This lab should grant you familiarity with the system clock/oscillator, as well as basic SCI operation. We'll be using both standard tools (oscilloscope/multimeter) and Putty to confirm proper operation.

Background Info

Clock:

The built-in S08 silicon oscillator is designed to run at 4 MHz on power up. However, this oscillator is an RC circuit and the values of those components will vary (substantially) from chip to chip. The 4 MHz operating frequency can vary by ~30%, though your chip is probably a few percent off. The HCS08 has an oscillator "trim" feature so that each chip can be brought within 0.2% of 4 MHz. See Section 10.3.3 page 149 of the **Data Sheet** for details.

Trimming is accomplished by storing the desired 8-bit trim value in **ICSTRM** ($003A). The factory pre-programmed trim setting is stored in address $FFAF. Additionally, there is a FTRIM (Fine Trim) bit in the ICSSC register, which we are going to ignore (it's stored at bit0 of $FFAE).

Pro-tip: for precision timing, the right tool is a crystal oscillator – they are frequency accurate to tens of parts per million, or ~80x as accurate as even our trimmed silicon oscillator. We won't use one in lab, but they might be necessary for final projects and/or in your eventual work life.

SCI (other people call it UART):

One hardware I/O peripheral built into the S08 is a Serial Communication Interface (SCI), described on Chapter 14, page 191-210 of the *Data Sheet*. We will use the SCI to transfer data in 8-bit packets (one byte). It can also be configured to send 9 bits at a time, and/or to use one bit for parity. For this lab you will use only 8-bit mode, without parity. **Successful SCI communication requires that the sending and receiving units have agreed on the data rate, data size, and parity. Further, their respective clocks must be within 5 percent of one another**.

SCI Basics: When the SCI transmitter is **enabled**, the SCI peripheral converts bytes of data written to the SCI register (address $0027) into a sequential stream of bit values that appear as time-varying digital voltages on pin 11 (PTB1/TxD). When the SCI receiver is enabled, streams of bits received on pin 12 (PTB0/RxD) are converted into bytes available to be read from the same memory address. Between transmissions, the pin stays at the voltage level considered the **idle** state. The idle state is logic level 1, or Vdd (3.3v when using the DEMO board).

SCI Format: Data is transferred in packets containing one byte of data (weirdly, LSB first). The data byte is preceded by a start bit (logic level 0) and followed by a stop bit (logic level 1). Thus, each packet is 10 bits long. Therefore, a 9600 baud rate transfers 960 bytes/second. During an idle period, a start bit can begin at any time after a stop bit has completed (the stop bit is one bit wide). The only synchronization is the leading edge of the start bit, which is always a falling edge.

SCI Timing: In this lab we will be sending data from our S08 to a PC. The receivers in the PCs are crystal controlled so we can consider them perfect (leaving the 5% tolerance for the HCS08 clock). Getting down to 5% will require trimming for at least some S08's in the lab.

The S08's SCI's baud rate is set by storing a 13-bit value (16 bits will be stored, but the top 3 are ignored) in a pair of baud rate control registers SCIBDH ($0020) and SCIBDL ($0021). Alternatively, the pair can be viewed as a 16-bit value labeled SCIBD ($0020). You can set both

simultaneously with a `ldhx #___, sthx SCIBD` combination (where you'd put a 16-bit value into the blank). Alternatively, for small dividers, you could set just SCIBDL (SCIBDH is zero on reset). You could use `lda, sta` or just `mov #8bitval,SCIBDL`. Note mov requires direct-addressable memory, which SCIBD is.
The formula for the baud rate is **baud = bus_frequency/(16*scibd)**, where **scibd** is the 13-bit unsigned integer in the pair of registers. The bus_frequency is the system clock (usually 4MHz).

We'll use a fairly standard baud rate of 9600 baud for the lab, so you'll need a setting of 26. 4000000/(16*26) =9615, which is about .15% error (well within 5%) provided the clock speed is perfect. Note while SCI works at "any" speed, there are historical "standard" rates:
300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200,
230400, 460800, and 921600.

SCI Setup/Initialization: Aside from setting the baud rate, it's usually a good idea to turn on the transmit/receive functions (for this lab, just transmit). This requires setting the TE/RE bits in the SCIC2 (SCI Control 2) register.

SCI Use: Once properly initialized, you use the SCI by writing the SCID to transmit, or reading from SCID to get the last received byte. Since SCI communication is much slower than CPU instructions, you need to check the TDRE flag in SCIS1 before transmitting, or the RDRF flag before reading. I've provided in the shell code a one-line "wait until TDRE is set" – make sure you understand it. It uses the "*" notation introduced last lab. "*" refers to the start of the line it is on – so, for example, `bra *` would be a single-line infinite loop.

PC Communication: We will use the Windows program PuTTY (a terminal emulator) to display characters received on the PC's COM port and to send characters typed on the keyboard out the COM port. You'll need to select the proper COM port and set baud rate, parity, etc. A note: the serial port on the PC uses ±12V, with opposite polarity to our 0 and 3.3V. The DEMO board takes care of this for you, but you'll notice it if you hook up an oscilloscope and need to know about it if you build your own boards for the final project.

Bit naming: The naming convention for bit numbers is the register name followed by an underscore followed by the bit name. This bit name is equated to the bit number for use in **bset**, **bclr**, **brset**, and **brclr** instructions. You should reference the data sheet to see what these instructions do. All the register and bit names can be found on pages 41-44 and also throughout the chapters of the *Data Sheet*.

Tasks:

1. Fill out the code shell with your best efforts (can be done before class).
2. Breadboard your S08, connecting both the usual 4-pin header and the 2-pin header. Your TA will demonstrate the correct orientation and position for the 2-pin lead.
3. Start a PuTTY session with the correct COM port and 9600 baud. Make sure the COM cable as well as the USB cable is connected to the DEMO board you are using. Start your program by clicking on the debug icon. Click run and see if a never-ending stream of U's appear on the Putty screen.
    - Note: A "U" was picked as its ASCII value is $55. This is %01010101. The SCI packets consist of a start bit (0), the 8-data bits (bit 0 first through bit 7), then the stop bit (1). Thus each packet will be 0101010101 – remember the data is in the reverse order from the way we normally write it. As we are sending the character repeatedly as fast as possible the packets will be back-to-back with no gap. Thus we are producing a square wave with each half cycle 1/9615 second long. Each period is therefore 2/9615 which gives a frequency of ~4808 Hz.
4. Use either the Fluke 45 Multimeter or the oscilloscope and measure the frequency.
5. If the frequency you are reading is anywhere within 4.9% of 4808 you should be seeing the U's filling the Putty screen. Your first task is to find the ICSTRM value that gets you closest to our target frequency. The lab shell has an instruction to set it the ICSTRM to $80; you can change it to other values.
    - Hint: set a breakpoint at the sta ICSTRM instruction, click the Stop icon, click the Reset icon, click Run and when it breaks change the value in the A register to the new value you want to try. ICSTRM's value is considered to be an unsigned integer so is initially 128 in a range of 0 to 255. Increasing the value in ICSTRM increases the period so decreases the frequency.
6. **Record all the values you try and their resulting frequencies.**

7. Once the best value for producing our target frequency has been determined, try to find the fastest frequency that allows the U's to appear in the PuTTY window.

8. **Record the values you try and their resulting frequencies.** For the fastest value you were able to attain that still allowed correct communication calculate the percent error.

9. Repeat 7/8 except slow the clock down.

10. Compare the value you determined to be the best trim value to the one the factory stored at location FFAF. It should be very close.

11. Change the lda #$80 in the **init** section to lda NV_ICSTRM (this is location FFAF), comment out the U loop in main, change the **rmsg** if you like to something you want to see displayed and run the program. This time the screen should fill with lines of **rmsg**. Be sure you understand how **putstr** knows when to return (i.e. is done writing the string).

12. Calculate the correct SCIBD value to run at 300 baud. Modify the program to use that value. Change PuTTY to use that value. Rerun the program. Notice why we don't like to use 300 baud any more.