Løsningsforslag til eksamen i ITD13012 Datateknikk 13.05.2019

Oppgaver vil kunne løses på andre måter enn det som er vist i dette løsningsforslaget.

Oppgave 1. (10%)

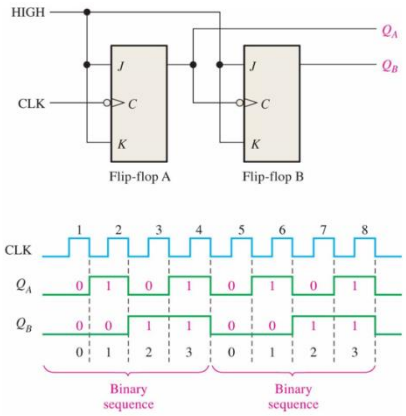
Vipper kan benyttes til;

- Holde en digital verdi, som et slags minne.
- Skiftregister.
- Pulstog med en gitt frekvens. Firkantpulser
- Frekvensdeling
- Tellere
- For å generere en enkelt firkant puls

JK-vippene trigger **på negativ flanke.**
Q_A vil få frekvensen 1000/2= 500Hz
Q_B vil få frekvensen 1000/4 = 250Hz

Hvis vi benytter utgangene Q_A for å angi bit det laveste bit (LSB) og $Q_B$for å angi bit 1, så vil disse to signalene gjennomløpe 4 tilstander; 00, 01, 10, 11.
Det betyr at de kan telle 0, 1, 2 og 3 og så fra start igjen.

Hvis noen tegner en figur så vil det kunne se slik ut;



Oppgave 2. (7%)

Et 8 bit skiftregister vil bestå av 8 vipper. Det kan f.eks være 8 D-vipper.
Hver vippe kan holde 1 bit.
Ved å klokke inn 8 bit på A inngangen styrt av klokkepulsen CLK kan vi lagre 8 bit i skiftregisteret. Vi har følgelig fått inn 8 bit som er på seriell form inn i 8 minner (vipper).
Vi kan nå avlese disse 8 bit-verdiene på utgangene Q_0 til Q_7.
Dermed har vi fått verdien fra seriell form til parallell form.
B må legges høyt for å tillate at vi kan skifte inn verider. Enable funksjon.
Med CLR kan vi slette verdiene som vippene holder hvis det er ønskelig.

Med et slikt skiftregister kan vi følgelig legge en byte (8bit) ut på parallelle utganger.
Disse utgangene Q_0 til Q_7 kan vi benytte som digitale utganger for å styre Led-lys eller andre enheter som kan slås av/på med henholdsvis 0 og 1. Skiftregisteret kan følgelig benyttes i datasystemer for å øke antall digitale utganger på en enkel/rimelig måte.

Oppgave 3. (7%)

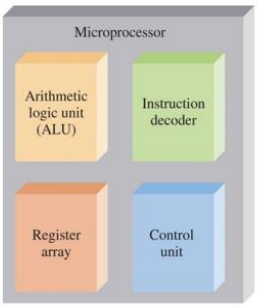
Primærminne
Arbeidsminne. Hurtige kretser nær prosessoren. Det er minne/hukommelse som CPU'en benytter for lagring av program/instruksjoner og data når maskinkode skal utføres/kjøres.
Typisk primærminne er RAM og ROM kretser.
Det å ha mye primærminne i en datamaskin vil kunne påvirke hastigheten på utførelse vesentlig.
Primærminne er mer kostbart enn sekundærminne.

Sekundærminne
Minne for lagring av program og data over tid. Benyttes for å holde større datamengder.
Holder data ved bortfall av «power». → Permanent lagring av kode/data.
Er mer langsomt enn primærminne, men også rimeligere.
Typisk sekundærminne er hard-disker, SSD-disker, Flash-minne, magnetisk minne…

Dynamisk RAM består av 1 transistor + 1 kondensator. Tar liten plass, og kan dermed integreres tett på kretser.
Statisk RAM benytter vipper som hver bygges opp av minst 6 transistorer. Tar dermed mer plass på en krets. Statisk RAM er raskere enn dynamisk RAM.
Statisk RAM er dyrere en dynamisk RAM.

Cache-minne er raske RAM-brikker (statisk RAM) som ligger tett koblet mot CPU'en.
Benyttes for å lagre kode/data som det er størst sannsynlighet for vil bli benyttet i de neste operasjoner. Vil føre til en forbedret ytelse i et datasystem.
Det er flere nivåer av Cache-minne kalt L1, L2, L3. Hvor L1 er Cache-minne bygd inn på selve CPU'en.

Oppgave 4. (8%)



---

1. ALU (Aritmetisk Logisk enhet).
Utfører **aritmetiske** og **logiske** operasjoner på en eller to operander.
Enheten som utfører selve instruksjonene i kode.
Det kan være addering av tall, sammenligning osv…..

2. Register-arrayet
Hurtig minne nært CPU'en for lagring av instruksjoner, data og adresser.
Der finner vi akkumulator-register, instruksjonspeker, hjelpe-registre, stakk-pekere osv…

3. Instruksjons-dekoderen
Behandler instruksjonene som skal utføres.
En **instruksjon** er en **binær kode «Op-code»** som forteller prosessoren hva den skal gjøre.
F.eks addere to tall.
Dekoder instruksjonen og setter opp riktige kontrollsignaler.

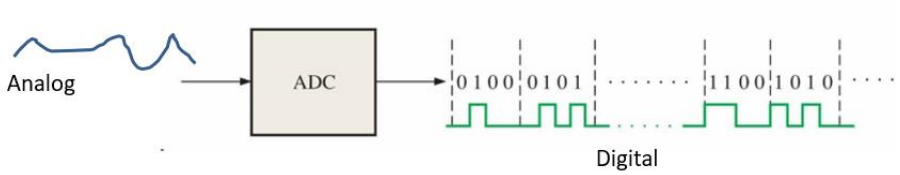
4. Kontroll-enheten
Styrer utførelsen av instruksjoner som er dekodet.
Setter opp **riktige kontrollsignaler**, **tids-styrer** operasjonene.
Håndterer flytting av instruksjoner og data.

En CPU er normalt tilknyttet 3 lokale busser.

Adressebuss.
Benyttes for å angi **lokasjonen** (adressen) i minne hvor data skal leses/skrives til/fra CPU'en.
Enveis kommunikasjonskanal hvor mikroprosessoren legger ut adresser til minne eller porter som det skal leses/skrives data til/fra.
Antall linjer (bit) i databussen angir mulig adresseområde. Med 64 bit kan man adressere 2^{64} lokasjoner.
Databuss.
Instruksjoner (programkode) samt data blir overført via databussen etterhvert som CPU'en trenger dem. Toveis kanal som overfører data/bytes til fra CPU'en.
Databussen består av et antall linjer/kanaler som det kan fraktes bit over.
Mikrokontrollere kan f.eks benytte 8 bit databuss. Nye kraftige CPU'er benytter 64 bit eller mer til databuss.
Kontrollbuss.
Vil bestå av et antall kontrollsignaler.
CPU'en styrer alle operasjoner via **kontrollsignaler** som brukes internt i CPU'en, samt ut på kontrollbussen til minne og IO-systemet.
Benytter signaler for å enable/disable operasjoner mot minne og IO-porter.

Oppgave 5. (10%)

1.
Benyttes for å omforme et analogt signal til en binær verdi som en datamaskin kan benytte.
Det analoge signalet blir samplet og holdt i en krets. ADC'en vil benytte et antall bit for å representere den analoge verdien. Omforming fra analog til binær verdi kan gjøres på flere vis. (Beskrevet nedenfor).
Vi sier at ADC'en kvantiserer den analoge verdien.
Den digitale koden skal være proposjonal med den analoge inngangen.
Antall bit som benyttes vil gi oppløsningen til ADC'en.



Det kan benyttes ulike metoder for ADC-oppgaven.
Den vanligste er **suksessiv approksimasjon.**
Der benyttes en DAC (D/A-omformer) for skrittvis å bestemme hvilken digital verdi som best overensstemmer med en analoge. Godt forklart i notater.

En annen metode er Flash-A/D.
Benytter en rekke komparatorer for å bestemme nivået på det analoge signalet.
Rask metode, men krever mange komparatorer.

2.
Målområdet utgjør 200 grader.
Vi vet at en ADC aldri vil oppnå en bedre nøyaktighet enn +/- 0.5 LSB.
1 LSB skal følgelig være mindre enn **0.02 grader** i dette tilfellet.
Vi må derfor teste med noen bit for å finne hva som vi minimum må benytte.

1. Prøve seg fram
Vi prøver med **12 bit**:
Da vil **1 LSB** være: (Måle-området) / 2^{12} = 200/4096 = 0.044 grader. Det ser vi er for dårlig.
Med **13 bit vil** vi fått: 200/ 8192 = 0.024 grader. **(Får dårlig)**
Tester med **14 bit**: 200/ 16384 = 0.012 grader: (**Mindre enn 0.02 og dermed OK.)**
Det gir oss en nøyaktighet på: +/- 0.5 LSB = 0.006 grader, og oppfyller kravet.
Vi må følgelig benytte minimum 14 bit ADC for å oppnå ønsket nøyaktighet.

2. Matematisk kan vi sette opp kravet slik.

$200/2^n * 0.5 < 0.01 \rightarrow 2^n > 100/0.01 = 10000 \rightarrow 2^n > 10000$

$\log_2(2^n) > \log_2(10000)$

n > $\log_2(10000) = \log_{10}(10000)/\log_{10}(2) =$ **13.28**

n må være mer enn 13 bit, og da må vi benytte minst 14 bit for å oppnå kravet.

Hvis vi måler 15 grader vil vi kunne beregne bit-verdien på følgende vis;
15 grader ligger **65 grader høyere** enn laveste verdi som er – 50 grader.
Området som måles utgjør 200 grader. Og med 10 bit har vi 1024 bitnivåer. (0 → 1023)

bitverdi = (65/200) *1024 = 332 (Svaret blir 332.8, men bitverdien er et **heltall** avrundet mot null)

Alias-frekvenser er frekvenser i et samplet signal som **ikke** skulle vært der.
Dette kalles alias-frekvenser eller falske frekvenser.
Ved sampling vil frekvenser som ligger over halvparten av samplingsfrekvensen **nedfoldes** og dukke opp som aliasfrekvenser.
Dette kan vi unngå enten ved at vi sampler så fort at det ikke er frekvenser over $F_s/2$ før sampling.
Eller så kan vi benytte et analogt **lavpassfilter,** og filtrere bort frekvenser over $F_s/2$ før sampling.

Oppgave 6. (10%)
---
while()-løkka sist i programmet sørger for at loop() bare kjøres 1 gang.
Utskrift til monitoren blir:

Arduino Test har startet: 1
resultat = 0
resultat = 4
verdi = 0.50
Sommer 2019
11
res = 12
bokser = 42

Oppgave 7. (22%)

Generelle kommentarer til denne oppgaven.
Deklarering av variabler og konstanter er litt opp til programmereren.
Kan gjøres på flere måter. Selve kalibreringen skal utføres i setup() i løpet av 5 sekunder.
Benytter millis()-funksjonen for å avlese tid siden start av programmet.

loop() funksjonen kaller opp skaler()-funksjonen.
Den kan ta med seg parametere som er nødvendig. Benyttes globale variabler som jeg gjør i løsningen er det ikke behov for å overføre parametere til skaler().

Selve skaler() funksjonen må settes opp riktig for at sensorområdet skal map'es til gitt frekvensområde. Må passe på at en ikke lager en feil heltallsdivisjon der.

Mulig løsning:

```
// Deklarerer konstanter og variabler.
const int lightSensor = A0;
const int blueLedPin = 4;
int sensorValue = 0;
int sensorHigh = 0;      // Setter den til minimum
int sensorLow = 1023;    // Setter den til maks verdien
int freqLow = 100;       // Laveste frekvens som skal avspilles
int freqHigh = 4000;     // Høyeste frekvens som skal avspilles


// Kalibrerer systemet i setup()
void setup()
{
  pinMode (blueLedPin, OUTPUT);       // Blått Ledlys
  pinMode (8, OUTPUT);     // Lydalarm. Ikke krav at den må angis som output før bruk med tone()
  digitalWrite(blueLedPin, LOW );     // Slår av Blå Led før start av kalibrering

  while ( millis() < 5000)     // sjekker om det har gått 5 sekunder siden start av programmet
  {
    // Må kalibrere maks og min-vedi fra fotomotstand.
    sensorValue = analogRead(lightSensor);
    if ( sensorValue > sensorHigh )
      sensorHigh = sensorValue;
    if (sensorValue < sensorLow)
      sensorLow = sensorValue;
  }

  digitalWrite(blueLedPin, HIGH );    // Slår på Blå Led etter kalibrering
}


void loop()
{
  int frequency = 0;
  sensorValue = analogRead(lightSensor);   // Leser av lys-sensor
  // Kaller skaler-funksjonen. Ettersom jeg benytter glovbale variabler. trenger jeg ikke flere parametre
  frequency = skaler(sensorValue);
  tone(8, frequency, 20);       // avspiller frekvens i 20 ms
  delay(20);              // Legger inn et lite delay tilsvarende varighen til tone()
 }

// Denne funksjonen map'er sensorValue til en frekvens i området freqLow til freqHigh.
// Viktig at den settes riktig opp matematisk. Unngå helltallsdivisjon.
int skaler()
{
  int freq = 0;
  freq =  ( float (sensorValue - sensorLow)/(sensorHigh- sensorLow))
                              *(freqHigh -freqLow) + freqLow;
  return freq;
}
```

Oppgave 8. (26%)

```
/* alarm_system. Oppg 8 våren 2019.
   Innganger:
   2 analoge:   A0 - Temperatur.   A1 - Nivå
   Digital inn: Pinne 4.  Passiv/aktivt alarm-system.
   Digital inn: Pinne 2.  Dør-bryter for innbruddsalarm.
   Utganger: tone() på pinne 8.
                              RR 2019  oppg8_alarmsys.ino    */


// Deklarerer
const int BUTTON_INT = 0;      //Interrupt 0 (pin 2 on the Uno)
const int tempSensor = A0;
const int nivaaSensor = A1;
volatile int aktivSystemSwitch = 0; // Bør være volatile pga av bruk i interrupt-rutina


// Initierer
void setup()
{
  pinMode (4, INPUT);       // Passivt eller aktivt alarmsystem. Benytter bryter
  pinMode (2, INPUT);       // Dør-bryter, samt start av interrupt-rutina.
  pinMode (8, OUTPUT);      // Lydalarm med tone.
                            // Ikke krav at den må angis som output før bruk med tone()
  Serial.begin(9600);       // Klargjør seriell kommunikasjon til monitoren
  attachInterrupt(BUTTON_INT, alarm, RISING); // setter opp interrupt-rutina. Viktig.
}


void loop()
{
  float temperatur = 0.0;
  float vannNivaa = 0.0;
  int doorSwitch = 0;
  int tempBitverdi = 0;
  int vannNivaaBitverdi = 0;

  // Leser av digitale innganger
  doorSwitch = digitalRead(2);          // Leser av om dør til kabinen er åpen eller lukket
  aktivSystemSwitch = digitalRead(4);   // Leser av om alarmsystemet er aktivert eller
                                        // deaktivert.

  // Hvis lyd-alarmen går kan vi selv slå av lyden ved å slå aktivSystembryter av.
  // Stopper lyd med noTone().
  if (aktivSystemSwitch == 0 )
     noTone(8);             // slår av eventuell alarmlyd
```

```
  // Leser av temperatursensor, skalerer og skriver til monitor
  tempBitverdi = analogRead(tempSensor);
  temperatur = (tempBitverdi/1024.0)*150.0-50.0; // Hele området utgjør 150 grader.
  Serial.print("Temperaturen er: ");
  Serial.print(temperatur);
  Serial.println(" grader Celcius");

  // Leser av nivåsensor, skalerer og skriver til monitor
  vannNivaaBitverdi = analogRead(nivaaSensor);
  vannNivaa = (vannNivaaBitverdi/1024.0)*20.0;   // Hele området utgjør 20 cm
  Serial.print("Vann-nivaaet er: ");
  Serial.print(vannNivaa);
  Serial.println(" cm");

  // Sjekker om alarmsystemet er aktivert ved å sjekke aktivSystemSwitch
  if (aktivSystemSwitch == 0 )
    Serial.println("Alarmsystemet er Deaktivert.");
   else
    Serial.println("Alarmsystemet er Aktivert.");

  // Utskrtift om kabindør er lukket eller åpen.
  if (doorSwitch == 0 )
    Serial.println("Kabindoera er lukket.");
   else
    Serial.println("Kabindoera er aapen.");

  delay(100);      // Kan legge inn en delay() her , men det er ikke krav i oppgaven
 }



// Interrupt rutine som blir utført hvis kabindøra går fra lukket til åpen.
void alarm(void)
{
  if ( aktivSystemSwitch )   // Bruker gsm og alarm kun når systemet er aktivt.
  {
    tone(8,1000,300000);     // Slår på alarm-tone. 1000Hz i 5 minutter (300000 ms)
    gsmVarsel();             // kaller opp gsmVarsel-rutina
 }
}



void gsmVarsel(void)
{
  Serial.println("");
  Serial.println("Sender tekstmelding til eier !!!!! ");
}
```