

Machine-Oriented Programming

Timing & Synchronization

Pedro Trancoso

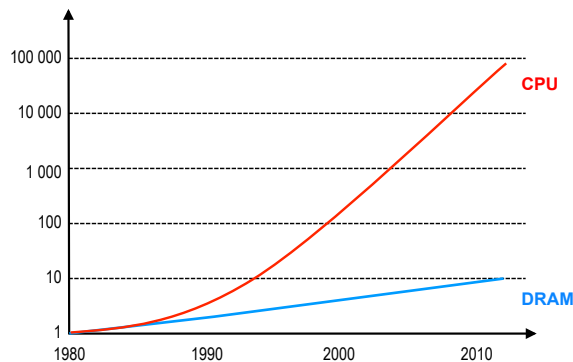
ppedro@chalmers.se

Original slides by Ulf Assarsson

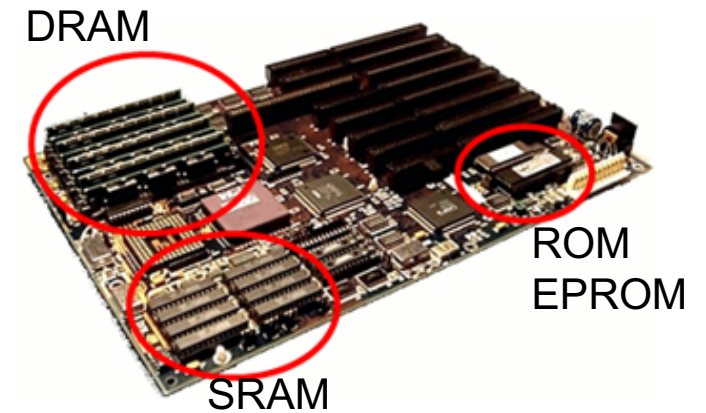
Objectives

- Topics:
 - Access Time
 - Synchronous/Asynchronous Interface
 - Time diagram
 - System Clock "SysTick"
 - Examples
- Reading:
 - "Arbetsbok" chapter 5
 - Datasheet "ASCII-display"

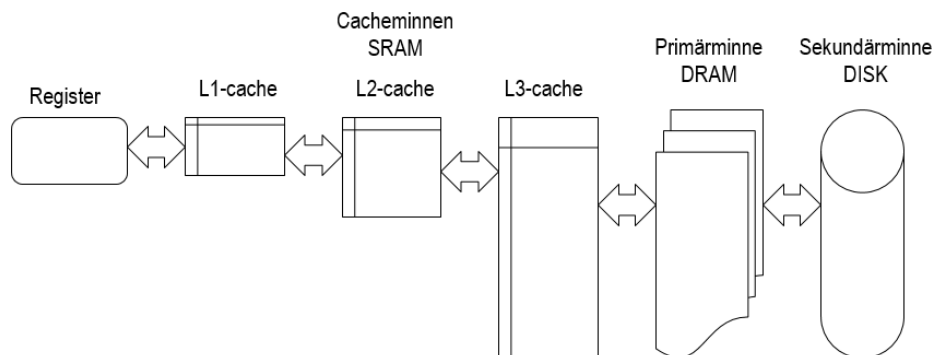
Access Time for Different Units



Performance gap CPU/DRAM



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



Intel i7-6700:

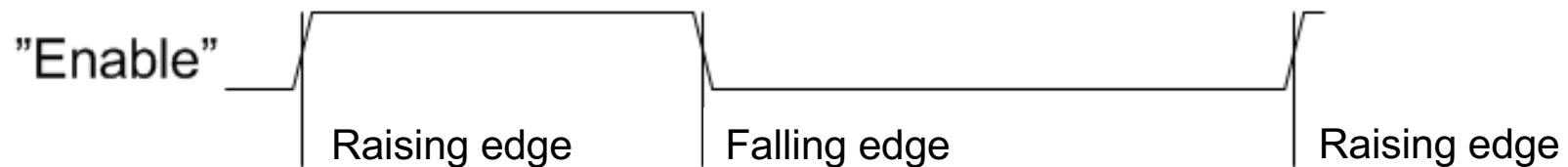
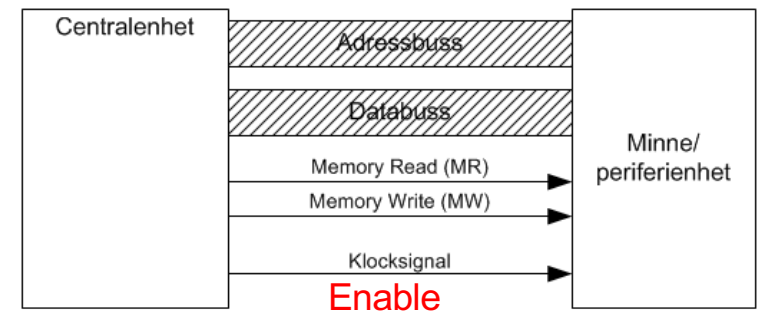
- L1 D\$ 32KB + I\$ 32KB (4 cycles)
- L2 256KB (12 cycles)
- L3 8MB (4 core) (42 cycles)
- RAM (93 cycles)

Highly varying performance results in major differences in access time

Synchronous Transfer

Requires synchronization

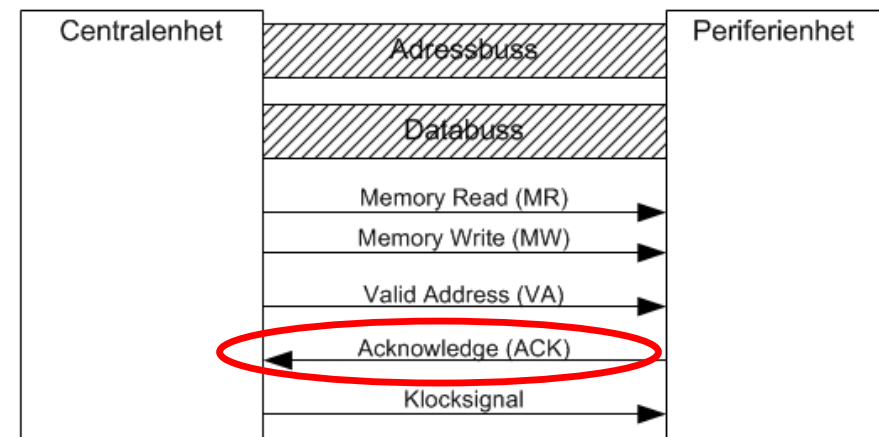
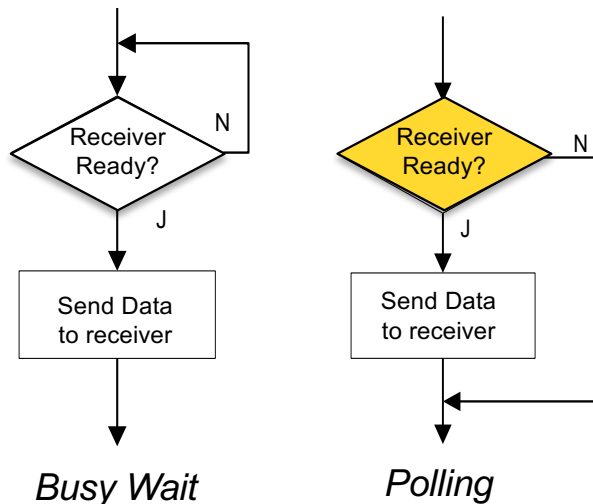
- A special signal, "Enable", has to be used to specify exactly when data exchange is to take place.



The rising/falling edges define the synchronization. The signal is often called "clock signal"

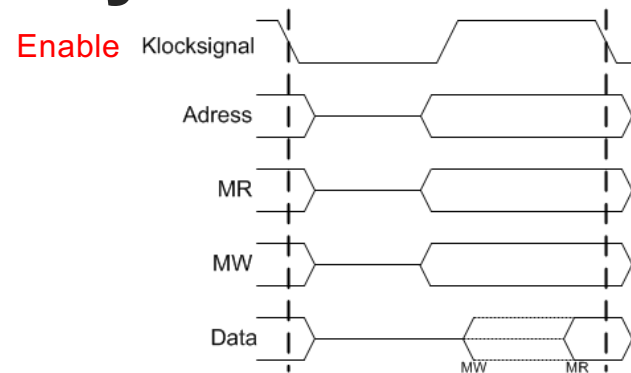
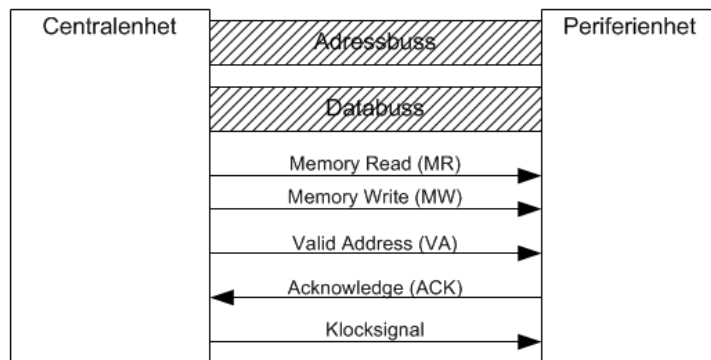
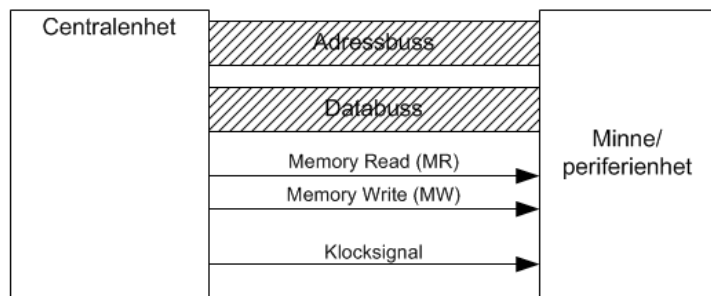
Asynchronous Transfer

The peripheral unit has an interface (register) with a status bit indicating whether or not data is available.



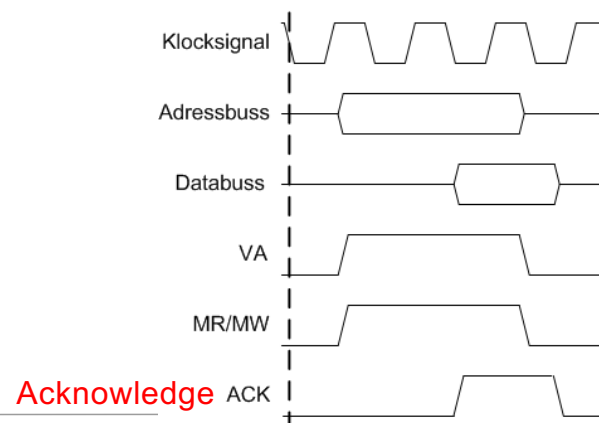
Does not require synchronization but requires special "handshake signals". Therefore, such an interface we call "asynchronous".

Synchronous and Asynchronous Interfaces



Synchronous:

A clock signal from the central unit determines when data exchange occurs

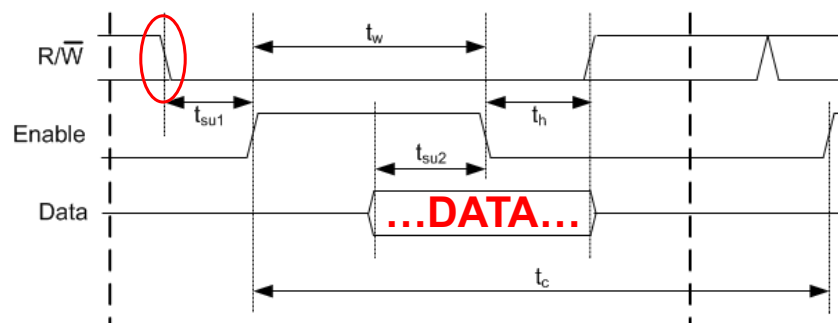


Asynchronous:

Handshake signals determine when data exchange can occur

Time Diagram – Basis for Synchronization

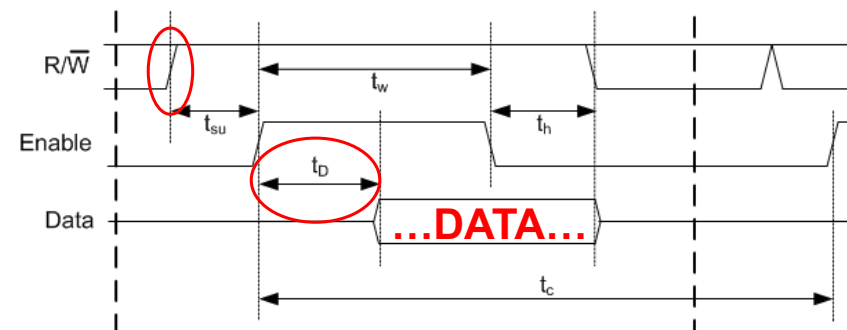
Write cycle: data is transferred from the CPU to peripheral device



Write cycle:

1. Set the R/W-signal to 0 (a bit in a specific port)
2. Wait at least t_{su1} ns.
3. Set the enable-bit and wait at least t_w ns.
4. At least t_{su2} ns before resetting the enable-bit, write the data to the data bus (a port)
5. Reset the enable-bit.
6. Wait at least t_h ns until starting again (and set R/W). Wait for at least a total of t_c ns (cycle time).

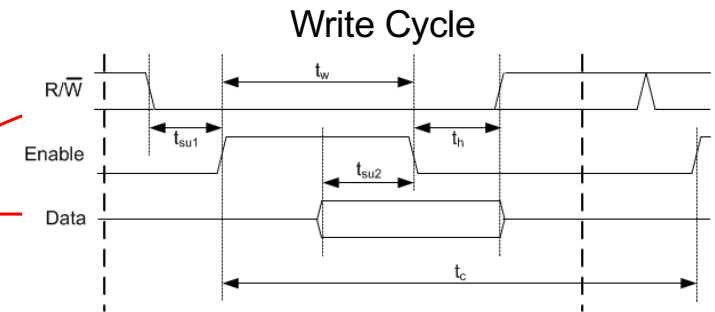
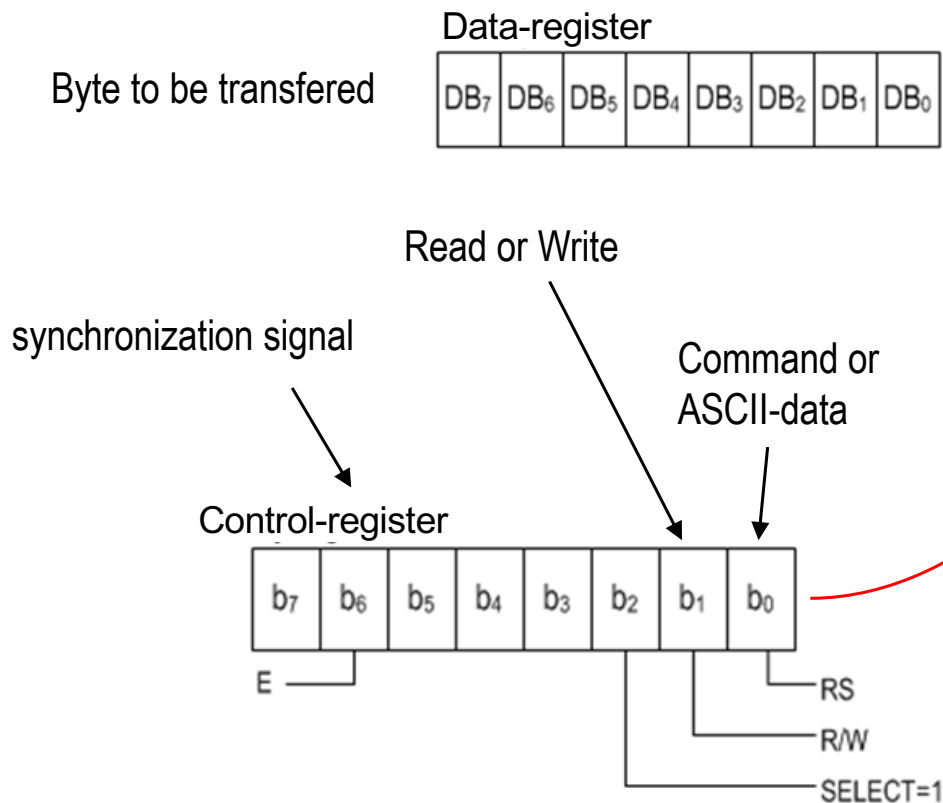
Read cycle: data is transferred from peripheral device to CPU



Read cycle:

1. Set the R/W-bit to 1.
2. Wait at least t_{su} ns.
3. Set the enable-bit to 1
4. Wait at least t_D ns.
5. Read from the data bus.
6. Wait at least t_w have passed.
7. Set enable-bit to 0.
8. Wait at least t_h ns.
9. Modify R/W if you want, but wait for a complete cycle time of t_c .

Examples: Interface/Algorithms



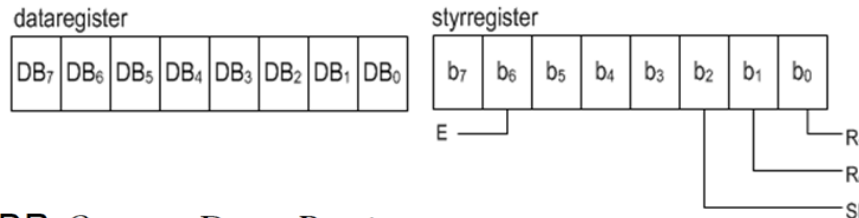
Transfer 8 bits to the ascii controller:

```

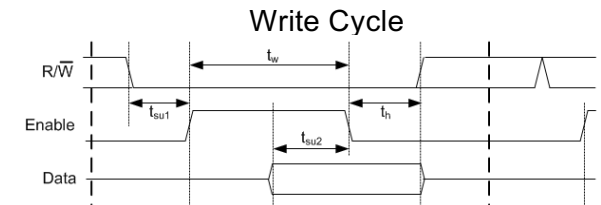
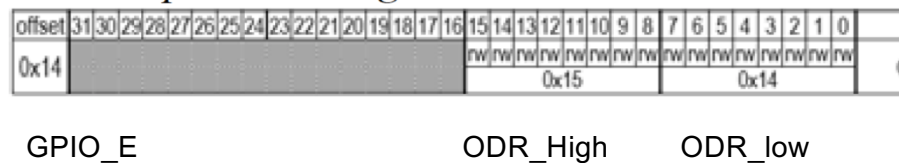
Control-register(RW)=0;
Wait tsu1;
Control-register(E)=1;
Data-register = (8 bits);
Wait tsu2; (until at least tw passes)
E = 0;
Wait until a total tc passes;
    
```

Examples: Interface/Algorithms

1. For example setup port GPIO_E for the ascii-display
2. The Ascii-display expects:
Data-register (bit 8 - 15) => ODR_high-byte
Control-register (bit 0 - 7) => ODR_low-byte



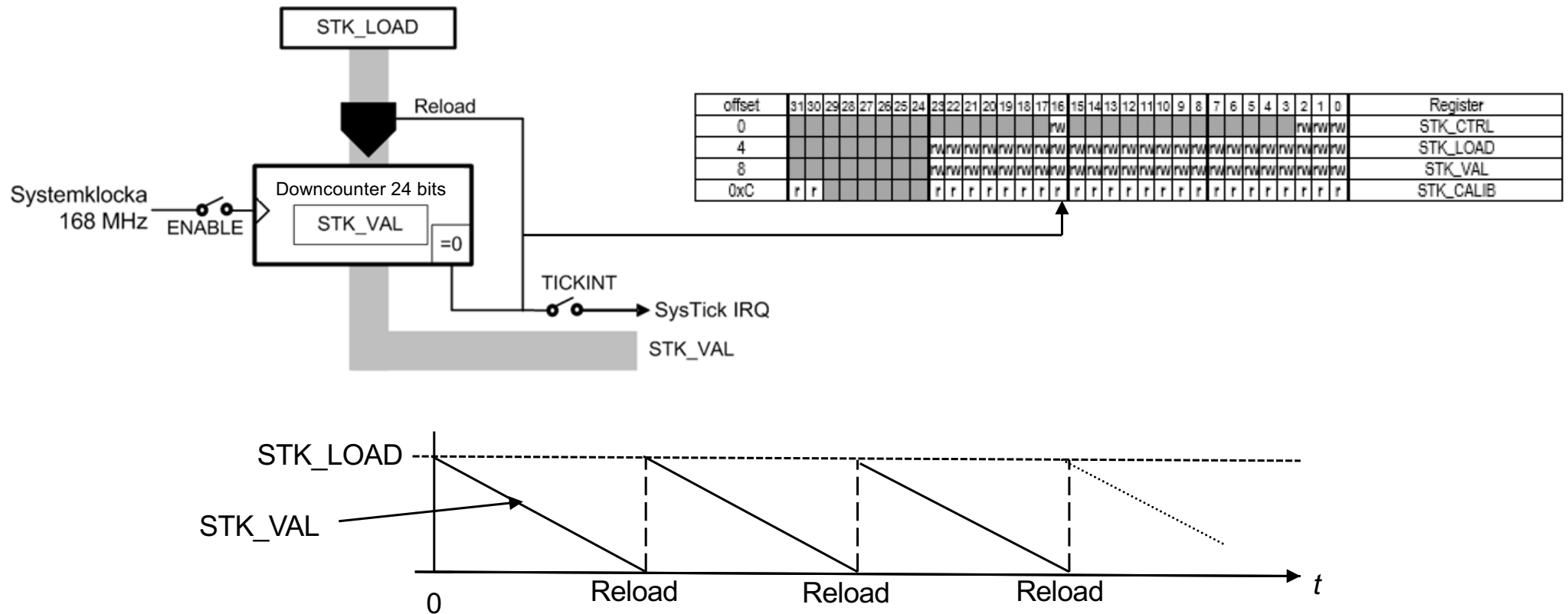
ODR Output Data Register



```
#define GPIO_E      0x40021000      /* MD407 port E */
#define portOdrLow  ((volatile unsigned char *)(GPIO_E+0x14))
#define portOdrHigh ((volatile unsigned char *)(GPIO_E+0x14+1))
#define B_E         0x40
#define B_SELECT     0x04
#define B_RW         0x02
#define B_RS         0x01

// RS is set to 0 for command or to 1 for data read/write
*portOdrLow |= B_SELECT; // set the select-bit
*portOdrLow &= ~B_RW; // turn off RW
delay(t_su1= ~40ns);
*portOdrLow |= B_E; // set enable
*portOdrHigh = our 8 bits of data;
delay(max(t_su2=80ns, t_w=240ns));
*portOdrLow &= ~B_E; // reset enable
delay( (t_c=500ns) - 240ns);
```

Counting Circuit - "SysTick"



Counting Circuit - 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																	STK_CTRL

STK_LOAD (0xE000E014) Räknavintervall

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

STK_VAL (0xE000E018) Räknavärde

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
8									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_VAL

Delay():

Algoritm:

```

STK_CTRL = 0      Reset SysTick
STK_LOAD = CountValue
STK_VAL = 0       Reset the counter register
STK_CTRL = 5      Restart the counter
Wait until COUNTFLAG=1
STK_CTRL = 0      Reset SysTick
  
```

STK_CTRL 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 2: (CLKSOURCE) biten är 1 efter RESET:

0: Systemklocka/8

1: Systemklocka

Bit 1: (TICKINT): Aktivera avbrott

0: Inget avbrott genereras.

1: Då räknaren slår om till 0 genereras SysTick avbrott.

Anm: Man kan programvarumässigt undersöka om räknaren räknat ned till 0 genom att kontrollera biten COUNTFLAG, det är alltså inte nödvändigt att använda avbrottsmekanismen.

Bit 0: (ENABLE) Aktivera räknare

Då ENABLE ettställs laddas värdet från STK_LOAD till STK_VAL och räknaren börjar räkna ned. Då räknaren nått 0, ettställs COUNTFLAG och proceduren upprepas.

0: Räknare deaktiverad

1: Räknare aktiverad

78

STK_CTRL = 0 Reset SysTick

STK_LOAD = CountValue

STK_VAL = 0 Reset the counter register

STK_CTRL = 5 Restart the counter

Wait until COUNTFLAG=1

STK_CTRL = 0 Reset SysTick

ister

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

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

STK_VAL (0xE000E018) Räknarvärde

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
8									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_VAL

Bit0 = 1 (enable)
Bit1 = 0 (no interrupt)
Bit2 = 1 (system clock)

Example 1: 250 ns delay with “SysTick”

How many cycles?

1. Create a function `delay_250ns(void)` which blocks (delays) the calling function by least 250 ns.
2. Also show how this can be used to create a delay routine `delay_milli( unsigned int millisec)` which delays the application execution by a variable number of milliseconds.

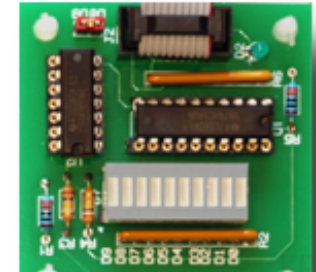
```
#define STK_CTRL ((volatile unsigned int *)(0xE000E010))
#define STK_LOAD ((volatile unsigned int *)(0xE000E014))
#define STK_VAL ((volatile unsigned int *)(0xE000E018))

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;
}
```

```
void delay_micro(unsigned int us)
{
    while(us--) {
        delay_250ns();
        delay_250ns();
        delay_250ns();
        delay_250ns();
    }
}

// delay_milli(): 1 ms = 1000 us
void delay_milli( unsigned int ms )
{
    while( ms-- )
        delay_micro(1000);
}
```

Example 2: Periodic blinking led



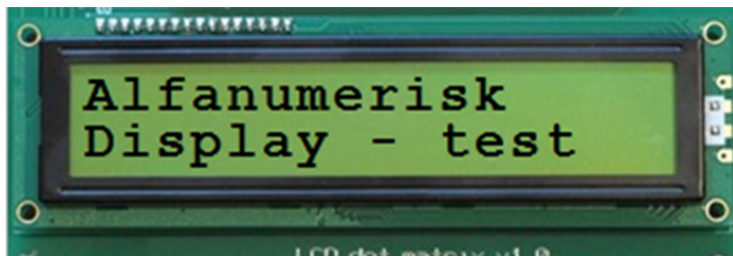
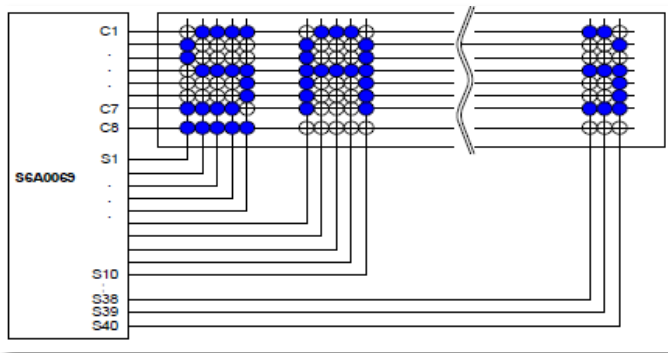
```
#define PORT_BARGRAPH_BASE 0x40020C00 /* MD407 port D */

#define portBargraphModer ((volatile unsigned int *) (PORT_BARGRAPH_BASE))
#define portBargraphOtyper ((volatile unsigned short *) (PORT_BARGRAPH_BASE+0x4))
#define portBargraphOspeedr ((volatile unsigned int *) (PORT_BARGRAPH_BASE+0x8))
#define portBargraphOdrLow ((volatile unsigned char *) (PORT_BARGRAPH_BASE+0x14))

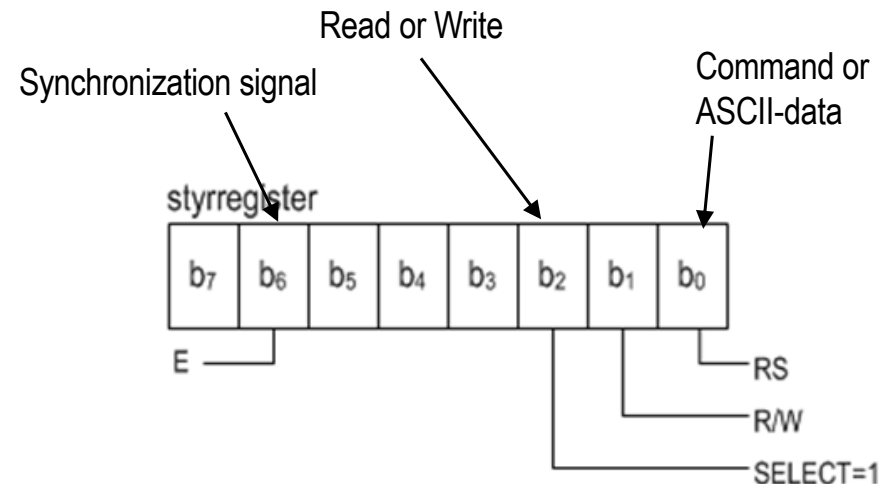
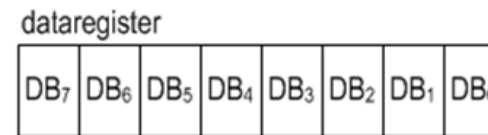
void main(void)
{
    init_app();
    while(1)
    {
        *portBargraphOdrLow = 0;
        delay_milli(500);
        *portBargraphOdrLow = 0xFF;
        delay_milli(500);
    }
}
```

Programming the ASCII-display

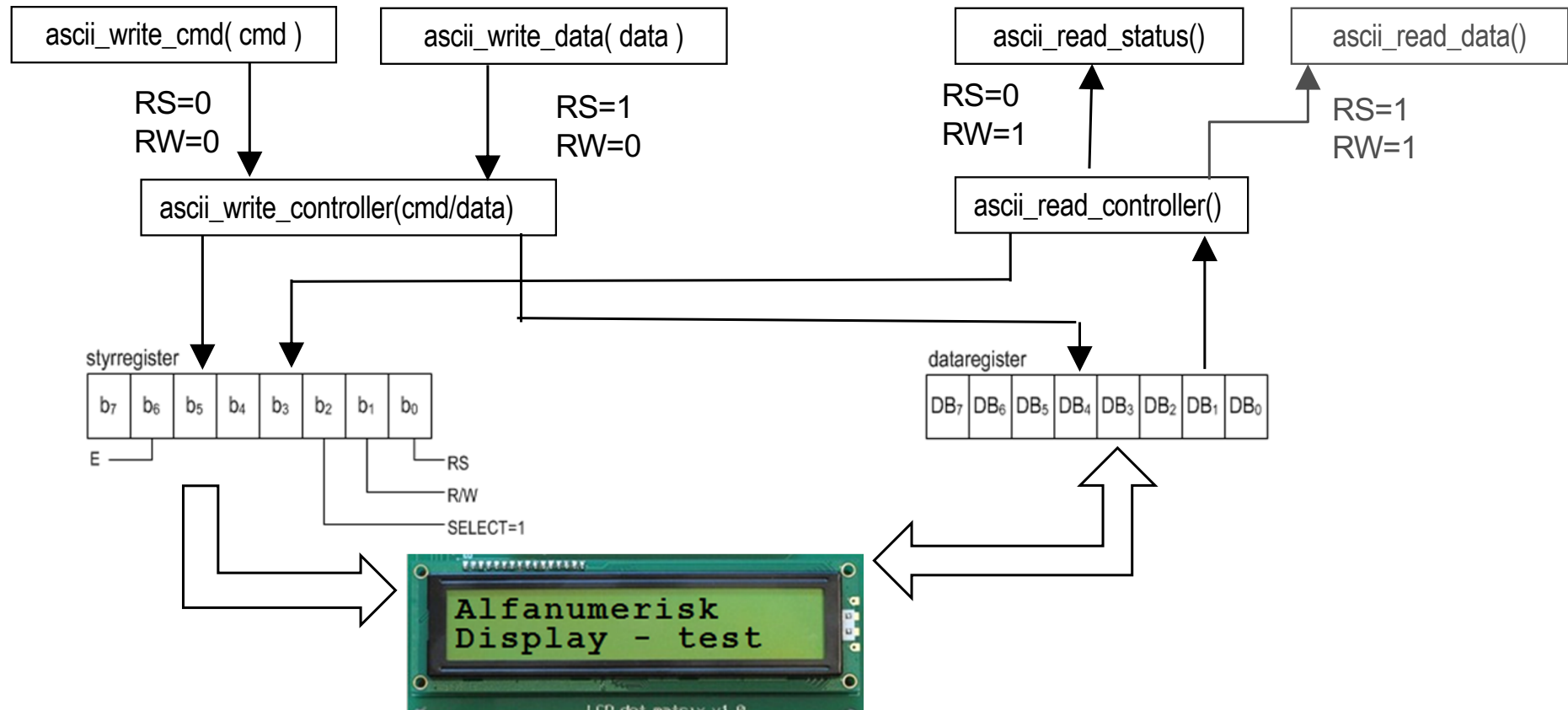
The circuit translates ASCII characters into corresponding bit patterns via a simple interface :
control register - 4 bits used
data register - 8 bits used



Byte to be transferred

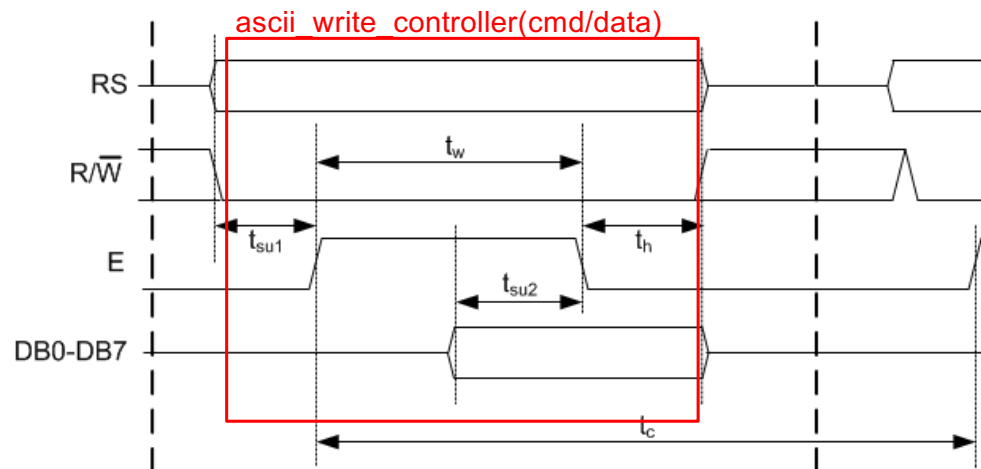


Program Structure



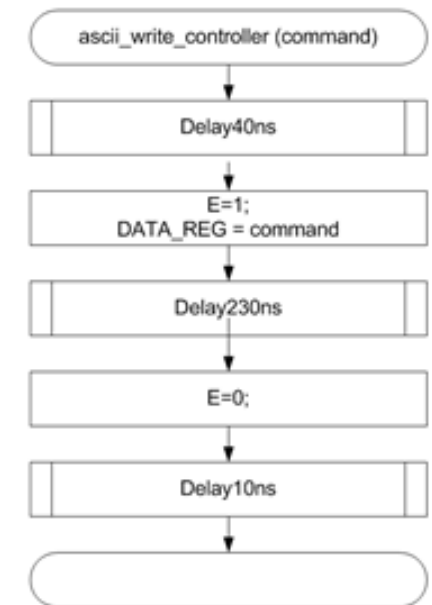
Write Cycle - Characteristics

The Datasheet time diagram illustrates how the control signal and data are activated for a write cycle.



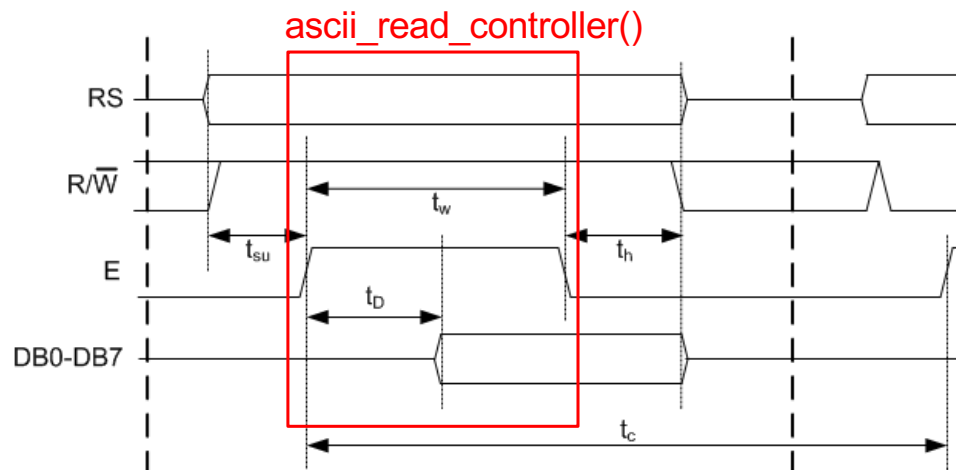
		min
t_c	Cycle time	500 ns
t_w	Clock pulse ("Enable") duration (high and low)	230 ns
t_{su1}	Control signal setup-time, before positive edge	40 ns
t_{su2}	setup-time for data, write, before negative edge	80 ns
t_h	hold-time, duration (after negativ edge)	10 ns

Algorithm:



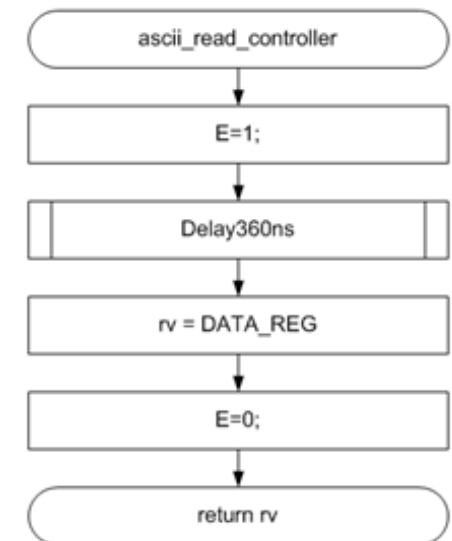
Read Cycle - Characteristics

The Datasheet time diagram illustrates how the control signal should be set and how the display module answers with data.



		min	max
t_c	Cycle time	1000 ns	
t_w	Clock pulse ("Enable") duration (high and low)	450 ns	
t_{su}	Control signal setup-time, before positive edge	60 ns	
t_D	setup-time for data, reading, before negative edge		360 ns
t_h	hold-time, duration (after negative edge)	10 ns	

Algorithm:



Example 3: Programming of the ASCII-Display

Implement functions:

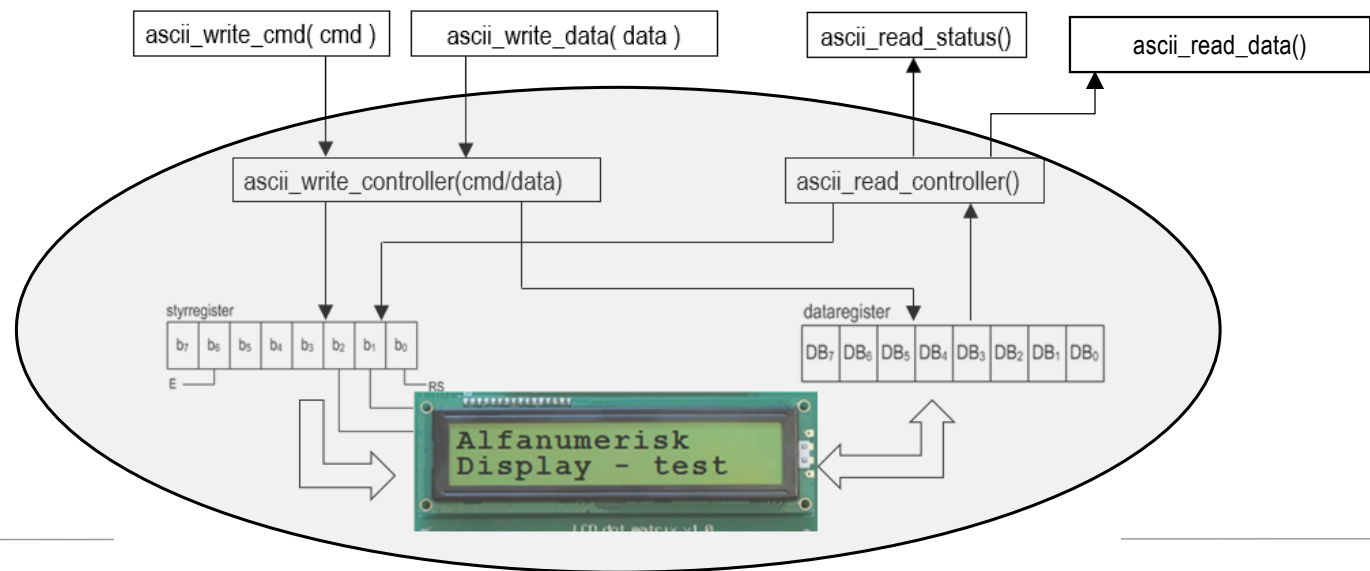
`ascii_write_controller(cmd/data)`
`ascii_read_controller()`

`ascii_init()`

`ascii_write_char()`

`ascii_gotoxy()`

Programstruktur



Example 3:

```
#define B_E      0x40
#define B_SELECT 4
#define B_RW     2
#define B_RS     1

void ascii_ctrl_bit_set( unsigned char x )
{
    GPIO_E.odrLow |= ( B_SELECT | x );
}

void ascii_ctrl_bit_clear( unsigned char x )
{
    GPIO_E.odrLow &= ( B_SELECT | ~x);
}
```

```
void ascii_write_controller( unsigned char c)
{
    ascii_ctrl_bit_set(B_E);
    GPIO_E.odrHigh = c;
    delay_250ns();
    ascii_ctrl_bit_clear(B_E);
}
```

```
unsigned char ascii_read_controller( void )
{
    ascii_ctrl_bit_set(B_E);
    delay_250ns();
    delay_250ns();
    char c = GPIO_E.idrHigh;
    ascii_ctrl_bit_clear(B_E);
    return c;
}
```

GPIO

General Purpose Input Output

GPIO A: 0x40020000

GPIO B: 0x40020400

GPIO C: 0x40020800

GPIO D: 0x40020C00

GPIO E; 0x40021000

[illegible]

MODER *Port mode 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	mnemonic
0x0	pin15		pin14		pin13		pin12		pin11		pin10		pin9		pin8		pin7		pin6		pin5		pin4		pin3		pin2		pin1		pin0		GPIO_MODER

För varje portpinne används 2 bitar i MODER för att konfigurera pinnen enligt:

00: ingång

01: utgång

10: alternativ funktion

11: analog

Registret har organiserats så att bitar 31,30 konfigurerar portpinne 15, bitar 29,28 konfigurerar portpinne 14 osv.

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	mnemonic								
0x04																	rw						rw						rw						rw						GPIO_OTYPER
																	0x05						0x04																		

Bitar 31:16 är reserverade och ska inte ändras. För varje annan portpinne används en bit för att konfigurera pinnen enligt:

0: push-pull

1: open drain

OSPEEDR *Output SPEED 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	mnemonic																																
0x08	pin15				pin14				pin13				pin12				pin11				pin10				pin9				pin8				pin7				pin6				pin5				pin4				pin3				pin2				pin1				pin0				GPIO_OSPEED

Registret används för att kontrollera uppdateringsfrekvensen för en portpinne. Exakta frekvenser anges i processorns datablad.

00: low speed

01: medium speed

10: fast speed

11: high speed

PUPDR *Pull-Up/Pull-Down Register*

[illegible]

För varje portpinne används 2 bitar i PUPDR för att konfigurera pinnen enligt:

00: floating

01: pull-up

10: pull-down

11: reserverad

IDR *Input Data 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	mnemonic		
0x10																	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	GPIO_IDR
																	0x11						0x10												

Bitar 16 tom 31 används inte och ska hållas vid sitt RESET-värde, dvs 0. Bitar 0 t.o.m 15 avspeglar portens nivåer då pinnarna är konfigurerade som ingångar.

ODR *Output Data 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	mnemonic									
0x14																	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	GPIODR

Bitar 16 tom 31 används inte och ska hållas vid sitt RESET-värde, dvs 0. Bitar 0 t.o.m 15 sätter portens nivåer då pinnarna är konfigurerade som utgångar. Bitarna är både skriv- och läsbara, vid läsning ger biten det senast skrivna värdet till samma bit.

BSSR *Bit set reset register*

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
0x18	BR15	BR14	BR13	BR12	BR11	BR10	BSR	BSR	BSR	BSR	BSR	BSR	BSR	BSR	BSR	BSR	GPIO_BSRR
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

LCKR *Port configuration lock register*[illegible]AFRL *Alternate function low 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	mnemonic
0x20	AFRL	7	3	0	AFRL	6	3	0	AFRL	5	3	0	AFRL	4	3	0	AFRL	3	3	0	AFRL	2	3	0	AFRL	1	3	0	AFRL	0	3	0	GPIO AFRL

AFRH *Alternate function high 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	mnemonic				
0x24	AFRH15[3]				AFRH14[3]				AFRH13[3]				AFRH12[3]				AFRH11[3]				AFRH10[3]				AFRH9[3]				AFRH8[3]								GPIO_AFRH
	0				0				0				0				0				1				0												

Remarks

Arbetsboken, page 85:

ascii_gotoxy(row, column)

address = row-1

om column = 2

address = address + 0x40

ascii_write_cmd(0x80 | address)

instruction-specific delay (i.e 43 μ s)

```
void ascii_gotoxy( unsigned char x, unsigned char y) {  
    unsigned char address;  
  
    if(y!=1)  
        address=0x40 | (x-1);  
    else  
        address=x-1;  
    ascii_write_cmd( 0x80 | address);  
    delay_micro( 45 );  
}
```

← Add this row for safety.