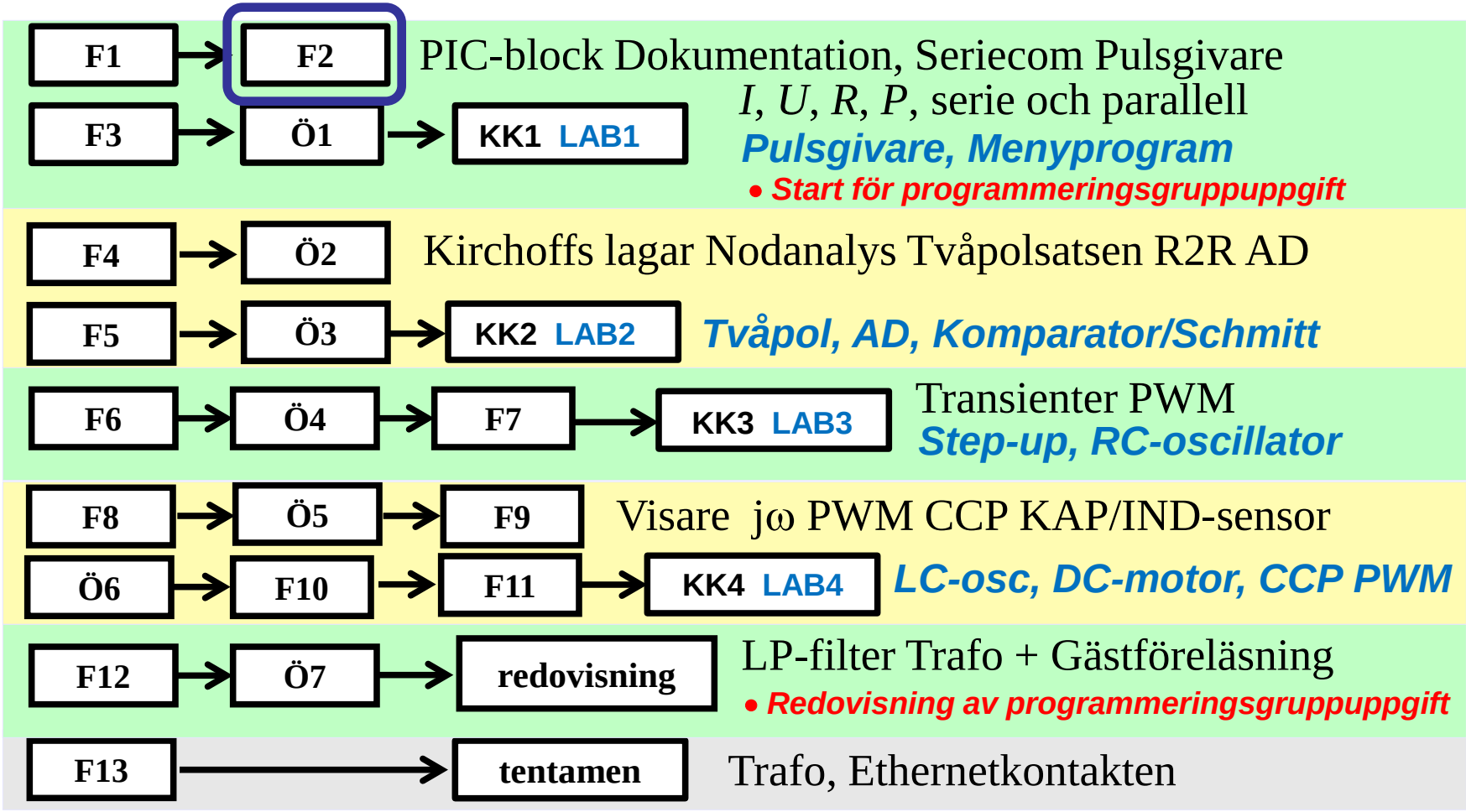
IE1206 Inbyggd Elektronik



William Sandqvist william@kth.se

Kommunikation

William Sandqvist william@kth.se

ASCII-tabellen

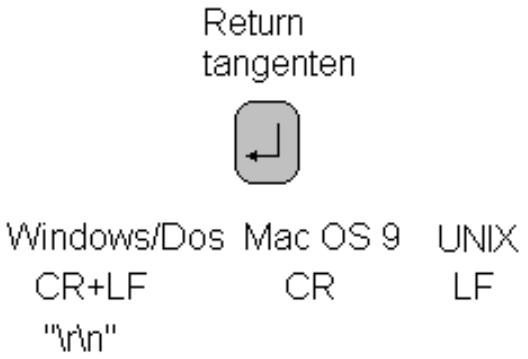
	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Varje tecken lagras i en **Byte**, **`char`**.

”Hej!”

48 65 6A 21 00

01001000 01100101 01101010 00100001 00000000



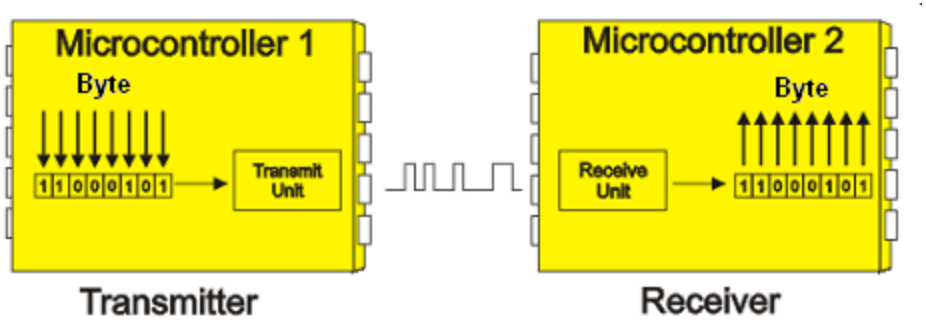
http://ascii-table.com/

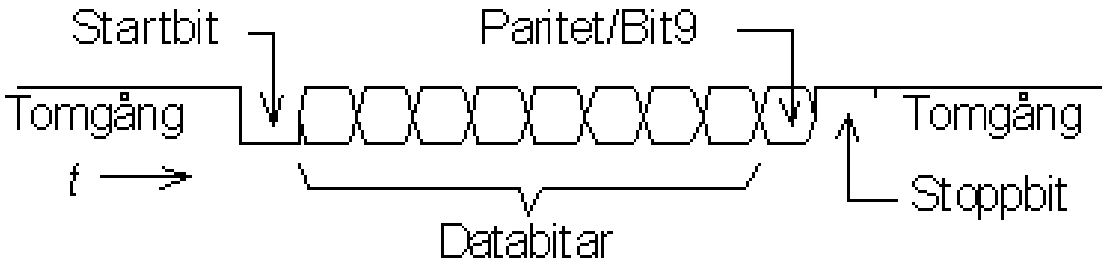
PICKit 2 UART Tool använder **`\r\n`**

William Sandqvist william@kth.se

Seriekommunikation

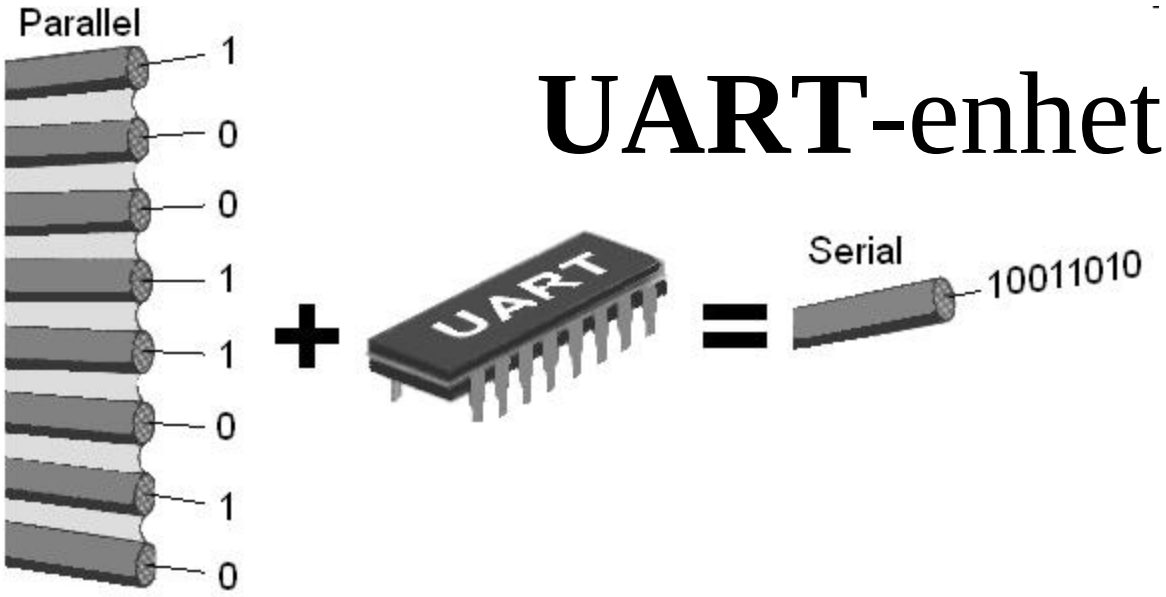
parallell-serie-parallell omvandling





William Sandqvist william@kth.se

UART-enhet



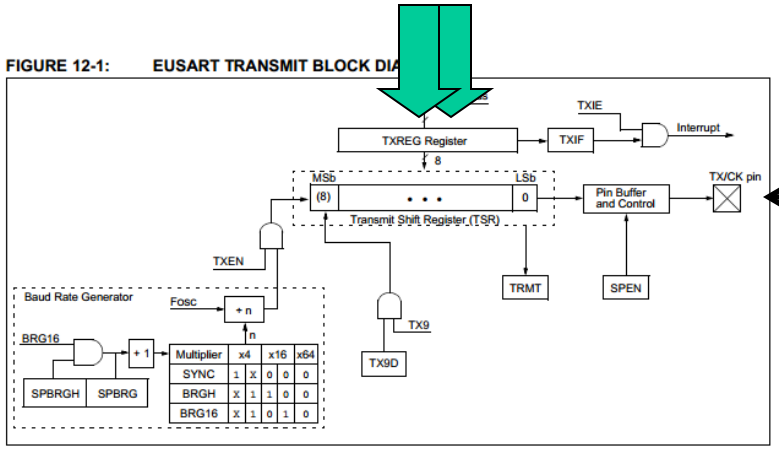
Serie/Parallell-omvandlingen på bitnivå tas ofta om hand med en specialkrets UART (**U**niversal **A**synchronous **R**eciever/ **T**ransmitter), så att processorn kan leverera/mottaga hela tecken.

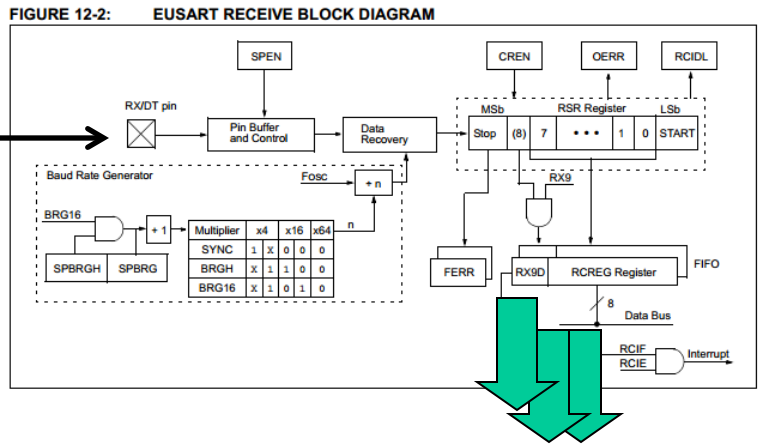
En sådan finns inbyggd i de flesta PIC-processorer (USART/EUSART).

William Sandqvist william@kth.se

Seriekommunikationsenhet

Självgående seriekommunikationsenhet





Sändaren kan hålla **två** tecken i kö från processorn.

Mottagaren kan ta emot upp till **tre** tecken innan processorn behöver agera.

Medan kommunikationen pågår kan processorn göra annat!

William Sandqvist william@kth.se

PIC16F690 EUSART

PIC 16F690 innehåller en inbyggd seriekommunikations-enhet, **EUSART** (Enhenced Universal Synchronous or Asynchronous Reciever and Transmitter).
Som namnet anger är denna enhet användbar både för synkron och asynkron seriekommunikation, men vi kommer bara utnyttja den för ***asynkron*** seriekommunikation.

EUSART består av tre delar.

- **SPBRG** (Serialunit Programable BaudRateGenerator) är en programmerbar Baudgenerator för överföringshastigheten.
- **USART Transmitter** är sändarenhet
- **USART Reciever** är mottagarenhet.

William Sandqvist william@kth.se

Bitrate

Vid seriekommunikation är det nödvändigt att sändare och mottagare opererar med samma i förväg överenskommna hastighet. Den hastighet med vilken bitarna överförs kallas för **Bitrate** [bit/sek].

Vanliga Bitrate´s är multiplar av 75 bit/sek som: 75, 150, 300, 600, 1200, **9600**, 19200 och 38400 bit/sek.

Bitrateklockan tas från en Baudrategenerator.

William Sandqvist william@kth.se

Baud Rate Generator **BRG**

REGISTER 12-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

REGISTER 12-3: BAUDCTL: BAUD RATE CONTROL REGISTER

R-0	R-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN
bit 7							bit 0

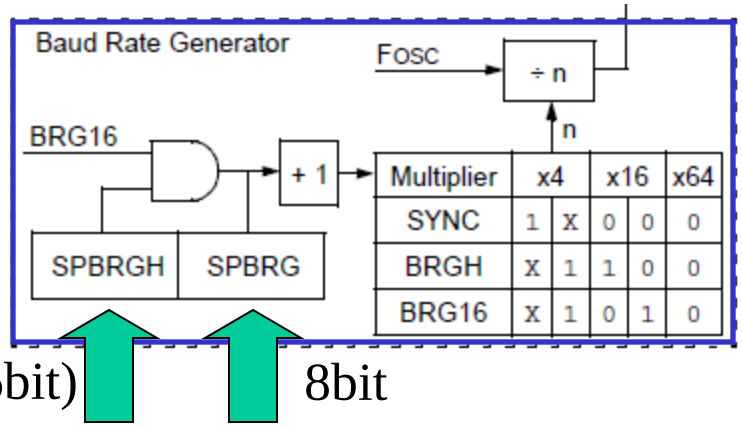


TABLE 12-3: BAUD RATE FORMULAS

Configuration Bits		BRG/EUSART Mode	Baud Rate Formula
BRG16	BRGH		
0	0	8-bit/Asynchronous	Fosc/[64 (n+1)]
0	1	8-bit/Asynchronous	Fosc/[16 (n+1)]
1	0	16-bit/Asynchronous	
1	1	16-bit/Asynchronous	Fosc/[4 (n+1)]

En bit **BRGH** bestämmer låghastighets eller hög-hastighets läge. En bit **BRG16** inför 16-bitars neddelningstal.

- *Vår inställning*:

```
/* 9600 Baud @ 4 MHz */
BRG16=0; BRGH=1; SPBRG = 26-1;
```

Ett register **SPBRG** innehåller neddelningstalet 8/16-bitar.

William Sandqvist william@kth.se

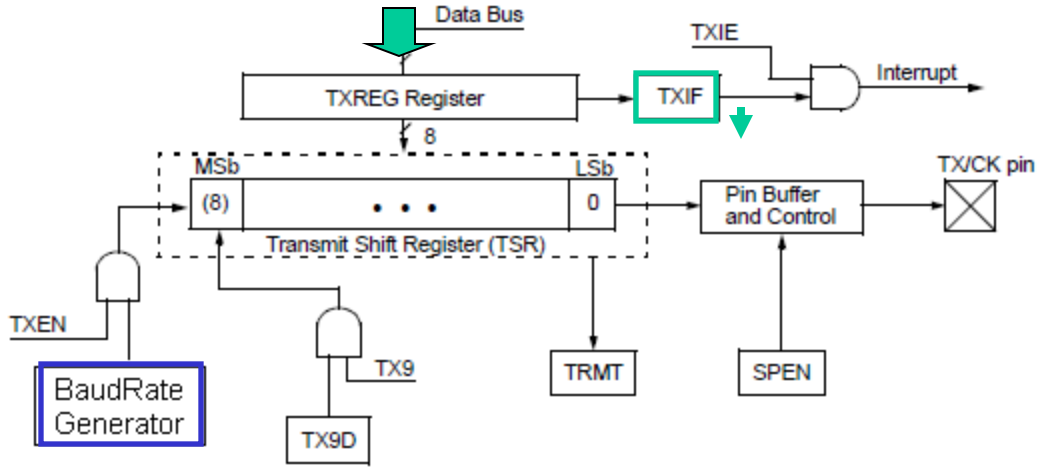
Baud Rate Generator **BRG**

De omfattande inställningsmöjligheterna är till för att man ska kunna hitta en kombination som ger så **noggrann** bitrate som möjligt.

Två processorer som kommunicerar asynkront med varandra måste ha bitrate’s som överenstämmer bättre än ±2,5%. Annars riskerar man att tecken blir förvrängda.

William Sandqvist william@kth.se

Transmitter



För att sända ett tecken räcker det med att ”lägga det” i **TXREG**-registret. När sändarregistret TSR är "redo" kopieras tecknet över till detta och skiftas ut seriellt på pinnen TX/CK pin. Om man har ytterligare ett tecken att sända kan man nu placera detta i "väntkö" i TXREG. Så fort TSR är tomt laddas nästa tecken in från TXREG automatiskt till TSR.

I blockdiagrammet visas flaggan **TXIF** (Transmitter Interupt Flag) som anger om sändarregistret TXREG är fullt eller ej. Flaggan 0-ställs automatiskt när ett tecken hamnar i TSR.

Transmitter settings

REGISTER 12-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

bit 6 = 0 **TX9**: Ingen niobits-sändning.

bit 5 = 1 **TXEN**: Transmit Enable bit. Måste vara på.

bit 4 = 0 **SYNK**: Usart mode select bit. Vi väljer ***asynkront*** arbetssätt.

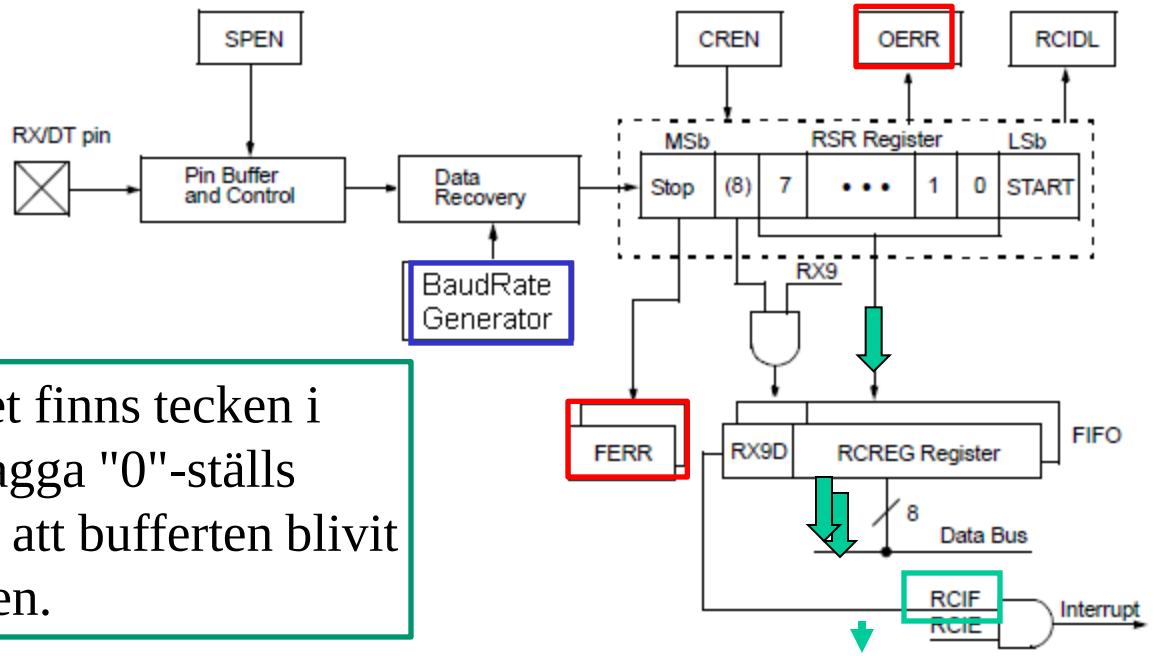
bit 2 = 1 **BRGH**: High Baudrate select bit. Vi väljer höghastighetsläge.

bit 1 TRMT: Flagga som är ”1” om TSR är tomt.

Reciever

Tecken inkommer från pinnen RX/DT till mottagarregistret RSR. När mottagningen av ett tecken är klart förs det över till **RCREG** som är en så kallad FIFO-buffert. Denna buffert rymmer *två* tecken som läses i samma ordning som de kom.

Bufferten medför att ett program kan låta bli att *bevaka* mottagarregistret under den tid det tar att mottaga *tre* tecken.



Flaggan **RCIF** anger om det finns tecken i bufferten eller ej. Denna flagga "0"-ställs *automatiskt* när man läst så att bufferten blivit tom, dvs. efter ett /två tecken.

Flaggorna OERR, FERR varnar för felaktigt mottagna tecken.

William Sandqvist william@kth.se

Reciever settings

REGISTER 12-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER(1)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 = 1 **SPEN**: Enablar serieporten.

bit 6 = 0 **RX9**: Ingen niobitsmottagning.

bit 4 = 1 **CREN**: Continuous Receive Enable bit. Mottag fler tecken i följd.

bit 2 och **bit 1** **FERR OERR** Flaggor som anger om mottagningen blivit fel.

Det är biten/bitvariabeln **RCIF** som anger om det finns tecken att hämta.

Initiering av serieporten

```
void initserial( void )
/* initialise serialcom port 16F690  */
{
  SPEN = 1;
  BRGH = 1; /* Async high speed      */
  BRG16= 0; /* SPRG n is 8-bit       */
  TXEN = 1; /* transmit enable       */
  SPBRG = 26-1; /* 9600 Baud @ 4 MHz */
  CREN = 1; /* Continuous receive    */
  RX9 = 0;  /* 8 bit reception       */
  TRISB.7 = 0; /* TX is output       */
  TRISB.5 = 1; /* RX is input        */
}
```

- Görs en gång i början av programmet.

Seriecom-funktioner

```
char getchar( void ) /* recieves one char */
{
  char d_in;
  while ( !RCIF ) ;  /* wait for char  */
  d_in = RCREG;
  return d_in;
}
```

OBS! Blockerande funktion!

Här blir man sittande tills ett tecken inkommer!

```
void putchar( char d_out ) /* sends one char   */
{
  /* wait until previous character transmitted */
  while (!TXIF) ;
  TXREG = d_out;
}
```

Varning! Recievern kan låsa sig!

Programmet måste läsa mottagarenheten ***innan*** den hunnit mottaga *tre* tecken - annars låser den sig!
När man ansluter seriekontakten kanske man "darrar" på handen på sådant sätt att "kontaktstudsarna" blir till många mottagna tecken. Om mottagarenheten då "låser sig" är detta naturligtvis ett mycket svårt/omöjligt "programfel" att hitta!

Lösningen är en **upplåsningsrutin** att kunna ta till vid behov. Man anropar upplåsningsfunktionen direkt innan man förväntar sig inmatning via serieporten.

William Sandqvist william@kth.se

OverrunRecover()

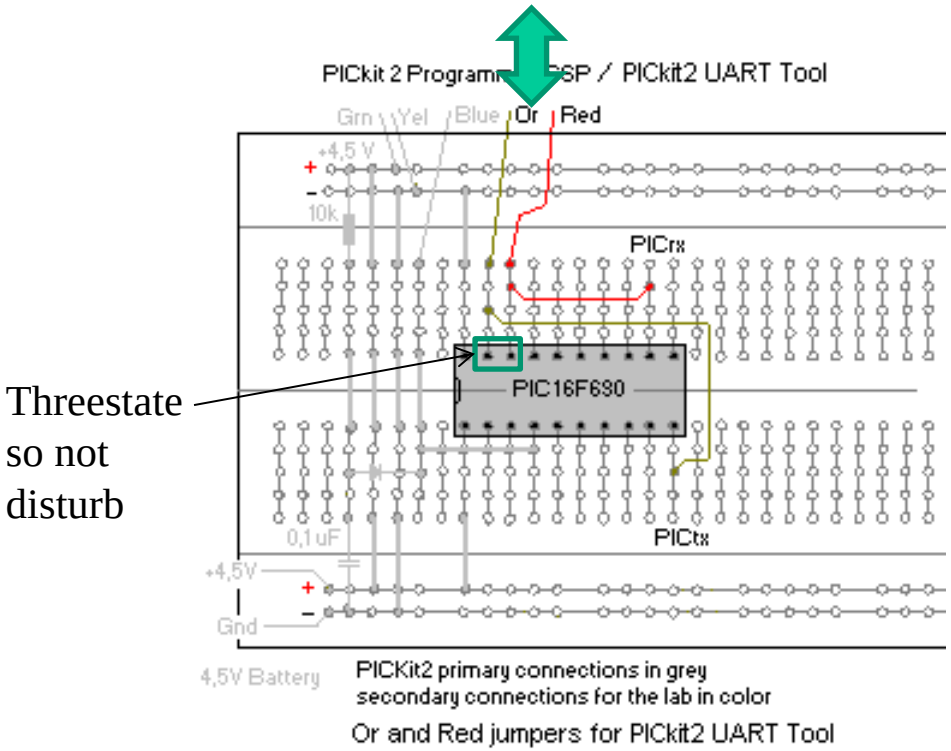
```
void OverrunRecover( void )
{
  char trash;
  trash = RCREG;
  trash = RCREG;
  CREN = 0;
  CREN = 1;
}
```

- *Upplåsningsrutinen.*

Note:	If the receive FIFO is overrun, no additional characters will be received until the overrun condition is cleared. See ***Section 12.1.2.5 "Receive Overrun Error"*** for more information on overrun errors.

Seriekom - **Hårdvara**

1) PICKIT 2 UART Tool genom programmerings-ledningarna



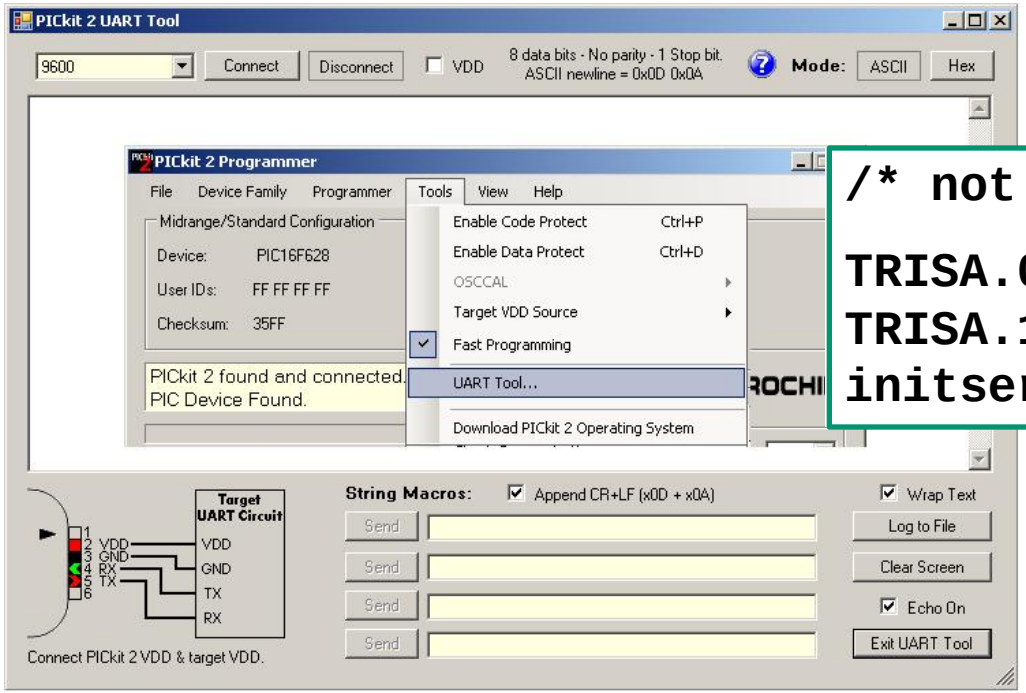
Threestate so not disturb

Bygla PIC-processorns **serieport** till **programmerings-ledningarna** (**Or**, **Red**).

William Sandqvist william@kth.se

Seriekom - **Terminalprogram**

1) PICKIT 2 UART Tool, kan användas som terminalprogram genom programmerings-ledningarna.




```
/* not disturb UART-Tool */
TRISA.0 = 1;
TRISA.1 = 1;
initserial();
```

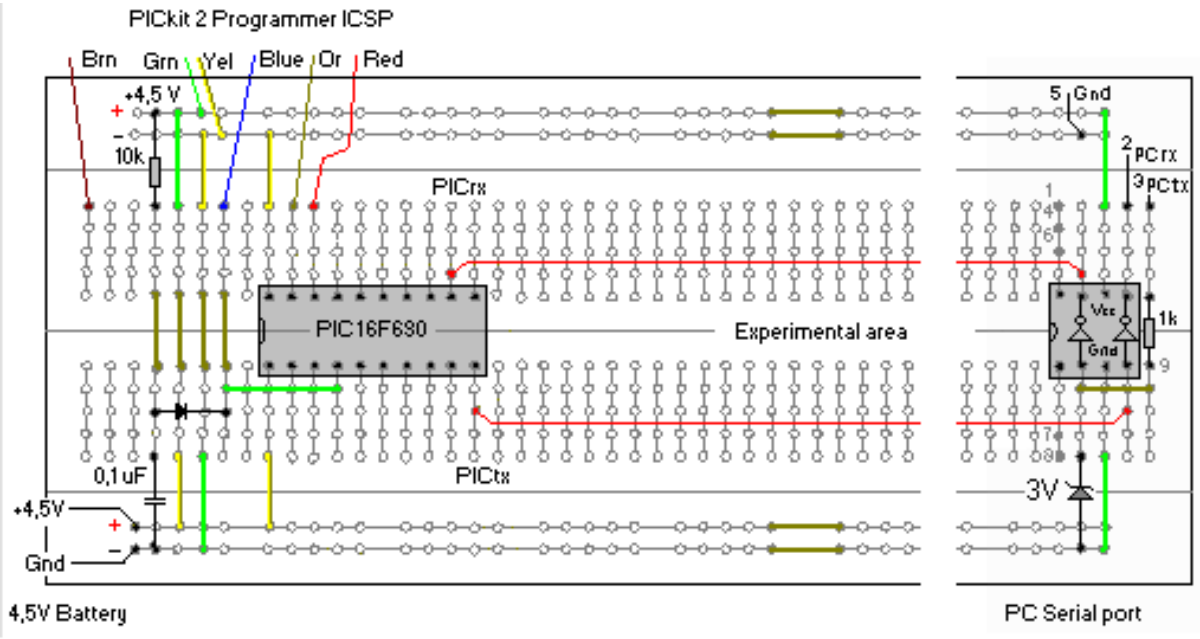
Threestate på programmerings-pinnarna!

William Sandqvist william@kth.se

(Seriekom – **Hårdvara)**

PC-serieport

2) PC med serieport



Inverterare

ICL7667

Invertera signalerna till/från PIC-processorns serieport innan den ansluts till PC:ns serieport. (Ska egentligen vara ±12V, men inverterare brukar räcka).

(Det finns också speciella kretsar som genererar ±12V signaler för seriekommunikation.)

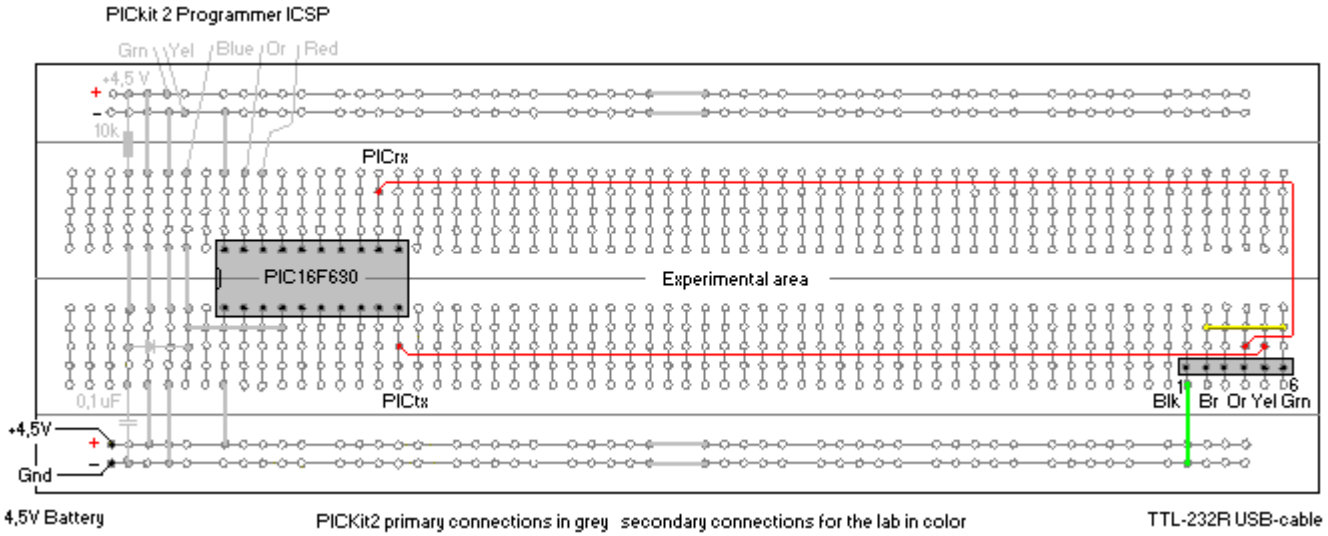
Seriekommunikation **USB-serial-TTL**

3) **FTDI TTL232R** ansluts ***direkt*** till processorpinnarna.

Rättvända logiknivåer

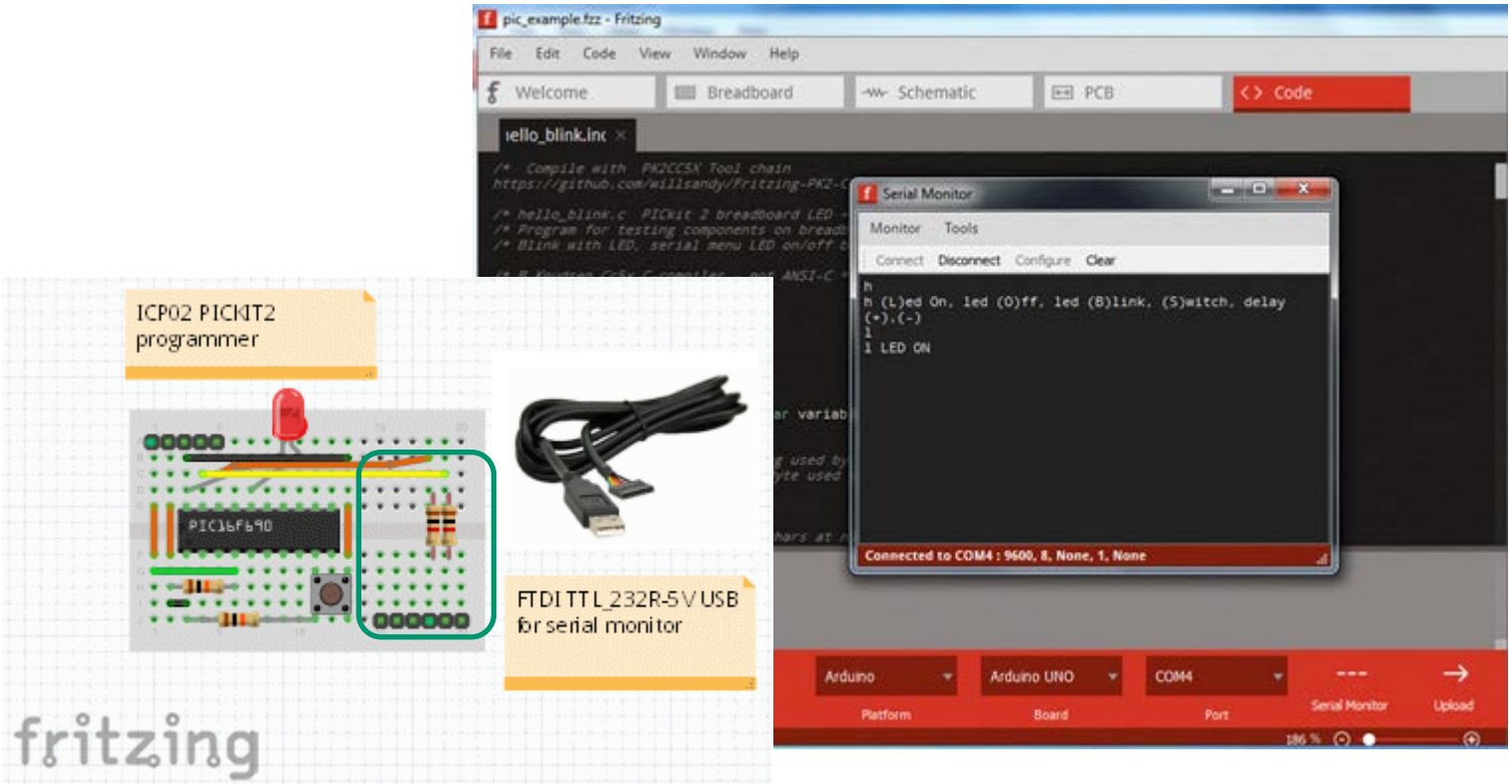
De flesta PC saknar numera serieport, en drivrutin kan installera en ***virtuell*** **USB-serieport**.

Drivrutinen finns redan i Windows



William Sandqvist william@kth.se

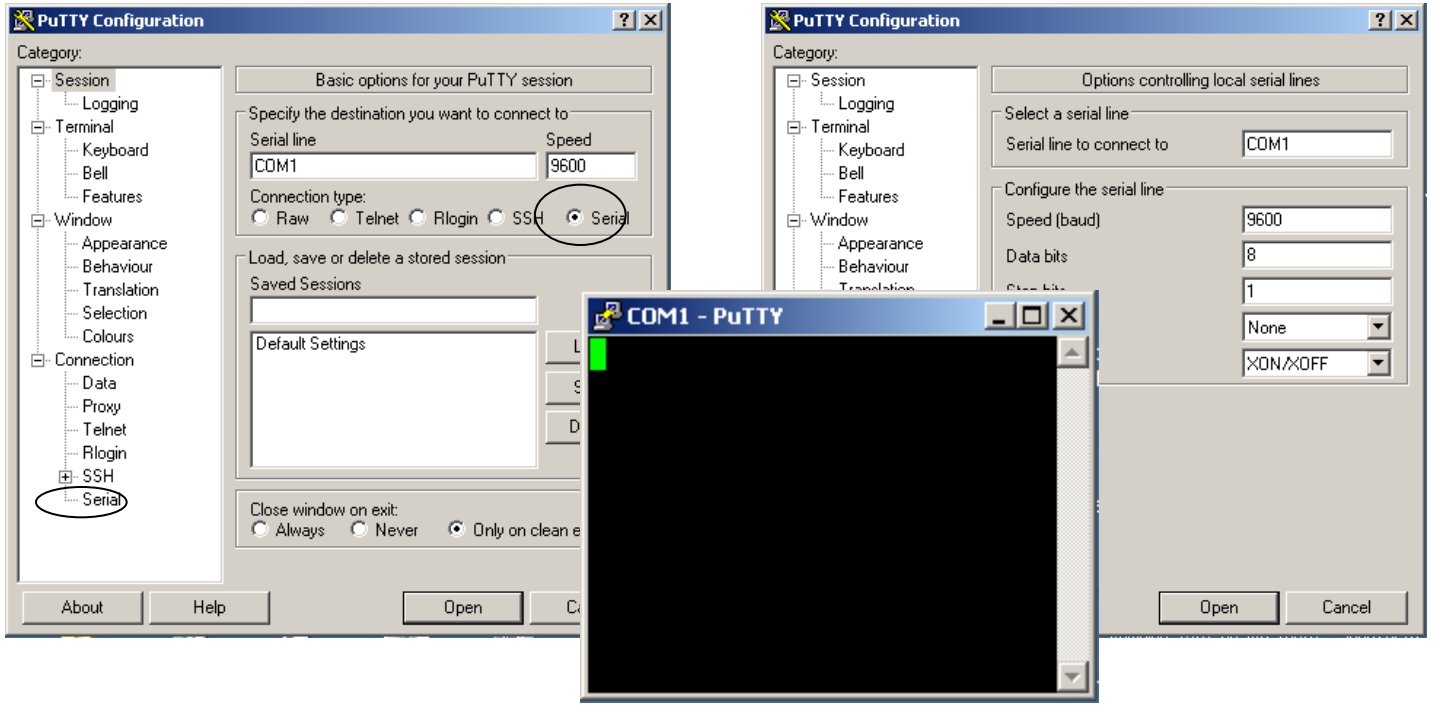
Fritzing Serial Monitor



William Sandqvist william@kth.se

Terminalprogram till **PC**

Om Du använder USB-virtuell serieport – tag reda på COM-portens nummer först (med Enhetshanteraren/Device Manager) …



putty.exe

PuTTY

PuTTY

William Sandqvist william@kth.se

Testprogram: **echo()/crypto()**

Om PIC-processorn ”ekar” de skrivna tecknen så fungerar kommunikationen.

```
void main( void)
{
  char c;
  TRISB.6 = 1; /* not to disturb UART-Tool */
  TRISB.7 = 1; /* not to disturb UART-Tool */
  initserial();
  delay10(100); /* 1 sek delay */
  /* 1 sek to turn on VDD and Connect UART-Tool */
  while( 1)
   {
     c = getchar( ); /* input 1 character */
     if( c == '\r'||c == '\n')
      putchar(c);
     else putchar(c); /* echo the character */
     /* putchar(c+1) => Crypto! */
   }
}
```

Ändå säkrare variant: krypto ! **A→B**

William Sandqvist william@kth.se

William Sandqvist william@kth.se

Seriekommunikation direkt, med valfri pinne

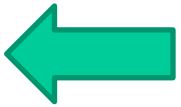
William Sandqvist william@kth.se

Bit-banging

Det är mycket vanligt att man programmerar seriekommunikation ”bit för bit”. ***Valfri*** **portpinne** kan användas så om inte annat är detta ett mycket bra felsökningshjälpmedel.

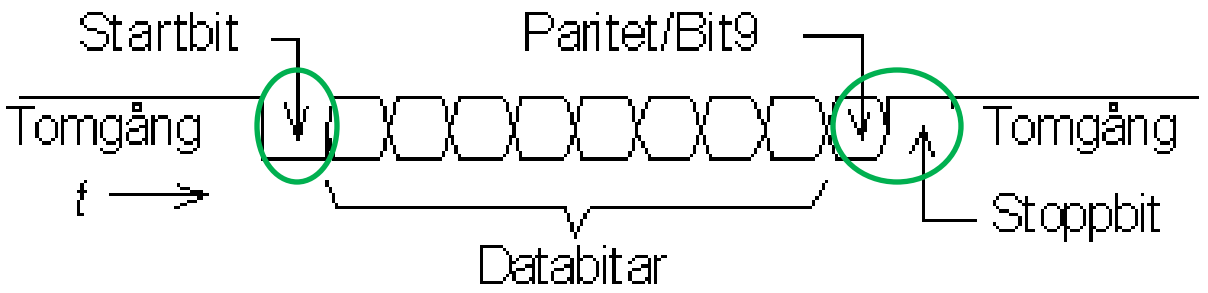
En lämplig Bitrate är då **9600**. $T = 1/9600 = 104{,}17\ \mu s$. Om processorns klockfrekvens är 4 MHz behövs det en fördröjningsloop som tar 104 instruktioner.

```
/* delay one bit 104 usec at 4 MHz       */
/* 5+18*5-1+1+9=104 without optimization */
i = 18;
do ;
while( --i > 0);
nop();
```



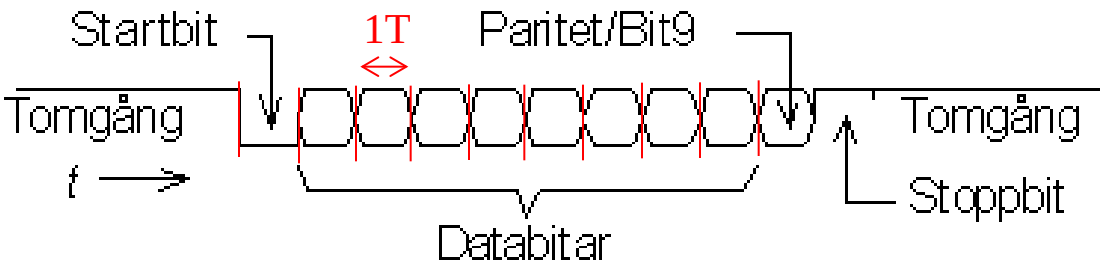
Assemblera och räkna instruktionerna. Varje instruktion tar 1 μs.

Bitar och extrabitar



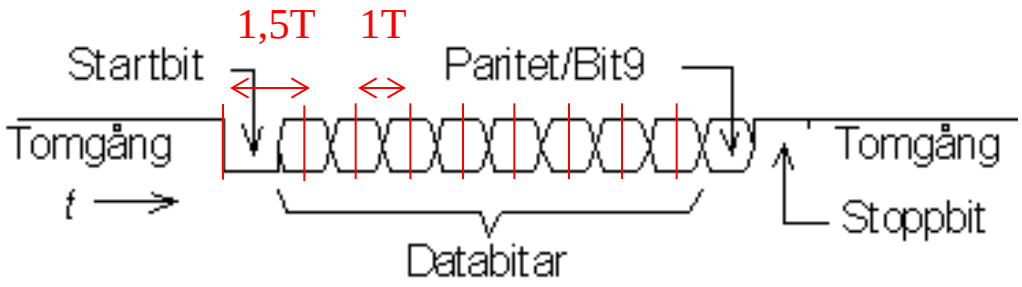
Den asynkrona överföringstekniken innebär att man för varje byte lägger till **extra bitar** som ska göra det möjligt att skilja ut byten från bitströmmen. Ofta lägger man dessutom till en bit för felindikering.

Skicka ett tecken …



- Dataöverföringens startar med att datalinjen hålls låg "0" under ett tidsintervall som är en bit långt (T = 1/Bitrate). Detta är ***startbiten***.
- Under 8 lika långa tidsintervall följer sedan databitarna, ettor eller nollor, med den minst signifikantabiten först och den mest signifikanta biten sist.
- (Därefter *kan* följa en ***paritetsbit***, ett hjälpmedel vid detekteringen av överföringsfel.)
- Överföringen avslutas slutligen av att datalinjen för åtminstone ett bit-tidsintervall är hög. Det är ***stoppbiten***.

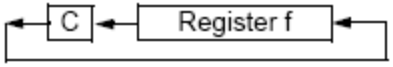
Mottaga ett tecken



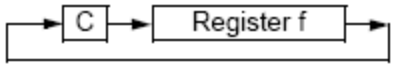
Mottagningen av data sker genom att man först inväntar startbitens negativa flank, för att därefter först registrera datalinjen efter 1,5T fördröjning och därefter efter 1T (registrering vid databitarnas "mittpunkter").

Mottagaren "synkroniseras om" på nytt vid varje startflank.

Rotation av tal

RLF	Rotate Left f through Carry
Syntax:	[*label*] RLF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	See description below
Status Affected:	C
Description:	The contents of register 'f' are rotated one bit to the left through the Carry flag. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is stored back in register 'f'.
	
Words:	1
Cycles:	1

Example:	`RLF REG1,0`
	Before Instruction
	`REG1 = 1110 0110`
	`C = 0`
	After Instruction
	`REG1 = 1110 0110`
	`W = 1100 1100`
	`C = 1`

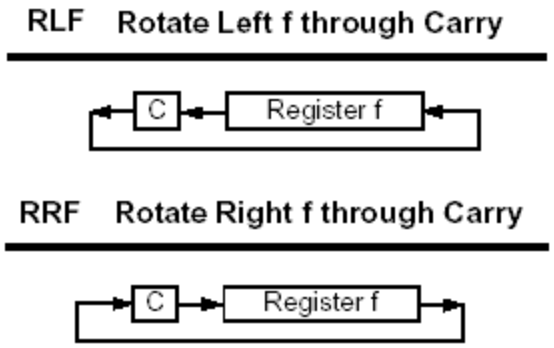
RRF	Rotate Right f through Carry
Syntax:	[*label*] RRF f,d
Operands:	0 ≤ f ≤ 127 d ∈ [0,1]
Operation:	See description below
Status Affected:	C
Description:	The contents of register 'f' are rotated one bit to the right through the Carry flag. If 'd' is '0', the result is placed in the W register. If 'd' is '1', the result is placed back in register 'f'.
	

PIC-processorerna har två instruktioner för att ”rotera” tal **RLF** och **RRF**.

Dom här instruktionerna behöver vi i fortsättningen …

William Sandqvist william@kth.se

Cc5x har ***inbyggda*** funktioner **rl()** och **rr()**




```
char rl( char );
```

```
char rr( char );
```

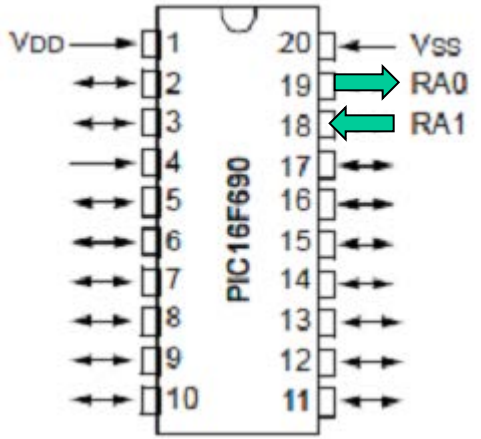
C-språket har två skiftoperatorer högerskift **>>** och vänsterskift **<<** , någon egentlig "rotations"-operator finns inte.
För att man trots detta ska kunna utnyttja PIC-processorernas rotations-instruktioner, har kompilatorn **Cc5x** lagt till två ***interna*** funktioner **`char rl( char );`** och **`char rr( char );`**.
Funktionerna genererar direkt assemblerinstruktionerna **RLF** och **RRF**.

Carryflaggan når man som bitvariabeln **`bit Carry;`**

Debug-kommunikation

Chip-programmering **och** kommunikation.

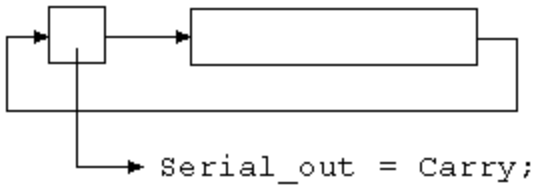
PICKit2 UART-tool kan användas som ett enkelt debug-verktyg. *Samma ledningar* som används till att programmera PIC-chippet används av UART-tool för seriekommunikation.



Man behöver därför en bitbangingrutin för serie-kommunikation med *dessa pinnar.*

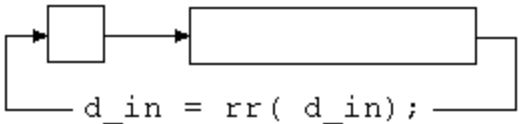
```
void initserial( void ) /* init PIC16F690 serialcom */
{
  ANSEL.0 = 0; /* No AD on RA0 */
  ANSEL.1 = 0; /* No AD on RA1 */
  PORTA.0 = 1; /* marking line */
  TRISA.0 = 0; /* output to PK2 UART-tool  */
  TRISA.1 = 1; /* input from PK2 UART-tool */
}
```

void putchar(char)




```
void putchar(char d_out)
{ char count, i;
  Serial_out = 0; /* set startbit */
  for(count = 10; count > 0; count--)
    { /* delay 104 usec                */
      i = 18; do ; while( --i > 0); nop();
      Carry = 1;
      d_out = rr(d_out);
      Serial_out = Carry;
    }
}
```

char getchar(void)




```
char getchar( void )
{
  char d_in, count, i;
  while( Serial_in == 1) /* wait for startbit */;
  /* 1.5 bit 156 usec no optimization          */
  i = 28; do ; while( --i > 0); nop(); nop2();
  for(count = 8; count > 0; count--)
    {
      Carry = Serial_in;
      d_in = rr( d_in );
      /* 1 bit 104 usec no optimization        */
      i = 18; do ; while( --ti > 0); nop();
    }
  return d_in;
}
```

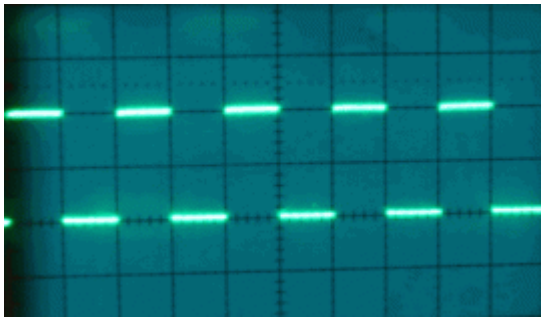
William Sandqvist william@kth.se

Testprogram: **fyrkantvåg**

Man kan kontrollera om bitrate är korrekt med ett oscilloskop.

9600 bit/sek. Om man sänder kontinuerligt 8 bitar med **startbit** och **stoppbit** bokstaven ’**U**’ (**1**01010101**0**) får man en fyrkatvåg med f = 4800 Hz. Detta test är bra att känna till.

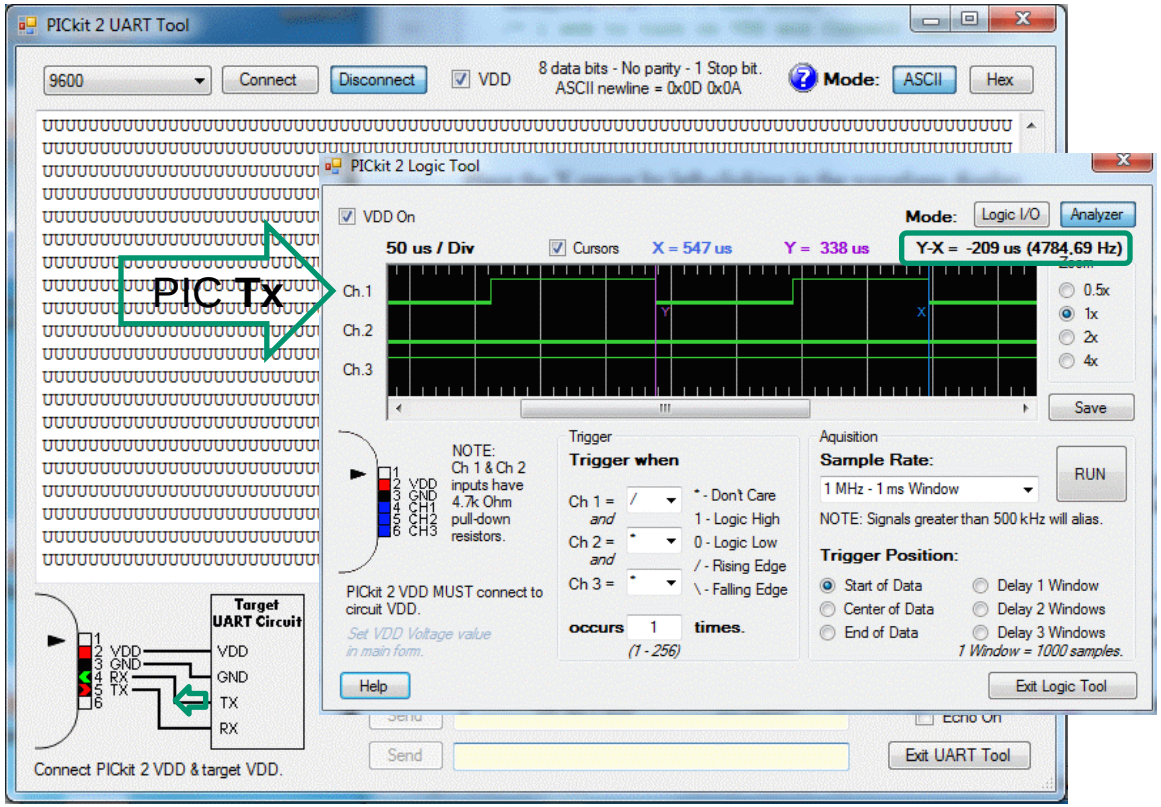
```
while(1) putchar('U');
```



Om Du ***inte*** har något oscilloskop?

```
while(1) putchar('U');
```

PICKit2 Logic Tool



Vi kan se detaljer som att stoppbiten blivit lite längre än övriga bitar …

För att mäta frekvensen klickar man markörerna på plats med vänster och höger musknapp. Frekvensen är 4785 Hz (≈4800).

William Sandqvist william@kth.se

William Sandqvist william@kth.se