# TIVA ARM M4

Burhan BARAKLI
Department of Electrical and Electronic Science
Technion— Institute of Engineering
Technion City, Sakarya 54000, Sakarya

11 Kasım 2018

# İçindekiler

**Özet**

Sonra bayanlar beyler ...

# Bölüm 1

# Başlangıç Ayarlarımız

## 1.1 Giriş

This is time for all good men to come to the aid of their party!

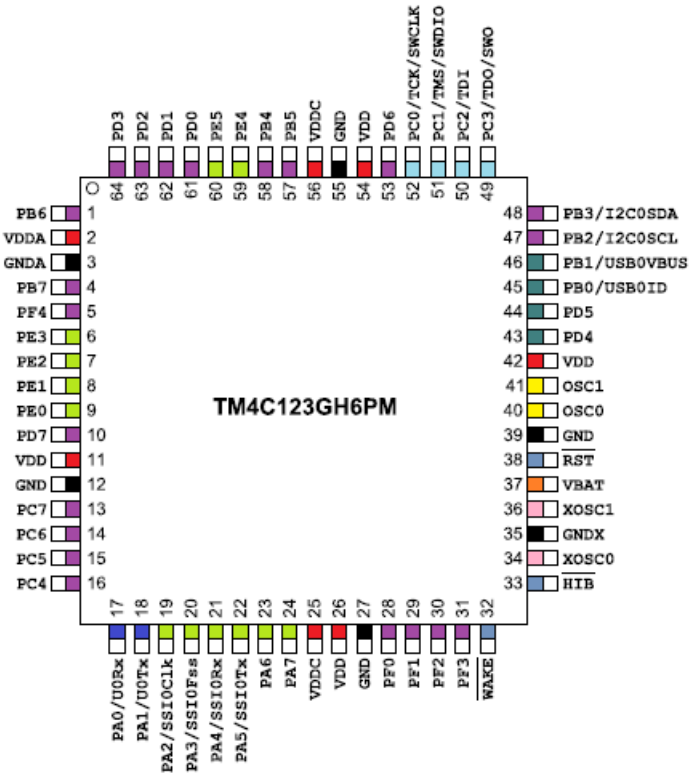
# Bölüm 2

# GPIO-General Purpose Input Output

GPIO, Genel Amaçlı Giriş / Çıkışlar anlamına gelir; bu, sinyalleri alma ve iletme yeteneğine sahip bir modül veya çevre birimi anlamına gelir. Sayısal sinyallerle çalışırlar, fakat diğer çevre fonksiyonlarını (ADC, SSI, UART, vb.) bir mux entegresi ile kullanabilirler.

Mikroişlemci kartları üzerinde elektronik sistemlerle haberleşme ve bu cihazların kontrolü amacıyla kullanılan portlar, genel amaçlı giriş çıkış portları (General Purpose Input/Output) GPIO olarak adlandırılır. GPIO portları mikro denetleyici cihazların çevresel cihazlarla iletişim kurmakta en çok kullandıkları yoldur. Bir GPIO portu aynı anda hem **giriş** hem de **çıkış** hattı olarak kullanılabilir. Örneğin bir LED diyotu kontrol etmek amacıyla çıkış hattı kullanılırken, bir buton veya anahtara basılıp basılmadığını belirlemek için giriş hattı kullanılır. GPIO portlarında low durumunu 0V, high durumunu ise pozitif bir gerilim (genellikle 3.3V) belirtir. Girişe uygulanan voltaj 1.7V-5.5V aralığında olmalıdır. 5.5V'dan daha büyük gerilimler donanımınıza zarar verecektir. Port değeri boolean (lojik seviye) bir değer olarak saklanır.

## 2.1 Tiva GPIO Yapısı




Şekil 2.1: TM4C123GH6PM pinleri

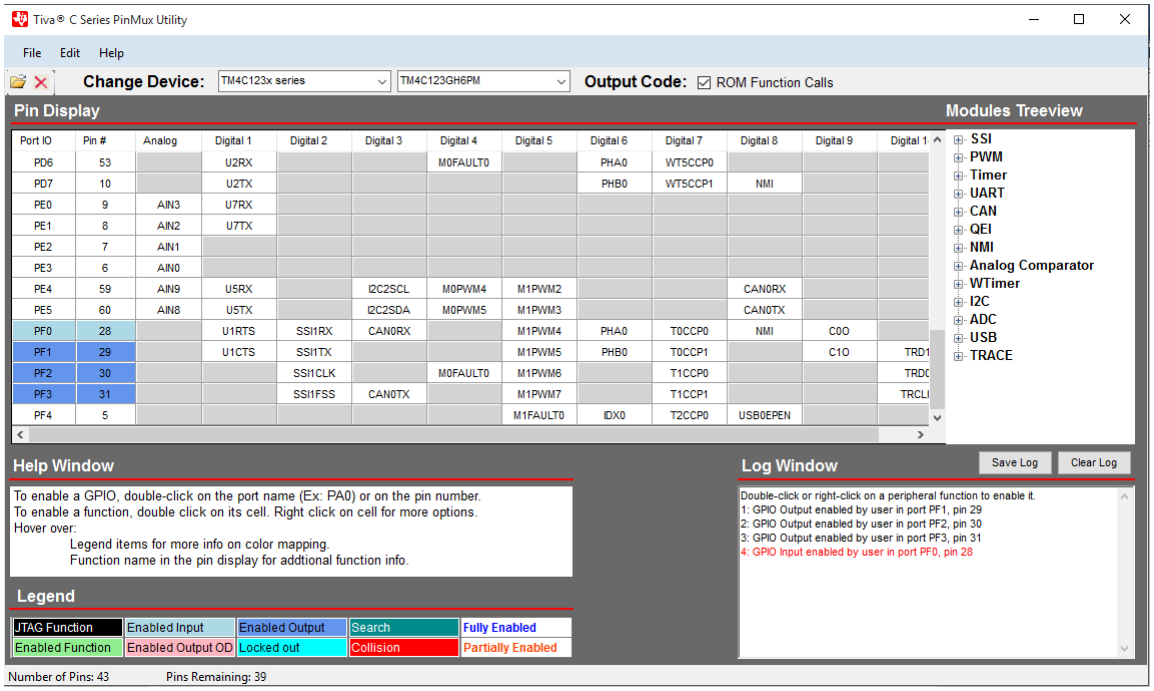
TM4C123GH6PMI Tiva mikrokontrolör kartı, 64 adet pini mevcuttur ve 40'dan fazla sayısal giriş veya çıkış olarak kullacağımız pine sahiptir. Tiva C mikro düşük güç ARM Cortex M4 MCUS ve tipik olarak 3.3V

çalışır ve böylece GPIO pinlerinin mantık seviyelerini tahmin edebilirsiniz. Ancak, birkaç GPIO (PB0, PB1, PD4 ve PD5) hariç tüm GPIO pinleri 5V toleranslıdır.

Bazı GPIO pinlerine Launchpad kartında erişilemezken, diğerleri fiziksel olarak mevcut değildir, örn. PF7!

Maksimum sınır 25mA olmasına rağmen, 10 - 15mA'nın ötesindeki GPIO pinlerini zorlamayın. Yüksek güç yüklerini sürmek için opto izolatörler, FET'ler ve BJT'ler gibi harici anahtarlama elemanlarını kullanın.

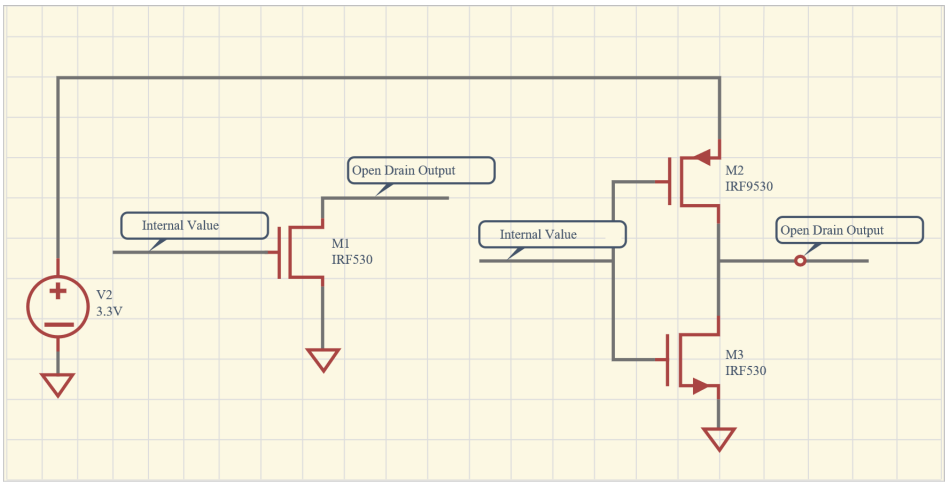
GPIO pinleri alternatif fonksiyonlara sahiptir ve bu yüzden gerekli GPIO pinlerini bulmak için Pinmux yazılımını kullanın. Ayrıca, diğer fonksiyon pinlerini de GPIO kütüphanesinin altındaki MikroC derleyicinin yardım bölümünde kontrol edebilirsiniz. Derleyicide, GPIO yazar ve CTRL ile Boşluk tuşlarına basarsanız, GPIO için tüm kullanılabilir seçenekleri görürsünüz.




Şekil 2.2: pinmux programı

Hem GPIO'lar hem de yazmaçlarının hepsi 8 bitlik genişliktedir ve her port piniyle ilgili birçok yazmaç ve seçenek vardır.

### 2.1.1 Giriş Çıkış modları




Şekil 2.3: Giriş çıkış modları

Mikrodenetleyicinin genel amaçlı giriş çıkış (general purpose input/output (GPIO)) pinleri, giriş çıkış bakımından farklı modlarda konfigure edilebilir. Giriş modları arasında pull-up, pull-down dirençleri, hysteresis,

ya da benzeri kombinasyonlar söz konusu olabilir. Benzer şekilde çıkış modları push-pull, high-drive ya da open-drain olabilir.

GPIO giriş modları genellikle, yüksek empedans(high impedance, high-z), pull-up, pull-down ve tekrarlayıcı(repeater) olarak tasarlanır. Yine bazı mikrodenetleyiciler I/O arayüzlerinde histerizis (hysteresis) özelliği de barındırır ve bu sayede suni (bouncing gibi) durum değişiklikleri önlenir.

**Pull Up/Down** Eğer bir giriş, dahili (mikrodenetleyicinin içindeki) pull-up modunda ise o pin içeride bir direnç ile lojik bir seviyesine çekilmiştir. Bu da şu demektir ki, o pin dışarıdan lojik sıfıra sürülene kadar, lojik 1 durumunda kalmaya devam edecektir. Tam tersi durum da pull-down için geçerlidir. Eğer bir giriş, dahili pull-down modunda ise o pin içeride bir direnç ile lojik sıfır seviyesine çekilmiştir. Bu da şu demektir ki, o pin dışarıdan lojik bire sürülene kadar, lojik sıfır durumunda kalmaya devam edecektir.

**Repeater:** Bazı mikrokontrolörlerde bazı pinlerin durumu dinamik olarak pull-up ya da pull-down olarak değiştirilebilmektedir ve bunun için repeater mod kullanılır.

**High Impedance:** Bir giriş pini yüksek empedans(high impedance, high-z) durumunda olduğunda pinin durumu, o pin lojik sıfıra ya da lojik bire bağlanana kadar bilinemez.

**Floating:** Yüksek empedans yapısındaki bir pin, lojik sıfıra ya da lojik bire bağlanmadıysa, pin floating (belirsiz, salınan) moddadır denir.

**Tri-stated:** Üç durumlu anlamındaki bu mod, floating ile aynı durum için kullanılır.

**Hysteresis** Bir pinin lojik sıfırda mı lojik birde mi olduğuna karar verebilmek için genelde bir eşik değeri kullanılır. Ancak bu eşik değeri bir aralık ifade etmediğinden eşik değerinin çok az üstü de lojik bir olur, epey üstü de lojik bir olur. Benzer şekilde eşik değerinin çok az altı da lojik sıfır olur, epey altı da lojik sıfır olur. Gerçek hayatta bu durum bouncing (sekme, salınma) problemini doğurur. Siz lojik birdeki bir pini lojik sıfıra çekerken pin eşik değerinin etrafında kararsız bir duruma girer ve ardışıl olarak 1,0,1,0,1,0 durumları görülür. Bunu önlemek için güvenli bir aralık belirlenmiştir.

**Push-Pull** Push-pull bir çıkış hem source hem de sink akımı akıtabilir. Bu da pin çıkışı sıfıra da çekilse, bire de çekilse o pin üzerinden akım akıtabilmeyi sağlar. TTL ve CMOS devreler push-pull çıkış kullanır.

Şekil 2.3'de **Open-Drain** yazan kısmı görebilirsiniz. Gerilimle sürülen bir transistör türü olan MOSFET'i tanıyanlar için "open-drain" kavramı zaten açıklamaya mahal gerek bırakmayacak kadar anlaşılırdır. Ancak MOSFET'i bilmeyenler için anlatmak gerekirse bir MOSFET üç pine sahiptir: gate(kapı, giriş), source (kaynak), drain(akaç). Open-drain durumda source toprağa bağlıdır,gate içeriden sürülmüş durumdadır ve drain açıktadır. Open-drain çıkış yalnızca sink akımı akıtabilir yani dışarıdan akım çekebilir. Dışarıya doğru akım basamaz. Esasen bu da iki durumda kalabilmesine imkan verir: düşük empedans ve yüksek empedans.

**Output open-drain** Pinin çıkış bacağı, P-Mos ile N-Mos mosfetleri arasında bulunmaktadır. Eğer çıkışı open-drain ayarlarsanız, pin bacağı GND'ye bağlanır. VDD ile arasında sonsuz empedans oluşur. Bu durumda pini high olarak çıkış vermek için dışarıdan pull-up dirençleri eklemeniz gerekmekte. Bu özellikle, çıkışa bağlanacak olan devre elemanının daha fazla akım çekmesi sağlanabilmektedir. Röle gibi akım gerektiren devre elemanlarını sürmek için kullanılır. Akım mikroişlemciden değil, pull-up direnci üzerinden çekilir.

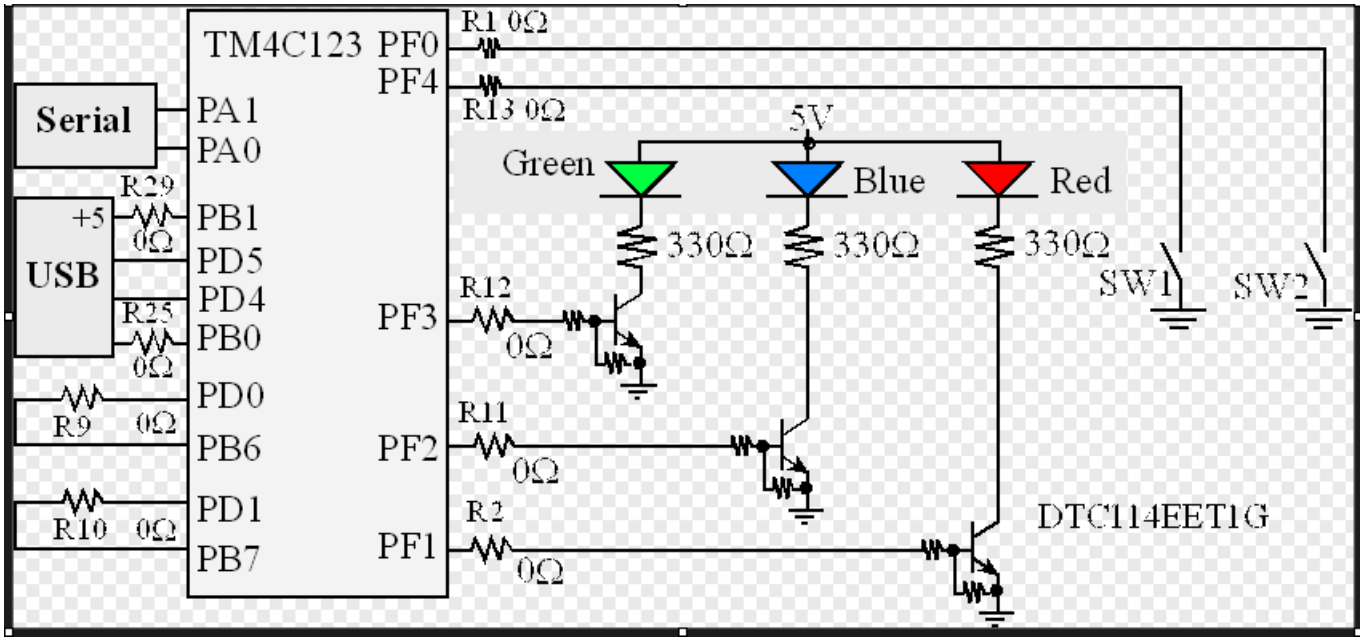
**Output push-pull** Pin bacağı P-Mos mosfeti ile VDD pinine bağlanır. Akım gerektirmeyen işlemlerde, çıkışı logik 1 yapmak için bu mod kullanılır.

**High Drive** High Drive (yüksek sürüşlü) pinler esasen yüksek akım verebilen push-pull pinlerdir. Normal bir push pull pin +/- 8mA akım akıtabilirken high drive pinler 40mA'e kadar akım akıtabilir. Ancak genelde elektriksel karakteristik datasheette belirtilir. Eğer pin ile doğrudan yüksek akım çeken elemanlar sürülecekse, pinin üst sınırına kadar akım çekecek cihazlar, ek bir devre olmasızın high drive pinler ile sürülebilirler.

### 2.1.2 Tiva Kitindeki örnek Giriş ve Çıkışlar

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
        // 50 Mhz
```

```
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); // çevre birimi aktifleştirme
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3); //
       portf'in 1,2,3 pinleri output olarak ayarlandı
    while(1)
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2); //
           kırmızı led, pf1=3.3V
    }
}
```




Şekil 2.4: portf tiva

### 2.1.3 Derste yapılan örnekler

Şimdilik örnekleri verip bazı açıklamalar ile devam edeceğim. Zamanım bu aralar kısıtlı olduğundan bazı açıklamları daha sonradan güncelleyeceğim. gpio.h ilgilendiğimiz kütüphane.

```
extern int32_t GPIOPinRead(uint32_t ui32Port, uint8_t ui8Pins); // geriye değer
   döndürüyor ve 2 par. alıyor.
extern void GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val);// 3 par
   . alıyor
extern void GPIOPinTypeGPIOInput(uint32_t ui32Port, uint8_t ui8Pins); //2 par. alıyor
extern void GPIOPinTypeGPIOOutput(uint32_t ui32Port, uint8_t ui8Pins); // 2 par.
   alıyor
```

```
GPIOPinTypeGPIOOutput(uint32_t ui32Port, uint8_t ui8Pins);
extern void GPIOPinTypeGPIOInput(uint32_t ui32Port, uint8_t ui8Pins); //2 par. alıyor
```

Bu fonksiyon ilgilendiğimiz portun pinlerini çıkış veya giriş olarak ayarlıyor. `GPIO_PORTF_DIR_R` registirini incelediğimizde çıkışlar için 1 değeri, girişler için 0 değerini aldığını göreceksiniz.

```
extern int32_t GPIOPinRead(uint32_t ui32Port, uint8_t ui8Pins); // geriye değer
   döndürüyor ve 2 par. alıyor.
extern void GPIOPinWrite(uint32_t ui32Port, uint8_t ui8Pins, uint8_t ui8Val);// 3 par
   . alıyor
```

Bu fonksiyonlar ile ilgilenilen pinlerin değeri okunur veya pinin lojik seviyesi ayarlanır. `GPIO_PORTF_DATA_R` registerında ise portların lojik durumlarını inceleyebilirsiniz.

```
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
```

Kod içinde kullandığımız | karakteri bitsel **or** operatörüdür.`GPIO_PIN_1`=2, `GPIO_PIN_2`=4, `GPIO_PIN_3`=8 değerlerini almaktadır. Üç değerin or operatörü ile çıktısı 14'tür. `GPIO_PORTF_BASE`=0x40025000'dir.

```
GPIOPinTypeGPIOOutput(0x40025000, 14);
```

fonksiyonun bir üstteki kod bloğundan bir farkı bulunmamaktadır.

Aşağıdaki kodda pin4 input olarak ayarlanıyor. Fakat lojik seviyesi 1 olarak ayarlanıyor. Butona tıklandığından lojik seviye 0'a çekilmektedir. Bu nedenle 0 testini gerçekleştirmemiz gerekecektir. Kodu inceleyiniz.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_4MA,
       GPIO_PIN_TYPE_STD_WPU);
    // pin 4, pull-up olarak ayarlandı.
    while(1)
    {
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)==0)
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2);
        }
        //veya
        if(!GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4))
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2);
        }
        //veya
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)!=16)
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2);
        }
        // bu üç if yapısı aynı mantıkta çalışmaktadır.
    }
}
```

Aşağıdaki fonksiyon giriş olarak ayarlayacağımız yapının pull-up veya pull-down durumunu ayarlamak için kullanacağız. Kullandığımız kitte portf'nin 4. pini butonla direkt olarak GND olarak ayarlanmıştır. Bu nedenle biz içerdeki lojik seviyeyi 1 olarak ayarlamalıyız.

```
    extern void GPIOPadConfigSet(uint32_t ui32Port, uint8_t ui8Pins,uint32_t
       ui32Strength, uint32_t ui32PadType);
```

Parametreler aşağıdaki gibi gpio.h'da düzenlenmiştir.

```
//*****************************************************************************
//
// Values that can be passed to GPIOPadConfigSet as the ui32Strength parameter,
// and returned by GPIOPadConfigGet in the *pui32Strength parameter.
//
//*****************************************************************************
#define GPIO_STRENGTH_2MA       0x00000001  // 2mA drive strength
#define GPIO_STRENGTH_4MA       0x00000002  // 4mA drive strength
#define GPIO_STRENGTH_6MA       0x00000065  // 6mA drive strength
#define GPIO_STRENGTH_8MA       0x00000066  // 8mA drive strength
#define GPIO_STRENGTH_8MA_SC    0x0000006E  // 8mA drive with slew rate control
#define GPIO_STRENGTH_10MA      0x00000075  // 10mA drive strength
#define GPIO_STRENGTH_12MA      0x00000077  // 12mA drive strength

//*****************************************************************************
//
```

```
// Values that can be passed to GPIOPadConfigSet as the ui32PadType parameter,
// and returned by GPIOPadConfigGet in the *pui32PadType parameter.
//
//*****************************************************************************
#define GPIO_PIN_TYPE_STD       0x00000008  // Push-pull
#define GPIO_PIN_TYPE_STD_WPU   0x0000000A  // Push-pull with weak pull-up
#define GPIO_PIN_TYPE_STD_WPD   0x0000000C  // Push-pull with weak pull-down
#define GPIO_PIN_TYPE_OD        0x00000009  // Open-drain
#define GPIO_PIN_TYPE_ANALOG    0x00000000  // Analog comparator
#define GPIO_PIN_TYPE_WAKE_HIGH 0x00000208  // Hibernate wake, high
#define GPIO_PIN_TYPE_WAKE_LOW  0x00000108  // Hibernate wake, low

//*****************************************************************************
```

Gpio portf'in 0. pini Gpio ve jtag arasında bir paylaşım söz konusudur. Bu nedenle 0. pine ait kilidi portf için şimdilik kaldırmalıyız. Aşağıda verilen kodlar 0. pini giriş ayarlamadan önce koymalıyız.

```
        HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
        HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
        HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
```

Artık 0. pini input olarak kullanabilmekteyiz.

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h" // bunu unutmayın !!!!
int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);

    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_4MA,
       GPIO_PIN_TYPE_STD_WPU);
    // pin 0 ve 4, pull-up olarak ayarlandı.

    while(1)
    {
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)==0)
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2);
        }
        else if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0)==0)
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4);
        }
    }
}
```

Derste tartıştığımız diğer bir konu inputların ikisini birden okuduğumuzda gelen değerler ile alakalıydı. Bir üstte tek tek girişler okunmuştu fakat burada durum biraz farklıydı. Kodu inceleyin ve butonlara tıkladığımızda gelen değerleri belirleyin.

```
#include <stdint.h>
```

```c
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h" // bunu unutmayın !!!!
int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);

    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_4MA,
       GPIO_PIN_TYPE_STD_WPU);
    // pin 4, pull-up olarak ayarlandı.
    while(1)
    {
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==16) // sw1, pf.0=0
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2);
        }
        else if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==1) //sw2 pf.4=0
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 4);
        }
    }
}
```

### 2.1.4 c ve Fonksiyon oluşturma

Oluşturabileceğimiz değişken tipleri (int,double,float) klasik c ve türevleri programlama dillerindeki gibi oluşturabiliriz. Bunun için internette bir sürü kaynak bulabilirsiniz. CCS'de bulunan değişik tipte hazır oluşturulmuş tiplerde mevcuttur. stdint.h kütüphanesinde oluşturulan değişkenler aşağıdaki şekildedir. `defined(_TMS320C` başlıklarına bakabilirsiniz. Ancak derste hep klasik değişken tiplerini oluşturacağız.

```c
#elif defined(_TMS320C6X) || defined(__ARM_ARCH) || defined(__ARP32__) || \
      defined(__PRU__)    || defined(__FROZEN__)
    typedef   signed char   int8_t;
    typedef unsigned char  uint8_t;
    typedef          short  int16_t;
    typedef unsigned short uint16_t;
    typedef          int    int32_t;
    typedef unsigned int   uint32_t;
    .
    .
    .
#if defined(__ARM_ARCH) || defined(_TMS320C6X) || defined(__ARP32__) || \
    defined(__MSP430__) || defined(__PRU__)    || defined(__FROZEN__)
    typedef          long long  int64_t;
    typedef unsigned long long uint64_t;
    .
    .
    .
```

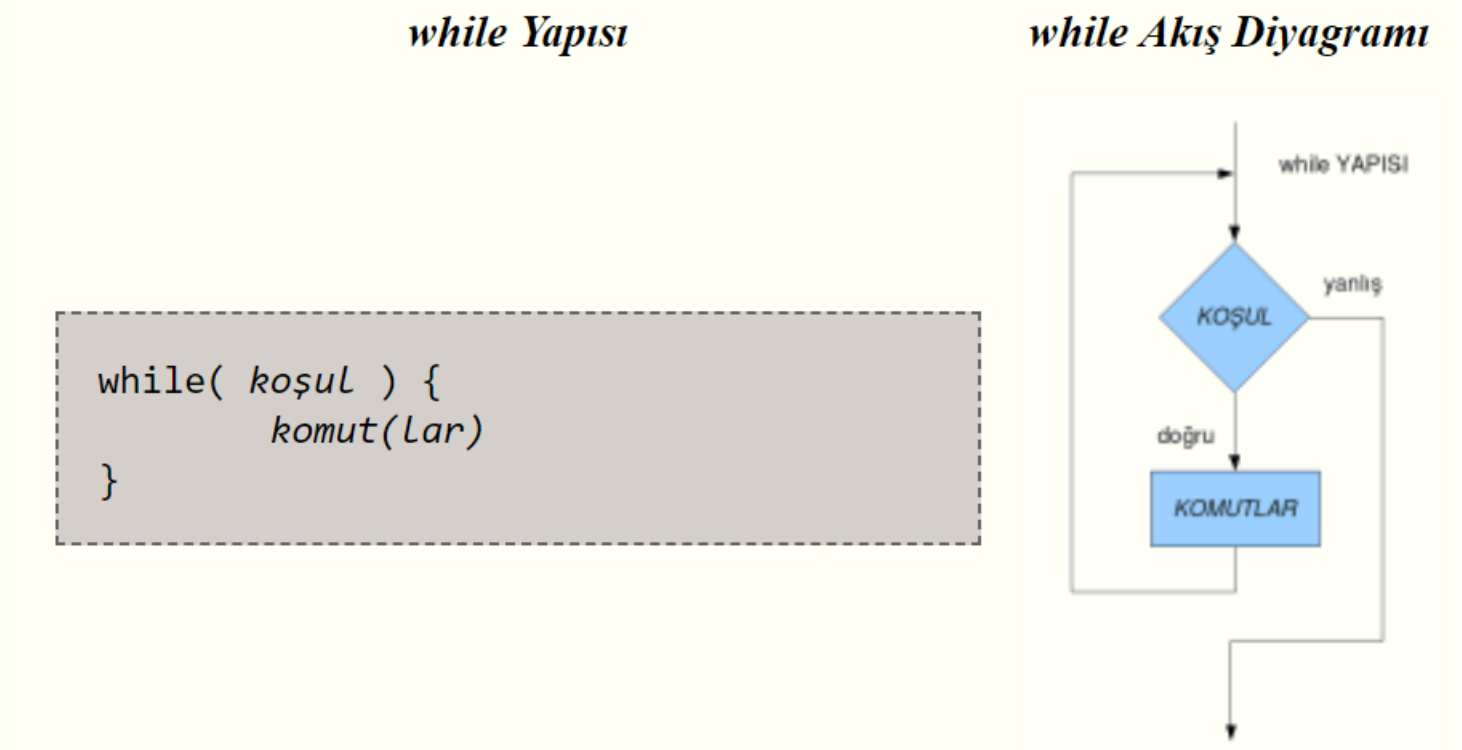
```
TIP               BOYUT          MIN             MAX
=============================================================
char            : 1 byte(s)     -128            127
short           : 2 byte(s)     -32768          32767
int             : 4 byte(s)     -2147483648     2147483647
unsigned int    : 4 byte(s)                     4294967295
long            : 4 byte(s)     -2147483648     2147483647
float           : 4 byte(s)     1.175494e-38    3.402823e+38
double          : 8 byte(s)     2.225074e-308   1.797693e+308
```

Şekil 2.5: Değişken Tipleri ve ram'de kapladıkları boyut

**While, For, Break, Continue, Goto**

while döngüsü, en temel döngü tipimizdir. Bir kontrol ifadesiyle döngünün devam edilip edilmeyeceği kontrol edilirken, scope içinde ( yani ayraç işaretleri arasında ) kalan bütün alan işleme sokulur. İşleme sokulan kod kısmı döngü yapılacak adet kadar tekrar eder.

while döngüsünün genel yapısını ve akış şemasını aşağıda görebilirsiniz:

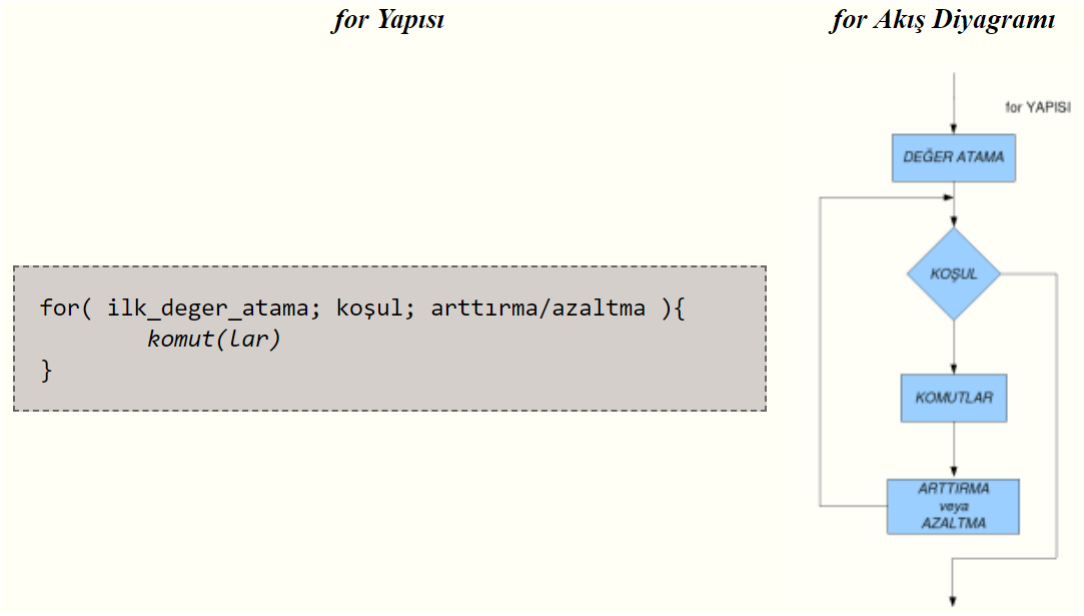



Şekil 2.6: Döngü 1 while

while dışında bir döngü tipi olarak, for yapısı bulunmaktadır. Diğer iki döngüden farklı olarak, for yapısı, yenilemeli-tekrarlamalı (İngilizce iterative) yapılarda kullanıma daha uygundur. Bunu performans anlamında söylemiyorum. Demek istediğim yazım tekniği olarak, for döngüsünün daha kullanışlı olmasıdır. Örneğin birbirini, sürekli tekrar eden işlemlerin yapıldığı Nümerik Analiz gibi alanlar, for döngüsü için iyi bir örnek olabilir. Ancak bu dediklerim sizi yanıltmasın; for döngüsü sadece soyut alanlarda çalışsın diye yaratılmış bir şey değildir.

Programlarda, diğer iki döngüden çok daha fazla for kullanırsınız. Çünkü for sadece matematiksel hesaplama işlemlerinde değil, diziler ( array ) gibi konularda sürekli kullanılan bir yapıdır. Yazımı diğerlerine nazaran daha sade olduğundan, iteratif işlemlerde kullanılması elbette ki tesadüf olarak düşünülemez.

Aşağıda for döngüsünün genel yazımını ve akış diyagramını göreceksiniz:



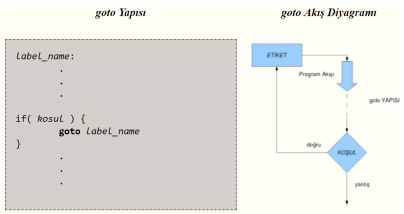

Şekil 2.7: Döngü 2 for

**break Komutu** Bazı durumlarda, döngüyü aniden sonlandırmak isteriz. Bunun için 'break' komutunu kullanırız. Döngü içinde break; komutu görüldüğü anda döngü sonlanır.

**continue Komutu** break komutunun, döngüyü kırmak için olduğundan bahsetmiştik. Bunun dışında işlem yapmadan döngüyü devam ettirmek gibi durumlara da ihtiyacımız vardır. Bunun içinde continue ( Türkçe: devam ) komutunu kullanırız.

goto Yapısı C programlama dilinde bulunan bir başka yapı, goto deyimidir. Koyacağınız etiketler sayesinde, programın bir noktasından bir başka noktasına atlamanızı sağlar. goto, bir döngü değildir ancak döngü olarak kullanılabilir.

**goto,** çalışabilmek için etiketlere ihtiyaç duyar. Etiketler, vereceğiniz herhangi bir isme sahip olabilir. Etiket oluşturmak için bütün yapmanız gereken; etiket adını belirleyip, sonuna iki nokta üst üste eklemek ( : ) ve programın herhangi bir yerine bunu yazmaktır. goto deyimi kullanarak bu etiketleri çağırırsanız, etiketin altında bulunan kodlardan devam edilir. goto ve etiketlere dair genel yapıyı, akış diyagramıyla birlikte aşağıda bulabilirsiniz:




Şekil 2.8: Döngü 3 goto

### Kaydırma ( Shift ) Operatörleri

Kaydırma operatörleri, özellikle Assembly ile uğraşanlara tanıdık gelecektir. Bunları kullanarak son derece hızlı çarpma ve bölme yapılabilir. C'deyse benzer amaçlarla kullanmanız elbette mümkündür. İki çeşit kaydırma operatörü vardır:

1. Sola Kaydırma - Shift Left ( « )
2. Sağa Kaydırma - Shift Right ( » )

Aşağıdaki örnek, sola kaydırma operatörü kullanılarak yapılan bir işlemi göstermektedir. x değişkeni, 10 tabanında 22 sayısını tutmaktadır. 2 adım sola kaydırılması sonucu, sayı 88 olmuş ve y'ye atanmıştır.

```
x = ( 0001 0110 )2 ==> 22
y = x << 2
y = ( 0101 1000 )2 ==> 88
```

### Atama Operatörleri

Bu operatörler bir değişkene, bir sabit vaya bir aritmetik ifade atamak (eşitlemek) için kullanılır. Birleşik atama: bazı ifadelerde işlem operatörü ile atama operatörü birlikte kullanılarak, ifadeler daha kısa yazılabilir. Eğer ifade

değişken = değişken [operatör] aritmetik ifade;

şeklinde ise, daha kısa bir biçimde

değişken [operatör]= aritmetik ifade;

olarak yazılabilir.

| Operatör | Açıklama | Örnek | Anlamı |
|---|---|---|---|
| = | atama | x = 7; | x = 7; |
| += | ekleyerek atama | x += 3 | x = x + 3 |
| -= | eksilterek atama | x -= 5 | x = x - 5 |
| *= | çarparak atama | x *= 4 | x = x * 4 |
| /= | bölerek atama | x /= 2 | x = x / 2 |
| %= | bölüp, kalanını atama | x %= 9 | x = x % 9 |
| ++ | bir arttırma | x++ veya ++x | x = x + 1 |
| -- | bir azaltma | x-- veya --x | x = x - 1 |

Şekil 2.9: Atama Operatörleri

### Matematiksel Fonksiyonlar (math.h)

Matematiksel fonksiyonların hemen hemen hepsi double veri tipindedir. Bu fonksiyonlardan biri program içinde kullanılacaksa math.h başlık dosyası program içine eklenmelidir.

Trigonometrik (sin, cos, tan) fonksiyonlar kendisine parametre olarak gelen değeri radyan olarakkabul eder ve sonucu hesaplar. Eğer açılar derece cinsinden hesaplanması gerekiyorsa şu dönüşüm kullanılanılabilir: Masaüstü bir program olarak c çalıştırıldığında örnek sin değerleri aşağıdaki gibidir. Fakat buradaki işlemler M4 işlemcisinde bir miktar vakit alacaktır. Bunun yerine başka bir çevre birimini ilerde kullanacağız.

radyan = (3.141593/180.0) * derece;

```c
/*  30 dercelik açının sinüs, kosinüs, tanjant ve kotanjant değerleri */
#include <stdio.h>
#include <math.h>
#define PI 3.141593
int main()
{
 double aci = 30.0;
 aci *= PI/180.0; /* radyana çevir */
 puts("30 derecenin");
 printf("sinusu : %lf\n", sin(aci));
 printf("kosinusu : %lf\n", cos(aci));
 printf("tanjanti : %lf\n", tan(aci));
 printf("kotanjanti: %lf\n", 1.0/tan(aci));
return 0;
}
```

| Fonksiyon Bildirimi | Açıklama | Örnek | Sonuç |
|---|---|---|---|
| `int abs(int x);` | x tamsayısının mutlak değerini hesaplar | `abs(-4)` | 4 |
| `double fabs(double x);` | x gerçel sayısının mutlak değerini hesaplar | `fabs(-4.0)` | 4.000000 |
| `int floor(double x);` | x'e (x'den büyük olmayan) en yakın tamsayıyı gönderir | `abs(-0.5)` | -1 |
| `int ceil(double x);` | x'e (x'den küçük olmayan) en yakın tamsayıyı gönderir | `ceil(-0.5)` | 0 |
| `double sqrt(double x);` | pozitif x sayısının karekökünü hesaplar | `sqrt(4.0)` | 2.000000 |
| `double pow(double x, double y);` | $x^y$ değerini hesaplar | `pow(2., 3.)` | 8.000000 |
| `double log(double x);` | pozitif x sayısının doğal logaritmasını hesaplar, ln(x) | `log(4.0)` | 1.386294 |
| `double log10(double x);` | pozitif x sayısının 10 tabanındaki logaritmasını hesaplar | `log10(4.0)` | 0.602060 |
| `double sin(double x);` | radyan cinsinden girilien x sayısının sinüs değerini hesaplar | `sin(3.14)` | 0.001593 |
| `double cos(double x);` | radyan cinsinden girilien x sayısının kosinüs değerini hesaplar | `cos(3.14)` | -0.999999 |
| `double tan(double x);` | radyan cinsinden girilien x sayısının tanjant değerini hesaplar | `tan(3.14)` | -0.001593 |
| `double asin(double x);` | sinüs değeri x olan açıyı gönderir. Açı -pi/2 ile pi/2 arasındadır. | `asin(0.5)` | 0.523599 |
| `double acos(double x);` | cosinüs değeri x olan açıyı gönderir. Açı -pi/2 ile pi/2 arasındadır. | `acos(0.5)` | 1.047198 |
| `double atan(double x);` | tanjant değeri x olan açıyı gönderir. Açı -pi/2 ile pi/2 arasındadır. | `atan(0.5)` | 0.463648 |

Şekil 2.10: math.h

**ÇIKTI**

```
30 derecenin
sinusu    : 0.500000
kosinusu  : 0.866025
tanjanti  : 0.577350
kotanjanti: 1.732051
```

Şekil 2.11: yukarıdaki programın çıktısı

### Fonksiyon

C Programlama Dili fonksiyon olarak adlandırılan alt programların birleştirilmesi kavramına dayanır. Bir C programı bir yada daha çok fonksiyonun bir araya gelmesi ile oluşur. Bu özellik bütün Yapısal Diller'in (C, Fortran, Pascal, ...) temelini oluşturur. Yapısal Diller'e hakim olmak için fonksiyon oluşturmayı ve kullanmayı iyi öğrenmek gerekir.

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h"

//Basit bir fonksiyon, parametresi yok
void portfnin2pininilojik1yap()
{
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
}

//geriye deger döndürür, 2 param alıyor
int topla(int sayi1,int sayi2)
{
    int sonuc;
    sonuc=sayi1+sayi2;
    return sonuc;
}
```

```
int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_4MA,
       GPIO_PIN_TYPE_STD_WPU);
    // pin 4, pull-up olarak ayarlandı.
    while(1)
    {
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==16)// sw1, pf.0=0
        {
            void portfnin2pininilojik1yap();
            //fonksiyon çağırıldı.
        }
        else if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==1) //sw2 pf.4=0
        {
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 8);
            int k=topla(12, 13); // topla fonk
            // geri deger döner ve k'ya atanır.
        }
    }
}
```

Derste yapılan bazı örnekler ektedir:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h"
#include "math.h" // bunu unutma

//Basit bir fonksiyon, parametresi yok
void portfnin2pininilojik1yap()
{
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
}

//daha mantıklı bir fonksiyon
void portfnin_x_nolu_pini_lojik_1_yap(int pinno)
{
    GPIOPinWrite(GPIO_PORTF_BASE, pow(2,pinno),pow(2,pinno));
}

int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
```

```
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_4MA,
        GPIO_PIN_TYPE_STD_WPU);
    // pin 4, pull-up olarak ayarlandı.
    while(1)
    {
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==16)// sw1, pf.0=0
        {
            portfnin_x_nolu_pini_lojik_1_yap(1); // red
        }
        else if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==1) //sw2 pf.4=0
        {
            portfnin_x_nolu_pini_lojik_1_yap(2); // blue
        }
    }
}
```

Bir üstteki örnekleri tek fonksiyona daha mantıklı olacak şekilde yapalım:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h"
#include "math.h"

// fonksiyonu burada tanımlamak istemiyorsan prototipini
// buraya yaz
//fonksiyonu main fonskiyonundan sonra aşağıda yaz
void PORTXPINY_Lojik_1_Yap(double port,int pinno); // prototip

int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_4MA,
        GPIO_PIN_TYPE_STD_WPU);
    // pin 4, pull-up olarak ayarlandı.
    while(1)
    {
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==16)// sw1, pf.0=0
        {
            PORTXPINY_Lojik_1_Yap(GPIO_PORTF_BASE, 2);// BLUE
        }
        else if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==1) //sw2 pf.4=0
        {
            PORTXPINY_Lojik_1_Yap(GPIO_PORTF_BASE, 3);// GREEN
        }
    }
}

void PORTXPINY_Lojik_1_Yap(double port,int pinno)
{
    GPIOPinWrite(port, pow(2,pinno),pow(2,pinno));
}
```

### HWREG ile Bit Adresleme

**Bit Adresleme**

İlgilendiğimiz pinlerin ram adresi:

$$PortunBaseAdresi + 4 * 2^b$$

b: ilgilenilen bit veya bitler

| Port | Base address |
|---|---|
| PortA | 0x40004000 |
| PortB | 0x40005000 |
| PortC | 0x40006000 |
| PortD | 0x40007000 |
| PortE | 0x40024000 |
| PortF | 0x40025000 |

Table 6.4. Base Addresses for bit-specific addressing of ports A-F

Şekil 2.12: Portların base adresi

| *If we wish to access bit* | *Constant* |
|---|---|
| 7 | 0x0200 |
| 6 | 0x0100 |
| 5 | 0x0080 |
| 4 | 0x0040 |
| 3 | 0x0020 |
| 2 | 0x0010 |
| 1 | 0x0008 |
| 0 | 0x0004 |

**Table 6.3. Address offsets used to specify individual data port bits.**

Şekil 2.13: Verilen formüldeki $4 * 2^b$ ile elde edilen sabitler

**Bit Adresleme**

PORTA'nın 1, 2, 3 pinleriyle ilgilenelim. 0x40004000+0x0008+0x0010 +0x0020=0x4000.4038 0x4000.4038 adresi okunursa pin 1, 2, 3 okunur. Yada bu adrese bir data yazılırsa 1, 2, 3 pinlerine yazılır.

Eğer PortA'nın 5. pini ile ilgileniyorsak oluşan sabit ram adresi : 0x0080 =» 0x4000.4000 == 0x40004080

```
#define PA5    (*((volatile unsigned long *)0x40004080)) // yada
PA5 EQU 0x40004080 // pa5 etiketini c'de kendinize göre isimlendirebilirsiniz.

PA5 = 0x20;          // make PA5 high

PA5 = 0x00;          // make PA5 low

PA5 = PA5^0x20;      // toggle PA5
```

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_gpio.h"
#include "math.h"

// fonksiyonu burada tanımlamak istemiyorsan prototipini
// buraya yaz
//fonksiyonu main fonskiyonundan sonra aşağıda yaz
void PORTXPINY_Lojik_1_Yap(int port,int pinno); // prototip

int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
    GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_4MA,
       GPIO_PIN_TYPE_STD_WPU);
    // pin 4, pull-up olarak ayarlandı.
    while(1)
    {
        if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==16)// sw1, pf.0=0
        {
            PORTXPINY_Lojik_1_Yap(GPIO_PORTF_BASE, 2);// BLUE
        }
        else if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4)==1) //sw2 pf.4=0
        {
            PORTXPINY_Lojik_1_Yap(GPIO_PORTF_BASE, 3);// GREEN
        }
    }
}

void PORTXPINY_Lojik_1_Yap(int port,int pinno)
{
    //GPIOPinWrite(port, pow(2,pinno),pow(2,pinno));
    int a=4*pow(2,pinno);
    HWREG(port+a)=pow(2,pinno);
    // ilgilenilenport+ 4*2^pin=> ilgilendiğimiz pinin ram adresi
}
```

**********

Butona bir kere basıp çekince çalışan kod örneği;

```c
#include <inc/hw_types.h>
#include <inc/hw_memmap.h>
#include <driverlib/gpio.h>
#include <driverlib/sysctl.h>
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"

void main(void)
{
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
           SYSCTL_OSC_MAIN);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY_DD;
```

```
        HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
        HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
        GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,GPIO_PIN_4|GPIO_PIN_0);
        GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4|GPIO_PIN_0, GPIO_STRENGTH_4MA,
            GPIO_PIN_TYPE_STD_WPU);
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00);

        while(1)
        {
                if (GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_4)==0)
                {
                        while(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_4)==0); //
                            butondan kaldırmayı bekler
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                            GPIO_PIN_3, 2);
                        SysCtlDelay(2000);
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                            GPIO_PIN_3, 0);

                }
                if (GPIOPinRead(GPIO_PORTF_BASE,GPIO_PIN_0)==0)
                {
                        while(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_0)==0)
                        {
                                // butondan kaldırana kod burada dönüp durur
                        }
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                            GPIO_PIN_3, 4);
                        SysCtlDelay(2000);
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                            GPIO_PIN_3, 0);

                }
        }

}
```

**********

### c dilinde define kullanımı

```
 #include "inc/hw_gpio.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/sysctl.h"

// # define komutu ile pine isimlendirme verebiliriz.
#define LED_RED GPIO_PIN_1
#define LED_BLUE GPIO_PIN_2
#define LED_GREEN GPIO_PIN_3

#define BUTTON_1 GPIO_PIN_0
#define BUTTON_2 GPIO_PIN_4

void main(void) {
        int light;
        SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
            SYSCTL_OSC_MAIN);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, LED_RED|LED_BLUE|LED_GREEN);
        GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, BUTTON_1|BUTTON_2);
```

```
        GPIOPadConfigSet(GPIO_PORTF_BASE, BUTTON_1|BUTTON_2, GPIO_STRENGTH_2MA,
            GPIO_PIN_TYPE_STD_WPU);

        for (;;) { // while(1) ile aynı görevi görür.
                        light = 0;

                        if(!GPIOPinRead(GPIO_PORTF_BASE, BUTTON_1))
                                        light = LED_RED;

                        if(!GPIOPinRead(GPIO_PORTF_BASE, BUTTON_2))
                                        light = LED_BLUE;

                        GPIOPinWrite(GPIO_PORTF_BASE, LED_RED|LED_BLUE|LED_GREEN,
                            light);

                        SysCtlDelay(500000);
        }
}
```

### Döngü örnekleri

Aşağıdaki örnekte çalıştırmak istediğiniz while döngüsünü while(1) yapınız.

```
#include <inc/hw_types.h>
#include <inc/hw_memmap.h>
#include <driverlib/gpio.h>
#include <driverlib/sysctl.h>

void main(void)
{
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
            SYSCTL_OSC_MAIN);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
        GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4, GPIO_STRENGTH_4MA,
            GPIO_PIN_TYPE_STD_WPU);
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0x00);

        long k=2;
        while(0)
        {

                while (k<9)
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                            GPIO_PIN_3, k);
                        SysCtlDelay(1000000);
                        k=k*2;
                }
                k=8;
                while (k>1)
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                            GPIO_PIN_3, k);
                        SysCtlDelay(1000000);
                        k=k/2;
                }
                k=2;
        }

        while(0)
        {

                while (k<9)
```

```
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                           GPIO_PIN_3, k);
                        SysCtlDelay(1000000);
                        k=k<<1;
                }
                k=8;
                while (k>1)
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                           GPIO_PIN_3, k);
                        SysCtlDelay(1000000);
                        k=k>>1;
                }
                k=2;
        }

        while(0)
        {
                int i;
                k=2;
                for (i=1;i<4;i++)
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                           GPIO_PIN_3, k);
                        SysCtlDelay(1000000);
                        k=k<<1;
                }
                k=8;
                while (k>1)
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                           GPIO_PIN_3, k);
                        SysCtlDelay(1000000);
                        k=k>>1;
                }
                k=2;
        }
        while(1)
        {
                int i;
                for (i=2;i<9;i=i*2)
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                           GPIO_PIN_3, i);
                        SysCtlDelay(1000000);
                }
                for (i=8;i>1;i=i/2)
                {
                        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|
                           GPIO_PIN_3, i);
                        SysCtlDelay(1000000);
                }
        }
}
```

Şimdiye kadar yapılan örneklerde hazır fonksiyonları kullandık. Bu fonksiyonlara **api** denmektedir. Aslıda geçen seneden de bildiğiniz gibi bu fonksiyonlar ramdeki ayar kaydedicilerindeki bazı ayarları değiştirmek için yapılmaktadır. Örnek olarak geçen sene ADCCON kaydedicisi bulunmaktaydı burada ise bu işlemleri **hazır** fonksiyonlar ile ilgili ADC kaydedicilerinde yapmaktadır. Yukarıda bahsedilen GpioPinTypeGpioPinOutput gibi bir fonksiyonun yaptığı ayar Gpio registerlarında bazı değişikler yapmaktadır. Eğer biz kodlarımızın en üstünde verilen header dosyalarını kullanmadan sadece **"inc/tm4c123gh6pm.h"** kütüphanesini kullanarak da tüm işlemlerimizi gerekleştirebiliriz.

```c
#include "stdint.h"
#include "inc/tm4c123gh6pm.h"

// global değişkenlerim

unsigned long In;
unsigned long Out;

void PortF_AYARLARIni_Yap(void); // fonksiyon prototipi


int main(void)
{

  PortF_AYARLARIni_Yap();

  while(1)
  {
    In = GPIO_PORTF_DATA_R&0x10;    // Sw1 oku, maskeleme ile okundu
    In = In>>2;                     // gelen değeri 2 sağa ötele ?? neden düşün
    Out = GPIO_PORTF_DATA_R;
    Out = Out&0xFB;
    Out = Out|In;
    GPIO_PORTF_DATA_R = Out;         // çıkış
  }
}

void PortF_AYARLARIni_Yap(void){

    unsigned long delay;
    SYSCTL_RCGC2_R |= 0x00000020;      // 1) PORTF Saati aktifleştir.
    GPIO_PORTF_LOCK_R = 0x4C4F434B;    // 2) GPIO Port F kilidini aç.
    GPIO_PORTF_CR_R = 0x1F;            // portf'in 5 pininde değişik yapma hakkı ver.
    GPIO_PORTF_AMSEL_R = 0x00;         // 3) Analog işlem fonksiyonunu iptal et
    GPIO_PORTF_PCTL_R = 0x00000000;    // 4) PCTL GPIO on PF4-0 alternatif
       fonksiyonları kapat
    GPIO_PORTF_DIR_R = 0x0E;           // 5) PF4,PF0 girişler, PF3-1 çıkış pinleri
    GPIO_PORTF_AFSEL_R = 0x00;         // 6) PF4-0 alternatif fonksiyonları kapat
    GPIO_PORTF_PUR_R = 0x11;           // enable pull-up on PF0 and PF4
    GPIO_PORTF_DEN_R = 0x1F;           // 7) enable digital I/O on PF4-0 işlem hakkı
       ver.
}
```

### 2.1.5 Kesme Fonksiyonları

**18.2 API Functions**

Functions

- void IntDisable (uint32_t ui32Interrupt)
- void IntEnable (uint32_t ui32Interrupt)
- uint32_t IntIsEnabled (uint32_t ui32Interrupt)
- bool IntMasterDisable (void)
- bool IntMasterEnable (void)
- void IntPendClear (uint32_t ui32Interrupt)
- void IntPendSet (uint32_t ui32Interrupt)
- int32_t IntPriorityGet (uint32_t ui32Interrupt)
- uint32_t IntPriorityGroupingGet (void)
- void IntPriorityGroupingSet (uint32_t ui32Bits)
- uint32_t IntPriorityMaskGet (void)
- void IntPriorityMaskSet (uint32_t ui32PriorityMask)
- void IntPrioritySet (uint32_t ui32Interrupt, uint8_t ui8Priority)
- void IntRegister (uint32_t ui32Interrupt, void (*pfnHandler)(void))
- void IntTrigger (uint32_t ui32Interrupt)
- void IntUnregister (uint32_t ui32Interrupt)

Şekil 2.14: Kesme Fonksiyonları

GPIO kesmesi ve diğer kesmelerde ana 3 fonksiyon kullanılmaktadır. Diğer ayrıntılar için farklı fonksiyonlarda bulunmaktadır. Bakınız Şekil 2.14

**IntMasterEnable();** fonksiyonu işlemcinin kesme fonksiyonlarını aktif hale getirir.

***IntEnable(Çevre Birimi handler'i);*** fonksiyonu ilgilendiğimiz çevre biriminin kesmesini aktif hale getirmektedir.

**void IntRegister(uint32_t ui32Interrupt, void (*pfnHandler)(void));** fonksiyonu kesme oluştuğunda hangi alt fonksiyonun çalışacağını belirlemektedir.

```
//
// UART 0 kesmesinde çalışacak fonksiyon
//
void UART0KesmesiFonksiyonu(void)
{
//
// kesme fonksiyonu
//
}
//
// Set the UART 0 interrupt handler.
//
IntRegister(INT_UART0, UART0KesmesiFonksiyonu);
```

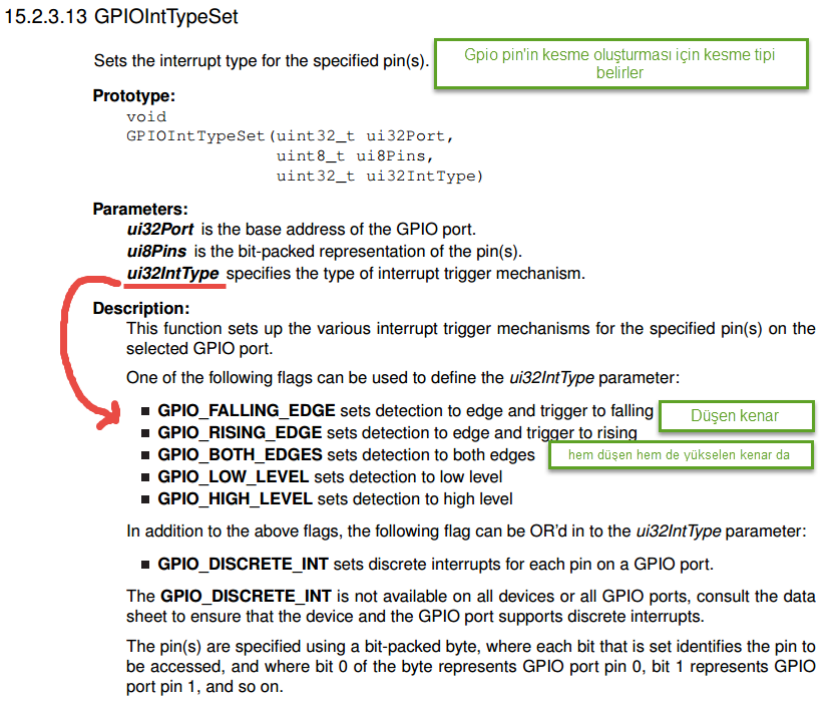
### 2.1.6 Gpio Kesme Fonksiyonları

void GPIOIntTypeSet(uint32_t ui32Port, uint8_t ui8Pins, uint32_t ui32IntType) : Pinin kesme oluşturması için kesme tipini belirler.

void GPIOIntEnable(uint32_t ui32Port, uint32_t ui32IntFlags) : pin'in kesmesini aktif eder.

void GPIOIntRegister(uint32_t ui32Port, void (*pfnIntHandler)(void)) : İlgili portun pinlerinden kesme gelirse gidilecek alt fonksiyonu belirler.

void GPIOIntClear(uint32_t ui32Port, uint32_t ui32IntFlags) : kesme oluşturan pinin kesmesini temizler.



15.2.3.13 GPIOIntTypeSet

Sets the interrupt type for the specified pin(s).

Gpio pin'in kesme oluşturması için kesme tipi belirler

**Prototype:**

```
void
GPIOIntTypeSet(uint32_t ui32Port,
               uint8_t ui8Pins,
               uint32_t ui32IntType)
```

**Parameters:**

***ui32Port*** is the base address of the GPIO port.
***ui8Pins*** is the bit-packed representation of the pin(s).
***ui32IntType*** specifies the type of interrupt trigger mechanism.

**Description:**

This function sets up the various interrupt trigger mechanisms for the specified pin(s) on the selected GPIO port.

One of the following flags can be used to define the *ui32IntType* parameter:

- **GPIO_FALLING_EDGE** sets detection to edge and trigger to falling
- **GPIO_RISING_EDGE** sets detection to edge and trigger to rising
- **GPIO_BOTH_EDGES** sets detection to both edges
- **GPIO_LOW_LEVEL** sets detection to low level
- **GPIO_HIGH_LEVEL** sets detection to high level

Düşen kenar

hem düşen hem de yükselen kenar da

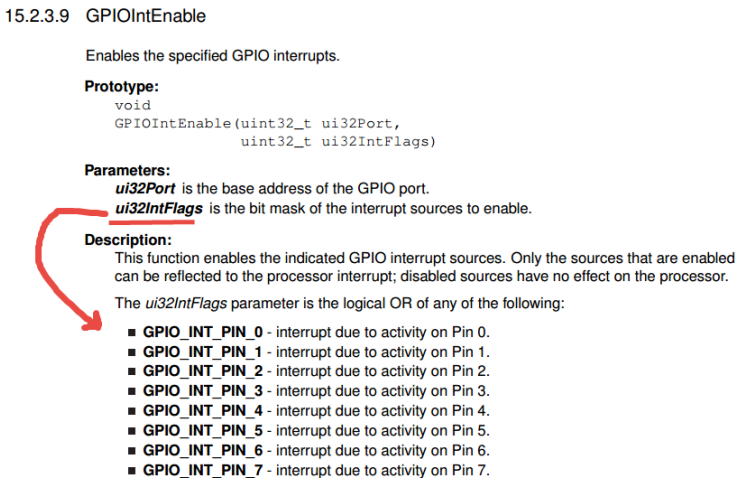
In addition to the above flags, the following flag can be OR'd in to the *ui32IntType* parameter:

- **GPIO_DISCRETE_INT** sets discrete interrupts for each pin on a GPIO port.

The **GPIO_DISCRETE_INT** is not available on all devices or all GPIO ports, consult the data sheet to ensure that the device and the GPIO port supports discrete interrupts.

The pin(s) are specified using a bit-packed byte, where each bit that is set identifies the pin to be accessed, and where bit 0 of the byte represents GPIO port pin 0, bit 1 represents GPIO port pin 1, and so on.

Şekil 2.15: Bir Pin'in kesme oluşturması amacıyla kesme tipi belirleme fonksiyonu



15.2.3.9 GPIOIntEnable

Enables the specified GPIO interrupts.

**Prototype:**

```
void
GPIOIntEnable(uint32_t ui32Port,
              uint32_t ui32IntFlags)
```

**Parameters:**

***ui32Port*** is the base address of the GPIO port.
***ui32IntFlags*** is the bit mask of the interrupt sources to enable.

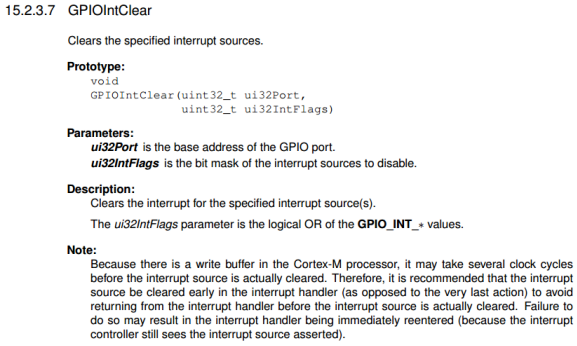
**Description:**

This function enables the indicated GPIO interrupt sources. Only the sources that are enabled can be reflected to the processor interrupt; disabled sources have no effect on the processor.

The *ui32IntFlags* parameter is the logical OR of any of the following:

- **GPIO_INT_PIN_0** - interrupt due to activity on Pin 0.
- **GPIO_INT_PIN_1** - interrupt due to activity on Pin 1.
- **GPIO_INT_PIN_2** - interrupt due to activity on Pin 2.
- **GPIO_INT_PIN_3** - interrupt due to activity on Pin 3.
- **GPIO_INT_PIN_4** - interrupt due to activity on Pin 4.
- **GPIO_INT_PIN_5** - interrupt due to activity on Pin 5.
- **GPIO_INT_PIN_6** - interrupt due to activity on Pin 6.
- **GPIO_INT_PIN_7** - interrupt due to activity on Pin 7.

Şekil 2.16: GPIOIntEnable Fonksiyonu açıklamaları



15.2.3.7 GPIOIntClear

Clears the specified interrupt sources.

**Prototype:**

```
void
GPIOIntClear(uint32_t ui32Port,
             uint32_t ui32IntFlags)
```

**Parameters:**

***ui32Port*** is the base address of the GPIO port.
***ui32IntFlags*** is the bit mask of the interrupt sources to disable.

**Description:**

Clears the interrupt for the specified interrupt source(s).

The *ui32IntFlags* parameter is the logical OR of the **GPIO_INT_*** values.

**Note:**

Because there is a write buffer in the Cortex-M processor, it may take several clock cycles before the interrupt source is actually cleared. Therefore, it is recommended that the interrupt source be cleared early in the interrupt handler (as opposed to the very last action) to avoid returning from the interrupt handler before the interrupt source is actually cleared. Failure to do so may result in the interrupt handler being immediately reentered (because the interrupt controller still sees the interrupt source asserted).

Şekil 2.17: GPIOIntClear Fonksiyonu açıklamaları

### kesme örneği

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "inc/hw_gpio.h"

void kesme(void)
{
    GPIOPinIntClear(GPIO_PORTF_BASE, GPIO_PIN_4);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2);
    SysCtlDelay(2000000);
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
}

void main(void)
{
        IntMasterEnable(); // 1
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
           SYSCTL_OSC_MAIN);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        //HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY_DD;
        //HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
        //HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
        GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_4);
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
```

```
        GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_STRENGTH_4MA,
            GPIO_PIN_TYPE_STD_WPU);
        GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_4, GPIO_FALLING_EDGE); //3
        GPIOPinIntClear(GPIO_PORTF_BASE, GPIO_PIN_4);
        GPIOPinIntEnable(GPIO_PORTF_BASE, GPIO_PIN_4); //4
        GPIOIntRegister(GPIO_PORTF_BASE,kesme); //5
        IntEnable(INT_GPIOF); // 2
        while(1)
        {
                // ana döngü
        }
}
```

Derste yapılan örnek:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_ints.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "inc/hw_gpio.h"

void kesmefonksiyonu()
{
    int a=GPIOIntStatus(GPIO_PORTF_BASE, true);
    // GPIOIntStatus ile hangi kaynaktan (pinden) kesme geldiğini belirliyoruz

    if (a==1)
    {
        GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_0); // bayragı temizle
        GPIOPinWrite(GPIO_PORTF_BASE, 2, 2);
        SysCtlDelay(20000);
        GPIOPinWrite(GPIO_PORTF_BASE, 2, 0);
        SysCtlDelay(20000);
    }
    else if(a==16)
    {
        GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_4); // bayragı temizle
        GPIOPinWrite(GPIO_PORTF_BASE, 4, 4);
        SysCtlDelay(20000);
        GPIOPinWrite(GPIO_PORTF_BASE, 4, 0);
        SysCtlDelay(20000);
    }
}

int main(void)
{
    IntMasterEnable(); //1.ayar
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    HWREG ( GPIO_PORTF_BASE + GPIO_O_LOCK ) = GPIO_LOCK_KEY ;
    HWREG ( GPIO_PORTF_BASE + GPIO_O_CR ) |= 0x01 ;
    HWREG ( GPIO_PORTF_BASE + GPIO_O_LOCK ) = 0;

    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, 17);  //GPIO_PIN_0|GPIO_PIN_4;

    GPIOPadConfigSet(GPIO_PORTF_BASE, 17, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
    // 2, 3, ve 4. ayarlarımız

    GPIOIntTypeSet(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_INT_PIN_4, GPIO_FALLING_EDGE);
    // gpio_pin_4
```

```
    GPIOIntEnable(GPIO_PORTF_BASE, GPIO_PIN_0|GPIO_PIN_4);
    GPIOIntRegister(GPIO_PORTF_BASE, kesmefonksiyonu);

    //5 ayar
    IntEnable(INT_GPIOF);

    while(1)
    {

    }
}
```

# Bölüm 3

# General Purpose Timer Module(G.P.T.M)-Genel Amaçlı Zamanlayıcı Modülü

Genel Amaçlı Zamanlayıcı Modülü veya GPTM, 6, 16 veya 32 bit ve 6, 32 veya 64 bit genel amaçlı zamanlayıcılara sahiptir. Yakalama, karşılaştırma veya darbe sayımı için kullanılabilirler. Mevcut beş farklı zaman modu vardır. Tek sayım, periyodik, giriş kenarı sayımı veya zaman yakalama, PWM üretimi ve gerçek zamanlı saat.

Zamanlayıcılar, yukarı veya aşağı sayabilirsiniz. Modüller, çoklu zamanlayıcılar arasında senkronizasyonu, zincirleme zincirlemeyi ve hata ayıklama sırasında kullanıcı tarafından etkinleştirmeyi destekler. Bu son özellik, kullanıcı durma düğmesine bastığında ve istenmeyen zamanlayıcı kesintilerinin meydana gelmesini engellediğinde hata ayıklama sırasında zamanlayıcıları durur. Zamanlayıcılar ayrıca ADC'nin çevrime başlatmasını veya DMA transferlerini de tetikleyebilir.

## 3.1 Timer

Bazı zamanlayıcılar, 16-bit (half-width) yarı genişlikte zamanlayıcılar ve 32-bit (full width) tam genişlikli zamanlayıcı sağlarken, diğerleri 32-bit yarı genişlikli zamanlayıcılar ve 64-bitlik tam genişlikli bir zamanlayıcı sağlar.

API'nın amaçları için, bir zamanlayıcı modülü tarafından sağlanan iki yarım genişlikli zamanlayıcılar TimerA ve TimerB olarak adlandırılır ve tam genişlikli zamanlayıcı TimerA olarak adlandırılır.

Tam genişlikli veya yarım genişlikli bir zamanlayıcı olarak yapılandırıldığında, bir çevrim zamanlayıcı veya sürekli zamanlayıcı olarak çalışacak şekilde bir zamanlayıcı ayarlanabilir.

Tek çevrim modda yapılandırılmışsa, zamanlayıcı geri sayarken ya da sayma sırasında üst değere ulaştığında saymayı durdurur.

Sürekli modda yapılandırılırsa, zamanlayıcı sıfıra (geri sayma) veya son değerine (saymaya) kadar sayar, ardından yeniden yükler ve saymaya devam eder.

Tam genişlikli bir zamanlayıcı olarak yapılandırıldığında, zamanlayıcı ayrıca bir RTC olarak çalışacak şekilde de yapılandırılabilir.

Yarı genişlik modundayken, zamanlayıcı olay yakalama veya Darbe Genişliği Modülasyonu (PWM) olarak da yapılandırılabilir.

Olay yakalama için yapılandırıldığında, zamanlayıcı sayaç olarak işlev görür. Olaylar veya olayların kendileri arasındaki zamanı saymak için yapılandırılabilir. Sayılan olayın türü, yükselen kenar, düşen kenar veya her iki kenar olarak yapılandırılabilir.

Timer ayarlarını ve uygulamalarını yazmak için driverlib/timer.c ile driverlib/timer.h kütüphaneleri kullanılır.

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"

void timerkesmefonksiyonu(void);

int main(void)
{
    uint32_t ui32Period;

    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    // timer çevre birimi aktif

    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    // sürekli çevrim modu, aşağıya doğru sayısı

    ui32Period = (SysCtlClockGet() / 10) / 2;
    // başlangıç değeri

    TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
    // timer ilk değeri yükleniyor.
    // 32 bitlik sayıcı

    IntEnable(INT_TIMER0A);
    // 1. kesme ayarı
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
    // 3. kesme ayarı

    IntMasterEnable();
    // 2. kesme ayarı

    TimerEnable(TIMER0_BASE, TIMER_A);
    // timer çalıştı

    TimerIntRegister(TIMER0_BASE, TIMER_A, timerkesmefonksiyonu);
    // Timer kesme fonksiyonu ayarı
    while(1)
    {
    }
}

void timerkesmefonksiyonu(void)
{
    // kesmeyi temizle
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
```

```
        }
}
```

*Time match fonksiyonu kullanımı örneği*

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "driverlib/timer.h"

void kesmefonk(void);

int main(void)
{

    SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
    TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC); // aşağı yönde sayıcı

    TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet());
    TimerMatchSet(TIMER0_BASE, TIMER_A, 38000000); // bu değere gelince kesme
       üretecek
    IntEnable(INT_TIMER0A);
    TimerIntEnable(TIMER0_BASE, TIMER_TIMA_MATCH); // match olunca kesme oluşsun
       dedik
    TimerIntRegister(TIMER0_BASE, TIMER_A, kesmefonk);
    IntMasterEnable();
    TimerEnable(TIMER0_BASE, TIMER_A);
    while(1)
    {
    }
}

void kesmefonk(void)
{
    TimerIntClear(TIMER0_BASE, TIMER_TIMA_MATCH); // bayrağı temizle unutma...

    TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet());
    // kesme oluştu tekrar ilk degeri yuklememiz gerekiyor

    if(GPIOPinRead(GPIO_PORTF_BASE, GPIO_PIN_2))
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
    }
    else
    {
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_2, 4);
    }
}
```

## 29.2 API Functions

### Functions

- uint32_t TimerADCEventGet (uint32_t ui32Base)
- void TimerADCEventSet (uint32_t ui32Base, uint32_t ui32ADCEvent)
- uint32_t TimerClockSourceGet (uint32_t ui32Base)
- void TimerClockSourceSet (uint32_t ui32Base, uint32_t ui32Source)
- void TimerConfigure (uint32_t ui32Base, uint32_t ui32Config)
- void TimerControlEvent (uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Event)
- void TimerControlLevel (uint32_t ui32Base, uint32_t ui32Timer, bool bInvert)
- void TimerControlStall (uint32_t ui32Base, uint32_t ui32Timer, bool bStall)
- void TimerControlTrigger (uint32_t ui32Base, uint32_t ui32Timer, bool bEnable)
- void TimerControlWaitOnTrigger (uint32_t ui32Base, uint32_t ui32Timer, bool bWait)
- void TimerDisable (uint32_t ui32Base, uint32_t ui32Timer)
- uint32_t TimerDMAEventGet (uint32_t ui32Base)
- void TimerDMAEventSet (uint32_t ui32Base, uint32_t ui32DMAEvent)
- void TimerEnable (uint32_t ui32Base, uint32_t ui32Timer)
- void TimerIntClear (uint32_t ui32Base, uint32_t ui32IntFlags)
- void TimerIntDisable (uint32_t ui32Base, uint32_t ui32IntFlags)
- void TimerIntEnable (uint32_t ui32Base, uint32_t ui32IntFlags)
- void TimerIntRegister (uint32_t ui32Base, uint32_t ui32Timer, void (*pfnHandler)(void))
- uint32_t TimerIntStatus (uint32_t ui32Base, bool bMasked)
- void TimerIntUnregister (uint32_t ui32Base, uint32_t ui32Timer)
- uint32_t TimerLoadGet (uint32_t ui32Base, uint32_t ui32Timer)
- uint64_t TimerLoadGet64 (uint32_t ui32Base)
- void TimerLoadSet (uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)
- void TimerLoadSet64 (uint32_t ui32Base, uint64_t ui64Value)
- uint32_t TimerMatchGet (uint32_t ui32Base, uint32_t ui32Timer)
- uint64_t TimerMatchGet64 (uint32_t ui32Base)
- void TimerMatchSet (uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)
- void TimerMatchSet64 (uint32_t ui32Base, uint64_t ui64Value)
- uint32_t TimerPrescaleGet (uint32_t ui32Base, uint32_t ui32Timer)
- uint32_t TimerPrescaleMatchGet (uint32_t ui32Base, uint32_t ui32Timer)
- void TimerPrescaleMatchSet (uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)
- void TimerPrescaleSet (uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Value)
- void TimerRTCDisable (uint32_t ui32Base)
- void TimerRTCEnable (uint32_t ui32Base)
- void TimerSynchronize (uint32_t ui32Base, uint32_t ui32Timers)
- void TimerUpdateMode (uint32_t ui32Base, uint32_t ui32Timer, uint32_t ui32Config)
- uint32_t TimerValueGet (uint32_t ui32Base, uint32_t ui32Timer)
- uint64_t TimerValueGet64 (uint32_t ui32Base)



Şekil 3.1: Timer API Fonksiyonları

# Bölüm 4

# ADC

## 4.1 ADC

Tiva mikrodenetleyicisinde eşit özelliklere sahip 2 ADC çevre birimi bulunmaktadır. ADC çevre birimleri 12 bit hassasiyetinde olup, 12 kanalı + dahili sıcaklık sensörünü okuyabilmektedir. Ayrıca her biri 4 adet sırayılıcı (sequencer) özelliği vardır.

**Figure 13-1. Implementation of Two ADC Blocks**

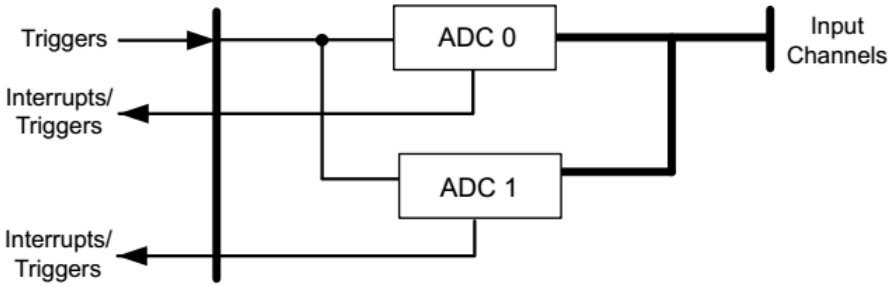



Figure 13-2 on page 801 provides details on the internal configuration of the ADC controls and data registers.

Şekil 4.1: ADC Blok Diyagramı

Şekil 4.1'de ADC Blok diyagramı verilmiştir. ADC'ler yazılım ile tetiklenir. ADC'nin çevrimi tamamladığını kesme bayrağından anlaşılabilir. Ayrıca ADC çevrimi bitince başka bir olayı da tetikleyebilir.

ADC'nin diğer teknik özellikleri:

- Reference voltages:
  - VREFP tied to 3.3V on the Launchpad.
  - VREFN tied to ground on the Launchpad.
- Range and Resolution
  - Range = 0->4095
  - Resolution = 3.3V ÷ 4096 = 8.05 mV (p. 803)
- Max Sampling Speed = 1 million samples÷second
- Hardware Averaging: Averages 4 samples in hardware during conversion

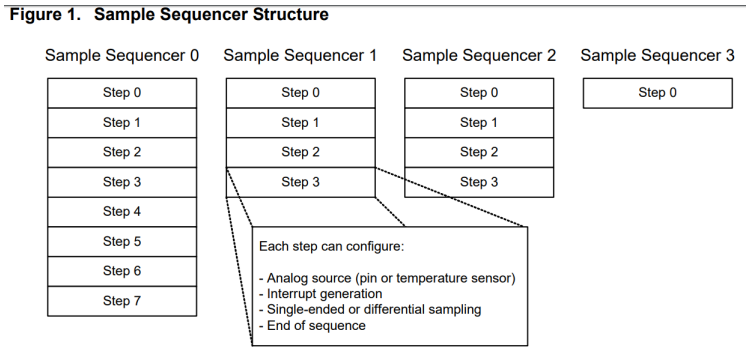
Tiva adc çevre birimini rahat kullanabilmek için bir sıralayıcı (sequencer) yapısı oluşturulmuştur. Bir sequencer'ı ayarlayabilmek için aşağıdaki 4 adım uygulanmalıdır:

- Input source for each sample

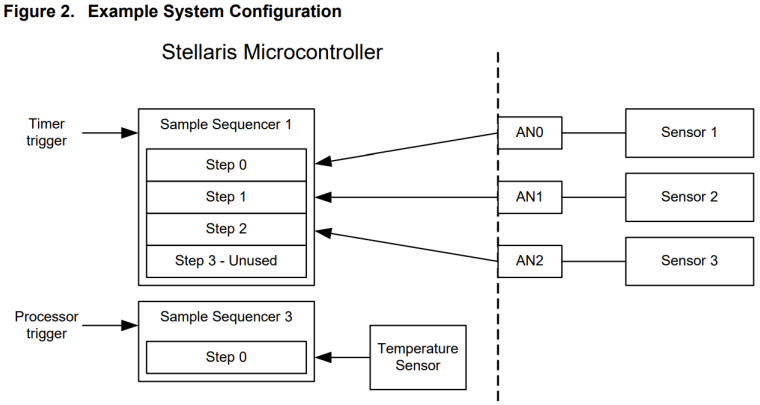
| Sequencer | Number of Samples | Depth of FIFO |
|---|---|---|
| SS3 | 1 | 1 |
| SS2 | 4 | 4 |
| SS1 | 4 | 4 |
| SS0 | 8 | 8 |

Şekil 4.2: ADC Sıralayıcı Yapısı

- Mode (single-ended, or differential) for each sample
- Interrupt generation on sample completion for each sample
- Indicator for the last sample in the sequence




Şekil 4.3: ADC Sıralayıcı ve step Yapısı




Şekil 4.4: Örnek

ADC kullanılırken çevrim sonucunda bir değer oluşturabilmek için referans değere ihtiyaç vardır. Bu referans değer mikroişlemcinin içinden dahili olarak VREFP ve VREFN'den alınır. VREFN değeri ADC'den okunan değerin sayısal karşılığı olan 0x000 değerini, VREFP değeri ADC'den okunan değerin sayısal karşılığı olan 0xFFF (4095) değerini ifade eder. Bu değerler arasındaki farkın ADC'den okunan her değere karşılık gelen gerilim değerini hesaplamak için ise aşağıdaki eşitlik kullanılır.

mV per ADC code= (VREFP - VREFN) ÷ 4096

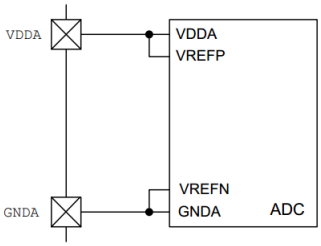
Örnek olarak VREFP=3.3 V, VREFN=0 V olursa;

mV per ADC code= (3.3 - 0) ÷ 4096= 8.05 mV ÷ ADC code olur. Ve buradan ADC den okunan değerin sayısal karşılındaki her değişimin gerilim olarak 8.05 mV luk değişime karşılık geldiği bulunur.

| Pin Name | Pin Number | Pin Assignment |
|---|---|---|
| AIN0 | 6 | PE3 |
| AIN1 | 7 | PE2 |
| AIN2 | 8 | PE1 |
| AIN3 | 9 | PE0 |
| AIN4 | 64 | PD3 |
| AIN5 | 63 | PD2 |
| AIN6 | 62 | PD1 |
| AIN7 | 61 | PD0 |
| AIN8 | 60 | PE5 |
| AIN9 | 59 | PE4 |
| AIN10 | 58 | PB4 |
| AIN11 | 57 | PB5 |

Şekil 4.5: Analog kanallar

**Figure 13-8. ADC Voltage Reference**




The range of this conversion value is from 0x000 to 0xFFF. In single-ended-input mode, the 0x000 value corresponds to the voltage level on VREFN; the 0xFFF value corresponds to the voltage level on VREFP. This configuration results in a resolution that can be calculated using the following equation:

```
mV per ADC code = (VREFP - VREFN) / 4096
```

Şekil 4.6: ADC Referans Gerilimi

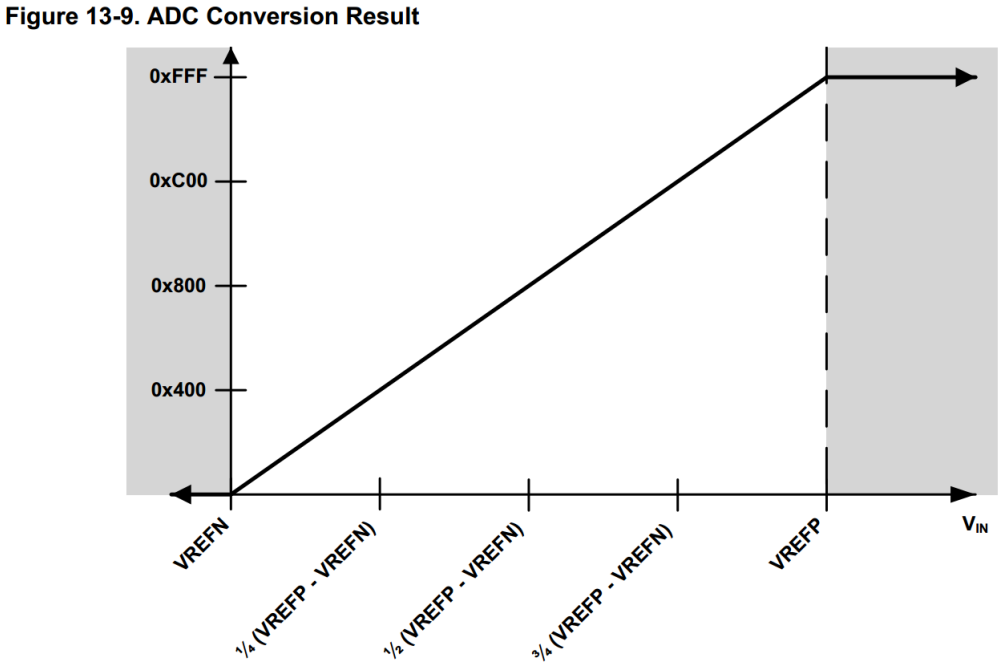
Dahili sıcaklık sensörü: 1 adet dahili sıcaklık sensörü vardır. Dahili sıcaklık sensörü VREFP beslemesi 3.3 V, VREFN beslemesi ise 0V a bağlıdır. Bu dahili sensör mikroişlemcinin sıcaklık değerini gerilime çevirir ve bu çevrim sensörün sıcaklık-gerilim grafiğinden elde edilen denklem ile sağlanır. Şekil 4.8'de grafik dahili sıcaklık sensörünün sıcaklık-gerilim grafiğidir.

Şekil 4.8'deki grafiğin denklemi çıkarıldığında; VTSENS= 2.7 - ((TEMP + 55) ÷ 75) olarak bulunur.

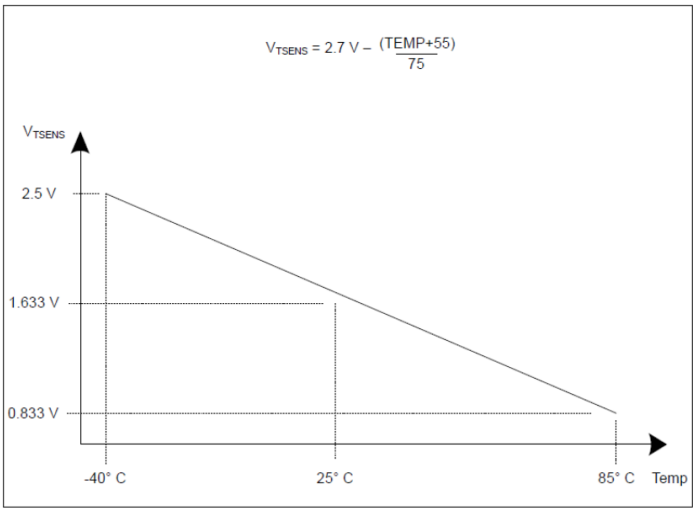
Eşitlik düzenlenip sıcaklık değişkeni yalnız bırakıldığında;

$TEMP = 147.5 - (75 * VTSENS)$

$TEMP = 147.5 - ((75 * (VREFP - VREFN) * ADCCODE) \div 4096)$ elde edilir.




Şekil 4.7: ADC Çevrim Sonuçları




Şekil 4.8: ADC dahili sıcaklık sensörü çevrimi

## 4.2 ADC Kod Açıklamaları

Evet şu ana kadar hep Tiva geliştiricilerinin bizim için hazırladığı hazır kodları (api denmektedir) kullanarak işlemleri gerçekleştirdik. Yavaş yavaş direkt ram adresleri ile bu işlemleri de açıklamaya başlayacağız.

Tabiki ilk önce **(1.)** bir çevre birimi önce aktif edilmelidir.

Biz çevre birimlerini SysCtlPeripheralEnable(çevre biriminin adı); fonksiyonu ile hayata geçiriyorduk. ADC çevre birimi de SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC); ile aktifleştirilebilir. Ancak bunu api fonksiyonu kullanmadan gerçekleştirmek istersek,

```
//
// Enable the clock to the ADC module
//
// System Control RCGC0 register
//
SYSCTL_RCGC0_R |= SYSCTL_RCGC0_ADC;
```

**(2.)** olarak ADC'nin çevrim hızı ayarlanması gerekir. Eğer biz bunu api fonksiyonları ile gerçekleştirirsek;

```
//
// Configure the ADC to sample at 500KSps
```

```
    //
    SysCtlADCSpeedSet(SYSCTL_SET0_ADCSPEED_500KSPS)
```

Apisiz olarak bunu ayarlamak isteseydik;

```
    //
    // Configure the ADC to sample at 500KSps
    //
    // System Control RCGC0 register
    //
    SYSCTL_RCGC0_R |= SYSCTL_RCGC0_ADCSPD500K;
```

**(3.)** olarak hangi sequence kullanılacaksa ayarları gerçekleştirilir. ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_TIMER, 1); ile ADC nin 1. sequence ayarlandı ve bu dizi yazılım tetiklemeli olarak çevrime başlayacaktır.

```
Prototip:
void ADCSequenceConfigure(uint32_t ui32Base, uint32_t ui32SequenceNum,uint32_t
   ui32Trigger,uint32_t ui32Priority)

Parameters:

ui32Base is the base address of the ADC module.

ui32SequenceNum is the sample sequence number.

ui32Trigger is the trigger source that initiates the sample sequence; must be one of
   the ADC_TRIGGER_* values.

ui32Priority is the relative priority of the sample sequence with respect to the
   other sample sequences.

ADC_TRIGGER_PROCESSOR
ADC_TRIGGER_COMP0
ADC_TRIGGER_COMP
ADC_TRIGGER_COMP2
ADC_TRIGGER_EXTERNAL
ADC_TRIGGER_TIMER
ADC_TRIGGER_PWM0
ADC_TRIGGER_PWM1
ADC_TRIGGER_PWM2
ADC_TRIGGER_PWM3
ADC_TRIGGER_ALWAYS - A trigger that is always asserted, causing the sample sequence
   to capture repeatedly
```

Örnek olarak aşağıda farklı iki ayar verilmiştir.

```
    //
    // Configure sample sequence 1: timer trigger, priority = 1
    //
    ADCSequenceConfigure(ADC_BASE, 1, ADC_TRIGGER_TIMER, 1);
    //
    // Configure sample sequence 3: processor trigger, priority = 0
    //
    ADCSequenceConfigure(ADC_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
```

**(4.)** olarak ayarladığımız sequence'a ait step ayarlarını gerçekleştirmemiz gerekmektedir. Bu fonksiyon ADCSequenceStepConfigure'dur

```
    Prototype:
    void ADCSequenceStepConfigure(uint32_t ui32Base, uint32_t ui32SequenceNum,
       uint32_t ui32Step, uint32_t ui32Config); şeklindedir.

    Parameters:
```

```
    ui32Base is the base address of the ADC module.

    ui32SequenceNum is the sample sequence number.

    ui32Step is the step to be configured.

    ui32Config is the configuration of this step; must be a logical OR of ADC_CTL_TS,
        ADC_CTL_IE, ADC_CTL_END, ADC_CTL_D, one of the input channel selects (
       ADC_CTL_CH0 through ADC_CTL_CH23), and one of the digital comparator selects(
       ADC_CTL_CMP0 through ADC_CTL_CMP7).
```

Örnek olarak aşağıda farklı iki ayar verilmiştir.

```
// Configure sample sequence 1 control
//
// Configure sample sequence 3 steps 0, 1 and 2
//
// - Step 0: Single-ended, No temp sensor, No interrupt
// - Step 1: Single-ended, No temp sensor, No interrupt
// - Step 2: Single-ended, No temp sensor, Interrupt, End of sequence
//
ADCSequenceStepConfigure(ADC_BASE, 1, 0, ADC_CTL_CH0);
ADCSequenceStepConfigure(ADC_BASE, 1, 1, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 2, ADC_CTL_CH2 | ADC_CTL_IE | ADC_CTL_END)


// Configure sample sequence 3 control
//
// - Step 1: Single-ended, temp sensor, no interrupt, end of sequence
// Configure sample sequence 3 step 0
//
ADCSequenceStepConfigure(ADC_BASE, 3, 0, ADC_CTL_TS | ADC_CTL_END);
```

**(5.)** olarak çevrimi bitmiş bir adc'den veriler ADCSequenceDataGet fonksiyonu ile elde edilir.

```
Gets the captured data for a sample sequence.
Prototype:
int32_t ADCSequenceDataGet(uint32_t ui32Base, uint32_t ui32SequenceNum, uint32_t
    *pui32Buffer);

Parameters:

ui32Base is the base address of the ADC module.

ui32SequenceNum is the sample sequence number.

pui32Buffer is the address where the data is stored.
```

Örnek kod aşağıda verilmiştir.

```
//
// Retrieve data from sample sequence 1 FIFO
//
ADCSequenceDataGet(ADC_BASE, 1, &ulSeq1DataBuffer);
//
// ulSeq1DataBuffer tanımlanmış bir dizi değişkenidir.
```

## 4.3 Dersteki örnekler

ilk örneğimiz dahili sıcaklık sensörünün okunması, kesme fonksiyonu kullanılmyacak

### 4.3.1 Örnek 1: kesmesiz

```c
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/sysctl.h"
#include "driverlib/adc.h" // ADC için unutma

int main(void)
{
        uint32_t ui32ADC0Value[4]; // 4 elemanlı dizimiz
        volatile uint32_t ui32TempAvg;
        volatile uint32_t ui32TempValueC;
        volatile uint32_t ui32TempValueF;

        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|
           SYSCTL_XTAL_16MHZ);

        // çevre birimi aktif
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

        // sequence 1 -> 4 kanallı -> yazılım ile çevrim başlayacak
        ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);

        // sequnce ait step ayarları
        // tümünde sadece dahili sıcaklık sensörü okunuyor.
        ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_TS);
        ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_TS);
        ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_TS);
        ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_TS|ADC_CTL_IE|ADC_CTL_END);

        // sequnce 1 aktif edildi.
        ADCSequenceEnable(ADC0_BASE, 1);

        // yoklama ile adc'nin çevriminin bitip bitmediği
        // kontrol edilecek.

        while(1)
        {
           // bayrağı temizle
           ADCIntClear(ADC0_BASE, 1);

           // çevrime başla
           ADCProcessorTrigger(ADC0_BASE, 1);

           // çevrim bitmedi ise bekle
           while(!ADCIntStatus(ADC0_BASE, 1, false))
           {
           }

           // verileri al
           ADCSequenceDataGet(ADC0_BASE, 1, ui32ADC0Value);
           ui32TempAvg = (ui32ADC0Value[0] + ui32ADC0Value[1] + ui32ADC0Value[2] +
              ui32ADC0Value[3] + 2)/4;
           ui32TempValueC = (1475 - ((2475 * ui32TempAvg)) / 4096)/10;
           ui32TempValueF = ((ui32TempValueC * 9) + 160) / 5;
        }
}
```