19Z604 - EMBEDDED SYSTEMS

NUMBER PLATE DETECTION SYSTEM

PSG COLLEGE OF TECHNOLOGY

TEAM - 11

Team Members

20Z204 - Adarsh G

20Z220 - Jeevan Krishna K V

20Z267 - Nirmal M

21Z431 - Ajay Deepak P M

INTRODUCTION

A number plate detection system, also known as a license plate recognition system or automatic number plate recognition (ANPR) is a device that extracts alphanumeric characters from license plates using image processing and pattern recognition algorithms. The technology works by utilizing a camera to take an image or video of a license plate from a moving vehicle, which is then processed to identify the characters on the plate. Parking management, toll collection, the recovery of stolen vehicles and the detection of traffic infractions are some of the applications for this technology. Due to the number plate detection system's excellent precision, speed and capacity to function in a variety of lighting and weather circumstances, it has grown in popularity over the past several years.

PROBLEM STATEMENT

The number plate detection system is designed to extract the license plate number from a vehicle image or video in real-time. The system must be precise, effective, and able to function in a variety of lighting and weather situations. In spite of changes in font, size, and orientation, the system ought to be able to read the characters on the license plate. The system's goal is to help law enforcement organizations find automobiles that break traffic laws, parking rules, etc. The system can also be used for parking management, toll collecting, and restricted area access control. The primary difficulty is to create a reliable algorithm that can quickly and reliably detect and identify number plates from various types of automobiles.

PURPOSE

The purpose of a number plate detection system is to automatically detect and recognise the license plates of vehicles. This system uses image processing techniques to capture an image of a vehicle's license plate, isolate the characters on the plate, and convert them into a text string that can be used for various purposes.

- Law Enforcers: Law enforcement agencies can utilize the number plate detection system to look for stolen cars, enforce parking laws and keep track of traffic infractions.
- Toll Collection: The system can be used to collect tolls on bridges, tunnels and highways. The system is able to automatically deduct the correct toll fee from the driver's account by reading the license plate of a car as it passes through a toll booth.

- Security: The number plate detection system can be employed for security reasons such as monitoring vehicle traffic in critical places like airports, military facilities and governmental structures.

PRODUCT FEATURES

The product features of a number plate detection system can vary depending on the specific product and its intended application. Some of the common features of our number plate detection system includes

- Automatic License Plate Recognition (ALPR): The system is capable of automatically recognising license plates using image processing techniques.
- High Accuracy: The system has a high accuracy rate in recognising license plates in various lighting and weather conditions.
- Real-time Processing: The system can process license plate data in real-time that allows applications like toll collecting, parking management and law enforcement to take rapid action.
- Security: The system has adequate security measures to protect the data it collects using encryption and access control.
- Customisation: The system has various customization options for various applications, including support for various languages, license plate forms and legal requirements.

HARDWARE DESIGN

The hardware design of a number plate detection system using Arduino and OV7670 involves the integration of the OV7670 camera module and an Arduino microcontroller board. The system is designed to capture an image of a vehicle's number plate, process the image to extract the number plate text, and display the text on an LCD screen or transmit it wirelessly to a remote computer.

The cameras used in a number plate detection system are typically high-resolution digital cameras that are designed to capture clear images of license plates. The camera used in our project is OV7670. The OV7670 is a popular low-cost image sensor camera module that is commonly used in embedded systems, robotics, and other electronic projects. The OV7670 camera module features a 1/6-inch CMOS image sensor with a maximum resolution of 640x480 pixels (VGA) and supports a variety of image formats including RAW RGB, YUV (4:2:2) and YCbCr (4:2:2). It has a 10-bit parallel DVP interface that allows for fast data transfer and is capable of capturing 30 frames per second at full VGA resolution.

The microcontroller is the heart of the number plate detection system's hardware design. The microcontroller used in this project is Arduino. The Arduino board is based on the Atmel AVR microcontroller, which is a low-power, high-performance chip that can be programmed using the Arduino Integrated Development Environment (IDE). The IDE is a software tool that allows users to write and upload code to the microcontroller using a USB connection.

In addition to these primary components, the hardware design of a number plate detection system can also include specialized lighting systems, weatherproof enclosures and communication modules to improve the performance of the system and to communicate with other devices and systems.

IMPLEMENTATION

The implementation of a number plate detection system using Arduino and OV7670 involves several steps including System design, programming the Arduino board and testing.

The first stage of implementing a number plate detection system is system design. Setting up the Hardware - The first step in the implementation of the system is to connect the OV7670 camera module and the LCD screen to the Arduino board. The 10-bit parallel DVP interface is used to link the camera module to the Arduino board.

The second stage of implementation involves programming the Arduino board. The next step is to program the Arduino board to control the OV7670 camera module, process the image to extract the number plate text, and transmit the text to the system. The Arduino IDE, a software package that enables users to create and upload code to the microcontroller using a USB connection, can be used for programming.

Testing the System - Once the hardware is set up and the programming is done, the system can be tested by capturing an image of a vehicle's number plate and processing the image to extract the number plate text. The recovered text can be seen on the computer.

Refining the System - After testing the system, it may be necessary to refine the algorithms used to process the image and extract the number plate text. This can be accomplished by changing the algorithms' parameters and retesting the system.

In conclusion, the implementation of a number plate detection system is a complex process that involves several stages, including system design, installation and configuration, testing and optimization, and deployment.

CODING

/Code for Image Processing

/Install and import dependencies

```
!pip install easyocr
!pip install imutils
import cv2
from matplotlib import pyplot as plt
import numpy as np
import imutils
import easyocr
```

/Read the image in grayscale and blurred format

```
img = cv2.imread('image4.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(cv2.cvtColor(gray, cv2.COLOR_BGR2RGB))
```

/Apply filter and find the edges for localization

```
bfilter = cv2.bilateralFilter(gray, 11, 17, 17)
edged = cv2.Canny(bfilter, 30, 200)
plt.imshow(cv2.cvtColor(edged, cv2.COLOR_BGR2RGB))
```

/Find the contours and apply mask

```
Keypoints = cv2.findContours(edged.copy(), cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(keypoints)
contours = sorted(contours, key=cv2.contourArea, reverse=True)[:10]
location = None
for contour in contours:
    approx = cv2.approxPolyDP(contour, 10, True)
    if len(approx) == 4:
        location = approx
        break
```

```
location
mask = np.zeros(gray.shape, np.uint8)
new_image = cv2.drawContours(mask, [location], 0,255, -1)
new_image = cv2.bitwise_and(img, img, mask=mask)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
(x,y) = np.where(mask==255)
(x1, y1) = (np.min(x), np.min(y))
(x2, y2) = (np.max(x), np.max(y))
cropped_image = gray[x1:x2+1, y1:y2+1]
plt.imshow(cv2.cvtColor(cropped_image, cv2.COLOR_BGR2RGB))
```

/Using EasyOCR to read text

```
reader = easyocr.Reader(['en'])
result = reader.readtext(cropped_image)
result
```

/Render result

```
text = result[0][-2]
font = cv2.FONT_HERSHEY_SIMPLEX
res = cv2.putText(img, text=text, org=(approx[0][0][0], approx[1][0][1]+60),
fontFace=font, fontScale=1, color=(0,255,0), thickness=2, lineType=cv2.LINE_AA)
res = cv2.rectangle(img, tuple(approx[0][0]), tuple(approx[2][0]), (0,255,0),3)
plt.imshow(cv2.cvtColor(res, cv2.COLOR_BGR2RGB))
```

/Printing the result

```
Result[0][-2]
```

/Data is stored in CSV file

```
raw_data = {'date': [time.asctime( time.localtime(time.time()) )],
    'v_number': [result[0][-2]]}
df = pd.DataFrame(raw_data, columns = ['date', 'v_number'])
df.to_csv('data.csv')
```

/Downloading the CSV file

```
from google.colab import files
```

```
files.download('data.csv')
```

/Code of Arduino

```
#include "setup.h"
void setup() {
  CLKPR = 0x80; // enter clock rate change mode
  CLKPR = 0; // set prescaler to 0. WAVGAT MCU has it 3 by default.
  initializeScreenAndCamera();
}

void loop() {
  processFrame();
}

/**************************************************
  This is a library for the Adafruit 1.8" SPI display.

This library works with the Adafruit 1.8" TFT Breakout w/SD card
  ----> http://www.adafruit.com/products/358
The 1.8" TFT shield
  ----> https://www.adafruit.com/product/802
The 1.44" TFT breakout
  ----> https://www.adafruit.com/product/2088
as well as Adafruit raw 1.8" TFT display
  ----> http://www.adafruit.com/products/618

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!
  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/

#include "Adafruit_ST7735_mod.h"
#include <limits.h>
#include "pins_arduino.h"
#include "wiring_private.h"
#include <SPI.h>

inline uint16_t swapcolor(uint16_t x) {
  return (x << 11) | (x & 0x07E0) | (x >> 11);
}

#if defined (SPI_HAS_TRANSACTION)
  static SPISettings mySPISettings;
#elif defined (__AVR__)
  static uint8_t SPCRbackup;
  static uint8_t mySPCR;
#endif



// Constructor when using software SPI.  All output pins are configurable.
Adafruit_ST7735_mod::Adafruit_ST7735_mod(int8_t cs, int8_t rs, int8_t sid, int8_t sclk, int8_t rst)
  : Adafruit_GFX(ST7735_TFTWIDTH, ST7735_TFTHEIGHT_18)
{
  _cs   = cs;
  _rs   = rs;
  _sid  = sid;
  _sclk = sclk;
  _rst  = rst;
  hwSPI = false;
}
```

```
// Constructor when using hardware SPI.  Faster, but must use SPI pins
// specific to each board type (e.g. 11,13 for Uno, 51,52 for Mega, etc.)
Adafruit_ST7735_mod::Adafruit_ST7735_mod(int8_t cs, int8_t rs, int8_t rst)
  : Adafruit_GFX(ST7735_TFTWIDTH, ST7735_TFTHEIGHT_18) {
  _cs   = cs;
  _rs   = rs;
  _rst  = rst;
  hwSPI = true;
  _sid  = _sclk = 0;
}

#if defined(CORE_TEENSY) && !defined(__AVR__)
#define __AVR__
#endif

inline void Adafruit_ST7735_mod::spiwrite(uint8_t c) {

  //Serial.println(c, HEX);

  if (hwSPI) {
#if defined (SPI_HAS_TRANSACTION)
      SPI.transfer(c);
#elif defined (__AVR__)
      SPCRbackup = SPCR;
      SPCR = mySPCR;
      SPI.transfer(c);
      SPCR = SPCRbackup;
//      SPDR = c;
//      while(!(SPSR & _BV(SPIF)));
#elif defined (__arm__)
      SPI.setClockDivider(21); //4MHz
      SPI.setDataMode(SPI_MODE0);
      SPI.transfer(c);
#endif
  } else {
    // Fast SPI bitbang swiped from LPD8806 library
    for(uint8_t bit = 0x80; bit; bit >>= 1) {
      if(c & bit) *dataport |=  datapinmask;
      else        *dataport &= ~datapinmask;
      *clkport |=  clkpinmask;
      *clkport &= ~clkpinmask;
    }
  }

  // Add a little delay so it will work with WAVGAT MCU
  // For some reason with WAVGAT Nano SPI.transfer(c) doesn't wait until byte is sent
  __asm__("nop");
  __asm__("nop");
  __asm__("nop");
}



void Adafruit_ST7735_mod::writecommand(uint8_t c) {
#if defined (SPI_HAS_TRANSACTION)
  SPI.beginTransaction(mySPISettings);
#endif
  *rsport &= ~rspinmask;
  *csport &= ~cspinmask;

  //Serial.print("C ");
  spiwrite(c);

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}



void Adafruit_ST7735_mod::writedata(uint8_t c) {
#if defined (SPI_HAS_TRANSACTION)
```

```
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;

  //Serial.print("D ");
  spiwrite(c);

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}

// Rather than a bazillion writecommand() and writedata() calls, screen
// initialization commands and arguments are organized in these tables
// stored in PROGMEM.  The table may look bulky, but that's mostly the
// formatting -- storage-wise this is hundreds of bytes more compact
// than the equivalent code.  Companion function follows.
#define DELAY 0x80
static const uint8_t PROGMEM
  Bcmd[] = {                  // Initialization commands for 7735B screens
    18,                       // 18 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, no args, w/delay
      50,                     //     50 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, no args, w/delay
      255,                    //     255 = 500 ms delay
    ST7735_COLMOD , 1+DELAY,  //  3: Set color mode, 1 arg + delay:
      0x05,                   //     16-bit color
      10,                     //     10 ms delay
    ST7735_FRMCTR1, 3+DELAY,  //  4: Frame rate control, 3 args + delay:
      0x00,                   //     fastest refresh
      0x06,                   //     6 lines front porch
      0x03,                   //     3 lines back porch
      10,                     //     10 ms delay
    ST7735_MADCTL , 1      ,  //  5: Memory access ctrl (directions), 1 arg:
      0x08,                   //     Row addr/col addr, bottom to top refresh
```

```
ST7735_DISSET5, 2      ,  //  6: Display settings #5, 2 args, no delay:
  0x15,                  //     1 clk cycle nonoverlap, 2 cycle gate
                        //     rise, 3 cycle osc equalize
  0x02,                  //     Fix on VTL
ST7735_INVCTR , 1      ,  //  7: Display inversion control, 1 arg:
  0x0,                   //     Line inversion
ST7735_PWCTR1 , 2+DELAY,  //  8: Power control, 2 args + delay:
  0x02,                  //     GVDD = 4.7V
  0x70,                  //     1.0uA
  10,                    //     10 ms delay
ST7735_PWCTR2 , 1      ,  //  9: Power control, 1 arg, no delay:
  0x05,                  //     VGH = 14.7V, VGL = -7.35V
ST7735_PWCTR3 , 2      ,  // 10: Power control, 2 args, no delay:
  0x01,                  //     Opamp current small
  0x02,                  //     Boost frequency
ST7735_VMCTR1 , 2+DELAY,  // 11: Power control, 2 args + delay:
  0x3C,                  //     VCOMH = 4V
  0x38,                  //     VCOML = -1.1V
  10,                    //     10 ms delay
ST7735_PWCTR6 , 2      ,  // 12: Power control, 2 args, no delay:
  0x11, 0x15,
ST7735_GMCTRP1,16      ,  // 13: Magical unicorn dust, 16 args, no delay:
  0x09, 0x16, 0x09, 0x20, //     (seriously though, not sure what
  0x21, 0x1B, 0x13, 0x19, //      these config values represent)
  0x17, 0x15, 0x1E, 0x2B,
  0x04, 0x05, 0x02, 0x0E,
ST7735_GMCTRN1,16+DELAY,  // 14: Sparkles and rainbows, 16 args + delay:
  0x0B, 0x14, 0x08, 0x1E, //     (ditto)
  0x22, 0x1D, 0x18, 0x1E,
  0x1B, 0x1A, 0x24, 0x2B,
  0x06, 0x06, 0x02, 0x0F,
  10,                    //     10 ms delay
ST7735_CASET  , 4      ,  // 15: Column addr set, 4 args, no delay:
  0x00, 0x02,            //     XSTART = 2
  0x00, 0x81,            //     XEND = 129
ST7735_RASET  , 4      ,  // 16: Row addr set, 4 args, no delay:
  0x00, 0x02,            //     XSTART = 1
  0x00, 0x81,            //     XEND = 160
  ST7735_NORON  ,   DELAY,  // 17: Normal display on, no args, w/delay
    10,                     //     10 ms delay
  ST7735_DISPON ,   DELAY,  // 18: Main screen turn on, no args, w/delay
    255 },                  //     255 = 500 ms delay

Rcmd1[] = {                 // Init for 7735R, part 1 (red or green tab)
  15,                       // 15 commands in list:
  ST7735_SWRESET,   DELAY,  //  1: Software reset, 0 args, w/delay
    150,                    //     150 ms delay
  ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, 0 args, w/delay
    255,                    //     500 ms delay
  ST7735_FRMCTR1, 3      ,  //  3: Frame rate ctrl - normal mode, 3 args:
    0x01, 0x2C, 0x2D,       //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
  ST7735_FRMCTR2, 3      ,  //  4: Frame rate control - idle mode, 3 args:
    0x01, 0x2C, 0x2D,       //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
  ST7735_FRMCTR3, 6      ,  //  5: Frame rate ctrl - partial mode, 6 args:
    0x01, 0x2C, 0x2D,       //     Dot inversion mode
    0x01, 0x2C, 0x2D,       //     Line inversion mode
  ST7735_INVCTR , 1      ,  //  6: Display inversion ctrl, 1 arg, no delay:
    0x07,                   //     No inversion
  ST7735_PWCTR1 , 3      ,  //  7: Power control, 3 args, no delay:
    0xA2,
    0x02,                   //     -4.6V
    0x84,                   //     AUTO mode
  ST7735_PWCTR2 , 1      ,  //  8: Power control, 1 arg, no delay:
    0xC5,                   //     VGH25 = 2.4C VGSEL = -10 VGH = 3 * AVDD
  ST7735_PWCTR3 , 2      ,  //  9: Power control, 2 args, no delay:
    0x0A,                   //     Opamp current small
    0x00,                   //     Boost frequency
  ST7735_PWCTR4 , 2      ,  // 10: Power control, 2 args, no delay:
    0x8A,                   //     BCLK/2, Opamp current small & Medium low
    0x2A,
  ST7735_PWCTR5 , 2      ,  // 11: Power control, 2 args, no delay:
    0x8A, 0xEE,
  ST7735_VMCTR1 , 1      ,  // 12: Power control, 1 arg, no delay:
    0x0E,
  ST7735_INVOFF , 0      ,  // 13: Don't invert display, no args, no delay
  ST7735_MADCTL , 1      ,  // 14: Memory access control (directions), 1 arg:
    0xC8,                 //     row addr/col addr, bottom to top refresh
  ST7735_COLMOD , 1      ,  // 15: set color mode, 1 arg, no delay:
    0x05 },               //     16-bit color

Rcmd2green[] = {            // Init for 7735R, part 2 (green tab only)
  2,                        //  2 commands in list:
  ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
    0x00, 0x02,             //     XSTART = 0
    0x00, 0x7F+0x02,        //     XEND = 127
  ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
    0x00, 0x01,             //     XSTART = 0
    0x00, 0x9F+0x01 },      //     XEND = 159
Rcmd2red[] = {              // Init for 7735R, part 2 (red tab only)
  2,                        //  2 commands in list:
  ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
    0x00, 0x00,             //     XSTART = 0
    0x00, 0x7F,             //     XEND = 127
  ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
    0x00, 0x00,             //     XSTART = 0
    0x00, 0x9F },           //     XEND = 159

Rcmd2green144[] = {             // Init for 7735R, part 2 (green 1.44 tab)
  2,                        //  2 commands in list:
  ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
    0x00, 0x00,             //     XSTART = 0
    0x00, 0x7F,             //     XEND = 127
  ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
    0x00, 0x00,             //     XSTART = 0
    0x00, 0x7F },           //     XEND = 127

Rcmd3[] = {                 // Init for 7735R, part 3 (red or green tab)
  4,                        //  4 commands in list:
  ST7735_GMCTRP1, 16      , //  1: Magical unicorn dust, 16 args, no delay:
    0x02, 0x1c, 0x07, 0x12,
    0x37, 0x32, 0x29, 0x2d,
    0x29, 0x25, 0x2B, 0x39,
    0x00, 0x01, 0x03, 0x10,
  ST7735_GMCTRN1, 16      , //  2: Sparkles and rainbows, 16 args, no delay:
    0x03, 0x1d, 0x07, 0x06,
    0x2E, 0x2C, 0x29, 0x2D,
    0x2E, 0x2E, 0x37, 0x3F,
    0x00, 0x00, 0x02, 0x10,
  ST7735_NORON  ,    DELAY, //  3: Normal display on, no args, w/delay
    10,                     //     10 ms delay
  ST7735_DISPON ,    DELAY, //  4: Main screen turn on, no args w/delay
    100 };                  //     100 ms delay


// Companion code to the above tables.  Reads and issues
// a series of LCD commands stored in PROGMEM byte array.
void Adafruit_ST7735_mod::commandList(const uint8_t *addr) {

  uint8_t  numCommands, numArgs;
  uint16_t ms;

  numCommands = pgm_read_byte(addr++);   // Number of commands to follow
  while(numCommands--) {                 // For each command...
    writecommand(pgm_read_byte(addr++)); //   Read, issue command
    numArgs  = pgm_read_byte(addr++);    //   Number of args to follow
    ms       = numArgs & DELAY;          //   If hibit set, delay follows args
    numArgs &= ~DELAY;                   //   Mask out delay bit
    while(numArgs--) {                   //   For each argument...
      writedata(pgm_read_byte(addr++));  //     Read, issue argument
    }

    if(ms) {
      ms = pgm_read_byte(addr++); // Read post-command delay time (ms)
      if(ms == 255) ms = 500;     // If 255, delay for 500 ms
      delay(ms);
    }
  }
}
```

```
// Initialization code common to both 'B' and 'R' type displays
void Adafruit_ST7735_mod::commonInit(const uint8_t *cmdList) {
  colstart  = rowstart = 0; // May be overridden in init func

  pinMode(_rs, OUTPUT);
  pinMode(_cs, OUTPUT);
  csport    = portOutputRegister(digitalPinToPort(_cs));
  rsport    = portOutputRegister(digitalPinToPort(_rs));
  cspinmask = digitalPinToBitMask(_cs);
  rspinmask = digitalPinToBitMask(_rs);

  if(hwSPI) { // Using hardware SPI
#if defined (SPI_HAS_TRANSACTION)
    SPI.begin();
    mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);
#elif defined (__AVR__)
    SPCRbackup = SPCR;
    SPI.begin();
    SPI.setClockDivider(SPI_CLOCK_DIV4);
    SPI.setDataMode(SPI_MODE0);
    mySPCR = SPCR; // save our preferred state
    //Serial.print("mySPCR = 0x"); Serial.println(SPCR, HEX);
    SPCR = SPCRbackup;  // then restore
#elif defined (__SAM3X8E__)
    SPI.begin();
    SPI.setClockDivider(21); //4MHz
    SPI.setDataMode(SPI_MODE0);
#endif
  } else {
    pinMode(_sclk, OUTPUT);
    pinMode(_sid , OUTPUT);
    clkport     = portOutputRegister(digitalPinToPort(_sclk));
    dataport    = portOutputRegister(digitalPinToPort(_sid));
    clkpinmask  = digitalPinToBitMask(_sclk);
    datapinmask = digitalPinToBitMask(_sid);
    *clkport   &= ~clkpinmask;
    *dataport  &= ~datapinmask;
  }
  // toggle RST low to reset; CS low so it'll listen to us
  *csport &= ~cspinmask;
  if (_rst) {
    pinMode(_rst, OUTPUT);
    digitalWrite(_rst, HIGH);
    delay(500);
    digitalWrite(_rst, LOW);
    delay(500);
    digitalWrite(_rst, HIGH);
    delay(500);
  }
  if(cmdList) commandList(cmdList);
}
// Initialization for ST7735B screens
void Adafruit_ST7735_mod::initB(void) {
  commonInit(Bcmd);
}
// Initialization for ST7735R screens (green or red tabs)
void Adafruit_ST7735_mod::initR(uint8_t options) {
  commonInit(Rcmd1);
  if(options == INITR_GREENTAB) {
    commandList(Rcmd2green);
    colstart = 2;
    rowstart = 1;
  } else if(options == INITR_144GREENTAB) {
    _height = ST7735_TFTHEIGHT_144;
    commandList(Rcmd2green144);
    colstart = 2;
    rowstart = 3;
  } else {
    // colstart, rowstart left at default '0' values
    commandList(Rcmd2red);
  }
  commandList(Rcmd3);

  // if black, change MADCTL color filter
  if (options == INITR_BLACKTAB) {
    writecommand(ST7735_MADCTL);
    writedata(0xC0);
  }

  tabcolor = options;
}

void Adafruit_ST7735_mod::setAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1) {

  writecommand(ST7735_CASET); // Column addr set
  writedata(0x00);
  writedata(x0+colstart);     // XSTART
  writedata(0x00);
  writedata(x1+colstart);     // XEND
  writecommand(ST7735_RASET); // Row addr set
  writedata(0x00);
  writedata(y0+rowstart);     // YSTART
  writedata(0x00);
  writedata(y1+rowstart);     // YEND

  writecommand(ST7735_RAMWR); // write to RAM
}
void Adafruit_ST7735_mod::startAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1) {
  setAddrWindow(x0, y0, x1, y1);
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
}



void Adafruit_ST7735_mod::endAddrWindow() {
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
```

```
    SPI.endTransaction();
#endif
}



void Adafruit_ST7735_mod::pushColor(uint16_t color) {
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;

  spiwrite(color >> 8);
  spiwrite(color);

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
void Adafruit_ST7735_mod::drawPixel(int16_t x, int16_t y, uint16_t color) {
  if((x < 0) ||(x >= _width) || (y < 0) || (y >= _height)) return;
  setAddrWindow(x,y,x+1,y+1);
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  spiwrite(color >> 8);
  spiwrite(color);

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
```

```
void Adafruit_ST7735_mod::drawFastVLine(int16_t x, int16_t y, int16_t h,
 uint16_t color) {
  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((y+h-1) >= _height) h = _height-y;
  setAddrWindow(x, y, x, y+h-1);
  uint8_t hi = color >> 8, lo = color;
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  while (h--) {
    spiwrite(hi);
    spiwrite(lo);
  }
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}



void Adafruit_ST7735_mod::drawFastHLine(int16_t x, int16_t y, int16_t w,
  uint16_t color) {

  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((x+w-1) >= _width)  w = _width-x;
  setAddrWindow(x, y, x+w-1, y);
  uint8_t hi = color >> 8, lo = color;
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  while (w--) {
    spiwrite(hi);
    spiwrite(lo);
  }
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
void Adafruit_ST7735_mod::fillScreen(uint16_t color) {
  fillRect(0, 0,  _width, _height, color);
}




// fill a rectangle
void Adafruit_ST7735_mod::fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
  uint16_t color) {

  // rudimentary clipping (drawChar w/big text requires this)
  if((x >= _width) || (y >= _height)) return;
  if((x + w - 1) >= _width)  w = _width  - x;
  if((y + h - 1) >= _height) h = _height - y;

  setAddrWindow(x, y, x+w-1, y+h-1);

  uint8_t hi = color >> 8, lo = color;

#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  for(y=h; y>0; y--) {
    for(x=w; x>0; x--) {
      spiwrite(hi);
      spiwrite(lo);
    }
  }

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}



// Pass 8-bit (each) R,G,B, get back 16-bit packed color
uint16_t Adafruit_ST7735_mod::Color565(uint8_t r, uint8_t g, uint8_t b) {
  return ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | (b >> 3);
}



#define MADCTL_MY  0x80
#define MADCTL_MX  0x40
#define MADCTL_MV  0x20
#define MADCTL_ML  0x10
#define MADCTL_RGB 0x00
#define MADCTL_BGR 0x08
#define MADCTL_MH  0x04

void Adafruit_ST7735_mod::setRotation(uint8_t m) {

  writecommand(ST7735_MADCTL);
  rotation = m % 4; // can't be higher than 3
  switch (rotation) {
   case 0:
     if (tabcolor == INITR_BLACKTAB) {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_RGB);
     } else {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_BGR);
     }
     _width  = ST7735_TFTWIDTH;
```

```
       if (tabcolor == INITR_144GREENTAB)
         _height = ST7735_TFTHEIGHT_144;
       else
         _height = ST7735_TFTHEIGHT_18;

       break;
     case 1:
       if (tabcolor == INITR_BLACKTAB) {
         writedata(MADCTL_MY | MADCTL_MV | MADCTL_RGB);
       } else {
         writedata(MADCTL_MY | MADCTL_MV | MADCTL_BGR);
       }

       if (tabcolor == INITR_144GREENTAB)
         _width = ST7735_TFTHEIGHT_144;
       else
         _width = ST7735_TFTHEIGHT_18;

       _height = ST7735_TFTWIDTH;
       break;
    case 2:
       if (tabcolor == INITR_BLACKTAB) {
         writedata(MADCTL_RGB);
       } else {
         writedata(MADCTL_BGR);
       }
       _width  = ST7735_TFTWIDTH;
       if (tabcolor == INITR_144GREENTAB)
         _height = ST7735_TFTHEIGHT_144;
       else
         _height = ST7735_TFTHEIGHT_18;

      break;
     case 3:
       if (tabcolor == INITR_BLACKTAB) {
```

```
      writedata(MADCTL_MX | MADCTL_MV | MADCTL_RGB);
    } else {
      writedata(MADCTL_MX | MADCTL_MV | MADCTL_BGR);
    }
    if (tabcolor == INITR_144GREENTAB)
      _width = ST7735_TFTHEIGHT_144;
    else
      _width = ST7735_TFTHEIGHT_18;

    _height = ST7735_TFTWIDTH;
    break;
  }
}



void Adafruit_ST7735_mod::invertDisplay(boolean i) {
  writecommand(i ? ST7735_INVON : ST7735_INVOFF);
}/***************************************************
  This is a library for the Adafruit 1.8" SPI display.

This library works with the Adafruit 1.8" TFT Breakout w/SD card
  ----> http://www.adafruit.com/products/358
The 1.8" TFT shield
  ----> https://www.adafruit.com/product/802
The 1.44" TFT breakout
  ----> https://www.adafruit.com/product/2088
as well as Adafruit raw 1.8" TFT display
  ----> http://www.adafruit.com/products/618

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!
  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/
#include "Adafruit_ST7735_mod.h"
#include <limits.h>
#include "pins_arduino.h"
#include "wiring_private.h"
#include <SPI.h>

inline uint16_t swapcolor(uint16_t x) {
  return (x << 11) | (x & 0x07E0) | (x >> 11);
}

#if defined (SPI_HAS_TRANSACTION)
  static SPISettings mySPISettings;
#elif defined (__AVR__)
  static uint8_t SPCRbackup;
  static uint8_t mySPCR;
#endif
// Constructor when using software SPI.  All output pins are configurable.
Adafruit_ST7735_mod::Adafruit_ST7735_mod(int8_t cs, int8_t rs, int8_t sid, int8_t sclk, int8_t
rst)
  : Adafruit_GFX(ST7735_TFTWIDTH, ST7735_TFTHEIGHT_18)
{
  _cs   = cs;
  _rs   = rs;
  _sid  = sid;
  _sclk = sclk;
  _rst  = rst;
  hwSPI = false;
}
// Constructor when using hardware SPI.  Faster, but must use SPI pins
// specific to each board type (e.g. 11,13 for Uno, 51,52 for Mega, etc.)
Adafruit_ST7735_mod::Adafruit_ST7735_mod(int8_t cs, int8_t rs, int8_t rst)
  : Adafruit_GFX(ST7735_TFTWIDTH, ST7735_TFTHEIGHT_18) {
  _cs   = cs;
  _rs   = rs;
  _rst  = rst;
  hwSPI = true;
  _sid  = _sclk = 0;
}

#if defined(CORE_TEENSY) && !defined(__AVR__)
#define __AVR__
#endif

inline void Adafruit_ST7735_mod::spiwrite(uint8_t c) {

  //Serial.println(c, HEX);

  if (hwSPI) {
#if defined (SPI_HAS_TRANSACTION)
      SPI.transfer(c);
#elif defined (__AVR__)
      SPCRbackup = SPCR;
      SPCR = mySPCR;
      SPI.transfer(c);
      SPCR = SPCRbackup;
//      SPDR = c;
//      while(!(SPSR & _BV(SPIF)));
#elif defined (__arm__)
      SPI.setClockDivider(21); //4MHz
      SPI.setDataMode(SPI_MODE0);
      SPI.transfer(c);
#endif
  } else {
    // Fast SPI bitbang swiped from LPD8806 library
    for(uint8_t bit = 0x80; bit; bit >>= 1) {
      if(c & bit) *dataport |=  datapinmask;
      else        *dataport &= ~datapinmask;
      *clkport |=  clkpinmask;
      *clkport &= ~clkpinmask;
    }
  }
  // Add a little delay so it will work with WAVGAT MCU
  // For some reason with WAVGAT Nano SPI.transfer(c) doesn't wait until byte is sent
  __asm__("nop");
  __asm__("nop");
  __asm__("nop");
}
void Adafruit_ST7735_mod::writecommand(uint8_t c) {
#if defined (SPI_HAS_TRANSACTION)
  SPI.beginTransaction(mySPISettings);
#endif
  *rsport &= ~rspinmask;
  *csport &= ~cspinmask;
  //Serial.print("C ");
  spiwrite(c);

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}


void Adafruit_ST7735_mod::writedata(uint8_t c) {
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;

  //Serial.print("D ");
  spiwrite(c);

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
```

```
}

// Rather than a bazillion writecommand() and writedata() calls, screen
// initialization commands and arguments are organized in these tables
// stored in PROGMEM.  The table may look bulky, but that's mostly the
// formatting -- storage-wise this is hundreds of bytes more compact
// than the equivalent code.  Companion function follows.
#define DELAY 0x80
static const uint8_t PROGMEM
  Bcmd[] = {                  // Initialization commands for 7735B screens
    18,                       // 18 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, no args, w/delay
      50,                     //     50 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, no args, w/delay
      255,                    //     255 = 500 ms delay
    ST7735_COLMOD , 1+DELAY,  //  3: Set color mode, 1 arg + delay:
      0x05,                   //     16-bit color
      10,                     //     10 ms delay
    ST7735_FRMCTR1, 3+DELAY,  //  4: Frame rate control, 3 args + delay:
      0x00,                   //     fastest refresh
      0x06,                   //     6 lines front porch
      0x03,                   //     3 lines back porch
      10,                     //     10 ms delay
    ST7735_MADCTL , 1      ,  //  5: Memory access ctrl (directions), 1 arg:
      0x08,                   //     Row addr/col addr, bottom to top refresh
    ST7735_DISSET5, 2      ,  //  6: Display settings #5, 2 args, no delay:
      0x15,                   //     1 clk cycle nonoverlap, 2 cycle gate
                              //     rise, 3 cycle osc equalize
      0x02,                   //     Fix on VTL
    ST7735_INVCTR , 1      ,  //  7: Display inversion control, 1 arg:
      0x0,                    //     Line inversion
    ST7735_PWCTR1 , 2+DELAY,  //  8: Power control, 2 args + delay:
      0x02,                   //     GVDD = 4.7V
      0x70,                   //     1.0uA
      10,                     //     10 ms delay
    ST7735_PWCTR2 , 1      ,  //  9: Power control, 1 arg, no delay:
      0x05,                   //     VGH = 14.7V, VGL = -7.35V
    ST7735_PWCTR3 , 2      ,  // 10: Power control, 2 args, no delay:
      0x01,                   //     Opamp current small
      0x02,                   //     Boost frequency
    ST7735_VMCTR1 , 2+DELAY,  // 11: Power control, 2 args + delay:
      0x3C,                   //     VCOMH = 4V
      0x38,                   //     VCOML = -1.1V
      10,                     //     10 ms delay
    ST7735_PWCTR6 , 2      ,  // 12: Power control, 2 args, no delay:
      0x11, 0x15,
    ST7735_GMCTRP1,16      ,  // 13: Magical unicorn dust, 16 args, no delay:
      0x09, 0x16, 0x09, 0x20, //     (seriously though, not sure what
      0x21, 0x1B, 0x13, 0x19, //      these config values represent)
      0x17, 0x15, 0x1E, 0x2B,
      0x04, 0x05, 0x02, 0x0E,
    ST7735_GMCTRN1,16+DELAY,  // 14: Sparkles and rainbows, 16 args + delay:
      0x0B, 0x14, 0x08, 0x1E, //     (ditto)
      0x22, 0x1D, 0x18, 0x1E,
      0x1B, 0x1A, 0x24, 0x2B,
      0x06, 0x06, 0x02, 0x0F,
      10,                     //     10 ms delay
    ST7735_CASET  , 4      ,  // 15: Column addr set, 4 args, no delay:
      0x00, 0x02,             //     XSTART = 2
      0x00, 0x81,             //     XEND = 129
    ST7735_RASET  , 4      ,  // 16: Row addr set, 4 args, no delay:
      0x00, 0x02,             //     XSTART = 1
      0x00, 0x81,             //     XEND = 160
    ST7735_NORON  ,   DELAY,  // 17: Normal display on, no args, w/delay
      10,                     //     10 ms delay
    ST7735_DISPON ,   DELAY,  // 18: Main screen turn on, no args, w/delay
      255 },                  //     255 = 500 ms delay


  Rcmd1[] = {                 // Init for 7735R, part 1 (red or green tab)
    15,                       // 15 commands in list:
    ST7735_SWRESET,   DELAY,  //  1: Software reset, 0 args, w/delay
      150,                    //     150 ms delay
    ST7735_SLPOUT ,   DELAY,  //  2: Out of sleep mode, 0 args, w/delay
      255,                    //     500 ms delay
    ST7735_FRMCTR1, 3      ,  //  3: Frame rate ctrl - normal mode, 3 args:
      0x01, 0x2C, 0x2D,       //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
    ST7735_FRMCTR2, 3      ,  //  4: Frame rate control - idle mode, 3 args:
      0x01, 0x2C, 0x2D,       //     Rate = fosc/(1x2+40) * (LINE+2C+2D)
    ST7735_FRMCTR3, 6      ,  //  5: Frame rate ctrl - partial mode, 6 args:
      0x01, 0x2C, 0x2D,       //     Dot inversion mode
      0x01, 0x2C, 0x2D,       //     Line inversion mode
    ST7735_INVCTR , 1      ,  //  6: Display inversion ctrl, 1 arg, no delay:
      0x07,                   //     No inversion
    ST7735_PWCTR1 , 3      ,  //  7: Power control, 3 args, no delay:
      0xA2,
      0x02,                   //     -4.6V
      0x84,                   //     AUTO mode
    ST7735_PWCTR2 , 1      ,  //  8: Power control, 1 arg, no delay:
      0xC5,                   //     VGH25 = 2.4C VGSEL = -10 VGH = 3 * AVDD
    ST7735_PWCTR3 , 2      ,  //  9: Power control, 2 args, no delay:
      0x0A,                   //     Opamp current small
      0x00,                   //     Boost frequency
    ST7735_PWCTR4 , 2      ,  // 10: Power control, 2 args, no delay:
      0x8A,                   //     BCLK/2, Opamp current small & Medium low
      0x2A,
    ST7735_PWCTR5 , 2      ,  // 11: Power control, 2 args, no delay:
      0x8A, 0xEE,
    ST7735_VMCTR1 , 1      ,  // 12: Power control, 1 arg, no delay:
      0x0E,
    ST7735_INVOFF , 0      ,  // 13: Don't invert display, no args, no delay
    ST7735_MADCTL , 1      ,  // 14: Memory access control (directions), 1 arg:
      0xC8,                   //     row addr/col addr, bottom to top refresh
    ST7735_COLMOD , 1      ,  // 15: set color mode, 1 arg, no delay:
      0x05 },                 //     16-bit color


  Rcmd2green[] = {            // Init for 7735R, part 2 (green tab only)
    2,                        //  2 commands in list:
    ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
      0x00, 0x02,             //     XSTART = 0
      0x00, 0x7F+0x02,        //     XEND = 127
    ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
        0x00, 0x01,             //     XSTART = 0
        0x00, 0x9F+0x01 },      //     XEND = 159
    Rcmd2red[] = {              // Init for 7735R, part 2 (red tab only)
      2,                        //  2 commands in list:
      ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
        0x00, 0x00,             //     XSTART = 0
        0x00, 0x7F,             //     XEND = 127
      ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
        0x00, 0x00,             //     XSTART = 0
        0x00, 0x9F },           //     XEND = 159

    Rcmd2green144[] = {              // Init for 7735R, part 2 (green 1.44 tab)
      2,                        //  2 commands in list:
      ST7735_CASET  , 4      ,  //  1: Column addr set, 4 args, no delay:
        0x00, 0x00,             //     XSTART = 0
        0x00, 0x7F,             //     XEND = 127
      ST7735_RASET  , 4      ,  //  2: Row addr set, 4 args, no delay:
        0x00, 0x00,             //     XSTART = 0
        0x00, 0x7F },           //     XEND = 127

    Rcmd3[] = {                 // Init for 7735R, part 3 (red or green tab)
      4,                        //  4 commands in list:
      ST7735_GMCTRP1, 16      , //  1: Magical unicorn dust, 16 args, no delay:
        0x02, 0x1c, 0x07, 0x12,
        0x37, 0x32, 0x29, 0x2d,
        0x29, 0x25, 0x2B, 0x39,
        0x00, 0x01, 0x03, 0x10,
      ST7735_GMCTRN1, 16      , //  2: Sparkles and rainbows, 16 args, no delay:
        0x03, 0x1d, 0x07, 0x06,
        0x2E, 0x2C, 0x29, 0x2D,
        0x2E, 0x2E, 0x37, 0x3F,
        0x00, 0x00, 0x02, 0x10,
      ST7735_NORON  ,    DELAY, //  3: Normal display on, no args, w/delay
        10,                     //     10 ms delay
      ST7735_DISPON ,    DELAY, //  4: Main screen turn on, no args w/delay
        100 };                  //     100 ms delay
  // Companion code to the above tables.  Reads and issues
  // a series of LCD commands stored in PROGMEM byte array.
  void Adafruit_ST7735_mod::commandList(const uint8_t *addr) {

    uint8_t  numCommands, numArgs;
    uint16_t ms;

    numCommands = pgm_read_byte(addr++);   // Number of commands to follow
    while(numCommands--) {                 // For each command...
      writecommand(pgm_read_byte(addr++)); //   Read, issue command
      numArgs  = pgm_read_byte(addr++);    //   Number of args to follow
      ms       = numArgs & DELAY;          //   If hibit set, delay follows args
      numArgs &= ~DELAY;                   //   Mask out delay bit
      while(numArgs--) {                   //   For each argument...
        writedata(pgm_read_byte(addr++));  //     Read, issue argument
      }
      if(ms) {
        ms = pgm_read_byte(addr++); // Read post-command delay time (ms)
        if(ms == 255) ms = 500;     // If 255, delay for 500 ms
        delay(ms);
      }
    }
  }
  // Initialization code common to both 'B' and 'R' type displays
  void Adafruit_ST7735_mod::commonInit(const uint8_t *cmdList) {
    colstart  = rowstart = 0; // May be overridden in init func
    pinMode(_rs, OUTPUT);
    pinMode(_cs, OUTPUT);
    csport    = portOutputRegister(digitalPinToPort(_cs));
    rsport    = portOutputRegister(digitalPinToPort(_rs));
    cspinmask = digitalPinToBitMask(_cs);
    rspinmask = digitalPinToBitMask(_rs);

    if(hwSPI) { // Using hardware SPI
  #if defined (SPI_HAS_TRANSACTION)
      SPI.begin();
      mySPISettings = SPISettings(8000000, MSBFIRST, SPI_MODE0);
  #elif defined (__AVR__)
```

```
    SPCRbackup = SPCR;
    SPI.begin();
    SPI.setClockDivider(SPI_CLOCK_DIV4);
    SPI.setDataMode(SPI_MODE0);
    mySPCR = SPCR; // save our preferred state
    //Serial.print("mySPCR = 0x"); Serial.println(SPCR, HEX);
    SPCR = SPCRbackup;  // then restore
#elif defined (__SAM3X8E__)
    SPI.begin();
    SPI.setClockDivider(21); //4MHz
    SPI.setDataMode(SPI_MODE0);
#endif
  } else {
    pinMode(_sclk, OUTPUT);
    pinMode(_sid , OUTPUT);
    clkport     = portOutputRegister(digitalPinToPort(_sclk));
    dataport    = portOutputRegister(digitalPinToPort(_sid));
    clkpinmask  = digitalPinToBitMask(_sclk);
    datapinmask = digitalPinToBitMask(_sid);
    *clkport   &= ~clkpinmask;
    *dataport  &= ~datapinmask;
  }
  // toggle RST low to reset; CS low so it'll listen to us
  *csport &= ~cspinmask;
  if (_rst) {
    pinMode(_rst, OUTPUT);
    digitalWrite(_rst, HIGH);
    delay(500);
    digitalWrite(_rst, LOW);
    delay(500);
    digitalWrite(_rst, HIGH);
    delay(500);
  }

  if(cmdList) commandList(cmdList);
}
// Initialization for ST7735B screens
```

```
void Adafruit_ST7735_mod::initB(void) {
  commonInit(Bcmd);
}
// Initialization for ST7735R screens (green or red tabs)
void Adafruit_ST7735_mod::initR(uint8_t options) {
  commonInit(Rcmd1);
  if(options == INITR_GREENTAB) {
    commandList(Rcmd2green);
    colstart = 2;
    rowstart = 1;
  } else if(options == INITR_144GREENTAB) {
    _height = ST7735_TFTHEIGHT_144;
    commandList(Rcmd2green144);
    colstart = 2;
    rowstart = 3;
  } else {
    // colstart, rowstart left at default '0' values
    commandList(Rcmd2red);
  }
  commandList(Rcmd3);

  // if black, change MADCTL color filter
  if (options == INITR_BLACKTAB) {
    writecommand(ST7735_MADCTL);
    writedata(0xC0);
  }

  tabcolor = options;
}
void Adafruit_ST7735_mod::setAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1) {
  writecommand(ST7735_CASET); // Column addr set
  writedata(0x00);
  writedata(x0+colstart);     // XSTART
  writedata(0x00);
  writedata(x1+colstart);     // XEND

  writecommand(ST7735_RASET); // Row addr set
  writedata(0x00);
  writedata(y0+rowstart);     // YSTART
  writedata(0x00);
  writedata(y1+rowstart);     // YEND

  writecommand(ST7735_RAMWR); // write to RAM
}
void Adafruit_ST7735_mod::startAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1) {
  setAddrWindow(x0, y0, x1, y1);
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
}
void Adafruit_ST7735_mod::endAddrWindow() {
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
void Adafruit_ST7735_mod::pushColor(uint16_t color) {
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  spiwrite(color >> 8);
  spiwrite(color);
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
void Adafruit_ST7735_mod::drawPixel(int16_t x, int16_t y, uint16_t color) {
  if((x < 0) ||(x >= _width) || (y < 0) || (y >= _height)) return;
  setAddrWindow(x,y,x+1,y+1);
```

```
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  spiwrite(color >> 8);
  spiwrite(color);

  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
void Adafruit_ST7735_mod::drawFastVLine(int16_t x, int16_t y, int16_t h,
 uint16_t color) {
  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((y+h-1) >= _height) h = _height-y;
  setAddrWindow(x, y, x, y+h-1);
  uint8_t hi = color >> 8, lo = color;

#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  while (h--) {
    spiwrite(hi);
    spiwrite(lo);
  }
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
void Adafruit_ST7735_mod::drawFastHLine(int16_t x, int16_t y, int16_t w,
  uint16_t color) {
```

```
  // Rudimentary clipping
  if((x >= _width) || (y >= _height)) return;
  if((x+w-1) >= _width)  w = _width-x;
  setAddrWindow(x, y, x+w-1, y);

  uint8_t hi = color >> 8, lo = color;

#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  while (w--) {
    spiwrite(hi);
    spiwrite(lo);
  }
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
void Adafruit_ST7735_mod::fillScreen(uint16_t color) {
  fillRect(0, 0,  _width, _height, color);
}
// fill a rectangle
void Adafruit_ST7735_mod::fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
  uint16_t color) {

  // rudimentary clipping (drawChar w/big text requires this)
  if((x >= _width) || (y >= _height)) return;
  if((x + w - 1) >= _width)  w = _width  - x;
  if((y + h - 1) >= _height) h = _height - y;
  setAddrWindow(x, y, x+w-1, y+h-1);
  uint8_t hi = color >> 8, lo = color;
#if defined (SPI_HAS_TRANSACTION)
    SPI.beginTransaction(mySPISettings);
```

```
#endif
  *rsport |=  rspinmask;
  *csport &= ~cspinmask;
  for(y=h; y>0; y--) {
    for(x=w; x>0; x--) {
      spiwrite(hi);
      spiwrite(lo);
    }
  }
  *csport |= cspinmask;
#if defined (SPI_HAS_TRANSACTION)
    SPI.endTransaction();
#endif
}
// Pass 8-bit (each) R,G,B, get back 16-bit packed color
uint16_t Adafruit_ST7735_mod::Color565(uint8_t r, uint8_t g, uint8_t b) {
  return ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | (b >> 3);
}
#define MADCTL_MY  0x80
#define MADCTL_MX  0x40
#define MADCTL_MV  0x20
#define MADCTL_ML  0x10
#define MADCTL_RGB 0x00
#define MADCTL_BGR 0x08
#define MADCTL_MH  0x04

void Adafruit_ST7735_mod::setRotation(uint8_t m) {

  writecommand(ST7735_MADCTL);
  rotation = m % 4; // can't be higher than 3
  switch (rotation) {
   case 0:
     if (tabcolor == INITR_BLACKTAB) {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_RGB);
     } else {
       writedata(MADCTL_MX | MADCTL_MY | MADCTL_BGR);
     }
     _width  = ST7735_TFTWIDTH;

     if (tabcolor == INITR_144GREENTAB)
       _height = ST7735_TFTHEIGHT_144;
     else
       _height = ST7735_TFTHEIGHT_18;

     break;
   case 1:
     if (tabcolor == INITR_BLACKTAB) {
       writedata(MADCTL_MY | MADCTL_MV | MADCTL_RGB);
     } else {
       writedata(MADCTL_MY | MADCTL_MV | MADCTL_BGR);
     }

     if (tabcolor == INITR_144GREENTAB)
       _width = ST7735_TFTHEIGHT_144;
     else
       _width = ST7735_TFTHEIGHT_18;

     _height = ST7735_TFTWIDTH;
     break;
  case 2:
     if (tabcolor == INITR_BLACKTAB) {
       writedata(MADCTL_RGB);
     } else {
       writedata(MADCTL_BGR);
     }
     _width  = ST7735_TFTWIDTH;
     if (tabcolor == INITR_144GREENTAB)
       _height = ST7735_TFTHEIGHT_144;
     else
       _height = ST7735_TFTHEIGHT_18;

    break;
   case 3:
     if (tabcolor == INITR_BLACKTAB) {
       writedata(MADCTL_MX | MADCTL_MV | MADCTL_RGB);
     } else {
       writedata(MADCTL_MX | MADCTL_MV | MADCTL_BGR);
     }
     if (tabcolor == INITR_144GREENTAB)
       _width = ST7735_TFTHEIGHT_144;
     else
       _width = ST7735_TFTHEIGHT_18;

     _height = ST7735_TFTWIDTH;
     break;
  }
}

void Adafruit_ST7735_mod::invertDisplay(boolean i) {
  writecommand(i ? ST7735_INVON : ST7735_INVOFF);
}

#ifndef _ADAFRUIT_ST7735_MOD_H_
#define _ADAFRUIT_ST7735_MOD_H_

#if ARDUINO >= 100
 #include "Arduino.h"
 #include "Print.h"
#else
 #include "WProgram.h"
#endif

#include <Adafruit_GFX.h>

#if defined(__SAM3X8E__)
  #include <include/pio.h>
  #define PROGMEM
  #define pgm_read_byte(addr) (*(const unsigned char *)(addr))
  #define pgm_read_word(addr) (*(const unsigned short *)(addr))
  typedef unsigned char prog_uchar;
#elif defined(__AVR__)
  #include <avr/pgmspace.h>
#elif defined(ESP8266)
  #include <pgmspace.h>
#endif

#if defined(__SAM3X8E__)
    #undef __FlashStringHelper::F(string_literal)
    #define F(string_literal) string_literal
#endif

// some flags for initR() :(
#define INITR_GREENTAB 0x0
#define INITR_REDTAB   0x1
#define INITR_BLACKTAB   0x2

#define INITR_18GREENTAB    INITR_GREENTAB
#define INITR_18REDTAB      INITR_REDTAB
#define INITR_18BLACKTAB    INITR_BLACKTAB
#define INITR_144GREENTAB   0x1

#define ST7735_TFTWIDTH  128
// for 1.44" display
#define ST7735_TFTHEIGHT_144 128
// for 1.8" display
#define ST7735_TFTHEIGHT_18  160

#define ST7735_NOP     0x00
#define ST7735_SWRESET 0x01
#define ST7735_RDDID   0x04
#define ST7735_RDDST   0x09

#define ST7735_SLPIN   0x10
#define ST7735_SLPOUT  0x11
#define ST7735_PTLON   0x12
```

```
#define ST7735_NORON   0x13

#define ST7735_INVOFF  0x20
#define ST7735_INVON   0x21
#define ST7735_DISPOFF 0x28
#define ST7735_DISPON  0x29
#define ST7735_CASET   0x2A
#define ST7735_RASET   0x2B
#define ST7735_RAMWR   0x2C
#define ST7735_RAMRD   0x2E

#define ST7735_PTLAR   0x30
#define ST7735_COLMOD  0x3A
#define ST7735_MADCTL  0x36

#define ST7735_FRMCTR1 0xB1
#define ST7735_FRMCTR2 0xB2
#define ST7735_FRMCTR3 0xB3
#define ST7735_INVCTR  0xB4
#define ST7735_DISSET5 0xB6

#define ST7735_PWCTR1  0xC0
#define ST7735_PWCTR2  0xC1
#define ST7735_PWCTR3  0xC2
#define ST7735_PWCTR4  0xC3
#define ST7735_PWCTR5  0xC4
#define ST7735_VMCTR1  0xC5

#define ST7735_RDID1   0xDA
#define ST7735_RDID2   0xDB
#define ST7735_RDID3   0xDC
#define ST7735_RDID4   0xDD

#define ST7735_PWCTR6  0xFC

#define ST7735_GMCTRP1 0xE0
```

```
#define ST7735_GMCTRN1 0xE1

// Color definitions   R5:G6:B5
#define ST7735_BLACK   0x0000
#define ST7735_BLUE    0x001F
#define ST7735_RED     0xF800
#define ST7735_GREEN   0x07E0
#define ST7735_CYAN    0x07FF
#define ST7735_MAGENTA 0xF81F
#define ST7735_YELLOW  0xFFE0
#define ST7735_WHITE   0xFFFF



class Adafruit_ST7735_mod : public Adafruit_GFX {

 public:

  Adafruit_ST7735_mod(int8_t CS, int8_t RS, int8_t SID, int8_t SCLK, int8_t RST = -1);
  Adafruit_ST7735_mod(int8_t CS, int8_t RS, int8_t RST = -1);

  void     initB(void),                             // for ST7735B displays
           initR(uint8_t options = INITR_GREENTAB), // for ST7735R
           setAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1),
           startAddrWindow(uint8_t x0, uint8_t y0, uint8_t x1, uint8_t y1),
           endAddrWindow(),
           pushColor(uint16_t color),
           fillScreen(uint16_t color),
           drawPixel(int16_t x, int16_t y, uint16_t color),
           drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color),
           drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color),
           fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color),
           setRotation(uint8_t r),
           invertDisplay(boolean i);

  uint16_t Color565(uint8_t r, uint8_t g, uint8_t b);
  /* These are not for current use, 8-bit protocol only!
  uint8_t  readdata(void),
           readcommand8(uint8_t);
  uint16_t readcommand16(uint8_t);
  uint32_t readcommand32(uint8_t);
  void     dummyclock(void);
  */

 private:
  uint8_t  tabcolor;

  void     spiwrite(uint8_t),
           writecommand(uint8_t c),
           writedata(uint8_t d),
           commandList(const uint8_t *addr),
           commonInit(const uint8_t *cmdList);
//uint8_t  spiread(void);

  boolean  hwSPI;

#if defined(__AVR__) || defined(CORE_TEENSY)
  volatile uint8_t *dataport, *clkport, *csport, *rsport;
  uint8_t  _cs, _rs, _rst, _sid, _sclk,
           datapinmask, clkpinmask, cspinmask, rspinmask,
           colstart, rowstart; // some displays need this changed
#elif defined(__arm__)
  volatile RwReg  *dataport, *clkport, *csport, *rsport;
  uint32_t  _cs, _rs, _sid, _sclk,
            datapinmask, clkpinmask, cspinmask, rspinmask,
            colstart, rowstart; // some displays need this changed
  int32_t   _rst;  // Must use signed type since a -1 sentinel is assigned.
#endif

};
#endif
```

```
#include "setup.h"
#if EXAMPLE == 4
#include "Arduino.h"
#include "Adafruit_ST7735_mod.h"
#include <BufferedCameraOV7670_QQVGA_20hz_Grayscale.h>
#include "GrayScaleTable.h"
BufferedCameraOV7670_QQVGA_20hz_Grayscale camera;
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
int TFT_RST = 49;
int TFT_CS  = 53;
int TFT_DC  = 48;
// TFT_SPI_clock = 52 and TFT_SPI_data = 51
#else
int TFT_RST = 10;
int TFT_CS  = 9;
int TFT_DC  = 8;
// TFT_SPI_clock = 13 and TFT_SPI_data = 11
#endif
Adafruit_ST7735_mod tft = Adafruit_ST7735_mod(TFT_CS, TFT_DC, TFT_RST);
// this is called in Arduino setup() function
void initializeScreenAndCamera() {
  bool cameraInitialized = camera.init();
  tft.initR(INITR_BLACKTAB);
  if (cameraInitialized) {
    // flash green screen if camera setup was successful
    tft.fillScreen(ST7735_GREEN);
    delay(1000);
    tft.fillScreen(ST7735_BLACK);
  } else {
    // red screen if failed to connect to camera
    tft.fillScreen(ST7735_RED);
    delay(3000);
  }

  TIMSK0 = 0; // disable "millis" timer interrupt
}
inline void sendLineToDisplay() __attribute__((always_inline));
inline void screenLineStart(void) __attribute__((always_inline));
inline void screenLineEnd(void) __attribute__((always_inline));
inline void sendPixelByte(uint8_t byte) __attribute__((always_inline));


// Normally it is a portrait screen. Use it as landscape
uint8_t screen_w = ST7735_TFTHEIGHT_18;
uint8_t screen_h = ST7735_TFTWIDTH;
uint8_t screenLineIndex;
bool alternateLine = true;
// this is called in Arduino loop() function
void processFrame() {
  screenLineIndex = screen_h;
  camera.waitForVsync();
  if (alternateLine) {
    screenLineIndex--;
    camera.readLine();
  }
  alternateLine = !alternateLine;
  for (uint8_t i = 0; i < camera.getLineCount() / 2; i++) {
    camera.readLine();
    sendLineToDisplay();
  }
}
void sendLineToDisplay() {
  if (screenLineIndex > 0) {
    screenLineStart();

    for (uint16_t i=0; i<camera.getLineLength(); i++) {
      sendPixelByte(graysScaleTableHigh[camera.getPixelByte(i)]);
      sendPixelByte(graysScaleTableLow[camera.getPixelByte(i)]);
    }
    screenLineEnd();
  }
}

void screenLineStart()   {
  if (screenLineIndex > 0) screenLineIndex-=2;
  tft.startAddrWindow(screenLineIndex, 0, screenLineIndex, screen_w-1);
}

void screenLineEnd() {
  tft.endAddrWindow();
}

void sendPixelByte(uint8_t byte) {
  SPDR = byte;
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");

}


#endif

#include "setup.h"
#if EXAMPLE == 1
#include "Arduino.h"
#include "Adafruit_ST7735_mod.h"
#include <BufferedCameraOV7670_QQVGA_10hz.h>
#include <BufferedCameraOV7670_QQVGA.h>
#include <BufferedCameraOV7670_QVGA.h>
#include <BufferedCameraOV7670_QQVGA_10hz_Grayscale.h>
#include "GrayScaleTable.h"
#define GRAYSCALE_PIXELS 0

#if GRAYSCALE_PIXELS == 1
BufferedCameraOV7670_QQVGA_10hz_Grayscale camera;
```

```
#else
BufferedCameraOV7670_QQVGA_10hz camera(CameraOV7670::PIXEL_RGB565);
//BufferedCameraOV7670_QQVGA camera(CameraOV7670::PIXEL_RGB565, BufferedCameraOV7670_QQVGA::FPS_5_Hz);
//BufferedCameraOV7670_QQVGA camera(CameraOV7670::PIXEL_RGB565, BufferedCameraOV7670_QQVGA::FPS_2_Hz);
//BufferedCameraOV7670_QVGA camera(CameraOV7670::PIXEL_RGB565, BufferedCameraOV7670_QVGA::FPS_2p5_Hz);
#endif
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
int TFT_RST = 49;
int TFT_CS  = 53;
int TFT_DC  = 48;
// TFT_SPI_clock = 52 and TFT_SPI_data = 51
#else
int TFT_RST = 10;
int TFT_CS  = 9;
int TFT_DC  = 8;
// TFT_SPI_clock = 13 and TFT_SPI_data = 11
#endif
Adafruit_ST7735_mod tft = Adafruit_ST7735_mod(TFT_CS, TFT_DC, TFT_RST);

// this is called in Arduino setup() function
void initializeScreenAndCamera() {
  bool cameraInitialized = camera.init();
  tft.initR(INITR_BLACKTAB);
  if (cameraInitialized) {
    // flash green screen if camera setup was successful
    tft.fillScreen(ST7735_GREEN);
    delay(1000);
    tft.fillScreen(ST7735_BLACK);
  } else {
    // red screen if failed to connect to camera
    tft.fillScreen(ST7735_RED);
    delay(3000);
  }
}
```

```
inline void sendLineToDisplay() __attribute__((always_inline));
inline void screenLineStart(void) __attribute__((always_inline));
inline void screenLineEnd(void) __attribute__((always_inline));
inline void sendPixelByte(uint8_t byte) __attribute__((always_inline));


// Normally it is a portrait screen. Use it as landscape
uint8_t screen_w = ST7735_TFTHEIGHT_18;
uint8_t screen_h = ST7735_TFTWIDTH;
uint8_t screenLineIndex;

// this is called in Arduino loop() function
void processFrame() {
  screenLineIndex = screen_h;

  noInterrupts();
  camera.waitForVsync();
  camera.ignoreVerticalPadding();

  for (uint8_t i = 0; i < camera.getLineCount(); i++) {
    camera.readLine();
    sendLineToDisplay();
  }
  interrupts();
}
static const uint16_t byteCountForDisplay = camera.getPixelBufferLength() < screen_w*2 ?
                                            camera.getPixelBufferLength() : screen_w*2;
void sendLineToDisplay() {
  if (screenLineIndex > 0) {
    screenLineStart();
#if GRAYSCALE_PIXELS == 1
    for (uint16_t i=0; i<camera.getLineLength(); i++) {
      sendPixelByte(graysScaleTableHigh[camera.getPixelByte(i)]);
      sendPixelByte(graysScaleTableLow[camera.getPixelByte(i)]);
    }
#else
    for (uint16_t i=0; i<byteCountForDisplay; i++) {
      sendPixelByte(camera.getPixelByte(i));
    }
#endif
    screenLineEnd();
  }
}

void screenLineStart()   {
  if (screenLineIndex > 0) screenLineIndex--;
  tft.startAddrWindow(screenLineIndex, 0, screenLineIndex, screen_w-1);
}

void screenLineEnd() {
  tft.endAddrWindow();
}
void sendPixelByte(uint8_t byte) {
  SPDR = byte;

  // this must be adjusted if sending loop has more/less instructions

  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");

#if GRAYSCALE_PIXELS == 1
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
#endif
}
#endif

#include "setup.h"
#if EXAMPLE == 2
```

```
#include "Arduino.h"
#include "Adafruit_ST7735_mod.h"
#include "CameraOV7670.h"

// scaler values for specific refresh rates
static const uint8_t FPS_1_Hz = 9;
static const uint8_t FPS_0p5_Hz = 19;
static const uint8_t FPS_0p33_Hz = 29;
static const uint16_t lineLength = 640;
static const uint16_t lineCount = 480;
// Since the 1.8" TFT screen is only 160x128 only top right corner of the VGA picture is visible.
CameraOV7670 camera(CameraOV7670::RESOLUTION_VGA_640x480,
CameraOV7670::PIXEL_RGB565, FPS_1_Hz);
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
int TFT_RST = 49;
int TFT_CS  = 53;
int TFT_DC  = 48;
// TFT_SPI_clock = 52 and TFT_SPI_data = 51
#else
int TFT_RST = 10;
int TFT_CS  = 9;
int TFT_DC  = 8;
// TFT_SPI_clock = 13 and TFT_SPI_data = 11
#endif
Adafruit_ST7735_mod tft = Adafruit_ST7735_mod(TFT_CS, TFT_DC, TFT_RST);

// this is called in Arduino setup() function
void initializeScreenAndCamera() {
  bool cameraInitialized = camera.init();
  tft.initR(INITR_BLACKTAB);
  if (cameraInitialized) {
    tft.fillScreen(ST7735_BLACK);
  } else {
    tft.fillScreen(ST7735_RED);
    delay(3000);
  }

  TIMSK0 = 0; // disable "millis" timer interrupt
```

```
}
inline void screenLineStart(void) __attribute__((always_inline));
inline void screenLineEnd(void) __attribute__((always_inline));
inline void sendPixelByte(uint8_t byte) __attribute__((always_inline));
inline void pixelSendingDelay() __attribute__((always_inline));



// Normally it is a portrait screen. Use it as landscape
uint8_t screen_w = ST7735_TFTHEIGHT_18;
uint8_t screen_h = ST7735_TFTWIDTH;
uint8_t screenLineIndex;

// this is called in Arduino loop() function
void processFrame() {
  uint8_t pixelByte;
  screenLineIndex = screen_h;

  camera.waitForVsync();
  camera.ignoreVerticalPadding();

  for (uint16_t y = 0; y < lineCount; y++) {
    screenLineStart();
    camera.ignoreHorizontalPaddingLeft();

    for (uint16_t x = 0; x < lineLength; x++) {

      camera.waitForPixelClockRisingEdge();
      camera.readPixelByte(pixelByte);
      sendPixelByte(pixelByte);

      camera.waitForPixelClockRisingEdge();
      camera.readPixelByte(pixelByte);
      sendPixelByte(pixelByte);
    }

    camera.ignoreHorizontalPaddingRight();
    pixelSendingDelay(); // prevent sending collision
    screenLineEnd();
  }
}
void screenLineStart()   {
  if (screenLineIndex > 0) screenLineIndex--;
  tft.startAddrWindow(screenLineIndex, 0, screenLineIndex, screen_w-1);
}
void screenLineEnd() {
  tft.endAddrWindow();
}
void sendPixelByte(uint8_t byte) {
  SPDR = byte;

  // this must be adjusted if sending loop has more/less instructions

  /*
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  */
}
void pixelSendingDelay() {
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
  asm volatile("nop");
}
#endif
```

CONCLUSION

In conclusion, the number plate detection system has become an essential tool for law enforcement agencies, toll booth operators, and parking management companies. The system uses computer vision technology to process pictures of license plates, which makes it possible for the plate number and vehicle identification to be automatically recognized. The number plate detection system provides many benefits, including increased efficiency, accuracy, and security in various applications. It can aid in reducing manual work and human error, enhancing public safety, and enhancing traffic flow.

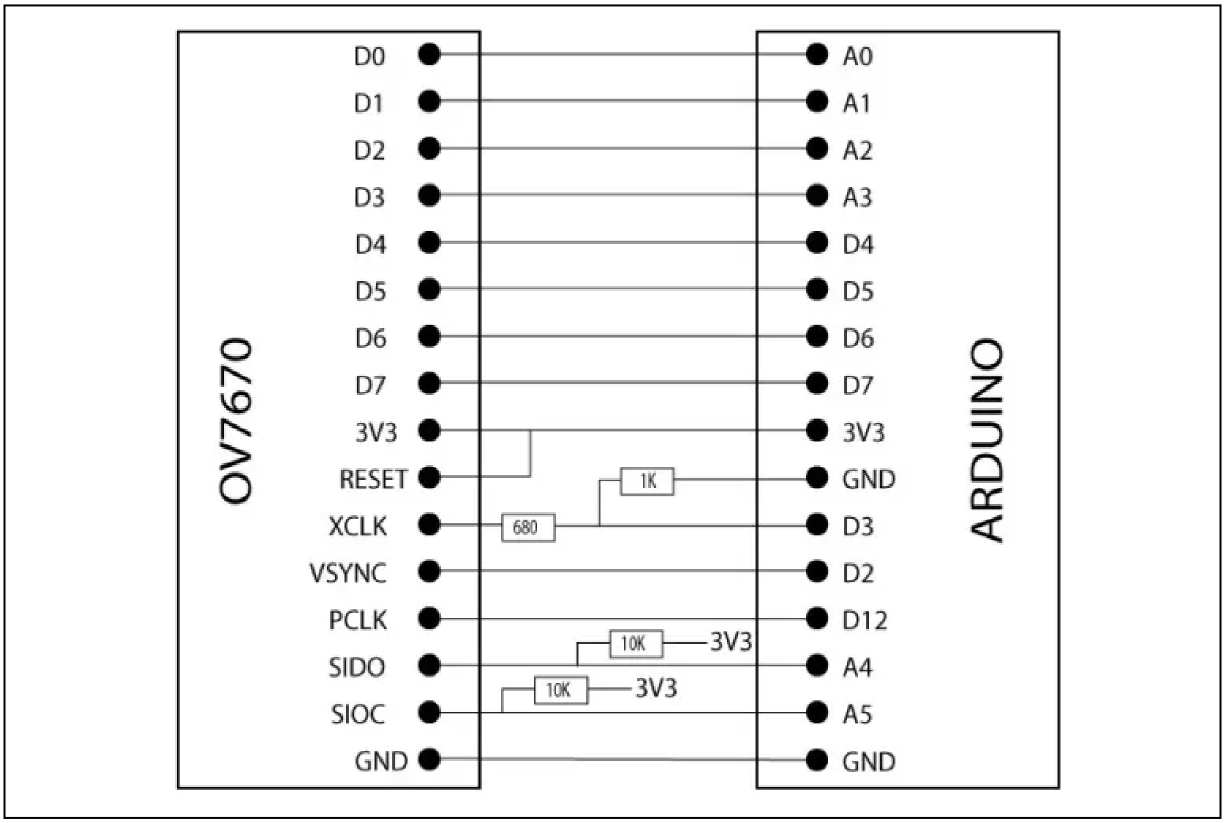
Despite the fact that there are still certain difficulties to be solved, such as variances in license plate formats and picture quality, advances in machine learning and artificial intelligence are steadily enhancing the precision and dependability of the system. Overall, the number plate detection system has proven to be a valuable technology that has revolutionized the way we manage and monitor vehicle traffic. In the upcoming years, it is anticipated that the system will continue to develop and get better, becoming a more crucial tool for numerous businesses.

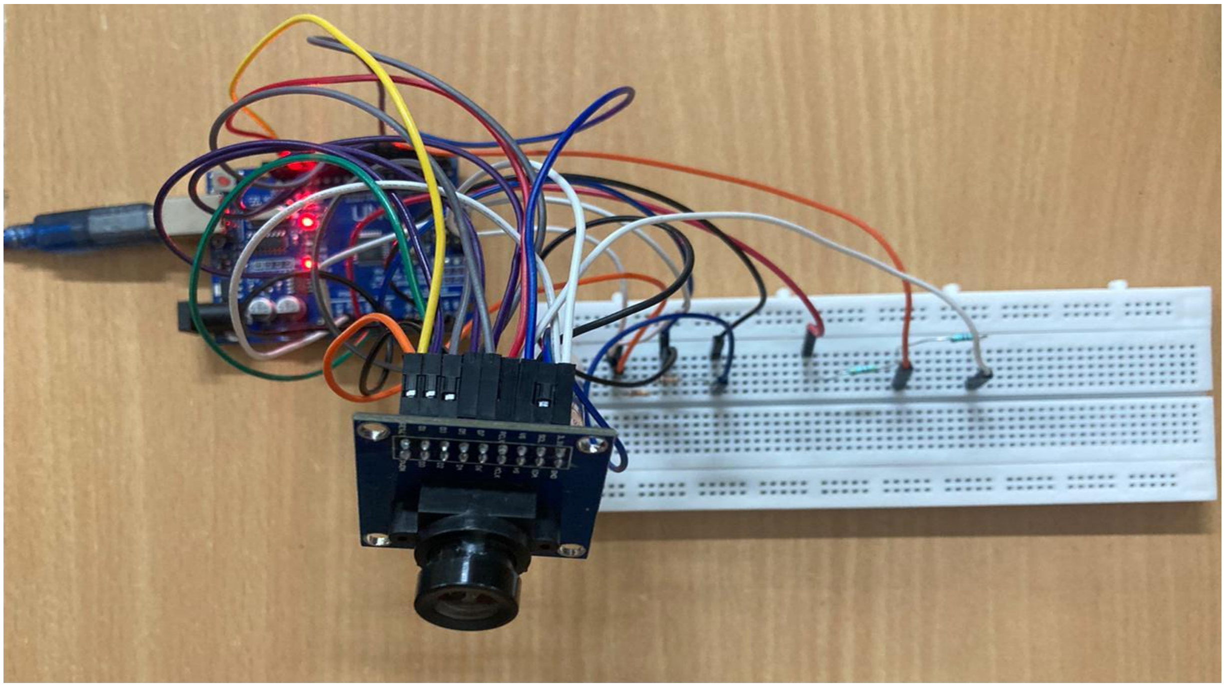
REFERENCES

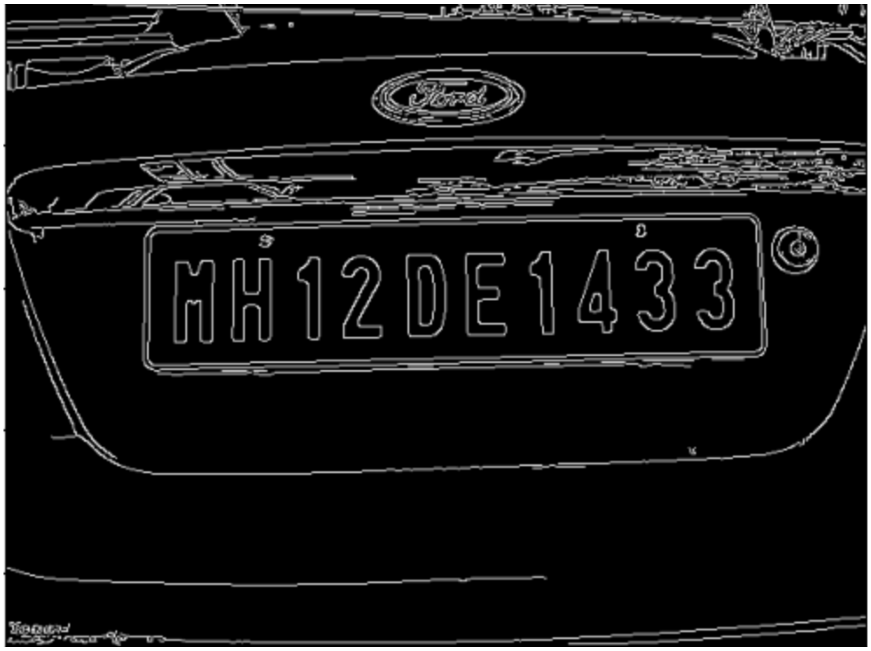
- Automatic Number Plate Recognition (ANPR) - 2023 Guide - viso.ai
- Realtime Number Plate Detection using Yolov7 - Easiest Explanation - 2023 - Machine Learning Projects
- Number Plate Detection using OpenCV & Python | by Apoorva Sinha | QuikNapp | Medium
- Srivastav, Gaurav. (2020). Automatic Number Plate Recognition. International Journal for Research in Applied Science and Engineering Technology. 8. 1105-1108. 10.22214/ijraset.2020.6178.
- Rai, Vanshika and Kamthania, Deepali, Automatic Number Plate Recognition (July 3, 2021). Proceedings of the International Conference on Innovative Computing & Communication (ICICC) 2021.
- Shahid, Umar. (2017). NUMBER PLATE RECOGNITION SYSTEM. 10.13140/RG.2.2.23058.71361.
- A. Kashyap, B. Suresh, A. Patil, S. Sharma and A. Jaiswal, "Automatic Number Plate Recognition," 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 2018, pp. 838-843, doi: 10.1109/ICACCCN.2018.8748287.

APPENDIX I:

The Images of the project are given below







data

	date	v_number
0	Tue Apr 4 05:41:21 2023	MH12 DE 1433