Borg 16 – Getting Started

Christian Kroll

Chaostreff Dortmund / LABOR e.V.

09.11.2014 / Borg16-Folien

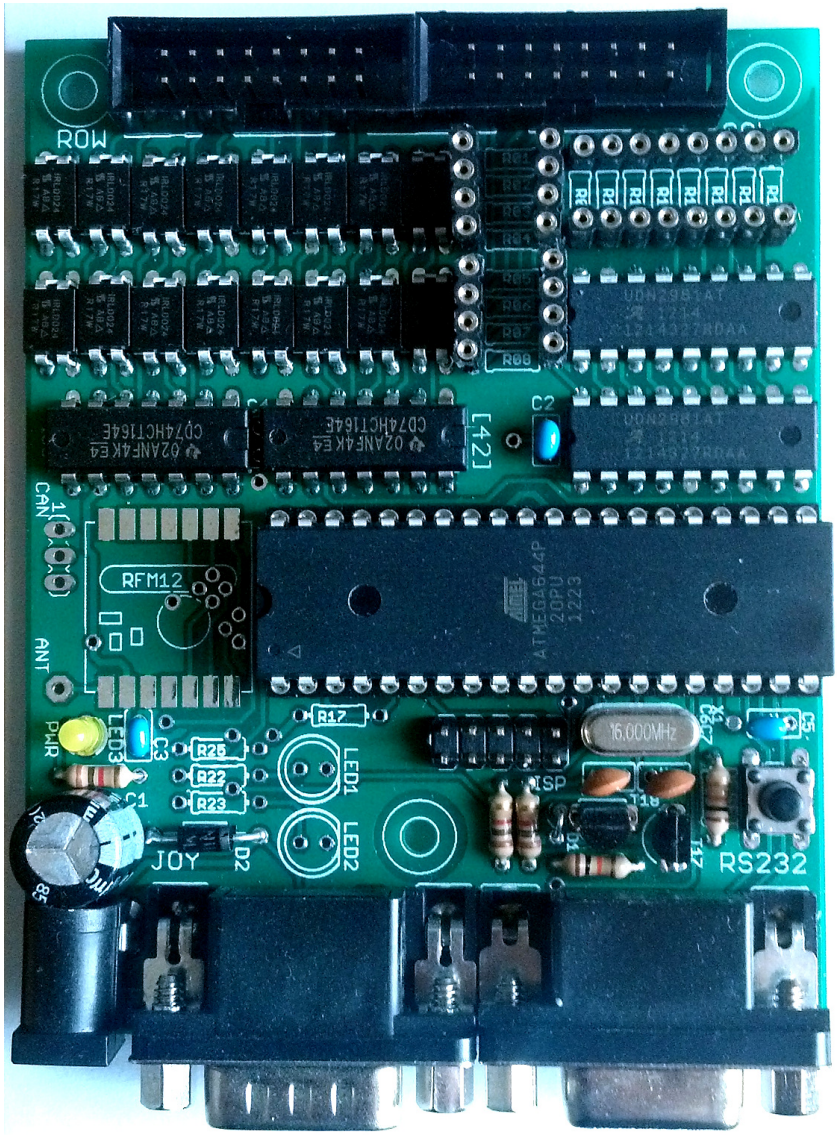
Gliederung

Gliederung

Der Borg 16

- der Borg 16 ist eine einfache Schaltung für 16×16-LED-Matrizen
- den Kern bildet eine professionell gefertigte Leerplatine für bedrahtete Bauteile
- die Bausätze enthalten alle Bauteile bis auf die LEDs und deren Vorwiderstände
- optional lässt sich noch ein CAN-Controller (SMD) oder ein RFM12-Funk-Chip verbauen (nicht in den Bausätzen enthalten)

Draufsicht auf die Platine des Borg 16

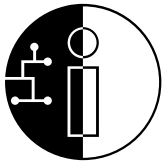


Hardware des Borg 16

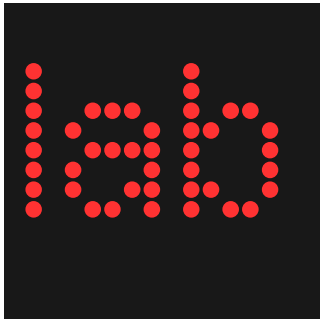
- AVR `ATmega32` @ 16 MHz, 2 KiB RAM, 32 KiB Flash, 1 KiB EEPROM
- auch unterstützt: `ATmega644(P)` und `ATmega1284(P)`
- 2 `74HCT164`-Schieberegister
- 16 `IRLD024`-MOSFETs (sperrend) als Zeilentreiber
- 2 `UDN2981A`-Treiber für die Spalten
- Anschluss für Joystick nach dem 9 pol. Atari-Standard
- Anschluss für RS232
- ISP über 5×2 pol. Header ausgeführt (STK200-Belegung)
- optionale Komponenten:
  - CAN-Bus-Controller `MCP2515SO` mit Transceiver `MCP2551SN`
  - `RFM12`-Controller
  - CAN und RFM12 schließen sich leider gegenseitig aus

Software des Borg 16: Die Borgware 2D

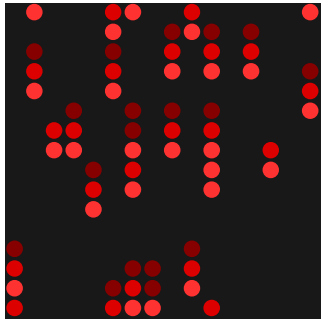
- enthält über 25 Animationen und 5 Spiele (+2 Varianten)
- lässt sich wie der Linux-Kernel menügeführt zusammenstellen
  - Auswahl von Spielen und Animationen
  - Optionen für das Finetuning
- enthält eine einfache API für Grafik- und Joystick-Operationen
- enthält einen API-Simulator, um Animationen und Spiele direkt auf dem PC debuggen zu können (für Linux, *BSD und Windows)

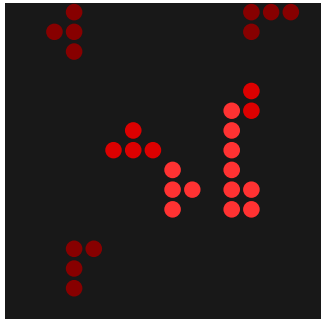


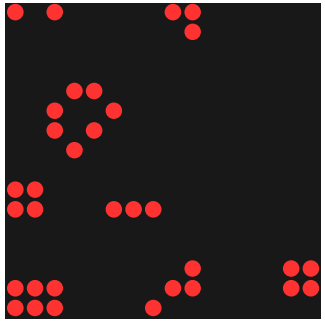
Impressionen

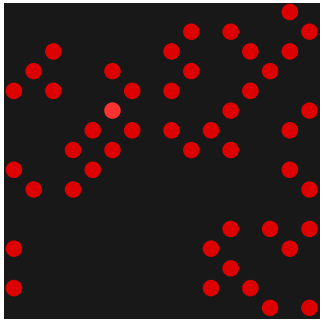

Scrolltext


Fire


Matrix


Stonefly

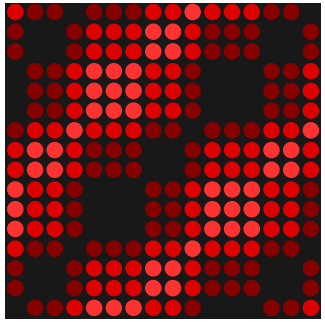

Game of Life


Langton's Ant

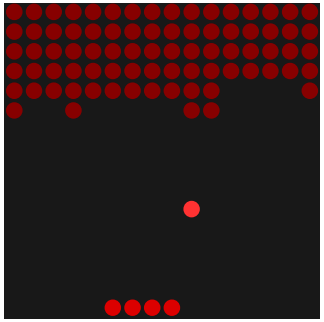

Bitmap Scroller

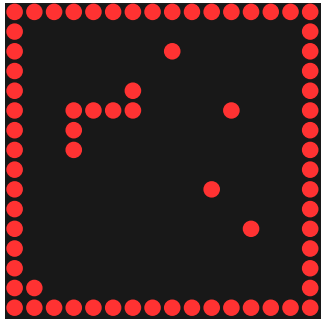

Plasma

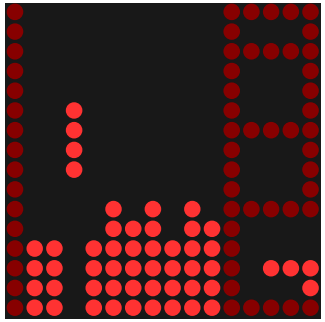

Psychedelic

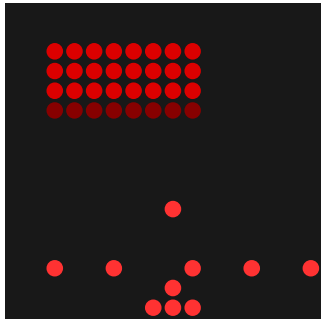

Squares


Menu


Breakout


Snake


Tetris

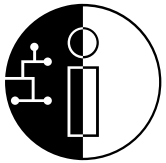

Space Invaders

Gliederung

Voraussetzungen unter Linux

Am Beispiel von Ubuntu (ähnlich für andere Distributionen)

- notwendige Pakete, um Build-System samt AVR-Image zu bauen:
  - AVR-Crosscompiler-Toolchain: `gcc-avr`, `avr-libc` und `binutils-avr`
  - Host-Compiler-Toolchain: `gcc`, `libc6-dev` und `binutils`
  - `make` als eigentliches Build-Backend
  - `libncurses5-dev` für die Menükonfiguration
- der Simulator benötigt noch `libfreeglut3-dev`
- `avrdude` um das Image auf den AVR flashen zu können
- `git` um die Borgware 2D zu beschaffen bzw. zu aktualisieren



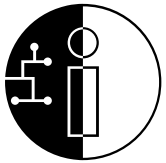
AVR-Programmer unter Linux

- Um den USBasp bzw. andere Programmer ohne Root-Rechte nutzen zu können, muss man eine udev-Regeldatei (z.B. *99-usbasp.rules*) mit folgendem Inhalt anlegen:

/etc/udev/rules.d/99-usbasp.rules

```
# Atmel AVR ISP mkII
ATTR{idVendor}=="03eb", ATTR{idProduct}=="2104", GROUP="plugdev", MODE="660"

# usbprog bootloader
ATTR{idVendor}=="1781", ATTR{idProduct}=="0c62", GROUP="plugdev", MODE="660"

# USBasp programmer
ATTR{idVendor}=="16c0", ATTR{idProduct}=="05dc", GROUP="plugdev", MODE="660"
```

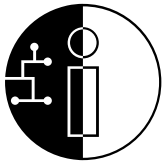
- Anschließend kann jeder Nutzer, der der Gruppe *plugdev* angehört, den Programmer nutzen.



Voraussetzungen unter Windows: WinAVR

Crosscompiler für die AVR-Plattform unter Windows

- `WinAVR`[1] ist eine auf dem gcc basierende AVR-Crosscompiler-Toolchain
- enthält zusätzlich `avrdude` in einer Windows-Version
- bei der Installation sicherstellen, dass im Setup-Dialog die Optionen *Install Files* und *Add Directories To PATH* ausgewählt sind
- enthält noch einen erweiterten Quellcode-Editor, der für die Borgware 2D aber nicht benötigt wird

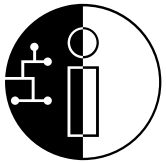


[1] http://winavr.sourceforge.net/download.html

Voraussetzungen unter Windows: Cygwin

Linux für Windows

- `Cygwin`[2] bildet eine Linux-Umgebung unter Windows nach
- liefert einen gcc-Compiler (Host) für Windows mit
- über das Cygwin-Setup lassen sich zusätzlich viele Anwendungspakete aus dem Linux-Umfeld in Windows installieren
- folgende dieser Pakete benötigt die Borgware 2D
  - `gcc4`
  - `make`
  - `bc`
  - `libncurses-devel`
  - ggf. `git` (falls nicht anderweitig installiert)
- die systemweite *PATH*-Umgebungsvariable muss evtl. um den Pfad *C:\Cygwin\bin* (o.ä.) ergänzt werden, wobei dieser Pfad *vor* dem des WinAVR stehen muss

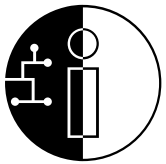


[2] http://www.cygwin.com

USBasp unter Windows

Installation des Treibers mit `libusb-win32`

- aktuelles `libusb-win32`[3] Release herunterladen und entpacken
- nun den USBasp-Programmer in den USB-Port stecken
- unter *libusb-win32.w.x.y.z\bin* die `inf-wizard.exe` starten
- im sich öffnenden Dialog *USBasp* auswählen und `<Next>` wählen
- im folgenden *Device Configuration*-Dialog wieder `<Next>` wählen
- anschließend einen Pfad für die zu erstellende INF-Datei angeben
- jetzt erscheint ein Dialog mit einem `<Install Now...>`-Button
- Treiber-Überprüfungswarnung ignorieren
- eh voilà, jetzt sollte ein *Installation successful* erscheinen
- evtl. den USBasp nochmal ein- und ausstecken



[3] http://sourceforge.net/projects/libusb-win32/files/

Gliederung

Die Borgware 2D beschaffen

Ein Terminal (Cygwin unter Windows) starten:

Quellcode aus GitHub auschecken

```
git clone https://github.com/das-labor/borgware-2d
cd borgware-2d
git submodule init
git submodule update
```

Die letzten beiden Schritte binden die RFM12-Library in den Quellcode ein. Mit einem `git pull origin master` im *borgware-2d*-Verzeichnis lässt sich der Quellbaum später aktualisieren.

Die erste Konfiguration

- im frisch ausgecheckten Verzeichnis ein `make menuconfig` ausführen

```
Borg Configuration

                              Main Menu
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes.
  <Esc><Esc> to exit, <?> for Help.  Legend: [*] enabled  [ ] disabled
  [-] not available

            Load a Default Configuration  --->
            General Setup  --->
            Borg Hardware  --->
            Features  --->
            [ ] Scrolltext Support  ---> (NEW)
            [ ] RFM12 Support  ---> (NEW)
            [ ] joystick support  ---> (NEW)
            [ ] CAN bus support  ---> (NEW)
            [-] Menu Support
            Games  --->
            v(+)

                <Select>    < Exit >    < Help >
```

- klappt es nicht, fehlen oft die Header von ncurses (wirklich `libncurses5-dev` bzw. unter Cygwin `libncurses-devel` installiert?)

Der erste Schritt nach einem frischen Checkout

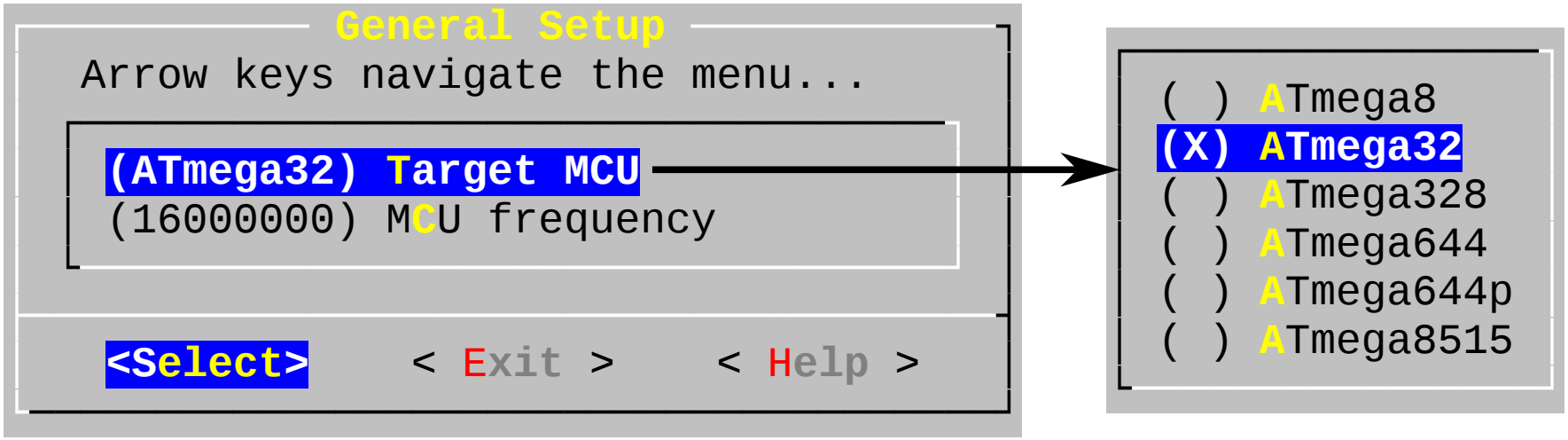
- zunächst muss unter `Load a Default Configuration` *einmalig* das Profil `borg-16` geladen werden

```
Load Default Configuration
Select the default profile you wish to load. <Esc><Esc> to abort.

                    ( ) borg-16
                    ( ) borg-ancient
                    ( ) borg-andre
                    ( ) borgjacke
                    ( ) borg-ls
                    ( ) FFM-Jochen
                    ( ) FFM-LedBrett

                    <Select>      < Help >
```

- ein Return bringt uns unmittelbar zum Hauptmenü zurück
- wir haben nun eine sinnvolle Anfangskonfiguration

Borgware 2D: General Setup

- im `General Setup` lassen sich der verwendete Mikrocontroller samt Taktfrequenz einstellen



- gültige Werte für den Borg 16 sind `ATmega32` und `16000000` Hz
- wird der pinkompatible ATmega644(P) benutzt, so ist dies hier entsprechend umzutragen

Borgware 2D: Borg Hardware

- unter `Borg Hardware` werden der zu verwendende Hardware-Treiber und die Eigenschaften der Matrix konfiguriert
- für den Borg 16 sind die hier gezeigten Werte richtig

```
Borg Hardware
Arrow keys navigate the menu...

(16) Number of rows
(16) Number of columns
(3) Number of brightnes-levels
(Borg-16) Hardware Driver
Borg16 port setup  --->

<Select>    < Exit >    < Help >
```

```
Borg16 port setup
Arrow keys navigate the menu...

(PORTC) Column Port 1 (right)
(PORTA) Column Port 2 (left)
(PORTD) port for row shiftregisters
--- pin numbers on shiftregister port
(Pin4) /MCLR Pin
(Pin6) CLK Pin
(Pin7) DATA Pin
--- fixing hardwareproblems in software
[ ] reverse cols
[ ] invert rows
--- for borg jacket
[ ] interlaced rows
[ ] interlaced cols

<Select>    < Exit >    < Help >
```

- die Matrix-Parameter müssen mit den fest verdrahteten Werten im ausgewählten Treiber übereinstimmen

Borgware 2D: Features

- `Features` enthält einen kryptographisch sicheren Zufallsgenerator

```
Features
Arrow keys navigate the menu...

[*] prng random number generator
[-] CAN Time Extension

<Select>    < Exit >    < Help >
```

- viele Animationen und Spiele lassen sich nur auswählen, wenn er aktiviert ist
- `CAN Time Extension` ist nur in Verbindung mit der Labor-CAN-Infrastruktur sinnvoll

Borgware 2D: Scrolltext Support

- `[*] Scrolltext Support -->` aktiviert die API-Funktion für Lauftexte, die alle Spiele benötigen
- im Untermenü lassen sich noch Schriftart, Scrollgeschwindigkeit und der regelmäßig erscheinende Lauftext einstellen

```
Scrolltext Support
Arrow keys navigate the menu...

(Arial_8) Scrolltext Font
(128) Scrolltest buffer size
(20) Default x speed
(20) Default y speed
     Default text: "</#www.das-labor.org"

<Select>    < Exit >    < Help >
```

- der Text muss mit „</#“ beginnen (Lauftext von rechts nach links), sonst erscheint nichts
- tatsächlich verbirgt sich dahinter eine Syntax für Lauftexteffekte, es sei an dieser Stelle auf das Labor-Wiki[4] verwiesen

[4] http://www.das-labor.org/wiki/BorgTextAnim

Borgware 2D: Joystick Support

- `Joystick Support -->` Aktvierung der Joystick-Unterstützung (Voraussetzung für Spiele) und Einstellung des Joystick-Typs
- Hardware-spezifische Eigenschaften befinden sich im Untermenü

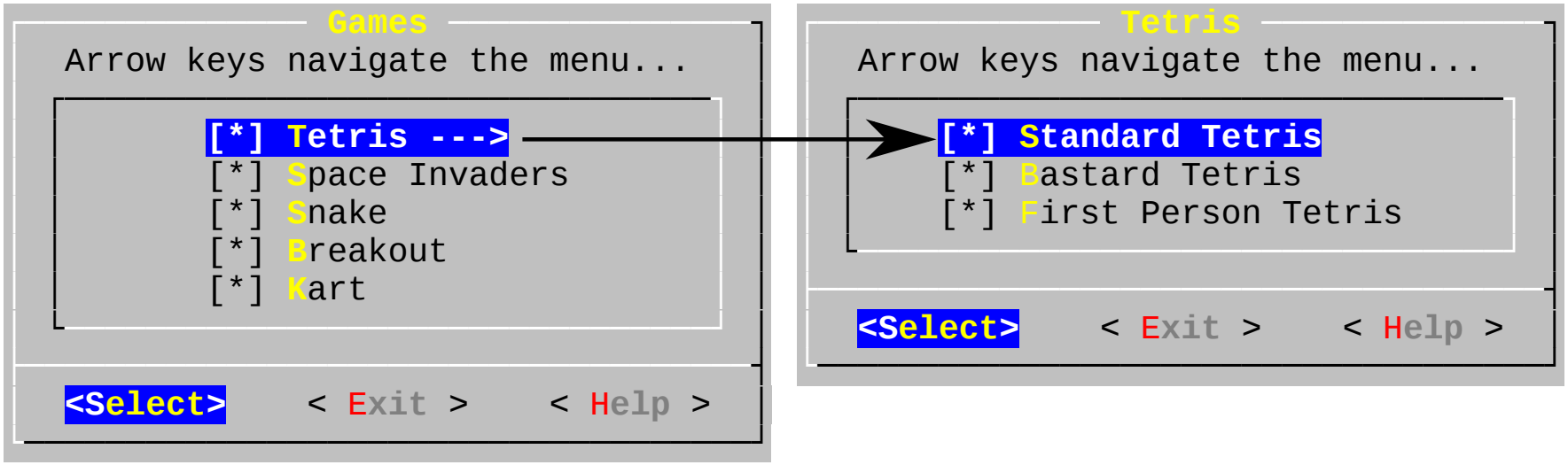
```
Joystick Support
Arrow keys navigate the menu...

[*] Joystick Support
(Atari-9-Pin) Joystick Type
Joystick Settings  --->

<Select>    < Exit >    < Help >
```

```
Joystick Settings
Arrow keys navigate the menu...

(PINB) Pin up
(Bit0) Bit up
(PINB) Pin down
(Bit1) Bit down
(PINB) Pin left
(Bit2) Bit left
(PINB) Pin right
(Bit3) Bit right
(PIND) Pin fire
(Bit3) Bit fire

<Select>    < Exit >    < Help >
```

- die hier gezeigten Werte sind die richtigen für den Borg 16

Borgware 2D: Menu Support, Games, Animations

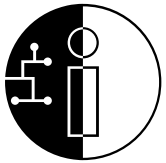
- `[*] Menu Support` ermöglicht die Spielauswahl mit dem Joystick
- `Games -->` beherbergt eben diese Spiele:



- `Animations -->` enhält im Untermenü eine Vielzahl von Animationen (siehe Impressionen)

Borgware 2D: CAN und RFM12

- `[ ] RFM12 support -->` und `[ ] CAN bus support -->` sind nur sinnvoll, wenn diese Komponenten auch verbaut sind
- die RFM12-Library verfügt zudem über keine Simulator-Stubs, so dass ein Simulator-Build fehlschlägt, wenn diese Option aktiv ist

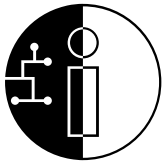


Speichern der Konfiguration

- wählt man im Hauptmenü `<Exit>`, so wird nach dem Speichern der Konfiguration gefragt

```
Do you wish to save your new Borg configuration?

                < Yes >        <  No  >
```

- dies bejahen wir, woraufhin eine *.config*-Datei (Build-System) und eine *autoconf.h*-Datei (Quellcode) erzeugt werden



Der erste Build

Ein make stößt den Build-Vorgang an:

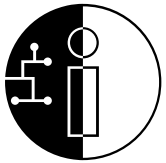
```
make

checking dependencies for pixel.c
checking dependencies for display_loop.c
...
==============================
image compiled for: atmega32
size is:

Program: 26320 bytes
(.text + .data)

Data:  618 bytes
(.data + .bss)
==============================
```

- hat alles geklappt liegt nun das Image *image.hex* im Intel-Hex-Format im *borgware-2d*-Verzeichnis

Gliederung

Voraussetzungen bei der Hardware

- die folgenden Schritte gehen davon aus, dass...
  - ...die Platine des Borg 16 vollständig aufgebaut ist
  - ...der USBasp am PC angeschlossen und eingerichtet ist
  - ...die ISP-Schnittstelle des Borg 16 mit dem USBasp verbunden ist
- die ISP-Leitungen des Borg 16 sind über einen 5×2-Header ausgeführt mit Pin 1 oben rechts (D-Sub-Anschlüssen zeigen in Südrichtung)
- bei erstmaliger Inbetriebnahme des ATMega muss der `SLOW`-Jumper (`J3`) des USBasp gesetzt sein



Einen jungfräulichen Mikrocontroller vorbereiten

Fuses setzen mit USBasp, avrdude und ISP-Schnittstelle

- der Mikrocontroller wird *einmalig* über sog. Fuses eingestellt, wobei wir zwischen *lfuse*, *hfuse* und bei ATmega644(P)/1284(P) auch *efuse* unterscheiden
- die genaue Bedeutung der einzelnen Bits der Fuses lässt sich aus dem Datenblatt des jeweiligen Mikrocontrollers entnehmen
- konkret stellen wir bei den ATmegas folgendes ein:
  - externes Quarz mit 16 MHz
  - EEPROM-Inhalte bleiben beim Flashen erhalten
  - Bootloader nutzen (optional)
  - Bootloadergröße 512 Words ($\widehat{=}$1024 Bytes)
  - serielle Programmierung über ISP zulassen
  - Brownout-Detection bei 4,3V

Keine Experimente an den Fuses!

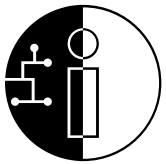
Vorsicht beim Ändern der Fuses! Eine falsche Einstellung kann den ATmega dauerhaft unbrauchbar machen!

Fuses setzen mit USBasp und avrdude

Passende Fuses für den Borg 16 (ohne Bootloader)

MCU	lfuse	hfuse	efuse	avrdude-Befehl
ATmega32	0x2f	0xc5	-	`avrdude -c usbasp -p m32 -U lfuse:w:0x2f:m \` `-U hfuse:w:0xc5:m`
ATmega644	0xe7	0xd7	0xfc	`avrdude -c usbasp -p m644 -U lfuse:w:0xe7:m \` `-U hfuse:w:0xd7:m -U efuse:w:0xfc:m`
ATmega644P	0xe7	0xd7	0xfc	`avrdude -c usbasp -p m644p -U lfuse:w:0xe7:m \` `-U hfuse:w:0xd7:m -U efuse:w:0xfc:m`
ATmega1284	0xe7	0xd7	0xfc	`avrdude -c usbasp -p m1284 -U lfuse:w:0xe7:m \` `-U hfuse:w:0xd7:m -U efuse:w:0xfc:m`
ATmega1284P	0xe7	0xd7	0xfc	`avrdude -c usbasp -p m1284p -U lfuse:w:0xe7:m \` `-U hfuse:w:0xd7:m -U efuse:w:0xfc:m`

- über den Fuse-Calculator[5] von *Engbedded* lassen sich die hexadezimalen Werte in sprechende Namen umwandeln
- die dort verwendeten Bezeichnungen kann man dann bequem im Datenblatt des jeweiligen Controllers nachschlagen



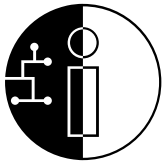
[5] http://www.engbedded.com/fusecalc

Fuses setzen mit USBasp und avrdude

Passende Fuses für den Borg 16 (mit Bootloader)

MCU	lfuse	hfuse	efuse	avrdude-Befehl
ATmega32	0x2f	0xc4	-	`avrdude -c usbasp -p m32 -U lfuse:w:0x2f:m \ -U hfuse:w:0xc4:m`
ATmega644	0xe7	0xd6	0xfc	`avrdude -c usbasp -p m644 -U lfuse:w:0xe7:m \ -U hfuse:w:0xd6:m -U efuse:w:0xfc:m`
ATmega644P	0xe7	0xd6	0xfc	`avrdude -c usbasp -p m644p -U lfuse:w:0xe7:m \ -U hfuse:w:0xd6:m -U efuse:w:0xfc:m`
ATmega1284	0xe7	0xd6	0xfc	`avrdude -c usbasp -p m1284 -U lfuse:w:0xe7:m \ -U hfuse:w:0xd6:m -U efuse:w:0xfc:m`
ATmega1284P	0xe7	0xd6	0xfc	`avrdude -c usbasp -p m1284p -U lfuse:w:0xe7:m \ -U hfuse:w:0xd6:m -U efuse:w:0xfc:m`

- *mit Bootloader* bedeutet, dass die letzten 512 Words (≙1024 Bytes) des Flash-ROMs für ein von der Borgware-2D unabhängiges Programm (dem *Bootloader*) reserviert sind
- die o.g. Fuse-Einstellungen sorgen dafür, dass der ATmega nach einem Reset den Bootloader anstatt die Borgware-2D startet



Den Borg 16 mit dem USBasp flashen

Programmierung über den ISP

Die *image.hex*-Datei mit avrdude und USBasp flashen

ATmega32	`avrdude -c usbasp -p m32 -U f:w:image.hex`
ATmega644	`avrdude -c usbasp -p m644 -U f:w:image.hex`
ATmega644P	`avrdude -c usbasp -p m644p -U f:w:image.hex`
ATmega1284	`avrdude -c usbasp -p m1284 -U f:w:image.hex`
ATmega1284p	`avrdude -c usbasp -p m1284p -U f:w:image.hex`

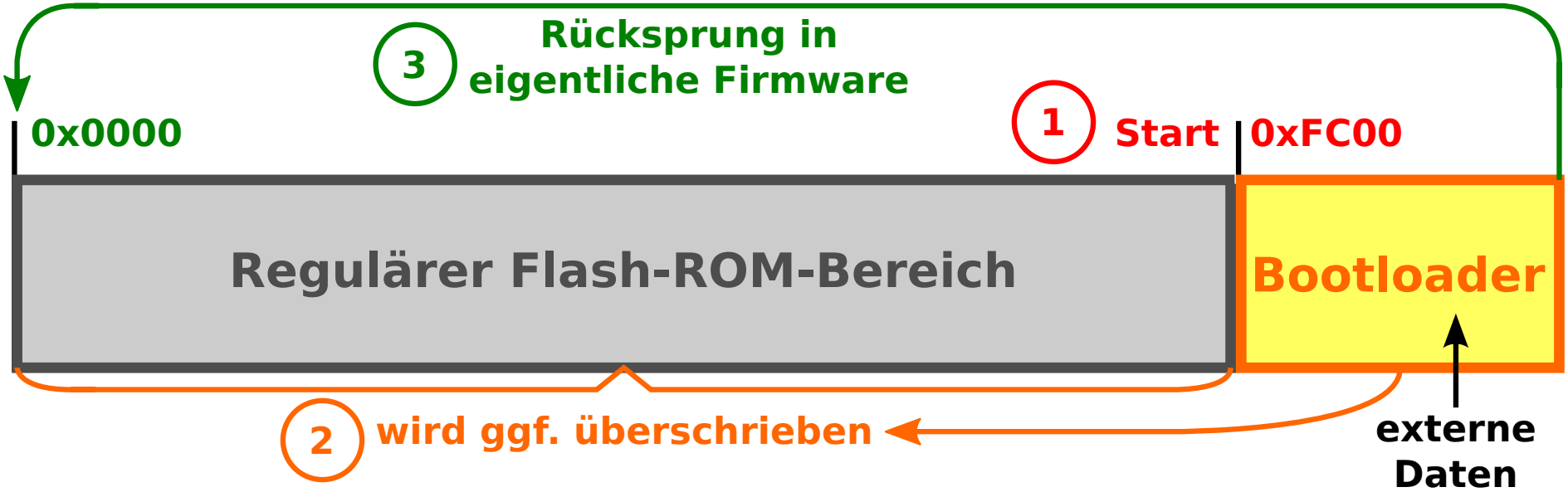
Dies weißt avrdude an, als Programmer `-c` einen USBasp zu verwenden, als Target `-p` einen ATmega32/644(P)/1284(P) anzunehmen und eine Speicheroperation `-U` im Flash-ROM `f` durchzuführen, und zwar das Schreiben `w` des Inhalts von `image.hex`.

Bootloader

Wird die Borgware 2D direkt mittels ISP auf den ATmega übertragen, geht ein evtl. zuvor aufgespielter Bootloader verloren. Dementsprechend muss der Bootloader in den Fuses ggf. deaktiviert werden.

Bootloader

- anstatt mit einem dediziertem Programmiergerät wie dem USBasp kann sich ein AVR auch „selber" flashen
- der hintere Teil des Flash-ROMs kann per Fuse für einen Bootloader reserviert werden, der nach einem Reset direkt gestartet wird
- dieser kann bei Bedarf die eigentliche Firmware von anderen Quellen wie der seriellen Schnittstelle nachladen und vorne abspeichern



Serieller Bootloader für den Borg16

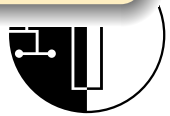
- in der AVR109 Application Note[6] beschreibt Atmel ein Protokoll zum Flashen von AVR-Controllern über die serielle Schnittstelle
- dieses Protokoll wird von vielen AVR-zentrierten Werkzeugen (u.a. von `avrdude`) unterstützt
- wir verwenden `AVRProg Boot 0.85` von Martin Thomas et al.[7]
- den vorgenannten genannten Bootloader[8] herunterladen, entpacken und in das Verzeichnis wechseln

[6] http://www.atmel.com/Images/doc1644.pdf
[7] http://www.siwawi.arubi.uni-kl.de/avr_projects/index.html#avrprog_boot
[8] http://www.siwawi.arubi.uni-kl.de/avr_projects/avrprog_boot_v0_85_20081203.zip

AVRProg Boot konfigurieren, bauen und flashen

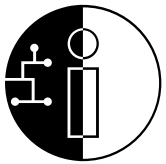
AVRProg Boot

- im *makefile* die Variable `MCU` auf `atmega32` setzen (für ATmega644/P entsprechend adaptieren) und `BOOTSIZE` auf `512` ändern
- in der *main.c* müssen folgende `#defines` geändert werden:
  ```
  #define F_CPU 16000000
  #define BAUDRATE 19200
  // #define START_SIMPLE auskommentieren!!
  #define START_WAIT
  ```
- anschließend kann der Bootloader mit `make` gebaut werden
- mittels avrdude den Bootloader flashen (dieses Mal noch über ISP) und ggf. Fuses setzen
  ```
  avrdude -c usbasp -p m32 -U f:w:main.hex
  ```



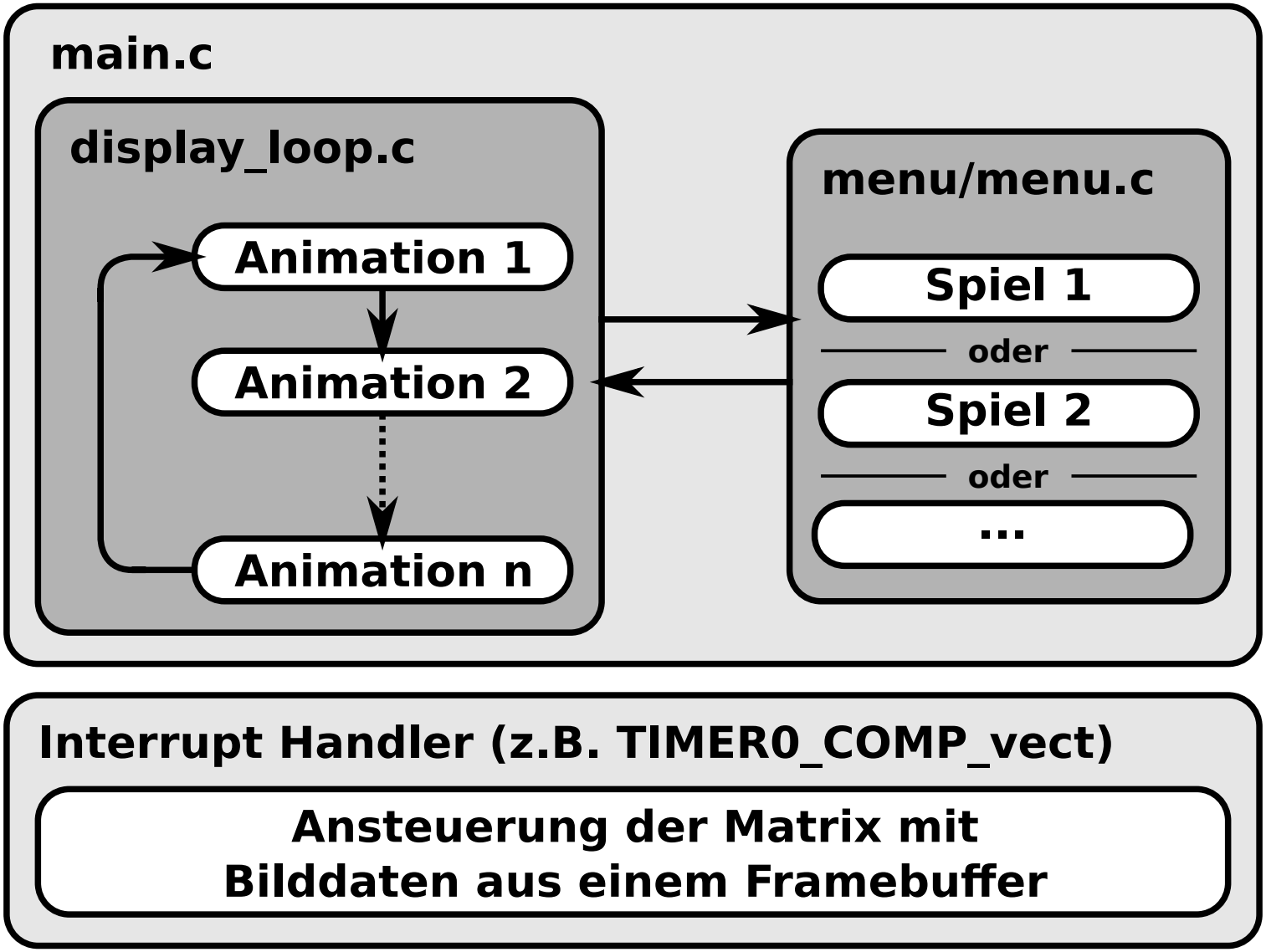
Flashen über die serielle Schnittstelle

- der ATmega enthält nun lediglich den Bootloader und die Borgware 2D muss jetzt seriell überspielt werden
- dazu ist der Borg 16 mit einem seriellen Kabel oder einem USB-Seriell-Adapter mit dem PC zu verbinden
- `avrdude` im *borgware-2d*-Verzeichnis folgendermaßen aufrufen:
  ```
  avrdude -c avr109 -b 19200 -P /dev/ttyS0|COM1 \
  -p m32|m644|m644p -U f:w:image.hex
  ```
- die grau hinterlegten Angaben sind der Situation entsprechend anzupassen
- wenn man den Borg 16 seriell flashen möchte, den Resetknopf 1-2 Sekunden drücken, *unmittelbar* danach obiges avrdude-Kommando absetzen, dann sollte der Flash-Vorgang einsetzen
- es kann ein paar Versuche kosten, bis es klappt



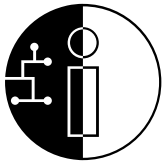
Gliederung

Grober Ablauf in der Borgware 2D



Wichtige Anlaufpunkte

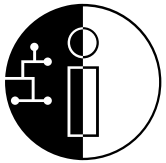
- *src/config.h* enthält (indirekt über *src/autoconf.h*) wichtige Parameter wie Abmessungen und Farbtiefe der Matrix
  - `NUMPLANES` – die Anzahl der Helligkeitsstufen
  - `NUM_ROWS` – die Anzahl der Zeilen
  - `NUM_COLS` – die Anzahl der Spalten
- *src/main.c* ist der Einstiegspunkt, der die Hardware initialisiert und anschließend eine Endlosschleife startet
- diese ist in *src/display_loop.c*, und geht alle konfigurierten Animationen durch



Die Pixel-API in *src/pixel.h*

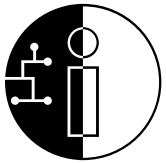
In *src/pixel.h* befinden sich folgende Schnittstellen:

- der Typ `pixel`, eine Struktur mit den Attributen `x` und `y`
- `(pixel){0, 0}` ist hierbei RECHTS OBEN
- die Funktion `setpixel(pixel p, unsigned char c)`, wobei `c` ein Helligkeitswert von 0 bis 3 sein muss
- für die meisten Dinge ist `setpixel(...)` vollkommen ausreichend
- wer mehr Geschwindigkeit braucht, kann direkt in den *Framebuffer* schreiben

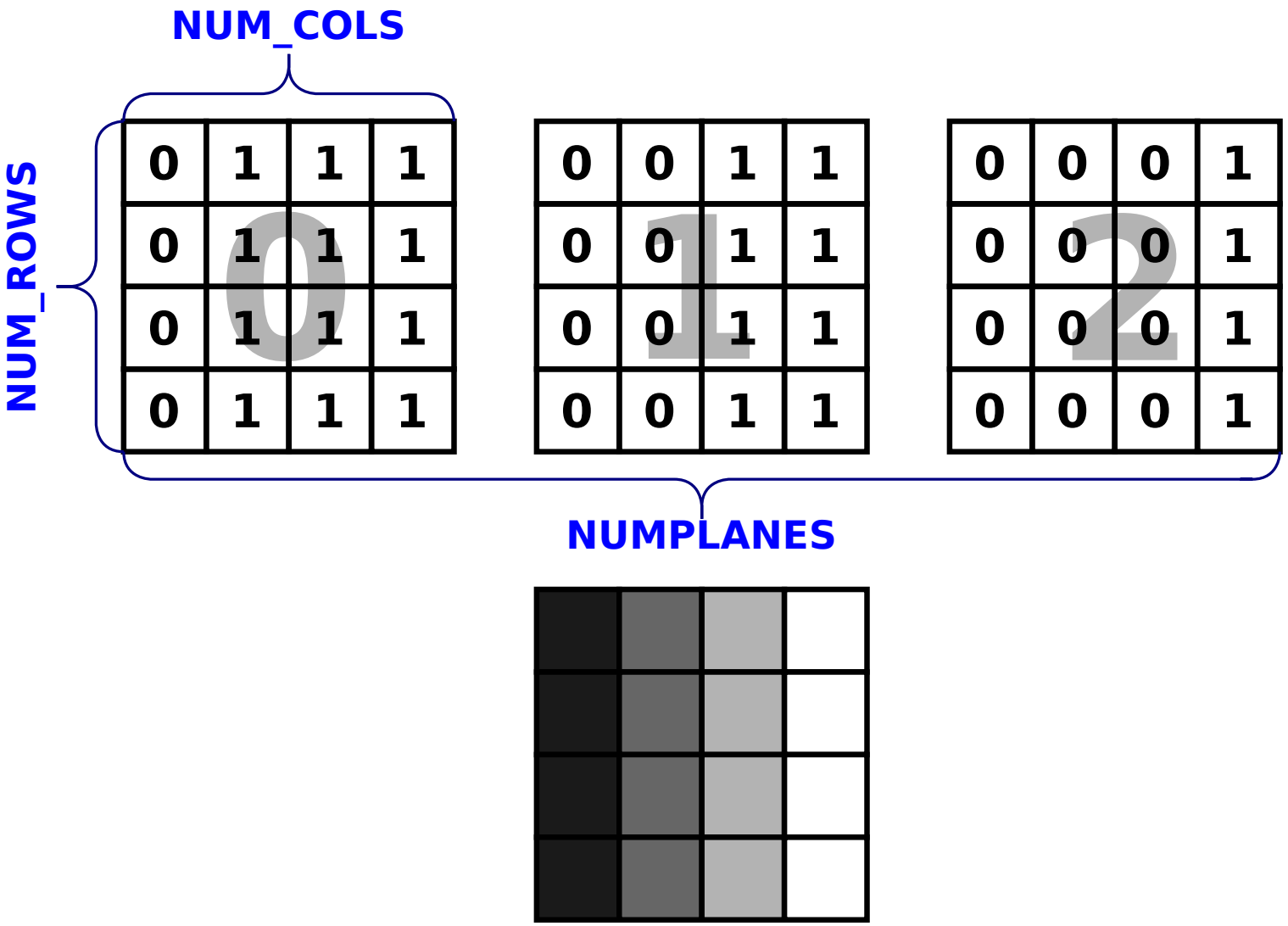


Der Framebuffer

- `unsigned char pixmap[NUMPLANE][NUM_ROWS][LINEBYTES]` enthält den Framebuffer
- `LINEBYTES` entepricht dabei `NUM_COLS / 8` (immer aufgerundet)
- ein Pixel verteilt sich über drei Planes, wo er jeweils durch ein Bit repräsentiert wird
- in je mehr Planes das jeweils ihn repräsentierende Bit gesetzt ist, desto heller wird er dargestellt
- technisch ist es so, dass die entsprechende LED länger leuchtet, je öfter das Bit gesetzt ist

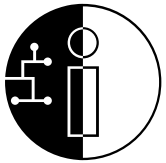


Der Framebuffer anschaulich



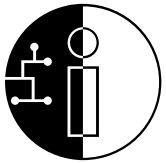
Nützliche Funktionen

- in *src/util.h* gibt es die `wait(int ms)`-Funktion, die `ms` Millisekunden wartet
- eine Animation sollte immer ein `wait(x)` enthalten, da dort ein Hook enthalten ist, der u.a. den Menüaufruf ermöglicht
- in *src/random/prng.h* befindet sich der Zufallsgenerator, der mittels `random8()` eine zufällige Zahl zwischen 0 und 255 zurückgibt
- *src/scrolltext/scrolltext.h* enthält die `scrolltext(char *str)`-Funktion, die einen Lauftext ausgibt
- nicht vergessen: ein String muss mit einem Kommando wie „`</#`“ eingeleitet werden



Joystick-Abfrage in *src/joystick/joystick.h*

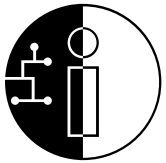
- um den Joystick abzufragen gibt es 5 Makros
  - `JOYISUP`
  - `JOYISDOWN`
  - `JOYISLEFT`
  - `JOYISRIGHT`
  - `JOYISFIRE`
- diese Makros liefern einen Wert ungleich 0, sobald die entsprechende Richtung gedrückt wird
- wenn man den Joystick in einem sehr kurzen Zeitintervall abfragt, muss man das Nachprellen der Kontakte berücksichtigen (siehe *src/games/tetris/input.c*



Gliederung

Vorgehensweise zum Einbinden einer Animation

- ein kompilierfähiges Grundgerüst in Form einer *.c*- und *.h*-Datei im Verzeichnis *src/animations* anlegen
- das Konfigrationsmenü über die neue Animation in Kenntnis setzen, d.h. die Datei *src/animations/config.in* anpassen
- das Build-System ebenfalls in Kenntnis setzen und die Datei *src/animations/Makefile* ergänzen
- den eingangs erwähnten Header in die *src/display_loop.c* inkludieren
- im großem switch/case-Block der `display_loop()`-Funktion eine noch nicht vergebene Nummer suchen
- anhand dieser Nummer einen neuen `case` anfügen und dort die Animation aufrufen

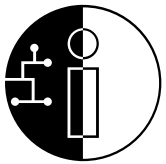


Die Header-Datei *src/animations/myanim.h*

```
#ifndef MYANIM_H_
#define MYANIM_H_

void myanim(void);

#endif /* MYANIM_H_ */
```

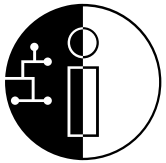
Die C-Datei *src/animations/myanim.c*

```
#include "../config.h"
#include "../pixel.h"
#include "../util.h"
#include "../random/prng.h"
#include "myanim.h"

void myanim(void) {
    unsigned char startcolor = 0, color;

    unsigned int const times = 100 + (random8() % 100);
    for (unsigned int t = 0; t < times; ++t) {
        for (unsigned char y = 0; y < NUM_ROWS; ++y) {
            color = startcolor;
            for (unsigned char x = 0; x < NUM_COLS; ++x) {
                setpixel((pixel){x, y}, color);
                color = (color + 1) % (NUMPLANE + 1);
            }
        }
        startcolor = (startcolor + 1) % (NUMPLANE + 1);
        wait(150);
    }
}
```



Menükonfiguration anpassen

- in den *src/animations/config.in* sollte vorzugsweise am Ende folgendes einfügt werden:
  `bool „MyAnim" ANIMATION_MYANIM $RANDOM_SUPPORT`
- dies erzeugt eine Checkbox *MyAnim* im Animations-Untermenü
- und erstellt ein `#define ANIMATION_MYANIM` und eine gleichlautende Env-Variable für Make
- und wir setzen voraus, dass der Zufallsgenerator konfiguriert sein muss

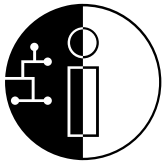


Makefile für Animation anpassen

- in *src/animations/Makefile* folgendes ergänzen:

```
ifeq ($(ANIMATION_MYANIM),y)
  SRC += myanim.c
endif
```

- hier ist die Env-Variable zu nehmen, die man in der *src/animations/config.in*-Datei festgelegt hat



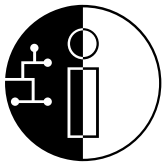
Anpassen der *src/display_loop.c*

- am Anfang der *src/display_loop.c* den Header einbinden:

```
#include "myanim.h"
```

- freie Nummer (hier z.B. 26) im switch/case-Block finden und einen `case` für diese Nummer anlegen, der die neue Animation aufruft, diesen zusätzlich mit dem im *src/animations/config.in* definierten `#define` konditional einbinden:

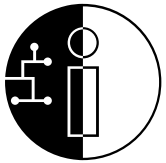
```
#ifdef ANIMATION_MYANIM
    case 26:
        myanim();
        break;
#endif
```



Gliederung

Vorgehensweise zum Einbinden eines Spiels

- ein neues Verzeichnis (z.B. *mygame*) unterhalb von *src/games* erstellen
- ein kompilierfähiges Grundgerüst in Form einer *.c*- und *.h*-Datei im Verzeichnis *src/games/mygame* anlegen
- das Konfigrationsmenü über das neue Spiel in Kenntnis setzen, d.h. die Datei *src/games/config.in* anpassen
- das Build-System ebenfalls in Kenntnis setzen und die Datei *src/games/config.mk* ergänzen sowie in *src/games/mygame* ein neues *Makefile* anlegen
- einen Menü-Datensatz erstellen

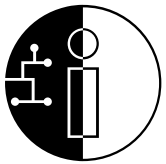


Die Header-Datei *src/games/mygame/mygame.h*

```
#ifndef MYGAME_H_
#define MYGAME_H_

void mygame(void);

#endif /* MYGAME_H_ */
```

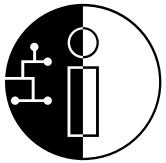
Die C-Datei *src/games/mygame/mygame.c*

```c
#include "../../config.h"
#include "../../pixel.h"
#include "../../util.h"
#include "../../compat/pgmspace.h"
#include "../../menu/menu.h"
#include "../../joystick/joystick.h"
#include "../../scrolltext/scrolltext.h"
#include "mygame.h"

void mygame(void) {
    pixel hero = {NUM_COLS / 2, NUM_ROWS / 2};
    old = {0,0};
    clear_screen(0);

    while (1) {
        if (old.x != hero.x || old.y != hero.y) {
            setpixel(old, 0);
            old = hero;
        }
        setpixel(hero, NUMPLANE);

        if (JOYISDOWN)
            ++hero.y;
        else if (JOYISUP)
            --hero.y;
        else if (JOYISLEFT)
            ++hero.x;
        else if (JOYISRIGHT)
            --hero.x;

        /* Hero, don't touch the border! */
        if (hero.x == 0 || hero.x == (NUM_COLS-1) ||
            hero.y == 0 || hero.y == (NUM_ROWS-1))
            break;

        wait(100);
    }
    scrolltext("</#Game Over!");
}
```



Menükonfiguration anpassen

- in den *src/games/config.in* sollte vorzugsweise am Ende folgendes einfügt werden:
  `bool "MyGame" GAME_MYGAME $JOYSTICK_SUPPORT`
- dies erzeugt eine Checkbox *MyGame* im Games-Untermenü
- und erstellt ein `#define GAME_MYGAME` und eine gleichlautende Env-Variable für Make
- und wir setzen voraus, dass der Joystick-Support konfiguriert sein muss

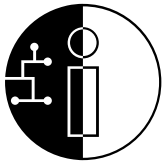


Makefile für die Spiele anpassen

- in *src/games/games.mk* folgendes ergänzen:

```
ifeq ($(GAME_MYGAME),y)
SUBDIRS += games/mygame
endif
```

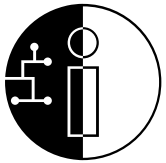
- hier ist einfach das neu erstellte Verzeichnis zu `$SUBDIRS` hinzuzufügen



Neues Makefile in games/mygame erstellen

- *src/games/mygame/Makefile* anlegen und folgendes eintragen:

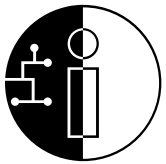
```
MAKETOPDIR = ../../..
TARGET =

include $(MAKETOPDIR)/defaults.mk

SRC = mygame.c

include $(MAKETOPDIR)/rules.mk
include $(MAKETOPDIR)/depend.mk
```



Einbindung in das In-Game-Menü

- innerhalb der der Quelldatei des Spiels (typischerweise nach den Includes) folgende globale Variable definieren:

```
#if defined MENU_SUPPORT && defined GAME_MYGAME
// mygame icon (MSB is leftmost pixel)
static const uint8_t icon[8] PROGMEM =
    {0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa};

game_descriptor_t mygame_game_descriptor
    __attribute__((section(".game_descriptors"))) =
{
    &mygame,
    icon,
};
#endif
```

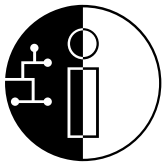


Einbindung in das In-Game-Menü

- `icon` definiert dabei das Icon in den Abmessungen 8×8 als Array von 8 `uint8_t`-Elementen
- ein Element steht für eine komplette Zeile des Icons
- das Bitmuster des Werts entspricht der grafischen Repräsentation der Zeile (MSB links außen, LSB rechts außen)
- die `game_descriptor_t`-Struktur enthält einen Pointer auf dieses Icon und einen Function-Pointer auf die Funktion, die das Spiel startet
- diese Funktion darf keine Argument erwarten
- die kryptischen Attribute sind wichtig, damit die Datenstrukturen beim Linken in die passenden Speicherbereiche verfrachtet werden

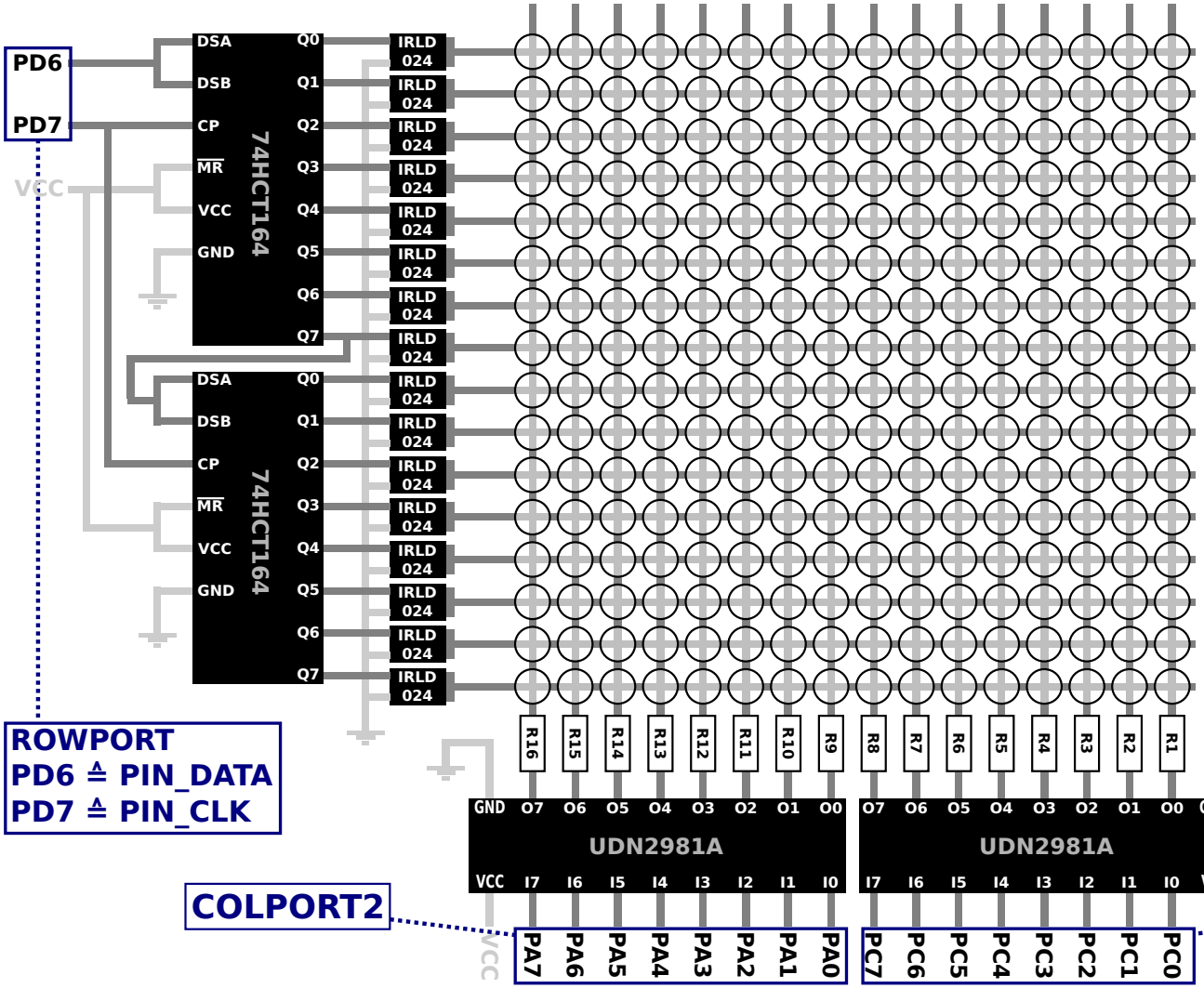
Die *fertige* C-Datei *src/games/mygame/mygame.c*

```c
#include "../../config.h"
#include "../../pixel.h"
#include "../../util.h"
#include "../../compat/pgmspace.h"
#include "../../menu/menu.h"
#include "../../joystick/joystick.h"
#include "../../scrolltext/scrolltext.h"
#include "mygame.h"

#if defined MENU_SUPPORT && defined GAME_MYGAME
// mygame icon (MSB is leftmost pixel)
static const uint8_t icon[8] PROGMEM =
    {0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa, 0x55, 0xaa};

game_descriptor_t mygame_game_descriptor
      __attribute__((section(".game_descriptors"))) =
{
    &mygame,
    icon,
};
#endif

void mygame(void) {
    pixel hero = {NUM_COLS / 2, NUM_ROWS / 2};
    old = {0,0};
    clear_screen(0);
    while (1) {
        if (old.x != hero.x || old.y != hero.y) {
            setpixel(old, 0);
            old = hero;
        }
        setpixel(hero, NUMPLANE);

        if (JOYISDOWN)
            ++hero.y;
        else if (JOYISUP)
            --hero.y;
        else if (JOYISLEFT)
            ++hero.x;
        else if (JOYISRIGHT)
            --hero.x;

        /* Hero, don't touch the border! */
        if (hero.x == 0 || hero.x == (NUM_COLS-1) ||
            hero.y == 0 || hero.y == (NUM_ROWS-1))
            break;

        wait(100);
    }
    scrolltext("</#Game Over!");
}
```



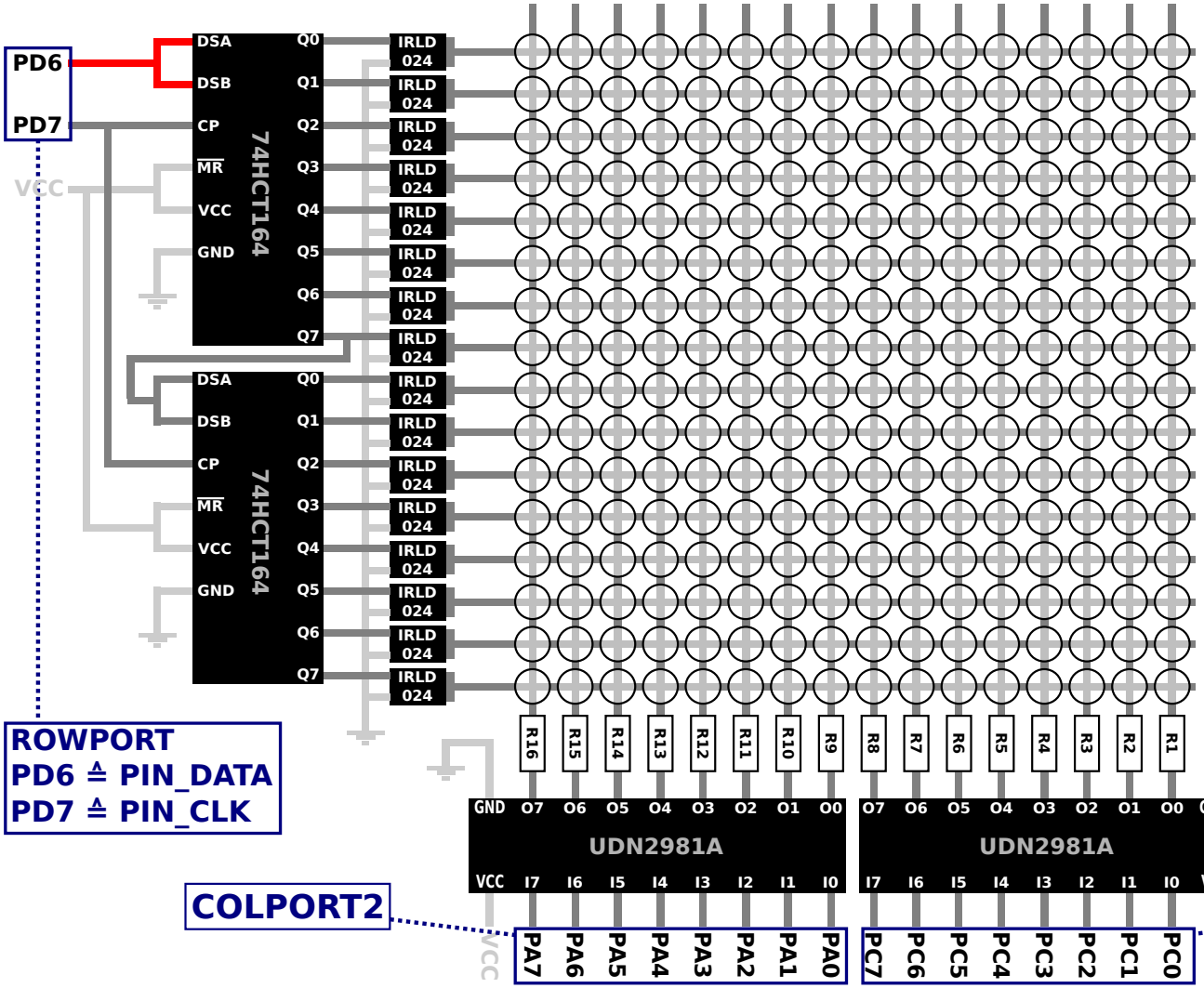
Gliederung

Die LED-Matrix



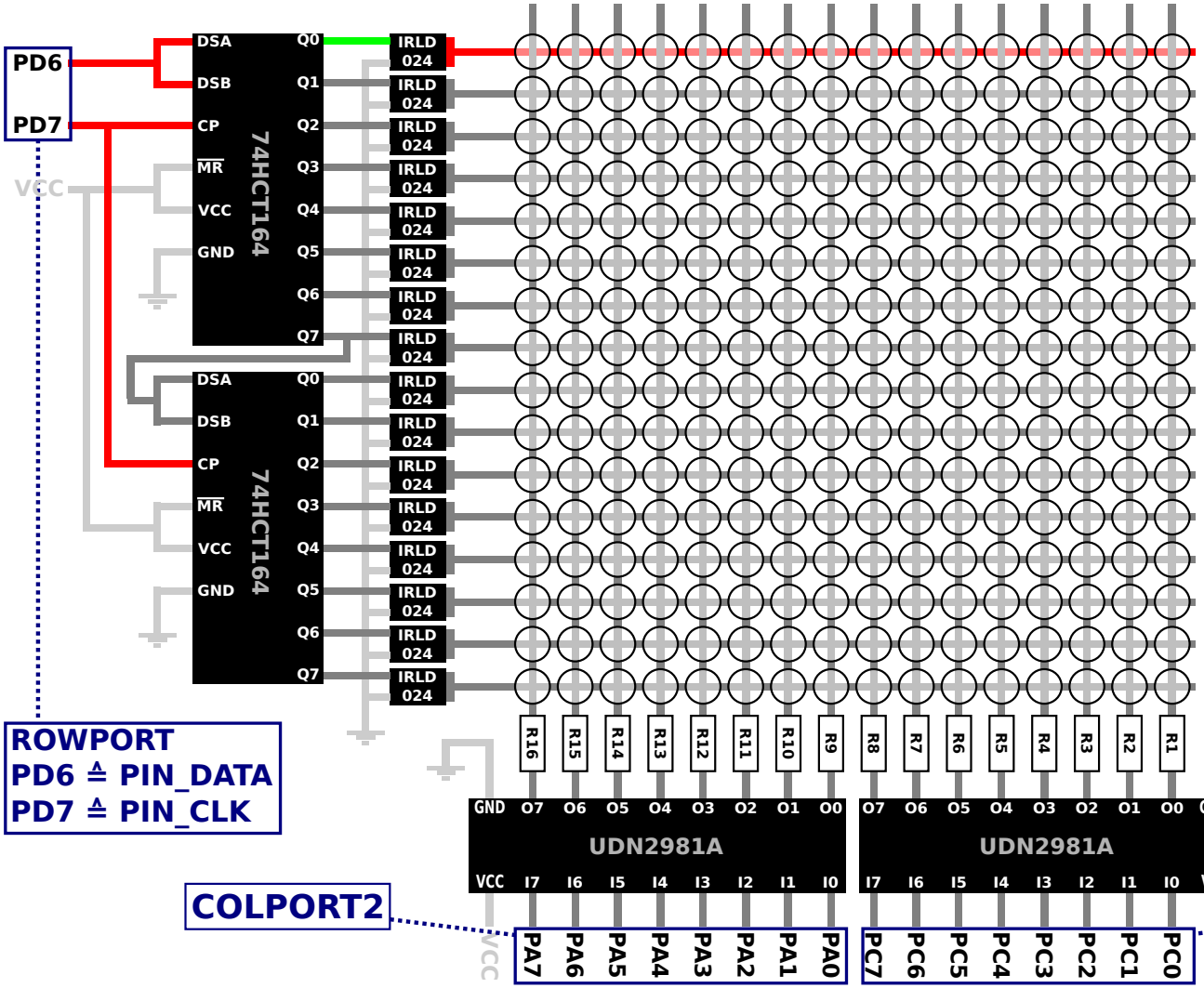

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 0
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



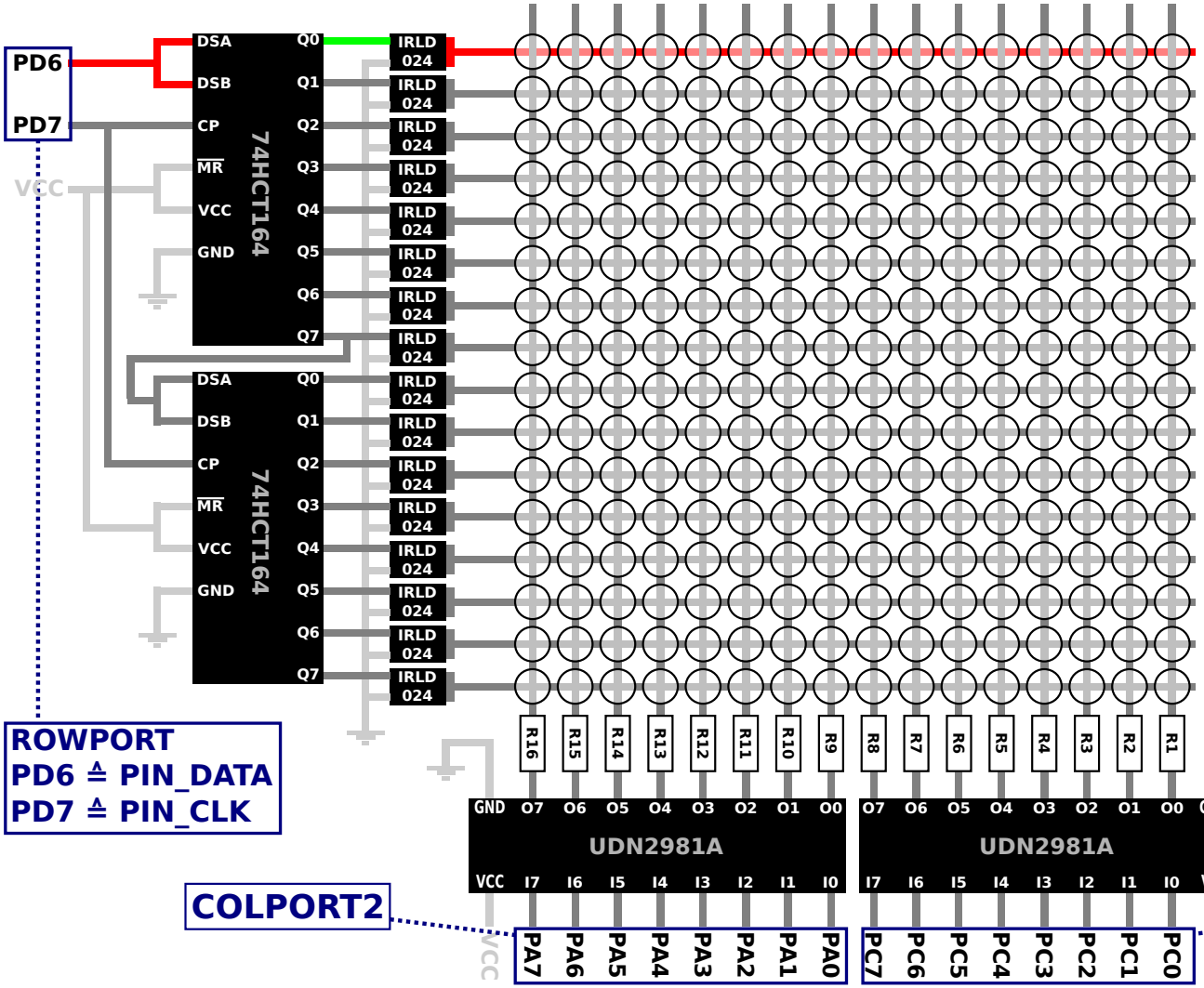

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 0
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



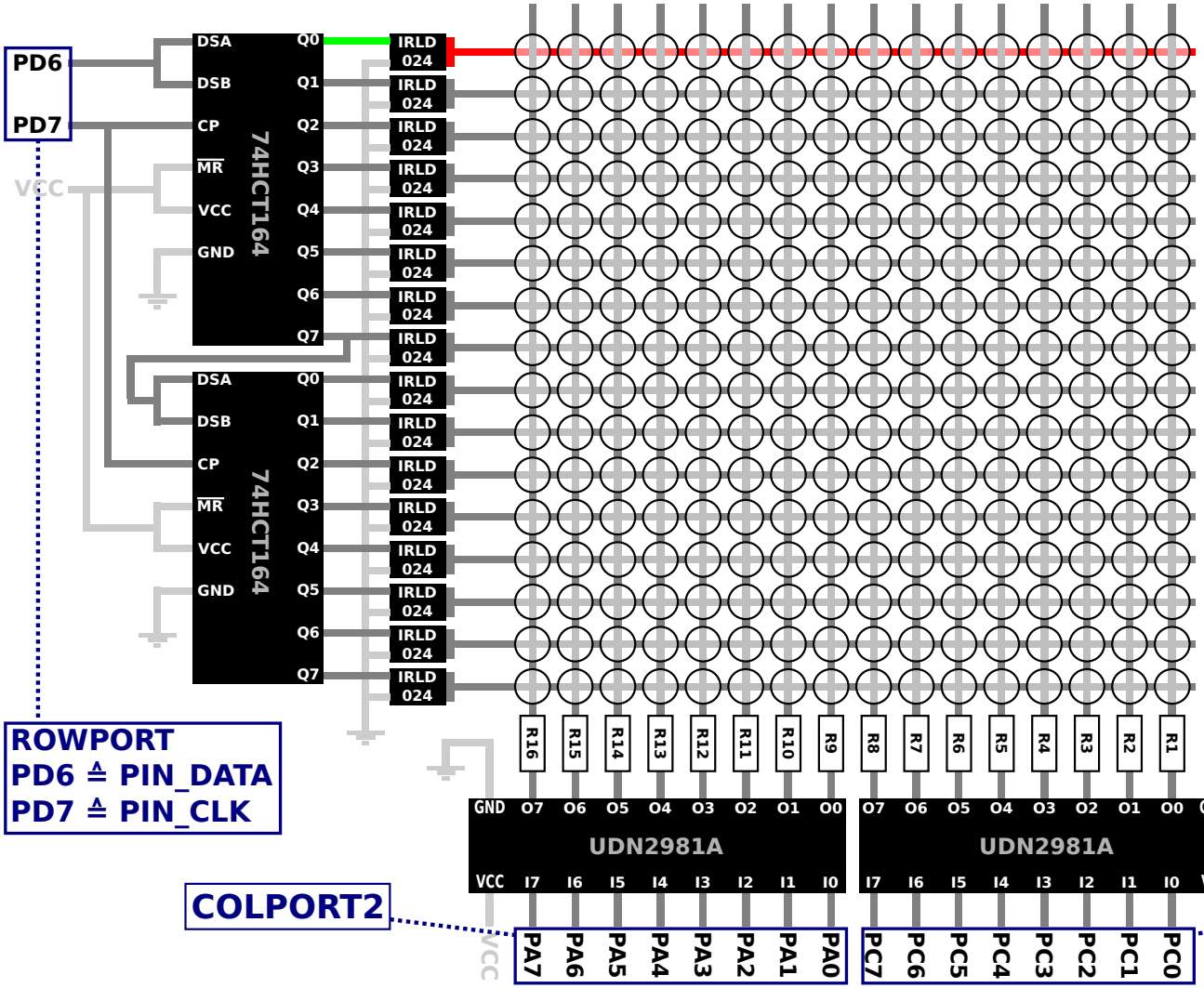

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 0
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



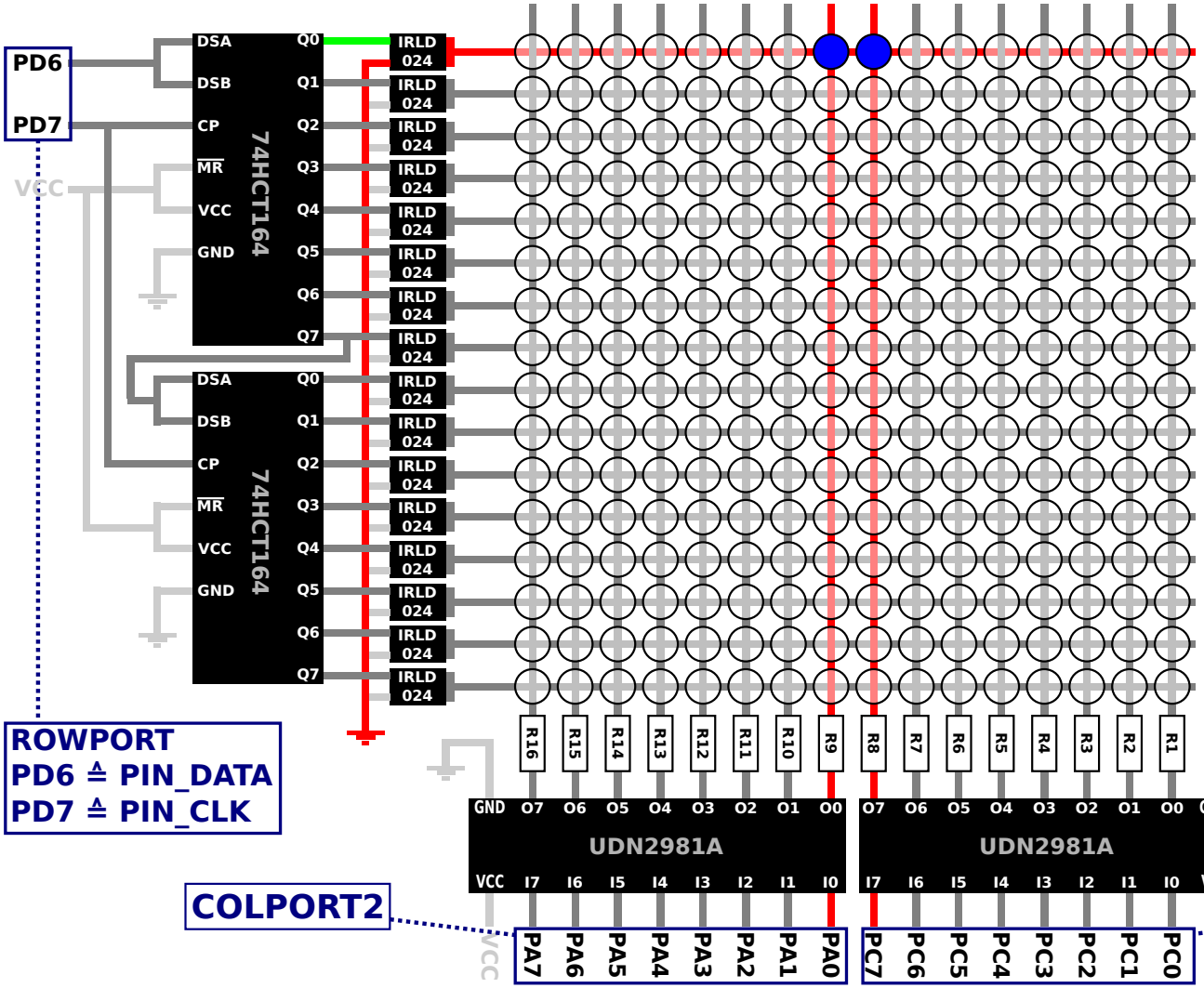

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 0
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



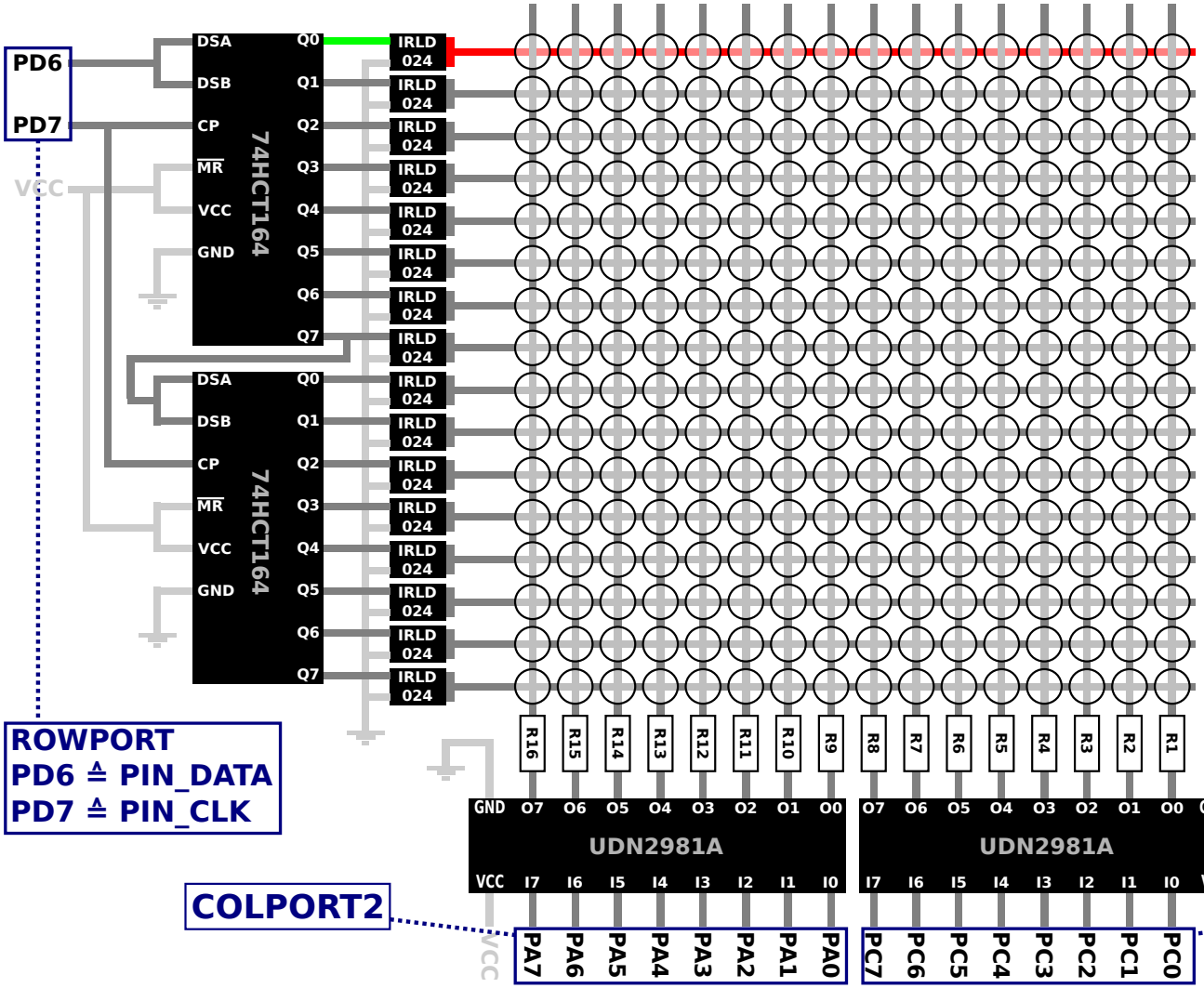

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 0
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



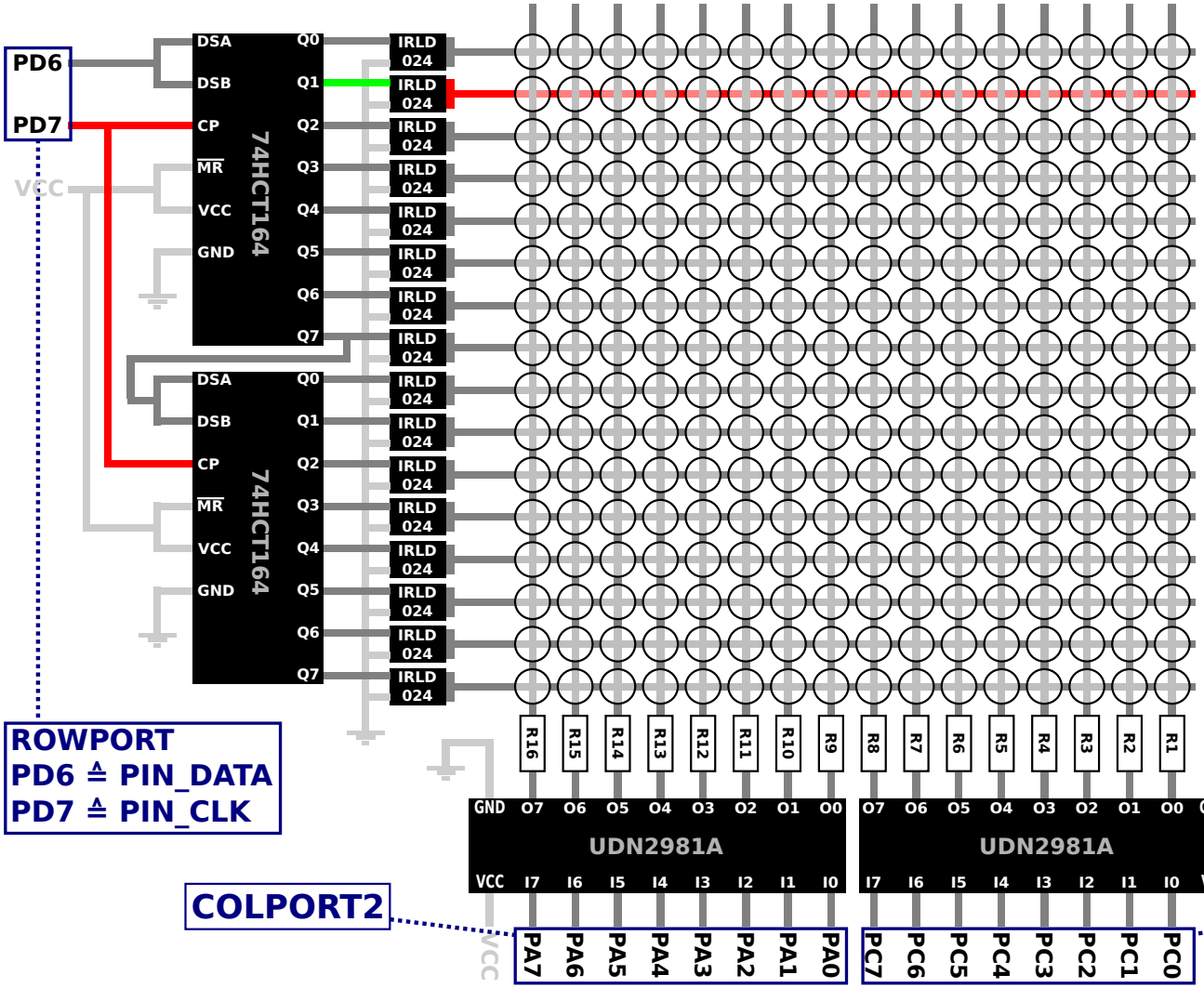

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 1
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



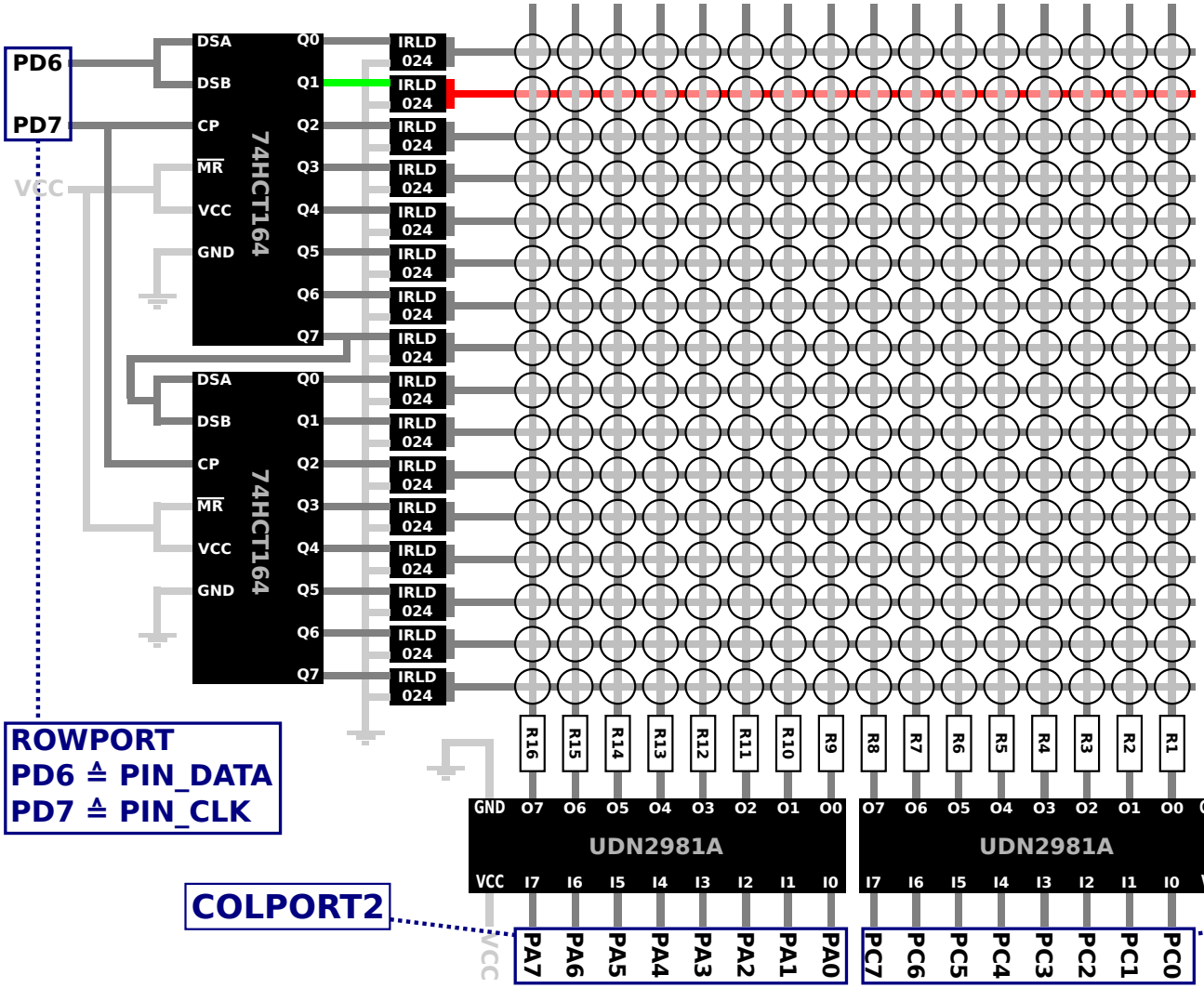

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 1
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



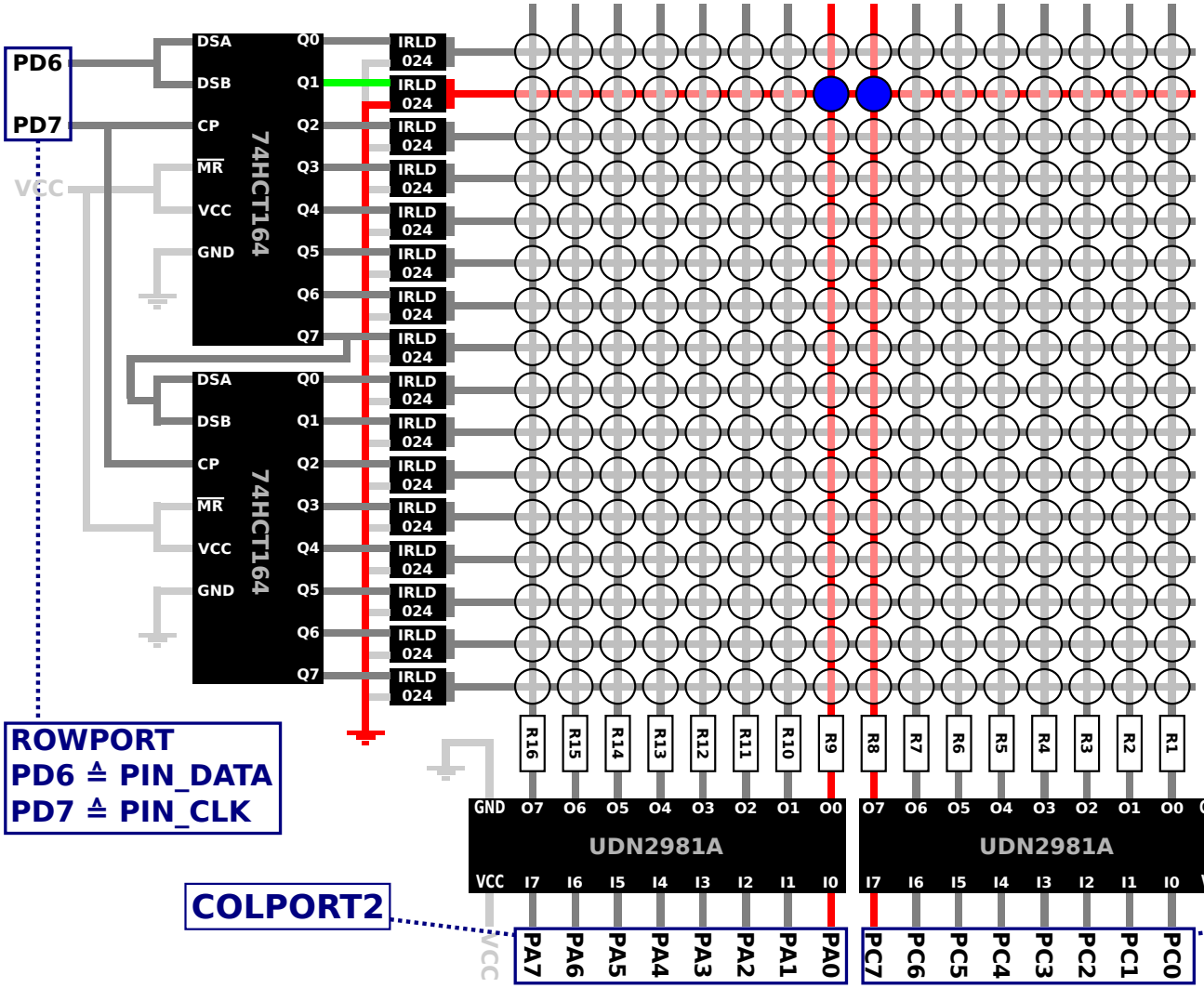

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 1
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



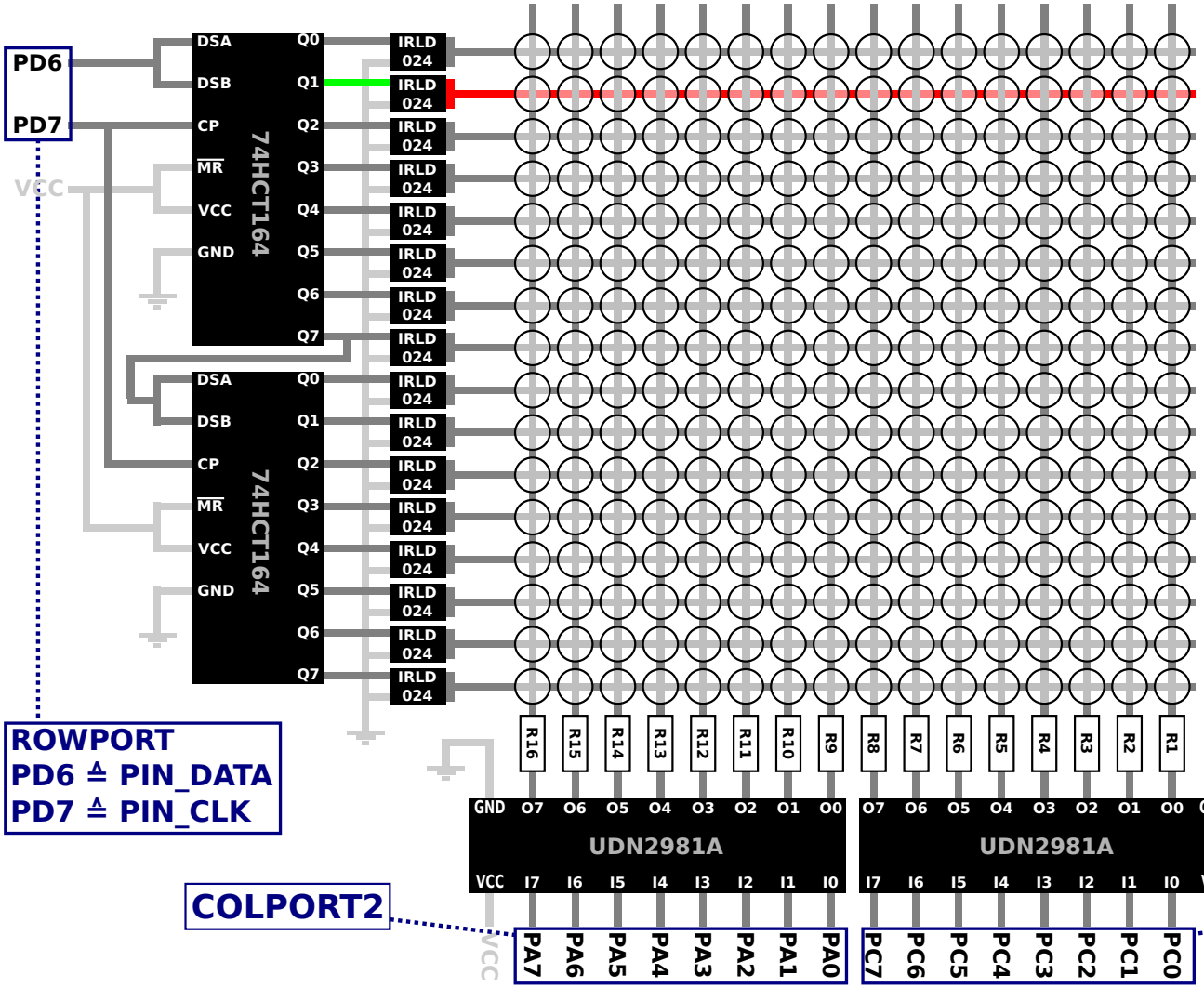

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 1
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



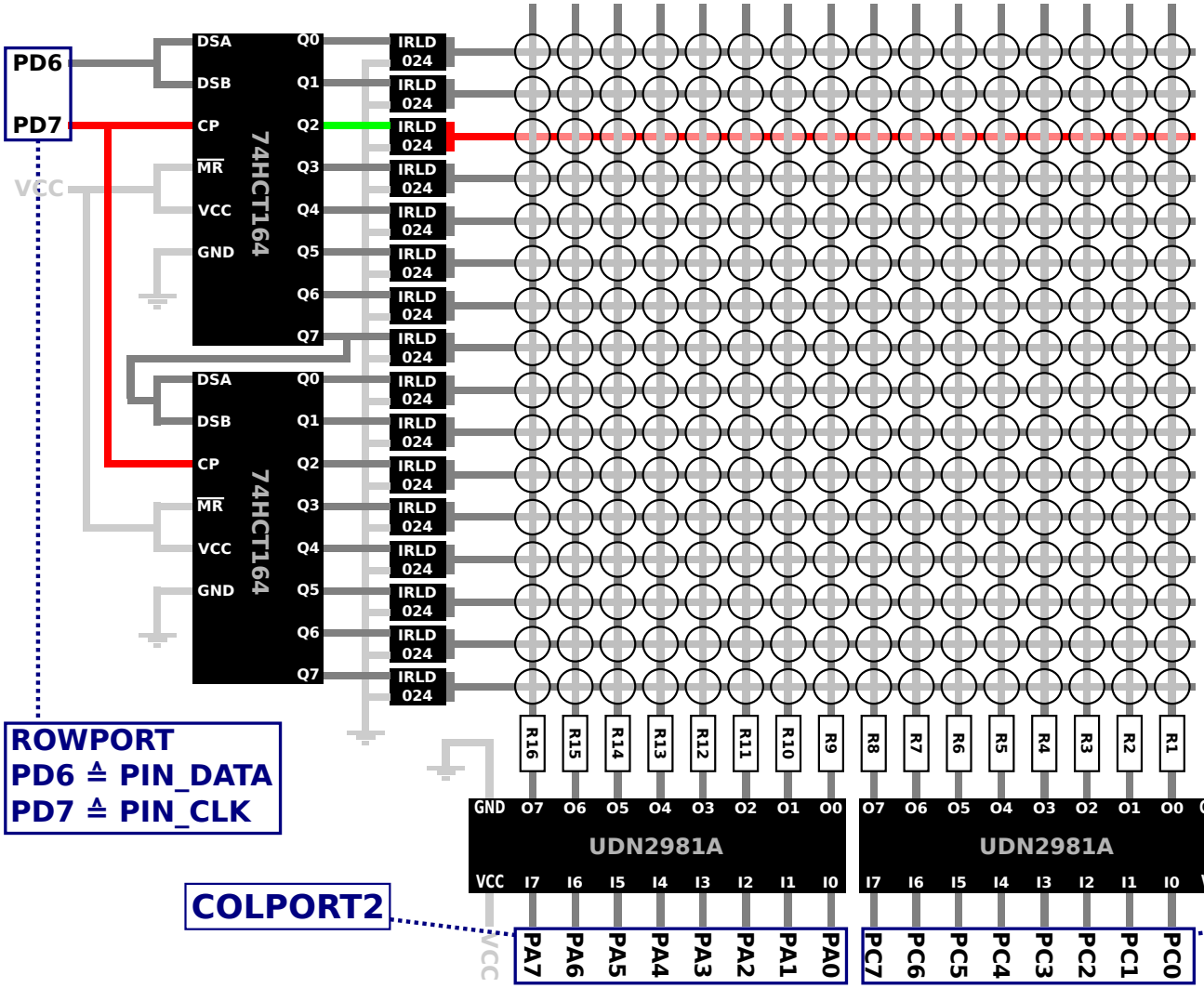

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 1
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x01  CP1: 0x80
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



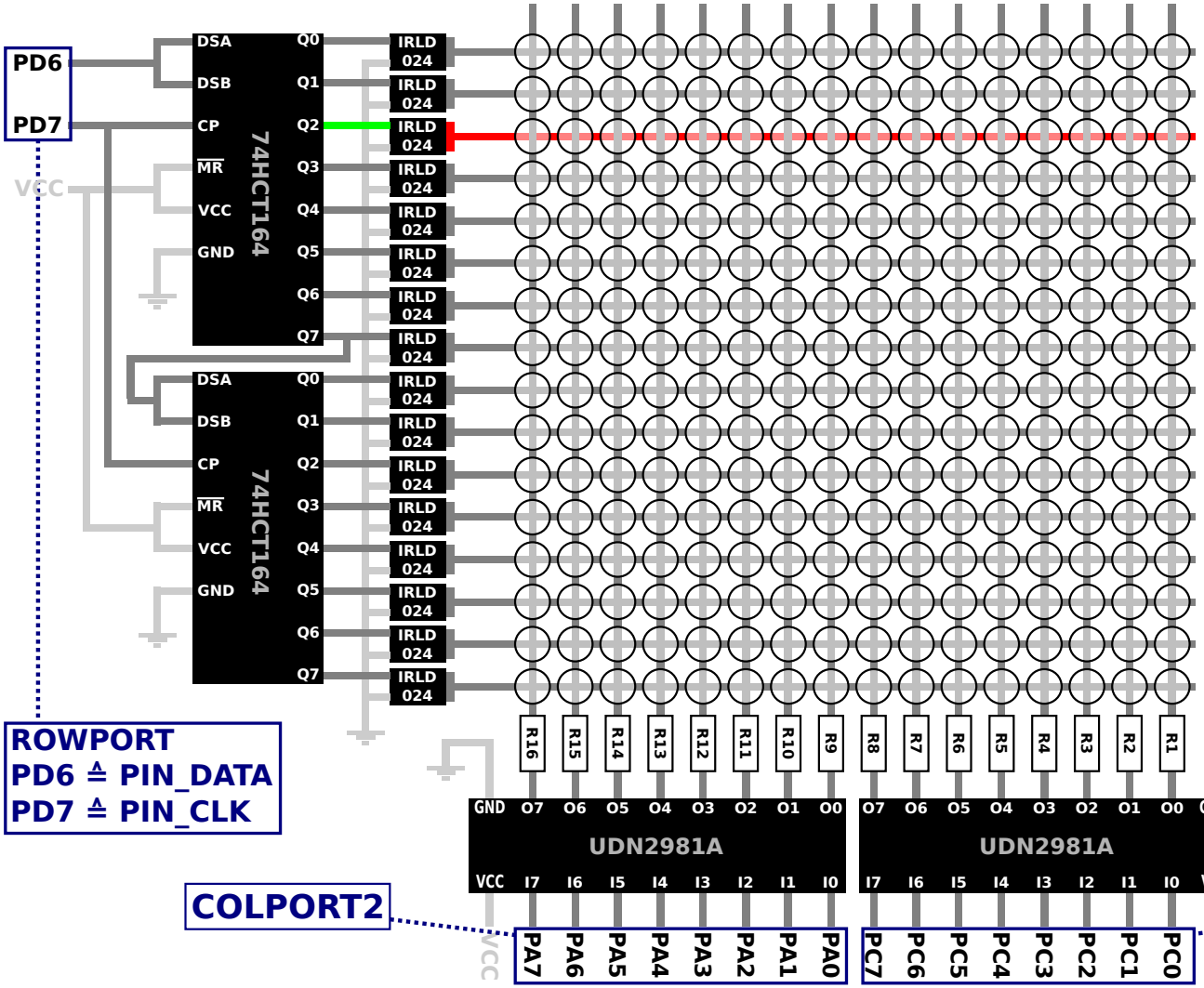

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 2
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x03  CP1: 0xC0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



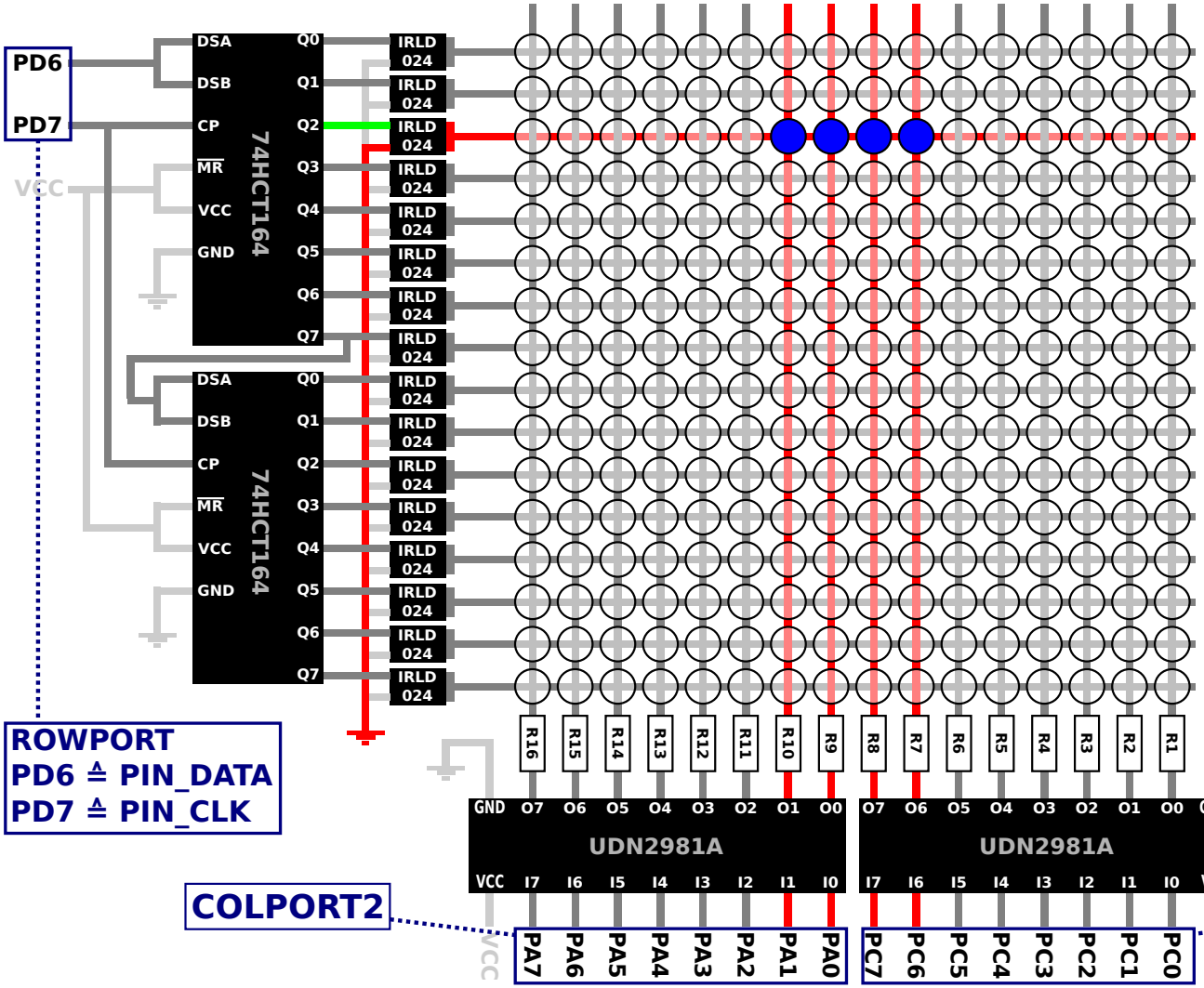

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 2
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x03  CP1: 0xC0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



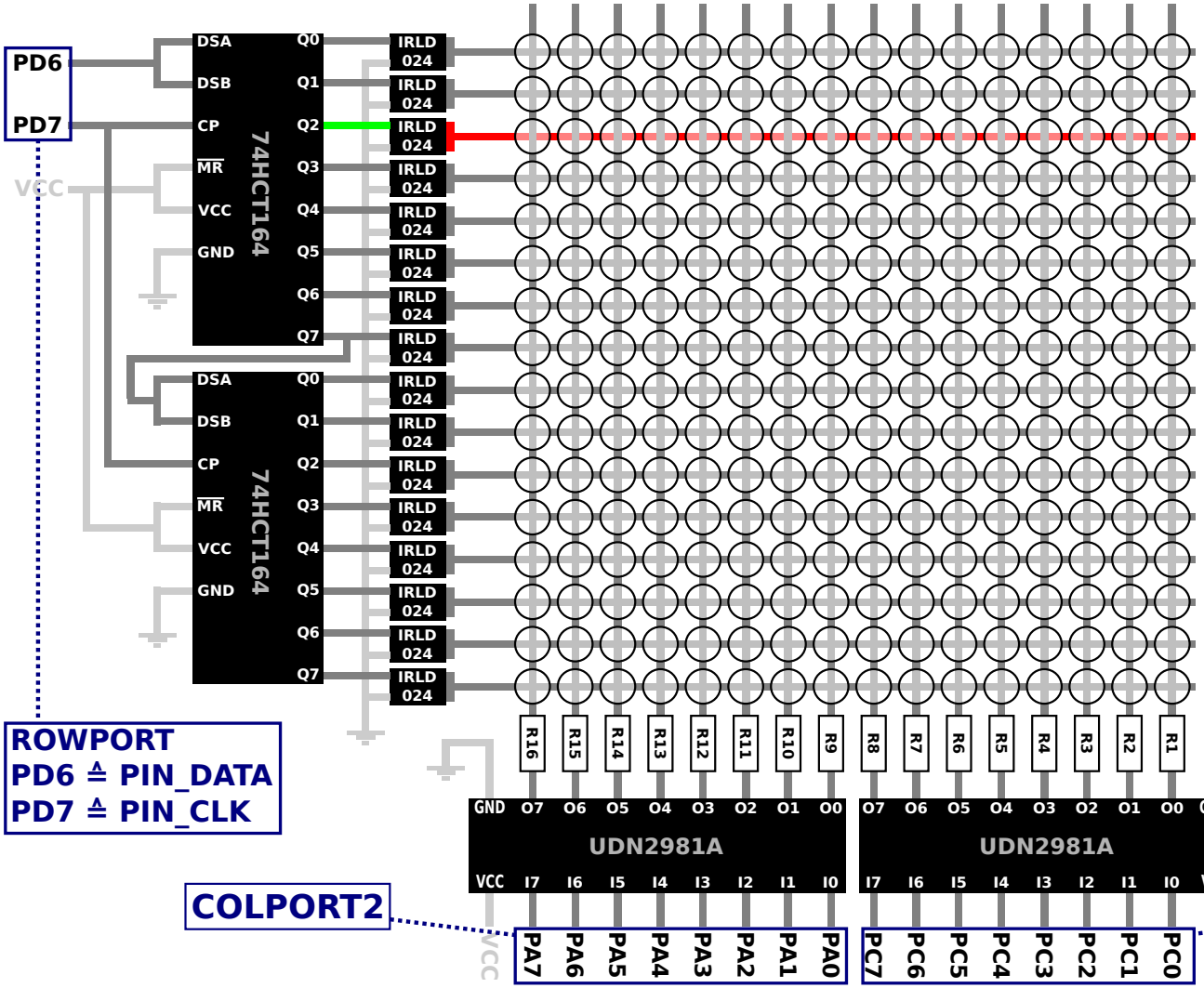

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 2
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x03  CP1: 0xC0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



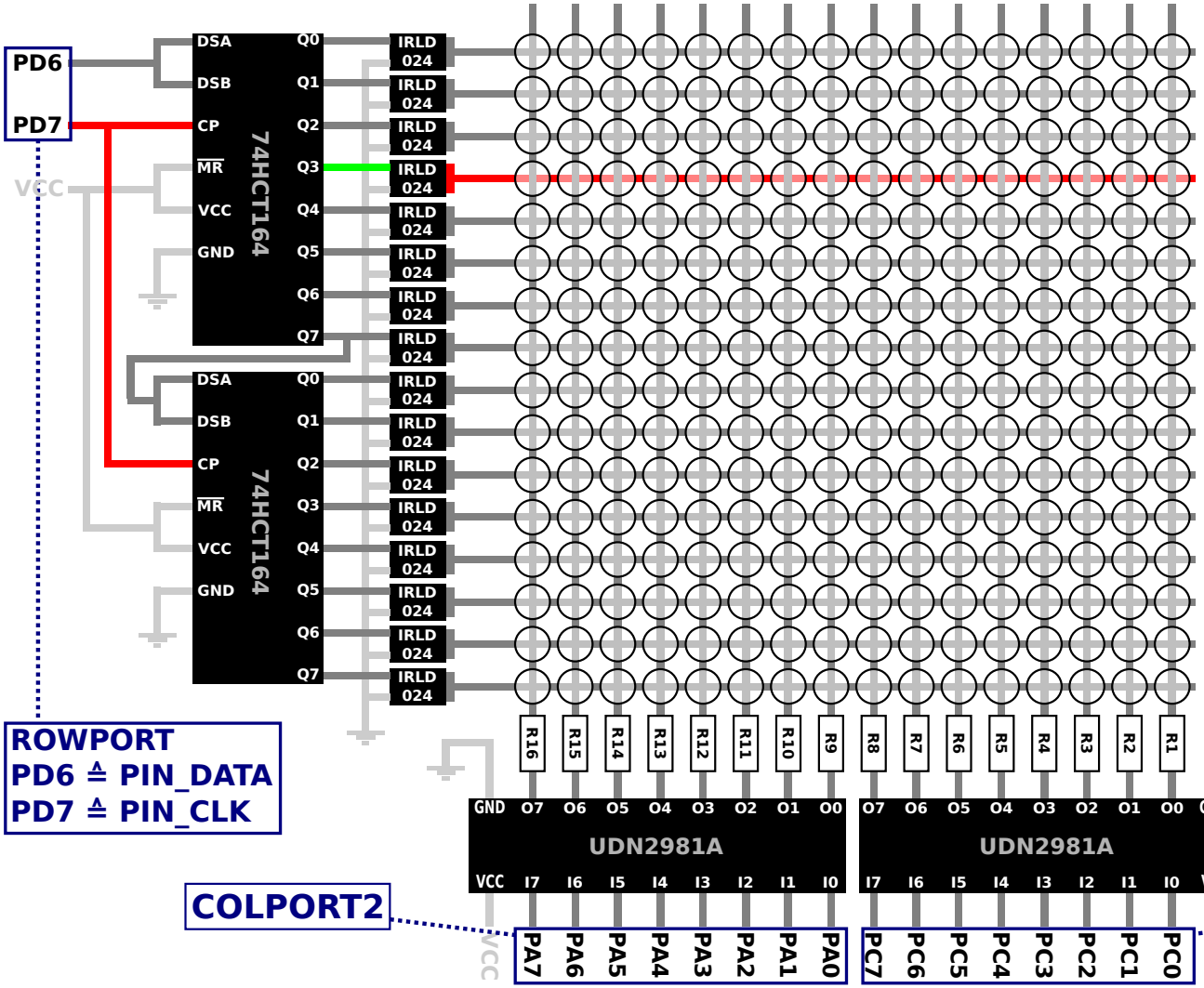

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 2
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x03  CP1: 0xC0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



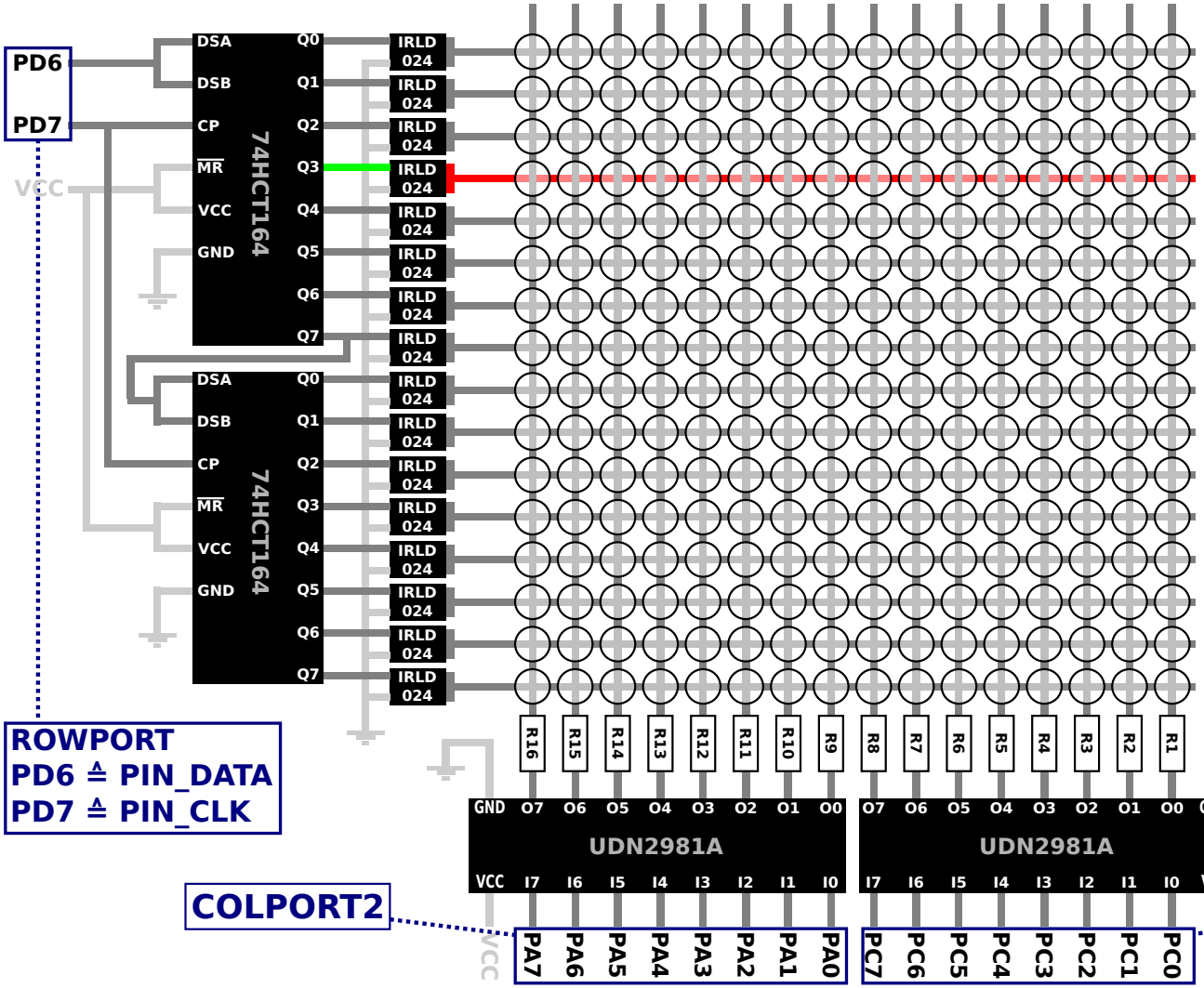

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 3
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x07  CP1: 0xE0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



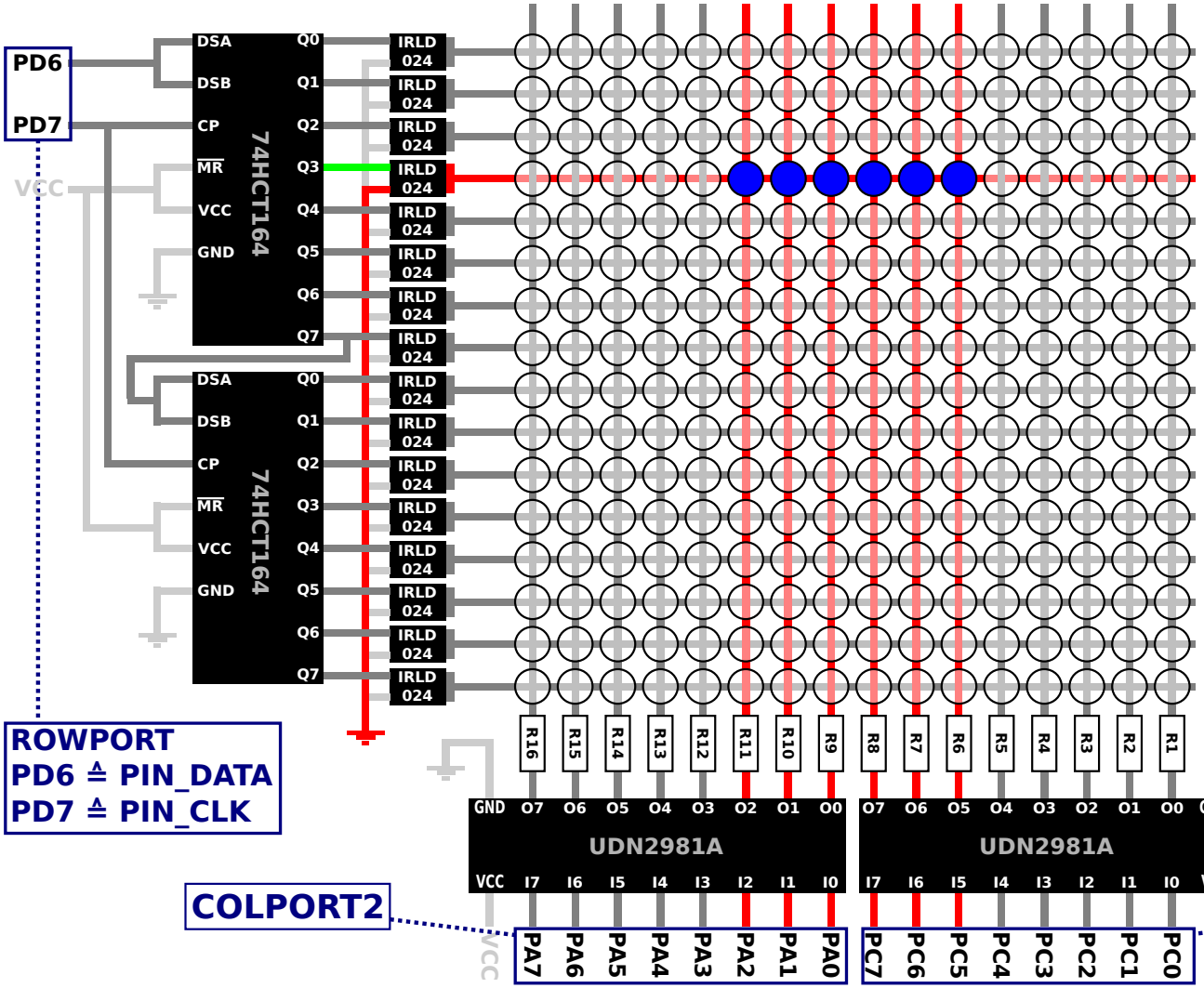

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 3
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x07  CP1: 0xE0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



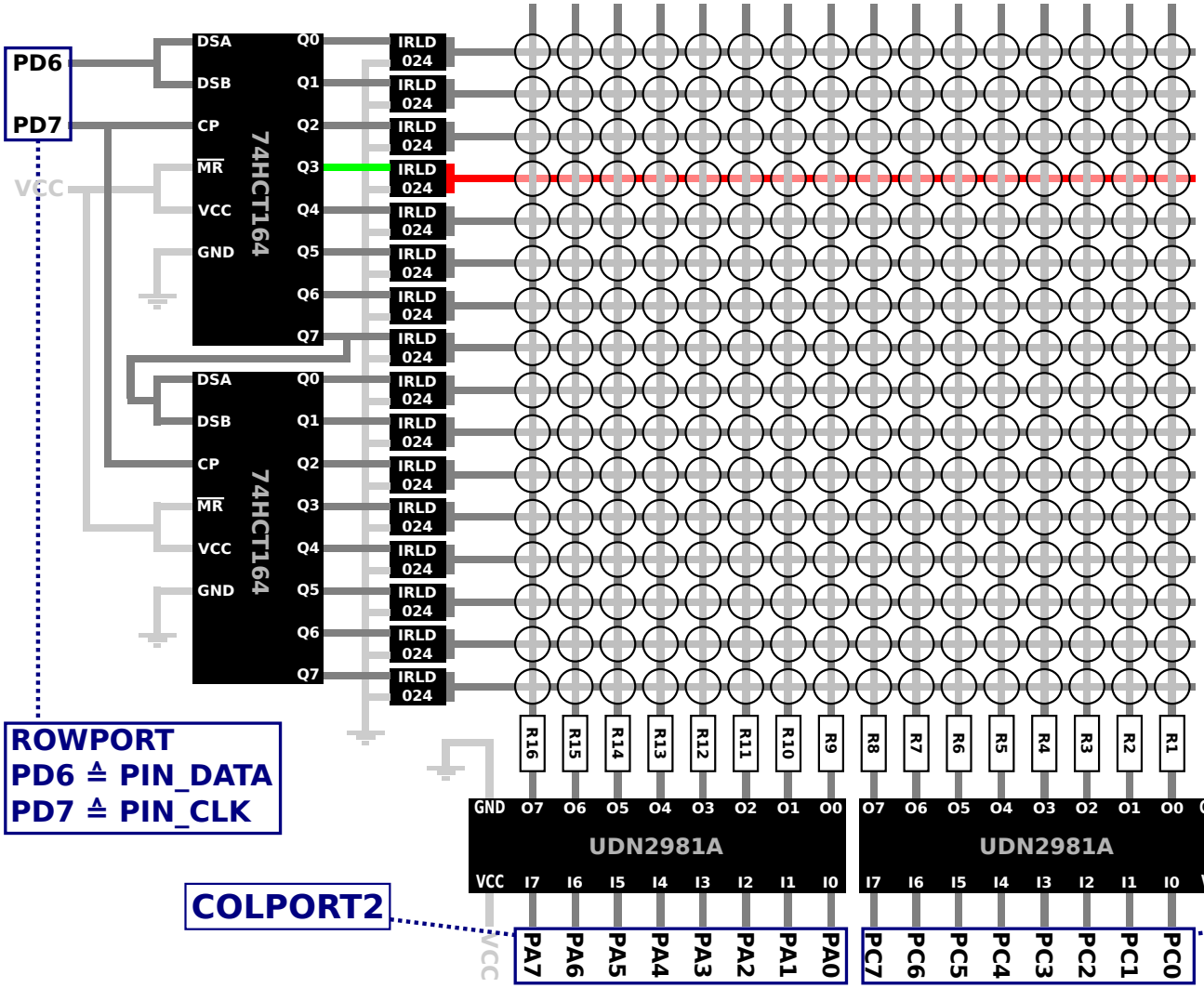

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 3
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x07  CP1: 0xE0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



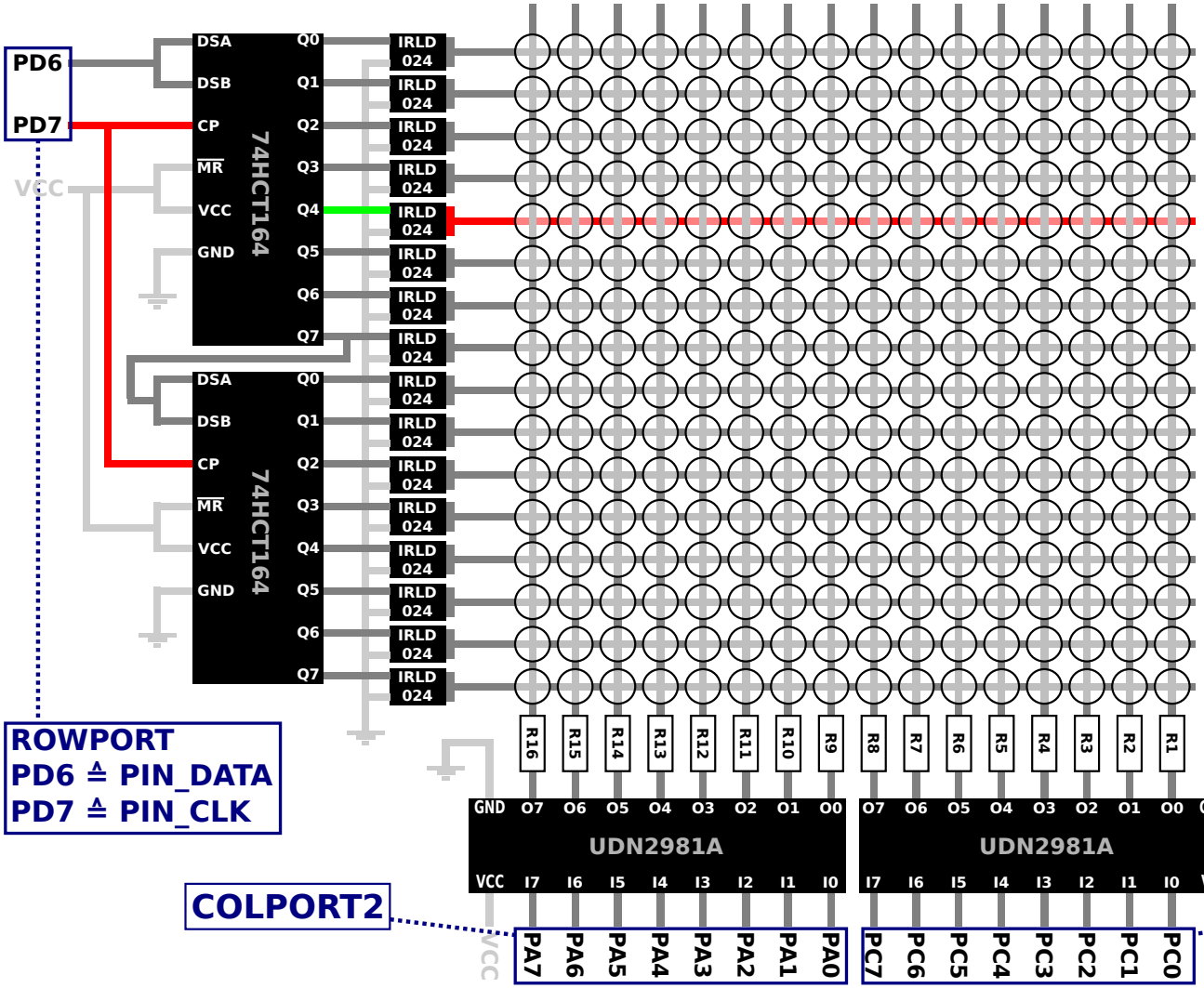

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 3
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x07  CP1: 0xE0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



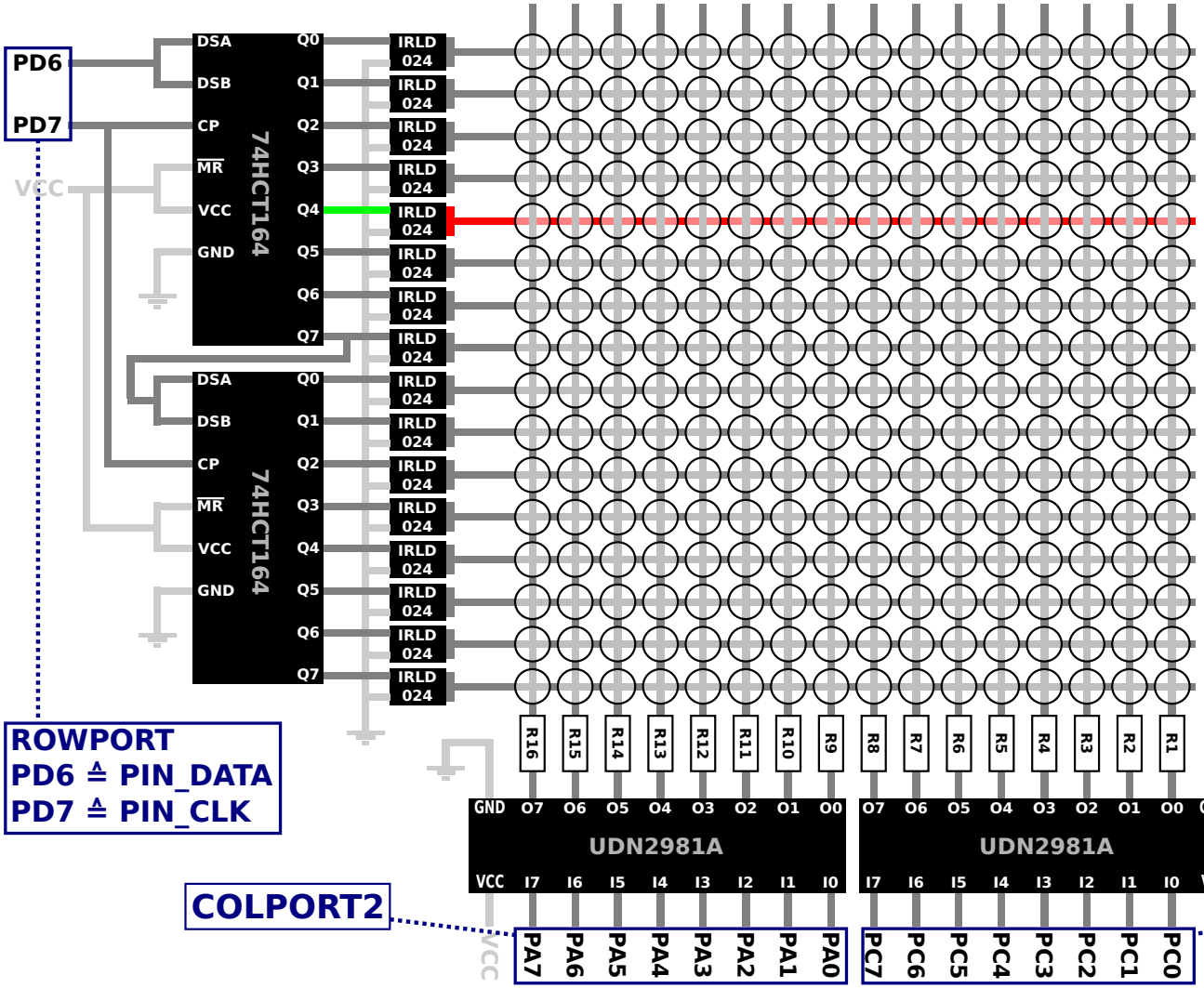

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 4
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xFF  CP1: 0xFF
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



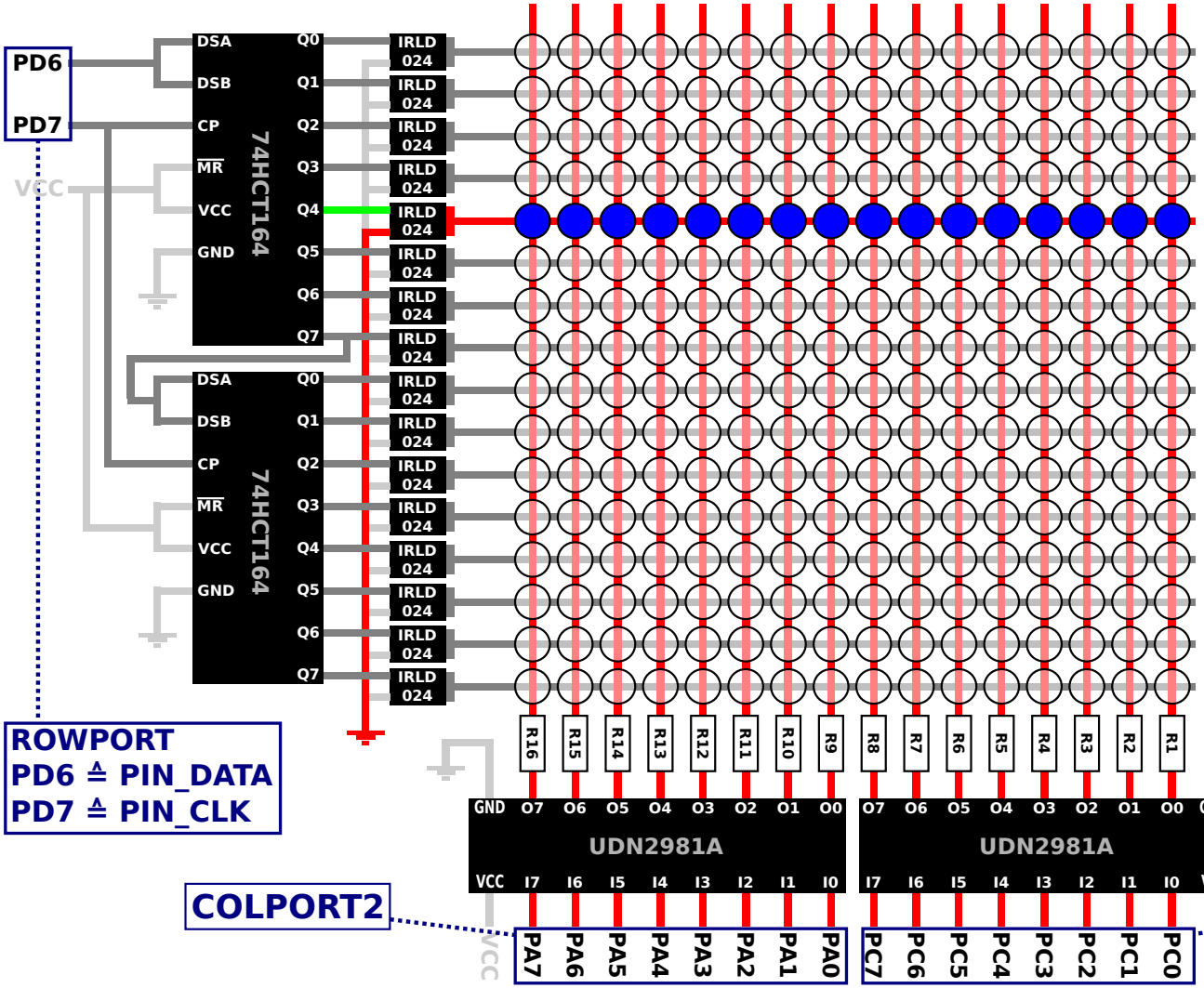

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 4
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xFF  CP1: 0xFF
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



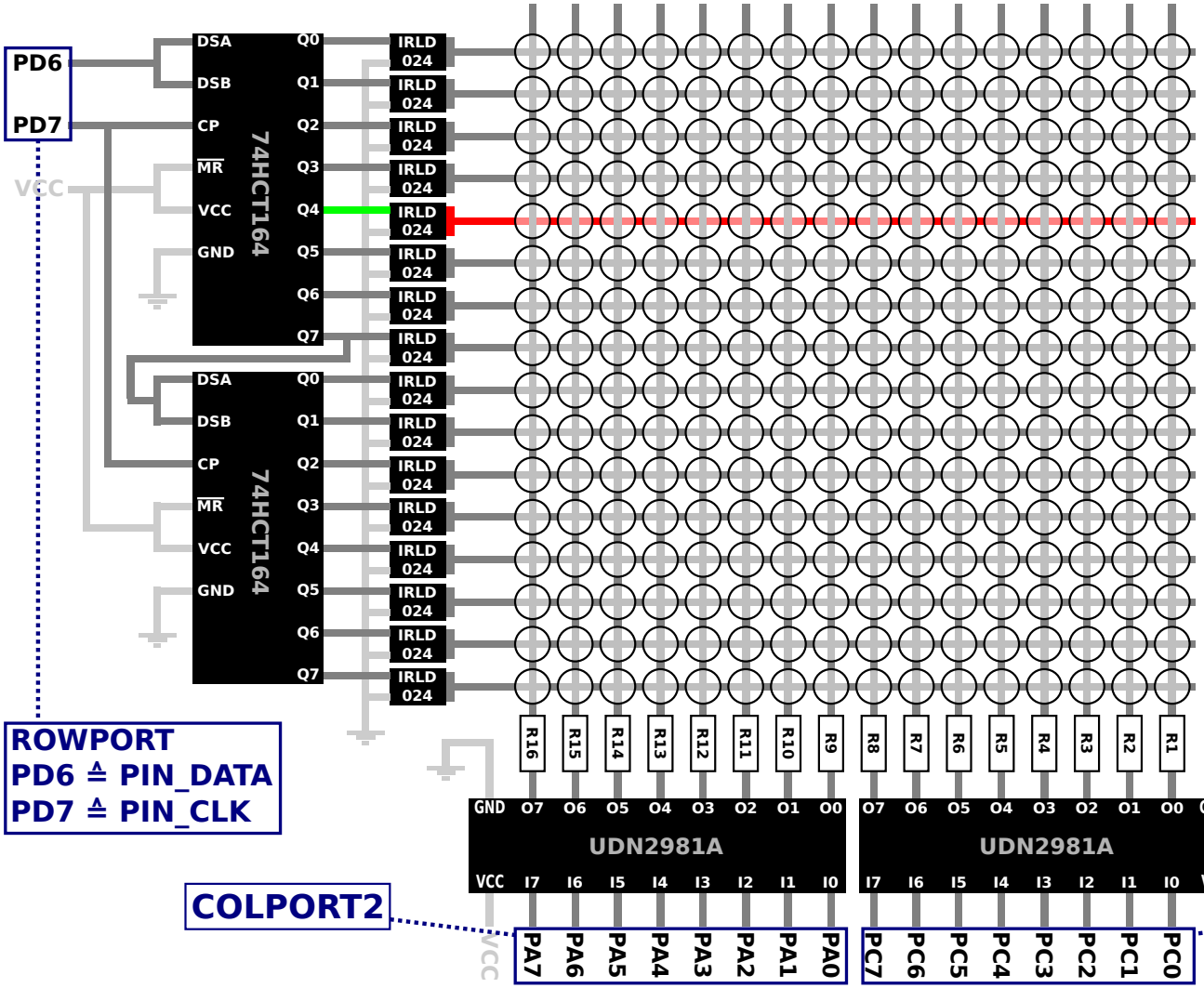

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 4
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xFF  CP1: 0xFF
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



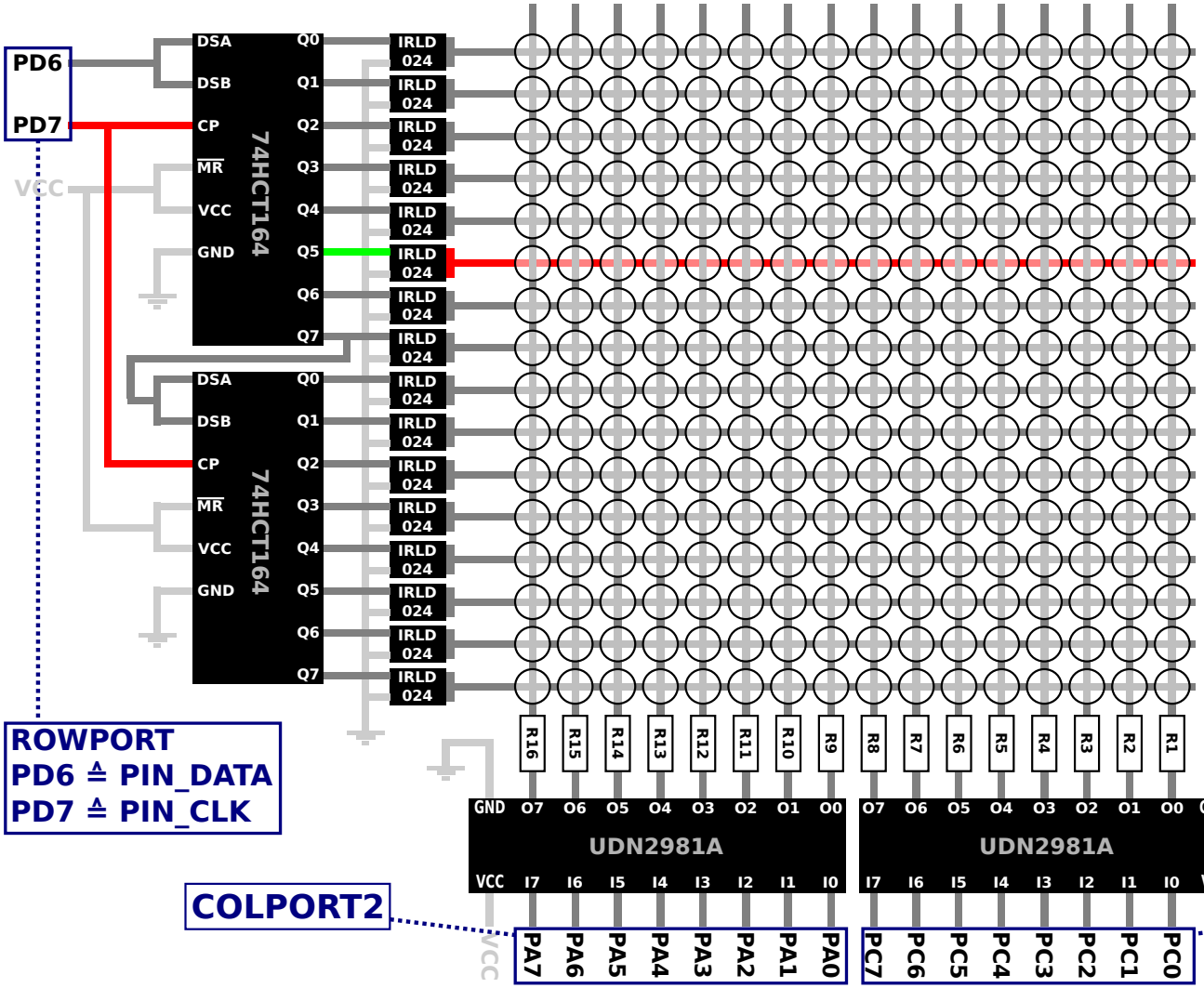

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 4
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xFF  CP1: 0xFF
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



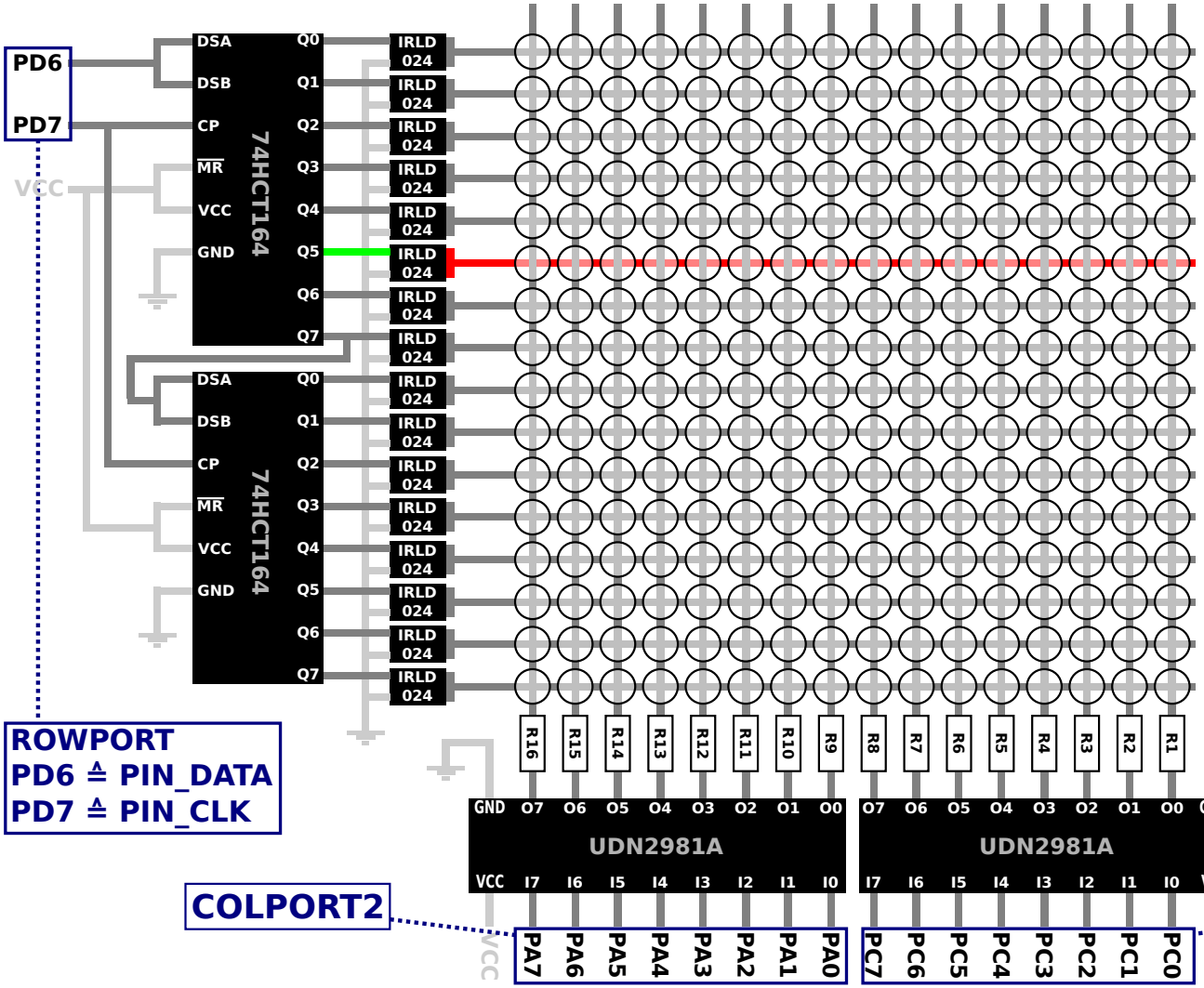

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 5
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x7F  CP1: 0xFE
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



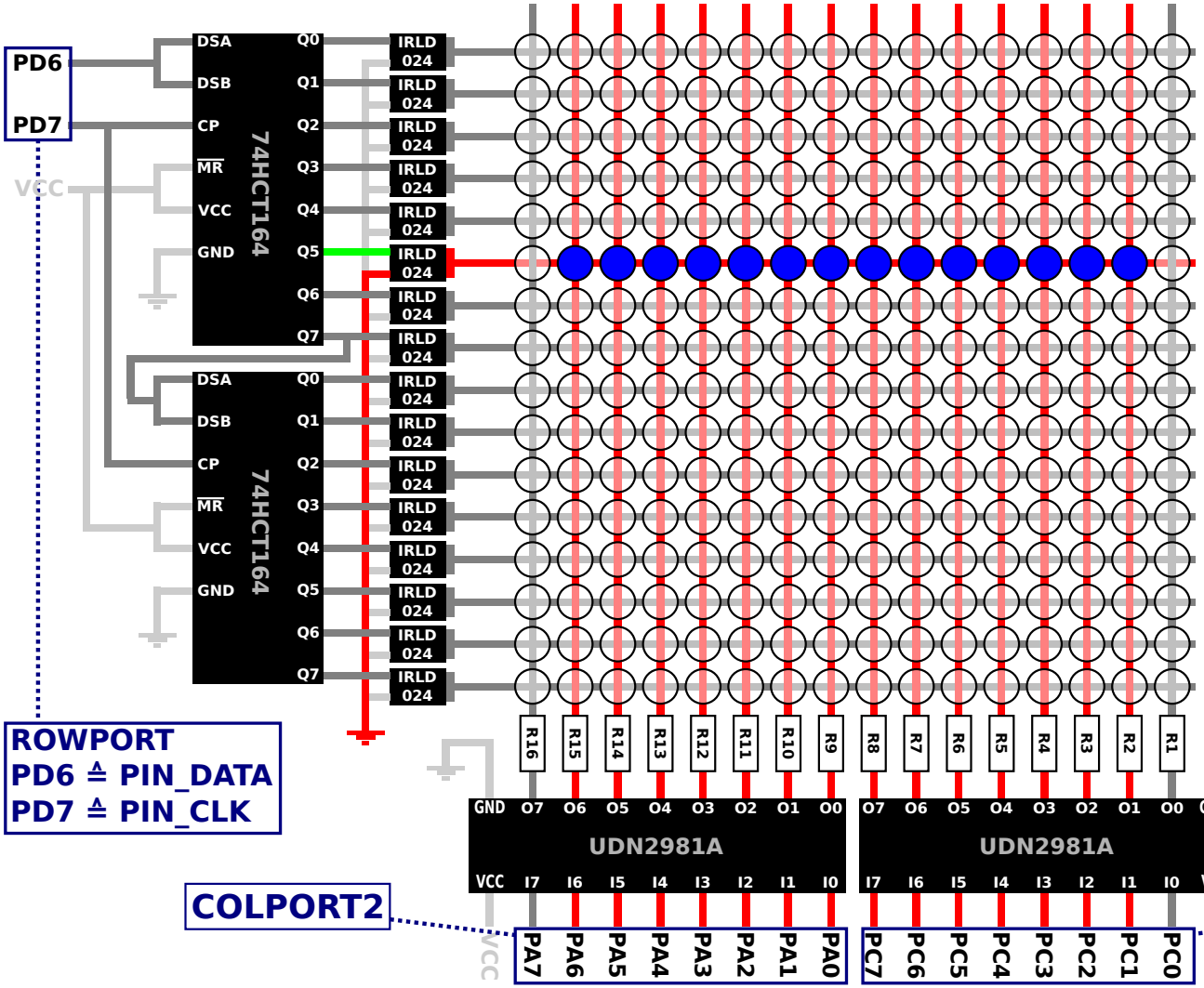

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 5
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x7F  CP1: 0xFE
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



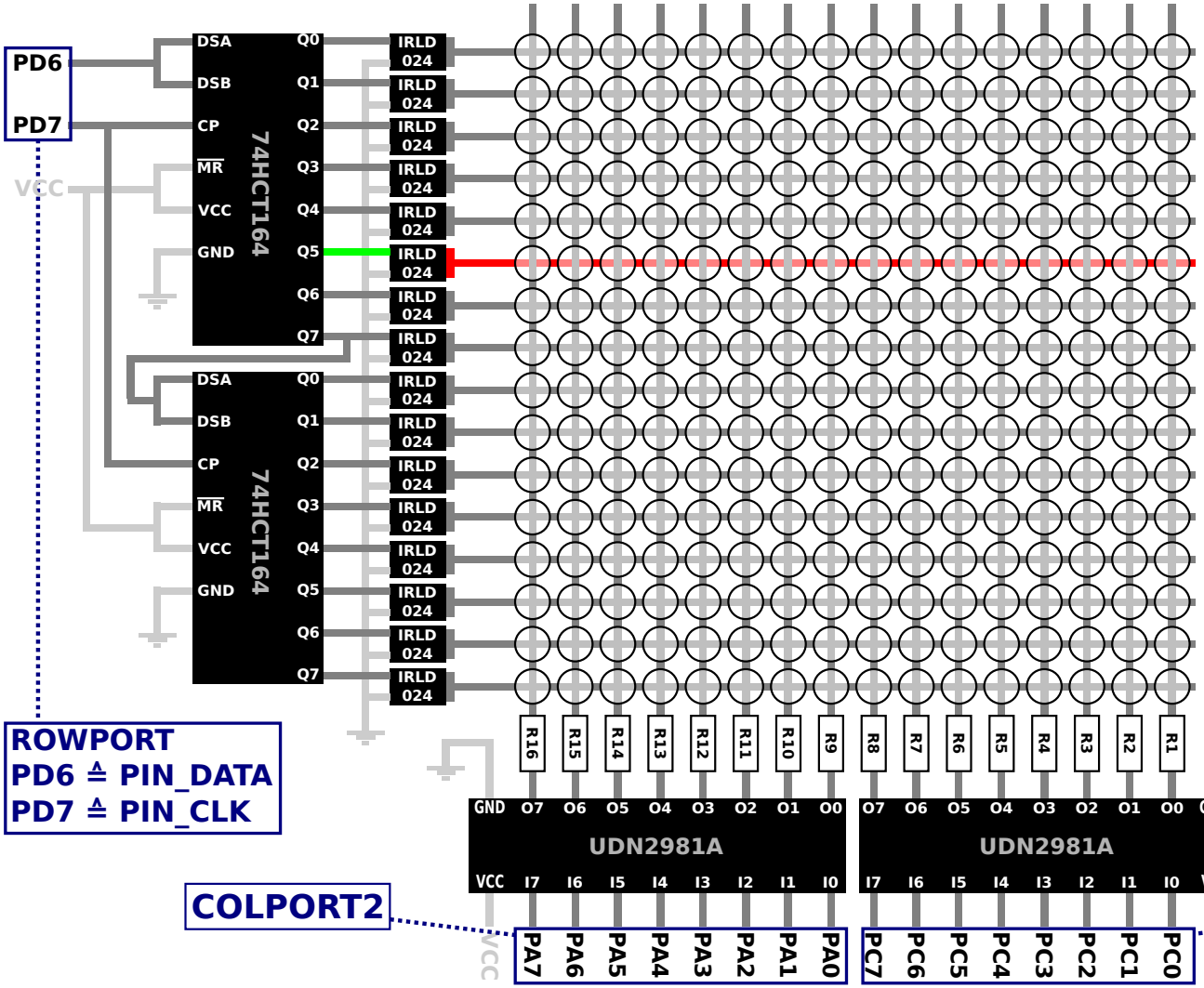

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 5
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x7F  CP1: 0xFE
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



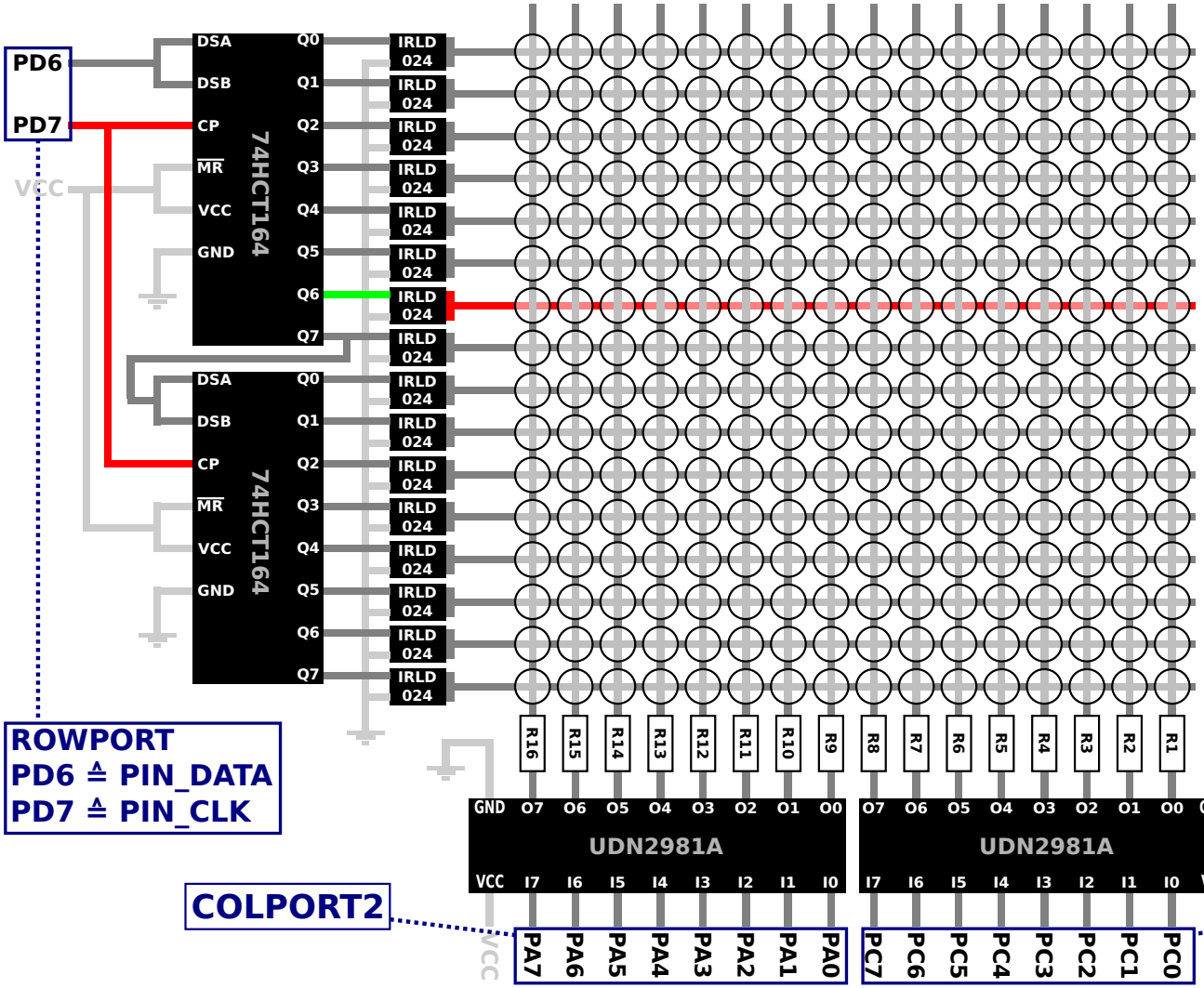

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 5
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x7F  CP1: 0xFE
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



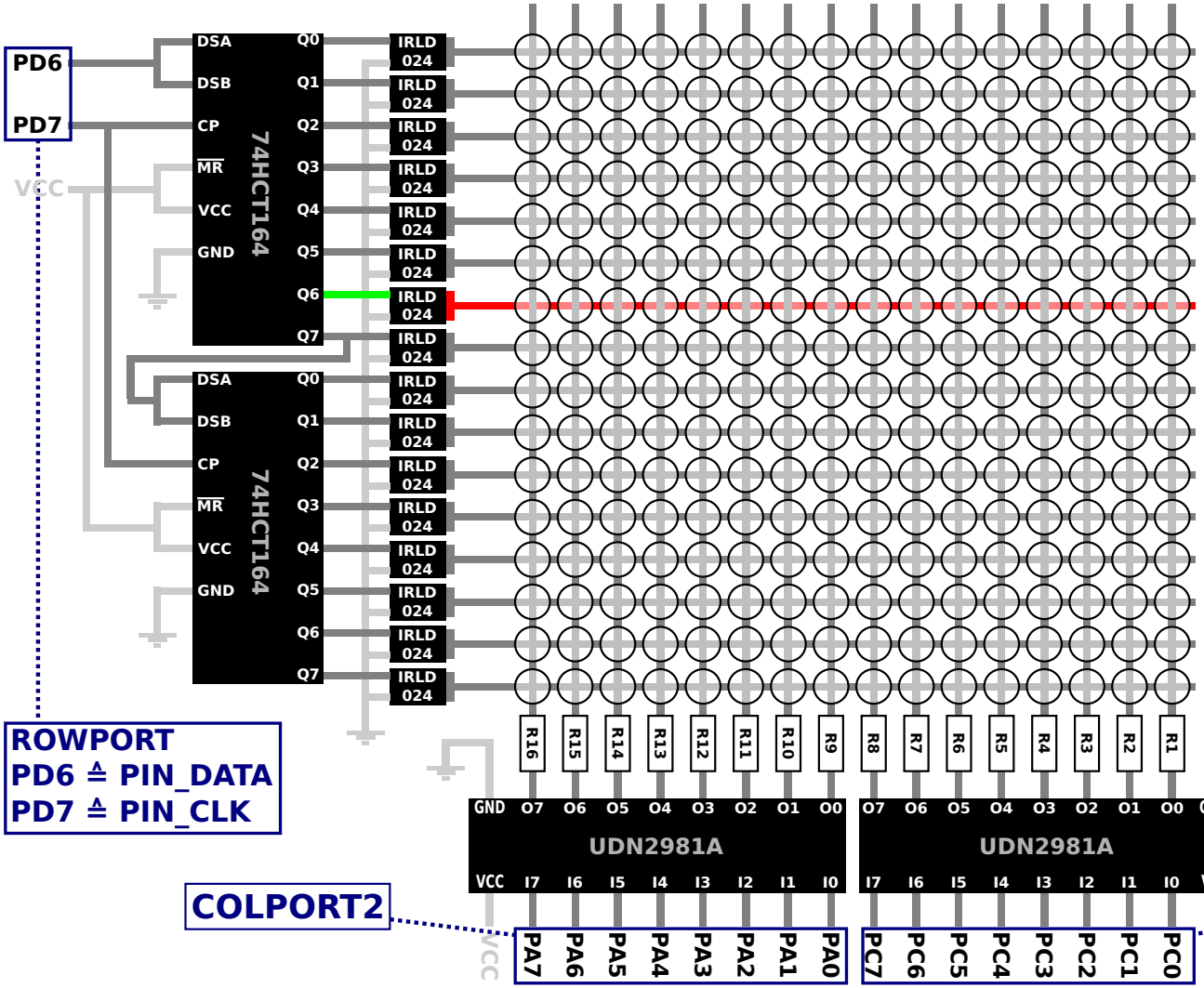

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 6
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



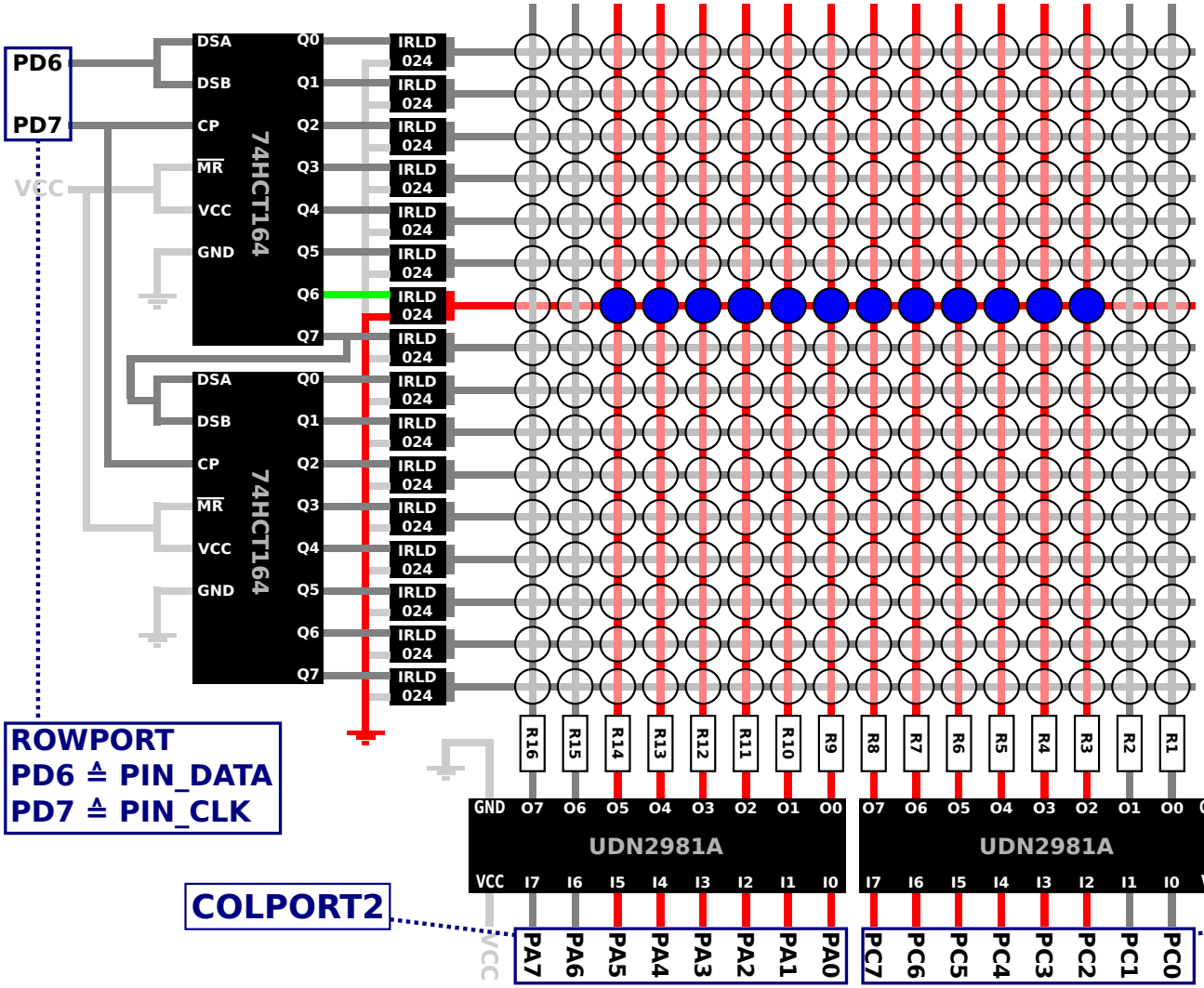

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 6
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



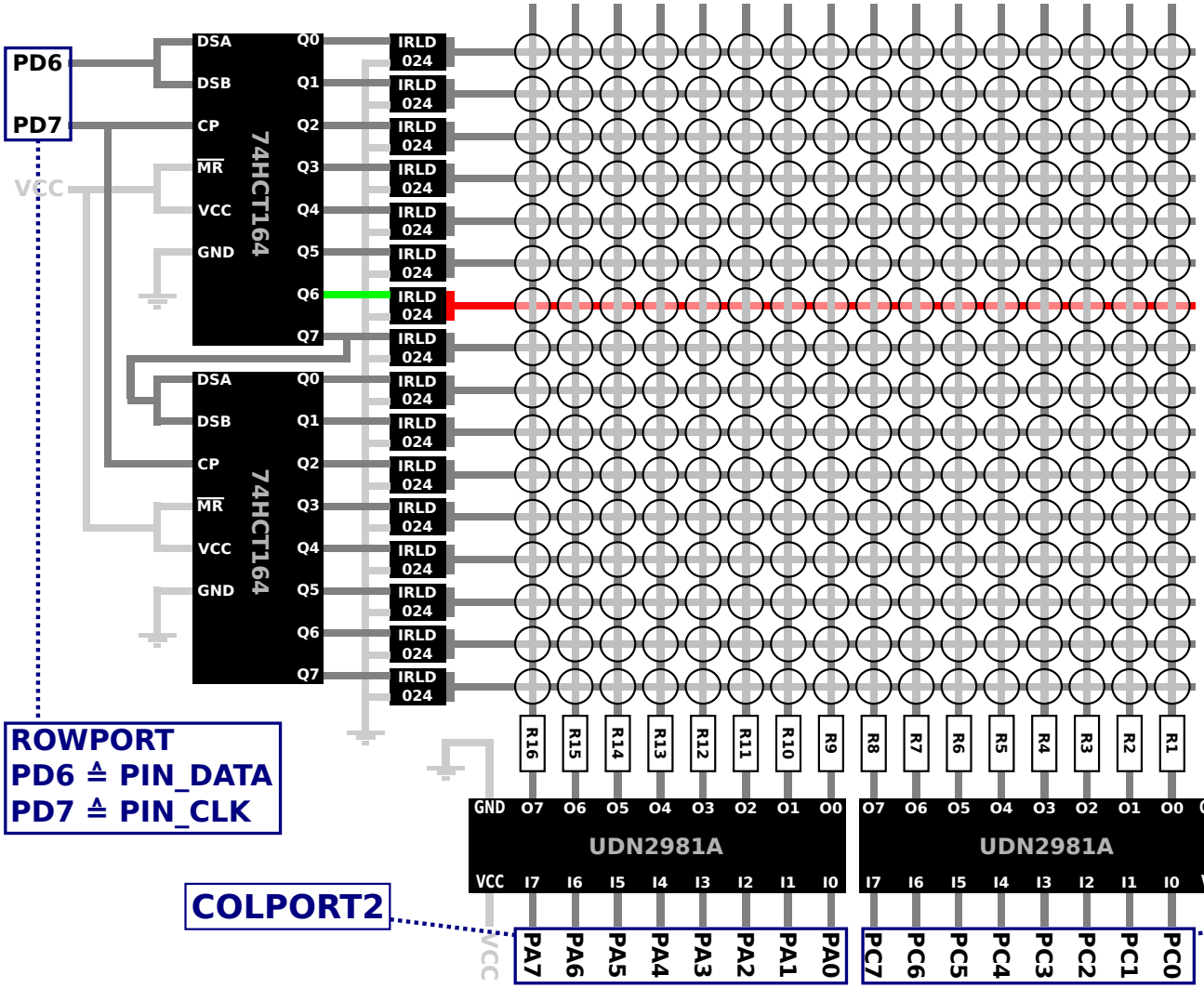

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 6
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



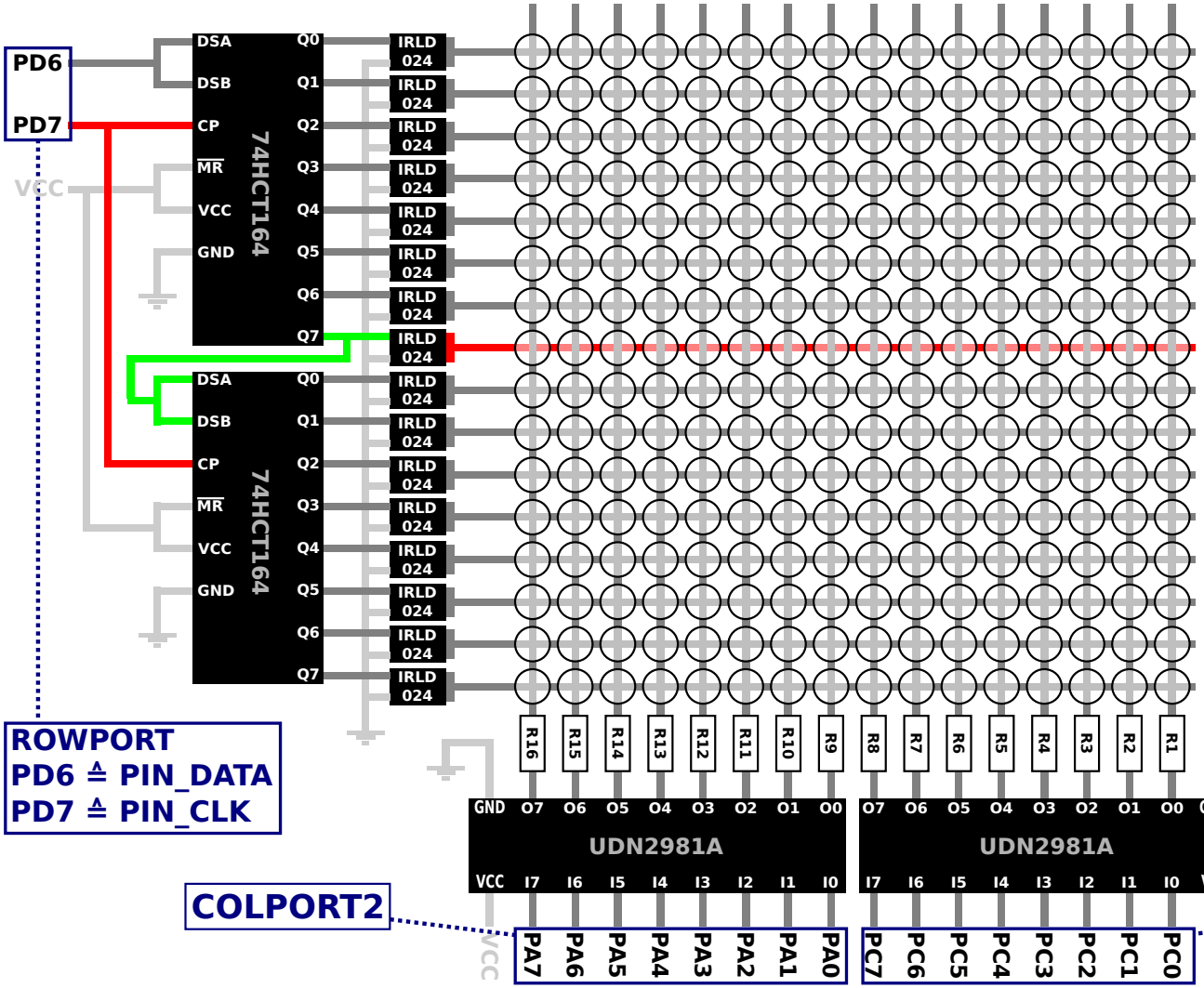

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 6
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



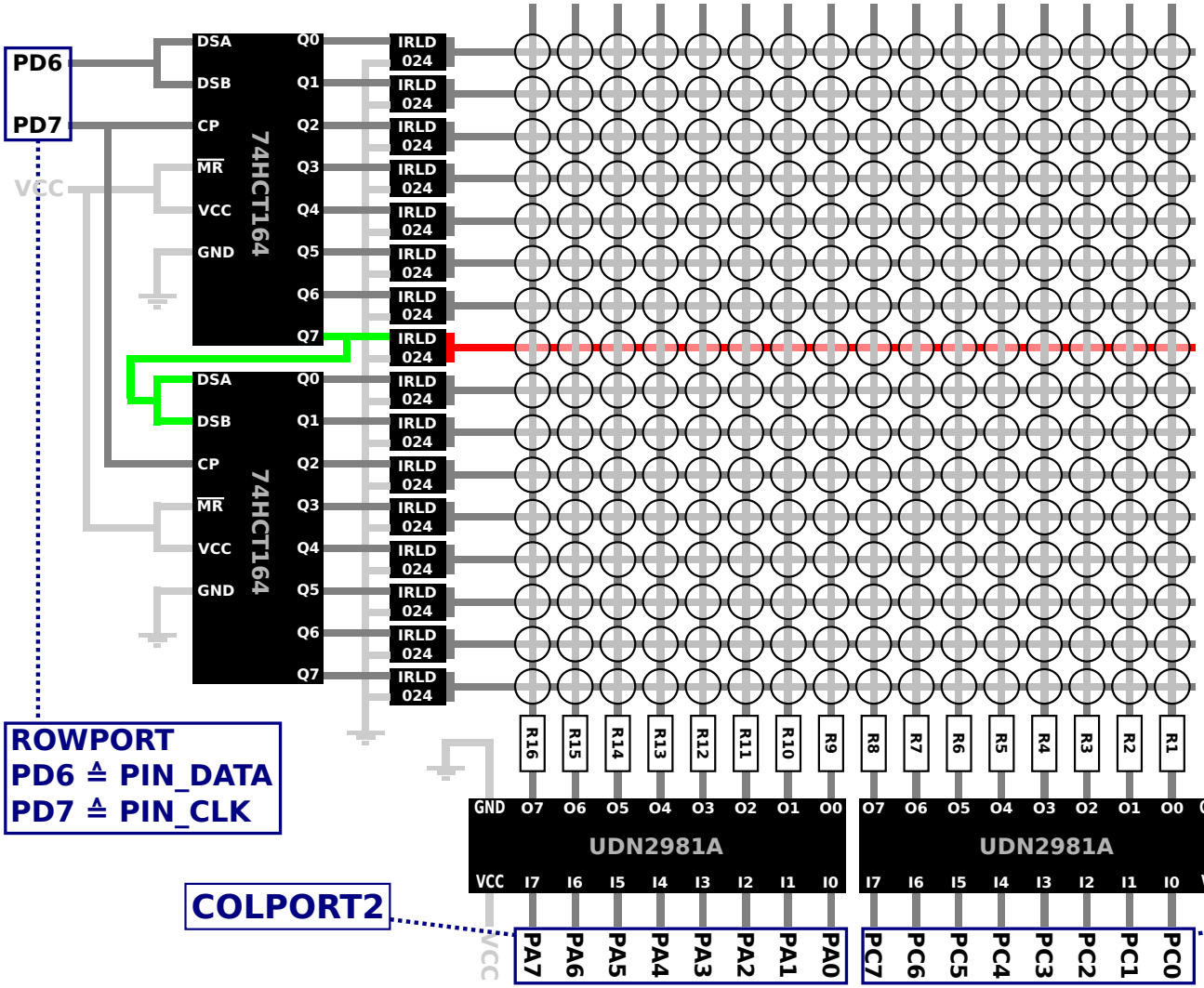

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 7
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



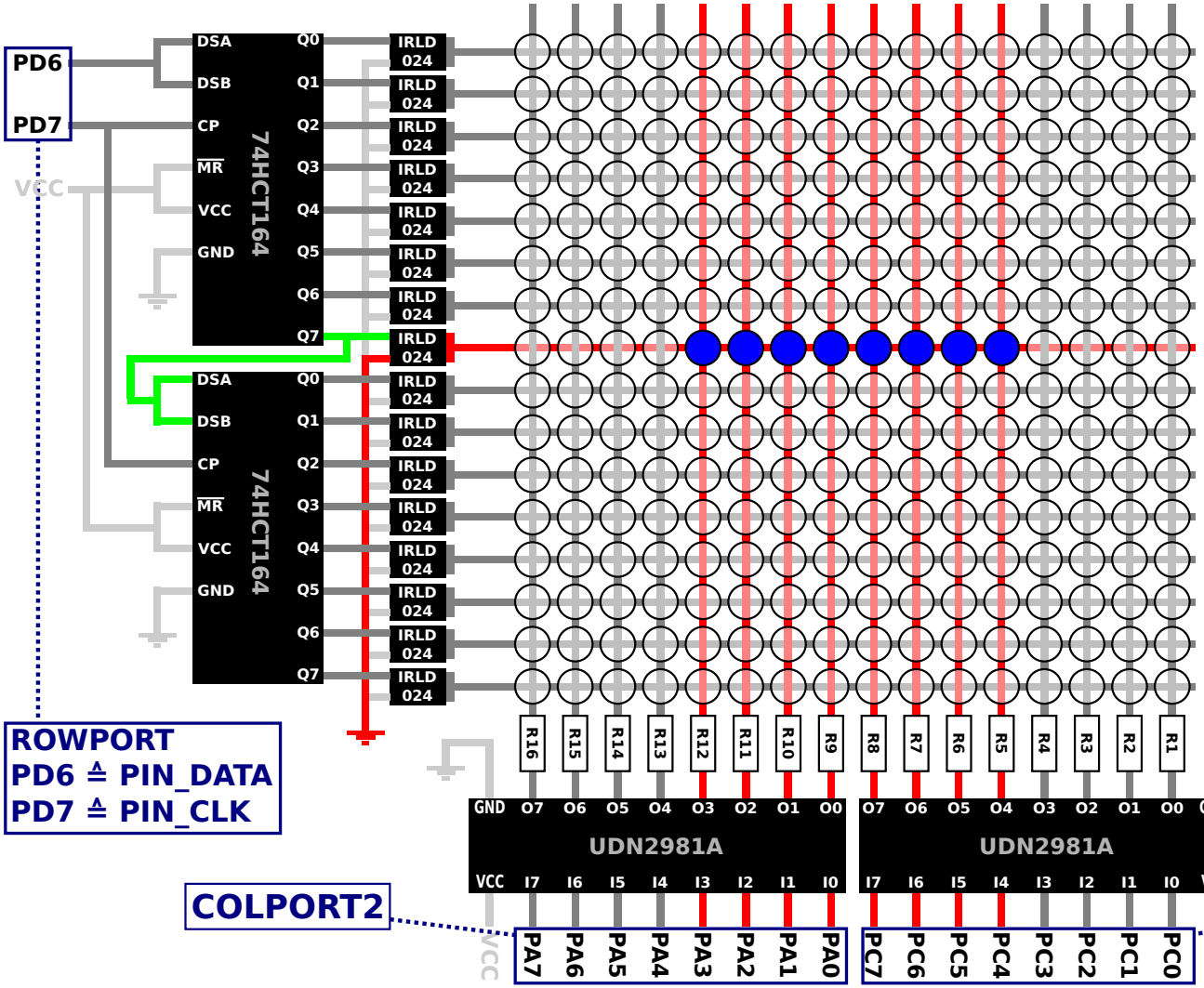

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 7
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



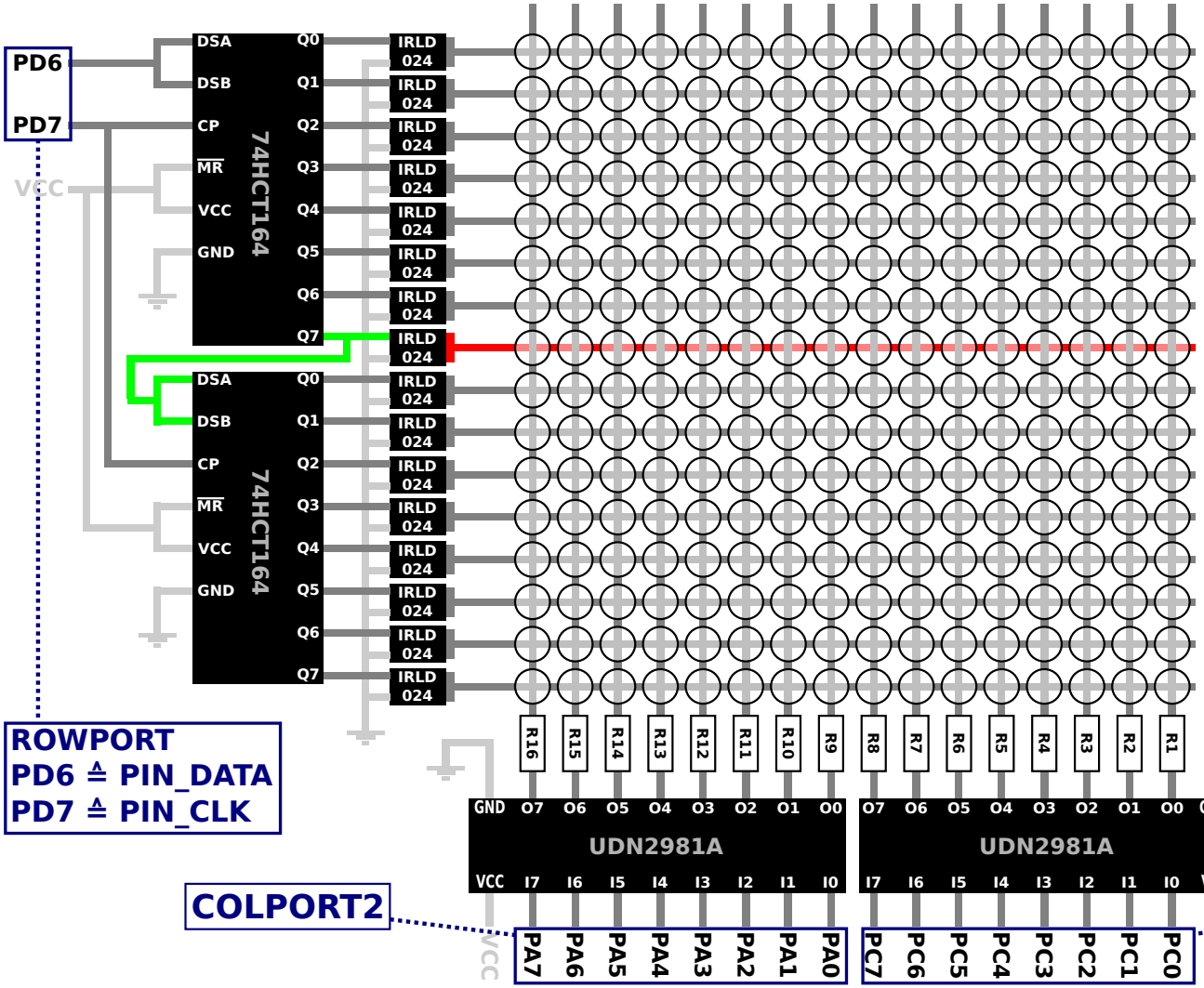

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 7
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



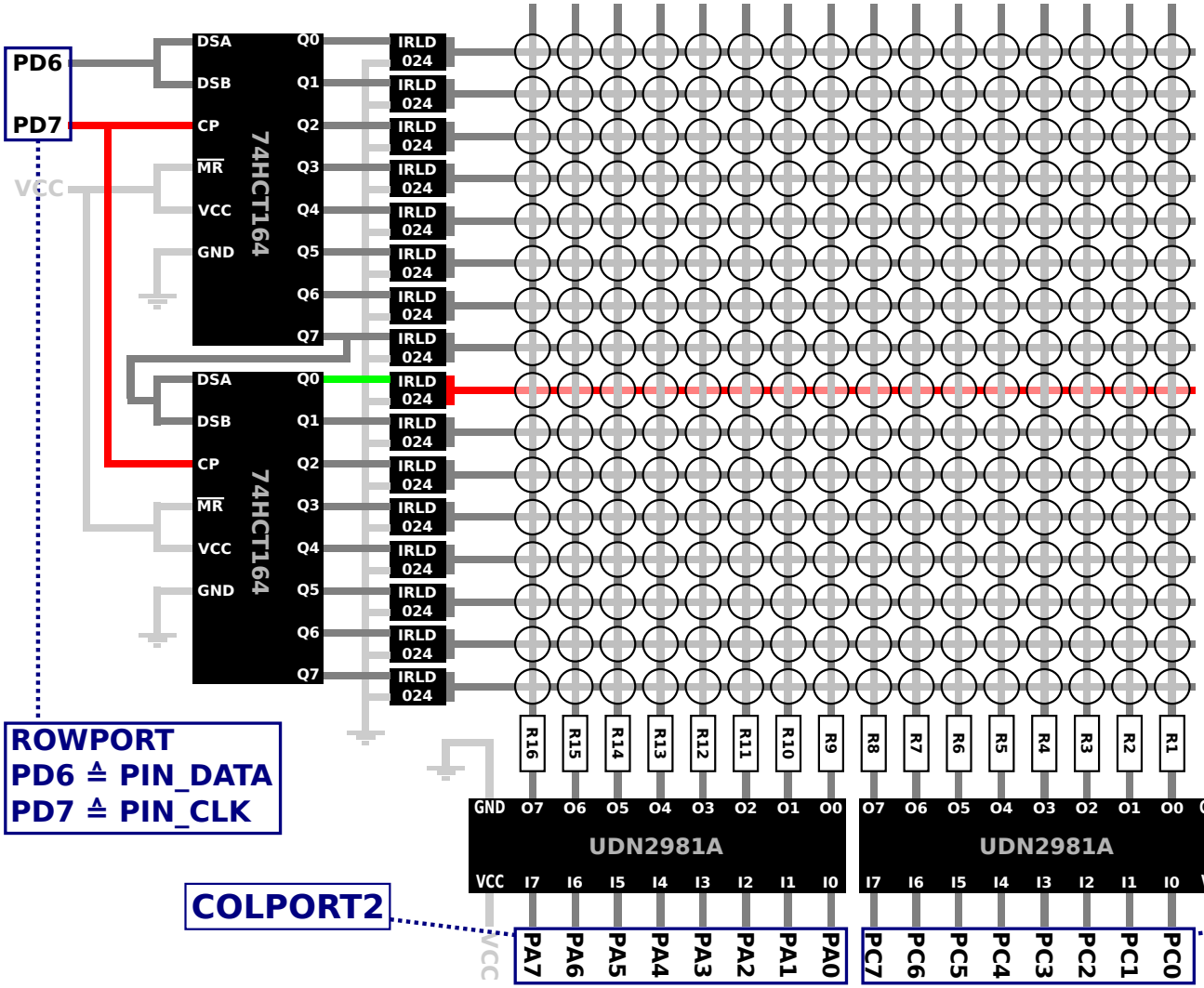

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 7
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



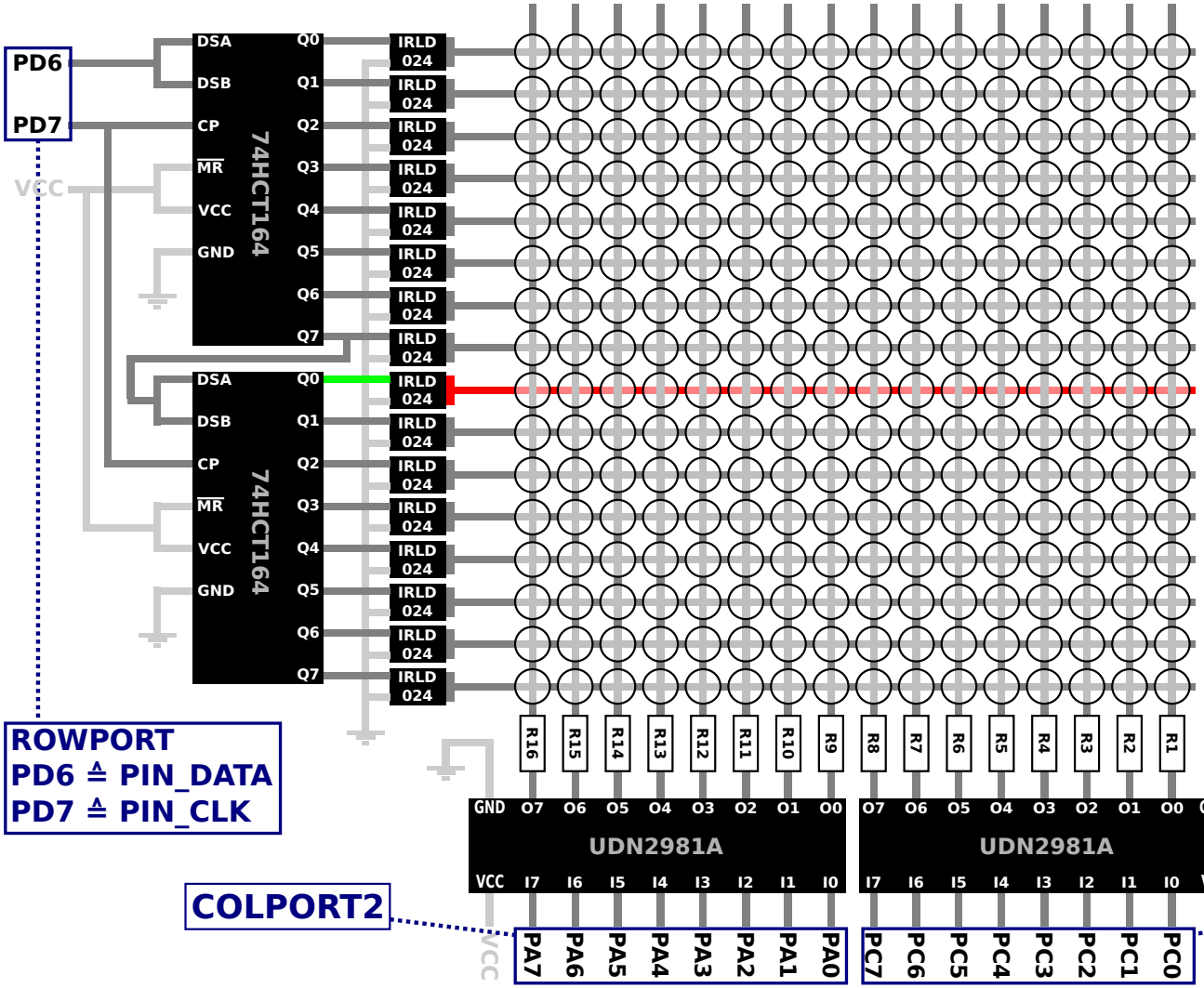

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 8
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



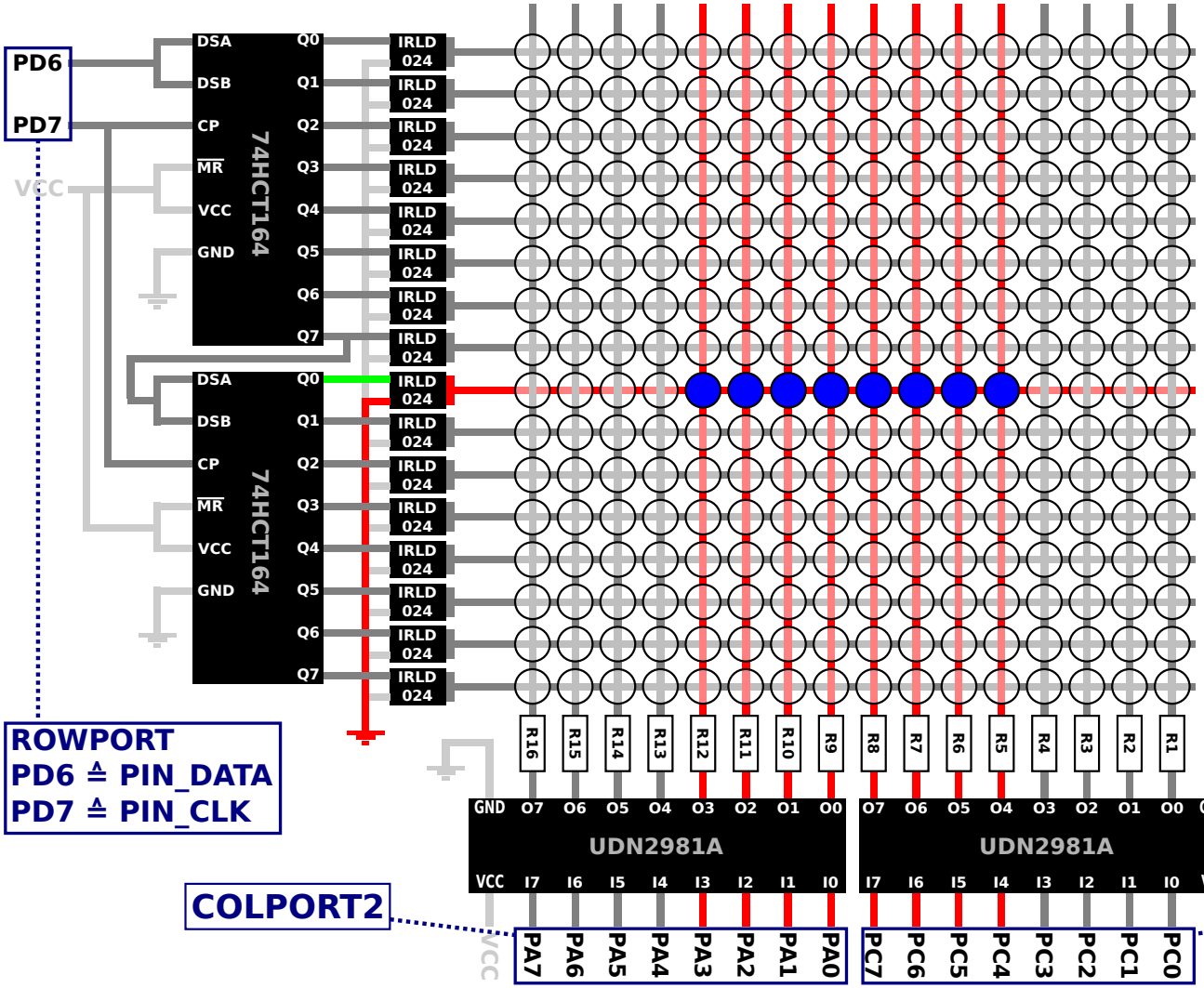

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 8
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



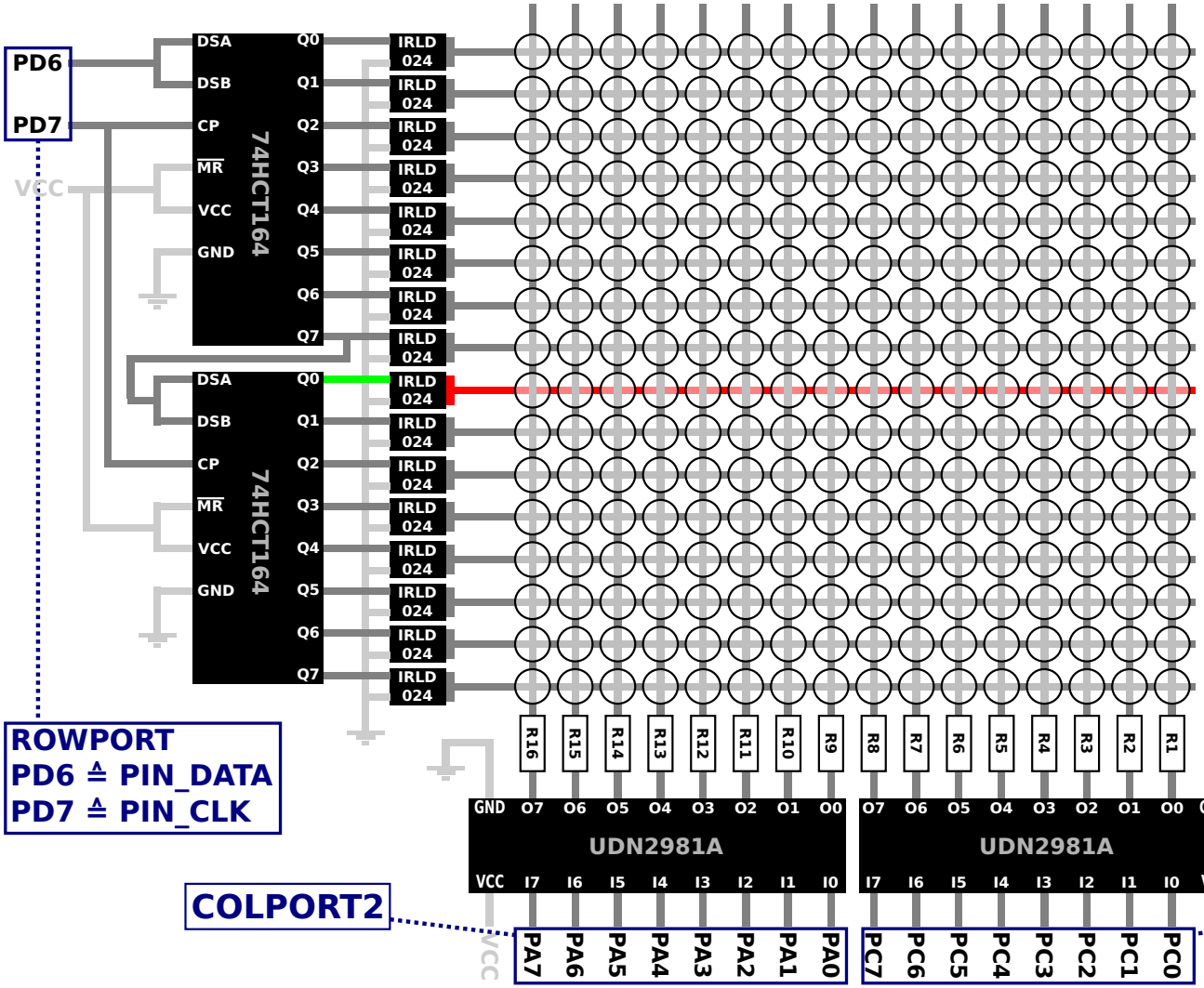

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 8
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



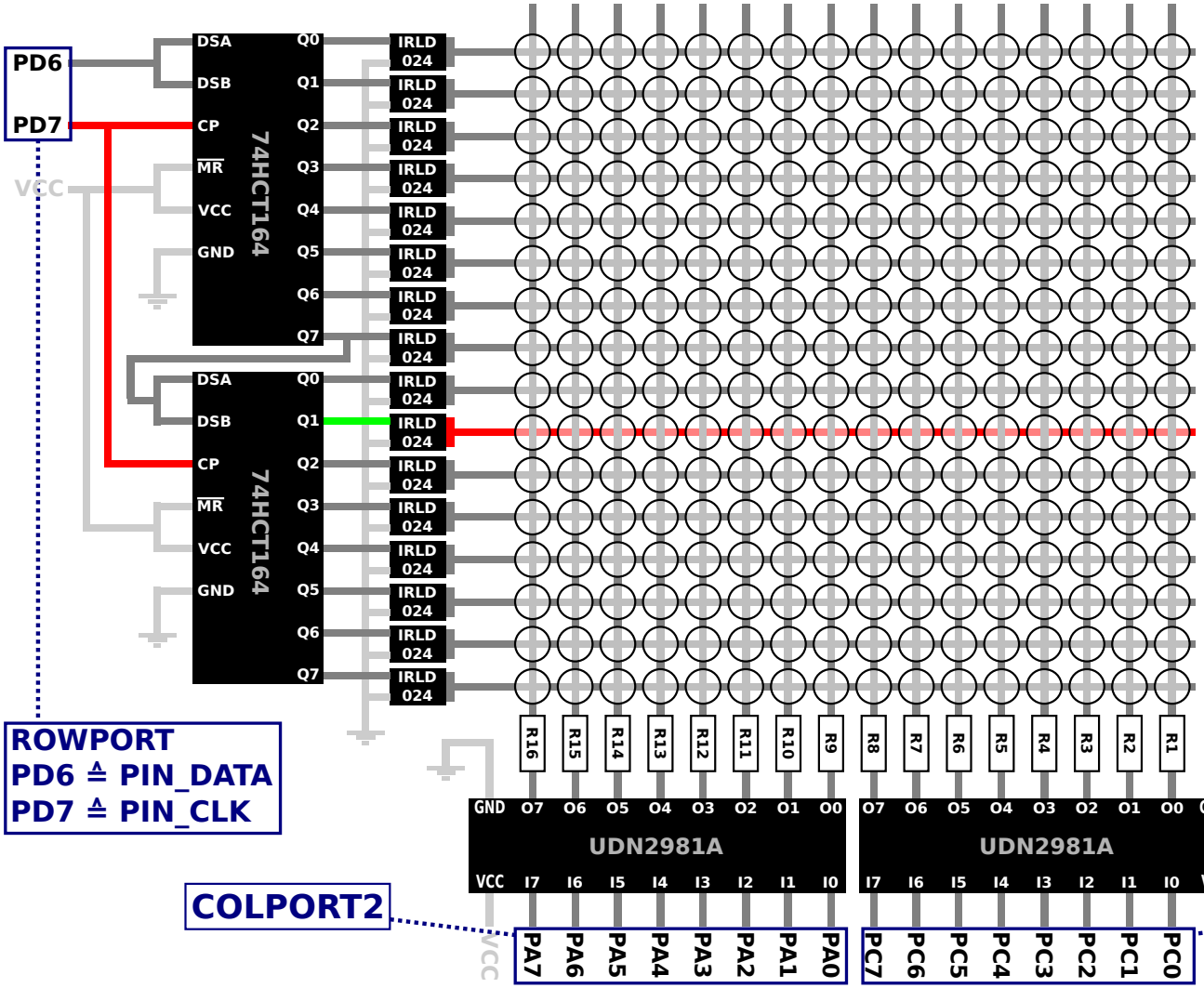

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 8
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x0F  CP1: 0xF0
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



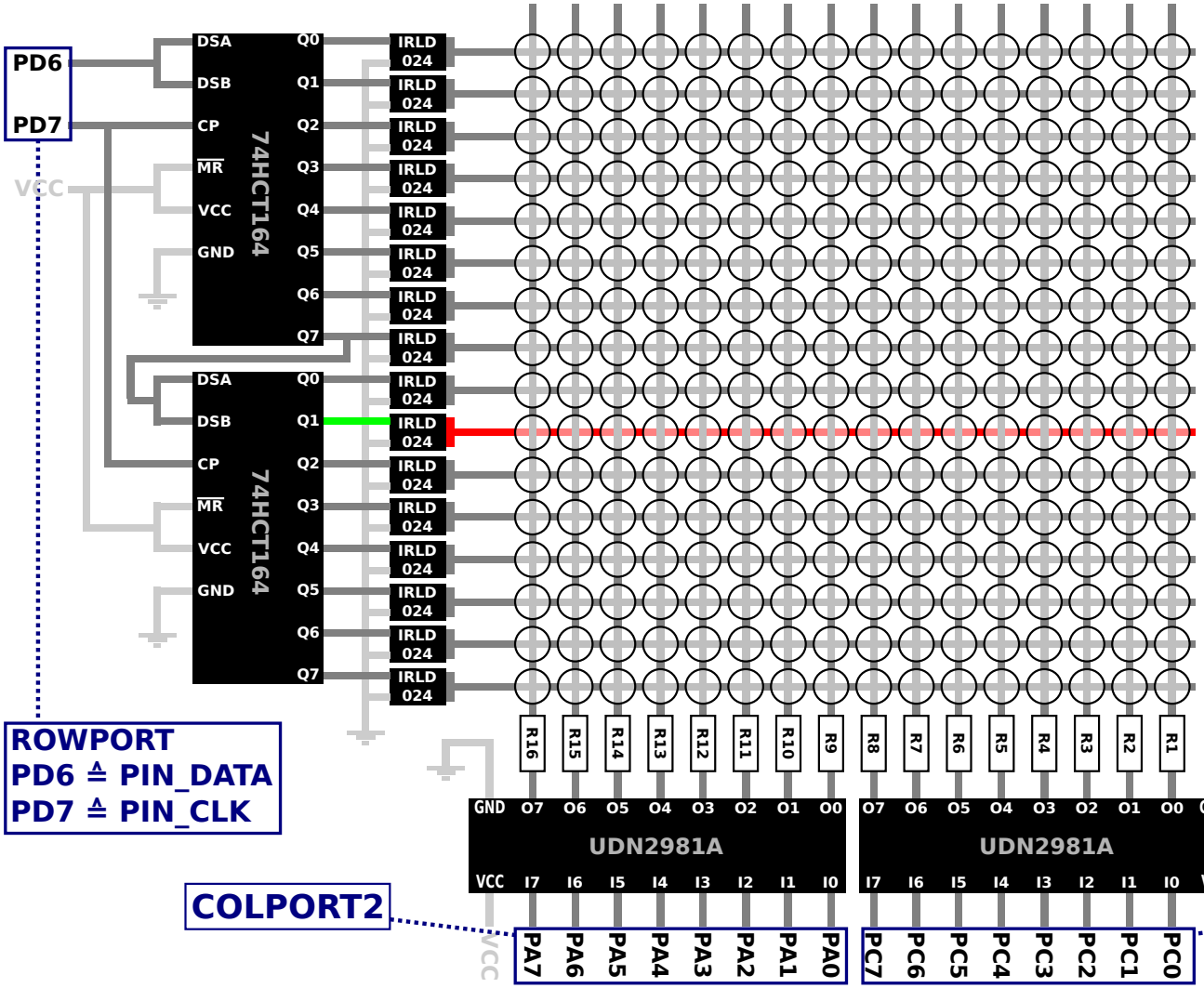

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 9
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x1F  CP1: 0xF8
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



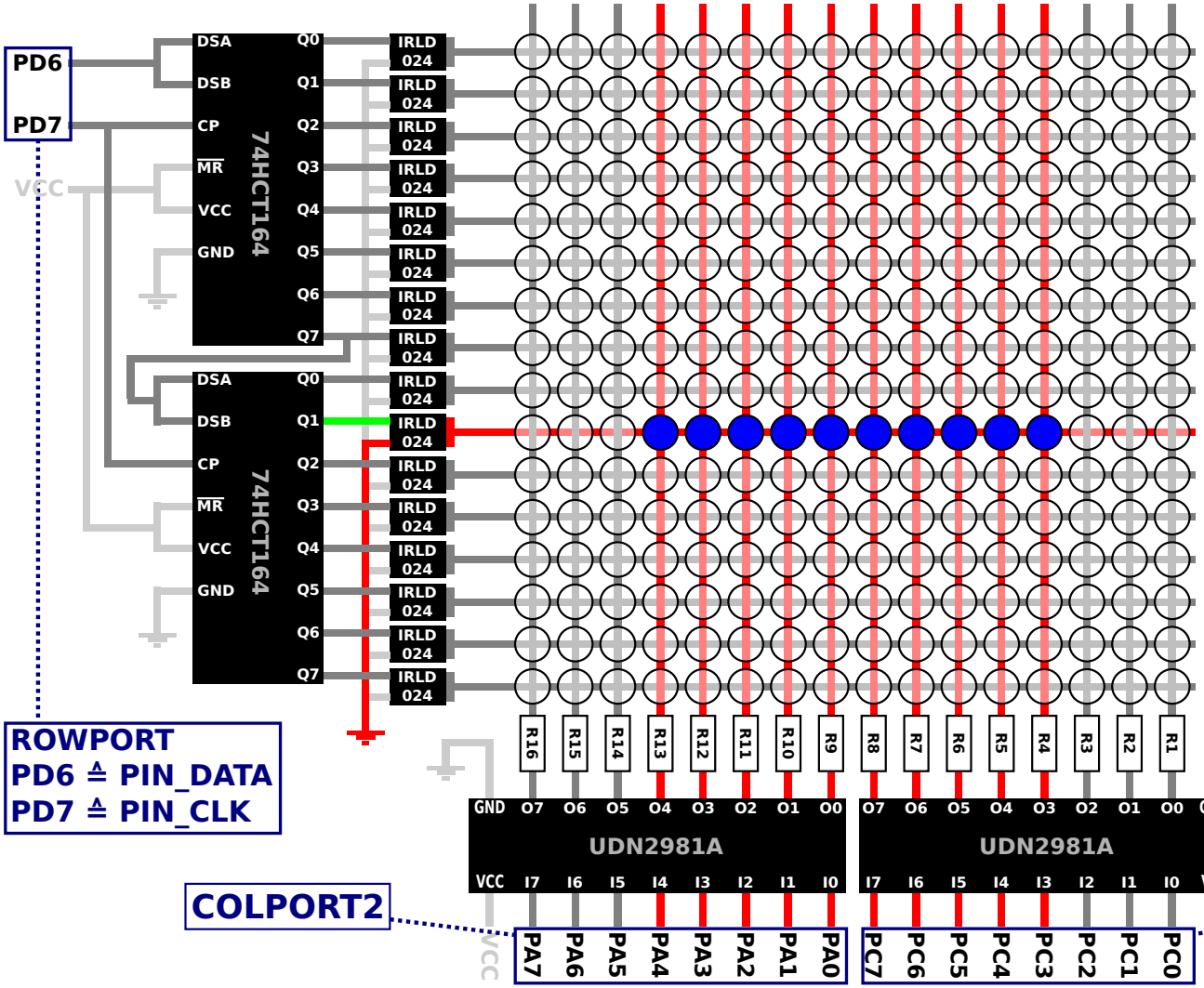

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 9
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x1F  CP1: 0xF8
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



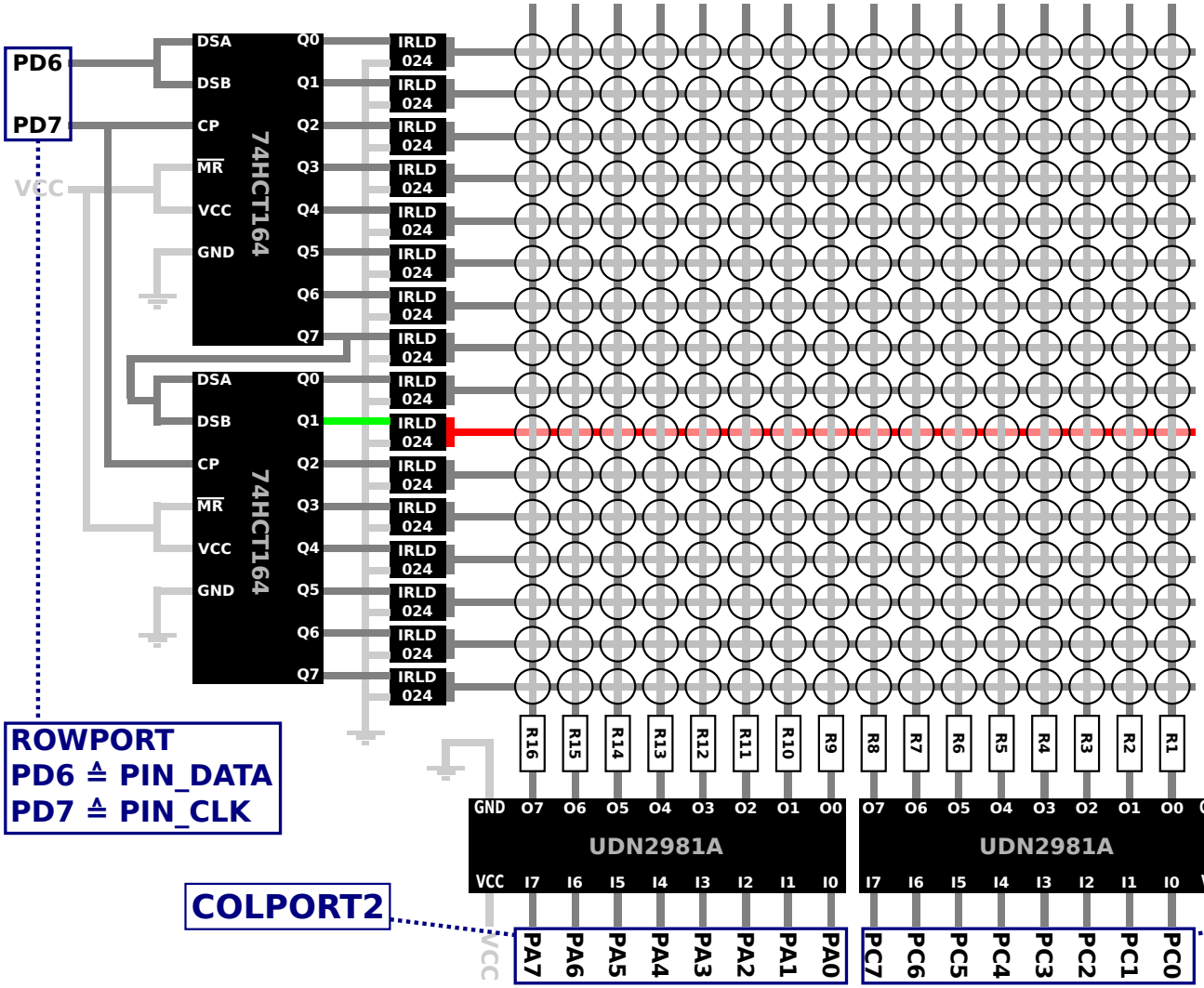

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 9
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x1F  CP1: 0xF8
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



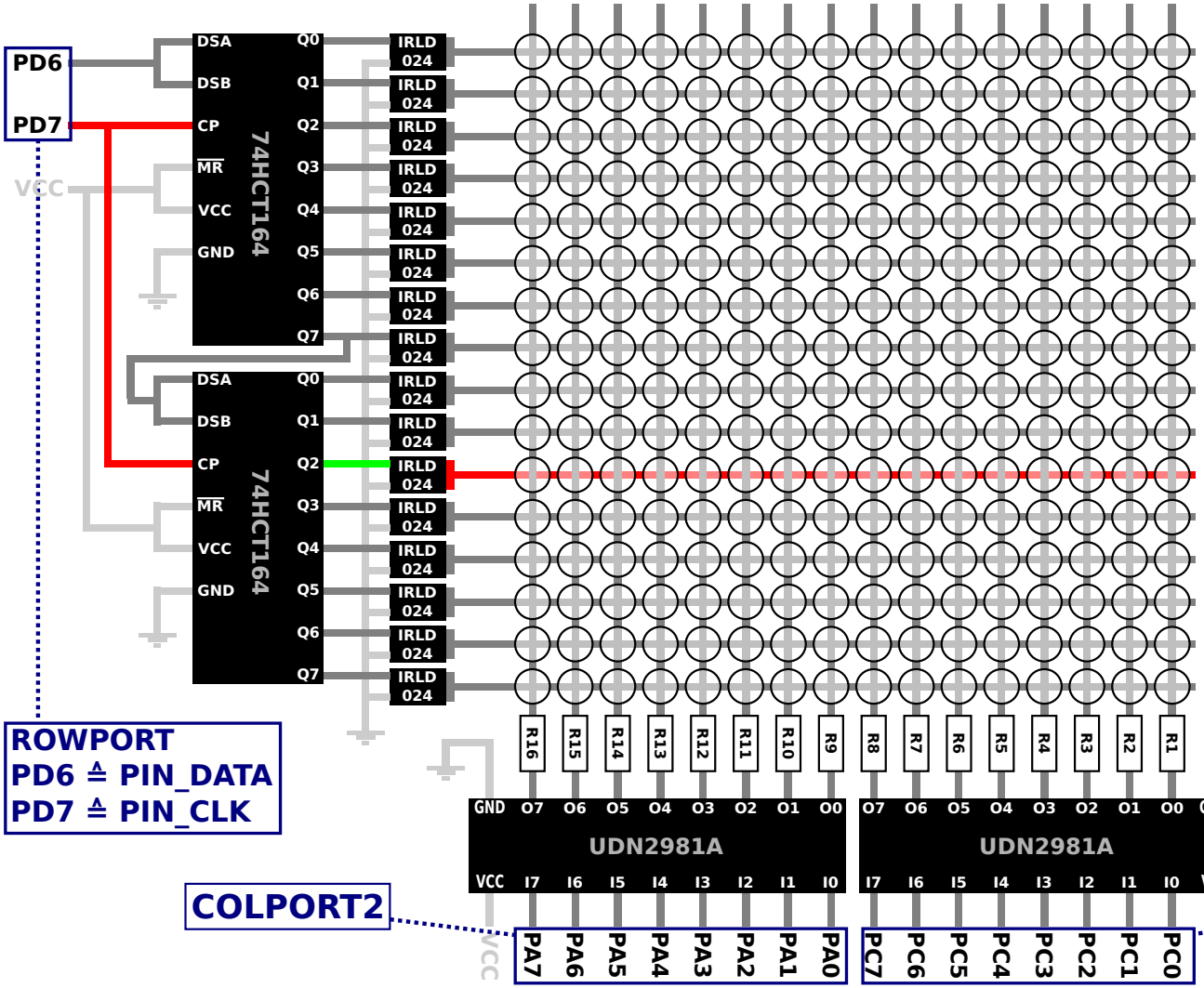

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 9
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x1F  CP1: 0xF8
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



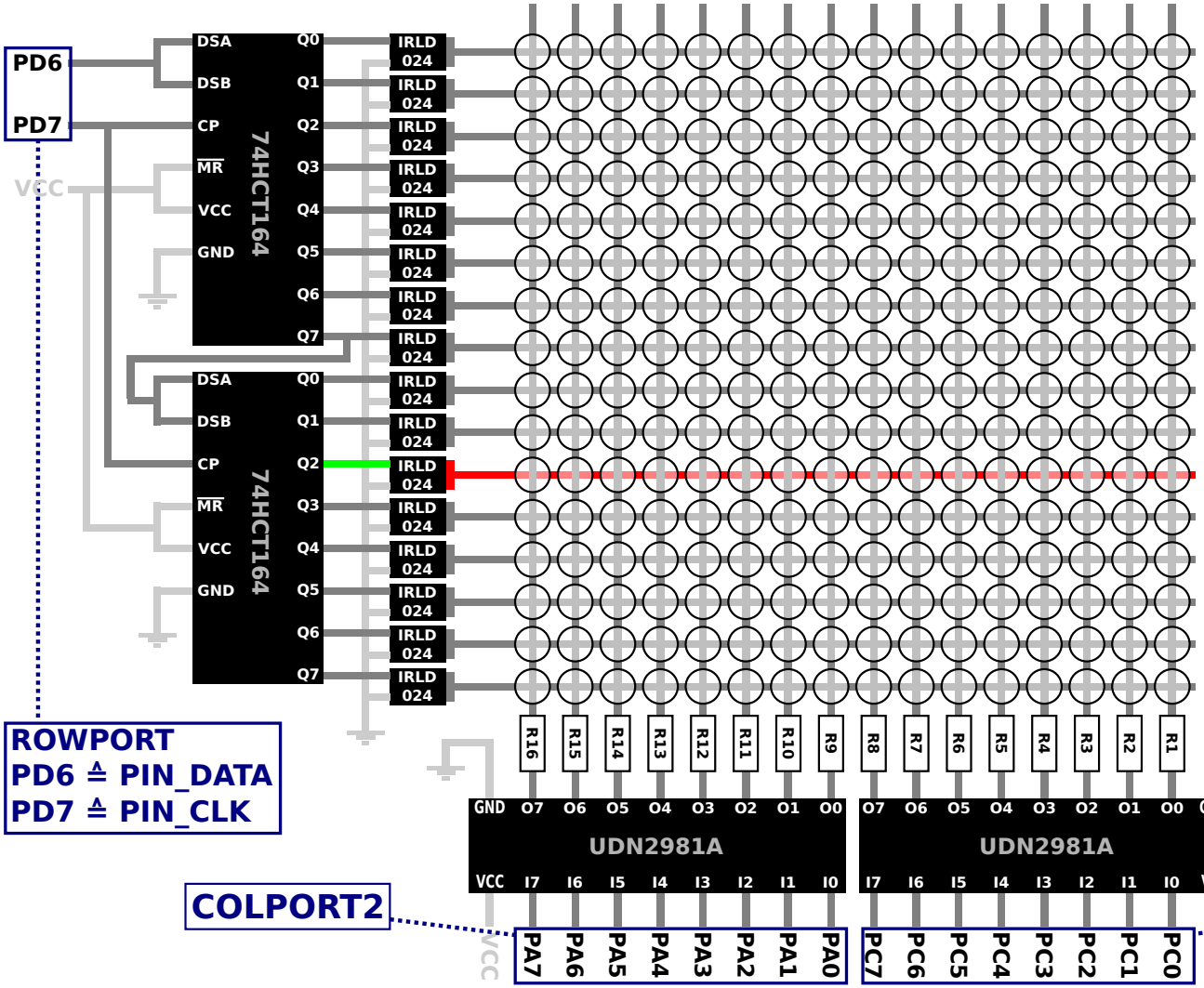

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 10
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



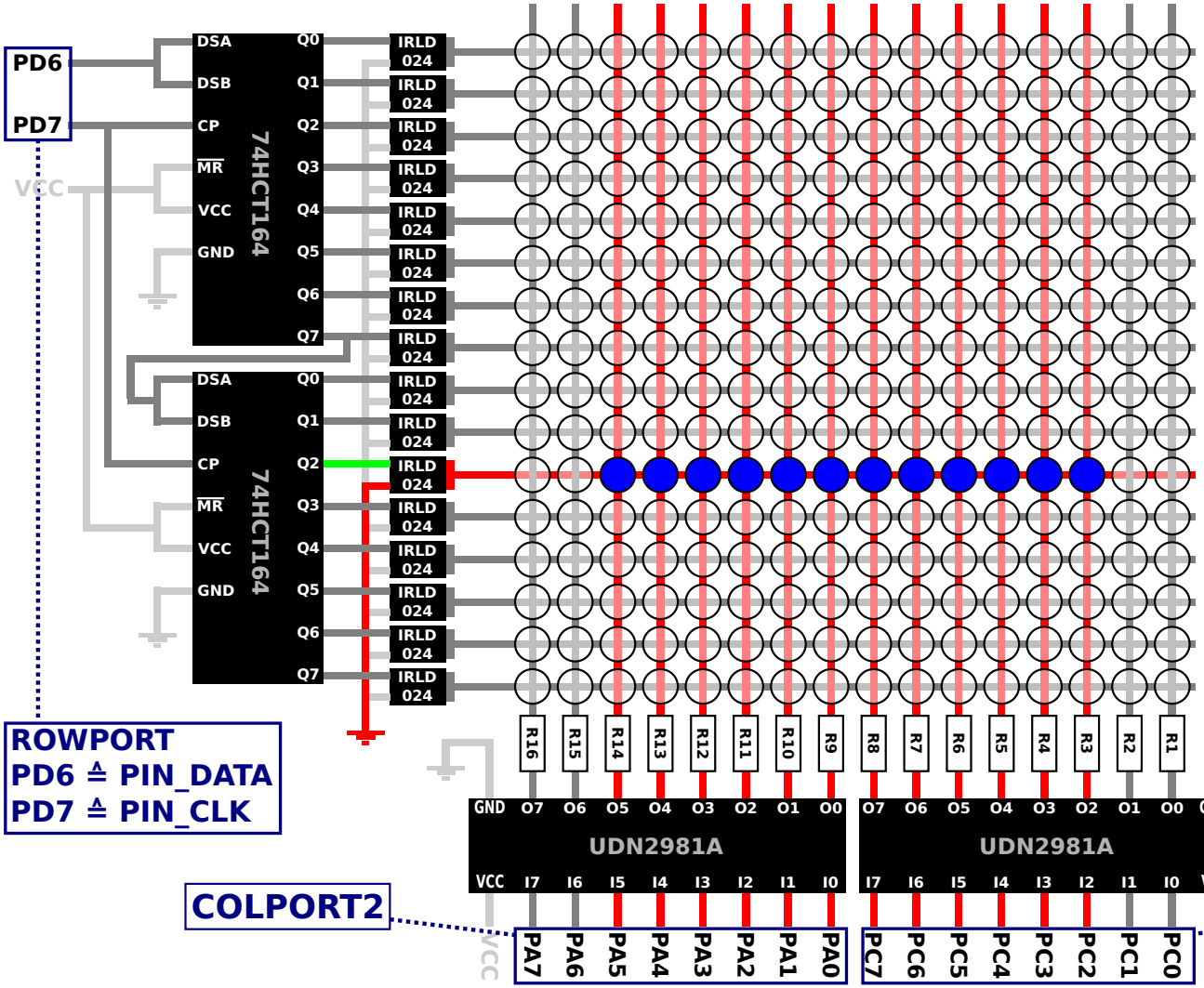

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 10
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



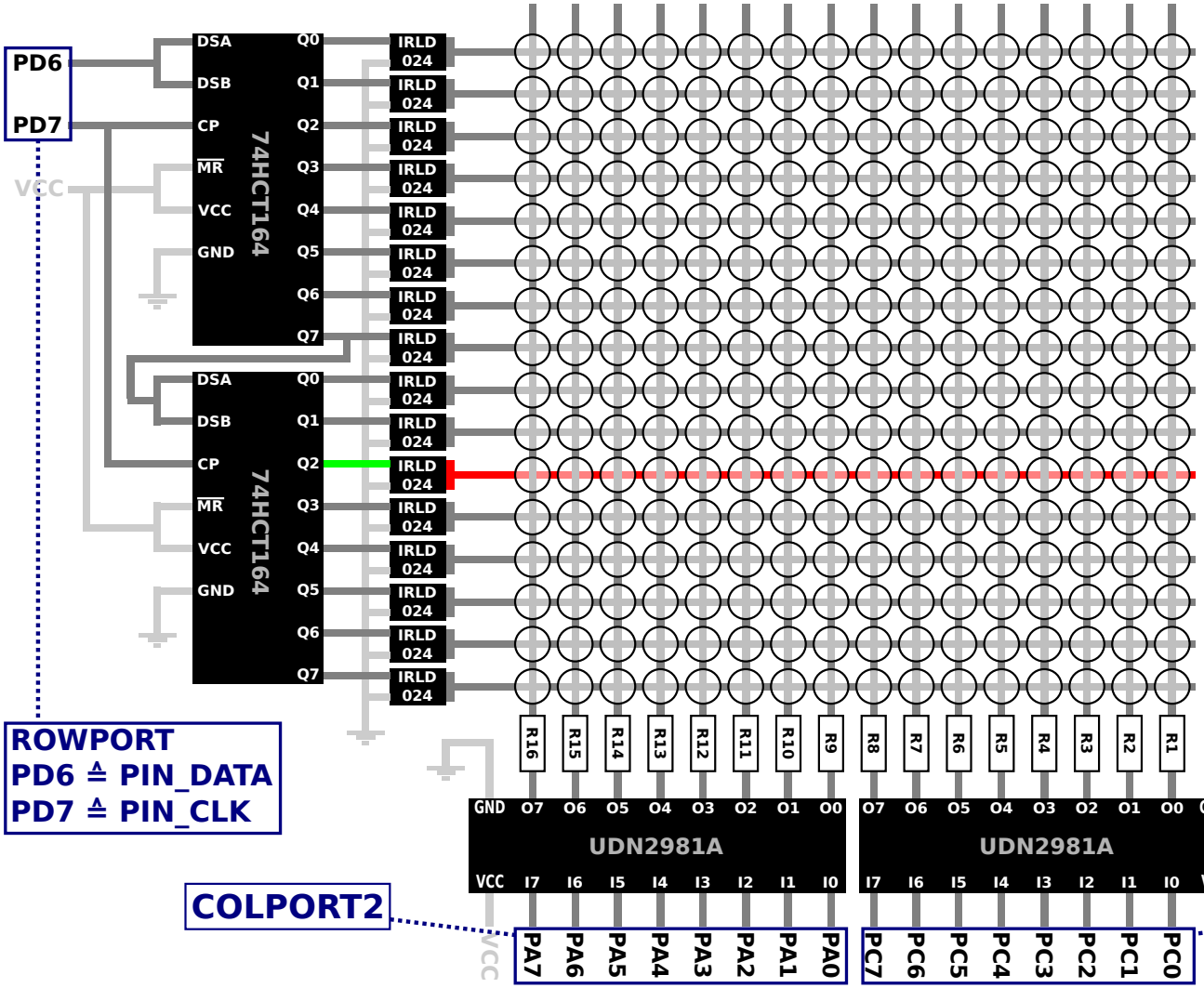

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 10
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



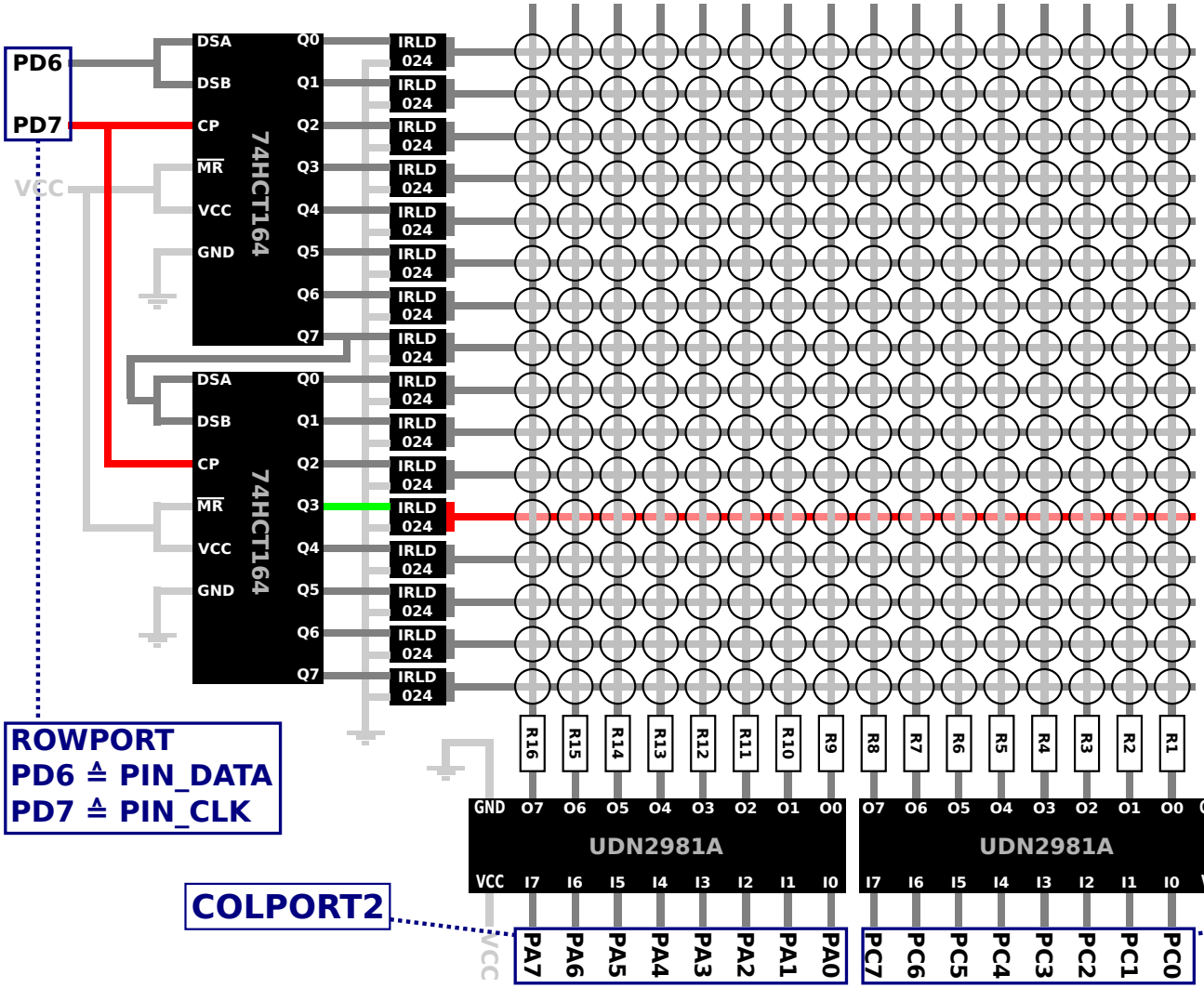

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 10
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3F  CP1: 0xFC
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



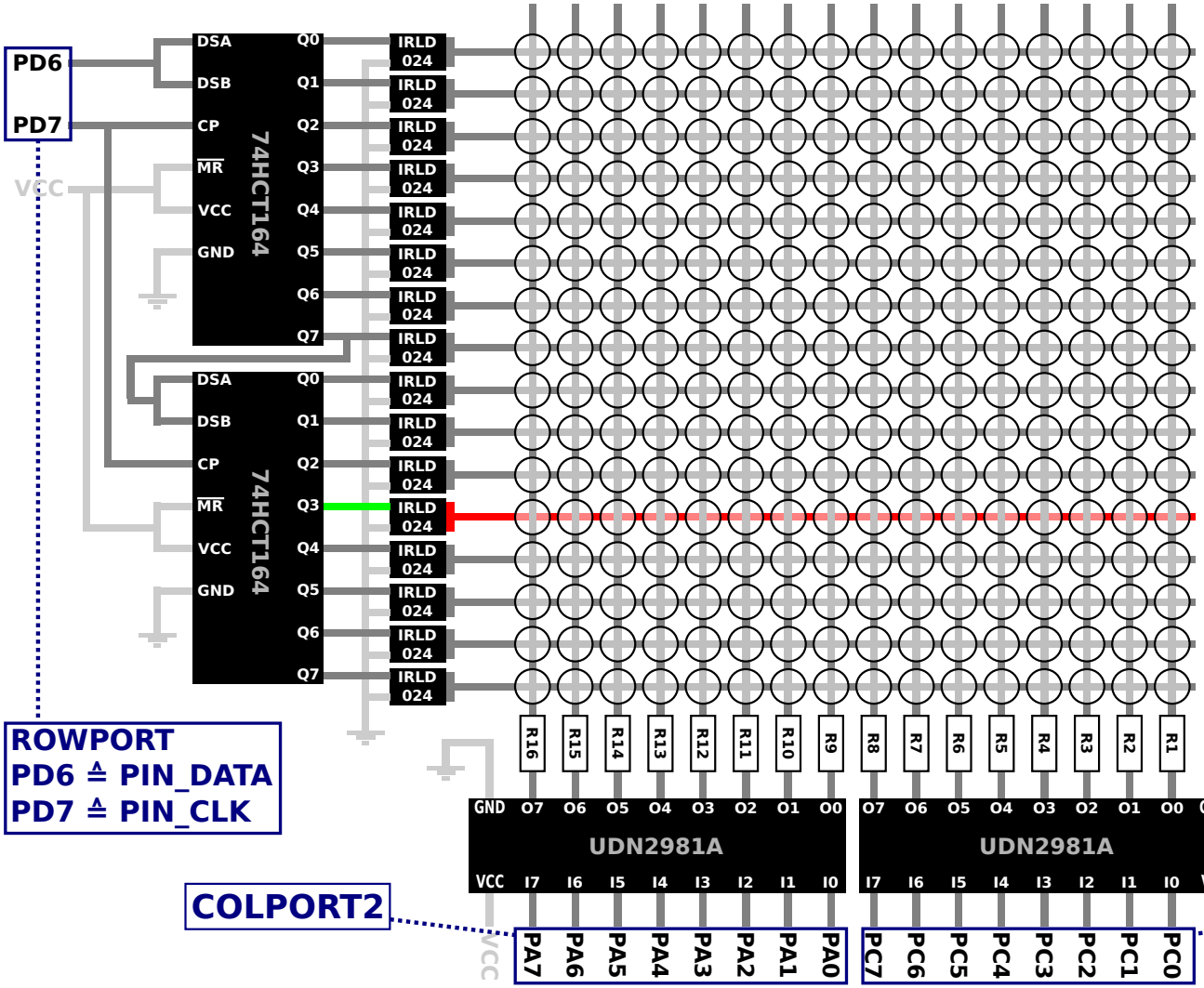

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 11
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3E  CP1: 0x7C
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



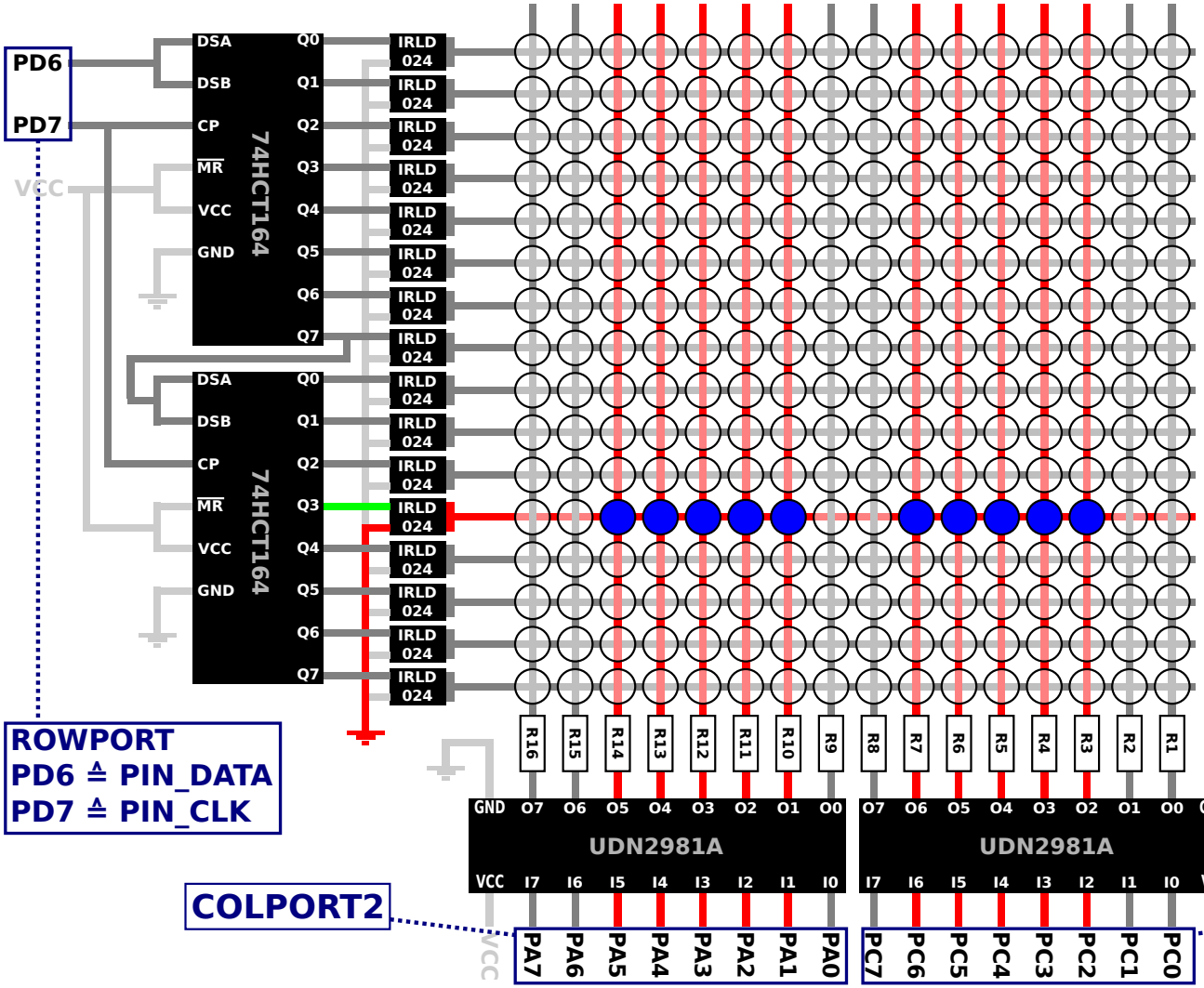

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 11
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3E  CP1: 0x7C
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



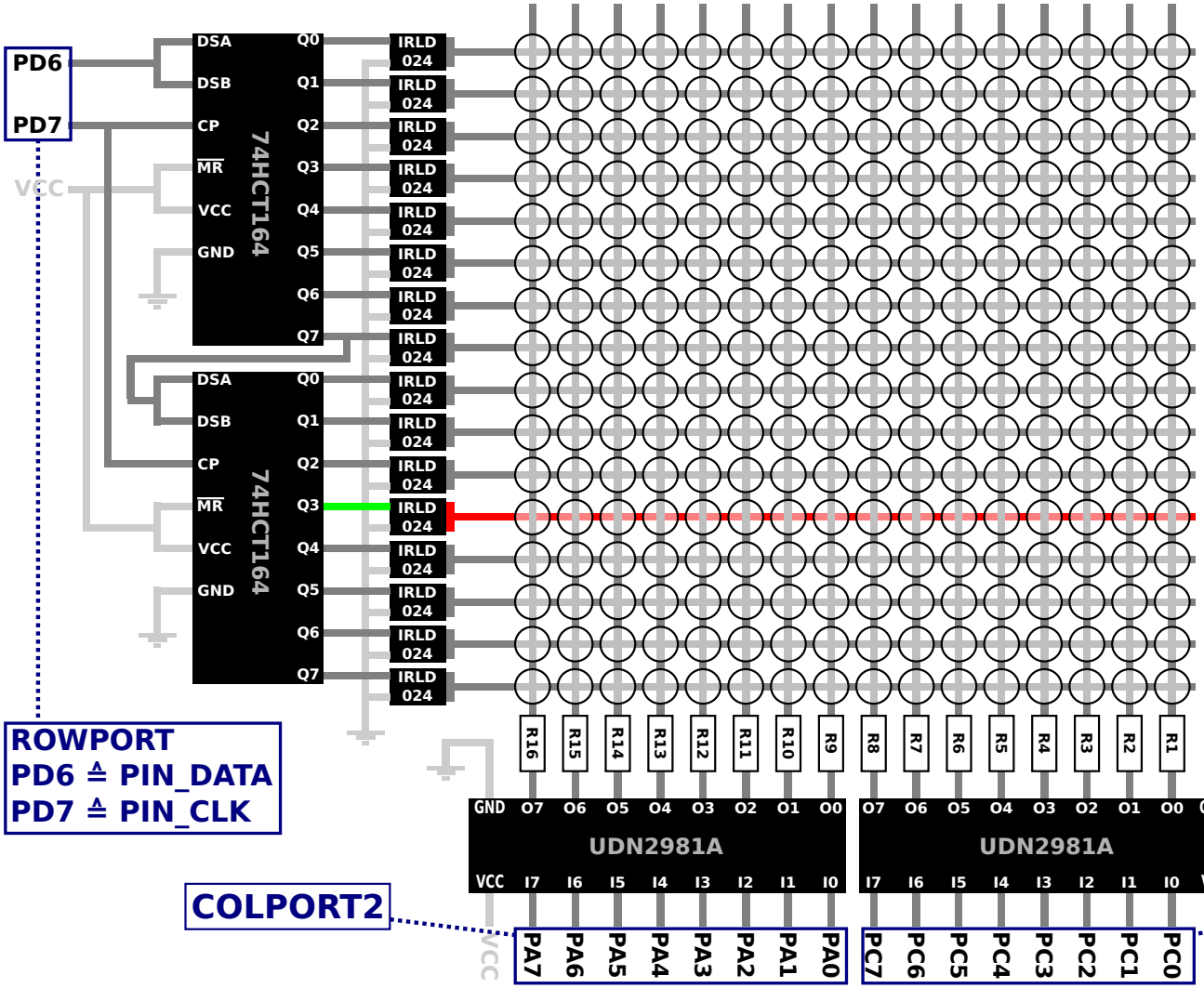

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 11
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3E  CP1: 0x7C
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



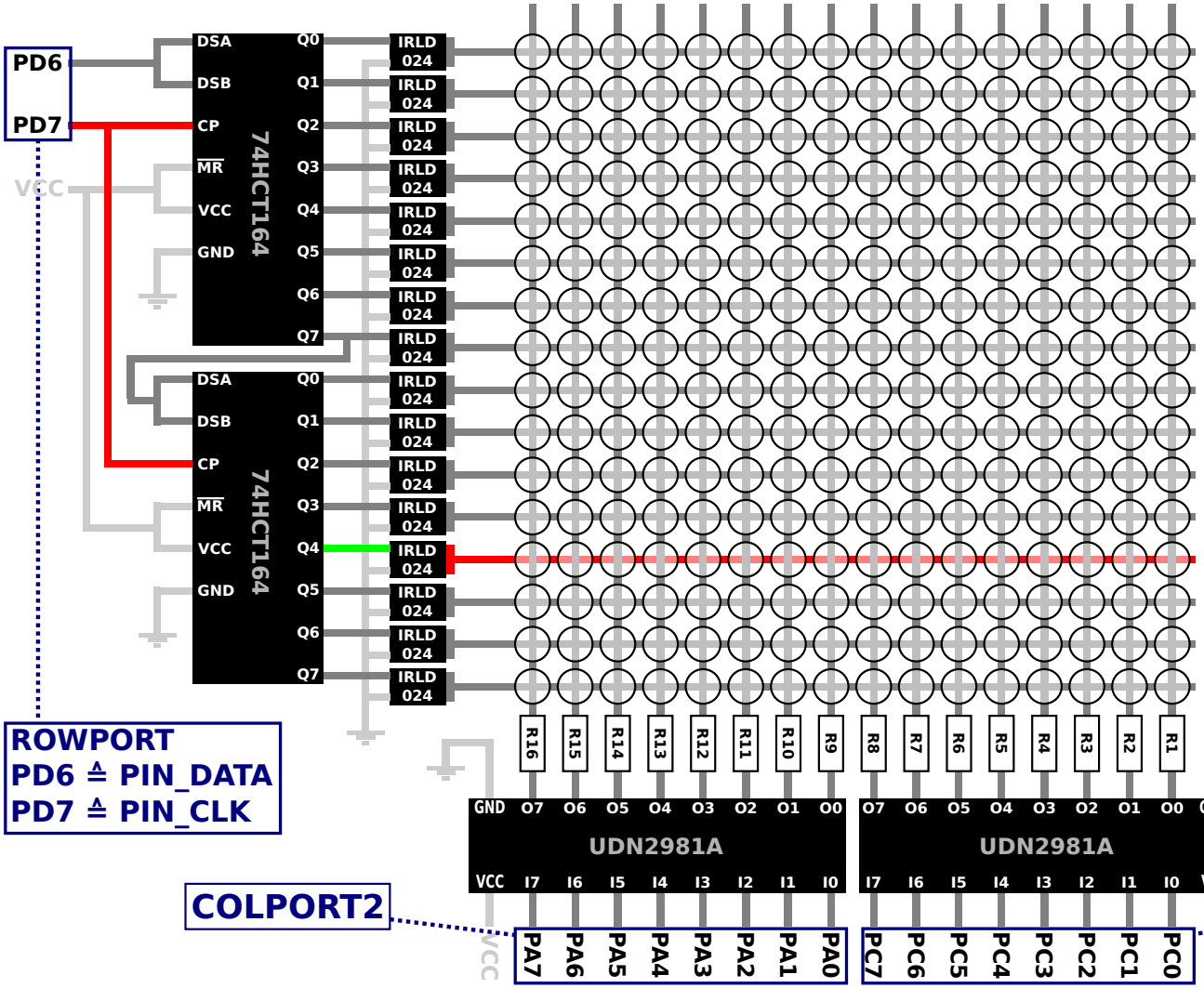

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 11
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x3E  CP1: 0x7C
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



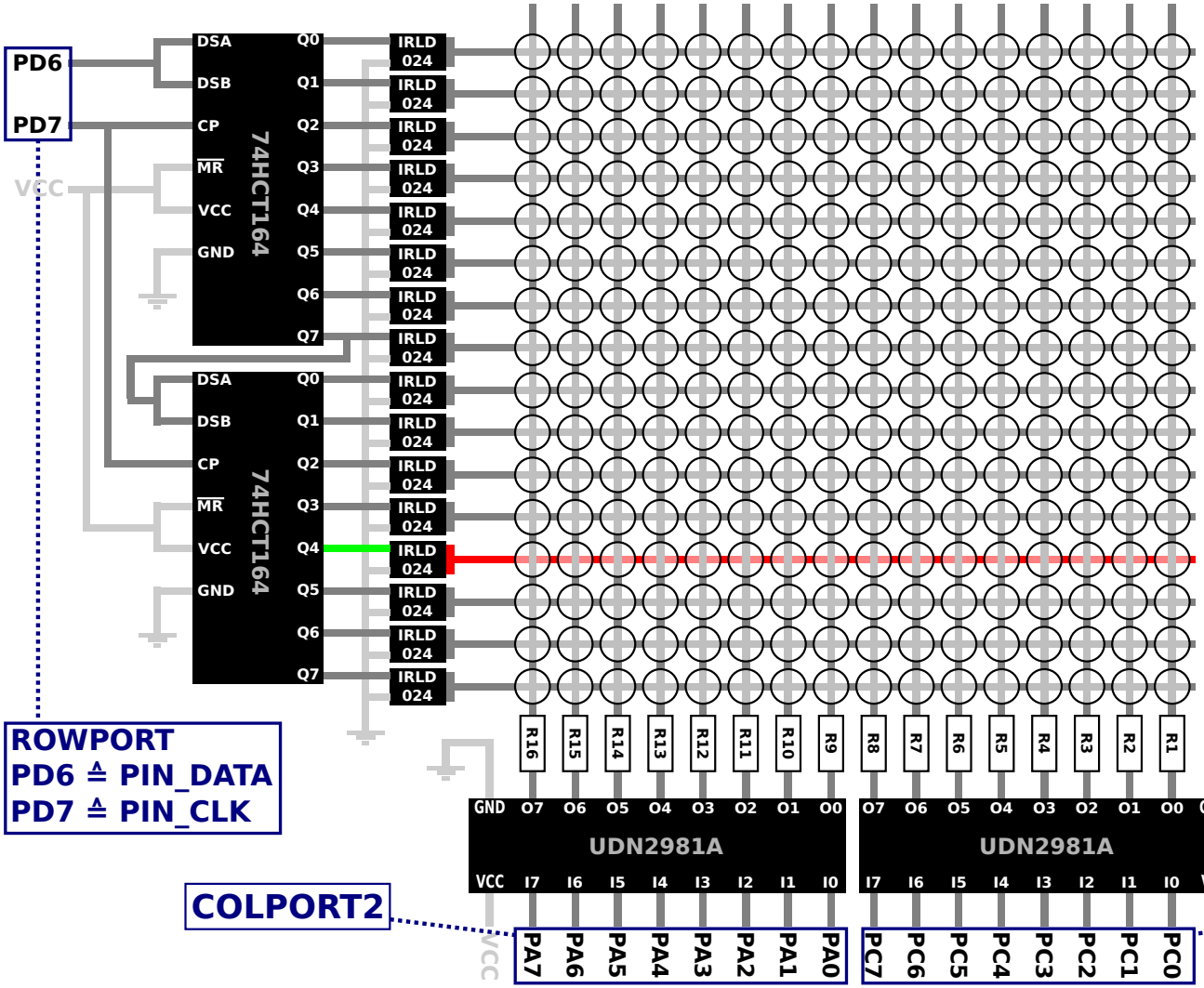

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 12
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x78  CP1: 0x1E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



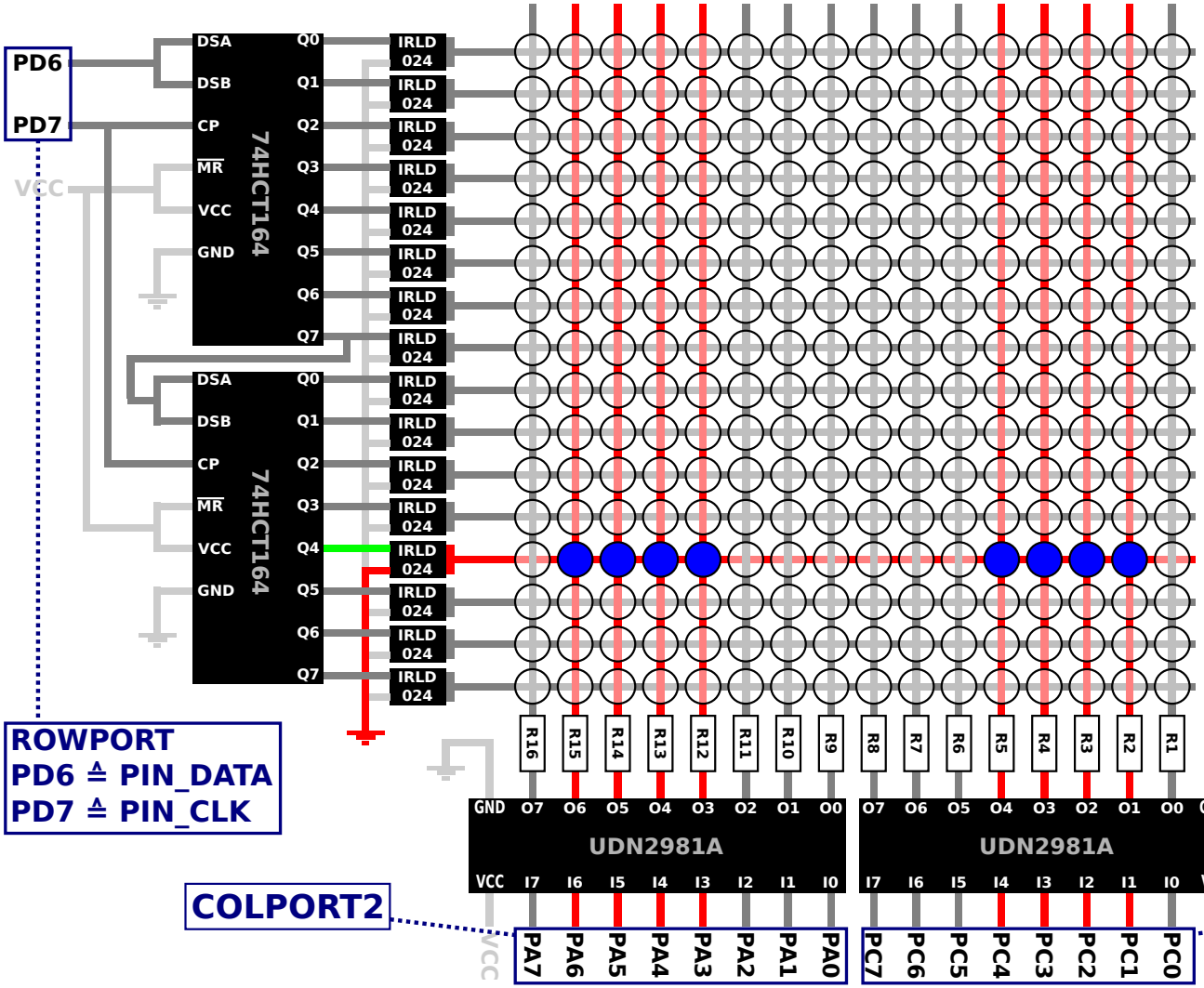

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 12
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x78  CP1: 0x1E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



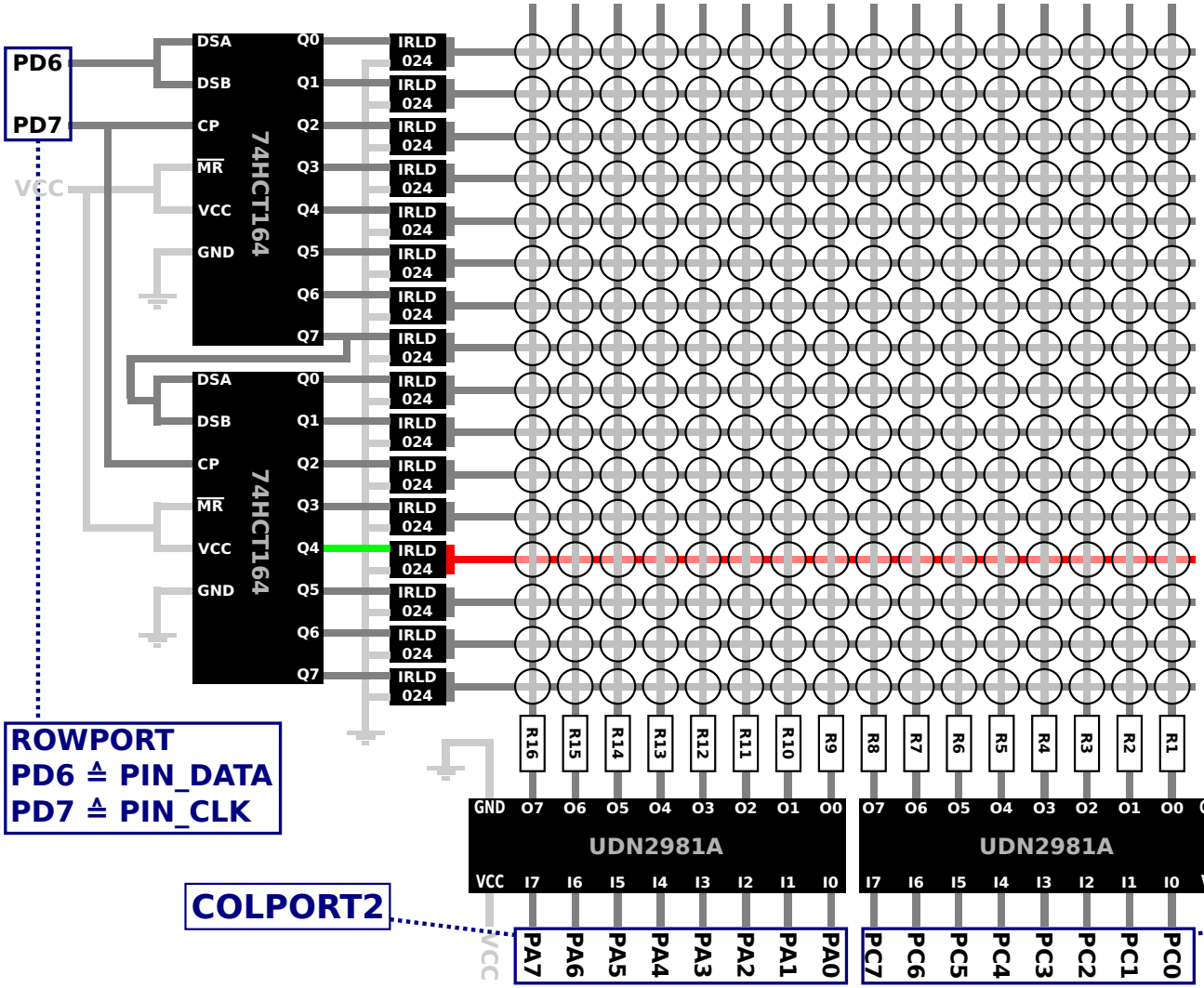

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 12
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x78  CP1: 0x1E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



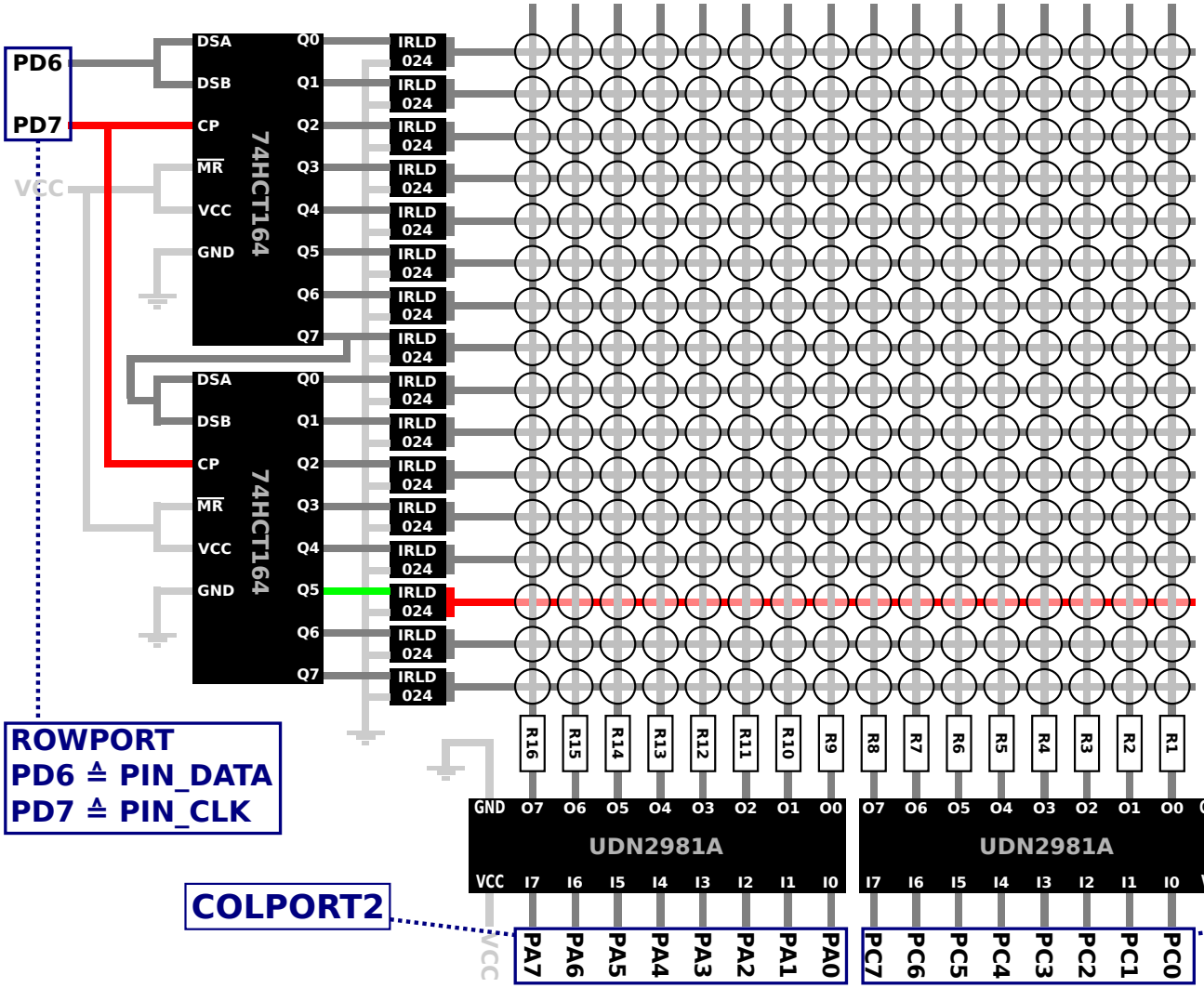

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 12
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x78  CP1: 0x1E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



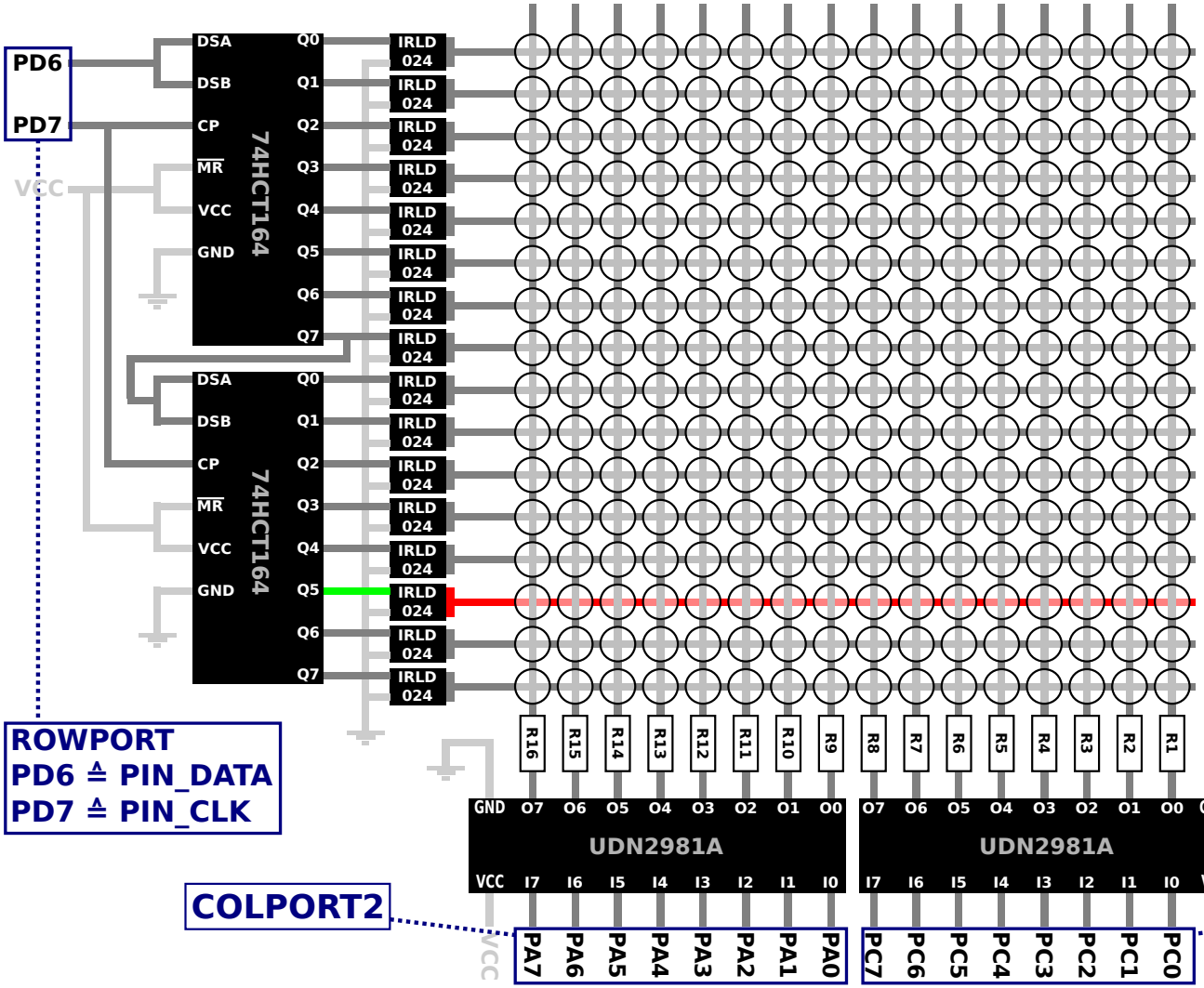

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 13
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x70  CP1: 0x0E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



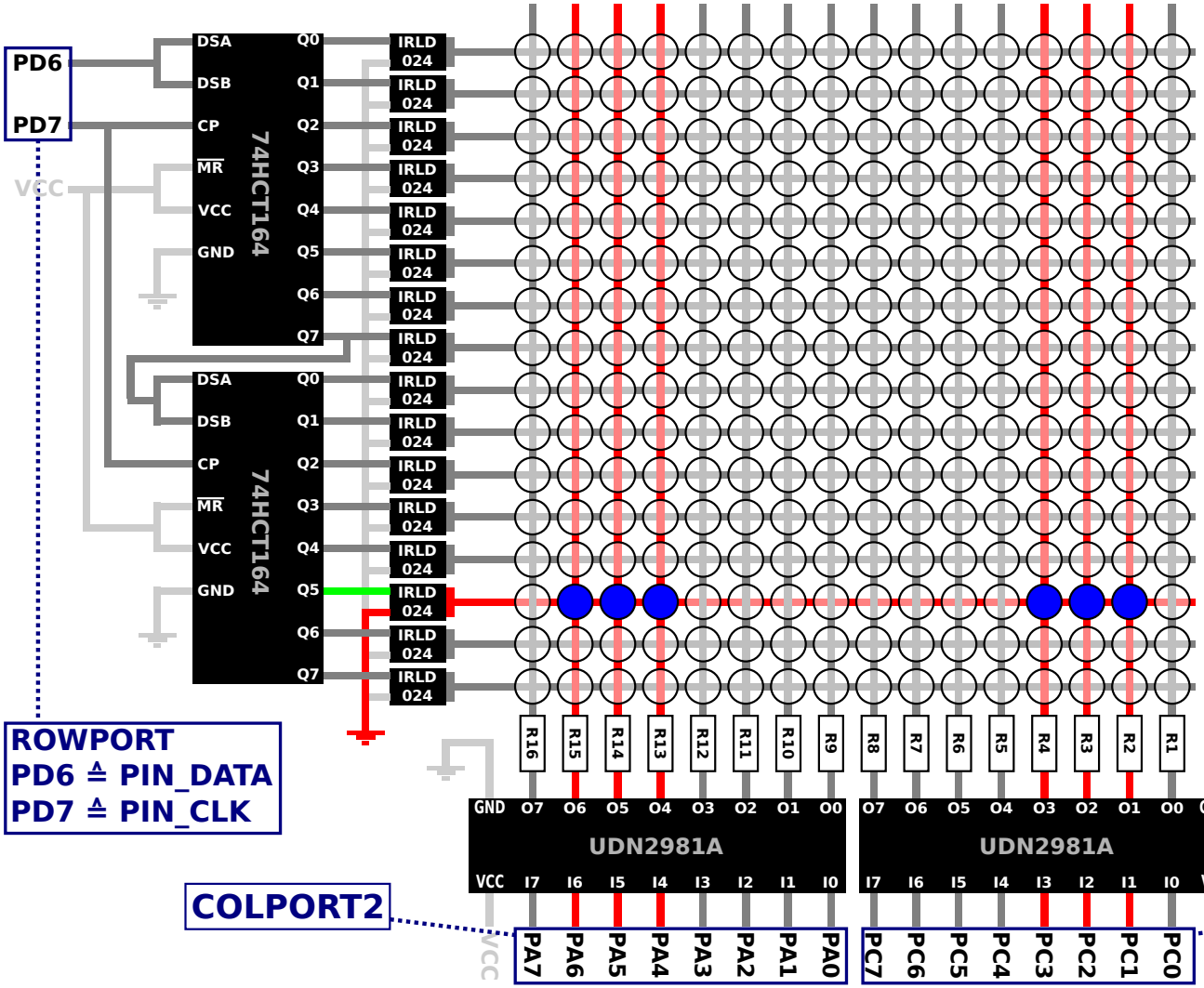

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 13
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x70  CP1: 0x0E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



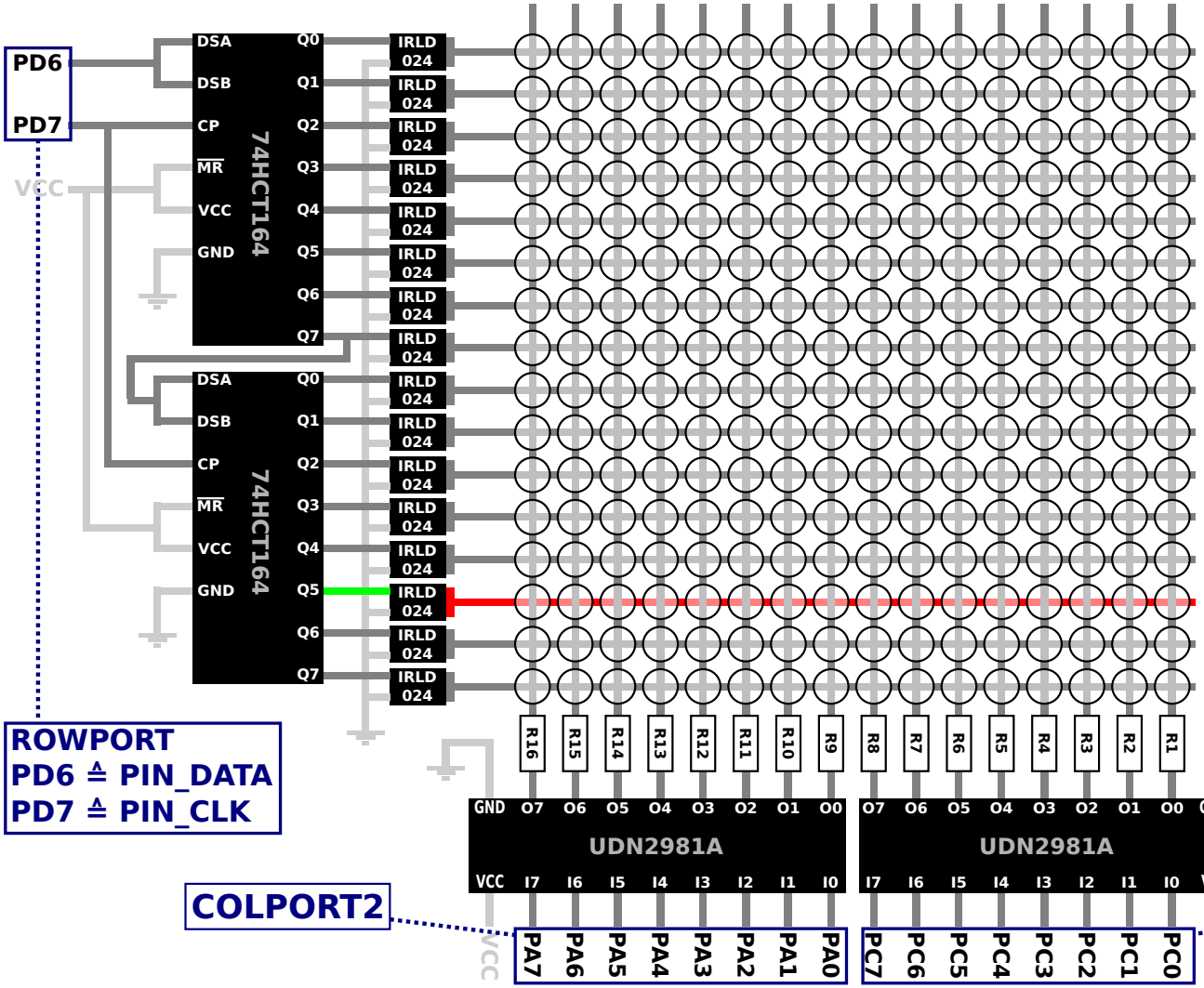

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 13
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x70  CP1: 0x0E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



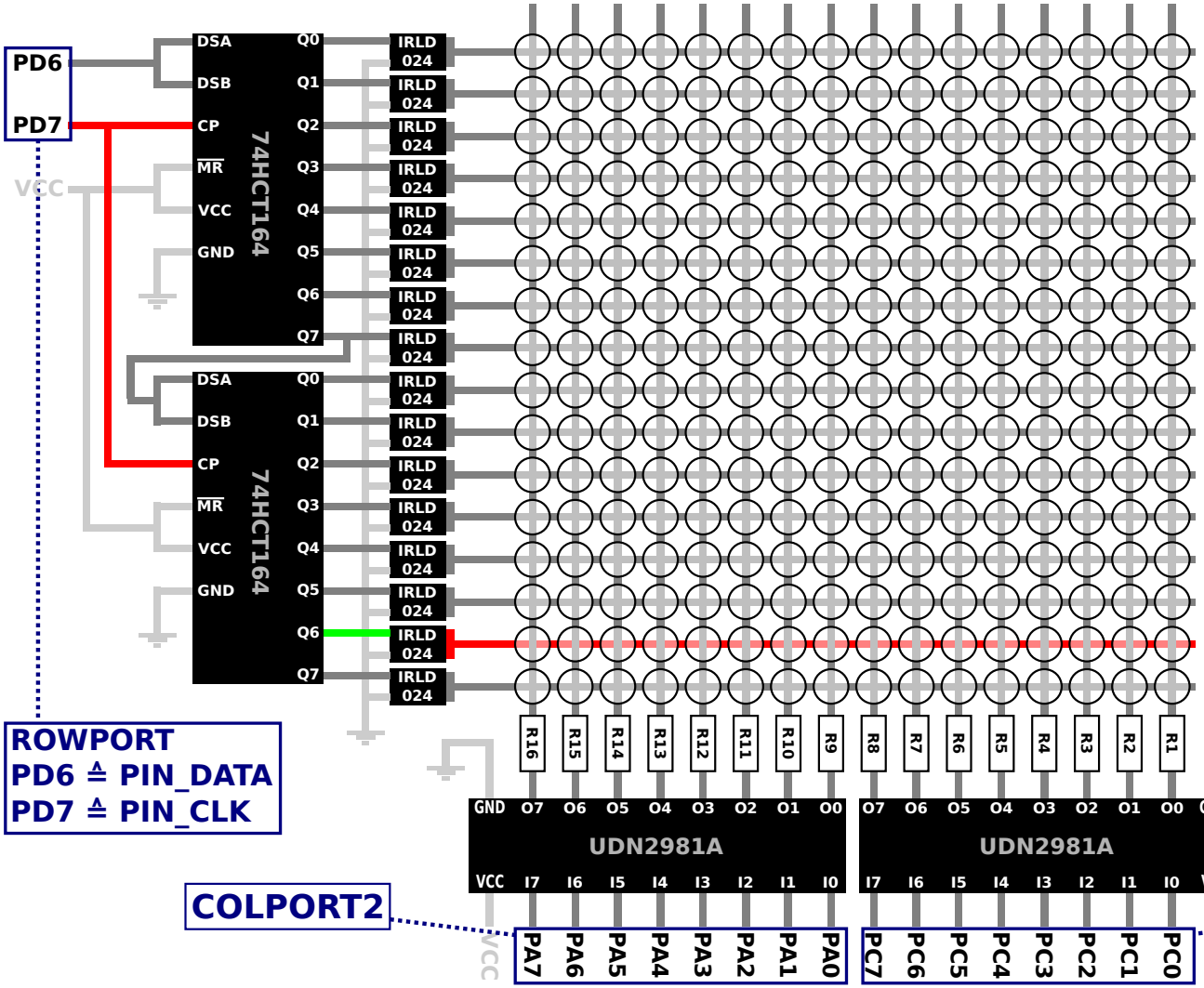

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 13
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x70  CP1: 0x0E
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



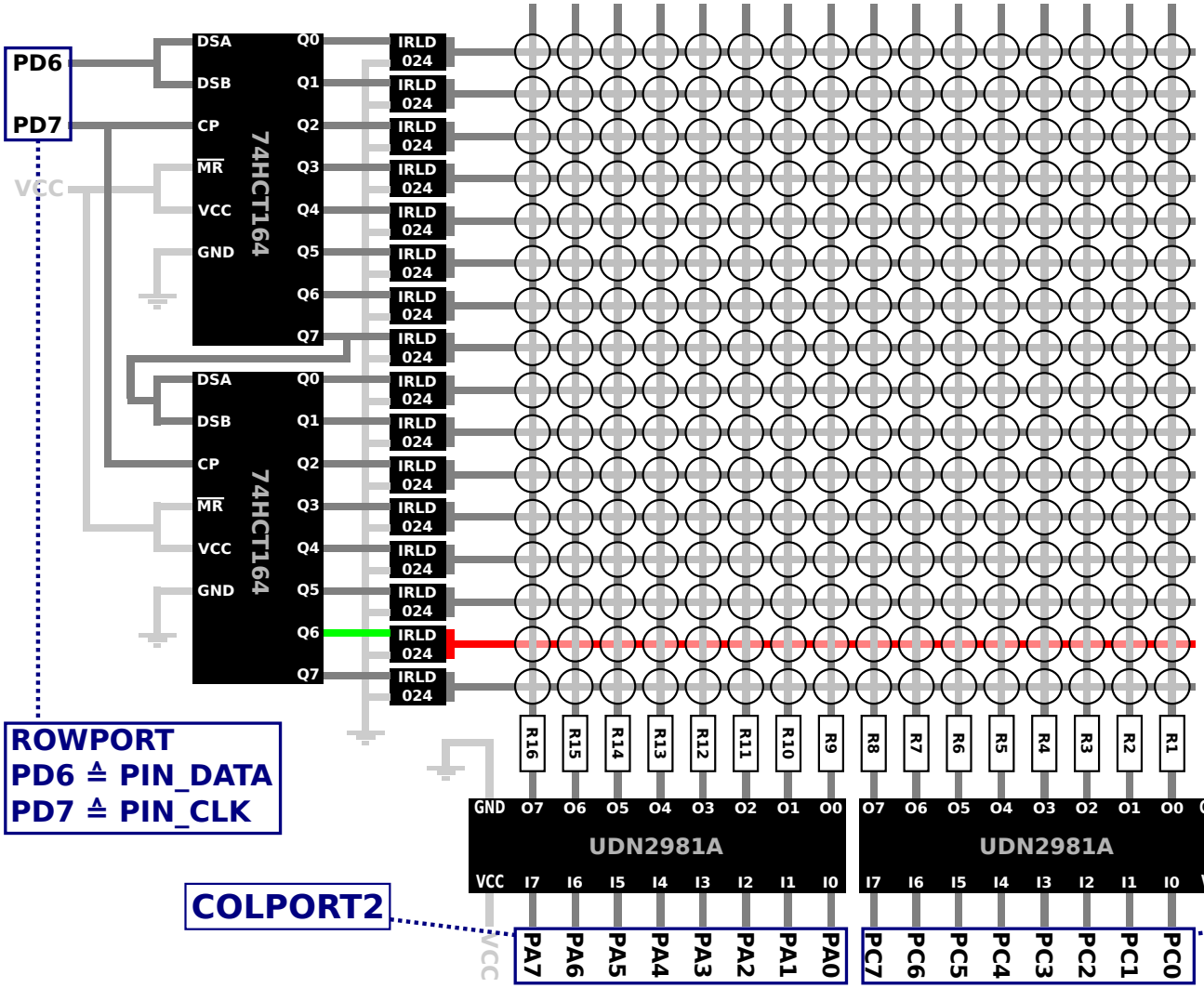

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 14
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x60  CP1: 0x06
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



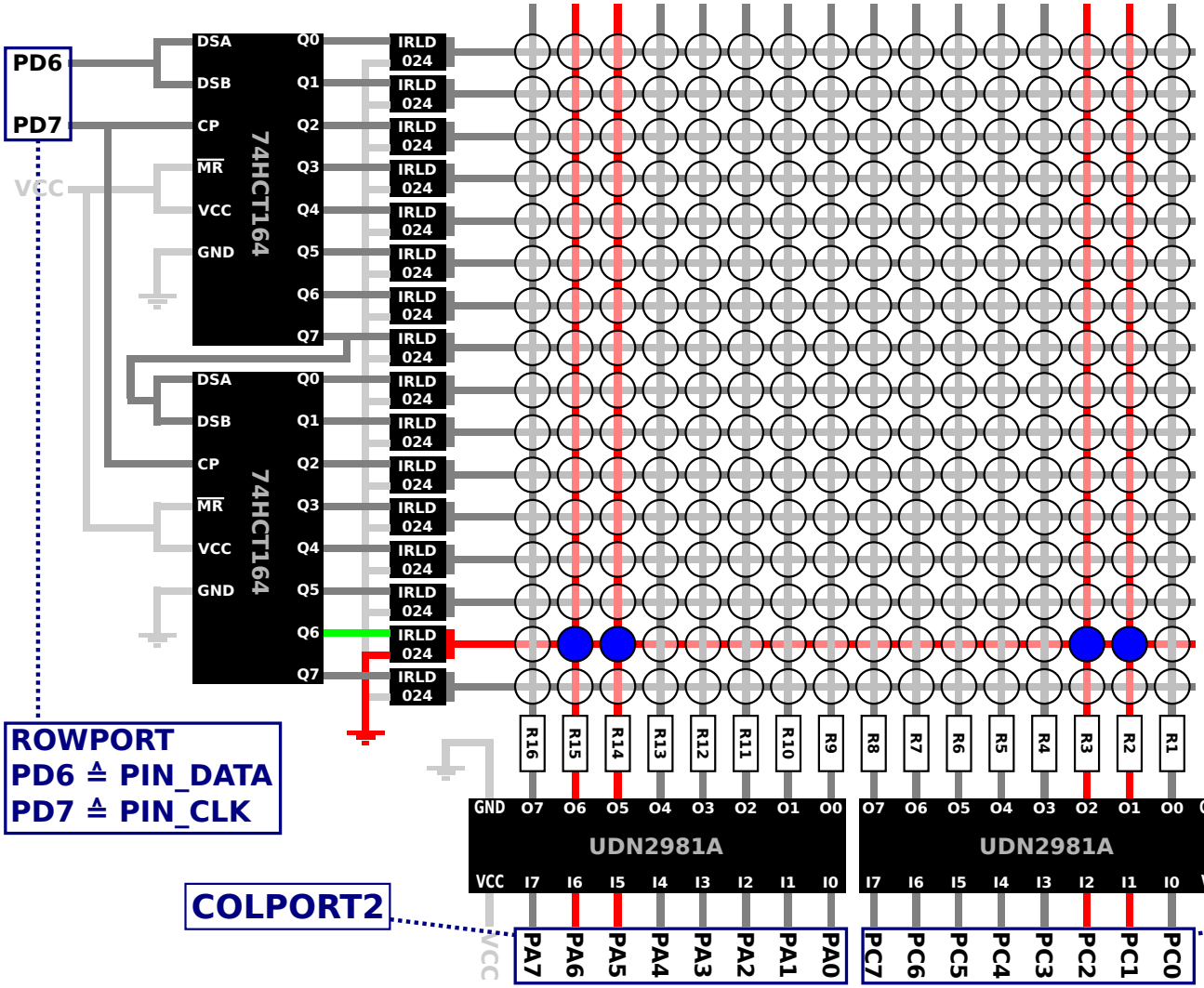

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 14
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x60  CP1: 0x06
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



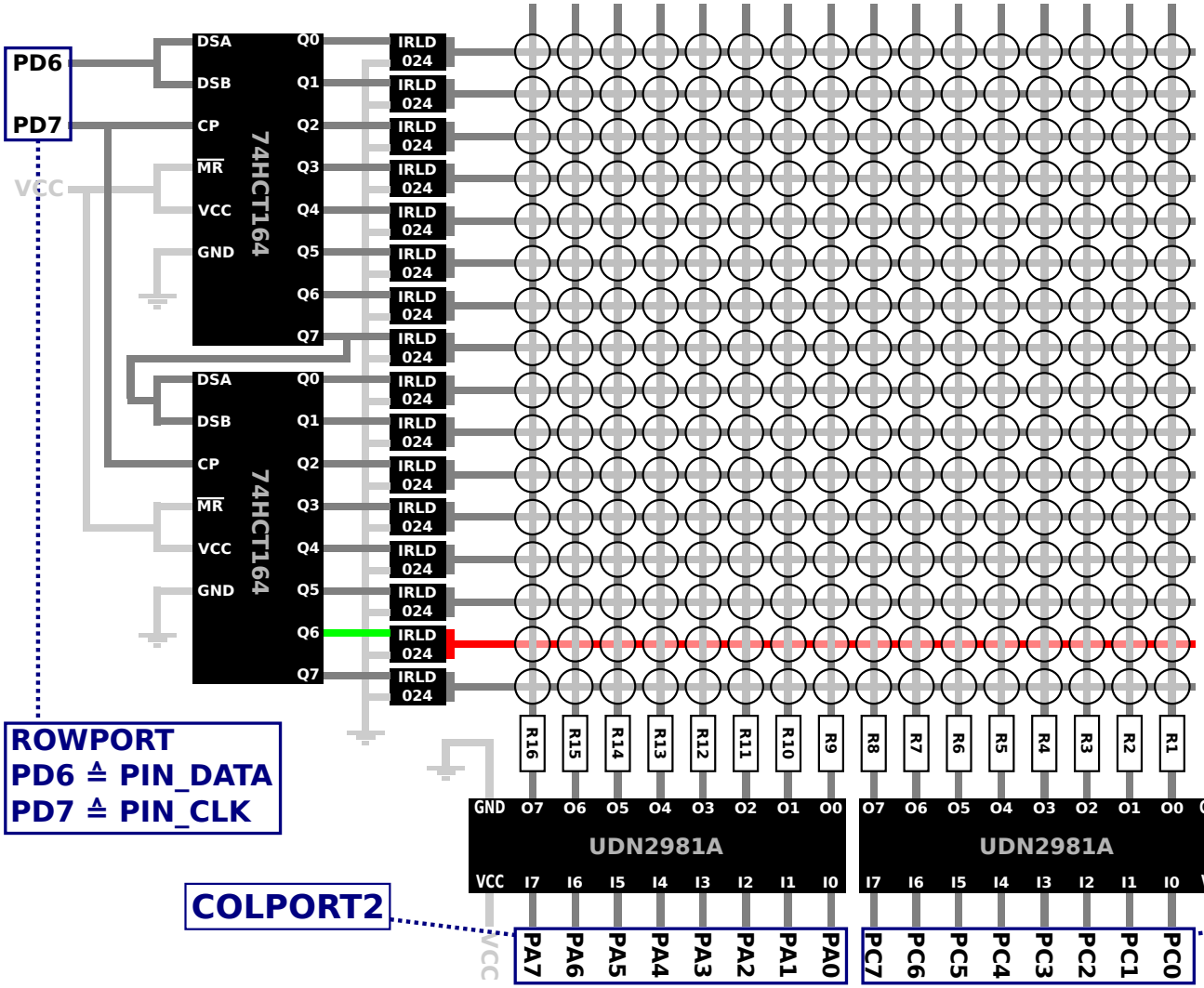

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 14
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x60  CP1: 0x06
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



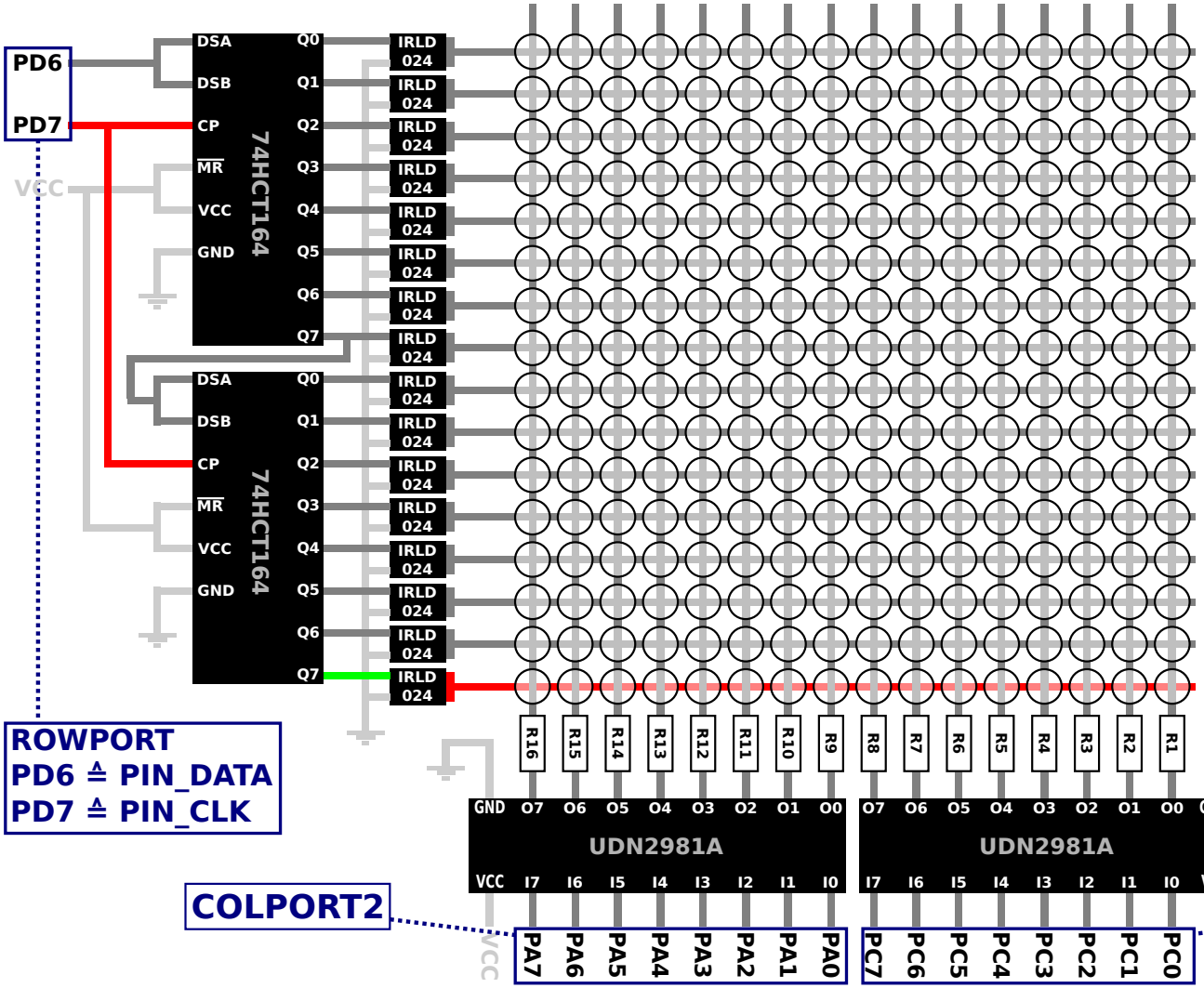

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 14
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x60  CP1: 0x06
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



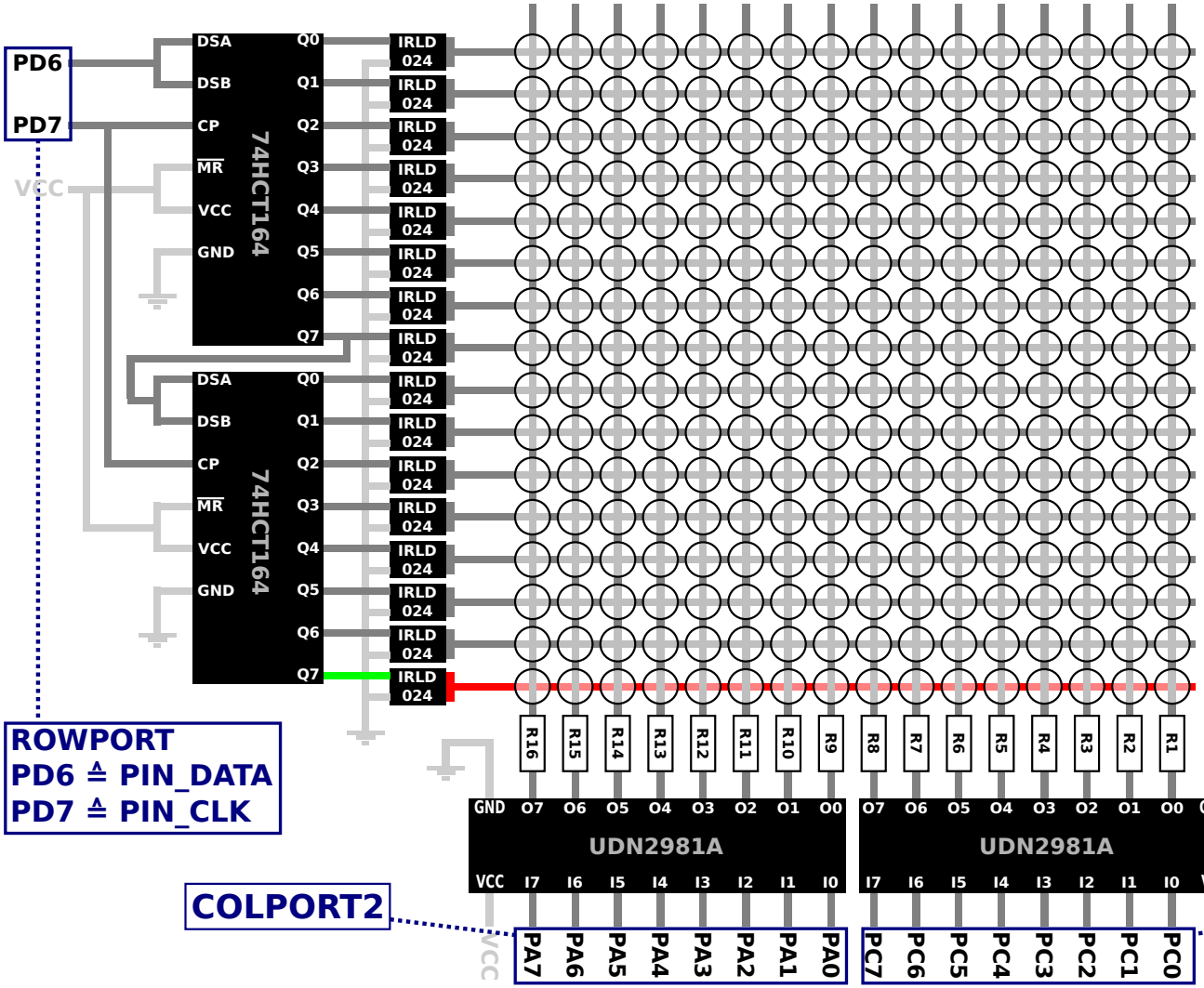

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 15
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xC0  CP1: 0x03
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



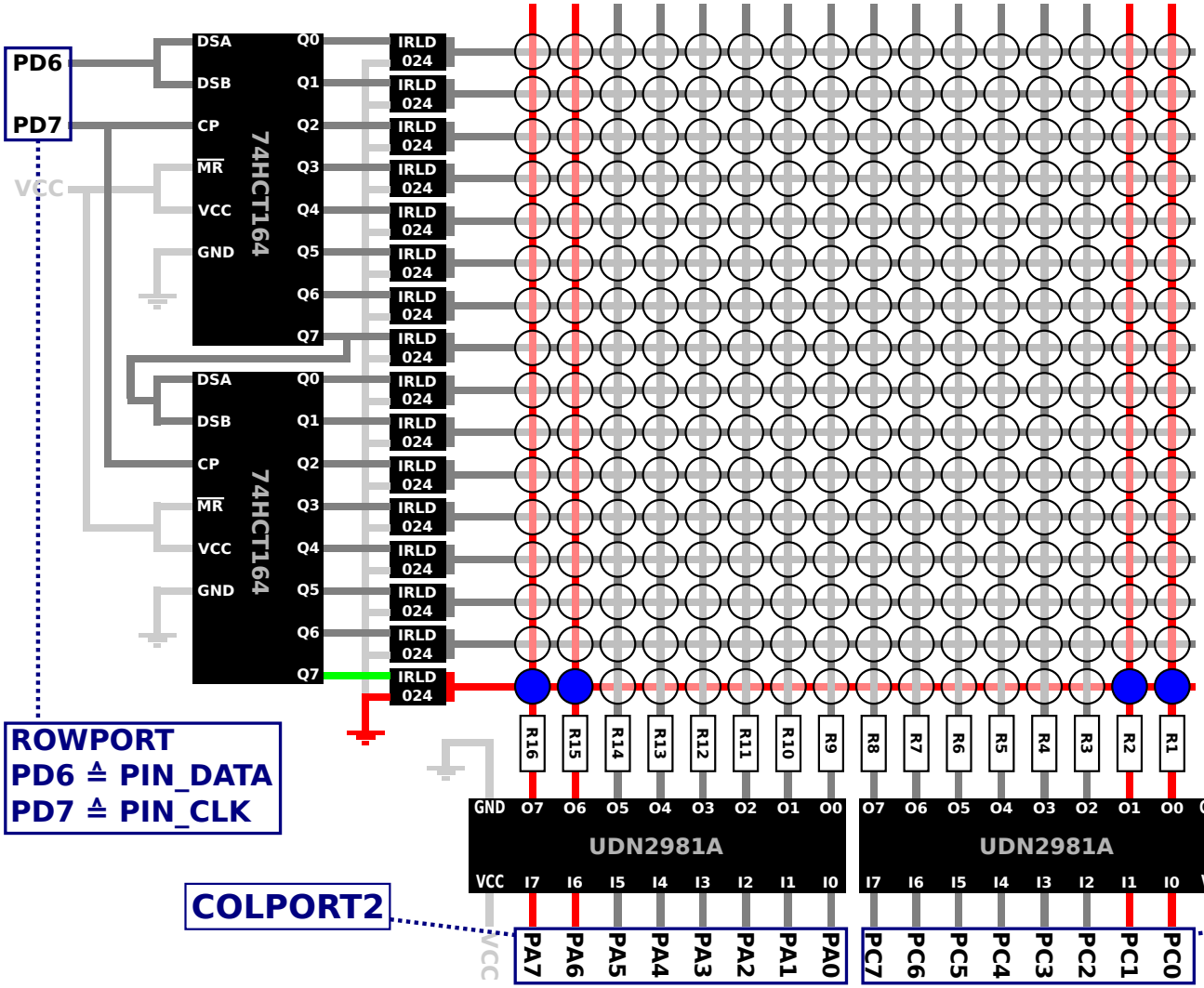

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 15
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xC0  CP1: 0x03
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix



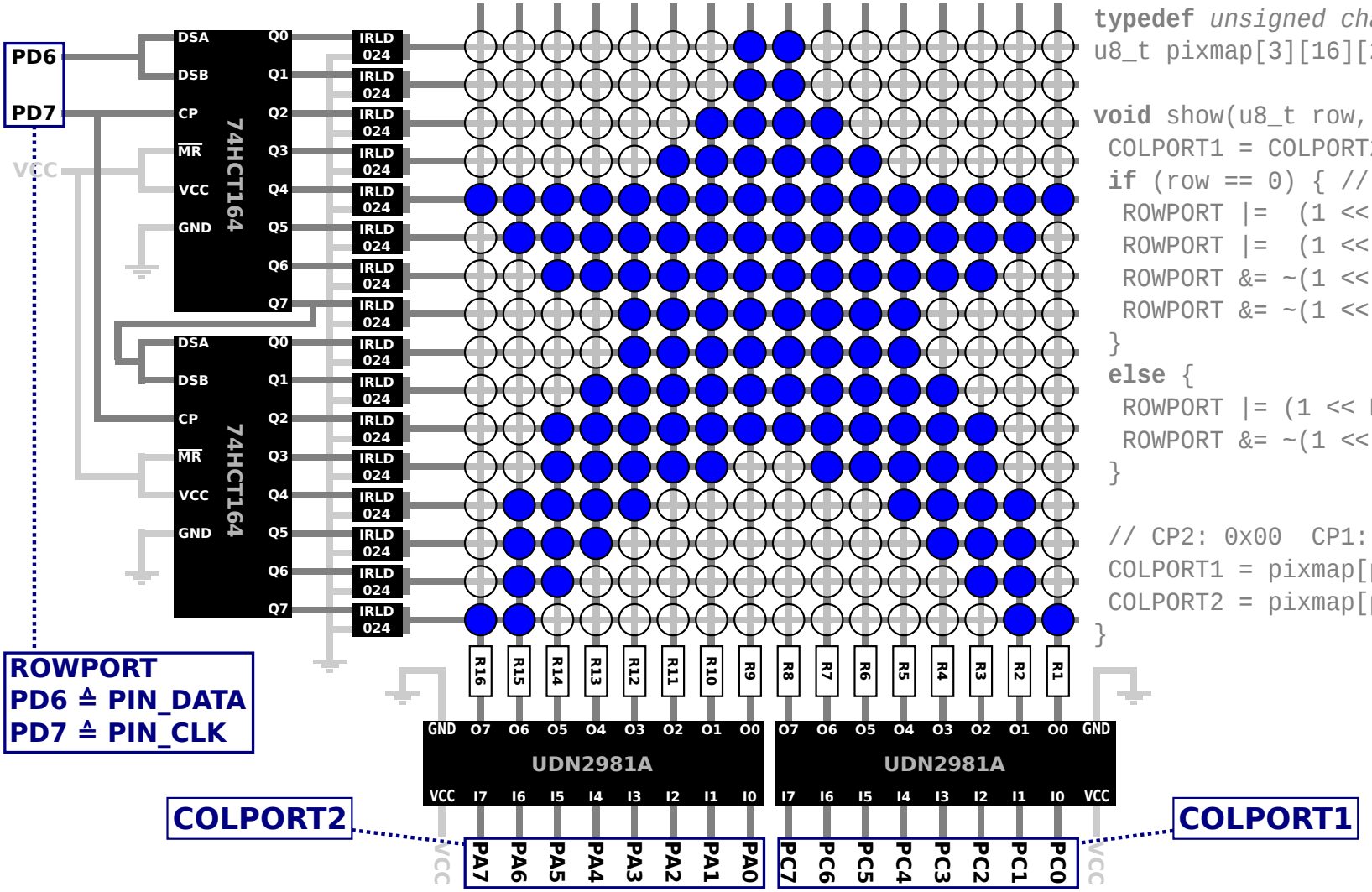

```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 15
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xC0  CP1: 0x03
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix




```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 15
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0xC0  CP1: 0x03
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```

Die LED-Matrix




```
typedef unsigned char u8_t;
u8_t pixmap[3][16][2];

void show(u8_t row, u8_t p) {
 COLPORT1 = COLPORT2 = 0;
 if (row == 0) { // row: 0
  ROWPORT |=  (1 << PIN_DATA);
  ROWPORT |=  (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_DATA);
 }
 else {
  ROWPORT |= (1 << PIN_CLK);
  ROWPORT &= ~(1 << PIN_CLK);
 }

 // CP2: 0x00  CP1: 0x00
 COLPORT1 = pixmap[p][row][0];
 COLPORT2 = pixmap[p][row][1];
}
```