1. *Motivation*

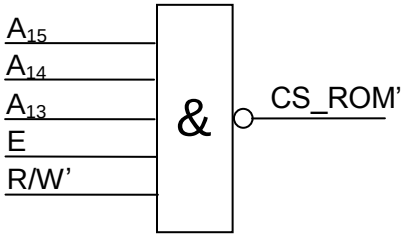
ROM. 8kbyte $\Rightarrow 2^3 \bullet 2^{10}$ byte $\Rightarrow$ 13 Adressbitar $\Rightarrow$ [A12,A0] direkt till ROM-kapsel.
RWM. 32kbyte $\Rightarrow 2^5 \bullet 2^{10}$ byte $\Rightarrow$ 15 Adressbitar $\Rightarrow$ [A14,A0] direkt till RWM-kapsel.
Minnesmodulerna och I/O-portarna tar upp följande adressområden:
Väljer att lägga IN- och UTport på samma adress då detta ger färre grindar.

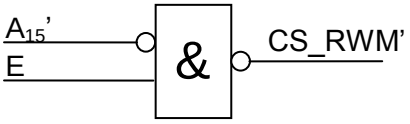
Lösning

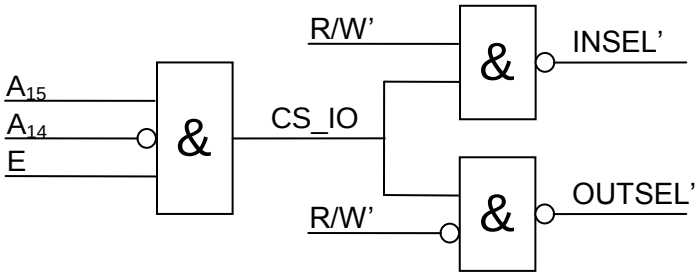
		A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
RWM	Start: 0000 -	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: 7FFF	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ROM	Start: E000	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	Stop: FFFF	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
I/O	8000	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Realisering

Alla chip select signaler är aktiva låga.
CS-ROM: $\{A_{15} \cdot A_{14} \cdot A_{13} \cdot E \cdot R/W\}'$
CS-RWM: $\{A_{15}' \cdot E\}'$
I/O: Bilda först IOSEL för både portarna enligt $\{A_{15}\ A_{14}' \cdot E\}$ Välj sedan OUTSEL enligt {IOSEL R/W'}'och slutligen INSEL enligt {IOSEL R/W}'







NY UPPGIFT - NY SIDA

2.

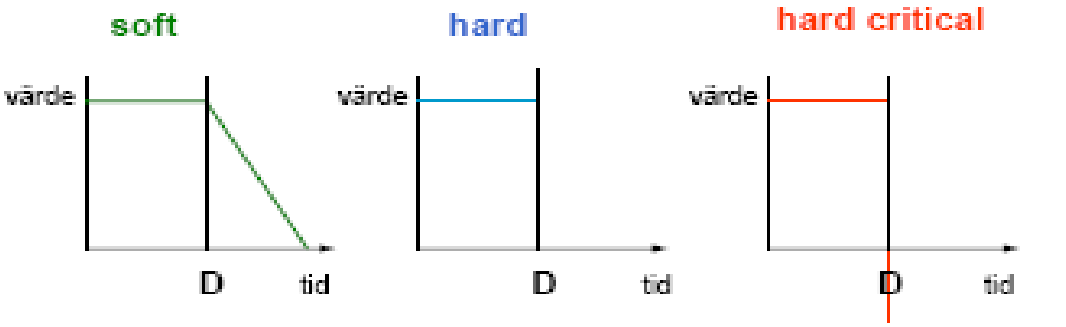
```
*****************************************************************
* Subrutin IRQINIT initierar avbrottssystemet.
* Variabeln CLOCK nollställs och variabeln TMP sätts till 100
* IN-parametrar:       -
* UT-parametrar:       -
* Påverkade register:  -
* Anrop:               JSR IRQINIT
* Anropade rutiner:    -
*****************************************************************
IRQINIT   psha
          pshx
          movw    #0,CLOCK        nollställ kolckan
          movb    #0,CLOCK+2
          ldaa    #100            Avbrottsräknare
          staa    TEMP
          staa    IRQRES          nollställ avbrottsvippan
          ldx     #IRQ            avbrottsvektor
          stx     $fff2           (alt 3ff2)
          cli
          pulx
          pula
          rts

TEMP      rmb     1    Avbrottsräknare (100 IRQ = 1s)

*****************************************************************
* Avbrottsrutin IRQ som startar var 10:de ms uppdaterar klockan enligt:
*    1) Kvittera avbrott
*    2) Minska variabeln TMP (startvärde 100).
*    3) OM TMP blir noll:
*          öka den globala variabeln CLOCK
* IN/UT-parameter: Global variabel CLOCK
*****************************************************************
IRQ       staa    IRQRES          nollställ avbrottsvippan
          dec     TEMP            100 avbrott?
          bne     IExit             nej
          ldaa    #100            Avbrottsräknare
          staa    TEMP
 * Öka sekunder
          ldaa    CLOCK+2
          adda    #1
          daa
          staa    CLOCK+2
          cmpa    #60             Hel minut?
          bne     IExit             nej
 * Öka minuter
          clr     CLOCK+2         Nolla sekunder
          ldaa    CLOCK+1
          adda    #1
          daa
          staa    CLOCK+1
          cmpa    #60             Hel timme?
          bne     IExit             nej
 * Öka timmar
          clr     CLOCK+1         Nolla minuter
          ldaa    CLOCK
          adda    #1
          daa
          staa    CLOCK
          cmpa    #24             24 timmar?
          bne     IExit             nej
          clr     CLOCK           Nolla timmar
IExit     rti
```

NY UPPGIFT - NY SIDA

3.

a. Flexibelt, Multimaster, Multicast, Hög överföringshastighet, Kort svarstid, Automatisk omsändning, Atomic broadcast, Synkroniserad exekvering, Avlastar processorn.

b. En deadline's värdefunktion beskriver resultatet av det som händer om systemet ej levererar sitt svar i tid: Man skiljer på



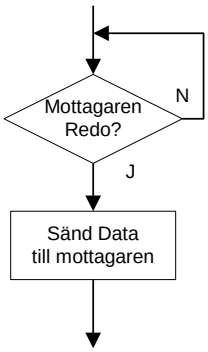
Kvalitén hos vissa tjänster kan komma att sänkas.

Resultatet är värdelöst" för systemet.

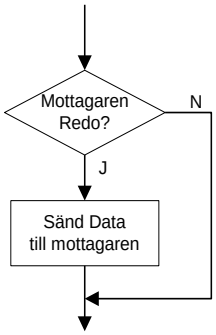
Resultatet är nödvändigt för undvikande av katastrof.

c)

Busy Wait och Polling används för att synkronisera processorn med en yttre enhet.



Busy Wait



Polling

Fördelar:
Enheten får service direkt

Nackdelar:
Andra enheter får vänta ty processorn är upptagen med att vänta

Typexempel på kod:

```
      ---
Wait
      ldaa Statusregister
      bita #ReadyBit
      beq  Wait

      stab Dataregister
      ---
```

Fördelar:
Flera enheter kan "servas samtidigt"

Nackdelar:
Service ges inte direkt till en redo enhet ty processorn kan jobba med annat

Typexempel på kod:

```
      ---
      ldaa Statusregister
      bita #ReadyBit
      beq  TestLater

      stab Dataregister

TestLater
      ---
```

d) Pass 1: Skapa symboltabell;
Pass 2: Lägg ut kod

e) Exempelvis

```
Loop   BRSET  STATUS,#4,Loop
```

kan bytas mot

```
       PSHA
Loop   LDAA   STATUS
       BITA   #4
       BNE    Loop
       PULA
```

UPG 4

```
char *strstr(const char *s1, const char *s2) {
   const char *p1, *p2;
   for (; *s1; s1++) {
     for (p1 = s1, p2 = s2; *p2 && *p1==*p2; p1++, p2++)
       ;
     if (!*p2)
       return s1;
   }
   return 0;
}
```

UPG5

// **Deluppgift a**

```
// Filen monitor.h
#include "clock.h"
void monitor(time_type interval, int (*test)(void), void (*alarm)(void));

// Filen monitor.cpp
#include "monitor.h"
#include "clock.h"


void monitor(time_type interval, int (*test)(void), void (*alarm)(void)) {
  while (1) {
    hold(interval);
    if (!test())
      alarm();
  }
}
```

// **Deluppgift b**

```
// Filen heart_ports.h
#ifndef HEART_PORTS_H
#define HEART_PORTS_H

// Färdiga Typer
typedef unsigned char port;         // port
typedef unsigned char *portptr;     // pekare till en port
typedef void (**vecptr) (void);     // pekare till avbrottsvektor

// Färdiga Användbara makron
#define set(r, mask)     (r) = (r) | mask
#define clear(r, mask)   (r) = (r) & ~mask

// Egna nya definitioner
#define HEARTREG_ADR     0xD00
#define HEARTREG         *((portptr) HEARTREG_ADR)

#define HEART_VEC_ADR      0xFF90   // Adress till avbrottsvektor
#define HEART_VEC          *((vecptr) HEART_VEC_ADR)

#define POWERREG_ADR     0xD02
#define POWERREG         *((portptr) POWERREG_ADR)

#define on_bit        0x01
#define intr_bit      0x40
#define reset_bit     0x80
#endif
```

```
// Filen heart.c
#include "heart_ports.h"
#include "monitor.h"

void call_nurse(void);          // Färdigskriven funktion
void heart_trap(void);          // Avbrottsrutin i assembler
static int beat = 0;

void heart_inter(void) {        // Anropas av rutinen heart_trap
  clear(HEARTREG, reset_bit);
  beat = 1;
}

void init_heart(void) {
  HEART_VEC = heart_trap;
  set(HEARTREG, on_bit);
  set(HEARTREG, intr_bit);
}

int heart_test(void) {
  int b = beat;
  beat = 0;
  return b;
}

void heart_alarm(void) {
  POWERREG = 6 << 4;
  set(POWERREG, on_bit);
  call_nurse();
}


int main() {
  init_clock();
  init_heart();
  monitor(5000, heart_test, heart_alarm);
}




}
```