ECE3411 – Fall 2015

Lecture 2a.

UART: Universal Asynchronous Receiver & Transmitter

Marten van Dijk, Syed Kamran Haider
Department of Electrical & Computer Engineering
University of Connecticut
Email: {vandijk, syed.haider}@engr.uconn.edu

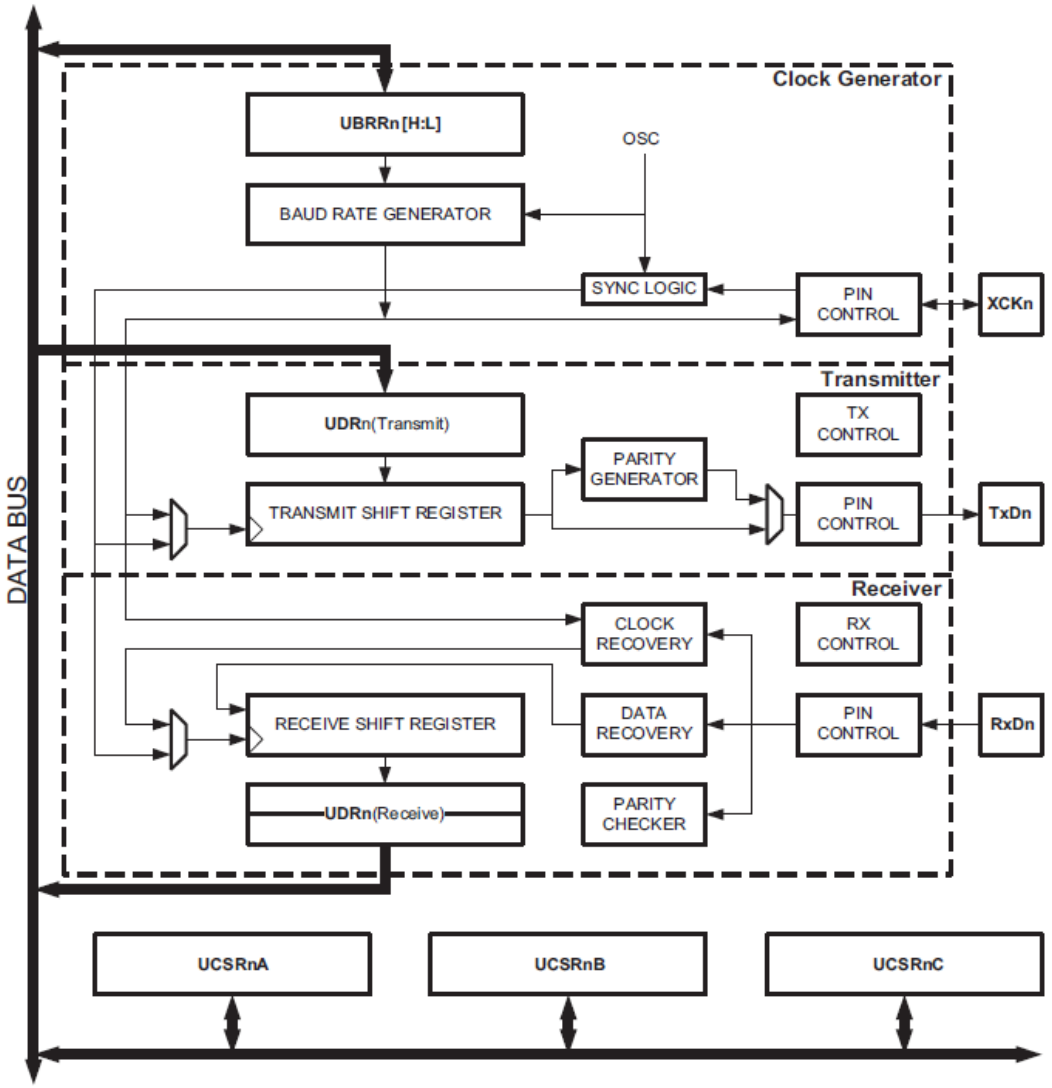
UCONN

Based on the Atmega328P datasheet and material from Bruce Land's video lectures at Cornel

USART0 (Ch. 19 ATmega328P Datasheet)

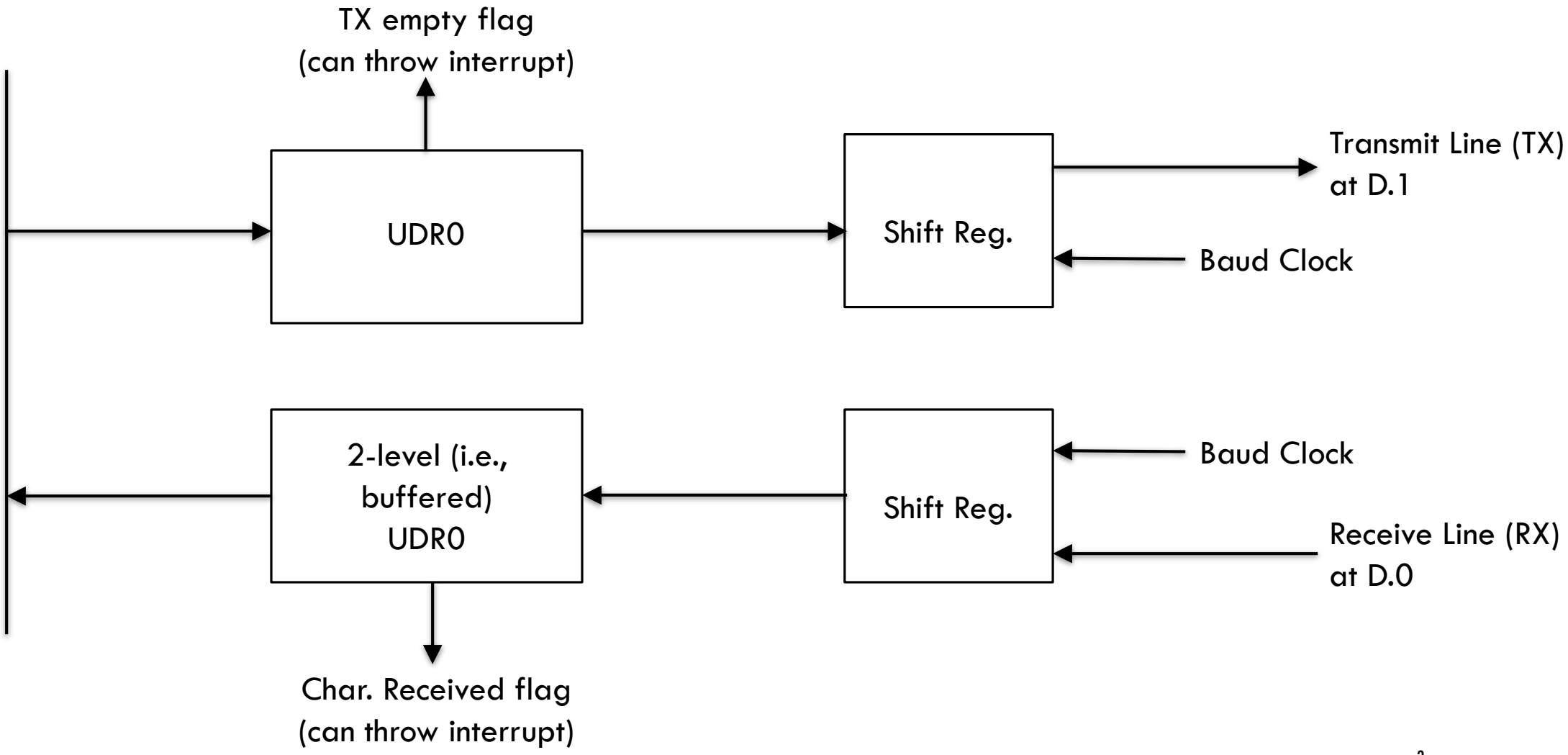
- USART = Universal Synchronous and Asynchronous serial Receiver and Transmitter
- Clock generator, Transmitter, Receiver
- Bolted on to the MCU

Figure 19-1. USART Block Diagram[1]

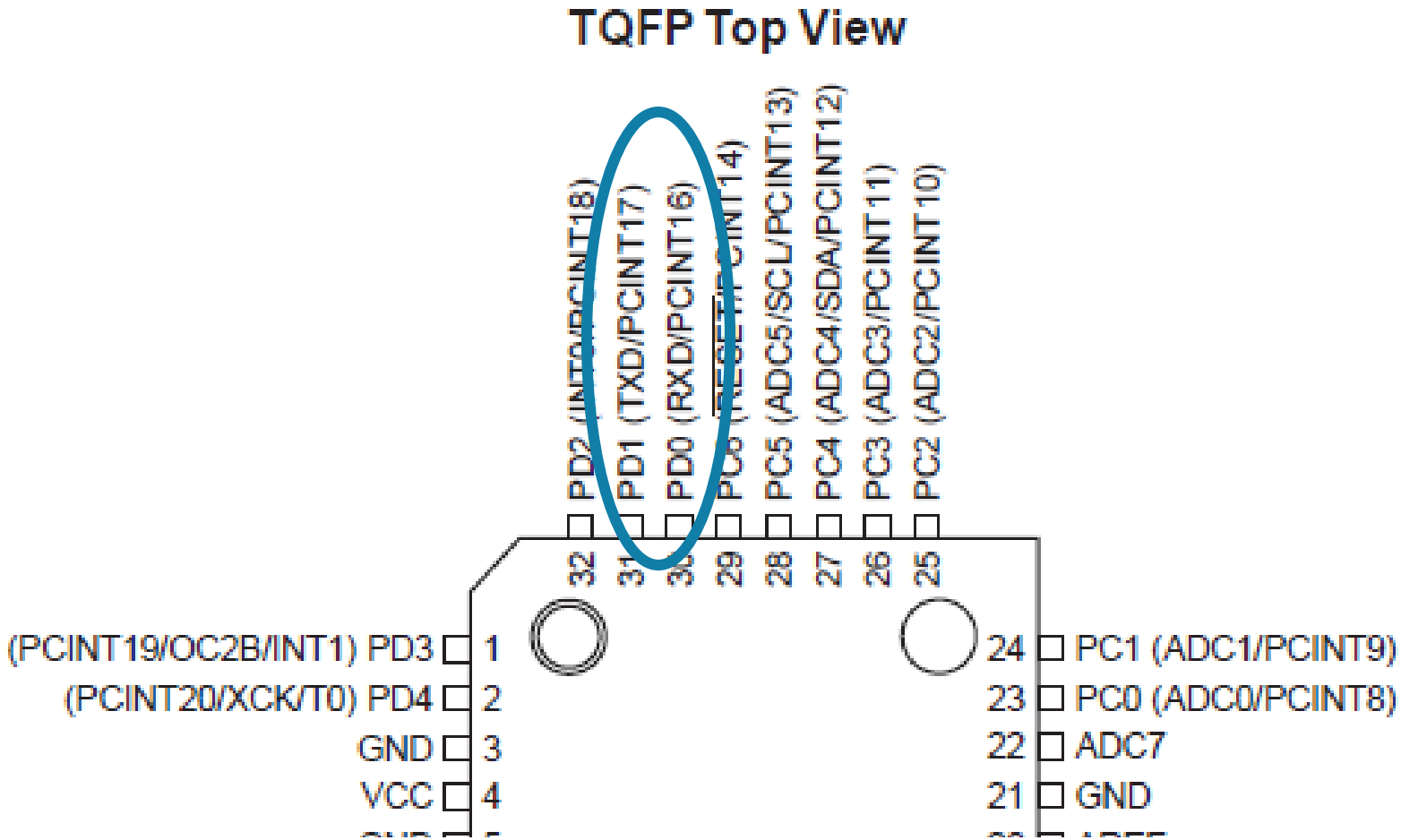


Control registers

USART


TX empty flag
(can throw interrupt)
UDR0
Shift Reg.
Transmit Line (TX)
at D.1
Baud Clock
2-level (i.e.,
buffered)
UDR0
Shift Reg.
Baud Clock
Receive Line (RX)
at D.0
Char. Received flag
(can throw interrupt)

TX and RX at PORTD



USART

- USART communicates over a 3-wire cable: TX, RX, Gnd
- Designed for a mechanical printer, a long time ago; protocol is slow
- HW allows full-duplex, i.e., HW can transmit and receive at exactly the same time
  - Need interrupt to utilize this in SW
- Baud rate in bits per second: 9600 Bd is approximately 0.1ms per bit
  - This is slow: Therefore, in SW start transmitting a character, then do something else!
  - In theory the Baud rate can be very large (1Mbit per second) but this can only be realized between MCUs
  - The used cable limits the maximum possible Baud rate
- Per bit the receiving clock makes 4 measurements and they all need to match: All, e.g. 10, bits within a frame give 40 measurements that all need to match
  - The Baud rates of the receiving and transmitting devices need to match within $1/40 = 2.5\%$

UBRR0H and UBRR0L

- Baud rate is translated relative to the system oscillator clock frequency f_OSC to two registers UBRR0H and UBRR0L, the high and low value of UBRR0 which is in the range [0,4095]

Table 19-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0) 4 samples per bit	$BAUD = \frac{f_{OSC}}{16(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1) 2 samples per bit	$BAUD = \frac{f_{OSC}}{8(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

UBRR0H and UBRR0L

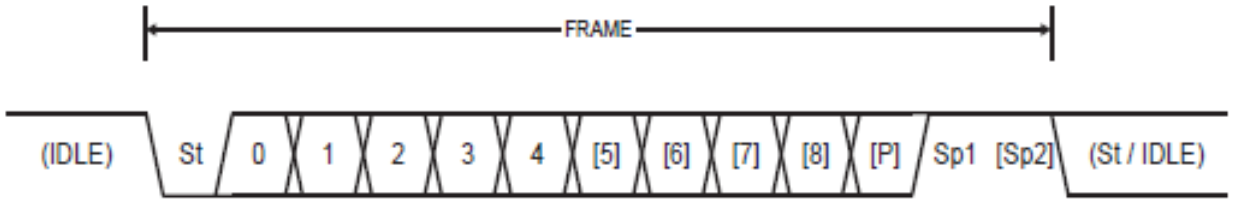
19.10.5 UBRRnL and UBRRnH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Frame Format

- To transmit a byte (i.e., one char) we need at least one start bit (receiving clock starts when falling edge is received), 8 data bits, and one stop bit: Total of 10 bits.

Figure 19-4. Frame Formats



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxDn or TxDn). An IDLE line must be high.

UDR0 for Transmission and Receiving

19.10.1 UDRn – USART I/O Data Register n

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDRn. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDRn Register location. Reading the UDRn Register location will return the contents of the Receive Data Buffer Register (RXB).

(The receive and transmit buffers RXB and TXB are different in HW; in SW their names, i.e. I/O addresses, are the same. The shared name UDR0 in read mode means that RXB is read, and UDR0 in write mode means that TXB is written. Notice that reading and writing of bits in UDR0 can be done simultaneously since they affect different hardware buffers!)

Control register: UCR0A

19.10.2 UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- RXC0: Receive character complete → There is something in the receive register worth reading
- TXC0: Transmit character compare → Is set when both entries in the Transmit Shift Register and Transmit Buffer (UDR0) are shifted out → Not very useful
- UDRE0: Transmit data empty → Goes high when 1 of the two buffers (see above) is empty → Time to refill
- FE0: Frame error if 4 samples of a bit do not match → Detects bad clock rate
- DOR0: Data overrun: If a new character is complete and RXC0 is still set, implies a lost char → SW did not read often enough

Control register: UCR0A

19.10.2 UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- UPE0: Parity error
- U2X0: Double speed (twice the baud rate) → reduces error checking (only 2 samples per bit)
- MPCM0: Multiple processor address mode (can connect more than 2 devices to the line)

Control register: UCSR0B

Multiprocessor Stuff

19.10.3 UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- RXCIE0: Receive character complete interrupt enable → You can write an ISR for this
- TXCIE0: Enables interrupt for both members in TX queue being empty
- UDRIE0: Enables interrupt if the first of the output pipeline is empty
- RXEN0: RX enable → Disables D.0 for general I/O (completely overrides any other I/O)
- TXEN0: TX enable → Disables D.1 for general I/O (completely overrides any other I/O)
- UCSZ02: see next slides

Control register: UCSR0C

19.10.4 UCSRnC – USART Control and Status Register n C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **Bits 7:6 – UMSELn1:0 USART Mode Select**

These bits select the mode of operation of the USARTn as shown in Table 19-4.

Table 19-4. UMSELn Bits Settings

UMSELn1	UMSELn0	Mode
0	0	Asynchronous USART
0	1	Synchronous USART
1	0	(Reserved)
1	1	Master SPI (MSPIM)(1)

Control register: UCSR0C

Table 19-5. UPMn Bits Settings

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Table 19-6. USBS Bit Settings

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Table 19-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Table 19-8. UCPOLn Bit Settings

UCPOLn	Transmitted Data Changed (Output of TxDn Pin)	Received Data Sampled (Input on RxDn Pin)
0	Rising XCKn Edge	Falling XCKn Edge
1	Falling XCKn Edge	Rising XCKn Edge

Default: Frames of 10 bits.

Initialization

```
#define F_CPU 16000000UL
#define BAUD 9600
#define MYUBRR F_CPU/16/BAUD-1
int main()
{
    …
    UART_Init(MYUBRR);
    …
}

/* Function Body */
void UART_Init(unsigned int ubrr)
{
    UBRR0H = (unsigned char) (ubrr>>8);
    UBRR0L = (unsigned char) ubrr;
    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
}
```

Transmission (19.6.1 datasheet & uart.c)

```
int uart_putchar(char c, FILE *stream)
{
    /* Alarm (Beep, Bell) */
    if (c == '\a')
    {
      fputs("*ring*\n", stderr);
      return 0;
    }

   /* Newline is translated into a Carriage Return */
   if (c == '\n')  {uart_putchar('\r', stream); return 0;}

   /* In uart.c: loop_until_bit_is_set(UCSR0A, UDRE0); */
   while ( !(UCSR0A & (1<<UDRE0)) ) ;
   UDR0 = c;

   return 0;
}
```

```
/* avr/io.h implements useful macros besides defining
  * names for bit positions, registers like DDx  (or do we
  * use DDRx?) etc.
*/

#define _BV(bit) (1 << (bit))
#define bit_is_set(sfr, bit)        (_SFR_BYTE(sfr) & _BV(bit))
#define bit_is_clear(sfr, bit)  (!(_SFR_BYTE(sfr) & _BV(bit)))
#define loop_until_bit_is_set(sfr, bit)
            do { } while (bit_is_clear(sfr, bit))
#define loop_until_bit_is_clear(sfr, bit)
            do { } while (bit_is_set(sfr, bit))
```

Receiving

- int uart_getchar(FILE *stream) in uart.c is a simple line-editor that allows to delete and re-edit the characters entered, until either CR or NL is entered
- printable characters entered will be echoed using uart_putchar()
  - So you can see the character received by the MCU and you can verify whether the transmission was without error if you recognize the character as the transmitted one (as pressed by the keyboard)
- The core part in uart_getchar is

```
int uart_getchar(FILE *stream)
{
   …
   while ( !(UCSR0A & (1<<RXC0)) ) ;
   c = UDR0;
   …
   uart_putchar(c, stream);
   …
}
```

ASCII Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Using uart.c

```
#include "uart.h“
…
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);
…
int main(void)
{
   uart_init();                                  // Initialize UART
   stdout = stdin = stderr = &uart_str; // Set File outputs to point to UART stream
   ….
   // Can use fprintf and fscanf anywhere: here or in subroutines
   …
   return 0;
}
```