Protocolbuffer implementálása NUCLEO-STM32H7A3ZI-Q kártyára

A program rövid leírása

A PC-n futó Python script felhasználói parancsra képes fix hosszúságú üzenetet küldeni protocolbuffer segítségével az STM32H7A3ZI-Q kártyára. Ezt a terminálon keresztül a ‘on’, ‘off’ parancsal teheti meg a felhasználó. Az üzenet a LED1 ledet képes bekapcsolni illetve kikapcsolni.

A program használata

1. A C nyelvű programot fel kell tölteni a mikrokontrollerre, ami az STM32CubeIDE környezetbe történő projektimportálással lehetséges. Amennyiben erre nincs lehetőség a protobuffer/Debug mappában található .hex fájl letölthető más programozó program segítségével is.
2. Ezután a Python scriptet (user.py) kell futtatni, majd a terminálon keresztül kiadhatóak a parancsok a mikrokontorllernek.

Példa

Terminál:

```
Enter a command ('on' / 'off'): on
Command sent: ON
Enter a command ('on' / 'off'): off
Command sent: OFF
```

A program részletes leírára és működése

A felhasznált könyvtár (https://github.com/nanopb/nanopb) segítségével a proto_generator\messeges mappában található led_blink.proto fájlból generált led_blink.c fájlt a C nyelvű programba, míg a led_blink.py fájlt a Python programba kell beilleszteni.Ezen kívül a könyvtár pb_common.c/.h, pb_encode.c/.h, pb_encode.c/.h és a pb.h fájlok beillesztése is elengedhetetlen a mikrokontrolleres program esetében.

C nyelvű program

Felhasznált perifériák:

- USART3-on keresztül kommunikál a STM32 a PC-vel. Aszinkron interruptos módban, 115200 Baudrate-el van felkonfigurálva.
- LED1 led

Main.c

Globális változók:

- volatile uint8_t buffer[BUFFER_LEN] (itt tárolódnak az adatok)
- static volatile uint8_t data[32] (karakterenként itt gyűlik össze egy adat)
- static volatile bool LedStat (üzenet dekódolása sikeres)
- static uint32_t DataCounter (számolja, hány feldolgozatlan adat van)
- static volatile uint8_t UartSign (érvényes adatok deklarálása)

Definok:

- #define MESSAGE_LENGTH 2 (előre definiált az üzenet hossza)
- #define MESSAGE_ID 0x08 (Az adott üzenet azonosítója)

- #define LED_ON 1
- #define LED_OFF 0

Függvények

- ***void HAL_UART_RxCpltCallback (UART_HandleTypeDef * huart)***

  Uart interrupt Callback fügvénye. Ellenőrzi, hogy a USART3-tól jött-e a megszakítás. Majd az érvényes üzeneteket message_length hosszan beolvassa és eltárolja a cirkuláris bufferben. A nem érvényyes adatokat figyelmen kívül hagyja. Amennyiben a bufferben nincs hely, feldolgoz egy adatot.

- ***LedStatus decode()***
  A függvény feladata, hogy dekódolja a protocolbuffer üzenetet és visszaadja azt.

- ***void execute(LedStatus message)***
  Feladata, hogy beállítsa a Led állapotát az üzenetnek megfelelően. Ha nem megfelelő karakter érkezik azt egyszerűen figyelmen kívül hagyja.

- ***int __io_putchar(int ch)***
  Ez egy felülírt fügvény, melyet a write függvény használ, hogy a a debugoláshoz használható legyen a printf() függvény.

Cirkularbuffer.c/.h

Globális változók:

- volatile uint32_t writePtr
- volatile uint32_t readPtr

Define:

- #define BUFFER_LEN (500) (A buffer mérete (minimum MESSAGE_LENGTH))

Függvények:

- ***void getdata_frombuffer(uint8_t data[],uint8_t len)***
  A paramétrül kapott *data* változóba tesz *len* méretű üzenetet a bufferből a readptr által mutatott részről. Ez fügvény cask akkor hívódik meg ha van feldolgozatlan adat, tehát mindig képes visszaadni egy üzenetet.

- ***bool writedata_tobuffer(uint8_t data[],uint8_t len)***
  A paraméterül kapott *data* vátozóból *len* méretű üzenetet helyez a bufferbe. Ha a a writePtr utolérte a ReadPtr, nem írja be az adatot a még feldolgozatlan adat helyére. Ilyenkor visszatérési értéke 1

Python script – user.py

A Python script lehetővé teszi, hogy a felhasználó a terminálon keresztül on és off parancsokat adjon meg.
A script:

- Beolvassa a felhasználó által megadott parancsot.
- Beállítja a **message.status** értékét a megfelelő módon.
- Soros porton elküldi a kódolt üzenetet a mikrokontrollernek.