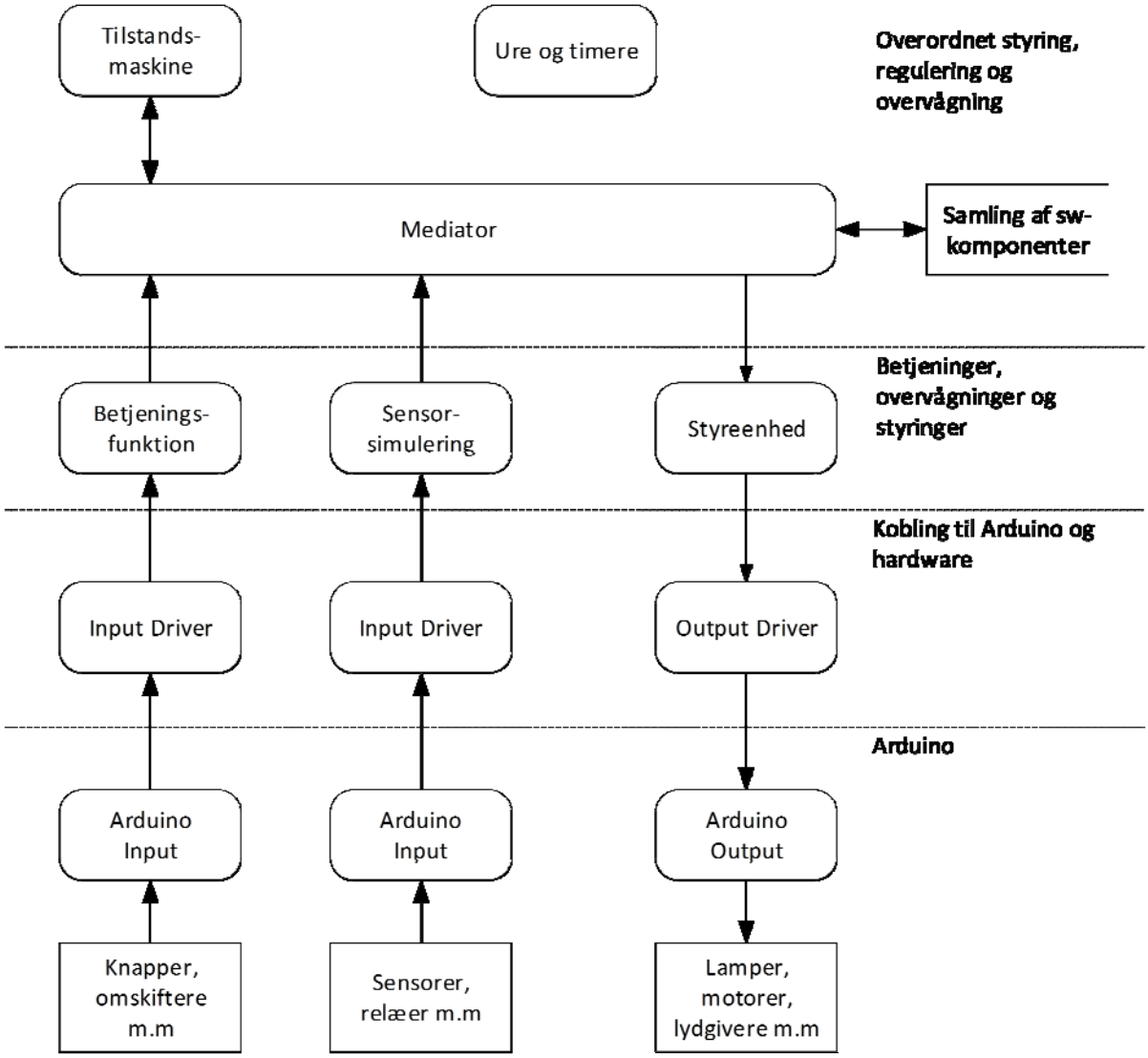
Arduino applikation

7. januar 2023 21:34

Nogle gange er der behov for at kode en simpel Arduino applikation på op til ca. 50 programlinjer. Den kan bruge procedural programmering.

De store applikationer bliver bygget med softwarekomponenter. Det gør applikationen fleksibel og giver mulighed for at opdatere i flere cyklusser efterhånden som kravene bliver afklaret. Softwarekomponenter er bygget færdig og uden behov for modifikation. Nye komponenter kan blive tilføjet.

Den store applikation er bygget således:



Software består af komponenter, hver med deres grænseflade og specifikke funktion.

- Arduino er koblet til betjeninger, sensorer, lamper, motorer m.m. Det er hardwarelaget.
- Drivere konfigurerer Arduino parallelle porte og seriel kommunikation. En driver kan udskiftes med en anden driver, uden det giver behov for at opdatere applikationen. Dermed bliver applikationen uafhængig af tilkoblingen til Arduino.
- Applikationen indeholder funktioner for betjeninger og sensorer. I det lag kan der også indbygges behandling af inputdata: Beregninger, binær logik, lagring af data til senere brug osv.
- Til styringer indeholder applikationen styreenheder for lamper, motorer m.m. I det lag kan der også indbygges behandling af outputdata: Beregninger, binær logik, blink til lamper, styring af

motor osv.

- Tilstandsmaskinen udfører enhver tilstand: Ændring af tilstand som følge af inputdata. Opdatering af output, både når der bliver skiftet til en tilstand og når en tilstand forlades. En ny tilstand kan også udløses når en tid er udløbet.
- Mediator formidler kommunikation imellem softwarekomponenter. Mediator bruger en samling af softwarekomponenter. Mediator udfører også tidsstyring i betjening, sensorer og styreenheder.

Hvis softwarekomponenter udfører direkte funktionskald, skal komponenten tilpasses de komponenter softwaren består af. Men så er princippet, at komponenter er færdigbygget brudt. Alle softwarekomponenter bliver lagt i en samling, hvor de får en adresse. Dataoverførsel formidles via adresser imellem softwarekomponenterne af mediator. Opsætning af samlingen af softwarekomponenter sker ved opstart af Arduino.

En konkret applikation skal have sit eget design dokument, der forklarer hvordan applikationen er bygget og hvorfor den er bygget på den måde.

Arduino tilpasning

7. januar 2023 21:41

Arduino Uno har ikke et operativsystem. Program skal levere al maskinkode. Der er begrænset memory til data 2kbyte og 32kb til program.

Memory allokering og deallokering kan udvide brug af memory, men indfører risiko for at fylde memory op med døde data. I denne model er alt memory lagt fast på compile tidspunktet.
Når der kun afsættes statisk memory får det som konsekvens:

- Alle objekter bliver instantieret.
- Antal elementer i et array bliver lagt fast.

Normalt håndteres tryk på en knap og udløb af en timer med interrupt. Det er en kompleks mekanisme, der skal programmeres og testes omhyggeligt, for at forebygge runtime fejl. Istedet for bliver der brugt en model med polling. Tidsinterval for polling bliver så kort, at f.eks. et knaptryk bliver besvaret hurtigt nok. Software moduler med tidstyring skal indeholde metoden doClockCycle.

Fejlhåndtering med exceptions og throw er også en kompleks mekanisme, der så vidt vides ikke understøttes af arduino. Softwarefejl må forebygges med grundig testning.

Grænseflader - softwarelag

5. februar 2023 14:41

Mellem hver af de 3 lag software er der en grænseflade, med metodenavne og dataprotokol.

En softwarekomponent har pointere til de softwarekomponenter den skal kommunikere med. Det gør det muligt at udpege en anden softwarekomponent, i stedet for at ombygge.

Softwarekomponenter skal have fast indbyggede navne for metoder, så metodenavne ikke skal tilpasses, hvilket bryder princippet, at komponenter er færdigbygget.

Softwarekomponenter skal have fast indbyggede dataprotokoller, så de ikke skal tilpasses, hvilket bryder princippet, at komponenter er færdigbygget.

- Digitale drivere - grænseflade.
  Datatype er bool.
  Bruger hardware begreber: LOW og HIGH.
- Analoge drivere - grænseflade.
  Datatype er int.
- Digitale funktioner, multivibratorer og lignende - grænseflade.
  Datatype er bool.
  Bruger funktions begreber: OFF og ON.
- Multivibratorer, timere og lignende - grænseflade.
  Datatype er bool.
  Bruger funktions begreber: OFF og ON.
- Betjeninger, sensorer og styreenheder - - grænseflade.
  Datatype er byte.
  Bruger status begreber: OFF, ON, FREE, OCCUPIED, UP, DOWN o.s.v. Der er mulighed for 256 tilstande.

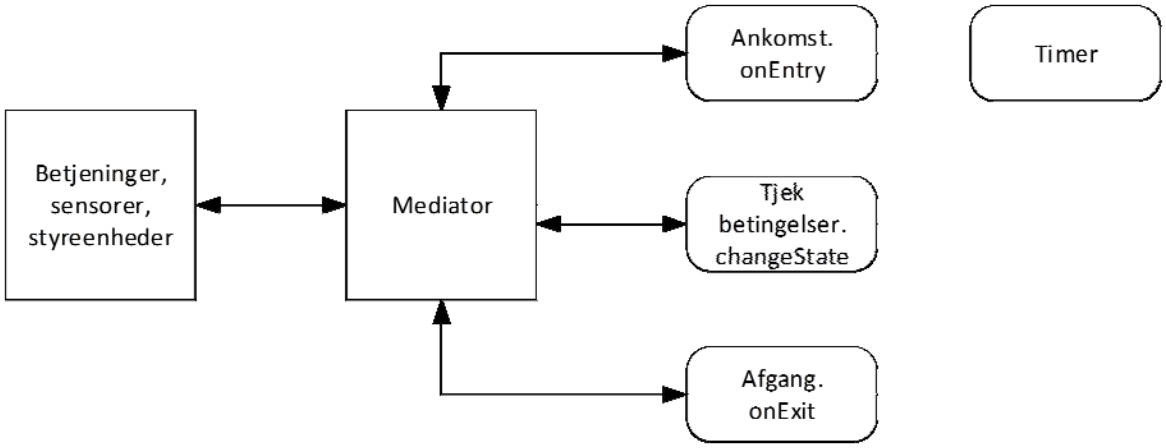
Der bliver brugt positiv logik i applikationen. Så aktiv er for eksempel ON = 1.

Tilstandsmaskine

12. januar 2023 22:23

Dette design for en tilstandsmaskine, gør programmering af hver tilstand enkel og overskuelig. Alt i alt kan tilstandsmaskinen blive meget kompleks også meget mere kompleks end "Switch" "Case" konstruktionen giver mulighed for.

Der ligger en skabelon til tilstandsmaskine.



Der skal programmeres en subklasse per tilstand.

Mediator udpeger den aktuelle tilstand.
I hver klokkecyklus kalder mediator den aktuelle tilstand, som melder tilbage om betingelser for skift af tilstand er opfyldt.

Mediator udfører skift af tilstand igennem 2 klokkecyklus.

1. Tilstandsmaskinen melder skift til næste tilstand og udpeger næste tilstand.
    a. For den tilstand der skal forlades, bliver onExit metoden kaldt, så tilstandens afslutning bliver udført.
    b. Mediator udpeger næste tilstand.
2. Mediator starter næste tilstand .
    a. For den nye aktuelle tilstand, bliver onEntry metoden kaldt, så tilstandens start bliver udført.
    b. Kalder den nye aktuelle tilstand, som melder tilbage om betingelser for skift af tilstand er opfyldt.

onEntry:
    Kan bruges til at starte en timer med udløb for tilstanden.
    Betjeningsenheder og sensorer kan blive reset.
    Styreenheder kan blive opdateret.
changeState:
    Læser status fra betjeningsenheder og sensorer.
    Tjekker for udløb af timer.
    Tjekker om betingelser for transition er opfyldt.
    Melder tilbage om skift til næste tilstand.
    Returnerer næste tilstand til applikationen.
onExit:
    Betjeningsenheder og sensorer kan blive reset.
    Styreenheder kan blive opdateret.

Til en applikation bliver der gennemført en tilstandsanalyse, der bliver dokumenteret i et tilstandsdiagram. Tilstandsdiagram viser:

- Hvilke betjeninger og sensorer der giver anledning til skift af tilstand.
- Hvilke styreenheder der skal opdateres og deres nye tilstand.
- Hvilke betjeninger og sensorer, der skal resettes.
- Om opdatering skal ske ved ankomst eller afgang fra en tilstand.

Normalt bliver styreenheder opdateret ved ankomst i en tilstand. Er der en forgreninger af tilstande efter en tilstand, hvor styreenheder skal have de samme opdateringer, kan de opdateringer med fordel udføres i afgang fra tilstand. Krav til applikationens funktion kan også give anledning til at placere opdatering i afgang fra en tilstand.

Mediator

12. januar 2023 22:23

Når applikationen bliver programmeret, skabes også en mediator.

En mediator varetager al kommunikation mellem komponenterne for betjeninger, sensorer, styreenheder og tilstandsmaskine. Det giver mulighed for at tilføje nye betjeninger, sensorer, styreenheder og tilstande, så fleksibelt at det ikke trækker en række følgerettelser med sig.

Der ligger en skabelon for mediator, der i hovedtræk fungerer således:

- Alle betjeninger er samlet i et array[], som får tildelt memory på compile time. Indeks i array er en simpel enum. Hver betjening har en fast plads, som gør opslag i samlingen enkel.
- Alle sensorer er samlet i et array[], som får tildelt memory på compile time. Indeks i array er en simpel enum. Hver sensor har en fast plads, som gør opslag i samlingen enkel.
- Alle styreenheder er samlet i et array[], som får tildelt memory på compile time. Indeks i array er en simpel enum. Hver styreenheder har en fast plads, som gør opslag i samlingen enkel.
- Alle applikationens tilstande er samlet i et array[], som får tildelt memory på compile time. Indeks i array er en simpel enum.. Hver tilstand har en fast plads, som gør opslag i samlingen enkel.

Mediator er designet ud fra softwarepattern mediator.

Eksempel mediator

12. februar 2023 17:54

En mediator er passet til den konkrete applikation. Der er et eksempel herunder.

```
// Ansvar: Varetager kommunikation mellem betjeninger, sensorer, styrede enheder og tilstandsmaskine.
// Designet gør det muligt at koble forskellige typer af komponenter sammen, så forskellige overkørsler kan modelleres.
// stateNo: Nuværende tilstand
// entryState: Hver gang der skiftes en tilstand skal indgangsmetoden kaldes. Det holder variablen styr på.
// manuals[]: Vektor med pointere til alle betjeninger
// sensor: Pointer til sensor
// ctrlUnits[]: Vektor med pointere til alle styreenheder
// states[]: Vektor med pointere til alle tilstande
// begin(...): Initialiserer den første tilstand, som applikationen skal starte med.
// set...(...): Konfigurerer mediator.
// doClockCycle(...): Sørger for at alle tilkoblede komponenter udfører polling.
// Desuden varetager metoden styring af overkørslens tilstand.
// status(...): Er en service til et tilstandsobjekt, som leverer en betjeningsenhed eller sensorenheds status.
// reset(...): Er en service til et tilstandsobjekt, som kan resette en betjeningsenhed eller sensorenhed.
// to(...): Er en service til et tilstandsobjekt, som kan sende en besked til en ydre enhed.
class t_Mediator {
private:
  byte stateNo;
  bool entryState;
  t_Manual *manuals[MaxNoManuals];
  t_Sensor *sensors[MaxNoSensors];
  t_CtrlUnit *ctrlUnits[MaxNoCtrlUnits];
  t_StateMachine *states[MaxNoStates];
public:
  t_Mediator(void) {}
  void begin(byte stateName) {stateNo = stateName; entryState = true;}
  void setManual(byte manualName, t_Manual *manual) {manuals[manualName] = manual;}
  void setSensor(byte sensorName, t_Sensor *sensor) {sensors[sensorName] = sensor;}
  void setCtrlUnit(byte ctrlUnitName, t_CtrlUnit *ctrlUnit) {ctrlUnits[ctrlUnitName] = ctrlUnit;}
  void setState(byte stateName, t_StateMachine *state) {states[stateName] = state;}
  void doClockCycle(void);
  byte statusManual(byte manualName) {return manuals[manualName]->status();}
  byte statusSensor(byte sensorName) {return sensors[sensorName]->status();}
  void to(byte ctrlUnitName, byte ctrlUnitState) {ctrlUnits[ctrlUnitName]->to(ctrlUnitState);}
};

void t_Mediator::doClockCycle(void) {
  byte cnt;  // Loop tæller
  byte nextStateNo;
  for (cnt=0; cnt < MaxNoManuals; cnt++) manuals[cnt]->doClockCycle();
  for (cnt=0; cnt < MaxNoSensors; cnt++) sensors[cnt]->doClockCycle();
  if (entryState == true) {
    states[stateNo]->onEntry();
    entryState = false;
  }
  if (states[stateNo]->changeState(&nextStateNo) == true) {
    states[stateNo]->onExit();
    stateNo = nextStateNo;
    entryState = true;
  }
}
```

Arduino hovedprogram

7. januar 2023 11:05

Hovedprogrammet specificerer data, starter applikationen og udfører applikation.

Opsætning
Det er til at blive forvirret af, hvad der kan udføres af opsætning af parametre og opstart af software i en Arduino kørsel. Følgende beskrivelse er baseret på egne erfaringer og efter gennemsyn af andres kode.

Global opsætning
Den globale erklæres ligger først i hovedprogram.

- Indsættelse af header-filer.
- Erklæring af konstanter og enum.
- Erkæring af globale variable.
- Erklæring af strukturer.
- Erklæring af klasser og deres objekter. Objekters konstante medlemmer bliver initialiseret.

Setup()
I setup rutine begynder Arduino at afvikle maskinkode. Her kan sættes op:

1. Intern opsætning af objekt.
2. Forbindelse af objekter med hinanden.
3. Udfør kode, der starter hele applikationen.

Kørsel
Loop()
Tidsstyring bliver udført med polling. Databehandling i softwarekomponenter bliver udført i en pollingcyklus.

1. Kalder pendulum, der er polling uret.
2. Kalder masterblinker.
3. Kalder globale multivibratorer og timere, der er specifikke for applikationen.
4. Kalder input-drivere der har timere. Hver input-protokol får sin egen loop.
5. Kalder mediator, der koordinerer styring, regulering og overvågning.

Der er mulighed for at indbygge andre funktioner til applikationen, som så sættes op globalt, initaliseres i setup() og køres i loop().