LED Matrix Module (HUB08 und ähnliche)

Simple Ansteuerung durch Schieberegister. Kaskadierbar (die Anzahl der Bits, die man „reinschieben“ muss vergrössert sich entsprechend.

Das erste Bit, welches man shifted, ist am Ende des Shifts am weitesten vom jeweiligen Anschluss weg. Am Ende des Shift-Vorgangs werden die Bits gespeichert / gelatched, durch einen kurzen LOW-Impuls auf dem L-Eingang (Latch).

Die Bits die man shifted adressieren die LEDs in den Spalten. Welche Zeile man beim Einschalten der LEDs (Output-Enable) leuchtet, bestimmt die Zeilen Adresse (A,B,C,D)

Zeilen-Adressen stehen maximal 16 zur Verfügung (typischerweise 4Bit-Decoder 74xx138). Hat das Modul mehr als 16 Zeilen, dann werden zwei (oder mehr) Zeilen auf einmal selektiert.

Für den zweiten „Satz“ an Zeilen existiert dann ein weitere Reihe Schieberegister und ein weiterer Eingang.

Beim DMD LED-Kit ist die Geometrie 2 Module a 64x32 nebeneinander.

Sie werden mit 16 Zeilen Adressen angesteuert immer zwei Zeilen gleichzeitig (z.B. die 1. und die 17. Zeile).

Pro Modulzeile werden 64 Bit oder 8 Byte geshifted, insgesamt pro Zeile 16 Byte mal 32 Zeilen macht 512 Byte per Buffer, wenn jedes Bit eine LED repräsentiert.

Auf Grund der Ansteuerung können bei so einem Modul niemals alle LEDs auch einmal leuchten, sondern immer nur zwei Zeilen also maximal 256 LEDs.

Daher muss die Ansteuerungslogik ständig alle Zeilen nacheinander aufleuchten lassen, so schnell, dass es das Auge nicht wahrnimmt (wie früher beim Röhrenfernseher).

Bei der Display Geometrie 128x32 in 16 Gruppen zu 2 Zeilen, ist der Zyklus eben 16.

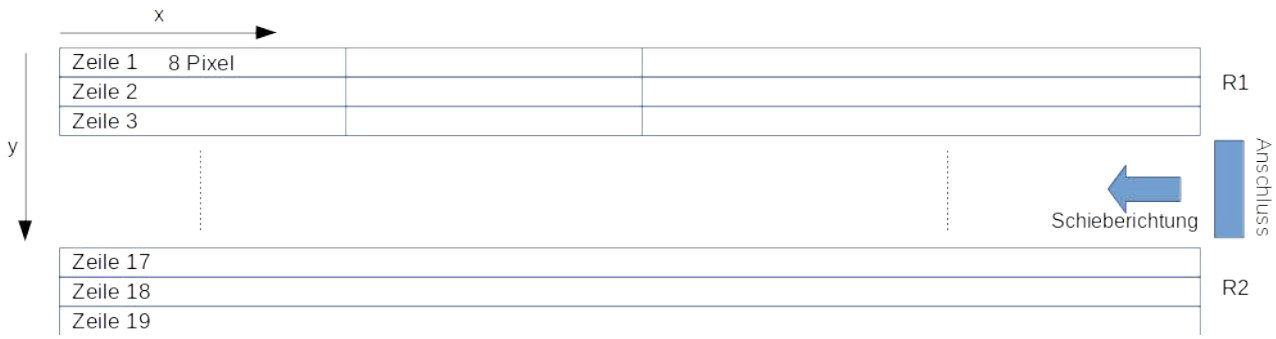
Eine weitere Einschränkung ist die Tatsache, dass eine LED nur an oder aus geschaltet werden kann. Will man unterschiedliche Helligkeitwerte erreichen, dann geht das nur über die relative Einschaltdauer (RED).

Da man ja sowieso ständig die Zeilen erneuert, kann man das dadurch erreichen, dass man z.B. bestimmte LED in einem Durchlauf mit leuchten lässt und beim nächsten nicht. Diese erscheint nur halb so hell wie eine LED, die bei allen Durchläufen leuchtet.

Für die unterschiedlichen Eingänge der Schieberegister weisen die Module entsprechende Anschlüsse auf. Bei einem einfarbigen mit Zeilen Gruppen zu 2 Zeilen z.B. R1 / R2 (rote LEDs, deswegen „R“).

Eine LED leuchtet übrigens dann wenn man ein „0“ Bit schreibt (negative Logik).

Ein zweifarbiges Modul hat entsprechend weitere Eingänge z.B. G1/G2 für ein rot / grün Modul in der Geometrie wie oben.



Zeile 1 und 17 bilden jeweils eine Gruppe, genau wie Zeile 2 und 18 usw.

Bits die über R1 eingespeist werden, erscheinen in der oberen Hälfte, Bits über R2 in der unteren Hälfte (ab Zeile 17).

Damit das Auffrischen des LED Moduls möglichst einfach / schnell erfolgen kann, ist der Aufbau des Buffers genau an der Pixel-Struktur der Module orientiert.

D.h. die Bits / Pixel liegen genau in der Reihenfolge im Speicher vor, wie sie über die Eingänge der Schieberegister ausgegeben werden müssen. Beim Aufbau wie oben, sind das je 2 Bit (R1/R2) pro Shift.

Dadurch liegen im 1. Byte die ersten vier Pixel der ersten Zeile und die ersten 4 Pixel der 17. Zeile ! und zwar immer abwechseln. Wenn man also die Bits im erste Byte vom höchsten (MSB) zum niedrigsten (LSB) anschaut, ergibt sich diese Zuordnung:

Bit	X	Y
7	0	0
6	0	16
5	1	0
4	1	16
3	2	0
2	2	16
1	3	0
0	3	16

Will man einen bestimmten Punkt im Display setzen, dann geht das so (Pseudo-Code):
(Divisionen sind immer ganzzahlig ohne Rest)

```
Definition: Region (R1/R2 oben/unten)

bitmask = {
      01111111,
      11011111,
      11110111,
      11111101,
      10111111,
      11101111,
      11111011,
      11111110 };
```

```
setzePunkt( x, y ):
      bytesProZeile = breite / 4;  // nicht durch 8, da nur 4 Pixel pro byte
      region = y / 16;
      zeile = y mod 16;
      index = zeile * bytesProZeile + x / 4;
      buffer[index] = buffer[index] | bitmask[x mod 4 + region*4 ];
```

Die Bitmaske mit der Or-verknüpft wird, um das passende Bit auf 0 zu setzen, wird so gewählt, dass passend für die Region und x mod 4 immer das richtige Bit „getroffen“ wird.

Diese komplexe Anordnung der Bits im Buffer bewirkt, dass der Update Algorithmus zum Auffrischen der LEDs relativ einfach gehalten werden kann:

```
refreshScreen:
      for( zeile = 0 .. höhe / regions ) {
            bytesProScanZeile = breite /4;
            for( k = 0 .. bytesProScanZeile ) {
              wert = buffer[zeile*bytesProScanZeile+k];
              for( i = 0 .. 3 ) {
                ausgabe bit 7&6 von wert an port für R1/R2
                wert = wert << 2;
              }
            }

            // halte die ausgegebenen bits fest
            pulse latch to low;

            // LEDs dieser Zeile(n-Gruppe) für eine Weile leuchten lassen
            // je länger die relative Einschaltdauer, desto heller
            hold output enable on low for some time;
      }
```

Das Erzeugen von Schrift (allgemein Sprites / Mustern) wird der Einfachheit wegen nur auf Bytegrenzen (in X-Richtung) erlaubt, also alle 4 Pixel.

Mit dem Wissen bzgl. des Bufferaufbaus, kann man ein Sprite / ein Buchstaben eines Fonts so vorberechnen, dass die Bits schon genau auf den Aufbau des Buffers passen. Ist das Sprite allerdings höher als eine Region (hier 16 Pixel), dann müssen die Bitmuster für alle Zeilen die in die untere Region fallen um ein Bit nach rechts geschoben werden.

Im allereinfachsten Fall macht man das für eine bestimmte y-Position ebenfalls schon vorher, dann kann man die Bytes einfach an der richten Stelle in den Buffer kopieren. Die y-Position ist in diesem Fall allerings fest. Vorteil: man braucht keine zusätzliche Rechenleistung beim Reinkopieren.

Analog ist das Vorgehen bei kompletten Bildern, die die gesamte Fläche der LED Matrix einnehmen. Auch hier wir das Bild komplett so umgerechnet, dass es auf die Anordnung im Buffer passt. So braucht man nur einfache Kopier-Funktionen um ein Bild anzuzeigen.

Mehrere Helligkeitsstufen

Will man ein Pixel unterschiedlich hell aufleuchten lassen können, dann steuert man dies über die relative Einschaltdauer, wie weiter oben schon erläutert.

Zusammen mit der Refresh-Logik aus dem letzten Abschnitt ergibt sich folgende Erweiterung:

- man benutzt nicht einen Hintergrund Buffer sondern zwei: b1 / b2
- Bits, die *weder* in b1 noch in b2 gesetzt sind bleiben schwarz
- Bits, die *nur* in b1 gesetzt sind, bekommen eine relative Einschaltdauer von 33%
- Bits, die *nur* in b2 gesetzt sind, bekommen eine relative Einschaltdauer von 66%
- Bits, die in b1 *und* b2 gesetzt sind, bekommen eine relative Einschaltdauer von 100%

Auf diese Art kann man 4 verschiedene Helligkeitsstufen realisieren.

Um diese relativen Einschaltdauern zu realisieren, wird die „refreshScreen“ Logik erweitert. Zunächst wird nicht ein Buffer durchlaufen, sondern zwei gleichzeitig. Ausserdem werden die nacheinander kommenden „refreshScreen“ Läufe qualifiziert: es gibt drei „Arten“ ein B1, B2 und ein B12 analog zu oben (bei B12 = 100% RED eigentlich nur zwei).

In einem B1 Lauf werden nur LEDs eingeschaltet, die in b1 eingesetzt sind usw. Dann verteilt man die Arten der Läufe in einem Zyklus so, dass man auf die gewünschten relativen Einschaltdauern kommt.

Mode	B1	B2	B12	LED
1	x			
2		x		
3		x		
4	x	x	x	

```
mode = 1 .. 3; // mode wird vom Aufrufer gesteuert

refreshScreen( mode ):
      for( zeile = 0 .. höhe / regions ) {
            bytesProScanZeile = breite /4;
            for( k = 0 .. bytesProScanZeile ) {
              wert1 = buffer1[zeile*bytesProScanZeile+k];
              wert2 = buffer2[zeile*bytesProScanZeile+k];
              switch( mode ) {
                case 1:   wert = wert1;
                case 2,3: wert = wert2;
                case 3
              }

              for( i = 0 .. 3 ) {
                ausgabe bit 7&6 von wert an port für R1/R2
                wert = wert << 2;
              }
            }

            // halte die ausgegebenen bits fest
            pulse latch to low;

            // LEDs dieser Zeile(n-Gruppe) für eine Weile leuchten lassen
            // je länger die relative Einschaltdauer, desto heller
            hold output enable on low for some time;
      }
```