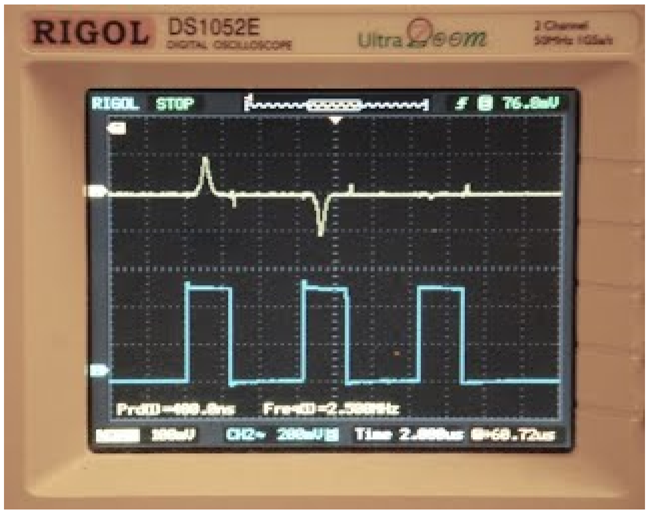
Wayne's Tinkering Page

One Bit Ferrite Core Memory

When I was in my late teens and early twenties, the computers I used were all based on ferrite core memories. It's a pretty unique, now mostly forgotten technology that relies on magnetic hysteresis to flip small toroid "cores" made from a special type of hard ferrite (the toroids used to make transformers use a different, "soft" variant of ferrite) between two magnetic states. If you thread a wire through a core and then pulse it with a current of sufficient strength, the magnetic domains will align such that the core will become magnetized in the one circular direction, or the other. The magnetic property that makes this work as a memory technology is that when a core flips from one state to another, it kicks back a small induced current pulse that can be sensed with a simple transistor amplifier. This kickback pulse will not happen, when the pulse is aligned in the direction in which the core is already magnetized. If you designate one magnetic orientation as "1" and the other as "0", the kickback pulse will happen when a core flips from 1 to 0, or from 0 to 1. This induced pulse can be seen in the top scope trace shown below. You'll notice that there is a delay between the current pulse that flips the core and the time when the kickback pulse appears.



Drive pulses (lower) & induced pulse (upper)

The trick that makes this into a scalable memory system, is to thread two wires through a set of cores arranged in an X/Y grid. Then, to address an individual core, a current pulse of 1/2 the strength needed to flip a core from one state to another is fed through a selected pair of X and Y wires. Only the core at the intersection of these two pulses will receive a magnetic force sufficient to flip the core, so this approach is called coincident current selection. A third wire called the "sense" wire is then threaded through all the cores in a special, serpentine pattern that tries to cancel out the effect of the current induced by the X/Y drive lines. I have no idea how, but I've read that commercial core memory systems were built using hundreds of thousands of nearly microscopic cores that were threaded together by hand. I admire the manual dexterity of anyone who could do this.

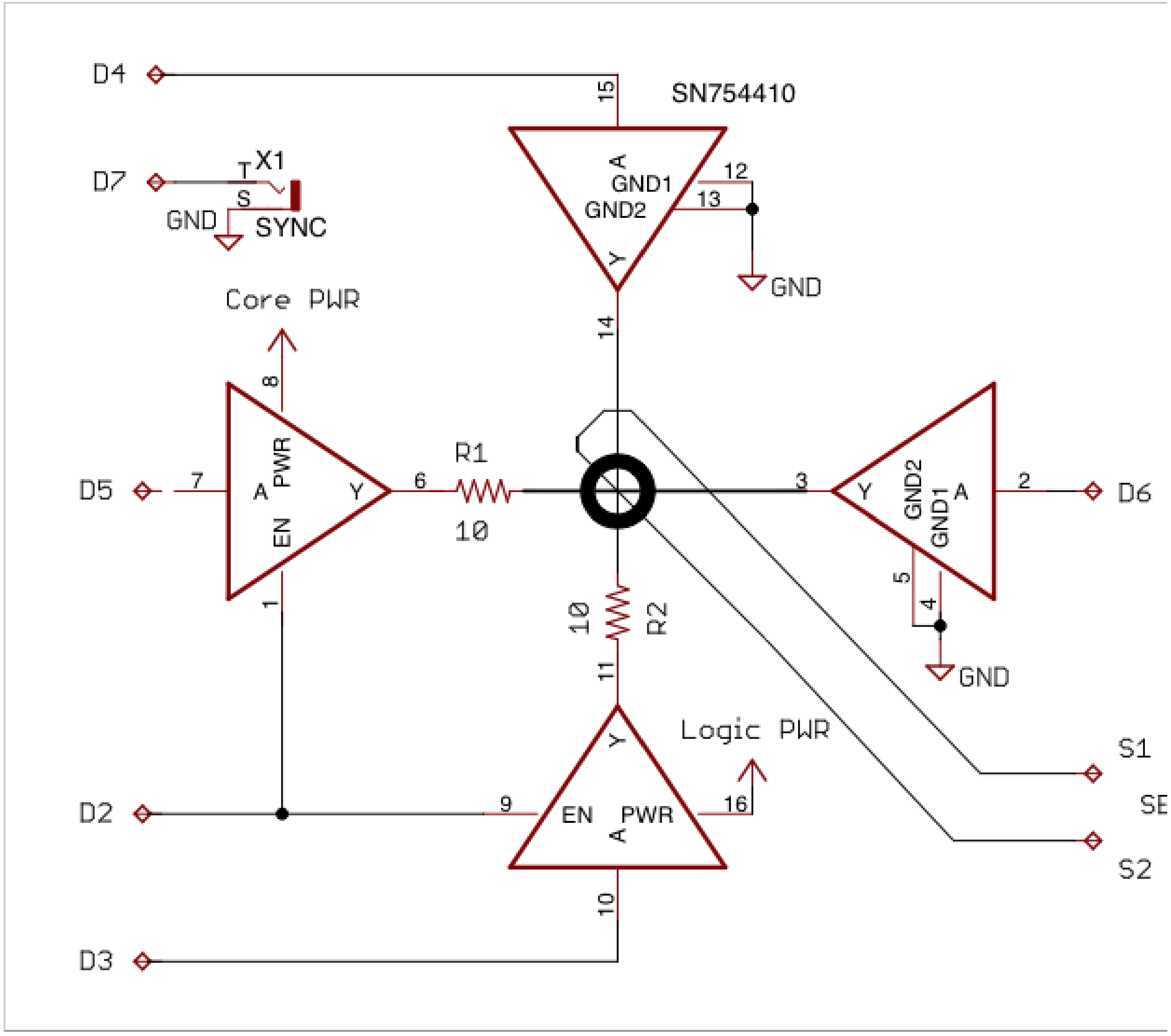
My One Bit Wonder

Before solid state memories appeared and devastated the market for magnetic core, I had schemed to build my own computer using a surplus core stack I kept in my garage for years and years. But, the new darling of the era, the Intel 8008 caught my eye and these plans were long forgotten. But, in the back of my mind, I've always wondered if I had the "chops" to build a magnetic core memory that would actually work. I'm a pretty decent digital designer, but my analog skills are not as strong and, to be honest, the whole process seemed a bit funky. But, the bug to try bit me one day, so I set out to do a few experiments and find out if I could make it all work. Most of the components used to build these old memory systems, such as X/Y drivers and sense amplifiers have been obsolete, and virtually unobtainable for many, many years. But, with some creative substitutions, I've come up with a simple, one bit design that demonstrates all the basic principles of a working, coincident current, ferrite core memory.

Before I could build the support circuitry, I need to get my hands on some actual, hard ferrite cores. After spending a lot of time trying to find something in current production, I started looking on eBay hoping to find an old core plane at a reasonable price. But, it seems these must be in demand by collectors, or museums, as most of the ones I saw sold for in excess of $100, or more. I finally noticed a guy from Bulgaria selling small containers of the raw cores for around $30. So, I bid on some and, a few weeks later, I had in my hand a round plastic container filled with some 45,000 ferrite toroids, each about 1 mm in diameter which, as you can see in the photo below, is pretty damn small. But, from my reading on the subject, I realized that smaller cores would take less power to flip, so I figured that would simplify the X/Y drive requirements.

1 mm Ferrite Cores, fresh from eBay

Based on my research, I estimated the X/Y drive lines needed to handle less than 1 Amp of current, so I settled on using a Dual H-Bridge made by TI (SN754410) to flip a single test core. After some experimentation, I found that with 5 volts as the drive voltage, I could use a 10 Ohm, 2 Watt resistor in series with each X/Y output line to produce the 1/2 current pulse. The SN754410 uses split supplies (one for the internal logic and one for the output transistors), so I used my trusty 5 Volt, 3 Amp "ArcherKit" power supply for the drive current (attached to pin 8 of the SN754410.) I used an Arduino Nano to supply the control signals and connected its 5 Volt power output to supply the SN754410's logic circuits (pin 16 of the SN754410.) The schematic for this design is shown below. The pins labeled D2-D7 show where to connect to the Arduino Nano. (Note: pin D7 is used to supply a trigger signal to the scope to take the photo shown above.)



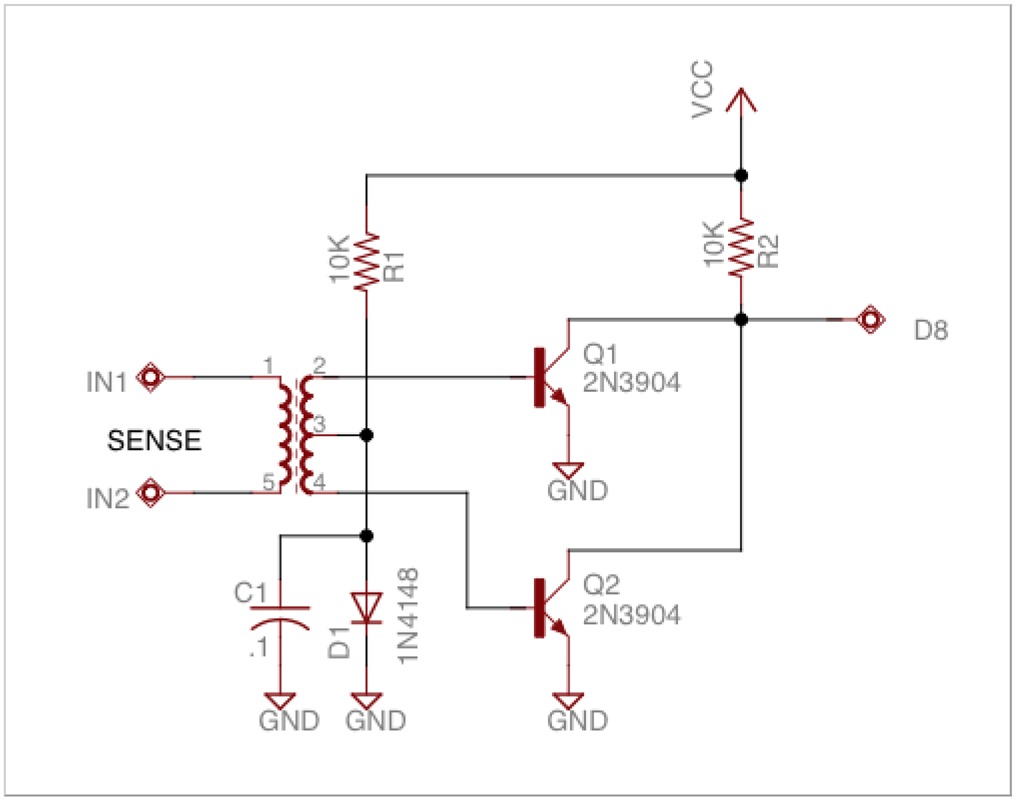
One bit X/Y Drive Circuit

The test code shown below writes a repeating set of three pulses that flip the core to the "1" state, then to the "0" state, then to the "0" state again (which leaves the core in the "0" state for the next series of pulses.) To write a "1" to the core, the code sets D3 and D5 high, D4 and D6 low and then pulses D2 for about 2 microseconds. To write a "0" the code sets D3 and D5 low, D4 and D6 high and again pulses D2. Actually, the designation of one state, or the other as the "1", or the "0" state is completely arbitrary. The important idea is that you select one state as the "1" state so you can test for cores set to this state by forcing them to the opposite state and watching for the kickback pulse. It you see the pulse the bit *was* in the "1" state. if not, it was already in the "0" state.

```
#define  PULSE  6
#define  DELAY  10

#define  EN    0x04
#define  SYNC  0x80
#define  WR    0x28
#define  RD    0x50

void setup() {
  Serial.begin(19200);
  DDRD = DDRD | 0xFC;  // D2-D7 are outputs
  DDRB = DDRB | 0xFE;  // D8 is input
}
```

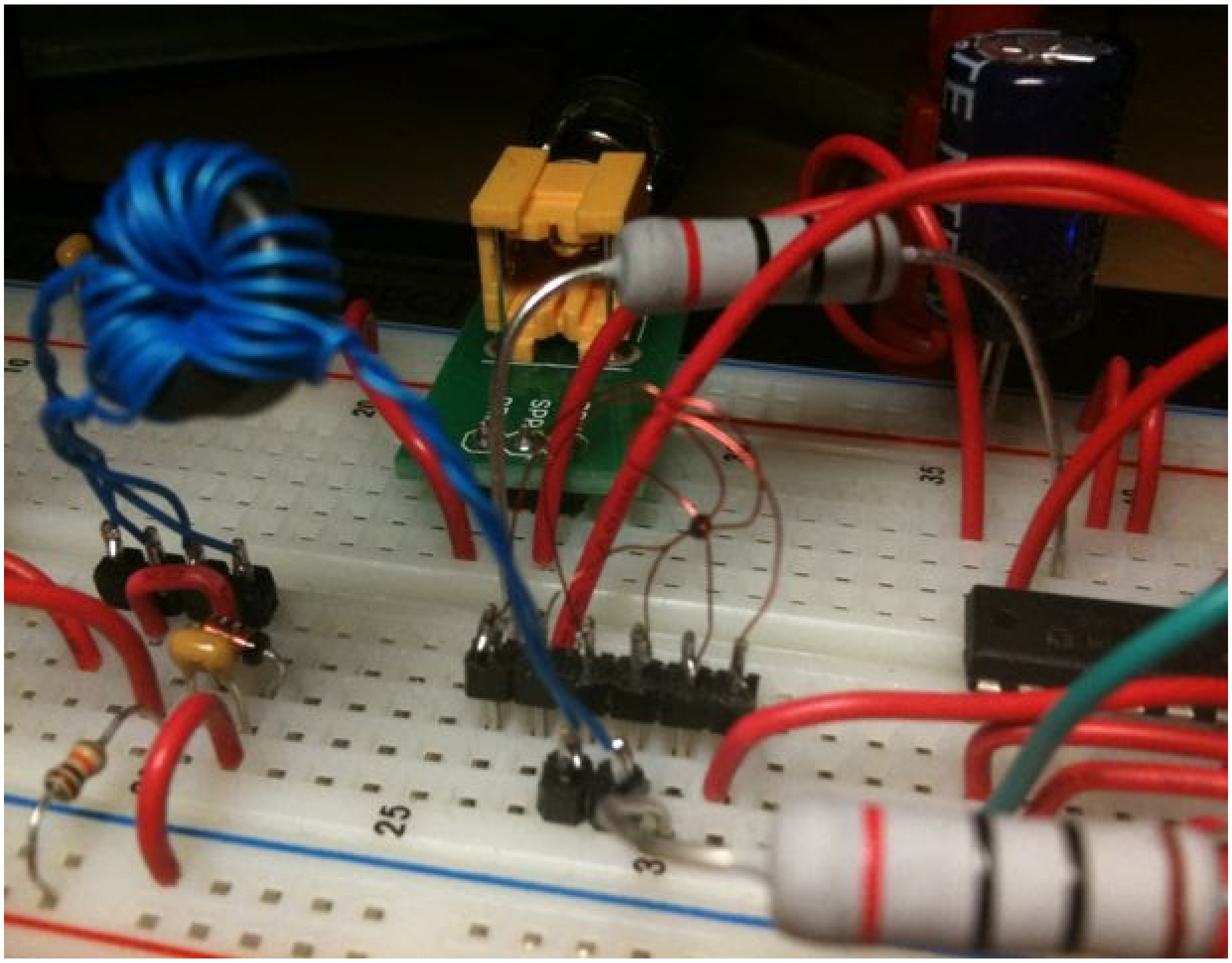
```
void sDelay (byte cnt) {
  for (byte ii = 0; ii < cnt; ii++) {
    __asm__ ("nop\n\t");    // NOP delays 62.5 ns
    __asm__ ("nop\n\t");
  }
}

byte pulse (byte lines) {
  PORTD = (lines + EN);
  __asm__ ("nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t");
  __asm__ ("nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t");
  __asm__ ("nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t");
   __asm__ ("nop\n\t");
  // sample sense amp output after ~1,200 ns delay
  byte smpl = PINB & 0x01;
  __asm__ ("nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t");  //
  __asm__ ("nop\n\tnop\n\tnop\n\tnop\n\tnop\n\tnop\n\t");  //
  PORTD = 0x00;
  sDelay(DELAY);
  return smpl;
}

void loop() {
  byte test = 0;
  Serial.print("RD: ");
  test = pulse(WR + SYNC);
  Serial.print(test, DEC);
  test = pulse(RD);
  Serial.print(test, DEC);
  test = pulse(RD);
  Serial.println(test, DEC);
  delay(1000);
}
```

You can test this circuit by connecting the sense line to the first channel of an oscilloscope, pin D2 to the second channel and D7 to the external trigger input. This should let you see a signal similar to the one shown at the top of this page. Or, you can add a simple sense amplifier, like the one shown below, and connect it to the Nano's D7 pin. In the code above you'll notice that the `pulse()` subroutine contains an embedded delay of about 1.2 microseconds followed by a read operation of the D8 line (the "PINB & 0x01" operation is a fast way to read D8 line.) This delay allows time for the core to flip before it samples the sense amp's output.



Sense Amplifier

The transformer has a split secondary that feeds two 2N3904 NPN transistors. This lets the circuit read either a positive, or negative kickback pulse (notice there's one of each in the scope trace at the top of the page.) I made the transformer by winding three separate coils, of 10 turns each, through a small, "soft" ferrite toroid, such as the ones used for antenna matching (I used a Philmore FT37-43 I purchased at my local Fry's.) One coil connects to the sense wire and the other two are connected with one wire from each coil in common to form a center tapped winding. You can see the transformer I made in the photo below wrapped in blue, wire wrap wire along with my single bit core (the small black dot with the tangle of fine, 34 gauge copper magnet wires passing through it near the center of the photo.) The sense amp circuit is adapted from this page, which also has a wealth of information on core memory systems.



Handmade transformer (blue wire) and One Bit Core (middle)

Once I put all the pieces together, I engaged the Arduino's Serial Monitor window (set the baud rate to 19200), and the console began to repeatedly display the string "RD: 001" which shows that the first two pulses flipped the core and the third pulse did not, which is how the circuit should work. Success!

All that for One Lousy Bit?

Yes, it's mostly useless in its current form, but it's a demonstration of the "technique" needed to build larger arrays. If you add just one more SN754410 you can directly address a 2x2 bit array. Use four SN754410s and you can address 16 bits in a 4x4 array, or step up to an 8x8, 64 bit array by using eight SN754410 chips. For each doubling of the X/Y drivers, you get 4 times the bits and you still need only a single sense amp.   In case you were wondering, the reason the sense amp is designed to handle both a positive and negative kickback pulse is because you may recall that, in a larger array, the sense winding will be threaded in a way to cancel out currents induced by the drives lines. Because of this, the cores will be in different orientations relative to the readout pulse and, as a consequence, the sense amp needs to be able read both polarities to detect a flip in all bits. Remember, it's the presence of the kickback pulse that indicates what state the core was in, not the polarity of the kickback pulse. The astute reader will also realize that the act of reading the state of a core also destroys the information in the core, so these systems would always write back the state read after reading it. This need to write after each read also led some computer engineers to leverage this fact in the design of instruction sets.

BTW, there's another clever technique from the core era that uses "steering" diodes to double the X/Y lines you can drive from a single side of the H-Bridge. Unfortunately, using an H-Bridge like the SN754410 makes this scheme difficult to implement because of the way the high and low side output transistors are connected together. For an example on how to use steering diodes see figure 3b on this page. And, once you've mastered this technique, you can learn how to stack multiple core "planes" into a 3 dimensional array that adds an "inhibit" wire (or uses the sense wire for the same purpose) so that one set of X/Y drivers can drive all the stacked planes. But, for now, I leave that as an exercise for the reader. I'm eventually planning on building a larger array, but I need to figure out a way to thread the cores that doesn't overly tax my limited dexterity.

If you enjoyed this page, let me know at "wayne dot holder at gmail dot com".