Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

DAT017 Maskinorienterad programmering IT

DIT151 Maskinorienterad programmering GU

EDA481 Programmering av inbyggda system D

EDA486 Programmering av inbyggda system Z

DAT016 Programmering av inbyggda system IT

DIT152 Programmering av inbyggda system GU

Måndag 10 april 2017, kl. 14.00 - 18.00

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:
Roger Johansson

Tillåtna hjälpmedel

Utgåvor som distribuerats inom ramen för kursen, häftena:

- *Instruktionslista för CPU12*
- *Quick Guide MOP*

Understrykningar (”överstrykningar”) får vara införda i dessa häften.
Instruktionslista för CPU12 får innehålla rättelser.
Quick Guide MOP får inte innehålla några egna anteckningar. Rättelser till *Quick Guide* finns som bilaga 3 i denna tes.

Tabellverk eller miniräknare får ej användas.

Lösningar
anslås senast dagen efter tentamen via kursens hemsida.

Granskning
Tid och plats anges på kursens hemsida.

Allmänt
Tentamen är anpassad för både det äldre laborations-systemet (*MC12*) och det nyare (*MD407*). Varje uppgift föregås av någon av bokstäverna A, B eller C.

A: uppgift avser laborationssystem *MC12*.

B: uppgift avser laborationssystem *MD407*.

C: uppgift kan besvaras av alla.

Du väljer själv om du vill besvara A *eller* B-märkta uppgifter. Du ska inte besvara *både* A *och* B-märkta uppgifter. Om din lösning innehåller både A och B-märkta uppgifter kommer de A-märkta uppgifterna att bedömas.

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.

Tentamenspoäng ger slutbetyg enligt:
(EDA/DAT/LEU):
$20p \leq$ betyg 3 $< 30p \leq$ betyg 4 $< 40p \leq$ betyg 5
respektive (DIT):
$20p \leq$ betyg G $< 35p \leq$ VG

Uppgift A-1 (12p)

a) Redogör för vad som händer vid RESET och varför detta sker. (2p)

b) Förklara kortfattat vad som händer vid ett IRQ avbrott om I-flaggan i CC är nollställd. (2p)

c) Vid IRQ-avbrott sätts I-flaggan automatiskt till 1. Varför sker detta? (1p)

d) Visa med en instruktionssekvens hur man i en IRQ-avbrottsrutin kan förhindra att processorn utför nya avbrott efter återhopp till det avbrutna programmet. (3p)

e) Antag att avbrottsfunktionen `irq`, är implementerad i assemblerspråk. Visa hur en avbrottsvektor på adress 0x3e00 initieras med en pekare till avbrottsfunktionen, såväl i assemblerspråk som i C. (4p)

Uppgift A-2 (6p) *Assemblerprogrammering*

Följande funktion finns given i "C". Implementera en motsvarande subrutin i assemblerspråk för HC12. Du ska inte förutsätta några speciella kompilatorkonventioner i denna uppgift. Parametern 'cp' skickas i register X med anropet, returvärdet från subrutinen ska finnas i register Y efter att subrutinen utförts.

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

Uppgift A-3 (10p) *Kodningskonventioner (C/assemblerspråk)*

I denna uppgift ska du förutsätta samma konventioner som i XCC12, (se bilaga 1).

Följande C-deklarationer har gjorts på "toppnivå" (global synlighet):

```
unsigned char     p;
unsigned int      a;
unsigned int      g_array[5];
int               *k;
```

a) Visa hur dessa deklarationer översätts till assemblerdirektiv för HCS12. (4p)

b) Med variabeldeklarationerna enligt a), visa hur tilldelningssatsen `a = g_array[3];` kan kodas i HCS12 assemblerspråk. (2p)

c) Implementera en assembler subrutin som kan anropas från ett C-program.

```
unsigned short int getY( void );
```

Funktionen ska returnera det värde som register Y innehåller vid den punkt i programflödet där anropet görs. (2p)

Uppgift B-1 (14p)

En periferienhet med ett 8 bitars gränssnitt ska anslutas till ett MD407-system.

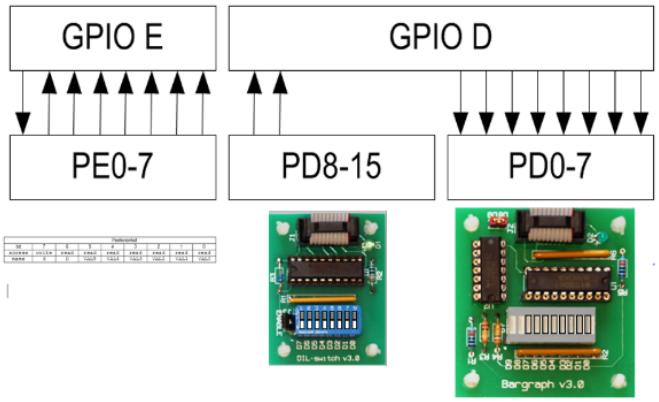
Periferienhet								
bit	7	6	5	4	3	2	1	0
access	read/ write	read	read	read	read	read	read	read
namn	S	D	VAL5	VAL4	VAL3	VAL2	VAL1	VAL0

Från RESET är bit S=0, medan bit D och VAL-bitarna är odefinierade.

Periferienheten styrs och fungerar på följande sätt:

- Programvaran sätter bit S till 1 för att starta operationen. Kretsen behöver nu 100 ms innan statusbiten D är giltig. Då resultatet är färdigt sätts bit D till 1 av periferienheten. Nu är värdet VAL giltigt och kan läsas av från dessa bitar i gränssnittet. Ett giltigt värde är alltid skilt från 0, observera dock att då D är 0 kan VAL innehålla ett värde skilt från 0 trots att detta är ogiltigt.
- Programvaran ska nu läsa av värdet VAL.
- Slutligen återställs periferienheten genom att bit S sätts till 0 av programvaran. Periferikretsen nollställer då bit D vilket indikerar att värdet VAL inte längre är giltigt.
- Periferikretsen är nu klar för nästa operation.

Konstruera en applikation som kontinuerligt läser ett giltigt värde från periferikretsen och skriver detta till en diodramp. Dessutom ska 2 bitar läsas från en strömställare och skrivas till diodrampen. Port E (0-7) ska användas för periferikretsen, medan port D (bit 15 och bit 14) ska användas för strömställare och port D (0-7) till diodrampen. Eftersom de två bitarna från strömställaren ska uppdateras kontinuerligt utan onödig fördröjning kan inte fördröjningsfunktionen utformas som en blockerande funktion.



Dela upp applikationen i följande dellösningar:

a) Samtliga register (portar) som avänds i denna uppgift ska deklareras med typkonverterande makron så som vi rekommenderat i kursen. Visa dessa makron. Du kan sedan använda dem i resterande deluppgifter. (2p)

b) En initieringsfunktion `init_app`, där du visar hur portarna D och E ska initieras, alla utgångar ska vara *push-pull*, alla ingångar ska vara *pull-down*. IO-pinnar som inte används kan anses vara odefinierade. (2p)

c) Använd SYSTICK för att implementera en icke blockerande fördröjning, dvs. det ska finnas en funktion `int is_timeout(void)` som returnerar *true* om 100 ms passerat annars *false*. Systemets klockfrekvens är 168 MHz, dvs. 1 mikrosekund är 168 räknade klockpulser. Eftersom räknaren inte kan programmeras för så långa intervall som 100 ms ska du utnyttja avbrottsperioden 10 ms. (5p)

d) Skriv huvudprogrammet (5p), följande algoritm kan användas:

```
algoritm:  main
      variabler: c, peripheral_input;
      init_app;
      peripheral_input = 0;

repetera:
            Om bit S==0
                  S=1;
                  starta 100 ms fördröjning;
            c = dipswitch, bitar 7 och 6;
            Om bit D är giltig, dvs. 100 ms passerat, och bit D==1
                  peripheral_input = VALbitar;
                  S=0;
            skriv ut:  c | peripheral_input ;
```

Uppgift B-2 (8p)

I denna uppgift ska du förutsätta samma konventioner som i GCC för ARM. Funktionen `unsigned int countOnes(unsigned int)` är definierad sedan tidigare.

Följande deklarationer (data och funktioner) är givna i "C".

```
unsigned int func( unsigned int a, unsigned int b)
{
   if( countOnes(a)  >= b  )
      return ~b;
   else
      return b;
}

unsigned int   c,d,e;

void  callfunc(void )
{
   e = func( c, d );
}
```

Implementera motsvarande variabeldeklarationer i assemblerspråk för ARM-v6. Implementera också de båda funktionerna i assemblerspråk.

Uppgift B-3 (6p)

Förbered en enkel applikation som använder PD10 hos MD407 som avbrottsingång. Dvs, skriv, i C, en sekvens som:

- Kopplar PD10 till EXTI10
- Konfigurerar EXTI10 för att generera avbrott på *positiv* flank
- Konfigurerar NVIC.

Ange också offseten i vektortabellen för den vektor som ska initieras för avbrottet.

Du kan förutsätta att alla moduler startats och behöver inte ta hänsyn till klockor (RCC). Observera dock att andra eventuella EXTI- eller NVIC- konfigurationer *inte* får ändras av din programsekvens. För full poäng ska du visa hur preprocessordirektiv och ev. typdeklarationer används för att skapa begriplig programkod. Typkonverteringar ska göras på sådant sätt som rekommenderats i kursen.

Uppgift C-4 (8p) *C-programmering*

Konstruera en C-funktion som undersöker en parameter med avseende på antalet 0-ställda bitar. Funktionen deklareras:

```
int  bitcheck( unsigned int *p, long * num );
```

- `p` är en pekare till det värde som ska undersökas
- `num` är en pekare till en plats dit antalet 0-ställda bitar hos parametern `p`, ska skrivas

Returvärdet för funktionen ska vara skilt från 0 om antalet nollor hos parametern är jämt delbart med 8, annars ska returvärdet vara 0.

Uppgift C-5 (14p) *Maskinnära programmering i C*

Denna uppgift handlar om att konstruera drivrutiner för en tänkt kommunikationsenhet. Enheten kan ta emot och sända data från resp. till en annan dator. Enheten har därför två separata kanaler: en mottagningskanal och en sändningskanal. Deluppgift a behandlar endast mottagningskanalen, medan deluppgift b behandlar sändingskanalen samt koppling mellan kanalerna. Skriv programmet i C. För full poäng ska du strukturera din lösning, visa hur olika delar lämpligen placeras i olika filer och använda lämpliga definitioner av typer och makron.

Drivrutinerna för enheten använder sig av två buffertar: en inbuffert i vilken inlästa data läggs och en utbuffert som utgående data hämtas ifrån. Det är meningen att ett program som vill använda sig av kommunikationsenheten ska göra det via dessa buffertar. Vill programmet ta emot data via kommunikationsenheten ska det alltså hämta data från inbufferten och vill det sända data skall det lägga dessa data i utbufferten. Det finns en *färdigskriven* modul för buffertar som du ska använda dig av. (Men du behöver inte skriva dess `.c` fil.) Den har följande `.h` fil:

```
/* Filen buffer.h */
typedef struct bufferstruct buffer;  /* Full definition finns i filen buffer.c */
buffer *create_buffer(int max);   /* Ger en buffert med plats för max element */
int put(buffer *b, char c);       /* Försöker lägga in c sist i *b.
                                     Ger true om det gick, false annars */
int get(buffer *b, char *pc);     /* Försöker hämta första tecknet i *b till *pc.
                                     Ger true om det gick, false annars */
```

Följande specifikationer gäller för kommunikationenheten:

Det finns fyra åttabitars register: `SEND_DATA`, `SEND_CTRL`, `RECEIVE_DATA` och `RECEIVE _CTRL`.

Dessa ligger på de hexadecimala adresserna 400, 401, 402 resp. 403. I dataregistren finns de data som ska tas emot eller sändas. I kontrollregistren används tre bitar:

- Bit 0, `RDY` sätts automatiskt till 1 när en överföring är klar. Den ska nollställas vid en ny överföring.
- Bit 1, `GO` sätter man till 1 för att starta en överföring. Den nollställs automatiskt.
- Bit 2, `IE` aktiverar enhetens avbrottsmekanism genom att sättas till 1. Då genereras ett avbrott när en överföring är klar.

Deluppgift a (8p)

Konstruera den del av drivrutinerna som har att göra med mottagning. Det ska bl.a finnas en fil `driver.h` med en tillhörande fil `driver.c`. Lägg dessutom gärna alla specifikationer för hårdvaran i en egen fil `channel.h`. Filen `driver.h` ska utgöra gränssnittet mot övriga program i datorn. I denna fil ska två funktioner deklareras: `init_input` som initierar mottagningsdelen av enheten och `get_input_buffer` som ger en pekare till den inbuffert drivrutinen skapat och använder sig av.

Dessa två funktioner ska förstås definieras i filen `driver.c`. I initieringsfunktionen ska bl.a. bufferten skapas. Låt den ha 100 platser. Initieringsfunktionen måste vara utformad så att det inte kan skapas mer än en inbuffert, även om funktionen skulle bli anropad mer än en gång.

I filen `driver.c` skall det dessutom finnas en funktion `notify_input`. Denna anropas varje gång ett avbrott sker. Detta görs från en avbrottsfunktion som redan är definierad, du ska inte skriva avbrottsrutinen, bara utgå från att den finns. I funktionen `notify_input` ska det inlästa tecknet tas om hand och enheten sedan ställas i ett tillstånd som tillåter en ny överföring. I denna deluppgift behöver du inte kontrollera att ett inläst tecken ryms i bufferten.

Deluppgift b (6p)

Konstruera den del av drivrutinerna som har att göra med sändning. Utöka filen `driver.h` med deklarationer av följande tre funktioner: `init_output` som initierar sändningsdelen av enheten, `get_output_buffer` som ger en pekare till den utbuffert som drivrutinen använder och `notify_output` som egentligen anropas varje gång ett avbrott har skett men som även måste kunna anropas från andra delar av programmet för att ”väcka” sändaren när den varit overksam.

Lägg till definitioner av dessa tre funktioner i filen `driver.c`. Initieringsfunktionen ska fungera på liknande sätt som för mottagninsdelen. Funktionen `notify_output` anropas från en avbrottsrutin men du ska inte skriva denna avbrottsrutin heller.

I funktionen `notify_output` ska normalt nästa tecken hämtas från utbufferten och sändas ut, men det ska finnas en koppling mellan mottagardelen och sändardelen av enheten. I deluppgift a behövde du inte ta hänsyn till om ett mottaget tecken fick plats i inbufferten, men det ska du göra nu. Det ska fungera på följande sätt.

Varje gång man i funktionen `notify_input` fått in ett tecken ska man kvittera det. Om det mottagna tecknet kunde läggas i inbufferten ska man skicka tecknet ACK (ASCII-kod 6, hexadecimalt) som svar och om tecknet inte kunde tas emot ska man skicka tecknet NACK (ASCII-kod 15, hexadecimalt). Själva sändningen ska förstås göras med hjälp av sändningsdelen. Sändning av dessa kvitton har prioritet framför sändning av normala data, vilket innebär att funktionen `notify_output` alltid måste kontrollera om ACK eller NACK väntar på att sändas innan den hämtar ett tecken från utbufferten. För att förenkla det något får du förutsätta att högst ett ACK- eller NACK-tecken väntar på att sändas i varje ögonblick.

Du måste här förstås också visa vilka tillägg som ska göras i funktionen `notify_input` från deluppgift a för att det hela ska fungera.

Bilaga 1: Kompilatorkonvention XCC12:

- Parametrar överförs till en funktion via stacken och den anropande funktionen återställer stacken efter funktionsanropet.
- Då parametrarna placeras på stacken bearbetas parameterlistan från höger till vänster.
- Lokala variabler översätts i den ordning de påträffas i källtexten.
- *Prolog* kallas den kod som reserverar utrymme för lokala variabler.
- *Epilog* kallas den kod som återställer (återlämnar) utrymme för lokala variabler.
- Den del av stacken som används för parametrar och lokala variabler kallas *aktiveringspost*.
- Beroende på datatyp används för returparameter HC12:s register enligt följande tabell:

Storlek	Benämning	C-typ	Register
8 bitar	byte	char	B
16 bitar	word	short int *och* pekartyp	D
32 bitar	long float	long int float	Y/D

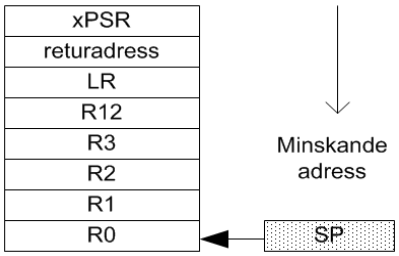
Bilaga 2: Assemblerdirektiv för MC68HC12.

Assemblerspråket använder sig av mnemoniska beteckningar som tillverkaren Freescale specificerat för maskininstruktioner och instruktioner till assemblatorn, s.k. pseudoinstruktioner eller assemblerdirektiv. Pseudoinstruktionerna framgår av följande tabell:

Direktiv	Förklaring
`ORG N`	Placerar den efterföljande koden med början på adress N (ORG för ORiGin = ursprung)
`L: RMB N`	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adress L. (RMB för Reserve Memory Bytes)
`L: EQU N`	Ger label L konstantvärdet N (EQU för EQUates = beräknas till)
`L: FCB N1,N2..`	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FCB för Form Constant Byte)
`L: FDB N1,N2..`	Avsätter i följd i minnet ett bytepar (två bytes) för varje argument. Respektive bytepar ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L. (FDB för Form Double Byte)
`L: FCS "ABC"`	Avsätter en följd av bytes i minnet, en för varje tecken i teckensträngen "ABC". Respektive byte ges ASCII-värdet för A, B, C etc. Följden placeras med början på adress L. (FCS för Form Caracter String)

Bilaga 3: Rättelser, tillägg till ”Quick Guide för MOP”

Quick guiden ska kompletteras med följande figur:



Stackens utseende vid undantagshantering

Quick guiden ska kompletteras med följande tabell:
Assemblerdirektiv:

Direktiv	Förklaring
`L: .SPACE N`	Avsätter N bytes i följd i minnet (utan att ge dem värden), så att programmet kan använda dem. Följden placeras med början på adress L.
`L: .BYTE N1,N2..`	Avsätter i följd i minnet en byte för varje argument. Respektive byte ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
`L: .HWORD N1,N2..`	Avsätter i följd i minnet ett 16 bitars ord för varje argument. Respektive ord ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
`L: .WORD N1,N2..`	Avsätter i följd i minnet ett 32 bitars ord för varje argument. Respektive ord ges konstantvärdet N1, N2 etc. Följden placeras med början på adress L.
`.ALIGN`	Garanterar att påföljande adress är jämnt delbar med 4 (”word aligned”)

Sidan 10: växlat BCC och BCS, ska vara:

C-operator	Betydelse	Datatyp	Instruktion
==	Lika	signed/unsigned	BEQ
!=	Skild från	signed/unsigned	BNE
<	Mindre än	signed	BLT
		unsigned	BCC
<=	Mindre än eller lika	signed	BLE
		unsigned	BLS
>	Större än	signed	BGT
		unsigned	BHI
>=	Större än eller lika	signed	BGE
		unsigned	BCS

Sidan 12, fel basadresser för USART, ska vara:

USART
Universal synchronous asynchronous receiver transmitter
USART1: 0x40011000
USART2: 0x40004400

Sidan 17, beskrivning av EXTI_PR, ska vara:

EXTI_PR *Pending 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	mnemonic
0x14										rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	EXTI_PR

Bit PR[22..0]:
Motsvarande bit sätts i detta register då ett triggvillkor är uppfyllt. Biten återställs genom att skrivas med '1'.
0: Ingen Trigg.
1: Trigg har uppträtt

Lösningsförslag

Uppgift A-1:

a) RESET är en asynkron signal till processorn med syftet att återställa processorns i ett väldefinierat bygynnelsetillstånd.

b) Processorn sparar alla registerinnehåll på stacken, läser adressen till avbrottshanteringsrutinen från avbrottsvektorn och placerar denna i PC.

c) För att undvika en rekursiv behandling av samma avbrott.

d) Modifiera det CC-innehåll som återställs från stacken vid RTI, dvs:

```
LDAB    0,SP
ORAB    #$10
STAB    0,SP
```

e)

```
LDX #irq
STX 0x3e00
*( ( unsigned short *) 0x3e00) =  (void) (**irq) (void);
```

Uppgift A-2:

```
;  int  convert(char *cp )
;   Parametrar:    'cp' i register X vid anrop
;   resultatet i register Y vid utträde
convert:
; {
;     int converted = 0;
    LDY     #0
;     while( *cp ){
convert_2:
    TST     ,X
    BEQ     convert_3
;       if( *cp < 0 ){
    BGE     convert_4
;           *cp = -(*cp);
    NEG     ,X
;           converted++;
    INY
convert_4:
    INX
;       }
;       cp++;
;       }
    BRA     convert_2
convert_3:
;       return converted;
;    }
    RTS
```

Uppgift A-3:

a)

```
_p          RMB     1
_a          RMB     2
_g_array    RMB     10
_k          RMB     2
```

b)

```
            LDD     _g_array+6
            STD _a
```

c)

```
getY:
            TFR     Y,D
            RTS
```

Uppgift B-1:

```
#define    STK_CTRL          ((volatile unsigned int *)    0xE000E010 )
#define    STK_LOAD          ((volatile unsigned int *)    0xE000E014 )
#define    STK_VAL           ((volatile unsigned int *)    0xE000E018 )
#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_HIGH_IDR   ((volatile unsigned char *)   (PORT_D_BASE+0x11))
#define    GPIO_D_LOW_ODR    ((volatile unsigned char *)   (PORT_D_BASE+0x14))
#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_PUPDR      ((volatile unsigned int *)    (PORT_E_BASE+0xC))
#define    GPIO_E_LOW_IDR    ((volatile unsigned char *)   (PORT_E_BASE+0x10))
```

a)

```
void init_app( void )
{
    /*  PORT D, b15-8  ingångar, b7-b0 utgångar  */
    *GPIO_D_MODER  = 0x00005555;
    *GPIO_D_PUPDR  = 0xAAAA0000;      /* pull down */
    *GPIO_D_OTYPER  = 0x00000000;     /* push/pull */
    /*  PORT E, b7 utgång , b6-0 ingångar    */
    *GPIO_E_MODER  = 0x4000;
    *GPIO_E_PUPDR  = 0x2AAA;      /* b6-0 pull down */
    *GPIO_E_OTYPER  = 0x0000;     /* b7 push/pull */
}
```

b)

```
static int d_valid, irq_count;

void   systick_irq_handler ( void )
{
    irq_count--;
    if( irq_count <= 0 )
    {   /* 100 ms har passerat… */
        *STK_CTRL = 0;
        d_valid = 1;
    }
}
void start_timeout( void )
{
    *((void (**)(void) ) 0x2001C03C ) = systick_irq_handler;
    *STK_CTRL = 0;
    *STK_LOAD = ( 1680000 - 1 );  /* 10 ms */
    *STK_VAL = 0;
    *STK_CTRL = 7;
    d_valid = 0;
    irq_count = 10;
}
int is_timeout(void)
{
    return d_valid;
}
```

c)

```
void main(void)
{
    unsigned char c, peripheral_input;
    init_app();
    peripheral_input = 0;
    d_valid = 0;
    *GPIO_E_LOW_ODR = 0;
    while(1)
    {
        if( (*GPIO_E_LOW_ODR & 0x80 ) ==0)
        {   /* initiera periferikrets... */
            *GPIO_E_LOW_ODR = 0x80;   /* S = 1 */
            start_timeout();
        }
        c = *GPIO_D_HIGH_IDR & 0xC0;  /* bit 7 och 6 */
        if( is_timeout() && (*GPIO_E_LOW_IDR & 0x40) )
        {   /* Periferikrets klar... */
            peripheral_input = *GPIO_D_HIGH_IDR & 0x3F;
            *GPIO_E_LOW_ODR = 0;  /* S = 0 */
        }
        GPIO_D_LOW_ODR = c | peripheral_input ;
    }
}
```

Uppgift B-2:

```
unsigned int func( unsigned int a, unsigned int b )
{
   if( countOnes(a)  >= b  )
      return ~b;
   else
      return b;
}
```

```
func:
    PUSH    {R1,LR}
**** int max( unsigned int a, unsigned int b)
**** {
****    if( countOnes(a) >= b )
    BL      countOnes
    POP     {R1}
    CMP     R0,R1
    BCS     L1
****    else
    MOV     R0,R1       @ ”return b”
    B       L2
L1: MVN     R0,R1       @ ”return ~b”
**** }
L2: POP     {PC}

@ Konvention säger att register R0,R1 används för parametrar.
@ Returvärden kommer också i R0
callfunc:
****
void    callfunc(void )
**** {
    PUSH    {LR}
****    e = min( c, d);
    LDR     R0,c
    LDR     R1,d
    BL      func
    LDR     R3,=e
    STR     R0,[R3]
    POP     {PC}

**** int   c,d,e;
    .ALIGN
c:  .SPACE  4
d:  .SPACE  4
e:  .SPACE  4
```

Uppgift B-3:

Endast de bitar som konfigurerar EXTI10 får/ska initieras.

```
#define SYSCFG_EXTICR3 0x40013810
#define EXTI_IMR       0x40013C00
#define EXTI_FTSR      0x40013C0C
#define EXTI_RTSR      0x40013C08
#define NVIC_ISER1     0xE000E104

*((volatile unsigned short *) SYSCFG_EXTICR3) &= 0xF0FF;    /* nollställ bitfält EXTI10 */
*((volatile unsigned short *) SYSCFG_EXTICR3) |= 0x0300;    /* PD10->EXTI10  */
*((volatile unsigned int *)   EXTI_IMR) |= (1<<10);         /* aktivera avbrott EXTI10 */
*((volatile unsigned int *)   EXTI_RTSR) |= (1<<10);        /* aktivera trigger på positiv flank */
*((volatile unsigned int *)   EXTI_FTSR) &= ~(1<<10);       /* deaktivera trigger på negativ flank */
*((volatile unsigned int *)   NVIC_ISER1) |= (1<<8);        /* aktivera avbrott i NVIC */
```

Vektor nummer 40 (offset 0xE0)

Uppgift C-4:

```
int bitcheck( unsigned int *p, int *num)
{   /* Vi räknar ettorna, det är enklast…*/
    *num = 0;
    while(*p)
    {
        if( *pp & 1 ) *num++;
        *p >>= 1;
    }
    *num = 32 - *num;  /* Antal nollor */
    return !(*num & 7);
}
```

Uppgift C-5

```
/* filen driver.h */
/* deluppgift a */
#include "buffer.h"
void init_input(void);
buffer *get_input_buffer();

/* deluppgift b */
void init_output(void);
buffer *get_output_buffer();
void notify_output(void);

/* filen channel.h */
typedef void (*vec) (void);  /* avbrottsvektor */
typedef vec *vecptr;         /* pekare till avbrottsvektor */

/* till deluppgift a */
#define RECEIVE_DATA (*(unsigned char *) 0x400 )
#define RECEIVE_CTRL (*(unsigned char *) 0x401 )
#define RECEIVE_VEC_ADR  0xFFE8
#define RECEIVE_VEC      *((vecptr) RECEIVE_VEC_ADR)
#define RDY 0x01
#define GO  0x02
#define IE  0x04

/* till deluppgift b */
#define SEND_DATA (*(unsigned char *) 0x402 )
#define SEND_CTRL (*(unsigned char *) 0x403 )
#define SEND_VEC_ADR  0xFFEC
#define SEND_VEC      *((vecptr) SEND_VEC_ADR)
#define ACK  0x06
#define NACK 0x15

/* filen driver.c */
/* deluppgift a */
#include "driver.h"
#include "channel.h"
#include "buffer.h"
#define BUF_SIZE 100

static buffer *input_buf = 0;

void init_input(void) {
     if (!input_buf) {
        input_buf = create_buffer(BUF_SIZE);
        RECEIVE_CTRL = IE;
     }
}

buffer *get_input_buffer() {
  return input_buf;
}

void notify_input(void) {   /* anropas av input_intr  */
   if (RECEIVE_CTRL & RDY) {
     int ok = put(input_buf, RECEIVE_DATA);
     RECEIVE_CTRL = GO & IE;
     /* tillägg i deluppgift b */
     if (ok)
       answer = ACK;
     else
       answer = NACK;
     notify_output();
  }
}

/* deluppgift b */
static char answer = 0;
static buffer *output_buf = 0;

void init_output(void) {
     if (!output_buf) {
        output_buf = create_buffer(BUF_SIZE);
     }
     SEND_CTRL = RDY & IE;
}

buffer *get_output_buffer() {
  return output_buf;
}

void notify_output(void) {  /* anropas av output_intr eller av användare */
```

```
   char c;
   if (SEND_CTRL & RDY) {
     if (answer) {
       SEND_DATA = answer;
       answer = 0;
     }
     else if (get(input_buf, &c)) {
       SEND_DATA = c;
     }
     SEND_CTRL = GO & IE;
  }
}
```