Karan Vora (kv2154)
Real-Time Embedded Systems Assignment 4 (EL648HW4)

Problem 1):

Solution a):

The line

while (! (UCSROA & 0x20));

waits until the USART Data Register Empty Register (bit 5 in the UCSROA Register) is set to one, indicating that the transmit buffer is empty and ready to be written to.

The line

UDRO = x

places the byte in x in the transmit buffer, to be sent over the serial interface. As a whole, the code snippet sends a byte of data over the serial interface.

Solution b):

If the transmit buffer is empty already when we run this code, the condition in the while loop will be false the first time it is check. In this scenario, it take a few CPU cycles to check the flag status and place the byte in the transmit buffer. But if there is a byte in the transmit buffer immediately before we run this code, then the transmit buffer will not become empty for about 139 µs. During that time, the processor will keep re-evaluating the condition in the while loop over and over again, consuming about 1112 clock cycles without doing any useful work.

Solution c):

The calling program will wait forever for readbyte() to return.

Solution d):

A call to readbyte() is blocking. The calling program cannot continue until a byte has been received. If no byte is received, the calling program will be blocked indefinitely. One way to implement a non-blocking receive would be to pass an additional argument by reference, and set its value to indicate whether a byte was received. The function would return immediately, whether a byte is ready to be received or not.

```
uint8_t readbyte(uint8_t *status)
{
        if (!(UCSROA & 0x80))
        {
                *status = 0;
                return 0;
        }
        *status = 1;
        return UDRO;
}
```

The calling program can check the value in status to know whether or not a byte was received. If it wasn't, the calling program can decide whether to try again immediately, or do some other work. It isn't forced to wait and consume useless CPU cycles.

Solution e):

Here, we calculate UBRR = $\frac{8*10^6}{16*57600}$ = 1 ≈ 7.68. Since we can only write an integer value to the UBRR register, so approx 8.

Then, to calculate the achieved baud rate, we will use B_{ach} = $\frac{8*10^6}{16*(8+1)}$ ≈ 55555.

Finally, to calculate the percent error, we'll use $(\frac{55555}{57600}-1)\,x\,100\,\%$ = -3.55%

Thus,
The closet achievable baud rate is ≈ 55555
Write 8 to UBRR register to achieve this rate.
The percent error is -3.55%, Negative percent error indicates that baud rate is too slow

Solution f):

Here, we calculate UBRR = $\frac{2*10^6}{16*57600}-1$ ≈ 1.17. Since we can only write an integer value to the UBRR register, we'll round to the nearest integer, which is 1.

Then, to calculate the achieved baud rate, we will use B_{ach} = $\frac{2*10^6}{16*(1+1)}$ = 62500.

The percent error from $57600*(\frac{62500}{57600}-1)\,x\,100\,\%$ ≈ 8.5%, A positive percent error indicating the baud rate is too fast.

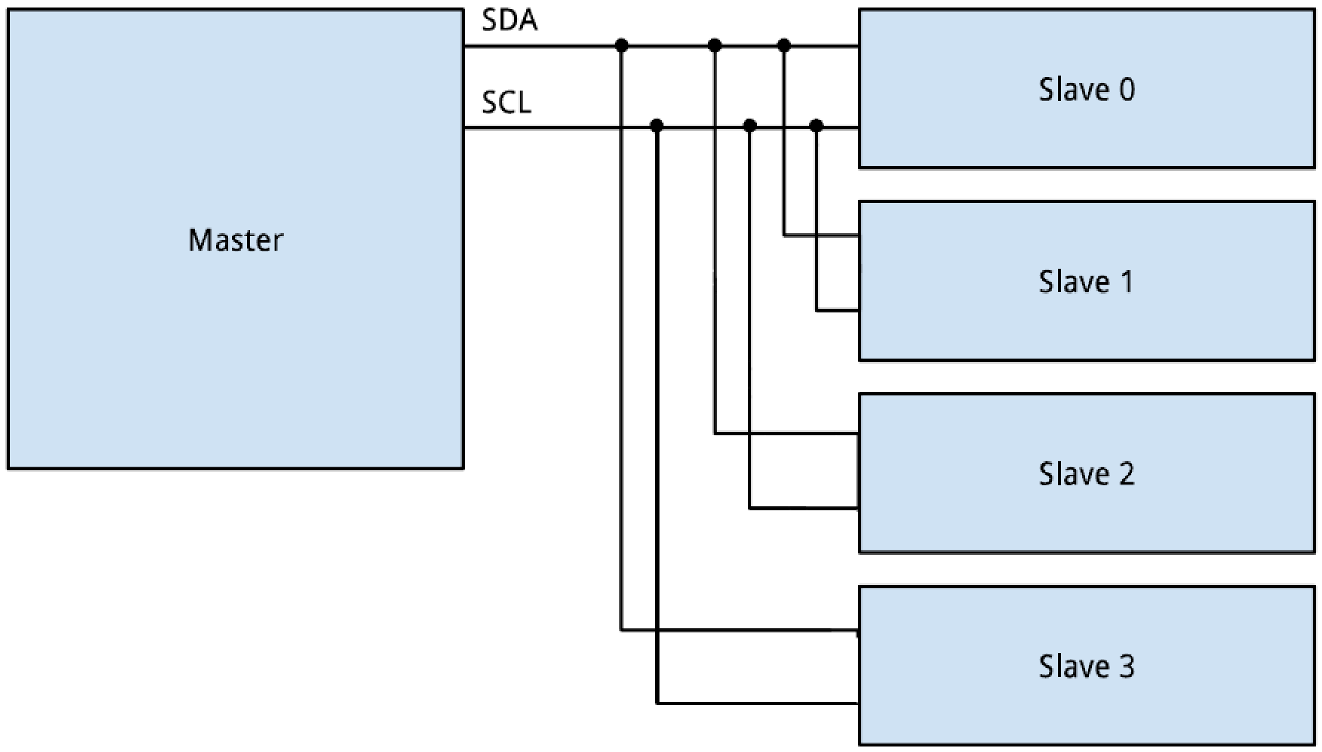
The percent error from 55555 is $(\frac{62500}{55555})\,x\,100\,\%$ ≈ 12.5%

This device is operating 8.5% faster than is supposed to and 12.5% faster than the other communication partner.

Problem 2):

Solution a):

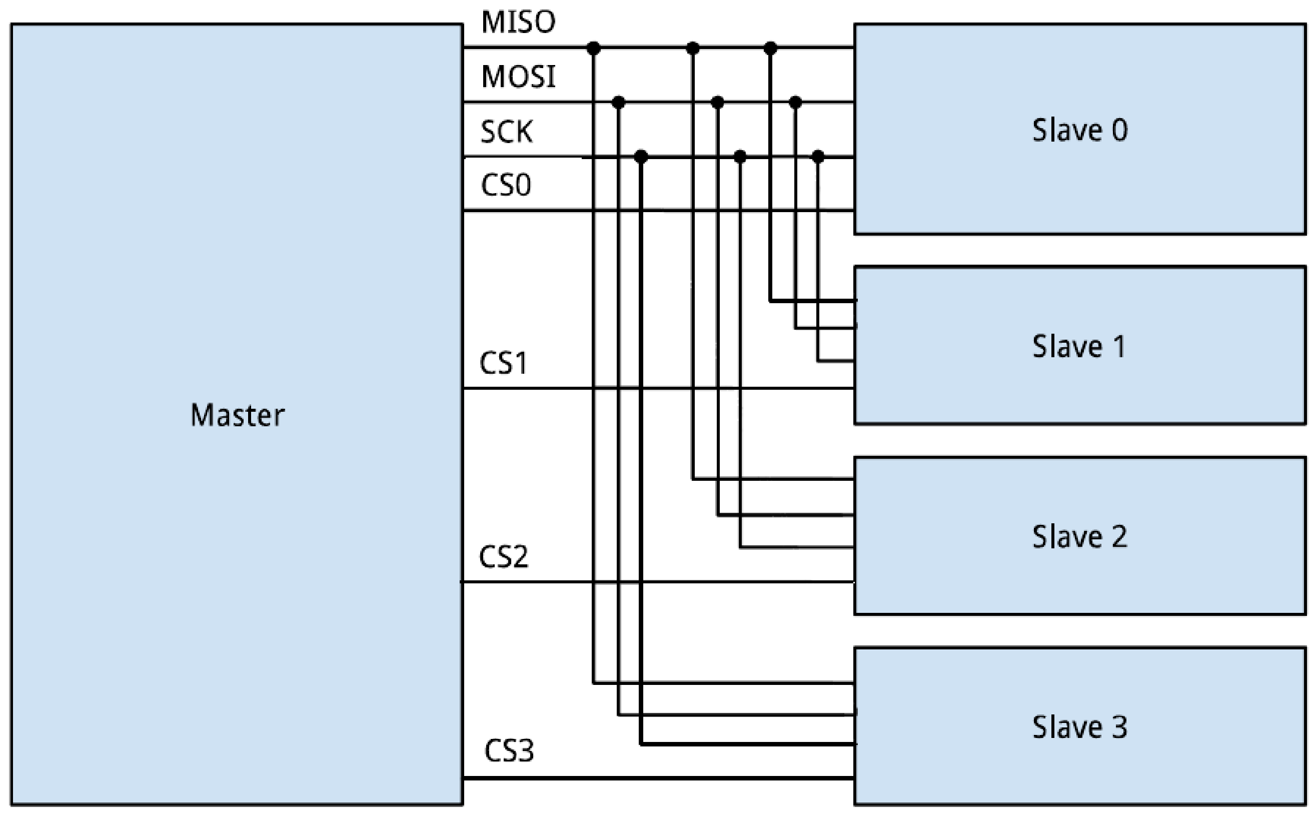
2 Wires



Solution b):

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

Solution c):



7 wires.

An SPI Transaction involves the following: a register address + read bit is sent to the slave (8 bits), and 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. About 16 bits with a duration of 1/400000 s each, for a total of 40 µs per transaction. Since we are polling four devices in sequence, each device will get to send one byte every four transaction (160 µs) for a data rate of about 50 kbps.