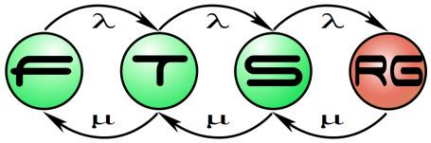
Tesztelési feladatok és kihívások a FALCON projektben

Búr Márton

Szoftver Verifikáció és Validáció

2016.12.07.



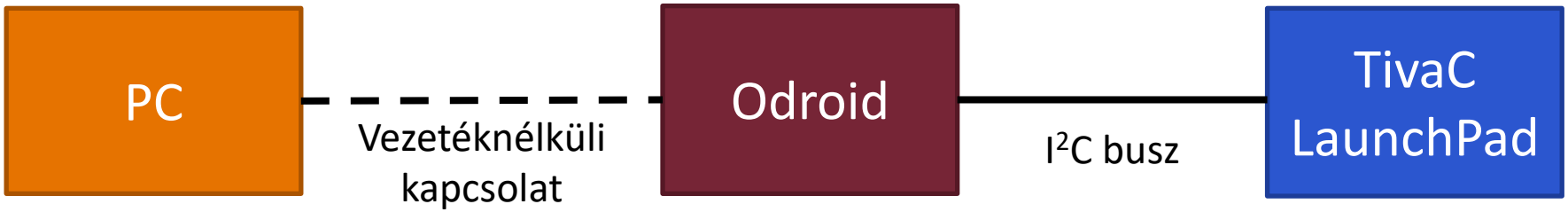
A FALCON projekt

- **F**irst **A**uto **L**ocating **C**amera **ON**-board

- Közvetítő drón jégkorong mérkőzésekhez
- Két fős mérnökcsapat (mechatronikus, informatikus)
- Hobbi projekt – szakmai ismeretek bővítése a cél
- Fejlesztés folyamatban 2015 január óta (nem befejezett)

A projekt rövid áttekintése

- Mechanikai részek
  - Saját tervezésű és gyártású alkatrészek: burkolat, kameratartó
  - Kész alkatrészek: váz, propellerek
- Elektronika
  - Saját tervezésű kiegészítő áramkörök: teljesítményelektronika, szintillesztő áramkör
  - Kész alkatrészek: motorvezérlők, mikrokontroller kártya
- Szoftverek
  - Futtató platformok: PC, egykártyás számítógép, mikrokontroller



A projekt rövid áttekintése

- Mechanikai részek
  - Saját tervezésű és gyártású alkatrészek: burkolat, kameratartó
  - Kész alkatrészek: váz, propellerek
- Elektronika
  - Saját tervezésű kiegészítő áramkörök: teljesítményelektronika, szintillesztő áramkör
  - Kész alkatrészek: mo
- Szoftverek
  - Futtató platformok: PC, egykártyás számítógép, okontroller

Prezentáció fókusza az itt futó szoftver vizsgálatának bemutatása

A beágyazott vezérlő szoftvere

- Tiszta C nyelvű forrás
  - ~3500 SLOC + külső mikrokontroller könyvtárak
- Feladatai:
  - Szenzorok figyelése
    - Jelek szűrése, feldolgozása
  - Kommunikáció a csatlakozó egykártyás számítógéppel
  - Repülésirányító szoftver
    - PID szabályozó algoritmus
- Időzítéskritikus feladatok

Szoftver vizsgálata

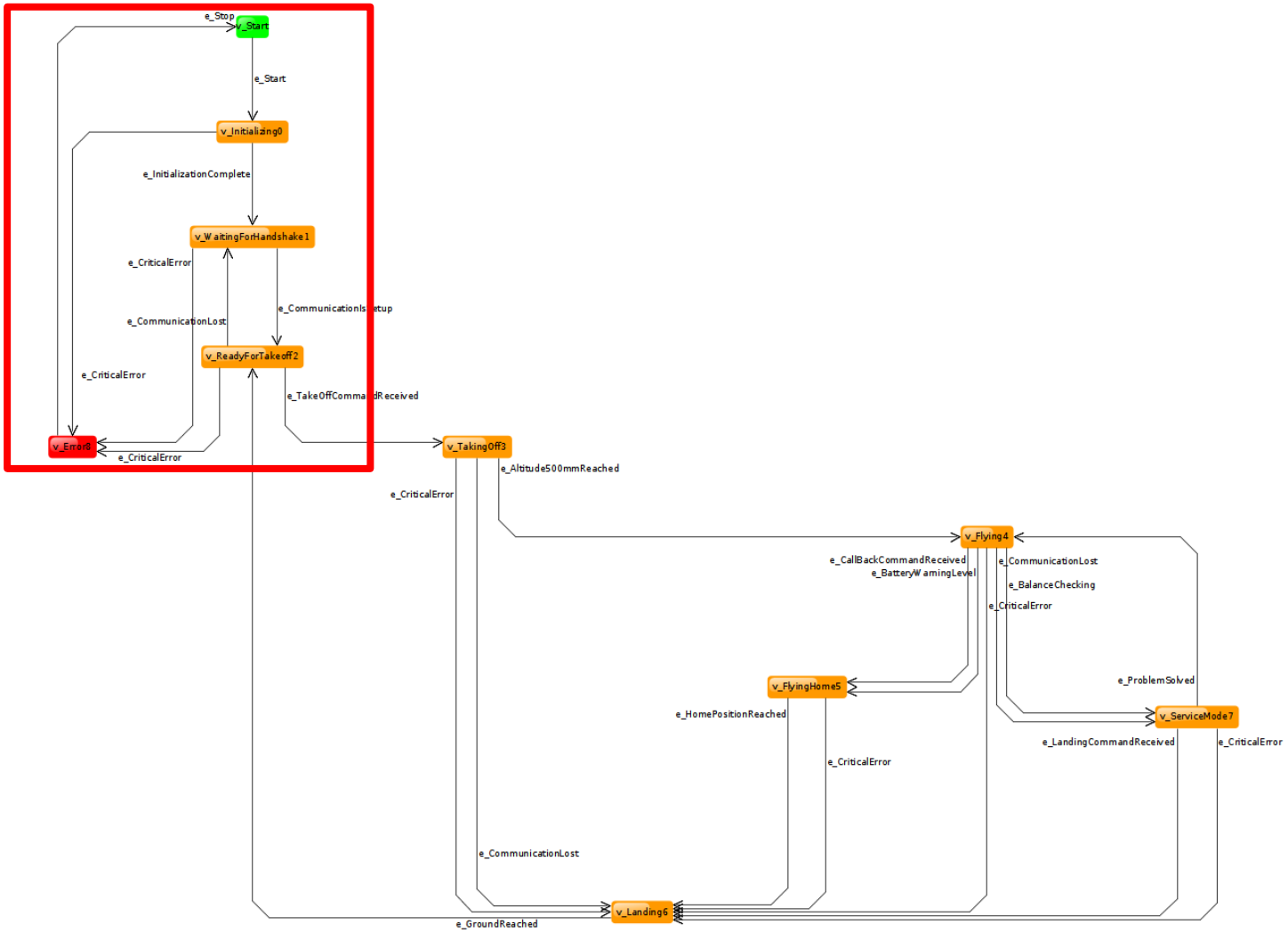
- Unit tesztek
  - Modell-alapú teszttervezési technika alkalmazása
    - Tesztmodell készítése
    - *GraphWalker* eszköz alkalmazása tesztesetek generálására
  - Unit/modulok izolációja kihívás
    - Kezdetben nem volt fontos szempont a tesztelhetőség
    - *Cmockery* teszt keretrendszer alkalmazása
- Integrációs tesztek
  - Komponensek együttműködése
- Interruptok priorizálása
  - Rate-monotonic scheduling algoritmus

Modell-alapú tesztelés

- Egy komponensre vonatkozó követelmények alapján viselkedésmodelleket készítettünk
  - Black-box teszteléshez
- Viselkedésmodellek: yEd-ben, formátuma graphml
- Tesztesetek származtatása *GraphWalker*rel
  - Csak azt vizsgáltuk, hogy az események hatására melyik állapotot melyik állapot követ

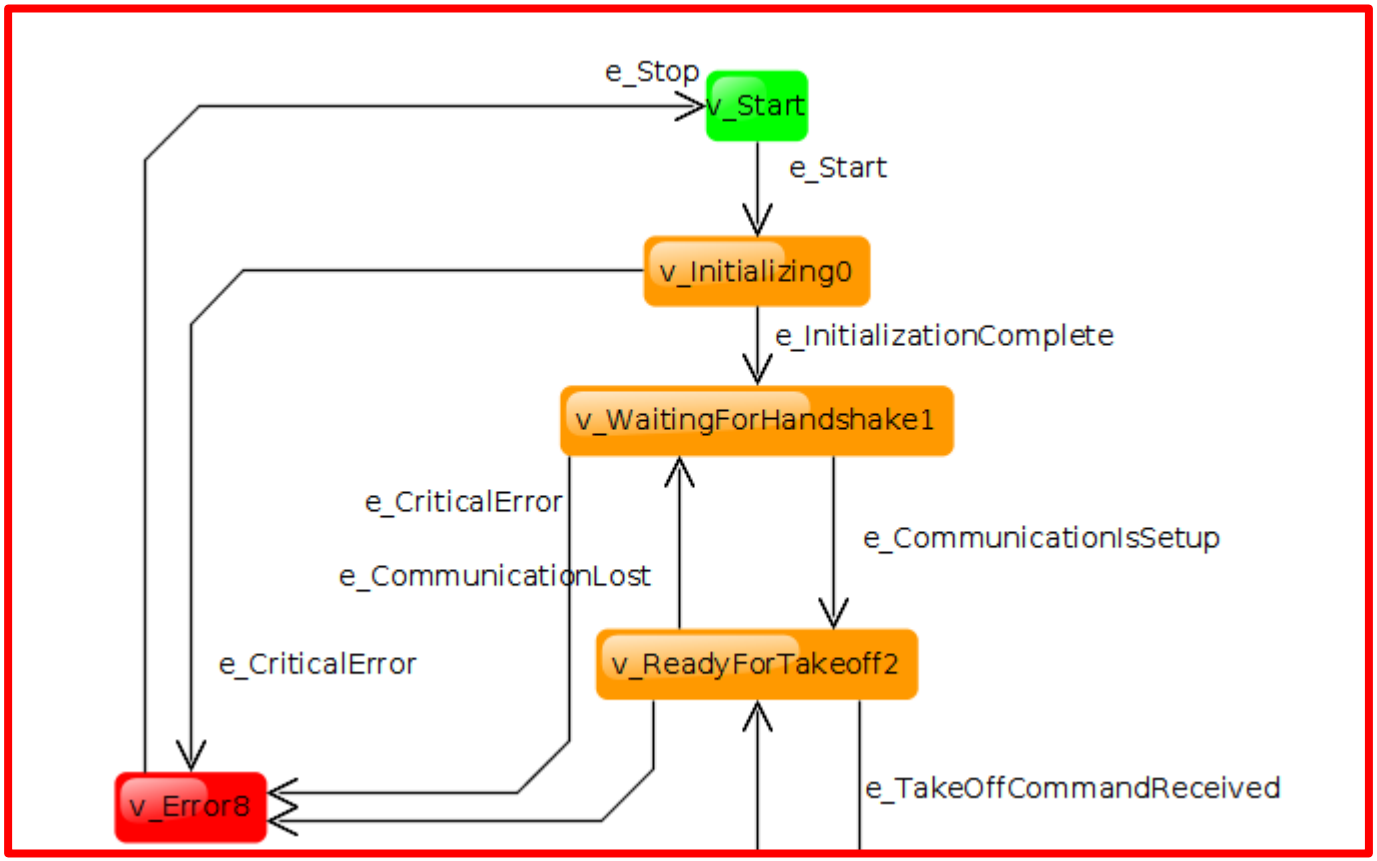
Tesztesetek generálása GraphWalkerrel

- Bemenet:



Tesztesetek generálása GraphWalkerrel

- Bemenet (részlet kinagyítva):



Tesztesetek generálása GraphWalkerrel

- Eszköz paraméterei:
  - Kezdőállapot
  - Cél
    - Állapot/átmenetfedési kritérium
    - Célállapot
  - Használt algoritmus
- Kimenet: állapotok és átmenetek sorozata

```
Quick random vertex traversal
{"currentElementName":"v_Start"}
{"currentElementName":"e_Start"}
{"currentElementName":"v_Initializing0"}
{"currentElementName":"e_InitializationComplete"}
{"currentElementName":"v_WaitingForHandshake1"}
…
```

Unit tesztek megvalósítása

- A tesztesetek alapján unit tesztek kézi megvalósítása
- Nehézségek:
  - Számos közös, globális változó
  - Perifériákat kezelő függvények
  - Kommunikáció más komponensekkel a működés során

Unit tesztek megvalósítása – megoldások

- Forráskód felkészítése a tesztelésre
  - `#define UNIT_TESTING 1` preprocesszor makró segítségével a meglévő kód kiegészítése és felkészítése unit tesztelésre

```
#ifdef UNIT_TESTING
volatile uint32_t time5us;
float eulersRef[3];
float eulersAct[3];
float eulersErr[3];
#else
extern volatile uint32_t time5us;
extern float eulersRef[3];
extern float eulersAct[3];
extern float eulersErr[3];
#endif


void ESCDutiesRefresh() {
#ifndef UNIT_TESTING
    PWMPulseWidthSet(PWM0_BASE, PWM_ESC1, PWM_PERIOD_400HZ * duties[0] / ESC_DUTY_MAX);
    PWMPulseWidthSet(PWM0_BASE, PWM_ESC2, PWM_PERIOD_400HZ * duties[1] / ESC_DUTY_MAX);
    PWMPulseWidthSet(PWM0_BASE, PWM_ESC3, PWM_PERIOD_400HZ * duties[2] / ESC_DUTY_MAX);
    PWMPulseWidthSet(PWM0_BASE, PWM_ESC4, PWM_PERIOD_400HZ * duties[3] / ESC_DUTY_MAX);
#endif
}
```

Unit tesztek megvalósítása – megoldások

- *Cmockery* használata mockok készítésére
  - Mock: kommunikáció esetére az egyes komponensek között

```
#if UNIT_TESTING 1
char I2CSlaveDataGet(int reg_addr){
    return (char)mock();
}
#endif
```

Perifériakezelő függvény definiálása a Cmockery framework egy makrójával

Unit tesztek megvalósítása – megoldások

- *Cmockery* használata mockok készítésére
  - Mock: kommunikáció esetére az egyes komponensek között

```
#define I2C_TAKE_OFF_REQUEST 0x02
#define FLY_BACK_HOME_REQUEST 0x04
void test_from_takeoff_to_flyhome(){
    int i; //Test setup - initial state, mocking communication
    will_return(I2CSlaveDataGet,I2C_TAKE_OFF_REQUEST);
    for (i = 0; i < 4; ++i) { // a message is 5 bytes long
        will_return(I2CSlaveDataGet,0);
    }
    will_return(I2CSlaveDataGet,FLY_BACK_HOME_REQUEST);
    setDeviceState(STATE_READY_FOR_TAKEOFF);
    // Execute test scenario & check states
    for (i = 0; i < 5; ++i) { // read a message
        readI2CByte();
    }
    assert_int_equal(getDeviceState(),STATE_TAKING_OFF);
    updateAltitude(5000); // setting altitude - callback for interrupts
    assert_int_equal(getDeviceState(),STATE_FLYING);
    readI2CByte(); // read only the next message header
    assert_int_equal(getDeviceState(),STATE_FLYING_HOME);
}
```

Unit tesztek megvalósítása – megoldások

- *Cmockery* használata mockok készítésére
  - Mock: kommunikáció esetére az egyes komponensek között

```
#define I2C_TAKE_OFF_REQUEST 0x02
#define FLY_BACK_HOME_REQUEST 0x04
void test_from_takeoff_to_flyhome(){
    int i; //Test setup - initial state, mocking communication
    will_return(I2CSlaveDataGet,I2C_TAKE_OFF_REQUEST);
    for (i = 0; i < 4; ++i) { // a message is 5 bytes long
        will_return(I2CSlaveDataGet,0);
    }
    will_return(I2CSlaveDataGet,FLY_BACK_HOME_REQUEST);
    setDeviceState(STATE_READY_FOR_TAKEOFF);
    // Execute test sc & check states
    for (i = 0; i < 5
        readI2CByte(
    }
    assert_int_equal
    updateAltitude(5                                    for interrupts
    assert_int_equal
    readI2CByte(); //
    assert_int_equal(getDeviceState(),STATE_FLYING_HOME);
}
```

Egy sorba kerülnek a kívánt visszatérési értékek (record-replay)

Unit tesztek megvalósítása – megoldások

- *Cmockery* használata mockok készítésére
  - Mock: kommunikáció esetére az egyes komponensek között

```
#define I2C_TAKE_OFF_REQUEST 0x02
#define FLY_BACK_HOME_REQUEST 0x04
void test_from_takeoff_to_flyhome(){
    int i; //Test setup - initial state, mocking communication
    will_return(I2CSlaveDataGet,I2C_TAKE_OFF_REQUEST);
    for (i = 0; i < 4; ++i) { // a message is 5 bytes long
        will_return(I2CSlaveDataGet,0);
    }
    will_return(I2CSlaveDataGet,FLY_BACK_HOME_REQUEST);
    setDeviceState(STATE_READY_FOR_TAKEOFF);
    // Execute test se          & check states
    for (i = 0; i <
        readI2CByte
    }
    assert_int_equa
    updateAltitude(                                    interrupts
    assert_int_equa
    readI2CByte(); /
    assert_int_equal(getDeviceState(),STATE_FLYING_HOME);
}
```

Kiinduló állapot beállítása – kisebb módosító és lekérdező függvények kellhetnek

Unit tesztek megvalósítása – megoldások

- *Cmockery* használata mockok készítésére
  - Mock: kommunikáció esetére az egyes komponensek között

```
#define I2C_TAKE_OFF_REQUEST 0x02
#define FLY_BACK_HOME_REQUEST 0x04
void test_from_takeoff_to_flyhome(){
    int i; //Test setup - initial state, mocking communication
    will_return(I2CSlaveDataGet,I2C_TAKE_OFF_REQUEST);
                                    message is 5 bytes long
                                  ,0);

                                  _BACK_HOME_REQUEST);
                                  TAKEOFF);
    // E          test scenario & check states
    for (i =      i < 5; ++i) { // read a message
        rea      Byte();
    }
    assert_int_equal(getDeviceState(),STATE_TAKING_OFF);
    updateAltitude(5000); // setting altitude - callback for interrupts
    assert_int_equal(getDeviceState(),STATE_FLYING);
    readI2CByte(); // read only the next message header
    assert_int_equal(getDeviceState(),STATE_FLYING_HOME);
}
```

Assertionök a modul állapotára

Cmockery példa kimenet

```
test_from_takeoff_to_flyhome: Starting test
test_from_takeoff_to_flyhome: Test completed successfully.

test_from_taking_off_to_ready_to_takeoff: Starting test
2 != 0
ERROR: ../src/main.c:77 Failure!
test_from_taking_off_to_ready_to_takeoff: Test failed.

1 out of 2 tests failed!
    test_from_taking_off_to_ready_to_takeoff
```

Integrációs tesztek

- Ugyanazzal az eszközkészlettel, mint a unit tesztek
- Kevesebb mockolt résztvevő (függvény)
- Példa: I^2C kommunikáció integrációs tesztje:
  - Mindkét résztvevő egy egyszerű kommunikációs forgatókönyvet játszik le
  - Még mindig csak a komponensek kis része valós

Interruptok priorizálása

- Periodikus taszkok prioritási szintjeinek beállítása
- Offline feladat
- Alapötlet: rövidebb feladatok nagyobb prioritási szinten
- Figyelembe vett jellemzők:
  - n: interrupt rutinok száma (taszkok száma)
  - C_i: i. interrupt futási ideje
  - T_i: i. interrupt periódusideje
  - μ_i: i. interrupt CPU kihasználása; $\mu_i = C_i / T_i$

Interruptok priorizálása

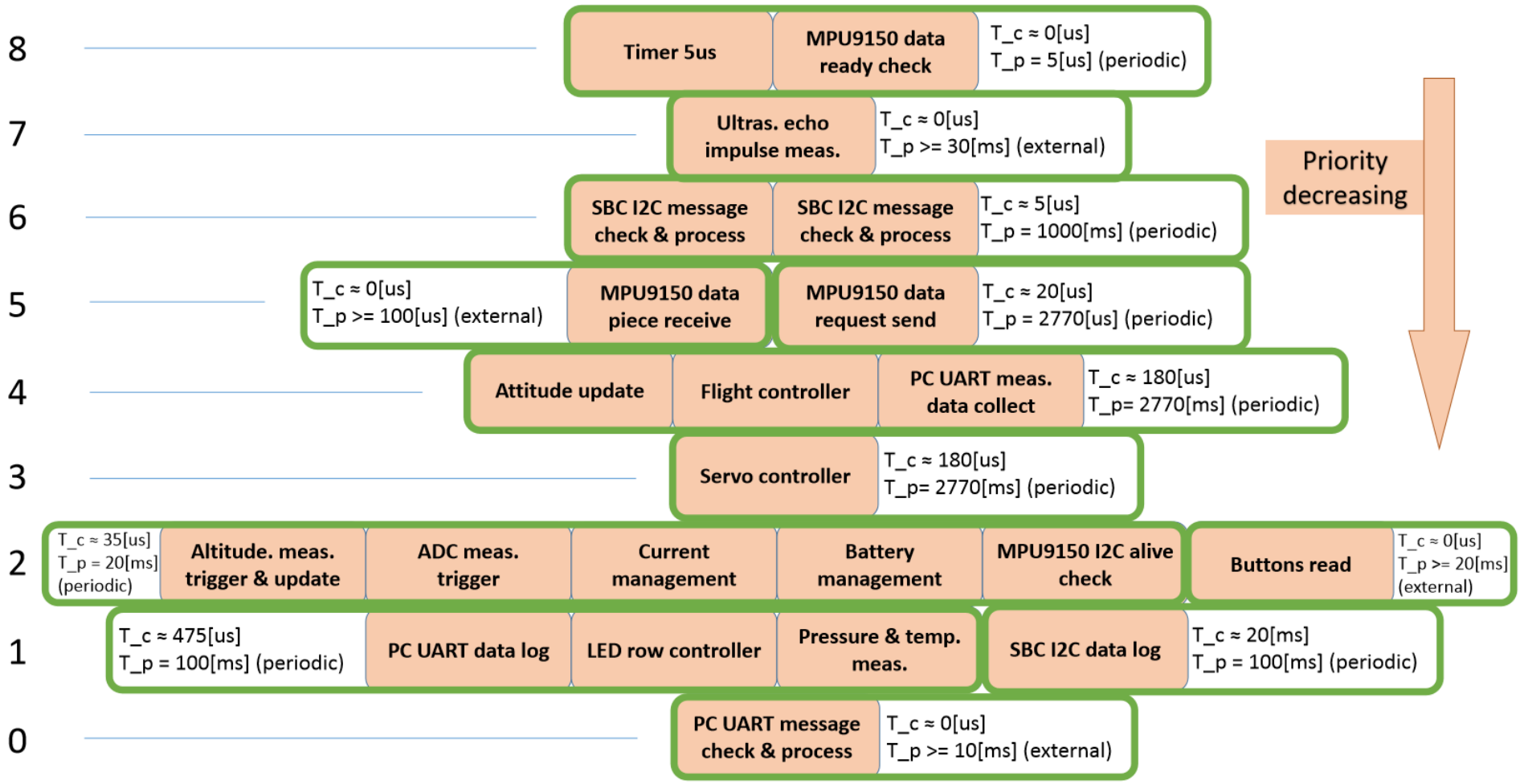
- Ütemezhetőség feltétele:

$$\mu = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(\sqrt[n]{2} - 1\right)$$

- FALCON esetén:
  - $n = 12$
  - C_i: az egyes rutinok futási idejét hardveres időzítővel lemértük
    - Flagek beállítása, értékek lementése sokszor közel 0 idő alatt megtörténik
  - T_i: periódusidőket mi határoztuk meg
    - 0,005 ms – 100 ms közötti értékek

$$\mu = 0.284 < 12\left(\sqrt[12]{2} - 1\right) = 0.714$$

Interrupt priorizálás példa kimenet



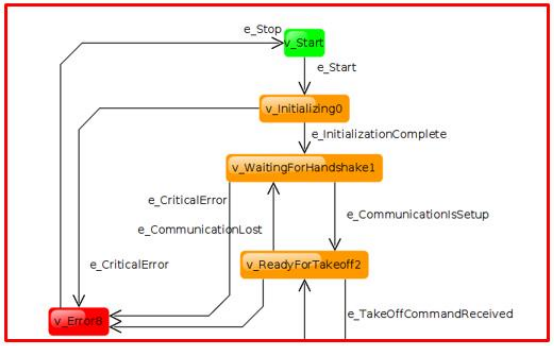
Összefoglalás

A beágyazott vezérlő szoftvere

- Tiszta C nyelvű forrás
  - ~3500 SLOC + külső mikrokontroller könyvtárak
- Feladatai:
  - Szenzorok figyelése
    - Jelek szűrése, feldolgozása
  - Kommunikáció a csatlakozó egykártyás számítógéppel
  - Repülésirányító szoftver
    - PID szabályozó algoritmus
- Időzítéskritikus feladatok

Tesztesetek generálása GraphWalkerrel

- Bemenet (részlet kinagyítva):

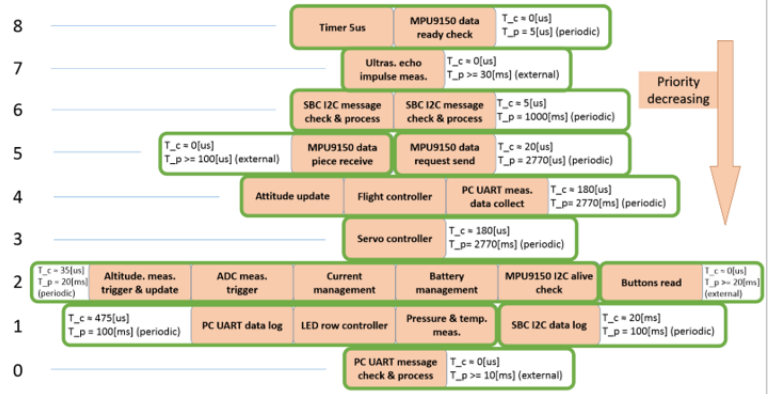


Unit tesztek megvalósítása – megoldások

- *Cmockery* használata mockok készítésére
  - Mock: kommunikáció esetére az egyes komponensek között

```
#define I2C_TAKE_OFF_REQUEST 0x02
#define FLY_BACK_HOME_REQUEST 0x04
void test_from_takeoff_to_flyhome(){
    int i; //Test setup - initial state, mocking communication
    will_return(I2CSlaveDataGet,I2C_TAKE_OFF_REQUEST);
    for (i = 0; i < 4; ++i) { // a message is 5 bytes long
        will_return(I2CSlaveDataGet,0);
    }
    will_return(I2CSlaveDataGet,FLY_BACK_HOME_REQUEST);
    setDeviceState(STATE_READY_FOR_TAKEOFF);
    // Execute test scenario & check states
    for (i = 0; i < 5; ++i) { // read a message
        readI2CByte();
    }
    assert_int_equal(getDeviceState(),STATE_TAKING_OFF);
    updateAltitude(5000); // setting altitude - callback for interrupts
    assert_int_equal(getDeviceState(),STATE_FLYING);
    readI2CByte(); // read only the next message header
    assert_int_equal(getDeviceState(),STATE_FLYING_HOME);
}
```


Interrupt priorizálás példa kimenet



Hivatkozások

- GraphWalker: graphwalker.github.io
- Cmockery: https://github.com/google/cmockery
- http://blogs.grammatech.com/unit-testing-c-programs-with-mock-functions
- https://en.wikipedia.org/wiki/Rate-monotonic_scheduling

Fénykép az eszközről (burkolat nélkül)