

Synkronisering

Ur innehållet:

- Arbetsstakter - jämförelser

- Repetition: synkrona gränssnitt och tidsdiagram

- "Verklig tid" - räknarkrets "SysTick"

- Programmering av LCD-ASCII displaymodul

Läsanvisningar:

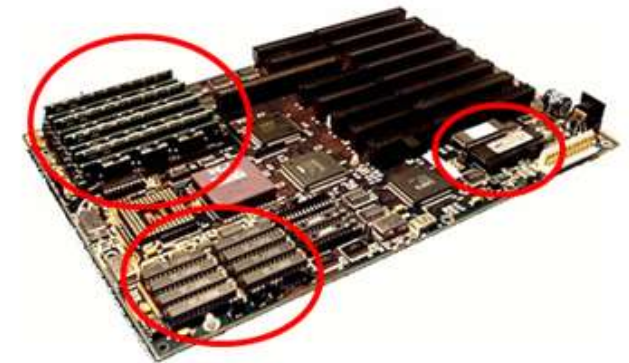
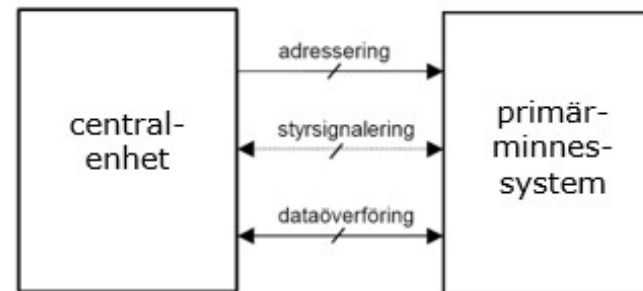
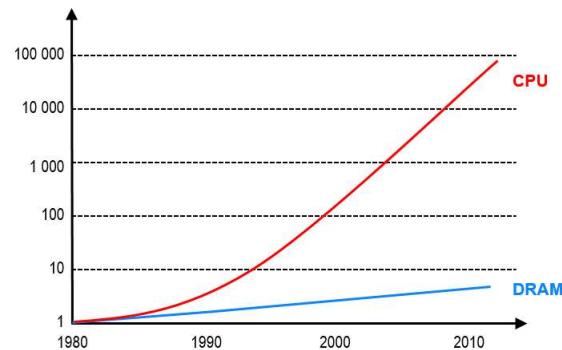
- Arbetsbok kapitel 5, t.o.m 5.3

- Datablad "ASCII-display"

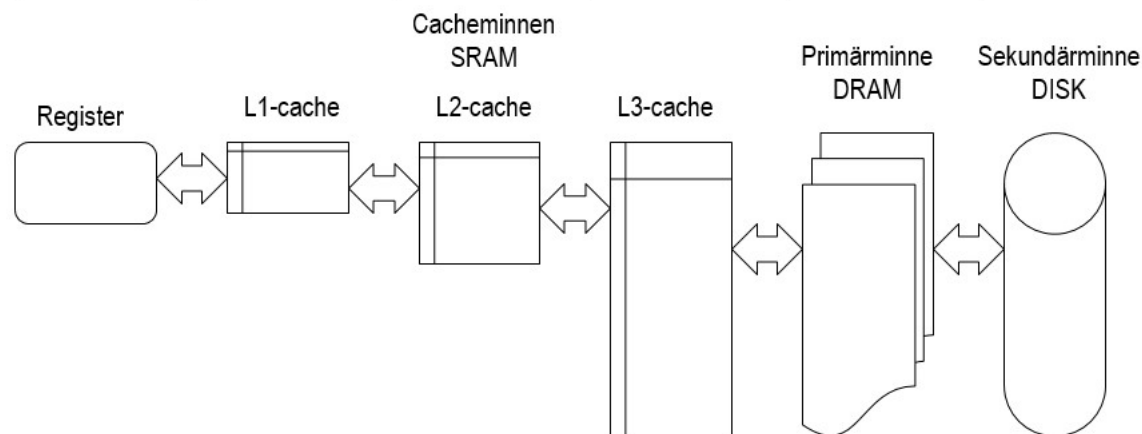
Målsättningar:

- Kunna implementera och testa laborationsuppgifter 2.2 och 2.3

Arbetstakt - typiska åtkomsttider



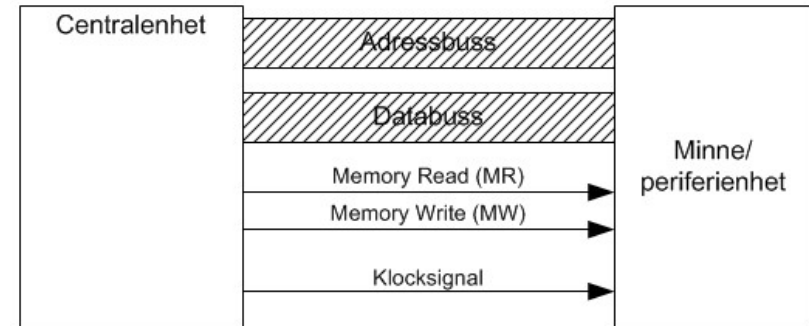
Storlek	1 KB	64 KB	256 KB	2-4 MB	4-16GB	4-16 TB
Åtkomsttid	300 ps	1 ns	3-10 ns	10-20 ns	50-100 ns	5-10 ms



Kraftigt varierande prestanda resulterar i stora skillnader i åtkomsttid

Ovillkorlig överföring

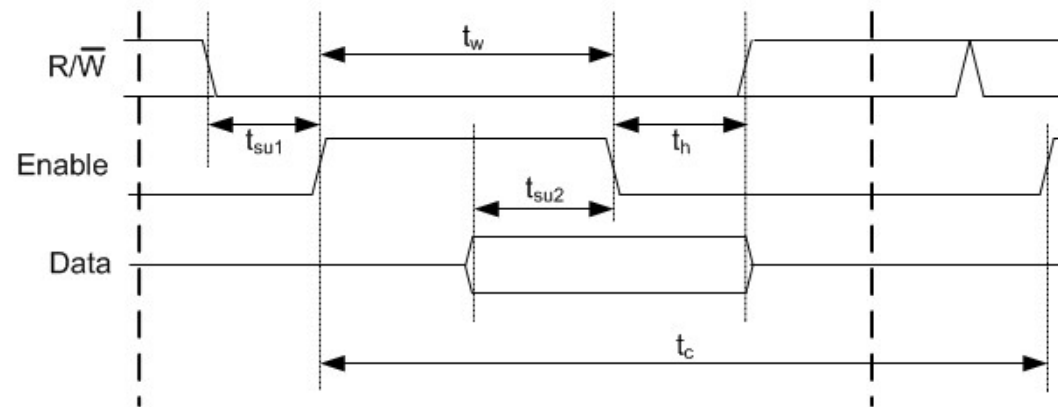
Kräver samtidighet "synkron" - en speciell signal, "Enable", används då för att ange exakt när datautbyte ska ske.



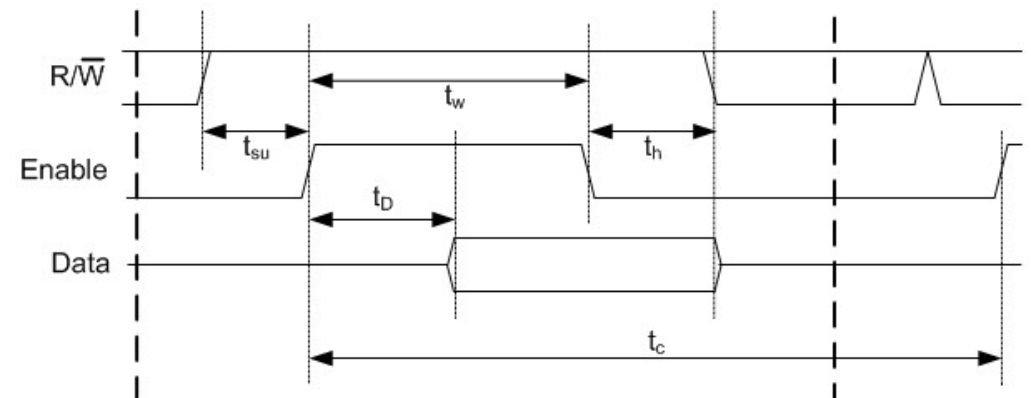
Flankerna definierar samtidigheten. Signalen kallas ofta "klocksignal".

Tidsdiagram - underlag för synkronisering

Skrivcykel - data överförs från CPU till periferienhet



Läscykel - data överförs från periferienhet till CPU

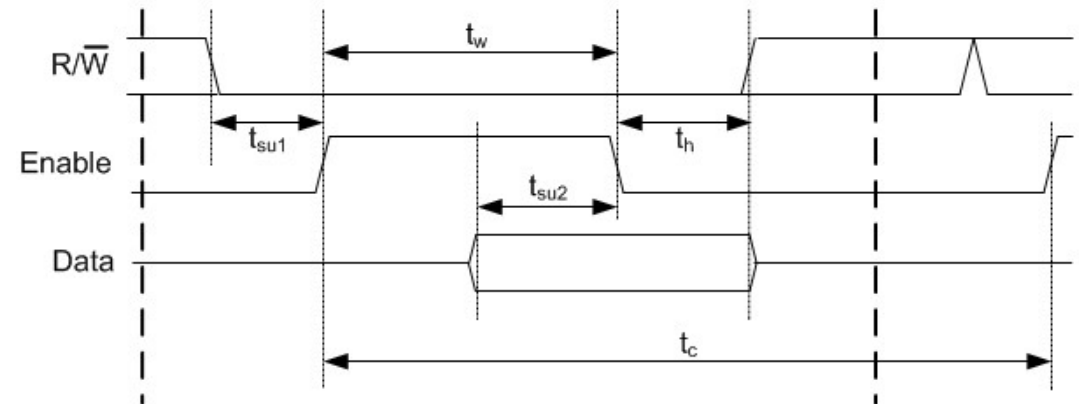
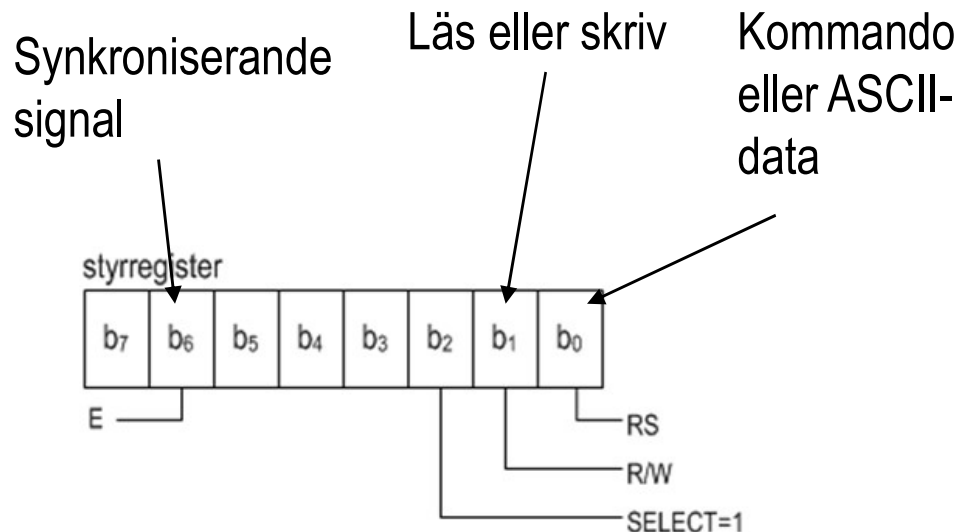
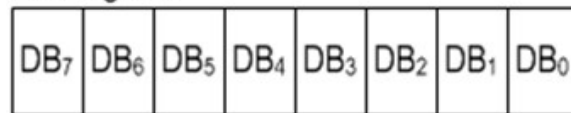


		min
t_c	cykel tid	min
t_w	klockpuls ("Enable") varaktighet (hög och låg)	min
t_{su1}	styrsignalernas setup-tid, före positiv E-flank	min
t_{su2}	setup-tid för data, skrivning, före negativ E-flank	min
t_D	setup-tid för data, läsning, före negativ E-flank	max
t_h	hold-tid, varaktighet (efter negativ E-flank)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

Härledning av algoritm

Byte som ska överföras

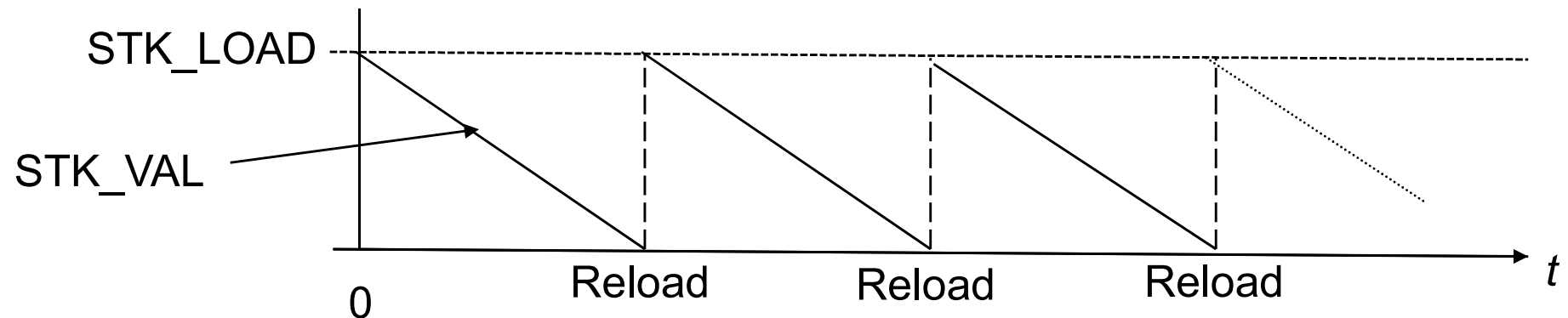
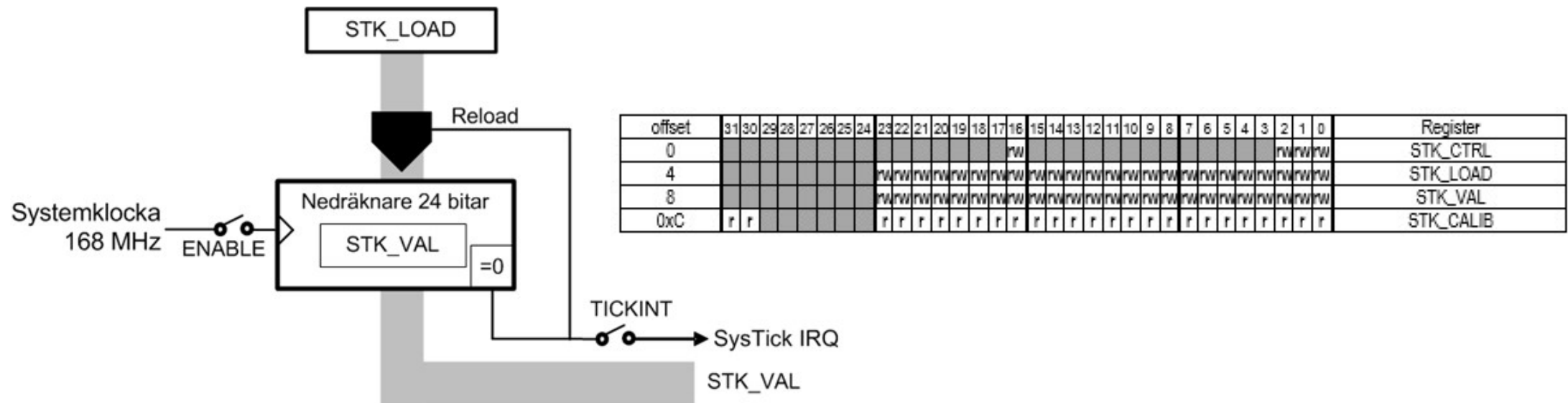
dataregister



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;
```

Räknarkrets - "SysTick"

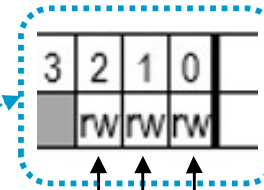


Räknarkrets - register

STK_CTRL (0xE000E010) Status och styrregister

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																rw														rw	rw	rw	STK_CTRL

Bit 16: (COUNTFLAG):
Biten är 1 om räknaren räknat ned till 0.
Biten nollställs då registret läses.



Bit 0: (ENABLE) Aktivera räknare
0: Räknare deaktiverad
1: Räknare aktiverad

Bit 1: (TICKINT): Aktivera avbrott
0: Inget avbrott genereras.
1: Då räknaren slagit om till 0 genererar SysTick avbrott.

Bit 2: (CLKSOURCE) biten är 1 efter RESET:
0: Systemklocka/8 (168Mhz/8 = 21MHz)
1: Systemklocka (168MHz)

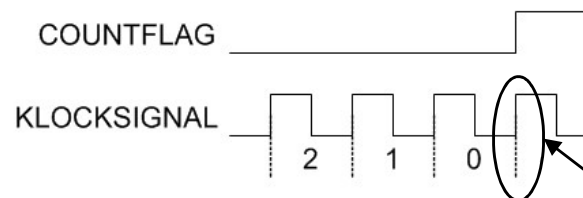
STK_LOAD (0xE000E014) Räknarintervall

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
4									rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	STK_LOAD

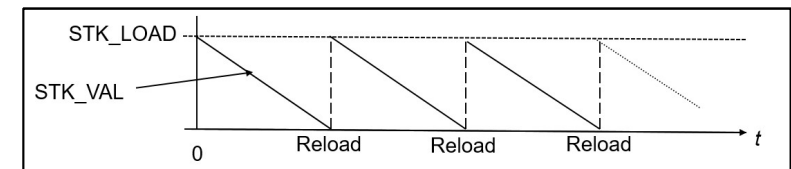
Exempel: med 168MHz

$$\text{Periodtid} = \frac{1}{168 \times 10^{-6}} \text{ s} \approx 6\text{ns}$$

För $\frac{168}{168 \times 10^{-6}}$ får vi räknarintervallet 1µs



Omslaget till 0 sker i slutet av perioden vilket gör att den korrekta initieringen för 1µs fördröjning blir **168-1** klockcykler



Fördröjning med "SysTick"

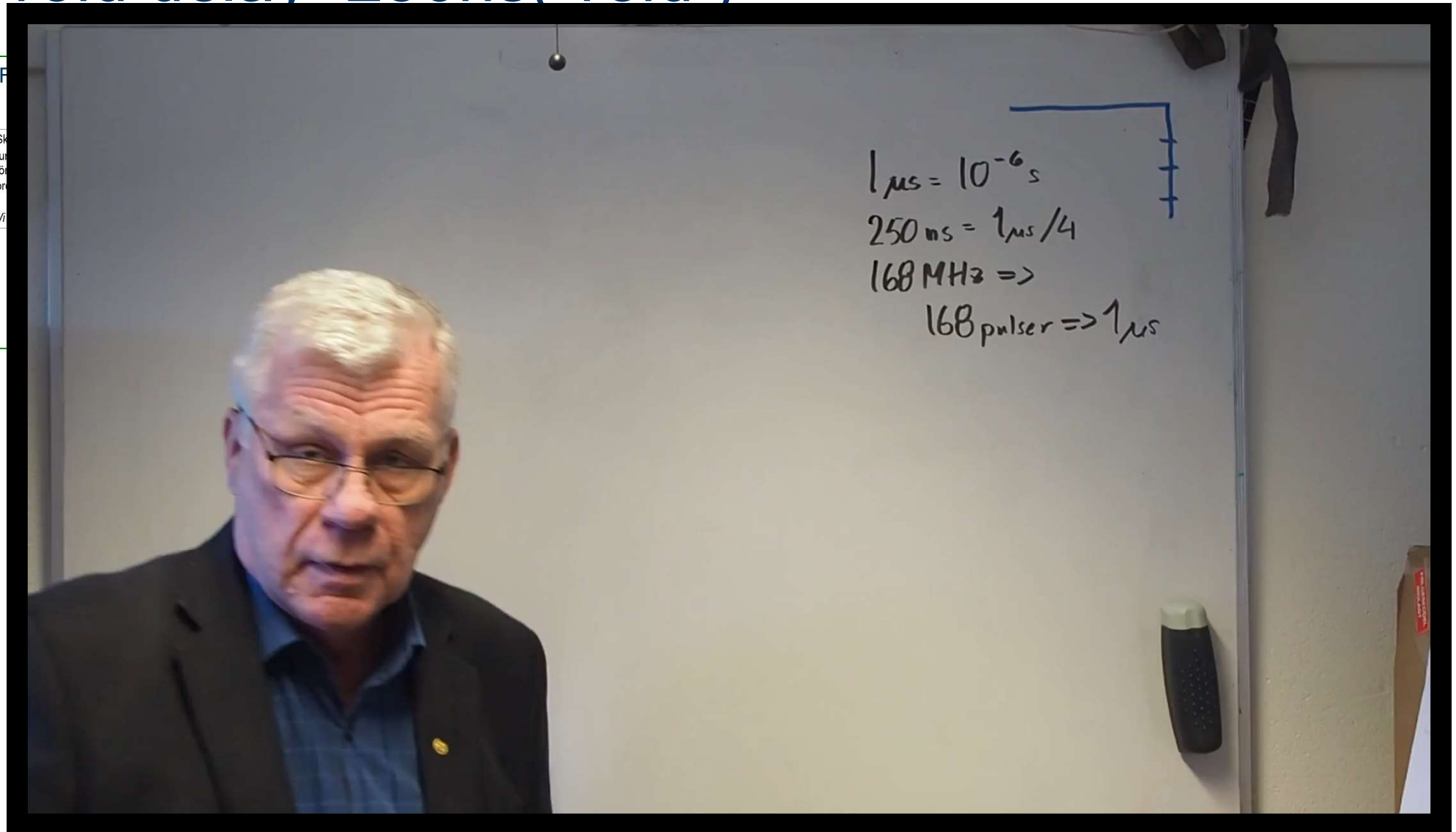
Skapa en funktion `delay_250ns(void)` som blockerar (fördröjer) den anropande funktionen med minst 250 ns. Visa också hur denna kan användas för att skapa en fördröjningsrutin `delay_mikro( unsigned int us)` som fördröjer programexekveringen variabelt antal mikrosekunder.

Vi löser på tavlan...

Algoritm:

```
STK_CTRL = 0      Återställ SysTick
STK_LOAD = CountValue
STK_VAL = 0        Nollställ räknarregistret
STK_CTRL = 5      Starta om räknaren
Vänta till COUNTFLAG=1
STK_CTRL = 0      Återställ SysTick
```

void delay 250ns(void)



```
void delay_mikro( int us )
```



Periodiskt blinkande diodramp

Laborationsuppgift 2.2



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

se Demonstration: Laborationsuppgift 2.2 under hemsidans "Laboration 2".

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

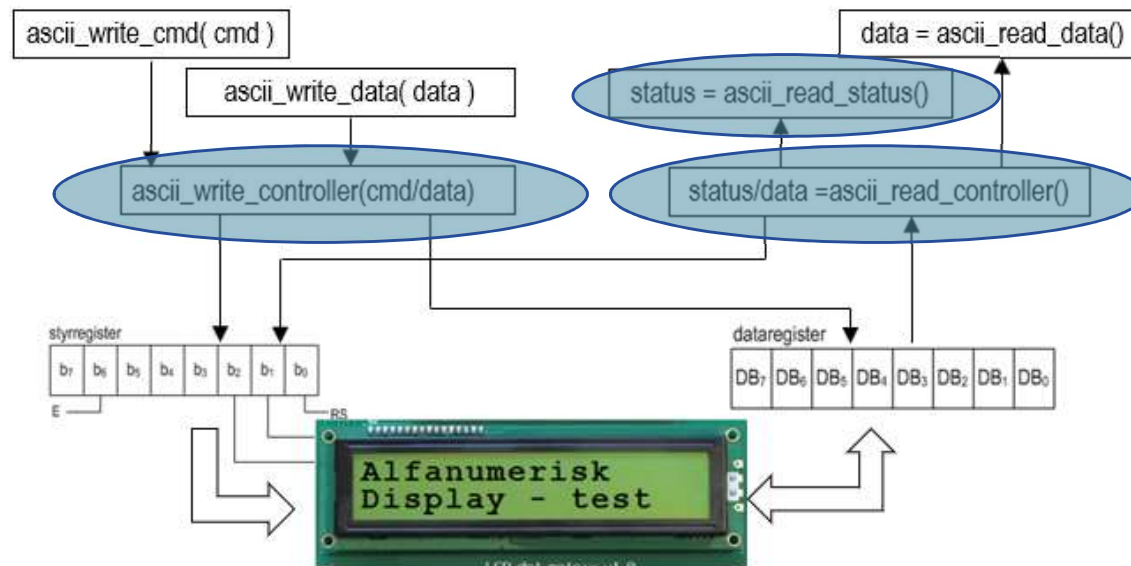
Programmering av ASCII-Display

Laborationsuppgift 2.3

Vi implementerar funktioner:

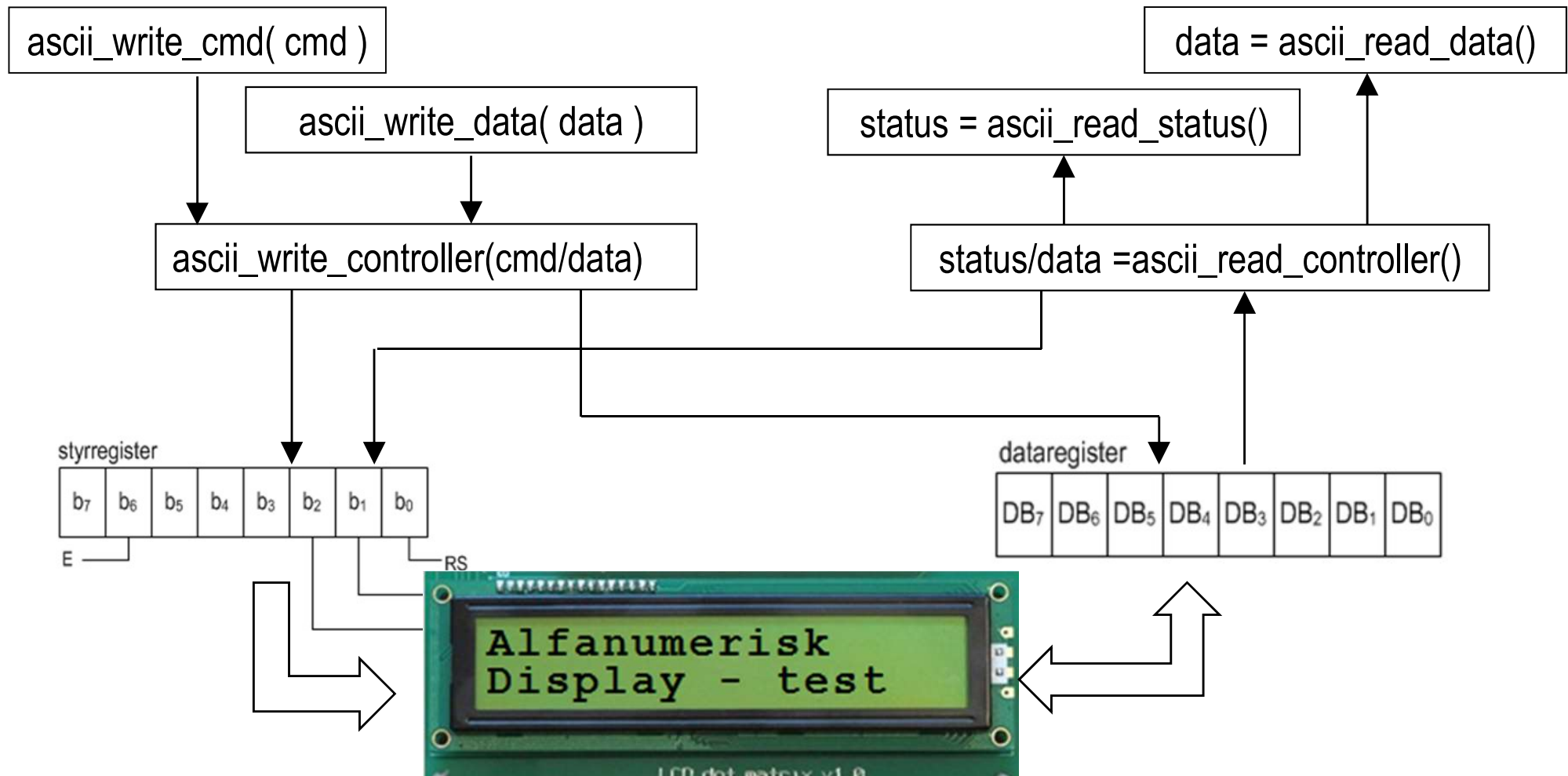
```
ascii_write_controller(cmd/data)
status/data=ascii_read_controller()
status=ascii_read_status()
```

Programstruktur



*resten av programpaketet,
implementering, integration och test
är del av laborationsprojektet...*

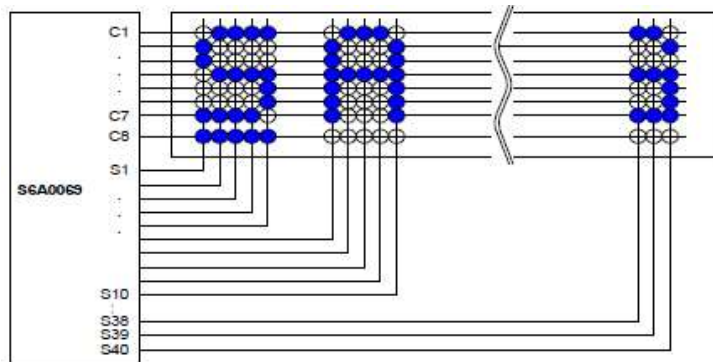
Programstruktur



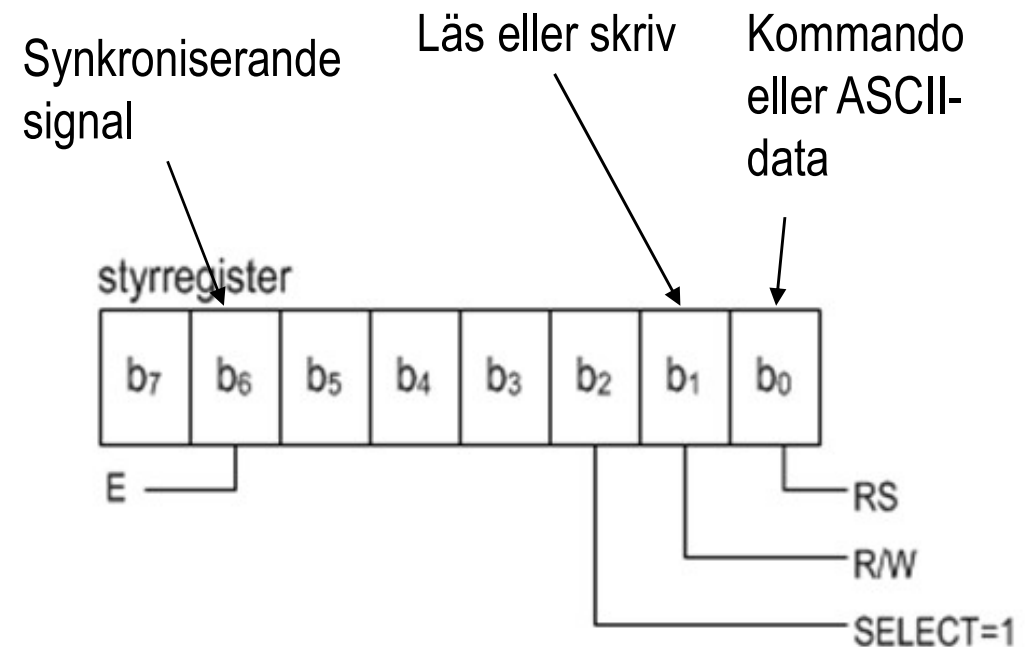
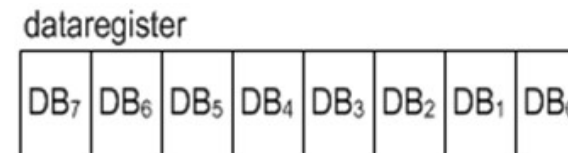
Programmering av ASCII-display

Kretsen översätter ASCII-tecken till motsvarande bit-mönster via ett enkelt gränssnitt:

- styrregister - 4 bitar används
- dataregister - 8 bitar används



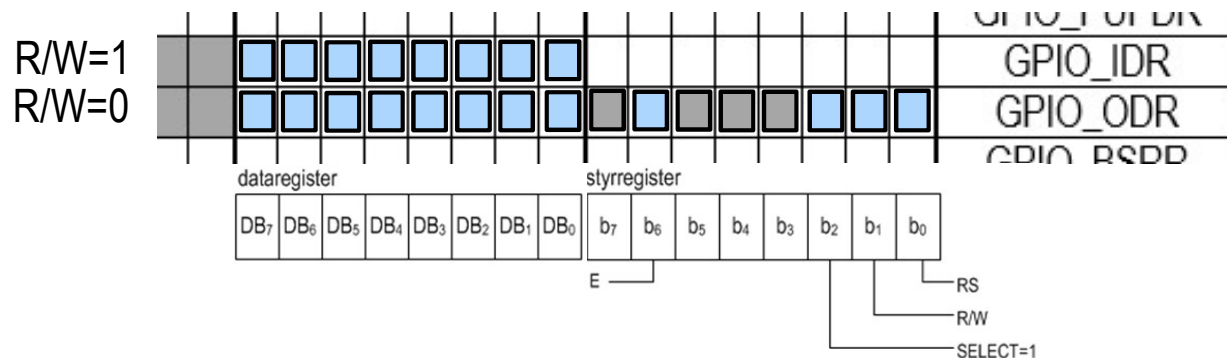
Byte som ska överföras



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



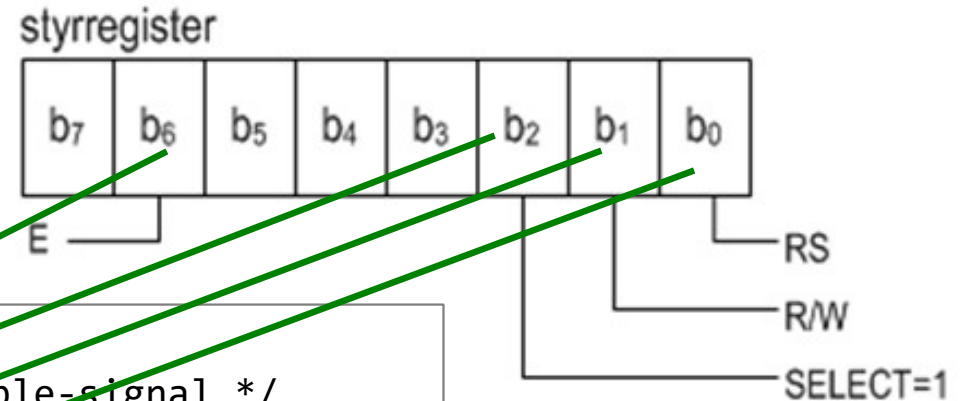
LCD-modul DATA PE8-15 är en utport för **data till** ASCII-styrkretsen men vänds till en inport då vi läser **data från** ASCII-styrkretsen



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

I styrregistret ändrar vi enstaka bitar åt gången

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

Kommando	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Beskrivning	Exekveringstid
Clear display	0	0	0	0	0	0	0	0	0	1	Skriver 0x20 till DDRAM och sätter adressregistret till 0.	1.53 ms
Return home	0	0	0	0	0	0	0	0	1	X	Sätt adressregister till 0 och återställ markör ("cursor")	1,53 ms
Entry mode set	0	0	0	0	0	0	0	1	ID	SH	Ange markörens riktning och aktivera skift	39us
	ID	Increment/Decrement mode, 0: markören flyttas till vänster, 1: markören flyttas till höger										
	SH	Helt Bildminne skift, 0:skift av, 1: skift på										
Display control	0	0	0	0	0	0	1	D	C	B	Sätt displayens funktion	39us
	D	Display av/på, 0:av, 1:på.										
	C	Markör av/på, 0:av, 1:på										
	B	Blinkande markör av/på, 0:av, 1:på										
Function set	0	0	0	0	1	1	N	F	X	X		
	N	Antal rader, 0:1 rad, 1: 2 rader										
	F	Teckenstorlek, 0: 5x8 punkter, 1: 5x11 punkter										
Adress	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Sätt adress	
Läs statusbit och adress	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	BF är 1 om kommando, 0 annars. Med detta kommando läser man samtidigt adressen för nästa insättning i teckenminnet	
Skriv data	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Skriv data till teckenminne	43us
Läs data	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Läs data från teckenminne	43us

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 */
```

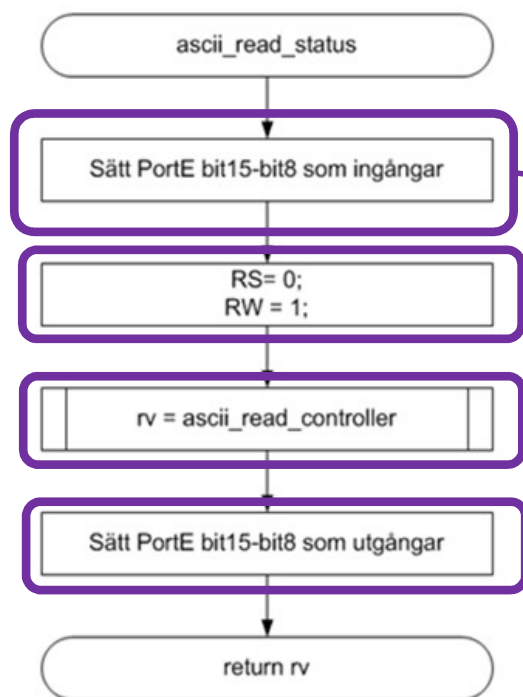
Anm: Funktioner som ännu ej finns i simulator:

- Entry mode set
- Display on/off
- Function set (endast 5x8, 2 rader)

status = ascii_read_status()

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

Algorithm:



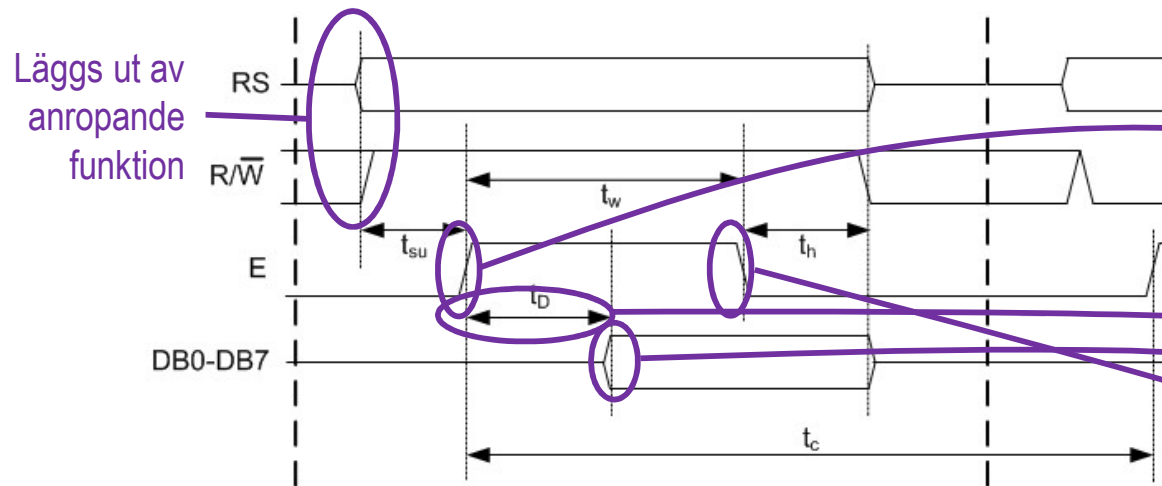
```
char ascii_read_status( void )
{
    char c;

    *GPIO_E_MODER = 0x00005555;
    ascii_ctrl_bit_set( B_RW );
    ascii_ctrl_bit_clear( B_RS );
    c = ascii_read_controller();

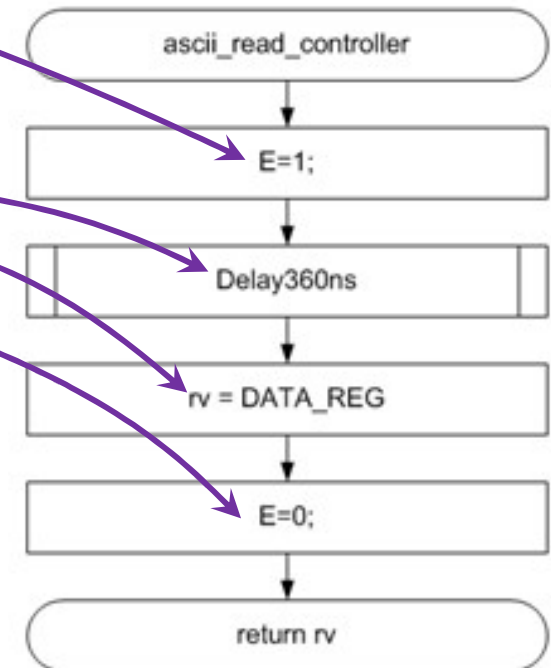
    *GPIO_E_MODER = 0x55555555;
    return c;
}
```

Läscopykel - karakteristika

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



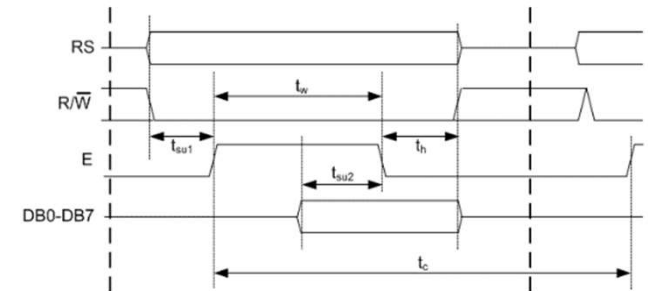
Algoritm:



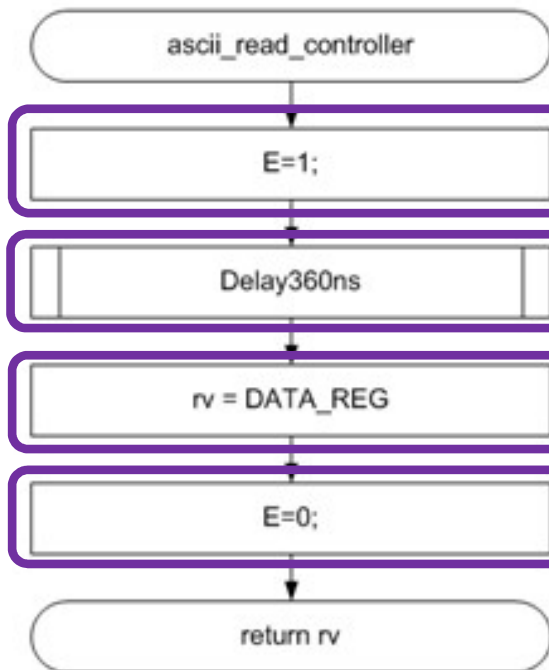
		min	max
t_c	cykel tid	1000 ns	
t_w	klockpuls ("Enable") varaktighet (hög och låg)	450 ns	
t_{su1}	styrsignalernas setup-tid, före positiv E-flank	60 ns	
t_D	setup-tid för data, läsning, före negativ E-flank		360 ns
t_h	hold-tid, varaktighet (efter negativ E-flank)	10 ns	

ascii_read_controller(command)

RS och RW har satts korrekt av anropande funktion



Algorithm:

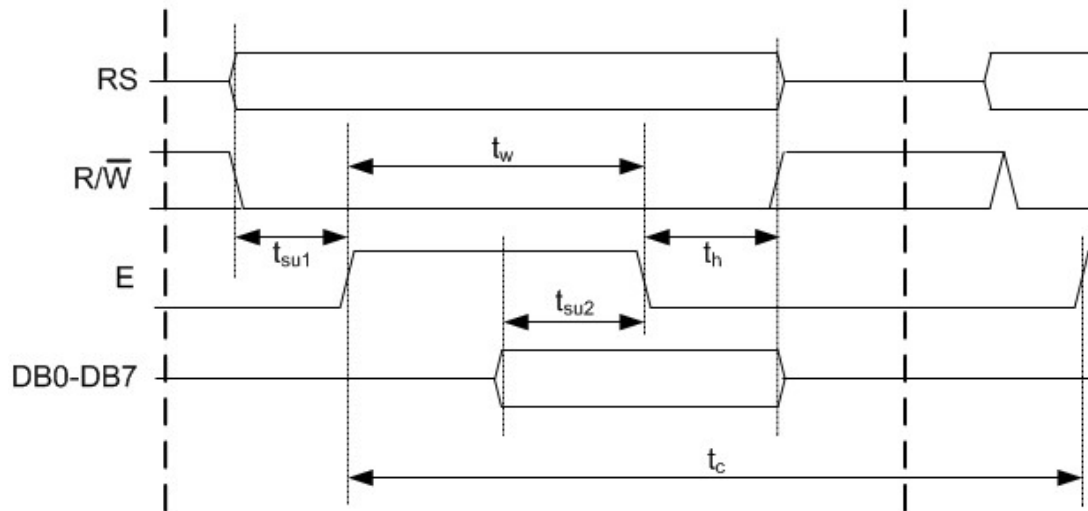


```

char ascii_read_controller( void )
{
    char c;
    ascii_ctrl_bit_set( B_E );
    delay_250ns();
    delay_250ns();
    c = *GPIO_E_IDRHIGH;
    ascii_ctrl_bit_clear( B_E );
    return c;
}
  
```

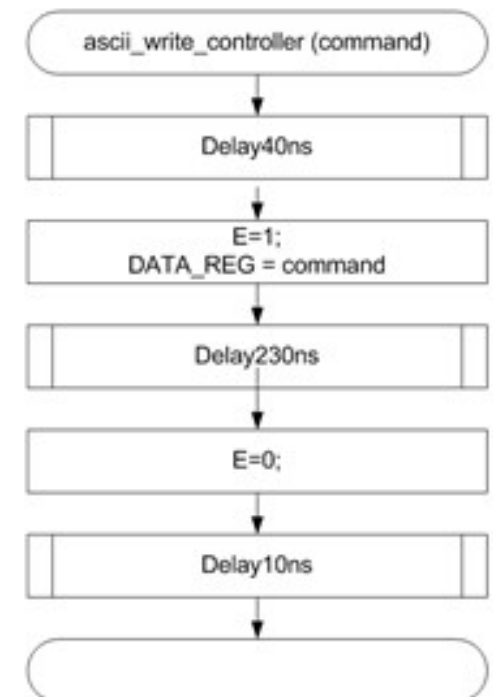
Skrivcykel - karakteristika

Databladets tidsdiagram illustrerar hur styrsignaler och data ska läggas ut

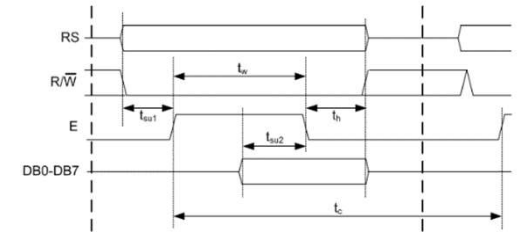


		min
t_c	cykel tid	500 ns
t_w	klockpuls ("Enable") varaktighet (hög och låg)	230 ns
t_{su1}	styrsignalernas setup-tid, före positiv E-flank	40 ns
t_{su2}	setup-tid för data, skrivning, före negativ E-flank	80 ns
t_h	hold-tid, varaktighet (efter negativ E-flank)	10 ns

Algoritm:

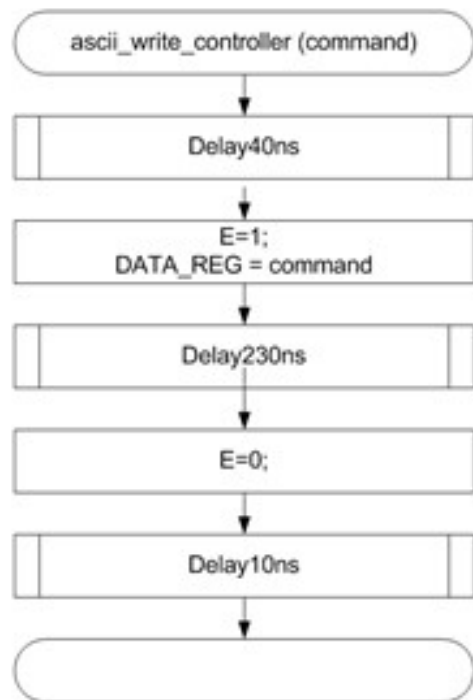


ascii_write_controller(command)



Implementering av algoritmen RS och RW är klara

Algorithm:



```

void ascii_write_controller( char c )
{
    ascii_ctrl_bit_set( B_E );
    *GPIO_E_ODRHIGH = c;
    ascii_ctrl_bit_clear( B_E );
    delay_250ns();
}
  
```