



# BlackLib - Kullanım Kılavuzu

---

Beaglebone Black C++ Kütüphanesi

Yazar: Yiğit Yüce

22.04.2014

[blacklib.yigityuce.com](http://blacklib.yigityuce.com)

v1.0



## İÇİNDEKİLER

1. ÖZET .....	3
2. ANAHTAR KELİMELER .....	3
3. GİRİŞ .....	3
4. LİSANS.....	3
5. BİLİNMESİ GEREKENLER .....	4
1. Device Tree .....	4
2. C++11.....	4
6. KULLANIM.....	4
1. Kütüphaneyi Dâhil Etme .....	4
2. Özel Tanımlamalar.....	5
a. ADC Tanımlamaları .....	5
b. PWM Tanımlamaları .....	6
c. GPIO Tanımlamaları .....	7
3. Son Kullanıcı İle Etkileşime Geçen Fonksiyonlar .....	8
a. ADC Yönetim Fonksiyonları .....	8
b. PWM Yönetim Fonksiyonları.....	10
c. GPIO Yönetim Fonksiyonları.....	17
4. Hata Ayıklama ve Hata Bayrakları .....	19
a. ADC Hata Bayrakları ve Kullanımları .....	19
b. PWM Hata Bayrakları ve Kullanımları .....	20
c. GPIO Hata Bayrakları ve Kullanımları.....	22
7. KAYNAKÇA .....	23

## 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 kullanılabilir. Kernel v3.7 ve sonrasını destekleyen Beaglebone Black cihazlarında çalışabilmektedir ve LGPL lisansı ile lisanslanmıştır.

## 4. LİSANS

BlackLib Library controls Beaglebone Black input and outputs Copyright (C) 2013-2014 by Yigit YUCE

This file is part of BlackLib library. BlackLib library is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. BlackLib library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

For any comment or suggestion please contact the creator of BlackLib Library at [ygytyce@gmail.com](mailto:ygytyce@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 yapıları 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(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

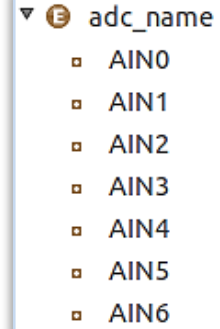
Beaglebone Black üzerindeki özelliklerden kullanılmak istenen seçildikten sonra, bu özelliğe ait alt seçenekler, yeri geldiği zaman seçilmelidir. Bizim uygulamamızda bu seçenekler, özel tanımlamalar ile seçilebilmektedir. Her bir özellik için kullanılabilecek seçenekler özel olarak tanımlanmıştır ve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

### a. ADC Tanımlamaları

adc\_name 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.

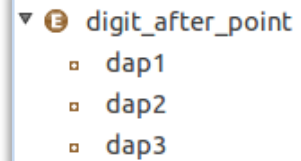
Adc isim seçimi, kodunuzda BlackADC sınıfından nesne üretilirken sizden beklenen ilk ve tek parametredir. Bu parametre "adc\_name" cinsinden bir değişken olabilmektedir. adc\_name seçeneğinin kullanımı aşağıdaki gibidir.

- 
- ▼ E adc\_name
    - AIN0
    - AIN1
    - AIN2
    - AIN3
    - AIN4
    - AIN5
    - AIN6

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

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

ADC okunurken, 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 volta çevrilmiş veride noktadan sonra kaç basamak görüntülenmesi istendiği belirtilmektedir. Aşağıda verilen örnek, bu fonksiyon ve "digit\_after\_point" değişkeninin kullanımı ile ilgilidir.

- 
- ▼ E digit\_after\_point
    - dap1
    - dap2
    - dap3

```
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 kullanılabılır 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.

- ▼ pwm\_pin\_name
  - P8\_13
  - P8\_19
  - P9\_14
  - P9\_16
  - P9\_21
  - P9\_22
  - P9\_42

- ▼ pwm\_bus\_name
  - EHRPWM2B
  - EHRPWM2A
  - EHRPWM1A
  - EHRPWM1B
  - EHRPWM0B
  - EHRPWM0A
  - ECAP0

```
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 “setRunState()” 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.

- ▼ pwm\_value
  - stop
  - run

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

### c. GPIO Tanımlamaları

“*pin\_name*”, GPIO pinlerinden, hangisinin seçileceğini 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.

pin_name	
GPIO_30	GPIO_26
GPIO_60	GPIO_47
GPIO_31	GPIO_46
GPIO_40	GPIO_27
GPIO_48	GPIO_65
GPIO_51	GPIO_22
GPIO_4	GPIO_63
GPIO_5	GPIO_62
GPIO_3	GPIO_37
GPIO_2	GPIO_36
GPIO_49	GPIO_33
GPIO_15	GPIO_32
GPIO_117	GPIO_61
GPIO_14	GPIO_86
GPIO_125	GPIO_88
GPIO_123	GPIO_87
GPIO_121	GPIO_89
GPIO_122	GPIO_10
GPIO_120	GPIO_11
GPIO_20	GPIO_9
GPIO_7	GPIO_81
GPIO_38	GPIO_8
GPIO_39	GPIO_80
GPIO_34	GPIO_78
GPIO_35	GPIO_79
GPIO_66	GPIO_76
GPIO_67	GPIO_77
GPIO_69	GPIO_74
GPIO_68	GPIO_75
GPIO_45	GPIO_72
GPIO_44	GPIO_73
GPIO_23	GPIO_70
	GPIO_71

```
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.

pin_type	pin_value
input	low
output	high

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

### 3. Son Kullanıcı İle Etkileşime Geçen Fonksiyonlar

#### a. ADC Yönetim Fonksiyonları

##### i. BlackADC::getValue() : string

Bu fonksiyon analog giriş değerini döndürür. 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 döndürür. 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 döndürür. 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, diğer durumlarda yanlış 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 çağırdıktan sonra bunu kullanabilirsiniz. **flags** tipi değişken alır ve bu değişken seçilen 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 döndürür. 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 döndürür. BlackPWM::periodPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya pwm sinyalinin periyot değerini nanosaniye(ns) mertebesinde tutar. String tipi, 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 döndürür. BlackPWM::dutyPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya pwm sinyalinin duty değerini nanosaniye(ns) mertebesinde tutar. String tipi, 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 durumunu döndürür. BlackPWM::runPath değişkeninde tanımlı konumdaki dosyayı okur. Bu dosya pwm sinyalinin çalışma durumunu 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 döndürür. 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 döndürür. Pwm sinyalinin periyot ve duty değerini bulmak için getNumericPeriodValue() ve getNumericDutyValue() fonksiyonlarını çağırır. Bunu yaptıktan sonra bu değerleri kullanarak yüzdelik değeri hesaplar. Float tipi, doluluk boşluk oranı yüzdesini döndürür.

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

#### vii. BlackPWM::getNumericPeriodValue () : uint32\_t

Bu fonksiyon pwm sinyalinin periyot değerini sayısal olarak döndürür. Pwm sinyalinin periyot değerini bulmak için getPeriodValue() fonksiyonunu çağırır. Bunu yaptıktan sonra, bu periyot değeri, strtoumax() fonksiyonu kullanılarak uint32\_t tipine çevrilir. uint32\_t tipindeki periyot değerini 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 döndürür. Pwm sinyalinin duty değerini bulmak için getDutyValue() fonksiyonunu çağırır. Bunu yaptıktan sonra bu duty değeri, strtoumax() fonksiyonu kullanılarak uint32\_t tipine çevrilir. uint32\_t tipindeki duty değerini 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 yüzde 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. Yeni değer ayarlama başarılı olursa doğru, diğer durumlarda yanlış 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. Yeni değer ayarlama başarılı olursa doğru, diğer durumlarda yanlış 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 kaydederek yeni duty değerini ayarlar. **uint32\_t** tipinde bir değişken alır ve bu değişken yeni boşluk süresi değerini ayarlamak için kullanılır. Yeni değer ayarlama başarılı olursa doğru, diğer durumlarda yanlış 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 değer, duty dosyasına kaydedilir. **uint32\_t** tipinde bir değişken alır ve bu değişken yeni çalışma süresi değerini ayarlamak için kullanılır. Yeni değer ayarlama başarılı olursa doğru, diğer durumlarda yanlış döndürür.

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

### xiii. **BlackPWM::setPolarity (pwm\_polarity) : bool**

Bu fonksiyon pwm sinyalinin polaritesini ayarlar. Polarite değerini 1 ya da 0 yapar. Gönderilen parametre 1 veya 0'a çevrilir ve bu değer polarite dosyasına kaydedilir. **pwm\_polarity** tipinde değişken alır ve bu değişken polarite tipini ayarlamak için kullanılır. Yeni polarite ayarlama başarılı olursa doğru, diğer durumlarda yanlış döndürür.

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

### xiv. **BlackPWM::setRunState (pwm\_value) : bool**

Bu fonksiyon pwm sinyalinin çalışma durumunu ayarlar. Çalışma değerini 1 ya da 0 yapar. Gönderilen parametre 1 veya 0'a çevrilir ve bu değer çalışma durumunun tutulduğu dosyaya kaydedilir. **pwm\_value** tipinde değişken alır ve bu değişken çalışma durumunu ayarlamak için kullanılır. Yeni çalışma durumu ayarlama başarılı olursa doğru, diğer durumlarda yanlış döndürür.

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

### xv. **BlackPWM::toggleRunState () : void**

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

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

### xvi. **BlackPWM::togglePolarity () : void**

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

```
test->togglePolarity();
```

#### xvii. BlackPWM::isRunning() : bool

Bu fonksiyon pwm sinyalinin çalışma durumunu kontrol eder. getRunValue() fonksiyonunu çağırır ve bu fonksiyonun geri dönüş değerini değerlendirir. Geri dönüş değeri 0 ise yanlış, diğer durumlarda ise doğru 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. Geri dönüş değeri 0 ise doğru, diğer durumlarda ise yanlış 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. Geri dönüş değeri 0 ise yanlış, diğer durumlarda ise doğru 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, diğer durumlarda yanlış 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 çağırdıktan sonra kullanabilirsiniz. **flags** tipi değişken alır ve bu değişken seçilen bir hatanın durumunu öğrenmek için kullanılır. Seçilen hatanın durumunu döndürür.

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

### c. GPIO Yönetim Fonksiyonları

#### i. BlackGPIO::getValue() : string

Bu fonksiyon GPIO pin değerini döndürür. 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, pin değerini 1 ya da 0 yapar. Gönderilen parametre 1 veya 0'a çevrilir ve değer dosyasına yazılır. **gpio\_value** tipinde bir değişken alır ve bu değişken pin değerini ayarlamak için kullanılır. Yeni değer ayarlama başarılı olursa doğru, diğer durumlarda yanlış 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, diğer durumlarda doğru döndürür.

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

#### iv. BlackGPIO::toggleValue () : void

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

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

#### v. BlackGPIO::fail () : bool

Bu fonksiyon genel hata ayıklama işlemi için kullanılır. Eğer herhangi bir hata meydana geldiyse doğru, diğer durumlarda yanlış 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 çağırdıktan sonra kullanabilirsiniz. **flags** tipi değişken alır ve bu değişken seçilen bir hatanın durumunu öğrenmek için kullanılır. Seçilen hatanın durumunu döndürür.

```
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 hatası 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	Tanımlandığı Veri Yapısı	Kullanım Yeri ve Sebebi
<b><i>cpmgrErr</i></b>	Capemgr ismi bulma hatası	<code>errorCore</code>	Kodunuzun içinde herhangi bir yerde <code>capemgr</code> ismini kullanmadan önce
<b><i>ocpErr</i></b>	Ocp ismi bulma hatası	<code>errorCore</code>	Kodunuzun içinde herhangi bir yerde <code>ocp</code> ismini kullanmadan önce
<b><i>dtErr</i></b>	Device tree yükleme hatası	<code>errorCoreADC</code>	BlackADC sınıfından nesne üretildikten sonra, device tree yükleme işleminin kontrolü için
<b><i>helperErr</i></b>	Helper ismi bulma hatası	<code>errorCoreADC</code>	BlackADC sınıfından nesne üretildikten ve device tree yükleme işleminden sonra, ADC driver kontrolü için
<b><i>readErr</i></b>	Dosya okuma hatası	<code>errorADC</code>	Analog değer okuma işlemi yapıldıktan sonra doğruluk kontrolü için <sup>1</sup>

<sup>1</sup> Bu bayrağı şu fonksiyonları çağırıdıktan sonra kontrol edebilirsiniz.

- › `getValue()`
- › `getNumericValue()`
- › `getParsedValue()`

### b. PWM Hata Bayrakları ve Kullanımları

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

<sup>1</sup> Bu bayrağı şu fonksiyonları çağırdıktan sonra kontrol edebilirsiniz.

```
> getValue()  
> getNumericValue()  
> getPeriodValue()  
> getNumericPeriodValue()  
> setPeriodTime()  
> setSpaceRatioTime()  
> setLoadRatioTime()
```

<sup>2</sup> Bu bayrağı şu fonksiyonları çağırdıktan sonra kontrol edebilirsiniz.

```
> getValue()  
> getNumericValue()  
> getDutyValue()  
> getNumericDutyValue()  
> setDutyPercent()  
> setPeriodTime()  
> setSpaceRatioTime()  
> setLoadRatioTime()
```

<b><i>runFileErr</i></b>	Çalışma dosyası açma hatası	errorPWM	Çalışma dosyası açma işlemi yapan bir fonksiyon çağırıldıktan sonra <sup>1</sup>
<b><i>polarityFileErr</i></b>	Polarite dosyası açma hatası	errorPWM	Polarite dosyası açma işlemi yapan bir fonksiyon çağırıldıktan sonra <sup>2</sup>
<b><i>outOfRangeErr</i></b>	Girilen değer aralık hatası	errorPWM	Sayısal değişken alan ve bu değişkeni ayar yapmak için kullanan bir fonksiyon çağırıldıktan sonra <sup>3</sup>

---

<sup>1</sup> Bu bayrağı şu fonksiyonları çağırıldıktan sonra kontrol edebilirsiniz.

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

<sup>2</sup> Bu bayrağı şu fonksiyonları çağırıldıktan sonra kontrol edebilirsiniz.

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

<sup>3</sup> Bu bayrağı şu fonksiyonları çağırıldıktan sonra kontrol edebilirsiniz.

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

### c. GPIO Hata Bayrakları ve Kullanımları

Bayrak Adı	Tanımı	Tanımlandığı Veri Yapısı	Kullanım Yeri ve Sebebi
<b>exportFileErr</b>	Export dosyası açma hatası	errorCoreGPIO	Pin çıkarma işlemi yapıldıktan sonra
<b>directionFileErr</b>	Yön dosyası açma hatası	errorCoreGPIO	Pin yön ayarlama işlemi yapıldıktan sonra
<b>initializeErr</b>	İklendirme hatası	errorCoreGPIO	Eğer GPIO bir kez daha iklendirilmeye çalışılırsa bu hata oluşur. Normal şartlarda BlackGPIO sınıfından sadece bir kere iklendirme yapılabilir.
<b>exportErr</b>	Pin export hatası	errorGPIO	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. <sup>1</sup>
<b>directionErr</b>	Pin yönü hatası	errorGPIO	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. <sup>2</sup>
<b>readErr</b>	Değer dosyası okuma hatası	errorGPIO	Okuma işlemi yapıldıktan sonra doğruluk kontrolü için <sup>3</sup>
<b>writeErr</b>	Değer dosyası yazma hatası	errorGPIO	Yazma işlemi yapıldıktan sonra doğruluk kontrolü için <sup>4</sup>
<b>forcingErr</b>	Pin değeri zorlama hatası	errorGPIO	Eğer giriş tipi bir pine değer yazmaya çalışırsanız oluşacak hatadır. Yazma işleminden sonra kullanılabilir. <sup>5</sup>

<sup>1</sup> Bu bayrağı şu fonksiyonları çağırdıktan sonra kontrol edebilirsiniz.

```
> getValue()  
> isHigh()  
> toggleValue()  
> setValue()
```

<sup>2</sup> Bu bayrağı şu fonksiyonları çağırdıktan sonra kontrol edebilirsiniz.

```
> getValue()  
> isHigh()  
> toggleValue()  
> setValue()
```

<sup>3</sup> Bu bayrağı şu fonksiyonları çağırdıktan sonra kontrol edebilirsiniz.

```
> getValue()  
> isHigh()  
> toggleValue()
```

<sup>4</sup> Bu bayrağı şu fonksiyonları çağırdıktan sonra kontrol edebilirsiniz.

```
> setValue()
```

<sup>5</sup> Bu bayrağı şu fonksiyonları çağırdıktan 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, gerçekleşen problemleri göz önünde bulundurarak bayrakları 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.

Detaylı teknik doküman için lütfen internet sitesini inceleyiniz. [blacklib.yigityuce.com](http://blacklib.yigityuce.com)

## 7. KAYNAKÇA

- [1] «Device Tree,» [Çevrimiçi]. Available: [http://www.devicetree.org/Main\\_Page](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>.