Oscilloscope Playtime

Oscilloscopes are to view voltages changes too fast for the human eye, the measurement device. This means it is $voltage\ over\ t$ ... $v(t)$.

But: they for no particular reason, a remnant of time long gone, have a XY-Mode, means we are in control of $value$ **and** $time$.

distribution fee: 1,5$ or 1,5€.

Datum: $Date: 2026-02-02 13:57:25 +0100 (Mo, 02. Feb 2026) $

Contents

1 Simple signals from arduino

We can only use $digitalWrite$ because getting the $DC-value$ from an $analogWrite$ requires lowpass filtering which makes the changes to slow ... we could watch a moving point at best.

To remedy this we have to use a $DigitaltoAnalogConverter$, in short DAC.

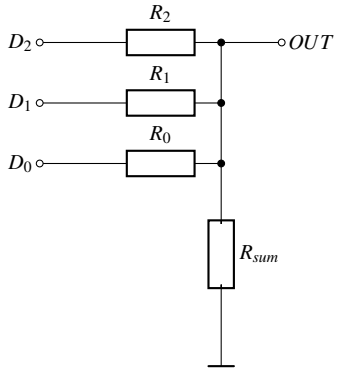
2 Lets build an DAC

I need 3 to 4 bits resulting in 8x8 or 16x16 display.

Simplest version

- $D2$ should have twice the effect on Out than $D1$.
- $D1$ should have twice the effect on Out than $D0$.

We put weights on the inputs ...



If $R1$ has twice the resistor value of $R2$ and $R0$ has twice the resistor value of $R1$ the voltage over R ... if R is smaller the crosstalk between bits might be bearable.

For example

- R2: $10k\Omega$
- R1: $22k\Omega$
- R0: $47k\Omega$
- R: $1k\Omega$

Note: I call this a weighted current sum DAC

The resistors at the D inputs applay the weight

the resulting current is summed up over $R/10$. Because thsi resistor is smaller, the inputs with their weight resistors function as current sources.

Different approach would be to block currents flowing into the data inputs with diodes ... but the large difference in the current means the diodes work at different points in their nonlinear curve.

Figure 1: $R = 12k$ and $R/10 = 470\Omega$

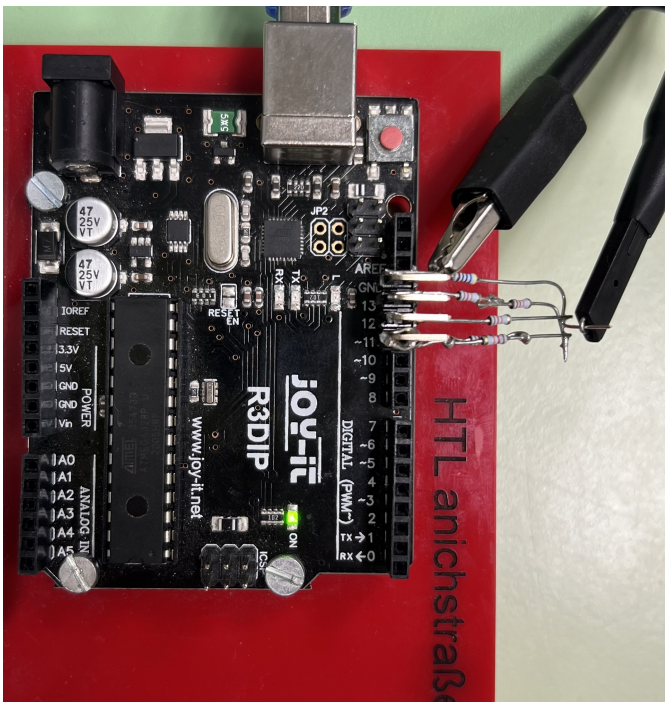


Figure 2: on an arduino uno

the arduino programm

```
const int MAXPIN = 3;
int pin[MAXPIN] = { 13, 12, 11 };

void setup() {
  for (int i = 0; i < MAXPIN; i++) {
    pinMode(pin[i], OUTPUT);
  }
}

void aWrite(int v) {
  digitalWrite(pin[0], v & 1);
  digitalWrite(pin[1], v & 2);
  digitalWrite(pin[2], v & 4);
}

void loop() {
  for (int i = 0; i < 8; i++) {
    aWrite(i);
  }
  delay(1);
}
```

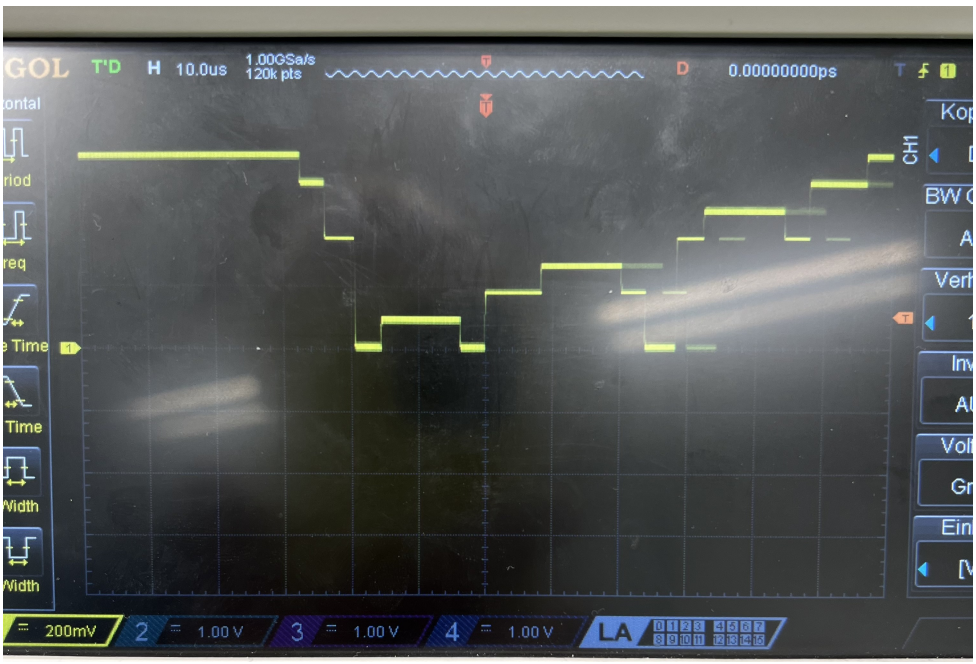


Figure 3: works .. *almost*

WRONG: it does work but we see the glidges.

When changing from 001 to 010 $aWrite$ will be set bit0 to 0 before setting bit1 to 1.

The same happens

- between 3 and 4,
- 5 and 6, here the output only falls to 4 before going to 6
- and 7 to 0, falls
  - first to 6 when bit0 goes 0,
  - then to 4 when bit1 goes 0
  - finally to 0 when bit2 is 0.

When we put in a delay longer the runtime of $digitalWrite$ the glitches are hardly visible.

```
const int MAXPIN = 3;
int pin[MAXPIN] = { 13, 12, 11 };

void setup() {
  for (int i = 0; i < MAXPIN; i++) {
    pinMode(pin[i], OUTPUT);
  }
}

void aWrite(int v) {
  digitalWrite(pin[0], v & 1);
  digitalWrite(pin[1], v & 2);
  digitalWrite(pin[2], v & 4);
}

void loop() {
  for (int i = 0; i < 8; i++) {
    aWrite(i);
    delay(1);
  }
  delay(10);
}
```

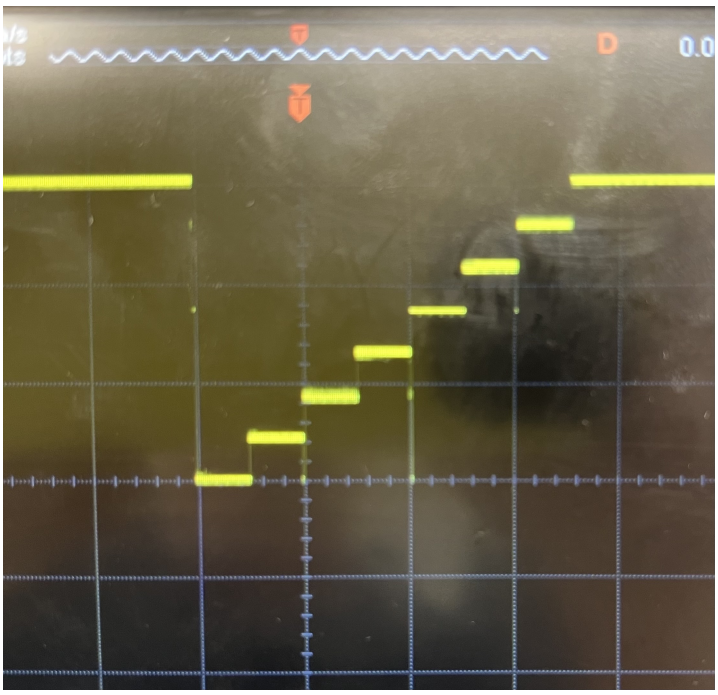


Figure 4: works .. *enough*

2.1 Other construction ... SMD

I don‘t want a pcb with SMD either ... so maybe the SMT-pins can possibly work ...

Figure 5: the parts

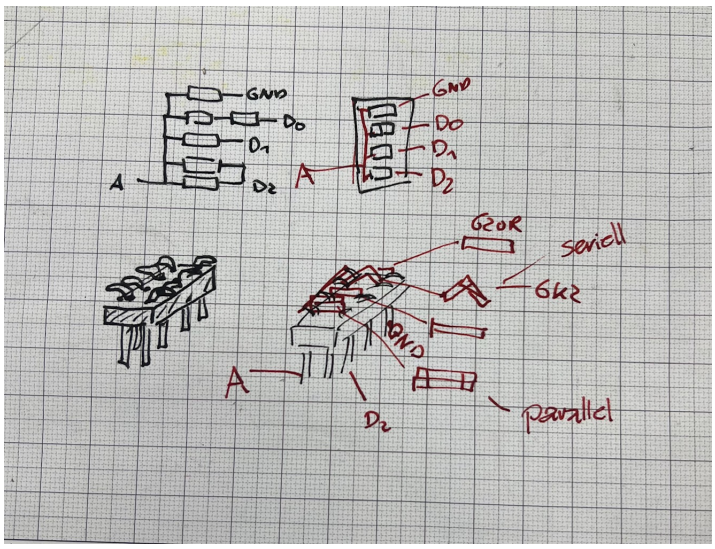


Figure 6: the design

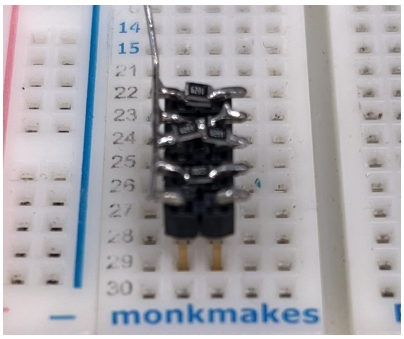


Figure 7: soldering practice ... result

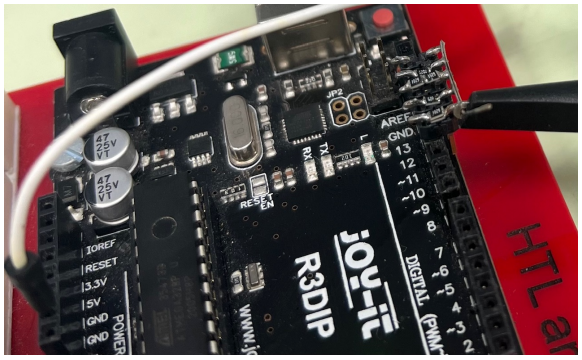


Figure 8: as arduino shield

Figure 9: DAC output ...

same program as with the wired DAC.

Works *almost*

- start 0 after a long $HIGH$
- $1, 2, 3$ stepping up
- **BUT** 4 is only 2

  4 in binary is 100 this is $D2$ input $HIGH$

  $D2$ has 2 resistors in parallel ... looks like $D1$

  means the *parallel* thing did not work.

Figure 10: maybe this here

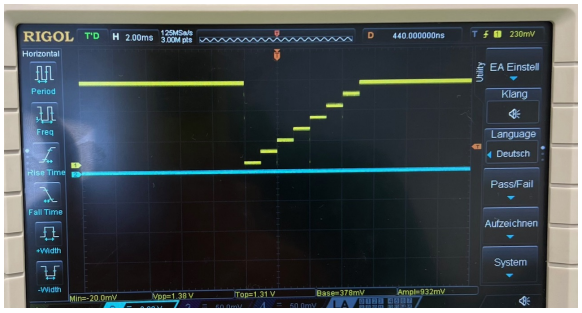


Figure 11: **WORKS**

3 Oscilloscope XY

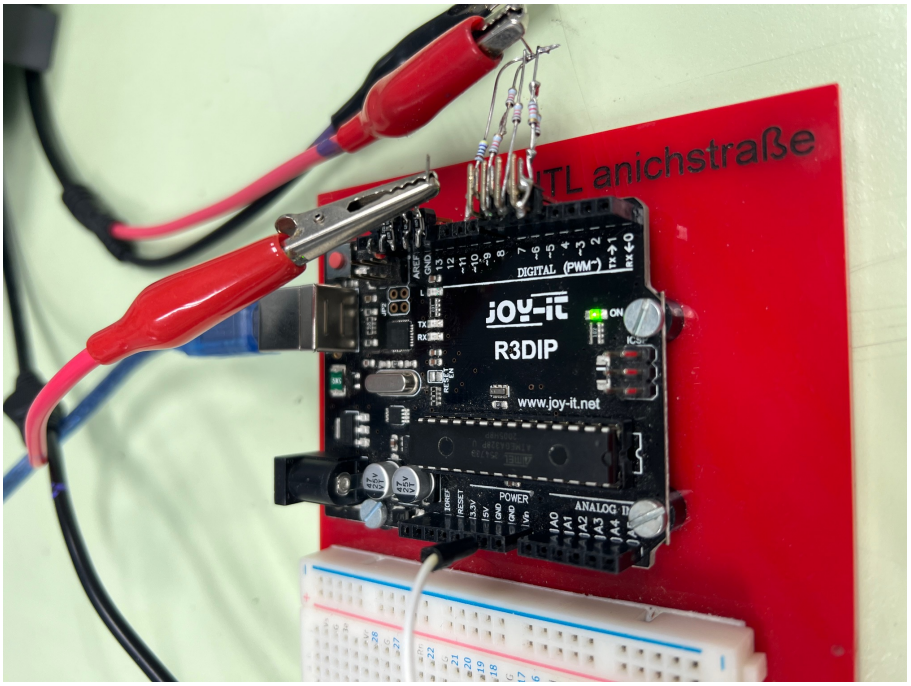


Figure 12: both DACs

We have more pins, a GND and 3 for the second channel.

```
const int MAXPIN = 6;
int pin[MAXPIN] = { 13, 12, 11, 6, 5, 4 };
//                  D0  D1  D2  D0 D1 D2
//        channel   A           B
int pinGND = 7;

void setup() {
  pinMode(pinGND, OUTPUT);
  digitalWrite(pinGND, LOW);
  for (int i = 0; i < MAXPIN; i++) {
    pinMode(pin[i], OUTPUT);
  }
}
```

We add a channel parameter to $aWrite$.

```
// channel "A" or "B"
void aWrite(char channel, int v) {
  if (channel == 'A') {
    digitalWrite(pin[0], v & 1);
    digitalWrite(pin[1], v & 2);
    digitalWrite(pin[2], v & 4);
  } else {
    digitalWrite(pin[3], v & 1);
    digitalWrite(pin[4], v & 2);
    digitalWrite(pin[5], v & 4);
  }
}

void loop() {
  for (int i = 0; i < 8; i++) {
    aWrite('A', i);
    aWrite('B', i);
```

```
    delay(1);
  }
}
```

Figure 13: both DAC outputs

3.1 und in XY-Mode

Figure 14: XY-mode

shouldn't this be a diagonal structure ?

This is a *specialtity* of RIGOL DS1704 ... maybe

- switching to XY switches the time base to $100ns$

cool feature (of DS1704) switch display from $Full$ to $Split$

Figure 15: XY-mode and Yt

the oscilloscope zooms into *one* slope.

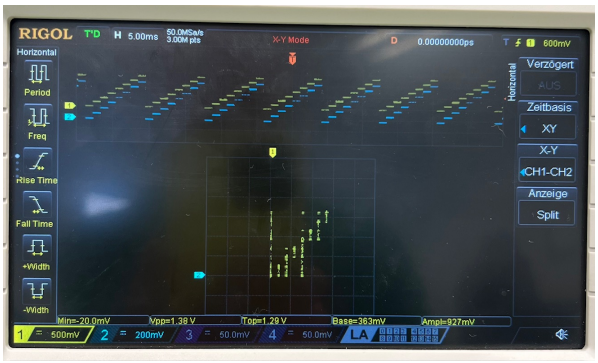
zooming out to $2s/DIV$

we only count upwards

```
for (int i = 0; i < 8; i++) {
```

therefore I assume these are the steps produced by the run time of the $digitalWrite$ calls ... approximately $3s$

zooming out to $5ms/DIV$ means $50ms$ measurement time.



a lot of artefacts/noise that could be less by making

- the $delay$ longer
- use direct port access instead of $digitalWrite$

... maybe later.

4 Display a letter

How to display any signal sequence.

We put the value pairs into a list.

```
int val[][2] = {
  //
  { 1, 7 }, { 2, 7 }, { 3, 7 }, { 4, 7 }, //
};
```

```
int MAXval = sizeof(val)/sizeof(val[0]);
// the size of the whole divided by the size of the first
// gives the number of element
```

and in *loop* we do

```
for (int i = 0; i < MAXval; i++) {
  aWrite('A', val[i][0]);
  aWrite('B', val[i][1]);
  delay(1);
}
```

why 5 dots in a horicontal line.

4.1 Try a digit 5

```
int val[][2] = {
  //
  { 1, 7 },  { 2, 7 },  { 3, 7 },  { 4, 7 }, //
  { 1, 6 },  //
  { 1, 5 },  { 2, 5 },  { 3, 5 },   //
  { 1, 4 },                        { 4, 4 }, //
                                   { 3, 4 }, //
  { 1, 2 },                        { 4, 2 }, //
             { 2, 1 },  { 3, 1 } //
};
```

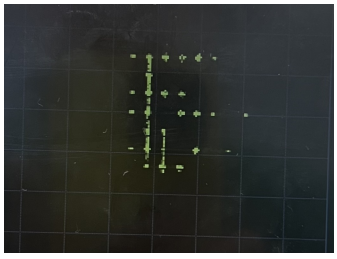


Figure 16: noisy 5

the leftmost dots are glitches because we do not have a 0 in X.

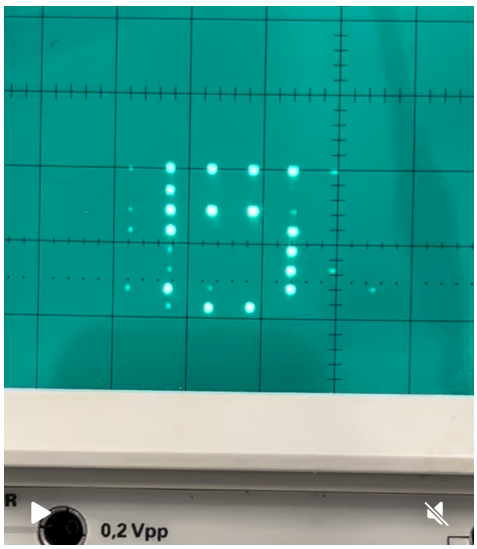


Figure 17: better on analog oscilloscope

4.2 Digit 4 and 5

```
int val[][2] = {
  // four
              { 2, 7 },             //
              { 2, 6 },   //
  { 1, 5 },               //
  { 1, 4 },                                 { 4, 4 }, //
  { 1, 3 },   { 2, 3 },   { 3, 3 },   { 4, 3 }, //
                                            { 4, 2 }, //
                                            { 4, 1 }, //
                                            { 4, 0 }, //
  // five
  { 1, 7 },   { 2, 7 },   { 3, 7 },   { 4, 7 }, //
  { 1, 6 },   //
  { 1, 5 },   { 2, 5 },   { 3, 5 },    //
  { 1, 4 },                                 { 4, 4 }, //
                                            { 4, 3 }, //
                                            { 4, 2 }, //
  { 1, 1 },                                 { 4, 1 }, //
               { 2, 0 },   { 3, 0 } //
};
```

But the digits are plotted over each other.

We should separate them, plot 4 for some time *and* then 5.

4.2.1 Building a REPEAT option into the data

the engine code in the loop-function gets two options

- REPEAT start: must
  - remember where the loop/repeat starts: the line number
  - how often the section should be repeated
- REPEAT NEXT: must
  - check if we already made the requested repetitions.
  - if not, jump back to the start line of the REPEAT.

For readability and because I donot want to have to remember the value or write comments we define two constants

```
const int REPEAT = -1;
const int REPEAT_NEXT = -2;
```

and check for them in the code

```
if (val[i][0] == REPEAT) {
  repeat_start = i+1;  // remember start line of block
  repeat_count = val[i][1];
}
else if (val[i][0] == REPEAT_NEXT) {
  if (repeat_count > 0) {
    repeat_count--;
    i = repeat_start;
  }
}
else {
  *old code
```

I do not know if changing the variable of the for-loop will work ... actually the loop variable gets incremented at the start, therefore it is

```
repeat_start = i;  // remember start line of block
```

We put

- a repeat $n-times$ line before a digit block and

  ```
  { REPEAT, 10 }
  ```

- a repeat-next add the end

  ```
  { REPEAT_NEXT, -1 }
  ```

It works ... how to do show a video ? on the web: http://sis.htlinn.ac.at/blaetter

4.2.2 Variables and constants

At the start:

```
const int REPEAT = -1;
const int REPEAT_NEXT = -2;
```

then we can use these in the patterns:

```
...
{ 1, 1 },                          { 4, 1 }, //
            { 2, 0 },  { 3, 0 }, //
{ REPEAT_NEXT, -1 },
// four
{ REPEAT, 50 },
                     { 3, 7 },           //
...
```

Inside loop:

```
static int repeat_start = 0;
static int repeat_count = 0;
for (int i = 0; i < MAXval; i++) {
  if (val[i][0] == REPEAT) {
    repeat_start = i;  // remember start line of block
    // CAUTION i is already next line
    repeat_count = val[i][1];
  }
  else if (val[i][0] == REPEAT_NEXT) {
```

5 Next things

- better REPEAT. The time a picture ist displayed, depends on the number of points, because only a repeat count is specified.

  Time would be better.

- better pictures, read bitmaps.
- bigger points or a 4 bit DAC.

6 Appendix

6.1 full code

```
1 const int MAXPIN = 6;
2 int pin[MAXPIN] = { 13, 12, 11, 6, 5, 4 };
3 //                  D0  D1  D2  D0 D1 D2
4 //        channel   A           B
5 //        Port      B5  B4  B3  D6 D5 D4
6 int pinGND = 7;
7
8 const int REPEAT = -1;
9 const int REPEAT_NEXT = -2;
10
11 int val[][2] = {
12   // five
13   { REPEAT, 50 },
14   { 1, 7 },  { 2, 7 },  { 3, 7 },  { 4, 7 }, //
15   { 1, 6 },  //
16   { 1, 5 },  { 2, 5 },  { 3, 5 },   //
17   { 1, 4 },                         { 4, 4 }, //
18                                     { 4, 3 }, //
19                                     { 4, 2 }, //
20   { 1, 1 },                         { 4, 1 }, //
21               { 2, 0 },  { 3, 0 }, //
22   { REPEAT_NEXT, -1 },
23   // four
24   { REPEAT, 50 },
25                        { 3, 7 },           //
26             { 2, 6 },  //
27   { 1, 5 },             //
28   { 1, 4 },                         { 4, 4 }, //
29   { 1, 3 },  { 2, 3 },  { 3, 3 },  { 4, 3 }, //
30                                     { 4, 2 }, //
31                                     { 4, 1 }, //
32                                     { 4, 0 }, //
33   { REPEAT_NEXT, -1 },
34   // three
35   { REPEAT, 50 },
36             { 2, 7 },           //
37   { 1, 6 },           { 3, 6 },  //
38                       { 3, 5 },            //
39   { 1, 4 }, { 2, 4 }, //
40             { 2, 3 }, { 3, 3 }, //
41                       { 3, 2 }, //
42   { 1, 1 },           { 3, 1 }, //
43             { 2, 0 }, //
44   { REPEAT_NEXT, -1 },
45   // two
46   { REPEAT, 50 },
47             { 2, 7 },           //
```

```
48   { 1, 6 },           { 2, 6 },  //
49                       { 3, 5 },             //
50             { 2, 4 }, //
51   { 1, 3 },  //
52   { 1, 2 }, //
53   { 1, 1 }, { 2, 1 }, { 3, 1 }, //
54   { REPEAT_NEXT, -1 },
55 };
56
57 int MAXval = sizeof(val) / sizeof(val[0]);
58 // the size of the whole divided by the size of the first
59 // gives the number of element
60
61 void setup() {
62   pinMode(pinGND, OUTPUT);
63   digitalWrite(pinGND, LOW);
64   for (int i = 0; i < MAXPIN; i++) {
65     pinMode(pin[i], OUTPUT);
66   }
67 }
68
69 // channel "A" or "B"
70 void aWrite(char channel, int v) {
71   if (channel == 'A') {
72     digitalWrite(pin[0], v & 1);
73     digitalWrite(pin[1], v & 2);
74     digitalWrite(pin[2], v & 4);
75     //PortB bit 5,4,3
76     // int d = ((v&1) << 2) + (v&2) + ((v%4) >> 2);
77     // PORTB = (v&7) << 3;
78   } else {
79     digitalWrite(pin[3], v & 1);
80     digitalWrite(pin[4], v & 2);
81     digitalWrite(pin[5], v & 4);
82     // PortD bit 6,5,4
83     //int d = (v&1) + (v&2) + (v&4)
84     //PORTD = (v & 0x7) << 3;
85   }
86 }
87
88 void loop() {
89   static int repeat_start = 0;
90   static int repeat_count = 0;
91   for (int i = 0; i < MAXval; i++) {
92     if (val[i][0] == REPEAT) {
93       repeat_start = i;  // remember start line of block
94       // CAUTION i is already next line
95       repeat_count = val[i][1];
96     }
97     else if (val[i][0] == REPEAT_NEXT) {
98       if (repeat_count > 0) {
99         repeat_count--;
100         i = repeat_start;
101       }
102     }
103     else {
104       aWrite('A', val[i][0]);
105       aWrite('B', val[i][1]);
106       delay(1);
107     }
108   }
```

```
109 }
```