Adafruit TFP401 HDMI/DVI Decoder to 40-pin TTL Display

Created by lady ada

https://learn.adafruit.com/adafruit-tfp401-hdmi-slash-dvi-decoder-to-40-pin-ttl-display

Last updated on 2024-03-08 02:01:30 PM EST

Table of Contents

Overview 3

Touch Screen 6

Backlight 8

Editing the EDID 10

Downloads 14

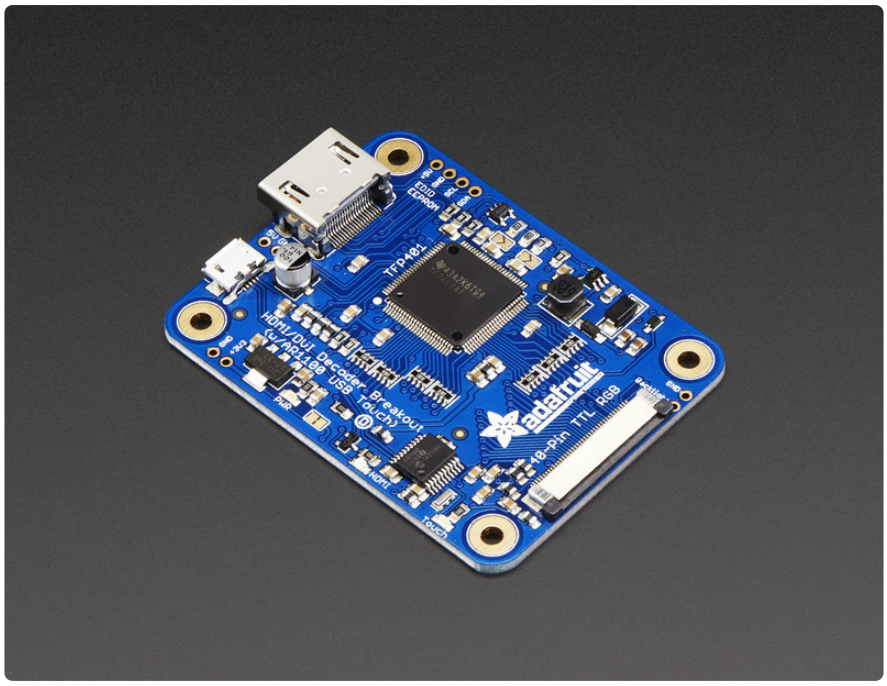
- Datasheets & Files
- Schematics
- Fabrication Print

Raspberry Pi Config 15

Overview

Its a mini HDMI decoder board! So small and simple, you can use this board as an all-in-one display driver for TTL displays, or perhaps decoding HDMI/DVI video for some other project. This breakout features the TFP401 for decoding video, and for the touch version, an AR1100 USB resistive touch screen driver.

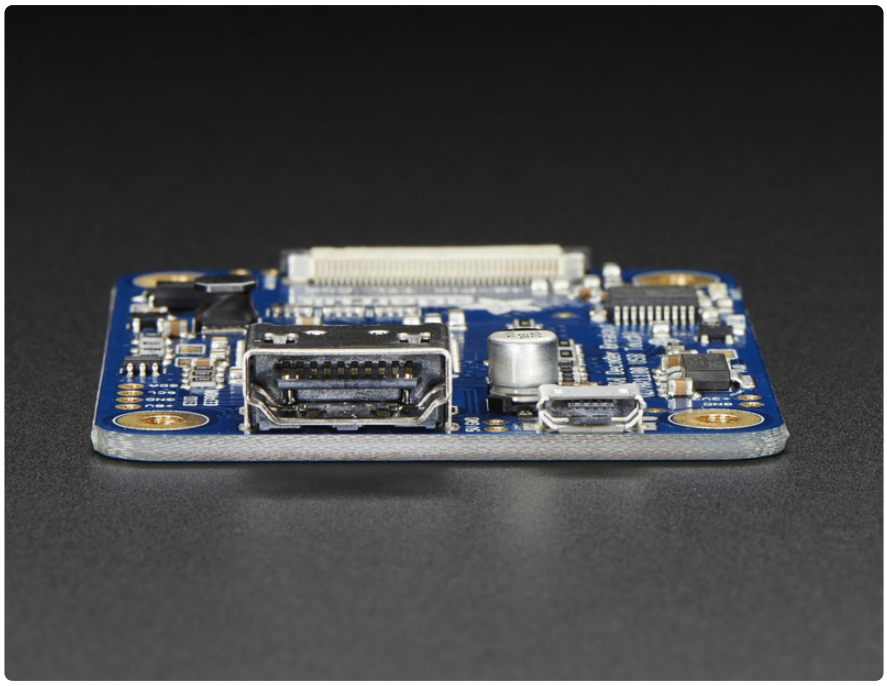
The TFP401 is a beefy DVI/HDMI decoder from TI. It can take unencrypted video and pipe out the raw 24-bit color pixel data - HDCP not supported! It will decode any resolution from 25-165MHz pixel clock, basically up to 1080p. We've used this breakout with 800x480 displays, so we have not specifically tested it with higher resolutions. We added a bunch of supporting circuitry like a backlight driver and configured it for running basic TTL display panels such as the ones we have in the shop



You can even power the display and decoder from a USB port. For example, with a 5" 800x480 display and 50mA backlight current, the current draw is 500mA total. You can reduce that down 370mA by running the backlight at half-brightness (25mA). With the backlight off, the decoder and display itself draws 250mA. If you want more backlight control, there's a PWM input, connect that to your microcontroller or other PWM output and you can continuously dim the backlight as desired

We have two versions, one is video only and one is video+touch. If you want a screen that you can poke at, get the +touch version and pair it with a screen that has a resistive touch overlay. The USB port then acts as both power and data, with the touch screen appearing like a USB mouse.

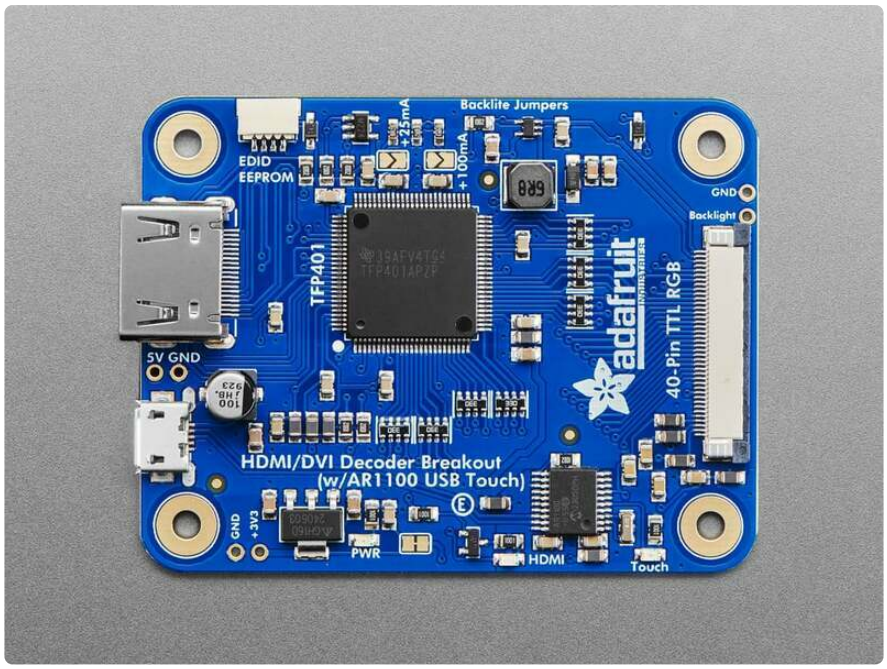
This driver is designed specifically as a small and easy to use display driver for our 40-pin TTL displays. In particular, we suggest it for use with single board computers (or desktop/laptops!) with DVI/HDMI output like the Raspberry Pi or BeagleBone Black. You can power the driver over USB and then feed it video via the HDMI port. It's a very small board so great for tucking into an enclosure. It can drive our 4.3", 5.0" or 7.0" displays but we really only recommend the 5" or 7" 800x480 as some computers do not like the low resolution of the 4.3" **and the TFP401 does not contain a video scaler, it will not resize/shrink video!**



We ship this board with an 800x480 resolution EDID so it will be auto-detected at that resolution. For advanced users, the EDID can be reprogrammed using our example Arduino code. Or, for computers that use linux, you can always just force the resolution to whatever display you have connected.

This is just a decoder breakout, a display is not included! We recommend either the 800x480 5" with touch (http://adafru.it/1596), 5" without touch (http://adafru.it/1680), 7" with touch (http://adafru.it/2354) or 7" without touch (http://adafru.it/2353)

Please check out the detailed tutorial on adjusting the backlight brightness. We also have information on how to tweak the EDID if you want to use other display resolutions. If you need a little more distance between the driver and display, check out the 40-pin FPC extension board. (http://adafru.it/2098)



Revision History:

- **As of February 7, 2024** - We updated this PCB with Adafruit Pinguin (https://adafru.it/11b7) to make a lovely and legible silkscreen, as well as with a plug-n-play Stemma QT connector (https://adafru.it/HMF) for the EDID EEPROM, in case you want to reprogram it.

Touch Screen

If you purchased the version of the decoder with touch support, you will receive a decoder board with extra circuitry for a resistive touch screen decoder. The circuit is an AR1100 USB resistive touch driver, so it basically just uses the same USB port you use to power, but for the data

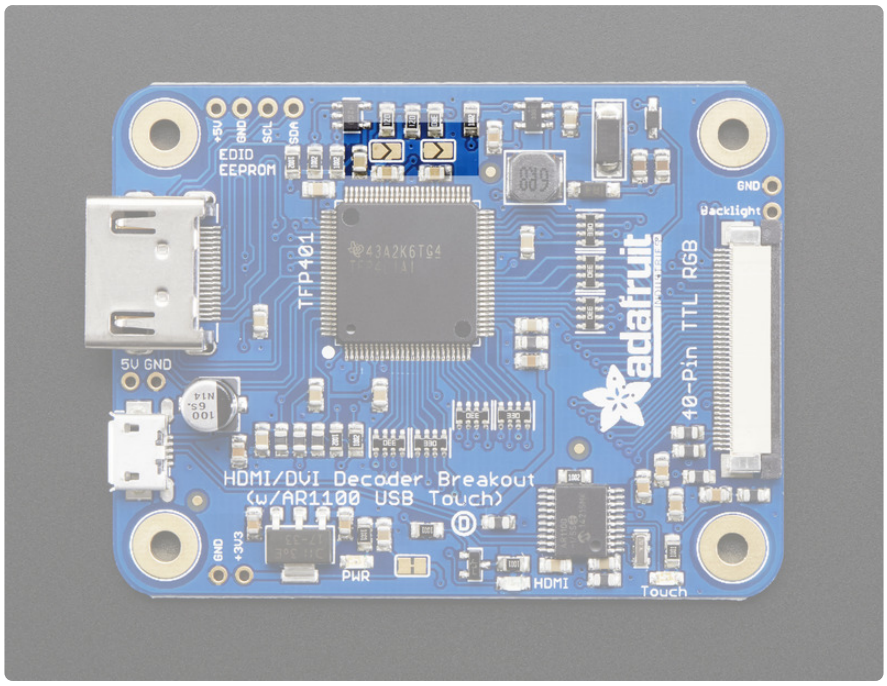
The AR1100 shows up like a USB mouse, it works on all operating systems as the computer doesn't even know its a touch screen, it just thinks its a mouse! You can adjust the AR1100 to consider itself a Touch Digitizer, supported by many but possibly not all computers.

You can also re-calibrate the touch screen. We do calibrate it for our 800x480 5" screens but we recommend re-calibrate it, especially if you are not using the exact same display we sell.

The software is Windows only, but you only have to configure/calibrate the touch controller once, then it can be used on any computer!

All of this happens with the AR1100 software. We have a tutorial specifically on using that software over here, so please go there (https://adafru.it/kQf) to learn all about it!

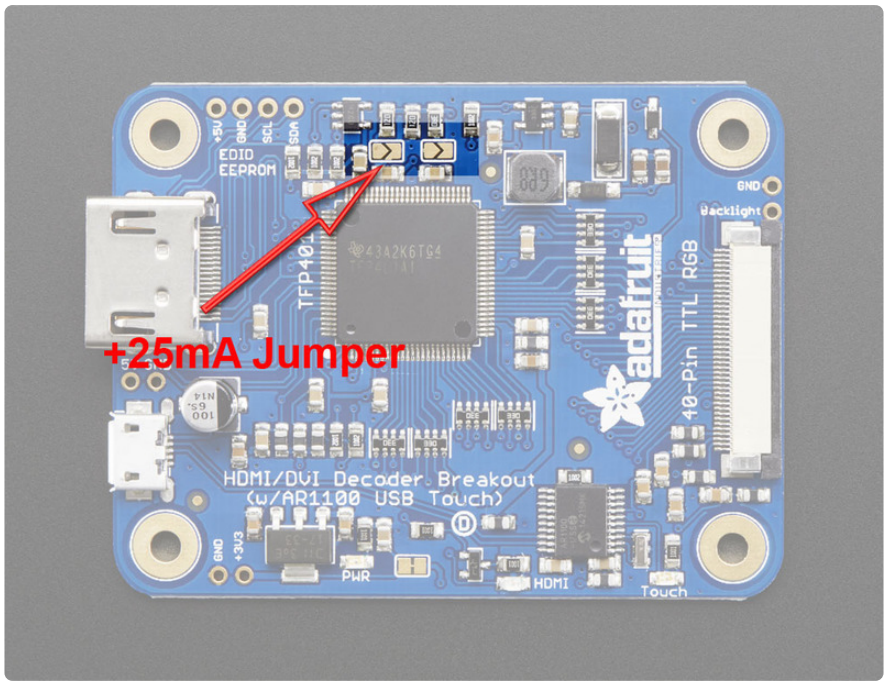
Backlight



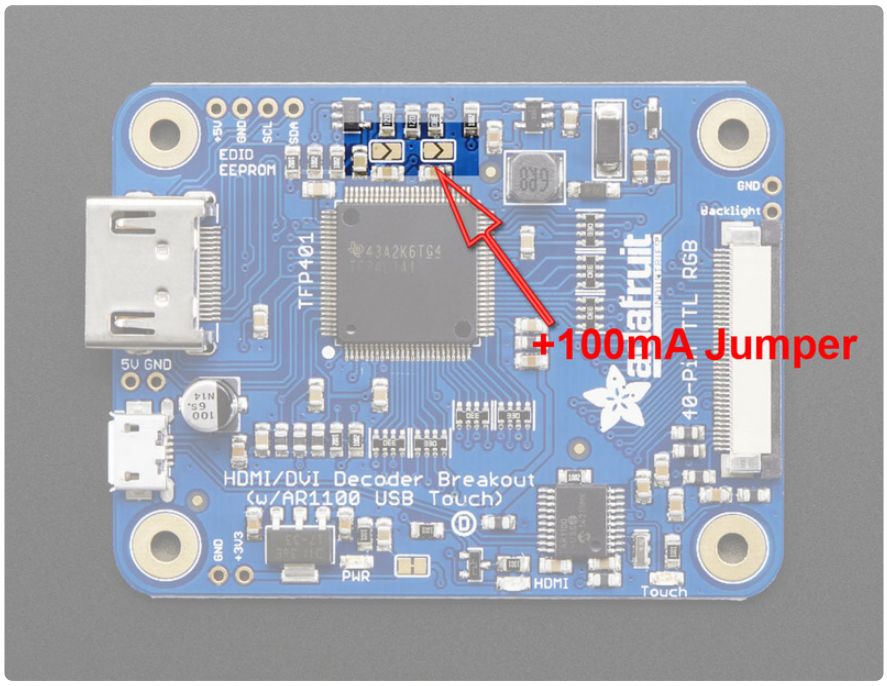
This is a generic TTL display driver, and each display has a slightly different backlight configuration. For that reason, you may need to make some adjustments to your board.

To make sure you don't accidentally damage your backlight, we make the default backlight current 25mA. Since the backlight driver is a constant-current boost, it will adjust the voltage up to 25V until it gets 25mA of draw.

If you are using a 4.3" diagonal screen, chances are its a 25mA backlight, so you should keep both jumpers clear



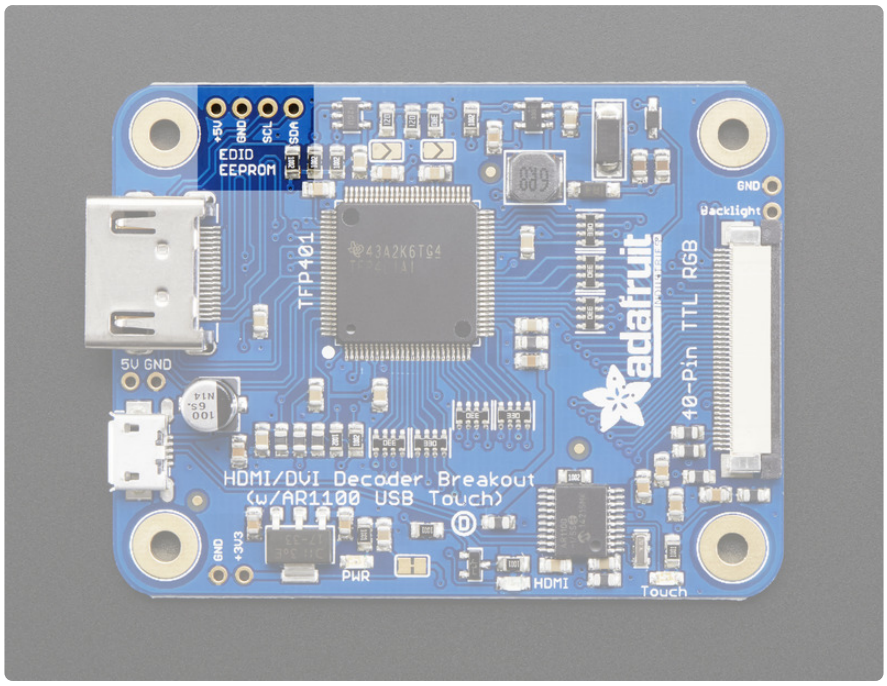
If you are using a 5.0" diagonal screen, chances are its a 50mA backlight. You can run it at 25mA but for the best look, close the +25mA jumper



If you are using a 7.0" diagonal screen, chances are its a 125mA backlight - the ones we sell in the Adafruit shop are 125mA. If not using an Adafruit screen **please check, as we have also seen some backlights that are 50mA!** You can run it at 25mA but it will be very dim. You should close the +100mA jumper

If you want to adjust the backlight brightness, you can feed a PWM signal (1KHz or greater) into the Backlight pin, 3-5V logic level. Or you can just connect it to ground to turn off the backlight driver. This will greatly reduce the power usage

Editing the EDID



One thing that often confuses people poking at DVI/HDMI signals is the EDID. The EDID is the 'device identifier data' that lets the computer know what kind of monitor is attached. To make it simple, the EDID is stored on an i2c EEPROM. If you reprogram the EEPROM, you've changed around the EDID.

The TFP401 video decoder chip never reads or writes the EEPROM/EDID, it has NO IDEA what is stored in the EDID!

Which sounds kinda odd - how does the TFP401 know what resolution to display then? The answer is that the **computer** defines the resolution of the monitor, and bases the decision on the EDID contents.

Usually the EDID tells the computer about half a dozen or so resolution options. However, with the TFP401, ther's only one resolution you should use: the native resolution of TTL display. That's because **the TFP401 does not contain a video scaler** - if you set the computer resolution to 800x480 while it is connected to a 1024x600 TTL display, you'll only get video in the top left corner. If you set the computer resolution to 1024x600 while it is connected to a 800x480 display, you'll get video that is cut off, and does not include the top right or bottom left sections.

So, basically, make sure the EDID contains the resolution you'll be connecting! We assume you'll be going with the ultra-common 800x480. You can reprogram the EDID

using an Arduino or (possibly) a computer using the HDMI/DVI port if you have software to write the EDID that way.

To reprogram the EEPROM, disconnect the HDMI connector, and connect

- 5V pin to Arduino 5V
- GND pin to Arduino GND
- SCL to Arduino SCL (A5 on an Arduino Uno)
- SDA to Arduino SDA (A4 on an Arduino Uno)

As of February 7, 2024, this board has been revised to include a plug-n-play Stemma QT connector (https://adafru.it/HMF) for the EDID EEPROM. If you have that version, you can use a STEMMA QT cable.

and use the following code to update the EDID:

```
#include &lt;Wire.h&gt; //I2C library


/* 480x272 @ 25mhz

uint8_t PROGMEM eepromdat[128] = {
0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x04, 0x81, 0x43, 0x00, 0x01, 0x00,
0x00, 0x00,
0x0C, 0x17, 0x01, 0x03, 0x81, 0x0A, 0x06, 0x78, 0x8A, 0xA5, 0x8E, 0xA6, 0x54, 0x4A,
0x9C, 0x26,
0x12, 0x45, 0x46, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01,
0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0xC4, 0x09, 0xE0, 0x33, 0x10, 0x10, 0x14, 0x10,
0x08, 0x05,
0x4A, 0x00, 0x5F, 0x36, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0xFC, 0x00, 0x41,
0x44, 0x41,
0x46, 0x52, 0x55, 0x49, 0x54, 0x20, 0x34, 0x33, 0x20, 0x20, 0x00, 0x00, 0x00, 0x10,
0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x10,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x42
};
*/
/* Adafruit breakout @ 800x480 ! */
uint8_t PROGMEM eepromdat[128] = {
0x00, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x00, 0x04, 0x81, 0x04, 0x00, 0x01, 0x00,
0x00, 0x00,
0x01, 0x11, 0x01, 0x03, 0x80, 0x0F, 0x0A, 0x00, 0x0A, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01, 0x01,
0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x80, 0x0C, 0x20, 0x80, 0x30, 0xE0, 0x2D, 0x10,
0x28, 0x30,
0xD3, 0x00, 0x6C, 0x44, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00,
0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10,
0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x10,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x17,
```

```
};

// I2C eeprom dumper
byte i2c_eeprom_read_byte(uint8_t deviceaddress, uint16_t eeaddress );

#define EEPROMSIZE 256UL  // 0.5 Kb
#define ADDRESS_SIZE 8

byte i2c_eeprom_read_byte(uint8_t deviceaddress, uint16_t eeaddress ) {
    byte rdata = 0xFF;
#if (ADDRESS_SIZE == 16)
    Wire.beginTransmission(deviceaddress);
    Wire.write((eeaddress &gt;&gt; 8)); // MSB
#else
    //deviceaddress |= (eeaddress &gt;&gt; 8);
    Wire.beginTransmission(deviceaddress); // MSB
#endif
    Wire.write(eeaddress); // LSB
    Wire.endTransmission();
    Wire.requestFrom(deviceaddress, (uint8_t)1);
    while (!Wire.available());
    rdata = Wire.read();
    return rdata;
}

void i2c_eeprom_write_byte(uint8_t deviceaddress, uint16_t eeaddress, byte data ) {
#if (ADDRESS_SIZE == 16)
    Wire.beginTransmission(deviceaddress);
    Wire.write((eeaddress &gt;&gt; 8)); // MSB
#else
    //deviceaddress |= (eeaddress &gt;&gt; 8);
    Wire.beginTransmission(deviceaddress); // MSB
#endif
    Wire.write((byte)eeaddress); // LSB
    Wire.write((byte)data);
    Wire.endTransmission();
}


void setup() {
  Wire.begin(); // initialise the connection
  Serial.begin(9600);
  Serial.println(F("EEPROM WRITER"));
  Serial.print(F("EEPROM data size: "));
  Serial.println(sizeof(eepromdat));
  Serial.println(F("Hit any key &amp; return to start"));
  while (!Serial.available());
  byte b;
  Serial.println("Starting");
  for (uint16_t addr = 0; addr &lt; EEPROMSIZE; addr++) {
    if (addr &lt; sizeof(eepromdat)) {
      b = pgm_read_byte(eepromdat+addr);
    } else {
      b = 0xFF;
    }
     i2c_eeprom_write_byte(0x50, addr, b);
     delay(5);
     if ((addr % 32) == 0)
       Serial.println();
     Serial.print("0x");
     if (b &lt; 0x10) Serial.print('0');
     Serial.print(b, HEX); //print content to serial port
     Serial.print(", ");
  }
  Serial.println("\n\r========\n\rFinished!");
  for (uint16_t addr = 0; addr &lt; EEPROMSIZE; addr++) {
    if (addr &lt; sizeof(eepromdat)) {
      b = pgm_read_byte(eepromdat+addr);
    } else {
```

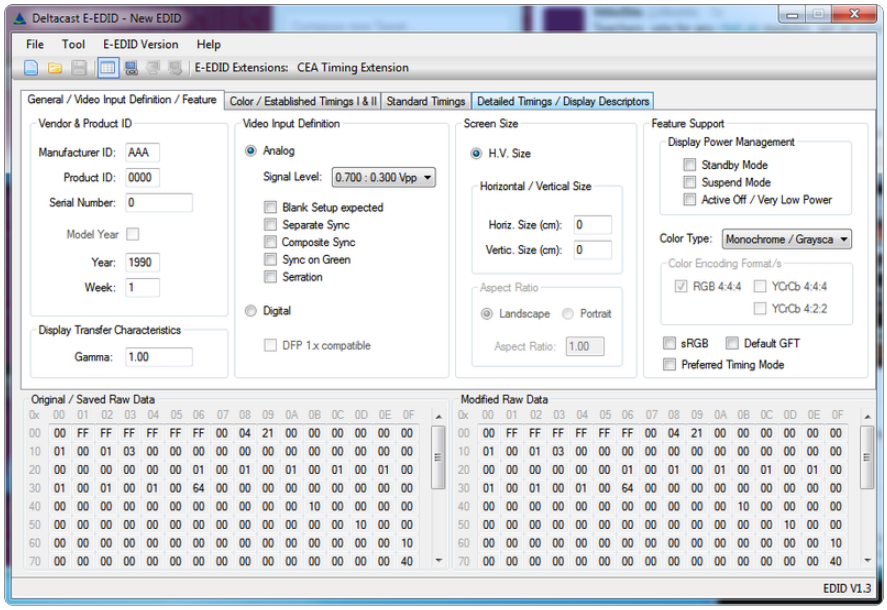
```
      b = 0xFF;
    }
    uint8_t d =  i2c_eeprom_read_byte(0x50, addr);
    if ((addr % 32) == 0)
      Serial.println();
    Serial.print("0x");
    if (d &lt; 0x10) Serial.print('0');
    Serial.print(d, HEX); //print content to serial port
    Serial.print(", ");

    if (b != d) {
       Serial.print(F("verification failed at 0x")); Serial.println(addr);
       while (1);
     }
  }
  Serial.println(F("\n\r\n\rVerified!"));
}

void loop() {
}
```

You can see we have an example for 480x272 there, for 4.3" screens. The reason we don't recommend it, tho, is that it overclocks the screens a bit and more-over, 480x272 is smaller than the 'minimal' 640x480 that most HDMI drivers really want. So during boot up or whatever, you won't get a display.

If you want to generate your own EDIDs and customize them, which is a great way to seriously lose like 4 hours of your life, you can download EDID editor software such as Deltacast or whatever you find when you google for "EDID editor" You only need the first 128 bytes (there are longer detailed EDID's that have an extra 128 byte 'chunk' but this decoder doesn't support all that stuff anyways)



While we dont think you could damage the TFP401 or display with a really messed up EDID, we still think you should only edit/customize the EDID if you're comfortable with some intense hex editing and EEPROM programming, its not for the faint of heart

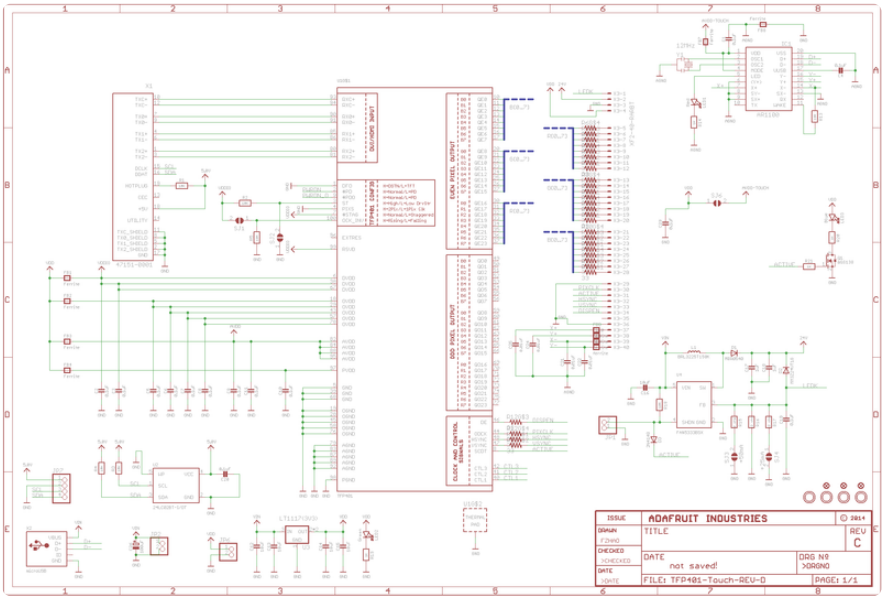
Downloads

Datasheets & Files

- TFP401A (https://adafru.it/e6U) - the DVI/HDMI decoder chip
- AR1100 (https://adafru.it/e6V) - the USB resistive touch chip
- FAN5333B (https://adafru.it/e6W) - the backlight driver
- EagleCAD PCB Files on GitHub (https://adafru.it/oDK)
- Fritzing Files in Adafruit Fritzing library (https://adafru.it/aP3)

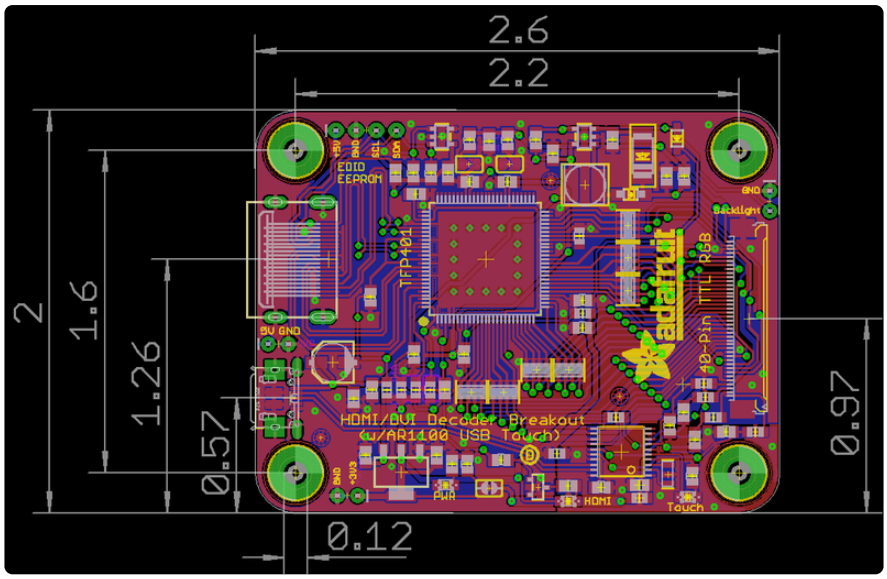
Schematics

The version without touch does not have the AR1100 circuitry in the top right corner



Fabrication Print

This is the same for both versions of the decoder, dims in inches



Raspberry Pi Config

This display has 800x480 pixels, and when used on Windows, at least, will autodetect and set the resolution. On Raspberry Pi, you're better off forcing the HDMI resolution by using the following config.txt file (in /boot/config.txt) - you can edit it by popping the SD card into your computer, the config.txt file is in the root directory

Remember, the TFP401 driver does not have a video scaler! If you don't feed it exactly 800x480 pixels the image will not stretch/shrink to fit!

```
# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

# uncomment this if your display has a black border of unused pixels visible
# and your display can output without overscan
#disable_overscan=1

# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's size minus
# overscan.
#framebuffer_width=1280
#framebuffer_height=720

# uncomment if hdmi display is not detected and composite is being output
hdmi_force_hotplug=1

# uncomment to force a specific HDMI mode (here we are forcing 800x480!)
hdmi_group=2
hdmi_mode=1
hdmi_mode=87
hdmi_cvt 800 480 60 6 0 0 0

# uncomment to force a HDMI mode rather than DVI. This can make audio work in
```

```
# DMT (computer monitor) modes
#hdmi_drive=2

# uncomment to increase signal to HDMI, if you have interference, blanking, or
# no display
#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
#arm_freq=800

# for more options see http://elinux.org/RPi_config.txt
```

It is possible to power the display from the onboard Pi USB port with the modification below, but a powered hub is ideal!

To let the Pi A+/B+ drive a display power over USB, first make sure you have a 2A power supply, with a good quality USB cable, a thin wire power cable is no good. Make sure its 24AWG or smaller, shorter USB cables are better too. Then add

```
max_usb_current=1
```

to **/boot/config.txt**

If you're getting wierd reboots, its likely the power supply and/or power USB cable is not good enough. A powered hub will also solve this problem