Ugly UART: The NXP 16550-like oddities

Jesús Arias

February 26, 2023

1 Introduction

NXP, formerly Philips Semiconductor, has a wide range of ARM controllers, and many of these include one or more UARTs that are an almost verbatim copy of the classic UART found in PCs decades ago: The National Semiconductor's 16550. At least the LPC21xx, LPC24xx, and LPC11xx microcontroller families include that kind of UART.

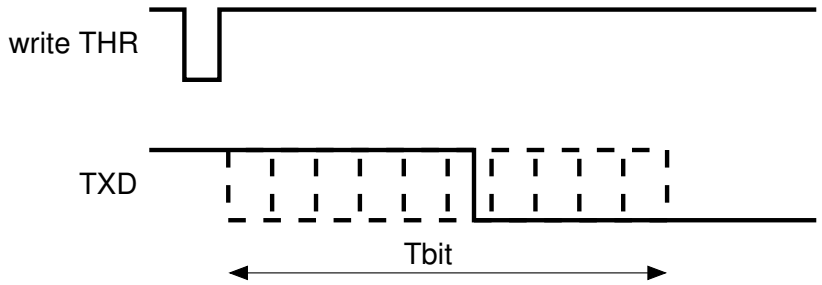
This 8-bit peripheral is, perhaps, out of place in an 32-bit microcontroller, but designers probably resorted to the motto "Better the known, even if bad...". But beware programmers: The datasheet omits to tell some weird behavior that somebody would want to name alternatively as "design bugs". Most of this knowledge was acquired the hard way: fixing communication failures, while some other curious facts were discovered serendipitously. At the end I also attached an old NMOS 8250 UART to the microcontroller in order to compare the flawed behavior of its internal UART with that of the original part. Here are the results for the NXP UART:

1. Parity errors not detected.
2. TEMP flag set half a bit time early.
3. Transmitter FIFO always active.
4. Variable time from write to THR and start bit when transmitter is idle.

Issues 1 and 2 weren't found in the 8250 UART, while point 4 was discovered in the old part before also being tested in the internal UART of the microcontroller. Lets comment each issue in more detail, starting with the least serious.

2 Variable TX delay

This was discovered in the old 8250 UART. After attaching that device to an LPC2103 microcontroller and while testing the required bitbanging code, I was displaying in an scope screen the write pulse along with the transmitter output, and the variable delay caught my attention immediately:



For me that was a kind of a surprise because I already have designed several UARTs for FPGA cores and I was always looking for ways to avoid this delay, just to find that commercial parts never cared about it. In my designs the baud rate counter gets reset when a write to THR is done and the transmitter is in an idle state. This synchronizes the transmitter clock with the write pulse and the falling edge of the start bit comes out immediately. Obviously, in the 8250 chip this isn't the case and the start bit just begins when the counter of the clock divider overflows.

I was curious about the integrated UARTs in the NXP microcontroller and, of course, I wrote a few test programs to confirm the identical behavior of the modern peripherals. In this case the write signal is internal and it can't be routed to an scope probe, but we can also use the integrated timers to measure the delay between register writes and falling edges in the transmitter pin.

So, old and new 8250-like UARTs have this variable delay. This is not a big problem and it results only in an almost negligible performance penalty, so we don't have to worry more about it.

3 TX FIFO always enabled

Unfortunately I don't have a legacy 16550 UART around, just its older 8250 cousin, and that part lacks the FIFOs, so I can't compare this unexpected behavior in the NXP part against the primitive one.

Parts with FIFOs have a new FIFO Control Register, FCR, where resides an enable bit (bit #0). The documentation states that a value of zero disables both the receiver and the transmitter FIFOs, turning the 16550 into a simple 8250. But, at least in the NXP implementation, you can still write up to 16 characters to THR and all of them are transmitted. This means that the TX FIFO is still in operation even when it is supposedly disabled in FCR.

So, as a conclusion, we can't trust datasheets very much. Therefore I also tested the receiver FIFO, and in this case it is effectively disabled when the FIFO enable bit in FCR is cleared.

4 Parity Error undetected

This is a serious bug, only found in the NXP UARTs, not in the 8250. This was discovered when, as a teaching exercise, the following data packet was received:

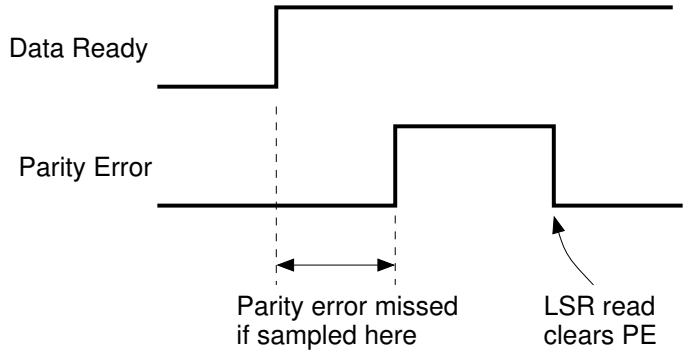
Header P	Data #0 P	...	Data #n p	Checksum P
length O	data E	...	data E	checksum E

In that protocol the first byte of the packet includes an Odd parity bit while the rest of the bytes have an Even parity bit. Thus, in the receiver a parity error signals the beginning of a data packet. A receiver code

using interrupts worked fine, but a simpler polling code was missing some packets. This also happened when "sticky" parity was used instead of the regular one. The polling code was more or less:

```
while(1){
   while (((lsr=U1LSR)&1)==0); // Data ready in bit #0
   rxd=U1RBR;
   if (lsr&4) {len=rxd; break;}
}
```

Where a copy of LSR, including the parity error bit, is stored in the local variable "lsr" before the data is read into "rxd". Then, if a parity error is found (bit #2 of LSR) the rest of the packet is received (code not shown), and, otherwise, the current character is discarded. Using this code a packet is lost from time to time, but when ported to the old 8250 not a single packet was lost.



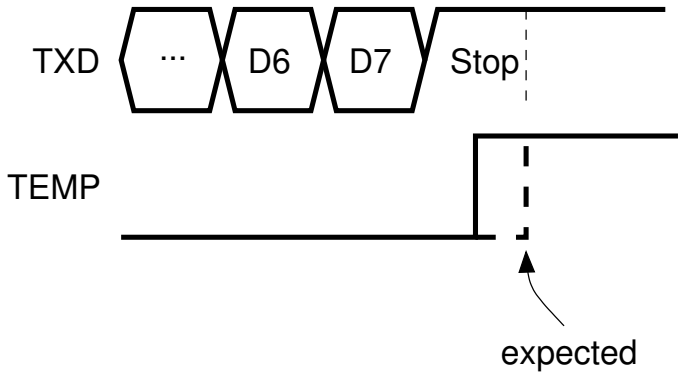
After some thinking I figured out what the problem was: When a character is received the data ready flag (bit #0 of LSR) is set **before** the parity error flag. This delay is an small one (after some statistics I found a rough estimate around 20ns), but, if the register LSR is read in the time interval after DR is one but before PE is also one, the parity error gets undetected.

Thus, a delay after detecting DR active would be required before reading LSR again. But, reading LSR also has the effect of clearing all the error flags, so, this isn't a solution. The only possible workaround is to activate the receiver interrupt with a FIFO threshold of only one byte, and to do the polling for a data received in the IIR register instead of LSR. Then read the LSR register, and finally the received data. Using this workaround no more packets were lost.

The estimated delay looks much like a single PCLK cycle and this would be a serious design flaw in the NXP UART. I found this problem in all NXP microcontrollers using this UART (tested chips were: LP2103, LPC2106, LPC2478, LPC2468, and LPC1114)

Also, I want to blame the designers of the 8250 at National Semiconductor for their stupid design choices: Error flags should be reset when the data register is read, not when the status register is read. And BTW, THRE should be set as long as the transmitter FIFO is not full, not when it is empty.

5 Transmitter Empty flag set early



The last issue found in NXP UARTs is the early activation of the TEMP flag. This was also discovered in an scope screen where the last stop bit of an RS485 transmission was found to be half of its supposed time. The code was like:

```
while((U1LSR&(1<<6))==0); // wait until TEMP
IO0CLR=(1<<7);    // switch to RX
```

Of course, the RS485 transceiver was switched to reception before it should be because the TEMP flag in the LSR register was also set before the end of the last stop bit. TEMP is actually set in the middle of the last stop bit time.

This problem is easy to workaround, a simple delay of half a bit duration before the switching to RX is enough, but it is a nuisance anyway. And it is an NXP issue because the old 8250 sets TEMP at its proper time.

But this last test also discovered another TEMP bug, this case in the old 8250. According to datasheets TEMP is set to one when there are no pending data to transmit in neither the Transmitter Holding register nor the shift register, but, a polling for TEMP found it briefly set to one between characters. In this case the workaround was to wait until both THRE and TEMP are set to one. It seems that the old 8250 lacks a simple AND gate.

So, TEMP seems to be a continuous troublemaker from the NMOS era to now.