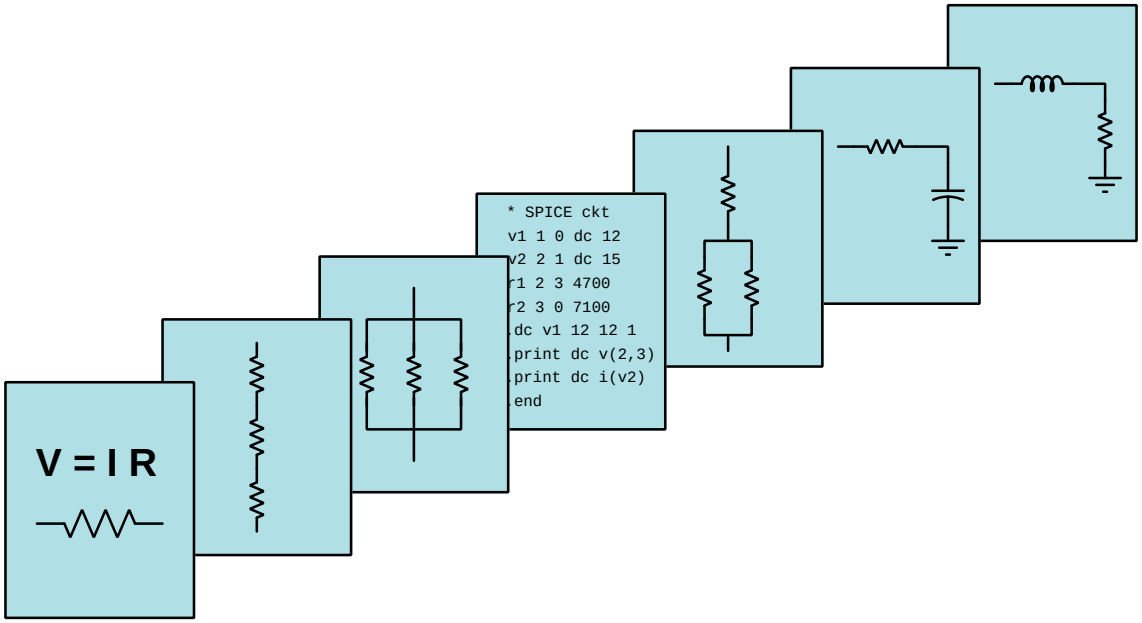
Modular Electronics Learning (ModEL) project



PN Junctions and Diodes

Last update = 20 March 2025

Contents

Chapter 1

Introduction

1.1 Recommendations for students

The PN semiconductor junction is the basis for most solid-state electronic devices, with diodes being the simplest and most direct application.

Important concepts related to diodes and PN semiconductor junctions include electric **charge carrier** motion, **atomic structure**, **electrons** versus **holes**, semiconductor **doping**, **depletion regions**, the **Shockley diode equation**, **exponential functions**, electrical **sources** versus **loads**, and diode **ratings**.

Here are some good questions to ask of yourself while studying this subject:

- How might an experiment be designed and conducted to measure the forward voltage drop of a PN junction? What hypothesis (i.e. prediction) might you pose for that experiment, and what result(s) would either support or disprove that hypothesis?
- What are some practical applications of PN-junction electronic devices?
- How do vacuum-tube electronic devices function?
- How does the operation of a solid-state electronic device compare to that of a vacuum tube?
- How do electric charge carriers move through a semiconducting material?
- Why are solid-state diodes more energy-efficient than vacuum-tube diodes?
- Why does a *depletion region* form at the interface of P-type and N-type semiconductor materials?
- Why is this "depletion region" called that? In what way is it depleted?
- What factors influence the size of a depletion region?
- What does *bias* mean with reference to a semiconductor diode?

- How do voltage and current relate to each other mathematically in a semiconductor diode?
- What are some common applications of PN junctions?
- Are diodes electrical sources or electrical loads?
- How may we electrically test a diode for proper operation?
- How do PN junctions tend to fail, compared to other common components such as resistors, capacitors, and/or inductors?
- What are some of the electrical limitations typically associated with diodes (e.g. ratings)?
- How do VOMs and DMMs differ in their ability to test diodes?
- What electrical concept is the word *shunt* synonymous with?
- How is it possible for a DMM to measure electrical resistance in a circuit without forward-biasing any PN junctions?
- What is a *Schottky diode* and what useful properties does it offer?
- What is a *Zener diode* and what useful properties does it offer?
- What is a *temperature coefficient* and what is its importance with reference to Zener diodes.

A helpful strategy for "actively" reading the Tutorial text is to apply the Shockley diode equation to the computer-simulated results presented to you in the text. Don't just read the results, but instead use a calculator to see if you can get results similar to those output by the circuit simulation software.

1.2 Challenging concepts related to semiconductor PN junctions

The following list cites concepts related to this module's topic that are easily misunderstood, along with suggestions for properly understanding them:

- **Electrical conduction in semiconductors** – what differentiates semiconductors from regular (Ohmic) conductors is the relative scarcity of free charge carriers, and how the concentration of charge carriers may be externally controlled in semiconductors by light and/or by strong electric fields. Current through a PN junction does not follow Ohm's Law (which is a great illustration of how Ohm's "Law" is not really a Law in the strict sense of the word, but rather a definition of resistance), but rather grows exponentially with applied voltage as described by the Shockley diode equation.

1.3 Recommendations for instructors

This section lists realistic student learning outcomes supported by the content of the module as well as suggested means of assessing (measuring) student learning. The outcomes state what learners should be able to do, and the assessments are specific challenges to prove students have learned.

- **Outcome** – Demonstrate effective technical reading and writing

  Assessment – Students present their outlines of this module's instructional chapters (e.g. Case Tutorial, Tutorial, Historical References, etc.) ideally as an entry to a larger Journal document chronicling their learning. These outlines should exhibit good-faith effort at summarizing major concepts explained in the text.

- **Outcome** – Calculate parameters in simple DC diode circuits

  Assessment – Calculate voltages and currents in simple diode circuits given device ratings; e.g. pose problems in the form of the "Voltages and currents in simple diode circuits" Quantitative Reasoning question.

- **Outcome** – Select appropriate components for simple DC diode circuits

  Assessment – Calculate necessary current-limiting resistor values for LED circuits; e.g. pose problems in the form of the "LED resistor sizing" Quantitative Reasoning question.

- **Outcome** – Diagnose a faulted diode circuit

  Assessment – Predict the effect(s) of a single component failing either open or shorted in a simple diode circuit; e.g. pose problems in the form of the "Faults in a diode-resistor network" Diagnostic Reasoning question.

- **Outcome** – Independent research

  Assessment – Read and summarize in your own words reliable source documents on the subject of semiconductor physics. Recommended readings include patents filed by early researchers (especially those owned by Bell Laboratories).

  Assessment – Locate rectifying diode and light-emitting diode datasheets and properly interpret some of the information contained in those documents including maximum forward current, typical forward voltage drop, maximum reverse voltage rating, power dissipation rating, etc.

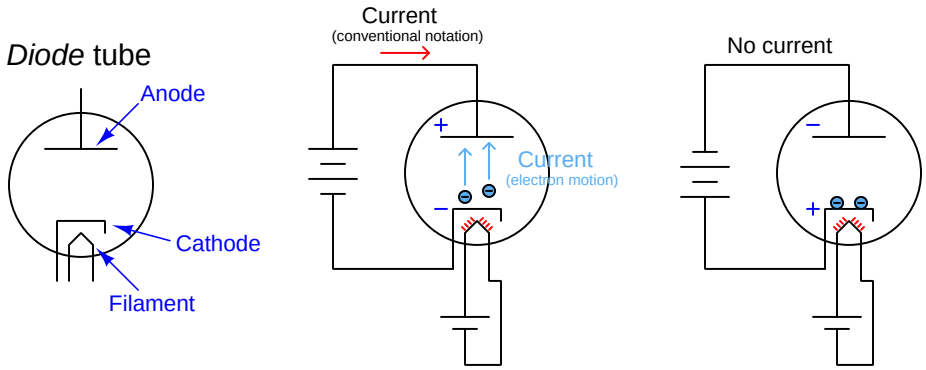
Chapter 2

Tutorial

2.1 Electrical conduction in a vacuum

Electronic components function by manipulating the energy states of electrons inside of it. The original electronic devices utilized glass tubes evacuated of air (called *vacuum tubes* or *electron tubes*) inside of which metal electrodes were placed. One electrode was heated to a dull glow by an electrically-energized filament (similar to an incandescent lamp) while the other electrode was constructed of unheated metal. Electrons within the heated electrode possessed more energy than electrons within the unheated electrode by virtue of the filament's heating, and this permitted electrons to leave the heated metal surface and enter the vacuum space – a phenomenon called *thermionic emission* – where they could travel to and be collected by the unheated metal electrode. Travel in the other direction was prohibited by the lack of thermal energy at the unheated electrode, where electrons there could not leave to enter the vacuum space. Thus, these early vacuum tubes naturally functioned as one-way "valves" for small electrical currents.

The heated electrode of a vacuum tube is called the *cathode* and the unheated "collector" electrode the *anode*. As with all uses of these terms, the cathode is intended to be the negative pole of the device while the anode is intended to be the positive pole. This makes sense based on the operational principle of a vacuum tube, where (negatively-charged) electrons may travel from the hot cathode to the cold anode but not in the other direction. These particular vacuum tubes became known as *diodes* in honor of having *two* electrodes. Later tube designs with three, four, and five electrodes became known as *triodes*, *tetrodes*, and *pentodes*, respectively:



Vacuum-tubes ushered in the era of electronics with their ability to control the flow of electric current in ways that were impossible with "passive" components such as resistors, capacitors, inductors, transformers, etc. However, vacuum-tube devices had significant limitations, among them being fragility (their glass envelopes were easily broken), high power demands (to maintain the cathode in a state of dull-red glowing temperature), and short life (due to the thin filaments breaking as well as air inevitably finding its way back into the tube's interior space). The next great breakthrough in electronic devices was the invention of so-called *solid-state* components where the travel of electrons occurred though solid crystalline materials rather than through empty vacuums. Specifically, a class of materials know as *semiconductors* permitted construction of solid-state diode and transistor components.

2.2 Electrical conduction in semiconductors

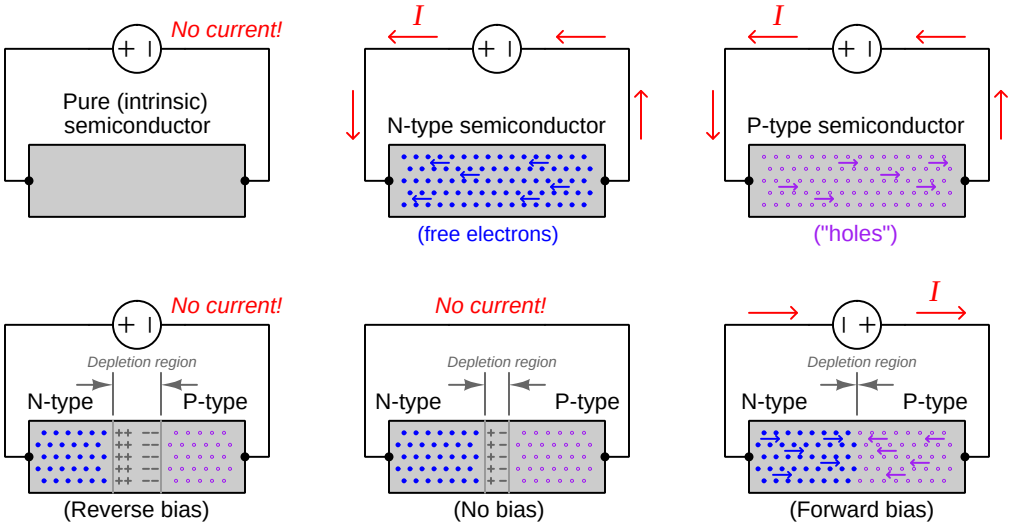
A "semiconducting" material is any substance whose electrons are *very nearly* free to move throughout. Substantial numbers of electrons residing within good conductors, by contrast, exist at sufficiently high energy levels to be entirely "unbound" from their parent atoms, and as such may freely drift throughout that solid's volume in response to any applied electric field. Electrons within the atoms of an insulating material exist at much lower energy levels, being tightly bound to their constituent atoms and unable to drift about. The nearly-conductive nature of semiconducting substances permits their higher-energy electrons to be manipulated to produce electrical conductivity on demand, much like vacuum tubes but without the high power requirements of heating and without the mechanical challenge of maintaining vacuum-tight seals. The most popular semiconducting material in modern use is the element *silicon*, found in the same vertical "group" of the Periodic Table of the Elements as the elements carbon, germanium, tin, and lead.

Semiconductor conductivity depends on a range of factors including temperature, exposure to light, addition of impurities, and the presence of electric fields to name a few. We may construct a diode from semiconducting materials by forming a "sandwich" of two differently-alloyed layers (called *P-type* and *N-type*). N-type semiconductors contain specific alloying elements contributing extra electrons to the crystalline lattice of the base material (e.g. phosphorus added to silicon), and these electrons naturally inhabit higher energy states than the electrons of the base material, allowing them to drift about like free electrons through metal. P-type semiconductor contain alloying elements contributing electron vacancies to the crystalline lattice (e.g. boron added to silicon). These vacancies, called *holes*, serve as waypoints for low-energy electrons to reside, and with enough of these "holes" in the crystalline lattice it becomes possible for electrons to move about[1] at far less energy than would normally be required for them to leave their original atoms.

When P-type and N-type semiconductor materials contact each other, high-energy "free" electrons from the N-type material fall into low-energy holes in the adjacent P-type material. This creates a thin region at the PN junction devoid of both free electrons and holes, i.e. a region *depleted* of charge carriers that might contribute to an electric current. The so-called *depletion region* is rather thin because its formation creates an internal electric field repelling additional electrons that might otherwise cross over and fill more holes. The width of this depletion region varies with the application of an external voltage: if we apply voltage with a polarity aiding the depletion region's internal electric field, the region grows larger and less conductive; if the external voltage's polarity cancels the depletion region's internal electric field, the region grows thinner and more conductive.

[1] In fact, the holes themselves may be thought to move in the opposite direction of the electrons occupying them, much like an air bubble moves through water opposite to the direction of the water molecules immediately adjacent to the bubble.

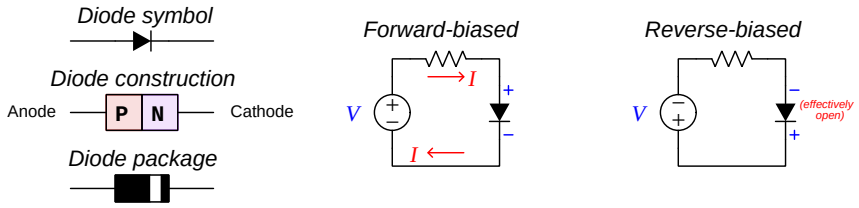
We may summarize these key behaviors of semiconductor materials by way of illustration:



As these illustrations show, a pure sample of semiconducting material such as silicon is essentially non-conducting unless and until it is alloyed ("doped") with impurities making either N-type or P-type. Either type of doping enhances conductivity, with greater doping resulting in more conductivity because more charge carriers are available to move within the material. The joining of N-type and P-type materials results in a depletion region, the width of that region controllable by a DC voltage applied across the junction. Connecting a DC voltage source with positive applied to the N-side and negative to the P-side expands the depletion region and prevents current. Reversing the DC voltage source's polarity to make the N-side negative and the P-side positive collapses the depletion region and permits current. Thus, a PN junction serves the same functional purpose as a vacuum-tube diode but with much greater efficiency and ruggedness.

2.3 PN junction diode behavior

Semiconductor diodes are simple, two-terminal devices with anode and cathode terminals. Its symbol is an arrowhead meeting with a perpendicular line. The arrowhead points in the direction of "conventional flow[2]" current notation. A colored stripe toward one end of the physical device marks the cathode terminal:



A PN junction connected with N-type negative and P-type positive (i.e. the polarity necessary for conduction) is said to be *forward-biased*, while the opposite polarity makes the PN junction *reverse-biased*.

The energy required to collapse the depletion region of a PN junction is small, but results in a relatively constant voltage drop when conducting. In other words, a forward-biased diode does not conduct as well as a metal wire (which extracts very little energy loss from passing charge carriers), but imposes an "energy penalty" on every passing charge carrier. This energy demand is not proportional to current, and so diodes do *not* obey[3] Ohm's Law. Instead, the forward voltage drop of a semiconductor diode is primarily a function of the type of semiconducting material it is made of, and secondarily a function of temperature[4]. Silicon diodes typically exhibit 0.5 to 0.7 Volts of forward-bias drop, while germanium diodes typically drop 0.3 Volts or less.

[2]Standard electrical engineering convention regards electric charge carriers as being *positive* in charge even though in metallic conductors it is actually the negatively-charged electrons that drift. The arrow-head shapes found in all semiconductor device symbols represent the easiest direction of current as sketched according to this convention, where the device (always functioning as a load) exhibits greater electrical potential on the incoming side than on the outgoing side.

[3]Ohm's Law is not so much a physical law as it is a *definition of electrical resistance* ($R = \frac{V}{I}$). Voltage and current do not remain in fixed proportion for PN junctions as they do for resistors, and so a diode does not exhibit a constant $\frac{V}{I}$ ratio like a resistor does. This is what we really mean by saying that diodes "do not obey Ohm's Law".

[4]This forward-bias voltage drop becomes smaller as temperature rises, because thermal energy assists charge carriers crossing the depletion zone and therefore less electrical energy is needed. Interestingly, this relationship between forward voltage drop and temperature for a semiconductor diode is precise enough to permit the use of diodes as electronic temperature sensors!

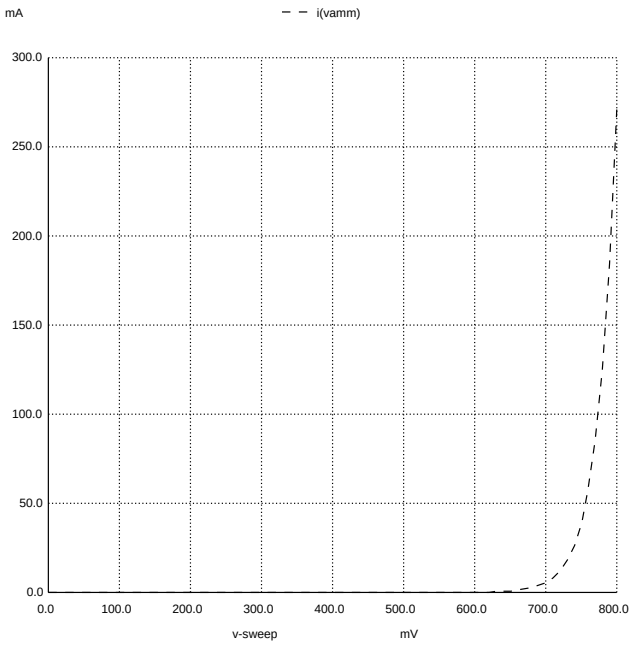
The *Shockley diode equation*, named after semiconductor pioneer William Shockley, predicts forward-bias current as a function of applied voltage and temperature:

$$I = I_S \left(e^{\frac{qV}{nKT}} - 1 \right)$$

Where,

I = Forward-bias current through the diode, Amperes
I_S = Reverse-bias saturation current[5] through the diode, Amperes
e = Euler's constant (≈ 2.71828)
V = Voltage applied to the PN junction externally, Volts
q = Elementary charge of an electron (1.602×10^{-19} Coulombs)
n = Ideality factor (1 for a perfect junction)
k = Boltzmann's constant ($1.3806504 \times 10^{-23}$ J / K)
T = Absolute temperature (Kelvin), 273.15 more than degrees Celsius

The following graph shows calculated current values for a typical diode in forward-bias mode, for an applied voltage varying from 0 Volts to 800 milliVolts (0.8 Volts):



As shown by the graph, the diode's forward current does not become significant until the applied

[5] A very small amount of current will still flow in the reverse-biased condition, due to so-called *minority carriers* in the P and N halves of the diode. This tiny current, usually in the range of *nano-Amperes*, is referred to as the *reverse saturation current* because its value does not increase appreciably with greater reverse-bias voltage but rather "saturates" or "plateaus" at a constant value. This saturation current, while fairly independent of applied voltage, varies greatly with changes in device *temperature*.

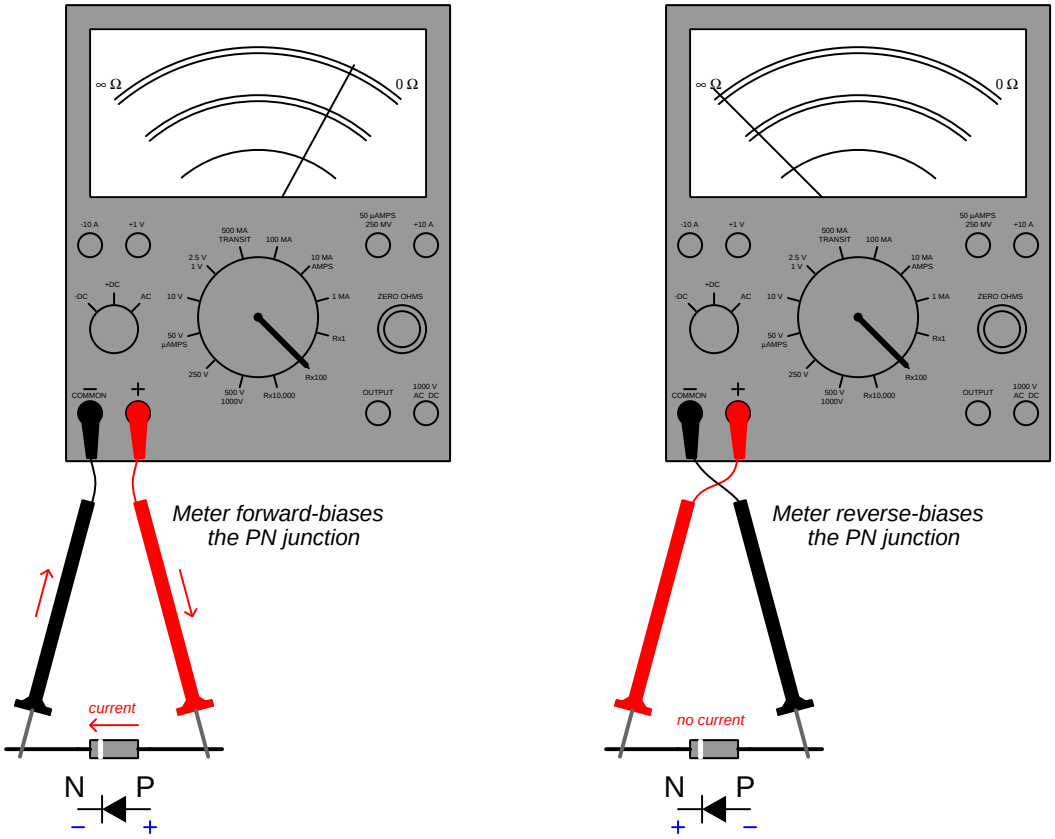
voltage is well in excess of 0.5 Volt. A computer simulation for this particular diode[6] yields forward-bias current values of 567.7×10^{-15} Amperes at 0.1 Volt, 52.06×10^{-9} Amperes at 0.4 Volts, 0.1188 $\times 10^{-3}$ Amperes at 0.6 Volts, 5.675 mA at 0.7 Volts, and 271.1 mA at 0.8 Volts. For reasons of simplicity, a constant forward voltage drop of 0.5 Volts to 0.7 Volts is often assumed for silicon diodes at modest current values (i.e. less than the diode's maximum rated current). This voltage drop means that diodes behave as electrical *loads*, with charge carriers losing some energy in the form of heat and light as they pass through the diode in its conductive state.

Many different types of diodes exist, some designed to function as *rectifiers* (acting as one-way valves for electric power), others designed to produce very constant amounts of voltage drop, and yet others designed to emit light (i.e. *light-emitting diodes*, or LEDs). A *photovoltaic cell* or *solar cell* is a special-purpose diode with its PN junction exposed to sunlight to produce electricity when illuminated, acting as an electrical source rather than an electrical load. Diodes may also be used as temperature-sensing devices, since temperature has a very strong effect on the junction's saturation current value (I_S).

[6] This simulation used a saturation current value of approximately 1.10326×10^{-14} Amperes and a junction temperature of approximately 28 degrees Celsius (approximately 301 Kelvin).

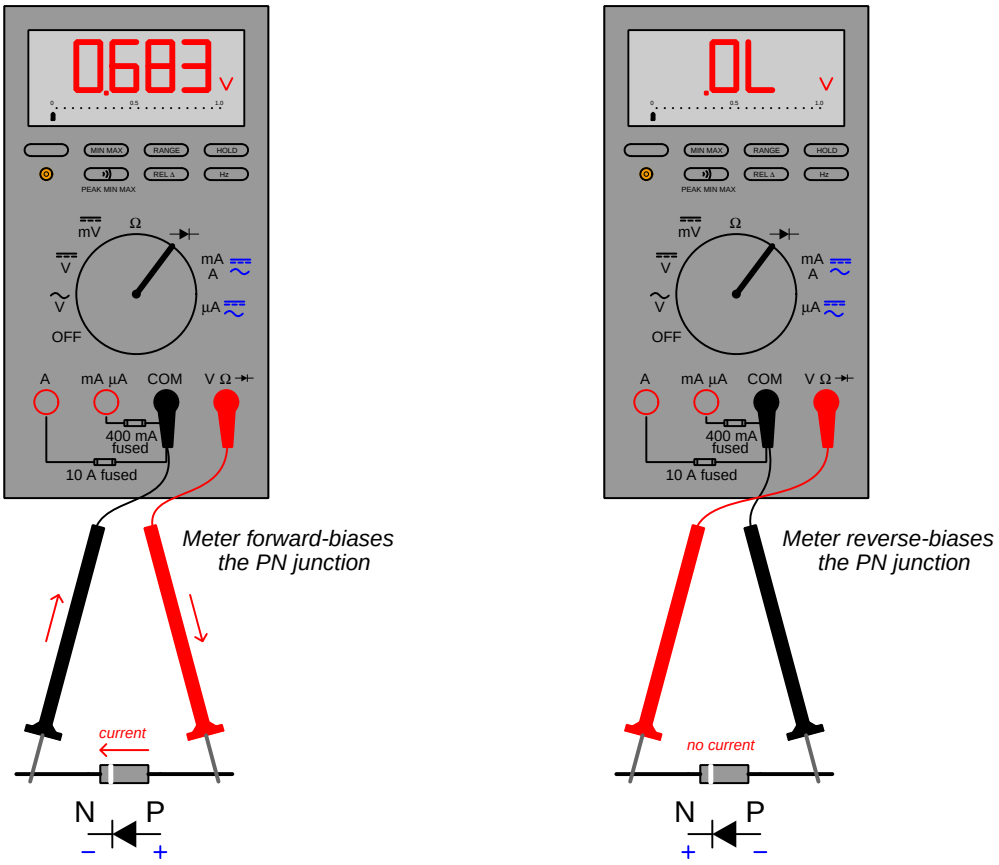
2.4 Testing PN semiconductor junctions

PN junctions – whether found in a plain diode or in any other semiconductor device – are easy to test using a multimeter. An analog Volt-Ohm-Milliammeter (VOM) can test a PN junction using its resistance (Ohms) mode, the meter's internal battery serving as the electrical source that either forward- or reverse-biases the PN junction:



When forward-biased, the meter will register a low resistance signifying conductivity; when reverse-biased the meter will show infinite resistance (open). A failed-open PN junction will exhibit extremely high resistance in both polarities, while a failed-shorted PN junction will register low resistance in both directions. Interestingly, PN junctions tend to fail *shorted* more often than many other types of components such as resistors.

Most digital multimeters (DMMs) provide a special *diode test* function in addition to resistance/continuity testing, the diode test mode displaying the voltage dropped by the PN junction when the meter passes a small amount of current through it.



Such a voltage-drop measurement is far better-suited to the testing of a PN junction than an analog ohmmeter's indication, since the voltage/current characteristics of a forward-biased PN junction do not obey Ohm's Law and therefore does not have "resistance" in the regular sense of the word.

Furthermore, the resistance function (not the diode-test function) of such a DMM is designed to apply a test voltage significantly less than 0.7 Volts so that it *cannot* forward-bias any PN junction to the point of significant conduction. This is designed intentionally so that PN junctions will not be "activated" by the meter when the user's intent is to strictly measure Ohmic resistance of other components – a particularly useful feature when PN-junction devices are found connected in-circuit with resistors and other devices, and you only wish to measure the resistance of the non-semiconductor devices.

2.5 PN junction diode ratings

Diodes, like all electrical components, have ratings which must be respected for reliable operation. The following list shows some of the more important ratings typical for diodes:

- **Maximum average forward current** – the amount of continuous forward current the diode should be able to carry
- **Maximum surge forward current** – the amount of forward current the diode should be able to carry for brief periods of time (typically milliseconds)
- **Typical forward voltage** – the amount of voltage drop expected under normal operating conditions.
- **DC blocking voltage** – the amount of continuous reverse-bias voltage the diode should be able to withstand without "breaking down" and passing reverse current
- **Peak inverse voltage** – also called "peak reverse voltage", this is the amount of reverse-bias voltage the diode should be able to *momentarily* withstand while still preventing current
- **Reverse leakage current** – this is the amount of current that will pass through the diode "backwards" when it is subjected to a reverse-bias voltage. Ideally, of course, there should be no "leakage" current at all, but all diodes exhibit some.

The best source of information for any particular diode's ratings is the manufacturer's *datasheet*.

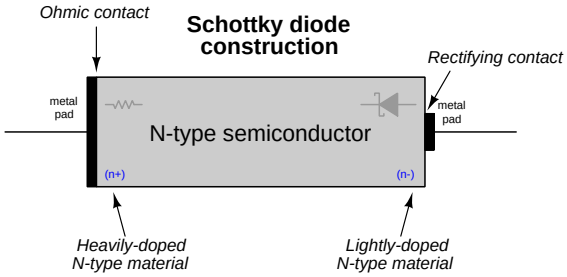
2.6 Schottky diodes

A special type of rectifying diode known as a *Schottky diode*, named after the German physicist Walter Schottky, exhibits significantly less forward voltage drop than regular silicon PN-junction diodes, typically only 0.4 Volts when conducting as opposed to 0.7 Volts. Its symbol is shown below:

Schottky diode

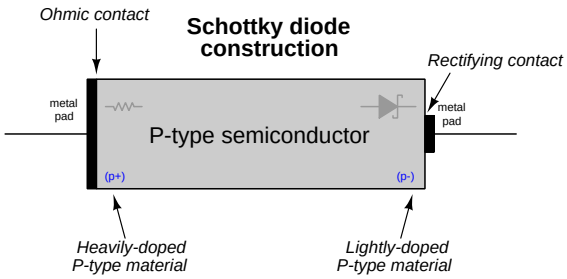
Rather than utilizing a junction between N-doped silicon and P-doped silicon as do standard rectifying diodes, the Schottky diode employs a special junction between *metal* and doped silicon, typically N-type silicon. Metal-to-silicon junctions are a basic necessity whenever we must equip a semiconductor device with metal leads or connection terminals, as we must have some way of attaching those metal parts to the semiconductor material. However, the concentration of *doping* within the semiconducting material significantly affects the electrical characteristics of any metal-semiconductor junction. For high doping concentrations the metal-semiconductor junction will exhibit what is called *ohmic* characteristics, so named because it obeys Ohm's Law with current being proportional to voltage drop regardless of polarity. In other words, an "ohmic" metal-semiconductor junction formed between heavily-doped silicon and metal does not rectify at all, and this is the type of contact we desire when attaching metal terminals to any semiconductor device. For lesser doping concentrations, though, the junction between metal and semiconducting silicon will exhibit rectifying characteristics (i.e. passing current much easier in one direction than the other). It is this latter phenomenon that forms the basis of the Schottky diode.

Inside a typical Schottky diode is a piece of N-doped silicon with an ohmic metal contact on one end and a rectifying metal contact on the other. The metal contact pad on the ohmic side is usually large in area to reduce the ohmic resistance to a bare minimum, the goal being for charge carriers to see that metal-semiconductor junction as completely seamless while the other (rectifying) metal-semiconductor junction acts as a true diode. Modern semiconductor manufacturing techniques make it easy to create a *gradient* of doping concentration along any length of semiconducting material, such that one end has a heavier dopant concentration[7] than the other:



[7] Dopant concentration is typically denoted by "+" and "−" symbols following "p" or "n" designators, the "+" and "−" symbols representing heavier versus lighter concentrations. Note that "+" and "−" do *not* represent polarity or charge carrier type, but merely how concentrated the P-type or N-type dopants happen to be in a particular region of the semiconductor's bulk. For example, "p+" means heavily-doped P-type and "n+" means heavily-doped N-type.

Schottky diodes may be manufactured from either N-type or P-type semiconducting material. In the case of N-doping, the rectifying contact is such that the lightly-doped N-type semiconductor ("n−") forms the cathode and the metal pad forms the anode, as shown in the previous illustration. In the case of P-doping, the lightly-doped P-type semiconductor ("p−") forms the anode and the metal forms the cathode at the rectifying contact as shown below:

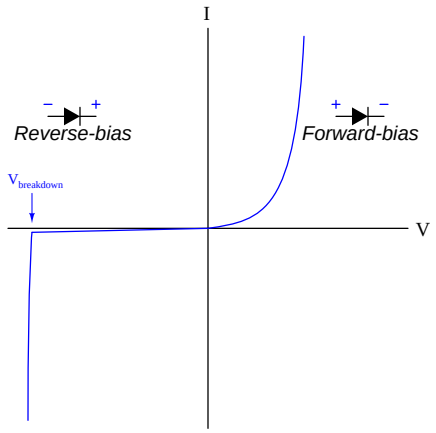


Not only do Schottky diodes have a much lower forward voltage drop than PN-junction silicon diodes, but they also have almost no *stored charge* within their depletion regions which means they are able to turn off (i.e. stop conducting) much more rapidly following a reversal of applied voltage polarity than standard PN junction diodes. Here, "stored charge" refers to *charge carriers* (i.e. electrons and holes) inhabiting the depletion region as a result of the diode having been forward-biased, with those injected charge carriers having to "clear out" of the depletion region once voltage polarity switches to reverse bias. Only after clearing out of the depletion region of the PN junction is that region truly "depleted" of charge carriers again and therefore non-conductive. The motion of those charge carriers as they vacate the depletion region constitutes a very brief current in the "wrong" direction, which means for a very short period of time following polarity reversal the diode fails to block reverse current as it ideally should. Schottky diodes don't have as many stored charges within their depletion regions as regular PN silicon diodes do, and so this means they exhibit reverse current for a much shorter amount of time when polarity is switched from forward- to reverse-bias. This characteristic makes Schottky diodes well-suited for high-speed digital switching applications as well as high-frequency switch-mode power conversion circuits where their low forward-voltage drop is a further benefit over standard rectifying diodes.

Schottky diodes typically have lower reverse-voltage breakdown ratings and exhibit higher reverse leakage current than comparable silicon PN junction diodes.

2.7 Zener diodes

Another special type of diode is called the *Zener diode* in honor of the American physicist Clarence Zener who studied breakdown of electrical insulators. All diodes will "break down" and conduct electric current in the reverse direction given sufficient applied reverse-bias voltage. With normal rectifying diodes this is considered a weakness or a flaw, but with Zener diodes it is considered a useful feature. The forward- and reverse-biased characteristics of typical diodes are shown in the following graph plotting current (I) on the vertical axis and voltage (V) on the horizontal axis:



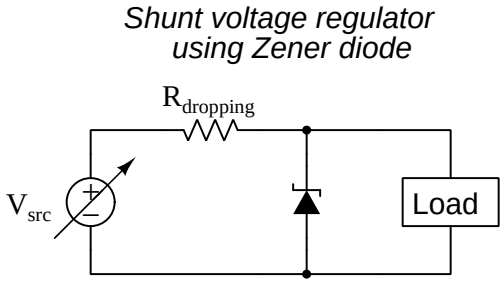
Zener diodes are differentiated from other types of diodes in schematic diagrams by the following symbol:

Zener diode

A Zener diode's breakdown voltage, often referred to as its *Zener voltage*, is determined by precise control of the doping concentrations used to make the P-type and N-type semiconductor regions within its PN junction. This voltage also varies with temperature, usually represented as a *temperature coefficient* in units of percent per degree Celsius. Lower-voltage Zener diodes use highly-doped N-type and P-type regions, and their Zener voltage values tend to decrease as temperature rises – i.e. a negative temperature coefficient. Higher-voltage Zener diodes use lighter-doped N- and P-type semiconductor regions[8], and their Zener voltage values increase with increasing junction temperature – i.e. a positive temperature coefficient. Zener diodes with breakdown ratings around 5 Volts tend to exhibit the temperature coefficients closest to zero.

[8] These Zener diodes actually break down in a different manner than lower-voltage Zener diodes, the mechanism of higher-voltage breakdown being *avalanche* rather than the lower-voltage *Zener effect* which is based on quantum tunneling of charge carriers between N- and P-doped sides of the silicon junction.

Perhaps the most common application for Zener diodes is *voltage-regulator* networks, where a variable DC voltage is stabilized to a lower voltage value set by the breakdown rating of a Zener diode. An example of this use is the following "shunt" voltage regulator network using a Zener diode connected in parallel (i.e. *shunt*) with the load and a "dropping" resistor connecting to the unregulated DC voltage source to drop any excess voltage:



Chapter 3

Historical References

This chapter is where you will find references to historical texts and technologies related to the module's topic.

Readers may wonder why historical references might be included in any modern lesson on a subject. Why dwell on old ideas and obsolete technologies? One answer to this question is that the initial discoveries and early applications of scientific principles typically present those principles in forms that are unusually easy to grasp. Anyone who first discovers a new principle must necessarily do so from a perspective of ignorance (i.e. if you truly *discover* something yourself, it means you must have come to that discovery with no prior knowledge of it and no hints from others knowledgeable in it), and in so doing the discoverer lacks any hindsight or advantage that might have otherwise come from a more advanced perspective. Thus, discoverers are forced to think and express themselves in less-advanced terms, and this often makes their explanations more readily accessible to others who, like the discoverer, comes to this idea with no prior knowledge. Furthermore, early discoverers often faced the daunting challenge of explaining their new and complex ideas to a naturally skeptical scientific community, and this pressure incentivized clear and compelling communication. As James Clerk Maxwell eloquently stated in the Preface to his book *A Treatise on Electricity and Magnetism* written in 1873,

> It is of great advantage to the student of any subject to read the original memoirs on that subject, for science is always most completely assimilated when it is in its nascent state . . . `[page xi]`

Furthermore, grasping the historical context of technological discoveries is important for understanding how science intersects with culture and civilization, which is ever important because new discoveries and new applications of existing discoveries will always continue to impact our lives. One will often find themselves impressed by the ingenuity of previous generations, and by the high degree of refinement to which now-obsolete technologies were once raised. There is much to learn and much inspiration to be drawn from the technological past, and to the inquisitive mind these historical references are treasures waiting to be (re)-discovered.

3.1 Crystal detectors

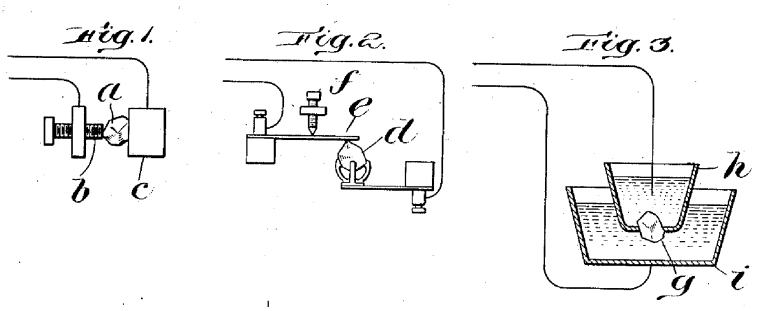
An important component within primitive AM (amplitude-modulation) radio receiver circuits is the *detector*, which is basically a diode with a very high frequency capability. Vacuum-tube diodes are one early technology suitable for this task, but others exist as well. The following US patent (number 879,117) describes a solid-state diode constructed of naturally occurring crystal materials, which of course enjoys the advantages of physical ruggedness and lower power requirements than a heated-filament vacuum tube.

The inventor in this patent, George W. Pierce of the Massachusetts Wireless Equipment Company, describes the purpose and application of his crystal-based detector in a set of paragraphs found on page 1:

> The invention relates to rectifiers and detectors for electric currents or electric oscillations and more particularly to rectifiers and detectors such as may be successfully used as receivers in wireless telegraphy or electric wave signaling systems.
>
> In practicing my invention I employ as a rectifier or detector a substance known as hessite which occurs in nature usually as telluride of silver, although the silver is sometimes wholly or partially replaced by gold. I have discovered that this substance is asymmetrically conductive when used in connection with small currents, and I have also discovered that by reason of this property of asymmetrical conductivity and possibly of other unknown properties this substance when properly placed between conductive electrodes is highly sensitive as a receiver for electromagnetic waves.
>
> The rectifying or detecting material may be utilized in many and various shapes and may be connected in the circuit of the receiving or other apparatus, in various ways. [page 1]

Key to realizing the "asymmetrical conductivity" of these crystals is the means of electrical contact made with their faces. The inventor describes and illustrates three different methods he found to function well:

> In the arrangement of Fig. 1, a piece a of hessite, is held in a clamp, the jaws b and c of which form electrodes making contact with the rectifying or detecting material.
>
> I have secured the best results by cutting a piece from a crystal on a plane substantially at right angles to its axis and engaging, one of the electrodes with a point or corner of the crystal so that a small area of contact is secured.
>
> In the arrangement of Fig.2 a crystal or fragment of a crystal d is held in an ordinary jewel mounting and a point of the crystal is engaged by a metallic spring e. The contact between the spring e and the detecting material may be adjusted by a screw f to secure the character of contact which will give the best results.
>
> In the arrangement shown in Fig. 3 a piece g of hessite is sealed in the end of a glass tube h. The piece g is in contact with a mass of conducting liquid within the tube h and with a second mass of conducting liquid within a receptacle i, the masses of liquid forming the electrodes which contact with the piece g. The liquid should be such that it will not act destructively upon the piece g.
>
> Any other suitable form and arrangement of the electrodes and of the rectifying or detecting material may be employed which may be found desirable or best suited to the conditions under which the particular apparatus in which the invention is embodied is to be used. [page 1]



It should be apparent from Pierce's descriptions that the theory of crystal diode behavior was not understood at that time. One would simply obtain a piece of crystal and experiment with different attachment methods until suitable rectification occurred. Radio technicians would later come to refer to the crystal-based detectors as *cat's whisker detectors* because they often employed a spring-steel wire (the "whisker") for one contact point on the crystal, the optimum placement of that spring wire being a matter of trial and error.

From our modern perspective, these early crystal "detectors" are more precisely understood as being *Schottky diodes* where the rectifying action takes place within a junction between a semiconducting material and plain metal.

A photograph of an actual AM radio receiver designed around a crystal detector is seen in the following photograph:

The silver-grey colored crystal is clearly visible inside the brass holder, with a thin metal wire "whisker" touching it on top. The four brass terminals on top are:

- **Aerial** – attaches to a long-wire antenna suspended above the ground
- **Phone** – attaches to one terminal of a speaker headset ("phone")
- **Phone** – attaches to the other terminal of a speaker headset ("phone")
- **Earth** – attaches to a metal rod or pipe buried in the Earth for a solid ground connection

A variable (slide-wire) inductor provides a means for tuning different AM frequencies, this variable inductance combining with a fixed capacitor (not seen in the photo) to form a resonant LC network acting as a frequency-selective *filter* so that the listener would hear only one AM broadcast signal frequency at any given time.

Chapter 4

Programming References

A powerful tool for mathematical modeling is text-based *computer programming*. This is where you type coded commands in text form which the computer is able to interpret. Many different text-based languages exist for this purpose, but we will focus here on just two of them, *C++* and *Python*.

4.1 Programming in C++

One of the more popular text-based computer programming languages is called *C++*. This is a *compiled* language, which means you must create a plain-text file containing C++ code using a program called a *text editor*, then execute a software application called a *compiler* to translate your "source code" into instructions directly understandable to the computer. Here is an example of "source code" for a very simple C++ program intended to perform some basic arithmetic operations and print the results to the computer's console:

```
#include <iostream>
using namespace std;

int main (void)
{
  float x, y;

  x = 200;
  y = -560.5;

  cout << "This simple program performs basic arithmetic on" << endl;
  cout << "the two numbers " << x << " and " << y << " and then" << endl;
  cout << "displays the results on the computer's console." << endl;

  cout << endl;

  cout << "Sum = " << x + y << endl;
  cout << "Difference = " << x - y << endl;
  cout << "Product = " << x * y << endl;
  cout << "Quotient of " << x / y << endl;

  return 0;
}
```

Computer languages such as C++ are designed to make sense when read by human programmers. The general order of execution is left-to-right, top-to-bottom just the same as reading any text document written in English. Blank lines, indentation, and other "whitespace" is largely irrelevant in C++ code, and is included only to make the code more pleasing[1] to view.

[1] Although not included in this example, *comments* preceded by double-forward slash characters (`//`) may be added to source code as well to provide explanations of what the code is supposed to do, for the benefit of anyone reading it. The compiler application will ignore all comments.

Let's examine the C++ source code to explain what it means:

- `#include <iostream>` and `using namespace std;` are set-up instructions to the compiler giving it some context in which to interpret your code. The code specific to your task is located between the brace symbols ({ and }, often referred to as "curly-braces").

- `int main (void)` labels the "Main" function for the computer: the instructions within this function (lying between the { and } symbols) it will be commanded to execute. Every complete C++ program contains a `main` function at minimum, and often additional functions as well, but the `main` function is where execution always begins. The `int` declares this function will return an *integer* number value when complete, which helps to explain the purpose of the `return 0;` statement at the end of the `main` function: providing a numerical value of zero at the program's completion as promised by `int`. This returned value is rather incidental to our purpose here, but it is fairly standard practice in C++ programming.

- Grouping symbols such as (parentheses) and {braces} abound in C, C++, and other languages (e.g. Java). Parentheses typically group data to be processed by a function, called *arguments* to that function. Braces surround lines of executable code belonging to a particular function.

- The `float` declaration reserves places in the computer's memory for two *floating-point* variables, in this case the variables' names being `x` and `y`. In most text-based programming languages, variables may be named by single letters or by combinations of letters (e.g. `xyz` would be a single variable).

- The next two lines assign numerical values to the two variables. Note how each line terminates with a semicolon character (`;`) and how this pattern holds true for most of the lines in this program. In C++ semicolons are analogous to periods at the ends of English sentences. This demarcation of each line's end is necessary because C++ ignores whitespace on the page and doesn't "know" otherwise where one line ends and another begins.

- All the other instructions take the form of a `cout` command which prints characters to the "standard output" stream of the computer, which in this case will be text displayed on the console. The double-less-than symbols (`<<`) show data being sent *toward* the `cout` command. Note how verbatim text is enclosed in quotation marks, while variables such as `x` or mathematical expressions such as `x - y` are not enclosed in quotations because we want the computer to display the numerical values represented, not the literal text.

- Standard arithmetic operations (add, subtract, multiply, divide) are represented as `+`, `-`, `*`, and `/`, respectively.

- The `endl` found at the end of every `cout` statement marks the end of a line of text printed to the computer's console display. If not for these `endl` inclusions, the displayed text would resemble a run-on sentence rather than a paragraph. Note the `cout << endl;` line, which does nothing but create a blank line on the screen, for no reason other than esthetics.

After saving this *source code* text to a file with its own name (e.g. `myprogram.cpp`), you would then *compile* the source code into an *executable* file which the computer may then run. If you are using a console-based compiler such as *GCC* (very popular within variants of the Unix operating system[2], such as Linux and Apple's OS X), you would type the following command and press the Enter key:

```
g++ -o myprogram.exe myprogram.cpp
```

This command instructs the `GCC` compiler to take your source code (`myprogram.cpp`) and create with it an executable file named `myprogram.exe`. Simply typing `./myprogram.exe` at the command-line will then execute your program:

```
./myprogram.exe
```

If you are using a graphic-based C++ development system such as Microsoft Visual Studio[3], you may simply create a new console application "project" using this software, then paste or type your code into the example template appearing in the editor window, and finally run your application to test its output.

As this program runs, it displays the following text to the console:

```
This simple program performs basic arithmetic on
the two numbers 200 and -560.5 and then
displays the results on the computer's console.

Sum = -360.5
Difference = 760.5
Product = -112100
Quotient of -0.356824
```

As crude as this example program is, it serves the purpose of showing how easy it is to write and execute simple programs in a computer using the C++ language. As you encounter C++ example programs (shown as source code) in any of these modules, feel free to directly copy-and-paste the source code text into a text editor's screen, then follow the rest of the instructions given here (i.e. save to a file, compile, and finally run your program). You will find that it is generally easier to

[2] A very functional option for users of Microsoft Windows is called *Cygwin*, which provides a Unix-like console environment complete with all the customary utility applications such as GCC!

[3] Using Microsoft Visual Studio community version 2017 at the time of this writing to test this example, here are the steps I needed to follow in order to successfully compile and run a simple program such as this: (1) Start up Visual Studio and select the option to create a New Project; (2) Select the Windows Console Application template, as this will perform necessary set-up steps to generate a console-based program which will save you time and effort as well as avoid simple errors of omission; (3) When the editing screen appears, type or paste the C++ code within the `main()` function provided in the template, deleting the "Hello World" `cout` line that came with the template; (4) Type or paste any preprocessor directives (e.g. `#include` statements, `namespace` statements) necessary for your code that did not come with the template; (5) Lastly, under the Debug drop-down menu choose either Start Debugging (F5 hot-key) or Start Without Debugging (Ctrl-F5 hotkeys) to compile ("Build") and run your new program. Upon execution a console window will appear showing the output of your program.

learn computer programming by closely examining others' example programs and modifying them than it is to write your own programs starting from a blank screen.

4.2 Programming in Python

Another text-based computer programming language called *Python* allows you to type instructions at a terminal prompt and receive immediate results without having to compile that code. This is because Python is an *interpreted* language: a software application called an *interpreter* reads your source code, translates it into computer-understandable instructions, and then executes those instructions in one step.

The following shows what happens on my personal computer when I start up the Python interpreter on my personal computer, by typing `python3`[4] and pressing the Enter key:

```
Python 3.7.2 (default, Feb 19 2019, 18:15:18)
[GCC 4.1.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The `>>>` symbols represent the prompt within the Python interpreter "shell", signifying readiness to accept Python commands entered by the user.

Shown here is an example of the same arithmetic operations performed on the same quantities, using a Python interpreter. All lines shown preceded by the `>>>` prompt are entries typed by the human programmer, and all lines shown without the `>>>` prompt are responses from the Python interpreter software:

```
>>> x = 200
>>> y = -560.5
>>> x + y
-360.5
>>> x - y
760.5
>>> x * y
-112100.0
>>> x / y
-0.35682426404995538
>>> quit()
```

[4] Using version 3 of Python, which is the latest at the time of this writing.

More advanced mathematical functions are accessible in Python by first entering the line `from math import *` which "imports" these functions from Python's math *library* (with functions identical to those available for the C programming language, and included on any computer with Python installed). Some examples show some of these functions in use, demonstrating how the Python interpreter may be used as a scientific calculator:

```
>>> from math import *
>>> sin(30.0)
-0.98803162409286183
>>> sin(radians(30.0))
0.49999999999999994
>>> pow(2.0, 5.0)
32.0
>>> log10(10000.0)
4.0
>>> e
2.7182818284590451
>>> pi
3.1415926535897931
>>> log(pow(e,6.0))
6.0
>>> asin(0.7071068)
0.78539819000368838
>>> degrees(asin(0.7071068))
45.000001524425265
>>> quit()
```

Note how trigonometric functions assume angles expressed in *radians* rather than *degrees*, and how Python provides convenient functions for translating between the two. Logarithms assume a base of e unless otherwise stated (e.g. the `log10` function for common logarithms).

The interpreted (versus compiled) nature of Python, as well as its relatively simple syntax, makes it a good choice as a person's first programming language. For complex applications, interpreted languages such as Python execute slower than compiled languages such as C++, but for the very simple examples used in these learning modules speed is not a concern.

Another Python math library is `cmath`, giving Python the ability to perform arithmetic on complex numbers. This is very useful for AC circuit analysis using *phasors*[5] as shown in the following example. Here we see Python's interpreter used as a scientific calculator to show series and parallel impedances of a resistor, capacitor, and inductor in a 60 Hz AC circuit:

```
>>> from math import *
>>> from cmath import *
>>> r = complex(400,0)
>>> f = 60.0
>>> xc = 1/(2 * pi * f * 4.7e-6)
>>> zc = complex(0,-xc)
>>> xl = 2 * pi * f * 1.0
>>> zl = complex(0,xl)
>>> r + zc + zl
(400-187.38811239154882j)
>>> 1/(1/r + 1/zc + 1/zl)
(355.837695813625+125.35793777619385j)
>>> polar(r + zc + zl)
(441.717448903332, -0.4381072059213295)
>>> abs(r + zc + zl)
441.717448903332
>>> phase(r + zc + zl)
-0.4381072059213295
>>> degrees(phase(r + zc + zl))
-25.10169387356105
```

When entering a value in rectangular form, we use the `complex()` function where the arguments are the real and imaginary quantities, respectively. If we had opted to enter the impedance values in polar form, we would have used the `rect()` function where the first argument is the magnitude and the second argument is the angle in radians. For example, we could have set the capacitor's impedance (`zc`) as $X_C \angle -90^o$ with the command `zc = rect(xc,radians(-90))` rather than with the command `zc = complex(0,-xc)` and it would have worked the same.

Note how Python defaults to rectangular form for complex quantities. Here we defined a 400 Ohm resistance as a complex value in rectangular form (400 +j0 Ω), then computed capacitive and inductive reactances at 60 Hz and defined each of those as complex (phasor) values ($0 - jX_c$ Ω and $0 + jX_l$ Ω, respectively). After that we computed total impedance in series, then total impedance in parallel. Polar-form representation was then shown for the series impedance (441.717 Ω $\angle -25.102^o$). Note the use of different functions to show the polar-form series impedance value: `polar()` takes the complex quantity and returns its polar magnitude and phase angle in *radians*; `abs()` returns just the polar magnitude; `phase()` returns just the polar angle, once again in radians. To find the polar phase angle in degrees, we nest the `degrees()` and `phase()` functions together.

The utility of Python's interpreter environment as a scientific calculator should be clear from these examples. Not only does it offer a powerful array of mathematical functions, but also unlimited

[5] A "phasor" is a voltage, current, or impedance represented as a complex number, either in rectangular or polar form.

assignment of variables as well as a convenient text record[6] of all calculations performed which may be easily copied and pasted into a text document for archival.

It is also possible to save a set of Python commands to a text file using a text editor application, and then instruct the Python interpreter to execute it at once rather than having to type it line-by-line in the interpreter's shell. For example, consider the following Python program, saved under the filename `myprogram.py`:

```
x = 200
y = -560.5

print("Sum")
print(x + y)

print("Difference")
print(x - y)

print("Product")
print(x * y)

print("Quotient")
print(x / y)
```

As with C++, the interpreter will read this source code from left-to-right, top-to-bottom, just the same as you or I would read a document written in English. Interestingly, whitespace *is* significant in the Python language (unlike C++), but this simple example program makes no use of that.

To execute this Python program, I would need to type `python myprogram.py` and then press the Enter key at my computer console's prompt, at which point it would display the following result:

```
Sum
-360.5
Difference
760.5
Product
-112100.0
Quotient
-0.35682426405
```

As you can see, syntax within the Python programming language is simpler than C++, which is one reason why it is often a preferred language for beginning programmers.

[6] Like many command-line computing environments, Python's interpreter supports "up-arrow" recall of previous entries. This allows quick recall of previously typed commands for editing and re-evaluation.

If you are interested in learning more about computer programming in *any* language, you will find a wide variety of books and free tutorials available on those subjects. Otherwise, feel free to learn by the examples presented in these modules.

4.3 Modeling a diode using C++

Suppose we wished to graphically plot the voltage/current characteristic function for a PN semiconductor diode, based on the Shockley diode equation:

$$I = I_S \left(e^{\frac{qV}{nKT}} - 1 \right)$$

To plot this function manually we would "plug in" a wide range of values for V, calculate I for each one of those voltage values, and then plot those number pairs on a two-dimensional graph. Obtaining a high-resolution graph would require many such calculations, a tedious task at best to perform manually. This is precisely the sort of thing computers are good at: performing mathematical calculations *very quickly*.

The following C++ code evaluates the Shockley diode equation at 0.7 Volts:

```
#include <iostream>
#include <math.h>
using namespace std;

int main(void)
{
  float v, i, is, n, k, T, q;

  is = 20e-9;    // Reverse saturation current, 20 nA
  n = 1.3;       // Ideality factor, typically between 1 and 2
  k = 1.38e-23;  // Boltzmann's constant, 1.380 x 10^(-23) Joules per Kelvin
  T = 293.15;    // Absolute temperature, 293.15 Kelvin = 20 degrees C
  q = 1.602e-19; // Elementary charge of electron, 1.602 x 10^(-19) Coulombs

  v = 0.7;

  i = is * (exp(q * v / (n * k * T)) - 1);

  cout << "Current (A)" << endl;
  cout << i << endl;

  return 0;
}
```

The result, when compiled and executed is this:

```
Current (A)
36.4326
```

Let's analyze how this program works, discussing the following programming principles as we do:

- Preprocessor directives, namespaces
- The `main` function: return values, arguments
- Delimiter characters (e.g. `{ } ;`)
- Variable types (`float`), names, and declarations
- Variable assignment/initialization (`=`)
- Comments (`//`)
- Basic arithmetic (`+`, `-`, `*`, `/`)
- Arithmetic functions (`exp`)
- Printing text output (`cout`, `<<`, `endl`)
- Loops (`for`)
- Comparison (`<`)
- Order of execution
- Compiler commands
- Redirecting console text to files (`>`, `|`, `tee`)
- Nested loops

The first three lines (`#include` and `using namespace`) merely instruct the compiler software how to interpret many of the lines that follow. In this program the compiler is told to use the "standard" namespace, and to include the `iostream` and `math` header files which define certain instructions used later in the program such as `cout` and `exp`.

Every C++ program has a `main` function where execution begins. All of our code appears between the curly-brace ({ }) symbols belonging to the `main` function, those braces instructing the compiler where `main`'s content begins and ends.

The line beginning with `float` declares seven *floating-point* variables our program will use in its computation. This is where we define the names of our variables, and how the compiler will know how much of the computer's memory to reserve to store them all. Note how C++ permits the use of single-letter variable names as well as multi-letter names (e.g. `is`).

This line of code ends with a semicolon delimiter (`;`) telling the compiler where the line ends. You will notice most of the lines of code end with semicolons, exceptions being `main` (because its curly-brace symbols serve the same purpose) and the `#include` directives which are technically set-up instructions for the compiler and not executed at run-time.

The next five lines of code *initialize* several of our floating-point variables with values equal to physical constants necessary for the Shockley diode equation, using power-of-ten notation (e.g. `1.38e-23` is equivalent to 1.38×10^{-23}). In order to make this code more sensible to human readers, each initialization is immediately followed by a *comment*. In C and C++ any text following a double-slash (`//`) is completely ignored by the compiler, giving the programmer a way to include helpful notes for future reference. Comments are an excellent practice for any programmer in any language, as they improve the readability of the code for the benefit of anyone viewing it. This includes the person who originally wrote the code, who might appreciate having notes when they revisit that code years later!

Following those five initializing lines, we have one more to initialize our voltage variable (`v`) at 0.7 (Volts).

It is worth noting that we could have omitted several variables in this program and *all* of the initialization lines simply by placing those exact same numerical values within the math statement as shown in the following example:

```
#include <iostream>
#include <math.h>
using namespace std;

int main(void)
{
  float i;

  i = 20e-9 * (exp(1.602e-19 * 0.7 / (1.3 * 1.38e-23 * 293.15)) - 1);

  cout << "Current (A)" << endl;
  cout << i << endl;

  return 0;
}
```

This condensed version, you might agree, is not quite as easy to understand as the original. Once again, we see the merits of extra coding for the sake of *human-readability*. There are times when terse code is better, for example when writing software for *microcontrollers* which have very limited memory and processing speed. However, for the majority of modern computer programming applications, the benefit we reap by writing easy-to-understand code usually outweighs any performance costs.

Moving along in our exploration of the code, we come to the line where diode current gets calculated. This is, obviously, an arithmetic function involving multiplication (`*`), division (`/`), subtraction (`-`), and an exponential function (e^x, coded as `exp()` where the exponent value x fits within the parentheses). In the C and C++ languages, a single "equals" symbol (`=`) means *assignment*, with the computed value on the right-hand side stored in the variable on the left-hand side. In other words, we should read the `=` symbol as "*set equal to*".

The `cout` lines instruct the computer to print text to the *console* of the computer so that people can see the results. Anything directed to `cout` within quotation marks will be printed in its literal form. Any variable directed to `cout` (e.g. `cout << i`) will have its numerical value printed to the console. `endl` instructs the computer to "end the line" of text on the screen – without these the two `cout` instructions would print as a run-on sentence rather than as two separate lines. The fact that the two `cout` instructions appear as two different lines *of code* does not necessarily mean their outputs will display as two different lines of text *on the console*. This must be explicitly instructed by using the `endl` characters.

Finally, the `main` function concludes with a `return` statement instructing the computer to

generate a numerical value at the function's successful completion. This is not strictly necessary, but it is a good programming practice. Note how the *integer* variable type precedes the `main` label near the top of the program, and how the numerical value (zero) is an integer number.

Now that we have explored how this simple program functions, it's time to make some dramatic improvements. First and foremost, we should remedy the fact that all it does is calculate diode current for *a single* voltage value (0.7 Volts). If our goal is to plot the characteristic function of a diode, we will need a many current calculations for many different voltage values. It would be tedious to repeatedly edit the source code and recompile just to receive one current value per execution of the program. What would be much better would be a program that does this repetition internally. C and C++ offer a very effective way to do this, called a `for` loop, shown in the following program example:

```
#include <iostream>
#include <math.h>

using namespace std;

int main (void)
{
  float v, i, is, n, k, T, q;

  is = 20e-9;    // Reverse saturation current, 20 nA
  n = 1.3;       // Ideality factor, typically between 1 and 2
  k = 1.38e-23;  // Boltzmann's constant, 1.380 x 10^(-23) Joules per Kelvin
  T = 293.15;    // Absolute temperature, 293.15 Kelvin = 20 degrees C
  q = 1.602e-19; // Elementary charge of electron, 1.602 x 10^(-19) Coulombs

  for (v = 0.0 ; v < 0.7 ; v = v + 0.05)
  {
    i = is * (exp (q * v / (n * k * T)) - 1);
    cout << v << " , " << i << endl;
  }

  return 0;
}
```

The `for` statement has a similar structure to the `main` function, in that it has parentheses containing information it requires for operation, followed by a collection of code lines encapsulated between curly-brace symbols. The indentation used in this example is not required by the programming language, but is included to make the program easier for humans to read: seeing each new set of braced ({ }) lines indented makes it clear to see which function or statement they belong to.

Three statements contained within the `for` statement's parentheses and separated by semicolons instruct the `for` loop how many times to execute. The variable `v` will begin at a value of zero, the loop will continue so long as `v` is less than 0.7 (Volts), and with each iteration `v` will be incremented by 0.05 (Volts). All lines of code contained between the `for` statement's braces will be executed in normal order (left-to-right, top-to-bottom) with each iteration.

As we see in this example, with each pass through the loop the computer calculates diode current, and then immediately prints both the voltage and current values to the console, each number pair separated by a comma character. When run, the program generates the following output:

```
0 , 0
0.05 , 7.17258e-08
0.1 , 4.00681e-07
0.15 , 1.90936e-06
0.2 , 8.82863e-06
0.25 , 4.05624e-05
0.3 , 0.000186102
0.35 , 0.000853592
0.4 , 0.00391489
0.45 , 0.0179549
0.5 , 0.0823465
0.55 , 0.377665
0.6 , 1.73208
0.65 , 7.94383
```

These coordinate pairs are now ready to be plotted on a graph, the result being a curve for the diode's characteristic voltage/current function. Unfortunately, the C++ language does not include a standard graphics library, and so instructing the computer to plot these values to a screen requires much more advanced programming than is worth the effort to present here[7].

However, there is a simple solution useful on any computer with standard business spreadsheet software installed (e.g. Microsoft Excel). We can run our code, and *redirect* its output from the console to a text file to make it accessible to other software applications. All modern spreadsheets are capable of reading plain-text data in *comma-delimited* format (i.e. numerical values separated by comma characters), and also capable of generating *scatter plots* based on columns of numbers[8].

[7] Many graphics libraries for C++ are *system-dependent*, which means they may not work on all computers, on any operating system software. My goal in presenting programming examples is to show code that will execute on *any* computer system.

[8] This technique of using a spreadsheet application as a visual *front-end* for text-based data is very useful, and once having mastered it you will find many practical applications on your own. As you can see, writing C++ code to generate columns of numbers is fairly simple, which means in combination with a spreadsheet you have a ready-made graphical interface for any code you might wish to write.

The following instructions will show you how to accomplish this redirection. Suppose we compiled our source code (located in a file named `diode.cpp`) with the following command:

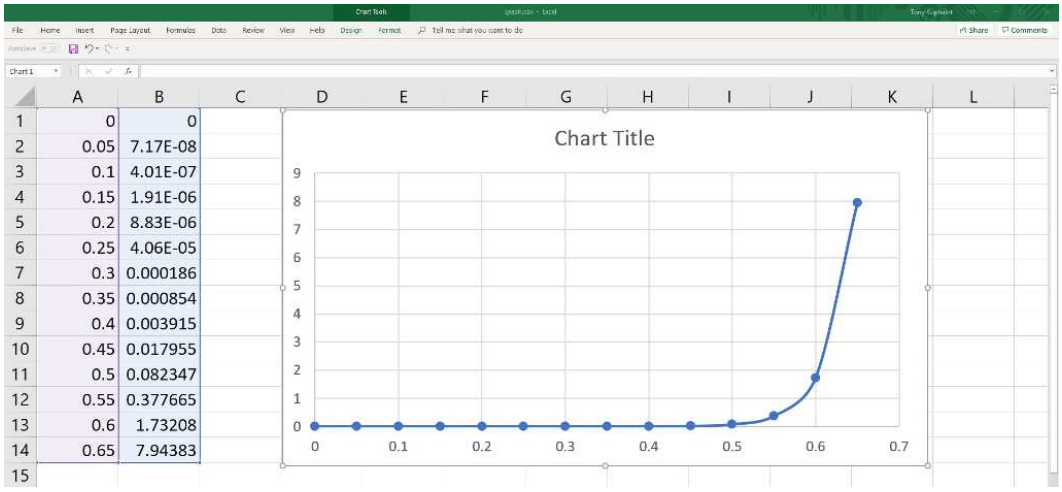
```
g++ -o diode.exe diode.cpp
```

Instead of running the executable file by typing `./diode.exe` we instead append that command with a *redirection* character and the name of a new file we wish the text to go into:

```
./diode.exe > graph.csv
```

Our simulated voltage and current data will now be saved in a file named `graph.csv`. We choose the filename suffix `.csv` because it stands for *comma-separated values* which any modern spreadsheet software should understand. After opening this file using the spreadsheet application, we will see the voltage and current values appearing in separate columns of the workbook. Simply highlight those columns and then command the spreadsheet to insert a graph (i.e. chart) of the scatter-plot style.

If you are using a computer system lacking a command-line interface, and therefore cannot redirect your C++ program's output, you still have the option of selecting and copying the text using the computer's mouse (left-click and drag over the text, then press `<Ctrl>-C` to copy) and then pasting that text (press `<Ctrl>-V`) into an open spreadsheet window. The following screenshot shows the graphic plot generated from this text data, using Microsoft Excel version 2010:



	A	B
1	0	0
2	0.05	7.17E-08
3	0.1	4.01E-07
4	0.15	1.91E-06
5	0.2	8.83E-06
6	0.25	4.06E-05
7	0.3	0.000186
8	0.35	0.000854
9	0.4	0.003915
10	0.45	0.017955
11	0.5	0.082347
12	0.55	0.377665
13	0.6	1.73208
14	0.65	7.94383

Another option we have with `for` loops is *nesting* them inside of each other. This is useful in our diode-modeling program for plotting current as a function of *two* variables, for instance voltage (V) and the ideality factor (n). Examine the following C++ code example:

```
#include <iostream>
#include <math.h>

using namespace std;

int main (void)
{
  float v, i, is, n, k, T, q;

  is = 20e-9;    // Reverse saturation current, 20 nA
  n = 1.3;       // Ideality factor, typically between 1 and 2
  k = 1.38e-23;  // Boltzmann's constant, 1.380 x 10^(-23) Joules per Kelvin
  T = 293.15;    // Absolute temperature, 293.15 Kelvin = 20 degrees C
  q = 1.602e-19; // Elementary charge of electron, 1.602 x 10^(-19) Coulombs

  for (v = 0.0 ; v < 0.7 ; v = v + 0.05)
  {
    cout << v << " , " ;
    for (n = 1.2 ; n < 1.5 ; n = n + 0.1)
    {
      i = is * (exp (q * v / (n * k * T)) - 1);
      cout << i << " , " ;
    }
    cout << endl;
  }

  return 0;
}
```

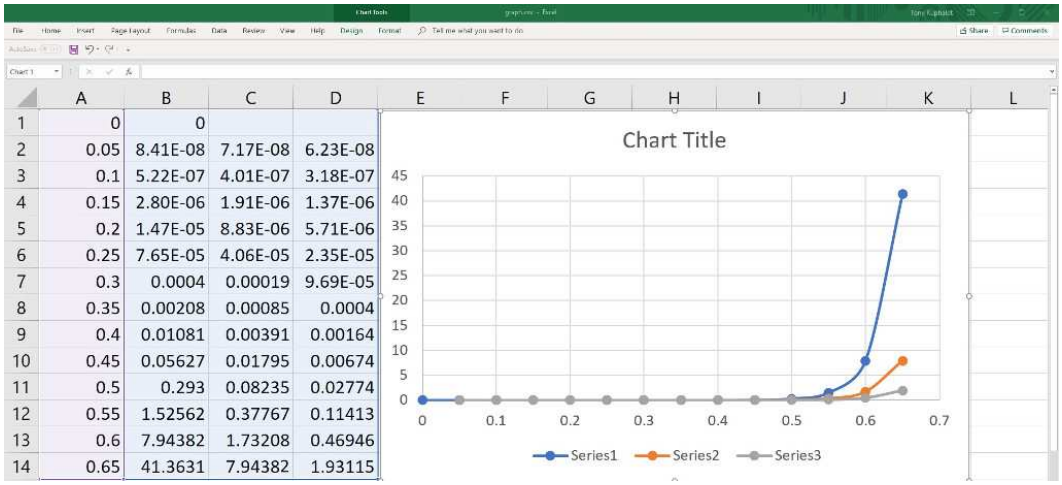
This program is written with two `for` loops: an *outer loop* increasing `v` by 0.05 Volt increments, and an *inner loop* incrementing `n` from 1.2 to 1.4 in increments of 0.1. Perhaps the most confusing aspect of this code are the `cout` statements. For each new iteration of the outer loop it prints the voltage value without an end-line character, then as the inner loop cycles it calculated and prints current values for each n factor (at the same voltage), each of those current values preceded by a comma character. An `endl` character only gets printed at the very end of the outer loop, just before voltage gets incremented once more. The result of all this is a set of printed text lines, each line starting with the voltage value, followed by comma-separated current values (each representing a different ideality factor).

Here is what the output of this program looks like when printed to the console:

```
0 , 0 , 0 , 0
0.05 , 8.41389e-08 , 7.17258e-08 , 6.22705e-08
0.1 , 5.22246e-07 , 4.00681e-07 , 3.18422e-07
0.15 , 2.80345e-06 , 1.90936e-06 , 1.37211e-06
0.2 , 1.46815e-05 , 8.82862e-06 , 5.70646e-06
0.25 , 7.65301e-05 , 4.05623e-05 , 2.3536e-05
0.3 , 0.000398573 , 0.000186102 , 9.6878e-05
0.35 , 0.00207543 , 0.000853591 , 0.000398573
0.4 , 0.0108068 , 0.00391489 , 0.0016396
0.45 , 0.0562703 , 0.0179549 , 0.0067446
0.5 , 0.292997 , 0.0823464 , 0.0277442
0.55 , 1.52562 , 0.377664 , 0.114126
0.6 , 7.94382 , 1.73208 , 0.469462
0.65 , 41.3631 , 7.94382 , 1.93115
```

As with the simpler two-column output, this text data is ready to be entered into a spreadsheet application and graphically plotted as a "family" of curves representing diode current for three different diodes (with ideality factors of 1.2, 1.3, and 1.4).

A screenshot showing Microsoft Excel displaying this data as a "scatter plot" appears next:



	A	B	C	D
1	0	0		
2	0.05	8.41E-08	7.17E-08	6.23E-08
3	0.1	5.22E-07	4.01E-07	3.18E-07
4	0.15	2.80E-06	1.91E-06	1.37E-06
5	0.2	1.47E-05	8.83E-06	5.71E-06
6	0.25	7.65E-05	4.06E-05	2.35E-05
7	0.3	0.0004	0.00019	9.69E-05
8	0.35	0.00208	0.00085	0.0004
9	0.4	0.01081	0.00391	0.00164
10	0.45	0.05627	0.01795	0.00674
11	0.5	0.293	0.08235	0.02774
12	0.55	1.52562	0.37767	0.11413
13	0.6	7.94382	1.73208	0.46946
14	0.65	41.3631	7.94382	1.93115

Columns A through D are highlighted to select the data to be plotted. By default, Excel regards data in the first column as the *independent* variable (i.e. data for the graph's horizontal axis) and data in all subsequent columns as *dependent* variables for different functions. Each function is then named Series1, Series2, etc. and plotted in different colors on the same space. The result is our desired "family" of curves for diodes having three different ideality factors. Note how Excel has automatically re-scaled the vertical axis to accommodate the "tallest" of the three functions.

Chapter 5

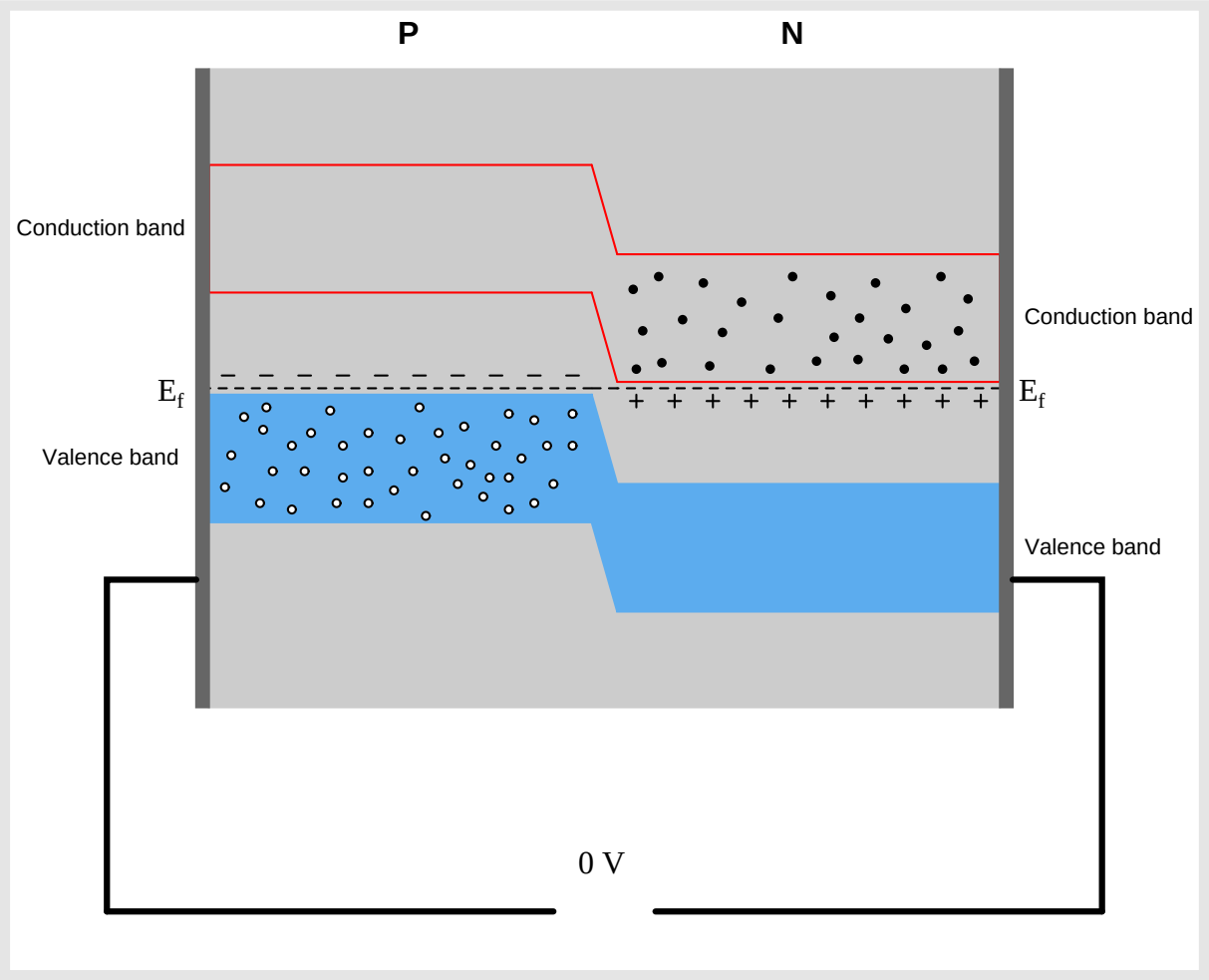
Animations

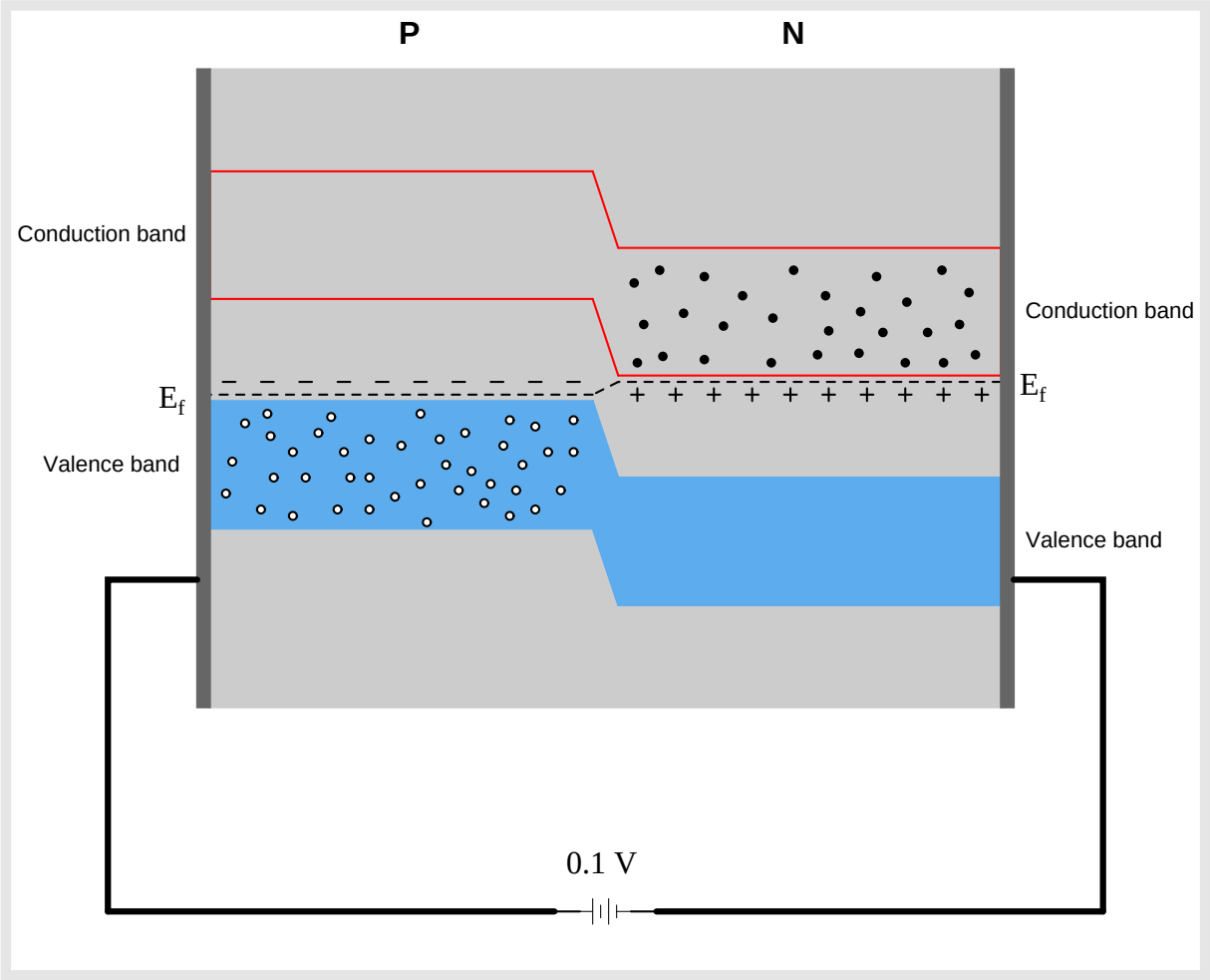
Some concepts are much easier to grasp when seen in *action*. A simple yet effective form of animation suitable to an electronic document such as this is a "flip-book" animation where a set of pages in the document show successive frames of a simple animation. Such "flip-book" animations are designed to be viewed by paging forward (and/or back) with the document-reading software application, watching it frame-by-frame. Unlike video which may be difficult to pause at certain moments, "flip-book" animations lend themselves very well to individual frame viewing.

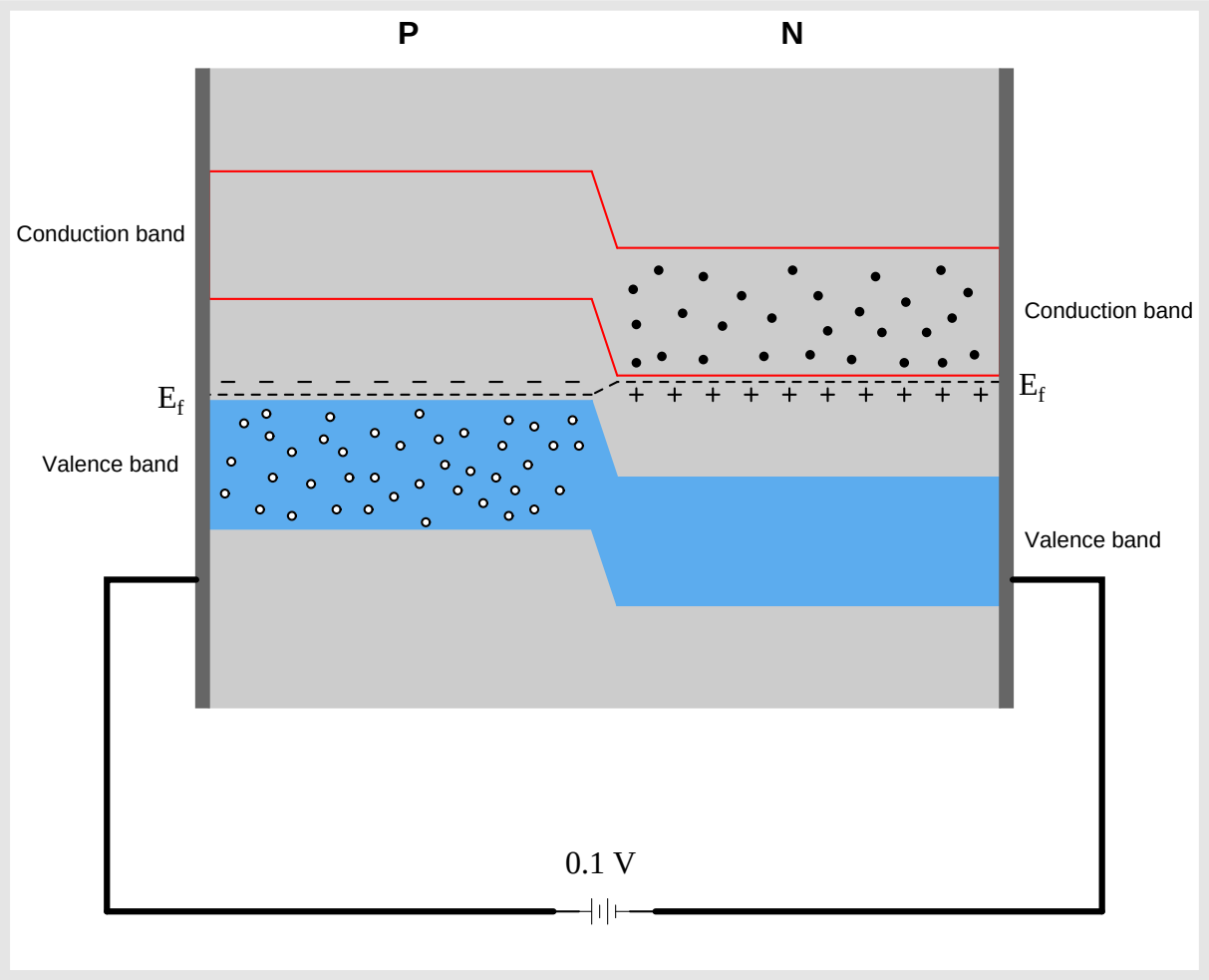
5.1 Animation of a forward-biased PN diode junction

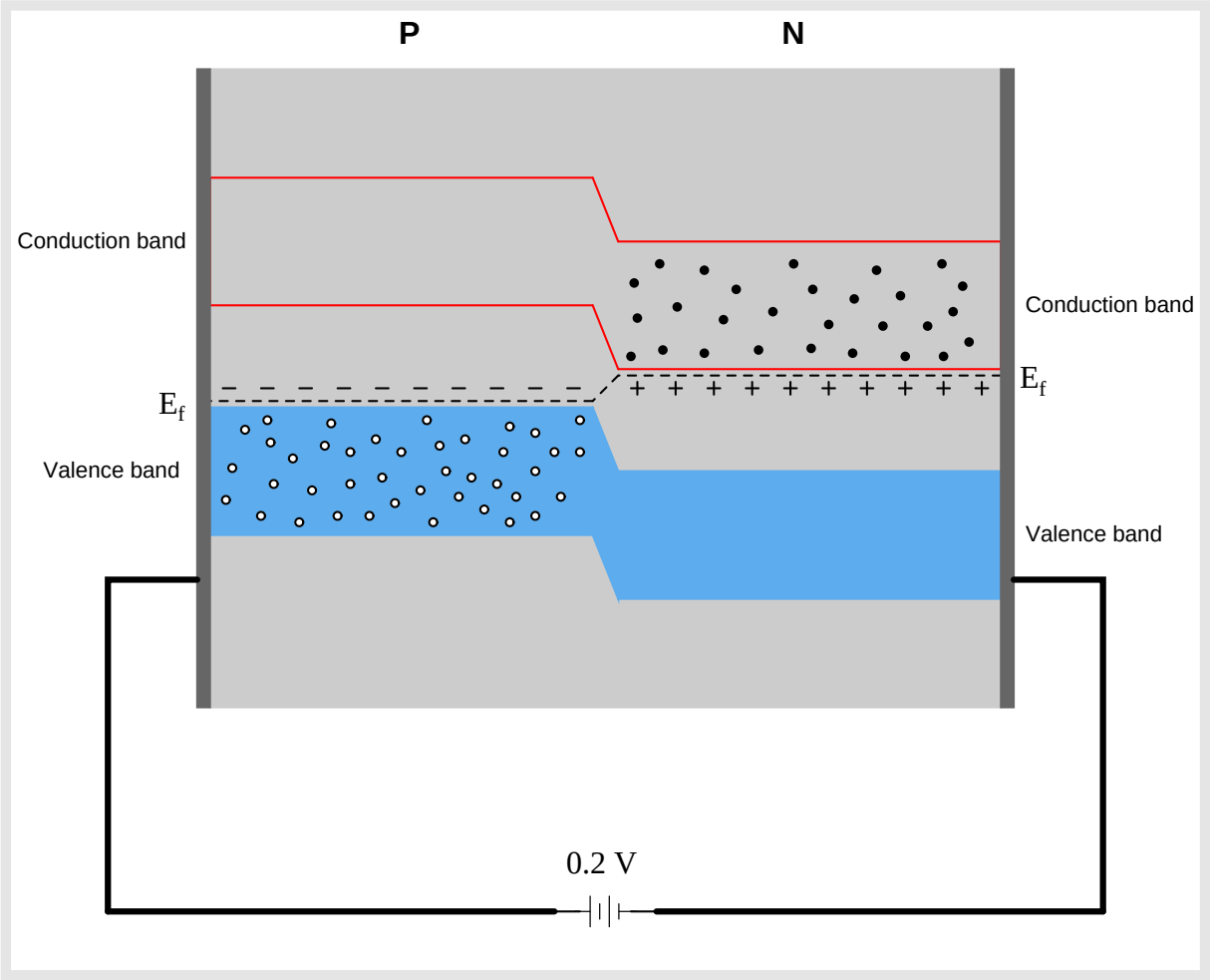
The following animation illustrates the behavior of mobile[1] charge carriers inside a semiconductor diode junction. A DC voltage source provides forward-biasing voltage to the PN junction, and this voltage source's value is progressively raised until the diode begins to conduct. Energy levels within the diode's semiconductor halves are shown relative to one another by means of height in the illustration.

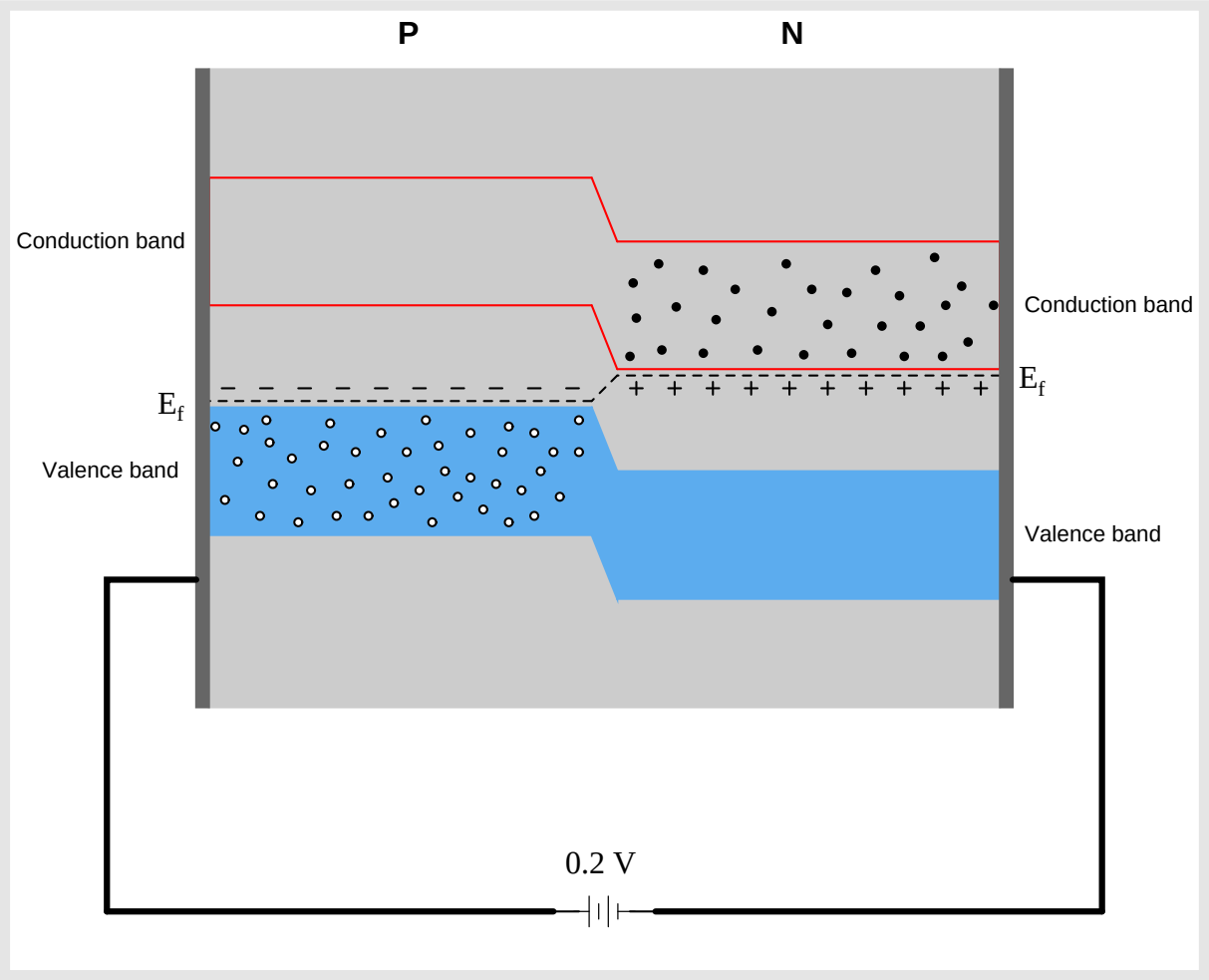
[1] In this animation, immobile electrons in the valence band are not shown – only the holes in that band.

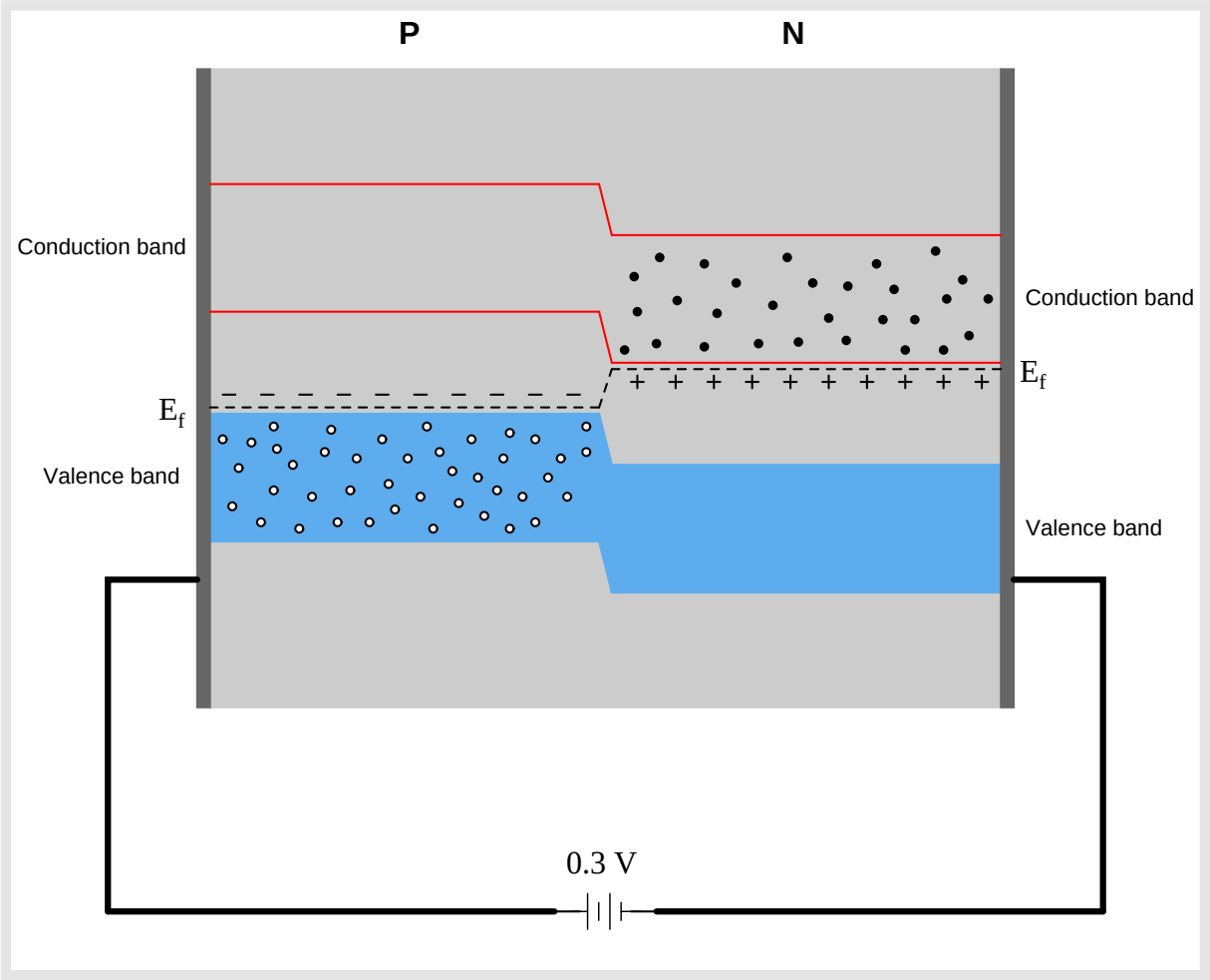

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0 V

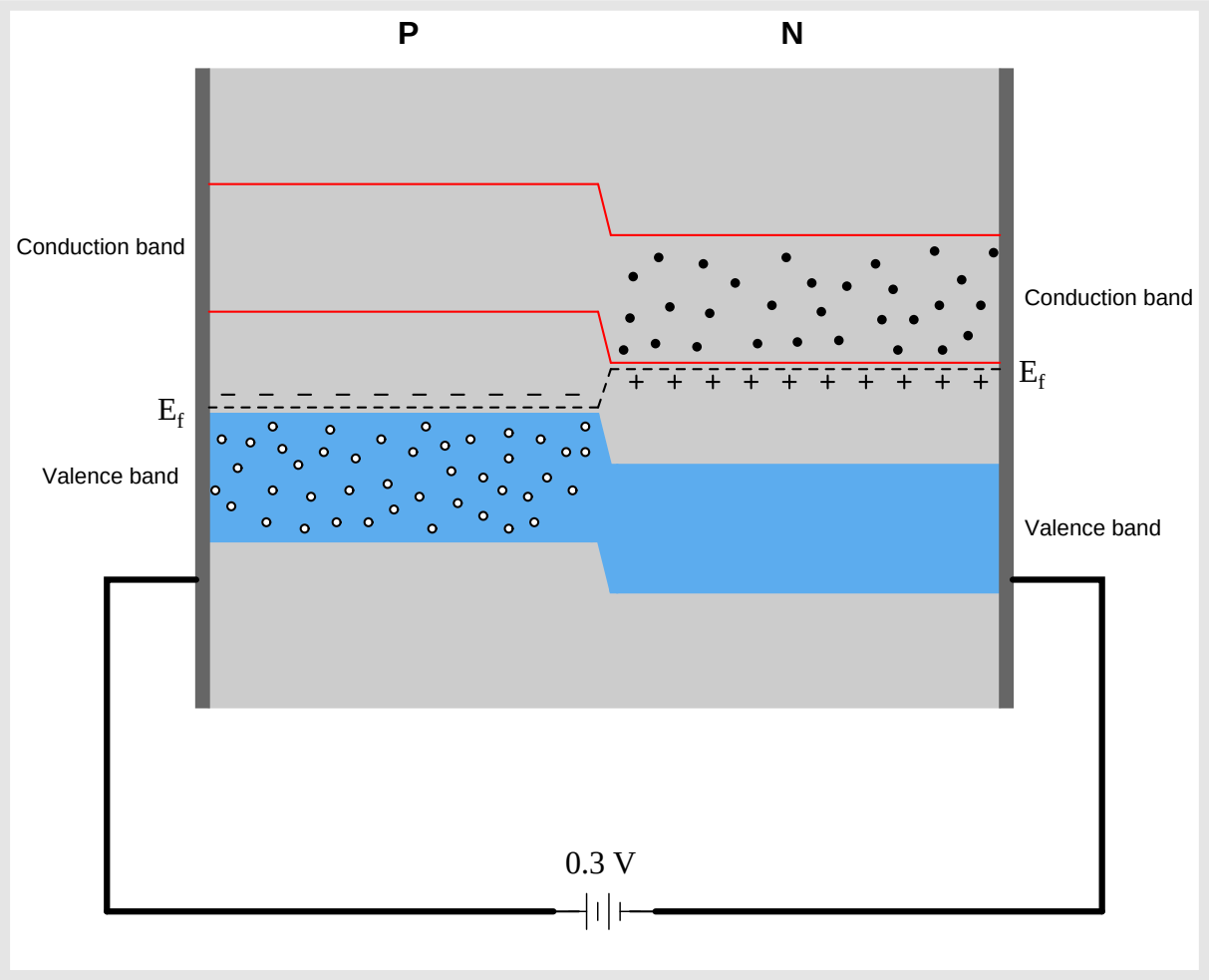

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.1 V

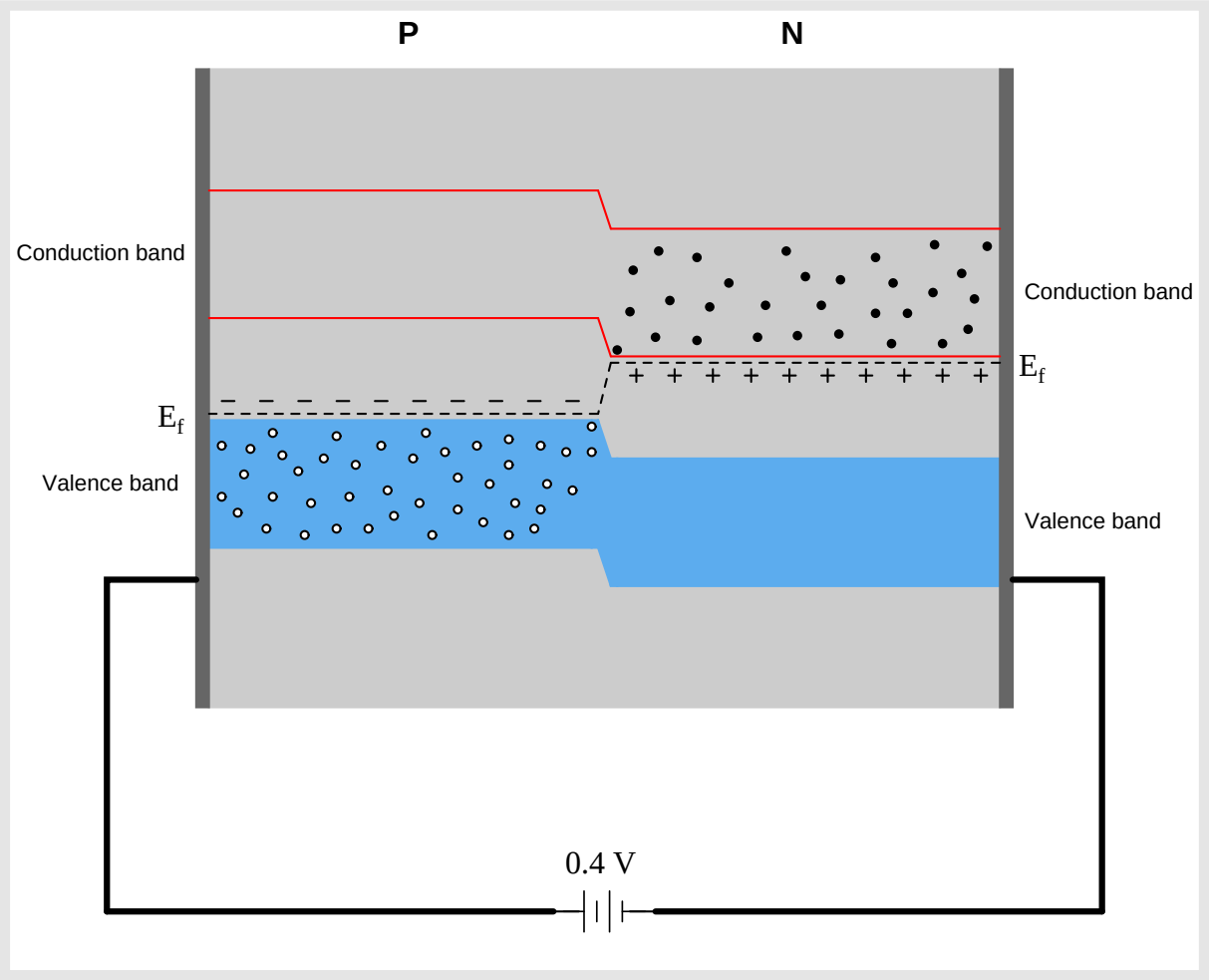

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.1 V

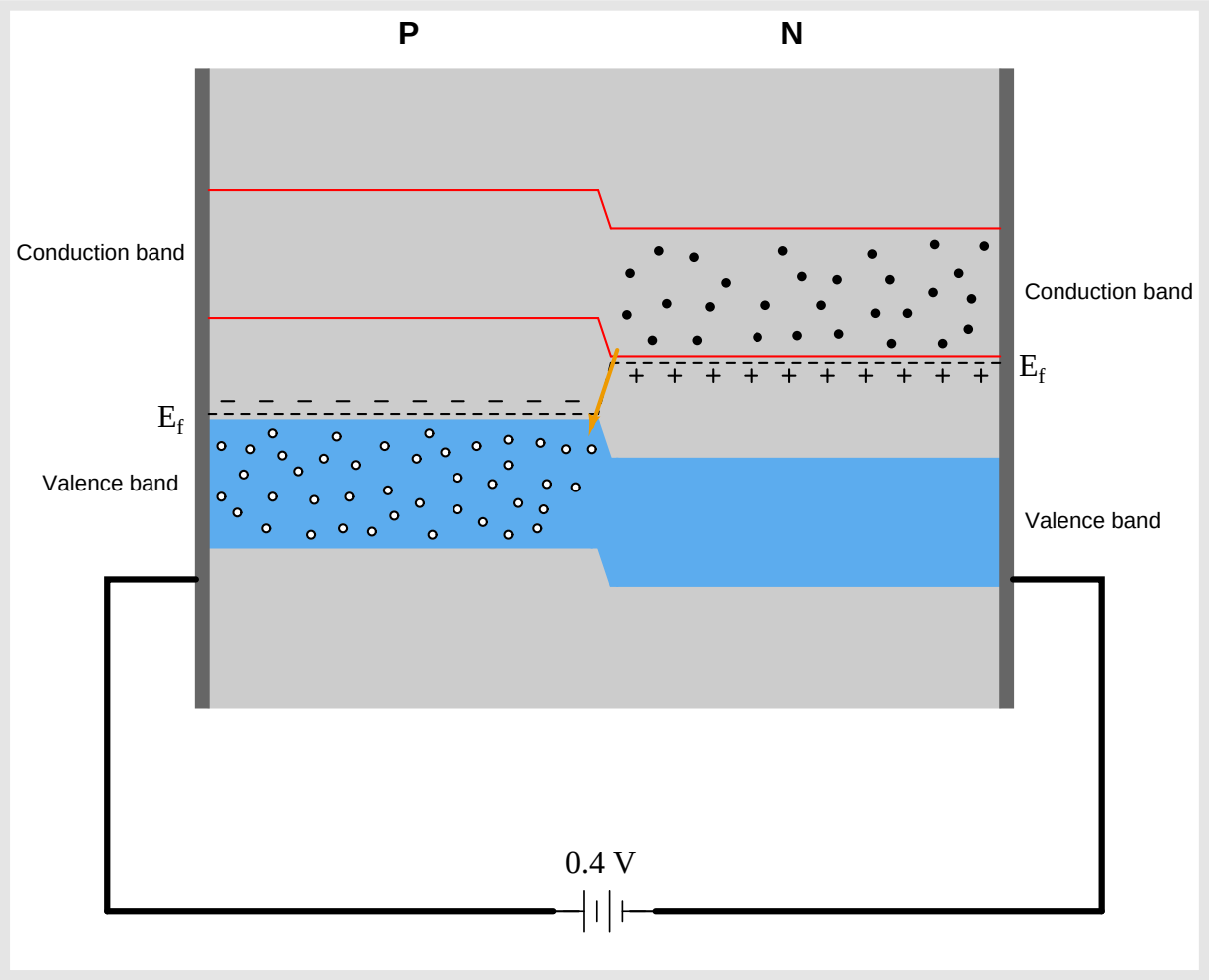

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.2 V

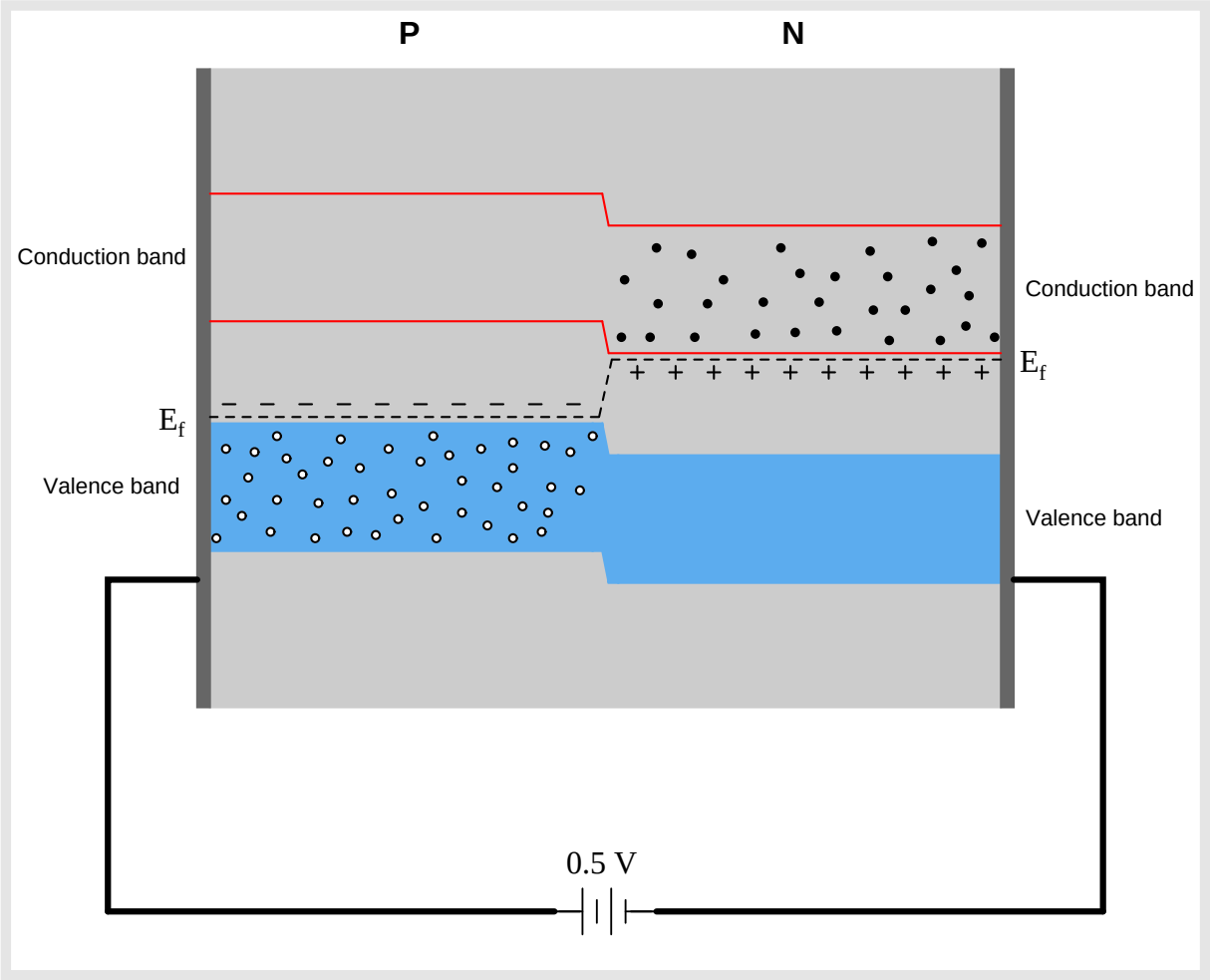

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.2 V

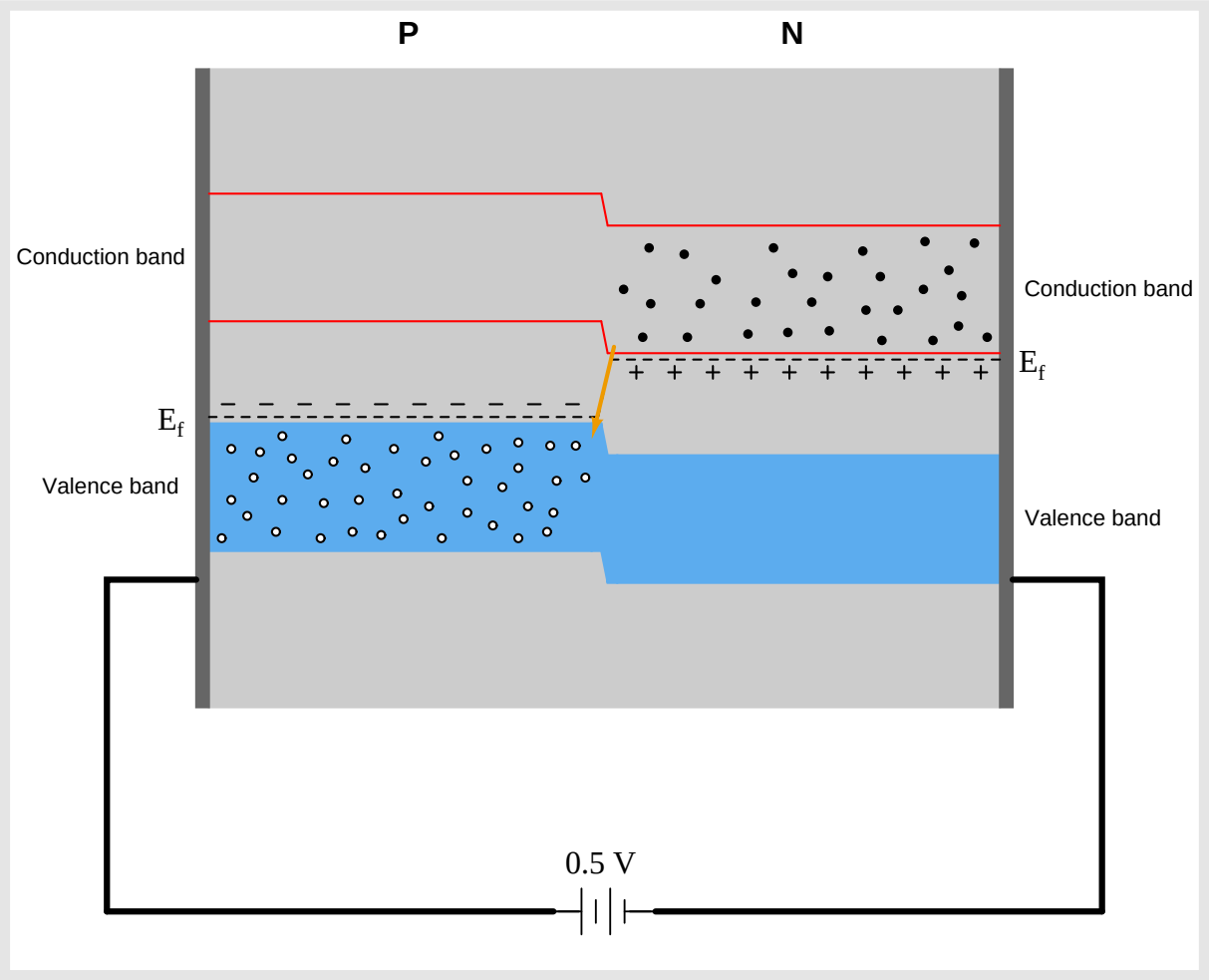

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.3 V

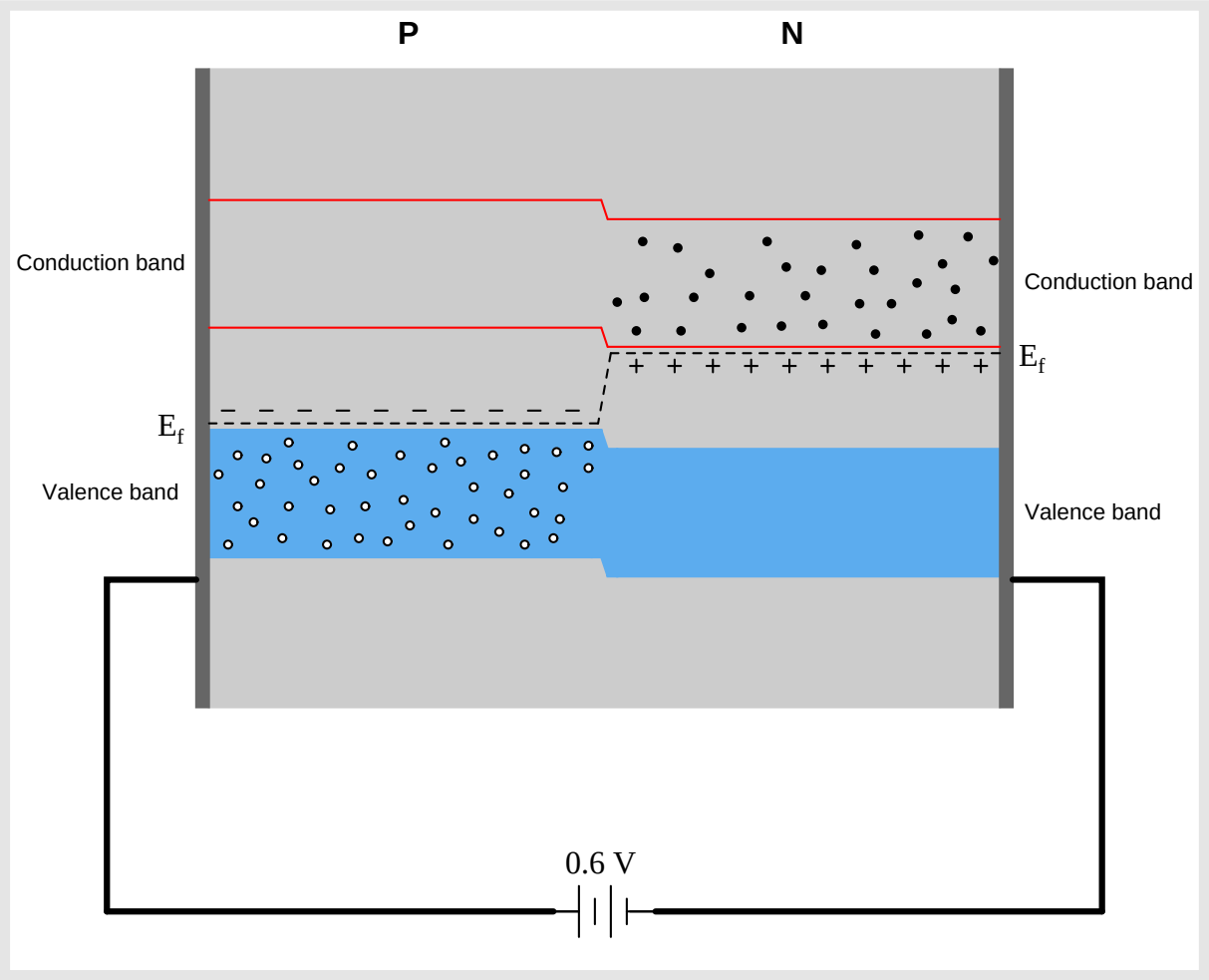

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.3 V

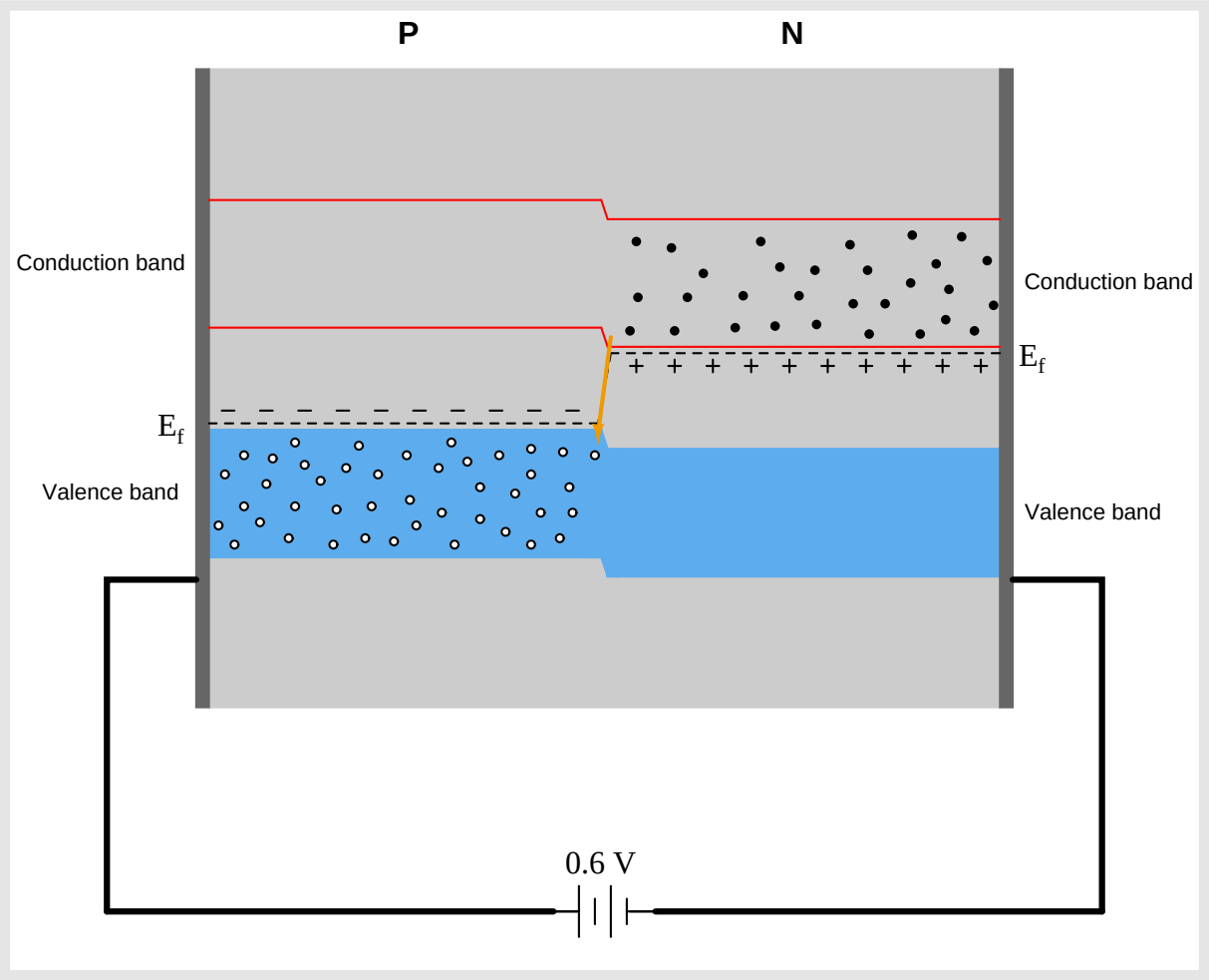

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.4 V

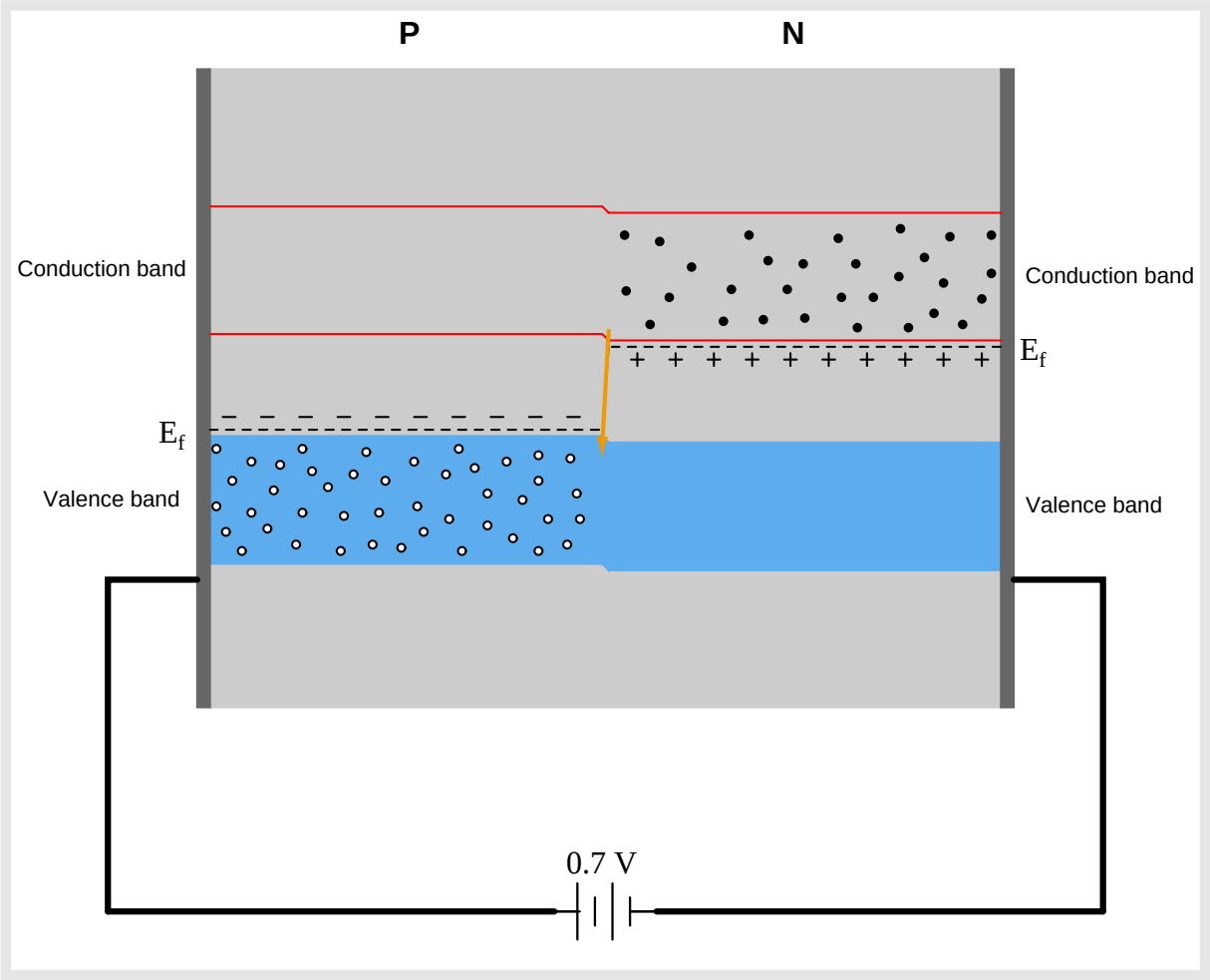

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.4 V

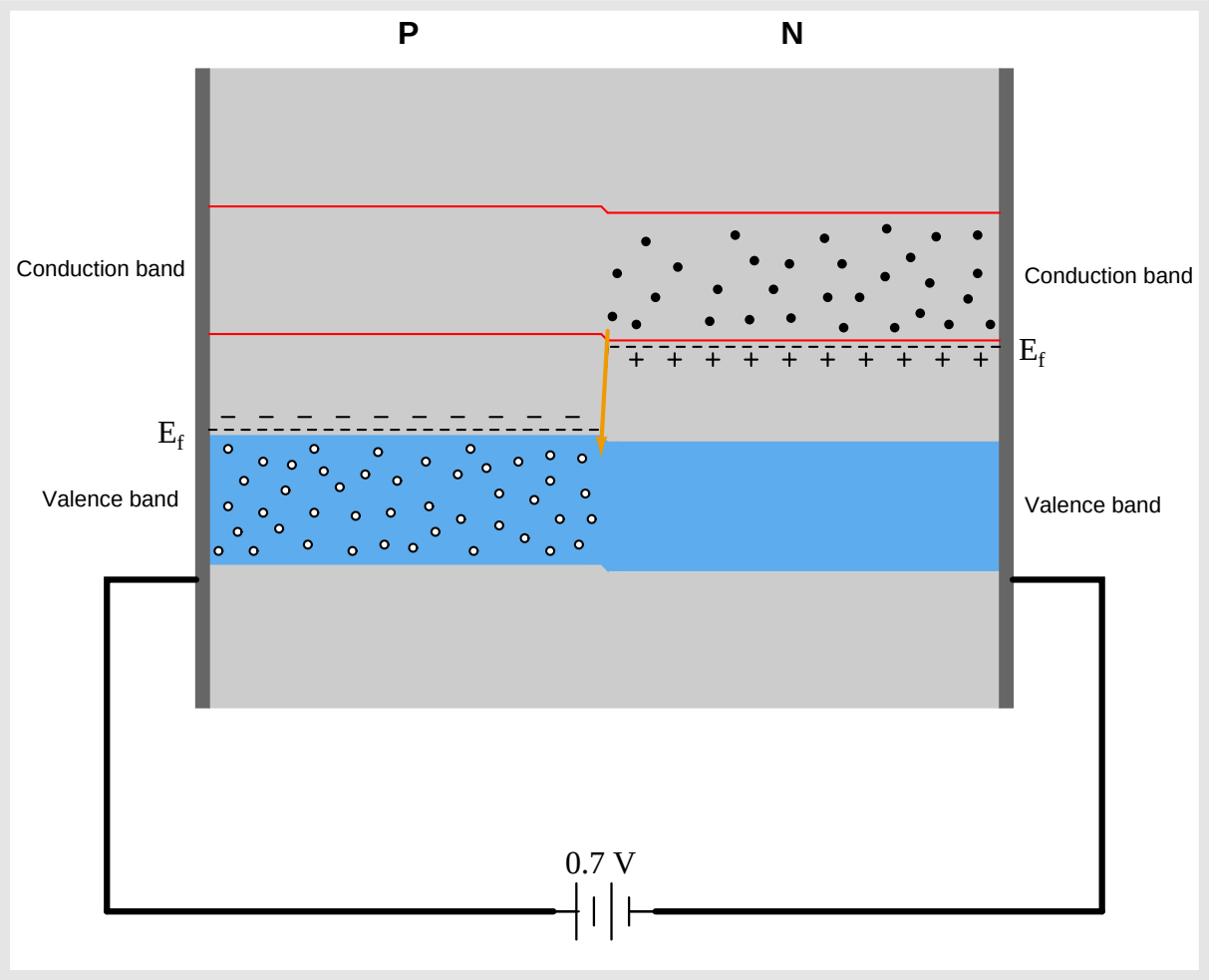

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.5 V

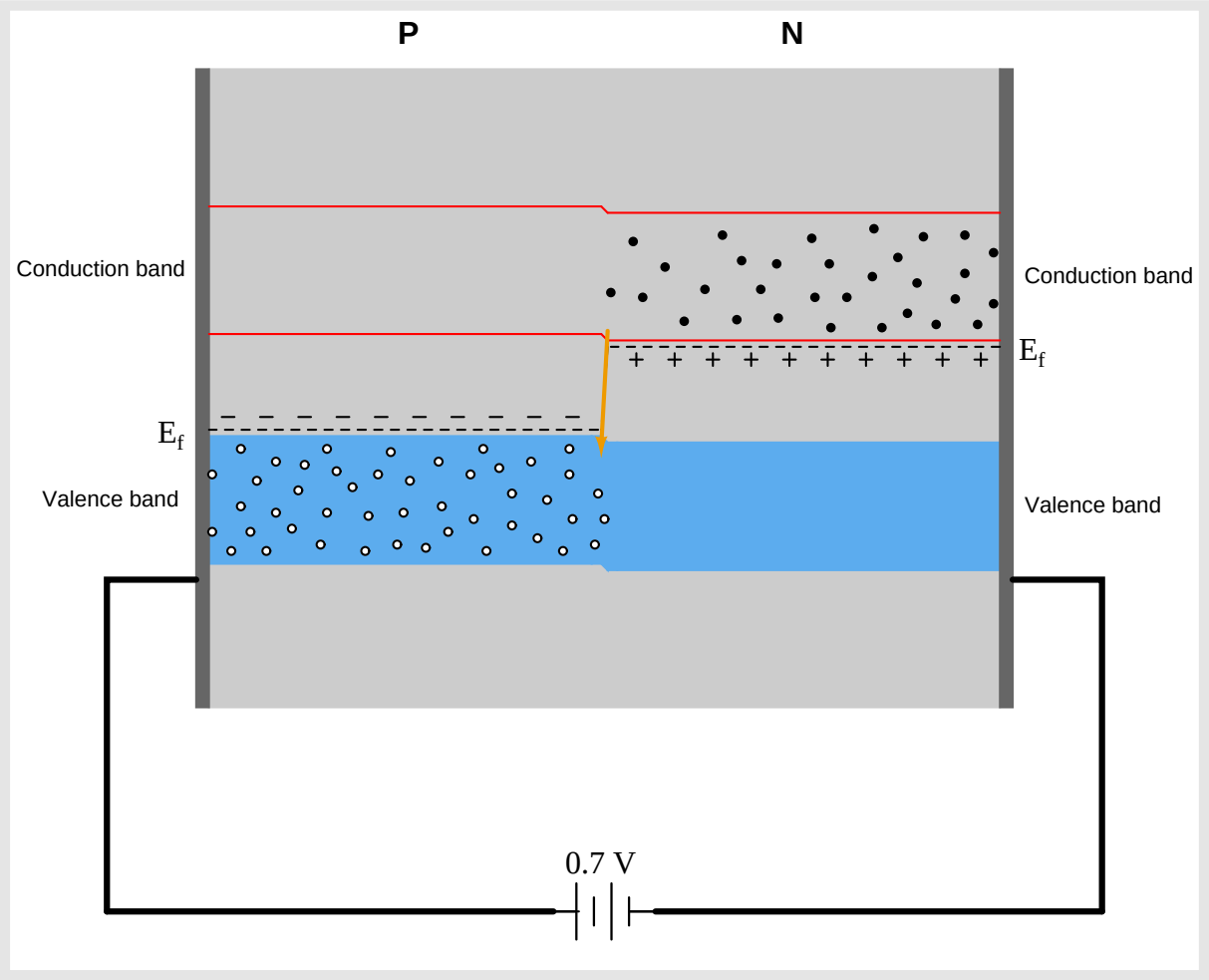

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.5 V

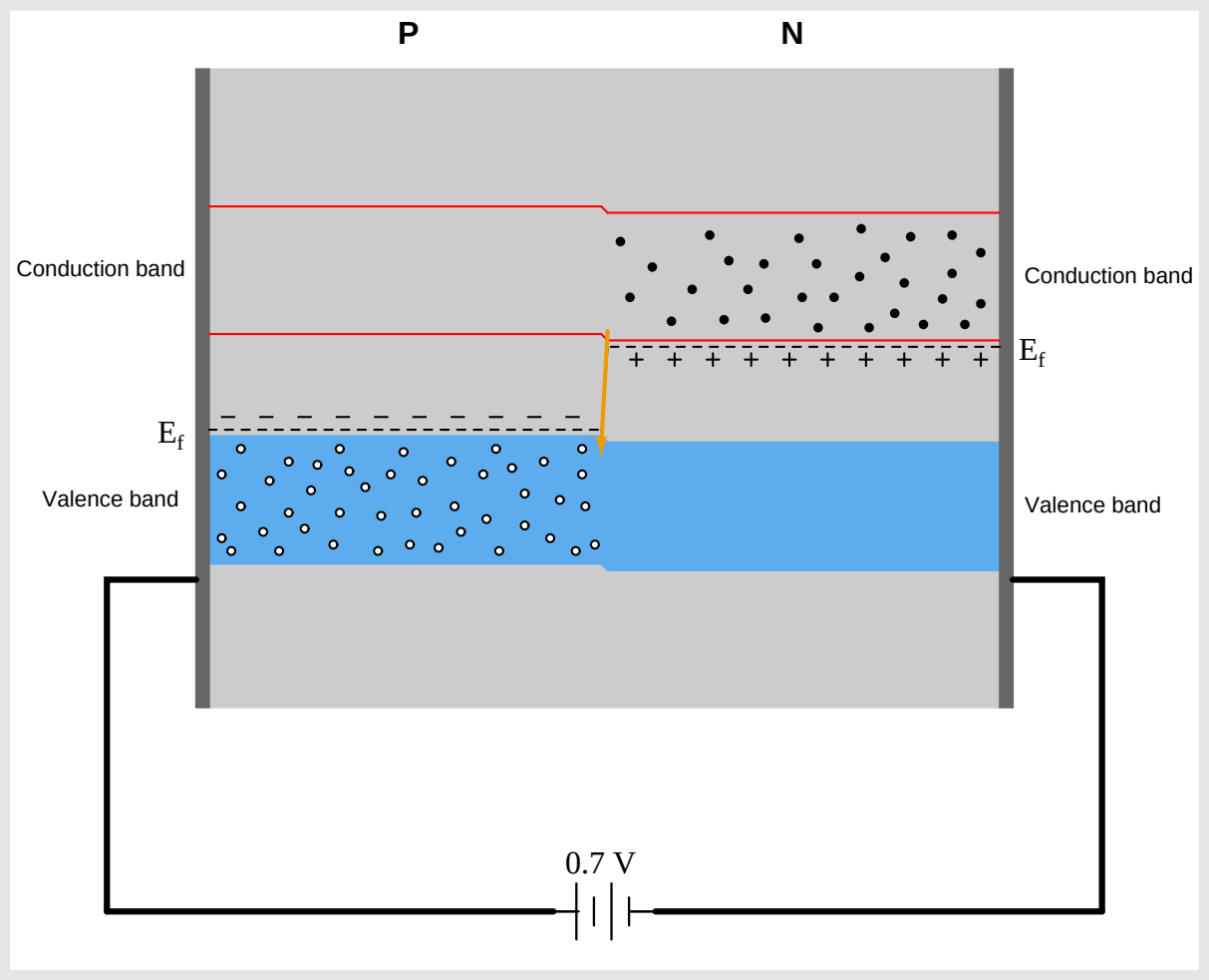

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.6 V

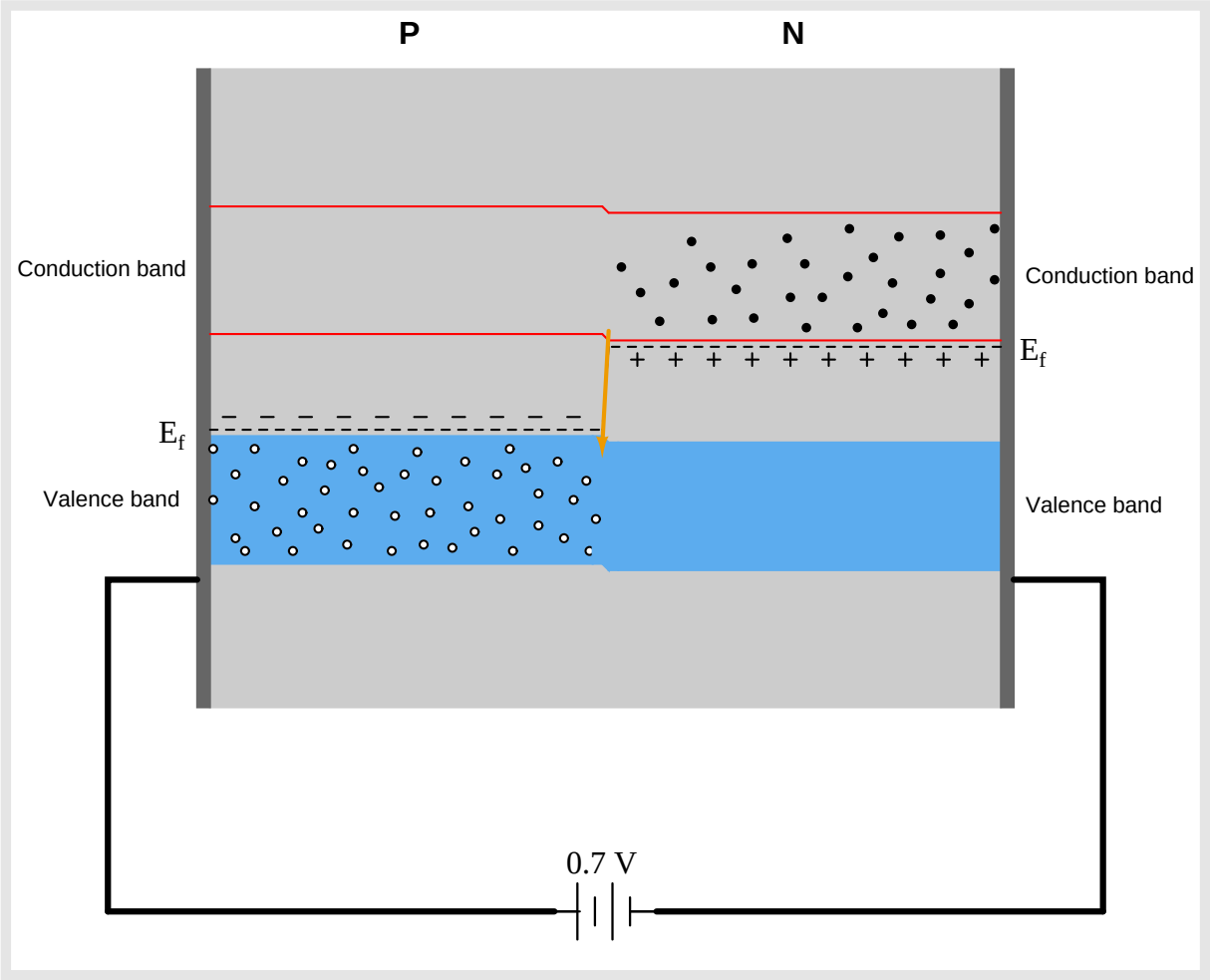

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.6 V

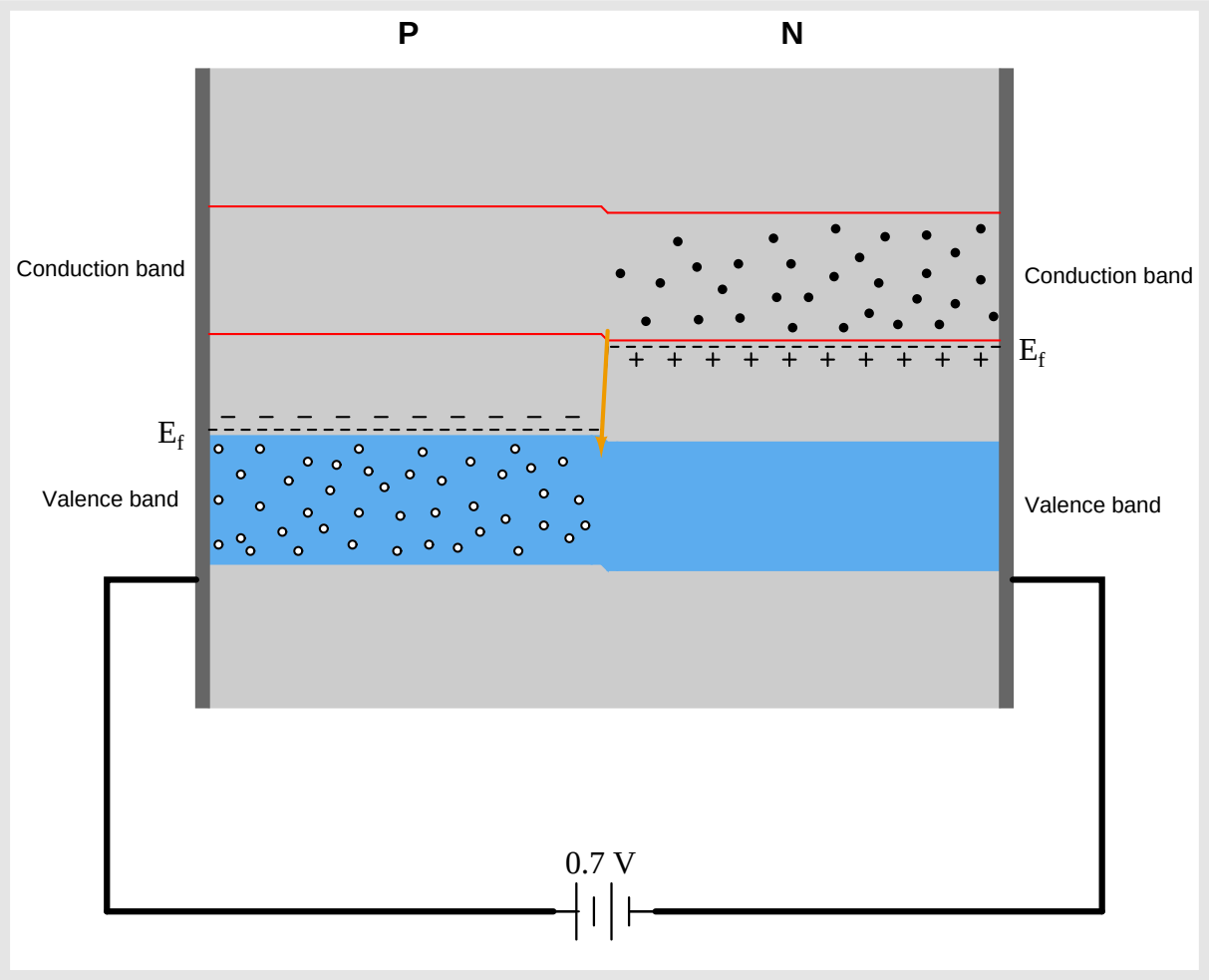

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V

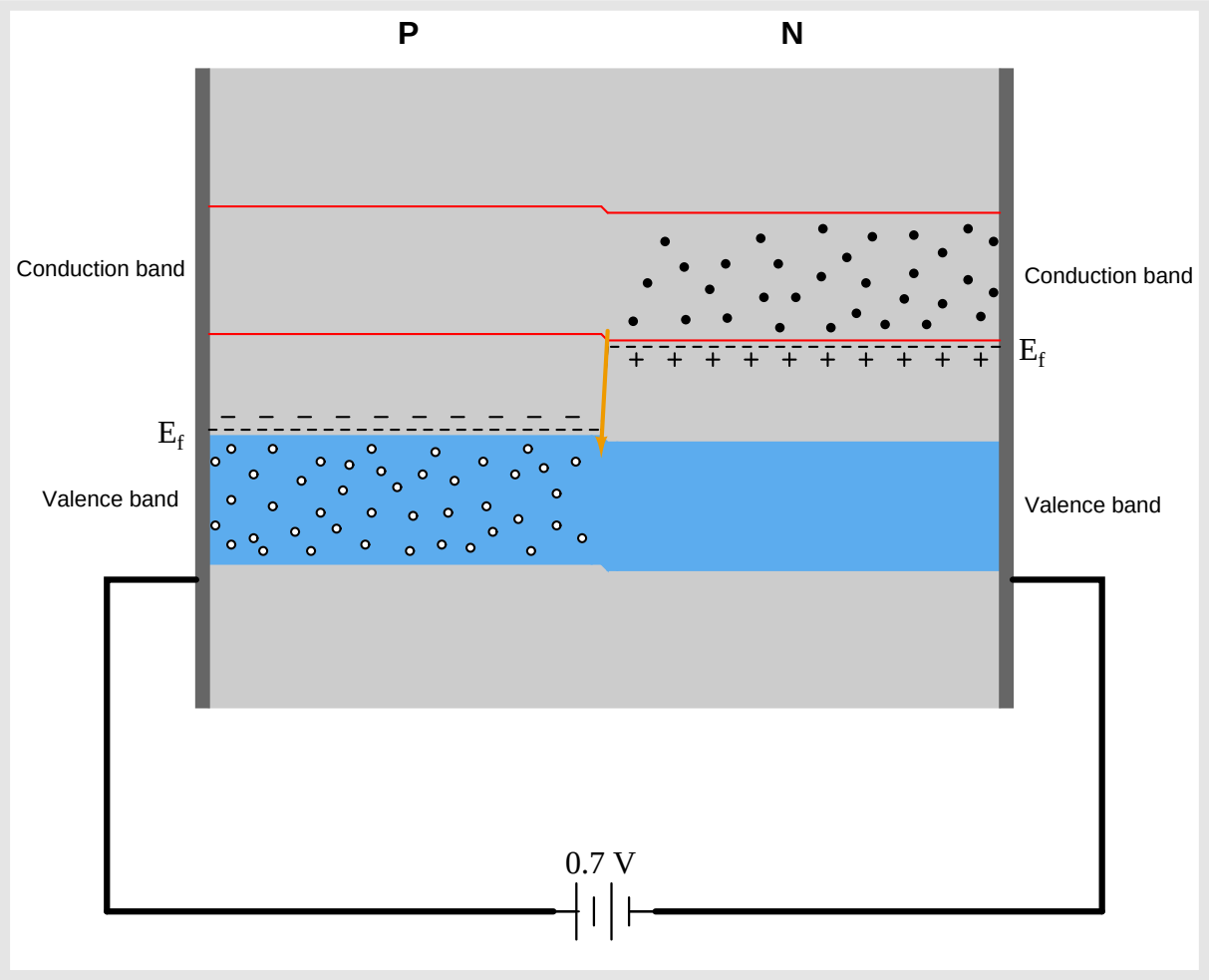

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V

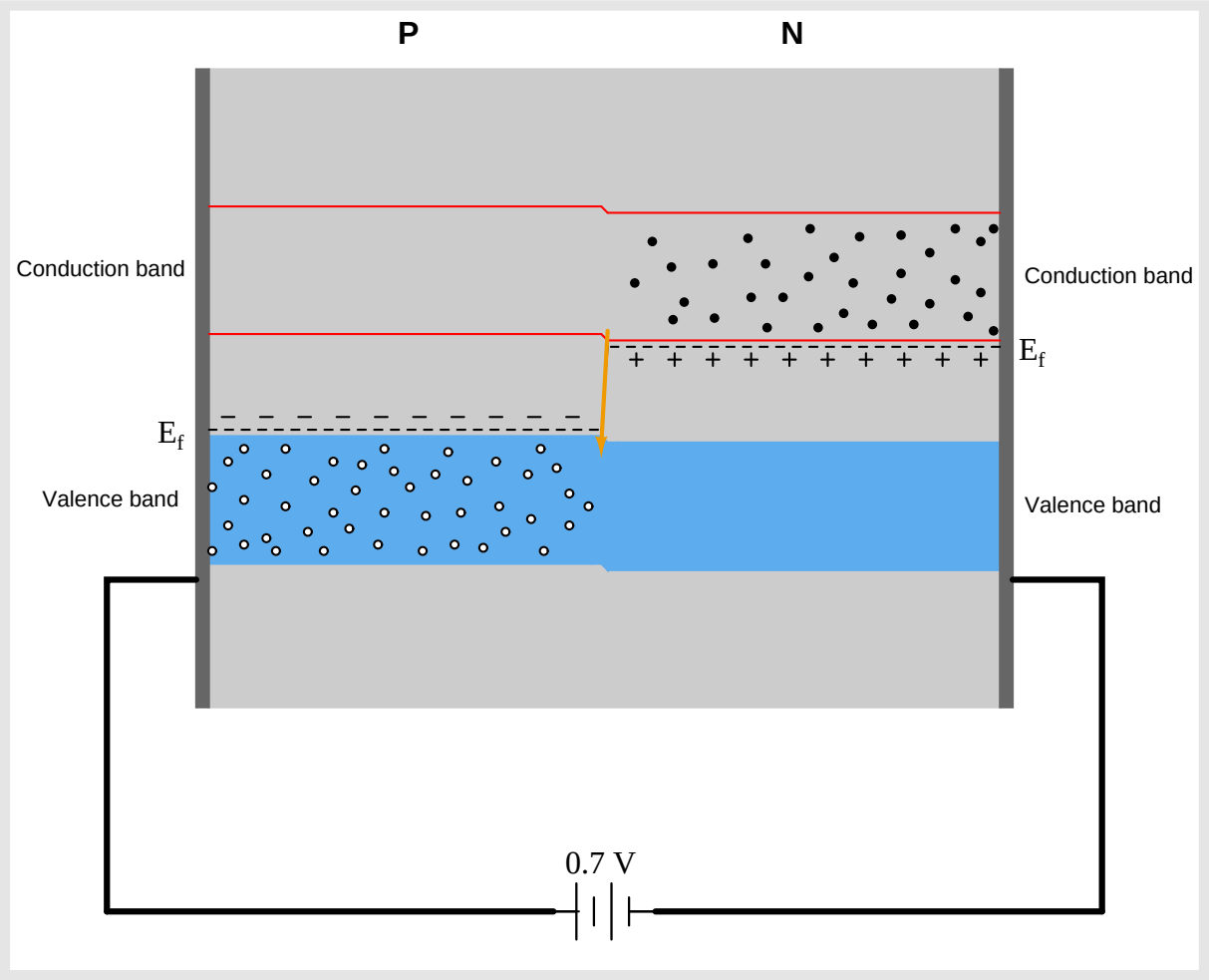

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V

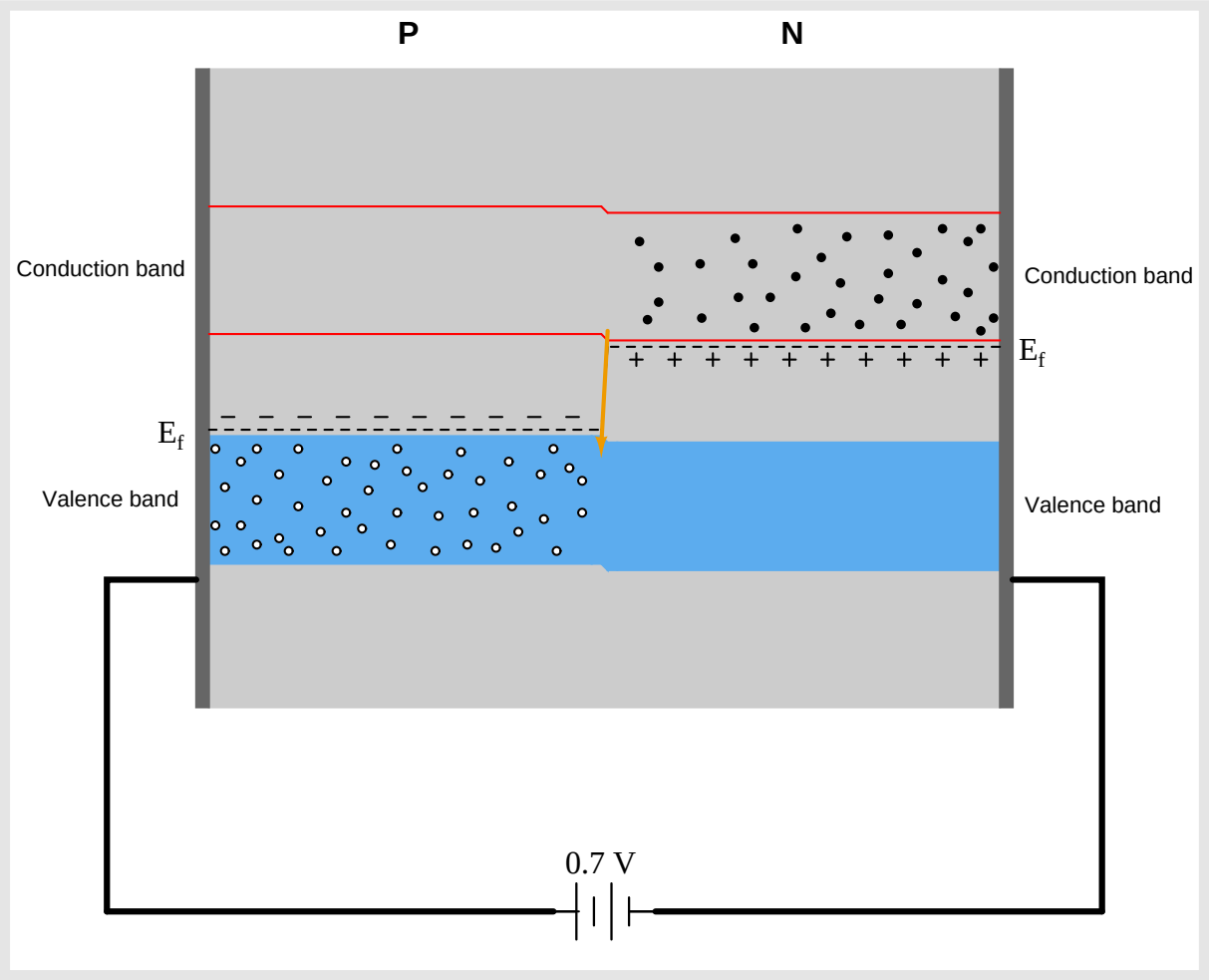

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V

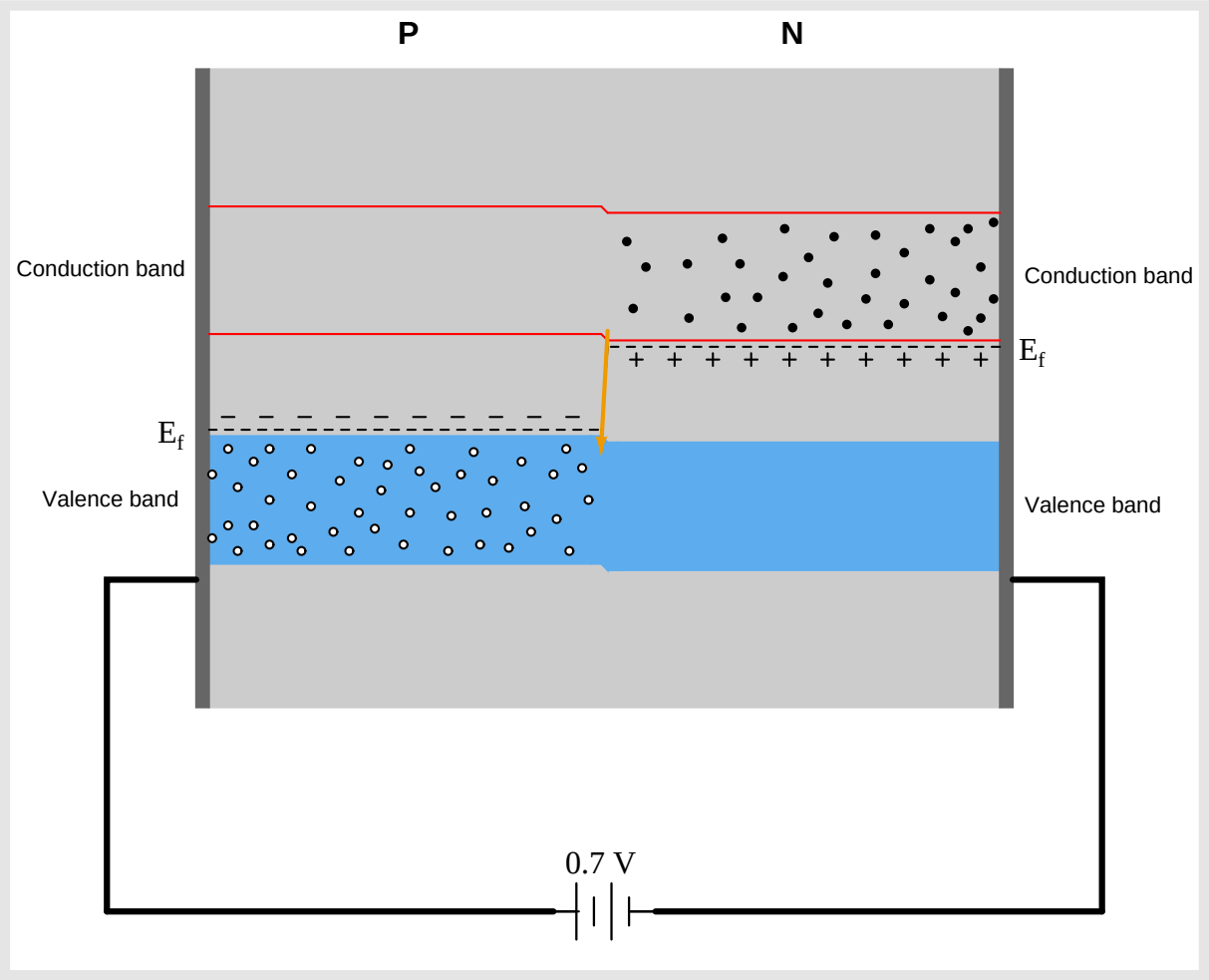

P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V


P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V


P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V


P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V


P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V


P
N
Conduction band
Conduction band
E_f
E_f
Valence band
Valence band
0.7 V

Chapter 6

Questions

This learning module, along with all others in the ModEL collection, is designed to be used in an inverted instructional environment where students independently read[1] the tutorials and attempt to answer questions on their own *prior* to the instructor's interaction with them. In place of lecture[2], the instructor engages with students in Socratic-style dialogue, probing and challenging their understanding of the subject matter through inquiry.

Answers are not provided for questions within this chapter, and this is by design. Solved problems may be found in the Tutorial and Derivation chapters, instead. The goal here is *independence*, and this requires students to be challenged in ways where others cannot think for them. Remember that you always have the tools of *experimentation* and *computer simulation* (e.g. SPICE) to explore concepts!

The following lists contain ideas for Socratic-style questions and challenges. Upon inspection, one will notice a strong theme of *metacognition* within these statements: they are designed to foster a regular habit of examining one's own thoughts as a means toward clearer thinking. As such these sample questions are useful both for instructor-led discussions as well as for self-study.

[1] Technical reading is an essential academic skill for any technical practitioner to possess for the simple reason that the most comprehensive, accurate, and useful information to be found for developing technical competence is in textual form. Technical careers in general are characterized by the need for continuous learning to remain current with standards and technology, and therefore any technical practitioner who cannot read well is handicapped in their professional development. An excellent resource for educators on improving students' reading prowess through intentional effort and strategy is the book textitReading For Understanding – How Reading Apprenticeship Improves Disciplinary Learning in Secondary and College Classrooms by Ruth Schoenbach, Cynthia Greenleaf, and Lynn Murphy.

[2] Lecture is popular as a teaching method because it is easy to implement: any reasonably articulate subject matter expert can talk to students, even with little preparation. However, it is also quite problematic. A good lecture always makes complicated concepts seem easier than they are, which is bad for students because it instills a false sense of confidence in their own understanding; reading and re-articulation requires more cognitive effort and serves to verify comprehension. A culture of teaching-by-lecture fosters a debilitating dependence upon direct personal instruction, whereas the challenges of modern life demand independent and critical thought made possible only by gathering information and perspectives from afar. Information presented in a lecture is ephemeral, easily lost to failures of memory and dictation; text is forever, and may be referenced at any time.

General challenges following tutorial reading

- Summarize as much of the text as you can in one paragraph of your own words. A helpful strategy is to explain ideas as you would for an intelligent child: as simple as you can without compromising too much accuracy.
- Simplify a particular section of the text, for example a paragraph or even a single sentence, so as to capture the same fundamental idea in fewer words.
- Where did the text make the most sense to you? What was it about the text's presentation that made it clear?
- Identify where it might be easy for someone to misunderstand the text, and explain why you think it could be confusing.
- Identify any new concept(s) presented in the text, and explain in your own words.
- Identify any familiar concept(s) such as physical laws or principles applied or referenced in the text.
- Devise a proof of concept experiment demonstrating an important principle, physical law, or technical innovation represented in the text.
- Devise an experiment to disprove a plausible misconception.
- Did the text reveal any misconceptions you might have harbored? If so, describe the misconception(s) and the reason(s) why you now know them to be incorrect.
- Describe any useful problem-solving strategies applied in the text.
- Devise a question of your own to challenge a reader's comprehension of the text.

General follow-up challenges for assigned problems

- Identify where any fundamental laws or principles apply to the solution of this problem, especially before applying any mathematical techniques.
- Devise a thought experiment to explore the characteristics of the problem scenario, applying known laws and principles to mentally model its behavior.
- Describe in detail your own strategy for solving this problem. How did you identify and organized the given information? Did you sketch any diagrams to help frame the problem?
- Is there more than one way to solve this problem? Which method seems best to you?
- Show the work you did in solving this problem, even if the solution is incomplete or incorrect.
- What would you say was the most challenging part of this problem, and why was it so?
- Was any important information missing from the problem which you had to research or recall?
- Was there any extraneous information presented within this problem? If so, what was it and why did it not matter?
- Examine someone else's solution to identify where they applied fundamental laws or principles.
- Simplify the problem from its given form and show how to solve this simpler version of it. Examples include eliminating certain variables or conditions, altering values to simpler (usually whole) numbers, applying a limiting case (i.e. altering a variable to some extreme or ultimate value).
- For quantitative problems, identify the real-world meaning of all intermediate calculations: their units of measurement, where they fit into the scenario at hand. Annotate any diagrams or illustrations with these calculated values.
- For quantitative problems, try approaching it qualitatively instead, thinking in terms of "increase" and "decrease" rather than definite values.
- For qualitative problems, try approaching it quantitatively instead, proposing simple numerical values for the variables.
- Were there any assumptions you made while solving this problem? Would your solution change if one of those assumptions were altered?
- Identify where it would be easy for someone to go astray in attempting to solve this problem.
- Formulate your own problem based on what you learned solving this one.

General follow-up challenges for experiments or projects

- In what way(s) was this experiment or project easy to complete?
- Identify some of the challenges you faced in completing this experiment or project.

- Show how thorough documentation assisted in the completion of this experiment or project.
- Which fundamental laws or principles are key to this system's function?
- Identify any way(s) in which one might obtain false or otherwise misleading measurements from test equipment in this system.
- What will happen if (component X) fails (open/shorted/etc.)?
- What would have to occur to make this system unsafe?

6.1 Conceptual reasoning

These questions are designed to stimulate your analytic and synthetic thinking[3]. In a Socratic discussion with your instructor, the goal is for these questions to prompt an extended dialogue where assumptions are revealed, conclusions are tested, and understanding is sharpened. Your instructor may also pose additional questions based on those assigned, in order to further probe and refine your conceptual understanding.

Questions that follow are presented to challenge and probe your understanding of various concepts presented in the tutorial. These questions are intended to serve as a guide for the Socratic dialogue between yourself and the instructor. Your instructor's task is to ensure you have a sound grasp of these concepts, and the questions contained in this document are merely a means to this end. Your instructor may, at his or her discretion, alter or substitute questions for the benefit of tailoring the discussion to each student's needs. The only absolute requirement is that each student is challenged and assessed at a level equal to or greater than that represented by the documented questions.

It is far more important that you convey your *reasoning* than it is to simply convey a correct answer. For this reason, you should refrain from researching other information sources to answer questions. What matters here is that *you* are doing the thinking. If the answer is incorrect, your instructor will work with you to correct it through proper reasoning. A correct answer without an adequate explanation of how you derived that answer is unacceptable, as it does not aid the learning or assessment process.

You will note a conspicuous lack of answers given for these conceptual questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your conceptual answers, where applicable, is to use circuit simulation software to explore the effects of changes made to circuits. For example, if one of these conceptual questions challenges you to predict the effects of altering some component parameter in a circuit, you may check the validity of your work by simulating that same parameter change within software and seeing if the results agree.

[3] *Analytical* thinking involves the "disassembly" of an idea into its constituent parts, analogous to dissection. *Synthetic* thinking involves the "assembly" of a new idea comprised of multiple concepts, analogous to construction. Both activities are high-level cognitive skills, extremely important for effective problem-solving, necessitating frequent challenge and regular practice to fully develop.

6.1.1 Reading outline and reflections

"Reading maketh a full man; conference a ready man; and writing an exact man" – Francis Bacon

Francis Bacon's advice is a blueprint for effective education: reading provides the learner with knowledge, writing focuses the learner's thoughts, and critical dialogue equips the learner to confidently communicate and apply their learning. Independent acquisition and application of knowledge is a powerful skill, well worth the effort to cultivate. To this end, students should read these educational resources closely, journal their own reflections on the reading, and discuss in detail their findings with classmates and instructor(s). You should be able to do all of the following after reading any instructional text:

☑ Briefly SUMMARIZE THE TEXT in the form of a journal entry documenting your learning as you progress through the course of study. Share this summary in dialogue with your classmates and instructor. Journaling is an excellent self-test of thorough reading because you cannot clearly express what you have not read or did not comprehend.

☑ Demonstrate ACTIVE READING STRATEGIES, including verbalizing your impressions as you read, simplifying long passages to convey the same ideas using fewer words, annotating text and illustrations with your own interpretations, working through mathematical examples shown in the text, cross-referencing passages with relevant illustrations and/or other passages, identifying problem-solving strategies applied by the author, etc. Technical reading is a special case of problem-solving, and so these strategies work precisely because they help solve any problem: paying attention to your own thoughts (metacognition), eliminating unnecessary complexities, identifying what makes sense, paying close attention to details, drawing connections between separated facts, and noting the successful strategies of others.

☑ Identify IMPORTANT THEMES, especially GENERAL LAWS and PRINCIPLES, expounded in the text and express them in the simplest of terms as though you were teaching an intelligent child. This emphasizes connections between related topics and develops your ability to communicate complex ideas to anyone.

☑ Form YOUR OWN QUESTIONS based on the reading, and then pose them to your instructor and classmates for their consideration. Anticipate both correct and incorrect answers, the incorrect answer(s) assuming one or more plausible misconceptions. This helps you view the subject from different perspectives to grasp it more fully.

☑ Devise EXPERIMENTS to test claims presented in the reading, or to disprove misconceptions. Predict possible outcomes of these experiments, and evaluate their meanings: what result(s) would confirm, and what would constitute disproof? Running mental simulations and evaluating results is essential to scientific and diagnostic reasoning.

☑ Specifically identify any points you found CONFUSING. The reason for doing this is to help diagnose misconceptions and overcome barriers to learning.

6.1.2 Foundational concepts

Correct analysis and diagnosis of electric circuits begins with a proper understanding of some basic concepts. The following is a list of some important concepts referenced in this module's full tutorial. Define each of them in your own words, and be prepared to illustrate each of these concepts with a description of a practical example and/or a live demonstration.

Energy

Thermionic emission

Cathode versus Anode

Vacuum tube

Doping

N-type semiconductor

P-type semiconductor

Depletion region

Diode

Forward-biasing

Reverse-biasing

Shockley diode equation

Ohm's Law

Electrical source

Electrical load

Light-emitting diode

Photovoltaic cell

Breakdown

6.1.3 Motor-effect eliminator

Suppose we have an application where a DC generator provides power to charge a secondary-cell battery:

The only problem with this setup is, the generator tries to act as a motor when the engine turning it is shut off, drawing power from the battery and discharging it. How could we use a diode to prevent this from happening?

Challenges

- Is there any other circuit arrangement with the diode that might work as well?
- Identify any relevant diode ratings we should be aware of when selecting a diode for this application.

6.1.4 Temperature sensor

For any given amount of forward current, a diode's voltage drop is a function of junction temperature. Sketch a circuit exploiting this principle, and describe how you could use it to sense ambient temperature.

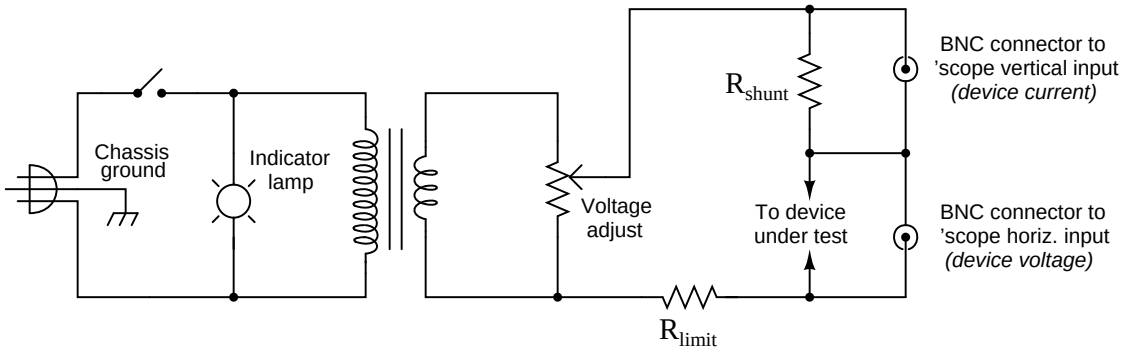
Challenges

- Do you anticipate any sources of error with your design?

6.1.5 Curve tracer circuit

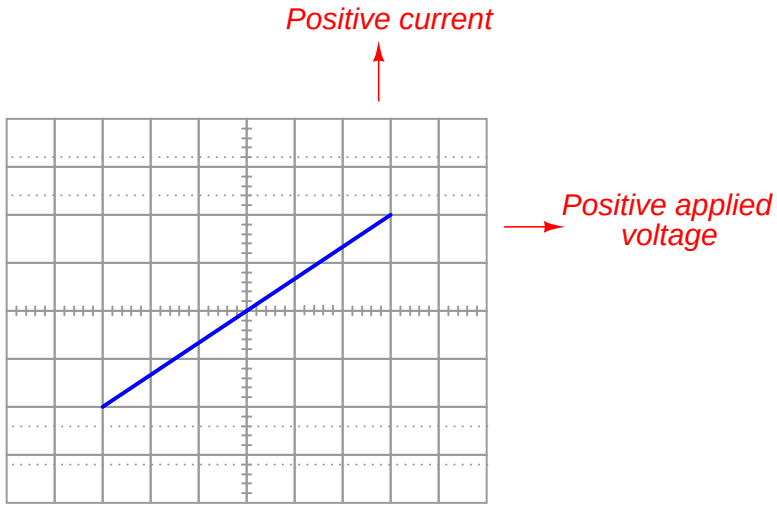
The following schematic diagram is of a simple *curve tracer circuit*, used to plot the current/voltage characteristics of different electronic components on an oscilloscope screen:

Simple curve tracer circuit



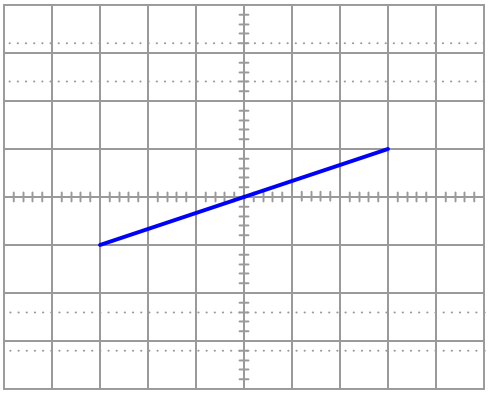
The way it works is by applying an AC voltage across the terminals of the device under test, outputting two different voltage signals to the oscilloscope. One signal, driving the horizontal axis of the oscilloscope, represents the voltage across the two terminals of the device. The other signal, driving the vertical axis of the oscilloscope, is the voltage dropped across the shunt resistor, representing current through the device. With the oscilloscope set for "X-Y" mode, the electron beam traces the device's characteristic curve.

For example, a simple resistor would generate this oscilloscope display:



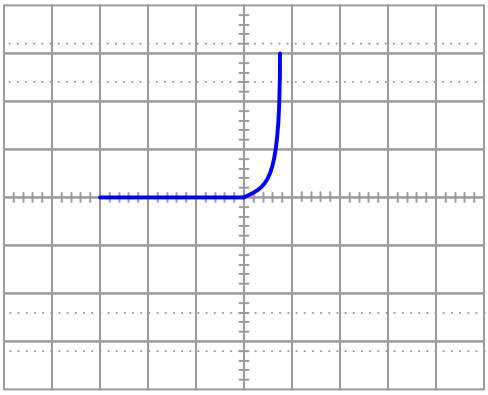
A resistor of greater value (more Ohms of resistance) would generate a characteristic plot with a shallower slope, representing less current for the same amount of applied voltage:

Higher-valued resistor



Curve tracer circuits find their real value in testing semiconductor components, whose voltage/current behaviors are nonlinear. Take for instance this characteristic curve for an ordinary rectifying diode:

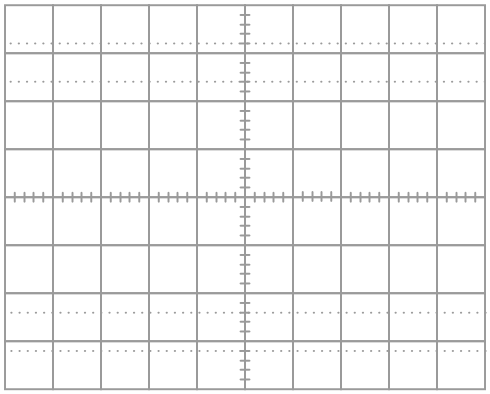
Rectifying diode curve



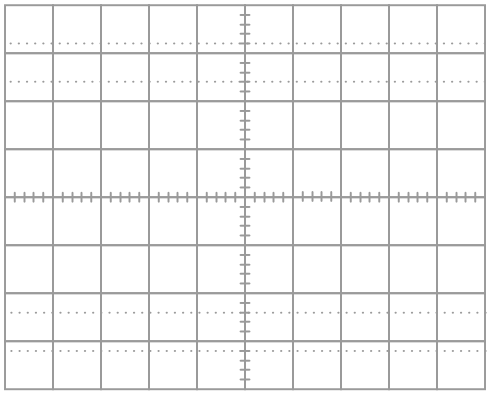
The trace is flat everywhere left of center where the applied voltage is negative, indicating no diode current when it is reverse-biased. To the right of center, though, the trace bends sharply upward, indicating exponential diode current with increasing applied voltage (forward-biased) just as Shockley's equation predicts.

On the following grids, plot the characteristic curve for a diode that is failed shorted, and also for one that is failed open:

Diode failed shorted



Diode failed open

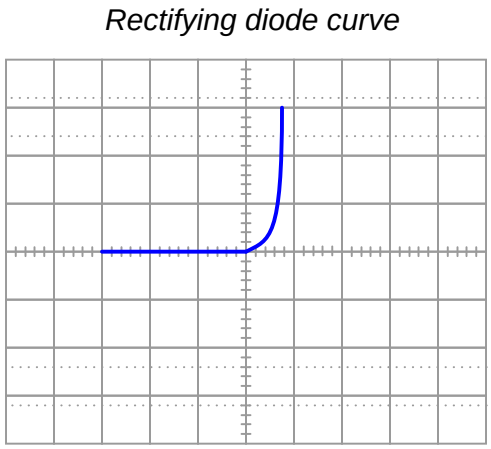


Challenges

- Does the amplitude (i.e. voltage) of the AC source matter to the operation of this circuit?
- Does the frequency of the AC source matter to the operation of this circuit?

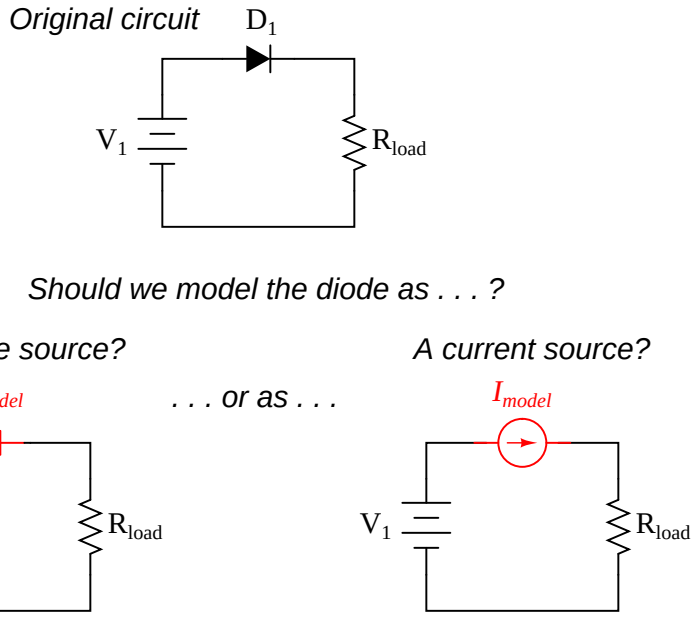
6.1.6 Diode modeled as a source

When plotted on a curve tracer, the characteristic curve for a normal PN junction rectifying diode looks something like this:



Label each axis (horizontal and vertical) of the curve tracer graph, then determine whether the diode behaves more like a voltage source or more like a current source (i.e. does it try to maintain constant voltage or does it try to maintain constant current?) when it is conducting current.

Models are very useful because they simplify circuit approximations. For example, we can analyze this diode circuit quite easily if we substitute an electrical source in place of the diode:



Challenges

- In what way(s) is a diode not a true electrical source?
- How might this information be useful when applying Thévenin's or Norton's theorem to a circuit?

6.1.7 Diode frequency limit

What diode performance parameter(s) establish the limit for maximum frequency of AC which it may rectify? If you were to examine a diode datasheet, what parameter (or parameters) would be the most important in answering this question?

6.2 Quantitative reasoning

These questions are designed to stimulate your computational thinking. In a Socratic discussion with your instructor, the goal is for these questions to reveal your mathematical approach(es) to problem-solving so that good technique and sound reasoning may be reinforced. Your instructor may also pose additional questions based on those assigned, in order to observe your problem-solving firsthand.

Mental arithmetic and estimations are strongly encouraged for all calculations, because without these abilities you will be unable to readily detect errors caused by calculator misuse (e.g. keystroke errors).

You will note a conspicuous lack of answers given for these quantitative questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. My advice is to use circuit simulation software such as SPICE to check the correctness of quantitative answers. Refer to those learning modules within this collection focusing on SPICE to see worked examples which you may use directly as practice problems for your own study, and/or as templates you may modify to run your own analyses and generate your own practice problems.

Completely worked example problems found in the Tutorial may also serve as "test cases[4]" for gaining proficiency in the use of circuit simulation software, and then once that proficiency is gained you will never need to rely[5] on an answer key!

[4] In other words, set up the circuit simulation software to analyze the same circuit examples found in the Tutorial. If the simulated results match the answers shown in the Tutorial, it confirms the simulation has properly run. If the simulated results disagree with the Tutorial's answers, something has been set up incorrectly in the simulation software. Using every Tutorial as practice in this way will quickly develop proficiency in the use of circuit simulation software.

[5] This approach is perfectly in keeping with the instructional philosophy of these learning modules: *teaching students to be self-sufficient thinkers*. Answer keys can be useful, but it is even more useful to your long-term success to have a set of tools on hand for checking your own work, because once you have left school and are on your own, there will no longer be "answer keys" available for the problems you will have to solve.

6.2.1 Miscellaneous physical constants

Note: constants shown in **bold** type are *exact*, not approximations. Values inside of parentheses show one standard deviation (σ) of uncertainty in the final digits: for example, the magnetic permeability of free space value given as $1.25663706212(19) \times 10^{-6}$ H/m represents a center value (i.e. the location parameter) of $1.25663706212 \times 10^{-6}$ Henrys per meter with one standard deviation of uncertainty equal to $0.0000000000019 \times 10^{-6}$ Henrys per meter.

Avogadro's number (N_A) = **6.02214076** $\times 10^{23}$ **per mole** (mol^{-1})

Boltzmann's constant (k) = **1.380649** $\times 10^{-23}$ **Joules per Kelvin** (J/K)

Electronic charge (e) = **1.602176634** $\times 10^{-19}$ **Coulomb** (C)

Faraday constant (F) = **96,485.33212...** $\times 10^{4}$ **Coulombs per mole** (C/mol)

Magnetic permeability of free space (μ_0) = $1.25663706212(19) \times 10^{-6}$ Henrys per meter (H/m)

Electric permittivity of free space (ϵ_0) = $8.8541878128(13) \times 10^{-12}$ Farads per meter (F/m)

Characteristic impedance of free space (Z_0) = 376.730313668(57) Ohms (Ω)

Gravitational constant (G) = $6.67430(15) \times 10^{-11}$ cubic meters per kilogram-seconds squared ($m^3/kg\text{-}s^2$)

Molar gas constant (R) = **8.314462618... Joules per mole-Kelvin** (J/mol-K) = 0.08205746(14) liters-atmospheres per mole-Kelvin

Planck constant (h) = **6.62607015** $\times 10^{-34}$ **joule-seconds** (J-s)

Stefan-Boltzmann constant (σ) = **5.670374419...** $\times 10^{-8}$ **Watts per square meter-Kelvin**^{4} ($W/m^2 \cdot K^4$)

Speed of light in a vacuum (c) = **299,792,458 meters per second** (m/s) = 186282.4 miles per second (mi/s)

Note: All constants taken from NIST data "Fundamental Physical Constants – Complete Listing", from `http://physics.nist.gov/constants`, National Institute of Standards and Technology (NIST), 2018 CODATA Adjustment.

6.2.2 Introduction to spreadsheets

A powerful computational tool you are encouraged to use in your work is a *spreadsheet*. Available on most personal computers (e.g. Microsoft Excel), *spreadsheet* software performs numerical calculations based on number values and formulae entered into cells of a grid. This grid is typically arranged as lettered columns and numbered rows, with each cell of the grid identified by its column/row coordinates (e.g. cell `B3`, cell `A8`). Each cell may contain a string of text, a number value, or a mathematical formula. The spreadsheet automatically updates the results of all mathematical formulae whenever the entered number values are changed. This means it is possible to set up a spreadsheet to perform a series of calculations on entered data, and those calculations will be re-done by the computer any time the data points are edited in any way.

For example, the following spreadsheet calculates average speed based on entered values of distance traveled and time elapsed:

	A	B	C	D
1	Distance traveled	46.9	Kilometers	
2	Time elapsed	1.18	Hours	
3	Average speed	= B1 / B2	km/h	
4				
5				

Text labels contained in cells `A1` through `A3` and cells `C1` through `C3` exist solely for readability and are not involved in any calculations. Cell `B1` contains a sample distance value while cell `B2` contains a sample time value. The formula for computing speed is contained in cell `B3`. Note how this formula begins with an "equals" symbol (`=`), references the values for distance and speed by lettered column and numbered row coordinates (`B1` and `B2`), and uses a forward slash symbol for division (`/`). The coordinates `B1` and `B2` function as *variables*[6] would in an algebraic formula.

When this spreadsheet is executed, the numerical value `39.74576` will appear in cell `B3` rather than the formula `= B1 / B2`, because 39.74576 is the computed speed value given 46.9 kilometers traveled over a period of 1.18 hours. If a different numerical value for distance is entered into cell `B1` or a different value for time is entered into cell `B2`, cell `B3`'s value will automatically update. All you need to do is set up the given values and any formulae into the spreadsheet, and the computer will do all the calculations for you.

Cell `B3` may be referenced by other formulae in the spreadsheet if desired, since it is a variable just like the given values contained in `B1` and `B2`. This means it is possible to set up an entire chain of calculations, one dependent on the result of another, in order to arrive at a final value. The arrangement of the given data and formulae need not follow any pattern on the grid, which means you may place them anywhere.

[6] Spreadsheets may also provide means to attach text labels to cells for use as variable names (Microsoft Excel simply calls these labels "names"), but for simple spreadsheets such as those shown here it's usually easier just to use the standard coordinate naming for each cell.

Common[7] arithmetic operations available for your use in a spreadsheet include the following:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)
- Powers (`^`)
- Square roots (`sqrt()`)
- Logarithms (`ln()` , `log10()`)

Parentheses may be used to ensure[8] proper order of operations within a complex formula. Consider this example of a spreadsheet implementing the *quadratic formula*, used to solve for roots of a polynomial expression in the form of $ax^2 + bx + c$:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

	A	B
1	x_1	= (-B4 + sqrt((B4^2) - (4*B3*B5))) / (2*B3)
2	x_2	= (-B4 - sqrt((B4^2) - (4*B3*B5))) / (2*B3)
3	a =	9
4	b =	5
5	c =	-2

This example is configured to compute roots[9] of the polynomial $9x^2 + 5x - 2$ because the values of 9, 5, and -2 have been inserted into cells `B3`, `B4`, and `B5`, respectively. Once this spreadsheet has been built, though, it may be used to calculate the roots of *any* second-degree polynomial expression simply by entering the new a, b, and c coefficients into cells `B3` through `B5`. The numerical values appearing in cells `B1` and `B2` will be automatically updated by the computer immediately following any changes made to the coefficients.

[7] Modern spreadsheet software offers a bewildering array of mathematical functions you may use in your computations. I recommend you consult the documentation for your particular spreadsheet for information on operations other than those listed here.

[8] Spreadsheet programs, like text-based programming languages, are designed to follow standard order of operations by default. However, my personal preference is to use parentheses even where strictly unnecessary just to make it clear to any other person viewing the formula what the intended order of operations is.

[9] Reviewing some algebra here, a *root* is a value for x that yields an overall value of zero for the polynomial. For this polynomial ($9x^2 + 5x - 2$) the two roots happen to be $x = 0.269381$ and $x = -0.82494$, with these values displayed in cells `B1` and `B2`, respectively upon execution of the spreadsheet.

Alternatively, one could break up the long quadratic formula into smaller pieces like this:

$$y = \sqrt{b^2 - 4ac} \qquad z = 2a$$

$$x = \frac{-b \pm y}{z}$$

	A	B	C
1	x_1	= (-B4 + C1) / C2	= sqrt((B4^2) - (4*B3*B5))
2	x_2	= (-B4 - C1) / C2	= 2*B3
3	a =	9	
4	b =	5	
5	c =	-2	

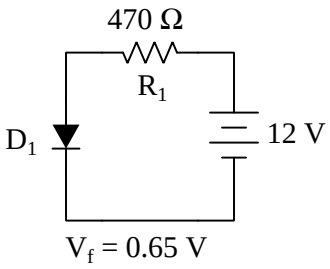
Note how the square-root term (y) is calculated in cell C1, and the denominator term (z) in cell C2. This makes the two final formulae (in cells B1 and B2) simpler to interpret. The positioning of all these cells on the grid is completely arbitrary[10] – all that matters is that they properly reference each other in the formulae.

Spreadsheets are particularly useful for situations where the same set of calculations representing a circuit or other system must be repeated for different initial conditions. The power of a spreadsheet is that it automates what would otherwise be a tedious set of calculations. One specific application of this is to simulate the effects of various components within a circuit failing with abnormal values (e.g. a shorted resistor simulated by making its value nearly zero; an open resistor simulated by making its value extremely large). Another application is analyzing the behavior of a circuit design given new components that are out of specification, and/or aging components experiencing drift over time.

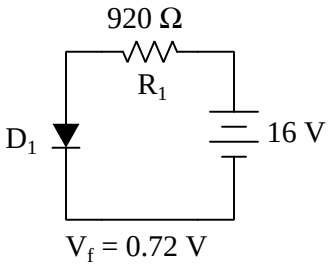
[10] My personal preference is to locate all the "given" data in the upper-left cells of the spreadsheet grid (each data point flanked by a sensible name in the cell to the left and units of measurement in the cell to the right as illustrated in the first distance/time spreadsheet example), sometimes coloring them in order to clearly distinguish which cells contain entered data versus which cells contain computed results from formulae. I like to place all formulae in cells below the given data, and try to arrange them in logical order so that anyone examining my spreadsheet will be able to figure out *how* I constructed a solution. This is a general principle I believe all computer programmers should follow: *document and arrange your code to make it easy for other people to learn from it.*

6.2.3 Voltages and currents in simple diode circuits

Complete the following table of values for this diode circuit, assuming the forward voltage drop values shown:



	R_1	D_1	Total
V			12 V
I			
R	470 Ω		
P			



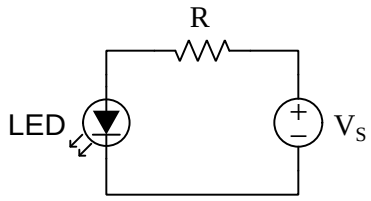
	R_1	D_1	Total
V			16 V
I			
R	920 Ω		
P			

Challenges

- How will voltages and currents be affected by the diode failing open?
- How will voltages and currents be affected by the diode failing shorted?
- How will voltages and currents be affected by the resistor failing open?
- How will voltages and currents be affected by the resistor failing shorted?
- How will voltages and currents be affected by the source polarity being reversed?

6.2.4 LED resistor sizing

Calculate the proper resistor value to limit LED voltage and current to 1.65 Volts and 22 milliAmperes, respectively, given the following source voltage ratings:



- V_S = 15 Volts ; R =
- V_S = 24 Volts ; R =
- V_S = 48 Volts ; R =

Challenges

- A very common mistake made by beginning students is to calculate resistor values of 681.8 Ω, 1090.9 Ω, and 2181.8 Ω, respectively. Identify the error implicit in these results.
- Calculate the amount of power dissipated by the resistor for each of these source voltages.

6.3 Diagnostic reasoning

These questions are designed to stimulate your deductive and inductive thinking, where you must apply general principles to specific scenarios (deductive) and also derive conclusions about the failed circuit from specific details (inductive). In a Socratic discussion with your instructor, the goal is for these questions to reinforce your recall and use of general circuit principles and also challenge your ability to integrate multiple symptoms into a sensible explanation of what's wrong in a circuit. Your instructor may also pose additional questions based on those assigned, in order to further challenge and sharpen your diagnostic abilities.

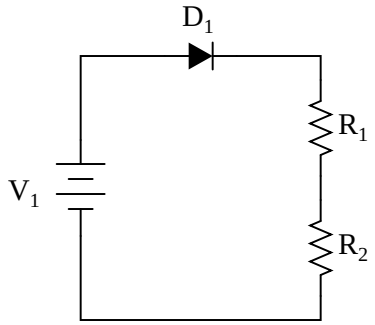
As always, your goal is to fully *explain* your analysis of each problem. Simply obtaining a correct answer is not good enough – you must also demonstrate sound reasoning in order to successfully complete the assignment. Your instructor's responsibility is to probe and challenge your understanding of the relevant principles and analytical processes in order to ensure you have a strong foundation upon which to build further understanding.

You will note a conspicuous lack of answers given for these diagnostic questions. Unlike standard textbooks where answers to every other question are given somewhere toward the back of the book, here in these learning modules students must rely on other means to check their work. The best way by far is to debate the answers with fellow students and also with the instructor during the Socratic dialogue sessions intended to be used with these learning modules. Reasoning through challenging questions with other people is an excellent tool for developing strong reasoning skills.

Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the validity of your work by simulating that same fault (substituting a very high resistance in place of that component for an open, and substituting a very low resistance for a short) within software and seeing if the results agree.

6.3.1 Faults in a diode-resistor network

Predict how all component voltages and currents in this circuit will be affected as a result of the following faults. Consider each fault independently (i.e. one at a time, no coincidental faults):



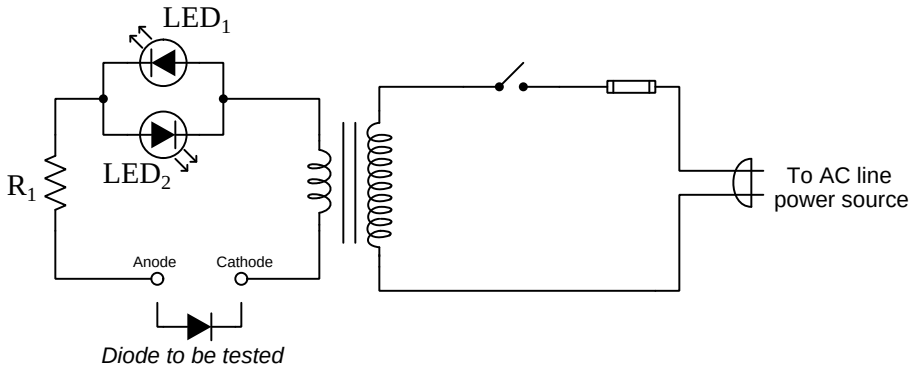
- Diode D_1 fails open:
- Diode D_1 fails shorted:
- Resistor R_1 fails open:
- Resistor R_1 fails shorted:
- Resistor R_2 fails open:
- Resistor R_2 fails shorted:

Challenges

- What will happen if the voltage source polarity becomes reversed?

6.3.2 Diode-testing circuit

Explain how the following diode-testing circuit is supposed to function for the following scenarios:



- Good diode
- Shorted diode
- Open diode
- Backwards diode

Challenges

- Explain how to properly size the resistor in this circuit.

Appendix A

Problem-Solving Strategies

The ability to solve complex problems is arguably one of the most valuable skills one can possess, and this skill is particularly important in any science-based discipline.

- Study principles, not procedures. Don't be satisfied with merely knowing how to compute solutions – learn *why* those solutions work.
- Identify what it is you need to solve, identify all relevant data, identify all units of measurement, identify any general principles or formulae linking the given information to the solution, and then identify any "missing pieces" to a solution. Annotate all diagrams with this data.
- Sketch a diagram to help visualize the problem. When building a real system, always devise a plan for that system and analyze its function *before* constructing it.
- Follow the units of measurement and meaning of every calculation. If you are ever performing mathematical calculations as part of a problem-solving procedure, and you find yourself unable to apply each and every intermediate result to some aspect of the problem, it means you don't understand what you are doing. Properly done, every mathematical result should have practical meaning for the problem, and not just be an abstract number. You should be able to identify the proper units of measurement for each and every calculated result, and show where that result fits into the problem.
- Perform "thought experiments" to explore the effects of different conditions for theoretical problems. When troubleshooting real systems, perform *diagnostic tests* rather than visually inspecting for faults, the best diagnostic test being the one giving you the most information about the nature and/or location of the fault with the fewest steps.
- Simplify the problem until the solution becomes obvious, and then use that obvious case as a model to follow in solving the more complex version of the problem.
- Check for exceptions to see if your solution is incorrect or incomplete. A good solution will work for *all* known conditions and criteria. A good example of this is the process of testing scientific hypotheses: the task of a scientist is not to find support for a new idea, but rather to *challenge* that new idea to see if it holds up under a battery of tests. The philosophical

principle of *reductio ad absurdum* (i.e. disproving a general idea by finding a specific case where it fails) is useful here.

- Work "backward" from a hypothetical solution to a new set of given conditions.
- Add quantities to problems that are qualitative in nature, because sometimes a little math helps illuminate the scenario.
- Sketch graphs illustrating how variables relate to each other. These may be quantitative (i.e. with realistic number values) or qualitative (i.e. simply showing increases and decreases).
- Treat quantitative problems as qualitative in order to discern the relative magnitudes and/or directions of change of the relevant variables. For example, try determining what happens if a certain variable were to increase or decrease before attempting to precisely calculate quantities: how will each of the dependent variables respond, by increasing, decreasing, or remaining the same as before?
- Consider limiting cases. This works especially well for qualitative problems where you need to determine which direction a variable will change. Take the given condition and magnify that condition to an extreme degree as a way of simplifying the direction of the system's response.
- Check your work. This means regularly testing your conclusions to see if they make sense. This does *not* mean repeating the same steps originally used to obtain the conclusion(s), but rather to use some other means to check validity. Simply repeating procedures often leads to *repeating the same errors* if any were made, which is why alternative paths are better.

Appendix B

Instructional philosophy

"The unexamined circuit is not worth energizing" – Socrates (if he had taught electricity)

These learning modules, although useful for self-study, were designed to be used in a formal learning environment where a subject-matter expert challenges students to digest the content and exercise their critical thinking abilities in the answering of questions and in the construction and testing of working circuits.

The following principles inform the instructional and assessment philosophies embodied in these learning modules:

- The first goal of education is to enhance clear and independent thought, in order that every student reach their fullest potential in a highly complex and inter-dependent world. Robust reasoning is *always* more important than particulars of any subject matter, because its application is universal.
- Literacy is fundamental to independent learning and thought because text continues to be the most efficient way to communicate complex ideas over space and time. Those who cannot read with ease are limited in their ability to acquire knowledge and perspective.
- Articulate communication is fundamental to work that is complex and interdisciplinary.
- Faulty assumptions and poor reasoning are best corrected through challenge, not presentation. The rhetorical technique of *reductio ad absurdum* (disproving an assertion by exposing an absurdity) works well to discipline student's minds, not only to correct the problem at hand but also to learn how to detect and correct future errors.
- Important principles should be repeatedly explored and widely applied throughout a course of study, not only to reinforce their importance and help ensure their mastery, but also to showcase the interconnectedness and utility of knowledge.

These learning modules were expressly designed to be used in an "inverted" teaching environment[1] where students first read the introductory and tutorial chapters on their own, then individually attempt to answer the questions and construct working circuits according to the experiment and project guidelines. The instructor never lectures, but instead meets regularly with each individual student to review their progress, answer questions, identify misconceptions, and challenge the student to new depths of understanding through further questioning. Regular meetings between instructor and student should resemble a Socratic[2] dialogue, where questions serve as scalpels to dissect topics and expose assumptions. The student passes each module only after consistently demonstrating their ability to logically analyze and correctly apply all major concepts in each question or project/experiment. The instructor must be vigilant in probing each student's understanding to ensure they are truly *reasoning* and not just *memorizing*. This is why "Challenge" points appear throughout, as prompts for students to think deeper about topics and as starting points for instructor queries. Sometimes these challenge points require additional knowledge that hasn't been covered in the series to answer in full. This is okay, as the major purpose of the Challenges is to stimulate analysis and synthesis on the part of each student.

The instructor must possess enough mastery of the subject matter and awareness of students' reasoning to generate their own follow-up questions to practically any student response. Even completely correct answers given by the student should be challenged by the instructor for the purpose of having students practice articulating their thoughts and defending their reasoning. Conceptual errors committed by the student should be exposed and corrected not by direct instruction, but rather by reducing the errors to an absurdity[3] through well-chosen questions and thought experiments posed by the instructor. Becoming proficient at this style of instruction requires time and dedication, but the positive effects on critical thinking for both student and instructor are spectacular.

An inspection of these learning modules reveals certain unique characteristics. One of these is a bias toward thorough explanations in the tutorial chapters. Without a live instructor to explain concepts and applications to students, the text itself must fulfill this role. This philosophy results in lengthier explanations than what you might typically find in a textbook, each step of the reasoning process fully explained, including footnotes addressing common questions and concerns students raise while learning these concepts. Each tutorial seeks to not only explain each major concept in sufficient detail, but also to explain the logic of each concept and how each may be developed

[1] In a traditional teaching environment, students first encounter new information via *lecture* from an expert, and then independently apply that information via *homework*. In an "inverted" course of study, students first encounter new information via *homework*, and then independently apply that information under the scrutiny of an expert. The expert's role in lecture is to simply *explain*, but the expert's role in an inverted session is to *challenge*, *critique*, and if necessary *explain* where gaps in understanding still exist.

[2] Socrates is a figure in ancient Greek philosophy famous for his unflinching style of questioning. Although he authored no texts, he appears as a character in Plato's many writings. The essence of Socratic philosophy is to leave no question unexamined and no point of view unchallenged. While purists may argue a topic such as electric circuits is too narrow for a true Socratic-style dialogue, I would argue that the essential thought processes involved with scientific reasoning on *any* topic are not far removed from the Socratic ideal, and that students of electricity and electronics would do very well to challenge assumptions, pose thought experiments, identify fallacies, and otherwise employ the arsenal of critical thinking skills modeled by Socrates.

[3] This rhetorical technique is known by the Latin phrase *reductio ad absurdum*. The concept is to expose errors by counter-example, since only one solid counter-example is necessary to disprove a universal claim. As an example of this, consider the common misconception among beginning students of electricity that voltage cannot exist without current. One way to apply *reductio ad absurdum* to this statement is to ask how much current passes through a fully-charged battery connected to nothing (i.e. a clear example of voltage existing without current).

from "first principles". Again, this reflects the goal of developing clear and independent thought in students' minds, by showing how clear and logical thought was used to forge each concept. Students benefit from witnessing a model of clear thinking in action, and these tutorials strive to be just that.

Another characteristic of these learning modules is a lack of step-by-step instructions in the Project and Experiment chapters. Unlike many modern workbooks and laboratory guides where step-by-step instructions are prescribed for each experiment, these modules take the approach that students must learn to closely read the tutorials and apply their own reasoning to identify the appropriate experimental steps. Sometimes these steps are plainly declared in the text, just not as a set of enumerated points. At other times certain steps are implied, an example being assumed competence in test equipment use where the student should not need to be told *again* how to use their multimeter because that was thoroughly explained in previous lessons. In some circumstances no steps are given at all, leaving the entire procedure up to the student.

This lack of prescription is not a flaw, but rather a feature. Close reading and clear thinking are foundational principles of this learning series, and in keeping with this philosophy all activities are designed to *require* those behaviors. Some students may find the lack of prescription frustrating, because it demands more from them than what their previous educational experiences required. This frustration should be interpreted as an unfamiliarity with autonomous thinking, a problem which must be corrected if the student is ever to become a self-directed learner and effective problem-solver. Ultimately, the need for students to read closely and think clearly is more important both in the near-term and far-term than any specific facet of the subject matter at hand. If a student takes longer than expected to complete a module because they are forced to outline, digest, and reason on their own, so be it. The future gains enjoyed by developing this mental discipline will be well worth the additional effort and delay.

Another feature of these learning modules is that they do not treat topics in isolation. Rather, important concepts are introduced early in the series, and appear repeatedly as stepping-stones toward other concepts in subsequent modules. This helps to avoid the "compartmentalization" of knowledge, demonstrating the inter-connectedness of concepts and simultaneously reinforcing them. Each module is fairly complete in itself, reserving the beginning of its tutorial to a review of foundational concepts.

This methodology of assigning text-based modules to students for digestion and then using Socratic dialogue to assess progress and hone students' thinking was developed over a period of several years by the author with his Electronics and Instrumentation students at the two-year college level. While decidedly unconventional and sometimes even unsettling for students accustomed to a more passive lecture environment, this instructional philosophy has proven its ability to convey conceptual mastery, foster careful analysis, and enhance employability so much better than lecture that the author refuses to ever teach by lecture again.

Problems which often go undiagnosed in a lecture environment are laid bare in this "inverted" format where students must articulate and logically defend their reasoning. This, too, may be unsettling for students accustomed to lecture sessions where the instructor cannot tell for sure who comprehends and who does not, and this vulnerability necessitates sensitivity on the part of the "inverted" session instructor in order that students never feel discouraged by having their errors exposed. *Everyone* makes mistakes from time to time, and learning is a lifelong process! Part of the instructor's job is to build a culture of learning among the students where errors are not seen as shameful, but rather as opportunities for progress.

To this end, instructors managing courses based on these modules should adhere to the following principles:

- Student questions are always welcome and demand thorough, honest answers. The only type of question an instructor should refuse to answer is one the student should be able to easily answer on their own. Remember, *the fundamental goal of education is for each student to learn to think clearly and independently.* This requires hard work on the part of the student, which no instructor should ever circumvent. Anything done to bypass the student's responsibility to do that hard work ultimately limits that student's potential and thereby does real harm.

- It is not only permissible, but encouraged, to answer a student's question by asking questions in return, these follow-up questions designed to guide the student to reach a correct answer through their own reasoning.

- All student answers demand to be challenged by the instructor and/or by other students. This includes both correct and incorrect answers – the goal is to practice the articulation and defense of one's own reasoning.

- No reading assignment is deemed complete unless and until the student demonstrates their ability to accurately summarize the major points in their own terms. Recitation of the original text is unacceptable. This is why every module contains an "Outline and reflections" question as well as a "Foundational concepts" question in the Conceptual reasoning section, to prompt reflective reading.

- No assigned question is deemed answered unless and until the student demonstrates their ability to consistently and correctly apply the concepts to *variations* of that question. This is why module questions typically contain multiple "Challenges" suggesting different applications of the concept(s) as well as variations on the same theme(s). Instructors are encouraged to devise as many of their own "Challenges" as they are able, in order to have a multitude of ways ready to probe students' understanding.

- No assigned experiment or project is deemed complete unless and until the student demonstrates the task in action. If this cannot be done "live" before the instructor, video-recordings showing the demonstration are acceptable. All relevant safety precautions must be followed, all test equipment must be used correctly, and the student must be able to properly explain all results. The student must also successfully answer all Challenges presented by the instructor for that experiment or project.

Students learning from these modules would do well to abide by the following principles:

- No text should be considered fully and adequately read unless and until you can express every idea *in your own words, using your own examples.*
- You should always articulate your thoughts as you read the text, noting points of agreement, confusion, and epiphanies. Feel free to print the text on paper and then write your notes in the margins. Alternatively, keep a journal for your own reflections as you read. This is truly a helpful tool when digesting complicated concepts.
- Never take the easy path of highlighting or underlining important text. Instead, *summarize* and/or *comment* on the text using your own words. This actively engages your mind, allowing you to more clearly perceive points of confusion or misunderstanding on your own.
- A very helpful strategy when learning new concepts is to place yourself in the role of a teacher, if only as a mental exercise. Either explain what you have recently learned to someone else, or at least *imagine* yourself explaining what you have learned to someone else. The simple act of having to articulate new knowledge and skill forces you to take on a different perspective, and will help reveal weaknesses in your understanding.
- Perform each and every mathematical calculation and thought experiment shown in the text on your own, referring back to the text to see that your results agree. This may seem trivial and unnecessary, but it is critically important to ensuring you actually understand what is presented, especially when the concepts at hand are complicated and easy to misunderstand. Apply this same strategy to become proficient in the use of *circuit simulation software*, checking to see if your simulated results agree with the results shown in the text.
- Above all, recognize that learning is hard work, and that a certain level of frustration is unavoidable. There are times when you will struggle to grasp some of these concepts, and that struggle is a natural thing. Take heart that it will yield with persistent and varied[4] effort, and never give up!

Students interested in using these modules for self-study will also find them beneficial, although the onus of responsibility for thoroughly reading and answering questions will of course lie with that individual alone. If a qualified instructor is not available to challenge students, a workable alternative is for students to form study groups where they challenge[5] one another.

To high standards of education,

Tony R. Kuphaldt

[4] As the old saying goes, "Insanity is trying the same thing over and over again, expecting different results." If you find yourself stumped by something in the text, you should attempt a different approach. Alter the thought experiment, change the mathematical parameters, do whatever you can to see the problem in a slightly different light, and then the solution will often present itself more readily.

[5] Avoid the temptation to simply share answers with study partners, as this is really counter-productive to learning. Always bear in mind that the answer to any question is far less important in the long run than the method(s) used to obtain that answer. The goal of education is to empower one's life through the improvement of clear and independent thought, literacy, expression, and various practical skills.

Appendix C

Tools used

I am indebted to the developers of many open-source software applications in the creation of these learning modules. The following is a list of these applications with some commentary on each.

You will notice a theme common to many of these applications: a bias toward *code*. Although I am by no means an expert programmer in any computer language, I understand and appreciate the flexibility offered by code-based applications where the user (you) enters commands into a plain ASCII text file, which the software then reads and processes to create the final output. Code-based computer applications are by their very nature *extensible*, while WYSIWYG (What You See Is What You Get) applications are generally limited to whatever user interface the developer makes for you.

The `GNU/Linux` computer operating system

> There is so much to be said about Linus Torvalds' `Linux` and Richard Stallman's `GNU` project. First, to credit just these two individuals is to fail to do justice to the *mob* of passionate volunteers who contributed to make this amazing software a reality. I first learned of `Linux` back in 1996, and have been using this operating system on my personal computers almost exclusively since then. It is *free*, it is completely *configurable*, and it permits the continued use of highly efficient `Unix` applications and scripting languages (e.g. shell scripts, Makefiles, `sed`, `awk`) developed over many decades. `Linux` not only provided me with a powerful computing platform, but its open design served to inspire my life's work of creating open-source educational resources.

Bram Moolenaar's `Vim` text editor

> Writing code for any code-based computer application requires a *text editor*, which may be thought of as a word processor strictly limited to outputting plain-ASCII text files. Many good text editors exist, and one's choice of text editor seems to be a deeply personal matter within the programming world. I prefer `Vim` because it operates very similarly to `vi` which is ubiquitous on `Unix`/`Linux` operating systems, and because it may be entirely operated via keyboard (i.e. no mouse required) which makes it fast to use.

Donald Knuth's TEX typesetting system

Developed in the late 1970's and early 1980's by computer scientist extraordinaire Donald Knuth to typeset his multi-volume magnum opus *The Art of Computer Programming*, this software allows the production of formatted text for screen-viewing or paper printing, all by writing plain-text code to describe how the formatted text is supposed to appear. TEX is not just a markup language for documents, but it is also a Turing-complete programming language in and of itself, allowing useful algorithms to be created to control the production of documents. Simply put, TEX *is a programmer's approach to word processing*. Since TEX is controlled by code written in a plain-text file, this means anyone may read that plain-text file to see exactly how the document was created. This openness afforded by the code-based nature of TEX makes it relatively easy to learn how other people have created their own TEX documents. By contrast, examining a beautiful document created in a conventional WYSIWYG word processor such as Microsoft `Word` suggests nothing to the reader about *how* that document was created, or what the user might do to create something similar. As Mr. Knuth himself once quipped, conventional word processing applications should be called WYSIAYG (What You See Is *All* You Get).

Leslie Lamport's LATEX extensions to TEX

Like all true programming languages, TEX is inherently extensible. So, years after the release of TEX to the public, Leslie Lamport decided to create a massive extension allowing easier compilation of book-length documents. The result was LATEX, which is the markup language used to create all ModEL module documents. You could say that TEX is to LATEX as `C` is to `C++`. This means it is permissible to use any and all TEX commands within LATEX source code, and it all still works. Some of the features offered by LATEX that would be challenging to implement in TEX include automatic index and table-of-content creation.

Tim Edwards' `Xcircuit` drafting program

This wonderful program is what I use to create all the schematic diagrams and illustrations (but not photographic images or mathematical plots) throughout the ModEL project. It natively outputs PostScript format which is a true vector graphic format (this is why the images do not pixellate when you zoom in for a closer view), and it is so simple to use that I have never had to read the manual! Object libraries are easy to create for `Xcircuit`, being plain-text files using PostScript programming conventions. Over the years I have collected a large set of object libraries useful for drawing electrical and electronic schematics, pictorial diagrams, and other technical illustrations.

`Gimp` graphic image manipulation program

Essentially an open-source clone of Adobe's `PhotoShop`, I use `Gimp` to resize, crop, and convert file formats for all of the photographic images appearing in the ModEL modules. Although `Gimp` does offer its own scripting language (called `Script-Fu`), I have never had occasion to use it. Thus, my utilization of `Gimp` to merely crop, resize, and convert graphic images is akin to using a sword to slice bread.

`SPICE` circuit simulation program

`SPICE` is to circuit analysis as TEX is to document creation: it is a form of markup language designed to describe a certain object to be processed in plain-ASCII text. When the plain-text "source file" is compiled by the software, it outputs the final result. More modern circuit analysis tools certainly exist, but I prefer `SPICE` for the following reasons: it is *free*, it is *fast*, it is *reliable*, and it is a fantastic tool for *teaching* students of electricity and electronics how to write simple code. I happen to use rather old versions of `SPICE`, version 2g6 being my "go to" application when I only require text-based output. `NGSPICE` (version 26), which is based on Berkeley `SPICE` version 3f5, is used when I require graphical output for such things as time-domain waveforms and Bode plots. In all `SPICE` example netlists I strive to use coding conventions compatible with all `SPICE` versions.

Andrew D. Hwang's `ePiX` mathematical visualization programming library

This amazing project is a `C++` library you may link to any `C/C++` code for the purpose of generating PostScript graphic images of mathematical functions. As a completely free and open-source project, it does all the plotting I would otherwise use a Computer Algebra System (CAS) such as `Mathematica` or `Maple` to do. It should be said that `ePiX` is *not* a Computer Algebra System like `Mathematica` or `Maple`, but merely a mathematical *visualization* tool. In other words, it won't determine integrals for you (you'll have to implement that in your own `C/C++` code!), but it can graph the results, and it does so beautifully. What I really admire about `ePiX` is that it is a `C++` programming library, which means it builds on the existing power and toolset available with that programming language. Mr. Hwang could have probably developed his own stand-alone application for mathematical plotting, but by creating a `C++` library to do the same thing he accomplished something much greater.

`gnuplot` mathematical visualization software

Another open-source tool for mathematical visualization is `gnuplot`. Interestingly, this tool is *not* part of Richard Stallman's GNU project, its name being a coincidence. For this reason the authors prefer "gnu" *not* be capitalized at all to avoid confusion. This is a much "lighter-weight" alternative to a spreadsheet for plotting tabular data, and the fact that it easily outputs directly to an X11 console or a file in a number of different graphical formats (including PostScript) is very helpful. I typically set my `gnuplot` output format to default (X11 on my Linux PC) for quick viewing while I'm developing a visualization, then switch to PostScript file export once the visual is ready to include in the document(s) I'm writing. As with my use of `Gimp` to do rudimentary image editing, my use of `gnuplot` only scratches the surface of its capabilities, but the important points are that it's *free* and that it *works well*.

`Python` programming language

Both Python and C++ find extensive use in these modules as instructional aids and exercises, but I'm listing Python here as a *tool* for myself because I use it almost daily as a *calculator*. If you open a Python interpreter console and type `from math import *` you can type mathematical expressions and have it return results just as you would on a hand calculator. Complex-number (i.e. *phasor*) arithmetic is similarly supported if you include the complex-math library (`from cmath import *`). Examples of this are shown in the Programming References chapter (if included) in each module. Of course, being a fully-featured programming language, Python also supports conditionals, loops, and other structures useful for calculation of quantities. Also, running in a console environment where all entries and returned values show as text in a chronologically-ordered list makes it easy to copy-and-paste those calculations to document exactly how they were performed.

Appendix D

Creative Commons License

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or

limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.

i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

A. reproduce and Share the Licensed Material, in whole or in part; and

B. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures.

For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:

A. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors,

whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

2. upon express reinstatement by the Licensor.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully

be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at `creativecommons.org/policies`, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at `creativecommons.org`.

Appendix E

References

"1N4001 thru 1N4007 General Purpose Plastic Rectifier", datasheet document number 88503, Vishay, 23 February 2011.

Pierce, George W., *US Patent 879,117*, "Rectifier and Detector", application 5 April 1907, patent granted 11 February 1908.

Stroustrup, Bjarne, *The C++ Programming Language*, fourth edition, Addison-Wesley, Upper Saddle River, NJ, 2013.

Appendix F

Version history

This is a list showing all significant additions, corrections, and other edits made to this learning module. Each entry is referenced by calendar date in reverse chronological order (newest version first), which appears on the front cover of every learning module for easy reference. Any contributors to this open-source document are listed here as well.

20 March 2025 – some edits to the Tutorial section on Schottky diodes, including the addition of a new illustration showing a Schottky diode formed of P-type semiconducting material.

14 March 2025 – added a photograph of an actual crystal-detector radio receiver to the Historical References section on crystal detectors.

30 October 2024 – added content to the Tutorial section on Schottky diodes, elaborating on their construction and characteristics.

11 October 2024 – divided the Introduction chapter into sections, one with recommendations for students, one with a listing of challenging concepts, and one with recommendations for instructors.

23 May 2024 – typographical error correction, courtesy of Ron Felix. Also added a reminder footnote on the meaning of conventional flow to the "PN junction diode behavior" section of the Tutorial at the request of Ron Felix.

21 March 2024 – minor edit to Schottky diode section clarifying the point about minimal stored charges.

9 March 2024 – added Tutorial sections on Zener diodes and Schottky diodes.

16 August 2023 – divided the Tutorial chapter into sections.

28 November 2022 – placed questions at the top of the itemized list in the Introduction chapter prompting students to devise experiments related to the tutorial content.

27 October 2022 – added some questions to the Introduction chapter and fixed some minor

typographical errors (one of them identified by Galen Bennett). Also made some minor additions to the Tutorial including edits to image_1888.

1 November 2021 – fixed cosmetic errors in image_4760.

9 May 2021 – commented out or deleted empty chapters.

20 April 2021 – minor additions to the "Faults in a diode-resistor network" Diagnostic Reasoning question.

16 April 2021 – minor additions to the Introduction chapter.

22 March 2021 – added more digits to Boltzmann's constant, and clarified that Kelvin is 273.15 more than Celsius.

18 March 2021 – corrected one instance of "volts" that should have been capitalized "Volts".

11 February 2021 – corrected several instances where "Ohms" was uncapitalized.

14 December 2020 – added commentary on multimeter testing of PN junctions to the Tutorial.

29 October 2020 – added some Challenge questions.

30 August 2020 – significantly edited the Introduction chapter to make it more suitable as a pre-study guide and to provide cues useful to instructors leading "inverted" teaching sessions. Also made some minor additions to the Tutorial chapter.

11 February 2020 – minor edits to the Tutorial.

24 January 2020 – added mention of "breakdown" to the Tutorial.

23 January 2020 – added Foundational Concepts to the list in the Conceptual Reasoning section.

5 January 2020 – added bullet-list of relevant programming principles to the Programming References section.

2 January 2020 – removed from from C++ code execution output, to clearly distinguish it from the source code listing which is still framed.

31 December 2019 – continued writing C++ code section modeling a PN junction.

30 December 2019 – began adding C++ code calculating current through a PN junction as a function of voltage.

18 December 2019 – minor edits to diagnostic questions, replacing "no multiple faults" with "no coincidental faults".

20 October 2019 – added patent number to the Historical Reference section on George Pierce's invention using crystals as rectifiers.

18 October 2019 – document first created.

Index