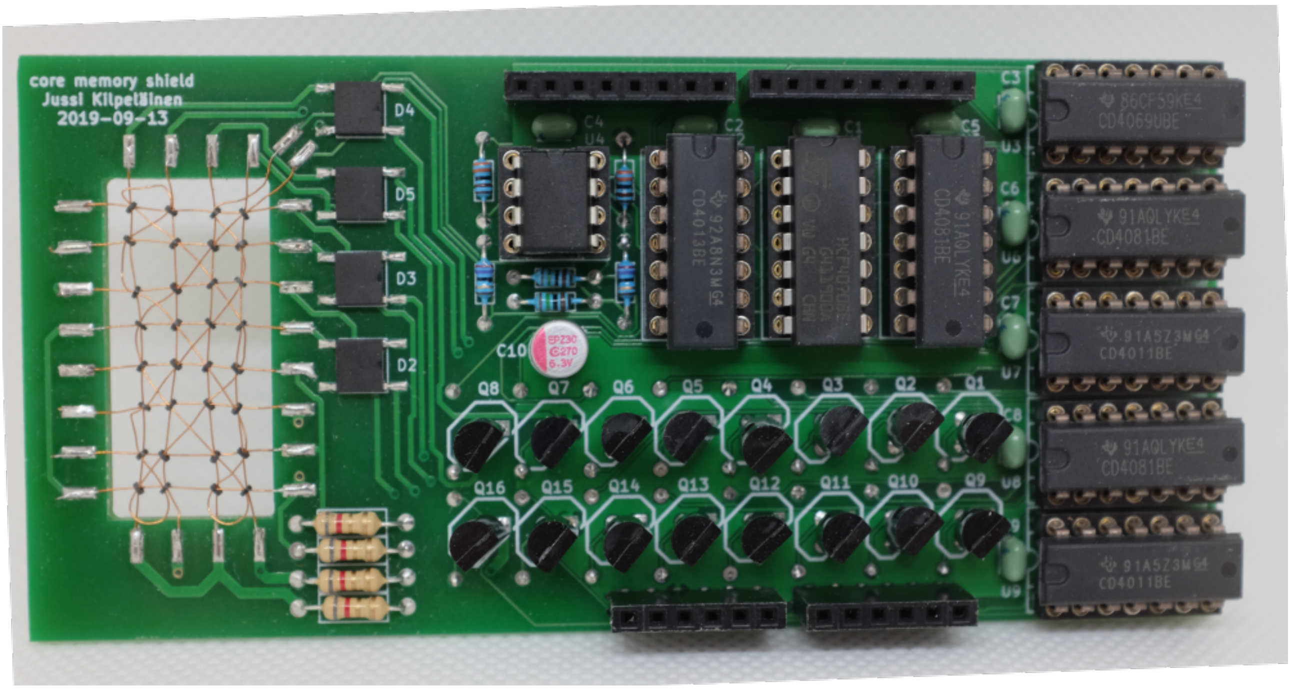
Core memory shield



Foreword

In the first part of the documentation the **principle** of ferrite core memories is explained. The second part of this document covers the **design** of the kit with some helpful notes for hackers and modders. In the third part I walk through the tricky parts of **putting together** this kit, and after that I give a few ideas how to **use** the memory. The last pages of this document contain all the appendices like the **schematic, circuit board artwork and bill of materials.**

The theory section is not very long; for more information please see

- B. Hilpert: Magnetic Core Memory Systems. http://www.cs.ubc.ca/~hilpert/e/coremem/
- B. North, O. Nash: Magnetic core memory reborn. http://www.corememoryshield.com/

Table of Contents

1 Theory

1.1 Physics

To get an understanding of the underlying physical phenomenon, let's look at a single ferrite core which has a wire going through it. A current in the wire (*I*) creates a magnetic field *B* like can be seen in Illustration 1.1.

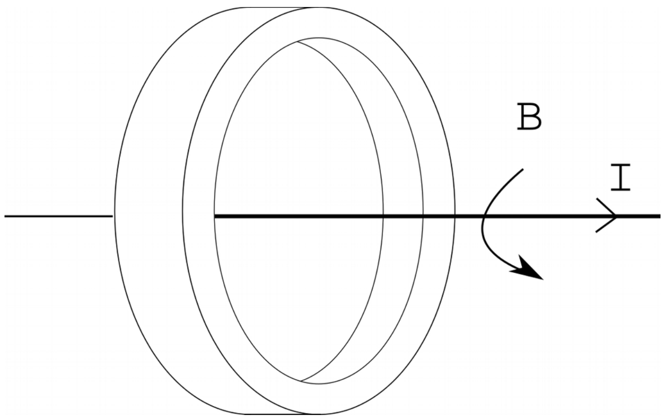


Illustration 1.1: Ferrite core and a wire.

The magnetic field in the ferrite core can be changed by modifying the amount and direction of the current in the wire. Let's take a look at Illustration 1.2. If the core is in state 1 and we increase the current, the magnetic field inside the core will change polarity to state 0. It will stay in that state even after we have stopped supplying the current. The only way to get it back to state 1 is to apply enough current in the opposite direction. (Naming the states 0 and 1 is just a matter of agreement.)

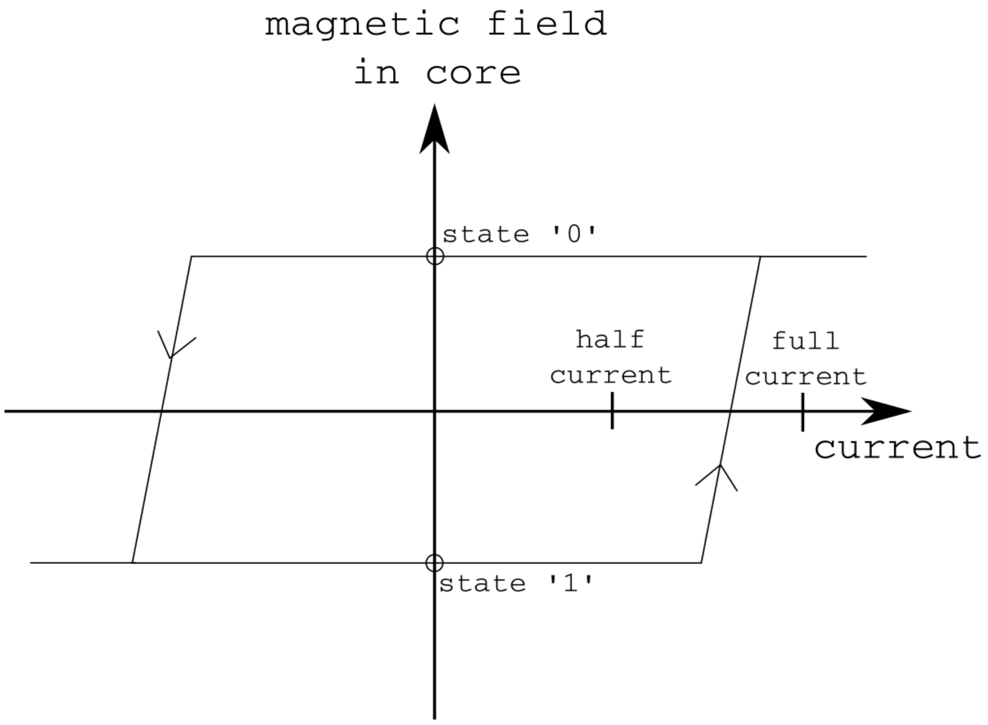


Illustration 1.2: Ferrite core hysteresis.

Now we know how the memory can be written, but how about reading? If we insert another wire (called sense wire) inside the core, a voltage pulse can be seen every time the magnetic field inside the core changes as can be seen in Illustration 1.3. There is some noise on the sense lines coming from the addressing currents but they can be distinguished quite easily.

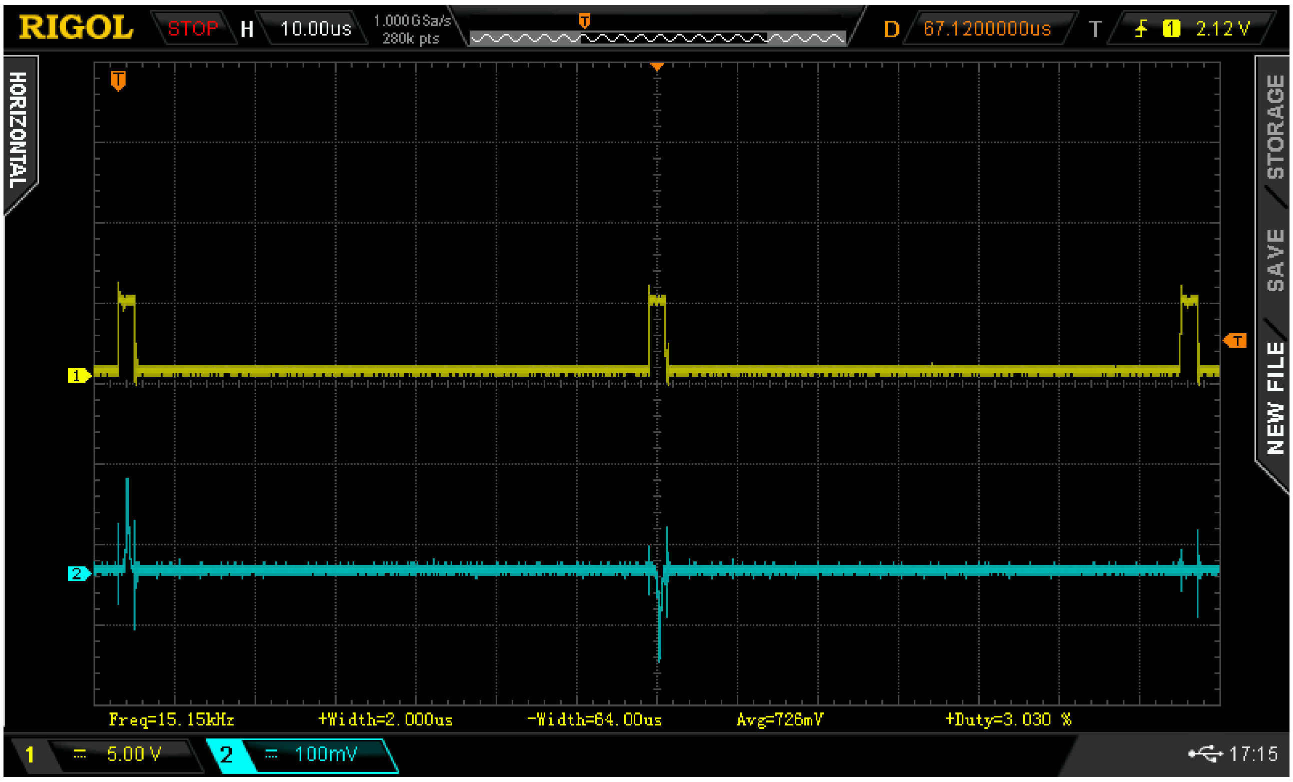


Illustration 1.3: Oscilloscope screenshot from three memory accesses. The first two changed the memory state and the last one didn't. Memory enable in yellow and sense line in blue.

If the state of the memory changed as a result of trying to figure out the state of the memory, the changed bit needs to be written back.

1.2 Optimizing the number of addressing lines

To keep the number of lines low the cores are usually laid out in a grid (Illustration 1.4).

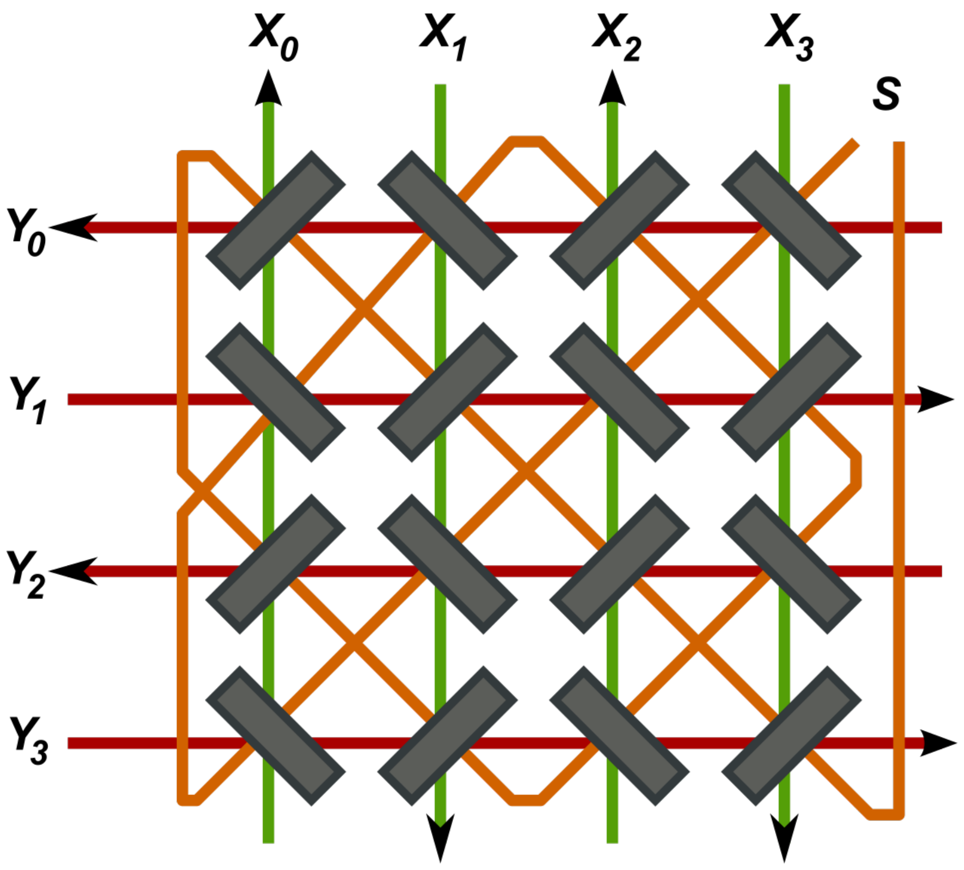


Illustration 1.4: Diagram of a 4×4 plane of magnetic core memory in an X/Y line coincident-current setup. X and Y are drive lines, S is sense. Arrows indicate the direction of current for writing. By Tetromino. Source: *https://commons.wikimedia.org/wiki/File:Coincident-current_magnetic_core.svg*. Creative Commons Attribution 3.0 Unported. Modified by Jussi Kilpeläinen.

If you wanted to change the state of the core in the top left corner, you would apply a ”half-current” to wires X0 and Y0. One ”half-current” by itself is not enough to flip the magnetic field in the core, but two is and therefore only one of the cores changes state. Notice how the cores next to each other are never in the same orientation; this makes looping the sense line (orange) easier and reduces interference from neighbouring cores.

There is one more trick to use in order to reduce the amount of drivers needed for the address lines – if we arrange the cores like shown in Illustration 1.5, number of lines is halved in one dimension. This is the exact topology of cores that is used in this kit, and you can use the figure as a guide when weaving your own.

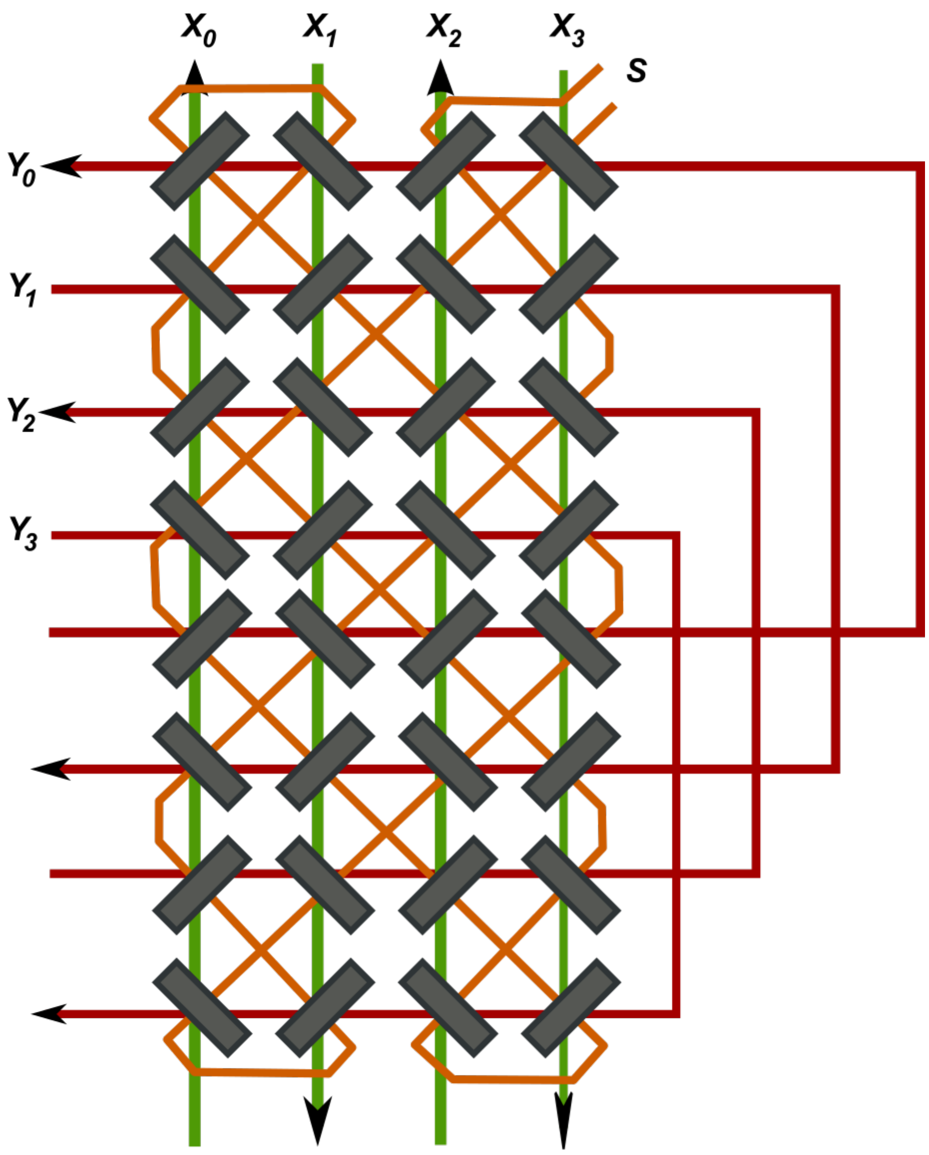


Illustration 1.5: Exploiting anti-coincidence. Modified further from Illustration 1.4.

To explain this how this anti-coindicence works, let's try to address the top left core again like shown in Illustration 1.6. For the top left core the X and Y currents add up. However, for the core four rows below that one the currents cancel each other (they go through the core in opposite directions), meaning that the resulting magnetic field is zero.

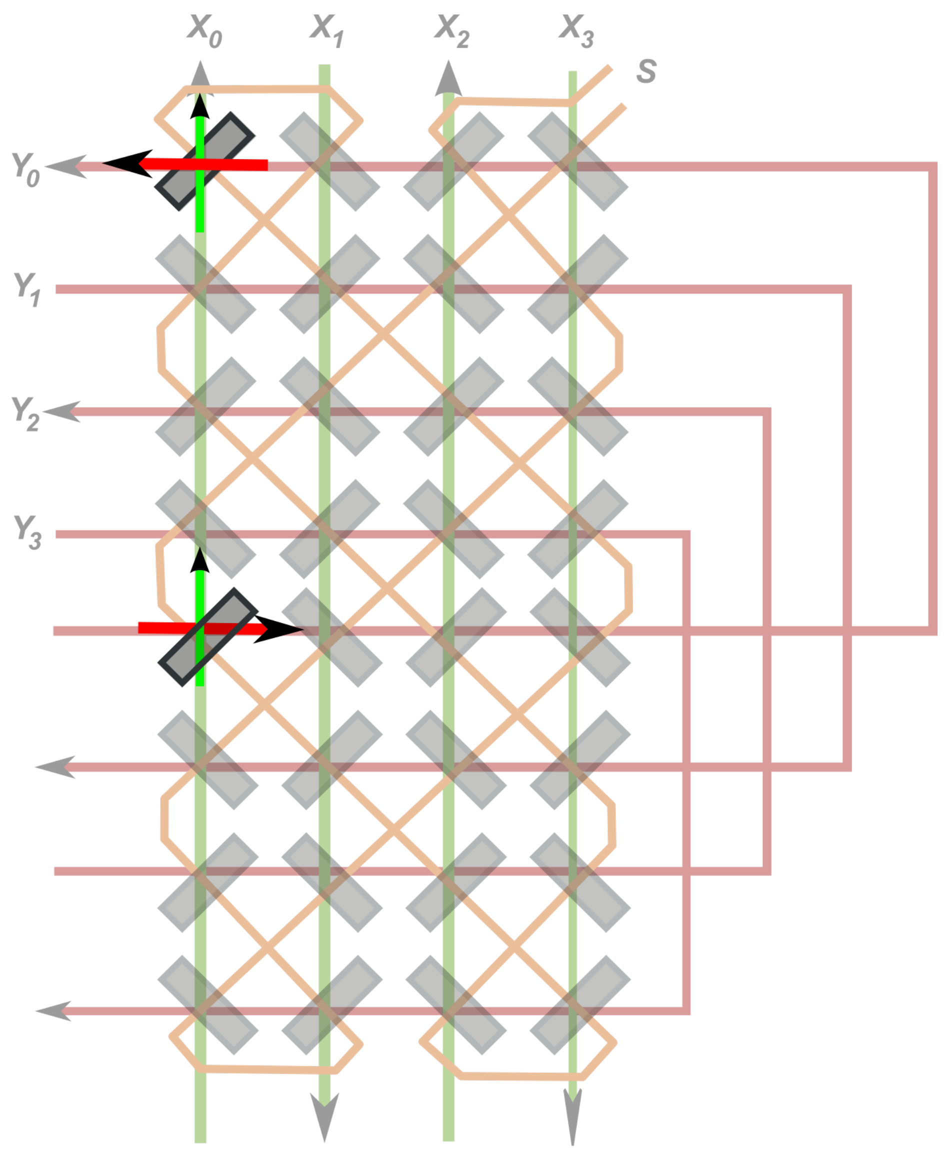


Illustration 1.6: Addressing a core when exploiting anti-coincidence. Modified further from Illustration 1.4.

2 Design of the kit

The kit is designed to be mounted on top of an Arduino board, but any 5 V microcontroller can be used. The processor drives the address, enable and write lines and monitors the read line. On the shield a set of 4000 series CMOS logic chips[1] decodes the address and turns on the drive transistors. Both the logic and the ferrite cores run on 5 volts[2].

If the voltage difference between the sense lines is over about 25 mV, a pulse will be sent and stored in a flip-flop.

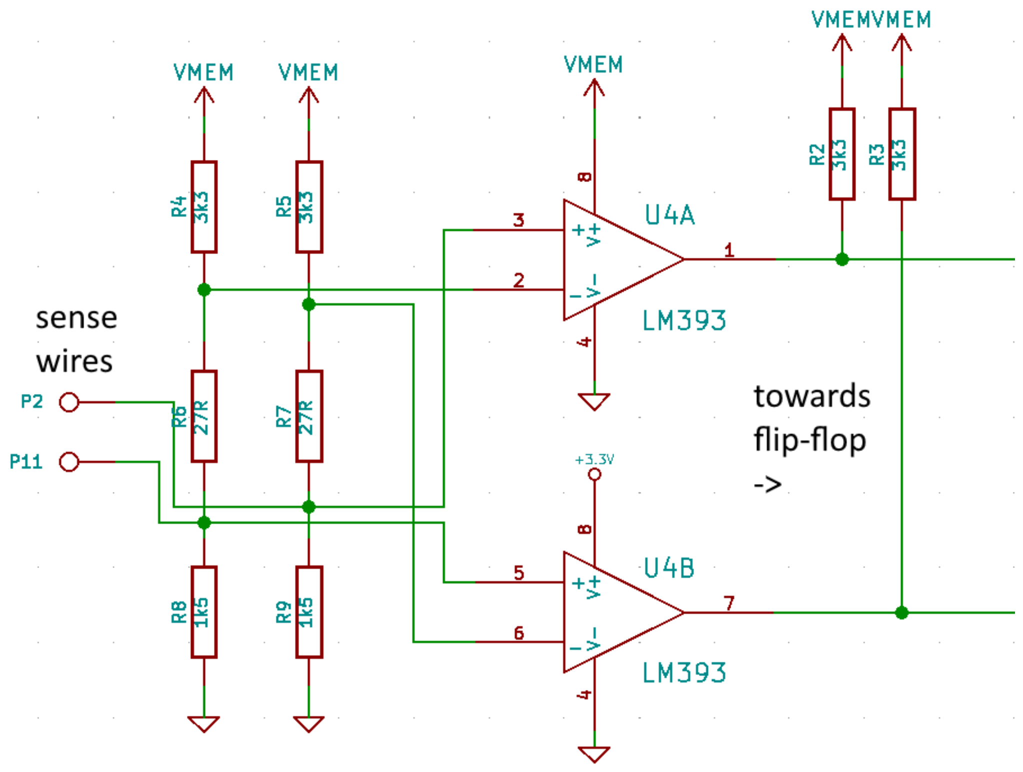


Illustration 2.1: Sense line amplifier.

If this read pulse goes high while following a rising edge of ENABLE it is a sure sign that the ferrite core has changed state. This is implemented by clearing the value of RD when ENABLE goes high, and connecting the read pulse to the SET pin of the flip-flop.

1 Previous versions of this kit (PCB date 2016 or 2017) used a CPLD for the logic; the logic is identical in both versions. The full changelog is available in section 6.5.

2 The CPLD version of this kit used 3.3 volts.

3 Assembly instructions

The kit uses almost exclusively through hole parts with the exception of the bridge rectifiers. The bill of materials (BOM) in the appendices lists the components in the suggested order of assembly – start with the components that have the lowest profile like resistors. Notice that there are some resistors to be mounted on the bottom side of the PCB (figure 3.1).

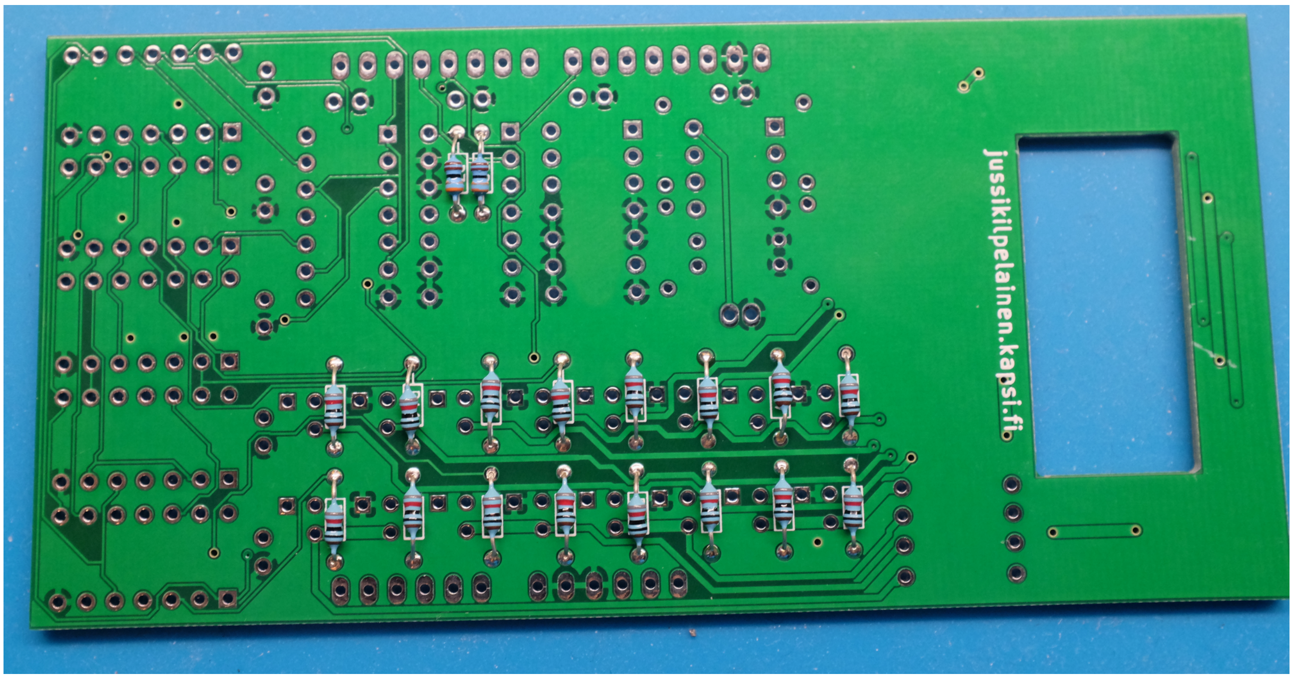


Illustration 3.1: 3.3k and 47k resistors are mounted on the bottom side.

3.1 Cores

The first tricky part are the ferrite cores themselves; Start with the vertical wires - put the wire through eight beads and solder the wire on the PCB as shown in the figure below. You probably want to use tweezers for handling the cores. The enamel on the wire burns off when heated.

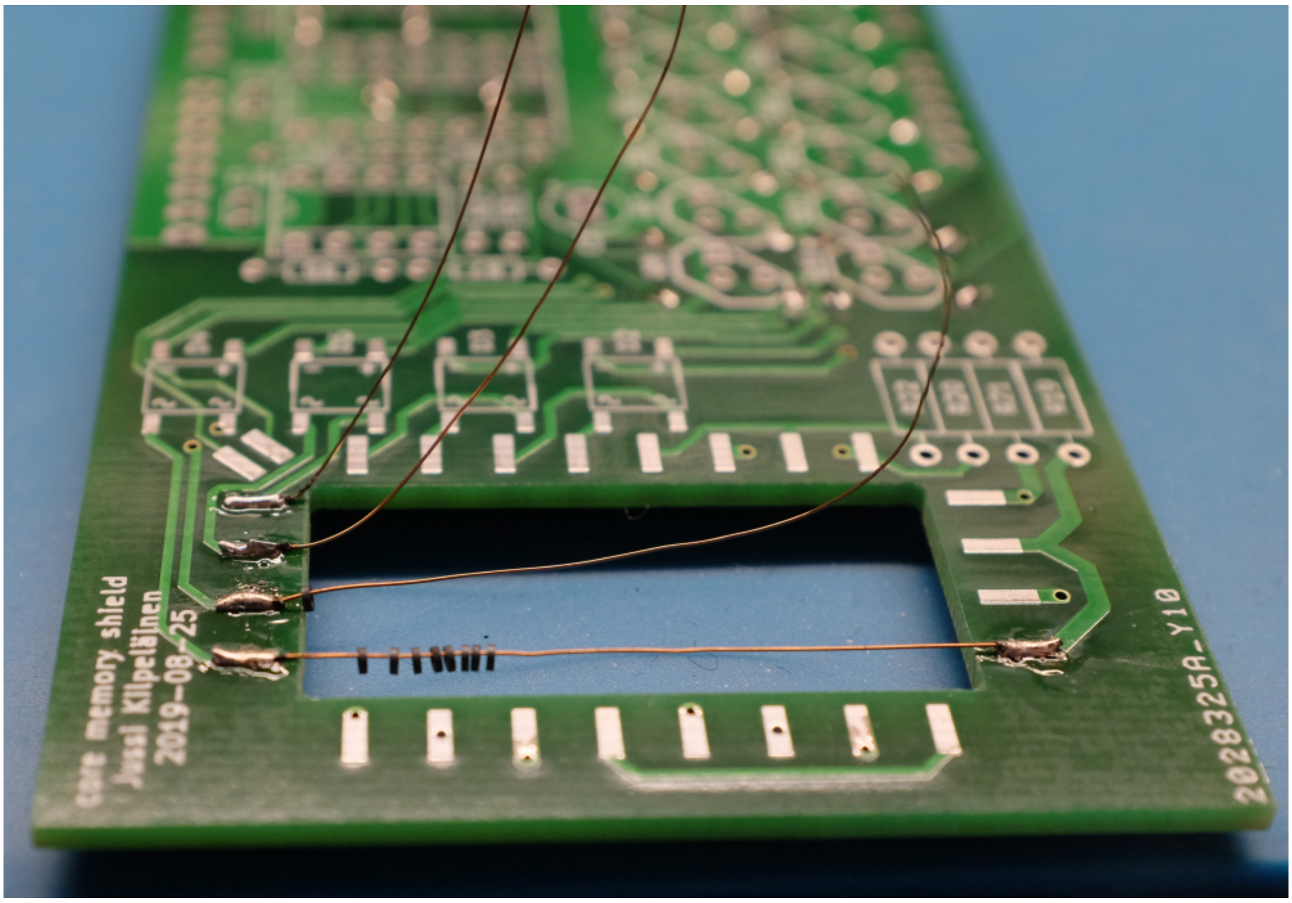


Illustration 3.2: Threading the cores through the wires.

Then you can move on to the horizontal wires. **Make sure that the cores next to each other are not in the same orientation!** See figures 1.5 and 3.3 before starting, and orient the cores in the exact same way as in those figures.

After the horizontal wires there is still the sense wire to be woven into the grid. After that you can congratulate yourself; the hardest part of this kit is done!

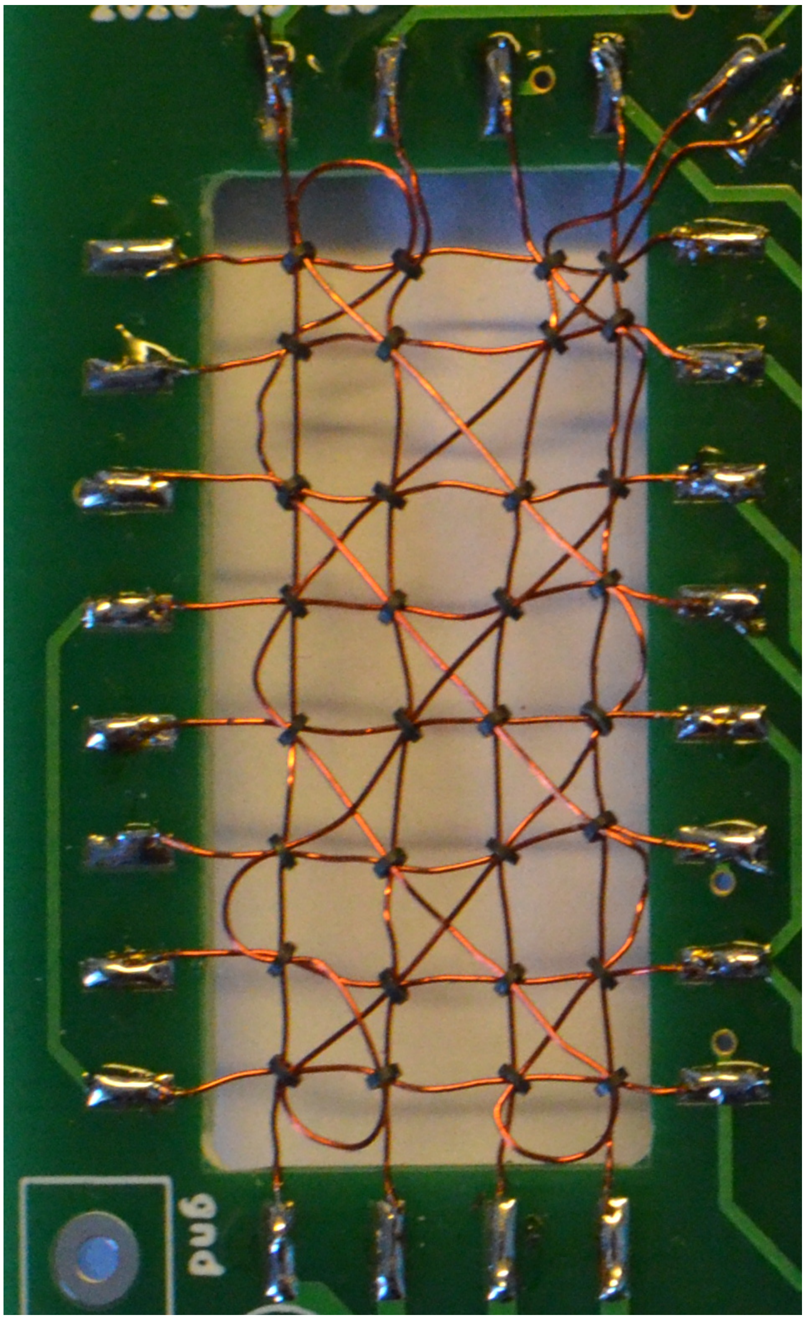


Illustration 3.3: Close-up of the core. The connector in the bottom left is not present in new versions of the kit.

3.2 Drive transistors

The drive transistors require some special attention – **make sure not to mix up the PNP and NPN transistors**! The transistors with an even reference (Q2, Q4, …) are NPN (BC517), odd references (Q1, Q3, …) are PNP (BC516).

You may also feel that there is not enough room to solder with the bottom side resistors getting on the way. It can be managed by repeating this process:

1. solder what you can
2. clip the leads
3. go back to step 1.

There shouldn't be anything special about the rest of the components - when it's all said and done it should be looking like this.

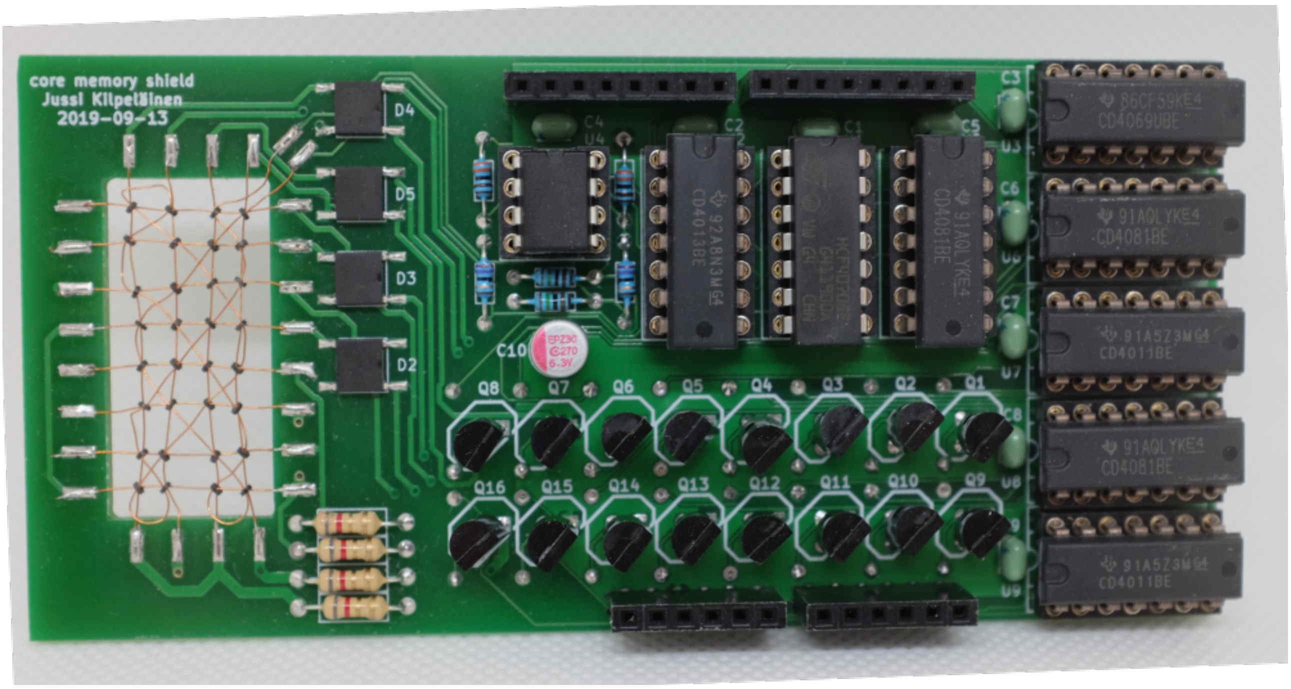
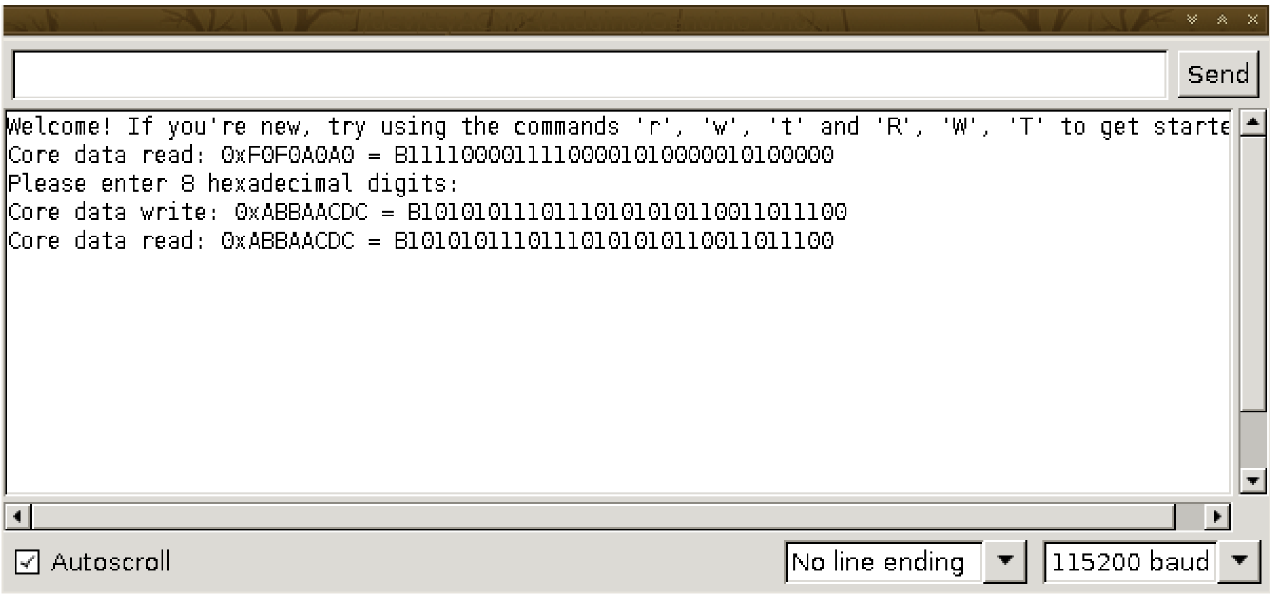


Illustration 3.4: Completed core memory shield.

4 Usage instructions

The best way to try your new core memory out is to use the nice software written by Ben North and Oliver Nash (http://www.corememoryshield.com). I have modified the program slightly to work with newer versions of Arduino and to adapt the timings to the specific components used in this kit; please see my website for details http://jussikilpelainen.kapsi.fi/?p=213.



```
Welcome! If you're new, try using the commands 'r', 'w', 't' and 'R', 'W', 'T' to get starte
Core data read: 0xF0F0A0A0 = B11110000111100001010000010100000
Please enter 8 hexadecimal digits:
Core data write: 0xABBAACDC = B10101011101110101010110011011100
Core data read: 0xABBAACDC = B10101011101110101010110011011100
```

☑ Autoscroll   No line ending   115200 baud

Illustration 4.1: Screen capture from Arduino serial monitor.

To store a word, type ”W” followed by the data in hexadecimal format. To read, type ”R”. One fun experiment is to write something, put a strong magnet near the cores and see if the data gets corrupted. ”T” initiates a reliability test for the memory.

Note that it is important to choose ”No line ending” from the drop-down menu near the bottom of the window. Otherwise the terminal sends newline characters every time you press enter, and the program will complain about those with a message ”Ignoring unknown command: A” or ”Ignoring unknown command: D”.

5 Troubleshooting

5.1 Unable to read the written data back

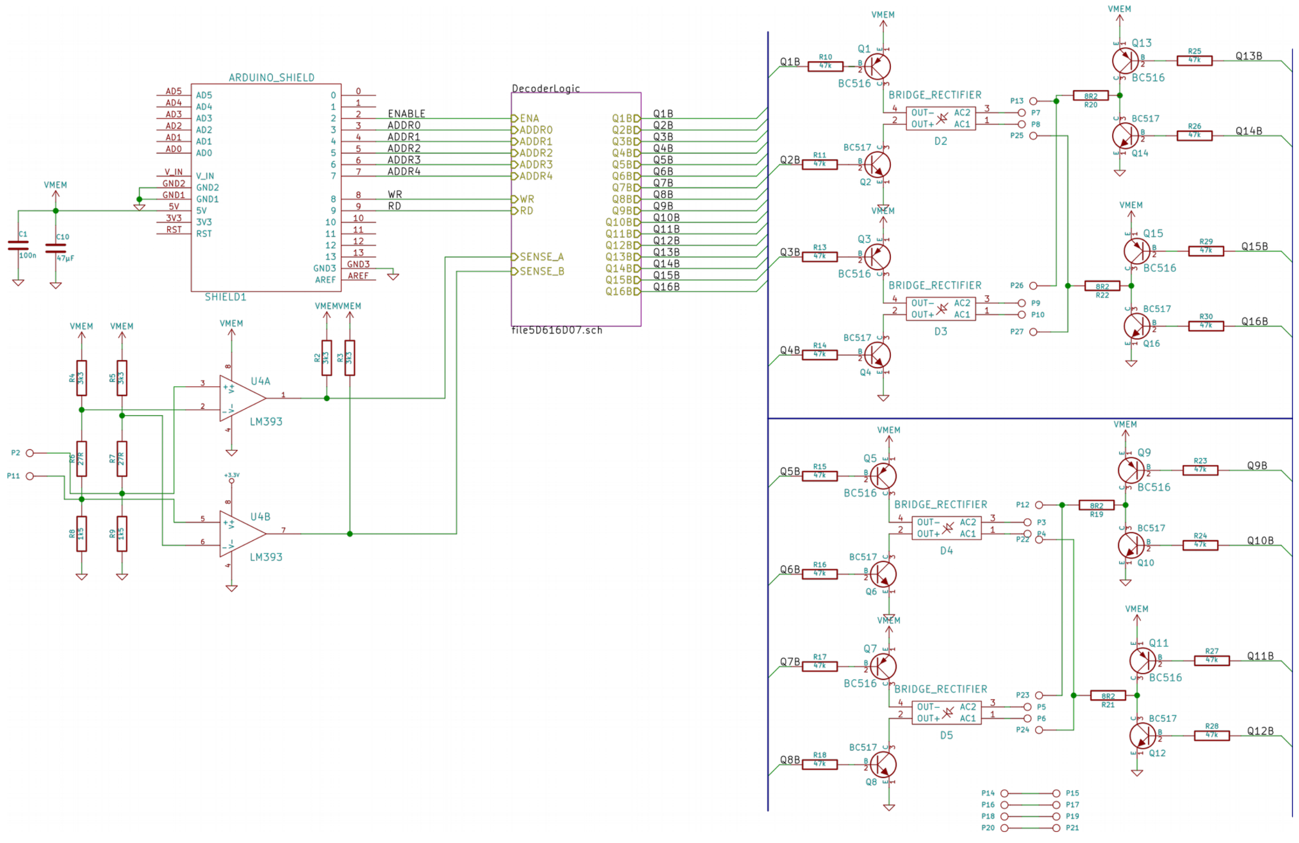
By far the most common problem are poor solder joints of the magnet wires. Check continuity of the pads that are connected by the magnet wires.

5.2 Writing / reading single bits work but not full 32-bit words

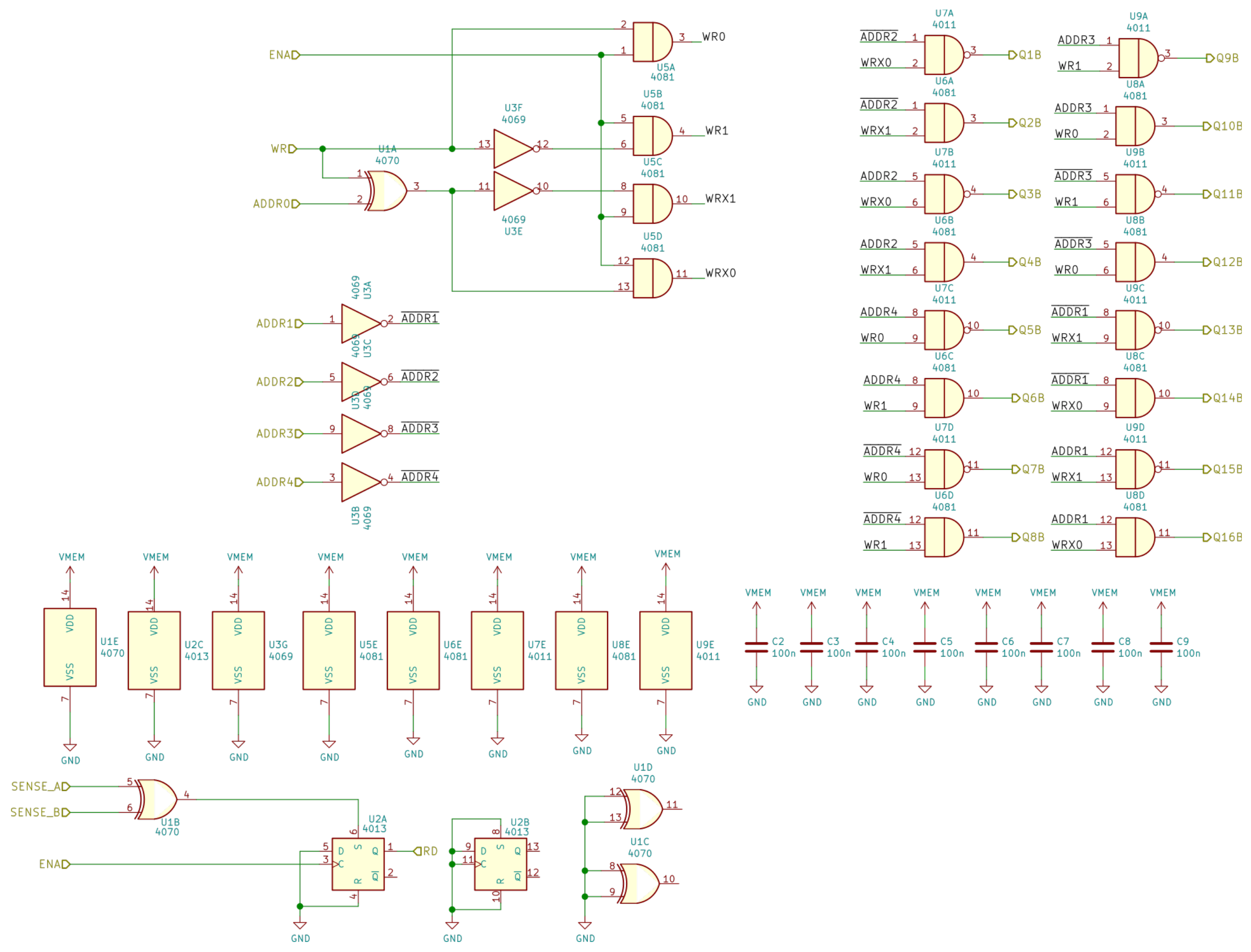
This might be caused by a sagging 5V line – remember that the currents required to bit the flips are hundreds of milliamperes. Try hooking up an external DC power supply to your Arduino.

6 Appendices

6.1 Schematic

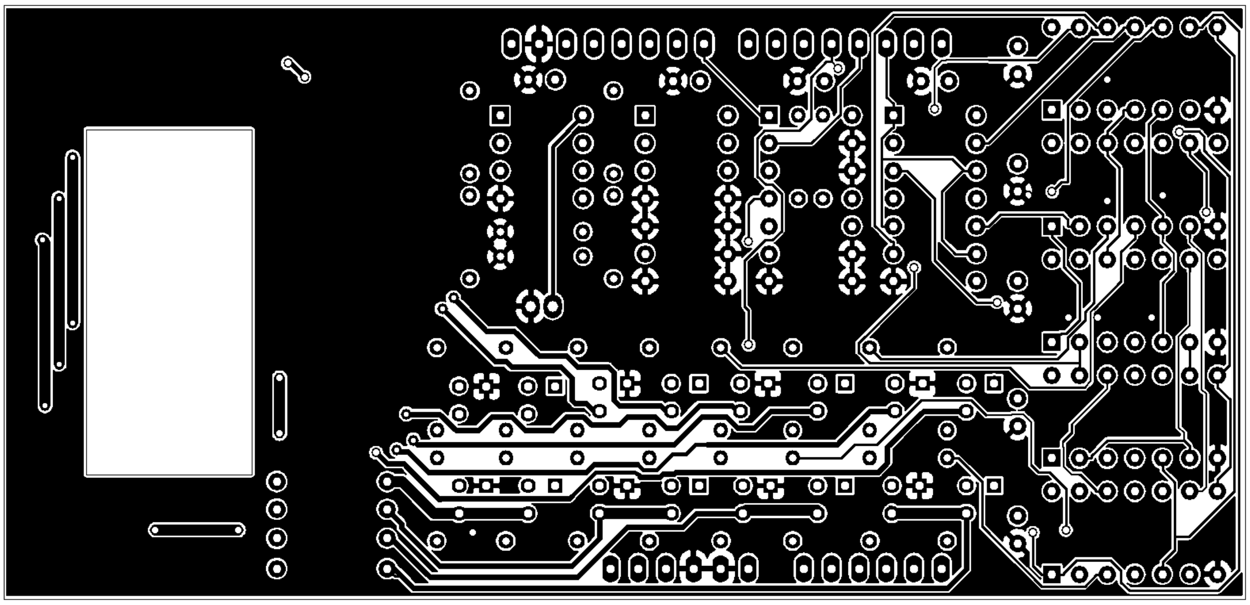


See the decoder logic on the following page.

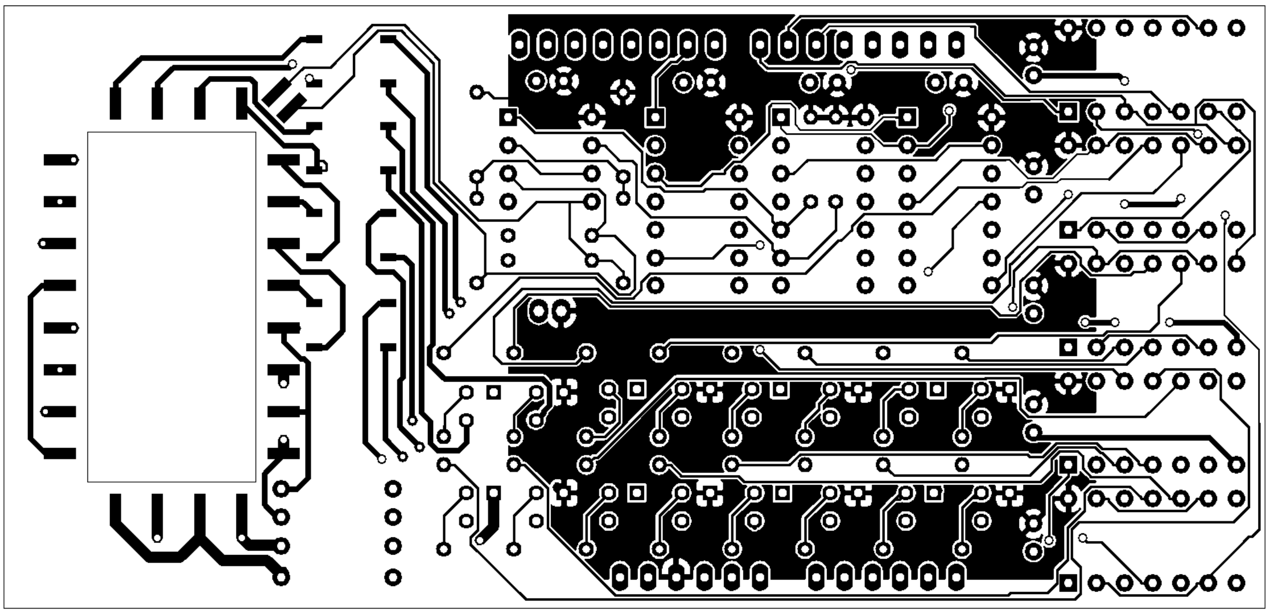

ENA
WR
ADDR0
U1A 4070
U3F 4069
U3E 4069
U5A 4081
U5B 4081
U5C 4081
U5D 4081
WR0
WR1
WRX1
WRX0
ADDR1
ADDR2
ADDR3
ADDR4
U3A 4069
U3C 4069
U3D 4069
U3B 4069
Q1B Q2B Q3B Q4B Q5B Q6B Q7B Q8B
Q9B Q10B Q11B Q12B Q13B Q14B Q15B Q16B
U6A U6B U6C U6D U7A U7B U7C U7D
U8A U8B U8C U8D U9A U9B U9C U9D
VMEM
GND
VDD
VSS
U1E 4070
U2C 4013
U3G 4069
U5E 4081
U6E 4081
U7E 4011
U8E 4081
U9E 4011
C2 100n C3 100n C4 100n C5 100n C6 100n C7 100n C8 100n C9 100n
SENSE_A
SENSE_B
U1B 4070
U2A 4013
U2B 4013
RD
U1D 4070
U1C 4070

6.2 PCB artwork

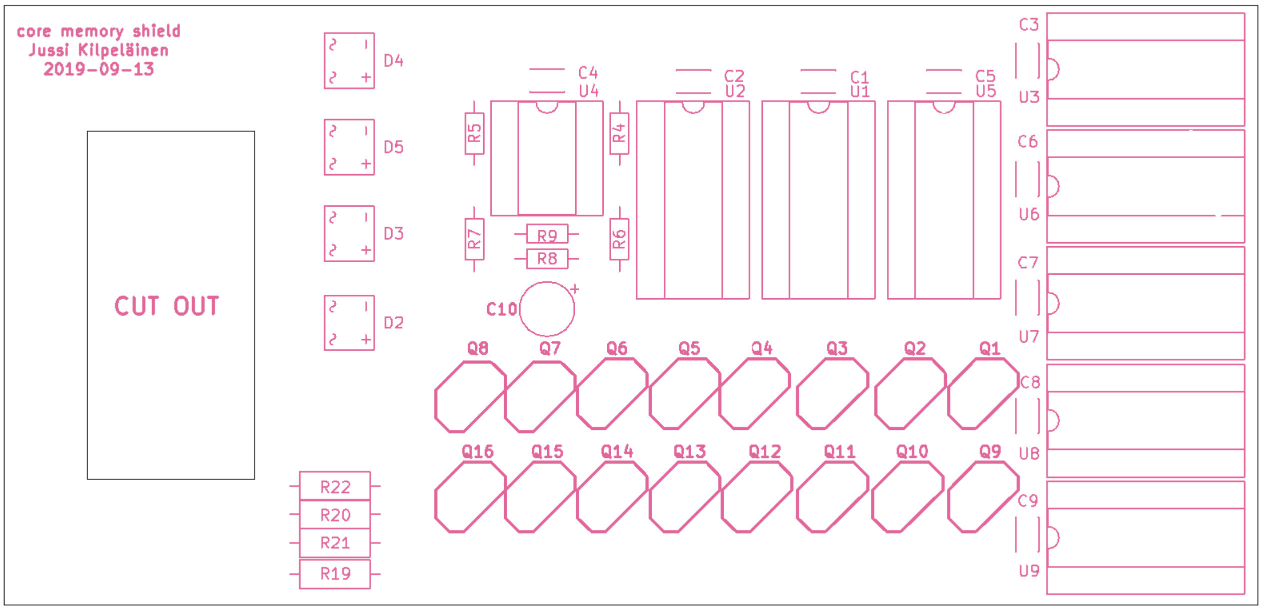
6.2.1 Bottom copper



6.2.2 Top copper



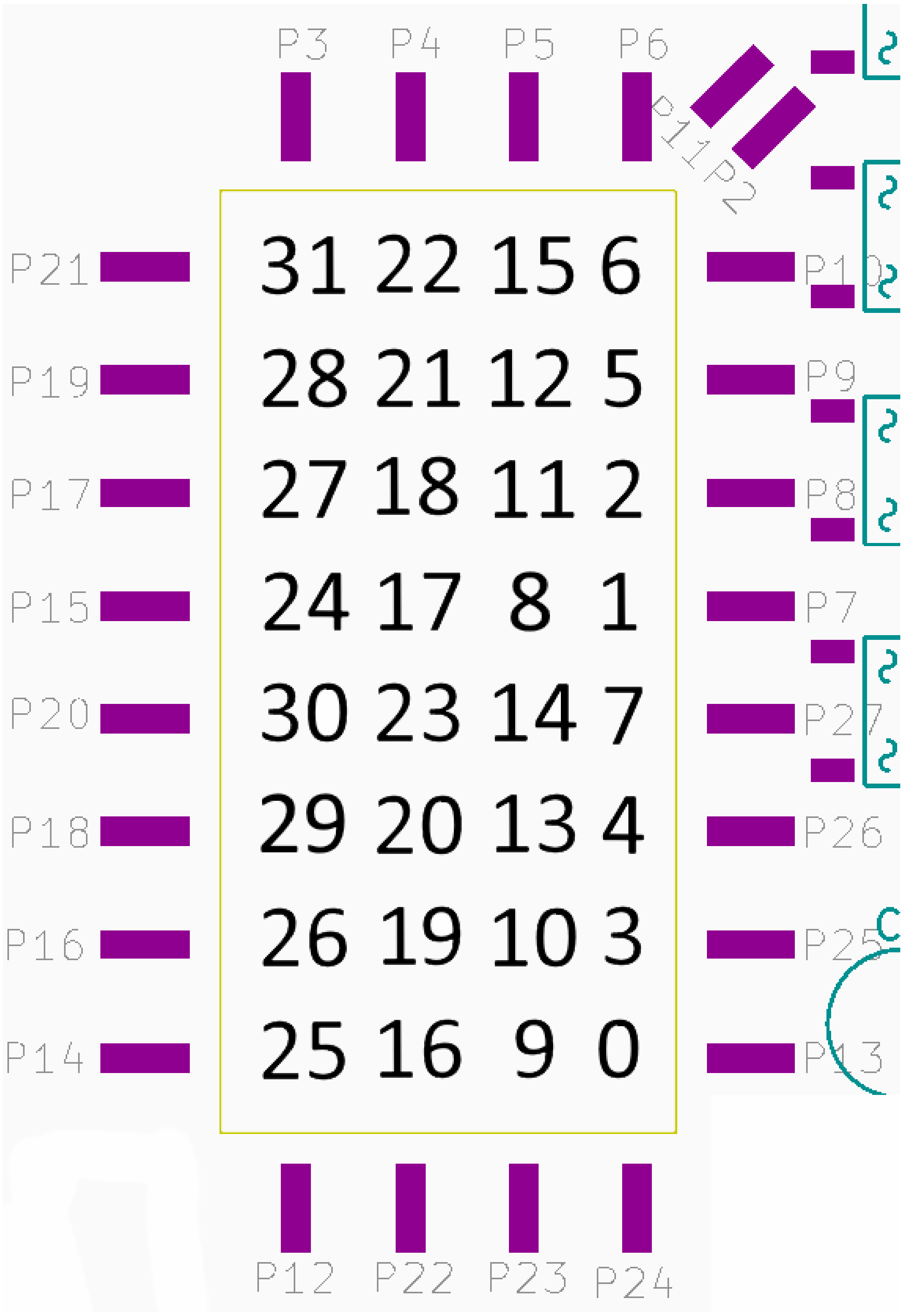
6.2.3 Top silkscreen



6.3 Bill of materials

Qty	Ref	Value	notes
1	PCB	PCB	
16	R23, R10, R11, R14, R24, R27, R28, R25, R16, R17, R18, R26, R30, R13, R29, R15,	47k	
4	R5, R4, R2, R3,	3k3	
2	R7, R6,	27R	
2	R9, R8,	1k5	
4	R19, R21, R22, R20,	8R2	
9	C1, C2, C4, C5, C6, C7, C8, C9, C3,	100n	
1	CORE1	0.15-0.2 mm magnet wire	
32	CORE2	ferrite cores	
4	D2, D3, D4, D5,	ABS10	surface mount
8	Q1, Q9, Q3, Q11, Q5, Q13, Q7, Q15,	BC516	
8	Q2, Q4, Q10, Q12, Q6, Q14, Q16, Q8,	BC517	
1	U1,	4070	+ sockets
1	U2,	4013	+ sockets
1	U3,	4069	+ sockets
1	U4,	LM393	+ sockets
3	U6, U8, U5,	4081	+ sockets
2	U7, U9,	4011	+ sockets
1	C10,	270 µF	
2	ARDUINO1	6-pin header	
2	ARDUINO2	8-pin header	

6.4 Core-to-bit mapping



6.5 Design changelog

This section lists how the different PCB layouts differ; from time to time there might have been small changes in the components provided due to availability.

6.5.1 PCB date 2016-03-28

First version.

6.5.2 PCB date 2017-01-12

- Changed to surface mount bridge rectifiers to avoid shorting the USB connector on Arduino boards with full-size B connector
- Removed the low-pass filter capacitors from the board.

6.5.3 PCB date 2019-09-13

- Replaced the CPLD with 4000 series CMOS logic.
  - VCC changed to 5V throughout → no need for external supply anymore
  - Basically all passive components needed to be re-calculated.
  - Darlington transistors used for the drive stage due to smaller drive current available.
  - Slightly larger circuit board.
  - Sockets now provided for all integrated circuits.