Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

# Hemtentamen

**EDA482 Maskinorienterad programmering D**
**EDA487 Maskinorienterad programmering Z**
**DIT151 Maskinorienterad programmering GU**

## Fredag 4 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 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

Ett program, givet i C, ska översättas till ARMv6 assemblerspråk.

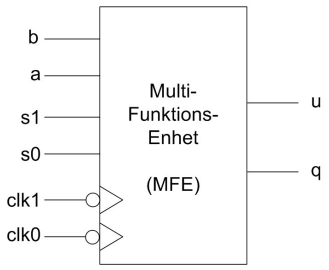
- Assemblerkoden ska kommenteras på sådant sätt att det tydligt framgår vilka delar av C-koden som översatts. Varje enskild instruktion behöver dock inte kommenteras så länge kodningen sker enligt de principer vi använt i kursen.
- Lokala variabler ska allokeras till register och användningen av register ska beskrivas utförligt med kommentarer.
- En lösning kan 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 minnestrafiken minimeras (åtskilliga load/store) försvinner. Deklarationer och referenser till globala variabler ska göras enligt anvisningar i kursen. En lösning som bara innehåller "dump" av kompilerad kod bedöms 2/0/1.

```
void minmax( char str[], int length, char *min, char *max )
{
   *min=0xFF;
   *max=0;
   for( int i=0; i< length; i++ )
   {
      if( *min > str[i] )  *min = str[i];
      if( *max < str[i] ) *max = str[i];
   }
}

char  teststring[32];
char min, max;

void main(void)
{
   int length = 15;
   minmax( teststring, length, &min, &max );
}
```

## 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 synkrona, 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:



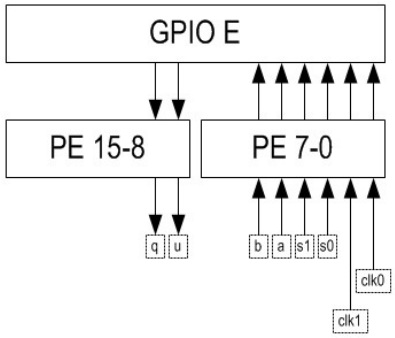

| s1 | s0 | u |
|---|---|---|
| 0 | 0 | a&&b |
| 0 | 1 | a\|\|b |
| 1 | 0 | !(a&&b) |
| 1 | 1 | !(a\|\|b) |

| clk1, clk2 | q |
|---|---|
| positiv flank clk0 | - |
| negativ flank clk0 | a |
| positiv flank clk1 | - |
| negativ flank clk1 | b |

Dvs. de asynkrona ingångarna `a` och `b` bestämmer utsignalen `u` via logikfunktioner. Selektorsignalerna `s1` och `s0` avgör vilken logikfunktion som ska appliceras. Den synkrona utsignalen `q` påverkas av `clk0` repektive `clk1`, båda vid negativ flank, ingen ändring sker vid positiv flank på klockingångarna.

GPIO-port `E` används för MFE:ns anslutningar enligt följande:

`q` ansluts till `PE9`
`u` ansluts till `PE8`
`b` ansluts till `PE5`
`a` ansluts till `PE4`
`s1` ansluts till `PE3`
`s0` ansluts till `PE2`
`clk1` ansluts till `PE1`
`clk0` ansluts till `PE0`




Det synkrona beteendet implementeras med hjälp av avbrott (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 (`a` eller `b`) från portpinnar, bestämmer utsignalen `q` och skriver till portpinnen.

c) Visa en funktion `char evaluate( char a, char b, 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 system för att generera ett enkelt pulståg ska konstrueras. Systemet byggs kring funktionen:

```
void pulsetrain( char sequense[], int length )
```

Ett pulståg anges med en längd `length`, dvs. antalet pulser och en vektor `sequense`, med samma längd. För varje pulsintervall (1 ms) anger vektorn om pulsen ska vara låg eller hög i intervallet. SYSTICK 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. Du får förutsätta att USART-kretsen är korrekt initierad.

Följande illustration ger exempel på förväntad utskrift från `pulsetrain` för tre olika testvektorer.

```
char test1[4] =  {1,0,1,0};
char test2[8] =  {1,0,0,1,1,1,0,1};
char test3[12] = {1,0,1,1,0,0,1,0,1,1,1,0};
   ...
   pulsetrain( test1, 4 );
   _outchar('\n') ;
   pulsetrain( test2, 8 );
   _outchar('\n') ;
   pulsetrain( test3, 12 );
   _outchar('\n') ;
```

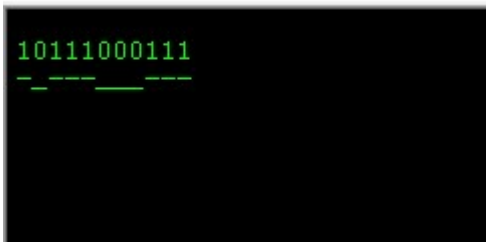
Terminal

Utskriften på terminalen blir då:

Ett testprogram (`main`) använder terminalen för inmatning av en texsträng, textsträngen får förutsättas bestå enbart av ASCII-tecknen ’0’ eller ’1’.
Inmatning avslutas med Enter-tangenten (`’\n’`).

Till höger visas ett exempel på användning av det färdiga testprogrammet, för varje tecken (0 eller 1) som matas in skriver inmatningsrutinen tillbaks till terminalen.
Efter att inmatningen avslutats skrivs motsvarande pulståg till terminalen.

Terminal




**Ledningar:**

- Konstruera en inmatningsrutin `get_line` som läser in en textsträng från terminalen. Låt `get_line` returnera antalet tecken i strängen (’0’ eller ’1’) .
- Tänk på att vektorn som skickas till `pulsetrain` ska innehålla de binära talen (0 eller 1)
- Du får förutsätta att den längsta sträng som kan matas in är 32 tecken.

Följande funktioner ska alltså konstrueras:

- En blockerande fördröjningsfunktion som använder SYSTICK.
- En funktion `pulsetrain` som skriver pulståget 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 övervakningssystem har installerats i en gruvgång där 4 givare placerats ut på olika ställen. Varje givare skickar en analog signal för omvandling i övervakningssystemets AD-omvandlare till ett positivt heltal. Givarna reagerar på andelen koloxid i gruvgången, där värdet 0 är 0% och värdet 4095 motsvarar 100%.

Ingången till gruvgången har också försetts med trafikljus, en grön, en gul och en röd lampa.

För att gruvgången ska betraktas som *säker* krävs att de fyra givarna samtidigt indikerar mindre än 25% koloxidhalt. Om alla 4 givare samtidigt indikerar större än 50% koloxidhalt betraktas gruvgången som *farlig*. I övriga fall betraktas gruvgången som *osäker*.

Översikt av övervakningssystemets 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 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | MINE_CR |
| 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | MINE_ADC |

Beskrivningar av register:

| offset | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | Register |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | LIGHT | | | | | RDY | | CNV | SELECT | | MINE_CR |

SELECT, läs- och skrivbara bitar. Bitfält för val av givare (0..3).

CNV: Conversion, skrivbar. Då denna bit sätts till 1, nollställs först RDY-biten av hårdvaran, därefter startas en omvandling av det analoga värdet från den givare som valts. Då omvandlingen är klar ställer hårdvaran om RDY-biten till 1 och nollställer CNV.

RDY: Ready: Nollställs av hårdvaran då en AD-omvandling startas, ettställs av hårdvaran då omvandlingen är klar och värde kan läsas från dataregistret.

LIGHT: läs- och skrivbara bitar. Bitfält för att tända/släcka indikatorlampor Grön/Gul/Röd. Då biten sätts till 1 tänds motsvarande indikatorlampa vid gruvgången.

| offset | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | Register |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | | | | | r | r | r | r | r | r | r | r | r | r | r | r | MINE_ADC |

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`, som kontinuerligt omvandlar varje givarvärde, och, baserat på resultatet tänder motsvarande indikatorlampa, *säker*=grön, *osäker*=gul, *farlig*=röd.