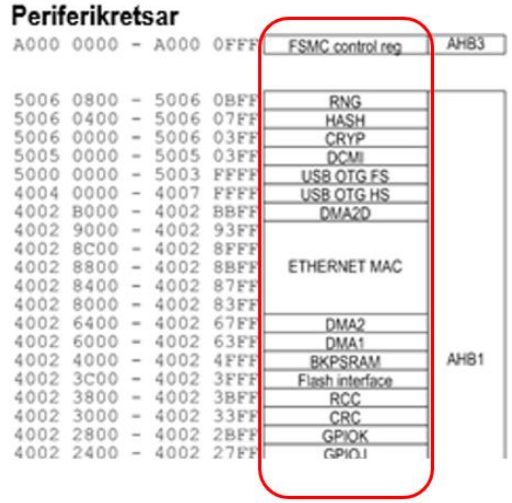
Syften:

- Belys och exemplifiera begreppet ”port”.
- Typkonvertering av konstanter
- Bitoperationer.

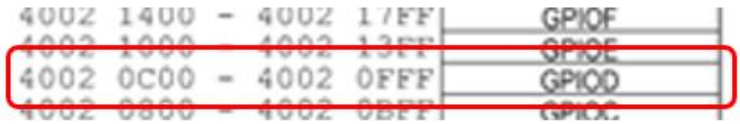
Vi behandlar ”portens anatomi”, portadressering och visar ett exempel på hur beskrivningen av en port avbildas som en ”programmerares bild”.

Kort introduktion:

Med ”extern enhet” (”periferikrets”) menas sådana kretsar som kan adresseras och manipuleras från centralenheten.

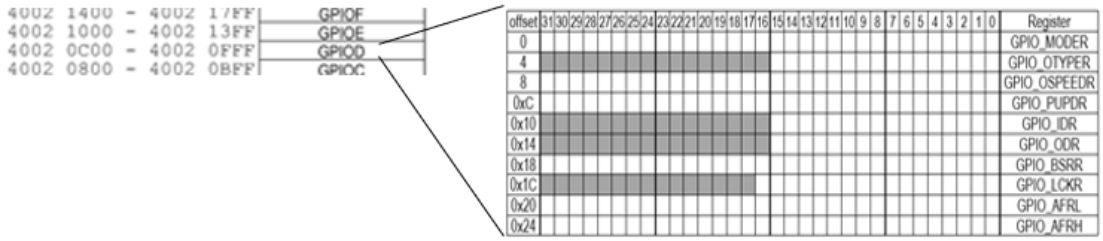


Ett hårdvarugränssnitt mot en extern enhet kallas ”port”.



En port har alltså en på förhand bestämd adress i adressrummet.

Gränssnittets portpinnar är på olika sätt representerade i portens register.
Beroende på funktion och komplexitet har porten ett antal olika register.



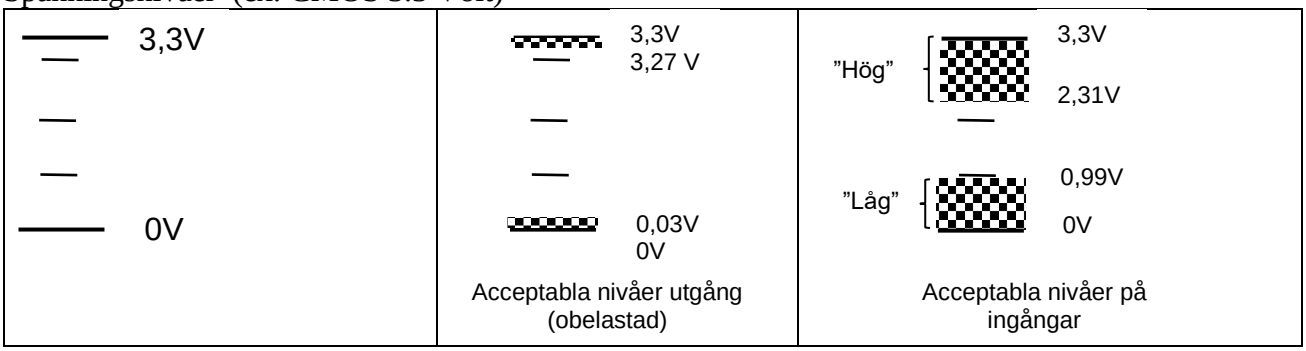
Exempel på portar med strukturellt olika egenskaper:

- Parallell in- och utmatning, digitalt format (”parallellkommunikation”)
- Seriell in- och utmatning, digitalt format (”seriekommunikation”)
- Analog/Digital- omvandling, Inmatning av kontinuerlig , analog signal)
- Digital/Analog- omvandling, utmatning till kontinuerlig signal

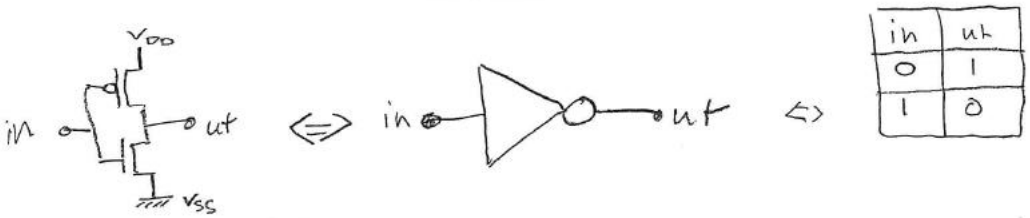
Vi behandlar här och nu enbart parallell in- och utmatning

Portens ”anatomi”

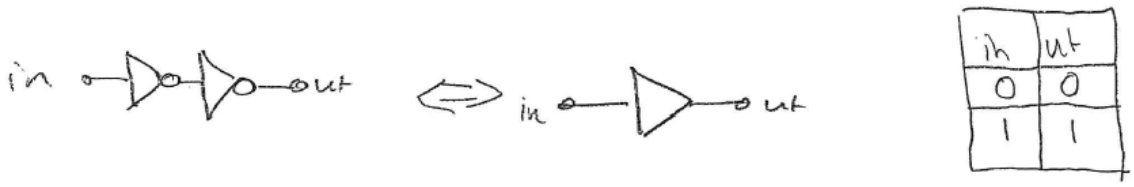
Spänningsnivåer (ex. CMOS 3.3 Volt)



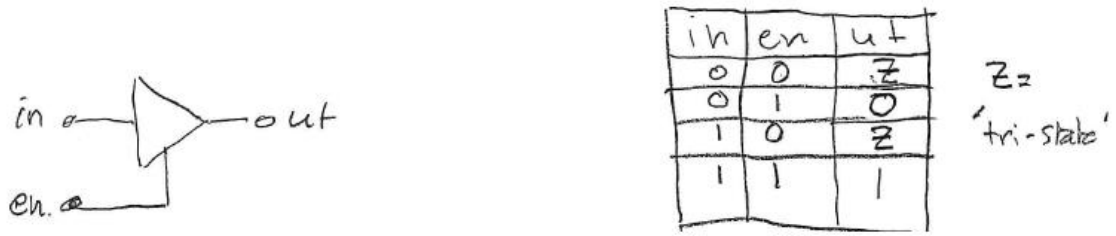
Vi påminner om (förenklad) beskrivning på MOS-transistornivå, speciellt för att förstå vad som skiljer porten från en variabel.
Förenklat schema CMOS inverterare



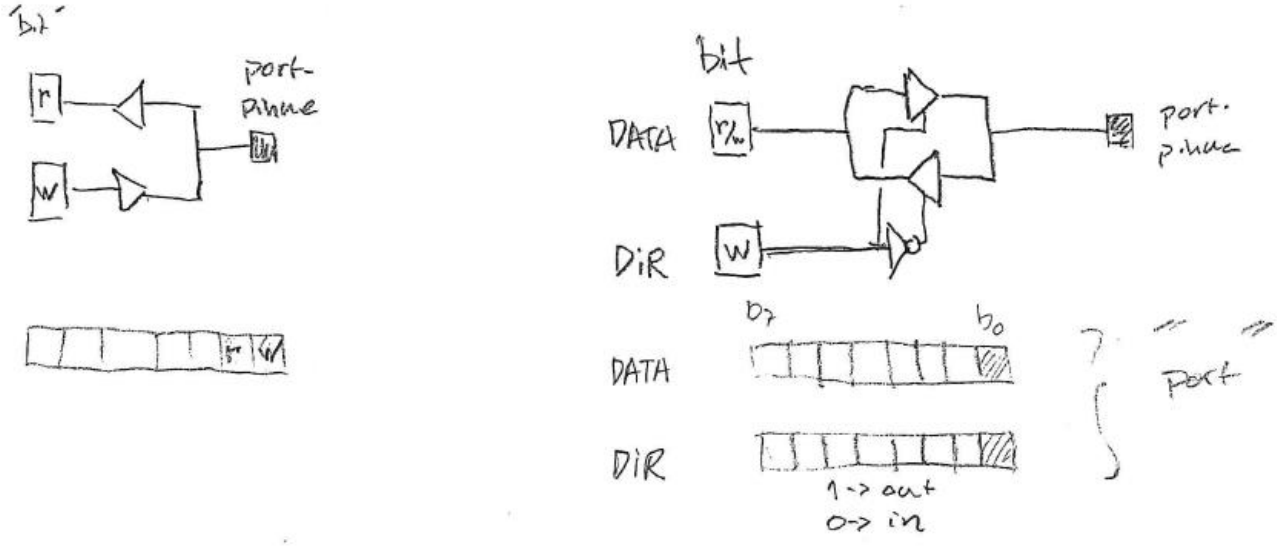
CMOS-anpassningsförstärkare används för att säkerställa att digitala signaler finns i väldefinierade nivåer.



Tri state grind, anpassningsförstärkare med ”enable” ingång.



Kopplingar mellan portpinne/register



Portadressering

En fast adress, dvs. en konstant uttrycks i C:
`{1..9}{0..9}*`
`'O'{0..7}+` oktal form
`0x{0..9,a..f}+` hexadecimal form

Exempelvis kan adressen `0xF8000000` på hexadecimal form också skrivas som det decimala talet:
`4160749568`
eller det oktala talet:
`O37000000000`

Hur "skriva" till portens register i C?
`0xF8000000 = värde; går inte...` en konstant har alltid typen `int.` En konstant kan inte tilldelas värde....

Vi måste *typkonvertera* konstanten, syntax exempel:
`(char *) 0xF8000000` betyder "F8000000 är en pekare till en char

```
#define PORTADD  (char *) 0xF8000000
```

Potens register kan nu läsas/skrivas indirekt:

```
*(PORTADD) = .....;        skriv till portens register
..... = *(PORTADD);        läs från portens register
```

Var noga med parenteser....

Bitoperationer:
Öka läsbarhet (minska risken för fel) genom att definiera macron för att ange bitpositioner, exempelvis:

```
#define bit0 (1<<0)
#define bit1 (1<<1)
#define bit2 (1<<2)
#define bit3 (1<<3)
```
osv...

Kodexempel "Sätt bit", "nollställ bit" och "toggla bit" i porten.

```
#define PORTADD  ((char *) 0xF8000000)
#define PORTREG  (*PORTADD)

PORTREG  |= bit1;     // Ettställ bit 3
PORTREG  &= ~bit2;    // Nollställ bit 2
PORTREG  ^= bit3;     // Toggla bit 3
```

Demonstrationsuppgift, (hämtad från gammal tentamensuppgift).

En robotarm styrs via ett gränssnitt med sex olika register: styrregister, statusregister, två dataregister och två positionsregister. Styrregistret används för att kontrollera robotarmens rörelser och dataregistren används för att ange x- respektive y-koordinater som mål vid robotarmens förflyttning. De båda positionsregisten anger de aktuella x- respektive y-koordinaterna för robotarmen. Registren i robotens gränssnitt beskrivs av följande:

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
0																	status								ctrl							
																				IP		ER			IE	IA			EN			RS
																				r		rw			rw	w			rw			rw
4	dataX																dataY															
	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
8	posX																posY															
	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Anm: **r**: biten är enbart läsbar, **w**: biten är enbart skrivbar, **rw**: biten är både läsbar och skrivbar. Försök att skriva en bit som är enbart läsbar har ingen effekt. Försök att läsa en bit som är enbart skrivbar, returnerar alltid värdet 0.

IP: *In Position*. Biten sätts till 1 då innehållen i data och positionsregistren överensstämmer. Om avbrott är aktiverat (IE) genereras detta med nummer 4 i vektortabellen.

ER: *Error*. Då ett fel som gör att robotarmen inte kan röra sig mot dataregistrens koordinater inträffat, stoppas robotarmen, och denna bit sätts till 1. Om avbrott är aktiverat genereras även detta. Biten återställs till noll genom att 1 skrivs till biten.

IE: *Interrupt Enable*, sätts till 1 för att aktivera avbrottsmekanismen i gränssnittet. Då avbrottsmekanismen är aktiverad genereras ett avbrott då data och positionsregistren överensstämmer, dvs. robotarmen nått målkoordinaterna eller då ett fel uppstår.

IA: *Interrupt Acknowledge*, då denna bit sätts till 1 återställs biten IRQ till 0 av robotens gränssnitt.

EN: *Enable*, sätts till 1 för att aktivera robotarmen, efter aktivering kommer denna att röra sig mot målkoordinaterna angivna i dataregistren. Positionsregistren uppdateras av roboten allt eftersom armen rör sig.

RS: *Reset*, då denna bit sätts till 1 återställs bitarna ERR, IRQ,IE och EN till 0 av robotens gränssnitt. RS-biten måste därefter återställas till 0 för att robotarmen ska kunna aktiveras.

a) Antag en sådan port på adress 0xF8000000 i adressrummet. Antag dessutom att status och kontrollregistren endast kan läsas som *byte*.
Visa makron av lämpliga typkonverteringar för dessa adresser.

Lösning:

```
#define CTRL    ((volatile unsigned char *)  0xF8000000)
#define STATUS  ((volatile unsigned char *)  0xF8000001)
```

b) Visa macron för lämpliga bitdefinitioner av status-registret.

Lösning:

```
#define IP (1<<4)
#define ER (1<<2)
```

c) Visa hur ER-biten i statusregistret nollställs.

Lösning:

```
*STATUS |= ER;     // Ettställ bit 2 i statusregistret
```

Diverse figurer, för OH etc...

Periferikretsar

A000 0000 - A000 0FFF	FSMC control reg	AHB3
5006 0800 - 5006 0BFF	RNG	AHB1
5006 0400 - 5006 07FF	HASH	
5006 0000 - 5006 03FF	CRYP	
5005 0000 - 5005 03FF	DCMI	
5000 0000 - 5003 FFFF	USB OTG FS	
4004 0000 - 4007 FFFF	USB OTG HS	
4002 B000 - 4002 BBFF	DMA2D	
4002 9000 - 4002 93FF	ETHERNET MAC	
4002 8C00 - 4002 8FFF		
4002 8800 - 4002 8BFF		
4002 8400 - 4002 87FF		
4002 8000 - 4002 83FF		
4002 6400 - 4002 67FF	DMA2	
4002 6000 - 4002 63FF	DMA1	
4002 4000 - 4002 4FFF	BKPSRAM	
4002 3C00 - 4002 3FFF	Flash interface	
4002 3800 - 4002 3BFF	RCC	
4002 3000 - 4002 33FF	CRC	
4002 2800 - 4002 2BFF	GPIOK	
4002 2400 - 4002 27FF	GPIOJ	
4002 2000 - 4002 23FF	GPIOI	
4002 1C00 - 4002 1FFF	GPIOH	
4002 1800 - 4002 1BFF	GPIOG	
4002 1400 - 4002 17FF	GPIOF	
4002 1000 - 4002 13FF	GPIOE	
4002 0C00 - 4002 0FFF	GPIOD	
4002 0800 - 4002 0BFF	GPIOC	
4002 0400 - 4002 07FF	GPIOB	
4002 0000 - 4002 03FF	GPIOA	
4001 6800 - 4001 6BFF	LCD-TFT	APB2
4001 5800 - 4001 4BFF	SAI1	
4001 5400 - 4001 57FF	SPI6	
4001 5000 - 4001 53FF	SPI5	
4001 4800 - 4001 4BFF	TIM11	
4001 4400 - 4001 47FF	TIM10	
4001 4000 - 4001 43FF	TIM9	
4001 3C00 - 4001 3FFF	EXTI	
4001 3800 - 4001 3BFF	SYSCFG	
4001 3400 - 4001 37FF	SPI4	
4001 3000 - 4001 33FF	SPI1	
4001 2C00 - 4001 2FFF	SDIO	
4001 2000 - 4001 23FF	ADC1-ADC2-ADC3	
4001 1400 - 4001 17FF	USART6	
4001 1000 - 4001 13FF	USART1	
4001 0400 - 4001 07FF	TIM8	
4001 0000 - 4001 03FF	TIM1	
4000 7C00 - 4000 7FFF	UART8	APB1
4000 7800 - 4000 7BFF	UART7	
4000 7400 - 4000 77FF	DAC	
4000 7000 - 4000 73FF	PWR	
4000 6800 - 4000 6BFF	CAN2	
4000 6400 - 4000 67FF	CAN1	
4000 5C00 - 4000 5FFF	I2C3	
4000 5800 - 4000 5BFF	I2C2	
4000 5400 - 4000 57FF	I2C1	
4000 5000 - 4000 53FF	UART5	
4000 4C00 - 4000 4FFF	UART4	
4000 4800 - 4000 4BFF	USART3	
4000 4400 - 4000 47FF	USART2	
4000 4000 - 4000 43FF	I2S3ext	
4000 3C00 - 4000 3FFF	SPI3/I2S3	
4000 3800 - 4000 3BFF	SPI2/I2S2	
4000 3400 - 4000 37FF	I2S2ext	
4000 3000 - 4000 33FF	IWDG	
4000 2C00 - 4000 2FFF	WWDG	
4000 2800 - 4000 2BFF	RTC & BKP Reg	
4000 2000 - 4000 23FF	TIM14	
4000 1C00 - 4000 1FFF	TIM13	
4000 1800 - 4000 1BFF	TIM12	
4000 1400 - 4000 17FF	TIM7	
4000 1000 - 4000 13FF	TIM6	
4000 0C00 - 4000 0FFF	TIM5	
4000 0800 - 4000 0BFF	TIM4	
4000 0400 - 4000 07FF	TIM3	
4000 0000 - 4000 03FF	TIM2	