Fluid Logic Circuits

Control Methods

In the lab, we used manual controls, pilot pressure, and electrical ladder diagrams to operate fluid power circuits. Another type of control uses air logic circuits. This method is good for environments with:

- **Explosive gases or dust**. An electrical system could create sparks, starting an explosion.
- **Vibration.** Air logic systems are more resistant to vibration and mechanical shock than electrical control systems.
- **High electrical or magnetic fields.** Air logic components do not contain magnets.

Air logic systems are rare, but you may have to troubleshoot one someday.

Air Logic Components

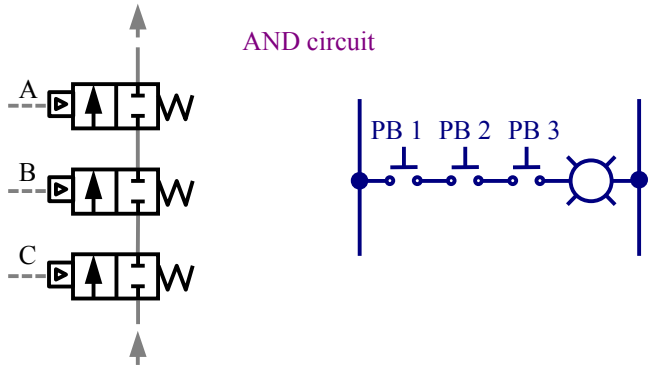
We'll look at six air logic components, then we will compare it with an electrical ladder diagram.

Logical AND

Here are three 2-way, 2-position, spring-return pneumatic directional control valves which are activated by pilot pressure. The valves are arranged in series. You have to supply pilot pressure from line A *and* line B *and* line C. So we call this an AND circuit.

You might use an AND circuit to control the operation of a press. The operator has to push the button on the left AND the button on the right in order to get the press to operate.

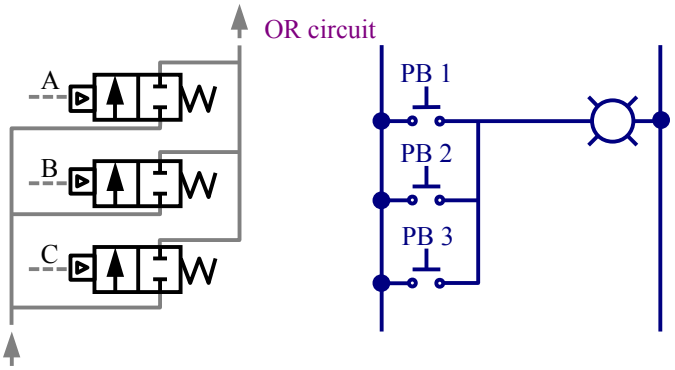
Electrically, this air logic circuit is like the electrical ladder diagram on the right. We have to depress buttons 1 *and* 2 *and* 3 in order to get the lamp to light.



Logical OR

Now we've rearranged the plumbing, so the same three valves are connected in parallel. If you supply pilot pressure from line A *or* from line B *or* from line C, then fluid will flow through the circuit. So we call this an OR circuit.

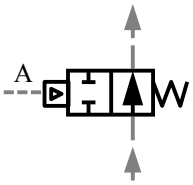
Electrically, this air logic circuit is like the electrical ladder diagram on the right. We can depress either button 1 *or* 2 *or* 3 in order to get the lamp to light.

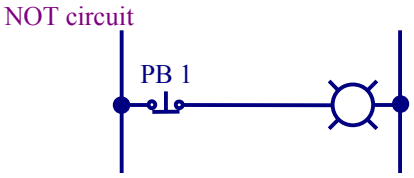


Logical NOT

We can set up the plumbing on a valve so that the spring-returned position is open, but if pilot pressure is applied, the valve shuts. In Logic, this is a NOT circuit…when pilot pressure is applied, fluid does not pass throught the valve.

The electrical ladder diagram analog is an NC push-button switch. Current flows to the lamp *unless* the push-button switch is depressed, then current does *not* flow to the lamp.

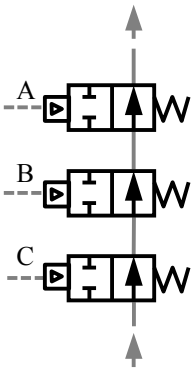


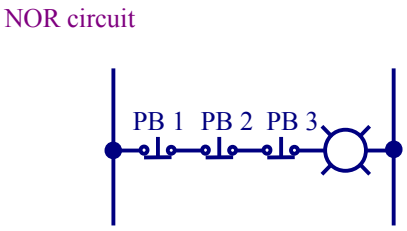


Logical NOR

The logical NOR circuit is built from NOT valves plumbed together in series. We get an output as long as neither input A *nor* input B *nor* input C is activated.

The electrical analog is a set of NC push-button switches wired together in series. Neither PB-1 *nor* PB-2 *nor* PB-3 can be depressed.

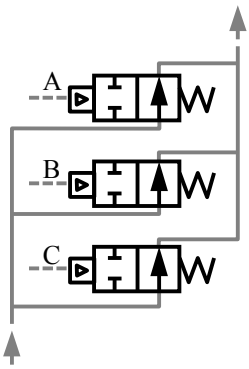


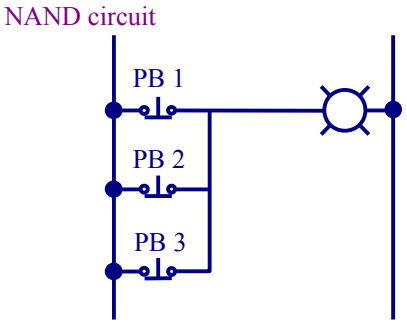


Logical NAND

A logical NAND circuit means NOT AND. We set up our valves in parallel. We lose our output signal only if pilot pressures at lines A *and* B *and* C are pressurized. If any one of these signals dies, then we get an output.

The electrical analog to this fluid logic circuit is a set of NC push-button switches wired in parallel. You have to push all three push-button switches to break the circuit.

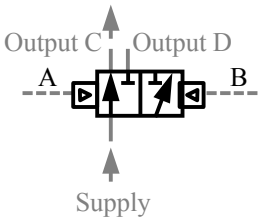


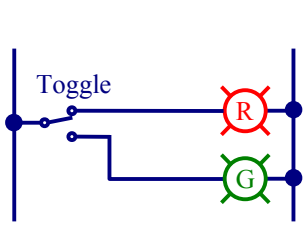


Memory

Electronic logic systems have memory…and so do fluid logic systems. By definition, memory is keeping track of what happened in the past. This valve remembers where the last pilot signal came from, and directs the flow accordingly. If pilot pressure was applied most recently to B, then the air goes to Output D. We can tell from the output which input pilot signal was the most recent.

The electrical analog is a toggle switch set to light either a red lamp or a green lamp. We can tell from the lamps which way the toggle switch is set.





Now let's look at a fluid power system controlled by limit switches and solenoids on a ladder diagram.

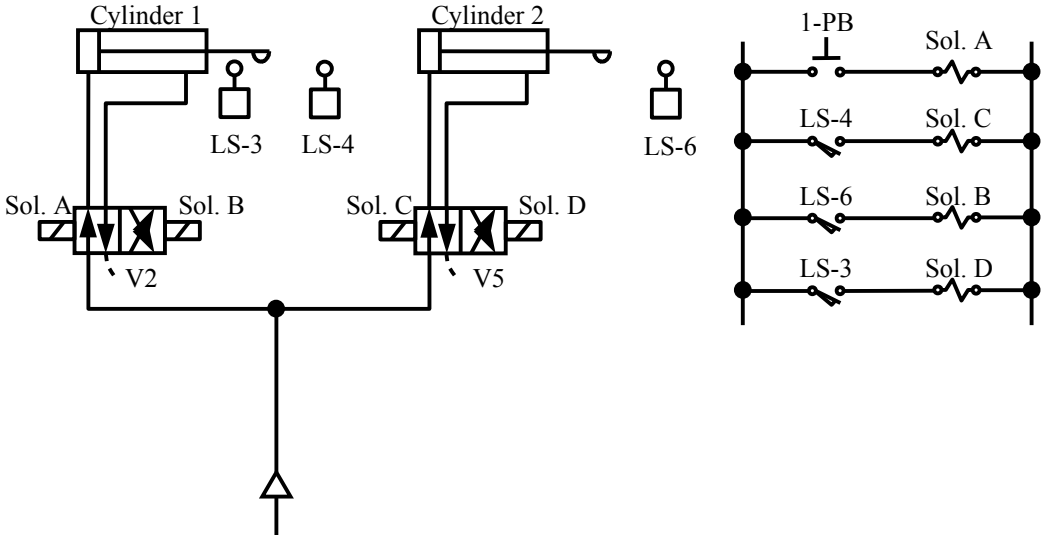
If you momentarily depress push-button 1, then solenoid A is energized.

Cylinder #1 extends, and trips Limit Switch #4.

Now solenoid C is energized, and Cylinder #2 extends, tripping Limit Switch #6.

Solenoid B is energized, so Cylinder #1 retracts, tripping Limit Switch #3.

Solenoid D is energized, so Cylinder #2 retracts.



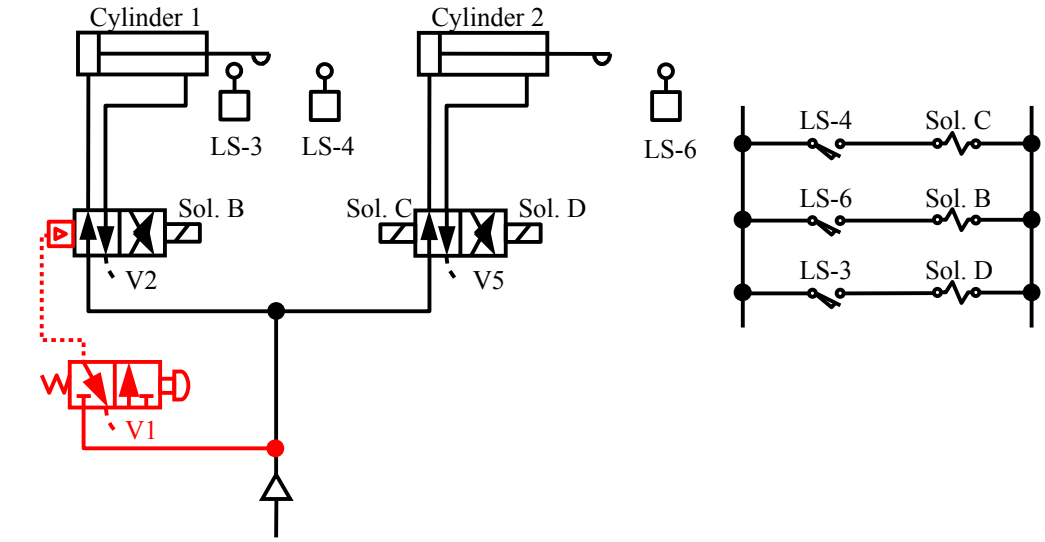
We can replace the ladder diagram with fluid logic controls.

First, let's replace the start switch on the ladder diagram with a push-button, spring-return, 2-way, 2-position valve that we'll call Valve #1 (marked in red).

Take a look at Valve #2: we've replaced Solenoid A on the left side of Valve #2 with a Pilot control.

Valve #1 is depressed, it sends a pilot signal to Valve #2.

Cylinder #1 extends.

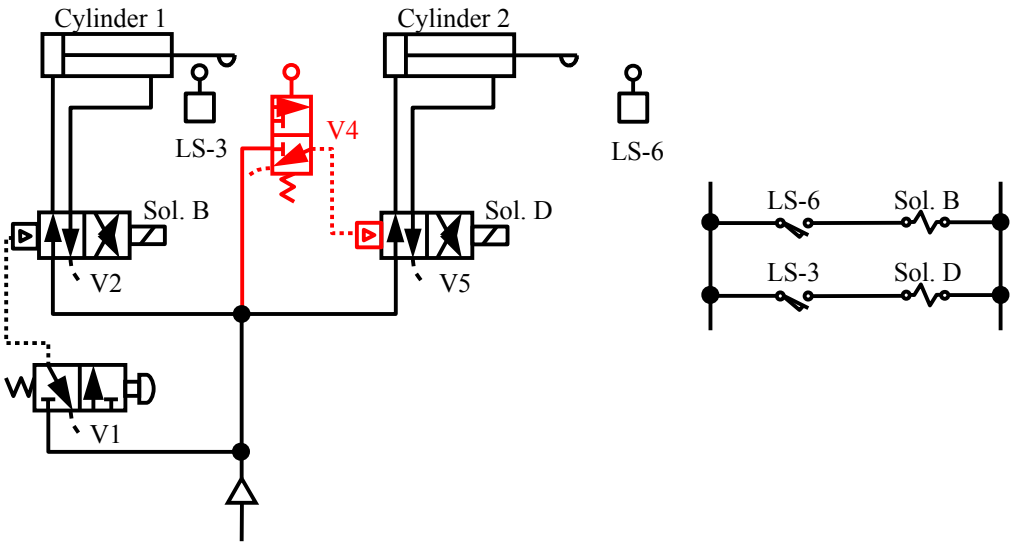


Next, let's replace Limit Switch #4 with Valve #4. We'll replace the Solenoid C on the left side of Valve #5 with a pilot control.

Cylinder #1 extends and activates Valve #4.

Pilot pressure will go to the left side of Valve #5.

Cylinder #2 extends.

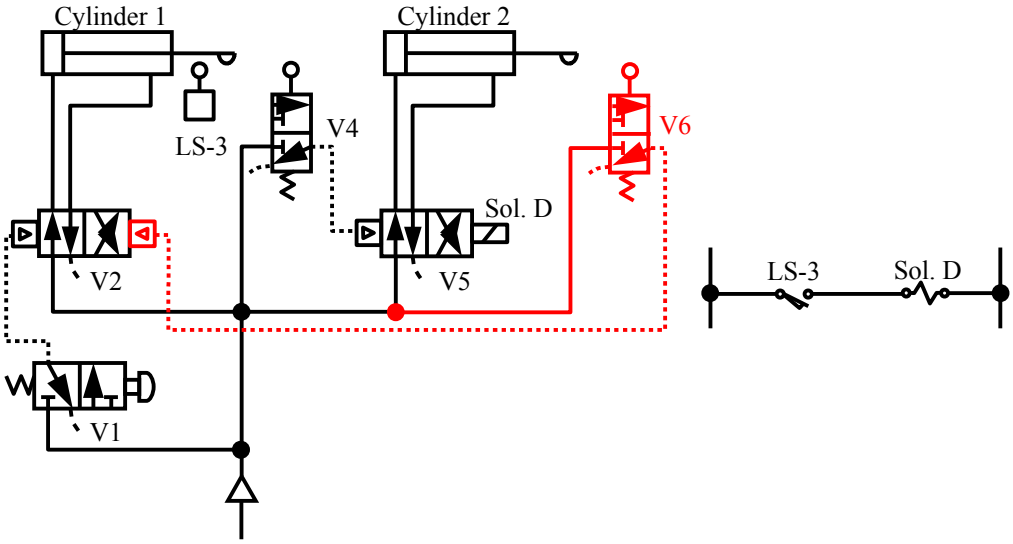


Now let's replace Limit Switch #6 with Valve #6. We'll replace the Solenoid B on the right side of Valve #2 with a pilot control.

When Cylinder #2 is fully extended, it activates Valve #6.

Pilot pressure goes to Valve #2.

Cylinder #1 retracts.

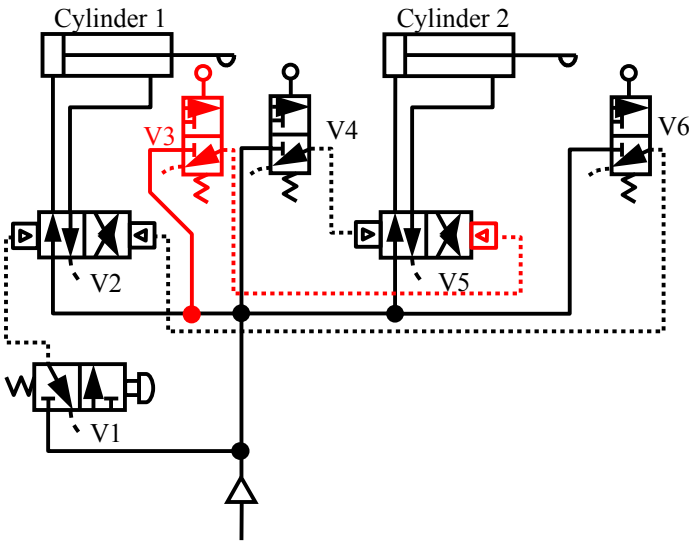


The last step is to replace Limit Switch #3 with Valve #3. We'll also replace the Solenoid D on the right side of Valve #5 with a pilot control.

When Cylinder #1 is fully retracted, it activates Valve #3.

Pilot pressure goes to Valve #5.

Cylinder #2 retracts.



This circuit operates exactly the same way as the circuit with all the solenoids and limit switches, except it doesn't require any electricity. You can run this circuit in an environment where sparks pose a hazard. This is the circuit in Figure 16-8, page 563 of the textbook.

Boolean Logic

George Boole was a self-taught mathematician who never went to college. He became Chair of Mathematics, Queen's College, Cork, Ireland, and published 50 papers on logic, probability, calculus, differential equations, and algebra.

What Boole is most remembered for is bringing the field of logic into Mathematics. Before Boole, logic was a subdiscipline of Philosophy, but Boole showed how you could perform logical calculations with mathematical operators. He divided logical outputs into a binary state: True or False, On or Off, Yes or No…or, as we apply his principles today in computers: 1 or 0.

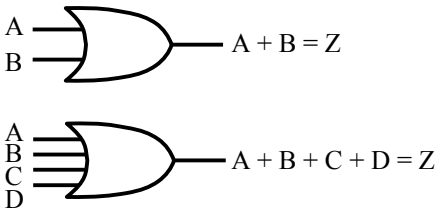
George Boole developed three operators: OR, AND, and NOT. He developed symbols for these operators, and he used truth tables to determine outputs.

Boolean OR

We'll let A and B stand for input conditions on the left, and Z will be the output condition on the right. So if A OR B is true, then Z is true.

The Boolean symbol for OR has the inputs on the left, and the output on the right.

The Boolean operator for OR is the plus sign. This can be a little confusing at first, because we're used to thinking that the plus sign means AND.



A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

We can create a truth table to list the possible inputs and outputs, where 1 = true, or yes; 0 = false, or no.

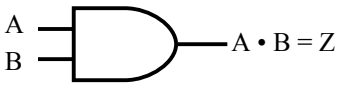
If either A = 1 or B = 1, then the output value Z = 1. So of the four possible input conditions, the output is 1 for ¾ of the conditions.

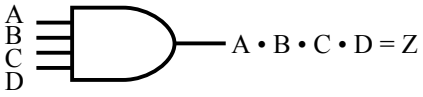
If we have lots of inputs, then we write the equation as A OR B OR C OR D = Z. So if *any* of the conditions is true, then the output will be true.

Boolean AND

The Boolean operator for AND is the dot. Both conditions have to be true for the output to be true.

Now the TRUTH TABLE has a different result. If you want an output value Z=1, then both A = 1 AND B = 1. Any other combination results in an output of Z =0. So of the four possible input conditions, the output is 1 for ¼ of the conditions.





A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

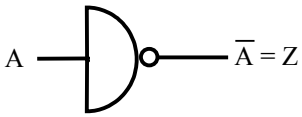
If we have lots of inputs, then we write the equation as A AND B AND C AND D = Z. So ALL of the inputs must be true for the output to be true.

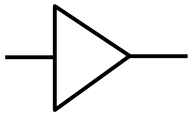
Boolean NOT

The third operator that Boole defined was the NOT operator…if the input is true, the output is false. If the input is false, the output is true.

The symbol for NOT is a horizontal line over the variable.

The Truth Table is simpler…only two options. If input A=1, then output Z=0. If input A=0, then output Z=1.





$\overline{A}$	Z
0	1
1	0

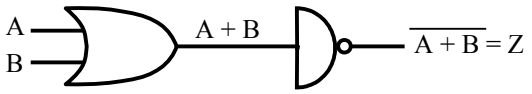
In our textbook, the NOT symbol is a half-circle with a little circle following it. In some references, you'll see a sideways triangle used for the NOT symbol.

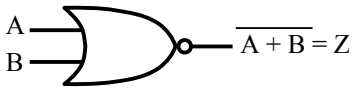
Boolean NOR

We can combine the NOT and OR operators to make the NOR operator.

The OR operator says A OR B must be true. Tack on the NOT operator, and neither A NOR B can be true for the result to be true.

We can combine these two operators into a single NOR operator, which looks like the OR symbol with a little circle after it.





A	B	A+B	Z
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

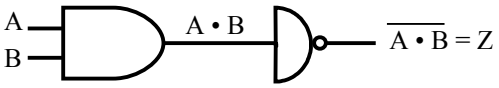
The truth table shows that output Z=1 only if neither A=1 NOR B=1.

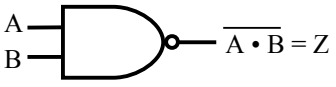
Boolean NAND

We can string NOT and AND operators in series to get the NAND operator. The result is the opposite of AND…so the result is true only if A and B are not both true.

We can combine these two operators into a single NAND operator, which looks like the AND symbol with a little circle after it.

The truth table shows that output Z=1 as long as A AND B are not both 1.





A	B	A•B	Z
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Setting Up a 2-Input Truth Table

Let's say you want to combine Boolean operators to solve a problem, such as (A + B) • B = Z. First, identify all the possible combinations of A and B. There are two inputs, and each has two settings (0 and 1), so we have 2^2 combinations.

A	B
0	0
0	1
1	0
1	1

Next, solve A + B and list the results in a column.

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Finally, solve (A + B) • B and list the results in the final column.

A	B	A+B	(A+B)•B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	1	1

Setting Up a 3-Input Truth Table

Now let's solve a problem with three inputs: $A\bullet(\overline{B+C})$. There are three inputs, each with two settings, so there are 2^3 combinations. The easiest way to figure out the combinations is to count in binary numbers (base 2), starting at zero: 0, 1, 10, 11, 100, 101, 110, 111. Place a zero in each leading place to make them three-digit numbers: 000, 001, 010, 011, 100, 101, 110, 111. These are the numbers in the first three columns A, B, and C.

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Next, solve B + C and list the results in a column.

A	B	C	B+C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

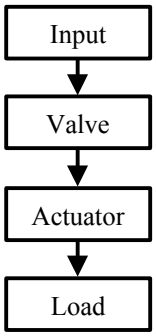
Flip zeroes to ones, ones to zeroes to find $\overline{B+C}$.

A	B	C	B+C	$\overline{B+C}$
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Finally, solve $A\bullet(\overline{B+C})$ and list the results in the final column.

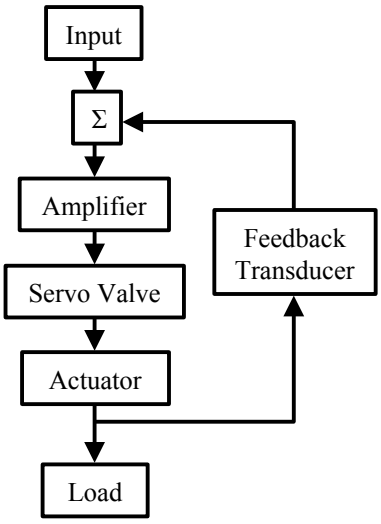
A	B	C	B+C	$\overline{B+C}$	$A\bullet(\overline{B+C})$
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

Servohydraulic Systems



Most of the control in the lab has been *open loop*, like this diagram: provide the input, a valve directs the flow of hydraulic fluid or compressed air, a motor or cylinder moves, and in the real world, that motor or cylinder would move some kind of load. In an open loop control system, there is no feedback from the output force to the input settings.

Imagine driving a car with open-loop control. Blindfold a driver, and ask the driver to turn the wheel ¼ turn to the right; the car turns right. Without feedback, the driver could turn too far, or not far enough.



If we add a feedback system, then we have a *closed loop* control system. This block diagram is a simplification of Figure 17-1.

Let's say we're using a control system to adjust the position of a hydraulic cylinder. The Greek letter Σ stands for a summing function, which compares the setpoint with the actual position. If the load increases and extends the cylinder too much, the system will retract the cylinder a little.

An example application is shown in Figure 17-3. Here, the height of a tractor hitch is controlled by an electrohydraulic servo system.

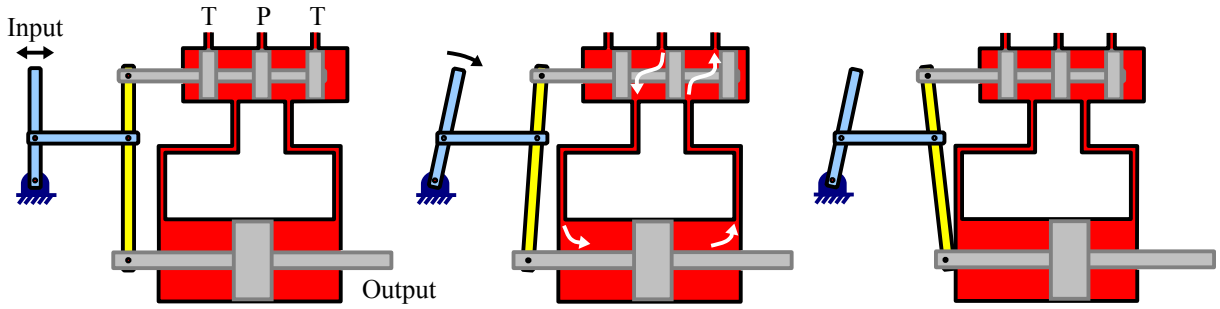
You would use closed-loop control when:

- Position or velocity have to be controlled carefully.
- Environmental conditions change the response. For example, the viscosity of hydraulic oil changes with temperature, so a hydraulic system responds differently on a cold morning.
- Operating conditions change. A good example is cruise control in a car; driving uphill requires more fuel than driving downhill.

A *servomechanism* is a feedback control system where the controlled variable is position or motion. Examples include:

- Power steering in a car, where the controlled variable is wheel angle.
- A robot arm, where the controlled variable is joint angle.
- A numerically-controlled machine tool, where the controlled variable is motion of the tool.

This cartoon shows a mechanical servo valve connected to a cylinder. When you push the input lever to the right, it moves the hydraulic valve spool to the right, and hydraulic fluid flows to the left side of the cylinder. The cylinder piston shifts to the right, causing the yellow lever to pivot and shift the hydraulic valve spool to the left, shutting off flow.

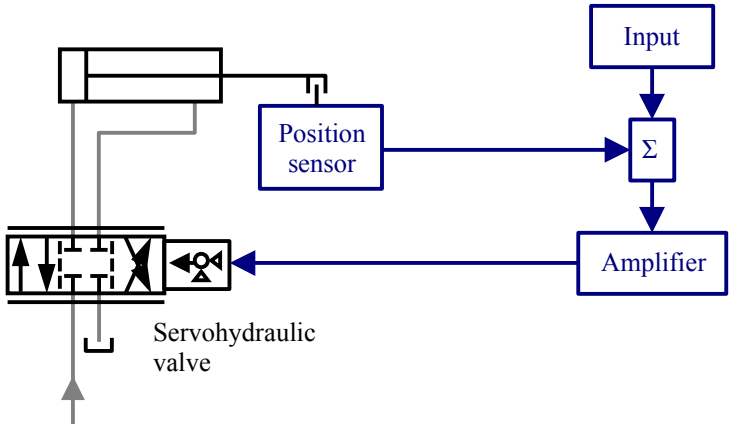


A small input load on the input lever results in a large output load from the hydraulic cylinder. Now the system is in equilibrium again, and no hydraulic fluid can flow. The cylinder is hydraulically locked…it won't move again unless the valve opens. If you replace the manual input lever with an electrical actuator, such as an electromagnet or a torque motor, you can convert a mechanical servo valve into an electrohydraulic servo valve.

In chapter 8, there's a cutaway of an actual valve, so you can see the internal parts.

This cartoon is a simplified version of a picture in the textbook of an electrohydraulic servo system. We have three parts to the system:

- **Servo valve**. The symbol looks a little different than other valve symbols. The spool position is determined by an electrical signal.
- **Feedback device**. The sensor that detects the position of the cylinder rod. Sometimes the feedback device is built into the actuator, and sometimes it's a separate unit.
- **Control unit**. This provides the input setpoint...in this case, the desired position of the valve.

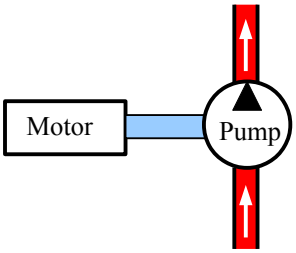


For example, let's say our setpoint is the position = 3 inches. The feedback device says our actual position is 2 inches. An error signal goes to the servo valve, and the valve lets fluid flow into the cap end of the cylinder. The cylinder moves until the position is 3 inches, so the error is 0 inches, and the valve locks the cylinder.

In any control system, you have to consider the *gain* of each component. The gain is the output signal divided by the input signal. For example, when you configure the mouse on your computer, you have a choice as to how far the cursor moves for a given travel of the mouse.

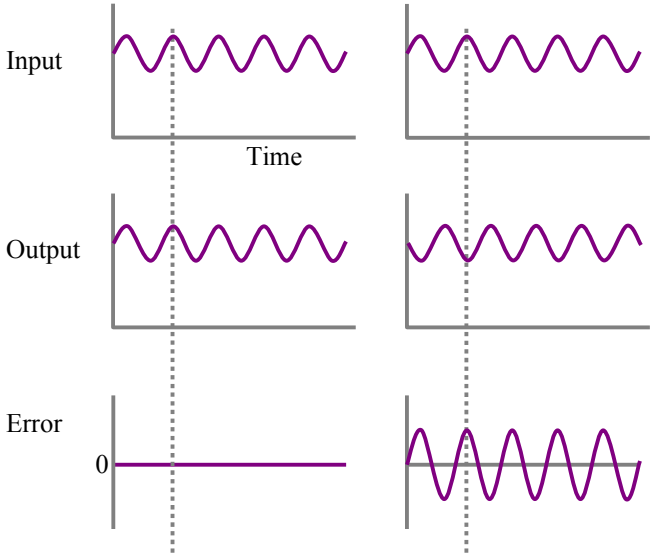
If you set it at the "slow" setting, the gain is small. The cursor moves, for example, 100 pixels on the screen for every 1 inch of movement of the mouse. If you set it at the "fast" setting, the gain is bigger. The cursor now moves, for example, 500 pixels on the screen for every 1 inch of movement of the mouse. You might want a low gain for CAD drawings, where you want to position the cursor accurately. High gain is more useful when you want the mouse to move fast across the scree without much wrist motion.

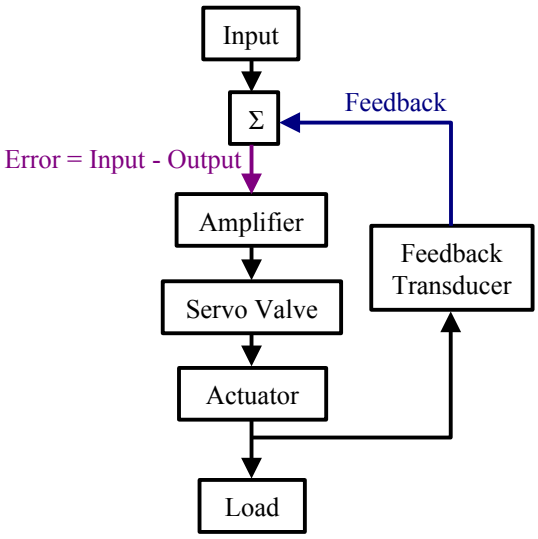
We can apply the same concept to individual components or a combination of components in a hydraulic system. Consider a motor operating drawing 2 amps of current, running at 2000 rpm, driving a 10 gpm pump. The gain of the motor is the output speed divided by the input current: $G_{motor} = \frac{2000\ \text{rpm}}{2\ \text{A}} = 1000\ \text{rpm/A}$. The gain of the pump is the output flow rate divided by the input speed: $G_{pump} = \frac{10\ \text{gpm}}{2000\ \text{rpm}} = 0.005\ \text{gpm/rpm}$. The gain of the overall system is the output flow rate divided by the input current: $G_{system} = \frac{10\ \text{gpm}}{2\ \text{A}} = 5\ \text{gpm/A}$. Alternatively, we can calculate the system gain by multiplying the individual gains: $G_{system} = G_{motor} \times G_{pump} = \frac{1000\ \text{rpm}}{\text{A}} \frac{0.005\ \text{gpm}}{\text{rpm}} = 5\ \text{gpm/A}$



Let's take another look at the feedback loop. The summing function, Σ, calculates the difference between the input (setpoint) and the output (feedback). This difference is called the *error*. The smaller the error, the less change in output.

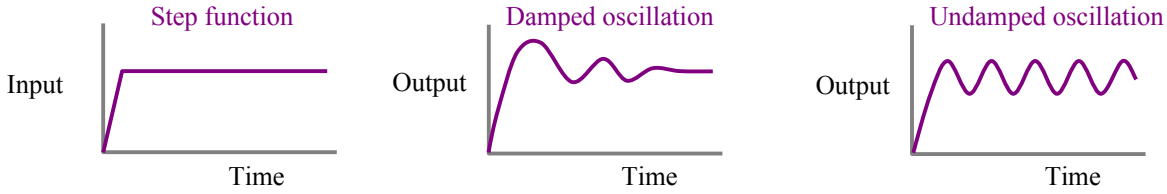
We can plot the input and output over time, and plot the error over time. If the input signal oscillates, then the output will also oscillate. As long as the two signals are in phase, the error signal will be zero, and the valve position won't change.





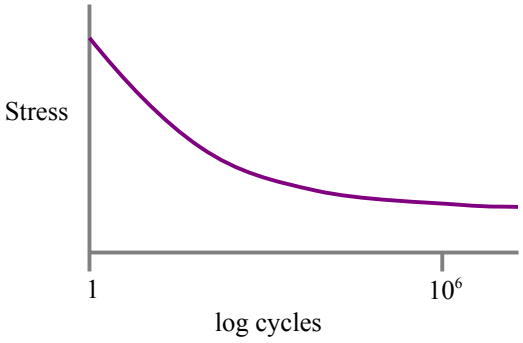
If the two signals are out of phase by 180°, then the error signal will be magnified…the valve will constantly chase a moving target, and it may shake the system apart. When you drive down a washboard road at a certain speed, it's really uncomfortable…that's the same effect.

Another issue with servohydraulic systems is their response to a *step function*. If you change the input condition suddenly, like when you turn the system on, or adjust the setting by some finite amount, it takes a while for the system to settle down.



If the output looks like the middle graph, then the system will settle down pretty quickly.

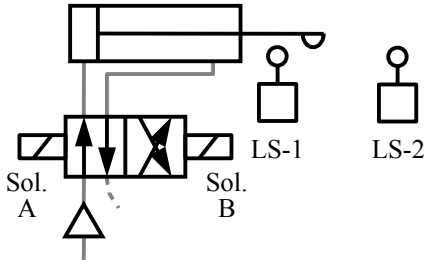
If the output looks like the righthand graph, then the oscillations are not getting damped out…this is bad, because undamped oscillations can shake the machine apart. They can lead to *fatigue failure*. Here's a classic S/N curve, where maximum stress is on the vertical axis, and the number of cycles is plotted on a logarithmic scale on the horizontal axis. It doesn't take long to reach a million cycles if the oscillations are continuous. Undamped oscillations of 1 Hz (1 cycle per second) reach 1 million cycles in just 11½ days. Therefore, it's important to check for oscillations when you set up a hydraulic system. You can change the frequency response by changing the length of hydraulic conductors or by changing the gain of the servo valve.



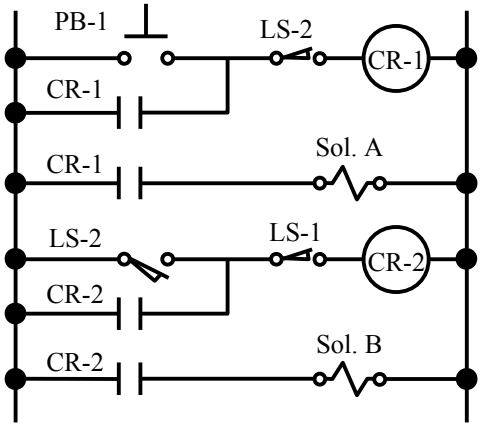
The rest of the chapter discusses PLCs. *Programmable Logic Controllers* were introduced in the 1970s to replace relay systems. They're essentially industrial computers which use ladder logic. By replacing electromechanical hardware with solid-state switches, we can increase the reliability and flexibility of the control system. This topic is covered in more depth in MET 487, and also in the EET course on PLCs.

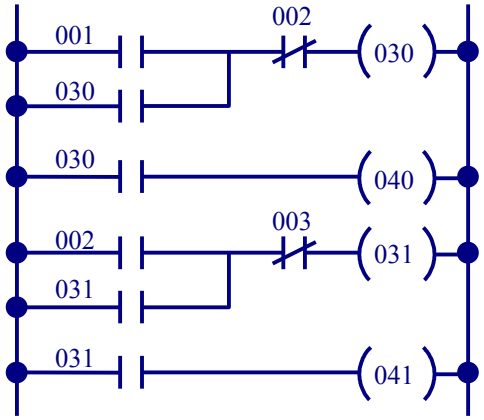
Let's look at the pneumatic circuit and conventional relay-based ladder diagram, below left.

- Momentarily depress push button PB-1
- Control Relay #1 is energized, and Solenoid A is energized
- The cylinder extends
- Limit Switch LS-2 is tripped, breaking Control Relay #1
- Control Relay #2 is energized, and Solenoid B is energized
- The cylinder retracts and stops



001	=	PB-1
002	=	LS-2
003	=	LS-1
030	=	CR-1
031	=	CR2
040	=	Sol. A
041	=	Sol. B





A PLC uses the same form of ladder diagram, except the symbols are different, as shown at the right. In a PLC ladder diagram, all inputs look like relay contacts, and all outputs look like parentheses. Sometimes the outputs are drawn as circles (like relay magnets). Since a PLC is a computer, it looks at one rung at a time, so you have to be careful about the order of the rungs. In comparison, a conventional relay-based ladder diagram "looks" at all rungs at once, so the order of the rungs is irrelevant to the function of the circuit (but highly relevant to debugging). You can read more about PLCs in the textbook.

Dr. Barry Dupen, Indiana University-Purdue University Fort Wayne. Revised June 2014. This document was created with Apache Software Foundation's OpenOffice software v.4.1.0.