Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Hemtentamen med delvisa lösningsförslag

EDA482 Maskinorienterad programmering D
EDA487 Maskinorienterad programmering Z
DIT151 Maskinorienterad programmering GU
DAT017 Maskinorienterad programmering IT
DAT390 Maskinorienterad programmering Hing
LEU500 Maskinorienterad programmering Hing

Måndag 15 mars 2021, kl. 14.00 - 18.00 (20.00)

Examinator
Erik Sintorn
Pedro Trancoso

Kontaktpersoner under tentamen:
Erik Sintorn, epost (Canvas)
Pedro Trancoso, 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. (18.10) inte kommer att bedömas. (Betraktas som blank inlämning). Tentander som beviljats förlängd skrivtid lämnar in senast kl. 20.10.

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

För EDA482/EDA487/DIT151, lp4 vt2020

Tentamensbetyg ges av hemtentamen.

Som slutbetyg på kursen ges en sammanfattande bedömning baserad på betyg från hemtentamen respektive betygsbedömd laboration förutsatt att båda dessa moment är godkända (minst betyg 3).

För omtentamina:

Tentamensbetyg ges av hemtentamen.

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

Bedömningskriterier 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

Koden nedan gör en strängkonkatenering. mystrcat lägger, i ousträngen out, resultatet av sammanfogningen av de två insträngarna, in1 och in2. Om du t.ex.kallar mystrcat med in1="Hello" och in2="world", så skall resultatet i utsträngen vara "Hello World".

```
void mystrcat(char *out, char *in1, char *in2) {
   int i = 0;
   while(*in1)
     out[i++] = *in1++;
   while(*in2)
     out[i++] = *in2++;
   out[i] = '\0';
}

char total[100];

int main(void) {
   char *str1 = "Hello";
   char *str2 = " world!!";
   mystrcat(total, str1, str2);
}
```

Om vi kompilerar koden med en online ARM-kompilator (t.ex. https://godbolt.org) får vi följande assembly kod:

```
mystrcat(char*, char*, char*):
   str  fp, [sp, #-4]!
   add  fp, sp, #0
   sub  sp, sp, #28
   str  r0, [fp, #-16]
   str  r1, [fp, #-20]
   str  r2, [fp, #-24]
   mov  r3, #0
   str  r3, [fp, #-8]
.L3:
   ldr  r3, [fp, #-20]
   ldrb r3, [r3]  @ zero_extendqisi2
   cmp  r3, #0
   beq  .L2
   ldr  r3, [fp, #-20]
   add  r2, r3, #1
   str  r2, [fp, #-20]
   ldr  r2, [fp, #-8]
   add  r1, r2, #1
   str  r1, [fp, #-8]
   mov  r1, r2
   ldr  r2, [fp, #-16]
   add  r2, r2, r1
   ldrb r3, [r3]  @ zero_extendqisi2
   strb r3, [r2]
   b    .L3
.L2:
   ldr  r3, [fp, #-24]
   ldrb r3, [r3]  @ zero_extendqisi2
   cmp  r3, #0
   beq  .L4
   ldr  r3, [fp, #-24]
   add  r2, r3, #1
   str  r2, [fp, #-24]
   ldr  r2, [fp, #-8]
   add  r1, r2, #1
   str  r1, [fp, #-8]
   mov  r1, r2
   ldr  r2, [fp, #-16]
   add  r2, r2, r1
   ldrb r3, [r3]  @ zero_extendqisi2
   strb r3, [r2]
   b    .L2
.L4:
   ldr  r3, [fp, #-8]
   ldr  r2, [fp, #-16]
   add  r3, r2, r3
   mov  r2, #0
   strb r2, [r3]
   nop
   add  sp, fp, #0
   ldr  fp, [sp], #4
   bx   lr
total:
.LC0:
.ascii  "Hello\000"
.LC1:
.ascii  " world!!\000"
main:
   push {fp, lr}
   add  fp, sp, #4
   sub  sp, sp, #8
   ldr  r3, .L7
   str  r3, [fp, #-8]
   ldr  r3, .L7+4
   str  r3, [fp, #-12]
   ldr  r2, [fp, #-12]
   ldr  r1, [fp, #-8]
   ldr  r0, .L7+8
   bl   mystrcat(char*, char*, char*)
   mov  r3, #0
   mov  r0, r3
   sub  sp, fp, #4
   pop  {fp, lr}
   bx  lr
.L7:
.word  .LC0
.word  .LC1
.word  total
```

a) Den här koden är betydligt svårlästare än vad vi har visat i kursen. Gör din egen kompilering av koden för hand och gör den mer lättläst genom att följa konventionerna du har lärt dig i kursen. Till exempel använder den kompilerade koden `fp` och `sp` på ointuitiva sätt som vi inte gjort i kursen. Du bör använda register för parametrarna och för returvärdet. Kom också ihåg konventionerna för registeranvändning.

b) Ändra C-koden för `mystrcat` funktionen så att den tar som extra parameter en bitvektor, `bitvec`, med en bit för varje bokstav i inputsträngen `in2`. Funktionen skall nu testa verje bit i `bitvec` och om biten är 1 skall motsvarande bokstav i `in2` läggas till resultatet men om biten är 0 skall bokstaven ignoreras. För enkelhets skull begränsar vi maxlängden på `in2` till 32 tecken, så att `bitvec` kan vara en unsigned int. Om vi i exemplet ovan skickar med en bitvektor som är 0x7 (00000111) så skall resultatet av konkateneringen bli ”Hello wo”. Om vi använde en bitvektor 0x20 (00100000) så blir konkateneringen ”Hellod”.

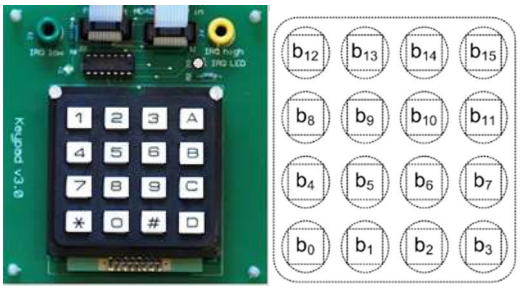
Uppgift 2

Under laborationerna konstruerade du en enkel tangentbordsrutin som avspeglar ett ”ögonblicks-värde”. En nedtryckt tangent upptäcks av en funktion som kontinuerligt undersöker tangentbordet.

I vissa fall räcker det inte med att kunna detektera exakt *en* nedtryckt tangent. Man vill i bland kunna detektera flera samtidigt nedtryckta tangenter, som exempelvis Ctrl- eller Alt- funktioner hos ett vanligt tangentbord. I denna uppgift ska du 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 figur 1 nedan, mellan bitposition och tangent:

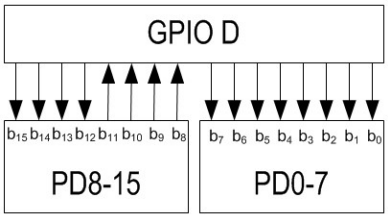
bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Kod (hex)	0A	03	02	01	0B	06	05	04	0C	09	08	07	0D	0F	00	0E

En funktion för inmatning från en keypad (figur 1), en funktion för konvertering av det inmatade värden och slutligen utskrift till en 7-segments display (figur 2) ska konstrueras. Enheterna ansluts till *MD407* port D enligt figur 3. Alla funktioner implementeras i programspråket C och varje deluppgift kan ge maximalt 5 p.



Figur 1: Keypad och tangentkoder

Figur 2

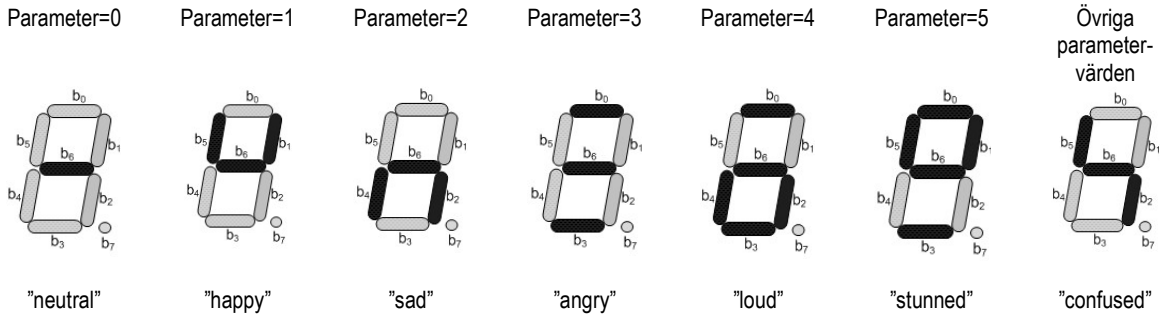


Figur 3

a) Implementera funktionen `unsigned short keyb_alt_ctrl(void).` Funtionen ska utformas så att varje nedtryckt tangent indikeras med en bit i ett statusord som returneras av rutinen. Implementera också funktionen `void init_gpio(void)` som initierar port D enligt figur 3.

b) Konstruera en funktion `unsigned char is_numeric( unsigned short s )` som avgör om någon av tangenterna 0 till 9 är nedtryckt baserat på statusordet i parametern `s`.
- Om någon av de numeriska tangenterna 0..9 indikeras, returneras koden (0..9) för denna. Om flera numeriska tangenter är nedtryckta samtidigt ska den lägsta koden returneras.
- Funktionen ska ignorera icke-numeriska tangenter (tangentkoder 0x0A..0x0F).
- Om ingen numerisk tangent är nedtryckt ska 0xFF returneras.

c) Under laborationerna skapade du också en funktion för utmatning av hexadecimala siffror till en 7-segments display. Genom att modifiera segmentskodstabellen kan vi skapa nya ”tecken-uppsättningar. Konstruera en funktion `void outEmoji( unsigned char c )` som matar ut en av följande figurer baserat på parameter `c`:



Uppgift 3

Ett *stoppur* ska konstrueras. För detta använder vi en MD407 och två typer av laborationskort: IRQ Flip Flop med en inbyggd räknarkrets och 2 st visningsenheter 7 segment display, se figur 1.

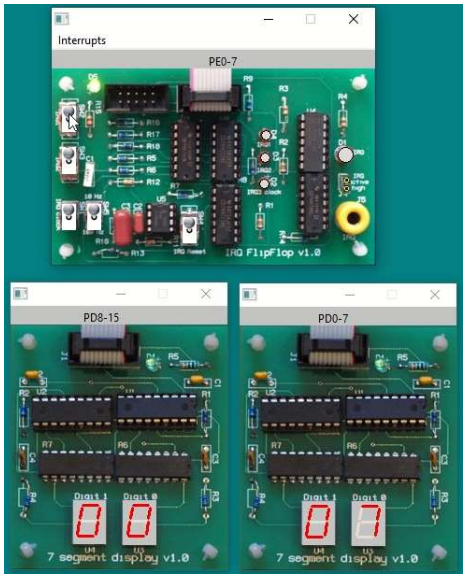
Stoppuret kontrolleras med två brytare. Överst till vänster finns *start/stopp*-funktionen. Första nedtryckningen startar klockan, nästa stoppar den, ytterligare nedtryckning startar den igen osv.

Nedanför denna brytare finns *återställningsfunktionen*. En tryckning på denna knapp återställer tiden till 00.00.

För tidmätningen används laborationskortets inbyggda räknarkrets med frekvensen 100 Hz.

Stoppurets mätperiod är max 60 sekunder och upplösningen 1/100-dels sekund. Tiden visas på de fyra sifferindikatorerna från vänster enligt:

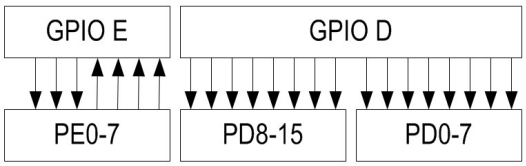
- 10-tals sekunder
- sekunder
- tiondels sekunder
- hundradels sekunder



Figur 1

Figur 2 visar hur laborationskorten ansluts till laborationsdatorn.

Alla *händelser* dvs. knapptryckningar och indikation av periodintervall från räknarkretsen, ska hanteras med processorns *avbrottsmekanismer*.



Figur 2

- Om mätningen uppnår maximal tid (59.99) ska den stannas och stå kvar i detta läge ända tills återställningsfunktionen aktiveras.
- Vid en återställning nollställs tiden oavsett om stoppuret har startats eller ej.

a) Skriv lämpliga makrodeklarationer för hanteringen av laborationskorten
b) Skriv en initieringsfunktion `init_app`, som initierar portar och laborationskort.
c) Definiera variabler, skriv en avbrottsfunktion och ett huvudprogram för applikationen.

Uppgift 4

Den första bemannade mars-landaren är bestyckad med en MD407 som skall ta hand om landningen. Det är din uppgift att programmera den. Maskinen har en extra periferikrets för att kontrollera landningsraketerna, som ser ut som följer:

Mars Lander Module (0xE000E0F0)																	
Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x00														w	w	w	ML_CTRL
0x02	ALT (r)								STATE (rw)								ML_STATE
0x04																	ML_LS
0x06	TILT								VEL								ML_ORI
0x08															rw	rw	ML_IRQ

Control Register (ML_CTRL):
Bit 0 (STAB): När en etta skrives aktiveras stabiliserings raketer kort, vilket rättställer landarens vinkel.
Bit 1 (BRAKE): När en etta skrives aktiveras bromsraketerna kort, och landarens hastighet minskar.
Bit 2 (PANIC): När en etta skrives aktiveras nödprotokollet och landaren försöker ta sig tillbaks till moderskeppet.

State Register (ML_STATE):
Bits 15:8 (ALT): Anger raketens altitud (i meter).
Bits 7:0 (STATE): Anger maskinens nuvarande state.

State 0xFF: “Docked”. Initialstatet.

State 0: ”Free fall” state:
Landarens enda uppgift är att upprätthålla sin vinkel mot marken genom att aktivera stabiliseringsraketer. När detta state sätts släpps landaren från moderskeppet. *Innan dess får inga raketer aktiveras*.

State 1: ”Landing” state:
Landarens hastighet (VEL) skall upprätthållas så att den är densamma som altituden (ALT), genom att vid behov aktivera bromsraketer.

State 2: ”Touchdown” state: Sätts när landaren landat (ALT=0).

Life Support (ML_LS) [DEPRECATED]:
Life support hanteras numera av ett separat system. Registret får under inga omständigheter läsas eller skrivas!

Orientation Register (ML_ORI):
Bits 15:8 (TILT): Anger vinkeln mot marken. Skall helst vara 0. Om den överskrider 20 genereras ett PANIC avbrott.
Bits 7:0 (VEL): Anger fallhastigheten i Km/h. Skall, när landaren är i ”landing” state, vara lika med ALT. Vid för hög hastighet genereras ett PANIC avbrott.

Interrupt Request Register (ML_IRQ):
Bit 0 (ATMO): Sätts till 1 när landaren är tillräckligt nära marken för att påbörja ”landing” state.
Bit 1 (PANIC): Sätts till 1 när landaren fått för hög vinkel mot marken eller för hög hastighet.

Vektortabellen har utökats med följande interrupt:

IRQ num	priority	name	description	vector offset
61	settable	ML_ATMO	Landaren skall sättas i ”landing” state	0x00000134
62	settable	ML_PANIC	Landaren skall aktivera nödprotokollen	0x00000138

Till skillnad från labbsystemen kan vi inte vara säkra på vilken plats vektortabellen ligger på, utan måste hämtas från System Control Block (SCB) registret.

Om interruptet ML_PANIC inträffar måste PANIC biten i ML_CTRL genast sättas, oavsett andra interrupts.

a) Skriv en C struct som representerar marslandarmodulens register.
b) Skriv en `init` function som initierar ML modulens register och interupt.
c) Skriv programmet och interrupthanteraren som säkert landar besättningen.
    a. Så länge landaren befinner sig i ”Free fall” state skall stabiliseringsraketer aktiveras när TILT > 0 (men inte annars, för att spara rymdbränsle)
    b. När ett ML_ATMO interrupt kommer skall landaren övergå till ”Landing state”. Då skall bromsraketerna aktiveras, närhelst VEL != ALT.
    c. När ALT = 0 skall raketen sättas i ”Touchdown” state.
    d. Närhelst ett PANIC interrupt inträffar skall PANIC biten i ML_CTRL omeddelbart sättas.

Lösningsförslag:
En lösning tas fram exempelvis genom att kompilera C-koden till assembler med GCC/ARM-kompilatorn. Den resulterande koden är dock väldigt svårläst och måste modifieras för hand. Kravet att lokala variabler ska registerallokeras leder till att åtskilliga load/store försvinner.
En lösning som bara innehåller "dump" av kompilerad kod bedöms 2/0/1.

Uppgift 1:

a)

```
mystrcat:
   push {r4,r5}
   mov  r3, #0    @ r3 stores i
.L3:
   ldrb r4, [r1]     @ r4 = *in1
   cmp  r4, #0
   beq  .L2
   add  r5, r0, r3
   strb r4, [r5]
   add  r3, r3, #1
   add  r1, r1, #1
   b    .L3
.L2:
   ldrb r4, [r2]  @ r4 = *in2
   cmp  r4, #0
   beq  .L4
   add  r5, r0, r3
   strb r4, [r5]
   add  r3, r3, #1
   add  r2, r2, #1
   b    .L2
.L4:
   mov  r4, #0
   add  r5, r0, r3
   strb r4, [r5]
   pop  {r4, r5}
   bx   lr
```

```
total:
.LC0:
.ascii  "Hello\000"
.LC1:
.ascii  " world!!\000"
main:
   push lr
   ldr  r0, =total
   ldr  r1, =.LC0
   ldr  r2, =.LC1
   bl   mystrcat
   bx   lr
.L7:
.word  .LC0
.word  .LC1
.space total 100
```

b)

```
void mystrcat(char *out, char *in1, char *in2, unsigned int bitvec) {
   int i = 0;
   unsigned int mask =0x1;
   while(*in1)
      out[i++] = *in1++;
   while(*in2) {
      if( bitvec & mask ) {
         out[i] = *in2;
         i++;
      }
      in2++;
      mask = mask << 1;
   }
   out[i] = '\0';
}

char total[100];

int main(void) {
   char *str1 = "Hello";
   char *str2 = " world!!";
   unsigned int bitv = 0x7;
   mystrcat(total, str1, str2, bitv);
}
```

Uppgift 2:

Definitioner och funktioner i vänstra kolumnen är kopierade direkt från laboration 2

```
/* Port D */
#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))


unsigned short keyb_alt_ctrl(void)
{
    int row;
    unsigned short pad;
    *GPIO_D_ODRHIGH = 0x80;
    pad = (*GPIO_D_IDRHIGH & 0xF);
    *GPIO_D_ODRHIGH = 0x40;
    pad |= ((*GPIO_D_IDRHIGH & 0xF)<<4);
    *GPIO_D_ODRHIGH = 0x20;
    pad |= ((*GPIO_D_IDRHIGH & 0xF)<<8);
    *GPIO_D_ODRHIGH = 0x10;
    pad |= ((*GPIO_D_IDRHIGH & 0xF)<<12);
    *GPIO_D_ODRHIGH = 0;
    return pad;
}
void init_app( void )
{
    *GPIO_D_MODER = 0x55005555;
    *GPIO_D_PUPDR = 0x00AA0000;
    *GPIO_D_OTYPER= 0x00000000;
}

#define NUMERIC 0x7772
unsigned char is_numeric( unsigned short s )
{
    switch( s & NUMERIC )
    {
        case 0x1000: return 1;
        case 0x2000: return 2;
        case 0x4000: return 3;
        case 0x0100: return 4;
        case 0x0200: return 5;
        case 0x0400: return 6;
        case 0x0010: return 7;
        case 0x0020: return 8;
        case 0x0040: return 9;
        case 0x0002: return 0;
    }
    return 0xFF;
}


void outEmoji( unsigned char c )
{
    unsigned char code;
    unsigned char segCode[]={
        0x40,0x62,0x54,0x49,0x5D,0x6B };
    if( c >= 6){
        code = 0x64;
    }else
        code = segCode[c];
    *GPIO_D_ODRLOW = code;
}
```

Uppgift 3:

```
#define SYSCFG_BASE     ((volatile unsigned int *)   (0x40013800))
#define SYSCFG_EXTICR1  ((volatile unsigned int *)   (0x40013808))
#define EXTI_BASE       ((volatile unsigned int *)   (0x40013C00))
#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_ICPR0      ((volatile unsigned int *)  (0xE000E280))
#define NVIC_IPR0       ((volatile unsigned int *)  (0xE000E400))

#define EXTI3_IRQVEC    0x2001C064
#define NVIC_EXTI3_IRQ_BPOS  (1<<9)
#define EXTI3_IRQ_BPOS  (1<<3)

/* Port E */
#define     PORT_E_BASE     0x40021000
#define     GPIO_E_MODER    ((volatile unsigned int *)  (PORT_E_BASE))
#define     GPIO_E_OTYPER   ((volatile unsigned short *)(PORT_E_BASE+0x4))
#define     GPIO_E_OSPEEDR  ((volatile unsigned int *)  (PORT_E_BASE+0x8))
#define     GPIO_E_PUPDR    ((volatile unsigned int *)  (PORT_E_BASE+0xC))
#define     GPIO_E_IDRLOW   ((volatile unsigned char *) (PORT_E_BASE+0x10))
#define     GPIO_E_ODRLOW   ((volatile unsigned char *) (PORT_E_BASE+0x14))

/* Port D */
#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))

struct {
    charsec_div100;
    charsec_div10;
    charsec;
    charsec_mul10;
}stoptime;
int overflow;
int running;

void reset_watch( void )
{
    stoptime.sec_div100 = 0;
    stoptime.sec_div10 = 0;
    stoptime.sec = 0;
    stoptime.sec_mul10 = 0;
    overflow = 0;
}

void bump_watch( void )
{
    if( overflow ) return;
    if( ! running ) return;
    stoptime.sec_div100++;
    if( stoptime.sec_div100 == 10 )
    {
        stoptime.sec_div10++;
        stoptime.sec_div100 = 0;
    }
    if( stoptime.sec_div10 == 10 )
    {
        stoptime.sec++;
        stoptime.sec_div10 = 0;
    }
    if( stoptime.sec == 10 )
    {
        stoptime.sec_mul10++;
        stoptime.sec = 0;
    }
    if( stoptime.sec_mul10 == 7 )
    {
        overflow++;
    }
}
```

```
void irq_handler ( void )
{
    unsigned char c = *GPIO_E_IDRLOW & 0xF;

    if(   *EXTI_PR & EXTI3_IRQ_BPOS )
    {
        if( c & 4 )
        {
            bump_watch();
            *GPIO_E_ODRLOW = 0x40;
        } else if ( c & 2 )
        {
            *GPIO_E_ODRLOW = 0x20;
            reset_watch();
        }else if ( c & 1 )
        {
            *GPIO_E_ODRLOW = 0x10;
            if( running ) running = 0;
            else running = 1;
        }
        *GPIO_E_ODRLOW = ~0x70;
    }
    *EXTI_PR |= EXTI3_IRQ_BPOS;
}



void init_app ( void )
{
    *GPIO_D_MODER = 0x55555555;
    *GPIO_D_OTYPER= 0x00000000;
    *GPIO_D_ODRLOW = 0;
    *GPIO_D_ODRHIGH = 0;

    *GPIO_E_MODER = 0x00005500;
    *GPIO_E_PUPDR = 0x00000000;
    *GPIO_E_OTYPER= 0x00000000;
    *GPIO_E_ODRLOW = 0;

    running = 0;
    *GPIO_E_ODRLOW = 0x70;
    *GPIO_E_ODRLOW = ~0x70;

    *SYSCFG_EXTICR1 |= 0x4000;

    *EXTI_IMR |= EXTI3_IRQ_BPOS;

    *EXTI_RTSR |= EXTI3_IRQ_BPOS;
    *EXTI_FTSR &= ~EXTI3_IRQ_BPOS;

    *((void (**)(void) ) EXTI3_IRQVEC ) = irq_handler;
    *((unsigned int *) NVIC_ISER0) |= NVIC_EXTI3_IRQ_BPOS;
 }

void main(void)
{
    init_app();
    reset_watch();
    while(1)
    {
            *GPIO_D_ODRLOW = (stoptime.sec_div10<<4)|stoptime.sec_div100;
            *GPIO_D_ODRHIGH = (stoptime.sec_mul10<<4)|stoptime.sec;
    }
}
```

Uppgift 4:

```
// a)

typedef struct
{
    unsigned short  CTRL; // Bitfield also okay. char and reserved also fine.
    unsigned char   STATE, ALT;
    unsigned short  DEPRECATED;
    unsigned char   VEL, TILT;
    unsigned short  IRQ; // Bitfield also okay. char and reserved also fine.
} ML_MODULE;

ML_MODULE * ml = ((ML_MODULE *) 0xE000E0F0);

// b)
void init() {
    // Initialize ML registers
    ml->STATE = 0; // This MUST be done BEFORE interrupts are enabled to avoid firing
                   // rockets while docked.
    // Set up interrupt handlers
    unsigned int VECTOR_TABLE_BASE = *((unsigned int *)(0xE000ED00 + 0x8));
    *((void (**)(void)) (VECTOR_TABLE_BASE + 0x188)) = ML_ATMO_handler;
    *((void (**)(void)) (VECTOR_TABLE_BASE + 0x18B)) = ML_PANIC_handler;
    // Set interrupt priorities
    const unsigned int NVIC_IPR15 = 0xE000E400 + 15  * 4; // Both interrupts 61 and 62 are in IPR15
    const int ML_ATMO_PRIO = 32; // It only matters that PANIC prio is less than ATMO
    const int ML_PANIC_PRIO = 0;
    *((unsigned int *) NVIC_IPR15) &= 0xFF0000FF;
    *((unsigned int *) NVIC_IPR15) |= (ML_ATMO_PRIO << 8);
    *((unsigned int *) NVIC_IPR15) |= (ML_PANIC_PRIO << 16);
    // Enable both interrupts
    const unsigned int NVIC_ISER1 = 0xE000E104;
    *((unsigned int*)NVIC_ISER1) = *((unsigned int*)NVIC_ISER1) | (1 << 29); // Enable ML_ATMO
    *((unsigned int*)NVIC_ISER1) = *((unsigned int*)NVIC_ISER1) | (1 << 30); // Enable ML_PANIC
}

// c)
void ML_ATMO_handler()
{
    // We have enetered the atmosphere. Engage landing mode!
    ml->STATE = 1;
}

void ML_PANIC_handler()
{
    // Tilt or speed is too high. Time to panic.
    ml->CTRL |= 0x4;
}

void main()
{
    // Initialize and drop lander
    init();

    // Ensure STATE is "Landing" (not necessary)
    while(ml->STATE != 0) {};

    // Run until touchdown
    while(ml->STATE != 2)
    {
        // Engage stabilization rockets if tilting
        if(ml->TILT > 0) {
            ml->CTRL |= 0x1;
        }
        // Engage break rockets if going to fast and in Landing state
        if(ml->STATE == 1) {
            if(ml->VEL > ml->ALT) ml->CTRL |= 0x2;
        }
        // Set touchdown state when ALT = 0
        if(ml->ALT == 0) {
            ml->STATE = 2;
        }
    }
}
```