Chapter 9
PIC USART

1

1

Objective

- Explain serial communication protocol
- Describe data transfer rate and bps rate
- Interface the PIC18 with an RS232 connector
- Describe the main registers used by serial communication of the PIC18
- Program the PIC18 serial port in Assembly

2

2

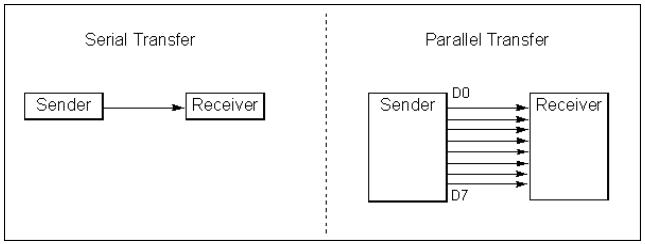
Introduction

- Computers transfer data in two ways: Parallel and Serial.
- Parallel: Eight or more data lines, few feet only, short time
- Serial: Single data line, long distance
- The PIC18 has serial communication capability built into it.

3

Basics of Serial Communication

- The byte of data must be converted to serial bits using a parallel-in-serial-out shift register



Serial versus Parallel Data Transfer

4

Basics of Serial Communication (cont'd)

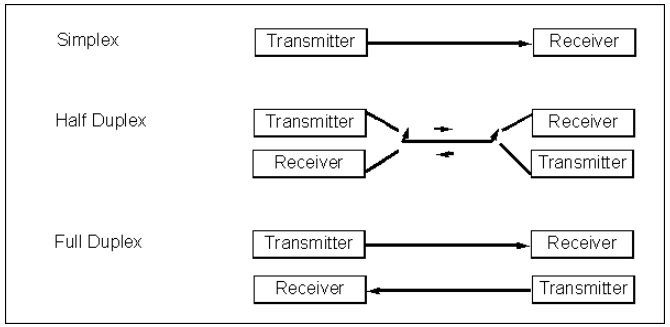
- The receiving end must be a serial-in-parallel-out shift register and pack them into a byte.
- Two methods of serial data communication: Asynchronous and Synchronous

Transfers a single byte at a time

Transfers a block of data at a time

5

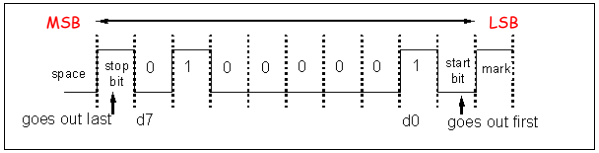
Half-and Full-Duplex Transmission



6

Start and Stop Bits

- In the asynchronous method, each character is placed between start and stop bits (framing)



Framing ASCII 'A' (41H)

7

Asynchronous serial data transmission

An asynchronous serial data link is said to be character oriented (7 or 8 information bits and plus several control bits). When no information is being transmitted, the line is in an idle state. Traditionally, idle state is referred to as the mark level - a logical 1 level.

A start bit: This bit precedes the first data bit indicating the beginning of a character. The start bit is a logical zero, commonly referred to as a space.
Seven to eight data bits: The number of data bits may be either seven or eight (software set by the user). The LSB bit is transmitted first and the MSB, last.
An optional even or odd parity bit: If used, a parity bit is added after the last (MSB) data bit. For even parity, the parity bit is set such that the total number of 1s, including itself, is even. For odd parity, the total number of 1s in data and parity bits is odd.
One or more stop bits: Each character is terminated with one or more stop bits. A stop bit is a logical 1, commonly referred to as a mark.

8

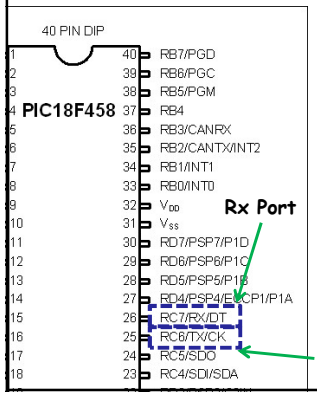
8

Data Transfer Rate

- Rate of data transfer: *bps* (bits per second)
- Another widely used terminology for bps is baud rate
- For Asynchronous serial data communication, the baud rate is generally limited to 100,000bps

9

SPIC18 Serial Port Programming in Assembly



- USART has both
  - Synchronous
  - Asynchronous
- 6 registers
  - SPBRG
  - TXREG
  - RCREG
  - TXSTA
  - RCSTA
  - PIR1

10

SPBRG Register and Baud Rate in the PIC18

- The baud rate can be calculated using the following equations:

BRGH	Baud Rate
0	$f_{osc}/[64(X+1)]$
1	$f_{osc}/[16(X+1)]$

- **X = in 8 bit mode (default)**

11

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

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

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)
1 = Transmit enabled
0 = Transmit disabled

10-12

12

TXSTA (Transmit Status and Control Register) (Cont'd)

bit 4 **SYNC:** EUSART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **SENDB:** Send Break Character bit
Asynchronous mode:
1 = Send Sync Break on next transmission (cleared by hardware upon completion)
0 = Sync Break transmission completed
Synchronous mode:
Don't care.

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 address/data bit or a parity bit.

10-13

Note 1: SREN/CREN overrides TXEN in Sync mode.

13

RCSTA (Receive Status and Control Register)

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

Legend:

R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled (held in Reset)

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.

10-14

14

RCSTA (Receive Status and Control Register) (Cont'd)

bit 4 **CREN:** Continuous Receive Enable bit
Asynchronous mode:
1 = Enables receiver
0 = Disables receiver
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 loads 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
Asynchronous mode 9-bit (RX9 = 0):
Don't care.

bit 2 **FERR:** Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receiving 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
This can be an address/data bit or a parity bit and must be calculated by user firmware.

10-15

15

PIR1 (Peripheral Interrupt Request Register 1)

---	--	RCIF	TXIF	--	---	--	

RCIF Receive interrupt flag bit
1 = The UART has received a byte of data and it is sitting in the RCREG register (receive buffer), waiting to be picked up.
Upon reading the RCREG register, the RCIF is cleared to allow the next byte to be received.
0 = The RCREG is empty.

TXIF Transmit interrupt flag bit
0 = The TXREG register is full.
1 = The TXREG (transmit buffer) register is empty.

The importance of TXIF: To transmit a byte of data, we write it into TXREG. Upon writing a byte into TXREG, the TXIF flag is cleared. When the entire byte is transmitted via the TX pin, the TXIF flag bit is raised to indicate that it is ready for the next byte. So, we must monitor this flag before we write a new byte into TXREG, otherwise, we wipe out the last byte before it is transmitted.

10-16

16

UART TX in more detail

1. Before transmission, three bits must be specified:
   - SPEN = 1 – to enable the UART module
   - TXEN = 1 – to enable the TX module
   - SYNC = 0 – asynchronous TX
2. TXIF (TXREG empty) flag is set when TXEN is set
3. Write to TXREG.
4. Two Events:
   - Content of TXREG is transferred to TSR. TXIF is set again.
   - TSR is filled. TRMT (TSR empty) flag becomes 0.
5. A byte, framed between the start and stop bit, is transmitted out.
6. Once the stop bit has been sent out, TRMT becomes 1.

17

17

Setting up Asynchronous TX (4 Mhz)

```
Main:    movlw   b'00100100'     ; TXEN = 1, SYNC = 0, BRGH = 1
         movwf   TXSTA
         movlw   b'10010000'     ; SPEN = 1
         movwf   RCSTA
         movlw   D'25'           ; Set Baud rate for 9600
         movwf   SPBRG

         movlw   'A'
         call    putChar
         bra     $

putChar: btfss   TXSTA, TRMT     ; wait until the last TX finishes
         bra     putChar         ; TRMT = 1 if TX finishes
         movwf   TXREG           ; Put 'A' into TXREG
         return
```

18

18

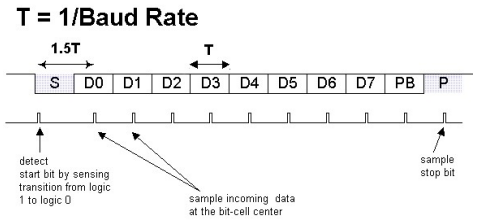
Setting up Asynchronous TX (10 Mhz)

```
Main:    movlw   b'00100100'     ; TXEN = 1, SYNC = 0, BRGH = 1
         movwf   TXSTA
         movlw   b'10010000'     ; SPEN = 1
         movwf   RCSTA
         movlw   D'64'           ; Set Baud rate for 9600
         movwf   SPBRG

         movlw   'A'
         call    putChar
         bra     $

putChar: btfss   TXSTA, TRMT     ; wait until the last TX finishes
         bra     putChar         ; TRMT = 1 if TX finishes
         movwf   TXREG           ; Put 'A' into TXREG
         return
```

19

19

PIC18 UART Reception



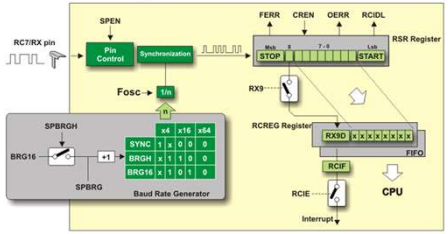
- Receiver must know the transmission baud rate in order to sample correctly.
- If the stop bit of 1 is not detected, *framing error* occurs.

20

20

PIC18 UART Reception

- USART peripheral needs to be enabled (RCSTA.SPEN)
- Receiving on UART needs to be enabled (RCSTA.CREN)
- The PIR1.RCIF flag will be raised by the microcontroller after it received a valid byte.
- The user needs to copy the received byte out of RCREG. This will automatically reset RCIF.



21

21

PIC18 UART Reception

This example receives bytes and output to PORTD (4 Mhz)

```
Main:     clrf    TRISD           ; set PORTD as output

          movlw   b'00100100'     ; SPEN = 1, SYNC = 0, BRGH = 1
          movwf   TXSTA
          movlw   b'10010000'     ; SPEN = 1, CREN = 1
          movwf   RCSTA
          movlw   25              ; Set Baud rate for 9600
          movwf   SPBRG

MainLoop: btfss   PIR1, RCIF      ; wait until receiving a complete byte
          bra     MainLoop
          movff   RCREG, PORTD    ; move the received byte to PORTD
          bra     MainLoop
```

22

22

PIC18 UART Reception

In practice, we will not use polling to wait an input. We use the interrupt concept.

In software,

- Configure some registers to enable the UART RX interrupt in the beginning part of the program.
- Write an interrupt service routine to process the input byte.

So, in the normal condition, the system serves some other tasks. When an input byte is arrived, the hardware will generate an interrupt (for UART RX). The CPU will finish the current instruction. After identify the source of interrupts (using polling to identify, check IF bits of each possible source), the CPU can jump the UART RX service routine.

23

23