

**Bachelier en Informatique et Systèmes
Informatique Industrielle
Bloc2**



Catégorie technique

Rapport Final

**Projet Multidisciplinaire 3.
Systèmes embarqués.
Développement Mobile.**

2019-2020

Wambe Thomas

Abstract :

In Third year in industrial informatics school, a project is required. Mine was a concatenation of a personal project and the requirement from professors. My personal was an E-Penne.

What is an E-Penne would you say? First what is a Penne? A Penne is a student headwear we can wear in certain student's festivities. So, an E-Penne is a connected Penne. That means that I can communicate with it. A headwear isn't an informatics stuff, we need to have something to communicate with. My project was to implement a led display in the back of this headwear and be able to control it in Bluetooth with a smartphone.

What was the professors' requirement so? We needed to be able to cast sensors information on internet and get them back into a smartphone application using LoRa module. With those requirement, I needed to make an application, but haven't an android smartphone, I've done one with an IOS smartphone, and that wasn't easy. I was forced to make an application to get those internet information, why could I not make my own application to communicate with the display. It's what I've done.

Started with no idea how to program under IOS system, ending with a totally functional application. Same for Bluetooth, I didn't know how to works with Bluetooth at the beginning of the project. Now, I can start a communication with a Bluetooth module, sends information, detect the module's state and end a communication with it. I also can take those information out the Bluetooth module and gives them to microprocessor like Arduino. Using LoRa is a little bit trickier. Started at the same point as application and Bluetooth module and ending with a semi-functional frames sender. All those points will be detailed in this document.

Table des matières :

ABSTRACT :	1
TABLE DES MATIERES :	2
INTRODUCTION GENERALE :	3
DEVELOPPEMENT :	4
SYSTEMES EMBARQUES :	4
PROJECT:	7
DEVELOPPEMENT MOBILE :	8
PROBLEMES RENCONTRES :	10
SYSTEMES EMBARQUES :	10
PROJECT :	10
DEVELOPPEMENT MOBILE :	11
PROJETS PERSONNELS :	12
CONCLUSION GENERALE :	12
LEXIQUE :	12
ANNEXES :	2
CODE PRESENT SUR L'ARDUINO PRO-MINI :	2
CODE PRESENT SUR L'ARDUINO UNO :	3
CODE PRESENT SUR L'IPHONE :	4
SCHEMAS ELECTRONIQUES :	8
DATASHEETS :	10

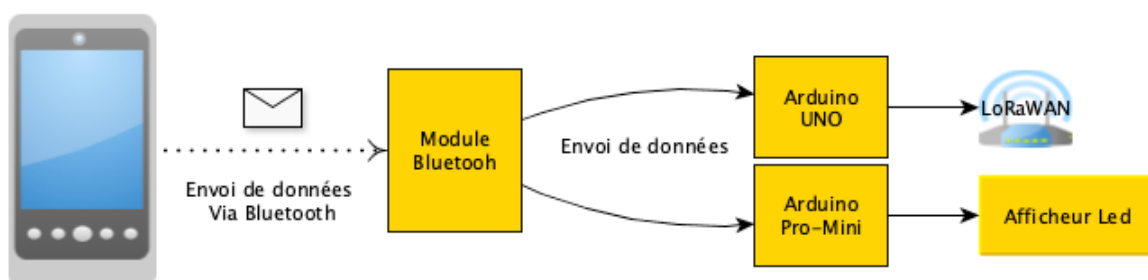
Introduction générale :

Lors de la troisième année en bachelier en informatique industriel, il nous a été demandé de réaliser un projet comprenant un microcontrôleur, un capteur pouvant communiquer avec un certain protocole, un actionneur et un module LoRa compatible avec notre microcontrôleur.

Le microcontrôleur imposé au départ fut un STM32. Cette idée fut vite abandonnée pour cause de problèmes de compatibilité avec le module LoRa. Ayant déjà pensé à un projet personnel au préalable, j'ai alors demandé aux professeurs encadrant ce projet si celui-ci convenait à leurs attentes.

Le projet personnel étant le suivant : Installer un afficheur led à l'arrière de ma pence (couvre-chef étudiantin), pouvant être commandé via mon téléphone en Bluetooth. Ce projet requiert en effet un microcontrôleur, un capteur communiquant avec deux protocoles (Bluetooth et UART) et un actionneur étant l'afficheur led. Effectivement le module LoRa dans ce projet n'est pas indispensable mais il permet d'avoir un historique, dans une base de données, des informations envoyées à l'afficheur.

Ayant eu l'aval des professeurs, j'ai pu commander les différentes pièces dont j'avais besoin. Celles-ci ont mis énormément de temps avant d'arriver (notamment le module Bluetooth qui était lui reparti en chine). Étant fort impatient et stressé à l'idée de ne pas les avoir à temps, je l'ai ai commandé de nouveau sur un autre site. J'ai ainsi reçu mes pièces en double. Le fait d'avoir toutes les pièces en double m'a, au final permis de pouvoir présenter un projet fini, satisfaisant les attentes des professeurs mais également de pouvoir présenter mon projet personnel, pensé à la base, sur ma pence.



Développement :

Systèmes embarqués :

Pour la réalisation de ce projet, j'ai décidé d'utiliser différents composants tels qu'un Arduino Uno, un module LoRaWan, un Arduino Pro-Mini, un Module Bluetooth de type SH-HC-08, un module de charge pour batterie 3.7V, un afficheur à matrice de leds et une batterie Li-Po de 3.7V 800mAh ou 500mAh.

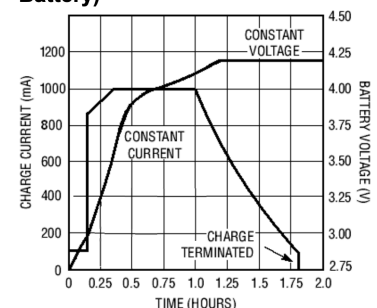
L'Arduino Uno vérifie constamment si une donnée est récupérée depuis le module Bluetooth et le cas échéant, l'Arduino la transmet, en brut, au module LoRa pour que celui-ci l'envoie aux différentes passerelles environnantes. (Ce point sera expliqué dans la partie « Project » de ce travail).

L'Arduino Pro-Mini vérifie également les trames reçues depuis le module Bluetooth et au même titre que l'Arduino Uno, il réagit seulement si une donnée a été envoyée. Lors de la réception d'une trame, celle-ci est alors déchiffrée et les différentes parties de cette trame font réagir l'afficheur à matrice de leds. La première partie de l'information est constituée de texte à faire défiler sur l'afficheur, la seconde partie sert à modifier l'intensité lumineuse des leds. Ces deux informations sont séparées par un caractère de séparation.

Le module de charge (TP4056) n'est fondamentalement pas nécessaire au bon fonctionnement du projet, mais il offre une grande sécurité au niveau du chargement des batteries Li-Po. En effet les batteries de ce type réagissent très mal aux surcharges, courts-circuits et décharges totales. Ce module permet une régulation correcte de la charge (Maximum 1000mA). Il arrête également la charge une fois la batterie totalement pleine.

La charge complète se fait en approximativement 1h45 pour des batteries de 1000mAh. Ayant des batteries de capacités différentes, le temps de chargement complet est différent selon la batterie utilisée. Pour une batterie de 800mAh, le temps de charge est d'environ 1h24, pour celle de 500mAh la charge se fait en 52 minutes et 30 secondes. (Valeurs théoriques)

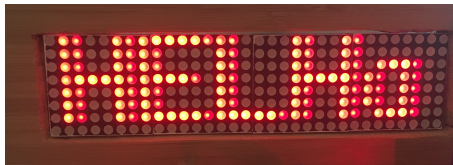
Complete Charge Cycle (1000mAh Battery)



La décharge, quant à elle, est moins « linéaire » et peut grandement varier. Cela est dû à la consommation en courant des différents modules. La consommation totale en sortie du module de charge est en partie proportionnelle au nombre de leds allumées sur l'afficheur et à l'intensité lumineuse de ces dernières. Comme précisé précédemment, la trame contient les informations pour gérer la luminosité de la matrice de led. Ainsi pour un même texte, la consommation en courant diffère comme ceci :

Luminosité :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Consommation (mA) :	90	100	110	120	130	140	150	160	170	180	190	200	210	220	230	240

Lors de l'allumage du système, la consommation atteint parfois un pic de 380mA pendant quelques millisecondes mais se stabilise très vite aux valeurs ci-dessus. À la réception d'une donnée, un autre pic survient durant quelques millisecondes. Si la luminosité est définie à 0 - leds très peu lumineuses -, il sera de 149mA. Si la luminosité est définie à 15 - leds allumées au maximum -, il sera alors de 290mA approximativement. Ces pics traduisent respectivement la mise sous tension des modules (ainsi que l'allumage de toutes les leds de l'afficheur) et l'envoi d'une trame par le module LoRaWan (cet envoi faisant monter la consommation générale un léger instant).



Luminosité 0



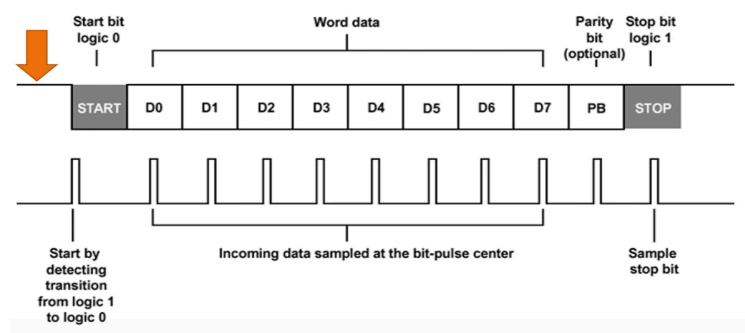
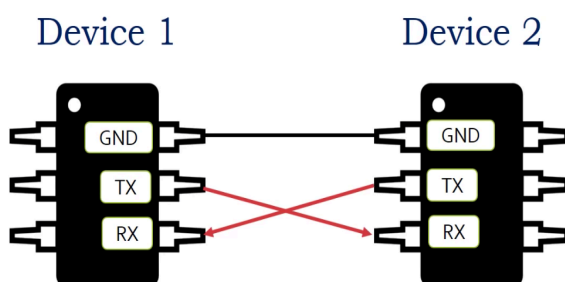
Luminosité 15

Pour ce qui est du module Bluetooth (SH-HC-08), il est branché en parallèle aux Arduino Uno et Arduino Pro-Mini par soucis d'espace mémoire dans l'Arduino Uno seul (point expliqué dans la partie « Problèmes rencontrés »). La communication entre le module qui envoie la donnée (dans ce cas un téléphone) et celui qui reçoit la donnée est le Bluetooth 4.0 également appelé « Bluetooth Low Energy (B.L.E.) ». Cette communication se fait par onde radio sur 40 canaux à une fréquence de 2MHz. La différence entre le Bluetooth classique et le B.L.E. est que le B.L.E. peut transmettre le même débit de données (1Mbit/s) avec une consommation moindre (0,1W au lieu de 1W qui sont les consommations de référence). C'est ce qui permet l'implantation du Bluetooth L.E. dans de petits appareils tels que des montres ou des téléphones. Le mode de modulation de fréquence dans un canal est le G.F.S.K. (Gaussian Frequency-Shift Keying). Ce mode de modulation permet entre autres de réduire les interférences entre les canaux.

Lorsque la donnée est reçue par le SH-HC-08, elle est transmise aux Arduino par le protocole de communication UART (Universal Asynchronous Receiver Transmitter). Ce protocole est un protocole asynchrone, ce qui veut dire que la transmission de données, en communication série, est calqué sur le signal appelé « clock » ou « CLK » qui est généré par la vitesse de transmission « Baud rate » (dans ce cas de 9600Bd). Ce signal est indépendant des deux modules. L'UART utilise également des bits de départ, de parité et de fin qui englobent la donnée. Cette communication nécessite uniquement 2 fils (TX->RX et RX->TX).

Il y a certains avantages à ce protocole :

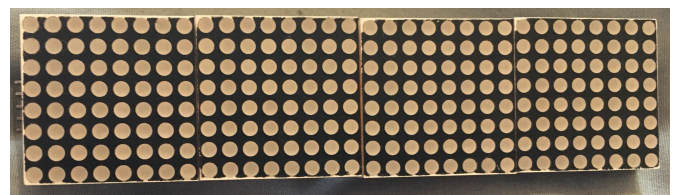
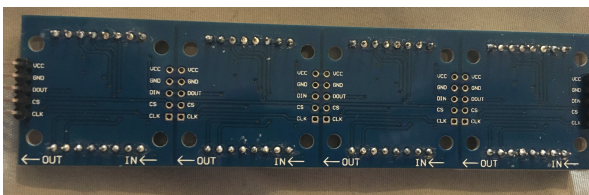
- il n'utilise que 2 fils pour communiquer.
- il n'a pas besoin de signal « clock » partagé.
- il fournit une erreur de détection grâce au bit de parité.
- son prix et sa taille sont faibles grâce à sa communication en série.



L'utilisation d'une bibliothèque a été requise pour faciliter la réception des données Bluetooth depuis les deux Arduino (SoftwareSerial). Celle-ci permet de simuler sur des entrées digitales des pins RX et TX permettant la réception et transmission des données.

L'afficheur à matrice de leds est composé de 4 blocs de 8*8 leds. Ceux-ci sont gérés par des microprocesseurs Max7219. L'utilisation de bibliothèques est préférable pour les contrôler (MD_Parola & MD_MAX72xx). Ces deux bibliothèques permettent d'afficher tout ce que bon nous semble. Par exemple, la luminosité des leds peut être réglée de la valeur 0 à 15. Il est possible d'afficher du texte statique ou défilant, mais également des dessins ou symboles en allant modifier certaines lignes dans les bibliothèques. Chaque élément de la table ascii (de base ainsi qu'étendu) est codé et traduit pour faire en sorte que les bonnes leds s'allument. Un lien GitHub est disponible pour éditer ces codes (<https://pjr.github.io/MDParolaFontEditor>). Cela a été utile pour gérer certains accents et pour ajouter certains dessins. Cependant, que ce soit depuis un Mac ou depuis un autre appareil de la marque Apple, l'envoi de données est crypté. Il y avait donc des parasites en plus lors de la transmission de texte comprenant des symboles ou lettres étant dans la table ascii étendu. Ce point sera expliqué plus en détails dans la partie « Problèmes rencontrés ».

La grande facilité avec cet afficheur est dans sa conception. En effet, il est constitué de 4 modules identiques comprenant son microprocesseur, ces composants internes et une matrice 8*8 de leds qui peuvent se retirer. Chaque module est indépendant des autres et donc un ajout ou une suppression de l'un d'entre eux est toujours possible. Seule une ligne sera à modifier dans le code pour indiquer le nombre de modules présents. Ceux-ci ne doivent pas non plus être collés les uns aux autres pour fonctionner. Une connexion filaire peut ainsi permettre d'ajouter autant de modules que souhaités.



Le protocole de communication des microprocesseurs Max7219 est le SPI (Serial Peripheral Interface). C'est un protocole full-duplex, c'est-à-dire qu'il peut recevoir et transmettre des données en même temps. Le SPI est un protocole de communication qui requiert une synchronisation, il y a donc une horloge commune entre le maître et l'esclave, ce qui lui permet d'être plus rapide que le l'UART ($\pm 16\text{Mbit/s}$ comparé à 1Mbit/s). Le SPI a besoin de 4 fils tandis que l'UART n'en a besoin lui, que de 2. Le plus gros avantage de ce protocole est le fait qu'il se trouve partout, ce qui permet une grande compatibilité.

Cette partie du projet fonctionne parfaitement et sans l'intervention humaine. En effet, un affichage par défaut est défini pour qu'il y ait défilement d'un texte à l'allumage du projet avant même qu'une donnée soit reçue depuis le module Bluetooth. Ceci permet, entre autres, de régler le souci de remise à zéro (lors d'une coupure de courant). Le module peut, avec une batterie de 800mAh complètement chargée, rester théoriquement allumé un peu moins de 8h à luminosité 0. À luminosité 15, pour une même batterie, il ne restera allumé que $\pm 3\text{h}30$. Une amélioration de ce projet peut être envisageable pour augmenter ce temps. Pour se faire, il serait possible de mettre l'Arduino en veille pour que celui-ci consomme moins. Malheureusement, un point resterait à régler dans ce cas : le défilement sur l'afficheur.

Project:

For this project, we needed to use a LoRaWan module who sends data from a slave to a gateway. The slave is commanded by an Arduino Uno and it use a library to make the programming stage easier. Once connected, the gateway throw the data on internet on web site TTN (The Things Network) and it allow the owner of the slave to get the data. LoRa is used in industries to centralize all the data from all sensors and actuators in one same database. It's also used in agriculture to helps farmer and get information about the temperatures or weather for example.

LoRaWan stand for Long Range Wide Area Network it's a LPWAN (Low Power Wide-Area Network) technology. Its means that it is based on a network to have the capability to support more than few slaves. This device work on wide area too, that means that they can be reached from everywhere in the future like AM or FM radio waves. The real difference between AM/FM radio waves and LoRa system is the consumption of the energy. In fact, AM/FM radio waves need a lot of power to emit and in case of IOT (Internet Of Things), we can't afford to use or spend a lot of energy, that will works on battery for years and years. It's why we want them to have a Low Power consumption. LoRaWan is a telecommunication protocol allowing the

The purpose of a utility of LoRa is the low power consumption, the token place to be able to fit into IOT devices (need to be the tinier as possible) and long range communication. Those communications are send by radio waves in license-free on the 868MHz frequency (for Europe). The communication protocol allows the devices to communicate in a circle zone of more than 15Km in rural areas (nothing between transmitter and receiver) and 5Km in urban areas.

In the network, the slaves are not really connected to a gateway, the slave use the them to sends information to internet. It uses a star of stars' systems that means that each server can have a throng of gateways.

The power consumption of LoRa isn't still the same at each transmission. It depends on the width of the frame and the distance to reach. This board contains theoretical values.

	Équipement	Tension (V)	Intensité émission (mA)	Intensité réception (mA)	Intensité attente (mA)	Intensité sommeil (mA)
LoraWan	RN2903 [archive]	3,3	124,4	13,5	2,7	2

The big problem with LoRa is the duty-cycle imposed by the fact that it uses a free radio channels that limits number of frames emitted by a single device. This time is also variable in function of the width of the frames. In fact, big frames will take more time than small ones.

Taille des données ↕	Temps de transmission ↕
20 octets	0,4 sec
20 octets	1,5 sec
40 octets	0,5 sec
40 octets	1,8 sec

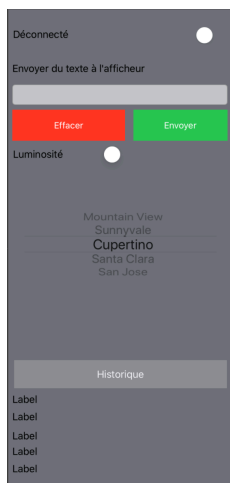
By default, the data received, from the slave to the gateway, is in hexadecimal. I putted some code into TTN/Applications/Payload formats to translate the data in hexadecimal into an ascii value. With this kind of payload, I don't need to format anything when I want them to be displayed.

TTN allow us to have some frames and keep them for a while but not enough for this project. We needed to add a database into TTN/Application/ Integrations to be able to have 7d of data. This database allows us to make some HTTP request to get them (point aborted into "Développement mobile"). This service is called and need an authorization of TTN to works.

Développement mobile :

Ayant eu des cours en Android mais ne possédant qu'un iPhone, l'application mobile fut un réel défi pour moi. La différence de langage de programmation s'est vite fait ressentir. L'option qui s'offrait à moi était de coder sous IOS. J'ai donc utilisé le logiciel Xcode version 11.3, sous environnement Mac, pour coder en Swift 5 pour appareils sous IOS 13.4. Mes connaissances dans ce langage étant nulles, j'ai donc décidé de procéder par la méthode « essais – erreurs ». La première étape fut de comprendre comment ce langage de type « évènementiel » fonctionnait et de m'acclimater à celui-ci.

La seconde étape fut de créer un « Storyboard » qui est l'affichage à l'écran de l'application et d'y ajouter les différents éléments tels que des boutons, des labels, une zone de texte, un menu déroulant, un bouton on-off et enfin un curseur. C'est ici que la décision ainsi que la disposition des éléments à ajouter se fit. À ce stade, tous les éléments présents ne pouvaient pas être affichés à l'écran par manque de place verticale. Une solution, basique en apparence, s'offrait à moi : utiliser une vue à « faire défiler » appelée « ScrollView ». Cet élément pourtant simple me prit ±2 mois pour bien le configurer. Certains points me sont encore aujourd'hui inconnus et mériteraient une amélioration future. J'ai également géré l'orientation du téléphone, ce qui ne fut pas une mince affaire. Certains problèmes sont abordés dans le point « Problèmes rencontrés » de ce rapport.



Application à 800px



*Première partie de
l'application à 675px*



*Seconde partie de
l'application à 675px*

Pour la troisième étape de mon projet, il a fallu coder et relier les différents éléments du « Storyboard » à ce code, pour assigner des noms à des éléments visibles graphiquement dans l'application. Ceci étant fait, les différentes actions de l'application, comme par exemple un appui sur un bouton, font une action bien particulière. Ainsi, lorsqu'on ajoute du texte au champ TextField et qu'on appuie sur le bouton « Effacer », le TextField devient à nouveau vide. J'ai également été capable de notifier l'état du Bluetooth lors d'un changement de celui-ci. Ainsi, lorsqu'on désactive le Bluetooth du téléphone, l'application nous informe que le celui-ci est désactivé et donc que la communication ne pourra pas être réalisée. Cependant, attendant la livraison du module Bluetooth, il m'était difficile de continuer et j'ai donc décidé d'ajouter quelques fonctions pratiques mais non-nécessaires à l'application. Lorsque le clavier apparaissait il m'était impossible de le faire disparaître ; j'y ai donc ajouté cette fonctionnalité. J'ai également ajouté une fonction qui est notifiée lors de la mise en arrière-plan de l'application.

Pour la quatrième étape il a fallu attendre l'arrivée du module Bluetooth SH-HC-08. Une fois réceptionné, j'ai alors, encore une fois, décidé de procéder par la méthode « d'essais-erreurs ». Comme j'avais déjà implémenté une bibliothèque pour vérifier l'état du Bluetooth, j'ai décidé de regarder les fonctions qu'il était possible d'utiliser. Beaucoup de fonctions avec des énoncés différents étaient présentes mais lesquelles utiliser ? Lors de mes nombreuses recherches sur internet, j'ai trouvé certaines pistes de personnes qui récupéraient des données depuis certains appareils Bluetooth mais rien concernant une connexion et un envoi. J'ai donc décidé après plusieurs semaines de tests de faire un appel à l'aide sur StackOverFlow et les réponses furent presque immédiates. Avec l'aide d'un Professeur en Swift de Toronto, avec qui j'ai échangé quelques phrases, il m'est venu la solution. J'ai alors pu commencer à communiquer avec mon module Bluetooth et j'ai pu comprendre comment celui-ci fonctionnait. (<https://stackoverflow.com/questions/59456657/send-informations-with-bluetooth-in-swift-ios-language>)

Send informations with Bluetooth in Swift IOS language

Asked 20 days ago · Active 20 days ago · Viewed 64 times

I'm actually in informatics school and I have some trouble with the connection between my own self signed app coded in Swift(Xcode 11.3 and for IOS 13.2 device) and a BLE module SH-HC-08 plugged on an Arduino. (UUID : FE60E626-C58C-69F5-324F-43C7C409A93D)

I'm totally new into swift language and self-learning this way of thinking in this language.

I'm using CoreBluetooth to detect if the Bluetooth on my iPhone is On or off. I can't actually check if my peripheral is detected, and can't send any information by my app. (don't understand how to configure the methods in CoreBluetooth)

I downloaded apps called *LightBlue* and *BluetoothLE* on app store. The first one give me few information about my Bluetooth device and the second one allow me to send Ascii Strings to my Arduino by my Bluetooth module.

LightBlue information 1:

As I said before, I already :

- Import CoreBluetooth
- Add CBCentralManagerDelegate & CBPeripheralDelegate to my class ViewController
- Instantiate CBCentralManager and CBPeripheral
- Setup the centralManagerDidUpdateState function

CoreBluetooth code:

```
import UIKit
import CoreBluetooth

var textToSend:String?
var test = false

class ViewController: UIViewController, CBCentralManagerDelegate, CBPeripheralDelegate {
    var manager: CBCentralManager?
    var module: CBPeripheral?

    @IBOutlet weak var labelConnection: UILabel!
    @IBOutlet weak var switchConnection: UISwitch!
    @IBOutlet weak var labelTextField: UITextField!
    @IBOutlet weak var labelTest: UILabel!
```

I tried to play with the others methods like *didDiscoverPeripheral* or *didConnected* but I don't understand what I do and can't find any help to send information (preferred in String) by a self made application with bluetooth.

I tried that to discover my module but I don't have any "print" in my logs and don't know if my request was done or not... if I do it well or not too.

```
func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral) {
    print("Name: \(peripheral.name)")
    self.module = peripheral
    self.module.delegate = self

    if(peripheral.name == "SG-HC-08"){
        manager.stopScan()
        manager.connect(self.module, options: nil)
        print(peripheral.name)
    }
}
```

Someone have any experience with that or any help to give it will help me so much. Thanks

La cinquième étape fut la gestion et l'affichage des données depuis la base de données comprise dans TTN. Pour se faire, une requête HTTP de type « GET » avec des champs spécifiques devait être réalisée. Les données dans cette base étaient stockées au format Json et donc également récupérables en ce format, ce qui fut fait. Le stockage de ces données et l'assignation de celles-ci aux labels fut périlleux et une autre demande sur StackOverFlow m'a aidé pour ce point (plus d'explications dans le point « Problèmes rencontrés »). (<https://stackoverflow.com/questions/59488685/text-from-string-to-a-label-text-isnt-working-the-first-time>)

Pour rentrer un peu plus dans les détails de l'application, les points plus complexes furent le fait d'ajouter le « ScrollView » et son bon fonctionnement, la connexion au module Bluetooth et enfin la récupération des Json sur la base de données.

- Pour le « ScrollView », il a fallu que je configure la taille de la vue pour qu'elle soit plus grande que celle de base (800px au lieu de 675px). J'ai également dû « associer » le « ScrollView » à la vue principale.
- La connexion au module Bluetooth se fait en plusieurs étapes, d'abord il y a la découverte des périphériques Bluetooth et la sélection du périphérique souhaité. Ensuite, il y a la découverte des différents services correspondant à ce périphérique, et pour finir la découverte des caractéristiques de ces services et la sélection de la caractéristique voulue. Ici le périphérique est choisi avec son UUID pour plus de fiabilité et la caractéristique voulue est FFE1.
- La récupération de Json depuis la base de données se fait grâce à une requête http avec la méthode GET et deux valeurs de type Headers ajoutées. L'un pour l'autorisation vers TTN et l'autre pour stipuler que la donnée est en Json (ces autorisations sont tirées de la commande Curl accessible sur la base de données).

Problèmes rencontrés :

Systèmes embarqués :

Lors de la conception du projet, pour la partie programmation du microcontrôleur, j'ai scindé le travail en 3 parties différentes pour faciliter le debugg. J'ai donc réalisé une partie pour l'afficheur. La partie pour l'envoi de donnée via le module LoRaWan. Et pour finir la partie réception de la trame grâce au module Bluetooth. Ces trois parties marchait indépendamment les unes des autres. L'afficheur tournait en boucle et quand on changeait le message (dans le code) le message affiché changeait également. Le module LoRaWan envoie bien sa trame (si l'intervalle de temps entre les envois est respecté). Et le module Bluetooth reçoit bien l'information transmise et l'Arduino sait la traduire pour en sortir un message. Le problème survint lorsque l'on assemble les trois parties ensemble. La bibliothèque utilisée pour le module LoRaWan et celle pour l'afficheur prennent ensemble trop de place mémoire pour un Arduino Uno (41510 bytes / 32256 bytes). La première solution fut de passer sous Arduino Méga, qui lui a un espace mémoire beaucoup plus grand (± 256000 bytes). Malheureusement cette solution ne fut pas possible car la bibliothèque utilisée pour commander l'afficheur demandait trop de changement complexe pour être fonctionnelle sur ce type de microcontrôleurs. Deux autres solutions s'offraient à moi. La première étant de pouvoir faire communiquer deux Arduino ensemble et ainsi permettre de faire passer la donnée reçue par le module Bluetooth à l'Arduino ne la recevant pas directement. La seconde, un peu plus facile, consistait à brancher en parallèle les pins RX et TX du module Bluetooth aux deux Arduino et comme cela permettre aux deux de recevoir la donnée. C'est la seconde que j'ai choisie. Ceci étant, cela crée une légère redondance de code dans le projet, les deux Arduino ayant une partie de code identique.

Le second problème réside dans les outils utilisés, je m'explique. J'ai en ma possession un ordinateur et un téléphone de la marque Apple, avec lesquels je travaille. Lors de l'envoi de données, que ce soit dans le code en dur ou par envoi via le module Bluetooth, les données sont « cryptées » et une sécurité Apple est ajoutée aux éléments faisant partie de la table ascii étendue. Ce qui conduit à avoir quelques parasites lors du décodage, par exemple au niveau des accents. De même les éléments de la table ascii étendue n'avait pas tous le même référencement, c'est-à-dire que mes appareils avaient par exemple pour l'élément « é » l'emplacement 215, alors que pour l'afficheur le « é » était codé sur l'emplacement 233. Pour pallier à ces problèmes, j'ai donc essayé de traiter les trames, au format « String », reçues et en détecter les différents éléments connus. Ainsi, les « é » sont remplacés par la chaîne de caractère situé à l'emplacement 233 de la table ascii étendue de l'afficheur et les parasites sont enlevés.

Project :

The first problem I've encountered is that the LoRa library was so difficult to setup on a STM32 and the idea to make the project on this microprocessor was abandoned. So, we used an Arduino Uno to be able to use the libraries (Hal/Hal and Lmic). The configuration of those libraries was difficult but with the help of some professors (Mr. Michaux) we succeed to send a frame with the LoRa module.

Another problem I had with LoRa was his gateway. Professors gave me a gateway to be able to try me code at home. The configuration of the gateway was already made but I still can't send frames to TTN. With some patience, I succeed to make the "connection" to TTN.

Last but not least, when I send frames from the slave to the gateway only 1 on 3 frames can reach the gateway or can be seen by TTN. This problem is maybe due by the time between 2 frames or a duplication of this frame into the gateway.

Développement mobile :

En réglant le premier problème du scroll dans l'application Swift, un autre m'est apparu. Le fait de pouvoir tourner son écran et avoir une interface pouvant s'adapter à l'orientation du téléphone. Ne réussissant pas à régler ce problème, l'idée de faire plusieurs pages m'est venue à l'esprit. Vu que la détection d'orientation était faisable pourquoi ne pas changer de page lorsque l'appareil tournait. Cette idée fut vite abandonnée quand j'ai réalisé les problèmes d'affichage que cela pourrait amener. Il m'a fallu quelques temps pour comprendre et réaliser que les contraintes sur les éléments pouvaient aider au bon fonctionnement de ce point et ainsi ne pas devoir changer l'orientation des éléments mais plutôt la place prise sur l'écran.

Un autre problème mineur rencontré : Lorsque le bouton On-Off était sur On et que l'on plaçait l'application en arrière-plan, lors du retour sur l'application, le bouton On-Off avait gardé son état « On » alors que le label en regard de ce bouton affichait « Déconnecté ». J'ai donc pallier ce problème en positionnant le bouton On-Off sur la position Off lors de la mise en arrière-plan de l'application. Ce qui engendre une déconnection du module Bluetooth.

Lors de mes essais établir la connexion entre mon application et mon module Bluetooth, je ne savais quels étaient les étapes à suivre et je ne trouvais rien expliquant cela sur internet. Plusieurs tutoriels étaient disponibles pour avoir accès à des informations sur un module Bluetooth tels qu'un bracelet connecté mais rien pour l'envoi sur un module externe. La solution, après de nombreux essais sans succès, m'a été apportée lors d'une discussion à ce sujet sur StackOverFlow sur un topic que j'ai créé.

J'ai tout d'abord décidé de placer les données reçues par la requête http dans une variable de type « String » et d'en scinder les éléments. De fait, il était facilement possible de récupérer les cinq dernières trames depuis cette variable et les formater pour en récupérer que les données voulues tels que le temps et le message reçu. Ce procédé demande énormément de ressources de calcul au processeur et donc génère des latences à la génération de l'application sur mon ordinateur ainsi qu'au lancement sur l'appareil mobile. La solution fut de récupérer les informations au format « Json » et comme cela, pouvoir différencier les différentes informations dans la trame et les récupérer sans trop de calcul de la part du processeur.

Une fois les données récupérées en Json et stockées dans un tableau, pour les trier dans le sens voulu, il m'a été impossible de les assigner à un Label se trouvant dans mon application. Il s'avérait que ces changements se faisaient en second plan dans l'application (sur un autre thread) et non sur le plan principal de l'application et ce changement ne pouvait n'être réalisé que sur le premier plan de l'application. La solution apportée fut de repasser cette action en avant plan de l'application (sur le thread principal).

Enfin un problème majeur mais sans solutions viable à l'heure d'aujourd'hui est que 'Apple Développement' offre la possibilité d'auto-signer les applications créées en mode 'Non développeur' ce qui offre la possibilité d'installer l'application créée sur son appareil mobile et que cette application tourne correctement. Cependant dans ce mode 'Non développeur' l'application nécessite une confirmation de certificat tous les 7 jours. Pour pouvoir passer en mode 'Développeur' et ainsi ne devoir confirmer le certificat que tous les 6 mois, un paiement de 150€ à l'année est demandé, ce qui m'est alors impossible de payer alors que je ne suis que simple étudiant.

Projets personnels :

La conception d'une E-Penne, comme j'aime l'appeler, fut possible grâce à ce projet et à l'aval des professeurs sur celui-ci. Effectivement comme précisé plus tôt, ce projet scolaire m'est venu d'une idée de penne connectée. Les différents problèmes à ce projet sont la place demandée par les composants car cela ne peut pas être dérangeant porté sur la tête. Et également ma sécurité car l'E-Penne sera portée lors de sorties et celle-ci peut être mouillée ou bousculée. Ce problème étant d'autant plus grave vu que L'E-Penne contiendra une batterie de type Li-Po pouvant prendre feu ou exploser dans les cas les plus extrêmes. Pour ce faire j'ai donc commandé des Arduino Pro-Mini pour diminuer la place prise dans L'E-Penne et j'ai sécurisé un minimum la batterie avec un module de charge adéquat.

L'autre projet personnel sera réalisé dans un futur proche. Possédant un système domotique DIY, gérant certains éléments de l'installation électrique tel que les lumières et volets, à la maison, nous avons eu dans l'idée mon papa et moi-même de connecter un Arduino aux entrées de l'automate programmable pour pouvoir communiquer avec ce dernier à moindre frais (ne nécessitant donc pas l'achat d'une carte externe compatible avec l'automate S7 300). Pour se faire on commencera par communiquer avec l'Arduino grâce à un module Bluetooth connecté à ce dernier et une application réalisée pour IOS étant tout deux détenteurs d'IPhones.

Conclusion générale :

La conception de ce projet m'a vraiment attiré, et ce projet m'a permis de réaliser entre autres mon E-Penne. Le fait que je puisse réaliser un projet personnel dans le cadre de mes cours fut un gros point positif à ce travail et cela m'a permis de m'impliquer davantage dans sa réalisation.

Je suis également assez fier de moi car j'ai su, notamment pour l'application sous IOS, partir de rien et me documenter et arriver à avoir quelque chose de fonctionnel. Avant de commencer à coder sous Swift je n'y connaissais rien et en essayant, en me documentant, en posant les bonnes questions aux bons endroits j'ai réussi à faire quelque chose de fonctionnel et j'en suis fier. Certes l'application n'est pas forcément faite de manière professionnelle et mériterait certaines améliorations, mais je trouve avoir grandement évoluer notamment en Swift mais aussi pour d'autres choses de ma vie future car ce n'est pas parce qu'on y connaît rien ou pas grand-chose qu'on ne peut pas y arriver en s'y prenant de la bonne façon.

Lexique :

B.L.E.	Bluetooth Low Energy
G.F.S.K.	Gaussian Frequency-Shift Keying
U.A.R.T.	Universal Asynchronous Receiver Transmitter
S.P.I.	Serial Peripheral Interface
L.P.W.A.N.	Low Power Wide-Area Network
I.O.T.	Internet Of Things
T.T.N.	The Things Network

Annexes :

Code présent sur l'Arduino Pro-Mini :

```
1  #include <MD_Parola.h>
2  #include <MD_MAX72xx.h>
3  #include <SPI.h>
4  #include <SoftwareSerial.h>
5
6  SoftwareSerial mySerial(3, 2); // coté arduino vert(RX), bleu(TX)  => coté BLE vert(TX), bleu(RX)
7
8  #define CLK_PIN  13
9  #define DATA_PIN 11
10 #define CS_PIN   10
11
12 char buffer[100];
13 int charsRead;
14
15 MD_Parola P = MD_Parola(MD_MAX72XX::FC16_HW, CS_PIN, 4);
16
17 String messageEnvoye = "Judas ISIH Charleroi %%%";
18 String stringTest;
19 int intensity = 0;
20
21 // Global message buffers shared by Serial and Scrolling functions
22 #define BUF_SIZE 75
23 char curMessage[BUF_SIZE] = { "" };
24 char newMessage[BUF_SIZE] = { "" };
25 bool newMessageAvailable = true;
26
27 String getValue(String data, char separator, int index)
28 {
29     int found = 0;
30     int strIndex[] = { 0, -1 };
31     int maxIndex = data.length()-1;
32
33     for(int i=0; i<=maxIndex && found<=index; i++){
34         if(data.charAt(i)==separator || i==maxIndex){
35             found++;
36             strIndex[0] = strIndex[1]+1;
37             strIndex[1] = (i == maxIndex) ? i+1 : i;
38         }
39     }
40     return found>index ? data.substring(strIndex[0], strIndex[1]) : "";
41 }
42
43 void setup()
44 {
45     pinMode(4,OUTPUT);
46     pinMode(5,OUTPUT);
47     digitalWrite(4,LOW);
48     digitalWrite(5,HIGH);
49
50     Serial.begin(9600);
51     mySerial.begin(9600);
52
53     P.begin();
54     P.setIntensity(intensity);
55     P.displayText(curMessage, PA_LEFT, 60, 0, PA_SCROLL_LEFT, PA_SCROLL_LEFT);
56 }
57
58 void loop()
59 {
60     if(mySerial.available() > 0)
61     {
62         charsRead = mySerial.readBytesUntil('\n', buffer, sizeof(buffer)-1);
63         buffer[charsRead] = '\0';
64         stringTest = buffer ;
65         messageEnvoye = getValue(stringTest, '', 0);
66         intensity = getValue(stringTest, '', 1).toInt();
67         P.setIntensity(intensity);
68         Serial.println(stringTest);
69     }
70     P.setIntensity(intensity);
71     if(messageEnvoye.indexOf('') > -1)
72     {
73         messageEnvoye.replace(' ', '\n');
74         int i = messageEnvoye.indexOf('\n');
75         while(messageEnvoye.indexOf('\n', i) > -1)
76         {
77             i = messageEnvoye.indexOf('\n', i);
78             Serial.println(messageEnvoye);
79             messageEnvoye.remove(i-2, 2);
80             i++;
81         }
82     }
83     if(messageEnvoye.indexOf("♥") > -1)
84     {
85         messageEnvoye.replace('♥', char(149));
86         //messageEnvoye.replace(' ', char(0));
87         int i = messageEnvoye.indexOf(char(149));
88         while(messageEnvoye.indexOf(char(149), i) > -1)
89         {
90             messageEnvoye.remove(i-2, 1);
91             messageEnvoye.remove(i, 3);
92             i = messageEnvoye.indexOf(char(149), i);
93             Serial.println(messageEnvoye);
94         }
95     }
96     if(messageEnvoye.indexOf('à') > -1)
```

```

96     if(messageEnvoye.indexOf('à') > -1)
97     {
98         messageEnvoye.replace('à', char(224));
99         int i =messageEnvoye.indexOf(char(224));
100         while(messageEnvoye.indexOf(char(224),i) > -1)
101         {
102             i =messageEnvoye.indexOf(char(224),i);
103             messageEnvoye.remove(i-1,1);
104             i++;
105             Serial.println(messageEnvoye);
106         }
107         messageEnvoye.remove(messageEnvoye.length()-2,2);
108     }
109     messageEnvoye.replace('é',char(233));
110     messageEnvoye.replace('è',char(232));
111     messageEnvoye.toCharArray(newMessage,BUF_SIZE);
112     if (P.displayAnimate())
113     {
114         if (newMessageAvailable)
115         {
116             strcpy(curMessage, newMessage);
117             newMessageAvailable = false;
118         }
119         P.displayReset();
120     }
121     messageEnvoye.toCharArray(curMessage,BUF_SIZE);
122 }
123

```

Code présent sur l'Arduino UNO :

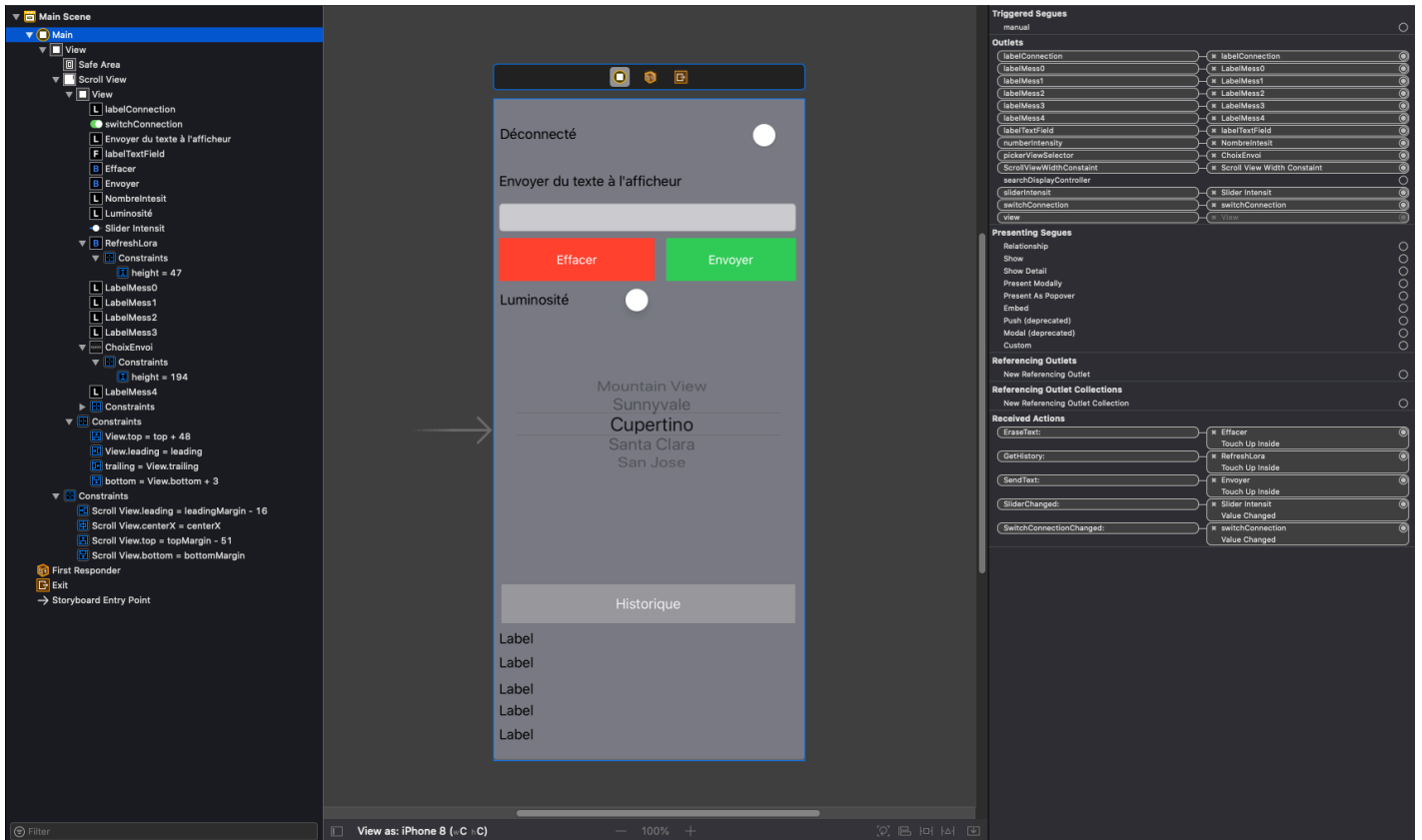
```

1  #include <Arduino.h>
2  #include <lmic.h>
3  #include <hal/nal.h>
4  #include <SPI.h>
5
6  #include <SoftwareSerial.h>
7
8  SoftwareSerial mySerial(4, 3); // côté arduino noir(RX), blanc(TX)    => côté BLE noir(TX), blanc(RX)
9
10 char buffer[100];
11 int charsRead;
12
13 static const PROGMEM u1_t NwksKey[16] = { 0xF7, 0xA7, 0x9B, 0x57, 0xA1, 0x87, 0xE3, 0x87, 0x4D, 0xBF, 0x91, 0x74, 0x8A, 0x9D, 0xE8, 0x22 };
14 static const u1_t PROGMEM AppSKey[16] = { 0x65, 0xF3, 0x5E, 0xCD, 0xAE, 0x34, 0xC9, 0x97, 0x88, 0x94, 0x24, 0xEC, 0x82, 0x55, 0xE9, 0xA8 };
15 static const u1_t DevAddr = 0x2601717;
16
17 void os_getArtEui (u1_t* buf) { };
18 void os_getDevEui (u1_t* buf) { };
19 void os_getDevKey (u1_t* buf) { };
20
21 String messageEnvoye;
22
23 const lmic_pinmap lmic_pins = {
24     .nss = 10,
25     .rxtx = LMIC_UNUSED_PIN,
26     .rst = 9,
27     .dio = {2, 6, 7},
28 };
29 void onEvent (ev_t ev) {
30     Serial.println(os_getTime());
31     Serial.print(" ");
32     switch(ev) {
33         case EV_TXCOMPLETE:
34             Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
35             if(LMIC.dataLen) {
36                 // data received in rx slot after tx
37                 Serial.print(F("Data Received: "));
38                 Serial.write(LMIC.frame+LMIC.dataBeg, LMIC.dataLen);
39                 Serial.println();
40             }
41             break;
42             default:
43                 Serial.println(F("Unknown event"));
44                 break;
45     }
46 }
47
48 void setup() {
49     mySerial.begin(9600);
50
51     Serial.begin(9600);
52     Serial.println(F("Starting"));
53     os_init();
54     LMIC_reset();
55     #ifdef PROGMEM
56     uint8_t appskKey[sizeof(APPKEY)];
57     uint8_t nwksKey[sizeof(NWKSKEY)];
58     memcpy_P(appskKey, APPKEY, sizeof(APPKEY));
59     memcpy_P(nwksKey, NWKSKEY, sizeof(NWKSKEY));
60     LMIC_setSession (0x1, DevAddr, nwksKey, appskKey);
61     #else
62     LMIC_setSession (0x1, DevAddr, NWKSKEY, APPKEY);
63     #endif
64     LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
65     LMIC_setupChannel(1, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
66     LMIC_setupChannel(2, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7), BAND_CENTI); // g-band
67     pinMode(0, INPUT);
68 }
69
70 void loop() {
71     os_runloop_once();
72     if(mySerial.available() > 0)
73     {
74         charsRead = mySerial.readBytesUntil('\n', buffer, sizeof(buffer)-1);
75         buffer[charsRead] = '\0';
76         messageEnvoye = buffer ;
77         int sizeString = (messageEnvoye.length()+1);
78         uint8_t mydata[sizeString];
79         Serial.println(messageEnvoye);
80         messageEnvoye.getBytes(mydata,sizeString);
81         LMIC_setTxData2(1, mydata, sizeString-1, 0);
82         delay(500);
83     }
84 }
85

```

Code présent sur l'iPhone :

Fichier Main.storyboard



Fichier ViewController.swift :

```

1 import UIKit
2 import CoreBluetooth
3 import SystemConfiguration
4
5 class ViewController: UIViewController, CBCentralManagerDelegate, CBPeripheralDelegate, UIPickerViewDelegate, UIPickerViewDataSource {
6
7     //*****VARIABLE*****
8
9     let reachability = SCNetworkReachabilityCreateWithName(nil, "www.google.com")
10    var gotConnection : Bool = false
11
12    var manager : CBCentralManager!
13    var testBluetooth : Bool = true
14    var module : CBPeripheral!
15    var characteristic : CBCharacteristic!
16
17    var textToSend : String!
18    var intensity : Int = 0
19
20    var tableau : [String]=[]
21
22    var timer : Timer!
23    var timeLeft : Int = 5
24
25    let definedSentences = ["HELHa Charleroi.",
26                            "Section Informatique.",
27                            "Les dessins : %❤️",
28                            "Je peux avoir plusieurs phrases ici."
29                            ]
30
31
32    struct Donnee: Decodable {
33        let device_id : String
34        let raw : String
35        let receivedString : String
36        let time : String
37
38        init(json : [String: Any]){
39            device_id = json["device_id"] as? String ?? " "
40            raw = json["raw"] as? String ?? " "
41            receivedString = json["receivedString"] as? String ?? " "
42            time = json["time"] as? String ?? " "
43        }
44    }
45
46    //*****OUTLETS*****
47
48    @IBOutlet weak var labelConnection: UILabel!
49    @IBOutlet weak var switchConnection: UISwitch!
50    @IBOutlet weak var labelTextField: UITextField!
51    @IBOutlet weak var numberIntensity: UILabel!
52    @IBOutlet weak var sliderIntensity: UISlider!
53    @IBOutlet weak var pickerViewSelector: UIPickerView!
54    @IBOutlet weak var ScrollViewWidthConstraint: NSLayoutConstraint!
55
56    @IBOutlet weak var labelMess0: UILabel!
57    @IBOutlet weak var labelMess1: UILabel!
58    @IBOutlet weak var labelMess2: UILabel!
59    @IBOutlet weak var labelMess3: UILabel!
60    @IBOutlet weak var labelMess4: UILabel!
61
62    //*****FUNCTIONS*****
63
64
65    override func viewDidLoad() {
66        super.viewDidLoad()
67
68        manager = CBCentralManager(delegate: self, queue: nil)
69
70        pickerViewSelector.dataSource = self
71        pickerViewSelector.delegate = self
72
73        refresh()
74
75        self.hideKeyboardWhenTappedAround()
76
77        let notificationCenter = NotificationCenter.default
78        notificationCenter.addObserver(self, selector: #selector(appMovedToBackground), name:
79            UIApplication.willResignActiveNotification, object: nil)
80    }
81
82    @objc func appMovedToBackground() {
83        if(module != nil && module.state == .connected)
84        {
85            manager.cancelPeripheralConnection(module)
86        }
87    }
88
89    func refresh(){
90
91        if testBluetooth && switchConnection.isOn{
92            if module != nil && module.state == .connected{
93                labelConnection.text = "Connecté à \"(module.name ?? \"Default\")"
94            }else{
95                manager.scanForPeripherals(withServices: nil, options: nil)
96            }
97        }else{
98            if manager.isScanning{
99                manager.stopScan()
100            }
101            if module != nil && module.state == .connected{
102                manager.cancelPeripheralConnection(module)
103            }
104            labelConnection.text = "Déconnecté"
105            switchConnection.isOn = false
106            numberIntensity.text = "0"
107        }
108    }
109

```

```

107         numberIntensity.text = "0"
108         sliderIntensity.setValue(0, animated: false)
109     }
110 }
111
112 override func didRotate(from fromInterfaceOrientation: UIInterfaceOrientation) {
113     ScrollViewWidthConstraint.constant = UIScreen.main.bounds.size.width
114     refresh()
115 }
116
117 @objc func onTimeFires(){
118
119     timeLeft -= 1
120     if timeLeft <= 0{
121         timer?.invalidate()
122         timer = nil
123
124         switchConnection.isOn = false
125         refresh()
126
127         let alertController = UIAlertController(title: "Bluetooth", message: "Le module Bluetooth est introuvable. Veuillez
128             l'allumer.", preferredStyle: UIAlertController.Style.alert)
129         alertController.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil ))
130         present(alertController, animated: true, completion: nil)
131
132         timeLeft = 5
133     }
134 }
135
136 //BLUETOOTH
137 func centralManagerDidUpdateState(_ central: CBCentralManager) {
138
139     if central.state == .poweredOff{
140         let alertController = UIAlertController(title: "Bluetooth", message: "Le Bluetooth est désactivé, veuillez l'activer et
141             vous connecter au module.", preferredStyle: UIAlertController.Style.alert)
142         alertController.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil ))
143         present(alertController, animated: true, completion: nil)
144
145         testBluetooth = false
146     }
147
148     if central.state == .poweredOn{
149         testBluetooth = true
150     }
151     refresh()
152 }
153
154 func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi
155     RSSI: NSNumber) {
156
157     timer = Timer.scheduledTimer(timeInterval: 1.0, target: self, selector: #selector(onTimeFires), userInfo: nil, repeats: false)
158     timer!.tolerance = 0.4
159
160     peripheral.delegate = self
161     labelConnection.text = "Connection au module..."
162
163     if peripheral.identifier == UUID.init(uuidString: "FE60E626-C58C-69F5-324F-43C7C409A93D") && peripheral.name == "SH-HC-08"{
164         manager.stopScan()
165         module = peripheral
166         timer?.invalidate()
167         timer = nil
168         manager.connect(module, options: nil)
169     }
170 }
171
172 func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
173     module.discoverServices(nil)
174     refresh()
175 }
176
177 func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
178
179     if let servicePeripherals = peripheral.services as [CBService]?{
180         for service in servicePeripherals{
181             peripheral.discoverCharacteristics(nil, for: service)
182         }
183     }
184 }
185
186 func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
187     if let characterArray = service.characteristics as [CBCharacteristic]?{
188         for character in characterArray{
189             if character.uuid.uuidString == "FFE1"{
190                 characteristic = character
191             }
192         }
193     }
194 }
195
196 func centralManager(_ central: CBCentralManager, didDisconnectPeripheral peripheral: CBPeripheral, error: Error?) {
197
198     let alertController = UIAlertController(title: "Bluetooth", message: "Le module à été déconnecté. Veuillez vous y
199         reconnecter.", preferredStyle: UIAlertController.Style.alert)
200     alertController.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil ))
201     present(alertController, animated: true, completion: nil)
202     switchConnection.isOn = false
203     refresh()
204 }
205
206 //NETWORK
207 private func checkReachable(){
208     var flags = SCNetworkReachabilityFlags()
209     SCNetworkReachabilityGetFlags(self.reachability!, &flags)
210
211     if (isNetworkReachable(with : flags))
212     {
213         gotConnection = true
214     }else if (isNetworkReachable(with : flags)){
215         gotConnection = false
216     }
217 }

```

```

212 private func isNetworkReachable (with flags: SCNetworkReachabilityFlags) -> Bool {
213     let isReachable = flags.contains(.reachable)
214     let needsConnection = flags.contains(.connectionRequired)
215     let canConnectAutomatically = flags.contains(.connectionOnDemand) || flags.contains(.connectionOnTraffic)
216     let canConnectWithoutUserInteraction = canConnectAutomatically && !flags.contains(.interventionRequired)
217
218     return isReachable && (!needsConnection || canConnectWithoutUserInteraction)
219 }
220
221 //****ACTIONS*****
222
223 @IBAction func SwitchConnectionChanged(_ sender: UISwitch) {
224     if sender.isOn
225     { if !testBluetooth
226         {
227             let alertController = UIAlertController(title: "Bluetooth", message: "Le Bluetooth est désactivé, pour vous connecter
228             au module, il doit être activé.", preferredStyle: UIAlertController.Style.alert)
229             alertController.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil ))
230             present(alertController, animated: true, completion: nil)
231
232             if manager.isScanning{
233                 manager.stopScan()
234             }
235             if module != nil && module.state == .connected{
236                 manager.cancelPeripheralConnection(module)
237             }
238         }
239     }else{
240         let alertController = UIAlertController(title: "Bluetooth", message: "Module Bluetooth déconnecté.", preferredStyle:
241         UIAlertController.Style.alert)
242         alertController.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil ))
243         present(alertController, animated: true, completion: nil)
244
245         if manager.isScanning{
246             manager.stopScan()
247         }
248         if module != nil && module.state == .connected{
249             manager.cancelPeripheralConnection(module)
250         }
251     }
252     refresh()
253 }
254
255 @IBAction func SliderChanged(_ sender: UISlider ) {
256     if module != nil && module.state == .connected{
257         intensity = Int(sender.value)
258         numberIntensity.text = "\(intensity)"
259     }else{
260         sliderIntensity.setValue(0, animated: false)
261     }
262 }
263
264 @IBAction func EraseText(_ sender: Any) {
265     labelTextField.text = ""
266 }
267
268 @IBAction func SendText(_ sender: Any) {
269     if module != nil && module.state == .connected {
270         textToSend = "\(labelTextField.text ?? "Défaut") " "\(intensity)"
271         if labelTextField.text != ""{
272             let valueString = (textToSend!).data(using: String.Encoding(rawValue: String.Encoding.utf8.rawValue))
273             module.writeValue(valueString!, for: characteristic, type: CBCharacteristicWriteType.withoutResponse)
274         }
275     }
276 }
277
278 @IBAction func GetHistory(_ sender: Any) {
279     checkReachable()
280
281     // Send HTTP GET Request
282
283     // Define server side script URL
284     let scriptUrl = "https://lora-thomas-slave.data.thethingsnetwork.org/api/v2/query?last=7d"
285
286     // Create NSURL Ibject
287     let myUrl = NSURL(string: scriptUrl);
288
289     // Create URL Request
290     let request = NSMutableURLRequest(url:myUrl! as URL);
291
292     // Set request HTTP method to GET. It could be POST as well
293     request.httpMethod = "GET"
294
295     // Or it could be a single Authorization Token value
296     request.addValue("application/json", forHTTPHeaderField: "Accept")
297     request.addValue("key ttn-account-v2.L18nzNblRaeXnN_P0vlyOC0UT222oAaCcka-L2nck7I", forHTTPHeaderField: "Authorization")
298
299     // Execute HTTP Request
300     if gotConnection {
301         URLSession.shared.dataTask(with: request as URLRequest) {(data, response, error) in
302             do{
303                 // Try to assign Json Lora data into an array of Structured data
304                 let donne = try JSONDecoder().decode([Donnee].self, from: data!)
305                 // For each data, print the received string
306                 let compt: Int = donne.count //compt = number of json objects
307                 var j : Int = 0
308                 var chaine: String = " "
309
310                 for i in donne{
311                     if (j > (compt-6)){
312                         chaine = i.time.split(separator: "T")[0] + " " + i.time.split(separator: "T")[1].split(separator: ".")[0] +
313                         " : " + i.receivedString
314                         self.tableau.append(chaine)
315                     }
316                     j += 1
317                 }
318                 DispatchQueue.main.async {

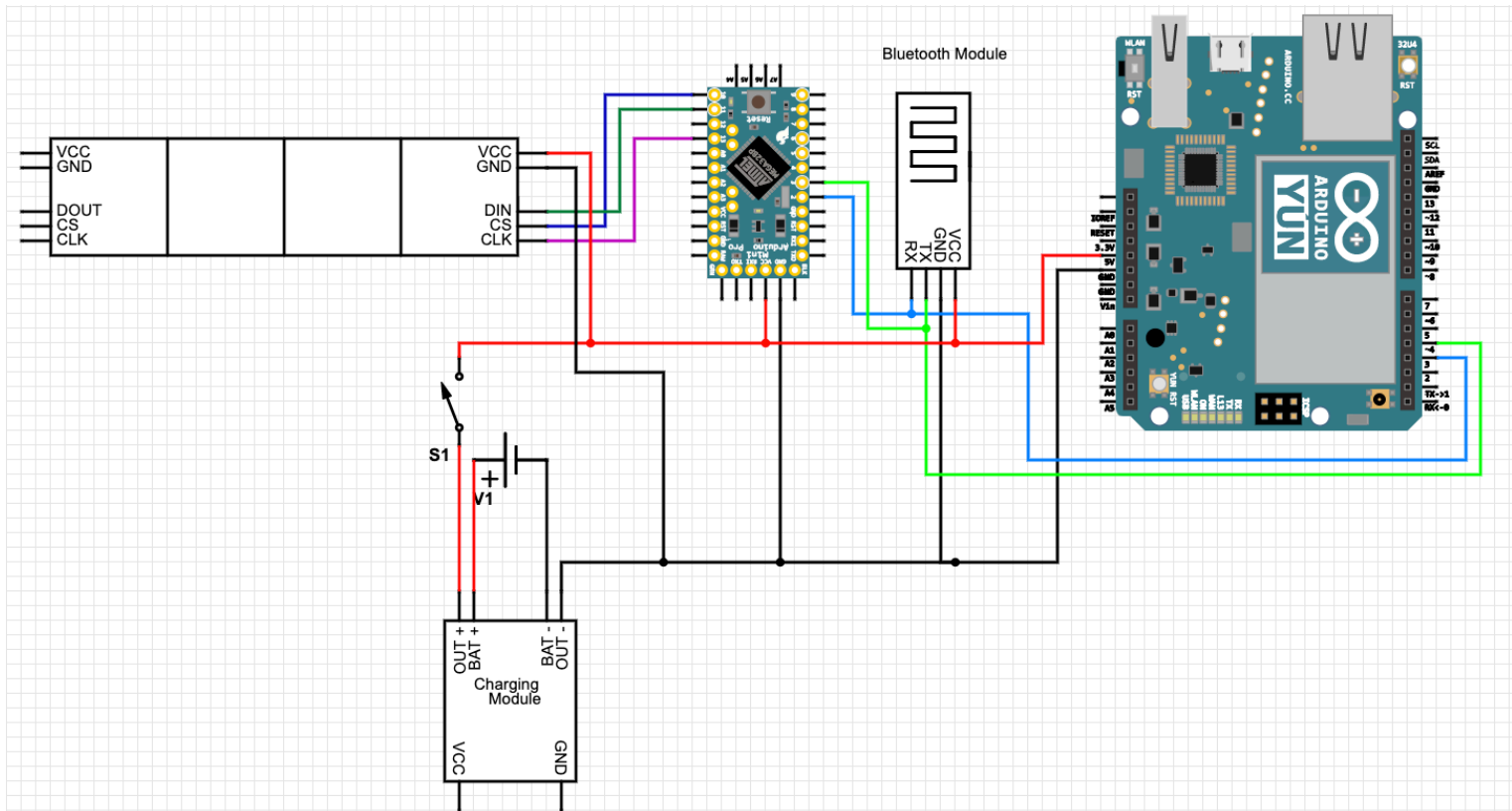
```

```

317         DispatchQueue.main.async {
318             if(self.tableau.count != 0){
319                 self.labelMess0.text = self.tableau[4]
320                 self.labelMess1.text = self.tableau[3]
321                 self.labelMess2.text = self.tableau[2]
322                 self.labelMess3.text = self.tableau[1]
323                 self.labelMess4.text = self.tableau[0]
324             }
325         }
326     }catch{
327     }
328 }
329 }.resume()
330 }else{
331     let alertController = UIAlertController(title: "Historique", message: "Impossible de récupérer l'historique. Veuillez à
332     avoir une connection à internet", preferredStyle: UIAlertController.Style.alert)
333     alertController.addAction(UIAlertAction(title: "OK", style: UIAlertAction.Style.default, handler: nil ))
334     self.present(alertController,animated: true,completion: nil)
335 }
336 self.tableau.removeAll()
337 }
338 }
339 //*****UIPICKER*****
340
341 func numberOfComponents(in pickerView : UIPickerView) -> Int {
342     return 1
343 }
344 func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
345     return definedSentences.count
346 }
347 func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
348     if module != nil && module.state == .connected{
349         labelTextField.text = definedSentences[row]
350     }
351 }
352 func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
353     return definedSentences[row]
354 }
355 }
356 }
357 //*****Make the keyboard disappear*****
358
359 extension UIViewController {
360     func hideKeyboardWhenTappedAround() {
361         let tap: UITapGestureRecognizer = UITapGestureRecognizer(target: self, action: #selector(UIViewController.dismissKeyboard))
362         tap.cancelsTouchesInView = false
363         view.addGestureRecognizer(tap)
364     }
365 }
366 @objc func dismissKeyboard() {
367     view.endEditing(true)
368 }
369 }
370 }

```

Schémas électroniques :



Modifications de librairies Max7219 :

```
219 MD_MAX72XX::fontType_t PROGMEM _sysfont[] =
220 {
221 #if USE_NEW_FONT
222 'F', 2, 0, 0, 0, 255, 8,
223 0, // 0 - 'Empty Cell'
224 5, 62, 91, 79, 91, 62, // 1 - 'Sad Smiley'
225 5, 62, 187, 79, 187, 62, // 2 - 'Happy Smiley'
226 5, 28, 62, 124, 62, 28, // 3 - 'Heart'
227 5, 24, 68, 126, 68, 24, // 4 - 'Diamond'
228 5, 28, 87, 125, 87, 28, // 5 - 'Clubs'
229 5, 28, 94, 127, 94, 28, // 6 - 'Spades'
230 4, 0, 24, 60, 24, // 7 - 'Bullet Point'
231 5, 255, 231, 195, 231, 255, // 8 - 'Rev Bullet Point'
232 4, 0, 24, 36, 24, // 9 - 'Hollow Bullet Point'
233 5, 255, 231, 219, 231, 255, // 10 - 'Rev Hollow BP'
234 5, 48, 72, 58, 6, 14, // 11 - 'Male'
235 5, 38, 41, 121, 41, 38, // 12 - 'Female'
236 5, 64, 127, 5, 5, 7, // 13 - 'Music Note 1'
237 5, 64, 127, 5, 37, 63, // 14 - 'Music Note 2'
238 5, 98, 68, 231, 68, 98, // 15 - 'Snowflake'
239 5, 127, 62, 28, 28, 8, // 16 - 'Right Pointer'
240 5, 8, 28, 28, 62, 127, // 17 - 'Left Pointer'
241 5, 28, 34, 127, 34, 28, // 18 - 'Updown Arrows'
242 5, 255, 255, 255, 255, 255, // 19 - 'Full Block'
243 5, 248, 248, 248, 248, 248, // 20 - 'Half Block Bottom'
244 3, 255, 255, 255, // 21 - 'Half Block LHS'
245 5, 0, 0, 0, 255, 255, // 22 - 'Half Block RHS'
246 5, 15, 15, 15, 15, 15, // 23 - 'Half Block Top'
247 5, 8, 4, 126, 4, 8, // 24 - 'Up Arrow'
248 5, 16, 32, 126, 32, 16, // 25 - 'Down Arrow'
249 5, 8, 8, 42, 28, 8, // 26 - 'Right Arrow'
250 5, 8, 28, 42, 8, 8, // 27 - 'Left Arrow'
251 5, 170, 0, 85, 0, 170, // 28 - '30% shading'
252 5, 170, 85, 170, 85, 170, // 29 - '50% shading'
253 5, 48, 56, 62, 56, 48, // 30 - 'Up Pointer'
254 5, 6, 14, 62, 14, 6, // 31 - 'Down Pointer'
255 2, 0, 0, // 32 - 'Space'
256 1, 95, // 33 - '!'
257 3, 7, 0, 7, // 34 - '***'
258 5, 28, 127, 28, 127, 28, // 35 - '#'
259 8, 224, 168, 254, 69, 69, 254, 168, 224, // 150 petit
260 //5, 68, 74, 255, 74, 58, // 36 - '$'
261 7, 254, 135, 135, 135, 255, 73, 112, // 151 Choppe
262 //5, 99, 19, 8, 188, 99, // 37 - '%'
263 5, 54, 73, 73, 54, 72, // 38 - '&'
264 1, 7, // 39 - '...'
265 3, 62, 65, 65, // 40 - '{'
266 3, 65, 65, 62, // 41 - '}'
267 5, 8, 42, 28, 42, 8, // 42 - '*'
268 5, 8, 8, 62, 8, 8, // 43 - '+'
269 2, 96, 224, // 44 - ','
270 4, 8, 8, 8, 8, // 45 - '-'
271 2, 96, 96, // 46 - '.'
272 5, 96, 16, 8, 4, 3, // 47 - '/'
273 5, 62, 81, 73, 69, 62, // 48 - '0'
274 3, 66, 127, 64, // 49 - '1'
275 5, 113, 73, 73, 73, 70, // 50 - '2'
276 5, 65, 73, 73, 73, 54, // 51 - '3'
277 5, 15, 8, 8, 8, 127, // 52 - '4'
278 5, 79, 73, 73, 73, 49, // 53 - '5'
279 5, 62, 73, 73, 73, 48, // 54 - '6'
280 5, 3, 1, 1, 1, 127, // 55 - '7'
281 5, 54, 73, 73, 73, 54, // 56 - '8'
282 5, 6, 73, 73, 73, 62, // 57 - '9'
283 2, 54, 54, // 58 - ':'
284 2, 188, 236, // 59 - ';'
285 3, 8, 20, 34, // 60 - '<'
286 4, 28, 28, 28, 28, // 61 - '='
287 3, 34, 28, 8, // 62 - '>'
288 5, 1, 89, 9, 9, 6, // 63 - '?'
289 5, 62, 65, 93, 89, 78, // 64 - '@'
290 5, 126, 9, 9, 9, 126, // 65 - 'A'
291 5, 127, 73, 73, 73, 54, // 66 - 'B'
292 5, 62, 65, 65, 65, 65, // 67 - 'C'
293 5, 127, 65, 65, 65, 62, // 68 - 'D'
294 5, 127, 73, 73, 73, 65, // 69 - 'E'
295 5, 127, 9, 9, 9, 1, // 70 - 'F'
296 5, 62, 65, 65, 73, 121, // 71 - 'G'
297 5, 127, 8, 8, 8, 127, // 72 - 'H'
298 3, 65, 127, 65, // 73 - 'I'
299 5, 48, 65, 65, 65, 63, // 74 - 'J'
300 5, 127, 8, 28, 34, 65, // 75 - 'K'
301 5, 127, 64, 64, 64, 64, // 76 - 'L'
302 5, 127, 2, 12, 2, 127, // 77 - 'M'
303 5, 127, 4, 8, 16, 127, // 78 - 'N'
304 5, 62, 65, 65, 65, 62, // 79 - 'O'
305 5, 127, 9, 9, 9, 6, // 80 - 'P'
306 5, 62, 65, 65, 97, 126, // 81 - 'Q'
307 5, 127, 9, 25, 41, 78, // 82 - 'R'
308 5, 78, 73, 73, 73, 49, // 83 - 'S'
309 5, 1, 1, 127, 1, 1, // 84 - 'T'
310 5, 63, 64, 64, 64, 63, // 85 - 'U'
311 5, 31, 32, 64, 32, 31, // 86 - 'V'
312 5, 63, 64, 56, 64, 63, // 87 - 'W'
313 5, 99, 28, 8, 28, 99, // 88 - 'X'
314 5, 3, 4, 128, 4, 3, // 89 - 'Y'
315 5, 97, 81, 73, 69, 67, // 90 - 'Z'
316 3, 127, 65, 65, // 91 - '['
317 5, 3, 4, 8, 16, 96, // 92 - '\'
318 3, 65, 65, 127, // 93 - ']'
319 5, 4, 2, 1, 2, 4, // 94 - '^'
320 4, 128, 128, 128, 128, // 95 - '_'
321 3, 1, 2, 4, // 96 - '...'
322 4, 56, 68, 68, 124, // 97 - 'a'
323 4, 127, 68, 68, 56, // 98 - 'b'
324 4, 56, 68, 68, 68, // 99 - 'c'
325 4, 56, 68, 68, 127, // 100 - 'd'
326 4, 56, 84, 84, 88, // 101 - 'e'
327 4, 4, 126, 8, 1, // 102 - 'f'
328 4, 24, 164, 164, 124, // 103 - 'g'
329 4, 127, 4, 4, 128, // 104 - 'h'
```

```
330 1, 125, // 105 - 'i'
331 3, 132, 133, 124, // 106 - 'j'
332 4, 127, 16, 48, 68, // 107 - 'k'
333 1, 127, // 108 - 'l'
334 5, 124, 4, 128, 4, 128, // 109 - 'm'
335 4, 124, 4, 4, 128, // 110 - 'n'
336 4, 56, 68, 68, 56, // 111 - 'o'
337 4, 252, 36, 36, 24, // 112 - 'p'
338 4, 24, 36, 36, 252, // 113 - 'q'
339 4, 124, 4, 4, 8, // 114 - 'r'
340 4, 88, 84, 84, 52, // 115 - 's'
341 3, 4, 127, 4, // 116 - 't'
342 4, 68, 64, 64, 124, // 117 - 'u'
343 4, 28, 32, 64, 124, // 118 - 'v'
344 5, 68, 64, 48, 64, 68, // 119 - 'w'
345 4, 188, 16, 16, 188, // 120 - 'x'
346 4, 28, 168, 168, 124, // 121 - 'y'
347 4, 188, 84, 84, 76, // 122 - 'z'
348 4, 8, 54, 65, 65, // 123 - '{'
349 1, 127, // 124 - '|'
350 4, 65, 65, 64, 8, // 125 - '}'
351 4, 2, 1, 2, 1, // 126 - '~'
352
353
354
355 5, 127, 65, 65, 65, 127, // 127 - 'Hollow Block'
356 5, 62, 85, 85, 85, 65, // 128 - 'Euro symbol'
357 5, 56, 68, 68, 56, 68, // 129 - 'Alpha'
358 5, 124, 42, 42, 62, 28, // 130 - 'Beta'
359 5, 126, 2, 2, 6, 6, // 131 - 'Gamma'
360 5, 2, 126, 2, 126, 2, // 132 - 'Pi'
361 5, 99, 85, 73, 65, 99, // 133 - 'Sigma'
362 5, 56, 68, 68, 60, 4, // 134 - 'Theta'
363 5, 64, 126, 32, 38, 32, // 135 - 'Mu'
364 5, 6, 2, 126, 2, 2, // 136 - 'Tau'
365 8, 99, 19, 8, 188, 99, 0, 96, 96, // 137 - 'Percent 80'
366 5, 42, 42, 42, 42, 42, // 138 - '3 Bar Equals'
367 3, 81, 74, 68, // 139 - '>='
368 3, 68, 74, 81, // 140 - '<='
369 5, 0, 0, 255, 1, 3, // 141 - 'Top of Integral'
370 3, 224, 128, 255, // 142 - 'Bot of Integral'
371 5, 54, 18, 64, 36, 54, // 143 - 'Wavy ='
372 3, 2, 5, 2, // 144 - 'Degree'
373 2, 24, 24, // 145 - 'Math Product'
374 2, 16, 16, // 146 - 'Short Dash'
375 5, 48, 64, 255, 1, 1, // 147 - 'Square Root'
376 4, 31, 1, 1, 30, // 148 - 'Superscript n'
377 8, 14, 17, 33, 66, 66, 33, 17, 14, // 149 - 'Gors coeur'
378 8, 224, 168, 254, 69, 69, 254, 168, 224, // 150 petit penis
379 8, 0, 252, 254, 254, 254, 84, 112, 0, // 151 Choppe
380 0, // 152
381 9, 1, 1, 127, 1, 127, 2, 12, 2, 127, // 153 - 'Trademark'
382 0, // 154
383 0, // 155
384 0, // 156
385 0, // 157
386 0, // 158
387 0, // 159
388 1, 0, // 160 - 'Non-breaking space'
389 1, 125, // 161 - '¡ Inverted Exclamation Mark'
390 4, 24, 36, 126, 36, // 162 - '¢ Cent sign'
391 4, 68, 126, 69, 65, // 163 - '£ Pound sign'
392 5, 34, 28, 28, 28, 34, // 164 - '¥ Currency sign'
393 5, 1, 42, 124, 42, 1, // 165 - '¥ Yen sign'
394 1, 119, // 166 - '¡ Broken bar'
395 4, 78, 85, 85, 57, // 167 - '§ Section sign'
396 3, 2, 0, 2, // 168 - '¨ Diaeresis (Umlaut)'
397 7, 126, 129, 189, 165, 165, 129, 126, // 169 - '© Copyright sign'
398 3, 38, 41, 47, // 170 - '® Feminine Ordinal Indicator'
399 8, 28, 42, 28, 34, // 171 - '« Left-pointing double angle quotation mark'
400 4, 8, 8, 8, 24, // 172 - '¬ Not sign'
401 3, 8, 8, 8, // 173 - 'Soft hyphen'
402 7, 126, 129, 189, 149, 169, 129, 126, // 174 - '® Registered sign'
403 5, 1, 1, 1, 1, 1, // 175 - '¨ macron'
404 3, 2, 5, 2, // 176 - '° Degree symbol'
405 3, 36, 46, 36, // 177 - '± Plus-minus sign'
406 3, 25, 21, 18, // 178 - '² Superscript two'
407 3, 21, 21, 18, // 179 - '³ Superscript three'
408 2, 2, 1, // 180 - '´ Acute accent'
409 4, 248, 32, 64, 128, // 181 - 'µ Micro sign'
410 5, 6, 9, 127, 1, 127, // 182 - '¶ Pilsrow sign'
411 2, 24, 24, // 183 - '· Middle dot'
412 4, 0, 128, 168, 192, // 184 - '¸ Cedilla'
413 3, 18, 31, 16, // 185 - '¹ Superscript one'
414 3, 38, 41, 38, // 186 - 'º Masculine ordinal indicator'
415 5, 34, 28, 42, 28, 8, // 187 - '» Right-pointing double angle quotation mark'
416 8, 64, 47, 16, 8, 4, 38, 17, 124, // 188 - '¼ Vulgar fraction one quarter'
417 8, 64, 47, 16, 8, 4, 98, 85, 76, // 189 - '½ Vulgar fraction one half'
418 9, 21, 85, 63, 16, 8, 4, 38, 17, 124, // 190 - '¾ Vulgar fraction three quarters'
419 5, 48, 72, 72, 77, 64, // 191 - '¿ Inverted Question Mark'
420 5, 128, 21, 22, 28, 128, // 192 - 'À Latin Capital letter A with grave'
421 5, 128, 28, 22, 21, 128, // 193 - 'Á Latin Capital letter A with acute'
422 0, //5, 122, 21, 28, 21, 122, // 194 - 'Ä Latin Capital letter A with circumflex'
423 0, // 195 - 'Å Latin Capital letter A with tilde'
424 5, 128, 21, 28, 21, 128, // 196 - 'Ä Latin Capital letter A with diaeresis'
425 5, 128, 28, 21, 28, 128, // 197 - 'Å Latin Capital letter A with ring above'
426 7, 126, 9, 9, 127, 73, 73, 65, // 198 - 'Æ Latin Capital letter Æ'
427 5, 158, 161, 97, 33, 33, // 199 - 'Ç Latin Capital letter C with cedilla'
428 5, 124, 84, 85, 78, 68, // 200 - 'È Latin Capital letter E with grave'
429 5, 124, 84, 86, 69, 68, // 201 - 'É Latin Capital letter E with acute'
430 5, 126, 85, 84, 69, 78, // 202 - 'Ê Latin Capital letter E with circumflex'
431 5, 124, 85, 84, 69, 68, // 203 - 'Ë Latin Capital letter E with diaeresis'
432 3, 69, 126, 68, // 204 - 'Ì Latin Capital letter I with grave'
433 3, 68, 126, 69, // 205 - 'Í Latin Capital letter I with acute'
434 3, 78, 125, 78, // 206 - 'Î Latin Capital letter I with circumflex'
435 3, 69, 124, 69, // 207 - 'Ï Latin Capital letter I with diaeresis'
436 5, 8, 127, 73, 65, 62, // 208 - 'Ð Latin Capital letter Eth'
437 5, 124, 18, 17, 34, 125, // 209 - 'Ñ Latin Capital letter N with tilde'
438 5, 56, 68, 69, 78, 56, // 210 - 'Ò Latin Capital letter O with grave'
439 5, 56, 68, 78, 69, 56, // 211 - 'Ó Latin Capital letter O with acute'
440 5, 58, 69, 68, 69, 58, // 212 - 'Ô Latin Capital letter O with circumflex'
```

Datasheets :

Module de chargement :



南京拓微集成电路有限公司
NanJing Top Power ASIC Corp.

TP4056 1A Standalone Linear Li-Ion Battery Charger with Thermal Regulation in SOP-8

DESCRIPTION

The TP4056 is a complete constant-current/constant-voltage linear charger for single cell lithium-ion batteries. Its SOP package and low external component count make the TP4056 ideally suited for portable applications. Furthermore, the TP4056 can work within USB and wall adapter.

No blocking diode is required due to the internal PMOSFET architecture and have prevent to negative Charge Current Circuit. Thermal feedback regulates the charge current to limit the die temperature during high power operation or high ambient temperature. The charge voltage is fixed at 4.2V, and the charge current can be programmed externally with a single resistor. The TP4056 automatically terminates the charge cycle when the charge current drops to 1/10th the programmed value after the final float voltage is reached.

TP4056 Other features include current monitor, under voltage lockout, automatic recharge and two status pin to indicate charge termination and the presence of an input voltage.

FEATURES

- Programmable Charge Current Up to 1000mA
- No MOSFET, Sense Resistor or Blocking Diode Required
- Complete Linear Charger in SOP-8 Package for Single Cell Lithium-Ion Batteries
- Constant-Current/Constant-Voltage
- Charges Single Cell Li-Ion Batteries Directly from USB Port
- Preset 4.2V Charge Voltage with 1.5% Accuracy
- Automatic Recharge
- two Charge Status Output Pins
- C/10 Charge Termination
- 2.9V Trickle Charge Threshold (TP4056)
- Soft-Start Limits Inrush Current
- Available Radiator in 8-Lead SOP Package, the Radiator need connect GND or impeding

PACKAGE/ORDER INFORMATION

TEMP1	8 CE
PROG2	7 CHRG
GND3	6 STDBY
V _{CC} 4	5 BAT
SOP-8	
photo	ORDER PART NUMBER
	TP4056-42-SOP8-PP
	PART MARKING TP4056

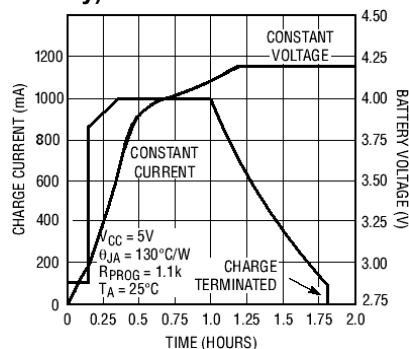
ABSOLUTE MAXIMUM RATINGS

- Input Supply Voltage(V_{CC}): -0.3V~8V
- TEMP: -0.3V~10V
- CE: -0.3V~10V
- BAT Short-Circuit Duration: Continuous
- BAT Pin Current: 1200mA
- PROG Pin Current: 1200uA
- Maximum Junction Temperature: 145°C
- Operating Ambient Temperature Range: -40°C~85°C
- Lead Temp.(Soldering, 10sec): 260°C

APPLICATIONS

- Cellular Telephones, PDAs, GPS
- Charging Docks and Cradles
- Digital Still Cameras, Portable Devices
- USB Bus-Powered Chargers, Chargers

Complete Charge Cycle (1000mAh Battery)



TP4056 1A Standalone Linear Li-Ion Battery Charger with Thermal Regulation in SOP-8

DESCRIPTION

The TP4056 is a complete constant-current/constant-voltage linear charger for single cell lithium-ion batteries. Its SOP package and low external component count make the TP4056 ideally suited for portable applications. Furthermore, the TP4056 can work within USB and wall adapter.

No blocking diode is required due to the internal PMOSFET architecture and have prevent to negative Charge Current Circuit. Thermal feedback regulates the charge current to limit the die temperature during high power operation or high ambient temperature. The charge voltage is fixed at 4.2V, and the charge current can be programmed externally with a single resistor. The TP4056 automatically terminates the charge cycle when the charge current drops to 1/10th the programmed value after the final float voltage is reached.

TP4056 Other features include current monitor, under voltage lockout, automatic recharge and two status pin to indicate charge termination and the presence of an input voltage.

FEATURES

- Programmable Charge Current Up to 1000mA
- No MOSFET, Sense Resistor or Blocking Diode Required
- Complete Linear Charger in SOP-8 Package for Single Cell Lithium-Ion Batteries
- Constant-Current/Constant-Voltage
- Charges Single Cell Li-Ion Batteries Directly from USB Port
- Preset 4.2V Charge Voltage with 1.5% Accuracy
- Automatic Recharge
- two Charge Status Output Pins
- C/10 Charge Termination
- 2.9V Trickle Charge Threshold (TP4056)
- Soft-Start Limits Inrush Current
- Available Radiator in 8-Lead SOP Package, the Radiator need connect GND or impending

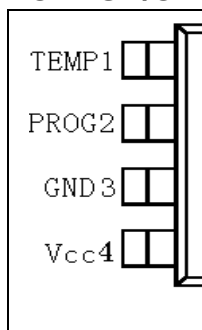
ABSOLUTE MAXIMUM RATINGS

- Input Supply Voltage(V_{CC}): $-0.3V \sim 8V$
- TEMP: $-0.3V \sim 10V$
- CE: $-0.3V \sim 10V$
- BAT Short-Circuit Duration: Continuous
- BAT Pin Current: 1200mA
- PROG Pin Current: 1200uA
- Maximum Junction Temperature: $145^{\circ}C$
- Operating Ambient Temperature Range: $-40^{\circ}C \sim 85^{\circ}C$
- Lead Temp.(Soldering, 10sec): $260^{\circ}C$

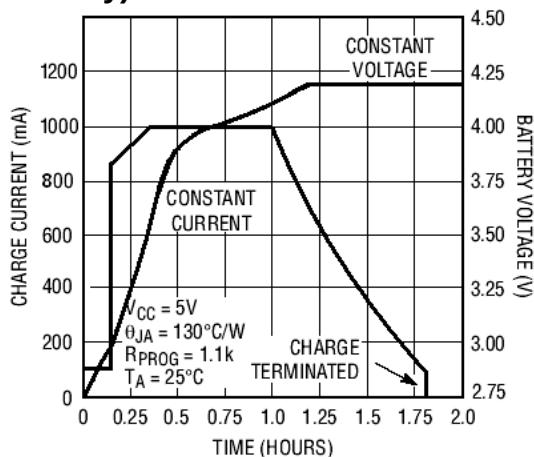
APPLICATIONS

- Cellular Telephones, PDAs, GPS
- Charging Docks and Cradles
- Digital Still Cameras, Portable Devices
- USB Bus-Powered Chargers,Chargers

PACKAGE/ORDER INFORMATION

 TEMP1 PROG2 GND3 V_{CC}4 8 CE 7 CHRG 6 STDBY 5 BAT SOP-8	ORDER PART NUMBER TP4056-42-SOP8-PP PART MARKING TP4056
photo 	

Complete Charge Cycle (1000mAh Battery)



TEMP(Pin 1) :Temperature Sense Input Connecting TEMP pin to NTC thermistor's output in Lithium ion battery pack. If TEMP pin's voltage is below 45% or above 80% of supply voltage V_{IN} for more than 0.15S, this means that battery's temperature is too high or too low, charging is suspended. The temperature sense function can be disabled by grounding the TEMP pin.

PROG(Pin 2): Constant Charge Current Setting and Charge Current Monitor Pin charge current is set by connecting a resistor R_{ISET} from this pin to GND. When in precharge mode, the ISET pin's voltage is regulated to 0.2V. When in constant charge current mode, the ISET pin's voltage is regulated to 2V. In all modes during charging, the voltage on ISET pin can be used to measure the charge current as follows:

GND(Pin3): Ground Terminal

$$I_{BAT} = \frac{V_{PROG}}{R_{PROG}} \times 1200 \quad (V_{PROG}=1V)$$

Vcc(Pin 4): Positive Input Supply Voltage V_{IN} is the power supply to the internal circuit. When V_{IN} drops to within 30mv of the BAT pin voltage, TP4056 enters low power sleep mode, dropping BAT pin's current to less than 2uA.

BAT(Pin5): Battery Connection Pin. Connect the positive terminal of the battery to BAT pin. BAT pin draws less than 2uA current in chip disable mode or in sleep mode. BAT pin provides charge current to the battery and provides regulation voltage of 4.2V.

STDBY(Pin6): Open Drain Charge Status Output When the battery Charge Termination, the $\overline{STDBY}$ pin is pulled low by an internal switch, otherwise $\overline{STDBY}$ pin is in high impedance state.

CHRG (Pin7): Open Drain Charge Status Output When the battery is being charged, the $\overline{CHRG}$ pin is pulled low by an internal switch, otherwise $\overline{CHRG}$ pin is in high impedance state.

CE(Pin8): Chip Enable Input. A high input will put the device in the normal operating mode.

Pulling the CE pin to low level will put the YP4056 into disable mode. The CE pin can be driven by TTL or CMOS logic level.

ELECTRICAL CHARACTERISTICS

The ● denotes specifications which apply over the full operating temperature range, otherwise specifications are at $T_A=25^\circ\text{C}$, $V_{CC}=5V$, unless otherwise noted.

SYMBOL	PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
V_{CC}	Input Supply Voltage		●	4.0	5	8.0	V
I_{CC}	Input Supply Current	Charge Mode, $R_{PROG} = 1.2k$	●		150	500	μA
		StandbyMode(Charge Terminated)	●		55	100	μA
		Shutdown Mode (R_{PROG} Not Connected, $V_{CC} < V_{BAT}$, or $V_{CC} < V_{UV}$)	●		55	100	μA
V_{FLOAL}	Regulated Output (Float) Voltage	$0^\circ\text{C} \leq T_A \leq 85^\circ\text{C}$, $I_{BAT}=40mA$		4.137	4.2	4.263	V
I_{BAT}	BAT Pin Current Text condition: $V_{BAT}=4.0V$	$R_{PROG} = 2.4k$, Current Mode	●	450	500	550	mA
		$R_{PROG} = 1.2k$, Current Mode	●	950	1000	1050	mA
		Standby Mode, $V_{BAT} = 4.2V$	●	0	-2.5	-6	μA
I_{TRIKL}	Trickle Charge Current	$V_{BAT} < V_{TRIKL}$, $R_{PROG}=1.2K$	●	120	130	140	mA
V_{TRIKL}	Trickle Charge Threshold Voltage	$R_{PROG}=1.2K$, V_{BAT} Rising		2.8	2.9	3.0	V
V_{TRHYS}	Trickle Charge Hysteresis Voltage	$R_{PROG}=1.2K$		60	80	100	mV
T_{LIM}	Junction Temperature in Constant Temperature Mode				145		$^\circ\text{C}$

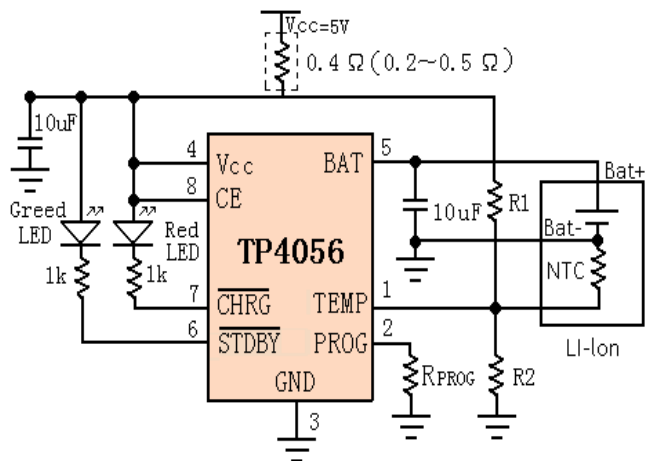
indicator light state

Charge state	Red LED $\overline{\text{CHRG}}$	Green LED $\overline{\text{STDBY}}$
charging	bright	extinguish
Charge Termination	extinguish	bright
Vin too low; Temperature of battery too low or too high; no battery	extinguish	extinguish
BAT PIN Connect 10u Capacitance; No battery	Green LED bright, Red LED Coruscate T=1-4 S	

Rprog Current Setting

R _{PROG} (k)	I _{BAT} (mA)
10	130
5	250
4	300
3	400
2	580
1.66	690
1.5	780
1.33	900
1.2	1000

TYPICAL APPLICATIONS



SH-HC-08 Data Sheet

SH-HC-08 Data Sheet

Bluetooth 4.0 BLE Module

1、 Overview	2
2、 Feature	2
3、 Product's picture	3
4. Application Fields	4
5. Physical Characteristics	4
6. Electrical Characteristics	5
7. Application Circuit Diagram	5
8、 PINs description	6
9. Other configuration	7
10.Layout Announcements	7
11、 AT Command	8

SH-HC-08 Data Sheet

1、 Overview

SH-HC-08 is a next-generation, class 2, Bluetooth 4.0 ble module. It uses TI CC2541 bluetooth 4.0 ble chip .Support the AT command, the user can according to need to change the baud rate of serial port, name of equipment, matching parameters such as password, use agile. SH-HC-08 is a highly integrated and sophisticated Bluetooth module, containing all the necessary elements .Therefore SH-HC-08 provides an ideal solution for developers who want to integrate Bluetooth wireless technology into their designs with limited knowledge of Bluetooth and RF technologies.

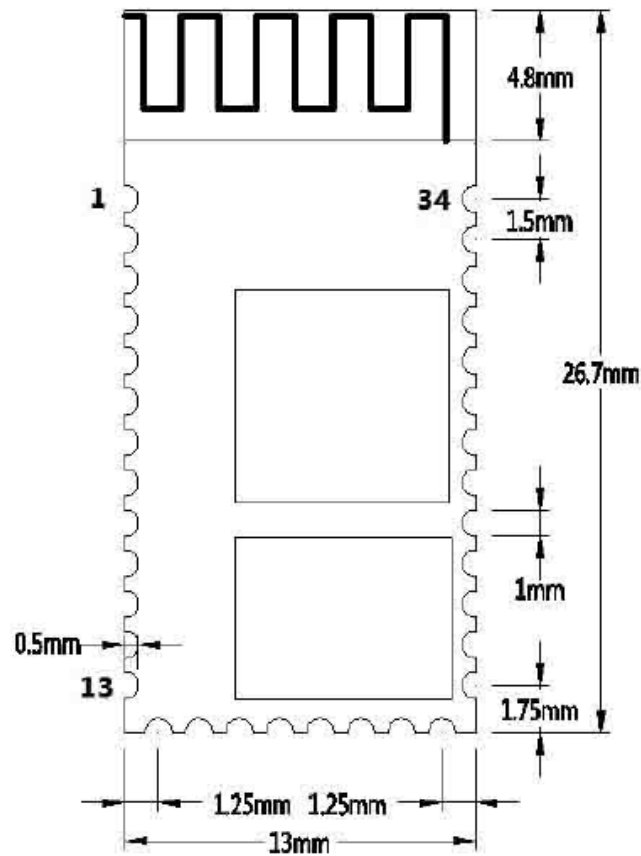
2、 Feature

- Bluetooth protocol: Bluetooth Specification V4.0 BLE,
- open environment and iphone 4 s can achieve 110 meters extreme distance communication.
- Based on TI CC2541 chipset
- Bluetooth class 2
- Industrial level SPP Bluetooth module
- Integrated chip antenna
- Size: 26.7 x 13 x 2 mm
- Industrial temperature range from -40℃ to +85℃
- Support for on-board applications
- Operating frequency: 2.4 GHz ISM band
- Modulation method: GFSK (Gaussian Frequency Shift Keying)
- Acuity: -84 DBM or less BER at 0.1%
- Transfer rate: Asynchronous: 6 KBPS Synchronous: KBPS
- Security features: Authentication and encryption
- Support services: Central and Peripheral UUID FFE0, FFE1

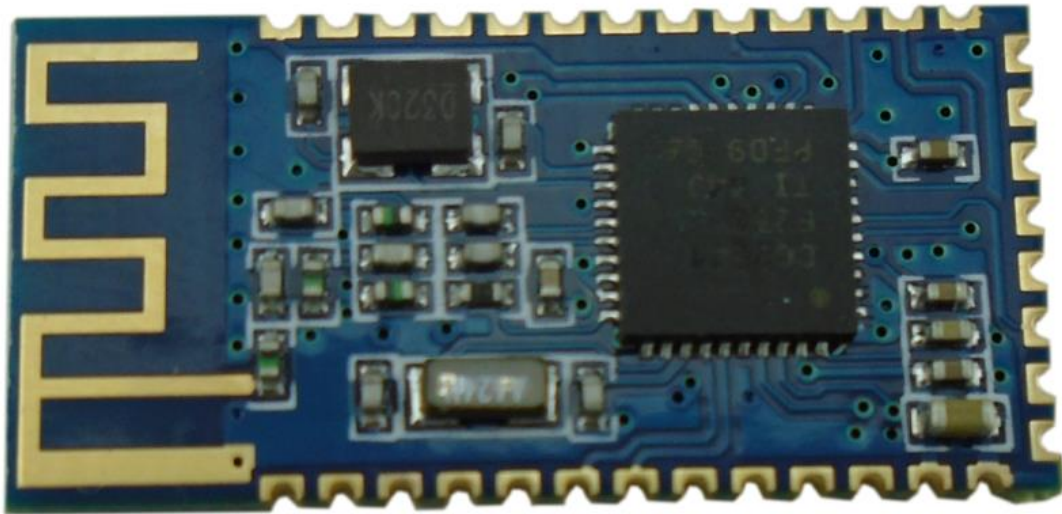
SH-HC-08 Data Sheet

- Power consumption: automatic sleep mode, the standby current 400 ~ 1.5 mA, transfer 8.5 mA.
- Power supply: + 3.3 VDC 50 MA

3、 Product's picture



SH-HC-08 Data Sheet



4. Application Fields

- Cable replacement
- Point-of-sales systems
- Barcode readers and pay terminals
- Telemetry and machine-to-machine devices
- Logistics and transportation systems
- Automotive inspection and measurement systems
- Medical systems
- Fitness and sports telemetry devices
- PDA and other portable terminals
- PCs and laptop
- OBD

5. Physical Characteristics

Operating Frequency Band	2.4GHz ISM band
Bluetooth Specification	BLE4.0 BLE
Output Power Class	Class 2

SH-HC-08 Data Sheet

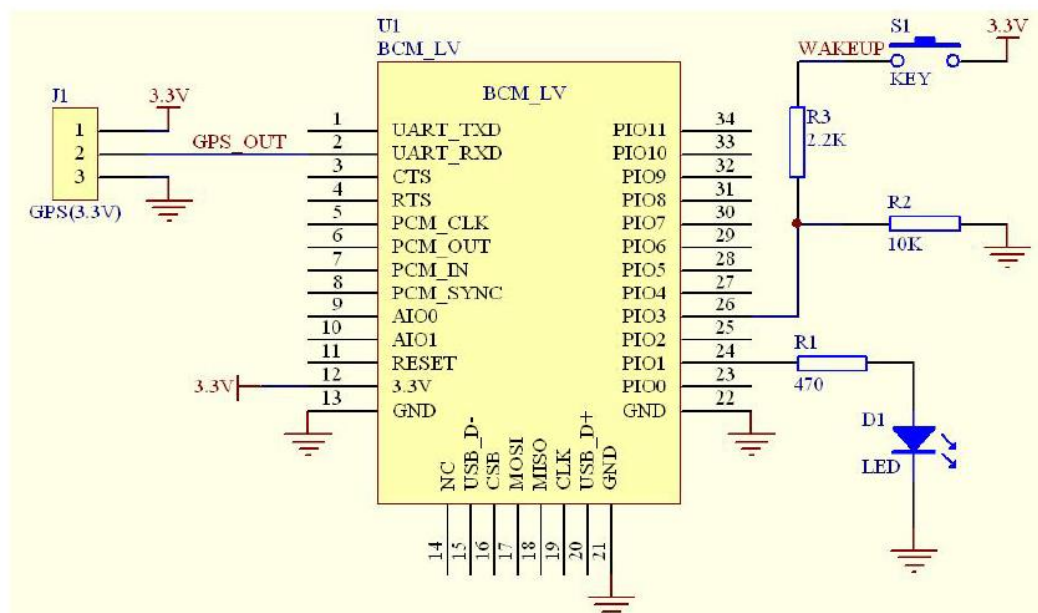
Operating Voltage	3.3V
Host Interface	UART
Dimension	27mm (L) x 13(W) mm x 2mm (H)

6. Electrical Characteristics

Absolute Maximum Ratings		
Rating	Min	Max
Storage temperature	-40℃	+150℃
Supply voltage: VBAT	-0.4V	5.6V
Other terminal voltages	VSS-0.4V	VDD+0.4V

Recommended Operating Conditions		
Operating Condition	Min	Max
Operating temperature range	-40℃	+150℃
Guaranteed RF performance range ^(a)	-40℃	+150℃
Supply voltage: VBAT	2.2V	4.2V ^(b)

7. Application Circuit Diagram



SH-HC-08 Data Sheet

8、 PINs description

PIN NO.	NAME	FUNCTION
1	UART_TX	UART Data Output (P0_3)
2	UART_RX	UART Data Input (P0_2)
3	UART_CTS	UART Clear To Send Active Low (P0_4)
4	UART_RTS	UART Request To Send Active Low (P0_5)
5	NC	NC
6	DC	Debug the clock P2_2
7	DD	Debug the data P2_1
8	NC	
9	NC	
10	NC	
11	RESETB	Reset if low Input debounced so must below for>5ms to cause a reset
12	VCC	+3.3V Supply
13	GND	Ground
14	NC	
15	NC	
16	P1_4	
17	P1_6	
18	P1_7	
19	P1_5	
20	NC	
21	NC	
22	GND	
23	P1_1	LED
24	P1_0	Connect status
25	NC	
26	P0_6	wakeup
27	NC	
28	NC	
29	NC	
30	NC	
31	NC	

SH-HC-08 Data Sheet

32	NC	
33	NC	
34	P0_7	

9. Other configuration

State Instructions LED: P1_2 (PIN24)

Model	LED Display	Status
Slave	Even slow flash (800ms-on,800ms-off)	Waiting for matching
	Long bright	connection
Master	Even slow flash (800ms-on,800ms-off)	Waiting for matching
	Long bright	connection

Module dormancy set

Is only meaningful in from dormancy mode module, from the mode via a serial port to send "AT + SLEEP", if no accident, the module will return "OK+SLEEP" and enter a dormant state.

Wake up the module Settings

Method 1: short press the system button SW1 (PIN26).

WAKE up, a serial port will output "+ WAKE OK" string.

10.Layout Announcements

1, SH-HC-08 bluetooth module serial level should be 3.3 V, if the connection and 5V level system need to increase the level conversion chip.

2,Bluetooth signal is highly affected by the surrounding,such as trees, metal, wall can have certain absorption on the bluetooth signal or block, so the installation is not recommended in the metal case.

3,Due to metal will weaken the function of antenna, it is suggested that Lay in the module board, don't lay GND and a line under the antenna module, it is best to hollow out.

SH-HC-08 Data Sheet

11、 AT Command

The way to the AT command mode: supply power to the module, it will enter to the AT mode if it needn't pair. The interval of command is about 1 second.

Default parameter: Baud rate:9600N81, Password:000000

Command 1 , Testing Connection Commands

Command	Answer	Parameter
AT	OK	None

Command 2 , Query/Set —— Baud Rate

Command	Answer	Parameter
Query: AT+BAUD?	OK+Get:[para1]	Para1: 4~8 4=9600;5=19200; 6=38400;7=57600; 8=115200 Default: 4 (9600)
Set: AT+BAUD[para1]	OK+Set:[para1]	

Example: Send:AT+BAUD6

Return:OK+Set:6

The baud rate to 38400

Send AT+BAUD?

Return: OK+Get:6

Command 3 , Query/Set —— Bluetooth Name

Command	Answer	Parameter
Query: AT+NAME?	OK+NAME:[para1]	Para1: the name of device
Set: AT+NAME[para1]	OK+Set[para1]	Default: SH-HC-08

SH-HC-08 Data Sheet

Example: Send: AT+NAME?

Return: OK+Name:SH-HC-08

Send: AT+NAMEyourname

Return : OK+Set:yourname

Command 4 , Query/Set ——PIN Code

Command	Answer	Parameter
Query: AT+PASS?	OK+PASS:[para1]	Para1 is PIN Code: 000000~999999 Default: 000000
Set: AT+PASS[para1]	OK+Set:[para1]	

e.g.

Query Pin Code

Send: AT+PASS?

Return: OK+PASS:000000

Setup Pin Code 123456

Send: AT+PASS123456

Return : OK+Set:123456

Command 5, Query/Set ——Module Bond Mode

Command	Answer	Parameter
Query: AT+TYPE?	OK+ Get:[para]	Para: 0 ~ 2 0: Not need PIN Code 1: Auth not need PIN 2: Auth with PIN Default: 0
Set: AT+TYPE[para]	OK+Set:[para]	

Command 6, Restore all setup value to factory setup

SH-HC-08 Data Sheet

Command	Answer	Parameter
AT+RENEW	OK+RENEW	None

Command 7, Restart module

Command	Answer	Parameter
AT+RESET	OK+RESET	None

Command 8, Query module bluetooth address

Command	Answer	Parameter
AT+ADDR?	OK+ADDR: [MAC]	MAC: Bluetooth address

Command 9, Query/Set —Module Power

Command	Answer	Parameter
Query: AT+TXPW?	OK+ TXPW:[para]	Para: 0 ~ 3
Set: AT+TXPW[para]	OK+Set:[para]	0: -23dbm、 1: -6dbm 2: 0dbm、 3: 4dbm Default: 2

Command 10, set module into sleep mode

Command	Answer	Parameter
AT+SLEEP	OK+SLEEP	None

Only support Peripheral role.

When pin26 is high level and the module will go to normal mode.

Command 11, query verison

HC-08 BLUETOOTH UART COMMUNICATION MODULE V2.0 USER MANUAL



Version

HC-08 V2.0

Date

2014-09-16

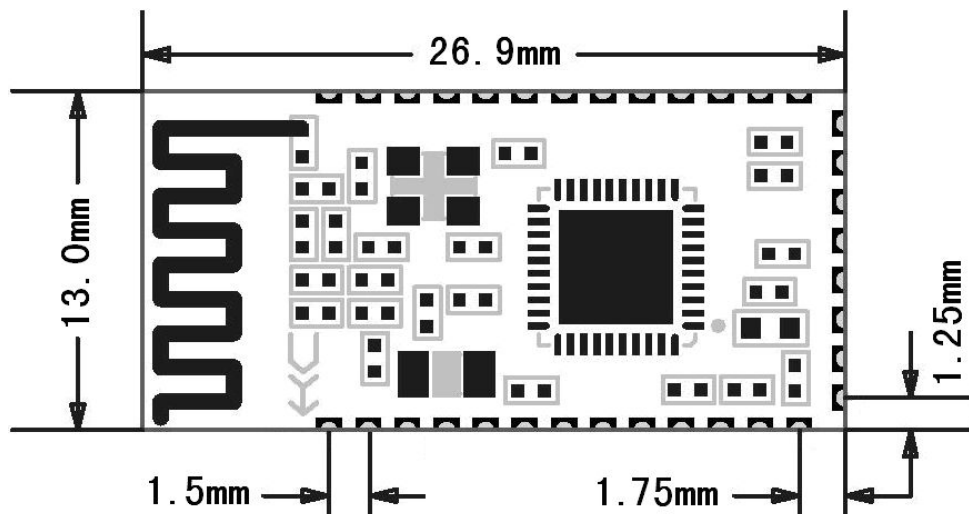
Product introduction

HC-08 Bluetooth UART communication module is a new generation of Bluetooth specification V4.0 BLE Bluetooth protocol based on the transmission module. Wireless working frequency is 2.4GHz ISM, modulation is GFSK. The maximum transmit power module 4dBm, the receiving sensitivity is -93dBm, and iphone4s can achieve 80 meters of super long distance communication under open environment.

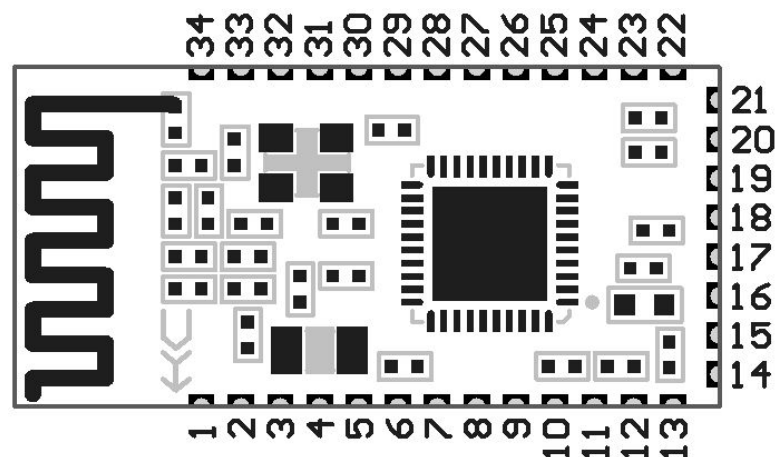
The module uses the stamp hole encapsulation, can patch welding, module size is 26.9mm * 13mm * 2.2mm, very convenient to the customer within the embedded application system.

The module uses the CC2540 chip, the configuration of the 256K Byte space, supports AT command, the user can according to need to change role and the serial baud rate, equipment name and other parameters, the use of flexible.

Product size



Pin definition



HC-08 BLUETOOTH UART COMMUNICATION MODULE USER MANUAL

The HC-08 module has 34 pins, on board PCB antenna, pin specific definitions are listed in the following table:

pin	definition	I/O	explain
1	TXD	output	URAT output, 3.3V TTL level
2	RXD	input,weak pull up	URAT input, 3.3V TTL level
3	NC	NC	
4	NC	NC	
5	NC		
6	DC	input	Debug clock
7	DD	Input/output	Debug data
8	PIO20	input,weak pull up	NC
9	PIO17	input,weak pull up	NC
10	PIO16	input,weak pull up	NC
11	RST	input,pull up	Module reset pin, a low level of not less than 5ms reset
12	VCC	input	Power pin, the requirements of 3.3V DC power supply, the supply current is not less than 100mA
13	GND		Ground
14	NC		
15	USB_D-	NC	
16	NC		
17	NC		
18	PIO15	input,weak pull up	NC
19	PIO14	input,weak pull up	NC
20	USB_D+	NC	
21	GND		Ground
22	GND		Ground
23	NC		
24	PIO13	output	LED output (Note①)
25	PIO11	input,weak pull up	NC
26	PIO12	input,weak pull up	The master module clear memory (Note②)
27	PIO10	input,weak pull up	NC
28	PIO07	input,weak pull up	NC
29	NC		
30	NC		
31	PIO06	input,weak pull up	NC
32	PIO01	input,weak pull up	NC
33	NC		
34	PIO00	input,weak pull up	NC

HC-08 BLUETOOTH UART COMMUNICATION MODULE USER MANUAL

Note①:Module indicating LED output pin, high level output, please use the resistance and LED connection.

The connection before,

From the slave module address the master module does not record, bright 100ms per second;

From the slave module address master module records, bright 900ms per second;

The slave module, LED light 1second very 2 seconds.

After connection, LED lights always.

Note②:Input pin, internal pull-down. This pin is connected with the high level, the master module to clear the slave module address have been recorded.

Electrical characteristics:

parameter	test condition		representative value
working voltage	-		DC2.0V~3.6V
working current (not LED)	master	not connected / connection	21mA/9mA
	slave	MODE0, not connected / connection	8.5mA/9mA
		MODE1, not connected / connection	340μA/1.6mA
		MODE2, not connected / connection	0.4μA/1.6mA
		MODE3, not connected / connection	1.2μA-160μA/1.6mA

AT COMMAND

The AT command to setting module parameter. Connection before, module can operating AT command. Connection after entering serial transparent transmission mode.

Module start is about 150ms, so the best after power on 200ms AT command operation. Unless otherwise indicated, the parameter setting of AT command is effective immediately. At the same time, parameters and functions of modification, the power down will not be lost.

After the success of AT command modify unified returns OK ("AT+RX,AT+VERSION" and so on the view of information command class except), no success does not return any information.

(1) AT COMMAND LIST

	AT Command ("x" - parameter)	Function	Default	Role
1	AT	Test command	-	M/S
2	AT+RX	Check the basic parameters	-	M/S
3	AT+DEFAULT	Restore factory setting	-	M/S
4	AT+RESET	Reset the module	-	M/S
5	AT+VERSION	Check version and date	-	M/S
6	AT+ROLE=x	Change master/slave role	S	M/S
7	AT+NAME=xxxxxxxxxxxx	Revise name	HC-08	M/S
8	AT+ADDR=xxxxxxxxxxxx	Revise address	Hardware address	M/S
9	AT+RFPM=x	Revise RF power	0(4dBm)	M/S
10	AT+BAUD=x,y	Revise UART baud	9600,N	M/S
11	AT+CONT=x	Set connectability	0(Can be connected)	M/S
12	AT+MODE=x	Set working mode	0	S
13	AT+AVDA=xxxxxxxxxxxx	Change the broadcast data	-	S
14	AT+TIME=x	Mode 3 broadcast cycle	5(s)	S

Note:

1. The AT command behind no newline; if no special instructions, all AT commands are not transmitted using newline.

2. The last 4 senior commands, must be used in combination, can play its due role BLE Bluetooth low energy. Using a Bluetooth low energy, there will be special instructions and program introduced in the following sections.

(2) COMMAND EXPLAIN

① Test command

Command:AT

Return:OK。

② Check the basic parameters

View the basic parameters such as bluetooth name, master/slave role, UART baud rate, address and password.

Command:AT+RX

Return:Name:HC-08

----->>>> bluetooth name

Role:Slave

----->>>> master/slave role

Baud:9600,NONE

----->>>> UART baud rate

Addr:xx,xx,xx,xx,xx,xx

----->>>> bluetooth address

PIN :000000

----->>>> bluetooth password

Note: Temporarily does not support change password!

③ Restore factory setting

Command:AT+DEFAULT

Return:OK

The module will automatically restart, please carry out new operation on the restart 200ms!

④ Reset the module

Command:AT+ RESET

Return:OK

The module will automatically restart, please carry out new operation on the restart 200ms!

⑤ Check version and date

Command:AT+ VERSION

Return:HC-08V2.0,2014-08-22

⑥ Change master/slave role

Set command:AT+ROLE=x

Query command:AT+ROLE=?

X:role(M or S), M:master, S:slave。

Default setting is S(slave)。

Send:AT+ROLE=M

Return:OK

Set master role,the module will automatically restart!

Send:AT+ROLE=?

Return:Master

You can view the role is the master module.

⑦ Revise name

Set command:AT+ NAME=xxxxxxxxxxxx

Query command:AT+ NAME=?

Default setting is HC-08, You can set the other name (12 characters limit, support the visual ASCII code and part of the escape character. The module supports chinese, but android devices must be converted to "UTF8 code" to normal display. To send more than 12 characters, then only the first 12 characters). Setup is complete, effective after module automatically reset!

Example:

Send:AT+NAME=HCKJ

Return:OKsetNAME

Send:AT+NAME=?

Return:HCKJ

⑧ Revise address

Set command:AT+ADDR=xxxxxxxxxxxx

Query command:AT+ADDR=?

The address must be 12 bit "0~F" uppercase characters, namely hexadecimal characters.

Example:

Send:AT+ADDR=1234567890AB

Return:OKsetADDR

Setup is complete, effective after module automatically reset!

Send:AT+ADDR=?

Return:1234567890AB

Send:**AT+ADDR=000000000000**

Return:OKsetADDR

Send "000000000", module to restore the default hardware address. Module factory default is hardware address.

⑨ Revise RF power

Set command:AT+RFPM=x

Query command:AT+RFPM=?

X:RF power,as shown in the following table:

Parameter	RF power
?	View the current RF power
0	4dBm (default)
1	0dBm
2	-6dBm
3	-23dBm

Example:

Send:AT+RFPM=2

Return:OK

RF power modified -6dBm.

Send:AT+RFPM=?

Return:-6dBm

RF power is -6dBm.

The peak current is more than 30mA (when RF power is 4dBm). Because the small discharge current button batteries, such as to use the button battery powered, the best setting for -6dBm or -23dBm.

⑩ Revise UART baud

Set command:AT+BAUD=x (Only modified the UART baud rate)

AT+BAUD=x,y (Modify the UART baud rate and parity bit)

Query command:AT+BAUD=?

x:UART baud rate,y: parity bit, As shown in the following table:

Parameter	UART baud :x	Parameter	parity bit :y
?	View the current baud rate		
1200	1200bps	N	No parity
2400	2400bps	E	Even parity
4800	4800bps	O	Odd parity
9600	9600bps (default)		
19200	19200bps		
38400	38400bps		
57600	57600bps		
115200	115200bps		

Example:

Send:AT+BAUD=19200

Return:OK19200

UART baud rate modified for 19200bps.

Send:AT+BAUD=4800,E

Return:OK4800,EVEN

UART baud rate modified for 4800bps,and even parity.

Send:AT+BAUD=?

Return:4800,EVEN

View UART baud rate and parity bit.

9600bps baud rate following each packet please do not exceed the maximum number of bytes to 500 bytes, 19200bps baud rate above each packet please refer to the following table, have a certain time interval between data packets. The following table is a variety of communication baud rate, the time interval of reference value:

HC-08 BLUETOOTH UART COMMUNICATION MODULE USER MANUAL

baud rate (bps)	1200	2400	4800	9600	19200	38400	57600	115200
500 bytes time interval (ms)	6800	3600	2000	1000				
300 bytes time interval (ms)	4200	2400	1200	600	400			
100 bytes time interval (ms)	1500	800	400	160	100	120		
80 bytes time interval (ms)	1000	650	320	120	80	60	100	
60 bytes time interval (ms)	800	500	250	100	60	60	60	100
20 bytes time interval (ms)	200	100	50	20	20	20	20	20

Note:

1. The above is the measured data, the fastest speed transceiver theory total:2500 bytes/sec, suggested that the speed control in the 2000 bytes/sec.
 2. Bytes of each packet, suggestion is an integer multiple of 20.
 3. Module sends data automatically subcontracting is an integer multiple of 20 bytes.
- Is to send a 100 bytes packet, will receive a plurality of packets at another end, each data packet is an integer multiple of 20, the total number of bytes for the full 100 bytes.

⑪ Set connectability

Set command:AT+CONT=x

Query command:AT+CONT=?

The X parameter functions are as follows:

Parameter	Master role	Slave role
0 (default)	Central Can be connected, the connection after entering ordinary transparent transmission mode	Peripheral Can be connected, the connection after entering ordinary transparent transmission mode
1	Observer The module can not be connected to other equipment, but will automatically scan the HC-08 from the broadcast datamachine package, fixed 2sec refresh time.	Broadcaster Not connected with the master role, but can be combined with low power mode 3, the realization of broadcast packets send.

Example:

Send:AT+CONT=1

Return:OK

Setup is complete, effective after module automatically reset!

Send:AT+CONT=?

Return:Non-Connectable

The command please with "AT+MODE", "AT+AVDA" and "AT+TIME" command with the use of.

Note:

1. The master/slave module "CONT=1" is masterly used for transmitting broadcast data. From the slave role to send broadcast data, broadcast master module will receive the corresponding data, and through the serial output.

2. This model is only for the user can grab this broadcast data package yourself.

The specific communication protocol is not described here, the intention to please the following official website consultation online customer service: <http://www.wavesen.com/>

⑫ **Set working mode (Only slaver)**

Set command:AT+MODE=x

Query command:AT+MODE=?

Command	Parameter	Return	Function
AT+ MODE	=?	0/1/2/3	Gets the current mode.
	=0	OK	Full power mode (default), LED open.
	=1		Level 1 power saving mode, LED close. No connection the current is 340 μ A, the connection speed like mode0.
	=2		Level 2 power saving mode, LED close. No connection the current is 0.4 μ A. Can not be found, not connected to wake up before, after awakening can be connected.
	=3		Level 3 power saving mode, LED close. No connection the current is 1.2μA~160μA (about 32μA default) . Combined with "AT+TIME" to set the broadcast time, thus to reduce power consumption. The use of specific methods please refer to the "AT+TIME" command.

Note:

1. Mode 3 is mainly used for:

A. Used to reduce the power consumption.

B. The slave module send the broadcast data to the master module, can be achieve one to many one-way communication (theory can be from a slave module to infinite master module).

C. As anti-lost alarm, attendance card, heart rate meter or other wireless device.

2. Mode 1/2/3 are available through the UART port to send 1 byte data to wake up, but front few bytes of data may be garbled after wake up. Therefore we recommend sending 10 bytes hexadecimal code of "0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF" to awaken the module, avoid the front several may be garbled byte. Since then, modules work on the full speed mode, the UART port can be normal to send

and receive data.

In non connected state, module after awakening into full speed mode and maintained for 5 minutes, and then return to the original mode. As long as 5 minutes in the UART has received data, then retiming.

If the module is in the connected state, then after awakening will remain in the full speed mode, before disconnecting, module will return the original power mode.

3. In addition to mode0, the other mode are closing LED, but after connection, LED will be light.

Example:

Send:AT+MODE=?

Return:0

View the current mode.

Send:AT+MODE=2

Return:OK

Setting mode 2, effective immediately.

⑬ Change the broadcast data (Only slave module)

Command:AT+AVDA=xxxxxxxxxxxx

Parameter “xxxxxxxxxxxx” can be any 1~12 bytes user data. If at this point the master module state of AT+CONT=1, then the master module UART port will output the “xxxxxxxxxxxx” data. The broadcast data will not be preserved permanently, will be deleted after the restart.

Example:

Slave role send:AT+AVDA=1234567890AB

Return:OK

If at this point the master module state of AT+CONT=1, UART port will output:1234567890AB.

⑭ Mode 3 broadcast cycle (Only slaver)

Set command:AT+TIME=x

Query command:AT+TIME=?

Parameter x setting range is as follows:

x	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
time/ sec	1	2	3	4	5 (default)	6	7	8	9	10	20	30	40	50	60
x	F	G	H	I	J	K									
time/ min	1	2	5	10	30	60									

Example:

Send:AT+TIME=F

Return:OK

Setting mode3 of the broadcast cycle for 60 seconds. Every 60 seconds, is sending a broadcast data.

Send:AT+TIME=?

Return:60s

Solution of low power mode (slave module):

1. The need to wake on wireless:

Enter "AT+MODE=1" or "AT+MODE=3", the module will enter a low power mode, until the master module request connection. After the connection the current is 1.6mA, such as data exchange module will automatically enter the full speed mode now, before disconnecting after it returned to the low power mode.

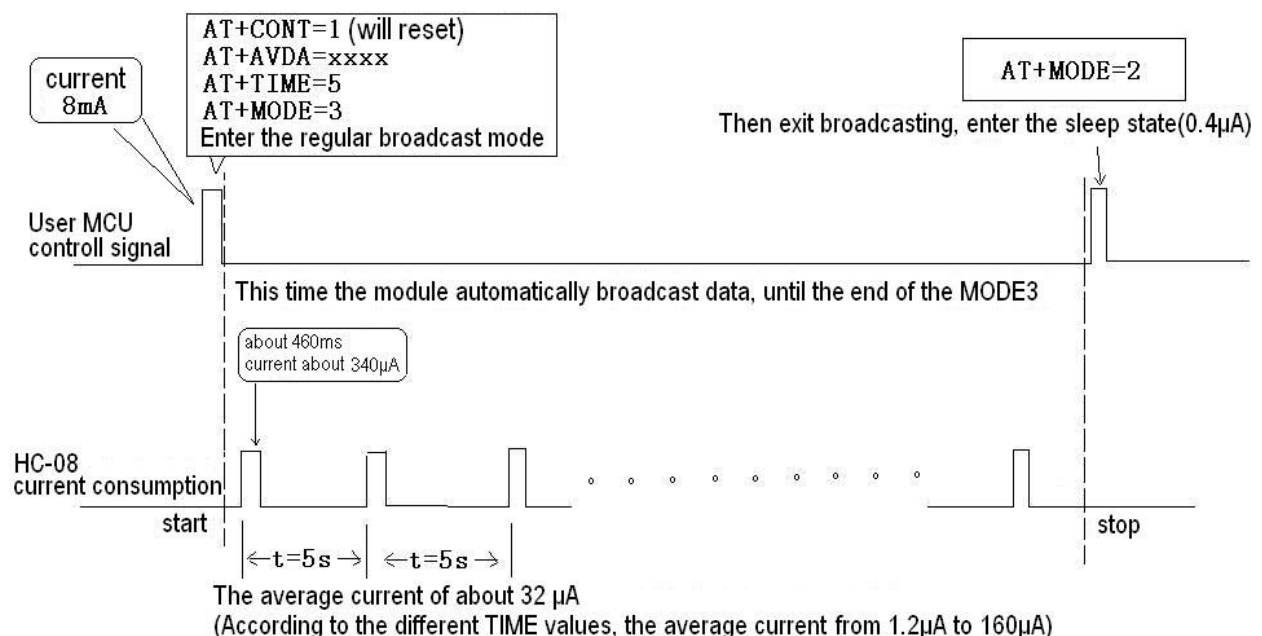
2. Can active connection case:

Enter "AT+MODE=2" to enter a low power consumption mode2, module to enter the deep sleep state, can not be master module discovery. The need to connect when you can send arbitrary data to awaken the module, and then after the connection can send and receive data.

Solution of low power broadcasting mode:

The first set master role:AT+CONT=1 -> AT+ROLE=M

And then set slave role:AT+CONT=1 -> AT+AVDA=1234 (≤12Bytes data)



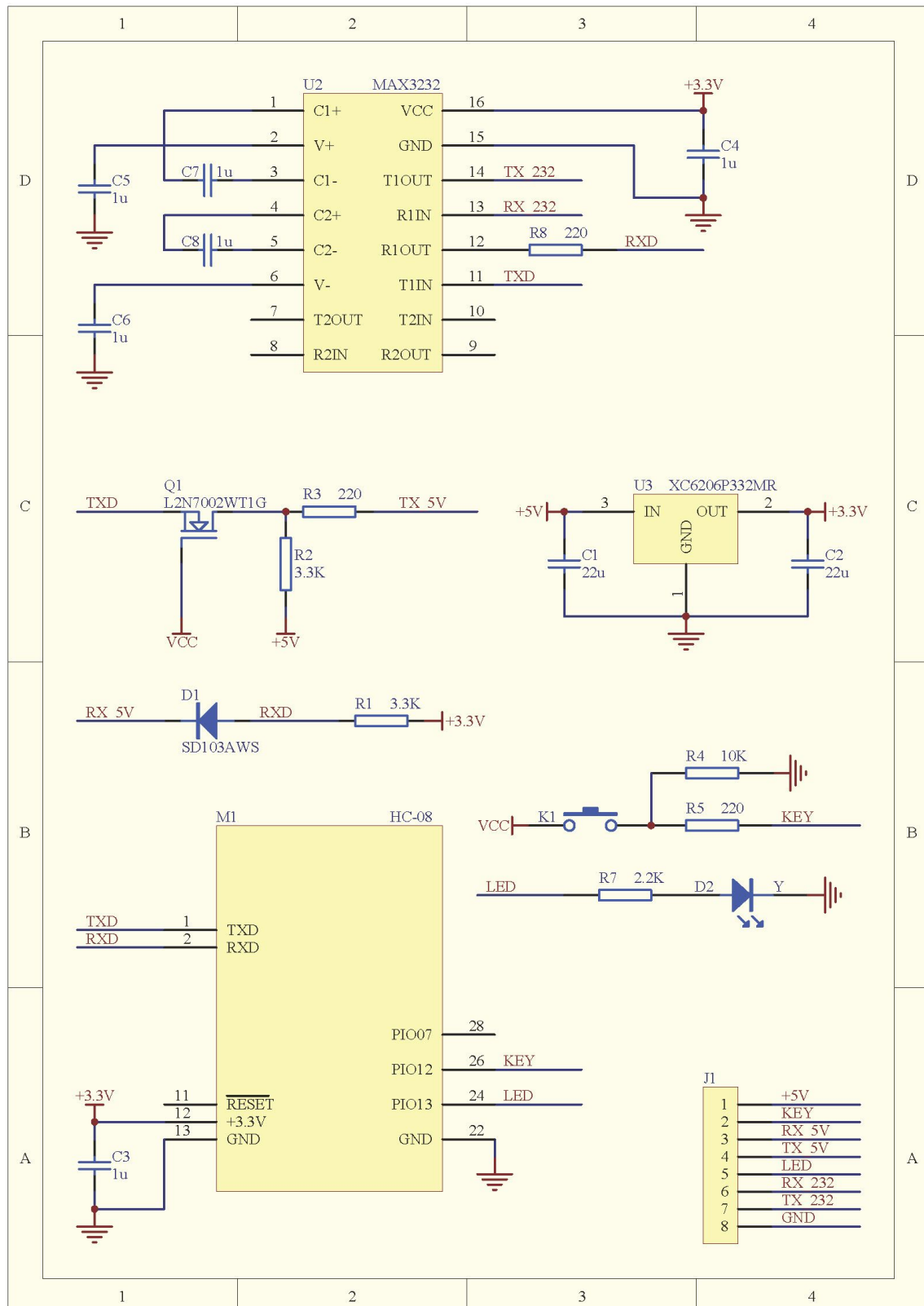
The user MCU to the module of UART port to send 10 bytes hexadecimal code of "0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF" to awaken the module, module into the full power mode. At this point the user can reasonably set according to need, the above "AT+CONT=1, AT+AVDA=xxxx, AT+MODE=3, AT+TIME=5" set to: broadcast and can not connected, the broadcast data is XXXX, mode3(period of 5

HC-08 BLUETOOTH UART COMMUNICATION MODULE USER MANUAL

seconds). According to the above, let the average current is less than 4 μA , TIME will be more than 1 minutes. This time is longer, lower power consumption.

The user wants to transmit data frequently, proposed to enter mode2 in idle time, need to transmit data to switch to the corresponding mode.

Reference schematic



SH-HC-08 Data Sheet

Command	Answer	Parameter
AT+VERSION	Version info	

Module Bluetooth :

SH-HC-08 Data Sheet

SH-HC-08 Data Sheet

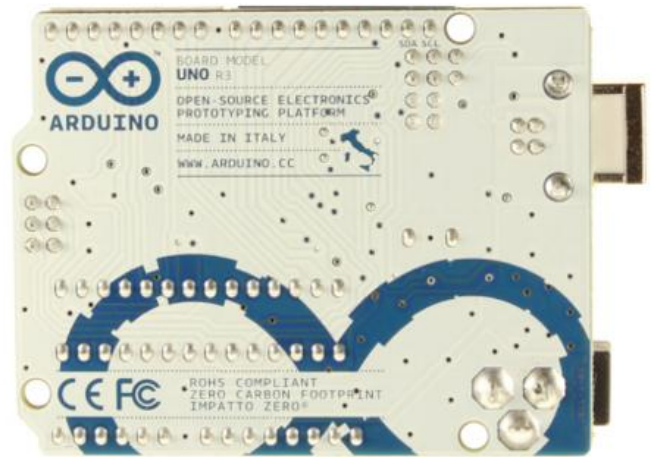
Bluetooth 4.0 BLE Module

1、 Overview	2
2、 Feature	2
3、 Product's picture	3
4. Application Fields	4
5. Physical Characteristics	4
6. Electrical Characteristics	5
7. Application Circuit Diagram	5
8、 PINs description	6
9. Other configuration	7
10.Layout Announcements	7
11、 AT Command	8

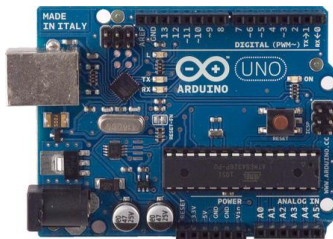
Arduino Uno



Arduino Uno R3 Front



Arduino Uno R3 Back



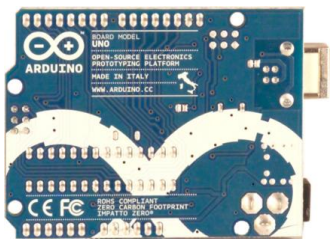
Arduino Uno R2 Front



Arduino Uno SMD



Arduino Uno Front



Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 ([datasheet](#)). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

| [Revision 2](#) of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into [DFU mode](#).

| [Revision 3](#) of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the [index of Arduino boards](#).

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V

Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Schematic & Reference Design

EAGLE files: [arduino-uno-Rev3-reference-design.zip](#) (NOTE: works with Eagle 6.0 and newer)

Schematic: [arduino-uno-Rev3-schematic.pdf](#)

Note: The Arduino reference design can use an Atmega8, 168, or 328, Current models use an ATmega328, but an Atmega8 is shown in the schematic for reference. The pin configuration is identical on all three processors.

Power

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega328 has 32 KB (with 0.5 KB used for the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the [EEPROM library](#)).

Input and Output

Each of the 14 digital pins on the Uno can be used as an input or output, using [pinMode\(\)](#), [digitalWrite\(\)](#), and [digitalRead\(\)](#) functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the [attachInterrupt\(\)](#) function for details.
- **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the [analogWrite\(\)](#) function.

- **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication using the [SPI library](#).
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the [analogReference\(\)](#) function. Additionally, some pins have specialized functionality:

- **TWI: A4 or SDA pin and A5 or SCL pin.** Support TWI communication using the [Wire library](#).

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with [analogReference\(\)](#).
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

See also the [mapping between Arduino pins and ATmega328 ports](#). The mapping for the Atmega8, 168, and 328 is identical.

Communication

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, [on Windows, a .inf file is required](#). The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A [SoftwareSerial library](#) allows for serial communication on any of the Uno's digital pins.

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the [documentation](#) for details. For SPI communication, use the [SPI library](#).

Programming

The Arduino Uno can be programmed with the Arduino software ([download](#)). Select "Arduino Uno from the **Tools > Board** menu (according to the microcontroller on your board). For details, see the [reference](#) and [tutorials](#).

The ATmega328 on the Arduino Uno comes preburned with a [bootloader](#) that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol ([reference](#), [C header files](#)).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see [these instructions](#) for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available . The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then resetting the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use [Atmel's FLIP software](#) (Windows) or the [DFU programmer](#) (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See [this user-contributed tutorial](#) for more information.

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino Uno is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see [this forum thread](#) for details.

USB Overcurrent Protection

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics

The maximum length and width of the Uno PCB are 2.7 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Four screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

Serially Interfaced, 8-Digit LED Display Drivers

General Description

The MAX7219/MAX7221 are compact, serial input/output common-cathode display drivers that interface microprocessors (μ Ps) to 7-segment numeric LED displays of up to 8 digits, bar-graph displays, or 64 individual LEDs. Included on-chip are a BCD code-B decoder, multiplex scan circuitry, segment and digit drivers, and an 8x8 static RAM that stores each digit. Only one external resistor is required to set the segment current for all LEDs. The MAX7221 is compatible with SPI™, QSPI™, and MICROWIRE™, and has slew-rate-limited segment drivers to reduce EMI.

A convenient 4-wire serial interface connects to all common μ Ps. Individual digits may be addressed and updated without rewriting the entire display. The MAX7219/MAX7221 also allow the user to select code-B decoding or no-decode for each digit.

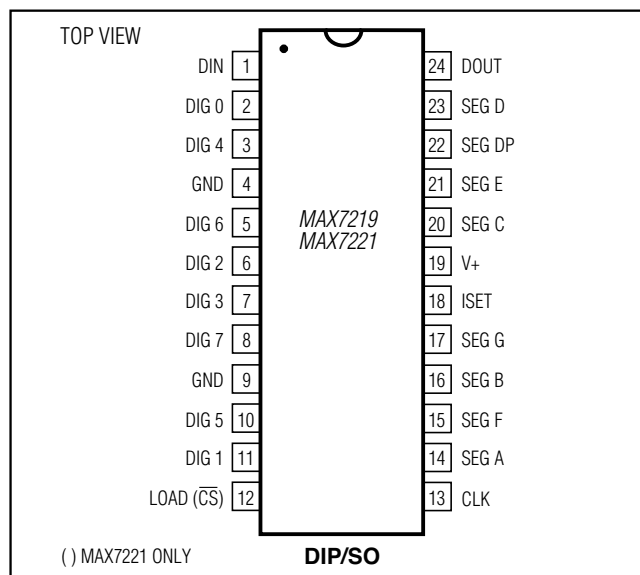
The devices include a 150 μ A low-power shutdown mode, analog and digital brightness control, a scan-limit register that allows the user to display from 1 to 8 digits, and a test mode that forces all LEDs on.

For applications requiring 3V operation or segment blinking, refer to the MAX6951 data sheet.

Applications

Bar-Graph Displays	Panel Meters
Industrial Controllers	LED Matrix Displays

Pin Configuration



SPI and QSPI are trademarks of Motorola Inc. MICROWIRE is a trademark of National Semiconductor Corp.

Features

- ◆ 10MHz Serial Interface
- ◆ Individual LED Segment Control
- ◆ Decode/No-Decode Digit Selection
- ◆ 150 μ A Low-Power Shutdown (Data Retained)
- ◆ Digital and Analog Brightness Control
- ◆ Display Blanked on Power-Up
- ◆ Drive Common-Cathode LED Display
- ◆ Slew-Rate Limited Segment Drivers for Lower EMI (MAX7221)
- ◆ SPI, QSPI, MICROWIRE Serial Interface (MAX7221)
- ◆ 24-Pin DIP and SO Packages

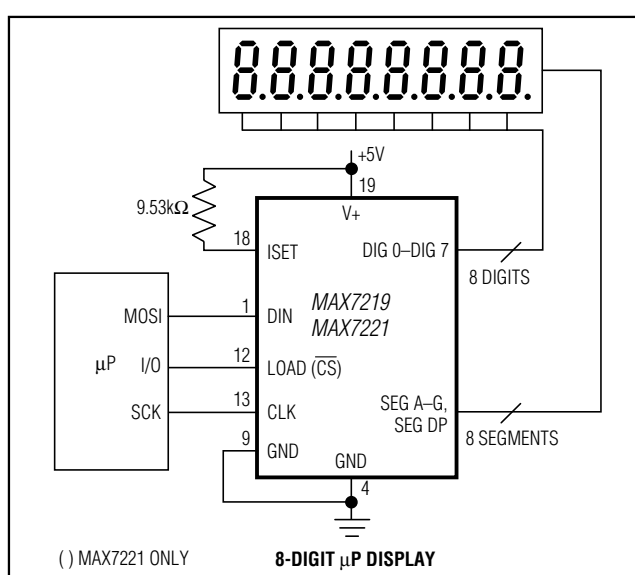
Ordering Information

PART	TEMP RANGE	PIN-PACKAGE
MAX7219CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX7219CWG	0°C to +70°C	24 Wide SO
MAX7219C/D	0°C to +70°C	Dice*
MAX7219ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX7219EWG	-40°C to +85°C	24 Wide SO
MAX7219ERG	-40°C to +85°C	24 Narrow Cerdip

Ordering Information continued at end of data sheet.

*Dice are specified at $T_A = +25^\circ\text{C}$.

Typical Application Circuit



MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

ABSOLUTE MAXIMUM RATINGS

Voltage (with respect to GND)

V+	-0.3V to 6V
DIN, CLK, LOAD, $\overline{CS}$	-0.3V to 6V
All Other Pins	-0.3V to (V+ + 0.3V)

Current

DIG 0–DIG 7 Sink Current	500mA
SEG A–G, DP Source Current	100mA

Continuous Power Dissipation ($T_A = +85^\circ\text{C}$)

Narrow Plastic DIP (derate 13.3mW/ $^\circ\text{C}$ above $+70^\circ\text{C}$)	1066mW
Wide SO (derate 11.8mW/ $^\circ\text{C}$ above $+70^\circ\text{C}$)	941mW
Narrow CERDIP (derate 12.5mW/ $^\circ\text{C}$ above $+70^\circ\text{C}$)	1000mW

Operating Temperature Ranges (T_{MIN} to T_{MAX})

MAX7219C_G/MAX7221C_G	0°C to $+70^\circ\text{C}$
MAX7219E_G/MAX7221E_G	-40°C to $+85^\circ\text{C}$
Storage Temperature Range	-65°C to $+160^\circ\text{C}$
Lead Temperature (soldering, 10s)	$+300^\circ\text{C}$

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS

($V_+ = 5\text{V} \pm 10\%$, $R_{\text{SET}} = 9.53\text{k}\Omega \pm 1\%$, $T_A = T_{\text{MIN}}$ to T_{MAX} , unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Operating Supply Voltage	V_+		4.0		5.5	V
Shutdown Supply Current	I_+	All digital inputs at V_+ or GND, $T_A = +25^\circ\text{C}$			150	μA
Operating Supply Current	I_+	$R_{\text{SET}} = \text{open circuit}$			8	mA
		All segments and decimal point on, $I_{\text{SEG}_-} = -40\text{mA}$		330		
Display Scan Rate	f_{OSC}	8 digits scanned	500	800	1300	Hz
Digit Drive Sink Current	I_{DIGIT}	$V_+ = 5\text{V}$, $V_{\text{OUT}} = 0.65\text{V}$	320			mA
Segment Drive Source Current	I_{SEG}	$T_A = +25^\circ\text{C}$, $V_+ = 5\text{V}$, $V_{\text{OUT}} = (V_+ - 1\text{V})$	-30	-40	-45	mA
Segment Current Slew Rate (MAX7221 only)	$\Delta I_{\text{SEG}}/\Delta t$	$T_A = +25^\circ\text{C}$, $V_+ = 5\text{V}$, $V_{\text{OUT}} = (V_+ - 1\text{V})$	10	20	50	mA/ μs
Segment Drive Current Matching	ΔI_{SEG}			3.0		%
Digit Drive Leakage (MAX7221 only)	I_{DIGIT}	Digit off, $V_{\text{DIGIT}} = V_+$			-10	μA
Segment Drive Leakage (MAX7221 only)	I_{SEG}	Segment off, $V_{\text{SEG}} = 0\text{V}$			1	μA
Digit Drive Source Current (MAX7219 only)	I_{DIGIT}	Digit off, $V_{\text{DIGIT}} = (V_+ - 0.3\text{V})$	-2			mA
Segment Drive Sink Current (MAX7219 only)	I_{SEG}	Segment off, $V_{\text{SEG}} = 0.3\text{V}$	5			mA

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

ELECTRICAL CHARACTERISTICS (continued)

(V₊ = 5V ±10%, R_{SET} = 9.53kΩ ±1%, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

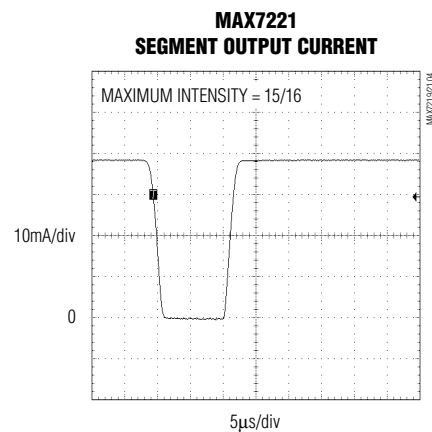
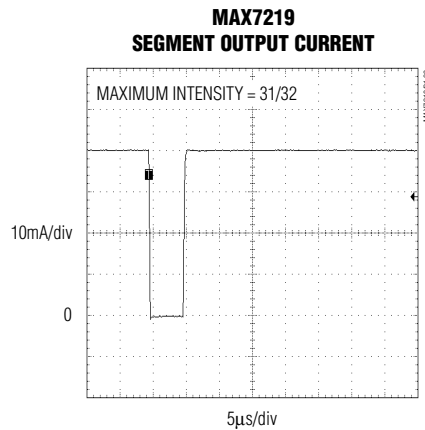
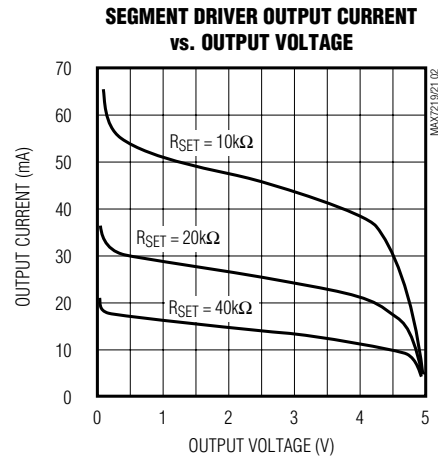
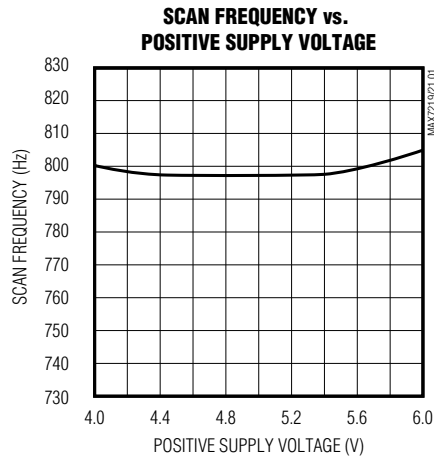
PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
LOGIC INPUTS						
Input Current DIN, CLK, LOAD, $\overline{\text{CS}}$	I _{IH} , I _{IL}	V _{IN} = 0V or V ₊	-1		1	μA
Logic High Input Voltage	V _{IH}		3.5			V
Logic Low Input Voltage	V _{IL}				0.8	V
Output High Voltage	V _{OH}	DOUT, I _{SOURCE} = -1mA	V ₊ - 1			V
Output Low Voltage	V _{OL}	DOUT, I _{SINK} = 1.6mA			0.4	V
Hysteresis Voltage	ΔV _I	DIN, CLK, LOAD, $\overline{\text{CS}}$		1		V
TIMING CHARACTERISTICS						
CLK Clock Period	t _{CP}		100			ns
CLK Pulse Width High	t _{CH}		50			ns
CLK Pulse Width Low	t _{CL}		50			ns
$\overline{\text{CS}}$ Fall to SCLK Rise Setup Time (MAX7221 only)	t _{CSS}		25			ns
CLK Rise to $\overline{\text{CS}}$ or LOAD Rise Hold Time	t _{CSH}		0			ns
DIN Setup Time	t _{DS}		25			ns
DIN Hold Time	t _{DH}		0			ns
Output Data Propagation Delay	t _{DO}	C _{LOAD} = 50pF			25	ns
Load-Rising Edge to Next Clock Rising Edge (MAX7219 only)	t _{LDCK}		50			ns
Minimum $\overline{\text{CS}}$ or LOAD Pulse High	t _{CSW}		50			ns
Data-to-Segment Delay	t _{DSPD}				2.25	ms

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Typical Operating Characteristics

($V_+ = +5V$, $T_A = +25^\circ C$, unless otherwise noted.)



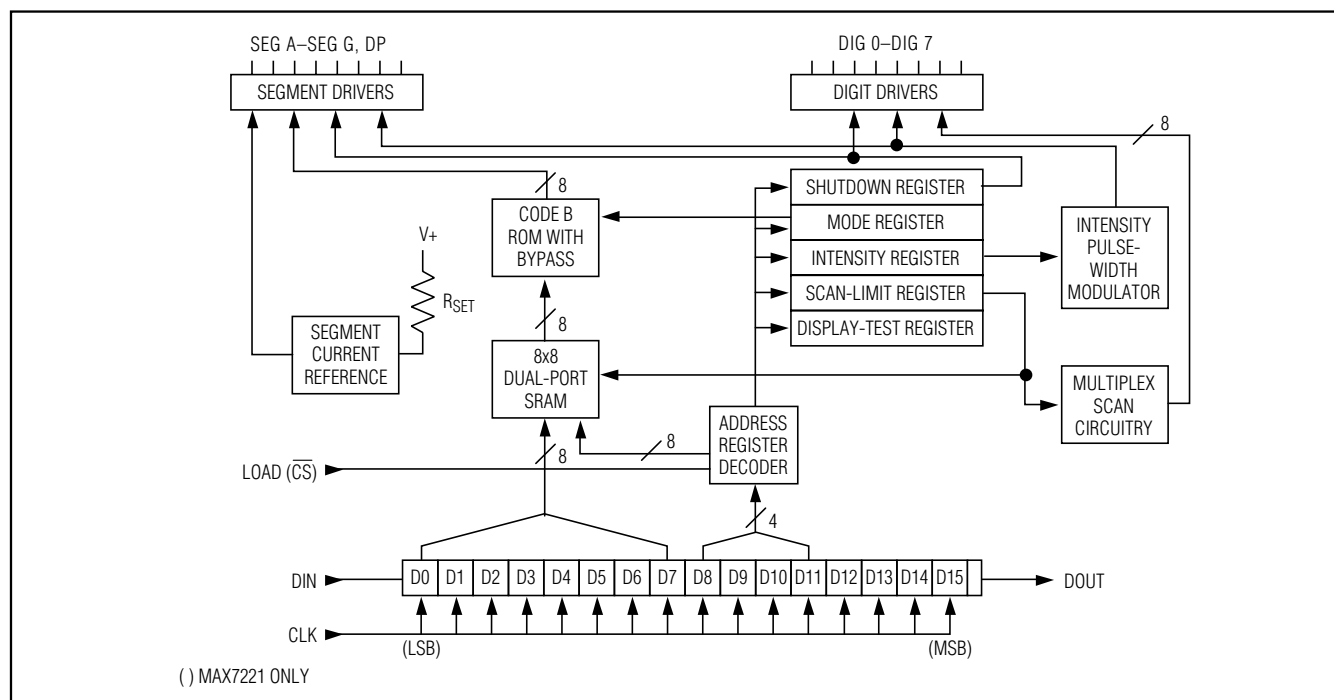
MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Pin Description

PIN	NAME	FUNCTION
1	DIN	Serial-Data Input. Data is loaded into the internal 16-bit shift register on CLK's rising edge.
2, 3, 5–8, 10, 11	DIG 0–DIG 7	Eight-Digit Drive Lines that sink current from the display common cathode. The MAX7219 pulls the digit outputs to V+ when turned off. The MAX7221's digit drivers are high-impedance when turned off.
4, 9	GND	Ground (both GND pins must be connected)
12	LOAD (MAX7219)	Load-Data Input. The last 16 bits of serial data are latched on LOAD's rising edge.
	$\overline{CS}$ (MAX7221)	Chip-Select Input. Serial data is loaded into the shift register while $\overline{CS}$ is low. The last 16 bits of serial data are latched on $\overline{CS}$'s rising edge.
13	CLK	Serial-Clock Input. 10MHz maximum rate. On CLK's rising edge, data is shifted into the internal shift register. On CLK's falling edge, data is clocked out of DOUT. On the MAX7221, the CLK input is active only while $\overline{CS}$ is low.
14–17, 20–23	SEG A–SEG G, DP	Seven Segment Drives and Decimal Point Drive that source current to the display. On the MAX7219, when a segment driver is turned off it is pulled to GND. The MAX7221 segment drivers are high-impedance when turned off.
18	ISSET	Connect to V _{DD} through a resistor (R _{SET}) to set the peak segment current (Refer to <i>Selecting R_{SET} Resistor and Using External Drivers</i> section).
19	V+	Positive Supply Voltage. Connect to +5V.
24	DOUT	Serial-Data Output. The data into DIN is valid at DOUT 16.5 clock cycles later. This pin is used to daisy-chain several MAX7219/MAX7221's and is never high-impedance.

Functional Diagram



MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

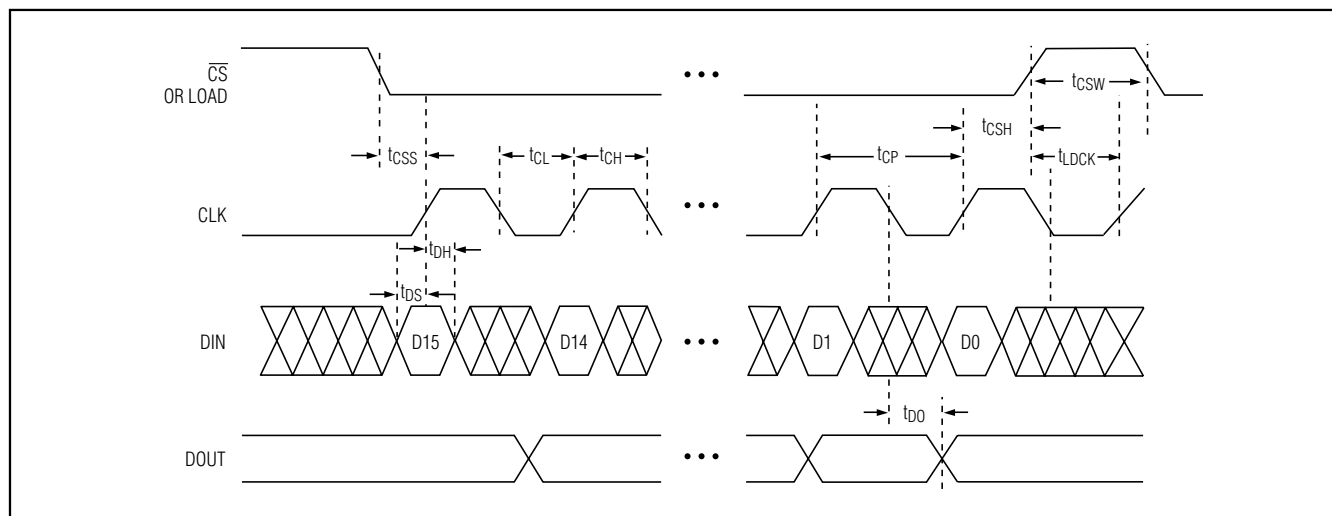


Figure 1. Timing Diagram

Table 1. Serial-Data Format (16 Bits)

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				MSB	DATA						LSB

Detailed Description

MAX7219/MAX7221 Differences

The MAX7219 and MAX7221 are identical except for two parameters: the MAX7221 segment drivers are slew-rate limited to reduce electromagnetic interference (EMI), and its serial interface is fully SPI compatible.

Serial-Addressing Modes

For the MAX7219, serial data at DIN, sent in 16-bit packets, is shifted into the internal 16-bit shift register with each rising edge of CLK regardless of the state of LOAD. For the MAX7221, CS must be low to clock data in or out. The data is then latched into either the digit or control registers on the rising edge of LOAD/CS. LOAD/CS must go high concurrently with or after the 16th rising clock edge, but before the next rising clock edge or data will be lost. Data at DIN is propagated through the shift register and appears at DOUT 16.5 clock cycles later. Data is clocked out on the falling edge of CLK. Data bits are labeled D0–D15 (Table 1). D8–D11 contain the register address. D0–D7 contain the data, and D12–D15 are “don’t care” bits. The first received is D15, the most significant bit (MSB).

Digit and Control Registers

Table 2 lists the 14 addressable digit and control registers. The digit registers are realized with an on-chip, 8x8 dual-port SRAM. They are addressed directly so that individual digits can be updated and retain data as long as V+ typically exceeds 2V. The control registers consist of decode mode, display intensity, scan limit (number of scanned digits), shutdown, and display test (all LEDs on).

Shutdown Mode

When the MAX7219 is in shutdown mode, the scan oscillator is halted, all segment current sources are pulled to ground, and all digit drivers are pulled to V+, thereby blanking the display. The MAX7221 is identical, except the drivers are high-impedance. Data in the digit and control registers remains unaltered. Shutdown can be used to save power or as an alarm to flash the display by successively entering and leaving shutdown mode. For minimum supply current in shutdown mode, logic inputs should be at ground or V+ (CMOS-logic levels).

Typically, it takes less than 250μs for the MAX7219/MAX7221 to leave shutdown mode. The display driver can be programmed while in shutdown mode, and shutdown mode can be overridden by the display-test function.

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Table 2. Register Address Map

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0xXA
Scan Limit	X	1	0	1	1	0xXB
Shutdown	X	1	1	0	0	0xXC
Display Test	X	1	1	1	1	0xFF

Initial Power-Up

On initial power-up, all control registers are reset, the display is blanked, and the MAX7219/MAX7221 enter shutdown mode. Program the display driver prior to display use. Otherwise, it will initially be set to scan one digit, it will not decode data in the data registers, and the intensity register will be set to its minimum value.

Decode-Mode Register

The decode-mode register sets BCD code B (0-9, E, H, L, P, and -) or no-decode operation for each digit. Each bit in the register corresponds to one digit. A logic high selects code B decoding while logic low bypasses the decoder. Examples of the decode mode control-register format are shown in Table 4.

When the code B decode mode is used, the decoder looks only at the lower nibble of the data in the digit registers (D3–D0), disregarding bits D4–D6. D7, which sets the decimal point (SEG DP), is independent of the decoder and is positive logic (D7 = 1 turns the decimal point on). Table 5 lists the code B font.

When no-decode is selected, data bits D7–D0 correspond to the segment lines of the MAX7219/MAX7221. Table 6 shows the one-to-one pairing of each data bit to the appropriate segment line.

Table 3. Shutdown Register Format (Address (Hex) = 0xXC)

MODE	ADDRESS CODE (HEX)	REGISTER DATA							
		D7	D6	D5	D4	D3	D2	D1	D0
Shutdown Mode	0xXC	X	X	X	X	X	X	X	0
Normal Operation	0xXC	X	X	X	X	X	X	X	1

Table 4. Decode-Mode Register Examples (Address (Hex) = 0xX9)

DECODE MODE	REGISTER DATA								HEX CODE
	D7	D6	D5	D4	D3	D2	D1	D0	
No decode for digits 7–0	0	0	0	0	0	0	0	0	0x00
Code B decode for digit 0 No decode for digits 7–1	0	0	0	0	0	0	0	1	0x01
Code B decode for digits 3–0 No decode for digits 7–4	0	0	0	0	1	1	1	1	0x0F
Code B decode for digits 7–0	1	1	1	1	1	1	1	1	0xFF

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Table 5. Code B Font

7-SEGMENT CHARACTER	REGISTER DATA						ON SEGMENTS = 1							
	D7*	D6–D4	D3	D2	D1	D0	DP*	A	B	C	D	E	F	G
0		X	0	0	0	0		1	1	1	1	1	1	0
1		X	0	0	0	1		0	1	1	0	0	0	0
2		X	0	0	1	0		1	1	0	1	1	0	1
3		X	0	0	1	1		1	1	1	1	0	0	1
4		X	0	1	0	0		0	1	1	0	0	1	1
5		X	0	1	0	1		1	0	1	1	0	1	1
6		X	0	1	1	0		1	0	1	1	1	1	1
7		X	0	1	1	1		1	1	1	0	0	0	0
8		X	1	0	0	0		1	1	1	1	1	1	1
9		X	1	0	0	1		1	1	1	1	0	1	1
—		X	1	0	1	0		0	0	0	0	0	0	1
E		X	1	0	1	1		1	0	0	1	1	1	1
H		X	1	1	0	0		0	1	1	0	1	1	1
L		X	1	1	0	1		0	0	0	1	1	1	0
P		X	1	1	1	0		1	1	0	0	1	1	1
blank		X	1	1	1	1		0	0	0	0	0	0	0

*The decimal point is set by bit D7 = 1

Table 6. No-Decode Mode Data Bits and Corresponding Segment Lines

The diagram shows a standard 7-segment LED display layout. The segments are labeled as follows: A (top horizontal), B (top-right vertical), C (bottom-right vertical), D (bottom horizontal), E (bottom-left vertical), F (top-left vertical), and G (middle horizontal). A small circle labeled DP represents the decimal point, located to the right of the bottom segment D.

STANDARD 7-SEGMENT LED

REGISTER DATA

	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Corresponding Segment Line	DP	A	B	C	D	E	F	G

Intensity Control and Interdigit Blanking

The MAX7219/MAX7221 allow display brightness to be controlled with an external resistor (RSET) connected between V+ and ISET. The peak current sourced from the segment drivers is nominally 100 times the current entering ISET. This resistor can either be fixed or variable to allow brightness adjustment from the front panel. Its minimum value should be 9.53kΩ, which typically sets the segment current at 40mA. Display brightness can also be controlled digitally by using the intensity register.

Digital control of display brightness is provided by an internal pulse-width modulator, which is controlled by the lower nibble of the intensity register. The modulator scales the average segment current in 16 steps from a maximum of 31/32 down to 1/32 of the peak current set by RSET (15/16 to 1/16 on MAX7221). Table 7 lists the intensity register format. The minimum interdigit blanking time is set to 1/32 of a cycle.

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Table 7. Intensity Register Format (Address (Hex) = 0xA)

DUTY CYCLE		D7	D6	D5	D4	D3	D2	D1	D0	HEX CODE
MAX7219	MAX7221									
1/32 (min on)	1/16 (min on)	X	X	X	X	0	0	0	0	0x0
3/32	2/16	X	X	X	X	0	0	0	1	0x1
5/32	3/16	X	X	X	X	0	0	1	0	0x2
7/32	4/16	X	X	X	X	0	0	1	1	0x3
9/32	5/16	X	X	X	X	0	1	0	0	0x4
11/32	6/16	X	X	X	X	0	1	0	1	0x5
13/32	7/16	X	X	X	X	0	1	1	0	0x6
15/32	8/16	X	X	X	X	0	1	1	1	0x7
17/32	9/16	X	X	X	X	1	0	0	0	0x8
19/32	10/16	X	X	X	X	1	0	0	1	0x9
21/32	11/16	X	X	X	X	1	0	1	0	0xA
23/32	12/16	X	X	X	X	1	0	1	1	0xB
25/32	13/16	X	X	X	X	1	1	0	0	0xC
27/32	14/16	X	X	X	X	1	1	0	1	0xD
29/32	15/16	X	X	X	X	1	1	1	0	0xE
31/32	15/16 (max on)	X	X	X	X	1	1	1	1	0xF

Table 8. Scan-Limit Register Format (Address (Hex) = 0xB)

SCAN LIMIT	REGISTER DATA								HEX CODE
	D7	D6	D5	D4	D3	D2	D1	D0	
Display digit 0 only*	X	X	X	X	X	0	0	0	0x0
Display digits 0 & 1*	X	X	X	X	X	0	0	1	0x1
Display digits 0 1 2*	X	X	X	X	X	0	1	0	0x2
Display digits 0 1 2 3	X	X	X	X	X	0	1	1	0x3
Display digits 0 1 2 3 4	X	X	X	X	X	1	0	0	0x4
Display digits 0 1 2 3 4 5	X	X	X	X	X	1	0	1	0x5
Display digits 0 1 2 3 4 5 6	X	X	X	X	X	1	1	0	0x6
Display digits 0 1 2 3 4 5 6 7	X	X	X	X	X	1	1	1	0x7

*See Scan-Limit Register section for application.

Scan-Limit Register

The scan-limit register sets how many digits are displayed, from 1 to 8. They are displayed in a multiplexed manner with a typical display scan rate of 800Hz with 8 digits displayed. If fewer digits are displayed, the scan rate is $8f_{OSC}/N$, where N is the number of digits

scanned. Since the number of scanned digits affects the display brightness, the scan-limit register should not be used to blank portions of the display (such as leading zero suppression). Table 8 lists the scan-limit register format.

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

If the scan-limit register is set for three digits or less, individual digit drivers will dissipate excessive amounts of power. Consequently, the value of the RSET resistor must be adjusted according to the number of digits displayed, to limit individual digit driver power dissipation. Table 9 lists the number of digits displayed and the corresponding maximum recommended segment current when the digit drivers are used.

Display-Test Register

The display-test register operates in two modes: normal and display test. Display-test mode turns all LEDs on by overriding, but not altering, all controls and digit registers (including the shutdown register). In display-test mode, 8 digits are scanned and the duty cycle is 31/32 (15/16 for MAX7221). Table 10 lists the display-test register format.

Table 9. Maximum Segment Current for 1-, 2-, or 3-Digit Displays

NUMBER OF DIGITS DISPLAYED	MAXIMUM SEGMENT CURRENT (mA)
1	10
2	20
3	30

Table 10. Display-Test Register Format (Address (Hex) = 0xXF)

MODE	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Normal Operation	X	X	X	X	X	X	X	0
Display Test Mode	X	X	X	X	X	X	X	1

Note: The MAX7219/MAX7221 remain in display-test mode (all LEDs on) until the display-test register is reconfigured for normal operation.

No-Op Register

The no-op register is used when cascading MAX7219s or MAX7221s. Connect all devices' LOAD/CS inputs together and connect DOUT to DIN on adjacent devices. DOUT is a CMOS logic-level output that easily drives DIN of successively cascaded parts. (Refer to the *Serial Addressing Modes* section for detailed information on serial input/output timing.) For example, if four MAX7219s are cascaded, then to write to the

fourth chip, sent the desired 16-bit word, followed by three no-op codes (hex 0xXX0X, see Table 2). When LOAD/CS goes high, data is latched in all devices. The first three chips receive no-op commands, and the fourth receives the intended data.

Applications Information

Supply Bypassing and Wiring

To minimize power-supply ripple due to the peak digit driver currents, connect a 10µF electrolytic and a 0.1µF ceramic capacitor between V+ and GND as close to the device as possible. The MAX7219/MAX7221 should be placed in close proximity to the LED display, and connections should be kept as short as possible to minimize the effects of wiring inductance and electromagnetic interference. Also, both GND pins must be connected to ground.

Selecting RSET Resistor and Using External Drivers

The current per segment is approximately 100 times the current in ISET. To select RSET, see Table 11. The MAX7219/MAX7221's maximum recommended segment current is 40mA. For segment current levels above these levels, external digit drivers will be needed. In this application, the MAX7219/MAX7221 serve only as controllers for other high-current drivers or transistors. Therefore, to conserve power, use RSET = 47kΩ when using external current sources as segment drivers.

The example in Figure 2 uses the MAX7219/MAX7221's segment drivers, a MAX394 single-pole double-throw analog switch, and external transistors to drive 2.3" AND2307SLC common-cathode displays. The 5.6V zener diode has been added in series with the decimal point LED because the decimal point LED forward voltage is typically 4.2V. For all other segments the LED forward voltage is typically 8V. Since external transistors are used to sink current (DIG 0 and DIG 1 are used as logic switches), peak segment currents of 45mA are allowed even though only two digits are displayed. In applications where the MAX7219/MAX7221's digit drivers are used to sink current and fewer than four digits are displayed, Table 9 specifies the maximum allowable segment current. RSET must be selected accordingly (Table 11).

Refer to the *Continuous Power Dissipation* section of the *Absolute Maximum Ratings* to calculate acceptable limits for ambient temperature, segment current, and the LED forward-voltage drop.

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Table 11. RSET vs. Segment Current and LED Forward Voltage

ISEG (mA)	VLED (V)				
	1.5	2.0	2.5	3.0	3.5
40	12.2	11.8	11.0	10.6	9.69
30	17.8	17.1	15.8	15.0	14.0
20	29.8	28.0	25.9	24.5	22.6
10	66.7	63.7	59.3	55.4	51.2

Table 12. Package Thermal Resistance Data

PACKAGE	THERMAL RESISTANCE (θJA)
24 Narrow DIP	+75°C/W
24 Wide SO	+85°C/W
24 Cerdip	+80°C/W
Maximum Junction Temperature (TJ) = +150°C	
Maximum Ambient Temperature (TA) = +85°C	

Computing Power Dissipation

The upper limit for power dissipation (PD) for the MAX7219/MAX7221 is determined from the following equation:

$$PD = (V+ \times 8mA) + (V+ - V_{LED})(DUTY \times I_{SEG} \times N)$$

where:

V+ = supply voltage

DUTY = duty cycle set by intensity register

N = number of segments driven (worst case is 8)

VLED = LED forward voltage

ISEG = segment current set by RSET

Dissipation example:

ISEG = 40mA, N = 8, DUTY = 31/32, VLED = 1.8V at 40mA, V+ = 5.25V

$$PD = (5.25V \times 8mA) + (5.25V - 1.8V)(31/32 \times 40mA \times 8) = 1.11W$$

Thus, for a Cerdip package (θJA = +80°C/W from Table 12), the maximum allowed ambient temperature TA is given by:

$$TJ(MAX) = TA + PD \times \theta_{JA}$$

$$150^{\circ}C = TA + 1.11W \times 80^{\circ}C/W$$

where TA = +61.2°C.

The TA limits for PDIP and SO packages in the dissipation example above are +66.7°C and +55.6°C, respectively.

Cascading Drivers

The example in Figure 3 drives 16 digits using a 3-wire μP interface. If the number of digits is not a multiple of 8, set both drivers' scan limits registers to the same number so one display will not appear brighter than the other. For example, if 12 digits are need, use 6 digits per display with both scan-limit registers set for 6 digits so that both displays have a 1/6 duty cycle per digit. If 11 digits are needed, set both scan-limit registers for 6 digits and leave one digit driver unconnected. If one display for 6 digits and the other for 5 digits, the second display will appear brighter because its duty cycle per digit will be 1/5 while the first display's will be 1/6. Refer to the *No-Op Register* section for additional information.

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

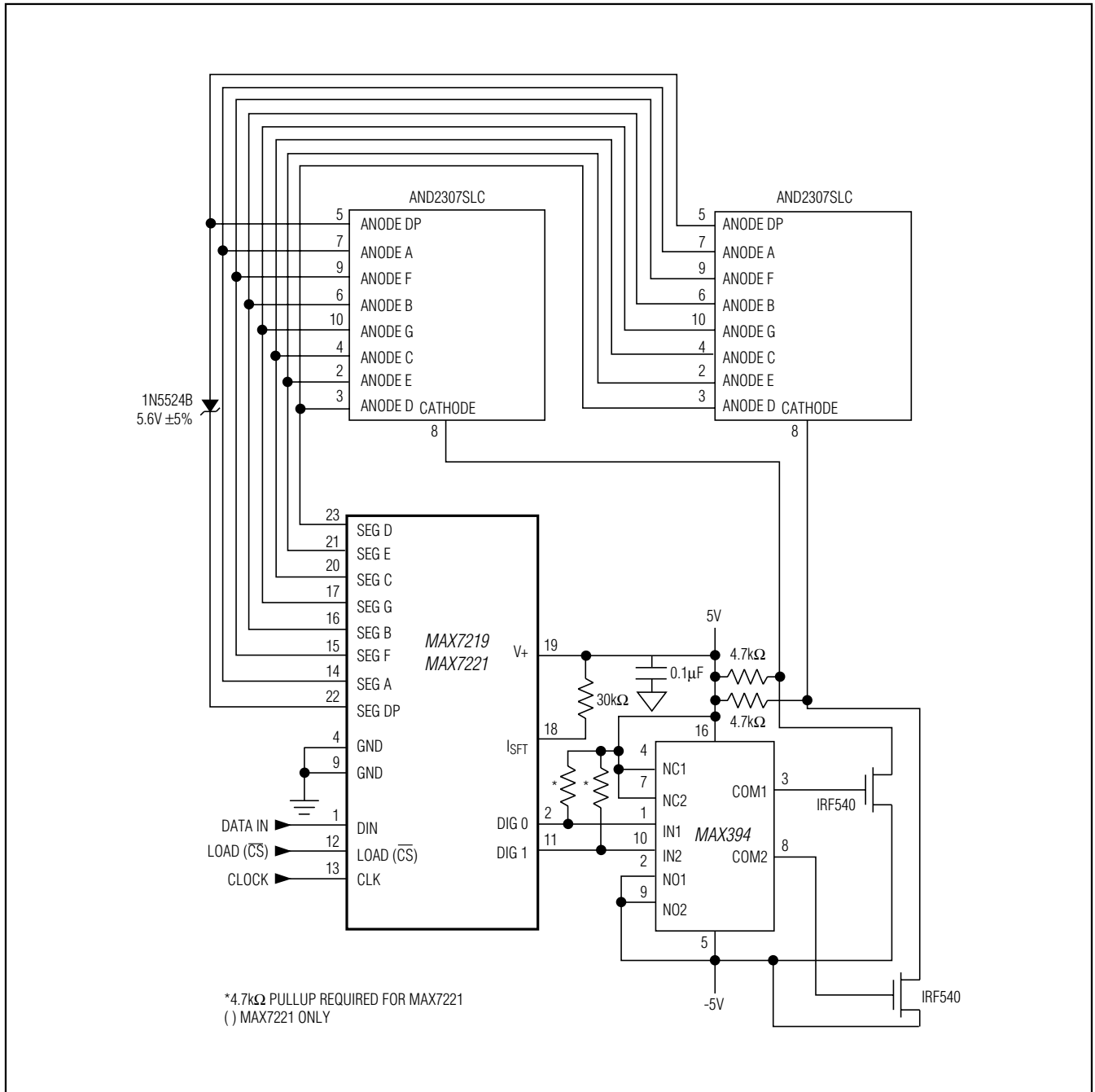


Figure 2. MAX7219/MAX7221 Driving 2.3in Displays

Serially Interfaced, 8-Digit LED Display Drivers

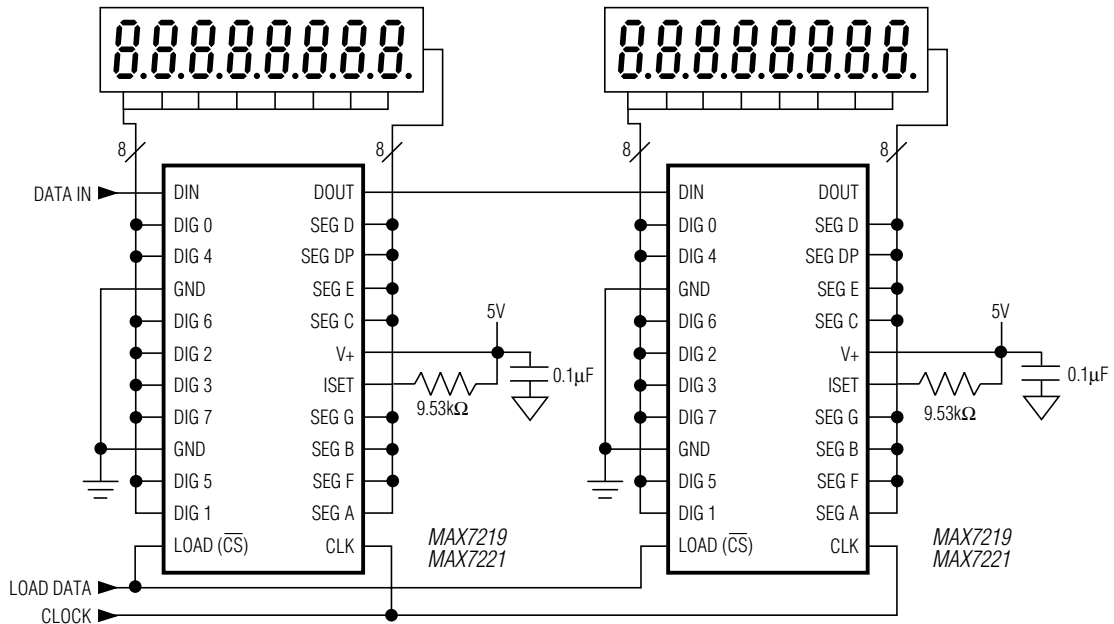


Figure 3. Cascading MAX7219/MAX7221s to Drive 16 Seven-Segment LED Digits

MAX7219/MAX7221

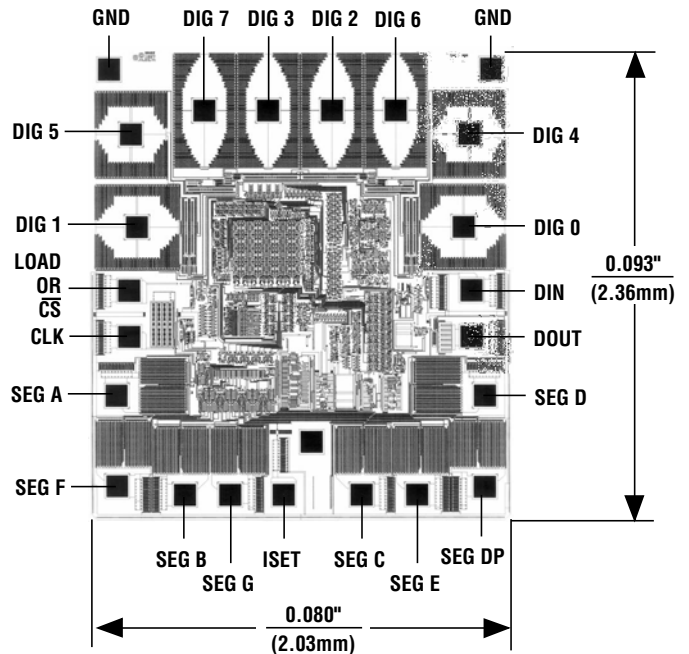
Serially Interfaced, 8-Digit LED Display Drivers

Ordering Information (continued)

PART	TEMP RANGE	PIN-PACKAGE
MAX7221CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX7221CWG	0°C to +70°C	24 Wide SO
MAX7221C/D	0°C to +70°C	Dice*
MAX7221ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX7221EWG	-40°C to +85°C	24 Wide SO
MAX7221ERG	-40°C to +85°C	24 Narrow Cerdip

*Dice are specified at $T_A = +25^{\circ}\text{C}$.

Chip Topography



TRANSISTOR COUNT: 5267

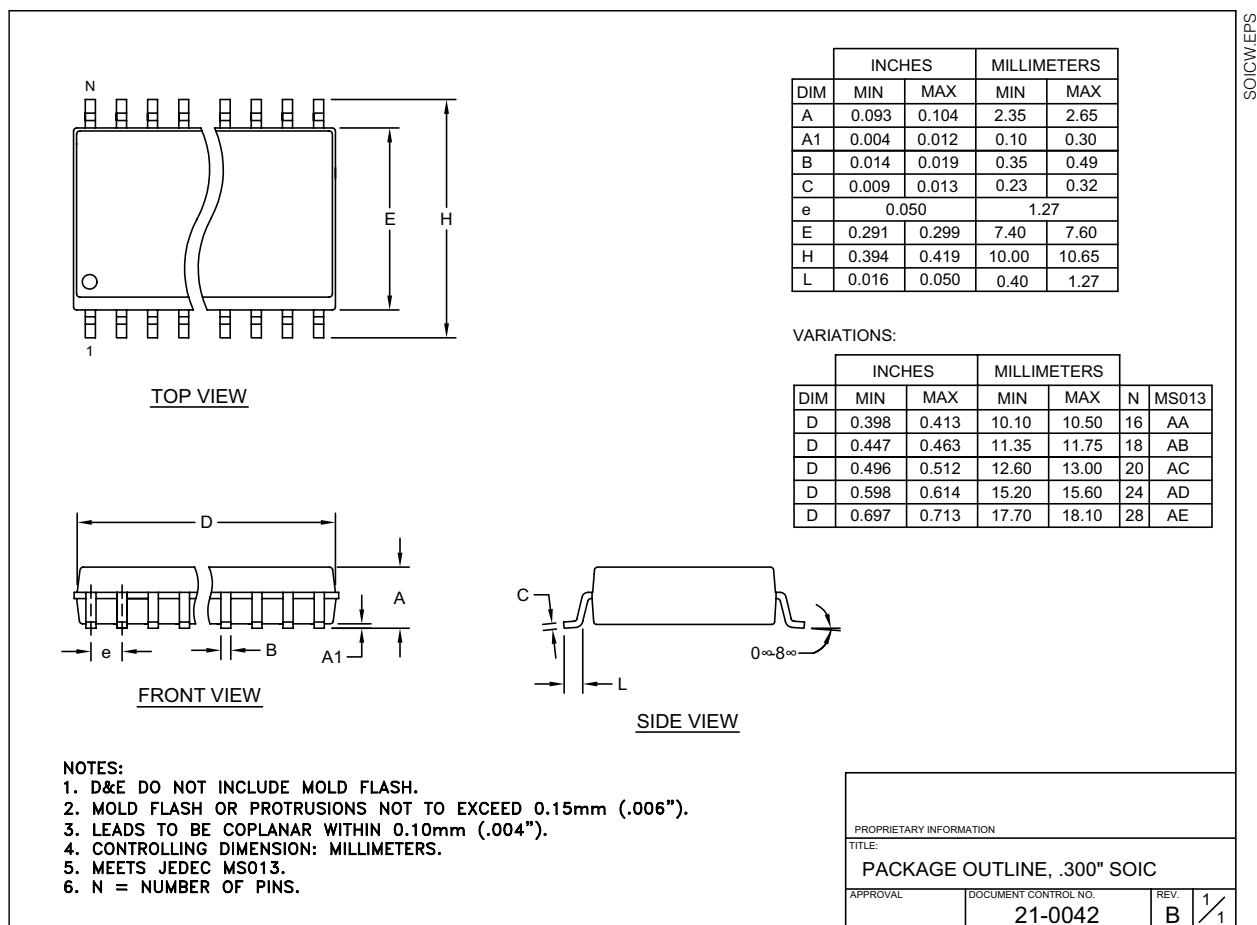
SUBSTRATE CONNECTED TO GND

MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Package Information

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information go to www.maxim-ic.com/packages.)

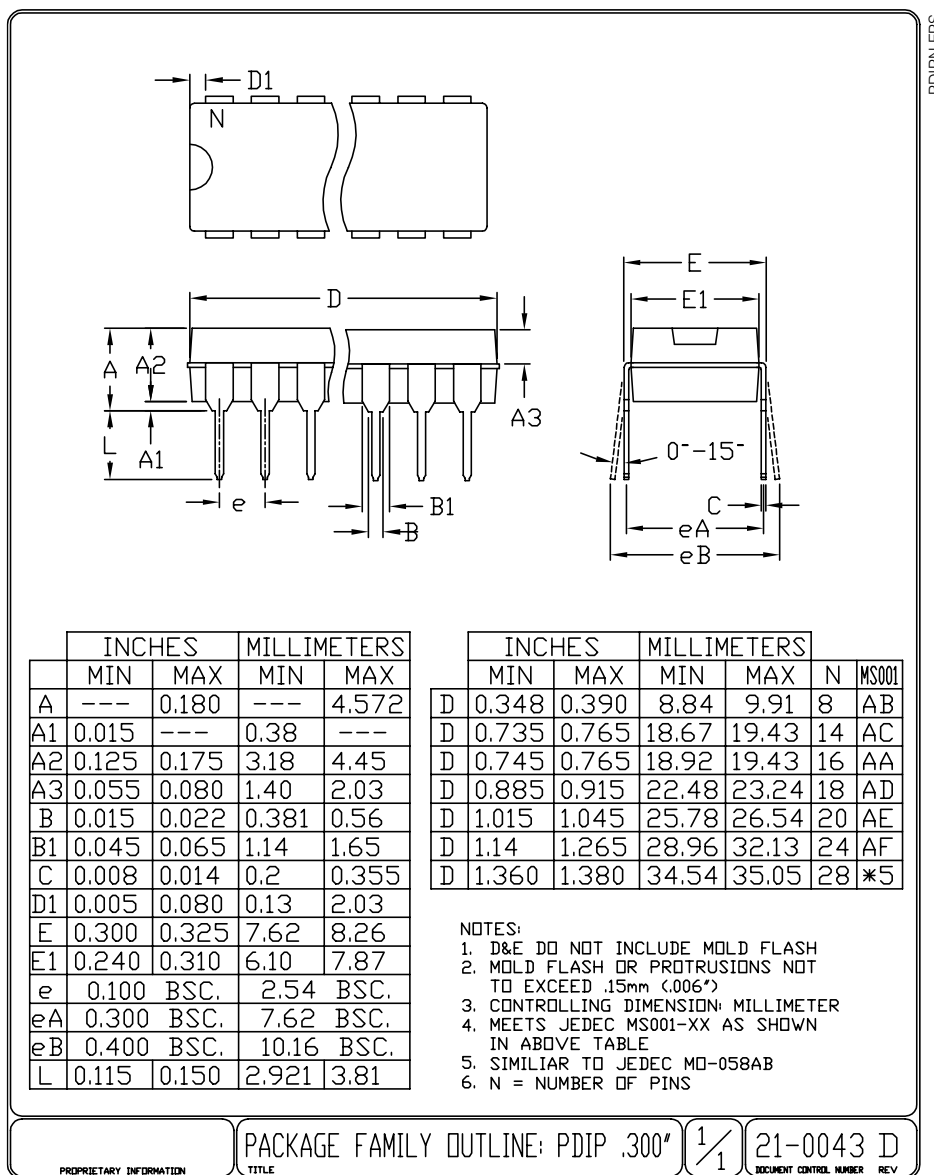


MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

Package Information (continued)

(The package drawing(s) in this data sheet may not reflect the most current specifications. For the latest package outline information go to www.maxim-ic.com/packages.)



MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers



Maxim cannot assume responsibility for use of any circuitry other than circuitry entirely embodied in a Maxim product. No circuit patent licenses are implied. Maxim reserves the right to change the circuitry and specifications without notice at any time. The parametric values (min and max limits) shown in the Electrical Characteristics table are guaranteed. Other parametric values quoted in this data sheet are provided for guidance.

Maxim Integrated 160 Rio Robles, San Jose, CA 95134 USA 1-408-601-1000

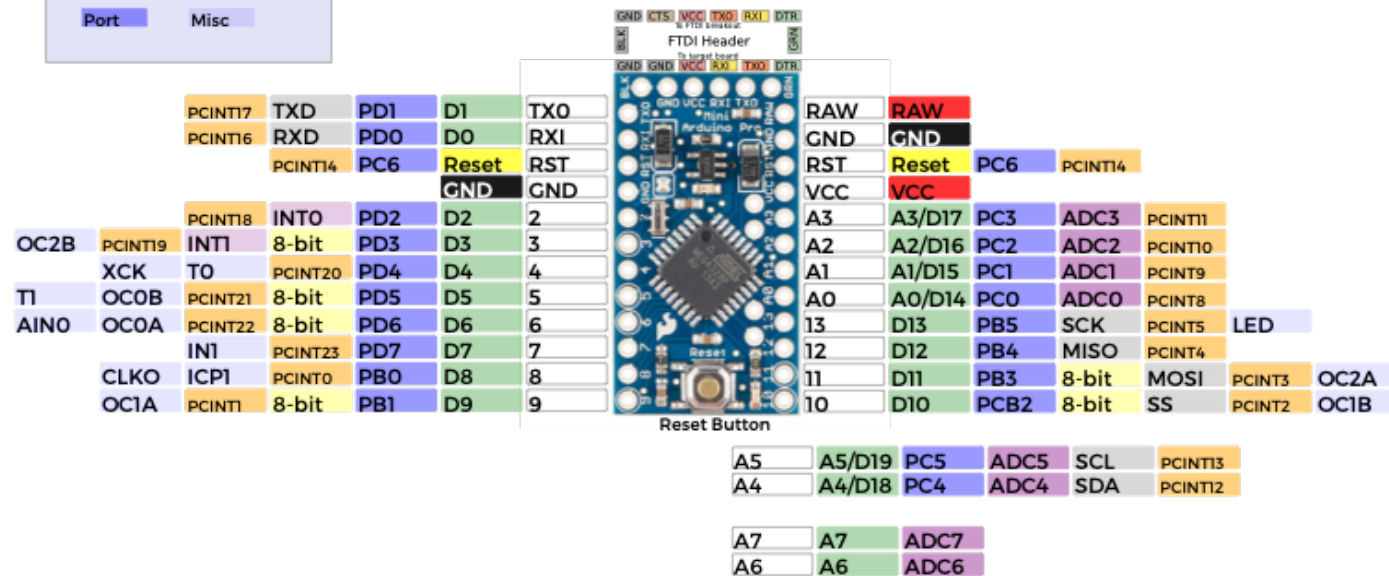
17

Arduino Pro-Mini :

Name	ADC
Power	PWM
GND	Serial
Control	Ext Interrupt
Arduino	PC Interrupt
Port	Misc

Arduino Pro Mini (DEV-11113)

Programmed as Arduino Pro Mini w/ ATmega328
16MHz/ 5V



Power
Raw: 5V-16V (6V-12V recommended)
VCC: 5V
Maximum current: 150mA @ 5V

ATmega328P
Absolute maximum VCC: 6V
Maximum current for chip: 200mA
Maximum current per pin: 40mA
Recommended current per pin: 20mA
8-bit Atmel AVR
Flash Program Memory: 32kB
EEPROM: 1kB
Internal SRAM: 2kB
ADC: 10-bit
PWM: 8bit

LEDs
Power: Red
User (D13): Green



Arduino Uno :

Arduino Uno



Arduino Uno R3 Front

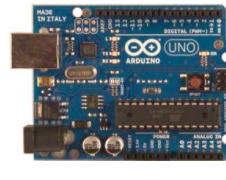
Arduino Uno R3 Back



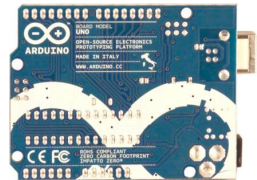
Arduino Uno R2 Front



Arduino Uno SMD



Arduino Uno Front



Arduino Uno Back

Overview

The Arduino Uno is a microcontroller board based on the ATmega328 ([datasheet](#)). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

[Revision 2](#) of the Uno board has a resistor pulling the 8U2 HWB line to ground, making it easier to put into [DFU mode](#).

[Revision 3](#) of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V. The second one is a not connected pin, that is reserved for future purposes.
- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

"Uno" means one in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the [index of Arduino boards](#).

Summary

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V



MAX7219/MAX7221

Serially Interfaced, 8-Digit LED Display Drivers

General Description

The MAX7219/MAX7221 are compact, serial input/output common-cathode display drivers that interface microprocessors (μ Ps) to 7-segment numeric LED displays of up to 8 digits, bar-graph displays, or 64 individual LEDs. Included on-chip are a BCD code-B decoder, multiplex scan circuitry, segment and digit drivers, and an 8x8 static RAM that stores each digit. Only one external resistor is required to set the segment current for all LEDs. The MAX7221 is compatible with SPI™, QSPI™, and MICROWIRE™, and has slew-rate-limited segment drivers to reduce EMI.

A convenient 4-wire serial interface connects to all common μ Ps. Individual digits may be addressed and updated without rewriting the entire display. The MAX7219/MAX7221 also allow the user to select code-B decoding or no-decode for each digit.

The devices include a 150 μ A low-power shutdown mode, analog and digital brightness control, a scan-limit register that allows the user to display from 1 to 8 digits, and a test mode that forces all LEDs on.

For applications requiring 3V operation or segment blinking, refer to the MAX6951 data sheet.

Applications

Bar-Graph Displays Panel Meters
Industrial Controllers LED Matrix Displays

Features

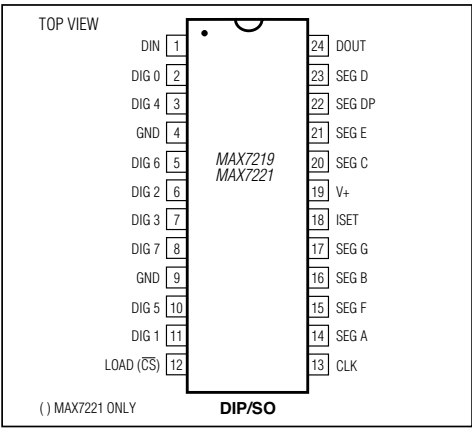
- ◆ 10MHz Serial Interface
- ◆ Individual LED Segment Control
- ◆ Decode/No-Decode Digit Selection
- ◆ 150 μ A Low-Power Shutdown (Data Retained)
- ◆ Digital and Analog Brightness Control
- ◆ Display Blanked on Power-Up
- ◆ Drive Common-Cathode LED Display
- ◆ Slew-Rate Limited Segment Drivers for Lower EMI (MAX7221)
- ◆ SPI, QSPI, MICROWIRE Serial Interface (MAX7221)
- ◆ 24-Pin DIP and SO Packages

Ordering Information

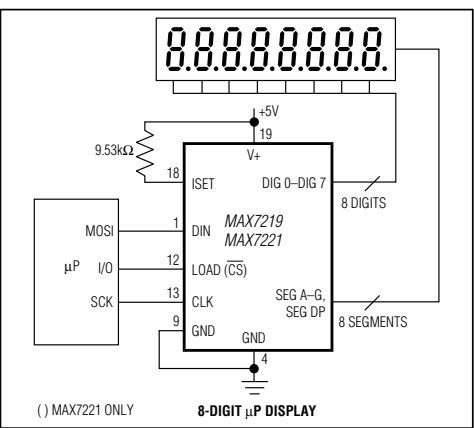
PART	TEMP RANGE	PIN-PACKAGE
MAX7219CNG	0°C to +70°C	24 Narrow Plastic DIP
MAX7219CWG	0°C to +70°C	24 Wide SO
MAX7219C/D	0°C to +70°C	Dice*
MAX7219ENG	-40°C to +85°C	24 Narrow Plastic DIP
MAX7219EWG	-40°C to +85°C	24 Wide SO
MAX7219ERG	-40°C to +85°C	24 Narrow CERDIP

Ordering Information continued at end of data sheet.
*Dice are specified at $T_A = +25^\circ\text{C}$.

Pin Configuration



Typical Application Circuit



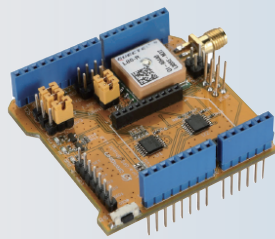
SPI and QSPI are trademarks of Motorola Inc. MICROWIRE is a trademark of National Semiconductor Corp.

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim's website at www.maximintegrated.com.

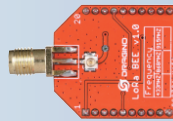


Long Range Wireless Transceiver for Arduino

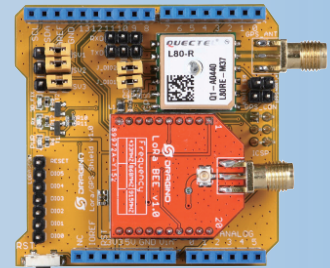
Lora/GPS Shield



Lora/GPS mother board



Lora BEE



Lora/GPS Shield

OVERVIEW:

The Dragino Lora/GPS Shield is an expansion board for LoRa™/GPS for using with the arduino. This product is intended for those interested in developing LoRa™/GPS solutions. The Lora/GPS Shield is composed of Lora/GPS Shield mother board and Lora BEE.

In the Lora part, the Lora/GPS Shield is based on the SX1276/SX1278 transceiver. The transceiver of the Lora/GPS Shield features the LoRa™ long range modem that provides ultra-long range spread spectrum communication and high interference immunity whilst minimising current consumption. LoRa™ also provides significant advantages in both blocking and selectivity over conventional modulation techniques, solving the traditional design compromise between range, interference immunity and energy consumption.

In the GPS part, the addition of L80 GPS (based on MTK MT3339) is designed for applications that require location or timing info. It is connected to the arduino via serial port.

Features:

- Compatible with Arduino Leonardo, UNO, Mega2560, etc
- Frequency Band: one of 433/868/915 MHz (Pre-configure in factory)
- Low power consumption
- FSK, GFSK, MSK, GMSK, LoRa™ and OOK modulation
- Support DGPS, SBAS (WAAS/EGNOS/MSAS/GAGAN)
- GPS support short circuit protection and antenna detection
- Automatic RF Sense and CAD with ultra-fast AFC
- Baud rate configurable

Specification:

Lora Spec

- 168 dB maximum link budget
- +20 dBm - 100 mW constant RF output vs
- +14 dBm high efficiency PA
- Programmable bit rate up to 300 kbps
- High sensitivity: down to -148 dBm
- Bullet-proof front end: IIP3 = -12.5 dBm
- Excellent blocking immunity
- Low RX current of 10.3 mA, 200 nA register retention

GPS Spec

- Based on MT3339
- Compliant with GPS, SBAS
- Programmable bit rate up to 300 kbps
- Update rate: 1Hz (Default), up to 10Hz
- Protocols: NMEA 0183, PMTK
- Horizontal Position Accuracy <2.5 m CEP
- Timing Accuracy: 1PPS out 10ns, Reacquisition Time <1s
- Velocity Accuracy Without aid <0.1m/s, Acceleration Accuracy Without aid 0.1m/s²

Dragino Technology Co., Limited

Room 7009, Zi'an Commercial Building, Qian Jin 1 Road,
Xin'an 6th District, Bao'an District; Shenzhen 518101, China
Direct: +86 755 86610829 | Fax: +86 755 86647123

WWW.DRAGINO.COM
sales@dragino.com

More Information: <https://www.dragino.com/products/module/item/102-lora-shield.html>