Andreas Axelsson
Alexander Göransson
Viktor Mann

JÖNKÖPING UNIVERSITY
School of Engineering

2023-04-26

Enchipsdatorer – Laboration 5
Mät spänning med en ADC

Målsättning

- Konfigurera en ADC för inläsning av ett värde.
- Konfigurera en ADC för inläsning av flera värden, i sekvens.
- Sätta upp en ADC för interrupt-driven användning.
- Skriva kod för att justera ADC-inläsningars värden.

Förberedelse

Läst på föreläsning och kursbok om ADC. Titta igenom kap. 1.1 (inledning) i detta häfte samt svara på frågorna i kap. 2.1.

Examination

Resultat för laborationen presenteras muntligt för en laborationshandledare under laborationstillfället. Frågor i **fet stil** ska besvaras under redovisningen. Krav på kodstil måste följas.

Genomförande

Laborationen har 4 timmar handledd tid, men kan ta mer tid att genomföra, vilket då görs på egen hand.

Det är viktigt att kodningsarbetet sker individuellt. Det är okej att diskutera problemlösning med andra men det är absolut förbjudet att kopiera kod från andra. De lösningar som tas fram skall förses med bra kommentarer, korrekt indentering och bra variabelnamn. Koden ska m.a.o. vara lätt att läsa och förstå.

Hårdvara

- Joystick med kontaktstift
- LM35 temperatursensor (lös eller monterad på pcb)
- Fotoresistor monterad på PCB
- LCD-skärm

Andreas Axelsson
Alexander Göransson
Viktor Mann

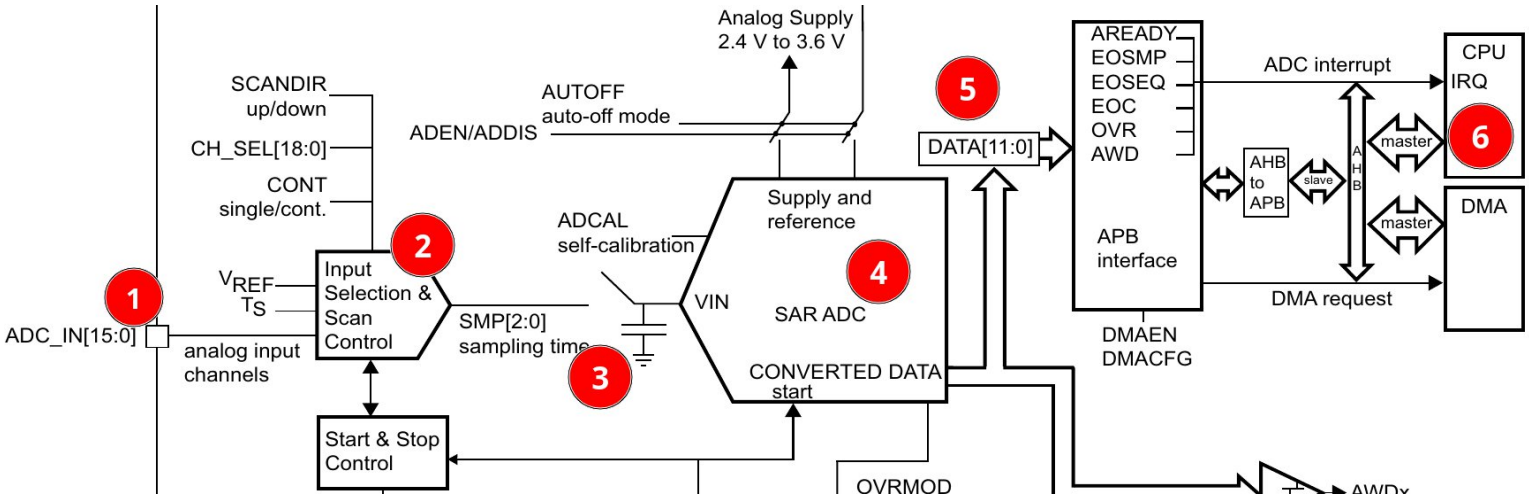
Enchipsdatorer – Laboration 5
ADC

1. Inledning och enkel avläsning

1.1. Konstruktion

Likt hur en timer kan brytas ner till kärnan (PSC, CNT, ARR samt interruptgenerering) och sedan annan funktionalitet runtom, så kan en ADC ses på samma sätt.

Bilden är från referensmanualen för STM32F0306. Den är tydligare för någon ny till ämnet. Själva konvertering sker vid stegen 3-4-5. Det är alltså där en spänning på 0.0V till 3.3V[1] omvandlas till ett tal 0-4095. Laborationen går ut på att konfigurera 1, 2 och 6 rätt.



1. Portarna som analogvärden ska läsas ifrån.
2. En analog-mux. Av alla insignaler ska endast en av dem släppas igenom åt gången.
3. Detta är en kondensator som blir uppladdad till spänningsnivån ADC_INx ligger på. Det sker när switchen i bilden är stängd. Kallas för ”sample and hold”.
4. ADC:n i nästan alla MCU:er är av typen SAR (successive approximation). Det är här som inläsning görs från kondensatorn för att sedan konverteras till ett heltal (12-bitars, i detta fall).
5. Register där det konverterade värdet sparas. Det går att läsa detta direkt om man gör en mjukvarukonvertering.
6. Bara de enklaste av program använder mjukvarukonvertering. Det är mycket effektivare att (likt hur en timer fungerar) starta upp enheten och sedan arbeta med den genom interrupts. Funktionen `HAL_ADC_ConvCpltCallback()` är vad som kommer användas under laborationen.

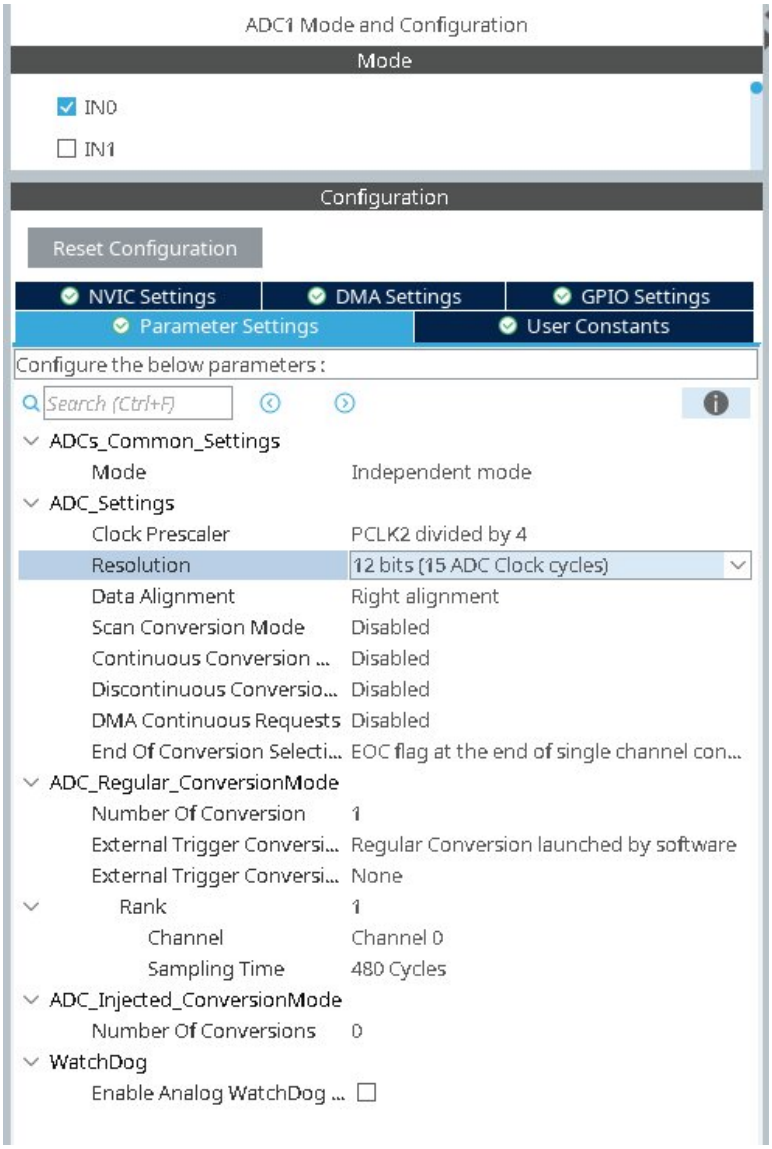
[1] 3.3Vgäller för F401RE-kortet. Kan ha andra värden beroende på spännnigsnivåerna till MCU:n.

Enchipsdatorer – Laboration 5
ADC

1.2.Läsa av ett analog-värde

Om du vill läsa av enbart *ett* värde och avläsningen inte är tidsberoende så fungerar denna metod bra. Inte något för produktionssystem, men för hobbyprojekt går det utmärkt.

Ställ in enheten ADC1 i CubeMX enligt bilden till höger. Det är bara kanalen IN0 som är förbockad i listan upptil.



De värden du läser in under denna laboration ska visas upp på LCD-skärmen som användes under förra laborationen. Dessa saker behöver göras:

- Kopiera `lcd.{c,h}`.
- Aktivera timern för `My_Delay()`.
- Aktivera I2C.
- Välj bra portar för SDA och SCL (om du vill ändra).

När det är gjort så kan du skriva in funktionen inunder i ditt program.

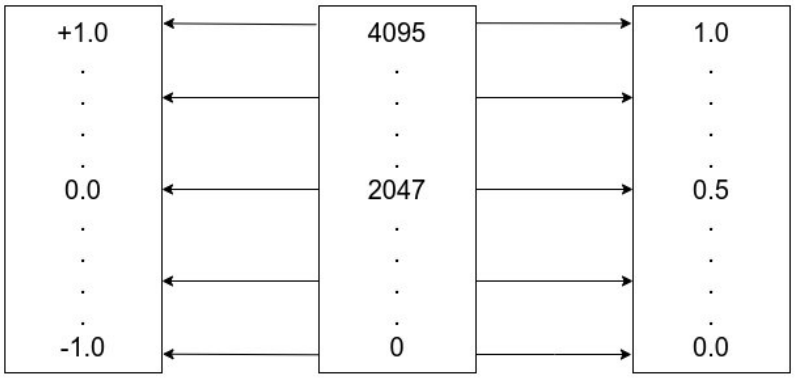
```
uint16_t read_one_adc_value(ADC_HandleTypeDef * hadc)
{
    HAL_ADC_Start(hadc);
    HAL_ADC_PollForConversion(hadc, 100);
    uint32_t reading = HAL_ADC_GetValue(hadc);
    HAL_ADC_Stop(hadc);
    return (uint16_t) reading;
}
```

Börja med att testa denna funktion med joysticken. Det kan vara skrivet +5V som matningsspänning på den, men det är **FEL**. Läs i Appendix (sist i detta dokument) för detaljer.

Kontrollera i CubeMX vilken port som IN0 sattes till och koppla en av joystickens axlar (X eller Y) dit. Porten lär få namnet ADC1_IN0 i CubeMX. Då porten inte används som GPIO Input/Output så kommer en User Label inte att kunna användas i din kod, men det kan vara bra att skriva en, för att ha bättre ordning på din lösning.

1.3.Normalisering och uppvisning

Det inlästa värdet ska normaliseras innan det visas upp på LCD-skärmen.



För analogvärden är det oftast smidigare att hantera dem som flyttal. Konstruera två funktioner som avbildar det inlästa värdet (0 till 4095) mot andra spann. Se bilden.

```
float normalize_12bit(uint16_t x);        // right in image
float normalize_12bit_posneg(uint16_t x); // left in image
```

För att skriva ut strängar med flyttal används enklast `sprintf()`. Denna länk är en väldigt god referens:

https://alvinalexander.com/programming/printf-format-cheat-sheet/

Användning av `sprintf()` tillsammans med flyttal bör generera ett felmeddelande. Men detta meddelande är väldigt snällt och berättar om hur du löser problemet.

När du skriver ut på LCD-skärmen så gör det med tecken (plus/minus), en entalssiffra och två decimaler.

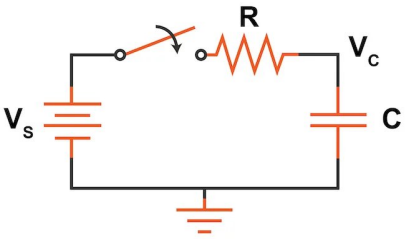
Exempel:

- Siffran 0.6666666666 visas som `+0.67`
- Siffran -0.1 ska vissa som `-0.10`

Joysticken passar sig bäst i spannet -1.0 till 1.0, då blir mitten 0.0. Om siffrorna du får diffar (t.ex kanske du får som mest 0.8 och aldrig 1.0) så gör det ingenting. Detta ska justeras senare i laborationen.

2. Icke-triviala avläsningar

2.1. Datablad och beräkningar



En kondensator tar tid att ladda upp. Sambandet är $\tau = RC$. τ är antalet sekunder det tar för kondensatorn att bli uppladdad till 63.2% av in-spänningen[2]. I bilden är det alltså V_C som ADC-omvandlingen använder sig av.

Detta är en bra artikel som dessutom har svar på de första två frågorna:
https://www.allaboutcircuits.com/tools/resistor-capacitor-time-constant-calculator/

Hur många procent har kondensatorn laddats efter $3 * \tau$? (ofta bra nog)

Efter $5 * \tau$? (hög precision)

Öppna fram MCUns *datablad* och slå upp kapitlet ”12-bit ADC characteristics”. Självklart är sample-and-hold-kondensatorn listad. Men det är även så att själva switchen som sluter kretsen (punkt 3 i bilden i kap. 1.1) har en resistans.

Vad är sample-and-hold-kondensatorns storlek? Värsta fallet är högsta värdet, så använd detta.

Vilken resistans ger switchen?

Givet enbart dessa, vad blir $5 * \tau$?

Hur lång tid som ADC-kretsen samplar anges inte i nanosekunder utan i ADC-klockcykler[3]. För att räkna ut f_{ADC} så behöver du öppna CubeMX. Dels måste du hitta ”Clock Prescaler” bland ADC-inställningarna. Du måste även titta i klockträdet (Clock Configuration) för att hitta frekvensen som f_{ADC} baseras på.

Hur lång är en av ADC:ns klockcykler?
Dvs. $T_{ADC} = 1/f_{ADC}$
(enheten är sekunder per klockcykel)

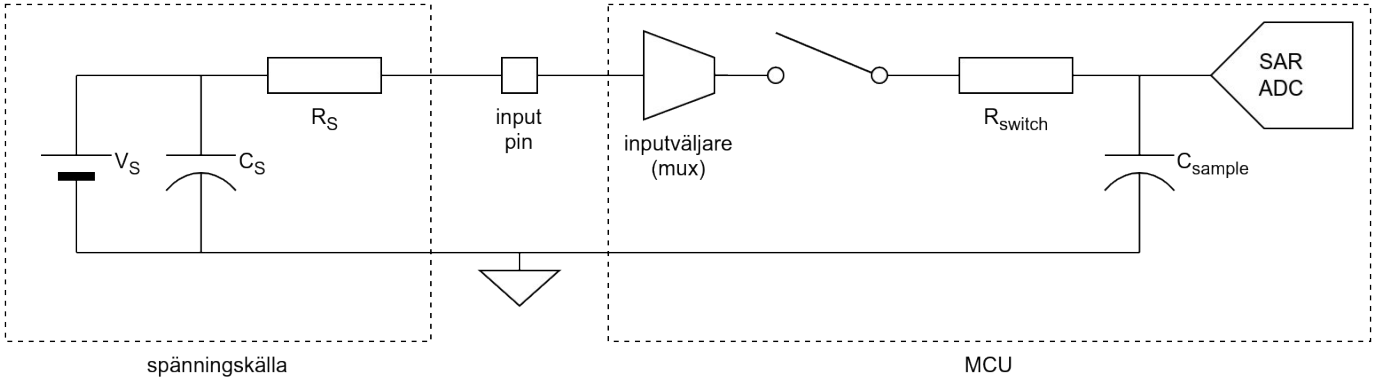
[2] Detta kommer att tas upp mer i kursen Elektriska gränssnitt.

[3] Expandera en av Rank-alternativen i CubeMX för att se vad som kan väljas.

Enchipsdatorer – Laboration 5
ADC

Hur många klockcykler krävs för att hinna med $5 * \tau$? Räkna alltså ut
$n_{SAMP} = {t_{5\tau}}/{T_{ADC}}$

Vilket är det minsta samplingalternativ i CubeMX som krävs för detta?



Något du samplar från är inte okomplicerat. Bilden ovan ger en mer realistisk bild av hur det kan se ut. Till exempel så kan R_S ges av en termistor (temperaturvarierande motstånd) som det finns en spänningsfördelning över. För en termistor är det inte ovanligt att dess resistans kan bli uppåt $20k\Omega$.

Antag att $C_s = 0$. Vad blir $5 * \tau$ när R_s ges av en termistor som den som nämns ovan?

Vad blir n_{SAMP} och vilket värde på samplingstid måste du välja i CubeMX för att garantera en god avläsning?

Vad kan hända om man samplar för kort tid? Tänk ett scenario med flera olika källor som konverteras med kort mellanrum. Vad händer om nr 1 är 0V och nr 2 är 3.3V?

Att sampla i 480 cykler verkar kanske lite mycket men om du använder interrupts för avläsning (vilket görs i denna laboration) och du samplar för kort tid, så kan MCU:n hänga sig. Detta då dina interrupts händer för ofta för att MCUn ska hinna med. Håll samplingstiderna så höga som möjligt om du inte har en väldigt god anledning att sätta dem lågt.

2.2. CubeMX

Du ska nu använda dig av CubeMX för att ställa in ADC-avläsningar så att din ADC-enhet läser av värden och anropar en Callback-funktion varje gång ett värde är färdigkonverterat.

Vill du bättre förstå dig på de möjligheter som finns så kan du läsa här:

https://embedded-lab.com/blog/stm32-adc-2/

Titta tillbaka på grafiken på s. 2. De inställningar du nu ska göra berör steg 2 och 6 i konverteringsprocessen. Steg 2 är den del av ADC:n som väljer vilken av alla insignaler som ska konverteras.

Till att börja med så ska du konfigurera den kö (utgörs av Rank 1 ... Rank n) av konverteringar som ADC:n ska utföra och hur den ska arbeta sig genom denna kö.

För att gå igenom hela köns element i ordning sätt

Scan Conversion Mode → Enabled

Därefter ska du sätta

Continuous Conversion Mode → Enabled

Detta gör att så fort ADC:n är klar med hela sin kö så kommer den att börja om från första elementet automatiskt.

Eftersom vi inte konfigurerar DMA i denna laboration (callbacks/interrupt används istället) så får du själv se till att kopiera värdet som finns i dataregistret efter varje konvertering. Annars skrivs det över av nästa konverterade värde. Sätt

End Of Conversion Selection → EOC flag at end of single channel conversion

Då kommer din callback att köras så fort en konvertering är klar.

Låt Number of Conversion (köns längd) vara 1. Rank 1 (först i kön) ska vara Channel 0 med 480 klockcyklers samplingstid.

Öppna fliken för NVIC och kryssa för så ADC1 ger interrupts. Spara och generera kod.

Skumma igenom kapitlet om specifika portinställningar i Appendix innan du fortsätter.

2.3.Programmera lösningen

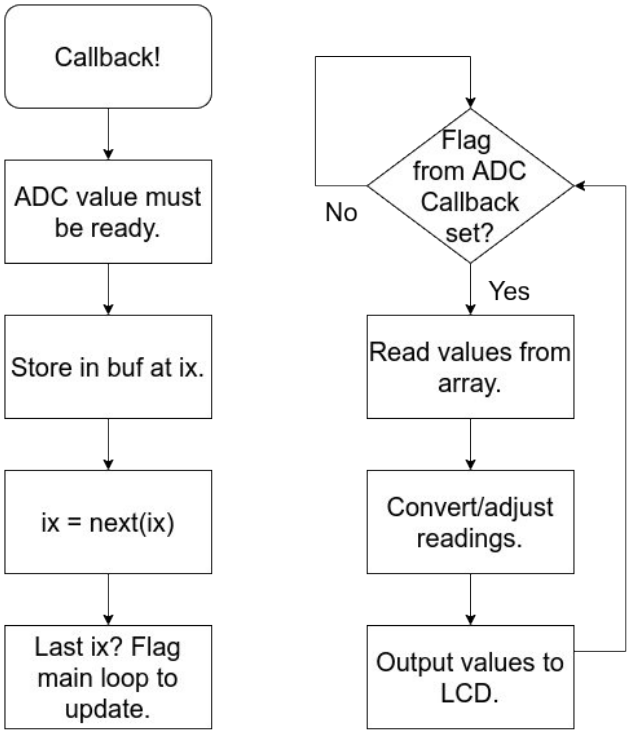
Nu består kön endast av ett element, men det ska bli fler än så. Tänk på det när du skriver din kod. Deklarera en global buffert som kan hålla alla dina konverterade siffror. Deklarera en `#define` intill den för dess storlek. Ha även ett index som representerar vilken plats i kön nästa värde ska skrivas till. Detta blir alltså:

```
#define ADC_BUF_SIZE 1
uint16_t adc_buffer[ADC_BUF_SIZE];
int adc_buf_ix = 0;
```

Funktionerna du behöver för att starta ADC:n, callback, och avläsning av värde finns alla i `JU_STM32_HAL-beskrivning.pdf`.

Din kod ska följa flödesschemat till höger.

Likt hur du ska kontrollera om det är rätt knapp/timer som skapat en interrupt ska du göra samma sak med ADC:n. D.v.s. du ska i din callback kontrollera att `hadc->Instance` är korrekt innan du läser av data.



När du fått den första avläsningen av ett värde att fungera så gå tillbaka till CubeMX:

Bocka för nästa IN-kanal eller aktivera enligt Appendix.

Number Of Conversions → 2

Rank 2 → Channel = Channel 1 (eller vald), Sampling Time = högsta möjliga

Öka `ADC_BUF_SIZE` och visa upp de inlästa värdena på LCDn. Det kan även vara en god idé att använda sig av defines för de olika platserna i bufferten. T.ex:

```
#define JOY_X_IX 0
#define JOY_Y_IX 1
#define LM35_IX  2
```

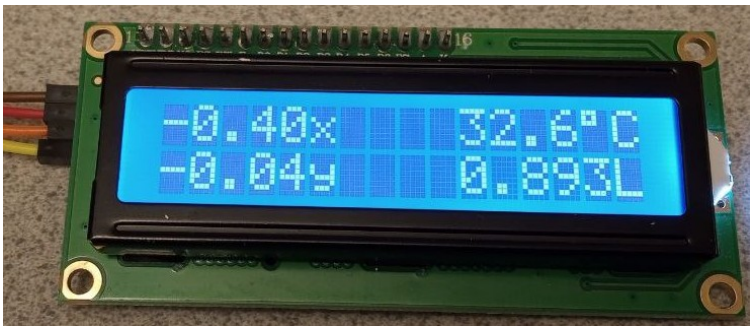
Enchipsdatorer – Laboration 5
ADC

När du lägger till din LM35 (se Appendix) så skriv en likartad funktion:

```
float lm35_to_celsius(uint16_t lm35_reading);
```

Fotoresistorns avläsning ska visas som ett värde mellan 0.000 och 1.000. Ju mer ljus som träffar den desto högre värde ska den visa. För att få bra värden kan potentiometern på komponenten behöva justeras. Se Appendix för detaljer.

När alla fyra avläsningarna kontinuerligt uppdateras på LCD:n så är du färdig!



Bra jobbat!

3. Appendix

3.1. Joystick

Komponenten består av två potentiometrar (variabla motstånd) kopplade på två axlar (upp-ner och höger-vänster). Potentiometrarna är kopplade mellan jord och matningsspänning. Genom att mäta spänningen på mittenstiftet på potentiometern får man ett värde som beror på spakens läge.

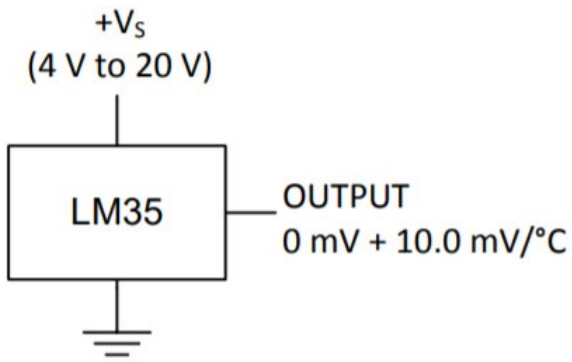
Det finns även en digital knapp som aktiveras när man trycker ner spaken. Den går att läsa av med hjälp av en digital ingång.

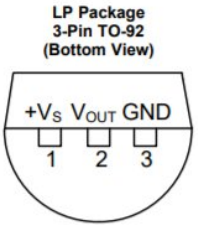
OBS!
Koppla matningsspänning till 3.3V och **INTE** till 5V – oavsett vad kortet säger.
OBS!

3.2. Temperatursensor LM35

LM35 är en temperatursensor som ger en analog signal som är proportionell mot temperaturen. Den ger 0V vid 0°C. Spänningen på utgången ökar sedan med 10mV/°C. T.ex. så ger den spänningen 0.25V vid 25°C.

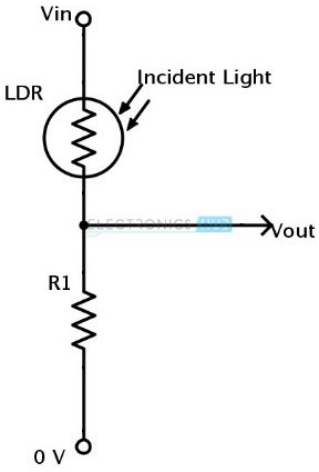
OBS!
Denna komponent **SKA** matas med 5V.
OBS!





Enchipsdatorer – Laboration 5
ADC

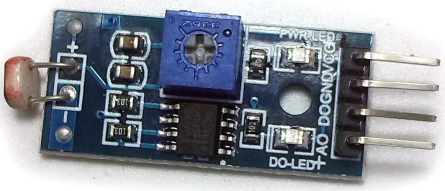
3.3.Fotoresistor

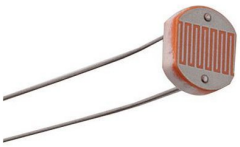


Med hjälp av en fotoresistor kan man mäta mängden ljus som träffar den. Dess resistans varierar med ljusmängden. Genom att koppla den i serie med ett annat motstånd kan man skapa en spänningsfördelning. Spänningen som uppmäts mellan de två komponenterna beror på mängden ljus som träffar fotoresistorn.

Den som kom med erat kit är utrustad med en potentiometer. Motståndet i serie med fotoresistorn kan alltså justeras med en skruvmejsel, om så behövs. Mät från anslutningen märkt A0.

Anslut 3.3V till VCC för denna komponent.





3.4.Välj specifika portar för ADC-läsning

För denna laboration så gör sig dessa GPIO-portar särskilt bra:

PA0, PA1, PA4, PB0, PC1, PC0 (Arduino port A0 till A5).

Om du vill använda någon av dessa portar så klicka på dem och välj alternativen enligt bilderna. Då kommer porten automatiskt att läggas till som analog-ingång i CubeMX.