BlackLib - Kullanım Kılavuzu

Beaglebone Black C++ Kütüphanesi

Yazar: Yiğit Yüce

31.01.2014

v1.0

İÇİNDEKİLER

1. ÖZET

Bu dosya BlackLib Kütüphanesinin dokümantasyonu olarak hazırlanmıştır. Bu dokümanın hedef kitlesi, son kullanıcıdır. Dokümana konu olan BlackLib Kütüphanesi, Beaglebone Black için, C++ programlama dili ile yazılmıştır.

2. ANAHTAR KELİMELER

Beaglebone Black, BlackLib, C++, kütüphane, PWM, ADC, GPIO

3. GİRİŞ

BlackLib kütüphanesi C++ programlama dili ile Beaglebone Black için yazılmıştır. Bu kütüphane sayesinde Beaglebone Black'e ait ADC, GPIO ve PWM özellikleri kullanılabilmektedir. Kernel v3.7 ve sonrasını destekleyen Beaglebone Black cihazlarında çalışabilmektedir. Kütüphane LGPL lisansı ile lisanslanmıştır.

4. LİSANS

For any comment or suggestion please contact the creator of BlackLib Library at ygtyce@gmail.com

5. BİLİNMESİ GEREKENLER

1. Device Tree

Device Tree, donanımları tanımlamak için kullanılan bir veri yapısıdır. Herhangi bir cihazın her detayını işletim sisteminin içinde kodlamak yerine, cihazın özellikleri bir veri yapısı içinde tutulabilir ve işletim sisteminin yüklenme anında işletim sistemine gönderilebilir.

Veri yapılar her türden bilgiyi tutabilir. Ancak cihazları tanımlayacak olan ve Device Tree 'de bulunan veri yapıları işletim sisteminin anlayabileceği şekilde hazırlanmış olmalıdır. "Bindings", Device Tree 'de bulunacak cihazların özelliklerinin tanımlarıdır. [1]

2. C++11

C++11(önceden bilinen adıyla C++0x), en temel haldeki standart C++ programlama dilinin en son versiyonudur. [2]

C++0x standardı ile yazılmış fonksiyonlar, derleyiciye C++0x bayrağı eklenmeden kullanılamaz. Bu sebeple kodunuzu derleme aşamasında bu bayrağı eklemeniz gerekmektedir. "Compiling C++11 with g++" konulu yazı bu linkte açıklanmıştır [3] veya eğer geliştirme yapmak için Eclipse kullanıyorsanız, "Eclipse CDT C++11/C++0x support" konulu yazı bu linkte açıklanmıştır. [4] Bu kütüphaneyi kullanan kodunuzu derleyebilmeniz için bu adımları yapmanız gerekmektedir.

6. KULLANIM

1. Kütüphaneyi Dâhil Etme

Kütüphaneyi kullanmak için şunları yapmanız gerekmektedir;

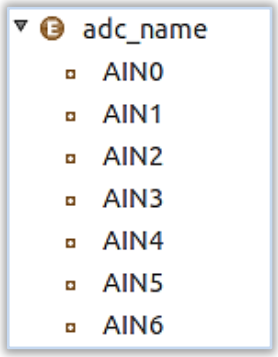
→ Yazdığınız kodun bulunduğu dizine, BlackLib dizinini kopyalayın.
→ Kodunuzun başına aşağıdaki satırı yazın.

```
#include "BlackLib/BlackLib.h"
```

2. Özel Tanımlamalar

Beaglebone Black üzerindeki özelliklerden kullanılmak istenen seçildikten sonra, bu özelliğe ait alt seçenekler, yeri geldiği zaman seçilmelidir. Bizim uygulamamızda bu seçenekler, özel tanımlamalar ile seçilebilmektedir. Her bir özellik için kullanılabilecek seçenekler özel olarak tanımlanmıştır ve her seçenek için bu isimler farklılık göstermektedir. Bu isimler enumeration özelliği ile tanımlanmıştır.

a. ADC Tanımlamaları



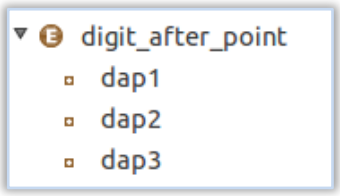
adc_name isimli ADC tanımlaması, Beaglebone Black'in analog girişlerinden veri okunmak istendiğinde kullanılabilecek alt seçenektir. Bu alt seçenek sayesinde hangi "AINx" girişi üzerinden analog veri okunacağı seçilebilir.

Beaglebone Black üzerinde 8 adet ADC bulunmasına rağmen bir tanesi Beaglebone Black için ayrılmış olup, dışarıdan bir bileşen takılmasına olanak sağlanmamıştır. Dolayısıyla Beaglebone Black ile yedi adet analog giriş okunabilir.

Adc isim seçimi, kodunuzda BlackADC sınıfından nesne üretilirken sizden beklenen ilk ve tek parametredir. Bu parametre "*adc_name*" cinsinden bir değişken olabilmektedir. adc_name seçeneğinin kullanımı aşağıdaki gibidir.

```
BlackADC *test = new BlackADC(AIN0);
```

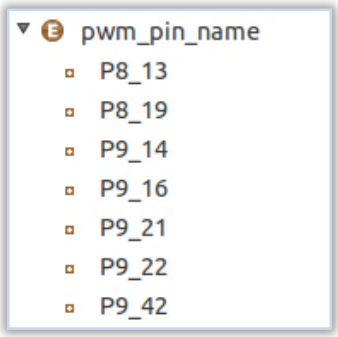
```
BlackADC test(AIN0);
```



ADC okunurken, okunan değer mV (mili volt) cinsinden okunur. Dolayısıyla çoğu zaman bu değeri ölçeklemek gerekli olacağından kütüphane içerisinde "*getParsedValue()*" adında bir fonksiyon yazılmıştır. Bu fonksiyon bir tane parametre alır ve bu "*digit_after_point*" tipinde bir değişken olabilir. Bu değişken ile çözümleme seviye seçimi yapılmakta ve volta çevrilmiş veride noktadan sonra kaç basamak görüntülenmesi istendiği belirtilmektedir. Aşağıda verilen örnek, bu fonksiyon ve "*digit_after_point*" değişkeninin kullanımı ile ilgilidir.

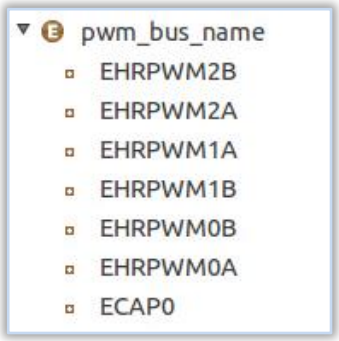
```
float deger = test->getParsedValue(dap2);
```

b. PWM Tanımlamaları



"pwm_pin_name" veya "*pwm_bus_name* " , Beaglebone Black'in pwm çıkışlarından, PWM üretilmek istendiğinde kullanılabilecek alt seçeneklerdir. Bu alt seçenekler sayesinde hangi PWM çıkışı üzerinden, PWM sinyali oluşturulacağı seçilmiş olunur.

Beaglebone Black üzerinde birden çok PWM çıkışı bulunmaktadır. Bunlardan kullanılabilir olanlar yanda verilmiştir. Bu PWM çıkışları P8 ve P9 numaralı portlarda bulunmaktadır.



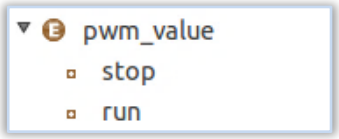
PWM çıkışları sadece pin isimleri ile değil ayrıca kendilerine ait sembolik isimlerle de kullanabilirsiniz. Bu sembolik isimler yanda belirtilmiştir. Yandaki listelerin sırası ikisi için de aynıdır. Yani "*P8_13*" ile "*EHRPWM2B*" , "*P9_42*" ile "*ECAP0*" aynı PWM çıkışına denk gelmektedir.

"*EHRPWM*", Gelişmiş Yüksek Çözünürlüklü Darbe Genişlik Modülasyonu (Enhanced High Resolution Pulse Width Modulation) , "*ECAP0*", Gelişmiş Yakalama Modu Darbe Genişlik Modülasyonu (Enhanced Capture Pulse Width Modulation) anlamına gelmektedir.

Pin adı veya bus adı seçimi, kodunuzda BlackPWM sınıfından nesne üretilirken sizden beklenen ilk ve tek parametredir. Bu parametre "*pwm_pin_name* " veya "*pwm_bus_name* " cinsinden bir değişken olabilmektedir. Bu seçeneklerin kullanımı aşağıdaki gibidir.

```
BlackPWM *test = new BlackPWM(P9_14);
veya
BlackPWM *test = new BlackPWM(EHRPWM1A);
```

```
BlackPWM test(P9_14);
veya
BlackPWM test(EHRPWM1A);
```



Pwm sinyalinin PWM çıkışlarından alınabilmesi için sinyalinin üretilmeye başlatılması gerekir. Pwm sinyalinin üretilmesini başlatmak ya da durdurmak için kütüphane içerisinde "*setSignalState()*" adında bir fonksiyon yazılmıştır. Bu fonksiyon bir tane parametre alır ve bu "*pwm_value*" tipinde bir değişken olabilir. Bu değişkenin değeri "*stop*" veya "*run*" olabilir. "*run*" sinyali üretmeyi başlatır, "stop" ise durdurur. Aşağıda verilen örnek bu fonksiyon ve "*pwm_value*" değişkeninin kullanımı ile ilgilidir.

```
test->setSignalState(stop);
```

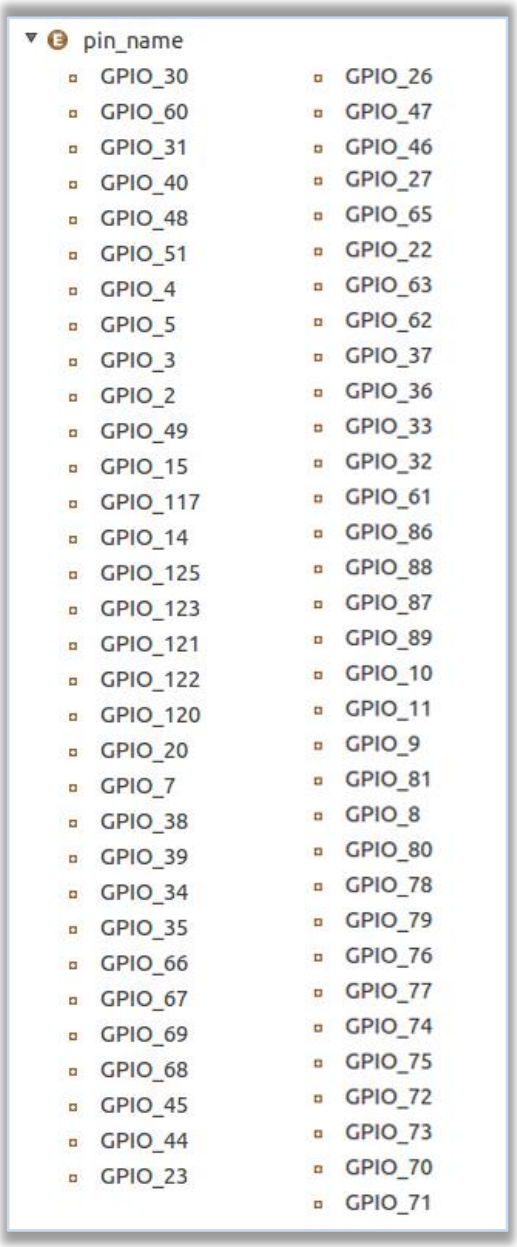
c. GPIO Tanımlamaları

"pin_name", GPIO pinlerinden, hangisinin kullanılacağını belirtmek için kullanılan alt seçenektir. Bu alt seçenek sayesinde Beaglebone Black üzerindeki hangi GPIO pininin kullanılacağını seçebilirsiniz.

Beaglebone Black'in iki adet GPIO portu vardır ve bunların her biri 46 adet pine sahiptir. Bu portlar P8 ve P9'dur. Ancak pinlerin tamamı son kullanıcı tarafından doğrudan kullanılamaz. Bu pinlerden bazıları Beaglebone Black'in HDMI çıkışı, SD kart, LCD ekran gibi kendi barındırdığı özellikleri için kullanılır. Dolayısıyla projenize başlamadan önce hangi pinlerin ve özelliklerin kullanılacağına karar vermelisiniz. Hatta bazen, bazı pinlerin kullanılabilmesi için Device Tree 'ye yükleme yapılması gerekebilir.

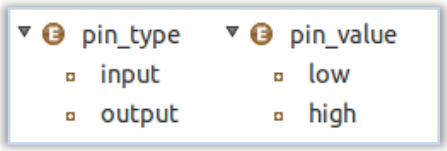
Genel amaçlı pinler iki tipte kullanılabilir. Bunlar giriş ya da çıkış olabilir. Giriş pinleri sayısal bir veri okumak için, çıkış pinleri ise sayısal bir değer üretmek için kullanılabilir. Sayısal veriler mantıksal 1 veya 0 olabilir. Pin tipi seçimi, *"pin_type"* seçeneği ile yapılır.

Kodunuzda BlackGPIO sınıfından nesne üretilirken sizden iki adet parametre beklenir. Bunlar pin numarası ve tipine ait seçimlerdir. Bu parametreler "`pin_name`" ve "`pin_type`" tipi değişkenlerdir. Bu seçeneklerin kullanımı aşağıdaki gibidir.




```
BlackGPIO *test = new BlackGPIO(GPIO_30, input);
                        or
BlackGPIO *test = new BlackGPIO(GPIO_60, output);
```

```
BlackGPIO test(GPIO_30, input);
                or
BlackGPIO test(GPIO_60, output);
```



GPIO pininin değerini ayarlayabilmek için pinin çıkış tipinde olması gerekmektedir. Bir pinin yönünün ayarlanma işlemi, BlackGPIO sınıfından nesne üretilirken yapılır. Eğer bu aşamada pin giriş tipi olarak ayarlanırsa, bu pinin değeri kullanıcı tarafından değiştirilemez. Pin değeri ayarlamak için kütüphane içerisinde setValue() fonksiyonu yazılmıştır. Bu fonksiyon "*pin_value*" tipinde bir değişken almaktadır. Bu değişkenin değeri yalnızca "*low*" veya "*high*" olabilir. "*low*", pini mantıksal 0 durumuna, "*high*" ise mantıksal 1 durumuna getirir. Aşağıda verilen örnek bu fonksiyon ve "*pin_value*" değişkeninin kullanımı ile ilgilidir.

```
test->setValue(high);
```

3. Son Kullanıcı İle Etkileşime Geçen Fonksiyonlar

a. ADC Yönetim Fonksiyonları

i. BlackADC::getValue() : string

Bu fonksiyon analog giriş değerini verir. BlackADC::ainPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya, analog giriş gerilimini, mili volt mertebesinde tutar. String tipi, analog giriş değerini döndürür. Eğer dosya açma işleminde hata oluşursa, hata mesajı döndürür.

```
cout << "AIN degeri = " << test->getValue() << " mV";
```

ii. BlackADC::getNumericValue() : int

Bu fonksiyon analog giriş değerini verir. BlackADC::ainPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya, analog giriş gerilimini, mili volt mertebesinde tutar. Integer tipi, analog giriş değerini döndürür. Eğer dosya açma işleminde hata oluşursa, -1 döndürür.

```
int x = test->getNumericValue() + 100;
cout << "AIN degeri + 0,1V = " << x << " mV";
```

iii. BlackADC::getParsedValue(digit_after_point) : float

Bu fonksiyon dönüştürülmüş analog giriş değerini verir. getNumericValue() fonksiyonunu kullanarak değeri okur ve fonksiyona gönderilen parametreye göre bu değeri çözümler. Çözümleme işlemi sonucunda okunan değer volt seviyesine gelir. ***digit_after_point*** tipi değişken alır. Bu değişken çözümleme seviyesini gösterir. Float tipi, çözümlenmiş analog giriş değerini döndürür.

```
float x = test->getParsedValue(dap3);
cout << "AIN(a,bcd tipinde)= " << x << " Volt";
```

iv. BlackADC::fail() : bool

Bu fonksiyon genel hata ayıklama işlemi için kullanılır. Eğer herhangi bir hata meydana geldiyse doğru(true), diğer durumlarda yanlış(false) döndürür.

```
if ( test->fail() )
{
    cout << "Hata: Genel bir hata olustu.";
}
```

v. BlackADC::fail(BlackADC::flags) : bool

Bu fonksiyon özel hata ayıklama işlemi için kullanılır. Kodunuzda BlackADC üye fonksiyonlarını kullandıktan sonra bunu kullanabilirsiniz. ***flags*** tipi değişken alır ve bu değişken istediğiniz bir hatanın durumunu öğrenmek için kullanılır. Seçilen hatanın durumunu döndürür (doğru ya da yanlış).

```
if ( test->fail(ReadErr) )
{
    cout << "Hata: Analog giris, dosya okuma hatasi.";
}
```

b. PWM Yönetim Fonksiyonları

i. BlackPWM::getValue() : string

Bu fonksiyon doluluk boşluk oranının yüzdelik değerini verir. PWM 'in periyot ve duty değerini bulmak için getNumericPeriodValue() ve getNumericDutyValue() fonksiyonunu çağırır. Bunu yaptıktan sonra bu değerleri kullanarak yüzdelik değeri hesaplar. String tipi doluluk boşluk oranı yüzdesini döndürür.

```
cout << "PWM yuzde degeri = " << test->getValue();
```

ii. BlackPWM::getPeriodValue () : string

Bu fonksiyon pwm sinyalinin periyot değerini verir. BlackPWM::periodPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya pwm sinyalinin periyot değerini nanosaniye(ns) mertebesinde tutar. String tipi nanosaniye seviyesindeki periyot değerini döndürür. Eğer dosya açma işleminde hata oluşursa, hata mesajı döndürür.

```
cout << "PWM periyot degeri = " << test->getPeriodValue();
```

iii. BlackPWM::getDutyValue (): string

Bu fonksiyon pwm sinyalinin duty değerini verir. BlackPWM::dutyPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya pwm sinyalinin duty değerini nanosaniye(ns) mertebesinde tutar. String tipi nanosaniye seviyesindeki duty değerini döndürür. Eğer dosya açma işleminde hata oluşursa, hata mesajı döndürür.

```
cout << "PWM duty degeri = " << test->getDutyValue();
```

iv. BlackPWM::getRunValue (): string

Bu fonksiyon pwm sinyalinin çalışma değerini verir. BlackPWM::runPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya pwm sinyalinin çalışma değerini tutar. Bu değer sadece 0 ya da 1 olabilir. String tipi çalışma değerini döndürür. Eğer dosya açma işleminde hata oluşursa, hata mesajı döndürür.

```
cout << "PWM calisma degeri = " << test->getRunValue();
```

v. BlackPWM::getPolarityValue (): string

Bu fonksiyon pwm sinyalinin polarite değerini verir. BlackPWM::polarityPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya pwm sinyalinin polarite değerini tutar. Bu değer sadece 0 ya da 1 olabilir. String tipi polarite değerini döndürür. Eğer dosya açma işleminde hata oluşursa, hata mesajı döndürür.

```
cout << "PWM polarite degeri = " << test->getPolarityValue();
```

vi. BlackPWM::getNumericValue (): float

Bu fonksiyon doluluk boşluk oranının yüzdelik değerini sayısal olarak verir. Pwm sinyalinin periyot ve duty değerini bulmak için getNumericPeriodValue() ve getNumericDutyValue() fonksiyonlarını çağırır. Bunu yaptıktan sonra bu değerleri yüzdelik değere çevirir. Float tipi doluluk boşluk oranı yüzdesini döndürür.

```
float x = test->getNumericValue();
cout << "PWM yüzdelik değeri = " << x;
```

vii. BlackPWM::getNumericPeriodValue () : uint32_t

Bu fonksiyon pwm sinyalinin periyot değerini sayısal olarak verir. Pwm sinyalinin periyot değerini bulmak için getPeriodValue() fonksiyonunu çağırır. Bunun ardından bulunan bu string tipi periyot değeri, strtoimax() fonksiyonu kullanılarak uint32_t tipine çevrilir. uint32_t tipinde, nanosaniye seviyesinde periyot değeri döndürür.

```
uint32_t x = test->getNumericPeriodValue();
cout << "PWM periyot değeri(ns) = " << x << endl;
cout << "PWM periyot değeri(us) = " << x/1000 << endl;
cout << "PWM periyot değeri(ms) = " << x/1000000;
```

viii. BlackPWM::getNumericDutyValue () : uint32_t

Bu fonksiyon pwm sinyalinin duty değerini sayısal olarak verir. Pwm sinyalinin duty değerini bulmak için getDutyValue() fonksiyonunu çağırır. Bunun ardından bulunan bu string tipi duty değeri, strtoimax() fonksiyonu kullanılarak uint32_t tipine çevrilir. uint32_t tipinde, nanosaniye seviyesinde duty değeri döndürür.

```
uint32_t x = test->getNumericDutyValue();
cout << "PWM duty değeri(ns) = " << x << endl;
cout << "PWM duty değeri(us) = " << x/1000 << endl;
cout << "PWM duty değeri(ms) = " << x/1000000;
```

ix. BlackPWM::setDutyPercent (float) : bool

Bu fonksiyon pwm sinyalinin yüzdelik değerini ayarlar. Eğer gönderilen parametre sınırlar dâhilinde (0 ile 100 arası) ise bu fonksiyon periyot değerini değiştirmeden duty değerini ayarlar. Yeni duty değerini hesaplamak için, şu anki periyot değeri ile girilen yüzdelik çarpılır ve bu çarpım periyot değerinden çıkarılır. Ardından hesaplanan bu değer, duty dosyasına kaydedilir. ***Float*** tipinde bir değişken alır ve bu değişken yeni yüzdelik değerini ayarlamak için kullanılır. Ayarlama işlemleri başarılı olursa doğru(true), diğer durumlarda yanlış(false) döndürür.

```
float x = 41,0;
test->setDutyPercent(x);
```

x. BlackPWM::setPeriodTime (uint32_t) : bool

Bu fonksiyon pwm sinyalinin periyot değerini ayarlar. Eğer gönderilen parametre sınırlar dâhilinde (0 veya o anki duty değeri ile 10^9 arası) ise bu fonksiyon, gönderilen değeri, periyot dosyasına kaydederek periyot değerini ayarlar. ***uint32_t*** tipinde bir değişken alır ve bu değişken yeni periyot değerini ayarlamak için kullanılır. Ayarlama işlemleri başarılı olursa doğru(true), diğer durumlarda yanlış(false) döndürür.

```
uint32_t x = 400000000;
test->setPeriodTime(x);
```

xi. BlackPWM::setSpaceRatioTime (uint32_t) : bool

Bu fonksiyon pwm sinyalinin boşluk süresinin değerini ayarlar. Eğer gönderilen parametre sınırlar dâhilinde (0 ile o anki periyot değeri arası) ise bu fonksiyon duty dosyasına bu değeri yazarak yeni duty değerini ayarlar. ***uint32_t*** tipinde bir değişken alır ve bu değişken yeni boşluk süresi değerini ayarlamak için kullanılır. Ayarlama işlemleri başarılı olursa doğru(true), diğer durumlarda yanlış(false) döndürür.

```
uint32_t x = 400000000;
test->setSpaceRatioTime(x);
```

xii. BlackPWM::setLoadRatioTime (uint32_t) : bool

Bu fonksiyon pwm sinyalinin çalışma süresinin değerini ayarlar. Eğer gönderilen parametre sınırlar dâhilinde (0 ile o anki periyot değeri arası) ise bu fonksiyon duty değerini ayarlar. Yeni duty değerini hesaplamak için, o anki periyot değerinden, girilen çalışma süresi değeri çıkarılır. Ardından hesaplanan yeni boşluk değeri, duty dosyasına kaydedilir. ***uint32_t*** tipinde bir değişken alır ve bu değişken yeni çalışma süresi değerini ayarlamak için kullanılır. Ayarlama işlemleri başarılı olursa doğru(true), diğer durumlarda yanlış(false) döndürür.

```
uint32_t x = 400000000;
test->setLoadRatioTime(x);
```

xiii. BlackPWM::setPolarity (pwm_polarity) : bool

Bu fonksiyon pwm sinyalinin polarite tipini ayarlar. Gönderilen parametreye göre polarite değerini 1 ya da 0 yapar. Bu polarite değeri, polarite dosyasına kaydedilir. ***pwm_polarity*** tipinde değişken alır ve bu değişken polarite tipini ayarlamak için kullanılır. Ayarlama işlemleri başarılı olursa doğru(true), diğer durumlarda yanlış(false) döndürür.

```
test->setPolarity(reverse);
```

xiv. BlackPWM::setRunState (pwm_value) : bool

Bu fonksiyon pwm sinyalinin çalışma durumunu ayarlar. Gönderilen parametreye göre çalışma değerini 1 ya da 0 yapar. Bu çalışma değeri, çalışma dosyasına kaydedilir. ***pwm_value*** tipinde değişken alır ve bu değişken çalışma durumunu ayarlamak için kullanılır. Ayarlama işlemleri başarılı olursa doğru(true), diğer durumlarda yanlış(false) döndürür.

```
test->setRunState(stop);
```

xv. BlackPWM::toggleRunState () : void

Bu fonksiyon pwm sinyalinin çalışma durumunu değiştirir. Var olan değeri göz önünde bulundurarak, çalışma değerini 1 ya da 0 yapar. Bu yeni çalışma değeri, çalışma dosyasına kaydedilir.

```
test-> toggleRunState();
```

xvi. BlackPWM::tooglePolarity () : void

Bu fonksiyon pwm sinyalinin polaritesini değiştirir. Var olan değeri göz önünde bulundurarak, polarite değerini 1 ya da 0 yapar. Bu yeni polarite değeri, polarite dosyasına kaydedilir.

```
test-> tooglePolarity();
```

xvii. BlackPWM::isRunning () : bool

Bu fonksiyon pwm sinyalinin çalışma durumunu kontrol eder. getRunValue() fonksiyonunu çağırır ve bu fonksiyonun geri dönüş değerini değerlendirir. Çalışma değeri 0 ise yanlış(false), değil ise doğru(true) döndürür.

```
if ( test->isRunning() )
{
    cout << "Pwm sinyali uretiliyor.";
}
```

xviii. BlackPWM::isPolarityStraight () : bool

Bu fonksiyon pwm sinyalinin polaritesini kontrol eder. getPolarityValue() fonksiyonunu çağırır ve geri dönüş değerini değerlendirir. Polarite değeri 0 ise doğru(true), değil ise yanlış(false) döndürür.

```
if ( test->isPolarityStraight() )
{
    cout << "Pwm sinyali duz polaritededir.";
}
```

xix. BlackPWM::isPolarityReverse () : bool

Bu fonksiyon pwm sinyalinin polaritesini kontrol eder. getPolarityValue() fonksiyonunu çağırır ve geri dönüş değerini değerlendirir. Polarite değeri 0 ise yanlış(false), değil ise doğru(true) döndürür.

```
if ( test->isPolarityReverse() )
{
    cout << "Pwm sinyali ters polaritededir.";
}
```

xx. BlackPWM::fail () : bool

Bu fonksiyon genel hata ayıklama işlemi için kullanılır. Eğer herhangi bir hata meydana geldiyse doğru(true), diğer durumlarda yanlış(false) döndürür.

```
if ( test->fail() )
{
    cout << "Hata: Genel bir hata olustu.";
}
```

xxi. BlackPWM::fail (BlackPWM::flags) : bool

Bu fonksiyon özel hata ayıklama işlemi için kullanılır. Kodunuzda BlackPWM üye fonksiyonlarını kullandıktan sonra kullanabilirsiniz. ***flags*** tipi değişken alır ve bu değişken istediğiniz bir hatanın durumunu öğrenmek için kullanılır. Seçilen hatanın durumunu döndürür (doğru ya da yanlış).

```
if ( test->fail(outOfRangeErr) )
{
    cout << "Hata: Pwm, girilen deger aralik disinda.";
}
```

c. GPIO Yönetim Fonksiyonları

i. BlackGPIO::getValue() : string

Bu fonksiyon GPIO pin değerini verir. isReady() fonksiyonunu çağırarak pin durumunu kontrol eder. Pin hazır durumdaysa, BlackGPIO::valuePath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya, gpio pin değerini tutar. String tipi, GPIO pin değerini döndürür. Eğer dosya açma işleminde hata oluşursa veya pin hazır durumda değilse, hata mesajı döndürür.

```
cout << "GPIO pin degeri = " << test->getValue();
```

ii. BlackGPIO::setValue (gpio_value) : bool

Bu fonksiyon GPIO pin değerini ayarlar. Eğer pin çıkış tipi ise ve pin hazır durumdaysa, gönderilen parametreye göre pin değerini 1 ya da 0 yapar. Bu değer, değer dosyasına yazılır. ***gpio_value*** tipinde bir değişken alır ve bu değişken pin değerini ayarlamak için kullanılır. Ayarlama işlemleri başarılı olursa doğru(true), diğer durumlarda yanlış(false) döndürür.

```
test->setValue(low);
```

iii. BlackGPIO::isHigh () : bool

Bu fonksiyon GPIO pininin değerini kontrol eder. getValue() fonksiyonunu çağırır ve bu fonksiyonun geri dönüş değerini değerlendirir. Pinin değeri 0 ise yanlış(false), değil ise doğru(true) döndürür.

```
if ( test->isHigh() )
{
    cout << "GPIO 1 konumundadir.";
}
```

iv. BlackGPIO::toggleValue () : void

Bu fonksiyon GPIO pin değerini değiştirir. Eğer pin çıkış tipi ise, var olan değeri göz önünde bulundurarak, gpio pin değerini 1 ya da 0 yapar. Bu yeni pin değeri, değer dosyasına kaydedilir.

```
test-> toggleValue();
```

v. BlackGPIO::fail () : bool

Bu fonksiyon genel hata ayıklama işlemi için kullanılır. Eğer herhangi bir hata meydana geldiyse doğru(true), diğer durumlarda yanlış(false) döndürür.

```
if ( test->fail() )
{
    cout << "Hata: Genel bir hata olustu.";
}
```

vi. BlackGPIO::fail (BlackGPIO::flags) : bool

Bu fonksiyon özel hata ayıklama için kullanılır. Kodunuzda BlackGPIO üye fonksiyonlarını kullandıktan sonra kullanabilirsiniz. ***flags*** tipi değişken alır ve bu değişken istediğiniz bir hatanın durumunu öğrenmek için kullanılır. Seçilen hatanın durumunu döndürür. (doğru ya da yanlış)

```
if ( test->fail(exportErr) )
{
    cout << "Hata: Pin export edilmemis.";
}
```

4. Hata Ayıklama ve Hata Bayrakları

Hata bayrakları üye fonksiyonlar çağırıldıktan sonra kontrol edilir. Bayrakların amacı problem takibidir. Eğer fonksiyonlar çalışırken bir sorun meydana gelirse, bayraklar bu problemlerin kaydını tutar. Hata tipleri fonksiyonun yaptığı işe göre farklılık gösterir. Dolayısıyla hangi fonksiyondan sonra hangi bayrağın kontrol edilmesi gerektiği bilinmelidir. Örneğin, eğer yazma işlemi yapan bir fonksiyon çağırıldıktan sonra okuma fonksiyonu sorunlarını tutan bayrağı kontrol ederseniz, bu kontrol sonucu hatalı olacaktır.

errorCore veya errorCore{ADC, PWM, GPIO} veri yapılarına ait olan bayrakları kullanmak son kullanıcı için gerekli değildir. Bu bayraklar kütüphanenin çekirdekleri üzerine kod yazmak isteyen geliştiriciler için tasarlanmıştır.

a. ADC Hata Bayrakları ve Kullanımları

Bayrak Adı	Tanımı	Kalıtıldığı Veri Yapısı	Kullanım Yeri ve Sebebi
cpmgrErr	Capemgr ismi bulma hatası	errorCore > errorCoreADC	Kodunuzun içinde herhangi bir yerde capemgr ismini kullanmadan önce
ocpErr	Ocp ismi bulma hatası	errorCore > errorCoreADC	Kodunuzun içinde herhangi bir yerde ocp ismini kullanmadan önce
dtErr	Device tree yükleme hatası	errorCoreADC	BlackADC sınıfından nesne üretildikten sonra, device tree yükleme işleminin kontrolü için
helperErr	Helper ismi bulma hatası	errorCoreADC	BlackADC sınıfından nesne üretildikten ve device tree yükleme işleminden sonra, ADC driver kontrolü için
readErr	Dosya okuma hatası	----	Analog değer okuma işlemi yapıldıktan sonra doğruluk kontrolü için[1]

[1] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.

- **getValue()**
- **getNumericValue()**
- **getParsedValue()**

b. PWM Hata Bayrakları ve Kullanımları

Bayrak Adı	Tanımı	Kalıtıldığı Veri Yapısı	Kullanım Yeri ve Sebebi
cpmgrErr	Capemgr ismi bulma hatası	errorCore > errorCorePWM	Kodunuzun içinde herhangi bir yerde capemgr ismini kullanmadan önce
ocpErr	Ocp ismi bulma hatası	errorCore > errorCorePWM	Kodunuzun içinde herhangi bir yerde ocp ismini kullanmadan önce
dtErr	Device tree yükleme hatası	errorCorePWM	BlackPWM sınıfından nesne üretildikten sonra, device tree yükleme işleminin kontrolü için
dtSubSystemErr	Device tree yükleme hatası	errorCorePWM	BlackPWM sınıfından nesne üretildikten sonra, device tree 'ye yapılan ikinci yükleme işleminin kontrolü için
pwmTestErr	pwm_test ismi bulma hatası	errorCorePWM	BlackPWM sınıfından nesne üretildikten ve device tree yükleme işlemlerinden sonra, PWM driver kontrolü için
initializeErr	İlklendirme hatası	errorCorePWM	Eğer pwm bir kez daha ilklendirilmeye çalışılırsa bu hata oluşur. Normal şartlarda BlackPWM sınıfından sadece bir kere ilklendirme yapılabilir.
periodFileErr	Periyot dosyası açma hatası	----	Periyot dosyası açma işlemi yapan bir fonksiyon çağırıldıktan sonra[1]
dutyFileErr	Duty dosyası açma hatası	----	Duty dosyası açma işlemi yapan bir fonksiyon çağırıldıktan sonra[2]

[1] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.

- **getValue()**
- **getNumericValue()**
- **getPeriodValue()**
- **getNumericPeriodValue()**
- **setPeriodTime()**
- **setSpaceRatioTime()**
- **setLoadRatioTime()**

[2] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.

- **getValue()**
- **getNumericValue()**
- **getDutyValue()**
- **getNumericDutyValue()**
- **setDutyPercent()**
- **setPeriodTime()**
- **setSpaceRatioTime()**
- **setLoadRatioTime()**

runFileErr	Çalışma dosyası açma hatası	----	Çalışma dosyası açma işlemi yapan bir fonksiyon çağırıldıktan sonra[1]
polarityFileErr	Polarite dosyası açma hatası	----	Polarite dosyası açma işlemi yapan bir fonksiyon çağırıldıktan sonra[2]
outOfRangeErr	Girilen değer aralık hatası	----	Sayısal değişken alan ve bu değişkeni ayar yapmak için kullanan bir fonksiyon çağırıldıktan sonra [3]

[1] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.

- **getRunValue()**
- **isRunning()**
- **setRunState()**
- **toggleRunState()**

[2] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.

- **getPolarityValue()**
- **isPolarityStraight()**
- **isPolarityReverse()**
- **setPolarity()**
- **tooglePolarity()**

[3] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.

- **setDutyPercent()**
- **setPeriodTime()**
- **setSpaceRatioTime()**
- **setLoadRatioTime()**

c. GPIO Hata Bayrakları ve Kullanımları

Bayrak Adı	Tanımı	Kalıtıldığı Veri Yapısı	Kullanım Yeri ve Sebebi
exportFileErr	Export dosyası açma hatası	errorCoreGPIO	Pin çıkarma işlemi yapıldıktan sonra
directionFileErr	Yön dosyası açma hatası	errorCoreGPIO	Pin yön ayarlama işlemi yapıldıktan sonra
initializeErr	İlklendirme hatası	errorCoreGPIO	Eğer GPIO bir kez daha ilklendirilmeye çalışılırsa bu hata oluşur. Normal şartlarda BlackGPIO sınıfından sadece bir kere ilklendirme yapılabilir.
exportErr	Pin export hatası	----	Okuma veya yazma hatası oluşursa, bu hatayı kontrol edebilirsiniz. Bu hatalara ek olarak, eğer exportErr hatası da varsa bu durum pinin export edilmediğini ya da sınıftan nesne üretildikten sonra el ile unexport edildiğini gösterir.[1]
directionErr	Pin yönü hatası	----	Okuma veya yazma hatası oluşursa, bu hatayı kontrol edebilirsiniz. Bu hatalara ek olarak, eğer directionErr hatası da varsa bu durum, sınıftan nesne üretildikten sonra el ile pin yönünün değiştirildiğini gösterir.[2]
readErr	Değer dosyası okuma hatası	----	Okuma işlemi yapıldıktan sonra doğruluk kontrolü için[3]
writeErr	Değer dosyası yazma hatası	----	Yazma işlemi yapıldıktan sonra doğruluk kontrolü için[4]
forcingErr	Pin değeri zorlama hatası	----	Eğer giriş tipi bir pine değer yazmaya çalışırsanız oluşacak hatadır. Yazma işleminden sonra kullanılabilir.[5]

[1] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.
- **getValue()**
- **isHigh()**
- **toggleValue()**
- **setValue()**

[2] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.
- **getValue()**
- **isHigh()**
- **toggleValue()**
- **setValue()**

[3] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.
- **getValue()**
- **isHigh()**
- **toggleValue()**

[4] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.
- **setValue()**

[5] Bu bayrağı şu fonksiyonlardan sonra kontrol edebilirsiniz.
- **setValue()**

GPIO bayraklarından bazıları aynı anda 1 durumunda olabilirler. Bu gibi durumlarda hatanın gerçek sebebini doğru bir şekilde anlamanız için, bu bayrakları birlikte değerlendirmeniz gerekmektedir. Hata takibi algoritması bayrakları, gerçekleşen problemleri göz önünde bulundurarak, doğru şekilde ayarlar. Bundan dolayı bu kombinasyonların bazılarının kendilerine özel anlamları vardır. Bu durum sadece GPIO bayrakları için geçerlidir. Bu kombinasyonların kullanımı için, doküman dosyaları klasörünün içinde yer alan "GpioBayraklarıKullanımTablosu" dosyası hazırlanmıştır.

7. KAYNAKÇA

[1] «Device Tree,» [Çevrimiçi]. Available: http://www.devicetree.org/Main_Page.

[2] «C++11,» Wikipedia, 2014. [Çevrimiçi]. Available: http://en.wikipedia.org/wiki/C%2B%2B11.

[3] «Compiling C++11 with g++ - Stack Overflow,» Stack Overflow, 2012. [Çevrimiçi]. Available: http://stackoverflow.com/questions/10363646/compiling-c11-with-g.

[4] «Eclipse CDT C++11/C++0x support - Stack Overflow,» Stack Overflow, 2012. [Çevrimiçi]. Available: http://stackoverflow.com/questions/9131763/eclipse-cdt-c11-c0x-support.