Common Pitfalls in UART Communication

Hardware and Software Solutions

By Jason Silic - October 2023

Embedded systems utilize a variety of low-level communication systems and protocols. One of the simplest is known as Universal Asynchronous Receiver / Transmitter (UART). The more important term here is "Asynchronous," which means either device in a point-to-point network can begin transmitting at any time. An advantage of UART over I2C or SPI is the simplicity and minimal hardware requirements. Inexpensive microcontrollers with multiple UART interfaces can be readily obtained. This paper will examine some of the common issues encountered when developing an embedded system utilizing UART communication. Practical solutions will be presented for both hardware and software issues.

Generally, there are two wires in this protocol, a transmit and receive line. However, with the use of a transmit/receive multiplexer circuit, even this requirement can be reduced to a single wire. In this paper we will look at simplified circuits with simulation of only one wire.

UART Basics

A data frame in standard UART communication consists of data (the payload), start / stop bits, and an optional parity bit. The start and stop bits help prevent clock drift between the sender and receiver which could otherwise occur due to the lack of a synchronizing clock in UART. The parity bit can be used to help ensure the integrity of each transmitted frame. For this paper we also define a packet as multiple frames combined into a single unit (often with CRC bytes for additional integrity checks).

In *Figure 1* below we are transmitting a single byte of data, 0x25. Note that data is transmitted starting at the least-significant bit. The type of parity used here is "Even" parity, which ensures that an even number of '1s' are present in the data + parity section of the frame. Because 0x25 has three '1s', the parity bit must be high to give an even total of 4 '1s'.

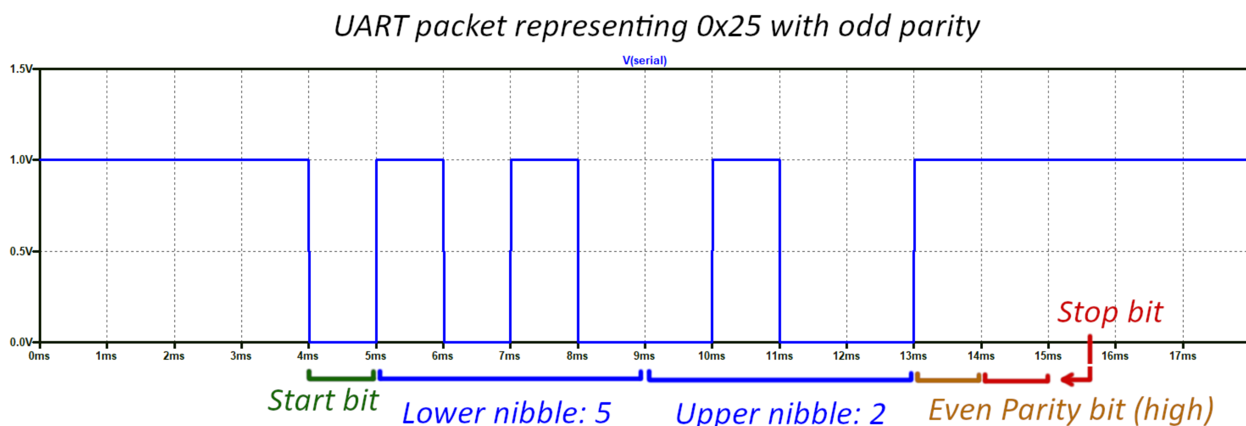

Figure 1 - A UART frame

An important fact to note here is that eleven bits are used to transmit this frame containing 8 data bits. This 27% overhead (3 out of 11) is one of the disadvantages of UART.

Software Pitfall: Premature TX Complete Interrupt

In some UART systems there is a transmit/receive buffer that can only work in one direction at a time. A common example would be UART over RS-485, a differential signaling standard. Hardware is required to interface between the two UART transmit/receive lines and the differential, bi-directional RS-485 channel.

This pitfall appears when utilizing hardware that provides the ability to write multiple bytes to the serial hardware automatically. A pointer and byte count are typically provided, and the hardware will continue to transmit until it has completed the task. This allows the main CPU to process other tasks while waiting for the “TX Complete” interrupt.

Note that the low-level serial hardware is very similar to a shift register. Once it has received a byte it will simply send the data out based on the configured baud rate. On common hardware (Such as Microchip’s SAM series of microcontrollers) the TX complete interrupt is generated as soon as the final buffered byte is placed into the serial transmit hardware. If our software disables the transmit buffer in the Interrupt Service Routine (ISR), we will have the situation seen below in *Figure 2*. The final byte in the transmit sequence will be truncated or eliminated.

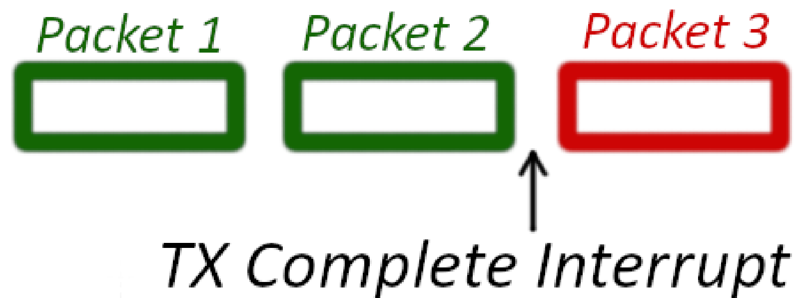

Figure 2 - TX complete ISR will execute before all data has been sent

If this issue applies to your system, a delay will need to be implemented before the buffer is set to receive mode when listening for a response. For high baud rates and precise timing this may unfortunately require a hardware timer. In many cases such precision is not needed; efficient low-resolution timers will be adequate. Fortunately, this issue is easy to catch during testing and will only add an extra state to the UART driver’s state machine (or equivalent) to track this waiting period.

Hardware Pitfall: Asymmetric Transmission Eroding Signal Integrity

Another issue that can really sneak up is degradation of the communication link quality as timing margin erodes. There are many factors that could cause issues here. In this section we present an example of asymmetrical transitions through optoisolators. If the delay for a rising edge is different from the delay for a falling edge then the sampling point for the receiver circuit begins to drift from the ideal location. This can begin to erode timing margin and lead to bit errors when combined with baud rate differences, extreme temperature, or other phenomena.

To illustrate how this issue might look in the real world, consider the circuit below. Two optoisolators are connected in series. The transmission line between them has some significant capacitance (C1 and C2).

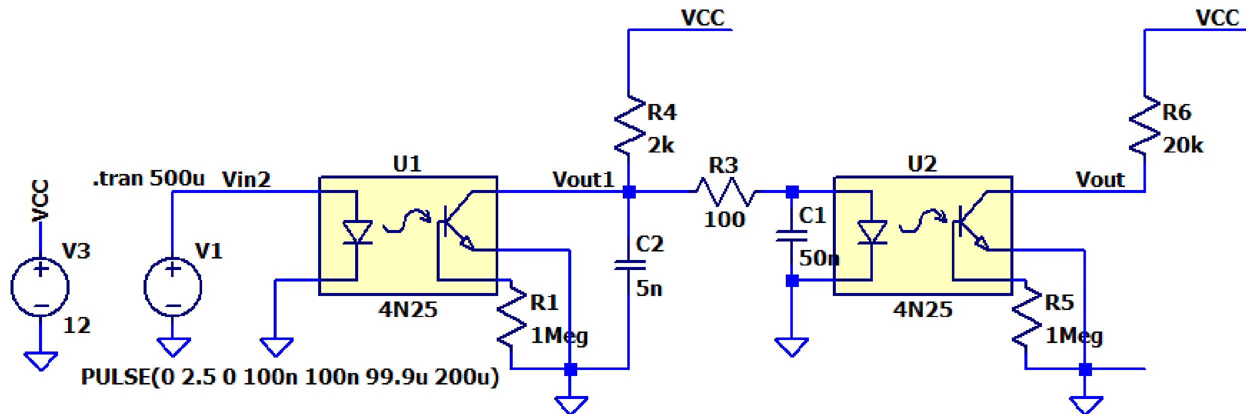

Figure 3 - Cascaded Optoisolators

A square wave is supplied to the circuit above and the resulting output waveform is shown in *Figure 4*. The input has a period of 200µs. If both low and high signals are considered a single bit then this is a 10k baud rate (each bit takes 100µs). The time from the midpoint of the falling edge to a rising edge would be 100µs with symmetrical transition times. In this simulation we find the actual time is: 244.4µs – 119.4µs = 125µs. If the receiver detects the start bit at the falling edge, it will sample subsequent bits (at the midpoint) with an offset of 50µs from the implied start of the bit (+50µs, +150µs, etc.). In this example we expected the edge of the second bit would be at a time of +100µs for a 50µs margin from the sample instant to the edge of the bit. Because our edge is at +125µs, we have lost 50% of our timing margin. At a baud rate of 20k we would have zero margin.

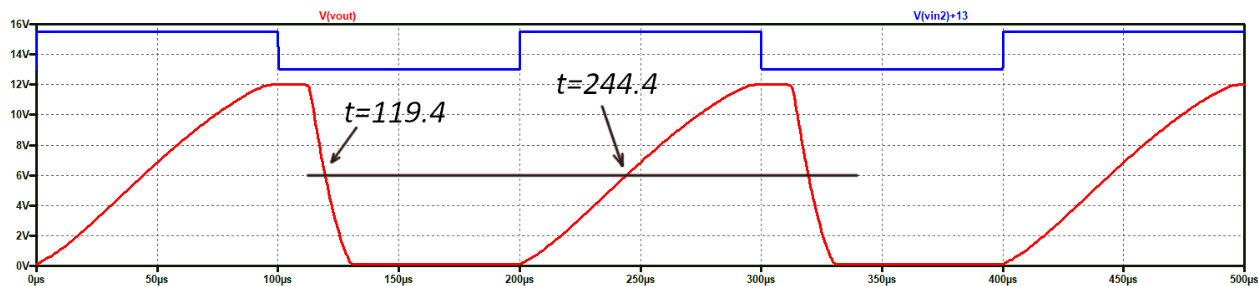

Figure 4 - Input and Output Waveforms

As noted above, baud rate differences between transmitter and receiver can lead to additional margin erosion toward the end of a frame. As this is a hardware issue there is no easy software fix. Two possible solutions are reducing the baud rate and avoiding the problem through hardware validation.

Reducing the baud rate gives the optoisolators more time to make signal transitions and the amount of skew becomes proportionally smaller as the bit time increases. For example, the 25µs delay noted above is 50% of timing margin at 10k baud. However, at 1k baud this is only 5% of the timing margin. Total margin is half of the bit width:

$$Margin = \frac{T}{2} = \frac{\frac{1s}{1000\ baud}}{2} = 500\mu s \rightarrow 5\%\ of\ 500\mu s\ is\ 25\mu s$$

There are two drawbacks to this approach. First, both transmitter and receiver must implement the reduced baud rate. If you are communicating with another piece of hardware not under your control this may not be practical. A second obvious consequence of a reduced baud rate is slower communication and less bandwidth. Depending on the application this could be perfectly acceptable or completely unworkable. These tradeoffs are one of the most enjoyable parts of engineering!

The best way of avoiding this issue is to utilize a correct design with crip edges. Hardware validation testing should include tests to ensure signal integrity throughout the operating envelope. In many projects this will unfortunately not be possible. Sometimes the hardware design is frozen before these issues are discovered or unforeseen changes exacerbate this problem after initial testing is complete. In that case the above solution will be helpful. However, being aware of this issue will hopefully allow the design engineer to ensure a working design by catching any potential issues early.

Verification For UART Serial Links

Design Verification is the critical step that can safeguard your system from these pitfalls. Quantify your timing margin at a range of temperatures throughout the design envelope. If the software driver automatically retries packets, track how often this occurs! If you neglect a serial interface, some late-breaking issues will probably become manifest for your enjoyment as the product moves toward release.

One final thought: build redundancy into your serial communication logic. Even if packets begin to drop due to the issues discussed here, packet retries can go a long way toward allowing the higher layers of the system to work as designed. This can be overdone and certainly introduces complexity to the software but will yield a more robust serial link. It is even possible for a large proportion of packets to be lost (> 10%) and still have functional communication. In short, work to ensure flawless communication in the lab, but be accepting of noise and failures in the field.