„inputenc margin=2.5cm

Statik Yük Bankası GUI
UART Simülatör Kullanım Dokümanı

Sürüm 1.1
Hazırlayan: Erva KANSU
Tarih: 09.10.2025

---

1. Amaç

Bu doküman, UART simülatör yazılımının test süreçlerinde nasıl kullanılacağını anlatmak amacıyla hazırlanmıştır. Simülatör, mikrodenetleyici davranışını yazılım ortamında taklit ederek, GUI üzerinden yapılan testlerin donanım olmadan da yürütülmesini sağlar. Bu sayede sistemin haberleşme yapısı, alarm koşulları ve röle/fan kontrolleri hızlıca doğrulanabilir.

2. Kapsam

Bu belge; simülatörün kurulumu, çalıştırma adımları, GUI ile haberleşme yapısı, test senaryoları, hata simülasyonları, CSV kayıt/replay ve temel sorun giderme adımlarını kapsar. Fiziksel donanım tasarım detayları kapsam dışıdır.

3. Gereksinimler

Gereksinim	Açıklama
İşletim Sistemi	Windows 10/11, Linux veya macOS
Python Sürümü	3.8 ve üzeri
Kütüphaneler	`pyserial`, `tkinter`, `csv`, `threading`
Seri Port	İki uçlu bağlantı (ör. Windows: com0com çifti)
Arayüz	`uart_demo_v6` GUI modülü

4. Çalışma Prensibi

Simülatör, belirlenen baud hızında UART üzerinden GUI'ye sıcaklık, fan devri, röle durumu ve alarm bilgilerini periyodik telemetri paketleri halinde gönderir. GUI, kullanıcı komutlarını (*röle aç/kapat, fan hızı ayarı*) simülatöre iletir; simülatör parametreleri günceller ve yeni durumu bildirir. Bu karşılıklı iletişim, cihaz olmadan fonksiyonel doğrulama yapılmasını sağlar.

5. Kurulum ve Başlatma

5.1. Sanal Port Oluşturma (Windows)

Windows için **com0com** ile bir sanal port çifti oluşturun ve **COM20 ↔ COM21** olarak adlandırın.

5.2. Hızlı Başlangıç (Aynı PC'de Sim + GUI)

Windows (com0com):

Listing 1: Windows'ta simülatör ve GUI başlatma

```
# Terminal A (SIM)
python uart_demo_v6.py sim --port COM20 --baud 9600 --tel 1000

# Terminal B (GUI)
python uart_demo_v6.py gui --port COM21 --baud 9600
```

Linux/macOS (iki ayrı USB-UART örneği):

Listing 2: Linux/macOS'ta simülatör ve GUI başlatma

```
# Terminal A (SIM)
python3 uart_demo_v6.py sim --port /dev/ttyUSB0 --baud 9600 --tel
    1000

# Terminal B (GUI)
python3 uart_demo_v6.py gui --port /dev/ttyUSB1 --baud 9600
```

5.3. Yardım Ekranı ve Kullanım

Komut satırına argüman verilmezse program yardım metnini yazdırır ve `exit code 0` ile çıkar.

Listing 3: Yardım/usage çıktısı örneği

```
python uart_demo_v6.py -h
# positional arguments: {sim,gui}
# sim Run MCU simulator (device)
# gui Run host GUI
# optional arguments:
# -h, --help show this help message and exit
```

6. Simülatörün Çalışma Mantığı (Detaylı)

Bu bölüm simülatörün iç mimarisini, veri akışını ve davranış modellerini daha teknik düzeyde açıklar. Amaç; geliştiricilerin, test mühendislerinin ve doğrulama ekiplerinin simülatörü hızlıca anlayıp gerektiğinde genişletebilmesidir.

6.1. Ana bileşenler

Simülatör aşağıdaki ana bileşenlerden oluşur:

- **Seri I/O katmanı:** Fiziksel veya sanal seri portla (pyserial) haberleşmeyi soyutlar.
- **Frame/Protokol katmanı:** Byte akışını çerçeveleyip (framing), CRC doğrulaması yapar ve komut/payload ayrıştırmasını gerçekleştirir.
- **Durum (State) deposu:** Röleler, fan ayarları, ölçülen RPM, sıcaklıklar, alarm bayrakları ve telemetri periyodu gibi çalışma değişkenlerini tutar.
- **Fiziksel model (isimsel):** Basit ısıl model ve fan dinamiği ile sensör/aktüatör davranışını taklit eder.
- **Fault Injection (hata enjeksiyon):** sensor_noise, sensor_stuck, fan_stall gibi bayraklarla anomali oluşturulmasını sağlar.
- **Zamanlama/Thread yapısı:** En az iki thread çalışır — RX (alıcı) ve Telemetry (periyodik yayın). Erişim senkronizasyonu `threading.Lock()` ile sağlanır.
- **Virtual Scope / Logger:** Gönderilen/alınan ham çerçeveleri hex formatında saklar ve insan okunur log üretir.

6.2. Başlatma akışı

Program `argparse` ile çalıştırma modunu ayırır:

- `sim` argümanı ile `Simulator` nesnesi oluşturulur ve `start()` çağrılır.
- Başlatma sırasında seri port açılır, başlatma durum snapshot'u alınır ve RX/Telemetry thread'leri devreye girer.

6.3. Frame yapısı ve CRC

Simülatörde kullanılan çerçeve örnek formatı:

Listing 4: Örnek çerçeve formatı

```
[0xAA] [CMD] [LEN_L] [LEN_H] [PAYLOAD...LEN] [CRC8] [0x55]
```

CRC hesaplaması CRC-8 (ATM, polinom 0x07, init 0x00) şeklindedir ve hesaplama genellikle `CMD+LEN+PAYLOAD` dizisi üzerinden yapılır. Güvenlik ve bütünlük için alıcı tarafta CRC doğrulanır; yanlışsa paket reddedilir ve hata olayı üretilir.

6.4. RX (alım) döngüsü — ayrıştırma mantığı

RX thread'i seri porttan gelen byte'ları okur ve bir ring-buffer / durum makinesi ile çerçeveleri toplar:

1. **START** (0xAA) aranır.
2. Başlık (CMD, LEN) okunduktan sonra payload beklenir.
3. Payload alındıktan sonra CRC ve END (0x55) kontrol edilir.
4. Eğer CRC ve uzunluk doğruysa `_handle_frame(cmd,payload)` çağrılır.

6.5. Komut işleyicileri (handler'lar)

Komutlar tipik olarak aşağıdaki şekilde ele alınır:

- `SET_REG` — Parametre yazma isteği: parametre doğrulaması yapılır, state güncellenir, hemen `SET_ACK` gönderilir ve isteğe bağlı kısa bir `TEL_EVT` tetiklenir.
- `GET_REG` — Mevcut parametrelerin snapshot'u alınır ve cevap olarak gönderilir.
- `TEL_EVT` — Normalde simülatör tarafından periyodik olarak gönderilen telemetri paketidir; alıcı (GUI) tarafında gösterim/CSV kaydı başlatılır.
- Hatalı paket veya bilinmeyen komutta `ERR_EVT` üretilir.

Basit bir handler örneği (psödo-kod):

```
def _handle_set_reg(key, value):
    with state_lock:
        if key == 'FAN_RPMS':
            state['fan_set'] = clamp(value, 0, max_rpm)
        elif key == 'RELAY':
            state['relays'][id] = bool(value)
        elif key == 'TEL_MS':
            state['tel_ms'] = int(value)
    send_set_ack(key)
    send_telemetry_snapshot()
```

6.6. Telemetri döngüsü ve fizik modeli

Telemetry thread'i konfigüre edilen periyot (`tel_ms`) kadar bekler ve her döngüde aşağıyı yapar:

1. Kısa bir fiziksel model integrasyonu: sıcaklıklar ve fan RPM güncellenir.
2. Alarm kontrolü: örn. `if T2 > 60.0: alarm = True`.
3. Anlık state snapshot'u alınır (kilit ile korunur).
4. Snapshot payload olarak çerçevelenir ve seri porta yazılır.

6.7. Fault Injection (hata enjeksiyonu)

Simülatör, runtime sırasında hataları açıp kapatmaya uygundur. Örnek etkiler:

- `sensor_noise = True` : Telemetriye rastgele küçük jitter eklenir.
- `sensor_stuck = True` : Belirlenen sensör sabitlenir, güncellenmez.
- `fan_stall = True` : Fan RPM fiziksel olarak sıfıra yaklaşır.

Bu bayraklar hem interaktif Python kabuğundan hem de özel test komutuyla değiştirilebilir; değişiklikler anında `SET_ACK` + yeni `TEL_EVT` ile bildirilir.

6.8. Virtual Scope ve loglama

- **Virtual Scope:** Seri port üzerinden gönderilen/alınan ham çerçeveleri hex biçiminde saklar; framing ve CRC kontrolü için kullanılır.

- **Logger:** İyi seviyeli olayları (TX SET_REG, RX SET_ACK, RX TEL_EVT) insan okunur şekilde yazar; debugging için hem hex hem açıklama sağlanır.

Scope çıktısı, log satırlarıyla birebir karşılaştırılarak framing ve CRC hataları hızlıca teşhis edilir.

6.9. CSV kaydı ve Replay

GUI tarafından başlatılan CSV kaydı telemetri alanlarını (timestamp, T1,T2,T3, FanRPM, RelayMask, AlarmFlag ...) satır satır yazar. Replay modu bu CSV'yi okuyup tekrarlı olarak GUI'ye veya test betiğine frame'ler göndererek zamanlama ve alarm doğrulaması yapar.

7. Test Senaryoları

Aşağıda hem manuel hem otomatik doğrulama için simülatöre özgü test senaryoları yer alır.

7.1. Komut Döngüsü Doğrulama

Adımlar:

1. GUI'den `SET_REG FAN_RPMS = 2000` gönder.
2. Simülatörün konsolunda veya logunda `tel changed ...` çıktısını gör.
3. GUI log: `TX SET_REG` → `RX SET_ACK` → `RX TEL_EVT` sırası olmalı.

7.2. Fault Davranışı Testi

1. `sim.faults.sensor_noise = True` yap.
2. Telemetride T değerlerinde jitter gözlemlenmelidir.
3. `sim.faults.sensor_stuck = True` yapılırsa seçilen sensör sabitlenmelidir.
4. `sim.faults.fan_stall = True` ile RPM düşmeli ve ilgili alarm tetiklenmelidir.

7.3. Performans / Zamanlama Testi

- `-tel` parametresini 1000 ms'den 200 ms'ye düşürün; telemetri frekansı artmalıdır.
- Komut yanıt süresi (SET_ACK gecikmesi) tipik 30 ms altında; maksimum tolerans dokümanlaştırılmalı (ör. 100 ms).

8. Sorun Giderme (Detaylı)

Aşağıdaki adımlar simülatör ile ilgili sık rastlanan problemlerin çözümünü içerir.

Port açılamıyor / Access denied: Doğru port seçildi mi? Windows'ta Aygıt Yöneticisi, Linux'ta `ls -l /dev/tty*`. Linux yetki sorunu için `sudo usermod -a -G dialout $USER` ve oturumu yeniden başlatın.

CRC / framing hatası: Virtual Scope ile ham byte'ları kontrol edin; gönderici ve alıcıda aynı baud ve çerçeve formatı kullanıldığından emin olun.

Telemetri gelmiyor: `tel_ms` değeri çok yüksek olabilir ya da telemetri thread'i başlamamış olabilir; sim log'unda "tel changed" veya "telemetry started" mesajlarını kontrol edin.

ModuleNotFoundError: serial: `pyserial` kurulu değil — `pip install pyserial`.

Deterministik olmayan testler: Fault jitter'i test amaçlı sabitlemek için `random.seed(x)` kullanın.

9. Güvenlik ve Kullanım Notları

- Simülatör sadece fonksiyonel doğrulama amaçlıdır; gerçek donanım güvenlik mekanizmalarını birebir simüle etmeyebilir.
- Test bittiğinde seri portları kapatın, log ve CSV dosyalarını uygun şekilde arşivleyin.
- Otomasyon betikleri üretirken timeout ve retry mantığını uygulayın (seri iletişim için).

Ek A – Komut Özeti & Hızlı Örnekler

Aşağıda simülatörle kullanılan başlıca komutlar ve bazı örnek değerler yer alır.

```
# Simulator (Windows)
python uart_demo_v6.py sim --port COM20 --baud 9600 --tel 1000

# GUI (Windows)
python uart_demo_v6.py gui --port COM21 --baud 9600

# Ornek: SET_REG cercevesi (psodo-bytes)
# [0xAA] [0x02=SET_REG] [len_l] [len_h] [payload...] [crc] [0x55]
# payload Orn: [KEY=0x10 (FAN_RPMS)] [VAL_L] [VAL_H]

# Hata enjeksiyonu (runtime, python shell)
sim.faults.sensor_noise = True
sim.faults.sensor_stuck = True
sim.faults.fan_stall = True

# Yardim:
python uart_demo_v6.py -h
```