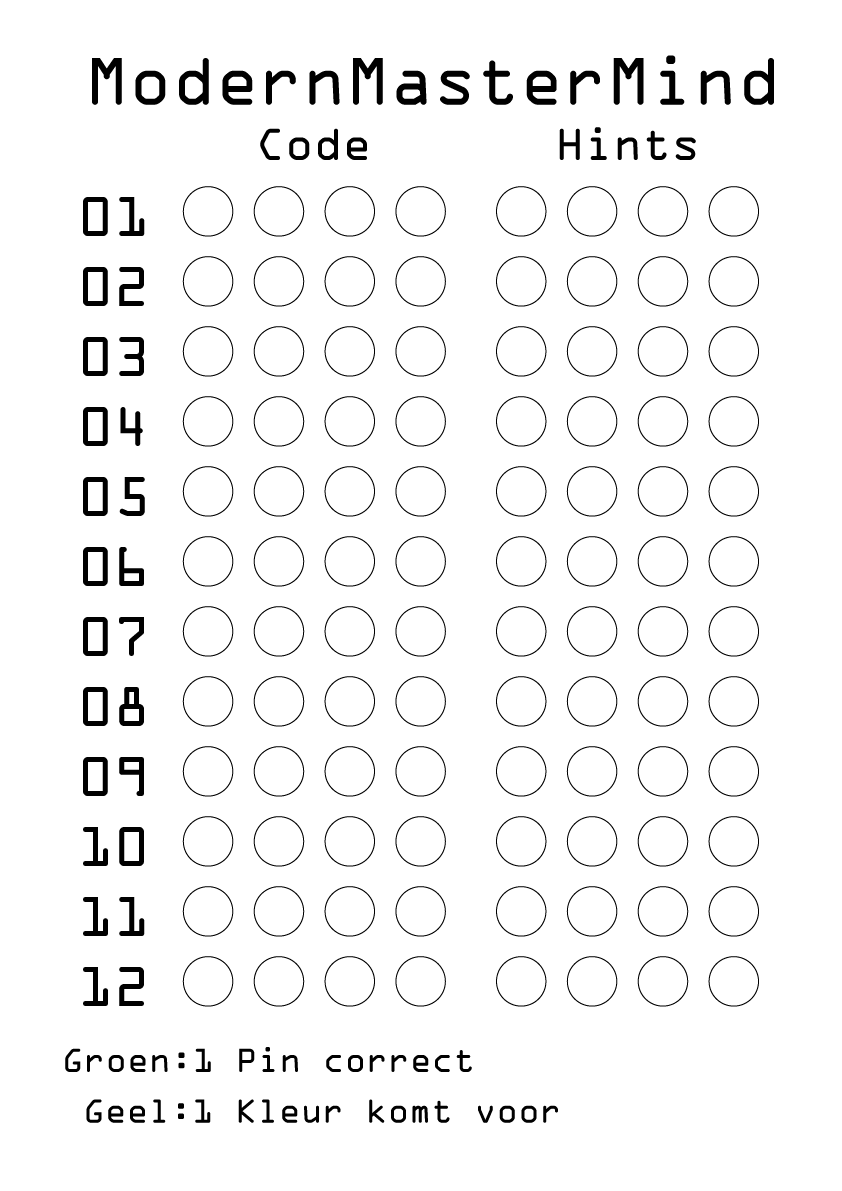
Dries Kennes | Project I | 2014-2015

ModernMasterMind

Project I – ICT-Elektronica



# Voorwoord

Ik ben Dries Kennes, student ICT-Elektronica Fase 1 aan Thomas More Mechelen op Campus De Nayer. Mijn voornaamste interesses zijn elektronica en alles wat te enigszins programmeren valt. Online ga ik door het leven als Dries007, een enthousiaste Java programmeur die zich in zijn vrije tijd vooral bezighoud met het maken van build-to-order Minecraft mods.

Tijdens de kerstvakantie ben ik op zoek gegaan naar projectonderwerpen. Vorig jaar (6e jaar middelbaar) wilden wij tijdens de lessen elektronica met een Arduino en een LED matrix spelletjes maken. Helaas was het juiste materiaal niet tijdig beschikbaar. Daarom heb ik er dit jaar voor gekozen om één van die spellen uit te werken. Het werd Mastermind omdat dit een uitdaging leek, onder andere dankzij de vereiste om kleuren te kunnen weergeven.

Mijn contacteren kan via email, [dries.kennes@student.thomasmore.be](mailto:dries.kennes@student.thomasmore.be), of via mijn website, [dries007.net](http://dries007.net).

Ten slotte wil ik Marc Roggemans bedanken voor het uitlenen van materiaal en Jurre De Weerdt voor het helpen met solderen van enkele SMD onderdelen.

# Inhoud

[1 Voorwoord 1](#_Toc419634062)

[2 Inhoud 2](#_Toc419634063)

[3 Hardware 3](#_Toc419634064)

[3.1 RIOT 3](#_Toc419634065)

[3.2 Het basisprincipe 3](#_Toc419634066)

[3.3 De PCBs 4](#_Toc419634067)

[4 Software 5](#_Toc419634068)

[4.1 De memory map 5](#_Toc419634069)

[4.2 Het WS2812 protocol 6](#_Toc419634070)

[5 Besluit 8](#_Toc419634071)

[6 Bijlagen 9](#_Toc419634072)

[6.1 PCB Schema’s & Layout 9](#_Toc419634073)

[6.2 Broncode 12](#_Toc419634074)

[6.2.1 mastermind.h 12](#_Toc419634075)

[6.2.2 mastermind.c 15](#_Toc419634076)

[6.2.3 HTML template 31](#_Toc419634077)

[6.2.4 AVR.h 33](#_Toc419634078)

[6.2.5 AVR.c 37](#_Toc419634079)

# Hardware

## RIOT

RIOT staat voor Remote Internet Operated Terminal, en is een ontwerp van M. Roggemans en D. Pauwels. RIOT werd ontworpen als educatief platform in 2002, maar is eigenlijk nooit gebruikt. Meneer Roggemans heeft tijdens de lessen interfacetechnieken dit platform wel aangehaald, en zo kreeg ik interesse om dit te gebruiken. Ik kreeg toegang tot de bordjes na de examens na de kerstvakantie, zodat ik hiermee kon experimenteren, en zien of ze bruikbaar zouden zijn als basis voor mijn project. Het gebruiken van een 10+ jaar oud platform zorgt voor een aantal hindernissen, meer hierover in het hoofdstuk *Software*.

Meer informatie over RIOT is beschikbaar in op Telescript[[1]](#footnote-1).

## Het basisprincipe

Het basisprincipe van RIOT (en dus ook mijn project) is dat 2 controllers met elkaar communiceren via een gedeeld geheugen (een Dual Port RAM, voortaan DP-RAM). Dit geheugen heeft een aantal ‘gewone’ adressen die vrij kunnen worden gebruikt, en 2 interrupt adressen. De interrupt adressen worden gebruikt om, zoals de naam al doet raden, van de ene controller naar de andere een interrupt te genereren. De controllers zijn een Beck SC12 en een AVR ATMega128A.

De reden voor het gebruiken van een dergelijk systeem is in dit geval de hoeveelheid IO verhogen. De SC12 heeft namelijk een zeer beperkte hoeveelheid IO, die vooral op pinnen zit die al door andere functies in gebruikt zijn (bijvoorbeeld UART of I²C). De SC12 heeft echter wel een gemultiplexte 8 bit data/adres bus, waarmee een extern geheugen van 256 bytes kan worden aangesproken. Door 2 IO pinnen te gebruiken als adres lijnen 8 en 9 kan het aanspreekbare geheugen worden uitgebreid tot 1kb (4 pages van 256 bytes).

Aangezien een Mastermind spelbord 12 rijen heeft van 8 gekleurde (RGB) pionnen, dit zijn in totaal dus 12 x 8 x 3 = 288 bytes is dit trucje geen overbodige luxe. Als men dan nog op een eenvoudige manier een LCD wil aansturen (20 karakters x 4 lijnen) heeft men nog eens 80 bytes nodig. Uiteindelijk zou 512 bytes genoeg zijn geweest, maar een DP-RAM chip met 1kb is commercieel beschikbaar, 512 bytes niet.

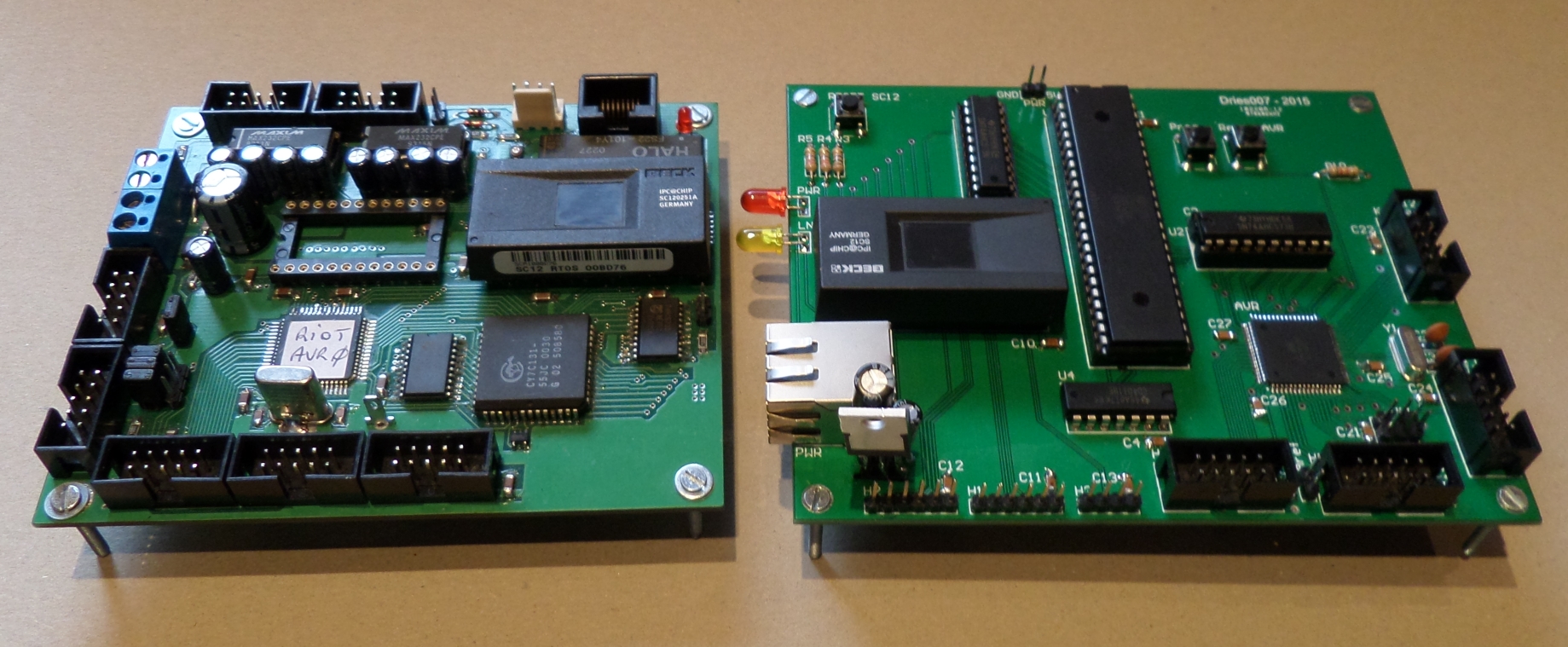
Aangezien de adres- en databus van beide controllers gemultiplext zijn, is het nodig om een adres latch te gebruiken. Ik gebruik 74AHC573 chips aangezien die aan de timingsspecificaties van de ATMega128A voldoen.

Voor de LEDs gebruik ik WS2812 LEDs. Deze LEDs zijn ideaal voor dit project omdat ze, ongeacht de hoeveelheid LEDs, maar 1 pin op een (relatief snelle) controller nodig hebben. Ze worden namelijk allemaal in serie geschakeld. De werking van het protocol word uitgelegd in het hoofdstuk ‘

*Het WS2812* protocol*’* .

## De PCBs

Op mijn versie van het bord zijn onnodige onderdelen weggelaten. Onder andere de UART naar RS232, de Real Time Clock en het voedingscircuit moesten er aan geloven.

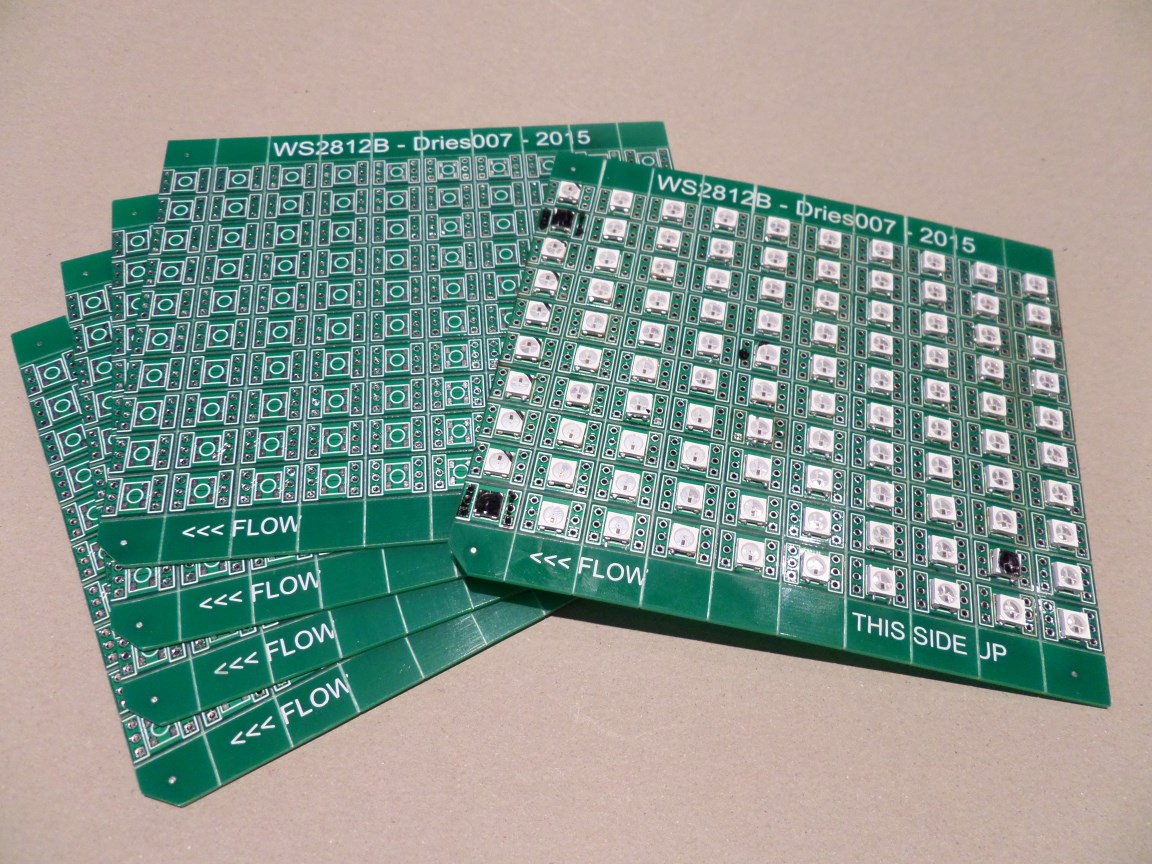


Figuur 3.1 RIOT links, Eigen PCB rechts

De PCBs zijn getekend met Altium. Het main bord en de LCD breakout zijn besteld bij Seedstudio, het LED panel bij multi-cb. De bestukking was manueel. De volgende fouten heb ik achteraf (tijdens het testen of solderen) ontdekt:

* De databus van het DP-RAM naar de AVR is in omgekeerde volgorde (de bits zitten omgekeerd).
* De WS2812 LEDs passen maar net op de getekende footprint, wat wil zeggen dat ze bijna onmogelijk met een gewone soldeerbout soldeerbaar zijn. Gelukkig heb ik op school hulp gekregen, en mocht ik een warme lucht soldeerstation gebruiken.

Het volledige schema en de layout van de PCBs is beschikbaar in de bijlage *PCB Schema’s & Layout*.



Figuur 3.2 Het LED panel

# Software

## De memory map

Dit is de verdeling van het DP-RAM. De 2 interrupt adressen liggen vast, namelijk 0x3FE en 0x3FF. Alle andere locaties zijn vrij gekozen.

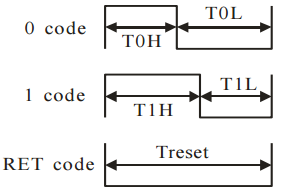
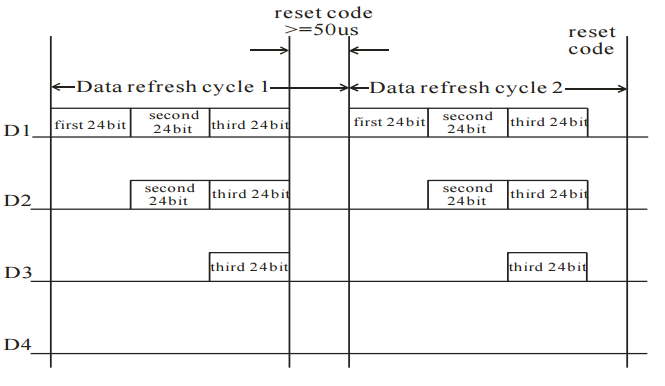
Er is extra plaats voorzien bij de LED data en bij de LCD buffer om te vermijden dat die verplaatst zouden moeten worden indien er zich kleine veranderingen voordoen in de specificaties.

|  |  |
| --- | --- |
| Adres | Functie |
| 0x000 -> 0x1FE | RGB data voor maximaal 170 LEDs |
| 0x1FF | Aantal LEDs |
| 0x200 -> 0x250 | LCD buffer (80 karakters) |
| 0x251 | LCD commando |
| 0x252 | LEDs dim |
| 0x253 | Laatst ingedrukte toets op keypad† |
| 0x254 -> 0x299 | *Beschikbaar* |
| 0x300 & 0x301 | Firmware id (42 en 0x42 respectievelijk) |
| 0x302 -> 0x3FD | *Beschikbaar* |
| 0x3FE | Interrupt SC12 -> AVR |
| 0x3FF | Interrupt AVR -> SC12† |

Tabel 4.1 De geheugenallocatie van het DP-RAM  
†: enkel gebruikt voor debug

## Het WS2812 protocol

De WS2812 LEDs werken als een lange serie schakeling, waarbij elke led de eerste 24 bits gebruikt om zijn kleur in te stellen. De andere bits worden doorgegeven, zie *Figuur 4.1*. De volgorde van de bits is niet RGB, maar GRB, met hoogste bit eerst.



Figuur 4.1 LED cascading & timing diagram

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Code | Betekenis | Tijd | Min | Typ | Max |
| T0H | 0 code - high voltage time | 0,35 µs ±150 ns | 200 ns | 350 ns | 500 ns |
| T1H | 1 code - high voltage time | 0,70 µs ±150 ns | 550 ns | 700 ns | 850 ns |
| T0L | 0 code - low voltage time | 0,80 µs ±150 ns | 650 ns | 800 ns | 950 ns |
| T1L | 1 code - low voltage time | 0,60 µs ±150 ns | 450 ns | 600 ns | 750 ns |
| Reset | low voltage time | > 50µs | 50 µs |  |  |
| TH+TL | Cyclus time | 1,25µs ±600ns | 660 ns | 1250 ns | 1850 ns |

Tabel 4.2 De timing tabel uit de datasheet, met genormaliseerde waarden

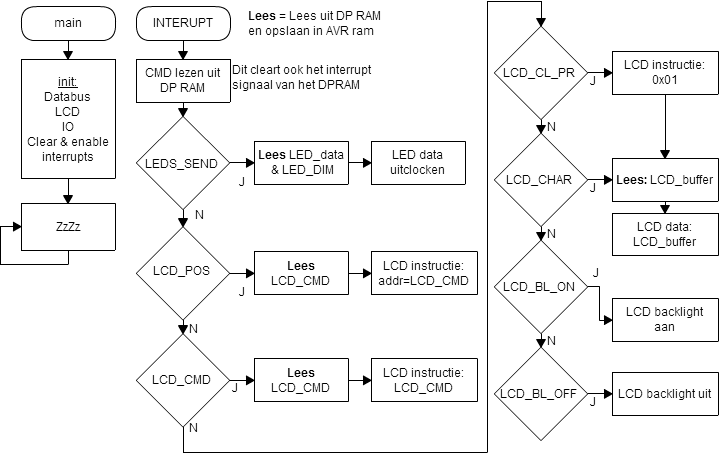
De software driver die vaak voor de ze LEDs word gebruikt (Adafruit NeoPixel Library[[2]](#footnote-2)) is ingewikkeld, geschreven in C++ en geschikt voor zowel 8, 12 als 16 MHz Arduino’s. Na verder zoeken vond ik een aantal interessante artikels[[3]](#footnote-3) die de timing van de LEDs analyseren.

Uit hun analyse blijkt dat de timing eigenlijk niet zo restrictief is als lijkt uit de datasheet. De enige kritieke tijd blijkt T0H te zijn, en die is net haalbaar op 4MHz (500ns \* 4MHz = 2 klokcycli). Alle andere timing restricties zijn langer, en kunnen dus met nops worden ingevuld.

Uiteindelijk heb ik ervoor gekozen een aangepaste versie van de *light\_ws2812 library[[4]](#footnote-4)* te gebruiken. Dit is de code die word beschreven in een van de artikels. Ze is simpel en eenvoudig te begrijpen.

## De AVR software

De software in de AVR is geschreven in C, met uitzondering van de WS2812 driver die in inline assembly is geschreven. Voor de gebruiksvriendelijkheid zijn alle configuratie opties zijn bovenaan in de header file als macro’s gedefinieerd. De code is voorzien van commentaar en sectie aanduidingen om groepen van bij elkaar hordende functies te groeperen. De volledige header- en codebestanden zijn is terug te vinden in de bijlage als respectievelijk *AVR.h* en *AVR.c*.



Figuur 4.2 De AVR flowchart

### Het hoofdprogramma (main)

Het hoofdprogramma in de AVR doet niet veel meer dan een aantal initialisaties en wacht dan om een interrupt van het DP-RAM.

De volgende initialisaties worden gedaan (met regelnummers uit *AVR.c*):

* De externe memory interface (adres- en databus) word ingeschakeld. *R259*
* De Firmware ID adressen in het DP-RAM worden op 42 en 0x42 gezet. *R262-263*
* De LCD poort word ingesteld als uitgang. *R266*
* De LED ping word ingesteld als uitgang. *R272-273*
* Externe interrupt 4 word geconfigureerd. *R276-277*
* De LCD word gedigitaliseerd in 4 bit mode. *R279-294*
* Lees DP-RMA 0x3FE om mogelijke interrupt te resetten. *R297*
* Schakel interrupts in.

Dan gaat het programma in een lus, wachtend op een interrupt.

### De interrupt routine

Deze interrupt routine voert uit wanneer de DP-RAM een schrijf instructie heeft verwerkt naar het adres 0x3FE vanuit de SC12. Hoe de AVR juist reageert is afhankelijk van wat er in naar dit dress geschreven word. Aangezien het hoofdprogramma niets doet in dit project, is het geen enkel probleem om in de interrupt routine software delays te gebruiken.

|  |  |
| --- | --- |
| Waarde | Functie |
| 0x00 | Geen functie |
| 0x01 | Update LEDs |
| 0x02 | Print LCD karakter buffer (tot max of 0x00) |
| 0x03 | Stuur LCD instructie |
| 0x04 | Stuur clear instructie en print LCD karakter buffer |
| 0x05 | Zet LCD cursor positie op LCD commando |
| 0x06 | LCD backlight aan |
| 0x07 | LCD backlight uit |

Tabel 4.3 De commando’s van de SC12 naar de AVR

#### 0x00 of een waarde die niet voorkomt in de tabel– Geen functie

De null byte (0x00) is expliciet gereserveerd om niets te doen (buiten het activeren van mogelijke debug code). Alle andere combinaties kunnen worden gebruikt voor extra functies.

#### 0x01 – Update LEDs

* Lees aantal LEDs. *R24*
* Lees LED dimmer waarde. (0xFF = volle sterkte) *R26*
* Check aantal LEDs met logisch maximum bepaald door geheugen allocatie. *R28*
* Loop over de volgende instructies voor elke LED: *R30-31*
  + Voor 3 x LED kleur (rood, groen en blauw). *R33 & 34 & 35*
    - Lees volgende byte uit LED data in DP-RAM.
    - Voor logische die byte AND uit met LED dimmer waarde.
* Geef deze data door aan de WS2812 driver. *R38*

#### 0x02 – Print LCD karakter buffer

* Loop over de volgende instructies tot het einde van de karakter buffer: *R63*
  + Lees volgende byte uit karakter buffer in DP-RAM. *R65*
  + Als karakter een 0x00 byte is, beëindig de lus onmiddellijk. *R66*
  + Stuur de byte naar de LCD driver. *R67*

#### 0x03 – Stuur LCD instructie

* Lees LCD commando byte uit DP-RAM en stuur die naar de LCD driver. *R50*

#### 0x04 – Stuur clear instructie en print LCD karakter buffer

* Stuur een clear & home instructie (0x01) naar LCD driver. *R56*
* Wacht 10ms. *R57*
* Voer de code uit die bij *0x02 – Print LCD karakter buffer* hoort. *R59*

#### 0x05 – Zet LCD cursor positie op LCD commando

Aangezien een positie instructie er zo uitziet: “1aaa aaaa” Waarbij de a’s staan voor een positie op de LCD, kan men een positie byte met een logische OR omzetten naar de juiste instructie:

* Lees LCD commando byte uit DP-RAM, OR met 0x80, stuur naar LCD driver. *R44*

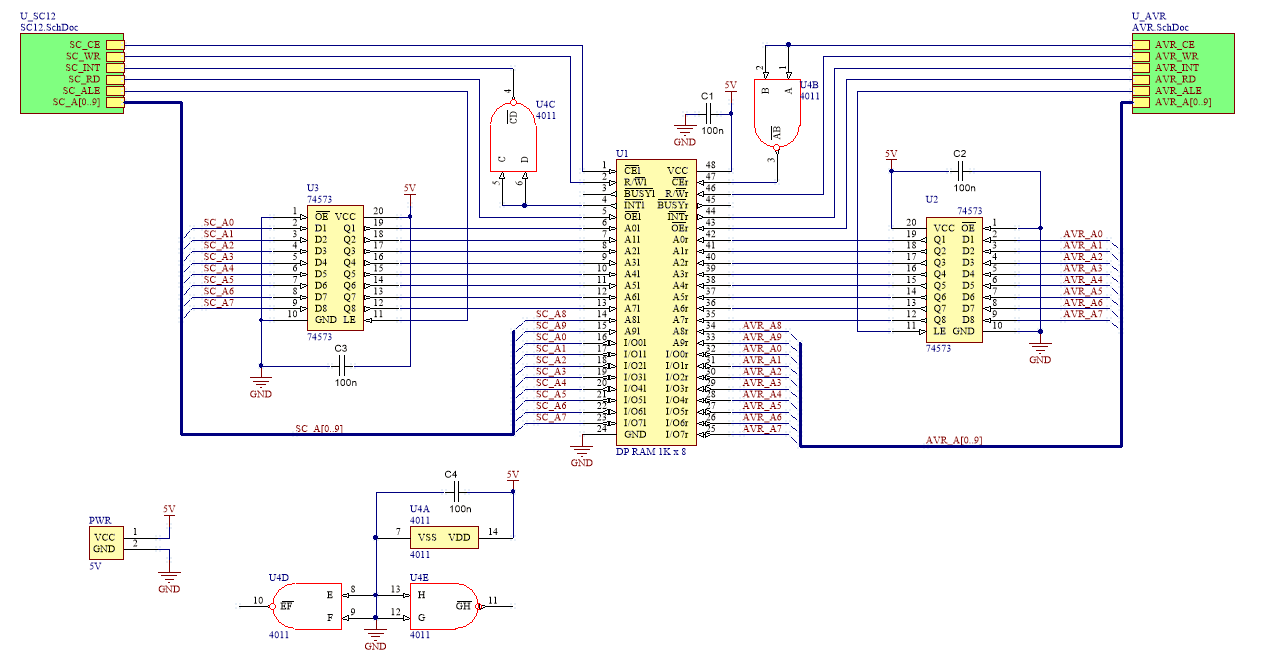
#### 0x06 of 0x07 –LCD backlight aan of uit

Dit commando set een interne byte op 1 of 0, voor aan en uit respectievelijk. De volgende keer dat er iets met de LCD driver word gedaan, heeft de verandering effect.

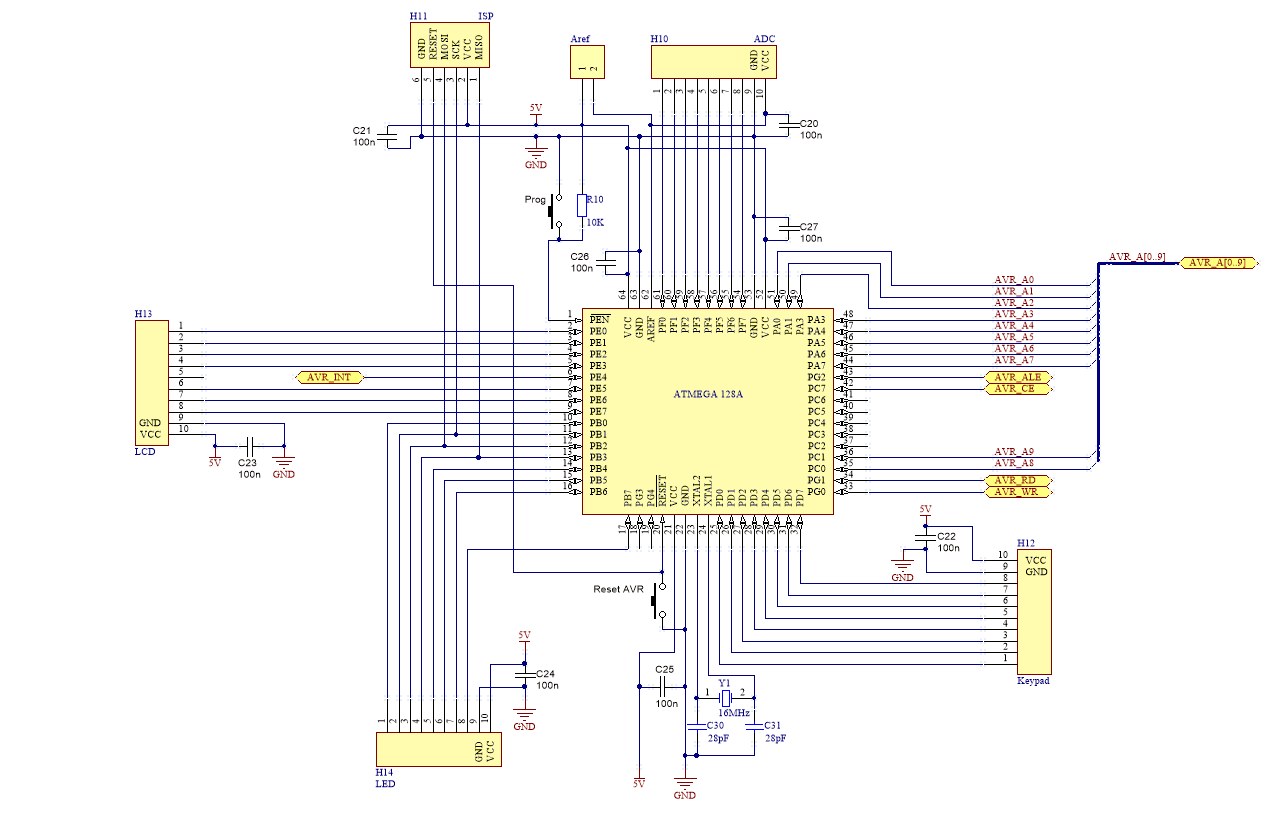
# Besluit

# Bijlagen

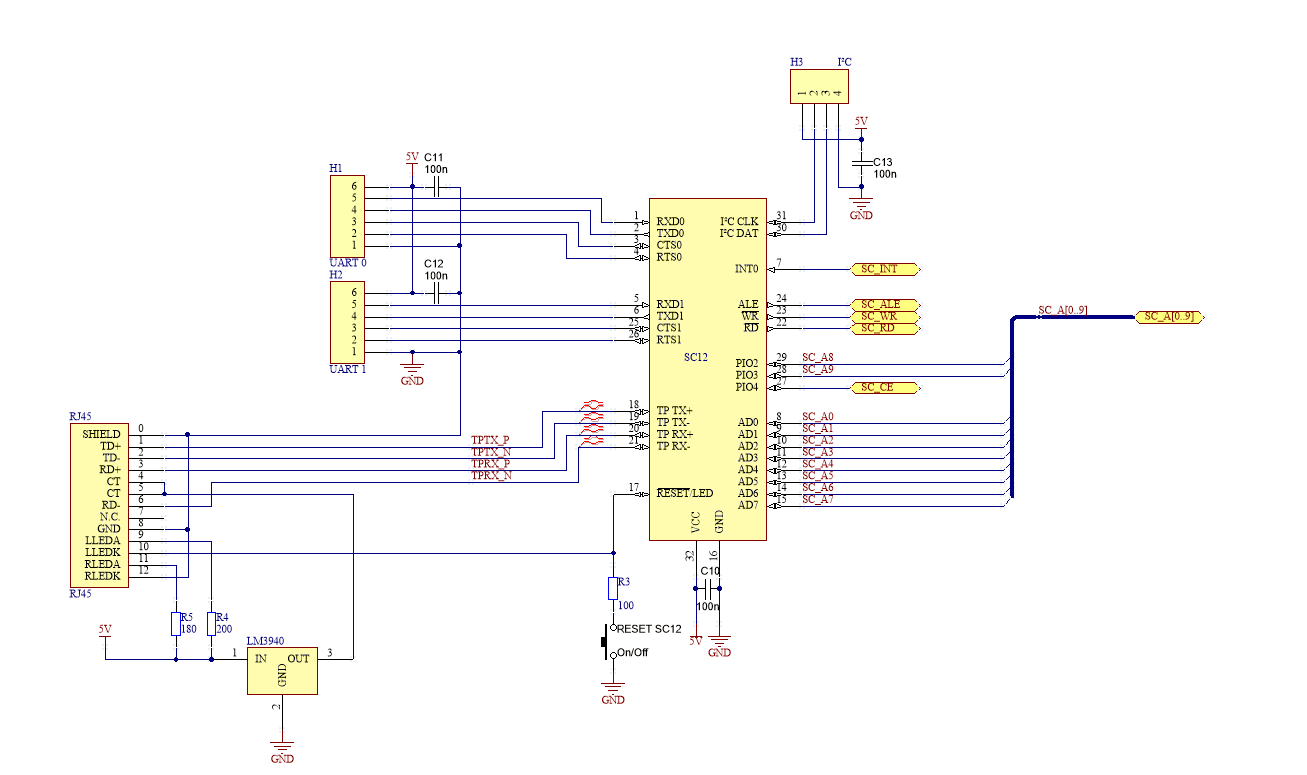
## PCB Schema’s & Layout



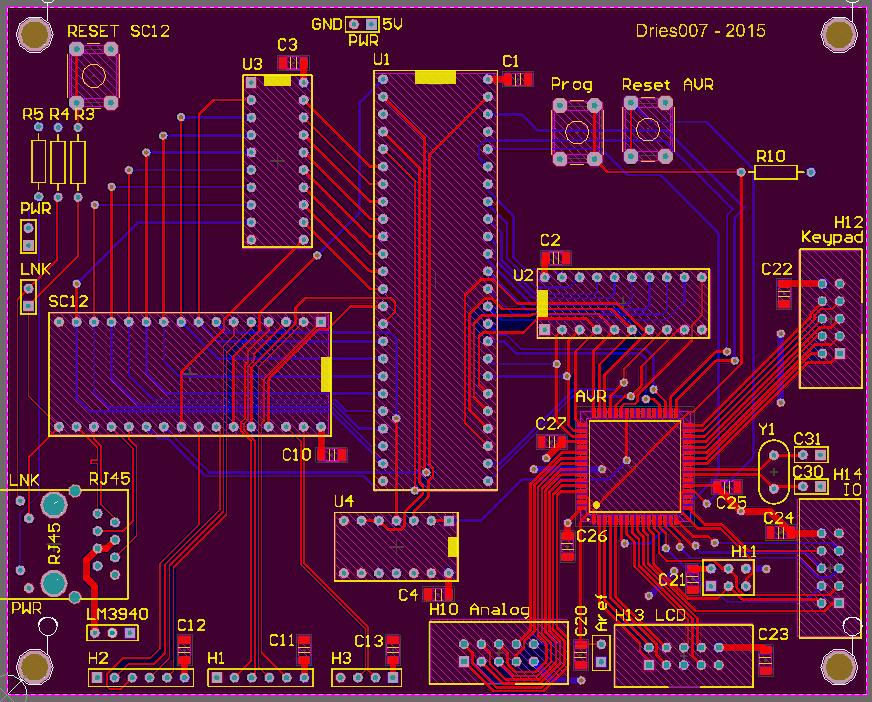
Figuur 6.1 Schema DP-RAM deel



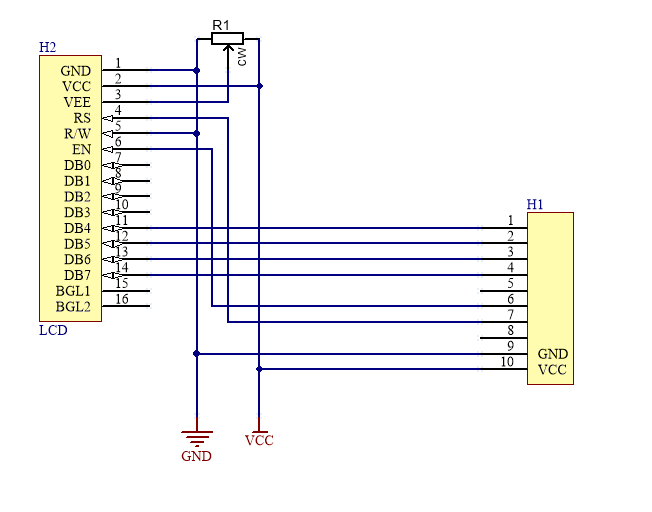
Figuur 6.2 Schema AVR deel



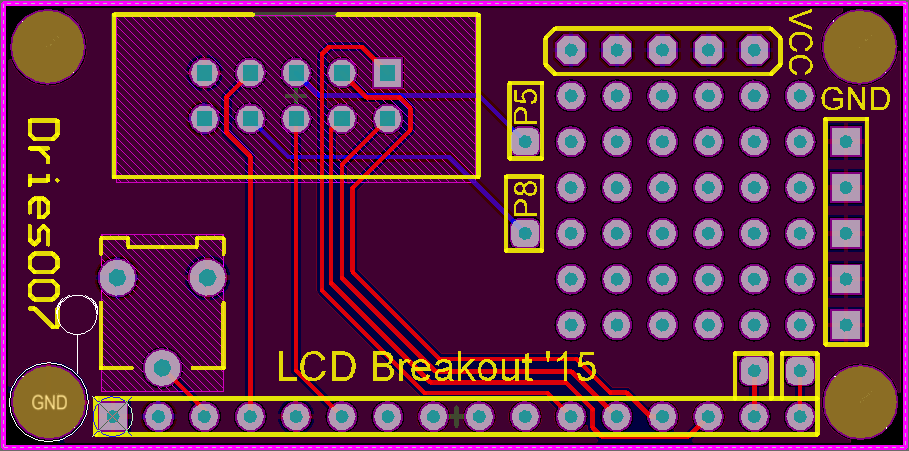
Figuur 6.3 Schema SC12 deel



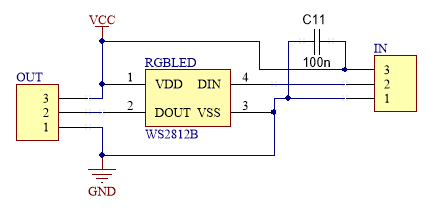
Figuur 6.4 Master PCB layout (125 x 100 mm)



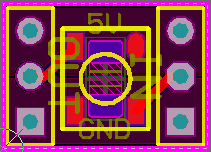
Figuur 6.5 Schema LCD-Breakout



Figuur 6.6 LCD-Breakout PCB layout (50 x 25 mm)



Figuur 6.7 WS2812B Schema



Figuur 6.8 WS2812B Layout (11 x 8 mm)

## Broncode

De volledige broncode is ook beschikbaar op [github.com/dries007/ModernMasterMind](https://github.com/dries007/ModernMasterMind).

Mastermind.c draait op de SC12, AVR.c op de ATMega128.

Het originele HTML template is ook toegevoegd, aangezien dit gecomprimeerd bijna onleesbaaar is.

De cLib library die bij mastermind.h code hoord is beschikbaar op [www.beck-ipc.com](http://www.beck-ipc.com).

### mastermind.h

#ifndef SRC\_MASTERMIND\_H\_  
#define SRC\_MASTERMIND\_H\_  
  
#pragma option -1 //create 80186 code  
  
#include <clib.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <limits.h>  
#include <dos.h>  
#include <stdarg.h>  
#include <ctype.h>  
#include <rtos.h>  
#include <i86.h>  
  
#include "ramdump.h"  
#include "httpcli.h"  
#include "base64.h"  
#include "dns.h"  
  
/\* ================= DEFINES ================= \*/  
  
#define DEBUG 0  
  
#define VERSION "0.2"  
#define CYEAR "15"  
  
#define TASK\_STACKSIZE 2048  
  
#define TCPIP\_int 0xAC  
  
#define LCD\_LINE\_SIZE 16  
#define LCD\_LINES 2  
  
#define MAX\_COLORS 8  
#define COLORS 4  
#define ROWS 12  
  
#define STATE\_NO\_GAME 0  
#define STATE\_GAME\_CONFIGURED 1  
#define STATE\_GAME\_STARTED 2  
#define STATE\_GAME\_OVER 3  
#define STATE\_GAME\_WON 4  
  
// amount of (positive) hours offset from GMT  
#define TIMEZONE\_OFFSET 2  
  
// 170 Leds max (x 3 bytes = 0x1FE)  
#define RAM\_LEDS\_START 0x000  
#define RAM\_LEDS\_END 0x1FE  
#define RAM\_LEDS\_AMOUNT 0x1FF  
// 80 bytes of char buffer for LCD  
#define RAM\_LCD\_START 0x200  
#define RAM\_LCD\_END 0x250  
#define RAM\_LCD\_CMD 0x251  
// Mask for global LED dimming  
#define RAM\_LEDS\_DIM 0x252  
#define RAM\_KP\_LASTKEY 0x253  
  
#define RAM\_VERSION\_1 0x300  
#define RAM\_VERSION\_2 0x301  
  
// Interrupt registers  
#define RAM\_INT\_SEND 0x3FF  
#define RAM\_INT\_GET 0x3FE  
  
#define MAX\_LEDS (RAM\_LEDS\_END - RAM\_LEDS\_START)  
#define MAX\_LCD\_CHARS (RAM\_LCD\_END - RAM\_LCD\_START)  
  
#define CMD\_LEDS\_SEND 0x01  
#define CMD\_LCD\_CHAR 0x02  
#define CMD\_LCD\_CMD 0x03  
#define CMD\_LCD\_CL\_PR 0x04  
#define CMD\_LCD\_POS 0x05  
#define CMD\_LCD\_BL\_ON 0x06  
#define CMD\_LCD\_BL\_OFF 0x07  
  
#define SATUS\_KP\_PRESS 0x01  
  
#define NORMALIZE\_ADDRESS(addr) ((addr & 0x0FF) | 0x100)  
#define BANK\_FROM\_ADDRESS(addr) (addr >> 8)  
  
/\* ================= STRUCTS ================= \*/  
  
typedef unsigned char byte;  
typedef unsigned short address;  
  
typedef struct  
{  
 unsigned long ip;  
 char name[21];  
} User;  
  
typedef struct  
{  
 byte r;  
 byte g;  
 byte b;  
} RGB;  
  
typedef struct  
{  
 byte state;  
 byte vsPlayer;  
 User \* host;  
 byte colors;  
 byte code[COLORS];  
 byte nrOfGuesses;  
 byte guesses[ROWS][COLORS + 2];  
} Game;  
  
struct userlist\_el  
{  
 User user;  
 struct userlist\_el \* next;  
};  
  
/\* ================= METHODS ================= \*/  
  
void endProgram();  
  
User \* getUserByIP(long far \* ip);  
User \* getUserByName(char \* name);  
void addUser(long ip, char name[21]);  
  
Game \* getGame();  
void resetGame();  
void setRndCode(byte colors);  
void guessRow(byte id);  
  
void enableDatabus(); // Enables databus  
byte readDatabus(address addr); // read byte from databus  
void writeDatabus(address addr, byte value); // write byte to databus  
//void initTime();  
  
void clearLCD();  
void setLCDLine(byte line, const char \*string);  
void setLCDLineFormat(byte line, const char \*format, ...);  
  
void installCGIMethods();  
void removeCGIMethods();  
  
void printAllUsers();  
  
/\* ================= GLOBALS ================= \*/  
  
const RGB BLACK = { 0, 0, 0 };  
const RGB WHITE = { 255, 255, 255 };  
  
const RGB RED = { 255, 0, 0 };  
const RGB GREEN = { 0, 255, 0 };  
const RGB BLUE = { 0, 0, 255 };  
  
const RGB PINK = { 255, 0, 255 };  
const RGB AQUA = { 0, 255, 255 };  
const RGB YELLOW = { 255, 255, 0 };  
  
const RGB PURPLE = { 130, 0, 255 };  
const RGB ORANGE = { 255, 130, 0 };  
  
const RGB ALL\_COLORS[10] = { { 255, 0, 0 }, { 0, 255, 0 }, { 0, 0, 255 }, {  
 255, 0, 255 }, { 0, 255, 200 }, { 255, 255, 0 }, { 255, 255, 255 }, {  
 255, 80, 0 }, { 0, 0, 0 } };  
const char \* ALL\_COLOR\_CLASSES[10] = { "red", "green", "blue", "pink",  
 "aqua", "yellow", "white", "orange", "black" };  
  
union REGS inregs;  
union REGS outregs;  
struct SREGS segregs;  
  
Game game;  
  
struct userlist\_el \* listHead = NULL;  
struct userlist\_el \* listTail = NULL;  
  
#endif /\* SRC\_MASTERMIND\_H\_ \*

### mastermind.c

#include "mastermind.h"  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* USER RELATED  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
User \* getUserByIP(long far \* ip)  
{  
 struct userlist\_el \* current = listHead;  
 while (current != NULL)  
 {  
 if (current->user.ip == \*((unsigned long \*) ip))  
 return &(current->user);  
 current = current->next;  
 }  
 return NULL;  
}  
  
User \* getUserByName(char \* name)  
{  
 for (int i = 0; name[i]; i++)  
 name[i] = tolower(name[i]);  
  
 struct userlist\_el \* current = listHead;  
 while (current != NULL)  
 {  
 if (strcmp(current->user.name, name) == 0) return &(current->user);  
 current = current->next;  
 }  
 return NULL;  
}  
  
void addUser(long ip, char name[21])  
{  
 for (int i = 0; name[i]; i++)  
 name[i] = tolower(name[i]);  
 struct userlist\_el \* newItem = (struct userlist\_el \*) malloc(  
 sizeof(struct userlist\_el));  
  
 newItem->user.ip = ip;  
 strcpy(newItem->user.name, name);  
  
 if (listHead == NULL)  
 {  
 listHead = listTail = newItem;  
 }  
 else  
 {  
 listTail->next = newItem;  
 }  
}  
  
void printAllUsers()  
{  
 if (listHead == NULL)  
 {  
 printf("No users in the user list.\n");  
 return;  
 }  
  
 printf("User list:\n");  
 struct userlist\_el \* current = listHead;  
 while (current != NULL)  
 {  
 printf("Username: %s Remote IP: %d.%d.%d.%d\n", current->user.name,  
 (int) ((current->user.ip & 0xFF000000l) >> 24),  
 (int) ((current->user.ip & 0x00FF0000l) >> 16),  
 (int) ((current->user.ip & 0x0000FF00l) >> 8),  
 (int) (current->user.ip & 0x000000FFl));  
 current = current->next;  
 }  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* LEDS RELATED  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
address sendProper(byte row, address addr)  
{  
 for (byte p = 0; p < COLORS; p++)  
 {  
 RGB rgb = ALL\_COLORS[game.guesses[row][p]];  
 writeDatabus(addr++, rgb.r);  
 writeDatabus(addr++, rgb.g);  
 writeDatabus(addr++, rgb.b);  
 }  
  
 byte g = game.guesses[row][COLORS];  
 byte r = game.guesses[row][COLORS + 1];  
  
 for (byte p = 0; p < g; p++)  
 {  
 writeDatabus(addr++, GREEN.r);  
 writeDatabus(addr++, GREEN.g);  
 writeDatabus(addr++, GREEN.b);  
 }  
  
 for (byte p = 0; p < r; p++)  
 {  
 writeDatabus(addr++, YELLOW.r);  
 writeDatabus(addr++, YELLOW.g);  
 writeDatabus(addr++, YELLOW.b);  
 }  
  
 for (byte p = g + r; p < COLORS; p++)  
 {  
 writeDatabus(addr++, BLACK.r);  
 writeDatabus(addr++, BLACK.g);  
 writeDatabus(addr++, BLACK.b);  
 }  
  
 return addr;  
}  
  
address sendReverse(byte row, address addr)  
{  
 byte g = game.guesses[row][COLORS];  
 byte r = game.guesses[row][COLORS + 1];  
  
 for (byte p = g + r; p < COLORS; p++)  
 {  
 writeDatabus(addr++, BLACK.r);  
 writeDatabus(addr++, BLACK.g);  
 writeDatabus(addr++, BLACK.b);  
 }  
  
 for (byte p = 0; p < r; p++)  
 {  
 writeDatabus(addr++, YELLOW.r);  
 writeDatabus(addr++, YELLOW.g);  
 writeDatabus(addr++, YELLOW.b);  
 }  
  
 for (byte p = 0; p < g; p++)  
 {  
 writeDatabus(addr++, GREEN.r);  
 writeDatabus(addr++, GREEN.g);  
 writeDatabus(addr++, GREEN.b);  
 }  
  
 for (byte p = 0; p < COLORS; p++)  
 {  
 RGB rgb = ALL\_COLORS[game.guesses[row][COLORS - 1 - p]];  
 writeDatabus(addr++, rgb.r);  
 writeDatabus(addr++, rgb.g);  
 writeDatabus(addr++, rgb.b);  
 }  
  
 return addr;  
}  
  
void sendLEDS()  
{  
 address addr = RAM\_LEDS\_START;  
 byte row = 0;  
 while (row < ROWS)  
 {  
 addr = sendProper(row++, addr);  
 addr = sendReverse(row++, addr);  
 }  
 writeDatabus(RAM\_INT\_SEND, CMD\_LEDS\_SEND);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* GAME RELATED  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
Game \* getGame()  
{  
 return &game;  
}  
  
void resetGame()  
{  
 game.state = STATE\_NO\_GAME;  
 game.nrOfGuesses = 0;  
 for (int i = 0; i < ROWS; i++)  
 {  
 for (int j = 0; j < COLORS; j++)  
 {  
 game.guesses[i][j] = 9; // 9 = black  
 }  
 game.guesses[i][COLORS] = 0;  
 game.guesses[i][COLORS + 1] = 0;  
 }  
 sendLEDS();  
}  
  
void setRndCode(byte colors)  
{  
 if (colors > MAX\_COLORS) colors = MAX\_COLORS;  
 for (byte i = 0; i < COLORS; i++)  
 {  
 game.code[i] = rand() % colors;  
 }  
}  
  
void guessRow(byte id)  
{  
 if (id >= ROWS)  
 {  
 game.state = STATE\_GAME\_OVER;  
 return;  
 }  
 byte usedUpPins[COLORS];  
 for (byte i = 0; i < COLORS; i++)  
 usedUpPins[i] = 0;  
  
 for (byte i = 0; i < COLORS; i++)  
 {  
 // Exact matches  
 if (game.guesses[id][i] == game.code[i])  
 {  
 if (usedUpPins[i] == 1)  
 {  
 game.guesses[id][COLORS + 1]--;  
 }  
  
 usedUpPins[i] = 1;  
 game.guesses[id][COLORS]++;  
 if (game.guesses[id][COLORS] == COLORS)  
 {  
 game.state = STATE\_GAME\_WON;  
 }  
 }  
 else  
 {  
 for (byte j = 0; j < COLORS; j++)  
 {  
 if (game.guesses[id][i] == game.code[j] && usedUpPins[j] == 0)  
 {  
 usedUpPins[j] = 1;  
 game.guesses[id][COLORS + 1]++;  
 break;  
 }  
 }  
 }  
 }  
 sendLEDS();  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* DATA BUS RELATED  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void enableDatabus()  
{  
 pfe\_enable\_bus(0xFF, 1); // all 8 bits enabled with ALE  
 pfe\_enable\_pcs(1); // Chip select 1 (PIO4)  
 pfe\_enable\_pio(2, 5); // PIO2 = output, low  
 pfe\_enable\_pio(3, 5); // PIO3 = output, low  
}  
  
byte readDatabus(address addr)  
{  
 byte bank = addr >> 8;  
 pfe\_enable\_pio(2, bank & 0x01 ? 4 : 5); // if bank is 1 or 3, set PIO2 high, otherwise set PIO2 low  
 pfe\_enable\_pio(3, bank & 0x02 ? 4 : 5); // if bank is 3 or 4, set PIO3 high, otherwise set PIO3 low  
 return hal\_read\_bus((addr & 0x0FF) | 0x100, 0xFFFF, 0x0000); // Read data bus on corrected address (always in 0x100..0x1FF range)  
}  
  
void writeDatabus(address addr, byte val)  
{  
 byte bank = addr >> 8;  
 // if bank is 1 or 3, set PIO2 high, otherwise set PIO2 low  
 pfe\_enable\_pio(2, bank & 0x01 ? 4 : 5);  
 // if bank is 3 or 4, set PIO3 high, otherwise set PIO3 low  
 pfe\_enable\_pio(3, bank & 0x02 ? 4 : 5);  
 // Write to data bus on corrected address (always in 0x100..0x1FF range)  
 hal\_write\_bus((addr & 0x0FF) | 0x100, val, 0xFFFF, 0x0000);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* LCD RELATED  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void clearLCD()  
{  
 for (address addr = RAM\_LCD\_START; addr <= RAM\_LCD\_END; addr++)  
 {  
 writeDatabus(addr, ' ');  
 }  
 writeDatabus(RAM\_LCD\_CMD, 0x01);  
 writeDatabus(RAM\_INT\_SEND, CMD\_LCD\_CMD);  
}  
  
void setLCDLine(byte line, const char \* text)  
{  
 address offset = RAM\_LCD\_START;  
  
#if LCD\_LINES == 2  
 if (line == 1 || line == 3) offset += 40;  
#elif LCD\_LINES == 4  
 if (line % 2 != 0) offset += 40;  
 if (line > 1) offset += 0x14;  
#endif  
  
 address i = 0;  
 for (; i < strlen(text); i++) // chars  
 {  
 writeDatabus(i + offset, text[i]);  
 }  
 for (; i < LCD\_LINE\_SIZE; i++) // spaces  
 {  
 writeDatabus(i + offset, ' ');  
 }  
 writeDatabus(RAM\_INT\_SEND, CMD\_LCD\_CL\_PR);  
 delay(100);  
}  
  
void setLCDLineFormat(byte line, const char \* format, ...)  
{  
 byte buffer[LCD\_LINE\_SIZE + 1];  
 buffer[LCD\_LINE\_SIZE] = 0x00;  
  
 /\* Start magic \*/  
 va\_list aptr;  
 va\_start(aptr, format);  
 vsprintf(buffer, format, aptr);  
 va\_end(aptr);  
 /\* End magic \*/  
  
 setLCDLine(line, buffer);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* TASKS  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
byte LCDupdateRunning;  
  
void LCDupdate()  
{  
 LCDupdateRunning = 1;  
 byte ip[16], oldIp[16];  
  
 while (LCDupdateRunning)  
 {  
 Get\_IPConfig(ip, NULL, NULL);  
  
 if (strcmp(ip, oldIp) != 0)  
 {  
 strcpy(oldIp, ip);  
 setLCDLine(1, ip);  
 }  
 RTX\_Sleep\_Time(2500);  
 }  
}  
  
unsigned int LCDUpdate\_stack[TASK\_STACKSIZE / sizeof(unsigned int)];  
int LCDupdateID;  
  
TaskDefBlock LCDupdateTaskDefBlock = { LCDupdate, { 'L', 'C', 'D', ' ' },  
 &LCDUpdate\_stack[TASK\_STACKSIZE / sizeof(unsigned int)], // top of stack  
 TASK\_STACKSIZE, // size of stack  
 0, // attributes, not supported  
 100, // lower priority than any system tasks  
 0, // time slice (if any), not supported  
 0, 0, 0, 0 // mailboxes  
 };  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* MAIN  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/\*  
 \* Ends all tasks, returns focus and exits  
 \*/  
void endProgram()  
{  
 setLCDLine(0, "Shutdown issued");  
  
 removeCGIMethods();  
  
 printf("Gracefully ending all tasks...\n");  
  
 LCDupdateRunning = 0; // tell tasks to stop  
  
 RTX\_Sleep\_Time(3000); // give tasks time to end  
  
 printf("Killing the non obedient tasks...\n");  
 RTX\_Delete\_Task(LCDupdateID);  
  
 printf("\nEND OF PROGRAM\n");  
 // Release input/output  
 BIOS\_Set\_Focus(FOCUS\_BOTH);  
 exit(0);  
}  
  
void test()  
{  
 writeDatabus(RAM\_LEDS\_DIM, 0xFF);  
 writeDatabus(RAM\_LEDS\_AMOUNT, 100);  
  
 address addr = RAM\_LEDS\_START;  
 for (byte r = 0; r < ROWS; r++)  
 {  
 for (byte p = 0; p < COLORS \* 2; p++)  
 {  
 RGB rgb = ALL\_COLORS[rand() % MAX\_COLORS];  
 writeDatabus(addr++, rgb.r);  
 writeDatabus(addr++, rgb.g);  
 writeDatabus(addr++, rgb.b);  
 }  
 }  
 writeDatabus(RAM\_INT\_SEND, CMD\_LEDS\_SEND);  
 delay(1000);  
}  
  
void debugGameState()  
{  
 printf(  
 "\nGame state: %d\nGame VS player: %d\nGame host: %s\n# colors: %d\nCode: ",  
 game.state, game.vsPlayer, game.host->name, game.colors);  
 for (byte i = 0; i < COLORS; i++)  
 printf("%s ", ALL\_COLOR\_CLASSES[game.code[i]]);  
 printf("\n# of guesses: %d\nGuesses Table:\n", game.nrOfGuesses);  
 for (byte r = 0; r < ROWS; r++)  
 {  
 printf("Guess #%d: ", r);  
 for (byte c = 0; c < COLORS; c++)  
 printf("%10s ", ALL\_COLOR\_CLASSES[game.guesses[r][c]]);  
 printf(" Exact: %d Color: %d\n", game.guesses[r][COLORS],  
 game.guesses[r][COLORS + 1]);  
 }  
 printf("Code: ");  
 for (byte c = 0; c < COLORS; c++)  
 printf("%10s ", ALL\_COLOR\_CLASSES[game.code[c]]);  
 printf("\n\n");  
}  
  
void main()  
{  
 // Get focus  
 BIOS\_Set\_Focus(FOCUS\_APPLICATION);  
  
 /\*\*  
 \* INIT All of the things!  
 \*/  
 enableDatabus();  
  
 writeDatabus(RAM\_INT\_SEND, CMD\_LCD\_BL\_ON);  
  
 writeDatabus(RAM\_LEDS\_DIM, 0xFF);  
 writeDatabus(RAM\_LEDS\_AMOUNT, ROWS \* 2 \* COLORS);  
  
 writeDatabus(RAM\_LCD\_CMD, 0x0C); // Blink off  
 writeDatabus(RAM\_INT\_SEND, CMD\_LCD\_CMD);  
  
 clearLCD();  
  
 printf("\n\nAVR firmware version id: %d 0x%02x\n\n\n",  
 readDatabus(RAM\_VERSION\_1), readDatabus(RAM\_VERSION\_2));  
 setLCDLine(0, "MMM by Dries007");  
 setLCDLine(1, "Booting...");  
  
 // Ethernet connection check  
 if (BIOS\_Ethernet\_State(NULL, NULL))  
 {  
 setLCDLine(0, "ERROR");  
 setLCDLine(1, "NO ETHERNET!");  
  
 while (1)  
 {  
 test();  
 }  
  
 return;  
 }  
  
 /\*\*  
 \* RUN ALL TASKS  
 \*/  
 int result = RTX\_Create\_Task(&LCDupdateID, &LCDupdateTaskDefBlock);  
  
 if (result != 0)  
 {  
 printf("Creating/restart LCDupdate failed %d, exit program\n", result);  
 //delete task1  
 RTX\_Delete\_Task(LCDupdateID);  
 endProgram();  
 }  
  
 /\*\*  
 \* CGI methods  
 \*/  
  
 printf("Installing CGI methods\n");  
 installCGIMethods();  
  
 printf("Reset Game status\n");  
 resetGame();  
  
 /\*  
 \* MENU  
 \*/  
 byte key;  
 while (1)  
 {  
 printf("-~= Menu =~-\n");  
 printf("------------\n");  
 printf("[X] End program\n");  
 printf("[R] Reboot\n");  
 printf("[D] Debug RAM Dump\n");  
 printf("[S] Set RAM manually\n");  
 printf("[G] Debug Game State\n");  
 printf("[U] Print all known users\n");  
 printf("[I] Interrupt test to AVR\n");  
  
 scanf("%c%\*c", &key);  
  
 switch (key & ~0x20)  
 {  
 case 'X':  
 endProgram();  
 break;  
 case 'R':  
 BIOS\_Reboot();  
 break;  
 case 'D':  
 ramdump();  
 break;  
 case 'S':  
 manualram();  
 break;  
 case 'G':  
 debugGameState();  
 break;  
 case 'U':  
 printAllUsers();  
 break;  
 case 'I':  
 test();  
 break;  
 default:  
 printf("Char not in menu: %c\n", key);  
 }  
 }  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* WEB  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* TEMPLATE PARTS  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
char \* pageHeader =  
 "<!doctype html><html><head><meta charset='us-ascii'/><meta name='viewport' content='width=400' /><title>Mastermind</title><link href='http://fonts.googleapis.com/css?family=Open+Sans:600,400' rel='stylesheet' type='text/css'><style type='text/css'>body,html{font-family:'Open Sans',sans-serif;margin:0 auto;padding:0;height:100%%;min-height:100%%;position:relative;max-width:500px}#wrapper{padding:10px 10px 30px}.center{text-align:center}.black{background-color:#000;color:#fff}.white{background-color:#fff}.orange{background-color:orange}.purple{background-color:purple;color:#fff}.yellow{background-color:#ff0}.aqua{background-color:#0ff}.pink{background-color:#ff1493}.blue{background-color:#00f;color:#fff}.green{background-color:green;color:#fff}.red{background-color:red}header h1{margin:1px}header ul{margin:1px;padding:1px}header ul li{display:inline;padding:0 10px;margin:1px;border:1px solid #000;border-radius:5px;box-shadow:2px 2px 3px #888}header ul li a{tekst-decoration:none;color:#000}.guesses{width:100%%;text-align:center;;border-collapse:collapse}.guesses .txt{padding:0 10px}.guesses \* tr td{border:1px solid #000}footer{position:absolute;bottom:0;height:30px;padding:0 10px}footer a{text-decoration:none;color:#d3d3d3;font-size:smaller}input[type=submit], .btn{display:inline;padding:3px 10px;margin:1px;border:1px solid #000;border-radius:5px;box-shadow:2px 2px 3px #888;background:#fff;text-decoration:none;color:#000}</style></head><body><div id='wrapper'>";  
  
char \* pageFooter =  
 "</div><footer><a class='center' href='http://www.dries007.net/'>&copy; Dries007.net - 2015</a></footer></body></html>";  
  
char \* pickUsername =  
 "<h2>Welcome new player!</h2><p>Before you can play, you need to pick a username:</p><form method='post' action='pickUsername'><input type='text' name='username' maxlength='20'/><input type='submit' value='Check'/></form>";  
  
char \* noGameYet = "<p>No game is going yet, but you can start one!</p>";  
  
char \* gameAvailable =  
 "<p>A game is currently being played. Go ahead an join!</p>";  
  
/\*\*  
 \* Needs 2 extra strings per count. First string is URL, second is name.  
 \*/  
void addMenuItems(char \* buffer, byte count, ...)  
{  
 strcat(buffer, "<header class='center'><h1>Mastermind</h1><ul>");  
  
 va\_list ap;  
 va\_start(ap, count);  
  
 for (byte i = 0; i < count; i++)  
 {  
 strcat(buffer, "<li><a href='");  
 strcat(buffer, va\_arg(ap, char \*));  
 strcat(buffer, "'>");  
 strcat(buffer, va\_arg(ap, char \*));  
 strcat(buffer, "</a></li>");  
 }  
  
 va\_end(ap);  
 strcat(buffer, "</ul></header>");  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* HOME (GET)  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void huge \_pascal \_saveregs cgiHomeFunction(rpCgiPtr CgiRequest)  
{  
 static char pageBuffer[2048]; // Buffer to contain web page  
 //char tmpBuffer[512]; // Buffer for string manipulation functions  
  
 sprintf(pageBuffer, pageHeader);  
  
 User \* user = getUserByIP(CgiRequest->fRemoteIPPtr);  
 if (user == NULL)  
 {  
 addMenuItems(pageBuffer, 0);  
 strcat(pageBuffer, pickUsername);  
 }  
 else  
 {  
 if (getGame()->state == STATE\_NO\_GAME) addMenuItems(pageBuffer, 1,  
 "start", "Start a game");  
 else addMenuItems(pageBuffer, 1, "play", "Play");  
  
 strcat(pageBuffer, "<h2>Welcome ");  
 strcat(pageBuffer, user->name);  
 strcat(pageBuffer, "</h2>");  
  
 if (getGame()->state == STATE\_NO\_GAME) strcat(pageBuffer, noGameYet);  
 else strcat(pageBuffer, gameAvailable);  
 }  
  
 strcat(pageBuffer, pageFooter);  
  
 CgiRequest->fHttpResponse = CgiHttpOk;  
 CgiRequest->fDataType = CGIDataTypeHtml;  
 CgiRequest->fResponseBufferPtr = pageBuffer;  
 CgiRequest->fResponseBufferLength = strlen(pageBuffer);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* PICK USERNAME (POST)  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void huge \_pascal \_saveregs cgiPickUsernameFunction(rpCgiPtr CgiRequest)  
{  
 static char pageBuffer[2048]; // Buffer to contain web page  
 //char tmpBuffer[512]; // Buffer for string manipulation functions  
  
 sprintf(pageBuffer, pageHeader);  
  
 char \* name;  
 char \* value;  
 while (CGI\_GetArgument(&name, &value, CgiRequest) == CGI\_ARGUMENT\_ERR\_OK)  
 {  
 if (strcmp(name, "username") == 0)  
 {  
 if (getUserByName(value) != NULL)  
 {  
 addMenuItems(pageBuffer, 0);  
  
 strcat(pageBuffer, "<p>Sorry, ");  
 strcat(pageBuffer, value);  
 strcat(pageBuffer,  
 " is already in use. Pick another name please:</p><form method='post' action='pickUsername'><input type='text' name='username'/><input type='submit' value='Check'/></form>");  
 }  
 else  
 {  
 addUser(\*((long \*) CgiRequest->fRemoteIPPtr), value);  
  
 if (getGame()->state == STATE\_NO\_GAME) addMenuItems(pageBuffer, 1,  
 "start", "Start a game");  
 else addMenuItems(pageBuffer, 1, "play", "Play");  
  
 strcat(pageBuffer, "<p>You are now known as ");  
 strcat(pageBuffer, value);  
 strcat(pageBuffer, "!</p>");  
  
 if (getGame()->state == STATE\_NO\_GAME) strcat(pageBuffer,  
 noGameYet);  
 else strcat(pageBuffer, gameAvailable);  
 }  
 }  
 }  
  
 strcat(pageBuffer, pageFooter);  
  
 CgiRequest->fHttpResponse = CgiHttpOk;  
 CgiRequest->fDataType = CGIDataTypeHtml;  
 CgiRequest->fResponseBufferPtr = pageBuffer;  
 CgiRequest->fResponseBufferLength = strlen(pageBuffer);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* START (BOTH)  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void huge \_pascal \_saveregs cgiStartFunction(rpCgiPtr CgiRequest)  
{  
 static char pageBuffer[2048]; // Buffer to contain web page  
 char tmpBuffer[512]; // Buffer for string manipulation functions  
  
 sprintf(pageBuffer, pageHeader);  
  
 User \* user = getUserByIP(CgiRequest->fRemoteIPPtr);  
 Game \* game = getGame();  
 if (user == NULL)  
 {  
 addMenuItems(pageBuffer, 0);  
 strcat(pageBuffer, pickUsername);  
 }  
 else if (game->state == STATE\_NO\_GAME  
 || game->state == STATE\_GAME\_CONFIGURED) // If game is not (fully) configured yet  
 {  
 char \* name;  
 char \* value;  
 while (CGI\_GetArgument(&name, &value, CgiRequest)  
 == CGI\_ARGUMENT\_ERR\_OK) // Argument parse loop  
 {  
 if (strcmp(name, "mode") == 0) // Gamemode  
 {  
 game->vsPlayer = strcmp(value, "Player") == 0;  
 game->state = STATE\_GAME\_CONFIGURED;  
 }  
 else if (strcmp(name, "colors") == 0) // # of colors  
 {  
 game->colors = atoi(value);  
 game->state = STATE\_GAME\_CONFIGURED;  
 }  
 else // Colors of code  
 {  
 int i;  
 sscanf(name, "c%d", &i);  
 game->code[i] = atoi(value);  
 game->state = STATE\_GAME\_STARTED;  
 }  
 }  
  
 if (game->state == STATE\_GAME\_CONFIGURED) // If game is partially configured (argument parser above)  
 {  
 if (game->vsPlayer) // Print color picker code  
 {  
 game->host = user;  
 addMenuItems(pageBuffer, 0);  
 strcat(pageBuffer,  
 "<p>Pick your code:</p><form method='get' action='start'>");  
 for (byte i = 0; i < 4; i++) // Color picker 1 -> 4  
 {  
 sprintf(tmpBuffer, "<select name='c%d'>", i);  
 strcat(pageBuffer, tmpBuffer);  
  
 for (byte c = 0; c < game->colors; c++)  
 {  
 sprintf(tmpBuffer, "<option value='%d' class='%s'>%s</option>",  
 c, ALL\_COLOR\_CLASSES[c], ALL\_COLOR\_CLASSES[c]);  
 strcat(pageBuffer, tmpBuffer);  
 }  
  
 strcat(pageBuffer, "</select>");  
 }  
 strcat(pageBuffer,  
 "<input type='submit' value='Choose!'/></form>");  
 }  
 else // VS computer  
 {  
 addMenuItems(pageBuffer, 1, "play", "Play");  
 strcat(pageBuffer, "<h2>Game started!</h2>");  
 setRndCode(game->colors);  
 game->state = STATE\_GAME\_STARTED;  
 }  
 }  
 else if (game->state == STATE\_GAME\_STARTED) // Game started  
 {  
 addMenuItems(pageBuffer, 1, "play", "Play");  
 strcat(pageBuffer, "<h2>Game started!</h2>");  
 }  
 else // New game form  
 {  
 addMenuItems(pageBuffer, 0);  
 strcat(pageBuffer,  
 "<h2>Start a new game</h2><form method='get' action='start'><p> Player(s) VS <label><input type='radio' name='mode' value='Player' checked/> Host</label><label><input type='radio' name='mode' value='Computer'/> Computer</label></p><p><label for='colors'># of colors: </label><select id='colors' name='colors'><option>4</option><option>6</option><option>8</option></select></p><input type='submit' value='Go!'/></form>");  
 }  
 }  
 else // Already going  
 {  
 addMenuItems(pageBuffer, 1, "play", "Play");  
 strcat(pageBuffer,  
 "<h2>Start a new game</h2><p>A game has already been started.</p>");  
 }  
  
 strcat(pageBuffer, pageFooter);  
  
 CgiRequest->fHttpResponse = CgiHttpOk;  
 CgiRequest->fDataType = CGIDataTypeHtml;  
 CgiRequest->fResponseBufferPtr = pageBuffer;  
 CgiRequest->fResponseBufferLength = strlen(pageBuffer);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* PLAY (GET)  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void huge \_pascal \_saveregs cgiPlayFunction(rpCgiPtr CgiRequest)  
{  
 static char pageBuffer[2048]; // Buffer to contain web page  
 char tmpBuffer[512]; // Buffer for string manipulation functions  
  
 sprintf(pageBuffer, pageHeader);  
  
 User \* user = getUserByIP(CgiRequest->fRemoteIPPtr);  
 Game \* game = getGame();  
  
 if (user == NULL)  
 {  
 addMenuItems(pageBuffer, 0);  
 strcat(pageBuffer, pickUsername);  
 }  
 else if (game->state == STATE\_NO\_GAME) // If game is not configured yet  
 {  
 addMenuItems(pageBuffer, 1, "start", "Start a game");  
 strcat(pageBuffer, noGameYet);  
 }  
 else  
 {  
 addMenuItems(pageBuffer, 0);  
  
 if (game->state == STATE\_GAME\_STARTED)  
 {  
 // Process guess, if any  
 char \* name;  
 char \* value;  
 byte i = 0xFF;  
 while (CGI\_GetArgument(&name, &value, CgiRequest)  
 == CGI\_ARGUMENT\_ERR\_OK)  
 {  
 int p, c; //p = position, c = color id  
 sscanf(name, "c%d", &p);  
 c = atoi(value);  
 if (i == 0xFF) i = game->nrOfGuesses++;  
 game->guesses[i][p] = c;  
 }  
 if (i != 0xFF) guessRow(i);  
 }  
  
 // Display guess table  
  
 strcat(pageBuffer,  
 "<p>Guesses:</p><table class='guesses' border> <tr> <th class='txt'>#</th>");  
 for (byte c = 0; c < COLORS; c++)  
 strcat(pageBuffer, "<th style='min-width: 50px;'></th>");  
 strcat(pageBuffer,  
 "<th class='txt'>Exact</th> <th class='txt'>Color</th> </tr>");  
  
 for (byte i = 0; i < game->nrOfGuesses; i++)  
 {  
 sprintf(tmpBuffer, "<tr><td style='padding: 0 10px'>%d</td>", i);  
 strcat(pageBuffer, tmpBuffer);  
  
 for (byte c = 0; c < COLORS; c++)  
 {  
 char \* color = ALL\_COLOR\_CLASSES[game->guesses[i][c]];  
 sprintf(tmpBuffer, "<td class='%s'>%s</td>", color, color);  
 strcat(pageBuffer, tmpBuffer);  
 }  
 sprintf(tmpBuffer, "<td>%d</td><td>%d</td></tr>",  
 game->guesses[i][COLORS], game->guesses[i][COLORS + 1]);  
 strcat(pageBuffer, tmpBuffer);  
 }  
  
 strcat(pageBuffer, "</table>");  
  
 if (game->state == STATE\_GAME\_OVER)  
 {  
 strcat(pageBuffer,  
 "<p>Game over! The host/computer won!</p><a href='reset' class='btn'>Reset</a>");  
 }  
 else if (game->state == STATE\_GAME\_WON)  
 {  
 strcat(pageBuffer,  
 "<p>Game over! The codebreaker(s) won!</p><a href='reset' class='btn'>Reset</a>");  
 }  
 else if (game->vsPlayer && game->host == user)  
 {  
 strcat(pageBuffer, "<p>You picked the code, you can't guess.</p>");  
 }  
 else if (game->state == STATE\_GAME\_STARTED) // Let user make guess  
 {  
 strcat(pageBuffer,  
 "<p>Make a guess:</p><form method='get' action='play'>");  
 for (byte i = 0; i < COLORS; i++) // Color picker 1 -> 4  
 {  
 sprintf(tmpBuffer, "<select name='c%d'>", i);  
 strcat(pageBuffer, tmpBuffer);  
  
 for (byte c = 0; c < game->colors; c++)  
 {  
 byte selected = game->nrOfGuesses != 0  
 && game->guesses[game->nrOfGuesses - 1][i] == c;  
 sprintf(tmpBuffer,  
 "<option value='%d' class='%s' %s >%s</option>", c,  
 ALL\_COLOR\_CLASSES[c], selected ? "selected" : "",  
 ALL\_COLOR\_CLASSES[c]);  
 strcat(pageBuffer, tmpBuffer);  
 }  
  
 strcat(pageBuffer, "</select>");  
 }  
 strcat(pageBuffer, "<input type='submit' value='Choose!'/></form>");  
 }  
 else  
 {  
 strcat(pageBuffer,  
 "<p>Code is not yet picked. Refresh the page to get a status update.</p>");  
 }  
 }  
 strcat(pageBuffer, pageFooter);  
  
 CgiRequest->fHttpResponse = CgiHttpOk;  
 CgiRequest->fDataType = CGIDataTypeHtml;  
 CgiRequest->fResponseBufferPtr = pageBuffer;  
 CgiRequest->fResponseBufferLength = strlen(pageBuffer);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* RESET (GET)  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
void huge \_pascal \_saveregs cgiResetFunction(rpCgiPtr CgiRequest)  
{  
 static char pageBuffer[2048]; // Buffer to contain web page  
 char tmpBuffer[512]; // Buffer for string manipulation functions  
  
 sprintf(pageBuffer, pageHeader);  
  
 User \* user = getUserByIP(CgiRequest->fRemoteIPPtr);  
 Game \* game = getGame();  
  
 if (user == NULL)  
 {  
 addMenuItems(pageBuffer, 0);  
 strcat(pageBuffer, pickUsername);  
 }  
 else if (game->state == STATE\_GAME\_OVER || game->state == STATE\_GAME\_WON) // If game done  
 {  
 resetGame();  
  
 addMenuItems(pageBuffer, 1, "start", "Start a game");  
 strcat(pageBuffer, "<p>The game has been reset.</p>");  
 }  
 else  
 {  
 strcat(pageBuffer, "<p>Incorrect game state.</p>");  
 }  
  
 strcat(pageBuffer, pageFooter);  
  
 CgiRequest->fHttpResponse = CgiHttpOk;  
 CgiRequest->fDataType = CGIDataTypeHtml;  
 CgiRequest->fResponseBufferPtr = pageBuffer;  
 CgiRequest->fResponseBufferLength = strlen(pageBuffer);  
}  
  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 \* ALL INSTALL / REMOVE LOGIC  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
  
typedef void huge \_pascal \_saveregs (\*CGIfn)(rpCgiPtr); // Because function pointer syntax in unreadable  
  
char \*cgiNames[] = { "home", "pickUsername", "start", "play", "reset" };  
int cgiMethods[] = { CgiHttpGet, CgiHttpPost, CgiHttpGet, CgiHttpGet,  
 CgiHttpGet };  
CGIfn cgiFunctions[] = { cgiHomeFunction, cgiPickUsernameFunction,  
 cgiStartFunction, cgiPlayFunction, cgiResetFunction };  
  
void installCGIMethods()  
{  
 CGI\_Entry cgiEntry;  
  
 for (byte i = 0; i < 5; i++)  
 {  
 cgiEntry.PathPtr = cgiNames[i];  
 cgiEntry.CgiFuncPtr = cgiFunctions[i];  
 cgiEntry.method = cgiMethods[i];  
  
 if (CGI\_Install(&cgiEntry) != 0)  
 {  
 printf("Installing CGI function %s failed\n", cgiEntry.PathPtr);  
 endProgram();  
 }  
 }  
}  
  
void removeCGIMethods()  
{  
 byte n = sizeof(cgiMethods) / sizeof(int);  
 for (byte i = 0; i < n; i++)  
 {  
 if (CGI\_Delete(cgiNames[i]))  
 {  
 printf("Removing %s failed\n", cgiNames[i]);  
 }  
 }

### HTML template

Dit is de niet gecondenseerde versie van de HTML code gebruikt in *mastermind.c* regel 554 t.e.m. 576.

<!DOCTYPE html>  
<html>  
 <head>  
 <meta charset='us-ascii'>  
 <meta content='width=400' name='viewport'>  
 <title>  
 Mastermind  
 </title>  
 <link href='http://fonts.googleapis.com/css?family=Open+Sans:600,400' rel='stylesheet' type='text/css'>  
 <style type='text/css'>  
 body,html {  
 font-family:'Open Sans',sans-serif;  
 margin:0 auto;  
 padding:0;  
 height:100%%;  
 min-height:100%%;  
 position:relative;  
 max-width:500px  
 }  
 header ul li {  
 display:inline;  
 padding:0 10px;  
 margin:1px;  
 border:1px solid #000;  
 border-radius:5px;  
 box-shadow:2px 2px 3px #888  
 }  
 guesses {  
 width:100%%;  
 text-align:center;  
 border-collapse:collapse  
 }  
 footer {  
 position:absolute;  
 bottom:0;  
 height:30px;  
 padding:0 10px  
 }  
 footer a {  
 text-decoration:none;  
 color:#d3d3d3;  
 font-size:smaller  
 }  
 input[type=submit],.btn {  
 display:inline;  
 padding:3px 10px;  
 margin:1px;  
 border:1px solid #000;  
 border-radius:5px;  
 box-shadow:2px 2px 3px #888;  
 background:#fff;  
 text-decoration:none;  
 color:#000  
 }  
 #wrapper { padding:10px 10px 30px }  
 .center { text-align:center }  
 .black { background-color:#000; color:#fff }  
 .white { background-color:#fff }  
 .orange { background-color:orange }  
 .purple { background-color:purple; color:#fff }  
 .yellow { background-color:#ff0 }  
 .aqua { background-color:#0ff }  
 .pink { background-color:#ff1493 }  
 .blue { background-color:#00f; color:#fff }  
 .green { background-color:green; color:#fff }  
 .red { background-color:red }  
 header h1 { margin:1px }  
 header ul { margin:1px; padding:1px }  
 header ul li a { tekst-decoration:none; color:#000 }  
 guesses .txt { padding:0 10px }  
 guesses \* tr td { border:1px solid #000 }  
 </style>  
 </head>  
 <body>  
 <div id='wrapper'></div>  
 <footer>  
 <a class='center' href='http://www.dries007.net/'>&copy; Dries007.net - 2015</a>  
 </footer>  
 </body>  
</html>

### AVR.h

#ifndef AVR\_H\_  
#define AVR\_H\_  
  
#define DEBUG 0  
  
/\* =============== PORT CONFIG ============== \*/  
// Mask to eliminate INT4  
#define LCD\_PORT PORTB  
#define LCD\_DDR DDRB  
// KP = Keypad  
#define KP\_PORT PORTD  
#define KP\_DDR DDRD  
#define KP\_PIN PIND  
// WS2812  
#define LEDS\_PORT PORTE  
#define LEDS\_DDR DDRE  
#define LEDS\_PIN 7  
  
/\* ============== RAM ADDRESSES ============= \*/  
// Offset for External RAM  
#define RAM\_OFFSET 0x8000  
// 170 Leds max (x 3 bytes = 0x1FE)  
#define RAM\_LEDS\_START (RAM\_OFFSET + 0x000)  
#define RAM\_LEDS\_END (RAM\_OFFSET + 0x1FE)  
#define RAM\_LEDS\_AMOUNT (RAM\_OFFSET + 0x1FF)  
// 80 bytes of char buffer for LCD  
#define RAM\_LCD\_START (RAM\_OFFSET + 0x200)  
#define RAM\_LCD\_END (RAM\_OFFSET + 0x250)  
#define RAM\_LCD\_CMD (RAM\_OFFSET + 0x251)  
// Mask for global LED dimming  
#define RAM\_LEDS\_DIM (RAM\_OFFSET + 0x252)  
#define RAM\_KP\_LASTKEY (RAM\_OFFSET + 0x253)  
  
#define RAM\_VERSION\_1 (RAM\_OFFSET + 0x300)  
#define RAM\_VERSION\_2 (RAM\_OFFSET + 0x301)  
  
// Interrupt registers  
#define RAM\_INT\_SEND (RAM\_OFFSET + 0x3FE)  
#define RAM\_INT\_GET (RAM\_OFFSET + 0x3FF)  
  
#define MAX\_LEDS ((uint8\_t)(RAM\_LEDS\_END - RAM\_LEDS\_START))  
#define MAX\_LCD\_CHARS ((uint8\_t)(RAM\_LCD\_END - RAM\_LCD\_START))  
  
/\* ================ CMD CODES =============== \*/  
#define CMD\_LEDS\_SEND 0x01 // Clock out LEDS  
#define CMD\_LCD\_CHAR 0x02 // Print char buffer to LCD  
#define CMD\_LCD\_CMD 0x03 // Send instruction byte to LCD  
#define CMD\_LCD\_CL\_PR 0x04 // Send clear + print out char buffer to LCD  
#define CMD\_LCD\_POS 0x05 // Set LCD cursor to position  
#define CMD\_LCD\_BL\_ON 0x06 // Set LCD\_BACKLIGHT = 1   
#define CMD\_LCD\_BL\_OFF 0x07 // Set LCD\_BACKLIGHT = 0  
  
#define SATUS\_KP\_PRESS 0x01 // Key was pressed  
  
/\* ================= MACROS ================= \*/  
/\*   
 !WARNING! The data-lines on my custom PCB are flipped on the AVR side.   
 This means that these macros use the '\_\_builtin\_avr\_insert\_bits' macro to flip the data read/written from/to the DP-RAM.  
 If you don't need this, replace '\_\_builtin\_avr\_insert\_bits (0x01234567, {value}, 0)' with '{value}'.  
\*/  
#define PONTER\_RAM(addr) ( ((volatile uint8\_t \*) addr) )  
#define READ\_RAM(addr) ( \_\_builtin\_avr\_insert\_bits (0x01234567, \*PONTER\_RAM(addr), 0) )  
#define WRITE\_RAM(addr, val) { \*PONTER\_RAM(addr) = \_\_builtin\_avr\_insert\_bits (0x01234567, val, 0); }  
  
/\* ================= STRUCTS ================ \*/  
// Correct byte order!  
struct cRGB { uint8\_t g; uint8\_t r; uint8\_t b; };  
  
/\* ================= GLOBALS ================ \*/  
struct cRGB LEDS[MAX\_LEDS]; // LED data, in correct byte order  
uint8\_t LCD\_BACKLIGHT = 0; // LCD back light pin status (pin 7)  
  
/\* ================ FUNCTIONS =============== \*/  
  
/\* ---------------- inits ----------------  
 \* Parameters: N/a  
 \* Actions:  
 \* - Enable External Memory Interface  
 \* - Set `LCD\_PORT` as output (with `LCD\_MASK`)  
 \* - Set `KP\_PORT`.[0->3] as output  
 \* - Set `KP\_PORT`.[4->8] as input  
 \* - Set `LEDS\_PORT`.`LEDS\_PIN` as output  
 \* - Init & Clear LCD  
 \* - Backlight LCD ON  
 \* - LCD Cursor OFF  
 \*/  
void inline inits();  
  
/\* ---------------- readMatrix ----------------  
 \* Parameters:  
 \* - matrix Pin readout of the matrix port (1 = active, invert when using pull-ups)  
 \* Actions:  
 \* - Convert matrix code to ACSII  
 \* Matrix layout:  
 \* Bit | 5 | 6 | 7 | 8   
 \* ----+---+---+---+---  
 \* 1 | 1 | 2 | 3 | A   
 \* 2 | 4 | 5 | 6 | B   
 \* 3 | 7 | 8 | 9 | C   
 \* 4 | \* | 0 | # | D   
 \* Pressing 2 buttons at once results in 0x00, same as no button pressed  
 \*/  
uint8\_t inline readMatrix(uint8\_t matrix);  
  
/\* ---------------- sendLEDS ----------------  
 \* Parameters:  
 \* - amountOfLeds The amount of led information to send  
 \* Actions:  
 \* - Send LED data in `LEDS` to `LEDS\_PORT`.`LEDS\_PIN`  
 \* - Delay 50µs for reset pulse  
 \* Prerequisites:  
 \* - `LEDS\_PORT`.`LEDS\_PIN` has been set as output  
 \*/  
void inline sendLEDS(uint16\_t amountOfLeds);  
  
/\* ---------------- sendLCDNible ----------------  
 \* Parameters:  
 \* - data Populate 4 lower bits  
 \* - rs 1 for character, 0 for instruction  
 \* Actions:  
 \* - Mask data with 0x0F  
 \* - Set data bit 7 (Backlight) to `LCD\_BACKLIGHT`  
 \* - Set data bit 6 (RegisterSelect) to`rs`  
 \* - Set data bit 5 (Enable) to 1  
 \* - Set data bit 4 (INT4) to 1 (for Pull-up)  
 \* - Write data to `LCD\_PORT`  
 \* - delay 1 µs  
 \* - Toggle bit 5 (Enable)  
 \* - delay 2 µs  
 \*/  
void sendLCDNible(uint8\_t data, uint8\_t rs);  
  
/\* ---------------- sendLCDInstructionByte ----------------  
 \* Parameters:  
 \* - data Data to send to the instruction register  
 \* Actions:  
 \* - Call sendLCDNible(data >> 4, 0)  
 \* - Call sendLCDNible(data, 0)  
 \* - delay 50µs  
 \*/  
void sendLCDInstructionByte(uint8\_t data);  
  
/\* ---------------- sendLCDCharacterByte ----------------  
 \* Parameters:  
 \* - data Data to send to the character register  
 \* Actions:  
 \* - Call sendLCDNible(data >> 4, 1)  
 \* - Call sendLCDNible(data, 1)  
 \* - delay 50µs  
 \*/  
void sendLCDCharacterByte(char data);  
  
/\* ---------------- sendLCDBuffer ----------------  
 \* Parameters:  
 \* - \*buffer Pointer to character buffer  
 \* Actions:  
 \* - For every character in the buffer or until `MAX\_LCD\_CHARS` is reached:  
 \* - Call sendLCDCharacterByte(data)  
 \*/  
void sendLCDBuffer(char \* buffer);  
  
/\* ############## START SECTION WS2812 DRIVER ##############  
 \* Original Source: https://github.com/cpldcpu/light\_ws2812/  
 \* Original Author: Tim (cpldcpu@gmail.com)   
 \* Original License: GNU GPL V2 (https://github.com/cpldcpu/light\_ws2812/blob/master/License.txt)  
 \* This license still applies to everything between the "WS2812 DRIVER" section lines.  
 \*  
 \* Modifications by Dries007:  
 \* - Changed configuration  
 \* - Merged library into single set of source and header files  
 \*/  
// Timing in ns  
#define w\_zeropulse 350  
#define w\_onepulse 900  
#define w\_totalperiod 1250  
  
// Fixed cycles used by the inner loop  
#define w\_fixedlow 2  
#define w\_fixedhigh 4  
#define w\_fixedtotal 8   
  
// Insert NOPs to match the timing, if possible  
#define w\_zerocycles (((F\_CPU/1000)\*w\_zeropulse )/1000000)  
#define w\_onecycles (((F\_CPU/1000)\*w\_onepulse +500000)/1000000)  
#define w\_totalcycles (((F\_CPU/1000)\*w\_totalperiod +500000)/1000000)  
  
// w1 - nops between rising edge and falling edge - low  
#define w1 (w\_zerocycles-w\_fixedlow)  
// w2 nops between fe low and fe high  
#define w2 (w\_onecycles-w\_fixedhigh-w1)  
// w3 nops to complete loop  
#define w3 (w\_totalcycles-w\_fixedtotal-w1-w2)  
  
#if w1>0  
 #define w1\_nops w1  
#else  
 #define w1\_nops 0  
#endif  
  
// The only critical timing parameter is the minimum pulse length of the "0"  
// Warn or throw error if this timing can not be met with current F\_CPU settings.  
#define w\_lowtime ((w1\_nops+w\_fixedlow)\*1000000)/(F\_CPU/1000)  
#if w\_lowtime>550  
 #error "WS2812 DRIVER: Sorry, the clock speed is too low. Did you set F\_CPU correctly?"  
#elif w\_lowtime>450  
 #warning "WS2812 DRIVER: The timing is critical and may only work on WS2812B, not on WS2812(S)."  
 #warning "Please consider a higher clockspeed, if possible"  
#endif   
  
#if w2>0  
#define w2\_nops w2  
#else  
#define w2\_nops 0  
#endif  
  
#if w3>0  
#define w3\_nops w3  
#else  
#define w3\_nops 0  
#endif  
  
#define w\_nop1 "nop \n\t"  
#define w\_nop2 "rjmp .+0 \n\t"  
#define w\_nop4 w\_nop2 w\_nop2  
#define w\_nop8 w\_nop4 w\_nop4  
#define w\_nop16 w\_nop8 w\_nop8  
  
/\* ############## END SECTION WS2812 DRIVER ############## \*/  
  
#endif /\* AVR\_H\_ \*/

### AVR.c

#include <stdio.h>  
#include <stdlib.h>  
#include <avr/interrupt.h>  
#include <avr/io.h>  
#include <avr/pgmspace.h>  
#include <util/delay.h>  
  
#include "AVR.h"  
  
/\* ======= INTERRUPT SERVICE ROUTINES ======= \*/  
// Interrupt Service Routine for INT4  
ISR(INT4\_vect)  
{  
 // read interrupt address, also clears interrupt signal  
 volatile uint8\_t cmd = READ\_RAM(RAM\_INT\_GET);   
   
 switch (cmd)  
 {  
 // Copy LED data from DPRAM into RAM (sets correct byte order) and clock out the data  
 case CMD\_LEDS\_SEND:   
 {  
 // Amount of LEDS connected  
 uint8\_t n = READ\_RAM(RAM\_LEDS\_AMOUNT);  
 // Global LED dimmer settings   
 uint8\_t dim = READ\_RAM(RAM\_LEDS\_DIM);   
 // Make sure that n <= MAX\_LEDS to prevent data corruption  
 if (n > MAX\_LEDS) n = MAX\_LEDS;  
 // used for DPRAM address offset from RAM\_LEDS\_START  
 uint16\_t offset = 0;  
 for (uint16\_t i = 0; i < n; i++)  
 {  
 LEDS[i].r = READ\_RAM(RAM\_LEDS\_START + (offset ++)) & dim;  
 LEDS[i].g = READ\_RAM(RAM\_LEDS\_START + (offset ++)) & dim;  
 LEDS[i].b = READ\_RAM(RAM\_LEDS\_START + (offset ++)) & dim;  
 }  
 // Clock out data  
 sendLEDS(n);  
 }  
 break;  
 // Shortcut command to set LCD cursor position  
 case CMD\_LCD\_POS:  
 {  
 sendLCDInstructionByte(READ\_RAM(RAM\_LCD\_CMD) | 0b10000000);  
 }  
 break;  
 // Send LCD an insrtuction byte  
 case CMD\_LCD\_CMD:  
 {  
 sendLCDInstructionByte(READ\_RAM(RAM\_LCD\_CMD));  
 }  
 break;  
 // Shortcut for clear & print  
 case CMD\_LCD\_CL\_PR:   
 {  
 sendLCDInstructionByte(0x01);  
 \_delay\_ms(10);  
 }  
 // no break!  
 // Print char buffer (until max chars or 0x00)  
 case CMD\_LCD\_CHAR:  
 {  
 for (uint8\_t i = 0; i < MAX\_LCD\_CHARS; i++)  
 {  
 uint8\_t c = READ\_RAM(RAM\_LCD\_START + i);  
 if (c == 0x00) break;  
 sendLCDCharacterByte(c);  
 }  
 }  
 break;  
 case CMD\_LCD\_BL\_ON: LCD\_BACKLIGHT = 1; break;  
 case CMD\_LCD\_BL\_OFF: LCD\_BACKLIGHT = 0; break;  
 }  
 /\* Interrupt detection debug code \*/  
 #if DEBUG  
 char buff[10];  
 sprintf(buff, "I:0x%02X", cmd);  
 sendLCDBuffer(buff);  
 #endif  
 sei();  
}  
  
/\* ================== DEBUG STUFF ================== \*/  
/\* HANDLE DEBUG KEYPRESS HERE \*/  
void debugKeypress(uint8\_t key)  
{  
 // buffer index pointer  
 static uint8\_t b = 0;  
 // buffer (20 chars = 1 line)  
 static char buffer[20];   
   
 switch (key)  
 {  
 // SEND  
 case '\*':  
 {  
 sendLEDS(30);  
   
 b = 0;  
 buffer[b] = 0;  
 sendLCDInstructionByte(0x01);  
 \_delay\_ms(2);  
 sprintf(buffer, " 0x%02X 0x%02X 0x%02X", LEDS[0].r, LEDS[0].g, LEDS[0].b);  
 sendLCDBuffer(buffer);  
 // 1e pos on lcd  
 sendLCDInstructionByte(0x80);  
 break;  
 }  
 // BACKSPACE  
 case '#':  
 {  
 if (b != 0) b--;  
 buffer[b] = 0;  
   
 sendLCDInstructionByte(0x01);  
 \_delay\_ms(2);  
   
 sendLCDBuffer(buffer);  
 break;  
 }  
 // SET RED  
 case 'A':  
 {  
 uint8\_t nr = atoi(buffer);  
 for (uint8\_t i = 0; i < 30; i++)  
 {  
 LEDS[i].r = nr;  
 }  
 b = 0;  
 buffer[b] = 0;  
 sendLCDInstructionByte(0x01);  
 \_delay\_ms(2);  
 break;  
 }  
 // SET GREEN  
 case 'B':  
 {  
 uint8\_t nr = atoi(buffer);  
 for (uint8\_t i = 0; i < 30; i++)  
 {  
 LEDS[i].g = nr;  
 }  
 b = 0;  
 buffer[b] = 0;  
 sendLCDInstructionByte(0x01);  
 \_delay\_ms(2);  
 break;  
 }  
 // SET BLUE  
 case 'C':  
 {  
 uint8\_t nr = atoi(buffer);  
 for (uint8\_t i = 0; i < 30; i++)  
 {  
 LEDS[i].b = nr;  
 }  
 b = 0;  
 buffer[b] = 0;  
 sendLCDInstructionByte(0x01);  
 \_delay\_ms(2);  
 break;  
 }  
 case 'D':  
 {  
 sendLCDInstructionByte(0x01);  
 for (uint8\_t i = 0; i < MAX\_LCD\_CHARS; i++)  
 {  
 uint8\_t c = READ\_RAM(RAM\_LCD\_START + i);  
 if (c == 0x00) break;  
 sendLCDCharacterByte(c);  
 }  
 }  
 break;  
 // NUMBER  
 default:  
 {  
 buffer[b++] = key;  
 buffer[b] = 0;  
 sendLCDCharacterByte(key);  
 break;  
 }  
 }  
}  
  
/\* ================== MAIN ================== \*/  
int main()  
{  
 // set ports & interrupt registers  
 inits();  
   
 #if DEBUG  
 // debounce variables  
 uint8\_t prevKey = 0x00;  
 uint16\_t downTime = 0;  
 uint16\_t upTime = 0;  
  
 // debug program loop (aka keypad scanner)  
 while (1)   
 {  
 // ROW loop  
 for (uint8\_t r = 0; r < 4; r++)  
 {  
 // all pins HIGH, the row we want to read LOW; bit 0-4 always high because they are inputs (pull-up).  
 KP\_PORT = 0x0F | (0b11101111 << r);  
 // Convert read port byte (inverted because pull-ups)   
 uint8\_t key = readMatrix(~KP\_PIN);   
 // no key pressed  
 if (key == 0x00)  
 {  
 // if no key was pressed for 100+ ms, reset debounce.  
 if (upTime++ > 100)  
 {  
 // Makes sure the next keypress will register instantly  
 prevKey = 0x00;  
   
 downTime = 0;  
 upTime = 0;  
 }  
 }  
 else // A key was pressed  
 {  
 // if the pressed key is different from the last one OR its been pressed for 500+ ms, acknowledge as a legitimate press  
 if (prevKey != key || downTime++ > 500)  
 {  
 // Store key in DPRAM for SC12  
 WRITE\_RAM(RAM\_KP\_LASTKEY, key);  
 // Send interrupt to SC12  
 WRITE\_RAM(RAM\_INT\_SEND, SATUS\_KP\_PRESS);   
   
 debugKeypress(key);  
   
 // Store current key for debounce  
 prevKey = key;   
   
 downTime = 0;  
 upTime = 0;  
 }  
 }  
   
 \_delay\_ms(1);  
 }  
 }  
 #else   
 while (1)  
 {  
   
 }  
 #endif  
}  
  
/\* ================ FUNCTIONS =============== \*/  
void inline inits()  
{  
 // Write SRE to 1 enables the External Memory Interface  
 MCUCR = 0x80;  
   
 // Magic numbers  
 WRITE\_RAM(RAM\_VERSION\_1, 42);   
 WRITE\_RAM(RAM\_VERSION\_2, 0x42);  
   
 // LCD Port Setup  
 LCD\_DDR = 0xFF;  
   
 // Keypad Port Setup (bit 0-3 = in; bit 4-7 = out)  
 KP\_DDR = 0xF0;  
   
 // LED port all output  
 LEDS\_DDR = 0xEF;  
 LEDS\_PORT = (uint8\_t)~0xEF;  
   
 // Enable falling edge interrupt INT4  
 EICRB = 0x02;  
 EIMSK = 0x10;  
   
 // LCD init  
 \_delay\_ms(100);  
   
 // Set 4 bit mode  
 sendLCDNible(0x02, 0);  
 // 2-line mode, display on  
 sendLCDInstructionByte(0x0C);  
 \_delay\_ms(100);  
   
 // Display ON/OFF Control  
 sendLCDInstructionByte(0x0F);  
 // Clear & home  
 sendLCDInstructionByte(0x01);  
 \_delay\_ms(20);  
 // Entry mode Increment & Entire shift off  
 sendLCDInstructionByte(0x06);  
   
 // Clear any open interrupts.  
 volatile uint8\_t i = READ\_RAM(RAM\_INT\_GET);  
 // Global interrupts ON  
 sei();  
}  
  
uint8\_t inline readMatrix(uint8\_t matrix)  
{  
 switch (matrix)  
 {  
 default: return 0x00;  
   
 case 0b00010001: return '1'; // 0x11  
 case 0b00010010: return '4'; // 0x12  
 case 0b00010100: return '7'; // 0x13  
 case 0b00011000: return '\*'; // 0x18  
   
 case 0b00100001: return '2'; // 0x21  
 case 0b00100010: return '5'; // 0x22  
 case 0b00100100: return '8'; // 0x24  
 case 0b00101000: return '0'; // 0x28  
   
 case 0b01000001: return '3'; // 0x41  
 case 0b01000010: return '6'; // 0x42  
 case 0b01000100: return '9'; // 0x44  
 case 0b01001000: return '#'; // 0x48  
   
 case 0b10000001: return 'A'; // 0x81  
 case 0b10000010: return 'B'; // 0x82  
 case 0b10000100: return 'C'; // 0x84  
 case 0b10001000: return 'D'; // 0x88  
 }  
}  
  
/\* ############## START SECTION WS2812 DRIVER ##############   
 \* Original Source: https://github.com/cpldcpu/light\_ws2812/  
 \* Original Author: Tim (cpldcpu@gmail.com)   
 \* Original License: GNU GPL V2 (https://github.com/cpldcpu/light\_ws2812/blob/master/License.txt)  
 \* This license still applies to everything between the "WS2812 DRIVER" section lines.  
 \*  
 \* Modifications by Dries007:  
 \* - Changed configuration  
 \* - Merged library into single set of source and header files  
 \*/  
void inline sendLEDS(uint16\_t leds)  
{  
 // 3 colors!  
 uint16\_t datlen = leds + leds + leds;  
 // Type cast  
 uint8\_t \* data = (uint8\_t \*) LEDS;  
   
 // Save interrupt status  
 uint8\_t sreg\_prev = SREG;  
 // We can't be interrupted!  
 cli();  
   
 uint8\_t maskhi = \_BV(LEDS\_PIN);  
 // Low mask  
 uint8\_t masklo = ~maskhi&LEDS\_PORT;  
 // High mask  
 maskhi |= LEDS\_PORT;  
   
 // used in ASM  
 uint8\_t curbyte, ctr;  
 while (datlen--)  
 {  
 curbyte = \*data ++; // Grab byte  
   
 asm volatile(  
 " ldi %0,8 \n\t" // Write 8 (00001000) to Loop counter (%0)  
 "loop%=: \n\t" // Loop entry point (%= is a unique number on each asm statement)  
 " out %2,%3 \n\t" // Write High mask (%3) to LED\_PORT (%2)  
 #if (w1\_nops&1) // w1 nops for timing  
 w\_nop1  
 #endif  
 #if (w1\_nops&2)  
 w\_nop2  
 #endif  
 #if (w1\_nops&4)  
 w\_nop4  
 #endif  
 #if (w1\_nops&8)  
 w\_nop8  
 #endif  
 #if (w1\_nops&16)  
 w\_nop16  
 #endif  
 " sbrs %1,7 \n\t" // Skip next instruction if bit 7 of Data (%1) is set  
 " out %2,%4 \n\t" // Write Low mask (%4) to LED\_PORT (%2)  
 " lsl %1 \n\t" // Shift Data (%1) left  
 #if (w2\_nops&1) // w2 nops for timing  
 w\_nop1  
 #endif  
 #if (w2\_nops&2)  
 w\_nop2  
 #endif  
 #if (w2\_nops&4)  
 w\_nop4  
 #endif  
 #if (w2\_nops&8)  
 w\_nop8  
 #endif  
 #if (w2\_nops&16)  
 w\_nop16  
 #endif  
 " out %2,%4 \n\t" // Write Low mask (%4) to LED\_PORT (%2)   
 #if (w3\_nops&1) // w3 nops for timing  
 w\_nop1  
 #endif  
 #if (w3\_nops&2)  
 w\_nop2  
 #endif  
 #if (w3\_nops&4)  
 w\_nop4  
 #endif  
 #if (w3\_nops&8)  
 w\_nop8  
 #endif  
 #if (w3\_nops&16)  
 w\_nop16  
 #endif  
 " dec %0 \n\t" // Decrement Loop counter (%0) (Also sets Z if 0x00)  
 " brne loop%=\n\t" // Jump to Loop entry point if Z is set.  
 // %0 = 8 bit loop counter  
 : "=&d" (ctr)  
 : "r" (curbyte), "I" (\_SFR\_IO\_ADDR(LEDS\_PORT)), "r" (maskhi), "r" (masklo)  
 // %s1 = Data %2 = LEDS\_PORT %3 = high mask %4 = low mask  
 );  
 }  
 SREG = sreg\_prev; // Restore interrupt status  
 \_delay\_us(50); // Reset delay  
}  
/\* ############## END SECTION WS2812 DRIVER ############## \*/  
  
void sendLCDNible(volatile uint8\_t data, uint8\_t rs)  
{  
 // Mask out fist 4 bits  
 data &= 0b00001111;  
 // Mask in LCD\_BACKLIGHT if required (pin 7)s  
 if (LCD\_BACKLIGHT) data |= 0b10000000;  
 // Mask in register select  
 if (rs) data |= 0b01000000;  
 // Bit 4 => 1, its the interrput pin, its on pull-up!  
 data |= 0b00010000;  
 // Set Data  
 LCD\_PORT = data;  
 // Small delay, data needs to be valid BEFORE enable  
 \_delay\_us(800);  
 // Toggle enable  
 LCD\_PORT |= 0b00100000;  
 // Larger delay, LCD needs time to process  
 \_delay\_us(800);  
}  
  
void sendLCDInstructionByte(uint8\_t data)  
{  
 // rs = 0 -> instruction  
 sendLCDNible(data >> 4, 0);  
 sendLCDNible(data, 0);  
}  
  
void sendLCDCharacterByte(char data)  
{  
 // rs = 1 -> data  
 sendLCDNible(data >> 4, 1);  
 sendLCDNible(data, 1);  
}  
  
void sendLCDBuffer(char \* buffer)  
{  
 // max chars  
 for (uint8\_t i = 0; i < MAX\_LCD\_CHARS; i++)  
 {  
 // End on 0x00  
 if (buffer[i] == 0x00) break;  
 sendLCDCharacterByte(buffer[i]);  
 }  
}

1. ftp://193.191.150.44/pub/CD-Microcontrollers/RIOT/ [↑](#footnote-ref-1)
2. [github.com/adafruit/Adafruit\_NeoPixel](https://github.com/adafruit/Adafruit_NeoPixel) [↑](#footnote-ref-2)
3. [wp.josh.com/2014/05/13/ws2812](http://wp.josh.com/2014/05/13/ws2812) en [cpldcpu.wordpress.com/2014/01/14/light\_ws2812-](http://cpldcpu.wordpress.com/2014/01/14/light_ws2812-) [↑](#footnote-ref-3)
4. [github.com/cpldcpu/light\_ws2812/tree/master/light\_ws2812\_AVR/Light\_WS2812](https://github.com/cpldcpu/light_ws2812/tree/master/light_ws2812_AVR/Light_WS2812) [↑](#footnote-ref-4)