

3.20 The Universal Asynchronous Receive Transmit (UART) device

EE302 – Real time and embedded systems

1

Learning outcomes

2

- **Describe** (with particular attention to timing) the operation of a UART
- **Show** how asynchronous serial communications can be implemented in software using UART based communication

10 November 2020

2

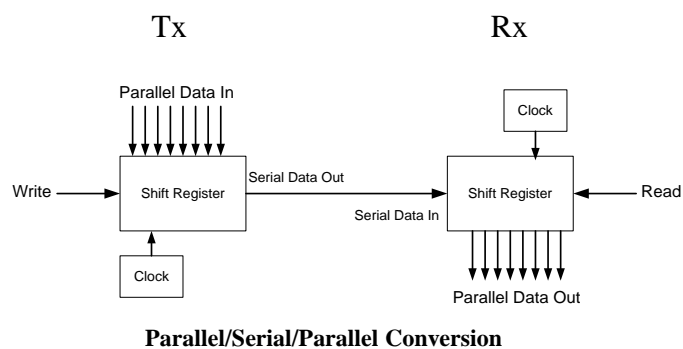
- UART (Universal Asynchronous Receive and Transmit)
 - A dedicated helper device for asynchronous serial communication
 - Tx: Converts Parallel Data to a serial bit stream
 - Rx: Converts serial bit stream to parallel data.
 - Controls handshaking signals to and from other UARTs
 - Handles bit timing of transmission and reception of data.
 - Provides interfacing signals to a CPU.

10 November 2020

3

UART data communication

- Tx: Parallel to serial
- Rx: serial to parallel
- A full duplex UART has both Tx and Rx parts

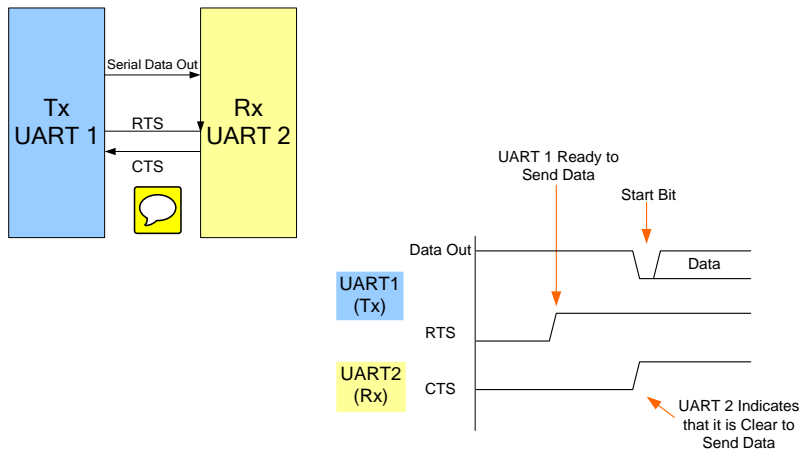


10 November 2020

4

UART flow control handshaking

5



10 November 2020

5

Example: Configure and use PIC16F877 UART

6

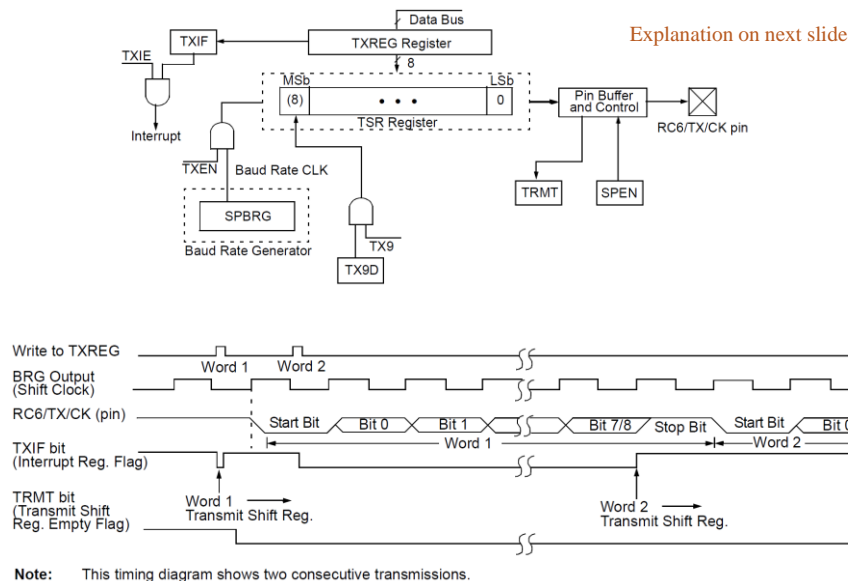
- PIC UART is addressable & register configured
 - So just configure internal registers in the UART
- The UART on the PIC16F877 supports the following:
 - 1 Start Bit
 - 8 (or 9) Data Bits
 - 1 Stop Bit
 - Software parity (uses data bit 9 – we need to calculate parity ourselves)
 - various baud rates by dividing F_{osc}
 - Software polled or interrupt driven I/O synchronization

10 November 2020

6

PIC UART – Tx

7



10 November 2020

7

PIC UART – Tx figures explained

8

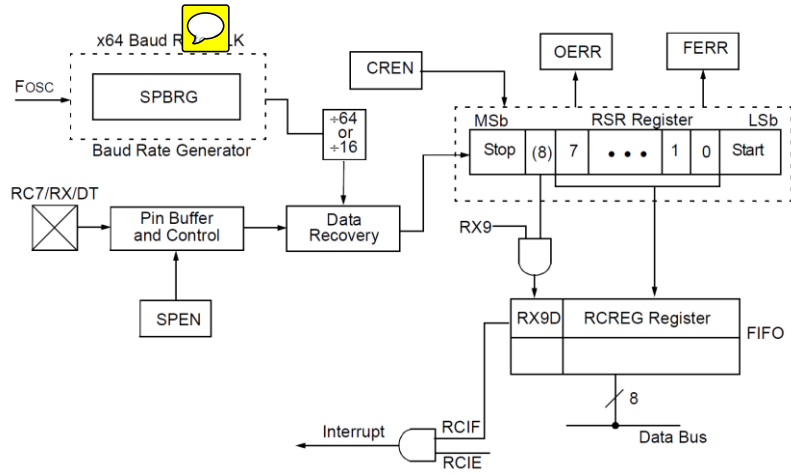
- TXREG is where your program writes data to be transmitted
- TSR (transmit shift register) is where the data is copied for shifting onto the wire. It means that one word can be “in progress” in the TSR when you write the next word to TXREG. This is shown in timing diagram as word 2 copied to TXREG just after word 1.
- TXIF (transmit interrupt flag) is set (1) only when TXREG is empty (not when a word has been fully transmitted) and it can only be cleared (to 0) by loading TXREG with a value.
- TRMT bit is set (1) only when TSR is empty (i.e. data has been fully transmitted). If TSR is currently transmitting a word, TRMT is clear (0).
- TX9D is 9th data bit to allow software parity implementation. It is optional.
- SPEN enables the UART transmitter – if clear, nothing can be sent.

10 November 2020

8

PIC UART - Rx

9

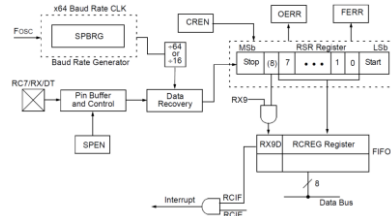


10 November 2020

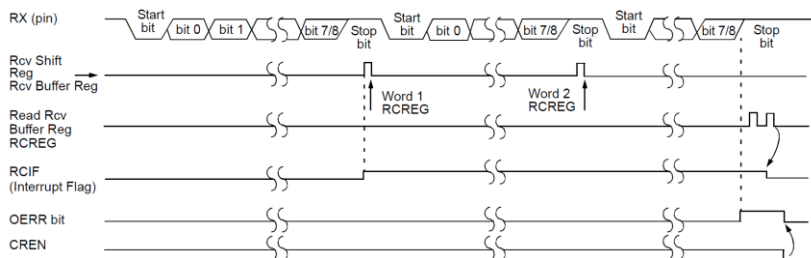
9

Contd.

10



Explanation on next slide



Note: This timing diagram shows three words appearing on the RX input. The RCREG (Receive Buffer) is read after the third word, causing the OERR (Overrun Error) bit to be set.

10 November 2020

10

PIC UART – Rx figures explained 1

11


- RSR (receive shift register): serial bits are shifted into RSR as they are received on the wire. When RSR has a full word, it is copied into the RCREG FIFO.
- RCREG is really 2 registers worth of memory so it can store 2 received words. This means that data won't be lost immediately even if your code does not read RCREG as soon as first word copied in. Because RCREG is FIFO, the first word received will also be the first word that your code gets if it read RCREG. If a third word is received before any word have been removed from RCREG, then overrun results
- RCIF (receive interrupt flag) is set when data is copied from RSR to RCREG. It can only be cleared by emptying RCREG (i.e. reading one or two words, depending on how many the FIFO contains)

10 November 2020

11

PIC UART – Rx figures explained 2

12

- OERR (overrun error) – this is set if the stop bit of an async word is received and RCREG FIFO is full, i.e. there is an attempt to copy in a 3rd word
- FERR (framing error) – this is set if the stop bit is detected as SPACE instead of MARK
- RX9D – can be used for software parity implementation
- RCSTA (register not shown above) contains the FERR and RX9D bits and must be read before RCREG – reading RCREG first can change the values. 
- SPEN (serial port enable) controls whether UART pins are enabled.
- CREN (continuous receive enable) determines whether the RSR allows words to follow directly after one another without signalling overflow – usually it should be enabled.

10 November 2020

12

UART pseudocode 1 - wait before transmit (1 byte)

13

```
Main...

setup()
    configure baud rate, serial port, transmission

// Very simple (and not very useful): simply send a dot every
// second to indicate system is working
loop()
    transmitByte(random(256))

// returns as soon as val is buffered (in TXREG) but
// before all its bits are completely transmitted
transmitByte(val)
    wait until TXIF is set // i.e. until TXREG is available
    set TXREG = val
    // return before val in TXREG has necessarily been transmitted
```

What are the timing implications of using this approach?

10 November 2020

13

UART pseudocode 2 – transmit completely (1 byte)

14

```
Main...

setup()
    configure baud rate, serial port, transmission

// Very simple (and useless): just send a dot every
// second to indicate system is working
loop()
    transmitByteCompletely(random(256))

// wait until value completely transmitted before return
transmitByteCompletely(val)
    set TXREG = val
    wait until TRMT is clear // fully transmitted
```

What are the timing implications of using this approach?
Better or worse than previous slide?

10 November 2020

14

UART pseudocode 3 – transmit a string (busy wait polling)

15

```
Main...  
  
setup()  
...  
  
loop()  
  transmitString("hello")  
  otherWork()
```

```
// wait until str completely  
// transmitted before returning  
transmitString(str)  
  while not at end of str  
    // see prev. transmitByte  
    wait until TXIF is set  
    set TXREG = next character in str
```

What are the timing implications of using this approach?

10 November 2020

15

UART pseudocode 4– transmit a string, multitask polling

16

```
global gTxBuffer
```

```
Main...
```

```
setup()...
```

```
// as before
```

```
loop()  
  tryTransmitString("hello")  
  pollTx()  
  otherWork()
```

```
tryTransmitString(str)  
  if gTxBuffer empty/sent  
    startTx(str)  
    return TRUE  
  else  
    return FALSE
```

```
// prepare to transmit  
startTx(str)  
  copy str to gTxBuffer
```

```
// transmit one character  
pollTx()
```

```
  if not at end of gTxBuffer  
    // is TXREG ready for more data?  
    if TXIF is set  
      set TXREG = next char from gTxBuffer
```



What are the timing implications of using this approach?

Does tryTransmitString or pollTx wait/delay (and hence block otherWork)?

10 November 2020

16