

Physics 219 – Fall, 2005
LabNotes 9

Make Your Own Instrument

Table of Contents

Make a 3-digit Decimal Display.	1
Logo code for multiplexing	3
Reflex Honing Machine: Make a reaction timer.....	5
Thanks for the memory.....	5
Logo code for reading and writing data to a 24LC64	6
Bit by bit on the serial bus	10
Make a Data Logger	11

Much the way a versatile construction kit like LEGO gives you a rich set of “primitives” that enables you a wide range of structures or mechanisms, the LogoChip is a good building block for making your own electronic instruments. One of the things that makes it a good building block is a kind of **modularity** that makes it is easy to connect it to other good building blocks. By way of example, in this lab you’ll see how to connect add a three–digit numerical display and external memory and use these to construct some simple scientific instruments.

Make a 3-digit Decimal Display.

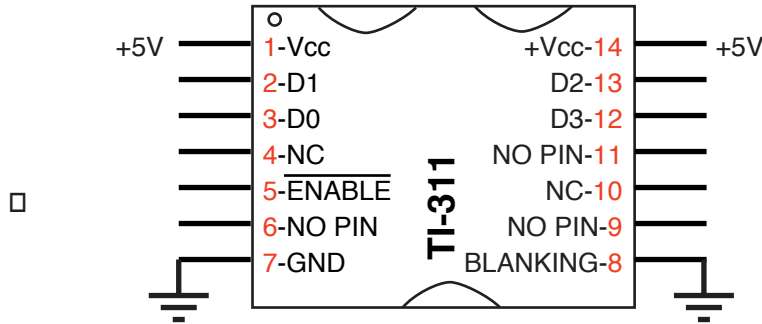
As a first step, add a three–digit numerical display to a LogoChip.

Texas Instruments (TI) Display Chips

For your display, use three TIL-311 display chips. These are **hexadecimal** (base 16) displays. Even though these displays are capable of displaying hexadecimal numbers 0 through F, in this project you will only use them to display numbers 0 through 9. These chips consume a fair amount of electrical power (~ 50 mA) so you should make sure to turn off your LogoChip when not in use if you are using batteries to power them.

The “pinout” diagram for the TIL-311 display chips is shown below: Four bits of input "data" are needed to select which of the 16 possible symbols will be displayed. The chips contain an additional digital input line called $\overline{\text{ENABLE}}$ that

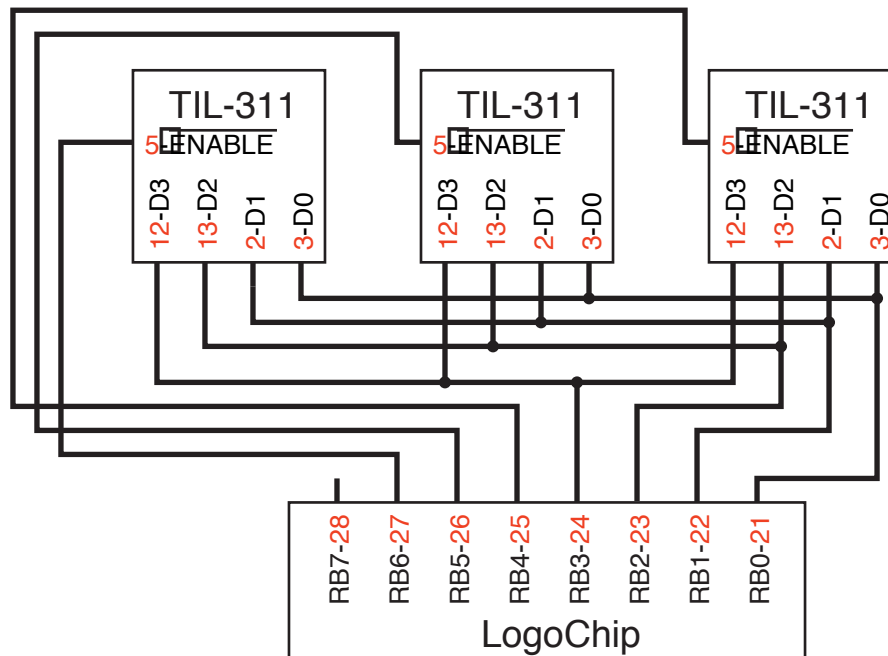
acts as follows. As long as the $\overline{\text{ENABLE}}$ line is LOW the displays will “follow” changes in the inputs. When this line goes HIGH the display stops following the input; it is “latched” to the most input that was present just before the $\overline{\text{ENABLE}}$ line went HIGH.



These displays are fairly expensive, about \$15 *each*. This is roughly 7 times the price of the microcontrollers you will use! The high price is of course due to the fact that the demand for hexadecimal displays is pretty low so they're not produced in large quantities. *The pins of these chips are fragile, so you should always use them with sockets!!*

Problem: You've got only 8 output pins available on PORTB, yet there are $3 \times 4 = 12$ input pins on the displays you will use (not even counting the $\overline{\text{ENABLE}}$ control lines). This is a common problem. No matter how many pins they put on a microcontroller you always wish you had more! What to do?

Multiplexing



Multiplexing Scheme

The answer to this common dilemma can be found in the idea of **multiplexing**. By adding “control lines” from the LogoChip that are connected to the displays’ ENABLE lines you can reduce the “cost” from 12 to 7 pins. (Additional displays can be added at the cost of one more control line per display.) The sketch above shows the basic idea. The data input lines of the displays share 4 LogoChip outputs. Numbers are written by the LogoChip to the displays one digit at a time, by placing the desired digit on the “bus” and then briefly enabling the corresponding display to “latch” the data. (The output of the displays will “follow” the input lines only while the ENABLE line is LOW; when the ENABLE line goes HIGH the output gets “frozen” into position.) Some relevant morsels of the Logo code to accomplish this are shown below.

- Try implementing the multiplexing scheme outlined above. Use the Logo code given below (and also contained in the file called display.txt that is posted in the course conference) to count from 0 to 999.

Logo code for multiplexing

Note the elegant “top-down” style in the code below. See if you can figure out how it works.

```

constants [[ones-latch 4] [tens-latch 5] [hundreds-latch
              6]]

to initialize
write portb-ddr $80           ;turn RB0 through RB6 into
                              outputs (they remain outputs
                              throughout the program)
write portb $ff              ;initially port b is all HIGH
end

to count
initialize
setn 0
loop [display n wait 1 setn n + 1]
end

to display :n
display1 :n
display10 :n
display100 :n
end

to display1 :n
write portb ((get-digit :n 1) or $f0) ;the "or" insures
                                         that only B0-B3 can change
strobe ones-latch
end

to display10 :n
write portb ((get-digit :n 10) or $f0)
strobe tens-latch
end

to display100 :n
write portb ((get-digit :n 100) or $f0)
strobe hundreds-latch
end

to strobe :n                       ;make the line go LOW briefly
clearbit :n portb
setbit :n portb
end

to get-digit :n :d
output ((:n / :d) % 10)           ;dividing modulo 10
                                   picks ;the last digit
end

```

The key line in the above code is

```
write portb ((get-digit :n 1) or $f0)
```

Can you figure out what the (bitwise) "or" is doing? An "or" logical function is "true" (equal to "1") if either (or both) of the inputs are "true". **Bit-masking** refers to the technique of using logical operations such as "or", "xor", and "and" to affect a specific set of bits within a register.

Examples:

```
portb or $f0           ;for upper bits of portb
                       high, leaving lower 4 bits
                       unchanged.

portb xor $87          ;are the contents of portb
                       equal to $87?

portb and 8            ;tests bit 3 of portb
```

- Use your displays to show a continuously updated value of an analog sensor such as a thermistor or a photocell that is plugged into one of the analog ports.

Reflex Honing Machine: Make a reaction timer.

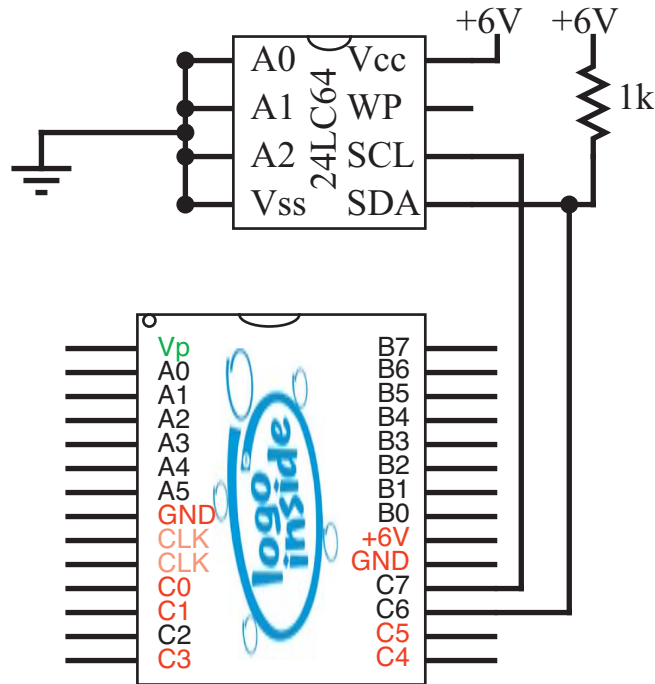
- Connect a touch sensor to pin C2 and an indicator LED to pin B7. Write a Logo program to wait a random amount of time before turning on the indicator LED, then time (using Logo's timer primitive) the amount of time, in milliseconds, it takes for you to press the touch sensor after seeing the light. Display the result on the LED displays.

Thanks for the memory

It is easy to add all sorts of "peripheral devices" to your LogoChip. Here you'll add an external memory chip that will greatly enhance the LogoChip's data storage capability.

The memory chip you will use, a Microchip 24LC64 is of the EEPROM (electrically erasable programmable read only memory) variety. It is capable of storing 8192 bytes (or 8 kilobytes) of data. EEPROMs are a "non-volatile" type of memory, which means that data remains stored in the chip when the power is turned off. A very nice feature!

- Connect a 24LC64 to a LogoChip as shown in the figure below.



Logo code for reading and writing data to a 24LC64

The Logo code below (also stored in the course conference is the file called eeprom.txt) contains the procedures needed for a LogoChip to be able to write and read data to and from a 24LC64.

To write a byte of data to the 24LC64 use

```
write-eeprom <address> <value>
```

where <address> is an integer from 0 to 8191 and <value> is an integer from 0 to 255.

- For example try

```
write-eeprom 100 57
```

This command stores the number 57 in memory location 100 in the 24LC64

To read a byte of data from the 24LC64 use

```
read-eeprom <address>
```

where *<address>* is an integer from 0 to 8191 and *<value>* is an integer from 0 to 255.

- If you try

```
print read-EEPROM 100
```

you should see the number 57 print in monitor window.

```
; data is stored on a Microchip 24LC64 8kbyte serial EEPROM
```

```
;;;;;;;;;;;;;
```

```
; EEPROM I2C protocol for use with Microchip 24LC64 8kbyte  
serial EEPROM
```

```
;;;;;;;;;;;;;
```

```
constants [[EEPROM_PORTC] [EEPROM_DDR PORTC_DDR]  
           [SCK 7] [SDAT 6]]  
global [EEPROM_ADDR EEPROM_DATA]
```

```
to powerup  
  setEEPROM_ADDR -1  
  clearbit SCK EEPROM_DDR  
  clearbit SDAT EEPROM_DDR  
end
```

```
to read-EEPROM :ADDRESS          ; read the byte located at  
                                :ADDRESS  
                                ; in the EEPROM and output it  
if (not (EEPROM_ADDR_MATCH? :ADDRESS)) ; if it's not a sequential  
                                read we need to send the address  
  [EEPROM_START                  ; send a start bit  
   EEPROM_SEND $A0               ; control byte  
   EEPROM_SEND HIGHBYTE :ADDRESS  
   EEPROM_SEND LOWBYTE :ADDRESS]
```

```
EEPROM_START                    ; send a start condition  
EEPROM_SEND $A1                 ; send current address read  
                                command  
setbit SDAT EEPROM_DDR          ; data pin to input  
setEEPROM_DATA 0  
repeat 8 [EEPROM_IN1]           ; read a byte into EEPROM_DATA
```

```
clearbit SDAT EEPROM_DDR        ; data pin back to output  
;ACK                            ; send an ACK (this seems to
```

```

                                be unnecessary....)
ee-stop                          ; send a stop condition
setee-addr :address + 1          ; increment address
output ee-data
end

to ee-addr-match? :address      ; check if ee-addr is the same as
                                the address

    ifelse (:address = ee-addr) [output 1] [output 0]
end

; read a data bit and shift it into ee-data
to eein1
    setee-data leftshift ee-data 1
    clearbit sck ee-port repeat 4 [no-op]
    setbit sck ee-port
    if testbit sdat ee-port
        [setee-data ee-data + 1]
end

to write-eprom :address :value  ; write a byte to the
                                ee-prom
    ee-start                      ; send a start bit
    ee-send $a0                   ; send the control byte
    ee-send highbyte :address
    ee-send lowbyte :address
    ee-send :value                ; then the data
    ee-stop                       ; send a stop bit
    setee-addr :address + 1       ; increment the address
    mwait 5                       ; wait 5 ms for write to finish
end

to ee-send :value                ; send the input byte out to
                                the ee-port
    setee-data :value            ; ee-data has the data
    repeat 8 [eeout1]
    ack
end

to eeout1                        ; shift a bit out from ee-data
    clearbit sck ee-port
    ifelse ee-data and 128
        [setbit sdat ee-port] [clearbit sdat ee-port]
    setbit sck ee-port
    setee-data leftshift ee-data 1

```

end

to ee-start

; send a start condition (falling
edge during sck high)

```
clearbit sck ee-port
setbit sdat ee-port
setbit sck ee-port
clearbit sdat ee-port
```

end

to ee-stop

; send a stop condition (rising
edge during sck high)

```
clearbit sck ee-port
clearbit sdat ee-port
setbit sck ee-port
setbit sdat ee-port
```

end

to ack

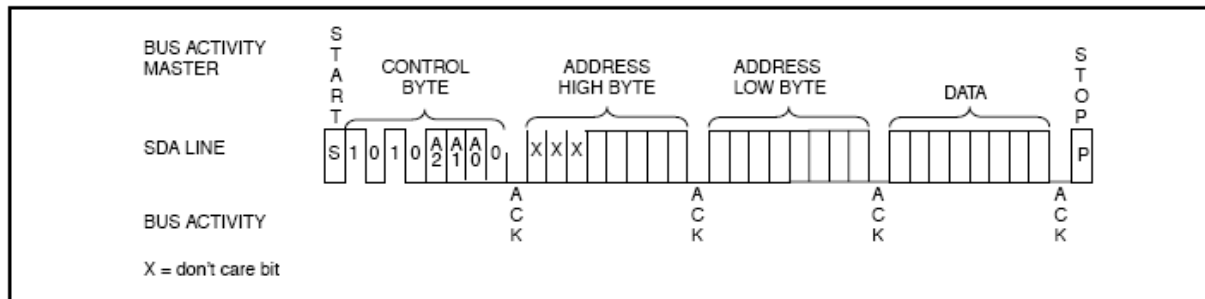
```
clearbit sck ee-port
setbit sdat ee-port
setbit sck ee-port
end
```

Bit by bit on the serial bus

The 24LC64 uses the **I²C protocol** to transfer data **serially** using only two lines. Data is transmitted as a series of 1's and 0's on one of the lines (SDAT) while the other line (SCLK) provides a periodic clock signal to indicate each time a new bit of data is on the line. In the example above, the LogoChip is “in charge” of running the clock signal, while the LogoChip and the 24LC64 take turns being in control of the data line, depending who is transmitting and who is receiving the data. (You can see this back and forth trading of who is running the data line in the Logo code by looking at the commands that set and clear bit 6 of the portc data direction register.)

The sequence of information that flows when the LogoChip is writing a byte of data to the 24LC64 is shown in the figure below. The sequence starts with a “control byte” that indicates that a write operation is going to take place. This is followed by two bytes that contain the address where the data is going to be stored, followed by the value of the byte that is to be written.

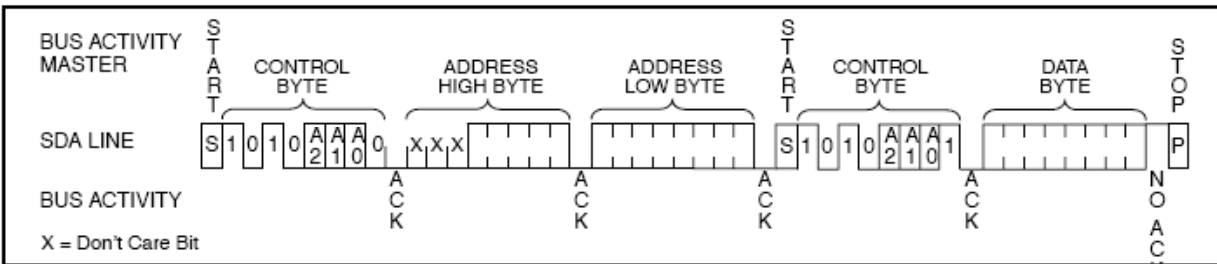
FIGURE 6-1: BYTE WRITE



- Try issuing `write-eprom` commands while using the oscilloscope to monitor both the data and clock lines. Can you connect what the you are seeing on the oscilloscope with the numbers that you are using as inputs to the `write-eprom` command?

The sequence of information that flows when the LogoChip is reading a byte of data to the 24LC64 is shown in the figure below. The sequence again starts with a “control byte” that indicates that a read operation is going to take place. This is followed by two bytes that contain the address where the data is going to be read from. This is followed by another control byte which turns control of the data line over to the 24LC64, which then transmits the value of the byte that is stored in the memory location hat has been addressed.

FIGURE 8-2: RANDOM READ



- Try issuing `print read-EEPROM` commands while using the oscilloscope to monitor both the data and clock lines. Can you connect what the you are seeing on the oscilloscope with the numbers that you are using as inputs to the `print read-EEPROM` command?

Make a Data Logger

Now that you have memory, you can make a “data logger”, capable of recording and playing back data recorded by a sensor.

- Hook up a light sensor and or a temperature sensor (made with a photocell or a thermistor) to one of the analog ports on your LogoChip.
- Write a Logo program to store the values of 100 measurements. Since the analog measurements are 10-bit numbers, you’ll have to use two bytes of memory to store them. Store the high byte of the sensor readings in even numbered memory locations and the low byte of the sensor readings in odd numbered memory locations. You should also display the sensor readings on your 3-digit display.
- Write a Logo program to “playback” your data, printing it in the monitor window. You can get make a graph of your data by copying the contents of the monitor and pasting it in Excel.
- Put your device somewhere “interesting” (e.g. a refrigerator, your pocket, your window sill) and record data for an extended period of time. To save batteries you might consider either disconnecting the 3-digit display or using LogoChip pin C2 connected to the TIL#11’s “blanking input” to turn the displays on only occasionally. (When HIGH, the display is blanked regardless of the levels of the other inputs. When LOW, a character is displayed as determined by the data in the latches. The blanking input may be pulsed for intensity modulation.)