ScreenKey TFT100 Dev Kit

User's Guide 1.0

October, 2009

Based on TFT128 version 1.5

www.ScreenKeys.com

SCREENKEYS

ScreenKey TFT100 Dev Kit

User’s Guide

Information in this document is subject to change without notice. The latest revisions may be accessed on the SKI Web site.

Web:	www.ScreenKeys.com

Technical Support is available

via Email	support@screenkeys.com
via Web:	www.screenkeys.com

DISCLAIMER:

ScreenKeys reserves the right to revise data file formats and functionality at any time.

Table of Contents

I N T R O D U C T I O N

ScreenKey TFT100 Dev Kit Overview

Introduction

The TFT100 Dev Kit allows developers to understand and experiment with the features and functionality of the TFT128 ScreenKey.

The TFT100 Dev Kit contains one on-board TFT128 switch and includes source firmware written in 'C' which can be easily modified and recompiled using a free 'C' compiler. Developers can easily modify the provided source code to generate their own images, text and other functionality to test and prototype using the TFT128 ScreenKey.

The TFT100 Dev Kit is reprogrammable via USB.

Software Updates

From time to time, SKI will issue new firmware updates as well as new programming tools. These can be downloaded from ScreenKeys website at www.ScreenKeys.com.

Sample Code

Sample code, in C, is provided but new versions may be released from time to time to illustrate how to interface to the TFT128 ScreenKey. Sample code can be accessed from the ScreenKeys web site at www.ScreenKeys.com.

Disclaimer

ScreenKeys reserves the right to revise this user manual or product specifications at any time. Code written to interface to a TFT128 ScreenKey should check the version number (see TFT128 Datasheet for reading version number) to ensure compatibility.

H A R D W A R E

TFT100 Schematic

Dev Kit Hardware

The TFT100 Dev Kit is a simple microcontroller unit that interfaces to a TFT128 ScreenKey via a 4-wire SPI interface.

The schematic is shown on the following page.

The MCU is an Atmel AT89C5131 which is reprogrammable in-circuit via USB. The power supply to the board is taken from the USB 5.0Vdc supply and down-converted to 3.3Vdc. The MCU and TFT128 are both powered at 3.3Vdc. The MCU has 32KB flash memory and 1Kb RAM.

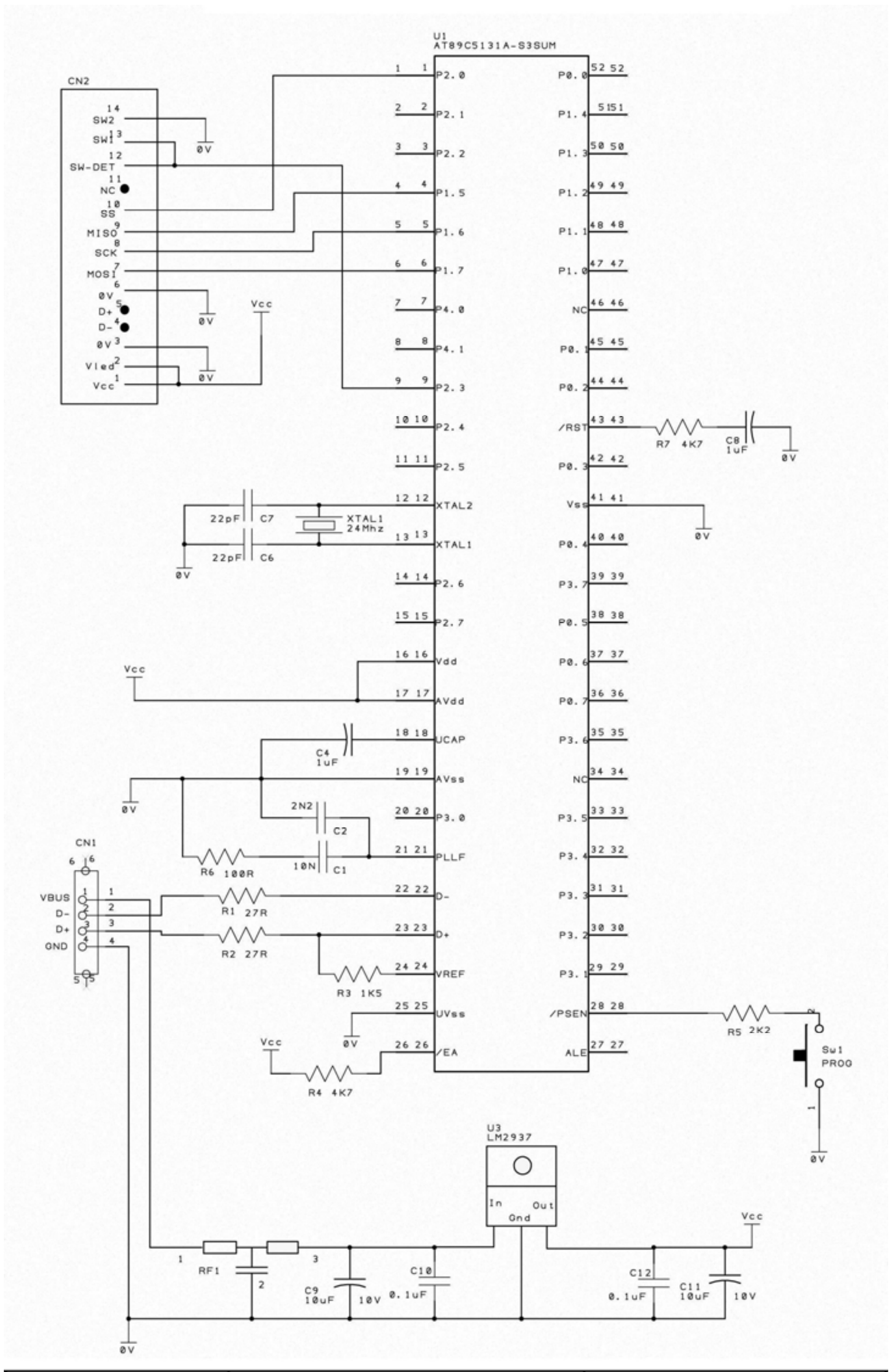
Currently, the USB interface is only used for programming and for power supply purposes. The TFT100 does not enumerate on the USB bus to offer direct control.

The Atmel AT89C5131 provides an integrated SPI interface. This is connected to the TFT128 as follows with the MCU initialised as the SPI **master**:

TFT128	***AT89C5131***
MISO	P1.5
SCK	P1.6
MOSI	P1.7
SS	P2.0

The SS (Slave Select) line into the TFT128 is not part of the MCU SPI interface. This is a generic output which is used to select the TFT128 when the MCU wants to begin a communication.

The switch detection from the TFT128 is configured for both internal and external monitoring. SW2 is grounded on the TFT100 and SW1 is looped into SW-DETECT. SW1 is also connected to the MCU P2.3 which is configured as an input. The MCU can independently monitor switch closures via this port input.



ScreenKeys	TFT100 Rev 2	May 22, 2009

G E T T I N G   S T A R T E D

Programming Tools

Atmel FLIP

The onboard MCU on the TFT100 is an Atmel AT89C5131. This is 8051-based CPU which is reprogrammable via USB.

Atmel provide a programming utility called FLIP which should be downloaded from Atmel's website:

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3886

To run FLIP also requires the Java Runtime Environment. Atmel offer a FLIP download that includes the Java Runtime Environment or a FLIP download without this for users who already have Java installed. There is also a Java Runtime install included on the accompanying TFT100 Dev Kit CD.

Use of the FLIP program is beyond the scope of this document as the "ScreenKey USB Programmer" provides a simple functional subset of FLIP for use with the TFT100. Installation of FLIP and Java Runtime is a necessary precursor to using ScreenKey USB Programmer.

FLIP is offered for both Linux and Windows.

ScreenKeys USB Programmer

The accompanying TFT100 Dev Kit CD includes a folder called "ScreenKey USB Programmer". This includes a command-line utility which identifies when the TFT100 is attached via USB and programs a specified hex file automatically.

To use this utility, copy the contents of the "ScreenKey USB Programmer" utility to a local folder on your own computer. Included in this folder is the current version of the TFT100 firmware (called TFTDemo.hex).

Run the utility by running the included batch file (USBProg.bat) or opening a command window, navigating to the local folder and typing:

```
<drive>:\<local folder>\”ScreenKey USB Programmer” TFTDemo.hex
```

i.e. pass the name of the hex file to be programmed to the utility on the command line.

Note that *ScreenKey USB Programmer* only runs under Windows.

Programming Mode

The TFT100 does not automatically boot into USB programming mode. Its default startup mode is to run the user programmed application.

To enter programming mode, hold the PROG button on the TFT100 while plugging it into a USB socket.

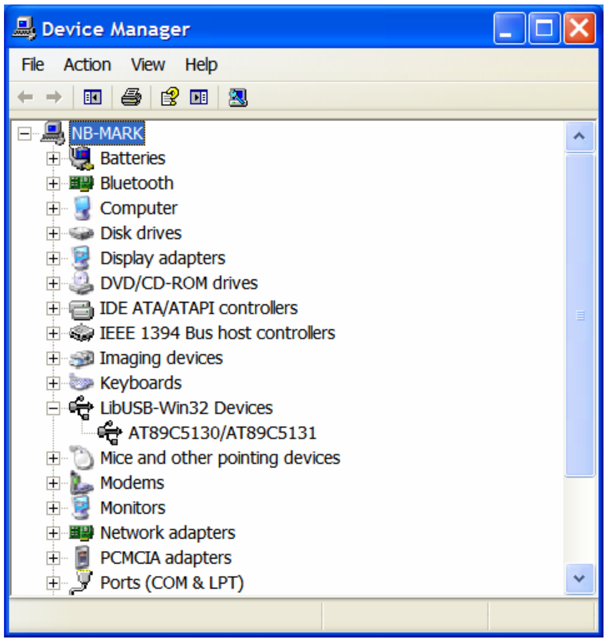
First-time Device Identification

The first time the TFT100 is plugged in, it will identify that an unknown device has been attached. Depending on your operating system, specify that you want to install a specific driver from a specific location.

Browse to the accompanying CD, and open the folder "USB DFU Driver". This will begin installation of the Atmel AT89C5131 programming driver files.

After installation completes, open *Device Manager* and you should see the following entry that confirms the device has been installed successfully:

LibUSB-Win32 Devices
AT89C5130/AT89C5131



NB-MARK
Batteries
Bluetooth
Computer
Disk drives
Display adapters
DVD/CD-ROM drives
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Imaging devices
Keyboards
LibUSB-Win32 Devices
AT89C5130/AT89C5131
Mice and other pointing devices
Modems
Monitors
Network adapters
PCMCIA adapters
Ports (COM & LPT)

SDCC Compiler

The accompanying TFT100 demonstration source code is written in “C” for the SDCC compiler.

SDCC is a free compiler which can be downloaded from the web:

http://sdcc.sourceforge.net/

Go the download page and download the relevant install for your operating system.

The supplied source code compiles correctly with version 2.9 of SDCC. A distributable install of 2.9 for Windows is included on the accompanying CD.

Startup Sequence

Before beginning programming the TFT100, the following sequence should be adhered to:

1. Attach TFT100 to USB socket and ensure pre-programmed application runs correctly. Press button and note that on-screen images change.
2. Remove USB cable and re-power while holding the PROG button. Release the PROG button after the TFT128 ScreenKey screen lights up.
3. Point your operating system to the Atmel USB DFU drivers on the accompanying CD in folder “USB DFU Driver”. Wait for computer to successfully finish driver installation.
4. Check *Device Manager* for a new entry called “LibUSB-Win32 Devices” and a sub-entry called “AT89C5130/AT89C5131”.
5. Install FLIP from web download or use install provided on CD.
6. Install SDCC from web download or use install provided on CD.
7. Copy CD folder called “ScreenKey USB Driver” to a suitable local drive/folder.
8. Copy the TFT100 demo source code from the folder called “TFT100 Source” to a suitable local drive/folder.

You are now ready to begin modifying the supplied source code and to make the TFT128 perform.

D E M O F I R M W A R E

Demonstration Source Code

File Listing

The source code for the TFT100 is supplied on the accompanying CD in the folder “TFT100 Source”.

The included files are:

TFTDemo.c	Main runtime file with all functions to access TFT128
Images.c	Storage for different images used in the demo
At89C5131.h	Includes SDCC compatible SFR definitions
Ext_5131.h	Defines masks for accessing SFR registers
Compiler.h	Generic definitions for various variable types
Build.bat	Command-line utility to build application
TFT100v1_0.hex	Original hex file as pre-programmed in TFT100

A subfolder called “temp” exists off the source folder. During compilation, SDCC generates various files (e.g. map, mem, asm, lst, etc) and these are stored in the temp folder.

A successful compilation results in a file called ***TFTDemo.hex*** created in the source folder. This can be directly programmed into the TFT100 using the *ScreenKey USB Programmer* utility by running the supplied batch file from the command line (remember to navigate to the folder where *ScreenKey USB Programmer* resides and also to copy the TFTDemo.hex file into this location):

<drive>:\<local folder>\USBProg

or, running the utility directly:

<drive>:\<local folder>\”ScreenKey USB Programmer” TFTDemo.hex

Remember to include the quotation marks as otherwise the command line will fail.

Source Code Structure

The source code for the TFT100 Dev Kit demo is quite simple. It simply scrolls repetitively through a series of screens or pages displayed on the TFT128 that each demonstrates a feature of the TFT128. Navigation through the sequence is controlled by pressing the TFT128.

Open TFTDemo.c and navigate to **main()** which is the final function in the module.

The first task implemented is to configure the TFT100 hardware. There are no interrupts or timers used, so this function simply configures the MCU registers accordingly and sets up the SPI interface.

Before beginning the page display sequence, main() displays a *TestCard* image on the TFT128 using the RLE compressed 256-color image from Images.c. It also downloads the ScreenKeys logo image into the TFT128 storage memory for later recall.

The application then begins to wait for a keypress detection. This is implemented as “external switch monitoring” from the TFT128 perspective. That is, the TFT100 MCU monitors its input line P2.3 which is held high until the TFT128 is pressed. Once a switch press is detected, the application executes the next command sequence. Each command sequence is designed to demonstrate a different aspect of the TFT128 command set.

The source code is well documented and does not need to be elaborated on here. The command sequences should be examined in conjunction with the TFT128 Datasheet (a copy of which is included on the accompanying CD).

TFT128 Command Implementation

Commands issued to the TFT128 must comply with the TFT128 datasheet, including adhering to the inter-byte delay between bytes and implementing the flow control based on the SSB return byte.

There is a low-level routine called **SPI_SEND_BYTE()** which transmits a passed byte over SPI and resends the byte if the returned SSB indicates that the TFT128 is currently busy.

There is a higher-level function (**SPISendPktC** and **SPISendPktX**) that use **SPI_SEND_BYTE** to transmit a full command packet. This function selects the TFT128 and sends the command header, computes and sends the XOR byte, and then sends the length and body of the rest of the command. Finally, before exiting it issues 00 bytes to read back any relevant data associated with the command. It deselects the key prior to exiting. As each byte is transmitted, this function checks the returned SSB for a NACK and exits the sending process if one is received after resyncing the key by sending 00 bytes until the key returns OK again.

Two functions are supplied for this process: **SPISendPktC** and **SPISendPktX**. The only difference between these is that the underlying compiled machine code is simplified and executes faster if the inherent pointers are predefined for commands stored in code space (use **SPISendPktC**) or stored in xdata space (use **SPISendPktX**).

Text Handling

Text handling is a major feature of the TFT128 using the internal font and character generator.

The TFT128 maintains an internal cursor position and it automatically displays text in sequence form the current cursor position.

However, many text displays will require accurate positioning to ensure test is centred or simply to format how the text is displayed to the user. To accommodate this, the TFT128 supports a cursor positioning command.

The TFT100 source code includes a function that receives a x/y starting position and the required text string, and then issues the appropriate commands to the TFT128 to display this text at the required position.

High-Speed Mode

For fast changing images, many users will want to operate the TFT128 in high-speed mode. This mode accommodates a frame refresh rate up to 10 frames per second.

The Atmel MCU does not have sufficient capacity or available memory space to supply full-screen images at 10 frames per second.

To help users to investigate high-speed mode, the supplied source code demonstrates how to activate high-speed mode and how to toggle the TFT128 back into command mode afterwards. It is possible that many users will want to jump between the two modes for different uses.

To display some data on the screen in high-speed mode, the source code converts 256-color images (see Images.c) into 16-bit color images using the color palette included in Images.c. Two functions are provided for decompressing RLE compressed 256-color images for this purpose: one is written entirely in 'C' and the other is mostly in assembler. These are provided for information purposes only. It is not recommended to convert RLE compressed or any 256-color image into a 16-bit color image for transmission via high-speed mode.

The fastest refresh rate is achieved using 256-color RLE compressed images. The frame refresh rate is primarily dependent on the communications speed to transmit the information according to the SPI restrictions. Although command mode has a longer inter-byte delay, the significantly reduced number of bytes to transmit an

RLE compressed image can often mean a massive reduction in the time required to display that image on the key.

Commands Demonstrated

The supplied TFT100 source code demonstrates almost all of the commands available with the TFT128:

- sending 256-color images (RLE compressed and uncompressed)
- sending 16-bit color images in high-speed mode
- exiting and exiting high-speed mode
- displaying text and setting text positioning
- blanking the display to a certain color
- changing the default text and background colors
- replacing a specific color in a sub-window of the display
- adjusting the LED backlight
- storing and recalling images from the TFT128 image store
- setup a flashing message on the screen
- operate in landscape and portrait modes
- the effects of changing the wipe direction for images
- overlaying text on a displayed image
- how to read information from the key, e.g. free memory in the image store, the TFT128 version number and current display contents.

G R A P H I C   I M A G I N G

Image Description & Preparation

The TFT128 supports several different types of images.

The most effective type which maximises the full color display at 16-bits per color or 65,536 colors is to use 16-bit full-color images. These images are defined with a 5-6-5 color definition for RGB (see TFT128 datasheet). When using this type of image it is best to transmit using high-speed mode as the transmission of 32kb (128 * 128 * 2 bytes) is most efficient with only a 3.3usec inter-byte delay.

However, storage and manipulation of multiple full-color images requires a high overhead in memory storage. The TFT128 supports other image types which help to reduce the memory overhead for image manipulation.

256-Color Images

A useful type of image when memory is restricted is 256-Color palletised images. These images are defined with one byte per pixel where this byte is an entry into a lookup table. The lookup table is a palette of colors which are then displayed for that pixel.

This means that we can have a maximum of 256 colors only in any particular image. However, the resulting image is very well defined and particularly where the device is a selection device and not a primary display device.

All the images used in the TFT100 source code are 256-color images.

The color palette used in the TFT128 is two-bytes per color (i.e. 16-bit) as per the 16-bit color definition in the TFT128 datasheet.

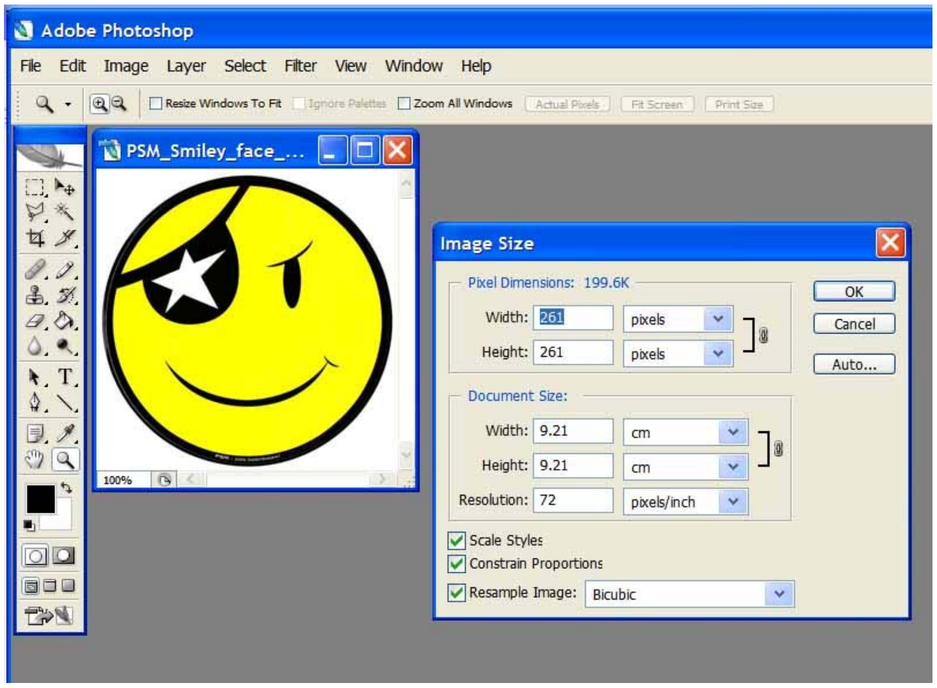
The TFT100 source code implements a 256 color palette that directly emulates the color palette used in Windows BMP files for display on Windows PC's. This is the same factory default color palette used by the TFT128.

Although the palette in the TFT128 can be modified it is preferable to always use the same palette for all TFT128 images so that the palette does not need to be modified and so that all images maintain the same color consistency as expected.

Adobe Photoshop is a useful tool for creating images for use on the TFT128.

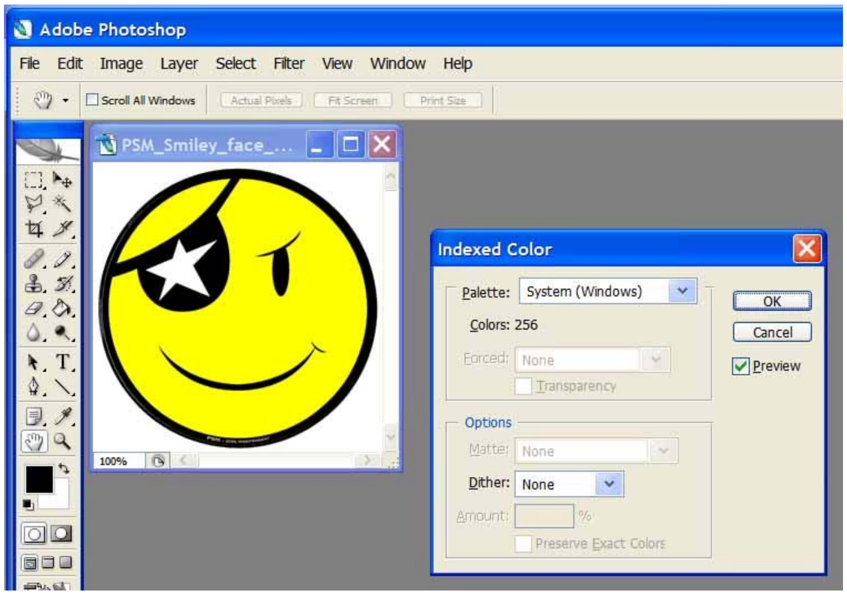
First open or create a new image in Photoshop. This should be a maximum size of 128 * 128 but smaller images will take up less space.

For example, here is a large image (261*261) that we want to prepare for use in the TFT128:



Pixel Dimensions: 199.6K
Width: 261 pixels
Height: 261 pixels
Document Size:
Width: 9.21 cm
Height: 9.21 cm
Resolution: 72 pixels/inch
[✓] Scale Styles
[✓] Constrain Proportions
[✓] Resample Image: Bicubic

First, it is necessary to change the image to use a color palette. Select “Image” -> “Mode” -> “Indexed Palette”, and then select “System (Windows)” from the dialog window:



Indexed Color

Palette: System (Windows)

Colors: 256

Forced: None

[ ] Transparency

Options

Matte: None

Dither: None

Amount: %

[ ] Preserve Exact Colors

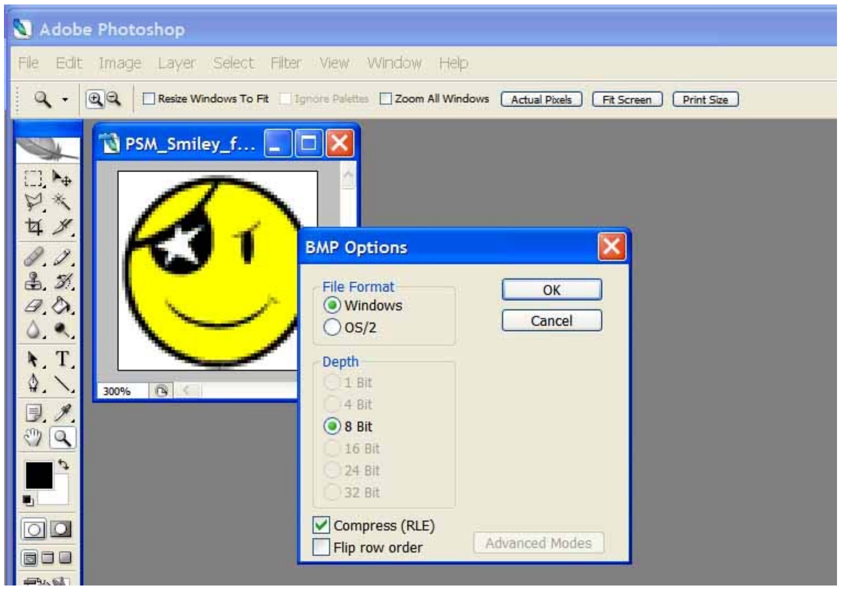
OK

Cancel

[✓] Preview

Next, adjust the image size to our requirements (e.g. 60*60). Select “Image” -> “Image Size…” and set the new size in the dialog box.

Finally, save the image as a BMP file and 8-bit color RLE compressed:



BMP Options

File Format

[✓ Windows] OS/2

Depth

1 Bit 4 Bit [✓ 8 Bit] 16 Bit 24 Bit 32 Bit

[✓] Compress (RLE)

[ ] Flip row order

OK

Cancel

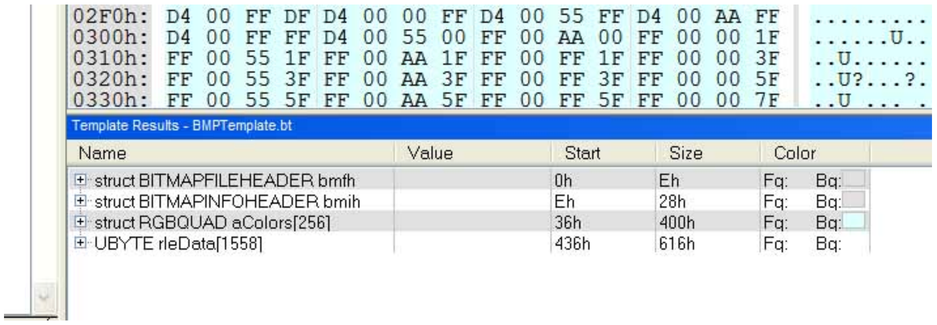
Advanced Modes

This will produce a BMP file where the image data bytes are stored from bottom-left position going left to right and from bottom to top of the displayed image.

When sending this image to the TFT128, remember to set the wipe direction as bottom-up.

Once the image is created as a BMP, the image data must be extracted into a C-array suitable for use in the C program. A good editor for this is the **010 Editor**.

Open the BMP file in 010 which includes a BMP format identifier:

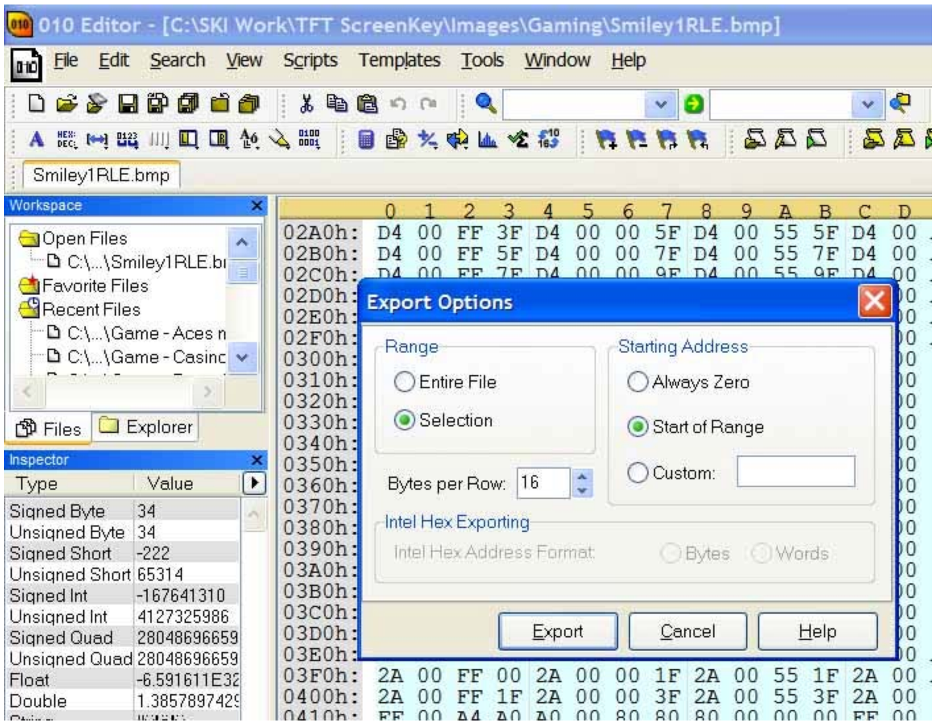


```
02F0h: D4 00 FF DF D4 00 00 FF D4 00 55 FF D4 00 AA FF  .........
0300h: D4 00 FF FF D4 00 55 00 FF 00 AA 00 FF 00 00 1F  ......U..
0310h: FF 00 55 1F FF 00 AA 1F FF 00 FF 1F FF 00 00 3F  ..U......
0320h: FF 00 55 3F FF 00 AA 3F FF 00 FF 3F FF 00 00 5F  ..U?...?.
0330h: FF 00 55 5F FF 00 AA 5F FF 00 FF 5F FF 00 00 7F  ..U ... .
```

Template Results - BMPTemplate.bt

Name	Value	Start	Size	Color
⊞ struct BITMAPFILEHEADER bmfh		0h	Eh	Fg: Bg:
⊞ struct BITMAPINFOHEADER bmih		Eh	28h	Fg: Bg:
⊞ struct RGBQUAD aColors[256]		36h	400h	Fg: Bg:
⊞ UBYTE rleData[1558]		436h	616h	Fg: Bg:

Highlight the "rleData" section of the identifier and this selects the image data only. Next export this as a C-code:



```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D
02A0h: D4 00 FF 3F D4 00 00 5F D4 00 55 5F D4 00
02B0h: D4 00 FF 5F D4 00 00 7F D4 00 55 7F D4 00
03F0h: 2A 00 FF 00 2A 00 00 1F 2A 00 55 1F 2A 00
0400h: 2A 00 FF 1F 2A 00 00 3F 2A 00 55 3F 2A 00
```

Export Options:
- Range: ( ) Entire File [✓ Selection]
- Starting Address: ( ) Always Zero [✓ Start of Range] ( ) Custom:
- Bytes per Row: 16
- Intel Hex Exporting: Intel Hex Address Format: ( ) Bytes ( ) Words

Inspector

Type	Value
Signed Byte	34
Unsigned Byte	34
Signed Short	-222
Unsigned Short	65314
Signed Int	-167641310
Unsigned Int	4127325986
Signed Quad	28048696659
Unsigned Quad	28048696659
Float	-6.591611E32
Double	1.3857897429

The output of this can be directly used in the Images.c file. Remember to provide a suitable name to the array and to specify that it is stored in code space.

16-bit Color Palette

The BMP file created above uses a 256-color palette. However if you examine this palette in 010 Editor it is obvious that the colors are not 16-bit but 24-bit. Each color is defined with 4 bytes, one byte each for red, green and blue and one byte always set to 00.

It is not necessary to change the color palette so long as all images are created with the Windows System palette. Then the factory default palette in the TFT128 may be used without modification.

However, to create a new palette from a Photoshop created 256-color image, the 010 Editor is again very useful. 010 provides a scripting feature which allows data to be manipulated with C-like code scripts. Here is a 010 script for converting a series of 4-byte color definitions (24-bit color) to a 16-bit (2 byte) color definition suitable for the TFT128:

```
// Define variables
const  int BLOCK_SIZE = 1024;
uchar  buffer[ BLOCK_SIZE ];
quad   size, pos;
int    i, bufsize;
int    CurrFile, NewFile;
uint16 newRed, newGreen, newBlue;
uint16 newCol;
float  tmp;

// Check that a file is open
if( FileCount() == 0 )
{
    MessageBox( idOk, "RGBQUADto565", "RGBQUADto565 can only be
executed when a file is loaded." );
    return -1;
}

// Get current file handle
CurrFile = GetFileNum();

// Open new file to store converted data
NewFile = FileNew();

// Read file as a set of blocks
//  - more efficient this way
FileSelect(CurrFile);
pos  = 0;
size = FileSize();

//Write start of C-array
FPrintf(NewFile,"unsigned int ColorTable[256] = { \r\n     ");
while( size > 0 )
{
    // Read set of bytes from the file
    bufsize = size < BLOCK_SIZE ? size : BLOCK_SIZE;
    ReadBytes( buffer, pos, bufsize );

    // RGBQUAD is 4 bytes - <BLUE> <GREEN> <RED> <reserved>
    for( i = 0; i < bufsize; i=i+4 )
    {
        // Calc RED value converted from 8-bit val (255)
        // to 5 bits (31)
        tmp = buffer[i] * 0x1F;
        tmp = tmp / 0xFF;
        newBlue = tmp;

        // Calc GREEN value converted from 8-bit val (255)
        // to 6 bits (63)
        tmp = buffer[i+1] * 0x3F;
        tmp = tmp / 0xFF;
        newGreen = tmp;

        // Calc BLUE value converted from 8-bit val (255)
        // to 5 bits (31)
        tmp = buffer[i+2] * 0x1F;
        tmp = tmp / 0xFF;
        newRed = tmp;

        // Ignore last byte in RGBQUAD

        // Create new 16 bit color value
        newCol = (newRed << 11) | (newGreen << 5) | (newBlue);

        // Write new 16-bit color value to new file
        //FPrintf(NewFile,"Buffer %x %x %x
%x\n",buffer[i+j],buffer[i+j+1],buffer[i+j+2],buffer[i+j+3]);
        //FPrintf(NewFile,"Red %2x  Green %2x  Blue
%2x\nCombined %2x\n",newRed,newGreen,newBlue,newCol);
        FPrintf(NewFile, "0x%2.04x, ",newCol);
    }
    // Advance to next block
    pos  += bufsize;
    size -= bufsize;
}

FPrintf(NewFile, "};\n");

return 1;
```

This routine is easily converted using another programming language or tool to extract 16-bit color form a 24-bit color palette. Remember to extract the palette information from the raw BMP file before running the conversion routine. The output of this routine is an ASCII file formatted for direct inclusion into a C source file.

APPENDIX A

Documentation Control

A.1 Change Control

This document is the responsibility of the author and is subject to formal change control after the initial approved release (i.e. issue 1.0).

A.2 Abbreviations Used/Terms of Reference

Fixed Keys	Standard POS Console keys--contrast with ScreenKeys.
LED	Light Emitting Diode. There are 4 LEDs on the Model 6000
LRC	Longitudinal Redundancy Check--a byte used to verify that the data read from a Magnetic Card is valid.
MSR	Magnetic Stripe Reader
O/S	Operating Systems -- MS DOS, UNIX, OS/2, Windows etc...
host	Personal Computer.

A.3 Historical Change Reference

Issue	Date	Author	Changes Made

A.4 Change Summary