CALIFORNIA STATE UNIVERSITY SAN BERNARDINO

DEPARTMENT OF COMPUTER SCIENCE

CSCI 310 DIGITAL LOGIC LABORATORY

Component List

	100, 150, 500, 2K, and 15K Ohm Resistors
	Diode and Light Emitting Diode (LED)
	7-Segment Display
TO92	NPN Transistor
74X00	2-Input NAND Gate
74X02	2-Input NOR Gate
74X04	Invereter
74X08	2-Input AND Gate
74X10	3-Input NAND Gate
74X11	3-Input AND Gate
74X20	4-Input NAND Gate
74X21	4-Input AND Gate
74X30	8-Input NAND Gate
74X32	2-Input OR Gate
74X86	2-Input XOR Gate
74X126	Buffer
74X138	3×8 Decoder/DeMUX
74X139	2×4 Decoder/DeMUX
74X157	Quadruple 2-to-1 MUX
74X670	4×4 Register File
74X181	4-bit 48-function ALU
74X348	8×3 Priority Encoder
74X373	Latch
74X374	D flip-flop
ATF16V8	16-Input 8-Output Programmable Logic Device (PLD)

Proto-Board
Upper Left Corner

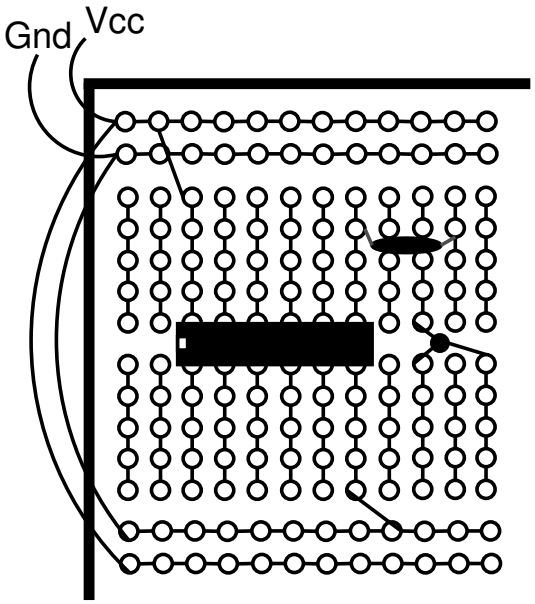


Table of Content

Lab 1: Ohm's Law, Transistors, Elementary Logic Gates

1 Resistor

Using a multi-meter (measures resistance, voltage, current, etc.), measure the resistance of two resistors, perhaps 100 and 150 Ohms (Ω). Measure the resistance of the resistors in series and in parallel. Calculate the series and parallel resistances and compare with observations. Calculate the expected currents and then observe for each resistor alone, in series, and in parallel. Resistors are color coded. There are four bands of color on a resistor. The right most band is either gold or silver and determines the accuracy, 5% or 10% respectively. The leftmost two bands determine magnitude and the middle band is the exponent value. Here are the color values:

0: black
1: brown
2: red
3: orange
4: yellow
5: green
6: blue
7: violet
8: gray
9: white

For example, if the colors from left to right are brown, red, orange, and gold on a resistor, its resistance is $12 \times 10^3\Omega \pm 5\% = 12K\Omega \pm 5\%$ accuracy.
Use the following laws:

Resistance in series: $R = R_1 + R_2$
Resistance in parallel: $1/R = 1/R_1 + 1/R_2$
Ohm's Law: $V(volts) = R(\Omega) \times I(amps)$

Data Measurements:

Resistor 1 alone: _____ Ω
Resistor 2 alone: _____ Ω
Resistors in series: _____ Ω
Resistors in parallel: _____ Ω

Verify by Calculation:

Resistors in series: _____ Ω
Resistors in parallel: _____ Ω

Predict by Calculation:

Current through resistor 1: _____ *milliamp*
Current through resistor 2: _____ *ma*
Current through resistors in series: _____ *ma*
Current through resistors in parallel: _____ *ma*

Use the multi-meter to measure the actual current for these four cases. Multi-meter must be in series with the circuit to measure current.

Data measurements:

Current through resistor 1: _____ *ma*
Current through resistor 2: _____ *ma*
Current through resistors in series: _____ *ma*
Current through resistors in parallel: _____ *ma*

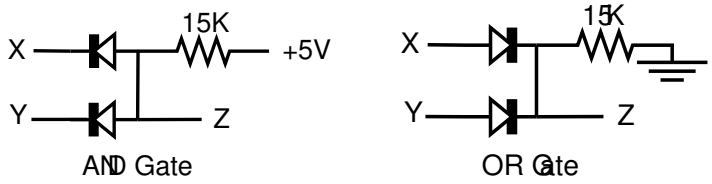
2 Diode

Measure the resistance of a diode both ways. The resistance should be very small one way and very large the other. What would an ideal diode show? It should show 0 ohm one way and largest possible resistance that multi-meter can show the other. Measure voltage across the diode in series with one resistor and in series with both resistors. Notice that voltage drop is approximately constant independent of current. Ohm's law applies to linear devices. Diodes and transistors are non-linear devices, described by no simple law, but by "characteristic curves".

Data measurements:

Diode one way: _____ Ω
Diode reversed: _____ Ω
Voltage across diode with one resistor: _____ *V*
Voltage across diode with two resistors in series: _____ *V*

Assemble a "wired" AND and OR gate using two diodes, and a resistor with about 15K Ω resistance as follows:



Connect X and Y inputs to two switches on the breadboard. Connect Z output to an LED (Light Emitting Diode) on the breadboard. Verify that the outputs match the following table.

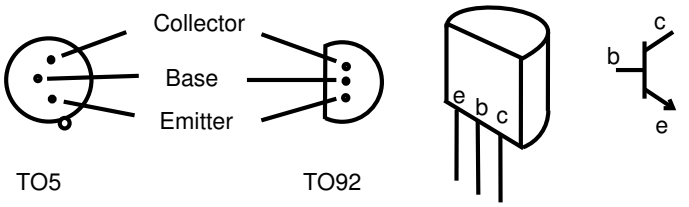
X	Y	$X\ AND\ Y$	$X\ OR\ Y$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Connect a 100 Ω resistor to an LED in series to make your own logic probe. Test the probe by connecting one end to ground potential and the other end to the output of one of the gates above.

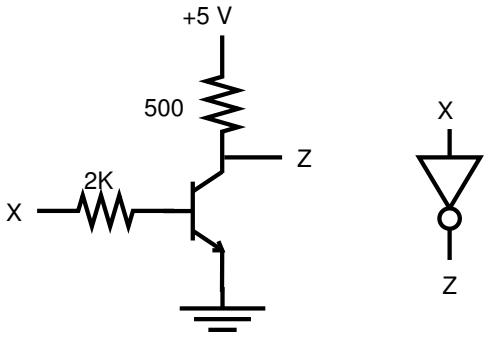
Which end must be connected to the ground?
What would happen if you use a resistor with a much larger resistance?
Demonstrate your circuits to the instructor.

3 Transistor

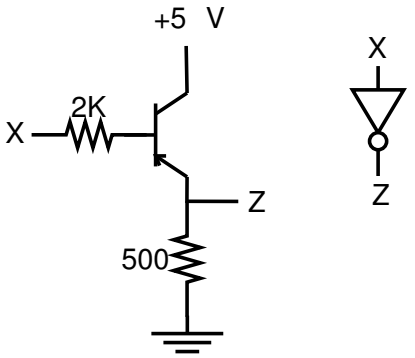
Two common packages in which NPN transistors are found are TO5 and TO92. Viewed from the bottom, i.e. with the wire leads coming out toward you, they are identified as below.



You may also use your multi-meter to determine the leads of a transistor.
Build an inverter using an NPN transistor as below.



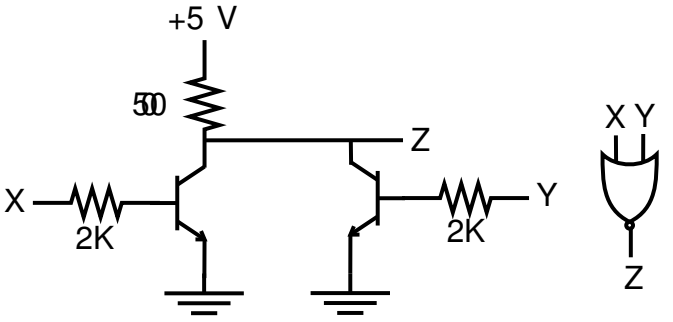
Test the functionality of your inverter. Alternatively you may use a PNP transistor as below.



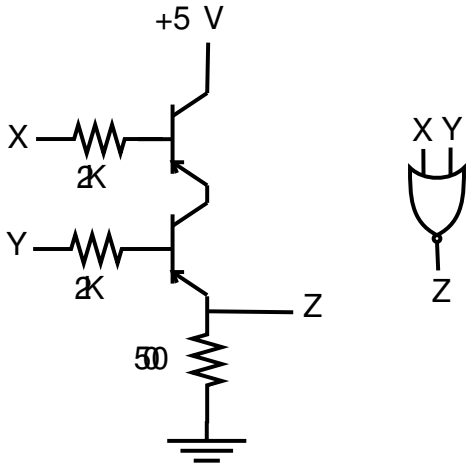
A NOR gate has the inverse functionality of an OR gate.

X	Y	$X\ OR\ Y$	$X\ NOR\ Y$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

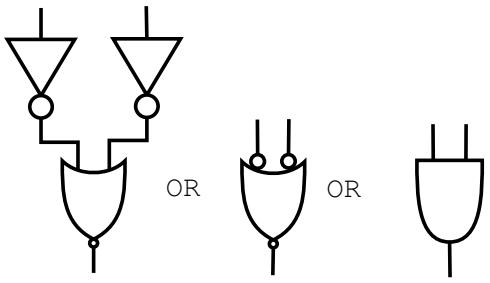
Do not break up your inverter. Build a NOR gate using NPN transistors as follows.



Alternatively you may use PNP transistors as follows.



Demonstrate your NOR gate to the instructor.
Now create an AND gate using two inverters and an NOR gate as follows.



Refer to the AND table on page 2. This design is based on DeMorgan's law:

$$\overline{\overline{x} + \overline{y}} = \overline{\overline{x}}\,\overline{\overline{y}} = xy$$

Demonstrate your AND gate functionality to the instructor.

Lab 2: Introduction to Verilog and Logic Gates

1 Preparation

1. Install VeriLogger Pro (VP) on your home computer from the CD accompanying your Mano's text. You may also use this program on the computers in the Windows lab. Verilog is an HDL (Hardware Description Language) and VP is its run time environment.

2. **Read** Section 3-9 (pages 99-106). Use a text editor to create `gates.v` to include Example 3-3 (page 103). This file consist of two modules, the stimulus (`stimcrct`) and the test circuit (`circuit_with_delay`). Stimulus is the driver for the circuit. Notice `cwd` is an instance of the circuit declared in the stimulus. The circuit appears in Figure 3-37 (page 101).
   Open VP. It opens four sub windows. Right click on the upper-left window and choose "Add HDL File(s) ...". Add `gates.v`. Compile by clicking the yellow octagon button. Make sure you don't have any errors on the lower-right window. Run by clicking on the large green right-arrow button. Enlarge the lower-left window which includes the timing diagram. Make sure the timing diagram looks like Figure 3-38 (page 104) and matches your expectation. For example, what do the gray areas represent, and why are they 30 ns and 10 ns long?
   Make sure you understand the meaning of #100 in the stimulus and #(30), #(20), and #(10) in the circuit.

3. Type in Example 3-5 (page 105) together with your own stimulus in `udp.v`. Make sure the stimulus is complete − tests all possibilities.

4. **Read** Section 4-11 (pages 147-160).
   Make sure you understand the differences among gate-level, dataflow, and behavioral modeling.
   Make sure you understand the difference between high impedance and unknown values (Table 4-9).
   Make sure you understand the functionality of three state gate.
   Make sure you understand vector declaration and use.
   Make sure you understand the syntax of `$monitor` statement.
   Type in Example 4-10 (page 160) in `analysis.v`.

2 Experiment

1. Demonstrate `gates.v, udp.v,` and `analysis.v` to the instructor. Hand in a print out of the source codes and the timing diagram and/or outputs.

2. Use the minimum number of chips, 7404 (inverter), 7408 (AND), and 7432 (OR) to build Figure 4-2 (page 113) circuit. Example 4-10 was the HDL for this circuit. Demonstrate your circuit to the instructor. Note this circuit adds. Why?

3. Refer to Figure 11-1 (page 439) and/or the following figures for the pin assignment of a few elementary IC packages.

PIN IDENTIFICATION
General:
Missing pins have no pin number.
Unconnected pins and test pins that should be left floating have no identification.
Clock signals are identified by CLK for positive-edge triggering or /CLK for negative-edge triggering.
Reset (clear) inputs are identified by RST.
Set inputs are identified by SET.
For 3-state devices, output enables are indicated by OE.
When a pin has two modes, or selects between two operations, then the two may be separated by a slash. An inversion slash may still be present as in SH//LD for shift or load select.
Pins that have more than one function (selected by programming or the state of another pin) are indicated by both functions separated by a space.
Power supply:
The main power supply is indicated by VCC.
System ground is indicated by GND.
A secondary positive power supply may be indicated by VDD.
A negative power supply is indicated by VEE.
Programming power supply (usually higher than VCC) is indicated by VPP.
Gates, line drivers etc.:
Inputs are identified by letters starting from A.
Outputs are indicated by Y.
Flip-flops:
Inputs are identified by J and K, or D.
Outputs are indicated by Q.
Counters:
Load inputs are indicated by P followed by the counter stage number.
Outputs are indicated by Q followed by the counter stage number. Thus (assuming a binary counter) Q0 is the /2 output, Q1 the /4 output.
Shift registers:
Parallel inputs or bidirectional parallel I/O pins are identified by P followed by the shifter stage number. The leftmost stage in a shift register is number 0.
Serial inputs are identified by letters starting from D (for right shift) or from L (for left shift). If more than one of either is available, the letter is followed by the shifter stage number it feeds.
Serial outputs are identified by Q, which may be followed by the shifter stage number if more than one serial output exists.
Parallel outputs are identified by Q (only if no serial outputs exist), Y (3-state outputs or output latch) followed by the shifter stage number.
Unidirectional shift registers shift towards higher stage numbers.
Bidirectional shift registers have separate shift-left and shift-right serial inputs.
Parallel-in parallel-out shift registers are called univeral.
Multiplexers:
Inputs are identified by A followed by a number.

Outputs are indicated by Y.
Select inputs are identified by S followed by a number starting at 0, unless there is only one select input in which case only S is specified. When the S inputs are taken as a binary number, the value indicates which input is selected.

Demultiplexers:
Inputs are indicated by A, preceded by a section number if more than one.
Outputs are identified by Y followed by a number. When there is more than one multiplexer section, inputs are prefixed by a number indicating to which section they belong.
Select inputs are identified by S followed by a number starting at 0, unless there is only one select input in which only S is specified. When the S inputs are taken as a binary number, the value indicates which output is selected.
For noninverting demultiplexers unselected outputs are 0, for inverting demultiplexers they are 1.

Analog multiplexers/demultiplexers:
Analog switches generally are bidirectional, and inputs and outputs can therefore be reversed. One side of the switch is indicated by X (optionally followed by a number), the other side is indicated by Y.
Select inputs are identified by S followed by a number starting at 0, unless there is only one select input in which only S is specified. When the S inputs are taken as a binary number, the value indicates which switch is selected.

Memories:
Address inputs are indicated by A followed by the bit number, starting from 0.
Multiport memories use RA or WA for separate read and write addresses, or A prefixed by the port number followed by the bit number.
Data inputs or data I/O are indicated by D followed by a number starting from 0.
Data outputs are indicated by Q followed by a number.

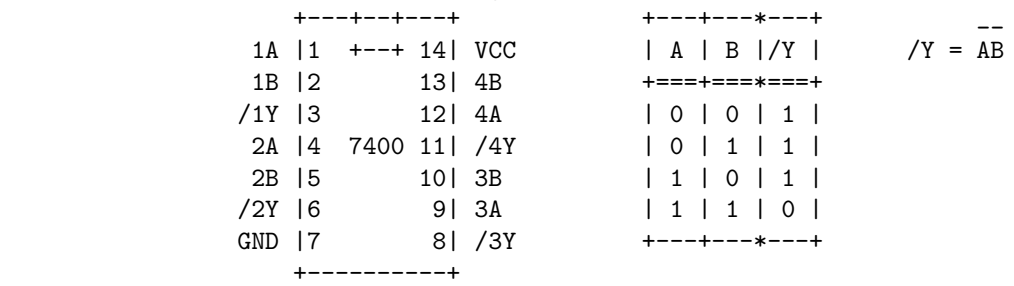
Oscillators:
One-inverter oscillators are indicated by XO and X1 pins, where XO is the inverters' output and X1 is the input. If I happen not to know which is which, the pins are indicated by X1 and X2. A crystal oscillator usually requires a crystal parallel to a 10M resistor, with two small capacitors to ground; but sometimes only a crystal is needed -- most often when a 32kHz watch crystal can be used.
Two-inverter oscillators are indicated by X1 (input), XO (middle node) and X2 (output). A crystal oscillator can then be made using XO and X1.

Sections:
When a device has several (largely) independent sections, I/O pins are prefixed by the section number, starting from 1, as in 1J or /1Q.
Multi-bit functions, such as counters or 3-state buffers have I/O pins suffixed by the bit number, usually starting from 0 (except sometimes for counters which may have some outputs missing).
The section/bit numbering is used in a different way for (de)multiplexers.

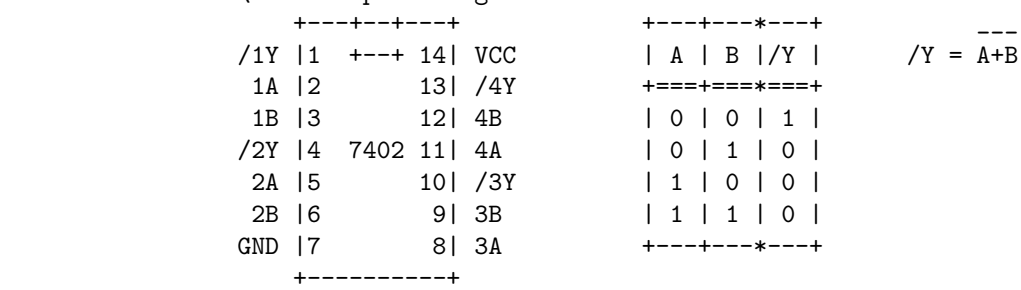
```
TRUTH TABLES
For inputs, the following notations are used:
   0  : logic low level
   1  : logic high level
   X  : don't care, either 0 or 1
   /  : rising or positive-edge clock input
   \  : falling or negative-edge clock input
   !/ : not a rising edge, either 0, 1 or \
   !\ : not a falling edge, either 0, 1 or /
   .  : 'continued', used in compressing the table
For outputs, the following notations are used:
   0 : logic low level
   1 : logic high level
   Z : high impedance, either 3-state or open-collector not driving output
   - : no change (latched in closed state, or register value not changed)
   ? : undefined (although some manufacturers may define a behaviour)
   . : 'continued', used in compressing the table
ASSUMPTIONS FOR DIGITAL DEVICES
Single/Dual Flip-flops:
  The clock is positive-edge triggered.
  Complementary outputs are available.
Multiple flip-flops:
  Only inverting or noninverting outputs are available.
Synchronous counters:
  The clock is positive-edge triggered.
  LOAD,SET and RESET are synchronous.
Asynchronous counters:
  The clock is negative-edge triggered.
  LOAD,SET and RESET are asynchronous.
Shift registers:
  The clock is positive-edge triggered.
  LOAD and RESET (if available) are synchronous.



ASSUMPTIONS FOR ANALOG DEVICES
Operational amplifiers:
  VCC usually is +15V, VEE -15V.
  For single/dual-supply op-amps, VEE can be connected to GND.
Analog multiplexers/demultiplexers:
  Analog switches generally are bidirectional, and inputs and outputs can
  therefore be reversed. One side of the switch is indicated by X (optionally
  followed by a number), the other side is indicated by Y.
  Select inputs are identified by S followed by a number starting at 0, unless
  there is only one select input in which only S is specified. When the S inputs
  are taken as a binary number, the value indicates which switch is selected.
  Select and enable inputs are at TTL level, except for 4000-series CMOS where
  these inputs must swing between GND and VCC.
```

Digtal/analog converters:
  Select and data inputs are at TTL level.
  Current output DACs may have both OUT and inverted /OUT output current pins.

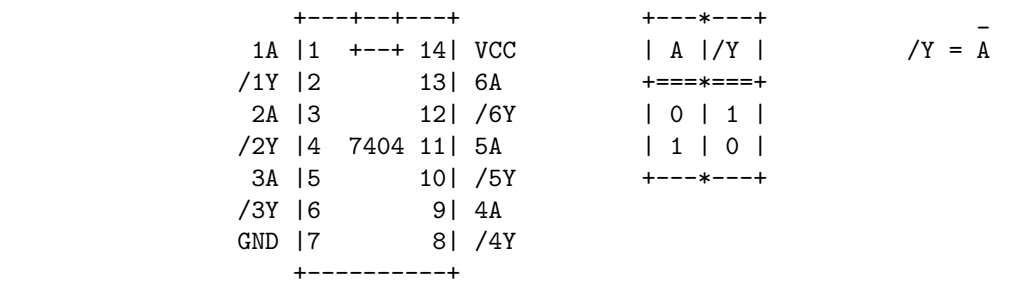
7400
Quad 2-input NAND gates.



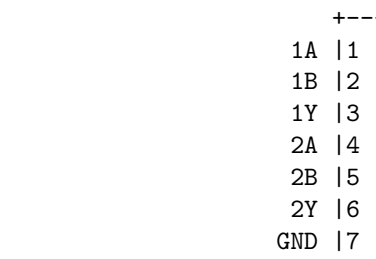
7402
Quad 2-input NOR gates.



7404
Hex inverters.



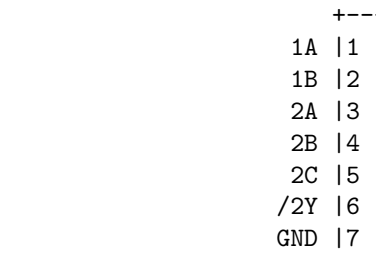
7408
Quad 2-input AND gates.



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = AB$

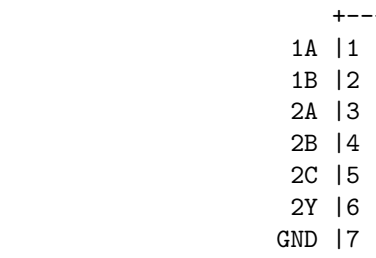
7410
Triple 3-input NAND gates.



A	B	C	/Y
0	X	X	1
1	0	X	1
1	1	0	1
1	1	1	0

$\overline{Y} = ABC$

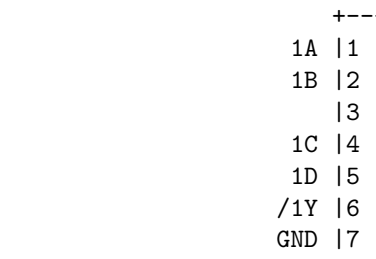
7411
Triple 3-input AND gates.



A	B	C	Y
0	X	X	0
1	0	X	0
1	1	0	0
1	1	1	1

$Y = ABC$

7420
Dual 4-input NAND gates.



A	B	C	D	/Y
0	X	X	X	1
1	0	X	X	1
1	1	0	X	1
1	1	1	0	1
1	1	1	1	0

$\overline{Y} = ABCD$

7421
Dual 4-input AND gates.

A	B	C	D	Y
0	X	X	X	0
1	0	X	X	0
1	1	0	X	0
1	1	1	0	0
1	1	1	1	1

Y = ABCD

7430
8-input NAND gate.

/Y = ABCDEFGH

7432
Quad 2-input OR gates.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Y = A+B

7486
Quad 2-input XOR gates.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = A \oplus B = (A.\overline{B})+(\overline{A}.B)$

Lab 3: Half and Full Adders

1 Preparation

1. Design a binary half-adder and a full-adder. For the half-adder we start from the following table:

X	Y	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Since C_{out} and S (Sum) are simply AND and XOR gates respectively. The logic diagram appears in Figure 4-5(b) (page 120).

2. For the full-adder we start from the following table:

X	Y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Here are the K-maps for C_{out} and S:

X \ $Y\ C_{in}$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

C_{out}

X \ $Y\ C_{in}$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

S

Let's look at S first. From the K-map we cannot further simplify the SOP, but we can find XOR operators:

$$\begin{aligned} S &= X\overline{Y}\,\overline{C_{in}} + \overline{X}\,\overline{Y}C_{in} + XYC_{in} + \overline{X}Y\overline{C_{in}} \\ &= X(\overline{Y}\,\overline{C_{in}} + YC_{in}) + \overline{X}(\overline{Y}C_{in} + Y\overline{C_{in}}) \\ &= X(\overline{Y \oplus C_{in}}) + \overline{X}(Y \oplus C_{in}) \\ &= X \oplus (Y \oplus C_{in}) \\ &= X \oplus Y \oplus C_{in} \end{aligned}$$

prove $\overline{Y}\,\overline{C_{in}} + YC_{in} = \overline{Y \oplus C_{in}}$.

Prove $X \oplus (Y \oplus C_{in}) = X \oplus Y \oplus C_{in}$ (XOR is associative).

The K-map simplification for C_{out} results in:
$C_{out} = XC_{in} + YC_{in} + XY$
We can reuse one of the XOR gates in S by simplifying C_{out}:

$$\begin{aligned} C_{out} &= C_{in}(X + Y) + XY \\ &= C_{in}((X \oplus Y) + XY) + XY \\ &= C_{in}(X \oplus Y) + C_{in}XY + XY \\ &= C_{in}(X \oplus Y) + XY(C_{in} + 1) \\ &= C_{in}(X \oplus Y) + XY \end{aligned}$$

prove $X + Y = (X \oplus Y) + XY$.

The equations for S and C_{out} correspond to Figure 4-8 (page 122) where ($C = C_{out}$ and $z = C_{in}$). Redraw this logic diagram to show that a full-adder can be built using two half-adders and an OR gate.

3. Type in Example 4-2 (page 150) which is the gate-level HDL for this half and full-adder design and a 4-bit adder. Add your own stimulus.

2 Experiment

1. Hand in a print out of Example 4-2 and its stimulus. Make sure your stimulus is complete. Demonstrate your design to the instructor.

2. Build a full-adder using two half-adders and an OR gate. Use XOR gate chips (7486), AND gate chips (7408), and OR gate chips (7432). Demonstrate your circuit to the instructor.

3. Build three more full-adders and test each one separately. In the next lab we will connect these full-adders, with some extra logic, to build a 4-bit ALU. Mark each set of input and output so that next week they are easily found.

Lab 4: Arithmetic and Logic Unit

1 Preparation

1. In the previous lab you made a full adder (FA). In this lab you will make a four-function ALU and test a 48-function ALU (74181).

2. Starting with the `_4bit_adder` module (Example 4-1) from lab 3, build a new module with mode line `M` as in Figure 4-13 (page 127). When $M = 0$, $S = A + B$. When $M = 1$, $S = A - B$. For simplicity we will not calculate overflow (V).

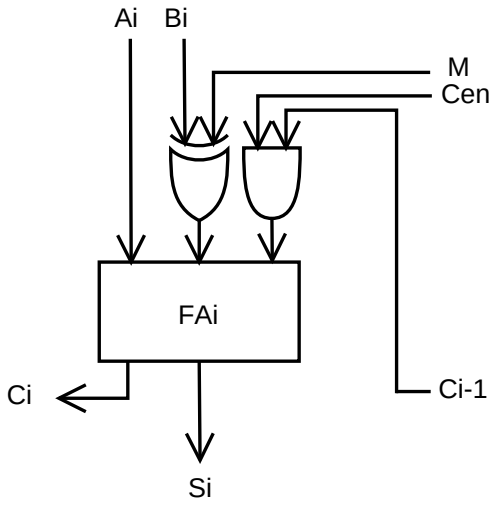
3. Add four XOR gates to `_4bit_adder` module on the path of input `B` as in Figure 4-13. Type in the following module plus the FA and HA Verilog descriptions from lab 3.

```
// name: four_bit_adder_subtracter
// desc: four bit ripple carry adder/subtracter
//       [Cout,S] = A+B if M=0      (add)
//       [Cout,S] = A-B if M=1      (subtract)
// date:
// by  :
module four_bit_adder_subtracter(S,Cout,A,B,M);
   input  M;
   input  [3:0] A,B;
   output Cout;
   output [3:0] S;
   wire C0,C1,C2;  //Intermediate carries
   wire [3:0] T;   //XOR outputs

   xor        X0(T[0], M, B[0]),
              X1(T[1], M, B[1]),
              X2(T[2], M, B[2]),
              X3(T[3], M, B[3]);

   fulladder  FA0 (S[0],C0,A[0],T[0],M),
              FA1 (S[1],C1,A[1],T[1],C0),
              FA2 (S[2],C2,A[2],T[2],C1),
              FA3 (S[3],Cout,A[3],T[3],C2);

endmodule
```

4. Test and demonstrate this design using the following stimulus.

```
// name: test
// desc: tests four bit ripple carry adder/subtracter
// date:
// by  :
module test();
  reg [3:0] a, b;
  reg m;
  wire [3:0] s;
  wire c4;

  // create instance of adder
  four_bit_adder_subtracter add_sub(s,c4,a,b,m);

  // set up the monitoring
  initial
  begin
    $display("A    B    M    C4 S     Time");
    $monitor("%b   %b   %b   %b %b    %d", a,b,m,c4,s,$time);
    end

  // run through a series of numbers
  initial
  begin
        a=4'b0000; b=4'b0000; m=1'b1;
    #10 a=4'b0100; b=4'b0000; m=1'b1;
    #10 a=4'b0100; b=4'b0011; m=1'b1;
    #10 a=4'b0100; b=4'b0011; m=1'b1;
    #10 a=4'b1100; b=4'b0011; m=1'b1;
    #10 a=4'b1100; b=4'b0011; m=1'b1;
    #10 a=4'b0100; b=4'b0000; m=1'b0;
    #10 a=4'b0100; b=4'b0011; m=1'b0;
    #10 a=4'b0100; b=4'b0011; m=1'b0;
    #10 a=4'b1100; b=4'b0011; m=1'b0;
    #10 a=4'b1100; b=4'b0011; m=1'b0;
    #10 $finish;
  end
endmodule
```

5. To build a four bit ALU, we add the carry enable line (`Cen`) and four AND gates to the previous design. In this design the carry-in line of each FA is preceded by an AND gate. One of the inputs of the AND gate is `Cen`. The other input is the `Cout` of the previous FA.



6. Complete the following module by ever so slightly modifying `four_bit_adder_subtracter`.

```
// name: four_bit_alu
// desc: four bit ALU
//       S = A+B if M=0 and Cen=1     (add)
//       S = A-B if M=1 and Cen=1     (subtract)
//       S = A^B if M=0 and Cen=0     (xor)
//       S = (A^B)' if M=1 and Cen=0  (xnor)
// date:
// by  :
module four_bit_alu(S,Cout,A,B,M,Cen);
   input  M, Cen;
   input  [3:0] A,B;
   output Cout;
   output [3:0] S;
   wire C0,C1,C2;  //Intermediate carries
   wire [3:0] T;   //XOR outputs

   xor        X0(T[0], M, B[0]),
              X1(T[1], M, B[1]),
              X2(T[2], M, B[2]),
              X3(T[3], M, B[3]);

   //Insert AND gates and FAs here

endmodule
```

7. Test and demonstrate this module by modifying stimulus `test`. Hand in a printout of your source and the timing diagram.

2 Experiment

1. Build three more FAs as in Lab 3. Connect the FAs plus four XOR gates as in Figure 4-13.

2. Demonstrate your circuit to the instructor.

3. Refer to the specification of the 74181 4-bit ALU.

```
      --  74181  --
   4-bit 16-function
arithmetic logic unit (ALU)
     +---+--+---+
  B0 |1  +--+ 24| VCC
  A0 |2       23| A1
  S3 |3       22| B1
  S2 |4       21| A2
  S1 |5       20| B2
  S0 |6   74  19| A3
 CIN |7  181  18| B3
   M |8       17| G
  F0 |9       16| /COUT
  F1 |10      15| P
  F2 |11      14| /A=B
 GND |12      13| F3
     +----------+
```

Notes

Inputs are:
[3:0]A, [3:0]B, [3:0]S, M, and CIN
Outputs are:
[3:0]F, G, P, $\overline{A=B}$, and $\overline{COUT}$
When
$M=1$ we have logic functions
$M=0$ we have arithmetic functions
Only in arithmetic is CIN significant

				M=1	M=0	
S3	S2	S1	S0		CIN=0	CIN=1
0	0	0	0	$\overline{A}$	A	$A+1$
0	0	0	1	$\overline{A\|B}$	$A\|B$	$A\|B+1$
0	0	1	0	$\overline{A}B$	$A\|\overline{B}$	$A\|\overline{B}+1$
0	0	1	1	0	-1	0
0	1	0	0	$\overline{AB}$	$A+A\overline{B}$	$A+A\overline{B}+1$
0	1	0	1	$\overline{B}$	$(A\|B)+A\overline{B}$	$(A\|B)+A\overline{B}+1$
0	1	1	0	$A\oplus B$	$A-B-1$	$A-B$
0	1	1	1	$A\overline{B}$	$AB-1$	AB
1	0	0	0	$\overline{A}\|B$	$A+AB$	$A+AB+1$
1	0	0	1	$\overline{A\oplus B}$	$A+B$	$A+B+1$
1	0	1	0	B	$(A\|\overline{B})+AB$	$(A\|\overline{B})+AB+1$
1	0	1	1	AB	$AB-1$	AB
1	1	0	0	1	$2*A$	$2*A+1$
1	1	0	1	$A\|\overline{B}$	$(A\|B)+A$	$(A\|B)+A+1$
1	1	1	0	$A\|B$	$(A\|\overline{B})+A$	$(A\|\overline{B})+A+1$
1	1	1	1	A	$A-1$	A

4. Set up the chip where input A is connected to four switches and select lines S to four other switches. Since we don't have enough switches for B, fix input B at "0011". Make sure $A0$ through $A3$ (and $S0$ through $S3$) are connected to 4 consecutive switch from right to left. Also $F0$ through $F3$ are connected to 4 consecutive LEDS from right to left. This way we can easily read the values of A, S, and F.

5. Verify several functions and demonstrate your setup to the instructor.

Lab 5: Decoder, Encoder, Code Converter

1 Preparation

1. Compare and contrast the Gate-level and Dataflow descriptions of 2×4 decoder (DEC) in HDL Example 4-1 (Page 149) and HDL Example 4-3 (Page 153) respectively. Figure 4-19 (Page 136) includes the logic diagram and the truth table for the DEC.

2. Design a 3×8 DEC using 2 instances of one of these descriptions. Draw the logic diagram of your design using block diagrams for 2×4 DECs. Supply a stimulus to test the functionality of the larger DEC. Use Table 4-6 (Page 136) to test all possibilities.

3. Hand in the logic diagram, a printout of the HDL source program, and the timing diagram.

2 Experiment

1. Setup and test the functionality of a 2×4 DEC. Use a 74X139 chip which can be used as either a DEC or De-Multiplexer (DEMUX). Note this chip contains two (dual) DECs/DEMUXs.

```
74139
Dual 1-of-4 inverting DEC/DEMUX.
     +---+--+---+            +---+---+---*---+---+---+---+
/1EN |1  +--+ 16| VCC        |/EN| S1| S0|/Y0|/Y1|/Y2|/Y3|
 1S0 |2       15| /2EN       +===+===+===*===+===+===+===+
 1S1 |3       14| 2S0        | 1 | X | X | 1 | 1 | 1 | 1 |
/1Y0 |4   74  13| 2S1        | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
/1Y1 |5  139  12| /2Y0       | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
/1Y2 |6       11| /2Y1       | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
/1Y3 |7       10| /2Y2       | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
 GND |8        9| /2Y3       +---+---+---*---+---+---+---+
     +----------+
```

2. Test the functionality of one of the DECs. Build a 4-output, 2-bit DEMUX by grouping the corresponding outputs of the two DEMUXs. Connect the corresponding select (S?) lines to the same input switch. Connect each Enable line to its own input switch. Demonstrate your circuit to the instructor.

3. Use 2 DECs (1 chip) to build a 3×8 DEC. Note Enable (EN) and Output (Y?) lines are active low in 74X139 (they are preceded by a / representing a bar). Demonstrate your circuit to the instructor.

4. Consider the pin assignment and the truth table of 74X138 3×8 DEC/DEMUX.

```
74138
1-of-8 inverting DEC/DEMUX.
     +---+--+---+           +---+----+----+---+---+---*---+---+---+---+---+
  S0 |1  +--+ 16| VCC       |EN1|/EN2|/EN3| S2| S1| S0|/Y0|/Y1|...|/Y7|
  S1 |2       15| /Y0       +===+====+====+===+===+===*===+===+===+===+===+
  S2 |3       14| /Y1       | 0 | X  |  X | X | X | X | 1 | 1 | 1 | 1 |
/EN3 |4   74  13| /Y2       | 1 | 1  |  X | X | X | X | 1 | 1 | 1 | 1 |
/EN2 |5  138  12| /Y3       | 1 | 0  |  1 | X | X | X | 1 | 1 | 1 | 1 |
 EN1 |6       11| /Y4       | 1 | 0  |  0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
 /Y7 |7       10| /Y5       | 1 | 0  |  0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
 GND |8        9| /Y6       | 1 | 0  |  0 | . | . | . | 1 | 1 | . | 1 |
     +----------+           | 1 | 0  |  0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
                            +---+----+----+---+---+---*---+---+---+---+---+
```

As in 74X139, connect pin 6 (EN1 port) of the DEC to a switch and see what happens to the functionality of the DEC/DEMUX when the switch is turned on and off. As an example of the functionality of this DEC, when active high 001 is selected on the inputs, all outputs are high except for output line 1 (pin 14).

5. Consider the pin assignment and the truth table of 74X348 8×3 priority encoder (ENC).

```
74348
8-to-3 line 3-state inverting priority ENC with cascade inputs.
    +---+--+---+          +---+---+---+---+---+---+---*---+---+---+---+---+
/A4 |1  +--+ 16| VCC      |/EI|/A0|/A1|...|/A5|/A6|/A7|/Y2|/Y1|/Y0|/GS|EO |
/A5 |2       15| /EO      +===+===+===+===+===+===+===*===+===+===+===+===+
/A6 |3       14| /GS      | 1 | X | X | X | X | X | X | Z | Z | Z | 1 | 1 |
/A7 |4   74  13| /A3      | 0 | H | H | H | H | H | H | Z | Z | Z | 1 | 0 |
/EI |5  348  12| /A2      | 0 | X | X | X | X | X | 0 | 0 | 0 | 0 | 0 | 1 |
/Y2 |6       11| /A1      | 0 | X | X | X | X | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
/Y1 |7       10| /A0      | 0 | X | X | X | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
GND |8        9| /Y0      | 0 | X | . | . | 1 | 1 | 1 | . | . | . | 0 | 1 |
    +----------+          | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
                          +---+---+---+---+---+---+---*---+---+---+---+---+
```

Note Z means high-impedance (as good as disconnected).

6. Test the functionality of the DEC/DEMUX by connecting the three inputs to three switches and the 8 outputs to 8 LEDs on the bread-board. What is the input port of the DEMUX?

7. Do not disassemble your DEC setup. Demonstrate the functionality of the ENC on a different area on your bread-board. Why is this called a *priority* ENC?

8. Connect output ports of the DEC to the input ports of the ENC to realize the gray code table:

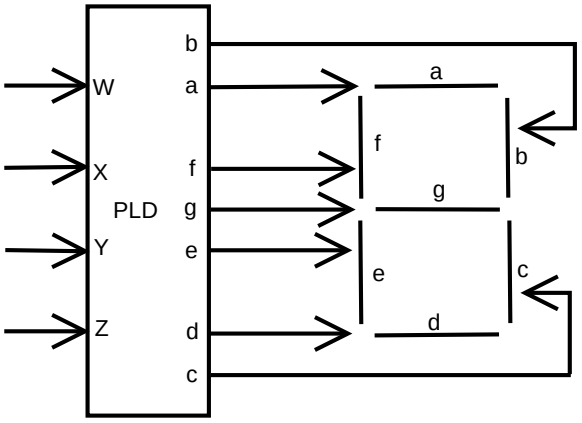
Input (Rank)	Output (Gray Code)
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Since outputs of the ENC are active-low, you need to use three inverters on its output lines. Demonstrate your circuit to the instructor.

Lab 6: HEX 7-Segment Code-Converter

1 Preparation

1. In this lab we will program and use a Programmable Logic Device (PLD) to display hexadecimal digits: 0, 1, ..., A, b, C, d, E, and F. The PLD or code-converter has 4 input lines (W, X, Y, Z) and 7 output lines (a, b, c, d, e, f, g).



2. Develop your input-output table with your lab partners. Individually, simplify each output column (a, b, ..., g) using a K-map. Double check your results with your partners. Do not simplify beyond the SOPs derived from K-maps.

3. Based on your SOPs type-in and save the following file on a diskette. Save the file with a `.PLD` extension. Bring the diskette and a printout of the file to the lab. Here `&,#,` and `!` represent AND, OR, and NOT respectively.

```
PARTNO       PLD01;
NAME         CODECONV;
DATE         today's date;
REV          01;
DESIGNER     your name;
COMPANY      CSUSB;
DEVICE       G16V8A;
ASSEMBLY     BREADBOARD;
LOCATION     JBH-356;

/* INPUTS */
PIN 1 = W;
PIN 2 = X;
PIN 3 = Y;
PIN 4 = Z;

/* OUTPUTS */
```

```
PIN 12 = a;
PIN 13 = b;
PIN 14 = c;
PIN 15 = d;
PIN 16 = e;
PIN 17 = f;
PIN 18 = g;

/* BOOLEAN FUNCTIONS */
a = !X&!Z # Z&W # Z&X # Y&!X;
b = ...;
.
.
.
```

2 Experiment

1. Use the WINCUPL program on the lab machines to compile your PLD program. Under the compiler options use `JED name = PLD name`. Perform a device dependent compile. This will create a JED file. Save the JED file on your diskette.

2. Run Chip Master 5000 program. Choose device ATF16V8B. Lock in your IC chip on the programmer control box. Erase the chip. Load your JED file. Program the chip.

3. Your PLD pin assignment is as follows.

```
ATMEL ATF16V8B PLD
   +---+--+---+
   |1  +--+ 20| VCC
I  |2       19|
N  |3       18|
P  |4       17| INPUT/
U  |5       16| OUTPUT
T  |6       15| PORTS
S  |7       14|
   |8       13|
   |9       12|
GND|10      11| /OE
   +----------+
```

4. Connect PIN 1-4 to 4 switches from right to left.

5. Connect PIN 12-18 to the 7-Segment display on the proto-board.

6. Test the functionality of the code converter. Demonstrate your design to the instructor.

7. Hand in a print-out of the PLD file and your K-map simplifications.

Lab 7: Verilog Simulation and Timing

1 Preparation

One of the main advantages of using a Hardware Description Language (HDL) like Verilog is the ability to simulate timing and performance of a circuit and work out any problems quickly before fabricating. In this lab we will be looking at the basic techniques of how this is done.

1. Use the VeriLogger Pro software that came with your book to do the following.
   (a) If you have not done so already, install Verilogger Pro.
   (b) Launch Verilogger Pro.
   (c) Under the "Project" tab, select "Add File(s)..." and add the files you created for Lab 3 and Lab 4. They should appear in the "Project" window and show you all the modules that are defined in them.
   (d) Press the green play arrow. VeriLogger will automatically check your syntax, compile, and run if no errors are found. If it runs you will see your signals automatically plotted in the "Diagram" window.
2. Add gate delays by adding "parameter delay=0" to the top of each module with gates, which sets the default value to be zero (no delay). You can edit a module by double clicking its name in the "Project" window. We set a clock parameter because it allows us to easily change it later when we need. Parameters can even be changed when we instantiate them by placing a "#(value)" between the module name and the instance name when instantiating. At each gate declaration modify them so that you pass the time delay to them by adding a "#(delay)" before the gate name (see HDL Example 3.2 in the book). For example an xor gate would now look like "xor #(delay) x0(T[0], M, B[0]);". The delays are used by the simulator to see how long it takes for the signal to propagate through the circuit. We can graph the signals over time and thus see what is happening in any system we design. Make sure you modify all the following modules.
   - halfadder
   - fulladder
   - four_bit_adder_subtractor
   - four_bit_alu
3. Run the Verilog code. It should produce the same results since the delay is zero.
4. Modify the test module for the four bit alu so that the instantiation is now "four_bit_alu #(5) alu(s,c4,a,b,m,cen)" and run it. What happens and why?
5. Modify the delay a few times and see if you can predict what will happen each time. How long does it take to get the solution? How long is that in terms of gate delays? Can you express it as a formula?

2 Experiment

In this lab we will be timing a simple computing device using the ALU we designed before.

1. Create a module to contain our simple computer.
2. Add two four bit registers named "ACC" and "Op2".
3. Now make two registers to hold the signals "sub" and "mode".
4. Next create four wires called "result" and a single wire called carry.
5. Make an instance of the adder-subtractor and pass the registers and wires you created to it.
6. Just like you did for the test units create an initial unit and set the values of the registers to
   - ACC=0
   - Op2=3
   - sub=0
   - mode=1

   and setup a "$monitor" command to track the registers and wires.
7. Make a parameter called "wait" and set its value to the time you calculated in the preparation to get the solution.
8. Then make an always unit to control the flow of data in the computer. This essentially tells the accumulator to load the result of the alu.

```
always begin
    #(wait) ACC=result;
end
```

9. Run the computer. What does it do? Show the output to the instructor.
10. Set "wait" to twice its value. Does it still give the correct results? Why or why not?
11. Now set "wait" to half its initial value. Does it still work? Why or why not?

Lab 8: MINI Register File

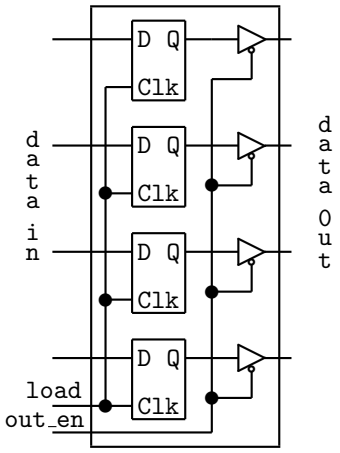
1 Preparation

In this lab you will be making the register file (memory) for the Mini. In the preparation you will be designing the register file in Verilog. First read section 5-5 in the book (pages 190-198). The registers in the Mini each hold one nibble (half a byte, i.e.: four bits). The register file is made up of four registers. We will design our register file in four steps:

1. **Create a D flip-flop**

   A D flip-flop must hold 1 bit of data, and it only changes its data when the clock changes. We want a positive edge triggered flip-flop. Enter the D flip-flop, "D_FF" from example 5-2 on page 192 of the book.

2. **Make a four bit register with D flip-flops**



   Create a module to hold our four bit register. Just like the picture.

```
// name: Nibble_Reg
// desc: four bit register with output enable (low),
//        made from D flip-flops
// date:
// by  :
module Nibble_Reg(data_out,data_in,load,out_en);
  input  [3:0] data_in;
  input        load,out_en;
  output [3:0] data_out;

  // wires between flip-flops and tri-state gates
  wire   [3:0] dff_out;

  // instantiate tri-state gates to do output enable
```

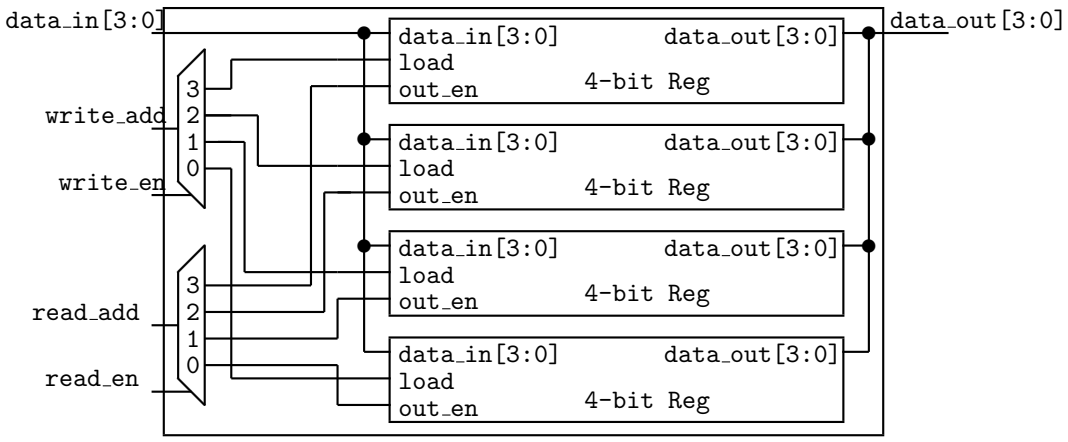
```
bufif0 tri3(data_out[3],dff_out[3],out_en);
bufif0 tri2(data_out[2],dff_out[2],out_en);
bufif0 tri11(data_out[1],dff_out[1],out_en); // tri1 is a gate type
bufif0 tri00(data_out[0],dff_out[0],out_en); // tri0 is a gate type


//instantiate D flip-flops here
D_FF Reg_Bit_3(dff_out[3],data_in[3],load);

   // finish making instances

endmodule
```

3. **Create a 2 to 4 line decoder**

   We will need two decoders in the final step of our design so we will create them now. Enter the 2 to 4 line decoder, "decoder_df" from example 4-3 on page 153 of the book. To follow standard design practices we will make a few modifications.

   - Put "D" first in the port list. As a general rule, outputs are always first.
   - The ports "A" and "B" are actually the address bits so combine them into one new port "A" that has two bits. Note you will have to change the port list, input line, and the assignments.
   - Change the bit ordering of "D" from "[0:3]"(big endian) to "[3:0]"(little endian) to be consistent with the rest of the design

4. **Build the register file from the registers**



Create a module to hold our register file. Just like the picture

```
// name: Register_File
// desc: 4x4 register file
```

```
// date:
// by  :
module Register_File(data_out,data_in,read_add,read_en,write_add,write_en);
  input  [3:0] data_in;            // data to write
  input  [1:0] read_add,write_add; // read address and write address
  input        read_en,write_en;   // read and write enable
  output [3:0] data_out;           // data to read

  wire   [3:0] read_sel,write_sel;

  //instantiate registers here
  decoder_df Dec_Read(read_sel,read_en,read_add);
  decoder_df Dec_Write(write_sel,write_en,write_add);


  //instantiate registers here
  Nibble_Reg Reg_0(data_out,data_in,write_sel[0],read_sel[0]);

     // finish making instances

endmodule
```

2 Experiment

1. Hand in a print-out of your complete Verilog code and the timing diagram for a sample run.

2. Design a simple accumulator machine. We will use the 670 4x4 register file, 181 ALU, and 374 D flip-flops as the basis of our design. Consult the pin out diagrams. The 670 is a four location register file with four bits per location. We only want to use addition and subtraction.

3. First, wire the 181 so you can switch from addition to subtraction with one of the logic switches. See the 181 data sheet for pin information.

4. Wire the 670 register file as follows

   (a) Build a 4-bit accumulator (ACC) using 374 very similar to the preparation, except that we will not use the tri-state gates. Simply setup the power and ground pins, connect the only CLK pin to a debounce switch, Ground pin 1 to enable output, and identify 4 out of 8 flip-flops to work with. Note that this chip contains positive-edge triggered D flip-flops all derived by a single clock (pin 11). Initialize ACC to 0, how?

```
74374
8-bit 3-state D flip-flop.
    +---+--+---+             +---+---+---*---+
/OE |1  +--+ 20| VCC         |/OE|CLK| D | Q |
 Q1 |2       19| Q8          +===+===+===*===+
 D1 |3       18| D8          | 1 | X | X | Z |
 D2 |4       17| D7          | 0 | / | 0 | 0 |
 Q2 |5   74  16| Q7          | 0 | / | 1 | 1 |
 Q3 |6  374  15| Q6          | 0 |!/ | X | - |
 D3 |7       14| D6          +---+---+---*---+
 D4 |8       13| D5
 Q4 |9       12| Q5
GND |10      11| CLK
    +----------+
```

(b) Hook the address lines of the 670 together and connect them to two switches on the development board. (Connect RA0 and WA0 to switch 0. Connect RA1 and WA1 to switch 1.)

(c) Wire four other logic switches (2 through 5) to the Din of 670.

(d) Ground read enable ($\overline{RE}$) and connect write enable ($\overline{WE}$) to another debounce switch. Use inverted output, why?

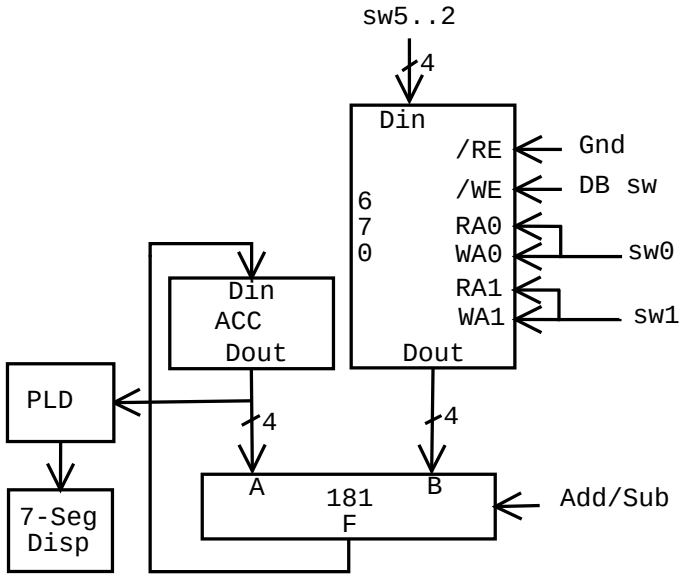
Store integers 2, 3, 7, and 5 in locations 0, 1, 2, and 3 respectively. How do you make sure that the correct values are stored?

(e) Connect the outputs of the 181 to the data inputs of the ACC.

(f) Connect the data outputs from the 670 to the "B" input of the 181.

(g) Connect the outputs of the ACC to the "A" inputs of the 181.

(h) Also connect the output from the ACC to your display from lab 6 so you can read its content.



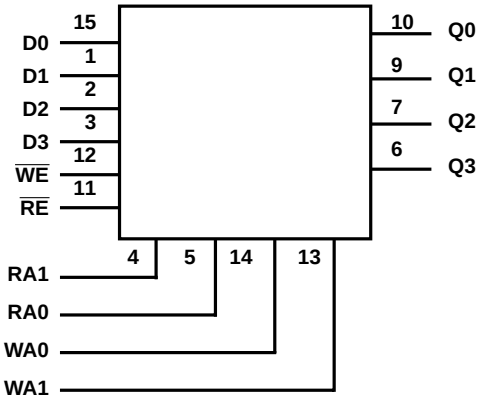
5. The simple computer you just made adds (or subtracts) the value of the four 4 registers in 670 in ACC. The accumulator loads when you press the debounce switch connected to its clock input. Be careful not to double press the clock as this will perform two (or more) operations. Don't forget to set the address of the word in 670 before pulsing the ACC clock. Fill-in the following table.

		A	A
B	Operation	Predicted Result	Circuit Result
2	+	2	
3	+		
7	−		
5	+		

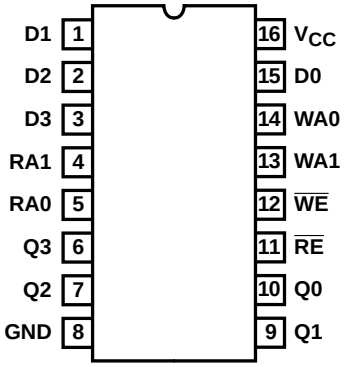
Show your design to the instructor.

DO NOT DISASSEMBLE. We will use this circuit in Lab 10.

74X670 Block Diagram, Pin Assignment, and Table



CD74HC670, CD74HCT670
(PDIP, SOIC)
TOP VIEW



WRITE MODE SELECT TABLE

OPERATING MODE	INPUTS		INTERNAL LATCHES (NOTE 3)
	$\overline{WE}$	D_N	
Write Data	L	L	L
	L	H	H
Data Latched	H	X	No Change

NOTE:

3. The Write Address (WA0 and WA1) to the "internal latches" must be stable while $\overline{WE}$ is LOW for conventional operation.

READ MODE SELECT TABLE

OPERATING MODE	INPUTS		OUTPUT Q_N
	$\overline{RE}$	INTERNAL LATCHES (NOTE 4)	
Read	L	L	L
	L	H	H
Disabled	H	X	(Z)

NOTE:

4. The selection of the "internal latches" by Read Address (RA0 and RA1) are not constrained by $\overline{WE}$ or $\overline{RE}$ operation.

H = High Voltage Level
L = Low Voltage Level
X= Don't Care
Z = High Impedance "Off" State

Lab 9: MINI Control Unit

1 Preparation

In this lab we will build a 3-bit counter to be used as the control unit for MINI − a simple CPU. This CPU will be very similar to the simple computer built in lab 7. We will built the CPU in the next lab and use the PLD from lab 6 to display the result of the operations.

1. Examine the pin-out diagram of the 74X374 from lab 7.

   Note all 8 flip-flops are triggered by a single clock pulse on pin 11. Can one build a ripple counter from a 74X374? Why not?

2. Design a 3-bit binary counter that counts from 0 to 7 and repeats, using 74X374. These are positive-edge triggered D flip-flops, but that does not matter here, why?

3. Start by filling the following table:

			D_2	D_1	D_0
$Q_2(t)$	$Q_1(t)$	$Q_0(t)$	$Q_2(t+1)$	$Q_1(t+1)$	$Q_0(t+1)$
0	0	0	0	0	1

4. Next, draw Karnaugh maps to find boolean expressions for each D input in terms of $Q_2(t)$, $Q_1(t)$, and $Q_0(t)$. Although the SOPs derived from the K-maps are optimal in terms of circuit timing, try to find XOR gates in the resulting expressions.

2 Experiment

1. Hand in your work from the Preparation section.

2. Build an SR latch using 4 NAND gates (Figure 5-5(a), page 171). Demonstrate your circuit to the instructor. Do your observations agree with what you expect (Figure 5-5(b))?

3. Build the counter you designed in the Preparation section without disassembling the circuit from the previous lab. We will use it in the next lab as the control unit for MINI.

4. Connect the output of the flip-flops to your PLD and the 7-segment display. What happens to the W input of the PLD? Demonstrate your circuit to the instructor.

5. How would you modify the state table and then the circuit to count to 4 back to 0 etc.? Demonstrate your circuit.

6. How would you modify the state table and then the circuit to count *from* 0? Why is this impossible?

7. Use a switch (input X in the state table) to start counting from 0 to 7 and maintain 7 when X = 1. When X = 0, the circuit reset itself to 0.

X	$Q_2(t)$	$Q_1(t)$	$Q_0(t)$	D_2 $Q_2(t+1)$	D_1 $Q_1(t+1)$	D_0 $Q_0(t+1)$
0	X	X	X	0	0	0
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Demonstrate your circuit.

Lab 10: MINI Machine Organization

1 Preparation

In this lab we will put together what we designed in the previous two labs to add the content of each word in the register file to the accumulator *automatically*.

Here is another way of looking at the state table of your 3-bit counter with the enable switch X from lab 9:

X	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1
1	1	1	1
⋮	⋮	⋮	⋮

Note Q_0 oscillates between 0 and 1 when the counter counts, therefore it generates a regular sequence of (clock) pulses. Also, when it goes from 0 to 1 (positive edge of the clock) Q_2Q_1 does not change. Therefore we can use Q_0 as the clock for the ACC and Q_2Q_1 as the address lines of Register File 670.

2 Experiment

1. Using the debounce switch connected to the register file and switch 0 through 5 store four different integers in locations 00, 01, 10, and 11.
2. Initialize ACC to 0.
3. Connect the counter in lab 9 to the ALU-REG-ACC device from lab 8 as follows:
   (a) Disconnect switch 0 and 1. Connect Q_1 and Q_2 out of the counter to where switch 0 and 1 were connected respectively. In your own words explain why this works.
   (b) Connect Q_0 to the clock input of the accumulator. In your own words explain why connecting Q_0 to the clock input of the ACC works when we are adding 4 numbers in register file but the counter is incremented 8 times.
   (c) Make sure the counter starts counting from 000_2, how? (We want to perform these operations in order:
   ACC += Reg[0]
   ACC += Reg[1]
   ACC += Reg[2]
   ACC += Reg[3])

(d) Make sure the counter does not count beyond $7 = 111_2$ because it would cause ACC to add content of the register file more than once. How do you accomplish this?

4. Demonstrate your circuit to the instructor.