1 INLÄMNINGSUPPGIFT/LABB 03

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 **Tisdag 9/11, 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 (alternativt .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 1–3 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 4 tillhandahålls.

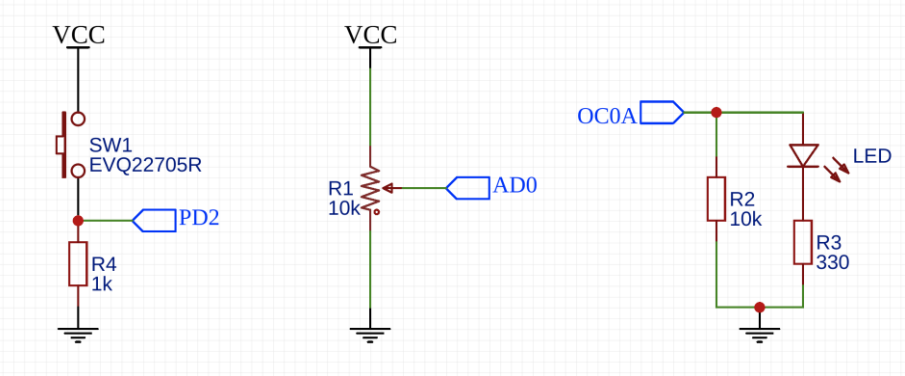
1.4 UPPGIFT

Labben går i stora drag ut på att dels använda ADC-enheten på Arduinon för att läsa av en potentiometer, en digitalingång för att läsa av en knapp, och dels använda timer/compare-enheter för att takta systemet. Interrupt ska användas, busy wait-loopar lämnar vi bakom oss!

Syftet med det är i första hand att bekanta dig med ADCn, interrupts, problematik som kan uppstå vid knappavläsning (knappstuds – fenomenet förekommer dock i fler elektromekaniska komponenter). Vi övar också 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.

1.4.1 DELUPPGIFT 1: KOPPLA UPP FÖLJANDE TRE KRETSAR PÅ DITT KOPPLINGSDÄCK



Rättelse: R4 ska vara 10k

Labben är i grunden tänkt att kopplas upp enligt ovan ”av skäl”. Kan du lista ut varför?
Om du ändå tänker använda din egen shield till den här labben bör du tänka både en och två gången hur / varför det skulle / inte skulle funka. Du e smart. Du listar nog ut varför jag tar upp det här…
Om du till slut ändå bestämmer dig för att använda din sköld ”till vissa delar” av den här labben ska det framgå tydligt i readme.txt, kommentarer och commits hur och varför du gjort det. Det blir väldigt svårt för mig att rätta din uppgift annars…

1.4.2 DELUPPGIFT 2: LÄS AV DIGITALINGÅNG

1. Konfigurera upp pinne PD2 som ingång, som använder ett externt pulldown motstånd.
2. Skriv, och anropa periodiskt, en funktion som läser av pinnen och skriver ut `'pushed\r\n'` och `'released\r\n'` på UARTen när knappen trycks in, respektive släpps upp. Meddelandena ska bara skrivas ut *en* gång per knapptryck
3. Städa upp din kod och skapa en git-commit med ovanstående.

1.4.3 DELUPPGIFT 3: LÄS SPÄNNING MED HJÄLP AV ADCN

1. Konfigurera upp ADC-enheten mha kap 28 i databladet:
    - Single conversion
    - Bit-data, vänsterjusterat
    - Prescaler = 8, referensspänning V_{CC}
    - Använd kanal ADC0
2. Skriv en avbrottshanterare för ’ADC conversion complete’, och i den, läs ut ADC-värdet och spara undan det i en global variabel
3. Konfigurera upp timer0 i Fast PWM-mod på samma sätt som i labb 2
    - Fast PWM-mod, 255 som TOP, prescaler = 64
4. Konfigurera upp timer2 i CTC-mod på samma sätt som i labb 2, men slå på interrupts för compare match, och använd inte busy wait-loopar
    - CTC-mod, 100 Hz, prescaler 1024, ’clear on match/set on bottom’
    - Skriv en avbrottshanterare för compare match A, och i den, starta en ADC-omvandling, och skriv föregående ADC-värde (sparat i variabeln) till OCR0A
5. Städa upp din kod och skapa **en** git-commit med ovanstående.

Förväntat beteende: potentiometern kan användas för att styra ljusstyrkan på LEDen. Moturs bottenläge = helt släckt, medurs bottenläge = helt på

Fortsättning →

1.4.4 DELUPPGIFT 4 (VG-KRAV): ANVÄNDARSTYRT BETEENDE

1. Kombinera koden från de två föregående deluppgifterna, och ändra den så att följande beteende uppnås:
    - 10ms-timern används också för att periodiskt läsa av knappen
    - När knappen tryckts in och sedan släppts upp helt, växlar LEDen beteende mellan följande moder, i den angivna sekvensen:
        1. Pulserande, med hjälp av simple_ramp-funktionen från labb 2 (Utgångsläge vid reset/uppstart)
        2. Potentiometerstyrd, med hjälp av deluppgift 2
        3. Blinkande av/på, i ca 1 Hz
        4. Släckt
2. Beteendet implementeras med fördel med hjälp av en tillståndsmaskin
3. Delay-funktioner ska ej användas, ej heller busy-wait på flaggor för timers eller ADC
4. Tänk på att avbrottsrutinen ska exekvera minimalt med kod
5. Städa upp din kod och skapa en git-commit med ovanstående.