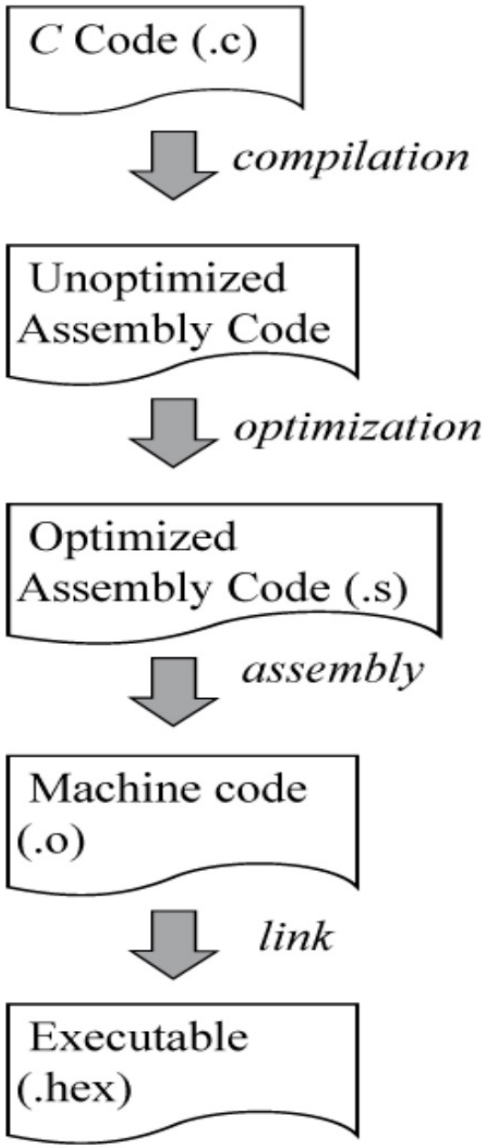
BÖLÜM 7
Giriş/Çıkış Portları, Reset, ve Watchdog Timer

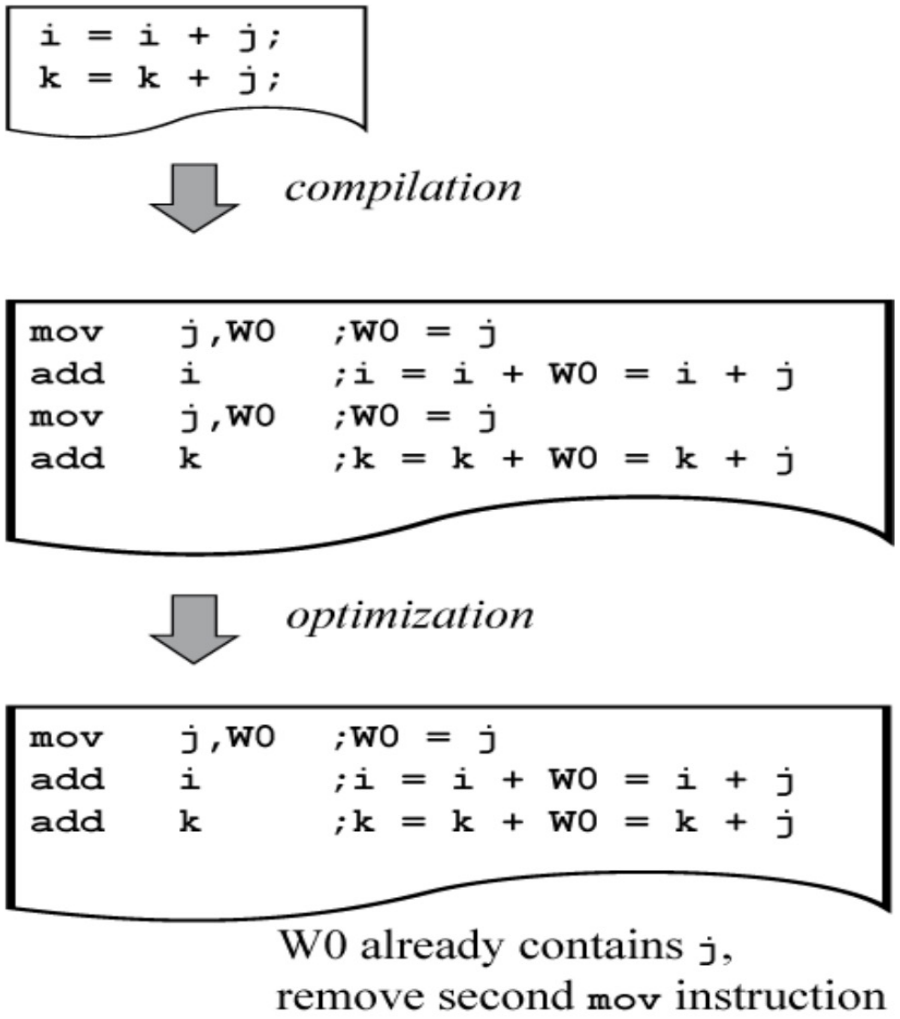
Doç. Dr. Fatih Evran
Elektrik ve Electronik Mühendisliği
Ofis: 421 Mühendislik Binası
Düzce Üniversitesi
Email: fatihevran@duzce.edu.tr

C Derleme

From .c to .hex



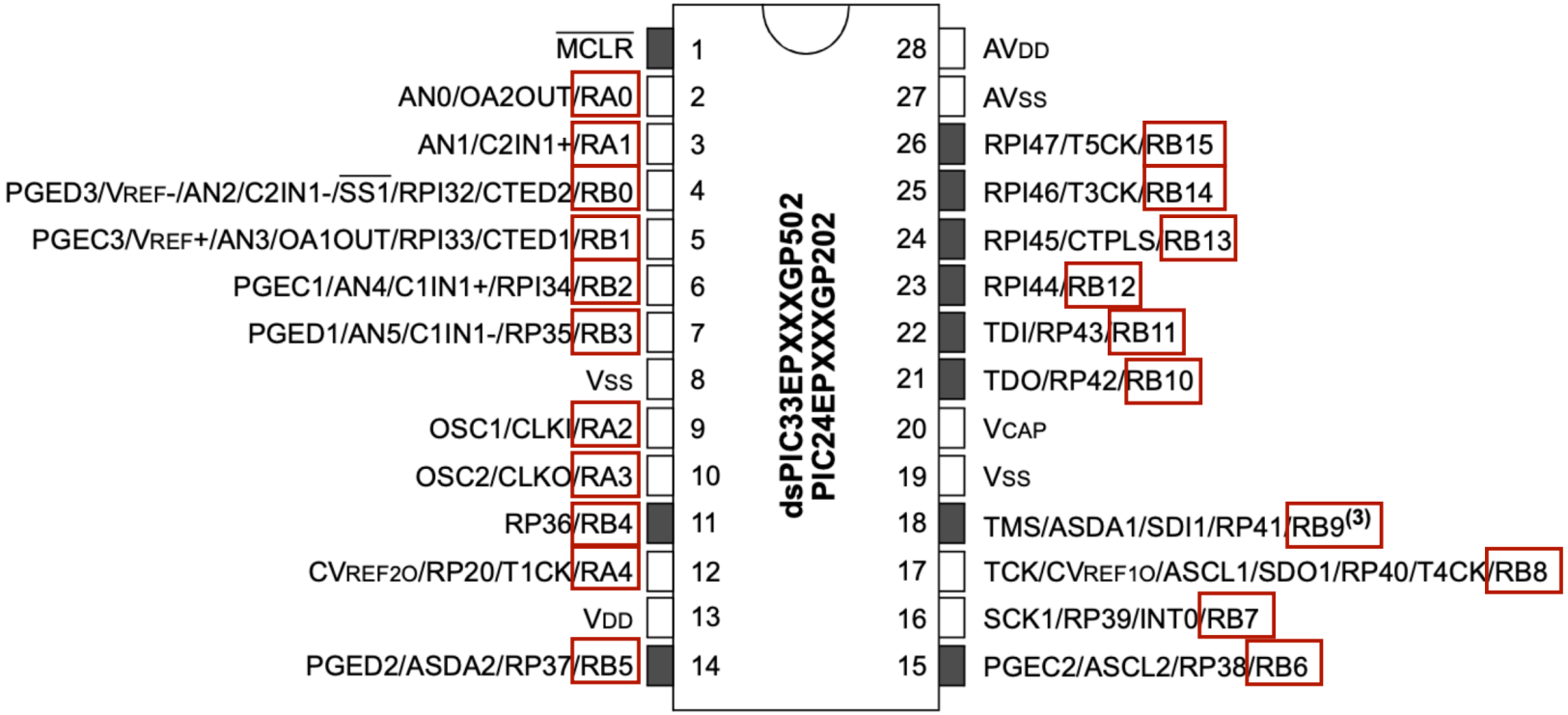
Example Optimization



W0 already contains `j`,
remove second `mov` instruction

Genel Amaçlı Giriş/Çıkış Portları (GPIO)

- Bir PIC µC birden fazla Giriş/Çıkış portlarına sahip olabilir
- Her bir Giriş/Çıkış portu birden fazla bağımsız GPIO pinine sahip olabilir
  A. PORTA: RA0' dan RA4' e kadar olan pinler
  B. PORTB: RB0' dan RB15' e kadar olan pinler



Giriş ve Çıkış (1)

- Her giriş/çıkış pini ya giriş (okuma) ya da çıkış (yazma) olabilir
- Bir pinin giriş veya çıkış durumu, TRIS saklayıcısındaki ilgili bitten kontrol edilir
  A. TRISx' teki #n bitini 1'e ayarlayınız —> PORTx' in #n pini giriş olacaktır
  B. TRISx' teki #n bitini 0 yapınız —> PORTx' in #n pini çıkış olacaktır

Örnek-1

RB3 ve RB5' i giriş olarak, PORTB' deki diğer pinleri çıkış olarak ayarlayın

TRISB' deki bit 3 ve 5' i 1 yapın → TRISB = 0b0000 0000 0010 1000

bit 5, bit 3

Örnek- 2

RA0 ve RA2' yi giriş olarak, PORTA' daki diğer pinleri çıkış olarak ayarlayın

TRISA' daki bit 0 ve 2' yi 1 yapın → TRISA = 0b0000 0000 0000 0101

bit 2, bit 0

Giriş ve Çıkış (2)

- Bir GPIO çıkış olarak yapılandırılırsa, çıkışın durumu (0 veya 1) PORT saklayıcısındaki ilgili bitten kontrol edilir.
  A. PORTx' teki #n bitini 1' e ayarlayınız —> PORTx' in #n biti 1 olacaktır
  B. PORTx' teki #n bitini 0 yapınız—> PORTx' in #n biti 0 olacaktır

Adım 1

RB13 ve RB15' i çıkış olarak, PORTB' deki diğer pinleri giriş olarak ayarlayınız

TRISB' deki bit 13 ve 15' i 0 yapın → TRISB = 0b0101 1111 1111 1111

bit 15 bit 13

Adım 2

RB13 = RB15 = 1, PORTB' deki diğer pinler giriş olsun

PORTB' deki bit 13 ve 15' i 1 yapın → PORTB = 0b1010 0000 0000 0101

bit 15 bit 13

Giriş ve Çıkış (3)

- Bir GPIO pini giriş olarak yapılandırılmışsa, PORT saklayıcındaki karşılık gelen bit, pinden okunan değer olacaktır.

Örnek

A. RB0' ın 1 olup olmadığının test edilmesi (yani giriş = 1):

```
                0b0000 0000 0000 0001
    if ((PORTB & 0x0001) == 1)
```

A. RA3' ün 0 olup olmadığının test edilmesi (yani giriş = 0) :

```
                0b0000 0000 0000 1000
    if ((PORTA & 0x0008) == 0)
```

A. RB0 ila RB3' ün 1 olup olmadığının test edilmesi:

```
    if ((PORTB & 0x000F) == 0x000F)

                                0b0000 0000 0000 1111
```

Giriş ve Çıkış (4)

- Okuma veya yazma için birden fazla pin kullanıyorsak PORT saklayıcısının adının kullanılması uygundur.
- Okuma veya yazma için sadece tek bir pin kullanıyorsak, PORT saklayıcısındaki pinin adını kullanabiliriz (örneğin _RA2, _RB3 gibi).

Örnek

A. PORTB' deki tüm GPIO' ları çıkış olarak ayarladıktan sonra _RB5 = 1 yapınız—> yani RB5=1 olmaktadır
B. PORTA' daki tüm GPIO' ları giriş olarak ayarladığımızda, _RA0 RA0 pininden giriş değerini oku anlamına gelir

LATx ve PORTx (1)

- LATx saklayıcısı, PORTx saklayıcısına yazılan son değeri tutar.
- GPIO çıkış olarak yapılandırılmışsa, LATx saklayıcısına yazmak PORTx saklayıcısına yazmakla aynıdır.

Örnek

RB0' ın çıkış olarak yapılandırılması durumunda, _RB0 = 1 veya _LATB0 = 1 kullanımı, RB0 pin çıkışının lojik '1' olmasını sağlar.

- Bir GPIO giriş olarak yapılandırılmışsa:

1. LATx saklayıcısını okumak, PORTx saklayıcısına yazılan son değeri okumak anlamına gelir.
2. PORTx saklayıcısını okumak ise pin üzerindeki gerilimi (0 ya da 1) okumak anlamına gelir.

LATx ve PORTx (2)

- Giriş modunda LATx ve PORTx arasındaki fark:
- RB3' ün toprağa bağlı olduğunu varsayalım ve ardından aşağıdakileri yapalım:

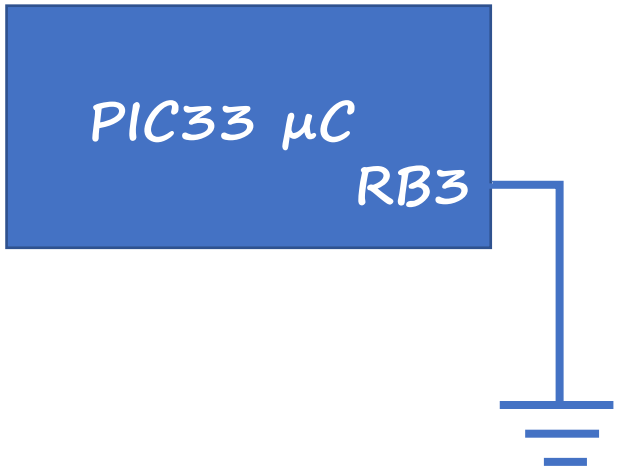
Adım 1: RB3' ü çıkış olarak ayarlayınız, ardından RB3=1 yapınız

```
_TRISB3 = 0;                    // RB3 çıkış modunda
_RB3 = 1;                       // RB3 = '1'
```

Bu, PORTB' deki bit 3' ü 1 yapacaktır.

Adım 2: LATB3' ü okumak, RB3' e yazılan son değeri okumaktır. Yani okunan değer 1 olmaktadır.

Adım 3: RB3' ü okumak ise RB3 pini üzerindeki gerilimi okumak anlamına gelmektedir. RB3 toprağa bağlanıldığı için okunan değer 0 olmaktadır.



LATx ve PORTx (3)

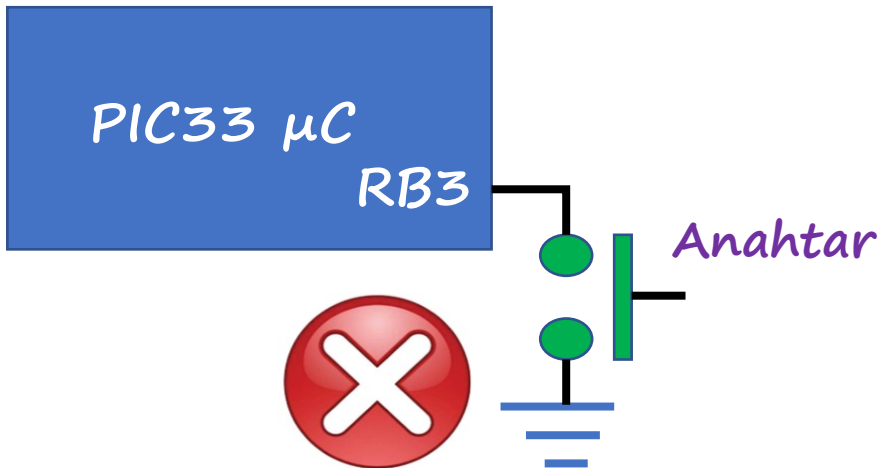
```
_LATB0=1;        bset LATB,#0
_LATB1=1;        bset LATB,#1
```

'bset/bclr' oku/değiştir/yaz (read-modify-write) türünden komuttur. Yukarıdaki komut LATB' yi okur, içeriğini değiştirir, ve sonucu LATB' ye yazar. Bu komut beklenildiği gibi çalışacaktır.

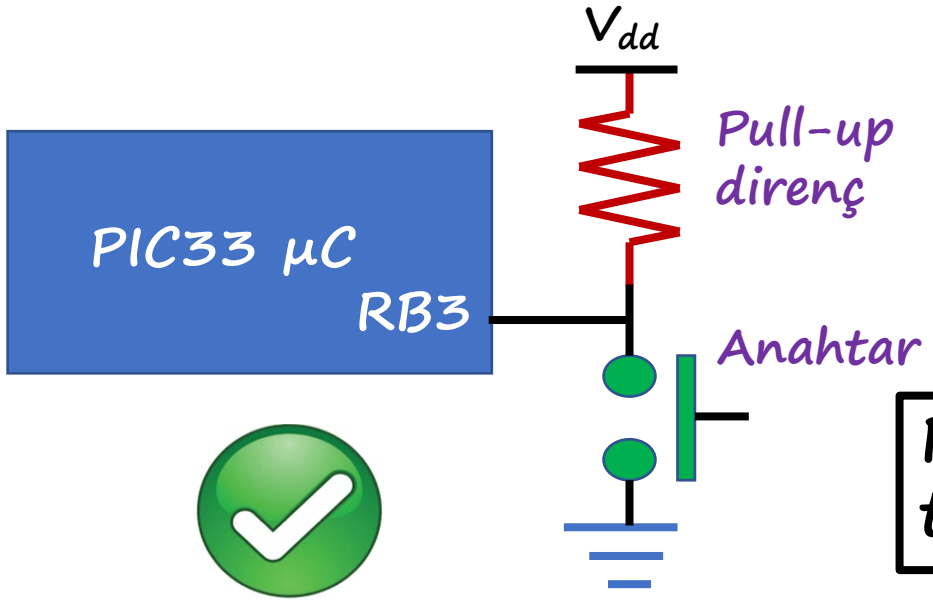
```
_RB0=1;          bset PORTB,#0
_RB1=1;          bset PORTB,#1
```

PORTB'nin başlangıçta içeriğinin sıfır olduğunu ve tüm pinlerin çıkışa göre ayarlandığını varsayalım. Boş bir kondansatörü RB0 pinine bağlanmıştır. İlk komut sonunda RB0=1 olmaktadır. İkinci komutun okuma (read) işlemi sırasında, RB0 pinine bağlı kondansatör kısa devre olduğu için RB0=0 olarak okunur ve yazma (write) işlemi sonunda RB0=0 ve RB1=1 olmaktadır. Pinin kapasitif yüklenmesi ve yüksek CPU hızları nedeniyle, read-modify-write komutları düzgün çalışmayabilir. (Ayrıntılar için işlemcinin kullanma kılavuzuna bakınız). Bu nedenle, örneklerimizde bir pine yazma işleminde PORTx saklayıcısı yerine LATx saklayıcısı kullanılmaktadır.

Giriş Modunda Anahtar Bağlantısı



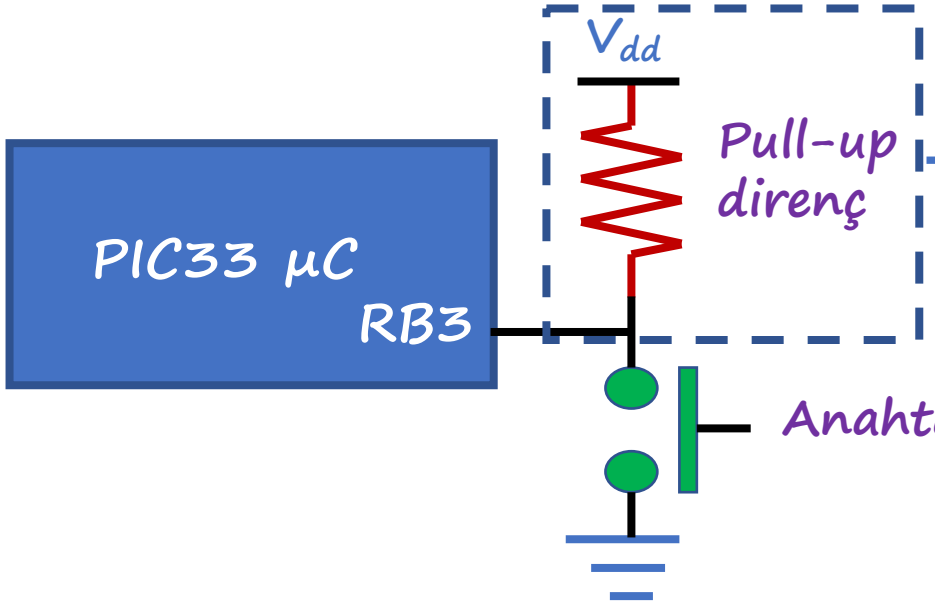
- Anahtara basıldığında, RB3 toprağa bağlı olduğu için, _RB3= 0 olarak okunur.
- Anahtar bırakıldığında, RB3 yüksek empedanslı duruma geçtiği için, _RB3 rastgele bir değer (0 ya da 1) olarak okunur.



- Anahtara basıldığında, RB3 toprağa bağlı olduğu için, _RB3= 0 olarak okunur.
- Anahtar bırakıldığında, RB3 pull-up direnci üzerinden V_{dd}' ye bağlandiği için, _RB3=1 olarak okunur.

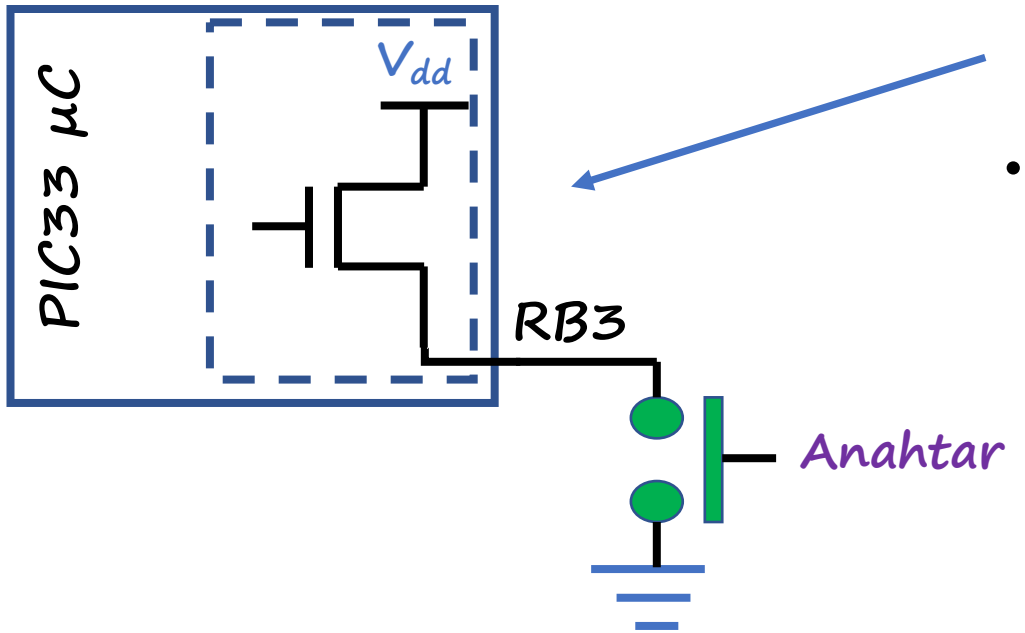
Pull-up direnci anahtara basıldığında V_{dd} ile toprak arasındaki kısa devreyi engeller.

Dahili Zayıf Pull-Up Girişleri



- Yüksek empedanslı durumu önlemek için, pini harici bir pull-up direnci ve harici güç kaynağı (V_{dd}) üzerinden bağlamamız gerekir.

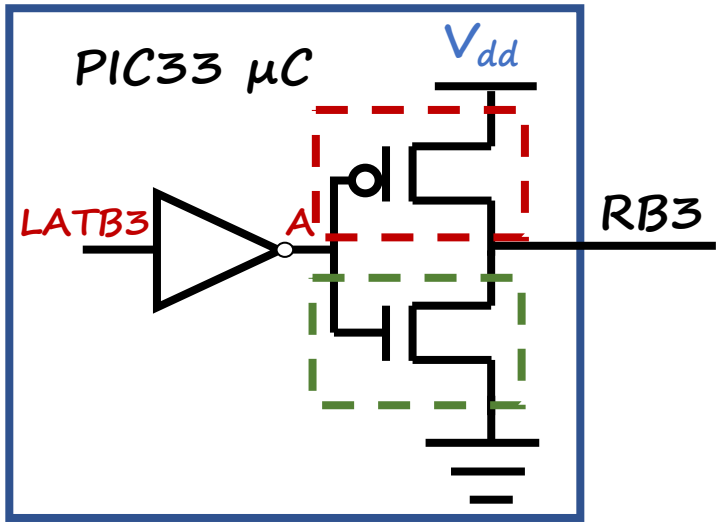
- PIC33'teki GPIO pinleri dahili güç kaynağına bağlanan dahili pull-up dirençlerine sahiptir.
- Bu dirençler PMOS tranzsitörden yapılmıştır ve zayıf pull-up olarak adlandırılır. Çünkü direnç değeri yüksektir ve çok az akım çekerler.
- PIC33EP128GP502 µC' de her pin, sırasıyla CNPUx ve CNPDx saklayıcıları kullanılarak yapılandırılabilen dahili zayıf pull-up ve pull-down direncine sahiptir.




```
_CNPUB3=1; //Enable the Weak Pull Up
```

Çıkış Devresi

- Bir GPIO **normal çıkış** olarak yapılandırıldığında, içi şöyle görünür:

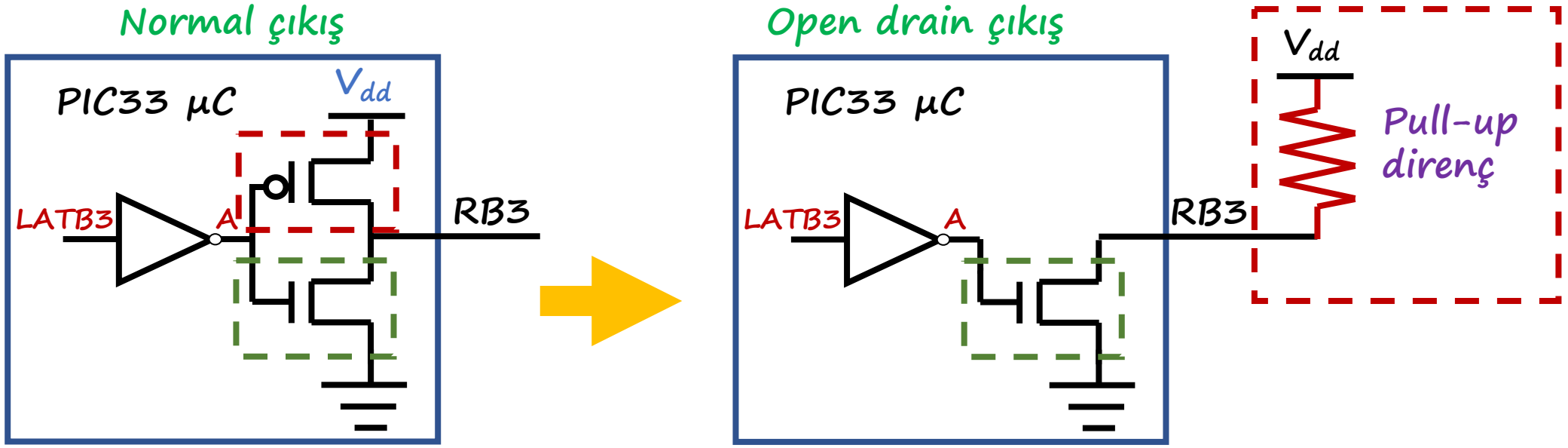


- LATB3 = 0 —> A = 1 —> Üst transistör kesimdedir ve alt transistör iletimdedir —> RB3 alt transistör üzerinden toprağa bağlanır —> RB3 lojik '0' verir.
- LATB3 = 1 —> A = 0 —> Üst transistör iletimdedir ve alt transistör kesimdedir —> RB3 üst transistör üzerinden V_{dd} kaynağına bağlanır —> RB3 lojik '1' verir.

Open Drain Çıkış (1)

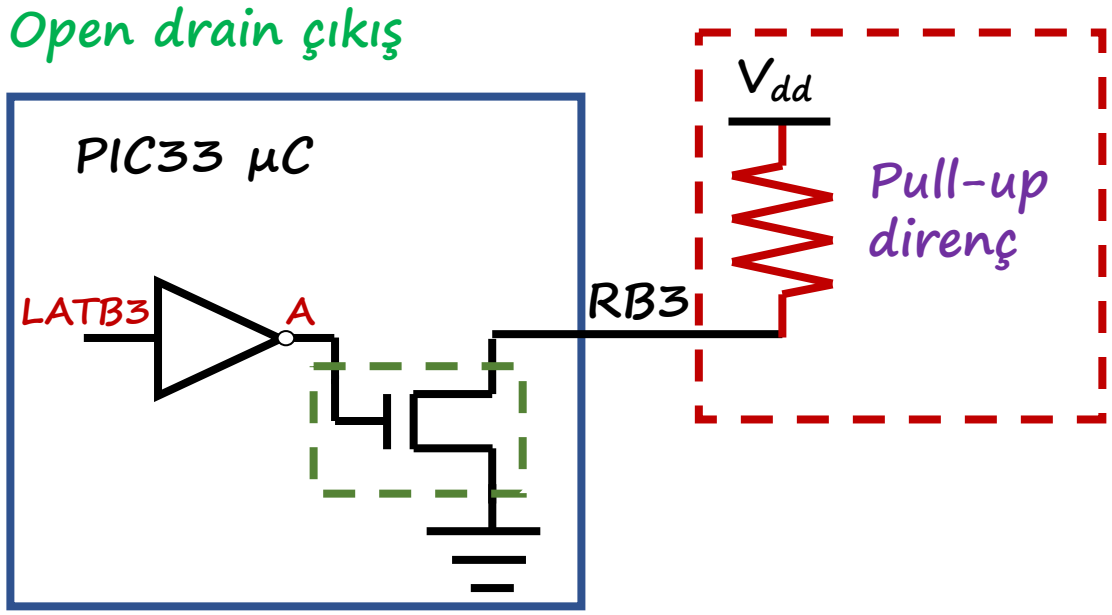
- PIC ayrıca 'open drain çıkış' adı verilen başka bir çıkış modunu da destekler.
- Open drain çıkışında, dahili güç kaynağı (V_{dd}) ve dahili pull-up direnci (üst transistör) devre dışı bırakılır.

- Pin çıkışı harici pull-up direnci üzerinden harici güç kaynağına bağlanmalıdır.



Open Drain Çıkış (2)

- LATB3 = 0 —> A noktası= 1 —> Transistör iletimdedir—> RB3 tranzistör üzerinden toprağa bağlanır —> RB3 lojik '0' verir.
- LATB3 = 1 —> A noktası = 0 —> Transistör kesimdedir—> RB3 harici pull-up direnci üzerinden harici V_{dd}' ye bağlanır —> RB3 lojik '1' verir.



- _ODCxy = 1 'open drain çıkışı' etkinleştirir, _ODCxy = 0 'open drain çıkışı' devre dışı bırakır..

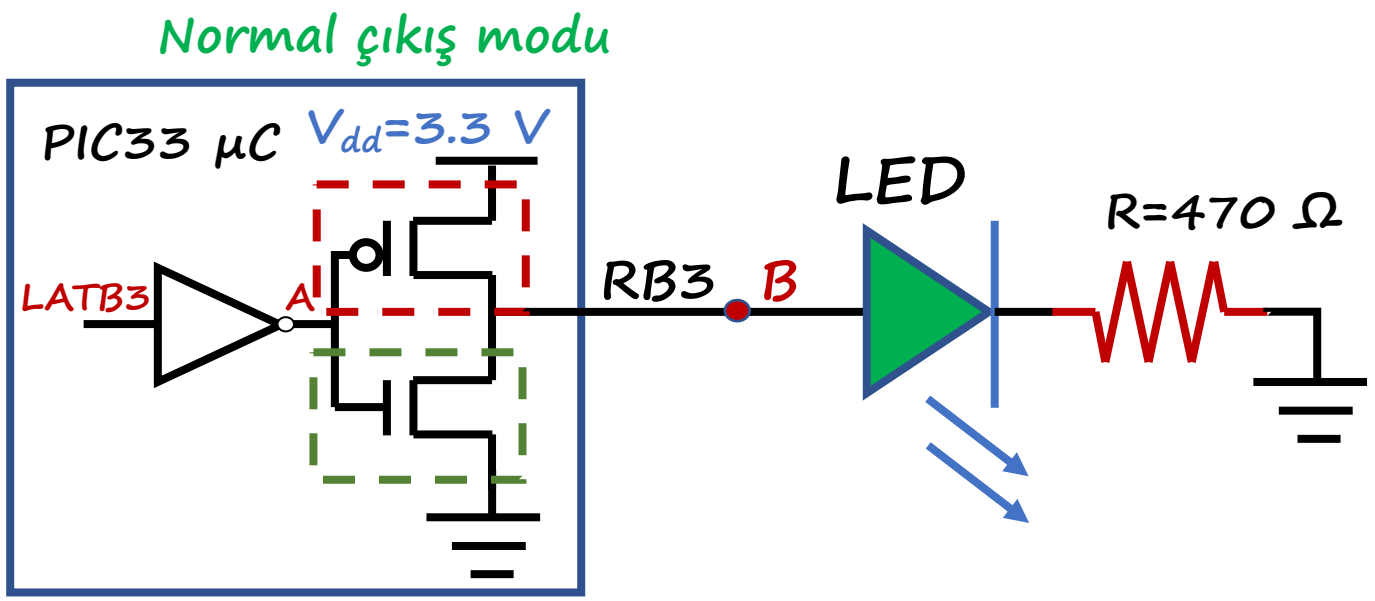
```
_ODCB3=1; //RB3 'open drain' çıkış
```

Normal Çıkış Modunda LED Sürme

A. LATB3 = 0 —> A = 1 —> Üst transistör kesimdedir ve alt transistör iletimdedir—> RB3 alt tranzistör üzerinden toprağa bağlanır —> B = 0 —> LED söner.

B. LATB3 = 1 —> A = 0 —> Üst transistör iletimdedir ve alt transistör kesimdedir —> RB3 üst tranzistör üzerinden V_{dd} kaynağına bağlanır —> B = lojik '1' yani Vdd (3.3 V) —> LED yanar.

C. LED üzerinden geçen akımı sınırlamak için LED' e seri bir direnç (R) bağlanır; aksi takdirde LED hasar görebilir.

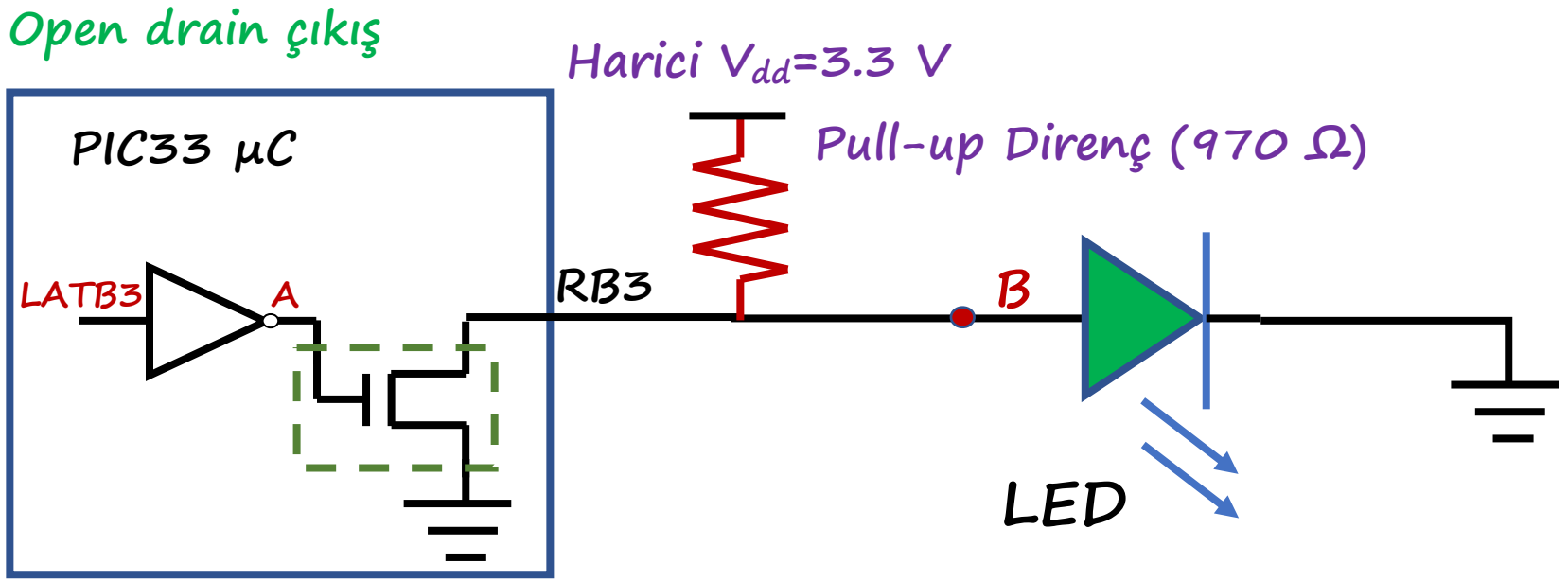


Open Drain Çıkış Modunda LED Sürme

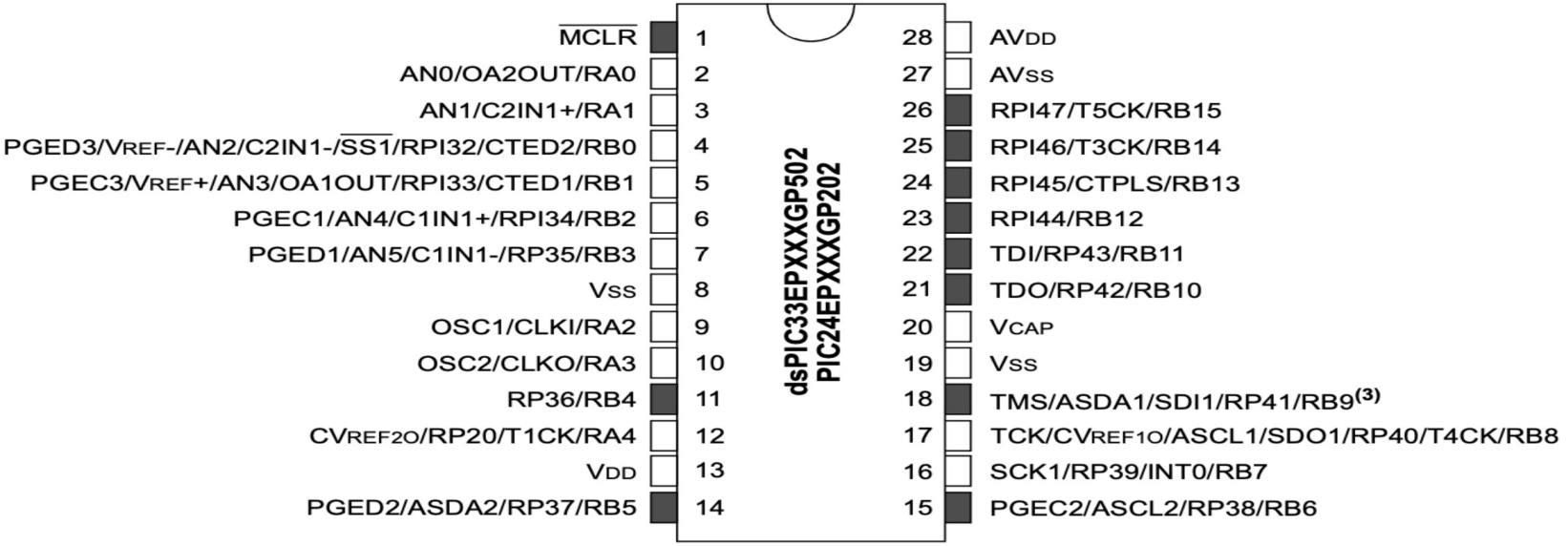
A. LATB3 = 0 —> A = 1 —> Transistor iletimdedir —> RB3 tranzistör üzerinden toprağa bağlanır —> B = 0 —> LED söner.

B. LATB3 = 1 —> A = 0 —> Transistor kesimdedir —> RB3 harici pull-up direnci üzerinden harici V_{dd} kaynağına bağlanır —> B = lojik '1' yani Vdd (3.3 V) —> LED yanar.

C. Pull-up direnci LED ve RB3' ten geçen akımı sınırlamak için uygulanır; aksi takdirde PIC ve LED hasar görebilir.



Analog/Digital pin ve Digital pin



A. PIC' deki çoğu pinin birden fazla özelliği olabilir.

B. Analog/dijital özelliğe sahip pinler (AN0 ve AN1 gibi) maksimum V_{dd} + 0,3 V = 3,3 V + 0,3 V = 3,6 V giriş gerilimine sahiptir

C. Analog kanala sahip olmayan (yalnızca digital) pinler maksimum 5,6 V giriş gerilimine sahiptir.

D. Çoğu GPIO pinleri maksimum 4 mA değerinde bir akım kaynaklayabilir ya da çöktürebilir —> Akımı kısıtlamak için bir

PORTx Paylaşılan Pin Fonksiyonları

- Çip üzerindeki çevresel arabirimler aynı pini kullanabilir. Pini ortak kullanan diğer çevresel arabirimlerin durumuna göre, bir PORTx pinini giriş olarak yapılandırmak için sadece _TRISx=1 yapmak yeterli olmayabilir:

RAO (digital pin), çip üzerindeki Analog-Dijital Dönüştürücünün (ADC) analog girişi olan ANO ile paylaşılır. RAO digital giriş/çıkış olarak kullanılmak isteniyorsa, bu pin üzerindeki Analog özellik devre dışı bırakılmalıdır

```
_ANSAO = 0;      //ANSELAbits.ANSAO=0; disable ADC (ANO)
_TRISAO = 1;     // TRISAbits.TRISAO=1; configure as input
```

```
_ANSBO = 0;      //disable ADC (AN2)
_TRISBO = 0;     // TRISBbits.TRISBO=0; configure as output
```

Port Konfigürasyon Makroları (1)

- PIC'deki çoğu pinin birden fazla görevi vardır -> Bir pinde sadece bir görev kullanılır.

Örnek: RB15 pininin sayısal çıkış olarak ayarlanması

```
; Configure RB15 as output
BCLR TRISB, #15
; Initialize RB15 to output zero
BCLR LATB, #15
; Disable RB15 analog
BCLR ANSELB, #15
; Disable RB15 open drain
BCLR ODCB, #15
```

Makrolar

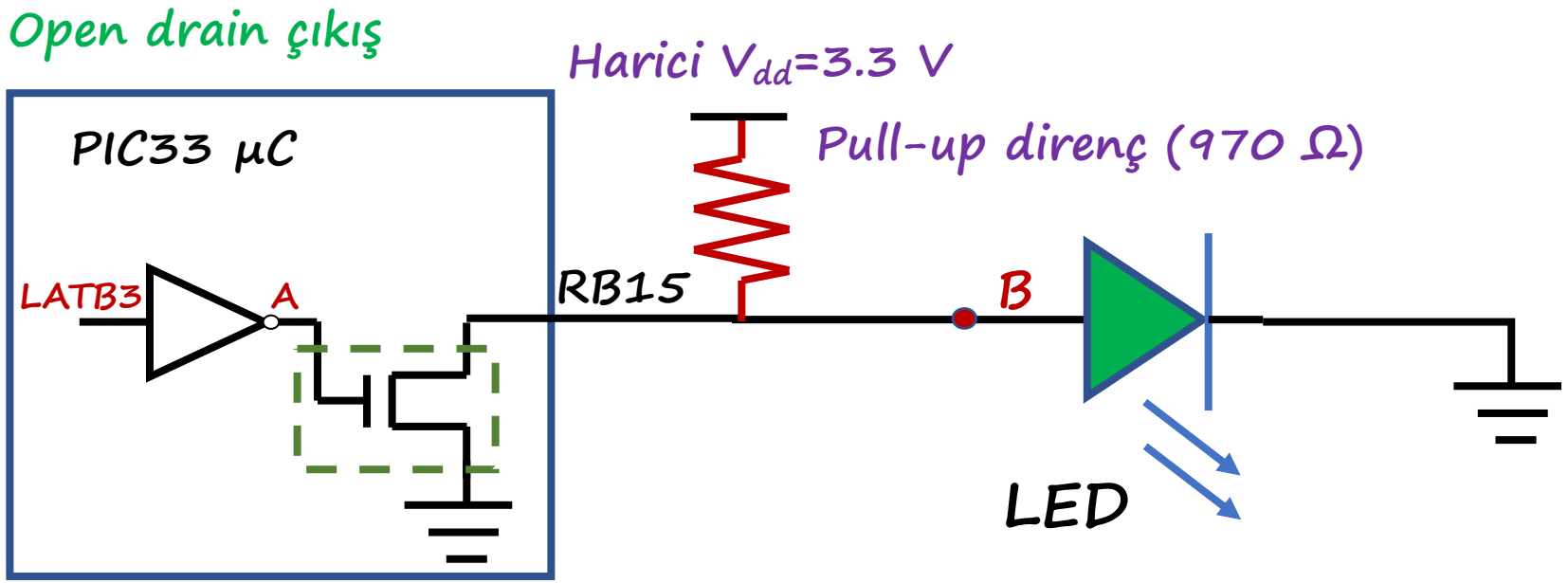
CONFIG_RB15_AS_DIG_OUTPUT( );

Port Konfigürasyon Makroları (2)

- Makrolar pin yapılandırmasını kolaylaştırır.
- Pin konfigürasyonları için makrolar pic24_ports.h dosyasına dahil edilmiştir.
- Pin konfigürasyonları için Makrolar hakkında örnekler:
  A. ENABLE_RB15_PULLUP ( );
  B. DISABLE_RB15_PULLUP ( );
  C. ENABLE_RB13_OPENDRAIN ( );
  D. DISABLE_RB13_OPENDRAIN ( );
  E. CONFIG_RB8_AS_DIG_OD_OUTPUT ( );

LED Sürme— Donanım

- LATB15 = 0 —> A = 1 —> Transistör kesimdedir —> RB15 tranzistör üzerinden toprağa bağlanmıştır —> B = 0 —> LED söner.
- LATB15 = 1 —> A = 0 —> Transistör iletimdedir—> RB15 harici pull-up direnci üzerinden harici V_{dd} kaynağına bağlanmıştır —> B =1 = V_{dd} (3.3 V) —> LED yanar.



LED Sürme— Yöntem 1

```
# include "pic24_all.h"
```

```
void a_delay (void)
{
  uint16_t u16_i, u16_k;
  for (u16_k = 1800; u16_k > 0; --u16_k )
    {
     for (u16_i = 1200; u16_i > 0; --u16_i);
    }
}
```

Basit bir gecikme programı

```
void main (void)
{
    CLOCK_Initialize();
  // Enable open drain
  _ODCB15 = 1;
  // Configure RB15 as output
  _TRISB15 = 0;
  // RB15 initially low (LED off)
  _LATB15 = 0;
while (1)
  {
     //Let  RB15  maintains  current
    status a certain time
    a_delay ();
    //Toggle RB15 output (Toggle LED)
    _LATB15 = !_LATB15;
  }
}
```

LED Sürme — Yöntem 2

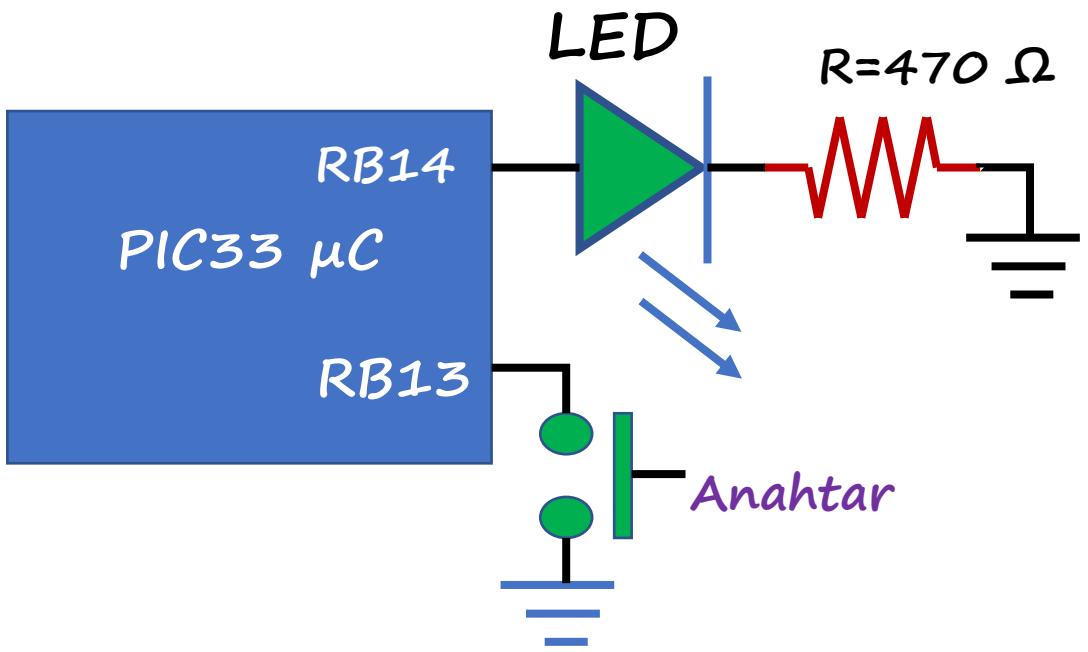
Kodun anlaşırlığını makrolar kolaylaştırır

```
# include "pic24_all.h"

#define LED _LATB15

void config_led () {
 CONFIG_RB15_AS_DIG_OUTPUT ();
 ENABLE_RB15_OPENDRAIN ();
}
void main (void)
{
        configClock ();
        config_led ();   //Macros: configure RB15 as open drain output
        LED = 0;         // Initially turn off LED
        while (1)
           { Bu, 'header' dosyasında tanımlanan bir gecikme fonksiyonudur. Doğrudan çağırabiliriz.
              DELAY_MS (250);  //LED maintains current status 250 ms
                LED = !LED;          //Toggle LED
           }
}
```

Push Button Anahtar Uygulaması (Donanım)



- Aşağıdaki adımları uygulayınız:

1. Anahtara bas ve bırak: LED' i yak
2. Anahtara tekrar bas ve bırak : LED'i söndür
3. Yukarıdaki iki adımı tekrarlayınız

Çözümleme:

- RB13 pinine harici bir pull-up direnci ve V_{dd} uygulanmadı

  ↓

  RB13, anahtarın durumunu algılamak için zayıf pull-up girişi olarak yapılandırılmalıdır
- RB14 pinine harici pull-up direncini ve V_{dd} kaynağını bağlayınız

  ↓

  RB14 LED' i sürmek için RB14 normal çıkış modunda kullanılmalıdır

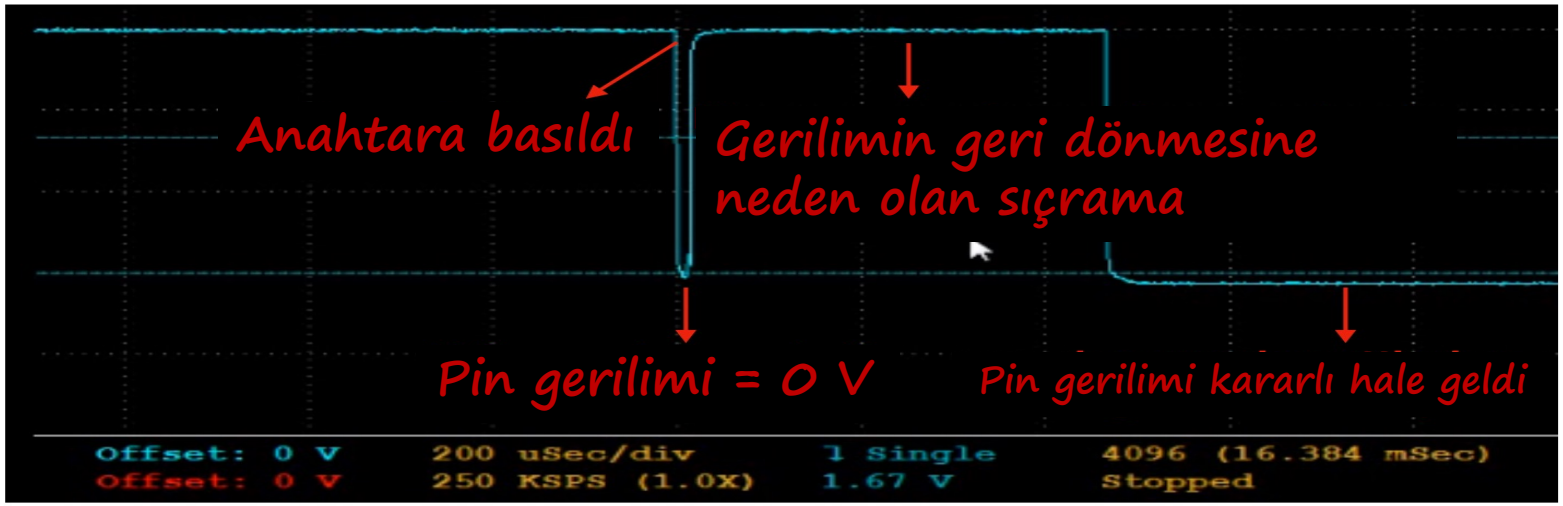
Push Button Anahtar Uygulaması (Yazılım)

```
# include "pic24_all.h"
#define LED _LATB14
// Configure RB14 as normal digital output
#define CONFIG_LED() CONFIG_RB14_AS_DIG_OUTPUT ()
#define SW _RB13
// If SW1 = _RB13 = 0 —> Switch is pressed
#define SW_PRESSED() SW == 0
//If SW1 = _RB13 = 1 —> Switch is released
 #define SW_RELEASED() SW == 1
void CONFIG_SW ()
{
   CONFIG_RB13_AS_DIG_INPUT ();
   ENABLE_RB13_PULLUP ();
}
```

```
void main (void)
 {        configClock ();
          CONFIG_SW(); // Configure _RB13
     //Delay a short time to enable weak pull-up
          DELAY_US (1);
          CONFIG_LED(); // Configure _RB14
          LED=1;
          while (1)
          {
                    //Wait for pressing SW
                    while (SW_RELEASED());
                    DELAY_MS (15);
                    //Wait for releasing SW
                    while (SW_PRESSED());
                    DELAY_MS (15);
                    LED = !LED;
          }
 }
```

Anahtar sıçramasını önlemek için kısa bir süre gecikme ekleyiniz.

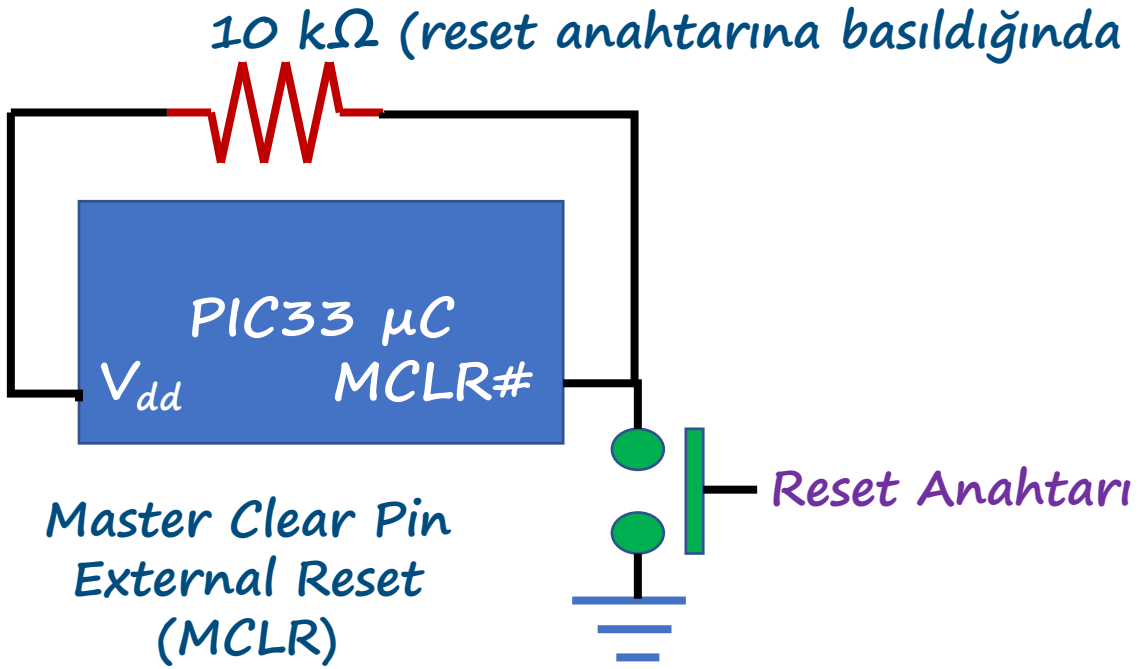
Mekanik Anahtar Sıçraması (Switch Bounce)



- Mekanik anahtarlar, basıldığında birden çok kez durum değiştirebilirler
- Anahtar üzerindeki gerilim sabitlenene kadar giriş pinini tekrar okumayın

```
while (SW_RELEASED ()); // Wait for pressing SW
DELAY_MS (15); <— wait until switch bounce has settled
while (SW_PRESSED ()); // Wait for releasing SW
DELAY_MS (15); <— wait until switch bounce has settled
```

Reset



- PIC, MCLR# pinindeki gerilimi algılar. Gerilim düştüğünde PIC sıfırlanır
- Reset anahtarına basıldığında —> MCLR# toprağa bağlanır —> MCLR# pin gerilimi sıfır olur—> PIC sıfırlanır —> program counter = 0 —> bir sonraki komutun adresi 0 (sıfır) olmaktadır
- Tüm µC' ler, µC' yi bilinen bir duruma zorlamak için bir sıfırlama pinine sahiptir

Reset Türleri

- Sıfırlamaya neden olabilecek birçok neden vardır. Hangi tür sıfırlamanın gerçekleştiğini bilmek için RCON saklayıcısındaki her biti kontrol edebiliriz.
- RCON' daki tüm Sıfırlama bayrak bitleri, kullanıcı yazılımı tarafından birlenebilir veya temizlenebilir

Flag Bit	Set by:	Cleared by:
TRAPR (RCON<15>)	Trap conflict event	POR, BOR
IOPUWR (RCON<14>)	Illegal opcode or initialized W register access	POR, BOR
CM (RCON<9>)	Configuration Mismatch	POR,BOR
EXTR (RCON<7>)	MCLR# Reset	POR
SWR (RCON<6>)	`reset` instruction	POR, BOR
WDTO (RCON<4>)	WDT time-out	`pwrsav` instruction, `clrwdt` instruction, POR,BOR
SLEEP (RCON<3>)	`pwrsav #0` instruction	POR,BOR
IDLE (RCON<2>)	`pwrsav #1` instruction	POR,BOR
BOR (RCON<1>)	BOR	n/a
POR (RCON<0>)	POR	n/a

Power on reset (işlemciye güç verildiğinde meydana gelir)

Brown-out reset (V_{dd} gerilimi bir değerin altına düştüğünde, meydana gelir)

Sıfırlama Nedenini Kontrol Eden Alt Program

```
void printResetCause(void) {
  if (_SLEEP) {
    outString("\nDevice has been in sleep mode\n"); _SLEEP = 0;
  }
  if (_IDLE) {
    outString("\nDevice has been in idle mode\n");  _IDLE = 0;
  }
  outString("\nReset cause: ");
  if (_POR) {
    outString("Power-on.\n");  _POR = 0; _BOR = 0; //clear both
  } else { //non-POR causes
   if (_SWR) {
    outString("Software Reset.\n");            _SWR = 0;     }
   if (_WDTO) {
    outString("Watchdog Timeout. \n");         _WDTO = 0;    }
   if (_EXTR) {
    outString("MCLR assertion.\n");            _EXTR = 0;    }
   if (_BOR) {
    outString("Brown-out.\n");                 _BOR = 0;     }
   if (_TRAPR) {
    outString("Trap Conflict.\n");             _TRAPR = 0;   }
   if (_IOPUWR) {
    outString("Illegal Condition.\n");         _IOPUWR = 0;  }
   if (_CM) {
    outString("Configuration Mismatch.\n");    _CM = 0;      }
  }//end non-POR causes
  checkDeviceAndRevision();
  checkOscOption();
}
```

A status bit is cleared if it has been set.

Print status on processor ID and revision, and clock source.

Watchdog Timer (WDT)

- Watchdog zamanlayıcısı, µC arızalarını tespit etmek ve düzeltmek için kullanılan bir zamanlayıcıdır.
- Normal çalışma sırasında programcı, zaman aşımına uğramasını önlemek için watchdog zamanlayıcısını düzenli olarak sıfırlar.
- Bir donanım hatası veya program hatası varsa, µC watchdog zamanlayıcısını sıfırlayamaz, zamanlayıcı zaman aşımına uğrar ve ardından µC'yi sıfırlamak için bir zaman aşımı kesmesi (interrupt) oluşturur.

WDT Özellikleri (1)

- WDT, saat olarak bağımsız ve serbest çalışan RC osilatörü kullanır. Bu satin çalışma frekans değeri 32.768 kHz' dir, normal saat durduğunda bile çalışır.
- WDT zaman aşımı (timeout), sayıcı maksimum değerden sıfıra düşdüğünde meydana gelir. Zaman aşımı süresi aşağıdaki gibi hesaplanabilir:

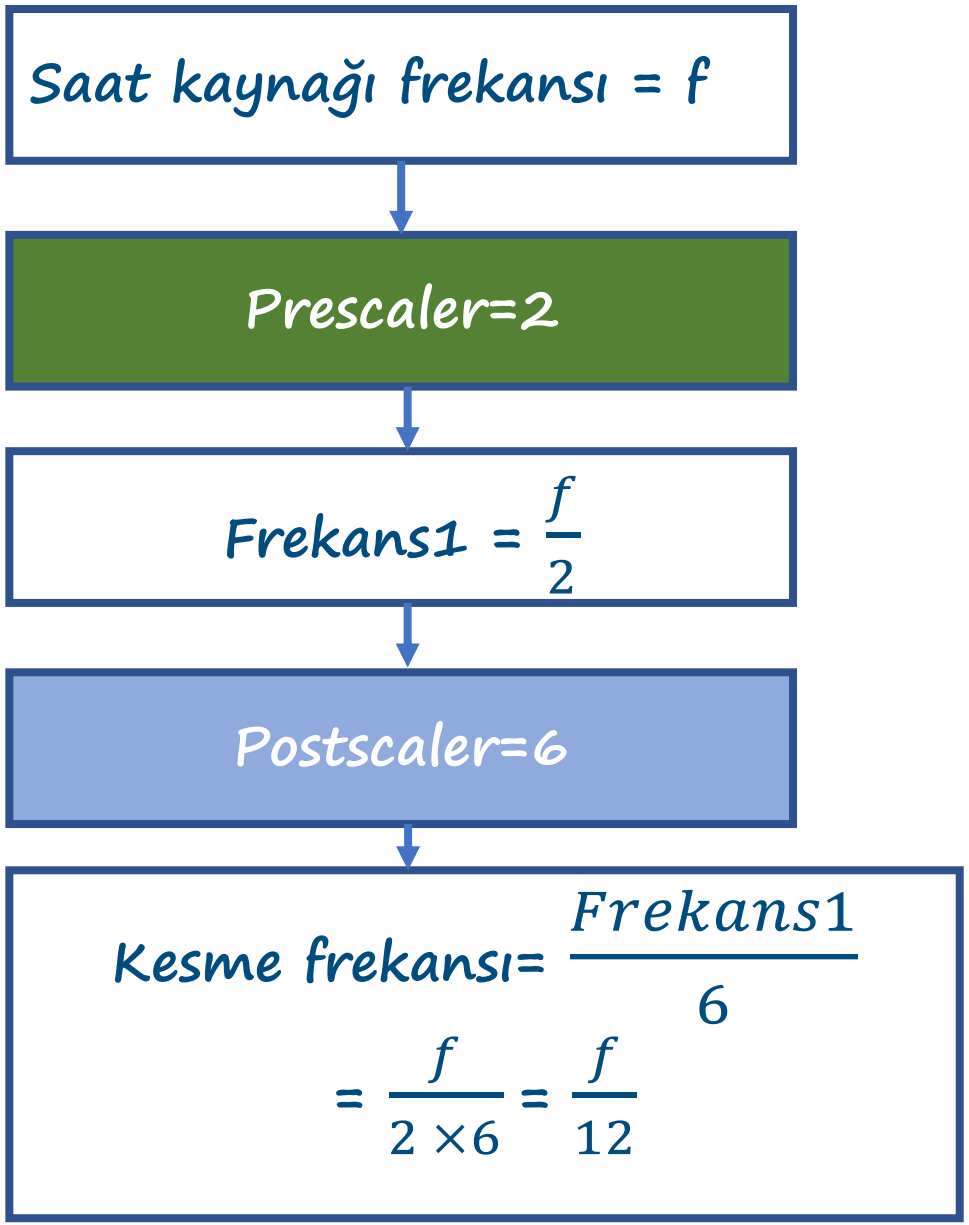
WDT zaman aşımı = Saat Periyodu × WDT önbölücü (presclaer) × WDT sonbölücü (postscaler)

$$= \frac{1}{32.768\ kHz} \times WDT\ prescaler \times WDT\ postscaler$$

Prescaler, WDT zamanlayıcısına girmeden önce saat kaynağı frekansının bölünmesini sağlayan bir frekans bölücüdür

Postscaler bir sayıcıdır ve WDT' nin ne kadar sıklıkta zaman aşımı kesmesini üreteceğini belirler

WDT Özellikleri (2)



- WDT önbölücüsü 32 ya da 128, olabilir, yalnızca iki seçenek vardır:
  A. WDTPRE = 0 —> WDT prescaler = 32
  B. WDTPRE = 1 —> WDT prescaler = 128
- 15 bitlik WDTPOST ayarlanarak WDT postcaler 1, 2, 4, 8, …, 2^{15} olabilir.
- WDTPRE = 0 ve WDTPOST = 1 olduğunda WDT minimum zaman aşımı süresini alır:

$$\text{WDT timeout} = \frac{1}{32.768\ kHz} \times 32 \times 1 \approx 1\ ms$$

- WDTPRE = 1 ve WDTPOST = 2^{15} olduğunda WDT maksimum zaman aşımı süresini alır:

$$\text{WDT timeout} = \frac{1}{32.768\ kHz} \times 128 \times 2^{15} \approx 131\ s$$

- WDTPRE ve WDTPOST hafızanın ROM bölgesinde bulunan konfigürasyon bitleridir.

WDT Kullanımı (1)

Önemli !

- Normal çalışma sırasında bir WDT zaman aşımı, işlemciyi sıfırlayacaktır.
- Uyku veya boş mod sırasında bir WDT zaman aşımı, işlemciyi uyandırır.

- _SWDTEN biti WDT' yi etkinleştirmek/devre dışı bırakmak için kullanılabilir
  A. _SWDTEN = 0 —> WDT devre dışı bırakıldı.
  B. _SWDTEN = 1 —> WDT etkinleştirildi ve saymaya başladı.

- clrwdt komutu WDT zamanlayıcısını temizler, zaman aşımını önler.

WDT Kullanımı (2)

- **Hata düzeltme:** µC, bir çevre biriminin (ör. klavye) yanıtını beklemek üzere tasarlanmışsa. WDT, belirli bir zaman diliminde bir yanıt geri gelmezse µC'yi sıfırlayarak denetleyiciyi sonsuz bir bekleme döngüsünden çıkarabilir.

- **Uykudan uyandırma:** µC bir uyku moduna geçirilmişse, WDT, WDT zaman aşımı süresi geçtikten sonra denetleyiciyi uyandırabilir.

Güç Tasarruf Modları

- PIC, birkaç farklı güç tasarrufu modu sağlar.
- Uyku (Sleep) modu:
  A. CPU ve tüm çevre birimleri çalışmayı durdurur.
  B. WDT zaman aşımı ve harici kesme ile uyandırılabilir.
  C. pwrsav #0 komutu uyku moduna girmek için kullanılır.
- Bekleme (idle) modu:
  A. CPU çalışmayı durdurur.
  B. Çevre birimleri hala çalışabilir (örneğin UART üzerinden veri alınabilir).
  C. pwrsav #1 komutu bekleme moduna girmek için kullanılır.
- Şekerleme (Doze) modu:
  A. CPU ve çevre birimleri hala çalışabilir.
  B. CPU saati 'doze prescaler-önbölücü (2, 4, ..., 128)' tarafından bölünür.
  C. Çevresel ara birimlerin saati etkilenmez. CPU daha düşük frekansta çalışır, fakat çevre birimleri tam hızda çalışır.

Akım Tüketimi

Mod	PIC24@40MHz (mA)	PIC24@16MHz (mA)
Normal	42.3	5.6
Sleep	0.03	0.004
Idle	17.6	2.0
Doze/2	32.2	4.0
Doze/128	17.9	2.0