Synkronisering (sammanfattning)

Ur innehållet:

- Arbetstakter - jämförelser
- Repetition: synkrona gränssnitt och tidsdiagram
- "Verklig tid" - räknarkrets "SysTick"
- Programmering av LCD-ASCII displaymodul

Läsanvisningar:

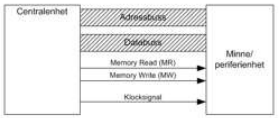
- Arbetsbok kapitel 5, t.o.m 5.3
- Datablad "ASCII-display"

Målsättningar:

- Kunna implementera och testa laborationsuppgift 2

Ovillkorlig överföring

Kräver samtidighet "synkron" - en speciell signal, "Enable", används då för att ange exakt när datautbyte ska ske.



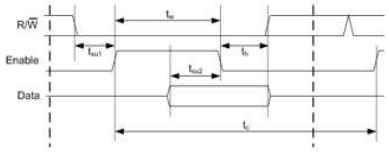
Flankerna definierar samtidigheten. Signalen kallas ofta "klocksignal".

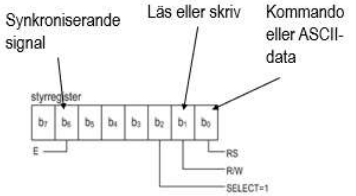
Härledning av algoritm

Byte som ska överföras

dataregister

DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0





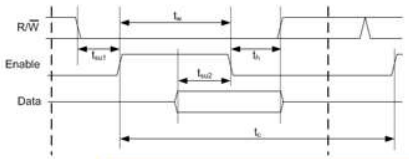
1

Skriv 8 bitar till controller:

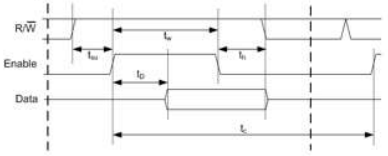
styrregister(RW)=0;
Vänta t_{su1};
styrregister(E)=1;
dataregister = (8 bitar);
Vänta t_{su2}; (dock tills minst t_w förflutit)
E = 0;
vänta tills totalt t_c förflutit;

Tidsdiagram - underlag för synkronisering

Skrivcykel - data överförs från CPU till periferienhet

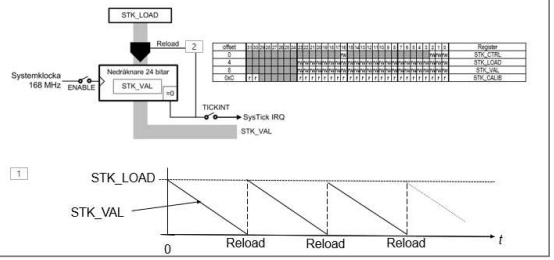


Läscykel - data överförs från periferienhet till CPU

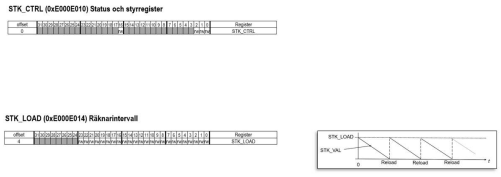


		min
t_c	cykel tid	min
t_w	klockpuls ("Enable") varaktighet (hög och låg)	min
t_{su1}	styrsignalernas setup-tid, före positiv E-flank	min
t_{su2}	setup-tid för data, skrivning, före negativ E-flank	min
t_D	setup-tid för data, läsning, före negativ E-flank	max
t_h	hold-tid, varaktighet (efter negativ E-flank)	min

Räknarkrets - "SysTick"



Räknarkrets - register



void delay_mikro(int us)

```
#define     STK_CTRL    ((volatile unsigned int *)(0xE000E010))
#define     STK_LOAD    ((volatile unsigned int *)(0xE000E014))
#define     STK_VAL     ((volatile unsigned int *)(0xE000E018))

void    delay_micro(unsigned int us)
{
#ifdef  SIMULATOR
  us = us / 1000;
  us++;
#endif
  while( us > 0 )
  {
        delay_250ns();
        delay_250ns();
        delay_250ns();
        delay_250ns();
        us--;
  }
}
```

void delay_250ns(void)

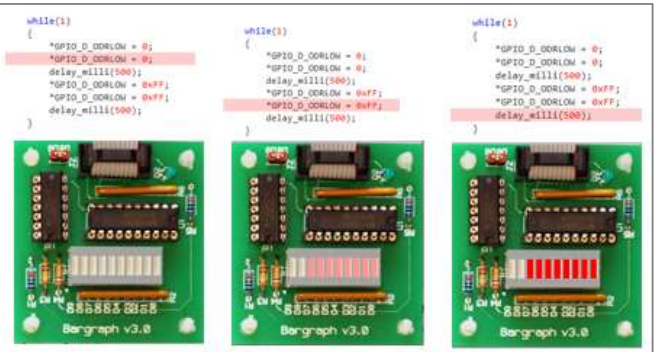
```
#define     STK_CTRL    ((volatile unsigned int *)(0xE000E010))
#define     STK_LOAD    ((volatile unsigned int *)(0xE000E014))
#define     STK_VAL     ((volatile unsigned int *)(0xE000E018))

void delay_250ns( void )
{
  /* SystemCoreClock = 168000000 */
  *STK_CTRL = 0;
  *STK_LOAD = ( (168/4) -1 );
  *STK_VAL = 0;
  *STK_CTRL = 5;
  while( (*STK_CTRL & 0x10000 )== 0 );
  *STK_CTRL = 0;
}
```

Periodiskt blinkande diodramp

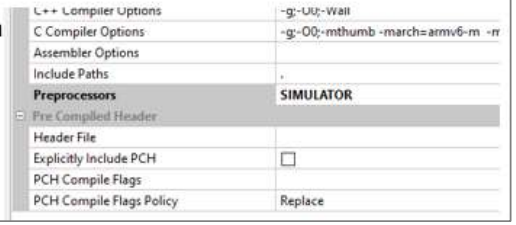
Vi har en diodramp ansluten till port D (0-7) och demonstrerar en applikation som får hela rampen att blinka 1 gång/sekund.

```
void main(void)
{
  init_app();
  while(1)
  {
    *GPIO_D_ODRLOW = 0;
    *GPIO_D_ODRLOW = 0;
    delay_milli(500);
    *GPIO_D_ODRLOW = 0xFF;
    *GPIO_D_ODRLOW = 0xFF;
    delay_milli(500);
  }
}
```



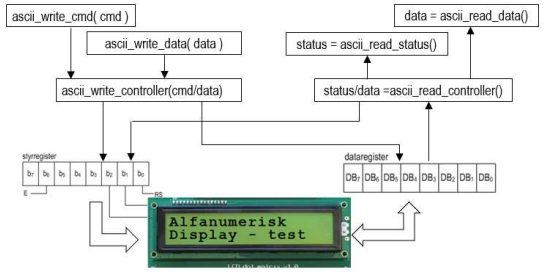
Program har helt olika tidsegenskaper då dom utförs i simulator respektive hårdvara
Använd villkorlig kompilering för att anpassa fördröjningen i exemplet för simulator

```
void       delay_milli(unsigned int ms)
{
#ifdef     SIMULATOR
  ms = ms / 1000;
  ms++;
#endif
   ...
```



Programmering av ASCII-Display

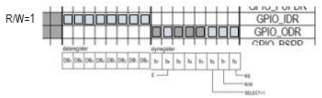
Programstruktur



Anslutning av ASCII-display

Vi konfigurerar två 8-bitars portar i GPIO E.

LCD-modul CTRL PE0-7 ansluts till ASCII-styrkretsens styrregister, styrregistret är endast skrivbart



LCD-modul DATA PE8-15 är en utport för ***data till*** ASCII-styrkretsen

men vänds till en inport då vi läser ***data från*** ASCII-styrkretsen

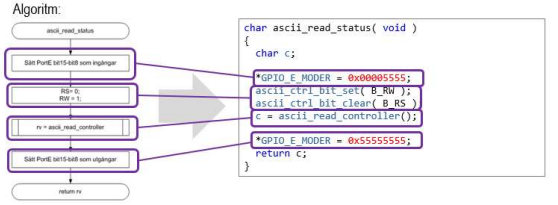
I dataregistret skriver/läser vi alltid 8 bitar åt gången.

I styrregistret ändrar vi enstaka bitar åt gången

status = ascii_read_status()

För att läsa från styrkretsen måste vi "vända" riktningen hos port E15-E8 ("dataregister")

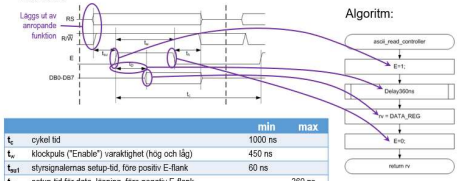
Algoritm:



Läscykel - karakteristika

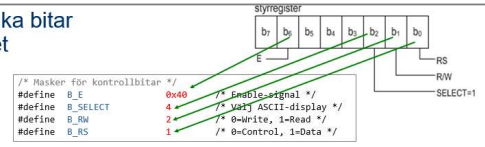
Databladets tidsdiagram illustrerar hur styrsignaler ska läggas ut och hur displaymodulen svarar med data.

Algoritm:



		min	max
t_c	cykel tid	1000 ns	
t_w	klockpuls ("Enable") varaktighet (hög och låg)	450 ns	
t_{su1}	styrsignalernas setup-tid, före positiv E-flank	60 ns	
t_D	setup-tid för data, läsning, före negativ E-flank		360 ns
t_h	hold-tid, varaktighet (efter negativ E-flank)	10 ns	

Ändra enstaka bitar i styrregistret




```
/* Masker för kontrollbitar */
#define  B_E        0x40    /* Enable-signal */
#define  B_SELECT   4       /* Välj ASCII-display */
#define  B_RW       2       /* 0=Write, 1=Read */
#define  B_RS       1       /* 0=Control, 1=Data */
```

```
void ascii_ctrl_bit_set( char x )
{ /* x: bitmask bits are 1 to set */
  char c;
  c = *GPIO_E_ODRLOW;
  *GPIO_E_ODRLOW = B_SELECT | x | c;
}
```

```
void ascii_ctrl_bit_clear( char x )
{ /* x: bitmask bits are 1 to clear */
  char c;
  c = *GPIO_E_ODRLOW;
  c = c & ~x;
  *GPIO_E_ODRLOW = B_SELECT | c;
}
```

Alternativt, använd "macro"-definitioner

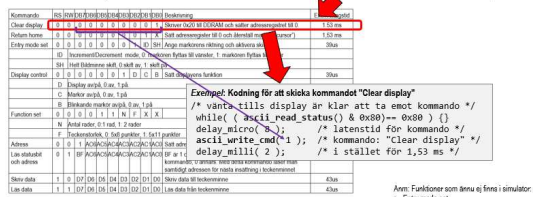
```
#define ASCII_CTRL_BIT_SET(x)  \
        *GPIO_E_ODRLOW = \
        (B_SELECT | x | *GPIO_E_ODRLOW)

#define ASCII_CTRL_BIT_CLEAR(x) \
        *GPIO_E_ODRLOW = \
        (B_SELECT | ( *GPIO_E_ODRLOW & ~(x) ))
```

Ge kommando till styrkrets

Formulering av algoritm för att överföra ett specifikt kommando till styrkretsen

Algoritm: **ascii_command(command)**
Vänta tills statusflaggan är 0
Fördröj 8 us
ascii_write_cmd(command)
Fördröj kommandospecifik fördröjning



Exempel: **Kodning för att skicka kommandot "Clear display"**

```
/* vänta tills display är klar att ta emot kommando */
while( ( ascii_read_status() & 0x80)== 0x80 ) {}
delay_micro( 8 );      /* latenstid för kommando */
ascii_write_cmd( 1 );  /* kommando: "Clear display" */
delay_milli( 2 );      /* i stället för 1,53 ms */
```

Anm: Funktioner som ännu ej finns i simulator:
- Entry mode set
- Display on/off
- Function set (endast 5x8, 2 rader)

ascii_write_controller(command)

Implementering av algoritmen RS och RW är klara

Algoritm: