The TFT FT810 shield Tutorial, Reference and Cookbook

(Preliminary)

Contents

1. Overview

TFT_FT810 is a shield that adds a bright 2.8 inch touchscreen, an embedded GPU, headphone jack and microSD slot to your Arduino - or anything else with an SPI interface. Everything you need to create compelling games -- right in your hand.

- video output is 320x240 pixels in 24-bit color
- OpenGL-style command set
- Up to 2000 sprites, any size
- 1 Mbyte of video RAM
- smooth sprite rotate and zoom with bilinear filtering
- smooth circle and line drawing in hardware - 16x antialiased
- JPEG loading in hardware
- built-in rendering of gradients, text, dials and buttons

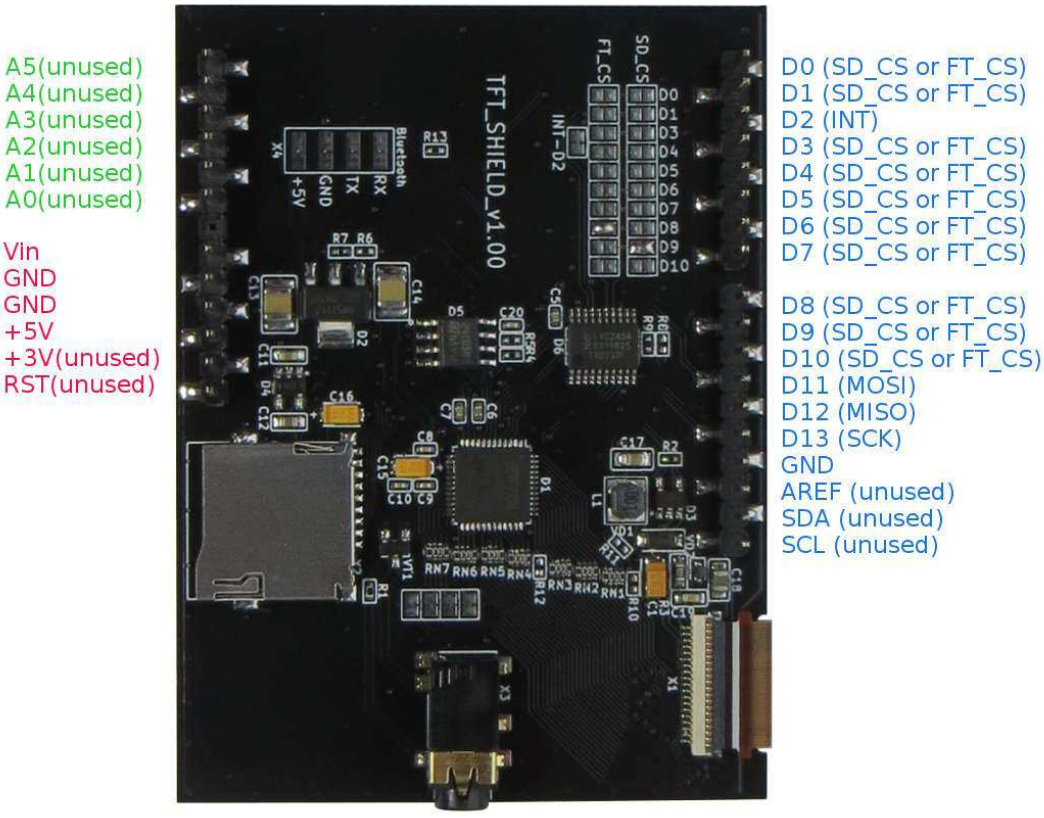
Sound output is via an amplified headphone jack. The system supports a selection of built-in samples and instruments, and can also play samples from video memory, at up to 48KHz.
The FT800 can handle media in several formats directly. These save CPU memory, and mean simpler code on the CPU.

- decodes JPEGs directly
- for lossless image loading, decompresses (zlib INFLATE) directly
- audio samples can be 8-bit linear, ulaw or ADPCM

Built into the GPU's ROM are:

- high-quality fonts in 6 sizes
- samples of 8 musical instruments, playable by MIDI note
- samples of 10 percussion sounds

and of course you can load your own fonts and audio samples into the 256KB RAM.



2. Quickstart

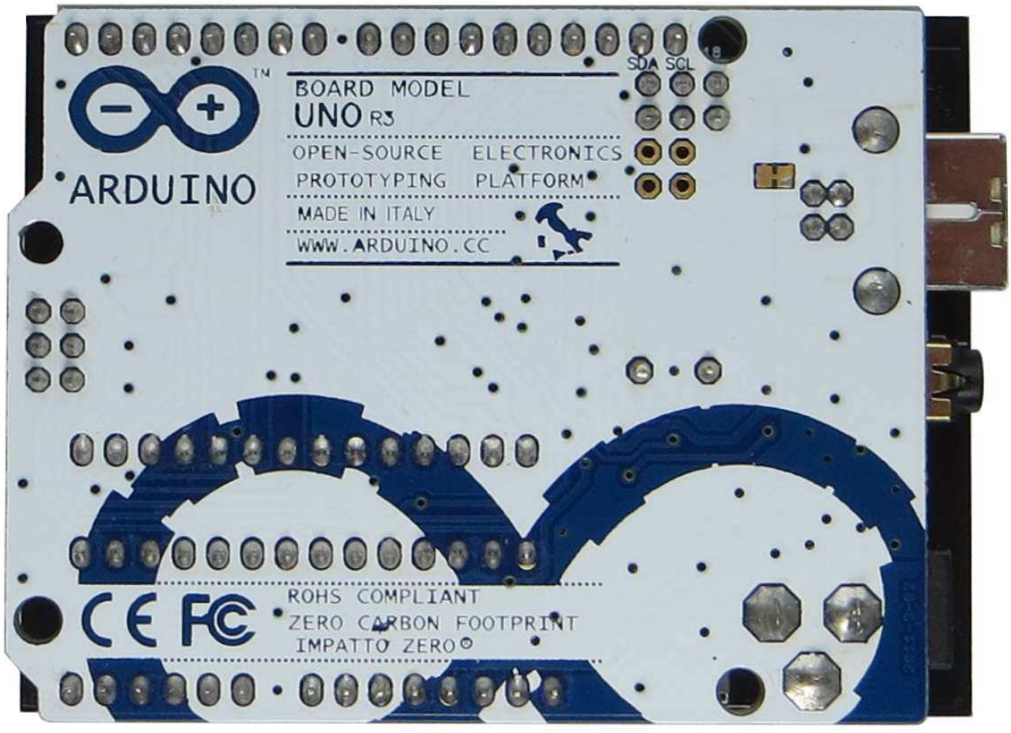
2.1 Preparation for Work

2.1.1 Installing Libraries

Download the Gameduino 2 library Gameduino2.zip from http://gameduino.com/code and install it in the Arduino IDE. Instructions for doing this are at http://arduino.cc/en/Guide/Libraries.

2.1.2 Device assembly

Attach TFT_FT810 shield to the Arduino, making sure that the pins are aligned properly, and that none are bent.
Power up the Arduino. Nothing will appear on the TFT_FT810 shield screen until a sketch is loaded on the Arduino.
Copy the necessary files (images, video, sound) to the microSD and install the microSD into the appropriate slot on the shield.



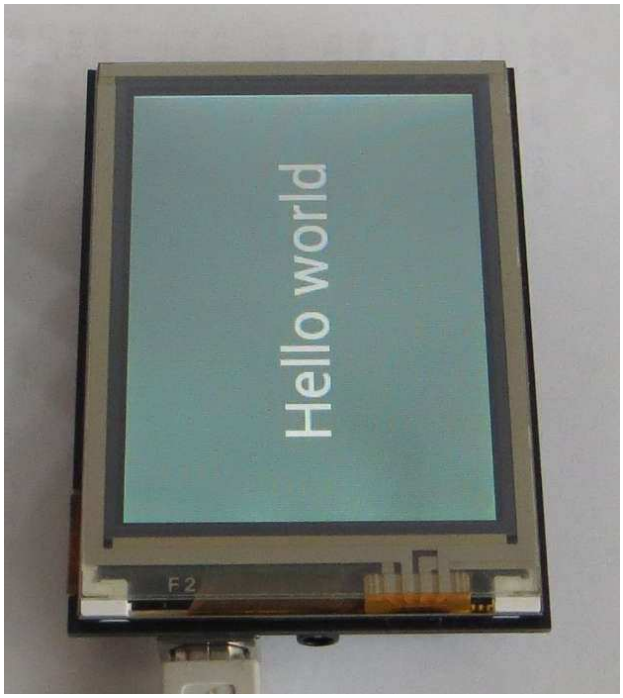
3. Examples

3.1 Draw text “Hello”. Rotate screen.

Copy sketch *hello.ino*:

```
#include <EEPROM.h>
#include <SPI.h>
#include <GD2.h>

void setup()
{
  GD.begin(0);
  TftFt810Init(0);
}

void loop()
{
  GD.ClearColorRGB(0x103000);
  GD.Clear();
  GD.cmd_text(GD.w / 2, GD.h / 2, 31, OPT_CENTER, "Hello world");
  GD.swap();
}

void  TftFt810Init(byte  rotate)
{
  GD.wr16(REG_HCYCLE, 263); GD.wr16(REG_HOFFSET, 22);
  GD.wr16(REG_HSIZE, 240);  GD.wr16(REG_HSYNC0, 10);
  GD.wr16(REG_HSYNC1, 12);  GD.wr16(REG_VCYCLE, 436);
  GD.wr16(REG_VOFFSET, 115);  GD.wr16(REG_VSIZE, 320);
  GD.wr16(REG_VSYNC0, 111);  GD.wr16(REG_VSYNC1, 113);
  GD.wr32(REG_SWIZZLE, 2);  GD.wr16(REG_CSPREAD, 1);
  GD.wr16(REG_DITHER, 1);  GD.wr16(REG_PCLK_POL, 0);
  GD.wr16(REG_PCLK, 5);  GD.cmd_setrotate(rotate);
  if (rotate > 1) { GD.w = 320; GD.h = 240;}
  else { GD.w = 240; GD.h = 320; }
  static const byte canned_calibration[24] = {
  0x76, 0x00, 0x00, 0x00, 0xBC, 0xB1, 0xFF, 0xFF,  0x6A, 0x12, 0x12, 0x01,
  0xC4, 0x9E, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xBA, 0x9B, 0x63, 0x01};
  for (int i = 0; i < 24; i++) GD.wr(REG_TOUCH_TRANSFORM_A + i, canned_calibration[i]);
}
```

Additional settings for the screen parameters are carried out in the *TftFt810Init(byte rotate)* procedure, the argument of which is the *rotate* parameter, which is responsible for rotating the screen. Below are various screen images depending on the value of the *rotate* parameter:

rotate = 0	*rotate* = 1
	
rotate = 2	*rotate* = 3
	

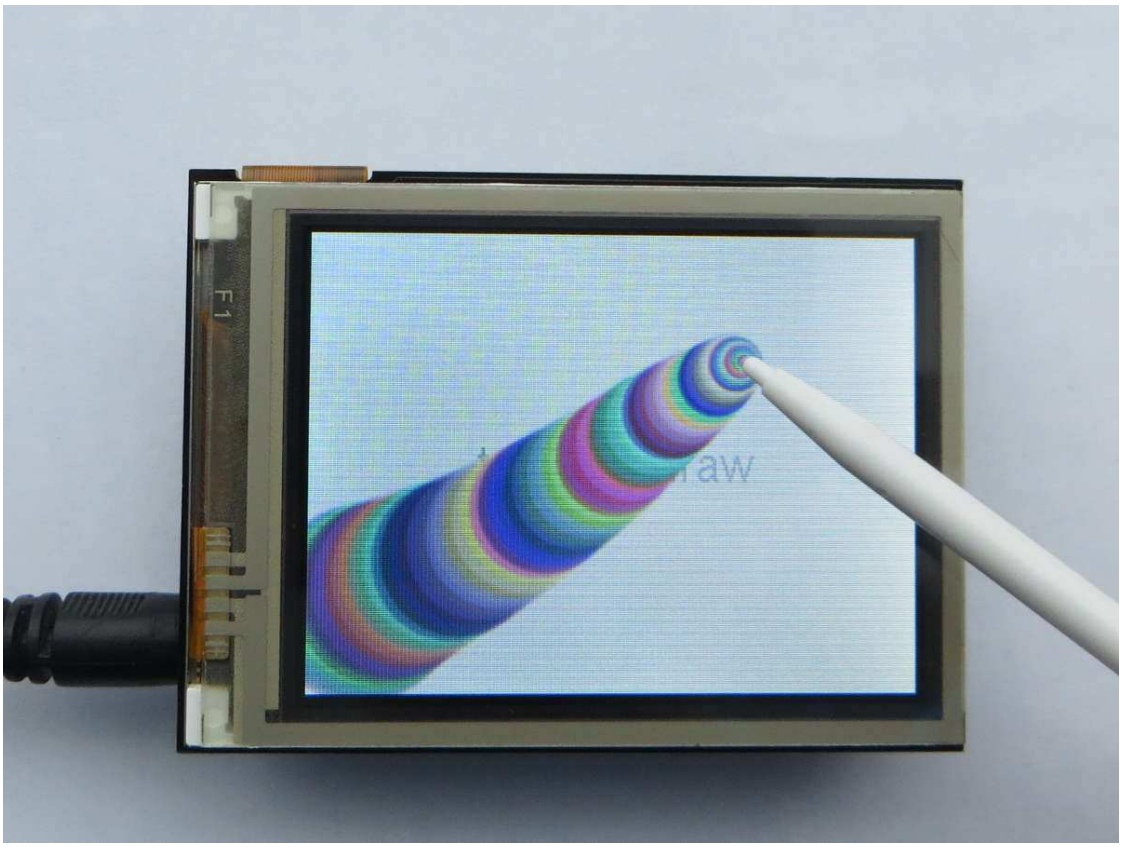
3.2 Work with Touchscreen. Blobs.

Copy sketch *blobs.ino*:

```
#include <EEPROM.h>
#include <SPI.h>
#include <GD2.h>

#define NBLOBS      128

xy blobs[NBLOBS];
const xy offscreen = {-16384, -16384};

void setup()
{
  GD.begin(0);
  TftFt810Init(3);

  for (int i = 0; i < NBLOBS; i++)
    blobs[i] = offscreen;
}

void loop()
{
  static byte blob_i;
  GD.get_inputs();
  if (GD.inputs.x != -32768)
    blobs[blob_i] = GD.inputs.xytouch;
  else
    blobs[blob_i] = offscreen;
  blob_i = (blob_i + 1) & (NBLOBS - 1);

  GD.ClearColorRGB(0xe0e0e0);
  GD.Clear();

  GD.ColorRGB(0xa0a0a0);
  GD.cmd_text(GD.w/2, GD.h/2, 24, OPT_CENTER, "touch to draw");


  GD.Begin(POINTS);
  for (int i = 0; i < NBLOBS; i++) {
    // Blobs fade away and swell as they age
    GD.ColorA(i << 1);
    GD.PointSize((1024 + 16) - (i << 3));

    // Random color for each blob, keyed from (blob_i + i)
    uint8_t j = (blob_i + i) & (NBLOBS - 1);
```

```
    byte r = j * 17;
    byte g = j * 23;
    byte b = j * 147;
    GD.ColorRGB(r, g, b);

    // Draw it!
    blobs[j].draw();
  }
  GD.swap();
}

void  TftFt810Init(byte  rotate)
{
  GD.wr16(REG_HCYCLE, 263); GD.wr16(REG_HOFFSET, 22);
  GD.wr16(REG_HSIZE, 240); GD.wr16(REG_HSYNC0, 10);
  GD.wr16(REG_HSYNC1, 12); GD.wr16(REG_VCYCLE, 436);
  GD.wr16(REG_VOFFSET, 115); GD.wr16(REG_VSIZE, 320);
  GD.wr16(REG_VSYNC0, 111); GD.wr16(REG_VSYNC1, 113);
  GD.wr32(REG_SWIZZLE, 2); GD.wr16(REG_CSPREAD, 1);
  GD.wr16(REG_DITHER, 1); GD.wr16(REG_PCLK_POL, 0);
  GD.wr16(REG_PCLK, 5); GD.cmd_setrotate(rotate);
  if (rotate > 1) { GD.w = 320; GD.h = 240; }
  else { GD.w = 240; GD.h = 320; }
  static const byte canned_calibration[24] = {
  0x76, 0x00, 0x00, 0x00, 0xBC, 0xB1, 0xFF, 0xFF, 0x6A, 0x12, 0x12, 0x01,
  0xC4, 0x9E, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xBA, 0x9B, 0x63, 0x01 };
  for (int i = 0; i < 24; i++) GD.wr(REG_TOUCH_TRANSFORM_A + i, canned_calibration[i]);
}
```

Some images:



3.3 Built-in text, dials and buttons. Widgets.

Copy sketch *widgets.ino*:

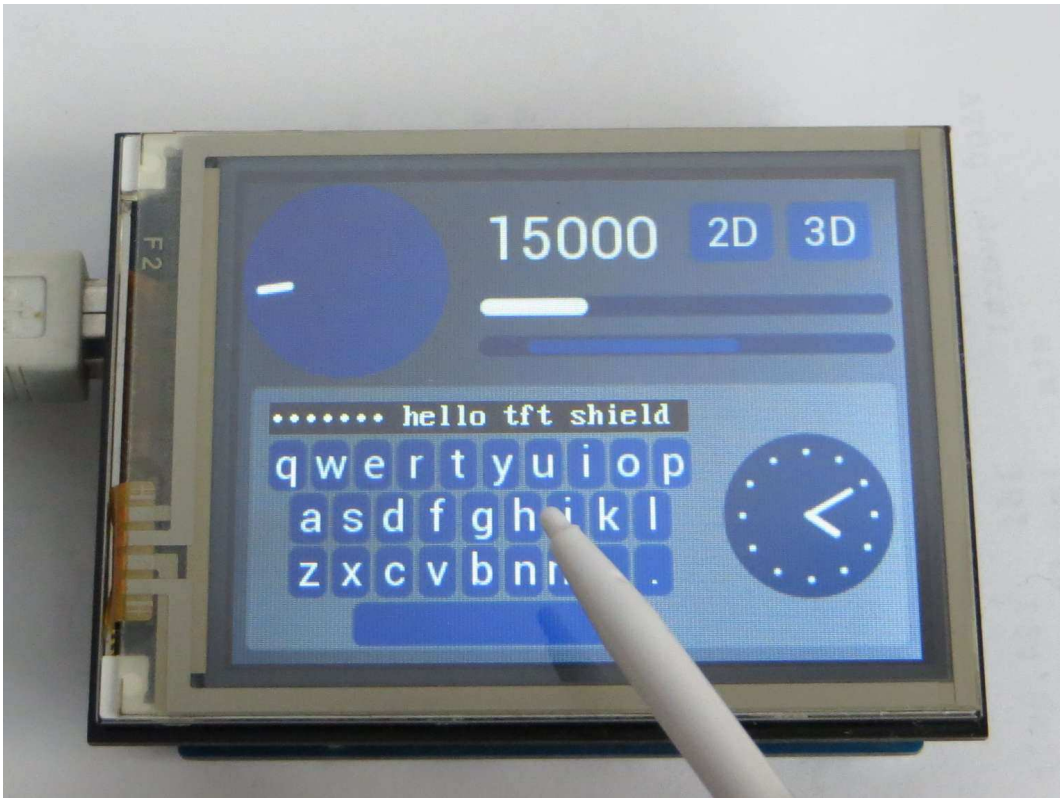
```
#include <EEPROM.h>
#include <SPI.h>
#include <GD2.h>

static uint16_t value = 15000;      // every widget is hooked to this value
static char message[41];            // 40 character text entry field
static uint16_t options = OPT_FLAT;
static byte prevkey;

void setup()
{
  memset(message, 7, 25);
  GD.begin(0);
  TftFt810Init(2);
}

#define TAG_DIAL      200
#define TAG_SLIDER    201
#define TAG_TOGGLE    202
#define TAG_BUTTON1   203
#define TAG_BUTTON2   204

void loop()
{

  GD.get_inputs();

  switch (GD.inputs.track_tag & 0xff) {
  case TAG_DIAL:
  case TAG_SLIDER:
  case TAG_TOGGLE:
    value = GD.inputs.track_val;
  }
  switch (GD.inputs.tag) {
  case TAG_BUTTON1:
    options = OPT_FLAT;
    break;
  case TAG_BUTTON2:
    options = 0;
    break;
  }
  byte key = GD.inputs.tag;
  if ((prevkey == 0x00) && (' ' <= key) && (key < 0x7f)) {
```

```
    memmove(message, message + 1, 24);
    message[24] = key;
  }
  prevkey = key;

  GD.cmd_gradient(0, 0,   0x404044, 320, 320, 0x606068);
  GD.ColorRGB(0x707070);

  GD.LineWidth(4 * 16);
  GD.Begin(RECTS);

  GD.Vertex2ii(10, 109);
  GD.Vertex2ii(310, 229);
  GD.ColorRGB(0xffffff);

  GD.Tag(TAG_DIAL);
  GD.cmd_dial(50, 50, 50, options, value);
  GD.cmd_track(50, 50, 1, 1, TAG_DIAL);

  GD.Tag(255);
  GD.cmd_number(160, 30, 30, OPT_CENTER | 5, value);

  GD.cmd_clock(270, 170, 40, options | OPT_NOSECS, 0, 0, value, 0);

  GD.Tag(TAG_BUTTON1);
  GD.cmd_button(218, 12, 40, 30, 28, options,  "2D");
  GD.Tag(TAG_BUTTON2);
  GD.cmd_button(266, 12, 40, 30, 28, options,  "3D");

  GD.Tag(255);
  GD.cmd_progress(120, 60, 190, 10, options, value, 65535);
  GD.cmd_scrollbar(120, 80, 190, 10, options, value / 2, 32768, 65535);

  GD.cmd_keys(10, 133, 210, 24, 28, options | OPT_CENTER | key, "qwertyuiop");
  GD.cmd_keys(10, 159, 210, 24, 28, options | OPT_CENTER | key,   "asdfghjkl");
  GD.cmd_keys(10, 185, 210, 24, 28, options | OPT_CENTER | key,   "zxcvbnm,.");
  GD.Tag(' ');
  GD.cmd_button(56, 211, 120, 20, 28, options,   "");

  GD.BlendFunc(SRC_ALPHA, ZERO);
  GD.cmd_text(15, 113, 18, 0, message);

  GD.swap();
}

void  TftFt810Init(byte  rotate)
{
  GD.wr16(REG_HCYCLE, 263); GD.wr16(REG_HOFFSET, 22);
```

```
  GD.wr16(REG_HSIZE, 240); GD.wr16(REG_HSYNC0, 10);
  GD.wr16(REG_HSYNC1, 12); GD.wr16(REG_VCYCLE, 436);
  GD.wr16(REG_VOFFSET, 115); GD.wr16(REG_VSIZE, 320);
  GD.wr16(REG_VSYNC0, 111); GD.wr16(REG_VSYNC1, 113);
  GD.wr32(REG_SWIZZLE, 2); GD.wr16(REG_CSPREAD, 1);
  GD.wr16(REG_DITHER, 1); GD.wr16(REG_PCLK_POL, 0);
  GD.wr16(REG_PCLK, 5); GD.cmd_setrotate(rotate);
  if (rotate > 1) { GD.w = 320; GD.h = 240; }
  else { GD.w = 240; GD.h = 320; }
  static const byte canned_calibration[24] = {
  0x76, 0x00, 0x00, 0x00, 0xBC, 0xB1, 0xFF, 0xFF, 0x6A, 0x12, 0x12, 0x01,
  0xC4, 0x9E, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xBA, 0x9B, 0x63, 0x01 };
  for (int i = 0; i < 24; i++) GD.wr(REG_TOUCH_TRANSFORM_A + i, canned_calibration[i]);
}
```

Some images.
Start screen:

Keyboard:



Dial:

3D button:

2D button:

3.4 View JPEG file. Jpeg.

Copy sketch *jpeg.ino*:

```
#include <EEPROM.h>
#include <SPI.h>
#include <GD2.h>

void setup()
{
  GD.begin();
  TftFt810Init(0);

  GD.cmd_loadimage(0, 0);
  GD.load("healsky3.jpg");

}

void loop()
{
  GD.Clear();
  GD.Begin(BITMAPS);
  GD.Vertex2ii(0, 0);
  GD.swap();
}

void  TftFt810Init(byte  rotate)
{
  GD.wr16(REG_HCYCLE, 263); GD.wr16(REG_HOFFSET, 22);
  GD.wr16(REG_HSIZE, 240); GD.wr16(REG_HSYNC0, 10);
  GD.wr16(REG_HSYNC1, 12); GD.wr16(REG_VCYCLE, 436);
  GD.wr16(REG_VOFFSET, 115); GD.wr16(REG_VSIZE, 320);
  GD.wr16(REG_VSYNC0, 111); GD.wr16(REG_VSYNC1, 113);
  GD.wr32(REG_SWIZZLE, 2); GD.wr16(REG_CSPREAD, 1);
  GD.wr16(REG_DITHER, 1); GD.wr16(REG_PCLK_POL, 0);
  GD.wr16(REG_PCLK, 5); GD.cmd_setrotate(rotate);
  if (rotate > 1) { GD.w = 320; GD.h = 240; }
  else { GD.w = 240; GD.h = 320; }
  static const byte canned_calibration[24] = {
  0x76, 0x00, 0x00, 0x00, 0xBC, 0xB1, 0xFF, 0xFF, 0x6A, 0x12, 0x12, 0x01,
  0xC4, 0x9E, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xBA, 0x9B, 0x63, 0x01 };
  for (int i = 0; i < 24; i++) GD.wr(REG_TOUCH_TRANSFORM_A + i, canned_calibration[i]);
}
```

3.5 Play AVI file. Video.

Copy sketch *video.ino*:

```
#include <EEPROM.h>
#include <SPI.h>
#include <GD2.h>

void setup()
{
  GD.begin();
  TftFt810Init(0);
}

void loop()
{
  MoviePlayer mp;

  mp.begin("fun-1500.avi");
  mp.play();
}

void  TftFt810Init(byte  rotate)
{
  GD.wr16(REG_HCYCLE, 263); GD.wr16(REG_HOFFSET, 22);
  GD.wr16(REG_HSIZE, 240);  GD.wr16(REG_HSYNC0, 10);
  GD.wr16(REG_HSYNC1, 12);  GD.wr16(REG_VCYCLE, 436);
  GD.wr16(REG_VOFFSET, 115);  GD.wr16(REG_VSIZE, 320);
  GD.wr16(REG_VSYNC0, 111);  GD.wr16(REG_VSYNC1, 113);
  GD.wr32(REG_SWIZZLE, 2);  GD.wr16(REG_CSPREAD, 1);
  GD.wr16(REG_DITHER, 1);  GD.wr16(REG_PCLK_POL, 0);
  GD.wr16(REG_PCLK, 5);  GD.cmd_setrotate(rotate);
  if (rotate > 1) { GD.w = 320; GD.h = 240;}
  else { GD.w = 240; GD.h = 320; }
  static const byte canned_calibration[24] = {
  0x76, 0x00, 0x00, 0x00, 0xBC, 0xB1, 0xFF, 0xFF,  0x6A, 0x12, 0x12, 0x01,
  0xC4, 0x9E, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xBA, 0x9B, 0x63, 0x01};
  for (int i = 0; i < 24; i++) GD.wr(REG_TOUCH_TRANSFORM_A + i, canned_calibration[i]);
}
```

3.5 Play IMA sound file. Song.

Copy sketch *song.ino*:

```
#include <EEPROM.h>
#include <SPI.h>
#include <GD2.h>

#define MUSICFILE   "mesmeriz.ima"
static Streamer stream;

void setup()
{
  GD.begin();
  TftFt810Init(0);
  stream.begin(MUSICFILE);
}

void loop()
{
  GD.cmd_gradient(0, 40, 0x282830,  0, GD.h, 0x606040);
  GD.cmd_text(GD.w/2, 100, 24, OPT_CENTER, MUSICFILE);
  uint16_t val, range;
  stream.progress(val, range);
  GD.cmd_slider(30, GD.h/2, GD.w-30-30, 8, 0, val, range);
  GD.swap();
  GD.finish();
  stream.feed();
}

void  TftFt810Init(byte  rotate)
{
  GD.wr16(REG_HCYCLE, 263); GD.wr16(REG_HOFFSET, 22);
  GD.wr16(REG_HSIZE, 240);  GD.wr16(REG_HSYNC0, 10);
  GD.wr16(REG_HSYNC1, 12);  GD.wr16(REG_VCYCLE, 436);
  GD.wr16(REG_VOFFSET, 115);  GD.wr16(REG_VSIZE, 320);
  GD.wr16(REG_VSYNC0, 111);  GD.wr16(REG_VSYNC1, 113);
  GD.wr32(REG_SWIZZLE, 2);  GD.wr16(REG_CSPREAD, 1);
  GD.wr16(REG_DITHER, 1);  GD.wr16(REG_PCLK_POL, 0);
  GD.wr16(REG_PCLK, 5);  GD.cmd_setrotate(rotate);
  if (rotate > 1) { GD.w = 320; GD.h = 240;}
  else { GD.w = 240; GD.h = 320; }
  static const byte canned_calibration[24] = {
  0x76, 0x00, 0x00, 0x00, 0xBC, 0xB1, 0xFF, 0xFF,  0x6A, 0x12, 0x12, 0x01,
  0xC4, 0x9E, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xBA, 0x9B, 0x63, 0x01};
  for (int i = 0; i < 24; i++) GD.wr(REG_TOUCH_TRANSFORM_A + i, canned_calibration[i]);
}
```

Image: