**TÀI LIỆU FRDM KL46Z – ĐỒNG HỒ SỐ**

**Loại: Nội bộ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **THÔNG TIN VỀ TÀI LIỆU** | | | | | |
| Mã số | HDE – 01.01 | | Lần sửa đổi | 3 | |
| Người khởi tạo | QuyetVV | | Người duyệt | BaTM | |
| Thông tin cập nhật | | | | | |
| Ngày | Mục | Tình trạng (Thêm/Sửa/Xóa) | Nội dung | Phiên bản | Người viết |
| 19/04/2017 | 3 | Sửa | **Sửa:**  **-** Hình 25, 27 . | 1.3 | QuyetVV |
| **Thông tin phân công công việc** | | | | | |
| Thành viên | Trách nhiệm  (Toàn bộ, Chính, Ngang bằng, Hỗ trợ) | | Nội dung thực hiện | | |
| QuyetVV | Chính | | * Tìm hiểu lý thuyết. * Mô phỏng. * Viết mã nguồn. | | |
| BaTM | Hỗ trợ | | * Duyệt báo cáo. | | |

# Mục lục

[Mục lục 2](#_Toc480888131)

[1. Tìm hiểu Kit FRDM KL46Z 4](#_Toc480888132)

[1.1 Tổng quan về Kit FRDM KL46Z 4](#_Toc480888133)

[1.2 Mô tả phần cứng FRDM KL46Z 6](#_Toc480888134)

[1.2.1 Cung cấp năng lượng 6](#_Toc480888135)

[1.2.2 Bộ nối tiếp và bộ giải mã (OpenSDA) 7](#_Toc480888136)

[1.3 Vi điều khiển MKL46Z 9](#_Toc480888137)

[1.3.1 Nguồn Clock 10](#_Toc480888138)

[1.3.2 USB interface 10](#_Toc480888139)

[1.3.3 Cổng nối tiếp 10](#_Toc480888140)

[1.3.4 Reset 11](#_Toc480888141)

[1.3.5 Debug 11](#_Toc480888142)

[1.3.6 Phân đoạn LCD 11](#_Toc480888143)

[1.3.7 Thanh trượt cảm ứng điện dung 11](#_Toc480888144)

[1.3.8 Bộ gia tốc kế 3 trục 11](#_Toc480888145)

[1.3.9 Máy Đo Từ Số 3 trục 12](#_Toc480888146)

[1.3.10 LED 12](#_Toc480888147)

[1.3.11 Cảm biến ánh sáng 13](#_Toc480888148)

[1.3.12 Kết nối ngõ vào/ra 13](#_Toc480888149)

[1.3.13 Tương thích Arduino 14](#_Toc480888150)

[2. Giới thiệu giao tiếp I2C và DS1307 14](#_Toc480888151)

[2.1 Giới thiệu I2C 14](#_Toc480888152)

[2.2 Giới thiệu IC thời gian thực DS1307 18](#_Toc480888153)

[3. Lập trình giao tiếp KL46Z và IC DS1307 23](#_Toc480888154)

[3.1 Giao tiếp KL46Z với DS1307 23](#_Toc480888155)

[3.2 Tính lịch âm với DS1307 24](#_Toc480888156)

[3.2.1 Lý thuyết 24](#_Toc480888157)

[3.2.2 Cách tính lịch âm dựa trên cách tra bảng 25](#_Toc480888158)

[3.3 Lập trình 27](#_Toc480888159)

[3.3.1 Tạo thư viện 27](#_Toc480888160)

[3.3.2 Chương trình chính 43](#_Toc480888161)

# Tìm hiểu Kit FRDM KL46Z

## Tổng quan về Kit FRDM KL46Z

***Nguồn:*** [cache.freescale.com/files/microcontrollers/doc/user\_guide/FRDM-KL46Z\_UM.pdf?fpsp=1](http://cache.freescale.com/files/microcontrollers/doc/user_guide/FRDM-KL46Z_UM.pdf?fpsp=1)

Nền tảng phát triển Freescale Freedom là một bộ công cụ phần mềm và phần cứng dùng để đánh giá và phát triển. Đó là ý tưởng nhằm xây dựng nguyên mẫu các ứng dụng dựa trên vi điều khiển một cánh nhanh chóng. Phần cứng FRDM-KL46Z, là một thiết kế đơn giản, tinh vi với bộ vi điều khiển Kinetis L nối tiếp, được xây dựng trên lõi ARM Cortex ™.

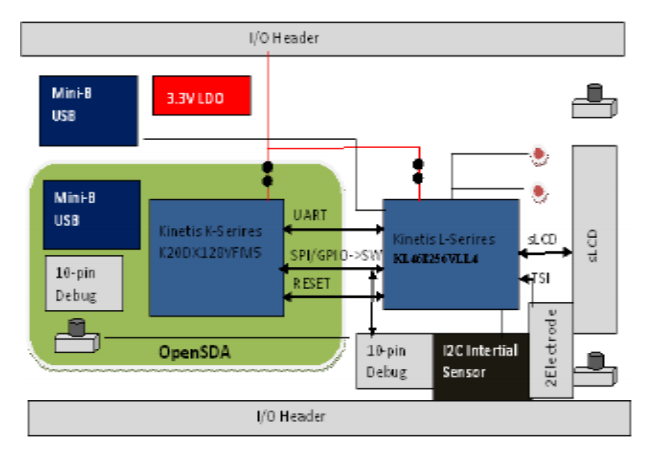
FRDM-KL46Z có thể được sử dụng để đánh giá các thiết bị KL46, KL36, KL26 và KL16 Kinetis L series. Điểm mạnh của Kit là có tần số hoạt động tối đa 48MHz, 256KB flash, 32KB RAM, bộ điều khiển USB tốc độ cao, bộ điều khiển LCD phân đoạn và nhiều thiết bị ngoại vi tương tự và số. Phần cứng FRDM-KL46Z tương thích với bố trí pin Arduino™ R3, cung cấp nhiều tùy chọn bảng mạch mở rộng. Các giao diện trên board bao gồm một màn hình LCD 4 ký tự, một gia tốc số 3 trục, máy đo từ số 3 trục, thanh trượt cảm ứng điện dung, và cảm biến ánh sáng.

FRDM-KL46Z có tính năng Freescale mở với chuẩn nhúng nối tiếp và bộ điều chỉnh gỡ lỗi gọi là OpenSDA. Mạch này cung cấp một số tùy chọn cho truyền thông nối tiếp, lập trình flash và kiểm soát lỗi.

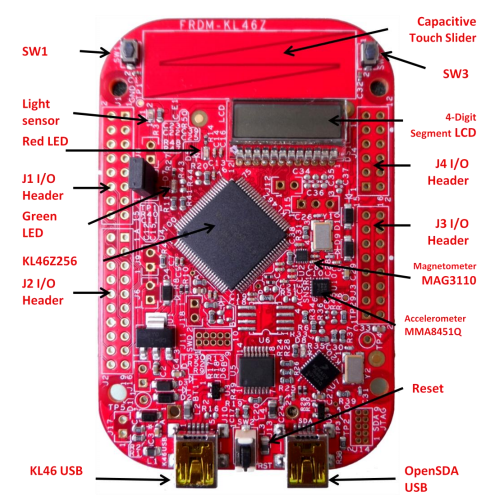
Các tính năng của FRDM-KL46Z bao gồm:

* Vi điều khiển MKL46Z256VLLZ4 (tần số hoạt động 48 MHz, 256KB Flash, RAM 32KB, bộ nguồn năng lượng thấp, đóng gói 100LQFP).
* Giao diện USB kép với đầu nối USB mini-B.
* Cổng Open SDA.
* Module LCD phân đoạn 4 chữ số.
* Thanh trượt cảm ứng điện dung.
* Cảm biến ánh sáng.
* Gia tốc kế MMA8451Q.
* Cảm ứng từ trường MAG3110.
* 2 đèn LED.
* 2 nút nhấn.
* Lựa chọn nguồn linh hoạt có thể sử dụng nguồn 5V từ USB, pin hay nguồn ngoài.
* Dễ dàng truy cập vào MCU I/O thông qua các cổng I/O tương thích với Arduino R3.
* Giao diện gỡ lỗi OpenSDA có thể lập trình với nhiều ứng dụng bao gồm:
* Giao diện lập trình flash của thiết bị lưu trữ khối.
* Giao diện Debug P&E cung cấp khả năng kiểm tra và tương thích với các công cụ IDE.
* Giao diện CMSIS-DAP tiêu chuẩn ARM mới cho giao diện gỡ lỗi khi lập trình nhúng.
* Ứng dụng ghi dữ liệu.
* Tương thích Arduino R3.

Hình 1 cho thấy một sơ đồ khối của FRDM-KL46Z. Các bộ phận chính và vị trí lắp ráp của chúng trên phần cứng được chỉ ra trong Hình 2*.*



Hình : Sơ đồ khối FRDM-KL46Z



Hình : Vị trí các bộ phận trên FRDM-KL46Z

## Mô tả phần cứng FRDM KL46Z

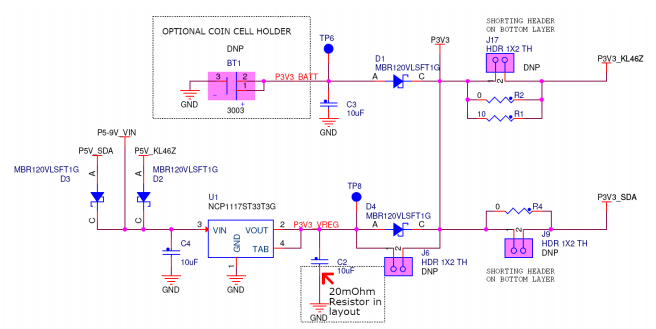
### Cung cấp năng lượng

Có nhiều lựa chọn cấp điện cho FRDM-KL46Z. Nó có thể được cung cấp từ một trong hai cổng kết nối USB, chân VIN trên I/O, một pin trên bo mạch, hoặc một bộ cung cấp 1.71-3.6V bên ngoài. Các thiết bị USB và VIN được điều chỉnh trên bo mạch bằng cách sử dụng một đường dây 3.3V dùng để điều tiết và cung cấp điện chính. Bảng 1 cung cấp các thông tin chi tiết hoạt động và yêu cầu cho nguồn điện.

Bảng : Yêu cầu nguồn của FRDM-KL46Z

|  |  |  |  |
| --- | --- | --- | --- |
| **Nguồn cung cấp** | **Vùng điện áp** | **OpenSDA hoạt động** | **Điều chỉnh trên Board** |
| OpenSDA USB | 5 V | Yes | Yes |
| K20 USB | 5 V | No | Yes |
| VIN Pin | 4.3-9 V | No | Yes |
| 3.3V Pin | 1.71-3.6 V | No | No |
| Coin Cell Battery | 1.71-3.6 V | No | No |

Lưu ý rằng mạch OpenSDA chỉ hoạt động khi một cáp USB được kết nối và cung cấp năng lượng cho OpenSDA USB. Tuy nhiên, các mạch bảo vệ được thiết lập để cho phép nhiều nguồn được cung cấp cùng một lúc.



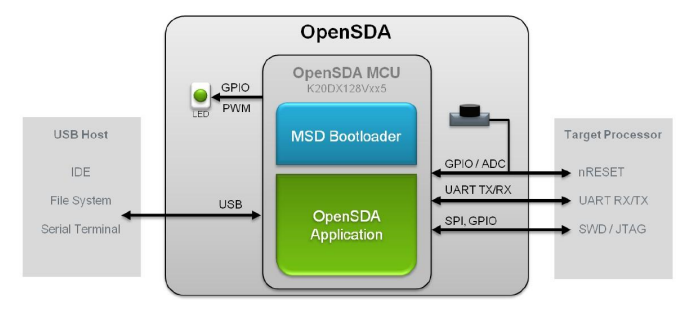
Hình : Sơ đồ nguyên lý của mạch nguồn

Bảng : Nguồn điện FRDM–KL46Z

|  |  |
| --- | --- |
| **Tên nguồn điện** | **Mô tả** |
| P5-9V\_VIN | Nguồn cung cấp từ chân VIN của đầu I/O (chân J9). Một diode Schottky được dùng để bảo vệ ổ đĩa ở khi có dòng ngược. |
| P5V\_SDA | Nguồn cung cấp từ ổ cắm USB OpenSDA. Một Diode Schottky được dùng để bảo vệ ổ đĩa khi có dòng ngược. |
| P5V\_KL46Z | Nguồn cung cấp từ đầu nối USB KL46Z. Một Diode Schottky được dùng để bảo vệ ổ đĩa khi có dòng ngược. |
| P3V3\_VREG | Cung cấp 3.3V quy định. Nguồn điện cung cấp tới P3V3 thông qua một Diode Schottky để bảo vệ ổ đĩa. |
| P3V3\_BATT | Điện áp cung cấp của Pin đồng xu. Nguồn điện cho P3V3  cung cấp qua một Diode Schottky diode bảo vệ ổ đĩa. |
| P3V3 | Đường dây cung cấp chính cho tổ hợp FRDM-KL46Z. Có thể được lấy từ P3V3\_VREG, P3V3\_BATT, hoặc trực tiếp từ các I/O (chân J9 số 8). |
| P3V3\_KL46Z | KL46Z MCU cung cấp. Header J17 cung cấp một công cụ thuận lợi cho việc đo lượng tiêu thụ năng lượng. |
| P3V3\_SDA | Cung cấp mạch OpenSDA. Header J9 cung cấp một công cụ thuận lợi cho phép đo tiêu thụ năng lượng. |
| P5V\_USB | Nominal 5V cung cấp cho I/O (J3 pin 10). Có nguồn gốc từ việc cung cấp P5V\_K20D50M hoặc P5V\_OSDA thông qua Diode Schottky bảo vệ ổ đĩa khi có dòng ngược. |

### Bộ nối tiếp và bộ giải mã (OpenSDA)

OpenSDA là một bộ nối tiếp mở và gỡ lỗi chuẩn. Nó kết nối các giao tiếp nối tiếp và gỡ rối giữa máy chủ USB và bộ xử lý đích được nhúng được thể hiện trong Hình 4. Mạch phần cứng dựa trên bộ vi điều khiển Kinetis Freescale K20 (MCU) với 128 KB flash và bộ điều khiển USB tích hợp. OpenSDA có một bộ nạp khởi động MSD, cung cấp một cơ chế nhanh chóng và dễ dàng để tải các ứng dụng OpenSDA khác nhau như các trình lập trình flash, chạy các giao diện gỡ lỗi kiểm soát, chuyển đổi nối tiếp USB và nhiều ứng dụng khác.

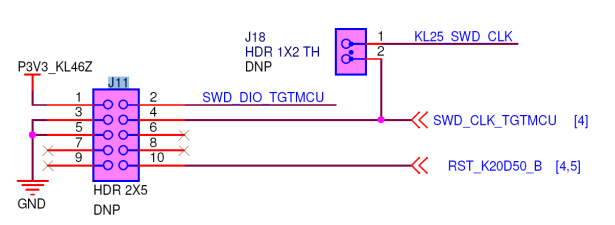


Hình : Biểu đồ khối OpenSDA mức cao

OpenSDA được quản lý bởi một vi điều khiển Kinetis K20 được xây dựng trên lõi ARM Cortex-M4. Mạch OpenSDA bao gồm LED trạng thái (D8) và nút nhấn (SW2). Nút ấn xác nhận tín hiệu Reset đến vi điều khiển KL46. Nó cũng có thể được dùng để đặt mạch OpenSDA vào chế độ Bootloader. Tín hiệu SPI và GPIO cung cấp một giao diện cho cổng gỡ lỗi SWD của K20. Ngoài ra, các kết nối tín hiệu có sẵn để thực hiện một kênh nối tiếp UART. Mạch OpenSDA nhận nguồn khi USB J13 được cắm vào máy chủ USB.

***Giao diện gỡ lỗi***

Các tín hiệu có khả năng SPI và GPIO được sử dụng để kết nối trực tiếp với SWD của KL46. Những tín hiệu này cũng được đưa ra một tiêu chuẩn 10-pin đầu nối Debug Cortex (J11). Nó có thể cô lập MCU KL46 từ mạch OpenSDA và sử dụng J11 để kết nối với một MCU off-board. Để thực hiện việc này, hãy cắt theo dõi phía dưới cùng của PCB ở đó kết nối pin J18 2 đến J11 pin 2. Điều này sẽ ngắt kết nối pin SWD\_CLK đến KL46 để nó sẽ không can thiệp vào các thông tin liên lạc đến một MCU off-board kết nối với J11.



Hình : Trình kết nối gỡ lỗi SWD

Chú ý rằng J11 không được phổ biến. Có thể thêm đầu nối Samtec FTSH-105-02-F-D hoặc kết nối tương thích vào đầu nối thông qua J11. Một cáp kết hợp, chẳng hạn như cáp SamSec FFSD IDC được sử dụng để kết nối từ OpenSDA của FRDM-KL46Z sang đầu nối SWD bên ngoài.

***Cổng nối tiếp ảo***

Một cổng kết nối nối tiếp có sẵn giữa OpenUA MCU và chân PTA1, PTA2 của KL46. Một số ứng dụng OpenSDA mặc định được cung cấp bởi Freescale, bao gồm MSD Flash Programmer và P&E ứng dụng gỡ lỗi, cung cấp một giao diện thiết bị truyền thông USB (CDC) giao tiếp nối tiếp giữa USB chủ và giao diện nối tiếp này trên K20.

## Vi điều khiển MKL46Z

Vi điều khiển FRDM-KL46Z là KL462Z256VLL4, đóng gói 100 LQFP. Các tính năng của KL46Z MCU bao gồm:

* Lõi ARM Cortex-M0+ 32-bit:
* Tần số hoạt động 48 MHz.
* Cổng truy cập I/O có chu kỳ tốc độ đơn.
* Memories: 256 KB flash.
* SRAM 32 KB.
* Tích hợp hệ thống:
* Quản lý năng lượng và điều khiển chế độ.
* Bộ phận báo động rò rỉ thấp.
* Thao tác xử lý bit cho các thao tác ngoại vi đọc, sửa đổi, ghi.
* Bộ điều khiển truy cập bộ nhớ trực tiếp (DMA).
* Máy tính hoạt động đúng (COP) Watchdog Timer.
* Clock:
* Module tạo Clock với FLL và PLL cho hệ thống và tạo Clock cho CPU.
* Clock tham chiếu nội với bộ 4 MHz và 32 kHz.
* Dao động hệ thống hỗ trợ tinh thể bên ngoài hoặc cộng hưởng.
* Dao động RC 1KHz công suất thấp cho cơ quan giám sát RTC và COP.
* Thiết bị ngoại vi tương tự:
* Hỗ trợ 16-bit SAR ADC w/DMA.
* Hỗ trợ DAC 12 bit với DMA.
* Bộ so sánh tốc độ cao.
* Thiết bị ngoại vi truyền thông:
* Một giao diện âm thanh Interchip tích hợp (I2S) và (SAI).
* Hai giao diện nối tiếp ngoại vi 8 bit (SPI).
* Bộ điều khiển kép USB với bộ thu phát FS/LS tích hợp sẵn.
* Bộ điều chỉnh điện áp USB.
* 2 Module I2C.
* 1 UART công suất thấp và 2 Module UART tiêu chuẩn.
* Timers:
* 1 Module Timer/PWM 6 kênh.
* 2 Module Timer/PWM 2 kênh.
* Khoảng thời gian gián đoạn 2 kênh (PIT).
* Đọc giá trị thời gian thực (RTC).
* Timer công suất thấp (LPT).
* Hệ thống Timer đánh dấu.
* Giao diện người - máy (HMI).
* Bộ điều khiển phân đoạn LCD. Phân đoạn tối đa là 8x47 hoặc 4x51.
* Bộ điều khiển đầu vào đầu ra chung.
* Cảm ứng điện dung đầu vào với giao diện mô-đun phần cứng.

### Nguồn Clock

Bộ vi điều khiển Kinetis KL46 có khả năng tương thích với 3 dải tần số đầu vào hoặc tần số cộng hưởng: 32-40 KHz (tần số thấp), 3-8 MHz (chế độ cao, tần số thấp) và 8-32 MHz (Chế độ tần số cao , phạm vi cao).

### USB interface

Tính năng bộ điều khiển Kinetis KL46 là điều khiển USB kép với hai cổng phát tốc độ đầy đủ và tốc độ thấp trên chip. Giao diện USB trên FRDM-KL46Z được cấu hình như một thiết bị USB tốc độ cao. VREGIN phải được cung cấp năng lượng cho mạch điện nội bộ của USB (bằng Jumper J7).

### Cổng nối tiếp

Tín hiệu giao diện của các cổng nối tiếp chính là PTA1 UART0\_RX và PTA2 UART0\_TX. Những tín hiệu này được kết nối OpenSDA.

### Reset

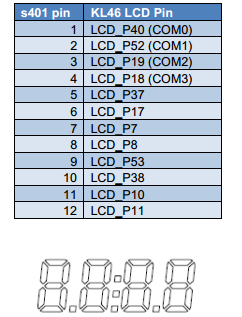
Tín hiệu RESET trên K20 được nối bên ngoài với một nút bấm SW2 và với mạch OpenSDA. Nút reset cũng có thể được sử dụng để bắt OpenSDA vào chế độ bootloader.

### Debug

Giao diện gỡ lỗi duy nhất trên tất cả các thiết bị Kinetis L Series là một cổng Debug Serial Wire (SWD). Bộ điều khiển chính của giao diện này trên FRDM-KL46Z là mạch OpenSDA trên bo mạch. Tuy nhiên, đầu cắm Debug 10-pin ( (J11) cho phép truy cập vào các tín hiệu SWD. Bộ ghép nối SamTec FTSH-105-02-F-D hoặc tương thích có thể được thêm vào đầu nối gỡ lỗi J11 qua lỗ trống để cho phép nối cáp gỡ lỗi bên ngoài.

### Phân đoạn LCD

FRDM-KL46Z đang sử dụng màn hình hiển thị 4 chữ số (LUMEX LCD-S401M16KR) 4x8. Hình 6 hiển thị kết nối từ KL46 đến màn hình s401.



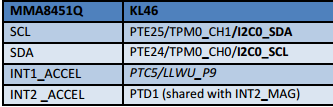
Hình : Kết nối và cấu trúc của LCD

### Thanh trượt cảm ứng điện dung

Hai tín hiệu đầu vào cảm ứng cảm ứng (TSI), TSI0\_CH9 / PTB16 và TSI0\_CH10 / PTB17 được kết nối với điện dung. Điện cực được định cấu hình như một thanh trượt cảm ứng. Phần mềm Touch Sense của Freescale (TSS) cung cấp một thư viện phần mềm cho thực hiện thanh trượt cảm ứng điện dung.

### Bộ gia tốc kế 3 trục

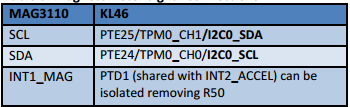
Một máy đo gia tốc công suất nhỏ củaFreescale MMA8451Q được nối qua bus I2C và GPIO. Tín hiệu như thể hiện trong Hình 7 dưới đây. Theo mặc định, địa chỉ I2C là 0x1D (SA0 kéo lên mức cao).



Hình : Kết nối tín hiệu bộ gia tốc kế

### Máy Đo Từ Số 3 trục

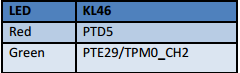
Một Freescale MAG3110 Ba-Axis, giao tiếp thông qua một bus I2C, và một GPIO tín hiệu như thể hiện trong Hình 8 dưới đây.



Hình : Giao tiếp tín hiệu máy đo từ

### LED

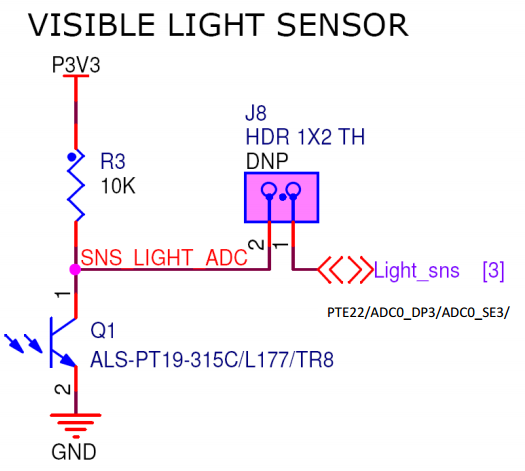
Có hai LED, trong đó LED Xanh có khả năng PWM, các kết nối tín hiệu được hiển thị trong Hình 9.



Hình : Giao tiếp ín hiệu LED

### Cảm biến ánh sáng

FRDM-KL46Z có cảm biến ánh sáng đượckết nối với ADC0\_SE3.



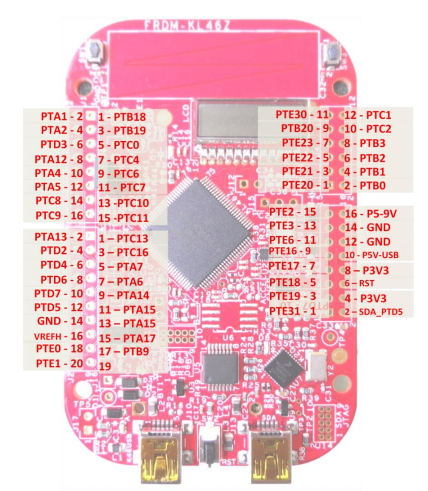
Hình : Sơ đồ nguyên lí cảm biến ánh sáng

### Kết nối ngõ vào/ra

Các vi điều khiển MKL46Z256VLL4 được đóng gói 100-pin LQFP. Một số chân được sử dụng trong mạch on-board, nhưng nhiều kết nối trực tiếp với 1 trong 4 header I/O.

Các chân trên vi điều khiển KL46 được đặt tên cho mục đích chung của họ đầu vào/ra của các pin chức năng. Ví dụ, đầu tiên trên cổng A được gọi là PTA1. Tên pin kết nối I/O được đặt cùng tên với pin KL46 được kết nối với nó.

**Chú ý**: tất cả các pin out dữ liệu có sẵn ở định dạng bảng tính trong FRDM-KL46Z.



Hình : FRDM-KL46 Pin Out

### Tương thích Arduino

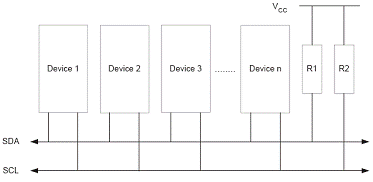
Các tiêu đề I/O trên FRDM-KL46Z được bố trí để cho phép tương thích với các ngoại vi kết nối với các bảng điều khiển tương thích Arduino ™ và Arduino. Các hàng bên ngoài của chân (các chân được đánh số) trên các header có cùng khoảng cách cơ học và vị trí như header I/O trên tiêu chuẩn Arduino Revision 3 (R3).

# Giới thiệu giao tiếp I2C và DS1307

## Giới thiệu I2C

***Nguồn:*** <http://www.hocavr.com/index.php/lectures/i2c>

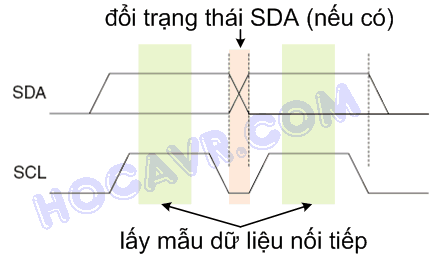
I2C (Inter Integrated Circuit) là một chuẩn truyền thông nối tiếp đa chip chủ (multi - master serial computer bus). Khái niệm “multi – master” được hiểu là trên cùng một bus có thể có nhiều hơn một thiết bị làm master, đồng thời một slave có thể trở thành một master nếu nó có khả năng. I2C thực hiện trên 2 đường SDA (Serial Data) và SCL (Serial Clock) trong đó SDA là đường truyền/nhận dữ liệu và SCL là đường xung nhịp. Căn cứ theo chuẩn I2C, các đường SDA và SCL trên các thiết bị có cấu hình “cực góp mở” (open – drain hoặc open – collector), nghĩa là cần có các “điện trở kéo lên” (pull – up resistor) cho các đường này. Ở trạng thái nghỉ (Idle), 2 chân SDA và SCL ở mức cao.



Hình : Mạng I2C với nhiều thiết bị và 2 điện trở kéo lên cho SDA, SCL

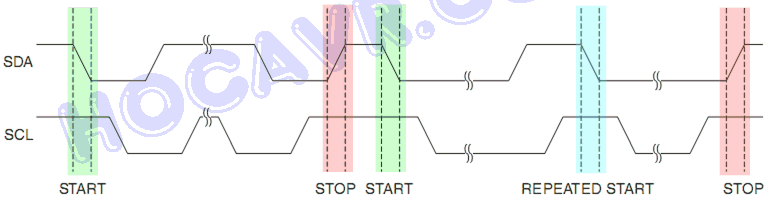
Một số khái niệm và đặc điểm của I2C:

* Master: chip khởi động quá trình truyền nhận, phát đi địa chỉ của thiết bị cần giao tiếp và tạo xung giữ nhịp trên đường SCL.
* Slave: chip có một địa chỉ cố định, được gọi bởi Master và phục vụ yêu cầu từ Master.
* SDA: đường dữ liệu nối tiếp, tất cả các thông tin về địa chỉ hay dữ liệu đều được truyền trên đường này theo thứ tự từng bit một. Bit có trọng số lớn nhất (MSB) được truyền trước.
* SCL: đường giữ nhịp nối tiếp. I2C là chuẩn truyền thông nối tiếp đồng bộ, cần có 1 đường tạo xung giữ nhịp cho quá trình truyền/nhận, cứ mỗi xung trên đường giữ nhịp SCL, một bit dữ liệu trên đường SDA sẽ được lấy mẫu (sample). Dữ liệu nối tiếp trên đường SDA được lấy mẫu khi đường SCL ở mức cao trong một chu kỳ giữ nhịp, vì thế đường SDA không được đổi trạng thái khi SCL ở mức cao (trừ START và STOP condition). Chân SDA có thể thay đổi được trạng thái khi SCL ở mức thấp.



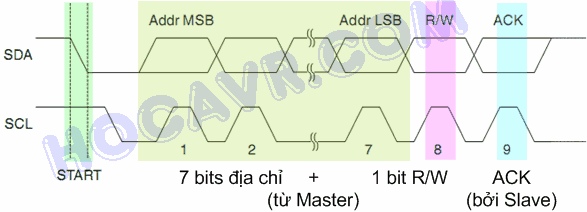
Hình : Lấy mẫu dữ liệu

* START Condition: điều kiện bắt đầu. Từ trạng thái nghỉ, khi cả SDA và SCL ở mức cao, nếu master muốn thực hiện một “cuộc gọi”, master sẽ kéo chân SDA xuống thấp trong khi SCL vẫn ở mức cao. Trạng thái này gọi là START Condition (S).
* STOP Condition: điều kiện kết thúc. Sau khi thực hiện truyền/nhận dữ liệu, nếu master muốn kết thúc quá trình nó sẽ tạo ra một stop condition. Stop condition được master thức hiện bằng cách kéo chân SDA lên mức cao trong khi đường SCL đang ở mức cao. Stop condition chỉ được tạo ra sau khi địa chỉ hoặc dữ liệu đã được truyền nhận.
* REPEAT START: bắt đầu lặp lại. Khoảng giữa Start và Stop condition là khoảng bận của đường, các master khác không được tác động vào đường truyền trong khoảng này. Trong trường hợp này sau khi kết thúc truyền/nhận mà master không gửi stop condition mà lại gửi thêm 1 start condition gọi là repeat start. Khả năng này thường được dùng khi master muốn lấy dữ liệu liên tiếp từ các slaves.



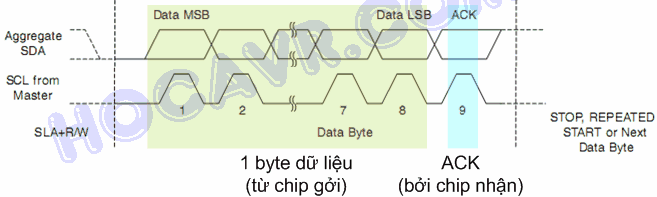
Hình : Mô tả master tạo ra start, stop, repeat start

* Address Packet Format: Định dạng gói địa chỉ. Trên I2C, tất cả các chip đều có thể là master hay slave. Mỗi thiết bị có một địa chỉ cố định gọi là Device address. Khi một master muốn giao tiếp với một slave nào đó, trước hết nó tạo ra một start condition và tiếp theo gửi địa chỉ device address của slave cần giao tiếp trên đường truyền, vì thế có khái niệm gói địa chỉ (Address Packet). Gói địa chỉ trong I2C có định dạng 9 bit trong đó 7 bit đầu (gọi là SLA, được gửi liền sau start condition) chứa địa chỉ slave, một bit READ/WRITE và một bit ACK (**Ackknowledge**). Do đó địa chỉ có độ dài 7 bit nên về mặt lý thuyết trên 1 mạng I2C có thể có tối đa  thiết bị có địa chỉ riêng biệt. Tuy nhiên, có một số địa chỉ không được như sử dụng như các địa chỉ có định dạng 1111xxx (tức các địa chỉ lớn hơn hoặc bằng 120 không được dùng). Riêng địa chỉ 0 được dùng cho “cuộc gọi chung” (General call). Bit READ/WRITE (R/W) được truyền sau 7 bit địa chỉ là bit báo cho slave biết master muốn “đọc” hay “ghi” vào slave. Nếu R/W = 0 thì quá trình “ghi” dữ liệu vào slave được yêu cầu, R/W = 1 thì master muốn “đọc” dữ liệu từ slave. 8-bit trên (SLA + R/W) được master phát ra sau khi phát start condition, nếu một slave trên mạng nhận ra rằng địa chỉ mà master yêu cầu trùng khớp với Device address của mình, nó sẽ “đáp trả” lại master bằng cách phát ra 1 tín hiệu “xác nhận” ACK bằng cách kéo chân SDA xuống thấp trong xung thứ 9. Ngược lại, nếu không có slave đáp ứng lại, chân SDA vẫn ở mức cáo trong xung giữ nhịp thứ 9 thì gọi là tín hiệu “không xác nhận” - NOT ACK, lúc này master cần có những ứng xử phù hợp tùy theo mỗi trường hợp cụ thể. Như vậy trong 9-bit của gói địa chỉ có 8-bit được gửi từ master 1-bit còn lại từ slave. Ví dụ, master muốn yêu cầu “đọc” dữ liệu từ slave có địa chỉ 43, nó cần phát đi 1byte như sau : (43 >> 1) + 1, trong đó (43 >> 1) là dịch số 43 về phía trái 1 vị trí vì 7-bit địa chỉ nằm ở các vị trí cao trong gói địa chỉ, sau đó cộng giá trị này với 1 tức là quá trình đọc được yêu cầu.



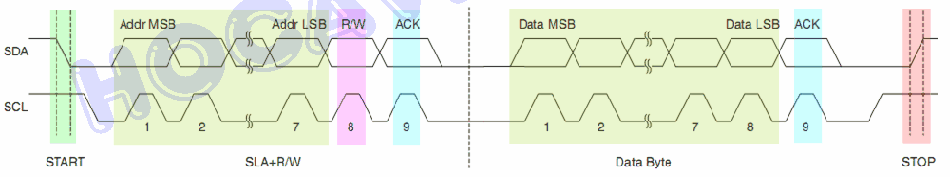
Hình : Cấu trúc Address Packet

* General call: cuộc gọi chung. Khi master phát đi gọi địa chỉ có dạng 0 (thực chất là 0 + W) tức nó muốn thực hiện một cuộc gọi chung đến tất cả slave. Nếu các slave được cài đặt cho phép cuộc gọi chung, chúng sẽ đáp lại master bằng ACK. Cuộc gọi chung thường xảy ra khi master muốn gửi dữ liệu đến các slave. Lưu ý master không thể nhận dữ liệu từ các slave cùng một thời điểm.
* Data Packet Format: Định dạng gói dữ liệu. Sau khi địa chỉ đã được phát đi, slave đã đáp lại master bằng ACK thì quá trình truyền/nhận dữ liệu sẽ diễn ra giữa cặp master/slave này. Tùy vào bit R/W trong gói địa chỉ, dữ liệu có thể được truyền theo hướng master đến slave hay ngược lại. Gói dữ liệu luôn bao gồm 9 bit trong đó 8-bit đầu là dữ liệu và 1-bit cuối là ACK. 8-bit dữ liệu do thiết bị phát gửi, và bit ACK do thiết bị nhận tạo ra. Ví dụ master thực hiện quá trình gửi dữ liệu đến slave, nó sẽ phát ra 8-bit dữ liệu, slave nhận và phát lại ACK (kéo SDA xuống 0 ở xung thứ 9), sau đó master sẽ quyết định gửi tiếp byte dữ liệu khác hay không. Nếu slave phát tín hiệu NOT ACK sau khi nhận dữ liệu thì master sẽ kết thúc quá trình gửi bằng cách phát đi stop condition.



Hình : Định dạng gói dữ liệu

* Phối hợp gói địa chỉ và dữ liệu: một quá trình truyền/nhận I2C thường được bắt đầu từ master, master phát đi một start condition sau đó gửi gói địa chỉ SLA + R/W trên đường truyền. Tiếp theo nếu có một slave đáp ứng lại, dữ liệu có thể truyền/nhận liên tiếp trên đường truyền (1 hoặc nhiều byte liên tiếp). Hình 17 mô tả khung truyền.



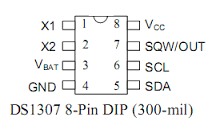
Hình : Khung truyền I2C

* Multi – Master Bus: Đường truyền đa chip chủ. Như đã trình bày ở trên, I2C là chuẩn truyền thông đa chip chủ, tức là tại một thời điểm có thể có nhiều hơn 1 chip làm master nếu các chip này phát ra start condition cùng lúc. Nếu các master có cùng yêu cầu và thao tác đối với slave thì chúng có thể “cùng tồn tại” và quá trình truyền/nhận có thể thành công.

## Giới thiệu IC thời gian thực DS1307

***Nguồn:*** <http://www.hocavr.com/index.php/vi/app/ds1307>

DS1307 là chip đồng hồ thời gian thực (RTC – Read Time Clock), khái niệm thời gian thức ở đây được dùng với ý nghĩa thời gian tuyệt đối mà con người đang sử dụng. Chip này có 7 thanh ghi 8-bit chứa thời gian là: giây, phút, giờ, thứ, ngày, tháng, năm. Ngoài ra DS1307 còn có 1 thanh ghi điều khiển ngõ ra phụ và 56 thanh ghi trống có thể dùng như RAM. DS1307 được đọc và ghi thông qua giao tiếp I2C.

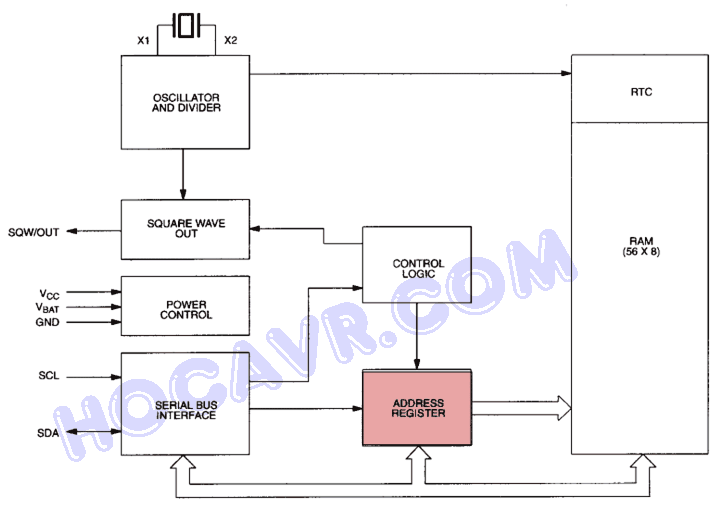


Hình : Sơ đồ chân DS1307

Các chân DS1307 được mô tả như sau:

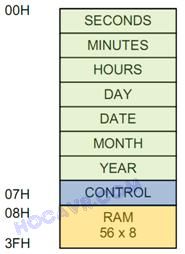
* Chân X1 và X2 (1, 2): 2 ngõ kết nối với thạch anh 32.768KHz làm nguồn tạo dao động cho chip.
* VBAT (3): cực dương của một nguồn pin 3V nuôi chip.
* GDN (4): chân đất chung cho cả pin 3V và Vcc.
* Vcc(8): nguồn cho giao diện I2C, thường là 5V và dùng chung với vi điều khiển. Chú ý là nếu Vcc không được cấp nguồn nhưng VBAT được cấp thì DS1307 vẫn đang hoạt động (nhưng không ghi và đọc được).
* SQW/OUT (7): ngõ ra phụ tạo xung vuông (Square Wave/Output Driver), tần số của xung được tạo có thể được lập trình. Như vậy chân này hầu như không liên quan đến chức năng của DS1307 là đồng hồ thời gian thực, thường chân này bị bỏ trống khi nối mạch.
* SCL (6): chân giữ xung nhịp.
* SDA (5): chân truyền dữ liệu.

Cấu tạo bên trong bao gồm: mạch nguồn, mạch dao động, mạch điều khiển logic, mạch giao tiếp I2C, con trỏ địa chỉ và các thanh ghi (RAM).



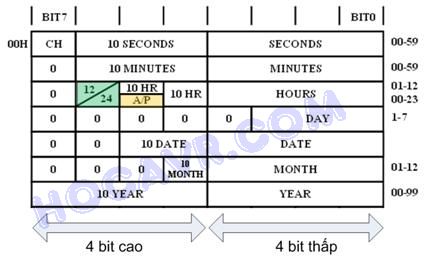
Hình : Cấu trúc của DS1307

Bộ nhớ DS1307 có tất cả 64 thanh ghi 8-bit được đánh địa chỉ từ 0 đến 63 (0x00 đến 0x3F). Chỉ có 8 thanh ghi đầu cho chức năng “đồng hồ” (RTC), còn lại 56 thanh ghi bỏ trống có thể được chứa như biến tạm RAM nếu muốn. 7 thanh ghi đầu tiên chứa thông tin về thời gian của đồng hồ bao gồm: giây (SENCONDS), phút (MINUETS), giờ (HOURS), thứ (DAY), ngày (DATE), tháng (MONTH), và năm (YEAR). Việc ghi giá trị vào 7 thanh ghi này tương đương với việc “cài đặt” thời gian khởi động cho RTC. Việc đọc giá từ 7 thanh ghi là đọc thời gian thực mà chip tạo ra. Thanh ghi thứ 8 (CONTROL) là thanh ghi điều khiển xung ngõ ra SQW/OUT. Tuy nhiên do không dùng SQW/OUT nên có thể bỏ qua thanh ghi thứ 8.



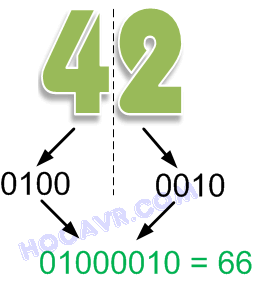
Hình : Tổ chức bộ nhớ của DS1307

Khảo sát 7 thanh ghi dùng cho chức năng “đồng hồ”.



Hình : Tổ chức các thanh ghi thời gian

Các giá trị thời gian lưu trong thanh ghi theo dạng mã nhị phân (BCD). Ví dụ muốn cài đặt cho thanh ghi MINUTES giá trị 42. Thông thường chúng ta chỉ cần gán MINUTES = 42 hay MINUTES = 0x2A. Tuy nhiên, vì các thanh ghi này chứa giá trị BCD nên số 42 sẽ được tách thành 2 chữ số 4 và 2. Mỗi chữ số sau đó được đổi sang mã nhị phân 4-bit: 4 = 0100, 2 = 0010.



Hình : Số BCD

Ghép mã nhị phân của 2 chữ số lại được một số BCD 8-bit. Với trường hợp này là 01000010 = 66. Như vậy, để đặt số 42 cho DS1307 chúng ta cần ghi vào thanh ghi MINUTES giá trị 66.

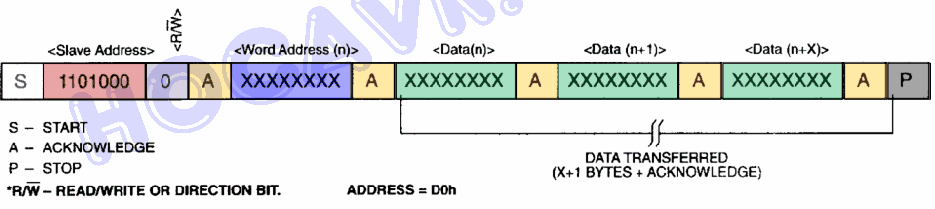
* Thanh ghi giây (SECONDS): thanh ghi đầu tiên trong bộ nhớ của DS1307, địa chỉ là 0x00. 4-bit thấp chứa mã BCD 4-bit của chữ số hàng đơn vị của giá trị giây. Do giá trị cao nhất của chữ số hàng chục là 5 (không có giây 60) nên chỉ cần 3 bit để mã hóa (5 = 101). Bit cao nhất, bit 7 là 1 điều khiển có tên CH (Clock Halt - treo đồng hồ), nếu bit này được set bằng 1, bộ dao động trong chip bị vô hiệu hóa, đồng hồ không hoạt động. Vì vậy phải reset bit này xuống 0 từ đầu.
* Thanh ghi phút (MINUTES): có địa chỉ 0x01, chứa giá trị phút của đồng hồ. Tương tự như thanh ghi giây, chỉ có 7-bit của thanh ghi này được dùng lưu mã BCD, bit 7 luôn bằng 0.
* Thanh ghi giờ (HOURS): có địa chỉ 0x02. 4-bit thấp của thanh ghi này được dùng cho chữ số hàng đơn vị của giờ. Do DS1307 hỗ trợ 2 loại hệ thống hiển thị giờ là 12h và 24h, bit 6 xác lập hệ thống giờ. Nếu bit 6 = 0 thì hệ thống 24h được chọn, khi đó 2 bit cao 4 và 5 dùng mã hóa chữ số hàng chục của giá trị giờ. Do giá trị lớn nhất của trong trường hợp này là 2 nên chỉ cần 2 bit 4 và bit 5 để mã hóa. Nếu bit 6 = 1, thì hệ thống 12h được chọn, với trường hợp này chỉ có bit 4 dùng mã hóa chữ số hàng chục của giờ còn bit 5 chỉ buổi trong ngày AM hoặc PM. Bit 5 = 0 là AM và bit 5 = 1 là PM. Bit 7 luôn bằng 0.
* Thanh ghi thứ (DAY): địa chỉ 0x03. Thanh ghi này chỉ chứ giá trị từ 1 đến 7 nên chỉ có 3 bit thấp trong thanh ghi này có ý nghĩa.
* Thanh ghi ngày (DATE): địa chỉ 0x04. Dùng 5-bit để mã hóa (0 - 31). Giá trị của ngày từ 1 đến 31.
* Thanh ghi tháng (MONTH): địa chỉ 0x05. Dùng 4-bit để mã hóa giá (0 - 16). Giá trị của tháng từ 1 đến 12.
* Thanh ghi năm (YEAR): địa chỉ 0x06. Dùng 7-bit để mã hóa. Giá trị của năm từ 0 đến 99. Chú ý DS1307 chỉ dùng cho 100 năm, nên giá trị năm chỉ có 2 chữ số, phần đầu của năm do người dùng thêm vào (ví dụ 20xx).

Ngoài các thanh ghi trong bộ nhớ, DS1307 có 1 thanh ghi địa chỉ. Giá trị của thanh ghi này là địa chỉ của thanh ghi trong bộ nhớ mà người dùng muốn truy cập. Giá trị của thanh địa chỉ được thiết lập trong lệnh Write.

# Lập trình giao tiếp KL46Z và IC DS1307

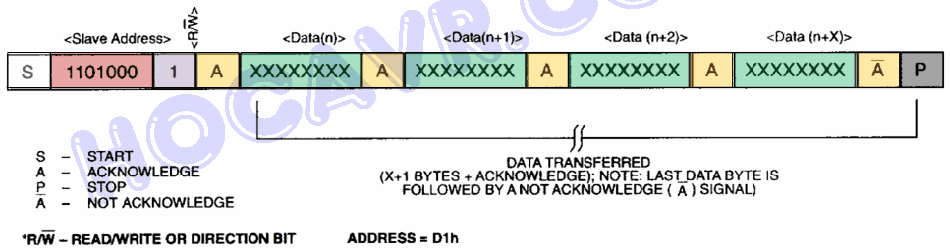
## Giao tiếp KL46Z với DS1307

DS1307 và KL46Z giao tiếp với nhau qua chuẩn I2C. DS1307 đóng vai trò là slave còn KL46Z là master. Để thực hiện cuộc gọi ở chế độ master, KL46Z sẽ gửi start condition, tiếp theo là 7-bit địa chỉ slave (SLA) + 1-bit R/W, tiếp theo là quá trình đọc ghi dữ liệu giữa master và slave bằng các byte 8-bit (có thể 1 hoặc nhiều byte), cứ sau mỗi byte sẽ có 1-bit ACK hoặc NOT ACK. Cuộc gọi kết thúc với việc master gửi stop condition. DS1307 có 2 chế độ giao tiếp là Data Write và Data Read. Chế độ Data Write được dùng khi xác lập giá trị ban đầu cho các thanh ghi thời gian hoặc dùng để canh chỉnh thời gian. Khi đó KL46Z sẽ gửi dữ liệu đến DS1307. Ở chế độ Data Read, KL46Z đọc thời gian từ đồng hồ DS1307. Khi đó KL46Z sẽ nhận dữ liệu và DS1307 truyền dữ liệu.



Hình : Chế độ Data Write

Chip DS1307 có một địa chỉ cố định SLA là 1101000. Do SLA của DS1307 cố định nên trong 1 mạng I2C không thể có cùng lúc 2 chip này. Sau khi gửi đi start condition sẽ là 7-bit địa chỉ SLA. Do chế độ này là Data Write lên bit W (0) sẽ được gửi kèm sau SLA. Bit ACK (A) được DS1307 trả về cho Master sau mỗi quá trình giao tiếp. Tiếp theo sau địa chỉ SLA sẽ là 1 byte chứa địa chỉ của thanh ghi cần truy cập (tạm gọi là Addr\_Reg). Cần phân biệt địa chỉ thanh ghi cần truy cập và địa chỉ SLA. Địa chỉ của thanh ghi cần truy cập sẽ được lưu trong thanh ghi địa chỉ, vì vậy byte dữ liệu đầu tiên sẽ được chứa trong thanh ghi địa chỉ của DS1307. Sau byte thanh ghi địa chỉ là một dãy các byte dữ liệu được ghi vào bộ nhớ của DS1307. Byte dữ liệu đầu tiên sẽ được ghi vào thanh ghi có địa chỉ được chỉ định bởi Addr\_Reg, sau khi ghi 1 byte, Addr\_Reg được tự động tăng nên các byte tiếp theo sẽ được ghi liên tiếp vào các thanh ghi kế sau. Số lượng byte dữ liệu cần ghi do master quyết định và không được vượt quá dung lượng bộ nhớ của DS1307.



Hình : Chế độ Data Read

Ví dụ sau khi gửi SLA + W, master gửi 8 bytes gồm 1 byte đầu 0x00 và 7 bytes khác thì con trỏ địa chỉ sẽ trỏ đến thanh ghi đầu tiên (0x00 – thanh ghi SECONDS) và ghi liên tiếp 7 byte vào 7 thanh ghi thời gian của DS1307. Quá trình ghi kết thúc khi master gửi stop condition.

Chú ý, nếu sau khi gửi byte Addr\_Reg, master không gửi các byte dữ liệu mà gửi stop condition thì không có thanh ghi nào được ghi. Trường hợp này dùng để set địa chỉ Addr\_Reg phục vụ cho quá trình đọc.

Trong chế độ Data Read, bit R (1) được gửi đi kèm sau 7-bit SLA. Sau đó là liên tiếp các byte dữ liệu được truyền từ DS1307 đến KL46Z. Ở chế độ này, không có byte địa chỉ thanh ghi dữ liệu được gửi đến. Tất cả các bytes theo sau SlA + R đều là dữ liệu đọc từ bộ nhớ của DS1307. Thứ tự đọc thanh ghi được chỉ định bởi con trỏ địa chỉ, giá trị này được lưu lại trong các lần thao tác trước đó. Như vậy, muốn đọc dữ liệu từ một địa chỉ nào đó, chúng ta cần thực hiện quá trình ghi giá trị cho con trỏ địa chỉ trước. Để ghi giá trị vào con trỏ địa chỉ, cần gọi chương trình Data Write với chỉ 1 byte được ghi sau SLA + W ở trên.

## Tính lịch âm với DS1307

### Lý thuyết

***Nguồn:*** <https://www.informatik.uni-leipzig.de/~duc/amlich/calrules.html>

Âm lịch Việt Nam là một loại lịch thiên văn. Nó được tính toán dựa trên sự chuyển động của mặt trời, trái đất và mặt trăng. Sóc là thời điểm hội diện, đó là khi trái đất, mặt trăng và mặt trời nằm trên một đường thẳng và mặt trăng nằm giữa trái đất và mặt trời (như thế góc giữa mặt trăng và mặt trời bằng **0 độ**). Gọi là "hội diện" vì mặt trăng và mặt trời ở cùng một hướng đối với trái đất. Chu kỳ của điểm Sóc là khoảng 29,5 ngày. Ngày chứa điểm Sóc được gọi là ngày Sóc và đó là ngày bắt đầu tháng âm lịch. Trung khí là các điểm chia đường hoàng đạo thành 12 phần bằng nhau. Trong đó, bốn Trung khí giữa bốn mùa là đặc biệt nhất: Xuân phân (khoảng 20/3), Hạ chí (khoảng 22/6), Thu phân (khoảng 23/9) và Đông chí (khoảng 22/12

Để tính ngày tháng âm lịch có thể sử dụng 2 cách sau:

**Cách 1**: Dựa trên tính toán ngày Sóc.

Nguyên tắc:

* Ngày đầu tiên của tháng âm lịch là ngày chứa điểm Sóc.
* Một năm bình thường có 12 tháng âm lịch, một năm nhuận có 13 tháng âm lịch. Do trái đất quay quanh mặt trời một vòng hết 365 ngày 5 giờ 48 phút 46 giây. Và lịch dường sử dụng hiện nay coi một năm có 365 ngày, như vậy 1 năm dư ra 5 giờ 48 phút 46 giây. Cứ 4 năm tích lũy thời gian dư thành 1 ngày. Ngày đó theo quy ước là vào thánh 2. Như vậy tháng 2 của năm nhuận có thêm một ngày là ngày 29/2.
* Đông chí luôn rơi vào tháng 11 âm lịch. Tiết đông chí theo lịch của Trung Quốc cổ đại, là tiết khí khởi đầu bằng điểm giữa mùa đông, nó là 1 trong 24 tiết khí trong nông lịch.
* Việc tính toán dựa trên kinh tuyến 105° đông.

Bởi vì dựa trên cả mặt trời và mặt trăng nên lịch Việt Nam không phải là thuần âm lịch mà là âm-dương-lịch. Theo các nguyên tắc trên, để tính ngày tháng âm lịch cho một năm bất kỳ trước hết cần xác định những ngày nào trong năm chứa các thời điểm Sóc (New moon). Một khi đã tính được ngày Sóc, biết được ngày bắt đầu và kết thúc của một tháng âm lịch: ngày mùng một của tháng âm lịch là ngày chứa điểm Sóc. Sau khi đã biết ngày bắt đầu/kết thúc các tháng âm lịch, tính xem các Trung khí (Major solar term) rơi vào tháng nào để từ đó xác định tên các tháng và tìm tháng nhuận.

Đông chí luôn rơi vào tháng 11 của năm âm lịch. Bởi vậy cần tính 2 điểm Sóc: Sóc A ngay trước ngày Đông chí thứ nhất và Sóc B ngay trước ngày Đông chí thứ hai. Nếu khoảng cách giữa A và B là dưới 365 ngày thì năm âm lịch có 12 tháng, và những tháng đó có tên là: tháng 11, tháng 12, tháng 1, tháng 2, …, tháng 10. Ngược lại, nếu khoảng cách giữa hai sóc A và B là trên 365 ngày thì năm âm lịch này là năm nhuận, và cần tìm xem đâu là tháng nhuận. Để làm việc này, xem xét tất cả các tháng giữa A và B, tháng đầu tiên không chứa Trung khí sau ngày Đông chí thứ nhất là tháng nhuận. Tháng đó sẽ được mang tên của tháng trước nó kèm chữ "nhuận".

**Cách 2**: Tra bảng

Khai báo bảng chứa các ngày dương lịch đầu tháng âm lịch, ngày âm lịch của đầu tháng dương lịch, tháng âm lịch đầu tháng dương lịch, tháng âm lịch đầu tháng âm lịch tương ứng với các năm.

### Cách tính lịch âm dựa trên cách tra bảng

***Nguồn:*** [*Tính âm lịch*](http://www.dientuvietnam.net/forums/forum/vi-%C4%91i%E1%BB%81u-khi%E1%BB%83n-mcu-b%E1%BB%99-%C4%91i%E1%BB%81u-khi%E1%BB%83n-t%C3%ADn-hi%E1%BB%87u-s%E1%BB%91-dsc/vi-%C4%91i%E1%BB%81u-khi%E1%BB%83n-h%E1%BB%8D-8051/73127-thu%E1%BA%ADt-to%C3%A1n-t%C3%ADnh-%C3%82m-l%E1%BB%8Bch-t%E1%BB%AB-d%C6%B0%C6%A1ng-l%E1%BB%8Bch-d%C3%B9ng-ph%C6%B0%C6%A1ng-ph%C3%A1p-tra-b%E1%BA%A3ng)

Tính ngày âm lịch:

Nếu ngày Dương lịch tương ứng với ngày đầu của tháng Âm lịch bằng ngày hiện tại thì ngày Âm lịch lịch bằng 1. Hoặc ngày Dương lịch tương ứng với ngày đầu tháng âm lịch trừ ngày hiện tại:

* Nếu không có nhớ (C = 0) thì ngày Âm lịch bằng (ngày dương lịch hiện tại) cộng ngày Âm lịch của đầu tháng Dương lịch rồi trừ 1.
* Nếu có nhớ (C = 0) thì ngày Âm lịch hiện tại bằng ngày Dương lịch hiện tại trừ ngày Dương lịch đầu tháng Âm lịch rồi cộng 1.

Ví dụ: Tính ngày âm lịch của ngày 13/03/2017

* Ngày dương lịch hiện tại: 13.
* Ngày dương lịch tương ứng với đầu tháng âm lich: 26. (Ngày 26/2 dương lịch là ngày 1/2 âm lịch - tra bảng).
* Ngày âm lịch của đầu tháng dương lịch: 4.(Ngày 4/2 âm lịch là ngày 1/3 dương lịch - tra bảng).

Ngày dương lịch ứng với ngày đầu tháng âm lịch khác ngày hiện tại.

Ngày dương lịch ứng với ngày đầu tháng âm lịch trừ ngày hiện tại: 26 – 13 = 13. C = 0. Ngày âm lịch: 13 + 4 – 1 = 16.

Tính tháng Âm lịch:

Bằng cách lấy ngày Dương lịch tương ứng với đầu tháng Âm lịch trong tháng trừ ngày hiện tại:

* Nếu không có nhớ (C = 0) thì tháng Âm lịch hiện tại bằng tháng Âm lịch của đầu tháng Dương lịch.
* Nếu có nhớ (C = 1) thì tháng Âm lịch hiện tại bằng tháng Âm lịch của ngày đầu tháng Âm lịch.

***Ví dụ:*** Tính tháng âm lịch của ngày 13/03/2017

Tháng âm lịch của đầu tháng dương lịch: 2 (tra bảng).

Ngày dương lịch ứng với đầu tháng âm lịch trừ ngày hiện tại: 26 – 13 = 13. C = 0. Như vậy tháng âm hiện tại bằng tháng âm lịch của đầu tháng dương lịch: 2.

Tính năm Âm lịch:

Bằng cách lấy tháng Dương lịch trừ cho tháng Âm lịch:

* Nếu không có nhớ (C = 0) thì năm Âm lịch bằng năm Dương lịch.
* Nếu có nhớ (C = 1) thì năm Âm lịch hiện tại bằng năm Dương lịch trừ 1.

Ví dụ: Tính năm âm lịch của ngày 13/03/2017

Tháng dương lịch trừ tháng âm lịch: 3 – 2 = 1.

Như vậy năm âm lịch bằng năm dương lịch: 2017

Vây, ngày âm lịch của ngày 13/3/2017 là ngày 16/2/2017.

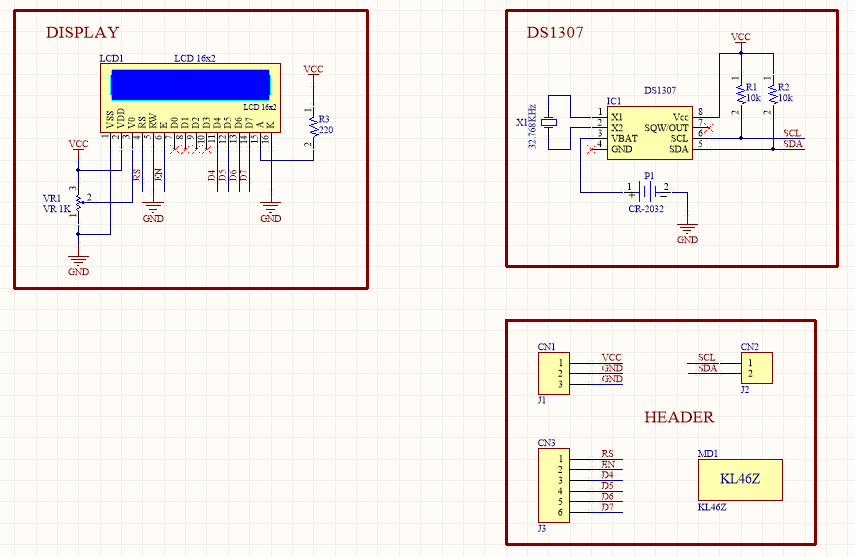
Trong đó:

* Ngày Dương lịch: có sẵn trong IC thời gian thực.
* Ngày Âm lịch: kết quả thu được
* Ngày Dương lịch đầu tháng Âm lịch: tra bảng
* Ngày Âm lịch của đầu tháng Dương lịch: tra bảng
* Tháng Dương lịch: có sẵn trong IC thời gian thực.
* Tháng Âmlịch: kết quả thu được.
* Tháng Âm lịch đầu tháng Dương lịch: tra bảng.
* Năm Dương lịch: có sẵn trong IC thời gian thực.
* Năm Âm lịch: kết quả thu được.

Bảng ngày tháng Âm, Dương lịch được khai báo trong code.

## Lập trình

**Sơ đồ nguyên lý**

****

Hình : Sơ đồ nguyên lý

**Các bước lập trình:**

* Bước 1: Xây dựng thư viện LCD16x2, I2C, DS1307 giao tiếp với KL46Z.
* Bước 2: Sử dụng thư viện ghi giá trị khởi tạo cho DS1307.
* Bước 3: Đọc các giá trị từ DS1307 và hiển thị lên màn hình LCD16x2.

### Tạo thư viện

Các thư viện được chia thành 2 file chính: file .h dùng để định nghĩa các kết nối, hằng số… khai báo các nguyên mẫu hàm và file .c dùng để định nghĩa các hàm đã được khai báo trong file .h.

**Thư viện LCD16x2**

File LCD16x2.h

#ifndef \_LCD16x2\_H

#define \_LCD16x2\_H

#include "MKL46Z4.h"

#include "Define.h"

#define LCD\_RS PTC, 1

#define LCD\_EN PTC, 2

#define LCD\_D4 PTB, 3

#define LCD\_D5 PTB, 2

#define LCD\_D6 PTB, 1

#define LCD\_D7 PTB, 0

//================================================================

// LCD16x2

//================================================================

#define LCD16x2\_LINE\_2 0x40 // line 2 tu 0x40 den 0x4F

// cac lenh xoa va xuong dong LCD

#define LCD\_CL 0x01

#define LCD\_ENTER 0x02

// font set

#define LCD\_F1 0x28 //giao tiep 4 bit, 2 dong voi font 5x8

#define LCD\_F2 0x2c //giao tiep 4 bit, 2 dong voi font 5x10

#define LCD\_F3 0x38 //giao tiep 8 bit, 2 dong voi font 5x8

#define LCD\_F4 0x3c //giao tiep 8 bit, 2 dong voi font 5x10

// on off control

#define LCD\_ON\_CTR\_1 0x0e // hien thi, con tro ko nhap nhay

#define LCD\_ON\_CTR\_2 0x0f// hien thi, con tro co nhap nhay

#define LCD\_ON\_CTR\_3 0x0c // hien thi, ko hien thi con tro

// set che do nhap

#define LCD\_T1 0x04 // hien thi giam va khong shift

#define LCD\_T2 0x05 // hien thi giam va shift

#define LCD\_T3 0x06 // hien thi tang va khong shift

#define LCD\_T4 0x07 // hien thi tang va shift

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: enable LCD

\* Parameters: void

\* Return: void

\*/

void LCD\_Enable ( void );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: send 4bit to LCD

\* Parameters: unsigned char Data

\* Return: void

\*/

void LCD\_Send4Bit ( unsigned char Data );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: send command to LCD

\* Parameters: unsigned char command

\* Return: void

\*/

void LCD\_SendCommand (unsigned char command );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: Init LCD

\* Parameters: void

\* Return: void

\*/

void LCD\_Init ( void );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: go to row x col y

\* Parameters: unsigned char x, unsigned char y

\* Return: void

\*/

void LCD\_Gotoxy (unsigned char x, unsigned char y);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: clear LCD

\* Parameters: void

\* Return: void

\*/

void LCD\_Clear ( void );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: send char to LCD

\* Parameters: unsigned char Data

\* Return: void

\*/

void LCD\_PutChar ( unsigned char Data );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Function: send string to LCD

\* Parameters: char \*s

\* Return: void

\*/

void LCD\_Puts (char \*s);

#endif

File LCD16x2.c

#include "LCD16x2.h"

void LCD\_Enable (void)

{

GPIO\_SetPortBit (LCD\_EN);

delay\_us (3);

GPIO\_ClearPortBit (LCD\_EN);

delay\_us (50);

}

void LCD\_Send4Bit ( uint8\_t Data )

{

GPIO\_WritePortBit (LCD\_D4, Data & 0x01);

GPIO\_WritePortBit (LCD\_D5, (Data >> 1) & 0x01);

GPIO\_WritePortBit (LCD\_D6, (Data >> 2) & 0x01);

GPIO\_WritePortBit (LCD\_D7, (Data >> 3) & 0x01);

}

void LCD\_SendCommand (uint8\_t command )

{

LCD\_Send4Bit ( command >>4 );/\* Gui 4 bit cao \*/

LCD\_Enable () ;

LCD\_Send4Bit ( command ); /\* Gui 4 bit thap\*/

LCD\_Enable () ;

}

void LCD\_Init ( void )

{

GPIO\_SetPortBit (LCD\_D4);

GPIO\_SetPortBit (LCD\_D5);

GPIO\_SetPortBit (LCD\_D6);

GPIO\_SetPortBit (LCD\_D7);

GPIO\_SetPortBit (LCD\_EN);

GPIO\_SetPortBit (LCD\_RS);

GPIO\_ClearPortBit (LCD\_RS);

delay\_ms(20);

LCD\_SendCommand (0x30);

delay\_ms(10);

LCD\_SendCommand (0x30);

delay\_ms(10);

LCD\_SendCommand (0x30);

delay\_us(10);

LCD\_SendCommand (LCD\_ENTER);

delay\_ms(10);

LCD\_SendCommand ( LCD\_F1 ); // giao thuc 4 bit, hien thi 2 hang, ki tu 5x8

LCD\_SendCommand ( LCD\_ON\_CTR\_3); // cho phep hien thi man hinh

LCD\_SendCommand ( LCD\_T3 ); // tang ID, khong dich khung hinh

LCD\_SendCommand ( LCD\_CL ); // xoa toan bo khung hinh

delay\_ms(10);

}

void LCD\_Gotoxy (unsigned char x, unsigned char y)

{

unsigned char address;

if (!y)

address = (0x80+x);

else

address = (0xC0+x);

delay\_us (1000);

LCD\_SendCommand(address);

delay\_us (50);

}

void LCD\_Clear ()

{

LCD\_SendCommand (0x01);

delay\_ms (10);

}

void LCD\_PutChar ( unsigned char Data )

{

GPIO\_SetPortBit (LCD\_RS);

LCD\_SendCommand ( Data );

GPIO\_ClearPortBit (LCD\_RS);

delay\_us (37);

}

void LCD\_Puts (char \*s)

{

while (\*s)

{

LCD\_PutChar (\*s);

s++;

}

}

**Thư viện I2C**

File I2C.h

#ifndef \_I2C\_H

#define \_I2C\_H

#include "MKL46Z4.h"

typedef enum{

I2C\_ACK = 0,

I2C\_NACK = 1,

} I2C\_Acknowledge\_Bit;

void I2C\_Disable(I2C\_Type\* i2c);

void I2C\_DisableInt(I2C\_Type\* i2c);

void I2C\_Enable(I2C\_Type\* i2c);

void I2C\_EnableInt(I2C\_Type\* i2c);

uint8\_t I2C\_ReadByte(I2C\_Type\* i2c, uint8\_t ack);

void I2C\_Restart(I2C\_Type\* i2c);

void I2C\_Start(I2C\_Type\* i2c);

void I2C\_Stop(I2C\_Type\* i2c);

uint8\_t I2C\_WriteByte(I2C\_Type\* i2c, uint8\_t data);

#endif

File I2C.c

#include "I2C.h"

// Ham vo hieu hoa I2C

void I2C\_Disable(I2C\_Type\* i2c)

{

i2c->C1 &= ~I2C\_C1\_IICEN\_MASK;

}

void I2C\_DisableInt(I2C\_Type\* i2c){

i2c->C1 &= ~I2C\_C1\_IICIE\_MASK;

}

// Ham cho phep I2C hoat dong

void I2C\_Enable(I2C\_Type\* i2c){

i2c->C1 |= I2C\_C1\_IICEN\_MASK;

}

void I2C\_EnableInt(I2C\_Type\* i2c){

i2c->C1 |= I2C\_C1\_IICIE\_MASK;

}

// Ham doc 1 byte

uint8\_t I2C\_ReadByte(I2C\_Type\* i2c, uint8\_t ack){

// Lua chon che do nhan

i2c->C1 &= ~I2C\_C1\_TX\_MASK;

// chuan bi bit ACK/NACK neu FACK == 0

if((i2c->SMB & I2C\_SMB\_FACK\_MASK) == 0)

i2c->C1 = (ack == I2C\_NACK) ? i2c->C1 | I2C\_C1\_TXAK\_MASK : i2c->C1 & ~I2C\_C1\_TXAK\_MASK;

// Xoa co IICIF

i2c->S |= I2C\_S\_IICIF\_MASK;

// Khoi tao du lieu truyen

(void)i2c->D;

// Cho den khi truyen xong

while((i2c->S & I2C\_S\_IICIF\_MASK) == 0);

// Gui bit ACK/NACK neu FACK == 1

if((i2c->SMB & I2C\_SMB\_FACK\_MASK) != 0)

i2c->C1 = (ack == I2C\_NACK) ? i2c->C1 | I2C\_C1\_TXAK\_MASK : i2c->C1 & ~I2C\_C1\_TXAK\_MASK;

// Lua chon che do truyen

i2c->C1 |= I2C\_C1\_TX\_MASK;

// Tra ve du lieu nhan

return i2c->D;

}

void I2C\_Restart(I2C\_Type\* i2c){

i2c->C1 |= I2C\_C1\_RSTA\_MASK;

}

void I2C\_Start(I2C\_Type\* i2c){

i2c->C1 |= I2C\_C1\_MST\_MASK;

}

void I2C\_Stop(I2C\_Type \* i2c){

// Xoa co STOPF

i2c->FLT |= I2C\_FLT\_STOPF\_MASK;

// Viet bit STOP

i2c->C1 &= ~I2C\_C1\_MST\_MASK;

// cho cho den khi bit STOP duoc gui

while((i2c->FLT & I2C\_FLT\_STOPF\_MASK) == 0){

i2c->C1 &= ~I2C\_C1\_MST\_MASK;

}

}

uint8\_t I2C\_WriteByte(I2C\_Type\* i2c, uint8\_t data){

// Lua chon che do truyen

i2c->C1 |= I2C\_C1\_TX\_MASK;

// Xoa co IICIF

i2c->S |= I2C\_S\_IICIF\_MASK;

// Khoi tao du lieu truyen

i2c->D = data;

// Cho cho den khi truyen xong

while((i2c->S & I2C\_S\_IICIF\_MASK) == 0);

// Tra ve bit ACK nhan duoc

return ((i2c->S & I2C\_S\_RXAK\_MASK) == I2C\_S\_RXAK\_MASK ? I2C\_NACK : I2C\_ACK);

}

**Thư viện DS1307**

File DS1307.h.

**Lưu ý:** Trong báo cáo chỉ khai báo dữ liệu cho năm 2000, 2001 để làm ví dụ.

#ifndef \_DS1307\_H

#define \_DS1307\_H

#include "MKL46Z4.h"

void Ds1307\_Init(void);

uint8\_t Ds1307\_Read(uint8\_t address);

void Ds1307\_Write(uint8\_t address, uint8\_t data);

#define BEGINNING\_YEAR 0

unsigned int LUNAR\_CALENDAR\_LOOKUP\_TABLE[] =

{

//2000

0x1B79,0x099A,0x1A3A,0x125B,0x187B,0x109D,0x1ABE,0x18E2,0x1304,0x1924,0x1346,0x1B66,

//2001

0x1987,0x0229,0x1A47,0x1068,0x1A89,0x108A,0x1AAB,0x18CC,0x10EE,0x1B0F,0x1130,0x1B51,

};

struct MONTH\_INFO{

unsigned int N\_AL\_DT\_DL :5;

unsigned int T\_AL\_DT\_DL :4;

unsigned int SN\_CT\_AL :1;

unsigned int TN\_B\_THT :1;

unsigned int SN\_CT\_DL :2;

};

union LUNAR\_RECORD

{

unsigned int Word;

struct MONTH\_INFO Info;

};

#endif

File DS1307.c

#include "DS1307.h"

uint8\_t dataTran, dataRec;

void Ds1307\_Init(void)

{

I2C\_Enable(I2C1); // cho phep I2C hoat dong

I2C\_Start(I2C1); // Status

I2C\_WriteByte(I2C1,0xD0); // SLA+W

I2C\_WriteByte(I2C1,0x00); // Dia chi thanh ghi Giay

I2C\_WriteByte(I2C1,0x01); // Thiet lap gia tri thanh ghi Giay

I2C\_WriteByte(I2C1,0x26); // Thiet lap gia tri thanh ghi Phut

I2C\_WriteByte(I2C1,0x17); // Thiet lap gia tri thanh ghi Gio

I2C\_WriteByte(I2C1,0x02); // Thiet lap gia tri thanh ghi Thu

I2C\_WriteByte(I2C1,0x24); // Thiet lap gia tri thanh ghi Ngay

I2C\_WriteByte(I2C1,0x04); // Thiet lap gia tri thanh ghi Thang

I2C\_WriteByte(I2C1,0x17); // Thiet lap gia tri thanh ghi Nam

I2C\_WriteByte(I2C1,0x90);

I2C\_Stop(I2C1);

}

uint8\_t Ds1307\_Read(uint8\_t address)

{

uint8\_t data;

I2C\_Start(I2C1);

I2C\_WriteByte(I2C1,0xD0); // SLA+W

I2C\_WriteByte(I2C1,address); // Dia chi thanh ghi

I2C\_Restart(I2C1); // Tin hieu Restart

I2C\_WriteByte(I2C1,0xD1); // SLA+R

data = I2C\_ReadByte(I2C1,I2C\_NACK);

//tmp = data;

// dataRec = ((tmp>>4)\*10) + data&0x0F;

dataRec = (data>>4) ;

dataRec = dataRec \* 10;

dataRec += data & 0x0F;

I2C\_Stop(I2C1);

return dataRec;

}

void Ds1307\_Write(uint8\_t address, uint8\_t data)

{

I2C\_Start(I2C1);

I2C\_WriteByte(I2C1,0xD0); // SLA+W

I2C\_WriteByte(I2C1,address);

dataTran = (data/10)<<4 | data%10;

I2C\_WriteByte(I2C1,data);

I2C\_Stop(I2C1);

}

void Solar2Lunar(unsigned char SolarDate, unsigned char SolarMonth, unsigned char SolarYear,

unsigned char \* LunarDate, unsigned char \* LunarMonth, char \* LunarYear)

{

unsigned char N\_AL\_DT\_DL;

unsigned char T\_AL\_DT\_DL;

unsigned char SN\_CT\_AL;

unsigned char TN\_B\_THT;

unsigned char N\_AL\_DT\_DL\_TT;

unsigned char T\_AL\_DT\_DL\_TT;

union LUNAR\_RECORD lr;

lr.Word = LUNAR\_CALENDAR\_LOOKUP\_TABLE[(SolarYear-BEGINNING\_YEAR)\*12+SolarMonth -1];

N\_AL\_DT\_DL = lr.Info.N\_AL\_DT\_DL;

T\_AL\_DT\_DL = lr.Info.T\_AL\_DT\_DL;

SN\_CT\_AL = lr.Info.SN\_CT\_AL + 29;

TN\_B\_THT = lr.Info.TN\_B\_THT;

lr.Word = LUNAR\_CALENDAR\_LOOKUP\_TABLE[(SolarYear-BEGINNING\_YEAR)\*12+SolarMonth];

N\_AL\_DT\_DL\_TT = lr.Info.N\_AL\_DT\_DL;

T\_AL\_DT\_DL\_TT = lr.Info.T\_AL\_DT\_DL;

// Tinh ngay & thang

if(N\_AL\_DT\_DL == SN\_CT\_AL && N\_AL\_DT\_DL\_TT == 2)

{

if(SolarDate==1)

{

(\*LunarDate) = N\_AL\_DT\_DL;

(\*LunarMonth) = T\_AL\_DT\_DL;

}

else if(SolarDate==31)

{

(\*LunarDate) = 1;

(\*LunarMonth) = T\_AL\_DT\_DL\_TT;

}

else

{

(\*LunarDate) = SolarDate - 1;

if(TN\_B\_THT)

{

(\*LunarMonth) = T\_AL\_DT\_DL;

}

else

{

(\*LunarMonth) = T\_AL\_DT\_DL==12?1:(T\_AL\_DT\_DL + 1);

}

}

}

else

{

(\*LunarDate) = SolarDate + N\_AL\_DT\_DL - 1;

if((\*LunarDate)<= SN\_CT\_AL)

{

(\*LunarMonth) = T\_AL\_DT\_DL;

}

else

{

(\*LunarDate) -= SN\_CT\_AL;

(\*LunarMonth) = T\_AL\_DT\_DL + 1 - TN\_B\_THT;

if((\*LunarMonth) == 13) (\*LunarMonth) = 1;

}

}

// Tinh Nam

if(SolarMonth >= (\*LunarMonth))

{

(\*LunarYear) = SolarYear;

}

else

{

(\*LunarYear) = SolarYear - 1;

}

}

Ngoài ra, cần sử dụng thêm thư viện define.h để thuận tiện trong quá trình viết code:

#include "MKL46Z4.h"

typedef enum

{

INPUT = 0,

OUTPUT = 1,

} GPIO\_IO\_Dir;

typedef enum

{

LOW = 0,

HIGH = 1,

} GPIO\_OUT\_State;

void GPIO\_ChangePortBitDirection(GPIO\_Type\* port, uint8\_t bit, GPIO\_IO\_Dir dir);

void GPIO\_ClearPortBit(GPIO\_Type\* port, uint8\_t bit);

uint8\_t GPIO\_ReadPortBit(GPIO\_Type\* port, uint8\_t bit);

void GPIO\_SetPortBit(GPIO\_Type\* port, uint8\_t bit);

void GPIO\_TogglePortBit(GPIO\_Type\* port, uint8\_t bit);

void GPIO\_WritePortBit(GPIO\_Type\* port, uint8\_t bit, uint8\_t state);

void delay\_1ms(void);

void delay\_ms(int timeMS);

#define delayCounter\_ms 7200

#define delayCounter\_us 7

#define PIN\_B0 0

#define PIN\_B1 1

#define PIN\_B2 2

#define PIN\_B3 3

#define PIN\_B4 4

#define PIN\_B5 5

#define PIN\_B6 6

#define PIN\_B7 7

#define PIN\_B8 8

#define PIN\_B9 9

#define PIN\_B10 10

#define PIN\_B11 11

#define PIN\_B12 12

#define PIN\_B13 13

#define PIN\_B14 14

#define PIN\_B15 15

#define PIN\_B16 16

#define PIN\_B17 17

#define PIN\_B18 18

#define PIN\_B19 19

#define PIN\_B20 20

#define PIN\_B21 21

#define PIN\_B22 22

#define PIN\_B23 23

#define PIN\_B24 24

#define PIN\_B25 25

#define PIN\_B26 26

#define PIN\_B27 27

#define PIN\_B28 28

#define PIN\_B29 29

#define PIN\_B30 30

void delay\_1us(void)

{

uint32\_t counter = 0;

while(counter < delayCounter\_us)

{ counter++; }

}

void delay\_1ms(void)

{

uint32\_t counter = 0;

while(counter < delayCounter\_ms)

{ counter++; }

}

void delay\_ms(int timeMS)

{

uint32\_t time = 0;

while(time < timeMS)

{

time++;

delay\_1ms();

}

}

void delay\_us(int timeuS)

{

uint32\_t time = 0;

while(time < timeuS)

{

time++;

delay\_1us();

}

}

void Clock\_Config(void)

{

SystemCoreClockUpdate();

SIM->SCGC4 |= SIM\_SCGC4\_I2C1\_MASK; // Enable clock for I2C1

SIM->SCGC5 |= SIM\_SCGC5\_PORTA\_MASK // Enable clock for Port A

| SIM\_SCGC5\_PORTB\_MASK // Enable clock for Port B

| SIM\_SCGC5\_PORTC\_MASK // Enable clock for Port C

| SIM\_SCGC5\_PORTD\_MASK // Enable clock for Port D

| SIM\_SCGC5\_PORTE\_MASK; // Enable clock for Port E

}

void GPIO\_ChangePortBitDirection(GPIO\_Type\* port, uint8\_t bit, GPIO\_IO\_Dir dir){

if(dir == INPUT){

port->PDDR &= ~((uint32\_t)1<<bit);

}

else if(dir == OUTPUT){

port->PDDR |= ((uint32\_t)1<<bit);

}

}

void GPIO\_ClearPortBit(GPIO\_Type\* port, uint8\_t bit){

port->PCOR = (uint32\_t)1<<bit;

}

void GPIO\_SetPortBit(GPIO\_Type\* port, uint8\_t bit){

port->PSOR = (uint32\_t)1<<bit;

}

void GPIO\_TogglePortBit(GPIO\_Type\* port, uint8\_t bit){

port->PTOR = (uint32\_t)1<<bit;

}

void GPIO\_WritePortBit(GPIO\_Type\* port, uint8\_t bit, uint8\_t state){

port->PDOR = (port->PDOR & ~((uint32\_t)1<<bit)) | ((uint32\_t)state<<bit);

}

uint8\_t GPIO\_ReadPortBit(GPIO\_Type\* port, uint8\_t bit){

return (uint8\_t)(((port->PDIR)>>bit)&0x1);

}

### Chương trình chính

Tạo mới project và thêm các file với nội dung như sau:

File main.h

#ifndef \_MAIN\_H

#define \_MAIN\_H

#include "MKL46Z4.h" // Device header

#include "LCD16x2.c"

#include "I2C.c"

#include "DS1307.c"

#endif

File main.c

#include "main.h"

void GPIO\_Config(void);

void I2C\_Init(void);

int main(void)

{

uint8\_t second = 0, minute = 0, hour = 0;

uint8\_t day = 0, date = 0,month = 0, year = 0;

uint8\_t SolarDate, SolarMonth;

char SolarYear;

char str[16];

Clock\_Config();

GPIO\_Config();

LCD\_Init();

I2C\_Init();

Ds1307\_Init();

while(1)

{

second = Ds1307\_Read(0x00);

minute = Ds1307\_Read(0x01);

hour = Ds1307\_Read(0x02);

day = Ds1307\_Read(0x03);

date = Ds1307\_Read(0x04);

month = Ds1307\_Read(0x05);

year = Ds1307\_Read(0x06);

LCD\_Gotoxy(0,0);

sprintf(str, "%2u:%2u:",hour,minute);

LCD\_Puts(str);

if(second < 10)

{

LCD\_Gotoxy(6,0);

sprintf(str, "0");

LCD\_Puts(str);

LCD\_Gotoxy(7,0);

sprintf(str, "%1u",second);

LCD\_Puts(str);

}

else

{

LCD\_Gotoxy(6,0);

sprintf(str, "%2u",second);

LCD\_Puts(str);

}

delay\_ms(10);

LCD\_Gotoxy(0,1);

sprintf(str, "%2u/",date);

LCD\_Puts(str);

if(month < 10)

{

LCD\_Gotoxy(3,1);

sprintf(str, "0");

LCD\_Puts(str);

LCD\_Gotoxy(4,1);

sprintf(str, "%1u",month);

LCD\_Puts(str);

}

else

{

LCD\_Gotoxy(3,1);

sprintf(str, "%2u",month);

LCD\_Puts(str);

}

LCD\_Gotoxy(5,1);

sprintf(str, "/20");

LCD\_Puts(str);

LCD\_Gotoxy(8,1);

sprintf(str, "%2u",year);

LCD\_Puts(str);

delay\_ms(10);

Solar2Lunar(date,month,year,&SolarDate,&SolarMonth,&SolarYear);

if(SolarDate < 10)

{

LCD\_Gotoxy(11,1);

sprintf(str, "0");

LCD\_Puts(str);

LCD\_Gotoxy(12,1);

sprintf(str, "%1u/",SolarDate);

LCD\_Puts(str);

}

else

{

LCD\_Gotoxy(11,1);

sprintf(str, "%2u/",SolarDate);

LCD\_Puts(str);

}

if(SolarMonth < 10)

{

LCD\_Gotoxy(14,1);

sprintf(str, "0");

LCD\_Puts(str);

LCD\_Gotoxy(15,1);

sprintf(str, "%1u",SolarMonth);

LCD\_Puts(str);

}

else

{

LCD\_Gotoxy(14,1);

sprintf(str, "%2u",SolarMonth);

LCD\_Puts(str);

}

delay\_ms(10);

switch(day)

{

case 1:

{

LCD\_Gotoxy(10,0);

sprintf(str, "CN");

LCD\_Puts(str);

break;

}

case 2:

{

LCD\_Gotoxy(10,0);

sprintf(str, "Thu 2");

LCD\_Puts(str);

break;

}

case 3:

{

LCD\_Gotoxy(10,0);

sprintf(str, "Thu 3");

LCD\_Puts(str);

break;

}

case 4:

{

LCD\_Gotoxy(10,0);

sprintf(str, "Thu 4");

LCD\_Puts(str);

break;

}

case 5:

{

LCD\_Gotoxy(10,0);

sprintf(str, "Thu 5");

LCD\_Puts(str);

break;

}

case 6:

{

LCD\_Gotoxy(10,0);

sprintf(str, "Thu 6");

LCD\_Puts(str);

break;

}

case 7:

{

LCD\_Gotoxy(10,0);

sprintf(str, "Thu 7");

LCD\_Puts(str);

break;

}

}

}

}

void GPIO\_Config(void)

{

PORTC->PCR[PIN\_B1] = PORT\_PCR\_MUX(OUTPUT); // Alternative function: 1 (GPIO)

PORTC->PCR[PIN\_B2] = PORT\_PCR\_MUX(OUTPUT); // Alternative function: 1 (GPIO)

PORTB->PCR[PIN\_B3] = PORT\_PCR\_MUX(OUTPUT);

PORTB->PCR[PIN\_B2] = PORT\_PCR\_MUX(OUTPUT);

PORTB->PCR[PIN\_B1] = PORT\_PCR\_MUX(OUTPUT);

PORTB->PCR[PIN\_B0] = PORT\_PCR\_MUX(OUTPUT);

GPIO\_ChangePortBitDirection(LCD\_RS, OUTPUT);

GPIO\_ChangePortBitDirection(LCD\_EN, OUTPUT);

GPIO\_ChangePortBitDirection(LCD\_D4, OUTPUT);

GPIO\_ChangePortBitDirection(LCD\_D5, OUTPUT);

GPIO\_ChangePortBitDirection(LCD\_D6, OUTPUT);

GPIO\_ChangePortBitDirection(LCD\_D7, OUTPUT);

PORTE->PCR[0] = PORT\_PCR\_MUX(6); // Alternative function: 6 (I2C1\_SDA)

PORTE->PCR[1] = PORT\_PCR\_MUX(6); // Alternative function: 6 (I2C1\_SCL)

}

void I2C\_Init(void)

{

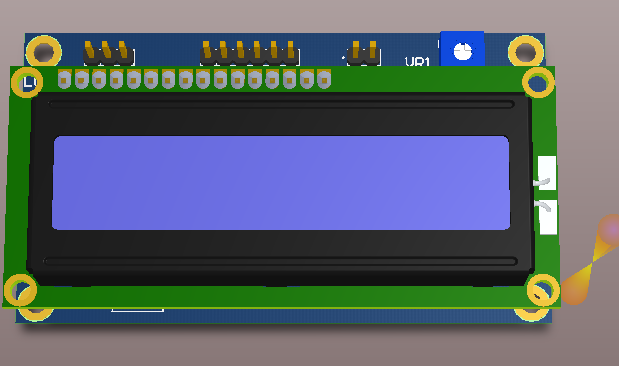
// Set baudrate to 400 kb/s

I2C1->F = I2C\_F\_MULT(2)

| I2C\_F\_ICR(0x5);

}

**Kết quả**

****

Hình : Thiết kế mạch PCB

****

Hình : Kết quả thực tế

Trên màn hình LCD:

* Hàng 1: Hiển thị giờ, phút, giây, thứ.
* Hàng 2: Hiển thị ngày tháng năm dương, ngày tháng âm lịch.