**Listing 1. Podstawowe funkcje narzędziowe odpowiedzialne za wysyłanie rozkazów sterujących lub danych do wyświetlacza TFT**

void writeCommand**(**uint8\_t Command**)**

**{**

RESET\_DCX**;** //Command

RESET\_WRX**;**

TFT\_DATA\_PORT **=** Command**;**

SET\_WRX**;** //TFT reads data at the rising edge

SET\_DCX**;** //DCX is 1 by default to increase DRAM data transfer

**}**

void writeData**(**uint8\_t Data**)**

**{**

RESET\_WRX**;**

TFT\_DATA\_PORT **=** Data**;**

SET\_WRX**;** //TFT reads data at the rising edge

**}**

**Listing 2. Funcja inicjalizacyjna sterownika ekranu ILI9341 wyświetlacza Rivierdi RVT28AETNWN00**

void TFTinit**(**void**)**

**{**

TFT\_DATA\_DDR **=** 0xFF**;** //Data port as output with 0x00

TFT\_CTRL\_PORT**|=** **(**1**<<**WRX\_PIN**)|(**1**<<**DCX\_PIN**)|(**1**<<**RESX\_PIN**)|(**1**<<**RDX\_PIN**);** //ChipSelect (CSX) is 0 by default

TFT\_CTRL\_DDR **|=** **(**1**<<**WRX\_PIN**)|(**1**<<**CSX\_PIN**)|(**1**<<**DCX\_PIN**)|(**1**<<**RESX\_PIN**)|(**1**<<**RDX\_PIN**);** //All control ports are outputs

\_delay\_ms**(**5**);**

//Hardware reset

RESET\_RESX**;**

\_delay\_ms**(**5**);**

SET\_RESX**;**

\_delay\_ms**(**50**);**

//Software reset and configuration

writeCommand**(**CMD\_SOFTWARE\_RESET**);**

\_delay\_ms**(**5**);**

writeCommand**(**CMD\_DISPLAY\_OFF**);**

writeCommand**(**CMD\_PWR\_CTRL\_B**);**

writeData**(**0x00**);** //Default values

writeData**(**0x83**);**

writeData**(**0x30**);**

writeCommand**(**CMD\_PWR\_ON\_SEQ\_CTRL**);**

writeData**(**0x64**);** //Default values

writeData**(**0x03**);**

writeData**(**0x12**);**

writeData**(**0x81**);**

writeCommand**(**CMD\_DRIVER\_TIM\_CTRL\_A**);**

writeData**(**0x85**);** //Default values

writeData**(**0x01**);**

writeData**(**0x79**);**

writeCommand**(**CMD\_PWR\_CTRL\_A**);**

writeData**(**0x39**);** //Default value

writeData**(**0x2c**);** //Default value

writeData**(**0x00**);** //Default value

writeData**(**VCORE\_1\_60**);**

writeData**(**DDVDH\_5\_6**);**

writeCommand**(**CMD\_PUMP\_RATIO\_CTRL**);**

writeData**(**0x20**);** //Default value

writeCommand**(**CMD\_DRIVER\_TIM\_CTRL\_B**);**

writeData**(**0x00**);** //Default value

writeData**(**0x00**);** //Default value

//Power Control

writeCommand**(**CMD\_PWR\_CTRL\_1**);**

writeData**(**0x26**);** //Default value

writeCommand**(**CMD\_PWR\_CTRL\_2**);**

writeData**(**0x11**);** //Default value

//VCOM setting

writeCommand**(**CMD\_VCOM\_CTRL\_1**);**

writeData**(**0x35**);** //Default value

writeData**(**0x3e**);** //Default value

writeCommand**(**CMD\_VCOM\_CTRL\_2**);**

writeData**(**0xbe**);** //Default value

//Memory Access Control

writeCommand**(**CMD\_MEM\_ACCESS\_CTRL**);**

writeData**(**ROW\_COLUMN\_EXCHANGE\_ON**|**RGB\_ORDER\_BGR**);**

writeCommand**(**CMD\_PIXEL\_FORMAT**);**

writeData**(**RGB\_INTERFACE\_16BITS**|**MCU\_INTERFACE\_16BITS**);**

//Frame Rate

writeCommand**(**CMD\_FRAME\_RATE**);**

writeData**(**DIVISION\_RATIO\_FOSC**);**

writeData**(**FRAME\_RATE\_119KHZ**);**

//Gamma

writeCommand**(**CMD\_GAMMA3\_FUNCTION\_DISABLE**);**

writeData**(**0x08**);** //Default value

writeCommand**(**CMD\_SET\_GAMMA\_CURVE\_NR**);**

writeData**(**0x01**);** //One curve

writeCommand**(**CMD\_POSITIVE\_GAMMA\_CORR**);**

writeData**(**0x1f**);**

writeData**(**0x1a**);**

writeData**(**0x18**);**

writeData**(**0x0a**);**

writeData**(**0x0f**);**

writeData**(**0x06**);**

writeData**(**0x45**);**

writeData**(**0x87**);**

writeData**(**0x32**);**

writeData**(**0x0a**);**

writeData**(**0x07**);**

writeData**(**0x02**);**

writeData**(**0x07**);**

writeData**(**0x05**);**

writeData**(**0x00**);**

writeCommand**(**CMD\_NEGATIVE\_GAMMA\_CORR**);**

writeData**(**0x00**);**

writeData**(**0x25**);**

writeData**(**0x27**);**

writeData**(**0x05**);**

writeData**(**0x10**);**

writeData**(**0x09**);**

writeData**(**0x3a**);**

writeData**(**0x78**);**

writeData**(**0x4d**);**

writeData**(**0x05**);**

writeData**(**0x18**);**

writeData**(**0x0d**);**

writeData**(**0x38**);**

writeData**(**0x3a**);**

writeData**(**0x1f**);**

//DISPLAY RAM

writeCommand**(**CMD\_COLUMN\_ADDR\_SET**);**

writeData**(**0x00**);**

writeData**(**0x00**);**

writeData**((**TFT\_WIDTH**-**1**)>>**8**);**

writeData**((**uint8\_t**)** TFT\_WIDTH**-**1**);**

writeCommand**(**CMD\_PAGE\_ADDR\_SET**);**

writeData**(**0x00**);**

writeData**(**0x00**);**

writeData**((**TFT\_HEIGHT**-**1**)>>**8**);**

writeData**(**TFT\_HEIGHT**-**1**);**

writeCommand**(**CMD\_ENTRY\_MODE**);**

writeData**(**GON\_DTE\_NORMAL\_DISPLAY**|**LOW\_VOLTAGE\_DETECT\_DISBLE**);**

//Display

writeCommand**(**CMD\_DISPLAY\_FUNCTION\_CTRL**);**

writeData**(**INTERVAL\_SCAN**|**0b10**);**

writeData**(**LCD\_NORMALLY\_WHITE**|**GATE\_OUT\_SCAN\_DIR\_NORMAL**|**SOURCE\_OUT\_SCAN\_DIR\_NORMAL**|**GATE\_OUT\_SEQUENCE\_NORMAL**|**SCAN\_CYCLE\_INTERVAL\_5FRAME**);**

writeData**(**NUMBER\_OF\_LINES\_320**);**

writeData**(**0x00**);** //Default value

writeCommand**(**CMD\_SLEEP\_OUT**);**

\_delay\_ms**(**5**);**

writeCommand**(**CMD\_DISPLAY\_ON**);**

\_delay\_ms**(**5**);**

**}**

**Listing 3. Plik nagłówkowy sterownika ekranu ILI9341 wyświetlacza Rivierdi RVT28AETNWN00**

//PHYSICAL PARAMETERS - LANDSCAPE MODE

#define TFT\_WIDTH 320

#define TFT\_HEIGHT 240

//PORTS CONFIGURATION. We assume that RDX (read signal) is permanently pulled high

#define TFT\_DATA\_PORT PORTA

#define TFT\_DATA\_DDR DDRA

#define TFT\_CTRL\_PORT PORTB

#define TFT\_CTRL\_DDR DDRB

#define WRX\_PIN PB1 //Write signal

#define RDX\_PIN PB7 //Read signal

#define CSX\_PIN PB3 //Chip select signal

#define DCX\_PIN PB2 //Command/Data signal: 1->DRAM data, 0->Command

#define RESX\_PIN PB4 //Reset signal

//MACROS

#define SET\_WRX TFT\_CTRL\_PORT |= (1<<WRX\_PIN)

#define RESET\_WRX TFT\_CTRL\_PORT &= ~(1<<WRX\_PIN)

#define SET\_CSX TFT\_CTRL\_PORT |= (1<<CSX\_PIN)

#define RESET\_CSX TFT\_CTRL\_PORT &= ~(1<<CSX\_PIN)

#define SET\_DCX TFT\_CTRL\_PORT |= (1<<DCX\_PIN)

#define RESET\_DCX TFT\_CTRL\_PORT &= ~(1<<DCX\_PIN)

#define SET\_RESX TFT\_CTRL\_PORT |= (1<<RESX\_PIN)

#define RESET\_RESX TFT\_CTRL\_PORT &= ~(1<<RESX\_PIN)

//ILI9341 COMMANDS AND PARAMETERS

#define CMD\_SOFTWARE\_RESET 0x01 //5ms delay is needed after this command

#define CMD\_SLEEP\_OUT 0x11 //5ms delay is needed after this command

#define CMD\_SET\_GAMMA\_CURVE\_NR 0x26 //Selects gamma curve number

#define CMD\_DISPLAY\_OFF 0x28

#define CMD\_DISPLAY\_ON 0x29

#define CMD\_COLUMN\_ADDR\_SET 0x2A //Sets current column address

#define CMD\_PAGE\_ADDR\_SET 0x2B //Sets current page address

#define CMD\_MEMORY\_WRITE 0x2C //Starts frame memory writting

#define CMD\_MEM\_ACCESS\_CTRL 0x36 //Defines read/write scanning direction of frame memory

#define ROW\_ADDR\_ORDER\_NORMAL (0<<7)

#define ROW\_ADDR\_ORDER\_REVERSE (1<<7)

#define COLUMN\_ADDR\_ORDER\_NORMAL (0<<6)

#define COLUMN\_ADDR\_ORDER\_REVERSE (1<<6)

#define ROW\_COLUMN\_EXCHANGE\_OFF (0<<5)

#define ROW\_COLUMN\_EXCHANGE\_ON (1<<5)

#define VERTICAL\_REFRESH\_ORDER\_NORMAL (0<<4)

#define VERTICAL\_REFRESH\_ORDER\_REVERSE (1<<4)

#define RGB\_ORDER\_RGB (0<<3)

#define RGB\_ORDER\_BGR (1<<3)

#define HORIZONTAL\_REFRESH\_ORDER\_NORMAL (0<<2)

#define HORIZONTAL\_REFRESH\_ORDER\_REVERSE (1<<2)

#define CMD\_PIXEL\_FORMAT 0x3A //Sets pixel data format

#define RGB\_INTERFACE\_16BITS (0b101<<4)

#define RGB\_INTERFACE\_18BITS (0b110<<4)

#define MCU\_INTERFACE\_16BITS (0b101)

#define MCU\_INTERFACE\_18BITS (0b110)

#define CMD\_FRAME\_RATE 0xB1 //Sets the division ratio for internal clocks

#define DIVISION\_RATIO\_FOSC 0x00 //1st parameter

#define DIVISION\_RATIO\_FOSC\_2 0x01

#define DIVISION\_RATIO\_FOSC\_4 0x02

#define DIVISION\_RATIO\_FOSC\_8 0x03

#define FRAME\_RATE\_119KHZ 0b10000 //2nd parameter

#define FRAME\_RATE\_112KHZ 0b10001

#define FRAME\_RATE\_106KHZ 0b10010

#define FRAME\_RATE\_100KHZ 0b10011

#define FRAME\_RATE\_95KHZ 0b10100

#define FRAME\_RATE\_90KHZ 0b10101

#define FRAME\_RATE\_85KHZ 0b10110

#define FRAME\_RATE\_83KHZ 0b10111

#define CMD\_DISPLAY\_FUNCTION\_CTRL 0xB6

#define NORMAL\_SCAN (0b00<<2) //1st parameter

#define INTERVAL\_SCAN (0b10<<2)

#define LCD\_NORMALLY\_BLACK (0<<7) //2nd parameter

#define LCD\_NORMALLY\_WHITE (1<<7)

#define GATE\_OUT\_SCAN\_DIR\_NORMAL (0<<6)

#define GATE\_OUT\_SCAN\_DIR\_REVERSE (1<<6)

#define SOURCE\_OUT\_SCAN\_DIR\_NORMAL (0<<5)

#define SOURCE\_OUT\_SCAN\_DIR\_REVERSE (1<<5)

#define GATE\_OUT\_SEQUENCE\_NORMAL (0<<4)

#define GATE\_OUT\_SEQUENCE\_REVERSE (1<<4)

#define SCAN\_CYCLE\_INTERVAL\_1FRAME (0b0000)

#define SCAN\_CYCLE\_INTERVAL\_3FRAME (0b0001)

#define SCAN\_CYCLE\_INTERVAL\_5FRAME (0b0010)

#define SCAN\_CYCLE\_INTERVAL\_7FRAME (0b0011)

#define SCAN\_CYCLE\_INTERVAL\_9FRAME (0b0100)

#define SCAN\_CYCLE\_INTERVAL\_11FRAME (0b0101)

#define SCAN\_CYCLE\_INTERVAL\_13FRAME (0b0110)

#define SCAN\_CYCLE\_INTERVAL\_15FRAME (0b0111)

#define NUMBER\_OF\_LINES\_320 0x27 //3rd parameter

#define NUMBER\_OF\_LINES\_240 0x1D

#define CMD\_ENTRY\_MODE 0xB7

#define LOW\_VOLTAGE\_DETECT\_ENABLE (0x00)

#define LOW\_VOLTAGE\_DETECT\_DISBLE (0x01)

#define GON\_DTE\_VGH (0x00<<1)

#define GON\_DTE\_VGH2 (0x01<<1)

#define GON\_DTE\_VGL (0x02<<1)

#define GON\_DTE\_NORMAL\_DISPLAY (0x03<<1)

#define CMD\_PWR\_CTRL\_1 0xC0 //Sets the GVDD level, which is a reference level for the VCOM

#define CMD\_PWR\_CTRL\_2 0xC1 //Sets the factor used in the step-up circuits.

#define CMD\_VCOM\_CTRL\_1 0xC5 //Sets the VCOM voltage.

#define CMD\_VCOM\_CTRL\_2 0xC7 //Set the VCOM offset voltage

#define CMD\_PWR\_CTRL\_A 0xCB

#define VCORE\_1\_55 0x30

#define VCORE\_1\_40 0x31

#define VCORE\_1\_50 0x32

#define VCORE\_1\_65 0x33

#define VCORE\_1\_60 0x34

#define VCORE\_1\_70 0x35

#define DDVDH\_5\_8 0x00

#define DDVDH\_5\_7 0x01

#define DDVDH\_5\_6 0x02

#define DDVDH\_5\_5 0x03

#define DDVDH\_5\_4 0x04

#define DDVDH\_5\_3 0x05

#define DDVDH\_5\_2 0x06

#define CMD\_PWR\_CTRL\_B 0xCF

#define CMD\_POSITIVE\_GAMMA\_CORR 0xE0 //Set the gray scale voltage to adjust the gamma of the TFT

#define CMD\_NEGATIVE\_GAMMA\_CORR 0xE1 //Set the gray scale voltage to adjust the gamma of the TFT

#define CMD\_DRIVER\_TIM\_CTRL\_A 0xE8 //EQ timing for Internal clock

#define CMD\_DRIVER\_TIM\_CTRL\_B 0xEA

#define CMD\_PWR\_ON\_SEQ\_CTRL 0xED

#define CMD\_GAMMA3\_FUNCTION\_DISABLE 0xF2

#define CMD\_PUMP\_RATIO\_CTRL 0xF7

//Auxiliary macros - used in selected functions

#define SOLID\_TEXT 0

#define TRANSPARENT\_TEXT 1

**Listing 4. Funkcja narzędziowa odpowiedzialna za ustawienie aktywnego obszaru ekranu, w ramach którego przeprowadzany jest zapis do pamięci ekranu sterownika ILI9341**

void TFTsetActiveWindow**(**uint16\_t X1**,** uint8\_t Y1**,** uint16\_t X2**,** uint8\_t Y2**)**

**{**

writeCommand**(**CMD\_COLUMN\_ADDR\_SET**);**

writeData**(**X1 **>>** 8**);**

writeData**(**X1 **&** 0xFF**);**

writeData**(**X2 **>>** 8**);**

writeData**(**X2 **&** 0xFF**);**

writeCommand**(**CMD\_PAGE\_ADDR\_SET**);**

writeData**(**Y1 **>>** 8**);**

writeData**(**Y1**);**

writeData**(**Y2 **>>** 8**);**

writeData**(**Y2**);**

**}**

**Listing 5. Funkcje umożliwiające wyświetlanie wypełnionego i „pustego” prostokąta na ekranie wyświetlacza TFT**

void TFTdrawFilledBox**(**uint16\_t X1**,** uint8\_t Y1**,** uint16\_t X2**,** uint8\_t Y2**)**

**{**

uint32\_t pixelsToSend **=** **(**X2**-**X1**+**1**)\*(**Y2**-**Y1**+**1**);** //We calculate how many pixels we need to send (2 bytes/pixel)

//We define display active area to simplify writing

TFTsetActiveWindow**(**X1**,** Y1**,** X2**,** Y2**);**

//We start memory writing

writeCommand**(**CMD\_MEMORY\_WRITE**);**

**while(**pixelsToSend **--)** **{**writeData**(**Colour**>>**8**);** writeData**(**Colour**&**0xFF**);}** //(2 bytes/pixel)

**}**

void TFTdrawRectangle**(**uint16\_t X1**,** uint8\_t Y1**,** uint16\_t X2**,** uint8\_t Y2**)**

**{**

TFTdrawFilledBox**(**X1 **,** Y1 **,** X2 **,** Y1**);**

TFTdrawFilledBox**(**X1 **,** Y2 **,** X2 **,** Y2**);**

TFTdrawFilledBox**(**X1 **,** Y1 **,** X1 **,** Y2**);**

TFTdrawFilledBox**(**X2 **,** Y1 **,** X2 **,** Y2**);**

**}**

**Listing 6. Funkcja odpowiedzialna za wyświetlanie obrazków na ekranie wyświetlacza TFT**

void TFTdrawPicture**(**uint16\_t X1**,** uint8\_t Y1**,** const uint16\_t **\***Picture**)**

**{**

register uint16\_t pixelData**,** pixelsToSend**;**

register uint8\_t Width**,** Height**;**

//We read the first word that holds the picture width and height (MSB and LSB)

pixelData **=** pgm\_read\_word**(**Picture**++);**

//We calculate the picture width and height

Width **=** pixelData **>>** 8**;** Height **=** pixelData **&** 0xFF**;**

//We calculate how many pixels we need to send

pixelsToSend **=** Width **\*** Height**;**

//We define display active area to simplify writing

TFTsetActiveWindow**(**X1**,** Y1**,** X1**+**Width**-**1**,** Y1**+**Height**-**1**);**

//We start memory writing

writeCommand**(**CMD\_MEMORY\_WRITE**);**

**while(**pixelsToSend**--)**

**{**

pixelData **=** pgm\_read\_word**(**Picture**++);**

writeData**(**pixelData **>>** 8**);** writeData**(**pixelData **&** 0xFF**);**

**}**

**}**

**Listing 7. Funkcja odpowiedzialna za wyświetlanie skompresowanych obrazków na ekranie wyświetlacza TFT**

void TFTdrawCompressedPicture**(**uint16\_t X1**,** uint8\_t Y1**,** const uint16\_t **\***Picture**)**

**{**

register uint16\_t pixelsToSend**,** pixelA**,** pixelB**;**

register uint8\_t Width**,** Height**;**

//We read the first word that holds the picture width and height (MSB and LSB)

pixelA **=** pgm\_read\_word**(**Picture**++);**

//We calculate the picture width and height

Width **=** pixelA **>>** 8**;** Height **=** pixelA **&** 0xFF**;**

//We calculate how many pixels we need to send

pixelsToSend **=** Width **\*** Height**;**

//We define display active area to simplify writing

TFTsetActiveWindow**(**X1**,** Y1**,** X1**+**Width**-**1**,** Y1**+**Height**-**1**);**

//We start memory writing

writeCommand**(**CMD\_MEMORY\_WRITE**);**

**while(**pixelsToSend**)**

**{**

//We read pixel n and n+1

pixelA **=** pgm\_read\_word**(**Picture**++);**

pixelB **=** pgm\_read\_word**(**Picture**);**

//If the pixel n is different than the pixel n+1 or if it is the last

//pixel, we send it to the TFT

**if(**pixelA **!=** pixelB **||** pixelsToSend **==** 1**)**

**{**

writeData**(**pixelA **>>** 8**);** writeData**(**pixelA **&** 0xFF**);**

pixelsToSend**--;**

**}**

**else**

**{**

//Otherwise we read how many the same pixels we need to send (third word)

pixelB **=** pgm\_read\_word**(++**Picture**);**

pixelsToSend **-=** pixelB**;**

Picture**++;**

//We sent them to the TFT

**while(**pixelB**--)** **{**writeData**(**pixelA **>>** 8**);** writeData**(**pixelA **&** 0xFF**);}**

**}**

**}**

**}**

**Listing 8. Definicja nowego typu danych odpowiedzialnego za przechowywanie parametrów bieżącej czcionki ekranowej**

//Deklaracja struktury przechowującej parametry bieżącej czcionki ekranowej

**typedef** struct

**{**

uint8\_t Width**;** //Current font widht (px)

uint8\_t Height**;** //Current font height (px)

uint8\_t Interspace**;** //Font interspace (px)

uint8\_t BytesPerChar**;** //Bytes per char definition

uint8\_t FirstCharCode**;** //First char ASCII code

const uint8\_t **\***Bitmap**;** //Pointer to the font table

**}** fontDescription**;**

**Listing 9. Funkcja odpowiedzialna za ustawienie bieżącej czcionki ekranowej**

void OLEDsetFont**(**constfontDescription **\***Font**)**

**{**

CurrentFont**.**Width **=** pgm\_read\_byte**(&**Font**->**Width**);** //Szerokość czcionki

CurrentFont**.**Height **=** pgm\_read\_byte**(&**Font**->**Height**);** //Wysokość czcionki

CurrentFont**.**Interspace **=** pgm\_read\_byte**(&**Font**->**Interspace**);** //Odstęp

CurrentFont**.**BytesPerChar **=** pgm\_read\_byte**(&**Font**->**BytesPerChar**);** //Liczba bajtów na definicję pojedyńczego znaku

CurrentFont**.**FirstCharCode **=** pgm\_read\_byte**(&**Font**->**FirstCharCode**);** //Kod ASCII definicji pierwszego znaku

CurrentFont**.**Bitmap **=** **(**uint8\_t**\*)**pgm\_read\_word**(&**Font**->**Bitmap**);** //Wskaźnik do tablicy wzorców tej czcionki

**}**

**Listing 10. Funkcja odpowiedzialna za rysowanie znaków, przy użyciu bieżącej czcionki ekranowej**

void TFTdrawChar**(**uint16\_t X1**,** uint8\_t Y1**,** char Character**,** const uint8\_t Transparency**)**

**{**

register uint8\_t widthIndex**,** heightIndex**,** readByte**,** pixelsNr**,** i**;**

const uint8\_t **\***dataPointer**;**

**if(**Transparency **!=** TRANSPARENT\_TEXT**)**

**{**

//We define display active area to simplify writing

TFTsetActiveWindow**(**X1**,** Y1**,** X1**+**CurrentFont**.**Width**-**1**,** Y1**+**CurrentFont**.**Height**-**1**);**

//We start memory writing

writeCommand**(**CMD\_MEMORY\_WRITE**);**

**}**

//Now we calculate start address of the current character definition

dataPointer **=** **&**CurrentFont**.**Bitmap**[(**CurrentFont**.**BytesPerChar**\*(**Character**-**CurrentFont**.**FirstCharCode**))];**

**for(**heightIndex **=** 0**;** heightIndex **<** CurrentFont**.**Height**;** heightIndex**++)**

**{**

**for(**widthIndex **=** 0**;** widthIndex **<** CurrentFont**.**Width**;** widthIndex **+=** 8**)**

**{**

//We read character definition byte by byte

readByte **=** pgm\_read\_byte**(**dataPointer**++);**

//For fonts which width is not a multiple of 8 we need to calculate

//useful number of pixels to be sent

pixelsNr **=** widthIndex**+**8 **<=** CurrentFont**.**Width **?** 8 **:** CurrentFont**.**Width **-** widthIndex**;**

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

**{**

//We check if the text background is transparent

**if(**Transparency **==** TRANSPARENT\_TEXT**)**

**{**

//We check the pixel presence

**if(**readByte **&** 0x80**)**

**{**

//We define display active area for one active pixel to simplify writing

TFTsetActiveWindow**(**X1**+**widthIndex**+**i**,** Y1**+**heightIndex**,** X1**+**widthIndex**+**i**,** Y1**+**heightIndex**);**

writeCommand**(**CMD\_MEMORY\_WRITE**);**

writeData**(**Colour **>>** 8**);** writeData**(**Colour **&** 0xFF**);**

**}**

**}**

**else**

**{**

//Pixel color depends on the pixel presence

**if(**readByte **&** 0x80**)** **{**writeData**(**Colour **>>** 8**);** writeData**(**Colour **&** 0xFF**);}**

**else** **{**writeData**(**Background **>>** 8**);** writeData**(**Background **&** 0xFF**);}**

**}**

readByte**<<=**1**;**

**}**

**}**

**}**

**}**

**Listing 11. Funkcje umożliwiające wyświetlenie ciągu znaków z pamięci RAM, jak i pamięci programu (Flash)**

void TFTdrawString**(**uint16\_t X1**,** uint8\_t Y1**,** char **\***String**,** const uint8\_t Transparency**)**

**{**

**while(\***String**)**

**{**

TFTdrawChar**(**X1**,** Y1**,** **\***String**++,** Transparency**);**

X1 **+=** CurrentFont**.**Width **+** CurrentFont**.**Interspace**;**

**}**

**}**

void TFTdrawString\_P**(**uint16\_t X1**,** uint8\_t Y1**,** const char **\***String**,** const uint8\_t Transparency**)**

**{**

register char Character**;**

**while((**Character **=** pgm\_read\_byte**(**String**++)))**

**{**

TFTdrawChar**(**X1**,** Y1**,** Character**,** Transparency**);**

X1 **+=** CurrentFont**.**Width **+** CurrentFont**.**Interspace**;**

**}**

**}**

**Listing 12. Sposób obliczania zmiennych przez oprogramowanie**

spentFuelPer1s **=** **((**1UL**\***Config**.**Cylinders**\***injectionTime**\***Config**.**CcPerMin**)/**2880UL**);**

Accu**.**spentFuel **+=** spentFuelPer1s**;**

**if(**Accu**.**remainingFuel **>=** spentFuelPer1s**)** Accu**.**remainingFuel **-=** spentFuelPer1s**;**

Accu**.**Distance **+=** **((**1UL**\***WEGpulses**\***Config**.**Wheel**)** **/** **(**100UL**\***Config**.**PulsPerRot**));**

Speed **=** **((**36UL**\***WEGpulses**\***Config**.**Wheel**)** **/** **(**1000UL**\***Config**.**PulsPerRot**));**

**if(**Speed**<=**5**)** Consum **=** **((**5UL**\***Config**.**Cylinders**\***injectionTime**\***Config**.**CcPerMin**)** **/** 400000UL**);**

**else** Consum **=** **((**5UL**\***Config**.**Cylinders**\***injectionTime**\***Config**.**CcPerMin**\***Config**.**PulsPerRot**)** **/** **(**144UL**\***WEGpulses**\***Config**.**Wheel**));**

SpeedAvg **=** **((**36UL**\***Accu**.**Distance**)/(**10UL**\***Accu**.**Measurements**));**

**if(**Accu**.**Distance**>**999**)**

**{**

ConsumAvg **=** **(**Accu**.**spentFuel**/**Accu**.**Distance**);** //l/100km \*10

availableDistance **=** **(((**Accu**.**remainingFuel**/**1000**)\*(**Accu**.**Distance**/**10**))** **/** Accu**.**spentFuel**)\***10**;** //km

**}**

**else** **{**ConsumAvg **=** 0**;** availableDistance **=** 0**;}**

/\*

InjectionTime – sumaryczny czas wtrysku zliczony w czasie 1s [ms\*48]

WEGpulses – liczba impulsów z przetwornika drogi zliczona w czasie 1 sekundy

spentFuelPer1s – paliwo spalone w czasie ostatniej sekundy [ul]

Accu.Measurements – akumulator liczby interwałów pomiarowych [s]

Accu.spentFuel – akumulator ilości spalonego paliwa [ul]

Accu.remainingFuel – akumulator ilości paliwa pozostającego w baku [ul]

Accu.Distance – akumulator przejechanego dystansu [m]

Consum – chwilowe zużycie paliwa [l\*10/h], dla prędkości≤5 km/h lub [l\*10/100km], dla prędkości>5 km/h

ConsumAvg – średnie zużycie paliwa [l\*10/100km]

Speed – prędkość chwilowa [km/h]

SpeedAvg – prędkośc średnia [km/h]

availableDistance – orientacyjny, dostępny dystans na paliwie pozostającym w baku [km]

Config.CcPerMin – stała wtryskiwacza [ml/min]

Config.PulsPerRot – stała przetwornika drogi [imp/obr]

Config.Cylinders – liczba wtryskiwaczy paliwa

Config.Wheel – obwód opony [cm]

\*/