# Overkørsel st. enkeltsporet strækning

| Type | Vejledning | Oprettet | 19-05-2021 |
|---|---|---|---|
| Forfatter | Jan Birch | Rettet | 20-05-2021 |
| Noter: | | | |

## Sådan bygger man en overkørsel

Denne vejledning viser hvordan man programmerer en styring til en konkret overkørsel.
Hardware bliver kun nævnt, den skal bygges specifikt til overkørslen.

Til arduino oprettes en folder med et ".ino" program.

Et program består af:

- Global opsætning til overkørsel.
- Indsættelse af model fra bibliotek.
- Specifikation af porte til arduino.
- Specifikation af hardwaredrivere.
- Specifikation af sensor og betjeningsenheder.
- Specifikation ydre enheder.
- Programmering af tilstandsmaskine.
- Sammenkobling af objekter i setup().
- Overkørslens start i setup().
- Kørsel af program i loop().

## Global opsætning

### Globale specifikationer

Den globale opsætning er en række konstanter, som bruges af software komponenter.

```
// Tidsenhed til urværk
enum {MSEC, SECONDS};
// Kontakttyper for trykknap
enum {NOPEN, NCLOSED};
// Overkørslens betjeningsenheder magasin kan konfigureres
enum {BISTABLE, ONESHOT};
// Overkørslens betjeningsenheder leverer
enum {OFF, ON};
// Overkørslens ydre enheder kan blive sat til
enum {BLOCK, PASS};
```

Denne blok af sætninger er obligatorisk og skal indsættes uændret. Til programmering har man konstanternes navne som en huskeliste.

```
// Betjenings- og sensorenheder
const byte MaxNoCtrls = 1;
enum {MANUELBETJ};
// Overkørslens ydre enheder
const byte MaxNoDevices = 4;
enum {BANESIGNAB, BANESIGNBA, VEJSIGNAL, VEJKLOKKE};
// Overkørslens tilstande
const byte MaxNoStates = 5;
enum {IKKESIKRET, FORRING, SIKRET, OPLOES, BILISTTID};
```

MaxNoCtrls, MaxNoDevices og MaxNoStates skal sættes til korrekt antal, for at få allokeret memory korrekt. Navnene på disse konstanter er obligatoriske.

Navnene på komponenterne specificeres til den konkret overkørsel. I blokken ovenover er vist navne som eksempel.

### Indsætte model

Model indsættes simpelt med en inklude sætning.

```
// Overkørslens moduler
#include <Ovkoersel.h>
```

### Specifikation af porte

Herunder er der et eksempel på specifikation af porte til arduino.

```
// Arduino pins
struct {
  const byte ManuelKnap = 2;
  const byte OUSignAB = 7;
  const byte OUSignBA = 8;
  const byte VejKlokker = 10;
  const byte VejLys = 11;
} ARDPin;
```

Det er valgfrit at bruge enum i stedet for. Eventuelt kan pin indsættes direkte i opsætning af driver, men det anbefales ikke, fordi det slører formålet med en pin.

Pin opsætning kan kopieres til et hardware testprogram, som tester for korrekt tilkobling af hardware til arduino.

### Specifikation af hardwaredrivere

Hardwaredrivere er objekter, der kobler til arduino porte ved instantiering.

For eksempel:

```
// Hardware drivere til den overkørsel, som dette program leverer
t_PushButton manuelKnapDrv(ARDPin.ManuelKnap, NCLOSED);
t_SimpleOnOff vejLysDrv(ARDPin.VejLys);
```

#### Knap og sensor

```
t_PushButton <driverobjekt>(<Arduino-pin>, <Kontakttype>);
```

Kontakttype:

- NOPEN: Arduino port er forbundet til stel og bliver aktiv, når en kontakt slutter til 5V.
- NCLOSED: Arduino port er forbundet til 5V og bliver aktiv når en kontakt slutter til stel. Typisk bruges den interne pullup modstand.

#### LED eller buzzer

```
t_SimpleOnOff <driverobjekt>(<Arduino-pin>, <Start tilstand>);
eller
t_SimpleOnOff <driverobjekt>(<Arduino-pin>);
```

Start tilstand: HIGH eller LOW. Default er LOW.

### Specifikation af sensor, betjeningsenhed og flipflop

Sensor og betjeningsenhed, der er en af overkørslens komponenter.

For eksempel:

```
// Overkørslens betjeningsenheder
t_CrossingCtrl manuelBetj;
// Flipflop til manuelknap
t_FlipFlop manFF(NCLOSED);

t_CrossingCtrl <enhedsobjekt>;
```

Brug af en flipflop er valgfrit, den bruges hvis input skal magasineres til senere brug af tilstandsmaskine.

```
t_FlipFlop <flipflopobjekt>(<Kontakttype>);
```

Kontakttype skal passe med den tilsluttede driver, for at få sat flipflop i korrekt start tilstand.
Man kan vælge at kombinere for eksempel NCLOSED – NOPEN, hvis negeret logik ønskes anvendt.

### Specifikation ydre enhed

En ydre enhed, er en af overkørslens komponenter.

```
For ekspempel:
// Overkørslens betjenings- og ydre enheder
t_RailSignal OUSignAB;
t_RoadSignal vejKlokker(PASS);
```

t_RailSignal <enhedsobjekt>;
eller
t_RailSignal <enhedsobjekt>(<Start tilstand>);
Start tilstand: PASS eller BLOCK. Default er BLOCK.

t_RoadSignal <enhedsobjekt>;
eller
t_RoadSignal <enhedsobjekt>(<Start tilstand>);
Start tilstand: PASS eller BLOCK. Default er BLOCK.

## Programmering af tilstandsmaskine

Tilstandsmaskine skal programmeres til den specifikke overkørsel. Programmeringen er gjort så enkel som muligt.
Der er et internt objekt i model ”crossing”, som kører tilstandsmaskine og som tilstandsmaskine kommunikerer med.
En tilstand har kun ansvar for sig selv, fra opstart til transition. En tilstand forbereder ikke næste tilstand, men overdrager kun navnet på næste tilstand til ”crossing”.

For eksempel:

```
class t_BillisttidState: public t_StateMachine {
public:
  t_BillisttidState(void) : t_StateMachine() {}
  void onEntry(void) {
    const unsigned long defer = 10;
    clockWork.setDuration(defer, SECONDS);
```

```
  }
  byte doCondition(byte currentStateNo) {
    byte nextState = currentStateNo;
    if (clockWork.triggered() == true) nextState = IKKESIKRET;
    return nextState;
  }
} billisttid;
```

### Kommunikation med overkørsels objekt

Tilstandsmaskinen kommunikerer med overkørslens objekt med føgende metoder:

```
crossing.status(<navn på betjeningsenhed eller sensor>);
```

Metoden leverer status på en betjeningsenhed eller en sensor: ON eller OFF.

```
crossing.reset(<navn på betjeningsenhed eller sensor>);
```

Metoden resetter magasin for betjeningsenhed eller en sensor. Det vil sige, når der er tilsluttet en flipflop.

```
crossing.to(<navn på ydre enhed>, <ny ydre tilstand>);
```

Metoden sætter en ydre enhed i en ny tilstand: BLOCK eller PASS.

I en klokcyklus bliver beregning for transition udført. Ved transition bliver tilstand afsluttet i samme klokcyklus. Efter transition huskes næste tilstand til næste klokcyklus.

### Skabelon for en tilstand

```
class t_<type navn>: public t_StateMachine {
public:
  t_<type navn>(void) : t_StateMachine() {}
  void onEntry(void) {
    const unsigned long defer = <udløbstid i sekunder>;
    crossing.to(<navn på ydre enhed>, <ny ydre tilstand>);
    clockWork.setDuration(defer, SECONDS);
  }
  byte doCondition(byte currentStateNo) {
    byte nextState = currentStateNo;
    if (clockWork.triggered() == true) nextState = <næste tilstand>;
    if (crossing.status(<navn på betj. eller sensorenhed>) == ON)
      nextState = <næste tilstand>;
    return nextState;
  }
  void onExit(void) {
    crossing.to(<navn på ydre enhed>, <ny ydre tilstand>);
    crossing.reset(<navn på betj. eller sensorenhed>);
  }
} <tilstandsobjekt>;
```

### Metoden onEntry

Metoden kan bruges til omskiftning af overkørslens enheder og til start af timer.
Metoden kaldes 1 gang ved opstart af tilstand.
Udelades metoden, forbliver overkørslen uændret ved start af tilstand.

Er der brug for at udløse transition på tid, bruges tilstandsmaskinens indbyggede timer. Det er vist i skabelonen. Timer kan angives i millisekunder eller sekunder: MSEC, SECONDS.

Omskiftning af overkørslens enheder programmeres specifikt til denne tilstand.

### Metoden doCondition

Metoden tjekker om betingelse for transition er opfyldt.
Metoden kaldes i hver klokcyklus.
Metoden er obligatorisk, fordi enhver tilstand skal foretage transition til næste tilstand.
Metoden returnerer navn på næste tilstand, når der skal ske en transition.
Beregning af betingelser for transition programmeres specifikt til denne tilstand. Beregningen kan være simpel, den kan også være kompleks. Der kan angives flere forskellige næste tilstand.

### Metoden onExit

Metoden kan bruges til omskiftning af overkørslens enheder.
Metoden kaldes 1 gang ved transition af tilstand.
Udelades metoden, forbliver overkørslen uændret ved afslutning af tilstand.

Omskiftning af overkørslens enheder programmeres specifikt til denne tilstand.

## Setup()

### Sammenkobling af objekter

Objekter bliver koblet sammen med pointere. De kan kun sættes op, når arduino program kører, det vil sige i setup().

Adresse operator ”&” skal bruges i alle argumentlister.

### Tilkobling af drivere og flipflop

Drivere kobles til overkørslens enheder. For eksempel.

```
// Drivere kobles til betjenings- og ydre enheder
  manuelBetj.setDriver(&manuelKnapDrv);
  OUSignAB.setDriver(&OUSignABDrv);
```

<enhedsobjekt>.setDriver(&<driverobjekt>);

En flipflop er valgfri. For eksempel.

```
// Konfiguration af manuelbetjening
  manuelBetj.setFlipFlop(&manFF);

<enhedsobjekt>.setFlipFlop(&<flipflopobjekt>);
Eller
<enhedsobjekt>.setFlipFlop(&<flipflopobjekt>, <Flipfloptype>);
```

Flipfloptype:

- BISTABLE: Hver gang en driver sender en impuls, skifter flipflop tilstand.
- ONESHOT: Flipflop skifter kun tilstand, første gang den modtager en impuls.

BISTABLE er default.

En hvid lanterne i et banesignal er valgfri. For eksempel.

```
// Konfiguration af banesignal
  OUSignAB.setWhiteLamp(&OUSignABWhiteDrv);

<enhedsobjekt>.setWhiteLamp(&<driverobjekt>);
```

### Opsætning af overkørslens enheder

Overkørslens enheder skal indsættes i en samling, som er center for kommunikation.

For eksempel.

```
// Opsætning af overkørsel
  collection.initialize();
  crossing.setCtrl(MANUELBETJ, &manuelBetj);
  crossing.setDevice(VEJSIGNAL, &vejLys);

collection.initialize();
```

Denne opsætning er obligatorisk og indgår i softwarens kontrol for korrekt opsætning.

```
crossing.setCtrl(<navn på betj. eller sensor>, &<enhedsobjekt>);

crossing.setDevice(<navn på ydre enhed>, &<enhedsobjekt>);
```

### Opsætning af tilstandsmaskine

Overkørslens tilstande skal indsættes i en samling, som er center for kommunikation.

For eksempel.

```
// Opsætning af tilstandsmaskine
  crossing.setState(IKKESIKRET, &ikkesikret);
  crossing.setState(FORRING, &forring);

crossing.setState(<navn på tilstand>, <tilstandsobjekt>);
```

### Overkørsel i starttilstand

Opsætning af overkørsel slutter med at sætte start tilstand.

For eksempel.

```
// Start tilstandsmaskine
  crossing.initState(IKKESIKRET);

crossing.initState(<navn på tilstand>);
```

Hvis drivere og overkørslens enheder er instantieret til overkørslens starttilstand, er der ikke behov for flere opsætninger og overkørslen begynder at køre i loop().

## Loop()

Hovedprogrammet skal programmes til.

```
  Clock::pendulum();
  Blinker::doClockCycle();
  crossing.doClockCycle();
```