University of San Carlos | Department of
COMPUTER ENGINEERING

CpE 3201
Embedded Systems

Universal Synchronous Asynchronous Receiver Transmitter (USART)

This chapter will cover:

- **Universal Synchronous Asynchronous Receiver Transmitter (USART**) with examples on **asynchronous transmit and receive** using the PIC16F877A USART module.

Universal Synchronous Asynchronous Receiver Transmitter (USART)

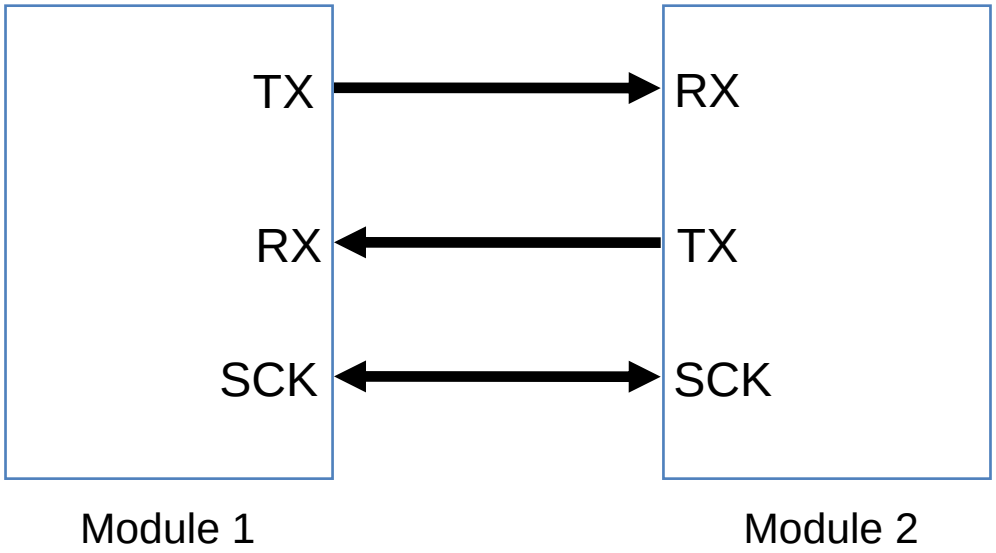
- It is a key component of the **serial communications** subsystem of a computer.
- The USART takes bytes of data and transmits the individual bits in a sequential fashion; at the destination, a second USART re-assembles the bits into complete bytes
- **Not a communication protocol** like SPI and I^2C, but a **physical circuit**.

Serial Transmission

- **Synchronous**
  - requires that the sender and receiver **share a clock** with one another, or that the sender **provide a strobe or other timing signal** so that the receiver knows when to “read” the next bit of the data
  - **usually more efficient** because only data bits are transmitted between sender and receiver, however **can be more costly** if extra wiring and circuits are required to share a clock signal between the sender and receiver

Serial Transmission

(Synchronous)

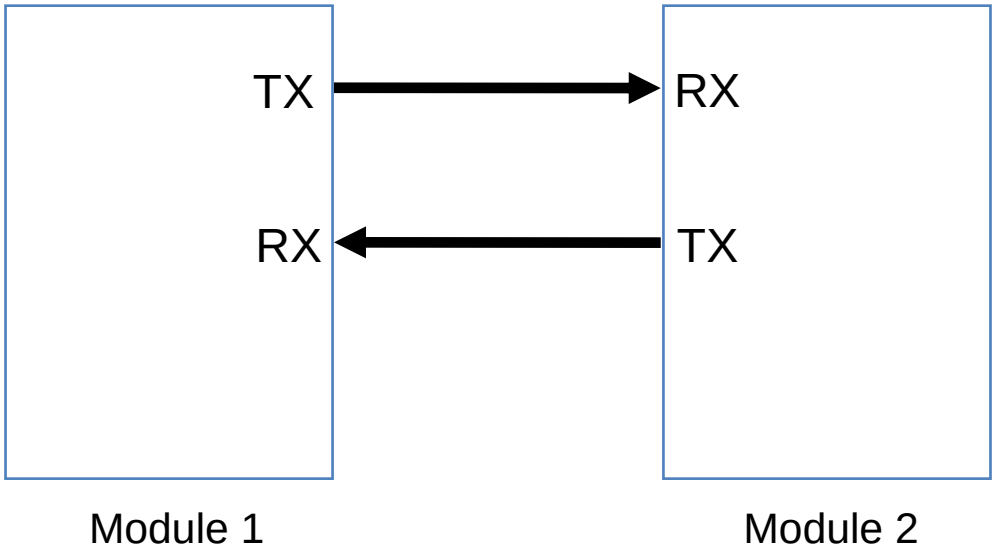


Serial Transmission

- **Asynchronous**
  - allows data to be transmitted **without** the sender **having to send a clock signal** to the receiver. Instead, the sender and receiver must agree on **timing parameters** in advance and **special bits** are added to each word which are used to synchronize the sending and receiving units

Serial Transmission

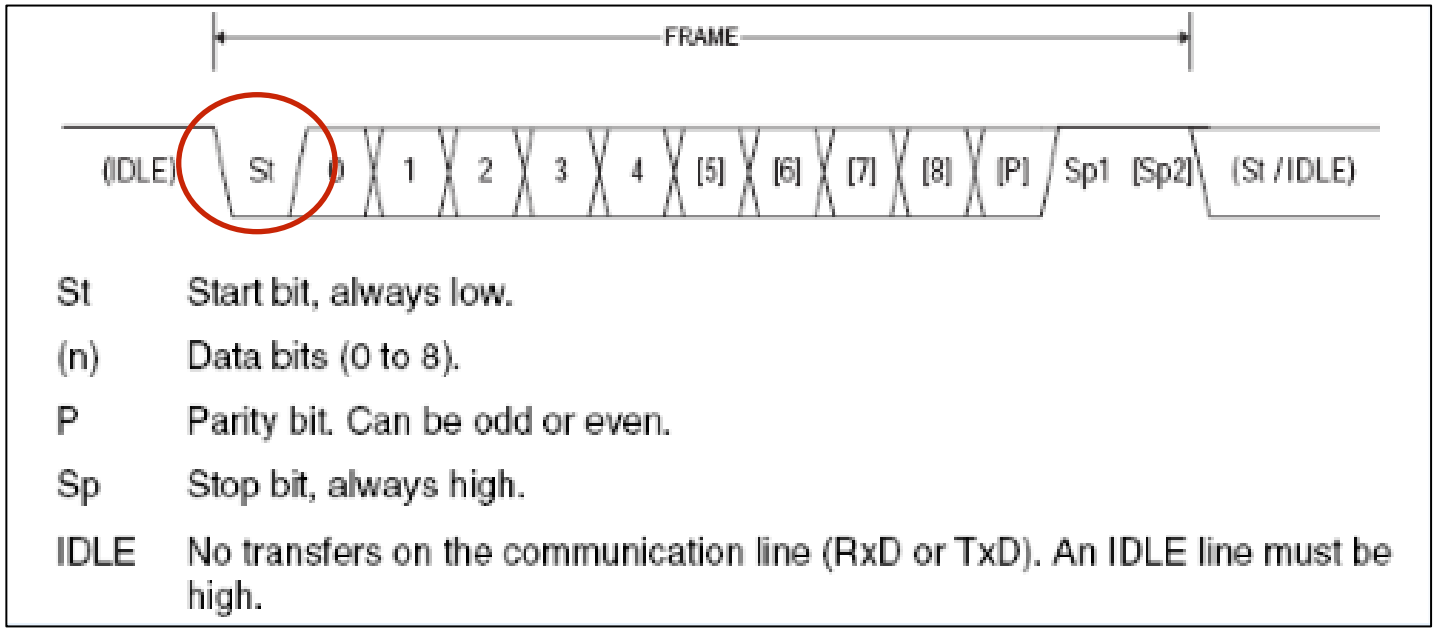
(Asynchronous)



Asynchronous Data Transmission

- When a word is given to the USART for Asynchronous transmissions, a bit called the **"Start Bit"** is added to the beginning of each word that is to be transmitted.
- The *Start Bit* is used to **alert the receiver** that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter.

Serial Frame Format



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 (RxD or TxD). An IDLE line must be high.

Every frame will have at least:

- one start bit
- some data bits (5,6,7,8 or 9)
- one stop bit
- parity bit is optional

Asynchronous Data Transmission

- After the Start Bit, the individual bits of the word of data are sent, with the **Least Significant Bit (LSB) being sent first**.
- Each bit in the transmission is **transmitted for exactly the same amount of time** as all of the other bits, and the receiver “looks” at the wire at approximately halfway through the period assigned to each bit to determine if the bit is a **1** or a **0.**

Asynchronous Data Transmission

- The sender does not know when the receiver has “looked” at the value of the bit; the sender only knows when the clock says to begin transmitting the next bit of the word.
- When the entire data word has been sent, the transmitter may add a ***Parity Bit*** that the transmitter generates.

Internal Clock & Baud Rate

- The internal clock is derived from the internal CPU clock.
- Both transmitter and receiver use the same clock to operate from.
- **The transmitter is synchronous to the clock**; **incoming data to the receiver is not**.
- The baud rates are not exact power of 2 multiples of the CPU clock, thus baud rate inaccuracies occur at some settings.

Baud Rates

TABLE 10-3: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

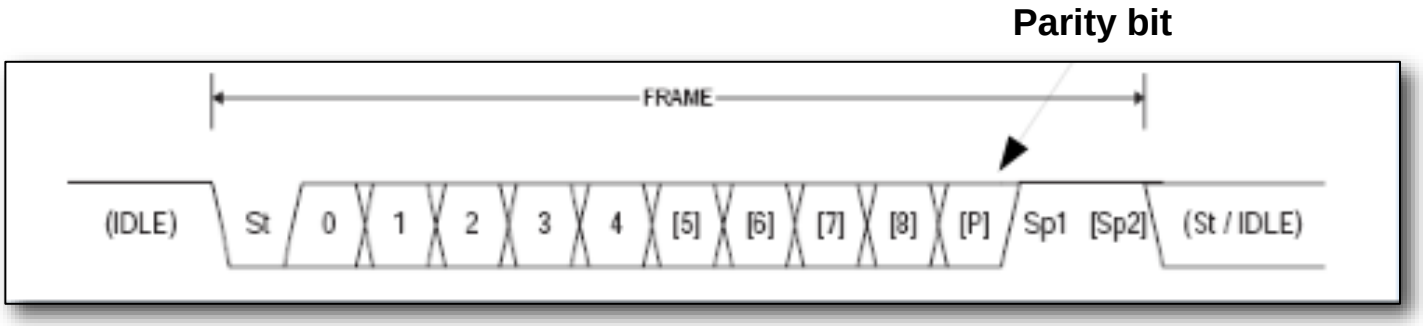
Baud Rate Table for Synchronous and Asynchronous operation (PIC16F877A)

Error Detection

- **Parity**
  - created by an XOR of the data bits
  - parity can be even or odd
  - $P_{even} = d_n$ XOR d_{n-1} XOR d_{n-2} ..... XOR d_0
  - $P_{odd} = d_n$ XOR d_{n-1} XOR d_{n-2} ..... XOR d_0 XOR 1

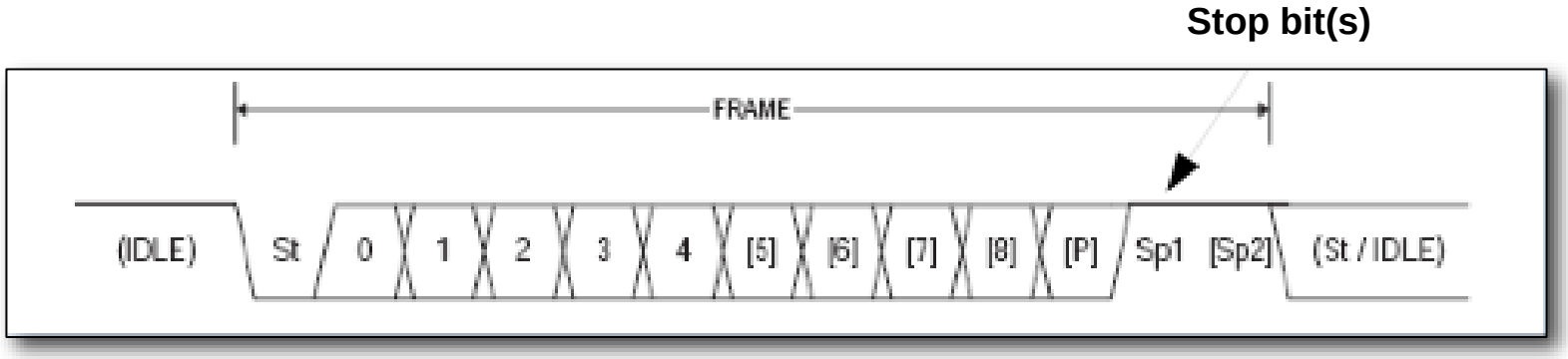
Parity Bit

- If we have a data byte, **00101101** and we want **odd parity**, the parity bit is set to a “one” to make a total of 9 bits which is an odd number of 1's.... i.e., odd parity
- Thus, the new data byte with parity is: **00101101** plus the parity bit **1**



Error Detection

- **Frame Error**
  - occurs when the receiver is expecting the stop bit and does not find it
  - also known as a **synchronization failure** - the frame (byte) must be resent

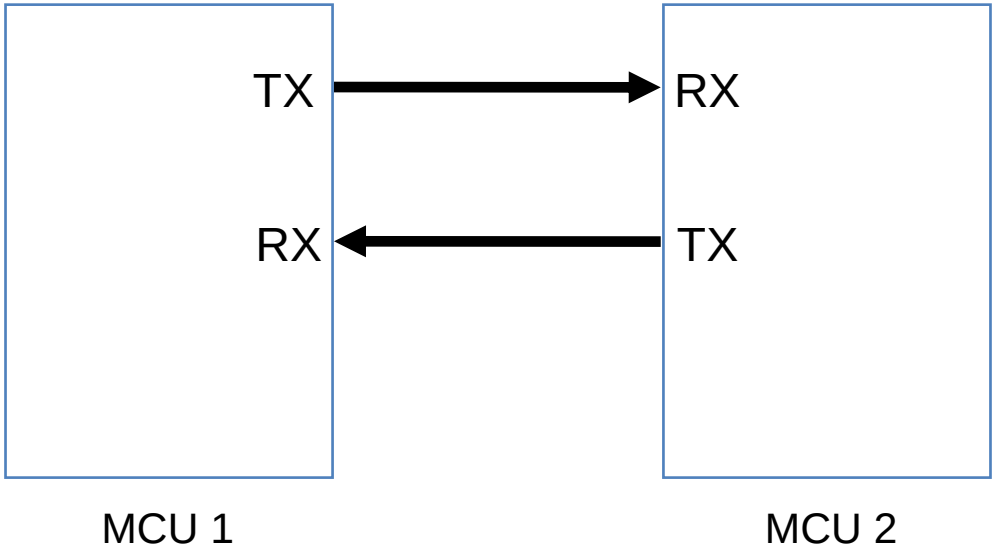


Error Detection

- **Data overrun error**
  - data was not removed in the receive buffer before another byte came and overwrote it

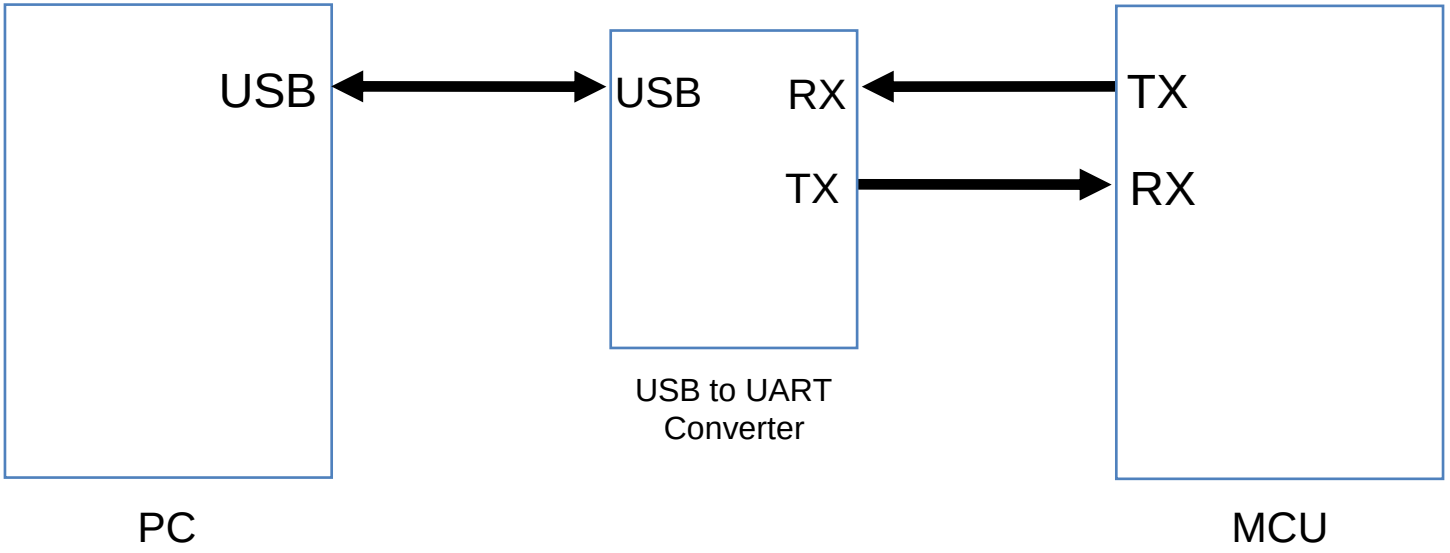
MCU to MCU Communications

- In MCU to MCU, the connections are straightforward as long as the *number of bits*, *number of stop bits*, *parity bit* and *baud rates* are the same in both MCU.



PC to MCU Communications

- Most PC connects to an MCU with a UART module via an emulated UART through USB port.



The USB signals will be converted to UART signals via an IC such as the FT232 USB to UART interface.

USART Module of the PIC16F877A

- The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the two serial I/O modules.
- The USART can be configured as a **full-duplex asynchronous system** that can communicate with peripheral devices.
- The USART can be configured in the following modes:
  - **Asynchronous (full-duplex)**
  - **Synchronous–Master (half-duplex)**
  - **Synchronous–Slave (half-duplex)**

REGISTER 10-1: TXSTA: TRANSMIT STATUS AND CONTROL REGISTER (ADDRESS 98h)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit

Asynchronous mode:
Don't care.

Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit

1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit

1 = Transmit enabled
0 = Transmit disabled

Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit

1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit

Asynchronous mode:
1 = High speed
0 = Low speed

Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit

1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

REGISTER 10-2: RCSTA: RECEIVE STATUS AND CONTROL REGISTER (ADDRESS 18h)

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 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)

USART Baud Rate Generator (BRG)

- The BRG* supports both the Asynchronous and Synchronous modes of the USART.
- It is a dedicated **8-bit baud rate generator**. The **SPBRG register** controls the **period of a free running 8-bit timer**.
- In Asynchronous mode, bit BRGH (TXSTA<2>) also controls the baud rate.
- In Synchronous mode, bit BRGH is ignored.

TABLE 10-1: BAUD RATE FORMULA

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64 (X + 1))	Baud Rate = FOSC/(16 (X + 1))
1	(Synchronous) Baud Rate = FOSC/(4 (X + 1))	N/A

Legend: X = value in SPBRG (0 to 255)

* see page 113 of the PIC16F87X data sheet for more information

TABLE 10-3: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

Baud rates for **low speed** operation.

TABLE 10-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

Baud rates for **high speed** operation.

Baud Rate Calculation

- From Table 10-1 and Tables 10-3 and 10-4, we can calculate the **value of SPBRG register** depending on the desired baud rate and given Fosc.
- For example, if the desired baud rate for asynchronous, high-speed operation is **9.6Kbd** and **Fosc = 4MHz** then the value of SPBRG is calculated as:

$$BaudRate = \frac{F_{osc}}{16(X+1)}$$

From Table 10, where X is the value of SPBRG (0 to 255)

$$9.6\text{Kbd} = \frac{4\text{MHz}}{16(X+1)}$$

$$X = \frac{4\text{MHz}}{16(9.6\text{Kbd})} - 1$$

$$X = 25.041 \text{or } 25\ (0\text{x}19)$$

Registers for Baud Rate

TABLE 10-2: REGISTERS ASSOCIATED WITH BAUD RATE GENERATOR

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used by the BRG.

Asynchronous Mode (Transmit)

- **Steps in setting up asynchronous transmit:**
  1. **Initialize the SPBRG** register for the appropriate baud rate. **If a high-speed baud rate is desired**, set bit **BRGH** (Section 10.1 “USART Baud Rate Generator (BRG)”).
  2. **Enable the asynchronous serial port** by clearing bit **SYNC** and setting bit **SPEN**.
  3. **If interrupts are desired**, then set enable bit **TXIE**.
  4. **If 9-bit transmission is desired**, then set transmit bit **TX9**.
  5. **Enable the transmission** by setting bit **TXEN**, which will also set bit **TXIF**.
  6. **If 9-bit transmission is selected**, the ninth bit should be loaded in bit **TX9D**.
  7. **Load data to the TXREG register** *(starts transmission).*
  8. **If using interrupts**, ensure that **GIE** and **PEIE** (bits 7 and 6) of the **INTCON** register are set.

*** RC6 (TX) and RC7 (RX) are automatically set to output/input when SPEN=1.**

This program will **send a character (8-bit)** using **asynchronous transmit**. It also shows the **configuration of the USART for asynchronous transmission**.

```
void main(void)
{
  SPBRG = 0x19;     // 9.6K baud rate @ FOSC=4MHz high speed
                    // (see formula in Table 10-1)
  SYNC = 0;         // asynchronous mode (TXSTA reg)
  TX9 = 0;          // 8-bit transmission (TXSTA reg)
  TXEN = 1;         // transmit enable (TXSTA reg)
  BRGH = 1;         // asynchronous high speed
  SPEN = 1;         // enable serial port (RCSTA reg)

  for(;;)         // foreground routine
  {
       while(!TRMT);    // wait until TSR buffer is empty
       TXREG = 'A';     // send character ASCII code of 'A'
  }
}
```

Before transmitting, the transmit shift register should be empty.

Registers for Asynchronous Transmit

TABLE 10-5: REGISTERS ASSOCIATED WITH ASYNCHRONOUS TRANSMISSION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous transmission.
Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

Asynchronous Mode (Receive)

- **Steps in setting up asynchronous mode receive:**
  1. **Initialize the SPBRG** register for the appropriate baud rate**. If a high-speed baud rate is desired**, set bit **BRGH** (Section10.1 “USART Baud Rate Generator (BRG)”).
  2. **Enable the asynchronous serial port** by clearing bit **SYNC** and setting bit **SPEN**.
  3. **If interrupts are desired**, then set enable bit **RCIE**.
  4. **If 9-bit reception is desired**, then set bit **RX9**.
  5. **Enable the reception** by setting bit **CREN.**
  6. **Flag bit RCIF will be set when reception is complete** and an **interrupt will be generated if enable bit RCIE is set.**
  7. **Read the RCSTA** register to get the **ninth bit** *(if enabled)* and **determine if any error occurred** during reception.
  8. **Read the 8-bit received data** by reading the **RCREG** register.
  9. **If any error occurred,** clear the error by clearing enable bit **CREN**.
  10. **If using interrupts,** ensure that **GIE** and **PEIE** (bits 7 and 6) of the **INTCON** register are set.

* RC6 (TX) and RC7 (RX) are automatically set to output/input when SPEN=1.

This program will **receive a character (8-bit)** using **asynchronous receive**. It also shows the **configuration of the USART for asynchronous reception**.

```
void main(void)
{
  SPBRG = 0x19;     // 9.6K baud rate @ FOSC=4MHz high speed
                    // (see formula in Table 10-1)
  SYNC = 0;         // asynchronous mode (TXSTA reg)
  BRGH = 1;         // asynchronous high speed
  SPEN = 1;         // enable serial port (RCSTA reg)
  CREN = 1;         // enable continuous receive (RCSTA reg)
  RX9 = 0;          // 8-bit reception (RCSTA reg)
  TRISB = 0x00;     // set all ports in PORTB to output
  PORTB = 0x00;     // initial value of PORTB

   for(;;)         // foreground routine
   {
        while(!RCIF);     // what until the data receive is complete
        PORTB = RCREG;    // read RCREG
  }
}
```

RCIF is set to 1 when a frame is received. This flag is set regardless of RCIE.

Registers for Asynchronous Receive

TABLE 10-6: REGISTERS ASSOCIATED WITH ASYNCHRONOUS RECEPTION

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.

Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

Further Reading

- Read about the steps in **Synchronous Transmit** (page 121, PIC16F8X data sheet).
- Read about the steps in **Synchronous Receive** (page 123, PIC16F8X data sheet).
- Check Canvas for the **seat work on Synchronous Transmit/Receive**.

University of San Carlos | Department of
COMPUTER ENGINEERING

CpE 3201
Embedded Systems

End of Lecture

References:

- Goankar, Ramesh, Fundamentals of Microcontrollers and Applications in Embedded Systems (with the PIC18 MCU Family),2007.
- PIC16F87X Data Sheet, Microchip Technology Inc. 2003.