Name: Dan Kelly

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.

26/100 = F

33/100 = D+

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)

$\frac{1}{56000} = 17.86\ \mu s$ per cycle → 8 388 608 bits

$17.86\ \mu s \times \frac{8\,388\,608}{9} = 16646.64\ \mu s$

6

Start and stop bits neglected.

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)

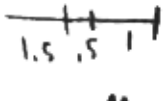
a.) Assuming instantaneous transitions of the signal at the receiver (assume an infinite slope when signals change state) **what is the maximum receiver rate in baud** that could work without error. That is, what is the maximum receiver clock rate that will not cause the sampling of the last bit of each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

$\frac{1}{10,000} = 100\ \mu s$ 10,000

5

8



Max = 9600 baud < 10000 ∴ slow X

b.) $\frac{100\ \mu s}{9600}$ = 1.04 % X

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)

This table is a good start

Int-pt A	15 µs	1000 µs	10 µs to execute
Int B	500 µs	1 µs – 300 µs	15 µs – 50 µs
Int C	600 µs		

$\frac{15}{1000} + \frac{500}{300} +$

7

$T_c = 600\,\mu s$ $\quad \frac{15\,\mu s}{1000\,\mu s} = .015 + \frac{300}{500} = .6 + \frac{600}{3600} + .1666 = .781666 < 1.0$

They are reliable

$T_A = 15\,\mu s$ $\quad T_B = 1\,ms - 300\,ms$ $\quad T_C = 600\,ms$

$T_{PA} = 1000\,ms$ $\quad T_{PB} = 500\,ms$ $\quad T_P = ?$

Not enough analysis was done t identify the faulure.
What analysis has been done is so sketchy that I cannot follow it.

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)

1

.2 seconds = 200 ms > 100 ms

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

190 ms − 150 ms = 40 ms = .04 seconds

1

$\frac{.04}{.2}$ = 20%

A leading decimal is unprofessional. (Except in baseball!) This should be "0.2 s."

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 allowable interrupt hold-off interval, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

$\frac{1}{f} = t_{pi}$ $\frac{1}{.002}$ = 500 ms

$\frac{10}{.002}$ = 5000 ms

6

10

There is a typo in part C that I did not recognized before the test.
Þveryone gets full credit (5 points) for the ti part.

Name: Patrick Munsey

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.

~~66/100 = B+~~ 75/100 = B+

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)

$\frac{1,048,576 \text{ bytes}}{7168 \text{ bytes/s}}$ 56 kband = 7168 bytes/s ? X

4

Start and stop bits ignored

= 2.44 minutes X

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?

a) Clock "resets" after each word with a catch at the start bit, so by bit 8, the clock cannot jump ahead by (1/20000)'s of a second. ✓

So, $\left(\frac{1}{20000}\right) \cdot \left(\frac{1}{7}\right) = \left(\frac{1}{140,000}\right)_s$ (total jump time after 7 bits; jump per bit)

$\left(\frac{1}{140,000}\right) + \left(\frac{1}{10,000}\right) = 9,333.\overline{33}$ baud

slowest failure rate

~~20~~ 25

rate must be $> 9,333.\overline{33}$ baud ≤ 10000

∴ Slow

b) % error $= \left(\frac{10000 + 9333.\overline{33}}{10,000}\right) \cdot 100$

% error = 6.66 % X

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)

Interrupt density: $\left(\frac{15}{1000}\right) + \left(\frac{300}{500}\right) + \left(\frac{600}{3.6 \times 10^9}\right) < 1$ ✓

Int. A: $50 + N(1,1)^{(15)} \cdot T_1 = 65 < 1000$ ms ✓

Int. B: $50 + N(2,2)^{(300)} T_2 + N(1,1)^{(30)} \cdot T_1 = 380 < 500$ ✓

Int. C: $50 + N(3,3)^{(600)} T_3 + N(3,2)^{(300)} T_2 + N(3,1)^{(15)} T_1 = 965 < 3.6 \times 10^9$ ✓

Will be reliable ✓

20

On the one hand this is sort of a simple blunder--only one number is wrong. On the other hand, the analysis is rather sketchy. There is not able of specifications for each ISR and the Ti+ times are not carefully analyzed. Poor technique.

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.

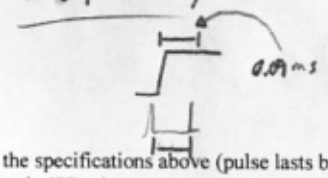
a.) Suppose the input pin ISR has just started updating `pulseWidth`. The global variable `pulseStartTime` contains 65413_{10} and global variable `pulseEndTime` contains 00375_{10}.
What should the global variable `pulseWidth` receive if the ISR is working correctly?
This is equivalent to asking: "What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

14 65535 − 65413 = 122 497 · (0.1 ms) = 49.7 ms ✗
122 + 375 = 497

You forgot to count the zero in . . .65535, 00000, 00001. . .

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

2 ±0.18 ms ✗, since each side could be off 0.09 ms on catching the edge.



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 allowable interrupt hold-off interval, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

6 t_{pi} must be at least 2.1 ms ✗ so the ISR doesn't start again before finishing up the last ISR

10 t_i must be less than 8 ms ✗ in order to catch every pulse (occurring 10 ms

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.

Name: Lucas Nelson

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.

~~82/100 = A-~~ 92/100 = A

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 → 1 byte

1,048,576 × 8 → 8388608

8388608 bits of data

1048,576 stop bits

1,049,576 start bits

Need a start bit, stop bit, then start again

+1 start at beginning

+1 stop at end

→ 10485762 bits to transfer = 187.246 seconds

56,000 bits per second

≈ 3 min 7 sec

10

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)

Running low on time

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?

a) Instantaneous change ↑ or ↓ (square wave) so it could sample immediately at start of that bit

typically

but instantaneous means it could sample

going to be > 10,000 baud

During a discussion between Lucas and Prof. dDB Lucas explained his method. Prof. decided he knew more than he was initially given credit for.

~~15~~ 25

I understand what it is asking. Need a correct answer from part A to get part B

≈ 11,000 baud X

b) 10% error

1,000 / 10,000 = .1

Part (b) would be correct if part (a) had been correct. Full credit here.

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)

Prepare for worst case scenario

Has critical regions

$T_A = 15\,ms$ $T_{AP} = 1000\,ms$ $T_{A+} = 600\,ms$

$T_B = 300\,ms$ $T_{BP} = 500\,ms$ $T_{B+} = 600\,ms$

$T_C = 600\,ms$ (max) $T_{CP} = 3.6\times10^9\,ms$ $T_{C+} = 50\,ms$ (critical region)

30

Interrupt density: $\frac{15}{1000} + \frac{300}{500} + \frac{600}{3.6\times10^9} = 0.615 < 1$ so it is good

$N(i,x) = \text{ceil}\{(T_{Pi} - T_i)/T_{Px}\}$

$N(A,A) = 1$ $N(B,A) = 1$ $N(C,A) = 3.6\times10^6$

$N(B,B) = 1$ $N(C,B) = 7.199999\times10^6$

$N(C,C) = 1$

$N(B,A) = \frac{T_{PB} - T_B}{T_{PA}} = 1$ $N(C,A) = \frac{T_{PC} - T_C}{T_{PA}} = 3.6\times10^6$ $N(C,B) = \frac{T_{PC} - T_C}{T_B} = 7.199999\times10^6$

For interrupt A:

$T_{A+} + N(A,A)T_A < T_{PA} \rightarrow 600_{ms} + (1)15 < 1000 : 615 < 1000$ ✓

For interrupt B:

$T_{B+} + N(B,B)T_B + N(B,A)T_A < T_{PB} \rightarrow 600_{ms} + (1)300 + (1)15 < 500 \rightarrow 915 < 500$ ✗

Unreliable because in the case of interrupt B the inequality is not met

I did this for worst case scenario. It may work fairly reliably if the longest amount of time for everything was not selected. I would be happy to answer any questions about my work.

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)

If hit overflow → 65413 65535 − 65413 = 122 + 376 to allow for 00000

15

498 increments tick is .1ms .1 × 498

49.8 ms → 0.0498 seconds

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

up to .1 ms either side ± 0.2ms

2

Not sure how often pulse happens → potentially logic-0 in timer of 20 ms but technically pulse going

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 interrupt, t_{pi}, and the maximum ISR execution time, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

2 ms period

10

100ms : 20ms

give it up to max

700/70 = 5

10ms + 20ms + 2ms

minimum period would be 32ms

Max ISR time 2ms

There is a typo in part C that I did not recognized before the test. Þveryone gets full credit (5 points) for the ti part.

Name: Zachary Sanford

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.

64/100 = B+ → 72/100 = B+

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)

X 10 bit cells per word

8 bits = 1 byte

+ 1 stop bit

9 bits total for each word 1 start bit

8

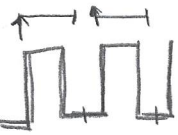
$\left(\frac{1{,}048{,}576 \text{ bytes}}{1 \text{ MB file}}\right)\left(\frac{9 \text{ bits}}{1 \text{ byte}}\right) = 9.437E6$

$\frac{\text{B}}{\text{Second}}$

$\frac{56\text{kHz}}{17.85\,\mu\text{s}}$

$\frac{9.437E6}{17.85\,\mu\text{s}} = 168.454 \text{ sec} = 2\text{min } 48 \text{ sec}$ close to 157 seconds = 2min 37 sec

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?

0 I did not get this in time

5

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)

30

#1 $T_{p1} = 1000\,\mu s$ $T_1 = 15\,\mu s$ $T_{1+} = 600\,\mu s$

#2 $T_{p2} = 500\,\mu s$ $T_2 = 300\,\mu s$ ← worst case scenario $T_{2+} = 600\,\mu s$

#3 $T_{p3} = 3600\,sec$ $T_3 = 600\,\mu s$ $T_{3+} = 10\,\mu s$

longest instruction takes 10 µs to execute

critical regions 15 µs to 50 µs to execute

Step 1 check density $\frac{15\,\mu s}{1000\,\mu s} + \frac{300\,\mu s}{500\,\mu s} + \frac{600\,\mu s}{3.6\,E9\,\mu s} = \frac{615\,\mu s}{1000\,\mu s}$ ok

Step 2 check max latency

$T_{1+} + N(1,1)T_1 < T_{p1} \rightarrow 600\,\mu s + 1(15) = 615 < 1000\,\mu s$ ok

$T_{2+} + N(2,2)T_2 + N(2,1)T_1 < T_{p2} \rightarrow 600\,\mu s + \quad < 500\,\mu s$ Not ok

This will not operate reliably. Because T_{2+} ($600\,\mu s$) is greater than T_{p2} ($500\,\mu s$).

There is not enough time between interupts to allow interup B to execute without being interupted by another interupt or it interferes with a critical region.

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.

10 - 100ms

10 - 100m

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)

122 > 497 → bit
375

14

497 bit changes within 0.1 ms tick-clock interrupt source checks
pulseWidth seconds
497 bits * 0.1 ms = 0.0497 seconds
49.7 ms ✗ 49.8

Forgot to count the zero in . . .65535, 00000, 00001. . .

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

0.0497 sec 49.7 ms 0.1 ms Period 497 bits

5

$\Delta t_P = 1/f_{ck}$ ✓ 49.7 ms 10,000 Hz ✓ $\frac{1}{10{,}000\ Hz}$

100E-6 49.7ms × 100E-6 4.97E-3

0.0001 % uncertainty or 4.97 µs ✗

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 interrupt, t_{pi}, and the maximum ISR execution time, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

497 pulses bit 0.1 ms 497 count period $t_p = N t_{ck}$ t_p = period measured N = count
$t_{ck} = 1/f_{ck}$ counter clock period

7

100 ms → longest + 20 ms → longest + 2 ms → ISR 122 ms
↓ logic-1 ↓ logic-0

10

10 + 20 + 2 = 32 ms

t_{pi} = 497 ms − 32 ms = 17.7 ms interrupt period ✗

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.

Name: Shane Tinklenberg

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.

84/100 = A-

93/100 = A

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)

9

1 Byte => 8 data bits + 1 start bit + 1 stop bit

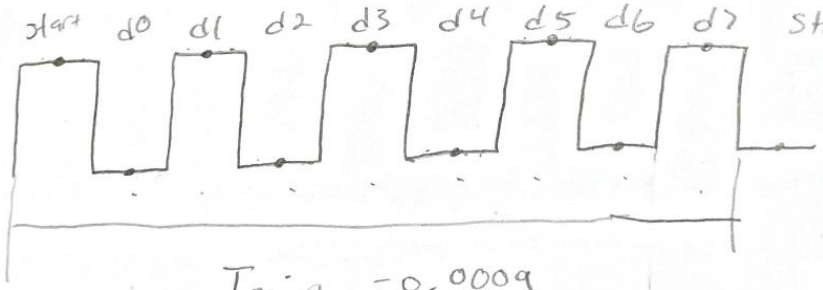
1 Byte = 10 baud Period = $\frac{1}{f} = \frac{1}{53{,}000}$

$1{,}048{,}576 \cdot 10 \text{ baud} \cdot \frac{1}{53{,}000} \approx$ 197.8445 s X

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?



20

25

$f = \frac{1}{P}$

$f = \frac{1}{0.0009}$

$T_{min} = 0.0009$

1 3 4 5 6 7 8 9

0.0001

$\frac{9}{10{,}000}$

$\frac{9}{10000} + \frac{1}{20000} = 5.23\%$

$5.23\% \cdot 10{,}000 = 9473.68$ < 10000 X

Slow

f = 9473.68 baud

% error = 5.23%

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)

30

Interrupt Density:

$$\frac{15}{1000} + \frac{300}{500} + \frac{600}{3.6 \cdot 10^9} = 0.615 < 1$$

Interrupt Density is OK ✓

Interrupt Constraint Inequalities: $N(i,k) = \text{ceil}(T_{pi-1}/T_k)$

$T_1 = 15\text{us}$ $T_2 = 300\text{us}$ $T_3 = 600$

$T_{P1} = 1000\text{us}$ $T_{P2} = 500$ $T_{P3} = 3.6 \cdot 10^9$

$N(1,1) = 1$ $N(2,1) = 1$ $N(3,1) = 3{,}600{,}000$

$N(2,2) = 1$ $N(3,2) = 71{,}999{,}999$

$N(3,3) = 1$

For interrupt 1:

$$T_{1+} + N(1,1)T_1 < T_{P1}$$

$$600 + (1)(15) < 1000\,\mu s$$

$$600_{\mu s} + 15\,\mu s < 1000\,\mu s$$

$$615 < 1000\,\mu s \quad ✓$$

For Interrupt 2:

$$T_{2+} + N(2,2)T_2 + N(2,1)T_1 < T_{P2}$$

$$600 + (1)300 + (1)(1) \times < 500$$

$$901 \not< 500 \quad \text{fail}$$

Continued on back!

#3 continued

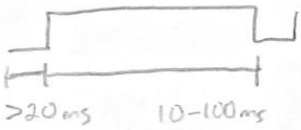
Because Interrupt 2 does not pass the interrupt interval constraint inequalities it will not opperate reliably, despite passing the interrupt density check.

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)

$(65535 - 65413) + 375 = 497$ ticks

497 ticks $\cdot 0.1$ ms $= 49.7$ ms 49.8 ms

14

You forgot to count the zero in . . .65535, 00000, 00001. . .

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

$P = \frac{1}{f} \Rightarrow f = \frac{1}{P} = \frac{1}{0.1 \cdot 10^{-3}}$

Uncertainty is up to $\pm 1/f_{clk}$

Thus uncertainty $\Rightarrow \frac{1}{10{,}000} = 0.0001$ s $= \boxed{0.01\%}$

5

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 allowable interrupt hold-off interval, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

Because it is rising or falling edge

$t_{pi} \geq 10$ ms

$t_i = 20$ ms

6

10

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.

Name: _____Nolan Vande Griend___

Dordt University Engineering Department

74/100 = B+

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.

84/100 = A-

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)

10

As a result of 8N1, it will take each byte 10 bit-cells of time to run. As this is going at 56000 baud with that being bits-per-second, so we are transmitting a bit cell every 1/56000 s. With this the every byte will take 1/5600 seconds to go through. Thus, following that math you would take approximately 187.2457142 seconds to transfer a 1 MB file.

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)

15

20

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.

My best answer would be that it could run at 10500 baud as it is checking at every half bit after the first has passed through. Though that is quite literally a shot in the dark, a shot is better than not even trying. The baud has been a bit difficult for me to understand recently.

It is wrong, but at least there is some method to it. Some sketching of a timing diagram would help your refine the idea.

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

I honestly have no answer for this as it is just blanking from my head, I would say that the error would be approximately 5 percent or so as it seems like you would subtract the difference between the 2 clock rates and then divide by the original and then multiply by 100 to find the error percentage.

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 things first, the interrupts seem to be able to operate smoothly as if you use the formula from the final slide from the notes on 2/24, you will find that the formula is equivalent to $\frac{15}{1000} + \frac{300}{500} + \frac{600}{3600000000}$ which will be less than 1, therefore passing the first check. There is a weird situation that may occur once the third interrupt has initiated as it always takes the same amount of time but with a timeframe longer than that of interrupt B which should occur every 500 microseconds and has a higher priority. Also, if A has been called between that as well then B will be skipped over for the equivalent running of 2 of the interrupts and not be able to run effectively. Therefore, this seems to be unreliable though this is more for the fact of making this question complete now instead as I am running out of time.

It is unreliable—it fails.

30

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)

The timeframe for the pulsewidth should be approximately 49.8ms as it took that many bits to count between the tick-clock values of pulseStartTime and pulseEndTime.

15

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

The time that the pulse width is active is between 10ms and 100ms but not more/less therefore the uncertainty we can have associated with the value of pulseWidth is ±45ms from 55ms if that's the proper way of saying that. X

1

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 interrupt, t_{pi}, and the maximum ISR execution time, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

X

I would hazard to say that the maximum period of time would be around 102 ms as it will take the maximum timeframe of 100ms pulse and including the runtime of 2 ms that would make sense. The minimum period of the interrupt should be approximately every 10 ms as it should be checking relatively frequently to see if the pulseStartTime has met the conditions necessary.

X

5

10

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.

Name: Matthew Van Eps

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.

76/100 = B+

83/100 = A-

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)

There are 9 bit-intervals/word.

$\frac{1}{56{,}000}$ = 17.86 µs per clock cycle = 142.8 µs per byte

8

Each clock cycle sends 1 bit. Each bit takes 17.86 µs. 1 stop bit in between each each 8 bit word. XON starts the sending and will have 8 bits. XOFF stops the sending and will contain 8 bits. $\frac{1{,}048{,}576}{8}$ = 1048576 stop bits

(142.8 µs × 1048576) + (1048576 × 178) + (16 × 17.8 µs)

= 168 seconds X

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)

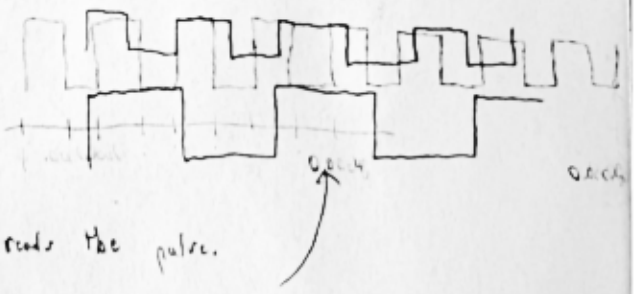
a.) Assuming instantaneous transitions of the signal at the receiver (assume an infinite slope when signals change state) **what is the maximum receiver rate in baud** that could work without error. That is, what is the maximum receiver clock rate that will not cause the sampling of the last bit of each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

~~17~~ 20

Word = 8 bits

$\frac{1}{10000}$ = 0.0001 second per bit



Need to make sure the clock edge reads the pulse. This receiver is going to fast

You would not want it going faster than

One word will take 0.0008 seconds so you don't want it to exceed 0.0001 seconds faster by the 5th bit which would be 0.0007 seconds.

$\frac{1}{x} \times 8 = 0.0007$ X

$\frac{1}{x} = 0.0000875 \rightarrow x = 11428$ baud A.

$100 - \left(\frac{10000}{11428} \times 100\right) = 12.5\%$ error B.

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)

$T_1 = 15\,\mu s$ $T_{p1} = 1000\,\mu s$ $T_{1t} = 600\,\mu s$

$T_2 = 300\,\mu s$ $T_{p2} = 500\,\mu s$ $T_{2t} = 600\,\mu s$

$T_3 = 600\,\mu s$ $T_{p3} = 1{,}000{,}000\,s$ $T_{3t} = 50\,\mu s$

$$N(i,x) = \frac{(T_{pi} - T_i)}{T_{px}}$$

$N(1,1) = 1$ $N(2,1) = 1$ $N(3,1) = 1000$

$N(2,2) = 1$ $N(3,2) = 2000$

$N(3,3) = 1$

30

interrupt density $= \frac{15}{1000} + \frac{300}{500} + \frac{600}{1000000} = 0.651 < 1$ ✓

① $T_{1t} + N(1,1)T_1 < T_{p1}$

$600 + 1(15) < 1000$

$615 < 1000$ ✓

② $600 + 1(300) + 1(15) < 500$

$915 < 500$ WRONG

It will fail here because under the worst circumstances, there will not be enough time to run everything. Latency is too long. 3rd scenario doesn't need to be tested.

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)

You forgot to count the zero in . . .65535, 00000, 00001. . .

14

if start is at 65413_{10} and stop is at 00375 and each tick clock is 0.1ms, then from (375 + (65535 − 65413))x 0.1ms

= 49.7 ms = pulse width of most recent pulse 49.8 ms

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

at least 20 ms to 100 ms

1

Anywhere from 20 to 100ms possible

49.7ms − 20 = 29.7ms ± 29.7 ms

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 allowable interrupt hold-off interval, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

if ISR is at most 2ms, then the max allowable hold off period $T_i = 2$

6

10

T_{pi} should be 10 ms because that is the shortest time it could be requested

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.

Name: Kyle Waas

Dordt University Engineering Department

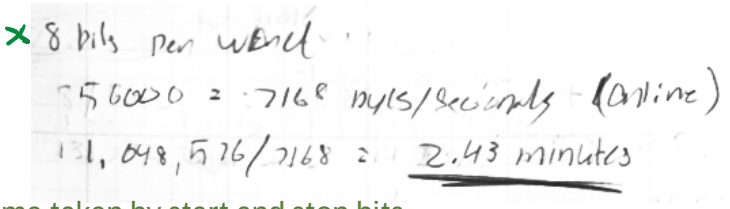
41/100 = C

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.

51/100 = B-

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)



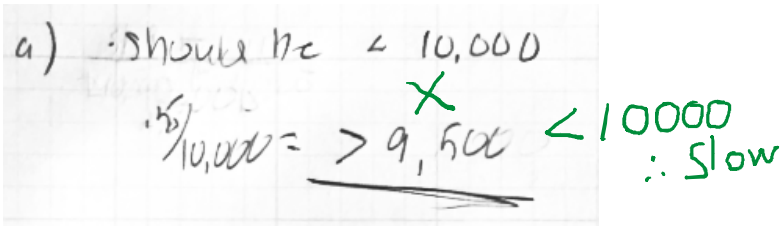
6

You ignored the time taken by start and stop bits.

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)

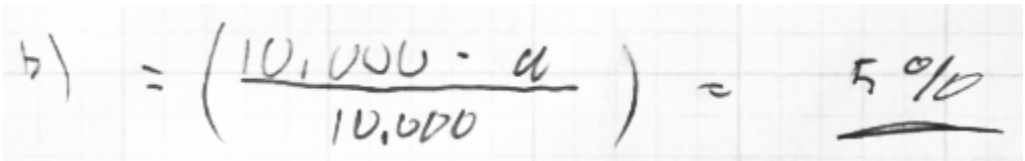
   a.) Assuming instantaneous transitions of the signal at the receiver (assume an infinite slope when signals change state) **what is the maximum receiver rate in baud** that could work without error. That is, what is the maximum receiver clock rate that will not cause the sampling of the last bit of each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



5

10



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.

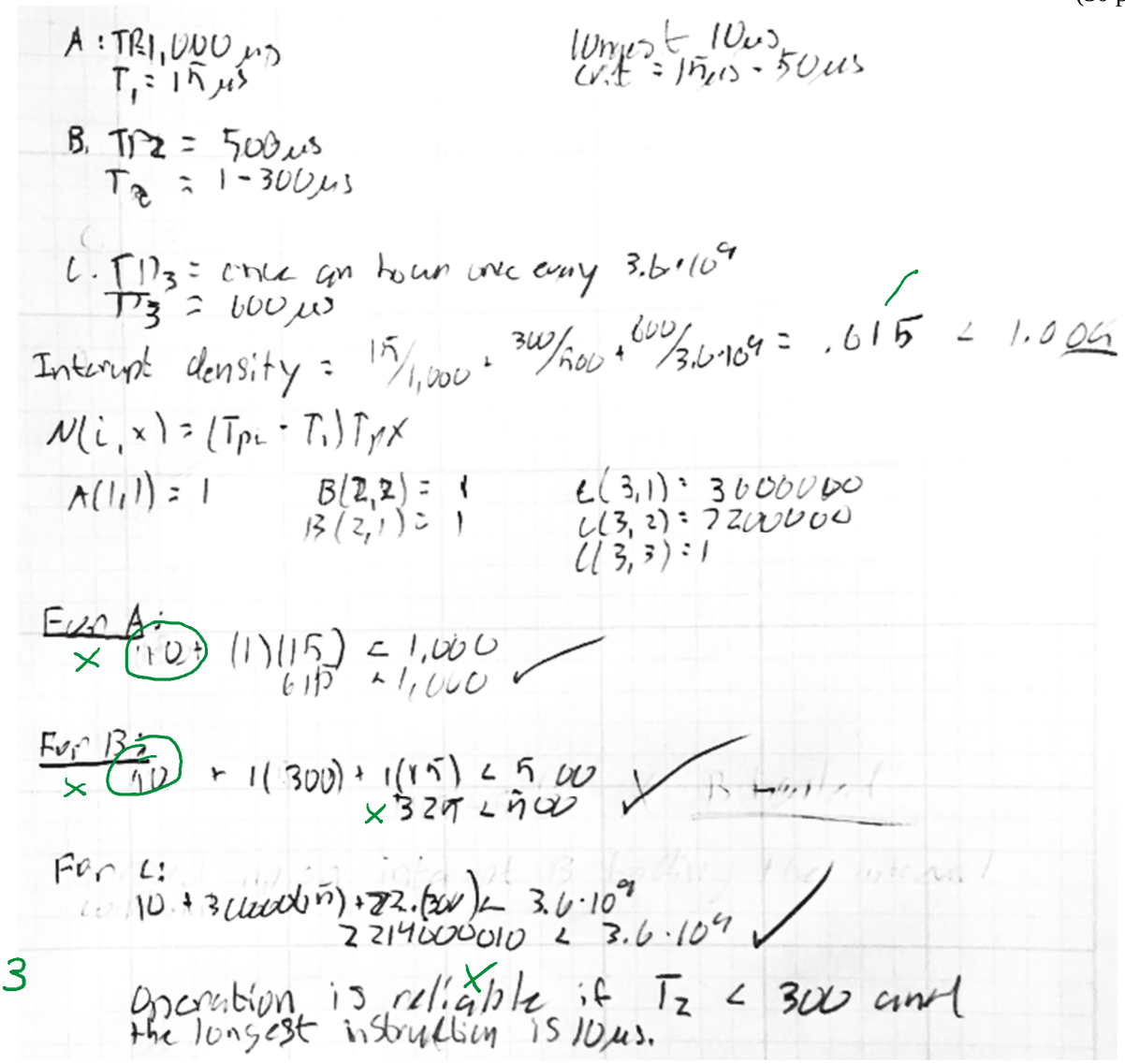
   Interrupt B is a tick-clock interrupt. It is periodic with a period of 500 µs. Depending on what is scheduled for a tick, it might take as little as 1 µs or as much as 300 µs to service.

   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)



The effects of lower-priority interrupts, t_{i+} intervals, has been incorrectly handled.

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)

1.00 when got here

a) $T = 1/f = 00375/65413 = .005$ ✗

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

b) This would depend what would happen w/ the tick clock. ✗

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 interrupt, t_{pi}, and the maximum ISR execution time, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

5 c)

10

There is a typo in part C that I did not recognized before the test. Everyone gets full credit (5 points) for the ti part.

DORDT UNIVERSITY
ENGINEERING DEPARTMENT

COURSE: EGR 304 SUBJECT: _____ NAME Kyle Wood

PAGE _____ OF _____ DATE: _____

1. 8 bits per word

56000 = 7168 bytes/seconds (online)

131,048,576/7168 = 18.28 minutes

2. a) should be < 10,000

.95/10,000 = > 9,500

b) = $\left(\frac{10,000 \cdot a}{10,000}\right)$ = 5%

DORDT UNIVERSITY
ENGINEERING DEPARTMENT

NAME _____

URSE: _____ SUBJECT: _____ PAGE _____ OF _____ DATE: _____

3. A: TR 1,000 μs
T_1 = 15 μs

10 μs + 10 μs
C.T = 15 μs - 50 μs

B. TR2 = 500 μs
T_2 = 1-300 μs

C. TR3 = once an hour once every $3.6 \cdot 10^9$
T_3 = 600 μs

Interrupt density = $15/1{,}000 + 300/500 + 600/3.6 \cdot 10^9 = .615 < 1.0$ OK

$N(i,x) = (T_{PL} - T_i) T_{MX}$

A(1,1) = 1 B(2,2) = 1 B(2,1) = 1 C(3,1) = 3600000 C(3,2) = 7200000 C(3,3) = 1

For A:
$10 + (1)(15) < 1{,}000$
$615 < 1{,}000$ ✓

For B:
$10 + 1(300) + 1(15) < 500$
$325 < 500$ ✓

For C:
$10 + 3(10000000) + 22.(300) < 3.6 \cdot 10^9$
$2214000010 < 3.6 \cdot 10^9$ ✓

Operation is reliable if $T_2 < 300$ and the longest instruction is 10 μs.

DORDT UNIVERSITY
ENGINEERING DEPARTMENT

NAME _____

SE: _____ SUBJECT: _____ PAGE _____ OF _____ DATE: _____

4) 1:00 when got here

a) $T = 1/f = .00375/63413 = .005$

b) This would depend what would happen w/ the tick clock.

c)

Name: Stefan Walkrod

Dordt University Engineering Department

80/100 = A-

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.

89/100 = A

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)

include start/stop bits

$$1048576 \text{ bytes} \cdot \frac{8 \text{ bits}}{\text{byte}} \cdot \frac{\text{baud}}{1 \text{ bit}} \cdot \frac{1 \text{ s}}{56000 \text{ baud}} \cdot \frac{10}{8}$$

10

$$= \boxed{187.25 \text{ s} = 3.12 \text{ min}}$$

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?

a) We can allow a max error of ± 1/2 bit interval

15 20 With 10 bits (8 data + start & stop), $\frac{0.5}{10} = 5\%$

Max receiver rate then is **10500 baud** ✗

At least there is some method here. A timing diagram would help you refine this idea and correct it.

b) This is 100% failure where every bit is sampled incorrectly ✗

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)

(µs)

high priority	A	$T_{PA} = 1000$	$T_A = 15$	$T_{A+} = 600$
	B	$T_{PB} = 500$	$T_B = 300$	$T_{B+} = 600$
	C	$T_{PC} = 3.6 \times 10^9$	$T_C = 600$	$T_{C+} = 10$

10 µs

$$T_{i+} \geq \sum_{x=1}^{i} N(i,x)\, T_x \leq T_{Pi}$$

$$600 + N(1,1) \cdot 15 \leq 1000$$

criticals

$$[50] + 600 + 15 < 1000 \quad \checkmark \text{ checks out, safe!}$$

Density:

$$\frac{15}{1000} + \frac{3}{5} + \frac{600}{3.6 \times 10^9} =$$

$$1.5E^{-2} + 6E^{-1} + 1.6E^{-7} = 0.615$$

Density is OK!

$$\left\lceil \frac{T_{PA} - T_A}{T_{PA}} \right\rceil = 1$$

$$N(i,x) = \left\lceil \frac{T_{Pi} - T_i}{T_{Px}} \right\rceil$$

$$[50] + 600 + N(2,2) \cdot 300 + N(2,1) \cdot 15 < 500$$

Fail. T_{B+} is too high for T_{PB}. Either make T_C take less time to service, put it up in priority, or make T_{PB} higher (more rare).

If C is called in the period between B needing service, it won't get done in time.

30

10 - 100 ms, 20 ms +

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)

You forgot to count the zero in . . .65535, 00000, 00001. . .

14

65413_{10} = start
00375_{10} = end

Assume 1 rollover

$$\begin{array}{r} 00375 \\ +65535 \\ \hline 65910 \\ -65413 \\ \hline 497 \end{array}$$

497 ticks · 0.1 ms = 49.7 ms

X

49.8 ms

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

5

$\frac{0.1\ ms}{10\ ms}$ = 1% max uncertainty

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 allowable interrupt hold-off interval, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

Counter timer ISR: $T_1 = 100$ ms $T_{1P} = 120$ $T_{1r} = t_i$

Pin ISR: $T_2 = 2$ ms $T_{2P} = t_{pi}$ $T_{2r} = t_i$

10

$t_i + 1 \cdot 100 < 120$

$t_i < 20$ ms

X

$t_{pi} > 122$ ms

X

10

$t_i + 1 \cdot 2ms + \frac{t_{pi} - R}{120} \cdot 100 < t_{pi}$

$20 + 2 + \frac{t_{pi} - 2}{120} \cdot 100 < t_{pi}$

There is a typo in part C that I did not recognized before the test. Everyone gets full credit (5 points) for the ti part.

Name: Ty White

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.

82/100 = A-

92/100 = A

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)

10 bits/Byte = 10 · 1/56k = 0.000178571 s/Byte

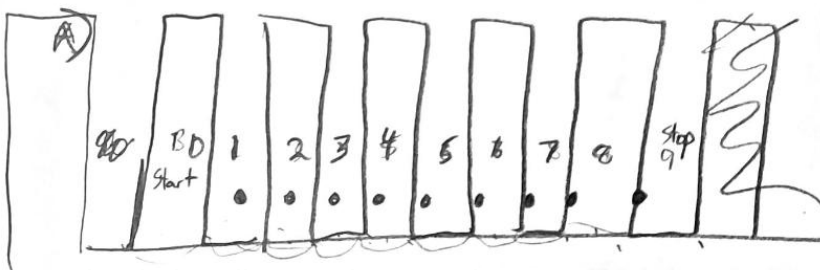
10

1048576 byte · 0.000178571 s / Byte = 187.25 s

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?



1/10000 = 1 bit length (7.5) = 0.000075 s faster per bit

The timing diagram is a good start.
The analysis following it is wrong.

7.5 bits lengths = minimum

0.0001 - 0.000075 = 0.000025 Bit/s

= 40000 Baud maximum

15

25

25%

normal = 1/10000 Bit/s

fastest = 1/40000 Bit/s

(1/40000) / (1/10000) = 25% error max

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)

A)

B)

C)

30

$T_{P1} = 1000\mu s$ $T_1 = 15\mu s$ $T_{1+} = 600\mu s$

$T_{P2} = 500\mu s$ $T_2 = 300\mu s$ $T_{2+} = 600\mu s$

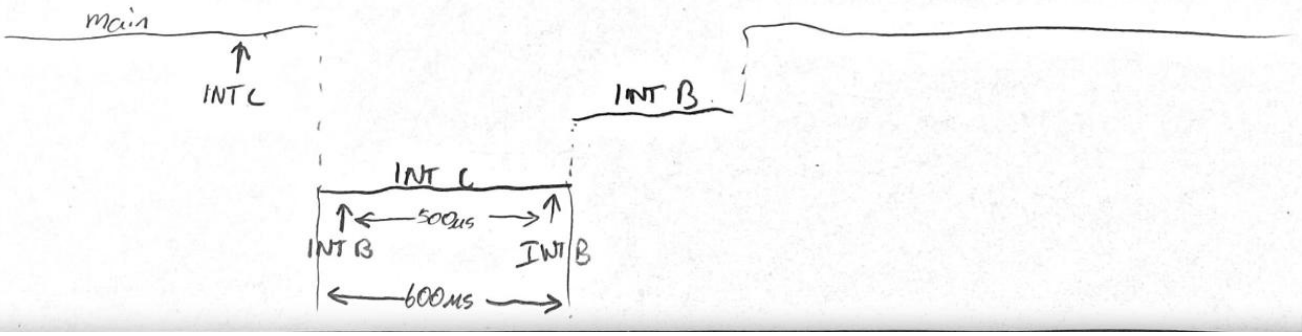
$T_{P3} = 3.6\times10^9\ \mu s$ $T_3 = 600\mu s$ $T_{3+} = 50\mu s$

1) $T_{1+} + N(1,1)T_1 \overset{?}{<} T_{P1} \rightarrow 600\mu s + 1(15\mu s) < 1000\mu s$ OKAY

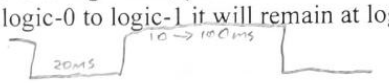
2) $T_{2+} + N(2,1)T_1 + N(2,2)T_2 < T_{P2} \rightarrow 600\mu s + \left(\frac{500\mu s - 300}{1000}\right)(15\mu s) + 1(300\mu s) \overset{?}{<} 500\mu s$ NO!

Error

If Interrupt C is called, interrupt B will be called twice, so one calling of B will be missed.



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},. . .)

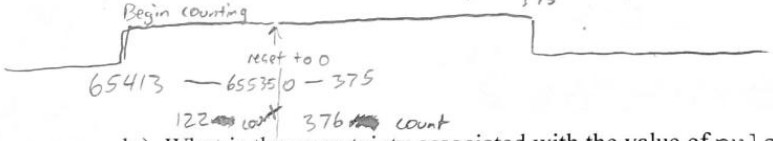
The pulse to be measured is applied to an input pin that is set to interrupt on each edge, rising or falling. (This input pin interrupt is a second interrupt source in addition to the tick-clock interrupt.) This same hardware interrupt is set up to immediately (via hardware) *capture* the tick-clock 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)

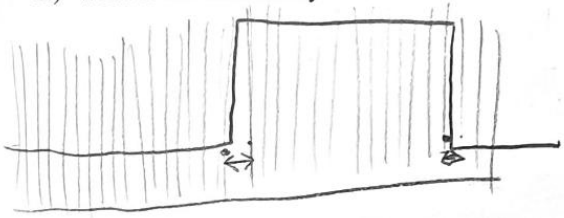
15

$0.1ms((65535 - 65413) + 376) = 498\,ms = 0.0498$ seconds ✓



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

2



small vertical = tick clock
Big Signal = Pulse

error max $= 2(0.1\,ms) = \pm 0.2\,ms$ ✓

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 interrupt, t_{pi}, and the maximum ISR execution time, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

10

$T_{P1} = 30ms$ (10+20) $T_1 = 2ms$ $T_{1+} =$
$T_{P2} =$ $T_2 =$ $T_{2+} =$

$T_{pi} = 10ms + 20ms = 30ms$ minimum ✓
$t_i = 2ms$ max ✓

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.

Name: Charley Young

Dordt University Engineering Department

~~61/100 = B+~~ 71/100 = B+

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)

because baud rate is pulses per second
assuming one pulse = one bit
also 1 byte = 8 bits
every byte needs 1 stop bit

$$\frac{1048\,576 \cdot (8+1)}{56000} = 168.52 \text{ seconds}$$

8

Every byte needs 1 stop bit—and also 1 start bit. 10 bit-intervals/word.

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?

Did not have time

0

5

As per our Zoom meeting, you spent time evaluating this problem which gave you less time to work the other problems, which are done generally well, thus some sympathy points here.

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)

30

$T_{PA} = 1000\,\mu s$ $T_A = 15\,\mu s$

$T_{PB} = 500\,\mu s$ $T_B = 1\text{–}300\,\mu s$

$T_{PC} = 3.6\cdot10^9$ $T_C = 600\,\mu s$

Int Density Ratio

$\text{Int Density} = \frac{15}{1000} + \frac{300}{500} + \frac{600}{3.6\cdot10^9} = 0.615$

$0.615 < 1$ OK ✓

$N(A,A) = 1$

$N(B,A) = \frac{500-300}{1000} = 1$ (Call this)

or $\frac{500-1}{1000} \approx 1$

$N(B,B) = \frac{500-500}{500} = 1$ ✓ (should equal 1)

$N(C,A) = \frac{3.6\cdot10^9 - 600}{1000} = 3,600,000$

$N(C,B) = \frac{3.6\cdot10^9 - 600}{300} = 11,999,998$

or $\frac{3.6\cdot10^9 - 600}{1} = 3,599,999,400$

all fractions are rounded up to nearest integer

Gen: $N(C,x) = \left(\frac{T_{Px} - T_x}{T_{Px}}\right)$

$N(C,C) = 1$

$T_{A0} = 600\,\mu s$ $T_{B0} = 600\,\mu s$ $T_{C0} = 50\,\mu s$

For interrupt A

$T_{A0} + N(B,B)T_B < T_{PA}$

$600 + (1)(300\,\mu s) < 1000$

$900\,\mu s < 1000\,\mu s$ Int A is OK ✓

For Interrupt B

$T_{B0} + T_B + N(B,A)T_A < T_{PB}$

$600 + 300 + (1)15 < 500$

$915 \not< 500$

even if $T_B = 1$ then $601 \not< 500$

This is unreliable

The latency is too long causing this interrupt to possibly not get serviced on time

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)

375 - 65413 + 65535 = 498 ✓

65535 00375 ← because it should count 00000
13 −65413 +00001
122 + 376 = 498 ms ← length of pulse in ms
= 0.498 seconds ✗

Power of 10 error

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

5 $\frac{1}{\text{PrescaledClock}} \pm \frac{1}{1000\text{Hz}}$ = ± 0.001 seconds ✗

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 interrupt, t_{pi}, and the maximum ISR execution time, t_i, that should be specified for the pin ISR? (5 + 5 = 10 points)

2ms

5 ~~5~~
10

it should be greater than the sum of $T_{pi} + T_i$

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.

Name: Ryan Zevenbergen

Dordt University Engineering Department

95/100 = A

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.

98/100 = A

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)

$$\frac{1{,}048{,}576 \text{ bytes}}{1} \times \frac{8 \text{ bits}}{1 \text{ byte}} \times \frac{\text{sec}}{56{,}000 \text{ bits}} = 149.80 \text{ sec for data}$$

10

$$\frac{1 \text{ start}}{\text{word}} \times \frac{1{,}048{,}576 \text{ words}}{56{,}000 \text{ bits}} = 18.72 \text{ sec} \qquad 149.8 + 18.72 + 18.72 = \boxed{187.25 \text{ s}}$$

$$\frac{1 \text{ stop}}{\text{word}} \times \frac{1{,}048{,}576 \text{ words}}{56{,}000 \text{ bits}} = 18.72 \text{ sec} \qquad = \boxed{3 \text{ min } 7 \text{ sec}}$$

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?

a.) If start and stop bits are omitted, and only the data is referenced:

The clock would need to move just over half the bit length over the course of 8 bits. This is 0.5/8 % of the bit, or 6.25%

28

6.25% · 10,000 = 625

baud R_x = $\boxed{10{,}625 \text{ baud (just greater than)}}$

b.) $\frac{10{,}000}{10{,}625} = 94.1\%$ reliable → $\boxed{5.9\% \text{ error}}$

Wrong reference. You want (10625/10000) - 1 = 6.25%

* For inclusion of start and stops, replace 8 with 10 in these calculations

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)

$T_A = 15_{\mu s}$, $TP_A = 1000_{\mu s}$

$T_B = 300_{\mu s}$, $TP_B = 500_{\mu s}$

$T_C = 600_{\mu s}$, $TP_C = 3.6 \times 10^9 \mu s$

$N(A,A) = 1$ $N(B,A) = 1$ $N(CA,) = 3{,}600{,}000$

$N(B,B) = 1$ $N(C,B) = 7{,}200{,}000$

$N(C,C) = 1$

$\frac{15}{1000} + \frac{300}{500} + \frac{600}{3.6\times10^9} = 0.6150 < 1 = ok$

$T_{A+} = 600_{\mu s}$

$T_{B+} = 600_{\mu s}$

$T_{C+} = X$

For interrupt A

$600_{\mu s} + N(A,A) \cdot T_A = 615_{\mu s} < 1000_{\mu s} \rightarrow$ ok!

For interrupt B

$600_{\mu s} + N(B,B)T_2 + N(B,A)T_1 = 915_{\mu s} > 500_{\mu s} \rightarrow$ Not ok!

This means that interrupt B could be called again while it is still running! This causes a loop, in which the main program will never be run again.

This description is not exactly correct the the gist of it is OK-- the program will fail, and specifically the function of ISR B will fail.

30

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)

15 If working correctly, it would recognize the overflow and return the proper value: 49.8 ms = 0.0498 s ✓
If there were no checks for overflow, however, it would erroneously return −6503.8 ms = −6.5038 s.

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

5 Clock resolution: 0.1 ms
Min pulse: 10 ms

$$\frac{0.1\text{ ms}}{10\text{ ms}} = 1\%$$ ✓

Interestingly, with unsigned binary subtraction checking for rollover is not needed, just ignore the most significant borrow!

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 interrupt, t_{pi}, and the maximum ISR execution time, t_i that should be specified for the pin ISR? (5 + 5 = 10 points)

7
10 Error = 1% · longest pulse = 1 ms

t_i = 2 ms + error = 3 ms ✓

t_{pi} = 20 ms + t_i = 23 ms ✗

There is a typo in part C that I did not recognized before the test.
Everyone gets full credit (5 points) for the ti part.