RCoS+ 3.0 Kullanım Kılavuzu

arçelik Embedded Software Framework

RCoS+ 3.0 USER MANUAL

Arçelik Embedded Platform Framework

10.01.2024

Prepared By: Arçelik A.Ş.

R&D Center

34950

Tuzla/Istanbul/Turkey

ÖZET

Gerçek Zamanlı İşletim Sistemleri (RTOS), gömülü sistemlerde, kritik uygulamalarda ve çeşitli gerçek zamanlı bilgi işlem ortamlarında görevlerin zamanında ve öngörülebilir şekilde yürütülmesini sağlamada çok önemli bir rol oynar.

Arçelik, gelecekte tüm ürünlerini tek bir ana yazılım çerçevesine bağlamanın kendi ekosistemini yaratacağına inanıyor. Bu ekosistem sayesinde daha iyi ve daha kolay yazılımlar geliştirilebilir, ürün bakım süreleri kısaltılabilir, güvenli, hızlı, kaliteli ürünler üretilebilir.

Bu belge, RCoS+ 3.0 Arçelik Gömülü Platform Çerçevesi altyapısına yönelik yazılım tasarım prosedürünün temel fonksiyonlarını ve tanımlarını sağlar. Bu belgede sadece genel altyapılar tanımlanmıştır. Ayrıca yazılım için başka bir detaya ihtiyaç duyulursa Merkezi Ar-Ge yazılım uzmanları tarafından sağlanabilmektedir. Bu belge yalnızca Merkezi Ar-Ge Elektronik Grup Yönetimi tarafından yayınlanmış ve değiştirilmiştir.

1. GİRİŞ

RCoS+ 3.0 gerçek zamanlı gömülü bir işletim sistemidir. Hedefleri şunlardır:

• RCOS+ 3.0 ürünlerimizin yeniden kullanılabilirlik, modülerlik vb. özellikleri içeren modern yazılım ihtiyaçlarına yönelik gereksinimlerin belirlenmesi ve gerekli yazılım yapısının tanımlanarak ürünlerimize uygulanmasını amaçlayan bir projedir.

• Kodlaması kolaydır.

• Hataları düzeltmek kolaydır.

• Kodlamayı değiştirmek kolaydır.

• Her ne kadar birçok ürünün kendine ait yazılımı olsa da hepsinin ortak bir yazılımı olması mümkündür.

• RCoS+ 3.0, Donanım değişikliklerine hızla uyum sağlayabilir.

• RCoS+ 3.0 güvenilir yazılım bileşenlerine sahiptir.

RCoS+ 3.0 kullanmanın temel amacı gelecekte her ürün için ortak bir yazılım çerçevesi oluşturmaktır.

İÇERİK

2. YAPI

RCoS+'ta process'ler ve event'ler vardır. Yapılacak görevler sırasıyla events queue'ya eklenir. Process'ler handler'ları içerir, handler'lar ise içindeki event'leri içerir. Event'ler, "timer event" yardımıyla tekrar tekrar event queue'ya eklenir. Timer event 7. bölümde detaylandırılacaktır. Process'in kullanılmasının asıl amacı görevin konusunu tanımlamaktır.

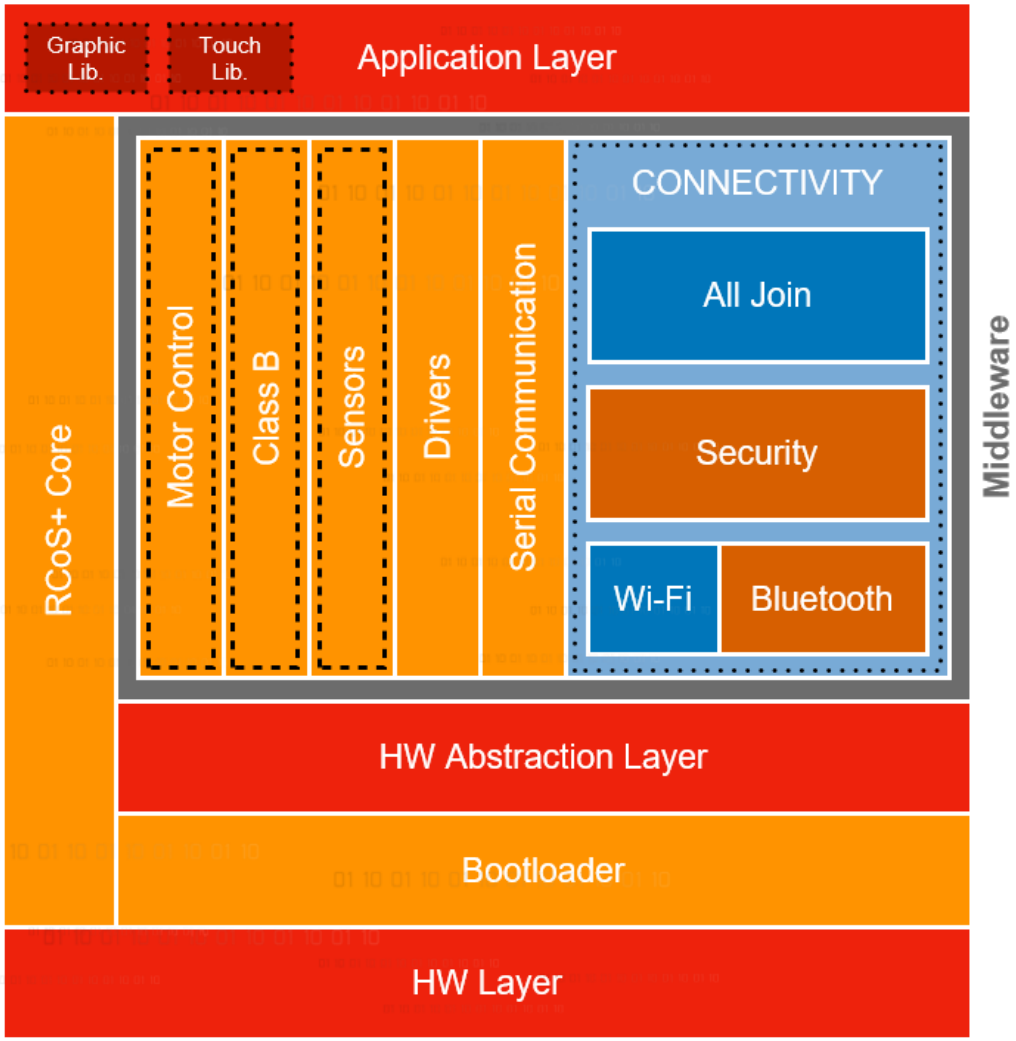
3. ÇEKİRDEK

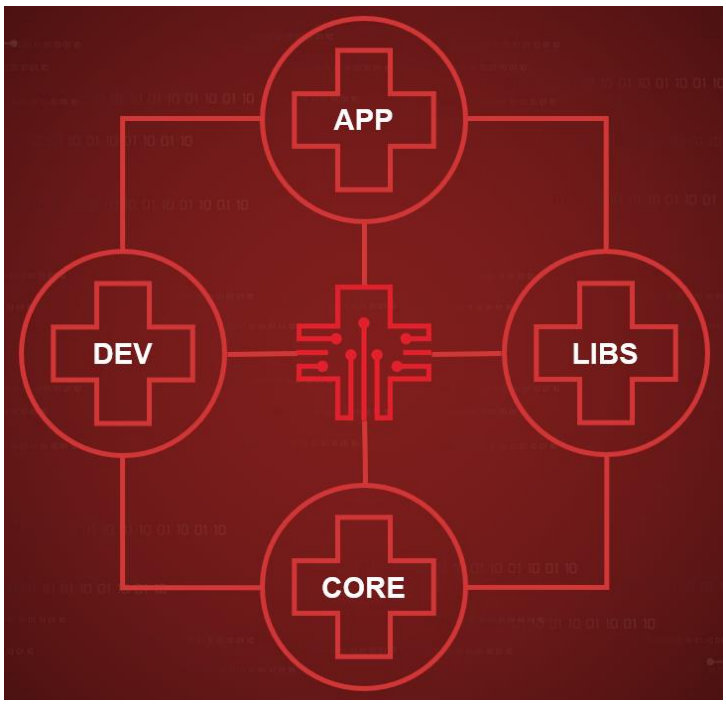
core.c, core.h, RCoS+ 3.0'ın timer, event, process, protothread ve MW işlevleri gibi tüm dahili/harici işlevlerini oluşturmak için kullanılır.

4. UYGULAMALAR

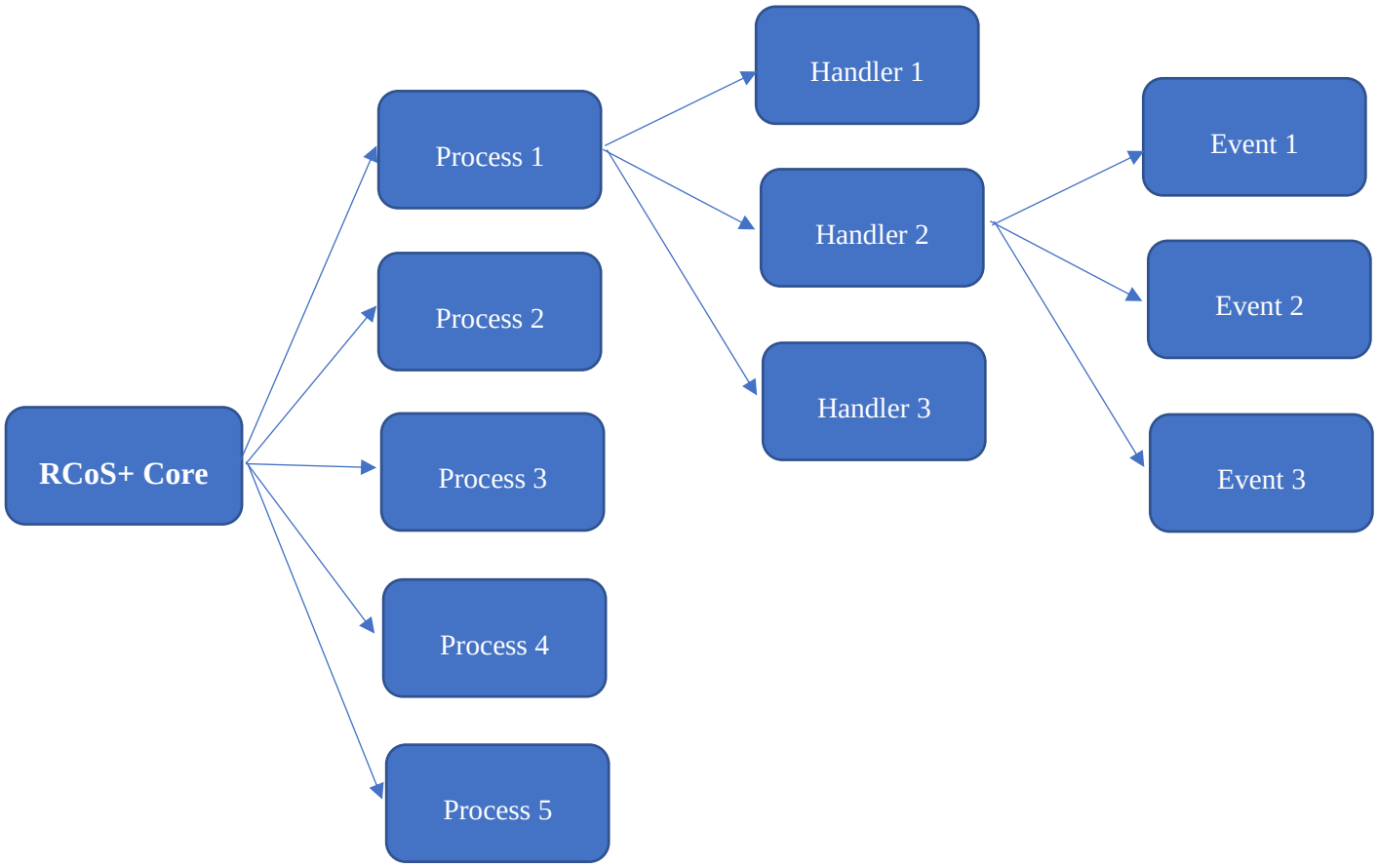
CORE, LIBS ve DEV tarafından sağlanan altyapılar kullanılarak geliştirilen uygulamalardan oluşan bir kütüphane. Bu isim altında hem alana özel uygulamalar hem de RCoS+ 3.0 ile sağlanan genel amaçlı hazır uygulamalar yer alacak. Proje oluşturulurken ihtiyaç duyulan uygulamalar projeye eklenebilir ve temel özellikleri sağlayan uygulamalar hızlı bir şekilde çalıştırılabilir.

- Udaq
- Serial Communication
- Data Linker
- Backup
- Multi Chart
- Uart echo





RCoS+ Yapısı



4.1 MIDDLEWARE

4.1.1 backup.h

Başlatma sırasında yedekleme işlemi, tüm veri öğelerini geri yüklemek için NVM'yi okumaya çalışır. Başlatma sonrasında, eğer değişen herhangi bir veri öğesi varsa, her saniye bir güncelleme işlemi gerçekleştirir.

PROCESS_BACKUP_CREATE(_name, _mem, _start, _itemArr, _updatePeriod)

name → Process'in ismi
mem → kullanılacak NVM device'ı
start → Nvm'deki yedekleme alanının başlangıç adresi
itemArr → Yedekleme için bellek alanlarını seçmek için tsBackupItem dizisi
updatePeriod →

Yedekleme güncelleme işlemi için milisaniye cinsinden süre

#define BACKUP_SIGNATURE (0x2c05)
uint16_t signature; ///< Burada zaten bir yedekleme olup olmadığını kontrol et.
uint16_t itemArrId; ///< itemArr öğesinin değişip değişmediğini kontrol et
uint16_t totalSize; ///< Dizi ögelerinin toplam boyutu.
} tsBackupHeader;

4.1.2 buttons.h

Bir veya birden fazla düğmenin basılı durumunu tutan ve tanımlara göre karşılık gelen event'leri oluşturan bir devIo'yu okur.

PROCESS_BUTTONS_CREATE(_name, _devIo, _targetProcess, _resolution, ...)

name → Process'in ismi
devIo → butonların durumlarını almak için kullanılacak devIo cihaz adı
targetProcess → Basılan ve bırakılan düğmelere ilişkin bilgileri alacak hedef process numaralandırması
resolution → Basılan zaman bilgilerinin milisaniye cinsinden çözünürlüğü
... → Düğme tanımları, BUTTONS_ITEM'i kullanın

#define BUTTONS_ITEM(_val, _press, _release)

val → Bir düğmenin (veya düğme kombinasyonunun) 32 bitlik ham değeri
press → butona basılan an
release → butonun bırakıldığı an

4.1.3 com_bridge.h

Com-Bridge, birinden diğerine veri aktarmak için 2 devCom cihazını birbirine bağlamak için kullanılır.

PROCESS_COM_BRIDGE_CREATE(_name, _com1, _target1, _com2, _target2)

name→ Process'in ismi
com1 → ilk devCom
target1 → ilk target
com2 → ikinci devCom
target2 → ikinci target

4.1.4 csd_tuner.h

Ayrıntılar için " PSoC 4 Capacitive Sensing"i okuyun.

PROCESS_CSD_TUNER_CREATE(_name, _uart, _period, _cyName)

name → Process'in ismi
uart → Çıkış verilerin alınacağı UART device'ı
period → güncelleme periyodu.
cyName → CYPRESS PSoC Creator komponent ismi.

4.1.5 data_linker.h

Event'ler tarafından taşınan verileri geçici hafızaya(memory) bağlar.

PROCESS_DATA_LINKER_CREATE(_name, _enum, _dataSetArr)

name → Process'in ismi
enum → Kullanılacağı process'in numarası
dataSetArr → DataSet dizi adı (tsDataSet öğelerinin bir dizisi)

4.1.6 event_mapper.h

Event mapper process'i, bir eventi bir kaynaktan başka bir hedef için başka bir event'e çevirecektir..

PROCESS_EVENT_MAPPER_CREATE(_name, _enum, _array)

name → Process'in ismi
enum → Kullanılacağı process'in numarası
array → Eşleme için kullanılacak tsEventMapperItem öğelerini içeren dizi adı

4.1.7 log_csv.h

UART kanalı aracılığıyla CSV formatında bir çıktı oluşturun.

PROCESS_LOG_CSV_CREATE(_name, _uart, _target, _list, _period, _window)

name → Process'in ismi
uart → UART haberleşmesi için oluşturulmuş device
target → Communication target
list → LOG_ITEM makroları tarafından oluşturulan const tsLogItem dizisi
period → loglama periyodu
window → Pencere çizgisi boyutu

4.1.8 multi_chart.h

Multi Chart processi tarafından kaydedilecek verileri UART kanalı üzerinden gönderin.

PROCESS_MULTI_CHART_CREATE(_name, _uart, _target, _array, _period)

name → Process'in ismi
uart → Kullanılacak UART device'
target → UART target
array → Veri kaynaklarını belirten const tsMultiChartDataSource dizisinin adı
period → data yenileme periyodu

```
typedef struct
{
    uint16_t *raw;      ///< Sensör raw data
    uint16_t *baseline; ///< Sensör baseline değeri
    uint16_t *signal;   ///< sensörün mevcut sinyal durumu
} tsMultiChartDataSource;
```

4.1.9 sc_spi_sl.h

SPI Slave üzerinden seri iletişim protokolü.

PROCESS_SC_SPI_SL_CREATE(_name, _enum, _devCom, _devComTarget, _mapName, _queueSize)

name → Process'in ismi
enum → Process numarası
devCom → Kullanılacak SPI device'ı
devComTarget → SPI target
mapName → TsScMapItem nesneleri dizisinin adı
queueSize → Bayt cinsinden dahili queue boyutu.

```
typedef struct
{
   uint8_t identifier;   ///< Target frame
   tEventEnum event;     ///< Target event numarası
   tProcessEnum process; ///< Target process numarası
} tsScMapItem;
```

4.1.10 sc_spi.h

SPI Master üzerinden seri iletişim protokolü.

PROCESS_SC_SPI_CREATE(_name, _enum, _devCom, _devComTarget, _mapName, _queueSize)

name → Process'in ismi
enum → Process numarası
devCom → Kullanılacak SPI device'ı
devComTarget → SPI target
mapName → TsScMapItem nesneleri dizisinin adı
queueSize → Bayt cinsinden dahili queue boyutu.

```
typedef struct
{
   uint8_t identifier;   ///< Target frame
   tEventEnum event;     ///< Target event numarası
   tProcessEnum process; ///< Target process numarası
} tsScMapItem;
```

4.1.11 tft_spi.h

SPI Master üzerinden seri iletişim protokolü.

PROCESS_TFT_SPI_CREATE(_name, _enum, _devCom, _devComTarget, _tftBusy, _targetProcess, _receivedEvent, _errorEvent, _queueSize)

name → Process'in ismi
enum → Process numarası
devCom → Kullanılacak SPI device'ı
devComTarget → SPI target(CS pin)
tftBusy → Tft'nin busy sinyali gönderdiği IO device'ı.
targetProcess → Gelen tft board yanıtını alan target process
receivedEvent → Gelen tft board yanıtını işleyen target process event'i
errorEvent → İletişimde hata durumunu işleyen event
queueSize → İletişim queue boyutu.

4.1.12 uart_echo.h

Uart'tan veri alır ve geri döndürür.

PROCESS_UARTECHO_CREATE(_name, _uart)

name → Kullanılacağı process'in ismi
uart → Kullanılan UART device'ı

4.1.13 udaq.h

ARÇELİK Evrensel Veri Toplama ve Kontrol process'i.

PROCESS_UDAQ_CREATE(_name, _com, _target, _itemArray, _machineType, _modelNo, _modelId, _romVer, _eepVer, _cryptoKey)

name → Process'in ismi
com → Kullanılacak com device'ı
target → Kullanılacak Com device kanalı
itemArray → tsUdaqItem dizisinin adı
machineType → Makine tipi
modelNo → Model numarası
modelId → Model kimliği
romVer → ROM versionu
eepVer → EEPROM versionu
cryptoKey → UDAQ geçişlerinde kullanılacak kripto anahtarı

```
typedef struct
{
   void *address; ///< Address of item
   uint8_t size;  ///< Size of item
} tsUdaqItem;
```

5. DEV

RCoS+ 3.0 cihaz tanımlarına yönelik makrolar içerir. 4 farklı device tipi için temel fonksiyonlar ve nesne yapıları tanımlanmıştır.

Description	Doxygen Module Group	typedef
I/O	DEV_IO	tsDevIo
Communication	DEV_COM	tsDevCom
Memory	DEV_MEM	tsDevMem
Complex	DEV_CPX	tsDevCpx

5.1 DEV_IO

Input/Output device'ları anlık bilgi girişi ve/veya çıkışı sağlayan birimlerdir. 4 tür global fonksiyon içerir.

- devIoInit
- devIoDeinit
- devIoGet
- devIoPut

5.1.1 devIoInit

Kullanılacak tüm cihazların başlangıçta başlatılması gerekir. devIoInit(**const tsDevIo** *device, **const void** *config) işlevi kullanılarak başlatılabilir.

Örnek_1:

```
#include “gpio.h”

const tsDevIo *redLed; //IDE içinde oluşturulan I/O device’ın adı.
DEV_IO_GPIO_CREATE(name_redLed, redLed, 1)
devIoInit(&redLed,  NULL);
```

5.1.2 devIoDeinit

Başlatmama işlevi amaca aittir. Kullanıcı devIoDeinit(**const tsDevIo** *device) fonksiyonunu kullanara I/O cihazını sıfırlayabilir.

Example:

```
#include “gpio.h”

const tsDevIo *redLed; //IDE içinde oluşturulan I/O device’ın adı.
DEV_IO_GPIO_CREATE(name_redLed, redLed, 1)
devIoDeinit(&redLed);
```

5.1.3 devIoPut

I/O cihazları devIoPut işlevi kullanılarak yönetilebilir. Cihazın içinde saklanan verileri günceller. Kullanıcı devIoPut(**const tsDevIo** *device, **uint32_t** data) kullanarak cihaza uygun değer tipini koyabilir.

Example:

```
#include "gpio.h"

#define HIGH 1

const tsDevIo *redLed; //IDE içinde oluşturulan I/O device'ın adı.
DEV_IO_GPIO_CREATE(name_redLed, redLed, 1)
devIoPut(&redLed,  HIGH);
```

5.1.4 devIoGet

I/O cihazı devIoGet fonksiyonu kullanılarak izlenebilir. devIoGet(**const tsDevIo** *device) kullanılarak cihazın değerini döndürür.

Example:

```
#include "gpio.h"

const tsDevIo *redLed; //IDE içinde oluşturulan I/O device'ın adı.
DEV_IO_GPIO_CREATE(name_redLed, redLed, 1)
devIoGet(&redLed);
```

5.2 DEV_COM

Haberleşme cihazları, çevre birimlerle fiziksel ve/veya mantıksal düzeyde bağlantı sağlayan birimlerdir..

- devComInit
- devComDeinit
- DevComOpen
- devComClose
- devComSend
- devComReceive

5.2.1 devComInit

Kullanılacak tüm cihazların başlangıçta başlatılması gerekir. devComInit(**const tsDevCom** *device) fonksiyonu kullanılarak başlatılabilir.

Example:

```
#include "dev/psoc4/i2c.h"


const tsDevCom *I2C_device; //IDE içerisinde oluşturulan com device'ın adı.
DEV_COM_I2C_CREATE(name_I2C, I2C_device ,100)
devComInit(&name_I2C);
```

5.2.2 devComDeinit

Başlatmama işlevi amaca aittir. Kullanıcı devComDeinit(**const tsDevCom** *device) fonksiyonunu kullanarak haberleşme cihazını sıfırlayabilir.

Example:

```
#include "dev/psoc4/i2c.h"

const tsDevCom *I2C_device; //IDE içerisinde oluşturulan com device'ın adı.
DEV_COM_I2C_CREATE(name_I2C, I2C_device ,100)
devComDeinit(&name_I2C);
```

5.2.3 devComOpen

Haberleşme cihazları başlatma işleminden sonra açılmalıdır. Aksi halde cihazlar güç tüketecek ancak iletişim kuramayacaklardır.

Cihazlar devComOpen(**const tsDevCom** *device, **tsTarget** *target) fonksiyonu kullanılarak açılabilir.

*target iletişim hedefine bağlıdır. UART'ta hedef cihaz yoktur, NULL olarak doldurulabilir. Ancak I2C veya SPI gibi diğer iletişimler çoklu master veya çoklu slave çalışma prensibini kullanır. Bu nedenle devComOpen fonksiyonunun ikinci giriş argümanı olarak bir hedefe ihtiyacı vardır.

Example:

```
#include "dev/psoc4/i2c.h"

const tsDevCom *I2C_device; //IDE içerisinde oluşturulan com device'ın adı.
tsTarget *target_channel
DEV_COM_I2C_CREATE(name_I2C, I2C_device ,100)
devComOpen(&name_I2C,&target_channel);
```

5.2.4 devComClose

devComClose işlevi amaca bağlıdır.

devComClose(**const tsDevCom** *device, **tsTarget** *target) ile iletişim kanalını kapatır.

*target iletişim hedefine bağlıdır. UART'ta hedef cihaz yoktur, NULL olarak doldurulabilir. Ancak I2C veya SPI gibi diğer iletişimler çoklu master veya çoklu slave çalışma prensibini kullanır. Bu nedenle devComOpen fonksiyonunun ikinci giriş argümanı olarak bir hedefe ihtiyacı vardır.

Example:

```
#include "dev/psoc4/i2c.h"


const tsDevCom *I2C_device; //IDE içerisinde oluşturulan com device'ın adı.
tsTarget *target_channel
DEV_COM_I2C_CREATE(name_I2C, I2C_device ,100)
devComClose(&name_I2C, &target_channel);
```

5.2.5 devComSend

Başlatma işleminden ve cihazı açtıktan sonra veriler devComSend(**const tsDevCom** *device, **const void** *txb, **uint16_t** length) fonksiyonu kullanılarak gönderilebilir.

*txb alıcıya gönderilecek buffer'dır. "**uint16_t** length" buffer'ın uzunluğudur.

EXAMPLE:

```
#include "dev/psoc4/i2c.h"

char *buffer[50];
const tsDevCom *I2C_device; //IDE içerisinde oluşturulan com device'ın adı.
DEV_COM_I2C_CREATE(name_I2C, I2C_device ,100)
devComSend(name_I2C, buffer[0], 10);
```

5.2.6 devComReceive

İletişim cihazları verilerini;

devComReceive(**const tsDevCom** *device, **void** *rxb, **uint16_t** length) kullanarak alabilirler.

*rxb alınan verilerden yazılan buffer'dır. . "**uint16_t** length" buffer'ın uzunluğudur.

EXAMPLE:

```
#include "dev/psoc4/i2c.h"

char *buffer[50];
const tsDevCom *I2C_device; //IDE içerisinde oluşturulan com device'ın adı
DEV_COM_I2C_CREATE(name_I2C, I2C_device ,100)
devComReceive(name_I2C, buffer[0], 10);
```

5.3 DEV_MEM

Memory device'ları veri depolamak için kullanılan birimlerdir.

- devMemInit
- devMemDeinit
- DevMemRead
- devMemWrite
- devMemProgram
- devComErase

5.3.1 devMemInit

Kullanılacak tüm cihazların başlangıçta başlatılması gerekir.

devMemInit(**const tsDevMem** *device) fonksiyonu kullanılarak başlatılabilir.

EXAMPLE:

```
const tsDevMem*device;
devMemInit(&device);
```

5.3.2 devMemDeinit

Başlatmama işlevi amaca aittir. Kullanıcı devMemDeinit(**const tsDevMem** *device) fonksiyonu ile device'ı sıfırlayabilir.

```
const tsDevMem*device;
devMemDeinit(&device);
```

5.3.3 devMemRead

Cihazlarda tutulan değerlere devMemRead(**const tsDevMem** *device, **uint32_t** address, **void** *readData**, uint16_t** length) fonksiyonu kullanılarak erişilebilir

"**uint32_t** address" memory device adresidir, "**void** *readData" verinin okunacağı buffer pointer'dır ve "**uint16_t** length" okunacak verinin uzunluğudur.

5.3.4 devMemWrite

Değişkenler devMemWrite(**const tsDevMem** *device, **uint32_t** address, **void** *writeData**, uint16_t** length) kullanılarak hafıza cihazına yazılabilir.

"**uint32_t** address" memory device adresidir, "**void** *writeData" verinin yazıldığı buffer pointer'dır ve "**uint16_t** length" yazılacak verinin uzunluğudur.

5.3.5 devMemProgram

5.3.6 devMemErase

5.4 DEV_CPX

Complex device'lar, üç temel cihaz kategorisine uymayan ve/veya benzersiz özelliklere sahip birimlerdir.

- DevCpxInit
- DevCpxDeinit

5.4.1 DevCpxInit

Kullanılacak tüm cihazların başlangıçta başlatılması gerekir. devCpxInit(**const tsDevCpx ***device**, const void ***config) fonksiyonu kullanılarak başlatılabilir.

EXAMPLE:

```
#include"dev/psoc4/adc.h"
#include"dev/psoc4/gpio.h"

const tsDevCpx *ADC_SAR;  //IDE içerisinde oluşturulan complex device'ın adı.

DEV_CPX_ADC_CORE_CREATE(core_ADC, ADC_SAR)
DEV_IO_ADC_CREATE(ADC_device, core_ADC, 0)

devCpxInit(&core_ADC,  NULL);
devIoInit(&ADC_device,  NULL);
```

5.4.2 DevCpxDeinit

Başlatmama işlevi amaca aittir. Kullanıcı complex device'ı devCpxDeinit(**const tsDevCpx** *device) fonksiyonunu kullanarak sıfırlayabilir.

```
#include"dev/psoc4/adc.h"
#include"dev/psoc4/gpio.h"

const tsDevCpx *ADC_SAR;  //IDE içerisinde oluşturulan complex device'ın adı.

DEV_CPX_ADC_CORE_CREATE(core_ADC, ADC_SAR)
DEV_IO_ADC_CREATE(ADC_device, core_ADC, 0)

devCpxDeinit(&core_ADC);
devIoDeinit(&ADC_device);
```

5.5 İŞLEMCİYE ÖZEL OLMAYAN DEVICE'LAR

rcos/dev klasörü içerisinde yer alan ve her çeşit işlemci ile kullanılabilen yazılımlardır.

5.5.1 24cxx.h

24cXX EEPROM device sürücüsü

DEV_MEM_24CXX_CREATE(_name, _i2c, _target, _aaa, _xx)

aaa → I2C slave adresi değiştirici
xx → EEPROM kodu

5.5.2 accelerate.h

Giriş değerine bağlı olarak hedef devIo'nun değerini sürekli olarak değiştirir.

DEV_IO_ACCELERATE_CREATE(_name, _target, _timeout)

target → Hedef devIo
timeout → Değer güncelleme periyotu

5.5.3 apos.h

Analog Konum - Bir giriş devIo değerinin belirli aralıklarını numaralandırmaya dönüştürür.

DEV_IO_APOS_CREATE(_name, _devIo, ...)

Name → Device ismi
devIo → Hedef ADC device
... → devIo device'ını kullanan tanımlanmış aralıkların listesi APOS_RANGE

APOS_RANGE(_n, _x)

n → Bu menzildeki minimum değer
x → Bu menzildeki maksimum değer

5.5.4 bdc_motor

Bu device'ın amacı brushed bir DC motoru basit bir arayüz ile kontrol etmektir (durdur, saat yönünde döndür, saat yönünün tersinde döndür).

Bu cihaz, motorun yönünü değiştirmeden veya motora yeniden enerji vermeden önce kullanılan bir dead time uygular.

DEV_IO_BDC_MOTOR_CREATE(_name, _cwDev, _ccwDev, _deadTimeMs)

name → Device ismi
cwDev → Motoru 1 yazarak saat yönünde çevirmeye yarayan cihaz
ccwDev → Motoru 1 yazarak saat yönünün tersine çevirmeye yarayan cihaz
deadTimeMs → Motorun yönünü değiştirmeden veya yeniden enerji vermeden önce beklenecek ölü süre

5.5.5 bu16501ks2.h

I2C uyumlu ve 3-wire Serial Interface ile kullanılabilen 8x16 LEDs in Dot Matrix.

DEV_CPX_BU16501KS2_CREATE(_name, _i2c, _config, _ceLvl)

name → Device ismi
i2c → I2C communication device'ı
config → I2C konfigürasyonları
ceLvl → chip CE sinyal seviyesi

5.5.6 contdebounce.h

Birden fazla I/O device için hazırlanmış **sürekli** “debounce(geri dönüş” yapan bir kütüphane.

DEV_CPX_CONT_DEBOUNCE_CREATE(_name, _period, ...)

name → Device ismi
period → DevIo dizisini kontrol etmek için her dönemin süresi
... → CONT_DEBOUNCE_ITEM tarafından oluşturulan sürekli geri dönme öğelerinin başlangıç değerleri

CONT_DEBOUNCE_ITEM(_devIo, _contDebCnt, _contDebHysPer)

devIo → Hedef IO device’ı
contDebCnt → Stabilite için sayılan sayı
contDebHysPer → Kararlılık için histerezis yüzdesi (0-100, _contDebCnt = 10 ve, contDebHysPer = 80 => 8 of last 10 values must be the same for stability)

5.5.7 debin.h

Giriş IO'su için geri dönme filtresi.

DEV_IO_DEBIN_CREATE(_name, _devIo, _limit)

name → Device ismi
devIo → Geri dönüş için okunacak hedef cihaz
limit → stable değere karar vermek için count limit

5.5.8 debounce.h

Birden fazla I/O device için hazırlanmış “debounce(geri dönüş” yapan bir kütüphane.

DEV_CPX_DEBOUNCE_CREATE(_name, _period, ...)

name → Device ismi
period → DevIo dizisini kontrol etmek için her dönemin periyodu
... → DEBOUNCE_ITEM tarafından oluşturulan geri dönme öğelerinin başlangıç değerleri

DEBOUNCE_ITEM(_devIo, _debCnt)

devIo → Hedef IO device’ı
debCnt → Stabilite için sayılan değer

5.5.9 dev.h

5. bölümde anlatılan tüm cihazlar burada tanımlanmıştır.

5.5.10 enc_inc_abs.h

absolute encoder verilerini incremental encoder benzer şekilde dönüştürür.

DEV_IO_ENC_INC_ABS_CREATE(_name, _encAbs, _stepCount, _timeout)

name → Device’ın ismi
encAbs → Dönme verileri sağlayacak absolute encoder
stepCount → Absolute encoderin adım sayısı (0..n)
timeout → Verileri yenilemek için zaman aşımı

5.5.11 enc_inc_qd.h

Quadrature decoder kullanan incremental encoder device kütüphanesi.

DEV_IO_ENC_INC_QD_CREATE(_name, _sigA, _sigB, _quadDec, _quadDecBit, _noiseLimit, _time)

name → Device ismi
sigA → Encoderin A sinyali
sigB → Encoderin B sinyali
quadDec → Quadrature decoder device
quadDecBit → Quadrature decoder değeri bit sayısı
noiseLimit → Sayım bildirilmeden önce kontrol edilecek değer
time → Sinyal bütünlüğü kontrolü için süre (her kontrolde tek bir sayım oluşturulur)

5.5.12 enc_inc.h

ISR'li incremental encoder device kütüphanesi.

DEV_IO_ENC_INC_CREATE(_name, _signalA, _signalB, _isrA)

name → Device ismi
signalA → devIo için A sinyali
signalB → devIo için B sinyali
isrA → Complex device için ISR oluşturan A sinyali

5.5.13 fade.h

Her adımda I/O device içerisinde bulunan değeri belirli sürelerde değiştiren bir kütüphane.

DEV_IO_FADE_CREATE(_name, _target, _change, _duration)

name → Device ismi
target → Hedef device ismi
change → Adım başına değişim değeri
duration → Her adımın değişmesi için gerekli süre

5.5.14 filter.h

Bu device, başka bir devIo device'ından alınan verileri filtrelemek için kullanılır. devIoGet fonksiyonu her çağrıldığında, filtre hedef device'dan bir değer okur ve filtrelenmiş bir veri döndürür. Filtrelenen değer aşağıdaki gibi hesaplanır:

• Okuma verileri değerleri artan sırada sıralanır (sayımları filterSize ile belirtilir)

• Ortalama değerlerin ortancası hesaplanır (onların sayısı, MeanCount ile belirtilir)

• Hesaplanan sonucun medyanı ve eski filtrelenen değer yeniden hesaplanır

Filtre verileri int32_t olarak depolanır, böylece filtre negatif değerler döndüren devIo'larla çalışabilir. İlk okuma işleminde filtrenin kuyruğu aynı okuma değeriyle doldurulur.

Medyan değerlerin, sıralanmış bir değer dizisinin merkezindeki değerler olduğunu unutmayın.

Ortalama değerin bir değerler dizisinin aritmetik ortalaması olduğunu unutmayın.

DEV_IO_FILTER_CREATE(_name, _target, _filterSize, _meanCount)

name → Device ismi
target → Filtrelenmemiş veri sağlayacak hedef cihaz
filterSize → Filtre kuyruğunda tutulacak öğe sayısı
meanCount → Medyan değerlerinin hesaplanacağı ortalama değerlerin sayısı.

5.5.15 hc595.h

8 bitlik seri girişli, seri veya paralel latchlerler ile register değerlerini kaydıran kütüphane.

DEV_IO_HC595_CREATE(_name, _shcp, _stcp, _ds, _mrPtr, _oePtr, _initValue, _actLevels, _cascade)

name → Device ismi
shcp → Shift register clock girişi
stcp → Storage register clock girişi
ds → Serial data girişi
mrPtr → Master reset (aktif LOW)
oePtr → Output enable (aktif LOW)
initValue → Başlangıç değeri
actLevels → Tüm pinlerin aktif seviyeleri (0 ya da 1)
cascade → Kademeli cihaz sayısı (0..3)

5.5.16 hpo.h

Yüksek güçlü çıkış değişiklikleri bir ayırma ile gerçekleştirilir.

DEV_IO_HPO_TCB_CREATE(_name, _target, _timeout)

name → Device ismi
target → Hedef IO device'ı
timeout → Her değişiklik arasındaki süre

5.5.17 iocomb.h

Bir dizi IO cihazını maskelerle birleştirmek için genel I/O cihazı.

DEV_IO_IOCOMB_CREATE(_name, ...)

name → Device ismi
... → Bu IOCOMB'da birleştirilecek IOCOMB_ITEM nesneleri

IOCOMB_ITEM(_devIo, _bitCount)

devIo → Device ismi
bitCount → Bağlanacak bit sayısı

5.5.18 iodublicate.h

Yerleştirilen değeri birden fazla hedef cihaza kopyalayan ve ilkinden değer alan bir cihaz.

DEV_IO_IODUPLICATE_CREATE(_name, ...)

name → Device ismi
... → devIO verilerinin kopyalanacağına dair pointerlar

5.5.19 iolink.h

uint32_t türündeki verilere erişmek için kullanılan bir device.

DEV_IO_IOLINK_CREATE(_name, _dataPtr, _dataSize)

name → Device ismi
dataPtr → Hedef değişken pointeri
dataSize → Hedef değişken boyutu

5.5.20 iopart.h

Io bölümleme modülü, bir devIo'nun bir bölümüne erişime izin verir.

DEV_IO_IOPART_CREATE(_name, _devIo, _bitCount, _shift)

name → Device ismi
devIo → Bir parçası alınacak hedef devIo
bitCount → Alınacak bit sayısı
shift → Başlangıçtaki boş bitlerin sayısı

5.5.21 iovirtual.h

Uint32_t türündeki verilere erişmek için basit bir cihaz. Geçerli değerine ulaşmak için "_name" ve "Value"yu birleştirir.

DEV_IO_IOVIRTUAL_CREATE(_name)

name → Device ismi

5.5.22 lerp.h

Doğrusal İnterpolasyon devIo, hedef devIo'dan X değerlerini alır ve bir tabloda sağlanan değerlerden gerekli hesaplamaları yapar.

DEV_IO_LERP_CREATE(_name, _devX, _table)

name → Device ismi
devX → devIo X değeri olarak kullanılacak
table → Grafikteki noktaları temsil eden tsLerpPoint dizisilut.h

Bir girdiden hedef çıktıya eşleme yapmak için Arama Tablosu.

```
typedef struct
{
   int32_t x; ///< X value of point
   int32_t y; ///< Y value of point
} tsLerpPoint;
```

DEV_IO_LUT_CREATE(_name, _target, _lut)

name → Device ismi
target → LUT(Look Up Table) değerlerinin çıktısını almak için devIo'yu hedefleyin
lut → Arama tablosu olarak kullanılacak uint32_t değerleri dizisi

5.5.23 m41t100

ST M41T00 Seri Gerçek Zamanlı Saat

DEV_CPX_M41T00_CORE_CREATE(_name, _i2c, _target)

name → Device ismi
i2c → I2C communication device'ı
target → Communication kanalı bilgisi

5.5.24 matrixout.h

1 ms'lik periyotlarla sütunlardan birine enerji verilir ve ilgili veriler satırlara aktarılır. Sütunların active-low olduğu varsayılmıştır. Sütunlara bağlı ve başka bir gpio ile ortak olan pinlere sahip bir absolute encoder yerleştirmek için bir çözüm eklenmiştir.

DEV_CPX_MATRIXOUT_CREATE(_name, _rowPort, _colPort, _colCount, _deadTime, _encCommon)

name → Device ismi
rowPort → Hedef devIO ham değeri
colPort → Hedef devIO sütun değeri
colCount → Toplam sütun sayısı
deadTime → Her sütun arasındaki ölü bölgenin döngü sayısı
encCommon → Absolute encoder ortak değeri

5.5.25 mean.h

Hedef devIo'nun ortalama değerini hesaplar.

DEV_IO_MEAN_CREATE(_name, _target, _count, _disExt)

name → Device ismi
target → Değerleri alacağı devIO
count → Ortalama değerin hesaplanması için toplanacak değer sayısı (en az 3 ve 255'ten az olmalıdır)
disExt → TRUE ya da FALSE

5.5.26 mem_hamming.h

Herhangi bir veriyi [8,4]Hamming yöntemiyle kodlayan/kod çözen bir device.

DEV_MEM_MEM_HAMMING_CREATE(_name, _mem)

name → Device ismi
mem → Hamming kodlu verileri tutacak hedef hafıza cihazı

5.5.27 memvirtual.h

Ram bölgesini kullanan bir Memory device'ı.

DEV_MEM_MEMVIRTUAL_CREATE(_name, _size)

name → Device ismi
size → Kullanılacak alanın boyutu(uint16_t'yi geçmemelidir)

5.5.28 morse.h

Morse kodu yaratmak için kullanılan bir kütüphane.

DEV_COM_MORSE_CREATE(_name, _output, _onValue, _offValue, _unit, _bufferSize)

name → Device ismi
output → Output devIo
onValue → Çıkış devIO için ON değeri
offValue → Çıkış devIo'su için OFF değeri
unit → Milisaniye cinsinden 1 birim sembol zamanlaması
bufferSize → İletim için arabellek boyutu

5.5.29 pattern.h

Zamanlamalı bir değer dizisi olan bir yol çıktısı üretir.

DEV_IO_PATTERN_CREATE(_name, _target, _offValue)

name → Device ismi
target → PATTERN çıktısı olarak kullanılacak hedef devIo
offValue → Hedef device için off değeri

5.5.30 pulse.h

Belirlenen bir zaman boyunca output çıktısı üretir.

DEV_IO_PULSE_CREATE(_name, _target, _on, _off)

name → Device ismi
target → Hedef devIO ismi
on → Aktif pulse değeri
off → Deaktif pulse değeri

5.5.31 sht3x.h

Bu cihaz, I2C bus üzerinden sht3x sensöründen sıcaklık ve nem ölçümlerini okumak için kullanılır. Sıcaklık ve nemi almak için 1 çekirdek cihaz ve diğer iki cihazdan oluşur..

DEV_IO_SHT3X_CORE_CREATE(_name, _i2c, _slaveAddress, _intervalMs)

name → Device ismi
slaveAddress → SHT3X slave adresi, 0x44 or 0x45 olabilir
intervalMs → ms cinsinden iki ardışık ölçüm arasındaki aralık.

5.5.32 soft_pwm.h

Periyodik bir kesme zamanlayıcısı kullanarak normal bir GPIO pini üzerinde PWM sinyalleri üretmek için kullanılır. Bir PWM çekirdek cihazının birçok kanalı olabilir, her kanalın farklı bir görev döngüsü olabilir ancak çekirdek frekansının kendisi olan aynı frekansı paylaşırlar. Çekirdek frekansı ve bir kanalın frekansı, "put" fonksiyonuyla değiştirilebilir. Çekirdek bir Periyodik Kesinti Zamanlayıcısı (Periodic Interrupt Timer) cihazı gerektirir ve bunu periyodik bir işlevi çağırmak için kullanır. Bu fonksiyonda kanalların dijital seviyeleri, kanalın ayarlanan görev döngülerine ve dönemdeki mevcut konumuna göre değiştirilir. Üretilen PWM sinyallerinin çözünürlüğü ayarlanabilir. Bu, periyodik işlevi daha az kez çağırarak MCU üzerindeki ek yükü azaltmak için kullanışlıdır.

Örneğin, modül sabit %50 görev döngüsüne sahip ancak farklı frekanslarda PWM sinyalleri üretmek için kullanılıyorsa çözünürlük 50'ye ayarlanmalıdır çünkü periyodik fonksiyonu %1, %2, %3, 4'te çağırmanın bir anlamı yoktur. Daha sonra periyodik fonksiyon yalnızca %0 ve %50 konumlarında çağrılacaktır. Aynı şekilde kanalların görev döngüsü değerleri sadece %10'un katları ise (%0, %10, %20, ,,, %100) çözünürlük %10'a ayarlanmalıdır. Çekirdek frekansı güncellendiğinde mevcut çalışma süresinin sonunda yeni frekans uygulanacaktır.

DEV_IO_SOFT_PWM_CORE_CREATE(_name, _pitDevice, _defaultFreqHz, _resolution, ...)

name → Device ismi
pitDevice → Periyodik fonksiyon çağrıları oluşturmak için kullanılacak PIT cihazı
defaultFreqHz → çekirdeğin varsayılan frekansı
resolution → yüzde değeri [1-100]. üretilen PWM sinyalleri görevinin çözünürlüğünü belirtir
… → SOFT_PWM_CHNL() makrosu ile başlatılan PWM Kanallarının listesi

```
#define SOFT_PWM_CHNL(_defaultDuty)
```

defaultDuty → duty cycle ayarı

5.5.33 stepper_motor.h

Bu cihaz 4 GPIO pini kullanarak step (adım) motorları sürmek için kullanılır. Step motor üç yöntemle çalıştırılabilir: Tam adım, Yarım adım ve Dalga sürüşü. Cihaz birçok etkinlikte kalibrasyon yapacak şekilde yapılandırılabilir:

• 1- Cihaz başlatılırken,

• 2- Belirli saatler geçtikten sonra,

• 3- Belirli hareket sayısından sonra.

Sürüş yöntemi ve kalibrasyon ayarları, devIoInit func ile konfigürasyonlar olarak cihaza aktarılır. Cihaz, motorun yönünü değiştirmeden veya motora yeniden enerji vermeden önce kullanılan bir blokaj süresi uygular. Motorun başlatıldıktan sonraki konumu 0'dır. DevIoPut fonksiyonuyla verilen herhangi bir konum değeri, motorun saat yönde çalışmasına neden olur. Konum değeri, başlangıç konumundan (0) uzaktaki adım sayısını temsil eder

Step motorların çalıştırılması hakkında FMI'a bakınız:
https://www.ti.com/lit/an/slva767a/slva767a.pdf

5.5.34 timeout.h

PUT geri sayım değerini değiştirmek için kullanılır, INIT geri sayımı başlatmak için kullanılır, DEINIT duraklatmak için kullanılır, mevcut değer GET tarafından okunabilir.

DEV_IO_TIMEOUT_CREATE(_name)

name → Device ismi

6. LIBS

- Bit manipülasyonları için kullanılan fonksiyonlar
- Cyclic Redundancy Check (CRC-8-MAXIM and CRC-16-CCITT) fonksiyonları
- Debug kayıt makroları
- Double Ended Queue yapıları ve fonksiyonları
- Hamming Code fonksiyonları
- Doubly Linked List yapıları ve fonksyionları
- Otomatik Sürüm oluşturmaya yönelik Makrolar

6.1 bits.h

Temel bit işlemleri için gerekli makroları ve işlevleri içerir.

- **#define** BIT(_idx) (1ul << (_idx)) → Dizin numarasıyla bitin dönüş maskesi.
- **#define** BIT_LSB(_data) find_lsb(_data) → LSB dizin değerini döndürme makrosu
- **#define** BIT_MSB(_data) BIN_ENCODE(_data) → Verinin MSB'sini dizin numarasına kodlama makrosu
- **#define** BIN_ENCODE(_data) find_msb(_data) → Verilerin MSB'sini dizin numarasına kodlama makrosu.
- **#define** BIN_DECODE(_value) BIT(_value) → Değeri bit temsiline göre çözen makro.
- **#define** BIT_COUNT(_byte) bit_count(_byte) → Bir baytın bitlerini sayan makro.

6.2 crc.h

8 ve 16 bitlik CRC algoritması içeririr. 8 bit CRC hesaplamasında **MAXIM** algoritması kullanılır**.**

16 bit CRC hesaplamasında **Comite Consultatif International de Telegraphique et Telephonique(CCITT)** kullanılır**.**

- **uint8_t** crc8MaximInit(void) → CRC'nin başlangıç değerini döndürür
- **uint8_t** crc8MaximAddData(**uint8**_t currentCrc, **uint8_t** data → CRC'nin yeni değerini döndürür
- **uint8_t** crc8MaximFinish(**uint8**_t currentCrc) → CRC'nin final değerini döndürür
- **uint16_t** crc16CcittInit(**void**) → CRC'nin başlangıç değerini döndürür
- **uint16_t** crc16CcittAddData(**uint16**_t currentCrc, **uint8_t** data) → CRC'nin yeni değerini döndürür
- **uint16_t** crc16CcittFinish(**uint16**_t currentCrc) → CRC'nin final değerini döndürür

6.3 debug.h

Derleme sırasında açılıp kapatılan hata ayıklama kayıt altyapısını içerir.

«rcos.h» dosyasında RCOS_DEBUG tanımı ENABLED olduğunda tüm altyapı geçerli hale gelir. Her dosyada tanımlanan istenilen sayıda farklı RCOS_DEBUG_PORT arasından ENABLE olarak tanımlananlar derleme sırasında koda eklenir ve çalışma zamanı sırasında bir hata ayıklama kaydı üretir

Örnek:

```
#define RCOS_DEBUG_FILE_NAME "adc"
#define RCOS_DEBUG_PORT_INIT ENABLE
#define RCOS_DEBUG_PORT_LOW_LEVEL DISABLE
...
RCOS_DEBUG_PRINT(RCOS_DEBUG_PORT_INIT, "module initialized.");
...
RCOS_DEBUG_PRINT(RCOS_DEBUG_PORT_LOW_LEVEL, "read value: %d",
adcValue);
```

6.4 deque.h

Doubly Ended Queue yapılarını ve bu yapıları kullanan temel queue fonksiyonlarını içerir. TsDeque tipindeki değişkenler uygun makrolar kullanılarak FIFO ve/veya LIFO olarak kullanılabilir.

- dequeAddBack(**tsDeque** *deq, **uint16_t** size) → boşluk varsa parametrede iletilen öğeyi Deque'nin sonuna ekler
- dequeAddFront(**tsDeque** *deq, **uint16_t** size) → boşluk varsa parametrede iletilen öğeyi Deque'nin başına ekler
- dequeRemBack(**tsDeque** *deq, **uint16_t** size) → Parametrede bulunan ögeyi Deque'nin sonundan alır
- dequeRemFront(**tsDeque** *deq, **uint16_t** size) → Parametrede bulunan ögeyi Deque'nin başından alır
- dequeFlush(**tsDeque** *deq) → buffer'da bulunan tüm verileri kaldırır
- dequeClear(**tsDeque** *deq) → bufferda bulunan tüm verileri kaldırır ve "0" yapar
- dequePopBack(**tsDeque** *deq, **void** *destination, **uint16_t** size) → Mümkün olduğu kadar okur ve okunan miktarı döndürür
- dequePopFront(**tsDeque** *deq, **void** *destination, **uint16_t** size) → Mümkün olduğu kadar okur ve okunan miktarı döndürür
- dequePushBack(**tsDeque** *deq, **const void** *source, **uint16_t** size) → Bir dizi baytı ara belleğe yerleştirin ve gerekirse üzerine yazın
- dequePushFront(**tsDeque** *deq, **const void** *source, **uint16_t** size) → Bir dizi baytı ara belleğe yerleştirin ve gerekirse üzerine yazın
- dequePeek(**tsDeque** *deq, **uint16_t** index, **void** *destination, **uint16_t** size) → Bufferın en öndeki değerini döndürür

6.5 hamming.h

Hamming kodu hata tespiti ve düzeltme algoritmaları için dönüştürme fonksiyonlarını ve gerekli tanımları içerir.

- hammingDistance(**uint8_t** data1, **uint8_t** data2)
- hammingEncodeByte(**uint8_t** data);
- hammingDecodeByte(**uint16_t** encoded);
- hammingEncodeArray(**uint16_t** *destination, **const uint8_t** *source, **uint32_t** count);
- hammingDecodeArray(**uint8_t** *destination, **const uint16_t** *source, **uint32_t** count)
- hammingEncode(**uint8_t** data)
- hammingDecode(**uint8_t** data)

6.6 hash.h

6.7 json.h

6.8 libs.h

Genel amaçlarla kullanılan birçok sabit bilgi ve makroyu içerir. Bunlardan bazıları çekirdekte zaten kullanılıyor.

Örnek:

#define ENABLE, **#define** DISABLE, **#define** ARRAY_SIZE
#define SIZE_OF_MEMBER, **#define** MAX, **#define** MIN

6.9 list_dl.h, list_sl.h

List_dl.h ile list_sl.h arasındaki fark, list_dl'nin Çift Bağlantılı Liste yapılarını içermesi, list_sl'nin ise Tek Bağlantılı yapıları içermesidir. Her ikisinin de bu yapılarda kullanılan temel işlevleri vardır. tsListItem ve tsList türleri işlevlerin kullanımına olanak sağlar.

- lslInsertAfter(tsLsl *list, void *item, void *newItem)
- lslInsertBefore(tsLsl *list, void *item, void *newItem)
- lslInsertHead(tsLsl *list, void *newItem)
- lslInsertTail(tsLsl *list, void *newItem)
- lslDelete(tsLsl *list, void *item)
- *lslPopHead(tsLsl *list)
- *lslPopTail(tsLsl *list)
- *lslTailFrom(void *item)
- lslRecount(tsLsl *list)
- *lslItemAtIdx(tsLsl *list, uint32_t idx)
- lslIdxOfItem(tsLsl *list, void *item)

6.10 rassert.h

6.11 rcosVersion.h

C programlama dili tarafından önceden tanımlanmış __DATE__ ve __TIME__ makrolarının işlenmesine yardımcı olmak için tanımlanmış basit makroları içerir..

#define BUILD_YEAR, **#define** BUILD_MONTH, **#define** BUILD_DAY

#define BUILD_HOUR, **#define** BUILD_MIN, **#define** BUILD_SEC,

6.12 ticket.h

Basit bir kilit mekanizması sağlar. Bir kaynağın birden fazla uygulama tarafından sırayla kullanılmasını sağlamak için geliştirilmiştir. Her kaynak bir Bilet Satış Makinesi (TVM) içerir ve bu kaynağa erişmek isteyen herkesin bir tTicket'i vardır. Uygulama, kullanım sırasını belirlemek için öncelikle tek kullanımlık bilet satın alıyor ve sırasını bekliyor.

Sırası geldiğinde süreci başlatabilir ve gerekirse sürecin tamamlanmasını bekleyebilir. Her işlem sonunda aktif Biletin silinmesi işlemi ilgili kaynak tarafından arka planda gerçekleştirilmelidir..

#define TICKET_GET(_tvm)

#define TICKET_IS_UP(_tvm, _ticket)

#define TICKET_IS_DONE(_tvm, _ticket)

#define TICKET_REMOVE(_tvm)

6.13 utf.h

7. ZAMANLAYICILAR

2 çeşit zamanlayıcı bulunur.

- TIMER_CALLBACK_FUNC
- TIMER_EVENT

7.1 TIMER_CALLBACK_FUNC

Bu timer kendisini queue'nun başına ekler. Kullanılması **önerilmez**.

```
TIMER_CALLBACK_FUNC(Timer_Func)  //Timer fonksiyonu
{
   //Yapılması gereken kodu buraya yazın
   return 1000;
}
TIMER_CALLBACK_CREATE(Timer_Func_name, Timer_Func, 0) // Create makrosu
void rcosMainLoop(void)
{
   platformInit();
   coreInit();
   timerCallBackStart(&Timer_Func_name, 500); //Timer başlangıcı.
   coreRun();
}
```

Bu örnekte coreRun(); execute olduktan 500 milisaniye sonra Timer_Func execute olacak ve içerisindeki kodu gerçekleştirecek Daha sonrasında her 1000 milisaniyede bir kendisini queue'nun en başına ekleyecek.

7.2 TIMER_EVENT

tsTimerEvent timer

Yukarıdaki fonksiyon başlık dosyasının içindeki params yapısının içine yazılmalıdır.

.timer = TIMER_EVENT_INIT(_enum, _enum, event)

Yukarıdaki fonksiyonun makro parametre tanımlarının içine yazılması gerekir.

timerEventStart(&(params->timer), **uint32_t** duration)

Zamanlayıcıyı başlatmak için yukarıdaki fonksiyonun PROCESS_INIT_PROTO(_name) içine yazılması gerekir.

timerEventStart(&(params->timer), **uint32_t** duration)

Yukarıdaki fonksiyonun kendisini event queue'nun sonuna ekleyebilmesi için event'lerin içine yazılması gerekir.