Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Hemtentamen med delvisa lösningsförslag

EDA488 Maskinorienterad programmering Z

Onsdag 2 juni 2021, kl. 8.30 - 12.30 (14.30)

Examinator
Roger Johansson

Kontaktpersoner under tentamen:
Roger Johansson, 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 kurshemsidan.

Allmänt
På Canvas-sidan för tentamen finns en FAQ där vanliga frågor av generell natur publiceras. Se först om du hittar svar på din fråga här, skicka annars din fråga (i Canvas-rummet) till kontaktpersonen.

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:

$30p \leq$ betyg $3 < 40p \leq$ betyg $4 < 50p \leq$ betyg 5.

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

Emil får i uppgift att översätta följande program, givet i C, till ARMv6 assemblerspråk.

```
int differential( char d[],
       int length, int *diff, int *xpos )
{
    *diff = *xpos = 0;
    for( int i = 0; i < length-1; i++)
    {
        if( ( d[i+1] - d[i] ) > *diff )
        {
            *diff = d[i];
            *xpos = i;
        }
    }
    return 1;
}
```

```
char   teststring[32];
int result, diff, xpos;

void main(void)
{
    int length = 15;
    result = differential( teststring,
        length, &diff, &xpos );
}
```

Resultatet är följande assemblerprogram :

```
@int   differential( char d[], int length,
@      int *diff, int *xpos )
@{
1   PUSH  {R5,R6}
@   *diff = *xpos = 0;
2   MOV   R5,#0
3   STR   R5,[R3]
4   STR   R5,[R2]
@   for( int i = 0; i < length-1; i++ )
5   MOV   R4,#0
6   B     L1
@   {
@       if( ( d[i+1] - d[i] ) > *diff )
L3:
7   MOV   R5,R4
8   ADD   R5,#1
9   ADD   R5,R5,R0
10  LDRB  R5,[R5]
11  ADD   R6,R4,R0
12  LDRB  R6,[R6]
13  SUB   R5,R5,R6
14  LDRB  R6,[R2]
15  CMP   R5,R6
16  BLE   L2
@       {
@           *diff = d[i];
17  ADD   R6,R0,R4
18  LDRB  R5,[R6]
19  STR   R5,[R2]
@           *xpos = i;
20  MOV   R3,[R4]
@   for( int i = 0; i < length-1; i++)
L2:
21  ADD   R4,R4,#1
```

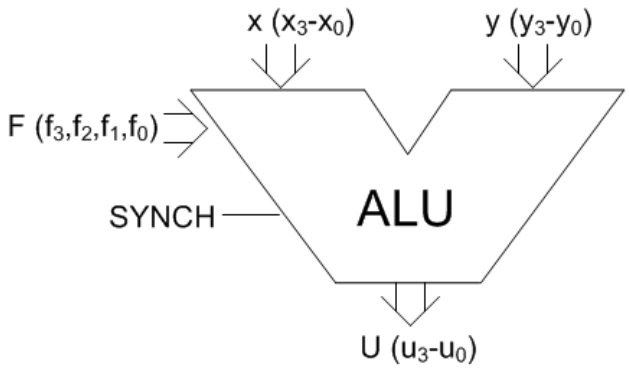
```
@   for( int i = 0; i < length-1; i++)
L1:
22  MOV   R5,R1
23  SUB   R5,#1
24  CMP   R5,R4
25  BLT   L3
@       }
@   }
26  MOV   R0,#1
27  POP   {R5,R6}
28  BX    LR
@}

@char  teststring[32];
@int result, diff, xpos;

29  .ALIGN
30  teststring:  .SPACE 32
31  result:      .SPACE 4
32  diff:        .SPACE 4
33  xpos:        .SPACE 4

@void main(void)
@{
34  PUSH  {R4,LR}
@   int length = 15;
35  MOV   R4,#15
@   result = differential( teststring,
@          length, &diff, &xpos );
36  LDR   R3,=xpos
37  LDR   R2=diff
38  LDR   R1,=length
39  LDR   R0,=teststring
40  B     differential
41  LDR   R1,=result
42  STR   R0,[R1]
43  POP   {R4,PC}
@}
```

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

Hitta raderna, som här har numrerats så att du säkert kan identifiera raderna, för varje felaktig rad, *kommentera* (beskriv) felet och visa hur Emil borde gjort (*rätta felet*), du får ange maximalt 5 rader.

Uppgift 2

En applikation för simulering av en 4-bitars Aritmetik-/Logik- enhet (ALU) med 16 olika funktioner ska konstrueras med hjälp av en laborationsdator *MD407*. ALU:n ska ha ett synkront beteende. Dvs. utsignalerna ändras baserat på insignalerna endast då en klocksignal (SYNCH) ges. Följande gäller för ALU:ns funktioner:

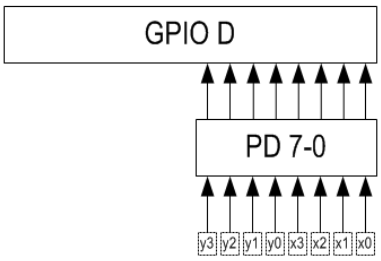


funktion				*operation*	*utsignaler*			
f_3	f_2	f_1	f_0	RTN	u_3	u_2	u_1	u_0
0	0	0	0	U= 0	0	0	0	0
0	0	0	1	U= F_{16}	1	1	1	1
0	0	1	0	U= x	x_3	x_2	x_1	x_0
0	0	1	1	U= y	y_3	y_2	y_1	y_0
0	1	0	0	U=x+y	summa			
0	1	0	1	U=x-y	skillnad			
0	1	1	0	U=x*y	produkt			
0	1	1	1	U=x/y	heltalsdivision			
1	0	0	0	U=x%y	restdivision			
1	0	0	1	U= x|y	bitvis OR			
1	0	1	0	U= x&y	bitvis AND			
1	0	1	1	U= x^y	bitvis EOR			
1	1	0	0	U= y<<1	y_2	y_1	y_0	1
1	1	0	1	U= x<<1	x_2	x_1	x_0	0
1	1	1	0	U= 1>>y	1	y_3	y_2	y_1
1	1	1	1	U=1>>x	0	x_3	x_2	x_1

GPIO-portarna D och E används för ALU:ns anslutningar enligt följande:

y3-y0 ansluts till PD7-PD4

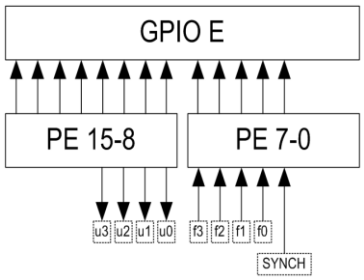
x3-x0 ansluts till PD3-PD0



u3-u0 ansluts till PE11-PE8

f3-f0 ansluts till PE7-PE4

SYNCH ansluts till PE3



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

a) Visa en initieringsfunktion

`void init(void):`

1) Initierar Portarna D och E för den önskade funktionen.
2) Initierar systemet för att hantera avbrott med funktionen `alu_eval` (se nedan).

b) Visa en avbrottsfunktion

`void alu_eval(void):` som

1) Läser insignalerna (x,y,f) från portpinnar
2) Evaluerar utsignalerna (u) enligt funktionstabellen
3) Skriver utsignalerna till portpinnar.

Anm: Vid evaluering av utsignalerna för heltalsdivision: om y är 0 sätts u till 0xF. Vid evaluering av utsignalerna för restdivision: om y är 0 sätts u till 0.

Implementera koden i programspråket C.

Uppgift 3

Ett system för att generera en enkel PWM-signal ska konstrueras. Systemet byggs kring funktionen:

```
void pwm( int dutycycle, int period, int repetitions);
```

En PWM-puls varar i en `period`, nivån är hög under `dutycycle` och låg under resten av periodtiden, Totalt skapas `repetitions` pulser vid ett anrop av funktionen. Enheten för `dutycycle` och `period` är millisekunder. TIMER_6 ska användas för att skapa en blockerande fördröjning (1 ms).

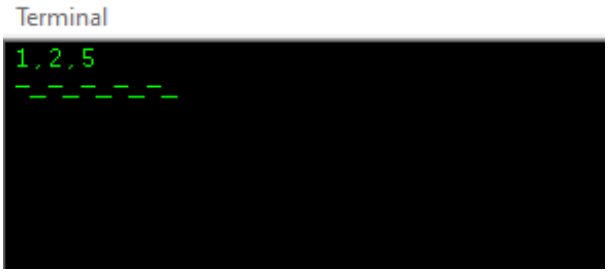
För att kontrollera funktionen används en serieterminal, du måste skriva in- och utmatningsfunktioner (`inchar`/`outchar`) för terminalen men du får förutsätta att USART-kretsen är korrekt initierad.

Testprogrammet använder terminalen för inmatning av en texsträng, tre heltal separerade med kommatecken (*dutycycle*,*period*,*repetitions*). Inmatningen avslutas med Enter-tangenten (`'\n'`).

Funktionen `pwm` illustrerar PWM-signalen genom att skriva ASCII-tecknen '-' för hög nivå och '_' för låg nivå till terminalen.

Följande illustrationer ger exempel på resultat av testprogrammet:

Sekvensen 1,2,5 matas in
dvs. *dutycycle* är 1 ms, periodtiden är 2 ms och 5 PWM-pulser ska genereras. Utskriften på terminalen blir då:



Sekvensen 4,5,8 matas in
dvs. *dutycycle* är 4 ms, periodtiden är 5 ms och 8 PWM-pulser ska genereras. Utskriften på terminalen blir då:

Ledningar:

Konstruera en inmatningsrutin "`get_line`" som läser in en textsträng från terminalen.
Du får förutsätta att textsträngen alltid ges på formatet X,Y,Z där:

X är `'dutycycle'`
Y är `'period'`
Z är `'repetitions'`

Du får dessutom förutsätta att X,Y,Z alla är ASCII-tecknen för heltalen, 0-9.

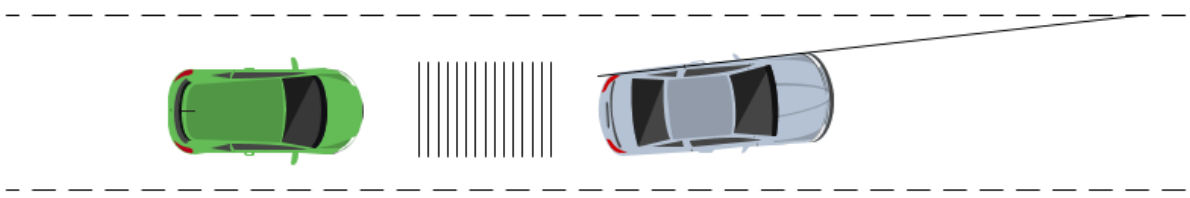
Följande funktioner ska alltså konstrueras:

- En blockerande fördröjningsfunktion som använder TIMER_6.
- En funktion `pwm` som skriver PWM-signalen till terminalen
- En funktion `get_line` som läser en textsträng från terminalen.
- Ett huvudprogram `main` som kan användas för att testa funktionerna.

Det är *inte* tillåtet att använda funktioner ur Standard-C biblioteket i denna uppgift.

Uppgift 4

Ett assistanssystem i ett fordon har kameror och radar för att detektera såväl avvikelse från en vägmarkering som är parallell med färdriktningen som avstånd till framförvarande fordon.



Avståndet till framförvarande fordon (eller annat hinder) mäts i 16 steg och avvikelse från vägmarkering mäts i vinkelangivelser 0-7°. En speciell signal varnar om vinkeln är större än 7°.

Översikt av assistanssystemets register

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																																	CTRL
4																																	STATUS

Beskrivningar av register:

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0						ABF	ABW	ABO				ALW	LR	ASWA			CTRL

ASWA, Apply Steering Wheel Angle, skrivbara bitar. Styrning korrigeras automatiskt (1-7grader) då detta fält är skilt från 0.

LR: Left/Right, skrivbar bit. Anger riktning för automatisk korrigering av styrning.

ALW: Apply Lane Warning: skrivbar bit. Då denna bit sätts till 1 ger assistanssystemet en varningssignal från färdriktningssystemet om färdriktningen avviker från vägmarkeringen .

ABW: Apply Brake Warning: skrivbar bit. Då denna bit sätts till 1 ger assistanssystemet en varningssignal från bromssystemet.

ABO: Auto Break Operating, läs- och skrivbar bit. Anger att assistanssystements bromsfunktion är aktiverad.

ABF: Apply Brake Force, skrivbar bit, lägg an bromskraft.

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
4	BRKDA					LDA		LDW	LRS					LDD			STATUS

LDD, Lane Departure Deviation, enbart läsbara bitar. Anger vinkeln mellan färdriktning och vägmarkering (0-7°).

LRS: Left/Right Sense, enbart läsbar bit, anger om vinkeln avser vänster eller höger vägmarkering.

LDW, Lane Departure Warning, enbart läsbar bit. Är 1 om vinkeln mellan färdriktning och vägmarkering är större än 7°.

LDA, Lane Deviation Active, enbart läsbar, biten är 1 om assistanssystemet har en giltig vägmarkering

BRKDA, enbart läsbara bitar. Anger avstånd till framförvarande hinder i 16 steg, där 0 anger oändligt" och 0xF anger "omedelbart nära".

Bitar som inte används ska hållas i sina respektive initialvärden.

Registermodulen har placerats på adress `0xF0000000` i en *MD407* mikrodator.

a) Visa en typdeklaration med **bitfält** som representerar registermodulen.
b) Visa en funktion `main`, enligt följande specifikation:
   Om det finns en giltig vägmarkering ska följande utföras:
   Om vinkeln mellan färdriktning och vägmarkering är större än 7 grader ska varningssignal aktiveras.
   Om vinkeln är större än 0 grader men mindre än 8 grader ska färdriktningen korrigeras så att vinkeln blir 0 grader.

   Om systemets automatiska broms är aktiverad ska följande utföras:
   Om avstånd till framförvarande hinder är mindre än halva mätintervallet ska varningssignal ges.
   Om avståndet dessutom är "omedelbart nära" ska full bromskraft läggas an.

Lösningsförslag:

Uppgift 1:

```
@int   differential( char d[], int length,
@      int *diff, int *xpos )
@{
    PUSH   {R5,R6}
@   *diff = *xpos = 0;
    MOV    R5,#0
    STR    R5,[R3]
    STR    R5,[R2]
@   for( int i = 0; i < length-1; i++ )
    MOV    R4,#0
    B      L1
@   {
@      if( ( d[i+1] - d[i] ) > *diff )
L3:
    MOV    R5,R4          @R5<-i
    ADD    R5,#1          @R5<-i+1
    ADD    R5,R5,R0       @R5<-&d+i+1
    LDRB   R5,[R5]        @R5<-d[i+1]
    ADD    R6,R4,R0       @R6<-&d+i
    LDRB   R6,[R6]        @R6<-d[i]
    SUB    R5,R5,R6       @R5<- d[i+1]-d[i]
14  LDR    R6,[R2]        @R6<-*xpos
    CMP    R5,R6
    BLE    L2
@      {
@         *diff = d[i];
    ADD    R6,R0,R4
    LDRB   R5,[R6]
    STR    R5,[R2]
@         *xpos = i;
20  STR    R4,[R3]
@   for( int i = 0; i < length-1; i++)
L2:
    ADD    R4,R4,#1
```

```
@   for( int i = 0; i < length-1; i++)
L1:
    MOV    R5,R1
    SUB    R5,#1
24  CMP    R4,R5
    BLT    L3
@      }
@   }
    MOV    R0,#1
    POP    {R5,R6}
    BX     LR
@}

@char  teststring[32];
@int result, diff, xpos;

    .ALIGN
teststring:  .SPACE 32
result:      .SPACE 4
diff:        .SPACE 4
xpos:        .SPACE 4

@void main(void)
@{
    PUSH   {R4,LR}
@   int length = 15;
    MOV    R4,#15
@   result = differential( teststring,
@         length, &diff, &xpos );
    LDR    R3,=xpos
    LDR    R2=diff
38  MOV    R1,R0
    LDR    R0,=teststring
40  BL     differential
    LDR    R1,=result
    STR    R0,[R1]
    POP    {R4,PC}
@}
```

14, Fel storlek “BYTE”
20, Fel indirektion “MOV”
24, Fel jämförelseordning
38, Fel syntax
40, Fel instruktion

Uppgift 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_PUPDR    ((volatile unsigned int *)   (PORT_D_BASE+0xC))
#define     GPIO_D_IDRLOW   ((volatile unsigned char *)  (PORT_D_BASE+0x10))

#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_IDRLOW   ((volatile unsigned char *) (GPIO_E_BASE+0x10))
#define     GPIO_E_ODRHIGH  ((volatile unsigned char *) (GPIO_E_BASE+0x14+1))

#define     SYSCFG_EXTICR1  ((volatile unsigned int *)  0x40013808)
#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     EXTI3_IRQVEC    ((volatile unsigned int *)  0x2001C064)
#define     NVIC_ISER0      ((volatile unsigned int *)  0xE000E100 )

#define NVIC_EXTI3_IRQ_BPOS  (1<<9)
#define EXTI3_IRQ_BPOS  (1<<3)


void init( void )
{
    *GPIO_E_MODER = 0x00550000;
    *GPIO_E_PUPDR = 0x0000AA80;
    *GPIO_E_OTYPER= 0x00000000;
    *SYSCFG_EXTICR1 &= ~0x4000;
    *SYSCFG_EXTICR1 |= 0x4000;
    *EXTI_IMR |= EXTI3_IRQ_BPOS;
    *EXTI_RTSR |= EXTI3_IRQ_BPOS;
    *EXTI_FTSR &= ~EXTI3_IRQ_BPOS;
    *((void (**)(void) ) EXTI3_IRQVEC ) = alu_eval;
    *NVIC_ISER0 |= NVIC_EXTI3_IRQ_BPOS;
}

void alu_eval ( void )
{
    char x,y,f,u;
    x = *GPIO_D_IDRLOW & 0xF;
    y = *GPIO_D_IDRLOW >>4;
    f = *GPIO_E_IDRLOW >>4;
    switch( f )
    {
       case 0: u = 0; break;
       case 1: u = 0xF; break;
       case 2: u = x; break;
       case 3: u = y; break;
       case 4: u = x+y; break;
       case 5: u = x-y; break;
       case 6: u = (x*y) & 0xF; break;
       case 7: if( y == 0) u = 0xF; u = x/y; break;
       case 8: if( y == 0) u = 0; else u = x%y; break;
       case 9: u = x|y; break;
       case 10: u = x&y; break;
       case 11: u = x^y; break;
       case 12: u = ((y<<1)|1) & 0xF; break;
       case 13: u = (x<<1) & 0xF; break;
       case 14: u = (y>>1)|0x8; break;
       case 15: u = x>>1; break;
   }
   *GPIO_E_ODRHIGH = u;
   *((unsigned int *) EXTI_PR) |= EXTI3_IRQ_BPOS;
}
```

Uppgift 3:

```
typedef struct tag_usart
{
    volatile unsigned short sr;
    volatile unsigned short Unused0;
    volatile unsigned short dr;
    volatile unsigned short Unused1;
    volatile unsigned short brr;
    volatile unsigned short Unused2;
    volatile unsigned short cr1;
    volatile unsigned short Unused3;
    volatile unsigned short cr2;
    volatile unsigned short Unused4;
    volatile unsigned short cr3;
    volatile unsigned short Unused5;
    volatile unsigned short gtpr;
} USART;
#define USART1  ((USART *) 0x40011000)

/* Timer 6 */
#define TIM6_CR1  ((volatile unsigned short *)   0x40001000)
#define TIM6_DIER ((volatile unsigned short *)   0x4000100C)
#define TIM6_SR   ((volatile unsigned short *)   0x40001010)
#define TIM6_PSC  ((volatile unsigned short *)   0x40001028)
#define TIM6_ARR  ((volatile unsigned short *)   0x4000102C)
#define UIF       (1<<0)
#define UIE       (1<<0)
#define CEN       (1<<0)


void _outchar( char c )
{
    while (( USART1->sr & (1<<7))==0);
    USART1->dr = (unsigned short) c;
}

char _tstchar( void )
{
    if( (USART1->sr & (1<<5) )==0)
        return 0;
    return (char) USART1->dr;
}


char _inchar( void )
{
    char c;
    while ( (c=_tstchar() )==0);
    return c;
}

int get_line(char * buffer)
{
    unsigned char  c ;
    int cursor_pos = 0;
    do
    {
        c = _inchar();
        _outchar( c );
        switch( c )
        {
            case '\n': break;
            default:
                buffer[cursor_pos] = c;
                cursor_pos ++;
        }
    }while( c != '\n');
    buffer[cursor_pos] = 0;
    return cursor_pos;
}
```

```
void delay_1ms( void )
{
    *TIM6_CR1 &= ~CEN;
    *TIM6_PSC= 83;
    *TIM6_ARR = 999;
    *TIM6_DIER |= UIE;
    *TIM6_CR1 |= CEN;
    while(( *TIM6_SR & UIF ) == 0 );
    *TIM6_CR1 &= ~CEN;
}

void pwm( int dutycycle, int period, int
repetitions)
{
    int noduty;
    while( repetitions > 0)
    {
        noduty = period - dutycycle;
        for( int i = 0; i < dutycycle; i++ )
        {
            _outchar('-' ) ;
            delay_1ms();
        }
        for( int i = 0; i < noduty; i++ )
        {
            _outchar('_') ;
            delay_1ms();
        }
        repetitions--;
    }
}

void main(void)
{
    char buffer[32];
    int dutycycle, period, repetitions;

    unsigned int w,p;
    while( 1 )
    {
        if( get_line( buffer ) == 5 )
        {
            dutycycle = buffer[0]-'0';
            period = buffer[2]-'0';
            repetitions = buffer[4]-'0';
            pwm( dutycycle, period, repetitions );
        }
   }
}
```

Uppgift 4:

```
typedef struct lanebrake_tag
{
    volatile unsigned int ASWA:4;
    volatile unsigned int LR:1;
    volatile unsigned int ALW:1;
    unsigned int :3;
    volatile unsigned int ABO:1;
    volatile unsigned int ABW:1;
    volatile unsigned int ABF:1;
    unsigned int :0;
    volatile unsigned int LDD:3;
    unsigned int :4;
    volatile unsigned int LRS:1;
    volatile unsigned int LDW:1;
    unsigned int :1;
    volatile unsigned int LDA:1;
    unsigned int :1;
    volatile unsigned int BRKDA:4;
} LANEBRAKE_REG;

#define REGBASE (( LANEBRAKE_REG *) 0xF0000000)

int main( void)
{
    if (REGBASE->LDA)
    {
        if( REGBASE->LDW )
            REGBASE->ALW = 1;
        if( REGBASE->LDD > 0 )
        {
            if( REGBASE->LRS)
                REGBASE->LR = 0;
            else
                REGBASE->LR = 1;
        }
    }

    if( REGBASE->ABO)
    {
        if( REGBASE->BRKDA > 7 )
            REGBASE->ABW = 1;
        if( REGBASE->BRKDA == 0xF )
            REGBASE->ABF = 1;
    }
}
```