Name: _____dDB key_____

Dordt University Engineering Department

EGR 304, Microprocessor Interfacing— Test 2, April 8, 2020.
Open book, open notes, open computer, no smartphone, no social media except Zoom to Prof dDB.

1. An RS-232 link operates at 56 k-baud. (56000 baud exactly) The link is specified to operate with an 8N1 format. (8 data bits/word, No parity, 1 stop bit). The link uses the XON/XOFF software protocol and has a three-conductor cable. How long will it take to transfer a 1 MB file (1,048,576 bytes exactly) over this connection? (Hint: do not confuse bits and bytes.) (10 points)

   8 bits of data + start + stop = 10 bits per word sent.
   (1,048,576 bytes)(10 bits/byte)/(56000 bits/sec) = 187.25 sec = 3 min, 7.25 sec

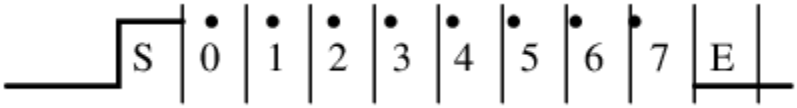
   It takes 187.25 seconds.

2. For a certain RS-232 connection operating at the non-standard rate of 10000 baud the receiver clock happens to be running a little too fast. (The transmitter runs at exactly 10000 baud. The receiver clock runs a little faster than 10000 baud). However, it still works reliably. Tolerance of inaccurate clock speeds is one of the advantages of RS-232. The signaling format used is 8N1. Assume that the receiver uses the rising edge of the start bit as the timing reference and then waits 1.5 bit intervals before sampling a data bit. Thus it waits for the start bit to pass and for half of a bit-interval to pass, then samples a bit. After that it waits one bit interval to sample each successive bit. Thus, a receiver running at exactly the transmitter's speed would sample each bit in the exact middle of each bit interval. However in this case, the intervals are timed by the slightly fast receiver clock so the sample times each get a little earlier in each successive bit interval as time passes. If the receiver is not running too fast, this will not matter since the bit is present for the entire bit interval and does not need to be sampled exactly in the middle of the interval. (30 points)

   a.) Assuming instantaneous transitions of the signal at the receiver (assume an infinite slope when signals change state) **what is the maximum receiver rate in baud** that could work without error. That is, what is the maximum receiver clock rate that will not cause the sampling of the last bit of each word to be made before the last bit arrives at the receiver.

   b.) **What percentage of error** does this represent from the nominal receiver clock rate?

Solution to part (a)



In the illustration above, the dots show the sample times as measured out by the receiver's fast clock.
With a fast clock the sample times will drift earlier and earlier into the bit-cell, as shown above.
Each transmitted bit interval lasts (1/10000) s or 100 μs
From the rising edge of the start bit to the instant at which bit 7 (the MSB) starts, as sent by the transmitter, takes exactly 8 bit cell intervals, 800 μs.
At the receiver, the time interval from the rising edge of the start bit to the MIDDLE of the cell for bit 7 where the sample *should* be taken takes exactly 8.5 bit cells which *should* be 850 μs. However, and actual time interval of just over 800 μs will still land the sample in the correct bit cell.
This reveals that the true clock speed at the receiver could be as fast as 10000(850/800) = 10625 baud.

The maximum baud rate that could possibly work at the receiver is 10625 baud.

Solution to part (b)

[(10625/10000) – 1] x 100 = 6.25%

The receiver's clock could be as much as 6.25% fast and it might still work.

3. A microcontroller system employs three interrupt sources labeled A, B, and C. The interrupt service routines run with interrupts disabled. Interrupt A is the highest priority, C the lowest.

   Interrupt A interrupts at various random times, but after it has interrupted it is assured that it will not interrupt again for at least 1000 μs. It always takes 15 μs to service it.

   Interrupt B is a tick-clock interrupt. It is periodic with a period of 500 μs. Depending on what is scheduled for a tick, it might take as little as 1 μs or as much as 300 μs to service.

   Interrupt C is a rare interrupt, happening no more often than once an hour. It always takes exactly 600 μs to service this interrupt.

   The longest instruction takes 10 μs to execute. There are also critical regions in the main code that take between 15 μs and 50 μs to execute.

   Will these interrupts operate reliably? If operation is reliable, prove it. If it is unreliable, explain why or give an example of unreliable operation.

(30 points)

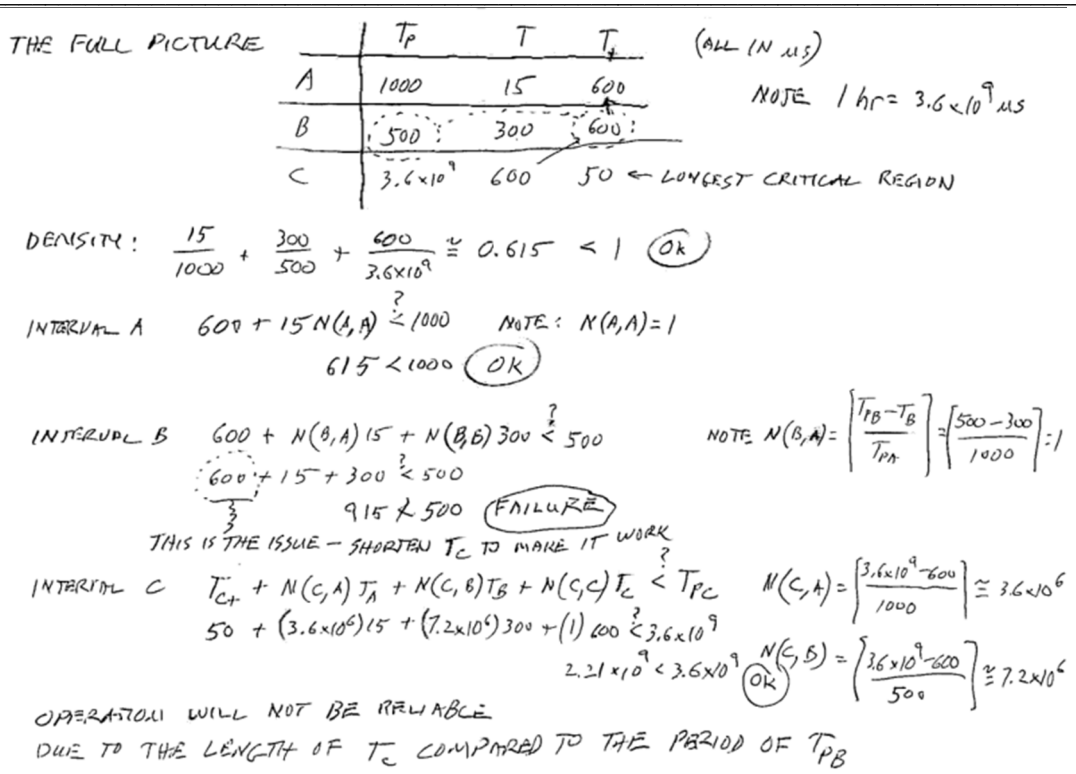
First I will identify the maximum run times and periods for each interrupt and get them organized in a table:

ISR A: t_{PA} = 1000 μs t_A = 15 μs
ISR B: t_{PB} = 500 μs t_B = 300 μs
ISR C: t_{PC} = 1 hr t_C = 600 μs

I also note that the longest operation is the critical region, at up to 50 μs

At this point I notice that $t_C > t_{PB}$
This means that each time ISR C get called it executes for longer than the period of ISR B.
Thus at least one request for service from ISR B will be ignored during each execution of ISR C. That is a failure.

Interrupts will not run reliably because $t_C > t_{PB}$



4 A pulse-width needs to be measured. The pulse always will be between 10 and 100 ms long and this is guaranteed. If a measurement of the pulse width shows less than 10 ms or more than 100 ms you can be assured that the measurement was done incorrectly. When the signal transitions from logic-1 to logic zero, you can be assured that it will remain at logic-0 for at least 20 ms. When the signal transitions from logic-0 to logic-1 it will remain at logic-1 for between 10 and 100 ms and then transition back to logic-0.

The microcontroller in use has a counter-timer system and a 0.1 ms tick-clock interrupt source. The counter-timer system is setup to count tick-clock interrupts from the moment when the microcontroller powers up and the interrupts are enabled and forever after. The counter is 16-bits wide and will roll over back to zero and then keep counting when it overflows. (. . .65533_{10}, 65534_{10}, 65535_{10}, 00000_{10}, 00001_{10}, 00002_{10},. . .)

The pulse to be measured is applied to an input pin that is set to interrupt on each edge, rising or falling. (This input pin interrupt is a second interrupt source in addition to the tick-clock interrupt.) This same hardware interrupt is set up to immediately (via hardware) *capture* the tick-clock count when the input pin (pulse) changes logic state and save that value in a buffer register that can be accessed by the interrupt service routine (ISR).

The ISR associated with the input pin (pulse) interrupt, called `edgeHappened`, reads the tick-clock counter value that was stored in the buffer at the time of the pulse edge, and also reads the input pin (about a fraction of a millisecond after the interrupt happened) to discover if the pulse input pin is now at logic-1 or logic-0. If the pulse input is at logic-1 it stores the buffer register value into global variable `pulseStartTime` but if the pulse input is at logic-0 it stores the buffer register value into global variable `pulseEndTime`. Also, if the pulse input is at logic-0 the ISR calculates the pulse width, in seconds, and places that in global variable `pulseWidth`. The global variable `pulseWidth` thus always contains the actual pulse width in seconds of the most recently completed calculation of the pulse width. Global variable `pulseWidth` is stored in floating-point format, whereas `pulseStartTime` and `pulseEndTime` are stored as 16-bit unsigned integers.

a.) Suppose the input pin ISR has just started updating `pulseWidth`. The global variable `pulseStartTime` contains 65413_{10} and global variable `pulseEndTime` contains 00375_{10}.
What should the global variable `pulseWidth` receive if the ISR is working correctly?
This is equivalent to asking: "What was the width, in seconds, of the most recent pulse?"
(15 points)

`pulseEndTime` – `pulseStartTime` = 00375 – 65413 < 0 ← illogical. A pulse can't have negative width.
The tick-clock timer must have rolled over. It is a 16-bit timer, so add 2^{16} to the end time and re-do the calculation.
(2^{16} + `pulseEndTime`) – `pulseStartTime` = (65536 + 00375) – 65413 = 498
Each count of the tick clock represents 0.1 ms. Thus the pulse width is 498(0.1 ms) = 49.8 ms = 0.0498 s

`pulseWidth` should receive the value of 0.0498 or 4.98×10^{-2}

b.) What is the uncertainty associated with the value of `pulseWidth`? (5 points)

The uncertainty is ± 1 tick, which is ±0.0001 s
(For rationale, see diagram on slide #4 from Friday, 2/28.)

c.) Given the specifications above (pulse lasts between 10 and 100 ms, pulse is at logic-0 at least 20 ms), and assuming that the pin ISR takes at most 2 ms to run, what is the minimum period of the pin interrupt, t_{pi}, and the maximum ISR execution time, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

t_i = maximum ISR execution time is given as 2 ms.

$$t_i = 2 \text{ ms}$$

t_{pi} = minimum period, in this case minimum pulse width plus minimum idle time = 10 ms + 20 ms = 30 ms

$$t_{pi} = 30 \text{ ms}$$