Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA

Tentamen med lösningsförslag

EDA481 Programmering av inbyggda system D
EDA486 Programmering av inbyggda system Z
DAT016 Programmering av inbyggda system IT
DIT152 Programmering av inbyggda system GU

Torsdag 2 juni 2016, kl. 14.00 - 18.00

Examinator
Roger Johansson, tel. 772 57 29

Kontaktperson under tentamen:
Roger Johansson

Tillåtna hjälpmedel

Häftet

Instruktionslista för CPU12
Quick Guide, Laborationsdator MD407 med tillbehör

Inget annat än rättelser och understrykningar får vara införda i häftena.
Du får också använda bladet:

C Reference Card

samt *en* av böckerna:

Vägen till C,
Bilting, Skansholm

C från början
Skansholm

The C Programming Language,
Kerninghan, Ritchie (även utskrift av PDF-versionen, hel eller delvis)

Endast ”överstrykningar” och understrykningar får vara införda i boken.

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 givits under kursen och tillräckligt dokumenterade.
- 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 (8p) *Kodningskonventioner (C/assemblerspråk)*

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

Inledningen (parameterlistan och lokala variabler) för en funktion ser ut på följande sätt:

```
void func( char *b, char a )
{
   char c;
   char  *d; ....
```

dessutom har följande C-deklarationer gjorts på ”toppnivå” (global synlighet):

```
char    *e,f;
```

a) Visa hur variabeldeklarationerna översätts till assemblerdirektiv för HCS12.

b) Visa också hur följande funktionsanrop översätts till assemblerkod för HCS12:

```
func( e , f );
```

c) Visa hur utrymme för lokala variabler skapas och aktiveringspostens utseende (dvs. stacken) i funktionen '`func`'.

Uppgift A-2 (6p) *Assemblerprogrammering*

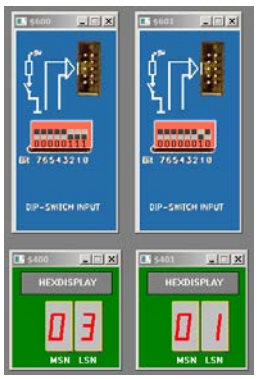
Subrutinen Outzero sköter utmatningen av styrsignaler (styrordet) till en kringenhet. Endast *en av* styrsignalerna kan nollställas åt gången. Porten (`DrillControl`) är inte läsbar så en kopia av styrordet lagras i vaiabeln `DCShadow`. Subrutinen Outzero specificeras av följande:

```
; Subrutin Outzero.
; Läser kopian av borrmaskinens styrord på adress 'DCShadow'. Nollställer en
; av bitarna och skriver det nya styrordet tillbaka till kopian
; 'DCShadow' samt till utporten 'DrillControl'.
; Biten som nollställs ges av innehållet i B-registret (0-7) vid anrop.
; Om (B) > 7 utförs ingenting.
; Anrop:       LDAB   #bitnummer
;          JSR    Outzero
; Bitnumrering framgår av följande:
```

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
7	6	5	4	3	2	1	0

Uppgift A-3 (8p) *In och utmatning beskriven i C*

I denna uppgift ska du bland annat demonstrera hur absolutadressering utförs i C. För full poäng ska du visa hur preprocessordirektiv och ev. typdeklarationer används för att skapa begriplig programkod.



Två 8-bitars strömbrytare, är anslutna till adresserna 0x600, 0x601 och två displayenheter som var och en visar en byte i form av två hexadecimala siffror är anslutna till adresserna 0x400 och 0x401 i ett MC12 mikrodatorsystem. Konstruera en funktion

```
void DivModHex( void )
```

som läser de båda strömbrytarnas inställda värden.

Om värdet på adress 0x601 är noll ska ***FF*** visas på båda displayenheter. Om värdet på adress 0x601 är skilt från noll ska resultatet från heltalsdivisionen mellan värden på adress 0x600 och 0x601 visas på displayindikator med adress 0x400 och resultatet från restdivisionen av samma tal visas på indikator med adress 0x401.

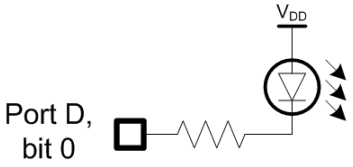
Uppgift A-4 (6p) *Avbrott, HCS12*

Redogör för hur avbrott går till i ett HCS12-system, med en yttre enhet ansluten till IRQ-ingången på processorn. Ditt svar skall bland annat innehålla

- en övergripande beskrivning av vad avbrott är och hur det fungerar
- en beskrivning av de olika programrutiner som är förknippade med avbrott (eventuellt pseudokod / assembler)

Uppgift B-1 (10p)

En ljusdiod har anslutits till en pinne hos port D enligt figuren till höger.

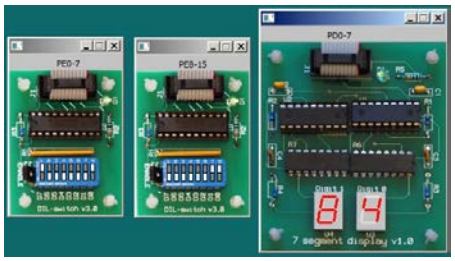


Skriv ett program, i C, som kontinuerligt tänder dioden under en halv sekund och därefter släcker den under en halv sekund. Din lösning ska fördelas på följande deluppgifter:

a) Visa hur SysTick kan användas för att skapa en (blockerande) fördröjning om 1ms, med funktionen `void delay_1ms( void);`. Systemets klockfrekvens är 168 MHz. (6p)
b) Skriv en funktion `void init_app( void );` som sätter upp port D, bit 0 som utsignal, ”push-pull”. Övriga inställningar i porten ska behållas, dvs. får inte ändras av initieringen. (2p).
c) Skriv huvudprogrammet som får dioden att blinka. Du kan använda `delay_1ms()` och `init_app()` även om du inte besvarat a) och/eller b). (2p)

Uppgift B-2 (6p)

Två DIL-strömbrytare och en 7-segments display kopplas enligt:



- DIL1 till Port E (0-7),
- DIL2 till Port E (8-15)
- Display till port D (0-7):

Skriv ett program i ARM/Thumb assemblerspråk som:

- Initierar portar D och E.
- Kontinuerlig läser de inställda värdena från DIL-strömbrytarna, adderar dessa och skriver resultatet till displayen. Om summan är större än 255 ska FF skrivas till displayen.

Uppgift B-3 (8p)

Förbered en enkel applikation som använder PE2 hos MD407 som avbrottsingång. Beskriv initieringsfunktioner och avbrottshantering, dvs. färdigställ följande:

```
void irq_handler ( void )
{
    Om avbrott från EXTI2:
        kvittera avbrott från EXTI2
}

void enable_interrupt( void )
{
    Nollställ processorns avbrottsmask
}

void app_init ( void )
{
    Koppla PE2 till avbrottslina EXTI2;

    Konfigurera EXTI2 för att generera avbrott;
    Konfigurera för avbrott på negativ flank

    Sätt upp avbrottsvektor

    Konfigurera de bitar i NVIC som kontrollerar den avbrottslina som EXTI2 kopplats till
}
```

Uppgift B-4 (4p)

Visa hur en GPIO-modul kan beskrivas med en sammansatt datatyp `struct`, i C.

Uppgift C-5 (6p) *Programmering med pekare*

Skriv C-funktionen `char *strrchr(const char *str, int c)`.

Funktionen ingår i standard C biblioteket och specificeras av följande:

Description: The C library function `char *strrchr(const char *str, int c)` searches for the last occurrence of the character `c` (an `unsigned char`) in the string pointed to, by the argument `str`.
Declaration: Following is the declaration for `strrchr()` function.

```
char *strrchr(const char *str, int c)
```

Parameters
`str` -- This is the C string.
`c` -- This is the character to be located. It is passed as its int promotion, but it is internally converted back to `char`.
Return Value: This function returns a pointer to the last occurrence of character in `str`. If the value is not found, the function returns a null pointer.

Du får *inte* använda dig av indexering, utan måste utnyttja pekare. Du får *inte* anropa någon annan standardfunktion. Du måste alltså skriva all kod själv.

Uppgift C-6 (16p) *Maskinnära programmering i C*

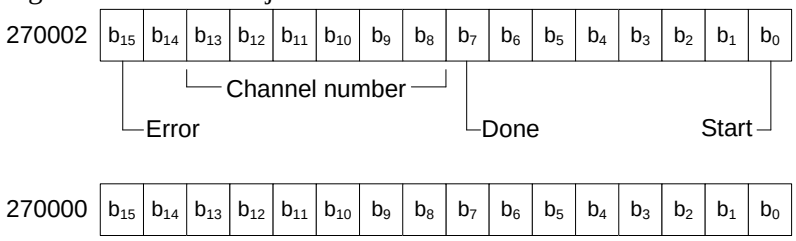
a) Varför behöver vi i bland använda det reserverade ordet `volatile`? (2p)

b) När behöver vi använda så kallade *include-guards*? (2p)

c) Ange 2D-index för talvärdet 10 i följande deklaration. (1p)

```
int arrayOfArrays[3][4] = { {1,2,3,4}, {5,6,7,8}, {9,10,11,12} };
```

d) En signalenhet som läser externa signaler är ansluten till en dator. Signalenheten har 64 ingångar (kanaler), numrerade från 0 till 63. Signalenheten kan bara avläsa en ingång i taget. Det avlästa värdet ges som ett positivt 16-bitars tal.
Signalenheten kopplas till datorn via ett dataregister och ett styrregister. Dessa register består båda av 16-bitar och de har de oktala adresserna 270000 resp. 270002. Dataregistret innehåller det avlästa värdet. Styrregistret används för att initiera avläsningar och kontrollera signalenhetens tillstånd.
Styrregistret innehåller följande bitar:



Bit	Namn	Användning
0	start	När en etta skrivs i detta startas en avläsning
7	Done	Skall sättas till 0 före avläsningen. Sätts automatiskt till 1 när avläsningen är klar.
8-13	Channel number	Här anger man vilken av de 64 ingångarna som skall avläsas.
15	Error	Sätts till 1 av signalenheten om avläsningen misslyckades.

När man skriver i styrregistret måste man skriva *alla 16-bitarna på en gång*.

Skriv en modul (med en .h fil: `sig_reader.h` och en .c fil: `sig_reader.c`) som innehåller två C-funktioner, `sig_read`, som initierar avläsning och `sig_get_value` som ger det avlästa värdet som resultat. Du måste också skriva de definitioner av typer och portar som behövs. Dessa definitioner ska placeras i headerfilen.
Funktionen `sig_read` skall som parameter få numret på den ingång som skall avläsas. Detta värde skall ligga i intervallet 0 till 63. Om ett felaktigt värde ges skall ingen avläsning initieras.
Resultattypen skall vara `void`.
Funktionen `sig_get_value` skall ge värdet -1 om avläsningen ännu inte är klar och värdet -9 om avläsningen misslyckades. Dessa två värden skall definieras som makron med namnen `BUSY` resp. `ERROR` i .h filen så att det anropande programmet kan använda dessa.

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: Assemblerdirektiv för ST32F407.

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”)

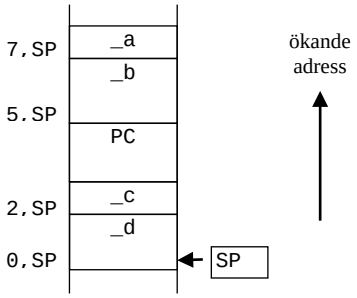
Lösningsförslag

Uppgift A-1:

a)
```
_e  RMB 2
_f  RMB 1
```

b)
```
    LDAB   _f
    PSHB
    LDD    _e
    PSHD
    JSR    _func
    LEAS   3,SP
```

c)
```
    LEAS   -3,SP
```



Uppgift A-2:

```
;   Bitmask för bitnummer 0-7
BitMask FCB     1,2,4,8,$10,$20,$40,$80

Outzero:
    CMPB    #7
    BHI Outzero_Exit
    PSHA
    PSHX
    LDX #BitMask
    LDAA    B,X
    COMA            ; Denna bit ska nollställas...
    ANDA    DCShadow
    STAA    DCShadow
    STAA    DrillControl
    PULX
    PULA
Outzero_Exit:
    RTS
```

Uppgift A-3:

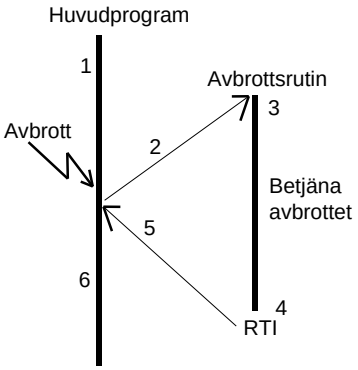
```
typedef unsigned char  *port8ptr;
#define ML4OUT_ADR1 0x400
#define ML4OUT_ADR2 0x401
#define ML4IN_ADR1 0x600
#define ML4IN_ADR2 0x601

#define ML4OUT1 *((port8ptr) ML4OUT_ADR1)
#define ML4OUT2 *((port8ptr) ML4OUT_ADR2)
#define ML4IN1 *((port8ptr) ML4IN_ADR1)
#define ML4IN2 *((port8ptr) ML4IN_ADR2)

void    DivModHex( void )
{
    unsigned char q,r,pa;
    pa = ML4IN2;
    if( pa != 0 )
    {
        q = ML4IN1/pa;
        r = ML4IN1%pa;
    }else{
        q = 0xFF;
        r = 0xFF;
    }
    ML4OUT1 = q;
    ML4OUT2 = r;
}
```

Uppgift A-4:

Ett sätt att avbryta processorns normala exekvering.

Händelseförloppet vid ett avbrott.

1. Processorn vars I-flagga är nollställd känner att IRQ är aktiverad.
2. Processorn sparar alla register på stacken. I-flaggan ettställs.
   Därefter läser processorn startadressen för avbrottsrutinen från den aktuella avbrottsvektorn som placeras i PC
3. Avbrottsrutinen startas
4. Avbrottsrutinen avslutas med instruktionen RTI dvs…
5. register innehållen återställs från stacken (Restore Status)
   Återhopp till huvudprogram med en nollställd I-flagga.
6. Därmed återstartas huvudprogrammet där det blev avbrutet.



Programrutiner: (Initieringsrutin och Avbrottsrutin)

```
* Initieringsrutin (En generell)
IrqInit
       ---    Init eventuella variabler som ingår i avbrottsrutinen
       ---       Nollställ avbrottsvippan
       LDX    #IrqRutin           Initiera avbrottsvektor
       STX    $FFF2
       CLI                        Aktivera avbrottsmekanismen
* Avbrottsrutin IrqRutin: En generell avbrottsrutin
IrqRutin
          ---       Kvitteta avbrott
          ---       Betjäna (Serva) Avbrottet
          rti
```

Uppgift B-1:

a)

```
#define    SysTickCtrl    ((volatile unsigned int *)(0xE000E010))
#define    SysTickLoad    ((volatile unsigned int *)(0xE000E014))
#define    SysTickVal     ((volatile unsigned int *)(0xE000E018))


void delay_1ms( void )
{
    /* SystemCoreClock = 168000000 */
    *SysTickCtrl = 0;
    *SysTickLoad = ( (168000) -1 );
    *SysTickVal = 0;
    *SysTickCtrl = 5;
    while( (*SysTickCtrl & 0x10000 )== 0 );
    *SysTickCtrl = 0;
}
```

b)

```
void init_app( void )
{
    /* PORT D */
    *portBargraphModer     &= ~3; /* återställ bit 0 mode */
    *portBargraphModer     |= 5;  /* bit 0 sätts som utgång */
    *portBargraphOtyper    &= ~1; /* återställ bit 0 typ, är nu push/pull */
}
```

c)

```
void main(void)
{
    int i;
    init_app();
    while(1)
    {
        *portBargraphOdrLow = 0;
        for(i=0;i<500;i++) delay_1ms();
        *portBargraphOdrLow = 0xFF;
        for(i=0;i<500;i++) delay_1ms();
    }
}
```

Uppgift B-2:

```
start:
@  initiera port D0-D7 som utport
   LDR    R0,=0x00005555
   LDR    R1,=0x40020C00
   STR    R0,[R1]
@  initiera port E0-E15 som inport
   LDR    R0,=0
   LDR    R1,=0x40021000
   STR    R0,[R1]
@ adressen till port D:s ut-dataregister till R5
   LDR    R5,=0x40020C14
@ adressen till port E:s in-dataregister till R6
   LDR    R6,=0x40021010

main:
   LDRB   R0,[R6]
   LDRB   R1,[R6,#1]
   ADD    R0,R0,R1
   CMP    R0,#255
   BLE    main_2
   MOVW   R0,#0xFF
main_2:
   STRB   R0,[R5]
   B      main
```

Uppgift B-3:

```
#define EXTI_PR         0x40013C14
#define EXTI2_IRQ_BPOS  (1<<2)
#define EXTI3_IRQVEC    0x2001C060
#define NVIC_EXTI2_IRQ_BPOS  (1<<8)

void irq_handler ( void )
{
    /* Om avbrott från EXTI2:
         kvittera avbrott från EXTI2 */
    if(   *((unsigned int *) EXTI_PR) & EXTI2_IRQ_BPOS )
    {
        *((unsigned int *) EXTI_PR) |= EXTI2_IRQ_BPOS;
    }
}

void enable_interrupt( void )
{
    /* Nollställ processorns avbrottsmask */
    __asm (
    "      cpsie  i\n"      /* set I=0 */
    );
}
```

eller:

```
void enable_interrupt ( void ) __attribute__( ( naked ) ) ;
void enable_interrupt( void )
{
    /* Nollställ processorns avbrottsmask */
    __asm (
    "      cpsie  i\n"      /* set I=0 */
    "      bx     lr\n"
    );
}

void app_init ( void )
{
    /* Koppla PE2 till avbrottslina EXTI2 */
     *((unsigned int *) SYSCFG_EXTICR1) |= 0x0400;
    /* Konfigurera EXTI2 för att generera avbrott */
     *((unsigned int *) EXTI_IMR) |= EXTI2_IRQ_BPOS;
    /* Konfigurera för avbrott på negativ flank */
     *((unsigned int *) EXTI_FTSR) |= EXTI2_IRQ_BPOS;
    /* Sätt upp avbrottsvektor */
     *((void (**)(void) ) EXTI2_IRQVEC ) = irq_handler;
    /* Konfigurera den bit i NVIC som kontrollerar den avbrottslina som EXTI2 kopplats till */
       *((unsigned int *) NVIC_ISER0) |= NVIC_EXTI3_IRQ_BPOS;
}
```

Uppgift B-4:

```
typedef struct _gpio
{
    volatile unsigned int moder;
    volatile unsigned int otyper;     // +0x4
    volatile unsigned int ospeedr;    // +0x8
    volatile unsigned int pupdr;      // +0xc
    volatile unsigned int idr;        // +0x10
    volatile unsigned int odr;        // +0x14
    volatile unsigned int bsrr;       // +0x18
    volatile unsigned int lckr;       // +0x1C
    volatile unsigned int afrl;       // +0x20
    volatile unsigned int afrh;       // +0x24
} GPIO;
```

Uppgift C-5:

```
char *strrchr(const char *s, int c)
{
    char *result;

    result = (char *) 0;
    do{
        if (*s == (char) c)
            result = s;
    } while (*s++ != 0);
    return(result);
}
```

Uppgift C-6

a) För att undvika att kompilatorn optimerar bort skrivningar/läsningar som den tror är överflödiga. Optimeraren vet inte att en viss minnesadress är en port där skrivningar/läsningar ger sidoeffekter utanför dess synfält

b) Då en .c-fil inkluderar en och samma .h-fil flera gånger - tex genom andra .h-filer.

c) `[2][1]`

d)

```
// Filen sig.h
typedef unsigned short int port;
typedef unsigned short int *portptr;

#define SIGDATA_ADR 0270000
#define SIGCTRL_ADR 0270002
#define SIGDATA *((portptr) SIGDATA_ADR)
#define SIGCTRL *((portptr) SIGCTRL_ADR)

#define start      0x0001
#define done       0x0080
#define channel    0x3f00
#define error      0x8000


// Filen sig_reader.h
#define BUSY  -1
#define ERROR -9
extern void sig_read(int channel);
extern int  sig_get_value(void);

// Filen sig_reader.c
#include "sig_reader.h"
#include "sig.h"

void sig_read(int chan) {
  port shadow = 0;
  if (chan >= 0 && chan <= 63) {
    shadow |= start;
    chan <<= 8;
    shadow |= chan;
    SIGCTRL = shadow;
  }
}

int  sig_get_value(void) {
  if (SIGCTRL & error)
    return ERROR;
  else if (SIGCTRL & done)
    return SIGDATA;
  else
    return BUSY;
}
```