ECE 353
Course Supplement #4

ADuC7026 UART

Michael G. Morrow

Table of Contents

1 ADuC7026 UART Operation

1.1 Fundamentals of Operation

1.1.1 Transmitter

The fundamentals of UART transmitter operation are covered in the course text.

1.1.2 Receiver

The fundamentals of UART receiver operation are covered in the course text. The ADuC7026 UART operates with an internal clock that is 16x time baud rate, permitting it to sample the incoming data in the middle of the bit interval.

1.1.3 Network Addressable Mode

The UART also supports a network addressable mode that will be covered only briefly here. In this mode, a serial network of up to 256 nodes can operate with a single master node or in a multi-master organization. The transmit data output pin can be configured with a 3-state driver to facilitate the direct connection of multiple devices' transmit data lines. If a node wants to communicate with another node, it first transmits the destination address, and then sends the data for that address. What would normally be the parity bit is used to distinguish between address and data frames. The necessary network protocols must be implemented in software.

1.2 Modem Control Signals

The UART modem control signals are used to provide a hardware interface with another serial device. In the original RS-232 implementation, the serial device was envisioned as a data communication equipment (DCE) and was normally a modem. The microprocessor UART is considered the Data Terminal Equipment (DTE). This history leads to the presence of such UART input signals as Ring Indicator (RI) and Data Carrier Detect (DCD). The remaining modem control signals supported are listed below.

- Data Set Ready (DSR) is an input to the UART that is interpreted as meaning the modem (the data set) has a connection.
- Data Terminal Ready (DTR) is an output from the UART that informs the modem that the terminal is ready. In the ADuC7026, this signal is directly controlled by the COMCON1.DTR bit.
- Ready To Send (RTS) is an output from the UART that is used to signal the modem that the UART has data ready to send. In the ADuC7026, this signal is directly controlled by the COMCON1.RTS bit.
- Clear To Send (CTS) is an input to the UART that indicates that the UART can send data to the modem. The combination of RTS and CTS is used to form a hardware flow control mechanism.

Any changes in the state of the CTS, DSR, and DCD inputs, as well as the detection of the trailing edge of RI, can be used to generate an interrupt. The ADuC7026 UART does not actually implement the RTS/CTS flow control mechanism in hardware. Rather, the signal input change interrupt and the ability to control the state of RTS allow for a

software implementation, if desired. Given the relatively low baud rates that are commonly used with asynchronous serial communications as compared to the microprocessor speed and interrupt latency, flow control is no longer commonly implemented.

1.3 Baud Rate Selection

There are two modes for selecting the UART baud rate. The normal 16450 mode is based on a simple divide-by-N counter. To obtain more precise baud rates, the fractional divider mode uses a more complex divider scheme (described in detail later). In both cases, note that the baud rate divider registers are only accessible by setting the COMCON0.DLAB bit to 1.

1.3.1 Normal Baud Rate Selection

In the normal baud rate mode, the baud rate is generated from the CPU clock. The CPU clock is derived from the 41.78MHz clock as divided by the clock divider configured in the POWCON register CD bit field. The CPU clock drives a divide-by-N counter from which the final baud clock is derived, where the N is a 16-bit value formed by the concatenation of the COMDIV1 and COMDIV0 registers. The baud rate is determined by the formula

$$\text{baud rate} = \left(\frac{41.78\text{MHz}}{2^{CD}}\right)\left(\frac{1}{16}\right)\left(\frac{1}{2 \cdot DL}\right).$$

Note that the DL term is the divide-by-N value, and that the 1/16 term is the over-sampling rate required for the UART receiver.

The available baud rates in this scheme are all integer submultiples of the CPU clock, which limits the ability to obtain a desired baud rate within the required error value. This is particularly true when using a lower-frequency CPU clock and higher baud rates. By rearranging the formula, we can express the divider value as a function of the baud rate desired, obtaining

$$DL = \left(\frac{41.78\text{MHz}}{2^{CD}}\right)\left(\frac{1}{16}\right)\left(\frac{1}{2 \cdot \text{baud rate}}\right).$$

For example, with a 41.78MHz CPU clock (CD=0), it is possible to obtain 38.4kbaud exactly with DL=34. However, if the CPU is operating with CD=3 (CPU clock is ~5MHz), the required DL value is then 4.25. Since only integer values are possible, the nearest obtainable baud rates are 40.8kbaud (DL=4, +6.25% error) and 32.64kbaud (DL=5, -15.0% error).

1.3.2 Baud Rate using the Fractional Divider

The fractional divider mode is derived in a similar fashion to the normal 16450 mode, but with an added divider stage. This additional divider stage is responsible for the introduction of a fractional term into the baud rate equation. The baud rate is determined by the formula

$$\text{baud rate} = \left(\frac{41.78\text{MHz}}{2^{\text{CD}}}\right)\left(\frac{1}{16}\right)\left(\frac{1}{2\cdot\text{DL}}\right)\left(\frac{1}{M+\frac{N}{2048}}\right).$$

The fractional mode is enabled by the FBEN bit in the COMDIV2 register. The M and N terms are coded into bit fields in the COMDIV2 register as well. The M term is constrained to the range 1-4, while N can range from 0-2047.
By rearranging the formula, we can express the divide-by-N and fractional divider values as a function of the baud rate desired, obtaining

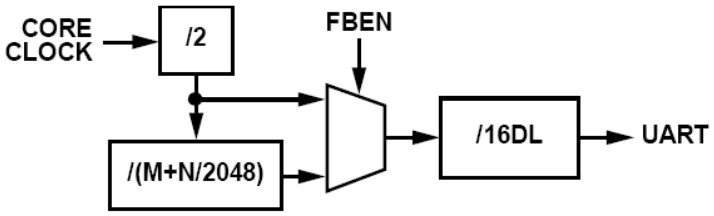
$$\text{DL}\left(\text{M}+\frac{\text{N}}{2048}\right) = \left(\frac{41.78\text{MHz}}{2^{\text{CD}}}\right)\left(\frac{1}{16}\right)\left(\frac{1}{2\cdot\text{baud rate}}\right).$$

With the additional precision of the fractional divider, we now find that even if the CPU is operating with CD=3 (CPU clock is ~5MHz), 38.4kbaud is exactly obtainable by setting DL=4, M=1 and N=128.

At first glance, implementing the N/2048 term of fractional divider may seem to be impossible without a clock rate 2048 times the actual CPU clock. (The M term could be implemented as a simple divide-by-M counter.) However, the N/2048 term can be implemented *on average* by dividing by M+1 for N baud clocks and then dividing by M for 2048 – N baud clocks. This results in an average division by M+(N/2048), the desired result.

It is important to note that while the baud clock over the 2048 cycles will have the correct *average frequency*, each individual baud clock cycle may not have the exact period that corresponds to the baud rate. In the extreme case where N = 1024, if the implementation operated as described above, the frequency would be too high for 1024 successive baud clocks and then too low for 1024 successive baud clocks. This could easily result in excessive frequency error and lost communications. To minimize the frequency error, we need to evenly distribute the *divide-by-M* and *divide-by-M+1* cycles across the 2048 baud clocks. So, for N =1024, the output should alternate between the *divide-by-M* and *divide-by-M+1* cycles. To evenly distribute the *divide-by-M* and *divide-by-M+1* cycles, fractional dividers may use a hardware implementation of the Bresenham algorithm or other methods.

To help reduce the baud period error, note that Analog Devices placed the fractional divider in the baud clock generation circuit at the highest possible frequency (see diagram below). In this way, the frequency variation minimized by virtue of the fact that the 2048 periods occur at 41.78MHz / 2, not over 2048 baud clocks.



1.4 UART Interrupts

The ADuC7026 UART generates a single interrupt request. The specific conditions that will generate the interrupt are controlled by the COMIEN0 register. The specific interrupt sources controlled by COMIEN0 are;

- Bit 3 – certain changes in the status of the modem control signals, as indicated by the least significant 4 bits of COMSTA1.
- Bit 2 – the availability of receive data, or the detection of parity, framing or overrun errors by the receiver.
- Bit 1 – the interrupt will be asserted whenever the transmit buffer is empty.
- Bit 0 – the interrupt will be asserted whenever there is receive data available.

The most basic operation of the UART only requires that interrupts be generated when the receive data register is full or the transmit data register is empty. In general, the transmit and receive data will be stored in queues in memory. The UART interrupt handler then is only responsible for getting incoming data from the UART and putting it in the receive data queue, and taking data from the transmit data queue and transmitting it. It is important to note that there may well be more than one reason that a UART interrupt occurs. When the UART interrupt is serviced, all possible causes of the interrupt must be determined by checking the COMSTA0 register and then handling the appropriate events.

If receive data ready is indicated (COMSTA0.DR = 1), the data is simply read from the COMRX register. Reading the COMRX register automatically clears the receive data ready interrupt request.

If a transmitter interrupt is indicated (COMSTA0.TEMT = 1), that means the COMTX register is empty (any data put in it has been moved to the shifter for transmission). If there is data waiting in the transmit queue, then the next byte of data can simply be written to COMTX, which clears the TEMT bit and hence the interrupt request. (Once data is written to COMTX, it is now known that there will be another transmit interrupt generated when that data is moved to the transmit shifter.) If there is no data waiting in the transmit queue, then the situation is different. The interrupt cannot simply be ignored, because the TEMT bit will still be set and the interrupt would still be asserted when the interrupt handler returned. This would cause the interrupt handler to be invoked again immediately ad infinitum. If there is no data to send, then the only option is to disable the transmit buffer empty interrupt. This solves the interrupt problem. It also means that the interrupt handler will no longer transmit data that is placed in the queue by some other part of the program. So, we will need to consider this whenever we put data in the transmit queue to be sure that the interrupt handler will send the

data. The direct solution in this case is to simply enable the UART transmit interrupt every time we put data in the transmit queue.

Note: The situation described above is typical for microprocessors that use level-sensitive interrupts, like the ARM7TDMI. In microprocessors that use edge-sensitive interrupts, the transmit interrupt handling is different. In a system with edge-sensitive interrupts, the transmit interrupt would still occur when the transmitter became empty. If there is no data to send, then we can simply do nothing. This works because another transmitter interrupt will not be generated until the transmitter has data loaded into it and then becomes empty again. So, when putting data in the transmit queue, the program needs to determine if another transmitter interrupt will occur. If an interrupt will occur, then the queue will be emptied by the interrupt handler. If another transmitter interrupt will not occur, then the first byte of data in the queue must be written directly into the transmit data register. This process is commonly referred to as “priming the pump”. It ensures that another transmitter interrupt will occur, so the remaining data in the queue will be transmitted by the interrupt handler. To determine if another transmitter interrupt will occur, a common technique is to use a semaphore that is updated by the interrupt handler when it responds to a transmitter interrupt. If the interrupt handler writes data to the transmit data register, then it knows another interrupt will occur, and so it sets the semaphore to 1. If there is no data to send, then it knows that there will not be another transmit interrupt, so it sets the semaphore to 0. When placing data in the transmit queue, the program disables interrupts, checks the semaphore to see if it needs to “prime the pump”, loads the queue, and then re-enables interrupts.

2 UART Configuration

2.1 Serial Port Mux

The serial port multiplexer controls the pin assignment for the serial peripherals, including the UART, and the programmable logic array. There are 10 GPIO pins (SPM0-SPM9) that can be assigned UART functions. The full UART signal complement including modem control signals can be assigned to SPM0-SPM7 (P1.0-P1.7), however this comes at the expense of not being able to use the other serial peripherals (SPI, I^2C). In basic UART usage, only the serial data input (SIN) and serial data output (SOUT) signals are necessary. The serial port multiplexer permits the SIN signal to be assigned to either P1.0 or P0.7. The SOUT signal can be assigned to either P1.1 or P2.0.

2.2 UART Registers

Most of the ADuC7026 UART registers are byte registers. Several registers occupy the the first two MMR addresses, with the register that is currently accessible determined by the setting of a bit in another register. This seemingly unusual arrangement is an artifact due to the basic design of the UART being based on the older 16450 discrete UART device. The 16450 UART used this tactic to reduce the number of address lines it required. The UART registers are listed below.

Address	Name	Access	Description
0xFFFF0700	COMMTX COMMRX	W-byte R-byte	Data written to this address will be transmitted. Incoming serial data is read from this register. (Accessible when COMCON0.DLAB = 0)
0xFFFF0700	COMDIV0	R/W-byte	Least significant byte of baud rate divisor. (Accessible when COMCON0.DLAB = 1)
0xFFFF0704	COMIEN0	R/W-byte	Interrupt enable register, determines which conditions will cause a UART interrupt. (Accessible when COMCON0.DLAB = 0)
0xFFFF0704	COMDIV1	R/W-byte	Most significant byte of baud rate divisor. (Accessible when COMCON0.DLAB = 1)
0xFFFF0708	COMIID0	R-byte	Interrupt identification register
0xFFFF070C	COMCON0	R/W-byte	Line control register
0xFFFF0710	COMCON1	R/W-byte	Modem control register
0xFFFF0714	COMSTA0	R-byte	UART status register
0xFFFF0718	COMSTA1	R-byte	Modem status register
0xFFFF071C	COMSCR	R/W-byte	Byte scratchpad register. Determines transmission type in network addressable UART mode
0xFFFF0720	COMIEN1	R/W-byte	Used only in network addressable UART mode
0xFFFF0724	COMIID1	R-byte	Used only in network addressable UART mode
0xFFFF0728	COMADR	R/W-byte	Used only in network addressable UART mode
0xFFFF072C	COMDIV2	R/W-halfword	Fractional baud rate divisor

2.3 Related Registers

In addition to the UART registers themselves, configuring the UART for operation involves a number of related ADuC7026 memory-mapped registers. The serial port multiplexer signal-to-pin assignments are made by setting the relevant GPIO control registers (GPxCON). If the UART interrupts are used, the UART interrupt request must be routed to either IRQ or FIQ mode, as set by the IRQEN or FIQEN registers. The appropriate global interrupt bit (CPSR.I or CPSR.F) must be cleared to unmask the required interrupt mode. If the transmitter interrupt must be disabled in the interrupt handler, it is most easily accomplished by writing to the IRQCLR or FIQCLR register as appropriate.

The clock frequency used by the baud rate generator is controlled by the clock source selected (PLLCON) and the clock divider being used (POWCON).

2.4 Configuration

The UART configuration involves the proper initialization of the UART registers and the related registers described above. A basic configuration scheme for UART operation that does not use the modem control signals is shown below.

1. Configure the GPIO pins that will be used by the serial port by writing to the appropriate GPxCON registers
2. Configure the UART
    a. Set COMCON0.DLAB = 1 for baud rate divisor access
    b. Write the desired baud rate divisor to COMDIV1, COMDIV0
    c. If the fractional baud rate divider will be used, write the COMDIV2 with the FBEN bit set and the desired M and N values
    d. Set up COMCON0
        i. DLAB = 0 for normal operation
        ii. SP, EPS, PEN for desired parity operation
        iii. STOP for the required number of stop bits
        iv. WLS for the desired number of data bits in the serial frame
    e. Enable the desired interrupt sources by writing COMIEN0
        i. ETBI to enable transmitter empty interrupt
        ii. ERBFI to enable receive data ready interrupt
    f. The remaining UART registers can stay at their reset values
3. Properly initialize any queues or other data structures that will be used by the UART interrupt handler
4. Select the interrupt mode by setting the UART bit in IRQEN or FIQEN.
5. Enable global interrupts by clearing the I bit or F bit in CPSR.