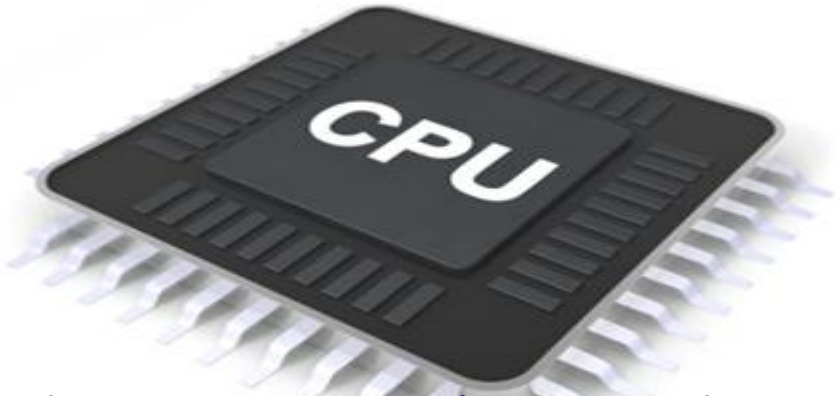


Universal Synchronous Asynchronous Receiver Transmitter

ENEE 3587

Microp Interfacing

Parallel vs Serial Connections

- Digital connections are multi-bit (n-bit) connections (aka bus connections)
- Parallel connection: physical pin is needed for each bit
  - Advantage: fast
  - Disadvantage: cost, power consumption, space restrictions
- Serial connection: few pins for bus connections
  - Convert the n-bits into a sequence of bits in time
  - Advantage: reduced cost, power consumption, space optimized
  - Disadvantage: latency (delay) conversion to/from sequence
  - Speed vs Cost/Power
  - Most MCU apps don’t have a high speed needs but have cost/power/space restrictions

RS-232

- Developed in early 1960s by Electronic Industries Association (EIA)
- Renamed EIA232 in 1992
- Slow but is very widely supported
  - PC manufacturers seem to have dropped serial comm opting for USB.
- Standard covers:
  - Electrical spec: voltage chars, rise/fall time signals, data rates, distance of comm
  - Functional spec: signal functions
  - Mechanical spec: pins, shape and dimensions of the connectors
    - 2 types of connectors: 9-pins (widely used); 25-pins
  - Procedural specs: sequence of events for data transmission and reception

Universal Synchronous Asynchronous Receiver Transmitter (USART)

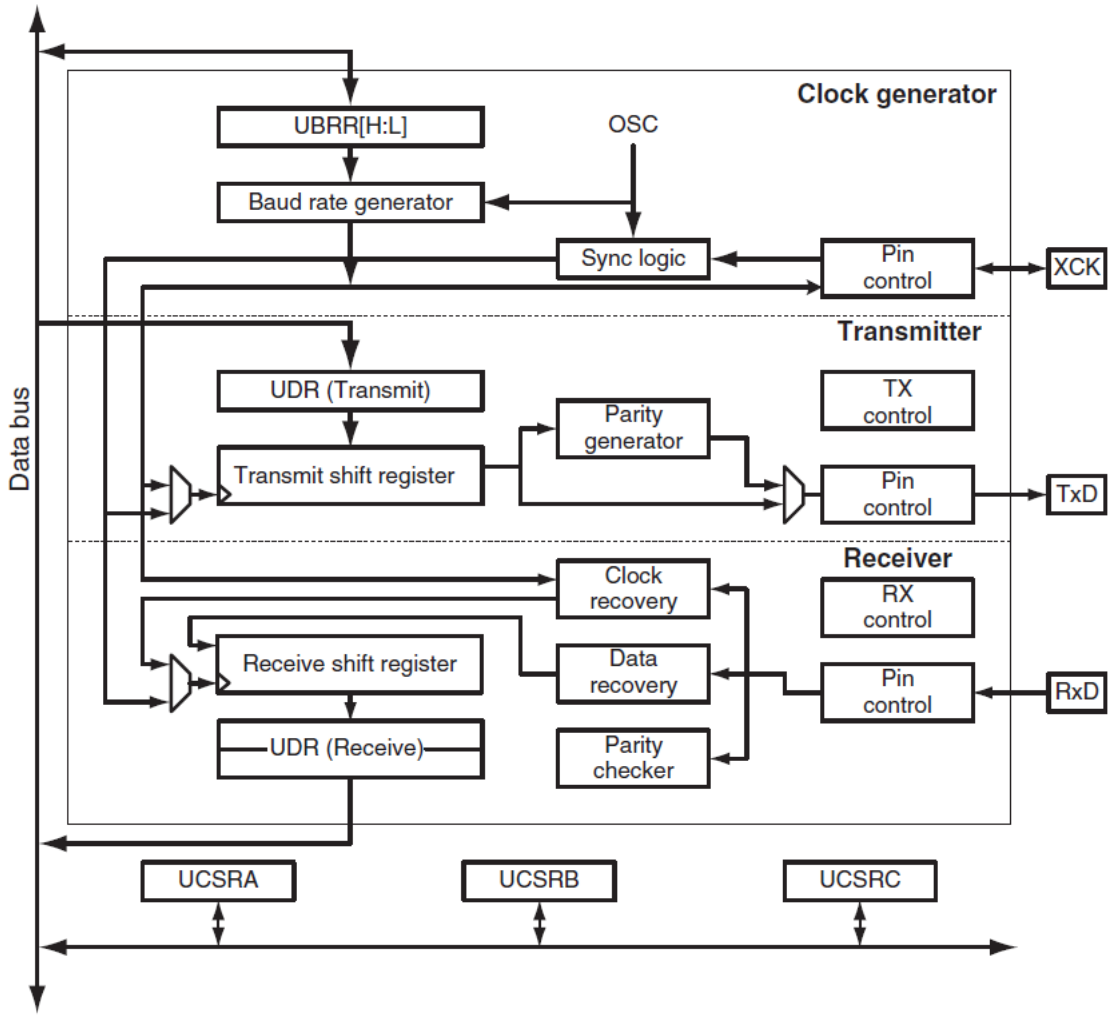
- Asynchronous vs asynchronous:
  - Asynchronous: clock is not transmitted (long distances)
  - Synchronous: clock is transmitted (SPI)
- Asynchronous data format:
  - Start bit – 5/6/7/8/9 data bits – parity bit – 1/2 stop bits
  - Idle state: High
  - Start bit: low
    - Sampled 16x the transmission speed
  - 7/8 data bits: common ASCII needs only 7 bits
    - NRZ transmission
  - Parity: optional error detection
  - Stop bits: framing

Transmission Errors

- Framing error: detected by the absence of the stop bit.
  - Indicates a synchronization problem, faulty transmission, or a break condition.
  - Break: transmission/reception of a logic 0 for one frame or more.
- Receiver overrun: receiver is not ready to receive more data
  - Receiver register(s) are full and have not been completely read.
- Parity error:
  - Odd number of bits have changed value
  - Can’t detect even number of bits

USART Module

- 3 components:
  - Clock generation (XCLK)
    - Syncing the 2 sides of the communication
  - Transmission (TxD)
    - Shifting out data
    - Adding start/stop/parity bits
    - Clock used for shifting
  - Reception (RxD)
    - Shifting the data
    - Detecting start/stop/parity bits
    - Clock used for detection



Mega USART pins

- Eah USART module has 3 pins:
  - Transmission (TX), Receiver (RX), Transmission clock (XCLK)
- USART0:
  - RXD0: PE0, TXD0: PE1, XCK0: PE2
- USART1:
  - RXD1: PD2, TXD1: PD3, XCK1: PD5
- USART2:
  - RXD2: PH0, TXD2: PH1, XCK2: PH2
- USART3:
  - RXD3: PJ0, TXD3: PJ1, XCK3: PJ2

ATmega USART**n** Registers

- **n** = 0,1,2,3
- UBRR**n**: 16-bit USART**n** baud rate register made up of UBRR**n**H, UBRR**n**L
- UCSR**n**A: USART**n** control status register A
- UCSR**n**B: USART**n** control status register B
- UCSR**n**C: USART**n** control status register C
- UDR**n**: USART**n** data register

Baud Rate Generation

- Baud units that measure symbols
  - If the signal represents 1 bit (2 symbols = Hi or LO) only then bit rate = baud rate
- Baud rate depends on mode, baud rate register (UBRR**n**), and clk_{IO}
- Operational modes:
  - Asynchronous normal mode (U2Xn = 0): baud < 16* clk_{IO}

$$\mathrm{UBBR}\mathbf{n} = \frac{f_{clkIO}}{16 \times \mathrm{Baud}} - 1$$

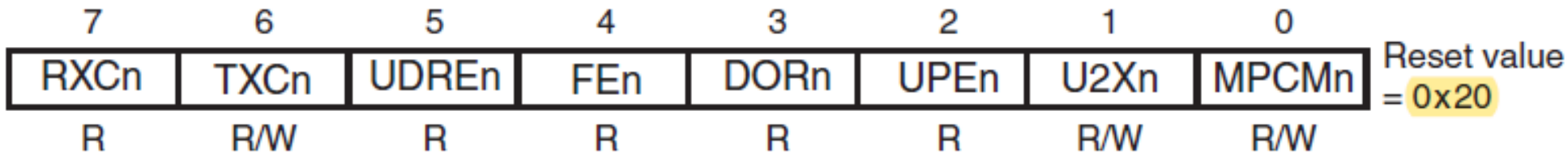
  - Asynchronous double speed mode (U2Xn = 1): baud < 8* clk_{IO}

$$\mathrm{UBBR}\mathbf{n} = \frac{f_{clkIO}}{8 \times \mathrm{Baud}} - 1$$

  - Synchronous master mode: baud < 2* clk_{IO}

$$\mathrm{UBBR}\mathbf{n} = \frac{f_{clkIO}}{2 \times \mathrm{Baud}} - 1$$

UCSRnA



RXCn: USART receive complete

0 = Receive buffer is empty.

1 = There are unread data in the receive buffer.

TXCn: USART transmit complete

0 = There is still data in either the transmit data register or shift register.

1 = There is no data in transmit buffer. Cleared when the corresponding interrupt service routine is entered.

UDREn: USART data register empty

0 = Transmit buffer (UDRn) is not empty.

1 = The transmit buffer is empty and ready to accept new data.

FEn: Framing error

1: receive buffer had a frame error when received (stop bit is 0). This bit is valid until the receive buffer (UDRn) is read.

DORn: Data overrun

1 = Data overrun condition has occurred. This bit will be cleared when the UDRn register is read.

UPEn: USART parity error

1 = received character had a parity error. Parity checking must be enabled.

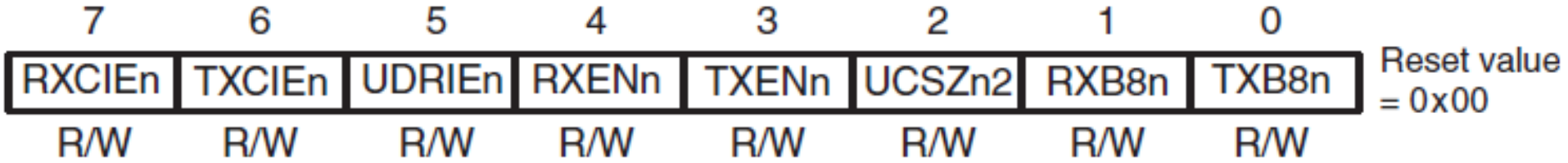
U2Xn: Double the USART transmission speed

0=16x speed, 1= 8x speed

MPCMn: Multi-processor communication mode

1 = Multiprocessor communication mode is enabled.

UCSRnB



RXCIEn: Receive complete interrupt enable

0 = RXCn flag interrupt disabled

1 = RXCn flag interrupt enabled

TXCIEn: transmit complete interrupt enable

0 = TXCn flag interrupt disabled

1 = TXCn flag interrupt enabled

UDRIEn: USART data register empty interrupt enable n

0 = UDREn flag interrupt disabled

1 = UDREn flag interrupt enabled

RXENn: Receiver enable n

Writing this bit to 1 enables USART receiver.

TXENn: Transmitter enable n

Writing this bit to 1 enables the USART transmitter.

Writing this bit to 0 will not disable transmitter until ongoing and pending transmissions are completed.

UCSZn2: Character size bit 2

This bit combined with the UCSZn1:0 (UCSRnC) sets the number of data bits in a frame

RXB8n: Receive data bit 8 n

ninth data bit of received character. Must be read before the lower 8 bits.

TXB8n: Transmit data bit 8 n

ninth data bit of transmitted character. Must be written before the lower 8 bits.

UCSRnC

7	6	5	4	3	2	1	0	
UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	Reset value = 0x00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

- **UMSELn**1:0: USART mode select
  - 00 = Asynchronous USARTn
  - 01 = Synchronous USARTn
  - 10 = Reserved
  - 11 = Master SPI (MSPIM)
- **UPMn**1:0: Parity mode
  - 00 = Disabled
  - 01 = Reserved
  - 10 = Enabled, even parity
  - 11 = Enabled, odd parity
- **USBSn**: Stop bit select
  - 0 = 1 stop bit
  - 1 = 2 stop bits
- **UCSZn**1:0: Character size bits ; with **UCSZn**2 (UCSRnB)
- **UCPOLn**: Clock polarity (used in synchronous mode only)
  - 0 = Transmitted data on the rising edge of XCKn; received data on the falling edge
  - 1 = Transmitted data on the falling edge of XCKn; received data on the rising edge

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	1	1	9-bit
4,5,6 reserved			

USART Operation: Setup

- Setup: Based on application specifications and solution algorithm:
  - Set direction on pins associated with USART
  - UBBR: baud rate
  - UCSRA: transmission speed (8x,16x); multi-processor enable.
  - UCSRB: receiver, transmission enable; interrupt enable; frame size
  - UCSRC: mode, parity, frame size, clock polarity

USART Operation: Transmission

- Write data to be transmitted to UDR**n**
- Flags:
  - UDRE**n** = 1 (UCSRnA = 0x10) => UDRn is ready for data load
  - TXC**n** = 1 (UCSR**n**A = 0x40) => transmission complete
  - Reading/writing in UDR**n** clears flags
- Polling method:

```
while(!(UCSR0A & 0x10));   // wait for empty buffer
UDR0 = data;               // data to be transmitted
```

- Interrupt method:

```
ISR (USART0_UDRE_vect)
{      UDR0 = data;   }    // data to be transmitted
```

USART Operation: Reception

- Read data received from UDRn
- Flags:
  - RXCn (UCSR**n**A = 0x80) => There are unread data in the receive buffer
  - FEn (UCSR**n**A =0x01) => Framing error
  - DORn (UCSR**n**A =0x08) => receiver overrun error
  - UPEn (UCSR**n**A =0x04) => parity error
- Polling method:

```
while(!(UCSR0A & 0x80));   // wait for empty buffer
data = UDR0;               // data to be transmitted
```

- Interrupt method:

```
ISR (USART0_RX_vect)
{      data = UDR0;   }    // data to be transmitted
```

Coding Example 1

- Initialize the USART1 module with the following specs:
  - 8 data bits, 1 stop bit, no parity
  - 19,200 baud rate (16x speed)
  - Transmission and reception enabled
  - Multiprocessor mode disabled
  - Normal asynchronous mode operation
  - Polling method
  - Assuming that clk_{IO} = 16 MHz

```
DDRD   = 0x08;          // PD2 = TX, PD3 = RX USART1
UBRR1  = 51;            // 16M/19200/16 – 1 = 51.08
UCSR1A = 0;             // 16x as divider
UCSR1B = 0x18;          // enable TX, RX, 8-bit data
UCSR1C = 0x06;          // asynchronous, no parity, 1 stop bit
```

Coding Example 2

- Write a function that uses polling method to transmit an 8-bit array on USART1:

```
void putcUSART1 (char data)
{   while(!(UCSR1A & 0x20)); // wait for empty transmit buffer
    UDR1 = data;
}

void putsUSART1(char *ptr)
{   while(*ptr)
    {      putcUSART1(*ptr);
           ptr++;
    }
}
```

Coding Example 3

- Write a function that uses polling method to receive an 8-bit array on USART1. The reception terminates when a 0 is received:

```
char getcUSART1 (void)
{   while(!(UCSR1A & 0x80));  // wait for TX to be complete
    return UDR1;
}

void getsUSART1(unsigned char *ptr)
{   do
    {       *ptr = getcUSART1();
            ptr++;
    }
    while(*(ptr-1));
}
```

n-bits TX/RX

- 5 <= n <= 9
- n < 9: byte data transmissions
- n = 9: byte + 1
  - TX: 9th bit in UCSR**n**B bit 0
  - RX: 9th bit in UCSR**n**B bit 1
  - 9th bit must be read/written first
    - Before remaining bits

Coding Example 4

- Write a function to receive 9-bit using polling on the USART1:

```
unsigned int getc9USART1(void)
{   unsigned hi, lo;
    while(!(UCSR1A & 0x80));          // wait for data to be received
    hi = (UCSR1B & 0x02)>>1;          // read 9th bit USCR1B bit 1
    lo = UDR1;                        // remaining 8 bits
    return (((unsigned int)hi*256) + (unsigned int)loByte);    //int val
}
```