register_on_exit(LPM3_bits) ile de bu moddan çıkış yapılabilir.

1.6) Frekans Değerlerini Ayarlamak İçin Faydalı Tablo

	#define DCOCTL_ (0x0056) /* DCO Clock Frequency Control */ DEFC(DCOCTL , DCOCTL_) #define BCSCTL1_ (0x0057) /* Basic Clock System Control 1 */ DEFC(BCSCTL1 , BCSCTL1_) #define BCSCTL2_ (0x0058) /* Basic Clock System Control 2 */ DEFC(BCSCTL2 , BCSCTL2_) #define BCSCTL3_ (0x0053) /* Basic Clock System Control 3 */ DEFC(BCSCTL3 , BCSCTL3_)
DCOCTL	#define MOD0 (0x01) /* Modulation Bit 0 */ #define MOD1 (0x02) /* Modulation Bit 1 */ #define MOD2 (0x04) /* Modulation Bit 2 */ #define MOD3 (0x08) /* Modulation Bit 3 */ #define MOD4 (0x10) /* Modulation Bit 4 */ #define DCO0 (0x20) /* DCO Select Bit 0 */ #define DCO1 (0x40) /* DCO Select Bit 1 */ #define DCO2 (0x80) /* DCO Select Bit 2 */
BCSCTL1	#define RSEL0 (0x01) /* Range Select Bit 0 */ #define RSEL1 (0x02) /* Range Select Bit 1 */ #define RSEL2 (0x04) /* Range Select Bit 2 */ #define RSEL3 (0x08) /* Range Select Bit 3 */ #define DIVA0 (0x10) /* ACLK Divider 0 */ #define DIVA1 (0x20) /* ACLK Divider 1 */ #define XTS (0x40) /* LFXTCLK 0:Low Freq. / 1: High Freq. */ #define XT2OFF (0x80) /* Enable XT2CLK */ #define DIVA_0 (0x00) /* ACLK Divider 0: /1 */ #define DIVA_1 (0x10) /* ACLK Divider 1: /2 */ #define DIVA_2 (0x20) /* ACLK Divider 2: /4 */ #define DIVA_3 (0x30) /* ACLK Divider 3: /8 */

BCSCTL2

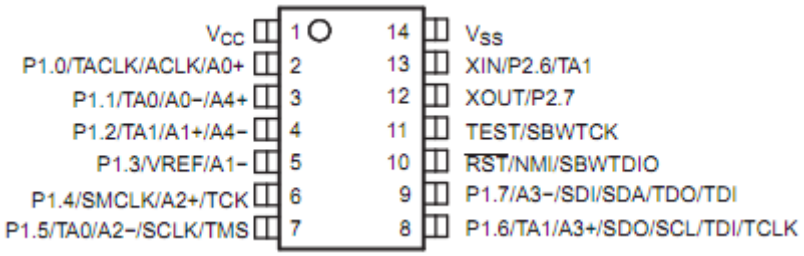
```
#define DIVS0          (0x02)  /* SMCLK Divider 0 */
#define DIVS1          (0x04)  /* SMCLK Divider 1 */
#define SELS           (0x08)  /* SMCLK Source Select 0:DCOCLK / 1:XT2CLK/LFXTCLK
*/
#define DIVM0          (0x10)  /* MCLK Divider 0 */
#define DIVM1          (0x20)  /* MCLK Divider 1 */
#define SELM0          (0x40)  /* MCLK Source Select 0 */
#define SELM1          (0x80)  /* MCLK Source Select 1 */

#define DIVS_0         (0x00)  /* SMCLK Divider 0: /1 */
#define DIVS_1         (0x02)  /* SMCLK Divider 1: /2 */
#define DIVS_2         (0x04)  /* SMCLK Divider 2: /4 */
#define DIVS_3         (0x06)  /* SMCLK Divider 3: /8 */

#define DIVM_0         (0x00)  /* MCLK Divider 0: /1 */
#define DIVM_1         (0x10)  /* MCLK Divider 1: /2 */
#define DIVM_2         (0x20)  /* MCLK Divider 2: /4 */
#define DIVM_3         (0x30)  /* MCLK Divider 3: /8 */

#define SELM_0         (0x00)  /* MCLK Source Select 0: DCOCLK */
#define SELM_1         (0x40)  /* MCLK Source Select 1: DCOCLK */
#define SELM_2         (0x80)  /* MCLK Source Select 2: XT2CLK/LFXTCLK */
#define SELM_3         (0xC0)  /* MCLK Source Select 3: LFXTCLK */
```

BCSCTL3

```
#define LFXT1OF        (0x01)  /* Low/high Frequency Oscillator Fault Flag */
#define XT2OF          (0x02)  /* High frequency oscillator 2 fault flag */
#define XCAP0          (0x04)  /* XIN/XOUT Cap 0 */
#define XCAP1          (0x08)  /* XIN/XOUT Cap 1 */
#define LFXT1S0        (0x10)  /* Mode 0 for LFXT1 (XTS = 0) */
#define LFXT1S1        (0x20)  /* Mode 1 for LFXT1 (XTS = 0) */
#define XT2S0          (0x40)  /* Mode 0 for XT2 */
#define XT2S1          (0x80)  /* Mode 1 for XT2 */

#define XCAP_0         (0x00)  /* XIN/XOUT Cap : 0 pF */
#define XCAP_1         (0x04)  /* XIN/XOUT Cap : 6 pF */
#define XCAP_2         (0x08)  /* XIN/XOUT Cap : 10 pF */
#define XCAP_3         (0x0C)  /* XIN/XOUT Cap : 12.5 pF */

#define LFXT1S_0       (0x00)  /* Mode 0 for LFXT1 : Normal operation */
#define LFXT1S_1       (0x10)  /* Mode 1 for LFXT1 : Reserved */
#define LFXT1S_2       (0x20)  /* Mode 2 for LFXT1 : VLO */
#define LFXT1S_3       (0x30)  /* Mode 3 for LFXT1 : Digital input signal */

#define XT2S_0         (0x00)  /* Mode 0 for XT2 : 0.4 - 1 MHz */
#define XT2S_1         (0x40)  /* Mode 1 for XT2 : 1 - 4 MHz */
#define XT2S_2         (0x80)  /* Mode 2 for XT2 : 2 - 16 MHz */
#define XT2S_3         (0xC0)  /* Mode 3 for XT2 : Digital input signal */
```

BÖLÜM 2 – MSP430 GİRİŞ ÇIKIŞ AYARLARI

MSP430F2013 denetleyicisinin pin yapısı aşağıdaki şekilde gösterilmiştir.



Şekil 4 – MSP430F2013 Pin Yapısı

Şekilde de görüleceği üzere pin sayısının düşük olmasından dolayı MSP430'da bir pine birden çok görev yüklenmiştir.

2.1) Pin Yönlendirmeleri ve Kullanımı

Diğer mikroişlemcilerden farklı olarak MSP430'da pin görevlerini özel olarak, diğer registerlerden bağımsız seçmek mümkündür.
Genel anlamda ise pin yönlendirmelerini sağlayan registerler ve görevleri kısaca şöyledir.

- **PxIN** : Port okuma registeridir. Sadece okuma yapılabilir.
- **PxOUT** : Port çıkış registeridir. Okuma ve yazma yapılabilir.
- **PxDIR** : Port yönlendirmesidir.
  Bit 1 olduğunda çıkış, 0 olduğunda giriştir.
  Bu özellik itibari ile AVR'lere benzemektedir.
- **PxREN** : Pull-up direnç ekleme ya da çıkarma registeridir.
  1 olduğunda ilgili pine pull-up direnci bağlamaktadır.
- **PxSEL VE PxSEL2** : Pinlerin birincil ve ikincil görevlerini seçme registerleridir.
  Özellikleri aşağıdaki tablo 2'de belirtilmiştir.

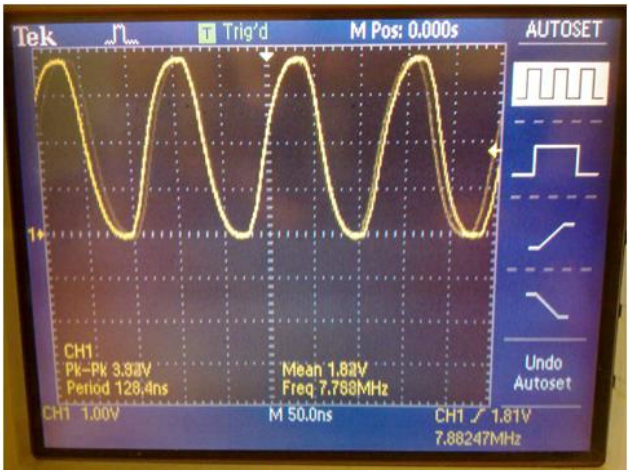
Tablo 2 - Pin Görevlerinin Tanımı

PxSEL2	PxSEL	Pin Function
0	0	I/O function is selected.
0	1	Primary peripheral module function is selected.
1	0	Reserved. See device-specific data sheet.
1	1	Secondary peripheral module function is selected.

İlk örneğimizde pinlerin birinci görevlerini kullanarak P1.0 ve P1.4'ün birincil görevleri olan sırasıyla ACLK ve SMCLK kristal frekanslarını bu bacaklardan görelim.
Bunun için öncelikle **P1SEL=0x11** olmalıdır. Geri kalan program ise aşağıdaki gibidir.

```
//                MSP430F20xx
//             -----------------
//         /|\|                  |
//          | |                  |
//          --|RST               |
//            |                  |
//            |       P1.4/SMCLK|-->SMCLK = Default DCO // 8MHz şu anlık
//            |             P1.1|-->MCLK/10 = DCO/10 // 800kHz
//            |        P1.0/ACLK|-->ACLK = 12kHz
//

#include  <msp430.h>

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;         // Watchdog timer kapatılıyor

  // 1Mhz
  // BCSCTL1 = CALBC1_1MHZ;
  // DCOCTL  = CALDCO_1MHZ;
  // BCSCTL3= LFXT1S_2;

  // 8Mhz
  BCSCTL1= CALBC1_8MHZ;              // Kristal şu anlık 8MHz ayarlanıyor
  DCOCTL = CALDCO_8MHZ;
  BCSCTL3= LFXT1S_2;

  // 12Mhz
  // BCSCTL1 = CALBC1_12MHZ;
  // DCOCTL = CALDCO_12MHZ;
  // BCSCTL3= LFXT1S_2;

  // 16Mhz ayarlamak için
  // BCSCTL1 = CALBC1_16MHZ;
  // DCOCTL = CALDCO_16MHZ;
  // BCSCTL3= LFXT1S_2;

    P1DIR |= 0xFF;                 // P1 portu çıkış olarak yönlendiriliyor
    P1SEL |= 0x11;                 // P1.0 ACLK, P1.4 SMCLK oluyor

    for(;;)
    {
        P1OUT ^= 0x02;            // P1.1 pini her döngüde değiştiriliyor
    }
}
```

Resim 1’de P1.4’den çıkan sinyalin osiloskop şeklini görebilirsiniz.



Resim 1 – 8MHz Clock Sinyali

2.2) Port Kesmeleri

P1 ve P2 portları üzerlerindeki değişimlere kesme üretebilmektedirler. Ayrıca bu kesmeleri tüm port değil, teker teker kontrol edebilmek de mümkündür.
Bu kesmeyi kontrol eden registerler ve görevleri ise şöyledir.

- **P1IE, P2IE :** P1 veya P2 port değişim kesmesi aktif
- **P1IFG, P2IFG :** P1 veya P2 port değişim kesme bayrakları. Burada dikkat edilmesi gereken en önemli konu, eğer port değişim kesmesi aktif edilirse ve o andan sonra herhangi bir giriş çıkış, yönlendirme, porttan okuma yapma gibi işlemlerde bu bayraklar set edilir. Bunun için programda bu hususa dikkat edilmelidir.
- **P1IES, P2IES :** P1 veya P2'de hangi değişikliklerin kontrol edileceği belirtilir. Hangi bitin hangi tetiklemeyi kontrol ettiğini ise aşağıda görebilirsiniz.

Tablo 3 - Tetikleme Kontrolü

Note: Writing to PxIESx

Writing to P1IES, or P2IES can result in setting the corresponding interrupt flags.

PxIESx	PxINx	PxIFGx
0 → 1	0	May be set
0 → 1	1	Unchanged
1 → 0	0	Unchanged
1 → 0	1	May be set

MSP430'da, port değişimi kesme kontrolü için aşağıdaki yapı kullanılır:

```
#pragma vector=PORT1_VECTOR
__interrupt void isim(void)
```

#pragma vector=PORT1_VECTOR yerine *#pragma vector=PORT2_VECTOR* yazılarak P2'nin değişim kesmesi de gözlenebilir. Kesmenin çıkışında ise aynı PIC'te olduğu gibi kesme bayrağı temizlenmelidir.

MSP430'da kesmeleri açmak için ise **_EINT();** fonksiyonu kullanılır.

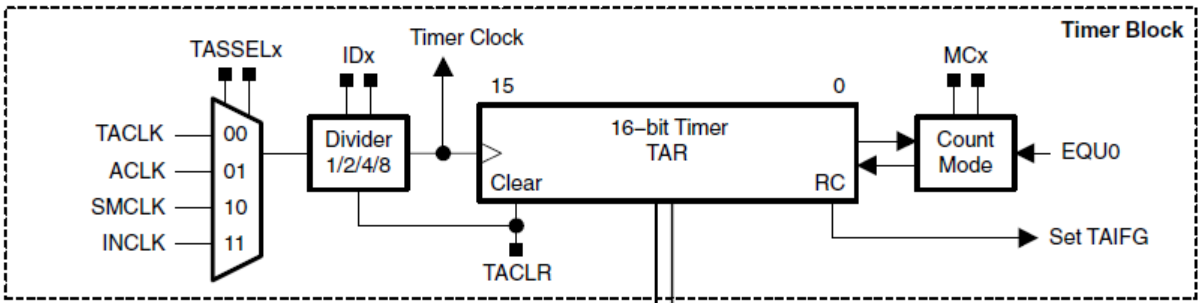
Kesmelerle ilgili bilgiler daha sonraki bölümlerde verilecektir.

BÖLÜM 3 – MSP430 TIMER AYARLARI

3.1) TimerA Birimi

TimerA birimi **16bit** ve **4 ayrı** moda çalışabilen bir timer birimidir. Saat kaynakları istenildiği gibi seçilebilir ve 2 ya da 3 adet capture/compare registeri içerir. Ayrıca PWM ayarlaması da yine bu sayıcı ile yapılabilir.

3.1.1) TimerA Sayıcı Modu



Şekil 6 - TimerA Sayıcı Kısmı

TimerA sayıcı kısmı Şekil 6'da görüleceği üzere farklı saat kaynaklarından beslenebilmektedir. Bu kaynağı seçen **TASSELx** bitleridir. Daha sonra seçilen saat kaynağı **IDx** ile bölüm oranına girer ve yükselen kenarda tetiklenmek üzere 16 bitlik sayıcı saymaya başlar. İstenilen anda **TACLR** biti 1 yapılarak **TAR** registerinin içeriği sıfırlanabilir. **MCx** ile sayıcı modları seçilir, sayıcı mod'a göre, dolduğunda kesme bayrağı çeker.

TimerA ayarlanırken zamanlayıcının kapalı olması MSP datasheetinde önerilmiştir. Ayrıca TimerA'nın çalışması için **MCx**'in mutlaka sıfırdan büyük olması gerekmektedir.

TimerA'yı kontrol eden registerler ise aşağıdaki gibidir.

TACTL, Timer_A Control Register

15	14	13	12	11	10	9	8
Unused						TASSELx	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
IDx		MCx		Unused	TACLR	TAIE	TAIFG
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	w–(0)	rw–(0)	rw–(0)

TASSELx	Saat Kaynağı Seçim Biti 00 – TACLK 01 – ACLK 10 – SMCLK 11 – INCLK
IDx	Divider (Bölücü) 00 – /1 01 – /2 10 – /4 11 – /8
MCx	Mode Kontrol 00 – Stop Mode 01 – Up Mode (TACCR0'a kadar sayacak) 10 – Continious Mode (0xFFFF'e kadar sayacak) 11 – TACCR0-0- TACCR0-0.. şeklinde sayacak
TACLR	TimerA'yı silen bit
TAIE	TimerA interrupt enable
TAIFG	TimerA interrupt flag

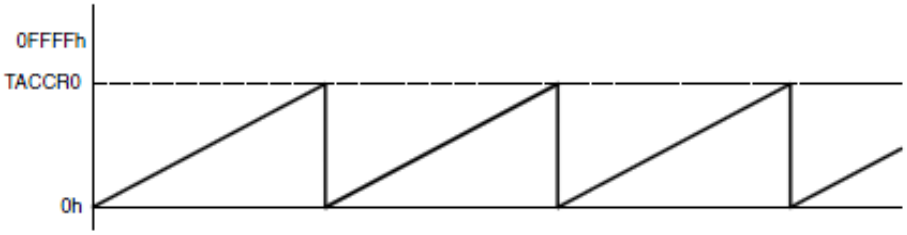
Diğer registerler için datasheet'e bakınız.

3.1.1.1) TimerA Stop Modu

Bu modda sayıcı durdurulur. Bir nevi sayıcının çalışmayacağı anlamına gelir.

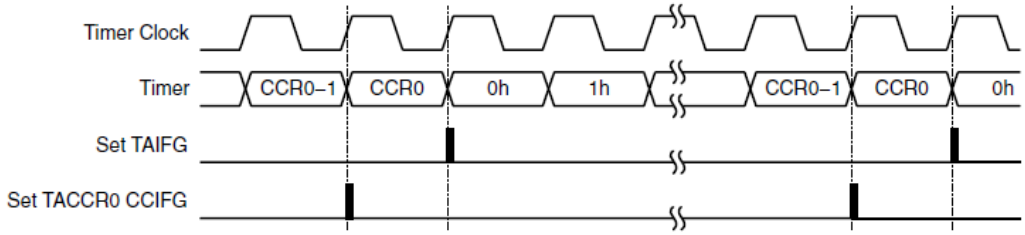
3.1.1.2) TimerA Up Modu

Bu modda sayıcı **TACCRx** registerine yüklenen değer kadar arttıktan sonra sayıcı sıfırlanır ve tekrar sayıcı kendini **TACCRx**'e kadar arttırır. Şekil 7'de bu modun çalışma şekli gösterilmiştir.



Şekil 7 – TimerA Up Modu

Sayıcı up moddayken sayıcı dolduğunda CCIFG, sıfır değerine ulaştığında ise TAIFG bayrağı set edilir. Bu kısım şekil 8'den çok net görülebilir.



Şekil 8 - Up Modda Set Edilen Bayraklar

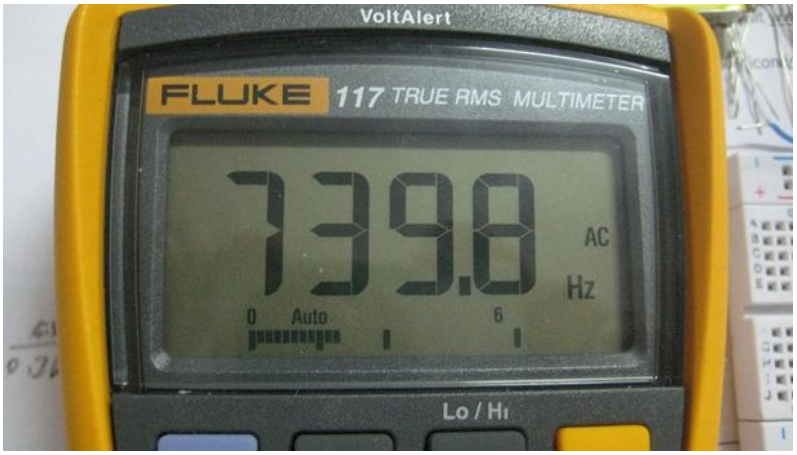
TimerA up mode çalışırken timer frekansı aşağıdaki formül ile hesaplanabilir.

$$f_{timer} = \frac{f_{clk}}{(Divider\ Value) \times (TACRRx)}$$

İlk örneğimizde 1500Hz'lik bir timer kuralım ve bunu P1.0'dan dışarıya alalım. Aşağıdaki program öncelikle TACCR0'a kadar sayıp kesme üretecektir. Her döngüde port döngüsü bir kere değişeceğinden pinden alacağımız frekans 750Hz olacaktır.

```
#include <msp430.h>              // MSP430 başlık dosyası

void main( void )
{
      WDTCTL = WDTPW + WDTHOLD; // Watchdog Timer'ı durdur.

      BCSCTL1= CALBC1_16MHZ;    // Kristal şu anlık 16MHz ayarlanıyor
      DCOCTL = CALDCO_16MHZ;
      BCSCTL3= LFXT1S_2;

      TAR=0x0000;       // TAR değeri sıfırlanıyor
      TACTL=0x02D6;
// SMCLK seçili, MOD1, 1:8, Interrupt Enable, Interrupt Flag temizleniyor
      TACCR0= 1334;
// TACCR0 değerine 16.000.000/8=2.000.000/1500=1.334 yükleniyor

      P1DIR=0xFF;   // P1 çıkış olarak ayarlanıyor

      for(;;)
      {
            if((TACTL&0x0001)==0x0001)     // TimerA kesmesi bekleniyor
            {
                  P1OUT^=0xFF;    // Kesme gelince P1 çıkışları tersleniyor
                  TACTL&=0xFFFE; // Kesme bayrağı temizleniyor
            }
      }
}
```

Yukarıdaki kodun çalışan halini resim 2'de görebilirsiniz. Yalnız bu tür yazım oldukça amatördür. Şimdi bunu gidermek için TimerA kesmesini kullanacağız.



Resim 2 – ~750Hz Clock Sinyali

```
#include <msp430.h>          // MSP430 başlık dosyası

// TimerA kesme vektörü
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT^=0xFF;               // Kesme gelince P1 çıkışları tersleniyor
}

void main( void )
{
  WDTCTL = WDTPW + WDTHOLD;   // Watchdog Timer'ı durdur.

  BCSCTL1= CALBC1_16MHZ;    // Kristal şu anlık 16MHz ayarlanıyor
  DCOCTL = CALDCO_16MHZ;
  BCSCTL3= LFXT1S_2;

  TAR=0x0000;              // TAR değeri sıfırlanıyor
  TACTL=0x02D6;
// SMCLK seçili, MOD1, 1:8, Interrupt Enable, Interrupt Flag temizleniyor
  TACCR0=1334;
  TACCTL0=CCIE;            // CCIFG interrupt'ı açılıyor

  P1DIR=0xFF;              // P1 çıkış olarak ayarlnıyor

  _EINT();                 // Genel kesmeler açılıyor

  for(;;);
}
```

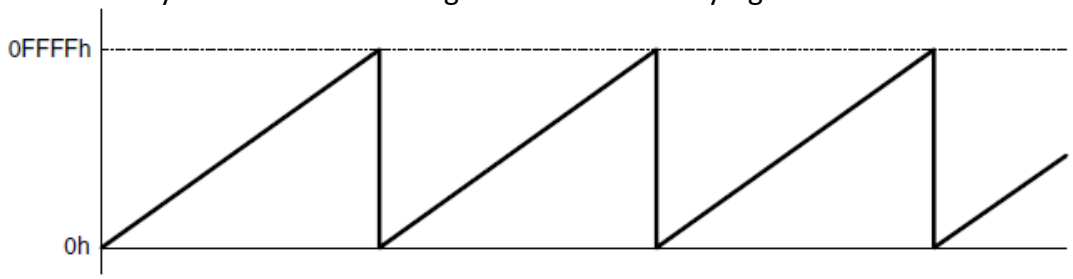
Bu örneğimizde ise TimerA biriminin saat kaynağını iç kristal olan 12KHz'den alıp bir saniyede bir led yakıp söndüren kodu yazacağız.

```
#include <msp430.h>          // MSP430 başlık dosyası

#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT^=0xFF;               // Kesme gelince P1 çıkışları tersleniyor
}

void main( void )
{
  WDTCTL = WDTPW + WDTHOLD;   // Watchdog Timer'ı durdur.

  BCSCTL3|=LFXT1S_2;       // ACLK, VLO'dan gelecek
  IFG1 &= ~OFIFG;          // OSC hata bayrağı siliniyor
  TAR=0x0000;              // TAR değeri sıfırlanıyor
  TACTL=0x0116;               // ACLK seçili, MOD1, 1:1, Interrupt Enable,
Interrupt Flag temizleniyor
  TACCR0=6000;                  // TACCR0 değerine 12.000/1=12.000/2=6.000
yükleniyor
  TACCTL0=CCIE;            // CCIFG interrupt'ı açılıyor

  P1DIR=0xFF;              // P1 çıkış olarak ayarlnıyor

  _EINT();                 // Genel kesmeler açılıyor

  for(;;);
}
```

Bu kod da istediğimiz gibi gerçekte çalışmaktadır.

3.1.1.3) TimerA Continuous Modu

TimerA continuous modda çalışırken şekil 9'da görüldüğü gibi öncelikle 0xFFFF'e kadar sayar ve daha sonra sayıcı sıfırlanır. Sıfırlandığı anda ise **TAIFG** bayrağı set edilir.



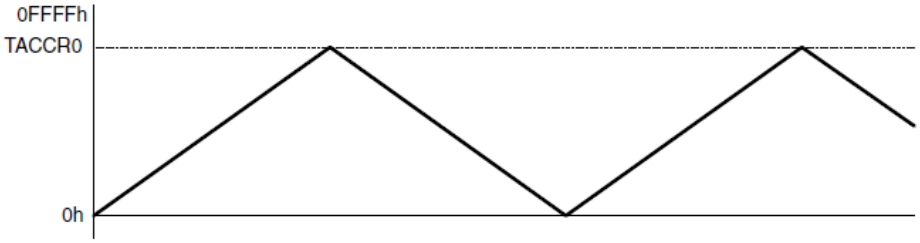
Şekil 9 - TimerA Cont. Mod

Bu konuda timer frekansı aşağıdaki formülle hesaplanır.

$$f_{timer} = \frac{f_{clk}}{(Divider\ Value) \times (65536)}$$

3.1.1.4) TimerA Up/Down Modu

TimerA up/down modda çalışırken sayıcı şekil-10'da görüldüğü gibi önce TACCRx'e kadar artarak sonra da azalarak sayar. Sayıcı her seferinde dolduğunda **CCIFG**ve sıfırlandığında ise **TAIFG** bayrağı set edilir.



Şekil 10 - TimerA Up/Down Modu

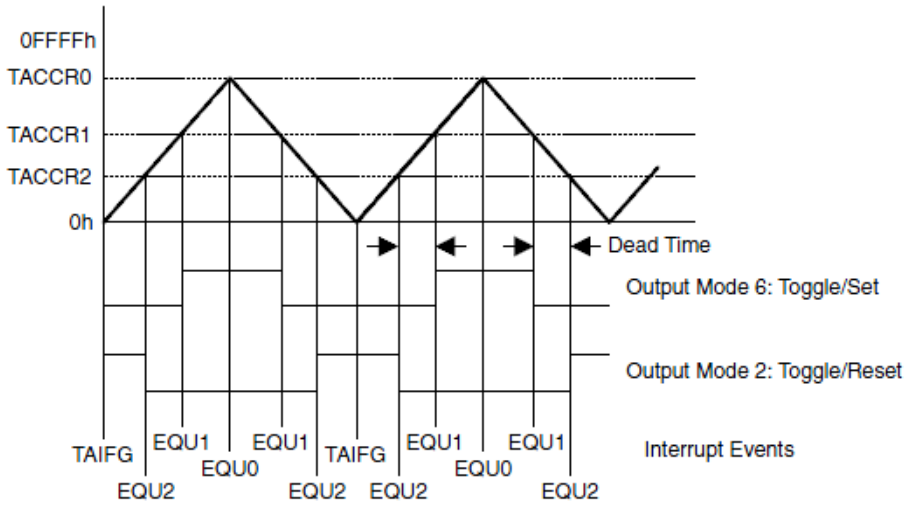
Bu konuda timer frekansı aşağıdaki formülle hesaplanır.

$$f_{timer} = \frac{f_{clk}}{2 \times (Divider\ Value) \times (TACCRx)}$$

3.1.2) TimerA Up/Down Modunu Kullanarak Basit PWM Sinyali Üretmek

TimerA'nın bir özelliği de basit şekilde PWM üretmektir. Bunun için sayıcıyı up/down moda almak ve TACCR0, TACCR1 ve TACCR2 registerlerini kontrol ederek basit PWM sinyali üretmek mümkündür.

Şekil-10'dan da görüleceği üzere öncelikle TimerA saat kaynağı seçimi ve TACCR0 ile frekans ayarlaması yapılır.



Şekil 11 - PWM Oluşturma Aşamaları ve Dead Time

Şekil-11'de görüleceği üzere sayıcı sıfırdan saymaya başlar ve TACCR2'ye kadar değeri 1 dir. TACCR2'ye ulaştıktan sonra sıfır olur. Bu sıfır olma kısmı TACCR1'e kadar sürer. Bu kısma dead time adı verilmektedir. Daha sonra TACCR1'den TACCR0 ve tekrar TACCR1'e kadar çıkış bir olduktan sonra periyodik şekilde bu sonsuza kadar devam eder. Bunlar oluşurken meydana gelen kesme olayları ise yine şekil-11'den görülebilir. Bu kesmelerden yola çıkılarak istenildiği gibi PWM sinyalleri üretilebilir.

Dead Time=(TACCR1-TACCR2) şeklinde hesaplanır.

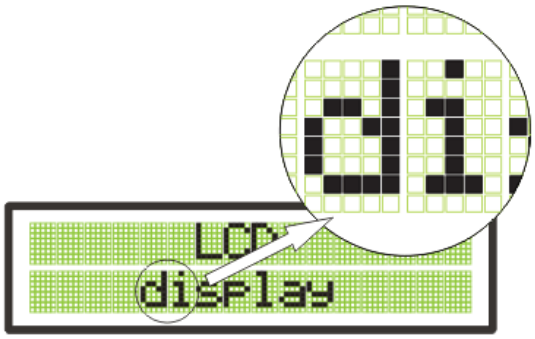
BÖLÜM 4 – MSP430 İLE LCD UYGULAMASI

4.1) LCD Ekran Hakkında Genel Bilgiler

Şekil-12'de bir örneği görülen karakter LCD'ler dışarıya bilgi aktarmak için kullanılan en yaygın birimlerdendir. Genel itibari ile Hitachi firmasının HD44780 entegresini ve türevlerini taşıyan karakter LCD'ler çeşitli metotlarla sürülürler. Biz bu bölümde şekil-12'de de görülebilecek 2x16 yani 2 satır ve her satırda 16 karakter yazabilen karakter LCD'leri inceleyeceğiz.

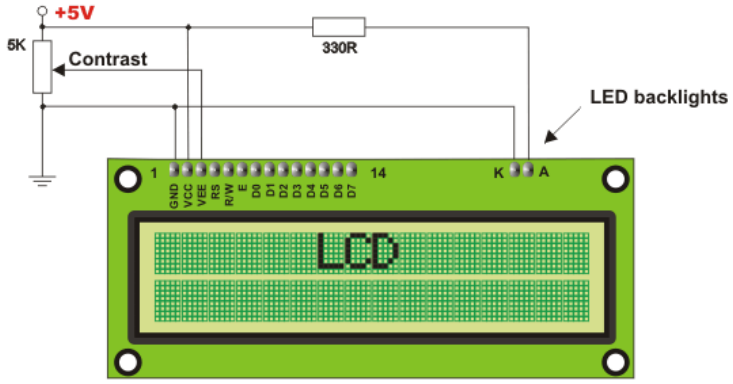
Şekil 12 - 2x16 Karekter LCD

Karakter LCD'lerin genelinde her harf şekil-13'te görüleceği gibi 5x7'lik birimler halinde şekillenirler. Altta boş kalan son birim ise imleç içindir.



Şekil 13 - 5x7 Karekter Oluşumu

LCD birimi genellikle normal entegre güç biriminden ayrı bir de arka aydınlatma ışığı gücü verilerek kullanılırlar. Bu birimin nasıl sürüleceği ise şekil-14'te gözükmektedir.



Şekil 14 - LCD Güç Bağlantısı

Karakter LCD'lerin oluşturabileceği her bir karakter ise karakter LCD'nin özel CGROM hafızasına kaydedilmişlerdir. ASCII karakter uyumu olan karakterlerin listesi tablo-4'de görülebilmektedir.

Tablo 4 - LCD Karakter Tablosu

Şekil-4'te de görüleceği üzere CGROM'un ilk 8 karakterlik (0x00..0x0F) kısmı boştur ve yazılabilirdir. Bu kullanıcıya tabloda olmayan karakterleri kendisi tanımlamasına olanak sağlar.

Karakter LCD'lerin genelinde 16 bacak bulunur. Bunların 14 tanesi LCD'yi kontrol etmek amaçlı kullanılırken, 15 ve 16. bacaklar genellikle LCD arka ışığı için kullanılırlar. LCD arka ışığı yazıların daha belirgin gözükmesi için gereklidir.

Bu bacakların görevini sırasıyla verecek olursak;

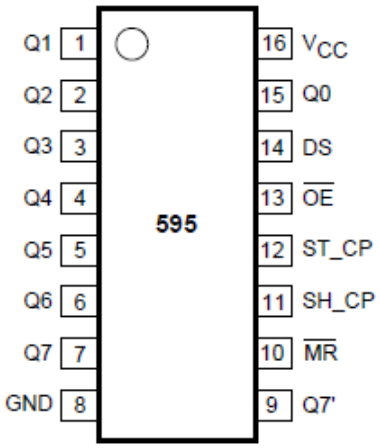
1 - GND : Toprak ucudur
2 - VCC : +5V verilecek uçtur
3 - VEE : Kontrast ucudur, bir pot vasıtasıyla +5V-0V aralığında sürülmelidir
4 - RS : Gelen bilginin komut mu data mı olduğu bu uçla belirlenir (0: Komut, 1: Data)
5 - RW : LCD'ye veri yazma ya da okuma yetkilendirme ucudur (0: Yazma, 1: Okuma)
6 - E : Enable ucudur, LCD'ye bilgi giriş çıkışını kontrol eden uçtur, düşen kenar tetiklemelidir
7..14 - Data : Data uçlarıdır, bilgi giriş çıkışları bu bacaklar sayesinde olur
15,16 - BL : Backlight anot, katot uçarıdır

Karakter LCD'lerin kullanılması, led, direnç sürümü gibi olmamaktadır. Karakter LCD kullanımında, enerjiyi ilk verdiğimiz anda karakter LCD'yi nasıl kullanmak istediğimizi LCD'ye belirli kurallar çerçevesinde iletmemiz gerekmektedir.

4.2) MSP430, 74HC595 ve LCD

MSP430'un bacak sayısının kısıtlı olmasından dolayı, diğer işlevleri yerine getirmek için LCD ekranı 74HC595 port çoğullacı ile kullanmak çoğu işlemde bizlere kolaylık sağlamaktadır.

Şekil-15'te görülebilecek 74HC595 entegresi 100Mhz'e kadar çalışabilen 8 bitlik bir shift registerdir. Bu özelliği sayesinde 74HC595 entegresi ayrıca DS ucundan girilen seri bilgileri istenildiği an Q0..Q7 uçlarından paralel bilgi olarak alınmasına olanak sağlar.



Şekil 15 - 74HC595 DIP Yapısı

Şekil-15'te görülen pinlerin görevleri ise şöyledir;

Q0..Q7- Paralel çıkışlar
DS - Seri data girişi
OE - Çıkış açık
SH_CP - Shift register saat kaynağı
ST_CP - Kaydedici registerin saat kaynağı
MR - Reset ucu
Q7' - Seri data çıkışı
GND - Toprak ucu
Vcc - +5V

74HC595'in çalışma mantığı ise şöyledir;
- OE ucu toprak hattına, MR ucu ise +5V'a bağlanır,
- DS ucundan seri bilgi girişi SH_CP'nin her yükselen kenarında shift edilerek kaydedici registere yazılır,
- 8 çıkışımız olduğu için bu işlem 8 kez gerçekleştirilir,
- Tüm bu işlemler gerçekleşirken Q0..Q7 çıkışlarında bir değişiklik olmaz,
- Son olarak kaydırma işlemi bittiğinde kaydedici registerdeki değerin Q0..Q7 uçlarına yansıması için ST_CP ucuna yükselen kenarda sinyal verilir,

- Son olarak Q0..Q7 uçlarında, ilk giren bilgi Q7 de son giren bilgi ise Q0'da olmak koşulu ile sıralama tamamlanır.
Örnek olarak 0xA5'nın uçlara iletimini örnek verelim;

0xA5=0b10101001 olur. İlk giren Q7'de olacağına göre öncelikle A5'in en yüksek biti olan 7. Biti göndermeliyiz. Daha sonra ise 6,5,4.. şeklinde bunu sürdürmeliyiz.
C'de bu işlemi aşağıdaki fonksiyon ile sağlayabiliriz.

```
void three_wire_control(unsigned char temp)
{
	char i;
	for(i=0;i<8;i++)
	{
		if(temp&0x80)
			LCD_PORT |=DataIO_Pin;
		else
			LCD_PORT&= ~DataIO_Pin;
		LCD_PORT|=Clock_Pin;
		LCD_PORT&=~Clock_Pin;

		temp*=2;;
	}
	LCD_PORT|=Enable_Pin;
	LCD_PORT&=~Enable_Pin;
}
```

İşlem takip edilirse öncelikle 7. bitin daha sonra ise diğer düşük bitlerin, en sonunda ise 0. bitin çıkışa gönderildiği görülebilir. Bu şekilde port genişletme işlemimizi de (3'ten 8'e) gerçekleştirmiştir.

Fakat bu işlemi LCD'de gerçekleştirmek için öncelikle LCD kütüphanesinde bazı değişiklikler yapmamız gerekmektedir. Yapacağımız modifiyenin örneğini tek fonksiyonda gösterirsek, kodlarımız aşağıdaki gibi olacaktır.

~LCD dosyasının orjinalini Hi-Tech ile Pic Programlama kitabında bulabilirsiniz~

```
void lcd_komut(unsigned char c)
{
	unsigned char temp;
	temp = 0x00;
	temp |= 0x04;
	three_wire_control(temp);
	temp |= ( c & 0xF0 );
	three_wire_control(temp);
	temp &= 0xF0;
	three_wire_control(temp);
	lcd_busy();
	temp = 0x00;
	temp |= 0x04;
	three_wire_control(temp);
	temp |= ( (c & 0x0F)<<4 );
	three_wire_control(temp);
	temp &= 0xF0;
	three_wire_control(temp);
	lcd_busy();
}
```

Kodları dikkatle incelerseniz aslında yapılan işin tüm giriş çıkışları 8 bit haline getirmek olduğu rahatlıkla görülebilir.

Tüm kodları bu şekilde modifiye edip, son olarak 3-Wire fonksiyonumuzu da ekleyip kütüphane haline getirirsek **lcd.h** dosyamız aşağıdaki gibi olacaktır.

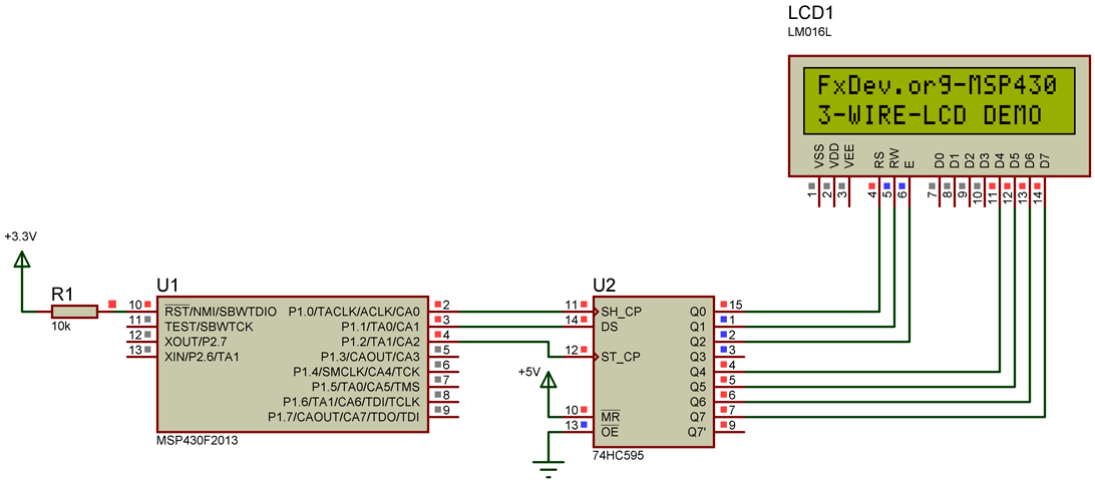
```
/*                          www.FxDev.org
 *               3 Kablolu 2x16 LCD Kullanım Klavuzu
 * 74LS595 ile birlikte kullanılmalıdır.
 * Cursor kapalıdır.
 *
 * lcd_init();          ile LCD'nin ilk ayarlarını yap
 * lcd_clear();         ile LCD'yi sil
 * lcd_yaz("deneme");   şeklinde yazı yazdır.
 * veri_yolla('F');     şeklinde tek ascii kodu yazdır.
 * lcd_gotoxy(1,13);    şeklinde LCD'nin istenilen yerine git.
 *                          www.FxDev.org
*/

#define LCD_PORT        P1OUT // LCD'nin bağlandığı port
#define Clock_Pin       BIT0  // 74LS595 Clk girişi, yükselen kenar
#define DataIO_Pin      BIT1  // 74LS595 Data girişi
#define Enable_Pin      BIT2  // 74LS595 Enable girişi

/* LCD'de kullanilan komutlarin tanimlamasi*/
#define Sil           1       // Ekrani temizler
#define BasaDon      2        // Imleci sol üst köseye getirir
#define SolaYaz      4        // Imlecin belirttigi adres azalarak gider
#define SagaYaz      6        // Imlecin belirttigi adres artarak gider
#define ImlecGizle    12      // Göstergeyi ac, kursor görünmesin
#define ImlecAltta    14      // Yanip sönen blok kursor
#define ImlecYanSon   15      // Yanip sönen blok kursor
#define ImlecGeri  16         // Kursorü bir karakter geri kaydir
#define KaydirSaga    24      // Göstergeyi bir karakter saga kaydir
#define KaydirSola    28      // Göstergeyi bir karakter sola kaydir
#define EkraniKapat    8      // Göstergeyi kapat (veriler silinmez)
#define BirinciSatir 128      // LCD'nin ilk satir baslangic adresi
                              // (DDRAM adres)
#define IkinciSatir  192      // Ikinci satirin baslangic adresi
#define KarakUretAdres 64     // Karakter üreteci adresini belirle
                              // (CGRAM adres)
/* LCD'de Kullanilan Fonksiyon Seçimi */
#define CiftSatir8Bit  56      // 8 bit ara birim, 2 satir, 5*7 piksel
#define TekSatir8Bit   48      // 8 bit ara birim, 1 satir, 5*7 piksel
#define CiftSatir4Bit  40      // 4 bit ara birim, 2 satir, 5*7 piksel
#define TekSatir4Bit   32      // 4 bit ara birim, 1 satir, 5*7 piksel

extern void three_wire_control(unsigned char temp);
extern void veri_yolla(unsigned char);
extern void lcd_clear(void);
extern void lcd_yaz(const char * s);
extern void lcd_gotoxy(unsigned char x, unsigned char y);
extern void lcd_init(void);
extern void lcd_enable(void);
extern void lcd_komut(unsigned char c);
```

Tüm bu fonksiyonları yerine getiren **lcd.c** dosyamız ise aşağıdaki gibi olacaktır.

```
#include    <msp430.h>
#include    "lcd.h"
#include    "delay.h"

void three_wire_control(unsigned char temp)
{
      char i;
      for(i=0;i<8;i++)
      {
            if(temp&0x80)
                  LCD_PORT |=DataIO_Pin;
            else
                  LCD_PORT&= ~DataIO_Pin;
            LCD_PORT|=Clock_Pin;
            LCD_PORT&=~Clock_Pin;

            temp*=2;;
      }
      LCD_PORT|=Enable_Pin;
      LCD_PORT&=~Enable_Pin;
}

void lcd_busy(void)
{
      DelayUs(100);
}

void lcd_komut(unsigned char c)
{
      unsigned char temp;
      temp = 0x00;
      temp |= 0x04;
      three_wire_control(temp);
      temp |= ( c & 0xF0 );
      three_wire_control(temp);
      temp &= 0xF0;
      three_wire_control(temp);
      lcd_busy();
      temp = 0x00;
      temp |= 0x04;
      three_wire_control(temp);
      temp |= ( (c & 0x0F)<<4 );
      three_wire_control(temp);
      temp &= 0xF0;
      three_wire_control(temp);
      lcd_busy();
}

void veri_yolla(unsigned char c)
{
      unsigned char temp=0;
      temp |= 0x05;
      three_wire_control(temp);
      temp |= ( c & 0xF0 );
      three_wire_control(temp);
      temp &= 0xF1;
      three_wire_control(temp);
      lcd_busy();
      temp = 0x00;
      temp |= 0x05;
      three_wire_control(temp);
```

```
	temp |= ( (c & 0x0F)<<4 );
	three_wire_control(temp);
	temp &= 0xF1;
	three_wire_control(temp);
	lcd_busy();
}

void lcd_clear(void)
{
	lcd_komut(0x1);
	DelayMs(2);
}

void lcd_yaz(const char *s)
{
	lcd_busy();
	while(*s)
		veri_yolla(*s++);
}

void lcd_gotoxy(unsigned char x,unsigned char y)
{
	if(x==1)
		lcd_komut(0x80+((y-1)%20));
	else
		lcd_komut(0xC0+((y-1)%20));
}

void lcd_enable(void)
{

three_wire_control(0x04);_NOP();_NOP();_NOP();_NOP();three_wire_control(0x
00);
}

void lcd_init()
{
	three_wire_control(0);

    DelayMs(100);
    lcd_komut(0x03);
    DelayMs(10);
    lcd_enable();
    DelayMs(10);
    lcd_enable();
    lcd_komut(0x02);
    DelayMs(20);

    lcd_komut(CiftSatir4Bit);
    lcd_komut(SagaYaz);
    lcd_komut(ImlecGizle);
    lcd_clear();

    lcd_komut(BirinciSatir);
}
```

Kütüphanemizi tanımladıktan sonra, istediğimiz herhangi bir yazıyı LCD'mize yazdıralım.

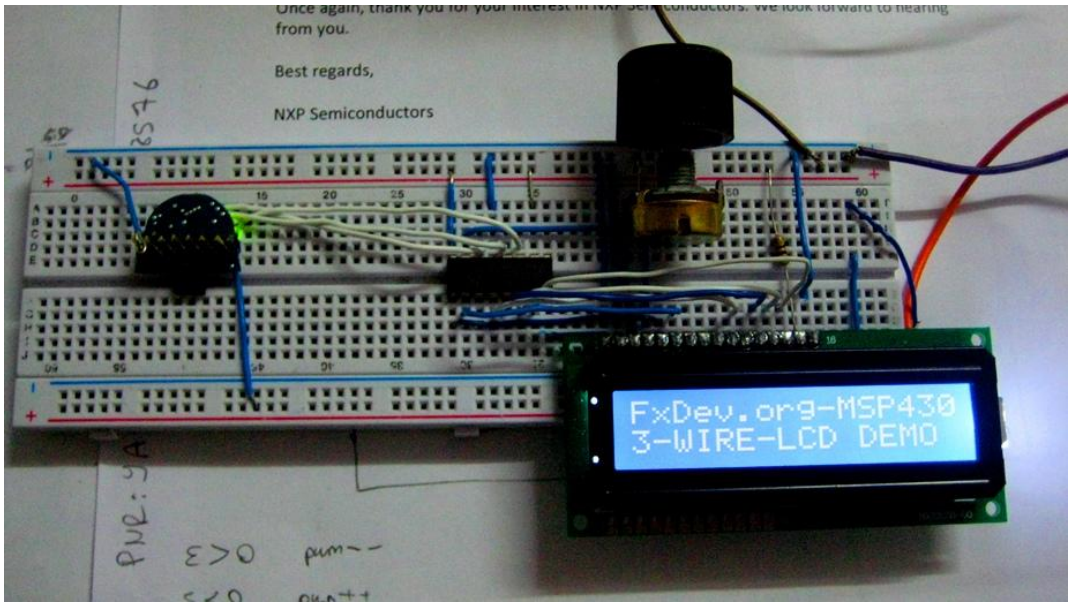


Şekil 16 - 74HC595 ile 2x16 LCD Uygulaması

Şekil-16'daki devreyi çalıştıran kodumuz ise aşağıdadır.

```
//                  MSP430F2013
//               .    -----------------
//            /|\  |                   |
//             |   |                   |
//              --|RST                 |
//                |              P1.0|-->Clock
//                |              P1.1|-->Data
//                |              P1.2|-->Enable
//

#include  <msp430.h>
#include "delay.h"
#include "lcd.h"

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;  // Watchdog timer kapatılıyor

      // Çalışma frekansı 16Mhz
      BCSCTL1 = CALBC1_16MHZ;
      DCOCTL = CALDCO_16MHZ;
      BCSCTL3= LFXT1S_2;

    P1DIR = 0x07;       // P1.0, P1.1, P1.2 çıkış olarak tanımlanıyor
    P1SEL = 0x00;       // P1 GPIO oldu

    DelayMs(250);       // LCD'nin işlemcisinin stabil olması
                        // için 250ms bekleniyor
      lcd_init();

      lcd_gotoxy(1,1);
      lcd_yaz("FxDev.org-MSP430");
      lcd_gotoxy(2,1);
      lcd_yaz("3-WIRE-LCD DEMO");

      for(;;);
}
```

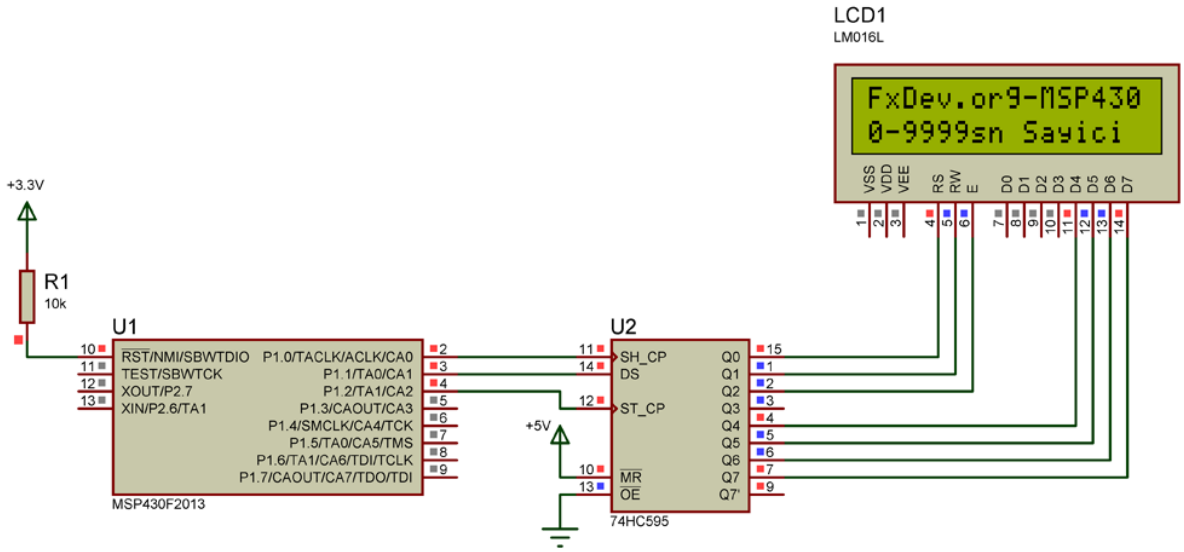
Devrenin gerçeklenmiş halini (LCD ve 74HC595'i) ise resim 3'te görebilirsiniz.



Resim-3 – 74HC595 ile 2x16 LCD Uygulamasının Gerçeklenmesi

4.3) MSP430, LCD ile Sayıcı Uygulaması

LCD kullanımının ardından bunu bir örneğe dökelim. Şu ana kadar öğrendiğimiz aşamalardan **0-9999 saniye sayıcı uygulamasını** rahatlıkla geliştirebiliriz. Bunun için öncelikle şekil 17'deki devreyi kuruyoruz.



Şekil 17 - 0/9999 Saniye Sayici Uygulaması

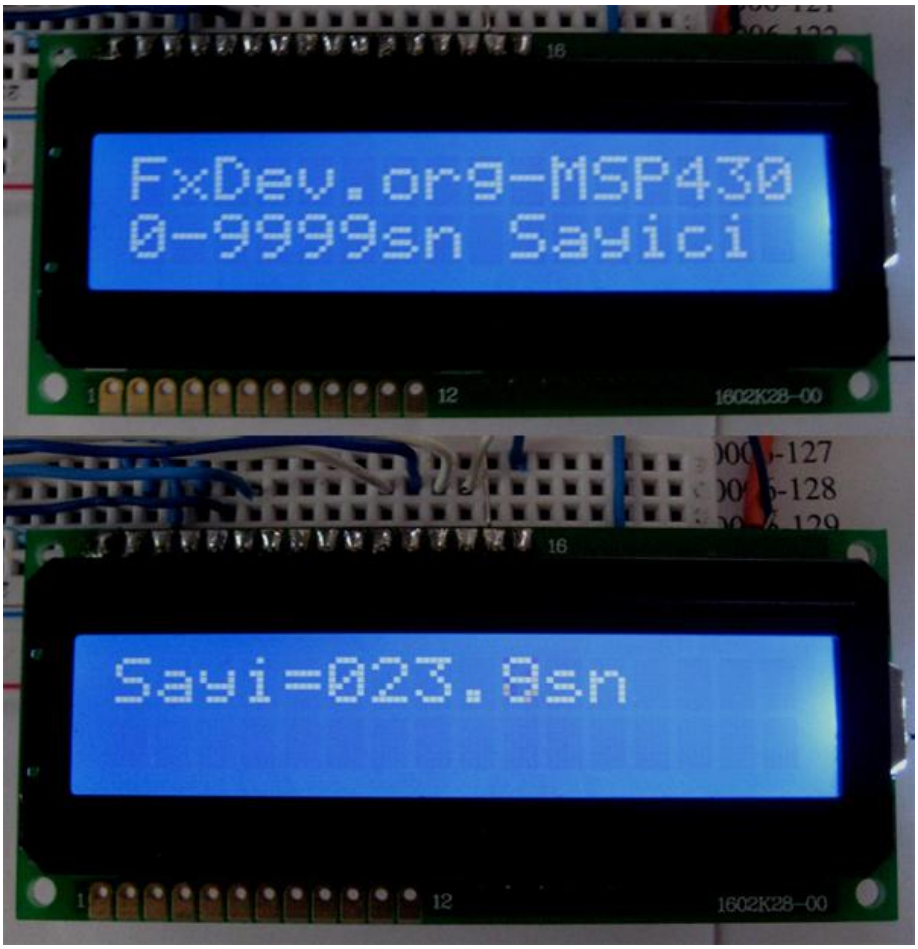
Şekil 17'deki devreyi kurduktan sonra bölüm 3.1.1'deki Timer A modunu kullanarak bir saniyede bir zamanlayıcı kesmesi veren programımızı yazalım. Bunun için saat kaynağımızın frekansını 8Mhz kabul ederek 3.1.1'de verilen formül 25ms'de bir kesme oluşacak şekilde işletildiğinde TACCRRx'e yüklenecek değer 25000 olarak bulunur.

Söylediklerimizin ışığında isteğimizi yerine getiren kod öbeğini aşağıda bulabilirsiniz.

```
//                 MSP430F2013
//          .   -----------------
//        /|\ |                   |
//         |  |                   |
//          --|RST                |
//            |              P1.0|-->Clock
//            |              P1.1|-->Data
//            |              P1.2|-->Enable
//            |                   |
//

#include  <msp430.h>
#include "delay.h"
#include "lcd.h"

char  flag=0,
            i     =0;
int   sayi=0;

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;  // Watchdog timer kapatılıyor

      // Çalışma frekansı 8Mhz
      BCSCTL1 = CALBC1_8MHZ;
      DCOCTL = CALDCO_8MHZ;
      BCSCTL3= LFXT1S_2;

    P1DIR = 0x07;       // P1.0, P1.1, P1.2 çıkış olarak tanımlanıyor
    P1SEL = 0x00;       // P1 GPIO oldu

    DelayMs(500);       // LCD'nin işlemcisinin stabil olması
                        // için 250ms bekleniyor
      lcd_init();

      lcd_gotoxy(1,1);
      lcd_yaz("FxDev.org-MSP430");
      lcd_gotoxy(2,1);
      lcd_yaz("0-9999sn Sayici");
      DelayMs(2000);

      TAR=0x0000;       // TAR değeri sıfırlanıyor
      TACTL = TASSEL_2 + MC_3 + TAIE;
// SMCLK seçili, MOD1, 1:8, Interrupt Enable, Interrupt Flag temizleniyor
      TACCR0=25000;
// TACCR0 değerine 8.000.000/8=1000000/40=25000 yükleniyor

      _EINT();

      lcd_clear();
      lcd_gotoxy(1,1);
      lcd_yaz("Sayi=");

      for(;;)
      {
            lcd_gotoxy(1,6);
            veri_yolla((sayi/1000)+48);
            veri_yolla((sayi%1000)/100+48);
            veri_yolla((sayi%100)/10+48);
            veri_yolla('.');
            veri_yolla((sayi%10)+48);
```

```
            lcd_yaz("sn");
      }
}

// TimerA kesme vektörü
#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A (void)
{
      i++;
      if(i==40)
      {
            if(sayi<10000) sayi++; else sayi=0;
            i=0;
      }

      TACTL&=0xFFFE; // Kesme bayrağı temizleniyor
}
```

Uygulamanın çalışır hailini resim 4'de görebilirsiniz.

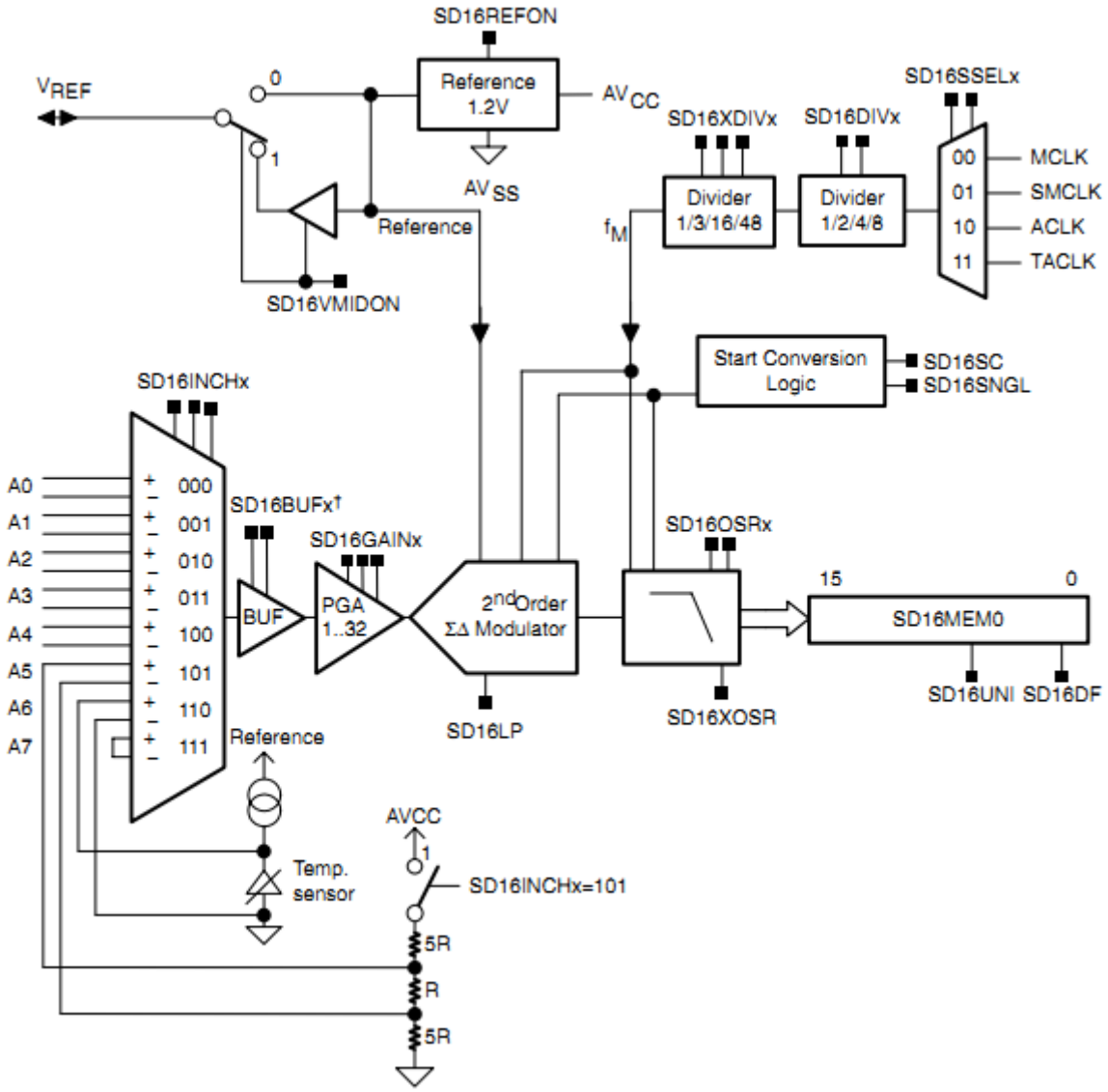


Resim-4 – 0/9999 Saniye Sayıcı Uygulaması

BÖLÜM 5 – MSP430 ADC İŞLEMLERİ

Bu kitabı hazırlarken bölüm 1'de bahsettiğimiz MSP430F2013'ün normal ADC bloğu yerine blok diyagramını şekil 18'de göreceğiniz 16 bitlik sigma delta modülasyonlu ADC mimarisine sahiptir. Bu birim aşağıdaki özelliklere sahiptir;

- ✓ Yazılımsal olarak seçilebilen referans gerilimi (1.2V)
- ✓ Yazılımsal olarak seçilebilen dış ya da iç referans kaynağı
- ✓ İçsel sıcaklık sensörü barındırma
- ✓ 1.1MHz hızına çıkabilme
- ✓ Yüksek empedans giriş bufferı

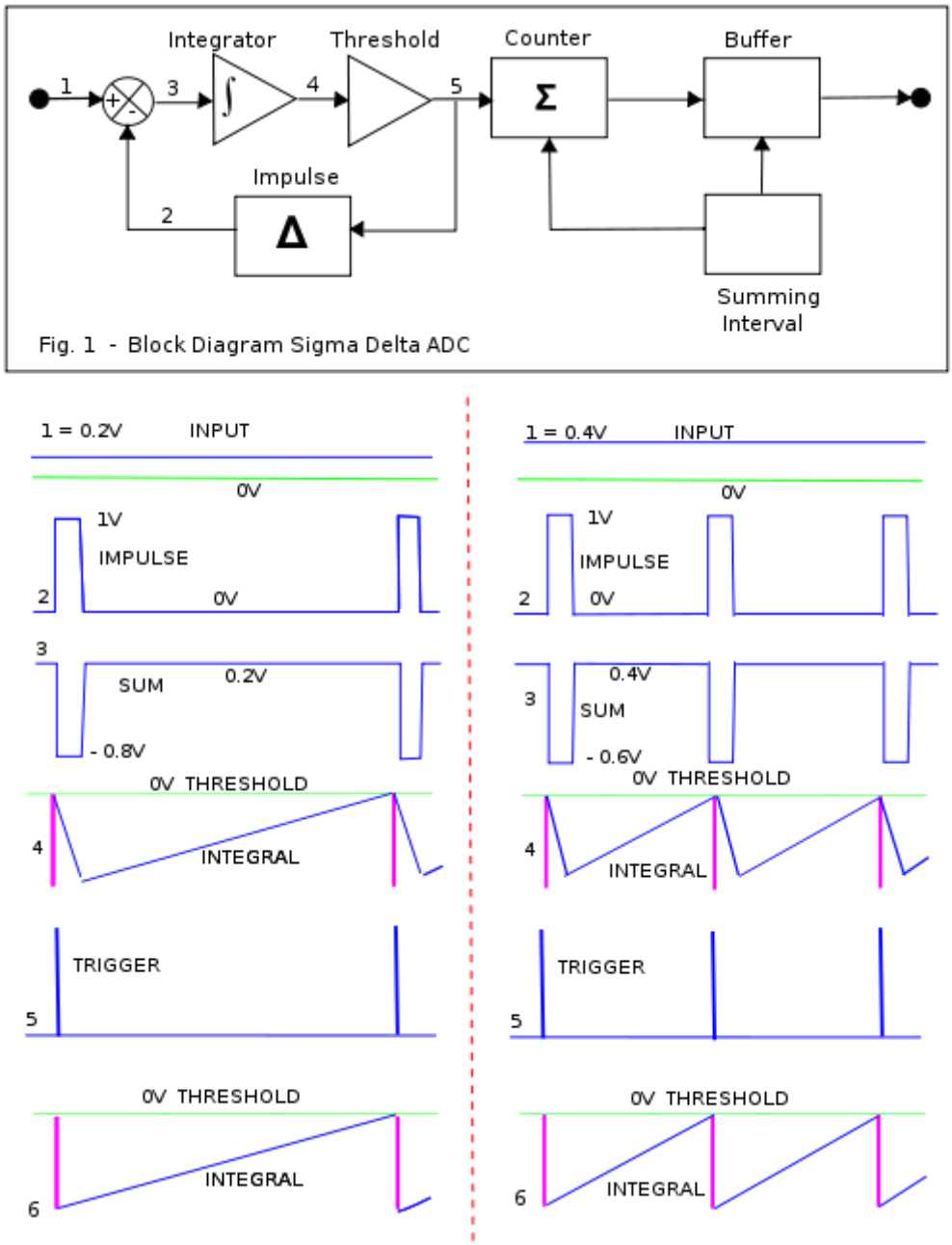


Şekil 18 - SD16_A Birimi

5.1) ΣΔ Modülasyon Nedir?

Sigma delta modülasyonu yüksek çözünürlüklü kodlama yapmak için kullanılan bir metottur. Analog ve dijital sinyal işlemede sıklıkça kullanılan bu yöntem ADC, DAC, frekans üreticiler, anahtarlamalı güç kaynakları ve motor denetleyiciler gibi bir çok uygulamada aktif olarak kullanılırlar. Özellikle MP3 gibi güncel uygulamaların kodlamasında da sigma delta modülasyon kullanılmaktadır.

Sigma delta modülasyonun çalışması şekil 19'da görülebilir.



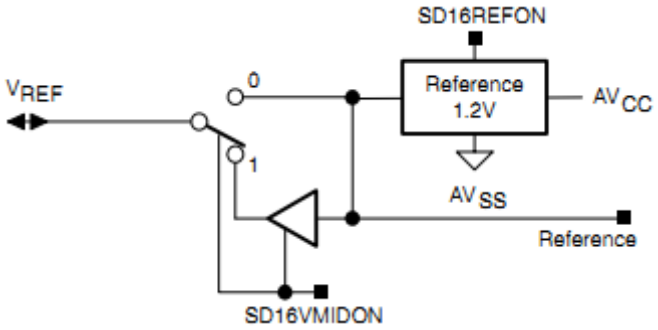
Şekil 19 - Sigma Delta Modülasyon

Şekil 19'dan da görüleceği üzere sigma delta modülasyon hassas çevrim yapabilmek için giriş gerilimine göre tatikleme sinyali üretir. Bu bir nevi PWM kullanarak DAC yapmaya benzer. Gerilim seviyesi arttıkça tetikleme sinyali sayısı artar , bu tetiklemelerin sayısı sayılarak gerilim değeri öğrenilebilir.

5.2) ADC İşlemleri

5.2.1) Referans Belirleme

MSP430 ile ADC işlemine başlamadan önce her ADC işleminde olduğu gibi güvenilir bir referans gerilimine ihtiyacımız vardır. Şekil 20'de görünen kısımda MSP430'un referans seçimi yapılabilmektedir.

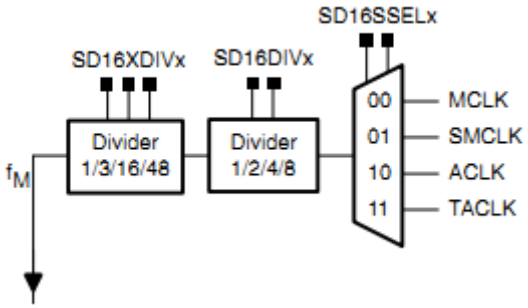


Şekil 20 - ADC Referans Bloğu

Şekil-20'den de görüleceği üzere **SD16CTL** registerinde bulunan **SD16REFON** biti ile içsel 1.2V referans kaynağı aktif edilir. Ayrıca bu gerilim **SDVMIDON** biti ile dışarıya alınabilir. Bu şekilde kullanımda bu kaynaktan en fazla 1mA almak mümkündür. 1.2V bu şekilde entegre dışına alındığında besleme uçları arasına 470nF koyulması önerilmiştir.

5.2.2) Çevrim Frekansı Belirleme

ADC biriminin bir diğer önemli ayarı da çevrim süresinin belirlenmesidir. Şekil-21'de MSP430'un ADC frekansı belirleme birimi görülmektedir.

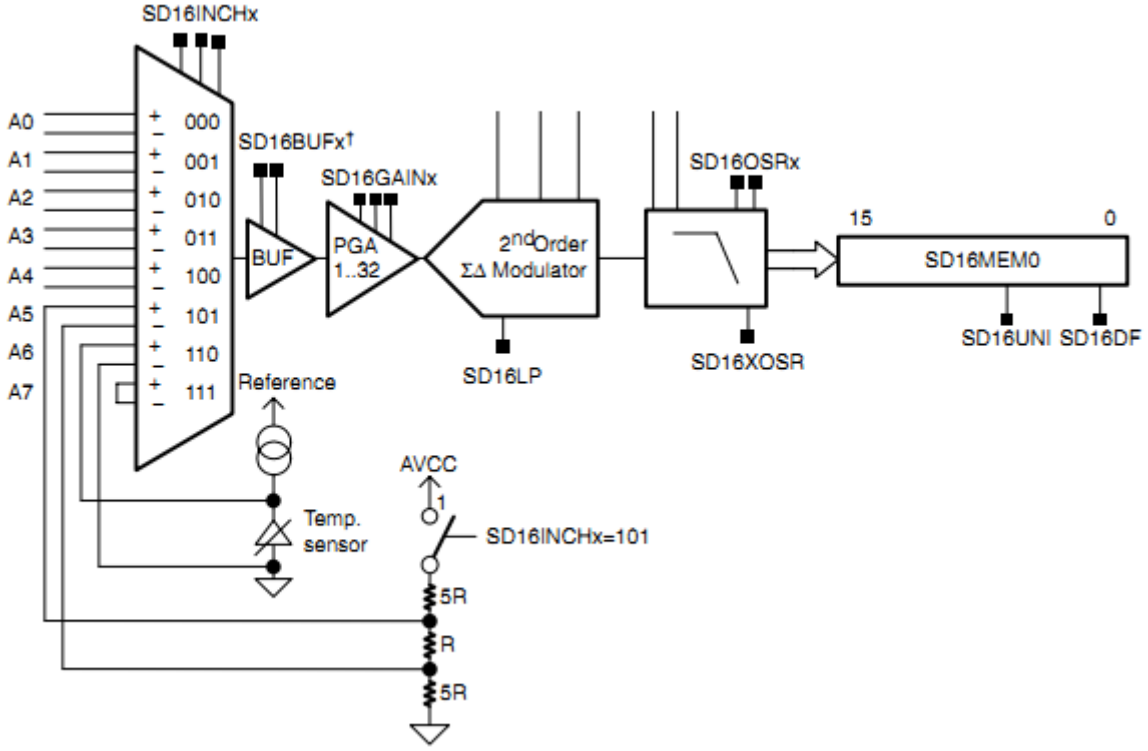


Şekil 21 - SD16_A Frekans Bloğu

Bu blok şekil 21'den de görüleceği üzere her türlü saat kaynağı ile beslenebilmektedir. Ayrıca iki ayrı frekans bölücü birimi de içerisinde barındırmaktadır. ADC çevrim hızı 1.1MHz'e kadar çıkabilmektedir.

5.2.3) Kanal Seçimi ve Filtre Blokları

MSP430 SD16_A biriminin son kısmı şekil 22'de görülen kısımdır. Bu bölüm kanal seçme birimi, harici buffer, kazanç birimi, 2. derece sigma delta modülasyonu ve filtre birimini barındırır.
İlk 5 kanal normal ADC girişleri iken, A5 içsel dirençlerle bölünmüş değeri ve A6 ise içsel sıcaklık sensörünü okumaktadır. A7 ise kendi içerisinde kısa devre edilmiştir.



Şekil 22 - ADC Giriş ve Filtre Birimleri

5.2.4) ADC Kullanım Registerleri

5.2.4.1) SD16CTL

MSP430'da ADC kullanılmadan ilk önce **SD16CTL** registeri ayarlanmalıdır. Bu registerin özellikleri aşağıda gösterilmiştir.

SD16CTL

15	14	13	12	11	10	9	8
Reserved				SD16XDIVx			SD16LP
r0	r0	r0	r0	rw–0	rw–0	rw–0	rw–0

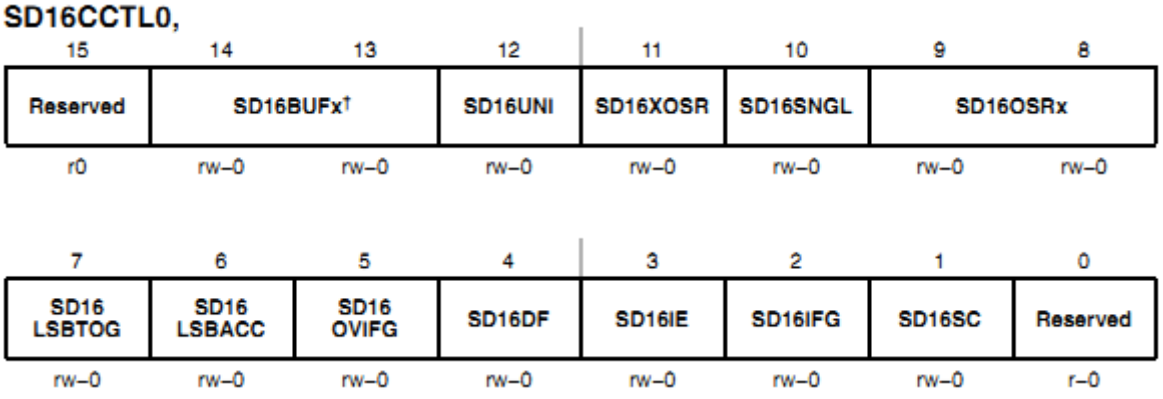
7	6	5	4	3	2	1	0
SD16DIVx		SD16SSELx		SD16 VMIDON	SD16 REFON	SD16OVIE	Reserved
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	r0

Bit 15-12	-	Ayrılmış alan
Bit 11-9	**SD16XDIVx**	000 - / 1 001 - / 3 010 - / 16 011 - / 48 1xx - / Ayrılmış alan
Bit 8	**SD16LP**	0: Low power mod deaktif 1: Low power mod aktif
Bit 7-6	**SD16DIVx**	00 - / 1 01 - / 2 10 - / 4 11 - / 8
Bit 5-4	**SD16SSELx**	00 - / MCLK 01 - / SMCLK 10 - / ACLK 11 - / External TACLK
Bit 3	**SD16VMIDON**	0: 1.2V çıkışa yansımayacak 1: 1.2V çıkışa yansıyacak
Bit 2	**SD16REFON**	0: Referans kapalı 1: Referans açık
Bit 1	**SD16OVIE**	0: Taşma kesmesi aktif değil 1: Taşma kesmesi aktif Bu bitten önce GIE aktif yapılmalı.
Bit 0	-	Ayrılmış alan

5.2.4.2) SD16CCTL0

Bu register ile ilgili birimin tüm filtre işlemleri ile çevrim modları ve sonuçların nasıl elde edileceği bilgileri ele alınmaktadır. Ayrıca kesme bayrakları ve kesme izin bitleri ile giriş bufferının kullanılıp kullanılmayacağı da yine bu registerle ayarlanabilmektedir.

SD16CCTL0,

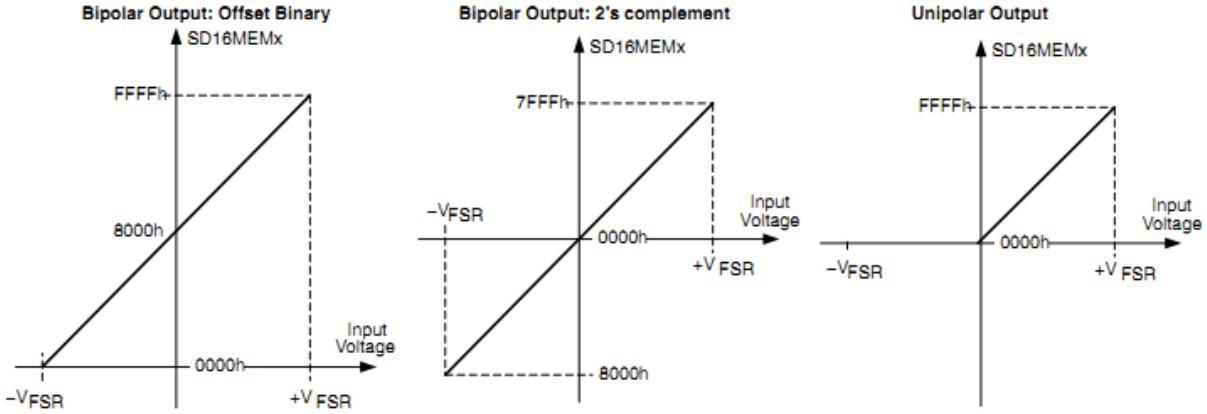


Bit 15	-	Ayrılmış alan
Bit 14-13	**SD16BUFx**	00 - / Buffer kullanılmayacak 01 - / Yavaş hız 10 - / Normal hız 11 - / Yüksek hız
Bit 12	**SD16UNI**	0: Bipolar mod 1: Unipolar mod
Bit 11	**SD16XOSR**	Geliştirilmiş aşırı örnekleme biti
Bit 10	**SD16SNGL**	0: Devamlı çevrim modu 1: Tek çevrim modu
Bit 9-8	**SD16OSRx**	**SD16XOSR=1 ise;** 00: 256 01: 128 10: 64 11: 32 **SD16XOSR=0 ise;** 00: 512 01: 1024 10: - 11: -
Bit 7	**LSBTOG**	LSB toggle biti, bu bit aktifse SD16LSBACC, SD16MEM0 okundukça sürekli toggle yapar. 0: Toggle kapalı 1: Toggle açık
Bit 6	**LSBACC**	Bu bit çevrimin düşük ya da yüksek bitli değerleri okuma izni verir. 0: SD16MEMx yüksek bite erişim izni verildi 1: SD16MEMx düşük bite erişim izni verildi
Bit 5	**SD16OVIFG**	Taşma kesmesi bayrağı 0: Taşma yok 1: Taşma var
Bit 4	**SD16DF**	SD16_A Data formatı 0: Offset binary 1: 2’s complement
Bit 3	**SD16IE**	SD16_A kesme izin biti 0: İzin yok 1: İzin var
Bit 2	**SD16IFG**	SD16_A kesme bayrağı 0: Kesme yok 1: Kesme oluştu
Bit 1	**SD16SC**	SD16_A çevrimini başlat 0: Çevrim başlatma 1: Çevrim başlat
Bit 0	-	Ayrılmış bit

Çok kısa register bitlerinin görevlerinden bahsedelim; öncelikle bit 14 ve 13'e bakıldığında girişteki bufferı ayarladığını görebilirsiniz. Hata oranını düşürmek için kullanılan yüksek empedans girişleri aşağıdaki şekilde tanımlanmıştır. Ölçüm frekansınıza göre bu empedansların seçimi, ölçüm hatalarını engelleyecektir.

SD16BUFx	Buffer	SD16 Modulator Frequency f_M
00	Buffer disabled	
01	Low speed/current	f_M < 200kHz
10	Medium speed/current	200kHz < f_M < 700kHz
11	High speed/current	700kHz < f_M < 1.1MHz

Bit 12 ve bit 4'te bahsi geçen bit ise okunan değerin işaretli olup olmamasını belirler. Bunun ayrıntısını şekil 23'te görebilirsiniz.



Şekil 23 - İşaret Seçme

Bit 10 biti ile de çevrimi ister sürekli isterseniz de tek seferde yapabilmeniz mümkündür.

5.2.4.3) SD16INCTL0

Bu register kesme zamanını, giriş kazancını ve kanal seçimi ile ilgilidir.

SD16INCTL0

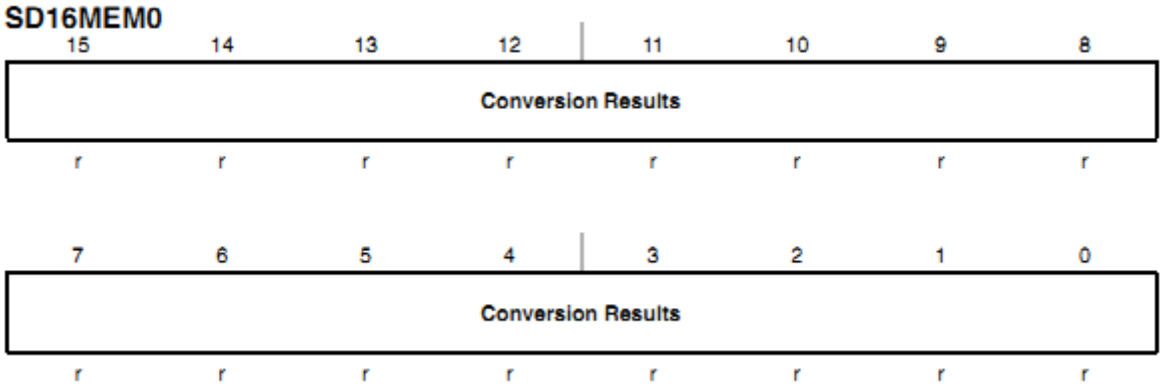
7	6	5	4	3	2	1	0
SD16INTDLYx		SD16GAINx			SD16INCHx		
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

Bit 7-6	**SD16INTDLYx**	Kesmenin ne zaman olacağını belirten bitler 00 - 4 örnek kesmeye neden olur 01 - 3 örnek kesmeye neden olur 10 - 2 örnek kesmeye neden olur 11 - 1 örnek kesmeye neden olur
Bit 5-3	**SD16GAINx**	Giriş ön kazanççısı 000 – x1 001 – x2

		010 – x4 011 – x8 100 – x16 101 – x32 110 – ayrılmış 111 – ayrılmış
Bit 2-0	**SD16INCHx**	Giriş kanalı seçme bitleri 000 – A0 001 – A1 010 – A2 011 – A3 100 – A4 101 – A5− (AVCC − AVSS) / 11 110 – A6 – Sıcaklık sensörü 111 – A7 – PGA ofset ölçümü için kısa devre

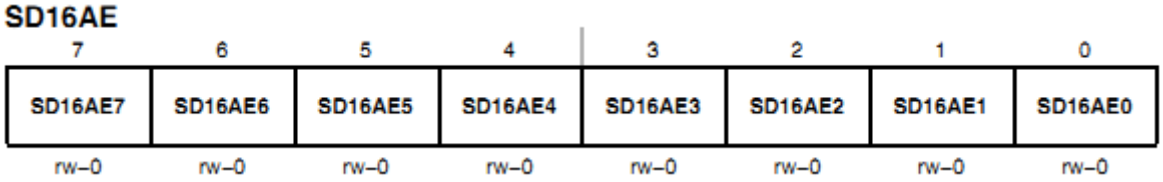
5.2.4.4) SD16MEM0

ADC çevriminin saklandığı 16bit registerdir.



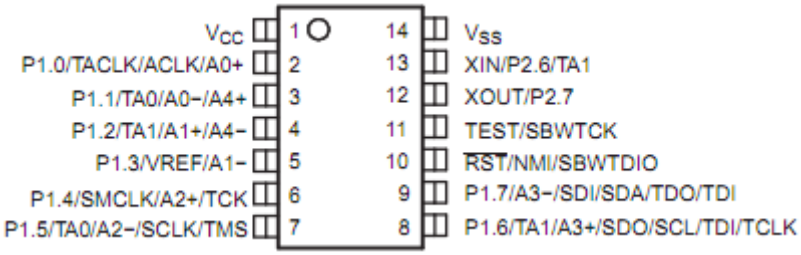
5.2.4.5) SD16AE

Bu register giriş birimlerinin toprak hatlarının nereye bağlanacağını belirtir. Normal durumda toprak hatlarının tümü MSP430'un VSS'ine bağlıdır. Birimlerin toprakları bitler 1 olduğunda dış kaynağa, sıfır olduğunda ise VSS'ye bağlı olurlar.



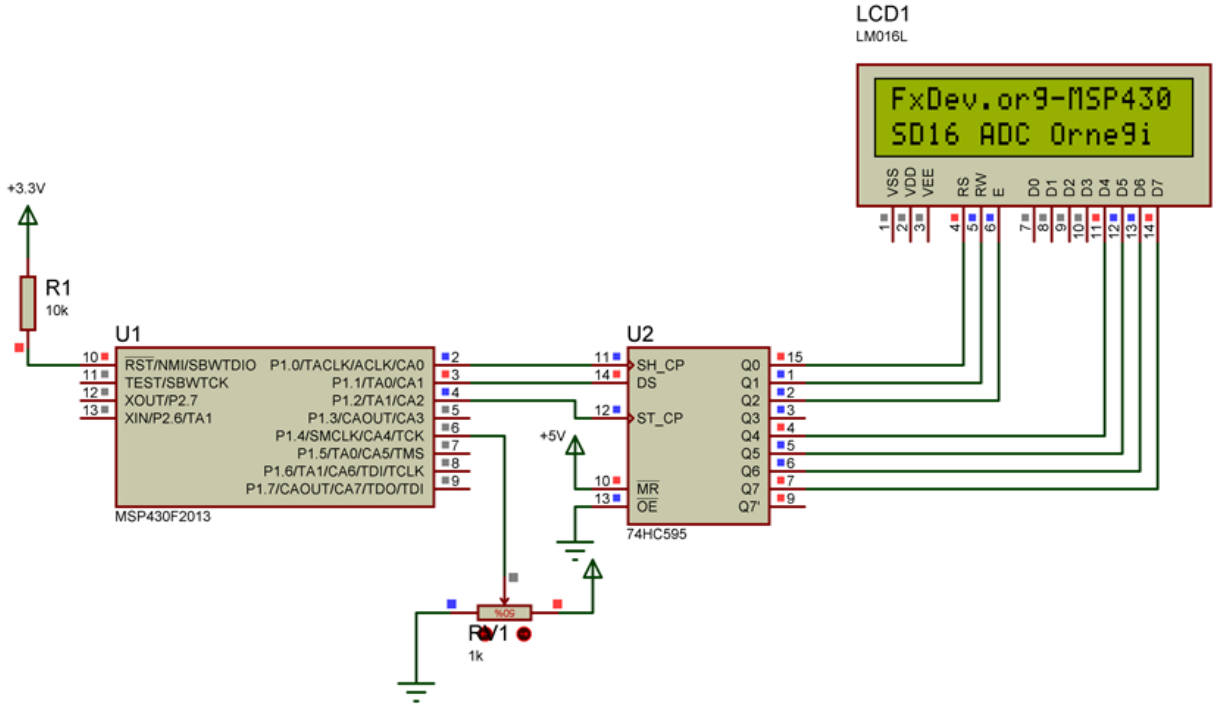
5.3) ADC Uygulaması

Yukarıdaki teorik kısımlardan sonra ilk uygulamamızı gerçekleştirelim. Bu uygulamamızda şekil 24'te görülen P1.4/A2 kanalına bir pot bağlayarak 0-65535 arasındaki register değerini LCD'ye yazdıralım.



Şekil 24 - MSP430 Pinout Şeması

Öncelikle şekil 25'teki devreyi kuralım.



Şekil 25 - ADC Ölçüm Devresi

Daha sonra aşağıdaki kod ile isteğimizi yerine getirelim. Burada sürekli çevrim ile registere sürekli çevrimi yazdıralım ve SD16 kesmesiyle çevrim bittiğinde okunan değeri 'sayi' değişkenine aktaralım.

```
//                 MSP430F2013
//             .    -----------------
//          /|\  |                   |
//           |   |                   |
//            --|RST                |
//              |              P1.0|-->Clock
//     Vin+ -->|A2+ P1.4      P1.1|-->Data
//              |A2- = VSS     P1.2|-->Enable
//              |                   |
//

#include  <msp430.h>
```

```
#include "delay.h"
#include "lcd.h"

char  flag=0,
      i     =0;
unsigned int      sayi=0;

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;  // Watchdog timer kapatılıyor

      // Çalışma frekansı 8MHz
      BCSCTL1 = CALBC1_8MHZ;
      DCOCTL = CALDCO_8MHZ;
      BCSCTL3= LFXT1S_2;

    P1DIR = 0x07;                     // P1.0, P1.1, P1.2 çıkış olarak
tanımlanıyor
                                                // P1.0, P1.1, P1.2 GPIO,
P1.4 A2 olarak işlev yapacak

    // Giriş frekansı 48'e bölünüyor, örnekleme frekansımız 166kHz
    // 1.2V referans gerilimi aktif
    // SMCLK frekans kaynağı olarak kullanılacak, 8MHz
    SD16CTL= SD16XDIV1 | SD16XDIV0 | SD16REFON | SD16SSEL_1;
    // A2+ seçildi, kazanç 1, 4 örnek sonra kesmeye girilecek
    SD16INCTL0=SD16INCH_2;
    // Kesme aktif
    // Çevrim unipolar, yani 0V=0, 3.3V=0xFFFF olacak
    // Çevrim memory'nin son 16 bitine atılacak
    // Çevrim unipolar
    // 256OSR
    SD16CCTL0 =  SD16UNI | SD16IE;
    // Tüm A-'ler VSS'ye bağlanacak
    SD16AE = 0;

    DelayMs(250);             // LCD'nin işlemcisinin stabil olması
                              // için 250ms bekleniyor
      lcd_init();

      lcd_gotoxy(1,1);
      lcd_yaz("FxDev.org-MSP430");
      lcd_gotoxy(2,1);
      lcd_yaz("SD16 ADC Ornegi");
      DelayMs(2000);

      // Çevrime başlanıyor
    SD16CCTL0 |= SD16SC;

      _EINT();                  // Enter LPM0 w/ interrupt

      lcd_clear();
      lcd_gotoxy(1,1);
      lcd_yaz("Sayi=");

      for(;;)
      {
            lcd_gotoxy(1,6);
            veri_yolla((sayi/10000)+48);
            veri_yolla((sayi%10000)/1000+48);
            veri_yolla((sayi%1000)/100+48);
```

```
            veri_yolla((sayi%100)/10+48);
            veri_yolla((sayi%10)+48);
      }
}

// SD16 kesme vektörü
#pragma vector = SD16_VECTOR
__interrupt void SD16ISR(void)
{
      sayi=SD16MEM0;
      SD16CCTL0&=~SD16IFG;    // Kesme bayrağı temizleniyor
}
```

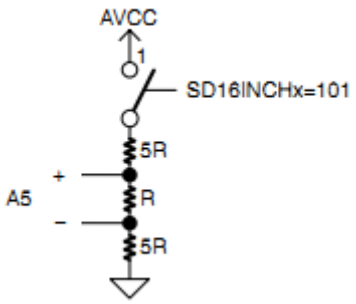
Yukarıdaki kodun çalışır halini ise resim 5'te görebilirsiniz.

Resim-5 – ADC Uygulaması

5.4) A5 Kanalını Okuma

Bölüm 5.2'de belirtilen registerlerden de görebileceğiniz üzere A5 kanalı özel bir kanaldır. Bize (Vdd-Vss)/11 gerilimini vermektedir. Buradaki gerilimi okumak için şekil 25'deki devreyi hiçbir değişiklik yapmadan kullanabilirsiniz.

A5'ten okuma yapmak için ise aşağıdaki koddan da görebileceğiniz üzere küçük bir değişiklik yapmamız gerekmektedir. Ayrıca gerilim referansı 1.2V olduğu için, Vss: 0, 1.2V: 65535 olacağından buna göre katsayımızı da programa adapte edelim.
Şekil 26'da görüleceği üzere gerilim değerimiz Vcc/5 olacaktır. A5- Vss'ye bağlanacaktır.



Şekil 26 - A5 Bölümü

İsteğimizi yerine getiren kod öbeğini aşağıda görebilirsiniz.

```
//                 MSP430F2013
//             .   -----------------
//          /|\ |                 |
//           |  |                 |
//           --|RST              |
//             |             P1.0|-->Clock
//             |A5           P1.1|-->Data
//             |A5- = VSS    P1.2|-->Enable
//             |                 |
//

#include  <msp430.h>
#include "delay.h"
#include "lcd.h"

char  flag=0,
      i     =0;
unsigned int      sayi=0,
                  temp=0;

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;  // Watchdog timer kapatılıyor

      // Çalışma frekansı 1MHz
      BCSCTL1 = CALBC1_8MHZ;
      DCOCTL = CALDCO_8MHZ;
      BCSCTL3= LFXT1S_2;

    P1DIR = 0x07; // P1.0, P1.1, P1.2 çıkış olarak tanımlanıyor
                  // P1.0, P1.1, P1.2 GPIO, P1.4 A2 olarak işlev yapacak

    // Giriş frekansı 48'e bölünüyor, örnekleme frekansımız 166kHz
    // 1.2V referans gerilimi aktif
    // SMCLK frekans kaynağı olarak kullanılacak, 8MHz
    SD16CTL= SD16XDIV1 | SD16XDIV0 | SD16REFON | SD16SSEL_1;
    // A5+ seçildi, kazanç 1, 4 örnek sonra kesmeye girilecek
    // A5=Vss/11 olacak
    SD16INCTL0=SD16INCH_5;
    // Kesme aktif
    // Çevrim unipolar, yani 0V=0, 3.3V=0xFFFF olacak
    // Çevrim memory'nin son 16 bitine atılacak
    // Çevrim unipolar
    // 256OSR
    SD16CCTL0 =  SD16UNI | SD16IE;
    // Tüm A-'ler VSS'ye bağlanacak
    SD16AE = 0;

    DelayMs(250);             // LCD'nin işlemcisinin stabil olması
                              // için 250ms bekleniyor
      lcd_init();

      lcd_gotoxy(1,1);
      lcd_yaz("FxDev.org-MSP430");
      lcd_gotoxy(2,1);
      lcd_yaz("SD16 ADC Orn. 2");
      DelayMs(2000);

      // Çevrime başlanıyor
```

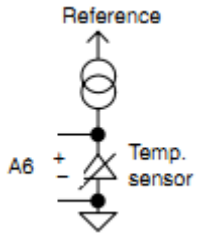
```
    SD16CCTL0 |= SD16SC;

      _EINT();                  // Enter LPM0 w/ interrupt

      lcd_clear();
      lcd_gotoxy(1,1);
      lcd_yaz("A5=");

      for(;;)
      {
            temp=sayi*0.001831;
            lcd_gotoxy(1,4);
            veri_yolla((temp%1000)/100+48);
            veri_yolla('.');
            veri_yolla((temp%100)/10+48);
            veri_yolla((temp%10)+48);
            veri_yolla('V');
      }
}

// SD16 kesme vektörü
#pragma vector = SD16_VECTOR
__interrupt void SD16ISR(void)
{
      sayi=SD16MEM0;
      SD16CCTL0&=~SD16IFG;    // Kesme bayrağı temizleniyor
}
```

Devrenin çalışan görüntüsünü ise resim 6'da görebilirsiniz.

Resim-6 – A5 İç Kanal Okuması

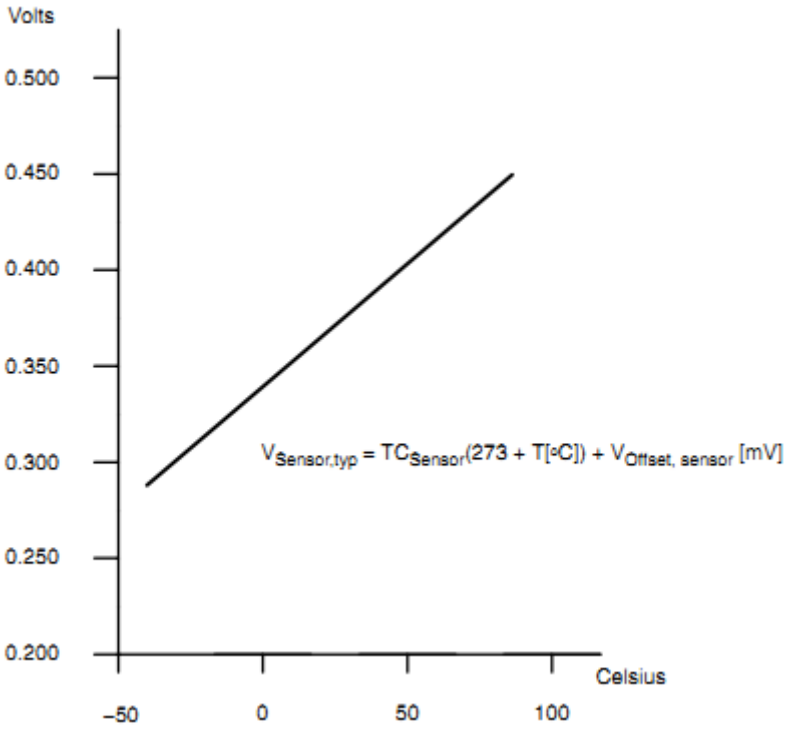
5.5) A6/Temp Kanalını Okuma

MSP430, yavaş yavaş çoğu mikrodenetleyicide görmeye başladığımız iç sıcaklık sensörüyle birlikte gelmektedir. Bu da profesyonel uygulamalarda hem entegrenin bulunduğu ortamın sıcaklığını öğrenmede, hem de iletişim hızları gibi kritik uygulamalarda, sıcaklıkla değişen frekansı tekrar kalibre etmek için kullanılmaktadır. MSP430'un içerisindeki sıcaklık sensörü şekil 27'de göreceğiniz üzere A6 kanalına bağlanmıştır.



Şekil 27 - Sıcaklık Sensörü

MSP430'un içerisindeki bu sıcaklık sensörü şekil 28'den de görüleceği üzere bize lineer bir karakteristik göstermektedir.



Şekil 28 - A6/Temp Kanalı Karakteristiği

Şekil 28'deki karakteristiği verilen sensörü kullanabilmek için şekil 29'daki bilgiler göz önüne alınmalıdır.

PARAMETER		TEST CONDITIONS	V_{CC}	MIN	TYP	MAX	UNIT
TC_{Sensor}	Sensor temperature coefficient			1.18	1.32	1.46	mV/°C
$V_{Offset,Sensor}$	Sensor offset voltage			-100		100	mV
V_{Sensor}	Sensor output voltage(2)	Temperature sensor voltage at T_A = 85°C	3 V	435	475	515	mV
		Temperature sensor voltage at T_A = 25°C		355	395	435	
		Temperature sensor voltage at T_A = 0°C		320	360	400	

(1) Values are not based on calculations using TC_{Sensor} or $V_{Offset,sensor}$ but on measurements.
(2) The following formula can be used to calculate the temperature sensor output voltage:
$V_{Sensor,typ} = TC_{Sensor} (273 + T [°C]) + V_{Offset,sensor}$ [mV] or
$V_{Sensor,typ} = TC_{Sensor} T [°C] + V_{Sensor}(T_A = 0°C)$ [mV]

Şekil 29 - Temp. Bilgileri

Şekil 29 göz önüne alındığında Voffset gerilimi 0 kabul edilirse sıcaklık değeri aşağıdaki formülle hesaplanabilir;

T=(Vsensor/TCsensor)-273
T=(Vsensor/1.32)-273

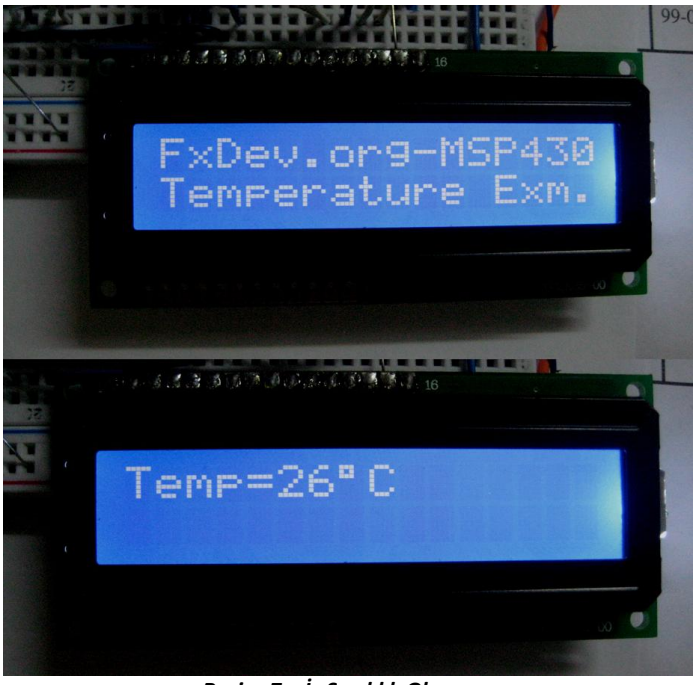
Örneğin Vsensor 395 olursa T, yukarıdaki formül işletildiğinde 26C bulunacaktır.

Tüm bunlar göz önüne alındığında şekil 25'teki devre kurulduğunda yazılacak kod aşağıdaki gibi olacaktır.

```
//                  MSP430F2013
//            .    -----------------
//         /|\ |                   |
//          |  |                   |
//           --|RST                |
//             |                P1.0|-->Clock
//     Vin+ -->|A6+/Temp        P1.1|-->Data
//             |A6- = VSS       P1.2|-->Enable
//             |                    |
//

#include  <msp430.h>
#include "delay.h"
#include "lcd.h"

unsigned int      sayi=0,
                  temp=0,
                  Vsensor=0;

void main(void)
{
    WDTCTL = WDTPW +WDTHOLD;  // Watchdog timer kapatılıyor

      // Çalışma frekansı 1MHz
      BCSCTL1 = CALBC1_8MHZ;
      DCOCTL = CALDCO_8MHZ;
      BCSCTL3= LFXT1S_2;

    P1DIR = 0x07;                    // P1.0, P1.1, P1.2 çıkış olarak
tanımlanıyor
                                                 // P1.0, P1.1, P1.2 GPIO,
P1.4 A2 olarak işlev yapacak

    // Giriş frekansı 48'e bölünüyor, örnekleme frekansımız 166kHz
```

```
    // 1.2V referans gerilimi aktif
    // SMCLK frekans kaynağı olarak kullanılacak, 8MHz
    SD16CTL= SD16XDIV1 | SD16XDIV0 | SD16REFON | SD16SSEL_1;
    // A6+ seçildi, kazanç 1, 4 örnek sonra kesmeye girilecek
    // A6=temp olacak
    SD16INCTL0=SD16INCH_6;
    // Kesme aktif
    // Çevrim unipolar, yani 0V=0, 3.3V=0xFFFF olacak
    // Çevrim memory'nin son 16 bitine atılacak
    // Çevrim unipolar
    // 256OSR
    SD16CCTL0 =  SD16UNI | SD16IE;
    // Tüm A-'ler VSS'ye bağlanacak
    SD16AE = 0;

    DelayMs(500);             // LCD'nin işlemcisinin stabil olması
                              // için 250ms bekleniyor
      lcd_init();

      lcd_gotoxy(1,1);
      lcd_yaz("FxDev.org-MSP430");
      lcd_gotoxy(2,1);
      lcd_yaz("Temperature Exm.");
      DelayMs(2000);

      // Çevrime başlanıyor
    SD16CCTL0 |= SD16SC;

      _EINT();                 // Enter LPM0 w/ interrupt

      lcd_clear();
      lcd_gotoxy(1,1);
      lcd_yaz("Temp=");

      for(;;)
      {
            Vsensor=(int)((float)sayi*0.00916);
            temp=(int)((float)Vsensor*1.32)-273;
            lcd_gotoxy(1,6);
            veri_yolla((temp%1000)/100+48);
            veri_yolla((temp%100)/10+48);
            veri_yolla(0xDF);
            veri_yolla('C');
      }
}

// SD16 kesme vektörü
#pragma vector = SD16_VECTOR
__interrupt void SD16ISR(void)
{
      sayi=SD16MEM0;
      SD16CCTL0&=~SD16IFG;    // Kesme bayrağı temizleniyor
}
```

Devrenin çalışır halini resim 7’de görebilirsiniz.

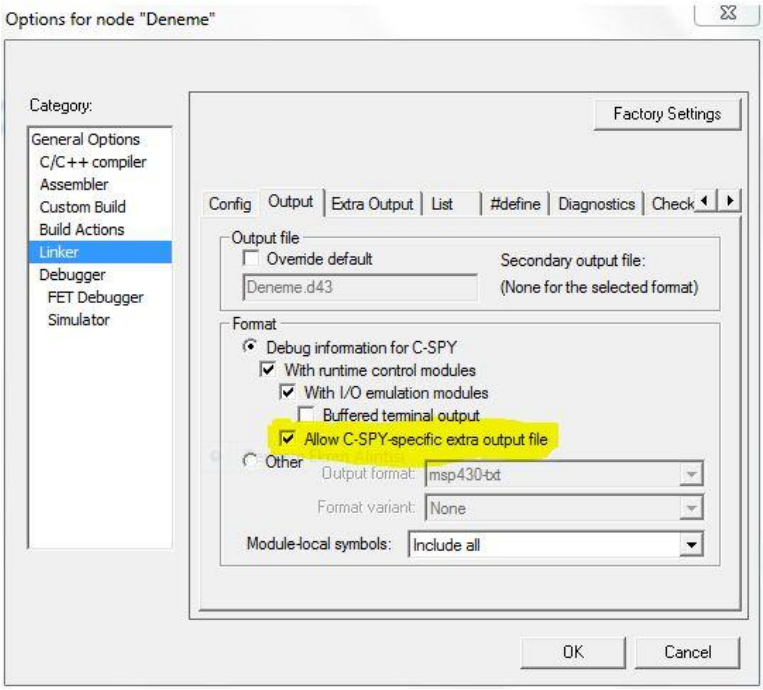


Resim-7 – İç Sıcaklık Okuması

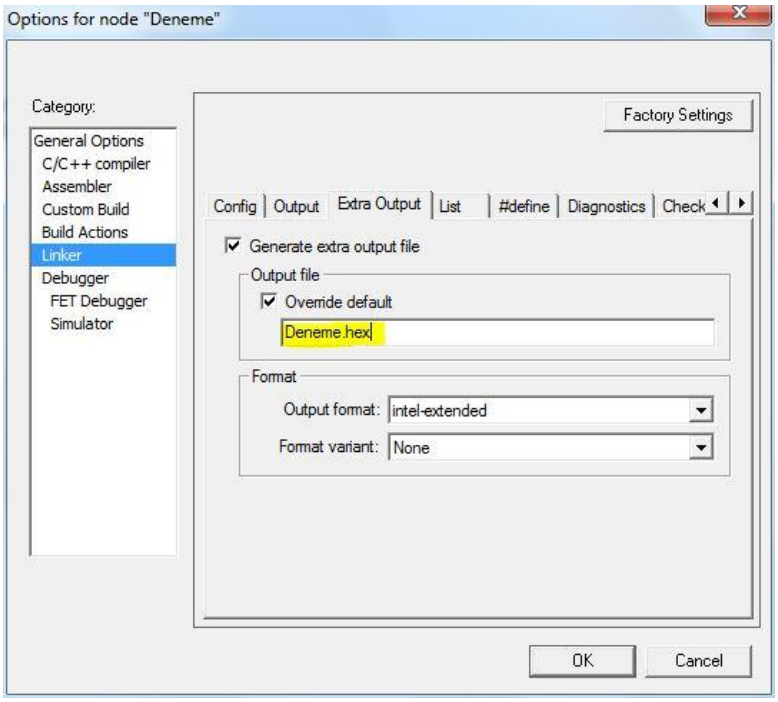
BÖLÜM 6 – EK KISIM/ DERLEYİCİLER ve KULLANILAN DONANIM

6.1) IAR Proje Özellikleri

MSP430'u Proteus'ta simüle etmek için IAR'da şekil 30 ve 31'deki aşağıdaki değişikliklerin yapılması gerekmektedir.



Şekil 30 - IAR Değişiklikleri 1



Şekil 31 - IAR Değişiklikleri 2

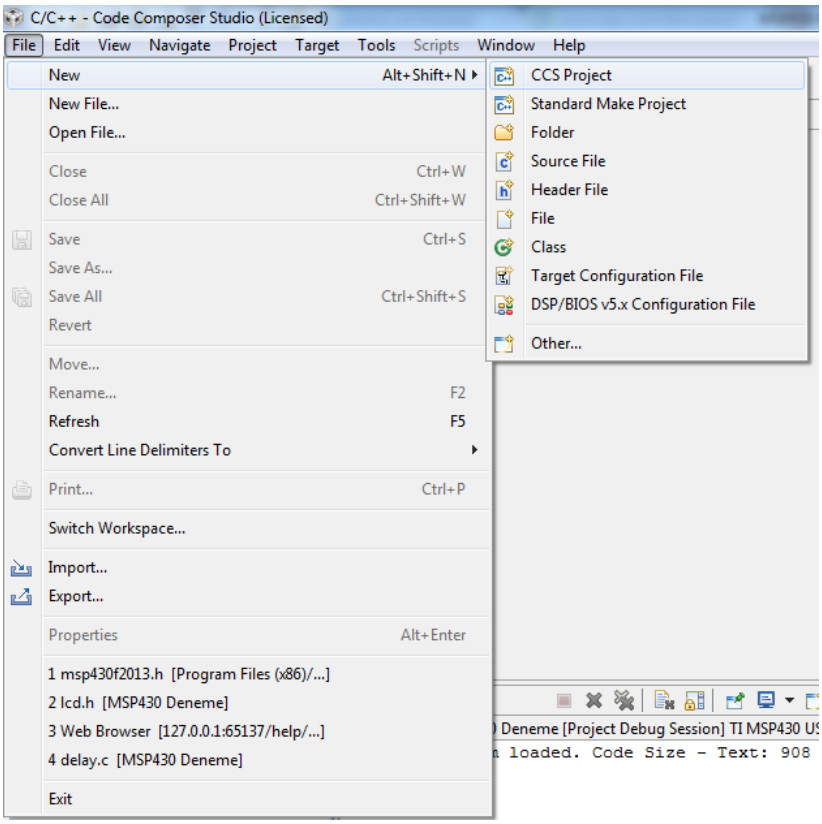
6.2) Code Composer Studio

Özellikle MSP430 mikrodenetleyicisini programlamak için kullanılan Code Composer Studio, TI'ın yayınladığı, Eclipse tabanlı bir derleyicidir. Özellikle IAR'ın sert yapısının yanında sunduğu Eclipse IDE'si avantajları kullanıcıya çok fazla kolaylık sağladığı için bu kitap içerisindeki tüm örnekler bu IDE ile yapılmıştır.

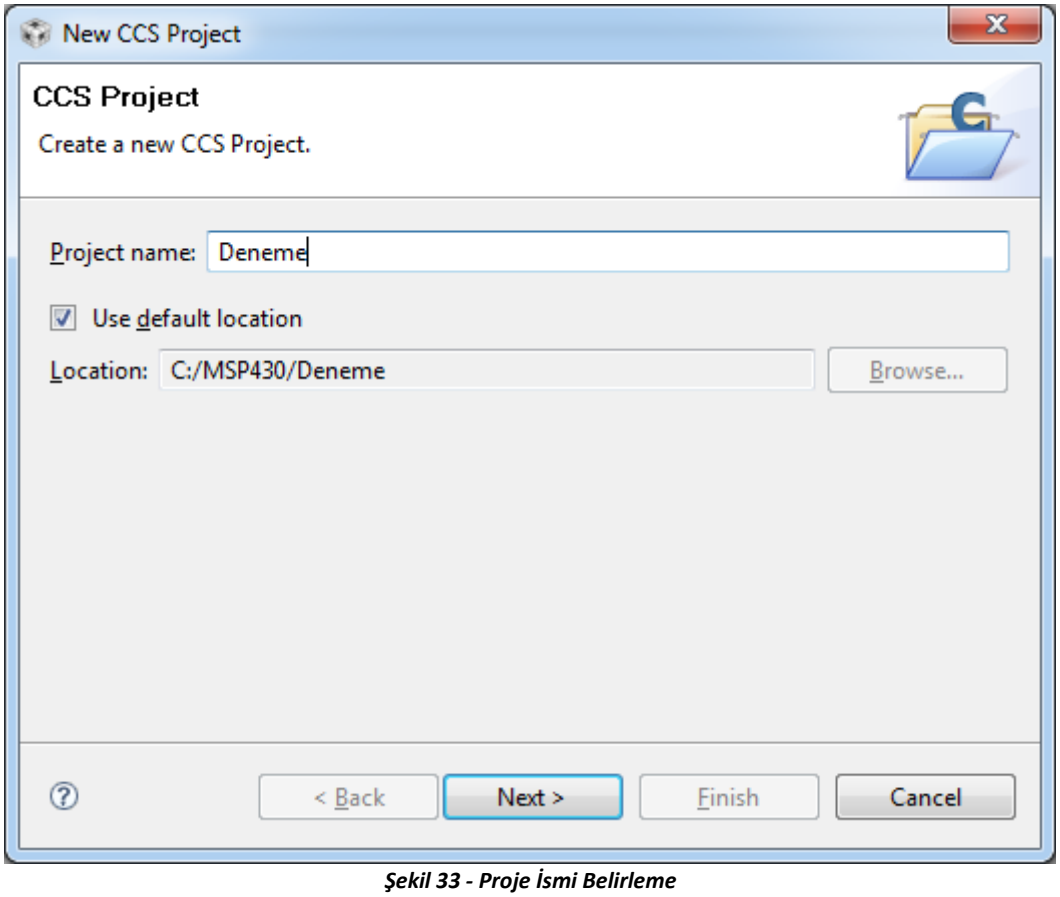
Code Composer Studio'nun sınırlı versiyonları aşağıdaki linkten indirilebilir. Kullandığımız mikrodenetleyicilerin hafızası sınırlı olduğundan indirilen sınırlı versiyonlar uygulamalarımızda bizlere sıkıntı çıkarmamaktadır.

http://focus.ti.com/docs/toolsw/folders/print/ccstudio.html?DCMP=dsp_ccs_v4&HQS=Other+OT+ccs

Code Composer Studio'da yeni bir proje oluşturmak için aşağıdaki adımlar sırasıyla uygulanmalıdır.

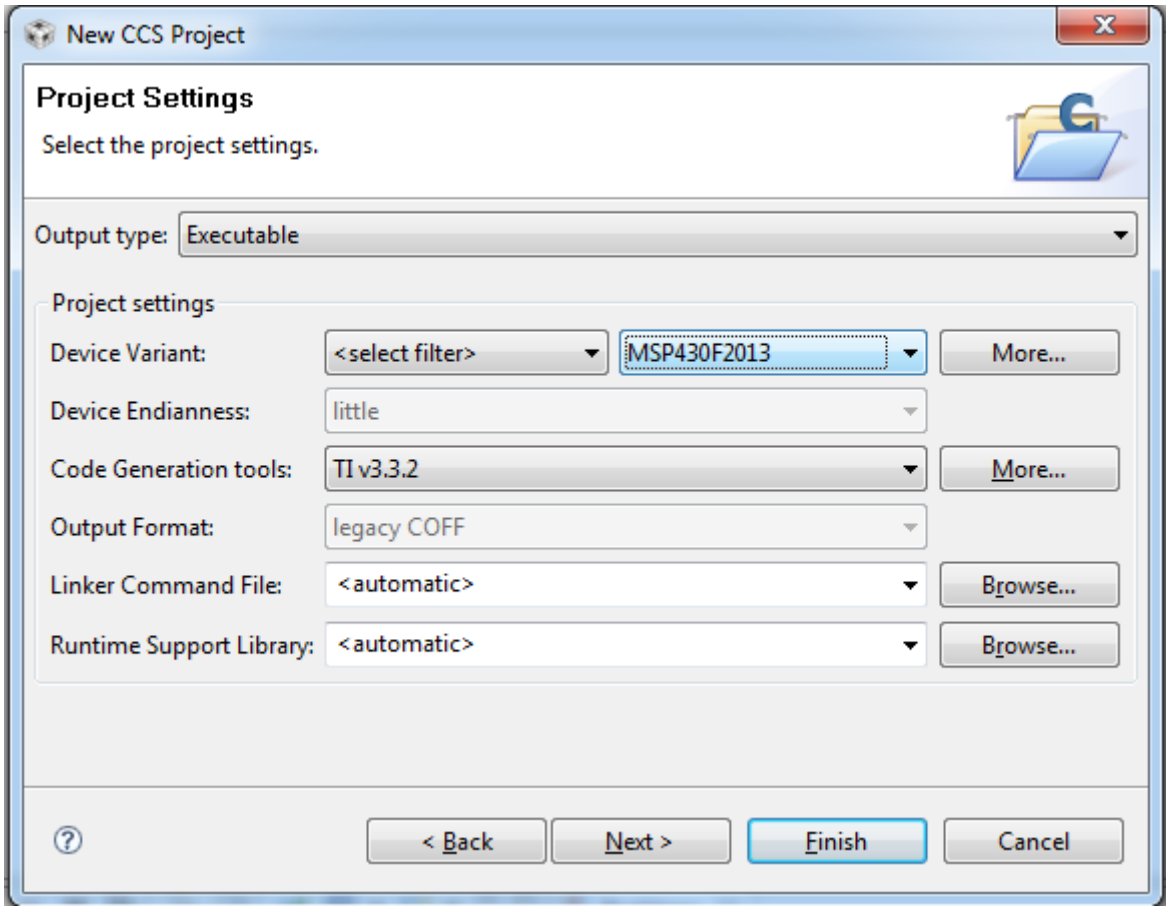


Şekil 32 - Proje Oluşturma



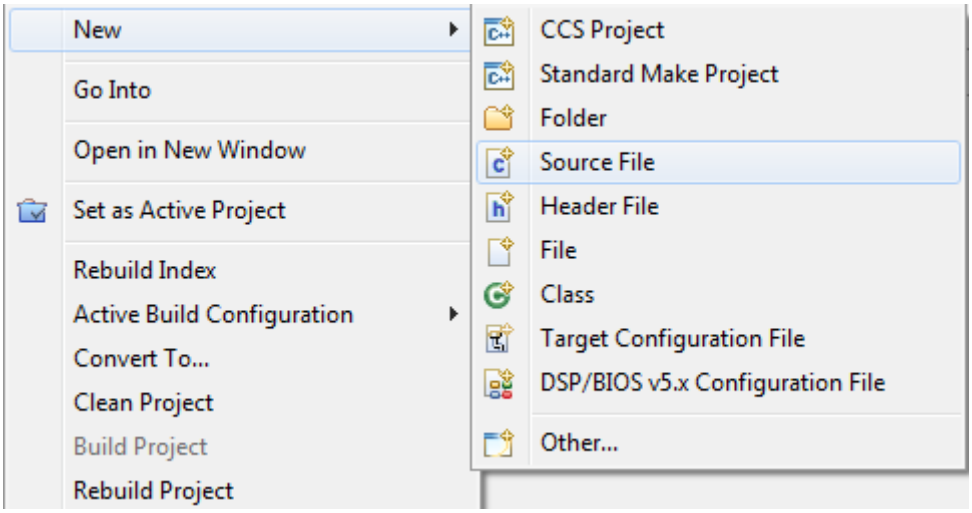
Şekil 33 - Proje İsmi Belirleme

Bu kısımdan sonra next sekmelerine basarak donanım seçme ekranına geliniz.



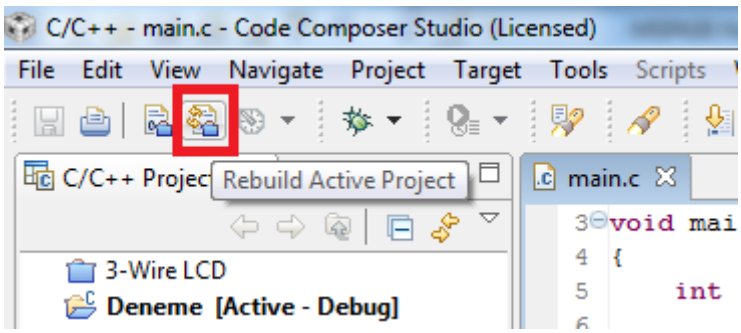
Şekil 34 - Donanım Seçim Ekranı

Daha sonra ise Finish diyerek projemizi oluşturmalıyız. Daha sonrasında proje isminin üzerine gelip sağ tıklayarak new ve source file diyerek projemize main.c dosyasını ekleyelim.



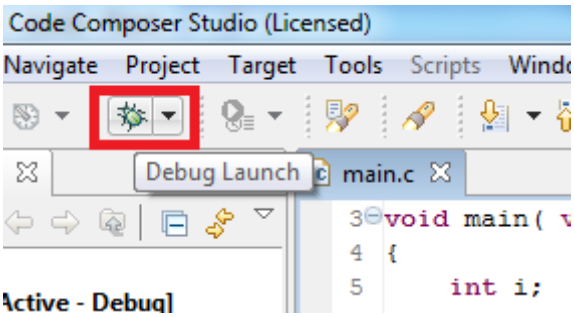
Şekil 35 - Source Dosyası Ekleme

Projede main.c dosyasını oluşturduktan sonra kitabımızdaki ilk örnek kodları bu dosyaya yapıştıralım ve şekil 36'da görülen buton ile derlemeyi gerçekleştirelim.



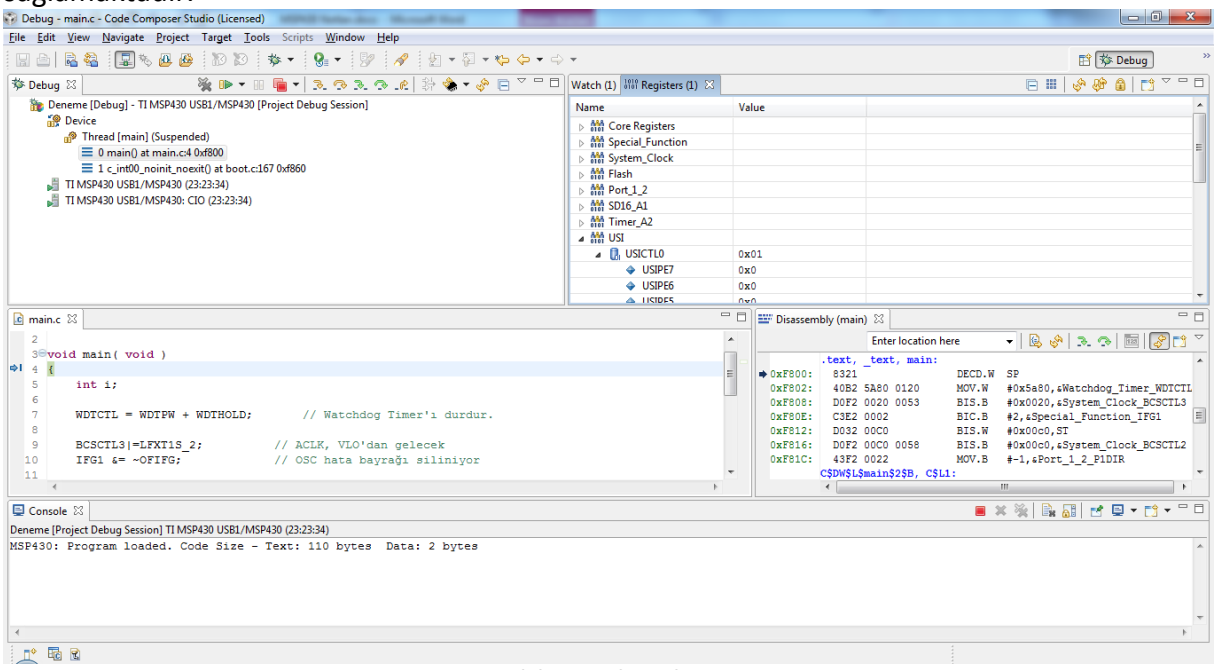
Şekil 36 - Derleme İşlemi

Daha sonra ise şekil 37'de görülen böcek şekline basarak debug işlemine geçerek donanımımıza programımızı yükleyelim.



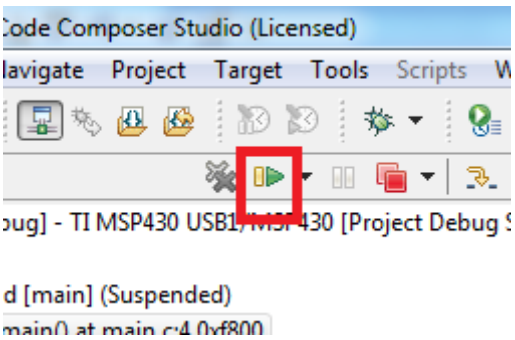
Şekil 37 - Donanıma Program Yükleme

Şekil 38'den de görüleceği üzere debug ekranı bize donanımız çalışırken tüm değişiklikleri aynı anda görmemizi sağlıyor. Bu bölüm özellikle hata ayıklamada bizlere oldukça kolaylık sağlamaktadır.



Şekil 38 - Debug Ekranı

Daha sonrasında programımızı çalıştırmak için sadece şekil 39'daki play tuşuna basmamız yeterlidir.

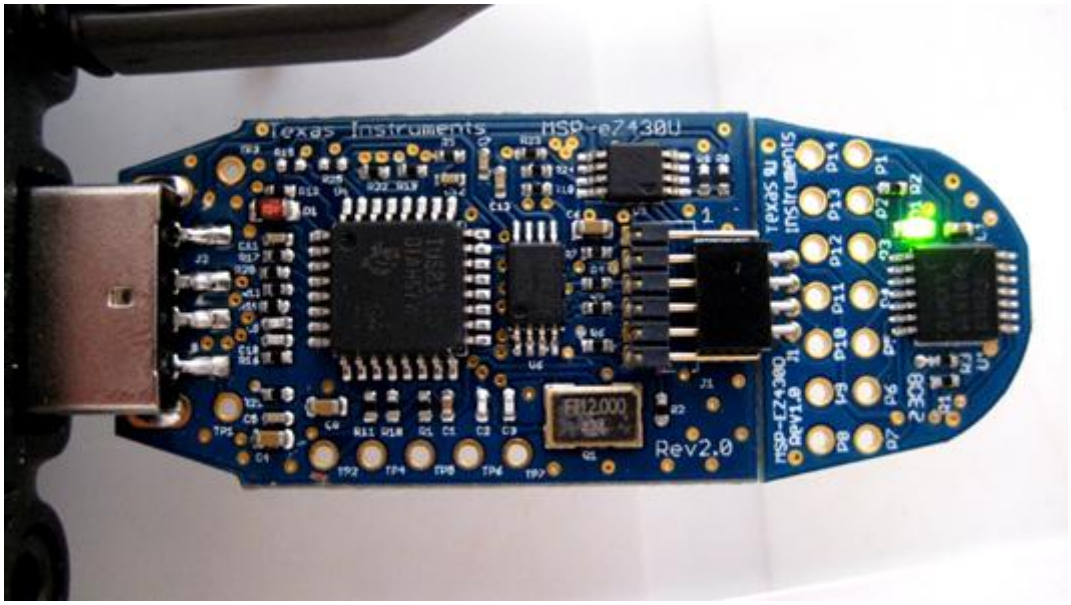


Şekil 39 - Programı Çalıştırmak

Ayrıca debug ekranında programımızı adım adım yürütmek de mümkündür. Elbette tüm bu işlemleri kullanabilmek için TI'ın ürettiği kitlere ya da programlayıcılara sahip olunmalıdır.

6.3) Kullanılan Donanım –

Tüm kitap boyunca kullandığım donanım resim 8'de görülen EZ430-F2013 donanımıdır. Bu donanım üzerinde dahili programlayıcı ve MSP4302013 donanımını barındırmaktadır. Detaylı bilgilere TI'ın sitesinden ulaşılabilmektedir.



Resim 8 – Kullanılan Donanım

Donanımla ilgili detaylı bilgiye aşağıdaki linkten ulaşabilirsiniz;

http://focus.ti.com/docs/toolsw/folders/print/ez430-f2013.html

BİTİRİRKEN…

Sizlere 5 bölüm boyunca Texas Inst. gelişmesi ve dünya çapında kullanımının artması için yoğun çaba harcadığı MSP430 mikrodenetleyicisini, IAR ve Code Composer Studio ile birlikte anlatmaya çalıştım. Kitabta özellikle sıkça kullandığımız mikrodenetleyicilerde görmediğimiz 16 bit çözünürlüklü, sigma/delta ADC birimine daha fazla eğilmeye özen gösterdim. Ayrıca diğer kitaplarımdan farklı olarak uygulamaların resimlerini de elimden geldiğince ilk kez mikrodenetleyici öğrenecek kişiler için koymaya çalıştım.

Mikrodenetleyicinin düşük hafıza ve kullanım zorluğu gibi dezavantajlarından dolayı SPI, I^2C ve UART gibi birimlerine kitabın ilk baskısında girmeyi düşünmedim, ileride elime daha güçlü MSP430 donanımı geçtiği taktirde kitabın ikinci baskısında bu kısımlarında ekleneceğini buradan söylemek isterim.

Kitabı takip eden ve katkısı olan herkese teşekkürler…

Fırat Deveci
Ağustos 2011
fxdev@fxdev.org
www.fxdev.org