Seriekommunikation

Ur innehållet

- Nätverkstopologier
- Nätverksprotokoll
- Asynkron/synkron seriell överföring
- Programmering av USART-krets

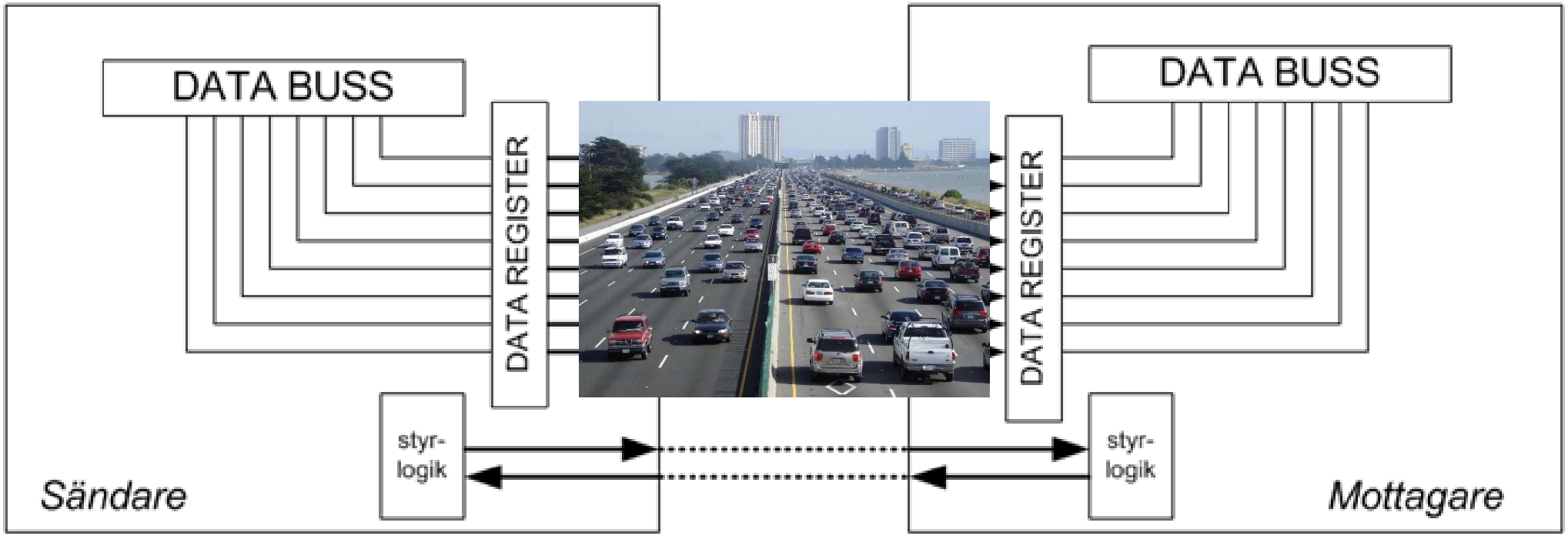
Läsanvisningar

- Arbetsbok kapitel 7

Målsättningar:

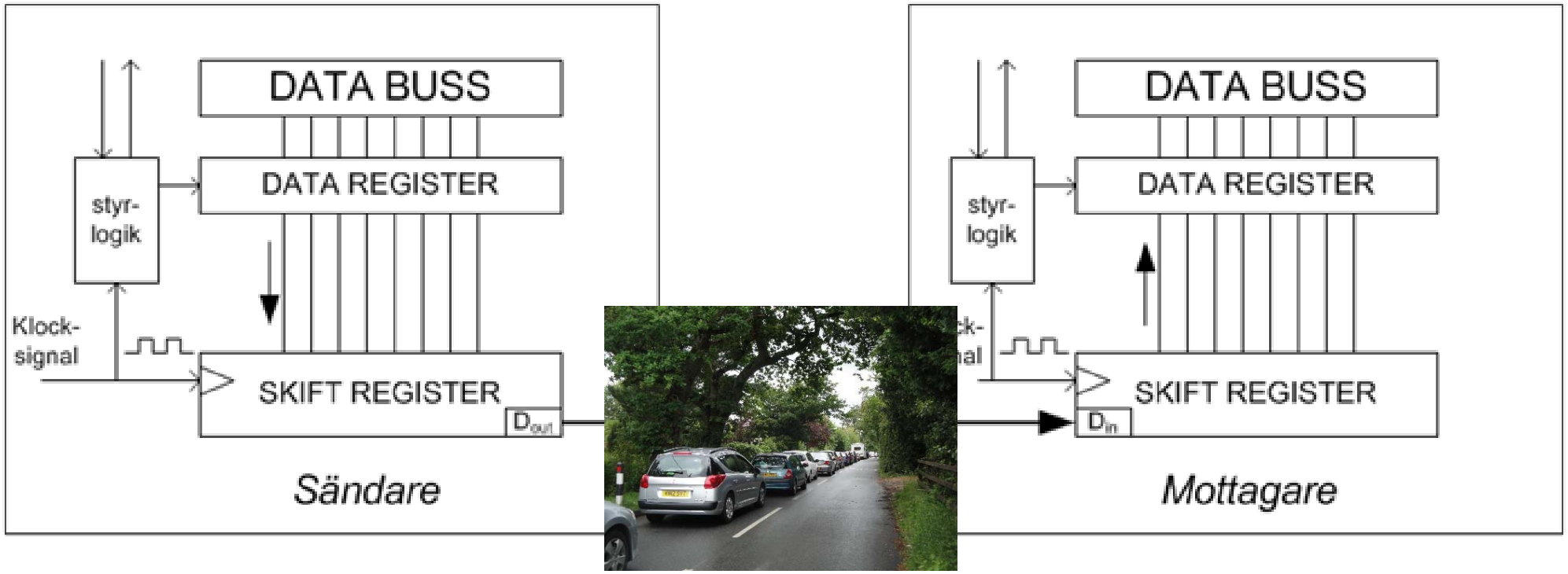
- Kunna programmera en USART-krets för RS232-kommunikation

Parallell överföring



+ god bandbredd
- många ledare ger dyrare överföringsmedia
mycket snabbt vid korta avstånd

Seriell överföring

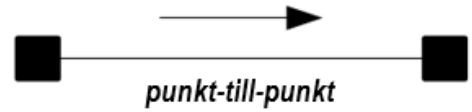


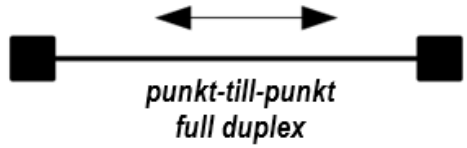
+ få ledare ger enklare (billigare) överföringsmedia
- sämre bandbredd
överföringshastighet efter prestandakrav

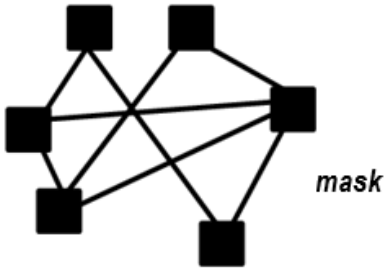
Nätverkstopologi – löst kopplade system

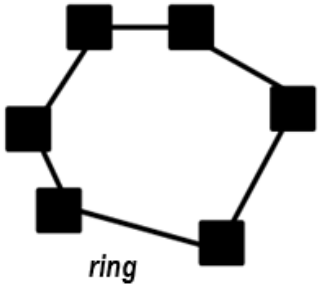
Strukturer för hur datorer kopplas i hop, ofta ser man kombinationer av de olika varianterna.

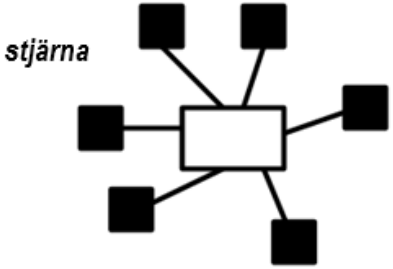
Olika ”protokoll” har utvecklats för olika strukturer, exvis:
RS232 (punkt till punkt)
Ethernet (mask, stjärna)
Token ring (ring)
USB (träd)
CAN, *LIN* (buss)

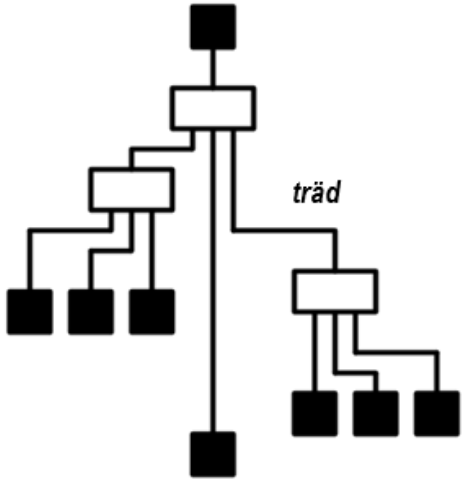


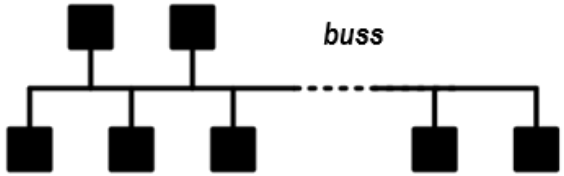






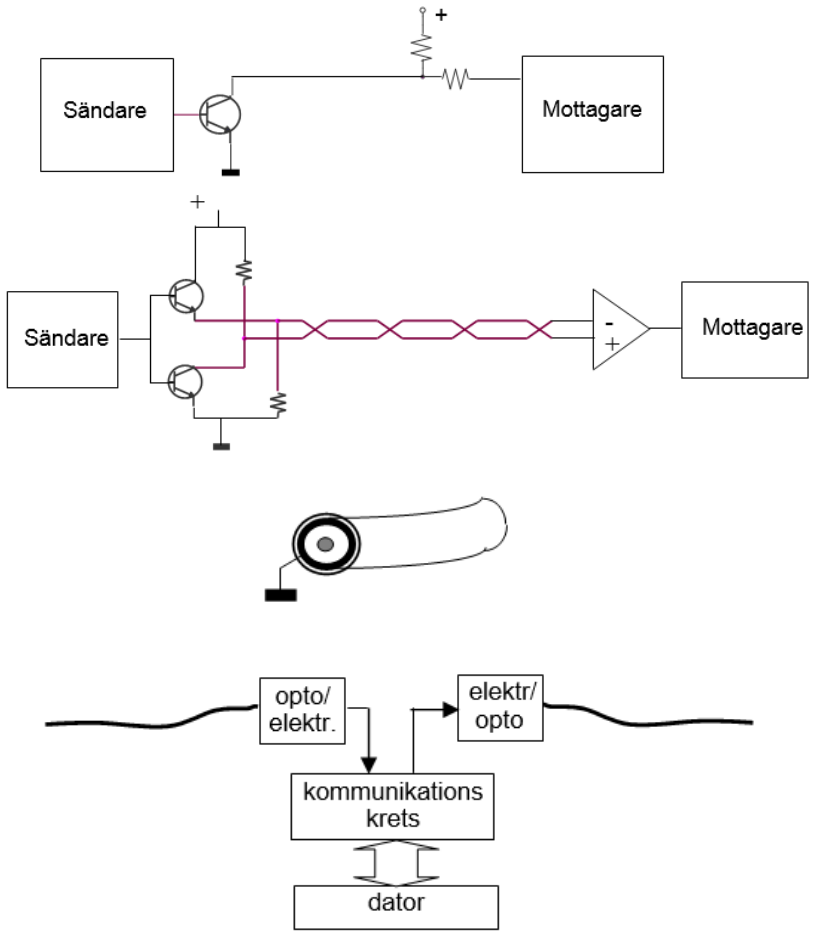






Nätverksprotokoll

Kommunikationsmedia

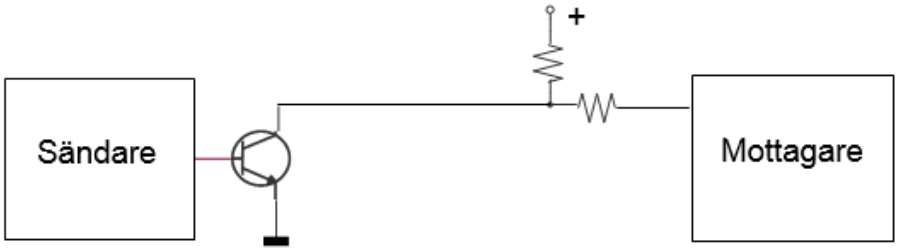


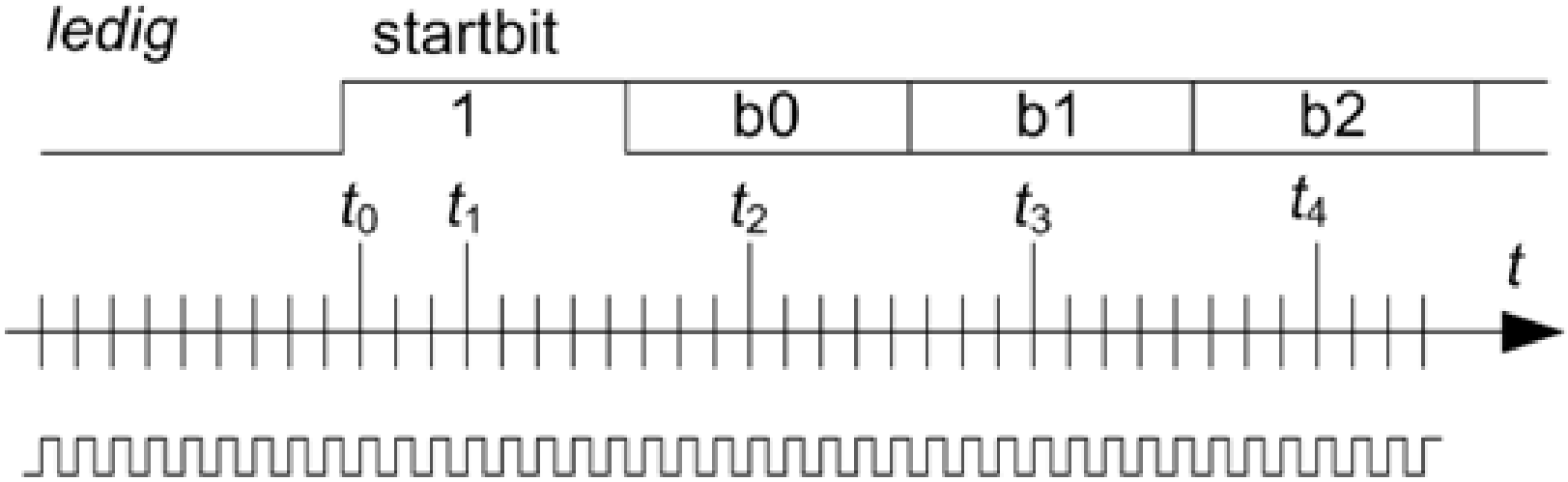
Ett "kommunikationsprotokoll" är en uppsättning regler som tillsammans entydigt specificerar hur datautbytet (datakommunikationen) ska gå till.
Protokollen baseras oftast på någon speciell nätverkstopologi och förutsätter någon speciell *accessmetod*.

Med *acessmetod* (åtkomstmetod) menar man den policy som används när en nod behöver skicka data och därför måste använda kommunikationsmediat. De vanligaste är:

- Master/Slave
- CSMA/CD, *Carrier Sense, Multiple Access, Collission Detect*
- CSMA/CR, *Carrier Sense, Collission Resolution, Multiple Access*
- TDMA , *Time Division, Multiple Access*
- "token", ring-protokoll

Asynkron överföring, Startbitsdetektering



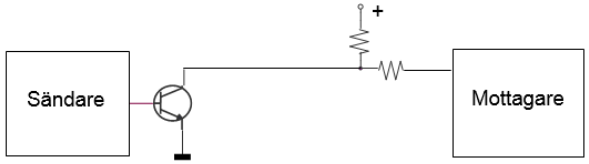


Protokollet anger en ”ledig”-nivå på kommunikationsledningen. Då nivån växlar tolkas detta som en ”startbit” (t_0).

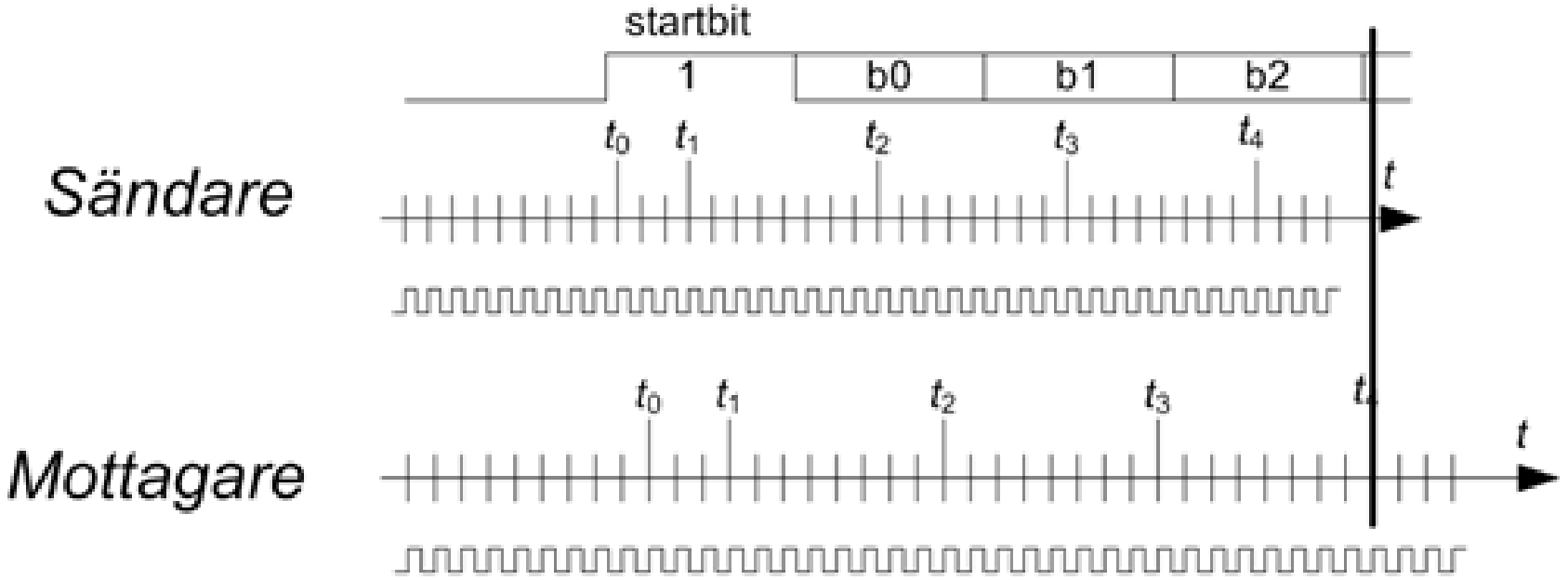
Mottagaren läser därefter av ledningen (”samplar”) efter ett halvt bitintervall (t_1) och därefter ytterligare hela bitintervall (t_2,t_3,t_4 osv).

Observera att sändare och mottagare på förhand måste veta vilken *bitrate* som används.

Klocksynkronisering



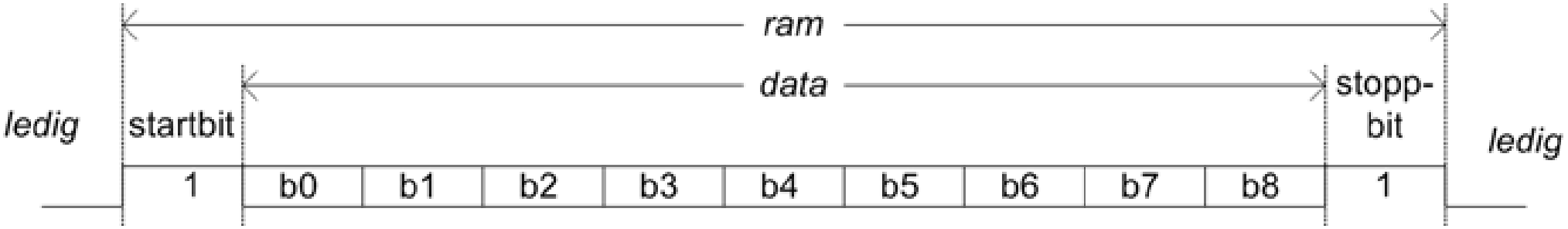
Även om *bitraten* är den samma kan skillnader i klockor orsaka en förskjutning av mottagarens sampling som så småningom resulterar i att fel bitintervall samplas...



Genom att specificera maximalt antal bitar som skickas vid ett tillfälle, en ”ram”, kan man också bestämma en största tillåten differens och därmed också acceptera skillnader hos sändares respektive mottagare klockor.

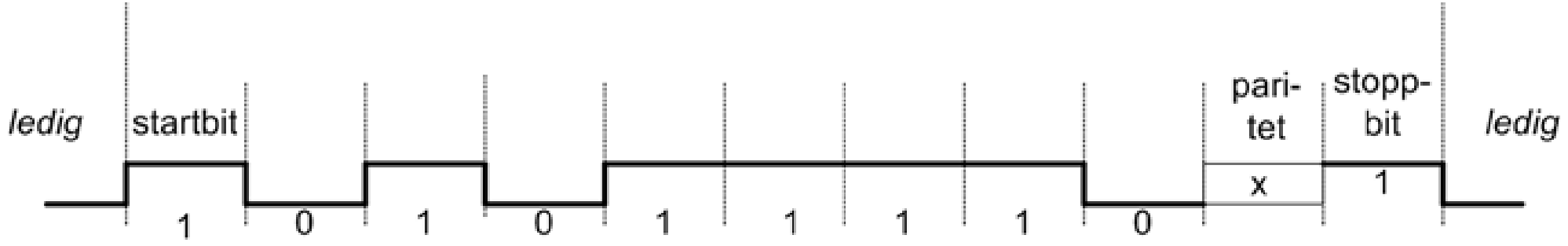
RS232, 9-bitars ram

Exempel på hur en ram kan se ut



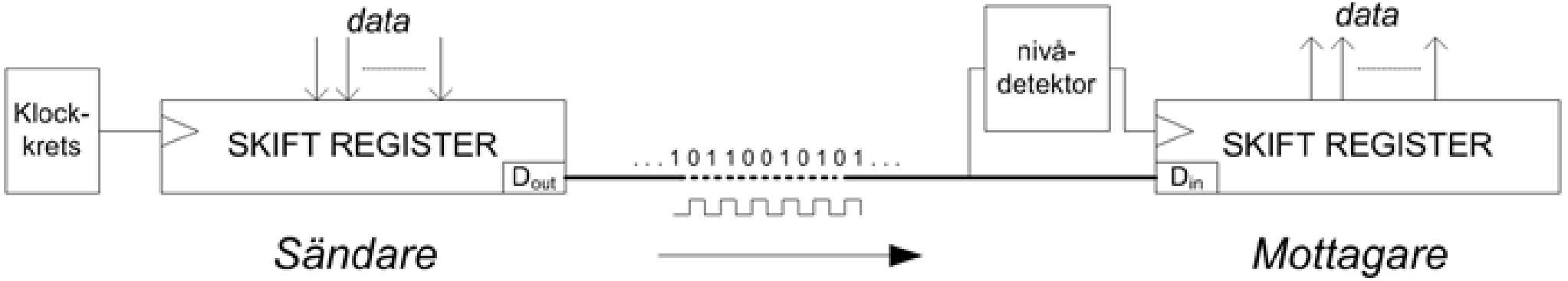
Bitarnas betydelse definieras mera exakt av protokollet

Exempelvis, ”RS232”:
ASCII-tecknet ’z’ (= 01111010=0x7A, LSB först) överförs då på följande sätt

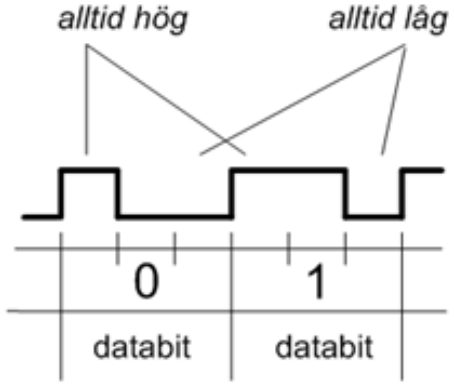


Synkron överföring

Gemensam klocksignal och data



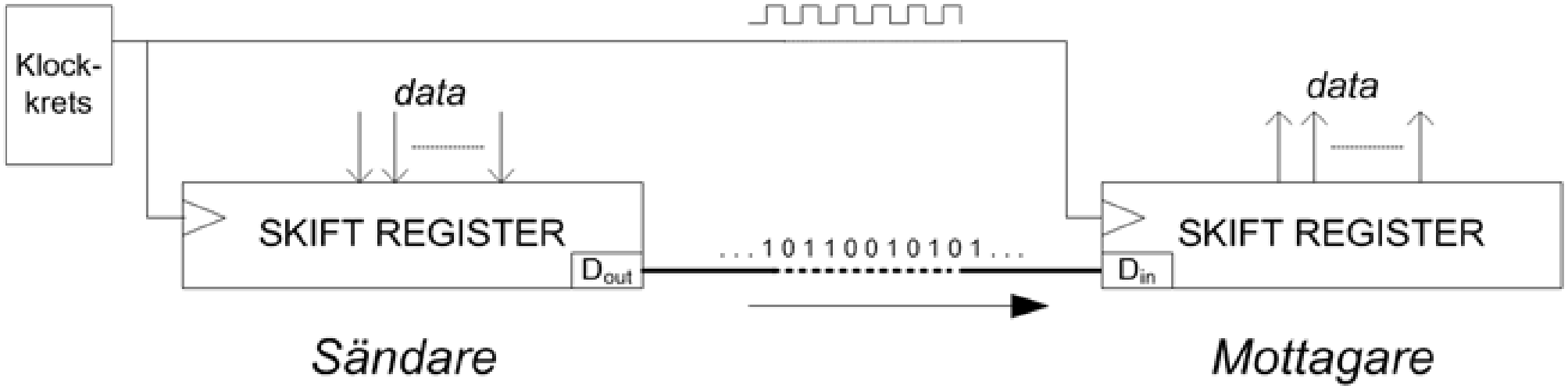
EXEMPEL: Manchester kodning, bitlängden bestäms av positiva flanker, nivån i mitten av biten anger 0 eller 1.
Metoden kallas *RZ* (Return to Zero)



Kräver bara en ledning men minskar bandbredden.

Synkron överföring

Klocksignal och data på skilda ledningar



Kräver två ledningar men ökar bandbredden.

Används för enklare tillämpningar, industristandarder:
SPI (Serial Peripheral Interface)
I^2C använder denna teknik.

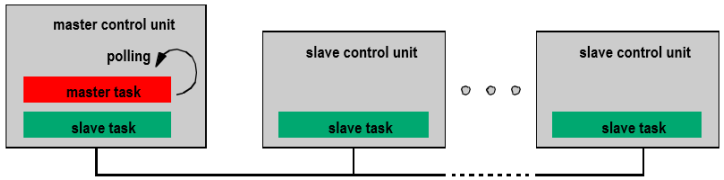
USART – Universal Synchronous/Asynchronous Receiver/Transmitter

Fysiskt gränssnitt (*transceiver*)

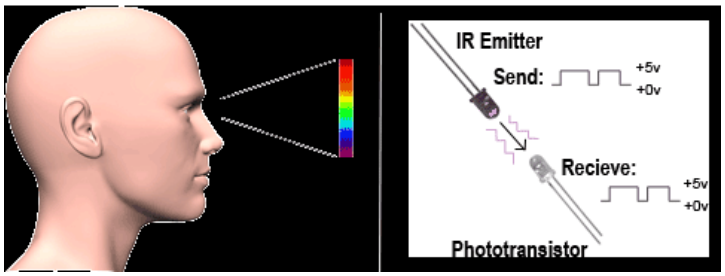
USART

⇔ RS232 (*PtoP, duplex*)

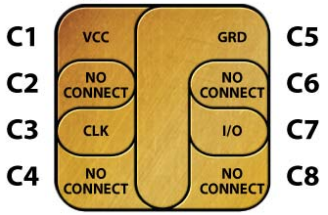
⇔ LIN (*asynkron buss*)



⇔ IR(*trådlöst*)



⇔ Smartcard (*synkron buss*)



USART – RS232

Ofta kommuniceras UTF8 (ASCII-) tecknen

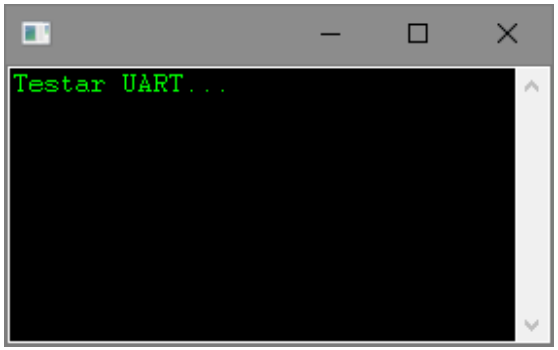
För teckenkoder 0-0x7F är ASCII och UTF-8 identiska.
American Standard Code for Interchange of Information.
Unicode Transformation Format.

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
0	NUL	20		40	@	60	
1	SOH	21	!	41	A	61	a
2	STX	22	"	42	B	62	b
3	ETX	23	#	43	C	63	c
4	EOT	24	$	44	D	64	d
5	ENQ	25	%	45	E	65	e
6	ACK	26	&	46	F	66	f
7	BEL	27	'	47	G	67	g
8	BS	28	(	48	H	68	h
9	HT	29	)	49	I	69	i
A	LF	2A	*	4A	J	6A	j
B	VT	2B	+	4B	K	6B	k
C	FF	2C	,	4C	L	6C	l
D	CR	2D	-	4D	M	6D	m
E	SO	2E	.	4E	N	6E	n
F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[Å	7B	{ ä
1C	FS	3C	<	5C	\ Ö	7C	| ö
1D	GS	3D	=	5D	] Ä	7D	} å
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

Förklaring av ASCII 01-1F

ACK	Acknowledge	GS	Group Separator
BEL	Bell	HT	Horizontal Tabulation
BS	Backspace	LF	Line Feed
CAN	Cancel	NAK	Negative Acknowlege
CR	Carriage Return	NUL	Null
DC	Device Control	RS	Record Separator
DEL	Delete	SI	Shift-In
DLE	Data Link Escape	SO	Shift-Out
EM	End of Medium	SOH	Start of Heading
ENQ	Enquiry	SP	Space
EOT	End of Transmission	STX	Start of Text
ESC	Escape	SUB	Substitute
ETB	End of Transmission Block	SYN	Synchronous Idle
ETX	End of Text	US	Unit Separator
FF	Form Feed	VT	Vertical Tabulation
FS	File Separator		

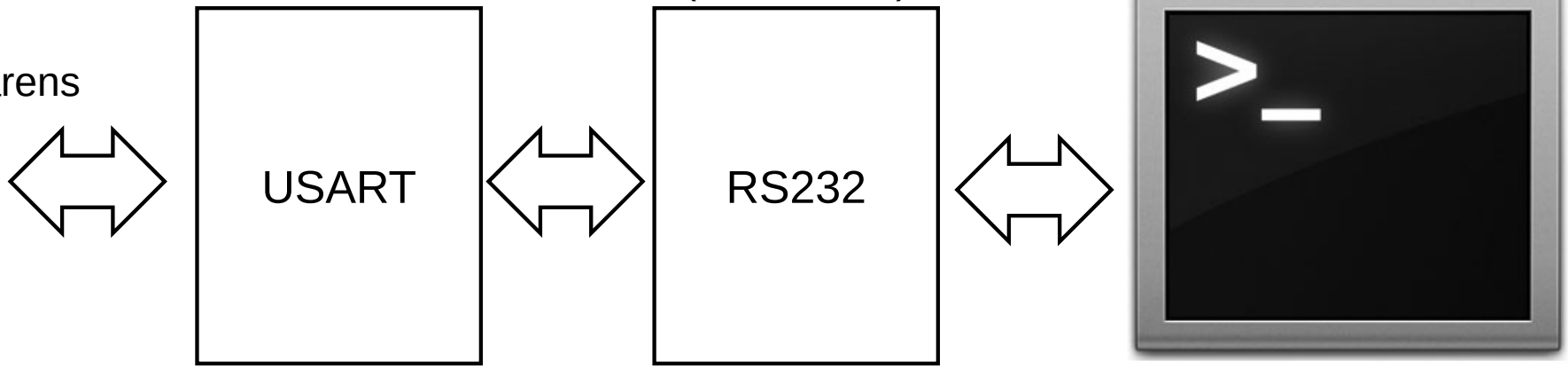
"Terminal"- konsollfönster



Det finns 8 olika seriekretsar...

4001 1400 - 4001 17FF	USART6
4001 1000 - 4001 13FF	USART1
4000 7C00 - 4000 7FFF	UART8
4000 7800 - 4000 7BFF	UART7
4000 5000 - 4000 53FF	UART5
4000 4C00 - 4000 4FFF	UART4
4000 4800 - 4000 4BFF	USART3
4000 4400 - 4000 47FF	USART2

Programmerarens bild



Fysiskt gränssnitt (*transceiver*)

Terminal

USART – programmerarens bild, som en struct...

Registeruppsättning

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																																	USART_SR
4																																	USART_DR
8																																	USART_BRR
0xC																																	USART_CR1
0x10																																	USART_CR2
0x14																																	USART_CR3
0x18																																	USART_GTPR

4001 1400 - 4001 17FF	USART6
4001 1000 - 4001 13FF	USART1
4000 7C00 - 4000 7FFF	UART8
4000 7800 - 4000 7BFF	UART7
4000 5000 - 4000 53FF	UART5
4000 4C00 - 4000 4FFF	UART4
4000 4800 - 4000 4BFF	USART3
4000 4400 - 4000 47FF	USART2

```
typedef struct tag_usart
{
  volatile unsigned short sr;
  volatile unsigned short Unused0;
  volatile unsigned short dr;
  volatile unsigned short Unused1;
  volatile unsigned short brr;
  volatile unsigned short Unused2;
  volatile unsigned short cr1;
  volatile unsigned short Unused3;
  volatile unsigned short cr2;
  volatile unsigned short Unused4;
  volatile unsigned short cr3;
  volatile unsigned short Unused5;
  volatile unsigned short gtpr;
} USART;

#define USART1  ((USART *) 0x40011000)
#define USART2  ((USART *) 0x40004400)
#define USART3  ((USART *) 0x40004800)
#define USART4  ((USART *) 0x40004C00)
```

osv...

USART Dataregister

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		Register
4							R9	R8	R7	R6	R5	R4	R3	R2	R1	R0	RDR	USART_DR
							T9	T8	T7	T6	T5	T4	T3	T2	T1	T0	TDR	

Läsning: RDR, Receive data register
Innehåller det senast anlända tecknet från serielänken

Skrivning TDR, Transmit data register
Tecken som skickas ut på serielänken

USART Statusregister

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0							CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE	USART_SR

Bit 7 ;**TXE**, Transmit data register empty
är 1 om TDR tomt (ledigt)
är 0 om tecken håller på att skickas (upptaget)
nollställs vid skrivning till DR

```
OUTCHAR(c):
// Vänta tills TXE är 1
// Skriv 'c' till dataregistret
```

Bit 5 ; **RXNE**, Receive data register not empty
är 1 om nytt tecken finns i RDR
är 0 om inget nytt tecken, nollställs vid läsning från DR

```
TSTCHAR: returnera c
// Är RXNE=0 ?
// Ja, returnera 'c' = 0
// Nej, returnera 'c' från dataregistret
```

EXEMPEL: Enkla in- och utmatningsrutiner med USART

Skapa funktioner:

`void _outchar(char c)` som matar ut ASCII-tecknet 'c' till en terminal.

`char _tstchar(void)` som kontrollerar om något tecken anlänt från terminalen, i så fall returnerar detta, annars returneras 0.

`char _inchar(void)` som väntar tills något tecken anlänt från terminalen, och returnerar detta.

För teckenkoder 0-0x7F är ASCII och UTF-8 identiska.
American Standard Code for Interchange of Information.
Unicode Transformation Format.

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
0	NUL	20		40	@	60	`
1	SOH	21	!	41	A	61	a
2	STX	22	"	42	B	62	b
3	ETX	23	#	43	C	63	c
4	EOT	24	$	44	D	64	d
5	ENQ	25	%	45	E	65	e
6	ACK	26	&	46	F	66	f
7	BEL	27	'	47	G	67	g
8	BS	28	(	48	H	68	h
9	HT	29	)	49	I	69	i
A	LF	2A	*	4A	J	6A	j
B	VT	2B	+	4B	K	6B	k
C	FF	2C	,	4C	L	6C	l
D	CR	2D	-	4D	M	6D	m
E	SO	2E	.	4E	N	6E	n
F	S1	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[Å	7B	{ ä
1C	FS	3C	<	5C	\ Ö	7C	| ö
1D	GS	3D	=	5D	] Å	7D	} å
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

Förklaring av ASCII 01-1F

ACK	Acknowledge	GS	Group Separator
BEL	Bell	HT	Horizontal Tabulation
BS	Backspace	LF	Line Feed
CAN	Cancel	NAK	Negative Acknowlege
CR	Carriage Return	NUL	Null
DC	Device Control	RS	Record Separator
DEL	Delete	SI	Shift-In
DLE	Data Link Escape	SO	Shift-Out
EM	End of Medium	SOH	Start of Heading
ENQ	Enquiry	SP	Space
EOT	End of Transmission	STX	Start of Text
ESC	Escape	SUB	Substitute
ETB	End of Transmission Block	SYN	Synchronous Idle
ETX	End of Text	US	Unit Separator
FF	Form Feed	VT	Vertical Tabulation
FS	File Separator		

Implementering

USART Statusregister

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0							CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE	USART_SR

TSTCHAR: returnera c
// Är RXNE=0 ?
// Ja, returnera 'c' = 0
// Nej, returnera 'c' från dataregistret

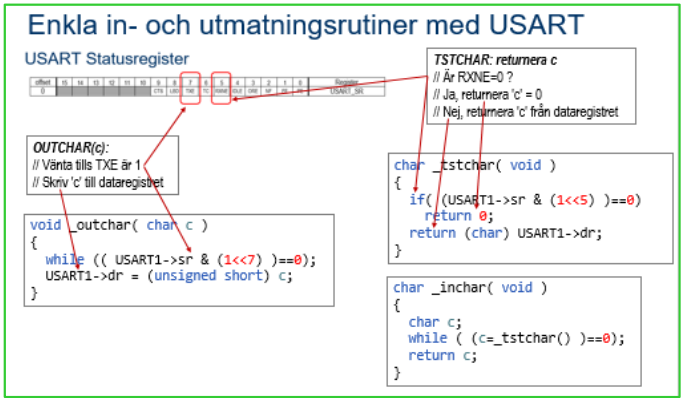
OUTCHAR(c):
// Vänta tills TXE är 1
// Skriv 'c' till dataregistret

```
void _outchar( char c )
{
  while (( USART1->sr & (1<<7) )==0);
  USART1->dr = (unsigned short) c;
}
```

```
char _tstchar( void )
{
  if( (USART1->sr & (1<<5) )==0)
    return 0;
  return (char) USART1->dr;
}
```

```
char _inchar( void )
{
  char c;
  while ( (c=_tstchar() )==0);
  return c;
}
```

Demonstration



File Edit View Search Workspace Build Debugger Plugins Perspective Settings PHP Help

usart_polling.c

```
void    _outchar( char c )
{
    while (( USART1->sr & (1<<7))==0);
    USART1->dr = (unsigned short) c;
}

char    _tstchar( void )
{
    if( (USART1->sr & (1<<5) )==0)
        return 0;
    return (char) USART1->dr;
}


char    _inchar( void )
{
    char c;
    while ( (c=_tstchar() )==0);
    return c;
}



void main(void)
{
    char c;
    while(1){
        c = _inchar();
        _outchar( c );
    }
 }
```

Ln 88, Col 15, Sel 2 SPACES LF C++ UTF-8

USART feldetektering

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0							CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE	USART_SR

Bit 3 ORE: Overrun Error
Denna bit sätts av hårdvaran om ett nytt tecken anländer samtidigt som det finns ett oläst tecken i dataregistret ("overrun error"). Ett avbrott genereras om RXNEIE = 1 i USART_CR1. Det återställs av en läsning från USART_SR följt av en läsning från USART_DR.
0: Inget förlorat tecken
1: Mottaget tecken är överskrivet (förlorat)

Bit 1 FE: Framing Error
Denna bit sätts av hårdvara när ett ramfel, oftast orsakat av förlorad synkronisering, upptäckts. Biten återställs av en läsning från USART_SR följt av en läsning från USART_DR.
0: Inget ramfel upptäcks
1: Ramfel eller BREAK-ram detekterad

Bit 2 NF: Noise detection Flag
Denna bit sätts av hårdvara när störningar i form av brus upptäcks i en mottagen ram. Biten återställs av en läsning från USART_SR följt av en läsning från USART_DR.
0: Ingen störning detekterad
1: Störning detekterad

Bit 0 PE: Parity Error
Denna bit sätts av hårdvara när ett paritetsfel uppträder hos mottagaren. Biten återställs av en läsning från USART_SR följt av en läsning från USART_DR. Programmet måste vänta på att RXNE-biten ettställts innan PE-biten återställs. Ett avbrott genereras om PEIE = 1 i USART_CR1.
0: Inget paritetsfel
1: Paritetsfel

EXEMPEL

Följande sekvens kontrollerar om något fel uppstått efter att förra tecknet tagits emot:

```
if(  USART1->sr & 0xF )
{  /* någon felindikator är aktiv */
}
```

USART initiering (CR1, CR2 och CR3)

USART_CR1 Control register 1 (vid RESET: 0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0xC	OVER8	Res.	UE	M	WAKE	PCE	PS	PEIE	TXE IE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	USART_CR1

```
#define     UE      (1<<13)
#define     TE      (1<<3)
#define     RE      (1<<2)
```

```
USART1->cr1 = UE | TE | RE;
```

USART_CR2 innehåller huvudsakligen kontroll för funktioner då USART implementerar LIN-protokollet.

USART_CR2 Control register 2 (vid RESET: 0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x10	Res.	LIN EN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBD IE	LBDL	Res.	ADD[3:0]				USART_CR2

USART_CR3 innehåller kontroll för handskakningssignaler (RS232) och för funktioner då USART implementerar SmartCard eller IR-protokoll.

USART_CR3 Control register 3 (vid RESET: 0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x14	Res.				ONEBIT	CTS IE	CTSE	RTSE	DMAT	DMA R	SCEN	NACK	HDSEL	IRLP	IREN	EIE	USART_CR3

USART initiering av "baudrate"

USART_BRR Baudrate register (vid RESET: 0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
8	DIV_Mantissa[11:0]												DIV_Fraction[3:0]				USART_BRR

Baudraten för standard och SPI-mod bestäms av:

$$\text{Baudrate} = f_{CK} / (8 \times (2 - \text{OVER8}) \times \text{USARTDIV})$$

där f_{CK}=168/2=84 MHz

USART_CR2 Control register 2 (vid RESET: 0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x10	Res.	LIN EN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBD IE	LBDL	Res.	ADD[3:0]				USART_CR2

USART_CR3 Control register 3 (vid RESET: 0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0x14	Res.				ONEBIT	CTS IE	CTSE	RTSE	DMAT	DMA R	SCEN	NACK	HDSEL	IRLP	IREN	EIE	USART_CR3

USART_CR1 Control register 1 (vid RESET: 0x0000)

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0xC	OVER8	Res.	UE	M	WAKE	PCE	PS	PEIE	TXE IE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	USART_CR1

EXEMPEL: Sätt baudrate hos MD407 till 115200 baud

1. Bestäm USART_BRR för baudrate 115200, OVER8=0

   $\text{USARTDIV} = f_{CK} / (\text{Baudrate} \times 16)$

   $\text{USARTDIV} = 84 \times 10^6 / (115\,200 \times 16) = 45{,}573$

2. Konvertera till hexadecimala tal:

   $45_{10} = 2D_{16}$

   $0{,}573_{10} \approx 9/16$

3. Sätt baudrate-registret:

```
USART1->brr = 0x2D9;
```

```
void    _init(void)
{
    USART1->brr = 0x2D9;
    USART1->cr2 = 0;
    USART1->cr3 = 0;
    USART1->cr1 = UE | TE | RE;
}
```

USART med avbrott

Att överföra ett 10-bitars ord med bithastigheten 115200 baud tar ca 80 µs.

Antag pessimistiskt att en genomsnittlig instruktion tar 4 klockcykler, cykeltiden (168 MHz är cirka 6 ns, dvs. en genomsnittlig instruktion något mindre än 24 ns.

Det innebär att den väntetid som krävs mellan överföringen av två tecken motsvarar $80 \times 10^{-6}/24 \times 10^{-9}$, dvs. exekveringstiden för ca 3300 instruktioner.

offset	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Register
0xC	OVER8	Res.	UE	M	WAKE	PCE	PS	PEIE	TXE IE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK	USART_CR1

Bit 7 TXEIE: TXE Interrupt Enable
Biten sätts och nollställs av programvara.
0: Ingen funktion
1: USART avbrott genereras då TXE=1 i USART_SR.

Bit 6 TCIE: TC Interrupt Enable
Biten sätts och nollställs av programvara.
0: Ingen funktion
1: USART avbrott genereras då TC =1 i USART_SR

Bit 5 RXNEIE: RXNE Interrupt Enable
Biten sätts och nollställs av programvara.
0: Ingen funktion
1: USART avbrott genereras då ORE=1 eller RXNE =1 i USART_SR

```
#define     TXEIE     (1<<7)
#define     TCIE      (1<<6)
#define     RXNEIE    (1<<5)
```

EXEMPEL: USART med avbrott

Skapa funktioner:

```
void  usart_outchar(char c)
char  usart_tstchar(void)
void  usart_init(void)
void  usart_irq_routine(void)
```

som initierar USART-kretsen för avbrottsdriven kommunikation med en enkel 1-teckenbuffert för mottagare resp. sändare.

Använd följande testprogram:

```
void main(void)
{
  char  c;
  usart_init();
  while( 1 )
  {
    c = usart_tstchar();
    if ( c ){
      if( c == 'p')
        printlusart("USART program");
      else
        usart_outchar(c);
    }
  }
}
```

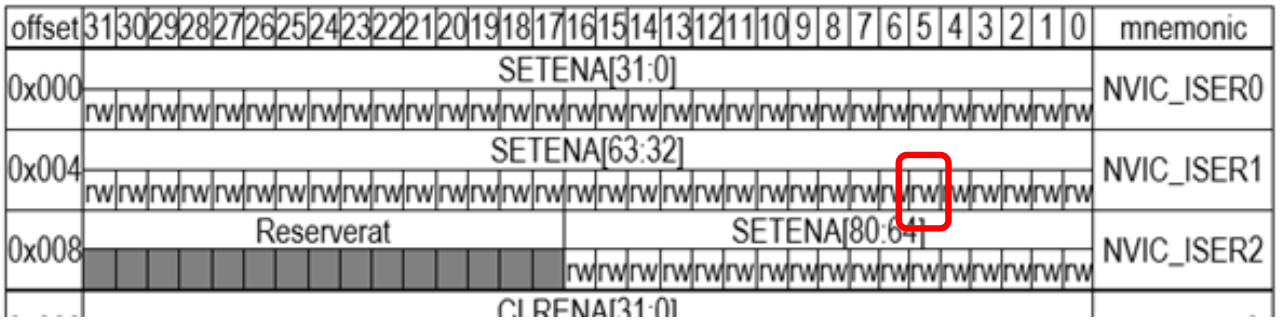
USART1 avbrott (se Quick Guide)

36	settable	SPI2	SPI2 global interrupt	0x0000 00D0
37	settable	USART1	USART1 global interrupt	0x0000 00D4
38	settable	USART2	USART2 global interrupt	0x0000 00D8

IRQ-nummer=37:

```
#define USART1_IRQVEC   0x2001C0D4
```

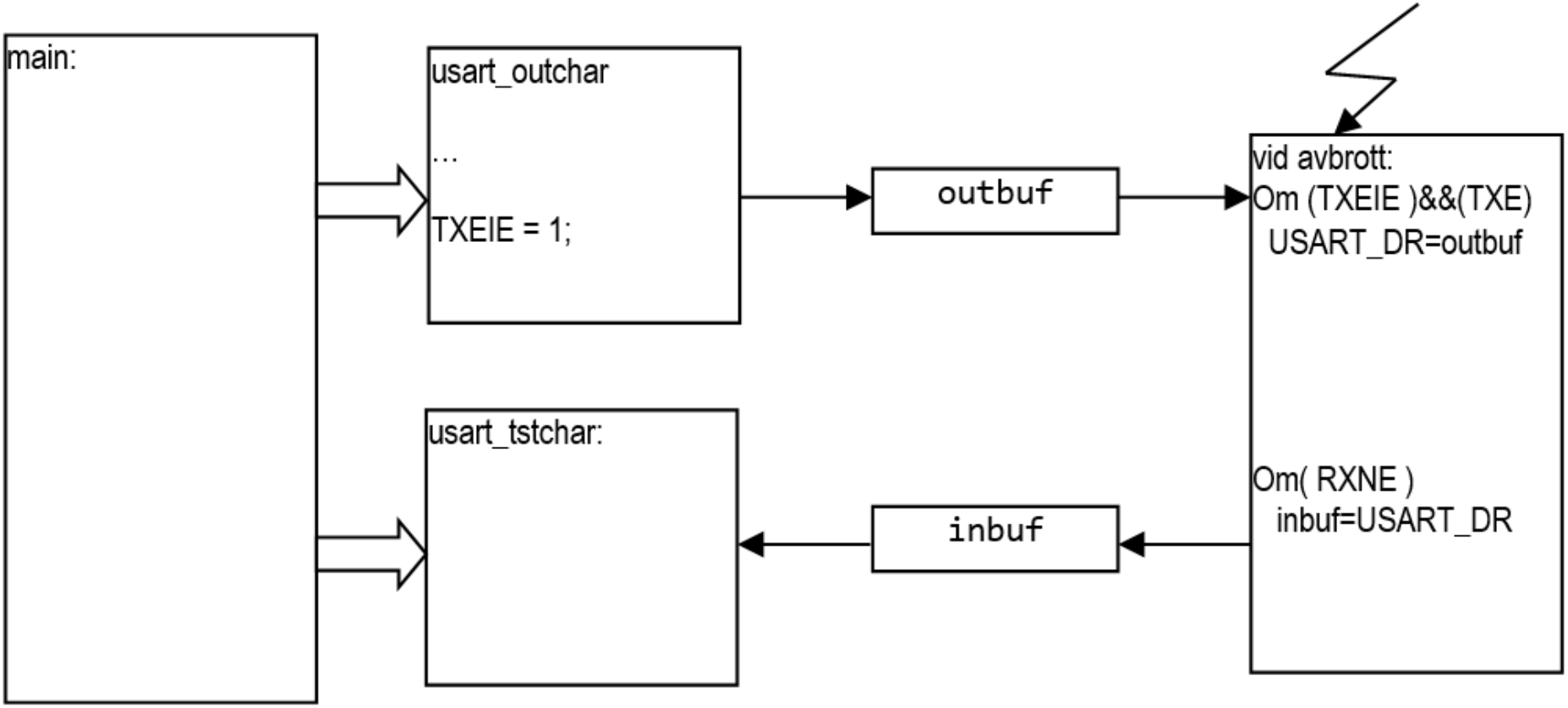
0xE000E100
0xE000E104
0xE000E108



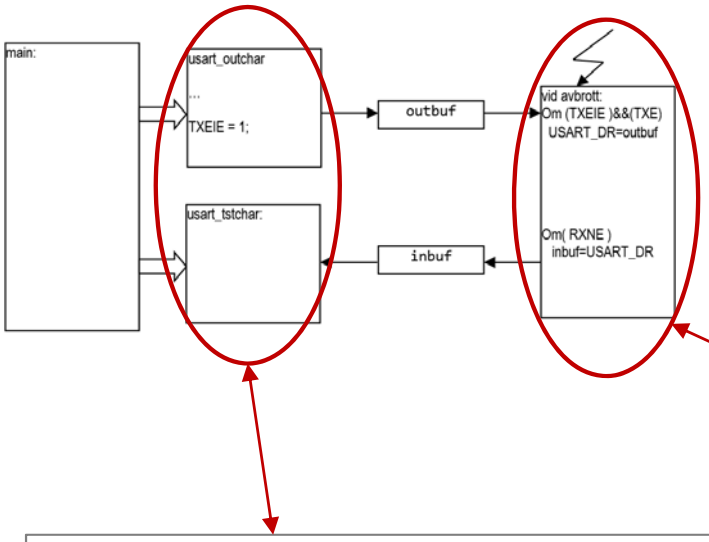
37/32 = 1 → ISER1
Bit = 37-32 = 5 → (1<<5)
Dvs. bit 5 i register NVIC_xxx1

```
#define NVIC_ISART1_ISER       0xE000E104
#define NVIC_USART1_IRQ_BPOS   (1<<5)
```

USART med avbrott, programstruktur



USART med avbrott, implementering




```
static char inbuf, outbuf;

void usart_init( void )
{
  *((void (**)(void) ) USART1_IRQVEC )  = usart_irq_routine;
  inbuf= 0;
  *((unsigned int *) NVIC_USART1_ISER) |= NVIC_USART1_IRQ_BPOS;

  USART1->brr = 0x2D9;
  USART1->cr3 = 0;
  USART1->cr2 = 0;
  USART1->cr1 = UE | RXNEIE | TE | RE;
}
```

```
char usart_tstchar ( void )
{
  char c = inbuf;
  inbuf = 0;
  return c;
}

void       usart_outchar( char c )
{
  outbuf = c;
  USART1->cr1 |= TXEIE;
}
```

```
void usart_irq_routine( void )
{
  if(( USART1->cr1 & TXEIE) && (USART1->sr & TXE))
  {
    USART1->dr = (unsigned short) outbuf;
    USART1->cr1 &= ~TXEIE;
  }
  if( USART1->sr & RXNE)
    inbuf = (char) USART1->dr;
}
```

Demonstration: USART med avbrott

EXEMPEL: USART med avbrott

Skapa funktioner:

```
void  usart_outchar(char c)
char  usart_tstchar(void)
void  usart_init(void)          som initierar USART-kretsen för avbrottsdriven kommunikation
void  usart_irq_routine(void)   med en enkel 1-teckenbuffert för mottagare resp. sändare.
```

Använd följande testprogram:

```
void main(void)
{
  char  c;
  usart_init();
  while( 1 )
  {
    c = usart_tstchar();
    if ( c ){
      if( c == 'p')
        printlusart("USART program");
      else
        usart_outchar(c);
    }
  }
}
```