Embedded Systems: Homework #3

Yousef Alaa Awad

October 19, 2025

1

Given: In a UART communication system, the receiver is configured to oversample at 16x the baud rate, and the communication happens at a baud rate of 9600 bps. The UART transmits 8 data bits, 1 start bit, 1 stop bit, and no parity.

A) What is the minimum required sampling frequency of the UART receiver

With a given baud rate of 9600bps and an oversampling factor of 16, the sampling frequency of the reciever can be calculated as the following:

$\text{Sampling Frequency} = \text{Baud Rate} * \text{Oversampling Factor} \rightarrow \text{Sampling Frequency} = 9600 * 16 = 153,600\ Hz$

B) Calculate how much time it will take to transmit a single byte (including the start and stop bits).

Now, since each byte contains 1 start bit, and one stop bit, this would mean that each frame has 10 bits total (start + data + stop). Now, the time to transmit one byte/message is simply:

$$t_{bit} = \frac{1}{\text{Baud Rate}} = \frac{1}{9600} \approx 104.17\mu s$$

Therefore, for one byte to transmit we simply need to do the following:

$$t_{byte} = t_{bit} * 10 = 104.17 * 10 = 10417\mu s = 1.0417ms$$

C) Suppose the transmitting UART has a clock that is running 2% faster than the receiving UART. Given that the receiver oversamples each bit 16 times, how many incorrect samples could occur due to this clock mismatch over the course of 10 bits (1 start bit, 8 data bits, and 1 stop bit)? Would the receiver still be able to correctly sample the bits?

To calculate the amount of incorrect samples that could occur we first have to calculate the drift per bit:

$$Drift_{bit} = 0.02 * 16 = 0.32 \text{ samples per bit}$$

And when done over 10 bits of message:

$$Drift_{total} = 0.32 * 10 = 3.2 \text{ samples}$$

This number indicates to us that there is about 3-4 possible incorrect samples that would occur due to the clock mismatch. Thankfully, though, the reciever samples each bit 16 times, making it so that the majority of the samples will still be correct.

2

Given: A system requires UART communication at a baud rate of 115200 bps, and you need to configure the UART for 8 data bits, no parity, 1 stop bit, and SMCLK as the clock source. The SMCLK frequency is set to 4 MHz.

A) What would be the hexadecimal value of UCAxCTLW0, UCAxMCTLW, and UCAxBRW for this configuration if oversampling is disabled.

Assuming that oversampling is disabled, the divider of the baud rate is simply just the SMLCLK divided by the baud rate, or:

$$\frac{4 * 10^6}{115200} \approx 34.722$$

Now, to get the register value of 34.722 we put the whole portion into the UCAxBRW register via converting 34 to hex (of which is 0x0022), and the 0.722 into UCBRSx into hex is 0xBB. Since oversampling is disabled we then also know that UCBRFx is 0x0 and UCOS16 is 0. Therefore, the following are the registers:

$$UCAxBRW = 0x0022$$
$$UCAxMCTLW = 0xBB00$$

Now, with 8 data bits, 0 parity bits, 1 stop bit, and the SMLK as the source, then you would have the following:

$$UCAxCTLW0\ (when\ operating) = 0x0880$$
$$UCAxCTLW0\ (when\ reset) = 0x0881$$

B) What would be the hexadecimal value of UCAxCTLW0, UCAxMCTLW, and UCAxBRW for this configuration if oversampling (with factor of 16x) is enabled.

Now when oversampling is enabled, then we will have a new baud rate divider of...

$$\frac{4 * 10^6}{115200 * 16} \approx 2.170$$

Now, again, UCAxBRW is simply the hex of the whole number, or HEX(2) which is 0x0002. Now, according to the table mapping the fractional portion we get a value for UCBRSx to be 0xBB. Now, unlike part A of this question, we have oversampling enabled with 16 times, so UCOS16 is now 1 AND UCBRFx is $0.17 * 16$ or 0x2. This therefore means that the following are the registers:

$$UCAxBRW = 0x0002$$
$$UCAxMCTLW = 0xBB21$$
$$UCAxCTLW0\ (when\ operating) = 0x0880$$
$$UCAxCTLW0\ (when\ reset) = 0x0881$$

3

Given: In an embedded system, the system clock (SMCLK) is running at 8 MHz, and the UART baud rate is determined by a clock divider stored in the UCAxBRW register. The oversampling mode can either be enabled or disabled. The UART supports a 16-bit divider, meaning the value of UCAxBRW can range from 0x0001 to 0xFFFF. The oversampling mode (if enabled) divides the effective clock frequency by 16 before applying the divider.

A) Calculate the maximum baud rate achievable when oversampling is disabled.

The maximum baud rate when oversampling is disabled is the following:

$$\frac{8,000,000}{1} = 8,000,000 \text{ bps}$$

B) Calculate the minimum baud rate achievable when oversampling is enabled.

The maximum baud rate when oversampling is disabled is the following:

$$New\ Clock_{Effective} = \frac{8,000,000}{1 * 16} = 500,000 \text{ Hz}$$

And with this we can calculate a minimum bps of...

$$\frac{500,000}{65,535} \approx 7.629 \text{ bps}$$

4

Given: An embedded microcontroller is communicating with a sensor using UART (8 data bits, no parity, 1 stop bit). The microcontroller's UART is configured to transmit at a baud rate of 115200 bps, derived from its 16 MHz system clock. However, due to a manufacturing defect, the microcontroller's clock is actually running at 15.6 MHz. The sensor's UART operates at an accurate baud rate of 115200 bps.

A) Calculate the actual baud rate at which the microcontroller is transmitting data due to the defective clock.

First we find the baud rate via the formula (used above again and again) of...

$$\frac{\text{System Clock}}{\text{Baud Rate Divider}} = \frac{16 * 10^6}{115200} \approx 138.889$$

Now, we will simply round to 139, as the divider must be an integer (whole number). Therefore, to calculate the actual baud rate (as we have a defective clock of 15.6MHz) we do the following:

$$\frac{15.6 * 10^6}{139} \approx 112,230.22 \text{ bps}$$

B) Determine the percentage difference between the microcontroller's actual baud rate and the sensor's baud rate.

Now, to determine the percentage difference we use the following formula:

$$\frac{|value1 - value2|}{(\frac{value1+value2}{2})} * 100 = \frac{|115,200 - 112,230.22|}{(\frac{115,200+112,230.22}{2})} * 100 \approx 2.6\%$$

C) UART communication can tolerate up to a ±2% difference in baud rates between the transmitter and receiver for reliable communication. Given the calculated baud rate difference, will the microcontroller be able to communicate reliably with the sensor?

Since the calculated difference is over the 2% allowed tolerance, the devices will not be guaranteed to have reliable communication.

D) Considering the cumulative timing error, after how many bits will the timing error amount to half a bit period, potentially causing a framing error?

To calculalte the total number of bits that it will take for the total timing error amount to be half a bit period we first need to calculate the bit times for each device:

$$Sensor_{bit\ time} = \frac{1}{115,200} \approx 8.6806\mu s$$
$$Microcontroller_{bit\ time} = \frac{1}{112,230.22} \approx 8.9065\mu s$$

Now, we calculate the difference of the bit times via $Microcontroller_{bit\ time} - Sensor_{bit\ time}$ getting approximately $0.2259\mu s$. Now, to calculate the half a bit period time of the sensor we simply half the bit period of the sensor (revolutionary, I know):

$$\frac{Sensor_{bit\ time}}{2} \approx 4.3403\mu s$$

And after this, we simply divide that number we just calculated via the difference of times to find the number of bits for the elapsed error of time to occur:

$$\frac{\frac{Sensor_{bit\ time}}{2}}{Difference_{bit\ time}} \approx 19.22 \text{ bits}$$

Now, to summarize, it would take around 19-20 bits for the cumulative error to reach half of a bit period, or, since each frame is 10 bits in our case, almost 2 whole frames.

5

Given: You are designing an embedded system that communicates over an I2C bus at 400 kHz (Fast Mode). The microcontroller you are using has a peripheral clock frequency of 8 MHz for its I2C module. The I2C peripheral requires you to configure the I2C clock speed by setting a prescaler and a divider to generate the desired clock frequency. The formula for calculating the I2C clock is given by:

$$f_{SCL} = \frac{f_{PCLK}}{2 * (PRESCALER + 1) * (DIVIDER + 1)}$$

Where f_{SCL} is the desired I2C clock frequency (in Hz), f_{PCLK} is the peripheral clock frequency (in Hz), PRESCALER is the value of the prescaler (0 to 15), and DIVIDER is the value of the divider (0 to 255).

A) Calculate the values of PRESCALER and DIVIDER to achieve an I2C clock frequency of 400 kHz. You are required to select the smallest possible values for both PRESCALER and DIVIDER.

Now, given the formula of...

$$f_{SCL} = \frac{f_{PCLK}}{2 * (PRESCALAR + 1) * (DIVIDER + 1)}$$

and plugging in the values given in the problem...

$$400 * 10^3 = \frac{8 * 10^6}{2 * (PRESCALAR + 1) * (DIVIDER + 1)}$$

and when rearranged to find the possible values of PRESCALAR and DIVIDER, we have te following:

$$10 = (PRESCALAR + 1) * (DIVIDER + 1)$$

Therefore meaning, since PRESCALAR can only be 0-15 and DIVIDER being only from 0-255, the smallest values for both are...

$$PRESCALAR = 0$$
$$DIVIDER = 9$$

B) What would the resulting actual I2C clock frequency be with your selected configuration, and how much error (%) does this configuration introduce compared to the desired 400 kHz frequency?

Now, for the selected error with the configuration with the desired 400KHz frequency would first depend on the actual f_{SCL} that the values create...

$$f_{SCL} = \frac{f_{PCLK}}{2 * (PRESCALAR + 1) * (DIVIDER + 1)} = \frac{8 * 10^6}{2 * (0 + 1) * (9 + 1)} = 400,000 \text{ Hz}$$

Since this value is exactly the same as the desired value, it means that we have **no** error, or **0%**.