Namn:
Laborationen godkänd:

Digitala system 15 hp

LUNDS TEKNISKA HÖGSKOLA
Lunds universitet
LTH Ingenjörshögskolan vid Campus Helsingborg

AVR 4 datorteknik

Tillståndsmaskin.

Syftet med den här laborationen är att förbättra knappavläsningen samt få förståelse till hur tillståndsmaskiner används för att lösa uppgifter.

I laborationen ska knappavläsningen kompletteras med en ”timeout”-funktion och på så vis byggs tillståndsmaskinen ut så den innehåller fler alternativ. Laborationen innehåller också uppgifter där hårdvarunära detaljer abstraheras bort för programmeraren genom att skapa funktioner som anropas istället för att operera på bitar.

Laborationens sista uppgift är frivillig för de som laborerar ensamma och i par om två. Om laborationsgruppen består av fler än två personer, är uppgiften obligatorisk.

Förberedelseuppgifter.

Läs igenom laborationen. Ta gärna fram lösningsförslag på följande funktioner:

- **Key_down** (uppgift 1, se sidan 4)
- **Count_and_display** (uppgift 2, se sidan 4)

Erik Larsson

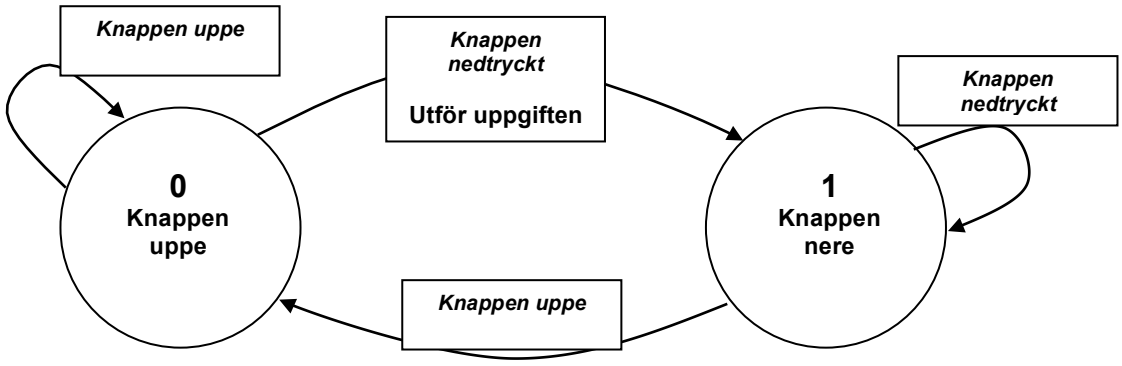
Laborationsuppgifter.

En tillståndsmaskin används för att beskriva ett problem på hög abstraktionsnivå, dvs innan problemet blivit beskrivet med hjälp av ett programmeringsspråk. Med en tillståndsmaskin ser man vad som ska göras beroende på indata, utdata och vad som hänt tidigare. Ett litet exempel är den tillståndsmaskin som användes under förra laborationen för att konstruera program. För den tillståndsmaskinen kunde programkod i avbrottsrutinen se ut så här:

```
ISR (SIG_OUTPUT_COMPARE1A){

    newPINA = PINA;          // Läs in porten.
    if ((newPINA & 0b10000000 == 0) && ((oldPINA & 0b10000000) != 0)){
//================================================================
        n = (n+1)%10;        // Öka n med ett och behåll entalsdelen.
        display(n);          // Anropa utskriftsfunktionen.
//================================================================
    }
    oldPINA = newPINA;       // Kom ihåg till nästa gång.
}
```

- **Om bit 7 i PINA är = 0 och om den förra gången inte var = 0, så utförs uppgiften.**

Värdet i oldPINA fungerar som en tillståndsvariabel; beteendet i avbrottsfunktionen blir olika beroende på om bit 7 är 0 eller 1. Nedan är en tillståndsgraf som visar vad som händer:



Cirklarna är tillstånd.
Pilarna pekar på nästa tillstånd; övergångarna kan bara ske vid varje avbrott (10 ms).
I rutorna på pilarna står

- ***villkoret för nästa tillstånd och***
- ***uppgiften som ska göras.***

Vill man göra sig en bild av tankegången så är tillståndsgrafen bra och den förklarar verkligen tydligt vad problemet går ut på.

I ord:

- Tillståndsmaskinen har två tillstånd, **Knappen uppe** och **Knappen nere**.
- Om tillståndsmaskinen är i tillstånd "**Knappen uppe**" så byter den tillstånd om knappen trycks ner.
- I tillståndet "**Knappen nere**" väntar man på att knappen ska släppas.

Observera att eventuella tillståndsändringar bara kan inträffa med vissa jämna tidsintervall (jämför med synkrona sekvensnät i digitaltekniken). I laborationen använder vi tidsintervallet 10 ms.

En fördel med tillståndsgrafen är att man relativt lätt kan omsätta den i programrader. Vi inför en tillståndsvariabel, **next_state**, som kan ha olika värden beroende på vilket tillstånd programmet befinner sig i. Varje värde på **next_state** motsvarar ett tillstånd. I grafen har vi två tillstånd; då kan vi t.ex. låta next_state vara 0 eller 1.

I avbrottsrutinen görs nu olika saker beroende på vilket tillståndet är. Nedan är exempel på två olika skrivsätt. Jämför de båda skrivsätten. I den högra är denna laborations uppgifter 1 och 2 använda. Läs igenom de uppgifterna för att förstå programkoden.

```
ISR (SIG_OUTPUT_COMPARE1A){
    static unsigned char next_state;
    if(next_state == 0){
        if((PINA&0b10000000) == 0){
            next_state = 1;
//== Count_and_display ============
            n = (n+1)%10;
            display(n);
//================================
        }
    }
    else if(next_state == 1){
        if((PINA&0b10000000) != 0){
            next_state = 0;
        }
    }
}
```

```
ISR (SIG_OUTPUT_COMPARE1A){
    static unsigned char next_state;
    switch(next_state){
        case 0:
            if(Key_down() == 1){
                next_state = 1;
                Count_and_display();
            }
            break;
        case 1:
            if(Key_down() == 0){
                next_state = 0;
            }
            break;
    }
}
```

Variabeldeklarationen `static unsigned char next_state;` innebär att variabeln är privat för avbrottsrutinen men har en speciell plats i minnet; det betyder att den inte glöms mellan avbrotten.

Uppgift 1. Funktion för att läsa av knappen, Key_down.

Studera den högra versionen av avbrottsfunktionen (sidan 3). Där står att

```
if((PINA&0b10000000)==0)
```

är ersatt av

```
if(Key_down()==1)
```

Fördelen med detta är att det blir mer lättläst.

- Skriv en funktion, **`Key_down`**, som läser av tryckknappen och returnerar funktionsvärdet 1 om den är nere och värdet 0 ifall den är uppe.

Man kan tycka att det är onödigt att göra en helt ny funktion bara för denna uppgift, men programmet blir mer lättförståeligt. En annan fördel är att om man vill ändra portkonfiguration så ändrar man bara i ett par funktioner. Om man inte har skrivit dessa funktioner måste man gå igenom all kod och leta efter ställen där man opererar på portkonfigurationen.

❑

Uppgift 2. Funktion för uppgiften, Count_and_display.

- För att få mer ordning och lättlästhet, så ska de uppgifter som vi gjort för varje knappnedtryckning samlas i en funktion med namnet **`Count_and_display`**.

❑

Uppgift 3. Avbrottsfunktion med tillståndsmaskin.

- Skriv in den nya avbrottsfunktionen enligt exemplet ovan. Du ska välja den högra versionen (med **case**-satser).

❑

Uppgift 4. Nollställning vid uppstart.

När du har startat programmet, men innan du tryckt första gången på knappen, så lyser inte displayen. När du trycker en gång så kommer en etta.
Det är inte riktigt snyggt. Komplettera programmet så att värdet på **`n`** (som är = 0) skrivs ut vid programstart.

- Provkör programmet och förvissa dig om att allt fungerar ordentligt!

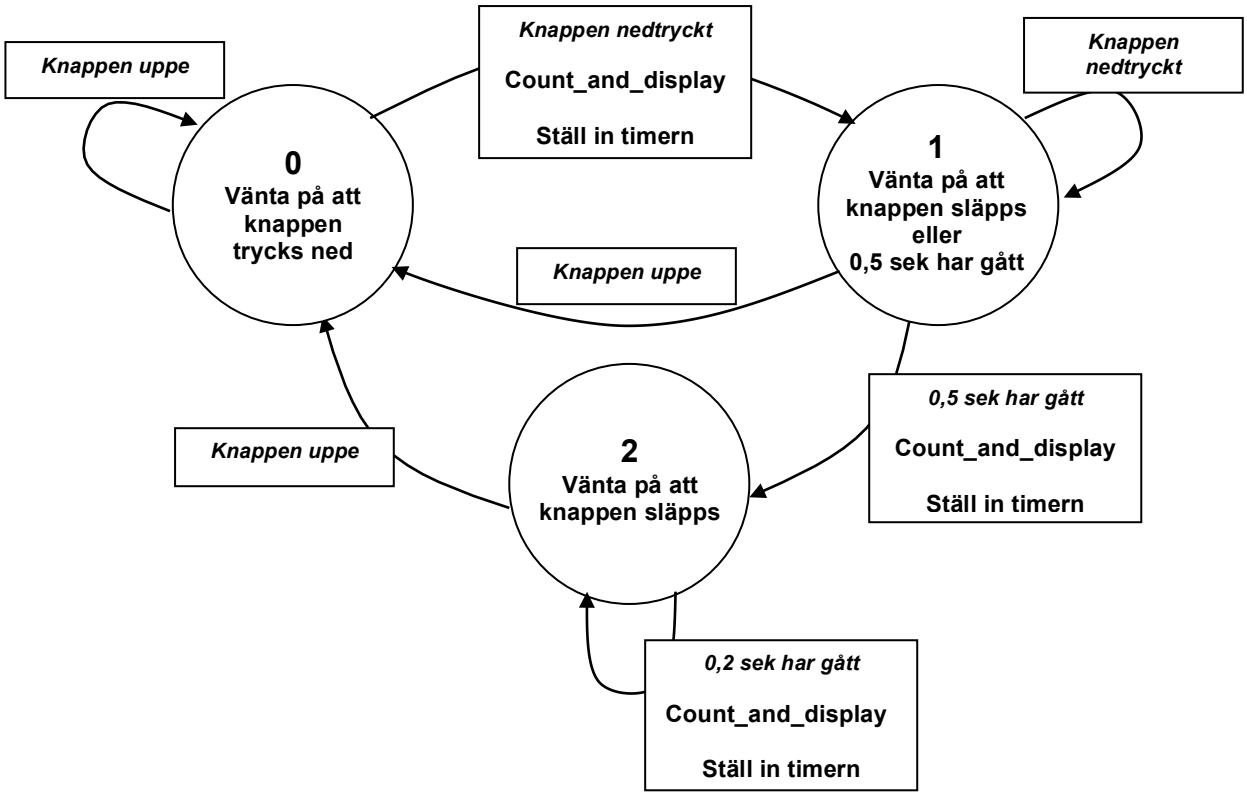
❑

Uppgift 5. Autorepetition

Om du håller en tangent nedtryckt på ett vanligt tangentbord en viss tid (en halv sekund kanske), så börjar det automatiskt att komma knappnedtryckningar av sig självt med ganska hög fart. I denna uppgift ska du konstruera om tillståndsgrafen så du får en sådan funktion:

- När knappen har hållits nere en halv sekund så utförs ”**`Count_and_display`**”, nästa tillstånd sätts till **2** och en variabel som får fungera som timer ställs in på 200 ms.
- Tillstånd **2** lämnas om knappen släpps, annars testas timern och varje gång den gått ner till noll så utförs ”**`Count_and_display`**” och timern sätts åter till 200 ms.

Se grafen här nedanför.



Gör ett eget förslag till lösning. Som hjälp har du här ett skal till programkod: Programkonstruktionen med **`switch – case`** visar tydligt vad programmet gör. Utan den här strukturen, är det lätt att röra till programmet så man helt tappar greppet.

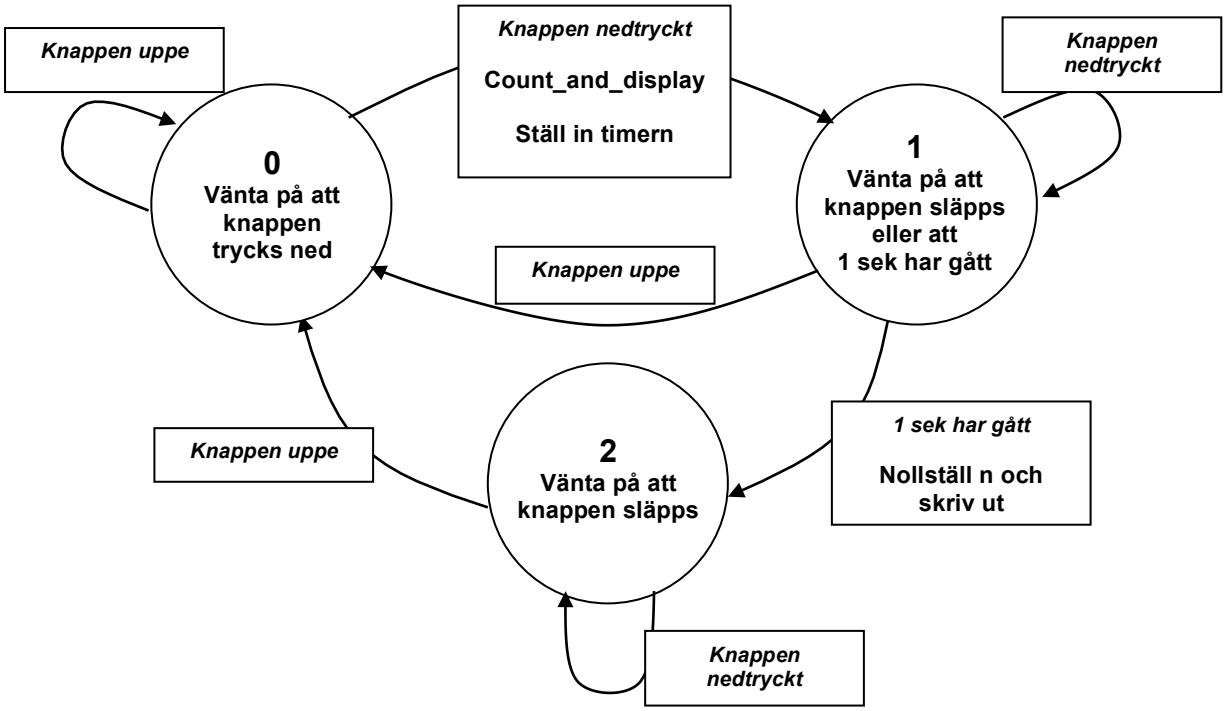
```
ISR (SIG_OUTPUT_COMPARE1A){
    static unsigned char next_state;
    switch (next_state){
        case 0:
            if(Key_down() == 1){
                next_state = 1;
                Count_and_display();

            // Sätt timern till ett lämpligt värde.

            }
            break;
        case 1:

            // Räkna ner timern.

            if(Key_down() == 0) next_state = 0; // Nästa tillstånd = 0
            else

            // Om timern blivit noll:
            //     Sätt timern till 0,2 sek, sätt next_state till 2
            //     och anropa Count_and_display.

            break;
        case 2:

            // Räkna ner timern.

            // Om knappen är uppe:
            //     Sätt next_state till 0.
            // Annars, om timern är noll sätts den till 0,2 sek och
            //     Count_and_display anropas

            break;
    }
}
```

- Skriv i din kod och testa!

❑

Uppgift 6. Manuell nollställning. Frivillig uppgift för de som laborerar ensamma eller i par om två

Modifiera föregående uppgift genom att ersätta autorepetitionen med ett sätt att göra manuell nollställning. Studera grafen nedan och modifiera din kod så att den stämmer med grafen.



❏