Charlie Plexing

Student Group

First Name	Surname	Matrikel Nr.

Table of Contents

Charlie Plexing

Charliplexing ist eine möglichkeit durch geschickte Verwendung der drei Zustände Low, $High$ und Z (= Tri-state) aus durch wenige Pins viele LEDs anzusteuern. Dabei wir ausgenutzt, dass in den hochohmigen Pin, welcher auf Z “liegt”, kein Strom fließen kann und dieser auch keinen Spannungswert definiert.

Beispiel eines Charlieplexing

In figure 1 ist ein Beispiel für Charlieplexing dargestellt. es ist folgendes zu sehen:

- Links ist die gewünschte Outputmatrix zu sehen, in welchem durch Druck auf H bzw. L die gewünschte LED aktiviert bzw. deaktiviert werden kann.
- In der Mitte ist die Logik zu sehen, auf die im gleich nochmals eingegangen wird.
- Rechts ist die LED Matrix zu finden, welche durch Charlieplexing angesteuert wird.

Es soll nun die Logik etwas näher beschrieben werden. Dies kann auch durch Reduktion der Simulationsgeschwindigkeit besser in der Simulation sichtbar gemacht werden:

1. Es ist ersichtlich, dass jeweils für ein nebeneinander liegendes LED-Pärchen im Charlieplexing Anode und Kathode vertauscht sind.
   Im Code werden deshalb immer zwei nebeneinander liegende Werte in der Outputmatrix gleichzeitig behandelt. (Siehe Zeile 141ff in `CharlieP.c`) Dabei ist für die LED links oben ([0][0]) die Kathode gerade der Pin des Ports $PORTx.0$.
2. UM eine LED leuchten zu lassen wird ein bestimmter Pin eines Ports auf $High$ gesetzt, z.B. $PORTx.0$. Für diesen Pin muss auch das Datenrichtungsregister auf $High$ gesetzt werden, dies geschieht durch das $ODER$-Gatter vor dem $DDRx.0$ Pin.
   Im Code wird dafür im Zeile 158 und 159 die `portMatrix` gefüllt.
3. Für die LEDs in der ersten Reihe ist $PORTx.0$ entweder die Kathode (jede zweite LED ab [0][0]) oder Anode (jede zweite LED ab [0][1]).
   Gleiches gilt für die zweite Reihe für $PORTx.1$. Für die dritte Reihe gilt dies nur für die ersten beiden LEDs.
   Im Code schlägt sich dies in der Ermittlung der `actCathode` für ein LED-Pärchen in Zeile 145 bzw. 148 nieder.

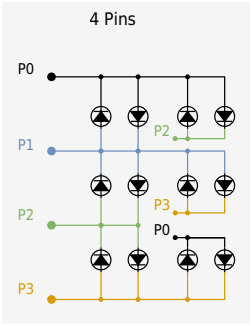
Fig. 1: Beispiel eines Charlieplexing

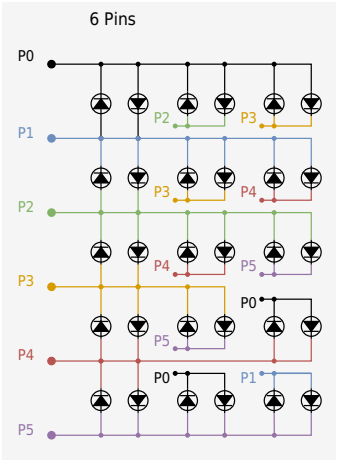
Lib

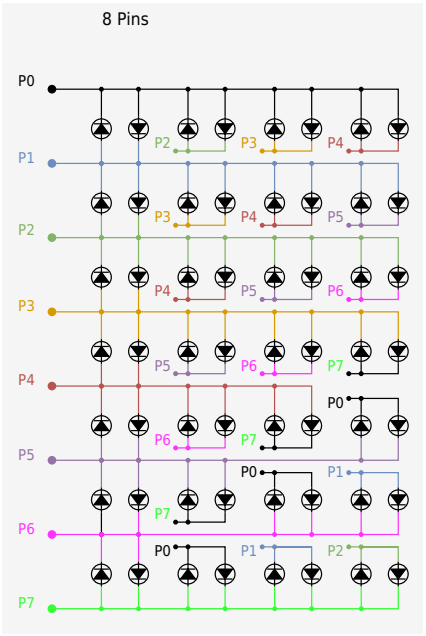
charlieplexing.zip

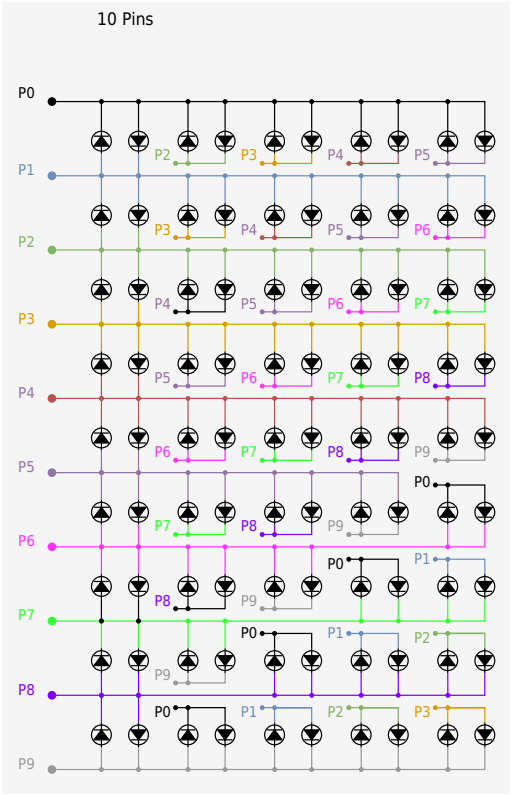
Wichtig:

- Die vorliegende Lib (und damit der Code unten) ist nur für eine gerade Anzahl von Pins geeignet.
- Zudem muss die Anordnung des Charlieplexings wie im folgenden Bild dargestellt eingehalten werden.









Text is not SVG - cannot display

Code

CharlieP.h

```
/*
 * CharlieP.h
 *
 * Created: 10.12.2022 22:30:57
 *  Author: Tim Fischer
 */


#ifndef CHARLIEP_H_
#define CHARLIEP_H_

//Makros for bit manipulation
#define SET_BIT(BYTE, BIT)      ((BYTE) |= (1<<BIT))
// set    single BIT in BYTE
#define CLR_BIT(BYTE, BIT)      ((BYTE) &= ~(1<<BIT))
// clear  single BIT in BYTE
#define CHG_BIT(BYTE, BIT, VAL) ((BYTE) = ((BYTE) & ~(1<<BIT)) | (VAL<<BIT))
// change single BIT in BYTE to VAL
#define TGL_BIT(BYTE, BIT)      ((BYTE) ^=  (1 << (BIT)))
// toggle single BIT in BYTE


#define PIN_COUNT       12
```

```
// overall number of pins used
#define PORT_COUNT      3
// number of ports used

enum    PORTS    {PORT_B, PORT_C, PORT_D, DDR_B, DDR_C, DDR_D};
// in this example three different Ports are used for output. This can also
be change to more or less
enum    P        {    PB,     PC,     PD};
// In this case, the matrizes outputPort, outputPin and portMatrix have to
be adapted

#define DONE            1
// return value for finished round over all anodes

//function prototypes
extern  uint8_t CharlieP_setPortsForLightingLeds(void);
extern void     CharlieP_setPortMatrix(void);

extern uint8_t outputMatrixXY   [PIN_COUNT]        [PIN_COUNT-1];
extern uint8_t portMatrix       [PORT_COUNT*2]    [PIN_COUNT];

#endif /* CHARLIEP-H_H_ */
```

CharlieP.c

```
/*
 * CharlieP.c
 *
 * Created: 10.12.2022 22:30:48
 *  Author: Tim Fischer
 *
 * The following code drives charlieplexed LED arrays. For this, the array
shall have the following structure:
 *
 *               colNr->     0   1      2   3
 *
 *   rowNr       PIN0  ----------------------------
 *     v                     |   |      |   |
 *                           |   |      ^   v   . . . . . . . . . . . . This
is LED[0,3]
 *     0                     ^   v      |   |
 *                           |   |      ------PIN2
 *                           |   |
 *               PIN1  ----------------------------
 *                           |   |      |   |
 *                           |   |      ^   v
 *     1                     ^   v      |   |
 *                           |   |      ------PIN3
 *                           |   |
```

```
 *               PIN2  ----------------------------
 *                           |   |
 *                           |   |      ------PIN0
 *     2                     ^   v      |   |
 *                           |   |      ^   v
 *                           |   |      |   |
 *               PIN3  ----------------------------
 *
 * This example is explained more:
 * - The numbers of pins is PIN_COUNT = 4.
 * - There are groups of independent LEDs. There are as much groups of
independent LEDs as numbers of pins.
 * - Example for independent LEDs:
 *      - assume only PIN0 is HIGH and all other pins are at high-impedance.
 *      - Then the anode on some LEDs is at 5V.
 *      - For LED[0,1], LED[0,3] and LED[2,3] anode is HIGH. These can
independently be activated, by switching their cathode to LOW.
 * - The active output anode has to be stepped through all PIN_COUNT pins.
 * - The arrangements of the ANODES can be read from the circuit above:
 *     rowNr, colNr->  0   1      2   3
 *      0            PIN1 PIN0  PIN2 PIN0
 *      1            PIN2 PIN1  PIN3 PIN1
 *      2            PIN3 PIN2   PIN3 PIN0
 *
 * - The arrangements of the CATHODE is just the groups of columns
rearranged (column 1 with 0, column 3 with 2)
 *   The ANODE becomes the CATHODE ans vice versa.
 * - For PIN_COUNT = 6, the  arrangements of the ANODES shall be as follows:
 *     rowNr, colNr->  0   1      2   3      4   5
 *      0            PIN1 PIN0  PIN2 PIN0  PIN3 PIN0
 *      1            PIN2 PIN1  PIN3 PIN1  PIN4 PIN1
 *      2            PIN3 PIN2  PIN4 PIN2  PIN5 PIN2
 *      3            PIN4 PIN3  PIN5 PIN3   PIN4 PIN0
 *      4            PIN5 PIN4   PIN5 PIN0  PIN5 PIN1
 *
 * - For PIN_COUNT = 8, the  arrangements of the ANODES shall be as follows:
 *     rowNr, colNr->  0   1      2   3      4   5      6   7
 *      0            PIN1 PIN0  PIN2 PIN0  PIN3 PIN0  PIN4 PIN0
 *      1            PIN2 PIN1  PIN3 PIN1  PIN4 PIN1  PIN5 PIN1
 *      2            PIN3 PIN2  PIN4 PIN2  PIN5 PIN2  PIN6 PIN2
 *      3            PIN4 PIN3  PIN5 PIN3  PIN6 PIN3  PIN7 PIN3
 *      4            PIN5 PIN4  PIN6 PIN4  PIN7 PIN4   PIN5 PIN9
 *      5            PIN6 PIN5  PIN7 PIN5   PIN6 PIN0  PIN6 PIN1
 *      6            PIN7 PIN6   PIN7 PIN0  PIN7 PIN1  PIN7 PIN2
 * - The systematic is as following:
 *      - every EVEN column (for   UPward pointing LED) is a progression of
rowNr + colNr + 1 until it                reaches PIN_COUNT-1.
 *        After this, the numbers reflect the rowNr+1
 *      - every  ODD column (for DOWNward pointing LED) is a progression of
rowNr              until rowNr + colNr + 1 reaches PIN_COUNT-1.
 *        After this, it is counting up again from 0 (= actAnode -
```

```
PIN_COUNT).
 *
 */

#include <avr/io.h>               // lib for I/O config
#include <stdbool.h>          // lib for Bit variables
#include <avr/interrupt.h>        // lib for standard interrupts
#include "charlieP.h"         // lib for charlie plexing

uint8_t outputPort[PIN_COUNT]={  DDR_C,  DDR_C,  DDR_D,  DDR_D,  DDR_D,
DDR_D,  DDR_D,  DDR_D,  DDR_D,  DDR_D,  DDR_B,  DDR_B};    // This array
shows the PORTs used for driving the charlieplexed LEDs
uint8_t outputPin [PIN_COUNT]={    PC2,    PC3,    PD0,    PD1,    PD2,
PD3,    PD4,    PD5,    PD6,    PD7,    PB0,    PB1};    // This array shows
the  PINs used for driving the charlieplexed LEDs

uint8_t outputMatrixXY[PIN_COUNT][PIN_COUNT-1]={};
// this array reflects the LEDs which have to be activated
// The dimensions are the same as in the geometry: [column][row]
uint8_t portMatrix [PORT_COUNT*2][PIN_COUNT]={
    //Definition of Port Nr        0       1       2       3       4       5
6       7       8       9       10      11    //    the portMatrix is used
to set the wanted ports to output HIGH or low
    //
//    When, no output is wanted only the anodes are set to output HIGH
    //
//    and the cathodes are set to high impedance (bit in DDRx = 0)
    /*PORT_B*/                {      0,      0,      0,      0,      0,
0,      0,      0,      0,      0, 1<<PB0, 1<<PB1},        //PORT_B
    /*PORT_C*/                { 1<<PC2, 1<<PC3,      0,      0,      0,
0,      0,      0,      0,      0,      0,      0},        //PORT_C
    /*PORT_D*/                {      0,      0, 1<<PD0, 1<<PD1, 1<<PD2,
1<<PD3, 1<<PD4, 1<<PD5, 1<<PD6, 1<<PD7,      0,      0},        //PORT_D
    /* DDR_B*/                {      0,      0,      0,      0,      0,
0,      0,      0,      0,      0, 1<<PB0, 1<<PB1},        // DDR_B
    /* DDR_C*/                { 1<<PC2, 1<<PC3,      0,      0,      0,
0,      0,      0,      0,      0,      0,      0},        // DDR_C
    /* DDR_D*/                {      0,      0, 1<<PD0, 1<<PD1, 1<<PD2,
1<<PD3, 1<<PD4, 1<<PD5, 1<<PD6, 1<<PD7,      0,      0}        // DDR_D
};

/*****************************************************************
 *
 * CharlieP_setPortsForLightingLeds
 *
 * - IMPORTANT: this function has to be in a cycle of about 10ms in order to
have a tradeoff between dynamic changes of the output but long persistence
for visibility
 * - Sets the ports for one anode (= one pin) active
 * - multiple independent LEDs can be activated
 * - function outputs DONE, when all anodes were set to active. otherwise it
```

```
outputs 0.
 *
 ***********************************************************************/
uint8_t CharlieP_setPortsForLightingLeds(void)
{
    static uint8_t actOutputAnode = 0;                                  // set
variable as local and persistent

    PORTB = portMatrix[PORT_B][actOutputAnode];                          //
set PORTB output values (HIGH or LOW)
    PORTC = portMatrix[PORT_C][actOutputAnode];                          //
set PORTC output values (HIGH or LOW)
    PORTD = portMatrix[PORT_D][actOutputAnode];                          //
set PORTD output values (HIGH or LOW)
    DDRB  = portMatrix[DDR_B ][actOutputAnode];                          //
set  DDRB output values (high-omhic or PORTB value)
    DDRC  = portMatrix[DDR_C ][actOutputAnode];                          //
set  DDRC output values (high-omhic or PORTD value)
    DDRD  =    portMatrix[DDR_D ][actOutputAnode];                          //
set  DDRD output values (high-omhic or PORTC value)
    actOutputAnode++;                                                   // next
anode
    if (actOutputAnode < PIN_COUNT) return 0;                           // when
not the last anode -> exit with returning 0

    actOutputAnode = 0;                                                  //
when the last anode -> set back to first
    return DONE;                                                        // when
the last anode -> exit with returning "DONE"
}

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////
// Mapping des SignalOutput Arrays
=========================================================
//mapped die Matrix auf den SignalOutout, um so die LEDs in Gruppen
einzuteilen

/***********************************************************************
 *
 * CharlieP_setPortMatrix
 *
 * - maps the outputMatrixXY (xy-matrix of activatable LEDs) to the
necessary port and pins
 *
 ***********************************************************************/
void CharlieP_setPortMatrix()
{
    uint8_t actCathode = 0, actPortCathode = 0, actPinCathode = 0,
bitInOutputMatrixForLedUpwards    = 0;
    uint8_t actAnode   = 0, actPortAnode   = 0, actPinAnode   = 0,
```

```
bitInOutputMatrixForLedDownwards = 0;
    for        (uint8_t rowNr = 0; rowNr <PIN_COUNT-1    ; rowNr++)
// loop over all rows
        for (uint8_t colNr = 0; colNr <PIN_COUNT/2     ; colNr++)
// loop over half the columns, since each 2nd column has antiparallel LEDs
(cathode and anode switched)
        {
// 'rowNr' and 'colNr' define two antiparallel LEDs
            bitInOutputMatrixForLedUpwards        = outputMatrixXY
[colNr*2]  [rowNr];        // read, whether   UPward pointing LED (LED in
even column) has to be activated
            bitInOutputMatrixForLedDownwards    = outputMatrixXY
[colNr*2+1][rowNr];        // read, whether DOWNward pointing LED (LED in
odd column) has to be activated

            actAnode            = colNr + rowNr + 1;
// sets the pin of the actual   anode. For details see comment in the
beginning of CharlieP.c
            actCathode            = rowNr;
// sets the pin of the actual cathode. For details see comment in the
beginning of CharlieP.c
            if (actAnode>= PIN_COUNT)
// once the change of the charlie plexing type in the row is reached
            {
//
                actCathode        = actAnode - PIN_COUNT;
// recalculate the pin of the actual cathode. For details see comment in the
beginning of CharlieP.c
                actAnode        = rowNr + 1;
// recalculate the pin of the actual   anode. For details see comment in the
beginning of CharlieP.c
            };
            actPortCathode            = outputPort[actCathode];
// get cathode PIN / PORTs for  upward pointing LED, which has to be
(de)activated
            actPinCathode            = outputPin [actCathode];

            actPortAnode            = outputPort[actAnode  ];
// get   anode PIN / PORTs for  upward pointing LED, which has to be
(de)activated
            actPinAnode                = outputPin [actAnode  ];
            CHG_BIT(portMatrix[actPortCathode][actAnode  ], actPinCathode,
bitInOutputMatrixForLedUpwards  );    // change the DDR for the pin and port
for the   UPward pointing LED
            CHG_BIT(portMatrix[actPortAnode  ][actCathode],   actPinAnode,
bitInOutputMatrixForLedDownwards);    // change the DDR for the pin and port
for the DOWNward pointing LED
        };
}
```