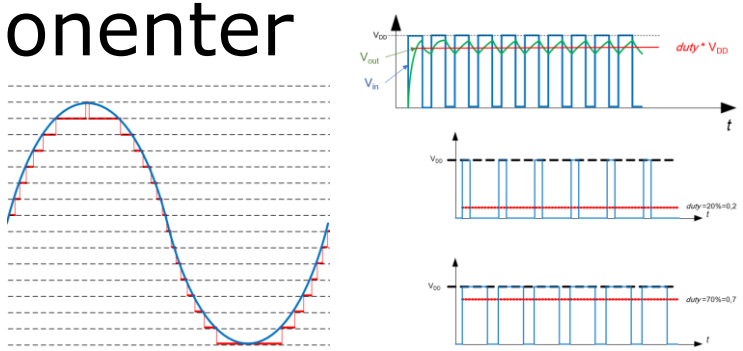
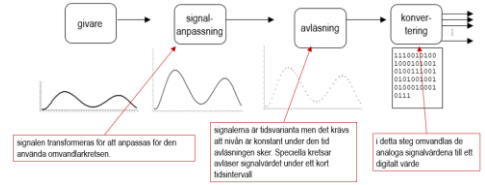
Gränssnitt mellan analoga och digitala komponenter



Ur innehållet

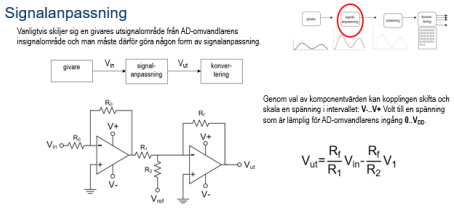
- Omvandling från digital till analog form (DA-omvandling)
  - Resistansstege eller pulsbreddsmodulering

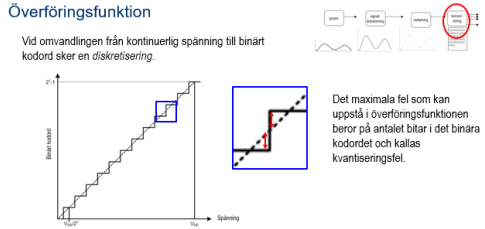
- Omvandling från analog till digital form (AD-omvandling)
  - Givare, signalanpassning och överföringsfunktionen
  - Olika metoder för omvandlingen



Målsättningar:

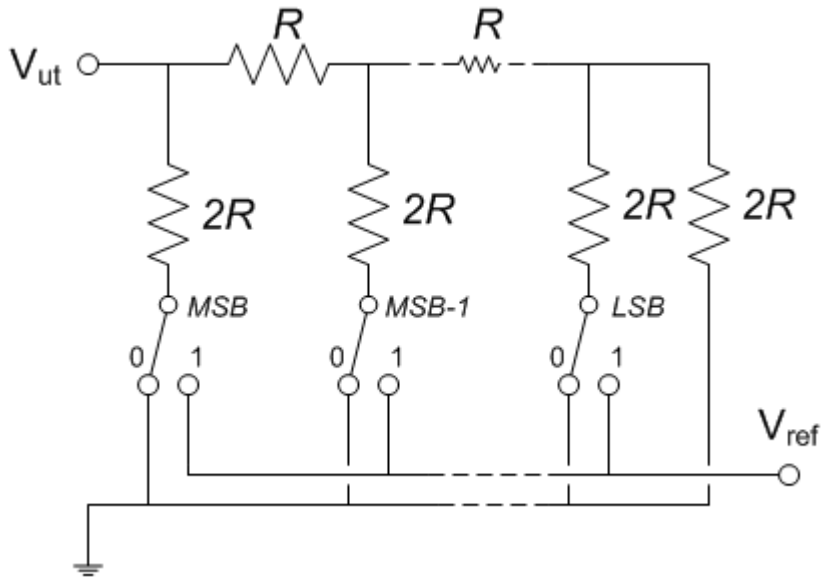
- Kunna redogöra för AD/DA-omvandling



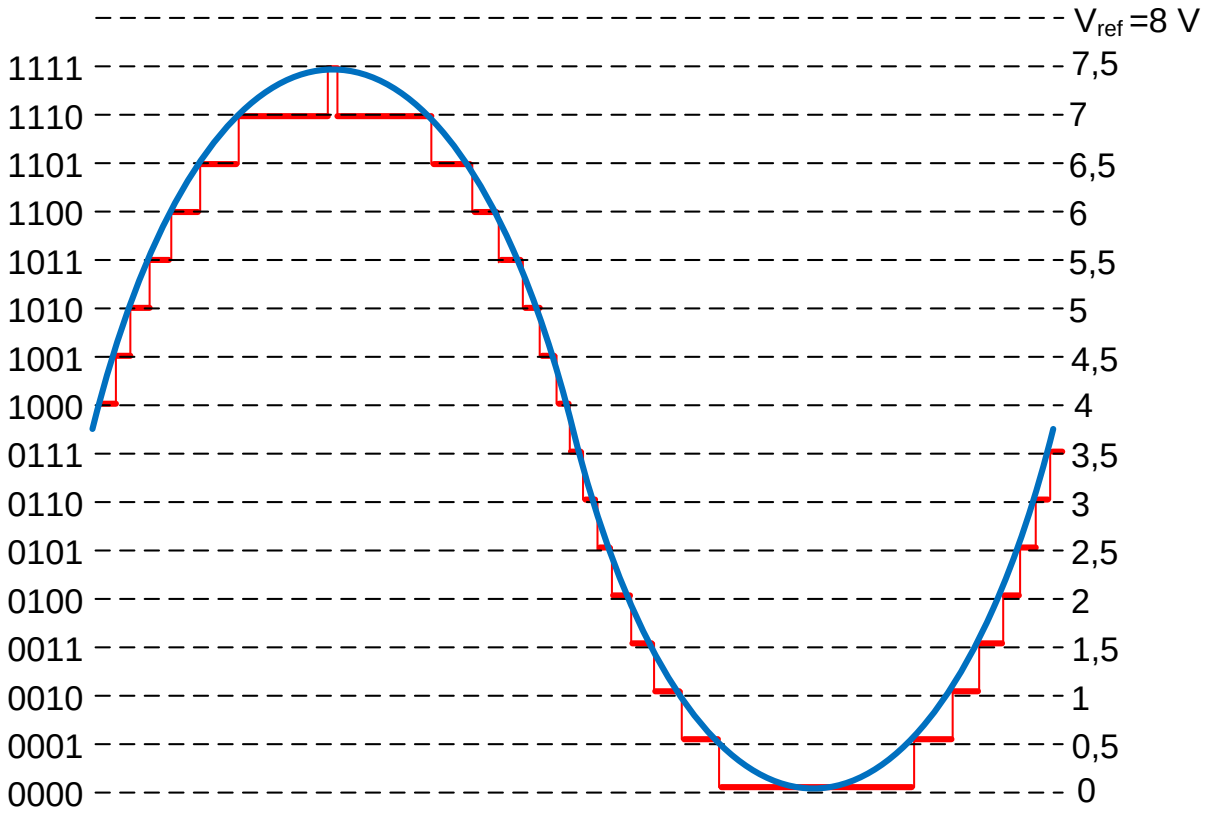


Omvandling av digital representation till elektrisk ström

Exempel: DA-omvandling med R-2R resistansstege.



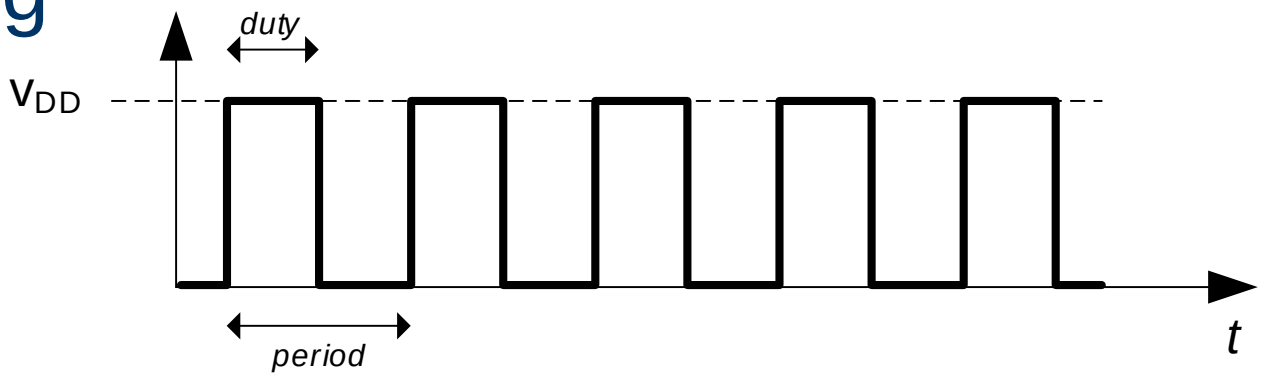
$$X(n):\ V_{ut} = V_{ref}\left(\frac{x_{n-1}}{2} + \frac{x_{n-2}}{4} + \ldots + \frac{x_1}{2^{n-1}} + \frac{x_0}{2^n}\right)$$

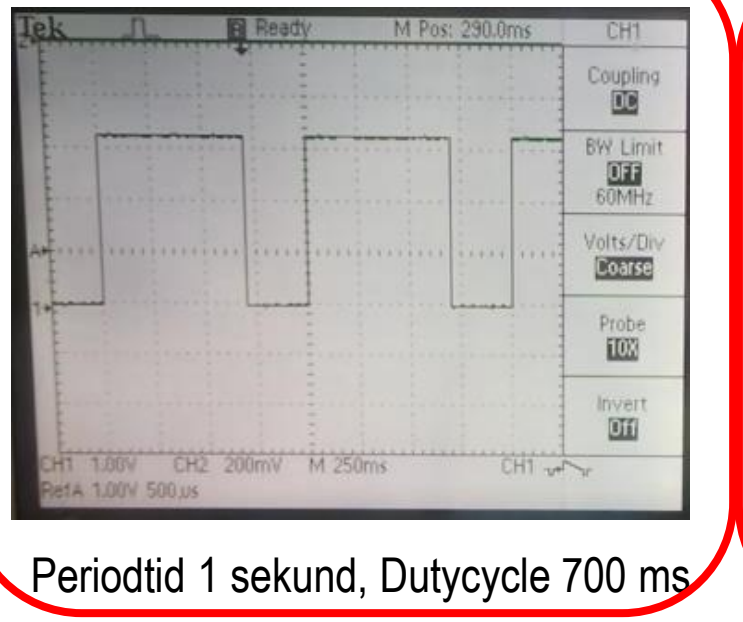


4-bitars DA-omvandlare. V_{ref}=8 Volt

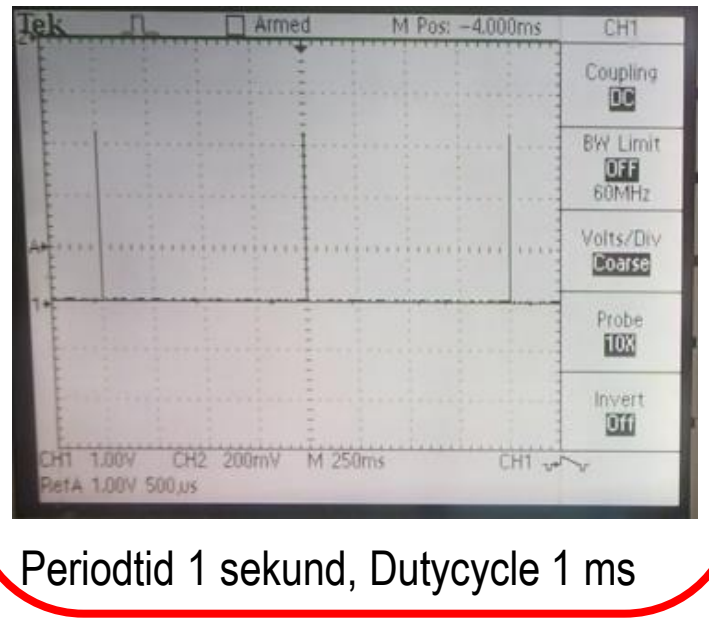
PWM - pulsbreddsmodulering

Karaktäriseras av periodtid och duty cycle, dvs. den del av periodtiden som pulsen är hög.

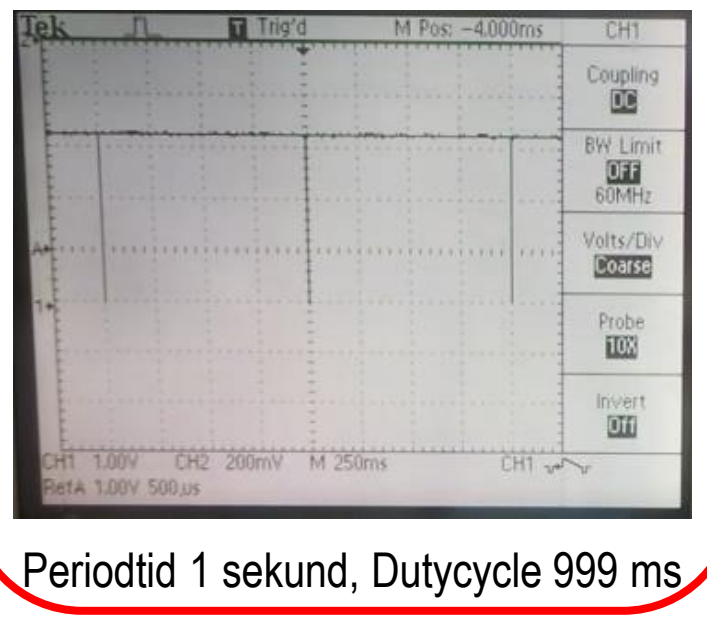




Periodtid 1 sekund, Dutycycle 700 ms



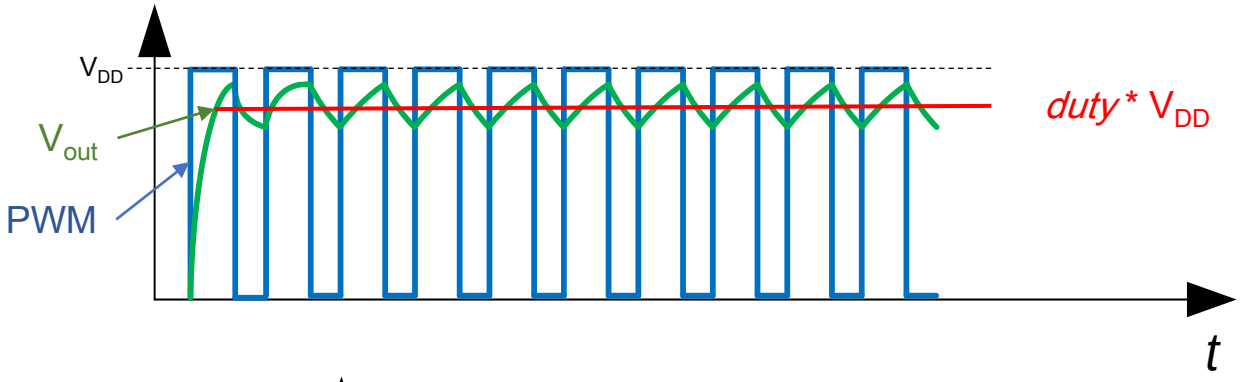
Periodtid 1 sekund, Dutycycle 1 ms

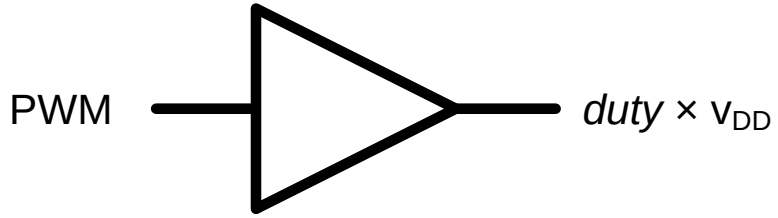


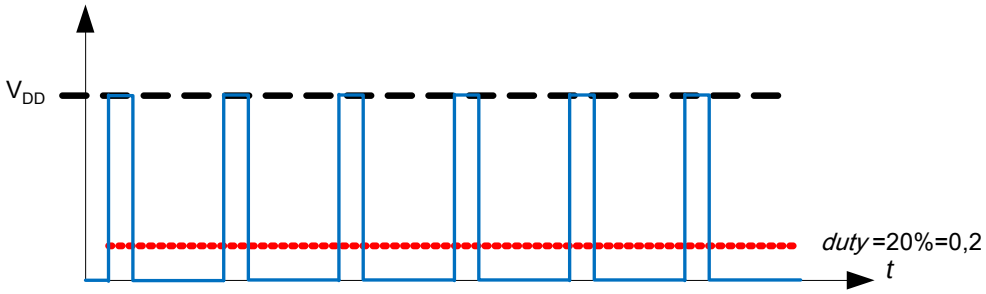
Periodtid 1 sekund, Dutycycle 999 ms

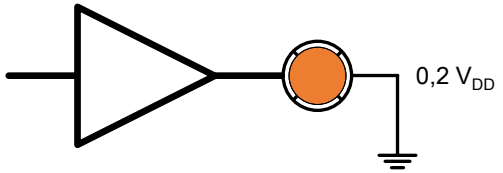
PWM

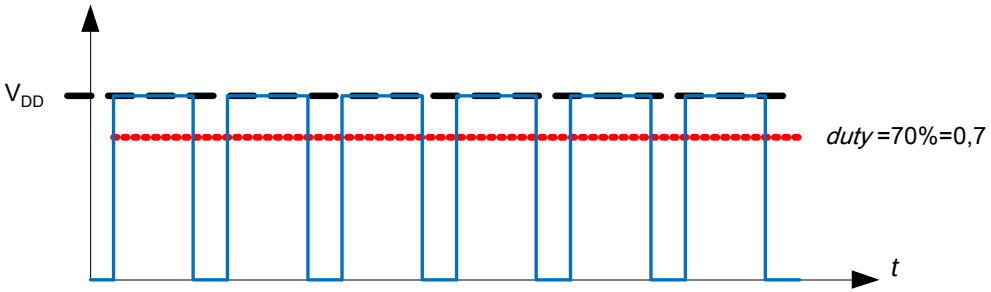
Vid högre frekvenser får man en medelvärdesbildning av effekten

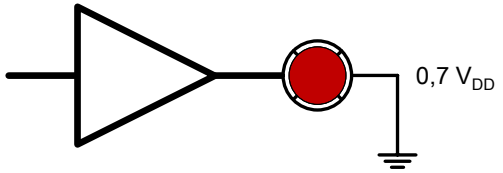












PWM i praktiken

Exempel, PWM med MD407

```
typedef struct tagTimDevice
{
  volatile uint16_t cr1;        /* TIM control register 1,              Address offset
  uint16_t res0;
  volatile uint16_t cr2;        /* TIM control register 2,              Address offset
  uint16_t res1;
  volatile uint16_t smcr;       /* TIM slave mode control register,     Address offset
  uint16_t res2;
  volatile uint16_t dier;       /* TIM DMA/interrupt enable register,   Address offset
  uint16_t res3;
  volatile uint16_t sr;         /* TIM status register,                 Address offset
  uint16_t res4;
  volatile uint16_t egr;        /* TIM event generation register,       Address offset
  uint16_t res5;
  volatile uint16_t ccmr1;      /* TIM capture/compare mode register 1, Address offset: 0x18 */
  uint16_t res6;
  volatile uint16_t ccmr2;      /* TIM capture/compare mode register 2, Address offset: 0x1C */
  uint16_t res7;
  volatile uint16_t ccer;       /* TIM capture/compare enable register, Address offset: 0x20 */
  uint16_t res8;
  volatile uint32_t cnt;        /* TIM counter register,                Address offset: 0x24 */
  volatile uint16_t psc;        /* TIM prescaler,                       Address offset: 0x28 */
  uint16_t res9;
  volatile uint32_t arr;        /* TIM auto-reload register,            Address offset: 0x2C */
  volatile uint16_t rcr;        /* TIM repetition counter register,     Address offset: 0x30 */
  uint16_t res10;
  volatile uint32_t ccr1;       /* TIM capture/compare register 1,      Address offset: 0x34 */
  volatile uint32_t ccr2;       /* TIM capture/compare register 2,      Address offset: 0x38 */
  volatile uint32_t ccr3;       /* TIM capture/compare register 3,      Address offset: 0x3C */
  volatile uint32_t ccr4;       /* TIM capture/compare register 4,      Address offset: 0x40 */
  volatile uint16_t bdtr;       /* TIM break and dead-time register,    Address offset: 0x44 */
  uint16_t res11;
  volatile uint16_t dcr;        /* TIM DMA control register,            Address offset: 0x48 */
  uint16_t res12;
  volatile uint16_t dmar;       /* TIM DMA address for full transfer,   Address offset: 0x4C */
  uint16_t res13;
  volatile uint16_t or;         /* TIM option register,                 Address offset: 0x50 */
  uint16_t res14;
} TIM_ARM_DEVICE, *PTIM_ARM_DEVICE;

#define TIM4_BASE 0x40000800
#define TIM4 ((PTIM_ARM_DEVICE) TIM4_BASE)
```

```
void init_app( void )
{
  RCC->apb1enr |= (1<<2); /* Aktivera klocka för TIM4 */
  RCC->apb1rstr |= (1<<2); /* Återställ TIM4 */
  RCC->apb1rstr &= ~(1<<2);

 /* Använd AF och koppla utgång från TIM4 till PD12 */
  GPIOD->moder |= 0x56550000;   /* Pin 12 är AF, övriga PD8-15 digital ut */
  GPIOD->ospeedr |= 0x02000000; /* 50 MHz */
  GPIOD->afrh |= 0x20000;       /* Anslut TIM4 till AF */
}
```

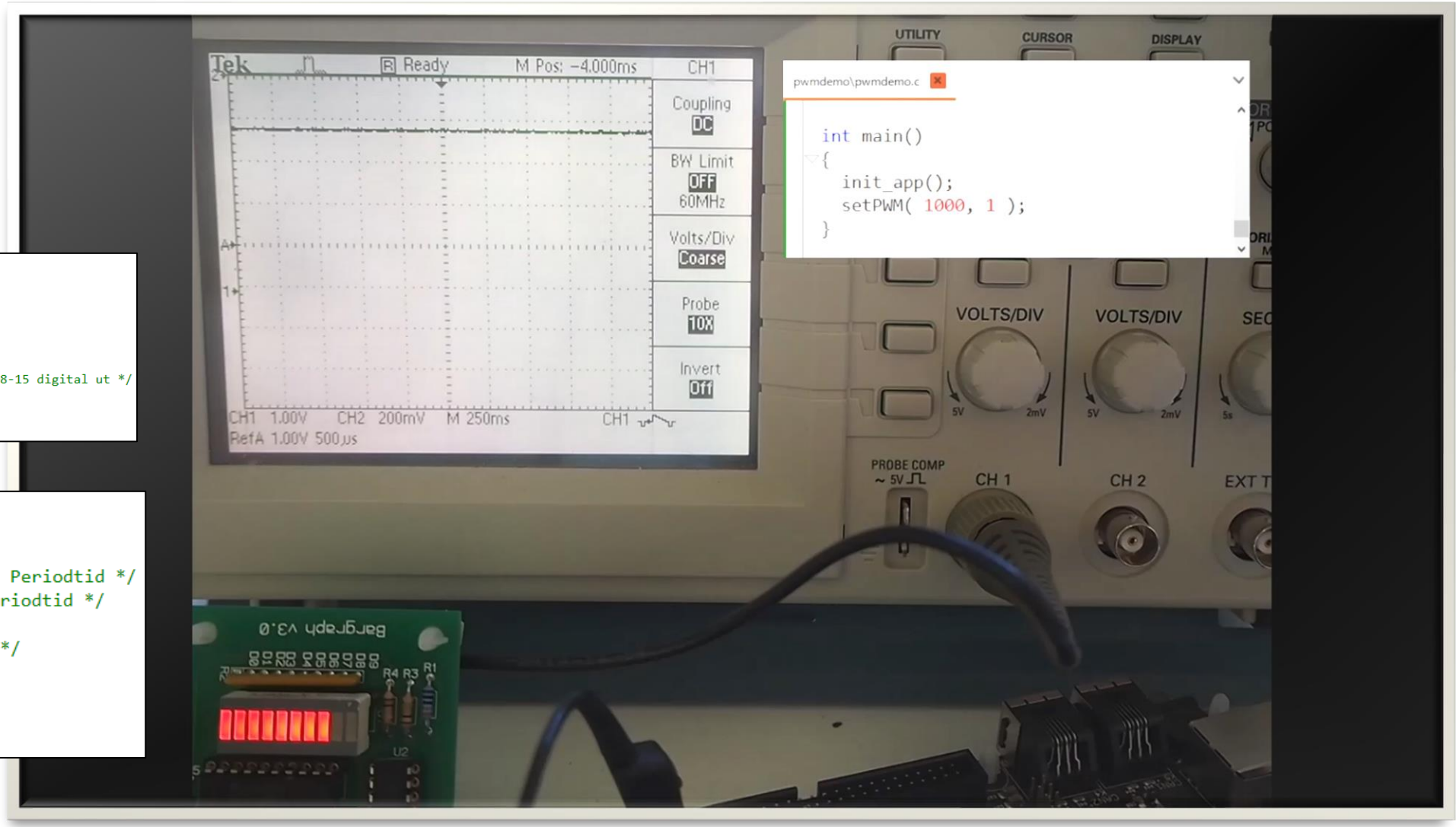
```
void setPWM( int period, int duty )
{
  TIM4->cr1 = 0; /* Räknare, mod etc.  */
  TIM4->arr = (unsigned short) period*2; /* Periodtid */
  TIM4->psc = 42000; /* Prescaler 0,5 ms periodtid */
  TIM4->cr1 = 0x81;  /* Aktivera TIM4 */
  TIM4->ccer &= ~1;  /* Avaktivera kanal 1 */
  TIM4->ccr1 = (unsigned short) duty*2;
  TIM4->ccmr1 = 0x60; /* PWM1 mod */
  TIM4->ccer = 1; /* Aktivera kanalen */
}
```

PWM i praktiken: med MD407

```
int main()
{
  init_app();
  setPWM( period, duty );
}
```

```
void init_app( void )
{
  RCC->apb1enr |= (1<<2); /* Aktivera klocka för TIM4 */
  RCC->apb1rstr |= (1<<2); /* Återställ TIM4 */
  RCC->apb1rstr &= ~(1<<2);

 /* Använd AF och koppla utgång från TIM4 till PD12 */
  GPIOD->moder |= 0x56550000;   /* Pin 12 är AF, övriga PD8-15 digital ut */
  GPIOD->ospeedr |= 0x02000000; /* 50 MHz */
  GPIOD->afrh |= 0x20000;       /* Anslut TIM4 till AF */
}
```

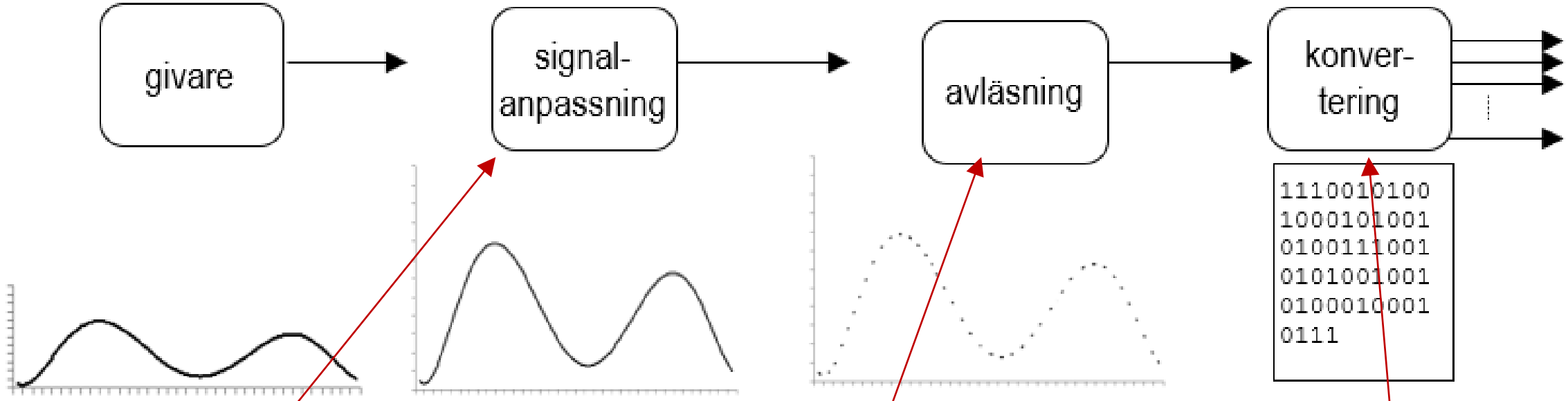
```
void setPWM( int period, int duty )
{
  TIM4->cr1 = 0; /* Räknare, mod etc.  */
  TIM4->arr = (unsigned short) period*2; /* Periodtid */
  TIM4->psc = 42000; /* Prescaler 0,5 ms periodtid */
  TIM4->cr1 = 0x81;  /* Aktivera TIM4 */
  TIM4->ccer &= ~1;  /* Avaktivera kanal 1 */
  TIM4->ccr1 = (unsigned short) duty*2;
  TIM4->ccmr1 = 0x60; /* PWM1 mod */
  TIM4->ccer = 1; /* Aktivera kanalen */
}
```



PWM i praktiken: *pulserande diod*

```
int main()
{
  int i;

  init_app();
  while (1)
  {
    for( i = 0; i <= 10; i++){
      setPWM( 10, i );
      delay_ms( 200 );
    }
    for( i = 9; i >= 0; i--){
      setPWM( 10, i );
      delay_ms( 200 );
    }
  delay_ms( 500 );
  }
}
```

Omvandling av fysikalisk storhet till digital representation



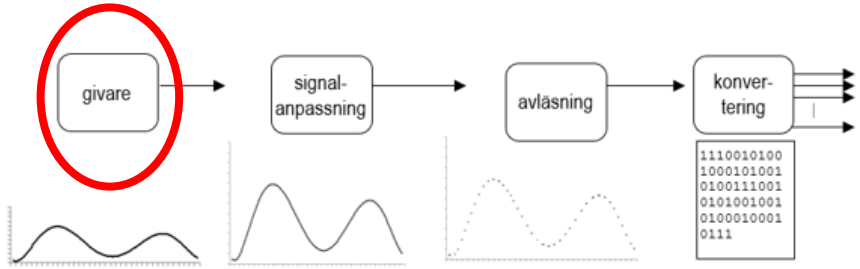
signalen transformeras för att anpassas för den använda omvandlarkretsen.

signalerna är tidsvarianta men det krävs att nivån är konstant under den tid avläsningen sker. Speciella kretsar avläser signalvärdet under ett kort tidsintervall

i detta steg omvandlas de analoga signalvärdena till ett digitalt värde

Givare (*sensor*)

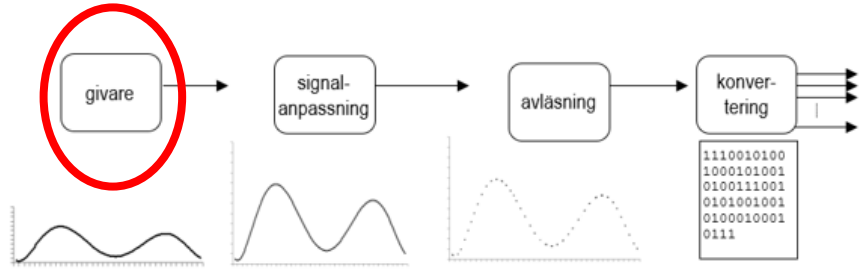
levererar en elektrisk ström (spänning) som ger en korrekt uppfattning av den uppmätta storheten.
Vi säger då att givarens levererade ström (spänning) är analog med den uppmätta storheten.

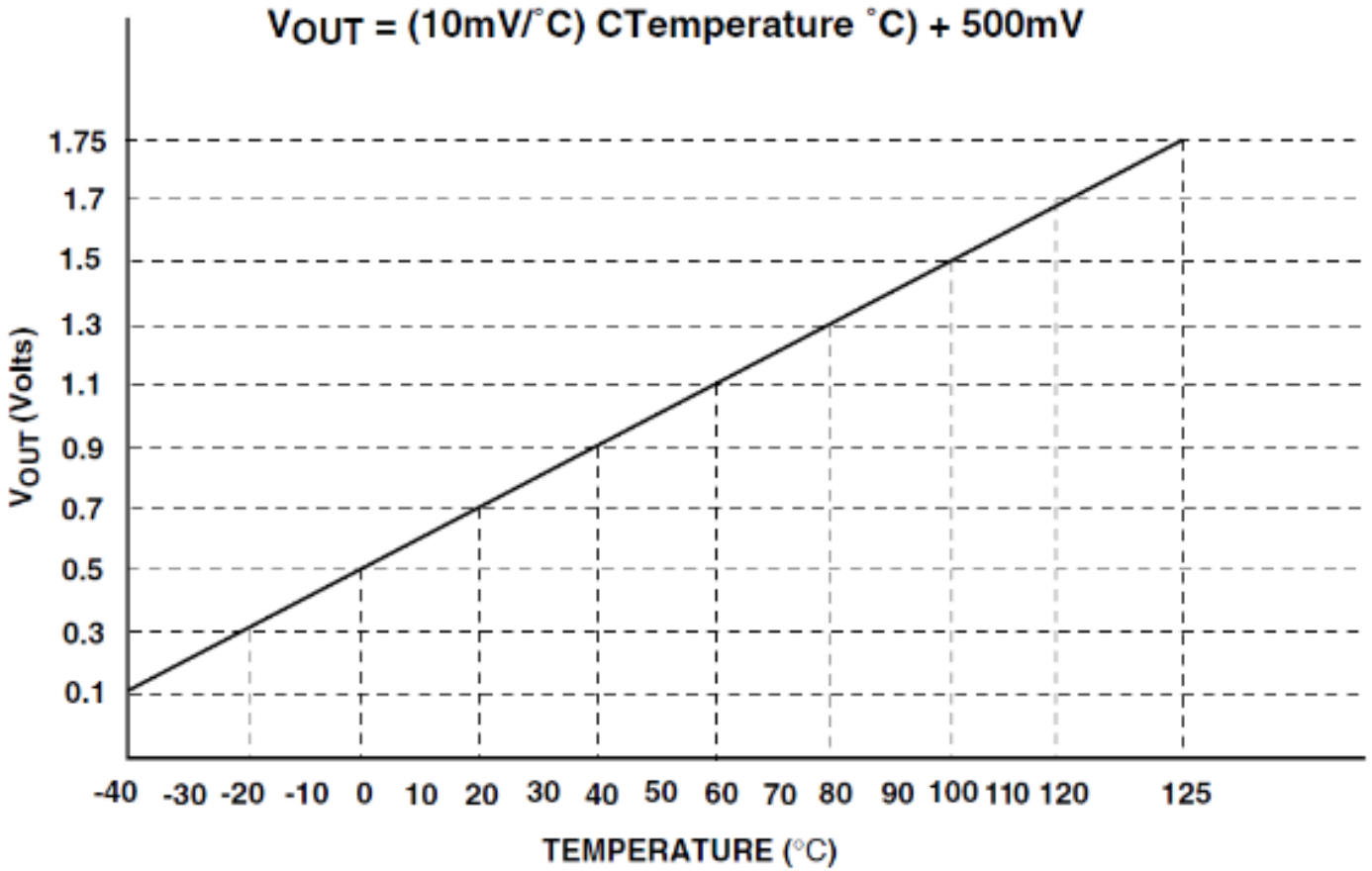


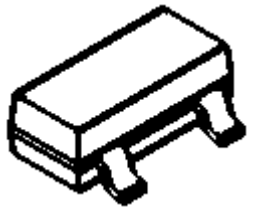
Givare finns för olika fysikaliska storheter, exempel:

- Temperatur
- Töjning
- Gas- och vätsketryck
- Ljus (frekvens) och ljusstyrka
- Ljud (frekvens) och ljudstyrka

Exempel: Temperaturgivare

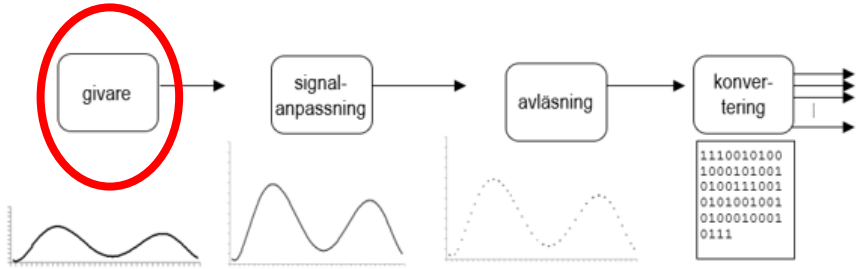




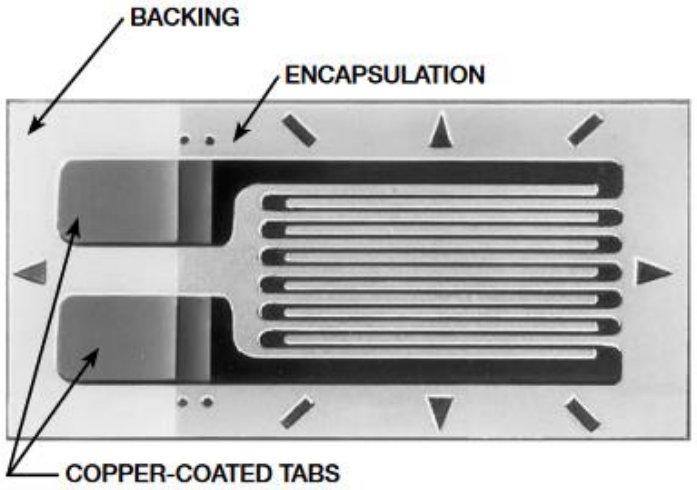


Ett exempel på en krets för temperaturmätning är *Microchips TC 1047*. Kretsen levererar en spänning som är linjär i intervallet -40 – 125 C. Upplösningen är 10mV/C.

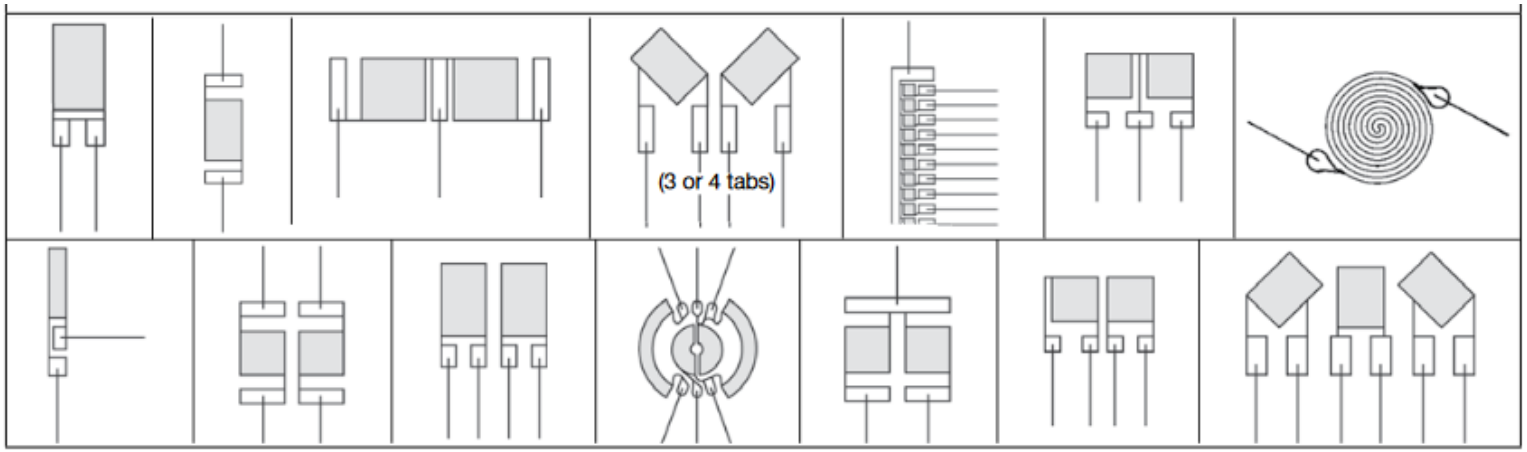
Exempel: Töjningsgivare



När ledaren, i form av en tråd, utsätts för en förlängning eller förkortning ändras resistansen i den. Genom att applicera en spänning över resistansen kan vi då mäta en ström som är analog med töjningen.

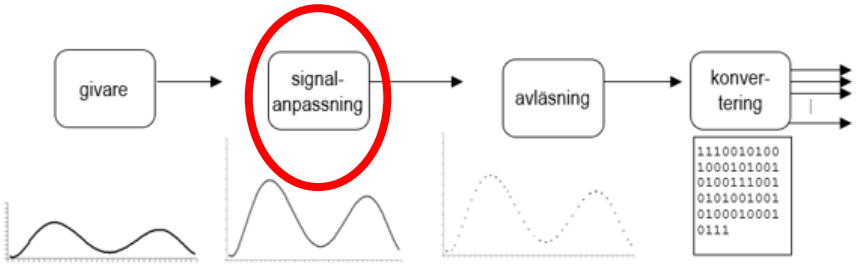


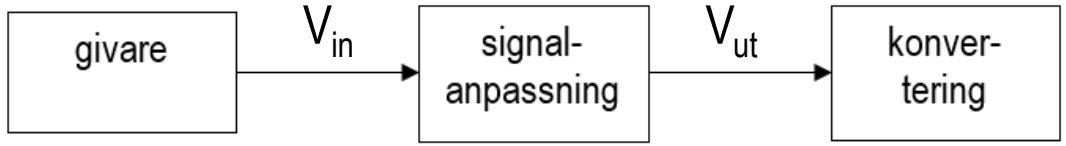
Töjningsgivaren kan anpassas för olika geometrier.

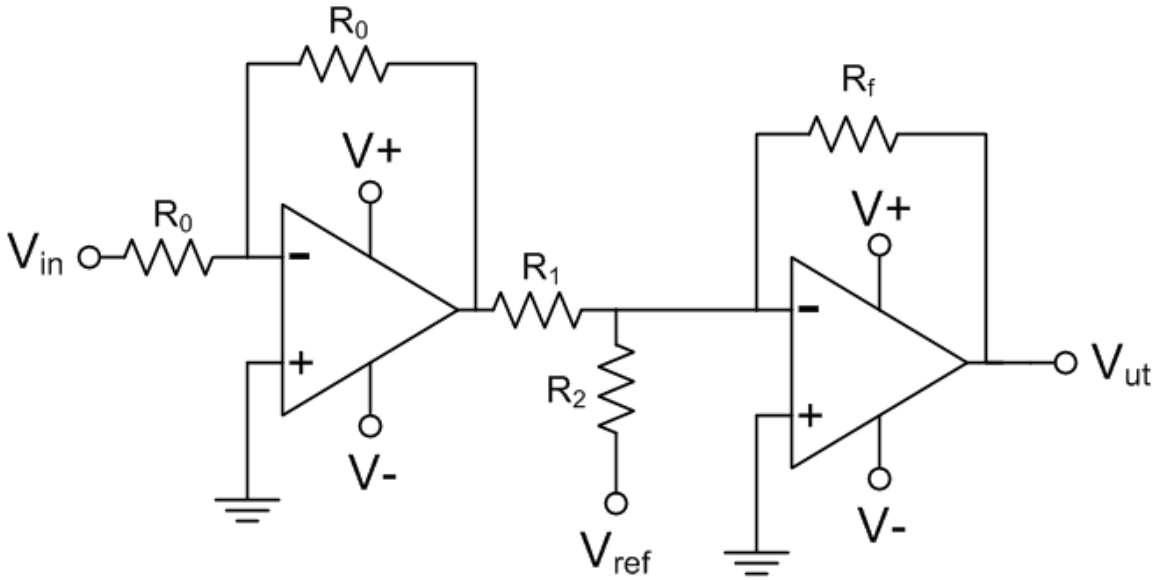


Signalanpassning

Vanligtvis skiljer sig en givares utsignalområde från AD-omvandlarens insignalområde och man måste därför göra någon form av signalanpassning.





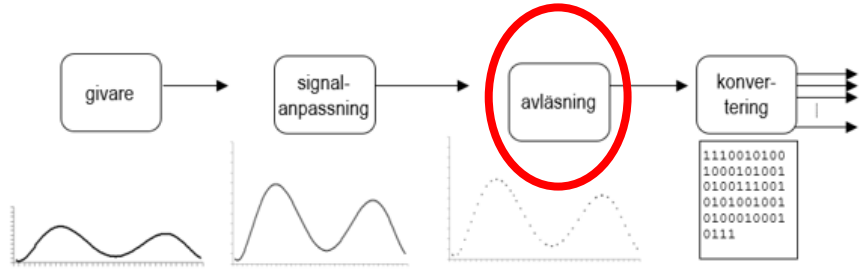


Genom val av komponentvärden kan kopplingen skifta och skala en spänning i intervallet: **V-..V+** Volt till en spänning som är lämplig för AD-omvandlarens ingång **0..V_{DD}**.

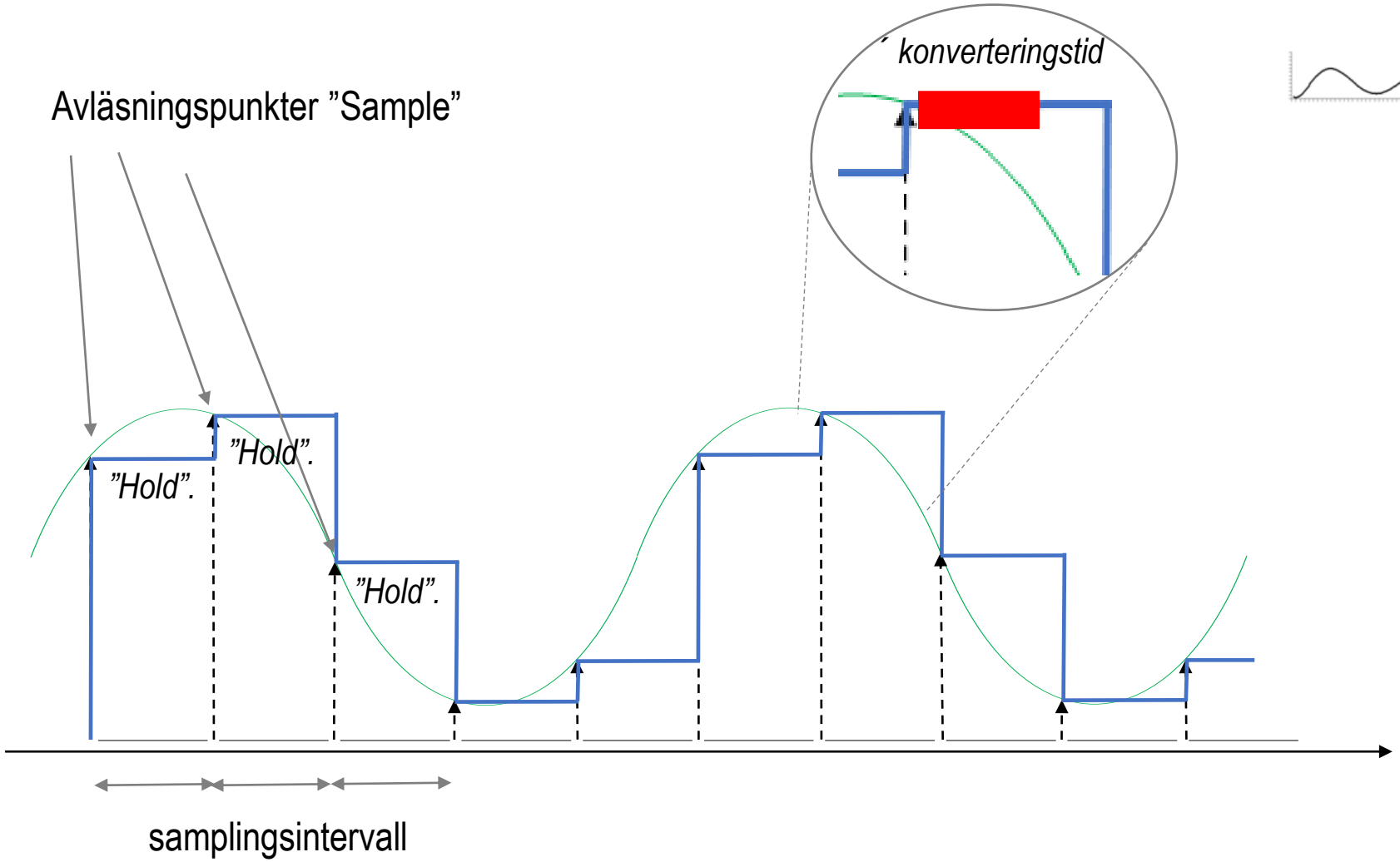
$$V_{ut} = \frac{R_f}{R_1} V_{in} - \frac{R_f}{R_2} V_{REF}$$

Sample and Hold

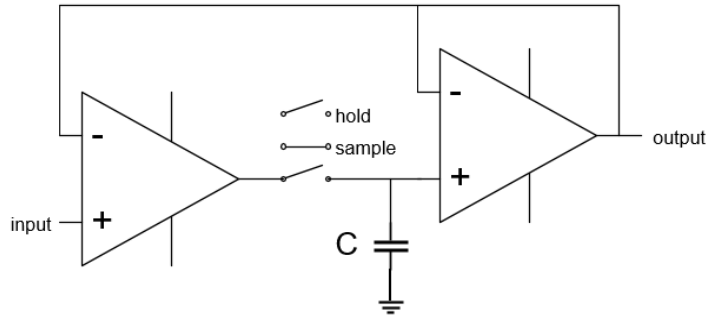
Vi har exempelvis en sinusformad insignal



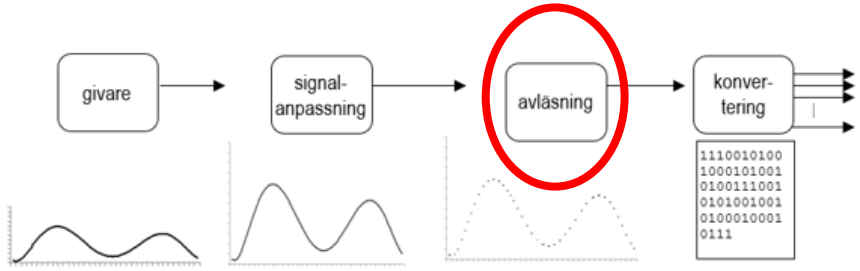
Avläsningspunkter ”Sample”



Principen för *sample and hold* - krets

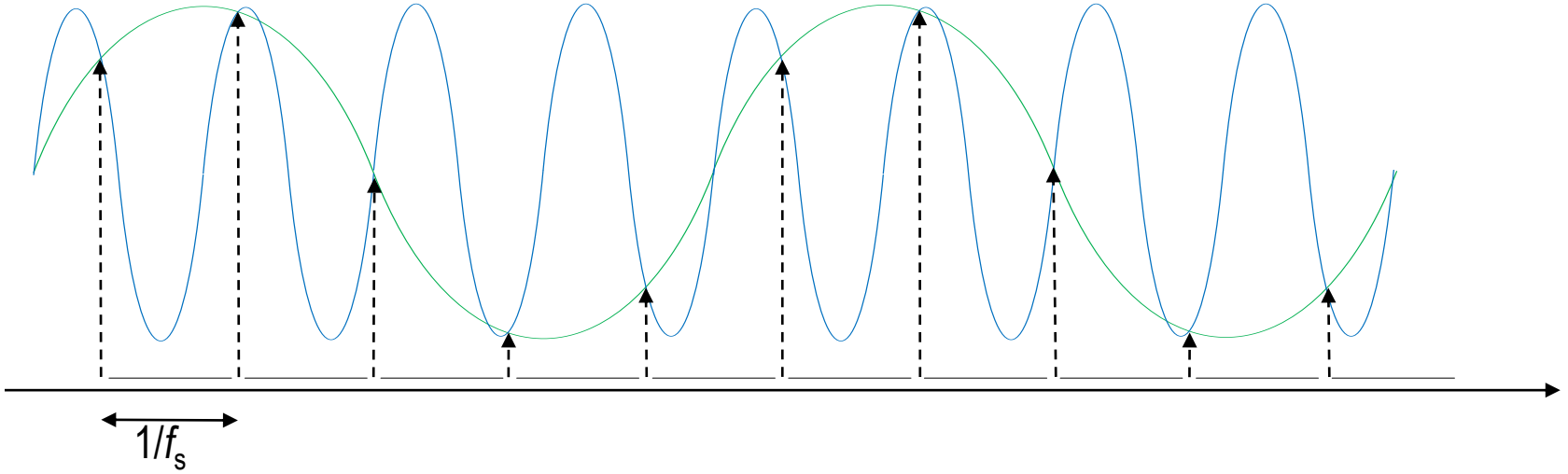


Konverteringstid



Insignalens frekvensomfång (bandbredd) betecknas f_b.

Den maximala samplingsfrekvensen f_s bestäms av omvandlarens snabbhet i konverteringen.

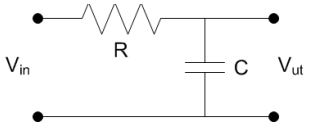


Signaler med högre frekvens kan tolkas felaktigt, fenomet kallas *vikning* (aliasing).

För att undvika detta måste det gälla (Nyqvist.Shannons teorem) att: $f_s \geq 2f_b$

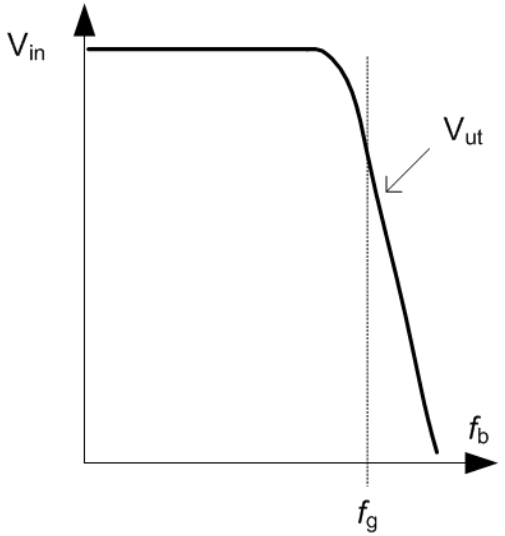
Dvs. samplingsfrekvensen måste vara minst dubbelt så stor som den högsta frekvens som ska mätas.

Antivikningsfilter

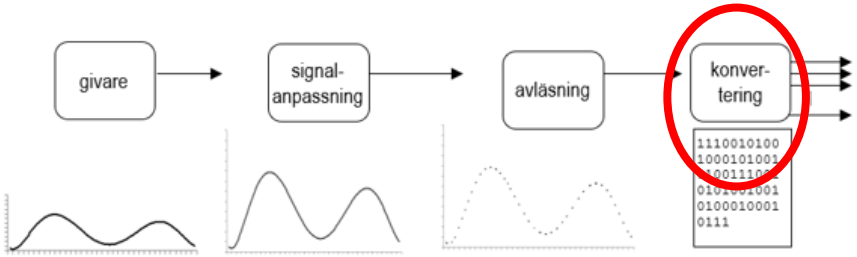


(lågpassfilter)

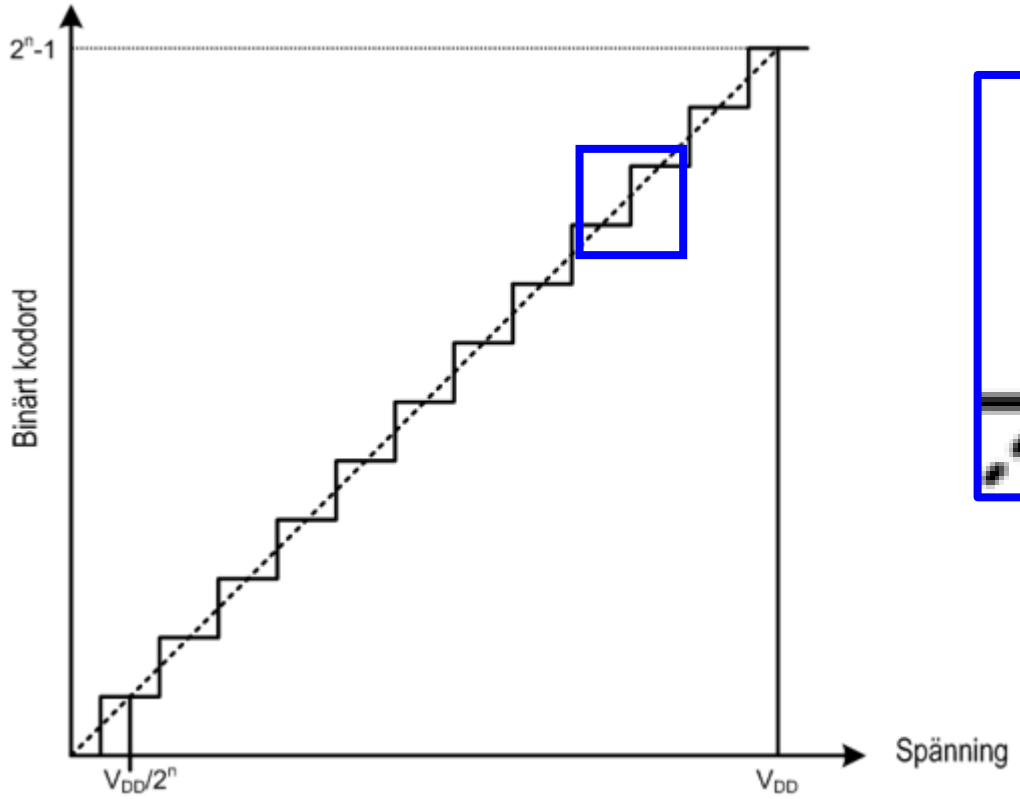
släpper igenom signaler med frekvens som ligger under gränsfrekvens f_g och dämpar övriga signaler.

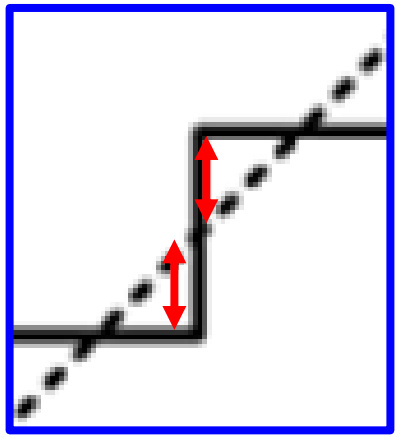


Överföringsfunktion



Vid omvandlingen från kontinuerlig spänning till binärt kodord sker en *diskretisering*.

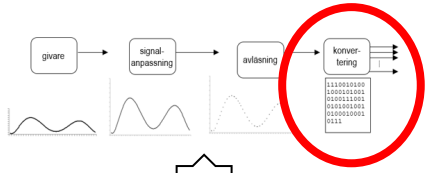




Det maximala fel som kan uppstå i överföringsfunktionen beror på antalet bitar i det binära kodordet och kallas kvantiseringsfel.

Succesiv approximation

Exempel: Princip för omvandling, Vin=2,2 Volt.



1 2 3 4 5

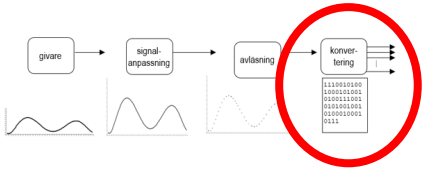
DAC

Succesivt Approximation Register

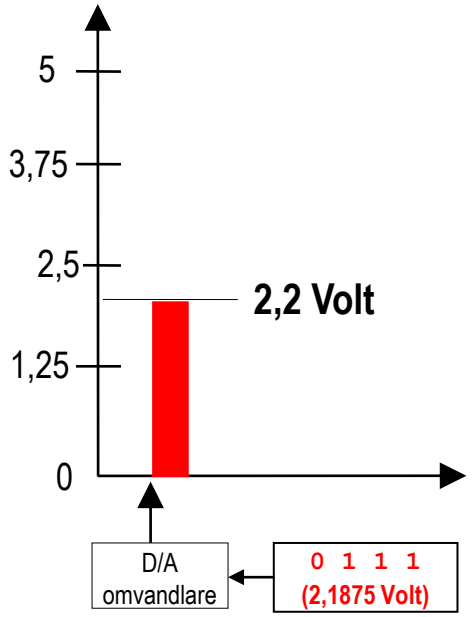
V_{in}

X(n)

Succesiv approximation



Då samtliga bitar prövats har vi alltså resultatet:



De möjliga värdena ges av tabellen:

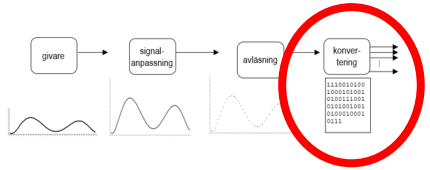
bitmönster	V_{DA} (Volt)
0 0 0 0	0
0 0 0 1	0,3125
0 0 1 0	0,625
0 0 1 1	0.9375
0 1 0 0	1,25
0 1 0 1	1,5625
0 1 1 0	1,875
0 1 1 1	2,1875
1 0 0 0	2,5
1 0 0 1	2,8125
1 0 1 0	3,125
1 0 1 1	3,4375
1 1 0 0	3,75
1 1 0 1	4,0625
1 1 1 0	4,375
1 1 1 1	4,6875

För V_{in}=2,2 Volt

blir då bästa approximationen
V_{DA}=2,1875 Volt
med kvantiseringsfelet

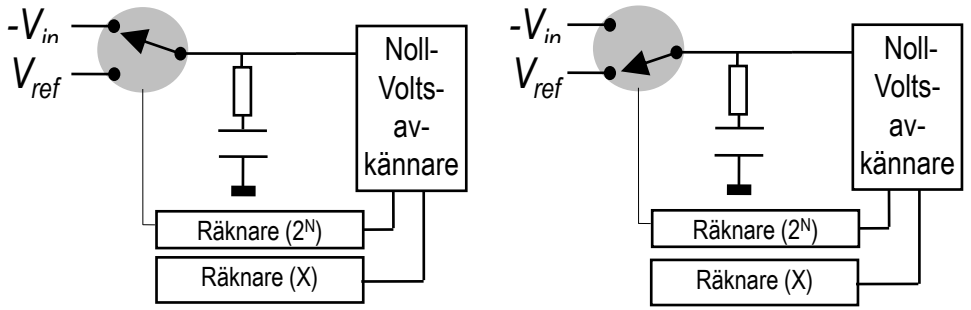
0,0125 Volt.

Rampomvandlare



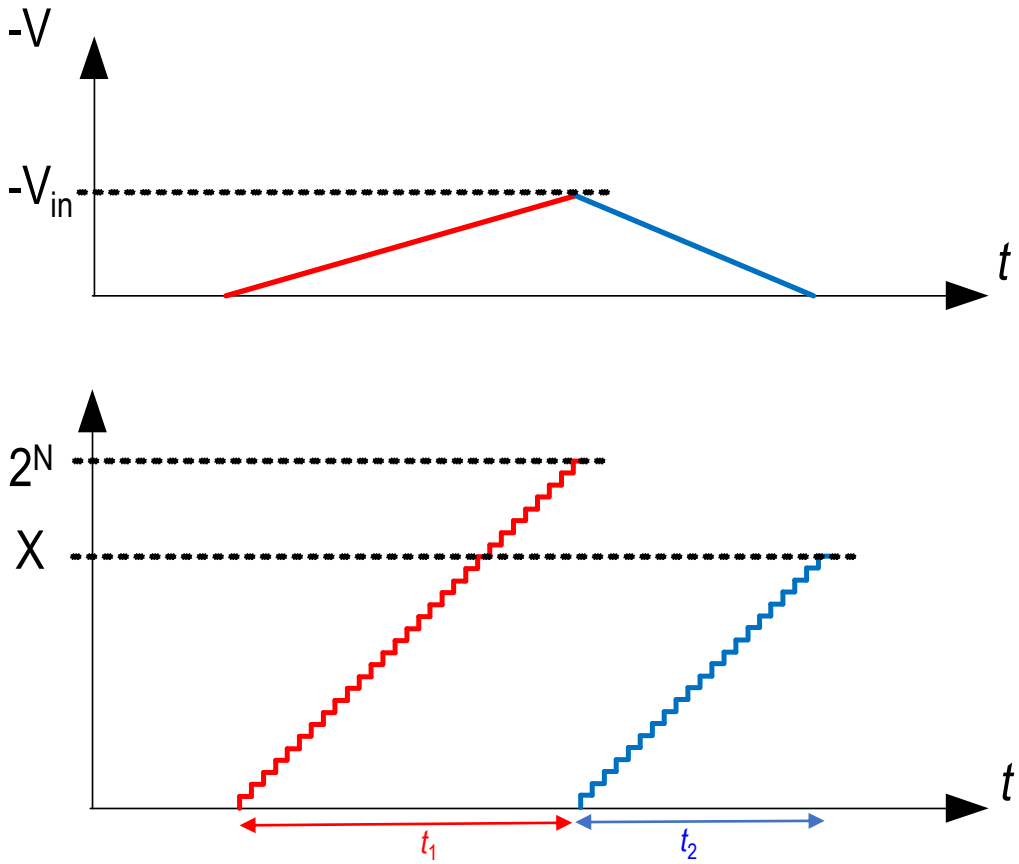
$$V_{in} t_1 = V_{ref} t_2 \qquad V_{in} = V_{ref} \frac{X}{2^N}$$

$$X = \frac{V_{in} 2^N}{V_{ref}}$$

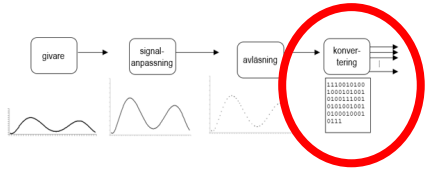


Räknaren räknar upp till 2^N. Under detta tidsintervall t_1 laddas kondensatorn av spänningen som ska mätas

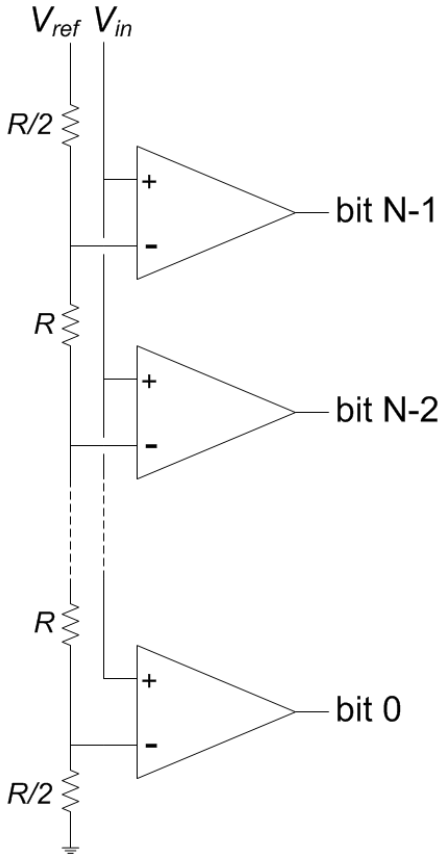
Räknaren startas på nytt, under tidsintervall t_2 laddas kondensatorn ur och en speciell krets känner av då spänningen över kondensatorn nått ursprungsnivån, då stoppas räknaren vid X räknade pulser



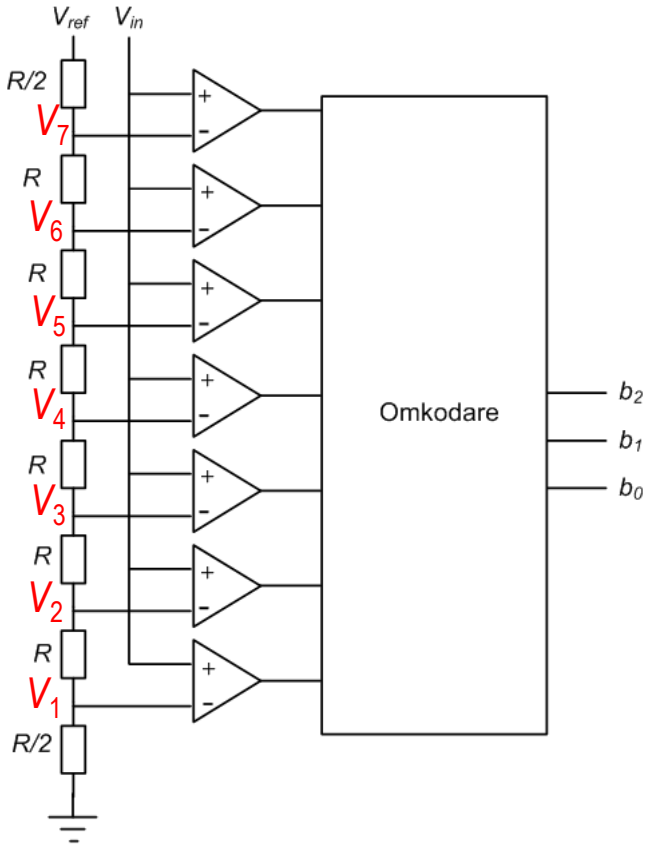
”Flash converter”

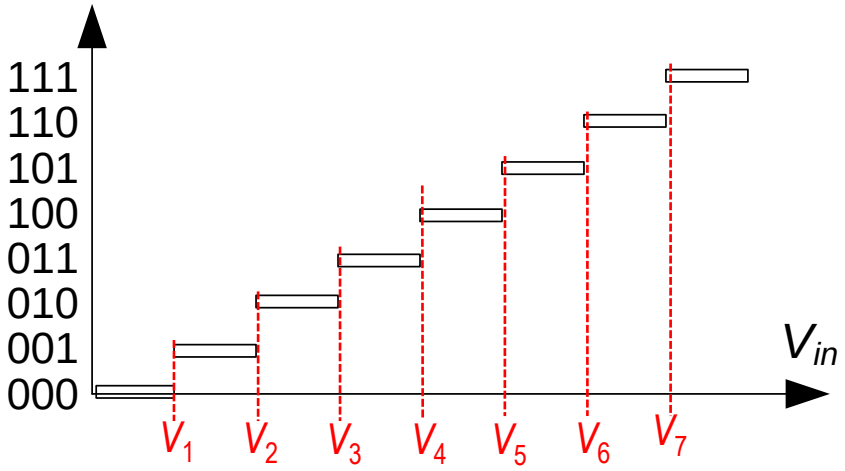


Princip för omvandling.



Exempel: 3-bitars omvandling





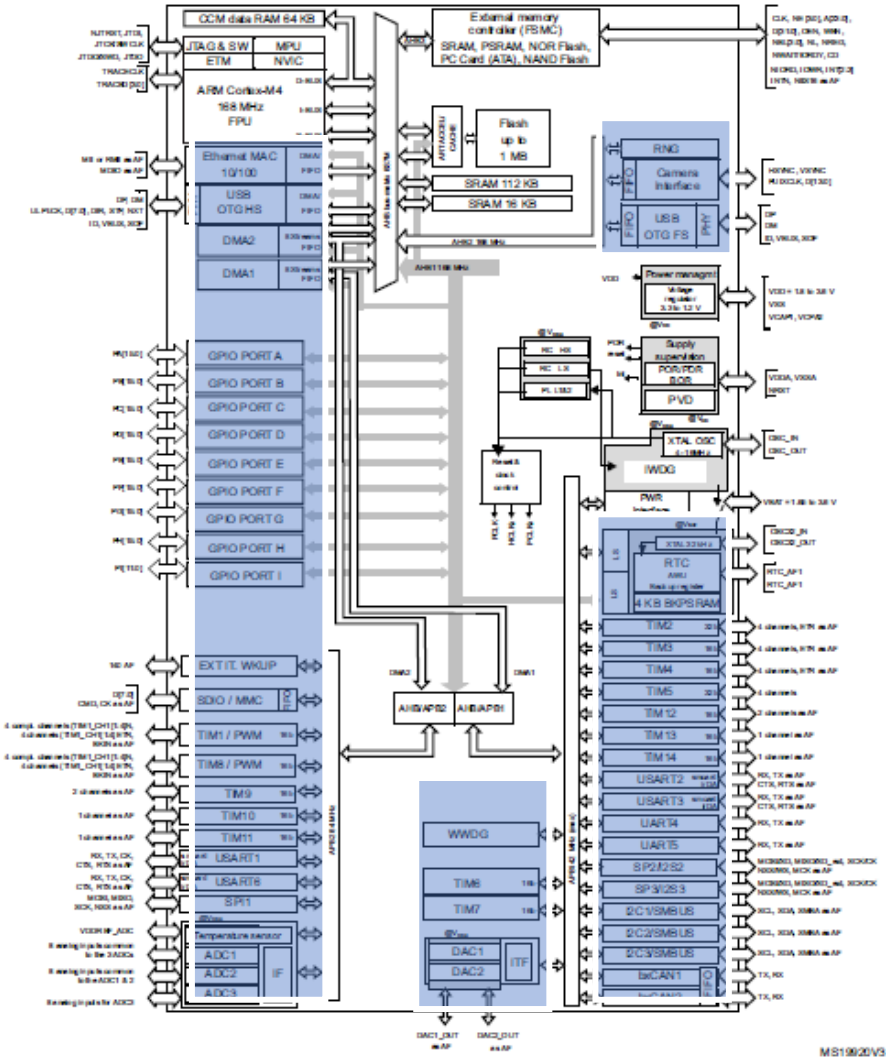
AD i praktiken

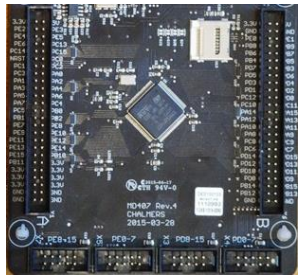
Exempel, ADC med MD407

```
typedef unsigned int uint32_t;
typedef unsigned short uint16_t;
typedef struct
{
  volatile uint32_t sr;     /* status register                          0x00 */
  volatile uint32_t cr1;    /* ontrol register 1                        0x04 */
  volatile uint32_t cr2;    /* control register 2                       0x08 */
  volatile uint32_t smpr1;  /* sample time register 1                   0x0C */
  volatile uint32_t smpr2;  /* sample time register 2                   0x10 */
  volatile uint32_t jofr1;  /* injected channel data offset register 1  0x14 */
  volatile uint32_t jofr2;  /* injected channel data offset register 2  0x18 */
  volatile uint32_t jofr3;  /* injected channel data offset register 3  0x1C */
  volatile uint32_t jofr4;  /* injected channel data offset register 4  0x20 */
  volatile uint32_t htr;    /* watchdog higher threshold register       0x24 */
  volatile uint32_t ltr;    /* watchdog lower threshold register        0x28 */
  volatile uint32_t sqr1;   /* regular sequence register 1              0x2C */
  volatile uint32_t sqr2;   /* regular sequence register 2              0x30 */
  volatile uint32_t sqr3;   /* regular sequence register 3              0x34 */
  volatile uint32_t jsqr;   /* injected sequence register               0x38 */
  volatile uint32_t jdr1;   /* injected data register 1                 0x3C */
  volatile uint32_t jdr2;   /* injected data register 2                 0x40 */
  volatile uint32_t jdr3;   /* injected data register 3                 0x44 */
  volatile uint32_t jdr4;   /* injected data register 4                 0x48 */
  volatile uint32_t dr;     /* regular data register                    0x4C */
} ADC_ARM_DEVICE, *PADC_ARM_DEVICE;

#define ADC1 ((PADC_ARM_DEVICE) 0x40012000)
```

```
void init_adc( void )
{
  /* Bestäm GPIO port och pinne, initiera som ANALOG ingång */

  /* ADC kanal 0,  Rank 1, Konverteringstid 56 cykler */
  ADC1->smpr2 = 3;
  ADC1->sqr3 = 0 ;
}

unsigned short read_adc( void )
{
  ADC1->cr2 |= (uint32_t) (1<<30) ;  /* Starta konvertering */
  while( (ADC1->sr & (1<<1) ) ==0 ); /* Vänta tills AD omvandling klar */
  return (unsigned short) ADC1->dr;  /* Returnera resultatet */
}

void main(void)
{
    uint16_t samples[1024];
    init_adc();
    for( int i = 0; i < sizeof(samples); i++ )
    {   /* Samplingsfrekvens, 10 Hz */
        samples[i] = read_adc();
        delay_ms( 100 );
    }
}
```

Periferikretsar i enchipsdatorn ST32F407





General Purpose IO (GPIO)
Advanced control timers (TIM1-TIM8)
General Purpose timers (TIM2-TIM5, TIM9-TIM14)
Analog-to-digital converter (ADC)
Universal synchronous asynchronous receiver transmitter (USART)
CRC calculation unit (CRC)
Digital–to-analog converter (DAC)
Digital Camera interface (DCMI)
LCD-TFT Controller (LTDC)
Independent watchdog (IWDG)
Window watchdog (WWDG)
Crypthographic processor (CRYP)
Random number generator (RNG)
Hash processor (HASH)
Real time clock (RTC)
Inter integrated circuit (I^2C) interface
Serial peripheral interface (SPI)
Serial Audio interface (SAI)
Secure digital input/output interface (SDIO)
Controller area network (CAN)
Ethernet nedia access control (MAC)
Universal serial bus (USB)