1 INLÄMNINGSUPPGIFT/LABB 04

1.1 ÖVERGRIPANDE

Uppgiften utförs och examineras individuellt. Det är självklart tillåtet att be andra om hjälp och att använda resurser från internet — så länge du inte bryter mot några licensavtal och korrekt anger källor på tredjepartskod du använt. Att inte göra det utgör akademiskt fusk, och resulterar i underkänd uppgift och/eller disciplinära åtgärder. Du ska kunna stå för och förklara varje rad kod du lämnar in.

Givna funktionssignaturer ska följas.

Deadline för labben är **Fredag 12/11, KLOCKAN 23:59**!

Är något otydligt, eller om du inte förstår uppgiften, är det ditt ansvar att påtala det och begära ett förtydligande.

1.2 INLÄMNING

Lösningen lämnas i samma form som du får ut baselinen, dvs ett `.tar.gz`-arkiv (eller .zip) innehållande ett gitrepo med lösningarna. Repot skall innehålla en (1) uppsnyggad och väl beskriven commit innehållande respektive deluppgift, i nedanstående ordning. Det ska *tydligt* framgå ur respektive commit-meddelande vilken deluppgift commiten avses lösa. Koden ska vara prydlig och tydlig — *Linux Kernel Coding Style* (https://www.kernel.org/ doc/html/v4.10/process/coding-style.html) är en lämplig grundplåt att utgå från.

1.3 BETYGSKRAV

För *Godkänt* krävs inlämnade lösningar på deluppgifter 0–2 med den efterfrågade funktionaliteten. Mindre misstag kan tolereras, men funktionen överlag ska finnas och indikera förståelse för problemet. Koden skall kompilera. Du skall kunna förklara din kod.

För *Väl Godkänt* krävs, förutom kraven för godkänt, att laborationen lämnats in i tid, är fullständigt korrekt, samt att en korrekt lösning på deluppgift 3 tillhandahålls.

1.4 UPPGIFT

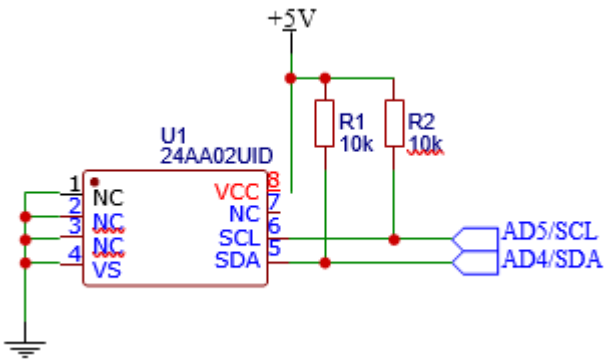
Labben går i stora drag ut på att konfigurera och använda I²C-controllern på Arduinon för att kommunicera med ett externt EEPROM, och få viss känsla för användningsområden för detta.

Syftet med detta är i första hand att bekanta dig med I²C-protokollet, dess användning och konfiguration, och hur det kan vara implementerat i praktiken. Du lär dig också använda ett externt minne för att spara och ladda parametrar. Återigen även lite elschemaläsning.

Du tillhandahålls en ’baseline’ med nödvändiga byggscript och källkodsfiler — utgå från den när du löser labben. Du får även ut en uppsättning nödvändiga elektronikkomponenter. Arduino UNO R3 samt kopplingsdäck och kablar tillhandahåller du själv. Du tillhandahålls också en uppsättning scheman och relevanta datablad.

1.4.0 DELUPPGIFT 0: KOPPLA UPP FÖLJANDE KRETS PÅ DITT KOPPLINGSDÄCK

OBS att det angivna EEPROMet inte ingår i några Arduino-kit eller liknande, och du ska inte använda något annat än det du får ut i kursen. EEPROMet skall vara av typen ’Microchip `24AA02UID`’.



1.4.1 DELUPPGIFT 1: KONFIGURERA OCH ANVÄND I²C GENERELLT

1. Implementera följande funktioner, deklarerade i `i2c.h`, med hjälp av databladet eller eventuella externa resurser.
   - `i2c_init()` — Kraftsätt och initiera enheten, sätt bit rate till 100 kHz
   - `i2c_start()` — Generera start condition
   - `i2c_stop()` — Generera stop condition
   - `i2c_get_status()` — Läs av och returnera status för föregående operation
   - `i2c_emit_address()` — Skicka SLA+R/W
   - `i2c_emit_byte()` — Skicka enstaka byte
   - `i2c_read_ACK()` — Läs enstaka byte, ACKa
   - `i2c_read_NAK()` — Läs enstaka byte, utan att ACKa
2. Du har även fått funktionen `i2c_meaningful_status()` på köpet i baselinen — den tar returvärdet från `i2c_get_status()` och skriver ut statuskodens innebörd i klartext på UARTen

   Samtliga funktioner ska blockera tills operationen utförts — välj själv om du väljer en lösning med avbrott + state machine eller rena busy wait-loopar
3. Använd förslagsvis funktionaliteten i `eeprom_read_byte()` för att testa ovanstående
4. Städa upp din kod och skapa en commit med ovanstående

Forts. →

1.4.2 DELUPPGIFT 2: IMPLEMENTERA OCH DEMONSTRERA EEPROM-KOMMUNIKATION

1. Implementera följande funktioner, även de i `i2c.[ch]`, **med hjälp av funktionerna du skrev i del 1**, och databladet för EEPROMet.
   - `eeprom_wait_until_write_complete()` — Blockera tills EEPROMets senaste skrivoperation slutförts; används internt — se avsnitt 7.0 *Acknowledge polling* i databladet
   - `eeprom_read_byte()` — Läs en enskild byte från EEPROMet; *Random read* i databladet
   - `eeprom_write_byte()` — Skriv en enskild byte till EEPROMet; *Byte write* i databladet
2. Testa lämpligen `eeprom_read_byte()` genom att läsa ut data från det inbyggda 'serial number'-blocket i EEPROMet.
3. Demonstrera `eeprom_write_byte()` genom att vid uppstart skriva in ditt förnamn på/från adress 0x10 och framåt i EEPROMet, och sedan läsa ut det löpande och skriva ut det på UARTen
4. Städa upp din kod och skapa en commit med ovanstående

1.4.3 DELUPPGIFT 3 (VG-KRAV): UTÖKAD FUNKTIONALITET

För deluppgift 3 är det tillåtet att ändra funktionssignaturerna vid behov

1. Implementera följande funktioner med hjälp av ovanstående och databladet
   - `eeprom_write_page()` — Skriv hel page (8 B) till EEPROMet
     - Läs noten i avsnitt 6.2 *page write* noggrannt!
   - `eeprom_sequential_read()` — Läs hela eller delar av EEPROMet sekvensiellt
2. Demonstrera funktionen genom följande:

Skriv några pages med data i EEPROMet vid uppstart, och läs sedan ut samtligt data i en operation och skriv ut en hexdump (samt "char:s" om det är text du skrivit in) av det på UARTen.

3. Implementera följande funktioner med hjälp av ovanstående funktioner
   - `eeprom_sequential_write()` — Skriv hela eller delar av EEPROMet sekvensiellt.

   För att göra det så optimerat som möjligt behöver datat som ska skrivas delas upp på så många fulla pages som möjligt, och de sista bytes-en som blir över (om det blir några) får skrivas in en och en.

Skriv godtycklig mängd med data i EEPROMet vid uppstart, och läs sedan ut samtligt data i en operation och skriv ut en hexdump (samt "char:s" om det är text du skrivit in) av det på UARTen.

4. Städa upp din kod och skapa en commit med ovanstående