Magnetic core memory reborn

Ben North, Oliver Nash

May 9, 2011

Abstract

We outline the theory of magnetic core memory, and describe the design and fabrication of a core memory Arduino shield.

1 Introduction

1.1 What is magnetic core memory?

Core memory is an old memory technology.

Magnetic core memory was the most widely used form of digital computer memory from its birth in the early 1950s until the era of integrated-circuit memory began in the early 1970s. Aside from being extremely reliable, magnetic core memory is an appealing technology because it is based on a very simple idea.

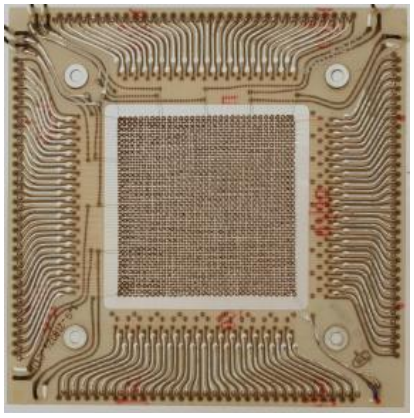
A *core*, a ring of magnetic material, stores one bit by the direction of its magnetisation.

A *magnetic core* is a ring of ferrite material. It can be permanently magnetised either clockwise or anti-clockwise about its axis just as a vertical bar magnet can be magnetised up or down. We can then turn a magnetic core into a bit of digital memory by letting these two magnetisation states correspond to 0 and 1.

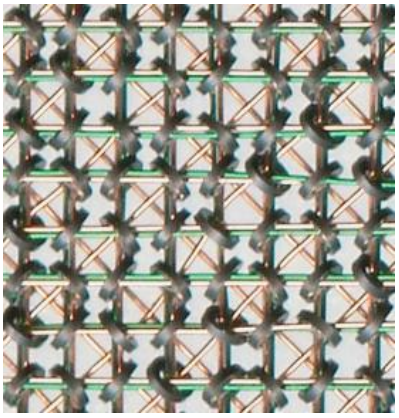
It provides non-volatile storage.

The core needs no power to retain its data. In other words, core memory is a form of non-volatile storage like modern hard disk drives, although in its day it fulfilled the 'high-speed' role of modern RAM.

With many such cores, large memory modules were made, such as this example from a CDC machine of the mid-1960s. The right-hand image shows a close-up of the cores themselves.



50 mm



2 mm

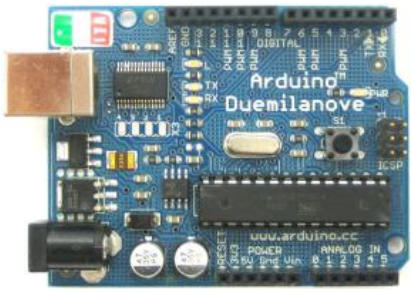
Cores were c.1 mm across, and ran at c.1 MHz. Core stores had up to 500,000 cores.

As the technology developed [1], the cores shrank from c.2 mm diameter in the early 1950s to c.0·4 mm by the early 1970s. Access speeds rose at the same time, from about 200 kHz to over 1 MHz, and core memory modules were manufactured with as many as over half a million cores. Furthermore, as recently as 2004, a magnetic core memory system was found still in service in a telephony control system.

It is still an interesting technology even today.

Magnetic core memory continues to capture the imaginations of modern enthusiasts [2, 3], and it is also the origin of the term *core dump*, to mean an on-disk image of the main memory of a process.

1.2 What is an Arduino?



The Arduino is a popular, easy-to-use, open design microprocessor board with IO capabilities.

The Arduino, whose Duemilanove version is shown above [5], is an open-source physical computing device. It has removed many of the hurdles for people wishing to explore embedded microprocessing. An Arduino is a small single-board computer based on an Atmel AVR microprocessor, with supporting components to handle USB communications and provide easy access to input/output pins. The developer programs the on-board microprocessor using an IDE running on a PC. It has proved very popular, with six-figure sales, and has been used in projects as diverse as autopilots for radio-controlled aeroplanes, and CNC sewing machines.

Add-on *shields* exist, providing specialist interfacing or control functions.

Several expansion modules exist, allowing the Arduino to perform a greater variety of tasks. These secondary modules, or *shields*, are circuit boards that plug into the Arduino's pin headers, and supply additional hardware to, for example, drive motors or servos, interface with wireless communication modules, or communicate over Ethernet.

1.3 A core-store shield

We set ourselves the challenge of creating a core memory Arduino shield.

We had developed an interest in the principles and practice of core memory, and the creation of a modern core memory module struck us as an appealing challenge. Eventually a pleasing idea suggested itself: we decided to build a magnetic core memory Arduino shield.

New-old-stock cores; modern components.

Our project was made possible thanks to the existence of a supply of surplus ferrite cores manufactured in the 1980s for the magnetic core memory systems of the time. We decided that we would use these old cores but permit ourselves to use modern components (transistors, integrated-circuit logic gates, etc.). This made our work enormously

easier than that of the original core memory inventors and manufacturers. Also, we knew that our end goal was possible.

We succeeded in building an extremely reliable 32-bit core memory shield for the Arduino. It can be used as storage for the Arduino, or alternatively, any modern computing device with a USB port can now read and write to magnetic core memory.

It worked.

2 Principles of core memory

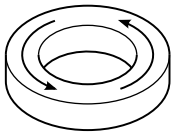
2.1 Magnetic hysteresis

A ferrimagnetic substance has two distinct permanently magnetised states. In our case, these are clockwise and anti-clockwise round the cores. The key to their use as computer memory is their behaviour when an external magnetic field is applied — they exhibit *hysteresis*.

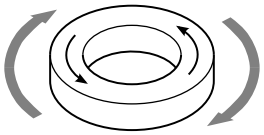
A weak external field has no effect

If we apply a weak external magnetic field around a core, and then remove it, there is no lasting effect on the core's magnetisation:

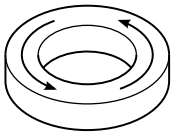
Start off with a core magnetised in the anti-clockwise direction:



Apply a weak clockwise external field. This reduces the strength of the anti-clockwise magnetism in the core:



But when the external field is removed, the magnetism in the core 'springs back' to its original state:

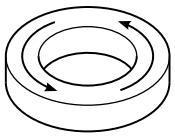


A weak anti-clockwise field similarly produces no lasting change.

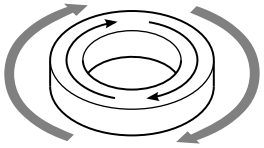
Strong external fields cause the magnetisation to *switch*

However, if we apply a sufficiently strong magnetic field in the opposite direction, the core will *switch* from one magnetic state to the other:

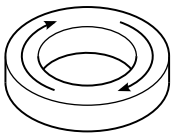
Start off with a core magnetised in the anti-clockwise direction:



Apply a strong clockwise external field. This switches the core's magnetism, and the core now has clockwise magnetism:



When this external field is removed, the magnetism in the core reduces slightly in strength, but remains clockwise:



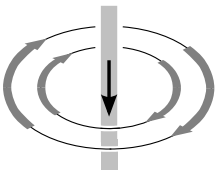
A strong field in the same direction as the core's existing magnetisation has no lasting effect — the core is *saturated* and can be magnetised no further in that direction.

2.2 The magnetic field of a current-carrying wire

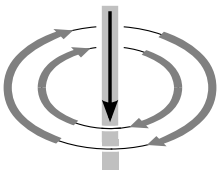
Current in a wire creates a magnetic field.

The next essential physical fact is that a DC current in a wire creates a magnetic field circulating about the wire, whose strength is proportional to the size of the current. If we reverse the direction of the current, the magnetic field circulates in the opposite direction:

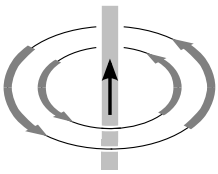
Small current down; weak clockwise field



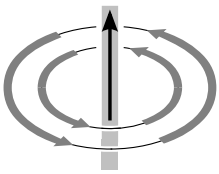
Large current down; strong clockwise field



Small current up; weak anti-c/w field



Large current up; strong anti-c/w field

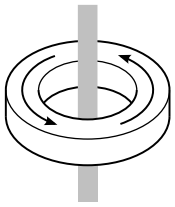


2.3 Writing to a one-bit core memory system

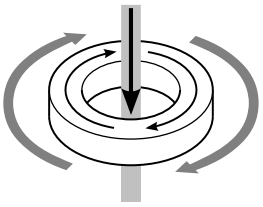
We can write a bit by driving appropriate currents.

We now know enough to write one bit of core memory. Having threaded a wire through a core, we can cause the core to become magnetised in the clockwise direction by passing a strong enough current in one direction through the wire, and we can cause the core to become magnetised in the anti-clockwise direction by passing a strong enough current in the opposite direction. These two operations correspond to writing 0 and 1 to the core. The magnetisation of the core, and so the bit stored, remains even after the current stops flowing.

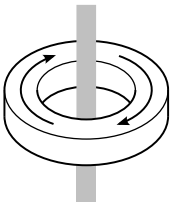
The core is magnetised anti-clockwise, and therefore holds the bit 1. No current flows through the wire.



A large current is supplied, generating a strong magnetic field. This switches the core's magnetisation to clockwise.



When the current is turned off, the core remains magnetised clockwise, therefore now holding 0.



2.4 Reading from a one-bit core memory system

But: how do we *read* the state of the core?

Core memory would probably have been a lot less popular had it been a write-only technology. To see how to read from the core, we need a further piece of physics: a changing magnetic field induces a voltage.

When a core switches, it induces a voltage.

Consider a core which is magnetised anti-clockwise. If we apply current which tries to increase this anti-clockwise magnetisation, it does not in fact change the magnetic field in the core much — it is already saturated. If, however, we apply enough current in the opposite direction, the core switches to be magnetised clockwise, as shown above. This process takes a small amount of time, around 1 μs for typical cores.

During that time, the magnetic field is undergoing a large change, and so induces a noticeable voltage.

A second wire senses the induced voltage.

The setup for reading the state of a core then is to have two wires passing through the core. We use one wire, the *drive line*, for driving current back and forth and so to set the state of the core, and we have circuitry connected to the second wire, the *sense line*, to measure the induced voltages.

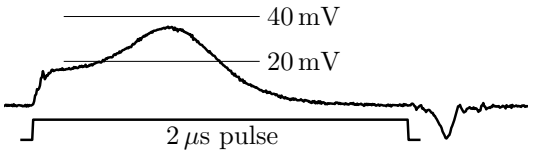
To read: write a 0; check if the core switched. If it did, it was a 1.

We start the process of reading the core by *writing* a 0 to it using the drive line. If we observe no significant induced voltage on the sense line then we learn the core was already in state 0. If, however, we observe a large induced voltage on the sense line then we learn that the core contained a 1 (and we then write 1 back to the core). The memory is said to operate with a *destructive read*.

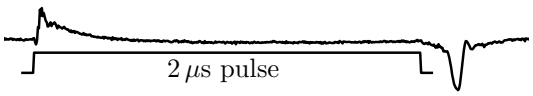
We measured the sense signals from our cores.

At this point it may help to make this more concrete by exhibiting some real data. Below we present the results of one of our experiments in which we measured the induced voltage on the sense line when writing 0 to a core which contained 1, and when writing 0 to a core which already contained 0.

If a core switches from one magnetisation direction to the other, the changing magnetic field induces a voltage that peaks at c.35 mV, staying above 20 mV for c.600 ns:



However, if the applied field is in the same direction as the core's existing magnetisation, no switching takes places, and we observe a negligible voltage:



We also see voltage 'spikes' caused by the drive line currents switching on and off. These *transformer-action* spikes are significant in magnitude but are short and temporally separated from the switching signal. The pulse shown is in fact the base drive of transistors, and the turn-on and turn-off delays are visible.

We could now build a one-bit read-write core memory system.

2.5 Arrays of cores and half-select currents

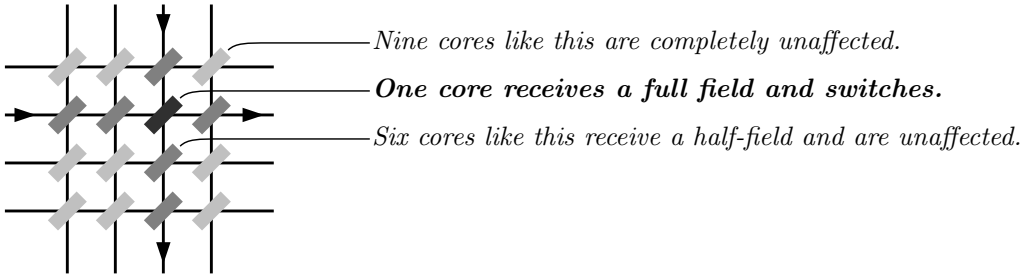
Imposing only half the field leaves the core alone.

However, we want to have more than one bit of memory without needing one drive line per core. We make use of a feature of the hysteresis behaviour of the core. We previously talked in terms of 'weak' or 'strong' external magnetic fields. Slightly more quantitatively: the core is *not* switched by a field which is half as strong as one which is just strong enough to switch the core. The field is proportional to the current in the drive line, so if we pass half the current required to just switch a core through the drive line, the core will not switch state.

A 2-D array gives efficient control of every core.

Now suppose we have nm cores and we arrange them in an $n \times m$ rectangular array such that each core has one of n horizontal and one of m vertical drive lines passing through it. With this setup we can

selectively set the (i, j)th core and no other by simultaneously driving the ith vertical drive line and jth horizontal drive line in the appropriate directions, each with half the current required for switching. For example, in a 4×4 array, with the cores viewed edge-on, we switch the core shown:



We only need $O(\sqrt{N})$ drive lines for an array of N cores.

In total, each core will have three wires threaded through it: two perpendicular drive lines, and a sense line. A single sense line can be threaded through all cores in an array, a point we return to below.

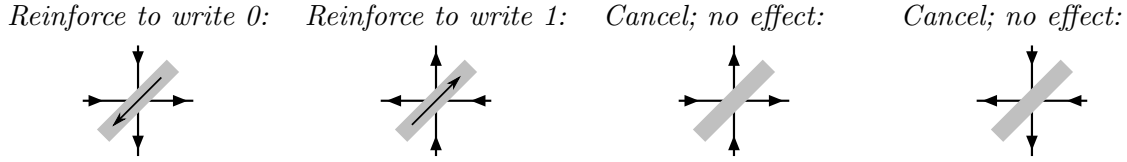
2.6 Exploiting anti-coincidence

One more trick doubles the number of cores we can address.

The picture we should now have in mind of a core memory module is a rectangular array of cores, with one vertical and one horizontal drive line passing through each core. There is a further drive-line economy, which cuts in half the number of drive lines in one dimension.

Two states out of four do nothing.

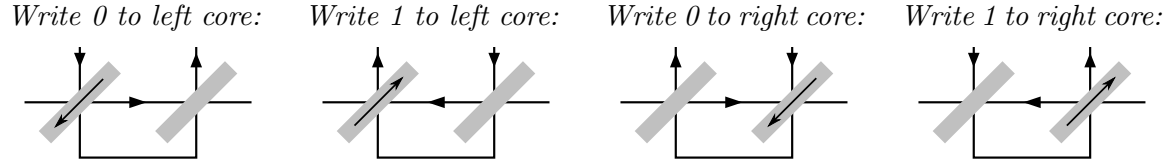
Consider a core with its two drive lines. There are then four combinations of current direction. Two combinations produce a reinforcing field round the core, but in the other two, the fields cancel round the core:



We say the currents or fields are *coincident* if they reinforce, and *anti-coincident* if they cancel.

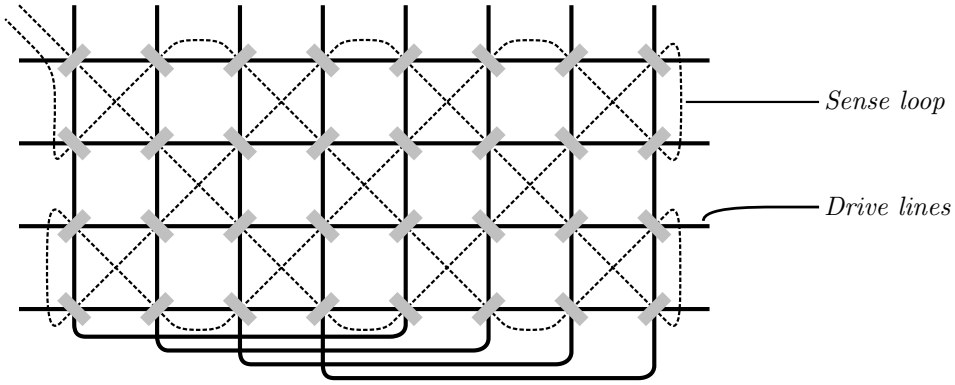
We can use those two states.

Now consider two cores arranged as below. We still have two drive lines, but we have turned one of the drive lines through two right angles so that it passes through both cores but in opposite directions. If we now consider the four possible simultaneous states of the drive lines, we find that all possible combinations are put to use:



One sense loop snakes through all cores.

To take the concrete example of the small core array we built, we show a 8×4 array. We also now illustrate the single sense line which runs through all cores. The alternating alignment of the cores makes the threading of the sense loop easier, and also reduces the transformer-action spikes.



The array of cores can be thought of as two halves, with each core in the left half having an anti-coincident partner in the right half sharing the same pair of drive lines.

3 Our implementation

To implement these principles, we have to solve a few fairly distinct problems. We must be able to drive current through the drive lines; decide which drive lines need to have current driven through them; and sense when an induced 'switching signal' occurs.

3.1 A current-driving circuit

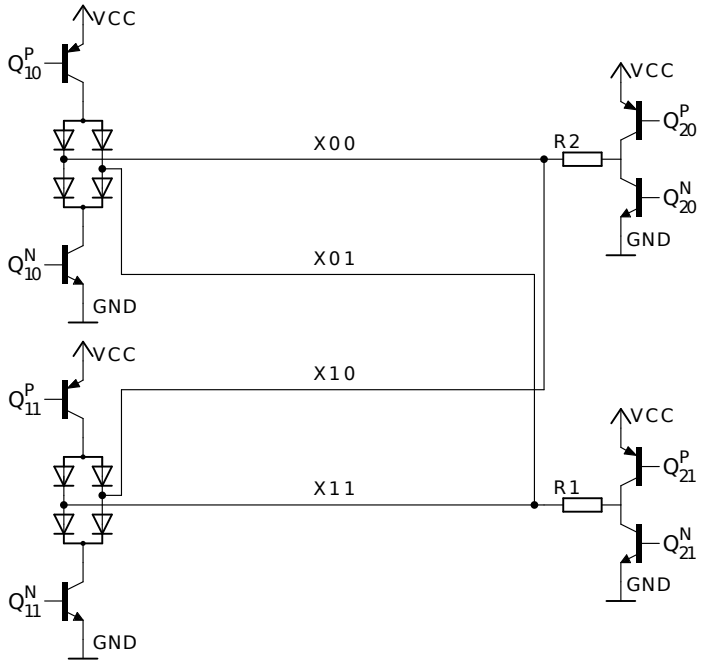
The drive circuit is two copies (X and Y) of one problem: driving one of four lines in either direction.

As noted above, we organise the 32-bit module as a rectangular array with 4 rows and 8 columns in which the columns are paired as above using anti-coincidence. As a result, we have 4 vertical and 4 horizontal drive lines to control. Our current-driving problem is thus two instances of the same problem: we need to be able to select one of four lines, and to specify a direction in which to drive current along that line.

Drive the four lines as 'two by two'.

Our current sources and sinks are transistors, and we could accomplish our goal by connecting each end of each drive line to the collectors of both a PNP and an NPN transistor (i.e., by using four H-bridge circuits). This would work, but would be inefficient in terms of the number of transistors used. By introducing a few diodes, we can use a design with half as many transistors.

By turning on the correct transistors in the following circuit, we can send current in either direction through any of the drive lines. The resistors R_1 and R_2 control how much current flows. We discuss their value below.



Calibrating the half-select current

Experiments determine how much current we should use.

As we have already discussed, it is important that we drive the right amount of current. It must be low enough that the magnetic field produced by the current in one drive line is not enough to switch a core, but high enough that the combined field from two coincident currents does produce switching. We determined the range of possible values of current for our cores empirically and established that any value in the range 250–340 mA works. We set the current by choosing appropriate values for the resistors in the drive circuit above.

3.2 Addressing

Which transistors should we turn on?

In the circuit above, we want to know which transistors to turn on to drive current left or right in one of the four lines. We must be able to pulse this current, and so we need to be able to stop all drive currents. Let the *enable* Boolean variable be E. Let the variable D_X give the direction, with $D_X = 0$ being leftwards. In order to drive the line $X_{A_1A_2}$ in the direction D_X (as long as E is asserted), the base of transistor Q^P_{11}, for example, must be taken high or low according to $Q^P_{11} = \overline{E \wedge D_X \wedge A_1}$. In words, Q^P_{11} must be on (i.e., its base must be pulled low) exactly when we are enabled ($E = 1$) and we wish to drive current rightwards ($D_X = 1$) through either X_{10} or X_{11} (those two lines have $A_1 = 1$). The formulae for all eight transistors are:

$$\begin{array}{ll} Q^P_{10} = \overline{E \wedge D_X \wedge \overline{A_1}} & Q^P_{20} = \overline{E \wedge \overline{D_X} \wedge \overline{A_2}} \\ Q^N_{10} = E \wedge \overline{D_X} \wedge \overline{A_1} & Q^N_{20} = E \wedge D_X \wedge \overline{A_2} \\ Q^P_{11} = \overline{E \wedge D_X \wedge A_1} & Q^P_{21} = \overline{E \wedge \overline{D_X} \wedge A_2} \\ Q^N_{11} = E \wedge \overline{D_X} \wedge A_1 & Q^N_{21} = E \wedge D_X \wedge A_2 \end{array}$$

Combining the X and Y drive addressing

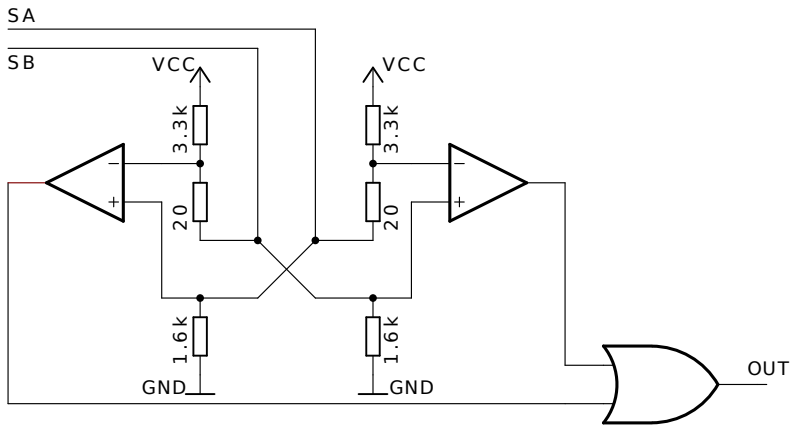
Translating from 'currents in lines' to 'address and data'.

Recall that the current-driving circuitry consists of two instances of this driving circuit, horizontal and vertical. We thus have bits A_1, A_2, and D_X for the first instance and A_3, A_4, and D_Y, say, for the second. We can generate a switching field in either direction for any of our 32 cores by choosing values for these six bits, and then controlling transistor bases accordingly. Furthermore, looking back at section 2.6, we see that the tuple (A_1, A_2, A_3, A_4) determines a pair of anti-coincident partner cores and that it is the value of $A_0 := D_X \oplus D_Y$ that distinguishes between these two partner cores. Therefore we reparameterise our six control bits in terms of the five-bit address $A_4A_3A_2A_1A_0$ and the single data bit $D := D_X$. We then have $D_Y = D_X \oplus A_0$. The 32 cores are now numbered, since each core has a unique five-bit address $A_4A_3A_2A_1A_0$. The layout of these addresses in the 8×4 array may appear a little haphazard, but this does not matter.

3.3 Sensing

How to detect the bulging switching signal?

With the current-driving and addressing circuitry in place, it remains only to describe how to detect the presence or absence of a core-switching signal on the sense line. We chose the comparator-based circuit described in [1] (and apparently of unknown German origin):



The sense line, described above, is threaded once through each of the 32 cores, and its ends are connected to the lines marked SA and SB.

Need to detect positive and negative sense bulges.

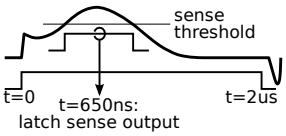
The obvious two-fold symmetry of this circuit reflects the possibility that a sense pulse from a switching core can bias SA positively or negatively relative to SB depending on whether a core is switching from a clockwise to an anti-clockwise magnetisation state relative to the sense line or vice-versa.

With an input of at least $\pm$20 mV, one comparator's output goes high.

The resistors ensure that, when no voltage is present across SA/SB, each comparator's inverting input is 20 mV above its non-inverting, and so both outputs are low. If a positive sense pulse occurs, causing SA to rise more than 20 mV above SB, the left comparator will detect this and output 'high'. Conversely for a negative sense pulse, the right

comparator will fire. We OR the two to detect pulses of either polarity. The threshold of 20 mV was chosen by experiment.

Capture sense output at the right moment.

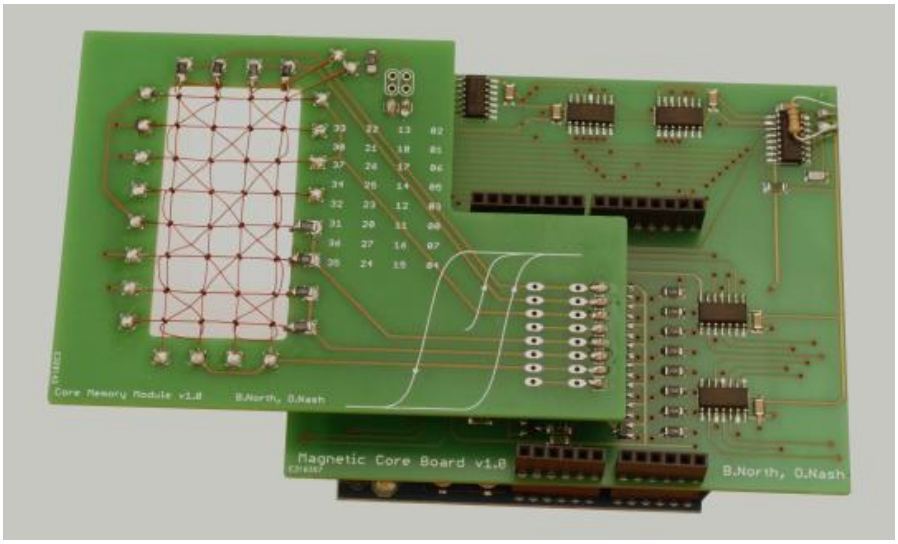


The final detail of the sense circuitry is that we used a timed latch to catch and store the output of the OR gate at the appropriate moment. This ensures that we are not affected by any high comparator outputs caused by the transformer-action spikes. The timer delay was chosen by experiment as 650 ns, although any value 500–900 ns worked.

4 Finished product

4.1 Complete circuit

Combining these pieces, we arrive at the final circuit, which we give in appendix A. The design splits the shield into a driver board and a plug-in core board, as shown in this photograph of the two boards plugged into the Arduino:



4.2 Fabrication

Our design files are available.

We finally achieved our goal of building a core memory shield by having our schematics fabricated as PCBs. Owing to some late design changes, we then had to patch the boards slightly; the circuits described here include the changes. Eagle files for these schematics and PCBs as well as the Gerber files are available in the online version of this document [6]. Also contained in the Eagle files are the part numbers for the various components (transistors, logic gates, comparators, etc.) that we used.

4.3 API and testing

How to write to and read from the cores from Arduino code.

Once the core board, driver board and Arduino have been appropriately mated, it is easy to use the Arduino to read from and write to the core memory array. To send a write pulse to a core, the Arduino writes the data bit to digital pin 8, writes the address of the core in question to

digital pins 3–7 and then asserts ENABLE (digital pin 2) for 2 μs. Once this is done, reading digital pin 9 reveals whether the core changed state. Cycle time is such that a 32-bit operation takes 0·5 ms, although preliminary experiments suggest that by being more aggressive on the timings we could bring this down to 0·3 ms.

A simple API and serial-based CLI is available.

These operations are presented as the functions `write_bit` and `read_bit`. Loops turn these into `write_word` and `read_word`, to act on the whole core board as a single 32-bit unsigned integer. A further natural operation is a *bit exchange*. This, in one operation, writes a new data bit to a core, and returns the old value, which can be deduced as (latched-sense-output) XOR (new-data). The source-code for this API, with a command-line interface presented on the serial port, is available for download [6].

The core memory shield performed error-free for 10 bln operations, running for 100 hrs.

We used this interface to extensively test the core memory shield. After running continuously for over one hundred hours, the shield had performed over ten billion successful bit-exchange operations without a single error.

5 Final words

We hope that our work here might encourage others to take an interest in both the theory and the practice of core memory. For further reading, excellent descriptions are given by Hilpert [1] and by Jones [4].

Building a reliable 32-bit core memory shield for the Arduino proved to be a natural stopping point for our work, but there is certainly more that could be done. Some ideas for future work that appealed to us:

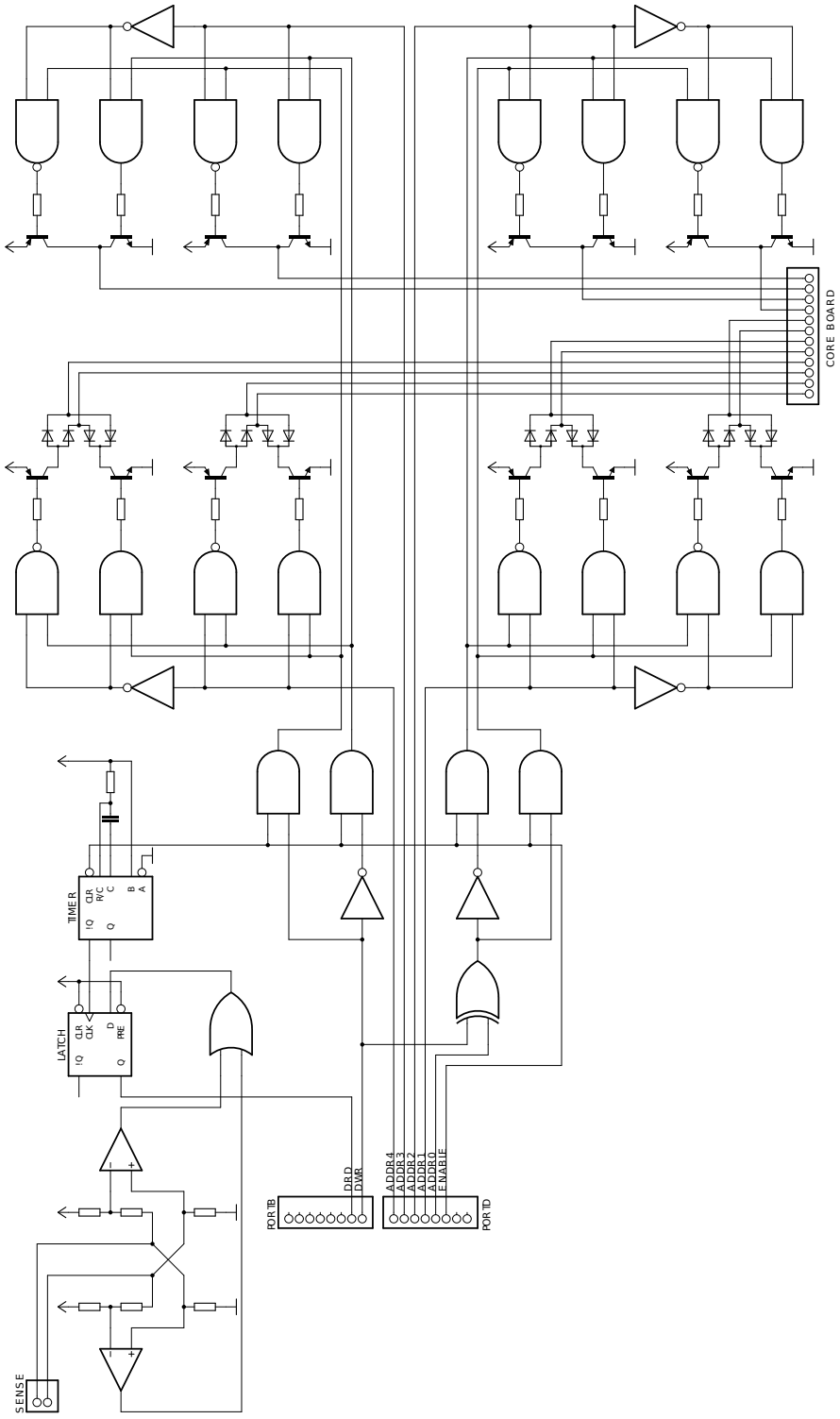
- Many core memory arrays from the heyday of core memory still exist and indeed are easy to find for sale (e.g., on eBay). It might be a fun challenge to build a driver module to write to and read from these old arrays. Perhaps with care and luck one could even recover some 50-year-old data?

- A completely different approach to core memory, with a non-destructive read cycle, was invented by Eiichi Goto in the mid-1950s, and was incorporated in the Japanese Parametron Computer 1 (see [7] for a description). It would be very interesting to re-implement this significantly different core memory technology.

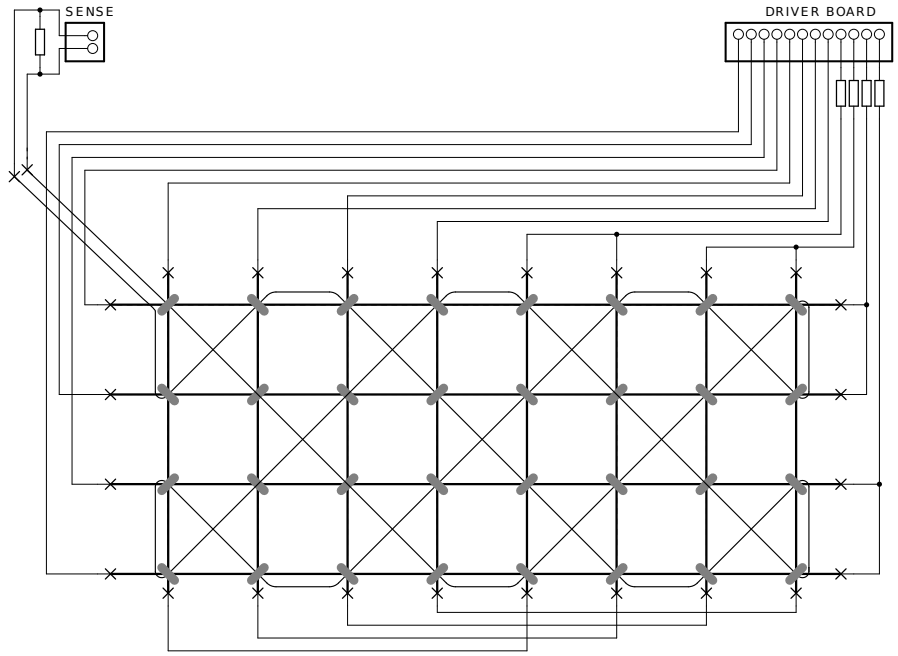
- In practice, a group of core memory modules of the type described here, known as *planes*, were combined into a core store. All planes could be accessed simultaneously by introducing an *inhibit* line [4]. Constructing an 8-bit wide store would be a worthwhile challenge.

A Circuits

Firstly, the schematic for the driver board:



Secondly, the core board's schematic:



The terminating resistor across the sense loop provides a path for the current caused by the induced voltage round the loop.

References

[1] B. Hilpert. Magnetic core memory systems. `http://www.cs.ubc.ca/~hilpert/e/coremem/index.html`.

[2] W. Holder. One bit ferrite core memory. `http://sites.google.com/site/wayneholder/one-bit-ferrite-core-memory`.

[3] A. Holme. Magnetic core memory. `http://www.holmea.demon.co.uk/Core/Flipper.htm`.

[4] J.R. Jones. Coincident current ferrite core memories. *Byte*, 11:6–16, July 1976.

[5] D. Mellis. Photograph of Arduino Duemilanove, CC-BY-2·0. `http://www.flickr.com/photos/mellis/4784965878/`.

[6] B. North and O. Nash. Magnetic core memory reborn. `http://www.corememoryshield.com/`.

[7] E. Wada. The Parametron Computer PC-1 and its initial input routine. In P. Rojas and U. Hashagen, editors, *The first computers: History and architecture*, pages 435–452. MIT Press, 2000.