**Listing 1. Wysłanie komendy sterującej do sterownika**

//wysłanie komendy do sterownika przez SPI

ssp\_err\_t **WriteCmd**(uint8\_t cmd){

 ssp\_err\_t err;

 uint8\_t buffer[2];

 data\_ready=0;

 buffer[0]=cmd;

 g\_ioport.p\_api->pinWrite (DC,*IOPORT\_LEVEL\_LOW*);//DC=0;

 g\_ioport.p\_api->pinWrite (CS,*IOPORT\_LEVEL\_LOW*);//CS=0;

 err=R\_SCI\_SPI\_Write(g\_spi0.p\_ctrl,buffer, 1, *SPI\_BIT\_WIDTH\_8\_BITS*);

 **if**(err!=*SSP\_SUCCESS*)

 **while**(1);

 **while**(data\_ready==0);

 g\_ioport.p\_api->pinWrite (CS,*IOPORT\_LEVEL\_HIGH*);//CS=1;

 g\_ioport.p\_api->pinWrite (DC,*IOPORT\_LEVEL\_HIGH*);//DC=1;

 **return**(err);

}

**Listing 2. Wysłanie danej do sterownika przez SPI**

//wysłanie danej do sterownika przez SPI

ssp\_err\_t **WriteData**(uint8\_t data){

 ssp\_err\_t err;

 uint8\_t buffer[2];

 data\_ready=0;

 buffer[0]=data;

 g\_ioport.p\_api->pinWrite (DC,*IOPORT\_LEVEL\_HIGH*);//DC=1;

 g\_ioport.p\_api->pinWrite (CS,*IOPORT\_LEVEL\_LOW*);//CS=0;

 err=R\_SCI\_SPI\_Write(g\_spi0.p\_ctrl,buffer, 1, *SPI\_BIT\_WIDTH\_8\_BITS*);

 **if**(err!=*SSP\_SUCCESS*)

 **while**(1);

 **while**(data\_ready==0);

 g\_ioport.p\_api->pinWrite (CS,*IOPORT\_LEVEL\_HIGH*);//CS=1;

 g\_ioport.p\_api->pinWrite (DC,*IOPORT\_LEVEL\_HIGH*);//DC=1;

 **return**(err);

}

**Listing 3. Przykład funkcji callback**

//prototyp funkcji calback dla kontroli transmisji SPI

**void** **spi\_callback**(spi\_callback\_args\_t \* p\_args)

{

 **if**(*SPI\_EVENT\_TRANSFER\_COMPLETE* == p\_args->event)

 {

 data\_ready=1;

 }

}

**Listing 4. Inicjalizacja sterownika SSD1306**

//inicjalizacja sterownika wyświeltacza

ssp\_err\_t **InitOled**(**void**)

{

 ssp\_err\_t err;

 uint16\_t i;

 err=R\_SCI\_SPI\_Open(g\_spi0.p\_ctrl, g\_spi0.p\_cfg);

 g\_ioport.p\_api->pinWrite (RES,*IOPORT\_LEVEL\_LOW*);//RES=0;

 **for**(i=0;i<0xffff;i++)

 i=i;

 g\_ioport.p\_api->pinWrite (RES,*IOPORT\_LEVEL\_HIGH*);//RES=1;

 **for**(i=0;i<28;i++)

 {err=WriteCmd(Buffer\_Init[i]);

 **if**(err!=*SSP\_SUCCESS*)

 **while**(1);

 }

 DisplayCls(0);

 **return**(err);

}

**Listing 5. Zerowanie wyświetlacza**

//zerowanie wyświetlacza

**void** **DisplayCls**(uint8\_t fill)

{

 uint8\_t i, j;

 **for** (i = 0; i < 8; i ++) {

 **for** (j = 0; j < 128; j ++) {

 DispBuff[j][i] = fill;

 }

 }

 RefreshRAM();//zawartosc bufora do RAM obrazu

}

**Listing 6. Zapisanie zawartości bufora DispBuff do pamięci wyświetlacza**

**void** **RefreshRAM**(**void**)

{

 uint8\_t i, j ;

 **for** (i = 0; i < 8; i ++) {

 WriteCmd(0xb0+i);

 SetColStart();

 **for** (j = 0; j < 128; j ++) {

 WriteData(DispBuff[j][i]);

 }

 }

}

**Listing 7. „Rysowanie” punktu w pamięci obrazu wyświetlacza**

//"rysowanie” punktu w buforze RAM mikrokontrolera Hosta

**void** **DrawPoint**(uint8\_t x ,uint8\_t y, uint8\_t p)

{

 uint8\_t chPos, chBx, chTemp = 0;

 **if** (x > 127 || y > 63) {

 **return**;

 }

 chPos = 7 - y / 8;

 chBx = y % 8;

 chTemp = 1 << (7 - chBx);

 **if** (p) {

 DispBuff[x][chPos] |= chTemp;

 } **else** {

 DispBuff[x][chPos] &= ~chTemp;

 }

}

**Listing 8. Wyświetlenie jednego znaku**

//wyświetlenie jednego znaku

//argumenty:

//x,y - współrzędne na ekranie

//Chr - kod ASCII znaku

//size - rozmiar 12, lub 16

//mode=1 znak wyświetlany normalnie, mode=0 znak wyświetlany w negatywie

**void** **DisplayChar**(uint8\_t x, uint8\_t y, uint8\_t Chr, uint8\_t size, uint8\_t mode)

{

 uint8\_t i, j;

 uint8\_t chTemp, chYpos0 = y;

 Chr =Chr - ' ';

 **for** (i = 0; i < size; i ++) {

 **if** (size == 12) {

 **if** (mode) {

 chTemp = c\_chFont1206[Chr][i];

 } **else** {

 chTemp = ~c\_chFont1206[Chr][i];

 }

 } **else** {

 **if** (mode) {

 chTemp = c\_chFont1608[Chr][i];

 } **else** {

 chTemp = ~c\_chFont1608[Chr][i];

 }

 }

 **for** (j = 0; j < 8; j ++) {

 **if** (chTemp & 0x80) {

 DrawPoint(x, y, 1);

 } **else** {

 DrawPoint(x, y, 0);

 }

 chTemp <<= 1;

 y ++;

 **if** ((y - chYpos0) == size) {

 y = chYpos0;

 x ++;

 **break**;

 }

 }

 }

}

**Listing 9. Wyświetlenie łańcucha znaków**

//wyświetlenie łańcucha znaków - napisu

//argumenty

//x,y - współrzędne na ekranie

//\*txt - wskaźnik na początek bufora zwierającego łańcuch znaków ASCII do wyświetlenia

//size - wysokość znaków 12, lub 16 pikseli

//mode=1 znaki wyświetlane normalnie, mode=0 znaki wyświetlane w negatywie

**void** **DispTxt**(uint8\_t x, uint8\_t y, **const** uint8\_t \*txt, uint8\_t size, uint8\_t mode)

{

 **while** (\*txt != '\0') {

 **if** (x > (SSD1306\_WIDTH - size / 2)) {

 x = 0;

 y += size;

 **if** (y > (SSD1306\_HEIGHT - size)) {

 y = x = 0;

 DisplayCls(0x00);

 }

 }

 DisplayChar(x, y, \*txt, size, mode);

 x += size / 2;

 txt ++;

 }

}

**Listing 10. Testowanie trybu alfanumerycznego**

**void** **hal\_entry**(**void**)

{

 InitOled();

 DispTxt(0, 0, " ARROW module", 16,1);

 DispTxt(0, 20, "Renesas Synergy", 16,1);

 DispTxt(0, 40, " text mode test ", 16,1);

 RefreshRAM();

 **while**(1);

}

**Listing 11. Rysowanie bitmapy**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//rysuj bitmapę

//argumenty

//x,y współrzędne początku bitmapy

//\*bmp wskaźnik na tablicę z przekonwertowaną bitmapą

//width - szerokość bitmapy

//height - wysokość bitmapy

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**void** **DrawBmp**(uint8\_t x, uint8\_t y, **const** uint8\_t \*bmp, uint8\_t width, uint8\_t height)

{

 uint16\_t i, j;

 uint16\_t byteWidth = (width + 7) / 8;

 **for**(j = 0; j < height; j ++){

 **for**(i = 0; i < width; i ++ ) {

 **if**(\*(bmp + j \* byteWidth + i / 8) & (128 >> (i & 7))) {

 DrawPoint(x + i, y + j, 1);

 }

 }

 }

}

**Listing 12. Rysowanie bitmapy o rozdzielczości 128×64 piksele**

DrawBmp(0,0, bmparray, 128, 64);

RefreshRAM();