Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Hemtentamen med delvisa lösningsförslag

EDA482 Maskinorienterad programmering D
EDA487 Maskinorienterad programmering Z
EDA488 Maskinorienterad programmering Z
DIT151 Maskinorienterad programmering GU
DAT017 Maskinorienterad programmering IT
DAT390 Maskinorienterad programmering Hing

Fredag 20 augusti 2021, kl. 8.30 - 12.30 (14.30)

Examinatorer
Roger Johansson
Pedro Trancoso
Erik Sintorn

Kontaktpersoner under tentamen:
Roger Johansson, epost (Canvas)
Pedro Trancoso, epost (Canvas)
Erik Sintorn, epost (Canvas)

Tillåtna hjälpmedel

Alla hjälpmedel är tillåtna så länge uppgifterna löses på individuell basis utan att konsultera någon annan än examinator under skrivningstiden.

Lösningsförslag
Anslås senast dagen efter tentamen via kursens hemsida.
Lösningsförslagen är endast vägledande för hur korrekt kod ska vara utformad och anvisar inte hur poängbedömning kommer att utföras.

Bedömning/Granskning
Tillfällen för granskning av bedömningar kommer att publiceras på respektive kurshemsida.

Allmänt
Svar kan avges på svenska eller engelska.

Tänk på att disponera din tid väl. Försök avge lösningsförslag på samtliga uppgifter.

Inlämning sker enligt anvisningar på Canvas. Observera att inlämningar efter tentamenstid + 10 min. (12.40) inte kommer att bedömas. (Betraktas som blank inlämning). Tentander som beviljats förlängd skrivtid lämnar in senast kl. 14.40.

Betygsättning för hemtentamen
Maximal poäng är 60 och tentamenspoäng ger betyg enligt: (EDA/DAT):
$30p \leq$ betyg 3 $< 40p \leq$ betyg 4 $< 50p \leq$ betyg 5.
respektive (DIT):
$30p \leq$ betyg G $< 45p \leq$ VG

Tentamensbetyg ges av hemtentamen.

Som slutbetyg på kursen ges betyg från hemtentamen under förutsättning att laborationskursen är godkänd.

Bedömningskriterier för programmeringsuppgifter vid hemtentamen:

En lösning bedöms utifrån tre olika aspekter där varje aspekt kan ge 0-5 poäng. En uppgift kan därför ge högst 15 poäng och lägst 0 poäng. De olika bedömningsaspekterna är:

- Kodkvalité och kodens fullständighet
- Dokumentation och kommentarers kvalité och fullständighet
- Relevans

Följande uppställningar ger kortfattade förtydliganden om de generella grunderna för respektive bedömning. Eftersom uppgifter kan ha skilda karaktärer och lösningar anta mycket olika former kan dock ytterligare (speciella) bedömningsgrunder komma att tillämpas.

Kodkvalité och kodens fullständighet

- Är koden syntaktiskt korrekt och följs de anvisningar och rekommendationer som kursen lärt ut?
- Följs eventuella kodkonventioner?
- Finns alla nödvändiga komponenter med, dvs. variabeldeklarationer och funktioner(subrutiner).

Bedömningsskala:

5. Högsta kvalitet och fullständig
4. Enstaka kvalitetsbrister men fullständig
3. Enstaka kvalitetsbrister, inte helt fullständig
2. Flera kvalitetsbrister och/eller ofullständighet
1. Stora kvalitetsbrister och/eller menlig ofullständighet
0. Kod kan inte bedömas eller saknas helt

Dokumentation och kommentarers kvalité och fullständighet

Dokumentation och kommentarer ska vara utformade på ett sätt som ger en uttömmande förklaring till hur koden är avsedd att fungera. Speciellt kontrolleras följande:

- Finns aktuella gränssnitt dokumenterade med förklaringar av parametrar och returvärden? För assemblerkod innebär detta också beskrivning av hur parametrar och returvärden överförs.
- Om uppgiften är att skriva C-kod, hur väl kopplas denna till de algoritmiska stegen i lösningen med hjälp av kommentarer? Dvs. om algoritmer är givna i uppgiften så ska dessa följas. Om konstruktion av algoritm ingår i lösningen ska denna först beskrivas.
- Om uppgiften är att skriva assemblerkod ska det finnas en tydlig koppling mellan de sekvenser av assemblerkoden och den C-kod (funktioner/variabeldeklarationer) som är assemblerkodens upphov.
- Det är tillåtet (inget krav) att använda kursens verktyg för att generera assemblerkod men tänk på att denna kod måste bearbetas och kommenenteras för att uppfylla kraven i bedömningskriterierna.

Tänk också på att en väl utförd modularisering av programmet tillsammans med väl valda funktionsnamn och variabelnamn i sig bidrar till "självdokumenterande kod" och kan minska behovet av explicit kommentering.

Bedömningsskala:

5. Högsta kvalitet och fullständig
4. Enstaka kvalitetsbrister men fullständig
3. Enstaka kvalitetsbrister, inte helt fullständig
2. Flera kvalitetsbrister och/eller ofullständighet
1. Stora kvalitetsbrister och/eller menlig ofullständighet
0. Dokumentation kan inte bedömas eller saknas helt

Relevans

Relevans baseras på kod *och* dokumentation och ska ge en sammanfattande bedömning av:

- Hur väl har uppgiften uppfattats och hur väl visar lösningen på en riktig förståelse?
- Finns speciella anvisningar för hur uppgiften ska lösas och hur har dessa i så fall efterlevts?

Relevans kan sällan bedömas fullt ut om kommentarer/dokumentation har stora brister eller saknas helt. I sådana fall når sällan denna bedömning över 1.

Bedömningsskala:

5. Högsta relevans
4. God relevans
3. Nöjaktig (godkänd) relevans
2. Bristfällig relevans
1. Mycket bristfällig relevans
0. Ej relevant/ej avgivet svar

Uppgift 1

Emilia får i uppgift att översätta följande funktion, given i C, till ARMv6 assemblerspråk.

```
int getItem( int i, int *first, int *second )
{
    int j = getSubItem( i );
    if( (i > 10) && (i < 30) )
        return first[j];
    return second[j];
}
```

Resultatet är följande assemblerprogram :

```
 1 @int getItem( int i, int *first, int *second )
 2 @{
 3 getItem:
 4     PUSH {R4,R5,LR}
 5     MOV  R0,R5
 6 @    int j = getSubItem( i );
 7     BL   getSubItem
 8     MOV  R4,R0
 9 @    if( i > 10 && (i < 30) )
10     CMP  R5,#10
11     BLE  L1
12     CMP  R5,#30
13     BGT  L1
14 @        return first[j];
15     MOV  R3,R4
16     LSL  R3,R3,#2
17     ADD  R0,R3,R1
18     LDR  R0,[R0]
19     B    L2
20 @    return second[j];
21 L1:
22     MOV  R3,R4
23     LSL  R3,R3,#2
24     ADD  R0,R3,R1
25     LDR  R0,[R0]
26 @}
27 L2:
28     POP  {R4,R5,PC}
```

Trots att Emilia har följt kursens anvisningar och tillämpat kompilatorkonventionerna för GCC blir det inte som Emilia tänkt sig. Det visar sig att Emilia gjort flera, mer eller mindre, allvarliga fel.

a) Din uppgift är att hjälpa Emilia genom att *hitta* raderna, som här har numrerats så att du säkert kan identifiera dom, för varje felaktig rad, *kommentera* (beskriv) felet och visa hur Emilia borde gjort (*rätta felet*), du får ange maximalt 3 rader. Varje enskilt fel kan alltså ge dig 3 poäng, 1 poäng för korrekt identifierad rad, ytterligare en poäng för beskrivning av felet och slutligen ytterligare en poäng för att du visar hur Emilia ska rätta felet.

b) Antag att följande deklarationer har gjorts:

```
int  length, result;
int  v1[10], v2[30];
```

Visa hur deklarationerna och följande subrutinanrop kodas i ARM v6 assemblerspråk (6p)

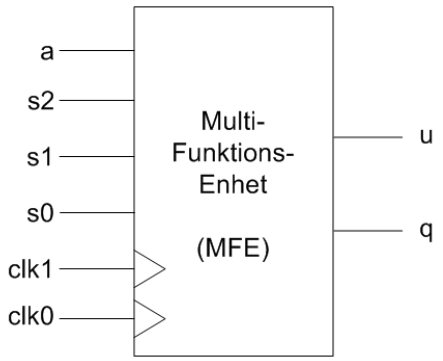
```
result = getItem( length, v1, &v2[10] );
```

Observera speciellt att för full poäng ska de kodningsanvisningar som givits i kursen följas.

Uppgift 2

Under kursen ”Grundläggande datorteknik” fick du lära dig hur en styrenhet är uppbyggd och hur den fungerar. Du fick också lära dig hur den konstrueras med hårdvara. Här ska du förbereda konstruktionen av en betydligt enklare multifunktionsenhet med såväl synkrona som asynkrona ingångar och utgångar.

En applikation för simulering av en multifunktionsenhet (MFE) med fyra asynkrona och två synkrona funktioner ska konstrueras med hjälp av en laborationsdator *MD407*. Följande gäller för MFE:ns funktioner:

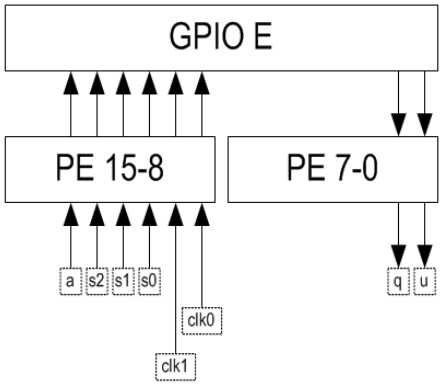


s2	s1	s0	u
0	0	0	0
0	0	1	a
0	1	0	!a
0	1	1	a
1	0	0	1
1	0	1	a
1	1	0	0
1	1	1	a

clk0, clk1	q
positiv flank clk0	a
negativ flank clk0	s2&&s1&&s0
positiv flank clk1	!a
negativ flank clk1	s2||s1||s0

Dvs. de asynkrona ingångarna a, s0,s1 och s2 bestämmer utsignalen u via logikfunktioner. Den synkrona utsignalen q påverkas av clk0 repektive clk1, beroende på flank (positiv eller negativ). GPIO-port E används för MFE:ns anslutningar enligt följande:

u ansluts till PE0
q ansluts till PE1
clk0 ansluts till PE8
clk1 ansluts till PE9
s0 ansluts till PE10
s1 ansluts till PE11
s2 ansluts till PE12
a ansluts till PE13



Det synkrona beteendet ska implementeras med hjälp av avbrott (positiv och negativ flank på klocksignalen). Alla utgångar ska vara *push/pull* och alla ingångar *pull down*.

a) Visa en initieringsfunktion
`void init_app(void)` som:
- Initierar port E för den önskade funktionen.
- Initierar systemet för att hantera avbrott med funktionerna `clk0_handler`, `clk1_handler` (se nedan).

b) Visa två avbrottsfunktioner
`void clk0_handler (void)` och `void clk1_handler (void)` som:
- Läser insignalerna från portpinnar, bestämmer utsignalen q och skriver till portpinnen.

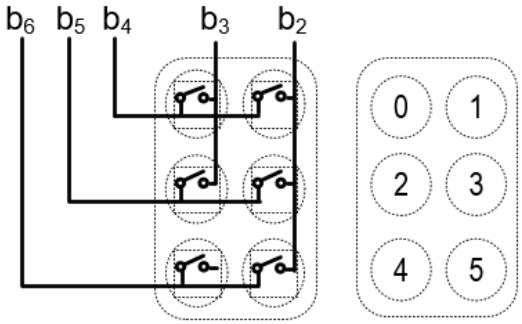
c) Visa en funktion `char evaluate( char a, char s )` som:
- bestämmer utsignalen u enligt funktionstabellen och returnerar detta värde.

d) Visa ett huvudprogram `main` som kontinuerligt:
- Läser de asynkrona insignalerna, anropar `evaluate` för evaluering av u och skriver resultatet till portpinnen.

Implementera koden i programspråket C.

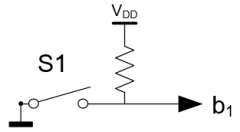
Uppgift 3

Ett tangentbord för inmatning av olika tecken ska konstrueras. Sex stycken återfjädrande omkopplare ansluts därför till port D hos *MD407*, enligt figuren till höger. En nedtryckt tangent ska alltså representera ett värde 0-5.



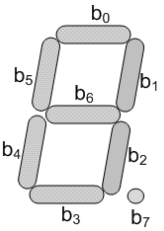
Nedtryckta tangenter kan detekteras genom att '1' skrivs till någon av bit 6, bit 5 eller bit 4, därefter avläses bit 3 respektive bit 2. För att detta ska vara tillförlitligt måste dessa bitars ingångar förses med "*pull-down*" samtidigt som utgångarna ska vara "*push-pull*".

Utöver tangentbordet ansluts en strömställare, S1 med "pull-up"-resistor. Via b_1 detekteras om S1 är öppen eller sluten.



Visningsenhet

Utöver detta ansluts en utmatningsenhet i form av en Sju-sifferindikator där de enskilda segmenten tänds av '1' i en motsvarande sifferposition, se figur till höger. Indikatorn används för visning av en hexadecimal siffra och ansluts till port D b8-b15.



a) Visa med en funktion `void gpio_init(void)` hur port D ska initieras för användning med tangentbord och visningsenhet.

b) Visa en tangentbordsfunktion `unsigned char get(void)`, som returnerar tangentbordets tillstånd. Detta kan vara 0-5 om en tangent tryckts ned, 0xE om ingen tangent tryckts ned eller 0xF om tangentbordet är låst. Om flera tangenter trycks ned samtidigt ska en av tangentkoderna (godtyckligt vilken) returneras. Strömbrytare S1 används för att låsa och låsa upp tangentbordet. Tangentbordet ska vara låst då S1 är sluten.

c) Skriv ett huvudprogram som kontinuerligt skriver ut information från tangentbordet i form av en hexadecimal siffra. Värdet från funktionen `get` ska först översättas till segmentskod för sjusifferindikatorn, därefter skrivs segmentskoden tillutmatningsenheten.

I din lösning ska du använda *lämpliga makrodefinitioner för registeradresser*.

Uppgift 4

En ny typ av grafisk display, med basadressen 0xF0000000, har konstruerats för MD407. Displayen är av en komplex typ med såväl beräkningskapacitet som ett stort bildminne. Olika geometrier kan exempelvis ritas via enkla kommandon till displayen och detta gör den väldigt enkel att programmera. Följande register utgör displayens programmerargränssnitt:

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0																																	CTRSTAT
4																																	RGB
8																																	COORD1
0x0C																																	COORD2

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x00	BUSY									CMD					ABT	EX	CTRSTAT

EX: Execute, skrivbar, vid övergång 0→1 hos denna bit startas det kommando som finns i CMD. Biten BUSY i statusregistret sätts av hårdvaran till 1 och återställs då hela kommandot utförts. Övergången 1→0 har ingen effekt.
ABT: Abort, skrivbar, avbryt ett kommando under exekvering.
CMD: Command, läs- och skrivbar, anger vilket kommando som ska utföras vid nästa Execute.

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
4	BGINT								RED								GREEN								BLUE								RGB

BGINT: Background Intensity, displayens bakgrund, gråskala från svart (0) till vit (0xFF).
RED, GREEN, BLUE: Palett, intensitet (24-bit RGB) hos förgrundsfärgen.

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
8							YCOORD																XCOORD										COORD1

YCOORD: Y-kordinat för punkt 1
XCOORD: X-koordinat för punkt 1

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0xC							YCOORD																XCOORD										COORD2

YCOORD: Y-kordinat för punkt 2
XCOORD: X-koordinat för punkt 2

De kommandon som kan utföras framgår av tabellen på nästa sida.

En geometri som man tidigare ritat kan ”raderas” genom att man ritar exakt samma geometri en gång till, men med samma färg som bakgrunden.

Under laborationerna har du provat på att göra enkla animeringar, en vanlig animeringstakt är 25 bilder per sekund, det innebär att varje bild ska visas under c:a 40 millisekunder.

a) Visa en typdeklaration `GAMEDISPLAY` med **bitfält** som representerar registermodulen.

b) Visa en funktion:
`void ellipse(int xorigo,int yorigo, int xwidth, int ywidth, int draw , int filled );`
som ritar/raderar en ellips på displayen. Om parametern `draw` är 0, raderas ellipsen annars ritas den. Om parametern `filled` är 0 ritas bara ellipsens kontur, annars fylls också ellipsen.

c) Visa en funktion `main`, som animerar en "pulserande" fylld röd cirkel mot svart bakgrund. Minsta radien är 10 pixels, största radien är 100 pixels. Varje radie visas under en "bild". Origo ska vara punkt 127,127 på displayen. SYSTICK ska användas för att skapa en fördröjningsfunktion "`delay`".

Kommando						Beskrivning
Clear display	0	0	0	0	0	Displayen raderas och tänds baserat på bakgrundsintensitet.
Display size 256x256	0	0	0	0	1	
Display size 512x512	0	0	0	1	0	
Display size 1024x1024	0	0	0	1	1	
	0	0	1	0	0	
Pixel	0	0	1	0	1	Rita pixel med färg RGB i COORD1
Clear pixel	0	0	1	1	0	Rita pixel med bakgrundsintensitet i COORD1
Line	0	0	1	1	1	Rita linje med färg RGB från COORD1 till COORD2
Clear line	0	1	0	0	0	Rita linje med bakgrundsintensitet från COORD1 till COORD2
Rectangle	0	1	0	0	1	Rita rektangel med färg RGB och motstående hörnen COORD1, COORD2
Clear rectangle	0	1	0	1	0	Rita rektangel med bakgrundsintensitet och motstående hörnen COORD1, COORD2
Filled rectangle	0	1	0	1	1	Rita fylld rektangel med färg RGB och motstående hörnen COORD1, COORD2
Clear filled rectangle	0	1	1	0	0	Rita fylld rektangel med bakgrundsintensitet och motstående hörnen COORD1, COORD2
Ellipse	0	1	1	0	1	Rita ellips med färg RGB, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)
Clear ellipse	0	1	1	1	0	Rita ellips med bakgrundsintensitet, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)
Filled ellipse	0	1	1	1	1	Rita fylld ellips med färg RGB, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)
Clear filled ellipse	1	0	0	0	0	Rita fylld ellips med bakgrundsintensitet, centrum i COORD1, bredd i COORD2(x) och höjd i COORD2(y)

Lösningsförslag:

Uppgift 1:

a) Den korrekta koden ska utformas enligt följande:

```
 1  @int getItem( int i, int *first, int *second )
 2  @{
 3  getItem:
 4      PUSH {R4,R5,LR}
 5      MOV  R5,R0
 6  @    int j = getSubItem( i );
 7      BL   getSubItem
 8      MOV  R4,R0
 9  @    if( i > 10 && (i < 30) )
10      CMP  R5,#10
11      BLE  L1
12      CMP  R5,#29
13      BGT  L1
14  @        return first[j];
15      MOV  R3,R4
16      LSL  R3,R3,#2
17      ADD  R0,R3,R1
18      LDR  R0,[R0]
19      B    L2
20  @    return second[j];
21  L1:
22      MOV  R3,R4
23      LSL  R3,R3,#2
24      ADD  R0,R3,R2
25      LDR  R0,[R0]
26  @}
27  L2:
28      POP  {R4,R5,PC}
```

Emilia gjorde fel i raderna:
Rad 5, förväxlat käll och destinationsregister.
Rad 12, fel konstant i jämförelse *alternativt*
Rad 13, fel villkorlig instruktion ska vara BGE
Rad 24, fel register i källoperand 2, ska vara R2 (&second)
Inför anropet av getSubItem (rad 7) måste dessutom R1 och R2 sparas (för att sedan återställas) efter funktionen (ny rad 8).

b)

```
length: .SPACE 4
result: .SPACE 4
v1: .SPACE 40
v2: .SPACE 120

   LDR  R0,length
   LDR  R1,=v1
   LDR  R2,=v2
   ADD  R2,R2,#40
   BL   getItem
   LDR  R1,=result
   STR  R0,[R1]
```

Uppgift 2:

```
#define GPIO_E_BASE      0x40021000      /* MD407 port E */
#define GPIO_E_MODER     ((volatile unsigned int *)  (GPIO_E_BASE))
#define GPIO_E_OTYPER    ((volatile unsigned short *)(GPIO_E_BASE+0x4))
#define GPIO_E_PUPDR     ((volatile unsigned int *)  (GPIO_E_BASE+0xC))
#define GPIO_E_IDRHIGH   ((volatile unsigned char *) (GPIO_E_BASE+0x10+1))
#define GPIO_E_ODRLOW    ((volatile unsigned char *) (GPIO_E_BASE+0x14))


#define SYSCFG_EXTICR3  ((volatile unsigned int *)  0x40013810)
#define EXTI_IMR        ((volatile unsigned int *)  0x40013C00)
#define EXTI_FTSR       ((volatile unsigned int *)  0x40013C0C)
#define EXTI_RTSR       ((volatile unsigned int *)  0x40013C08)
#define EXTI_PR         ((volatile unsigned int *)  0x40013C14)
#define NVIC_ISER0      ((volatile unsigned int *)  0xE000E100)
#define NVIC_IABR0      ((volatile unsigned int *)  0xE000E200)

#define NVIC_EXTI9_IRQ_BPOS (1<<15)
#define EXTI9_IRQ_BPOS      (1<<9)
#define NVIC_EXTI8_IRQ_BPOS (1<<14)
#define EXTI8_IRQ_BPOS      (1<<8)

#define EXTI8_9_IRQVEC    ((volatile unsigned int *) 0x2001C09C)

void    init_app( void )
{
   *GPIO_E_MODER = 0x00000005;
   *GPIO_E_PUPDR = 0x0AAA0000;
   *GPIO_E_OTYPER= 0x00000000;
   *SYSCFG_EXTICR3 |= 0x0044;
   *EXTI_IMR |= (EXTI8_IRQ_BPOS | EXTI9_IRQ_BPOS);
   *((void (**)(void) ) EXTI8_9_IRQVEC ) = clk_handler;
   *EXTI_RTSR |= EXTI8_IRQ_BPOS | EXTI9_IRQ_BPOS;
   *EXTI_FTSR |= EXTI8_IRQ_BPOS | EXTI9_IRQ_BPOS;
   *NVIC_ISER0 |= (NVIC_EXTI8_IRQ_BPOS | NVIC_EXTI9_IRQ_BPOS);
}
```

```
 char    evaluate( char a, char s)
 {
      switch( s )
     {
        case 0: case 6: return 0;
        case 1: case 3: case 5: case 7: return a;
        case 2: return !a;
        case 4: return 1;
     }
 }


 void main(void)
 {
 char a,s,u;
 init_app();
     while( 1 )
     {
         a = *GPIO_E_IDRHIGH & 0x20;
         s = (*GPIO_E_IDRHIGH >>2) & 7;
         u = evaluate(a,s);
         if( u )
             *GPIO_E_ODRLOW |= 1;
         else
             *GPIO_E_ODRLOW &= ~1;
     }
 }
 void clk_handler ( void )
 {
     if( *NVIC_IABR0 & NVIC_EXTI8_IRQ_BPOS )
     {
         clk0_handler();
     }else{
         clk0_handler();
     }
 }
```

```
void clk0_handler ( void )
{
    if( *GPIO_E_IDRHIGH & 1 )
    {
        if(*GPIO_E_IDRLOW & 0x10)
            *GPIO_E_ODRLOW |= 2;
        else
            *GPIO_E_ODRLOW &= ~2;
    }
    else{
        if( ((*GPIO_E_IDRHIGH >>2) & 7) ==7)
            *GPIO_E_ODRLOW |= 2;
        else
            *GPIO_E_ODRLOW &= ~2;
    }
    *EXTI_PR |= EXTI8_IRQ_BPOS;
}

void clk1_handler ( void )
{
    if( *GPIO_E_IDRHIGH & 1 )
    {
        if(*GPIO_E_IDRLOW & 0x10)
            *GPIO_E_ODRLOW &= ~2;
        else
            *GPIO_E_ODRLOW |= 2;
    }
    else{
        if( (*GPIO_E_IDRHIGH >>2) & 7)
            *GPIO_E_ODRLOW |= 2;
        else
            *GPIO_E_ODRLOW &= ~2;
    }
    *EXTI_PR |= EXTI9_IRQ_BPOS;
}
```

Uppgift 3 a

```
/* Port E */
#define    PORT_D_BASE    0x40020C00
#define    GPIO_D_MODER   ((volatile unsigned int *)    (PORT_D_BASE))
#define    GPIO_D_OTYPER  ((volatile unsigned short *)  (PORT_D_BASE+0x4))
#define    GPIO_D_OSPEEDR ((volatile unsigned int *)    (PORT_D_BASE+0x8))
#define    GPIO_D_PUPDR   ((volatile unsigned int *)    (PORT_D_BASE+0xC))
#define    GPIO_D_IDRLOW  ((volatile unsigned char *)   (PORT_D_BASE+0x10))
#define    GPIO_D_IDRHIGH ((volatile unsigned char *)   (PORT_D_BASE+0x11))
#define    GPIO_D_ODRLOW  ((volatile unsigned char *)   (PORT_D_BASE+0x14))
#define    GPIO_D_ODRHIGH ((volatile unsigned char *)   (PORT_D_BASE+0x15))

void init_gpio( void )
{
    *GPIO_D_MODER = 0x55550540;
    *GPIO_D_PUPDR = 0x000000A0;
    *GPIO_D_OTYPER= 0x00000000;
}
```

Uppgift 3 b

```
unsigned char get( void )
{
    if( *GPIO_D_IDRLOW & 2 )
        return 0xF;
    *GPIO_D_ODRLOW = 0x10;
    if( *GPIO_D_IDRLOW & 8 )
        return 0;
    if( *GPIO_D_IDRLOW & 4 )
        return 1;
    *GPIO_D_ODRLOW = 0x20;
    if( *GPIO_D_IDRLOW & 8 )
        return 2;
    if( *GPIO_D_IDRLOW & 4 )
        return 3;
    *GPIO_D_ODRLOW = 0x40;
    if( *GPIO_D_IDRLOW & 8 )
        return 4;
    if( *GPIO_D_IDRLOW & 4 )
        return 5;
    return 0xE;
}
```

Uppgift 3 c

```
void main(void)
{
static unsigned char segCode[]={
    0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,
    0x7F,0x67,0x77,0x7C,0x39,0x5E,0x79,0x71 };

    init_gpio();
    while( 1 )
    {
        *GPIO_E_ODRHIGH = seg_tab[ get() ];
    }
}
```

Uppgift 4:

a)

```
typedef struct gamedisplay
{
    unsigned int ex:1;
    unsigned int abt:1;
    unsigned int cmd:5;
    unsigned int :8;
    unsigned int busy:1;

    unsigned int :0;
    unsigned int blue:8;
    unsigned int green:8;
    unsigned int red:8;
    unsigned int bgint:8;

    unsigned int :0;
    unsigned int xcoord1:10;
    unsigned int :6;
    unsigned int ycoord1:10;

    unsigned int :0;
    unsigned int xcoord2:10;
    unsigned int :6;
    unsigned int ycoord2:10;

} GAMEDISPLAY;
```

b)

```
#define R ((GAMEDISPLAY *) 0xF0000000)
void rect(int xorigo,int yorigo,
       int xwidth, int ywidth,
       int draw , int filled )
{
    R->xcoord1 = xorigo;
    R->ycoord1 = yorigo;
    R->xcoord2 = xwidth+x;
    R->ycoord2 = ywidth+y;
    if( filled )
    {
        if( draw )
            R->cmd = 11;
        else
            R->cmd = 12;

    }else{
        if( draw )
            R->cmd = 9;
        else
            R->cmd = 10;
    }
    R->ex = 0;
    R->ex = 1;
}
```

c)

```
/* SysTick */
#define     STK_CTRL    ((volatile unsigned int *)(0xE000E010))
#define     STK_LOAD    ((volatile unsigned int *)(0xE000E014))
#define     STK_VAL ((volatile unsigned int *)(0xE000E018))

#define DELAY_VAL   168000/4-1
void delay( void )
{
    /* SystemCoreClock = 168000000 */
    *STK_CTRL = 0;
    *STK_LOAD = DELAY_VAL;
    *STK_VAL = 0;
    *STK_CTRL = 5;
    while( (*STK_CTRL & 0x10000 )== 0 );
    *STK_CTRL = 0;
}


#define DRAW_FILLED_ELLIPSE(a,b,c,d)    ellipse(a,b,c,d,1,1)
#define CLEAR_FILLED_ELLIPSE(a,b,c,d)   ellipse(a,b,c,d,0,1)

void main(void)
{
    int sx,xy;
    R->bgint = 0;
    R->cmd = 0;
    R->ex = 0;
    R->ex = 1;  while(1)
    {
        R->red = 255;
        R->green = 0;
        R->blue = 0;
        for( int i = 10; i<100;i++)
        {
            DRAW_FILLED_ELLIPSE(127-i/2,127-i/2,i,i);
            delay();
            CLEAR_FILLED_ELLIPSE(127-i/2,127-i/2,i,i);
        }
        for( int i = 100; i>10;i--)
        {
            DRAW_FILLED_ELLIPSE(127-i/2,127-i/2,i,i);
            delay();
            CLEAR_FILLED_ELLIPSE(127-i/2,127-i/2,i,i);
        }
    }
}
```