TB3156

Universal Asynchronous Receiver Transmitter (UART) on 8-Bit PIC® Microcontrollers

Author: Mary Tamar Tan and Michael Wyman
Microchip Technology Inc.

INTRODUCTION

UART is a communications hardware used to transfer data serially between devices. Three variations of UART modules are available on 8-bit PIC® microcontrollers:

- Universal Synchronous Asynchronous Receiver Transmitter (USART)
- Enhanced Universal Synchronous Asynchronous Receiver Transmitter (EUSART)
- Universal Asynchronous Receiver Transmitter (UART) with protocol support

The USART is mostly found in legacy PIC MCUs. In addition to the basic capabilities of hardware UART such as asynchronous transmit and receive, the USART can also operate in Synchronous Master and Slave modes. The EUSART features “enhanced” capabilities compared to the USART such as Break character support and auto-baud detection for LIN. It is usually found in enhanced mid-range and high-end devices. The UART with protocol support is the latest UART module featuring hardware support for standard protocols such as the DMX512, Digital Addressable Lighting Interface (DALI), and Local Interconnect Network (LIN). The PIC18FXXK42 family is one example of devices featuring this type of UART. A single 8-bit PIC MCU device may contain one or more UART modules.

This technical brief aims to provide an overview of the different capabilities of the UART module. Sample configurations are also shown to exhibit the robustness and flexibility of the module for implementation on a wide range of possible applications.

Note: This document uses “UART module” as a generic term to refer to the three modules (USART, EUSART and UART with protocol support), unless stated otherwise.

FEATURES AND PROTOCOL SUPPORT

To maintain robust serial communication between devices and to minimize software intervention during data transfer, the hardware fully supports interrupt-driven transmit and receive, FIFO buffering, and error detection. Several devices also feature advanced capabilities such as parity, checksum calculation, and hardware support for standard protocols including LIN, DMX and DALI.

Table 1 shows a side-by-side comparison between the three existing types of UART module in 8-bit PIC microcontrollers.

TABLE 1: SUPPORTED FEATURES AND PROTOCOLS

Hardware Feature	USART	EUSART	UART with Protocol Support
Full-duplex asynchronous transmit and receive	✓	✓	✓
Half-duplex synchronous master	✓	✓	—
Half-duplex synchronous slave	✓	✓	—
Two-character input buffer	✓	✓	✓
One-character output buffer	✓	✓	✓
Programmable character length	8-bit, 9-bit	8-bit, 9-bit	7-bit, 8-bit, 9-bit
9^{th} bit address detection	—	✓	✓
9^{th} bit even or odd parity	—	—	✓
Input buffer overrun error detection	✓	✓	✓
Received character framing error detection	✓	✓	✓
Hardware and software flow control	—	—	✓
Automatic checksums	—	—	✓
Programmable Stop bits	—	—	1, 1.5, 2
Programmable data polarity	—	✓	✓
Manchester encoder/decoder	—	—	✓
Automatic detection and calibration of the baud rate	—	✓	✓
Wake-up on Break reception	—	✓	✓
Automatic Break period generation	—	Fixed 13-bit character	Automatic LIN and DALI Break period and User-timed
Protocol Support			
LIN Master and Slave	—	—	✓
DMX mode	—	—	✓
DALI Control Gear and Control Device	—	—	✓

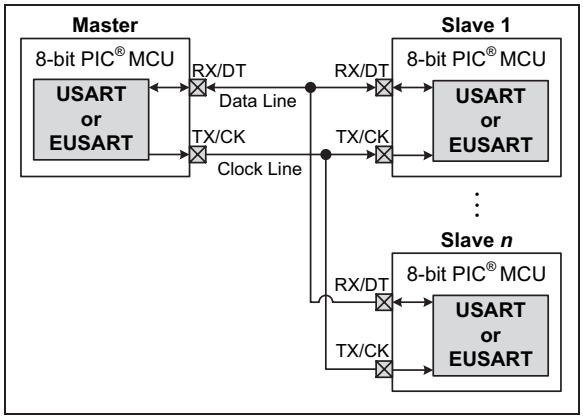
SYNCHRONOUS MODE

The ‘S’ in both the USART and EUSART modules stands for synchronous. Synchronous means that the data transfer between devices is being synchronized by a clock signal. This type of communication eliminates the use of signaling methods such as the Start and Stop bits implemented in asynchronous systems.

Synchronous communications are typically used in systems with a single master and one or more slaves. The master device contains the necessary circuitry for baud rate generation and supplies the clock for all devices in the system. Slave devices can take advantage of the master clock by eliminating the internal clock generation circuitry.

In Synchronous mode, two microcontroller pins are utilized: the RX/DT pin and the TX/CK pin. The RX/DT is a bidirectional data line while the TX/CK is the clock line. Slaves use the external clock supplied by the master to shift the serial data into and out of their respective Receive and Transmit Shift registers. Since the data line is bidirectional, synchronous operation is half-duplex only. Half-duplex refers to the fact that master and slave devices can receive and transmit data but not both simultaneously. The USART and EUSART can operate as either a master or slave device. A typical configuration is shown in Figure 1.

FIGURE 1: SYNCHRONOUS CONFIGURATION

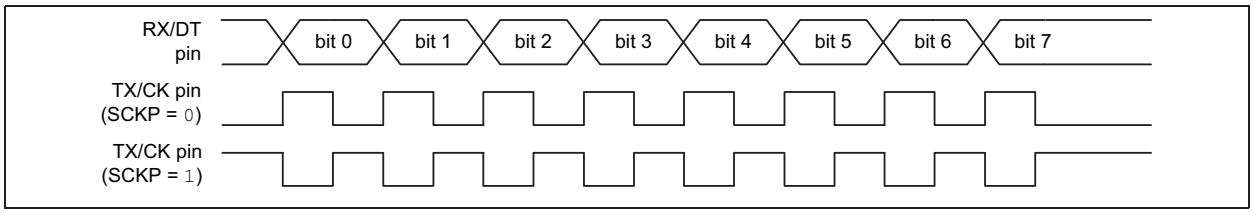


Synchronous Master Mode

In Synchronous Master mode, the master transmits the clock on the TX/CK line. Selectable clock polarity is made available for compatibility with Microwire. Refer to Figure 2.

During transmit, the RX/DT and TX/CK pin output drivers are automatically enabled when the module is configured for synchronous master operation. Data bits are shifted out through the RX/DT pin. Each data bit changes on the leading edge of the master clock and remains valid until the subsequent leading clock edge.

FIGURE 2: DATA AND CLOCK SYNCHRONIZATION



During synchronous master reception, the RX/DT pin output driver is automatically disabled. The number of generated clock cycles can be configured as either continuous or only equal to the number of data bits in a single character.

Synchronous Slave Mode

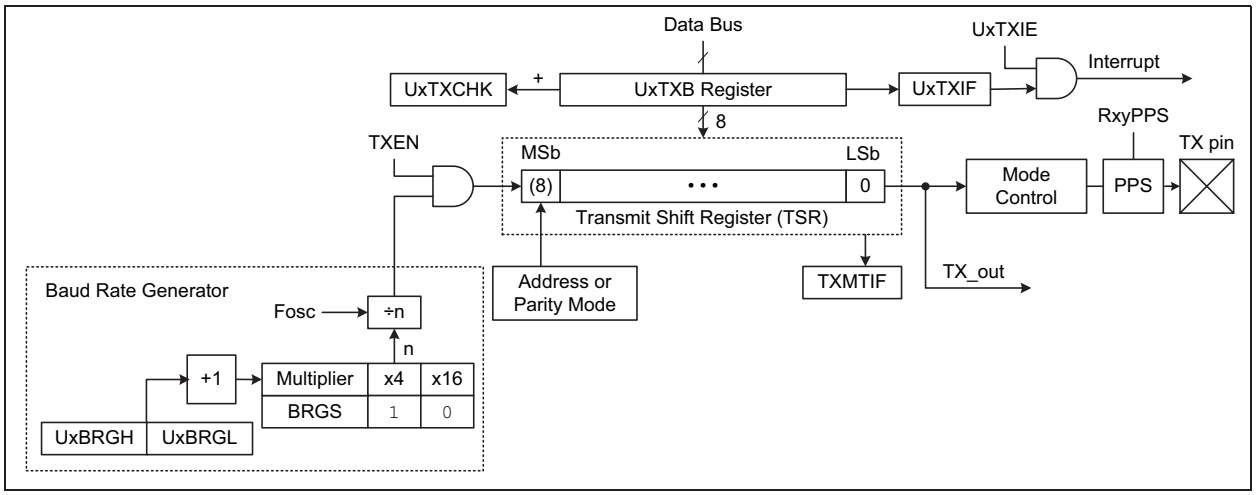
The slave receives the clock on the TX/CK line (see Figure 1). The TX/CK pin output driver is automatically disabled when the device is configured for synchronous slave transmit or receive operation. Serial data bits change on the leading edge to ensure they are valid at the trailing edge of each clock. One data bit is transferred for each clock cycle. Only as many clock cycles should be received as there are data bits.

ASYNCHRONOUS TRANSMISSION

The UART module block diagram of a PIC18FXXK42 device is shown in Figure 3. PIC18FXXK42 is the first family of 8-bit PIC microcontrollers featuring the UART with protocol support.

Note:	Transmit block diagram, registers and bit names may vary per device. Refer to the data sheet for device-specific information.

FIGURE 3: UART TRANSMIT BLOCK DIAGRAM



A transmission is initiated by writing a character to the transmit buffer (UxTXB). If this is the first character, or the previous character has been completely flushed from the TSR, the data in the UxTXB is immediately transferred to the TSR. If the TSR still contains all or part of a previous character, the new character data is held in the UxTXB until the previous character transmission is complete. The pending character in the UxTXB is then transferred to the TSR at the beginning of the previous character Stop bit transmission. The transmission of the Start bit, data bits and Stop bit sequence commences immediately following the completion of all of the previous character's Stop bits.

Setting the TXEN bit in the UxCON0 register enables the transmitter. The TSR shifts data bits out in a rate defined by the user in the UxBRGH:UxBRGL registers and BRGS bit of the UxCON0 register. The UART mode, including the 7-bit, 8-bit, 8-bit with address or parity, and any of the protocol-support modes, is selected through the MODE<3:0> bits of the UxCON0 register.

The UART transmit output (TX_out) is available through the TX pin which is selected via the Peripheral Pin Select (PPS) module's RxyPPs register. It can also be configured in software as an input to other on-chip peripherals such as the Configurable Logic Cell (CLC) through the corresponding peripheral's register. Internal connection to other peripherals reduces external wiring and also frees up a couple of pins (TX pin and peripheral input pin) for other uses.

Two flag bits can be used to determine the status of the transmitter: UxTXIF and UxTXMTIF. The UxTXIF Transmit Interrupt Flag bit is set whenever the UxTxB is empty, regardless of the state of the UxTXIE transmit interrupt enable bit. It is only clear when the TSR is busy with a character and a new character has been queued for transmission in the UxTXB. On the other hand, the UxTXMTIF bit is set when the TSR is empty and idle, and cleared when a character is transferred to the TSR from the UxTXB. These two bits are implemented depending on how data will be written to the module, which can either be interrupt-based or polling-based.

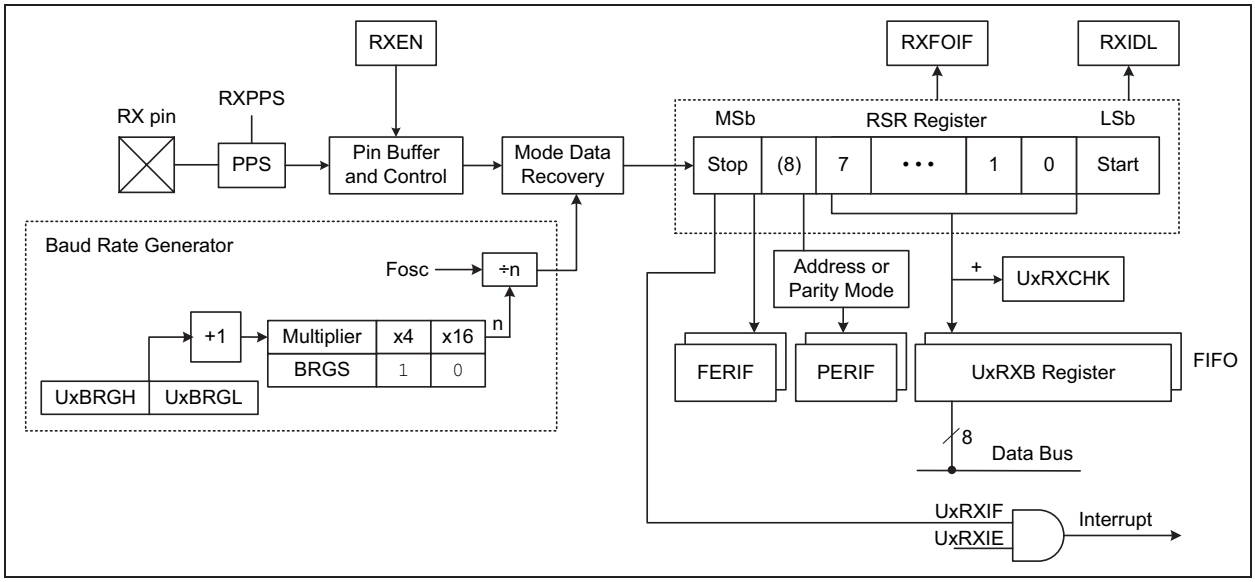
ASYNCHRONOUS RECEPTION

The UART receiver block diagram of the PIC18FXXK42 is shown in Figure 4.

Note:	Receive block diagram, registers and bit names may vary per device. Refer to the data sheet for device-specific information.

The RX pin is selected through the RXPPS register of the PPS module. Setting the RXEN bit in the UxCON0 register enables the receiver circuitry of the UART. The data is received on the RX pin and drives the data recovery block. The data recovery block is a high-speed shifter operating at four or 16 times the baud rate depending upon the configuration of the BRGS bit of the UxCON0 register. The serial Receive Shift register (RSR) shifts in the character bits at the configured baud rate. When all the bits of the character have been shifted in, they are immediately transferred to a two character First-in-First-Out (FIFO) memory. The UxRXIF interrupt flag in the PIR1 register is set at this time, provided it is not being suppressed. FIFO buffering allows reception of two complete characters before software must start servicing the UART receiver. The FIFO registers and RSR are not directly accessible by software. Access to the received data is through the receive buffer (UxRXB) register.

FIGURE 4: UART RECEIVE BLOCK DIAGRAM



The Framing Error Interrupt Flag bit (FERIF) of the UxERRIR register represents the frame status of the top unread character of the receive FIFO. It is cleared when the character at the top of the FIFO does not have a framing error or when all bytes in the receive FIFO have been read.

When operating in any parity mode, the Parity Error Interrupt Flag bit, PERIF, in the UxERRIR register represents the parity error of the top unread character of the receive FIFO rather that the parity bit itself. It can also represent as address or data indicator when Addressing mode is enabled or forward or backward frame indicator for DALI mode.

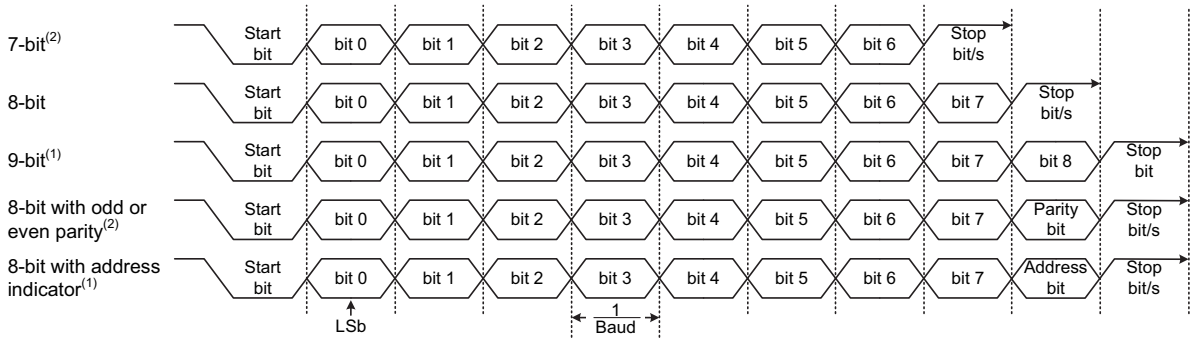
The RXFOIF Receive FIFO Overflow Interrupt Flag bit is set when more characters are received than what the receive FIFO can hold. The character causing the overflow condition is always discarded. The RXIDL Receive Pin Idle Status bit is set when the RX pin is in Idle state and cleared when the UART is receiving Start, Stop, Data, Auto-baud, or Break characters.

ASYNCHRONOUS MODES

The UART transmits and receives data using the standard Non-Return-to-Zero (NRZ) format, having two levels: a VOH Mark state ('1' data bit) and a VOL Space state ('0' data bit). In NRZ, consecutively transmitted data bits of the same value stay at the output level of that bit without returning to neutral level between each bit transmission and the port idles in the Mark state. The Start bit is always a space and stop bits are always marks. Each bit persists for a period of 1/(Baud Rate) with the baud rate derived from an on-chip dedicated 16-bit Baud Rate Generator (see **Section "Baud Rate Generator"**).

The UART can operate in different Asynchronous modes as depicted in Figure 5.

FIGURE 5: UART ASYNCHRONOUS MODES



Note 1: EUSART and UART with protocol support only.

2: UART with protocol support only.

7-Bit Mode

In 7-Bit mode, only the seven LSbs of the data are transmitted between the Start and Stop bits. This mode is usually implemented when the data being exchanged are 7-bit ASCII characters.

8-Bit Mode

This mode is composed of one Start bit, eight data bits with no parity, and one or more Stop bits. The most popular data format used in PC serial communications is the 8N1 configuration having eight data bits, no parity, and one Stop bit.

9-Bit Mode

The EUSART module supports 9-bit asynchronous communication with the ninth data bit's function (i.e., parity or address indicator) to be determined in software. The UART with protocol support, on the other hand, allows the use of the ninth bit either for parity checking or address detection in hardware.

8-bit Data with Even or Odd Parity

Parity is fully supported by the UART with protocol support module. Parity checking is a simple error detection method that uses an additional bit to indicate the number of '1' data bits and exists in two variants: even parity and odd parity.

In even parity, when the count of '1' data bits is odd, the parity bit is set to '1', otherwise, it is set to '0'. In odd parity, when the count of '1' data bits is even, the parity bit is set to '1', otherwise, it is set to '0'.

When either even or odd parity is selected, the hardware automatically determines and inserts the parity bit in the serial data stream during transmission. The UART receiver is also capable of performing the parity check. An error flag is set whenever a parity error is detected during reception.

8-Bit Data with Address Indicator

This mode implements an address bit as the ninth data bit. See **Section "Asynchronous Address Mode"** for details.

ASYNCHRONOUS ADDRESS MODE

Multi-drop applications wherein multiple receivers share the same transmission line, such as in RS-485 systems, require addressing to avoid contention on the communication bus. Two types of address modes are available depending upon the selected device's UART module.

EUSART Address Detection Mode

In EUSART, address detection is enabled by setting the ADDEN bit. When this bit is enabled, only characters with the ninth data bit set will be transferred to the receive FIFO buffer and all other characters will be ignored. Upon receiving an address character, user software determines if the address matches its own.

UART with Protocol Support Address Mode

The UART with protocol support supports Asynchronous Address mode wherein data is transmitted and received as 9-bit characters, as shown in Figure 6. The ninth bit determines whether the character is an address or data. When the ninth bit is set, the eight LSbs are the address. When the ninth bit is clear, the eight LSbs are data.

FIGURE 6: UART ADDRESS MODE

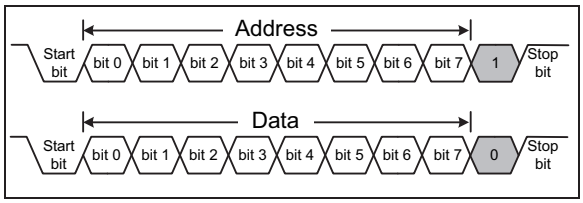
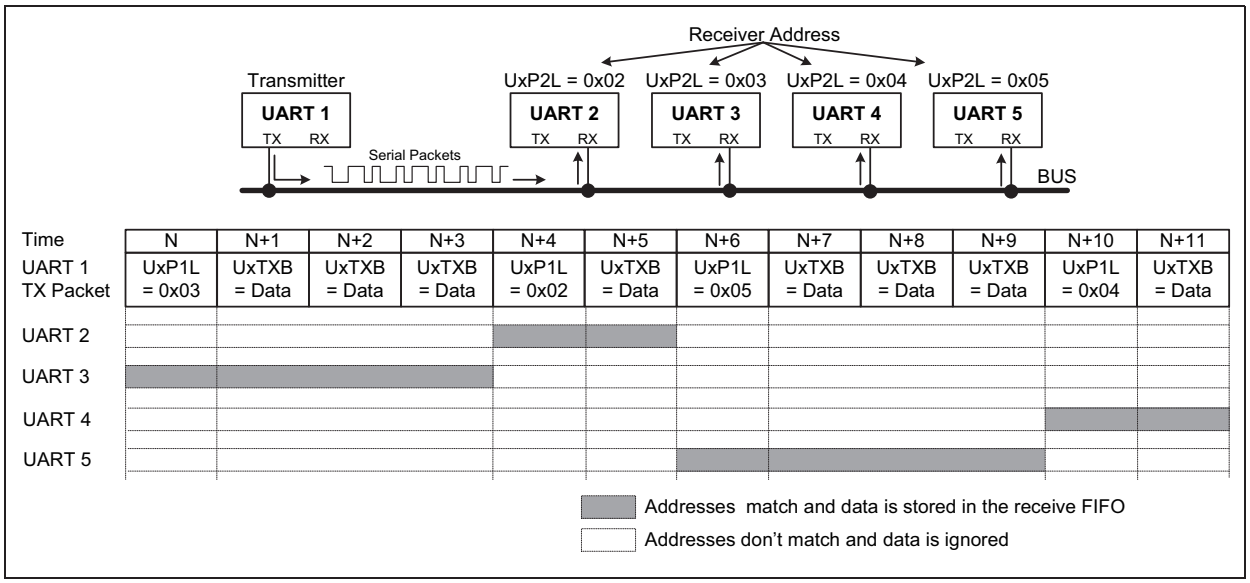


Figure 7 shows an example of a single-transmitter (master) multiple-receiver (slaves) system using a PIC18F25K42. The master initiates transmission of all addresses and data packets.

When a PIC18FXXK42 UART is used as master, character write is very simple: addresses are written to the UxP1L register while data are written to the UxTXB Transmit Buffer register. The hardware automatically appends the one or zero on the ninth bit during transmission.

FIGURE 7: UART ADDRESS MODE IMPLEMENTATION



The slaves perform address matching by comparing the receiver address (UxP2L) to the eight LSBs of every character received with a ninth bit set and verifying the result through a mask register. When a match occurs, all succeeding data are stored in the receive FIFO and are available for reading via the Receiver Buffer register. Otherwise, all succeeding data will be ignored and the slave will wait for another address in the bus.

Addressing mode can be used in any multiple-transmitter and multiple-receiver system. The user has to make sure that no two transmitters are active at the same time to prevent contention in the bus. This can be achieved through proper scheduling of tasks in software.

DMX MODE

The DMX mode is supported by the UART with protocol support only. DMX, or DMX512 is an asynchronous serial digital data transmission standard for controlling lighting equipment and accessories. It consists of a control console that sends out commands that are being received by other equipment connected on the same data link such as theater light dimmers, fog machines, and strobe lights.

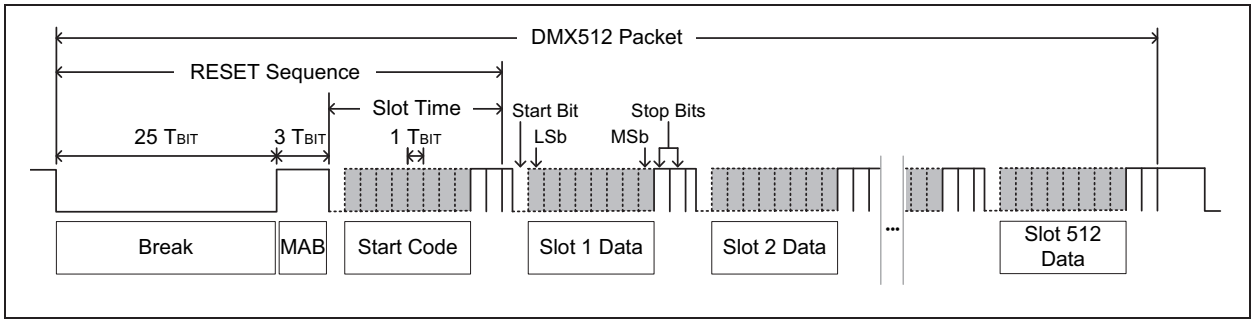
The UART with protocol support hardware supports the following DMX features:

- Automatic generation of Break and Make After Break sequence
- Break and Make After Break sequence detection
- Configurable internal data byte counter
- Dedicated registers for receiver first and last byte detection

DMX512 typically operates at a bit rate of 250 kbit/s. The bit rate is calculated using the formula given in the device data sheet. Sample bit rate formulas for PIC18FXXK42 are given in **Section "Baud Rate Generator"**.

A DMX512 packet is composed of a Reset sequence followed by a sequence of up to 512 data bytes as shown in Figure 8.

FIGURE 8: DMX512 PACKET



DMX Console Transmitter

The UART is capable of generating the Reset sequence which is made up of a Break, a “Make After Break” (MAB), and a Start Code. When using the module, the width of a Break is fixed at 25 bit times. This is followed by a MAB which is an idle period with a value of approximately three bit times. The Start Code specifies the function of the subsequent data bytes on the packet.

The Start Code and data bytes are written to the Transmit Buffer register. Software needs to keep track of the number of bytes written to the transmit buffer to ensure that no more and no less than the desired number of bytes are sent because the DMX state machine will automatically insert a Break and reset its internal counter after ‘n’ bytes are written.

Each data slot begins with a Start bit and is terminated by two Stop bits.

DMX Equipment Receiver

The Break character threshold for DMX equipment is 23 bit times. Breaks shorter than this period will be ignored and the DMX state machine remains in Idle mode. Upon receiving the Break, the DMX counters will be reset to align with the incoming data stream. The UART will see the “Make After Break” and store the Start Code in the receive FIFO.

After the Start Code, the first through 512th byte will be received, but only the ones of interest will be stored in the receive FIFO. The UART identifies when to start and stop storing data bytes through a couple of dedicated 16-bit registers.

DMX Mode Implementation

Using a PIC16F25K42, the UART is set to DMX mode by setting MODE<3:0> = `1010` and STP<1:0> = `10`.

FIGURE 9: DMX MODE IMPLEMENTATION

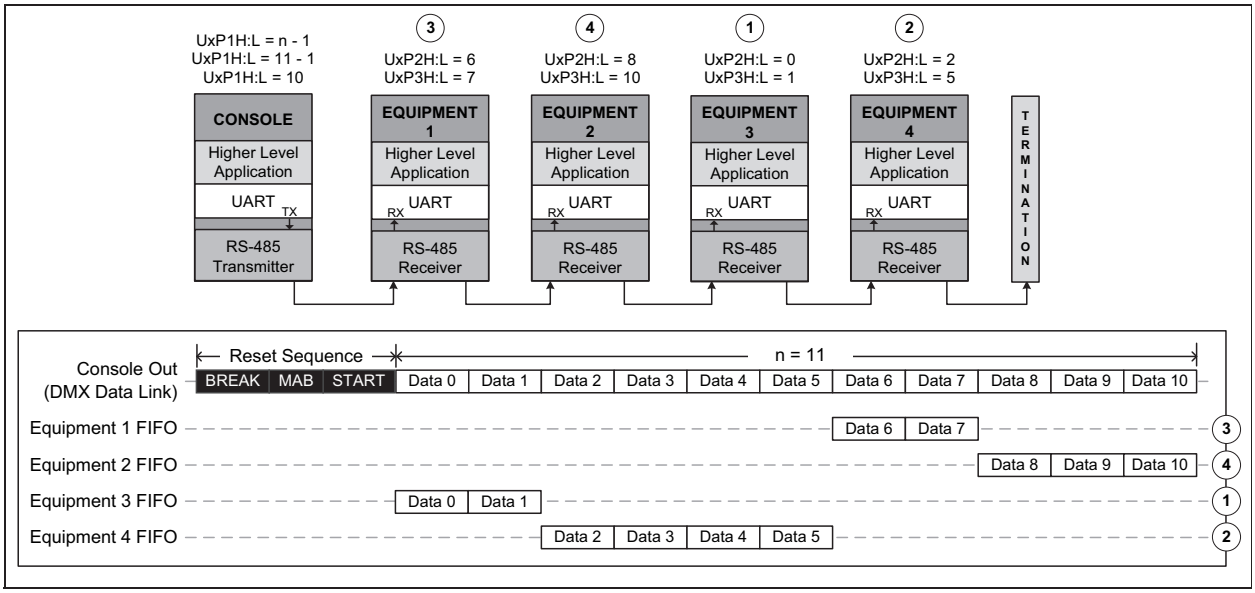


Figure 9 shows a sample DMX512 universe. One controller console is connected to four unit loads or equipment in a daisy-chained configuration. The number of data bytes to be transmitted by the console is determined by the UxP1H:L registers. The equipment then identifies when they would start and stop storing the received data in the FIFO through the UxP2H:L and UxP3H:L registers, respectively. The data packet is retransmitted continuously to update the data to the equipment as it changes.

Initialization code for the DMX console and DMX Equipment 2 in Figure 9 are shown in Example 1 and Example 2, respectively.

EXAMPLE 1: DMX CONSOLE INITIALIZATION CODE

```
// Initialize DMX Console to transmit 11 data bytes

void DMX_Console_Initialize (void){
    U1CON0bits.MODE = 0b1010;       // DMX mode
    U1CON0bits.TXEN = 1;            // Enable transmit
    U1CON0bits.RXEN = 0;            // Disable receive
    U1CON2bits.TXPOL = 0;           // Data polarity is not inverted
    U1P1H = 0x00;                   // n-1 = 11-1
    U1P1L = 0x0A;
    U1BRGH = 0x00;                  // Initialize baud rate to 250k; Fosc = 64 MHz
    U1BRGL = 0x0F;
    U1CON2bits.STP = 0b10;          // 2 Stop bits

    Pin_Initialize();               // Set the corresponding TX pin via PPS

    U1CON1bits.ON = 1;              // Enable serial port
}
```

EXAMPLE 2: INITIALIZATION CODE FOR DMX EQUIPMENT 2

```
// Initialize DMX Equipment 2 to receive data 8 through 10 from the console packet
void DMX_Equipment2_Initialize (void){
    U1CON0bits.MODE = 0b1010;       // DMX mode
    U1CON0bits.TXEN = 0;            // Disable transmit
    U1CON0bits.RXEN = 1;            // Enable receive
    U1CON2bits.RXPOL = 0;           // Data polarity is not inverted
    U1P2H = 0x00;                   // Receive data 8
    U1P2L = 0x08;                   // :
    U1P3H = 0x00;                   // :
    U1P3L = 0x0A;                   // through 10
    U1BRGH = 0x00;                  // Initialize Baud Rate to 250k; Fosc = 64 MHz
    U1BRGL = 0x0F;
    U1CON2bits.STP = 0b10;          // 2 Stop bits

    Pin_Initialize();               // Set the corresponding RX pin via PPS

    U1CON1bits.ON = 1;              // Enable serial port
}
```

Example 3 shows a sample transmit routine for the DMX console. Storing transmit data values in an unsigned 8-bit type array provides ease in counting the number of bytes to transmit as the array index corresponds to the byte number.

EXAMPLE 3: DMX CONSOLE TRANSMIT

```
void DMX_Console_Transmit (uint8_t txLength, uint8_t *txDataArray){
    UART_Write (START_CODE);

    for(uint8_t i = 0; i < txLength; i++){   // txLength = n; Max value = 512
    UART_Write(txDataArray[i]);
    }
}
```

The sample routine shown in Example 4 utilizes a variable to specify the number of bytes to receive and an array where the received data are stored to be available for use in the user application code.

EXAMPLE 4: DMX EQUIPMENT RECEIVE

```
void DMX_Equipment_Receive (uint16_t rxLength){  // rxLength = number of bytes to
                                                 // receive
    uint8_t startCode;
    uint8_t rxDataArray[arraySize];              // arraySize value should be at least
                                                 // equal to rxLength
    while (!U1ERRIRbits.RXBKIF);                 // Wait for break reception
    U1ERRIRbits.RXBKIF = 0;

    startCode = UART_Read();                     // Save Start code
    for (uint8_t i = 0; i < rxLength; i++){
    rxDataArray[i] = UART_Read();                // Save Data
    }

    /**Some code for Start code and received data processing**/
}
```

LIN MODES

Local Interconnect Network (LIN) is a protocol used primarily in automotive and industrial applications.

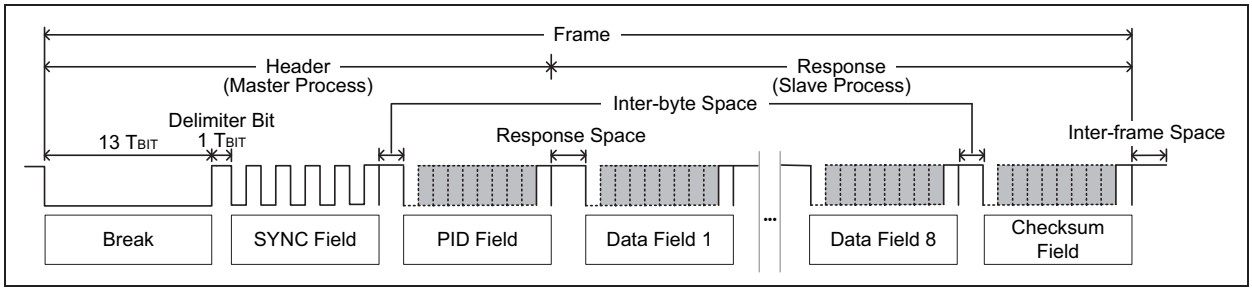
Note:	The different LIN modes are supported by the UART with protocol support only but the EUSART also has a few support features for the LIN protocol (refer to **Section "EUSART LIN Support"** Features subsection).

The UART with protocol support module features the following hardware support for LIN:

- Automatic checksum calculation
- Selection between legacy and enhanced checksum
- Automatic calculation of PID parity bits
- Automatic insertion of the Break, Delimiter Bit, Sync and PID in the serial stream
- Break detection on the RX pin
- Automatic baud rate detection for slave nodes

A LIN frame is composed of a master process and a slave process (see Figure 10). The UART with protocol support offers two modes for LIN: LIN Master/Slave mode and LIN Slave mode. The Master/Slave mode supports both the master and slave processes and is therefore implemented in master nodes. The Slave mode, on the other hand, only supports slave processes, and is implemented in slave nodes.

FIGURE 10: LIN FRAME



Example 5 shows sample initialization code for LIN Master/Slave mode using UART1. The LIN Slave mode follow the same initialization pattern except for the MODE<3:0> bits configuration which must be set to `0b1011`.

EXAMPLE 5: LIN MASTER/SLAVE MODE INITIALIZATION CODE

```
void LIN_MasterSlaveMode_Initialize (void){
    U1CON0bits.BRGS = 0;       // BRG operates in normal speed; Baud = Fosc/[16(n+1)]
    U1BRGH = 0x01;             // Initialize Baud Rate to 9600; Fosc = 64 MHz
    U1BRGL = 0x9F;
    U1CON0bits.MODE = 0b1100;  // LIN Master/Slave mode
    U1CON0bits.TXEN = 1;       // Enable transmit
    U1CON0bits.RXEN = 1;       // Enable receive
    U1CON2bits.TXPOL = 0;      // Data polarity is not inverted
    U1CON2bits.STP = 0b00;     // 1 Stop bit

    Pin_Initialize();          // Set the corresponding TX and RX pins via PPS

    U1CON2bits.C0EN = 1;       // Enhanced checksum; Clear to '0' for legacy checksum
    U1CON1bits.ON = 1;         // Enable serial port
}
```

Master Process

The UART generates the Break field as a space that is 13-bit periods wide and the Break delimiter as a mark that is one bit time long. It implements the eight data-bit asynchronous format for the Sync, PID and data fields. The number of Stop bits is selectable between 1, 1.5 and 2 bit periods. The Sync byte is a 'U' (55h) character while the PID byte is composed of six bits of frame identifier and two parity bits that are automatically calculated by the hardware. The Break, Delimiter bit, and Sync are automatically transmitted after writing the 6-bit ID to a dedicated register. Example 6 shows a sample code for the Master process.

EXAMPLE 6: MASTER PROCESS (TRANSMITTING A HEADER)

```
void LIN_MasterTransmitHeader (uint8_t slave_ID){
    U1P1L = slave_ID;
}
```

Slave Process

The PID determines which slave processes are expected to respond to the master. The slave process is responsible for transmitting the frame response when it is a publisher and for receiving the frame response when it is a subscriber. It is composed of the data and checksum fields. Data fields are the actual information transmitted or received during the slave process while the checksum field is used for error detection. Example 7 shows a sample code on how to get the PID during slave process while Example 8 and Example 9 show how to transmit and receive the data field using arrays.

EXAMPLE 7: GETTING THE PID DURING SLAVE PROCESS

```
uint8_t LIN_Get_PID (void) {
    uint8_t PID;

    while (!U1ERRIRbits.RXBKIF);  // Wait for break reception
    U1ERRIRbits.RXBKIF = 0;

    PID = UART_Read();

    return PID;
}
```

EXAMPLE 8: DATA FIELD TRANSMISSION DURING SLAVE PROCESS

```
void LIN_SlaveDataTransmit (uint8_t txLength, uint8_t *txDataArray){
    U1P2H = 0x00;
    U1P2L = txLength;                    // Size of the data array; Max value = 8

    for(uint8_t i = 0; i < txLength; i++){
        UART_Write(txDataArray[i]);
    }
}
```

EXAMPLE 9: DATA FIELD RECEPTION DURING SLAVE PROCESS

```
void LIN_SlaveDataReceive (uint8_t rxLength){      // rxLength = Number of data bytes
                                                   // to receive
      uint8_t rxDataArray[8];

      U1P3H = 0x00;
      U1P3L = rxLength;

      for (uint8_t i = 0; i < rxLength; i++) {
         rxDataArray[i] = UART_Read();             // Save data
      }

      /** Some code for checksum verification and processing of received data **/
}
```

Two methods for computing the checksum are available: legacy and enhanced. The legacy checksum includes only the data bytes while the enhanced checksum includes PID and the data bytes.

When a slave process sends data, the UART automatically calculates the checksum for the transmitted bytes as they are sent and appends the inverted checksum byte to the slave response. When a slave receives data, the checksum is accumulated on each byte as it is received using the same algorithm as the sending process. The last byte, which is the inverted checksum value calculated by the sending process, is added to the locally calculated checksum by the UART. The check passes when the result is all '1's, otherwise the check fails and the Checksum Error Flag bit is set. Example 10 shows how to verify if a checksum has passed or failed.

EXAMPLE 10: CHECKSUM VERIFICATION

```
bool LIN_VerifyRxChecksum (void){
    while(!PIR3bits.U1RXIF);

    if(U1ERRIRbits.CERIF == 0)
       return true;                          // Checksum passed
    }
    return false;
}
```

The checksum method, number of data bytes, and whether to send or receive data in slave process, is defined by software according to the PID. The inter-byte period, timeout and frame period are timed by software using a means other than the UART. On-chip hardware timers such as Timer0/1/2 can be used for this purpose.

EUSART LIN Support

The EUSART module of enhanced mid-range and high end PIC MCUs supports a fixed 13-bit Break transmission and wake-up on Break detection. The other tasks are left for the user to determine in software. Microchip's MPLAB® Code Configurator (MCC) LIN module allows the user to create schedule tables, and select checksum type and hardware timers through an easy to use GUI. It supports majority of devices with EUSART and is capable of generating software routines necessary for rapid prototyping purposes. For more information, refer to AN2059, *"LIN Basics and Implementation of the MCC LIN Stack Library on 8-bit PIC® Microcontrollers"*.

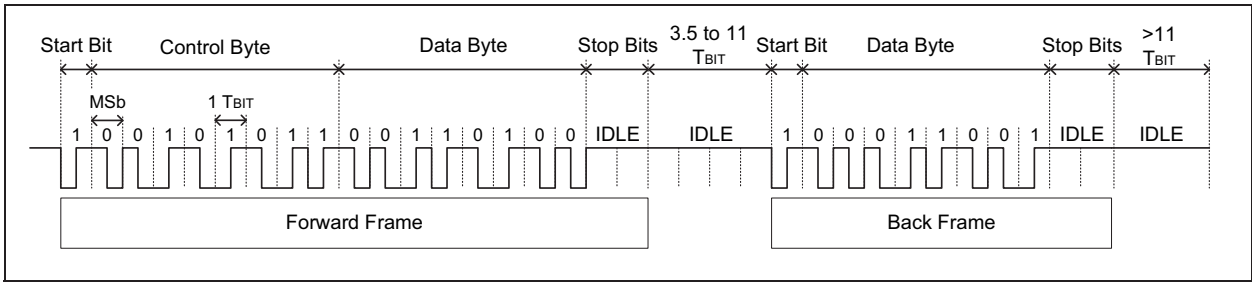
DALI MODE

Digital Addressable Lighting Interface (DALI) is a protocol used in digital lighting control in large buildings such as offices and factories. DALI support is only available in the UART with protocol support module. It provides the following features for DALI:

- Selectable modes between control device and control gear
- Manchester encoding
- Most Significant bit first
- No Stop periods between back-to-back bytes
- Two Stop bits transmission and detection
- Configurable forward and back wait period in half-bit steps
- Forward/Back Frame detection threshold delimiter configurable in half-bit steps
- Automatic transmission of back frame at the end of wait time

Two modes are available: DALI Control Device mode and DALI Control Gear mode. A control device is the main controller that sends out commands and data to the control gears (lighting fixtures). Figure 11 shows an example of DALI packets on a bus. All bytes are transmitted and received MSb first in Manchester encoded format.

FIGURE 11: DALI PACKETS



DALI Control Device Mode

Control Device mode is selected to transmit forward frames. The forward frame is the packet sent by the control device to the control gear. It consists of two bytes for DALI 1.0 or three bytes for DALI 2.0. Back-to-byte transmission without interruption is made possible by writing to the transmit buffer as soon as the transmit interrupt flag goes true and before the output shifter becomes empty. After the last Stop bit, the TX output is held in the Idle state for an additional wait time determined by a dedicated register. Example 10 shows a sample initialization code for the DALI Control Device.

EXAMPLE 11: DALI CONTROL DEVICE INITIALIZATION CODE

```
void DALI_ControlDevice_Initialize (void){
    U1CON0bits.MODE = 0b1000;   // DALI Control Device Mode
    U1CON0bits.TXEN = 1;        // Enable transmit
    U1CON0bits.RXEN = 1;        // Enable receive
    U1P1H = 0x00;               // 16 half-bit periods wait time after complete…
    U1P1L = 0x16;               // …transmission of forward frame
    U1P2H = 0x00;               // Frames following Idle periods more than 15 half-bit…
    U1P2L = 0x15;               // …periods are detected as Forward frames
    U1BRGH = 0x0D;
    U1BRGL = 0x04;              // Baud = 1200; Fosc = 64 MHz
    U1CON2bits.TXPOL = 0;       // Data polarity is not inverted; Set to '1' if inverted
                                // polarity is required
    U1CON2bits.STP  = 0b10;     // 2 Stop bits

    Pin_Initialize();           // Set the corresponding TX pin via PPS

    U1CON1bits.ON = 1;          // Enable serial port
}
```

The software waits for the output shifter to become empty before writing the next forward frame. If a back frame is received during the wait time, any bytes that may have been written to the Transmit Buffer register will not be transmitted until after the back frame plus the set wait time.

DALI Control Gear Mode

Control Gear mode is selected to receive forward frames and transmit back frames. The back frame is the response packet sent by the control gear back to the control device. DALI requires for a forward frame to be received first before a back frame can be transmitted. Only frames that follow an idle period longer than a set threshold value will be detected as forward frames. Example 11 shows a sample initialization code for the DALI Control Gear.

EXAMPLE 12: DALI CONTROL GEAR INITIALIZATION CODE

```
void DALI_ControlGear_Initialize (void){
    U1CON0bits.MODE = 0b1001;      // DALI Control Gear Mode
    U1CON0bits.TXEN = 1;           // Enable transmit
    U1CON0bits.RXEN = 1;           // Enable receive
    U1P1H = 0x00;                  // 8 half-bit periods wait time after complete…
    U1P1L = 0x08;                  // …reception of a forward frame
    U1P2H = 0x00;                  // Frames following Idle periods more than 15 half-
    U1P2L = 0x15;                  // bit periods are detected as Forward frames
    U1BRGH = 0x0D;
    U1BRGL = 0x04;                 // Baud = 1200; Fosc = 64 MHz
    U1CON2bits.TXPOL = 0;          // Data polarity is not inverted; Set to '1' if
                                   // inverted polarity is required
    U1CON2bits.RXPOL = 0;          // Same as TXPOL
    U1CON2bits.STP  = 0b10;        // 2 Stop bits

    Pin_Initialize();              // Set the corresponding TX pin via PPS

    U1CON1bits.ON = 1;             // Enable serial port
}
```

The forward frame is received one byte at a time in the receive FIFO and retrieved by reading the Receive Buffer register. The data received by a control gear from a forward frame is processed by the application software. It also decides whether a back frame should be transmitted or not. If a back frame data is written after the wait time has expired, the data is held until after the wait time following the next forward frame.

GENERAL PURPOSE MANCHESTER

General purpose Manchester is a subset of the DALI mode. It maintains all aspects of DALI mode. The only difference is that the wait time is set to zero. This allows full and half-duplex operation because writes are not held waiting for a receive operation to complete.

BAUD RATE GENERATOR

The Baud Rate Generator (BRG) is an 8-bit or 16-bit timer that is dedicated to support both the Synchronous and Asynchronous modes of operation.

The UART module allows selection between normal and high baud rate range. The high baud rate range is intended to extend the baud rate range when the desired baud rate is not possible otherwise. The normal baud rate range is recommended when the desired baud rate is achievable with either range.

Calculation of baud rate differs from each type of UART module and the configured speed. Baud rate formulas are usually given on the device data sheet. Table 2 shows the baud rate formulas for the PIC18FXXK42.

TABLE 2: PIC18FXXK42 BAUD RATE FORMULAS

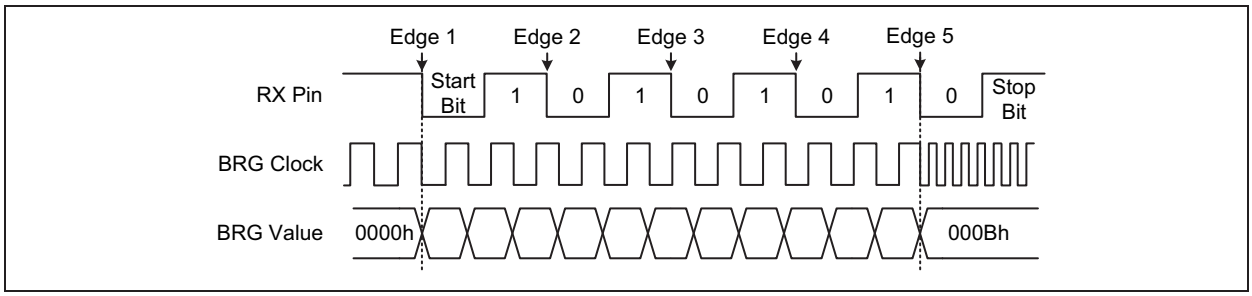
BRG/UART Mode	Baud Rate
High Rate	FOSC/[4(n+1)]
Normal Rate	FOSC/[16(n+1)]

The calculated baud rate is stored in a pair of 8-bit registers.

Auto-baud Detect

When Auto-Baud Detect (ABD) is active, the clock to the BRG is reversed. Rather than the BRG clocking the incoming RX signal, the RX signal is timing the BRG. The Baud Rate Generator is used to time the period of a received 55h (ASCII “U”), which is the Sync character for the LIN bus. The unique feature of this character is that it has five falling edges, including the Start bit edge, and five rising edges including the Stop bit edge. Refer to Figure 12.

FIGURE 12: AUTO-BAUD DETECTION



ABD is supported in 8-bit Asynchronous and LIN modes.

In 8-bit Asynchronous mode, auto-baud is enabled by setting the ABDEN bit. On the first falling edge of the receive line, the Baud Rate Generator begins counting up using the BRG counter clock. On the fifth falling edge, an accumulated value totaling the proper BRG period is left in BRG register pair, the ABDEN bit is automatically cleared and the ABD interrupt flag is set.

When LIN mode is selected, the hardware automatically performs ABD upon reception of the Sync character and setting of the ABDEN bit is neither necessary nor possible.

STOP BIT SELECTION

The UART with protocol support allows programmable number of Stop bit which includes:

- 1 transmit with receive verify on first
- 1.5 transmit with receive verify on first
- 2 transmit with receive verify on both
- 2 transmit with receiver verify on first only

In all modes except DALI, the transmitter is idle for the number of Stop bit periods between each consecutively transmitted word. In DALI, the Stop bits are generated after the last bit in the transmitted data stream.

AUTOMATIC CHECKSUM CALCULATION

The UART with protocol support has transmit and receive checksum adders. When enabled, the adders accumulate every byte that is transmitted or received. The accumulated sum includes the carry of the addition and is stored in a checksum register.

Software is responsible for clearing the checksum registers before a transaction and performing the check at the end of transaction. In LIN mode, automatic checksum calculation is always enabled and the hardware clears and checks the sums automatically.

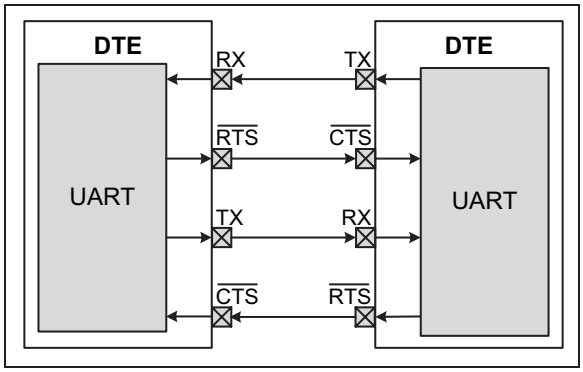
FLOW CONTROL

The UART with protocol support features both hardware and software flow control selectable through the handshake flow control bits. Flow control reduces the risk of data loss and prevents buffer overruns by signaling when a device is ready to transmit or receive data.

Hardware Flow Control

Hardware flow control uses a couple of microcontroller pins in addition to the TX and RX pins. The RS-232 signal names for these are $\overline{RTS}$ (Request to Send) and $\overline{CTS}$ (Clear to Send). A typical configuration between two Data Terminal Equipment (DTE) is shown in Figure 13. Each DTE uses the $\overline{RTS}$ to output if it is ready to accept new data and read $\overline{CTS}$ to see if it is allowed to send data to the other device. Both lines are low true and are configurable through the PPS module.

FIGURE 13: HARDWARE FLOW CONTROL CONFIGURATION



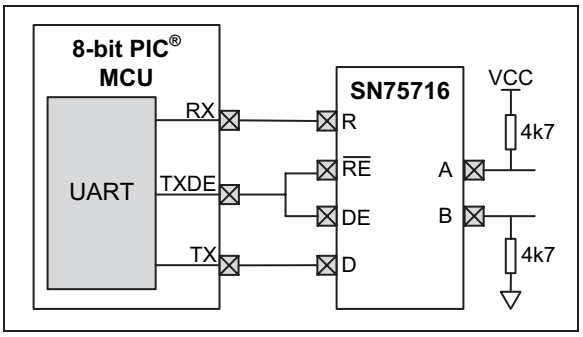
The UART receiving data asserts the $\overline{RTS}$ output low when the input FIFO is empty. When a character is received, the $\overline{RTS}$ output goes high until the receive buffer is read to free up both FIFO locations.

The UART sending data will only be able to transmit when the $\overline{CTS}$ input is detected low. When the $\overline{CTS}$ input goes high after a byte has started to transmit, the transmission will complete normally. The receiving UART accommodates this by accepting the character in the second FIFO location even when the $\overline{CTS}$ input is high.

RS-485 Transceiver Control

Another microcontroller pin may be used to control an RS-485 transceiver. The signal name for this is TXDE for Transmit Drive Enable. This output is high when the TX output is actively sending a character and low at all other times. Hardware flow control can be used to control the direction of the RS-485 transceiver. A sample configuration is shown in Figure 14. The TXDE pin is selected through the PPS module.

FIGURE 14: RS-485 CONFIGURATION



XON/XOFF Software Flow Control

XON/XOFF uses special characters to suspend and resume transmission. It requires full-duplex operation because the transmitter must be able to receive the signal to suspend transmitting while the transmission is in progress. The UART uses 13h for XOFF and 11h for XON.

The transmitter defaults to XON, or transmitter enabled. When an XOFF character is received, the transmitter stops transmitting after completing the character actively being transmitted. The transmitter remains disabled until an XON character is received.

On-chip software flow control allows reliable data exchange while reducing the cost of using additional pins required in hardware control. However, it has a drawback of latency since XON and XOFF characters are also inserted in the serial stream.

OPERATION WITH OTHER PERIPHERALS

This section provides a few sample implementations of the UART with other on-chip peripherals.

RX/TX Activity Timeout Using HLT

The Hardware Limit Timer (HLT) of even numbered timer modules can be used to monitor the activity of the UART TX and RX lines. The TX or RX lines can be configured as the Reset signal source for HLT. The HLT should be configured such that the UART activity will keep resetting the HLT to prevent a full HLT period from elapsing. When there has been no activity on the selected TX or RX line for longer than the HLT period, an HLT interrupt will occur signaling the timeout event.

UART Data Transfer Using DMA

Devices such as the PIC18FXXK42 feature an on-chip Direct Memory Access (DMA) controller. The DMA allows the UART to transfer data directly from the Receiver Buffer register to RAM or from RAM to the Transmit Buffer register, without intervention from the CPU. The UART RX and TX can also be configured to trigger an interrupt request to the DMA to start a new DMA message. Allowing the DMA to take over the memory bus during UART data transfers enables the CPU to perform other important tasks, resulting in a more efficient CPU usage and higher data throughput.

Digital Modulation Using UART and CLC

Several UART modules support internal connection of the UART transmit output to the Configurable Logic Cell (CLC) module. One significant implementation of the UART with CLC is in Data Signal Modulator (DSM). Digital modulation techniques such as On-Off Keying (OOK), Frequency Shift Keying (FSK), and Phase Shift Keying (PSK) are supported by the DSM. The CLC uses the UART as the input data source and other signal sources (e.g., System clock, LPRC, SOSC, etc.) as the carrier signal. Refer to AN2133, "*Extending PIC® MCU Capabilities Using CLC*" for detailed discussion.

CONCLUSION

This technical brief covers the different capabilities and features of the three existing varieties of UART module on 8-bit PIC microcontrollers. The UART with protocol support module was given much emphasis to help users understand quickly the different advanced features that are not supported by the USART and EUSART. Using this module allows designers to develop their UART-based applications with much less software overhead in a shorter period of time.

REFERENCES

- PIC18FXXK42 Data Sheet
- ANSI E1.11-2008 (R2013) Standard
- AN1659, "*DMX512A*" (DS00001659)
- AN2059, "*LIN Basics and Implementation of the MCC LIN Stack Library on 8-bit PIC® Microcontrollers*"
- AN1465, "*Digitally Addressable Lighting Interface (DALI) Communication*" (DS01465)
- AN2133, "*Extending PIC® MCU Capabilities Using CLC*"

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
═ ISO/TS 16949 ═

Trademarks

ISBN: 978-1-5224-1393-6

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820