

1. Giriş

Atmel genel ismi ile anılan mikro denetleyici ailesinden ATmega 16 ile başlayan çalışmalar ATmega 128 ile sürdürülmüş; ATmega 16 ya parça parça yazılan programlar, ATmega 128 ile daha bütünlüklü olarak ele alınmıştır. Söz konusu belgede çalışma önce Atmel ailesine ait mikro denetleyici hakkında genel özelliklerin anlatımı ile başlamış, temel bilgiler verildikten sonra, ayrı ayrı belirtilen amaçlara uygun algoritmalar C dili ile yazılarak söz konusu denetleyiciye aktarılmış, ilgili devre şeması ve modelleme sonuçları her bölümün sonunda verilmiştir.

ATmega ailesini C dili kullanarak programlamak için 'AVRStudio4' ile 'WinAVR' derleyicilerinden yararlanılmıştır. Yazılan programlar ise Proteus Isis 7.2 devre modelleme programı ile denenmiş, sonuçlar belgede gösterilmiştir.

2. Mikro denetleyici Nedir?

Mikroişlemci ve mikro denetleyici kavramları:

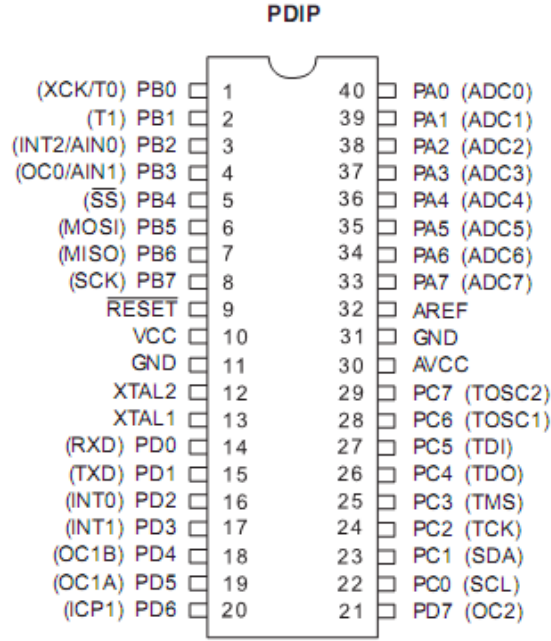
Mikroişlemci; kendisine gönderilen komutları işleyen çok küçük boyutlardaki bir sayısal devredir. Bu yapı bir bilgisayardaki MİB (Merkezi İşlemci Birimi-CPU) olarak düşünülebilir.

MİB tek başına bir bilgisayarı oluşturamaz ve üstelik tek başına kullanılamaz. MİB e benzeyen mikroişlemciler de onlar gibi tek başlarına kullanılamazlar. En azından bir bellek ve giriş/çıkış arabirimlerine ihtiyaç duyarlar. Bu ve bunun gibi bir çok arabirim bir araya getirilerek tek başlarına uygulamalarda kullanılabilen sayısal devrelere *mikro denetleyici* denir. Mikro denetleyiciler günümüzde kullanılan hemen hemen bütün elektronik cihazların içinde bulunmaktadır.

Mikro denetleyiciler, mikroişlemcilerin yanı sıra RAM, ROM, EPROM, EEPROM, Flash Bellek gibi bellek yapıları, G/Ç (Giriş/Çıkış) arabirimleri, analog-sayısal çeviriciler, zamanlama ve sayma birimleri, PWM çıkış modülleri, seri haberleşme portları, barındırırlar. Bu birimlerin düzenli ve bir arada çalışmasını sağlamak amacıyla ise "*kütük*" (*register*) denilen özel hafıza bölgelerinden yararlanır. Yine bu birimlerde kullanılan ve kütükler üzerinde yer alan, çeşitli işlemler için izinleri ayarlayan ya da işlemlerin hangi aşamada olduklarını gösterebilen, yerine göre yazılım ya da donanım tarafından değiştirilebilen "*bayrak*" (*flag*) yapıları mevcuttur.

Bu belgede staj boyunca ATmega16 ile başlayan süreç anlatılacak, ATmega 16 ve ATmega128 mikro denetleyicileri ile ele alınan konular anlatıldıktan sonra konuya uygun kod yazılacak ve ardından Proteus Isis programının ekran çıktıları ile oluşturulan programın sonuçları verilecektir.

3. ATMEL ATMEGA16 Pin Diyagramları



Şekil 1

ATmega16'nın üretici bilgi sayfasından (datasheet) alınan yukarıdaki şekild pin diyagramını görülebilir.

ATmega16 Genel Özellikler ve Bacak (Pin) Diyagramı

Aşağıda ATmega16'ya ait bir kısım özellikler görülmektedir.

- Yüksek performanslı düşük güçlü AVR® 8-bit Mikro denetleyici
- Gelişmiş RISC mimarisi
 - 32 x 8 Genel Amaçlı Çalışan Yazmaçlar (Kütükler-Registers)
 - Tamamıyla Statik (sabit) İşleyiş
 - 16 MHz de 16 MIPS'e Kadar Çıkabilme
 - Çip üzerinde 2 çevrimli çarpım
- Yüksek Dayanıklı Geçici Olmayan (Non-volatile) Bellek Bölümleri
 - 16K Byte Flash Program Belleği
 - 512 Byte EEPROM
 - 1K Byte Dahili SRAM
 - Yaz/Sil Sayısı: 10,000 Flash/100,000 EEPROM
 - Veri (Data) Saklama: 20 yıl 85°C'de/100 yıl 25°C' de(1)
- JTAG (IEEE std. 1149.1 Compliant) Arayüzü
 - Programlanabilir Flash, EEPROM, Sigortalar ve JTAG arayüzünde Kilit Bitleri
- Arayüz Özellikleri
 - 2 Adet 8-bit Zamanlayıcı/Sayıcı (Timer/Counters) ile Ayrık 'Prescalers' ve Karşılaştırma Modları
 - Bir adet 16-bit Zamanlayıcı/Sayıcı (Timer/Counter with Ayrık 'Prescaler' ve Karşılaştırma Modu ve 'Capture' Modu
 - Ayrık osilatör ile gerçek zamanlı sayaç
 - 4 PWM Kanalı
 - 8 Kanallı, 10-bit ADC

8 'Single-ended' Kanal

- Byte-yönelimli 2 kablolu seri arayüz
- Programlanabilir Seri USART
- Master/Slave SPI Seri Arayüz
- Programlanabilir 'Watchdog Timer' ile Ayrık On-chip Osilatörü
- On-chip Analog Karşılaştırıcı
- Özel Mikro denetleyici Özellikleri
- 'Power-on Reset' ve Programlanabilir 'Brown-out Detection'
- Dahili Ayarlı RC Osilatör
- Harici ve Dahili Kesme Kaynakları
- 6 Çeşit Uyku Modu : 'Idle', 'ADC Gürültü Azaltma (Noise Reduction)', 'Power-save', 'Power-down', 'Standby' ve 'Extended Standby'
- I/O ve Paketler(modüller)
- 32 Programlanabilir Giriş/Çıkış (I/O) Dizisi
- 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Çalışma Voltajı
- 2.7 - 5.5V ATmega16L için
- 4.5 - 5.5V ATmega16 için
- Hız Seviyeleri
- 0 - 8 MHz ATmega16L için
- 0 - 16 MHz ATmega16 için
- Güç Tüketimi @ 1 MHz, 3V, ve 25°C ATmega16L için
- Aktif: 1.1 mA
- Çalışmazken(Idle Mode): 0.35 mA
- Kapatıldığında (Power-down Mode): < 1 µA

Özelliklerden anlaşılacağı üzere mikro denetleyici oldukça fazla modül bulundurmaktadır. Bu modüller mikro denetleyicinin çok daha etkin bir biçimde kullanılabilmesinde son derece önemlidir. 20 iş günü süreli stajda bu modüllerden bir kısmı kullanılmıştır.

Şekil 1' de görülen pin diyagramında her bir bacağına ait çeşitli ifadeler yer almaktadır. Bu ifadeler aşağıda açıklanmıştır:

VCC - Dijital sinyal beslemesi. Diğer bir deyişle mikro denetleyiciyi besleme kaynağı.

GND - Toprak

PORT A (PA0...PA7) - A portları 8 bitlidir ve genel kullanımın yanı sıra ADC sinyallerinin de giriş çıkış arabirimidir. Her çıkış dahili olarak pull-up yapılmıştır. Yani bu pinleri giriş olarak ayarladığımızda ekstra bir pull up direnci bağlamamıza gerek yoktur.

PORT B (PB0...PB7) - B Portları da 8 bitlidir. Kendiliğinden Pull-up'lı haldedir. B portları da birçok özel fonksiyona ev sahipliği yapar.(SPI, Analog karşılaştırıcı, Timer1, Timer0 ve USART)

PORT C (PC0...PC7) - C portları da 8 bitlidir ve dahili olarak pull-up edilmiştir. C portları JTAG modülü bulundurmaktadır. JTAG aktive edilirse

PC3, PC4 ve PC5 pinleri, reset durumu olsa dahi aktive edilmiş olunur. Ayrıca zamanlayıcı osilatör girişleri ve de seri iltişim pinlerinden bazıları da burada bulunmaktadır.

PORT D (PD0...PD7) - D portları da 8 bitlidir ve dahili olarak pull-up edilmiştir. D portunda compare, capture ve bazı kesme modülleri de bulunmaktadır. Ayrıca USART giriş çıkış pinleri de buradadır.

!RESET - Terslenmiş yeniden başlatma girişi. RESET yazısının üzerindeki çizgi bu pinin terslenmiş olduğunu gösterir. Yani yeniden başlatma işlemini gerçekleştirmek için bacağı 0 sinyali(GND) verilmelidir.

XTAL1 - Osilatör girişi

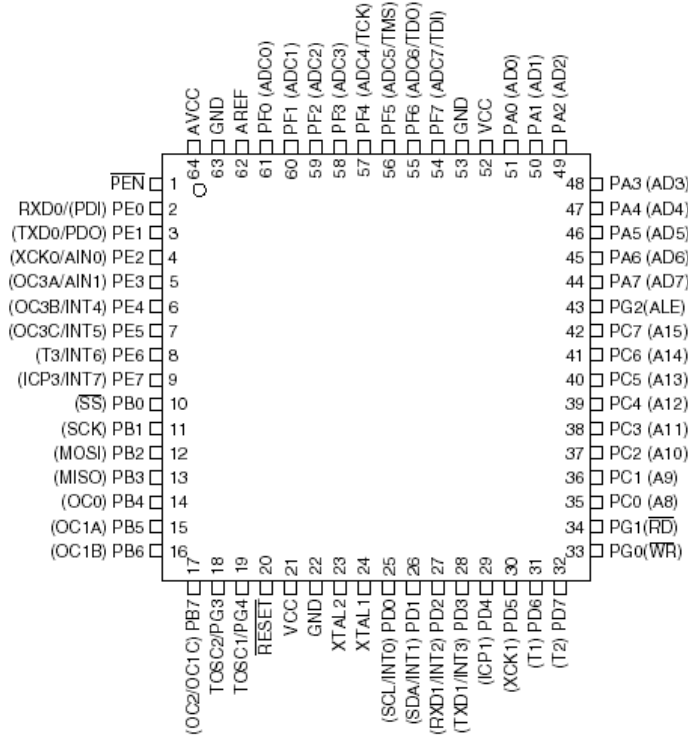
XTAL2 - Osilatör çıkışı

AVCC - PORTA için veya A/D çevirici için besleme girişi. ADC kullanılmıyorsa Vcc'ye bağlanabilir.

AREF - Bu pin de analog dijital çevirici için referans girişidir.

ATMEGA128

ATmega128, ATmega16 gibi Atmel ailesine mensup bir mikro denetleyicidir. 8 bitlik bir mikro denetleyici olan ATmega 128'e ait pin diyagramı aşağıdadır.



Şekil 2

Şekil 2'den anlaşılacağı üzere ATmega 128'de bazı modüllerden daha çok sayıda bulunmakta ve söz konusu stajda yazılacak olan algoritmalar için oldukça fazla imkan sunmaktadır.

Mikro Denetleyicide C Dili Kullanımı Üzerine Hatırlatmalar C Dilinde Bazı Veri Çeşitleri

Veri Çeşidi	Açıklama
bit	Bir bitlik veri taşır. 0 ya da 1 olabilir
char	8 bitlik veri taşır. -127 ve +127 arasında değerler alabilir
unsigned char	8 bitlik veri taşır. 0 ve +255 arasında değerler alabilir
int	16 bitlik veri taşır. -32767 ve +32767 arasında değerler alabilir
unsigned int	16 bitlik veri taşır. 0 ve 65535 arasında değerler alabilir
long	32 bitlik veri taşır. -2147483647 ve +2147483647 arasında değerler alabilir
unsigned long	32 bitlik veri taşır. 0 ve +4294967295 arasında değerler alabilir.
float	24 veya 32 bitlik veri taşır
double	24 veya 32 bitlik veri taşır

Değişkenler mikro denetleyicide RAM'de tutulur. Ancak buradaki kaynakları etkin kullanmak gerektiğinden eğer değişken program boyunca bir değer alıp program içerisinde değişmeyecekse tanımlamanın başına **"const"** ekleyerek bu değişkeni EEPROM ya da Flash bellekte tutmak yerinde olur.

Fonksiyon her çağırıldığında değerinin değişmesini istenmeyen değişkenler için yine tanımın başına **"static"** eklenir.

Eğer reset işlemi esnasında bir değişkenin değerinin değişmesi istenmiyor ise tanımın başına **"persistent"** yazılmalıdır.

Bir değişken sadece belli bir kod tarafından değil de bir kesme alt programı ya da donanım tarafından değiştirilebiliyorsa değişkenin başına **"volatile"** anahtar kelimesini eklenir. Özellikle kütük adresleri tanımlanırken bu anahtar kelimenin kullanılması önemlidir.

C'de değişkenleri 2'lik, 10'luk ve 16'lık sayı tabanlarında tanımlayabilirsiniz. Örnek vermek gerekirse bir x değişkeni için aşağıdaki örnekteki tüm eşitlemeler aynı değeri gösterecektir. Sayı tabanı kullanımı derleyici için hiçbir şey değiştirmeyecek ama kullanıcının sayıları rahat okuyup durumu anlayabilmesini kolaylaştıracaktır.

1. **unsigned char** x;
- 2.
3. x = 158; // ondalık
4. x = 0x9E; // onaltılık
5. x = 0b10011110; // ikilik
- 6.

C'de Bazı Operatörler

Operatör	Açıklama
() []	Parantez
!	Mantıksal değil
~	Bit tersi
*	İşaretçi operatörü
+ - * /	Aritmetik operatörler
%	Modül

++	1 artır
--	1 eksilt
&	Adres (önde kullanılırsa)
&	Ve operatörü (iki deęerin “ve”lenmiş halini döndürür)
	Veya operatörü (iki deęerin “veya”lanmış halini döndürür)
^	Xor operatörü (iki deęerin “xor”lanmış halini döndürür)
<< >>	Kayıdırma operatörleri
>= <= > <	Karşılaştırma operatörleri
sizeof	Boyut bulma operatörü
==	Mantıksal eşittir
!=	Mantıksal eşit deęil
	Mantıksal veya
&&	Mantıksal ve
=	Eşitleme operatörü
+= -= *= /=	Eşitlik operatörleri
&= = ^=	Eşitlik operatörleri
%= >>= <<=	Eşitlik operatörleri

Şekil 3

Dięer Konular

Aşağıda C diline dair kısaca hatırlatmalar yapılmıştır.

Program Akış Kontrolü

```

1. if (yargı) {
2.     işlemler;
3. }
4. else if (yargı) {
5.     işlemler;
6. }
7. else {
8.     işlemler;
9. }

```

Şekil 4

```

1. switch (yargı) {
2.     case deęer1:
3.         işlemler;
4.         break;
5.
6.     case deęer2:
7.         işlemler;
8.         break;
9.
10.    default:
11.        işlemler;
12. }

```

Şekil 5

Döngüler

1. **while** (yargı) {
2. işlemler;
3. }

Şekil 6

1. **for** (başlangıç işlemleri; yargı; döngü işlemleri) {
2. işlemler;
3. }

Şekil 7

Ön İşlemci Tanımlamaları

1. **#define** değişkenadı değer

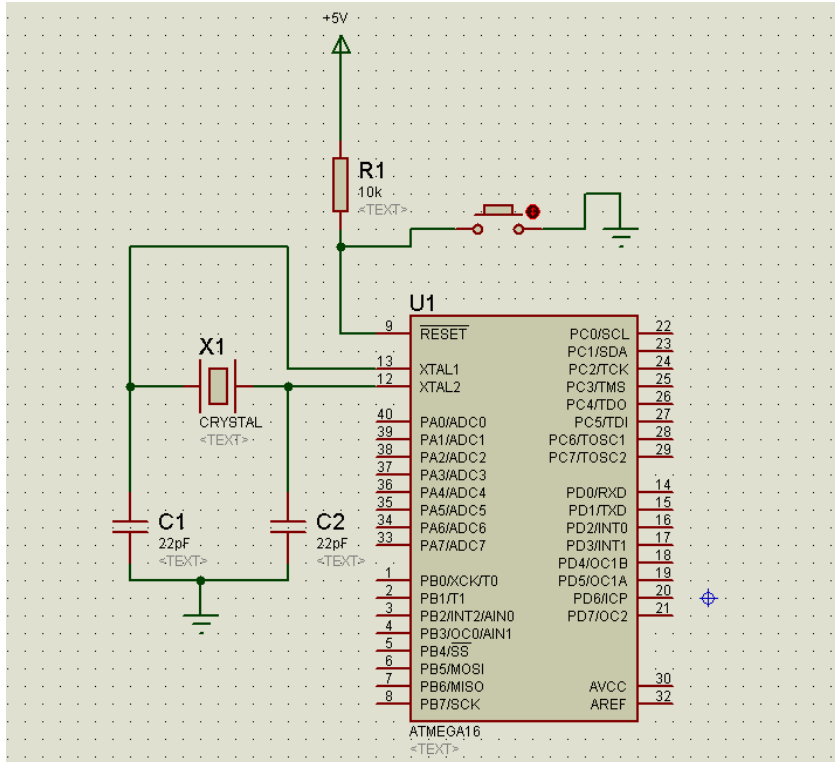
Şekil 8

Diğer Hususlar

Ayrıca kullanıcı fonksiyon tanımlamaları, yapılar (structures), diziler, işaretçiler, dinamik bellek kullanımı gibi C'nin pek çok özelliği 'AVRStudio4' C derleyicisi aracılığı ile ATmel'de de aynen kullanılabilir.

ATMEL BAĞLAMA ŞEKLİ

Hemen her mikro denetleyicide olduğu gibi Atmel'de de yaygın kullanıma sahip bağlama şekli aşağıdaki gibidir. Yalnız Proteus, devrenin beslemesini kendi yaptığından ve bu pinler simülasyonda gözükmediğinden VCC ve GND pinlerini aşağıdaki devrede görülmemektedir.



Şekil 9

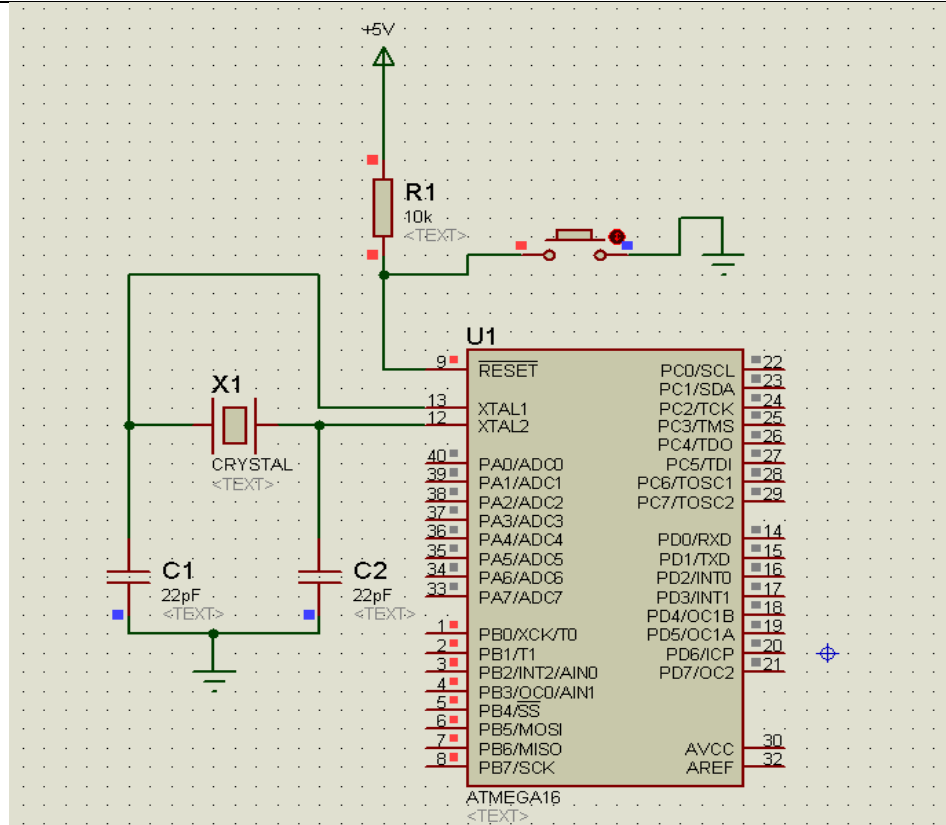
Bu temel bağlamada ATmega16'nın XTAL bacaklarına osilatör bağlanır. Osilatör mikro denetleyicinin bir anlamda kalbi gibidir. Ürettiği saat darbeleriyle (kalp atışları gibi) mikro denetleyicinin düzgün çalışmasını sağlar. Osilatörün doğru biçimde çalışması için hangi osilatörde hangi değerde kapasite kullanılacağı üreticinin bilgi sayfasından (datasheet) öğrenilebilir.

ATmega16 RESET bacağında mantıksal "0" değerini gördüğünde reset işlemi aktif olur. Bunu engellemek için bu bacak 10k'lık bir dirençle 5V'a bağlanır. Gerekliğinde reset işlemini yapabilmek amacı ile bir anahtarın şekildeki gibi bağlanması yeterli olacaktır. Proteus ISIS adlı benzetim programı yukarıdaki bağlama şeklini gerçekleştirilmese bile ATMEL'i çalıştıracaktır. Bu nedenle bundan sonra ATmega16'ya veya ATmega128'e sadece ilgili elemanlar bağlanacaktır. Ayrıca bu programda gösterilmeyen besleme bacakları normalde mutlaka bağlanmalıdır. Burada V_{SS} bacağı toprağa, V_{DD} bacağı ise 5V'a bağlanır.

ÖRNEK 1.0: İlk Program

```
#include <avr/io.h>

main()
{
    DDRB = 0xFF; // B portlarının tamamını çıkış yapıldı
    PORTB = 0xFF; // B portlarının tüm çıkışlarından lojik 1 verildi
}
```

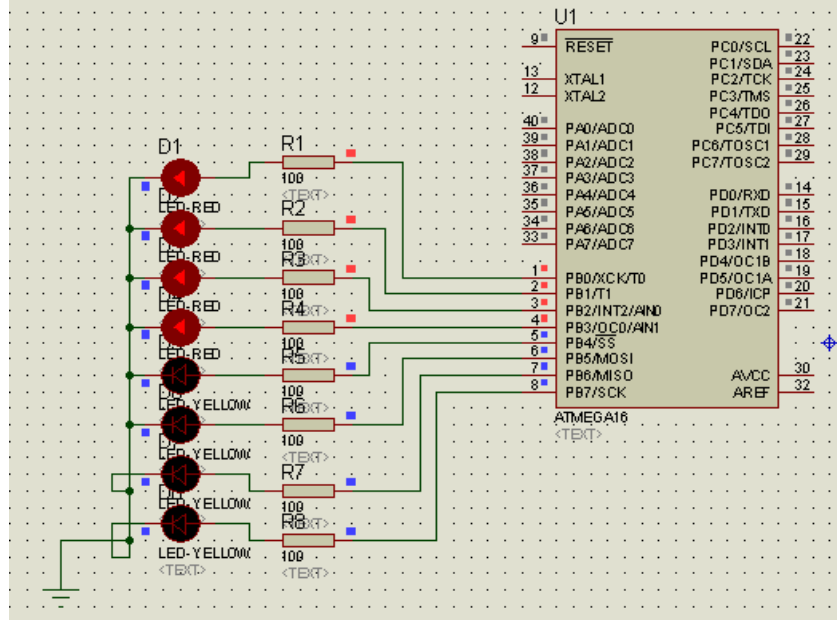


Şekil 10

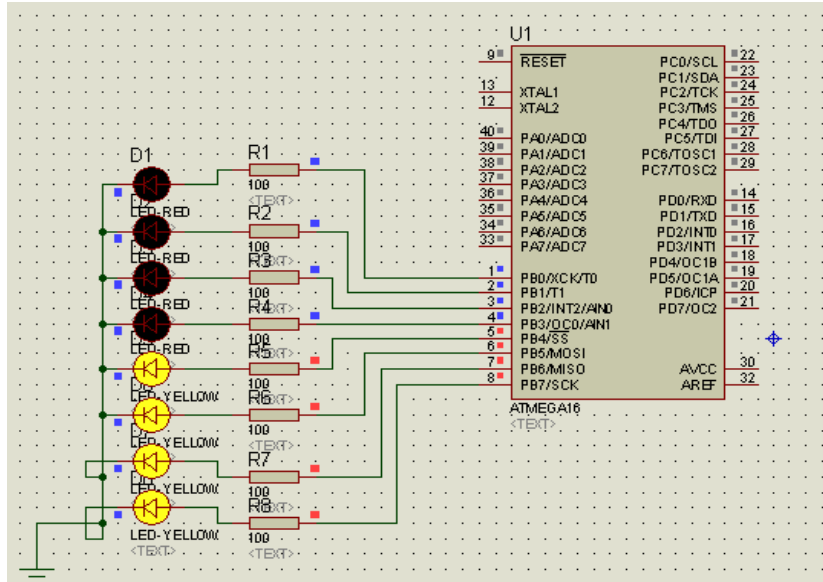
Yukarıda görüldüğü gibi 1 çıkışı verdiğimiz B portunun tüm bitleri kırmızı renkte. Proteusta kırmızı renk mantıksal 1 sinyalini mavi renke mantıksal 0 sinyalini temsil eder. Şekil 10'da görüleceği üzere kod doğru çalışmaktadır.

Örnek 1.1: LED Yakıp Söndürmek

Aşağıda Şekil 11 ve Şekil 12 de görülen devre şemalarında toplam 8 tane LED'i 1 saniye aralıkla yakıp söndüren programın koşturulmaktadır. Buradaki 1 saniye aralığı ayarlayan asıl unsur bu mikro denetleyicide bulunan kristaldir. Örneğin , bu süre 4 Mhz de 1 saniye olurken 1 MHz de 4 saniye olacaktır.



Şekil 11



Şekil 12

KOD:

```
#include <avr/io.h>
#include <util/delay.h>

unsigned char i; // sayaç için kullanacağımız karakter

main()
{
    DDRB = 0xFF; // tüm B portlarını çıkış yaptık
    PORTB=0x0F; // ilk dört bite 1 diğerlerine sıfır verdik

    while(1)
    {
        PORTB =PORTB^0xFF; // 1 olanı 0, 0 olanı 1 yapıyoruz ( XOR )
        for(i=0; i<5; i++) // 1 sn bekle
            _delay_ms(200);
    }
}
```

Şekil 11 ve 12’ de koşturulan kod yukarıda görülmektedir.

Öncelikle iki temel kütüphane eklendi: Bunlardan birisi WINAVR programının yüklü olduğu dizinde ‘avr’ klasörünün içinde io.h(giriş/çıkış kütüphanesi-input/output library). Diğeri de yine aynı dizinde ‘util’ klasörünün içindeki delay.h (bekleme kütüphanesi).

Bunlar yazıldıktan sonra bir değişken tanımlandı. ‘unsigned char’ tipi bu değişken sayaç için kullanılacaktır. Ana fonksiyon içinde DDR isimli bir ifade var. Bu ifade Atmel ailesine özgü mikro denetleyicilere ait genel bir terim. DDR, ‘Data Direction Register’ nin kısaltılmış halidir. Bu register (kütük-yazmaç) bir ‘Port’un giriş yönünde mi çıkış yönünde mi veri alışverişi yapacağını belirler.

ATmega16’ya ait üretici bilgi sayfası (datasheet) mantıksal (logic) 1’in çıkış (output), mantıksal 0’ın ise giriş (input) sinyali anlamına geldiğini yazar.

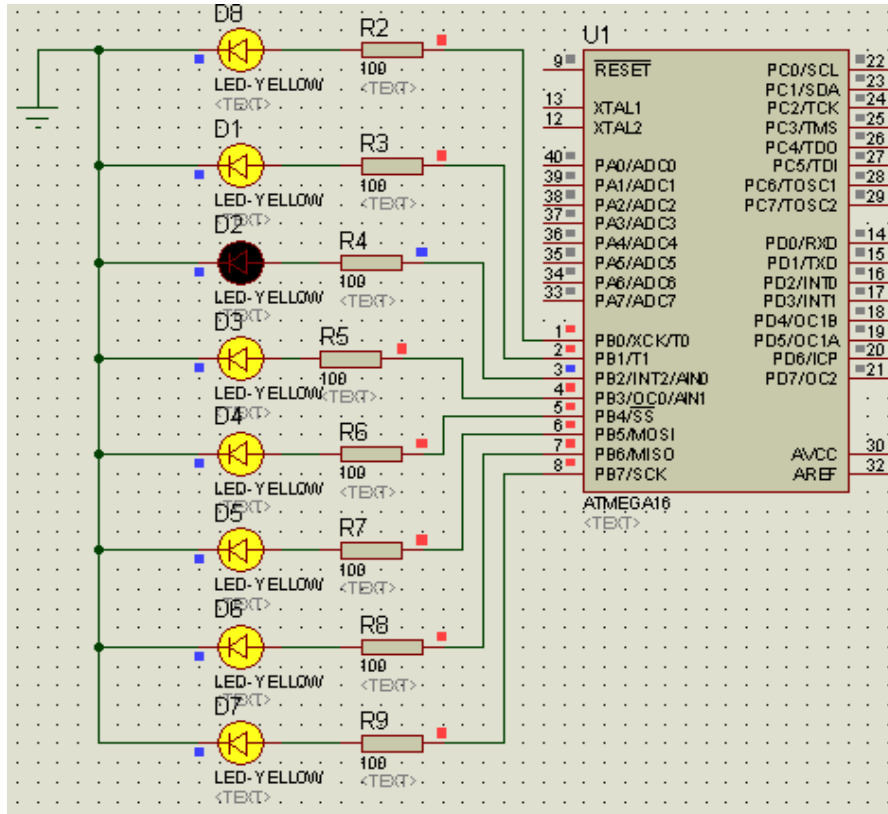
DDRB=0xFF yapılarak bir değere atanmış. 0xFF heksadesimal (16 tabanlı sayı sistemi) 2’li tabana çevrilirse karşılığı: 0xFF=0b11111111. Buradan anlaşılacağı üzere 8 adet bit 1 yapılmış, diğeri bir ifade ile çıkış yapılmıştır. Her bir bit pin ile karşılandığından tüm pinler çıkış olarak ayarlanmış olur.

Ardından PORTB=0x0F yapılmıştır. 0x0F=0b00001111 anlamına gelir. Bu ise sıfırıncı, birinci, ikinci ve üçüncü bitlerden mantıksal 1 çıkışı(output), geriye kalan 4. bittten 7. bite kadar olan bitler ise mantıksal (logic) 0 çıkışını verir.

Bir ‘while’ bloğu kullanıldı. ‘while(1)’ yazılarak program reset işlemine girmediği, mikro denetleyici çalıştığı süre boyunca bloğun içindeki işlem tekrarlanacaktır. Tekrarlanan işlem: ‘PORTB=PORTB^0xFF’ ifadesi ile PORTB’ nin her değeri 1 ile XOR(değil veya) işlemine tabi tutulur. 1 ile Xor işlemi yapılan bir binary sayı 1 ise 0; 0 ise 1 olur. Ardından 5 defa 200 milisaniyelik bir bekleme işlemi yapılır. Sonuçta ise sırasıyla Şekil 11ve Şekil’12 de görüleceği üzere önce ilk 4 led ardından son 4 led 1000 milisaniyelik bekleme ile yanıp söner.

Örnek 1.2:Karaşimşek v1

Bu uygulamada kara şimşek devresinden farklı olarak yanan değil yanmayan led kaymaktadır. Uygulamanın çıktısı Şekil 13’te görülmektedir.



Şekil 13

Uygulamanın kodları:

```
#include <avr/io.h>

void wait (void)
{
    asm volatile ("nop");
}

int main (void)
{
    unsigned char i;
    unsigned long j;

    DDRB = 0xFF; // PORTB'yi çıkış yap =)
    for(;;)
    {
        for (i=0x01; i<0x80; i<<=1)
        {
            PORTB = ~i;

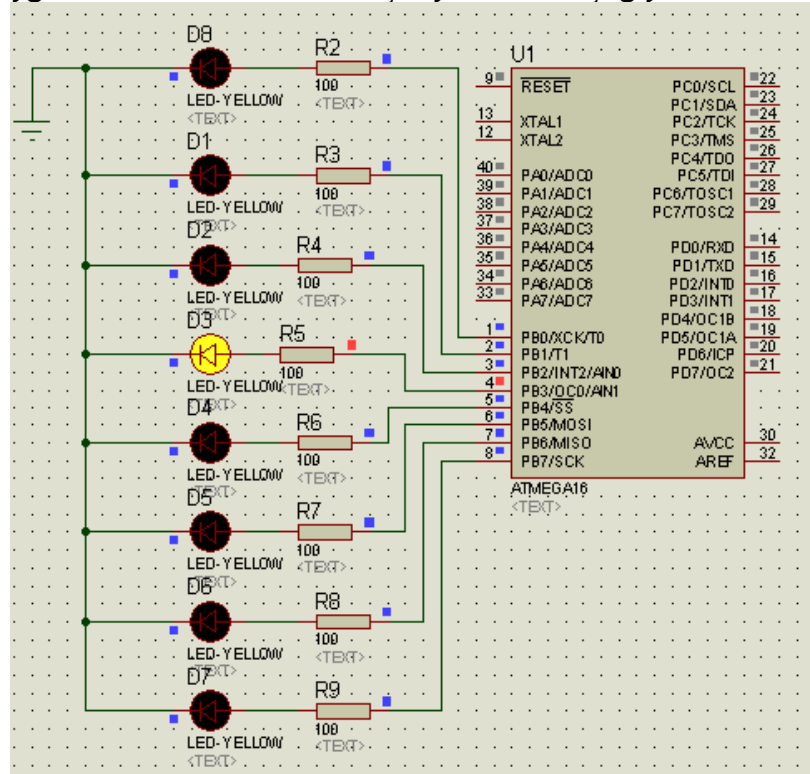
            /* Bekle */
            for (j=0; j<20000; j++) wait();
        }

        for (i=0x80; i>0x01; i>>=1)
        {
            PORTB = ~i;

            /* Delay, beklemeler kütüphanede tanımlı olduğundan kullanılabilir =) */
            for (j=0; j<20000; j++) wait();
        }
    }
}
```

Örnek 1.3:Karaşimşek v2

Önceki uygulamadan farklı olarak ışık yukarı ve aşağıya hareket



edebilmektedir.

Şekil 14

Uygulamanın C dilindeki kodları ise şöyledir:

```
#include <avr/io.h>
#include <util/delay.h>
short int i;

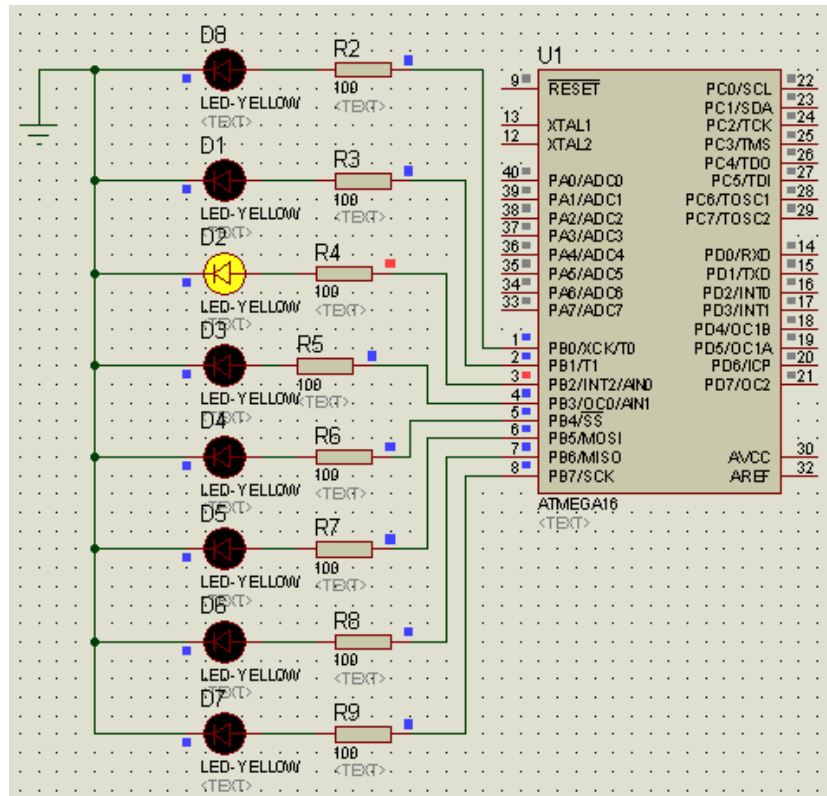
int main()
{
  DDRB = 0xFF; // Port B'nin tamamı çıkış
  PORTB=0; // Tüm çıkışlardan lojik 0 alıyoruz
  while(1)
  {
    PORTB = 1; // Portb nin son biti 1, yani sadece PIN B0 yanacak
    _delay_ms(100); // 100 ms gecikme koyuyoruz =)
    for(i=0;i<7;i++)
    {
      PORTB*=2; // ikilik tabanda ikiyle çarpmak biti 1 basamak sola kaydırma.
      _delay_ms(100);
    }
    for(i=0;i<6;i++)
    {
      PORTB /=2; // ikilik tabanda ikiye bölmek biti 1 basamak sağa kaydırma.
      _delay_ms(100);
    }
  }
}
```

Karaşimşek v3 (Örnek 1.4)

```
#include <avr/io.h>
#include <util/delay.h>

int y;
int main()
{
  DDRB=0xFF;
  y=0;

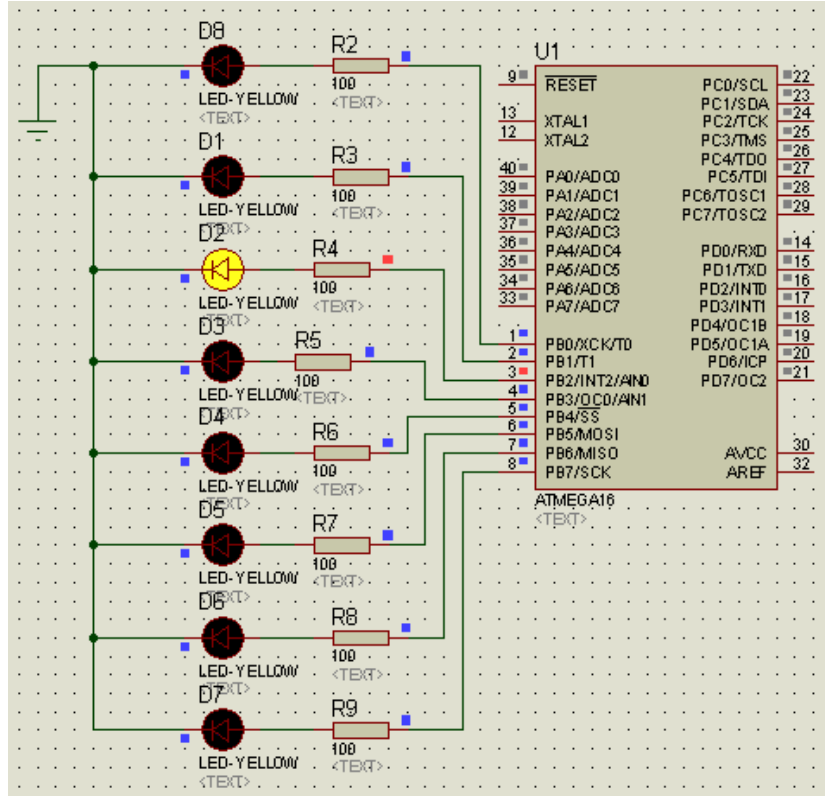
  _delay_ms(100);
  while(1)
  {
    PORTB = 0x01;
    _delay_ms(100);
    for(y=0;y<7;y++)
    {
      PORTB = PORTB << 1;
      _delay_ms(100);
    }
    for(y=0;y<6;y++)
    {
      PORTB = PORTB >> 1;
      _delay_ms(100);
    }
  }
}
```



Şekil 15

Yukarıdaki örneklerde ufak değişikliklerle aynı işlevi yapan çeşitli kodlar yazıldı. Genel olarak giriş çıkış işlemleri uygulamaya döküldü.

Bu uygulamalardan sonra stajda bir başka sayfaya geçildi. Bu noktadan sonra ATmega16 ve ATmega 128'e ait çeşitli modülleri kullanarak farklı uygulamalar yapıldı.



Şekil 16

```
#include <avr/io.h>
#include <util/delay.h>

int y;
int main()
{
  DDRB=0xFF;
  y=0;

  _delay_ms(100);
  while(1)
  {
    PORTB = 0x01;
    _delay_ms(100);
    for(y=0;y<7;y++)
    {
      PORTB = PORTB << 1;
      _delay_ms(100);
    }
    for(y=0;y<6;y++)
    {
      PORTB = PORTB >> 1;
      _delay_ms(100);
    }
  }
}
```

KESMELER

Kesme, gömülü sistemler için önemli bir kavramdır. Bir mikro denetleyicide bir program koşarken, kesmeye sebep olacak bir durum meydana geldiğinde (kesme bayrağını aktif hale getirecek-set edecek-bir durum) program nerede olursa olsun, hemen kesme ile ilgili olan alt programa dallanır. Kesme içindeki işlemler bitinceye veyahut kesme koşulu bir şekilde sonlandırılıncaya kadar bu dalın içinde kalır. Kesme koşulları sona erdiğinde program ana fonksiyona geri döner ve kaldığı yerden çalışmaya devam eder.

Atmega ailesine mensup olan mikro denetleyicilerde bir çok yazmaçta(register-kütük) kesmeye dair bayraklar ve yazmaçlar(kütük-register) bulunabilir.

Stajda ileriki uygulamalarda ele alınan projelerden fark edileceği üzere kesmeler bir çok yerde kullanılmaktadır. Bu sebepten daha stajın ilk günlerinden içlerinde kesmelere dair bit bulunduran, kesmeleri başlatan, bitiren... bir çok yazmaç ele alınmış ve bu yazmaçlara bağlı olarak uygulamalar yapılmıştır.

Bundan önce üreticinin bilgi sayfasında(datasheet) sayfa 45'ten alınan ATmega16'da bulunan kesme çeşitlerine bakılacak olursa:

Tablo 1

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$002	INT0	External Interrupt Request 0
3	\$004	INT1	External Interrupt Request 1
4	\$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	\$008	TIMER2 OVF	Timer/Counter2 Overflow
6	\$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	\$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	\$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	\$010	TIMER1 OVF	Timer/Counter1 Overflow
10	\$012	TIMER0 OVF	Timer/Counter0 Overflow
11	\$014	SPI, STC	Serial Transfer Complete
12	\$016	USART, RXC	USART, Rx Complete
13	\$018	USART, UDRE	USART Data Register Empty
14	\$01A	USART, TXC	USART, Tx Complete
15	\$01C	ADC	ADC Conversion Complete
16	\$01E	EE_RDY	EEPROM Ready
17	\$020	ANA_COMP	Analog Comparator
18	\$022	TWI	Two-wire Serial Interface
19	\$024	INT2	External Interrupt Request 2
20	\$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	\$028	SPM_RDY	Store Program Memory Ready

Tablo 1’de görüldüğü üzere ATmega16’da bir çok kesme çeşidi vardır. Şimdi bu kesmeleri kullanmayı öğrenmek için bakılan yazmaçlara ve onların görevlerine bakalım.

Staj süresince ilk ele alınan yazmaç Status Register ve onun içinde yer alan SREG’in 7. biri Global Interrupt Enable’dir.

Status register çalıştırılmış aritmetik işlemlerin sonuçları hakkındaki veriyi içerir. Bu bilgi programın akışını değiştirmek için kullanılabilir. SREG, Aritmetik mantıksal ünitenin (Arithmetic Logic Unit-ALU) işlemlerinden sonra güncellenir. Bu sayede programın daha hızlı koşurulması sağlanır. AVR Status Register-SREG- şöyle gösterilir:

Tablo 2

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

SREG’in 7. biti Staj süresince yapılacak uygulamalar için oldukça büyük önem taşımaktadır:

Bit 7 - I: Global Interrupt Enable : Kesmelerin aktif hale getirilmesi için Global Interrupt Enable ayarlanmalıdır(‘1’ yapılmalıdır-set edilmelidir-aktif hale getirilmelidir). Bu aşamadan sonra ayrı ayrı her interrupt kendi bayrakları sayesinde aktif hale getirilebilir. Ancak Global Interrupt Enable Register temizlendiğinde(‘aktif halden çıkarıldığında’) hiçbir kesme çalışmaz.

Bu yazmacın kesmeler için başlama noktası anlamına gelen 7. bitine bakıldıktan sonra genelden özele ilerleyerek, bir başka yazmaç ele alındı. General Interrupt Control Register ismini taşıyan bu yazmacın bit yapısı şu şekilde idi:

Tablo 3

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	-	-	-	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tablo 3’ten anlaşıldığı üzere GICR, harici kesmeleri temsil eden INT0-1-2 bitleri ile Interrupt vector select ile Interrupt Vector change Enable bitlerini taşımaktadır.

IVSEL ve IVCE, C kodu üzerinden işaretlenebildiğinden(Atmel’de vektör isimlerinin ayrı olup çağırıldıklarında bu bitler derleyici tarafından otomatik olarak ayarlanmaktadır) kod yazılırken dikkate alınmadı. Ancak harici kesmelerin aktif hale gelmesini sağlayan son 3 bit önemlidir. Bu nokta ileride anlatılacaktır.

Sonuç olarak stajın bu safhasında Atmel ailesine mensup bir mikro denetleyicide Kesme çeşitleri, haritası ve de kesme işlemleri için önce hangi adımın atılması gerektiği görüldüğü, bunun ardından ilk kesme çeşidi olan harici kesmeler ile ilgilenildi.

4. Harici Kesmeler

Harici kesmeler INT0,INT1,INT2 pinlerinin tetiklenmesi ile meydana gelir. INT0-1-2 pinleri çıkış olarak ayarlansalar da eğer aktif hale getirildilerse kesme meydana gelir. Harici kesmeler kendisine gelen elektriksel sinyalde düşen kenar, yükselen kenar veya herhangi bir mantıksal değişim sonucunda tetiklenebilir. MCU Control Register (MCUCR) isimli yazmaçta yapılan ayarlamalarla kesmenin ne zaman meydana geleceği ayarlanabilir. Kesme durumu meydana gelince program, harici kesme fonksiyonuna dallanır ve orada yazan kodu işlemeye başlar.

Tablo 4

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tablo 4'te harici kesme ile ilgili olan MCUCR(MCU Control Register) görülmektedir. Aşağıda ise bu bölümde işe kullanılacak olan bitler ele alınmıştır.

Bit 3, 2 - ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0

Harici kesme 1 (The External Interrupt 1) SREG ayarlanıp, GICR'de INT1 ile ilgili bit ayarlandığında aktif hale gelir. ISC11 ve ISC10 ise INT1 pinine gelen elektriksel sinyal ile ilgilidirler. Bu bitler hangi elektriksel sinyalde harici kesmenin aktif olacağını belirlerler. Tablo 4.1'de bu durum anlatılmaktadır.

Tablo 4.1 Interrupt 1 Sense Control

ISC11 ISC10 Açıklama

0	0	INT1 '0' (low level) iken kesme üretir.
0	1	Herhangi bir mantıksal değişim (Any logical change) kesme üretir.
1	0	INT1 düşen kenarda kesme üretir.
1	1	INT1 yükselen kenarda kesme üretir.

Bit 1, 0 - ISC01, ISC00: Interrupt Sense Control 0 Bit 1 and Bit 0

SREG ve ona ilişkin kesme maskesi aktif hale getirildiğinde INT 0 aktif hale gelir. Bu bitler de yukarıdaki bitlerle aynı görevi yaparlar. INT0 harici kesmesinin hangi elektriksel değişim sonucunda meydana geleceğini anlatan bu bitlerin hangi durumda ne yapacakları Tablo 4.2'da anlatılmaktadır.

Table 4.2 Interrupt 0 Sense Control

ISC01 ISC00 Açıklama

0	0	INT0 '0'(low level) iken kesme üretir.
0	1	Herhangi bir mantıksal değişimde INT0 kesme üretir.
1	0	INT0 düşen kenarda kesme üretir.
1	1	INT0 yükselen kenarda kesme üretir.

INT2 ile ilgili kesme aşağıdaki yazmacın sorumluluğundadır.
MCU Control and Status Register - MCUCSR isimli bu yazmacın bit haritası aşağıdaki gibidir.

Tablo 5

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0						See Bit Description

Bu bitlerden INT2 ile ilgili olan bit ISC2'dir.

Bit 6 - ISC2: Interrupt Sense Control 2

Asenkron Harici Kesme 2, SREG'e ait I-bitinin(Interrupt) ve de GICR de INT2'ye ait bitin aktif edilmesi ile çalışabilir. ISC2 '0 yapıldığında', INT2 harici kesmesi düşen kenarda aktif hale gelir. ISC2 '1' yapıldığında ise yükselen kenarda aktif hale gelir.

INT0-1-2 harici kesmelerini aktif hale getiren yazmaç Tablo 6'da bit haritası görülen GICR'dir.

Tablo 6

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Her 3 kesmeyi de aktif hale getirmek için, SREG biti ayarlanıp, ilgili ISCxx bitleri de istenilen kesmeye girme biçimine getirilip GICR'de ilgili kesme biti '1' yapıldığında harici kesme kullanıma hazır hale getirilmiş olur.

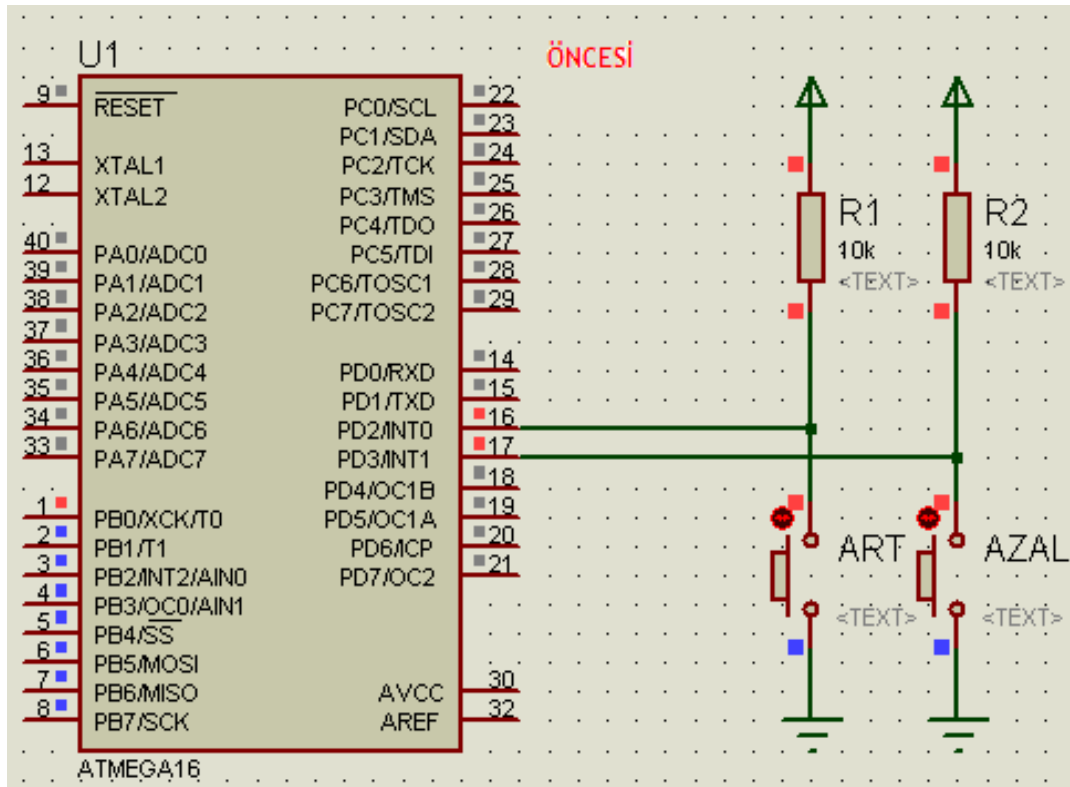
Örnek 2.0:INT1 ve INTO KESMESİ V.1

Aşağıdaki uygulamada iki adet dış kesme aynı anda anlatılmıştır. INTO kesmesi ve INT1 kesmesi sayaç gibi görevlendirilmiştir. Bir butona basılınca PORTB'nin değeri 1 artacak, diğer butona basınca da PORTB'nin değeri bir azalacaktır.

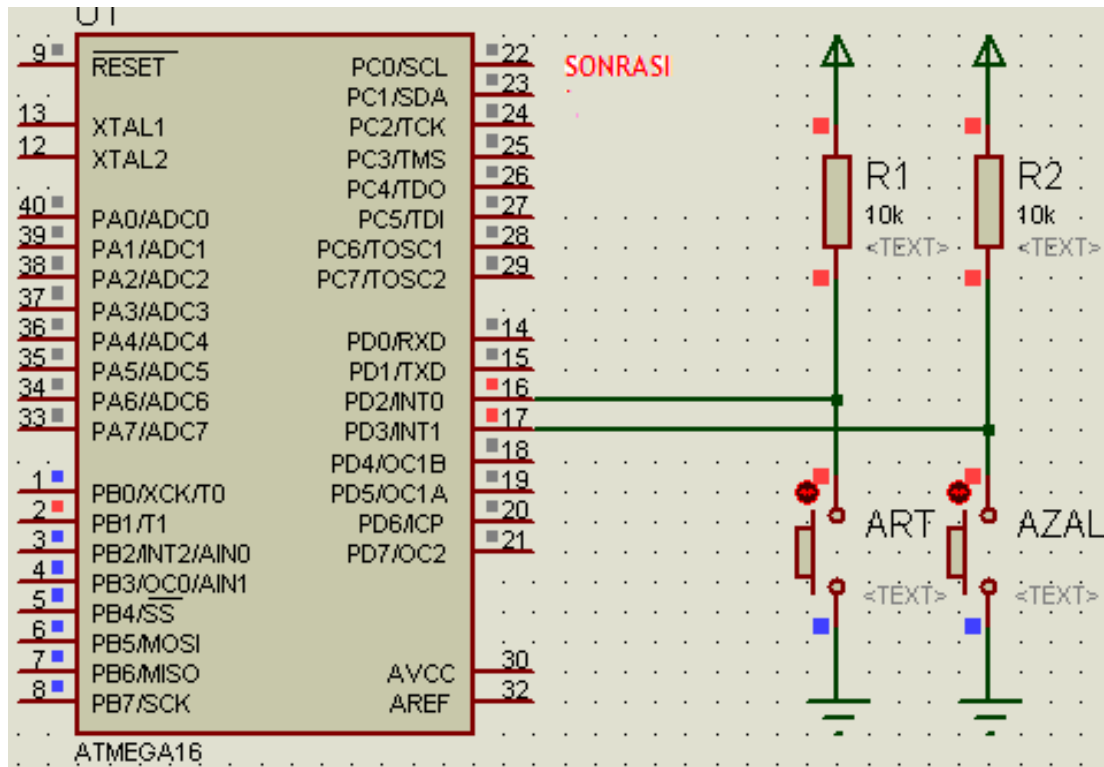
```
1  #include <avr/interrupt.h> // kesme kütüphanesi
2  // kesme kütüphanesinin içinde io.h bulunduğundan io.h'ı eklemedik
3  #include <util/delay.h>
4  // util klasörünün içindeki gecikme kütüphanesi
5  volatile short int i = 0;
6  ISR(INT0_vect) //3 cycle
7  {
8      PORTB +=1; // INTO kesme butonuna basıldığında PORTB'yi bir arttır
9      _delay_ms(200); //200 ms bekle
10 }
11 ISR(INT1_vect) //3 cycle
12 {
13     PORTB -=1; // INTO kesme butonuna basıldığında PORTB'yi bir arttır
14     _delay_ms(200);
15 }
16 void main()
17 {
18     DDRB = 0xFF; //B portlarının tamamı çıkış
19     DDRD = 0; // D portları giriş
20     PORTB = 0;
21     PORTD = 0;
22     SREG = 128; // SREG kütüğünün 7. biti 1 olunca kesmeler aktif olur
23     //sei(); de diyebilirdik ya da SREG |= _BV(7); bunların tamamı aynı anlamda
24     // General Interrupt Enable yaptık
25     GICR = 0b11000000; //4 cycle - Global İnterrupt Control
26     // INTO ve INT1 dış kesmelerini aktive ettik 6. biti de 1 yapsak INT2'yi de aktive ederdik
27     MCUCR |= _BV(ISC01) | _BV(ISC11); ; // 3cycle
28     // MCUCR |= (1<<ISC01) | (1<<ISC11); de diyebilirdik.
29     MCUCR &= ~_BV(ISC00) | ~_BV(ISC10); ; // 3 cycle
30     // MCUCR = MCUCR & ~_BV(ISC00) & ~_BV(ISC10); da diyebilirdik.
31     // Kesmeyi Azalan bacakta yaptırдық, bu registerı değiştirerek artan bacakta,
32     // ya da herhangi bir değişimde kesmeye sokabiliriz. bkz: datasheet
33     while(1);
34 }
```

Yukarıdaki programda açıklamalar kısmında gösterildiği üzere iki adet kesme aktif hale getirildi. Böylece artık normal giriş veya çıkış olmaktan çıkıp harici kesme haline gelen iki pine şekil 19 ve 20'deki gibi butonlar eklendi. Bu butonlar ile iki pine yükselen kenar, alçalan kenar ve herhangi bir mantıksal değişim işlemlerini yapmak olanaklı kılındı. Programda MCUCR (MCU control register) kütüğüne ait bitler çeşitli şekillerde ayarlanarak harici kesmeye hangi değişim olduğunda girileceği kararlaştırıldı. ISCxx (interrupt sense control) bitlerinin hangi kombinasyonlarda hangi değişimde kesmeye gireceği üretici bilgi sayfasında (datasheet) bulunabilir.

Koda ilişkin Proteus modeli de aşağıdadır. ART butonuna basmadan önceki ve sonraki durumları yan yana görebilirsiniz. PORTB'deki değişim şekil 19 ve 20'de fark edilecektir.



Şekil 17



Şekil 18

Örnek 2.1:INT1 ve INTO KESMESİ V.2

Bu uygulamada staj sırasında yazılan LCD sürmeye dair kütüphane de kullanılmıştır.

```
1  #include "ozen_lcd.h" // yazdığım lcd kütüphanesi =)
2  int i,j; // genel değişkenlerimiz
3
4  SIGNAL(SIG_OUTPUT_COMPARE0) // timer0 output compare kesmesi
5  {
6      Delay_sayac+=1; // sayacımızı 1 arttırıyoruz
7  }
8
9  ISR(INT1_vect) //int1 kesmesi
10 {
11     PORTF+=1;
12     for(i=0;i<10;i++)
13     {
14         _delay_ms(100);
15     }
16     j+=1;
17 }
18
19 ISR(INT0_vect) //int0 kesmesi
20 {
21     PORTF-=1;
22     for(i=0;i<10;i++)
23     {
24         _delay_ms(100);
25     }
26     j-=1;
27 }
28
29 int main() // ana fonksiyon =)
30 {
31     DDRF=0xFF; // F portları çıkış
32     EICRA = 0xAA; //düşen kenarda kesme
33     EICRB = 0xAA; // düşen kenarda kesme
34     EIMSK=0xFF; // tüm dış kesmeler aktif
35     lcd_on_hazirlik();
36     SREG |= _BV(7); //kesmeler aktif
37     LCD_hazirlama();
38     while(1)
39     {
40         LCD_katar_yaz(" OZEN OZKAYA          ");
41         LCD_sayi_yaz(j);
42         _delay_ms(200);
43         LCD_sil();
44     }
45 }
```

Header File olarak “ozen_lcd.h” ı kullandı. Bu da staj sırasında yazılan LCD kütüphanesidir.

```

1  #include <avr/interrupt.h>
2  #include <avr/pgmspace.h>
3  #include <util/delay.h>
4
5  void lcd_on_hazirlik(void);
6  void Delay_kostur(uint16_t delayUnits);
7  void delay50us(uint16_t delayUnits);
8  void LCD_git(uint8_t row, uint8_t column);
9  void LCD_sil(void);
10 void LCD_sayi_yaz(uint16_t integer);
11 void LCD_katar_yaz(uint8_t* stringPointer);
12 uint8_t LCD_byte_oku(uint8_t selectedRegister);
13 uint8_t LCD_nibble_oku(uint8_t selectedRegister);
14 void LCD_byte_yaz(uint8_t selectedRegister, uint8_t byte);
15 void LCD_nibble_yaz(uint8_t selectedRegister, uint8_t nibble);
16 void LCD_bekle(void);
17 void LCD_hazirlama(void);
18 void delay_ms(unsigned char time_ms);
19 void lcd_sur();
20
21
22 volatile uint16_t Delay_sayac;
23 int yuzde;
24 int prescale;
25 int yuzde_A;
26 int yuzde_B;
27 int temp=0;
28 int sayac=0;
29 int sayidegeri;
30 int basamakNum=1;
31 char karakter;
32 int konum;
33 int son;
34
35 #define CLOCK_FREQUENCY      16000000L
36 //Zamanlayıcı sabitleri
37 #if CLOCK_FREQUENCY == 16000000L
38 #define OCR_1MILLISECOND     248
39 #define OCR_1MICROSECOND     98
40 #elif CLOCK_FREQUENCY == 8000000L
41 #define OCR_1MILLISECOND     124
42 #define OCR_1MICROSECOND     49
43 #endif
44

```

```

45 #define LCD_OUT      PORTB
46 #define LCD_IN      PINB
47 #define LCD_DDR     DDRB
48 #define ENABLE      6
49 #define RW           5
50 #define RS           4
51 #define D7           3
52 #define D6           2
53 #define D5           1
54 #define D4           0
55 // Register seçimi sabitleri
56 #define DATA_REGISTER      0
57 #define COMMAND_REGISTER   1
58
59 #ifndef Tesekkurler
60 # warning "Hatalıysam ara: 05058623355"
61 #endif
62
63
64 /*void port_hazirla()
65 {
66     DDRA = 0xFF;
67     DDRB = 0xFF;
68     DDRC = 0xFF;
69     DDRD = 0b11110000;
70
71     PORTA=0;
72     PORTB=0b0;
73     PORTC=0;
74     PORTD=0;
75 }*/
76
77 /*-----LCD ve Kesmeler Ön Hazırlık -----*/
78
79 void lcd_on_hazirlık(void)
80 {
81     TCCR0 = 0;
82     TIFR |= _BV(OCF0);
83     TCCR0 |= _BV(WGM01) | _BV(CS01);
84     TCNT0 = 0;
85     TIMSK |= _BV(OCIE0);
86     OCR0 = OCR_1MICROSECOND;
87 }

```

```

88
89  /*-----
90  -----Bekletme Fonksiyonları-----
91  -----*/
92
93  void delay50us(uint16_t delayUnits)
94  {
95      Delay_kostur(delayUnits);
96  }
97
98  void Delay_kostur(uint16_t delayUnits)
99  {
100     TCNT0 = 0;
101
102     Delay_sayac = 0;
103
104     while (Delay_sayac != delayUnits)
105         ;
106 }
107
108 /*-----
109 -----LCD temizle-----
110 -----*/
111
112 void LCD_sil(void)
113 {
114     LCD_byte_yaz(COMMAND_REGISTER, 0x01);
115
116     konum = 0;
117 }
118
119 /*-----
120 -----LCD git-----
121 -----*/
122
123 void LCD_git(uint8_t row, uint8_t column)
124 {
125     //nereye gideceeen=)
126     konum = (row * 20) + column;
127
128     //buraya dikkat
129     if(konum < 20)
130         LCD_byte_yaz(COMMAND_REGISTER, 0x80 | konum);
131     else if(konum >= 20 && konum < 40)
132         LCD_byte_yaz(COMMAND_REGISTER, 0x80 | (konum % 20 + 0x40));
133     else if(konum >= 41 && konum < 60)
134         LCD_byte_yaz(COMMAND_REGISTER, 0x80 | (konum % 40 + 0x14));
135     else if(konum >= 60 && konum < 80)
136         LCD_byte_yaz(COMMAND_REGISTER, 0x80 | (konum % 60 + 0x54));
137 }

```

```

138  /*-----
139  -----LCD'ye Katar yaz -----
140  -----*/
141  void LCD_katar_yaz(uint8_t* stringPointer)
142  {
143      while (*stringPointer)
144          LCD_byte_yaz(DATA_REGISTER, *stringPointer++);
145  }
146
147  /*-----
148  -----LCD'ye sayi yaz -----
149  -----*/
150
151  void LCD_sayi_yaz(uint16_t integer)
152  {
153
154      uint8_t binler = integer / 1000;
155      LCD_byte_yaz(DATA_REGISTER, binler + 0x30);
156
157      uint8_t yuzler = (integer - binler*1000) / 100;
158      LCD_byte_yaz(DATA_REGISTER, yuzler + 0x30);
159
160      uint8_t onlar = (integer - binler*1000 - yuzler*100) / 10;
161      LCD_byte_yaz(DATA_REGISTER, onlar + 0x30);
162
163      uint8_t birler = (integer - binler*1000 - yuzler*100 - onlar*10);
164      LCD_byte_yaz(DATA_REGISTER, birler + 0x30);
165  }
166
167  /*-----
168  -----LCD Hazirlama Fonk.-----
169  -----*/
170  void LCD_hazirlama(void)
171  {
172      lcd_on_hazirlik();
173      LCD_DDR = 0x7f;
174      delay50us(300);
175      LCD_nibble_yaz(COMMAND_REGISTER, 0x03);
176      delay50us(100);
177      LCD_nibble_yaz(COMMAND_REGISTER, 0x03);
178      delay50us(100);
179      LCD_nibble_yaz(COMMAND_REGISTER, 0x03);
180      delay50us(100);
181      LCD_nibble_yaz(COMMAND_REGISTER, 0x02);
182      delay50us(100);
183      LCD_byte_yaz(COMMAND_REGISTER, 0x28);
184      delay50us(250);
185      LCD_byte_yaz(COMMAND_REGISTER, 0x08);
186      delay50us(250);
187      LCD_byte_yaz(COMMAND_REGISTER, 0x01);
188      delay50us(250);
189      LCD_byte_yaz(COMMAND_REGISTER, 0x0f);
190      delay50us(250);
191  }

```

```

192
193 /*-----
194 -----LCD Cevap süresi -----
195 -----*/
196 void LCD_bekle(void)
197 {
198     delay50us(1);
199 }
200
201 /*-----
202 -----LCD ayarla sil -----
203 -----*/
204 void LCD_nibble_yaz(uint8_t selectedRegister, uint8_t nibble)
205 {
206     LCD_OUT = _BV(ENABLE);
207     LCD_OUT |= nibble;
208     if(selectedRegister == DATA_REGISTER)
209     {
210         LCD_OUT |= _BV(RS);
211     }
212     else
213     {
214         LCD_OUT &= ~(_BV(RS));
215     }
216     LCD_OUT &= ~(_BV(ENABLE));
217 }
218
219 /*-----
220 -----LCD'ye byte yaz -----
221 -----*/
222 void LCD_byte_yaz(uint8_t selectedRegister, uint8_t byte)
223 {
224     uint8_t yuksekNibble = byte >> 4;
225     uint8_t dusukNibble = byte & 0x0f;
226     if(selectedRegister == DATA_REGISTER && konum == 20)
227     {
228         LCD_byte_yaz(COMMAND_REGISTER, 0xC0);
229         delay50us(25);
230     }
231     else if(selectedRegister == DATA_REGISTER && konum == 40)
232     {
233         LCD_byte_yaz(COMMAND_REGISTER, 0x94);
234         delay50us(25);
235     }
236     else if(selectedRegister == DATA_REGISTER && konum == 60)
237     {
238         LCD_byte_yaz(COMMAND_REGISTER, 0xd4);
239         delay50us(25);
240     }
241     LCD_bekle();
242     LCD_nibble_yaz(selectedRegister, yuksekNibble);
243     LCD_nibble_yaz(selectedRegister, dusukNibble);
244     if(selectedRegister == DATA_REGISTER && ++konum == 80)
245         konum = 0;
246 }
247

```

```

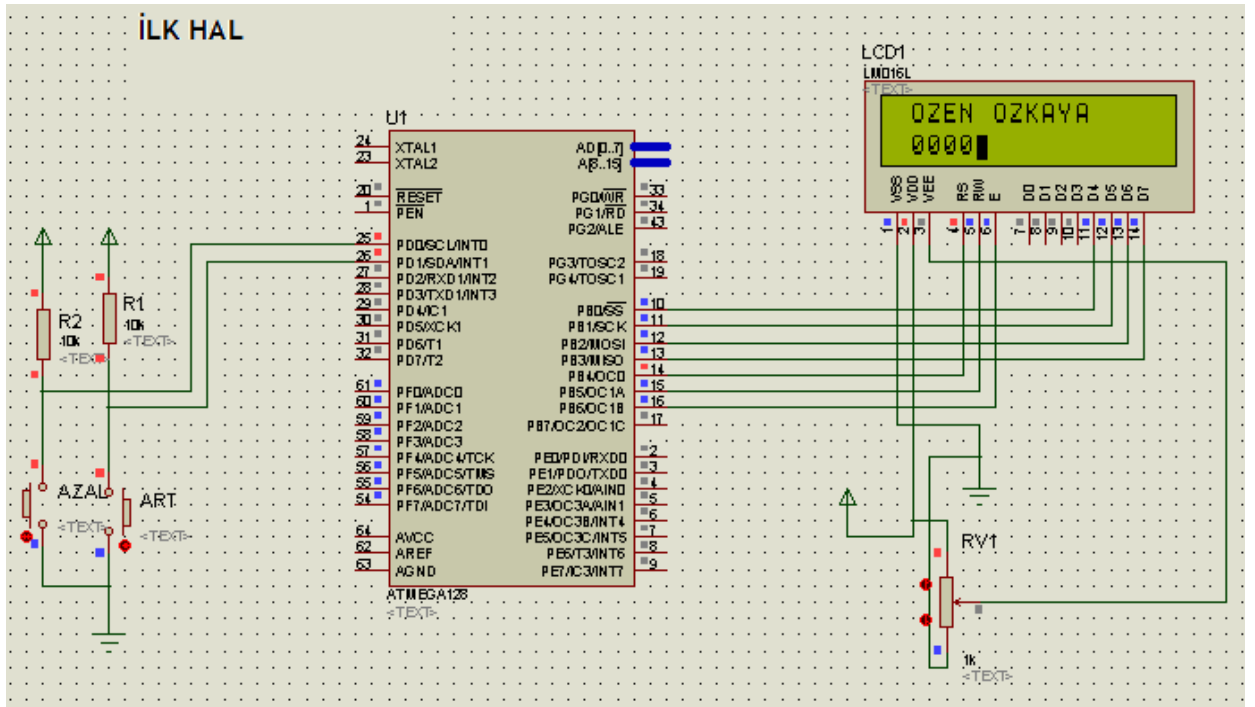
248     uint8_t LCD_nibble_oku(uint8_t selectedRegister)
249     {
250         uint8_t nibble = 0x00;
251         LCD_OUT |= _BV(ENABLE) | _BV(RW);
252         if(selectedRegister == DATA_REGISTER)
253         {
254             LCD_OUT |= _BV(RS);
255         }
256         else
257         {
258             LCD_OUT &= ~(_BV(RS));
259         }
260         asm volatile("nop\n\t");
261         asm volatile("nop\n\t");
262         nibble = LCD_IN & 0x0f;
263         LCD_OUT &= ~(_BV(ENABLE));
264         LCD_OUT &= ~(_BV(RW));
265         LCD_DDR = 0x7f;
266         return nibble;
267     }
268
269     uint8_t LCD_byte_oku(uint8_t selectedRegister)
270     {
271         uint8_t byte = 0x00;
272         uint8_t yuksekNibble = 0x00;
273         uint8_t dusukNibble = 0x00;
274
275         yuksekNibble = LCD_nibble_oku(selectedRegister);
276         dusukNibble = LCD_nibble_oku(selectedRegister);
277         byte = (yuksekNibble << 4) | dusukNibble;
278
279         return byte;
280     }
281

```

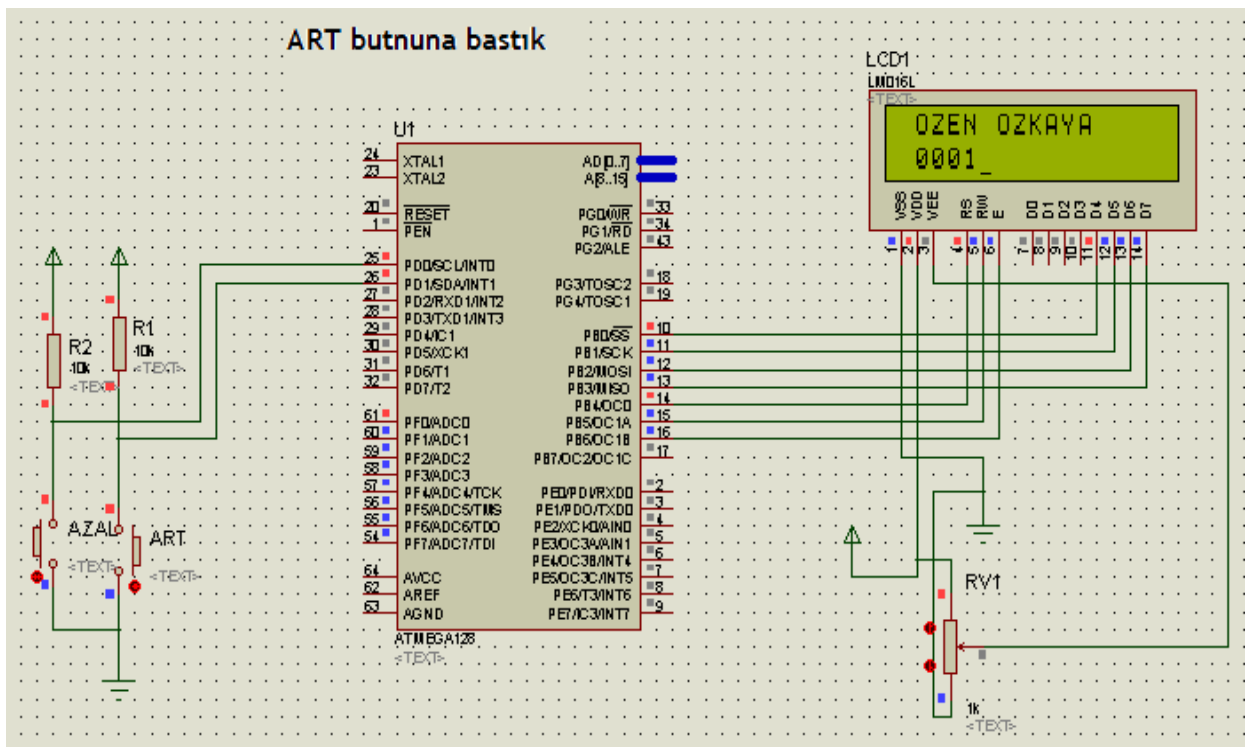
Örnek 2.1:INT1 ve INTO KESMESİ V.2'başlığının altında görülen koda dikkat edileceği üzere main() fonksiyonunun içerisinde LCD ile ilgili kimi fonksiyonlar bulunmaktadır. Bu fonksiyonların tamamı yukarıdaki LCD kütüphanesinde hazırlanmış olan fonksiyonlardır. Tüm o fonksiyonlar 'ozen_lcd.h' kütüphanesinden çağırılmıştır.

Şekil 21 ve 22'de Proteus modeli görülmektedir.,

Program kısaca isim yazıyor, LCD de sayaç tutuyor, butonlara bastıkça sayaçtaki sayı bir artıyor ya da bir azalıyor.



Şekil 19



Şekil 20

PWM (Pulse With Modulation - Darbe Genlik Modülasyonu)

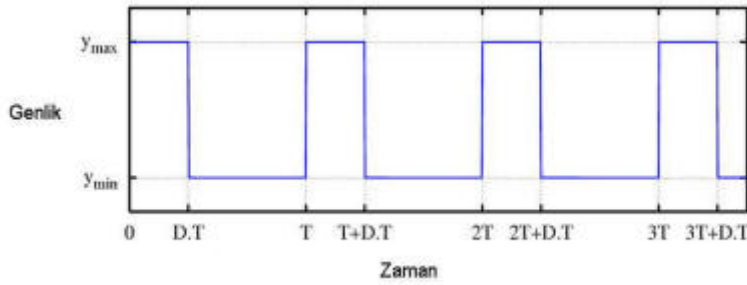
Üretilen kare dalga darbelerinin, genişliklerini kontrol ederek, çıkışta üretilmek istenen analog elektriksel değerin veya sinyalin elde edilmesi tekniğidir.

PWM elektrik ve elektronikte birçok alanda, farklı amaçlar için kullanılmaktadır. Telekomünikasyon, güç, voltaj düzenleyiciler, ses üreteçleri veya yükselteçler gibi çeşitli uygulama alanları ve farklı uygulamaları bulunmaktadır.

Heralde günümüzde PWM'in en çok duyulduğu yer, güç kaynaklarıdır. SMPS (Switched mode power supply) güç kaynakları, düzenlenecek olan çıkış voltajlarını bu teknikten yararlanarak elde etmektedirler. Bu sayede, yüksek akım ve düşük voltajlı güç elde edimimleri için, transformatörlerden çok daha etkin ve çok daha küçüklerdir. Bilgisayarınızın kasaındaki güç kaynağını düşündüğünüzde, 350Wattlık çıkış gücüne sahip olan bir güç kaynağının nasıl bu kadar küçük ve etkin tasarlandığının cevabı SMPS olmasıdır.

Temelleri

Üretilen kare dalga darbe sinyallerinin genişliklerinin ortalaması, çıkışta üretilmek istenen analog değerin elde edilmesini sağlar. En iyi açıklama sanırım aşağıdaki şekil üzerinden yapılabilir.



Şekil 21

Staj süresince çeşitli frekanslarda pwm üretmek gerektiğinden, aşağıda da görüleceği üzere bir çok pwm uygulaması mütalaa edildi ve bunlardan bir kısmı uygulandı.

Öncelikler basit pwm kodları ele alındı. Ardından daha profesyonel kodlar yazıldı.

Örnek 3.0: Basitçe PWM Üretilmesi

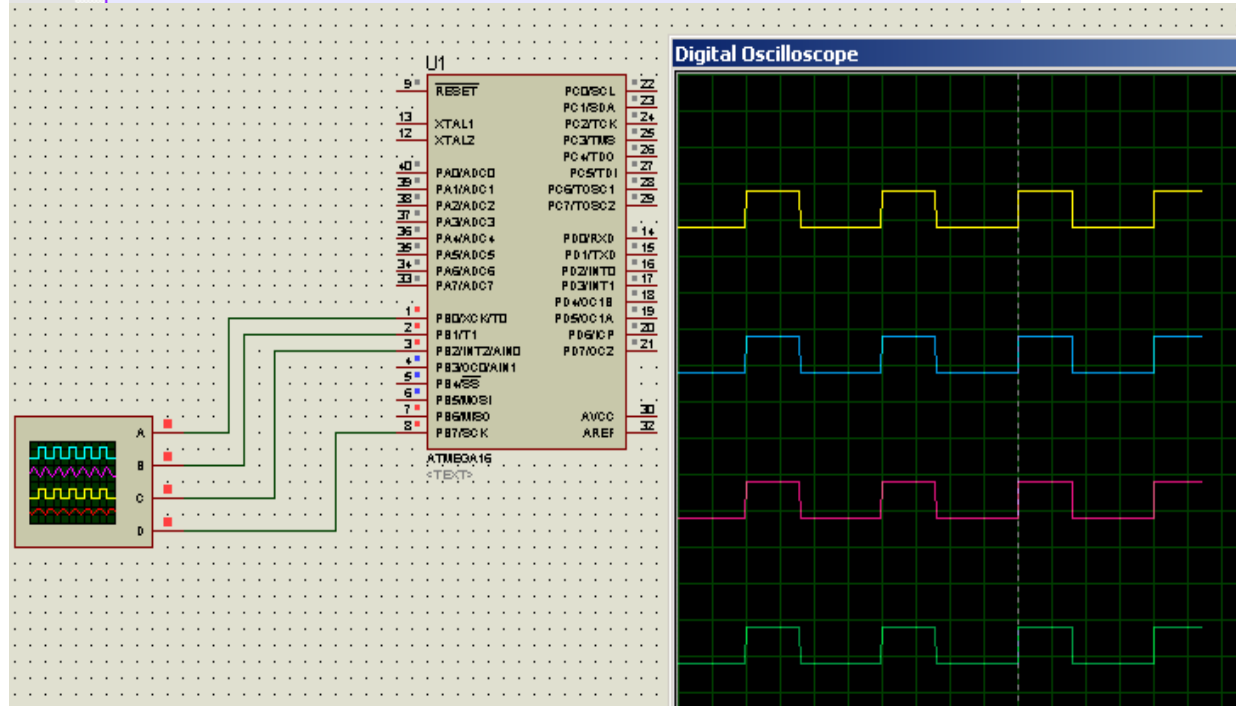
Bu örnekte yukarıdaki PWM tanımından hareketle temelde PWM'in nasıl çalıştığı gösterilmek istendi. Ancak belgenin devamında yapılacak olan uygulamalar ve staj süresince istenen algoritmaların karmaşıklığı göz önüne alındığında bu ve bunun gibi yöntemlerin yeterli olmayacağı belgenin devamında görülecektir.

Şimdilik asıl amaç PWM'in nasıl bir çalışma prensibinin olduğunu bir uygulama ile göstermektir.

```

1  #include <avr/interrupt.h>
2  #include <util/delay.h>
3
4  #define F_CPU 4000000
5  main()
6  {
7  DDRB=0xFF;
8  while(1)
9  {
10     //şimdi mesela ben %40 pwm yazmak istiyorum frekansı da 50hz olsun
11     // nasıl ayarlıcam ne kadar bekletme yapacağımı? kolay =)
12     // şimdi 3-5 hesap yapalım bi kere bizim 1 sn'de 50 defa pwm vermemiz lazım
13     // yani duty cycle dediğimiz olay 1/50 saniye olmalı
14     // %40 pwm vereceksek (0.4)*(1/50) sn = 8 ms boyunca 1 sinyali
15     // (0.6)*(1/50) saniye = 12 ms boyunca da 0 sinyali vermeliyiz.
16     PORTB = 11111111;
17     _delay_ms(8);
18     PORTB = 0;
19     _delay_ms(12);
20     // şimdi yaptık? ameliyan method ile pwm yazdık =P
21 }
22 }
23

```



Şekil 22

Yukarıda Proteus simülasyonunda programın çıktısı görülmektedir

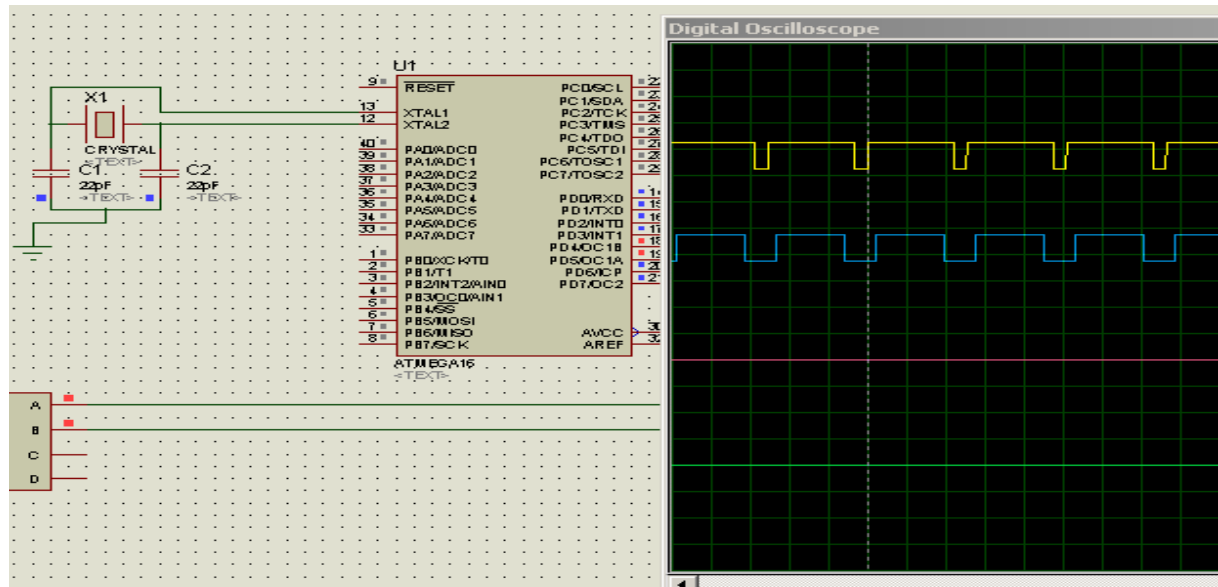
Örnek 3.2: Faz Doğrultmalı PWM

Bu örnekte daha profesyonel bir pwm uygulaması olan faz doğrultmalı PWM üreten bir program yazılmıştır. Bu PWM'i üretebilmek için ATmega16 mikro denetleyicisine ait zamanlayıcı kütükleri(register) kullanılmıştır.

```
1  /*the lengths that i will go to
2  the distance in your eyes */
3  // yukarıdaki şarkıyı dinlemenizi tavsiye ederim
4  //bkz: REM - losing my religion
5
6  #include <avr/interrupt.h>
7
8  main()
9  {
10  DDRD=0xFF; // D portunu çıkış olarak ayarlıyoruz.
11  //çünkü pwm vereceğimiz OCR bacakları orada =)
12  PORTD=0;
13  TCNT0=0;
14  OCR1A=0x00DF; //16 bit gözünürlükte pwm verebiliriz ama 8-bit veriyoruz
15  // 0xFF %100 oluyor 0xDF ise % 239/255 pwm oluyor =)
16  OCR1B=0x00AF;
17  // 0xFF %100 oluyor 0xDF ise % 175/255 pwm oluyor =)
18  // 8 bitlik pwm phrase correct ve non-inverting mode pwm
19  TCCR1A=0xA1; // datasheetten bakınız
20  TCCR1B=0x01; // datasheetten bakınız =)
21  while(1);
22  }
23
```

'delay()' fonksiyonu kullanıldığında mikro denetleyici başka hiçbir iş yapmaz iken bu şekilde ATmega16 ya ait zamanlayıcı modülü kullanılarak PWM üretirken başka işler yapmak da olanaklı hale gelmektedir.

TCCR1A ve TCCR1B denilen "timer-counter control register" üreticinin bilgi sayfasında belirtildiği üzere PWM'in hangi moda üretileceğini saat darbesi ölçeğini ayarlamakta kullanılmaktadır.Saat darbesi kaynağın yani osilatörün frekansının hızında, bu frekansın 1/8 katında 1/64 katında ya da 1/1024 katında pwm üretilebilir.



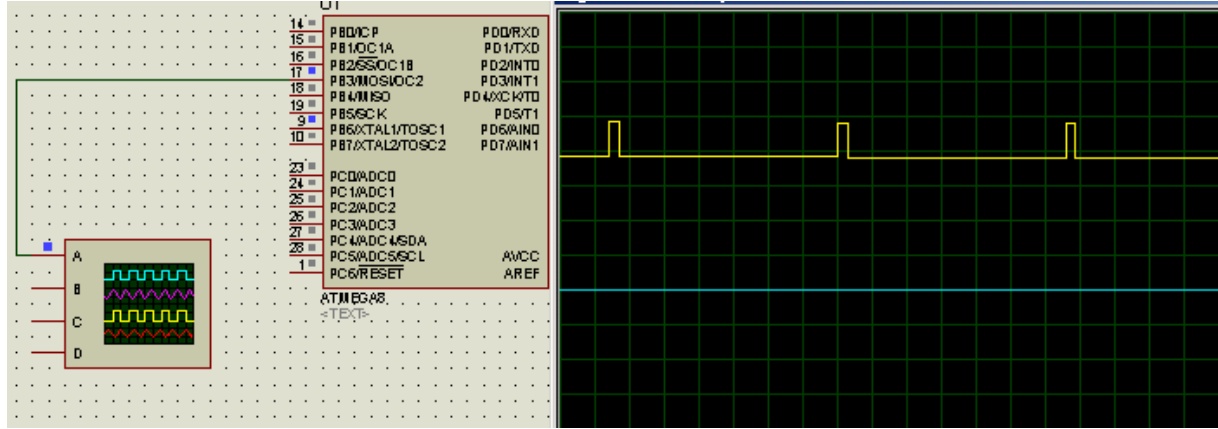
Örnek 3.3:Fast PWM

fast pwm üretiminin amacı kısaca ara frekans değerlerini elde etmektir. Şöyle ki fast pwm ile, faz doğrultmalı pwm modu ile üretebilecek herhangi bir pwm sinyalinin 2 katı frekansta pwm üretilebilir.

```
1  #include <avr/io.h>
2  void main(void)
3  {
4  // Giriş çıkışları ayarlama vakti =)
5  // Port B yi ayarlıyoruz...
6  DDRB=00001110; // PB1,PB2,PB3 çıkış
7  PORTB=0x00;
8  // Port C yi ayarlayalım =)
9  DDRC=0x00;
10 PORTC=0x00;
11 // Port D yi de ayarlıyoruz...
12 PORTD=0x00;
13 DDRD=0x00;
14 // Timer/Counter 0 i hazırlıyoruz
15 // Clock source( sinyal kaynağı): System Clock(sistemin sayacı neyse o mesela 4 mhz kristal)
16 // Clock değeri: Timer 0 yani zamanlayıcı duruyor
17 TCCR0=0x00;
18 TCNT0=0x00;
19 // Timer/Counter 1 i hazırlayalım
20 // Clock kaynağı: Sistem sayacı
21 // Mod: Fast PWM top=ICR1
22 // OC1A çıkışı: terslendi
23 // OC1B çıkışı: terslendi
24 TCCR1A=0b10100010;
25 TCCR1B=0b00011100; // Fosc/256 hızda çalıştırıyoruz =)
26
27 ICR1H=0x02; // kurduğumuz değer TOP= 513 periyodu şu değere ayarlar: 50Hz=20ms => 1ms=25.65(513/20ms)
28 ICR1L=0x01; // pwmin duty cyclesini burdan ayarlıyoruz :)
29 OCR1AH=0x00; // 1.5ms= 38,475= 26H
30 OCR1AL=0x26; // tersledik efendim
31 OCR1BH=0x00;
32 OCR1BL=0x26; // tersledik efendim =)
33 // Timer/Counter 2 yi hazırlayalım
34 // Clock kaynağı: System Clock
35 // Mode: Fast Correct PWM top=FFh
36 // OC2 output: Inverted PWM
37 ASSR=0x00;
38 TCCR2=0x66; // Fosc/256, PWM mod1
39 TCNT2=0xFF; // 8b
40 OCR2=0x0B;
41 // Dış kesmeleri ayarlıyoruz
42 // INT0: kapalı
43 // INT1: kapalı
44 MCUCR=0x00;
45 // Timerlar/Sayaçlar ve kesmeler hazırlanıyor
46 TIMSK=0x18;
47 TIFR=0x00; //
48 // Analog karşılaştırıcı hazırlanıyor
49 // Analog karşılaştırıcı kapalı
50 // Analog karşılaştırıcı girişinin Timer/Counter 1 'den alınması: kapalı
51 ACSR=0x80;
52 SFIOR=0x00;
53 asm("SEI");
54 while (1)
55 {
56 }
57 }
```

Yukarıdaki kodda bir öncekinden farklı olarak çalışmayan modüller kapatıldı. Bu yüzden kod biraz daha uzun sürdü.

Kodun açıklaması üzerinden olduğundan burada açıklama yinelenmedi. Şekil 25'te mikro denetleyicide koşturulan programın çıktısını görülmektedir.



Şekil 23

Örnek 3.3: PWM

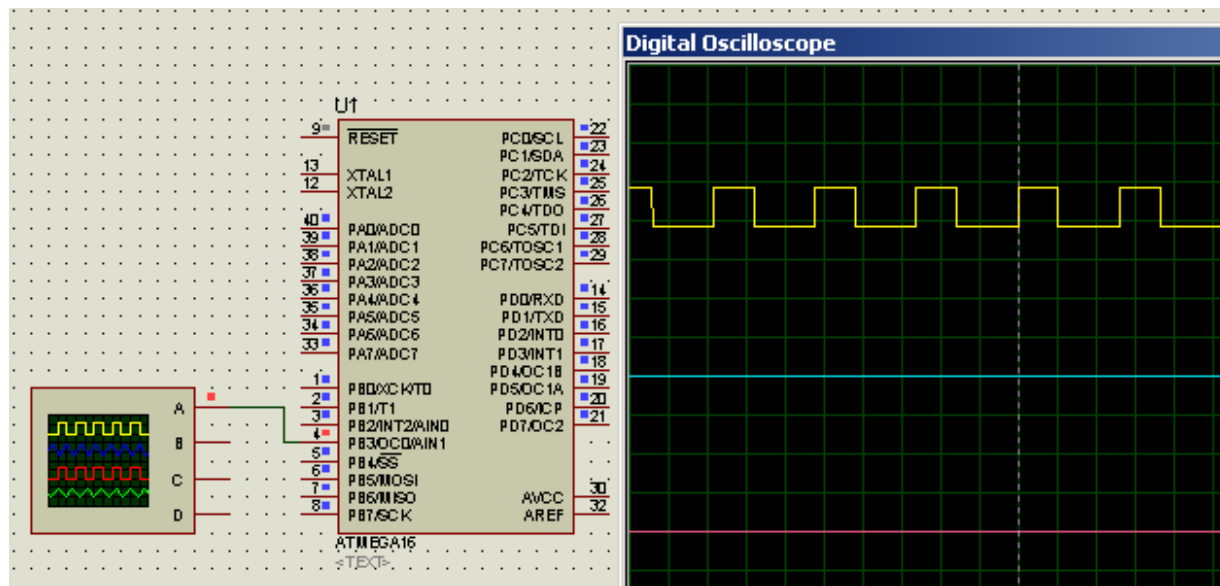
```
1 #include <avr/interrupt.h>
2
3 char i;
4 char yuzde;
5
6 void pwm_wer(int yuzde,unsigned short int prescale)
7 {
8     i = (255 * yuzde)/100;
9     OCRO=i;
10    switch(prescale)
11    {
12    case 1:
13    {
14        TCCRO = 0x61;
15        break;
16    }
17    case 8:
18    {
19        TCCRO = 0x62;
20        break;
21    }
22    case 64:
23    {
24        TCCRO = 0x63;
25        break;
26    }
27    case 256:
28    {
29        TCCRO = 0x64;
30        break;
```

```

31     }
32     case 1024:
33     {
34         TCCRO = 0x65;
35         break;
36     }
37
38     }
39 }
40 void port_hazirla()
41 {
42     //Tüm portları çıkış yapalım =>
43     //DDR = Data Direction Register :)
44     DDRA=0xFF;
45     DDRB=0xFF;
46     DDRC=0xFF;
47     DDRD=0xFF;
48     //Portları temizleyelim =>
49     PORTA=0;
50     PORTB=0;
51     PORTC=0;
52     PORTD=0;
53 }
54
55 int main()
56 {
57     port_hazirla();
58     TCNT0=0; // Zamanlayıcı 0'ı kullanıyoruz =>
59     TCCRO = 0x65;
60     //COM0:1=1 COM0=0; non inverting oluyor... inverting için ikisi de 1 olacak
61     //WGM00=1 Çünkü waveform generation aktif olmalı
62     //CS0=0x01 yani dahili osilatör/256 frekansında çalış...
63     pwm_wer(40,1024);
64     //40 PWM ile dahili osilatörün 1/1024ü kadar frekansta sinyal ürettik =>
65     while(1);
66
67 }

```

Şekil 24'da programın çıktısı görülmektedir.



Şekil 24

20 iş günü süreli bu stajda ATmega 16'da öğrenilen teknik bilgiler ATmega 128 üzerinde de denendi. Daha önce belirtildiği gibi öğrenilen parça parça bilgiler ATmega128 denetleyicisi ile daha bütünlüklü olarak ele alındı.

Bir sonraki örnekte de görüleceği üzere kesmeler ile pwm uygulaması birleştirilerek, bir dış kesme ile pwm frekansını artırıp azaltmak amaçlı bir kod yazılmış ve simülasyonda koşturulan programın çıktısı verilmiştir.

Örnek 3.4: ATmega128 PWM

Kodla ilgili açıklamaların önemli kısmını yukarıda yapıldı. Ancak aşağıda kodları incelendikten sonra birkaç ufak değerlendirme daha yapılacaktır.

```
1  /***** Kütüphaneler *****/
2  #include <avr/interrupt.h>
3  #include <util/delay.h>
4
5  /*****Genel Tanımlamalar*****/
6
7  char i,j,k;
8  char yuzde_A,yuzde_B;
9  char yuzde_1=50;
10 char yuzde_2=50;
11 /***** Prototipler *****/
12 void port_hazirla();
13 void dis_kesme_hazirla();
14 void pwm_hazirla(int prescale_A);
15 void pwm_ver_A(int yuzde_A);
16 void pwm_ver_B(int yuzde_B);
17 void pwm_ver_C(int yuzde_C);
18 void karasimsek(void);
19
20 /***** Dış Kesme Fonksiyonları *****/
21 ISR(INT0_vect)
22 {
23     yuzde_1+=10; //A cikisli pwmi yuzde 10 arttir
24 }
25
26 ISR(INT1_vect)
27 {
28     yuzde_1-=10; //A cikisli pwmi yuzde 10 azalt
29 }
30
31 ISR(INT2_vect)
32 {
33     yuzde_2+=10; //B cikisli pwmi yuzde 10 arttir
34 }
35
36 ISR(INT3_vect)
37 {
38     yuzde_2-=10; //B cikisli pwmi yuzde 10 azalt
39 }
40
```

```

41  /***** Ana Fonksiyon *****/
42  int main()
43  {
44  port_hazirla();
45  dis_kesme_hazirla();
46  pwm_hazirla(1024);
47  while(1)
48  {
49  pwm_ver_A(yuzde_1);
50  pwm_ver_B(yuzde_2);
51  karasimsek();
52  }
53  }
54
55  /***** Port Hazirla *****/
56  void port_hazirla()
57  {
58  DDRA=0xFF;
59  DDRB=0xFF;
60  DDRC=0xFF;
61  DDRD=0x0;
62  DDRE=0xFF;
63  DDRF=0xFF;
64  DDRG=0x1F;
65  PORTA=0;
66  PORTB=0;
67  PORTC=0;
68  PORTD=0;
69  PORTE=0;
70  PORTF=0;
83  /***** Pwm hazirla *****/
84  void pwm_hazirla(int prescale_A)
85  {
86  TCNT1=0;
87  TCCR1A=0xA9;
88  TCCR1B=0x01;
89  switch(prescale_A)
90  {
91  case 1:
92  {
93  TCCR1B = 0x01;
94  break;
95  }
96  case 8:
97  {
98  TCCR1B = 0x02;
99  break;
100 }
101 case 64:
102 {
103 TCCR1B = 0x03;
104 break;
105 }

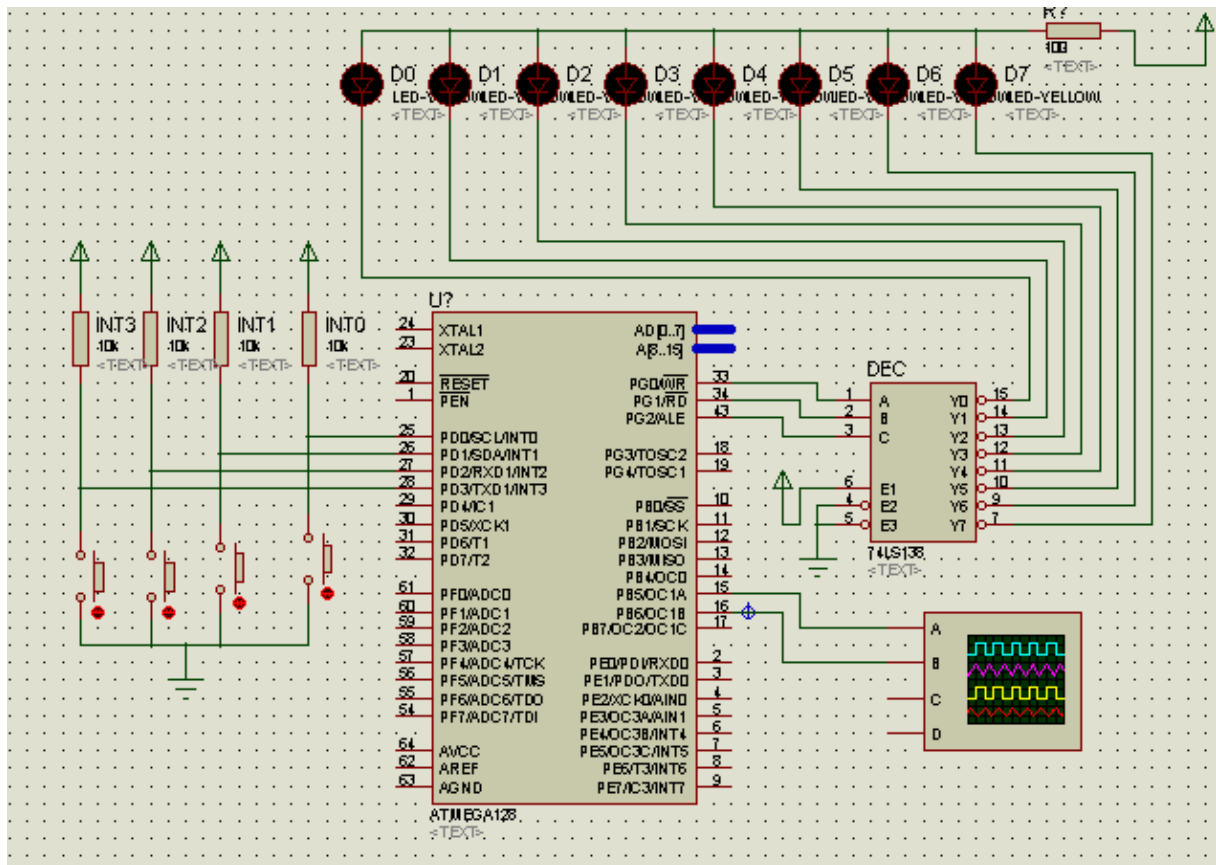
```

```

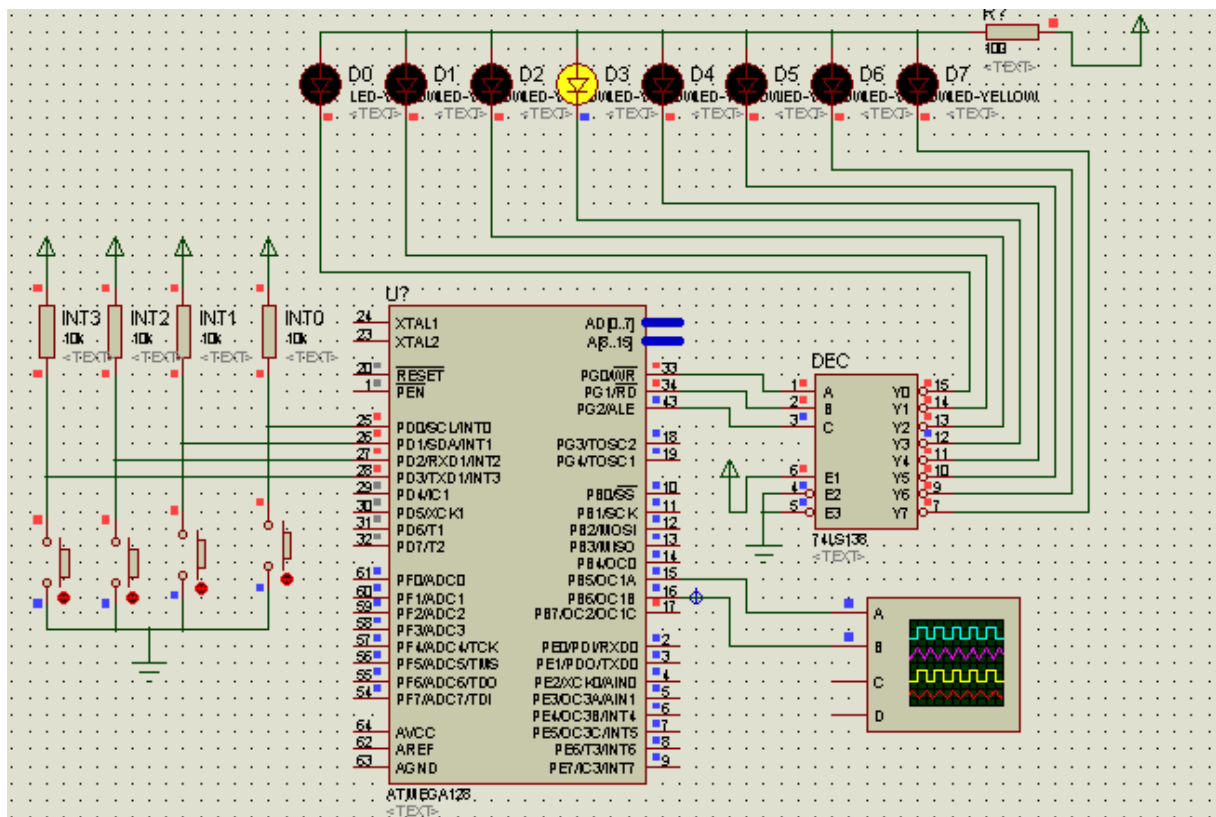
106         case 256:
107         {
108             TCCR1B = 0x04;
109             break;
110         }
111         case 1024:
112         {
113             TCCR1B = 0x05;
114             break;
115         }
116     }
117 }
118
119 /***** OCR1A'dan pwm ver *****/
120 void pwm_ver_A(int yuzde_A)
121 {
122     i = (255 * yuzde_A)/100;
123     OCR1A= i;
124 }
125
126 /***** OCR1B'den pwm ver *****/
127 void pwm_ver_B(int yuzde_B)
128 {
129     j = (255 * yuzde_B)/100;
130     OCR1B= j;
131 }
132
133 /***** Karasimsek *****/
134 void karasimsek(void)
135 {
136     PORTG=1;
137     PORTG=0;
138     _delay_ms(200);
139     for(i=0;i<7;i++)
140     {
141         PORTG+=1;
142         _delay_ms(200);
143     }
144     for(i=0;i<6;i++)
145     {
146         PORTG-=1;
147         _delay_ms(200);
148     }
149 }
150

```

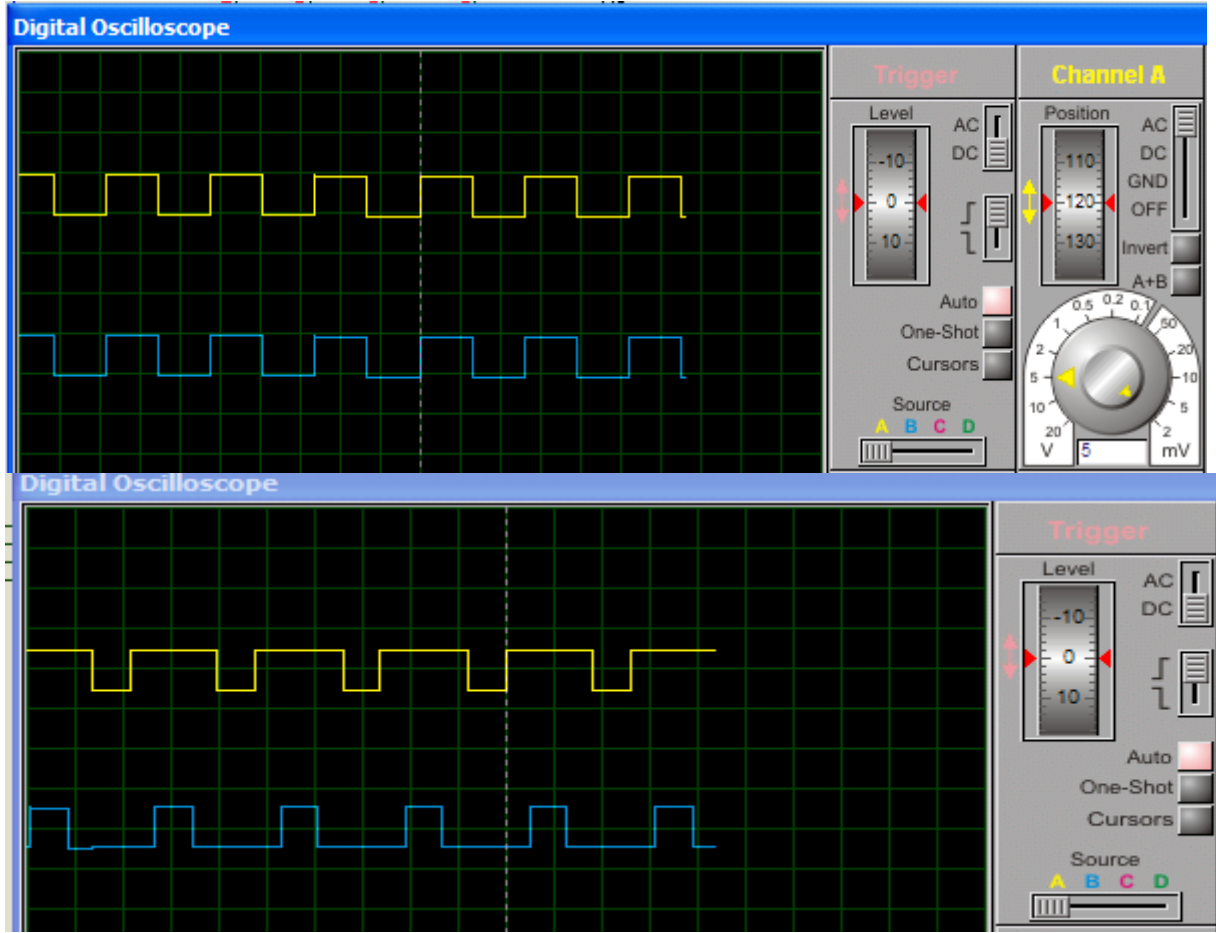
Koda ilişkin devre aşağıda gösterilmiştir.



Şekil 25



Şekil 26



ZAMANLAYICILAR (TIMERS)

Zamanlayıcıya kabaca mikro denetleyicinin kalbindeki saattir denilebilir. Bu saatin hızı, mikrodenetleyicinin osilatör frekansı ile yakından ilişkilidir. Bunu da örneklerimizde daha iyi anlayacağız.

Atmega16’da 8 bitlik 2, 16 bitlik 1 adet zamanlayıcı vardır.

Tablo 7

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tablo 7’de timer0’a ait yazmaç haritası görülmektedir.

Bu haritada görülen bitlerden WGMxx ile başlayan bitler ‘wave generation mode’ (dalga üretim modu) anlamına gelmektedir. Tablo 8’de bir bakıma bu bitlerin ne iş yaptığını özetlenmiştir. ATmega16 ve 128’de zamanlayıcıların bir çok fonksiyonu vardır. bir bitin değeri ile oynanılarak pwm üretimin frekansı değiştirilebilir, genliği değiştirilebilir ve bütün

bunların hemen hepsi birkaç bitte ufak oynalarla gerçekleşir. Uygulamalar yapıldıkça tüm bu bitler daha rahat anlaşılacaktır.

COMxx ile başlayan bitler compare match output mode ifadesinin kısaltılmış halidir. WGMxx'lerin durumlarına göre OC0 pinin davranışını belirlerler.

Tablo 8

Table 38. Waveform Generation Mode Bit Description⁽¹⁾

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Tablo 9

Table 39. Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Tablo 10

Table 40. Compare Output Mode, Fast PWM Mode⁽¹⁾

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (non-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

Tablo 11

Table 41. Compare Output Mode, Phase Correct PWM Mode⁽¹⁾

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

Üreticinin bilgi sayfasından alınmış olan yukarıdaki tablolardan şunlar anlaşılacaktır. Zamanlayıcının nasıl bir dalga üreteceği seçilir. Ardından seçilen dalgaya uygun OC0 pinin davranışı COMxx isimli bitler aracılığı ile belirlenir. Son olarak da bütün bu dalga üretiminin

hangi saat darbesi ile ne kadar sıklıkla yapılacağı CSxx (clock select) bitleri ile aşağıdaki tablodaki gibi belirlenir.

Tablo 12

Table 42. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk _{I/O} /(No prescaling)
0	1	0	clk _{I/O} /8 (From prescaler)
0	1	1	clk _{I/O} /64 (From prescaler)
1	0	0	clk _{I/O} /256 (From prescaler)
1	0	1	clk _{I/O} /1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Yine zamanlayıcı modülüne ait Tablo 13'te bit haritası görülen timer counter register isimli TCNT0 sayıcı uygulamalarında kullanılır.

Tablo 13

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tablo 14

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tablo 14'te ise output compare register isimli OCR0'a ilişkin bit haritası gösterilmiştir. OCR0 kendine atanan değeri sürekli olarak TCNT0 ile karşılaştırır. Bu karşılaştırmalar sonucunda bu ikisinin birbirlerini yakaladıkları nokta ya kesmeye girmenin tetikleyicisi olan ya da dalga üretimini başlatan noktadır. Ve bu noktada olan olaya göre OC0 pinin davranışı belirlenir.

Tablo 15

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tablo 15'te zamanlayıcı modülüne ait kesmelerden sorumlu timer interrupt mask register isimli yazmaç görülmektedir.

Bit 1 - OCIE0: Timer/Counter0 Output Compare Match Interrupt Enable

OCIE0 '1' yapıldığında(SREG'de kesmeler açılmış iken) Timer/Counter0 Compare Match interrupt kesmesi aktif olur. Timer/Counter0'da bir karşılaşma meydana geldiğinde Timer/Counter0 Output Compare Match Interrupt Enable kesmesi meydana gelir.

Bit 0 - TOIE0: Timer/Counter0 Overflow Interrupt Enable

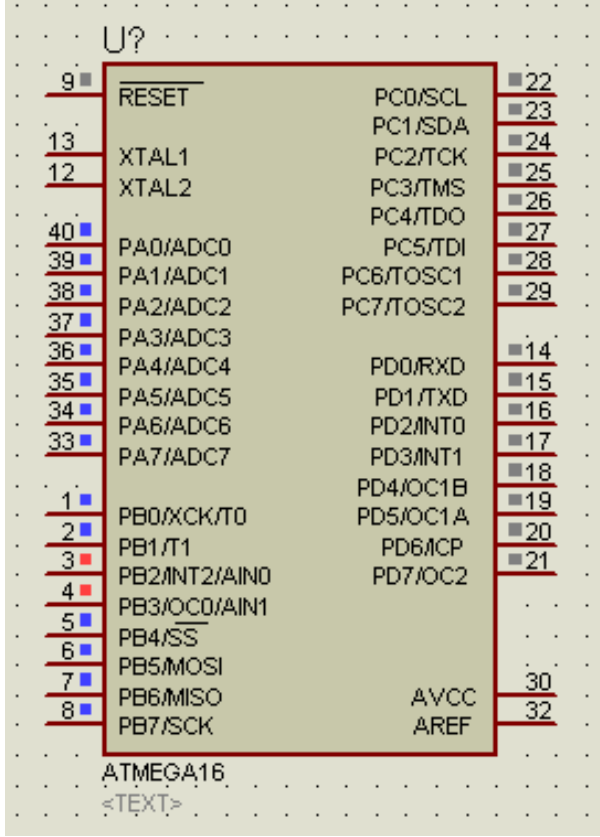
TOIE0 '1' yapıldığında(SREG'de I-biti açılmış iken) Timer/Counter0 Overflow interrupt kesmesi aktif olur. Timer/Counter0 'da taşma meydana geldiğinde Timer/Counter0 Overflow Interrupt Enable kesmesi meydana gelir.

Zamanlayıcılar global saat olarak veya sayıcı (counter) olarak kullanılabilirler. Onun dışında pwm gibi çeşitli uygulamalarda zamanlayıcıları kullanılabilir.

Örnek 4.0: ZAMANLAYICI 0

```
1  #include <avr/interrupt.h>
2  #define F_CPU 4000000UL
3
4  int t=0;
5  void port_hazirla();
6  void zamanlayici_hazirla();
7
8  ISR(TIMERO_OVF_vect)
9  {
10     t+=1;
11     if(t==10000)
12     {
13         PORTB+=1;
14         t=0;
15     }
16     TCNT0=55; //55ten 255'e kadar saymasını istiyoruz.
17     /* Aşağıda zamanlayıcıyı hazırlarken zamanlayıcıyı
18     55ten başlattık, bunun nedeni 55ten 255'e kadar
19     saydırmak istememizdir. 4 mhzde bu 50 us yapar.
20     biz 10000 defada portb'yi bir arttırarak 1/2 saniyede
21     Port B'nin bir artmasını sağladık...
22     */
23 }
24
25 int main()
26 {
27     port_hazirla();
28     SREG |= _BV(7); // Genel kesme aktivasyonu
29     zamanlayici_hazirla();
30     while(1);
31 }
32
33
34 void port_hazirla()
35 {
36     DDRA=0xFF;
37     DDRB = 0xFF;
38     DDRD=0;
39     PORTA=0;
40     PORTB=0;
41     PORTD=0;
42 }
43
44 void zamanlayici_hazirla()
45 {
46     TCNT0 = 55; // Timer başlangıç değeri...
47     TCCR0 = 0x41; // dalga üretimini açtık ve frekans kaynağı seçtik 4MHZ
48     TIMSK = 0x01; // timer taşma kesmesini aktive ettik
49     MCUCR |= _BV(ISC00) | _BV(ISC01) | _BV(ISC10) | _BV(ISC11);
50 }
```

Örneğe ilişkin devre Şekil 29'daki gibidir.



Şekil 27

Kodda görüldüğü üzere bekletme işlemi delay fonksiyonu kullanılmadan zamanlayıcı ile yapıldı. Zamanlayıcı kullanılarak çok hassas değerlerde bekletmeler yapılabilir. Yukarıda bir zamanlayıcı kesmesi kullanıldı. Bu kesmenin adı zamanlayıcı taşma kesmesidir(timer overflow interrupt). Bunun ise ifade ettiği şey şöyle açıklanabilir: Zamanlayıcı 8 bitlik olduğundan en fazla 2^8-1 yani 255 değerine kadar sayabilir. Bu değerden sonra zamanlayıcı taşar ve de sıfırlanır. Zamanlayıcı kesmesinde kesme alt programının sonunda tekrar zamanlayıcıyı 55'ten başlatılmasının sebebi de aslında budur. Şayet orada zamanlayıcı 55'ten başlatılmaz ise programda zamanlayıcı tek bir defa 55'ten başlayacak, sonraki her döngüde 0'dan 255'e kadar sayacaktır ki bu da istenilen bir olay değildir. Kodlardaki kütüklerle(register) ilgili her ayrıntı üreticinin bilgi sayfasında (datasheet) mevcuttur.

Bir önceki örnekte zamanlayıcı aslında zamanlayıcı kesmesi kullanılarak anlatılmaya çalışıldı. Olayın pekişmesi için diğer zamanlayıcılarla da örnekler yapıldı. Bu örneklerde de yine olayı biraz daha olsun somutlaştırmak için kesmelere yer verildi. Farklı zamanlayıcılarla çalışmanın aslında hiçbir zorluk getirmediğini öğrendikten sonra, yine geleneğimizi bozmayıp ATmega16'dan farklı bir Atmel ile bir örnek yapıldı.

Örnek 4.1: ZAMANLAYICI 1

Bir önceki örnekte zamanlayıcı 0 ile bir kod yazıldı. Şimdi de zamanlayıcı 1'i kullanarak bir örnek gerçekleştirilecek.

```
1  #include <avr/interrupt.h>
2  int j=0;
3
4  void zamanlayici_hazirla();
5  void port_hazirla();
6
7  ISR(TIMER1_OVF_vect)
8  {
9      j++;
10     if(j==10000)
11     {
12         PORTD+=1;
13         j=0;
14     }
15 }
16
17 int main()
18 {
19     port_hazirla(); //portları hazırlar
20     zamanlayici_hazirla(); // zamanlayıcı ve kesme hazırlığı
21     while(1);
22 }
23
24 void port_hazirla()
25 {
26     DDRD = 0b11111111;
27     PORTD = 0;
28     DDRB = 0b11111111;
29     PORTB = 0;
30 }
31
32 void zamanlayici_hazirla()
33 {
34
35     SREG = 0b10000000;
36     GICR = 0b00000000;
37     TCCR1A = 0b00000000;
38     TCCR1B = 1;
39     TIMSK = 0x04;
40     // bu sefer zamanlayıcı kendiliğinden Odan başlıyor
41 }
```

Bu örnek için herhangi bir açıklama yapmayacağız. Herhangi bir veri kağıdından, kütüklerle ilgili olan bilgileri bakıldığında örnek rahatlıkla anlaşılabilir. Konuya ilişkin proteus devresi de yine bir önceki örnektekinin aynısıdır. Bu nedenden dolayı ekstra bir devre çizimi yapılmayacaktır.

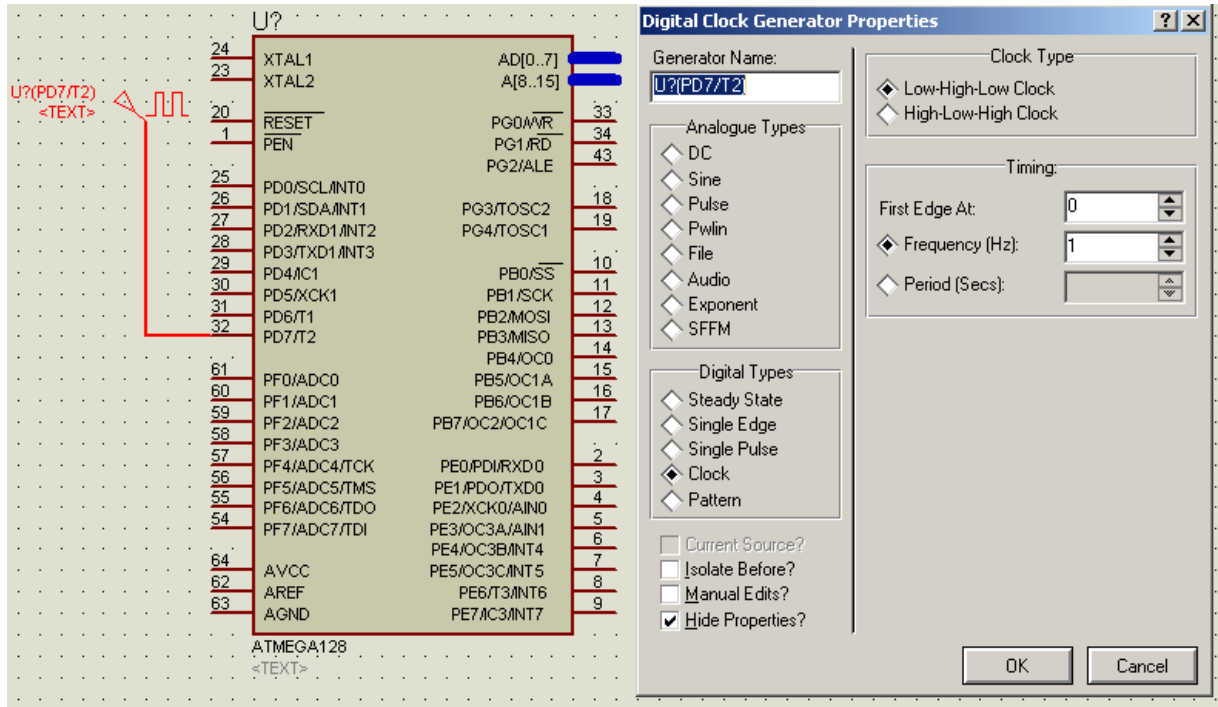
Encoder tarzı sensörlerden veri okumak için ya da sadece gelen saat darbelerini saymak için sayıcıları kullanılır. Aşağıdaki örnek de sayıcılar ile ilgili bir uygulama görülmektedir.

Örnek 4.2:SAYICI (COUNTER)

```
1 #include <avr/interrupt.h>
2 int main()
3 {
4   DDRD=0;
5   DDRF=0xff;
6   PORTF=0;
7   TCCR2=0x17;
8   while(1)
9   {
10    PORTF=TCNT2 ;
11  }
12 }
```

Yukarıdaki kod ile timer2 sayıcısı ile timer2 pinine gelen dalgalar sayılarak sayılan değer PORTF'ye aktarılmaktadır.

Şekil 30'da koda ilişkin devre görülmektedir.



Şekil 28

Burada saat darbesi frekansını 1 hz olarak ayarlanmasının sebebi, dalga sayısını gözlemleyebilmek. Kodda da daha önce belirtildiği üzere saat darbelerinin sayısını PORTF'ye yazdırılmaktadır. Zamanlayıcı 2'nin sayıcı modunu kullanıldığından sayılan sinyal T2 bacağına bağlıdır.

Aşağıda yukarıda yazılan tüm kodları daha bütünlüklü olarak ele alan uygulama görülmektedir.

Zamanlayıcı Bütünleyici (Örnek 4.3)

```
1  /***** Kütüphaneler *****/
2  #include <avr/interrupt.h>
3  #include <util/delay.h>
4  #include <math.h>
5
6  /*****Genel Tanımlamalar*****/
7
8  char i,j,k,l=0;
9  char yuzde_A,yuzde_B;
10 char yuzde_1=50;
11 char yuzde_2=50;
12
13 /***** Prototipler *****/
14 void port_hazirla();
15 void dis_kesme_hazirla();
16 void pwm_hazirla(int prescale_A);
17 void pwm_ver_A(int yuzde_A);
18 void pwm_ver_B(int yuzde_B);
19 void pwm_ver_C(int yuzde_C);
20 void karasimsek(void);
21 void timer_hazirla();
22 void counter_hazirla(void);
23
24 /***** Zamanlayıcı Kesmesi *****/
25 ISR(TIMER1_OVF_vect)
26 {
27     l+=1;
28     if(l==10)
29     {
30         PORTE+=1;
31         l=0;
32     }
33
34     TCNT1H=0;
35     TCNT1L=0;
36 }
37
38 /***** Dış Kesme Fonksiyonları *****/
39 ISR(INT0_vect)
40 {
41     yuzde_1+=10; //A cikisli pwmi yuzde 10 arttir
42 }
43
44 ISR(INT1_vect)
45 {
46     yuzde_1-=10; //A cikisli pwmi yuzde 10 azalt
47 }
```

```

48
49   ISR( INT2_vect)
50   {
51     yuzde_2+=10; //B cikisli pwmi yuzde 10 arttir
52   }
53
54   ISR( INT3_vect)
55   {
56     yuzde_2-=10; //B cikisli pwmi yuzde 10 azalt
57   }
58
59   /***** Ana Fonksiyon *****/
60   int main()
61   {
62     port_hazirla();
63     dis_kesme_hazirla();
64     counter_hazirla();
65     timer_hazirla();
66     pwm_hazirla(1024);
67     while(1)
68     {
69       pwm_ver_A(yuzde_1);
70       pwm_ver_B(yuzde_2);
71       karasimsek();
72       PORTF=(TCNT2%10); // burada sayacı 7 segmente yazdırıyoruz
73     }
74   }
75
76   /***** Port Hazırla *****/
77   void port_hazirla()
78   {
79     DDRA=0xFF;
80     DDRB=0xFF;
81     DDRC=0xFF;
82     DDRD=0x0;
83     DDRE=0xFF;
84     DDRF=0xFF;
85     DDRG=0x1F;
86     PORTA=0;
87     PORTB=0;
88     PORTC=0;
89     PORTD=0;
90     PORTE=0;
91     //PORTF=0;
92     PORTG=0;
93   }
94

```

```

95  /***** Dıř kesme hazırla *****/
96  void dis_kesme_hazirla()
97  {
98      SREG |= _BV(7);
99      EICRA = 0xAA;
100     EICRB = 0xAA;
101     EIMSK=0xFF;
102 }
103
104  /***** Pwm hazırla *****/
105  void pwm_hazirla(int prescale_A)
106  {
107     TCNT1=0;
108     TCCR1A=0xA9;
109     TCCR1B=0x01;
110     switch(prescale_A)
111     {
112     case 1:
113     {
114         TCCR1B = 0x01;
115         break;
116     }
117     case 8:
118     {
119         TCCR1B = 0x02;
120         break;
121     }
122     case 64:
123     {
124         TCCR1B = 0x03;
125         break;
126     }
127     case 256:
128     {
129         TCCR1B = 0x04;
130         break;
131     }
132     case 1024:
133     {
134         TCCR1B = 0x05;
135         break;
136     }
137     }
138 }
139
140  /***** OCR1A'dan pwm ver *****/
141  void pwm_ver_A(int yuzde_A)
142  {
143     i = (255 * yuzde_A)/100;
144     OCR1A= i;
145 }

```

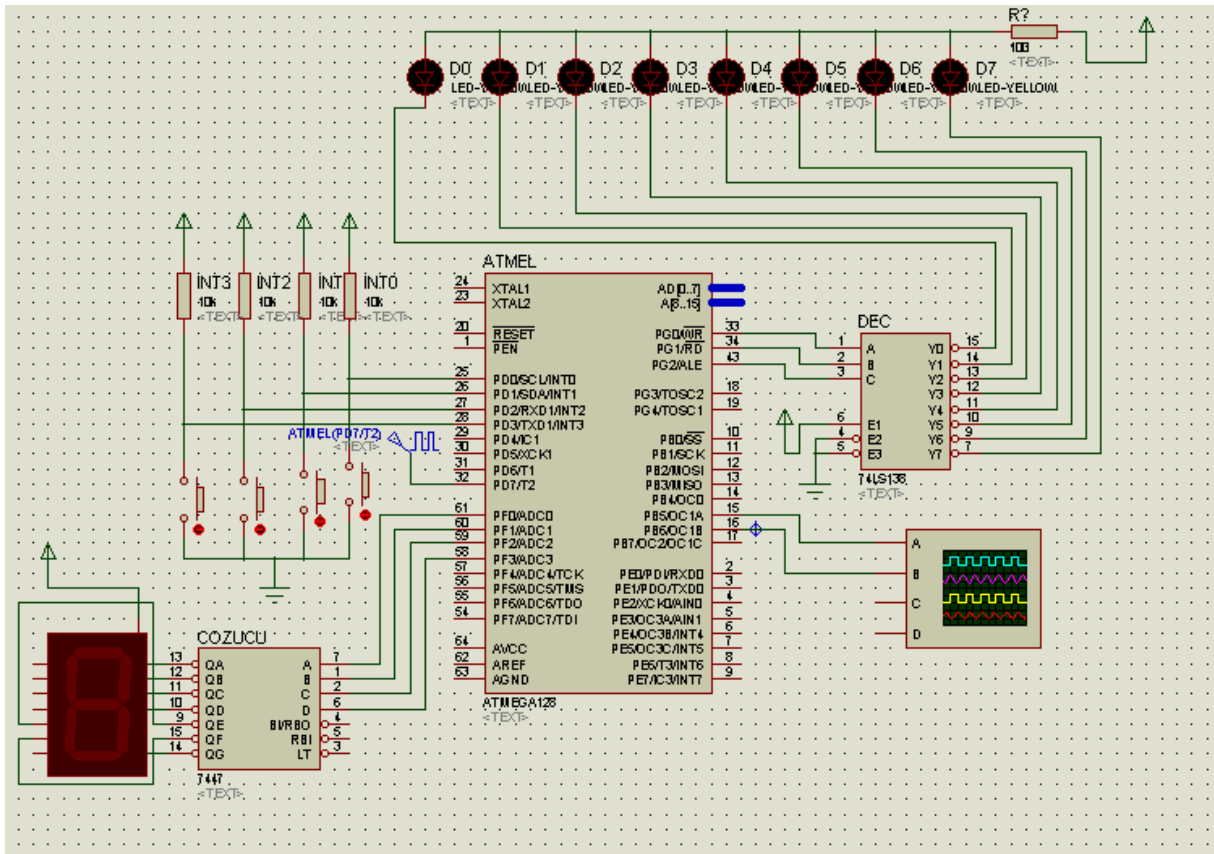
```

146
147 /***** OCR1B'den pwm ver *****/
148 void pwm_ver_B(int yuzde_B)
149 {
150     j = (255 * yuzde_B)/100;
151     OCR1B= j;
152 }
153
154 /***** Karasimsek *****/
155 void karasimsek(void)
156 {
157     PORTG=1;
158     PORTG=0;
159     _delay_ms(200);
160     for(i=0;i<7;i++)
161     {
162         PORTG+=1;
163         _delay_ms(200);
164     }
165     for(i=0;i<6;i++)
166     {
167         PORTG-=1;
168         _delay_ms(200);
169     }
170 }
171
172 /***** Sayıcı Hazırla *****/
173 void counter_hazirla(void)
174 {
175     TCCR2 |= _BV(CS20) | _BV(CS21) | _BV(CS22) | _BV(COM20);
176 }
177
178 /***** Timer Hazırla *****/
179 void timer_hazirla()
180 {
181     TCCR1B |= _BV(CS12) | _BV(CS10); // osilatör/1024
182     TIMSK = 0x04; // overflow kesmesi açık
183 }
184

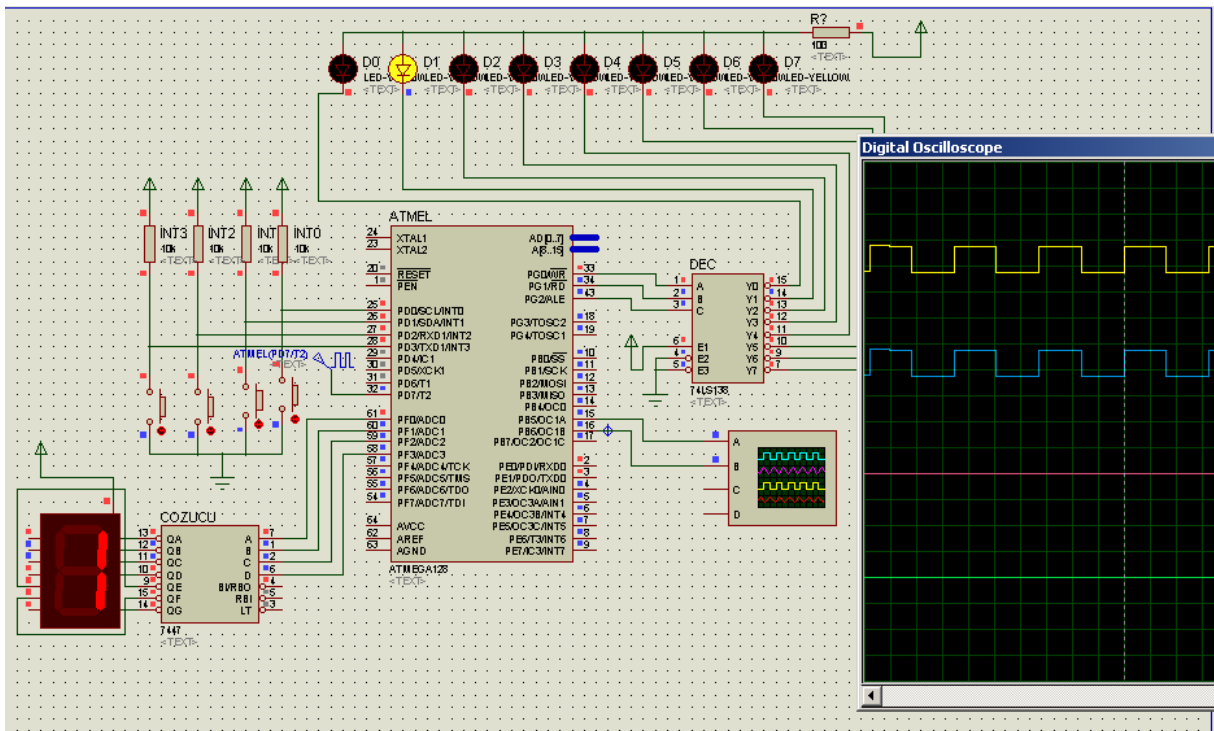
```

bu uygulamada T' bacağına gelen saat darbeleri sayılarak elde edilen deęer 10 modunda 7 segment göstergeye(display) yazdırılmaktadır. Bunun dıřında bir zamanlayıcı global saat olarak kullanılmakta, bir dięer bir zamanlayıcı ise PWM üretmek için kullanılmaktadır. Ayrıca dıř kesmelerle PWM genlięi de arttırılıp azaltılabilmektedir.

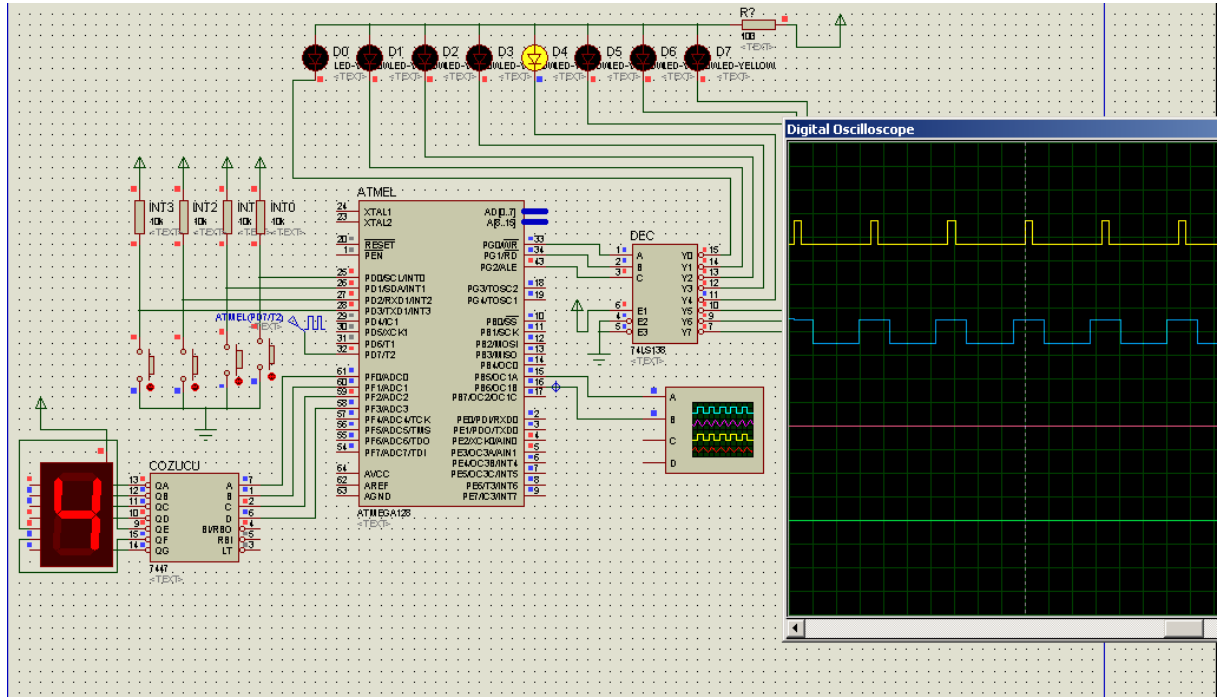
řekil 31,32ve 33'te Proteus devresi ve simülasyon sonuçları yer almaktadır.



Şekil 29



Şekil 30



Şekil 31

SPI (Serial Phreperhal Interface - Seri Çevresel Arayüz)

SPI çift yönlü seri arabirimdir, birden fazla master ve slave'leri idare edebilmek ve belirli bir veri yoluna bağlamak için tasarlanmıştır. Belirli bir veri aktarımı süresince sadece tek bir master ve tek bir slave arayüz üzerinden haberleşebilir. Veri aktarımı boyunca master her zaman slave'e verinin bir baytını gönderir ve slave de her zaman master'a verinin bir baytını gönderir.

Yapacağımız uygulamalarda master mode operation ve slave mode operation kullanım seçeneklerine ilişkin kodları ve simülasyonları görebileceksiniz. SPI modülü de neticede USART gibi bir seri haberleşme modülüdür. Ancak bazı yönleriyle USART'tan çok daha avantajlıdır. Özellikle birden çok donanımı kontrol edebilme imkanı sunması dolayısıyla SPI modülünün yaygın kullanım alanları vardır.

SPI Master Mode Operation (Örnek 5.0)

Bu uygulamamızı Atmega128 ile gerçeklemeye çalışacağız. Uygulama ile ilgili daha geniş bilgiye, ilgili mikrodenetleyicinin veri kağıtlarını inceleyerek ulaşabilirsiniz.

```

1 #include <avr/interrupt.h>
2 #define DDR_SPI DDRB //SPI veri yönü
3 #define DD_MOSI DDB2 //Master output slave input pini
4 #define DD_SCK DDB1 // clock kaynağı
5
6 void spi_master_hazirla();
7 void spi_master_yolla(char cData);
8
9 int main()
10 {
11     spi_master_hazirla();
12     spi_master_yolla(0x50);
13     spi_master_yolla(0x70);
14     while(1);
15 }
16

```

```

17 void spi_master_hazirla()
18 {
19     DDR_SPI = (1<<DD_MOSI) | (1<<DD_SCK); //MOSI ve SCK çıkış
20     SPCR |= _BV(SPE); //SPI enable
21     SPCR |= _BV(MSTR); // Master Modu
22     //SPCR |= _BV(SPRO) | _BV(SPR1);
23 }
24
25 void spi_master_yolla(char cData)
26 {
27     SPDR = cData;
28     /* Verinin yollanmasını bekle */
29     while (!(SPSR & (1<<SPIF)))
30     ;
31 }

```

Gördüğümüz üzere kodumuzda SPI'dan master moda 0x50 ve 0x70 verilerini yolladık. Proteusta karşılaştığımız sonuç aşağıdaki gibidir:

The image shows a Proteus simulation of an ATMEGA128 microcontroller. The pin connections are as follows:

Pin	Signal	Destination
11	PB0/SS	SS
12	PB1/SCK	SCK
13	PB2/MOSI	DOUT
14	PB3/MISO	DIN
16	PB5/OC1A	TRIG

The SPI Debug Monitor window shows the following data:

Sequence	Start Time	End Time	Data
2.875us	7.125us	?? ??	
2.875us	7.125us	50 70	

SPI hata ayıklayıcısı sayesinde aktarılan verinin yanı sıra veri aktarım hızı gibi bazı önemli bilgilere de ulaşabiliriz. Yine bu hata ayıklayıcısı seri haberleşmede, monitör modunda, master moda ya da slave modunda kullanabiliriz. SPI modülü ile ilgili kodlarımızda çok ayrıntıya yer vermeyeceğiz.

USART (Universal Synchronous - Asynchronous serial Receiver and Transmitter)

Senkron veya asenkron seri iletişim modüllerinden biridir. USART, seri haberleşmede en yaygın kullanıma sahip haberleşme modüllerinden biridir. Tıpkı SPI'da olduğu gibi USART'ta da master veya slave moda haberleşme yapabiliriz. Bunun yanı sıra asenkron haberleşme için de USART çok tercih edilen bir modüldür. Yine USART ile birkaç tür kesme uygulaması yapılabilmektedir. Aşağıda USART'a dair birtakım uygulamalar göreceksiniz.

ÖRNEK 6.0 (Asenkron veri yollama)

```
1  /*-----
2  |-----HEADER-----
3  |-----*/
4  #include <avr/interrupt.h>
5  #include <util/delay.h>
6
7  /*-----
8  |-----TANIMLA-----
9  |-----*/
10
11 #define Receive_DDR  DDRC
12 #define Receive_PORT  PORTC //ya JTAG'ı kapatmalıyız ya da bunu PORTC ya da PORTB yapmalıyız
13 /*-----
14 |-----FONKSİYONLAR-----
15 |-----*/
16
17 void Uart_hazirla( unsigned char baudrate );
18 unsigned char Byte_al( void );
19 void Byte_yolla( unsigned char veri );
20 void Port_hazirla(void);
21 void JTAG_kapat();
22
23 /*-----
24 |-----MAIN-----
25 |-----*/
26
27 int main()
28 {
29     JTAG_kapat();
30     Port_hazirla();
31     Uart_hazirla(51); /* baudrate = 2400 bps (8MHz crystal ile)*/
32     while(1){
33         Byte_yolla('i');
34         Byte_yolla('t');
35         Byte_yolla('u');
36         Byte_yolla(' ');
37         _delay_ms(100);
38     }
39 }
40
41 /*-----
42 |-----USART hazırlama fonksiyonları-----
43 |-----*/
44
```

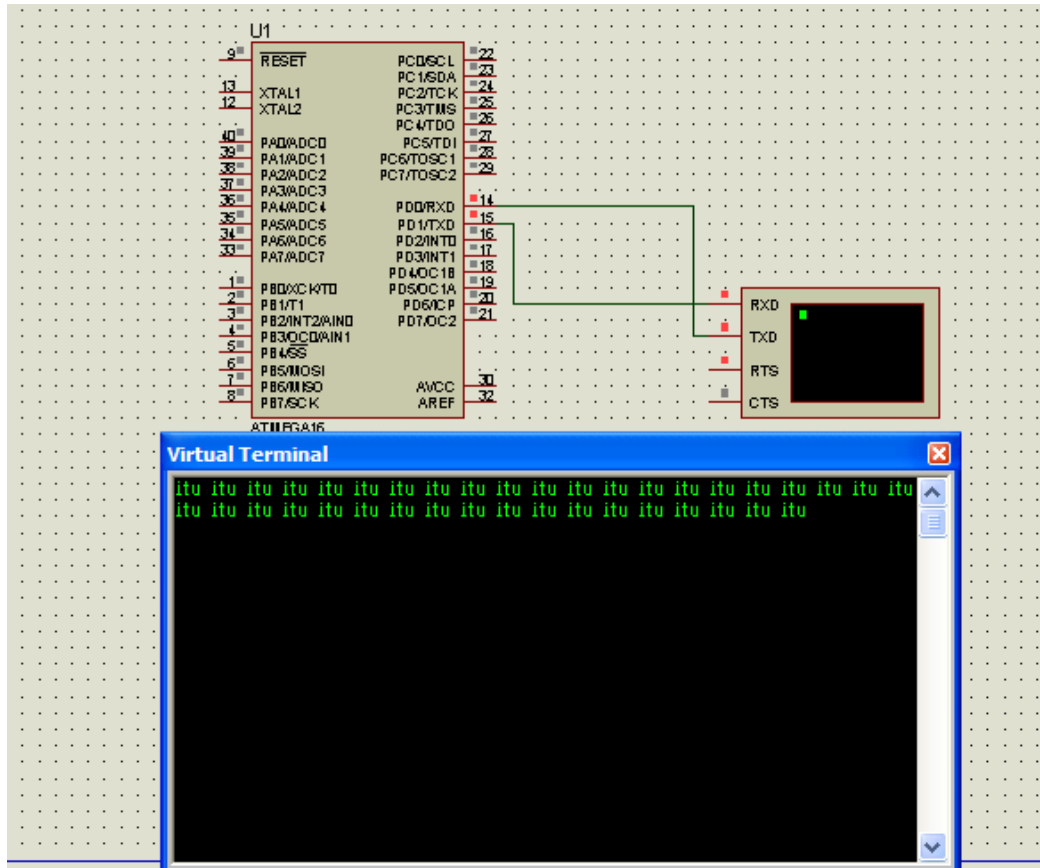
```

45 void Uart_hazirla( unsigned char baudrate )
46 {
47     UBRRL = baudrate; /* boud rate ayarla*/
48     UCSRB = (UCSRB | _BV(RXEN) | _BV(TXEN) ); /* UART aç alıcı ve verici yi aç (r.t)*/
49 }
50 /*-----
51 -----UART okuma fonksiyonları-----
52 -----*/
53 unsigned char Byte_al( void )
54 {
55     while ( !(UCSRA & (_BV(RXC))) ); /* veinin gelmesini bekle */
56     return UDR; /* gelen veriyi döndür */
57 }
58
59 /*-----
60 -----UART yazma fonksiyonları-----
61 -----*/
62 void Byte_yolla( unsigned char veri )
63 {
64     while ( !(UCSRA & (_BV(UDRE))) ); /* Veri gelmiyorsa...*/
65     UDR = veri; /* veri yollamayı başlat */
66 }
67
68
69
70 void Port_hazirla(void)
71 {
72     Receive_DDR=0xFF; //portu çıkış yap
73     Receive_PORT=0xFF; //başta her pini lojik 1 yap
74 }
75
76 /*-----
77 -----JTAG DEAKTİVASYON-----
78 -----*/
79 void JTAG_kapat ()
80 {
81     MCUCSR |=_BV(JTD) ;
82 }
83

```

Yukarıdaki koda bir asenkron haberleşme örneği yaptık. Koda ilişkin devreyi ve simülasyonu aşağıda bulabilirsiniz. Koda bulunan baudrate kavramı bit/sn cinsinden veri aktarım hızı olup, bu kavram asenkron haberleşmenin temelinde yatan öğelerden sayılmaktadır.

Yukarıda gördüğümüz üzere seri port aracılığı ile haberleşeceğimiz makineye ya da denetleyiciye, “itü” harflerini göndermiş olduk. Bu harfleri gözlemleyebilmek için Proteus programındaki Virtual Terminal’i kullanacağız. Virtual Terminal’i kullanmak için edit properties kısmında baudrate’i doğru ayarlamamız gerekmektedir. Biz programda baudrate’i 8 mhz kristal için 2400 seçtiğimizden, visual terminali de 2400 baudrate oranında ayarladık.



ÖRNEK 6.1 (Asenkron haberleşme)

Bu örneğimizde 2 adet mikrodenetleyicinin haberleşme hikayesini konu edinen bir uygulamaya yer vereceğiz. Bunlardan bir tanesi, yani veri basacak olan denetleyici yukarıdaki uygulamadaki kodu yollayacak. Diğer mikrodenetleyici ise veri geldiğini anladığında gelen veriyi alıp yollayacak.

```

28  /*-----*/
29  /*-----MAIN-----*/
30  /*-----*/
31
32  int main()
33  {
34      JTAG_kapat(); // JTAG debugger kapalı
35      Port_hazirla(); // portlar ayarlandı
36      Uart_hazirla( 51 ); /* baudrate = 9600 bps (8MHz crystal ile) */
37      kesme_hazirla(); // rx tamamlanma kesmesi ve global kesme aktivasyonu
38      while(1);
39  }
40
41
42  /*-----*/
43  /*-----UART hazırlama fonksiyonları-----*/
44  /*-----*/
45
46  void Uart_hazirla( unsigned char baudrate )
47  {
48      UBRRL = baudrate; /* boud rate ayarla*/
49      UCSRB = (UCSRB | _BV(RXEN) | _BV(TXEN) ); /* UART aç alıcı ve verici yi aç (r.t)*/
50  }

```

```

51  /*-----
52  -----UART okuma fonksiyonları-----
53  -----*/
54  unsigned char Byte_al( void )
55  {
56  while ( !(UCSRA & (_BV(RXC))) ); /* veinin gelmesini bekle */
57  return UDR; /* gelen veriyi döndür */
58  }
59
60  /*-----
61  -----UART yazma fonksiyonları-----
62  -----*/
63  void Byte_yolla( unsigned char veri )
64  {
65  while ( !(UCSRA & (_BV(UDRE))) ); /* Veri gelmiyorsa... */
66  UDR = veri; /* veri yollamayı başlat */
67  }
68
69
70
71  void Port_hazirla(void)
72  {
73  DDRB=0xFF;
74  PORTB=0;
75  Receive_DDR=0xFF; //portu çıkış yap
76  Receive_PORT=0xFF; //başta her pini lojik 1 yap
77  }
78
79  /*-----
80  -----JTAG DEAKTİVASYON-----
81  -----*/
82  void JTAG_kapat(void)
83  {
84  MCUCSR |=_BV(JTD); //JTAG disable
85  }
86
87  /*-----
88  -----KESME HAZIRLAMA-----
89  -----*/
90
91  void kesme_hazirla(void)
92  {
93  SREG |=_BV(7); // genel kesme aktivasyonu
94  UCSRB |=_BV(RXCIE); // reciver complete kesmesi
95  }
96  }

```

Şimdi de koda ilişkin proteus simülasyonunu görelim. Aşağıda göreceğiniz devrede birinci denetleyiciden alınan veri, ikinci denetleyici tarafından algılanarak virtual terminale aktarılmaktadır.


```

26  /*-----
27  -----USART KESMESİ-----
28  -----*/
29
30  ISR(USART0_RX_vect)
31  {
32      byte_yolla(TCNT2);
33      t+=1;
34      if(t==9600)
35      {
36          PORTB+=1;
37          t=0;
38      }
39  }
40
41
42  /*-----
43  -----MAIN-----
44  -----*/
45
46  int main()
47  {
48
49      jtag_kapat();
50      port_hazirla();
51      kesme_hazirla();
52      uart_hazirla(103); //baud = 9600 bit/sn , 16 mhz osilator
53      counter_hazirla();
54      Receive_PORT=byte_al();
55      byte_al(); /* gelen veriyi al*/
56      while(1)
57      {
58
59      }
60
61  }
62  /*-----
63  -----Jtag kapat-----
64  -----*/
65
66
67  void jtag_kapat()
68  {
69      MCUCSR |= _BV(JTD);
70  }

```

```

71  /*-----
72  |-----Port hazirla-----
73  |-----*/
74
75  void port_hazirla()
76  {
77      DDRB=0xFF;
78      DDRD=0b00001011;
79      PORTD=0;
80      DDRF=0xFF;
81      PORTF=0;
82      Receive_DDR=0xFF; //portu çıkış yap
83      Receive_PORT=0xFF; //başta her pini lojik 1 yap
84  }
85  /*-----
86  |-----Kesme hazirla-----
87  |-----*/
88  void kesme_hazirla()
89  {
90      SREG |= _BV(7);
91      UCSROB |= _BV(RXCIE0);
92  }
93  /*-----
94  |-----UART hazirla-----
95  |-----*/
96
97  void uart_hazirla(unsigned char baudrate)
98  {
99      UBRR0H = 0;
100     UBRR0L = baudrate;           /* boud rate ayarla*/
101     UCSROB = (UCSROB | _BV(RXEN0) | _BV(TXEN0) ); /* UART aç alıcı ve verici yi aç (r.t)*/
102  }
103  /*-----
104  |-----UART okuma fonksiyonları-----
105  |-----*/
106  unsigned char byte_al( void )
107  {
108     while ( !(UCSROA & (_BV(RXC0))) ); /* veinin gelmesini bekle */
109     return UDRO; /* gelen veriyi döndür */
110  }
111
112  /*-----
113  |-----UART yazma fonksiyonları-----
114  |-----*/
115  void byte_yolla( unsigned char veri )
116  {
117     while ( !(UCSROA & (_BV(UDRE0))) ); /* Veri gelmiyorsa... */
118     UDRO = veri; /* veri yollamayı başlat */
119  }
120

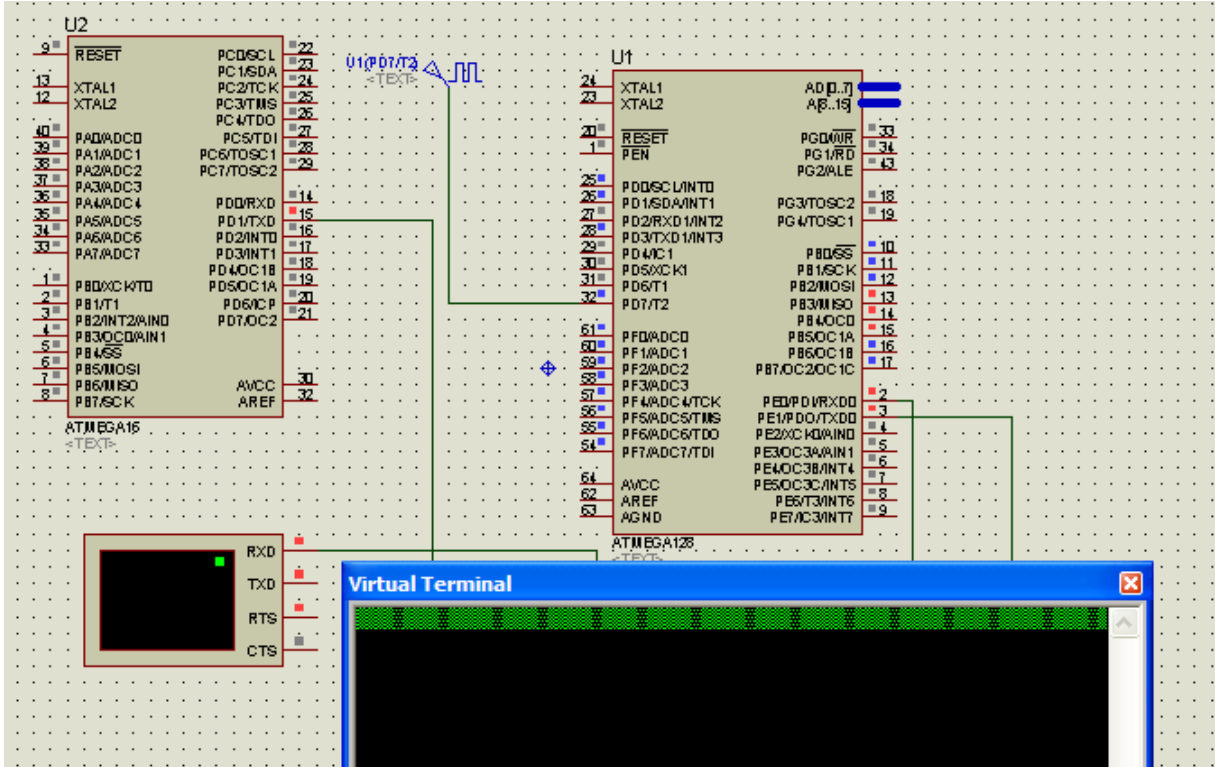
```

```

121  /*-----
122  -----COUNTER HAZIRLA-----
123  -----*/
124
125  void counter_hazirla(void)
126  {
127      TCCR2 |= _BV(CS20) | _BV(CS21) | _BV(CS22) | _BV(COM20);
128  }

```

Kodumuzda sayaçtan okuduğumuz pwm değerini, diğer denetleyiciden herhangi bir veri geldiğinde virtual terminale aktarıyoruz. Kodun simülasyonu aşağıdaki gibidir.



ÖRNEK 6.3 (USART TXC kesmesi)

```

1  /*-----
2  -----HEADER-----
3  -----*/
4  #include <avr/interrupt.h>
5  #include <util/delay.h>
6  int i=0,j=0;
7  /*-----
8  -----TANIMLA-----
9  -----*/
10 #define Receive_DDR  DDRC
11 #define Receive_PORT  PORTC

```

```

12  /*-----
13  |-----FONKSİYONLAR-----
14  |-----*/
15  void Uart_hazirla( unsigned char baudrate );
16  unsigned char Byte_al( void );
17  void Byte_yolla( unsigned char veri );
18  void Port_hazirla(void);
19  void JTAG_kapat();
20  void Kesme_hazirla();
21  /*-----
22  |-----Kesme-----
23  |-----*/
24  ISR(USART_TXC_vect)
25  {
26  |     PORTB+=1;
27  |     _delay_ms(100);
28  | }
29  /*-----
30  |-----MAIN-----
31  |-----*/
32  int main()
33  {
34  |     JTAG_kapat();
35  |     //Port_hazirla();
36  |     Kesme_hazirla();
37  |     Uart_hazirla(51); /* baudrate = 2400 bps (8MHz crystal ile)*/
38  |     char dizi[]={ 'O', 'Z', 'E', 'N', ' ', 'O', 'Z', 'K', 'A', 'Y', 'A' };
39  |     for(i=0;i<11;i++)
40  |     {
41  |         Byte_yolla(dizi[i]);
42  |         for(j=0;j<10;j++);
43  |         UCSRA |= _BV(TXC);
44  |     }
45  |     while(1);
46  | }
47  /*-----
48  |-----UART hazirlama fonksiyonları-----
49  |-----*/
50  void Uart_hazirla( unsigned char baudrate )
51  {
52  |     UBRR1L = baudrate; /* boud rate ayarla*/
53  |     UCSRB = (UCSRB | _BV(RXEN) | _BV(TXEN) ); /* UART aç alıcı ve verici yi aç (r.t)*/
54  | }
55  /*-----
56  |-----UART okuma fonksiyonları-----
57  |-----*/
58  unsigned char Byte_al( void )
59  {
60  |     while ( !(UCSRA & (_BV(RXC))) ); /* veinin gelmesini bekle */
61  |     return UDR; /* gelen veriyi döndür */
62  | }

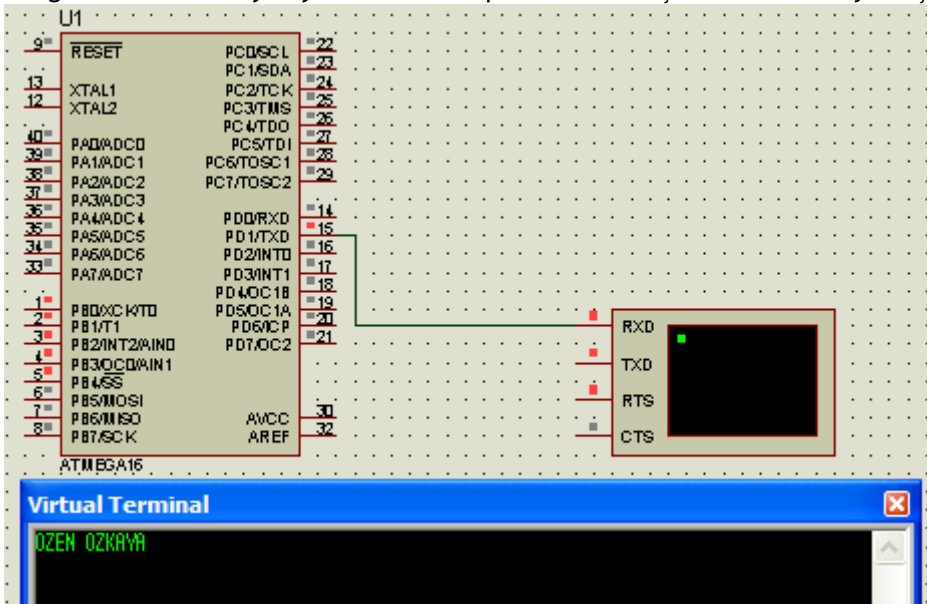
```

```

63  /*-----
64  -----UART yazma fonksiyonları-----
65  -----*/
66  void Byte_yolla( unsigned char veri )
67  {
68  while ( !(UCSRA & (_BV(UDRE))) );           /* Veri gelmiyorsa... */
69  UDR = veri; /* veri yollamayı başlat */
70  }
71  void Port_hazirla(void)
72  {
73  Receive_DDR=0xFF; //portu çıkış yap
74  Receive_PORT=0xFF; //başta her pini lojik 1 yap
75  }
76  /*-----
77  -----JTAG DEAKTİVASYON-----
78  -----*/
79  void JTAG_kapat ()
80  {
81  MCUCSR |=_BV(JTD) ;
82  }
83  /*-----
84  -----KESME HAZIRLIĞI -----
85  -----*/
86  void Kesme_hazirla()
87  {
88  UCSRB |=_BV(TXCIE) ;
89  SREG |=_BV(7) ;
90  }
91

```

Örnekte veri tamamlanma kesmesi ile PORTB'nin 1 artmasını sağladık. Onun haricinde dizi içine attığımız "özen özkaya" yazısını da seri porttan okumuş olduk. Simülasyon aşağıdaki gibidir.



EKSTRA ÖRNEKLER:

Örnek 3.1 (Grafik LCD)

Bu örneğimizde grafik lcd kullanacağız. Üst düzey uygulama olarak gösterilen grafik lcd sürme işleminin aslında çok da zor olmadığını fark edeceksiniz. Grafik LCD sürmek ciddi performans gerektiren bir iş olduğu için bu uygulamamızı 8051 içeren bir atmel ile yapacağız. Kütüphaneyi uyarlayarak, aynı işlemleri farklı denetleyiciler için de tekrarlayabiliriz. Bu uygulamamızda derleyici olarak keil microvision kullanacağız. Uygulamaya ilişkin kodlar aşağıdaki gibidir.

```
/******  
/* KS0108 G_LCD.h */  
/******  
// Grafik lcd kütüphanesi  
#define Acik 1  
#define Kapali 0  
/* Uç Tanımlamaları */  
#define GLCD_CS1 P1_0 // LCD'nin Sol kısmını seçme ucu '0' aktif  
#define GLCD_CS2 P1_1 // LCD'nin Sağ kısmını seçme ucu '0' aktif  
#define GLCD_DI P1_2 // Register Select ucu (komut mu veri mi)  
#define GLCD_RW P1_3 // Oku/Yaz ucu  
#define GLCD_E P1_4 // Yetki ucu  
#define GLCD_RST P1_5 // Reset ucu  
/* Gecikme (Delay) Fonksiyonu */  
void Gecikme (long int sure)  
{  
int i;  
for (i=0;i<sure;i++) ;  
}  
/* Grafik LCD'ye int Deger Yazdirma Fonksiyonu */  
// GLCD'nin seçilen kismına (64 piksellik) Veri bilgisini aktar //  
GLCD_int_Yaz (bit sec, int Veri)  
{  
if (sec != 0) // GLCD'nin kullanılacak kismini sec  
GLCD_CS1=0;  
else  
GLCD_CS2=0;  
GLCD_RW=0; // GLCD yaz modunda  
GLCD_E=1; // islemin yetkilendirilmesi için düşen kenar ayari  
P2=Veri; // GLCD data girislerine veri bilgisini aktar  
Gecikme(2);  
GLCD_E=0; // yetki için düşen kenar saglandi  
GLCD_CS1=1; // GLCD'nin seçili çizgilerini resetle  
GLCD_CS2=1;  
}  
/* Grafik LCD'den int Deger Okuma Fonksiyonu */  
// GLCD'nin seçilen kismından (64 piksellik) okuma yapilir  
// ve okunan deger oku degiskeni ile fonksiyona döndürülür  
int GLCD_int_Oku (bit sec)  
{  
int oku; // GLCD'den okunan degerin tutulacagi degisken  
if (sec !=0) // GLCD'nin kullanılacak kismini sec  
GLCD_CS1=0;  
else  
GLCD_CS2=0;  
P2=0xFF; // P2 portunu giris moduna kur  
GLCD_RW=1; // GLCD oku modunda  
GLCD_E=1; // islemin yetkilendirilmesi için düşen kenar ayari  
Gecikme(12);  
GLCD_E=0; // yetki için düşen kenar saglandi
```

```

GLCD_E=1; // islemin yetkilendirilmesi için düşen kenar ayari
Gecikme(12);
GLCD_E=0; // yetki için düşen kenar saglandi
oku=P2; // GLCD data girislerindeki veriyi oku degiskenine aktar
GLCD_CS1=1; // GLCD'nin seçili çizgilerini resetle
GLCD_CS2=1;
return oku; // GLCD uçlarından okunan degeri geri dondur
}
/* Grafik LCD'de koordinatlari (x, y) verilen bir noktaya int Deger Yazdirma Fonksiyonu */
void GLCD_int_8Bit (int x, int y, int Veri_Bayt)
{
bit sec=0; // GLCD'nin kullanılacak kismini sec
if (x >= 64) // GLCD'nin sag mi yoksa sol kismini seçili
{
if (x>127)
x=0;
else
{
x-=64;
sec=1;
}
}
GLCD_DI=0;
x&=0x7f;
x|=0x40;
GLCD_int_Yaz (sec, x);
GLCD_int_Yaz (sec, (y & 0xBF) | 0xB8);
GLCD_DI=1; // veri modu
// komut moduna geç ve koordinati belirli noktayi isaretle
GLCD_DI=0; // komut modu
GLCD_int_Yaz (sec, x); // yatay adresi kur
// Yatay (x) adresi belirlenen noktaya fonksiyondan gonderilen degeri yaz
GLCD_DI=1; // veri modu
GLCD_int_Yaz (sec, Veri_Bayt); // Veri_Bayt degerini piksele yaz
}
/* GLCD Ekranini Doldurmak ve Temizleme Islemi */
void GLCD_Ekran (bit renk)
{
int i, j, dolgu;
#define Acik_Bayt 0xFF
#define Kapali_Bayt 0x00
// dikey sayfalar vasitasiyla dongu
for (i=0; i<8; i++)
{
GLCD_DI=0; //komut modu
GLCD_int_Yaz (0, 0x40); // 0100 000 B yatay (sütun) adresi 0'a kur
GLCD_int_Yaz (1, 0x40); // 0100 000 B
GLCD_int_Yaz (0, i | 0xB8); // 1011 1000 B sayfa (saticir) adresini kur
GLCD_int_Yaz (1, i | 0xB8); // 1011 1000 B
GLCD_DI=1; //veri modu
// Yatay kisimler vasitasiyla döngü
if (renk)
dolgu=Acik_Bayt;
else
dolgu=Kapali_Bayt;
for(j = 0; j < 64; j++)
{
GLCD_int_Yaz (0, dolgu); // sütunlari doldur ya da temizle
GLCD_int_Yaz (1, dolgu);
}
}
}

```

```

}
}
/* GLCD'yi baslangic ayarlarina kurma */
void GLCD_Baslat (bit mod)
{
/* GLCD pinlerinin baslangic ayari*/
// GLCD_RST=0;
GLCD_E=0;
GLCD_CS1=1;
GLCD_CS2=1;
GLCD_DI=0; // komut modu
GLCD_int_Yaz (GLCD_CS1, 0xC0);
GLCD_int_Yaz (GLCD_CS2, 0xC0);
GLCD_int_Yaz (GLCD_CS1, 0x40); // sütun adresini 0'a kur
GLCD_int_Yaz (GLCD_CS2, 0x40);
GLCD_int_Yaz (GLCD_CS1, 0xB8); // sayfa adresini 0'a kur
GLCD_int_Yaz (GLCD_CS2, 0xB8);
if(!mod)
{
GLCD_int_Yaz (GLCD_CS1, 0x3F); // Ekran acik
GLCD_int_Yaz (GLCD_CS2, 0x3F);
}
else
{
GLCD_int_Yaz (GLCD_CS1, 0x3E); // Ekran kapali
GLCD_int_Yaz (GLCD_CS2, 0x3E);
}
GLCD_Ekran (Kapali); // GLCD'yi temizle
}
/* GLCD'ye Harf Yazdirma Fonksiyonu */
/* Belirtilen koordinatlarda (x,y), ilgili harfi ekrana bastirir */
void GLCD_Harf (int x, int y, char* karptr, int bekle, bit renk)
{
int memcopy;
int i, j;
int temp;
int piksel_veri [5]; // sabitler.h dosyasindaki karakter tablosu 99x5 bir matristir
if (!renk)
GLCD_int_8Bit (x++, y, 0xFF); // ilgili adresi doldur
else
GLCD_int_8Bit (x++, y, 0x00); // ilgili adresi temizle
for (i=0; karptr[i] != '\0'; i++)
{
for(memcopy=0; memcopy<5; memcopy++)
piksel_veri [memcopy]=Kar_Tablo[karptr[i]-' '][memcopy];
for(j=0; j<5; j++)
{
if(renk)
temp=piksel_veri[j];
else
temp=0x80 | ~ piksel_veri[j];
GLCD_int_8Bit (x+j, y, temp);
Gecikme (bekle); //sutunlar arasina gecikme konulur
}
}
if (!renk)
{ x+=5; GLCD_int_8Bit(x++, y, 0xff);}
else{ x+=5; GLCD_int_8Bit(x++, y, 0x00);}
}
}
/*

```

calismasi:

ust fonk. dan gelen x y degerlerine gore, textpointer in gosterdigi dizideki her karakteri '/' karakterine gelene

kadar ekrana 7*5 formatiyla yazilir.

karakter tanimi 2 farkli tablo ile yapilmistir */

Kütüphaneye ilişkin c dosyası aşağıdaki gibidir.

```
#include <89c51rd2.H> // islemci baslik dosyasi
#include <stdio.h> // C dili baslik dosyasi
#include "Tablo.h" // Resimlerimizin bulunduğu dosya
#include "G_LCD.h" // Grafik LCD baslik dosyasi
Kurt(){
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (15000);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableKurt[j++]);
}
}
Gecikme (15000);
}
}
Itu(){
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (15000);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableItu[j++]);
}
}
Gecikme (15000);
}
}
/* Türkiye haritasını ekrana basan fonksiyon */
/* resimlerin tanımlandığı Tablo.h dosyasından sırasıyla 8 bitlik degerler göstergeye gönderilir. Her resim basımından sonra resim gelen tablo degerlerinin '~' islemi ile terslenmesi sonucu ters renkte basılır. */
Turkiye(){
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (15000);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
```

```

{
if(i==0)
GLCD_int_8Bit (x, y, TableTurkiye[j++]);
}
}
Gecikme (15000);
}
}
/*Bayragimizi ekrana basan fonksiyon*/
Bayrak(){
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (15000);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableBayrak[j++]);
}
}
Gecikme (15000);
}
}
//Ulu önder Atatürk'ün resmini ekrana basan fonksiyon
Ataturk(){
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (15000);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableAtaturk[j++]);
}
}
Gecikme (15000);
}
}
//Dünya resimlerini basan fonksiyonlar
DispDUNYA1()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA1[j++]);
}
}
}
}
}
}

```

```

Gecikme (500);
}
}
DispDUNYA2()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA2[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA3()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA3[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA4()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA4[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA5()
{
int x,y,i,j;

```

```

for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA5[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA6()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA6[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA7()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA7[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA8()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{

```

```

for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA8[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA9()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA9[j++]);
}
}
Gecikme (500);
}
}
DispDUNYA10()
{
int x,y,i,j;
for(i=0; i<2; i++)
{
j=0;
Gecikme (500);
for(y=0; y<8; y++) // 8 sayfa içerisindeki döngü
{
for(x=0; x<=127; x++) // sütunlar arasında döngü
{
if(i==0)
GLCD_int_8Bit (x, y, TableDUNYA10[j++]);
}
}
Gecikme (500);
}
}
main()
{
int x=0,y=0,Data_X=0,i=0,j=0;
bit renk=0;
char TEXT0[]="*****";
char karakter[]=" ";
while(1)
{
GLCD_Ekran(0); //dolgu var
x=0;
y=0;
sprintf(TEXT0,"#####");
GLCD_Harf(1,0, TEXT0, 100, 1);
sprintf(TEXT0, " OZEN ");
GLCD_Harf(1,1, TEXT0, 100, 1);
}
}

```

```

sprintf(TEXT0, " OZKAYA ");
GLCD_Harf(1,2, TEXT0, 100, 1);
sprintf(TEXT0," ITU ");
GLCD_Harf(1,3, TEXT0, 100, 1);
sprintf(TEXT0, "ELEKTRIK - ELEKTRONIK");
GLCD_Harf(1,4, TEXT0, 100, 1);
sprintf(TEXT0, " FAKULTESI ");
GLCD_Harf(1,5, TEXT0, 100, 1);
sprintf(TEXT0, " ELEKTRONIK-2 ");
GLCD_Harf(1,6, TEXT0, 100, 1);
sprintf(TEXT0, " 040060342 ");
GLCD_Harf(1,7, TEXT0, 100, 1);
Gecikme(10000);
Itu();
Turkiye();
Kurt();
Bayrak();
Ataturk();
DispDUNYA1();
DispDUNYA2();
DispDUNYA3();
DispDUNYA4();
DispDUNYA5();
DispDUNYA6();
DispDUNYA7();
DispDUNYA8();
DispDUNYA9();
DispDUNYA10();
}
}

```

Şimdi de fotoğrafları görüntü dizisi halinde tuttuğumuz Tablo.h kütüphanesinin kodlarını göreceksiniz. Kullandığımız grafik lcd 128x64 'lük olduğu için her fotoğrafı 128x64 adet bit ile ifade edeceğiz. Bu da 128x8 byte yapar. Grafik lcdmizde her pixel ya yanmaktadır, ya da yanmamaktadır. Dolayısıyla bir her pixel verisi için 1 bit ayırmamız gerekmektedir. Harfleri 40 pixelde tutuyoruz. İsteğe göre daha büyük yahut daha küçük harfler tasarlayabileceğiniz gibi, farklı fontlarda harfler de tasarlayabilirsiniz. Kodun sonunda gördüğünüz 10 adet dünya fotoğrafının peşpeşe gösterilmesinin nedeni, dünyanın dönüşünü grafik olarak göstermek istememizdir.

Herhangi bir renki resmi yahut fotoğrafı, görüntü dizisine aktarmadan önce onu 128x64 boyutuna getirmelisiniz. Daha sonra matrox ya da opengl gibi görüntü işleme kütüphanelerini kullanarak, threshold fitler denen uygulama ile resmi siyah beyaz uzayına aktarabilirsiniz. Bu sayede elde edilen görüntü matrisi aşağıdaki gibi olacaktır. Bunu yapan hazır programlar da mevcuttur. Örneğin "Image2GLCD" programı ile tüm bu işlemler gerçekleştirilebilir.

Kodumuzda resmi gösterirken matrisi hangi sıra ile okuduğunuz önemlidir. Bu, aynı zamanda görüntüyü efekt ile açmamızı sağlar.

```

static code int _OR_SET[8]={0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
static code int _AND_CLEAR[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
static code char TableTurkiye[1024]={
0xFE,0xFF,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,0xFE,
0xFF,0xFE,
0x03,0x0F,0x0F,0x0F,0x0F,0x4F,0xEF,0xEF,0xEF,0xFF,0xEF,0xEF,0xEF,0xEF,0xEF,0xEF,0x8F,0x0F,0x0F,

```


0x14, 0x3E, 0x14, 0x3E, 0x14, // #
0x24, 0x2A, 0x7F, 0x2A, 0x12, // \$
0x43, 0x33, 0x08, 0x66, 0x61, // %
0x36, 0x49, 0x55, 0x22, 0x50, // &
0x00, 0x05, 0x03, 0x00, 0x00, // '
0x00, 0x1C, 0x22, 0x41, 0x00, // (
0x00, 0x41, 0x22, 0x1C, 0x00, //)
0x14, 0x08, 0x3E, 0x08, 0x14, // *
0x08, 0x08, 0x3E, 0x08, 0x08, // +
0x00, 0x50, 0x30, 0x00, 0x00, // ,
0x08, 0x08, 0x08, 0x08, 0x08, // -
0x00, 0x60, 0x60, 0x00, 0x00, // .
0x20, 0x10, 0x08, 0x04, 0x02, // /
0x3E, 0x51, 0x49, 0x45, 0x3E, // 0
0x04, 0x02, 0x7F, 0x00, 0x00, // 1
0x42, 0x61, 0x51, 0x49, 0x46, // 2
0x22, 0x41, 0x49, 0x49, 0x36, // 3
0x18, 0x14, 0x12, 0x7F, 0x10, // 4
0x27, 0x45, 0x45, 0x45, 0x39, // 5
0x3E, 0x49, 0x49, 0x49, 0x32, // 6
0x01, 0x01, 0x71, 0x09, 0x07, // 7
0x36, 0x49, 0x49, 0x49, 0x36, // 8
0x26, 0x49, 0x49, 0x49, 0x3E, // 9
0x00, 0x36, 0x36, 0x00, 0x00, // :
0x00, 0x56, 0x36, 0x00, 0x00, // ;
0x08, 0x14, 0x22, 0x41, 0x00, // <
0x14, 0x14, 0x14, 0x14, 0x14, // =
0x00, 0x41, 0x22, 0x14, 0x08, // >
0x02, 0x01, 0x51, 0x09, 0x06, // ?
0x3E, 0x41, 0x59, 0x55, 0x5E, // @
0x7E, 0x09, 0x09, 0x09, 0x7E, // A
0x7F, 0x49, 0x49, 0x49, 0x36, // B
0x3E, 0x41, 0x41, 0x41, 0x22, // C
0x7F, 0x41, 0x41, 0x41, 0x3E, // D
0x7F, 0x49, 0x49, 0x49, 0x41, // E
0x7F, 0x09, 0x09, 0x09, 0x01, // F
0x3E, 0x41, 0x41, 0x49, 0x3A, // G
0x7F, 0x08, 0x08, 0x08, 0x7F, // H
0x00, 0x41, 0x7F, 0x41, 0x00, // I
0x30, 0x40, 0x40, 0x40, 0x3F, // J
0x7F, 0x08, 0x14, 0x22, 0x41, // K
0x7F, 0x40, 0x40, 0x40, 0x40, // L
0x7F, 0x02, 0x0C, 0x02, 0x7F, // M
0x7F, 0x02, 0x04, 0x08, 0x7F, // N
0x3E, 0x41, 0x41, 0x41, 0x3E, // O
0x7F, 0x09, 0x09, 0x09, 0x06, // P
0x1E, 0x21, 0x21, 0x21, 0x5E, // Q
0x7F, 0x09, 0x09, 0x09, 0x76, // R
0x26, 0x49, 0x49, 0x49, 0x32, // S
0x01, 0x01, 0x7F, 0x01, 0x01, // T
0x3F, 0x40, 0x40, 0x40, 0x3F, // U
0x1F, 0x20, 0x40, 0x20, 0x1F, // V
0x7F, 0x20, 0x10, 0x20, 0x7F, // W
0x41, 0x22, 0x1C, 0x22, 0x41, // X
0x07, 0x08, 0x70, 0x08, 0x07, // Y
0x61, 0x51, 0x49, 0x45, 0x43, // Z
0x00, 0x7F, 0x41, 0x00, 0x00, // [
0x02, 0x04, 0x08, 0x10, 0x20, // \
0x00, 0x00, 0x41, 0x7F, 0x00, //]
0x04, 0x02, 0x01, 0x02, 0x04, // ^

```

0x40, 0x40, 0x40, 0x40, 0x40, // _
0x00, 0x01, 0x02, 0x04, 0x00, // `
0x20, 0x54, 0x54, 0x54, 0x78, // a
0x7F, 0x44, 0x44, 0x44, 0x38, // b
0x38, 0x44, 0x44, 0x44, 0x44, // c
0x38, 0x44, 0x44, 0x44, 0x7F, // d
0x38, 0x54, 0x54, 0x54, 0x18, // e
0x04, 0x04, 0x7E, 0x05, 0x05, // f
0x08, 0x54, 0x54, 0x54, 0x3C, // g
0x7F, 0x08, 0x04, 0x04, 0x78, // h
0x00, 0x44, 0x7D, 0x40, 0x00, // i
0x20, 0x40, 0x44, 0x3D, 0x00, // j
0x7F, 0x10, 0x28, 0x44, 0x00, // k
0x00, 0x41, 0x7F, 0x40, 0x00, // l
0x7C, 0x04, 0x78, 0x04, 0x78, // m
0x7C, 0x08, 0x04, 0x04, 0x78, // n
0x38, 0x44, 0x44, 0x44, 0x38, // o
0x7C, 0x14, 0x14, 0x14, 0x08, // p
0x08, 0x14, 0x14, 0x14, 0x7C, // q
0x00, 0x7C, 0x08, 0x04, 0x04, // r
0x48, 0x54, 0x54, 0x54, 0x20, // s
0x04, 0x04, 0x3F, 0x44, 0x44, // t
0x3C, 0x40, 0x40, 0x20, 0x7C, // u
0x1C, 0x20, 0x40, 0x20, 0x1C, // v
0x3C, 0x40, 0x30, 0x40, 0x3C, // w
0x44, 0x28, 0x10, 0x28, 0x44, // x
0x0C, 0x50, 0x50, 0x50, 0x3C, // y
0x44, 0x64, 0x54, 0x4C, 0x44, // z
0x00, 0x08, 0x36, 0x41, 0x41, // {
0x00, 0x00, 0x7F, 0x00, 0x00, // |
0x41, 0x41, 0x36, 0x08, 0x00, // }
0x02, 0x01, 0x02, 0x04, 0x02}; // ~
static code unsigned char TableKurt[1024] = {
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0xF0, 0x70, 0x50, 0x30, 0x10, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xE6, 0xFC, 0x07, 0x02,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x80, 0x80, 0x80, 0xC0, 0xC0, 0xC0, 0xC0,
0xC0,
0x80, 0x80, 0x20, 0x20, 0x20, 0x30, 0x38, 0x2E, 0x07, 0x11, 0x10, 0x10, 0x50, 0x50, 0x50,
0x40,
0x40, 0x48, 0x48, 0x48, 0xC8, 0x48, 0x40, 0x40, 0x40, 0x44, 0x44, 0x04, 0x2C, 0x2C,
0x3C,
0x3C, 0x3C, 0x3C, 0x3C, 0x3C, 0x2C, 0x2C, 0x3C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C, 0x1C,
0x1C,
0x1E, 0x1E, 0x1E, 0x0E, 0x3E, 0xFE, 0x3E, 0x9E, 0xFE, 0xCE, 0xFE, 0x76, 0xF6, 0xDE, 0xFE,
0x9E,
0xF6, 0xEA, 0xFB, 0xFB, 0xFF, 0xD7, 0xE1, 0xFB, 0xDD, 0xEB, 0xFF, 0xAF, 0x7C, 0xF8, 0x80,

```

0x70,
0xF0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x0F, 0x1E, 0xD8, 0xF0, 0xC0, 0x80, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x0F, 0x3D, 0xFB, 0xED, 0xFF, 0x7E, 0xCF, 0xBF, 0x7F,
0xFF,
0xF7, 0xE1, 0xF1, 0xEF, 0xDF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0xFF, 0xDF, 0xFE, 0x5F, 0xFF,
0xF8,
0x63, 0x3F, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x07, 0x0E, 0x1D, 0x1F, 0x3E,
0x38,
0x30, 0xE0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80, 0x00, 0x03, 0x87, 0x8F, 0x3B, 0x6F, 0xFD, 0x77,
0xBE,
0xF7, 0xAB, 0xFF, 0x57, 0xE7, 0x77, 0x7B, 0x2F, 0x1B, 0x17, 0x0B, 0x07, 0x06, 0x03, 0x03,
0x01,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x01, 0x01, 0x06, 0x04, 0x08, 0x37, 0x2F, 0x29, 0x32, 0x32, 0x2C, 0x2C, 0x20,
0x20,
0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20,
0x20,
0x20, 0x20, 0x20, 0x28, 0x28, 0x28, 0xE8, 0xE8, 0xC8, 0x68, 0xA8, 0xB8, 0x78, 0x50, 0x50,
0x50,
0xD0, 0xF0, 0xF0, 0xD0, 0xD2, 0xD3, 0xB7, 0xF4, 0x74, 0xFF, 0xFB, 0xB9, 0xF8, 0xF9, 0xFB,
0xFE,
0xFD, 0xFE, 0xFF, 0xFF, 0xFF, 0xF7, 0xFA, 0xFA, 0xFA, 0xFA, 0xFC, 0xFC, 0xFC, 0xFC, 0xF8,
0xF8,
0xA8, 0xF0, 0xE0, 0xF0, 0x90, 0xC0, 0xC0, 0xE0, 0xE0, 0xE0, 0x80, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x40,
0xE0, 0xB8, 0x08, 0xF4, 0xFE, 0x3F, 0x0F, 0x07, 0x01, 0x01, 0x41, 0x41, 0x41, 0x41, 0x41,

0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x04, 0x06, 0x06, 0x0E, 0x1F, 0x18, 0x10,
0x00,
0x00, 0x01, 0x03, 0x07, 0x02, 0x02, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0,
0xC0,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0x80, 0x80,
0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x02, 0x03, 0x03, 0x03, 0x0F,
0x0F,
0x07, 0x03, 0x11, 0x30, 0x18, 0x0C, 0x0C, 0x86, 0xC2, 0xE2, 0xB0, 0x90, 0xC0, 0x40, 0x40,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x20, 0x68, 0x6C, 0x48, 0x58, 0x79,
0x71,
0x33, 0x03, 0x06, 0x06, 0x03, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x3F, 0x1F, 0x1F, 0x0F, 0x07, 0x87, 0xC3, 0xC1,
0xE0,
0xF0, 0xF0, 0xF8, 0xF8, 0xF8, 0xF8, 0xF0, 0xE0, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x0C, 0x1C,
0x36,
0x0E, 0x8B, 0xCE, 0xC6, 0xCC, 0xE0, 0x60, 0xE0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x40, 0x40, 0xC4, 0xCE, 0xCF, 0xCF, 0xCF, 0xDA, 0x1E, 0x06, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x40, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x3F, 0x3F, 0x9F, 0x9F,
0xCF,
0xC7, 0xE3, 0xF3, 0xF1, 0xF8, 0xF8, 0xFC, 0xFC, 0xFE, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0xFC, 0x70, 0x40, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x02, 0x02, 0x33, 0x79, 0x61, 0x79, 0x38, 0x08, 0x38, 0x10, 0x00, 0x00,
0x00,
0x00, 0x62, 0x63, 0x63, 0x73, 0x52, 0xF2, 0xD2, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xC0,
0xC0,
0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0x80, 0x00, 0x00, 0xC0,
0xC0,
0xC0, 0xC0, 0xC0, 0xCF, 0xC7, 0xC7, 0xCB, 0xC9, 0xCD, 0xCC, 0xCE, 0xCF, 0xCF, 0xC7, 0xC7,
0xC7,
0xC7, 0xC7, 0xC7, 0xC7, 0xC7, 0xC7, 0xC3, 0x83, 0x03, 0x83, 0xC3, 0xC3, 0xE3, 0xE3, 0x03,
0x01,
0xE1, 0xE1, 0xE1, 0xE1, 0xE1, 0xE0, 0xE0, 0x00, 0x00, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0,
0x00,
0xE0, 0xE0, 0xC0, 0xC0, 0xC0, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

```
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x33, 0x33, 0x33, 0x33, 0x33, 0x33, 0x7B, 0x78,  
0x01,  
0x00, 0x0E, 0x1E, 0x18, 0x8C, 0x0C, 0x8C, 0xCC, 0xCC, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x00, 0xFF, 0xFF,  
0xFF,  
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x00, 0x00, 0xFF,  
0xFF,  
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,  
0xFF,  
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x00, 0x3F, 0x7F, 0x7F, 0xFF, 0xFF, 0x00,  
0x00,  
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,  
0x00,  
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0xFC, 0xE0, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xC0, 0x4E, 0x8E, 0xC2, 0xCE, 0xCE, 0x8A, 0x1E, 0x02,  
0x00,  
0x00, 0x00, 0x00, 0x03, 0x23, 0x22, 0x33, 0x33, 0x10, 0x13, 0x19, 0x18, 0x08, 0x40, 0x60,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x02,  
0x02,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x7F, 0x7F, 0x7F, 0x7F, 0x7F, 0x00, 0x00, 0x00, 0x7F, 0x7F, 0x7F, 0x7F, 0x7F, 0x00,  
0x00,  
0x00, 0x7F, 0x7F, 0x7F, 0x7F, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x20, 0x60, 0x28, 0x0C, 0x09, 0x09, 0x19, 0x19, 0x10, 0x03, 0x11, 0x01, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x20, 0xE0, 0xE0, 0x00, 0x00, 0x20, 0xE0, 0xE0, 0x20,  
0x20,  
0x20, 0x20, 0xE0, 0x60, 0x20, 0x20, 0xA0, 0xA0, 0xC0, 0xE0, 0xE0, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00,  
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
0x00  
};
```

Koda ilişkin proteus simülasyonarını aşağıda görebilirsiniz:

