ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

ĐỖ MINH PHƯƠNG

Xử lý ảnh video theo thời gian thực trên kit STM32

Ngành: **Công nghệ Kỹ thuật Điện tử, Truyền thông**
Chuyên ngành: **Kỹ thuật Điện tử**
Mã ngành: **60520203**

LUẬN VĂN THẠC SỸ NGÀNH CÔNG NGHỆ KỸ THUẬT ĐIỆN TỬ TRUYỀN THÔNG

NGƯỜI HƯỚNG DẪN KHOA HỌC: PGS.TS. TRẦN QUANG VINH

HÀ NỘI - 2016

LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của bản thân, được xuất phát từ yêu cầu phát sinh trong quá trình làm luận văn. Các tài liệu có nguồn gốc rõ ràng, tuân thủ đúng nguyên tắc, kết quả trình bày trong luận văn là kết quả quá trình nghiên cứu trung thực, chưa từng được ai công bố trước đây.

Hà Nội, tháng 12 năm 2016

Tác giả luận văn

Đỗ Minh Phương

MỤC LỤC

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

MỞ ĐẦU

1. Đặt vấn đề

Hệ thống xử lý ảnh số dùng vi điều khiển có nhiều ứng dụng trong thực tế ở hầu hết các lĩnh vực như truyền hình, nhận dạng chữ viết, vân tay, y học, viễn thám, quân sự, nghiên cứu khoa học… Xây dựng một hệ thống xử lý ảnh số đòi hỏi một phạm vi rộng các kiến thức về phần cứng, phần mềm. Cùng với sự phát triển của khoa học kỹ thuật, công nghệ xử lý ảnh ngày càng được phát triển cả về thiết bị phần cứng và các giải pháp phần mềm.

Cùng với sự phát triển của công nghệ vi điện tử, các vi điều khiển đã có nhiều cải tiến về cấu trúc, thiết kế hệ thống, khả năng xử lý, tái lập trình hệ thống giúp giảm chi phí sản xuất, rút ngắn thời gian cải tiến, nâng cấp, sản xuất hệ thống.

Cấu trúc vi xử lý ARM (viết tắt từ tên gốc là Advanced RISC Machine) là một loại cấu trúc vi xử lý 32 bit và 64 bit kiểu RISC được sử dụng rộng rãi trong các thiết kế hệ thống nhúng. Do có đặc điểm tiết kiệm năng lượng, các bộ vi xử lý ARM chiếm ưu thế trong các sản phẩm điện tử di động, mà với các sản phẩm này việc tiêu thụ công suất thấp là một mục tiêu thiết kế quan trọng hàng đầu.

Ngày nay, hơn 75% CPU nhúng 32-bit là thuộc họ ARM, điều này khiến ARM trở thành cấu trúc 32-bit được sản xuất nhiều nhất trên thế giới. Giải pháp hệ thống trên chip (System-On-Chip) dựa trên bộ vi xử lý nhúng ARM được ứng dụng vào rất nhiều thị trường khác nhau bao gồm các ứng dụng doanh nghiệp, các hệ thống ô tô, mạng gia đình và công nghệ mạng không dây. Nhiều thiết bị xử lý ảnh chuyên dụng đã được thiết kế sử dụng vi điều khiển ARM.

Dòng vi xử lý ARM Cortex dựa trên một kiến trúc chuẩn đủ để đáp ứng hầu hết các yêu cầu về hiệu năng làm việc trong tất cả các lĩnh vực trên. Thêm vào đó là việc lập trình được đơn giản hóa đáng kể giúp kiến trúc ARM trở thành một lựa chọn tốt cho ngay cả những ứng dụng đơn giản nhất.

Những đặc điểm nổi trội của dòng ARM Cortex đã thu hút các nhà sản xuất IC, hơn 240 dòng vi điều khiển dựa vào nhân Cortex đã được giới thiệu. Không nằm ngoài xu hướng đó, hãng sản xuất chip ST Microelectronic đã nhanh chóng đưa ra dòng STM32 là vi điều khiển dựa trên nền tảng lõi ARM Cortex®-M thế hệ mới do hãng ARM thiết kế.

Khả năng kết hợp trong thiết kế hệ thống vi điều khiển STM32 và các giải thuật phần mềm cho phép xây dựng một hệ thống xử lý luồng ảnh video thời gian thực đáp ứng yêu cầu cụ thể cần thiết kế.

2. Nội dung của đề tài, các vấn đề cần giải quyết:

Căn cứ vào các nhận xét kể trên, luận văn đề ra mục tiêu tổng quát là: Nghiên cứu, thiết kế và phát triển một hệ thống thu thập và xử lý ảnh video theo thời gian thực trên Kit STM32.

Để đạt được mục tiêu này, các vấn đề chính sau đây đã được giải quyết:

- Thiết kế lắp ráp hệ thống bắt ảnh gồm Camera OV7670 với Kit vi điều khiển STM32F4 Discovery hiển thị trên màn hình tinh thể lỏng LCD 3,2".

- Phát triển phần mềm nhúng cho phép cấu hình hệ thống và bắt các khung ảnh của luồng video.

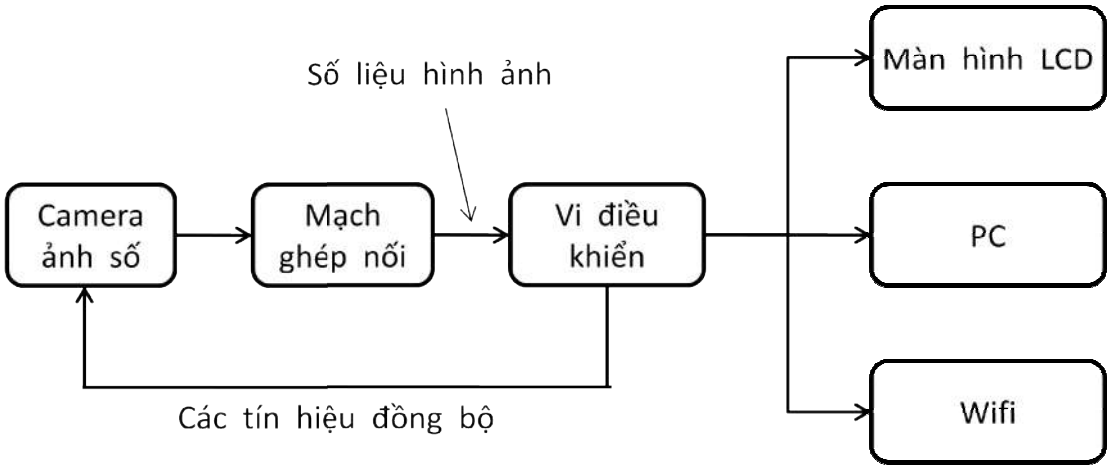
- Hiển thị hình ảnh theo thời gian thực ra màn hình LCD 3,2".

- Truyền dữ liệu ảnh về máy tính.

CHƯƠNG I: TỔNG QUAN VỀ HỆ THỐNG THU THẬP VÀ XỬ LÝ ẢNH DÙNG VI ĐIỀU KHIỂN STM32

1.1. Giới thiệu

Một hệ thống thu thập và xử lý ảnh dùng vi điều khiển có sơ đồ khối điển hình như sau:



Hình 1.1: Sơ đồ khối hệ thống thu thập xử lý ảnh

Hệ gồm có các thành phần sau:

- Camera ảnh số: là cảm biến biến đổi các pixel điểm ảnh (màu hoặc xám) ra thành các từ số liệu. Tốc độ biến đổi phải đủ nhanh để đáp ứng với yêu cầu xử lý theo thời gian thực.

- Mạch ghép nối: làm tương thích giữa camera và vi điều khiển.

- Vi điều khiển: có nhiệm vụ chính tạo xung nhịp cơ bản để đồng bộ các tín hiệu (mành VSYNC và dòng HYSNC) cho camera, thu thập các dữ liệu hình ảnh đưa vào bộ nhớ đệm ra màn hình (hoặc truyền ra các kênh thông tin, VD: cổng USART của PC, ra kênh Wifi-TCP/IP, v.v…).

- Màn hình: có nhiệm vụ hiển thị hình ảnh. Nếu là luồng video thi có thể cho các tốc độ đến 30fps.

Trong luận văn này, vi điều khiển được sử dụng là loại STM32F407VG Discovey thuộc dòng vi xử lý ARM hiện rất thông dụng cho các thiết bị di động. Các tính năng ưu việt của nó sẽ được trình bày ngay dưới đây.

1.1.1 Khái niệm về vi điều khiển

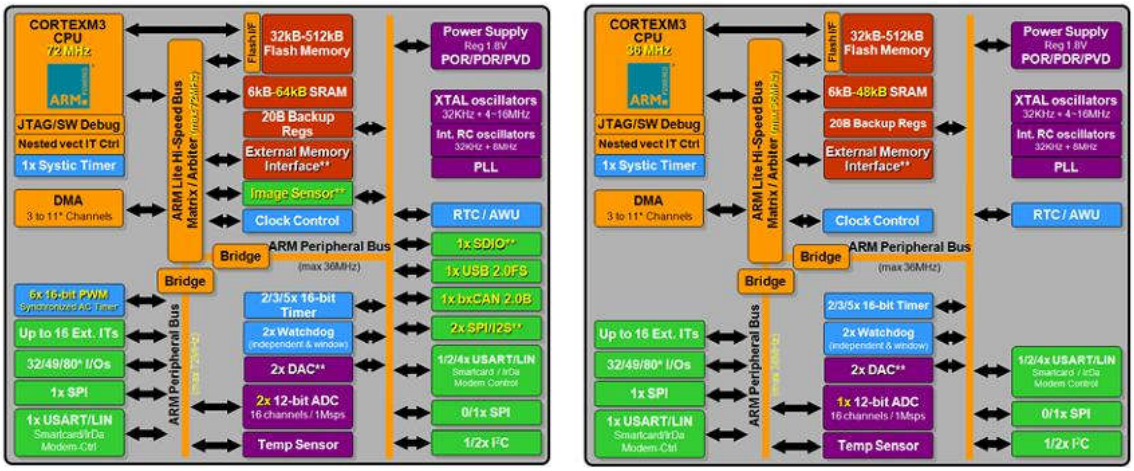
Vi điều khiển là một hệ thống được tích hợp trên một chip, bao gồm một vi xử lý có hiệu suất đủ dùng và giá thành thấp kết hợp với các khối ngoại vi như bộ nhớ, các mô đun vào/ra, các mô đun biến đổi ADC/DAC, bộ định thời...

1.1.2 Giới thiệu dòng vi điều khiển STM32 [1]

STM32 là vi điều khiển do hãng ST Microelectronic sản xuất dựa trên nền tảng lõi vi xử lý ARM Cortex®-M. Là một dòng sản phẩm vi xử điều khiển 32 bit kết hợp các ưu điểm về hiệu suất cao, khả năng xử lý thời gian thực, xử lý tín hiệu số, tiêu thụ ít năng lượng, hoạt động điện áp thấp, trong khi duy trì khả năng tích hợp đầy đủ và dễ dàng phát triển ứng dụng.

Vi điều khiển STM32 dựa trên một lõi tiêu chuẩn công nghiệp, có thể sử dụng nhiều công cụ và phần mềm để phát triển ứng dụng. Điều này làm cho dòng STM32 là sự lựa chọn lý tưởng đối với các dự án nhỏ hoặc cho thiết kế nền tảng.

ST đã đưa ra thị trường 4 dòng vi điều khiển dựa trên ARM7 và ARM9, nhưng STM32 là một bước tiến quan trọng trên đường cong chi phí và hiệu suất (price/performance), giá chỉ gần 1 Euro với số lượng lớn, STM32 là sự thách thức thật sự với các vi điều khiển 8 và 16 bit truyền thống. STM32 đầu tiên gồm 14 biến thể khác nhau, được phân thành hai dòng: dòng Performance có tần số hoạt động của CPU lên tới 72 Mhz và dòng Access có tần số hoạt động lên tới 36 Mhz. Các biến thể STM32 trong hai nhóm này tương thích hoàn toàn về cách bố trí chân (pin) và phần mềm, đồng thời kích thước bộ nhớ FLASH ROM có thể lên tới 512K và 64K SRAM. [1]



Hình 1.2: Kiến trúc của STM32 nhánh Performance và Access [1]

1.2.1 Sự tinh vi

Thoạt nhìn thì các ngoại vi của STM32 cũng giống như những vi điều khiển khác, như hai bộ chuyển đổi ADC, timer, I2C, SPI, CAN, USB và RTC. Tuy nhiên mỗi ngoại vi trên đều có rất nhiều đặc điểm thú vị. Ví dụ như bộ ADC 12 bit có tích hợp một cảm biến nhiệt

độ để tự động hiệu chỉnh khi nhiệt độ thay đổi và hỗ trợ nhiều chế độ chuyển đổi. Mỗi bộ định thời có 4 khối *capture compare* (dùng để bắt sự kiện với tính năng *input capture* và tạo dạng sóng ở ngõ ra với *output compare*), mỗi khối định thời có thể liên kết với các khối định thời khác để tạo ra một mảng các định thời chính xác hơn. Một bộ định thời chuyên hỗ trợ điều khiển động cơ với 6 đầu ra tín hiệu điều biến độ rộng xung PWM với dead time (khoảng thời gian được chèn vào giữa hai đầu tín hiệu xuất PWM bù nhau trong điều khiển mạch cầu H) lập trình được và một đường break input (khi phát hiện điều kiện dừng khẩn cấp) sẽ buộc tín hiệu PWM sang một trạng thái an toàn đã được cài sẵn. Ngoại vi nối tiếp SPI có một khối kiểm tổng (CRC) bằng phần cứng cho 8 và 16 word hỗ trợ tích cực cho giao tiếp thẻ nhớ SD hoặc MMC.

STM32 có hỗ trợ thêm tối đa 12 kênh DMA (Direct Memory Access). Mỗi kênh có thể được dùng để truyền dữ liệu đến các thanh ghi ngoại vi hoặc từ các thanh ghi ngoại vi với kích thước từ (word) dữ liệu truyền đi có thể là 8/16 hoặc 32 bit. Mỗi ngoại vi có thể có một bộ điều khiển DMA (DMA controller) đi kèm dùng để gửi hoặc truy vấn dữ liệu như yêu cầu. Một bộ phân xử bus nội (bus arbiter) và ma trận bus (bus matrix) tối thiểu hoá sự tranh chấp bus giữa truy cập dữ liệu thông qua CPU (CPU data access) và các kênh DMA. Điều đó cho phép các đơn vị DMA hoạt động linh hoạt, dễ dùng và tự động điều khiển các luồng dữ liệu bên trong vi điều khiển.

STM32 là một vi điều khiển tiêu thụ năng lượng thấp và đạt hiệu suất cao. Nó có thể hoạt động ở điện áp 2V, chạy ở tần số 72 MHz và dòng tiêu thụ chỉ có 36mA với tất cả các khối bên trong vi điều khiển đều được hoạt động. Kết hợp với các chế độ tiết kiệm năng lượng của Cortex, STM32 chỉ tiêu thụ 2µA khi ở chế độ Standby. Một bộ dao động nội RC 8 MHz cho phép chip nhanh chóng thoát khỏi chế độ tiết kiệm năng lượng trong khi bộ dao động ngoài đang khởi động. Khả năng nhanh đi vào và thoát khỏi các chế độ tiết kiệm năng lượng làm giảm nhiều sự tiêu thụ năng lượng tổng thể.

1.2.2. Khả năng an toàn

Ngày nay các ứng dụng hiện đại thường phải hoạt động trong môi trường khắt khe, đòi hỏi tính an toàn cao, cũng như đòi hỏi sức mạnh xử lý và càng nhiều thiết bị ngoại vi. Để đáp ứng các yêu cầu

khắt khe đó, STM32 cung cấp một số tính năng phần cứng hỗ trợ các ứng dụng một cách tốt nhất. Chúng bao gồm một bộ phát hiện điện áp thấp, một hệ thống bảo vệ xung Clock và hai bộ Watchdogs. Bộ đầu tiên là một Watchdog cửa sổ (windowed watchdog). Watchdog này phải được làm tươi trong một khung thời gian xác định. Nếu nhấn nó quá sớm, hoặc quá muộn, thì Watchdog sẽ kích hoạt. Bộ thứ hai là một Watchdog độc lập (independent watchdog), có bộ dao động bên ngoài tách biệt với xung nhịp hệ thống chính. Hệ thống bảo vệ xung nhịp có thể phát hiện lỗi của bộ dao động chính bên ngoài (thường là thạch anh) và tự động chuyển sang dùng bộ dao động nội RC 8 MHz.

1.2.3 Tính bảo mật

Một trong những yêu cầu khắt khe khác của thiết kế hiện đại là nhu cầu bảo mật mã chương trình để ngăn chặn sao chép trái phép phần mềm. Bộ nhớ Flash của STM32 có thể được khóa để chống truy cập đọc Flash thông qua cổng gỡ lỗi (Debug). Khi tính năng bảo vệ đọc được kích hoạt, bộ nhớ Flash cũng được bảo vệ chống ghi để ngăn chặn mã không tin cậy được chèn vào bảng vector ngắt. Hơn nữa bảo vệ ghi có thể được cho phép trong phần còn lại của bộ nhớ Flash. STM32 cũng có một đồng hồ thời gian thực và một khu vực nhỏ dữ liệu trên SRAM được nuôi nhờ nguồn pin. Khu vực này có một đầu vào chống giả mạo (anti-tamper input), có thể kích hoạt một sự kiện ngắt khi có sự thay đổi trạng thái ở đầu vào này. Ngoài ra một sự kiện chống giả mạo sẽ tự động xóa dữ liệu được lưu trữ trên SRAM được nuôi bằng nguồn pin.

1.2.4 Phát triển phần mềm

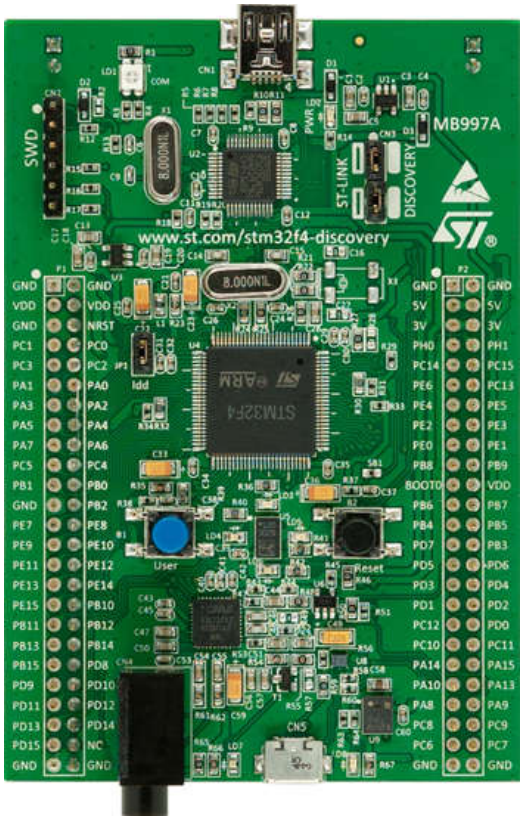
Các công cụ phát triển cho ARM hiện có đã được hỗ trợ tập lệnh Thumb-2 và dòng Cortex. Ngoài ra ST cũng cung cấp một thư viện điều khiển thiết bị ngoại vi, một bộ thư viện phát triển USB như là một thư viện ANSI C và mã nguồn tương thích với các thư viện trước đó được công bố cho vi điều khiển STR7 và STR9. Có rất nhiều hệ điều hành thời gian thực RTOS (Real Time Operating System) mã nguồn mở và thương mại và middleware (TCP/IP, hệ thống tập tin, v.v.) hỗ trợ cho họ Cortex. Dòng Cortex-M3 cũng đi kèm với một hệ thống gỡ lỗi hoàn toàn mới gọi là CoreSight. Truy cập vào hệ thống CoreSight thông qua cổng truy cập gỡ lỗi (Debug Access Port), cổng này hỗ trợ kết nối chuẩn JTAG hoặc giao diện 2 dây (serial wire-2

Pin), cũng như cung cấp trình điều khiển chạy gỡ lỗi, hệ thống CoreSight trên STM32 cung cấp hệ thống điểm truy cập (data watchpoint) và một công cụ theo dõi (instrumentation trace). Công cụ này có thể gửi thông tin về ứng dụng được lựa chọn đến công cụ gỡ lỗi. Điều này có thể cung cấp thêm các thông tin gỡ lỗi và cũng có thể được sử dụng trong quá trình thử nghiệm phần mềm.

STM32 có sẵn một bộ thư viện ngoại vi chuẩn và mẫu, ví dụ hỗ trợ lập trình mà không cần kiến thức chuyên sâu hay hiểu rõ datasheet của chip, giúp nhanh chóng tập trung vào việc viết chương trình, tiết kiệm thời gian phát triển sản phẩm.

1.1.3 Giới thiệu kit STM32F4 Discovery

Luận văn sử dụng Kit STM32F407VG Discovery cho bắt ảnh qua Camera OV7670.



Hình 1.3: Kit STM32F407VG Discovery

Bộ kit STM32F4-DISCOVERY với vi điều khiển hiệu suất cao STM32F407VGT6, cho phép người dùng dễ dàng phát triển các ứng dụng xử lý tín hiệu số (hình ảnh, video…). Nó bao gồm một công cụ ST-LINK tích hợp sẵn trên bảng mạch giúp nạp chương trình, gỡ lỗi nhanh chóng.

Các tính năng chính

- Vi điều khiển 32-bit ARM Cortex®-M4 STM32F407VGT6 với lõi FPU hỗ trợ xử lý tính toán dấu phẩy động, 1-MB bộ nhớ Flash, 192 Kbyte RAM.

- On-board ST-LINK/V2 trên STM32F4-DISCOVERY giúp nạp chương trình, gỡ lỗi.

- Nguồn điện cung cấp cho bảng mạch: thông qua cổng USB hoặc từ một nguồn cung cấp điện áp 5V bên ngoài.

- Từ bảng mạch, có thể cấp nguồn 3,3 V và 5 V cho các ứng dụng.

- Cảm biến chuyển động LIS302DL, ST MEMS 3 trục gia tốc.

- Cảm biến âm thanh MP45DT02 ST-MEMS, mic cảm biến âm thanh vô hướng kỹ thuật số.

- Bộ chuyển đổi DAC âm thanh CS43L22.

- Tám đèn LED:

+ LD1 (đỏ / xanh lá cây) để giao tiếp USB

+ LD2 (màu đỏ) báo hiệu nguồn 3,3 V on

+ Bốn đèn LED màu: LD3 (màu cam), LD4 (màu xanh lá cây), LD5 (màu đỏ) và LD6 (màu xanh dương).

+ Hai USB OTG LED LD7 (màu xanh lá cây) VBUS và LD8 (màu đỏ).

- Hai nút bấm (nút bấm User màu xanh, nút bấm Reset màu đen).

- OTG FS USB với cổng nối micro-AB.

- Header mở rộng cho tất cả LQFP100 I/O.

- Phần mềm miễn phí bao gồm một loạt các ví dụ, sử dụng thư viện chuẩn của ST.

1.2. Giới thiệu Camera OV7670

Luận văn sử dụng Camera OV7670 no FIFO (First In, First Out) là loại camera giá rẻ nhưng có tính năng đủ cao để sử dụng trong thí nghiệm.

1.2.1. Giới thiệu chung

OV7670 là một cảm biến ảnh kết hợp với xử lý tín hiệu số (DSP), cho độ phân giải VGA 640x480, tốc độ khung hình lên tới 30 fps. Hình ảnh thu nhận được có thể được tiền xử lý bởi khối DSP trước khi được truyền đi. Việc cấu

hình chế độ hoạt động của OV7670 được cấu hình qua Bus điều khiển camera tuần tự SCCB (Serial Camera Control Bus).

Hình 1.4: Hình ảnh Camera OV7670 no FIFO

Chức năng các chân tín hiệu Camera OV7670 no FIFO trong hình 1.4 được mô tả trong Bảng 1.1 dưới đây:

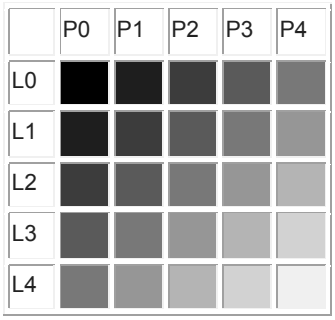
Bảng 1.1: Chức năng các chân tín hiệu của Camera OV7670

Chân	**Loại**	**Mô tả**
VDD	Nguồn	Nguồn 3,3V
GND	Nguồn	Ground
SDIOC	Đầu vào	Tín hiệu xung clock SCCB
SDIOD	Đầu vào	Tín hiệu SCCB data
VSYNC	Đầu ra	Xung đồng bộ mành
HREF	Đầu ra	Xung đồng bộ dòng
PCLK	Đầu ra	Tần số Pixel clock
XCLK	Đầu ra	Tần số cấp cho Camera hoạt động
D0-D7	Đầu ra	Tín hiệu video ra song song 8 bit
RESET	Đầu vào	Tín hiệu Reset (tích cực mức thấp)
PWDN	Đầu vào	Tắt nguồn Power down (tích cực mức cao)

1.2.2 Tín hiệu hình ảnh của Camera OV7670 [11]

Trước khi đi vào mô tả tín hiệu của Camera OV7670, cần tìm hiểu khái niệm video và hình ảnh được biểu diễn ở định dạng kỹ thuật số. [11]

Một đoạn video là một chuỗi các khung hình, một khung hình (frame) là một hình ảnh tĩnh chụp tại một thời điểm nhất định. Một khung hình được chia làm các dòng (line), mỗi dòng được chia thành các điểm ảnh (pixel). Một điểm ảnh là một phần nhỏ của một hình ảnh kỹ thuật số, và nó trông giống như một dấu chấm màu.



Hình 1.5: Ảnh 5x5 pixel

Ví dụ, hình 1.5 [11] có 5 dòng, mỗi dòng có 5 pixel. Điều này có nghĩa là hình ảnh có độ phân giải điểm ảnh 5x5. Đây là ảnh đơn sắc (đa mức xám), ngoài ra cũng có những ảnh màu. Các màu của hình ảnh có thể được mã hóa trong các định dạng khác nhau như RGB, YUV.

1.2.2.1 Định dạng ảnh đa mức xám (Monochrome)

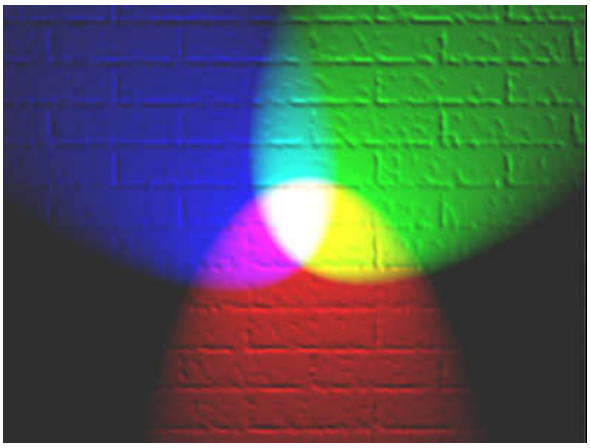
Trong ảnh đa mức xám, mỗi điểm ảnh được lưu trữ bởi 8 bit số liệu, đại diện cho mức độ xám trong dải từ 0 đến 255. Giá trị 0 là màu đen, 255 là màu trắng và các giá trị trung gian là màu xám.

1.2.2.2 Định dạng ảnh RGB

Trong thực tế, màu bất kỳ có thể được tạo bởi sự kết hợp của ba nguồn ánh sáng đỏ (Red), xanh lá cây (Green) và màu xanh dương (Blue) với những cường độ khác nhau. Cách tiếp cận này được gọi là mô hình màu RGB. Sử dụng mô hình này, mỗi điểm ảnh phải lưu trữ ba giá trị cường độ của ánh sáng đỏ, xanh lá cây và xanh dương.

Các định dạng phổ biến nhất là RGB888, ở định dạng này mỗi điểm ảnh được lưu trữ trong 24 bit, mỗi kênh màu đỏ, xanh lá cây và

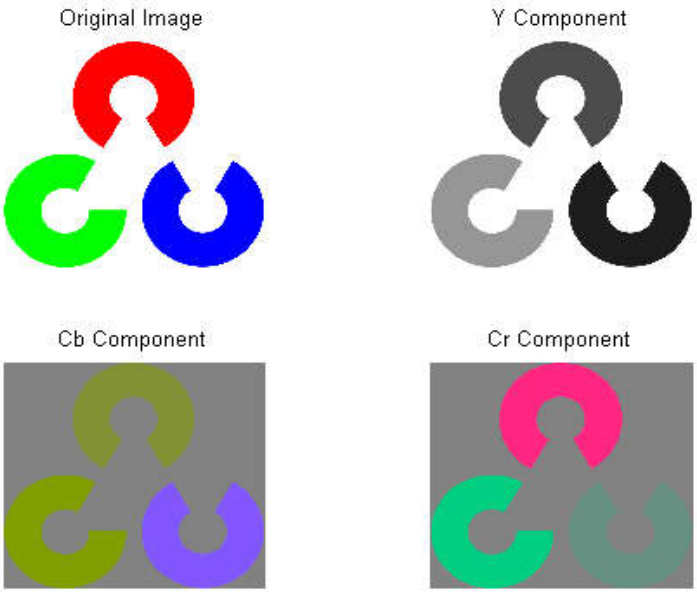
màu xanh dương được lưu trữ trong 8 bit. Cường độ của mỗi ánh sáng thành phần có thể trong dải từ 0 đến 255, trong đó 0 là không có ánh sáng, và 255 là cường độ sáng tối đa.



Hình 1.6: Mô hình màu RGB (hình ảnh từ wikipedia).

Các định dạng RGB được sử dụng bởi OV7670 là RGB565, RGB555 và RGB444. Sự khác biệt với định dạng RGB888 là số bit được gán cho mỗi kênh. Ví dụ, trong định dạng RGB565, kênh màu đỏ được lưu trữ với 5 bit, kênh màu xanh lá cây là 6 bit và kênh màu xanh dương là 5 bit. Các định dạng RGB565, RGB555 và RGB444 tốn ít bộ nhớ khi lưu trữ nhưng làm giảm số lượng màu sắc so với định dạng RGB888.

1.2.2.3 Định dạng ảnh YCbCr



Hình 1.7: Phân giải của một hình ảnh thành các thành phần Y, Cb và Cr

YCbCr là một định dạng trong đó một màu RGB có thể được mã hóa. Y hoặc độ sáng thành phần là lượng ánh sáng trắng của một màu sắc, và Cb và Cr là những thành phần chroma, tương ứng mã hóa các cấp độ màu xanh và màu đỏ tương đối so với các thành phần độ sáng.

Hình 1.7 thể hiện hình ảnh phân giải ảnh màu gốc YcbCr thành cách thành phần Y, Cb, Cr. Có thể thấy kênh Y mã hóa các mức độ màu xám của hình ảnh. Vì vậy, cách dễ nhất để có được một hình ảnh đa mức xám từ OV7670 là trích xuất kênh Y của định dạng YCbCr.

Giống như định dạng RGB, định dạng YCbCr cũng lưu trữ mỗi kênh là 8 bit (0-255), có thể chuyển đổi từ YCbCr sang RGB sử dụng các biểu thức sau đây:

$$R = Y + 1{,}402\ (Cr - 128)$$

$$G = Y - 0{,}34414\ (Cb - 128) - 0{,}71414\ (Cr - 128)$$

$$B = Y + 1{,}772\ (Cb - 128)$$

Camera OV7670 sử dụng định dạng YCbCr422, định dạng này được lưu trữ thành các từ theo Bảng 1.2

Bảng 1.2: Dữ liệu ảnh được lưu trữ thành từ (4 byte)

	Byte 0	Byte 1	Byte 2	Byte 3
Word 0	Cb0	Y0	Cr0	Y1
Word 1	Cb2	Y2	Cr2	Y3
Word 2	Cb4	Y4	Cr4	Y5

Hoặc tương đương, các dữ liệu đến theo thứ tự trong Bảng 1.3:

Bảng 1.3: Thứ tự dữ liệu đến dạng YCbCr422

Thứ tự	Byte
1st	Cb0
2nd	Y0
3rd	Cr0
4th	Y1
5th	Cb2

6th	Y2
7th	Cr2
8th	Y3
...	...

Bảng 1.4: Các điểm ảnh YCbCr422

Pixel 0	Y0 Cb0 Cr0
Pixel 1	Y1 Cb0 Cr0
Pixel 2	Y2 Cb2 Cr2
Pixel 3	Y3 Cb2 Cr2
Pixel 4	Y4 Cb4 Cr4
Pixel 5	Y5 Cb4 Cr4

Các điểm ảnh thực tế như Bảng 1.4, mỗi điểm ảnh gồm 3 byte (ví dụ pixel 0 gồm 3 byte Y0, Cb0 và Cr0) như trong các định dạng RGB. Nhưng trong định dạng YCbCr422, các kênh Cb và Cr được chia sẻ giữa hai điểm ảnh liên tiếp (ví dụ pixel 0 và 1 dùng chung phần Cb0 và Cr0). Do đó hai điểm ảnh được "nén" thành 4 byte hoặc 32 bit, điều này có nghĩa là trung bình mỗi điểm ảnh được lưu trữ như là 2 byte hoặc 16 bit. Từ ví dụ trên, 3 từ (12 byte) lưu trữ 6 pixel.

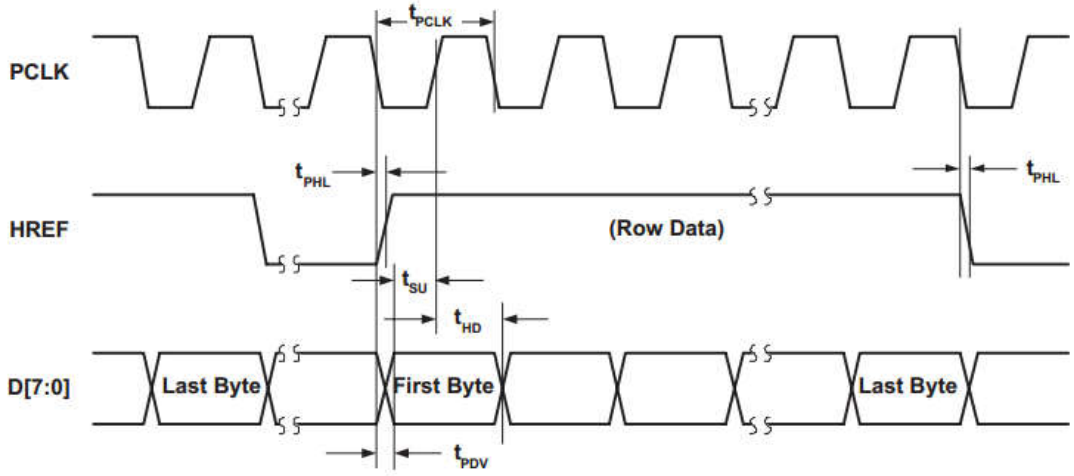
Ưu điểm của định dạng YCbCr là kênh Y là hình ảnh đa mức xám, trong khi ở định dạng RGB sẽ cần lấy trung bình của 3 kênh để có được hình ảnh đa mức xám.

1.2.2.4 Tín hiệu video của Camera OV7670

Trước hết, để cho Camera OV7670 hoạt động, cần cấp một tín hiệu xung clock đến chân XCLK. Theo datasheet, xung clock này phải có tần số trong dải từ 10 ÷ 48 MHz.

Nếu vi điều khiển có đầu ra timer hoặc dao động, có thể sử dụng để cấp xung clock cho OV7670. Nếu vi điều khiển không có khả năng tạo ra xung clock thích hợp, có thể sử dụng một nguồn dao động ngoài cấp cho OV7670.

Sau khi có tín hiệu xung clock đến chân XCLK, Camera OV7670 sẽ điều khiển xung đồng bộ VSYN, HREF, PCLK và gửi dữ liệu D0÷D7 dạng song song 8 bit. Giản đồ thời gian các tín hiệu xung clock đồng bộ dòng được mô tả trong hình 1.8. [5]



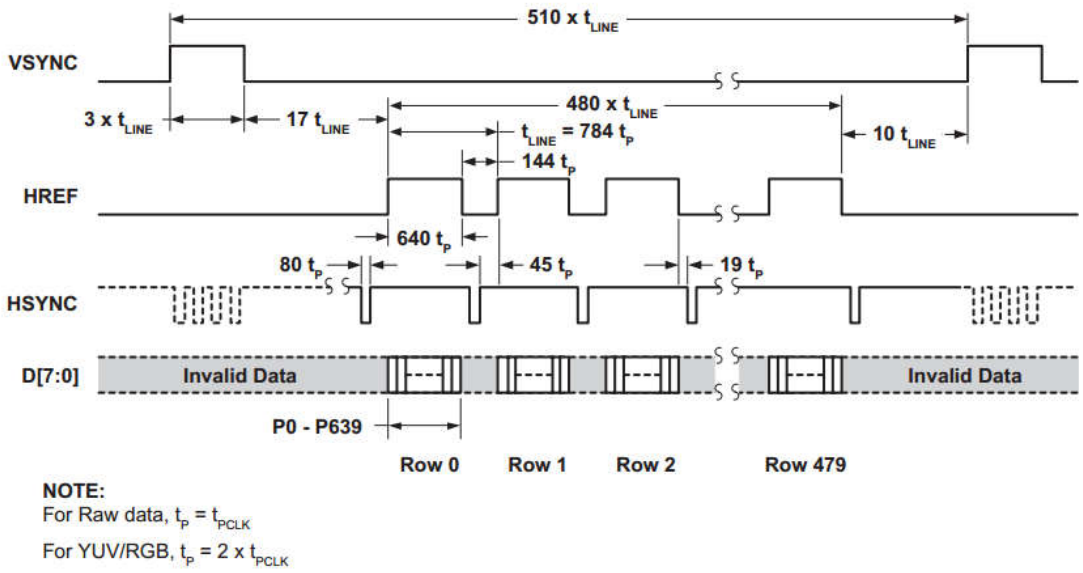
Hình 1.8: Đồng bộ dòng

Điều đầu tiên cần chú ý, dữ liệu D0÷D7 phải được lấy mẫu tại sườn lên của tín hiệu PCLK. Thứ hai, D0÷D7 phải được lấy mẫu chỉ khi HREF ở mức cao. Ngoài ra, sườn lên của tín hiệu HREF báo hiệu sự bắt đầu của một dòng, sườn xuống của tín hiệu HREF báo hiệu kết thúc của một dòng.

Tất cả các byte D0÷D7 được lấy mẫu khi HREF ở mức cao, tương ứng với các điểm ảnh trong một dòng. Lưu ý rằng một byte không phải là một điểm ảnh, nó phụ thuộc vào định dạng lựa chọn. Theo cài đặt mặc định của Camera OV7670, định dạng hình ảnh là YCbCr422, điều này có nghĩa là trung bình hai byte tương ứng với một điểm ảnh (16 bit/pixel).

Theo Hình 1.9 [5], sườn xuống của tín hiệu VSYNC bắt đầu một frame, sườn lên của tín hiệu VSYNC kết thúc một frame.

Khi tín hiệu HSYNC ở mức cao, khối thu nhận 640 pixel, tương ứng một dòng, 480 dòng tương ứng một frame được thu nhận khi tín hiệu VSYNC ở mức thấp.



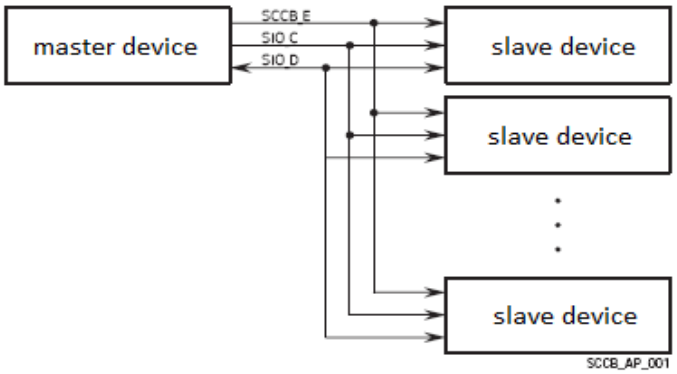
Hình 1.9: Mô tả tín hiệu một frame ảnh VGA (640x480)

Trên đây đã mô tả toàn bộ quá trình lấy một khung hình (frame). Theo mặc định, tín hiệu PCLK sẽ có cùng một tần số với XCLK, tuy nhiên bằng cách sử dụng SCCB để cấu hình tỉ lệ chia tần số prescalers và vòng khóa pha PLL, chúng ta có thể tạo ra tín hiệu PCLK có tần số khác nhau.

Thông thường, một tín hiệu PCLK 24 MHz sẽ cho tốc độ khung hình 30 fps, tín hiệu PCLK 12 MHz cho tốc độ khung hình là 15 fps. Điều này độc lập với định dạng hình ảnh (VGA, CIF, QCIF, …).

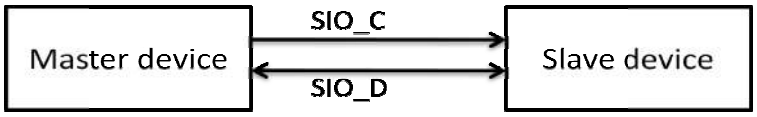
1.2.3. Bus điều khiển camera tuần tự SCCB

Bus điều khiển camera tuần tự SCCB do hãng OmniVision Technologies phát triển để điều khiển các chức năng của dòng cảm biến camera OmniVision, là một chuẩn bus tuần tự 3 dây, có sự tương thích với giao thức I2C (Inter-Integrated Circuit). [8]



Hình 1.10: Sơ đồ khối chức năng SCCB tổng quát sử dụng 3 dây

Trong các ứng dụng master chỉ làm việc với một slave, SCCB có thể hoạt động với hai dây tín hiệu, điều này giúp giảm số chân linh kiện (bỏ chân SCCB_E)



Hình 1.11: Sơ đồ khối chức năng SCCB tổng quát sử dụng 2 dây

1.2.3.1 Chức năng của các chân tín hiệu trong SCCB

Tín hiệu SCCB_E

Là tín hiệu một chiều, kích hoạt ở mức thấp, điều khiển bởi master. Báo hiệu bắt đầu hoặc ngừng truyền dữ liệu.

Mức thay đổi tín hiệu từ cao xuống thấp báo hiệu bắt đầu truyền dữ liệu. Mức thay đổi tín hiệu từ thấp lên cao báo hiệu ngừng truyền dữ liệu.

SCCB_E phải giữ mức thấp trong suốt quá trình truyền dữ liệu. SCCB_E ở mức logic 1 khi hệ thống ở trạng thái ide.

Tín hiệu SIO_C

Là tín hiệu một chiều, tích cực ở mức cao, điều khiển bởi master. Báo hiệu mỗi khi truyền một bít. Truyền dữ liệu bắt đầu khi SIO_C ở mức logic 0.

Mức logic 1 của SIO_C trong quá trình truyền dữ liệu báo hiệu 1 bit đã truyền xong. Vì vậy, tín hiệu SIO_D chỉ có thể truyền khi SIO_C ở mức 0. Master đặt SIO_C ở mức logic 1 khi bus ở trạng thái ide.

Chu kỳ truyền 1 bit thông thường khoảng 10μs => tần số 100 KHz.

Tín hiệu SIO_D

Là tín hiệu dữ liệu hai chiều, có thể được điều khiển bởi master hoặc slave. Một bít truyền được nhận biết bằng một mức logic 1 của SIO_C.

1.2.3.2 Báo hiệu truyền dữ liệu trong SCCB

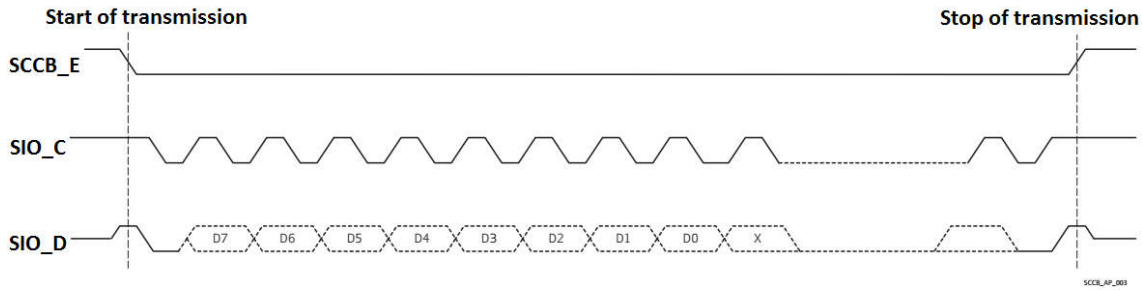
Thủ tục truyền dữ liệu trong giao thức SCCB gồm 03 bước:

1- Báo hiệu bắt đầu truyền dữ liệu.

2- Truyền dữ liệu.

3- Báo hiệu kết thúc truyền dữ liệu.

Giản đồ thời gian của quá trình truyền dữ liệu của SCCB 3 dây tổng quát thể hiện trong hình 1.12. [8]

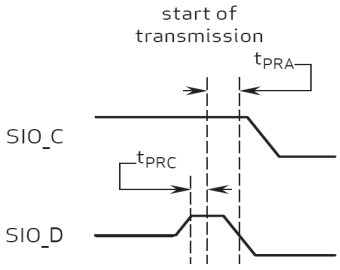


Hình 1.12: Quá trình truyền dữ liệu của SCCB 3 dây

Trong luận văn sử dụng truyền dữ liệu qua SCCB 2 dây, sau đây sẽ tập trung mô tả giao thức truyền SCCB 2 dây.

Báo hiệu bắt đầu truyền tín hiệu (Start):

Mức tín hiệu SIO_D, SIO_C ở mức logic 1 để nhận biết trạng thái đường truyền.

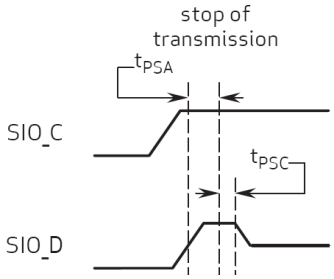


Hình 1.13: Tín hiệu báo hiệu Start

Start: Sau khi mức tín hiệu SIO_D chuyển trạng thái từ 1 xuống 0, tín hiệu SIO_C chuyển trạng thái từ 1 xuống 0 để báo hiệu bắt đầu truyền.

Báo hiệu ngừng truyền dữ liệu (Stop):

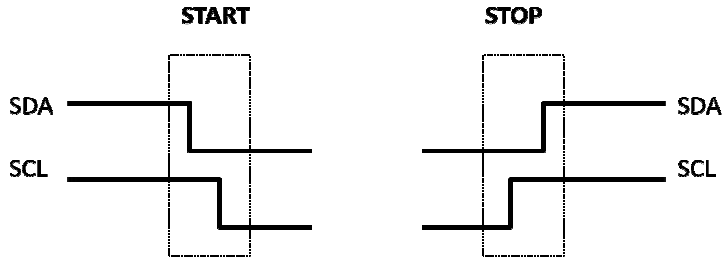
Sau khi mức tín hiệu SIO_C chuyển trạng thái từ 0 lên 1, tín hiệu SIO_D chuyển trạng thái từ 0 lên 1 để báo hiệu ngừng truyền dữ liệu.



Hình 1.14: Tín hiệu báo hiệu Stop

Stop: Tín hiệu SIO_D từ mức logic 0 lên 1 khi tín hiệu SIO_C ở mức logic 1

So sánh với tín hiệu báo hiệu bắt đầu và kết thúc truyền dữ liệu của I2C



Hình 1.15: Tín hiệu báo hiệu Start/Stop của I2C

Start: Tín hiệu SDA từ mức logic 1 xuống 0 khi tín hiệu SCL ở mức logic 1

Stop: Tín hiệu SDA từ mức logic 0 lên 1 khi tín hiệu SCL ở mức logic 1

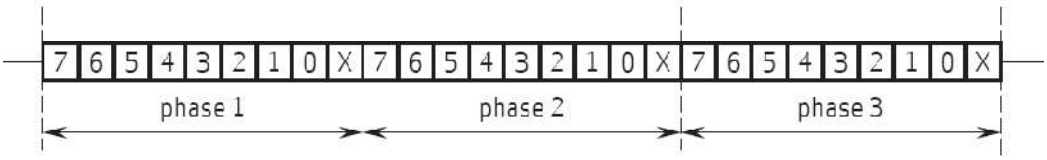
So sánh cho thấy tín hiệu báo hiệu truyền của SCCB giống I2C.

Tín hiệu SIO_C tương đương với tín hiệu SCL

Tín hiệu SIO_D tương đương với tín hiệu SDA

1.2.3.3 Chu kỳ truyền dữ liệu trong SCCB

Đơn vị truyền dữ liệu cơ bản trong SCCB được gọi là pha, một pha truyền dữ liệu gồm có 9 bít: 8 bit dữ liệu truyền tuần tự, bit thứ 9 là Don't Care bit hoặc NA bit phụ thuộc vào việc truyền dữ liệu là đọc hay ghi dữ liệu. Mục đích của bít thứ 9 là để nhận biết truyền dữ liệu thành công. Bit có trọng số cao MSB luôn được truyền đầu tiên.



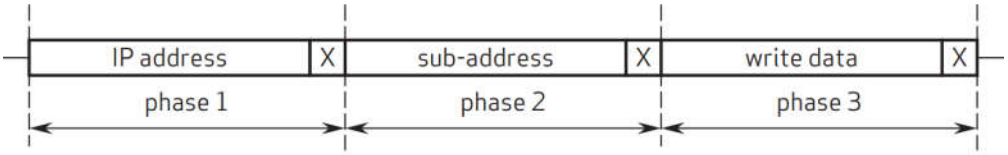
Hình 1.16: Pha truyền dữ liệu trong SCCB

Dưới đây sẽ mô tả 3 loại truyền dữ liệu. [8]

- Chu kỳ ghi dữ liệu 3 pha.
- Chu kỳ ghi dữ liệu 2 pha.
- Chu kỳ đọc dữ liệu 2 pha.

1.2.3.4 Chu kỳ ghi dữ liệu 3 pha

Là một chu kỳ ghi dữ liệu đầy đủ, master thực hiện ghi một byte dữ liệu lên thanh ghi của slave.



Hình 1.17: Chu kỳ ghi dữ liệu 3 pha trong SCCB

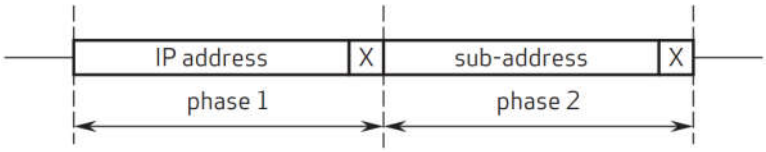
Pha 1: IP address => truyền địa chỉ slave.

Pha 2: sub address => truyền địa chỉ thanh ghi của slave.

Pha 3: write data => ghi dữ liệu 8 bit vào thanh ghi của slave.

Bít thứ 9 trong cả 3 pha là Don't Care bit.

1.2.3.5 Chu kỳ ghi dữ liệu 2 pha



Hình 1.18: Chu kỳ ghi dữ liệu 2 pha trong SCCB

Chu kỳ ghi dữ liệu 2 pha phải theo sau bởi một chu kỳ đọc dữ liệu 2 pha. Mục đích của chu kỳ ghi dữ liệu 2 pha là xác định thanh ghi của slave để master đọc trong chu kỳ đọc 2 pha tiếp theo.

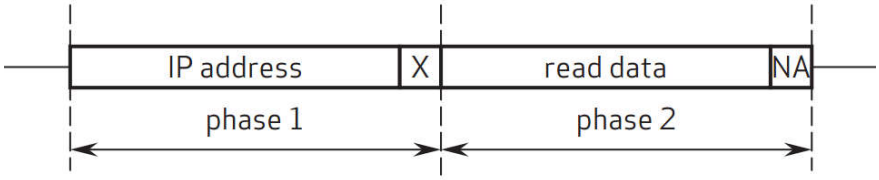
Pha 1: IP address => truyền địa chỉ slave.

Pha 2: sub address => truyền địa chỉ thanh ghi của slave.

Bít thứ 9 trong cả 2 pha là Don't Care bit.

1.2.3.6 Chu kỳ đọc dữ liệu 2 pha

Bản thân chu kỳ đọc 2 pha không thể nhận biết được địa chỉ thanh ghi của slave. Do đó, trước khi thực hiện chu kỳ đọc dữ liệu 2 pha, phải có một chu kỳ ghi dữ liệu 3 pha hoặc 2 pha để slave nhận biết địa chỉ thanh ghi cần đọc dữ liệu.



Hình 1.19: Chu kỳ đọc dữ liệu 2 pha trong SCCB

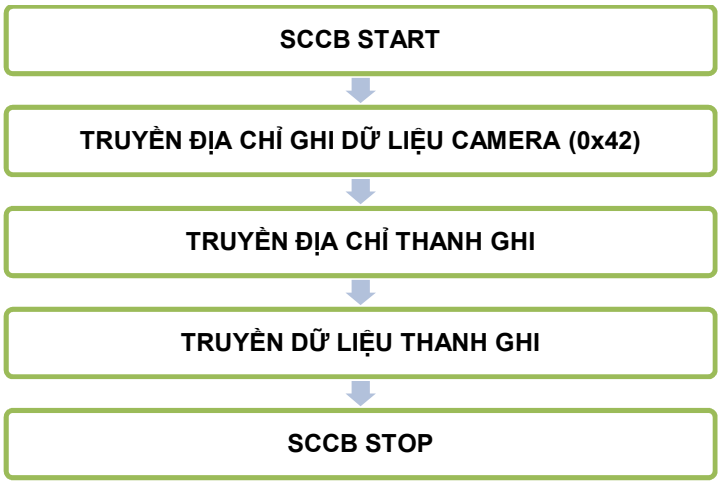
Pha 1: IP address => truyền địa chỉ slave. Bít thứ 9 trong pha 1 là Don't Care bit.

Pha 2: read data => đọc dữ liệu từ slave. Bit thứ 9 trong pha 2 của chu kỳ đọc là NA bit. Master phải kéo tín hiệu NA bit lên mức 1.

So sánh bit thứ 9 trong giao thức I2C với SCCB: đều dùng để để báo hiệu truyền dữ liệu thành công. Trong giao thức I2C, bit thứ 9 là bit ACK luôn ở mức logic 0; trong giao thức SCCB khi master truyền dữ liệu, bít thứ 9 là Don't care bit có mức logic 0, khi master đọc dữ liệu, bit thứ 9 là NA bit có mức logic 1.

1.2.3.7 Ghi dữ liệu vào thanh ghi OV7670

Để ghi dữ liệu vào thanh ghi của Camera OV7670 theo giao thức SCCB, sử dụng chu kỳ truyền dữ liệu 3 pha. Sơ đồ thuật toán như hình 1.20:



Hình 1.20: Ghi dữ liệu vào thanh ghi OV7670

1.2.3.8 Đọc dữ liệu vào thanh ghi OV7670.

Để đọc dữ liệu thanh ghi của Camera OV7670 theo giao thức SCCB, sử dụng hai chu kỳ truyền dữ liệu:

- Chu kỳ ghi dữ liệu 2 pha

- Chu kỳ đọc dữ liệu 2 pha

Sơ đồ thuật toán như hình 1.21:

Hình 1.21: Đọc dữ liệu thanh ghi OV7670

1.2.4 Cấu hình hoạt động của Camera OV7670

Camera OV7670 có nhiều thanh ghi cho phép cấu hình chế độ hoạt động như lựa chọn định dạng hình ảnh, độ phân giải, tốc độ khung hình, tiền xử lý ảnh, cân bằng trắng… Dưới đây sẽ mô tả một số thanh ghi quan trọng trong việc cấu hình chế độ hoạt động của Camera OV7670.

1.2.4.1 Cài đặt tần số dao động nội cho Camera OV7670 hoạt động

Để Camera OV7670 hoạt động, cần cấp xung clock từ bên ngoài đến chân XCLK, đầu tiên tín hiệu clock này được qua vòng khóa pha PLL để nhân tần số, sau đó qua bộ chia để tạo ra xung clock để Camera OV7670 làm việc.

Bộ nhân PLL được điều khiển bởi thanh ghi DBLV có địa chỉ 0x6B.

Bộ chia được điều khiển bởi thanh ghi CLKRC có địa chỉ 0x11.

Bảng 1.5: Các thanh ghi cài đặt tần số dao động nội Camera

Thanh ghi	**Địa chỉ**	**Giá trị mặc định**	**Mô tả**
CLKRC	0x11	0x80	Bit[6] = 0: cho phép đặt tỷ lệ chia tần số Bit[6] = 1: sử dụng xung clock ngoài Bit[5:0]: hệ số chia tần số, trong dải từ [0 0000] đến [1 1111] , tính theo công thức 2*(CLKRC[5:0]+1)

DBLV	0x6B	0x0A	Bit[7:6]: Điều khiển PLL để nhân tần số 00: Bypass PLL, giữ nguyên tần số 01: Input clock × 4 10: Input clock × 6 11: Input clock × 8

Tần số làm việc của Camera được tính như sau:

$f_{INT\ CLK} = f_{CLK} \times$ PLL_Multiplier / 2* (CLKRC[5:0]+1)

Ví dụ: tần số 16 MHz bên ngoài cấp cho Camera ở chân XCLK, để nhận được tần số dao động nội 24 MHz cấp cho Camera hoạt động, cần cấu hình như sau:

Đặt Bit[6] thanh ghi CLKRC về 0 để cho phép chia tỷ lệ.

Đặt Bit[5:0] thanh ghi CLKRC bằng 1 (00001) để thực hiện chia 4.

Đặt Bit[7:6] thanh ghi DBLV về 10 để PLL nhân 6.

```
// Tan so STM32F4 cap cho camera 16MHz
SCCB_write_reg(0x6B,0x80);  // nhan tan so voi 6
SCCB_write_reg(0x11, 0x01);// chia tan so cho 2(1+1)
// Tan so lam viec cua camera = 16MHz*6/2(1+1) = 24MHz
```

$f_{INT\ CLK}$ = f_{CLK} x PLL_Multiplier / (CLKRC[5:0]+1)

= 16 MHz × 6 / 4 = 24 MHz

Thông thường, tần số pixel clock ra tại chân PCLK Camera OV7670 có cùng tần số với tần số $f_{INT\ CLK}$. Bằng cách cài đặt một số thanh ghi, chúng ta có thể thay đổi tần số PCLK qua đó thay đổi tốc độ khung hình/giây.

1.2.4.2 Định dạng hình ảnh

Camera OV7670 hỗ trợ 4 định dạng hình ảnh: YCbCr, RGB565/RGB555, Bayer raw RGB và Processed raw RGB. Định dạng hình ảnh ra có thể được cấu hình bằng cách thiết lập thanh ghi các bit của hai thanh ghi COM7 (địa chỉ 0x12) và thanh ghi COM15 (địa chỉ 0x40) theo Bảng 1.6 [6]

Bảng 1.6: Thiết lập định dạng ảnh cho Camera OV7670

Định dạng hình ảnh	**Pixel Data Output**	**Cài đặt thanh ghi**			
		COM7[2]	COM7[0]	COM15[5]	COM15[4]
Raw Bayer RGB	8-bit R or 8-bit G or 8-bit B	0	1	x	0
Processed BayerRGB	8-bit R or 8-bit G or 8-bit B	1	1	x	0
YUV/YCbCr 4:2:2	8-bit Y, 8-bit U or 8-bit Y, 8-bit V	0	0	x	0
GRB 4:2:2	8-bit G, 8-bit R or 8-bit G, 8-bit B	1	0	x	0
RGB565	5-bit R, 6-bit G, 5-bit B	1	0	0	1
RGB555	5-bit R, 5-bit G, 5-bit B	1	0	1	1

Căn cứ tài liệu Camera OV7670 [6], có thể đặt giá trị một số thanh ghi để có định dạng hình ảnh RGB565 và YUV422 như sau:

Định dạng hình ảnh RGB565:

```
SCCB_write_reg(0x12, 0x04); // COM7, RGB format
SCCB_write_reg(0x40, 0xD0); // COM15, RGB565
SCCB_write_reg(0x04, 0x0);  // COM1, disable CCIR656
SCCB_write_reg(0x8C, 0x0);  // disable RGB444
```

Định dạng hình ảnh YUV422:

```
SCCB_write_reg(0x12, 0x0); // COM7, YUV mode
SCCB_write_reg(0x40, 0x0); // COM15
SCCB_write_reg(0x04, 0x0); // COM1, disable CCIR656
SCCB_write_reg(0x8C, 0x0); // disable RGB444
```

Với định dạng YUV, mặc định thứ tự tín hiệu ra là YUYV, nếu cần thay đổi thứ tự tín hiệu ra, thay đổi giá trị các bit thanh ghi COM13[1] địa chỉ 0x3D và thanh ghi TSLB[3] địa chỉ 0x3A theo Bảng 1.7

Bảng 1.7: Thứ tự tín hiệu YUV

Bit TSLB[3] (0x3A)	Bit COM13[1] (0x3D)	Thứ tự
0	0	Y U Y V
0	1	Y V Y U
1	0	U Y V Y
1	1	V Y U Y

1.2.4.3 Cài đặt độ phân giải hình ảnh

Camera OV7670 có độ phân giải mặc định là VGA (640x480), trong thanh ghi COM7 (0x12) có một số độ phân giải đã định nghĩa trước như QVGA, CIF và QCIF. Để lựa chọn các độ phân giải này, cài đặt các bit thanh ghi COM7 như Bảng 1.8 [6]

Bảng 1.8: Thiết lập độ phân giải QVGA, CIF, QCIF

Độ phân giải hình ảnh	Cài đặt thanh ghi		
	COM7[5]	COM7[4]	COM7[3]
QVGA (320x240)	0	1	0
CIF (352x288)	1	0	0
QCIF (176x144)	0	0	1

Căn cứ vào Bảng 1.7, 1.8, có thể thiết lập các giá trị khác nhau của thanh ghi COM7 để chọn độ phân giải và định dạng ảnh:

```
SCCB_write_reg(0x12, 0x0 );  // VGA, YUV
SCCB_write_reg(0x12, 0x4 );  // VGA, RGB
SCCB_write_reg(0x12, 0x14);  // QVGA, RGB
SCCB_write_reg(0x12, 0x10);  // QVGA, YUV
SCCB_write_reg(0x12, 0x8 );  // QCIF, YUV
SCCB_write_reg(0x12, 0x0C);  // QCIF, RGB
```

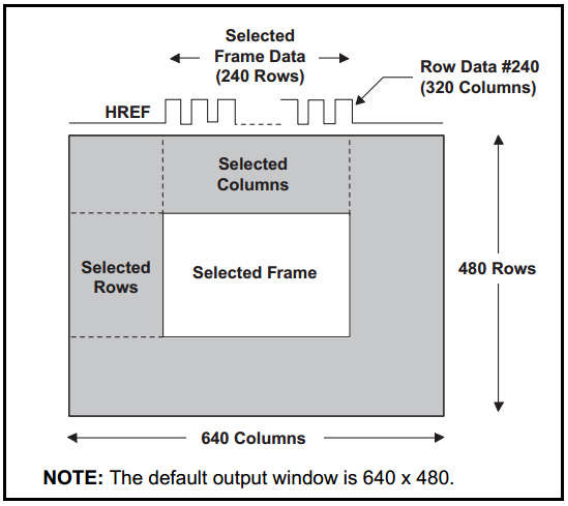
Các độ phân giải này đã được định nghĩa trước trong Camera OV7670. Trong cấu hình hoạt động của Camera, để đảm bảo độ phân giải ra đúng yêu cầu, có thể thay đổi các các thanh ghi:

Đặt các bit của thanh ghi COM14[3], và SCALING_PCLK_DELAY[7].

Đặt bit 0 của thanh ghi TSLB[0] (0x3Ah) về 0 và thay đổi giá trị các thanh ghi điều khiển cửa sổ theo Bảng 1.9. [6]

Bảng 1.9: Các thanh ghi thiết lập cửa sổ

Chức năng	Thanh ghi	Địa chỉ thanh ghi
Cột bắt đầu	HSTART[7:0], HREF[2:0]	0x17, 0x32
Cột kết thúc	HSTOP[7:0], HREF[5:3]	0x18, 0x32
Dòng bắt đầu	VSTRT[7:0], VREF[2:0]	0x19. 0x03
Dòng kết thúc	VSTOP[7:0], VREF[5:3]	0x1A, 0x03



Hình 1.22: Ví dụ về cửa sổ 320x240

Hình 1.22 [6] cho thấy một ví dụ về cửa sổ frame. Sau khi cài đặt các thanh ghi trong Bảng 1.9 với giá trị thích hợp, tín hiệu ra có độ phân giải là 320x240.

1.2.4.4 Tín hiệu đồng bộ

Camera OV7670 có hai tín hiệu đồng bộ VSYNC và HREF. Tín hiệu VSYNC là tín hiệu liên tục, tín hiệu HREF chỉ có giá trị khi có dữ liệu ra, nếu không có dữ liệu ra, tín hiệu HREF giữ mức thấp hoặc cao phụ thuộc vào cực tính của tín hiệu.

Cực tính của tín hiệu VSYNC, HREF được cài đặt tương ứng bởi các bit của thanh ghi COM10 (địa chỉ 0x15). Thông thường, ứng dụng sử dụng sườn lên của tín hiệu PCLK để lấy dữ liệu khi HREF ở mức cao. Tín hiệu PCLK chạy tự do theo mặc định, có thể đặt bit 5 của thanh ghi COM10[5] ở mức cao để tín hiệu PCLK chỉ chạy khi có tín hiệu HREF.

Bảng 1.10: Thanh ghi điều khiển tín hiệu đồng bộ của Camera

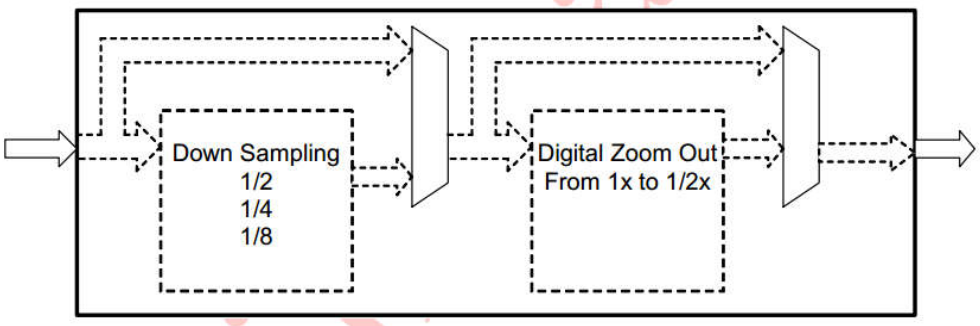
Thanh ghi	Địa chỉ	Giá trị mặc định	Mô tả
COM10	0x15	00	Bit[6]: HREF thay đổi theo tín hiệu HSYNC Bit[5] =0: PCLK luôn dao động Bit[5] =1: PCLK không dao động trong khoảng horizontal bank Bit[4]=0: dữ liệu ra ở sườn xuống PCLK Bit[4]=1: dữ liệu ra ở sườn lên PCLK Bit[3]=0: HREF cực tính dương Bit[3]=0: HREF cực tính âm

1.2.4.5 Tỷ lệ hình ảnh

Camera OV7670 cho phép cài đặt để hình ảnh ra có độ phân giải khác nhau được lấy tỷ lệ từ độ phân giải VGA.

Mạch điều khiển tỷ lệ hình ảnh gồm hai khối, lấy mẫu xuống (Down Sampling) và co dãn ảnh số (Digital Zoom Out) được mô tả trong hình 1.23. [6]

Khối Down Sampling làm giảm kích thước hình ảnh VGA theo tỷ lệ 1/2, 1/4, 1/8. Khối Digital Zoom Out cho phép co giãn ảnh theo hai chiều ngang dọc độc lập.



Hình 1.23: Mạch điều khiển tỷ lệ hình ảnh

Ví dụ: để lấy hình ảnh có độ phân giải 256x128. Đầu tiên, hình ảnh VGA qua khối Down Sampling xuống độ phân giải 320x240 với tỷ lệ 1/2 trên cả hai chiều ngang, dọc. Sau đó khối Digital Zoom Out lấy tỉ lệ từ 320x240 xuống 256x128 bằng cách đặt tỷ lệ 0,8 chiều ngang, 0,53 chiều dọc.

Bảng 1.11: Các thanh ghi cài đặt tỷ lệ hình ảnh [6]

Function	Register	Address	Description
Digital Zoom Enable Bit	COM3[3]	0x0C	0: Bypass 1: Enable
Down Sampling Enable Bit	COM3[2]	0x0C	0: Disable 1: Enable
Down Sampling Related Control Register	SCALING_DCWCTR[7:0]	0x72	See Table 6-2
Pixel Clock Divider	SCALING_PCLK_DIV[3:0]	0x73	DSP Output Clock Divider Bit [3]: 0: Bypass 1: Enable Bit [2:0]: 000: Divider = 1 001: Divider = 2 010: Divider = 4 011: Divider = 8 100: Divider = 16 101: Divider = 32 110: Divider = 64 111: Divider = Not allowed
Horizontal Scaling Ratio	REG74[6:0]	0x74	From 1x (0x20) to 0.5x (0x40) Horizontal Scaling Ratio = 0x20 / (REG74[6:0])
Vertical Scaling Ratio	REG75[6:0]	0x75	From 1x (0x20) to 0.5x (0x40) Vertical Scaling Ratio = 0x20 / (REG75[6:0])
Pixel Clock Delay	SCALING_PCLK_ DELAY[3:0]	0xA2	Original H size / Pixel clock divider - New H size

Bảng 1.12: Thanh ghi điều khiển co giãn ảnh [6]

Function	Register	Address	Description
Option for Vertical Average Calculation	SCALING_DCWCTR[7]	0x72	0: Vertical truncation 1: Vertical rounding
Option for Vertical Down Sampling	SCALING_DCWCTR[6]	0x72	0: Vertical truncation 1: Vertical rounding
Vertical Down Sampling Rate	SCALING_DCWCTR[5:4]	0x72	00: No vertical down sampling 01: Vertical down sample by 2 10: Vertical down sample by 4 11: Vertical down sample by 8
Option for Horizontal Average Calculation	SCALING_DCWCTR[3]	0x72	0: Horizontal truncation 1: Horizontal rounding
Option for Horizontal Down Sampling	SCALING_DCWCTR[2]	0x72	0: Horizontal truncation 1: Horizontal rounding
Horizontal Down Sampling Rate	SCALING_DCWCTR[1:0]	0x72	00: No horizontal down sampling 01: Horizontal down sample by 2 10: Horizontal down sample by 4 11: Horizontal down sample by 8

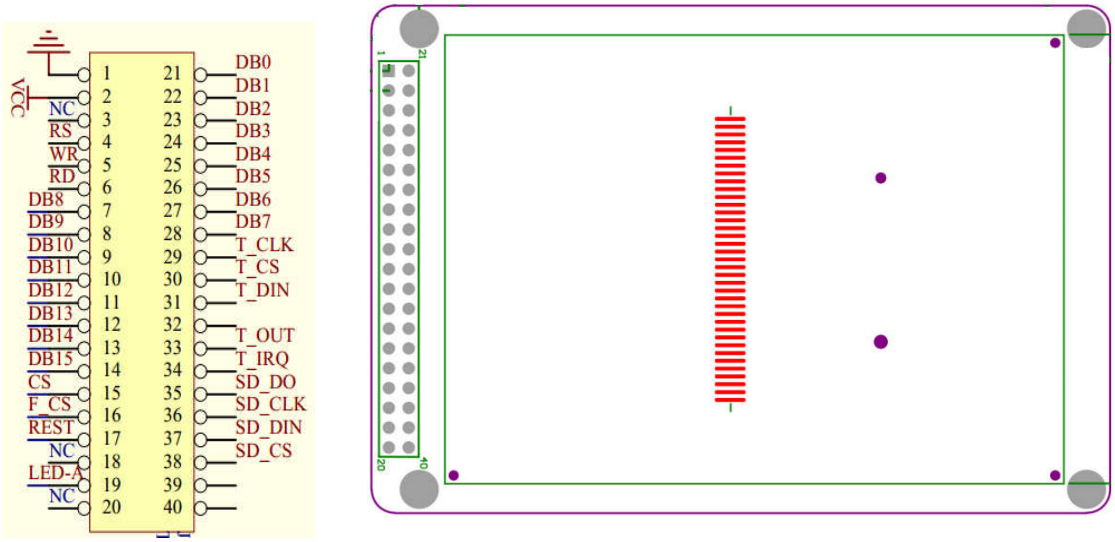
1.2.4.6 Lọc dải

Cường độ ánh sáng trong phòng thường thay đổi theo tần số nguồn AC. Với nguồn AC có tần số 50Hz, có thể thiết lập cấu hình bộ lọc như sau:

```
SCCB_write_reg(0x13, 0xe7); //banding filter enable
SCCB_write_reg(0x9d, 0xa5); //50Hz banding filter
```

```
SCCB_write_reg(0xa5, 0x02); //3 step for 50hz
SCCB_write_reg(0x3b, 0x0a); //Select 50Hz banding filter
```

1.3. Module màn hình LCD 3,2” ILI9341



Hình 1.24: Sơ đồ chân giao tiếp màn hình LCD 3,2” ILI9341

1.3.1 Đặc điểm kỹ thuật

Điện áp hoạt động: 3,3V.

Độ phân giải 240 cột × 320 hàng với 262K màu

Chuẩn giao tiếp 2 chế độ 8 bit và 16 bit.

Hỗ trợ cảm ứng điện trở, khe cắm thẻ nhớ SD.

IC điều khiển là ILI9341.

Chức năng các chân giao tiếp của màn hình LCD 3,2” ILI9341 với vi điều khiển được mô tả trong Bảng 1.12.

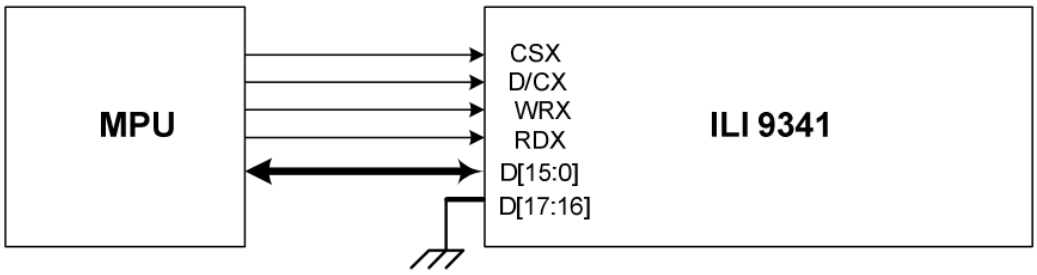
Bảng 1.12: Các chân giao tiếp màn hình LCD 3,2” ILI9341

Số chân	Tên chân	Chức năng
1	GND	Nối đất nguồn
2	VCC	Nối nguồn 3,3V
4	RS	RS=0 ghi lệnh lên LCD RS=1 ghi data lên LCD
5	WR	Ghi dữ liệu, sườn lên của xung trên chân WR sẽ ghi data lên LCD

6	RD	Đọc dữ liệu, sườn lên của xung trên chân RD đọc trạng thái LCD
7-14	DB8-DB15	8 bit cao data
15	CS	Chip select, tích cực mức thấp
17	REST	Reset LCD, tích cực mức thấp
19	LED_A	Chân Anot đèn nền
21-28	DB7-DB0	8 bit thấp data

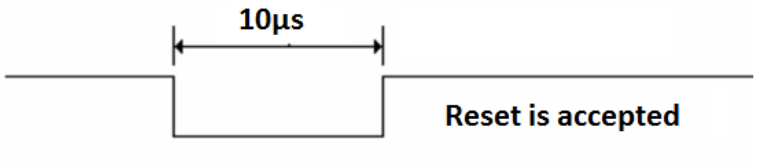
Màn hình có thể giao tiếp với vi điều khiển ở chế độ song song 8 bit hoặc 16 bit dữ liệu. Luận văn sử dụng giao tiếp màn hình ở chế độ song song 16 bit dữ liệu.

Hình 1.25 [9] dưới đây mô tả giao tiếp 16 bit giữa vi điều khiển với ILI9341. Vi điều khiển sử dụng 4 tín hiệu điều khiển CSX, D/CX (chân RS), WRX, RDX và 16 đường dữ liệu để giao tiếp với màn hình LCD.



Hình 1.25: Giao tiếp 16 bit với ILI9341

1.3.2 RESET màn hình



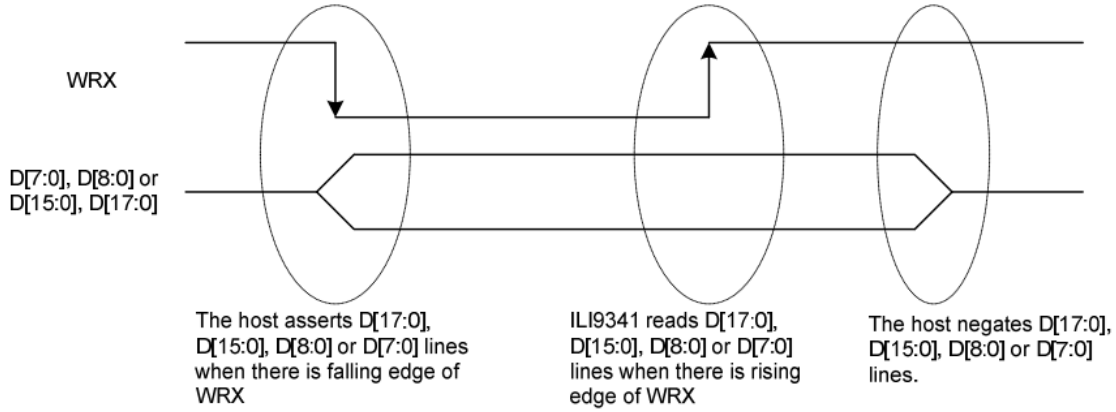
Hình 1.26: Tín hiệu Reset

Tín hiệu RESET chế độ hoạt động của màn hình thực hiện bằng cách kéo chân RESET từ mức cao xuống thấp, giữ tối thiểu 10µs, sau đó kéo chân RESET lên mức cao, giữ khoảng 120ms để đặt lại các giá trị mặc định.

1.3.3 Ghi dữ liệu vào thanh ghi ILI9341

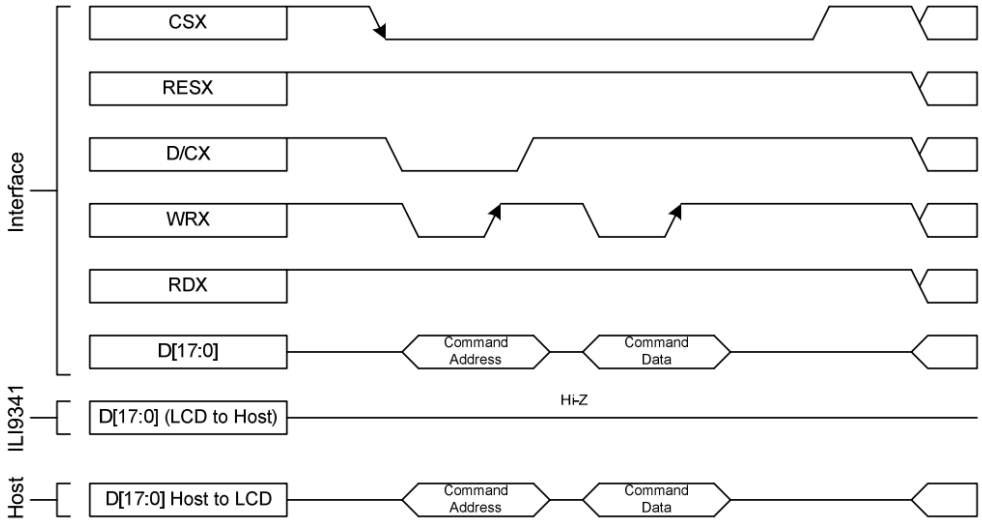
Hình 1.27 mô tả chu kỳ ghi dữ liệu với ILI9341 [9]. Chân WRX điều khiển ghi tín hiệu, ở sườn xuống khi tín hiệu WRX xuống mức thấp, vi xử lý

chốt dữ liệu trên bus data, ở sườn lên khi tín hiệu WRX kéo lên mức cao, ILI9341 đọc dữ liệu trên bus data.



Hình 1.27: Chu kỳ ghi dữ liệu với ILI9341

ILI9341 có hai chế độ ghi dữ liệu: ghi lệnh và ghi dữ liệu được điều khiển bằng chân RS. Nếu RS = 0: ghi lệnh, nếu RS = 1: ghi dữ liệu.



Hình 1.28: Quá trình ghi dữ liệu với ILI9341

Hình 1.28 [9] là giản đồ thời gian quá trình ghi dữ liệu vào thanh ghi của ILI9341 gồm hai bước:

+ Ghi lệnh: gửi địa chỉ thanh ghi

+ Ghi dữ liệu: gửi dữ liệu thanh ghi

Quá trình ghi lệnh

Chân CS từ mức cao xuống mức thấp.

Chân RESX (RESET) giữ mức cao.

Chân D/CX (RS) từ mức cao xuống mức thấp.

Chân WRX (WR) từ mức cao xuống mức thấp.

Chốt địa chỉ lệnh trên bus dữ liệu D[15:0].

Chân WRX (WR) từ mức thấp lên mức cao để ghi lệnh vào LCD.

Chân D/CX (RS) từ mức thấp lên mức cao để kết thúc ghi lệnh.

Quá trình ghi dữ liệu

Chân CS từ mức cao xuống mức thấp

Chân RESX (RESET) giữ mức cao.

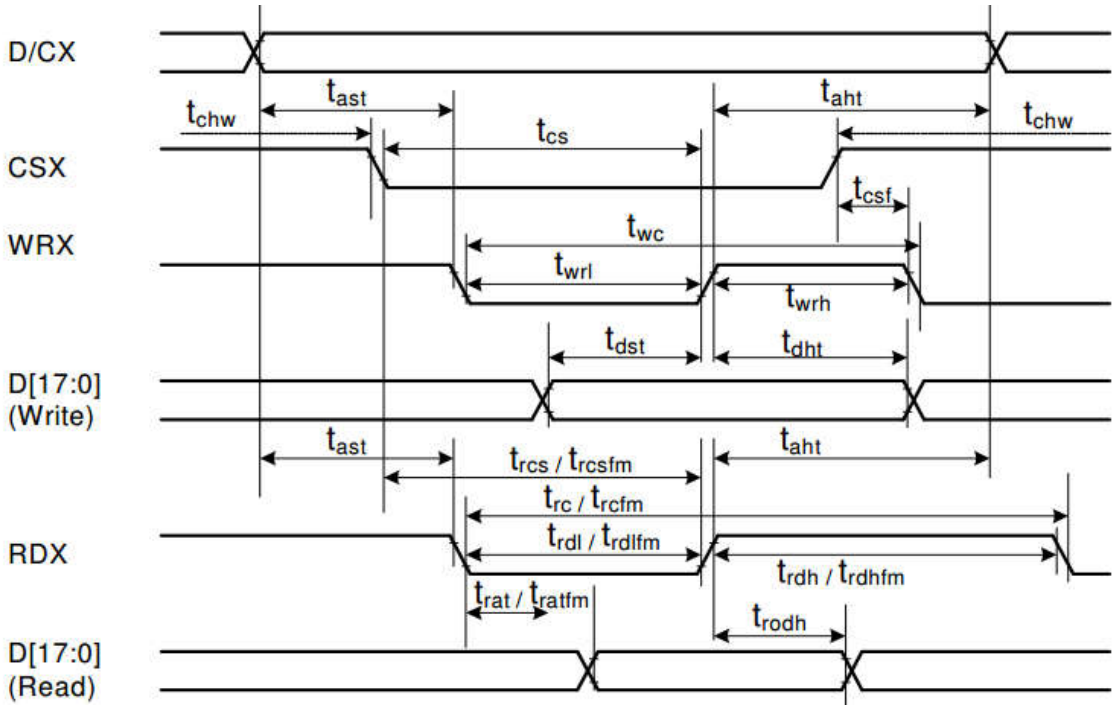
Chân D/CX (RS) giữ mức cao.

Chân WRX (WR) từ mức cao xuống mức thấp

Chốt data trên bus dữ liệu D[15:0].

Chân WRX (WR) từ mức thấp lên mức cao để ghi dữ liệu vào LCD.

Trong quá trình giao tiếp với LCD, cần chú ý đảm bảo theo giản đồ thời gian trong datasheet ILI9341 [9].



Hình 1.29: Giản đồ thời gian tín hiệu của ILI9341

Signal	Symbol	Parameter	min	max	Unit	Description
DCX	tast	Address setup time	0	-	ns	
	taht	Address hold time (Write/Read)	0	-	ns	
CSX	tchw	CSX "H" pulse width	0	-	ns	
	tcs	Chip Select setup time (Write)	15	-	ns	
	trcs	Chip Select setup time (Read ID)	45	-	ns	
	trcsfm	Chip Select setup time (Read FM)	355	-	ns	
	tcsf	Chip Select Wait time (Write/Read)	10	-	ns	
WRX	twc	Write cycle	66	-	ns	
	twrh	Write Control pulse H duration	15	-	ns	
	twrl	Write Control pulse L duration	15	-	ns	
RDX (FM)	trcfm	Read Cycle (FM)	450	-	ns	
	trdhfm	Read Control H duration (FM)	90	-	ns	
	trdlfm	Read Control L duration (FM)	355	-	ns	
RDX (ID)	trc	Read cycle (ID)	160	-	ns	
	trdh	Read Control pulse H duration	90	-	ns	
	trdl	Read Control pulse L duration	45	-	ns	
D[17:0], D[17:10]&D[8:1], D[17:10], D[17:9]	tdst	Write data setup time	10	-	ns	For maximum CL=30pF For minimum CL=8pF
	tdht	Write data hold time	10	-	ns	
	trat	Read access time	-	40	ns	
	tratfm	Read access time	-	340	ns	
	trod	Read output disable time	20	80	ns	

Hình 1.30: Tham số thời gian tín hiệu của ILI9341

Theo tham số thời gian trong hình 1.30 [9]. Thời gian tín hiệu WRX ở mức thấp (twrl) tối thiểu là 15ns, căn cứ vào thời gian này có thể điều chỉnh độ trễ thích hợp đối với hệ thống có tốc độ cao khi giao tiếp với LCD. Đối với STM32F407VG, tần số clock lên tới 168MHz, có chu kỳ khoảng 6ns, cần thực hiện hàm delay một số xung clock nhất định để đảm bảo định thời như trong tài liệu datasheet ILI9341.

1.3.4 Bảng các thanh ghi lệnh của ILI9341

IC ILI9341 có rất nhiều thanh ghi để cài đặt chế độ hoạt động cho màn hình, dưới đây chỉ đề cập tới một số thanh ghi thường dùng.

Bảng 1.13: Một số thanh ghi của ILI9341

Tên lệnh	Địa chỉ lệnh (hex)	Mô tả
Enter Sleep Mode	10h	Đặt màn hình vào chế độ tiêu thụ năng lượng tối thiểu
Sleep Out	11h	Tắt màn hình
Normal Display Mode ON	13h	Màn hình hiển thị ở chế độ bình thường
Display Inversion OFF	20h	Không thay đổi nội dung hình ảnh hiển thị
Display Inversion ON	21h	Đảo màu âm bản nội dung hình ảnh hiển thị

Display OFF	28h	Hình ảnh từ bộ nhớ không được hiển thị trên màn hình
Display ON	29h	Cho phép hình ảnh từ bộ nhớ được hiển thị trên màn hình
Column Address Set	2Ah	Đặt giới hạn tọa độ hiển thị theo chiều ngang từ cột x1 đến x2 (số cột tính bằng pixel)
Page Address Set	2Bh	Đặt giới hạn tọa độ hiển thị theo chiều dọc từ hàng y1 đến y2
Memory Write	2Ch	Lệnh truyền dữ liệu ra màn hình
Memory Access Control	36h	Điều khiển chiều quét hình ảnh từ trái sang phải, từ trên xuống dưới và ngược lại Có thể lật gương, xoay ngang/dọc màn hình từ thiết lập trên thanh ghi này
COLMOD: Pixel Format Set	3Ah	Thiết lập định dạng pixel 16 bit hoặc 18 bit
Write Display Brightness	51h	Thiết lập độ sáng màn hình
Write Content Adaptive Brightness Control	55h	Thiết lập độ tương phản hình ảnh

1.3.5 Cấu hình đèn nền LED_A

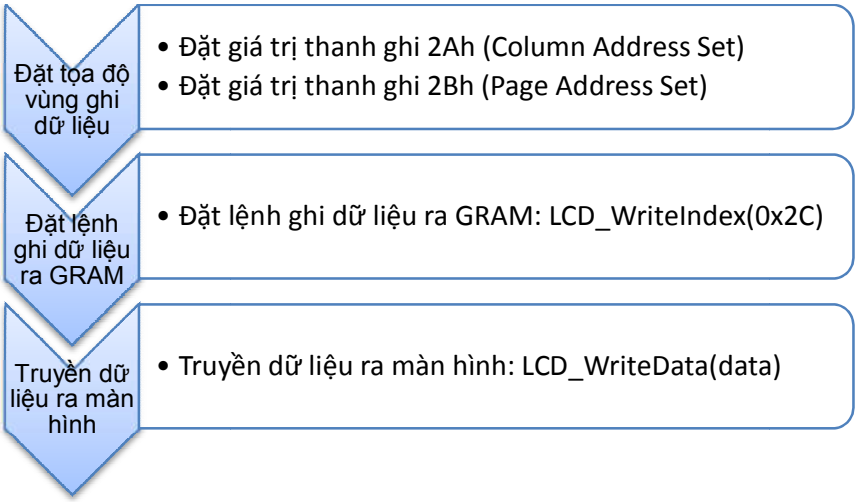
Module màn hình sử dụng đèn nền LED_A để điều khiển cường độ sáng màn hình. Đèn nền LED_A có dòng khoảng 15 mA. Trong luận văn lập trình với kit STM32F407-Discovery, sử dụng tín hiệu điều biến độ rộng xung PWM để điều chỉnh dòng làm việc của đèn nền LED_A trong khoảng 0÷15 mA qua đó điều chỉnh độ sáng nền màn hình.

Tín hiệu mức cao làm tăng dòng LED, tín hiệu mức thấp làm giảm dòng LED. Chu kỳ xung tín hiệu điều biến độ rộng xung PWM ở mức cao tương đương dòng LED ở mức cao. Tần số điều chế tín hiệu PWM lý tưởng nên đặt trong khoảng 5÷10 kHz.

1.3.6 Hiển thị dữ liệu ra LCD

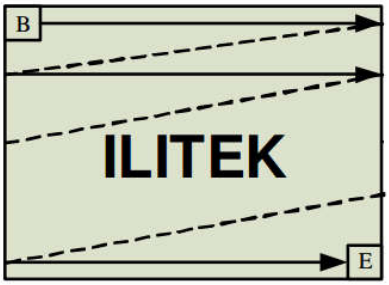
Để hiển thị dữ liệu ra LCD, cần ghi dữ liệu vào vùng nhớ GRAM của LCD. Vùng nhớ GRAM có kích thước 172.800 bytes với hình ảnh độ phân giải cao nhất 240x320, 18 bit/pixel. Ở chế độ 16 bit/pixel, dữ liệu hiển thị trên LCD theo định dạng RGB565.

Quá trình hiển thị hình ảnh ra LCD được thể hiện trên hình 1.31:



Hình 1.31: Quá trình hiển thị hình ảnh ra LCD

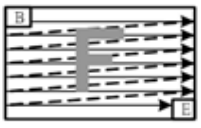
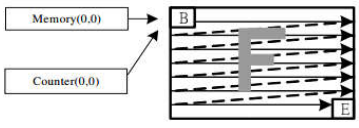
Ở chế độ bình thường, một frame ảnh trong bộ nhớ được truyền ra màn hình theo hướng từ trái qua phải, từ trên xuống dưới như hình 1.32 [9]

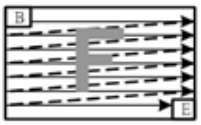
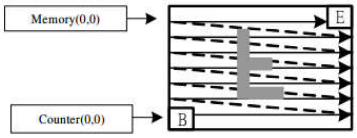
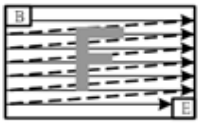
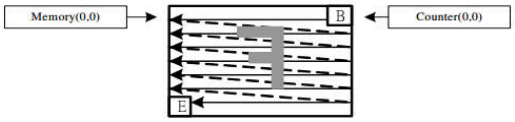
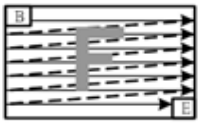
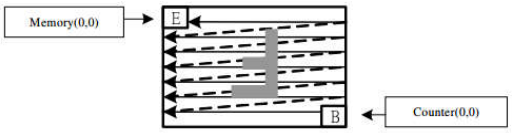
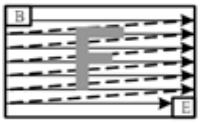
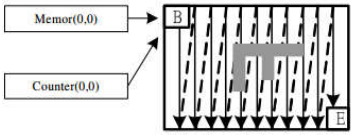
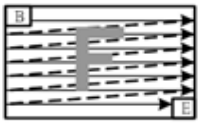
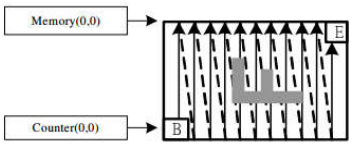
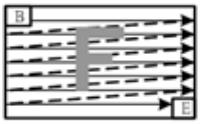
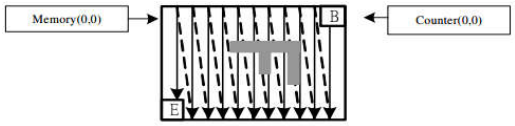
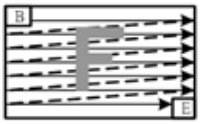
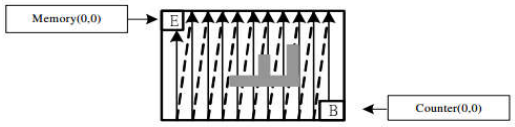


Hình 1.32: Hướng quét dữ liệu ra màn hình

ILI9341 có một thanh ghi điều khiển "Memory Data Access Control" (địa chỉ 0x36) cho phép xoay ngang/dọc hình ảnh, lật gương hình ảnh theo hai chiều ngang/dọc. Để làm điều này, thực hiện cài đặt 3 bit 5 (MV), bit 6 (MX), bit 7 (MY) của thanh ghi. Bảng 1.14 mô tả chi tiết cài đặt chế độ hiển thị hình ảnh.

Bảng 1.14: Cấu hình hiển thị hình ảnh từ bộ nhớ ra màn hình

Hiển thị hình ảnh	**Tham số thanh ghi 36h**			**Hình ảnh trong bộ nhớ**	**Hình ảnh trên màn hình**
	MV	**MX**	**MY**		
Bình thường	0	0	0		

Y-Mirror	0	0	1		
X-Mirror	0	1	0		
X-Mirror Y-Mirror	0	1	1		
X-Y Exchange	1	0	0		
X-Y Exchange Y-Mirror	1	0	1		
X-Y Exchange X-Mirror	1	1	0		
X-Y Exchange X-Y Mirror	1	1	1		

Ví dụ thiết lập giá trị thanh ghi 36h để định hướng màn hình hiển thị theo chiều dọc (240 cột x 320 hàng) hoặc chiều ngang (320 cột x 240 hàng):

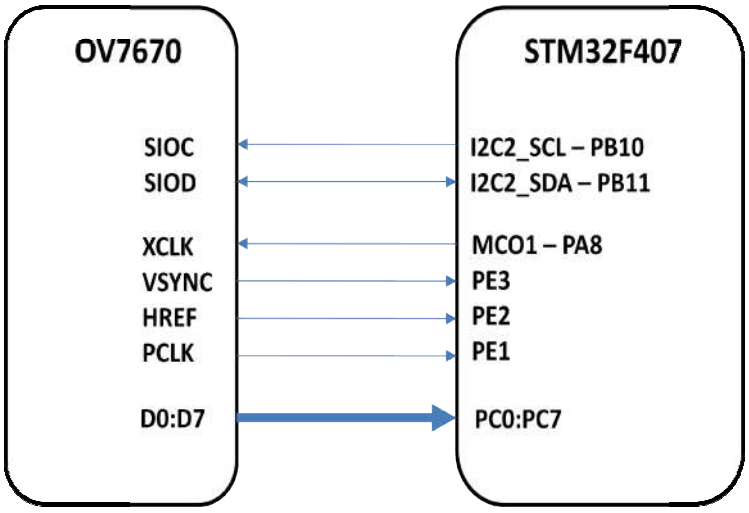
```
LCD_WriteData(0x88); // man hinh doc 240x320
LCD_WriteData(0x28); // man hinh ngang 320x240
```

CHƯƠNG II: THỰC NGHIỆM GHÉP NỐI HỆ THỐNG

2.1. Ghép nối STM32F4 – OV7670

2.1.1 Sơ đồ ghép nối

Sơ đồ ghép nối OV7670 với STM32F407VG như hình 2.1



Hình 2.1: Sơ đồ ghép nối chân tín hiệu OV7670 với STM32F407VG

Để thực hiện giao tiếp với Camera OV7670, luận văn xây dựng một số hàm sau:

Hàm **MCO1_init**, tạo xung clock 16 MHz ở chân PA8 đưa đến chân XCLK của OV7670.

Hàm **SCCB_init**, khai báo chân nối SCCB (chân PB10, PB11), khởi tạo chế độ hoạt động của khối I2C2.

Hàm **SCCB_write_reg**, **SCCB_read_reg** để ghi/đọc dữ liệu thanh ghi Camera OV7670.

Hàm **OV7670_pin_config**, cấu hình chế độ hoạt động các chân thu tín hiệu từ Camera OV7670 ở chế độ input: bus dữ liệu 8 bit PC0÷PC7, tín hiệu đồng bộ PE1 (PCLK), PE2 (HREF), PE3 (VSYNC).

Hàm **CaptureImage**, chụp một khung hình từ Camera.

2.1.2 Tạo xung clock đến chân XCLK của OV7670

Để Camera OV7670 hoạt động, cần cấp một xung clock đến chân XCLK của OV7670, tần số xung clock từ 10÷48 MHz theo datasheet.

Cấu hình chân PA8 (MCO1) để tạo dao động 16MHz với hàm **MCO1_init** như sau:

```
void MCO1_init(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   RCC_ClockSecuritySystemCmd(ENABLE);
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
   // GPIO config
   GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_8;   //PA8 -> XCLK
   GPIO_InitStructure.GPIO_Speed= GPIO_Speed_100MHz;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
   GPIO_InitStructure.GPIO_Otype= GPIO_OType_PP;
   GPIO_InitStructure.GPIO_PuPd= GPIO_PuPd_UP;
   GPIO_Init(GPIOA,&GPIO_InitStructure);
   // GPIO AF config, noi PA8 voi khoi MCO
   GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_MCO);
   // Nguon clock cho MCO1, HSI clock = 16MHz
   RCC_MCO1Config(RCC_MCO1Source_HSI, RCC_MCO1Div_1); //16 MHz
}
```

2.1.2 Lập trình SCCB qua I2C

Chế độ hoạt động cho Camera OV7670 được cấu hình bằng cách thiết lập giá trị thanh ghi với giá trị thích hợp qua giao thức SCCB, giao thức này có sự tương thích với giao thức I2C. Trong luận văn sử dụng khối I2C2 giao tiếp với Camera OV7670.

Đầu tiên, phải khai báo chân nối, cấu hình hoạt động của khối I2C2 cho phù hợp với hàm **SCCB_init.**

```
void SCCB_init(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   I2C_InitTypeDef I2C_InitStructure;
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C2, ENABLE);
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
   // GPIO config.
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10|GPIO_Pin_11;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
   GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
   GPIO_Init(GPIOB,&GPIO_InitStructure);
   // GPIO AF config, Khai bao noi chan I2C2: PB10, PB11
```

```
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_I2C2);
    GPIO_PinAFConfig(GPIOB, GPIO_PinSource11, GPIO_AF_I2C2);
    // Cau hinh che do hoat dong cua I2C2
    I2C_DeInit(I2C2);
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
    //50% duty cycle
    I2C_InitStructure.I2C_OwnAddress1 = 0x00;
    // dia chi I2C master tuy y
    I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
    I2C_InitStructure.I2C_AcknowledgedAddress =
    I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_ClockSpeed = 100000;  // tan so 100Khz
    I2C_ITConfig(I2C2, I2C_IT_ERR, ENABLE);
    I2C_Init(I2C2,&I2C_InitStructure);
    I2C_Cmd(I2C2, ENABLE);
}
```

1.2.1 Lập trình ghi dữ liệu vào thanh ghi OV7670

Để ghi dữ liệu vào thanh ghi của Camera OV7670 theo giao thức SCCB cần sử dụng chu kỳ truyền dữ liệu 3 pha. Lưu đồ thuật toán như hình 2.2

Hình 2.2: Lưu đồ thuật toán ghi dữ liệu thanh ghi camera OV7670

Chương trình ghi dữ liệu vào thanh ghi Camera OV7670 [4]

```
void SCCB_write_reg(uint8_t reg_addr, uint8_t data)
{
   // Kiem tra trang thai duong day.
   while(I2C_GetFlagStatus(I2C2, I2C_FLAG_BUSY)){};
   // I2C2 START
   I2C_GenerateSTART(I2C2, ENABLE);
   // Kiem tra STM32F4 o che do master
   while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_MODE_SELECT)){};
   // Pha 1: Gui dia chi cua Camera.
   I2C_Send7bitAddress(I2C2, OV7670_WRITE_ADDR,
                       I2C_Direction_Transmitter);
   // Doi den khi Camera xac nhan bang Don't Care bit
   while(!I2C_CheckEvent(I2C2,
                     I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED)){};
   // Pha 2: Gui dia chi thanh ghi cua Camera
   I2C_SendData(I2C2, reg_addr); //
   // Kiem tra truyen dia chi thanh ghi thanh cong
   while(!I2C_CheckEvent(I2C2,
                     I2C_EVENT_MASTER_BYTE_TRANSMITTED)){};
   // Pha 3: Truyen gia tri moi cua thanh ghi
   I2C_SendData(I2C2, data);
   // Kiem tra truyen gia tri thanh ghi thanh cong
   while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
   // Ket thuc truyen
   I2C_GenerateSTOP(I2C2, ENABLE);
}
```

1.2.1 Lập trình đọc dữ liệu thanh ghi OV7670

Để đọc dữ liệu thanh ghi của Camera OV7670 theo giao thức SCCB cần sử dụng hai chu kỳ truyền dữ liệu:

- Chu kỳ ghi dữ liệu 2 pha.

- Chu kỳ đọc dữ liệu 2 pha.

Lưu đồ thuật toán như hình 2.3

Hình 2.3: Lưu đồ thuật toán đọc dữ liệu thanh ghi camera OV7670

Chương trình đọc dữ liệu thanh ghi OV7670 xem trong phụ lục 1.

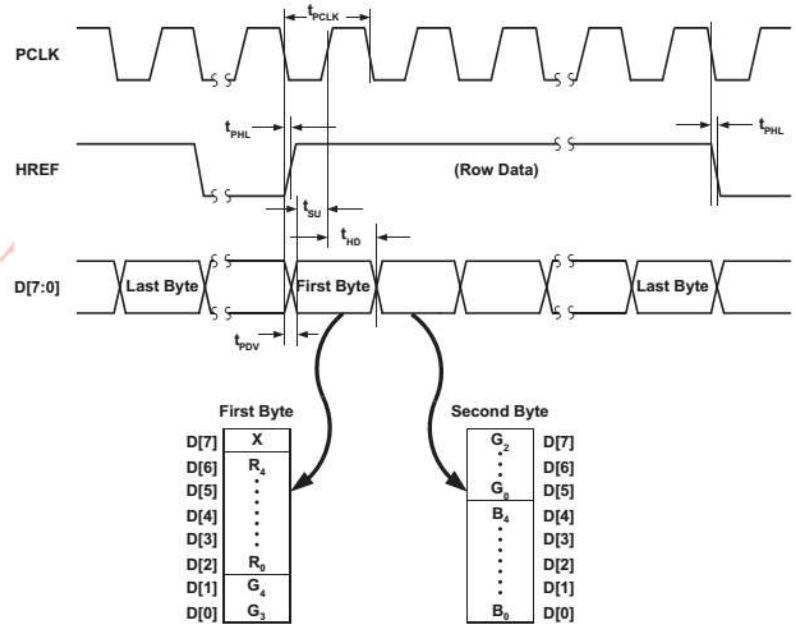
2.1.3 Bắt ảnh từ Camera

Để thu nhận ảnh từ Camera, phải cấu hình chân thu tín hiệu đồng bộ VSYNC, HREF, PCLK, bus dữ liệu D0÷D7 từ Camera ở chế độ input.

```
void OV7670_pin_config(void)
{
   GPIO_InitTypeDef GPIO_InitStructure;
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
   //PC0-PC7: D0:D7, input floating
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 |GPIO_Pin_1 |GPIO_Pin_2
   |GPIO_Pin_3 |GPIO_Pin_4 |GPIO_Pin_5 |GPIO_Pin_6 |GPIO_Pin_7;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
   GPIO_Init(GPIOC, &GPIO_InitStructure);
   // PE3 -> VSYNC, PE2 -> HREF, PE1 -> PCLK
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
}
```

2.1.3.1 Bắt ảnh màu RGB565 lưu vào bộ nhớ đệm



Hình 2.4: Giản đồ thời gian tín hiệu RGB565

Ảnh RGB565 gồm 16 bit/pixel, theo giản đồ thời gian hình 2.4 [5]. Vi xử lý cần lấy mẫu dữ liệu trong 2 xung clock PCLK để thu được một pixel.

Chương trình bắt ảnh được viết như sau [12]:

```
#define VSYNC ((GPIOE->IDR & 0x8) >> 3) //doc trang thai chan PE3
#define HREF  ((GPIOE->IDR & 0x4) >> 2) //doc trang thai chan PE2
#define PCLK  ((GPIOE->IDR & 0x2) >> 1) //doc trang thai chan PE1
void captureImage(uint16_t cot, uint16_t hang)
{
    i = 0;
    uint16_t x,y,z;
    while(!VSYNC); //-> tin hieu VSYNC cao, ket thuc mot frame
    while(VSYNC);  //-> tin hieu VSYNC xuong thap, bat dau frame
    y= hang; // so line
    while(y--)
```

```
    {
    x=cot; // so pixel tren mot line
    while(!HREF); // tin hieu HREF len muc cao, bat dau mot line
    while(x--)
      {
      while(PCLK); // doi tin hieu PCLK xuong thap
      z =(GPIOC->IDR& 0x00FF)<< 8; // byte1
      while(!PCLK); // doi tin hieu PCLK len cao
      while(PCLK); // doi tin hieu PCLK xuong thap
      // luu pixel vao bo nho demframe_buffer
      frame_buffer[i]=(GPIOC->IDR & 0x00FF)|z; // byte 2
      while(!PCLK);//doi tin hieu PCLK len cao
      i++;
      }
    }
}
```

Chương trình sẽ thu từng điểm ảnh và lưu vào **frame_buffer** dạng ma trận, mỗi phần tử của **frame_buffer** là một biến 16 bit tương đương một điểm ảnh.

Ảnh được lưu trong bộ nhớ đệm có định dạng RGB565, có thể được hiển thị trực tiếp ra màn hình LCD do màn hình hỗ trợ hiển thị RGB565.

2.1.3.2 Bắt ảnh đa mức xám lưu vào bộ nhớ đệm

Mặc định tín hiệu YUV422 do Camera OV7670 truyền đến vi điều khiển theo thứ tự YUYV, lấy trung bình là 2 byte/pixel. Màn hình LCD không hiển thị trực tiếp định dạng YUV do không hỗ trợ định dạng này, để hiển thị hình ảnh YUV thu được cần chuyển đổi định dạng YUV sang RGB565.

Do đặc điểm tín hiệu chói Y trong định dạng YUV chính là ảnh đa mức xám, khi Camera truyền dữ liệu định dạng YUV, chỉ cần lấy tín hiệu Y là đã có được hình ảnh đa mức xám.

Chương trình bắt ảnh đa mức xám tương tự như chương trình bắt ảnh màu RGB565, chỉ khác ở chỗ trong quá trình lấy mẫu dữ liệu trong 2 xung clock PCLK, vi xử lý chỉ lấy một byte ở xung clock đầu tiên (thu tín hiệu Y). [12]

```
void captureVGA(uint16_t cot, uint16_t hang) //
{
   i = 0;
   uint16_t x,y;
```

```
    while(!VSYNC); //-> tin hieu VSYNC cao
    while(VSYNC);  //-> tin hieu VSYNC xuong thap, bat dau frame
    y= hang; // so line
    while(y--)
    {
    x=cot; // so pixel tren mot line
    while(!HREF); // tin hieu HREF len muc cao, bat dau mot line
    while(x--)
      {
      while(PCLK); // doi tin hieu PCLK xuong thap
      frame_buffer[i]= GPIOC->IDR &0x00FF;//byte 1, tin hieu Y
      while(!PCLK); // doi tin hieu PCLK len cao
      while(PCLK); // doi tin hieu PCLK xuong thap
      while(!PCLK);//doi tin hieu PCLK len cao
      i++;
      }
    }
}
```

Ảnh đa mức xám được lưu vào frame_buffer dạng ma trận, mỗi phần tử của frame_buffer là một biến 8 bit tương đương một điểm ảnh.

Ảnh đa mức xám (8 bit/pixel) làm giảm kích thước bộ nhớ một nửa so với ảnh màu RGB565 (16 bit/pixel) trong cùng độ phân giải, phù hợp với vi xử lý có bộ nhớ đệm hạn chế.

2.1.4 Cấu hình hoạt động Camera OV7670

Hình 2.5: Lưu đồ cấu hình chế độ hoạt động của Camera OV7670

Hình 2.5 mô tả các bước cơ bản khi cấu hình chế độ hoạt động của Camera OV7670, trong quá trình thực nghiệm có thể thay đổi giá trị thanh ghi CLKRC (địa chỉ 0x11) để điều chỉnh tần số PCLK cho đến khi hệ thống bắt ảnh thành công.

Trong luận văn, thực nghiệm cấu hình Camera OV7670 hoạt động với ba chế độ:

1- Chế độ QQVGA, RGB565 [7, 12, 16, 17] (xem phụ lục 2)

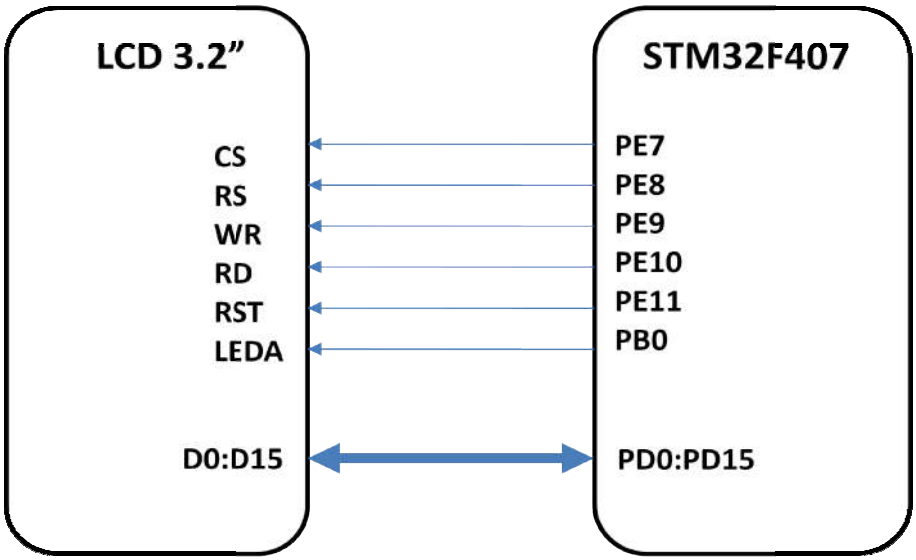
2- Chế độ QVGA, RGB565 [7, 17, 20] (xem phụ lục 3)

3- Chế độ QVGA, YUV [7, 12, 13, 17] (xem phụ lục 4)

2.2. Ghép nối STM32F4 – LCD 3,2” ILI9341

2.2.1 Sơ đồ ghép nối

Sơ đồ ghép nối STM32F407VG với màn hình LCD 3,2” ILI9341 như hình 2.6



Hình 2.6: Sơ đồ ghép nối STM32F4 – LCD 3,2” ILI9341

Theo hình 2.6, cài đặt tại vi điều khiển như sau:

Chân PB0 tạo xung PWM đến LED_A trên LCD để điều khiển độ sáng nền màn hình LCD.

Các chân PE7, PE8, PE9, PE10, PE11 điều khiển đọc/ghi với màn hình.

Cổng D sử dụng làm bus dữ liệu 16 bit giao tiếp với màn hình.

2.2.2 Lập trình RESET màn hình

RESET hoạt động của màn hình được thực hiện bằng cách kéo chân RESET từ mức cao xuống thấp, giữ tối thiểu 10µs, sau đó kéo chân RESET lên mức cao, giữ khoảng 120 ms để đặt lại các giá trị mặc định.

```
#define RST GPIO_SetBits(GPIOE,GPIO_Pin_11);   // PE11 = 1
#define RSTN GPIO_ResetBits(GPIOE,GPIO_Pin_11); // PE11 = 0
void LCD_Reset(void)
{
   RSTN // PE11 = 1
   delay_us(100);
   RST // PE11 = 0
   delay_ms(120);
}
```

2.2.3 Điều khiển độ sáng màn hình bằng PWM

Luận văn thực hiện lập trình khối TIMER 3, chế độ PWM, xuất tín hiệu ra chân PB0 đưa đến chân LED_A của màn hình LCD để điều khiển độ sáng nền màn hình. Chương trình xem trong phụ lục 5.

2.2.4 Lập trình ghi dữ liệu với LCD 3.2" ILI9341

Đầu tiên, định nghĩa hoạt động của các chân điều khiển LCD

```
#define CS GPIO_SetBits(GPIOE,GPIO_Pin_7);        // PE7 = 1
#define CSN  GPIO_ResetBits(GPIOE,GPIO_Pin_7);    // PE7 = 0
#define RS  GPIO_SetBits(GPIOE,GPIO_Pin_8);       // PE8 = 1
#define RSN GPIO_ResetBits(GPIOE,GPIO_Pin_8);     // PE8 = 0
#define WR GPIO_SetBits(GPIOE,GPIO_Pin_9);        // PE9 = 1
#define WRN GPIO_ResetBits(GPIOE,GPIO_Pin_9);     // PE9 = 0
#define RD GPIO_SetBits(GPIOE,GPIO_Pin_10);       // PE10 = 1
#define RDN GPIO_ResetBits(GPIOE,GPIO_Pin_10);    // PE10 = 0
```

Dựa theo phân tích quá trình ghi lệnh, ghi dữ liệu tại Chương 1, có thể viết hai hàm ghi lệnh và ghi dữ liệu như sau:

Hàm ghi lệnh LCD_WriteIndex

```
/*************************************************************
Function Name : LCD_WriteReg
Noi Dung      :   Ghi lenh vao TFT 3.2 Inch.
Tham Bien     :   index : Ma lenh can ghi.
Tra Ve        :   Khong.
*************************************************************/
```

```
static __inline void LCD_WriteIndex(uint16_t index)
{
   RSN                      // RS=0, ghi lenh
   CSN
   RD                       // RD=1, WR=0
   WRN                      //
   GPIO_Write(GPIOD,index);// Chot du lieu
   delay(1);
   WR                       // Ghi du lieu len LCD
   delay(1);
   CS                       //
}
```

Hàm ghi dữ liệu LCD_WriteData

```
/*****************************************************
Function Name : LCD_WriteData
Noi Dung      :   Ghi du lieu vao TFT 3.2 Inch.
Tham Bien     :   Data: Du lieu can ghi.
Tra Ve        :   Khong.
*****************************************************/
static __inline void LCD_WriteData(uint16_t data)
{
   RS                       // RS=0, ghi lenh
   CSN                      //
   RD                       // RD=1, WR=0
   WRN                      //
   GPIO_Write(GPIOD,data); // Chot du lieu
   delay(1);
   WR                       // Ghi du lieu len LCD
   delay(1);
   CS                       //
}
```

Sử dụng hai hàm cơ bản này có thể cấu hình toàn bộ hoạt động của LCD theo các yêu cầu cụ thể.

2.2.5 Xuất hình ảnh ra LCD

2.5.1 Khởi động LCD

Để hiển thị hình ảnh trên LCD, đầu tiên phải khởi tạo chế độ hoạt động của LCD. Chương trình khởi động LCD xem trong phụ lục 6.

2.5.2 Một số hàm đồ họa cơ bản

**** Xác định vùng hiển thị ảnh trên màn hình***

Để hiển thị một ảnh, đầu tiên cần đặt vùng ghi dữ liệu. Hàm ***LCD_SetCursorPosition*** xác định khu vực hình ảnh hiển thị trên màn hình, tọa độ được cung cấp bởi giá trị hai thanh ghi 0x2A, 0x2B. [15]

```
/************************************************************
Function Name : LCD_SetCursorPosition
Noi Dung    : Cau hinh dia chi cho man hinh.
Xac dinh vung ghi pixel gioi han boi hinh chu nhat
Tham Bien   :  Toa do goc tren ben trai: (x1, y1)
               Toa do goc duoi ben phai: (x2, y2)
Neu x1=x2, y1=y2 => toa do diem anh (x,y)
************************************************************/
void LCD_SetCursorPosition(uint16_t x1, uint16_t y1, uint16_t x2,
                           uint16_t y2)
{
   // xac dinh vi tri cot tu x1 - x2
   LCD_WriteIndex(ILI9341_COLUMN_ADDR);
   LCD_WriteData((x1>>8)&0xFF);
   LCD_WriteData((x1 & 0xFF));
   LCD_WriteData(x2>>8);
   LCD_WriteData(x2 & 0xFF);
   // xac dinh vi tri hang tu y1 - y2
   LCD_WriteIndex(ILI9341_PAGE_ADDR);
   LCD_WriteData((y1>>8)&0xFF);
   LCD_WriteData((y1 & 0xFF));
   LCD_WriteData((y2>>8)&0xFF);
   LCD_WriteData(y2 & 0xFF);
}
```

**** Vẽ một hình chữ nhật***

Hàm ***LCD_REC_Fill*** vẽ một hình chữ nhật theo 3 bước sau:

- Xác định tọa độ pixel bằng hàm ***LCD_SetCursorPosition***
- Đặt lệnh ghi dữ liệu ra GRAM: ***LCD_WriteIndex(ILI9341_GRAM)***

- Ghi dữ liệu màu cho pixel trong hình chữ nhật: ***LCD_WriteData(color)***

Chương trình cụ thể như sau [15]:

```
/************************************************************
Function Name  : LCD_REC_Fill
* To mau hinh chu nhat gioi han boi hai toa do
* Toa do goc tren ben trai: (x0, y0)
* Toa do goc duoi ben phai: (x1, y1)
************************************************************/
void LCD_REC_Fill(uint16_t x0, uint16_t y0, uint16_t x1,
                  uint16_t y1, uint16_t color)
{
   uint32_t pixels_count;
   uint32_t count =0;
   /* Set cursor position */
   LCD_SetCursorPosition(x0, y0, x1, y1); // toa do hinh chu nhat
   /* Set command for GRAM data */
   LCD_WriteIndex(ILI9341_GRAM);
   /* Tinh so pixel trong hinh chu nhat */
   pixels_count = (x1 - x0 + 1) * (y1 - y0 + 1);
   for (count=0; count < pixels_count; count++)
     {
          LCD_WriteData(color); // dat mau cho pixel
     }
}
```

2.5.3 Hiển thị ảnh RGB565 ra LCD

**** Hiển thị ảnh từ bộ nhớ đệm ra màn hình***

Sau khi một khung hình được lưu vào bộ nhớ đệm thành một ma trận với các phần tử 16 bit (tương ứng một pixel). Giả sử ảnh thu được với độ phân giải QQVGA (160x120), định dạng RGB565, hàm hiển thị hình ảnh lên màn hình LCD như sau:

```
void LCD_DisplayImage(uint16_t img[IMG_PIXEL])
{
   uint32_t n;
   LCD_SetCursorPosition(0, 0, 159, 119);
   LCD_WriteIndex(ILI9341_GRAM);
   for(n = 0; n < IMG_PIXEL; n++)
```

```
        {
            LCD_WriteData(img[n]);
        }
}
```

**** Hiển thị ảnh trực tiếp ra màn hình, không lưu vào bộ nhớ đệm***

Màn hình màu sử dụng trong luận văn có độ phân giải 320x240, định dạng RGB565 tương ứng 16 bit/pixel. Kích thước hình ảnh toàn màn hình là:

$320 \times 240 \times 16$ bit = 1.228.800 bit

Kích thước này lớn hơn bộ nhớ của STM32F407VG (1Mb bộ nhớ flash), do đó không thể lưu hình ảnh màu RGB565 với độ phân giải 320x240 vào bộ nhớ đệm rồi hiển thị ra màn hình. Để hiển thị ảnh màu toàn màn hình LCD, sử dụng kỹ thuật đường ống (pipe line), mỗi khi thu được một điểm ảnh thì hiển thị ngay trên màn hình.

Chương trình bắt ảnh và hiển thị trên màn hình tương tự như chương trình bắt ảnh màu RGB565 lưu vào bộ nhớ đệm, chỉ khác ở chỗ ngay sau khi thu được một điểm ảnh, thay vì lưu vào bộ nhớ đệm, điểm ảnh được ghi ra màn hình:

```
void QVGA_RGB565_display(void) // QVGA(320,240)
{
   LCD_SetCursorPosition(0,0,319,239);
   LCD_gram(); // LCD_WriteIndex(ILI9341_GRAM);
   i = 0;
   uint16_t x,y,temp,pixel;
   while(!VSYNC); //doi tin hieu VSYNC cao, ket thuc mot frame
   while(VSYNC); //doi tin hieu VSYNC xuong thap, bat dau mot frame
   y= 240; // so line
   while(y--)
    {
      x=320; // so pixel tren mot line
      while(!HREF); //doi tin hieu HREF len muc cao, bat dau mot line
      while(x--)
        {
            while(PCLK); // doi tin hieu PCLK xuong thap
            temp= (GPIOC->IDR & 0x00FF) << 8;
            //(GPIO_ReadInputData(GPIOC) & 0x00FF) << 8
            while(!PCLK); // doi tin hieu PCLK len cao
            while(PCLK); // doi tin hieu PCLK xuong thap
```

```
            pixel = (GPIOC->IDR & 0x00FF)|temp;
            //GPIO_ReadInputData(GPIOC)) & 0x00FF
            while(!PCLK);//doi tin hieu PCLK len cao
            // ghi pixel len man hinh
           LCD_data(pixel);//LCD_WriteData(pixel);
            i++;
        }
    }
}
```

2.5.4 Hiển thị hình ảnh đa mức xám ra LCD

Ảnh đa mức xám được lưu vào bộ nhớ đệm dạng ma trận, mỗi phần tử là một biến 8 bit tương đương một điểm ảnh. Để hiển thị đúng ảnh đa mức xám, cần chuyển đổi dữ liệu ảnh 8 bit (giá trị Y) sang dạng 16 bit (giá trị RGB) tương thích với màn hình LCD.

Do mỗi điểm ảnh đa mức xám tương đương với một điểm ảnh màu RGB trong đó các giá trị R, G, B bằng nhau. Với định dạng RGB565, thực hiện chuyển đổi như sau:

Giá trị R bằng 5 bit cao của giá trị Y.

Giá trị G bằng 6 bit cao của giá trị Y.

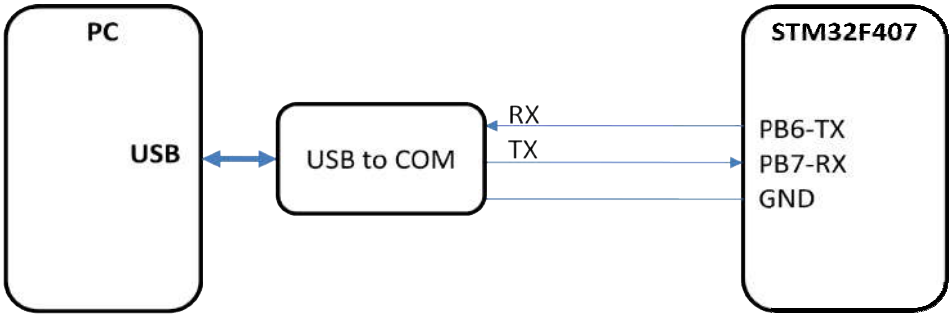
Giá trị B bằng 5 bit cao của giá trị Y.

Thực hiện thuật toán dịch bit và OR các giá trị RGB sẽ thu được ảnh RGB565 hiển thị đúng màu sắc của ảnh đa mức xám. Chương trình như sau:

```
void DisplayQVGAgray(uint8_t img[76800])
{
   uint32_t n;
   uint16_t graypixel=0;
   LCD_SetCursorPosition(0, 0, 319, 239);
   LCD_WriteIndex(ILI9341_GRAM);
   for (n = 0; n < 76800; n++)
     {
       // R = (img[n]&248)<<8; // 5 bit cao cua Y
       // G = (img[n]&252)<<3; // 6 bit cao cua Y
       // B = (img[n]&248)>>3; // 5 bit cao cua Y
    graypixel=((img[n]&248)<<8)|((img[n]&252)<<3)|((img[n]&248)>>3);
       LCD_WriteData(graypixel);
     }
}
```

2.3. Ghép nối STM32F4 – Máy tính

Sơ đồ ghép nối STM32F407VG với máy tính như hình 2.7



Hình 2.7: Sơ đồ ghép nối PC - STM32F4

Sau khi STM32F4 bắt ảnh từ Camera OV7670, ảnh được lưu trong bộ nhớ đệm có thể truyền về máy tính để xử lý. Trong luận văn sử dụng khối USART (Universal Synchronous Asynchronous Receiver Transmitter - truyền nhận nối tiếp đồng bộ và không đồng bộ) để truyền dữ liệu ảnh về máy tính.

2.3.1 Cấu hình hoạt động khối USART

Thực hiện cấu hình khối USART1 dùng hai chân PB6 (USART1-TX) và PB7 (USART1-RX) với hàm USART_Configuration (xem phụ lục 7).

2.3.2 Truyền dữ liệu về máy tính qua USART

Dữ liệu ảnh truyền về máy tính ở chế độ nối tiếp, lần lượt từng byte tương ứng với một điểm ảnh, hàm truyền một byte qua USART như sau [20]:

```
void USART1_send_byte(uint8_t *val) // ham gui mot byte
{
   while (USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
   // Wait for Empty
   USART_SendData(USART1, *val);
}
```

Truyền một frame ảnh về máy tính

Trong trường hợp hệ thống bắt ảnh đa mức xám độ phân giải QVGA (240x320) lưu trong bộ nhớ đệm **`qvga_frame`**. Chương trình gửi dữ liệu ảnh về máy tính sử dụng ngắt của USART1 hoạt động như sau: mỗi khi nhận ký tự B từ máy tính thì STM32F4 sẽ truyền dữ liệu ảnh về máy tính theo từng byte, mỗi byte tương ứng một pixel.

```
void USART1_IRQHandler(void)
{
    char c;
    if(USART_GetITStatus(USART1, USART_IT_RXNE)==SET)
     {
        c=USART_ReceiveData(USART1); //nhan duoc 1 ky tu
        if((c&0xff)=='B') // neu ky tu la B, gui anh ve may tinh
           {
            for (i = 0; i < 76800; i++)
              {
                USART1_send_byte(&qvga_frame[i]);
              }
           }
        USART_ClearFlag(USART1, USART_FLAG_RXNE);
     }
}
```

2.3.3 Nhận dữ liệu bằng Matlab

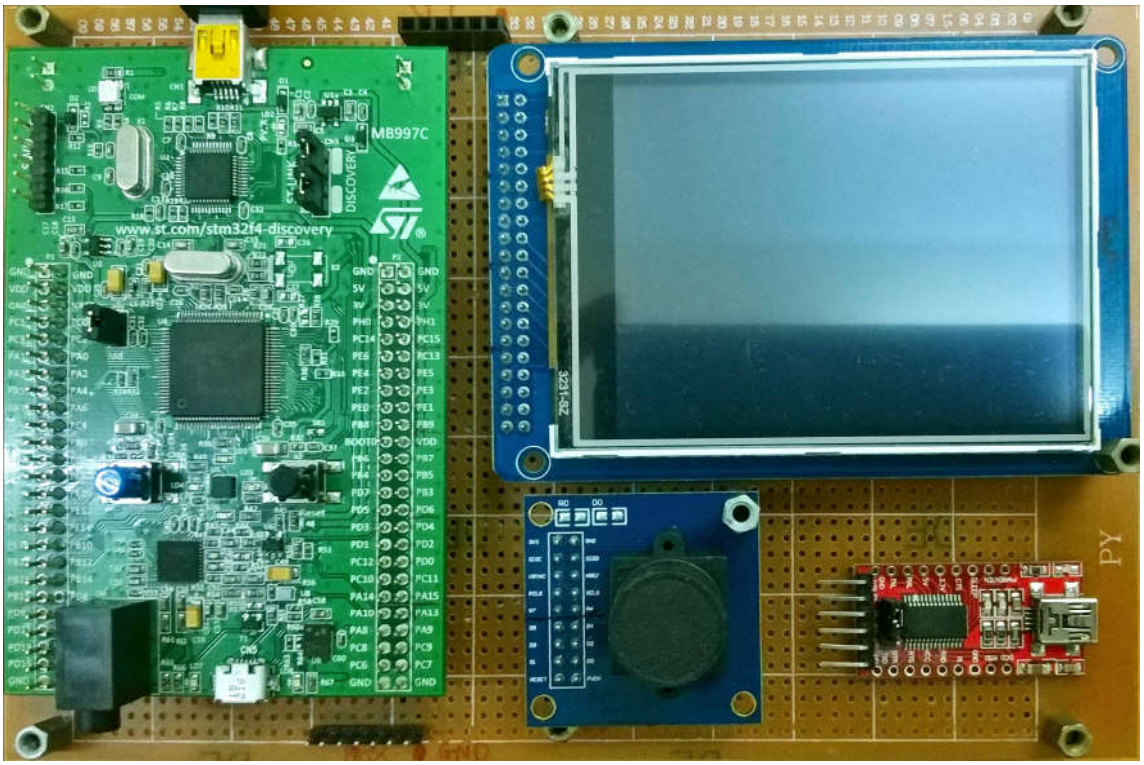
Matlab là một công cụ toán học mạnh mẽ, được ứng dụng nhiều trong nghiên cứu khoa học, tính toán trong kỹ thuật, đặc biệt là các bài toán về ma trận. Matlab còn cung cấp các Toolbox chuyên dụng để giải quyết các vấn đề cụ thể như xử lý ảnh, xử lý số tín hiệu, mô phỏng… Trong phạm vi luận văn, chỉ sử dụng Matlab để lập trình thu dữ liệu qua cổng COM, hiển thị hình ảnh lên màn hình. Chương trình như sau:

```
clear all
s = serial('COM4'); % tao doi tuong giao tiep voi cong COM4
s.Baudrate = 1000000; % truyen voi toc do 1Mbps
s.DataBits = 8;
s.StopBits = 1;
s.Parity = 'none';
s.Timeout = 10; % 10 giay
s.InputBufferSize = 76800; % buffer luu anh QVGA 240x320
fopen(s); % ket noi COM4
% gui ky tu B den kit, yeu cau gui anh ve PC
fprintf(s,'B');
% doc du lieu STM32F4 truyen ve qua USART1
out = fgetl(s);
A = double(out); % tra ket qua ve dang ma tran A 1x76800
```

```
% chuyen doi du lieu ve thanh ma tran anh 240x320
image = zeros(240,320); % khoi tao bo dem anh
row =1; col = 1;
for i = 1:76800;
 image(row,col) = A(i);
 if  mod(i,320) == 0
     row=row+1;
     col = 0;
 end
 col=col+1;
end
% chuyen doi thanh kieu du lieu 8 bit, anh da muc xam
x=uint8(image);
imshow(x); % xem anh
fclose(s); % ngat ket noi
delete(s); % xoa doi tuong
clear all;
```

CHƯƠNG III: KẾT QUẢ THỰC NGHIỆM

3.1. Ảnh chụp toàn bộ hệ thống



Hình 3.1: Hình ảnh hệ thống ghép nối

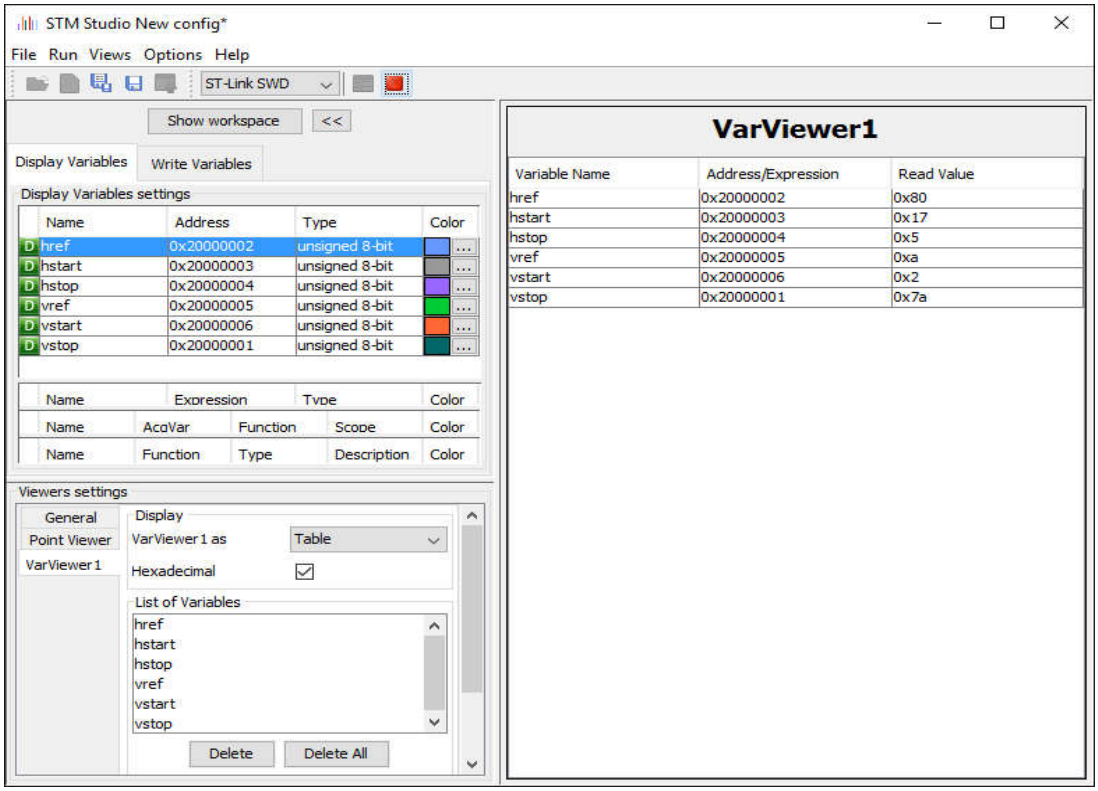
Hệ thống ghép nối bao gồm:

- 01 Kit STM32F407VG Discovery
- 01 Camera OV7670 no FIFO
- 01 màn hình LCD 3,2” (IC điều khiển ILI9341)
- 01 module chuyển đổi USB to COM FT232

3.2. Lập trình thanh ghi Camera

Sử dụng hàm ghi dữ liệu vào một số thanh ghi Camera OV7670, sau đó đọc lại giá trị thanh ghi vừa thay đổi giá trị. Dùng phần mềm STMStudio để quan sát giá trị đọc được của thanh ghi có đúng với giá trị ghi vào Camera.

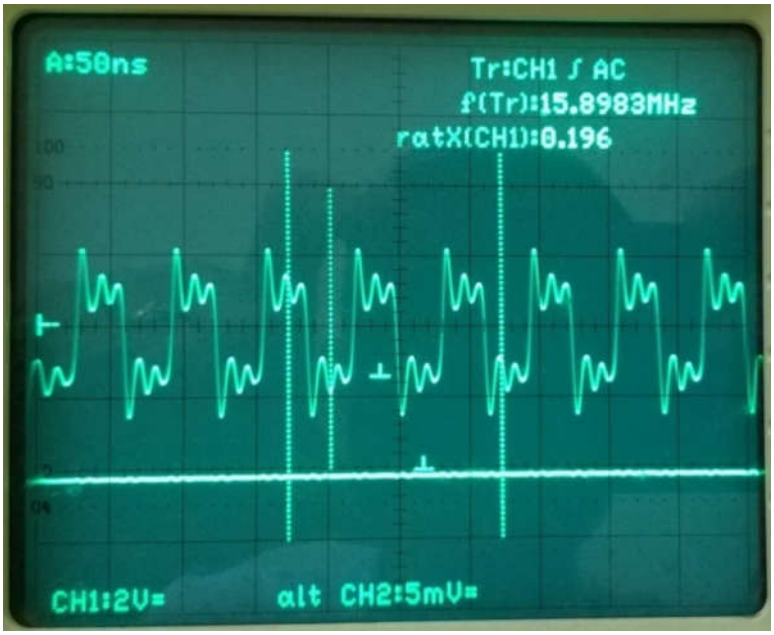
Hình 3.2 thể hiện kết quả đọc một số thanh ghi thiết lập cửa sổ của Camera OV7670.



Hình 3.2: Sử dụng STMStudio quan sát giá trị đọc từ thanh ghi OV7670

3.3. Đo tần số xung clock XCLK cấp cho Camera OV7670

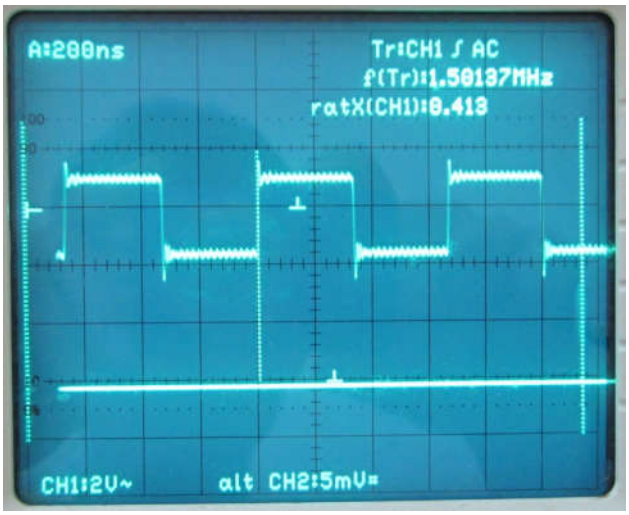
Hình 3.3 cho thấy tần số xung clock đo tại PA8 đưa đến chân XCLK của OV7670 là16 MHz.



Hình 3.3: Xung clock XCLK cấp cho Camera

3.4. Camera OV7670 hoạt động ở chế độ QQVGA, RGB565

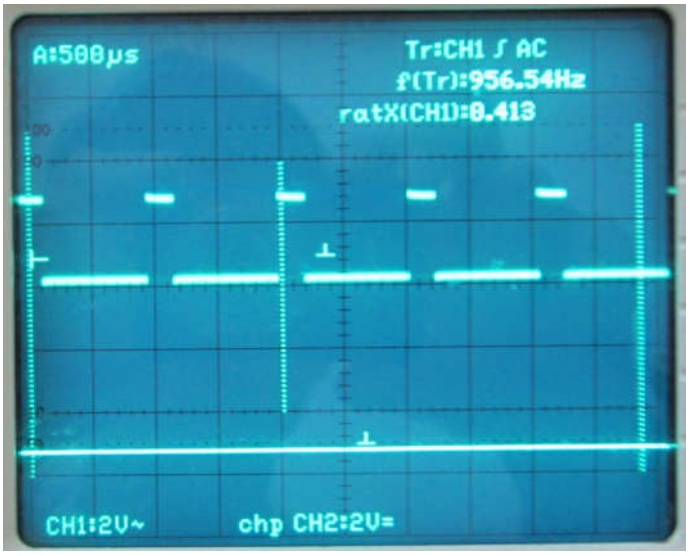
Đo tín hiệu PCLK



Hình 3.4: Tín hiệu PCLK ở chế độ QQVGA

Hình 3.4 cho thấy tín hiệu PCLK có tần số 1,5 MHz

Đo tín hiệu đồng bộ HREF



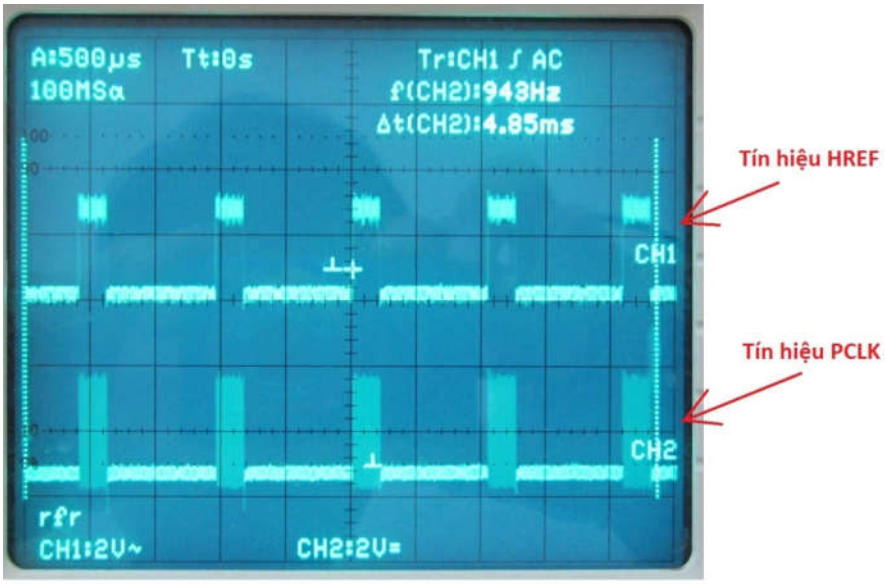
Hình 3.5: Tín hiệu HREF ở chế độ QQVGA

Hình 3.5 cho thấy tín hiệu HREF có tần số 956 Hz

Một frame có 120 dòng, như vậy tốc độ khung hình khoảng 7fps.

Để quan sát rõ tín hiệu đồng bộ dòng HREF từ Camera OV7670 và tín hiệu PCLK, cài đặt giá trị thanh ghi COM10 (địa chỉ 0x15) về giá trị 0x32 để tắt dao động PCLK khi tín hiệu HREF ở mức thấp bằng lệnh:

```
SCCB_write_reg(0x15, 0x32);
```

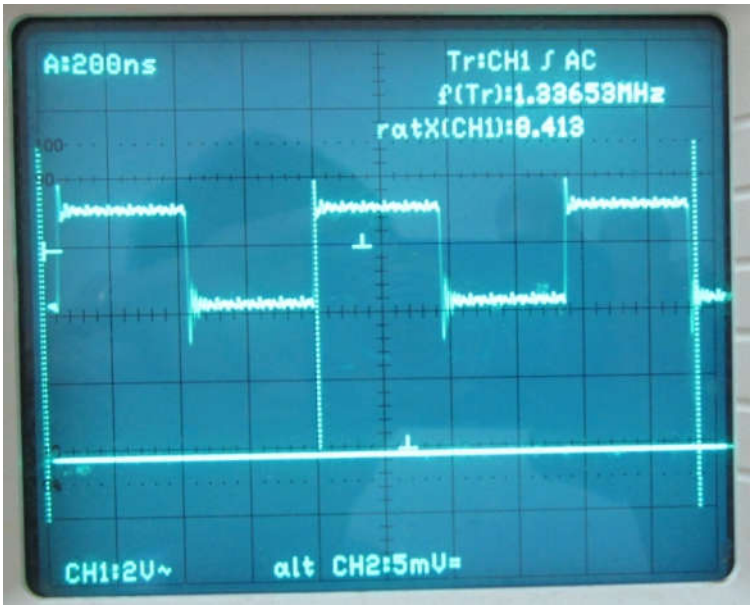


Hình 3.6: Tín hiệu đồng bộ HREF và PCLK ở chế độ QQVGA

Hình 3.6 thể hiện tín hiệu đồng bộ dòng HREF và tín hiệu pixel clock PCLK. Khi tín hiệu HREF ở mức thấp, không có tín hiệu ra (PCLK tắt), vi điều khiển không thu dữ liệu. Khi HREF ở mức cao, có xung PCLK, vi điều khiển lấy mẫu tín hiệu D0÷D7 ở sườn xuống của xung PCLK, lưu vào bộ nhớ đệm.

3.5. Camera OV7670 hoạt động ở chế độ QVGA, RGB565

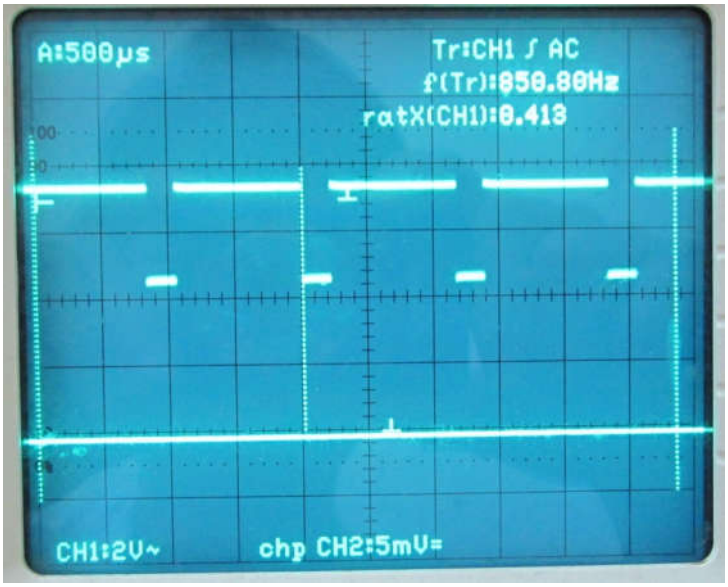
Đo tín hiệu PCLK



Hình 3.7: Tín hiệu PCLK ở chế độ QVGA, RGB565

Hình 3.7 cho thấy tần số tín hiệu PCLK là 1,336 MHz

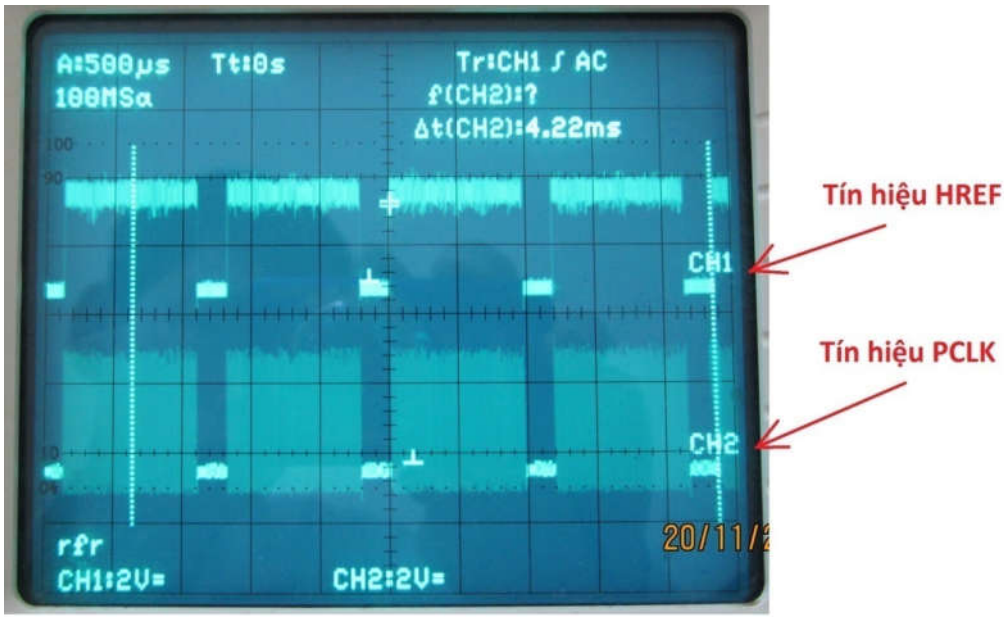
Đo tín hiệu đồng bộ HREF



Hình 3.8: Tín hiệu HREF ở chế độ QVGA, RGB565

Hình 3.8 cho thấy tần số tín hiệu HREF là 850Hz.

Một frame có 240 dòng, tốc độ khung hình khoảng 3fps.



Hình 3.9: Tín hiệu HREF và PCLK ở chế độ QVGA, RGB565

Hình 3.9 thể hiện tín hiệu đồng bộ dòng HREF và tín hiệu pixel clock PCLK khi Camera OV7670 ở chế độ QVGA, RGB565.

Khi tín hiệu HREF ở mức thấp, không có tín hiệu ra (PCLK tắt). Khi HREF ở mức cao, có xung PCLK.

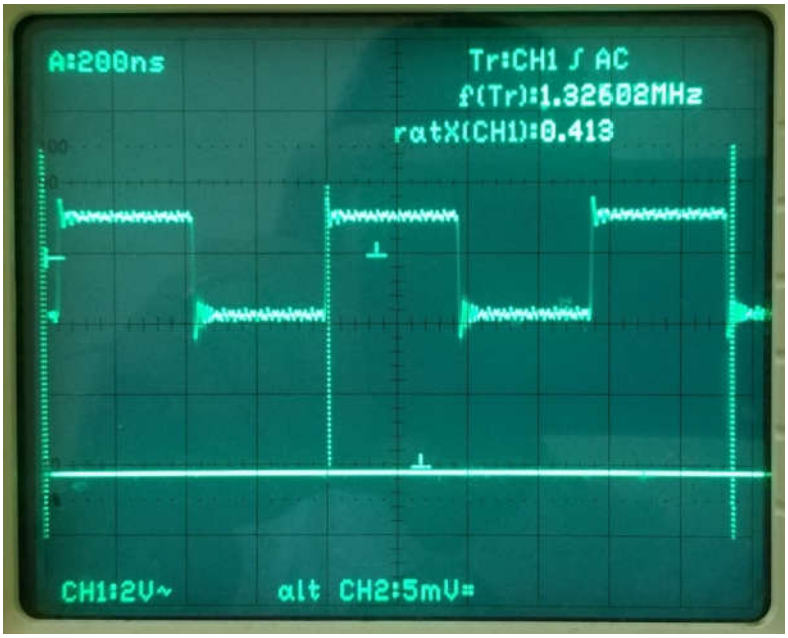
Hiển thị hình ảnh màu RGB565 lên màn hình LCD

Hình 3.10: Hiển thị ảnh màu RGB lên màn hình LCD 3,2”

3.6. Camera OV7670 hoạt động ở chế độ QVGA, YUV422

Khi Camera OV7670 phát tín hiệu video định dạng YUV, vi điều khiển thực hiện thu tín hiệu chói Y (ảnh đa mức xám) lưu vào bộ nhớ đệm, sau đó hiển thị ảnh đa mức xám ra màn hình.

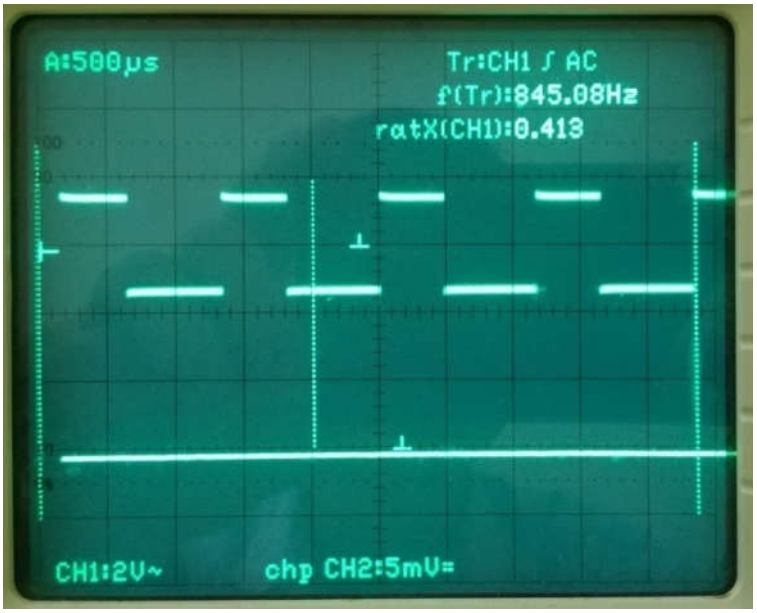
Đo tín hiệu PCLK



Hình 3.11: Tín hiệu PCLK ở chế độ QVGA, YUV

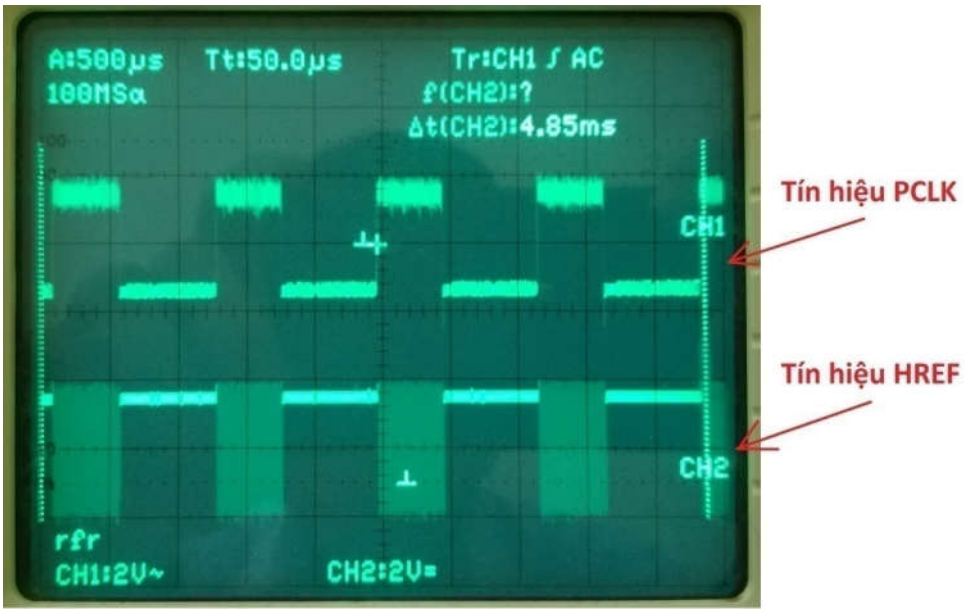
Hình 3.11 cho thấy tín hiệu PCLK có tần số 1,325 MHz

Đo tín hiệu đồng bộ HREF



Hình 3.12: Tín hiệu HREF ở chế độ QVGA, YUV

Hình 3.12 cho thấy tín hiệu HREF có tần số 845Hz. Một khung hình có 240 dòng, tốc độ khung hình khoảng 3fps.



Hình 3.13: Tín hiệu HREF và PCLK ở chế độ QVGA, YUV

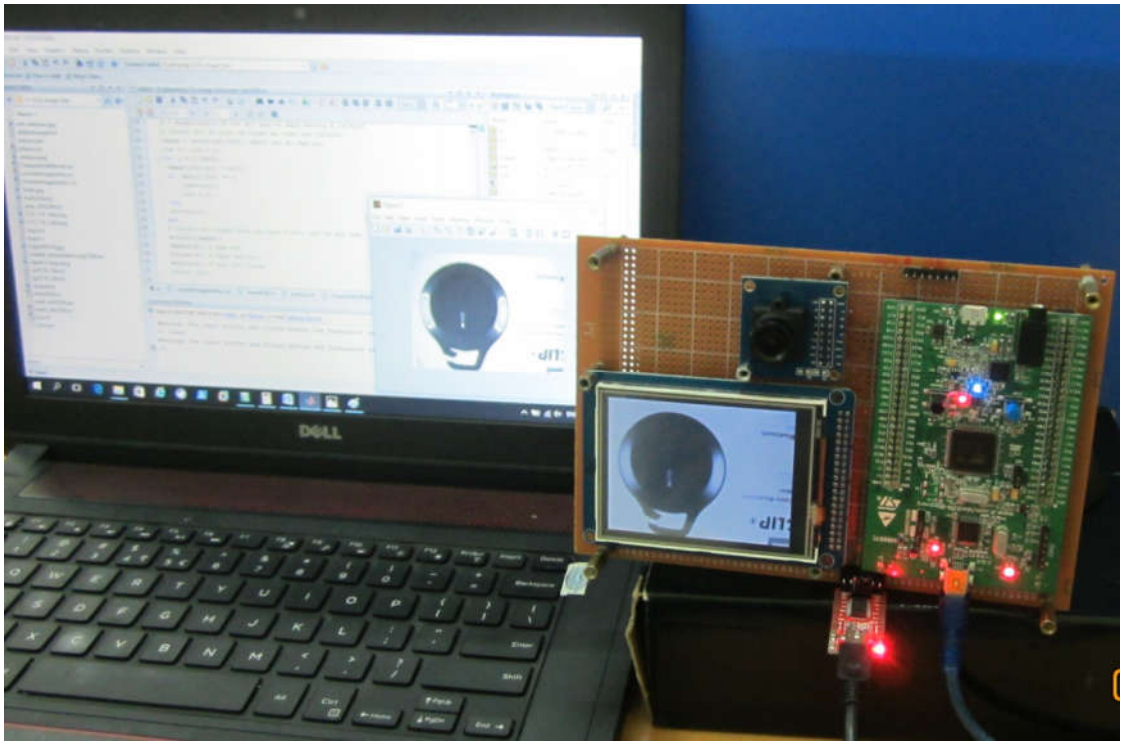
Hình 3.12 thể hiện tín hiệu đồng bộ dòng HREF và tín hiệu pixel clock PCLK khi Camera OV7670 ở chế độ QVGA, YUV.

Hiển thị hình ảnh đa mức xám lên màn hình LCD

Hình 3.14: Hình ảnh đa mức xám ở chế độ QVGA, YUV

3.7. Truyền hình ảnh về máy tính qua USART

Dữ liệu hình ảnh truyền về máy tính thành công với tốc độ 1Mbit/s



Hình 3.15: Truyền dữ liệu ảnh về máy tính qua USART

KẾT LUẬN

Căn cứ vào kết quả đạt được, luận văn đã đạt được mục tiêu đề ra. Qua thời gian làm luận văn, em đã có dịp đi sâu vào nghiên cứu tìm hiểu chi tiết các quá trình phát một ảnh màu và luồng video. Từ đó đã thiết kế lắp ráp và phát triển một hệ thống thu thập và xử lý ảnh video theo thời gian thực dựa trên Kit Vi điều khiển STM32 thành công.

Các vấn đề chính sau đây đã được giải quyết trong luận văn:

- Thiết kế lắp ráp hệ thống bắt ảnh gồm Camera OV7670 với Kit vi điều khiển STM32F4 Discovery hiển thị trên màn hình tinh thể lỏng LCD 3,2".

- Phát triển phần mềm nhúng cho phép cấu hình hệ thống và bắt các khung ảnh của luồng video.

- Hiển thị hình ảnh theo thời gian thực ra màn hình LCD 3,2".

- Truyền dữ liệu ảnh về máy tính qua USART với tốc độ lên tới 1Mbit/s.

Hệ thống có thể đáp ứng yêu cầu thời gian thực trong một số hệ thống giám sát, cảnh báo có tốc độ chậm, không yêu cầu xử lý nhanh.

Những kết quả này hy vọng sẽ là tiền đề cho những ứng dụng hữu ích trong tương lai với nhiệm vụ kỹ thuật giám sát hiện trường tại cơ quan mà em đang công tác.

TÀI LIỆU THAM KHẢO

Tiếng Việt

1. http://www.arm.vn/TinChiTiet/tabid/105/id/114/Default.aspx

Tiếng Anh

2. UM1472 User Manual STM32F4DISCOVERY, Doc ID 022256 Rev 2, www.st.com
3. RM0090 Reference manual STM32F40x, STM32F41x, STM32F42x, TM32F43x advanced ARM-based 32-bit MCUs, Doc ID 018909 Rev 3, www.st.com
4. STM32F4xx_StdPeriph_Examples\DCMI\DCMI_CameraExample, www.st.com
5. OV7670/OV7171 CMOS VGA (640x480) CameraChip™ Advanced Information Preliminary Datasheet, Version 1.3, April 5, 2006
6. OV7670/OV7171 CMOS VGA (640x480) CameraChip™ Implementation Guide, Document Version: 1.0
7. OV7670 Software Application Note.
8. OmniVision Serial Camera Control Bus (SCCB) Functional Specification, Document Version: 2.2
9. ILI9341 datasheet, Document No.: ILI9341_DS_V1.09.pdf, Version: V1.09
10. ILI9341 Application Notes, Version: Preliminary V0.6, Date: Mar. 11th 2011.
11. http://embeddedprogrammer.blogspot.com/2012/07/hacking-ov7670-camera-module-sccb-cheat.html
12. http://privateblog.info/arduino-uno-i-kamera-ov7670-primer-ispolzovaniya/
13. https://github.com/ComputerNerd/ov7670-no-ram-arduino-uno
14. https://github.com/ComputerNerd/arduino-camera-tft
15. https://stm32f4-discovery.net/2014/04/library-08-ili9341-lcd-on-stm32f429-discovery-board/
16. https://github.com/desaster/ov7670test
17. http://lxr.free-electrons.com/source/drivers/media/i2c/ov7670.c
18. http://supuntharanga.blogspot.com/2014/04/reading-serial-port-data-plotting-gray.html
19. http://www.urel.feec.vutbr.cz/MPOA/2014/cam-ov7670
20. https://github.com/erikandre/stm32-ov7670

Phụ lục 1: Chương trình đọc giá trị thanh ghi Camera OV7670 [4]

```
uint8_t SCCB_read_reg(uint8_t read_reg)
{
   // Chu ky ghi du lieu (2 pha)
   // Kiem tra trang thai duong day
   // Doi den khi trang thai duong day roi
   while(I2C_GetFlagStatus(I2C2, I2C_FLAG_BUSY));
   // I2C2 START
   I2C_GenerateSTART(I2C2, ENABLE);
   // Kiem tra trang thai duong day
   while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_MODE_SELECT)){
   };
   // Pha 1: Gui dia chi doc dư lieu cua Camera
   // Master o che do truyen du lieu.
   I2C_Send7bitAddress(I2C2, OV7670_READ_ADDR,
                           I2C_Direction_Transmitter);
   // Doi den khi Camera xac nhan bang Don't Care bit
   while(!I2C_CheckEvent(I2C2,
                         I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
   // Pha 2: Gui dia chi thanh ghi Camera can doc du lieu
   I2C_SendData(I2C2, read_reg);
   // Kiem tra, doi den khi truyen dia chi thanh ghi thanh cong
   while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_TRANSMITTED)){
   };
   I2C_GenerateSTOP(I2C2, ENABLE);
   // Ket thuc chu ky ghi du lieu 2 pha
   //-----------------------------------------------------
   // Chu ky doc du lieu thanh ghi (2 pha)
   I2C_GenerateSTART(I2C2, ENABLE);
   // Kiem tra trang thai duong day
   while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_MODE_SELECT)){ };
   // Pha 1: Gui dia chi cua Camera, Master o che do nhan du lieu.
   I2C_Send7bitAddress(I2C2, OV7670_READ_ADDR,
                                          I2C_Direction_Receiver);
   // Kiem tra slave da nhan biet che do truyen
   while(!I2C_CheckEvent(I2C2,
                         I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED)){};
   // Pha 2: doc du lieu thanh ghi
   // Tat ACK tu master de nhan biet NA bit khi ket thuc truyen
```

```
    I2C_AcknowledgeConfig(I2C2, DISABLE);
    // Kiem tra master da nhan duoc gia tri thanh ghi
    while(!I2C_CheckEvent(I2C2, I2C_EVENT_MASTER_BYTE_RECEIVED));
    // Doc du lieu tu thanh ghi OV7670, tra ve dang byte du lieu
    value_reg_OV7670 = I2C_ReceiveData(I2C2);
    I2C_GenerateSTOP(I2C2, ENABLE);
    // Ket thuc chu ky doc du lieu 2 pha
    return value_reg_OV7670; // tra ve ket qua du lieu thanh ghi
}
```

Phụ lục 2: Cấu hình chế độ QQVGA, RGB565 [7, 12, 16, 17]

```
void OV7670_QQVGA_RGB565_init(void) //
{
   SCCB_write_reg(0x12, 0x80); //COM7, RESET camera
   delay_ms(500);
   //Video format RGB565
   SCCB_write_reg( 0x12, 0x04); // COM7, output format RGB
   SCCB_write_reg( 0x40, 0xD0); // COM15, output format RGB565
   SCCB_write_reg( 0x8C, 0x00); // disable RGB444
   SCCB_write_reg( 0x04, 0x0 ); // COM1, disable CCIR656
   //Tan so dao dong noi camrea
   SCCB_write_reg( 0x6B, 0x80); //nhan tan so voi 6
   SCCB_write_reg( 0x11, 0x07); //chia tan so 16 => Fin = 6MHz
   // scalling windows
   SCCB_write_reg( 0x0C, 0x04 ); // COM3, DCW enable
   SCCB_write_reg( 0x3E, 0x1A ); // COM14, chia tan so PCLK: chia 4
   SCCB_write_reg( 0x70, 0x3A ); // SCALLING_XSC
   SCCB_write_reg( 0x71, 0x35 ); // SCALLING_YSC
   SCCB_write_reg( 0x72, 0x22 ); // downsample VGA by 4 -> QQVGA
   SCCB_write_reg( 0x73, 0xf2 ); //
   SCCB_write_reg( 0x1E, 0x0  ); // MVFP, Mirror/Flip hinh anh
   //SCCB_write_reg(  0x15, 0x32 ); // COM10, tat tin hieu PCLK khi
HREF bank
   // Hardware window QQVGA
   SCCB_write_reg( 0x3A, 0x04 ); // TSLB, cho phep thay doi tham so
hardware windows
   SCCB_write_reg( 0x17, 0x16 ); // REG_HSTART
   SCCB_write_reg( 0x18, 0x04 ); // REG_HSTOP
   SCCB_write_reg( 0x32, 0x24 ); // REG_HREF
   SCCB_write_reg( 0x19, 0x02 ); // REG_VSTART
   SCCB_write_reg( 0x1A, 0x7a ); // REG_VSTOP
   SCCB_write_reg( 0x03, 0x0a ); // REG_VREF
   // Matrix coefficients, saturation = 0,
   SCCB_write_reg( 0x4f, 0x80 ); // MTX1
   SCCB_write_reg( 0x50, 0x80 ); // MTX2
   SCCB_write_reg( 0x51, 0x0  ); // MTX3
   SCCB_write_reg( 0x52, 0x22 ); // MTX4
   SCCB_write_reg( 0x53, 0x5e ); // MTX5
   SCCB_write_reg( 0x54, 0x80 ); // MTX6
```

```
   SCCB_write_reg( 0x58, 0x9e ); // MTXS
   // Gamma
   SCCB_write_reg( 0x7a, 0x20 ); // SLOP
   SCCB_write_reg( 0x7b, 0x10 ); // GAM1
   SCCB_write_reg( 0x7c, 0x1e ); // GAM2
   SCCB_write_reg( 0x7d, 0x35 ); // GAM3
   SCCB_write_reg( 0x7e, 0x5a ); // GAM4
   SCCB_write_reg( 0x7f, 0x69 ); // GAM5
   SCCB_write_reg( 0x80, 0x76 ); // GAM6
   SCCB_write_reg( 0x81, 0x80 ); // GAM7
   SCCB_write_reg( 0x82, 0x88 ); // GAM8
   SCCB_write_reg( 0x83, 0x8f ); // GAM9
   SCCB_write_reg( 0x84, 0x96 ); // GAM10
   SCCB_write_reg( 0x85, 0xa3 ); // GAM11
   SCCB_write_reg( 0x86, 0xaf ); // GAM12
   SCCB_write_reg( 0x87, 0xc4 ); // GAM13
   SCCB_write_reg( 0x88, 0xd7 ); // GAM14
   SCCB_write_reg( 0x89, 0xe8 ); // GAM15
   //While balance, can bang trang
   SCCB_write_reg( 0x13, 0xA7 ); // COM8, AWB on | FASTAEC | AECSTEP
| AGC on | AEC on
   SCCB_write_reg( 0x43, 0x0a ); // AWBC1
   SCCB_write_reg( 0x44, 0xf0 ); // AWBC2
   SCCB_write_reg( 0x45, 0x34 ); // AWBC3
   SCCB_write_reg( 0x46, 0x58 ); // AWBC4
   SCCB_write_reg( 0x47, 0x28 ); // AWBC5
   SCCB_write_reg( 0x48, 0x3a ); // AWBC6
   SCCB_write_reg( 0x59, 0x88 ); // AWBC7
   SCCB_write_reg( 0x5a, 0x88 ); // AWBC8
   SCCB_write_reg( 0x5b, 0x44 ); // AWBC9
   SCCB_write_reg( 0x5c, 0x67 ); // AWBC10
   SCCB_write_reg( 0x5d, 0x49 ); // AWBC11
   SCCB_write_reg( 0x5e, 0x0e ); // AWBC12
   SCCB_write_reg( 0x6c, 0x0a ); // AWBCTR3
   SCCB_write_reg( 0x6d, 0x55 ); // AWBCTR2
   SCCB_write_reg( 0x6e, 0x11 ); // AWBCTR1
   SCCB_write_reg( 0x6f, 0x9f ); // AWBCTR0, 9e -> advance AWB, 9f -
> simple AWB
   SCCB_write_reg( 0x6a, 0x40 ); // G chanel gain
   SCCB_write_reg( 0x01, 0x40 ); // B chanel gain
```

```
    SCCB_write_reg( 0x02, 0x40 ); // R chanel gain
    SCCB_write_reg( 0x14, 0x68 ); // REG_COM9
    // Bright, do sang hinh anh
    SCCB_write_reg( 0x55, 0x00 ); // Bright =0
    //SCCB_write_reg( 0x55, 0x18 ); // Bright =1
    //SCCB_write_reg( 0x55, 0x30 ); // Bright =2
    //SCCB_write_reg( 0x55, 0x98 ); // Bright =-1
    //SCCB_write_reg( 0x55, 0xb0 ); // Bright =-2
    // Contrast, do tuong phan hinh anh
    SCCB_write_reg( 0x56, 0x40 ); // contrast =0
    //SCCB_write_reg( 0x56, 0x50 ); // contrast =1
    //SCCB_write_reg( 0x56, 0x60 ); // contrast =2
    //SCCB_write_reg( 0x56, 0x38 ); // contrast =-1
    //SCCB_write_reg( 0x56, 0x40 ); // contrast =-2
    // Hieu ung: normal
    SCCB_write_reg( 0x67, 0xc0 );
    SCCB_write_reg( 0x68, 0x80 );
    // Banding filter 50Hz
    SCCB_write_reg( 0x3B, 0x0A ); // COM11, chon bo loc 50Hz
    SCCB_write_reg( 0x9d, 0x4c ); // BD50ST, 50Hz banding filter
value, active khi COM8[5] high, COM11[3] high
    SCCB_write_reg( 0xA5, 0x05 ); // BD50MAX, max banding filter step
    SCCB_write_reg( 0x0E, 0x61 ); // COM5
    SCCB_write_reg( 0x0F, 0x4b ); // COM6
    SCCB_write_reg( 0x16, 0x02 ); //
    SCCB_write_reg( 0x21, 0x02 ); // ADCCTR1
    SCCB_write_reg( 0x22, 0x91 ); // ADCCTR2
    SCCB_write_reg( 0x29, 0x07 ); // RSVD
    SCCB_write_reg( 0x33, 0x0b ); // CHLF
    SCCB_write_reg( 0x35, 0x0b ); //
    SCCB_write_reg( 0x37, 0x1d ); // ADC control
    SCCB_write_reg( 0x38, 0x71 ); // ADC and Analog Common mode
control
    SCCB_write_reg( 0x39, 0x2a ); // ADC Offset control
    SCCB_write_reg( 0x4d, 0x40 ); // DM Pos, dummy row position
    SCCB_write_reg( 0x4e, 0x20 ); //
    SCCB_write_reg( 0x8d, 0x4f ); //
    SCCB_write_reg( 0x8e, 0x0  ); //
    SCCB_write_reg( 0x8f, 0x0  ); //
    SCCB_write_reg( 0x90, 0x0  ); //
```

```
   SCCB_write_reg( 0x91, 0x0  ); //
   SCCB_write_reg( 0x96, 0x0  ); //
   SCCB_write_reg( 0x9a, 0x0  ); //
   SCCB_write_reg( 0xb0, 0x84 ); // No document
   SCCB_write_reg(  0xb1,  0x0c  );  //  ABLC1,  1100,  enable  ABLC
function
   SCCB_write_reg( 0xb2, 0x0e ); //
   SCCB_write_reg( 0xb3, 0x82 ); // ABLC target
   SCCB_write_reg( 0xb8, 0x0a ); //
   SCCB_write_reg( 0x3F, 0x0  ); // REG_EDGE Enhancement Adjustment
   SCCB_write_reg( 0x74, 0x10 ); // REG74 0001 0000, digital gain
manual control bypass.
   SCCB_write_reg( 0x75, 0x05 ); // REG75, Edge enhancement lower
limit
   SCCB_write_reg( 0x76, 0xe1 ); // REG76, 1110 0001, [6:5]enable
black/white pixel correct, [4:0]Edge enhancement higher limit
   SCCB_write_reg( 0x77, 0x01 ); // REG77, de-noise range control
   SCCB_write_reg( 0x4c, 0x0  ); // De-noise strength
   SCCB_write_reg( 0x4b, 0x09 ); // UV average enable
   SCCB_write_reg( 0xc9, 0x60 ); // Saturation control, bao hoa
   SCCB_write_reg( 0x34, 0x11 ); // Array reference control
}
```

Phụ lục 3: Cấu hình chế độ QVGA, RGB565 [7, 17, 20]

```
void OV7670_QVGA_RGB565_init(void) //
{
   SCCB_write_reg(0x12, 0x80); //COM7, RESET camera
   delay_ms(500);
   //Video format RGB565
   SCCB_write_reg( 0x12, 0x04 ); // COM7, output format RGB
   SCCB_write_reg( 0x40, 0xD0 ); // COM15, output format RGB565
   SCCB_write_reg( 0x8C, 0x00 ); // disable RGB444
   SCCB_write_reg( 0x04, 0x0  ); // COM1, disable CCIR656
   //Tan so dao dong noi camrea
   SCCB_write_reg( 0x6B, 0x0  ); // bypass PLL
   SCCB_write_reg( 0x11, 0x05 ); // chia tan so 12 => Fin = PCLK =
1.325MHz

   SCCB_write_reg( 0x3a, 0x04 ); // TSLB, cho phep hardware window
   SCCB_write_reg( 0x32, 0x80 ); // HREF
   SCCB_write_reg( 0x17, 0x16 ); // HSTART
   SCCB_write_reg( 0x18, 0x04 ); // HSTOP
   SCCB_write_reg( 0x19, 0x02 ); // VSTART
   SCCB_write_reg( 0x1a, 0x7b ); // VSTOP
   SCCB_write_reg( 0x03, 0x06 ); // VREF

   SCCB_write_reg( 0x0c, 0x0  ); // COM3, VGA
   SCCB_write_reg( 0x3e, 0x0  ); // COM14,
   SCCB_write_reg( 0x70, 0x0  ); // SCALLING XSC
   SCCB_write_reg( 0x71, 0x0  ); // SCALLING YSC
   SCCB_write_reg( 0x72, 0x11 ); // SCALLING DCWCTR, downsample 2
   SCCB_write_reg( 0x73, 0x0  ); // SCALLING PCLK DIV, bypass
   SCCB_write_reg( 0xa2, 0x02 ); // SCALLING PCLK DELAY
   //SCCB_write_reg(0x15, 0x32); // tat dao dong PCLK khi HREF bannk
   /*------------- Color setting ------------------------*/
   // Matrix coefficients, saturation = 0,
   SCCB_write_reg( 0x4f, 0x80 ); // MTX1
   SCCB_write_reg( 0x50, 0x80 ); // MTX2
   SCCB_write_reg( 0x51, 0x0  ); // MTX3
   SCCB_write_reg( 0x52, 0x22 ); // MTX4
   SCCB_write_reg( 0x53, 0x5e ); // MTX5
```

```
   SCCB_write_reg( 0x54, 0x80 ); // MTX6
   SCCB_write_reg( 0x58, 0x9e ); // MTXS
   // Gamma
   SCCB_write_reg( 0x7a, 0x20 ); // SLOP
   SCCB_write_reg( 0x7b, 0x10 ); // GAM1
   SCCB_write_reg( 0x7c, 0x1e ); // GAM2
   SCCB_write_reg( 0x7d, 0x35 ); // GAM3
   SCCB_write_reg( 0x7e, 0x5a ); // GAM4
   SCCB_write_reg( 0x7f, 0x69 ); // GAM5
   SCCB_write_reg( 0x80, 0x76 ); // GAM6
   SCCB_write_reg( 0x81, 0x80 ); // GAM7
   SCCB_write_reg( 0x82, 0x88 ); // GAM8
   SCCB_write_reg( 0x83, 0x8f ); // GAM9
   SCCB_write_reg( 0x84, 0x96 ); // GAM10
   SCCB_write_reg( 0x85, 0xa3 ); // GAM11
   SCCB_write_reg( 0x86, 0xaf ); // GAM12
   SCCB_write_reg( 0x87, 0xc4 ); // GAM13
   SCCB_write_reg( 0x88, 0xd7 ); // GAM14
   SCCB_write_reg( 0x89, 0xe8 ); // GAM15
   //While balance, can bang trang
   SCCB_write_reg( 0x13, 0xA7 ); // COM8, AWB on | FASTAEC | AECSTEP
| AGC on | AEC on
   SCCB_write_reg( 0x43, 0x0a ); // AWBC1
   SCCB_write_reg( 0x44, 0xf0 ); // AWBC2
   SCCB_write_reg( 0x45, 0x34 ); // AWBC3
   SCCB_write_reg( 0x46, 0x58 ); // AWBC4
   SCCB_write_reg( 0x47, 0x28 ); // AWBC5
   SCCB_write_reg( 0x48, 0x3a ); // AWBC6
   SCCB_write_reg( 0x59, 0x88 ); // AWBC7
   SCCB_write_reg( 0x5a, 0x88 ); // AWBC8
   SCCB_write_reg( 0x5b, 0x44 ); // AWBC9
   SCCB_write_reg( 0x5c, 0x67 ); // AWBC10
   SCCB_write_reg( 0x5d, 0x49 ); // AWBC11
   SCCB_write_reg( 0x5e, 0x0e ); // AWBC12
   SCCB_write_reg( 0x6c, 0x0a ); // AWBCTR3
   SCCB_write_reg( 0x6d, 0x55 ); // AWBCTR2
   SCCB_write_reg( 0x6e, 0x11 ); // AWBCTR1
   SCCB_write_reg( 0x6f, 0x9f ); // AWBCTR0, 9e -> advance AWB, 9f -
> simple AWB
   SCCB_write_reg( 0x6a, 0x20 ); // G chanel gain
```

```
    SCCB_write_reg( 0x01, 0x20 ); // B chanel gain
    SCCB_write_reg( 0x02, 0x20 ); // R chanel gain
    SCCB_write_reg( 0x14, 0x48 ); // REG_COM9
    // Bright, do sang hinh anh
    SCCB_write_reg( 0x55, 0x00 ); // Bright =0
    //SCCB_write_reg( 0x55, 0x18 ); // Bright =1
    //SCCB_write_reg( 0x55, 0x30 ); // Bright =2
    //SCCB_write_reg( 0x55, 0x98 ); // Bright =-1
    //SCCB_write_reg( 0x55, 0xb0 ); // Bright =-2
    // Contrast, do tuong phan hinh anh
    SCCB_write_reg( 0x56, 0x40 ); // contrast =0
    //SCCB_write_reg( 0x56, 0x50 ); // contrast =1
    //SCCB_write_reg( 0x56, 0x60 ); // contrast =2
    //SCCB_write_reg( 0x56, 0x38 ); // contrast =-1
    //SCCB_write_reg( 0x56, 0x40 ); // contrast =-2
    // Hieu ung: normal
    SCCB_write_reg( 0x67, 0xc0 );
    SCCB_write_reg( 0x68, 0x80 );
    // Banding filter 50Hz
    SCCB_write_reg( 0x3B, 0x0A ); // COM11, chon bo loc 50Hz
    SCCB_write_reg( 0x9d, 0x4c ); // BD50ST, 50Hz banding filter
value, active khi COM8[5] high, COM11[3] high
    SCCB_write_reg( 0xA5, 0x05 ); // BD50MAX, max banding filter step
    SCCB_write_reg( 0x0E, 0x61 ); // REG_COM5
    SCCB_write_reg( 0x0F, 0x4b ); // REG_COM6
    SCCB_write_reg( 0x16, 0x02 ); //
    SCCB_write_reg( 0x21, 0x02 ); // ADCCTR1
    SCCB_write_reg( 0x22, 0x91 ); // ADCCTR2
    SCCB_write_reg( 0x29, 0x07 ); // RSVD
    SCCB_write_reg( 0x33, 0x0b ); // CHLF
    SCCB_write_reg( 0x35, 0x0b ); //
    SCCB_write_reg( 0x37, 0x1d ); // ADC control
    SCCB_write_reg( 0x38, 0x71 ); //
    SCCB_write_reg( 0x39, 0x2a ); // ADC Offset control
    SCCB_write_reg( 0x4d, 0x40 ); // DM Pos, dummy row position
    SCCB_write_reg( 0x4e, 0x20 ); //
    SCCB_write_reg( 0x8d, 0x4f ); //
    SCCB_write_reg( 0x8e, 0x0  ); //
    SCCB_write_reg( 0x8f, 0x0  ); //
    SCCB_write_reg( 0x90, 0x0  ); //
```

```
   SCCB_write_reg( 0x91, 0x0  ); //
   SCCB_write_reg( 0x96, 0x0  ); //
   SCCB_write_reg( 0x9a, 0x0  ); //
   SCCB_write_reg( 0xb0, 0x84 ); // No document
   SCCB_write_reg( 0xb1, 0x0c ); // ABLC1, enable ABLC function
   SCCB_write_reg( 0xb2, 0x0e ); //
   SCCB_write_reg( 0xb3, 0x82 ); // ABLC target
   SCCB_write_reg( 0xb8, 0x0a ); //
   SCCB_write_reg( 0x3F, 0x0  ); // REG_EDGE Enhancement Adjustment
   SCCB_write_reg( 0x74, 0x10 ); // REG74 0001 0000, digital gain
manual control bypass.
   SCCB_write_reg( 0x75, 0x05 ); // REG75, Edge enhancement lower
limit
   SCCB_write_reg( 0x76, 0xe1 ); // REG76, 1110 0001, [6:5]enable
black/white pixel correct, [4:0]Edge enhancement higher limit
   SCCB_write_reg( 0x77, 0x01 ); // REG77, de-noise range control
   SCCB_write_reg( 0x4c, 0x0  ); // De-noise strength
   SCCB_write_reg( 0x4b, 0x09 ); // UV average enable
   SCCB_write_reg( 0x34, 0x11 ); // Array reference control
   SCCB_write_reg( 0x3D, 0x40 ); // COM13, UV saturation auto, gia
tri luu tai thanh ghi 0xC9
   SCCB_write_reg( 0xc9, 0x60 ); // Saturation control, bao hoa
}
```

Phụ lục 4: Cấu hình chế độ QVGA, YUV [7, 12, 13, 17]

```
void OV7670_QVGA_YUV422_init(void) // PCLK= 1.324MHz, HREF= 845Hz
{
   SCCB_write_reg(0x12, 0x80); //COM7, RESET camera
   delay_ms(500);
   //Video format YUV422
   SCCB_write_reg( 0x12, 0x0 ); // COM7, output format YUV
   //SCCB_write_reg( 0x40, 0xD0 ); // COM15, output format RGB565
   SCCB_write_reg( 0x8C, 0x00 ); // disable RGB444
   SCCB_write_reg( 0x04, 0x0  ); // COM1, disable CCIR656
   //Tan so dao dong noi camrea
   //SCCB_write_reg( 0x6B, 0x80  ); // bypass PLL
   SCCB_write_reg( 0x11, 0x02 ); // chia tan so

   SCCB_write_reg( 0x3a, 0x04 ); // TSLB, cho phep hardware window
   SCCB_write_reg( 0x32, 0xA4 ); // HREF
   SCCB_write_reg( 0x17, 0x16 ); // HSTART
   SCCB_write_reg( 0x18, 0x04 ); // HSTOP
   SCCB_write_reg( 0x19, 0x02 ); // VSTART
   SCCB_write_reg( 0x1a, 0x7a ); // VSTOP
   SCCB_write_reg( 0x03, 0x0A ); // VREF

   SCCB_write_reg( 0x0c, 0x04 ); // COM3, VGA
   SCCB_write_reg( 0x3e, 0x19 ); // COM14,
   SCCB_write_reg( 0x70, 0x0  ); // SCALLING XSC
   SCCB_write_reg( 0x71, 0x0  ); // SCALLING YSC
   SCCB_write_reg( 0x72, 0x11 ); // SCALLING DCWCTR, downsample 2
   SCCB_write_reg( 0x73, 0xf1 ); // SCALLING PCLK DIV, bypass
   //SCCB_write_reg( 0xa2, 0x02 ); // SCALLING PCLK DELAY
   SCCB_write_reg(0x15, 0x32); // tat dao dong PCLK khi HREF bannk
   /*------------- Color setting ------------------------*/
   // Matrix coefficients, saturation = 0,
   SCCB_write_reg( 0x4f, 0x80 ); // MTX1
   SCCB_write_reg( 0x50, 0x80 ); // MTX2
   SCCB_write_reg( 0x51, 0x0  ); // MTX3
   SCCB_write_reg( 0x52, 0x22 ); // MTX4
   SCCB_write_reg( 0x53, 0x5e ); // MTX5
   SCCB_write_reg( 0x54, 0x80 ); // MTX6
```

```
    SCCB_write_reg( 0x58, 0x9e ); // MTXS
    // Gamma
    SCCB_write_reg( 0x7a, 0x20 ); // SLOP
    SCCB_write_reg( 0x7b, 0x10 ); // GAM1
    SCCB_write_reg( 0x7c, 0x1e ); // GAM2
    SCCB_write_reg( 0x7d, 0x35 ); // GAM3
    SCCB_write_reg( 0x7e, 0x5a ); // GAM4
    SCCB_write_reg( 0x7f, 0x69 ); // GAM5
    SCCB_write_reg( 0x80, 0x76 ); // GAM6
    SCCB_write_reg( 0x81, 0x80 ); // GAM7
    SCCB_write_reg( 0x82, 0x88 ); // GAM8
    SCCB_write_reg( 0x83, 0x8f ); // GAM9
    SCCB_write_reg( 0x84, 0x96 ); // GAM10
    SCCB_write_reg( 0x85, 0xa3 ); // GAM11
    SCCB_write_reg( 0x86, 0xaf ); // GAM12
    SCCB_write_reg( 0x87, 0xc4 ); // GAM13
    SCCB_write_reg( 0x88, 0xd7 ); // GAM14
    SCCB_write_reg( 0x89, 0xe8 ); // GAM15
    //While balance, can bang trang
    SCCB_write_reg( 0x13, 0xA7 ); // COM8, 1010 0111, AWB on |
FASTAEC | AECSTEP | AGC on | AEC on
    SCCB_write_reg( 0x43, 0x0a ); // AWBC1
    SCCB_write_reg( 0x44, 0xf0 ); // AWBC2
    SCCB_write_reg( 0x45, 0x34 ); // AWBC3
    SCCB_write_reg( 0x46, 0x58 ); // AWBC4
    SCCB_write_reg( 0x47, 0x28 ); // AWBC5
    SCCB_write_reg( 0x48, 0x3a ); // AWBC6
    SCCB_write_reg( 0x59, 0x88 ); // AWBC7
    SCCB_write_reg( 0x5a, 0x88 ); // AWBC8
    SCCB_write_reg( 0x5b, 0x44 ); // AWBC9
    SCCB_write_reg( 0x5c, 0x67 ); // AWBC10
    SCCB_write_reg( 0x5d, 0x49 ); // AWBC11
    SCCB_write_reg( 0x5e, 0x0e ); // AWBC12
    SCCB_write_reg( 0x6c, 0x0a ); // AWBCTR3
    SCCB_write_reg( 0x6d, 0x55 ); // AWBCTR2
    SCCB_write_reg( 0x6e, 0x11 ); // AWBCTR1
    SCCB_write_reg( 0x6f, 0x9e ); // AWBCTR0, 9e -> advance AWB, 9f -
> simple AWB
    SCCB_write_reg( 0x6a, 0x20 ); // G chanel gain
    SCCB_write_reg( 0x01, 0x20 ); // B chanel gain
```

```
   SCCB_write_reg( 0x02, 0x20 ); // R chanel gain
   SCCB_write_reg( 0x14, 0x28 ); // REG_COM9, automatic gain max 8x
+ magic rsvd bit
   // Bright, do sang hinh anh
   SCCB_write_reg( 0x55, 0x00 ); // Bright =0
   //SCCB_write_reg( 0x55, 0x18 ); // Bright =1
   //SCCB_write_reg( 0x55, 0x30 ); // Bright =2
   //SCCB_write_reg( 0x55, 0x98 ); // Bright =-1
   //SCCB_write_reg( 0x55, 0xb0 ); // Bright =-2
   // Contrast, do tuong phan hinh anh
   SCCB_write_reg( 0x56, 0x40 ); // contrast =0
   //SCCB_write_reg( 0x56, 0x50 ); // contrast =1
   //SCCB_write_reg( 0x56, 0x60 ); // contrast =2
   //SCCB_write_reg( 0x56, 0x38 ); // contrast =-1
   //SCCB_write_reg( 0x56, 0x40 ); // contrast =-2
   // Hieu ung: normal
   //SCCB_write_reg( 0x3a, 0x04 );
   SCCB_write_reg( 0x67, 0xc0 );
   SCCB_write_reg( 0x68, 0x80 );
   // Banding filter 50Hz
   SCCB_write_reg( 0x3B, 0x0A ); // COM11, 0000 1010, chon bo loc
50Hz: BD50ST (0x9D), exposure timmingc < limit
   SCCB_write_reg( 0x9d, 0x4c ); // BD50ST, 50Hz banding filter
value, active khi COM8[5] high, COM11[3] high
   SCCB_write_reg( 0xA5, 0x05 ); // BD50MAX, max banding filter step
   SCCB_write_reg( 0x0E, 0x61 ); // REG_COM5
   SCCB_write_reg( 0x0F, 0x4b ); // REG_COM6
   SCCB_write_reg( 0x16, 0x02 ); //
   SCCB_write_reg( 0x21, 0x02 ); // ADCCTR1
   SCCB_write_reg( 0x22, 0x91 ); // ADCCTR2
   SCCB_write_reg( 0x29, 0x07 ); // RSVD
   SCCB_write_reg( 0x33, 0x0b ); // CHLF
   SCCB_write_reg( 0x35, 0x0b ); //
   SCCB_write_reg( 0x37, 0x1d ); // ADC control
   SCCB_write_reg( 0x38, 0x71 ); // ADC and Analog Common mode
control
   SCCB_write_reg( 0x39, 0x2a ); // ADC Offset control
   SCCB_write_reg( 0x4d, 0x40 ); // DM Pos, dummy row position
   SCCB_write_reg( 0x4e, 0x20 ); //
   SCCB_write_reg( 0x8d, 0x4f ); //
   SCCB_write_reg( 0x8e, 0x0  ); //
```

```
   SCCB_write_reg( 0x8f, 0x0  ); //
   SCCB_write_reg( 0x90, 0x0  ); //
   SCCB_write_reg( 0x91, 0x0  ); //
   SCCB_write_reg( 0x96, 0x0  ); //
   SCCB_write_reg( 0x9a, 0x0  ); //
   SCCB_write_reg( 0xb0, 0x84 ); // No document
   SCCB_write_reg( 0xb1, 0x0c ); // ABLC1, 1100, enable ABLC
function
   SCCB_write_reg( 0xb2, 0x0e ); //
   SCCB_write_reg( 0xb3, 0x82 ); // ABLC target
   SCCB_write_reg( 0xb8, 0x0a ); //
   SCCB_write_reg( 0x3F, 0x0  ); // REG_EDGE Enhancement Adjustment
   SCCB_write_reg( 0x74, 0x10 ); // REG74 0001 0000, digital gain
manual control bypass.
   SCCB_write_reg( 0x75, 0x05 ); // REG75, Edge enhancement lower
limit
   SCCB_write_reg( 0x76, 0xe1 ); // REG76, 1110 0001, [6:5]enable
black/white pixel correct, [4:0]Edge enhancement higher limit
   SCCB_write_reg( 0x77, 0x01 ); // REG77, de-noise range control
   SCCB_write_reg( 0x4c, 0x0  ); // De-noise strength
   SCCB_write_reg( 0x4b, 0x09 ); // UV average enable
   SCCB_write_reg( 0x34, 0x11 ); // Array reference control
   SCCB_write_reg( 0x3D, 0x40 ); // COM13, UV saturation auto, gia
tri luu tai thanh ghi 0xC9
   SCCB_write_reg( 0xc9, 0x60 ); // Saturation control, bao hoa
}
```

Phụ lục 5: Cấu hình đèn nền LED_A

```
/*****************************************************
Cau hinh den nen LED_A
Noi chan: LED_A -> PB0
Cau hinh khoi TIMER3 che do xung PWM -> chan PB0, Tan so: 4.9 kHz
*****************************************************/
void configBacklightPWM(void)
   {GPIO_InitTypeDef GPIO_InitStructure;
   TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
   TIM_OCInitTypeDef TIM_OCInitStructure;
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);//TIM3 Clock
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB,ENABLE);//GPIOB Clock
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; //TIM3_CH3 -> PB0
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_25MHz;
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
   GPIO_Init(GPIOB, &GPIO_InitStructure);
   GPIO_PinAFConfig(GPIOB, GPIO_PinSource0, GPIO_AF_TIM3);
   /* PWM Setup */
   TIM_TimeBaseStructure.TIM_Prescaler = 10;
   TIM_TimeBaseStructure.TIM_Period = 499;
   TIM_TimeBaseStructure.TIM_ClockDivision = 0;
   TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
   TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
   /* PWM2 Mode configuration: Channel 3 */
   TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;
   TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
   TIM_OCInitStructure.TIM_Pulse = 50;  //
   TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
   TIM_OC3Init(TIM3, &TIM_OCInitStructure);
   TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);
   TIM_ARRPreloadConfig(TIM3, ENABLE);
   TIM_Cmd(TIM3, ENABLE);
}
```

Phụ lục 6: Chương trình khởi động LCD [9, 10, 15].

```
void LCD_Initializtion(void)
{
    delay_ms(10);LCD_WriteIndex(ILI9341_POWERA);
    LCD_WriteData(0x39);
    LCD_WriteData(0x2C);
    LCD_WriteData(0x00);
    LCD_WriteData(0x34);
    LCD_WriteData(0x02);
    LCD_WriteIndex(ILI9341_POWERB);
    LCD_WriteData(0x00);
    LCD_WriteData(0xC1);
    LCD_WriteData(0x30);
    LCD_WriteIndex(ILI9341_DTCA);
    LCD_WriteData(0x85);
    LCD_WriteData(0x00);
    LCD_WriteData(0x78);
    LCD_WriteIndex(ILI9341_DTCB);
    LCD_WriteData(0x00);
    LCD_WriteData(0x00);
    LCD_WriteIndex(ILI9341_POWER_SEQ);
    LCD_WriteData(0x64);
    LCD_WriteData(0x03);
    LCD_WriteData(0x12);
    LCD_WriteData(0x81);
    LCD_WriteIndex(ILI9341_PRC);
    LCD_WriteData(0x20);
    LCD_WriteIndex(ILI9341_POWER1);
    LCD_WriteData(0x23);
    LCD_WriteIndex(ILI9341_POWER2);
    LCD_WriteData(0x10);
    LCD_WriteIndex(ILI9341_VCOM1);//Contrast control
    LCD_WriteData(0x3E);
    LCD_WriteData(0x28);//28
    LCD_WriteIndex(ILI9341_WDB);//Bright control
    LCD_WriteData(0xFF);
    LCD_WriteIndex(ILI9341_WCD);//Bright CTRL control
    LCD_WriteData(0x2C);//24
```

```
LCD_WriteIndex(ILI9341_VCOM2);
LCD_WriteData(0x86);
LCD_WriteIndex(ILI9341_MAC);  // Memory Access Control
//LCD_WriteData(0x88); // orgin 240 cot x 320 hang
LCD_WriteData(0x28); // 320 cot x 240 hang
LCD_WriteIndex(ILI9341_PIXEL_FORMAT);  // 16bit/pixel
LCD_WriteData(0x55);
LCD_WriteIndex(ILI9341_FRC);
LCD_WriteData(0x00);
LCD_WriteData(0x18);
LCD_WriteIndex(ILI9341_DFC);
LCD_WriteData(0x08);
LCD_WriteData(0x82);
LCD_WriteData(0x27);
LCD_WriteIndex(ILI9341_3GAMMA_EN);
LCD_WriteData(0x00);
LCD_WriteIndex(ILI9341_COLUMN_ADDR);
LCD_WriteData(0x00);
LCD_WriteData(0x00);
LCD_WriteData(0x00);
LCD_WriteData(0xEF);
LCD_WriteIndex(ILI9341_PAGE_ADDR);
LCD_WriteData(0x00);
LCD_WriteData(0x00);
LCD_WriteData(0x01);
LCD_WriteData(0x3F);
LCD_WriteIndex(ILI9341_GAMMA);
LCD_WriteData(0x01);
LCD_WriteIndex(ILI9341_PGAMMA);
LCD_WriteData(0x0F);
LCD_WriteData(0x31);
LCD_WriteData(0x2B);
LCD_WriteData(0x0C);
LCD_WriteData(0x0E);
LCD_WriteData(0x08);
LCD_WriteData(0x4E);
LCD_WriteData(0xF1);
LCD_WriteData(0x37);
LCD_WriteData(0x07);
```

```
    LCD_WriteData(0x10);
    LCD_WriteData(0x03);
    LCD_WriteData(0x0E);
    LCD_WriteData(0x09);
    LCD_WriteData(0x00);
    LCD_WriteIndex(ILI9341_NGAMMA);
    LCD_WriteData(0x00);
    LCD_WriteData(0x0E);
    LCD_WriteData(0x14);
    LCD_WriteData(0x03);
    LCD_WriteData(0x11);
    LCD_WriteData(0x07);
    LCD_WriteData(0x31);
    LCD_WriteData(0xC1);
    LCD_WriteData(0x48);
    LCD_WriteData(0x08);
    LCD_WriteData(0x0F);
    LCD_WriteData(0x0C);
    LCD_WriteData(0x31);
    LCD_WriteData(0x36);
    LCD_WriteData(0x0F);
    LCD_WriteIndex(ILI9341_SLEEP_OUT);
    delay_ms(100);
    LCD_WriteIndex(ILI9341_DISPLAY_ON);
}
```

Phụ lục 7: Cấu hình chế độ hoạt động khối USART1

```
void USART_Configuration(unsigned int BaudRate)//
{
   USART_InitTypeDef USART_InitStructure;
   GPIO_InitTypeDef GPIO_InitStructure;
   NVIC_InitTypeDef NVIC_InitStruct;
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
   /* Configure USART Tx as alternate function  */
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
   GPIO_Init(GPIOB, &GPIO_InitStructure);
   /* Configure USART Rx as alternate function  */
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7;
   GPIO_Init(GPIOB, &GPIO_InitStructure);
   USART_InitStructure.USART_BaudRate = BaudRate;
   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
   USART_InitStructure.USART_StopBits = USART_StopBits_1;
   USART_InitStructure.USART_Parity = USART_Parity_No;
   USART_InitStructure.USART_HardwareFlowControl =
                       USART_HardwareFlowControl_None;
   USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
   USART_Init(USART1, &USART_InitStructure);
   GPIO_PinAFConfig(GPIOB,GPIO_PinSource6,GPIO_AF_USART1);
   GPIO_PinAFConfig(GPIOB,GPIO_PinSource7,GPIO_AF_USART1);

   USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
   NVIC_InitStruct.NVIC_IRQChannel = USART1_IRQn;
   NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
   NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
   NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
   NVIC_Init(&NVIC_InitStruct);
   USART_Cmd(USART1, ENABLE);
}
```