Laboration 1: Parallell in- och utmatning

Laboration 1 består av fem uppgifter. Den första uppgiften beskrivs utförligt i läroboken och de två andra uppgifterna baseras på lösningar av den första uppgiften. Du bör därför lösa uppgifterna i ordning.

Laborationsuppgifterna baseras på följande exempel och uppgifter i läroboken:

- Exempel 2.27 och uppgift 2.10. (Sju-sifferindikator)
- Exempel 4.1 t.o.m 4.4, uppgifter 4.1 och 4.2. (Enkelt tangentbord ”Keypad”).

För att kunna skapa projekt, implementera och testa dina program behöver du dessutom ha arbetat igenom kapitel 3.

För en fördjupad förståelse kan du också studera dokumenten:

- Beskrivning av 7-segmentsdisplay
- Beskrivning av Keypad

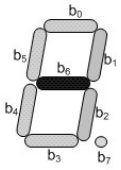
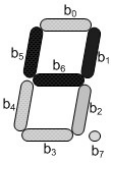
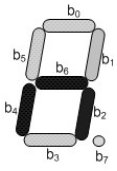
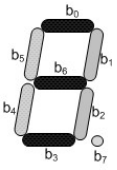
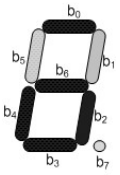
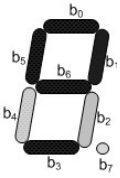
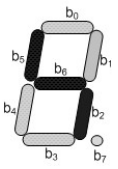
Laborationsuppgift 1.1: (Uppgift 4.2 i läroboken)

Skriv en applikation som kontinuerligt läser av tangentbordet. Om någon tangent är nedtryckt, ska dess hexadecimala tangentkod skrivas till 7-segmentsdisplayen. Om ingen tangent är nedtryckt ska displayen släckas. Applikationen ska vara komplett med startup, main, initieringsfunktion och portdefinitioner, etc.

- Kontrollera programmets funktion med *CodeLite* och *SimServer*.

Laborationsuppgift 1.2:

Genom att modifiera segmentskodstabellen i funktionen `out7seg`, kan vi skapa nya ”teckenuppsättningar. Du ska nu skapa en funktion `void outEmoji( unsigned char c )` som matar ut en av följande figurer baserat på parametern `c`:

Parameter= 0	Parameter= 1	Parameter= 2	Parameter= 3	Parameter= 4	Parameter= 5	Övriga parameter-värden
						
”neutral”	”happy”	”sad”	”angry”	”loud”	”stunned”	”confused”

- Skapa ett nytt projekt `emoji`.
- Implementera den nya funktionen `void outEmoji( unsigned char c )`.
- Använd huvudprogrammet till höger för att testa funktionen:
- Kontrollera programmets funktion med *CodeLite* och *SimServer*.

```
void main(void)
{
  unsigned char c;
  init_app();
  while( 1 )
  {
    outEmoji ( keyb() );
  }
}
```

I uppgift 1.1 används en enkel tangentbordsrutin som avspegalar ett ”ögonblicksvärde”. Vi kan upptäcka att en tangent är nedtryckt genom att kontinuerligt undersöka tangentbordet. Vi kan dock inte avgöra hur många gånger tangenten tryckts ned. För detta måste vi försäkra oss om att tangenten dessutom släppts upp. I nästa uppgift åstadkommer vi det genom att använda en (global) tillståndsvariabel som ”kommer i håg” att en tangentnedtryckning detekterats.

Laborationsuppgift 1.3:

Utgå från uppgift 1.1. Tangentbordsrutinen ska modifieras så att varje nedtryckning detekteras exakt en gång. Följande algoritm kan användas för uppgiften

```
Algoritm: keyb_enhanced:
returnerar kod för nedtryckt tangent en gång.
tillståndsvariabel anger: initialtillstånd eller väntetillstånd

Keyb_state = initialtillstånd;

keyb_enhanced;
        Om väntetillstånd
                Aktivera samtliga rader;
                Om någon tangent fortfarande nedtryckt
                        Stanna i väntetillstånd
                annars
                        Tangent har släppts upp. Återgå till initialtillstånd
                returnera 0xFF;
        Om initialtillstånd
                Avsök tangentbord, om någon tangent ndetryckt:
                        Övergå till väntetillstånd
                        returnera tangentkod
```

För att aktivera samtliga rader för avsökning kan exempelvis `kbdActivate` kompletteras enligt följande:

```
switch( row )
{
  ...
  case 5: *GPIO_D_ODRHIGH = 0xF0; break;
}
```

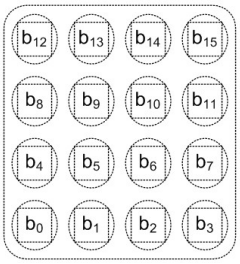
- Skapa ett nytt projekt `keyb_enhanced`.
- Implementera den nya tangentbordsrutinen `unsigned char keyb_enhanced(void)`.
- Använd följande huvudprogram för att testa funktionen:

```
void main(void)
{
  unsigned char c;
  init_app();
  while( 1 )
  {
    c = keyb_enhanced();
    if( c != 0xFF )
      out7seg( c );
  }
}
```

- Kontrollera programmets funktion med *CodeLite* och *SimServer*.

Laborationsuppgift 1.4:

I vissa fall räcker det inte med att kunna detektera exakt en nedtryckt tangent. Man vill i bland kunna detektera flera samtidigt nedtryckta tangenter, 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 figuren till höger mellan bitposition och tangent:



Följande gäller då för avbildning mellan *tangentkod* och bitposition:

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

Även denna uppgift baseras i viss mån på uppgift 1.1. Tangentbordsrutinen ska utformas så att varje nedtryckt tangent detekteras och indikeras med en bit i ett statusord som returneras av rutinen. Följande algoritm ska användas:

```
Algoritm: keyb_alt_ctrl
Keyb_status = 0;
Aktivera rad 4, placera kolumnvärdena i Keyb_status( b0..b3)
Aktivera rad 3, lägg till kolumnvärdena i Keyb_status( b4..b7)
Aktivera rad 2, lägg till kolumnvärdena i Keyb_status( b8..b11)
Aktivera rad 1, lägg till kolumnvärdena i Keyb_status( b12..b14)
returnera Keyb_status
```

- Du ska konstruera `unsigned short keyb_alt_ctrl(void)` baserat på den givna algoritmen.
- Du ska nu 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.

Det är lämpligt att definiera bitmasker för tangentkod eller grupp av tangentkoder. För att avgöra om exempelvis någon av tangenterna C (”Ctrl”) eller A (”Alt”) är nedtryckta kan vi använda:

```
#define      Ctrl   0x80
#define      Alt    0x8000
```

För att filtrera ut de numeriska tangenterna är det lämpligt att på samma sätt definiera en bitmask som motsvarar denna grupp.

- Skapa ett nytt projekt `keyb_alt_ctrl`.
- Implementera funktionen `unsigned char is_numeric( unsigned short s ).`
- Använd huvudprogrammet till höger för att testa funktionen.
- Kontrollera programmets funktion med *CodeLite* och *SimServer*.

```
void main(void)
{
  unsigned short s;
  unsigned char c;
  init_app();

  while( 1 )
  {
    s = keyb_alt_ctrl();
    c = is_numeric( s );
    if( c != 0xFF )
      out7seg( c );
  }
}
```

Laborationsuppgift 1.5:

Modifiera funktionerna `out7seg` och `outEmoji` från uppgift 1.2 så att dessa kan skriva ut decimalpunkten på 7-sifferindikatorn:

```
out7seg( unsigned char c, int dp );
void outEmoji( unsigned char c, int dp );
```

Om parametern `dp` är skild från 0 ska även decimalpunkten tändas.

- Skapa ett nytt projekt `decimal_point`.
- Implementera funktionen `unsigned char is_numeric( unsigned short s )`.
- Använd huvudprogrammet till höger för att testa funktionen.
- Kontrollera programmets funktion med *CodeLite* och *SimServer*.

```
void main(void)
{
  unsigned short s;
  unsigned char c;
  int dp;
  init_app();

  while( 1 )
  {
    s = keyb_alt_ctrl();
    dp = s & Alt;
    c = is_numeric( s );
    if( c != 0xFF )
    {
      if( s & Ctrl )
        outEmoji( c, dp );
      else
        out7seg( c, dp );
    }else
      *GPIO_D_ODRLOW = 0;
    }
}
```