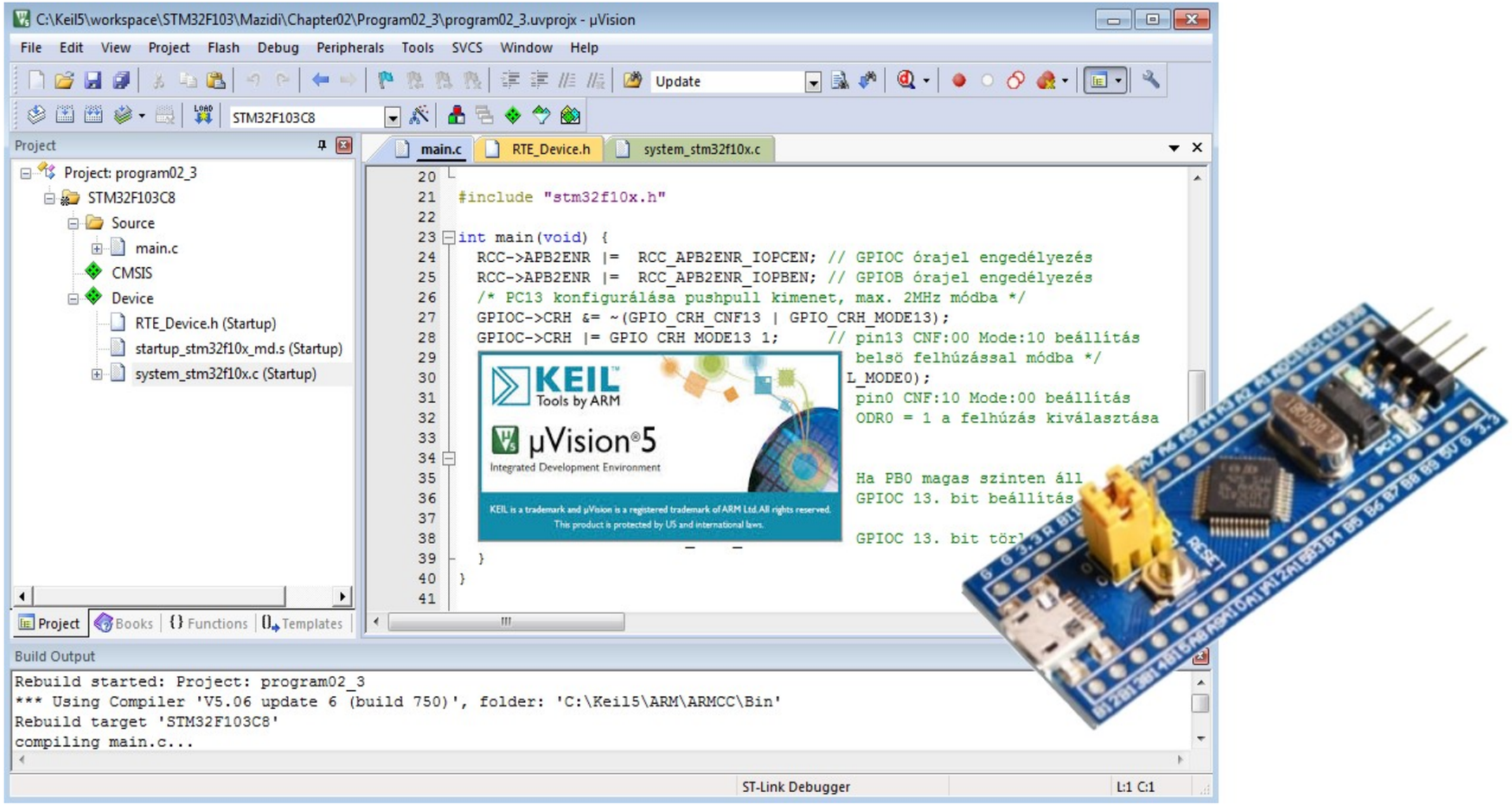
STM32 mikrovezérlők programozása ARM Keil környezetben

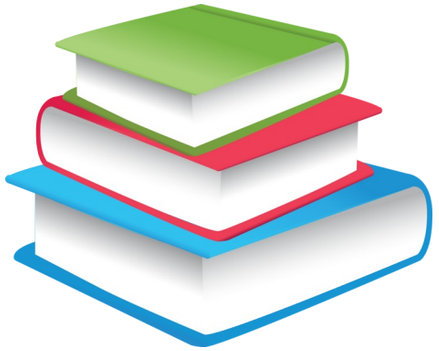


18. ST7735 1.8” színes TFT kijelzők vezérlése – 2. rész

Felhasznált és ajánlott irodalom

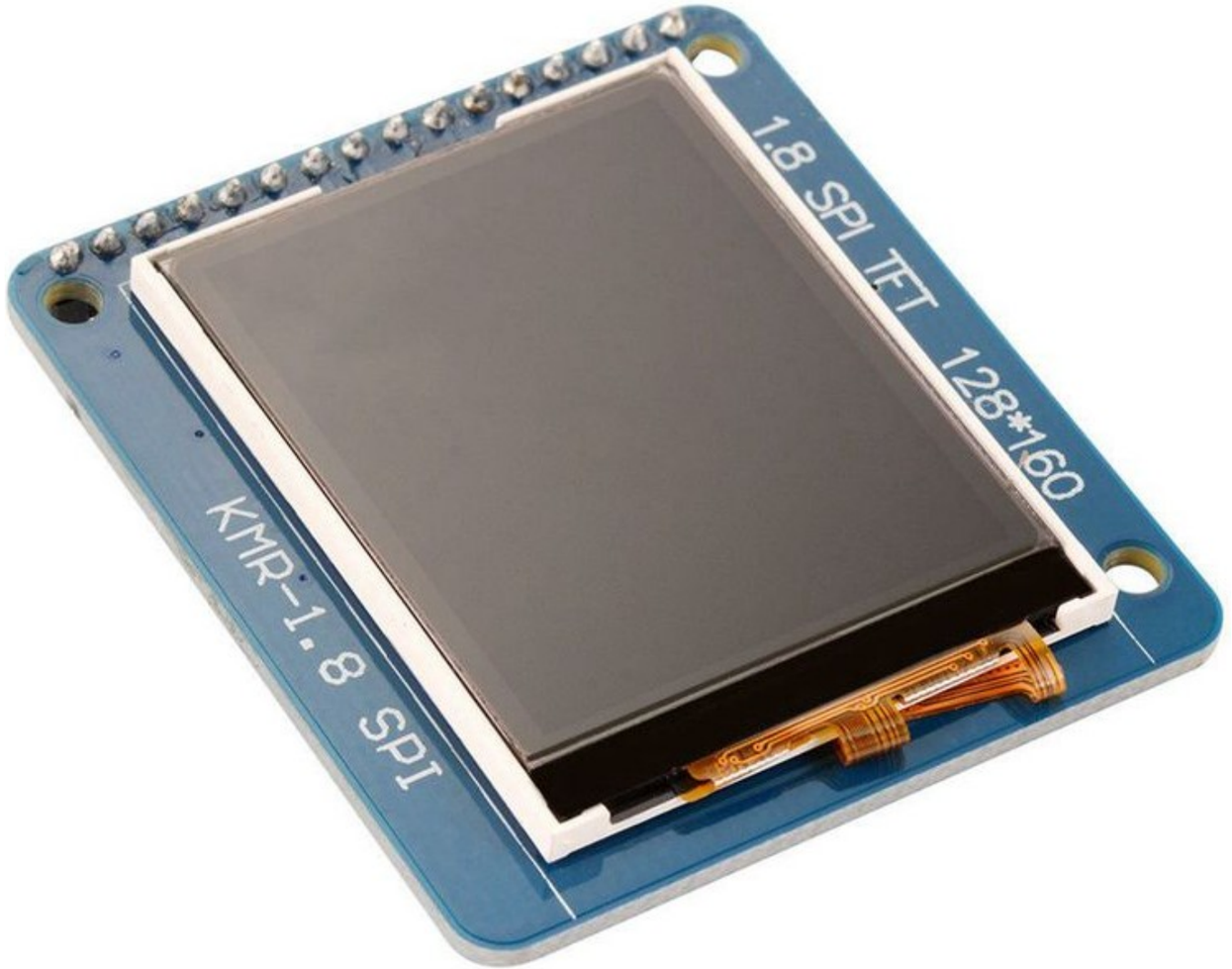
- Joseph Yiu: The Definitive Guide To The ARM CORTEX-M3
- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi: STM32 Arm Programming for Embedded Systems
- Alexander Tarasov: **Курс «Штурмуем STM32»**
- ARM Keil MDK Getting started
- **STM32F103C8** adatlap és termékinfo
- **STM32F103** Family Reference Manual

- Sitronix ST7735 adatlap
- Adafruit: Adafruit-ST7735-Library
- Dung-Yi Chao: STM32F4-ST7735 tutorial
- Michael Abrash: Graphics Programming Black Book



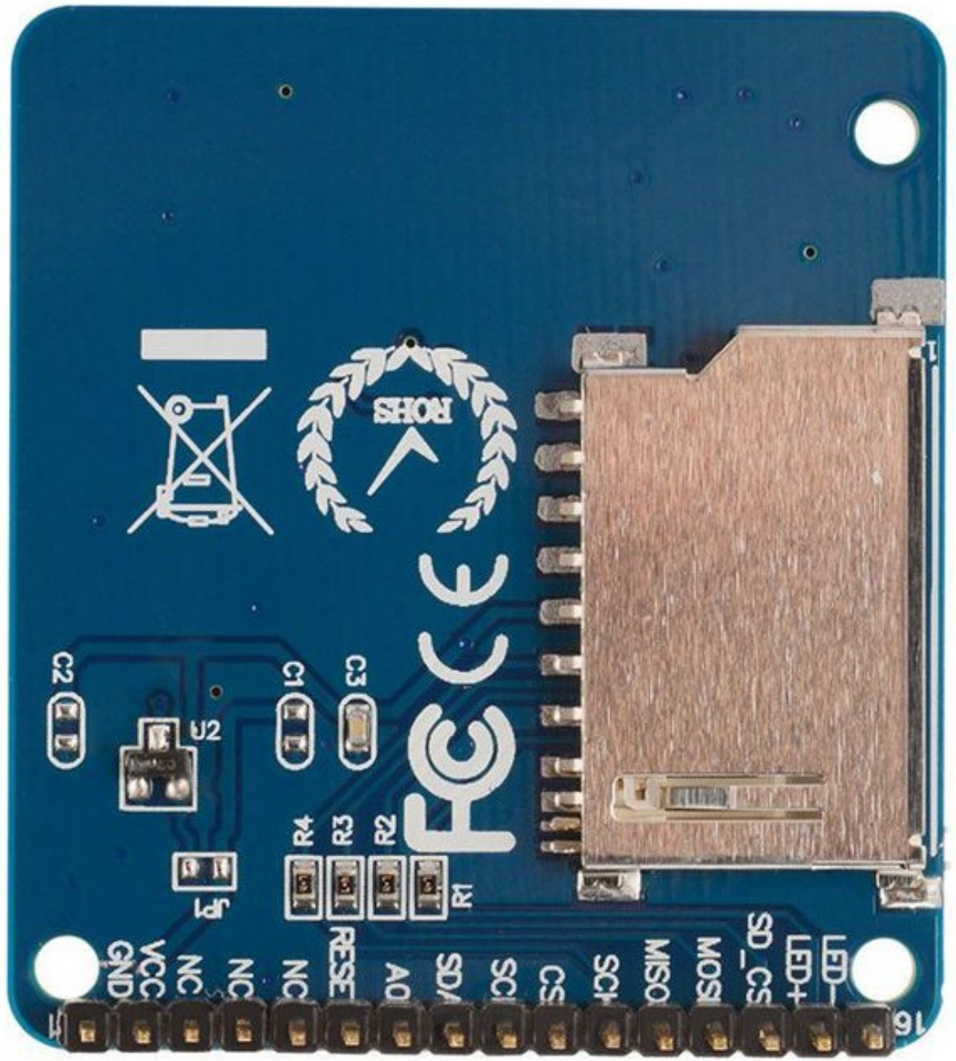
ST7735 1.8” színes LCD kijelző

- Vezérlő: **Sitronix ST7735**
- Interfész: SPI, csak írás (max. 10 – 15 MHz)
- Data/Command választás külön vonalon
- Kijelző: TFT, 65 536 szín (16 bit RGB, 5-6-5)
- Felbontás: **128 x 160** pixel
- SD kártya foglalat:
- Csak SPI mód kivezetések:
  - SD_CS
  - MOSI
  - MISO
  - SCK

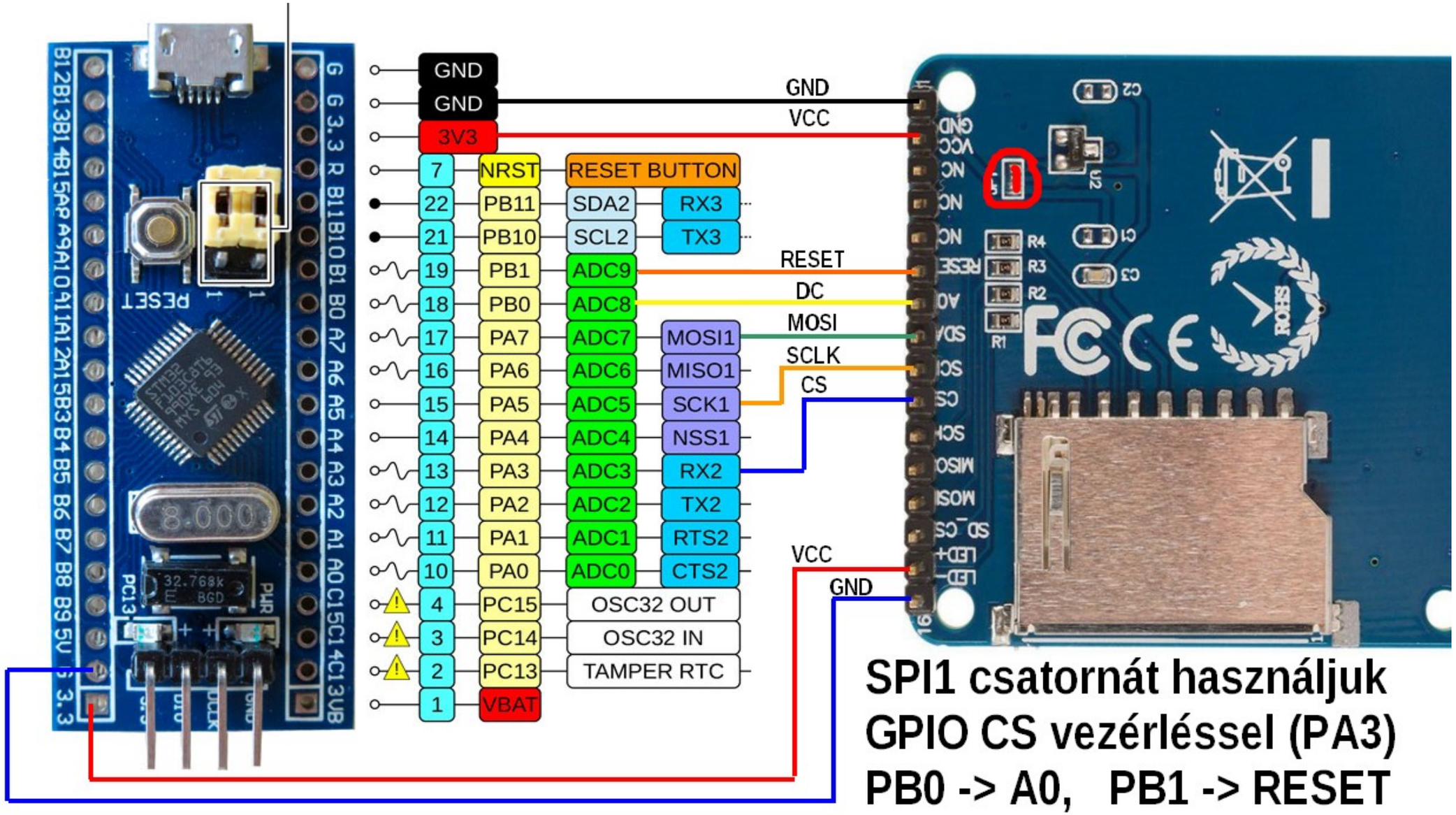


A kivezetések

- **GND** – a tápegység közös pontja
- **VCC** – tápfeszültség (5V, vagy **JP1** zárása után 3.3V)
- **NC** – nincs bekötve
- **RESET** – kijelző reset jel (0: aktív, 1: inaktív)
- **A0** – kijelző regisztertválasztó jel (RS, DC)
- **SDA** – kijelző SPI adatbemenet (MOSI jel)
- **SCL** – kijelző SPI órajel
- **CS** – kijelző SPI eszközválasztó jel (0: aktív)
- **SCK** – SD kártya SPI órajel
- **MISO** – SD kártya adatkimenete
- **MOSI** – SD kártya adatbemenete
- **SD_CS** – SD kártya eszközválasztó jel (0: aktív)
- **LED+** – kijelző háttérvilágítás (anód)
- **LED–** – kijelző háttérvilágítás (katód)

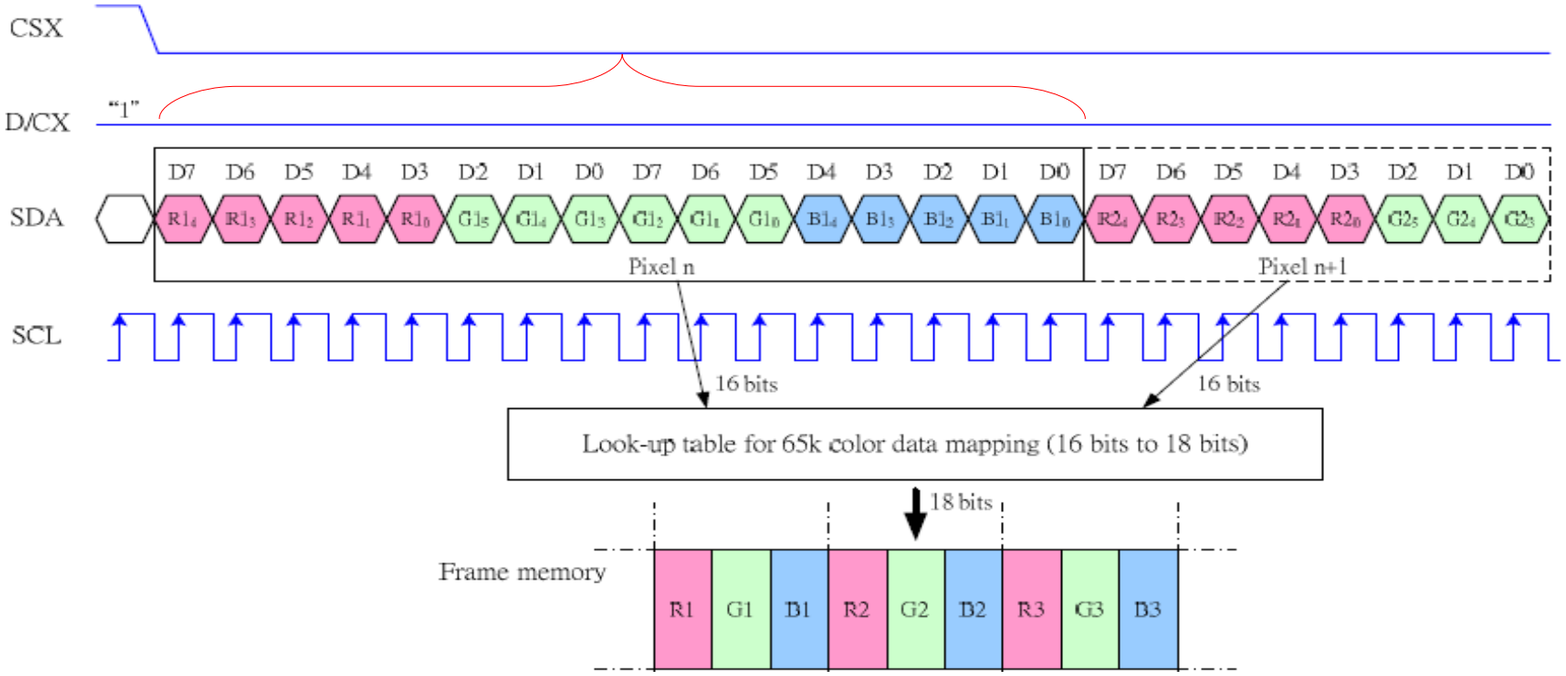


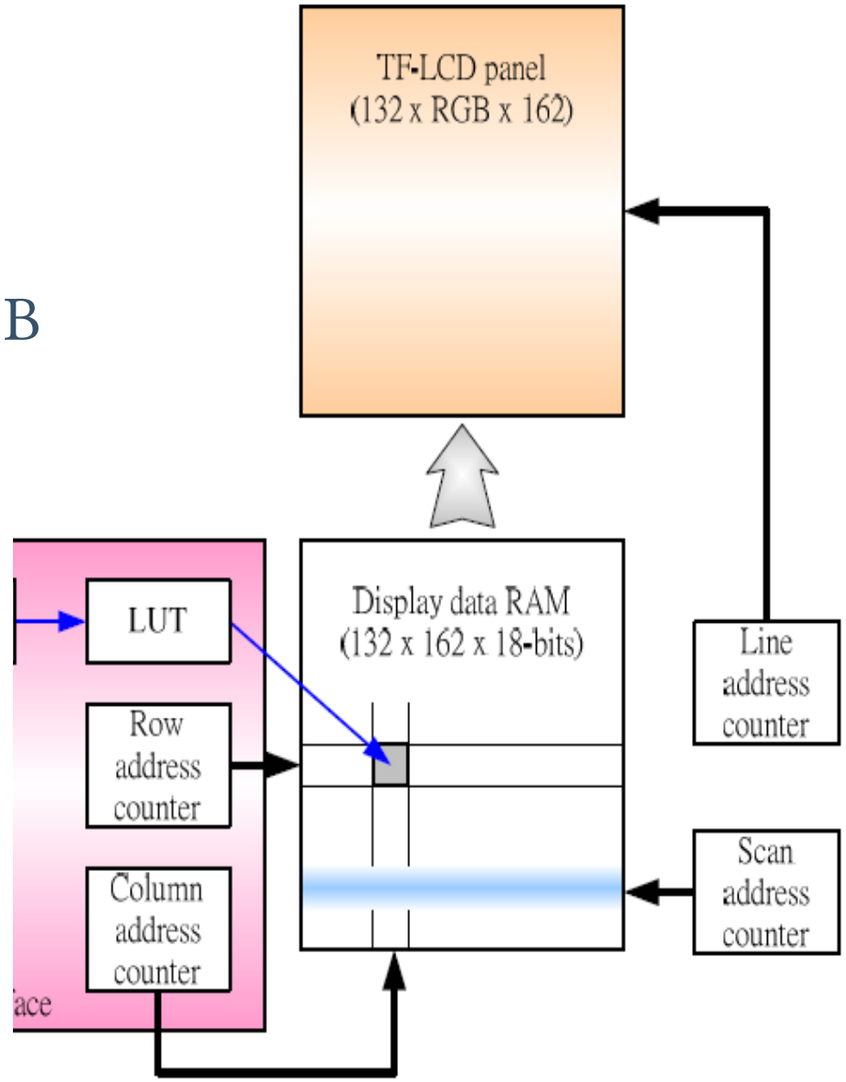
Bekötési vázlat



Színvezérlés 5-6-5 módban

- Az **ST7735** képmemóriája 132 x 162 x 18 bit (RGB)
- Színvezérlési módok: 12-, 16-, vagy 18 bites
- Mi a 16-bites (5-6-5) színvezérlési módot használjuk, amely a **COLMOD** (0x3A) = 0x05 paranccsal választható
- Minden pixelhez két bájtot kell küldeni: 5 bit R, 6 bit G, 5 bit B (az élsimításnál lesz szükségünk erre az információra)



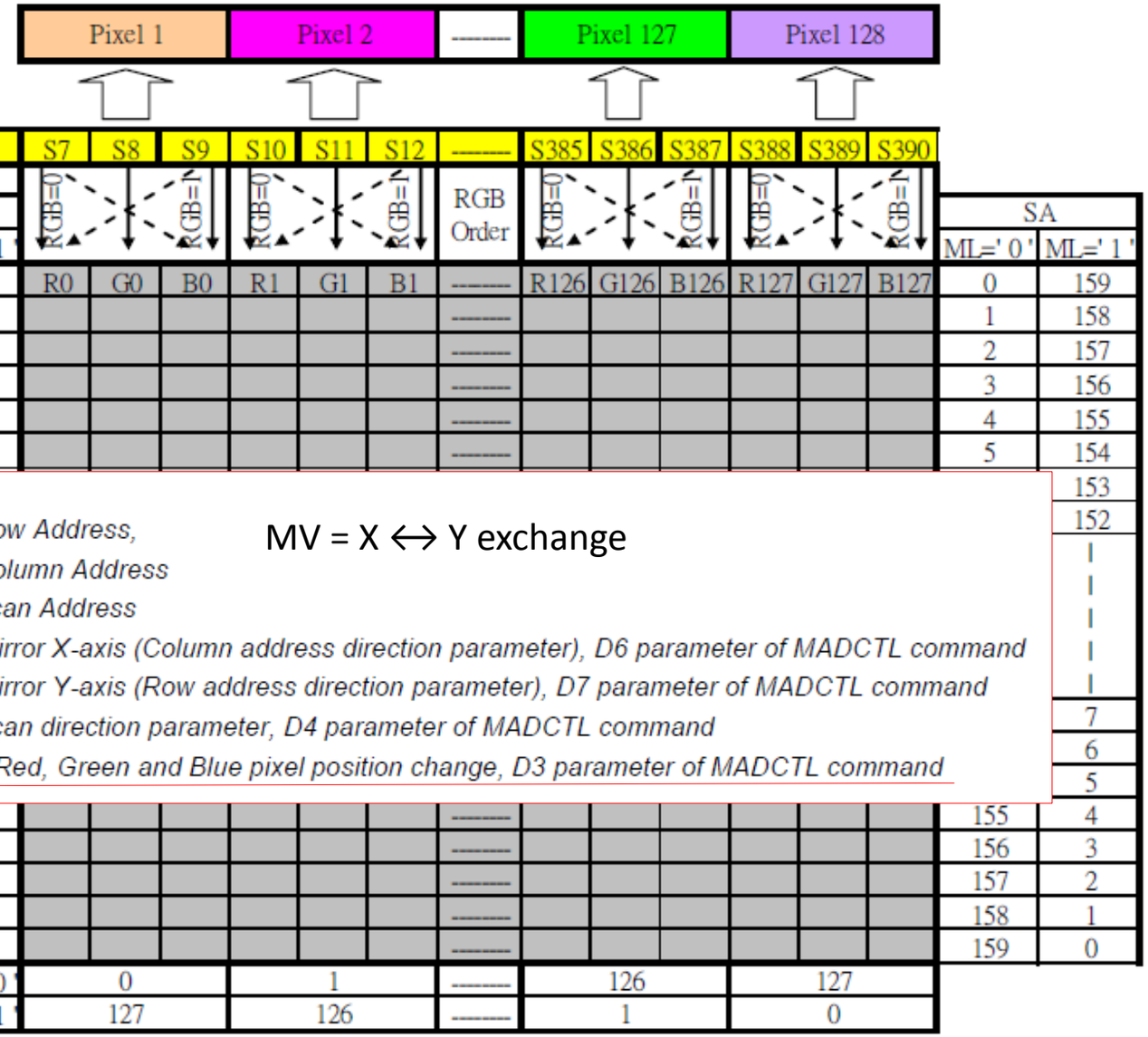


A képmemória

- A 132 x 162 képpontból a kijelzőnk csak 128 x 160-at tud megjeleníteni
- A memória és a kijelző képpontjainak összerendelése a címzést befolyásoló regisztereinek átírásával megváltoztatható
- Ugyancsak megváltoztatható az RGB sorrend BGR-re (bizonyos kijelzőtípusokhoz kell)
- **MADCTL** parancs:

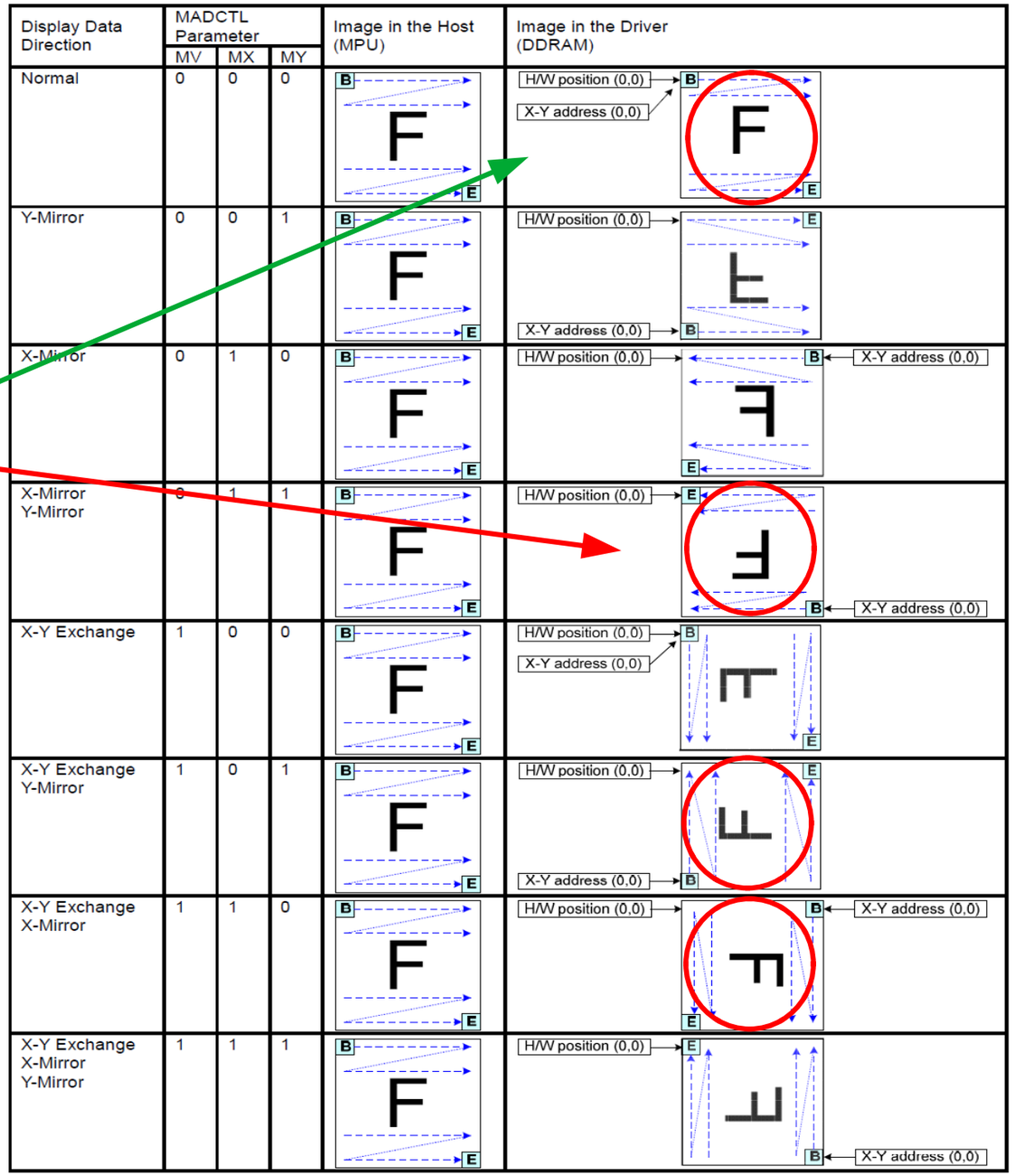
DC	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	1	1	0	1	1	0
1	MY	MX	MV	ML	RGB	MH	-	-

When using 128RGB x 160 resolution (GM[2:0] = "011", SMX=SMY=SRGB= '0')



Képernyőtükrözés, forgatás

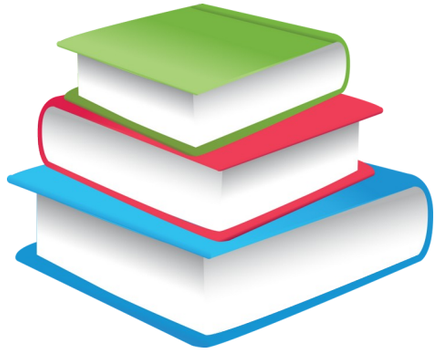
- Az *x* és *y* tengelyre történő tükrözések kombinálásával segítségével fejtetőre állíthatjuk a képet (180 fokos elforgatás)
- Az előző előadásban **ezt a beállítást használtuk MADCTL** (0x36), 0xC8 (MX =1, MY=1)
- **Ha a 0xC8 paraméter helyett nullát írunk, visszafordítható a kép!**
- Az *x*-*y* koordináták felcserélésével és *x* vagy *y* tengelyre történő tükrözéssel elforgathatjuk a képet 90, ill. 270 fokkal is *(lásd: Arduino TFT)*
- Léteznek olyan kijelzőmodulok, amelyeknél a kijelző képpontjai 1 - 2 képponttal eltolva vannak bekötve, ezt is lehet korrigálni a **CASET** és **RASET** parancsok kiadásánál



Kompatibilitás az Arduino ST7735 programkönyvtárakkal

- Ebben az előadásban a **MADCTL** parancs paraméterét (a **ST7735_ROTATION** szimbólum az **st7735.h** fejléc állományban) 0 értékkel definiáltuk. Így az **st7735.c** kódban található inicializálás megegyezik azzal, mintha az Adafruit_ST7735 programkönyvtárral így inicializálnánk a kijelzőt:

```
#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library
#include <SPI.h>
Adafruit_ST7735 tft = Adafruit_ST7735(cs, dc, rst);
tft.initR(INITR_BLACKTAB);
tft.rotation(0);
```



- Az **Arduino TFT** könyvtárában nekünk kell testre szabni a **TFT.cpp** állományt így:

```
TFT::TFT(uint8_t CS, uint8_t RS, uint8_t RST) : Adafruit_ST7735(CS, RS, RST) {
  _width = ST7735_TFTWIDTH;
  _height = ST7735_TFTHEIGHT;
}

void TFT::begin() {
  initR(INITR_BLACKTAB);
  setRotation(0);
}
```

Grafikus elemek kirajzolása

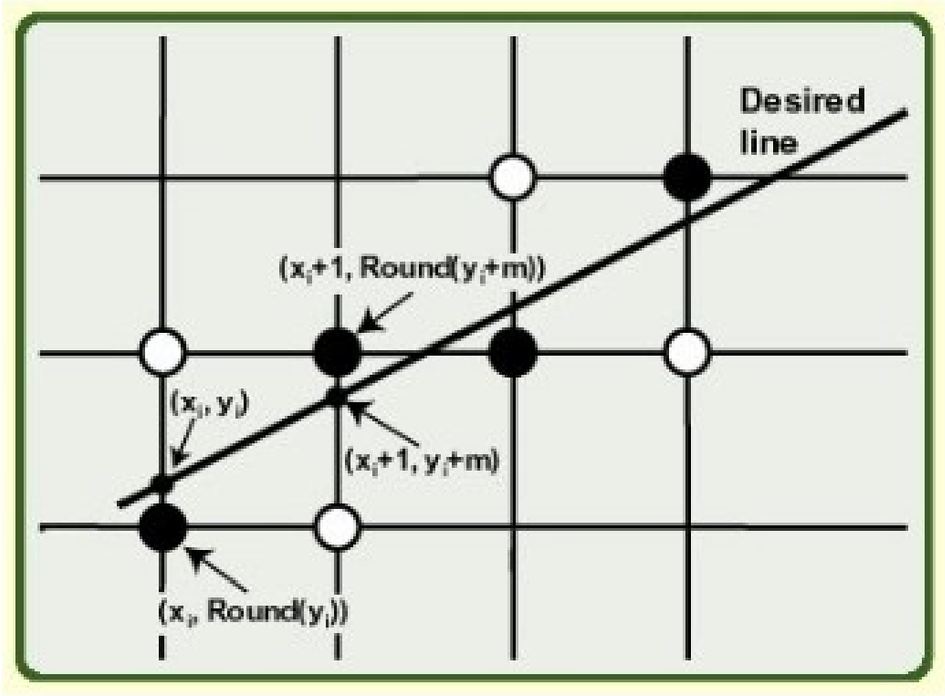
- Dung-Yi Chao STM32F4-ST7735 programkönyvtárában található grafikus függvények:
- void **ST7735_DrawPixel**(uint16_t x, uint16_t y, uint16_t color)
- uint32_t **ST7735_DrawString**(uint16_t x, uint16_t y, char *pt, int16_t textColor, uint8_t size);
  (ezt a függvényt módosítottuk, hogy a karakterméret változtatható legyen)
- void **ST7735_DrawCharS**(int16_t x, int16_t y,char c,int16_t textColor,int16_t bgColor, uint8_t size);
- void **ST7735_FillRectangle**(uint16_t x, uint16_t y, uint16_t w, uint16_t h, uint16_t color);
- void **ST7735_FillScreen**(uint16_t color)
- void **ST7735_InvertColors**(bool invert)
- void **ST7735_DrawFastHLine**(int16_t x, int16_t y, int16_t w, uint16_t color)
- void **ST7735_DrawFastVLine**(int16_t x, int16_t y, int16_t h, uint16_t color)

Új függvényekkel bővítjük:

- void **ST7735_DrawLine**(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t color)
- void **DrawWuLine**(uint16_t X0, uint16_t Y0, uint16_t X1, uint16_t Y1)
- void **ST7735_DrawCircle**(uint16_t x0, uint16_t y0, uint16_t r, uint16_t c)
- void **ST7735_DrawFilledCircle**(uint16_t x0, uint16_t y0, uint16_t r, uint16_t c)

Számítógépes grafika: szakasz rajzolása

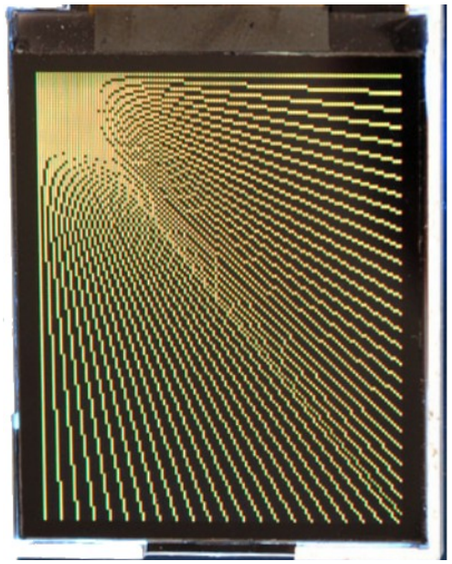
- A kijelzőn diszkrét pontokat tudunk kijelezni, a ferde szakaszok emiatt" lépcsősek" lesznek
- **Jack Elton Bresenham** (1962, IBM algoritmusa: Egy (x_k,y_k) pontban arról kell dönteni, hogy az (x_k+1,y_k), vagy az (x_k+1,y_k+1) pontok közül melyik esik közelebb az egyeneshez
- Elképzelés: **slope_error**-ban halmozódik az eltérés, s ha nagyobb 0.5-nél, ***y*** lép egyet, **slope_error** értékét pedig csökkentjük 1-gyel
- Egészaritmetika: az **m = (y0-y0)/(x1-x0)** meredeksé helyett annak (x1-x0)-lal felszorzott értékét használju a **slope_error** aktualizálásánál



- A fenti, 0.5-del való összevetés helyett -dy/2 eltolással kezdünk, így csak az előjelet kell figyelnünk az összehasonlításnál.
- Forrás: Bresenhams line generation algorithm (tutorialspont.dev) (lásd még: **keil19_10** előadás)

Szakaszrajzolás Bresenham algoritmussal (részlet)

```
void ST7735_DrawLine(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t color) {
    int16_t dx, dy, sx, sy, err, e2, tmp;
    dx = (x0 < x1) ? (x1 - x0) : (x0 - x1);
    dy = (y0 < y1) ? (y1 - y0) : (y0 - y1);
    sx = (x0 < x1) ? 1 : -1;
    sy = (y0 < y1) ? 1 : -1;
    err = ((dx > dy) ? dx : -dy) / 2;
    while (1) {
        ST7735_DrawPixel(x0, y0, color);
        if (x0 == x1 && y0 == y1) {
            break;
        }
        e2 = err;
        if (e2 > -dx) {
            err -= dy;
            x0 += sx;
        }
        if (e2 < dy) {
            err += dx;
            y0 += sy;
        }
    }
}
```



A **keil19_10 (2020. január 30.)** előadásban bemutatott, SSD1306 kijelzőre írt kódot adaptáltuk.

Itt most a függvénynek csak a releváns sorait mutatjuk, a valódi kód ezen kívül az esetleges kilógó részek levágását , illetve a triviális (függőleges, vagy vízszintes szakasz) elkülönítését tartalmazza.

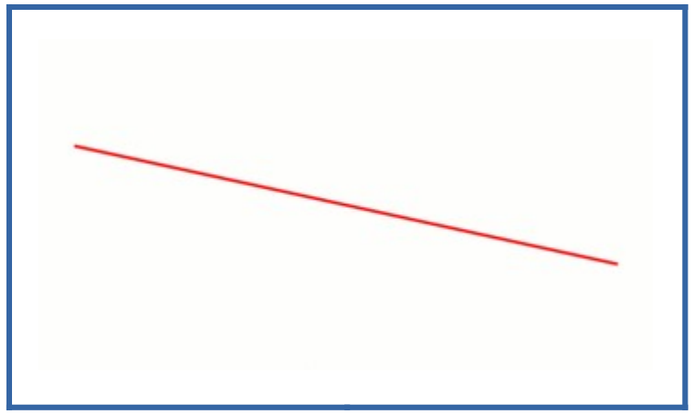
ST7735_3/main.c

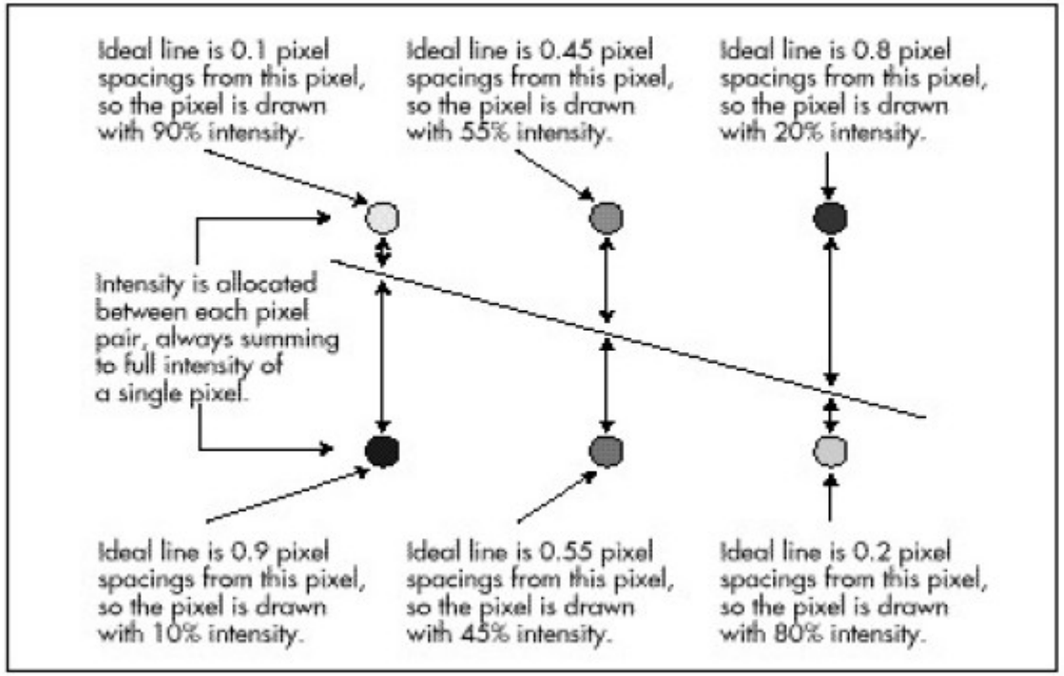
- A **testlines()** függvényt az Adafruit_ST7735 programkönyvtár mintaprogramjából vettük át
- Itt csak a (0,0) pontból kiinduló vonalakat rajzoló részét mutatjuk
- A főprogram elején inicializálni kell a **PB0** (a DC bemenet vezérlése) és a **PB1** (RESET láb vezérlése) kimeneteket
- Az **SPI1** csatorna és a kijelző inicializálása után a végtelen ciklusban végezzük a rajzolást, amely képernyőtörléssel (kék szín) kezdődik, majd fehér vonalakat húz

```
#include "st7735.h"

void testlines(uint16_t color) {
  ST7735_FillScreen(ST7735_BLUE);
  for (int16_t x=0; x < ST7735_WIDTH; x+=6) {
    ST7735_DrawLine(0, 0, x, ST7735_HEIGHT-1, color);
    DelayMs(50);
  }
  for (int16_t y=ST7735_HEIGHT-1; y>=0 ; y-=6) {
    ST7735_DrawLine(0, 0, ST7735_WIDTH-1, y, color);
    DelayMs(50);
  }
}

int main(void) {
  /* PB0-PB1 konfigurálása digitális kimenetként */
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN; // GPIOB órajel engedélyezés
  GPIOB->CRL &= ~0x000000FF;          // CNF és Mode bitek törlése
  GPIOB->CRL |=  0x00000033;          // PB0 és PB1 Pushpull kimenet
  SPI1_init();                        // Az SPI1 modul konfigurálása
  ST7735_Init();                      // A kijelző inicializálása
  while(1) {
    testlines(ST7735_WHITE);
    DelayMs(2000);
  }
}
```

Szakaszrajzolás élsimítással (anti-aliasing)

- Míg az **SSD1306** oled kijelzőnél csak fekete és fehér pontokat rajzolhatunk, az **ST7735** színes kijelző lehetővé teszi az árnyalatok többfokozatú megjelenítését is
- Az élsimítási úgy valósítjuk meg, hogy a képpontok színezését a képzeletbeli egyenestől való távolságuktól függően végezzük
- A **Bresenham algoritmusnál** arról kellett dönteni, hogy az (x_k,y_k) pont rákövetkezője az (x_k+1,y_k), vagy az (x_k+1,y_k+1) pont legyen, melyik esik közelebb az egyeneshez
- **A Xiaolin Wu élsimító algoritmusnál** pedig az (x_k+1,y_k) *és* az (x_k+1,y_k+1) pontot is megjelenítjük, az intenzitást viszont felosztjuk közöttük, az egyenestől való távolságukkal fordított arányban





Forrás: Xiaolin Wu's line algorithm, Wikipedia

Szakaszrajzolás Wu algoritmussal

```
void DrawWuLine(uint16_t X0, uint16_t Y0, uint16_t X1, uint16_t Y1) {
   uint16_t BaseColor = 0;
   uint16_t NumLevels = 32;     // 5 bites színmélységnél ennyi árnyalat lehetséges
   uint16_t IntensityBits = 5; // az 5-6-5 ábrázolásból adódóan...
   uint16_t IntensityShift, ErrorAdj, ErrorAcc;
   uint16_t ErrorAccTemp, Weighting, WeightingComplementMask;
   int16_t DeltaX, DeltaY, Temp, Xdir;
   /* Make sure the line runs top to bottom */
   if (Y0 > Y1) {
      Temp = Y0; Y0 = Y1; Y1 = Temp;
      Temp = X0; X0 = X1; X1 = Temp;
   }
   DeltaY = Y1 - Y0;
   DrawPixel(X0, Y0, BaseColor);
   if ((DeltaX = X1-X0) >= 0) XDir=1; else { XDir = -1; DeltaX = -DeltaX; } // make DeltaX positive
   ...
   /* line is not horizontal, diagonal, or vertical */
   ErrorAcc = 0;  /* initialize the line error accumulator to 0 */
   /* # of bits by which to shift ErrorAcc to get intensity level */
   IntensityShift = 16 - IntensityBits;
   /* Mask used to flip all bits in an intensity weighting, producing the
      result (1 - intensity weighting) */
   WeightingComplementMask = NumLevels - 1;
```

Itt elhagytuk azokat a sorokat, amelyek a vízszintes, függőleges vagy átlós szakaszok kirajzolását végzik

Forrás: Michael Abrash's Graphics Programming Black Book

Szakaszrajzolás Wu algoritmussal

```
if (DeltaY > DeltaX) {    /* Is this an X-major or Y-major line? */
   /* Y-major line; calculate 16-bit fixed-point fractional part of a
      pixel that X advances each time Y advances 1 pixel, truncating the
      result so that we won't overrun the endpoint along the X axis */
   ErrorAdj = ((unsigned long) DeltaX << 16) / (unsigned long) DeltaY;
   /* Draw all pixels other than the first and last */
   while (--DeltaY) {
      ErrorAccTemp = ErrorAcc;   /* remember currrent accumulated error */
      ErrorAcc += ErrorAdj;      /* calculate error for next pixel */
      if (ErrorAcc <= ErrorAccTemp) {
         /* The error accumulator turned over, so advance the X coord */
         X0 += XDir;
     }
      Y0++; /* Y-major, so always advance Y */
      /* The IntensityBits most significant bits of ErrorAcc give us the
         intensity weighting for this pixel, and the complement of the
         weighting for the paired pixel */
      Weighting = ErrorAcc >> IntensityShift;
      DrawPixel(X0, Y0, BaseColor + Weighting);
      DrawPixel(X0 + XDir, Y0, BaseColor + (Weighting ^ WeightingComplementMask));
   }
   DrawPixel(X1, Y1, BaseColor);
   return;
}
```

Forrás: Michael Abrash's Graphics Programming Black Book

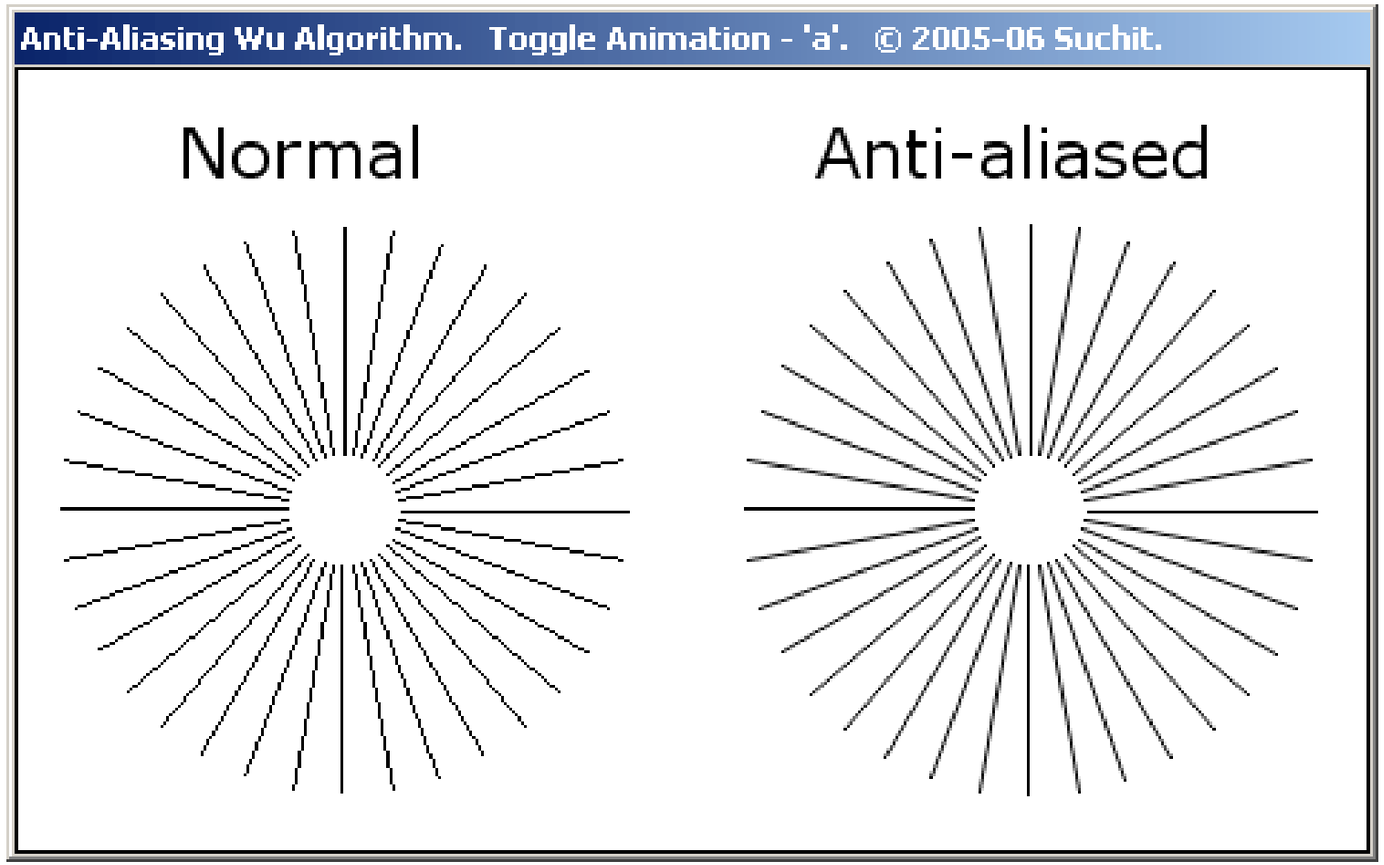
Szakaszrajzolás Wu algoritmussal

```
   /* It's an X-major line; calculate 16-bit fixed-point fractional part of a
      pixel that Y advances each time X advances 1 pixel, truncating the
      result to avoid overrunning the endpoint along the X axis */
   ErrorAdj = ((unsigned long) DeltaY << 16) / (unsigned long) DeltaX;
   /* Draw all pixels other than the first and last */
   while (--DeltaX) {
      ErrorAccTemp = ErrorAcc;   /* remember currrent accumulated error */
      ErrorAcc += ErrorAdj;      /* calculate error for next pixel */
      if (ErrorAcc <= ErrorAccTemp) {
         /* The error accumulator turned over, so advance the Y coord */
         Y0++;
      }
      X0 += XDir; /* X-major, so always advance X */
      /* The IntensityBits most significant bits of ErrorAcc give us the
         intensity weighting for this pixel, and the complement of the
         weighting for the paired pixel */
      Weighting = ErrorAcc >> IntensityShift;
      DrawPixel(X0, Y0, BaseColor + Weighting);
      DrawPixel(X0, Y0 + 1, BaseColor + (Weighting ^ WeightingComplementMask));
   }
   /* Draw the final pixel, which is always exactly intersected by the line and so needs no weighting */
   DrawPixel(X1, Y1, BaseColor);
}
```

Forrás: Michael Abrash's Graphics Programming Black Book

Szakaszrajzolás Wu algoritmussal

- A Code Project oldalán a Suchit nevű felhasználó közzétett egy Windows-os demóprogramot a simítatlan és élsimított szakaszrajzolás különbségének bemutatásra
- Az „a” gomb lenyomása elindítja/leállítja az animációt (az ábrák lassan forogni kezdenek)



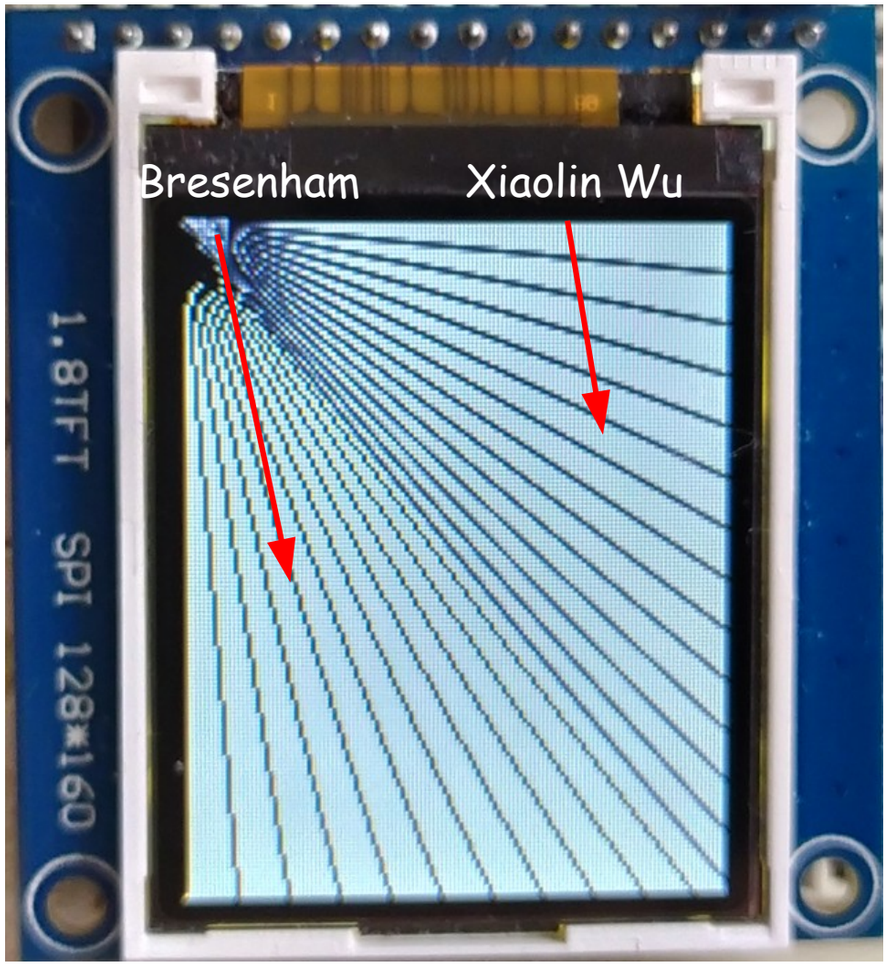
Szakaszrajzolás Wu algoritmussal

- Az előzőekben bemutatott **DrawWuLine()** függvény fehér alapon fekete vonalak rajzolására készült, az egyes pontok kirajzoláshoz a súlyfaktorból a színt így állítottuk elő:

```
void DrawPixel(uint16_t x, uint16_t y, uint16_t color) {
  uint16_t cc =  (color & 0x1F);
    uint16_t mycolor = (cc << 11) | (cc << 6 ) | cc;
  ST7735_DrawPixel(x, y, mycolor);
}
```

- Valódi színes vonalak és háttér esetén a súlyfaktorral a papírszín és a tintaszín közötti átmenetet kell képezni erre egy példa a **wxWidgets** kódrészlete:

```
wxColour GetColorAverage( wxColour color1, wxColour color2, float weight) {
  unsigned char r, g, b;
  r = (unsigned char)(color1.Red()*weight + color2.Red()*(1 - weight));
  g = (unsigned char)(color1.Green()*weight + color2.Green()*(1 - weight));
  b = (unsigned char)(color1.Blue()*weight + color2.Blue()*(1 - weight));
    return wxColour(r, g, b);
}
```



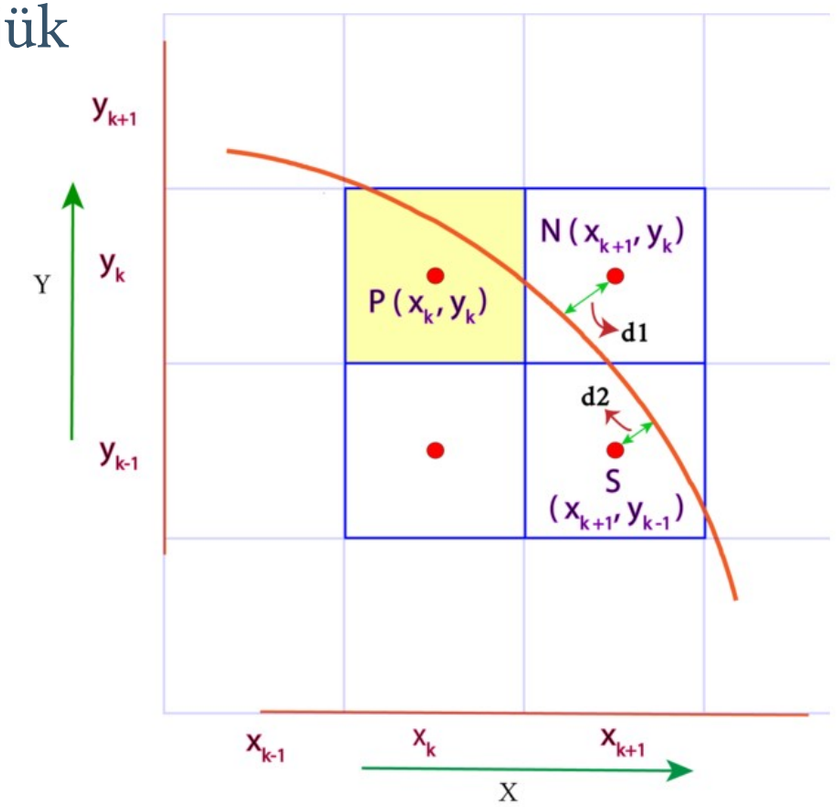
ST7735_4/main.c

- A **testlines()** függvény egyik felében **ST7735_DrawLine()** helyett **DrawWuLine()** áll, amely fehér háttérre fekete színnel rajzol ezért ehhez igazítottuk a színezést
- A főprogram elején inicializálni kell a **PB0** (a DC bemenet vezérlése) és a **PB1** (RESET láb vezérlése) kimeneteket
- Az **SPI1** csatorna és a kijelző inicializálása után a végtelen ciklusban végezzük a rajzolást

```
#include "st7735.h"

void testlines(uint16_t color) {
  for (int16_t x=0; x < ST7735_WIDTH; x+=12) {
    ST7735_DrawLine(0, 0, x, ST7735_HEIGHT-1, color);
    DelayMs(50);
  }
  for (int16_t y=0; y < ST7735_HEIGHT ; y+=12) {
    DrawWuLine(0, 0, ST7735_WIDTH-1,y);
    DelayMs(50);
  }
}

int main(void) {
  /* PB0-PB1 konfigurálása digitális kimenetként */
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN; // GPIOB órajel engedélyezés
  GPIOB->CRL &= ~0x000000FF;          // CNF és Mode bitek törlése
  GPIOB->CRL |=  0x00000033;          // PB0 és PB1 Pushpull kimenet
  SPI1_init();                        // Az SPI1 modul konfigurálása
  ST7735_Init();                      // A kijelző inicializálása
  while(1) {
    ST7735_FillScreen(ST7735_WHITE);
    testlines(ST7735_BLACK);
    DelayMs(5000);
  }
}
```

Kör és körlap rajzolása (Bresenham algoritmus)

- Korábban a **keil19_11 (2020. február 13.)** előadásban már érintettük ezt a témát
- Az $\mathbf{x^2 + y^2 = r^2}$ egyenlettel leírható kört pixelekkel közelítjük
- A Bresenham algoritmusnál azt kell eldönteni, hogy a $\mathbf{P}(x_k, y_k)$ pont után az $\boldsymbol{N}(x_k+1, y_k)$, vagy az $\boldsymbol{S}(x_k+1, y_k-1)$ pont van-e közelebb a körhöz

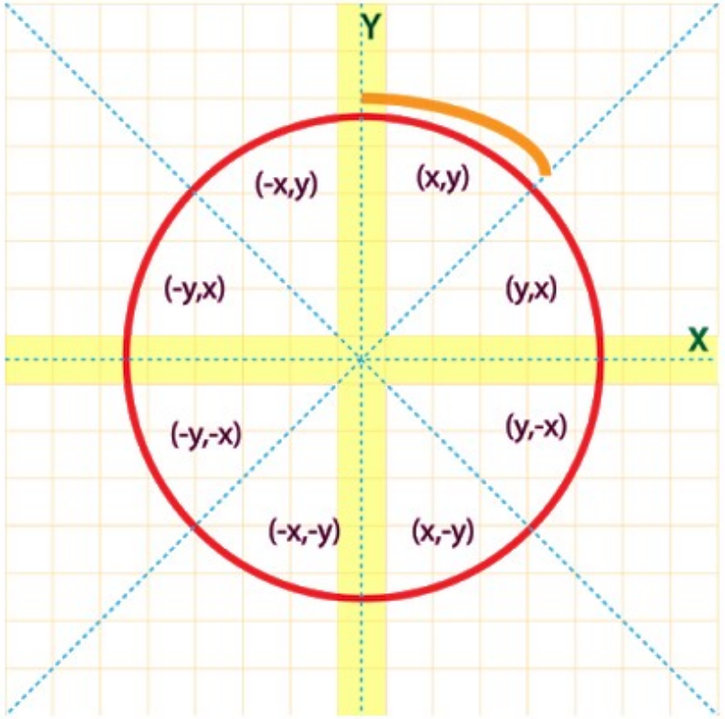


- Az $\boldsymbol{N}$ pont kívül esik a körön, ezért $\mathbf{F(N) = (x_k+1)^2 + y_k^2 - r^2}$ értéke pozitív
- Az $\boldsymbol{S}$ pont belül esik a körön, ezért $\mathbf{F(S) = (x_k+1)^2 + (y_k-1)^2 - r^2}$ értéke negatív
- **Apró trükk:** célszerűbb a $\mathbf{D_k = F(\boldsymbol{N})+F(\boldsymbol{S})}$ kifejezés értékét vizsgálni és lépésről lépésre frissíteni, így $\mathbf{D_k < 0}$ esetén az $\boldsymbol{N}$ pontot, $\mathbf{D_k > 0}$ esetén pedig az $\boldsymbol{S}$ pontot kell választanunk

Bővebben lásd: Computer Graphics Tutorials – Bresenham's Circle Drawing Derivation

Kör és körlap rajzolása (Bresenham algoritmus)

```
void ST7735_DrawCircle(uint16_t x0, uint16_t y0, uint16_t r, uint16_t c) {
    int16_t f = 1 - r;
    int16_t ddF_x = 1;
    int16_t ddF_y = -2 * r;
    int16_t x = 0;
    int16_t y = r;
    ST7735_DrawPixel(x0, y0 + r, c);
    ST7735_DrawPixel(x0, y0 - r, c);
    ST7735_DrawPixel(x0 + r, y0, c);
    ST7735_DrawPixel(x0 - r, y0, c);
    while (x < y) {
        if (f >= 0) {y--; ddF_y += 2; f += ddF_y; }
        x++; ddF_x += 2; f += ddF_x;
        ST7735_DrawPixel(x0 + x, y0 + y, c);
        ST7735_DrawPixel(x0 - x, y0 + y, c);
        ST7735_DrawPixel(x0 + x, y0 - y, c);
        ST7735_DrawPixel(x0 - x, y0 - y, c);
        ST7735_DrawPixel(x0 + y, y0 + x, c);
        ST7735_DrawPixel(x0 - y, y0 + x, c);
        ST7735_DrawPixel(x0 + y, y0 - x, c);
        ST7735_DrawPixel(x0 - y, y0 - x, c);
    }
}
```

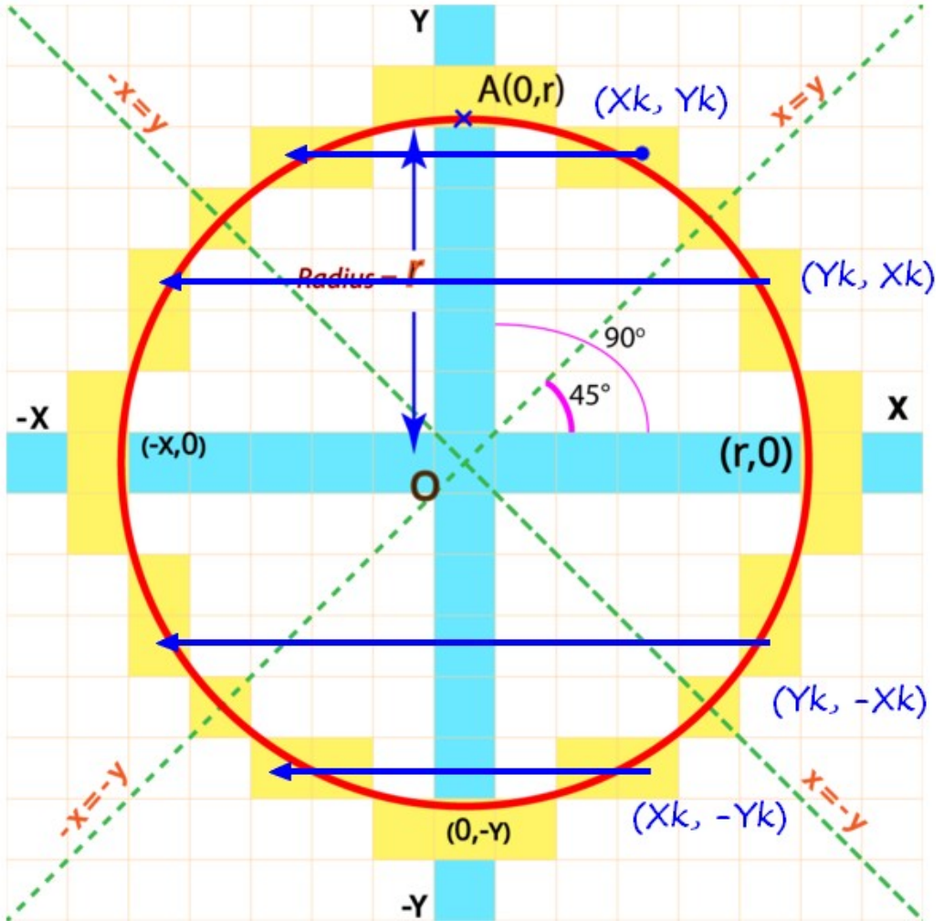


- A szimmetria miatt elég egy síknyolcadot kiszámolni, a többi pont koordinátái tükrözésekkel előállnak

Ábra: Computer Graphics Tutorials – Bresenham's Circle Drawing Derivation

Kör és körlap rajzolása (Bresenham algoritmus)

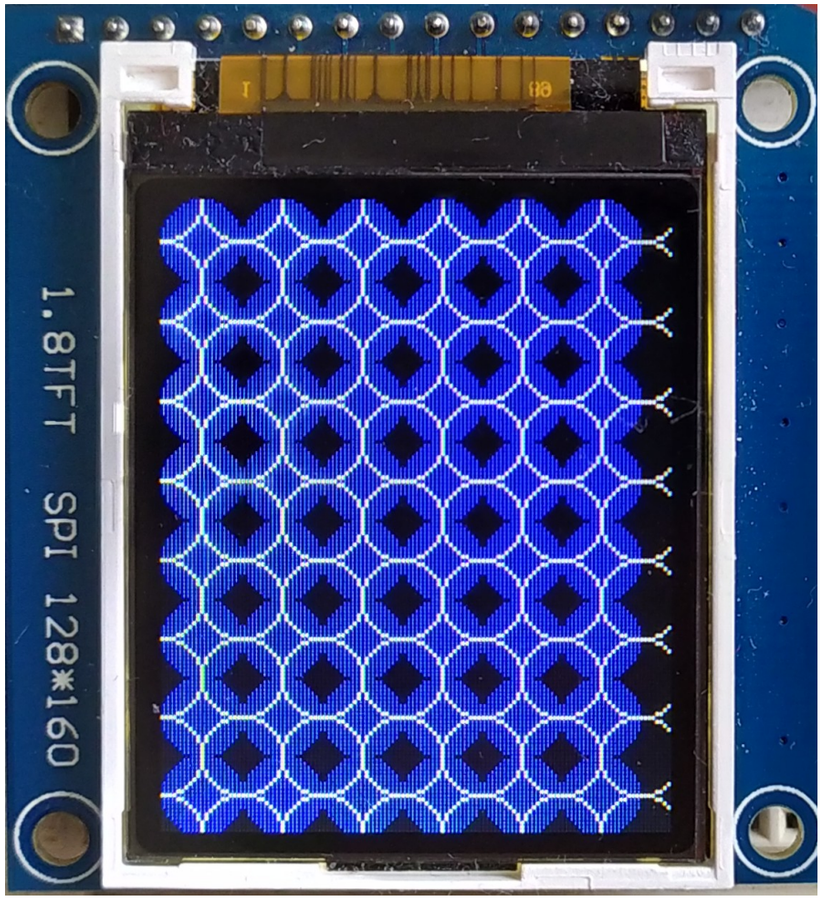
```
void ST7735_DrawFilledCircle(int16_t x0, int16_t y0, int16_t r, uint16_t c) {
  int16_t f = 1 - r;
  int16_t ddF_x = 1;
  int16_t ddF_y = -2 * r;
  int16_t x = 0;
  int16_t y = r;
  ST7735_DrawPixel(x0, y0 + r, c);
  ST7735_DrawPixel(x0, y0 - r, c);
  ST7735_DrawPixel(x0 + r, y0, c);
  ST7735_DrawPixel(x0 - r, y0, c);
  ST7735_DrawLine(x0 - r, y0, x0 + r, y0, c);
  while (x < y) {
    if (f >= 0) {y--; ddF_y += 2; f += ddF_y; }
    X++; ddF_x += 2; f += ddF_x;
    ST7735_DrawLine(x0 - x, y0 + y, x0 + x, y0 + y, c);
    ST7735_DrawLine(x0 + x, y0 - y, x0 - x, y0 - y, c);
    ST7735_DrawLine(x0 + y, y0 + x, x0 - y, y0 + x, c);
    ST7735_DrawLine(x0 + y, y0 - x, x0 - y, y0 - x, c);
  }
}
```



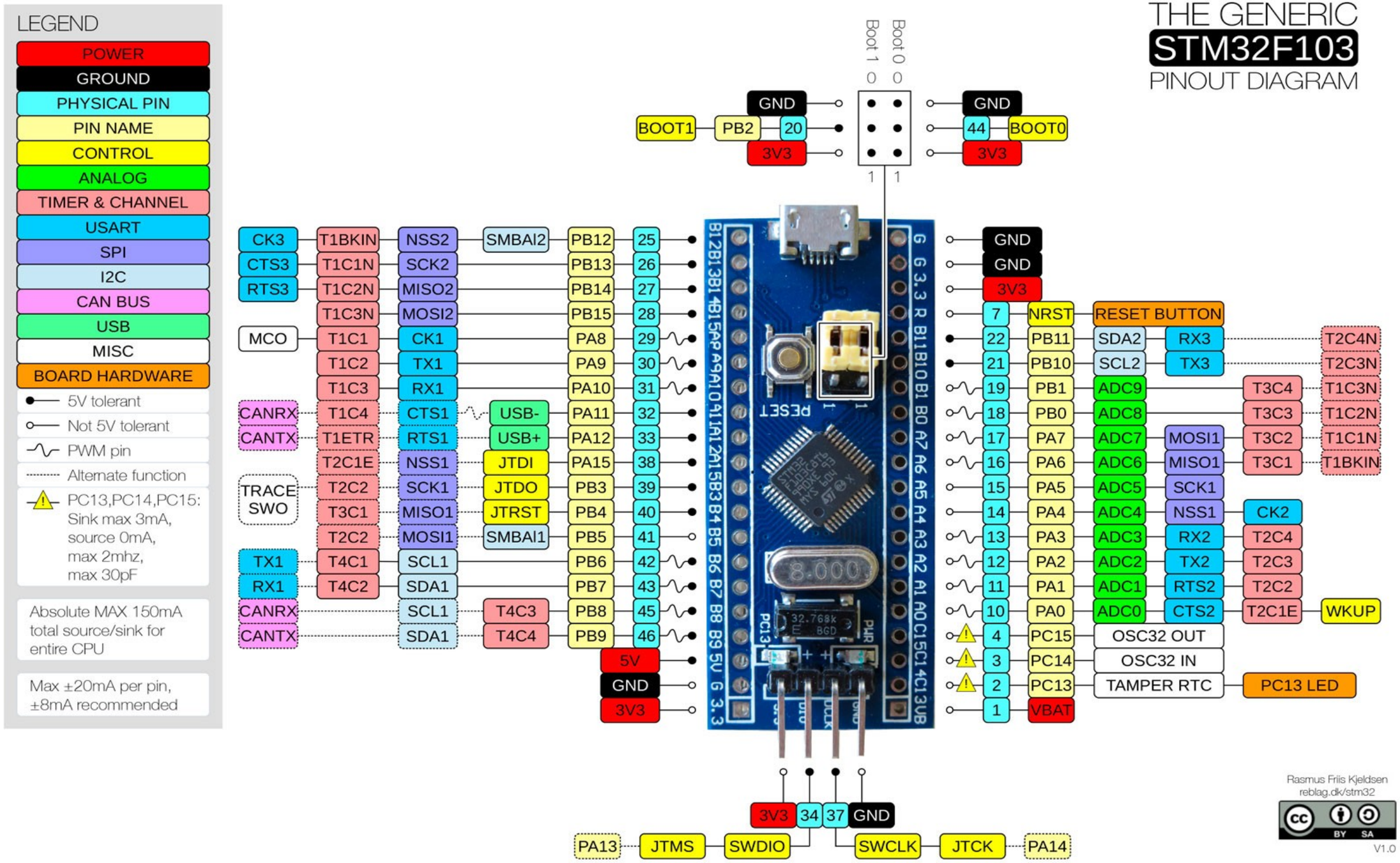
Ábra: Computer Graphics Tutorials – Bresenham's Circle Drawing Derivation

ST7735_5/main.c

- Ezzel a programmal reprodukáljuk az Adafruit_ST7735 library minta-programjának a körrajzoló részét




```
#include "st7735.h"
void testfillcircles(uint8_t radius, uint16_t color) {
  for (int16_t x=radius; x < ST7735_WIDTH; x+=radius*2) {
    for (int16_t y=radius; y < ST7735_HEIGHT; y+=radius*2)
      ST7735_DrawFilledCircle(x, y, radius, color);
  }
}
void testdrawcircles(uint8_t radius, uint16_t color) {
  for (int16_t x=0; x < ST7735_WIDTH+radius; x+=radius*2) {
    for (int16_t y=0; y < ST7735_HEIGHT+radius; y+=radius*2)
      ST7735_DrawCircle(x, y, radius, color);
  }
}

int main(void) {
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN; // GPIOB órajel engedélyezés
  GPIOB->CRL &= ~0x000000FF;          // CNF és Mode bitek törlése
  GPIOB->CRL |=  0x00000033;          // PB0 és PB1 Pushpull kimenet
  SPI1_init();                        // Az SPI1 modul konfigurálása
  ST7735_Init();                      // A kijelző inicializálása
  while(1) {
    ST7735_FillScreen(ST7735_BLACK);
    testfillcircles(10,ST7735_BLUE);
    testdrawcircles(10,ST7735_WHITE);
    DelayMs(5000);
  }
}
```


LEGEND
POWER
GROUND
PHYSICAL PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V tolerant
Not 5V tolerant
PWM pin
Alternate function
PC13,PC14,PC15: Sink max 3mA, source 0mA, max 2mhz, max 30pF
Absolute MAX 150mA total source/sink for entire CPU
Max ±20mA per pin, ±8mA recommended
THE GENERIC
STM32F103
PINOUT DIAGRAM
Boot 1
Boot 0
GND
BOOT1
PB2
20
3V3
GND
44
BOOT0
3V3
CK3
CTS3
RTS3
MCO
CANRX
CANTX
TRACE SWO
TX1
RX1
CANRX
CANTX
T1BKIN
T1C1N
T1C2N
T1C3N
T1C1
T1C2
T1C3
T1C4
T1ETR
T2C1E
T2C2
T3C1
T2C2
T4C1
T4C2
NSS2
SCK2
MISO2
MOSI2
CK1
TX1
RX1
CTS1
RTS1
NSS1
SCK1
MISO1
MOSI1
SCL1
SDA1
SCL1
SDA1
SMBAI2
USB-
USB+
JTDI
JTDO
JTRST
SMBAI1
T4C3
T4C4
PB12
PB13
PB14
PB15
PA8
PA9
PA10
PA11
PA12
PA15
PB3
PB4
PB5
PB6
PB7
PB8
PB9
25
26
27
28
29
30
31
32
33
38
39
40
41
42
43
45
46
5V
GND
3V3
GND
GND
3V3
7
22
21
19
18
17
16
15
14
13
12
11
10
4
3
2
1
NRST
PB11
PB10
PB1
PB0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC15
PC14
PC13
VBAT
RESET BUTTON
SDA2
SCL2
ADC9
ADC8
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
OSC32 OUT
OSC32 IN
TAMPER RTC
RX3
TX3
MOSI1
MISO1
SCK1
NSS1
RX2
TX2
RTS2
CTS2
T3C4
T3C3
T3C2
T3C1
CK2
T2C4
T2C3
T2C2
T2C1E
PC13 LED
T2C4N
T2C3N
T1C3N
T1C2N
T1C1N
T1BKIN
WKUP
3V3
34
37
GND
PA13
JTMS
SWDIO
SWCLK
JTCK
PA14
Rasmus Friis Kjeldsen
reblag.dk/stm32
V1.0