

Lab 5: Counter

Purpose

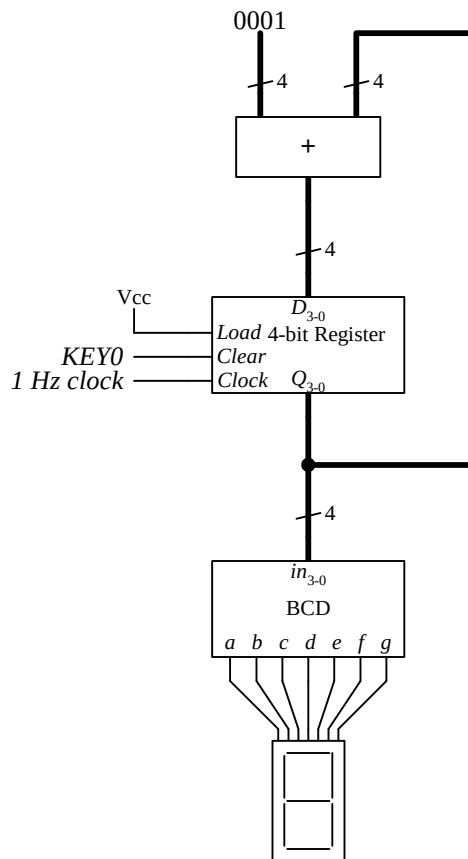
To learn how to design and implement a counter that counts from 0 to F.

Introduction

So far we have built a 4-bit adder, a 4-bit register, and a BCD converter to display a single hex digit when given a four bit binary number. We can now combine these three components to make a counter that counts from 0 to F. The count is displayed on a 7-segment LED display.

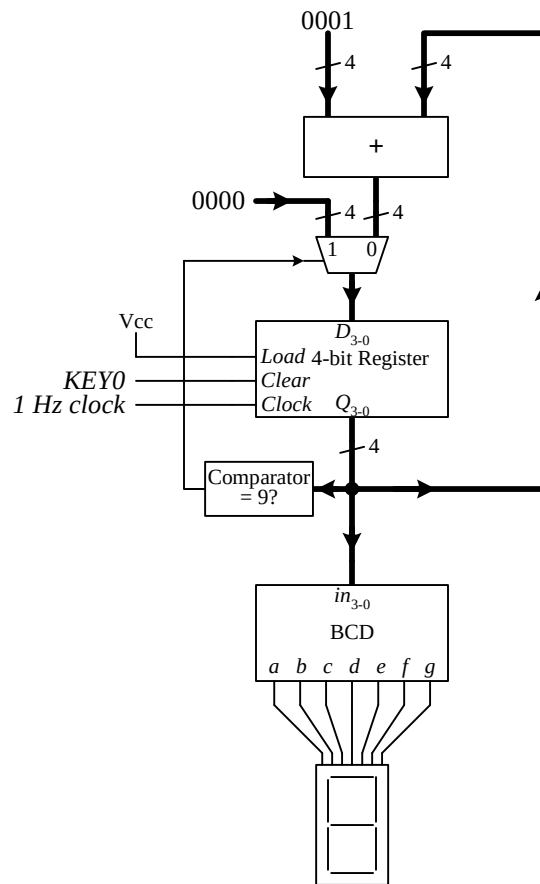
Procedure

1. The current value of the count is stored in the 4-bit register. Initially, the register is reset to 0. The 4-bit adder will take as its input the 4-bit value from the register and add a constant 1 to it. The resulting incremented value will be saved back into the register. The 4-bit output value from the register is connected to the input of the BCD which in turn is connected to the 7-segment display. You want to set the clock speed for the register to be 1 Hz and the *Load* signal to be enabled so that the count will increment every second. The circuit is shown next.



2. Modify the counter so that it will count from 0 to 9 instead of to F. You will need to add an equal-to-9 comparator that takes its input from the register so that when the count is equal to 9 it will assert the *Clear* signal to reset the count to zero.

3. What do you notice about the speed of the count between when it resets to zero and for the rest of the counts? In order to make the count timing the same, you can't use the *Clear* signal to reset the count. Instead you need to load the value zero into the register. To do this, you can use a mux to select whether you want the output from the adder or the constant zero to be loaded into the register. You will use a 4-bit 2-to-1 mux for this. The modified circuit is shown next.



Code and circuits that you will need from previous labs

Clock divider

```
module clock_divider
#(parameter [24:0] half = 25'd25000000) // 50M/2 = 1 Hz output clock
(
    input clock_in,           // 50MHz input clock
    output reg clock
);

    reg [24:0] count;

    always @(posedge clock_in) begin
        if (count == half) begin
            count <= 25'd0;
            clock <= ~clock;
        end
        else begin
            count <= count+1;
        end
    end
endmodule
```

4-bit register

