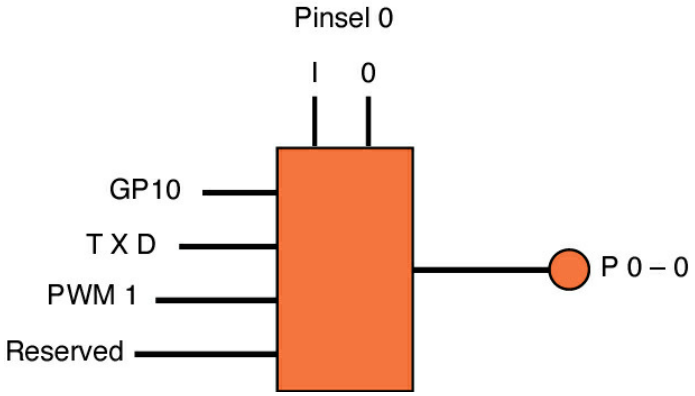
BÖLÜM 2 – ARM GİRİŞ ÇIKIŞ AYARLARI

2.1) PINSEL Ayarı

ARM'ların çoğunda pin sayıları değişse de pin ayarlamaları yapmak için öncelikle aşağıda şekli görülen multiplexer yardımı ile o pine ait hangi görevin kullanılacağı belirtilmelidir. Aşağıda bu görevi yapan Pin Collect Block yapısı görülebilir. Şekil-1'den de görüleceği üzere P0.0 pininin görevlerini seçmek için PINSEL0<0-1> bitlerini ayarlamak gerekmektedir.



Şekil 2.1 - Pin Collect Block Multiplexer Yapısı

Bu görevi yerine getiren registerler **PINSEL0** ve **PINSEL1**'dir. Pin sayısına göre PINSEL sayısı da değişmektedir. İlk örnek için kullanacağımız LPC2104'te 32 giriş-çıkış pini bulunduğundan 16bit boyutunda **PINSEL0** ve **PINSEL1** registerimizi ayarlamamız yeterli olacaktır. Aşağıdaki tabloda LPC2104'ün **PINSEL0** içeriğinin ilk değerleri görülebilir.

Bit	Symbol	Value	Function	Reset value
1:0	P0.0	00	GPIO Port 0.0	0
		01	TXD (UART0)	
		10	PWM1	
		11	Reserved	
3:2	P0.1	00	GPIO Port 0.1	0
		01	RxD (UART0)	
		10	PWM3	
		11	EINT0	
5:4	P0.2[1]	00	GPIO Port 0.2	0

Tablo 2.1 - Pin Collect Block Multiplexer Yapısı

2.2) GPIO (General Purpose IN/OUT)

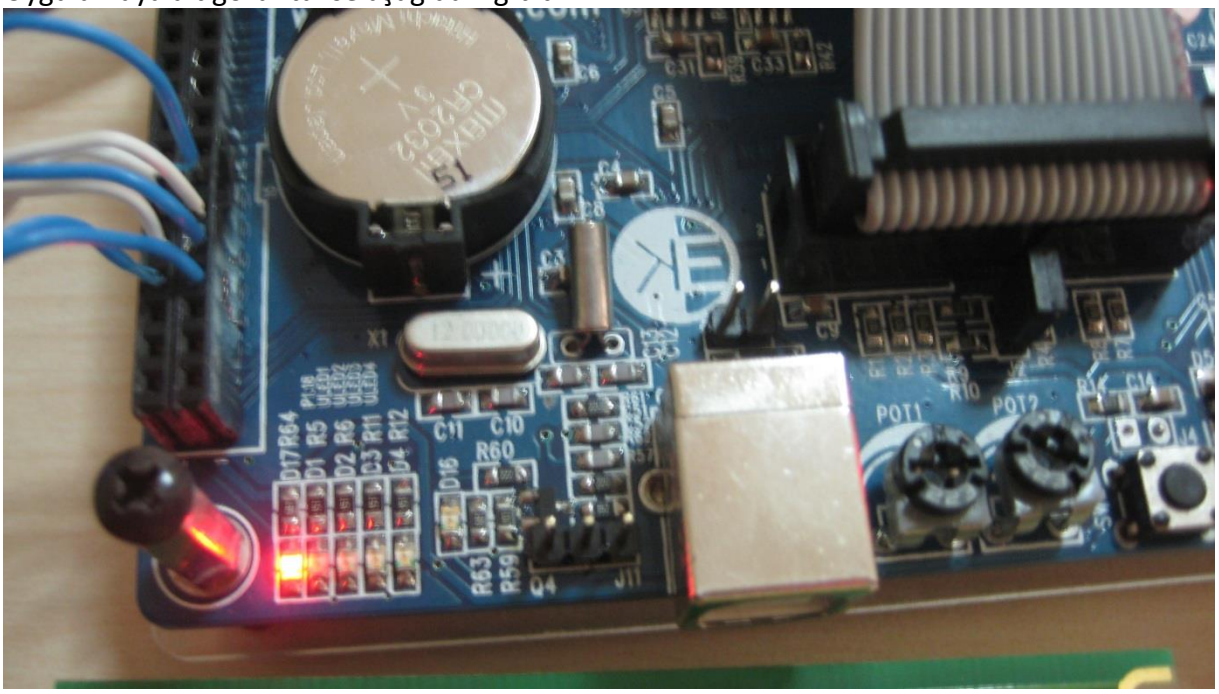
Pinleri giriş, çıkış işlemlerinde kullanmak için GPIO kullanılması gerekmektedir. ARM reset anında tüm pinleri GPIO yapar ve pinler bu anda giriş halindedir.
GPIO yapısını kontrol eden 4 adet register vardır;

- **IODIR:** Pinleri giriş çıkış olarak ayarlamak için kullanılan registerdir. 1 ise çıkış, 0 ise giriştir.
- **IOSET:** İstenilen pinin high seviyeye çekilmesini sağlar.
- **IOCLR:** İstenilen pinin low seviyeye çekilmesini sağlar.
- **IOPIN:** İstenilen pinin durumu okumak ve yazmak için kullanılır.

Yapacağımız ilk örnekte **5 adet ledi yakıp söndürelim.** Bunu sağlayan kod aşağıda görülmektedir.

```
#include <LPC23xx.h>
#include "delay.h"

char ULED[6]={18,10,21,26,27,28};
char i=0;

int main(void)
{
	PINSEL0=0;		// P0 GPIO olacak
	PINSEL1=0;
	PINSEL2=0;		// P1 GPIO olacak
	PINSEL3=0;

	IODIR0		= 0xFFFFFFFF; // P0'ın hepsi çıkış
	IODIR1		= 0xFFFFFFFF; // P1'in hepsi çıkış

	while(1)
	{
		IOCLR1=1<<ULED[0];
		DelayMs(200);
		IOSET1=1<<ULED[0];
		DelayMs(200);
		for(i=1;i<5;i++)
		{
			IOSET0=1<<ULED[i];
			DelayMs(200);
			IOCLR0=1<<ULED[i];
			DelayMs(200);
		}
	}
}
```

Uygulamaya ait görüntü ise aşağıdaki gibidir.



Şekil 2.2 – LED Uygulaması

Bu uygulamada ise daha sonraki uygulamalarda kullanacağımız karakter LCD kütüphanesini oluşturalım. Daha önce PIC programlama kitabımı takip edenlerin aşina olacağı kodlar (sırayla lcd.h ve lcd.c), aşağıda görülebilir.

```
/*---------------www.Firatdeveci.com-----------------------------------
------
 *    lcd_init();       ile LCD ilk ayarlari yapilir
 *    veri_yolla('c');  ile char ifade gönderilir
 *    lcd_yaz("data");  ile string ifade gönderilir
 *    lcd_gotoxy(y,x);  ile LCD'nin istenilen bölgesine gidilir
 *    lcd_clear();      ile LCD silinir
 *-------------------------------------------------------------------*/

#define LCD_DIR         IO0DIR
#define LCD_SET         IO0SET
#define LCD_CLR         IO0CLR
#define LCD_RS          19
#define LCD_RW          14
#define LCD_E           0
#define LCD_D4          6
#define LCD_D5          3
#define LCD_D6          10
#define LCD_D7          16
#define sutun           16    // Kaç sütun oldugu bilgisi

/* 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)
#define CiftSatir4Bit  40     // 4 bit ara birim, 2 satir, 5*7 piksel

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

Yukarıda da görüleceği üzere lcd.h dosyasında öncelikle PORT yönü, SET ve CLR isimleri ile LCD portlarının hangi numaralı pinlere bağlı olduğu belirtilir. lcd_init() fonksiyonundan önce de herhangi bir port yönlendirmesine gerek yoktur. Diğer pinlerin işleyişi ise LCD çalışması sırasında etkilenmez. lcd.c dosyası ise aşağıdaki şekilde tanımlanmıştır.

```
#include <LPC23xx.h>
#include "delay.h"
#include "lcd.h"

#define SPECIAL_CHAR    1

unsigned char special_char_1[8]={0x00,0x11,0x11,0x11,0x19,0x16,0x10,0x00};
//unsigned char special_char_2[8]={0x1F,0x11,0x15,0x11,0x17,0x11,0x1F,0x1F};
```

```
//unsigned char special_char_3[8]={0x1F,0x1B,0x1F,0x11,0x1B,0x11,0x1F,0x1F};
//unsigned char special_char_4[8]={0x1F,0x1F,0x11,0x17,0x13,0x17,0x1F,0x1F};
//unsigned char special_char_5[8]={0x1F,0x13,0x15,0x15,0x15,0x13,0x1F,0x1F};
//unsigned char special_char_6[8]={0x1F,0x13,0x15,0x15,0x15,0x13,0x1F,0x0};
//unsigned char special_char_7[8]={0x1F,0x13,0x15,0x15,0x15,0x13,0x1F,0x0};
//unsigned char special_char_8[8]={0x1F,0x13,0x15,0x15,0x15,0x13,0x1F,0x0};

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

void send_byte(unsigned char b)
{
      LCD_CLR=((0x01<<LCD_D4)|(0x01<<LCD_D5)|(0x01<<LCD_D6)|(0x01<<LCD_D7));
      // Öncelikle tüm cikislar sifir yapiliyor
      LCD_SET=(0x01<<LCD_E);                                  // Yuksek
degerlikli bitler gönderiliyor
      LCD_SET=(((b>>7)&0x01)<<LCD_D7)|(((b>>6)&0x01)<<LCD_D6)|(((b>>5)&0x01)
<<LCD_D5)|(((b>>4)&0x01)<<LCD_D4);
      LCD_CLR=(0x01<<LCD_E);
      lcd_busy();

      LCD_CLR=((0x01<<LCD_D4)|(0x01<<LCD_D5)|(0x01<<LCD_D6)|(0x01<<LCD_D7));
      // Öncelikle tüm cikislar sifir yapiliyor
      LCD_SET=(0x01<<LCD_E);                                  // Düşük
degerlikli bitler gönderiliyor
      LCD_SET=(((b>>3)&0x01)<<LCD_D7)|(((b>>2)&0x01)<<LCD_D6)|(((b>>1)&0x01)
<<LCD_D5)|(((b>>0)&0x01)<<LCD_D4);
      LCD_CLR=(0x01<<LCD_E);
      lcd_busy();
}

void lcd_komut(unsigned char c)
{
      LCD_CLR=(0x01<<LCD_RS);
      send_byte(c);
}

void veri_yolla(unsigned char c)
{
      LCD_SET=(0x01<<LCD_RS);
      send_byte(c);
}

void lcd_clear(void)
{
      lcd_komut(0x01);
      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)%sutun));
      else if(x==2)
            lcd_komut(0xC0+((y-1)%sutun));
      else if(x==3)
            lcd_komut(0x94+((y-1)%sutun));
      else
            lcd_komut(0xD4+((y-1)%sutun));
}

void special_char(void)
{
      char i;
      if(SPECIAL_CHAR>=1){for(i=0x40;i<=0x47;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x40]);}}
      if(SPECIAL_CHAR>=2){for(i=0x48;i<=0x4F;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x48]);}}
      if(SPECIAL_CHAR>=3){for(i=0x50;i<=0x57;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x50]);}}
      if(SPECIAL_CHAR>=4){for(i=0x58;i<=0x5F;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x58]);}}
      if(SPECIAL_CHAR>=5){for(i=0x60;i<=0x67;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x60]);}}
      if(SPECIAL_CHAR>=6){for(i=0x68;i<=0x6f;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x68]);}}
      if(SPECIAL_CHAR>=7){for(i=0x70;i<=0x77;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x70]);}}
      if(SPECIAL_CHAR>=8){for(i=0x78;i<=0x7F;i++)
      {lcd_komut(i);veri_yolla(special_char_1[i-0x78]);}}
      lcd_komut(BirinciSatir);
}

void lcd_init()
{
      LCD_CLR =
((0x01<<LCD_RS)|(0x01<<LCD_RW)|(0x01<<LCD_E)|(0x01<<LCD_D4)|(0x01<<LCD_D5)|(
0x01<<LCD_D6)|(0x01<<LCD_D7));      // Öncelikle tüm cikislar sifir
yapiliyor
      LCD_DIR     =
LCD_DIR|((0x01<<LCD_RS)|(0x01<<LCD_RW)|(0x01<<LCD_E)|(0x01<<LCD_D4)|(0x01<<L
CD_D5)|(0x01<<LCD_D6)|(0x01<<LCD_D7));          // LCD pinleri cikis olarak
ayarlaniyor

      DelayMs(100);
      LCD_CLR=(0x01<<LCD_RW);             // LCD'ye yazim yapilacak
      lcd_komut(0x02);
      DelayMs(2);

      lcd_komut(CiftSatir4Bit);
      lcd_komut(SagaYaz);
      lcd_komut(ImlecGizle);
      lcd_clear();
      special_char();
      lcd_komut(BirinciSatir);
}
```

Yukarıda da görüleceği üzere LCD kütüphanesinin diğer 8 bit kütüphanelerden tek farkı 32 bit ve and or gibi işlemlerin kullanılmış olmasıdır. Yine göreceğiniz gibi kullanılan delay.h kütüphanesi oldukça basit tasarlanmıştır. Gerçek uygulamalarda bu delay.h kütüphanesi yerine timer kullanarak gecikmelerimizi sağlamak daha profesyonel ve iyi bir yoldur.
Aşağıda denemelerimde basitçe kullanacağım bu kütüphaneyi görebilirsiniz.

```
#define FOSC      12000000UL

void DelayUs(unsigned char U);
void DelayMs(unsigned char M);
```

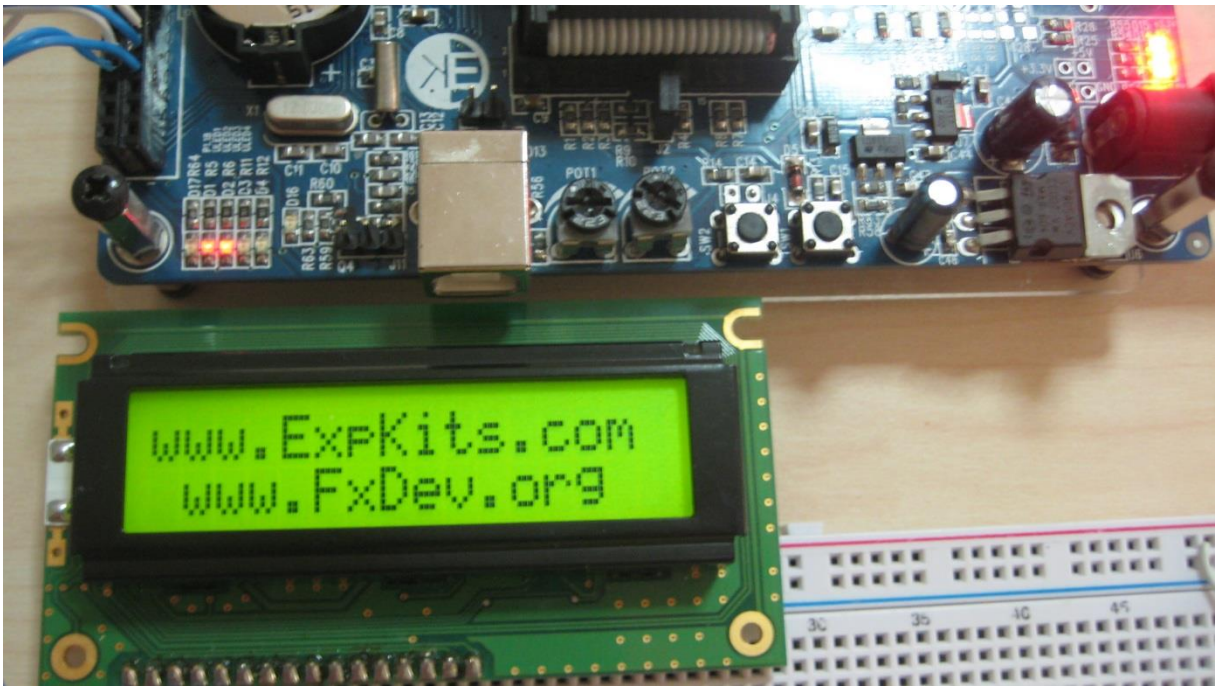
Yukarıdaki koda ait delay.c dosyası ise şöyledir.

```
#include <LPC21xx.H>
#include "delay.h"

void DelayUs(unsigned char U)
{
	while(U--)
	{
		char i=(char)(FOSC/1000000);
		while(i--);
	}
}

void DelayMs(unsigned char M)
{
	while(M--)
	{
		DelayUs(250);
		DelayUs(250);
		DelayUs(250);
		DelayUs(250);
	}
}
```

Yukarıdaki LCD kütüphanesini kullanarak ismimizi yazmak istersek aşağıdaki kodu kullanmamız yeterli olacaktır.

```
#include <LPC23xx.h>
#include "delay.h"
#include "lcd.h"

int main(void)
{
	PINSEL2=0;		// P1 GPIO olacak
	PINSEL3=0;

	lcd_init();
	lcd_yaz("www.ExpKits.com");
	lcd_gotoxy(2,1);
	lcd_yaz(" www.Firatdeveci.com");

	while(1);
}
```

Yukarıdaki kodun çalışan hali ise şekil-2.3'te görülebilir.



Şekil 2.3 – LCD Uygulaması

Yukarıdaki örneklerde de görüleceği üzere ARM'da dijital port kullanımları herhangi bir mikrodetleyiciden farklı değildir. Sadece işlemler 32bit bazında yürümektedir. Ayrıca yukarıdaki LCD kütüphanesi, herhangi bir değişiklik olmazsa ileriki örneklerde aynen kullanılacaktır.

BÖLÜM 3 – KESMELER

BÖLÜM 5 –ADC

Bu bölümde örnek olarak kullanacağımız LPC2368 serisi ARM mikrodenetleyicileri içerlerinde 0-V_{DDA} aralığında ölçüm yapabilen 10bit çözünürlüğe sahip tam 6 kanal ADC birimi bulunmaktadır. Bu birimin 10bitlik çevrim süresi yaklaşık olarak 2.44us'dir ve bu saniyede yaklaşık 400.000 çevrim anlamına gelmekte ve yaklaşık olarak 190-200kHz frekansındaki sinyallerin örneklenmesi için oldukça elverişli bir zemin hazırlamaktadır.
Pin durumları GPIO durumundayken de ADC ölçümü yapılabilir, fakat hassasiyetin arttırılması için pinlerin ADC için kullanılması gerekmektedir. **Ayrıca ADC bloğunun çalışması için PCONP registerinden ilgili bitin 1 yapılması gerekmektedir. Aksi taktirde ADC bloğu çalışmamaktadır.**
Bu bölümde öncelikle ADC çevrimi için gerekli registerler ve görevlerini verilecek daha sonrasında ise hem yazılımsal hem de donanımsal örnekler verilecektir.

5.1) ADC ile İlgili Registerler

ADC Control Register: AD0CR/ AD1CR

BIT	ADI	DEĞERİ	AÇIKLAMA	RESET
7:0	SEL	-	AD0.7:0 ve AD1.7:0 ADC kanallarını seçen bittir. Yazılım modunda bu bitlerden yalnızca biri, donanım modunda ise hepsi 1 olabilir.	0x01
15:8	CLKDIV	-	ADC için gerekli saat sinyalini ayarlama bitleridir. ADC çevrimi için ADCLK değeri en fazla 4.5MHz olmalıdır. Bu değer CLKDIV=(PCLK/ADCLK)-1 ile hesaplanabilir.	0x00
16	BURST	1 0	Bu bitin set edilmesi durumunda SEL alanında seçilen kanal CLKS alanında seçilen değere göre çevrime başlar. Eğer START bitleri 0 değilse çevrim başlamaz. Çevrimler yazılım kontrollüdür ve 11 cycle beklenmelidir.	0
19:17	CLKS	 000 001 010 011 100 101 110 111	BURST modunda kaç cycle'da çevrim yapılacağı belirtilir. Saat darbeleri aynı zamanda çözünürlük bitlerini de belirler. 11 cycle / 10bit çözünürlük 10 cycle / 9bit çözünürlük 9 cycle / 8bit çözünürlük 8 cycle / 7bit çözünürlük 7 cycle / 6bit çözünürlük 6 cycle / 5bit çözünürlük 5 cycle / 4bit çözünürlük 4 cycle / 3bit çözünürlük	000
20	-	-	Ayrılmış Bit	NA
21	PDN	1 0	ADC açık ADC kapalı	0
23:22	-		Ayrılmış Bit	NA
26:24	START	 000 001 010 011 100 101 110 111	BURST biti 0 yapıldığında ADC çevriminin neye göre başlayacağını belirtir. Çevrim başlamaz. Çevrim hemen başlar. Çevrim P0.16'da oluşan kenar değişikliği ile başlar Çevrim P0.22'de oluşan kenar değişikliği ile başlar Çevrim MAT0.1'de oluşan kenar değişikliği ile başlar Çevrim MAT0.3'de oluşan kenar değişikliği ile başlar Çevrim MAT1.0'de oluşan kenar değişikliği ile başlar Çevrim MAT1.1'de oluşan kenar değişikliği ile başlar	000
27	EDGE	 1 0	Bu bit sadece START bitleri 010 ve 111 arasındayken kullanılır. CAP/MAT girişindeki sinyalin düşen kenarında çevrimi başlat. CAP/MAT girişindeki sinyalin yükselen kenarında çevrimi başlat.	0
31:28	-	-	Ayrılmış Bit	NA

ADC Data Register: AD0GDR/AD1GDR

BIT	ADI	AÇIKLAMA	RESET
5:0	-	Ayrılmış Bit	NA
15:6	RESULT	Bu alanda ADC referans geriliminin V_{DDA}'ya göre olan bölümü bu alanda saklanır. Yani bu sonuç, ADC çevriminin sonucudur.	NA
23:16	-	Ayrılmış Bit	NA
26:24	CHN	Bu alan en son hangi kanalın çevrildiği bilgisini taşır.	
29:27	-	Ayrılmış Bit	NA
30	OVERUN	Bu bit önceki okunan değer yok olduğunda set edilir. Bu değerin okunmasıyla bu bit sıfırlanır.	0
31	DONE	Bu bit ADC çevrimi bittiğinde set edilir, bu değerin okunması ile de sıfırlanır.	0

ADC Global Start Register: ADGSR

BIT	ADI	DEĞERİ	AÇIKLAMA	RESET
15:0	-	-	Ayrılmış Bit	NA
16	BURST	1 0	Bu bitin set edilmesi durumunda SEL alanında seçilen kanal CLKS alanında seçilen değere göre çevrime başlar. Eğer START bitleri 0 değilse çevrim başlamaz. Çevrimler yazılım kontrollüdür ve 11 cycle beklenmelidir.	0
23:17	-	-	Ayrılmış Bit	NA
26:24	START	 000 001 010 011 100 101 110 111	BURST biti 0 yapıldığında ADC çevriminin neye göre başlayacağını belirtir. Çevrim başlamaz. Çevrim hemen başlar. Çevrim P0.16'da oluşan kenar değişikliği ile başlar Çevrim P0.22'de oluşan kenar değişikliği ile başlar Çevrim MAT0.1'de oluşan kenar değişikliği ile başlar Çevrim MAT0.3'de oluşan kenar değişikliği ile başlar Çevrim MAT1.0'de oluşan kenar değişikliği ile başlar Çevrim MAT1.1'de oluşan kenar değişikliği ile başlar	000
27	EDGE	 1 0	Bu bit sadece START bitleri 010 ve 111 arasındayken kullanılır. CAP/MAT girişindeki sinyalin düşen kenarında çevrimi başlat. CAP/MAT girişindeki sinyalin yükselen kenarında çevrimi başlat.	0
31:28	-	-	Ayrılmış Bit	NA

ADC Status Register: AD0STAT/AD1STAT

BIT	ADI	AÇIKLAMA	RESET
7:0	DONE0-7	Bu bit ADC çevrimlerinin sonucunu gösterir. Hangi kanal bittiyse, o bit birdir.	0
15:8	OVERUN0-7	Hangi ADC kanalının değeri yenilenmişse o kanalın değeri bir olur.	0

16	ADINT	ADC kesme bayrağı	0
31:17	-	Ayrılmış Bit	NA

ADC Interrupt Enable Register: AD0INTEN/AD1INTEN

BIT	ADI	AÇIKLAMA	RESET
7:0	ADINTEN0-7	Hangi ADC kanalının kesme oluşturacağını belirten bittir.	0
8	ADGINTEN	0: Sadece ADINTEN7:0 kanallarında kesme oluşabilir. 1: Sadece Global Done'da kesme oluşabilir.	1
31:9	-	Ayrılmış Bit	NA

ADC Global Data Register: AD0GDR/AD1GDR

BIT	ADI	AÇIKLAMA	RESET
5:0	-	Ayrılmış Bit	NA
15:6	RESULT	Çevrim sonucunun saklandığı alandır (10bit)	NA
29:16	-	Ayrılmış Bit	NA
30	OVERUN	Bu bit BURST modunda çevrim sonucunun silinmesi sonucu set edilir.	NA
31	DONE	Bu bit ADC çevrimi bittiğinde set edilir, bu değerin okunması ile de sıfırlanır.	NA

Görüleceği üzere ARM mikrodenetleyicilerde ADC'yi kontrol etmek için oldukça detaylı kontrol registerleri bulunmaktadır. Bu da bizlere ADC'yi etkin kullanabilmek için oldukça fazla alternatif sunmaktadır. Çevrim sonucunun kaybolmaması, her çevrim değerinin ayrı registerlerde kaydedilmesi, yerine göre değişmekle birlikte, diğer mikrodeneyleyicilere (PIC ve AVR) göre oldukça büyük bir avantajdır.

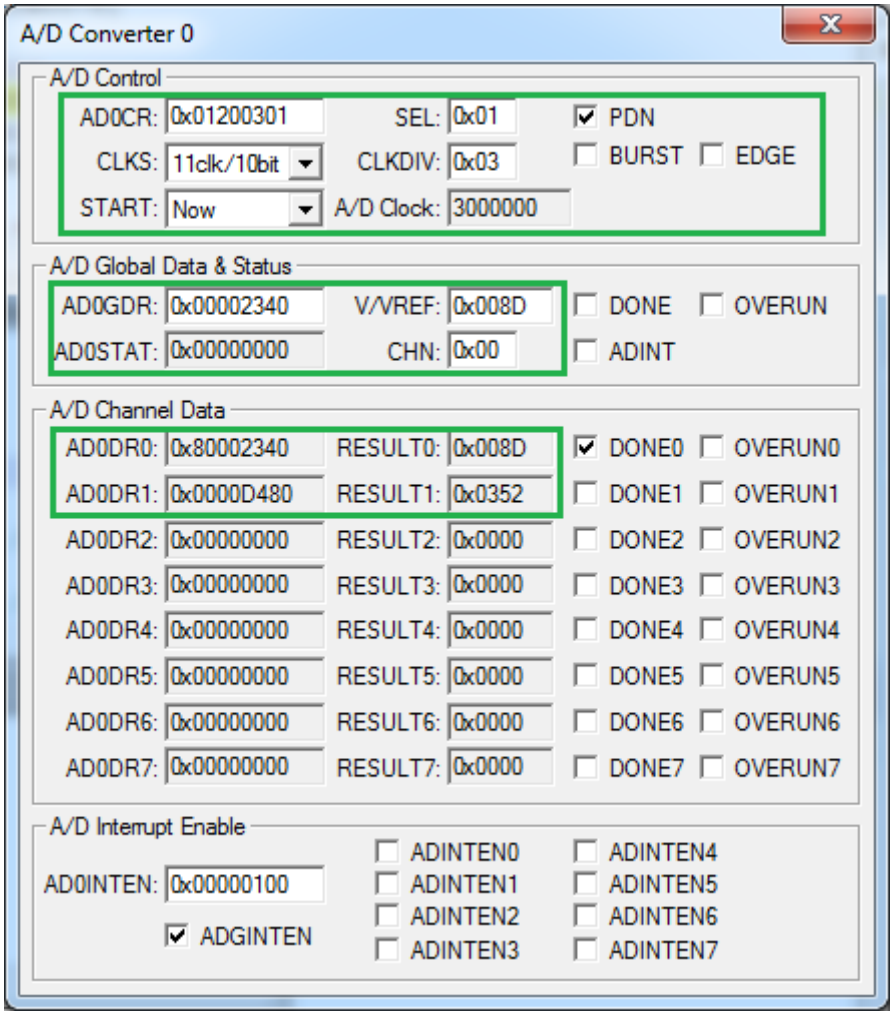
5.2) Yazılımsal ADC Örneği

İlk örneğimizde **0-3V arasında değişen gerilim değerini ADC0.0, ADC0.1 kanalından okuyalım pot değerlerine göre ledleri yakıp söndürelim**. Bu isteğimizi yerine getiren kodu aşağıda görebilirsiniz. Bu kod ADC işlemlerini yazılımsal olarak kontrol etmektedir.

```
#include <LPC23xx.H>

int adc_1,adc_2;

unsigned int adc_read(unsigned char channel)
{
      unsigned int i;
      // ADC bloğu açılıyor
      PCONP|= (1 << 12);
      // PCLK=12MHz, kanal seçimi yapılıyor
      AD0CR = 0x00200300 | ((0x01)<<channel);
      // A/D çevrimi başlatılıyor
      AD0CR|= 0x01000000;

      while(!(AD0GDR& 0x80000000));
      i = AD0GDR; // Çevrim bilgisi okunuyor

      return ((i>>6) & 0x03FF); // 15:6 arasındaki 10 bitlik veri ADC data
}
```

```
int main(void)
{
	PINSEL0=0x00000000;	// İlk 16 pin GPIO olarak ayarlanıyor
	PINSEL1=0x00014000;

	IODIR0=0xFFFFFFFF;

	for(;;)
	{
		adc_1=adc_read(0);
		adc_2=adc_read(1);
		if(adc_1>512 && adc_2>512)
			IOSET0=0xFFFFFFFF;		// Ledleri yak
		if(adc_2<512 && adc_1<512)
			IOCLR0=0xFFFFFFFF;		// Ledleri söndür
	}
}
```

Yukarıdaki kodun çalışan halini ise şekil-5.1'de görebilirsiniz.



Şekil 5.1 – Yazılımsal ADC Uygulaması

Şekil 5.1'de görüleceği gibi yeşille daireye alınmış değerler debug esnasında görülen değerlerdir. Yalnız yukarıdaki örnek yazılımsal olarak kontrol edildiğinden çok da profesyonelce değildir.

5.3) Donanımsal ADC Örneği

Yazılımsal olarak ADC kullanımı çok gerekmedikçe tercih edilmez. Bunun yerine işler yazılımdan donanıma devredilerek, mikrodenetleyicinin yükü hafifletilebilir. Bunun için ADC çevrim kesmesi kullanılabilir.
Bu örneğimizde **ADC0.0 ve ADC1.0'a bağlı iki LM35 ile sıcaklık ölçümü yapacağız.** Bu uygulamaya ait kodları aşağıda görebilirsiniz. Bu kodlar ile ADC kesmesi birlikte kullanılmıştır.

```
#include <LPC213x.H>
#include "lcd.h"

unsigned int ADCresult[2];

void ADC_Isr_1(void) __irq    // ADC0 kesmesi
{
      unsigned int r_1;
      r_1 = AD0DR;
      ADCresult[0] = (r_1>>6) & 0x03FF;
}

void ADC_Isr_2(void) __irq    // ADC1 kesmesi
{
      unsigned int r_2;
      r_2 = AD1DR;
      ADCresult[1] = (r_2>>6) & 0x03FF;
      VICVectAddr = 0; // Kesme adresi tekrar 0 oluyor
}

int main(void)
{
      int adc;

      AD0CR = 0x00207801; // Init ADC (Pclk = 12MHz)
      AD1CR = 0x00207801; // Init ADC (Pclk = 12MHz)

      VICVectAddr0  = (unsigned int) &ADC_Isr_1;
      VICVectAddr1  = (unsigned int) &ADC_Isr_2;
      // Datasheetten ADC0 kesmesi bit değeri 18
      // 18=0b10010, Etkin bit 1 olacak yani 0b110010=0x32
      VICVectCntl0  = 0x32;               // ADC0 kesmesi aktif
      // Datasheetten ADC0 kesmesi bit değeri 21
      // 21=0b10101, Etkin bit 1 olacak yani 0b110101=0x35
      VICVectCntl1  = 0x35;               // ADC1 kesmesi aktif
      VICIntEnable |= 0x00040000;   // 18. bit ADC0 için
      VICIntEnable |= 0x00200000;   // 21. bit ADC1 için

      lcd_init();

      for(;;)
      {
            AD0CR |= 0x00010000; // ADC0 BURST modunda başlatılıyor
            AD1CR |= 0x00010000; // ADC0 BURST modunda başlatılıyor
            lcd_gotoxy(1,1);
            adc=(int)(ADCresult[0]/3.103);
            veri_yolla(adc/100+48);
            veri_yolla('.');
            veri_yolla((adc%100)/10+48);
            veri_yolla((adc%10)+48);
            veri_yolla('V');
```

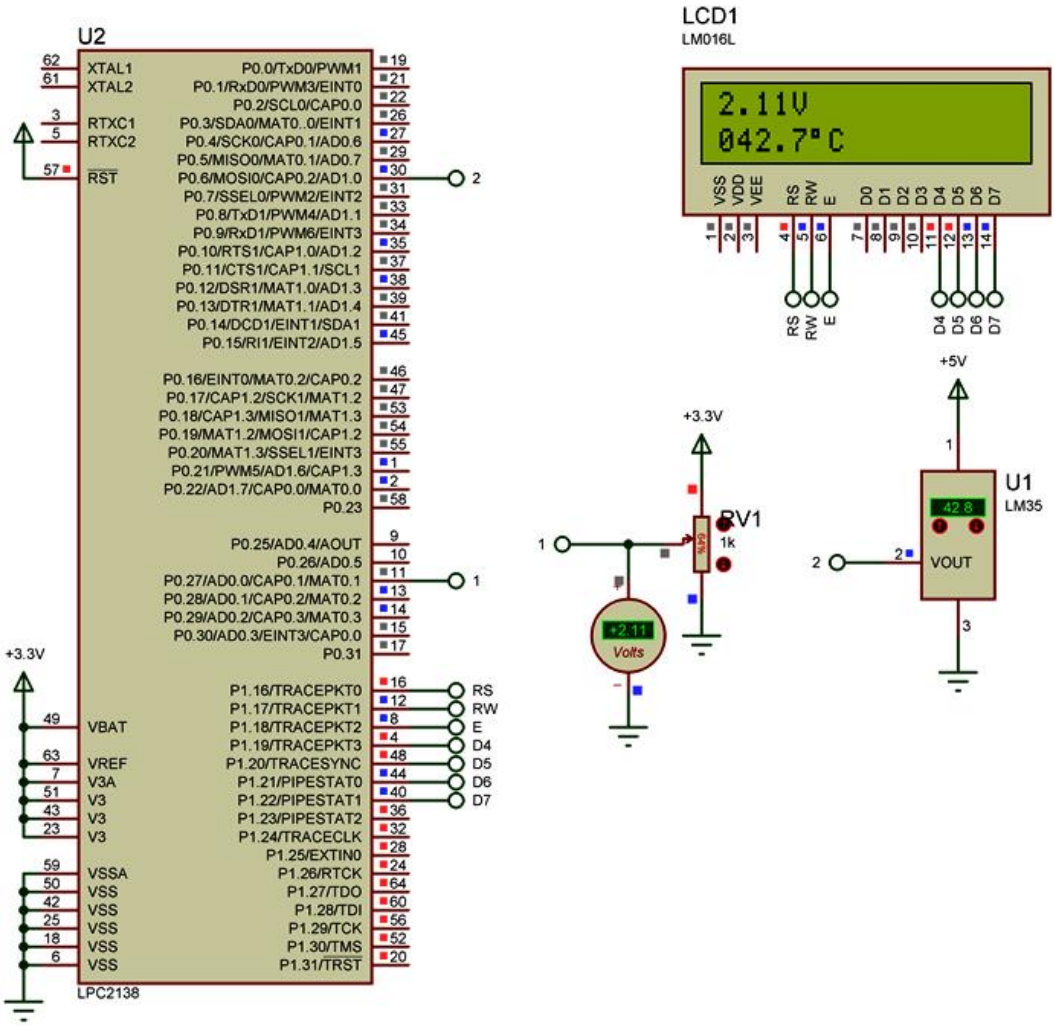
```
            lcd_gotoxy(2,1);
            adc=(int)(ADCresult[1]*32.22);
            veri_yolla(adc/10000+48);
            veri_yolla((adc%10000)/1000+48);
            veri_yolla((adc%1000)/100+48);
            veri_yolla('.');
            veri_yolla((adc%100)/10+48);
            veri_yolla(0xDF);
            veri_yolla('C');
      }
}
```

Yukarıda görebileceğiniz `VICVectCntl0`, `VICVectCntl1` ve `VICIntEnable` atamalarında aşağıda görülebilecek tablo-5.1'den yararlanılmıştır.

Bit	23	22	21	20	19	18	17	16
Symbol	-	-	AD1	BOD	I2C1	AD0	EINT3	EINT2
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SPI1/SSP	SPI0	I2C0	PWM0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Tablo 5.1 – Kesme Adresleri Tablosu

Tablo-5.1'den de görüleceği üzere **`VICVectCntl0`** atamasında aşağıdaki yol izlenmiştir. ADC0'ın bit değeri 18'dir. 18 binary olarak 0b10010 olmaktadır. Bu bizim kesmede kullanacağımız 5 bitlik IRQ slotumuzu dolduracaktır. Kesme izni için 6. bitin 1 olması gerekmektedir. Dolayısı ile **`VICVectCntl0`**'a yükleyeceğimiz değer 0b110010=0x32 olacaktır. Aynı şekilde ADC1'in bit değeri 21 binary olarak 0b10101 olmaktadır. Kesme izin bitiyle bu değer 0b110101=0x35 olur. Böylece kesmelerin ADC0 ve ADC1 kaynaklarından geleceği belirtilir. Bu uygulamaya ait çalışan ekran görüntüsünü şekil-5.2'de görebilirsiniz.



Şekil 5.2 – Donanımsal ADC Uygulaması

Ayrıca ADC0 ya da ADC1'de oluşacak birden fazla kesme için aşağıdaki kod öbeği kullanılabilir. Böylece bir kesme birimiyle, daha fazla kanal okunur. Bu kod NXP'nin sitesinden bire bir alınmıştır.

```
static unsigned short ADCresult[4];

void ADC_Isr_1(void) __irq
{
      unsigned int r,ch;
      r = AD0DR;
      ch = (r >> 24) & 0x07; // Hani kanalın çevrildiği öğreniliyor
      ADCresult[ch] = (r>>6) & 0x03FF;
      VICVectAddr = 0;
}

int main(void)
{
      VICVectAddr0 = (unsigned int) &ADC_Isr;
      VICVectCntl0 = 0x32;
      VICIntEnable |= 0x40000;
      AD0CR  = 0x0020030F;
      AD0CR |= 0x00010000;

      ...
}
```

BÖLÜM 6 –DAC

ARM ile DAC işlemleri oldukça basittir. Kullanacağımız LPC2138'de bir adet DAC bulunmaktadır ve P0.25 bacağından dışarı çıkmaktadır. Bu birimi kullanmak istediğimizde öncelikle PINSEL'den, ilgili pinin DAC fonksiyonunu seçmeli ve aşağıda içeriği verilen DACR registerine ilgili değeri yüklememiz yeterli olacaktır.

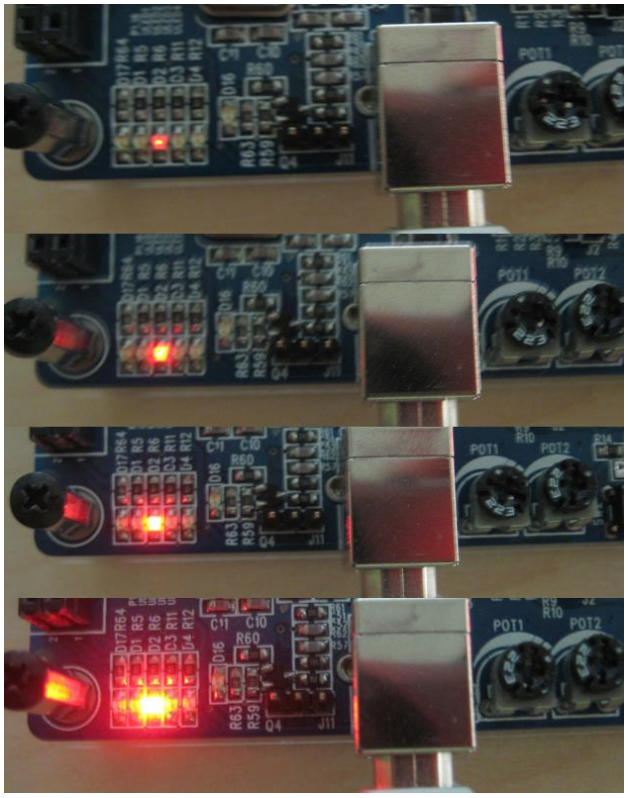
DAC Register: DACR

BIT	ADI	AÇIKLAMA	RESET
5:0	-	Ayrılmış Bit	NA
15:6	VALUE	BIAS ayarından sonra Vref gerilimini 10bitlik çözünürlük bölümüyle dışarı yansıtan değerdir.	NA
16	BIAS	0: DAC maksimum 1us'de yeni değere ulaşır ve maksimum çıkış akımı 700uA'dir. 1: DAC 2.5us'de yeni değerine ulaşır ve maksimum çıkış akımı 350uA'dir.	NA
31:17	-	Ayrılmış Bit	NA

DAC örneğimizde **çıkış geriliminin ADC0.0 kanalından gelen bilgi ile değiştirelim ve bir ledin kısık ya da yüksek yanmasını sağlayalım.** Bu isteğimizi yerine getiren kodu aşağıdan görebilirsiniz.

```
#include <LPC23xx.H>

unsigned int adc_read(unsigned char channel)
{
	unsigned int i;
	// ADC bloğu açılıyor
	PCONP|= (1 << 12);
	// PCLK=12MHz, kanal seçimi yapılıyor
	AD0CR = 0x00200300 | ((0x01)<<channel);
	// A/D çevrimi başlatılıyor
	AD0CR|= 0x01000000;

	while(!(AD0GDR& 0x80000000));
	i = AD0GDR; // Çevrim bilgisi okunuyor

	return ((i>>6) & 0x03FF); // 15:6 arasındaki 10 bitlik veri ADC data
}

int main(void)
{
	int adc_1=0;
	PINSEL0=0x00000000;	// İlk 16 pin GPIO olarak ayarlanıyor
	PINSEL1=0x00214000;

	IODIR0=0xFFFFFFFF;

	for(;;)
	{
		adc_1=adc_read(0);
		DACR=adc_1<<6;
	}
}
```

Uygulamaya ait ekran görüntüsünü şekil-6.1'de görebilirsiniz. ADC girişi olarak POT1 kullanılmıştır.



Şekil 6.1 - DAC Uygulaması

BÖLÜM 7 –UART0/ UART1

ARM serilerinin büyük bir kısmında olduğu gibi bu örnek için kullanacağımız LPC21xx serisinde 2 adet UART birimi bulunmaktadır. Her iki birimde 16byte büyüklüğünde FIFO'ya sahiptirler. Yalnız UART1'in UART0'dan tek farkı standart modemler için sinyal üretebilmesidir. Bu bölümde öncelikle UART0 anlatılacak, sonrasında ise UART1 detaylı şekilde incelenecektir.

7.1) UART0 Birimi

UART0'ın kontrolü için öncelikle bu birimi kontrol eden registerler incelenmelidir. Bu birimi kontrol eden registerlerin detayına ise tablo-7.1'den bakılabilir. Bu bölümde sadece önem arz eden registerlerin içerikleri açıklanacaktır. Diğer registerlerin içeriklerine datasheet'ten bakabilirsiniz.

Table 73: UART0 register map

Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U0RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE000 C000 (DLAB=0)
U0THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE000 C000 (DLAB=0)
U0DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE000 C000 (DLAB=1)
U0DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE000 C004 (DLAB=1)
U0IER	Interrupt Enable Register	Reserved	Reserved	Reserved	Reserved	Reserved	Enable RX Line Status Interrupt	Enable THRE Interrupt	Enable RX Data Available Interrupt	R/W	0x00	0xE000 C004 (DLAB=0)
U0IIR	Interrupt ID Register	FIFOs Enabled		Reserved	Reserved	IIR3	IIR2	IIR1	IIR0	RO	0x01	0xE000 C008
U0FCR	FIFO Control Register	RX Trigger		Reserved	Reserved	Reserved	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE000 C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0x00	0xE000 C00C
U0LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000 C014
U0SCR	Scratch Pad Register	8-bit Data								R/W	0x00	0xE000 C01C
U0TER	Transmit Enable Register	TXEN	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R/W	0x80	0xE000 C030

[1] Reset value relects the data stored in used bits only. It does not include reserved bits content.

Tablo 7.1 – UART0 Register Tablosu

İlgili UART0 pinlerinin GPIO'dan seçiminden sonra her UART biriminin ilk ayarlanmasında olduğu gibi öncelikle saat kaynağımızın değeri belirlenir. Boudrate hesabı için kullanılan registerler **U0DLL** ve **U0DLM**'dir.

Daha sonra sırasıyla, baud rate, data bitinin uzunluğu, parity ve stop bitleri seçilir. UART0 biriminin ayarında bu ayarları yapan register tablo7.1'den de görüleceği üzere **U0LCR** registeridir. Bu registerin içeriği aşağıdaki tablodan görülebilir.

UART0/UART1 Line Control Register: U0LCR/U1LCR

BIT	ADI	DEĞERİ	AÇIKLAMA	RESET
1:0	WORD LENGTH SELECT	00 01 10 11	5-Bit 6-Bit 7-Bit 8-Bit	00
2	STOP BIT SELECT	0 1	1 Stop Bit 2 Stop Bit (Eğer Word Uzunluğu 5 ise 1.5 Stop Bit)	0
3	PARITY ENABLE	0 1	Aktif değil Aktif	0
5:4	PARITY SELECT	00 01 10 11	Odd Parity Event Parity Mark Parity Space Parity	0

6	BREAK CONTROL	0 1	Kesim kontrolünü devre dışı bırak Kesim kontrolü açılır ve TXT Space Parity'e zorlanır	0
7	DIVISOR LATCH ACCESS BIT (DLAB)	0 1	Divisor yani bölücü registerine erişim izni yok. Divisor yani bölücü registerine erişim izni var.	0

UART0 biriminin baudrate hesabı için aşağıdaki formül kullanılır.

$$UART0_{baudrate} = \frac{PCLK}{16x(16xU0DLM + U0DLL)}$$

Yukarıdaki denkleme göre 20MHz PCLK için yüklenmesi gereken U0DLM:U0DLL değerleri tablo-7.2'de görebilirsiniz.

Table 78: Some baud-rates available when using 20 MHz peripheral clock (PCLK=20 MHz)

Desired baud-rate	U0DLM:U0DLL		% error[1]	Desired baud-rate	U0DLM:U0DLL		% error[1]
	hex	dec			hex	dec	
50	0x61A8	25000	0	4800	0x0104	260	0.1603
75	0x411B	16667	0.0020	7200	0x00AE	174	0.2235
110	0x2C64	11364	0.0032	9600	0x0082	130	0.1603
134.5	0x244E	9294	0.0034	19200	0x0041	65	0.1603
150	0x208D	8333	0.0040	38400	0x0021	33	1.3573
300	0x1047	4167	0.0080	56000	0x0021	22	1.4610
600	0x0823	2083	0.0160	57600	0x0016	22	1.3573
1200	0x0412	1042	0.0320	112000	0x000B	11	1.4610
1800	0x02B6	694	0.0640	115200	0x000B	11	1.3573
2000	0x0271	625	0	224000	0x0006	6	6.9940
2400	0x0209	521	0.0320	448000	0x0003	3	6.9940
3600	0x015B	347	0.0640				

[1] Relative error calculated as: actual_baudrate/desired_baudrate-1. Actual baudrate based on Equation 1.

Tablo 7.2 – PCLK 20MHz İçin U0DLM:U0DLL'e Yüklenecek Değerler Tablosu

Baudrate hesaplandıktan sonra bu değerlerin **U0DLM:U0DLL**'e yüklenmesi için **U0LCR**'den **DIVISOR LATCH ACCESS** biti set edilip **U0DLM:U0DLL**'e ilgili değerler yüklenmeli ve sonrasında yine **DIVISOR LATCH ACCESS** biti temizlenmelidir. Bu şekilde yapıldığında ayarlanan baudrate daha sonradan değiştirilemez.
Baudrate değerleri yüklendikten sonra ise **U0FCR** ile tüm Tx, Rx FIFO'ları temizlenmeli ve FIFO aktif edilmelidir. Bunun için **U0FCR**'ye **0x07** değeri aranmalıdır.
UART0'dan bir byte'lık ifade almak için öncelikle **U0LSR**'den **U0RBR** alım registerinin boş olup olmadığına bakılır. Eğer boş ise gelen ifade **U0RBR**'den çekilerek okunur.
UART0'dan bir bytle'lık gönderim için ise yine öncelikle **U0LSR**'den **U0THR** gönderim registerinin boş olup olmadığına bakılır. Eğer boş ise gönderilecek ifade **U0THR**'ye yüklenerek gönderilir. İlgili registerlerin içeriklerine Tablo-7.1'den ya da datasheet'ten bakılabilir.
Tüm bu işlemleri yapan kütüphanemizi aşağıda görebilirsiniz.

UART.H

```
#define UART_PIN  PINSEL0           // UART hangi PINSEL bloğuna bağlı

#define PCLK                  12000000 // PCLK değeri
#define UART0_BAUD_RATE       19200    // Baudrate
#define UART0_WORD_LENGTH     3     // 0: 5bit, 1: 6bit, 2: 7bit, 3:8bit
#define UART0_STOP_BIT        0     // 0: Stop bit=1, 1: Stop bit=2/1.5
#define UART0_PARITY          0     // 0: Parity yok, 1: Parity var
#define UART0_PARITY_SEL      0     // 0: Odd, 1: Event, 2: Mark, 3: Space

extern void uart_init(void);
extern void putch(unsigned char d);
extern char getch(void);
extern void uart0_string(const char *st);
```

UART.C

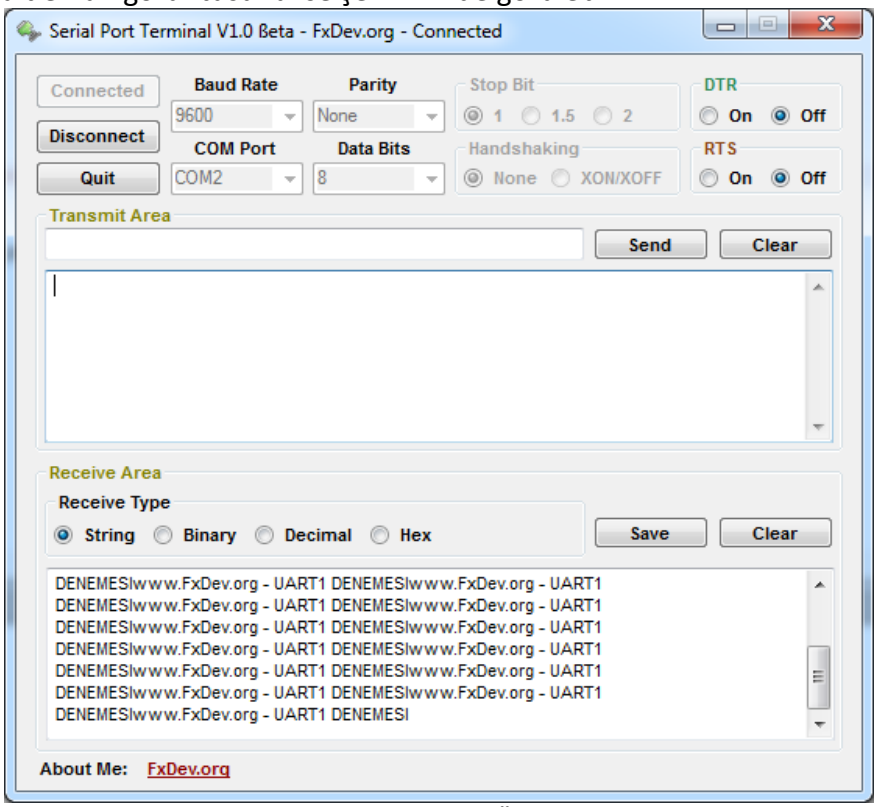
```
#include <LPC213x.H>
#include "UART.h"

void uart_init(void)
{
      int div;
      UART_PIN=0x00000050;
     // GPIO'dan P0.0 ve P0.1 UART için ayarlanıyor

      U0LCR=0x80|(UART0_PARITY_SEL<<5)|(UART0_PARITY<<3)|(UART0_STOP_BIT<<
2)|UART0_WORD_LENGTH;

      div = ( PCLK / 16 ) / UART0_BAUD_RATE ;
      // Baudrate hesabı yapılıyor
      U0DLM = div / 256;
      U0DLL = div % 256;
      // Bölme değerinin değişmemesi için DIVISOR'a erişim izni yok
      U0LCR =
(UART0_PARITY_SEL<<5)|(UART0_PARITY<<3)|(UART0_STOP_BIT<<2)|UART0_WORD_LEN
GTH;
      U0FCR = 0x07;     // FIFO'yu aç ve TX FIFO ve RX FIFO'yu resetle
}

void putch(unsigned char c)
{
      while(!(U0LSR&0x20));         // Transmit gönderim FIFO'su boş mu?
      U0THR=c;
}

char getch(void)
{
      while(!(U0LSR&0x01));         // Receive alım FIFO'su boş mu?
      return U0RBR;
}

void uart0_string(const char *st)
{
      while(*st)
            putch(*st++);
}
```

Seri Port Uygulaması

İlk seri port uygulamamızda **ismimizi seri porttan gönderip, echo uygulaması** yapalım. Bu isteğimizi yerine getiren kod aşağıda görülebilir.

```
#include <LPC213x.h>
#include "UART.h"

int main(void)
{
	uart_init();

	uart0_string("www.Firatdeveci.com\n\rFIRAT DEVECi");

	for(;;)
		putch(getch());
}
```

Uygulamaya ait ekran görüntüsünü ise şekil-7.1'de görülebilir.



Şekil 7.1 – UART0 Örneği

Şekil-7.1'den de görüleceği üzere ARM mikrodenetleyicilerde UART ile çalışmak oldukça basittir. **Burada en önemli faktör PCLK değerinin doğru olarak belirtilmesidir.**
Eğer Keil uVision kullanıyorsanız startup.s dosyasından ya da ilgili PLL registerlerinden PCLK düzgün bir şekilde ayarlanmalıdır.

7.2) UART1 Birimi

Daha önceden de söylediğimiz gibi UART1'in UART0'dan tek farkı modem desteği vermesidir. Diğer tüm özellikleri, registerlerin bazıları hariç, UART0 ile aynıdır. UART1'in verdiği modem

pinleri desteği ile yazılımsal olarak donanımları kontrol etmek mümkündür. UART1'in parity veya stop bit gibi ayarlamalarını **UART1 Line Control Registerinden** yani U1LCR'den görebilirsiniz. Bu registerin içeriği UART0 için ayrılmış registerle aynıdır. UART1 için kullanılan registerleri tablo-7.3'den görebilirsiniz.

Table 88: UART1 register map

Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U1RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE001 0000 (DLAB=0)
U1THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE001 0000 (DLAB=0)
U1DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE001 0000 (DLAB=1)
U1DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE001 0004 (DLAB=1)
U1IER	Interrupt Enable Register	Reserved	Reserved	Reserved	Reserved	Enable Modem Status interrupt[2]	Enable RX Line Status Interrupt	Enable THRE Interrupt	Enable RX Data Available Interrupt	R/W	0x00	0xE001 0004 (DLAB=0)
U1IIR	Interrupt ID Register	FIFOs Enabled		Reserved	Reserved	IIR3	IIR2	IIR1	IIR0	RO	0x01	0xE001 0008
U1FCR	FIFO Control Register	RX Trigger		Reserved	Reserved	Reserved	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE001 0008
U1LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Parity Select	Parity Enable	Number of Stop Bits	Word Length Select		R/W	0x00	0xE001 000C
U1MCR[2]	Modem Control Register	Reserved	Reserved	Reserved	Loop Back	Reserved	Reserved	RTS	DTR	R/W	0x00	0xE001 0010
U1LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE001 0014
U1MSR[2]	Modem Status Register	DCD	RI	DSR	CTS	Delta DCD	Trailing Edge RI	Delta DSR	Delta CTS	RO	0x00	0xE001 0018
U1SCR	Scratch Pad Register	8-bit Data								R/W	0x00	0xE001 001C
U1TER	Transmit Enable Register	TXEN	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	R/W	0x80	0xE001 0030

[1] Reset value relects the data stored in used bits only. It does not include reserved bits content.

[2] Modem specific features are available in LPC2134/6/8 only.

Tablo 7.3 – UART1 Register Tablosu

Tablo 7.3'den de görüleceği üzere UART0 registerlerinden farklı olarak UART1 registerlerinde modem ayarları için gerekli U1MCR ve U1MSR kaydedicileri mevcuttur. Modem ile ilgili bir anlatım burada yapılmayacaktır.

7.2) UART0/1 Kesmesi

Tablo5.1'den de görüleceği üzere UART1 ve UART0'ın Vektörel Kesme Kontrolcüsüne (VIC) bağlanan tek kesme kanalı vardır fakat tablo-7.3'ten UnIER registerinden görüleceği üzere RX alım, THRE gönderim ve RX Line kesmesidir. UnLSR registerinden de hangi olayın hangi kesmeye yol açtığı öğrenilebilir.

UART kesmesini kavramak için alım kesmesi örneği yapalım.

UART Alım Kesmesi Örneği

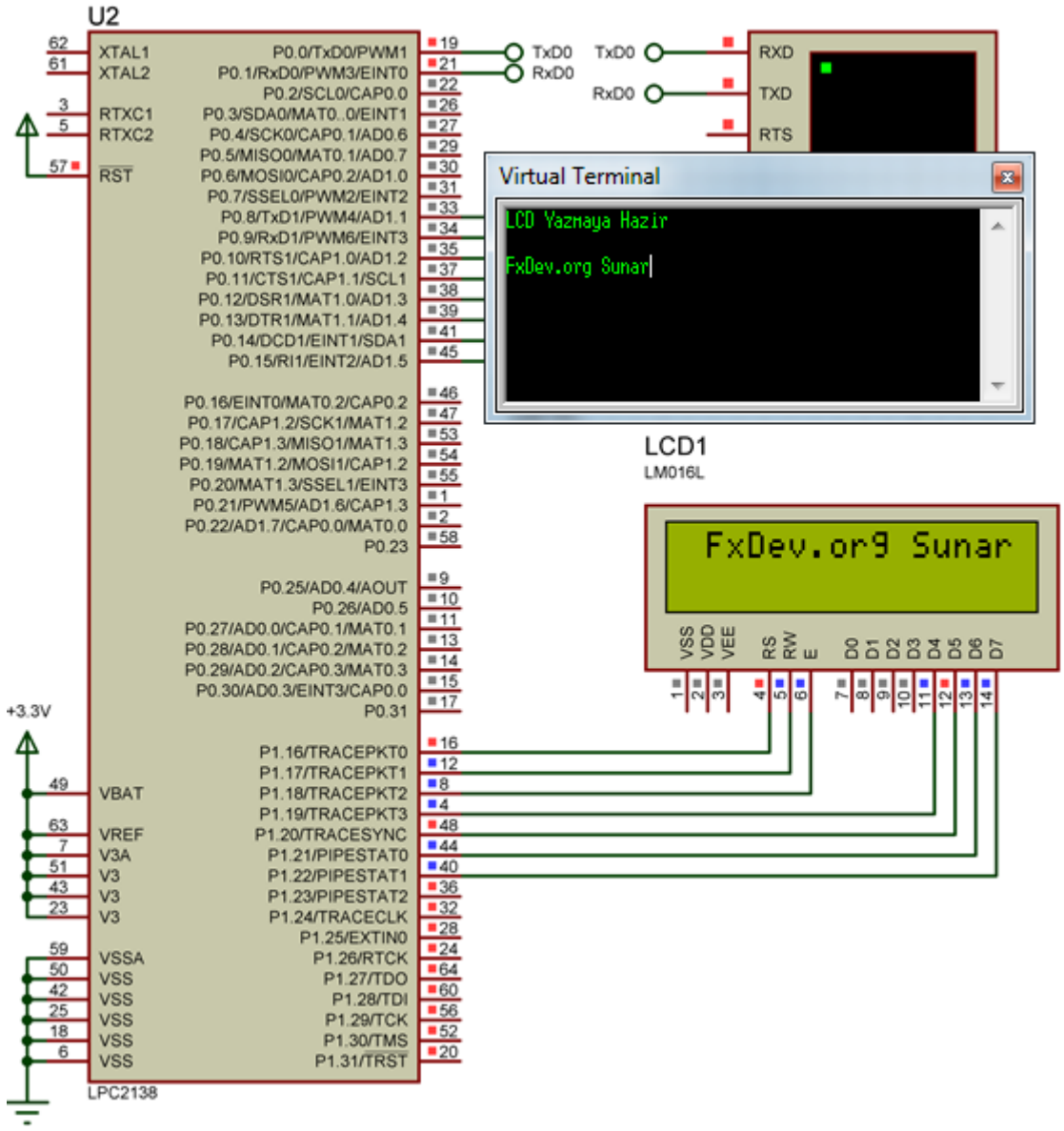
Receive kesmesinin iki türü vardır. Bunlardan ilki RDA kesmesidir ve alım gerçekleştiğinde oluşur. Yalnız diğer miktodenetleyicilerden farklı olarak RDA kesmesi tek byte alımında değil,

1, 4, 8 veya 14 byte veri alındıktan sonra oluşur. Bu sayının belirlenmesi için UnFCR registerinin 6 ve 7. bitleri üzerinde ayarlama yapılır. İkinci kesme türü ise CTI'dır ve alım FIFO'sunun içinde veri kaldığında ve bu veri 3-4 byte okuma süresinde okunmadığında oluşur.

Bu örneğimizde UART0, 1. byte'ı aldığında bir kesme oluşturmasını sağlayalım ve alınan veriyi LCD'de yazdıralım. Kesmenin oluştuğunu ise geriye "Kesme Olustu" bilgisi ile geri döndürelim. Bunu yapmak için öncelikle U0FCR registerinin 7. ve 6. Bitlerine sırasıyla "00" yükleyelim (UnFCR ile ilgili detaylı bilgi için datasheet'e göz atabilirsiniz) ve UART0 ile ilgili tüm ayarları yapalım. Daha sonra VIC kesme vektörüne UART0 kesmesini yükleyelim. Tablo-5.1'den görüleceği üzere UART0'ın kemse vektör biti 6 dır. 6 binary olarak 0b00110'a denk gelmektedir. VIC vektör kontrol değeri 0b100110 olur. VIC UART kesmesi aktif etmek için ise 0x00000020 yüklenmelidir. Tüm bunları yapan kodumuz aşağıda görülebilir.

```
#include <LPC213x.h>
#include "lcd.h"

unsigned char data,flag=0;
void UART_Isr(void) __irq
{
      unsigned char kesme_bilgisi;
      kesme_bilgisi=U0IIR;
     // Data FIFO'dan çekiliyor, dolayısı ile FIFO temizleniyor
      data=U0RBR;
      flag=1;                 // Bilgi alındı bayrağı
      VICVectAddr = 0;        // Kesme adresi tekrar 0 oluyor
}

void putc(unsigned char c)
{
      while(!(U0LSR&0x20));   // Transmit gönderim FIFO'su boş mu?
      U0THR=c;
}

void uart0_str(const char *st)
{
      while(*st)
            putch(*st++);
}

int main(void)
{
      int div;
      PINSEL0=0x00000005;     // GPIO'dan P0.0 UART0 için ayarlanıyor

      lcd_init();

      U0LCR=0x83;                         // 8bit, parity yok, stop bit 1
      div = ( 12000000 / 16 ) / 9600 ; // Baudrate hesabı yapılıyor
      U0DLM = div / 256;
      U0DLL = div % 256;
      U0LCR = 0x03;                       // 8bit, parity yok, stop bit 1
      U0IER = 0x01;
      U0FCR = 0x07;     // FIFO'yu aç ve TX FIFO ve RX FIFO'yu resetle
                        // Trigger seviyesi 0
      VICVectAddr0  = (unsigned int) &UART_Isr;
      // Datasheetten UART0 kesme bit değeri 6
      // 18=0b00110, Etkin bit 1 olacak yani 0b100110=0x26
      VICVectCntl0  = 0x26;               // UART0 kesmesi için
```

```
        VICIntEnable |= 0x00000040;   // UART0 kesmesi açılıyor

        uart0_str("LCD Yazmaya Hazir\n\r\n\r");

        for(;;)
        {
                if(flag)              // Bilgi geldiyse bunu LCD'ye bas
                {
                        veri_yolla(data);
                        flag=0;
                }
        }
}
```

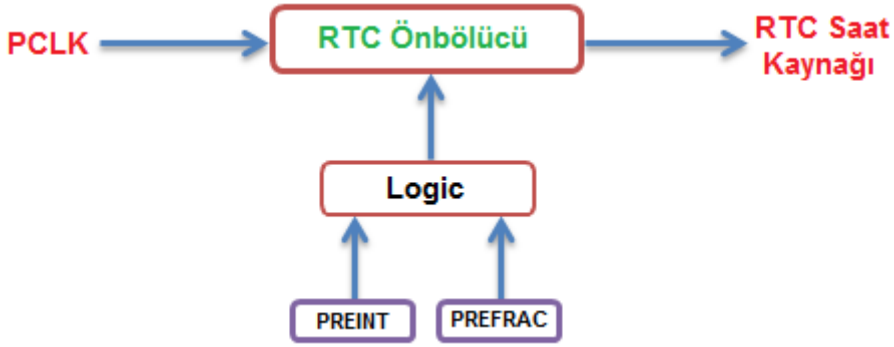


Şekil 7.2 – UART0 Kesme Örneği

BÖLÜM 8 –RTC

Özellikle profesyonel ve yarı profesyonel uygulamalarda tarh tutmak için çok sık kullanılan RTC işlevi, mikrodenetleyicilerinin büyük bir çoğunluğunda bulunmaktadır.

Tüm diğer RTC'lerde olduğu gibi, ARM içerisinde bulunan RTC de 32.768kHz'lik bir frekansla sürülmelidir. Bunun için ARM'ın ilgili bacaklarına bu değerde bir kristal bağlayabileceğiniz gibi PCLK değerini bölüp kullanabilirsiniz.
Ayrıca RTC'yi çalıştırmak için V_{BAT} bacağına 3V civarı bir gerilim uygulamalı ya da bir pil kullanılmalıdır. Eğer işlemcide RTC kullanılmayacaksa bu bacak Vss'ye çekilebilir.
RTC için frekans bölme registerleri oldukça gelişmiştir. PCLK bölme işlemleri ise **PREINT** ve **PREFRAC** registerleri ile gerçekleştirilir. Şekil-8.1'de ilgili işlemin blok şeması görülebilir.



Şekil 8.1 – RTC Frekans Bölücü Bloğu

PREINT ve **PREFRAC** registerlerinin değerleri aşağıdaki bağıntı ile hesaplanabilir.

$$PREINT=(int)(PCLK/32768)-1$$
$$PREFRAC=PCLK-((PREINT+1)x32768)$$

RTC'yi kontrol eden register ise CCR (Clock Control Register)'dir. Bu registerin içeriği aşağıdaki tablodan görülebilir.

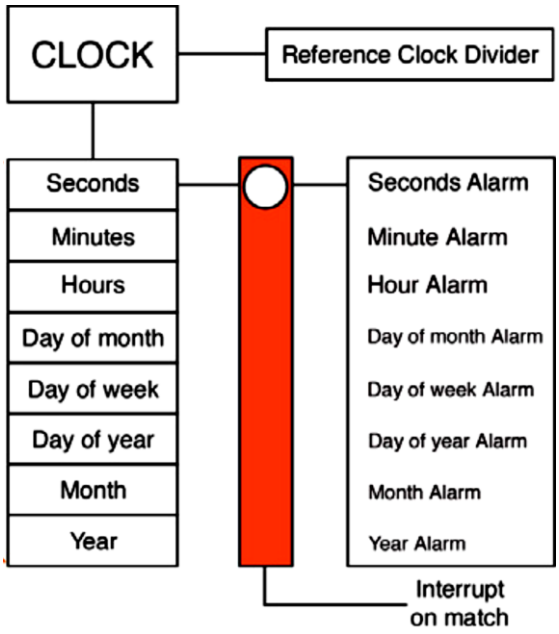
Clock Control Register: CCR

BIT	ADI	AÇIKLAMA	RESET
0	CLKEN	Saat'i açan bittir.	NA
1	CTCRST	Clock Tick Counter'ı sıfırlayan bittir.	NA
3:2	CTTEST	Testleri açar. Normal koşullarda bu bitler sıfırdır	NA
4	CLKSRC	Eğer bu bit sıfır ise frekans kaynağı PCLK, bir ise dış RTCX1 ve RTCX2 pinlerine bağlı kristaldir.	NA
7-5	-	Ayrılmış Bit	NA

RTC'den tarih-saat bilgisi almak ya da okumak için ise **SEC, MIN, HOUR, DOM, DOW, DOY, MONTH ve YEAR** registerleri ile sağlanır. Bu registerlere istenilen zamanda yazım ve okuma işlemi yapılabilir. Burada ilgili registerlerin hangi bilgiyi tuttuğunu aşağıda görebilirsiniz.

SEC : Saniye bilgisi (0-59)
MIN : Dakika bilgisi (0-59)
HOUR : Saat bilgisi (0-23)
DOM : Gün bilgisi (1-31, 1-30, 1-28, 1-29)
DOW : Haftanın günü bilgisi, pazar gününde 0 olacak (0-6)
DOY : Yılın günü bilgisi (1-365, 1-366)
MONTH : Ay bilgisi (1-12)
YEAR : Yıl bilgisi (0-4095)

Ayrıca birçok uygulamada kullanılabilecek alarm özelliği de ARM içerisinde bulunan RTC biriminde mevcuttur. Alarmı kurmak için **ALSEC, ALMIN, ALHOUR, ALDOM, ALDOW, ALDOY, ALMONTH ve ALYEAR** registerlerinden faydalanılır.
Her alarm registeri o anki zaman registerleriyle eşitlendiğinde bir kesme oluşur. Bu kesmenin oluşmasını şekil-8.2'de daha iyi görebilirsiniz.



Şekil 8.2 – RTC Kesme Bloğu

Tablo-5.1'den görüleceği üzere RTC'nin bir adet kesme kaynağı vardır (**13. Bit**). Dolayısı ile kesme oluştuğunda bunun saniye mi yoksa yıl alarmı mı olduğu bilinememektedir. Bu yüzden RTC'de hangi alarmın oluştuğunun kontrolü için **ILR** registerine bakılmalıdır. Bu registerin içeriği aşağıdaki tabloda görülebilir.

Interrupt Location Register: ILR

BIT	ADI	AÇIKLAMA	RESET
0	RTCCIF	RTC kesme bloğu sayı artımı kesmesi üretti. 1 yazılmasıyla bu kesme yazmacı temizlenir.	NA
1	RTCALF	Alarm registerleri kesme üretti. 1 yazılmasıyla bu kesme yazmacı temizlenir.	NA
7:2	-	Ayrılmış Bit	NA

RTC kesmelerini aktif etmek için ise **CIIR** ve **AMR** registerleri kullanılır. Bu registerlerin içeriklerini aşağıdan görebilirsiniz.

Clock Increment Interrupt Register: CIIR

BIT	ADI	AÇIKLAMA	RESET
0	IMSEC	Saniyenin artış kesmesi geldiğinde 1 olur.	NA
1	IMMIN	Dakikanın artış kesmesi geldiğinde 1 olur.	NA
2	IMHOUR	Saatin artış kesmesi geldiğinde 1 olur.	NA
3	IMDOM	Ay gününün artış kesmesi geldiğinde 1 olur.	NA
4	IMDOW	Hafta gününün artış kesmesi geldiğinde 1 olur.	NA

5	IMDOY	Yıl gününün artış kesmesi geldiğinde 1 olur.	NA		
6	IMMON	Ayın artış kesmesi geldiğinde 1 olur.	NA		
7	IMYEAR	Yılın artış kesmesi geldiğinde 1 olur.	NA		

Alarm Mask Register: AMR

BIT	ADI	AÇIKLAMA	RESET
0	AMRSEC	1 ise saniye alarm oluşturmayacak.	NA
1	AMRMIN	1 ise dakika alarm oluşturmayacak.	NA
2	AMRHOUR	1 ise saat alarm oluşturmayacak.	NA
3	AMRDOM	1 ise ay günü alarm oluşturmayacak.	NA
4	AMRDOW	1 ise hafta günü alarm oluşturmayacak.	NA
5	AMRDOY	1 ise yıl günü alarm oluşturmayacak.	NA
6	AMRMON	1 ise ay alarm oluşturmayacak.	NA
7	AMRYEAR	1 ise yıl alarm oluşturmayacak.	NA

Yukarıdaki tabloda görüleceği üzere AMR tam ters mantıkla çalışmaktadır. Örneğin dakika alarmı kurulmak isteniyorsa, dakika alarmıyla ilgili bit 0 yapılmalıdır.
RTC ile ilgili örneğimizde **RTC'yi yeni bir yıldan 15 saniye öncesine kuralım ve yeni yıla girişte kesme oluşturarak LCD ekrana 'Yeni Yiliniz/Kutlu Olsun' yazdıralım.** Bu isteğimizi yerine getiren kodlar aşağıda görülebilir.

```
#include <LPC213x.h>
#include "lcd.h"

#define PCLK                    12000000
#define RTC_CLK_SRC             1

typedef struct {
    char Sec;     // Saniye değeri        - [0,59]
    char Min;     // Dakika değeri        - [0,59]
    char Hour;    // Saat değeri          - [0,23]
    char Day;     // Saat değeri          - [1,31]
    char Mon;     // Ay değeri            - [1,12]
    int  Year;    // Yıl değeri           - [0,4095]
    char Dow;     // Haftanın günü değeri - [0,6]
} ARM_RTC;

const char
DAYS[7][4]={{"PAZR"},{"PZRT"},{"SALI"},{"ÇARŞ"},{"PERŞ"},{"CUMA"},{"CMRT"}
};
char yeni_yil_flag=0;

void RTC_isr(void) __irq
{
      if(ILR&0x01)      // Alarm kesmesi mi üretti
      {
            ILR=0x01;   // Kesme yazmacı temizleniyor
            yeni_yil_flag=1;
      }
      VICVectAddr=0;
}

void RTC_init(void)
{
```

```
	CCR=0x02;		// RTC değerleri sıfırlanıyor

	if(RTC_CLK_SRC)
	{
	CCR=0x11;
	}
	else
	{
		PREINT=(int)(PCLK/32768)-1;
		PREFRAC=PCLK-((PREINT+1)*32768);
		CCR=0x01;
	}
}

void set_RTC(char RTC_SEC,char RTC_MIN, char RTC_HOUR, char RTC_DOM, char
RTC_MONTH, int RTC_YEAR, char RTC_DOW)
{
	DOW	= RTC_DOW;
	YEAR	= RTC_YEAR;
	MONTH	= RTC_MONTH;
	DOM	= RTC_DOM;
	HOUR	= RTC_HOUR;
	MIN	= RTC_MIN;
	SEC	= RTC_SEC;
}

void RTC_read(char *RTC_SEC,char *RTC_MIN, char *RTC_HOUR, char *RTC_DOM,
char *RTC_MONTH, int *RTC_YEAR, char *RTC_DOW)
{
	*RTC_DOW	= DOW;
	*RTC_YEAR	= YEAR;
	*RTC_MONTH	= MONTH;
	*RTC_DOM	= DOM;
	*RTC_HOUR	= HOUR;
	*RTC_MIN	= MIN;
	*RTC_SEC	= SEC;
}

int main(void)
{
	char i;
	ARM_RTC RTC;
	lcd_init();
	AMR=0x7F;		// Yil kesmesi aktif
	CIIR=0x80;		// Yil artış kesmesi aktif
	RTC_init();
	set_RTC(50,59,23,31,12,2010,6);

	VICVectAddr0=(unsigned)RTC_isr;	// Kesme vektör adresi
	VICVectCntl0=0x0000002D;		// RTC kesmesi
	VICIntEnable=0x00002000;		// RTC kesmesi aktif

	for(;;)
	{

	RTC_read(&RTC.Sec,&RTC.Min,&RTC.Hour,&RTC.Day,&RTC.Mon,&RTC.Year,&RT
C.Dow);
		lcd_gotoxy(1,1);
		veri_yolla(RTC.Hour/10+48);
		veri_yolla(RTC.Hour%10+48);
		veri_yolla(':');
```

```
            veri_yolla(RTC.Min/10+48);
            veri_yolla(RTC.Min%10+48);
            veri_yolla(':');
            veri_yolla(RTC.Sec/10+48);
            veri_yolla(RTC.Sec%10+48);
            lcd_gotoxy(2,1);
            veri_yolla(RTC.Day/10+48);
            veri_yolla(RTC.Day%10+48);
            veri_yolla('/');
            veri_yolla(RTC.Mon/10+48);
            veri_yolla(RTC.Mon%10+48);
            veri_yolla('/');
            veri_yolla(RTC.Year/1000+48);
            veri_yolla((RTC.Year%1000)/100+48);
            veri_yolla((RTC.Year%100)/10+48);
            veri_yolla(RTC.Year%10+48);
            lcd_gotoxy(2,12);
            for(i=0;i<4;i++)
                  veri_yolla(DAYS[RTC.Dow][i]);

            if(yeni_yil_flag)
            {
                  yeni_yil_flag=0;
                  lcd_clear();
                  lcd_yaz("Yeni Yiliniz");
                  lcd_gotoxy(2,1);
                  lcd_yaz("Kutlu Olsun..");
                  for(;;)
                  {

      RTC_read(&RTC.Sec,&RTC.Min,&RTC.Hour,&RTC.Day,&RTC.Mon,&RTC.Year,&RT
C.Dow);
                        if(RTC.Sec==5)
                        {
                              lcd_clear();
                              break;
                        }
                  }
            }
      }
}
```

Yukarıdaki kodun çalışan halini ise şekil-8.3'te görebilirsiniz.

Şekil 8.3 – RTC Örneği

BÖLÜM 9 –SPI

Şekil9.1'de görülebilecek, Motorola tarafından ismi konan SPI birimi sensör okumadan, LCD'yi sürmeye kadar birçok donanımla birlikte kullanılabilir. LPC2138 işlemcisinin içinde **SPI0** ve **SPI1** olmak üzere iki adet SPI birimi bulunmaktadır. Her iki birimde birbiriyle neredeyse aynı olmasına karşın **SPI1** Motorola'nın ürettiği cihazlardaki SPI'ya uyumluluk, 4 bitten 16bit'e kadar veri uzunluğu seçme gibi çeşitli ek özelliklere sahiptir.

Şekil 9.1 – SPI İletişim Diyagramı

9.1) SPI0 Birimi

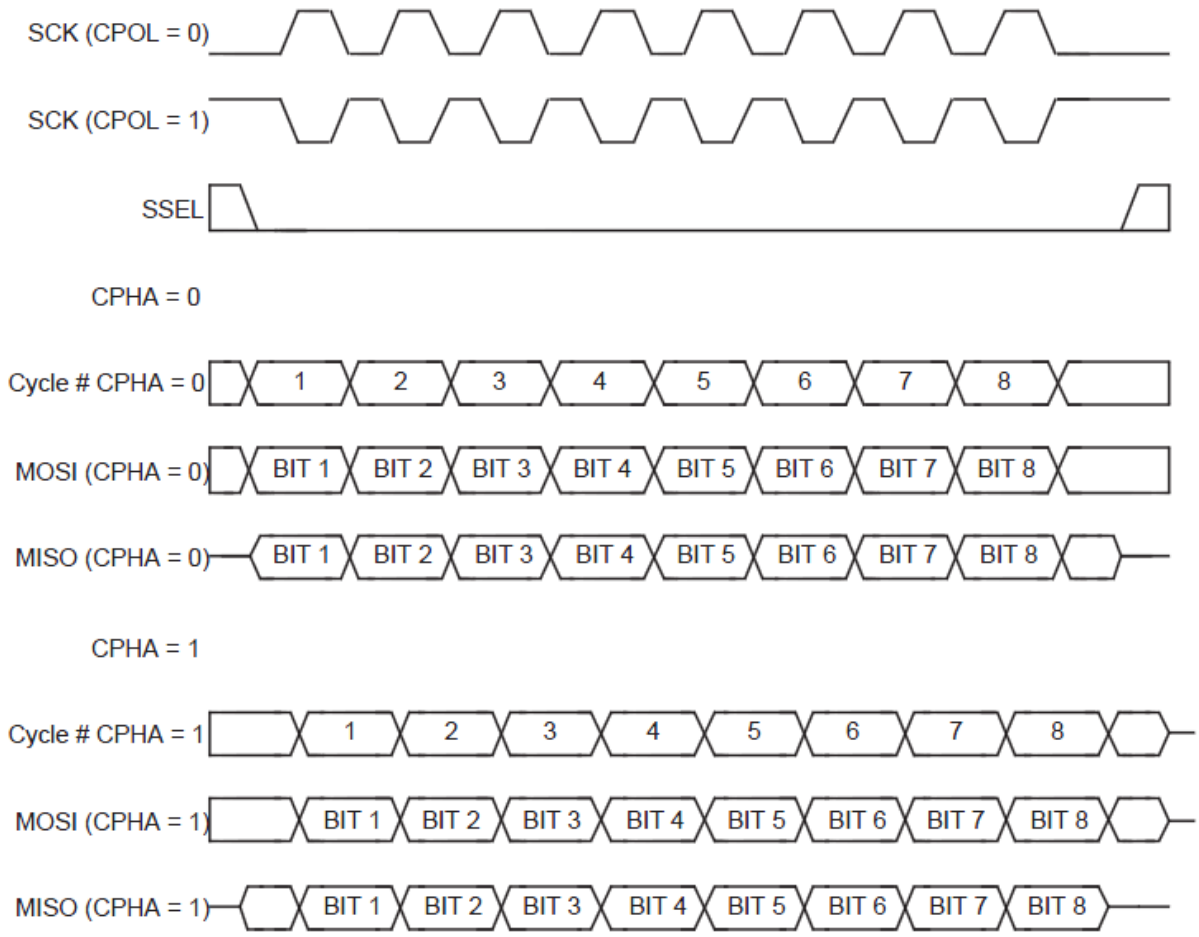
ARM içerisinde bulunan **SPI0** birimi 8bitten 16bit'e kadar veri uzunluğu seçimi, senkron, seri ve full dublex iletişimi destekleyen **SPI0** birimini kullanmak için öncelikle aşağıda tablosu görülebilecek **SPI0 Control Register**'i ayalanmalıdır.

SPI0 Control Register Register: S0SPCR

BIT	ADI	DEĞERİ	AÇIKLAMA	RESET
1:0	-	-	Ayrılmış Bit	NA
2	BitEnable	0 1	Her seferinde 8 bitlik veri gönderilecek ya da alınacak Her seferinde BITS'teki değere göre veri gönderilecek ya da alınacak	0
3	CPHA	0	Data ilk SCK saat kenarında örneklenir. Transfer SSEL sinyalinin pasif veya aktif olmasına göre	0

		1	başlar. Data ikinci SCK saat kenarında örneklenir. Transfer SSEL aktif olduğunda transfer ilk kenarında başlar ikinci kenarda biter.	
4	CPOL	0 1	SCK yüksek kenarda aktif SCK düşük kenarda aktif	0
5	MSTR	0 1	SPI slave(köle) modda SPI master(efendi) modda	0
6	LSBF	0 1	Veri önce yüksek, sonra düşük bitleri gönderir Veri önce düşük, sonra yüksek bitleri gönderir	0
7	SPIE	0 1	SPI kesmesi deaktif SPI kesmesi aktif	0
11:8	BITS	1000 1001 1010 1011 1100 1101 1110 1111 0000	8 bit transfer edilir 9 bit transfer edilir 10 bit transfer edilir 11 bit transfer edilir 12 bit transfer edilir 13 bit transfer edilir 14 bit transfer edilir 15 bit transfer edilir 16 bit transfer edilir	0000
12:15	-	-	Ayrılmış Bit	NA

Yukarıda anlatılan **CPOL** ve **CPHA** seçimleri sonucunda ortaya çıkan dalga şekilleri şekil-9.2'den görülebilir.



Şekil 9.2 – CPOL ve CPHA Değerliklerine Göre Oluşan İletişim Sinyalleri

Şekil-9.2'deki diyagramın açıklamasını aşağıdaki tabloda görebilirsiniz.

CPOL	CPHA	İlk Data Sürücüsü	Diğer Data Sürücüsü	Data Örnekleme
0	0	Önce SCK'nın ilk yükselen kenarında	SCK düşen kenarında	SCK yükselen kenarında
0	1	SCK'nın ilk yükselen kenarında	SCK yükselen kenarında	SCK düşen kenarında
1	0	Önce SCK'nın ilk düşen kenarında	SCK yükselen kenarında	SCK düşen kenarında
1	1	SCK'nın ilk düşen kenarında	SCK düşen kenarında	SCK yükselen kenarında

9.2) Master Operasyon

SPI iletişimde master olmak için aşağıdaki adımlar sırasıyla izlenmelidir:

- SPI pinleri PINSEL'den ayarlanır.
- Kullanılacak donanıma göre SPI saat hızı belirlenir.
- Donanımın gerektirdiği şekilde SPI için gerekli ayarlar yapılır.
- SPI data registerine(S0SPDR) bilgi yazılır. Bu yazımla birlikte bilgi otomatik gönderilir.

- Gönderimin tamamlanması için SPIF bitinin 1 olması beklenir. SPIF gönderilen bilginin son biti gönderilirken 1 olur.
- SPI status registeri okunur ve gerekliyse gelen bilgi okunur.
- Daha fazla bilgi gönderilmek isteniyorsa adım üçten itibaren yukarıdakiler tekrar yapılır.

Burada önemli olan her okuma ve yazım yapıldığında SPIF bitinin temizlenmesi gerektiğidir. Eğer bu bit temizlenmezse okumada ve alımda sorunlar oluşur.

9.3) Slave Operasyon

SPI iletişimde slave olmak için aşağıdaki adımlar sırasıyla izlenmelidir:

- SPI pinleri PINSEL'den ayarlanır.
- Donanımın gerektirdiği şekilde SPI için gerekli ayarlar yapılır.
- Eğer data gönderilecekse, data gönderimi bittiyse, bu işlem gerçekleştirilir.
- Gönderimin tamamlanması için SPIF bitinin 1 olması beklenir. SPIF gönderilen bilginin son biti gönderilirken 1 olur.
- SPI status registeri okunur ve gerekliyse gelen bilgi okunur.
- Eğer daha fazla bilgi göndermek gerekliyse adım ikiden itibaren yukarıdakiler tekrar yapılır.

Burada yine önemli olan her okuma ve yazım yapıldığında SPIF bitinin temizlenmesi gerektiğidir. Eğer bu bit temizlenmezse okumada ve alımda sorunlar oluşur.

9.3) Frekans Ayarlama

SPI için veri hızını ayarlamak için 8 bit genişliğinde **S0SPCCR** registeri kullanılır. Bu registerin 0. biti her zaman 0 olmak zorundadır. Yani register içeriği **her zaman çift ve 8 den büyük** bir sayı olmak zorundadır. SPI frekansının değeri **PCLK/S0SPCCR** olur.

9.4) SPI Örneği

SPI örneğimize **TC72 sıcaklık sensöründen bilgileri okuyup, bu bilgileri LCD'de gösterelim**. İsteğimizi yerine getiren kodu aşağıda görebilirsiniz.

```
#include <LPC213x.h>
#include "lcd.h"
#include "SPI.h"

int deneme=0;
#define SPI0_SEL  7           // SPI için enable biti hangi pin olacak
// SPI için enable biti CLR ve SET blokları olacak
#define SPI0_SET  IO0SET
#define SPI0_CLR  IO0CLR

void tc72_init(void)
{
      spi_init();
      SPI0_SET=1<<SPI0_SEL;   // Eğer chip yüksek seviyede aktifse
      spi_write(0x80);
      spi_write(0x04);
      SPI0_CLR=1<<SPI0_SEL;   // Eğer chip düşük seviyede aktifse
}
```

```
void tc72(unsigned char *msb, unsigned char *lsb, unsigned char *control)
{
	SPI0_SET=1<<SPI0_SEL;	// Eğer chip yüksek seviyede aktifse
	spi_write(0x02);
	*msb=spi_read();
	*lsb=spi_read();
	*control=spi_read();
	SPI0_CLR=1<<SPI0_SEL;	// Eğer chip düşük seviyede aktifse
}

int main(void)
{
	unsigned char onlar, ondalar, control;
	lcd_init();
	SPI0_DIR=1<<SPI0_SEL;	// Selection pini belirleniyor


	lcd_init();		// LCD ilk ayarları yapılıyor
	tc72_init();		// SPI ilk ayarları yapılıyor

	lcd_yaz("Sicaklik:");

	for(;;)
	{
		lcd_gotoxy(1,11);
// Sıcaklık ve kontrol kaydedicisi degerleri alınıp LCD'ye yazdırılıyor
		tc72(&onlar, &ondalar, &control);
		veri_yolla(onlar/100+48);
		veri_yolla((onlar%100)/10+48);
		veri_yolla(onlar%10+48);
		veri_yolla(0xDF);
		veri_yolla('C');
		lcd_gotoxy(2,1);
		lcd_yaz("Control : ");
		veri_yolla(control/16+48);
		veri_yolla(control%16+48);
		veri_yolla('H');
	}
}
```

Yukarıdaki kodlarda görülen **SPI.h** kodları ise aşağıdaki gibidir.

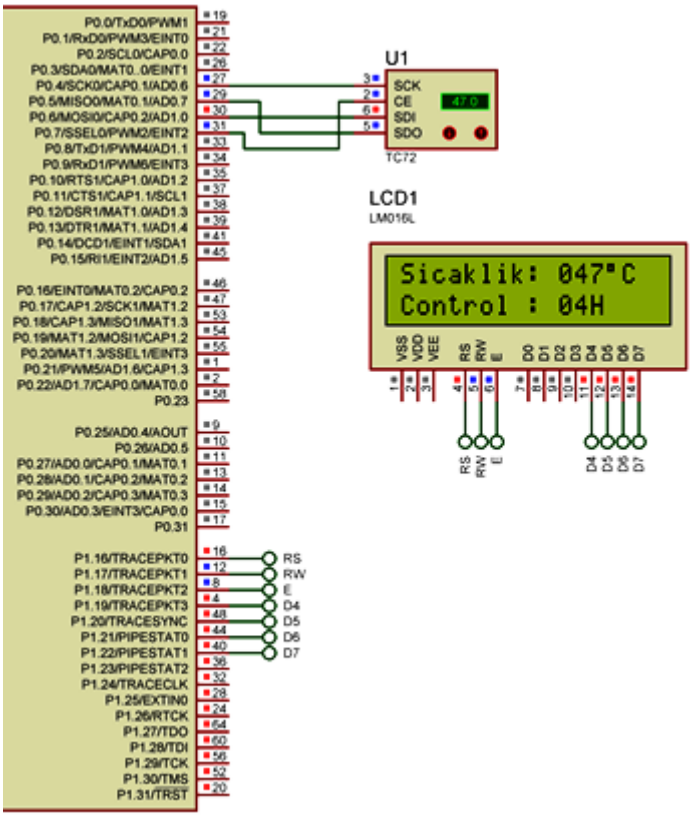
```
#define PCLK		12000000	// PCLK değeri
#define SPI0_EN		1		// SPI0 açılacak mı
#define SPI0_CLK	50000		// SPI0 hızı ne olacak
#define SPI0_PIN	PINSEL0		// SPI'nın bulunduğu PINSEL bloğu
#define SPI0_DIR	IO0DIR		// SPI için enable biti nerede olacak
#define SPI0_SEL	7		// SPI için enable biti hangi pin olacak
#define SPI0_SET	IO0SET		// SPI için enable biti CLR ve SET blokları
olacak
#define SPI0_CLR	IO0CLR
#define SPI0_SS		1	// Kullanılacak aletin enable'ı ne zaman aktif
#define BitEnable0	0	// 8 bitten başka uzunluk kullanılacaksa bu bit 1
yapılmalı
#define SPI0_CPHA	1	// 0 ise data ilk saat darbesinde, 1 ise ikinci
saat darbesinde örneklenir
#define SPI0_CPOL	0	// IDLE seviyesi
#define SPI0_MSTR	1	// Master için bu bit bir olacak
#define SPI0_LSBF	0	// 0 ise önce yüksek bitler, 1 ise önce düşük
```

```
bitler gönderilir
#define SPI0_BITS 8     // 8-15 bit için 8-15 yazılacak, 16bit için 0

extern void spi_init(void);
extern void spi_write(int veri);
extern int spi_read(void);
```

Son olarak **SPI.c** kodları ise aşağıdaki gibidir.

```
#include <LPC213x.H>
#include "SPI.h"

void spi_init(void)
{
      unsigned char div;
      if(SPI0_EN)
      {
            S0SPCR=0x00;            // Tüm SPI0 ayarları sıfırlanıyor
            SPI0_PIN &= 0xFFFF00FF; // SPI0 pinleri ayarlanıyor
            SPI0_PIN |= 0x00001500;

            SPI0_DIR=1<<SPI0_SEL;   // Selection pini belirleniyor
            //Ayarlar yapılırken chip aktif olmasın istiyoruz
            if(SPI0_SS==0)
                  SPI0_SET=1<<SPI0_SEL;
           // Eğer chip yüksek seviyede aktifse
            else
                  SPI0_CLR=1<<SPI0_SEL;
           // Eğer chip düşük seviyede aktifse

            // SPI0 hızı belirleniyor
            div=(PCLK/SPI0_CLK)+8;
            if((div%2)!=0)    // Eğer frekans değeri 3,5 gibiyse
                  div+=1;     // 4 ve 6 gibi çift rakama tamamlanıyor
            S0SPCCR=div;      // Frekans atanıyor


      S0SPCR=(BitEnable0<<2)|(SPI0_CPHA<<3)|(SPI0_CPOL<<4)|(SPI0_MSTR<<5)|(
SPI0_LSBF<<6)|(SPI0_BITS<<8);
      }
}

void spi_write(int veri)
{
      S0SPSR=0;
      S0SPDR=veri;
      while ( !(S0SPSR & (0x80)) );
}

int spi_read(void)
{
      int temp=0;

      S0SPDR = 0xFF;
      while ( !(S0SPSR & (0x80)) );
      temp=S0SPDR;

      return temp;
}
```

Uygulamaya ait ekran görüntüsünü şekil-9.3'te görebilirsiniz.

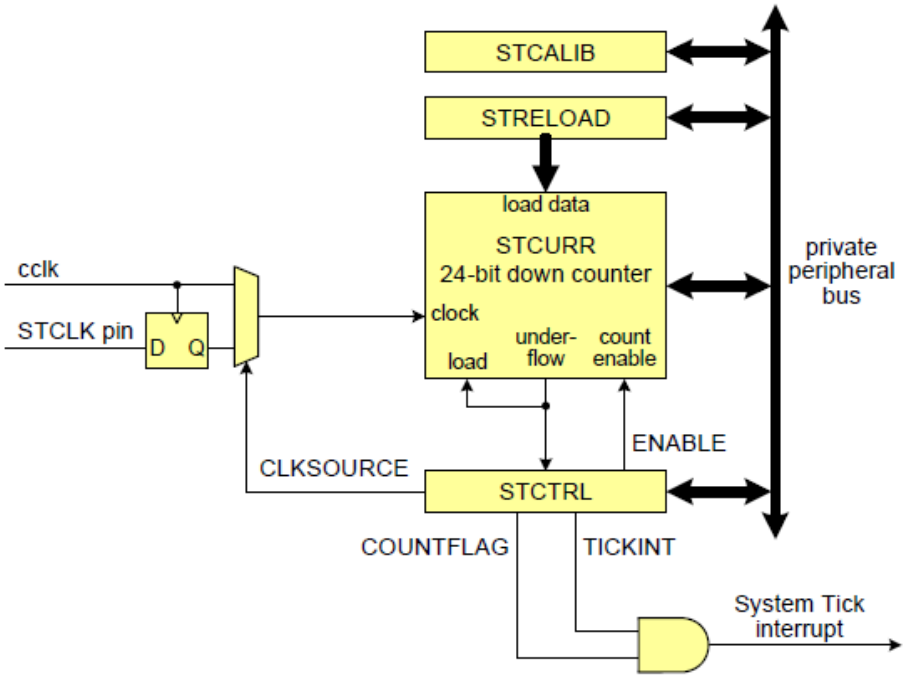


Şekil 9.3 – TC72 SPI Örneği

BÖLÜM 10 –I²C

BÖLÜM 11 – SYSTEM TICK TIMER

System Timer Tick Cortex M3'lerde bir işletim sistemi ya da başka bir işlem için 10ms'de bir kesme üretmek için tasarlanmış birimdir. Cortex M3'ler için ayrıca standart bir zamanlayıcıdır. System Timer Tick 24 bitlik bir sayıcıdır ve yüklenen değerden sıfıra doğru saymaya başlar ve değer sıfıra ulaştığında bir kesme elde edilir. Ayrıca yeni değer **STRELOAD** registerinden tekrar sayıcıya yüklenir. System Tick Timer CCLK'yı saat kaynağı olarak kullanabildiği gibi ayrıca şekil 11.1'den görüleceği üzere P3.26'daki STCLK da saat kaynağı olarak kullanabilir.



Şekil 11.1 – System Tick Timer Bloğu

System Tick Timer'ı kontrol etmek için dört adet registerden faydalanılır.

System Tick Timer Control and Status Register: STCTRL

BIT	ADI	AÇIKLAMA	RESET
0	ENABLE	Bit 1 olursa system tick timer açık, sıfırsa kapalı	0
1	TICKINT	System Tick Timer kesme izin biti. Eğer bu bit set edilirse sayım sıfıra ulaştığında kesme oluşur.	0
2	CLKSOURCE	Saat kaynağı seçim biti. Eğer 1 ise CCLK, 0 ise STCLK	1
15:3	-	Ayrılmış Bit	NA
16	COUNTFLAG	System Tick sayıcı bayrağı. Sayım değeri sıfır olduğunda bu bit 1 olur ve bu bit okunmasıyla birlikte sıfırlanır.	0
31:17	-	Ayrılmış Bit	NA

System Timer Reload Value Register: STRELOAD

BIT	ADI	AÇIKLAMA	RESET
23:0	RELOAD	Sayıcı sıfırlandığında tekrar yüklenecek değerdir	0
31:24	-	Ayrılmış Bit	NA

System Timer Current Value Register: STRCURR

BIT	ADI	AÇIKLAMA	RESET
23:0	CURRENT	O anki sayım değerini saklar. STCTRL'deki COUNTFLAG'e herhangi bir bit yazılması durumunda sıfırlanır	0
31:24	-	Ayrılmış Bit	NA

System Timer Calibration Value Register: STCALIB

BIT	ADI	AÇIKLAMA	RESET
23:0	TENMS	100MHz saat hızında 10ms kesme oluşturacak değerin	0xF423F

		tekrar yüklendiği alan	
29:24	-	Ayrılmış Bit	NA
30	SKEW	TENMS'e yüklenen değerin 10ms kesme oluşturup oluşturmayacağını gösteren bittir. 1 olduğunda 10ms de bir kesme oluşmayacak demektir.	0
31	NOREF	Harici bir referans saatinin mevcut olup olmadığını gösterir. Eğer bu değer 1 ise harici bir referans saati mevcut değildir.	0

System Tick Timer'ı kullanabilmek için öncelikle aşağıdaki formülden RELOAD değeri bulunmalıdır.

***RELOAD**=(CLK/100)-1*

Formüldeki CLK değeri o an için System Clock Timer'ın kullandığı frekans değeri olacaktır. RELOAD değeri de yüklendikten sonra kesmeler aktif edilmeli, priority yani kesme önceliği belirlenmeli ve System Tick Timer çalıştırılmalıdır. Böylelikle istenilen uzunlukta, gerçek değerlikli gecikmeler sağlanabilecektir.

System Tick Timer Örneği

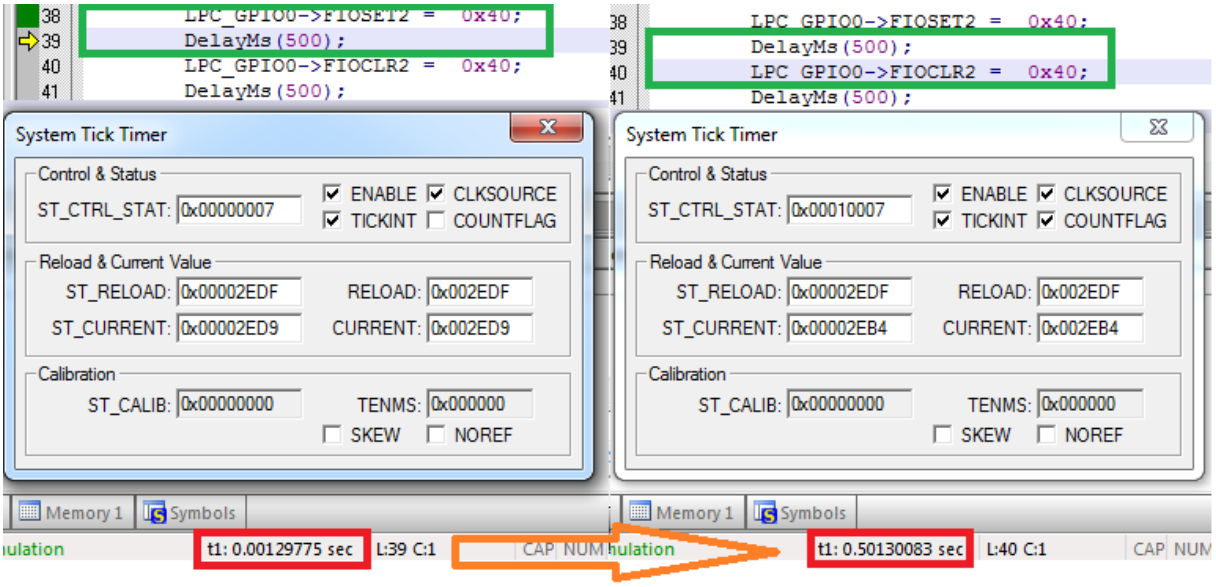
Bu örnekte **System Tick Timer kesmesini 10ms'de bir olmasını sağlayalım ve LPC1768'in P0.22'sine bağlı olan ledi 500ms'de bir yakıp söndürelim.**

Bunu gerçekleştiren kod öbeğini aşağıda görebilirsiniz.

```
#include <LPC17xx.h>

volatile unsigned long STT_Value;

void SysTick_Handler(void)
{
	STT_Value++;
}

void DelayMs(unsigned int t)
{
	unsigned long temp;

	temp = STT_Value;
	while ((STT_Value - temp) < t);
}

int main (void)
{
	SystemInit ();                          // İlk ayarlar yapılıyor

	LPC_PINCON->PINSEL0=0;                  // Pinler GPIO olarak ayarlanıyr
	LPC_PINCON->PINSEL1=0;

	LPC_GPIO0->FIODIR = 0xFFFFFFFF;         // GPIO'ların hepsi çıkış oluyor
	// 12MHz CCLK için 1ms kesme için yükleme değeri
	// (12MHz/1000)-1 olur.
	SysTick->LOAD=(12000000/1000)-1;        // 1ms de bir kesme oluşacak
	// System Tick timer kesmesi önceliği belirleniyor
	NVIC_SetPriority (SysTick_IRQn, 15);
```

```
        // System Tick Timer Value değeri sıfırlanıyor
        SysTick->VAL =(0x00);
        // System Tick Timer açılıyor, kesme aktif, kaynak CCLK
        SysTick->CTRL = 7;

        for (;;)
        {
              LPC_GPIO0->FIOSET2 =  0x40;
              DelayMs(500);
              LPC_GPIO0->FIOCLR2 =  0x40;
              DelayMs(500);
        }
}
```

Yukarıdaki kodun çalışma anı ise şekil 11.2'de görülebilir.



Şekil 11.2 – System Tick Timer Örneği

Şekil-11.2'den görüleceği üzere System Tick Timer ile çalışmak yazılımsal olarak yapılan gecikmelerden hem daha sağlıklı hem de daha fonksiyoneldir.