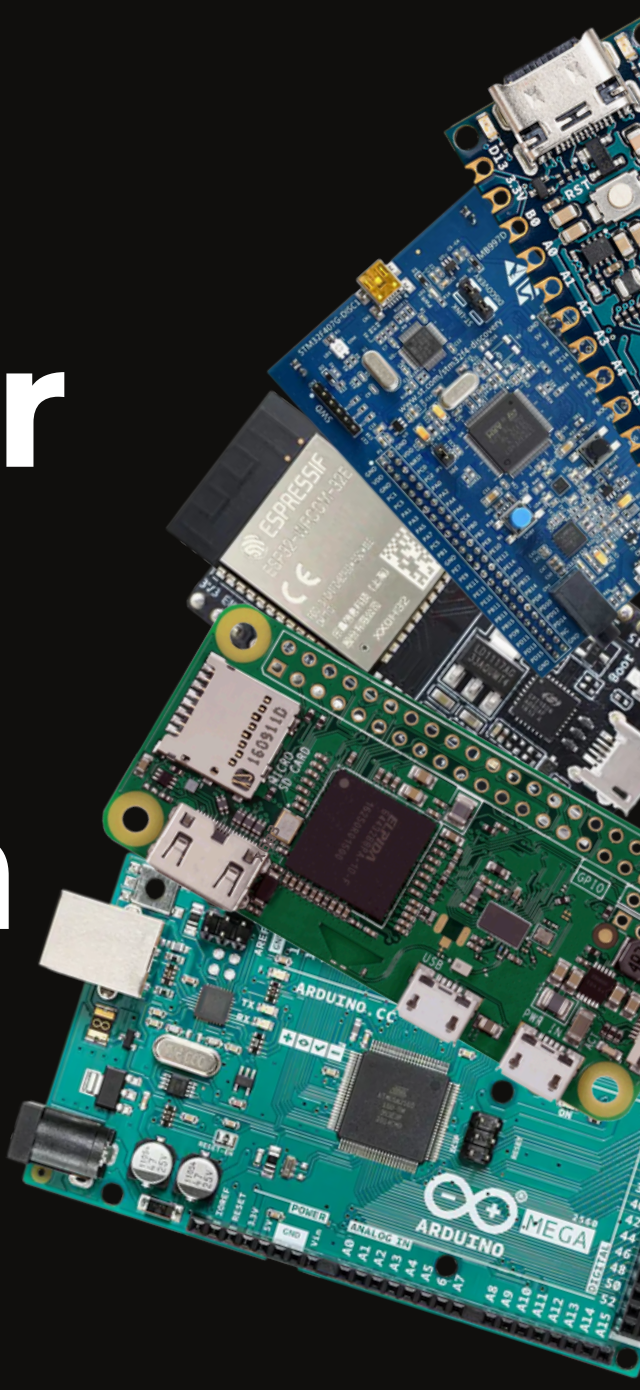
Altın Kariyer Akademi

EĞITMEN
Can Koç

Gömülü Sistemler Ve Robotik Kodlama



Adres

Altay Mahallesi
Atayıldız Platinum
Eryaman/Ankara

Telefon

+90 (312) 419 51 37

E-Mail

altinkariyerakademi@
gmail.com

EEPROM Başlangıç

Bu hafta ise biraz donanım tarafına yöneliyoruz ve kalıcı bellek konusuna dalıyoruz. Özellikle, elektrik kesilse bile veriyi tutabilen küçük ama önemli bir bellek türünü konuşacağız: **EEPROM'lar.**
Geçen hafta FreeRTOS'un task yapısını, queue'larını ve zamanlayıcısını konuştuk.
Şimdi bu sistemin bellekle nasıl konuştuğuna giriyoruz. RAM her reset'te her şeyi unutur; EEPROM unutmaz.
Ama unutmadığı her baytın bir bedeli vardır: yavaş yazma, sınırlı ömür, hataya açık veri.

Gerçek bir sistemde bu dengeyi sen kurarsın.
Yazacağın kod, sadece veri saklamaz — cihazın hafızasını yaşatır ya da bitirir.

EEPROM (Electrically Erasable Programmable Read-Only Memory), Türkçesiyle Elektrikle Silinebilir Programlanabilir Salt Okunur Bellek, gömülü sistemlerde sıkça kullanılan kalıcı bir hafıza türüdür.
Adında "salt okunur" yazmasına rağmen, elektronik olarak tekrar tekrar yazılabilir ve silinebilir. Fişi çekseniz bile, içindeki veriyi saklamaya devam eder.
Bu özelliğiyle, güç kesilse de cihazınızın kritik verileri korunur.

EEPROM Başlangıç

Peki Ya Neden EEPROM?

Çünkü sistem bir gün kapanacak. Güç kesilecek, batarya bitecek, mikrodenetleyici yeniden doğacak.
O anda en son durumu RAM'de tuttuysan geçmişin gider; EEPROM'da tuttuysan sistem kaldığı yerden devam eder.

Ama işin sırrı şu: EEPROM'a her şeyi yazamazsın.
Bu bellek hızlı değil, dayanıklı da değil. "Yazmak" orada fiziksel bir yıpranmadır.

FreeRTOS ile EEPROM Nasıl Yaşar?

RTOS ortamında her task bir işi sırayla yürütür.
Ama EEPROM beklemez. Bir task doğrudan **ee_write()** çağırırsa, 3–5 ms boyunca bus meşgul olur.
O sürede başka task'lar bloklanır.
Bunun çözümü, EEPROM erişimini bir "hafıza task"ına devretmektir.

```
void vEepromTask(void *pvParams) {
   eeprom_msg_t msg;
   for (;;) {
      if (xQueueReceive(xEepromQueue, &msg,
portMAX_DELAY)) {
         ee_write_page(msg.addr, msg.data, msg.len);
         xTaskNotify(msg.srcTask, EEPROM_OK,
eSetValueWithOverwrite);
      }
   }
```

Nedir Bu EEPROM?

void vEepromTask(void *pvParams) {

- FreeRTOS task fonksiyon imzası.
- pvParams ile bu task'a dışarıdan başlangıç parametresi verilebilir (mesela I²C handle'ı, EEPROM boyutu, sayfa uzunluğu, CRC seçimi).
- Kullanmasan da imza böyle olmalı.

eeprom_msg_t msg;

- Kuyruktan gelecek mesajı tutacağın geçici çalışma alanı.
- Tipi muhtemelen şöyle bir şey olmalı:

```
typedef struct {
    TaskHandle_t srcTask;   // Bittiğini kime haber vereceğiz
    uint32_t     addr;      // EEPROM hedef adres
    uint8_t*     data;      // Yazılacak buffer
    size_t       len;       // Kaç byte
} eeprom_msg_t;
```

for (;;) {

- Sonsuz döngü. RTOS dünyasında task'lar bu şekilde yaşar.
- CPU'yu yakmıyor çünkü içeride bloklayan bir çağrı var (kuyruk bekleme).

Nedir Bu EEPROM?

if (xQueueReceive(xEepromQueue, &msg, portMAX_DELAY)) {

- xEepromQueue kuyruğundan mesaj gelene kadar blokla.
- portMAX_DELAY demek “sabaha kadar bekle.” Güzel, çünkü bu task boşuna CPU yemiyor.
- Dönen değer true ise bir mesaj var. Mesaj yoksa zaten task uyuyor.

ee_write_page(msg.addr, msg.data, msg.len);

- I²C/SPI üzerinden EEPROM’a sayfa yazımı yapan sürücü fonksiyonu.

Burada kritik noktalar:

a. Sayfa hizası: EEPROM’larda tipik sayfa boyu 16, 32, 64 byte olur. addr sayfa sınırını aşarsa cihaz wrap yapabilir. Sürücü bu işi parçalara bölüp güvenli yazmalı.
b. Write-cycle zamanı: 3–5 ms tipik; sürücü ACK polling ile tamamlanmayı beklemeli. Yazma bitmeden ikinci komut verme.
c. Timeout ve retry: I²C hatası, bus çakışması, kablo gürültüsü… Hepsine karşı en az 2–3 retry iyi fikirdir.
d. Mutex: ee_write_page içinde I²C için bir mutex alınıp bırakılmalı. Başka task aynı anda I²C’ye dalmasın.

Nedir Bu EEPROM?

x**TaskNotify(msg.srcTask, EEPROM_OK, eSetValueWithOverwrite);**

- Kaynağa “iş bitti” sinyali.
- msg.srcTask: Hangi task senden sonuç bekliyor.
- EEPROM_OK: Özet durum kodu (başarısızlık varsa EEPROM_ERR gibi döndürürüz).
- eSetValueWithOverwrite: Önceki bildirimi ez. Kaynak task ulTaskNotifyTake() veya xTaskNotifyWait() ile bekliyor olmalı.

- **Döngü kapanır, task kuyruğa geri döner. Sıradaki iş için bekler.**

aşağıdaki örnek , gerçek zamanlı bir sistemde EEPROM erişimini yöneten bir özel task mimarisini gösterir.
Amaç, EEPROM’a yapılan yazma isteklerini sıraya alıp güvenli, hataya dayanıklı ve senkronize biçimde yürütmektir.

Normalde her task doğrudan ee_write() çağırırsa, I²C hattı kilitlenir, sistem deterministik davranışını kaybeder.
Bu kod, o karmaşayı ortadan kaldırmak için yazılmıştır. Her yazma isteği bir kuyruk (queue) üzerinden bu görev yöneticisine gelir. Görev, isteği işler, güvenli şekilde EEPROM’a yazar, ardından ilgili task’a “işlem tamamlandı” bildirimi gönderir.aşağıya doğru 3 sayfa olacak şekilde hazırladım dikkatlice incelemenizi isterim.

Nedir Bu EEPROM?

```
typedef enum {
    EEPROM_OK = 0,
    EEPROM_ERR_ARG,
    EEPROM_ERR_ALIGN,
    EEPROM_ERR_BUS,
    EEPROM_ERR_TIMEOUT,
} eeprom_status_t;

typedef struct {
    TaskHandle_t srcTask;
    uint32_t     addr;
    const uint8_t* data;
    size_t       len;
    uint16_t     crc;      // opsiyonel: veri bütünlüğü için
} eeprom_msg_t;

static const size_t EE_PAGE = 64;
extern QueueHandle_t xEepromQueue;
extern SemaphoreHandle_t xI2cMutex;

static eeprom_status_t ee_write_pages_checked(uint32_t addr,
const uint8_t* p, size_t n);

void vEepromTask(void *pvParams) {
    eeprom_msg_t msg;

    for (;;) {
        if (xQueueReceive(xEepromQueue, &msg,
portMAX_DELAY))
```

```
{
 eeprom_status_t st = EEPROM_OK;

 // 1) Argüman kontrolü
 if (msg.data == NULL || msg.len == 0) {
 st = EEPROM_ERR_ARG;
 }

 // 2) I2C mutex (bus paylaşımı)
 if (st == EEPROM_OK) {
 if (xSemaphoreTake(xI2cMutex, pdMS_TO_TICKS(50)) !=
pdTRUE) {
 st = EEPROM_ERR_BUS;
 }
 }

 // 3) Sayfa güvenli yazım + retry
 if (st == EEPROM_OK) {
 st = ee_write_pages_checked(msg.addr, msg.data, msg.len);
 xSemaphoreGive(xI2cMutex);
 }

 // 4) Kaynağa bildirim (NULL olabilir)
 if (msg.srcTask) {
 xTaskNotify(msg.srcTask, st, eSetValueWithOverwrite);
 }
 }
 }
}

static eeprom_status_t ee_write_pages_checked(uint32_t addr,
const uint8_t* p, size_t n) {
 const int MAX_RETRY = 3;
```

```
while (n > 0) {
 // Sayfa sınırı
 uint32_t page_off = addr % EE_PAGE;
 uint32_t chunk = EE_PAGE - page_off;
 if (chunk > n) chunk = n;

 // Retry döngüsü
 for (int i = 0; i < MAX_RETRY; i++) {
 if (ee_write_page(addr, p, chunk) == 0) { // sürücü 0: OK
 if (ee_wait_write_complete(pdMS_TO_TICKS(10)) == 0) { // ACK
polling
 break; // başarılı
 }
 }
 if (i == MAX_RETRY - 1) return EEPROM_ERR_TIMEOUT;
 vTaskDelay(pdMS_TO_TICKS(1)); // bir nefes alalım
 }

 addr += chunk;
 p += chunk;
 n -= chunk;
 }
 return EEPROM_OK;
}
```

Nedir Bu EEPROM?

Bu kod FreeRTOS altında EEPROM'a yapılan tüm yazma işlemlerini kontrol eden bir görev yapısı oluşturuyor. Parça parça gidelim.

İlk satırda **typedef enum { ... } eeprom_status_t;** var.
Bu kısım, her EEPROM işleminin sonunda dönecek sonucu temsil ediyor. Yani "işlem başarılı mı, hatalı mı, neden hatalı?" sorularına yanıt veriyor.
EEPROM_OK demek her şey yolunda.
EEPROM_ERR_ARG veri geçersiz — mesela NULL pointer ya da uzunluk sıfır gönderilmiş.
EEPROM_ERR_ALIGN sayfa hizalaması bozuk — EEPROM'un 64 byte'lık sınırlarını aşarsan veri sarmalanır ve yanlış yere yazılır.
EEPROM_ERR_BUS I^2C hattına ulaşılamamış, başka bir task kullanıyor ya da bus kilitli.
EEPROM_ERR_TIMEOUT cihaz belirli süre içinde yanıt vermemiş.
Bu enum'lar kodun dili gibi; task'lar arası anlaşma burada başlıyor.

Ardından **typedef struct { ... } eeprom_msg_t;** geliyor.
Bu yapı, kuyruktan gelen mesajın paketidir. Yani başka bir task EEPROM'a veri yazmak istediğinde bu struct'ı doldurur ve kuyruğa gönderir.
srcTask kimden geldiğini tutar, işlem bitince bu task'a haber verilecek.
addr yazılacak başlangıç adresi.
data yazılacak verinin adresi.
len verinin uzunluğu.
crc veri bütünlüğü için opsiyonel bir alan; istersen CRC kontrolü eklersin.
Bu struct bir çeşit kargo kolisi: nereye, neyi, kimden aldık hepsi burada.

Nedir Bu EEPROM?

Bu kod FreeRTOS altında EEPROM'a yapılan tüm yazma işlemlerini kontrol eden bir görev yapısı oluşturuyor. Parça parça gidelim.

İlk satırda **typedef enum { ... } eeprom_status_t;** var.
Bu kısım, her EEPROM işleminin sonunda dönecek sonucu temsil ediyor. Yani "işlem başarılı mı, hatalı mı, neden hatalı?" sorularına yanıt veriyor.
EEPROM_OK demek her şey yolunda.
EEPROM_ERR_ARG veri geçersiz — mesela NULL pointer ya da uzunluk sıfır gönderilmiş.
EEPROM_ERR_ALIGN sayfa hizalaması bozuk — EEPROM'un 64 byte'lık sınırlarını aşarsan veri sarmalanır ve yanlış yere yazılır.
EEPROM_ERR_BUS I^2C hattına ulaşılamamış, başka bir task kullanıyor ya da bus kilitli.
EEPROM_ERR_TIMEOUT cihaz belirli süre içinde yanıt vermemiş.
Bu enum'lar kodun dili gibi; task'lar arası anlaşma burada başlıyor.

Ardından **typedef struct { ... } eeprom_msg_t;** geliyor.
Bu yapı, kuyruktan gelen mesajın paketidir. Yani başka bir task EEPROM'a veri yazmak istediğinde bu struct'ı doldurur ve kuyruğa gönderir.
srcTask kimden geldiğini tutar, işlem bitince bu task'a haber verilecek.
addr yazılacak başlangıç adresi.
data yazılacak verinin adresi.
len verinin uzunluğu.
crc veri bütünlüğü için opsiyonel bir alan; istersen CRC kontrolü eklersin.
Bu struct bir çeşit kargo kolisi: nereye, neyi, kimden aldık hepsi burada.

EEPROM

static const size_t EE_PAGE = 64; kısmı EEPROM'un sayfa sınırını belirtiyor.
64 byte'tan fazla tek seferde yazarsan EEPROM başa döner, yani veri sarmalanır ve kayar.
Bu sabit, **ee_write_pages_checked()** fonksiyonuna ne kadar yazabileceğini öğretir.

extern QueueHandle_t xEepromQueue;
Bu, sistemdeki EEPROM görevine veri göndermek için kullanılan kuyruktur.
Her task yazmak istediğinde bu kuyruğa mesaj yollar, EEPROM task'ı sırayla alır.
Kuyruk olmazsa aynı anda iki task EEPROM'a yazar ve bus karışır.

extern SemaphoreHandle_t xI2cMutex;
I²C hattını koruyan kilittir.
EEPROM, sensör, RTC hepsi aynı hatta olabilir.
Mutex alınmadan bus'a çıkmak, yolda karşıdan gelen arabaya dalmak gibidir.
Kilit alınır, işlem yapılır, sonra bırakılır.

void vEepromTask(void *pvParams) — bu asıl görev fonksiyonu.
Sonsuza kadar çalışır ama CPU yemez çünkü kuyruktan mesaj bekler.
xQueueReceive(xEepromQueue, &msg, portMAX_DELAY) satırı yeni mesaj gelene kadar task'ı bekletir.
Bir mesaj geldiğinde onu msg yapısına kopyalar.

EEPROM

İlk iş olarak parametreleri kontrol eder.
msg.data == NULL || msg.len == 0 gibi bir durum varsa yazmayı iptal eder ve hata döner.
Bu kontrol olmazsa, EEPROM sürücüsüne çöp pointer girer, sistem çöker.

Sonra **xSemaphoreTake(xI2cMutex, pdMS_TO_TICKS(50))** satırıyla I²C hattını ister.
Eğer 50 ms boyunca kilit alınamazsa bus hatası döner.
Bu, sistemin kilitlenmesini önler.

Kilit alınmışsa **ee_write_pages_checked()** çağrılır.
Bu fonksiyon veriyi EEPROM'a yazarken sayfa hizasını gözetir, her yazma sonrası bekler, ACK kontrolü yapar, başarısız olursa birkaç kez yeniden dener.
Yani bu satır işi sadece "yazmak" değil, "doğru yazıldığından emin olmak" için yapar.

Yazma işlemi bittikten sonra **xSemaphoreGive(xI2cMutex)** ile hattı serbest bırakır.
Sonra **xTaskNotify(msg.srcTask, st, eSetValueWithOverwrite)** ile kaynağa "işin bitti" sinyali gönderir.
Böylece veri gönderen task, ulTaskNotifyTake() gibi bir fonksiyonla sonucu alabilir.

EEPROM

ee_write_pages_checked() fonksiyonu da kendi içinde küçük ama kritik bir döngüye sahip.
Adres sayfa sınırına denk gelmiyorsa önce kalan boşluk kadar yazar, sonra sıradaki sayfaya geçer.
Her sayfa yazımında 3 kez retry yapılır, çünkü bazen EEPROM ACK vermez.
Her denemeden sonra küçük bir gecikme (1 ms) bırakılır, böylece bus biraz nefes alır.
Tüm sayfalar sorunsuz yazıldıysa EEPROM_OK döner, aksi halde **EEPROM_ERR_TIMEOUT.**

Bu yapının amacı sadece “EEPROM’a veri yazmak” değil, sistemi sağlamlaştırmaktır.
Kuyruk sistemi task’ların sırayla çalışmasını sağlar, mutex bus çatışmasını önler, retry ve timeout sistemin dayanıklılığını artırır, notify ise deterministik yanıt sağlar

Şimdi sizleri duyar gibiyim Nerede kullanacağız? gibi sorularınız için gündelik hayattan örnekler ile ilerleyeceğim.

1) Ayar Saklama (Config)
Her cihazın bir “karakteri” olur: parlaklık, ses, dil… Kapatıp açınca “en son kaldığı yerden” devam etsin istiyorsun değil mi? İşte bu ayarlar, uçucu olmayan tekillik: EEPROM’da. Kullandığın alan genellikle birkaç byte. SD kartla, harici flashla uğraşmaya gerek yok. Reset atınca “kullanıcı ayarı” geri gelsin, konu kapanır.

EEPROM

2) Kalibrasyon ve Fabrika Parametreleri
Cihazı üretirken sensör ofsetini, ADC referansını kalibre ettin mi? O değeri EEPROM'a gömersin. Her başlatmada kod, "acaba fabrika ayarı var mı?" diye kontrol eder, bulursa onu çeker, yoksa default atar. Kullanıcıya özel kalibrasyonları da bu şekilde saklayıp, her cihazı tek tek karakteristik kılabilirsin.

3) Sayaçlar & Çalışma Zamanı
Örnek: bir cihaz kaç saat çalıştı, kaç defa açılıp kapandı? Bunu RAM'de tutsan güç gidince uçar. EEPROM'a her 100 artışta bir yazarsın, cihaz ömrü boyunca bakım periyodunu izlersin. Otomotivde, medikalde, sanayide – hepsinde ömür takibi böyle yapılır.

4) Olay Logları (Küçük Buffer)
Bir cihaz anlık hata verirse veya belli bir olayı kaydetmek gerekirse, son 10-20 hata kodunu EEPROM'da küçük bir ring buffer'da tutarsın. Güç gitti, geri geldi – olay geçmişi hala orada. Sahada inceleme, servis işlemi, hepsi için kritik.

5) Kimlik / Protokol / Seri No
CAN-Bus adresi, üretim seri numarası, cihaz MAC adresi... Bunlar "hardcoded" da olabilir ama programatik olarak ayarlanıp saklanacaksa adres EEPROM'da olur. Cihaz açılır açılmaz "ben kimim" diye bakar, EEPROM'dan kimlik bilgisini okur.

EEPROM

6) Kurtarma ve Checkpoint
Cihaz, kritik bir durumu veya ara hesaplama sonucunu kaydetmek zorunda. Aniden güç gittiğinde, tekrar açılınca "kaldığı yerden" devam etmesi istenir. Bunu RAM'de tutamazsın. O yüzden belirli aralıklarla EEPROM'a checkpoint atarsın.

Teknik Uyarı: Bu Güzelim Yöntemler Bile Kendi Ayağına Sıkabilir

Her güzel şeyin bir limiti var. EEPROM'da da:

- Yazma ömrü: Her hücre 1.000.000 yazma/erase döngüsüne dayanır (modern çipler, evet, eski çipler 100k idi). Aynı adrese sürekli veri gömersin, ömrünü yersin. "Wear leveling" (aşınma dengeleme) ile yazmayı adrese yay, sık yazılan veriyi buffer'dan toplayıp toplu at.
- Veri bütünlüğü: EEPROM'a veri yazarken güç giderse, yazdığın değer yarım kalabilir, hatta o adres bozulabilir. Bunu önlemek için Brown-Out Reset (BOR) ve Power Good (PG) sinyaliyle yazmayı zamanında kes. Yazılımda ise checksum ya da CRC ile veri bütünlüğü kontrol et. Mesela bir byte'ın yanına tamamlayıcı ekle, tekrar okurken kontrol et. Hata varsa "default" değerlere dön.

Cihazı Açan Adamın Hikâyesi (EEPROM’lu Radyo Senaryosu)

Gece vakti Can, garajda arabasının kapısını açtı. anahtarı ile arabayı açtı, elini torpidoya uzattı, eski ama hala çalışan teybin güç tuşuna bastı.
Teyp bir “bip” sesiyle canlandı. İşte olanlar sırayla şunlar:

AŞAMA 1 – Teyp Açılırken

- Teyp’in işlemcisi hemen uyanıyor, “**setup**” fonksiyonundan ilk işi yapıyor:
- “Bakalım, adam geçen sefer hangi frekansta, hangi ses seviyesindeydi?”
- EEPROM’un belirli adreslerine gidiyor (diyelim 0x00 adresinden frekans, 0x01’den ses seviyesi okunacak).
- Kod, EEPROM.read(0x00) ile frekansı, EEPROM.read(0x01) ile sesi çekiyor.
- Bir yandan da yanındaki “checksum” değeriyle kontrol ediyor; veri sağlam mı diye bakıyor.
- Eğer veri tutarlıysa:
    - Teyp, doğrudan en son dinlenen radyoyu ve sesi ayarlıyor.
    - Hoparlörlerden hafif bir cızırtı, sonra da geçen hafta dinlenen o “Elektronik Fm”kanalı çalmaya başlıyor.
- Eğer veri bozuksa:
    - Kod “Geçersiz veri, fabrika ayarına dön” diyor.
    - Frekans 99.0 MHz, ses seviyesi 10 olarak atanıyor. Can anlamıyor bile, sistem akıyor.

Cihazı Açan Adamın Hikâyesi (EEPROM’lu Radyo Senaryosu)

AŞAMA 2 – Ayar Değişikliği

- Can sesi açıyor, başka frekansa geçiyor.
- Kod, her tuş basışında yeni frekans ve sesi RAM’de güncelliyor ama hemen EEPROM’a yazmıyor.
- Çünkü:
  - EEPROM’a her dokunuşta ömründen çalıyorsun.
  - Akıllı yazılım: “3 saniye boyunca ayar değişmezse yaz” mantığında.
- 3 saniye geçti, değişiklik sabit:
  - Kod EEPROM’a yeni frekansı ve sesi yazıyor.
  - Yanına da güncel checksum bırakıyor.
  - Tüm işlem birkaç milisaniyede bitti, Can keyfine bakıyor.

AŞAMA 3 – Güç Kapatma / Reset

- Can arabayı stop etti, elektrik gitti.
- Teyp işlemcisi anında durdu.
- Bir hafta sonra tekrar garajda, yine güç tuşu.
- Sistem tekrar başlarken, EEPROM’daki son ayarları çekiyor, kontrol ediyor.
- Geçen seferki ayarlar sağlam, aynı frekans, aynı ses. Can için her şey “kaldığı yerden”.

Cihazı Açan Adamın Hikâyesi (EEPROM'lu Radyo Senaryosu)

AŞAMA 4 – Beklenmeyen Güç Kesintisi

- Diyelim ki Can tam frekansı değiştirirken arabanın aküsü çekildi.
- Kod tam da EEPROM'a yazma yapıyordu.
- Son yazma tamamlanmadı, veri yarıda kaldı.
- Ertesi sefer açıldığında kod checksum kontrolüyle "bu veri bozuk, default'a dön" dedi.
- Sistem bozulmadı, Can en fazla "Teyp kendini sıfırladı galiba" dedi, ama cihaz açıldı, çöp olmadı. Can ise ayarları tekrardan yapıp teybinin keyfini çıkardı

AŞAMA 5 – Ekstra: Olay Kaydı

- Can teybi 1000 kere açtı, kapattı, her defasında kaç defa açıldı, kaç defa ayar değişti, bu bilgiler EEPROM'da sayaç olarak tutuluyor.
- Servise gittiğinde teknisyen, "Abi bu teyp 9862 defa açılmış, tuşlar biraz yıpranmış" diyebiliyor.

- Bu gerçek bir gömülü sistemin öyküsü:
- Her tuşun, her ayarın, her kapanışın ardında EEPROM'a yazılan küçücük bir veri, kullanıcıya "kaldığı yerden devam" hissi veriyor. Yazılım mühendisi olarak sen, "cihaz kapandı, ayar gitti, müşteri sinirlendi" riskini sıfırlıyorsun.

Cihazı Açan Adamın Hikâyesi (EEPROM'lu Radyo Senaryosu)

AŞAMA 4 – Beklenmeyen Güç Kesintisi

- Diyelim ki Can tam frekansı değiştirirken arabanın aküsü çekildi.
- Kod tam da EEPROM'a yazma yapıyordu.
- Son yazma tamamlanmadı, veri yarıda kaldı.
- Ertesi sefer açıldığında kod checksum kontrolüyle "bu veri bozuk, default'a dön" dedi.
- Sistem bozulmadı, Can en fazla "Teyp kendini sıfırladı galiba" dedi, ama cihaz açıldı, çöp olmadı. Can ise ayarları tekrardan yapıp teybinin keyfini çıkardı

AŞAMA 5 – Ekstra: Olay Kaydı

- Can teybi 1000 kere açtı, kapattı, her defasında kaç defa açıldı, kaç defa ayar değişti, bu bilgiler EEPROM'da sayaç olarak tutuluyor.
- Servise gittiğinde teknisyen, "Abi bu teyp 9862 defa açılmış, tuşlar biraz yıpranmış" diyebiliyor.

- Bu gerçek bir gömülü sistemin öyküsü:
- Her tuşun, her ayarın, her kapanışın ardında EEPROM'a yazılan küçücük bir veri, kullanıcıya "kaldığı yerden devam" hissi veriyor. Yazılım mühendisi olarak sen, "cihaz kapandı, ayar gitti, müşteri sinirlendi" riskini sıfırlıyorsun.

EEPROM’lu Radyo Senaryosu

Can gece geç saatte garajda. Arabasının kapısını açıyor, içeriye oturuyor, elini teybin açma tuşuna atıyor. Aküde hâlâ voltaj var, o klasik sarı 12V kablosu görevde – kontak açılır açılmaz teybe güç geliyor.

AŞAMA 1 — Güç ve Kablolar

- Arabanın elektrik sistemi çalışıyor:
-
    - Sarı 12V: Kontak açıldığında aktif.
    - kırmızı 12V: Aküden direkt geliyor, sürekli voltajda.
-
- Teyp açıldıktan sonra, mikrodenetleyici (MCU) “güç stabil mi?” diye kontrol ediyor.
- MCU, ‘power good’ pinine bakıyor. Eğer voltaj düşükse hemen EEPROM’a yazmayı engelliyor.

AŞAMA 2 — EEPROM Okuma ve Kullanıcı Ayarları

- MCU açılışta EEPROM.read() fonksiyonu ile son frekans, ses seviyesi gibi kullanıcı ayarlarını çekiyor.
- Can radyonun tuşlarına basıyor, farklı frekanslara geçiyor, sesi kısıyor. Bu arada veri RAM’de tutuluyor, EEPROM’a anında yazılmıyor.
- Kullanıcı bir süre tuşa basmazsa, MCU “şimdi EEPROM’a kaydedebilirim” diyor, güncel ayarları EEPROM’a yazıyor.
- Yazarken küçük bir kontrol değeri (CRC veya checksum) da kaydediyor, veri bozulmasın diye.

EEPROM’lu Radyo Senaryosu

AŞAMA 3 — Güç Kaybı/Reset

- Can arabayı kapatınca, kontak kapanıyor, sarı 12V kablosu güç kesiyor.
- Aküden gelen bayt 12V hala var ama cihaz kapanıyor, MCU uykuya geçiyor.
- Ertesi gün Can arabayı tekrar açınca, MCU hemen EEPROM’dan son ayarları geri yüklüyor.
    - Eğer veri sağlam, teyp aynen bıraktığı frekansta ve seste başlıyor.
    - Eğer veri bozuksa, fabrika ayarları yükleniyor.

AŞAMA 4 — Kötü Senaryo: Güç Kesintisi Sırasında Yazma

- Can ayarı değiştirdi, kaydediyordu, tam o anda elektrik gitti.
- EEPROM yazma yarıda kaldıysa, MCU CRC kontrolüyle veri bütünlüğünü kontrol ediyor.
- Bozuksa, sistem fabrika ayarına dönüyor; Can tekrar ayar yapmak zorunda kalıyor.

Dipnotlar:

- Sarı 12V kablosu: Kontak açıldığında teybe enerji verir, hafıza koruma için kritiktir.
- kırmızı 12V kablosu: Aküden sürekli enerji taşır, hafıza ve bazı devrelerin enerjisi için gereklidir.
- CRC/Checksum: EEPROM’a yazarken veri yanında kontrol kodu da yazılır, açılışta veri sağlam mı diye kontrol edilir.

EEPROM’lu Radyo Senaryosu

AŞAMA 3 — Güç Kaybı/Reset

- Can arabayı kapatınca, kontak kapanıyor, sarı 12V kablosu güç kesiyor.
- Aküden gelen bayt 12V hala var ama cihaz kapanıyor, MCU uykuya geçiyor.
- Ertesi gün Can arabayı tekrar açınca, MCU hemen EEPROM’dan son ayarları geri yüklüyor.
    - Eğer veri sağlam, teyp aynen bıraktığı frekansta ve seste başlıyor.
    - Eğer veri bozuksa, fabrika ayarları yükleniyor.

AŞAMA 4 — Kötü Senaryo: Güç Kesintisi Sırasında Yazma

- Can ayarı değiştirdi, kaydediyordu, tam o anda elektrik gitti.
- EEPROM yazma yarıda kaldıysa, MCU CRC kontrolüyle veri bütünlüğünü kontrol ediyor.
- Bozuksa, sistem fabrika ayarına dönüyor; Can tekrar ayar yapmak zorunda kalıyor.

Dipnotlar:

- Sarı 12V kablosu: Kontak açıldığında teybe enerji verir, hafıza koruma için kritiktir.
- kırmızı 12V kablosu: Aküden sürekli enerji taşır, hafıza ve bazı devrelerin enerjisi için gereklidir.
- CRC/Checksum: EEPROM’a yazarken veri yanında kontrol kodu da yazılır, açılışta veri sağlam mı diye kontrol edilir.