Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA482 Maskinorienterad programmering D
EDA483 Maskinorienterad programmering E
DAT017 Maskinorienterad programmering IT
DIT153 Maskinorienterad programmering GU

Fredag 31 maj 2024, kl. 8.30 - 12.30

Examinatorer
Roger Johansson, tel. 772 57 29
Erik Sintorn, tel. 772 52 29

Kontaktperson under tentamen:
Roger Johansson, tel. 772 57 29
Erik Sintorn, tel. 772 52 29

Tillåtna hjälpmedel

Utgåvor som distribuerats inom ramen för kursen, häftet:

- *Quick Guide,*
  Laborationsdator MD407 med tillbehör

Inget annat än understrykningar (”överstrykningar”) får vara införda i detta häfte.

Tabellverk eller miniräknare får ej användas.

Lösningar

Anslås senast dagen efter tentamen via kursens hemsida.

Granskning

Tid och plats anges på kursens hemsida.

Allmänt

Uppgifter 1 t.o.m 3 utgörs av flervalsfrågor.
Försök först lösa uppgifterna, så som du lärt dig i kursen, jämför sedan med flervalsalternativen och ange det alternativ du anser är korrekt. Högst ett svarsalternativ får anges. Siffror inom parentes anger möjliga poäng på uppgiften.

Observera att, bland svarsalternativen finns alternativ som *kan* ge poängavdrag. Om du är mycket osäker på ditt svar bör du därför i stället välja att avstå.

Avge dina flervalssvar på blanketten, längst bak i tentamenstesen. Skicka med denna blankett med dina övriga lösningar.

För övriga uppgifter gäller:

Svar kan avges på svenska eller engelska.

Siffror inom parentes anger full poäng på uppgiften.

För full poäng krävs att:

- redovisningen av svar och lösningar är läslig och tydlig. Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift.
- lösningen ej är onödigt komplicerad.
- du har motiverat dina val och ställningstaganden
- assemblerprogram är utformade enligt de råd och anvisningar som getts under kursen.
- C-program är utformade enligt de råd och anvisningar som getts under kursen.
  I programtexterna skall raderna dras in så att man tydligt ser programmens struktur.

Betygsättning

För godkänt slutbetyg på kursen fordras att både tentamen och laborationer är godkända.

Maximal poäng är 50 och tentamenspoäng ger slutbetyg enligt:
$20p \leq$ betyg $3 < 30p \leq$ betyg $4 < 40p \leq$ betyg 5

Uppgift 1 (-1..6p)

En C-funktion `convert` är definierad enligt listning till höger.

Ange det alternativ (A-H) som visar en korrekt implementering (dvs. hur funktionen ska kodas) i ARM v6 assemblerspråk.

```
int convert( signed char *cp )
{
  int converted = 0;
  while( *cp )
  {
    if( *cp<0 )
    {
      *cp = -(*cp);
      converted++;
    }
    cp++;
  }
  return converted;
}
```

A

```
convert:
    MOV     R1,R0
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    CMP     R2,#0
    BGE     L2
    NEG     R2,R2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

B

```
convert:
    MOV     R0,R1
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    SXTB    R2,R2
    CMP     R2,#0
    BLT     L2
    NEG     R2,R2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

C

```
convert:
    MOV     R1,#0
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    CMP     R2,#0
    BGE     L2
    NEG     R2,R2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

D

```
convert:
    MOV     R1,R0
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    SXTB    R2,R2
    CMP     R2,#0
    BLT     L2
    NEG     R2,R2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

E

```
convert:
    MOV     R1,#0
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    CMP     R2,#0
    BLT     L2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

F

```
convert:
    MOV     R1,#0
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    SXTB    R2,R2
    CMP     R2,#0
    BGE     L2
    NEG     R2,R2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

G

```
convert:
    MOV     R1,#0
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    SXTB    R2,R2
    CMP     R2,#0
    BLT     L2
    NEG     R2,R2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

H

```
convert:
    MOV     R0,#0
L1:
    LDRB    R2,[R0]
    CMP     R2,#0
    BEQ     L3
    CMP     R2,#0
    BGE     L2
    NEG     R2,R2
    STRB    R2,[R0]
    ADD     R1,R1,#1
L2:
    ADD     R0,R0,#1
    B       L1
L3:
    MOV     R0,R1
    BX      LR
```

Uppgift 2 (-2..6p)

C-funktionen `bitcheck` undersöker en parameter med avseende på antalet 1-ställda bitar.

Funktionen deklareras:

```
int   bitcheck( unsigned int *pp, int * num );
```

- `pp` är en pekare till det värde som ska undersökas
- `num` är en pekare till en plats för returvärde, dvs. antalet 1-ställda bitar hos parametern

Funktionen returnerar 1 om antalet ettor hos parametern är udda, annars ska funktionsvärdet vara 0.

Sex olika programmerare får i uppgift att implementera funktionen. Resultaten blir likartade men bara en programmerare presterar ett helt korrekt resultat. De olika alternativen finns nedan, ange alternativet med den korrekta lösningen.

A

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;
  while(pp)
  {
    if( pp & 1 )
      retval++;
    pp >>= 1;
  }
  num = retval;
  return retval & 1;
}
```

B

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;
  while(*pp)
  {
    if( *pp & 1 )
      retval++;
    *pp >>= 1;
  }
  num = retval;
  return retval & 1;
}
```

C

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;
  while(*pp)
  {
    if( *pp & 1 )
      retval++;
    *pp >>= 1;
  }
  *num = retval;
  return retval & 1;
}
```

D

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;
  while(pp)
  {
    if( pp & 1 )
      retval++;
    pp >>= 1;
  }
  *num = retval;
  return retval & 1;
}
```

E

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;
  while(pp)
  {
    if( pp & 1 )
      retval = retval+sizeof(int);
    pp >>= 1;
  }
  num = retval;
  return retval & 1;
}
```

F

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;
  while(*pp)
  {
    if( *pp & 1 )
      retval = retval+sizeof(int);
    *pp >>= 1;
  }
  *num = retval;
  return retval & 1;
}
```

Uppgift 3 (-1..4p)

Följande deklarationer är givna på "toppnivå".

```
static unsigned char f,c;
int a;
int foo2(unsigned char);
short foo1(unsigned char,int);
```

Vi har nu tilldelningen:

```
a = foo1 ( c, foo2(f) );
```

Vilken av följande implementeringar i ARM v6-kod är korrekt?

A

```
LDR     R0,=f
LDRB    R0,[R0]
BL      foo2
LDR     R0,=c
LDRB    R0,[R0]
BL      foo1
SXTH    R0,R0
LDR     R1,=a
STRH    R0,[R1]
```

B

```
LDR     R0,=f
LDRB    R0,[R0]
BL      foo2
LDR     R0,=c
LDRB    R0,[R0]
BL      foo1
SXTH    R0,R0
LDR     R1,=a
STR     R0,[R1]
```

C

```
LDR     R0,=f
LDRB    R0,[R0]
BL      foo2
MOV     R1,R0
LDR     R0,=c
LDRB    R0,[R0]
BL      foo1
SXTH    R0,R0
LDR     R1,=a
STR     R0,[R1]
```

D

```
LDR     R0,=f
LDR R0,[R0]
BL      foo2
MOV     R1,R0
LDR     R0,=c
LDR R0,[R0]
BL      foo1
SXTH    R0,R0
LDR     R1,=a
STR     R0,[R1]
```

E

```
LDR     R0,=f
LDRB    R0,[R0]
BL      foo2
MOV     R1,R0
LDR     R0,=c
LDRB    R0,[R0]
BL      foo1
LDR     R1,=a
STRH    R0,[R1]
```

F

```
LDR     R0,=f
LDR     R0,[R0]
BL      foo2
LDR     R0,=c
LDR     R0,[R0]
BL      foo1
LDR     R1,=a
STRH        R0,[R1]
```

G

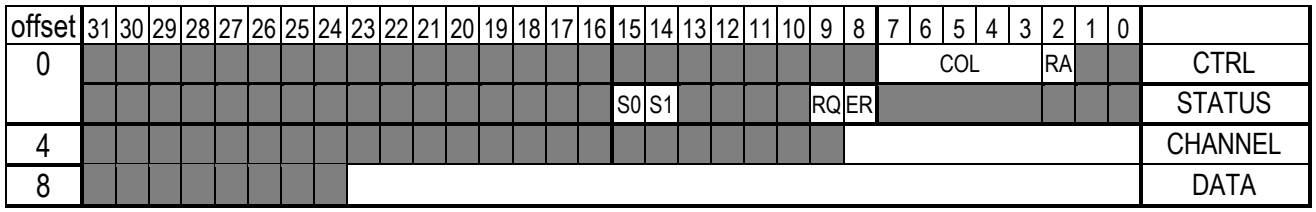
```
LDR     R0,=f
LDR     R0,[R0]
BL      foo2
LDR     R0,=c
LDR     R0,[R0]
BL      foo1
LDR     R1,=a
STR     R0,[R1]
```

H

```
LDR     R0,=f
LDRB    R0,[R0]
BL      foo2
MOV     R1,R0
LDR     R0,=c
LDRB    R0,[R0]
BL      foo1
LDR     R1,=a
STR     R0,[R1]
```

Uppgift 4 (10p)

Följande port som utgör ett gränssnitt mot en yttre periferienhet är placerad på adress `0xFF600000`:



a) Visa lämpliga makrodefinitioner för referens (åtkomst) av portens olika register (2p).

b) Visa med en typdefinition `PORT`, hur porten kan avbildas med en *struct*-definition (2p).
Visa speciellt hur delen `DATA` då refereras med hjälp av din typdefinition (2p).

c) Visa hur portens första register (offset=0) kan avbildas med en bitfältdeklaration där de olika bitarna kan refereras var för sig. (4p).

Uppgift 5 (5p)

Man vill implementera en klocka med hjälp av TIMER7.

Antag att en avbrottsfunktion `void timer7_interrupt(void)` är definierad.

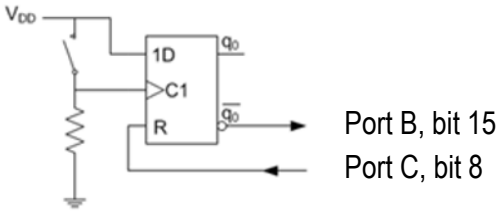
Visa med en funktion `void timer7_init(void)` hur TIMER7 konfigureras för avbrottsystemet (NVIC) och startar periodiska avbrott med 100 ms intervall. (Se även datablad i Appendix).

Följande makrodefinitioner kan förutsättas givna som pekare till respektive register.

`NVIC_ISER0` (*Set enable 0*), `TIM7_PSC`, `TIM7_ARR`, `TIM7_DIER`, `TIM7_CR1`

Uppgift 6 (5p)

En avbrottsvippa kopplas till *MD407* enligt följande figur:



Vi påminner om vippans funktion:

Då nivån på C1 ändras till 1 (positiv flank) ändras vippans interna tillstånd q_0 till 1D (1), om R samtidigt är 0.

Då R har värdet 1, tvingas q_0 till 0.

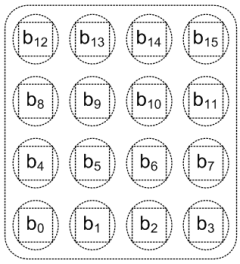
Den externa avbrottsmekanismen (EXTI) ska användas för att detektera knapptryckningar. Ingen hänsyn behöver tas till eventuella kontaktstudsar. Avbrott ska ske på negativ flank.

Förutsätt att makrodefinitioner för pekare till använda register finns givna med namnkonventioner enligt *QuickGuide*.

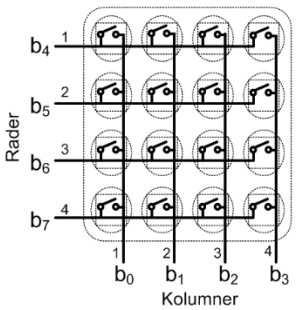
Visa med en funktion `app_init` hur avbrottsmekanismerna initieras, dvs. SYSCFG, EXTI, Port C, Port B och NVIC initieras. Tänk på att bara de pinnar som berörs får ändras vid konfigureringen.

Uppgift 7 (6p)

Under laborationerna har du arbetat med ett enkelt tangentbord där tangenterna organiserats i rader och kolumner. Man vill kunna detektera flera samtidigt nedtryckta tangenter och därför konstruera en tangentbordsrutin som returnerar ett statusord (16 bitar) där varje tangent har en bitposition och värdet 1 indikerar en nedtryckt tangent medan värdet 0 indikerar en uppsläppt tangent. På grund av tangentbordets konstruktion är det lämpligt att välja avbildning enligt figuren till höger mellan bitposition och tangent.



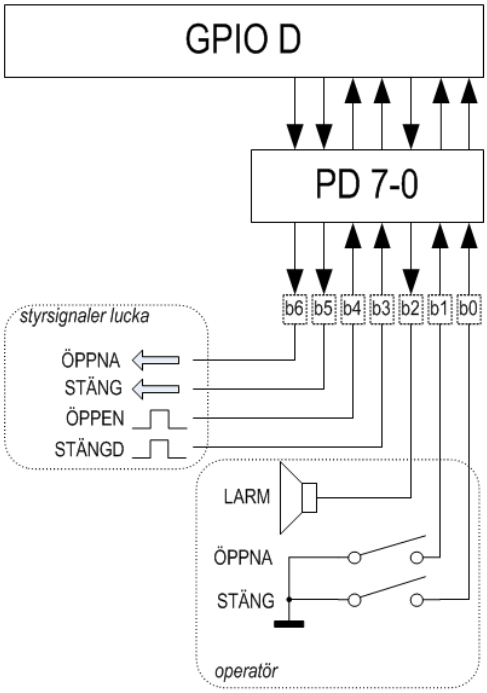
Antag att port D, bit 0..7, kopplats till tangentbordet enligt figuren till höger. Eftersom flera tangenter kan tryckas ned samtidigt vill vi göra en ”kortslutningssäker” lösning för koincidensavsökning av tangentbordet. Utgångarna ska därför vara OPEN-DRAIN. De avlästa kolumnerna ska vara PULL UP.
Skapa en initieringsfunktion `void init_app( void )` som konfigurerar port D för användning med tangentbordet och en tangentbordsrutin `unsigned short keyb (void)` som söker av tangentbordet och placerar kolumnvärde för varje rad i en `unsigned short` och returnerar denna.



Förutsätt att makrodefinitioner för pekare till använda register finns givna med namnkonventioner enligt *QuickGuide*.

Uppgift 8 (8p)

En elektroniskt styrd ventilationslucka ska konstrueras. Luckan har ett 7-bitars digitalt gränssnitt (se figur till höger) som beskrivs av följande:



Operatörsignaler:

En funktion för att ge larm via en högalare aktiveras med en etta på bit 2.

Funktioner för att öppna och stänga luckan utgörs av två återjädrande strömställare anslutna via b1 och b0. Då strömställarna är öppna är dessa ingångar flytande och måste därför förses med programmerad *pull-up*. Om båda strömställarna aktiveras samtidigt ska larmsignal ges.

Signaler till/från ventilationslucka:

Indikatorer för helt öppen respektive helt stängd lucka är anslutna via b4 och b3. Signalerna är aktiva vid hög nivå. Om båda dessa signaler är 0 betyder det att luckan håller på att öppnas eller stängas. Om båda signaler är 1 innebär detta något fel och larmet ska då aktiveras.

Signalerna b6 och b5 aktiveras (sätts till 1) för att öppna respektive stänga luckan. Dessa signaler får inte aktiveras samtidigt. I mekaniken som reglerar luckan finns en viss tröghet så det kan ta maximalt 1 sekund att öppna och maximalt 2 sekunder att stänga luckan. Om det skulle ta längre tid ska larmsignal ges.

För att kunna kontrollera att det inte tar mer än 1 sekund att öppna eller stänga luckan krävs en funktion för realtidsfördröjning användas. Du kan förutsätta att en sådan blockerande funktion `void delay_100ms(void);` som fördröjer det anropande programmet i 1/10 sekund finns tillgänglig. Du kan alltså använda denna men behöver inte visa funktionen här.

Förutsätt att makrodefinitioner för pekare till använda register finns givna med namnkonventioner enligt *QuickGuide*.

Konstruera tre funktioner, `init`, `open` och `close` enligt följande specifikationer:

a) `void init ( void );` initiera GPIOD för användning med luckan.
Oanvända pinnar i porten ska ställas som ingångar. (2p)

b) `int open ( void );` öppna luckan, vänta maximalt 1 sekund i 100 ms intervall.
Om dörren öppnas inom 1 sekund ska funktionen returnera 1,
annars ska funktionen returnera 0. (3p)

c) `int close (void);` stäng lucka, vänta maximalt 2 sekunder i 100 ms intervall.
Om dörren är stängd inom 2 sekunder ska funktionen returnera 1,
annars ska funktionen returnera 0. (3p)

APPENDIX: TIMER6 och TIMER7.

TIM6 och TIM7 styrregister CR1 (Control Register 1)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x00									ARPE				OPM	URS	UDIS	CEN	TIMxCR1

ARPE: *Auto-reload preload enable*
0: TIMx_ARR register ändras via ett buffertregister.
1: TIMx_ARR register ändras omedelbart.
Då registret uppdateras med ett nytt värde kan detta överföras omedelbart, eller efter nästa kompletta räknarintervall är klart.

OPM: *One-pulse mode*
0: Räknaren fortsätter efter ett räknarintervall (*kontinuerligt arbetssätt*)
1: Räknaren stoppas efter ett räknarintervall genom att CEN-biten nollställs.

URS: *Update request source*
Biten kontrollerar källorna till UEV (*update event*).

UDIS: *Update disable*
Biten bestämmer om UEV (*update event*) kan genereras.
0: UEV är möjligt och kan genereras. Se även URS.
Register ges begynnelsevärden från buffertar.
1: UEV är avstängd, inget UE (*update event*) genereras.
Skuggregister (hos buffrade register ARR och PCS) uppdateras bara då UG-biten sätts till 1.

CEN: *Counter enable*
0: Räknarkretsen är deaktiverad
1: Räknarkretsen är aktiverad

TIM6 och TIM7 styrregister DIER (DMA/interrupt enable register)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x0C								UDE								UIE	TIMxDIER

UDE: *Update DMA request enable*
0: *Update* vid DMA-begäran deaktiverad.
1: *Update* vid DMA-begäran aktiverad.
UIE: *Update interrupt enable*
0: *Update* genererar inget avbrott.
1: *Update* genererar avbrott.

Register för räknarvärden

PSC och ARR ger tillsammans räknarintervallet, dvs. maximala antalet pulser som räknas vilket då ger periodtiden till ett *update event*.

TIM6 och TIM7 räknare PSC (Prescaler)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x28								PSC[15:0]									TIMxPSC

TIM6 och TIM7 räknare ARR (Auto Reload Register)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x2C								ARR[15:0]									TIMxARR

TIMxPSC och TIMx_ARR anger räknarkretsens tidsbas, dvs. räknarintervall till *update event*. Räknarkretsarna använder samma klocka som tidsbas, antalet klockpulser under 1 sekund, dvs frekvensen för UE, bestäms av:

$$UE\ Hz = \frac{CLK\ \ Hz}{((PSC + 1)(ARR + 1))}$$

Räknarens klocka är här hälften av systemets klockfrekvens och för *MD407* innebär detta:

$$UE\ Hz = \frac{84\ \ MHz}{((PSC + 1)(ARR + 1))}$$

Svarsblankett

Anonym kod

Maskinorienterad programmering, tentamen flervalsfrågor.

Kryssa i det alternativ du anser vara korrekt, eller avstå.
Lämna in denna blankett tillsammans med dina övriga lösningar.

Fråga	A	B	C	D	E	F	G	H
1	0	0	1	-1	-1	6	0	0
2	0	-2	6	-2	0	0		
3	0	1	4	0	0	-1	-1	2

Lösningsförslag

Uppgift 1 (6p):

Svarsalternativ F

```
@ R0: cp
@ R1: converted
convert:
    MOV     R1,#0       @ converted = 0
L1: LDRB    R2,[R0]     @ R2<= *cp
    CMPI    R2,#0       @ *cp!== 0 ?
    BEQ     L3          @ if YES then L3
    SXTB    R2,R2       @ R2 <- (int) *cp
    CMPI    R2,#0       @ (int)*cp < 0 ?
    BGE     L2          @ if NO then L2
    NEG     R2,R2       @ R2 <-  -(*cp)
    STRB    R2,[R0]     @ *cp = -(*cp)
    ADDI    R1,R1,#1    @ converted++;
L2: ADDI    R0,R0,#1    @ cp++
    B       L1
L3: MOV     R0,R1       @ return converted;
    BX      LR
```

Uppgift 2 (6p):

Svarsalternativ C

```
int bitcheck( unsigned int *pp, int *num)
{
  int   retval = 0;
  while(*pp)
  {
    if( *pp & 1 )
      retval++;
    *pp >>= 1;
  }
  *num = retval;
  return retval & 1;
}
```

Uppgift 3 (4p):

Svarsalternativ C

```
    LDR R0,=f           @ R0 <- &f
    LDRB    R0,[R0]     @ R0 <- (unsigned char) f
    BL      d           @ R0 <- d(f)
    MOV R1,R0           @ R1 <- d(f)
    LDR R0,=c           @ R0 <- &c
    LDRB    R0,[R0]     @ R0 <- (unsigned char) c
    BL      b           @ R0 <- b( c, d(f))
    SXTH    R0,R0       @ R0 <- (int) b( c, d(f))

    LDR R1,=a           @ R1 <- &a
    STR R0,[R1]         @ a <- (int) b( c, d(f))
```

Uppgift 4 (10p):

Uppgift 2a:

```
#define CTRL    (( unsigned char  *) 0xFF600000)
#define STATUS  (( unsigned char  *) 0xFF600001)
#define CHANNEL (( unsigned short *) 0xFF600004)
#define DATA    (( unsigned long  *) 0xFF600008)
```

Uppgift 2b:

```
typedef struct{
  unsigned char CTRL;
  unsigned char STATUS;
  short unused0;
  unsigned short CHANNEL;
  unsigned short unused1;
  unsigned long DATA;
} PORT;
```

Referens: `( (PORT *) (0xFF600008))->data;`

Uppgift 2c:

```
typedef struct{
    unsigned int :2;
    unsigned int RA:1;
    unsigned int COL:5;
    unsigned int ER:1;
    unsigned int RQ:1;
    unsigned int :4;
    unsigned int S1:1;
    unsigned int S0:1;
}REG0;
```

Uppgift 5 (5p)

```
#define TIM7_IRQVEC 0x2001C011C
#define NVIC_TIM7_IRQ_BPOS  23
void timer7_init(void)
{
  *TIM7_IRQVEC = timer7_interrupt;
  *(NVIC_TIM7_ISER0+4) |= NVIC_TIM7_IRQ_BPOS;

  /* 100 ms tidbas*/
 *TIM7_PSC= 839;
 *TIM7_ARR = 9999;
 *TIM7_DIER |= UIE;
 *TIM7_CR1 |= CEN;
}
```

Uppgift 6 (5p)

```
#define NVIC_EXTI8_IRQ_BPOS  (1<<23)
#define EXTI8_IRQ_BPOS       (1<<8)
void app_init( void )
{
    /* Port C, bit 8 utgång */
    *GPIOC_MODER  &= ~0x0100;
    *GPIOC_MODER  |= 0x0100;
    /* Utgång ”push/pull” */
    *GPIOC_OTYPER &= ~0x0100;
    /* Port B, bit 15 ingång */
    *GPIOB_MODER  &= ~0xC000;
    /* Ingång ”floating” */
    *GPIOB_PUPDR  &= ~0xC000;
    /* PC8->EXTI0  */
    *SYSCFG_EXTI3 |= 0x0002;
    *EXTI_IMR |= EXTI8_IRQ_BPOS;
    *EXTI_FTSR |= EXTI8_IRQ_BPOS;
    *EXTI_RTSR &= ~EXTI8_IRQ_BPOS;
    *NVIC_ISER0 |= NVIC_EXTI8_IRQ_BPOS;
}
```

Uppgift 7 (6p)

```
void init_app( void )
{
    /*  PORT D
    b8-b4 used for output to rows
    b3-b0  used for input from columns
    */
    *GPIO_D_MODER   = 0x00005500;
    /* Input, pull up */
    *GPIO_D_PUPDR   = 0x00000055;
    /* outputs are open drain */
    *GPIO_D_OTYPER  = 0x00F0;
}

unsigned short keyb (void)
{
    int row;
    unsigned short pad;
    *GPIO_D_ODR_LOW = ~0x80;
    pad = (unsigned short)   (*GPIO_D_IDR_LOW & 0xF);
    *GPIO_D_ODR_LOW = ~0x40;
    pad |= (unsigned short)  ((*GPIO_D_IDR_LOW <<4) & 0x00F0);
    *GPIO_D_ODR_LOW = ~0x20;
    pad |= (unsigned short)  ((*GPIO_D_IDR_LOW <<8) & 0x0F00);
    *GPIO_D_ODR_LOW = ~0x10;
    pad |= (unsigned short)  ((*GPIO_D_IDR_LOW <<12) & 0xF000);
    *GPIO_D_ODR_LOW =  0xFF;
    return pad^0xFFFF;  /* Pattern is inverse */
}
```

Uppgift 8 (8p)

```
void init ( void )
{
  *GPIO_D_MODER  = 0x00001410;  /* b6,b5,b2 digital ut, övriga digital in */
  *GPIO_D_OTYPER = 0;  /* utgångar är push/pull */
  *GPIO_D_PUPDR  = 5;  /* b0, b1 är pull-up */
  *GPIO_D_ODR    = 0;  /* Deaktivera styrsignaler */
}


int open ( void )
{
    *GPIO_D_ODR = (1<<6);  /* b6=1 */
    for( int i = 0; i < 10; i++ )
    {
        if( (*GPIO_D_IDR & (1<<4) )
            return 1;   /* Dörr öppnad */
        delay_100ms();
    }
    return 0; /* Kunde inte öppna dörren */
}

int close( void )
{
    *GPIO_D_ODR = (1<<5);  /* b5=1 */
    for( int i = 0; i < 20; i++ )
    {
        if(*GPIO_D_IDR & (1<<3))
            return 1;   /* Dörr stängd */
        delay_100ms();
    }
    return 0; /* Kunde inte stänga dörren */
}
```