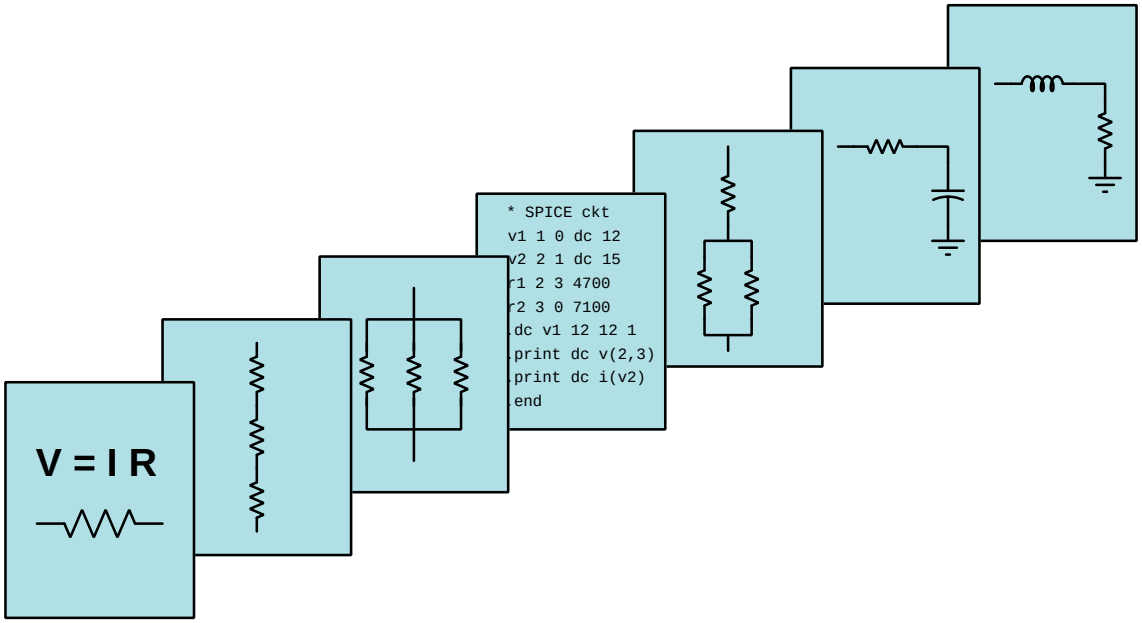
Modular Electronics Learning (ModEL) project



Encoders and Decoders

Last update = 18 November 2024

Contents

Chapter 1

Introduction

1.1 Recommendations for students

Encoder and decoder circuits translate between binary digital words and 1-of-n discrete digital states. For example, an 8-line to 3-line encoder circuit has eight input lines, one of which must be activated to generate a corresponding three-bit output word. A decoder does just the opposite: activate one of its n output lines in response to the binary word it receives at its input (e.g. a 3-line to 8-line decoder receives a three-bit binary word and activates the one output line out of eight corresponding to the binary word's value).

Important concepts related to encoders and decoders include **binary** numeration, **BCD** numeration, **powers of two**, **seven-segment displays**, display **driver** circuitry, **diode** behavior, **cathode** versus **anode** terminals, and **liquid crystal** displays.

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

- How might an experiment be designed and conducted to test the various functions of a 7-segment display decoder/driver IC? What hypotheses (i.e. predictions) might you pose for that experiment, and what result(s) would either support or disprove those hypotheses?
- What are some practical applications of encoder ICs?
- What are some practical applications of decoder ICs?
- Why is binary the preferred numeration system for digital electronic circuits?
- How does the concept of a *place-weight* apply to multiple numeration systems?
- How do the functions of encoders and decoders differ from each other?
- What distinguishes a priority encoder from a regular encoder?
- Which form of digital number representation is more efficient in terms of wiring, binary or one-of-n?

- How does BCD numeration differ from binary?
- What does it mean for a numerical code to be undefined?
- How are seven-segment displays used to represent decimal digits?
- Why are resistors necessary in conjunction with an LED seven-segment display?
- How do liquid-crystal displays operate?
- What is the "blanking" feature used for in a seven-segment decoder IC?
- What is the purpose of the "driver" circuitry within a seven-segment decoder IC?

1.2 Challenging concepts related to digital encoders and decoders

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

- **Confusing encoders, decoders, multiplexers, and demultiplexers** – students often mix up the identities of encoders, decoders, multiplexers, and demultiplexers. It is important for them to realize that while multiplexers and demultiplexers *contain* decoders, they are not the same as decoders (or encoders). In essence, muxes and demuxes "steer" signals to/from different locations, while encoders and decoders convert data from one format to another.
- **Gray versus Binary codes** – binary, like all place-weighted numeration systems, assigns different numerical "weights" to each character's place in the number. Gray code, by contrast, is *not* place-weighted although it is derived from binary by a series of Exclusive-OR (XOR) functions.

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** – Apply Boolean algebra to the design of decoder circuits

  Assessment – Devise Boolean expressions useful for designing combinational logic for a decoder circuit; e.g. pose problems in the form of the "SOP expressions for decoder circuit" Quantitative Reasoning question.

- **Outcome** – Design expanded digital circuits

  Assessment – Sketch a schematic diagram of a circuit employing a 4-line to 16-line decoders to form a 5-line to 32-line decoder; e.g. pose problems in the form of the "5-line to 32-line decoder" Quantitative Reasoning question.

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

  Assessment – Determine the probability of various component faults in a diode-based encoder circuit given symptoms and measured values; e.g. pose problems in the form of the "Faulted encoder circuit" Diagnostic Reasoning question.

- **Outcome** – Independent research

  Assessment – Locate IC encoder datasheets and properly interpret some of the information contained in those documents including truth tables, voltage levels, etc.

  Assessment – Locate IC decoder datasheets and properly interpret some of the information contained in those documents including truth tables, voltage levels, etc.

Chapter 2

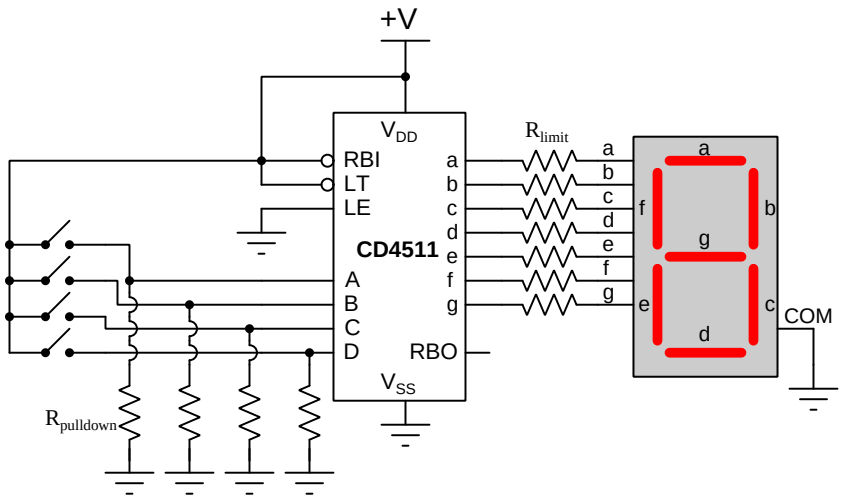
Case Tutorial

The idea behind a *Case Tutorial* is to explore new concepts by way of example. In this chapter you will read less presentation of theory compared to other Tutorial chapters, but by close observation and comparison of the given examples be able to discern patterns and principles much the same way as a scientific experimenter. Hopefully you will find these cases illuminating, and a good supplement to text-based tutorials.

These examples also serve well as challenges following your reading of the other Tutorial(s) in this module – can you explain *why* the circuits behave as they do?

2.1 Example: demonstration circuit using the CD4511

The model CD4511 is a CMOS integrated circuit providing a BCD-to-7-segment decoding function, along with several useful features. In this next diagram we see a simple proof-of-concept circuit showing how to make the CD4511 drive the seven LED segments of a display to show decimal digits:



Toggling the four switches between their different states causes the 7-segment display to show the following patterns:

D C B A	Display
0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9

Note the use of resistors in this demonstration circuit: four "pulldown" resistors necessary for providing an unambiguous "low" logic state when the respective toggle switch is open, and seven "current-limiting" resistors acting to limit each LED's current to a safe maximum value when the respective output line goes "high".

The Ripple Blanking Input (RBI) pin acts to blank the 7-segment display whenever it is placed in its active state. Being an active-low input, we must make the RBI input "high" in order to allow the display to show us anything at all.

The Lamp Test (LT) input is another active-low input terminal, designed to energize all seven of the LED segments regardless of the ABCD input states. We tie this "high" as well in order to allow the display to function normally.

The Latch Enable (LE) input (also known as the Strobe input) causes the decoder to latch, holding its last ABCD input states when this terminal goes to the "high" state. In order to allow the decoder to read new input states on its ABCD terminals, we must leave this input tied "low".

The Ripple Blanking Output (RBO) terminal is unused in this demonstrate circuit.

Chapter 3

Tutorial

3.1 Binary and BCD numeration

Modern digital electronic circuits represent numerical values by means of discrete (i.e. high or low) voltage states by treating those individual states as characters in a multi-digit pattern. Instead of calling the individual characters *digits* as we do for our common decimal numeration system, we call them *bits* and use the ciphers `1` and `0` to represent those high and low voltage states in a *binary* number. Each of these characters occupies a "place" in the number, each successive place-weight differing from the preceding by a factor of two.

A four-bit binary counting pattern from zero to fifteen represents sixteen different combinations ($2^4 = 16$), and is shown here in conjunction with their decimal and hexadecimal equivalent values:

Binary	Decimal	Hex
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9
1 0 1 0	1 0	A
1 0 1 1	1 1	B
1 1 0 0	1 2	C
1 1 0 1	1 3	D
1 1 1 0	1 4	E
1 1 1 1	1 5	F

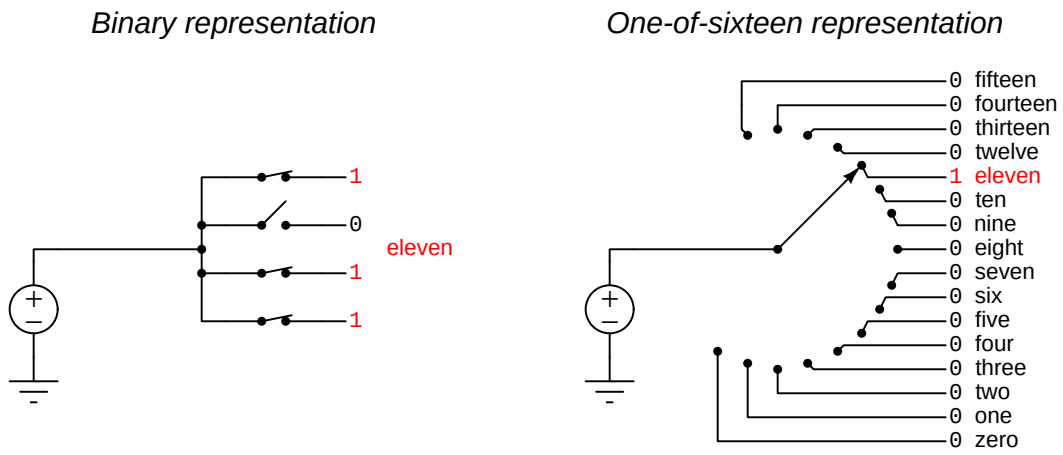
For applications where we only need to represent a single *decimal* character, we may use an abbreviated counting scheme called *Binary-Coded Decimal* (BCD) where four bits represent the decimal digits 0 through 9:

BCD	Decimal	Hex
0 0 0 0	0	0
0 0 0 1	1	1
0 0 1 0	2	2
0 0 1 1	3	3
0 1 0 0	4	4
0 1 0 1	5	5
0 1 1 0	6	6
0 1 1 1	7	7
1 0 0 0	8	8
1 0 0 1	9	9

In a BCD system the bit combinations 1010, 1011, 1100, 1101, 1110, and 1111 are undefined.

3.2 One-of-n numeration

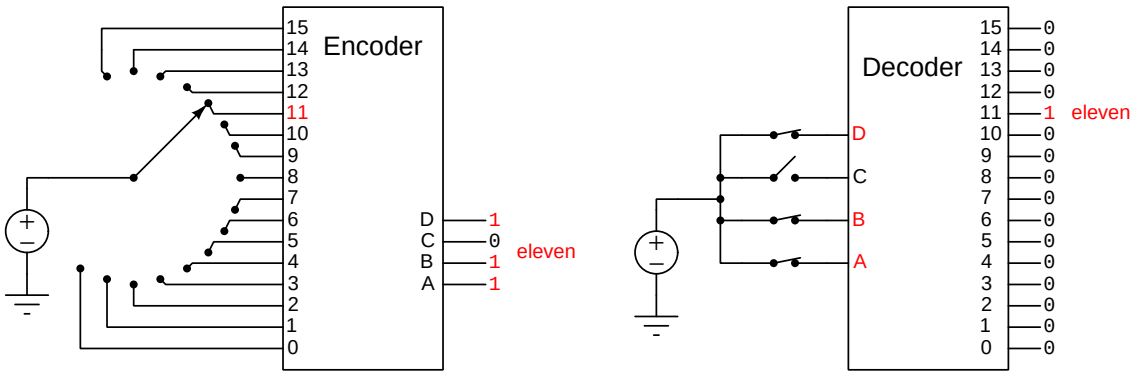
There are applications, though, where it is either useful or necessary to digitally represent a numerical value without using a binary code. For example, we could have a sixteen-position selector switch with one common terminal and sixteen individual terminals, representing the values of zero through fifteen simply by which of the sixteen lines is "high":



Both of these electrical switching networks are digital, in that they represent numerical values in discrete steps. However, their respective design philosophies are quite different: binary is very efficient in terms of switch contacts and wiring but requires an understanding of binary coding to interpret; one-of-n is very inefficient but is practically self-evident in its presentation of the number.

3.3 Encoding and decoding

A digital logic circuit designed to receive a one-of-n digital signal and translate it into a binary code is called an *encoder*. A digital circuit receiving a binary code and outputting a one-of-n (or any other non-binary representation of that same number) is called a *decoder*. Examples of each are shown in the following (simplified) schematic diagrams:



Some encoders and decoders operate on the full binary range available with four bits, having 4 input lines and 16 output lines (decoder), or 16 input lines and 4 output lines (encoder). Others operate on a BCD range, having 4 input lines and only 10 output lines (BCD decoder), or only 10 input lines and 4 output lines (BCD encoder).

Encoders and decoders alike are available as integrated circuits (ICs), and only for very unusual applications would be constructed from separate logic gates.

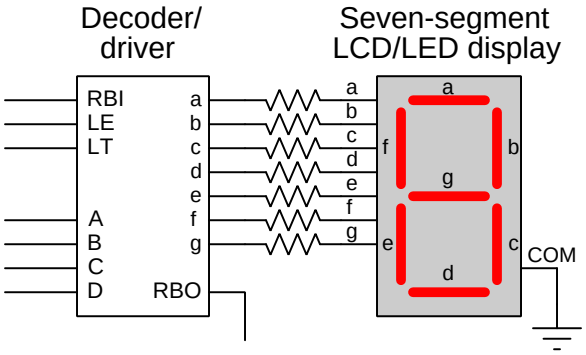
Even though the basic encoder concept relies on a single input being activated to the exclusion of all others (literally, *one*-of-n), some encoder designs anticipate the possibility of multiple inputs being activated whether intentionally or by accident. Such encoders are called *priority encoders*, and they translate the *highest-order* active input into a binary output. For example, if the "eight" and "nine" inputs were simultaneously activated on a priority encoder, only the number `1001` (nine) would appear at its output terminals. If multiple inputs activate on a non-priority encoder, the resulting binary will likely be nonsensical, representing *none* of the activated input terminals.

3.4 7-segment display decoders

Decoders also exist in more than one type. The most fundamental decoder receives a binary or BCD input word and activates one-of-n output lines. However, other forms of decoders exist to convert binary word inputs into specialized bit states necessary to drive decimal indicators such as 7-segment LCD (Liquid Crystal Display) and LED (Light Emitting Diode) displays. A typical array of red LED 7-segment displays appears on the face of this digital clock, each digit powered by the output lines of a 7-segment decoder:

Each decimal digit represented by one 7-segment display is equivalent to a four-bit BCD value.

A typical schematic diagram showing how such a decoder/driver IC would connect to a 7-segment display appears next:



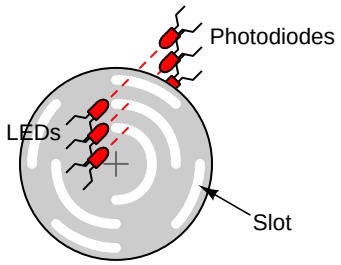
7-segment LED displays typically have seven lines (one for each segment), with either all the LED anodes or all the LED cathodes connected to an additional "common" line. Additional inputs such as LT (Lamp Test), LE (Latch Enable), and RBI (Ripple Blanking Input)[1] provide practical functions for disabling and testing the display regardless of the binary input. In this case, the IC also has an output line called RBO to extend the "blanking" function on to successive decimal digits.

Aside from the special logic features of a 7-segment decoder, these integrated circuits often contain special *driver* circuitry on the output lines matching the needs of the 7-segment display. For example, LED displays require more current per LED than what a typical logic gate might output in order to achieve full brightness, and so a decoder/driver IC contains additional transistor stages on the output lines to handle the extra current demand. Similarly, liquid-crystal displays (LCDs) actually require AC voltage rather than DC to darken each segment, and so decoder/driver ICs designed for LCD applications provide a *phase* input line (PH) to receive the square-wave signal from a low-frequency oscillator.

[1] *Blanking* refers to an optional feature to make leading zero digits blank in a multi-digit display.

3.5 Physical position encoders

When the physical position of an object must be encoded into a series of binary states, it is possible to use the standard binary (counting) code. For example, consider the wheel of a *rotary encoder* with eight sectors, each sector identified by a unique pattern of slots cut through the wheel allowing light to pass through. The purpose of such a device would be to represent the angular position of a shaft by a digital value, for example the position of a machine component, or the position of a manually-adjusted knob:



3-bit rotary encoder

The three circular rows of slots represent three bits' worth of data, and each of the eight identifiable positions on the wheel corresponds to a three-bit binary value. In the illustration shown above, all three light sources and light detectors have clear view of each other through the slots, and so the binary number `111` would be generated in this position (assuming the successful transmission of light through the wheel corresponds to a "1" state). Clockwise rotation of this wheel into the next sector would result in the binary count "rolling over" from `111` to `000`, and further clockwise rotation would result in the binary count incrementing: `001`, `010`, `011`, etc.

Using standard binary encoding certainly makes sense from the perspective of interfacing a rotary encoder to a digital circuit, because the signals generated are completely conventional and would work directly with decoder circuits, multiplexers, and other digital networks designed to input binary words. However, conventional binary code actually suffers from a major limitation when used to encode the position of an object. At the threshold between successive "count" values where more than one bit changes (for example, from `111` to `000`), anything short of *perfect* alignment between the slots on the encoder wheel will result in false transitional count values.

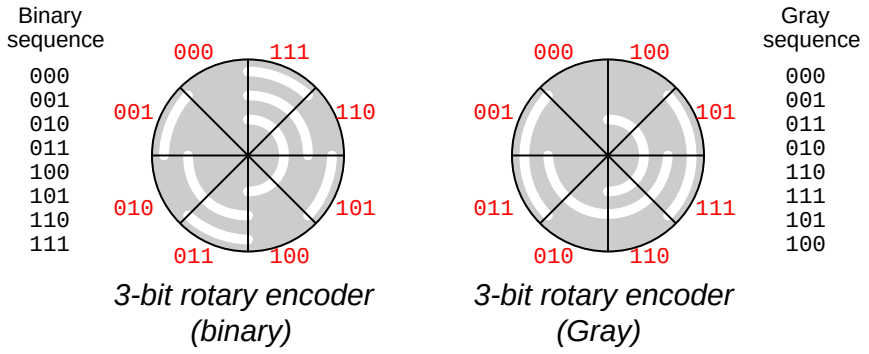
A thought experiment is helpful to picture the problem. Imagine the middle slot on the wheel was cut slightly too far counter-clockwise, so that when the wheel is rotated just past the end of the "seven" sector into the "zero" sector the middle photodetector continues to receive some light after the other two detectors sense darkness. This would result in the count sequence starting at `111`, then briefly transitioning to `010` before settling on `000`. Any circuit receiving this encoder's signal would "think" the shaft instantaneously moved from the "seven" position to the "two" position and then once again made an instantaneous jump to the "zero" position: an impossible feat. This transient error may or may not be significant depending on the application of the encoder, but it is none the less *wrong*. Furthermore, the root cause of such a counting error is the binary coding system itself, which demands absolute perfect physical alignment between the slots to guarantee error-free transitions between all sectors.

A very important point to note here is that these transient counting errors, and the root need for

perfect alignment between the slots cut into the encoder wheel, really only matters at the transitions between count values exhibiting multiple bit changes. Transitioning from 111 to 000 is the worst-case scenario because all three slots would have to end at exactly the same angular position for a clean 111-to-000 transition. However, going from 000 to 001 would not pose quite the same problem if the LSB slot were slightly out of alignment: the transition to 001 might not occur at precisely the right shaft angle, but at least there would be no completely nonsensical counts generated.

An ingenious solution to this problem is to redefine the binary combinations representing each increment in position such that only one bit changes during each transition from one position to the next. The inventor of this technique was a Bell Laboratories researcher named Frank Gray, and such a code is typically referred to as *Gray code* in honor of his innovation. Details of his invention, including instructions on how to determine bit sequences for a word of any bit-length, are presented in the Historical References chapter, section 4.1 beginning on page 22.

Two rotary encoder wheels are shown in the following illustration, the slots in the left-hand wheel representing conventional binary code and the slots in the right-hand wheel representing Gray code. Each of the eight sectors are divided and labeled for ease of interpretation:



Unlike binary, decimal, octal, or hexadecimal, *there are no definite place-weights in Gray code.* That is to say, in Gray code there is no one's place, two's place, four's place, etc. as there is in binary. The number of unique bit combinations for any Gray code is still equal to 2^n (where n is the number of bits), and there is still a definite ordering of bits from most-significant (MSB) to least-significant (LSB) in Gray code, but the similarities end there.

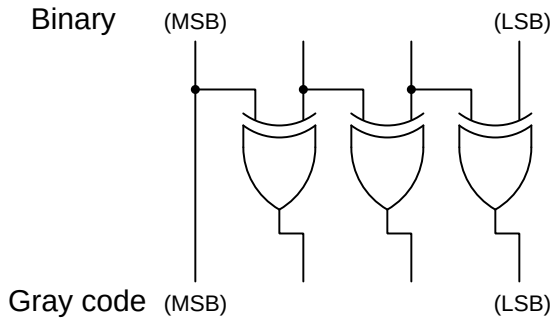
Gray code finds frequent application in position-sensing systems such as rotary and linear encoders, but it is really applicable anywhere slight alignment errors (position errors, timing errors) between parallel bits must be tolerated.

A table comparing a four-bit binary count sequence to Gray code is helpful to reveal patterns:

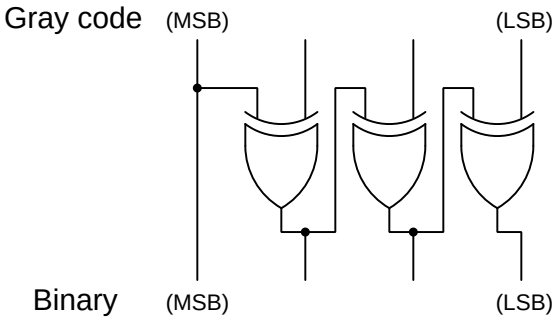
Quantity	**Binary**	**Gray code**
Zero	0000	0000
One	0001	0001
Two	0010	0011
Three	0011	0010
Four	0100	0110
Five	0101	0111
Six	0110	0101
Seven	0111	0100
Eight	1000	1100
Nine	1001	1101
Ten	1010	1111
Eleven	1011	1110
Twelve	1100	1010
Thirteen	1101	1011
Fourteen	1110	1001
Fifteen	1111	1000

If we examine the binary counting sequence from zero to fifteen you will notice how the one's place bit alternates back and forth between 0 and 1, while the two's place bit does the same half as often (e.g. 00110011 rather than 01010101). With each successive place-weight the *frequency* of the bit's alternation is halved. In Gray code, however, the least-signficant place's bit alternates every-other time in the sequence (at the same frequency as the binary code's *two's place* bit), with each successive bit toward the most significant alternating at half the frequency like binary, and with the added twist that each of these alternating-bit patterns begins half-way through the zero states (e.g. 00111100 rather than 00001111).

Four-bit Gray code may be converted into 4-bit binary through the use of Exclusive-OR (XOR) logic functions, as shown below:



Conversely, four-bit binary may be converted into four-bit Gray code by a slightly different network of XOR logic functions as we see here:



An excellent "active reading" exercise is to take any of the corresponding Gray-code and binary code pairs shown in the previous table and see for yourself how these XOR gate networks convert from one to the other by placing the bit-pattern of one code at the inputs and then working through the truth table for each XOR gate to see how the output code develops.

Chapter 4

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.

4.1 Frank Gray's code

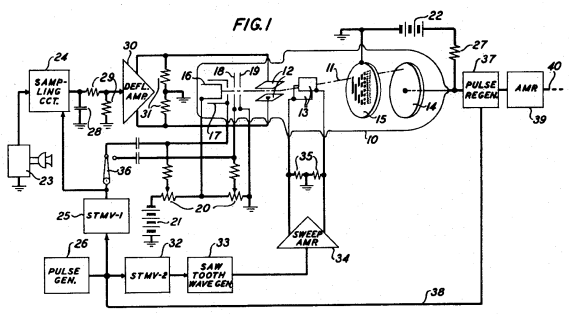
In 1947 a Bell Telephone Laboratories researcher named Frank Gray devised a way to improve a particular form of digital signal communication by replacing standard binary coding with what he called *reflected binary code*. This new code scheme came to be known by his last name, *Gray code*.

To understand Gray's invention, one must grasp the application for which it was invented. Gray's employer, Bell Telephone, designed, built, and maintained long-distance telephone communication networks across the United States. They were interested in converting audio telephone signals into digital form suitable for transmission over long distances with little corruption from external noise and cable attenuation. The digitization of audio telephone signals took the form of voltage pulses following each other in rapid succession, commonly known as a *serial data* stream. Sets of contiguous bits in this data stream represented individually-sampled values of the analog signal.

Gray begins his narrative on page 5 of the patent by explaining how binary coding works:

> This invention relates to pulse code transmission and particularly to the coding of a message signal in a novel code and to the decoding thereof.
>
> In communication by pulse code transmissions the instantaneous amplitudes of a message to be transmitted are successively sampled and each of the successive samples is translated into a code group of on-or-off pulses. By reason of the on-or-off character of the pulses, such a code is denoted a binary code. The number of pulse positions in a code group is the same from group to group. With five such positions the code is a 5-digit binary code. With seven it is a 7-digit binary code; and in general, with in such positions it is an n-digit binary code. A code pulse group of n pulse positions may contain any number, from 0 to n, of "on" pulses. In the conventional n-digit binary code, the number and arrangement of pulses is in accordance with the conventional binary number notation. Thus, for example, with three digits, the number five is written in the conventional binary number notation as 101. Correspondingly, in the conventional 3-digit binary pulse code, the pulses occur in the time sequence P, −, P, where "P" stands for an "on" pulse and "−" stands for an "off" pulse; i. e., a blank pulse position. [page 5]

The technology of that era for converting an analog audio signal into a series of digital pulses used a vacuum tube called a *cathode ray tube*, or CRT. Many different types of CRTs were manufactured, and they all shared the common features of an "electron gun" assembly at one far end generating a thin beam of high-speed electrons, as well as means to bend or deflect this beam before it struck an object at the other end of the evacuated tube. CRTs were once used for television and computer terminal displays, but in this application functioned as a form of *analog to digital converter*. The first page of Gray's patent contains a figure showing the general construction and supporting circuitry of the CRT:



Gray's narrative resumes with a description of the CRT and the analog-to-digital conversion process:

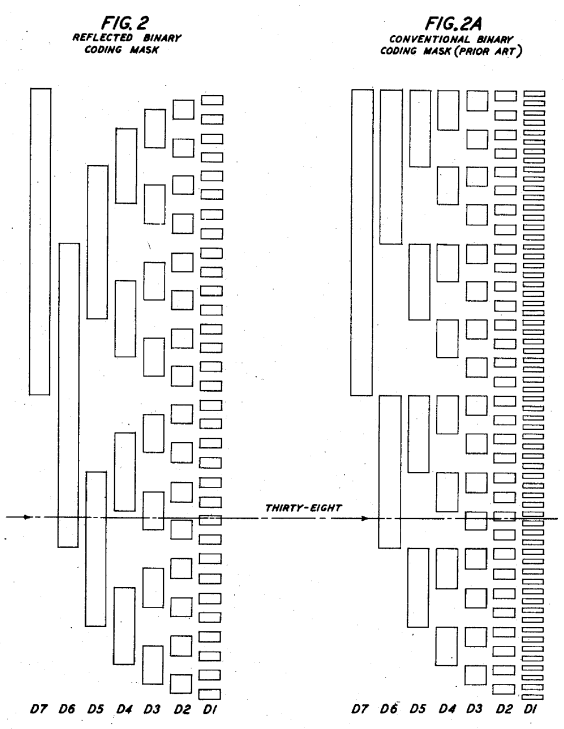
> In Italian Patent 437,300, published June 30, 1948, there is described an instrument for translating message signal samples into code pulse groups in the conventional binary code. In brief, it comprises a cathode beam tube having a coding mask, an electron gun for projecting an electron beam toward the mask, a collector anode for receiving electrons which pass through the mask and deriving pulses from them, means for deflecting the cathode beam in one direction along the mask to a location proportional to the signal sample amplitude, and means for sweeping the beam in a perpendicular direction across the mask between successive sample controlled deflections. As currently employed, the mask comprises a rectangular array of apertures arranged in n columns and 2^n rows, where n is the number of digits of the code. Each aperture row corresponds to a unique value of the signal-controlled beam deflection. The apertures of the various rows are located in conformity with the location of the 1's in a tabulation of successive binary numbers, while the blank portions of the mask are located in conformity with the 0's in the same tabulation. Thus, when the cathode beam is deflected under control of the signal to a particular aperture row and thereupon swept laterally along this row, a train of current pulses may be drawn from the collector whose location on the time scale is in accordance with the arrangement of the 1's and 0's in the binary number whose value is equal to the value of the signal sample being coded. `[page 5]`

The use of a cathode ray tube to convert an analog voltage signal into a serial data stream is rather ingenious[1] and deserves some elaboration. The beam of electrons repeatedly sweeps across the face of the mask in one axis by the direction of a "sawtooth" wave voltage signal applied to a

[1] Not only is this tube design ingenious, but it also stands as an example of just how flexible vacuum tube technology was. Even though vacuum tubes have been made obsolete by semiconductor electronic devices for all but the most

pair of metal plates located to either side of the beam. Another pair of plates perpendicular to the first pair deflects the beam in the other axis, this deflection occurring at the whim of the sampled analog signal. A sample-and-hold circuit (24 in the diagram) captures the analog signal value at the start of the beam sweep and holds that signal voltage constant for the duration of the sweep, so that the beam traces only straight-line paths across the mask. At the conclusion of each sweep, the sample-and-hold circuit re-reads the analog input signal and holds it steady for the next sweep. Holes placed in the mask allow the beam to pass through at certain points where it strikes an anode surface (14) and registers outside the tube as a current pulse. Therefore, as the beam sweeps across a row of the mask, an electrical pulse sequence develops at the anode corresponding to the placement of the holes in that row of the mask.

Figure 2 (from page 2 of the patent) shows two possible mask patterns, one standard binary and the other Gray's "reflected binary" code:



The electron beam's periodic sweep crosses the face of the mask in the horizontal axis (as the masks are shown above), so that the beam "probes" one row at a time. The vertical position of the beam on the mask is determined by the sampled analog signal. Each column of holes on the mask represents one bit (one "digit" as described by Gray) in the binary word. For the sample masks shown, a seven-bit binary sequence is generated for every sweep of the beam from left to right. For an

specialized applications, and for very good reasons (e.g. physical ruggedness, energy efficiency, size, service life), this does not change the fact that this one technology was capable of so many different types of electronic functions. The manipulation of electrons traveling through a vacuum lends itself to a wide range of applications, including amplification, oscillation, rectification, signal modulation, visual display, optical sensing, memory, etc.

analog signal corresponding to the value 38 (shown on the figure), the beam generates the sequence 0100110 as it passes from left to right over that row of holes. Tallying each bit's place-weight, we get 32 + 4 + 2 = 38. A maximum-value analog signal would deflect the beam all the way to the top row where the sequence would be 1111111 (decimal value 127). A minimum-value signal would pull the beam all the way to the bottom to generate 0000001 (decimal value 1). Thus, the sweeping electron beam and the mask together serve as the heart of a seven-bit analog-to-digital converter with a *serial* output.

Gray's narrative continues on page 1, describing the conventional binary coding of the mask shown in Figure 2A. Bear in mind that Gray consistently uses the word "digit" where we would now say "bit":

> It is a characteristic of the conventional binary number notation that a value change of unity may be reflected in the binary number notation by a simultaneous change in several of the digits. Thus, for example, with four digits the number seven is represented by 0111 while the next number, eight, is represented by 1000. In the course of changing the value by one unit, each of the four digits has been changed.
>
> This characteristic of the conventional binary number notation is duplicated in the coding mask of the Italian patent above referred to, and it is a consequence of the resulting arrangement of the apertures that a wander of the beam in the course of its lateral sweep from the correct aperture row to the row immediately above or below it, may result in a coding error which is far greater than the beam deflection error. Various arrangements have been suggested for reducing this coding error by constraining the cathode beam to start its sweep at the correct row and to remain there throughout the sweep . . . All such arrangements involve complexity of apparatus in various degrees.

Gray describes how relatively minor errors in the beam's lateral trajectory ("wander") could cause the beam to graze the holes (or non-holes) of an adjacent row, possibly resulting in a large coding error due to the fact that bit-values often change greatly from one binary integer to the next. To use his example of seven (0111) versus eight (1000), one can imagine the beam properly deflected to the "eight" row and beginning its left-to-right sweep, successfully passing through the first hole of the "eight" row to create a "1" pulse at the capturing anode, but then a slight downward drift of the beam later in that same sweep might cause it to pass through the three holes of the "seven" row to create three more "1" pulses, the result being a serial data word of 1111 (fifteen) instead of either 1000 (eight) or 0111 (seven). In this example, a vertical alignment "wander" of just 1 "count" results in a digitized error of *seven* counts (i.e. fifteen instead of eight).

Next, Gray begins to describe how his new coding scheme is superior to conventional binary.

> It is a principal object of the present invention to reduce the coding errors in a pulse code transmission system. A more specific object is to provide a pulse code, and a corresponding coding mask, in which the coding error is never greater than the beam deflection error.
>
> Another object is to simplify the manufacture of a coding mask. [page 5]
>
> The above objects are attained in accordance with the invention by the selection of a novel form of the binary pulse code which differs from the conventional form by virtue of

a rearrangement of the pulses of the various pulse groups in such a way that the sequence of "on"-pulses [page 5]

and off-pulses which form a pulse group representing a particular signal amplitude differs in only one pulse position from the sequences representing the next lower amplitude and next higher amplitude. The new code is no longer similar to the accepted binary number notation. When it is embodied in a coding mask, the arrangement of the apertures of any one row differs from that of the rows above and below it in not more than one aperture. The resulting mask has certain valuable auxiliary properties and aspects. First, the smallest apertures, that is to say the apertures of the various rows in the column of least digital significance, are twice as large as the apertures of the conventional binary code mask. This makes for ease of manufacture. Second, all of the apertures of the mask, with the sole exception of the single aperture of greatest digital significance, are symmetrically arranged-about a transverse center line. This permits treating the largest digit aperture as an index of polarity only, rectifying the wave to be coded, sampling the rectified wave, and coding the samples using only one half of the coding mask. At the price of some increased complexity of associated apparatus, this greatly reduces the physical dimensions of the coder tube itself. Third, for signals of normal average amplitude range, the beam deflections seldom extend beyond the aperture of the column of second greatest digital significance, so that in the resulting coded signal, the $(n-1)^{th}$ pulse position is nearly always filled. This uniformly filled pulse position affords a convenient source of marker pulses for use in holding a receiver in correct synchronism with the transmitter. [page 6]

Later in the patent (beginning on page 7) the inventor describes his process for developing a "reflected binary" code counting sequence, and in doing so the full meaning of the word "reflected" becomes clear:

> The manner in which the primary reflected binary number system is built up will now be explained.
>
> First: write down the first two numbers in the 1-digit orthodox binary number system, thus:
>
> ```
> Zero     0
> One      1
> ```
>
> Note that the symbols differ in Only one digit. Second: below this array write its "reflection" in a transverse axis:
>
> ```
> Zero     0
> One      1
> ----------
>          1
>          0
> ```
>
> The symbols still differ in not more than one digit. However, the first is identical with the fourth and the second with the third.
>
> Third: to remove this ambiguity, add a second digit to the left of each symbol, 0 for the first two symbols and 1 for the last two, thus:
>
> ```
> Zero    00
> One     01
> Two     11
> Three   10
> ```
>
> and identify the last two symbols with the numbers "two" and "three." Each symbol is now unique and differs from those above and below it in not more than one digit. The array is a representation of the first four numbers in the primary 2-digit reflected binary number system.

The process is next repeated, giving –

First:

```
Zero    00
One     01
Two     11
Three   10
```

Second:

```
Zero    00
One     01
Two     11
Three   10
----------
        10
        11
        01
        00
```

Third:

```
Zero   000
One    001
Two    011
Three  010
Four   110
Five   111
Six    101
Seven  100
```

[page 7]

Gray continues his exposition on this "reflected binary" coding system on page 8 of the patent by describing alternative forms. In every case, though, the coding sequence exhibits the same important property of exhibiting just one bit-change per step, and also the property of all bits being symmetrical about the mid-point except the most-significant bit (MSB).

> Other secondary forms of the reflected binary code may be obtained in various ways. Vertical columns may be interchanged. For any such transposition the pattern may be split along any horizontal division line between rows, and the lower part placed above the upper part, to give a new pattern with the same properties as the primary one. Again, the initial process of building up the code by reflection may be modified, giving two alternatives for the 1-digit code, four for the 2-digit code, and so on. The four alternatives for the 2-digit code are tabulated below. Of these the first is the primary one discussed above, the others being variants.
>
> | **Number** | **Primary** | **First variant** | **Second variant** | **Third variant** |
> |---|---|---|---|---|
> | Zero | 00 | 10 | 01 | 11 |
> | One | 01 | 11 | 00 | 10 |
> | Two | 11 | 01 | 10 | 00 |
> | Three | 10 | 00 | 11 | 01 |
>
> [page 8]

Chapter 5

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?

5.1 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.

5.1.1 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.

5.1.2 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.

Binary number

Encoding

Decoding

Lamp test

Enable

Strobing

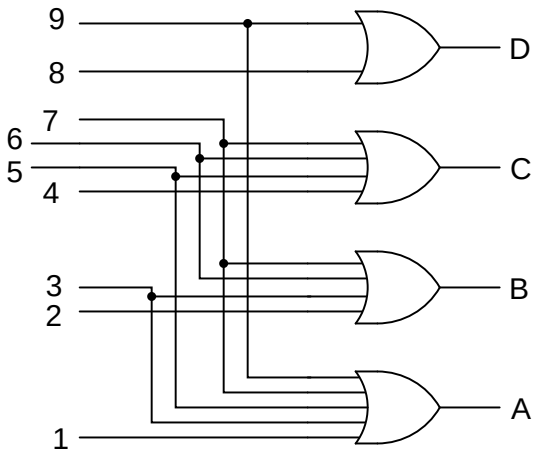
Blanking

Light-emitting diode

Liquid crystal display

5.1.3 OR-based encoder circuit

Explain how the following decimal-to-BCD encoder circuit works:



Also, determine which output (D or A) is the *most significant bit* of the BCD output.

Challenges

- What would have to be modified in order to turn this into a hexadecimal-to-binary (16 line to 4 line) encoder? Would any additional OR gates have to be added to the circuit?

5.1.4 74HC147 encoder

Identify the necessary input states to make a 74HC147 decimal-to-BCD encoder generate the binary code for the number *seven*, and how that quantity would be represented on the output (Y) lines.

Challenges

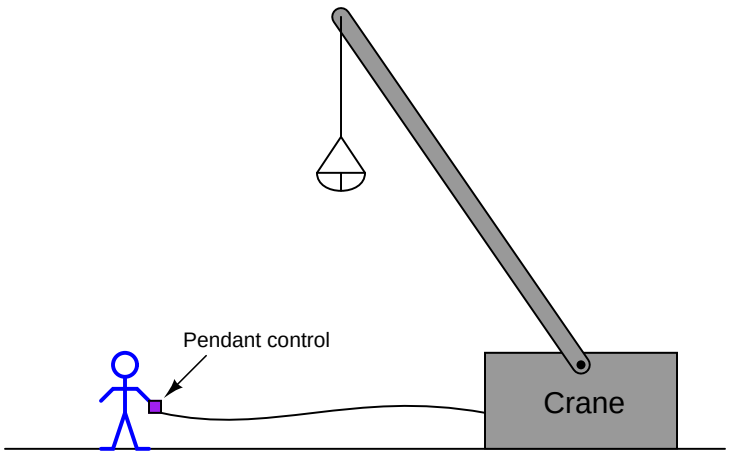
- Identify the acceptable power supply voltage range for this IC.

5.1.5 Pendant control

Suppose a crane has fifteen hydraulic solenoid valves controlling its motion:

- Tilt up (fast)
- Tilt down (fast)
- Tilt up (slow)
- Tilt down (slow)
- Turn left (fast)
- Turn right (fast)
- Turn left (slow)
- Turn right (slow)
- Cable up (fast)
- Cable down (fast)
- Cable up (slow)
- Cable down (slow)
- Bucket open (fast)
- Bucket open (slow)
- Bucket close (slow)

You are part of a team building a remote "pendant" control for this crane with fifteen buttons on it for controlling each of the fifteen solenoid valves. This control pendant connects to the main system by a multi-conductor cable, but you really want to limit the number of conductors in this cable to keep it as light-weight as possible:



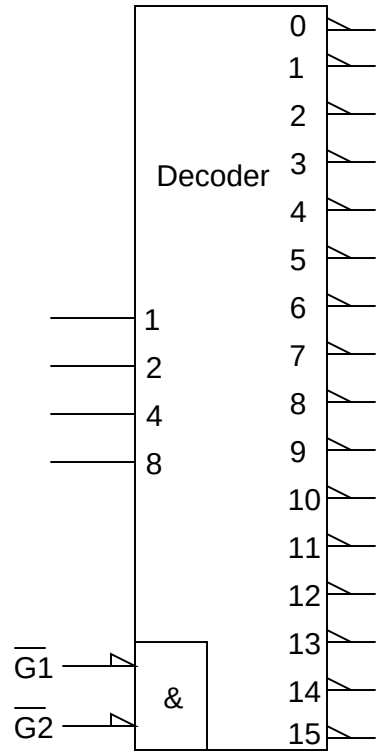
Draw a simple schematic diagram showing how a digital encoder and decoder circuit pair could be used to relay the same fifteen commands across fewer cable conductors, compared to if we used one conductor per pushbutton switch.

Challenges

- Identify a disadvantage of this crane control strategy, compared to using a thicker cable where each pushbutton has its own dedicated conductor?
- Describe some of the safety hazards intrinsic to this project, and specifically identify at least one circuit fault that could result in a destructive outcome.

5.1.6 74HC154 decoder

The type 74HC154 integrated circuit is a standard TTL decoder, 4-line to 16-line. Its block symbol looks like this:



What do the "wedge" symbols next to the output lines represent? Also, what purpose do the $\overline{G1}$ and $\overline{G2}$ inputs serve, and why is there an ampersand character (&) next to them?

Challenges

- Explain how you could combine two of these ICs to create a 5-line to 32-line decoding circuit.

5.2 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.

5.2.1 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.

5.2.2 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.*

5.2.3 Integer conversion table

Complete this table, performing all necessary conversions between numeration systems of these unsigned integer quantities:

Binary	Octal	Decimal	Hexadecimal
10010			12
	134	92	
	32		1A
110111	67		
1100101		101	
		290	122
1111101000		1000	
	336		DE
1011010110			2D6

Challenges

- Which type of conversion do you find most difficult to perform, and why is that?

5.2.4 SOP expressions for decoder circuit

The truth table shown here is for a 4-line to 16-line *binary decoder* circuit:

D	C	B	A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

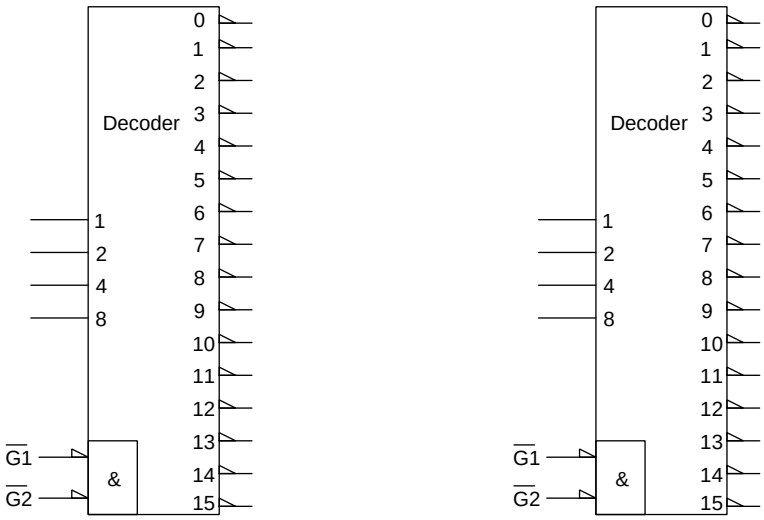
For each of the sixteen output lines, there is a Boolean SOP expression describing its function. Just for example, write the Boolean expressions for output lines 2, 5, 8, 11, 13, and 14.

Challenges

- Based on what you see here, what kind of logic gate circuitry is a decoder such as this comprised of?

5.2.5 5-line to 32-line decoder

Each of these 74HC154 decoder ICs is 4-line to 16-line:



Show how they could be connected together to form a 5-line to 32-line decoder circuit.

Challenges

- Identify how to connect *four* 74HC154 decoders together to form a 6-line to 64-line decoder circuit.

5.3 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.

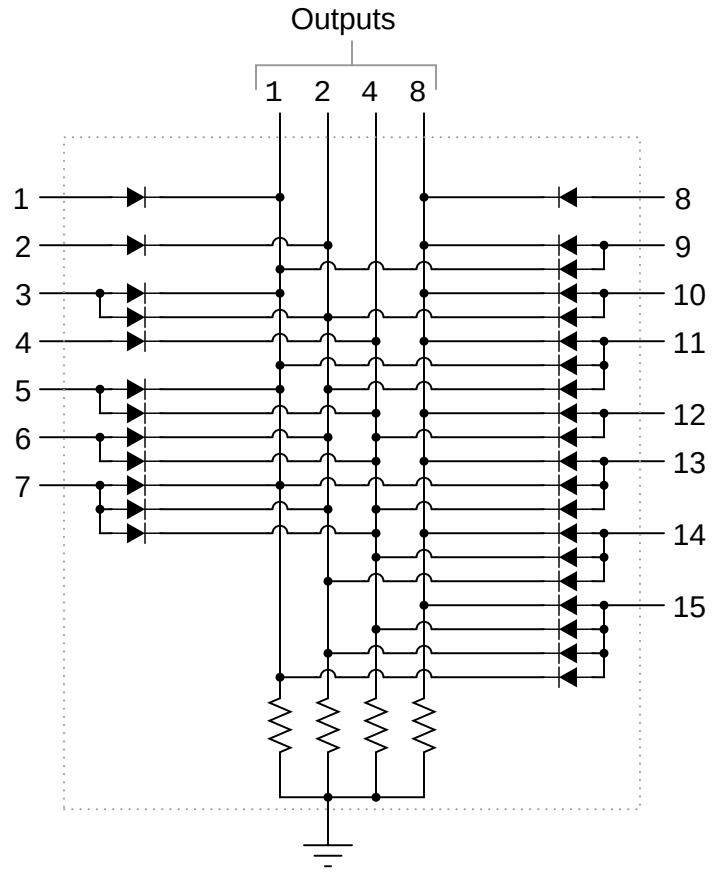
Another means of checking your diagnostic answers, where applicable, is to use circuit simulation software to explore the effects of faults placed in circuits. For example, if one of these diagnostic questions requires that you predict the effect of an open or a short in a circuit, you may check the 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.

5.3.1 Customer complaint

Having learned how to build simple encoder circuits using diode networks, you set out to form your own encoder manufacturing company: *Encoders, Inc.* After agreeing on a policy of truth in advertising, your board of directors drafts this slogan:

"Our encoder circuits are better because there's less to break"

After months of hard work, you unveil your latest masterpiece, the 16-line to 4-line encoder:



However, your first customer has a complaint with your encoder circuit. He claims it often outputs false codes. After sending it back to your workshop for warranty repair, you determine there is nothing wrong with the encoder circuit itself: it always outputs the correct codes when you energize the appropriate inputs. Perhaps the problem is in how the customer is using it.

You then telephone the customer and ask him how he is using the encoder. He tells you it is used as part of a fault diagnostic circuit for an important piece of machinery. Each input of the encoder is connected to a different sensor on the machine (low oil pressure switch, high temperature switch, out-of-limit travel switches, etc.), and then the encoder outputs drive a four-LED display for maintenance technicians to view. They would have rather used a separate LED for each "trouble"

sensor, but the display panel was too small to accommodate fifteen LEDs, so they decided to use four LEDs and an encoder, having their technicians interpret a binary code to determine which of the fifteen sensors is activating.

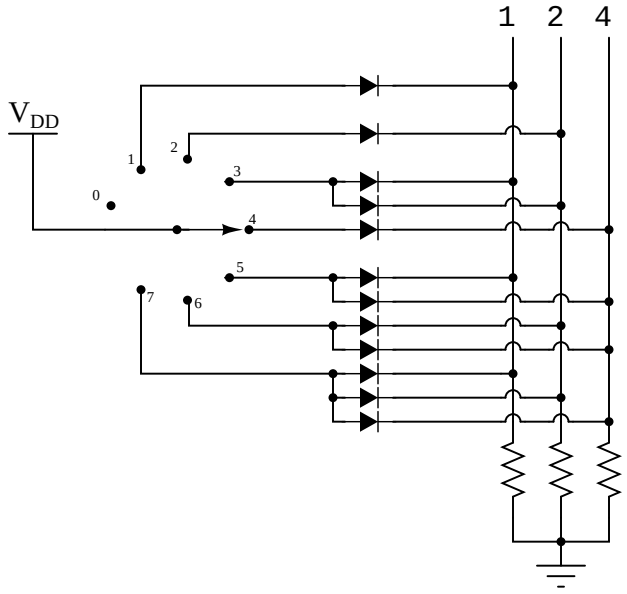
To the best of your ability, determine why your company's flagship encoder circuit sometimes produces false codes in this application. Then, recommend a solution for your customer.

Challenges

- Is this encoder design compatible with TTL logic, CMOS logic, or both?

5.3.2 Faulted encoder circuit

Identify which diode is failed in this circuit, given the following truth table (showing the actual operation of the encoder circuit, not what it *should* do):



Switch position	Output code
0	000
1	001
2	010
3	011
4	100
5	001
6	110
7	111

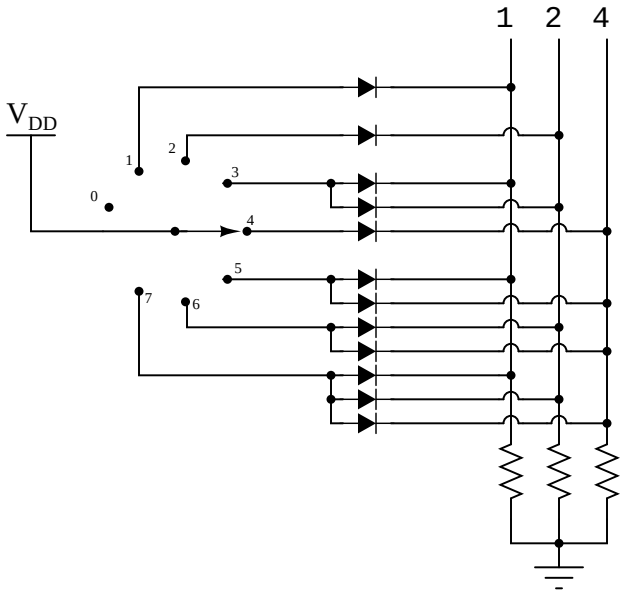
Be sure to specify whether you think the failed diode is *open* or *shorted*.

Challenges

- Suppose that same diode was failed in the opposite mode. How would that fault manifest in the encoder's operation?

5.3.3 Another faulted encoder circuit

Identify which diode is failed in this circuit, given the following truth table (showing the actual operation of the encoder circuit, not what it *should* do):



Switch position	Output code
0	000
1	001
2	110
3	111
4	100
5	101
6	110
7	111

Be sure to specify whether you think the failed diode is *open* or *shorted*.

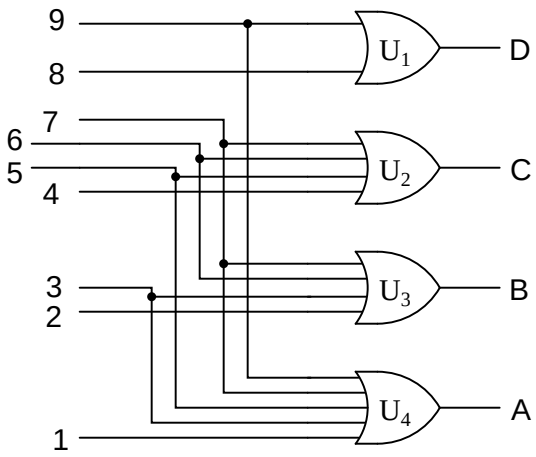
Challenges

- Suppose that same diode was failed in the opposite mode. How would that fault manifest in the encoder's operation?

- Explain how we can tell the diode connecting position 7 to line 2 is *not* failed.

5.3.4 Failed OR gate in encoder

Suppose OR gate U_3 were to fail with the output terminal always high. Which output codes would be affected by this fault?



Challenges

- If you didn't know which gate was faulted, but only had access to the list of codes generated for each input activation, how would you diagnose the fault?

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

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.

18 November 2024 – minor edits to the Tutorial.

9 November 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.

14 April 2024 – added Historical References chapter with a section on Frank Gray's code.

20 April 2023 – added Tutorial content on rotary encoders, and added a Quantitative Question on numeration conversion.

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.

26 April 2022 – updated the Introduction chapter to be consistent in format with other modules, including a foundational concepts list and a list of recommended questions. Also added some instructor comments and corrected a minor typographical error.

18 February 2022 – corrected a typo where "LE" was defined as "Lamp Enable" when it should have been "Latch Enable".

1 December 2021 – deleted redundant word "necessary" in the 74HC147 encoder Conceptual Reasoning question.

27 July 2021 – divided Tutorial into sections, and also added content to the Tutorial regarding BCD numeration as an alternative to binary. Also added a Case Tutorial chapter with a section showing how to use a CD4511 BCD-to-7-segment decoder/driver and common-cathode 7-segment LED display.

10 May 2021 – commented out or deleted empty chapters.

15 August 2020 – corrected minor typographical error in the Tutorial.

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

20 November 2019 – continued writing Simplified Tutorial. The Full Tutorial is de-prioritized right now, because all I need for my immediate teaching purposes is a Simplified Tutorial.

19 November 2019 – document first created.

Index