Keywords: MAXQ3210, UART, serial, serial port, USART, MAXQ3212

May 03, 2007

APPLICATION NOTE 4041

# Implementing a Software UART on the MAXQ3210

*Abstract: An asynchronous UART (serial port) is common on many microcontrollers, and provides a simple way for two devices to communicate without having to match system clock rates. This application note describes how to implement a 10-bit asynchronous UART in software on the MAXQ3210 microcontroller using two standard port pins. This simple modification allows the serial port to communicate over RS-232 and RS-485 networks, or to connect to a PC COM port.*

## Overview

A standard, asynchronous UART or serial port is a common feature on many microprocessors and microcontrollers. Asynchronous serial ports provide a simple way for two microcontrollers (or a microcontroller and a slave serial device) to communicate without having to match their system clock rates. By adding an appropriate voltage-level shifter, the serial port can also be used to communicate over RS-232 and RS-485 networks, or to connect to the COM port of a PC. Only two signal lines (Rx and Tx) are required to implement a full-duplex interface. If the devices on both sides of the serial link use the same voltage levels, bit format, and baud rate, then they do not need to know anything else about the another to transfer data successfully.

This application note describes a software implementation of the common 10-bit asynchronous serial port. The low-power MAXQ3210 microcontroller, which does not include hardware serial ports, will serve as our example. The technique presented here can also be used to add one or more serial ports to any MAXQ® microcontroller like the MAXQ2000, if those devices' hardware-based serial ports are insufficient for the application.

## Benefits of a Software UART

One can ask, why take to the trouble to implement a UART in software using a microcontroller's port pins when so many microcontrollers have serial ports provided in hardware? There are several reasons.

Although many microcontrollers have UARTs implemented in hardware, many do not. When selecting a microcontroller for a system design, it can be difficult to find a perfect fit for every design requirement. The ability to "fill in the gaps" in a microcontroller's feature set by implementing required peripherals in software increases the number of microcontrollers that can be considered for a given design. More choice among devices increases design flexibility.

A microcontroller can have a perfectly good hardware UART, but it still could be inadequate for the application in design. Perhaps the microcontroller needs to communicate with a peripheral that runs on a slightly modified version of the serial protocol. Perhaps the number of bits, parity checking features, or input and output buffers provided by the hardware UART do not fit the application requirements exactly. By building a software UART, the designer has far more flexibility in defining the UART capabilities and the details of the serial protocol.

A microcontroller can have hardware UARTs ideal for an application, but simply not enough of them. Why add a new chip to the design simply to increase the number of serial ports? One can simply add another UART in software, a new UART that matches the capabilities and feature sets of the microcontroller's existing UARTs.

It is also important to consider how much bandwidth a UART in software will drain from the main application. Most would agree that a principal reason for implementing a UART (or any other serial-communications peripheral) in hardware is to free the microcontroller from the low-level details of the serial protocol. Indeed, this is why the tedious bit-sampling, time-slot counting, and input and output bit shifting are all handled in hardware. This is why the UART alerts the main microcontroller (through an interrupt or other indicator flag) that it has either received a character or has finished transmitting one. The microcontroller can then quickly load or unload data from the UART buffer and return to its core tasks.

In simple terms, if implementing a software UART means that the application will spend considerable time spinning in

place while it watches port pins for serial activity, this is probably not worth doing.

Implementing a UART in software, however, does not drain crucial microcontroller time and resources. Consider the standard 10-bit asynchronous serial protocol (1 start bit, 1 stop bit, 8 data bits) when transmitting and receiving a character (**Figure 1**).

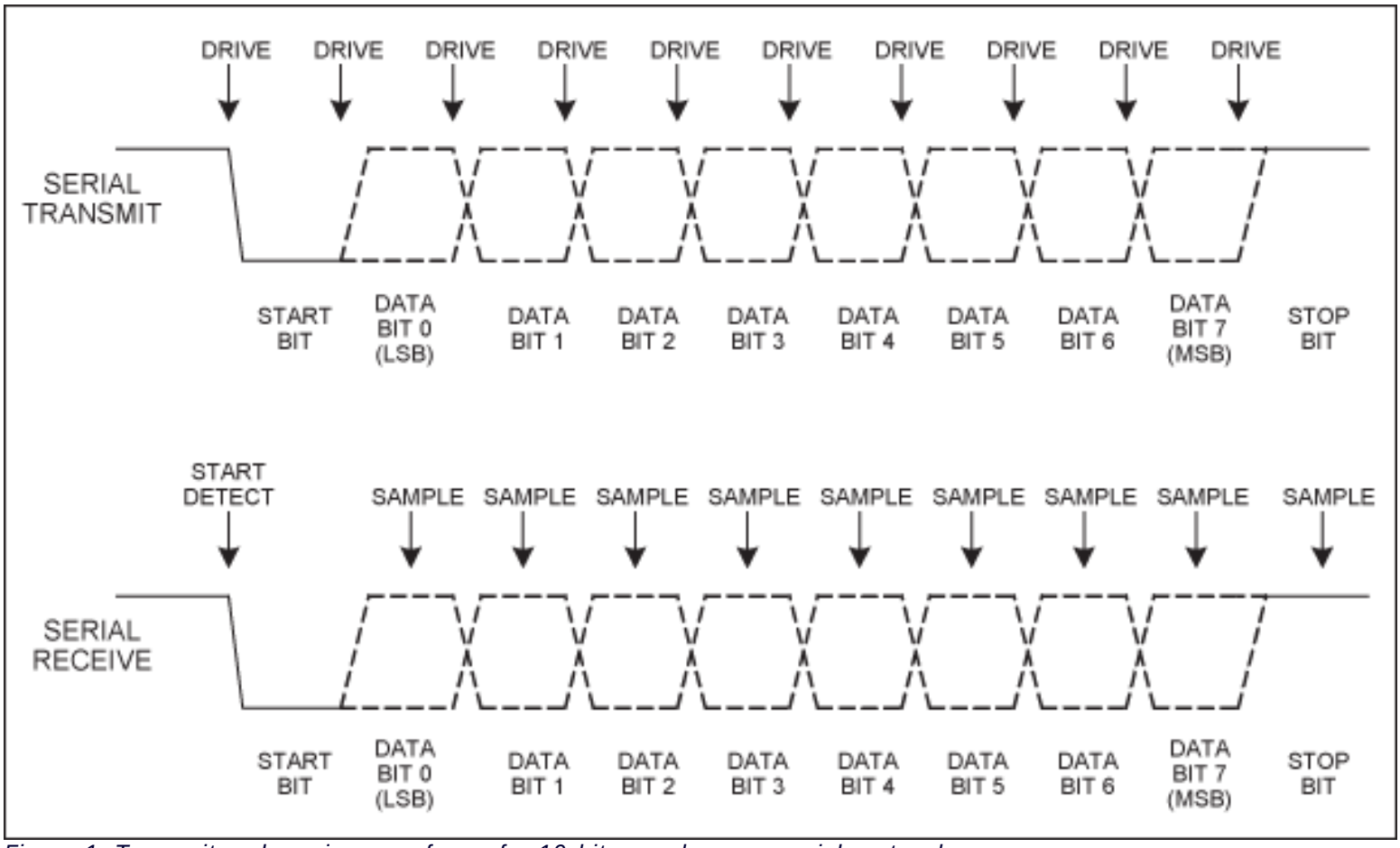

*Figure 1. Transmit and receive waveforms for 10-bit asynchronous serial protocol.*

Once a transmit or receive operation starts, the serial UART (in either software or hardware form) does not need to monitor the I/O lines continuously. When transmitting a character, the UART only needs to drive the state of the transmit line once every bit period; the UART is setting the level for each bit in turn, from the start bit through eight data bits to the stop bit. When receiving a character, the UART begins its activity on the first falling edge. After this point, the UART needs to sample the receive line state at least once per bit, in the middle of each bit time slot.

## How a Software UART Functions

We can represent the desired behavior of our software UART as a pair of state machines, one machine dedicated to transmitting characters and one to receiving them. For a full-duplex UART, both state machines run in parallel, which requires two independent timer interrupts. Both state machines have active and inactive modes. The "transmit" state machine transitions out of its idle state when given a character to transmit; it returns to an idle state following the transmission of the stop bit. The "receive" state machine transitions from its idle state when it detects a falling edge on the receive line. Once the receive state machine sees this initial low state (which indicates that the start bit has begun), it begins counting off its bit time-slots and samples the line for each bit as needed, including the stop bit.

To occupy as little of the main application's time as necessary, the UART state machines should be activated by periodic timer-based interrupts. The initial falling-edge detection for the receive line is handled separately, with an external edge-triggered interrupt. If a state machine's timer is set to initiate an interrupt once per bit-period, the state machines can perform any needed operations (and advance to the next state, if needed) each time that the interrupt is triggered. The code needed to implement the state machines should be optimized as much as possible, since it will be continuously running in the background whenever the software UART is active.

## Targeting the MAXQ3210

To understand the amount of processing power required for this UART functioning, consider a software UART implemented for the MAXQ3210 microcontroller. This 5V, 28-pin microcontroller runs at approximately 3.57MIPS and contains fifteen port pins, but no built-in UARTs. The MAXQ3210 also contains the other requirements for our design: a periodic timer, capable of generating interrupts for the transmit and receive state machines; and an external interrupt for falling-edge detection on the receive line. Since the MAXQ3210 only contains one general-purpose timer, the software UART will be a half-duplex implementation, which means that characters can only be transmitted or received at different times, not simultaneously. However, for many communications and control protocols, this half-duplex operation is not an issue.

On the MAXQ3210 the receive and transmit state machines, along with supporting routines to set modes and load and unload characters, can be implemented with a total instruction count of 171 words. For example, the transmit state machine only requires three states (data bit, stop bit, and return to idle) as shown below.

```
intTX_bit:
   move    GRH, PSF
   move    GRL, AP
   move    T2CNB.3, #0       ; Clear timer 2 overflow flag
   move    AP, #5
   rrc                       ; Start with least significant bit
   jump    C, intTX_bit_one
intTX_bit_zero:
   move    TXDO, #0
   jump    intTX_bit_next
intTX_bit_one:
   move    TXDO, #1
   jump    intTX_bit_next
intTX_bit_next:
   djnz    LC[1], intTX_bit_done
   move    IV, #intTX_stop
intTX_bit_done:
   move    AP, GRL
   move    PSF, GRH
   reti

intTX_stop:
   move    T2CNB.3, #0       ; Clear timer 2 overflow flag
   move    TXDO, #1          ; Float high
   move    IV, #intTX_idle
   reti

intTX_idle:
   move    A[4], #SER_MODE_TX_IDLE
   move    T2CNB.3, #0       ; Clear timer 2 overflow flag
   move    T2CNA.7, #0       ; Disable timer 2 interrupts
   move    T2CNA.3, #0       ; Stop timer
   reti
```

The complete code for this application note is available for download (ZIP, 5.6kB).

The worst-case interrupt code path for any of the states is 19 instruction cycles. **Table 1** shows the worst-case bandwidth estimates depending on baud rate. Note that the worst-case bandwidth only applies when characters are transmitted or received continuously. Once a character completes transmission, the UART shuts down, thus allowing the main application to run without interruption until the next character transmission begins.

**Table 1. Bandwidth Required for UART Implementations on the MAXQ3210**

| Baud Rate | Cycles per Timer Tick (at 3.57MHz) | Worst-Case Interrupt Cycle Count | Bandwidth Used by UART (%) |
|---|---|---|---|
| 9600 | 372 | 19 | 5 |
| 19200 | 186 | 19 | 10 |
| 28800 | 124 | 19 | 15 |
| 57600 | 62 | 19 | 31 |

## Conclusion

This application note shows how easy it is to implement a 10-bit asynchronous UART in software using two standard port pins. The techniques can be used to implement any type of serial peripheral, from SPI™ to SMBus™ to I²C, as long as the microcontroller has enough free port pins.

This test case uses a relatively low-speed MAXQ3210 microcontroller with negligible impact on core processing resources. A faster microcontroller like the MAXQ2000 or a high-speed 8051-compatible device would implement one or more UARTs in half- or full-duplex modes (or even more complex serial communications peripherals) in software without taking too much bandwidth from the main application. Finally, with the communications peripherals implemented in software, any aspect of the protocols can be updated to keep pace with changes to other devices or to evolving standards. Implementing a UART in software offers designers maximum flexibility and speed when creating new designs.

A similar article appeared in the online version of *Embedded Systems Design* in February 2007.



---

Application Note 4041: www.maxim-ic.com/an4041

**More Information**
For technical questions and support: www.maxim-ic.com/support
For samples: www.maxim-ic.com/samples
Other questions and comments: www.maxim-ic.com/contact

---

**Keep Me Informed**
Preview new application notes in your areas of interest as soon as they are published. Subscribe to EE-Mail - Application Notes for weekly updates.

---

**Related Parts**

MAXQ3210: QuickView -- Full (PDF) Data Sheet -- Free Samples

MAXQ3212: QuickView -- Full (PDF) Data Sheet -- Free Samples