Keywords: MAXQ, MAXQ610, UART, USART, serial, serial port

APPLICATION NOTE 4465

Using the Serial Port on the MAXQ610 Microcontroller

Aug 11, 2009

Abstract: The MAXQ610 microcontroller provides two standard USART serial ports. The asynchronous mode 1, one of the modes supported by the MAXQ610's serial ports, can be used to communicate with PC COM ports and many other types of legacy equipment. This application note explains how the interrupt handling support provided by the MAXQ610 for serial-port transmit and receive operations allows the application to respond quickly to the serial port when a new character is received or when a character has completed transmission.

Overview

The MAXQ610 microcontroller includes peripherals to allow communication with external devices or systems. These peripherals include a master/slave Serial Peripheral Interface (SPI™); a carrier generation/modulation system for infrared (IR) communication; and two separate universal synchronous/asynchronous receiver/transmitters (USARTs), commonly known as serial ports.

This application note demonstrates the use of the serial port on the MAXQ610 in 10-bit asynchronous mode. This mode is commonly used for communications with, and/or sending debug output to, a standard COM port on a PC. This mode allows two devices or systems to communicate by simply agreeing on a common bit format and baud rate, regardless of the actual operating frequency found on either side of the communications channel.

Demonstration code for this application note is written for the MAXQ610 and the MAXQ610 EV (evaluation) kit, using the assembly-based MAX-IDE environment. The code and project files for the demo application covered in this application note are available for download.

The following hardware and software are required to run the demonstration code.

- MAXQ610 EV kit board
- MAXQJTAG board (serial-to-JTAG interface)
- JTAG programming cable (2 x 5 female-to-female 0.100" header, 10-connector ribbon cable)
- DB9 straight-through male-to-female serial cable
- 5VDC regulated, center-post-positive power supply
- PC with available COM (serial) port or USB-to-serial adapter
- MAX-IDE environment
- Terminal emulator (such as MTK operating in the Dumb Terminal mode, HyperTerminal, or TeraTerm)

The latest installation package and documentation for the MAX-IDE environment are available for free download.

- MAX-IDE installation
- MAXQ Core Assembly Guide
- Development Tools Guide

To run the C version of the demonstration code, the IAR Embedded Workbench® must be installed. An evaluation copy of this software is available on the CD for the MAXQ610 EV kit.

Setting Up the MAXQ610 EV Kit

The demonstration code covered in this application note operates on the MAXQ610 EV kit. However, to make the code run properly and to enable the features in the code, the jumpers on the EV kit board must be configured correctly. The same jumper configuration is used on the kit board for both steps of running the demo (i.e., loading code and executing code).

Configure the jumpers on the MAXQ610 EV kit.

- JH1: connect pins 2 and 3.
- JH2: connect pins 2 and 3.
- JH3: connect pins 2 and 3.
- Connect the following jumpers (pins 1 and 2): JH14, JH20, JH22, JH23, JH24, JH25, and JH26.

Once the MAXQ610 EV kit board has been configured correctly, set up the hardware to compile and load the demonstration code.

- Connect jumpers JH1, JH2, and JH3 on the serial-to-JTAG board.
- Connect the 5VDC power supply to plug J2 on the serial-to-JTAG board.
- Connect the DB9 cable from the COM1 port on the PC to the J1 connector on the serial-to-JTAG board.
- Connect the JTAG programming cable from P2 on the serial-to-JTAG board to P5 on the MAXQ610 EV kit board. The red wire on the JTAG cable should go to pin 1 (TCK) on both connectors.
- Turn on the power to the 5VDC supply.

Serial Port Modes and Initialization

Each of the two USART peripherals on the MAXQ610 can operate in either a synchronous or an asynchronous mode. When operating in the synchronous mode (mode 0), the MAXQ610 acts as a bus master for all transactions. It configures the TXD line to act as a shift clock for both transmit and receive operations, while RXD is used as a bidirectional data line. Under this arrangement, data can only be transmitted in one direction at a time (half-duplex), and only at the request of the master.

In the asynchronous operation modes (modes 1, 2, and 3), there are two unidirectional data lines: TXD and RXD. The TXD line carries asynchronous data from the MAXQ610's serial port to an external device, while the RXD line carries asynchronous data from the external device back to the MAXQ610. The MAXQ610 and the external device must agree on a common format (number of data bits, parity bit, and stop bit) and a common baud rate (bus frequency) in order to communicate. These modes allow data to be transmitted in a full-duplex manner, since transfers on the TXD and RXD lines can occur independently and do not need to be synchronized to one another. The MAXQ610 can transmit data to the external device on the TXD line at any point, whether or not the external device is currently transmitting on RXD, and vice versa.

For this demonstration, we will select mode 1 which has the following characteristics.

- Asynchronous transmission (on TXD) and reception (on RXD) of serial data
- Baud clock provided by a dedicated baud clock generator (programmable using the PR register)
- Eight data bits, 1 start bit, 1 stop bit
- No parity

The baud rate (bits transmitted/received per second) used by the serial port in mode 1 is determined by the following two equations (as discussed in the MAXQ Family User's Guide):

$$\text{Baud clock frequency (BAUD)} = \text{system clock frequency} \times PR/2^{17} \quad \text{(Eq. 1)}$$

$$\text{Baud rate} = BAUD \times 2^{(SMOD \times 2)}/2^{6} \quad \text{(Eq. 2)}$$

Equation 1 describes the output of the baud clock generator. The frequency of the output generated is controlled by the contents of the 16-bit PR (phase) register. Loading a higher value into the PR register results in a higher-frequency baud clock.

Equation 2 factors in the effects of a control bit (SMOD) in the SMD or Serial Port Mode register. Setting this bit to 1 increases the final baud rate by a factor of 4. This means that the Equation 2 can also be written in either of the following ways, depending on the value of SMOD:

$$\text{When SMOD=0, baud rate} = BAUD/64 \quad \text{(Eq. 3)}$$

$$\text{When SMOD=1, baud rate} = BAUD/16 \quad \text{(Eq. 4)}$$

For most crystal-speed and baud-rate combinations, either setting for SMOD can be used. Note that the setting of SMOD affects the value to be loaded into the PR register. For our application, we will select SMOD=1, which means that the baud-rate equation reduces to the following:

$$\text{Baud rate} = (\text{system clock frequency} \times PR/2^{17})/16 \quad \text{(Eq. 5)}$$

Or:

$$\text{Baud rate} = (\text{system clock frequency} \times PR)/2^{21} \quad \text{(Eq. 6)}$$

Solving for the PR value gives us the equation that most interests us. Typically, the baud rate and system clock frequency are fixed, and we simply want to know the desired PR register setting that will produce the intended baud rate. Therefore:

$$PR = \text{baud rate} \times 2^{21}/(\text{system clock frequency}) \quad \text{(Eq. 7)}$$

For example, use the standard 12MHz crystal found on the MAXQ610 EV kit and select a baud rate of 9600 baud. The desired setting for the PR register is ($9600 \times 2^{21}/12000000$), which is approximately 1677 or 068Dh.

Once the values for PR and SMOD have been calculated, initializing the serial port is simply a matter of setting the USART to the proper mode and writing the PR and SMOD values. Since we are using serial port 0, the code will be writing to the appropriate set of registers for that serial port (SCON0, SMD0,

PR0, and SBUF0).

```
;=========================================================================
;=
;=  InitSerial0
;=
;=  Set up serial port 0 to run in 10-bit asynchronous mode at
;=  9600 baud.
;=

InitSerial0:
   move    SCON0.6, #1       ; Set to mode 1 (10-bit asynchronous).
   move    SCON0.4, #1       ; Enable receiver.
   move    SMD0.1,  #1       ; Baud rate = 1/16 of baud clock
   move    PR0, #0068Dh      ; P = 2^21 × 9600/12.000MHz

   move    SCON0.0, #0       ; Clear received character flag.
   move    SCON0.1, #0       ; Clear transmit character flag.
   move    SMD0.2,  #1       ; Enable receive/transmit interrupt.
   ret
```

For more information on the operating modes of the USART peripheral, refer to the MAXQ610 User's Guide and to **Section 10 (Serial I/O Module)** of the MAXQ Family User's Guide.

Transmitting and Receiving Characters

Once the serial port has been properly configured, it is ready to transmit and receive characters. In mode 1, a character consists of 1 start bit, 8 data bits, and 1 stop bit. The start and stop bits are used for synchronization purposes and are handled by the USART hardware. The remaining 8 bits carry the actual data, which means that a single 8-bit byte can be transmitted or received by the serial port at once.

Transmitting a character over the serial port involves the following three steps.

1. Write the byte value to transmit to the SBUF register.
2. Wait for the TI (transmit interrupt) bit in the SCON register to go high (set to 1). This indicates that the hardware has finished transmitting the character over the serial port.
3. Clear the TI bit to 0.

It is always necessary to wait for the completion of each transmission before starting a new one. The time required for a serial-port transmission is long, compared to the instruction execution cycle time. Consequently, it is possible to execute other operations in step 2 while waiting for the serial-port transmission to complete. For example, when transmitting at a 115200 baud rate using a 12MHz crystal, the total time required for the transmission is approximately 10 × (1/115200) or 87µs. In this same time frame, the MAXQ610 can execute as many as 1041 instructions (87µs/(1/12MHz)).

Receiving a character over the serial port is accomplished in a similar manner.

1. Wait for the RI (receive interrupt) bit to go high, which indicates that a new character has been received by the serial port.
2. Read from the SBUF register to obtain the data byte.
3. Clear the RI bit to 0.

Only one received character is stored (buffered) in the serial-port hardware at any given time. This means that if the RI bit is set to 1 by hardware (indicating that a character has been received), the character must be obtained by reading the SBUF register; the RI bit must be cleared before the next character is received by the serial port. If a new character is received by the serial port and the RI bit is still high, the new character will be lost.

Handling Serial-Port Interrupts

A simple method of operating the serial port would be to simply poll (check repeatedly) the values of the RI and TI bits, as needed, for each transmit or receive operation. For example, when writing a character, the application code would write the data byte to SBUF and then poll the TI bit until it was set to 1 by hardware. No other operations would occur until the transmission had completed. Similarly, to receive a new byte the application would simply poll the RI bit until it was set high by hardware, then unload the received byte by reading SBUF.

This method of polling the RI and TI bits sacrifices performance for simplicity, because considerable time is spent waiting for characters to be transmitted or received. Additionally, the application must know in advance when to expect a new character from the external device, and the application cannot transmit and receive characters at the same time.

A more flexible (but slightly more complicated) method takes advantage of the fact that TI and RI are not merely status bits, but are also synchronous interrupt sources. Instead of continuously polling the status of the TI and RI bits to determine when a transmit or receive operation has completed, the MAXQ610 can enable an interrupt to occur when TI or RI changes from 0 to 1. This method allows the application to spend its time more productively, as it only responds to the serial port when needed.

The first step in this process is to enable all interrupts by setting the IGE bit (IC.0) to 1. This enables interrupt handling at a global level for the entire MAXQ610.

Next, an interrupt handling routine must be installed. The MAXQ610 uses a multiple fixed-interrupt vector system, with different (nonprogrammable) vector addresses assigned to each interrupt source or group of interrupt sources. For our purposes, we are interested in the Serial Port Interrupt vector, which is assigned to word address 0040h.

The interrupt handler routine below performs the following functions.

1. Prevents other interrupts (of a higher priority) from occurring by temporarily setting IC.0 to 0.
2. Pushes the PSF and Acc registers to the stack to save their values. Since an interrupt can be triggered at any time, interrupt handlers must always save and restore any registers that are used by other application code.
3. Since the Serial Port Interrupt can be triggered by either the TI (transmit) or RI (receive) flags, the handler code must check to see why the interrupt was triggered.
4. If a transmit interrupt was triggered, clear the TI bit and set a flag value (stored in A[15]) to indicate to the application that the transmission had completed. This is simply for demonstration purposes; the interrupt handler could also react to this condition by loading a new value into SBUF to transmit the next character.
5. If a receive interrupt was triggered, clear the RI bit; read the received character from SBUF; and take the appropriate action, depending on the character that was received.
6. Restore the Acc and PSF register values by popping them from the stack.
7. Re-enable interrupts by setting IC.0 to 1.
8. Exit the interrupt handler using the RETI instruction.

```
org 0040h

serialInt:
   move    IC.0, #0          ; Block any other interrupts from triggering.
   push    PSF
   push    Acc
   move    C, SCON0.0        ; Check for receive character interrupt.
   jump    C, serialInt_Rx
```

```
serialInt_Tx:
   move    SCON0.1, #0       ; Clear transmit complete interrupt flag.
   move    A[15], #1         ; Set flag to indicate transmit complete.
serialInt_done:
   move    IC.0, #1          ; Re-enable interrupts.
   pop     Acc
   pop     PSF
   reti

serialInt_Rx:
   move    SCON0.0, #0       ; Clear receive character interrupt flag.
   move    Acc, SBUF0        ; Get character from serial port.
   cmp     #'0'
   jump    E, serialInt_Rx0
   cmp     #'1'
   jump    E, serialInt_Rx1
   cmp     #'2'
   jump    E, serialInt_Rx2
   cmp     #'3'
   jump    E, serialInt_Rx3
   cmp     #'4'
   jump    E, serialInt_Rx4
   jump    serialInt_done

serialInt_Rx0:
   move    Acc, PO3
   or      #0Fh              ; Turn all LEDs off.
   move    PO3, Acc
   jump    serialInt_done


....


serialInt_Rx3:
   move    Acc, PO3
   xor     #04h              ; Toggle P3.2 state.
   move    PO3, Acc
   jump    serialInt_done

serialInt_Rx4:
   move    Acc, PO3
   xor     #08h              ; Toggle P3.3 state.
   move    PO3, Acc
   jump    serialInt_done
```

Building the Demo Application

The overall framework of the serial demo application is as follows.

1. Initialize the port pins and serial port.
2. Enable interrupts.
3. Transmit the following banner text over the serial port.

```
            MAXQ610 Serial Port Demo
            Type characters "1"-"4" to toggle LEDs or '0' to turn all LEDs
   off.
```

4. As the characters "1," "2," "3," "4," or "0" are received, change the LED states appropriately.

As shown in the interrupt handler code above, the application responds to received characters by changing the output state of port pins P3.0, P3.1, P3.2, and P3.3. On the MAXQ610 EV kit board (with jumpers JH22, JH23, JH24, and JH25 closed), these port pins drive the 4 LEDs DS1, DS2, DS3, and DS4. A low state on a port pin (0) causes the LED to turn on (light), while a high state on a port pin

turns the LED off.

```
main:
   move    WDCN,  #0
   move    PD3.0, #1         ; Set P3.0 to output mode.
   move    PO3.0, #1         ; Drive P3.0 high (LED off).

   move    PD3.1, #1         ; Set P3.1 to output mode.
   move    PO3.1, #0         ; Drive P3.1 low (LED on).

   move    PD3.2, #1         ; Set P3.2 to output mode.
   move    PO3.2, #1         ; Drive P3.2 high (LED off).

   move    PD3.3, #1         ; Set P3.3 to output mode.
   move    PO3.3, #0         ; Drive P3.3 low (LED on).

   move    PD0.2, #1         ; Set P0.2 (TXD) to output mode.
   move    PO0.2, #1         ; Idle high when not transmitting.

   call    InitSerial0
   move    IC.0, #1          ; Enable interrupts.

   move    LC[0], #12000     ; Give transceiver time to power on.
   djnz    LC[0], $

   ; Print string to serial port.

   move    CP, #0800h
printLoop:
   move    Acc, @CP++
   nop
   jump    Z, printLoop_done
   move    GR, Acc
   move    Acc, GRL
   call    TxChar0
   move    Acc, GRH
   call    TxChar0
   jump    printLoop
printLoop_done:
   nop

mainLoop:
   nop
   jump    mainLoop


;=========================================================================
;=
;=  TxChar0
;=
;=  Outputs a character to serial port 0.
;=
;=  Inputs  : Acc - Character to send.
;=

TxChar0:
   push    Acc
   move    SBUF0, Acc        ; Send character.
   move    A[15], #0         ; Clear interrupt service routine flag.
TxChar0_Loop:
   nop
   nop
   nop
   move    Acc, A[15]        ; Check flag.
   jump    Z, TxChar0_Loop
   move    SCON0.1, #0       ; Clear the transmit flag.
   pop     Acc
   ret
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
org 800h

   db 0Dh, 0Ah, 0Dh, 0Ah
   db "MAXQ610 Serial Port Demo", 0Dh, 0Ah, 0Dh, 0Ah
   db "Type characters '1'-'4' to toggle LEDs or '0' to turn all LEDs off"
   db 0Dh, 0Ah, 0Dh, 0Ah
   dw 0000h
```

The banner text is stored using DB statements starting at word address 0800h. This constant data, which is loaded into flash memory on the MAXQ610 along with the rest of the program code, can be retrieved using the CP code pointer as shown above (printLoop). Since each fetch from program memory using CP retrieves one word of data, two characters are transmitted over the serial port after each word is read.

Running the Demo

Once the demo code has been compiled using MAX-IDE and loaded into the MAXQ610 EV kit, it can be executed as follows.

1. Turn power off and disconnect the JTAG, power and serial cables.
2. Connect the serial cable from COM1 on the PC to plug J1 on the MAXQ610 EV kit board.
3. Connect the power cable to plug J3 on the MAXQ610 EV kit board.
4. Turn power on.
5. Open your terminal emulator program on the PC. Configure it to communicate over COM1 at 9600 baud with 8 data bits, 1 stop bit, and no parity.
6. Press and release RESET (SW1) on the MAXQ610 EV kit board. The banner text (MAXQ610 Serial Port Demo...) should display on your terminal emulator screen. If it does not, check your connections and jumper settings.
7. Type characters to toggle the LED states as follows: type "1" to toggle DS1; type "2" to toggle DS2; type "3" to toggle DS3; type "4" to toggle DS4; or type "0" to turn all LEDs off. Characters are not echoed.

Demonstration Code in C

The demonstration code below shows the same application, implemented in C for IAR's™ Embedded Workbench IDE. For simplicity, the interrupt handling code has been removed. Note that the `putchar` function has been implemented to output a character over serial port 0. This allows the use of standard I/O library functions such as `puts()` and `printf()`.

```
int putchar(int c)
{
   SBUF0 = c;
   while ((SCON0 & 0x02) == 0);
   SCON0 = (SCON0 ^ 0x02);
   return c;
}

void initUSART0(void)
{
   int i2;
   PD0_bit.bit2 = 1;   // Hold Tx0 line High.
   PO0_bit.bit2 = 1;   // Hold Tx0 line High.
   SMD0 = 2;           // Set baud rate select bit.
   SCON0 = 0x50;       // Set mode 1 and receive enable for UART 0.
   PR0 = 0x068D;       // 9600 baud: PR0 = 2^21 * 9600 / 12.000MHz
   for (i2 = 1; i2 < 10000; i2++);   // Give transceiver time to power on.
```

```
   SCON0 = 0x50;
}

void main( void )
{
   int c;
   IC_bit.IGE = 0;
   WDCN       = 0;
   PD3 = 0x0F;
   PO3 = 0x05;    // Default - DS2 and DS4 on, DS1 and DS3 off

   initUSART0();
   puts("MAXQ610 Serial Port Demo");
   puts("Type characters "1"-"4" to toggle LEDs or "0" to turn all LEDs
off\n");

   while (1) {
      while ((SCON0 & 0x01) == 0);        // Wait for RI flag to go high.
      c = SBUF0;                          // Receive character.
      SCON0 = (SCON0 ^ 0x01);             // Clear RI flag.

      switch (c) {
         case '0' : PO3 = 0x0F;
                    break;
         case '1' : PO3 = (PO3 ^ 0x01);   // Toggle P3.0
                    break;
         case '2' : PO3 = (PO3 ^ 0x02);   // Toggle P3.1
                    break;
         case '3' : PO3 = (PO3 ^ 0x04);   // Toggle P3.2
                    break;
         case '4' : PO3 = (PO3 ^ 0x08);   // Toggle P3.3
                    break;
         default  : break;
      }
   }
}
```

Conclusion

The MAXQ610 provides two instances of the standard USART serial ports found on many MAXQ microcontrollers. One of the modes supported by the MAXQ610's serial ports, the asynchronous mode 1, can be used to communicate with PC COM ports and many other types of legacy equipment. The interrupt handling support provided by the MAXQ610 for serial-port transmit and receive operations allows the application to respond quickly to the serial port when a new character is received or a character has completed transmission.

The USART serial ports on the MAXQ610 support a wide range of configuration options and communications modes which are beyond the scope of this application note. For more information, refer to the *MAXQ610 User's Guide* and section 10 of the *MAXQ Family User's Guide* noted above.

Related Parts		
MAXQ610	16-Bit Microcontroller with Infrared Module	Free Samples

More Information
For Technical Support: http://www.maximintegrated.com/support
For Samples: http://www.maximintegrated.com/samples
Other Questions and Comments: http://www.maximintegrated.com/contact

Application Note 4465: http://www.maximintegrated.com/an4465
APPLICATION NOTE 4465, AN4465, AN 4465, APP4465, Appnote4465, Appnote 4465