Introduction to Computer Science and Engineering
Digital Lab #2

Introduction

In this lab, we will continue to learn more about the digital logic inside the computer that allows code to execute.

You will also learn about these concepts in CSE 2312 (Computer Organization and Assembly Language Programming), CSE 2315 (Discrete Structures), and CSE 2441 (Digital Logic Design).

Clock Generator

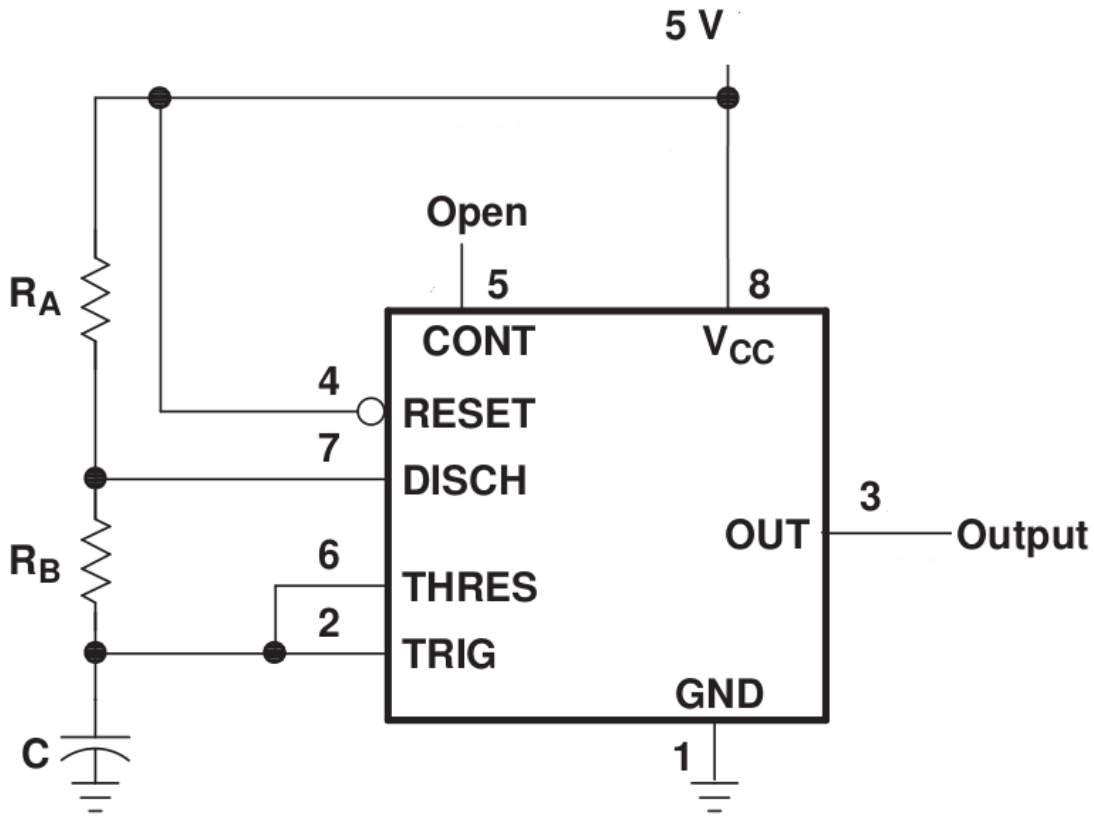
Inside a computer, a clock signal is responsible for deciding the time at which all operations occur. A clock is a periodic signal (a square wave) consisting of voltage levels that alternate between a high/true and low/false state. In modern reduced instruction set computer (RISC) machines, up to one instruction can be executed on each processor core in each clock period. Often, current clocks speeds are in the 2 to 4 GHz frequency range, meaning that the clock turns from 1 to 0 and back to 0 about 2 to 4 billion times per second, meaning 2 to 4 billion operations can are performed per second on each core.

If you had used a PC about 30 years ago, clock rates were generally below 100 MHz.

In the lab today, we will use a NE555 timer that will be configured to create clocks at approximately a 1 Hz rate (over a billion times slower than the clock in the Raspberry Pi).

The NE555 circuit to generate clocks is shown below. The chip uses only 8 pins, unlike the 14 pin parts used in the last lab. The values of the resistors and capacitors in the circuits are as follows:

Component	Value	Color Code
Ra	1kΩ	brown-black-red
Rb	100kΩ	brown-black-yellow
C	10uF	n/a

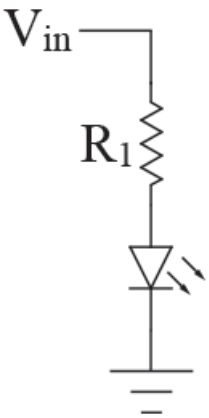


Note the + end of the capacitor is connected to pin 2 and the – end of the capacitor is connected to pin 1 (ground).

The diagram above indicates the following:

- A wire is connected between pins 2 and 6.
- A wire is connected between pins 4 and 8.
- Ra is connected between pins 7 and 8.
- Rb is connected between pins 6 and 7.
- C is connected between pin 2 and pin 1 (ground).

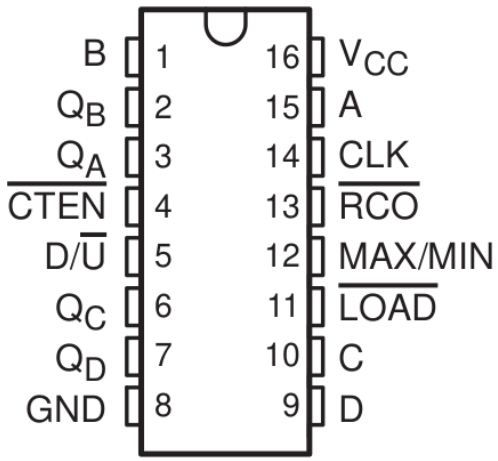
The clock is so slow that we can connect an LED to the NE555 timer output and see it turn on and off using the circuit from the last lab.



We will use this clock source to drive a binary counter.

Binary counter:

The output of the clock source above can be connected to a 4-bit binary counter (74HC191) that can either increment or decrement by one for each clock.



The overbar (sometimes ~) means the operation is enabled if the pin is low / false. So, for the D/~U pin, the count is up if low and down if high.

We will not use the A, B, C, D, ~RCO, or MIN/MAX pins.

5V and 0V should already be connected to the Vcc and GND pins. A 0.1 µF capacitor (yellow) is already connected across the power pins and acts to minimize electrical noise that causes the counter to misbehave (it will often skip zero when counting up).

The CLK input should be connected to the NE555 clock output. The ~CTEN input should be low to enable the clock input.

Again, the D/~U input is used to determine if the chip counts down (if the pin is high) or up (if pin is low).

The ~LOAD input is used to load a new value to the counter from the D:C:B:A inputs. Since this feature is not being used, make this pin high to disable this feature.

To summarize the counter's configuration:
- Pins 1, 9, 10, 12, 13, and 15 are not used.
- Pin 4 should be connected to 0V.
- Pin 11 should be connected to 5V.
- Pin 14 should be connected to the output of the clock (NE555 pin 3).

The 4-bit counter output consists of 4 outputs – Qd:Qc:Qb:Qa, where Qd is the most-significant bit (MSb) and Qa is the least-significant bit (LSb).

As the counter is set to increment, the 4-bit counter values repeats through these 2^4 = 16 binary values: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111. This modulo-16 counter next value is then x = (x+1) % 16.

This same modulo operation is observed when you write programs. Consider this C99 code, which increments the value of an 8-bit integer by one during each instruction, which operates every clock:

```
#include <stdint.h>
#include <stdbool.h>
uint8_t x = 0;
while (true)
{
     x++;
     printf("%u\n", x);
}
```

Note: `stdint.h` defines an 8-bit unsigned integer `uint8_t` data type and `stdbool.h` defines the `bool` data type and the `true/false` data values.

Since the size of the variable is fixed at 8 bits in length, there are only 2^8 possible values output by the computer as it increments the variable x repeatedly.

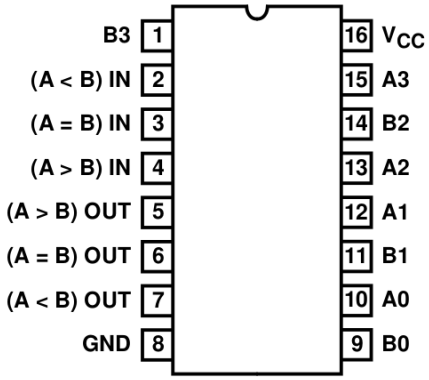
Digital Comparator

When you write programs, you often want to compare the value of two integers. Consider the following C99 code snippet that compares the value of two variables, x and y:

```
uint8_t x, y;
bool gt, eq, lt;

x = ____;
y = ____;
gt = (x > y);
eq = (x == y);
lt = (x < y);
```

Internal to the computer, two integers are compared using a digital comparator. For this lab, we will use a CD74HC85 4-bit comparator to show how this operation works. The pinout of this part is shown below:



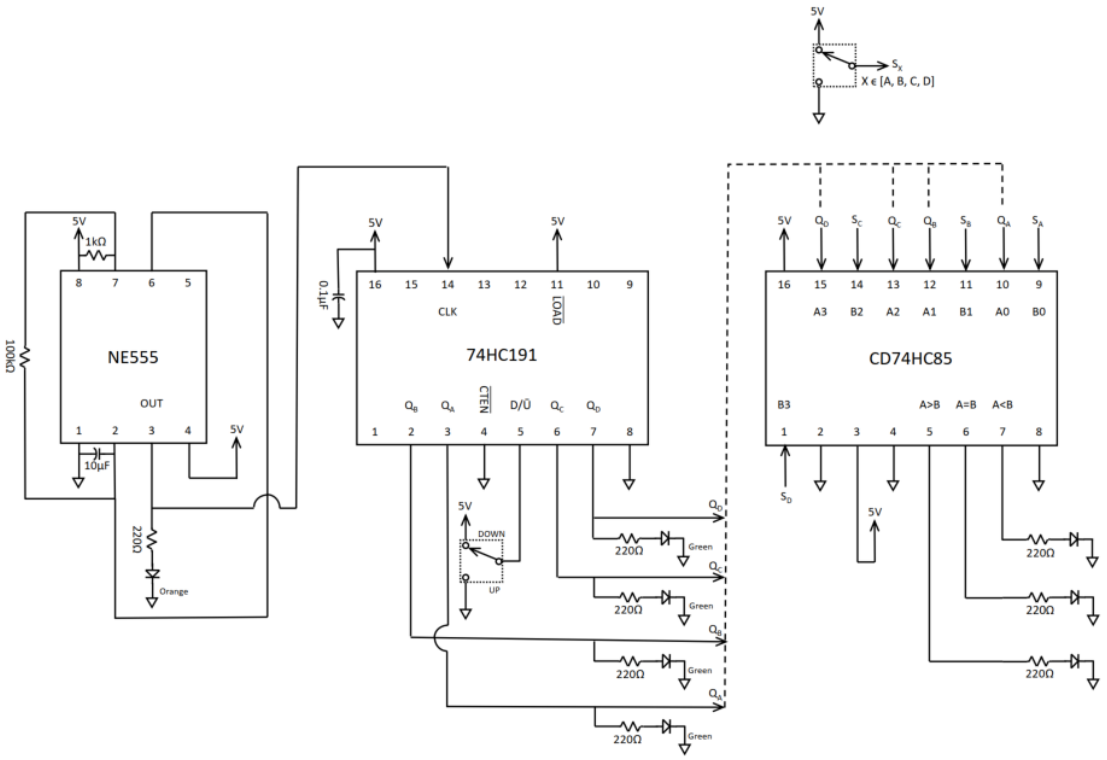
The chip should already be powered through the Vcc and GND pins. This chip compares two 4-bit numbers, A (A3:A2:A1:A0) and B (B3:B2:B1:B0).

This chip allows multiple comparators to be put in parallel to handle more bits, which require inputs from the more-significant bits. Since we are only comparing two 4-bit numbers, the (A = B) IN input should be high and the (A < B) IN and (A > B) IN inputs should be low.

The comparator outputs (A = B) OUT, (A < B) OUT, and (A > B) OUT indicate the status of these comparisons.

Complete Circuit Diagram

The following schematic shows the complete circuit for this lab:



Lab 3 Worksheet **Name _____**

Course/Section _____

Please complete the following steps to complete Lab 3:

1. Using a type A to micro B USB cable, connect the breadboard to the USB port of the computer.

2. Take the blue LED and a 220 ohm resistor (red-red-brown) in series as shown in the circuit above and connect them across the red and blue distribution strip at a point by the USB connector so it is out of the way for the next steps. It should glow brightly.

3. Configure the NE555 timer using the circuit provided above, using the resistor and capacitor values shown earlier in the document.

4. Connect the output of the timer to an orange LED and resistor in series. Count the number of times the orange LED lights up during a 1 minute interval and calculate the frequency of the clock below:

Number of flashes / minute: _____ Frequency of clock: _____ (Hz)

5. Configure the 4-bit counter using the circuit provided above, connecting the clock input to the output of the timer in step 3. Connect 4 green LEDs and 4 resistors in series to the Qd, Qc, Qb, and Qa outputs. Connect the D/~U pin to the center connection of the slide switch that is separated from the other 4 slide switches.

6. Move the slide switch so that the D/~U pin is low. Write down the LED sequence below:

Qd	Qc	Qb	Qa

7. Move the slide switch so that the D/~U pin is high. Write down the LED sequence below:

Qd	Qc	Qb	Qa

8. Configure the comparator as shown above. Connect the output of the counter (Qd:Qc:Qb:Qa) to comparator A inputs (A3:A2:A1:A0). Connect the center connection of the remaining 4 slide switches to the comparator B inputs (B3:B2:B1:B0). Connect 4 red LEDs and 4 series resistors to the output of each slide switch. Connect 3 LEDs with colors of your choosing with series resistors to the (A = B) OUT, (A < B) OUT, and (A > B) OUT.

9. Set the slide switch connected to the D/~U input so that it is low. Set the other 4 slide switches so that the B input to the comparator (B3:B2:B1:B0) is equal to 0110 (binary).

Record the value of the counter and status of the comparator outputs for each value below:

A3:A2:A1:A0	(A = B) OUT	(A < B) OUT	(A > B) OUT

10. Write a C99 program compiled with GCC on the Raspberry Pi that uses two 8-bit unsigned integer (`uint8_t`) variables named “a” and “b”. The initial value of a = 0 and b = 6. The program should contain a single for-loop that increments the “a” variable a total of 512 iterations (the counter will wrap around once). The program should output the resulting 512 values in the format below:

a	a == b	a < b	a > b
0	0	1	0
1	0	1	0

….

While the C program uses an 8-bit integer and the hardware circuit uses a 4-bit integer, the modulo nature of the counter is the same. The comparator operation is also the same.

11. Write a C99 program compiled with GCC on the Raspberry Pi that uses two 8-bit unsigned integers (`uint8_t`) variables named "a" and "b". The initial value of a = 0 and b = 6. The program should increment the "a" variable modulo 16 [a = (a+1) % 16] a total of 32 times using a for-loop (the counter will wrap around once). The program should output the resulting 32 values in the format below:

a	a == b	a < b	a > b
0	0	1	0
1	0	1	0

….

Verify that this program output matches the operation of the hardware exactly.

12. Lab checkout steps:

a. Show your working circuit and program to the grader.

b. Create a file, lastname_netid_lab3.zip, that includes the following files:

- A JPEG image of your working circuit

- The source code your program

- The output of your program

- This completed lab worksheet with all data completed

- Note: images do not need specific file names

c. Upload the zip file to Canvas.

d. After take have your picture and send the zip file, dismantle your circuit and return everything to the box on the shelf. Do not remove the USB adapter, slides switches, or the pre-loaded red and black wires.

Thank you for attending the lab. We hope some of this material was new and interesting to you.

Drs. Losh and Eary

Lab 3 Rubric

Prerequisite:
-40: Did not complete familiarization quiz on time

Assignment:
-40: No completed worksheet
-10: No image of working circuit
-20: No source code for program
-20: No output of source code
-10: Wrong file name