

Digital IO

Ur innehållet:

- Digital IO - parallell in och utmatning
- Ideala och verkliga signaler
- Bitvis in- och utmatning
- Anslutning - fysiskt gränssnitt
- GPIO (General Purpose Input Output)-modul – tillämpningar
- Programmering av enkelt tangentbord

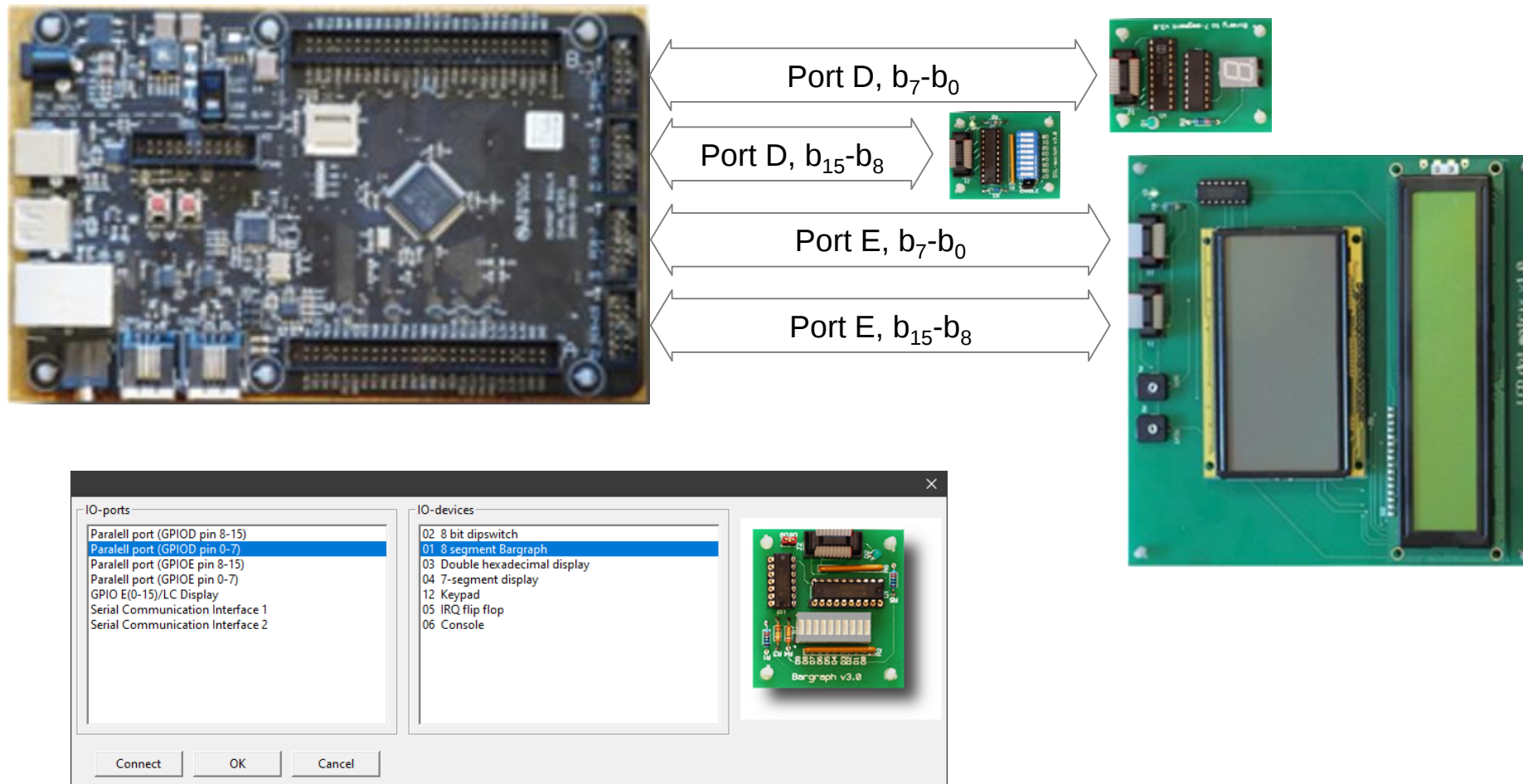
Läsanvisningar:

- Arbetsbok kapitel 4

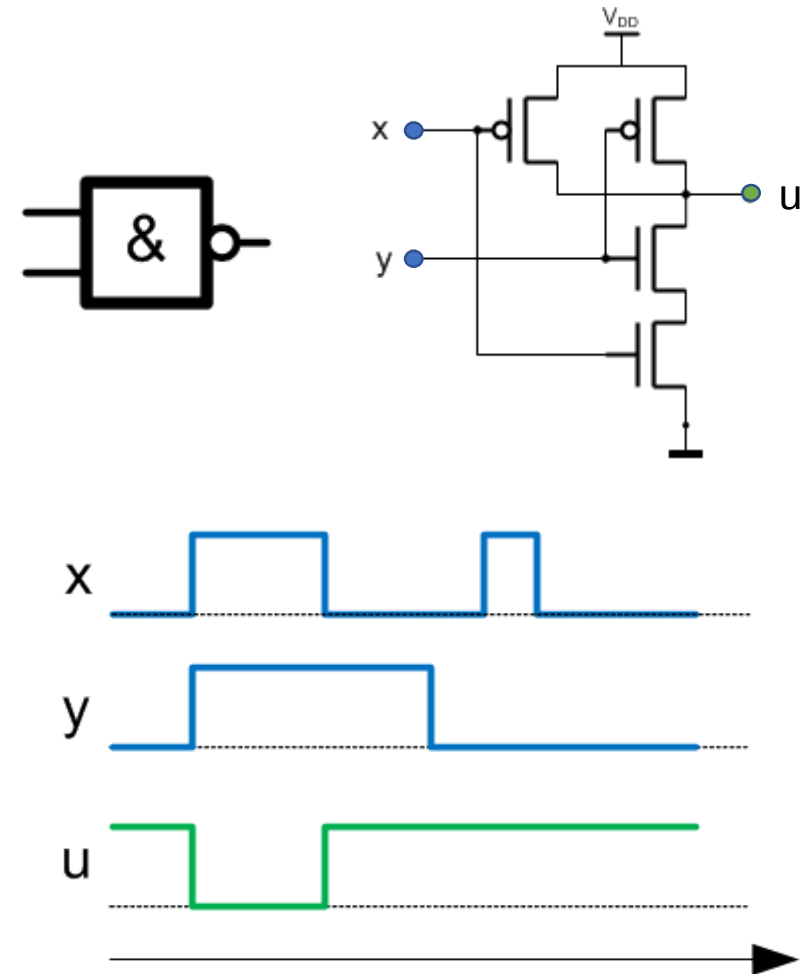
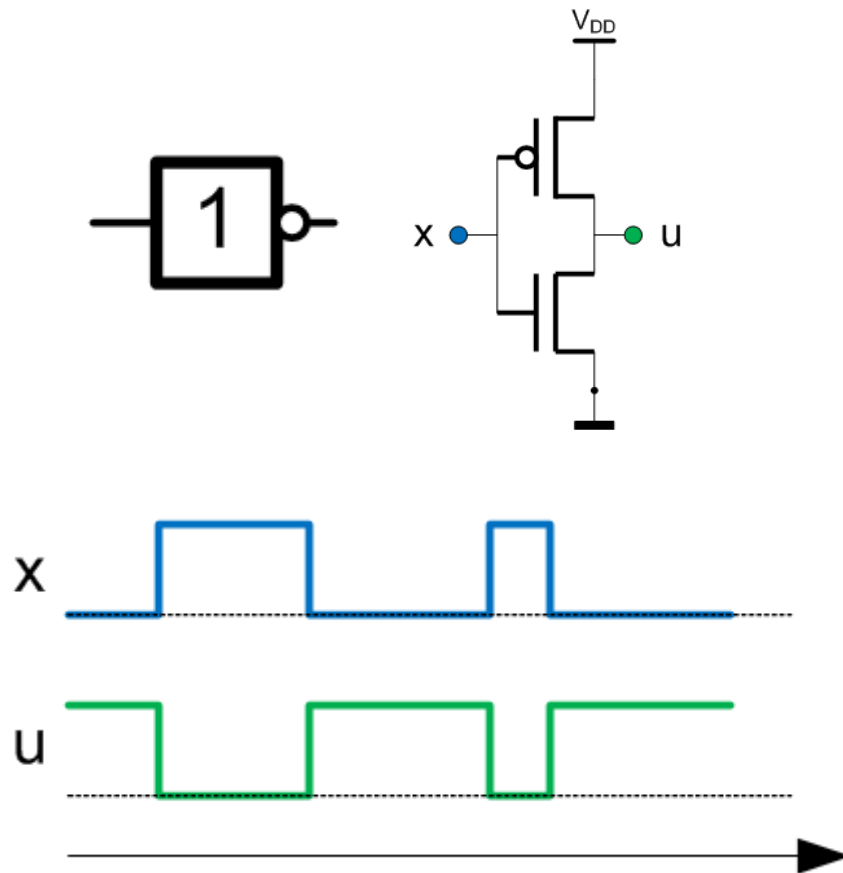
Målsättningar:

- Kunna konfigurera GPIO-modulen för parallell in-/ut- matning

Digital IO - parallell in- och utmatning

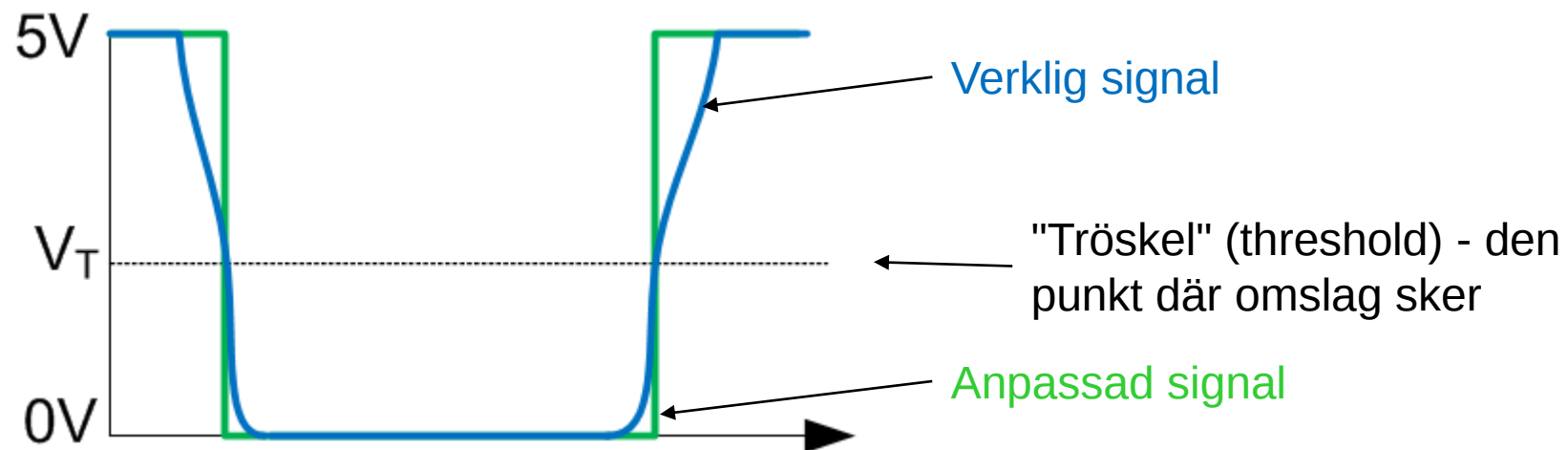


Ideala grindar - idealiserade signaler



Verkliga signaler - logiknivåer och anpassning

Verkliga signaler måste anpassas till distinkta CMOS-nivåer representerande logikvärdena '0' och '1'



Ideala och verkliga grindar

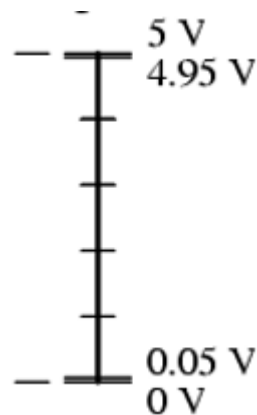
Hos ideala kretsar är tröskelnivån den samma.

Hos verkliga kretsar kan det vara spridning.

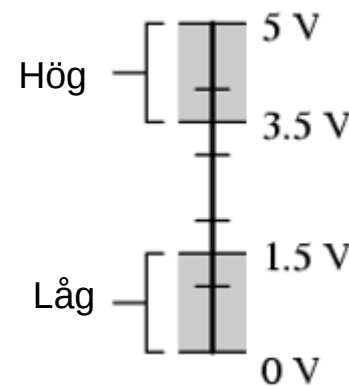
Tillverkarna specificerar "säkra" intervall för hög respektive låg nivå.

För CMOS-kretsar med matningsspänning 5 Volt gäller:

Acceptabla signaler på
utgångar



Acceptabla signaler på
ingångar



Skillnaden kallas **störmargin**:

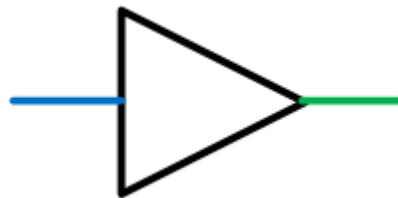
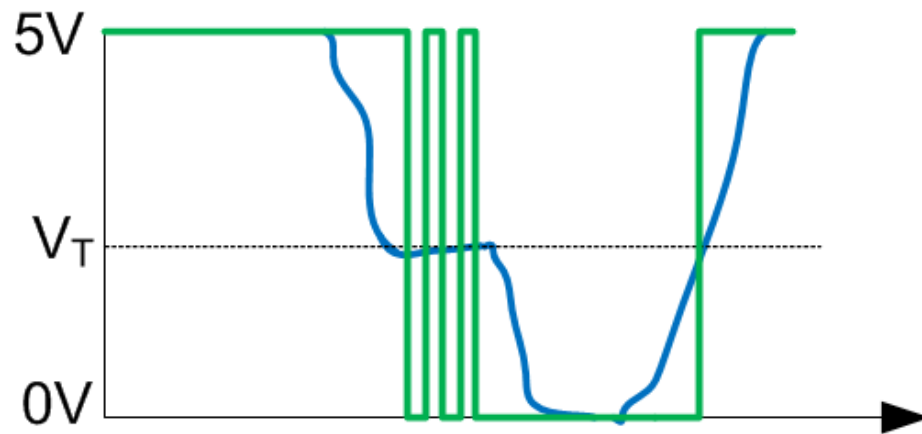
Hög nivå $4,95 - 3,5 = 1,4$ Volt

Låg nivå: $1,5 - 0,05 = 1,45$ Volt

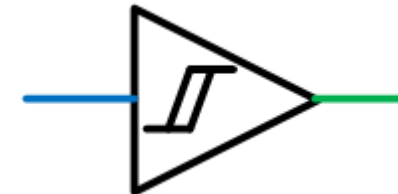
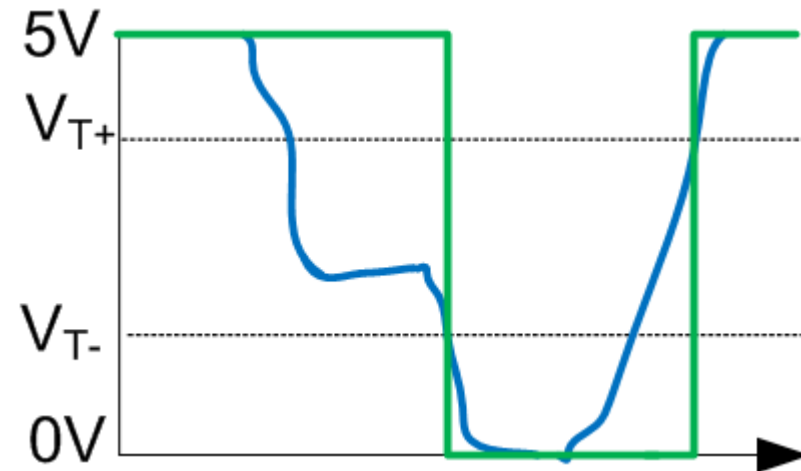
En signal mellan de angivna intervallen är **obestäm**d.

"Schmitt-trigger"

Brusiga insignaler kan generera
många oönskade omslag

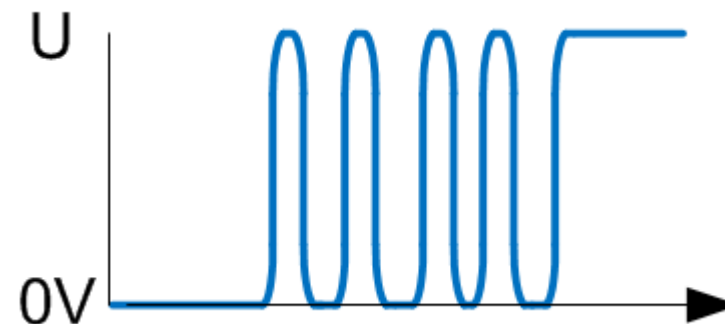
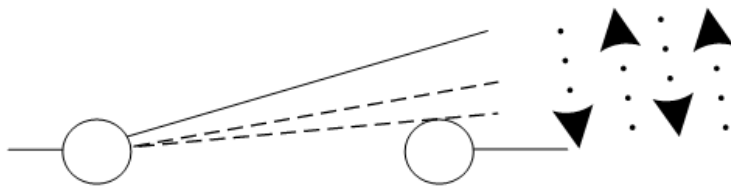
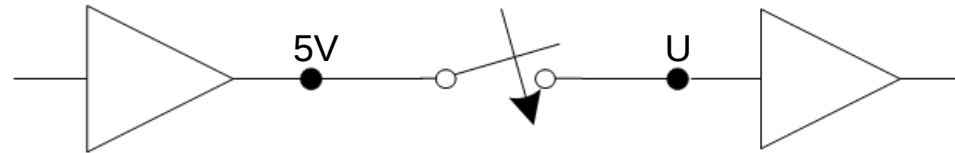


Med fastlagda tröskelnivåer
 V_{T+} för omslag från 0 till 1
och V_{T-} från 1 till 0



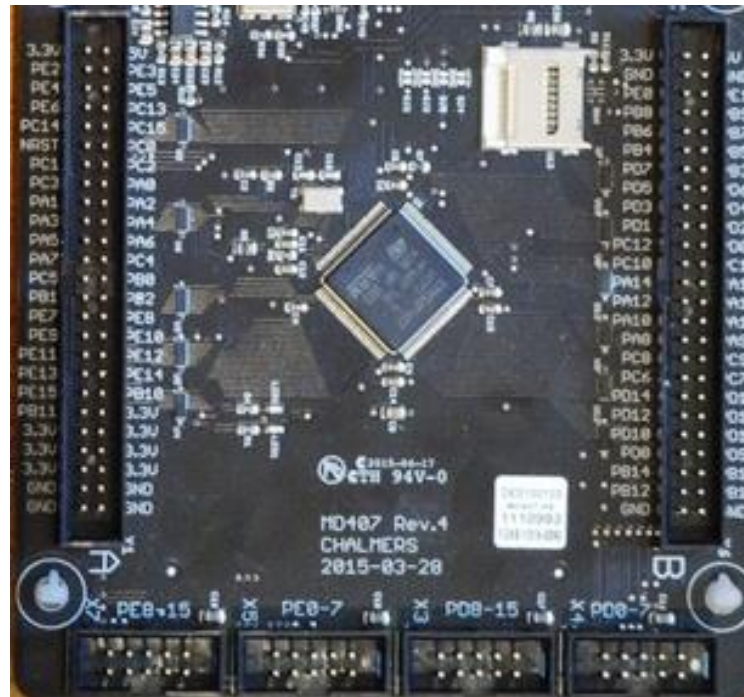
Kontaktstuds

Då en mekanisk omkopplare sluts kan den studsas flera gånger mot det slutande blecket innan den stabiliseras i slutet läge. Detta kan generera ett pulståg och kallas för "kontaktstudsar".

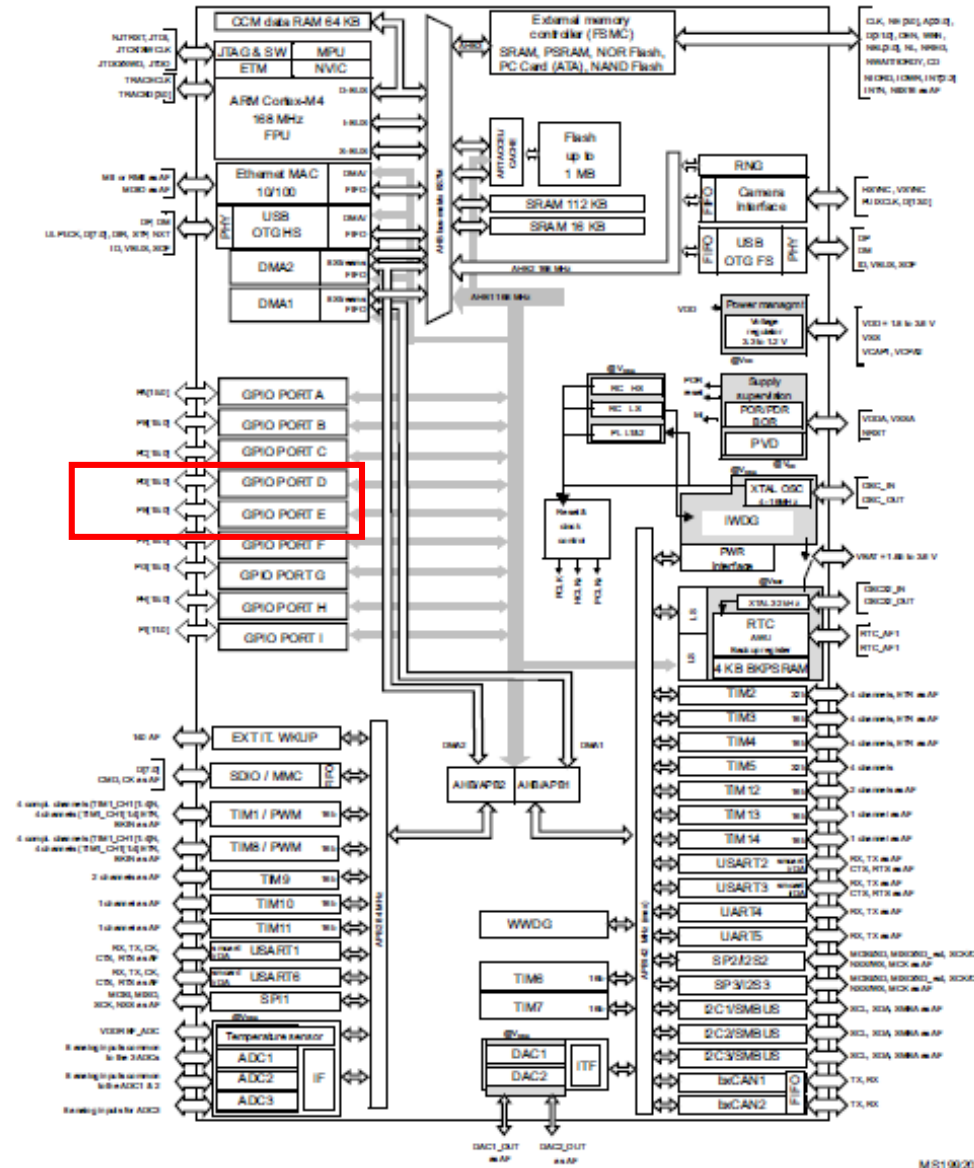


Anslutningar

STM32F407VGT7: Större delen av kretsens 100 pinnar har programmerbar funktion och organiserats i portar (A-E).



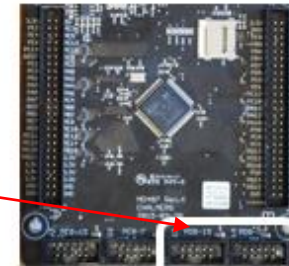
Hos MD407 används portar D och E för generell IO (32 pinnar) medan övriga portar i olika utsträckning används till förutbestämda funktioner.



GPIO-port, programmerarens bild

För varje port finns en uppsättning register där respektive pinnes funktion kan konfigureras. Registren kan läsas eller skrivas med *byte*, *halfword* eller *word*-operationer.

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																																	GPIO_MODER
4																																	GPIO_OTYPER
8																																	GPIO_OSPEEDR
0xC																																	GPIO_PUPDR
0x10																																	GPIO_IDR
0x14																																	GPIO_ODR
0x18																																	GPIO_BSRR
0x1C																																	GPIO_LCKR
0x20																																	GPIO_AFRH
0x24																																	GPIO_AFRH



Port D – 16 pinnar
IDR : Input data register
ODR: Output data register

De 16 pinnarna i en port kan konfigureras individuellt i *mode register* (MODER) för någon av funktionerna:

- **00: digital ingång**
- **01: digital utgång**
- *10: alternativ funktion*
- *11: analog funktion*

nu behandlar vi pinnarnas funktion som digital IO, dvs. de första två alternativen.

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	01		pin14				pin13				pin12				pin11				pin10				pin9				pin8				pin7				pin6				pin5				pin4				pin3				pin2				pin1				00		GPIO_MODER



GPIO - MODE-register

Vi vill sätta upp:

- port D bitar 0-7 som en 8-bitars utport
- port D bitar 8-15 som en 8-bitars inport.
- 00: digital ingång
- 01: digital utgång
- 10: alternativ funktion
- 11: analog funktion

GPIO

General Purpose Input Output

GPIO A: 0x40020000

GPIO B: 0x40020400

GPIO C: 0x40020800

GPIO D: 0x40020C00

GPIO E: 0x40021000

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																																	GPIO_MODER
4																																	GPIO_OTYPER
8																																	GPIO_OSPEEDR
0xC																																	GPIO_PUPDR
0x10																																	GPIO_IDR
0x14																																	GPIO_ODR
0x18																																	GPIO_BSRR
0x1C																																	GPIO_LCKR
0x20																																	GPIO_AFRH
0x24																																	GPIO_AFRH

Den önskade funktionen får vi om MODE-registret initieras enligt:

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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	GPIO_MODER
	0				0				0				0				5				5				5				5				

Vilket ger:

`*( (volatile unsigned int *) 0x40020C00 ) = 0x00005555;`

eller:

`LDR R0, =0x00005555`

`LDR R1, =0x40020C00`

`STR R0, [R1]`

"basic io"

Verifiera `app_init` genom att, i C, skriva ett enkelt testprogram som läser från inporten och skriver till utporten.

```
#define USE_ASM
void app_init ( void )
{
#ifdef USE_ASM
__asm__ volatile(" LDR R0,=0x00005555\n");
__asm__ volatile(" LDR R1,=0x40020C00\n");
__asm__ volatile(" STR R0,[R1]\n");
#else
*GPIO_MODER = 0x00005555;
#endif
}

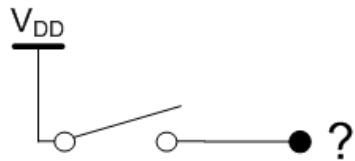
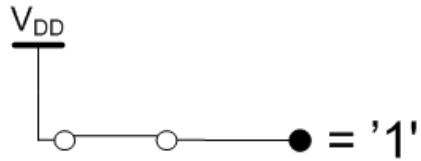
void main(void)
{
char c;
app_init();
while(1){
c = *GPIO_IDR_HIGH ;
*GPIO_ODR_LOW = c;
}
}
```

```
#define GPIO_D 0x40020C00
#define GPIO_MODER ((volatile unsigned int *) (GPIO_D))
#define GPIO_OTYPER ((volatile unsigned short *) (GPIO_D+0x4))
#define GPIO_PUPDR ((volatile unsigned int *) (GPIO_D+0xC))
#define GPIO_IDR_LOW ((volatile unsigned char *) (GPIO_D+0x10))
#define GPIO_IDR_HIGH ((volatile unsigned char *) (GPIO_D+0x11))
#define GPIO_ODR_LOW ((volatile unsigned char *) (GPIO_D+0x14))
#define GPIO_ODR_HIGH ((volatile unsigned char *) (GPIO_D+0x15))
```

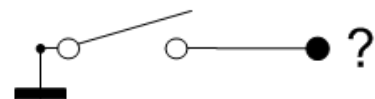
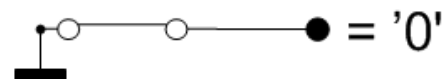
offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x10																		r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	GPIO_IDR

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																		r	w	r	w	r	w	r	w	r	w	r	w	r	w	r	GPIO_ODR

Digital ingång

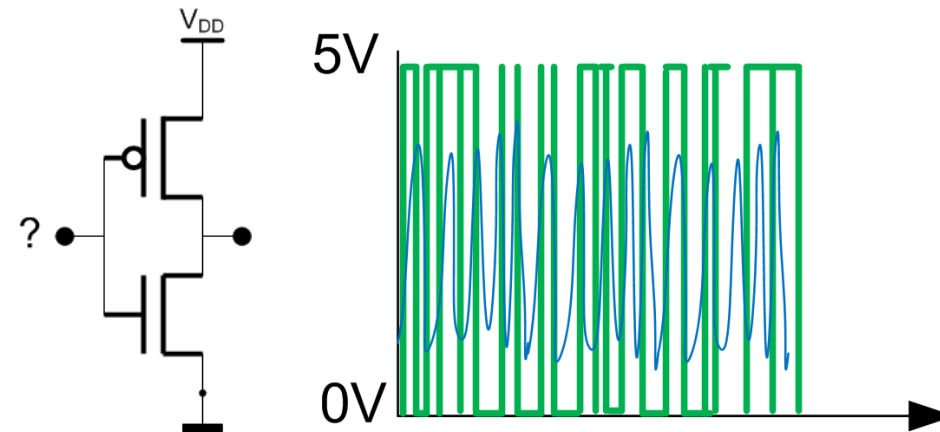


Ger "säker" etta då brytaren är sluten,
annars är signalen obestäm



Ger "säker" nolla då brytaren är sluten,
annars är signalen obestäm

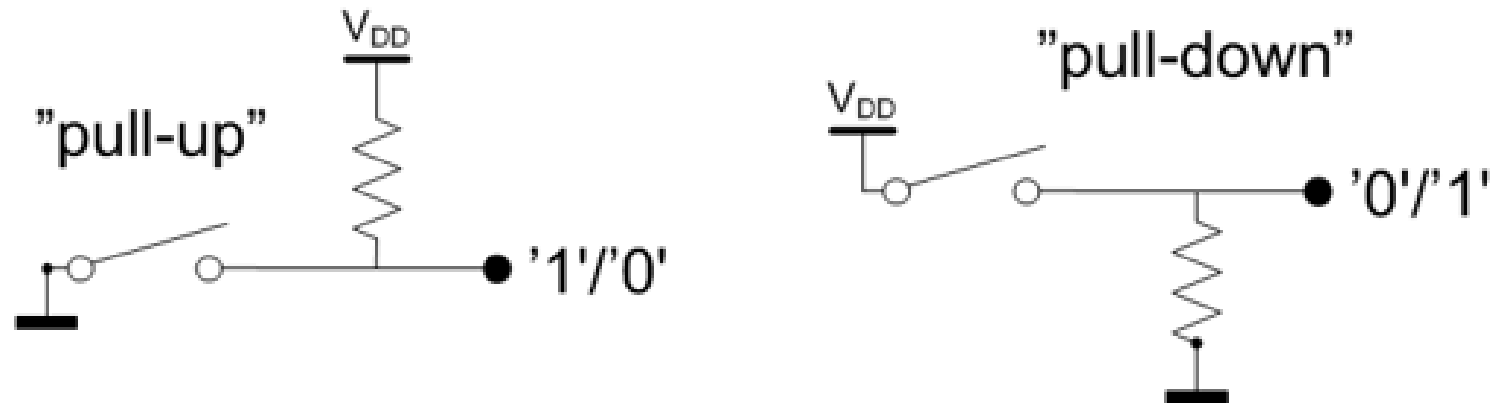
En *obestäm*d signal lämnar ingången i ett
"flytande" tillstånd



I ogynnsamma fall kan detta resultera i en
självsvängande krets som drar mycket ström
och dessutom kan orsaka störningar på
andra kretselement

"pull-up" eller "pull-down" ?

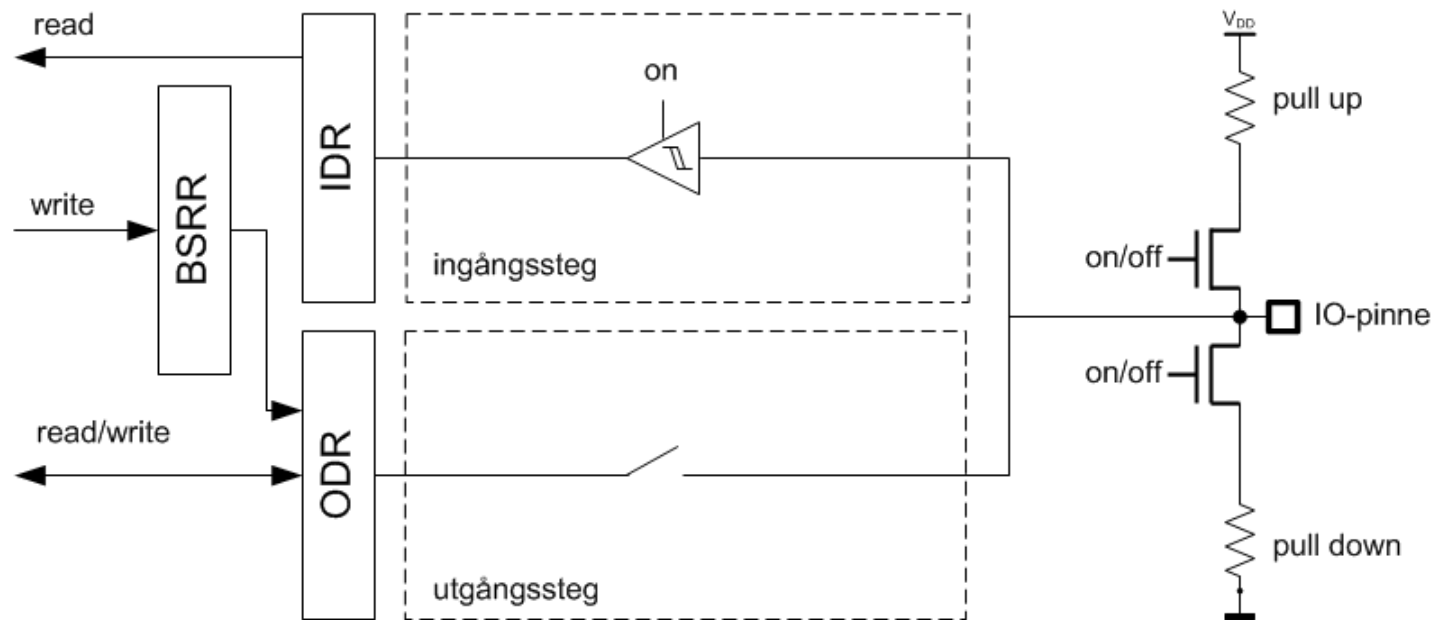
För att *säkerställa* en stabil nivå på ingången kan man koppla in ett motstånd till antingen V_{DD} eller GND.



Oavsett vilken lösning vi väljer kan vi alltid avgöra om brytaren är öppen eller stängd.

IO-pinne konfigurerad som ingång

Vi kan programmera "pull-up" eller "pull-down"-funktion, inget externt motstånd behövs.



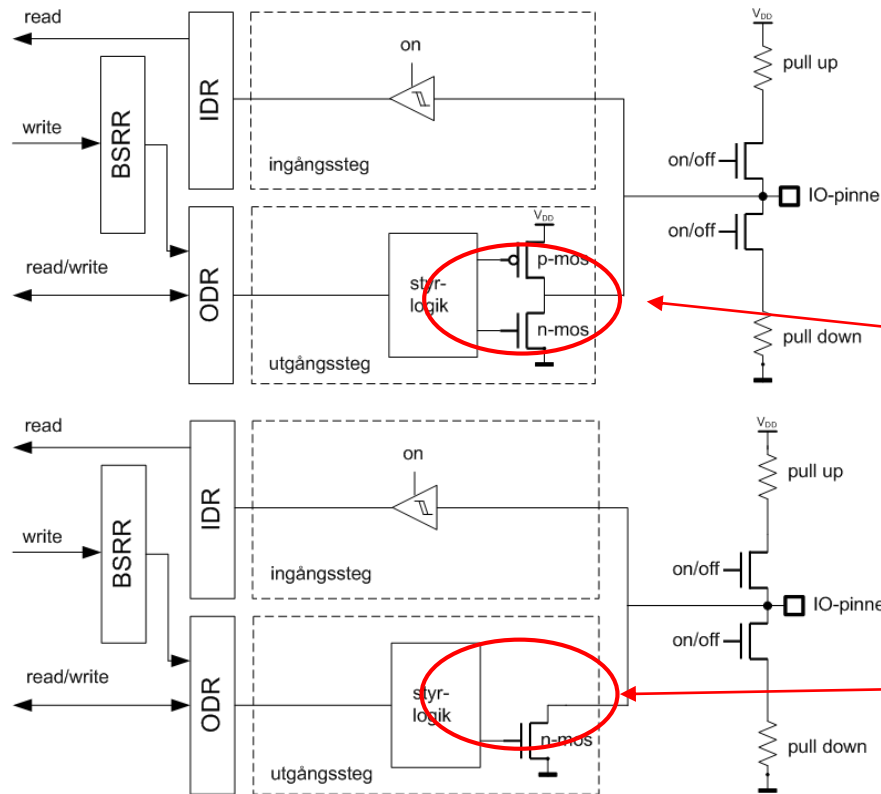
För varje portpinne används 2 bitar i PUPDR för att konfigurera pinnen enligt:

- 00: floating
- 01: pull-up
- 10: pull-down
- 11: reserverad

GPIO Pull Up Pull Down 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
0x0C	pin15	pin14	pin13	pin12	pin11	pin10	pin9	pin8	pin7	pin6	pin5	pin4	pin3	pin2	pin1	pin0											GPIO_PUPDR						
	0x0F				0x0E				0x0D				0x0C																				

IO-pinne konfigurerad som utgång



"push-pull" steg

då motsvarande bit i OTyPER är 0

"open drain" steg

då motsvarande bit i OTyPER är 1

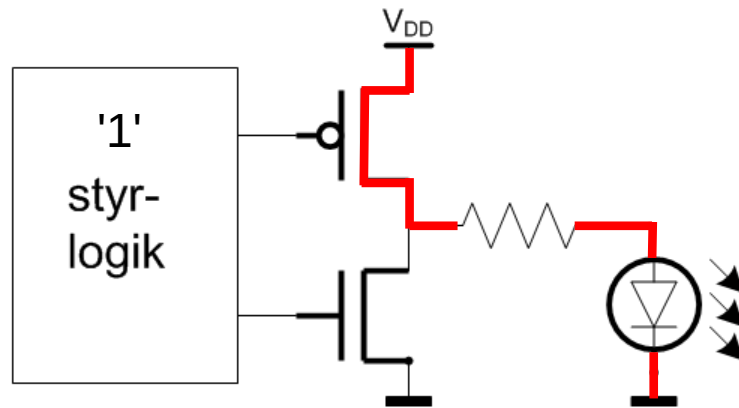
GPIO Output Type Register

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

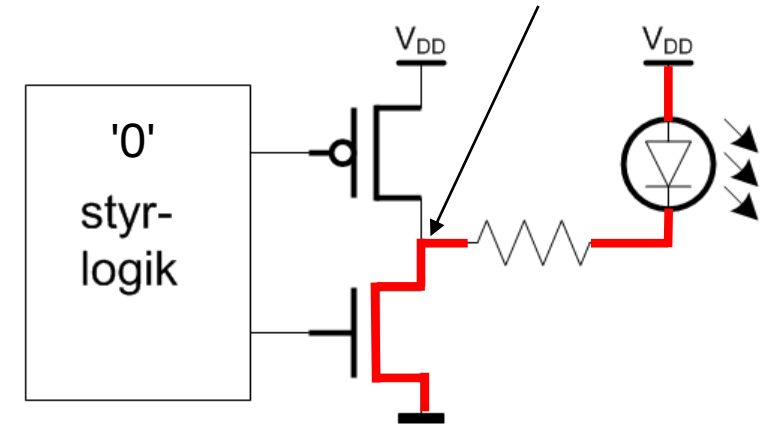
"push-pull"

Med "push-pull" menas att utgångssteget kopplar utsignalen antingen till V_{DD} eller GND.

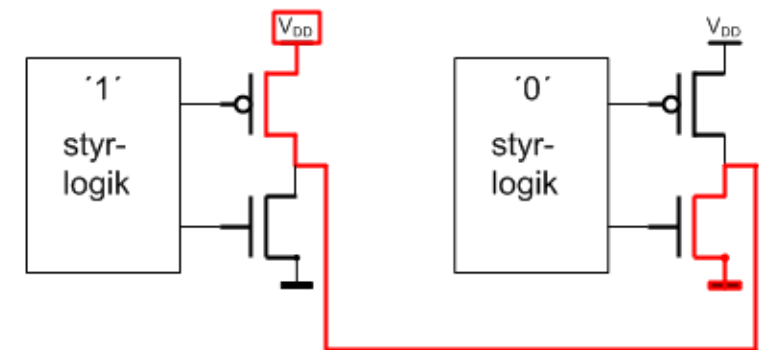
"push": Ljusdioden tänds då utgången är 1.



"pull": Ljusdioden tänds då utgången är 0.



Utgångar från flera push-pull steg får inte kopplas samman, eftersom det kan leda till kortslutning mellan V_{DD} och GND.

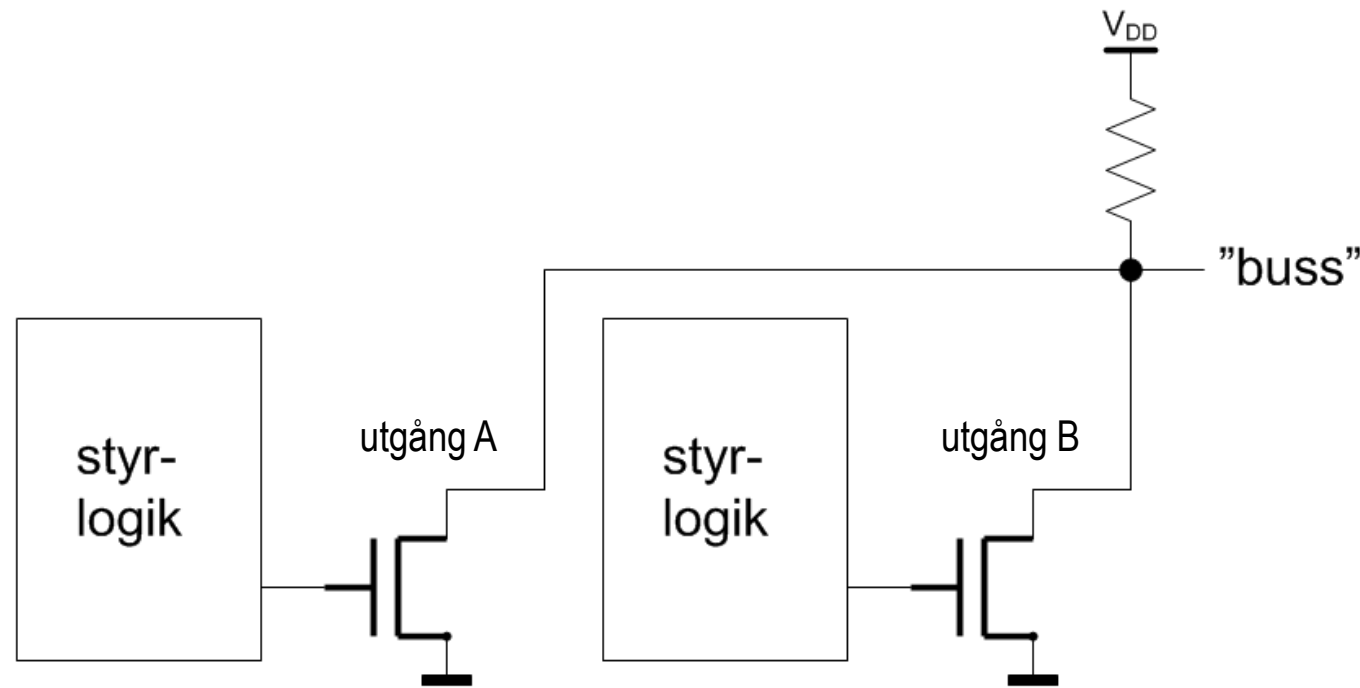


"open drain"

Utgångar från flera "open-drain" steg kan kopplas samman utan problem.

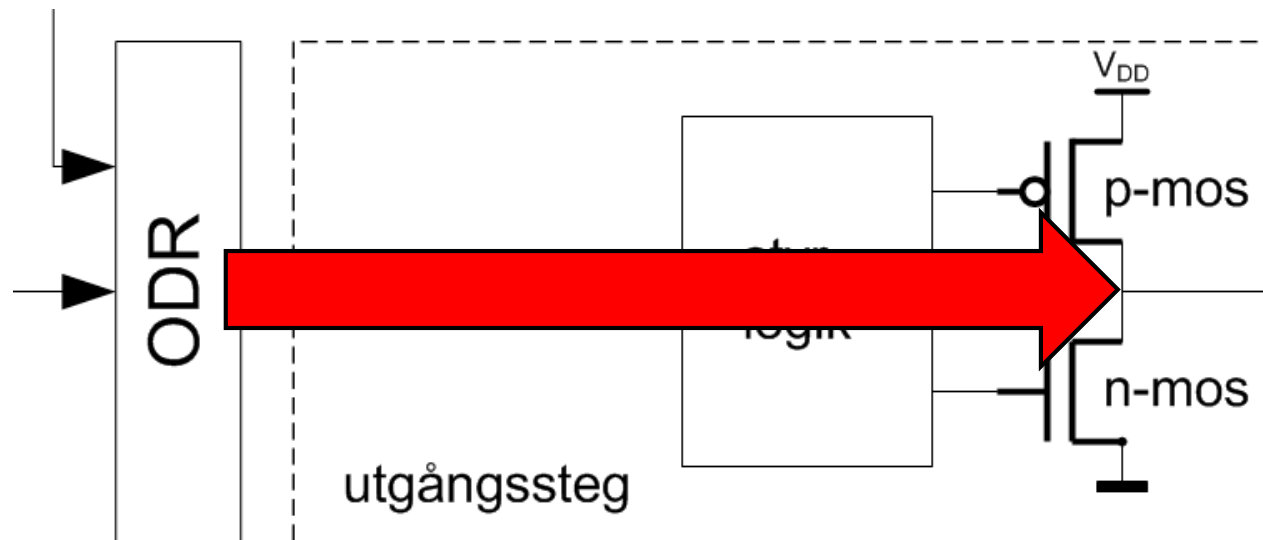
Nivån på den gemensamma "bussen" bestäms av att:

- Om alla utgångar är '1' är också bussnivån '1'
- Om någon utgång är '0' är också bussnivån '0'



"output speed"

Bestämmer hur ofta registrets innehåll överförs till utgångssteget.
Ju lägre frekvens desto mindre strömförbrukning.



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

- 00: Low speed, 2 MHz
- 01: Medium speed, 25 MHz
- 10: Fast speed, 50MHz
- 11: High speed, 100 MHz

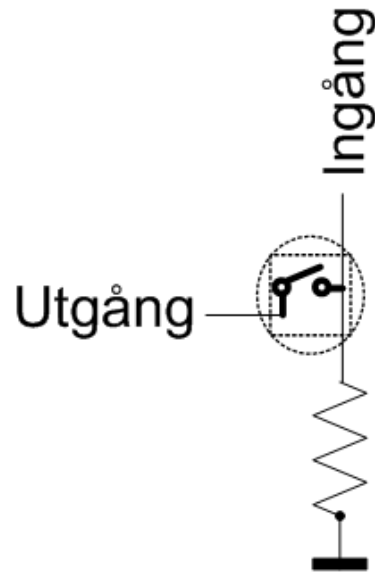
Om hastigheten är 50 MHz eller mer måste en så kallad "IO-kompensationscell" aktiveras

GPIO Output SPEED Register

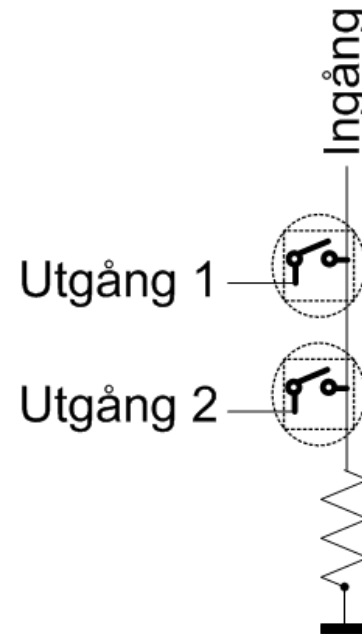
offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x08	pin15	pin14	pin13	pin12	pin11	pin10	pin9	pin8	pin7	pin6	pin5	pin4	pin3	pin2	pin1	pin0											GPIO_OSPEEDR						
	0x0B				0x0A				0x09				0x08																				

Multiplex funktion

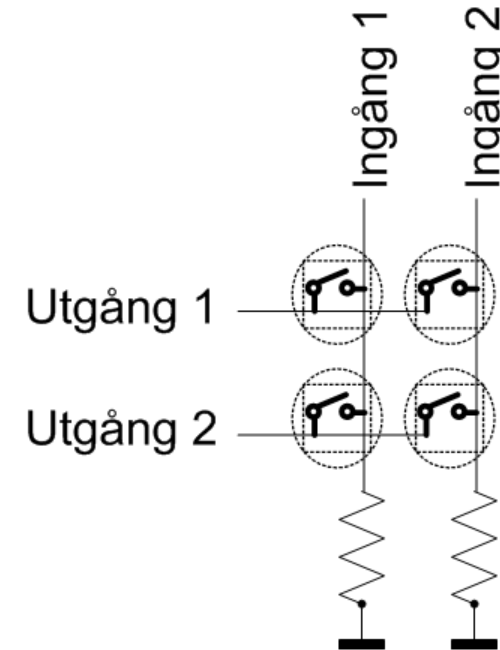
Använd samma ingång för flera olika funktioner.



1 funktion
2 anslutningar



2 funktioner
3 anslutningar



4 funktioner
4 anslutningar

keyb – Rutin för avsökning av tangentbord

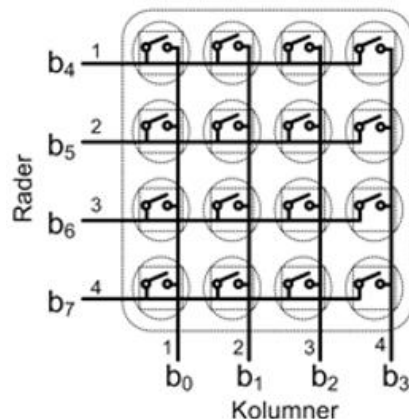
Exempel:

Konstruera en funktion:

```
char keyb( void );
```

Funktionen ska avsöka tangentbordet en gång.

- Om ingen tangent är nedtryckt ska funktionen returnera värdet 0xFF.
- Om någon tangent är nedtryckt ska dess tangentkod returneras.
- Om flera tangenter är nedtryckta är valet av tangentkod, bland dessa, godtyckligt.



En algoritm för tangentbordsfunktionen kan se ut som:

Algoritm keyb:

```
for row = 1..4
  ActivateRow( row );
  column = ReadColumn;
  if column != 0
    keyb = keyValue [pressed key ];
keyb = 0xFF;
```

Avsök tangentbord (1)

Vi förutsätter nu att Port D (8-15) är korrekt initierad

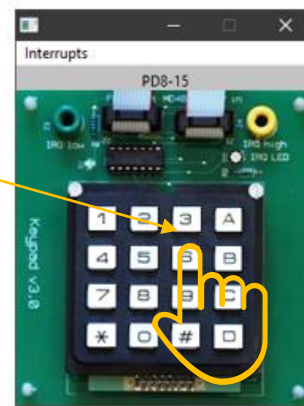
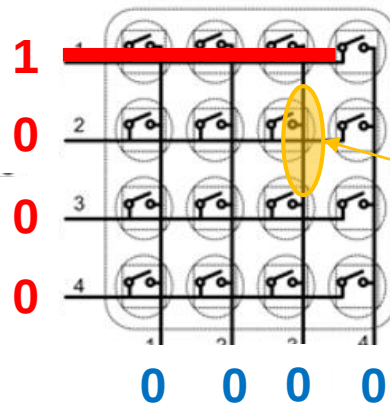
offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x10																		0	0	0	0	0	0	0	0		0	0	0	0	0	0	GPIO_IDR
																		0x14										0x10					



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																		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	GPIO_ODR
																		0x5										0x14					

0x0000

0x1000



En algoritm för tangentbordsfunktionen kan se ut som:

Algoritm keyb:

for row = 1..4

 ActivateRow(row);

 column = ReadColumn;

 if column != 0

 keyb = keyValue [pressed key];

keyb = 0xFF;

Avsök tangentbord (2)

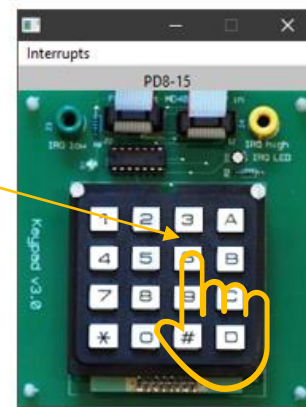
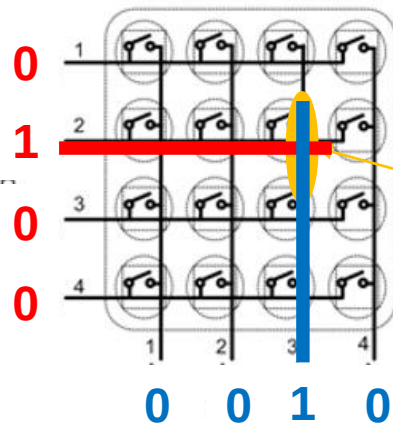
Vi förutsätter nu att Port D (8-15) är korrekt initierad

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																	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	GPIO_IDR

→ 0x0400

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																	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	GPIO_ODR

← 0x2000



En algoritm för tangentbordsfunktionen kan se ut som:

Algoritm keyb:

for row = 1..4

 ActivateRow(row);

 column = ReadColumn;

 if column != 0

 keyb = keyValue [pressed key];

keyb = 0xFF;

Hjälpfunktioner: kbdActivate och kbdGetCol

keyb – Rutin för avsökning av tangentbord

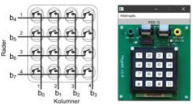
Exempel:

Konstruera en funktion:

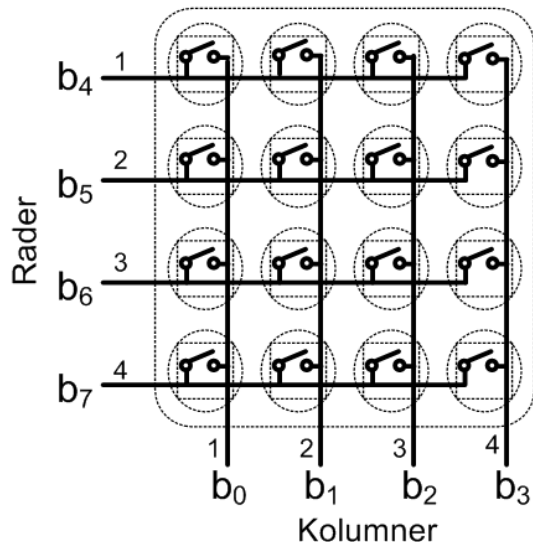
```
char keyb( void );
```

Funktionen ska avsöka tangentbordet en gång.

- Om ingen tangent är nedtryckt ska funktionen returnera värdet 0xFF.
- Om någon tangent är nedtryckt ska dess tangentkod returneras.
- Om flera tangenter är nedtryckta är valet av tangentkod, bland dessa, godtyckligt.



En algoritm för tangentbordsfunktionen kan se ut som:
Algoritm keyb:
for row = 1..4
 ActivateRow(row);
 column = ReadColumn();
 if column != 0
 keyb = keyValue [pressed key];
 keyb = 0xFF;



```
void kbdActivate( unsigned int row )  
{  
    /* Aktivera angiven rad hos tangentbordet, eller deaktivera samtliga */  
    switch( row )  
    {  
        case 1: *GPIO_ODR_HIGH = 0x10; break;  
        case 2: *GPIO_ODR_HIGH = 0x20; break;  
        case 3: *GPIO_ODR_HIGH = 0x40; break;  
        case 4: *GPIO_ODR_HIGH = 0x80; break;  
        case 0: *GPIO_ODR_HIGH = 0x00; break;  
    }  
}
```

```
int kbdGetCol ( void )  
{  
    /* Om någon tangent (i aktiverad rad) är nedtryckt, returnera dess kolumnnummer,  
     * annars, returnera 0 */  
    unsigned char c = *GPIO_IDR_HIGH;  
    if ( c & 0x8 ) return 4;  
    if ( c & 0x4 ) return 3;  
    if ( c & 0x2 ) return 2;  
    if ( c & 0x1 ) return 1;  
    return 0;  
}
```

Funktionen keyb

keyb – Rutin för avsökning av tangentbord

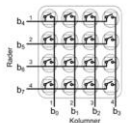
Exempel:

Konstruera en funktion:

```
char keyb( void );
```

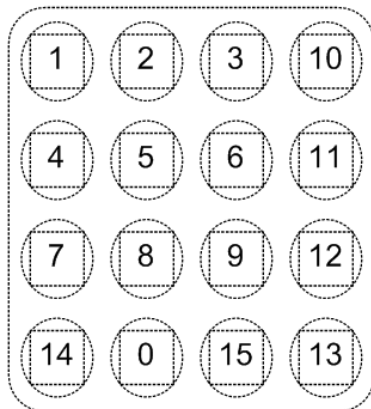
Funktionen ska avsöka tangentbordet en gång.

- Om ingen tangent är nedtryckt ska funktionen returnera värdet 0xFF.
- Om någon tangent är nedtryckt ska dess tangentkod returneras.
- Om flera tangenter är nedtryckta är valet av tangentkod, bland dessa, godtyckligt.



En algoritm för tangentbordsfunktionen kan se ut som:

```
Algoritm keyb:  
for row = 1..4  
  ActivateRow( row );  
  column = ReadColumn;  
  if column != 0  
    keyb = keyValue [pressed key];  
  keyb = 0xFF;
```



```
unsigned char keyb(void)  
{  
  unsigned char key[]={1,2,3,0xA,4,5,6,0xB,7,8,9,0xC,0xE,0,0xF,0xD};  
  int row, col;  
  for (row=1; row <=4 ; row++ ) {  
    kbdActivate( row );  
    if( (col = kbdGetCol () ) )  
    {  
      kbdActivate( 0 );  
      return key [4*(row-1)+(col-1) ];  
    }  
  }  
  kbdActivate( 0 );  
  return 0xFF;  
}
```

GPIO – Algoritm för initiering

keyb – Rutin för avsökning av tangentbord

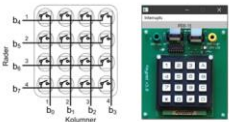
Exempel:

Konstruera en funktion:

```
char keyb( void );
```

Funktionen ska avsöka tangentbordet en gång.

- Om ingen tangent är nedtryckt ska funktionen returnera värdet 0xFF.
- Om någon tangent är nedtryckt ska dess tangentkod returneras.
- Om flera tangenter är nedtryckta är valet av tangentkod, bland dessa, godtyckligt.



En algoritm för tangentbordsfunktionen kan se ut som:
Algoritm keyb:
for row = 1..4
 ActivateRow(row);
 column = ReadColumn;
 if column != 0
 keyb = keyValue [pressed key];
keyb = 0xFF;

Initiering av GPIO-port

SetPortMODER:

```
if( mode_have_input_pins )  
    SetPUPDregister
```

```
if( mode_has_output_pins )  
    SetOTYPER  
    SetOSPEEDR
```

```
if( all_port_pins )  
    *GPIO_xxx = ...
```

```
else (only_specific_port_pins )  
    *GPIO &= ~ specific_port_pins //Clear all specific port pins  
    *GPIO |= specific_port_pins  // set according
```

BSRR – synkroniserad uppdatering

Kan användas för en synkroniserad ändring av flera utgångars pinnar.

bit 0..15: bit reset

bit 16-31: bit set.

BSRR (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	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0	GPIO_BSRR
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0	

"Bitlock" funktion

Då porten har programmerats kan konfigurationen låsas som skydd mot ofrivillig eller otillåten manipulation av portens inställningar. Detta sker genom att en speciell sekvens skrivs till detta register.

offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	mnemonic
0x1C																LCKK	GPIO_LCKR
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0	

Alternativ funktion – ”ruta” interna kretsar

AFRL (alternate function low register)

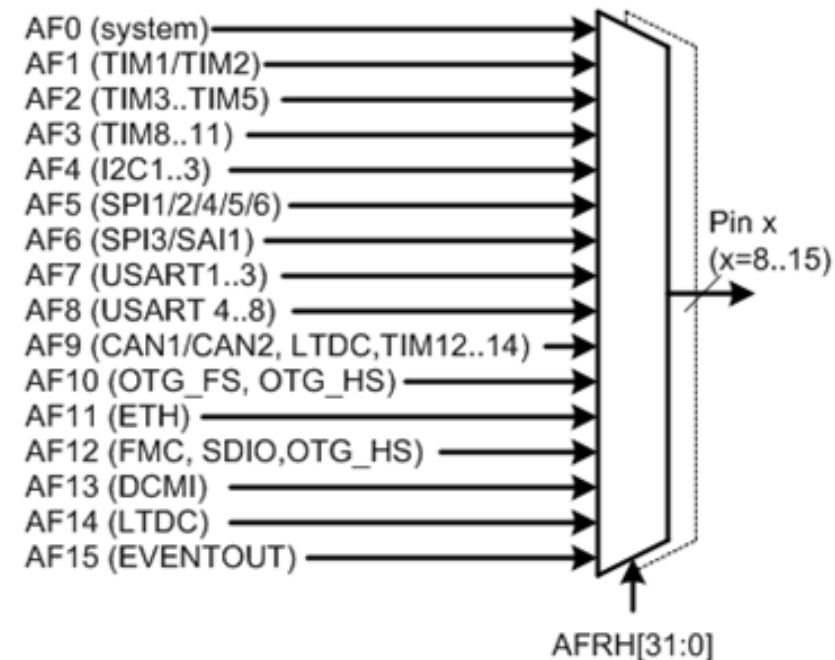
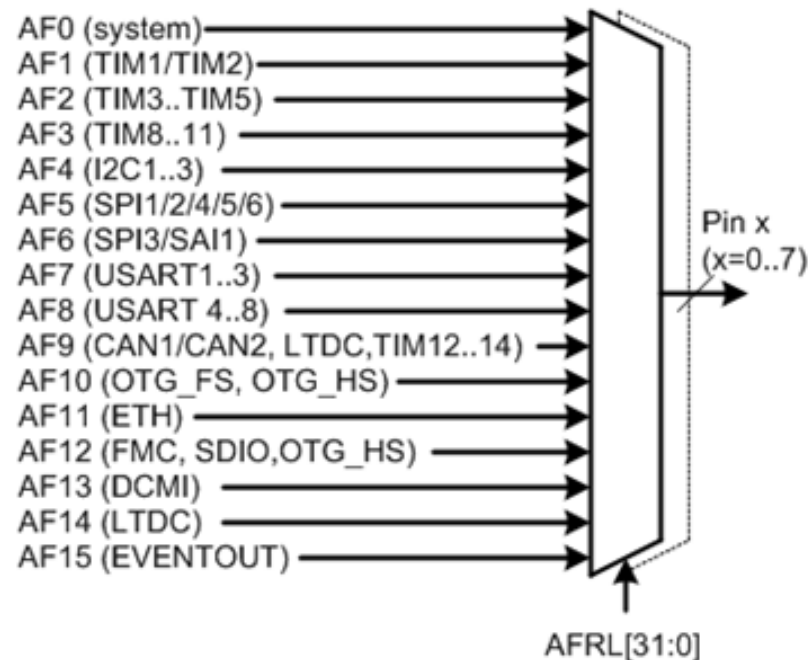
offset	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	mnemonic
0x20	AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]				AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]				GPIO_AFRL

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[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]				AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]				GPIO_AFRH

De 16 pinnarna i en port kan konfigureras individuellt i *mode register* (MODER) för någon av funktionerna:

- 00: digital ingång
- 01: digital utgång
- **10: alternativ funktion**
- 11: analog funktion



Analog funktion

De 16 pinnarna i en port kan konfigureras individuellt i *mode register* (MODER) för någon av funktionerna:

- 00: digital ingång
- 01: digital utgång
- 10: alternativ funktion
- **11: analog funktion**

