YOGYA SHARMA
ys5250@nyu.edu

EL-GY 6483
Communications

- You are encouraged to work in groups, and to use the Internet or any other tools available to learn the material in order to answer these questions.

1. The ATmega128 microcontroller includes a UART that can be used to provide a serial interface. The following code snippet is often seen in programs that use the UART interface:

```
while(!(UCSR0A & 0x20));
  UDR0 = x;
```

where `x` is a previously declared and initalized `uint8_t`; `UCSR0A` and `UDR0` are defined in header files to refer to memory locations corresponding to the USART Control and Status Register A and USART Data Register, respectively; and the UART interface has already been configured, i.e. is ready for use.

   (a) Refer to page 188 of the manual for the ATmega128, (available online). What does each line of the code snippet above do, with respect to the peripheral registers? What does the code snippet as a whole do?

Solution:

while (!(UCSR0A & 0x20)); This statement waits untill the USART Data register Empty register (bit 5 in the UCSR0A register), is set to 1. This indicates that the transmit buffer is empty and can now be written into.

UDR0 = x; This places the byte in x in the transmit buffer, to be sent over the serial interface.

This code snippet sends a byte data over the serial interface.

(b) Suppose that the serial port operates at 57600 baud and the processor operates at 8 MHz. Approximately how many processor cycles are consumed by the code snippet above?

Solution: If the transmit buffer is already empty when this code is run, the condition in the while loop will be false the first time it is checked. In this situation, it will take a few number of CPU cycles to check the flag status, and place the byte in the transmit buffer.

Alternatively, if there is already a byte in the transmit buffer immediately before the code is run, then the transmit buffer won't be empty for 138.9 μs (= 8 bits / baudrate). During this time the processor will keep on re-evaluating this condition in the while loop. During this it will take,
$= 8/57600 / \frac{1}{8000000} = 1112$ processor cycles wastefully.

(c) To receive a byte over the serial port, a programmer might use the following code snippet to implement a `readByte()` function:

```
uint8_t readByte() {
  while(!(UCSR0A & 0x80));
    return UDR0;
}
```

What will happen if `readByte()` is called and there is no incoming byte over the serial interface?

Solution: The program will wait for readByte() to return, indefinetly.

(d) We say that a call to an I/O function is *blocking* if it blocks the calling program from continuing until the communication has finished. (Look up "Asynchronous I/O" on Wikipedia for more details.) Is a call to `readByte()` blocking? Why might this be problematic in some cases? Can you implement a non-blocking version of `readByte()`?

Solution:
A call to readByte() is blocking. The calling program won't proceed until a byte has been received. Unless, the program will be stuck forever.

To implement a non-blocking recieve will be to pass an extra argument by reference, and set its value to indicate receipt of a byte.

```
uint8_t readByte (uint8_t *rec ) {
  if (!(UCSR0A & 0x80)) {
        *rec   = 0;
        return 0;
  }
  *rec = 1;
  return UDR0;
}
```

The calling program can check in "rec" to know whether or not the byte was received. In case not, the program can branch to another task.

In interrupt driven I/O, a signal is generated on certain I/O events, so that the processor doesn't need to poll.

(e) On this microcontroller, the baud rate is set by writing the value $UBRR = \frac{f_{osc}}{16 B_{des}} - 1$ to a `UBRR` register, where f_{osc} is the oscillator frequency in Hz and B_{des} is the desired baud rate in bits per second. The achieved baud rate is then $B_{ach} = \frac{f_{osc}}{16(UBRR+1)}$ (See page 172-173 of the ATmega128 reference manual for more details.) Because we can only write integer values to the register, not all baud rates can be achieved exactly.

- What is the closest we can get to 57600 baud (i.e., what is B_{ach}) if f_{osc} is 8 MHz? (Assume `U2X` is 0.)
- What value should be written to the `UBRR` register to achieve this baud rate?
- What is the percent error in this case, calculated as $\left(\frac{B_{ach}}{B_{des}} - 1\right) \times 100\%$?

Solution:

$UBRR = \frac{8\times10^6}{16\times57600} - 1 = 7.68$, can be approximated to 8.

For achieved baudrate, $B_{ach} = \frac{8\times10^6}{16(8+1)} \approx 55555$

For percentage error $\rightarrow \left(\frac{55555}{57600} - 1\right) \times 100 = -3.55\%$

(f) Suppose the other communication partner is an ATmega128 using $f_{osc} = 2$MHz. (Assume `U2X` is 0.) What will its B_{ach} be if it tries to operate at 57600 baud? What will be the total error between the pair, and is it less than the maximum error recommended in Table 75 of the ATmega128 reference manual (page 186)?

Solution:

6

$UBRR = \frac{2\times10^6}{16\times57600} - 1 = 1.17$, rounding to the nearest integer $\rightarrow 1$.

Achieved baudrate $B_{ach} = \frac{2\times10^6}{16(1+1)} = 62500$

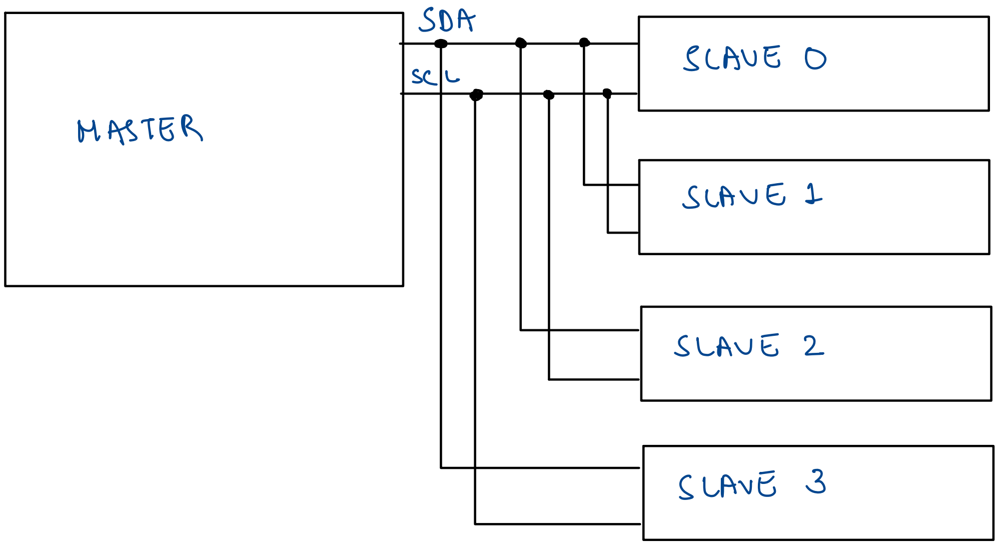
Percentage error : from 57600 $= \left(\frac{62500}{57600} - 1\right) \times 100 = 8.5\%$ } indicating too fast baud rate.

from 55555 $= \left(\frac{62500}{55555} - 1\right) \times 100 = 12.5\%$

This device is operating 8.5% faster than it is supposed to, and 12.5% faster than the communication partner. The total error is definitely higher than the maxi. error recommended in table 75.

2. Assume you have four (slave) devices connected to a (master) microcontroller over a shared I2C bus that uses standard (7-bit) addressing and is running at 400 kHz (most I2C devices can communicate at 100 kHz or 400 kHz).

   (a) Draw a connection diagram for this configuration. What is the total number of wires?

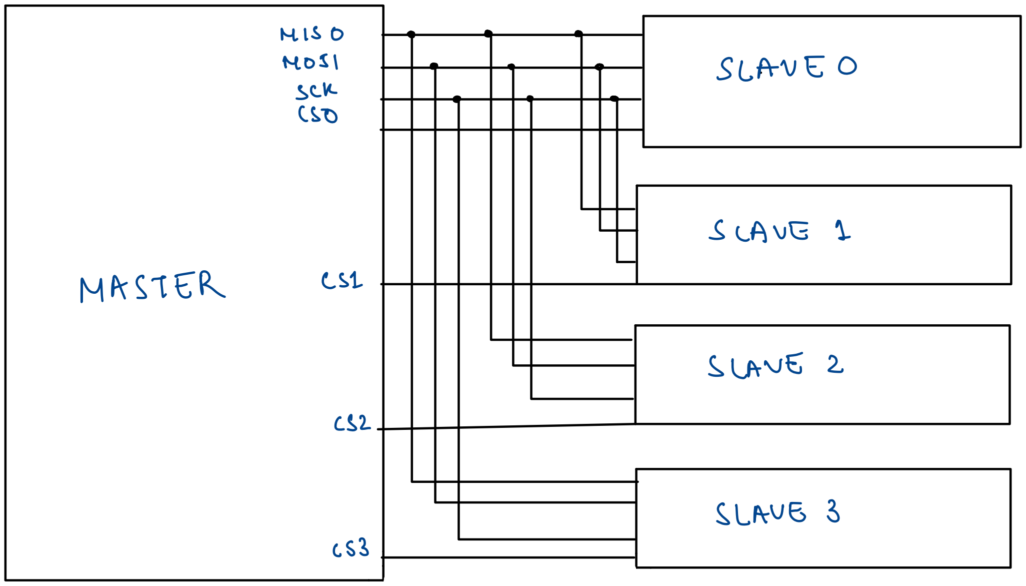


   (b) Suppose the microcontroller reads one data byte from each of the four devices sequentially. (This is similar to the single-byte read shown on page 33 of the lecture slides, but instead of a stop at the end, there is a repeated start condition followed by a different slave address.) What is the data transfer rate in (*data*) bits per second from *each device*? (In other words, over some long interval of time T, how many bytes can slave S_1 transmit?)

Solution:
lets call each single byte read an I2C read transaction. Each transaction involves approximately the following:
• a start signal • a slave address + write bit + acknowledgement (9 bits)
• a register address + acknowledgement (9 bits) • A repeated start signal • A slave address + read bit + acknowledgement (9 bits) • A data byte + acknowledgement (9 bits) • A stop or repeated start bit, for a total of 40 bits on the wire for each byte read from slave. That's about 40 bits with a duration of 1/400000 seconds each, for a total of 100µs per transaction. Since we're polling four devices in sequence, each device will get to send one byte every 4 transactions (400µs) for a data rate of about 20 kb/s.

(c) Repeat parts (a) and (b) for the SPI equivalent of the same setup.

Solution:



An SPI read transaction involves, approximately, the following:
- a register address + read bit is sent to the slave (8 bits) • The slave returns a data byte (8 bits), for a total of about 16 bits on the wire for each byte read from the slave. That's about 16 bits with a duration of 1/400000 seconds each, for a total of 40 μs per transactions. Since we are polling four devices in sequence, each device will get to send one byte every four transactions (160 μs) for a data rate of about 50 kb/s.