

# Microcontrôleurs

Cours – HAE404E – Licence 2 EEA / 2022

Mikhaël MYARA

[mikhael.myara@umontpellier.fr](mailto:mikhael.myara@umontpellier.fr)



*“La plupart des problèmes d'informatique proviennent de l'interface chaise-clavier.”*

Klaus Klages

## Microcontrôleurs - Cours - HAE404E - Licence 2 EEA/2022

11 février 2022

Programmation des Microcontrôleurs en C - Application au STM32.  
par Mikhaël Myara, pour le Department EEA, Université de Montpellier, France



*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.*

# Table des matières

<b>Chapitre I – Présentation des Microcontrôleurs .....</b>	<b>5</b>
I.1 Microprocesseur (CPU) ou Microcontrôleur ( $\mu$ P)?	5
I.2 Alors, qu'est ce qu'il reste à faire?	9
I.3 Choisir un microcontrôleur pour une application	11
I.4 Et maintenant?	13
<b>Chapitre II – Survol "bas niveau" de la plateforme de travail choisie .....</b>	<b>15</b>
II.1 Présentation générale	15
II.2 L'exemple que nous allons suivre	17
II.3 Quelles pins utiliser? Explorons la documentation	17
II.4 Ce que nous allons faire maintenant	19
II.5 Un peu d'architecture	20
II.6 Configurer une sortie tout-ou-rien	23
II.7 Piloter une sortie tout-ou-rien	24
II.8 Le programme complet	25
II.9 Et maintenant?	25
<b>Chapitre III – Sous-ensemble 1 : Entrées et Sorties "tout-ou-rien" .....</b>	<b>27</b>
III.1 Implantation des LEDs : aspect électronique	27
III.2 Implantation des LEDs : aspect informatique	29
III.3 Implantation de l'interrupteur : aspect électronique	29
III.4 Implantation de l'interrupteur : aspect informatique	31
III.5 Le programme complet	32
<b>Chapitre IV – Programmer par bibliothèques .....</b>	<b>34</b>
IV.1 Programmation type "bare-metal"	34
IV.2 Bibliothèques pour microcontrôleurs	34
IV.3 Un premier pas : "S'abstraire" de la Memory Map	35
IV.4 Vers plus d'abstraction : utilisation de CMSIS + ST Standard Library	37
IV.5 Autres Bibliothèques	41
<b>Chapitre V – Sous-ensemble 2 : Gestion du temps et Timers .....</b>	<b>42</b>
V.1 Constante de temps associée à un processeur : "CPU clock"	42
V.2 Principe de fonctionnement des Timers	43
V.3 Que faut-il faire concrètement?	44

V.4 Un exemple : une fonction pour "attendre" un temps calibré	44
<b>Chapitre VI – Sous-ensemble 3 : Gestion des signaux analogiques .....</b>	<b>46</b>
VI.1 Entrée analogique : ADC (Analog to Digital Converter)	46
VI.2 Sortie analogique type DAC (Digital to Analog Converter)	49
VI.3 Sortie analogique type PWM (Pulse Width Modulation)	50
VI.4 Comparaison DAC et PWM	54
<b>Chapitre VII – Sous-ensemble 4 : Interruptions Externes et Périodiques ..</b>	<b>56</b>
VII.1 Le problème à résoudre	56
VII.2 La bonne solution avec un microcontrôleur : "Interruption externe"	57
VII.3 Réalisation sur le STM32L152	58
VII.4 Interruptions périodiques	60
VII.5 Les interruptions sont partout!	61
<b>Chapitre VIII – Chaîne complète : ADC▶ traitement▶ DAC/PWM .....</b>	<b>63</b>
VIII.1 Contexte	63
VIII.2 Signaux échantillonnés	64
VIII.3 Gestion avec un microcontrôleur	65
VIII.4 Codage PWM de signaux variant dans le temps	65
VIII.5 DAC ou PWM : la suite!	67
<b>Chapitre IX – Sous-ensemble 5 : Communication .....</b>	<b>69</b>
<b>Annexe A – Documentation de la carte STM32L152C-Dicovery .....</b>	<b>71</b>
<b>Annexe B – Datasheet du microcontrôleur STM32L152RCT6 .....</b>	<b>110</b>
<b>Annexe C – Extraits du "Reference Manual" du STM32L152RCT6 .....</b>	<b>247</b>
<b>Annexe D – Documentation d'autres composants .....</b>	<b>280</b>
<b>Annexe E – Signaux tout-ou-rien : pas assez de tension ou de courant? ..</b>	<b>304</b>
E.1 Introduction	304
E.2 Le transistor en mode "tout-ou-rien" (saturé-bloqué)	304

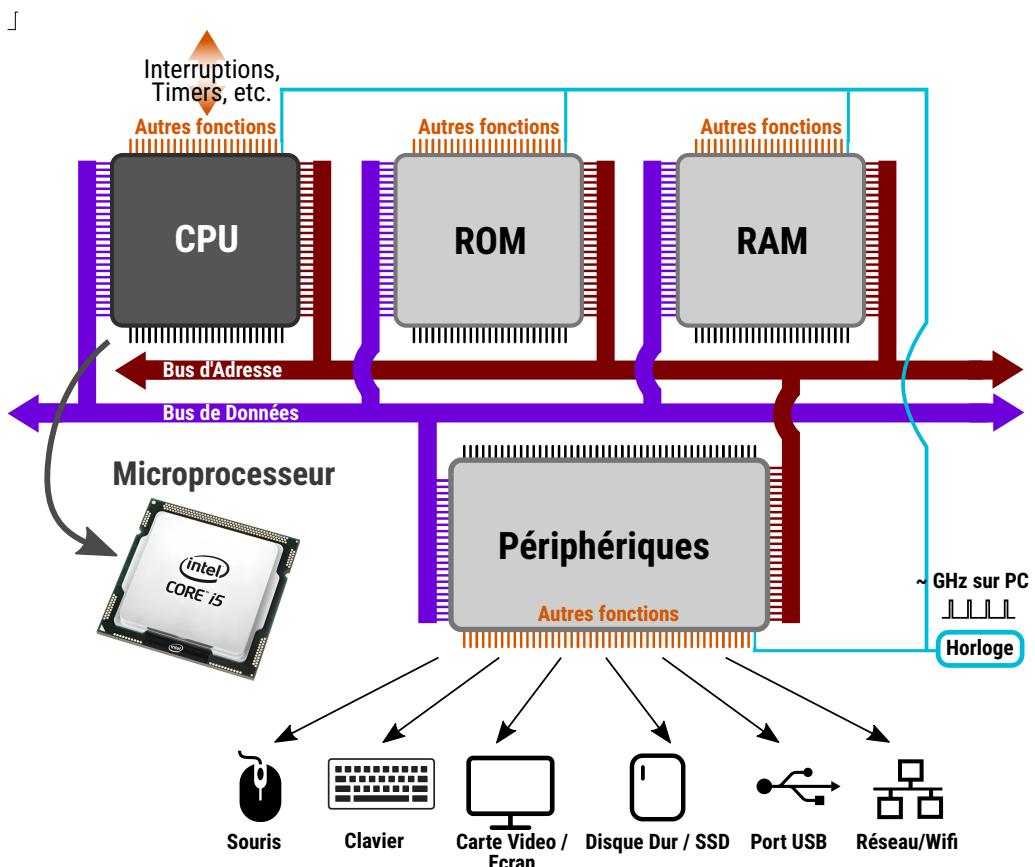
# Chapitre I – Présentation des Microcontrôleurs

## ① Motivations et objectifs

Dans ce chapitre, nous allons situer les microcontrôleurs : à quoi ils servent en général, comment ils se positionnent par rapport aux microprocesseurs des ordinateurs. Nous allons aussi survoler ce à quoi ressemble l'environnement d'un microcontrôleur dans une application concrète, et discuterons rapidement de la question difficile de "comment choisir un microcontrôleur" pour une application donnée.

### I.1 Microprocesseur (CPU) ou Microcontrôleur ( $\mu$ P) ?

Ce que vous utilisez de "plus proche" d'un microcontrôleur est le "**microprocesseur**", présent au sein des ordinateurs/PC. Sans forcément trop le savoir, vous l'utilisez en permanence quand vous utilisez un ordinateur, parce que c'est lui qui produit tous les calculs et déclenche la plupart des actions qui se déroulent quand vous utilisez l'ordinateur. Cette puce est simplement celle qui réalise les calculs et les échanges d'information à l'intérieur de l'ordinateur. Les "microprocesseurs" sont souvent appelés "CPU" pour "Central Processing Unit". Voici un schéma très simplifié qui montre comment un microprocesseur est intégré dans un ordinateur :



On voit donc que le microprocesseur est essentiellement relié aux autres composants ou périphériques par des "bus" : c'est ce qui lui permet d'échanger de l'information avec les autres composants (on expliquera pourquoi un peu plus loin). L'architecture représentée ici, avec un seul bus d'adresses et un seul bus de données, est commune à la plupart des CPU et se nomme "**architecture von Neumann**". Nous verrons qu'il existe d'autres façons, pour un CPU, de discuter avec les autres composants, mais celle qui est à l'oeuvre dans les PC est très proche de ce qui est montré ici.

Dans ce contexte, les rôles principaux d'un CPU sont donc :

- Effectuer des calculs, des opérations mathématiques par exemple, booléennes, etc
- Echanger des informations avec les composants reliés à la carte mère : RAM, ROM, périphériques. Par exemple stocker temporairement une valeur dans la RAM ou écouter le clavier pour savoir si une touche a été tapée, etc.

Il ne faut pas perdre de vue que **c'est le CPU qui exécute les programmes**, et donc toutes les lignes de codes que vous pouvez écrire, avec un langage comme le C ou l'assembleur notamment, s'adressent précisément à lui : c'est donc "de son point de vue" qu'il faut se placer pour programmer, et toujours s'interroger sur "comment le CPU voit son environnement". Et donc, notamment, c'est toujours le CPU qui impose l'état (la valeur) présent sur le bus d'Adresse et, selon l'opération, il va imposer aussi l'état du bus de Données (pour "envoyer une information") ou lire l'état du bus de données (pour "relever" une information).

## Important

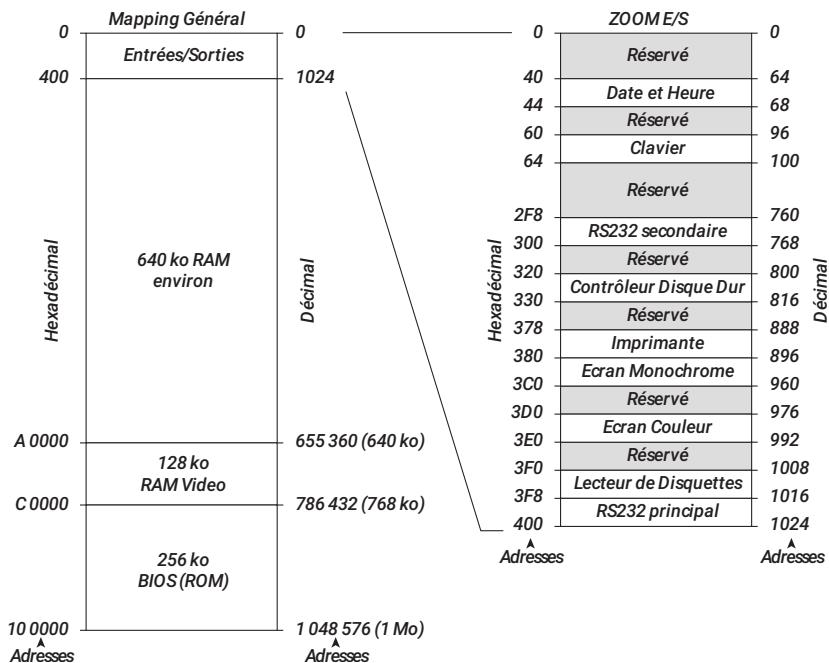
Le CPU décide donc, de par le programme qu'il exécute, de quelles informations sont à lire ou à écrire dans les différentes puces, grâce à l'adresse qu'il impose sur le bus d'adresse. Pour synchroniser les différentes puces avec les changements d'état des bus, une "horloge" envoie des impulsions, que tous les composants reçoivent simultanément. Ainsi, à chaque "coup" de l'horloge du bus, l'ensemble bus d'adresse et bus de données sont dans un certain état, qui correspond à un échange d'information, cette information faisant au maximum la taille du bus (de nos jours 8 octets, soit 8 "bytes", ou encore 64 bits). Pour échanger "plus", il faut plusieurs coups d'horloge. ■

Un ordre de grandeur : sur un PC dont le processeur est à 3 ou 4 GHz, la fréquence du bus est à une fréquence de l'ordre de 1 GHz de nos jours (soit  $10^9$  transferts possibles par seconde) ; autrement dit un processeur peut typiquement transférer, en lecture ou écriture, jusqu'à 8 Go/s avec un autre composant, ce qui est beaucoup. Comparez par exemple avec le débit d'une connexion à internet même par fibre optique (de l'ordre de 10 Mo/s, soit environ 1000 fois moins rapide<sup>1</sup>).

Cette organisation fait que tous les composants (RAM, ROM et périphériques) sont accessibles au CPU à travers une certaine organisation de "**L'espace adressable**", c'est à dire l'ensemble des adresses accessibles par le processeur : en 64 bits il y a  $2^{64} \approx 10^{19}$  adresses possibles, ce qui est monstrueux : en effet, sur un PC, le composant qui utilise le plus d'adresses est souvent la RAM, et nous avons, même sur un assez gros PC, typiquement 32 Go de RAM, soit  $\approx 10^{10}$ . Cela signifie que la taille du bus d'un PC 64 bits pourrait gérer un milliard ( $10^9$ ) de fois plus de RAM que ce qu'il ne le fait aujourd'hui.

"L'espace adressable" est souvent représenté par sa "cartographie de la mémoire", appelée en anglais "**memory map**". Ci-dessous un exemple, celui de l'IBM PC de 1988, qui correspond à une part (infime) de ce que les PC modernes contiennent encore aujourd'hui :

1. Cette différence est complètement liée aux distances concernées : internet concerne des grandeurs en km et bien au delà, une carte mère concerne des distances de l'ordre de 10 cm



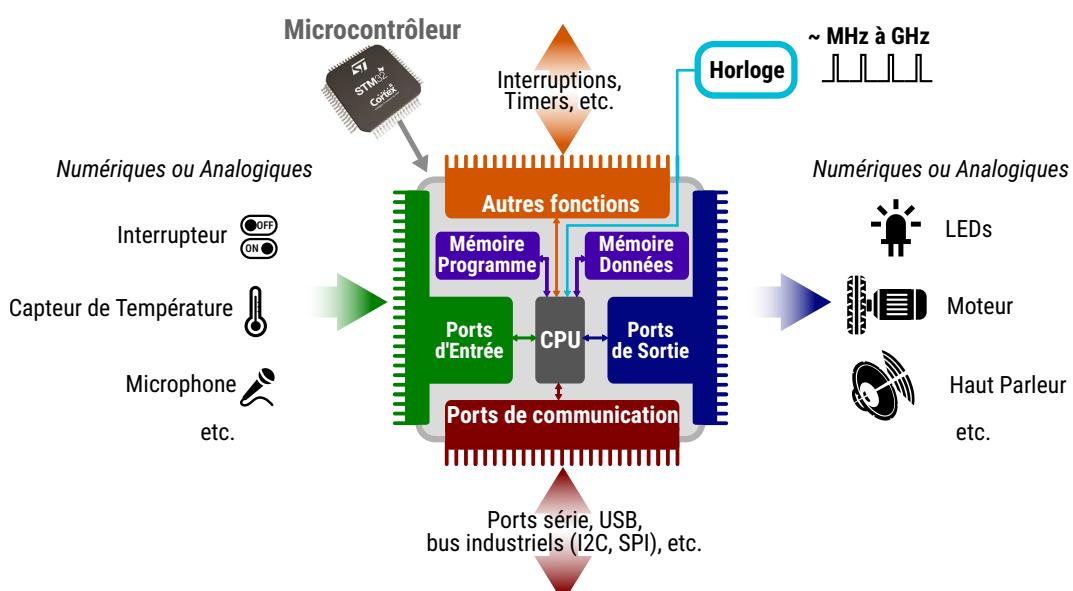
On voit bien qu'à chaque composant correspond une plage d'adresses, qui a une signification (à décrire au cas par cas). Par exemple les valeurs contenues dans la partie "Date et Heure" contient quelque chose qui est interprétable pour donner l'heure qu'il est, selon un certain codage (qui n'est pas notre sujet ici).

## Important

A ce stade, il faut avoir compris que vu d'un CPU, et donc vu d'un programme, la RAM, la ROM et les différents périphériques individuels sont simplement localisés par des adresses : avec le programme, il suffit donc de "taper" à telle ou telle adresse pour échanger des informations avec un périphérique donné : à partir de là, c'est l'électronique elle-même qui prend le relai pour réaliser les actions utiles. Le document qui rend compte de la position des différents périphériques en mémoire et qui décrit la façon d'y accéder est la "**Memory Map**". Il faut aussi noter que la "**Memory Map**" est spécifique à une catégorie d'ordinateur. Par exemple, elle est différente entre un PC et un Mac.

## Microcontrôleur : CPU + fonctionnalités

Parlons maintenant des **microcontrôleurs**, et voyons les similarités et différences.



Un microcontrôleur est aussi une puce électronique qui sait réaliser des calculs : elle contient également un CPU. Mais elle ne se limite pas à cela et contient bien d'autres choses, parce qu'elle est conçue pour s'intégrer dans un **système électronique embarqué**. Cela signifie qu'elle est faite pour pouvoir **interagir assez directement avec "le monde réel"**. C'est la différence essentielle avec un simple CPU, ce qui en fait un "contrôleur" plutôt qu'un simple "processeur". Un microcontrôleur contient donc, comme c'est représenté sur la figure ci-après :

- un **CPU**, pour les calculs,
- de la **mémoire** pour les programmes et pour les données, des choses qui s'apparentent à la RAM et à la ROM qui accompagnent un CPU,
- des **ports d'entrée** qui gèrent permettent **l'acquisition de signaux**, numériques ou analogiques,
- des **ports de sortie**, qui permettent de piloter **des actionneurs**, numériques ou analogiques,
- des **ports de communication**, qui permettent d'échanger de l'information, pour remplir différents rôles. Par exemple : échanger avec un autre microcontrôleur (c'est même assez fréquent), gérer des entrées/sorties qui ont été conçues pour fonctionner avec ces ports, piloter des périphériques particuliers, etc.
- il dispose également d'**autres fonctionnalités**, qu'il partage avec les CPU, pour gérer par exemple le temps ou "les interruptions" (qui seront discutées plus loin).

Les ports d'entrée, de sortie, de communication, ainsi que les autres fonctionnalités forment des **sous-ensembles** du microcontrôleur, que nous traiterons, chacun tour à tour, dans ce cours.

Un Microcontrôleur est donc un composant électronique qui contient les fonctions d'un CPU enrichies par des fonctionnalités qui permettent de travailler directement avec des capteurs, des actionneurs, ou divers périphériques. Cela lui donne la possibilité de fonctionner au coeur d'un circuit électronique pour prendre, grâce à son programme, des décisions liées à des informations issues de capteurs, décisions qui vont amener à produire des actions. Ce besoin concerne une très large variété d'applications : cela peut consister simplement à réguler la température d'un chauffage en mesurant la température d'une pièce, et aller jusqu'à gérer le pilotage automatique d'un avion ou d'une voiture.



## Important

Un microcontrôleur sert donc, dans beaucoup d'applications, à piloter des actionneurs en fonction de ce qu'il perçoit sur ses capteurs, par le biais du programme qu'il contient. On peut comprendre le microcontrôleur comme une puce électronique "prête à l'emploi" pour échanger de l'information avec des composants externes, analogiques ou numériques, et pour laquelle on va devoir programmer un certain "arbitrage", des "actions" à réaliser, de façon "intelligente" ou à minima "automatique". ■

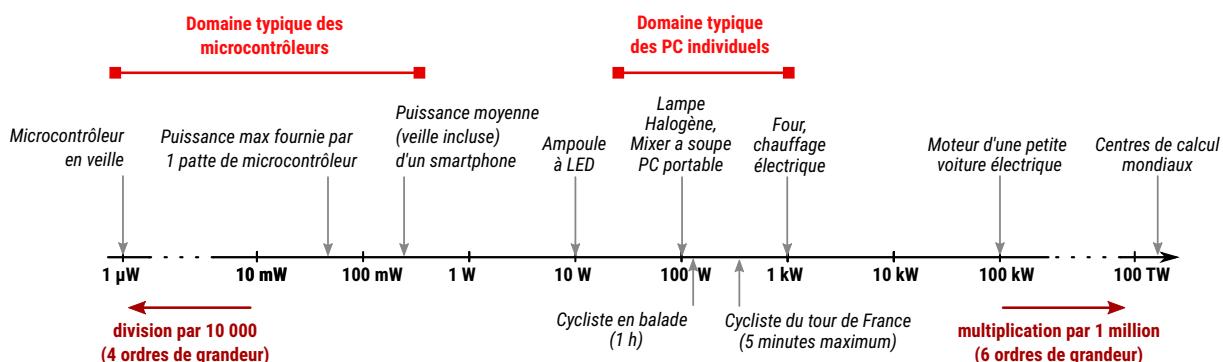
### Microcontrôleur : souvent faible consommation

Il y a une deuxième différence que l'on constate souvent : **un microcontrôleur contient en général une puissance de calcul assez inférieure à celle d'un microprocesseur**. Ca n'est pas "obligatoire", il n'y a pas de raison "de structure" pour cela, mais c'est l'usage de le constater. Cette différence provient surtout de l'aspect "embarqué" des microcontrôleurs.

En effet, si on reprend le cas des microprocesseurs de PC, la puissance de calcul d'un microprocesseur est très liée à sa consommation électrique : la puissance de calcul coûte de l'énergie. Et donc, quand on fabrique un PC, on a besoin de puissance de calcul et la priorité est donnée à cela : on ajuste la quantité d'énergie au besoin. Par exemple, le microprocesseur seul d'un PC un peu puissant d'aujourd'hui peut consommer des puissances de quelques centaines de Watt, ce qui commence à être beaucoup pour une machine qui "ne produit aucun travail visible", au sens où elle ne gère "que" de l'information, elle ne déplace rien, n'éclaire pas, ne réchauffe ou refroidit pas, etc (ou alors ce ne sont que des effets de bord, non souhaités). A titre de comparaison, avec 300 Watt on peut déjà alimenter un réfrigérateur, un mixer à

soupe, ou un sèche cheveux, c'est à dire des objets qui "font quelque chose" pour de vrai. Pour information, la totalité des centres de calculs du globe consomme environ 500 Tera Watt (soit  $\approx$  un million de fois plus qu'un PC) ...

Maintenant, reprenons le cas du microcontrôleur : on a souvent besoin de l'embarquer dans des contextes où l'énergie n'est pas si abondante que ça. Par exemple dans une voiture ou pire dans un drone, on travaille sur batterie et l'énergie est limitée. Et dans tous les cas, les opérations que l'on a besoin de faire nécessitent rarement (mais pas "jamais", ça pourrait être le cas) la puissance de calcul d'un PC. Et donc le compromis visé par le microcontrôleur, en ce qui concerne sa puissance de calcul, n'est pas le même : on préfère souvent plus d'autonomie, un faible coût et moins de puissance de calcul. On trouve par exemple des microcontrôleurs qui savent se mettre en veille pour descendre à des niveaux de consommation  $< 1\mu A$  (oui vous lisez bien !), et pour prendre un chiffre de la vie courante, on peut considérer la consommation moyenne d'un téléphone portable dans les  $100mW$ . Cette fois, on est des milliers, des millions voire des milliards de fois plus faibles en consommation électrique qu'un PC. L'objectif est donc très différent.



### Microcontrôleur : gestion du temps et actions "en parallèle"

Pour finir, nous en reparlerons, un microcontrôleur est fait pour réaliser diverses tâches simples mais "simultanément", et son architecture est adaptée à cela. En effet, la plupart des microcontrôleurs utilisent une architecture de type "**Harvard**" ou "**Harvard modifiée**", plus complexe que l'architecture "von Neumann" dont nous avons déjà discuté. En première approximation, cela revient à dire qu'il n'y a pas un seul bus d'adresse et de données, mais plusieurs, séparés et indépendants, et donc avec un peu de finesse dans les programmes on peut réaliser des communications en parallèle, à l'intérieur ou à l'extérieur du microcontrôleur, avec différents composants ou fonctions du microcontrôleur.

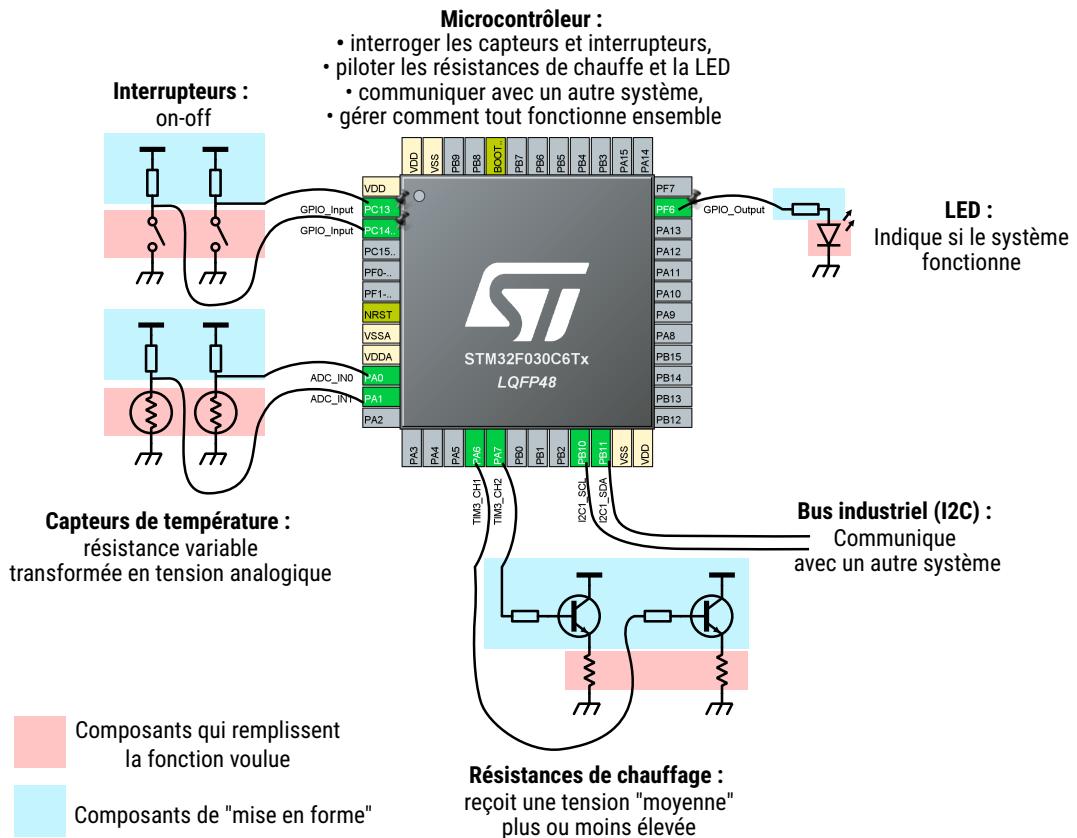
## I.2 Alors, qu'est ce qu'il reste à faire ?

On pourrait penser qu'il reste juste à "écrire des programmes", puisque nous avons insisté sur le fait que "tout se passe au niveau du CPU". Mais nous ne travaillons pas sur un simple microprocesseur : nous travaillons sur un microcontrôleur. Et donc ce que nous faisons a à voir avec l'électronique.

Il y a donc forcément une partie "hardware". Il faut notamment :

- choisir le microcontrôleur par rapport aux contraintes de l'application visée,
- choisir les composants externes
- au minimum relier les composants (capteurs, actionneurs, ...) au microcontrôleur, "savoir à quelle patte du microcontrôleur" relier les composants
- probablement aussi faire un peu d'électronique pour que les signaux émis par le composant soient compatibles avec ce que le microcontrôleur peut comprendre, ou réciproquement, adapter par de l'électronique les signaux du microcontrôleur pour qu'ils deviennent compatibles avec le composant à piloter. Rien qu'en regardant le graphe des puissances un peu plus haut, on voit bien que le microcontrôleur ne pourra pas piloter tout ce qu'on peut vouloir piloter et qu'il faudra des circuits d'adaptation.

Rien que ça ! Ensuite seulement on peut commencer à programmer. Prenons un exemple simple :



Sur cet exemple, commençons par identifier les entrées, les sorties, et ce qui sert à communiquer.

- Les entrées sont : les deux interrupteurs et les capteurs de température. Ce sont les composants que le microcontrôleur pourra, par programmation, interroger, et nous verrons comment.
- Les sorties sont les résistances de chauffage et la LED.
- Un "bus industriel", qui sert à faire de la communication avec un autre composant (qui peut être n'importe quoi : un capteur, un actionneur, un autre microcontrôleur, etc).

Cette carte est donc destinée à faire chauffer des résistances en fonction de l'état des capteurs de température (1 capteur pour chaque résistance), et selon l'état des interrupteurs (on peut allumer ou éteindre le chauffage). Le système indiquera qu'il fonctionne en allumant la LED. Ce fonctionnement, c'est bien entendu exprimé par le programme que l'on va écrire pour le microcontrôleur, qui va lui l'exécuter. Mais on voit bien qu'il y a "des choses en plus" que juste le programme sur ce schéma déjà simplifié.

On voit déjà que les pattes ont des fonctionnalités. Faisons un tour rapide :

- Les interrupteurs sont reliés à des pattes marquées comme étant des "**GPIO\_Input**". C'est donc une entrée (input), et cette entrée est une entrée numérique, "tout ou rien", sensible à une tension élevée pour représenter un état "haut", ou à une tension faible pour représenter un état "bas". Cet état est lisible "quelque part" dans la "memory map" du microcontrôleur.
- Les capteurs de température sont reliés à des pattes notées "**ADC\_IN**", c'est à dire à des "convertisseurs analogique-numérique". Autrement dit, ce sont des pattes qui sont des voltmètres : elles transforment une certaine tension, à l'entrée, qui représente ici la température vue par le capteur, en une valeur numérique lisible par le microcontrôleur. Cette valeur est lisible "quelque part" dans la "memory map" du microcontrôleur.
- les résistances de chauffage sont reliées à des **TIM**. Dans la notation ici ca n'est pas parfaitement clair et nous y reviendrons. Mais il s'agit de ce que l'on appelle une sortie **PWM** (Pulse Width

Modulation) qui est une façon de créer des sorties analogiques. Autrement dit, ces pattes sont plus ou moins des générateurs de tension pilotables par une valeur : on peut donc imposer par programmation, avec le microcontrôleur cette valeur de tension, grâce à un ensemble de cases mémoire, "quelque part" dans la "memory map" du microcontrôleur.

- La LED est reliée à une patte nommée "**GPIO\_Output**", c'est à dire à une "sortie numérique", en tout-ou-rien. Encore une fois, cet état peut être imposé "quelque part" dans la "memory map" du microcontrôleur.
- Un bus industriel de type "**I2C**" a été câblé aussi, pour communiquer avec d'autres composants ou microcontrôleurs qui n'ont pas été représentés ici. Là encore, piloter le bus I2C se fait "quelque part" dans la "memory map".

Enfin, un ensemble de composants, en plus des capteurs et actionneurs que l'on voulait, ont été ajouté autour du microcontrôleur : ce sont des composants de "**mise en forme**". Que sont ces composants ? Ils servent à faire en sorte que le composant soit "connectable" au microcontrôleur, de façon à ce que, par exemple, les niveaux de tension correspondent à ce que le microcontrôleur peut accepter, et que les niveaux de courants soient suffisants pour les différents composants. Le meilleur exemple, sur ce schéma, sont les résistances de chauffage : Pour avoir une puissance de chauffe importante, il faut à la fois une tension élevée et un courant élevé. Des choses comme par exemple 10 A et 10 V si on veut 100 Watt. Un microcontrôleur est complètement incapable de fournir des courants et des tensions aussi élevés. Il peut fournir "quelques volts" (en général 5V maximum, voire 3V de plus en plus) et "quelques mA" (10mA c'est déjà appréciable). Et donc il faut ajouter, ici, un transistor lui-même alimenté pour fournir ce qu'il faut à la résistance de chauffe.



### Important

Ainsi, on va devoir s'intéresser à deux aspects :

- Un **aspect électronique/hardware** : Comment "interfacer" les fonctions voulues avec le microcontrôleur, par quelle électronique on le fait ? Ca peut être très simple comme ici ou aller vers des choses assez compliquées, qui peut vous demander tout le savoir que vous avez acquis en électronique, analogique ou numérique, par ailleurs,
- Un **aspect logiciel** : Comment on utilise la "memory map" pour dialoguer avec ces composants via les ports du microcontrôleur. Les zones de mémoires associées à une fonctionnalité précise d'un microcontrôleur s'appellent des **registres**.

Nous allons traiter ces deux aspects dans la suite.



## I.3 Choisir un microcontrôleur pour une application

C'est un sujet compliqué et on ne pourra pas vraiment le traiter aussi tôt. Mais certaines choses peuvent déjà ce comprendre :

1. La première chose à vérifier c'est si le microcontrôleur dispose des fonctionnalités dont on a besoin, et en nombre suffisant. Par exemple, ça semble idiot, mais il doit disposer des bons types d'entrées/sorties nécessaires (tout-ou rien, analogique, BUS, etc.) et dans le bon sens (entrée ou sortie), des bons protocoles de communication (I2C, SPI, ...), et tout ça dans la bonne quantité. Et cela bien entendu, c'est le cahier des charges de l'application qui le définit.
2. Combien il consomme ? C'est une question qui se pose, comme la consommation de l'ensemble de la carte, selon si le projet en cours doit fonctionner sur batterie ou pas. Vous devez savoir que les microcontrôleurs "modernes" disposent de fonctionnalités dédiées à l'économie d'énergie. Par exemple : diminuer sa fréquence d'horloge quand "il ne fait rien d'exigeant", et la remettre à une valeur plus importante quand il doit produire du calcul. Et tout cela peut s'activer ou se désactiver par programmation, selon le modèle choisi.

3. Ensuite se pose souvent la question de la quantité de mémoire et de la puissance de calcul nécessaire. C'est un sujet difficile et même des professionnels peuvent se tromper. C'est difficile parce que ça dépend de beaucoup de choses : d'abord est-ce que le problème à traiter est "intensif" en termes de calcul, est-ce qu'il faut beaucoup de mémoire ? Ca suppose que l'on a déjà une idée de ce qu'il faut, et ça c'est l'expérience que l'on a en tant que programmeur. Et un deuxième aspect, toujours lié à notre propre expérience, est notre capacité, individuelle, à écrire des programmes qui s'exécuteront de façon plus ou moins rapides : c'est la question de "l'optimisation du code".
4. Et la puissance de calcul nous amène directement à une question : 8, 16, 32, ou même 64 bits (rare aujourd'hui pour les microcontrôleurs, standard dans les PC). C'est la question sur laquelle on va un peu s'attarder.

### 16 bits? 32 bits? ... 64 bits?

Comme pour un CPU, cette information donnée seule signifie en général (mais pas toujours) "la taille du bus d'adresses". C'est à dire le nombre de cases mémoire dont on peut disposer en tout, y compris les registres de la "memory map". Par exemple si on prend un microcontrôleur 16 bit, l'adresse la plus grande possible est :

$$\text{max\_adr} = 2^{16} - 1 = 65535$$

Comme chaque case fait 1 octet, on peut donc adresser  $\approx 65,5$  ko de mémoire. Cela ne signifie pas que l'on a  $\approx 65,5$  ko de RAM disponibles. Déjà parce qu'elle n'est pas forcément présente (il faut regarder la documentation pour le savoir), mais aussi parce que tous les registres associés aux différents périphériques utilisent des emplacements mémoire, et il peut y en avoir beaucoup (d'autant plus que le microcontrôleur a beaucoup de fonctionnalités).

Mais ce 16 bits signifie souvent aussi la taille du bus de données du CPU. Et donc cela signifie que le CPU peut transférer d'un seul coup 2 octets au maximum (16 bits = 2 octets). Cela suppose donc une capacité à transferer des données d'un registre vers la RAM ou l'inverse assez limitée, ce qui limite donc la vitesse de réaction du microcontrôleur, par exemple c'est 2 fois moins qu'un 32 bits et encore 4 fois moins qu'un 64 bit.

Et donc, "en général", puisqu'il est de toute façon limité sur sa capacité à transferer les données entre la RAM et les registres, un processeur 16 bits n'est pas fait pour être aussi rapide, y compris pour d'autres raisons liées à sa puissance de calcul elle-même, qu'un processeur 32 bit.

On peut donc penser, même si ça n'est pas une règle absolue, qu'un processeur 64 bits est plus rapide qu'un processeur 32 bit, lui-même plus rapide qu'un 16 bits et lui-même plus rapide qu'un 8 bit.

Maintenant il faut faire rentrer, dans le choix, l'état de la technologie et son histoire. Aujourd'hui, la technologie est telle qu'il n'est pas plus difficile de fabriquer un processeur 32 bits qu'un processeur 16 bits. Mais un 32 bits utilise plus de surface de silicium qu'un 16 bits (nécessairement, il y a 2 fois plus de "bits" à gérer simultanément). Mais la surface de silicium nécessaire est aussi liée au nombre de pattes autour du microcontrôleur. C'est aujourd'hui la limite la plus importante, et pour cette raison on tend à aller vers des processeurs 32 bits et à abandonner les 8 et 16 bits, sauf pour des raisons historiques (habitudes de travail, remplacer des composants dans des systèmes existants sans "tout refaire", etc.)

Autrement dit, sauf indication contraire et compte tenu de leur performance (y compris sur la gestion de la consommation électrique), il n'y a plus beaucoup d'arguments technologiques qui iraient dans le sens d'utiliser un processeur autre que 32 bits de nos jours pour une nouvelle application. Ainsi, la plupart des nouveaux projets, à l'époque à laquelle nous vivons (années 2020+), devraient se baser sur des microcontrôleurs 32 bit.

### Fréquence de l'horloge

C'est un autre paramètre qui influence beaucoup la puissance de calcul du microcontrôleur, celui qui a été le plus "poussé" dans les PC jusqu'en 2010 : un nouveau PC plus rapide était surtout un nouveau PC avec une fréquence d'horloge plus rapide. De façon *très approximative*, la période liée à la fréquence d'horloge définit le temps qui est nécessaire au processeur pour exécuter une opération élémentaire, comme copier une valeur ou lire une information à une adresse donnée. Pour des opérations

plus complexes, comme calculer la valeur d'un sinus ou d'une exponentielle, il faut plusieurs "coups" de cette horloge. Cela signifie que sur des opérations simples, un PC à 2GHz peut réaliser environ 2 milliards d'instructions basiques par seconde, ce qui est absolument considérable.

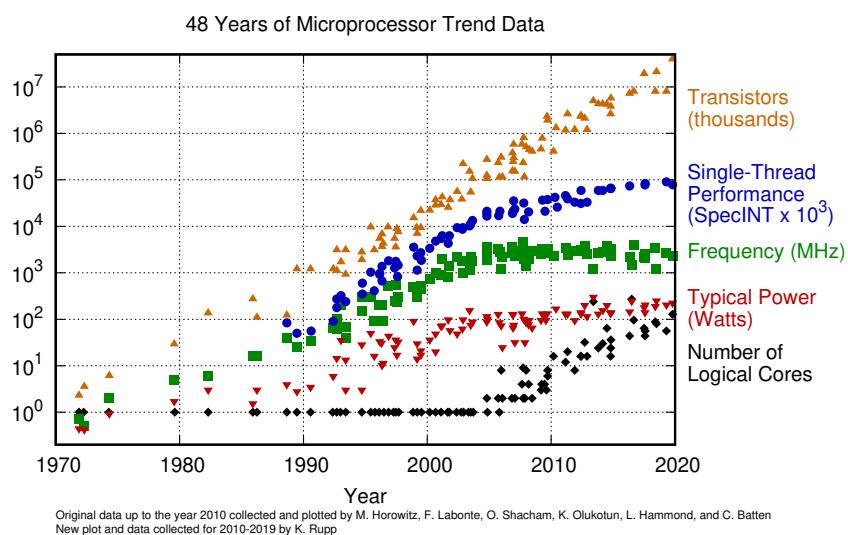
### Cœurs de calcul

Les progrès sur la puissance de calcul des PC est aujourd'hui très liée au fait que le PC dispose de plusieurs "cœurs" de calcul (montrer une copie d'écran). En effet, il est difficile aujourd'hui d'augmenter la puissance de calcul en augmentant la fréquence, les technologies en vogue ne le permettent plus trop. Utiliser plusieurs coeurs signifie que les calculs peuvent être répartis en plusieurs programmes qui fonctionnent en parallèle, comme si il y avait "plusieurs CPU indépendants" : en effet, un cœur de calcul ne peut exécuter qu'une seule instruction à la fois. Aujourd'hui un CPU de PC dispose typiquement de entre 4 et 16 coeurs de calcul. Cela signifie aussi que le programme qui fonctionne est écrit pour s'adapter à ce fonctionnement. La plupart des programmes n'utilisent en réalité qu'un cœur de calcul, et l'ensemble des coeurs n'est utilisé qu'assez rarement, dans des applications intenses comme les jeux vidéo ou le calcul scientifique. L'exemple phare aujourd'hui est les processeurs graphiques, "GPU", qui eux contiennent des milliers de coeurs de calculs, mais très spécialisés.

Sur un microcontrôleur, il est assez rare de disposer de plusieurs coeurs de calcul, aussi c'est quelque chose que nous n'aborderons pas.

### Synthèse sur la puissance de calcul (cas des CPU de PC)

On peut se faire une idée de l'évolution de la puissance de calcul et des moyens pour y parvenir en regardant l'évolution des PC depuis les années 1970, comme le montre le graphe ci-dessous :



On voit que la puissance de calcul "brute" (Single-Thread Performance) pour un cœur de calcul sature depuis une quinzaine d'années. C'est très lié au fait qu'il n'est plus tellement possible d'augmenter la fréquence de travail des processeurs, pour des raisons liées au principe physique et aux matériaux employés. Et on voit très bien que cette saturation est compensée par l'augmentation du nombre de coeurs. Notez que ça n'est pas exactement équivalent, ça suppose que les logiciels s'adaptent et que les calculs à faire sont "parallélisables" sur plusieurs coeurs (certains ne le sont pas). Mais dans le cas des applications vraiment intensives en calcul cela est en général presque équivalent.

## I.4 Et maintenant?

Dans le chapitre suivant, nous allons décrire la plateforme de travail qui a été choisie pour cet enseignement. Une fois cette description faite, nous aborderons différents sujets : entrées/sorties numériques, analogiques, communication par un bus industriel, interruptions et gestion du temps.

## ❖ En résumé ...

Nous avons vu qu'un microcontrôleur n'a pas la même vocation qu'un microprocesseur : le microcontrôleur **contient** un microprocesseur, pour gérer l'aspect calcul, échange de données et gestion du temps, mais comporte aussi beaucoup d'autres fonctionnalités qui lui permettent d'interagir avec des capteurs et actionneurs notamment, pour s'ancrer dans "le monde réel".

Les programmes que l'on va réaliser ont donc en général pour vocation d'organiser la lecture des capteurs et déclencher les bonnes actions sur les actionneurs. Aussi, un microcontrôleur peut communiquer, via un bus industriel, avec un autre système, peu importe sa nature, par exemple un autre microcontrôleur. Certains disposent aussi de pattes pour gérer des protocoles de haut niveau comme l'USB ou la connexion éthernet.

Nous avons également vu que nous avons deux objectifs différents à remplir pour utiliser un microcontrôleur :

- un aspect **hardware**, qui consiste à créer les circuits électroniques qui rendent "compatibles" les capteurs et actionneurs que l'on veut avec les entrées et sorties du microcontrôleur
- un aspect **logiciel**, qui consiste à écrire le programme qui remplit les bonnes fonctions au bon moment et dans le bon ordre, en interrogeant les capteurs et déclenchant les bonnes actions quand c'est utile.

Nous avons vu que la relation entre l'aspect électronique et l'aspect logiciel se fait par l'intermédiaire de mémoires appelées **registres**. Ces registres sont situés "quelque part en mémoire", et tout cela est résumé dans ce que l'on appelle la **memory map** : il s'agit de "taper" à la bonne adresse pour obtenir la bonne action.

Aussi, nous avons constaté qu'il y avait une grande diversité de d'entrées et sorties possibles (GPIO, ADC, PWM, bus industriels, ...) et nous allons apprendre comment utiliser ces fonctionnalités.

Enfin, nous avons discuté la question difficile de la puissance de calcul et de la mémoire nécessaire, et avons conclu qu'il n'y avait pas beaucoup d'arguments pour utiliser autre chose qu'un microcontrôleur 32 bits dans la plupart des applications en 2020. ■

## Chapitre II – Survol "bas niveau" de la plateforme de travail choisie

### II.1 Présentation générale

Nous nous baserons sur un **STM32L152RCT6**, qui est un microcontrôleur "dans l'air du temps" de chez "STMicroelectronics" qui répond à la spécification "**ARM**" ("Advanced RISC<sup>1</sup> Machine") modèle **Cortex M3** et que nous programmerons en C.



Cette spécification ARM, qui date des années 80-90, a vu des évolutions très marquantes (performance, autonomie) dans les années récentes, puisqu'elle est à la base de la plupart des processeurs de téléphone portable ou de tablettes aujourd'hui, et commence à s'imposer dans les ordinateurs depuis peu. Par exemple Apple a commercialisé les premiers ordinateurs à base d'ARM, en lieu et place des processeurs Intel, dans les années 2020. Parallèlement, toujours en ce qui concerne l'informatique grand public, Microsoft a écrit une version de Windows pour ARM, et développé un ordinateur pour fonctionner avec (Windows 11 est officiellement supporté). Et sans trop de surprise, Linux existe depuis longtemps sur plateforme ARM, par exemple sur Raspberry, mais aussi pour des systèmes serveur de calcul qui utilisent de plus en plus des ARM pour des raisons de consommation.

Mais il existe beaucoup d'autres catégories de microcontrôleurs. Par exemple les Arduino n'utilisent pas du tout un ARM mais un processeur de technologie plus "traditionnelle" (Atmel / Microchip). Pour "nous" qui programmons en C, ça ne change pas l'état d'esprit dans lequel on va écrire le code : cela change surtout l'accès aux fonctionnalités (donc aux registres), qui ne sont pas les mêmes en fonction des plateformes, et l'environnement de développement. Mais l'état d'esprit sera le même, il est surtout plus proche de ce que fait l'industrie avec un STM32 qu'avec un Arduino.

Il faut avoir en tête qu'un ARM Cortex M3 est déjà un microcontrôleur bien puissant qui permet de faire énormément de choses : ce ne sont pas du tout, dans cet enseignement, les performances du microcontrôleur qui nous limiteront, et ce serait déjà vrai avec un microcontrôleur bien moins puissant. Pour situer, sur la plupart des opérations, de l'ordre de 10 fois plus puissant, en puissance de calcul brute, qu'un ordinateur courant des années 90 ! Mais pour relativiser, comparons le à un PC "moyen" (mais complet : CPU + carte mère) aujourd'hui :

1. La technologie "RISC" est une technologie entérinée dans les années 80, qui vise à réduire le nombre d'instructions du langage machine pour accélérer les processeurs. Il faut se référer à des notions du cours d'architecture pour comprendre cela, c'est "un sujet" en soi. Cette évolution est très liée au fait que l'assembleur est un langage qui a été de moins en moins utilisé à partir des années 80, et que les compilateurs type compilateur C n'avaient pas besoin de tout un tas d'instructions disponibles alors.

	Horloge	RAM	Stockage des logiciels	Interfaces
PC	> 3 GHz	> 8 Go	[disque dur] > 200 Go	USB, éthernet, WiFi, Bluetooth, HDMI, ...
STM32L100RCT6	16 MHz (32 MHz max)	32 ko	[flash] > 256 ko	USB, SPI, I2C, GPIO, ADC, PWM, LCD, ...
Rapport : PC / STM32L100RCT6	$\approx 10^3$	$> 10^5$	$\approx 10^6$	-

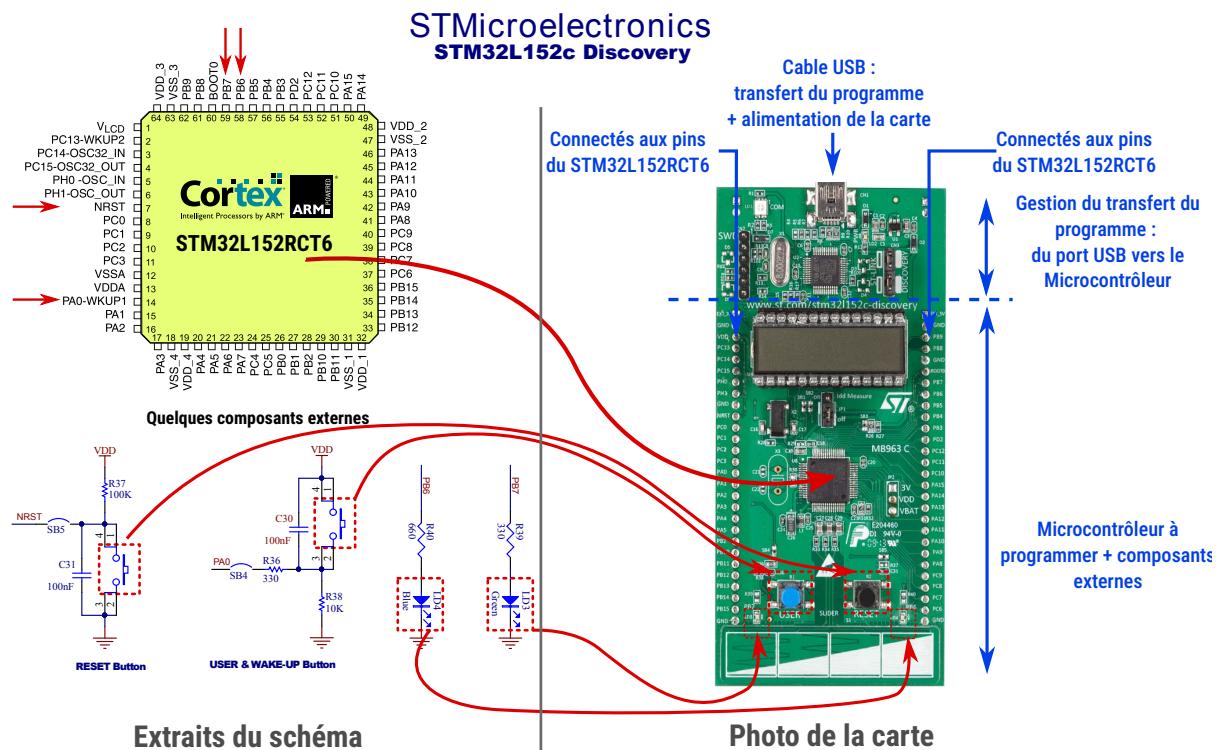
Cela peut sembler être incroyable comme rapport de performance : de façon très approximative, on peut dire qu'un PC est "en gros" 1000 fois plus rapide que notre microcontrôleur, et contient 100 000 fois plus de mémoire. Et c'est assez vrai en pratique. Mais ce qu'il faut réaliser, c'est que la puissance de calcul d'un PC est juste incroyable : rien que pour pouvoir exécuter un Windows récent, il faut quelque chose de l'ordre de cette puissance de calcul là. Mais ça n'est pas du tout ce que l'on va demander au microcontrôleur de faire, et donc il est parfaitement dimensionné (et même en réalité souvent surdimensionné) pour les applications classiques.

Pour nous, ce microcontrôleur sera utilisé à travers un "kit de développement", c'est à dire une carte déjà créée qui permet aux "développeurs", c'est à dire les gens qui créent de nouvelles applications, de prendre rapidement le composant en main sur une carte qui fonctionne déjà. Au final, pour nous, cela revient à dire qu'il sera déjà positionné sur une carte électronique avec "le minimum vital" pour le faire fonctionner.

Le kit que nous utiliserons a pour nom **STM32L152C Discovery Kit**. Sa documentation complète est disponible ici : <http://www.st.com/stm32l152c-discovery>, mais aussi en dans l'annexe A de ce document. Le schéma électrique est situé pages 102 à 107 de ce document.

La carte met notamment à disposition le nécessaire pour programmer le microcontrôleur à travers un port USB, et câble 2 interrupteurs et 2 LEDs déjà interfacés avec le microcontrôleur :

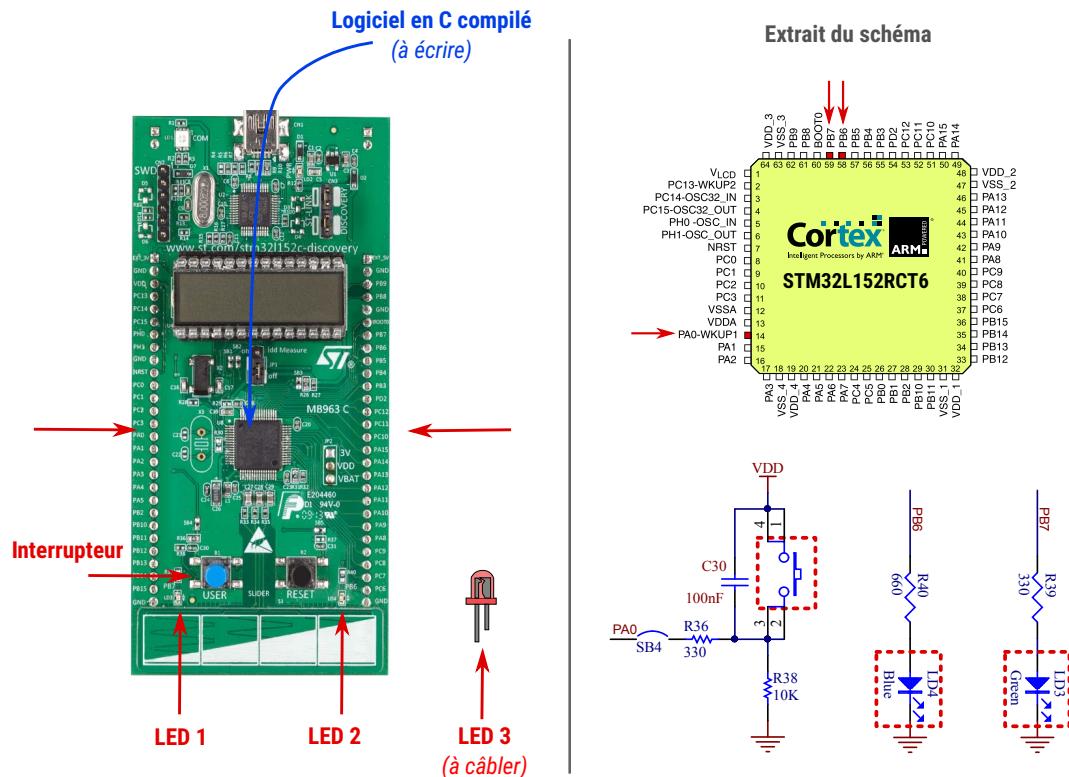
Discovery Kit User Manual  
disponible annexe A



D'autres composants sont également câblés sur la carte, (comme le suggère la photo il y a au moins un afficheur), mais ne sont pas décrits ici.

## II.2 L'exemple que nous allons suivre

On va commencer par chercher à réaliser une application simple : à chaque appui sur un interrupteur, on allumera la première des 3 LED disponibles, un nouvel appui allumera la suivante, et ainsi de suite, de façon cyclique. On part donc de la situation ci-dessous :



où le schéma a été pris dans annexe A de ce document page 102 et suivantes.

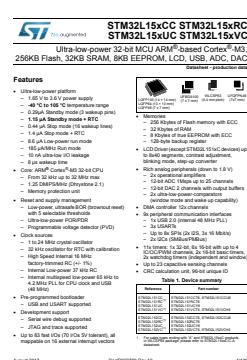
Mais pour faire ça, comme on l'a annoncé, il faut que l'on comprenne comment ce qui a été câblé a été câblé, quels choix ont été faits, de façon à pouvoir ajouter la 3ième LED nous mêmes. Et il y a déjà une question angoissante : à quel pin (et comment) va-t-on pouvoir connecter notre nouvelle LED ? Alors regardons tout ça.

## II.3 Quelles pins utiliser ? Explorons la documentation

Une fois que le microcontrôleur a été paramétré, chaque "pin" du microcontrôleur est associée à une fonctionnalité précise. On l'a déjà dit timidement, ces fonctionnalités vont consister à disposer d'entrées ou de sorties numériques ou analogiques, ou encore de ports de communication, ou d'autres fonctionnalités diverses. On doit donc déjà choisir des pin qui sont adaptées à cette fonction : toutes ne font pas "tout". Et cela ca ne s'invente pas, on le trouve dans la documentation du microcontrôleur. Et cela nous amène donc à considérer une documentation supplémentaire, celle du STM31L152RCT6, qui est donnée annexe B.

La première fois que l'on a à faire à ce genre de documentation, cela peut sembler difficile à lire, déjà parce que c'est en anglais, et ensuite parce qu'il y a beaucoup d'acronymes inconnus. Mais il faut bien avoir en tête que beaucoup d'informations ne seront pas utiles pour la plupart des projets et

**STM32L15xRC Datasheet**  
disponible annexe B



This is information on a product in development.

Document ID: STM32L152RCT6RM Rev 1.0

Version 1.0, 12/03/2012

on ne va donc pas s'atteler à lire cette documentation en entier (ce serait fou).

On va essayer déjà de regarder la table des matières (reproduite page 112 de ce document) et voir ce que l'on peut comprendre. La voici, simplifiée et commentée :

**1 - Introduction,**

**2 - Description,**

**3 - Functional overview** : c'est intéressant et utile, ça parle des fonctionnalités du microcontrôleur.

Ce qui peut attirer notre oeil, à ce niveau, c'est :

- d'abord le terme "**GPIO**", qui est un terme générique du jargon des microcontrôleurs, qui signifie "General Purpose Input Output", autrement dit "Entrée/Sortie d'utilité générale". En gros : une entrée/sortie "à tout faire". Tout ? Pas exactement. En fait les GPIO sont des pattes du microcontrôleur que l'on peut souvent paramétriser pour obtenir telle ou telle fonctionnalité. C'est donc un point de départ intéressant. Si on suit la documentation pour aller à la page en question ([page 133](#)), on lit :

*"Each of the GPIO pins can be configured by software as output (push-pull or open-drain), as input (with or without pull-up or pull-down) or as peripheral alternate function. Most of the GPIO pins are shared with digital or analog alternate functions, and can be individually remapped using dedicated AFIO registers".*

Autrement dit : ce sont des ports/pins que l'on peut configurer par "soft" (donc en programmant) pour remplir une fonction de "sortie numérique" (c'est ce qui se cache derrière "push-pul" or "open-drain"), "entrée numérique" (c'est ce qui se cache derrière "with or without pull-up or pull-down"), ou encore comme une "fonction alternative", qui peut être plein de choses : port USB, PWM, ou enfin comme "patte analogique" en entrée ou en sortie. Nous verrons certaines de ces choses par la suite. Et enfin, le document dit que "ces fonctions alternatives ou le mode analogique peuvent être configurés en utilisant des registres AFIO", où "AFIO" est encore un jargon qui signifie "Alternate Function Input/Output". Bref en résumé, on dispose de tout plein de "pins" que l'on peut plus ou moins configurer pour remplir la fonctionnalité que l'on veut par programmation. On fera ça plus loin dans ce document.

- On voit aussi "Memories" qui décrit la mémoire vive (SRAM) et la mémoire pour les programmes (Flash) en termes de quantité,
- Et puis on voit diverses choses : DMA, LCD, ADC, DAC, operational amplifier, comparators, communication interfaces, timers, etc. Tout cela correspond à des fonctionnalités du microcontrôleur. On reviendra sur certaines d'entre elles.

**4 - Pins description** : la première partie de cette section indique juste à quelle position, sur le composant, se situe chaque patte, identifiée par son nom. Pour la version de la puce que nous avons, c'est page 144. Mais ça ne nous intéresse pas puisqu'on utilise une carte sur laquelle le microcontrôleur est déjà soudé : ce travail est déjà fait pour nous et il faut plutôt regarder le schéma de la carte en réalité.

En revanche la seconde partie ([page 148](#) et suivantes) nous intéresse beaucoup : elle indique les fonctions remplies par chaque pin ! c'est un très grand tableau (Tableau 9) dont une colonne représente le "pin name" (nom de la pin), le type de pin (Entrée/Sortie, Entrée seulement, Alimentation, etc.), et les "fonctions alternatives" qu'elle peut remplir (alternative functions). Ainsi, on voit bien que toutes les pattes ne peuvent pas remplir toutes les fonctions possibles et il faudra donc choisir. Mais pour ce qui concerne les entrées ou sorties "tout-ou-rien", toutes celles qui sont identifiées "Entrée/Sortie" le peuvent, et elles sont très nombreuses. Donc on va pouvoir utiliser presque n'importe quelle patte. Bonne nouvelle !

**5 - Memory Mapping** : ça on sait déjà que ce sera utile et on peut y jeter un coup d'oeil, [page 161](#). On voit à quelles adresses de la mémoire se situent certaines fonctionnalités. C'est "un peu grossier" comme information (c'est un "survol" du mapping) pour le moment mais c'est un début, et ça nous sera utile quand on commencera à programmer.

- 6 - **Electrical characteristics** : ca aussi c'est intéressant, c'est une partie de la documentation qui parle des niveaux de tension et de courants acceptés, consommés, générés, etc par le microcontrôleur, pour l'ensemble des pins. Ca va donc nous concerner pour la LED : on va devoir savoir si, pour une LED donnée, le courant fourni par le microcontrôleur est suffisant pour l'éclairer. Ces informations sont données [page 162](#) et suivantes.
- 7 - **Package information** : concerne la géométrie de la puce, c'est à dire la position des pattes autour, sa taille, etc. Cela sert pour créer des cartes électroniques, mais pour nous ça n'est pas utile, la puce est déjà montée sur une carte.
- 8 - **Part Numbering** : explique quel nom précis porte le composant en fonction des "options" que l'on choisit. Nous avons déjà le composant, on ne s'y intéressera pas.
- 9 - **Revision History** : historique des évolutions de cette puce.

### Et donc, quelle pin pour une LED ?

Interessons nous d'abord aux LEDs déjà câblées. Comme on peut le lire [dans la documentation de la carte](#) sur le **schema de la carte** [page 104](#) (coordonnée B-4), et comme cela est aussi [confirmé par le tableau 8](#) [page 98](#) :

- la pin **PB6** est utilisée pour la LED bleue,
- la pin **PB7** est utilisée pour la LED verte,

Regardons maintenant [dans la documentation du microcontrôleur](#), dans la partie "**Pins description**", [page 155](#) (en haut). On voit que la pin **PB6** et la pin **PB7** ont bien la capacité d'être des "Entrée/Sortie" (notée **I/O**), et donc elles sont utilisables comme entrées ou sorties tout-ou-rien (et donc pour nous ce sera une entrée, mais c'est par le code que nous le paramètrons). On peut voir aussi que c'est le cas de la plupart des autres pin, et on décidera d'utiliser la **PB8** pour câbler la LED rouge dans notre exemple.

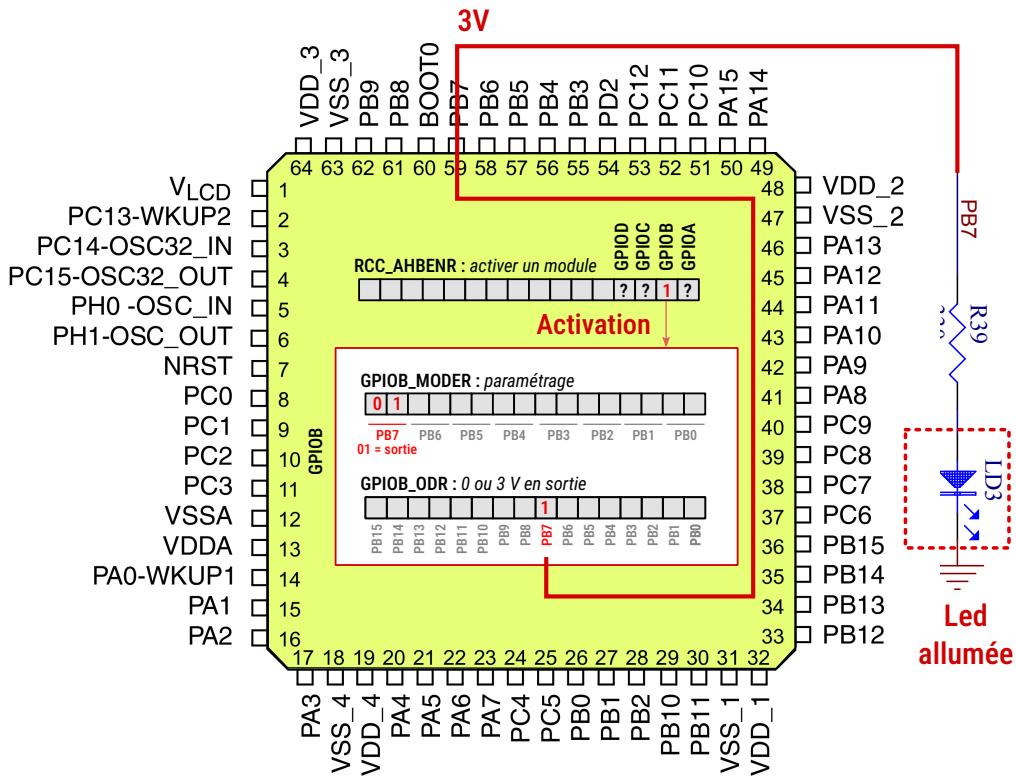
La simple recherche de "quelle patte utiliser" nous a amené assez loin mais nous en savons maintenant beaucoup plus sur le composant. On va donc maintenant pouvoir regarder d'un peu plus près ce que l'on cherche à faire.

## II.4 Ce que nous allons faire maintenant

Voici le fil directeur de ce que nous allons faire. Si on parcourt la documentation, on se rend compte que pour piloter la LED sur PB7, il faut faire 3 choses, à travers 3 registres différents :

- activer le port **GPIOB** dans le registre **RCC\_AHBENR**. Cette activation est nécessaire parce que le STM32 nous permet de n'activer que ce qui est utile dans le microcontrôleur pour notre application : cela réduit la consommation si nous n'utilisons pas la totalité des fonctionnalités (et il y en a beaucoup).
- paramétrier le port **GPIOB** pour que **PB7** soit considérée comme une sortie (on parle de son "mode" de fonctionnement). En effet, les pins du microcontrôleur peuvent endosser différentes fonctionnalités en fonction du besoin, et cela se paramètre, pour le **GPIOB** dans le registre **GPIOB\_MODER**.
- utiliser le **GPIOB** pour définir la tension de sortie de **PB7** comme valant 0V ou 3V. Cela se fait avec le registre **GPIOB\_ODR**.

Ces opérations sont représentées ci-dessous :



## II.5 Un peu d'architecture

Continuons à regarder le cas de la LED verte, autrement dit de la patte **PB7**. On pourrait se dire, et ce serait un bon réflexe, qu'il nous reste juste à chercher comment on pilote la **PB7** en explorant la documentation de la memory map : autrement dit "à quelle adresse taper pour l'allumer". Et c'est une très bonne idée. Mais avant cela, nous devons prendre un peu plus d'informations sur l'architecture du STM32. Ce qui nous intéresse précisément, là, maintenant, c'est de savoir **comment le CPU du STM32 est connecté à PB7** : cela va nous renseigner sur ce que nous devons faire pour configurer et activer/désactiver la pin.

Alors allons-y. D'abord : aussi surprenant que cela paraisse, la documentation de presque 150 pages de l'annexe B est ... un résumé ! La documentation complète est un document appelé "document de référence" qui fait lui plus de 900 pages. Et les informations utiles pour aller plus loin avec l'aspect informatique sont dans ce document. Des extraits utiles pour nous ont été mis en annexe C. Une description simplifiée du fonctionnement interne du STM32 est décrite page 249 et adaptée ci-dessous. On appelle cela "l'architecture" du microcontrôleur.

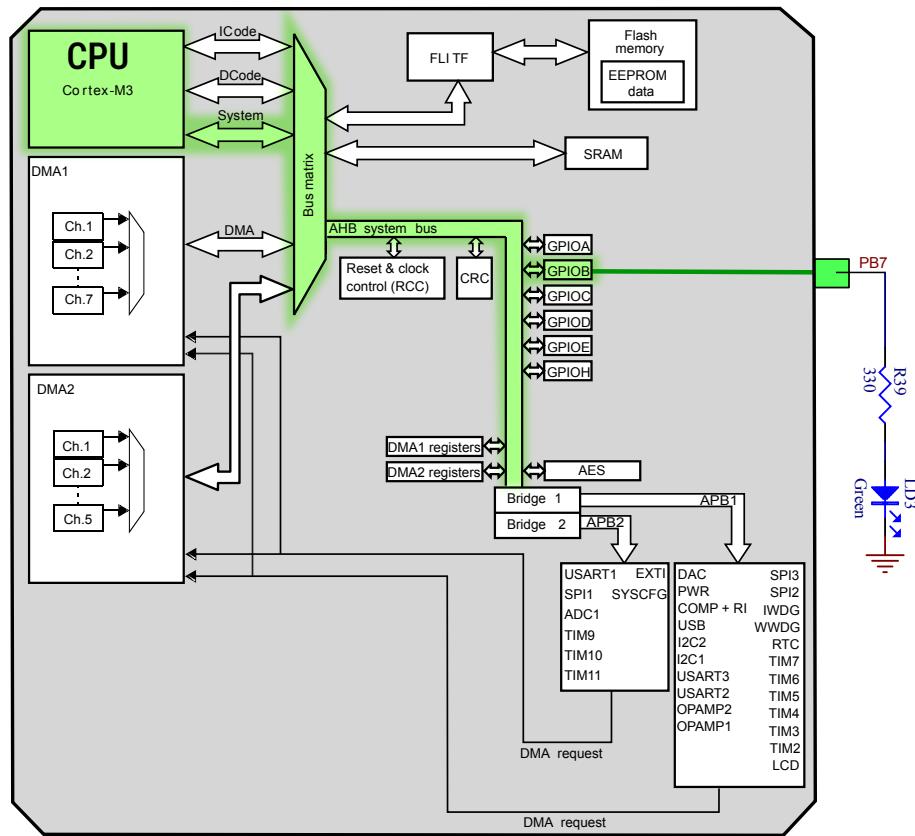
**STM32L152xx  
Reference Manual**  
extraits annexe C

**RM0038**  
**Reference manual**  
STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx  
advanced Arm®-based 32-bit MCUs

**Introduction**  
This reference manual targets application developers. It provides complete information on how to use the STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx microcontroller memory and peripherals. The STM32L151xx, STM32L152xx and STM32L162xx are referred to as STM32L1xx throughout the document, unless otherwise specified.  
The STM32L1xx is a family of microcontrollers with different memory sizes, packages and peripherals.  
For memory, package and peripheral characteristics and electrical device characteristics, refer to the corresponding datasheet.  
For information on programming, erased and protection of the internal non volatile memory, refer to the STM32L1xx Flash memory programming manual.  
For information on the Arm® Cortex®-M3 core, refer to the Cortex®-M3 Technical Reference Manual.

### Related documents

- [CubeMX User Reference Manual](#), available from <http://infocenter.arm.com>
- Available from [www.st.com](http://www.st.com)
- STM32L151xx and STM32L152xx datasheets
- STM32L151xx and STM32L152xx reference manual
- STM32L151xx and STM32L152xx datasheet
- STM32L1xx Flash memory programming manual
- Migrating from STM32L151xx to STM32L152xx and vice versa STM32L150xx to STM32L151xx
- Migrating from STM32L150xx-A to STM32L151xx-A and from STM32L150xx-B to STM32L151xx-B
- Migrating from STM32L150xx-C to STM32L151xx-C
- Migrating from STM32L150xx-D to STM32L151xx-D (Tm17)
- Migrating from STM32L150xx-E to STM32L151xx-E (Tm20)



On voit plusieurs informations sur ce graphique. D'abord : on voit que ce n'est pas une architecture "von Neumann", parce qu'il y a plusieurs bus. On est ici en présence d'une architecture "**architecture Harvard**" (ou même ici "architecture Harvard modifiée"), qui est caractérisée par une séparation entre les bus pour le programme et les bus pour les données<sup>2</sup>. Notez que pour nous ce n'est pas très important, ça ne remet pas en cause tout ce qui a été dit jusqu'ici et n'a pas tellement de conséquences sur la façon de programmer ensuite. Alors passons sur cet aspect.

Plus utile pour nous : on voit qu'en partant du CPU, c'est à dire le lieu où le programme est exécuté, il faut "traverser" un ensemble d'éléments pour arriver au **GPIO-B**, que l'on appelle aussi **port B**. Ce chemin a été surligné en vert sur le dessin. Or ces éléments ne sont pas actifs par défaut et on va donc devoir les activer. C'est en effet ce qui est indiqué à la page 46 du document de référence, copié page 252 de ce document :

*“After each device reset, all peripheral clocks are disabled (except for the SRAM and Flash interface). Before using a peripheral, its clock should be enabled in the RCC\_AHBENR, RCC\_APB1ENR or RCC\_APB2ENR register.”*

Comprendons ce qui est indiqué ici :

*“après chaque ‘reset’ du microcontrôleur (note : et donc au moins à chaque mise sous tension), les “horloges” des bus périphériques sont désactivées, sauf SRAM et Flash. Avant d’utiliser un périphérique, son horloge doit être activée avec les registres RCC\_AHBENR, RCC\_APB1ENR ou RCC\_APB2ENR.*

On peut déjà essayer d'avoir une compréhension un peu superficielle de ce qui est dit : par défaut, aucun bus n'est actif, sauf ceux qui relient le CPU à la SRAM (autrement dit à la RAM) et à la Flash (autrement dit la mémoire qui contient le programme). Déjà, il est bien normal que le bus Flash et SRAM soient actifs : Si ces bus ne sont pas actifs, le processeur n'a pas accès au programme, ni à la RAM, et donc impossible de travailler. Les autres concernent, comme on le voit sur le schéma, la relation aux périphériques (GPIO, USART, ADC, USB, I2C, ...) et à des fonctionnalités assez avancées que nous ne décrirons pas ici (par exemple DMA).

2. ce n'est pas "le bus de données" au sens de von Neumann

Toujours dans le texte, on parle de : **RCC\_AHBENR**, **RCC\_APB1ENR** et **RCC\_APB2ENR**. On comprend assez bien que le premier parle du bus **AHB** que nous avons identifié, et les deux autres des deux bus **APB**. Et donc pour continuer avec notre LED, nous devons comprendre comment fonctionne le registre **RCC\_APB1ENR** et donc regarder sa documentation, pour l'activer.

Maintenant qu'on en sait un peu plus, allons voir la memory map (page 253 de ce document). On lit deux informations utiles pour nous :

Catégorie de Registres	Plage d'adresses	Description
RCC	0x4002 3800 - 0x4002 3BFF	page 168 (page 257 ici)
GPIOB ( <i>port B</i> )	0x4002 0400 - 0x4002 07FF	page 189 (page 278 ici)

## Remarque

Les adresses sont notées, comme c'est presque toujours le cas, en hexadécimal. Le fait que c'est de l'hexadécimal est implicitement représenté par le **0x** qui précède les valeurs. C'est la même convention qui est utilisée en langage C et que nous utiliserons donc dans les codes que nous écrirons.

Maintenant, on voit sur la memory map du registre RCC (voir tableau ci-dessus), que le registre **RCC\_AHBENR** est à "l'offset" (= à la "distance") **0x1C**, et on voit qu'il fait 32 bits, autrement dit la taille d'un **long** en langage C. Ainsi, en langage C, on peut utiliser le code ci-dessous pour pointer sur ce registre. On va donc créer un pointeur qui porte le nom du registre (on pourrait mettre n'importe quoi mais le faire est juste plus simple pour s'y retrouver), ce pointeur regardera la mémoire comme des **long**, et pointera à l'adresse **0x4002 3800 + 0x1C** :

```
long* RCC_AHBENR = (long*) (0x4002 3800 + 0x1C) ;
```

Accéder à cette variable nous donne accès à la totalité des 32 bits du registre. Mais maintenant il faut se renseigner sur comment on doit indiquer à ce registre que l'on veut activer le bus **AHB**. Et pour cela il faut ... encore se documenter ! La documentation utile est dans la section 6 dédiée au "Reset and clock control (RCC)", dans le document de référence. La partie qui décrit le registre qui nous intéresse est disponible page 255 de ce document. On y lit que c'est le bit numéro 1 qui permet d'activer le bus AHB pour le port B, en le plaçant à 1.

En binaire, le bit 1 vaut  $2^1 = 2$ . Alors on pourrait penser qu'il suffit de mettre la valeur 2 dans le registre, et donc écrire :

```
long* RCC_AHBENR = (long*) (0x4002 3800 + 0x1C) ;
(*RCC_AHBENR) = 2; // mauvais code
```

Mais cette façon de faire est mauvaise parce qu'elle impose des 0 partout ailleurs dans le registre, or ce n'est pas ce que l'on veut : on va devoir faire en sorte de ne modifier que le bit 1.

On va utiliser les opérations binaires du C, résumées ci-dessous :

Opération binaire	Symbol en C	exemple en C	résultat de l'exemple
et bit à bit	&	0xE1 & 0x30	0x20
ou bit à bit		0xE1   0x12	0xF3
non bit à bit	~	~0xF2	0x0D
décalage de bit à gauche	<<	0x01 << 3	0x08
décalage de bit à droite	>>	0x20 >> 4	0x02

Et donc le bon code pour seulement activer le bit numero 1 est :

```
long* RCC_AHBENR = (long*) (0x40023800 + 0x1C) ;
(*RCC_AHBENR) = (*RCC_AHBENR) | 2; // bon code
```

## Remarque

En effet, on peut se représenter les choses comme suit, si on les représente en binaire sur un octet :

$$\begin{array}{r}
 \text{xxxx xxxx} \\
 \text{ou } 0001\ 0000 \\
 \hline
 \text{xxx1 xxxx}
 \end{array}$$

Mais on peut faire encore un peu mieux : on a eu à "calculer" que le bit d'intérêt soit le bit numéro 1, fait qu'il faut  $2^1$ , soit 2, et utiliser la valeur 2 dans le code. En utilisant un décalage de bit, on peut faire mieux :

```
long* RCC_AHBENR = (long*) (0x40023800 + 0x1C) ;
(*RCC_AHBENR) = (*RCC_AHBENR) | (1<<1); // code definitif
```

## Important

On vient de voir une **technique classique** en programmation microcontrôleurs, qui consiste à activer un bit dans un registre. Pour cela, on utilise les opérateurs binaires du C :

Opération binaire	Symbol en C	exemple en C	résultat de l'exemple
et bit à bit	&	0xE1 & 0x30	0x20
ou bit à bit		0xE1   0x12	0xF3
non bit à bit	~	~0xF2	0x0D
décalage de bit à gauche	<<	0x01 << 3	0x08
décalage de bit à droite	>>	0x20 >> 4	0x02

et on va les utiliser comme suit, sur un exemple, où on veut activer le bit numéro 5 d'un registre :

```
long* registre = (long*) (0x??????) ;
(*registre) = (*registre) | (1<<5);
```

## II.6 Configurer une pin pour en faire une sortie tout-ou-rien

On a donc activé le bus qui permet au GPIO "port B" de fonctionner. La démarche que nous venons d'avoir est assez classique : se documenter (beaucoup) sur le processeur, puis écrire les quelques lignes de C qui permettent d'arriver à nos fins.

Maintenant qu'on a enfin "le droit" d'utiliser le port B, il faut le configurer, puis l'utiliser pour allumer la LED. En effet, comme on l'a vu, les différentes pattes peuvent avoir des fonctionnalités variées (entrée ou sortie tout-ou-rien, ADC, USB, etc) et il faut bien paramétriser la fameuse patte que l'on va utiliser. On a déjà vu que la plage d'adresses des registres associés au port B est **0x4002\_0400 - 0x4002\_07FF**, et ce que l'on veut c'est placer la pin **PB7** (led verte) en tant que sortie tout-ou-rien. Alors repartons dans la documentation. Les GPIO sont documentés page [272](#). On voit qu'il faut paramétriser plusieurs registres de 32 bit. Mais en réalité pour démarrer avec quelque chose de simple on peut n'en paramétriser qu'un seul :

registre	signification	offset	bits pour PB7	valeurs possibles
<b>GPIOB_MODER</b>	Mode de fonctionnement	<b>0x00</b>	14 et 15	<b>00</b> Entrée <b>01</b> Sortie <b>10</b> Fonction alternative <b>11</b> Mode analogique

Et donc en C, on doit placer le bit 14 à 1 et le bit 15 à 0. On peut le faire ainsi :

```
long* GPIOB_MODER = (long) (0x40020400 + 0x0);
(*GPIOB_MODER) = (*GPIOB_MODER) | (1<<14);
(*GPIOB_MODER) = (*GPIOB_MODER) & (~(1<<15));
```

## Remarque

En effet, on a déjà placé un bit à 1 avec une opération "ou", et on voit ici, sur la 3ième ligne, comment placer un bit à 0 avec un "et-non". On peut se représenter les choses comme suit, si on les représente en binaire sur un octet :

$$\begin{array}{r}
 a = 0001\ 0000 \\
 \text{non } a = 1110\ 1111 \\
 \\ 
 \begin{array}{r}
 \text{xxxx}\ \text{xxxx} \\
 \text{et}\ \ 1110\ 1111 \\
 \hline
 \text{xxx0}\ \text{xxxx}
 \end{array}
 \end{array}$$

Mais en général, si on doit placer une certaine séquence de 0 et de 1 dans un registre, on préfère utiliser une technique dite "de masquage" : on place simplement des 0 sur toute la zone qui nous intéresse, avant de placer la valeur que l'on veut avec un "ou". En C, cela donne :

```
long* GPIOB_MODER = (long) (0x40020400 + 0x0);
(*GPIOB_MODER) = (*GPIOB_MODER) & (~(0b11<<14));
(*GPIOB_MODER) = (*GPIOB_MODER) | (0b01<<14);
```

où **0b** est la syntaxe en C pour parler des nombres binaires (de façon analogue à **0x** pour les nombres hexadécimaux).

## Important

La technique définitive pour placer des valeurs dans un registre fonctionne par masquage de bits. On se donne "un masque" qui indique la zone où on veut écrire une séquence de bits, puis la valeur binaire à placer. Par exemple, si on veut placer la séquence **0b010** sur les bits 8 à 11 bit d'un registre, on va d'abord créer un masque **0b111**. Le masquage se fait alors ainsi :

```
long* registre = (long) (0x????????);
// 1. masquage
(*registre) = (*registre) & (~(0b111 << 8));
// 2. inscription de la valeur voulue
(*registre) = (*registre) | (0b010 << 8);
```

## II.7 Piloter une sortie tout-ou-rien

On a donc configuré le bus **AHB** et la pin **PB7** du port B comme sortie. Il nous reste à piloter l'état de la pin **PB7**. Comme c'est une sortie tout-ou-rien, il suffit de paramétrier un bit "quelque part". Il suffit de trouver lequel. Alors repartons (encore !) dans la documentation. Les GPIO sont documentés page [272](#). On voit qu'il faut modifier un registre 32 bit :

registre	signification	offset	bits pour PB7	valeurs possibles
<b>GPIOB_ODR</b>	Pilotage d'un port de sortie	<b>0x14</b>	7	<b>0</b> Etat bas <b>1</b> Etat haut

Et donc, en C, on va écrire, pour allumer la LED verte :

```
long* GPIOB_ODR = (long) (0x40020400 + 0x14);
(*GPIOB_ODR) = (*GPIOB_ODR) | (0b1<<7);
```

Et pour l'éteindre :

```
1 long* GPIOB_ODR = (long*) (0x40020400 + 0x14);
2 (*GPIOB_ODR) = (*GPIOB_ODR) & (~(0b1<<7));
```

## II.8 Le programme complet

On va faire clignoter la LED avec un programme très simple :

```
1 int main() {
2 // # A. Configuration
3 // ## 1. Activer le bus AHB pour le GPIOB (port B)
4 long* RCC_AHBENR = (long*) (0x40023800 + 0x1C) ;
5 (*RCC_AHBENR) = (*RCC_AHBENR) | (1<<1); // code definitif
6
7 // ## 2. Parametrer la pin PB7 en tant que sortie :
8 long* GPIOB_MODER = (long*) (0x40020400 + 0x0);
9 (*GPIOB_MODER) = (*GPIOB_MODER) & (~(0b11<<14));
10 (*GPIOB_MODER) = (*GPIOB_MODER) | (0b01<<14);
11
12 // # B. Boucle principale
13 long* GPIOB_ODR = (long*) (0x40020400 + 0x14);
14 while(1) {
15     // ## 1. Allumer la LED
16     (*GPIOB_ODR) = (*GPIOB_ODR) | (0b1<<7);
17     for(int k=0; k<100000; k++) { } // perdre du temps pour "attendre"
18     // ## 2. Eteindre la LED
19     (*GPIOB_ODR) = (*GPIOB_ODR) & (~(0b1<<7));
20     for(int k=0; k<100000; k++) { } // perdre du temps pour "attendre"
21 }
22 return 0;
23 }
```

 [code/code-II-8-1.c]

## II.9 Et maintenant?

On n'a pas encore atteint notre objectif de piloter 3 LED dont une à l'extérieur de la carte, mais on a déjà bien progressé dans la compréhension de ce qu'il faut faire. On va continuer dans le prochain chapitre.

### ❖ En résumé ...

Nous avons vu beaucoup de choses !

D'abord, piloter un périphérique du microcontrôleur se fait parce que l'on a connaissance à la fois :

- des **fonctionnalités de chaque pin**, et donc choisir la bonne pin pour y connecter le bon composant ou la bonne électronique de pilotage,
- de la **memory map** et une certaine vision de **l'architecture** du microcontrôleur pour déterminer quels **registres** nous devons modifier pour paramétriser et activer ce qui nous intéresse.

Tout cela passe par **étudier la documentation**. Il n'y a pas de "méthode" pour ça, seulement l'expérience.

Ensuite, nous avons appris des **techniques de manipulation des bits** dans les registres, notamment comment utiliser les opérateurs **et**, **ou**, **non** et de **décalage de bit**. Nous avons également appris la technique de **masquage de bits** qu'il faut absolument connaître : ces techniques sont extrêmement fréquentes lorsque l'on utilise des microcontrôleurs.

*Malgré tout, il faut comprendre que l'exploration que nous avons fait fait partie des choses difficiles lors de la prise en main d'un nouveau processeur ou d'une nouvelle carte.* ■

## Chapitre III – Sous-ensemble 1 : Entrées et Sorties "tout-ou-rien"

### ⌚ Motivations et objectifs

Dans le survol que nous avons fait dans la partie précédente, nous sommes partis de la carte électronique qui contient déjà des éléments câblés. Et donc, nous avons essentiellement étudié la documentation de la carte, et la documentation du microcontrôleur. Et donc notre résultat a été pour l'essentiel **un programme**. Mais utiliser un microcontrôleur n'est pas seulement "programmer", c'est aussi **mettre en oeuvre l'électronique** utile pour piloter ou interroger les composants avec lesquels il se interfacer. Nous allons, ici, mettre tout ça ensemble, sur des exemples simples mais généraux, basés sur des entrées et sorties "tout-ou-rien".

### III.1 Implantation des LEDs : aspect électronique

La carte contient une LED bleue et une LED verte, et on veut en ajouter une rouge. On va commencer par regarder comment la LED "verte" a été câblée par exemple, et confronter cela à la documentation du microcontrôleur. A ce stade, ce qui nous intéresse c'est :

- Quelle pin a été utilisée et pourquoi ?
- Pourquoi le cablage qui a été réalisé fonctionne bien compte tenu des caractéristiques électriques de cette pin ?

Répondons donc à ces deux questions.

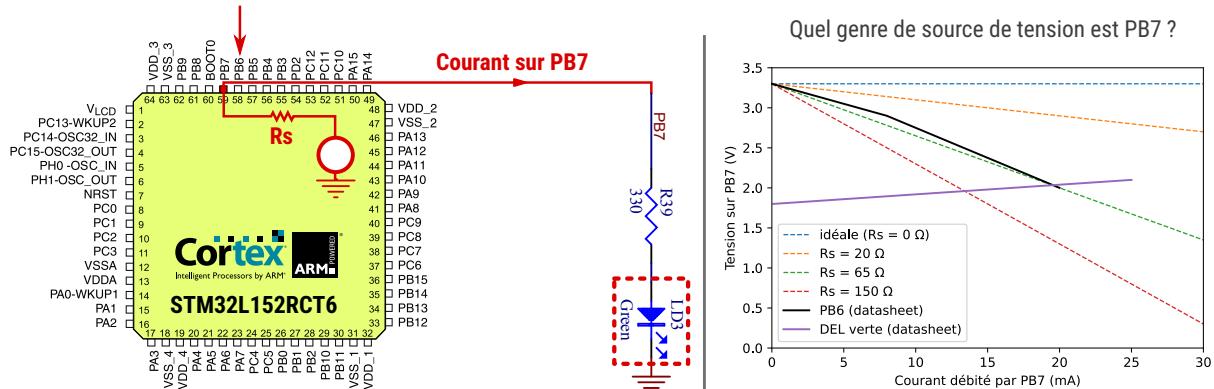
#### Quelle pin ?

Comme on peut le lire **dans la documentation de la carte** sur le **schema de la carte page 104** (coordonnée B-4), et comme cela est aussi **confirmé par le tableau 8 page 98**, c'est la pin **PB7** qui est utilisée.

Regardons maintenant **dans la documentation du microcontrôleur**, dans la partie "**Pins description**", **page 155** (en haut). On voit que la PIN **PB7** est bien une "Entrée/Sortie" (notée **I/O**), et donc elle est utilisable comme entrée ou sortie tout-ou-rien (et donc pour nous ce sera une entrée, mais c'est par le code que nous le paramétrons).

#### Pourquoi c'est câblé comme ça ?

D'abord : il faut comprendre que ce câblage est ce qu'il est parce que les concepteurs de la carte ont pris en compte le fait que la pin **PB7** sera configurée, par programmation, pour être une sortie. Dire que c'est "une sortie" signifie que, au niveau de **PB7**, le microcontrôleur va se comporter **comme une source de tension**, capable de générer uniquement 2 niveaux ou états : "tout" (3V) ou "rien" (0V). Cette tension va être imposé par l'état d'une variable (un registre) situé dans une case mémoire quelque part dans la memory map et on pourra modifier par programmation. Et donc pour allumer la LED bleue lorsque la pin envoie 3V, on a besoin de transformer cette tension en un signal électrique utilisable par la LED, assez fort pour l'allumer mais pas trop fort pour ne pas la détruire.



Nous devons donc regarder si la source de tension de **PB7** est suffisamment puissante pour piloter la LED.

Retournons dans la section "Electrical Characteristics" de la documentation du microcontrôleur, page 196, on a des informations sur les entrées/sorties. Ce qui nous intéresse est le tableau 44, qui donne, sur quelques points, la valeur de la tension livrée par **PB7** en fonction du courant qui est demandé. C'est ce qui a servi de base pour tracer les courbes de la figure ci-dessus. Et on se rend compte que le comportement de cette source peut se modéliser en utilisant  $R_s = 65 \Omega$ , ce qui dit qu'il ne faut pas s'attendre à délivrer des courants trop forts. Mais maintenant comparons avec ce que demande la DEL, dont la documentation est donnée Annexe D, page 281. La courbe "Forward Current vs Forward Voltage" (soit la courbe  $I(V)$  de cette diode) a été reportée sur le graphe ci-dessus et on voit bien que le **PB7** ne pourra pas piloter la LED au courant maximum. Alors regardons quel choix a été fait par les concepteurs de la carte : ils ont utilisé une résistance  $R_{39} = 330 \Omega$  pour faire chuter la tension. Faisons le calcul en écrivant une loi des mailles :

$$V_{DD} - R_s \times I - R_{39} \times I = V_{diode}$$

La tension de diode est environ 1,8 V à 2 V. Prenons 1,9 V, ca fait une erreur faible (environ 5%).  $R_s = 65 \Omega$  environ, et  $V_{DD} = 3 V$ . On obtient environ  $I = 2,8 mA$ . Donc la LED est loin d'éclairer à son maximum dans ce montage (et ce serait sûrement inutile).

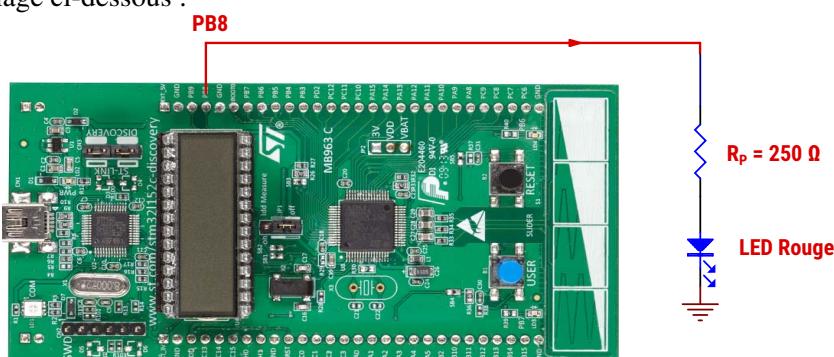
Si on calcule pour avoir le même courant avec la diode rouge, il faut prendre en compte la tension de la diode. Elle est entre 1,9 V et 2,4 V. Prenons quelque part au milieu, 2,1 V, et refaisons le calcul, en prenant un courant similaire au courant précédent, soit  $I = 2,8 mA$ , mais cette fois on cherche la valeur de la résistance de protection  $R_P$  :

$$V_{DD} - R_s \times I - R_P \times I = V_{diode}$$

soit :

$$R_P = \frac{V_{DD} - V_{diode}}{I} - R_s \approx \frac{3 - 2,1}{2,8 \times 10^{-3}} - 65 \approx 250 \Omega$$

Comme on va le justifier ci-dessous, on va utiliser la pin **PB8** pour cette LED, et cela va nous conduire à réaliser le cablage ci-dessous :



## III.2 Implantation des LEDs : aspect informatique

Nous avons déjà vu l'aspect informatique mais résumons ici ce que nous avons fait.

Les pins du microcontrôleur ont pour la plupart comme nom : **PAxx**, **PBxx**, **PCxx**, etc. jusqu'à **PHxx**. Dans cette écriture, **P** signifie "port".

Un port est un ensemble de pins, au maximum 16. Et des ports il y en a plusieurs, nommés **A**, **B**, **C**, ... jusqu'à **H**. Et pour chaque port, la pin est identifiée par un numéro entre 0 et 15. Par exemple, **PB6** est la 6ième patte du port B. Ce regroupement est utile pour nous parce que les ports sont référencés dans le mapping, comme on peut le voir page 161. Par exemple, nous qui travaillons avec les pins **PB6**, **PB7** et **PB8**, on s'intéresse au port B, qui dans le mapping est référencé à l'adresse **0x40020400**, comme indiqué page 161.

Pour aller plus loin, nous avons étudié l'architecture du microcontrôleur et avons vus qu'il fallait activer le bus port B à travers le bus **AHB**, ce que nous avons faits. On a vu que cela se faisait dans les registres du bus AHB, à l'adresse **0x4002381C**, en activant un unique bit.

Enfin, nous avons paramétré le GPIO correspondant à la pin **PB7** pour la configurer en tant que sortie tout-ou-rien. Et pour finir, nous avons pu activer le bit qui permet de placer un 1 sur **PB7**, ce qui revient à lui demander de forcer une tension de **3V**.

On peut donc adopter exactement la même approche pour activer les pins **PB6**, **PB7** et **PB8**, en modifiant légèrement le code de la page précédente, pour prendre en compte **PB6** et **PB8** :

```

1 int main() {
2 // # A. Configuration
3 // ## 1. Activer le bus AHB pour le GPIOB (port B)
4 long* RCC_AHBENR = (long*) (0x40023800 + 0x1C) ;
5 (*RCC_AHBENR) = (*RCC_AHBENR) | (1<<1); // code definitif
6
7 // ## 2. Parametrer la pin PB6 PB7 PB8 en tant que sortie :
8 long* GPIOB_MODER = (long*) (0x40020400 + 0x0);
9 (*GPIOB_MODER) = (*GPIOB_MODER) & (~(0b111111<<12));
10 (*GPIOB_MODER) = (*GPIOB_MODER) | (0b010101<<14);
11
12 // # B. Boucle principale
13 long* GPIOB_ODR = (long*) (0x40020400 + 0x14);
14 while(1) {
15     // ## 1. Allumer les 3 LEDs
16     (*GPIOB_ODR) = (*GPIOB_ODR) | (0b111<<6);
17     for(int k=0; k<100000;k++) { } // perdre du temps pour "attendre"
18     // ## 2. Eteindre les 3 LEDs
19     (*GPIOB_ODR) = (*GPIOB_ODR) & (~(0b111<<6));
20     for(int k=0; k<100000;k++) { } // perdre du temps pour "attendre"
21 }
22 return 0;
23 }
```

 [code/code-III-2-2.c]

## III.3 Implantation de l'interrupteur : aspect électronique

On va continuer en regardant comment l'interrupteur a été cablé, et confronter cela à la documentation du microcontrôleur, en répondant aux mêmes questions que précédemment.

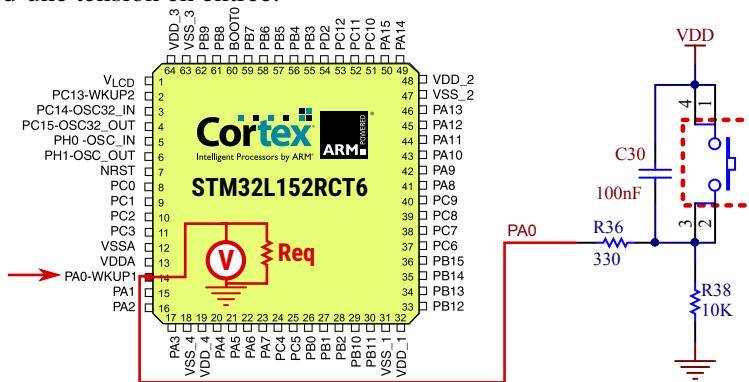
### Quelle pin ?

Comme on peut le lire [dans la documentation de la carte](#) sur le [schema de la carte](#) page 107, et comme cela est aussi [confirmé par le tableau 8](#) page 96, c'est la pin **PA0** qui est utilisée.

Regardons maintenant [dans la documentation du microcontrôleur](#), dans la partie "**Pins description**", [page 149](#) (en bas). On voit que la PIN **PA0** est bien une "Entrée/Sortie" (notée **I/O**), et donc elle est utilisable comme entrée ou sortie tout-ou-rien (et donc pour nous ce sera une entrée, mais c'est le code qui le paramètrera).

### Pourquoi c'est câblé comme ça ?

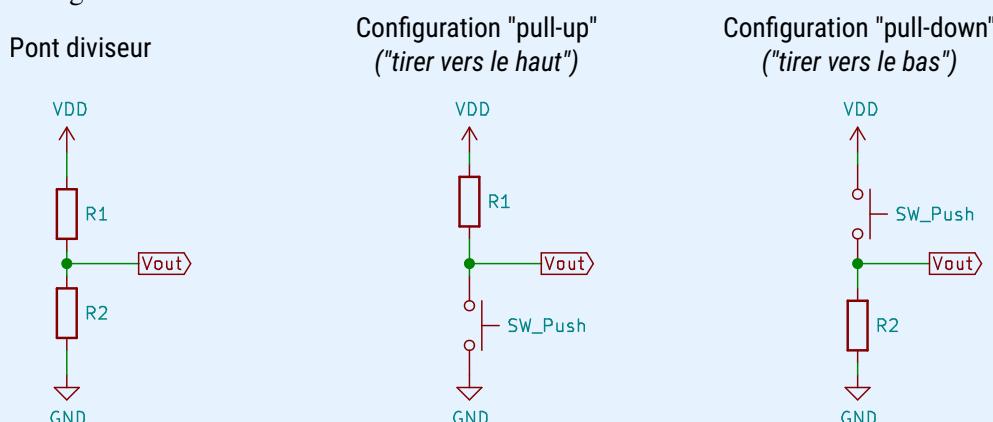
D'abord : il faut comprendre que ce câblage est ce qu'il est parce que les concepteurs de la carte ont pris en compte que la pin **PA0** sera configurée, par programmation, pour être une entrée. Dire que c'est "une entrée" signifie que, au niveau de **PA0**, le microcontrôleur va se comporter [comme un voltmètre](#), capable de mesurer 2 états : "tout" (3V) ou "rien" (0V). Cet état va être transféré dans une case mémoire quelque part dans la memory map et on pourra la récupérer par programmation. Et donc pour réaliser sa mesure, le voltmètre a besoin d'une tension en entrée.



### Important

Mais un interrupteur ne fournit pas "une tension" : c'est une impédance qui vaut (en exagérant)  $0\Omega$  ou  $+\infty\Omega$  selon que le bouton est appuyé ou pas. Il faut donc le conditionner pour ça. Et donc la question est : **comment je peux transformer une variation d'impédance en variation de tension ?**

En électronique ce genre de problème se résout presque toujours de la même façon : **par un pont diviseur !** Regardons cela sur le schéma ci-dessous.



Pour rappel : dans les conditions du pont diviseur (aucun courant consommé sur la branche  $V_{out}$ ), la tension  $V_{out}$  s'écrit :

$$V_{out} = \frac{R_2}{R_1 + R_2} \times V_{DD}$$

et donc on a deux configurations de pont diviseur possibles pour interroger l'état de l'interrupteur :

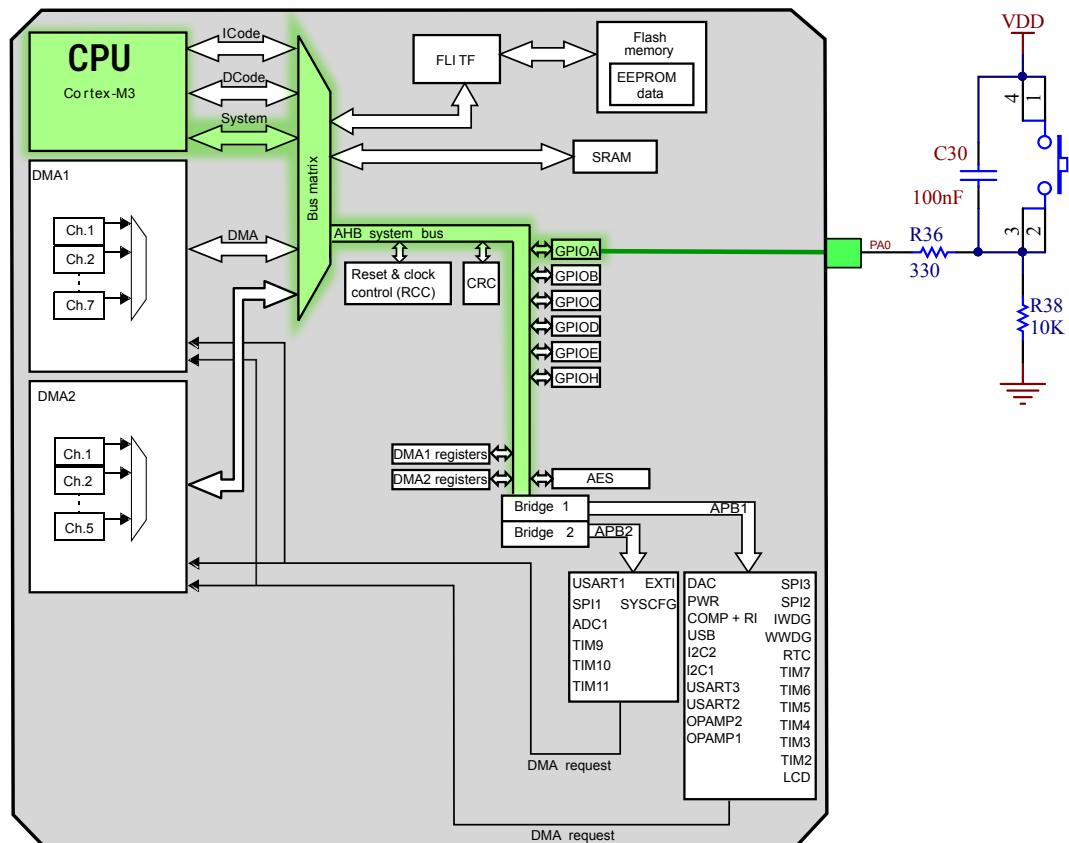
- en le plaçant en bas, configuration que l'on appelle "**pull-up**". Dans ce cas, si l'interrupteur est fermé, alors  $R_2 = 0$  et l'expression ci-dessus donne  $V_{out} = 0$ , et si l'interrupteur est ouvert,  $R_2 = +\infty$  et l'expression ci-dessus tend vers  $V_{out} = V_{DD}$ . C'est donc une configuration "inverseuse".

- en le plaçant en haut, configuration que l'on appelle "**pull-down**". Dans ce cas, si l'interrupteur est fermé, alors  $R_1 = 0V$  et l'expression ci-dessus donne  $V_{out} = V_{DD}$ , et si l'interrupteur est ouvert,  $R_1 = +\infty$  et l'expression ci-dessus tend vers  $V_{out} = 0V$ . C'est donc une configuration "non-inverseuse".

On a donc bien transformé l'état de l'interrupteur en une tension tout-ou-rien, lisible par l'entrée **PA0** du microcontrôleur ! Le schéma au schéma auquel nous avons abouti est similaire au schéma proposé par le fabricant de la carte, comme les illustrations précédentes l'ont montré. Sur le schéma que nous avons établi, il manque une capacité et une résistance par rapport au schéma proposé par le fabricant. Ces composants servent à gérer le problème des "rebonds", c'est à dire que lorsque l'on ferme ou ouvre un interrupteur, entre les deux états de repos il existe un état intermédiaire où le contact est intermittent (ouvert/fermé/ouvert/fermé rapidement et aléatoirement). Ces composants servent au filtrage de ces oscillations. *En effet : le filtre pass-bas dont on parle est composé par R36 et C30 : comme les signaux à filtrer sont des signaux alternatifs (le "rebond" est une sorte de créneau irrégulier), la tension continue VDD joue tout aussi bien le rôle d'une masse.*

### III.4 Implantation de l'interrupteur : aspect informatique

La démarche va être très similaire à la démarche que nous avons eu avec les LED. Repartons de l'architecture :



On voit que là aussi, le bus AHB doit être activé, mais pour le bus GPIO-A. On retourne donc à la page 255 qui décrit le registre associé à la configuration du bus AHB, et activons le GPIO-A, ce qui revient à mettre à 1 le bit 0 de **RCC\_AHBENR** :

```
long* RCC_AHBENR = (long) (0x40023800 + 0x1C) ;
(*RCC_AHBENR) = (*RCC_AHBENR) | 0b1;
```

Maintenant, nous devons paramétrer la pin PA0 pour qu'elle se comporte comme une entrée. On a besoin de connaitre la position du port "A" dans la memory map, alors repartons dans la documentation page 253. On voir les informations sur le port A :

Catégorie de Registres	Plage d'adresses	Description
GPIOA (port A)	0x4002 0000 - 0x4002 03FF	page 189 (page 278 ici)

Les GPIO sont documentés page 272. De façon similaire à ce que l'on a fait pour les LED, on va configurer le registre GPIO

registre	signification	offset	bits pour PA0	valeurs possibles
GPIOA_MODER	Mode de fonctionnement	0x00	00 et 01	00 Entrée 01 Sortie 10 Fonction alternative 11 Mode analogique

Ce qui va donner en C, en utilisant un masquage de bits :

```
long* GPIOA_MODER= (long*) (0x40020000) ;
(*GPIOA_MODER) = (*GPIOA_MODER) & (~ 0b11);
```

Et maintenant il reste à écouter l'état de l'interrupteur. On repart donc page 272, et un peu plus loin on voit le registre GPIOA\_IDR nommé "GPIO port input data register". On voit que la pin PA0 est associée au bit 0 du registre. Pour lire le bit qui nous intéresse et seulement lui, on va devoir simplement le masquer avec un et logique :

```
long* GPIOA_IDR= (long*) (0x40020000 + 0x10) ;
int switch_on = ((*GPIOA_MODER) & 0b1) != 0;
```

Le !=0 à la fin de l'expression sert à avoir simplement la variable `switch_on` à "false" (0) si l'interrupteur est ouvert, et à "true" (1) si l'interrupteur est fermé, sans dépendre du numéro du bit<sup>1</sup>.

## III.5 Le programme complet

Ecrivons maintenant enfin le programme complet, où chaque appui sur l'interrupteur provoque l'allumage de la LED suivante. On va devoir rassembler les initialisations des ports ensemble notamment, puis gérer le petit algorithme. Voici :

```
1 int main() {
2 // # A. Configuration
3 // ## 1. Activer le bus AHB pour le GPIOA et GPIOB
4 long* RCC_AHBENR = (long*) (0x40023800 + 0x1C) ;
5 (*RCC_AHBENR) = (*RCC_AHBENR) | (0b11);
6
7 // ## 2. Parametrer les pin PB6 PB7 PB8 en tant que sorties :
8 long* GPIOB_MODER = (long*) (0x40020400 + 0x0);
9 (*GPIOB_MODER) = (*GPIOB_MODER) & (~(0b111111<<12));
10 (*GPIOB_MODER) = (*GPIOB_MODER) | (0b010101<<12);
11
12 // ## 3. Parametrer la pin PA0 en tant qu'entree :
13 long* GPIOA_MODER = (long*) (0x40020000 + 0x0);
14 (*GPIOA_MODER) = (*GPIOA_MODER) & (~ 0b11);
15
16 // # B. Boucle principale
17 long* GPIOB_ODR = (long*) (0x40020400 + 0x14);
18 long* GPIOA_IDR = (long*) (0x40020000 + 0x10) ;
19 int activeLed = 0; // the green one
```

1. sur cet exemple ca n'est pas très convaincant parce que la carte utilise le bit 0, mais avec n'importe quel autre bit ca aurait du sens.

```

20 int prev_switch_status = 0;
21 while(1) {
22     int switch_status = ((*GPIOA_IDR) & 0b1) != 0;
23     if(switch_status == 1 && prev_switch_status == 0) {
24         activeLed = activeLed +1;
25         activeLed = activeLed%3;
26     }
27     prev_switch_status = switch_status;
28
29     if(activeLed == 0) {
30         (*GPIOB_ODR) = (*GPIOB_ODR) | (0b1<<6);
31         (*GPIOB_ODR) = (*GPIOB_ODR) & ~((0b110<<6));
32     }
33     if(activeLed == 1) {
34         (*GPIOB_ODR) = (*GPIOB_ODR) | (0b1<<7);
35         (*GPIOB_ODR) = (*GPIOB_ODR) & ~((0b101<<6));
36     }
37     if(activeLed == 2) {
38         (*GPIOB_ODR) = (*GPIOB_ODR) | (0b1<<8);
39         (*GPIOB_ODR) = (*GPIOB_ODR) & ~((0b011<<6));
40     }
41 }
42 return 0;
43 }
```

 [code/code-III-5-3.c]

## ❖ En résumé ...

Les GPIO en entrée et en sortie nous ont amené à regarder un ensemble de choses.

**Sur un plan électronique :** Nous avons vu que les GPIO peuvent se comporter comme des entrées ou sorties numériques.

- En tant qu'entrées, elles sont comparables à un voltmètre (impédance immense) et il est peu probable que l'impédance d'entrée représente un jour un problème.
- En tant que sorties, la **source de tension peut être de qualité insuffisante** pour piloter n'importe quel composant électronique, ce qui revient à dire qu'elle ne permet pas de générer des puissances électriques infinies (quoi de plus normal !), ce que l'on modélise par une résistance en série avec le générateur.

Nous avons vu comment le Datasheet pouvait rendre compte de la qualité des entrées/sorties sur un plan électronique.

Nous avons également vu les montages de base pour piloter une LED ou un interrupteur, et notamment le principe de "résistance de pull-up".

## Chapitre IV – Programmer par bibliothèques

### ⌚ Motivations et objectifs

Jusqu'ici nous avons utilisé la programmation que l'on nomme "bare-métal" ("métal à nu") pour explorer les bases de ce qu'est un microcontrôleur. Cette approche est bien entendu à avoir en tête, parce qu'elle est l'occasion de se faire une idée très concrète du fonctionnement profond d'un microcontrôleur, à travers l'exploration de la memory map et de l'architecture. Malgré tout, elle peut vite devenir complexe à mettre en oeuvre : pour des choses plus compliquées que juste des GPIO en "tout-ou-rien", l'écriture des programmes se complexifie assez naturellement. Pour éviter cette complexification, on peut recourir à des bibliothèques, et c'est la façon "moderne" de programmer sur microcontrôleur. C'est ce que l'on va regarder maintenant.

### IV.1 Programmation type "bare-metal"

Faisons un parallèle avec la programmation sur PC : celle-ci se fait toujours en utilisant des bibliothèques, parce que tous les programmes sont contraints d'utiliser le système d'exploitation, que ce soit pour afficher, gérer la mémoire, etc : l'accès direct aux ressource de l'ordinateur n'est permise qu'au système d'exploitation.

Avec les microcontrôleurs, jusqu'ici, on n'avait aucun `#include` dans nos codes : le signe que nous n'avons utilisé aucune bibliothèque, parce qu'il n'y avait jusqu'ici pas de nécessité. Mais en conséquence on a fait beaucoup d'allers-retours dans la documentation pour pouvoir programmer avec le microcontrôleur. C'est assez lié à une raison simple : nous avons programmé "from scratch", c'est à dire avec pour seule connaissance la documentation technique à notre disposition, pour connaître le mapping, les registres, etc. Cette approche que nous avons eu est appelée approche "bare-metal" pour ("metal à nu"), parce que c'est exactement comme si le "morceau de métal" qu'est la puce à semiconducteur était utilisée directement, sans code intermédiaire. Mais c'est souvent pas pratique, même si parfois on n'a pas le choix. Et dans la plupart des situations, on peut alléger tout ça par l'utilisation de bibliothèques.

### IV.2 Bibliothèques pour microcontrôleurs

Cette façon de programmer, type "bare-metal", a été la seule disponible pendant une longtemps dans l'histoire de l'utilisation des microcontrôleurs, et même à l'origine en programmation sur ordinateur. Mais depuis quelques dizaines d'années, on voit de plus en plus apparaître des bibliothèques pour microcontrôleur. Ici il faut être clair : il y aura difficilement des bibliothèques capables de vous permettre d'écrire un programme qui fonctionnera universellement sur tous les microcontrôleurs (ne serait-ce que parce que chacun a des capacités spécifiques), contrairement à ce que l'on peut faire en C et bien d'autres langages sur PC. Mais malgré tout il existe des choses qui vont beaucoup nous aider.

Par exemple :

- On trouve souvent **comme sur PC** des versions un peu incomplètes mais fonctionnelles de `stdio.h`, `string.h` ou `stdlib.h` sur un peu tous les microcontrôleurs. Cela aide déjà beaucoup pour gérer les chaînes de caractères ou la mémoire.

- Il existe aussi des choses plus spécifiques. Par exemple **de quoi gérer "du multitâche"** (c'est à dire pouvoir exécuter plusieurs tâches en parallèle), c'est très répandu (la plus connue est **freeRTOS**).
- Mais beaucoup plus intéressant pour nous à ce stade : **de quoi gérer les interfaces et fonctionnalités du microcontrôleur**. Ces bibliothèques sont souvent spécifiques à un microcontrôleur ou à une famille de microcontrôleur. Elles sont souvent assimilées à des "driver", avec un sens assez similaire des drivers sur PC. C'est à ce dernier aspect que l'on va s'intéresser ici.

### IV.3 Un premier pas : "S'abstraire" de la Memory Map

Une bibliothèque qui permettrait de "S'abstraire" de la Memory Map signifie que l'on peut travailler avec une bibliothèque qui nous permet de ne plus avoir à connaître les adresses de la memory map : elle les connaît à notre place, elle connaît l'organisation des registres dans la mémoire, et cela simplifie donc notre travail. C'est déjà très intéressant et on va commencer avec ça.

Pour les STM32, la bibliothèque porte le nom de la famille de processeurs sur laquelle on programme, et donc pour nous ce sera :

```
#include "stm32l1xx.h"
```

qui est la bibliothèque générique pour un ensemble de processeurs stm32, et qui convient à celui que l'on utilise : le STM32L152RT6. A titre d'exemple, on va comparer le code d'un programme qui fait clignoter une LED tel que nous l'avons déjà écrit, et la même chose mais en utilisant une abstraction de la memory map par la bibliothèque :

Code "baremetal"	Code "abstraction de la Memory Map"
<pre> 1 // aucune inclusion 2 3 4 int main() { 5 // # A. Configuration 6 // ## 1. Activer GPIOB (port B) 7 long* RCC_AHBENR=(long*)(0 8     x40023800 + 0x1C); 9 (*RCC_AHBENR)  = (1&lt;&lt;1); 10 11 // ## 2. PB7 en tant que sortie 12 long* GPIOB_MODER=(long*)(0 13     x40020400 + 0x0); 14 (*GPIOB_MODER) &amp;= ~(0b11&lt;&lt;14); 15 (*GPIOB_MODER)  = 0b01&lt;&lt;14; 16 17 // # B. Boucle principale 18 long* GPIOB_ODR=(long*)(0 19     x40020400 + 0x14); 20 while(1) { 21     // ## 1. Allumer la LED 22     (*GPIOB_ODR)  = 0b1&lt;&lt;7; 23     for(int k=0; k&lt;100000; k++){} 24     // ## 2. Eteindre la LED 25     (*GPIOB_ODR) &amp;= ~(0b1&lt;&lt;7); 26     for(int k=0; k&lt;100000; k++){} 27 } 28 return 0; 29 }</pre>	<pre> 1 // inclusion de la librairie 2 #include "stm32l1xx.h" 3 4 int main() { 5 // # A. Configuration 6 // ## 1. Activer GPIOB (port B) 7 8 (*RCC).AHBENR  =1&lt;&lt;1; 9 10 // ## 2. PB7 en tant que sortie 11 12 (*GPIOB).MODER &amp;= ~(0b11&lt;&lt;14); 13 (*GPIOB).MODER  = 0b01&lt;&lt;14; 14 15 16 // # B. Boucle principale 17 18 19 while(1) { 20     // ## 1. Allumer la LED 21     (*GPIOB).ODR  = 0b1&lt;&lt;7; 22     for(int k=0; k&lt;100000; k++){} 23     // ## 2. Eteindre la LED 24     (*GPIOB).ODR &amp;= ~(0b1&lt;&lt;7); 25     for(int k=0; k&lt;100000; k++){} 26 } 27 28 return 0; 29 }</pre>

 [code/code-IV-3-4.c]

 [code/code-IV-3-5.c]

## Remarque

C'est plus ou moins le même code, sauf qu'on n'a jamais eu à donner les adresses des registres que l'on utilise. Un premier survol de ce code :

- D'abord on utilise la bibliothèque `stm32l1xx.h` :

```
#include "stm32l1xx.h"
```

C'est la bibliothèque "Standard Library" de ST Microélectronics, dans sa version dédiée aux STM32L1xx, dont "le notre" fait partie : STM32L152RCT6.

- Ensuite, on voit qu'aucun des pointeurs `long*` que nous avions créé auparavant n'a été créé ici : ils existent déjà dans la bibliothèque, sous une forme plus pratique. Par exemple dans le nouveau code, la ligne 9, a été écrite :

```
(*RCC).AHBENR |= 0b11;
```

En effet, on sait, d'après la documentation, que parmi les registres `RCC`, le registre `AHBENR` est celui qui nous permet d'activer les bus pour les GPIO, ce que l'on fait ici, exactement à l'identique par rapport au code précédent, pour le `GPIOB`. Et donc au final, l'expression C `(*RCC).AHBENR` est en réalité similaire à un `long*` qui pointe à l'adresse `0x4002380+0x1C`

On pourrait avoir le même raisonnement avec la ligne 13 :

```
(*GPIOB).MODER &= ~(0b11<<14);
```

ou encore avec la ligne 22 :

```
(*GPIOB).ODR |= (0b1<<7);
```

On voit que des accès aux différents registres ont été prédéfinis à partir du mapping, et sous une forme très agréable à utiliser : les registres du `GPIOB` par exemple ont été rangés "ensemble" sous la forme de champs d'une structure.

Une façon de comprendre mieux ce qui a été fait est de regarder le contenu de la bibliothèque `stm32l1xx.h`. On y trouve ligne 162 (dans la version que j'utilise ...) :

```
#elif defined(STM32L152xC)
#include "stm32l152xc.h"
```

qui est le fichier important qui contient vraiment la description du processeur que nous utilisons. Allons dedans et regardons. Ce fichier fait plus de 9000 lignes et je donne ci-dessous des morceaux trouvés un peu partout et assemblés ici, avec l'intention de comprendre comment utiliser le port `GPIOB` :

```

1  /**
2   * @brief General Purpose IO
3   */
4
5  typedef struct
6  {
7      __IO uint32_t MODER;           /*!< GPIO port mode register */
8      __IO uint32_t OTYPER;          /*!< GPIO port output type register */
9      __IO uint32_t OSPEEDR;         /*!< GPIO port output speed register */
10     __IO uint32_t PUPDR;          /*!< GPIO port pull-up/pull-down register*/
11     __IO uint32_t IDR;            /*!< GPIO port input data register */
12     __IO uint32_t ODR;            /*!< GPIO port output data register */
13     __IO uint32_t BSRR;           /*!< GPIO port bit set/reset registerBSRR */
14     __IO uint32_t LCKR;           /*!< GPIO port configuration lock register*/
15     __IO uint32_t AFR[2];          /*!< GPIO alternate function register */
16     __IO uint32_t BRR;            /*!< GPIO bit reset register */
17 } GPIO_TypeDef;
```

```

19 // plus loin ...
20 #define PERIPH_BASE ((uint32_t)0x40000000U)
21
22 // plus loin ...
23 /*!< Peripheral memory map */
24 #define AHBPERIPH_BASE (PERIPH_BASE + 0x00020000U)
25
26 // plus loin ...
27 #define GPIOB_BASE (AHBPERIPH_BASE + 0x00000400U)
28
29 //.. plus loin
30 #define GPIOB ((GPIO_TypeDef *) GPIOB_BASE)

```

On voit très bien comment les choses sont définies : une structure **GPIO\_TypeDef** qui contient, dans l'ordre, les registres associés au **GPIOB** tels que définis dans la documentation, chacun étant un **uint32\_t** qui n'est ni plus ni moins qu'un entier 32 bits non signé, c'est à dire un synonyme de **unsigned long**.

Ensuite, on trouve diverses définitions qui aboutissent au fait que l'adresse du **GPIOB** est **0x40020400**, de façon à faire pointer la variable globale **GPIOB** à cette adresse en respectant la structure définie dans **GPIO\_TypeDef** : le fait que le pointeur **GPIOB** soit de type **GPIO\_TypeDef** ne signifie rien de plus que le fait qu'il voit la mémoire à l'adresse **0x40020400** sous la forme décrite par cette structure. On a donc bien les différents champs de la structure "calés" sur les adresses des registres, puisque tous les registres du STM32 à cet endroit sont des entiers 32 bit.

## Remarque

Ca n'est pas très important mais le "mot" **\_IO** est une définition (un **#define**) qui signifie **volatile**. Le mot **volatile** du C est un mot qui impose au compilateur de traiter de façon stricte les instructions liées à la variable considérée. En effet, quand on écrit "du code", le compilateur peut prendre des initiatives qu'il perçoit comme des optimisations mais qui peuvent aboutir à un fonctionnement non désiré. Dans le cas dans lequel nous sommes ici, **volatile**, appliqué à la définition d'une structure, signifie que le compilateur ne peut pas, par exemple, changer l'ordre des champs dans la structure (ce qui ne permettrait plus d'avoir les bons champs calés sur les bons registres par exemple), ou ajouter des octets supplémentaires de "padding" pour accélérer les traitements.

Ces opérations sont faites couramment par les compilateurs pour optimiser le programme, de façon automatique sans que cela ne semble, en apparence, perturber le programme final. Et dans un programme réalisé sur PC c'est pratiquement toujours le cas. Mais si les choses sont un peu fines, comme ici, il faut désactiver ces optimisations sur ces opérations, parce que la description de la structure doit absolument coller au positionnement des registres en mémoire, et il n'y a pas matière à vouloir optimiser ça, et c'est ce que l'on veut exprimer au compilateur. C'est précisément le rôle du mot **volatile**.

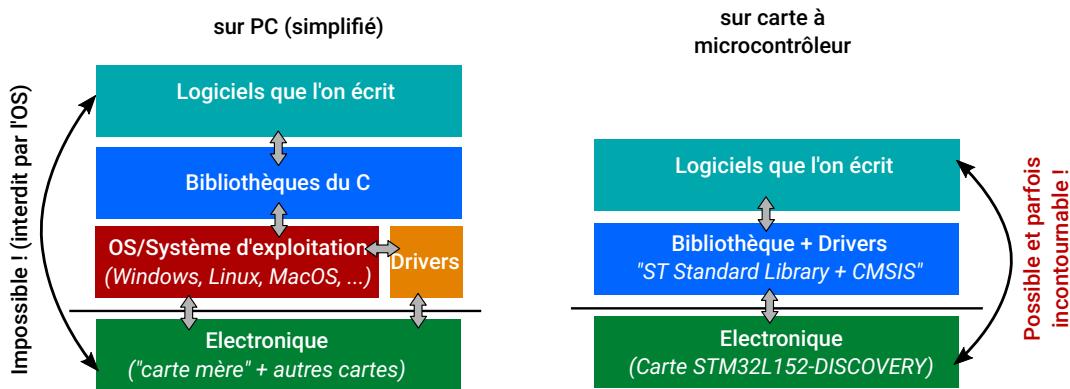
## IV.4 Vers plus d'abstraction : utilisation de CMSIS + ST Standard Library

Une ambition des bibliothèques c'est aussi de s'abstraire de l'électronique au delà encore de la **memory map** : concrètement, on voudrait aussi ne pas avoir à connaître la position des bits utiles dans les différents registres, et même, en exagérant, ne plus avoir à connaître les registres eux mêmes, mais uniquement les fonctionnalités. Les bibliothèques n'arrivent pas, en général, exactement jusqu'à ce niveau de raffinement pour la plupart, mais elles simplifient quand même beaucoup le travail. Dans cet état d'esprit, il y a un ensemble de choses disponibles, toujours dans :

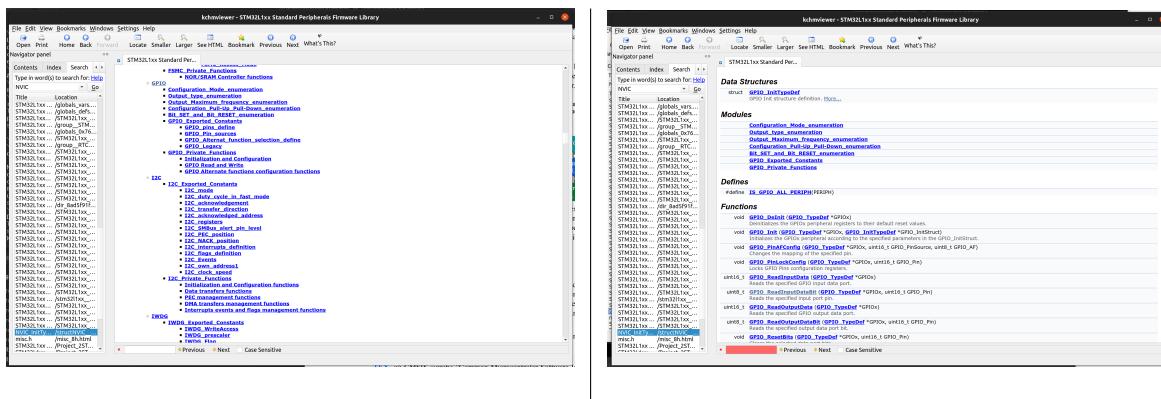
```
#include "stm32l1xx.h"
```

mais qui n'ont pas encore été décrites ici.

Si jusqu'ici ce qui était indispensable c'était connaitre le mapping et les registres, maintenant ce qu'il faut avoir sous la main c'est la documentation de la bibliothèque. L'idée est d'aboutir à quelque chose qui "ressemble" à ce qui se passe quand on programme avec un ordinateur, mais sans utiliser vraiment un système d'exploitation :



Mais il faut faire attention, la documentation de ces bibliothèques s'adresse à des développeurs chevronnés, et croire que l'on peut tout aborder juste avec cette documentation est illusoire : il faut avoir une idée du fonctionnement du microcontrôleur, tel que décrit dans les documentations déjà données, pour utiliser harmonieusement les bibliothèques. Comme souvent en C, ces bibliothèques sont formées par un ensemble de **structures** et de **fonctions**. Dans le cas qui nous concerne, la documentation de la bibliothèque standard pour les STM32L1xx est disponible [ici, sous forme d'un fichier CHM](#) (que normalement Windows sait ouvrir directement). Elle se présente comme ci-dessous :



Sur la copie d'écran de gauche on voit une liste partielle des différents "modules" que sait gérer la bibliothèque. Les modules s'apparentent en réalité à des "drivers", dans la mesure où ils permettent de gérer une fonctionnalité du microcontrôleur par bibliothèque. Ici on voit le module GPIO et le module I2C par exemple. Si on clique sur un module, on en obtient une description. On montre ici, sur la copie d'écran de droite, ce que l'on obtient si on clique sur le mot "GPIO". On observe que l'on aboutit à une description, un peu sommaire mais c'est déjà ça, des fonctions et structures que contient la bibliothèque.

Si on regarde dans cette bibliothèque, on comprend que l'on peut transformer notre code précédent qui fait clignoter une LED, en un nouveau code, qui n'a plus rien à voir, mais qui est plus lisible malgré tout .Nous comparons les deux versions ci-après.

Code "abstraction de la Memory Map"	Code "ST Standard Library"
<pre> 1 // inclusion de la librarie 2 #include "stm32l1xx.h" 3 4 int main() { 5 // # A. Configuration 6 // ## 1. Activer GPIOB (port B) 7 (*RCC).AHBENR  =1&lt;&lt;1; 8 9 // ## 2. PB7 en tant que sortie 10 (*GPIOB).MODER &amp;= ~(0b11&lt;&lt;14); 11 (*GPIOB).MODER  = 0b01&lt;&lt;14; 12 13 // # B. Boucle principale 14 15 while(1) { 16     // ## 1. Allumer la LED 17     (*GPIOB).ODR  = 0b1&lt;&lt;7; 18     for(int k=0; k&lt;100000; k++){} 19     // ## 2. Eteindre la LED 20     (*GPIOB).ODR &amp;= ~(0b1&lt;&lt;7); 21     for(int k=0; k&lt;100000; k++){} 22 } 23 return 0; 24 }</pre> <p style="text-align: right;"> [code/code-IV-4-6.c]</p>	<pre> 1 // inclusion de la librarie 2 #include "stm32l1xx.h" 3 4 int main() { 5 // # A. Configuration 6 // ## 1. Activer GPIOB (port B) 7 RCC_AHBPeriphClockCmd( 8     RCC_AHBPeriph_GPIOB ,ENABLE); 9 10 // ## 2. PB7 en tant que sortie 11 GPIO_InitTypeDef leds_PB; 12 GPIO_StructInit(&amp;leds_PB); 13 leds_PB.GPIO_Mode = 14     GPIO_Mode_OUT; 15 leds_PB.GPIO_Pin = GPIO_Pin_7; 16 GPIO_Init(GPIOB,&amp;leds_PB); 17 18 // # B. Boucle principale 19 20 while(1) { 21     // ## 1. Allumer la LED 22     GPIO_SetBits(GPIOB,GPIO_Pin_7); 23     for(int k=0; k&lt;100000; k++){} 24     // ## 2. Eteindre la LED 25     GPIO_ResetBits(GPIOB,GPIO_Pin_7 26         ); 27     for(int k=0; k&lt;100000; k++){} 28 } 29 return 0; 30 }</pre> <p style="text-align: right;"> [code/code-IV-4-7.c]</p>

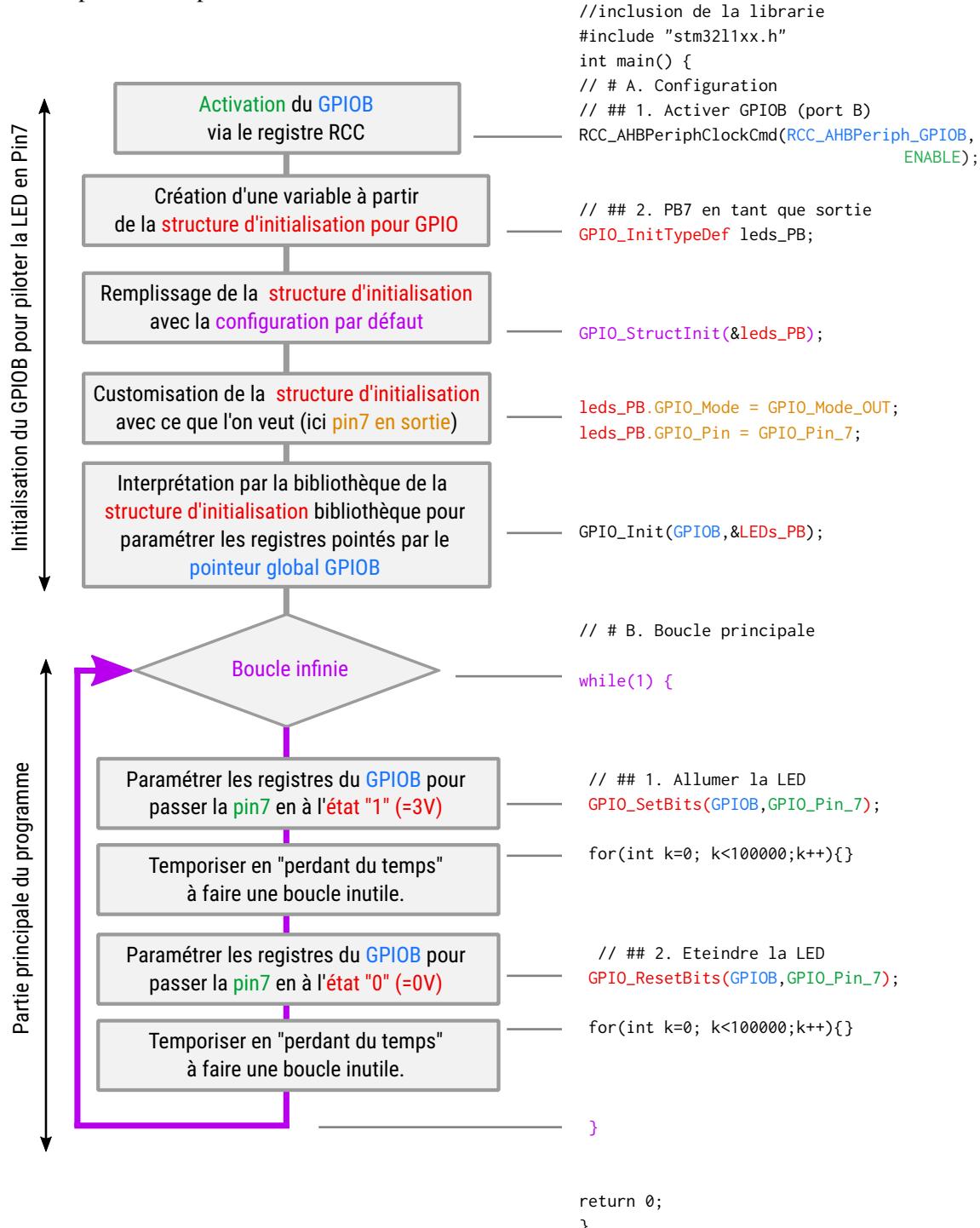
**La différence principale avec le code que nous écrivions sans bibliothèque réside donc dans le fait qu'on n'écrit plus directement dans les registres : on utilise des fonctions qui vont le faire à notre place :**

- Dans le cas de l'initialisation, et ce sera le cas pour tous les modules du microcontrôleur via la bibliothèque que nous utilisons, on commence par créer une structure dédiée à l'initialisation du module. Ces structures sont toujours nommées sur le modèle **MODULE\_xxxInitTypeDef** (où ici "MODULE" est ici "GPIO" et "xxx" n'est pas utilisé).
- Ensuite on remplit cette structure avec des valeurs par défaut, c'est à dire des valeurs qui conviennent à la plupart des applications. Ces fonctions sont toujours nommées sur le modèle **MODULE\_xxxStructInit** (où, à nouveau, "MODULE" est ici "GPIO" et "xxx" n'est pas utilisé). Notez que cette fonction utilise un passage par adresse pour modifier le contenu de la structure.
- Puis, on customise les fonctionnalités qui nous intéressent, en renseignant les différents champs de la structure que l'on veut modifier. Cela nécessite d'aller voir dans la documentation de la bibliothèque. Parfois cela peut amener aussi à retourner dans le PDF qui décrit le microcontrôleur, mais pas forcément.
- Ensuite, on demande à la bibliothèque d'interpréter la structure d'initialisation pour paramétriser les registres. C'est précisément en cela que la bibliothèque nous simplifie la vie, elle paramètre les registres qui "marchent ensemble" pour atteindre les fonctionnalités dont on a besoin. Encore une fois, avec cette bibliothèque, ces fonctions sont toujours nommées sur le même modèle, **MODULE\_xxxInit** (où, à nouveau, "MODULE" est ici "GPIO" et "xxx" n'est pas utilisé).

- On a alors fait, en général, le principal, et il nous reste à utiliser le module. Là c'est plus spécifique et dépend de ce que fait le module. Pour nous, c'était les fonctions `GPIO_SetBits` et `GPIO_ResetBits`.

## Remarque

Ce code peut se comprendre comme suit :



On le voit là encore, la plupart des applications consistent à avoir une phase d'initialisation de tous les modules que l'on utilise, suivie d'une boucle principale qui est le programme en lui-même. C'est l'architecture "de base" des programmes à microcontrôleur "simples", qui n'utilisent pas de

fonctionnalités trop avancées, telles que les "interruptions" ou le "DMA" (Direct Memory Access) par exemple. Nous verrons quelques uns de ces aspects, les autres seront à découvrir selon vos besoins.

**C'est la façon usuelle de programmer sur microcontrôleur : utiliser au maximum des bibliothèques et, parfois (rarement), utiliser directement les registres.**

### Remarque

Un peu de méthodologie : face à un code comme ça, "la première fois", ca peut sembler être la panique. Vous avez deux ressources avec vous :

- C'est un exemple qui fait peu de lignes. Alors il est raisonnable d'avoir l'approche suivante : si vous rencontrez un symbole qui n'est ni un mot-clé du C (comme `while` par exemple), ni une variable qui a été créée (comme `leds_PB` dans le code ci-dessus), alors c'est quelque chose qui a été défini dans la bibliothèque
- si c'est quelque chose qui a été défini dans la bibliothèque, vous avez deux sources de documentation : la documentation elle-même, c'est à dire dans notre cas le fichier [fichier CHM](#) mentionné plus haut, mais aussi le contenu du fichier .h inclus, `stm32l1xx.h`, qui est présent dans le projet et dans lequel vous pouvez faire des "rechercher ..." avec l'éditeur de texte de l'environnement de développement.

Bien entendu, tout cela suppose d'avoir les idées claires sur le C lui-même.

## IV.5 Autres Bibliothèques

ST propose aussi la **bibliothèque "HAL"**, qui elle est **spécifique aux STM32**, qui signifie "Hardware Abstraction Layer", et qui prétend permettre de s'éloigner encore plus du besoin de documentation sur le processeur donné. Elle a l'avantage de permettre d'écrire du code que l'on n'a pas à adapter en changeant de modèle de microcontrôleur STM32. Malgré tout, elle est incomplète, ce qui oblige à utiliser aussi par endroits du code type "ST standard library" ou dans des registres, ce qui rend à nouveau le code dépendant d'un microcontrôleur spécifique.

Enfin, on peut mentionner le concept de "Middleware". Un Middleware est un "driver étendu", c'est à dire un driver qui, en plus de gérer l'aspect électronique, apporte des fonctionnalités que l'on trouverait habituellement sur un système d'exploitation. Le cas typique est la gestion d'une carte SD avec un microcontrôleur : Un driver permettrait de gérer les signaux pour accéder à telle ou telle "case" de la carte SD. Un Middleware ajouterait la gestion d'un formatage, permettant de stocker des fichiers plutôt que des données brutes. Un autre exemple de Middleware est freeRTOS, une bibliothèque qui permet de gérer du multitâche.

**Dans la suite de ce cours, nous allons exclusivement travailler avec la ST Standard Library.**

### En résumé ...

L'utilisation de bibliothèques rend plus "lisible" le code que l'on génère : même sans avoir le datasheet sous les yeux, on peut comprendre, un peu superficiellement, ce que fait le code, et c'est déjà très important. En outre, pour des sous ensembles plus complexes, elle permet d'écrire du code plus rapidement. Mais dorénavant, on a besoin d'une nouvelle source de documentation : celle de la bibliothèque elle-même, et idéalement quelques exemples de son utilisation. Dans la suite de ce cours, c'est l'approche que nous allons continuer à avoir, en lieu et place de l'approche "bare-metal". ■

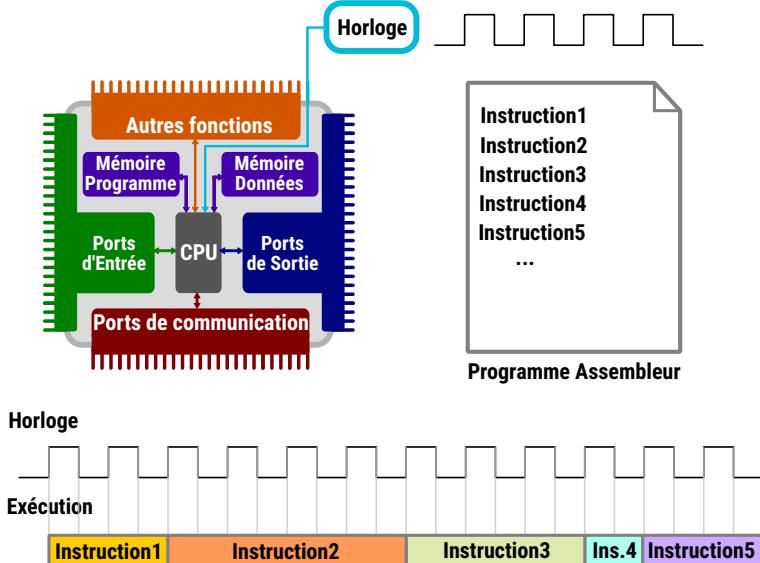
# Chapitre V – Sous-ensemble 2 : Gestion du temps et Timers

## ⌚ Motivations et objectifs

La gestion et le contrôle du temps sont fondamentaux sur un système embarqué à microcontrôleur. Notamment, on a besoin de synchroniser des signaux, de gérer des temps d'attente, etc. A cet effet, tous les microcontrôleurs comportent un ou plusieurs "Timers", qui le permettent, et qui se basent sur des compteurs en général. C'est ce que l'on va décrire.

## V.1 Constante de temps associée à un processeur : "CPU clock"

La plus grande part des processeurs utilisent une **horloge** pour gérer les signaux. Cette horloge est un signal en créneau qui sert de base pour parler de la cadence à laquelle les instructions (assembleur) sont exécutées. On montre ci-dessous le processus d'exécution d'un programme :



Les instructions n'ont pas nécessairement le même temps d'exécution (mais pour la plupart c'est souvent une seule période, appelée "temps de cycle d'horloge").

Cette horloge est appelée "cpu clock". Elle est en général assez précise et va servir, dans la plupart des situations, de base de temps pour l'ensemble des autres fonctionnalités dont on a besoin dans le microcontrôleur.

On comprend bien que ce temps est le temps le plus court auquel on est confronté avec le microcontrôleur (on ne peut pas aller plus vite que le temps nécessaire à une instruction rapide). Il est souvent de l'ordre de quelques MHz à quelques GHz, selon le processeur et l'application visée. Pour nous, ce sera  $16\text{MHz}$  pour le STML152RC6 que nous utilisons. Cela correspond à des périodes de  $62.5 \times 10^{-9}\text{s} = 62.5\text{ns}$ . C'est beaucoup plus rapide que la plupart des signaux que l'on a à gérer. Par exemple si on veut faire une simple temporisation pour faire clignoter les LED, on va vouloir des temps un peu inférieurs à la seconde, soit environ 6 ordres de grandeurs plus longs !

En effet, si on reprend l'exemple du clignotement des LED, on a utilisé, pour ralentir le processeur, des boucles du type :

```
for (int k=0; k<100000; k++) {}
```

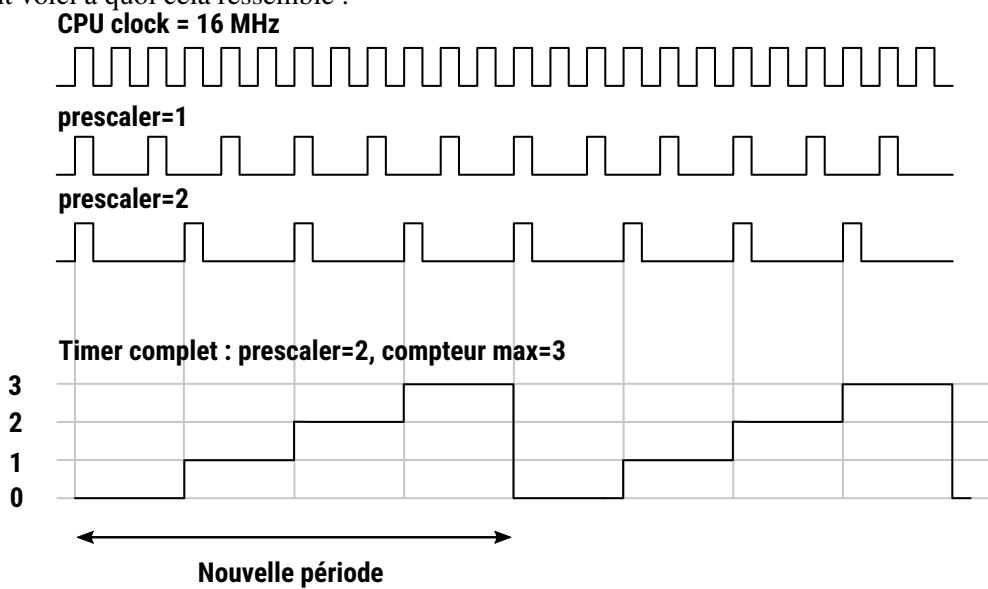
Or c'est assez empirique et peu précis. Imaginons par exemple que l'on fasse fonctionner plusieurs choses en parallèle (on verra comment plus loin), il est clair que cela va ralentir le processeur. Si on veut gérer des temps précis, il serait plus prudent de mesurer vraiment le temps qui s'écoule au lieu de jouer avec le ralentissement/l'occupation du processeur. C'est ce que l'on va faire.

## V.2 Principe de fonctionnement des Timers

Les microcontrôleurs contiennent souvent 1 ou plusieurs "Timers". Ces sous-ensembles sont dédiés à la gestion du temps. Le principe n'est pas très compliqué et repose sur 2 principes :

- générer de nouvelles horloges dont la période sont des multiples de la période de la "CPU clock". Cela donne lieu à un coefficient que l'on nomme "prescaler".
- On peut obtenir des temps encore plus longs en comptant un certain nombre de coups de l'une de ces horloges. Cela se fait avec un "compteur".

Un timer complet combine les deux et on peut le paramétrier via les registres du microcontrôleur. Schématiquement voici à quoi cela ressemble :



$$\text{Nouvelle période} = \text{CPUclock} \times (\text{prescaler}+1) \times (\text{compteurMax}+1)$$

Ainsi, la période du Timer peut s'écrire :

$$T_{\text{timer}} = T_{\text{CPUClock}} \times (\text{prescaler} + 1) \times (\text{compteurMax} + 1) = \frac{(\text{prescaler} + 1)(\text{compteurMax} + 1)}{f_{\text{CPUClock}}}$$

qui illustre juste le fait que le prescaler "saute" un certain nombre de tops d'horloge de la clock CPU, et donc la démultiplie, et que le compteur fait de même en "comptant" un certain nombre de "tops" de l'horloge démultipliée par le prescaler.

Les périodes accessibles avec ce système dépendent donc des valeurs maximales possibles avec le prescaler et le compteur. Si les registres sont 16 bit (c'est le cas avec la plupart des timers du STM32), on a des valeurs maximales de registre qui valent  $2^{16} - 1 = 65535$ , ce qui revient, avec une horloge à 16 MHz, à des périodes maximales de :

$$T_{\text{max}} = \frac{1}{16 \times 10^6} \times (65535 + 1) \times (65535 + 1) \approx 273 \text{ s}$$

Donc des périodes très longues. Autrement dit, on peut obtenir des périodes situés entre 62.5 ns et 273 s !

Le STM32 dispose aussi d'un timer basé sur des registres 32 bit (le Timer 5), ce qui multiplie le temps maximum de la période à des temps  $\approx 4 \times 10^9$  fois plus importants !

## V.3 Que faut-il faire concrètement ?

Avec le microcontrôleur voici ce qu'il reste à faire :

- D'abord, exprimer que l'on va utiliser un timer via les registres AHB et ponts associés (voir sur le schéma de l'architecture), page 21.
- Puis, choisir l'un des 10 timers du STM32L152. Ce choix est lié au fait que peut être on en utilise déjà certains dans le programme en cours, ou selon des caractéristiques propres. Par exemple sur notre microcontrôleur, ils utilisent tous un prescaler et un compteur 16 bit, sauf un (le Timer 5) qui fonctionne en 32 bit, ce qui permet d'accéder à des temps plus longs.
- Ensuite, paramétriser le Timer avec la valeur de prescaler que l'on veut, et la valeur de compteur que l'on veut, de façon à ce que la période convienne
- A partir de là, on peut interroger un registre qui contient la valeur courante du compteur, ce qui permet de se repérer dans le temps.

## V.4 Un exemple : une fonction pour "attendre" un temps calibré

On va créer une fonction pour permettre de faire des délais maîtrisés pour les temps de clignottement des LED. On veut donc aboutir à une fonction :

```
void TIM5_delay_ms(int délai);
```

qui attend un temps calibré, donné en millisecondes, et basé sur l'utilisation du timer 5 (TIM5). Alors commençons.

On va d'abord paramétriser le timer. Pour faire simple, l'idée est de créer un timer avec un prescaler tel que l'on a une base de temps de 1 ms, et à ce moment là le délai sera une valeur en milliseconde qui définira la valeur maximale du compteur. En faisant ça avec des registres 16 bit, on a au maximum des temps d'attente de 65 secondes, ce que l'on peut trouver dommage (on pourrait avoir une application avec un microcontrôleur qui reste allumé des jours entiers par exemple, et vouloir attendre des jours entiers au besoin). Alors on va choisir le seul timer 32 bit du STM32L152, pour se donner des possibilités bien plus longues en temps.

### Remarque

On va utiliser les "recettes" que l'on a compris avec l'utilisation des GPIO par la bibliothèque : on s'attend à avoir besoin de chercher, dans le module "TIM", d'une structure portant un nom en **TIM\_xxxInitTypeDef**, une fonction d'initialisation en **TIM\_xxxStructInit** et une autre fonction en **TIM\_xxxInit**. En cherchant un peu on obtient le code ci-dessous.

```

1 #include "stm32l1xx.h"
2
3 // fonctions pour configurer et utiliser le timer 5.
4 // voir après le main
5 void TIM5_Config();
6 void TIM5_delay_ms(int delay);
7
8 int main(void)
9 {
10     TIM5_Config();
11
12     /* Activer GPIOB sur AHB */

```

```

13     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB ,ENABLE);
14     /* Configurer PB7 */
15     GPIO_InitTypeDef gpio_b;
16     GPIO_StructInit(&gpio_b);
17     gpio_b.GPIO_Mode = GPIO_Mode_OUT;
18     gpio_b.GPIO_Pin = GPIO_Pin_7;
19     GPIO_Init(GPIOB ,&gpio_b);
20
21     while(1) {
22         GPIO_SetBits(GPIOB ,GPIO_Pin_7);
23         TIM5_delay_ms(500);
24         GPIO_ResetBits(GPIOB ,GPIO_Pin_7);
25         TIM5_delay_ms(500);
26     }
27 }
28
29 // ### TIM5 pour delai
30 // Configuration
31 void TIM5_Config()
32 {
33     /* Activer TIM5 sur APB1 */
34     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5 ,ENABLE);
35     /* Configurer TIM5 : prescaler a 1 ms, periode au maximum*/
36     TIM_TimeBaseInitTypeDef DelayTimer;
37     TIM_TimeBaseStructInit(&DelayTimer);
38     DelayTimer.TIM_Prescaler = 16000-1;
39     DelayTimer.TIM_Period = 0xFFFFFFFFU;
40     TIM_TimeBaseInit(TIM5 ,&DelayTimer);
41 }
42
43 // Fonction Delai en millisecondes
44 void TIM5_delay_ms(int delay)
45 {
46     TIM_SetCounter(TIM5 ,0);
47     TIM_Cmd(TIM5 , ENABLE);
48     while(TIM_GetCounter(TIM5)<delay)
49     {
50     }
51     TIM_Cmd(TIM5 , DISABLE);
52 }
```

 [code/code-V-4-8.c]

On a donc créé une fonction d'initialisation qui donne un prescaler à **16000-1**, ce qui donne une base de temps de  $((16000 - 1) + 1) \times 1/(16 \times 10^6)$ , et avec le compteur au maximum possible. Ensuite, on crée la fonction **TIM5\_delay\_ms** qui place la valeur du compteur, active le compteur, puis, dans une boucle, attend que le compteur atteigne la valeur du temps qui correspond. Une fois sorti de la boucle, on arrête le timer.

Ensuite, dans le code principal, on appelle la fonction d'initialisation, puis, dans la boucle, on invoque simplement la fonction de délai en remplacement des boucles **for**

## ❖ En résumé ...

Nous avons vu un sous-ensemble **essentiel** du micro-contrôleur : le Timer. Nous l'avons utilisé pour faire jusqu'ici quelque chose de simple mais utile, une fonction pour attendre un temps calibré en millisecondes. C'est utile dans beaucoup de circonstances, par exemple quand on envoie des données à un autre composant connecté au microcontrôleur, il y a souvent des temps d'attente à gérer. Mais nous verrons que les Timers servent à beaucoup d'autres choses. ■

# Chapitre VI – Sous-ensemble 3 : Gestion des signaux analogiques

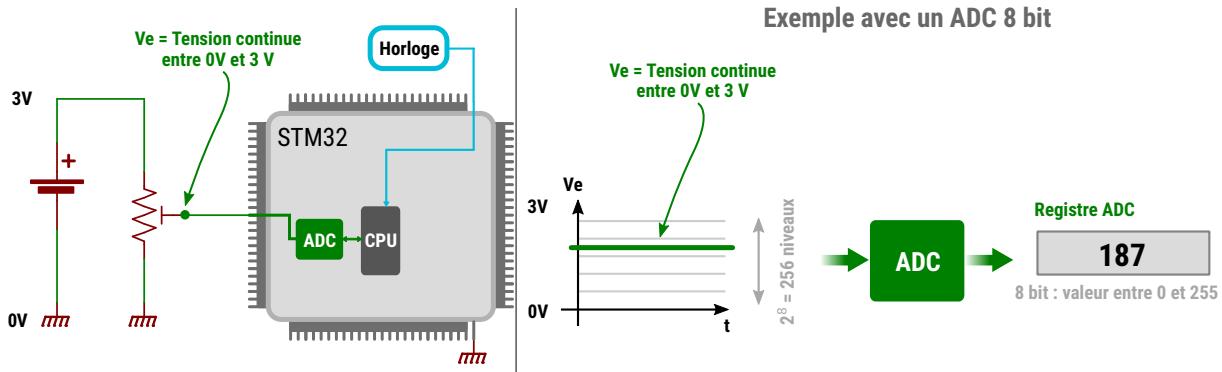
## ⌚ Motivations et objectifs

Nous avons vu comment gérer des signaux numériques, "tout-ou-rien", avec un microcontrôleur. Mais les microcontrôleurs sont aussi faits pour gérer des signaux analogiques, et c'est extrêmement utile. C'est ce que nous allons regarder ici.

### VI.1 Entrée analogique : ADC (Analog to Digital Converter)

#### Présentation Générale

Ce que l'on cherche à faire ici, c'est, en première approximation, récupérer la valeur d'une tension, peu importe ce qu'elle est, présente à l'extérieur du microcontrôleur : c'est "autre chose" qu'une tension qui vaut soit 0V pour représenter un 0 logique, soit 3V pour représenter un 1 : ca doit pouvoir être n'importe quelle valeur. On fait cela avec un "ADC", que l'on trouve souvent intégré au microcontrôleur (c'est le cas pour le STM32), mais qui pourrait aussi être un composant externe.



Dans un premier temps pour simplifier on va parler de tensions continues. Il faut prendre en compte quelques limitations :

- Il y a une limite à la tension minimale et à la tension maximale. En général, c'est simplement la tension d'alimentation du microcontrôleur (mais ça pourrait être différent), autrement dit pour nous c'est une tension entre 0 V et 3 V.
- Il y a une limite à la précision que l'on a sur cette tension. En effet, une tension analogique est une tension avec une précision infinie<sup>1</sup> (elle pourrait avoir un "nombre de chiffres après la virgule" infini), mais l'infini n'est pas gérable par un processeur : vouloir stocker un nombre infini de chiffres revient à disposer d'une mémoire infinie. Alors cette précision est limitée. Par exemple, elle peut être limitée à 8 bit, ce qui revient à dire qu'on dispose de 256 niveaux, ce qui revient à dire que l'on a une précision de  $1/255 \approx 0.5\%$ . C'est la même chose avec les nombres que peut stocker un calculateur, il a toujours une certaine précision (en calcul scientifique elle est de l'ordre de  $10^{-13}\%$ ).
- Cette conversion prend "un certain temps", et il faut le prendre en compte.

1. En fait on pourrait utiliser le bruit comme critère qui limite la précision d'un signal analogique, mais dans beaucoup de cas c'est un critère trop drastique par rapport aux besoins

Les convertisseurs analogique-numérique ne cherchent pas à fournir la valeur de la tension analogique. Par exemple, la réponse de l'ADC sera pas 2.14 V si il voit une tension de 2.14 V. Ils vont se contenter de fournir une valeur entière en "pleine échelle" par rapport au nombre de bits qui les caractérise. Par exemple, si on a une tension entre 0 V et 3 V et que l'on a un ADC 8 bit, alors 0V est codé par la valeur 0 et 3 V est codé par la valeur 255. Une tension intermédiaire s'obtient par une simple règle de 3. Ainsi, pour calculer la tension injectée sur une patte à partir de la valeur obtenue dans le registre, on fait simplement :

$$\text{tension} = \frac{\text{valeur duregistre}}{2^{\text{nbBits}} - 1} \times V_{\max}$$

Autrement dit pour nous :

$$\text{tension} = \frac{\text{valeur duregistre}}{255} \times 3$$

Et donc la précision en tension vaut, pour notre cas,  $3/255 = 11mV$ .

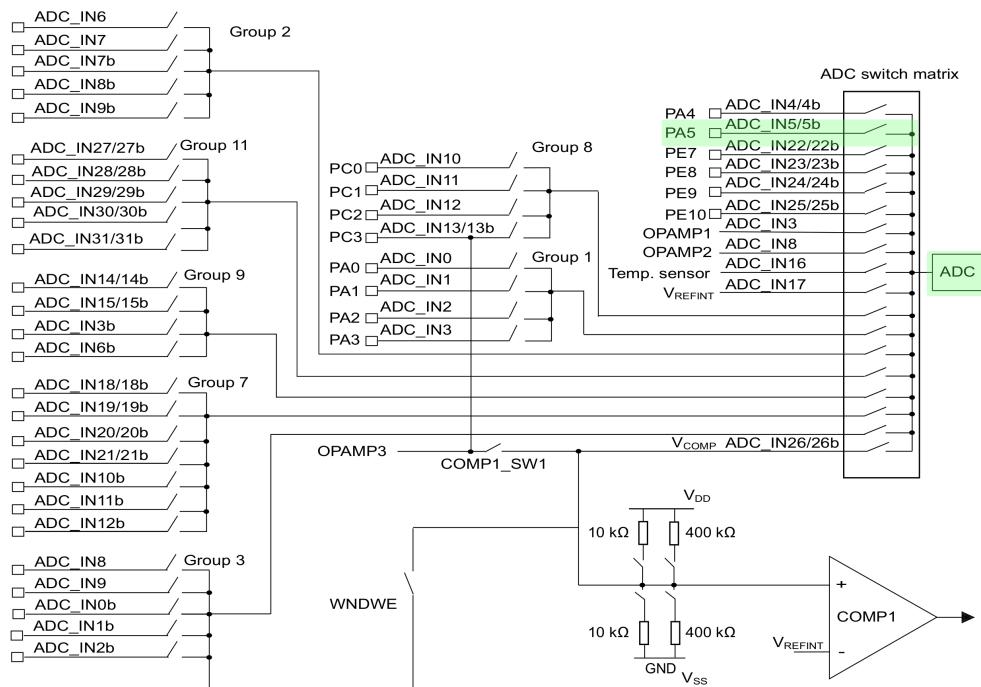
## En résumé ...

On a vu qu'un ADC a 3 caractéristiques :

- une plage de tensions d'entrée (par exemple 0 à 3 V)
- un nombre de bits qui définissent les valeurs qu'il va associer à cette plage de tensions, et donc sa précision (par exemple 8 bit signifie une plage de valeurs 0 à 256)
- un temps de conversion

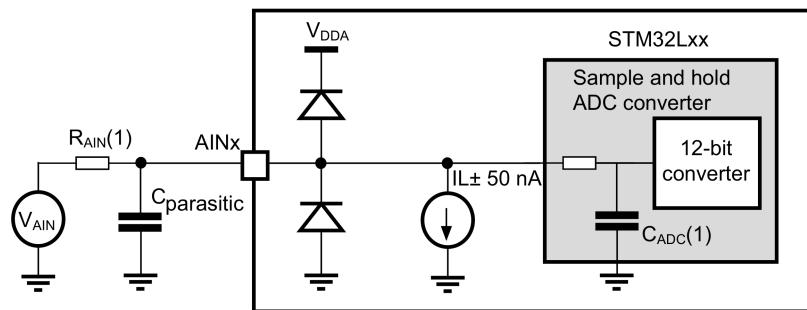
## ADC sur STM32L152

Les ADC sont des composants qui coutent cher, et donc il n'y en n'a pas toujours, et quand il y en a il n'y en n'a pas beaucoup. Sur le STM32L152, il y en a 1 seul, mais il peut être utilisé sur plusieurs pins successivement, comme le montre cet extrait de la documentation technique (issu de [PDF RM0038](#) page 192) :



Ainsi, au niveau logiciel, on va pouvoir paramétriser un ou plusieurs GPIO pour prendre la fonction "ADC", et ils pourront être interrogés tour à tour par le seul ADC intégré au STM32. Ainsi, il est impossible de mesurer rigoureusement simultanément deux signaux. D'autres microcontrôleurs disposent physiquement de plusieurs ADC pour obtenir un tel fonctionnement.

Sur STM32, il faut paramétrer le GPIO que l'on veut associer à la fonction "ADC" en exprimant qu'il va fonctionner en "mode analogique". Parallèlement, il faut paramétrer le ADC pour qu'il prenne le nombre de bits désiré (sur STM32L152 on a le choix entre 6, 8 et 12 bit). On peut aussi demander à l'ADC de réaliser plusieurs conversions successives sur la même pin pour échantillonner un signal mais c'est un mécanisme plus sophistiqué que notre besoin : nous indiquerons que l'on ne veut qu'une seule acquisition. Enfin, il faut indiquer quel "temps de conversion" on veut, rapide ou lent. Ce temps de conversion impacte en fait l'impédance d'entrée de l'ADC. En effet, les ADC ont une impédance d'entrée assez capacitive, comme on le voit dans ce graphe extrait du datasheet du STM32L152 (page 211 de ce document) :



En simplifiant, elle est donc de la forme :

$$|Z_{ADC}| = \left| \frac{1}{jC_{ADC}\omega} \right| = \frac{1}{C_{ADC}\omega} = \frac{1}{2\pi C_{ADC}f}$$

Cette impédance décroît donc quand on augmente la fréquence, et donc une acquisition rapide répétée conduit à une réduction de l'impédance d'entrée, ce qui est une dégradation : l'ADC devient un "mauvais voltmètre", et il n'est pas garanti que la source de tension que l'on mesure ait la qualité requise pour ça (ou alors il faut ajouter un peu d'électronique).

Dans tous les cas : il faudra de toute façon écrire le code nécessaire pour attendre que l'ADC ait terminé sa conversion. Pour le reste, cela reprend la logique que l'on a déjà vu pour les GPIO et les TIM : le sous-block ADC utilisent les mêmes points de repères et une syntaxe similaire pour l'initialisation.

Après un peu d'étude des documentations, il vient le code ci-dessous, qui ne fait "rien" en apparence, mais fait l'acquisition d'un signal par l'ADC :

```

1 #include "stm32l1xx.h"
2
3 void ADC1_Config();
4 uint16_t ADC1_Get(uint8_t ch);
5
6 int main(void)
7 {
8     ADC1_Config();
9     for(;;)
10    {
11        volatile uint16_t adcVal;
12        adcVal = ADC1_Get(ADC_Channel_5);
13        // ici adcVal vaut la valeur lue sur PA5
14    }
15 }
16
17 // ### ADC1 sur PA5
18 // Configuration
19 void ADC1_Config()
20 { /* Activer GPIOA sur AHB */}

```

```

21     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA , ENABLE);
22     /* Parametrer PA5 en mode analogique */
23     GPIO_InitTypeDef gpio_a;
24     gpio_a.GPIO_Pin = GPIO_Pin_5;
25     gpio_a.GPIO_Mode = GPIO_Mode_AN;
26     gpio_a.GPIO_PuPd = GPIO_PuPd_NOPULL ;
27     GPIO_Init(GPIOA, &gpio_a);
28
29     /* Activer ADC1 sur APB2 */
30     RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 , ENABLE);
31     /* Configurer ADC1 en 12 bit, simple acquisition */
32     ADC_InitTypeDef adc_1;
33     ADC_StructInit(&adc_1);
34     adc_1.ADC_NbrOfConversion = 1;
35     adc_1.ADC_Resolution = ADC_Resolution_12b;
36     ADC_Init(ADC1, &adc_1);
37     ADC_Cmd(ADC1, ENABLE);
38 }
39
40 // Acquisition d'une valeur
41 // PA5 : ch = ADC_Channel_5
42 uint16_t ADC1_Get(uint8_t ch)
43 {
44     ADC-RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_4Cycles);
45     ADC_SoftwareStartConv(ADC1);
46     while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET)
47         ;
48
49     return ADC_GetConversionValue(ADC1);
50 }
```

 [code/code-VI-1-9.c]

## VI.2 Sortie analogique type DAC (Digital to Analog Converter)

Le DAC est le composant réciproque du ADC : à partir d'un registre du microcontrôleur, qui prend une valeur entière sur un certain nombre de bits, on génère une tension analogique, entre 0 et 3V.

Le STM32 dispose de 2 DAC, fonctionnant en 12 bit par défaut. Les pin dédiées sont PA4 et PA5, voir RM0038. Le code est assez similaire avec la partie ADC.

```

1 #include "stm32l1xx.h"
2
3 void DAC1_Config();
4 void DAC1_Set(uint16_t value);
5
6 int main(void)
7 {
8     DAC1_Config();
9     uint16_t dac_value = 0;
10    while(1) {
11        if(dac_value <= 0) {
12            dac_value = 0xffff;
13        }
14        DAC1_Set(dac_value);
15        for(int k=0; k<200000; k++) { } // perdre du temps pour "attendre"
16        dac_value = dac_value-10;
17    }
}
```

```

18 }
19
20 // ### DAC1 (DAC Channel 1) sur PA4
21 // Configuration
22 void DAC1_Config()
23 {
24     /*Activer GPIOA sur AHB */
25     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA , ENABLE);
26     /* Configurer PA4 en mode analogique*/
27     GPIO_InitTypeDef gpio_a;
28     GPIO_StructInit(&gpio_a);
29     gpio_a.GPIO_Mode = GPIO_Mode_AN;
30     gpio_a.GPIO_Pin = GPIO_Pin_4;
31     GPIO_Init(GPIOA, &gpio_a);
32
33     /*Activer DAC sur APB1 */
34     RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC , ENABLE);
35     /* Configurer DAC1 avec parametres par defaut */
36     DAC_InitTypeDef dac_1;
37     DAC_StructInit(&dac_1);
38     DAC_Init(DAC_Channel_1, &dac_1);
39     /* Activer DAC1 */
40     DAC_Cmd(DAC_Channel_1 , ENABLE);
41 }
42
43 void DAC1_Set(uint16_t value)
44 {
45     DAC_SetChannel1Data( DAC_Align_12b_R , value );
46     DAC_SoftwareTriggerCmd( DAC_Channel_1 , ENABLE );
47 }
48 }
```

 [code/code-VI-2-10.c]

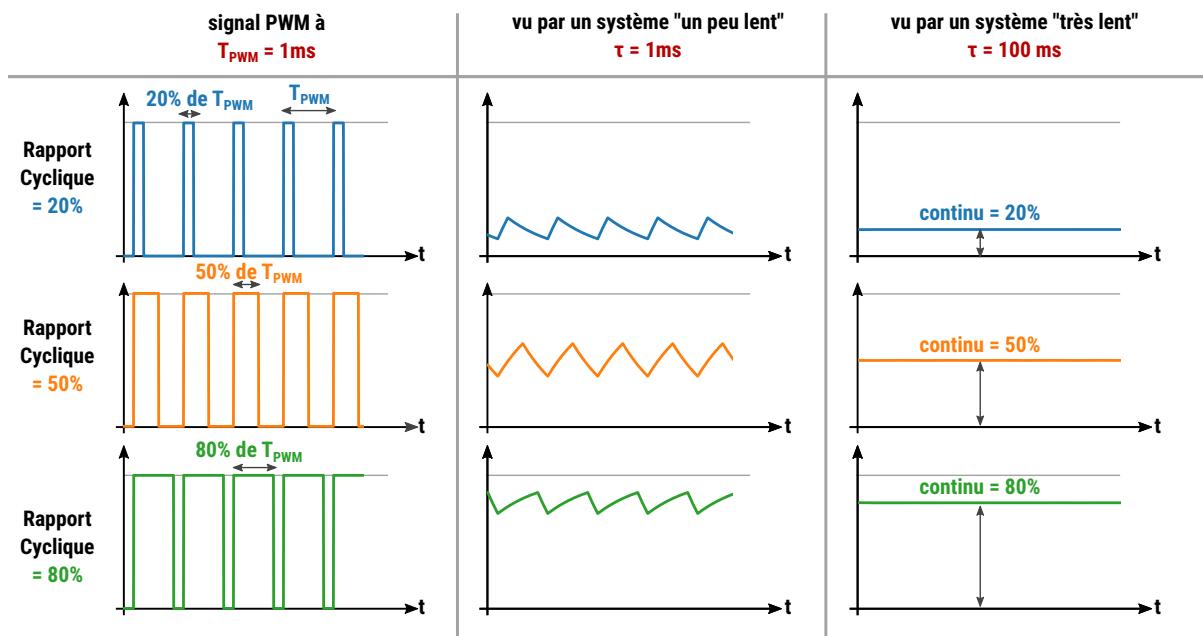
## VI.3 Sortie analogique type PWM (Pulse Width Modulation)

### Principe

Le STM32L152 dispose de seulement 2 sorties DAC. En réalité les DAC ne sont pas la façon la plus habituelle, pour des microcontrôleurs, de générer des signaux analogiques. Les DAC devraient être réservés à des applications où des signaux de "haute qualité" sont nécessaires, et à assez haute fréquence (au regard de la clock). Mais il est fréquent que l'on ait besoin de signaux de qualité plus moyenne ou que l'on soit face à des systèmes lents, toujours au regard de la clock. Dans ce genre de cas, qui est extrêmement habituel, on peut utiliser le mode "PWM".

Pour illustrer à quel point ce fonctionnement est plus habituel, pour un microcontrôleur, que l'utilisation d'un DAC, il suffit de regarder combien de sorties PWM sont possibles sur le STM32L152 : il y en a 20, soit 10 fois plus que de sorties DAC ! Et de ce point de vue le STM32L152 n'a rien de spécifique, c'est même assez classique. Alors décrivons le. Elle sera comparée, plus loin, au fonctionnement par DAC.

PWM signifie "Pulse Width Modulation". C'est une méthode efficace et économique pour générer des signaux analogiques, mais elle n'est pas, au premier abord, intuitive. Commençons par générer un signal continu en PWM. L'idée du PWM est de "coder" un signal continu par un signal en créneau à alternances positives uniquement et dont on adapte le rapport cyclique, en faisant confiance à l'idée que ce signal va passer à un système "suffisamment lent" :



Sur cette illustration on doit remarquer deux choses :

- Si on fait passer le signal PWM à travers un système "très lent", on obtient du continu
- Ce continu a une amplitude qui est proportionnelle au rapport cyclique du signal PWM

Autrement dit : pour un système suffisamment lent, un signal continu d'une certaine amplitude OU un signal créneau avec un rapport cyclique donné, c'est la même chose ! Or un signal créneau c'est intéressant parce que c'est un signal tout-ou-rien dans lequel l'amplitude est "codée" à travers le rapport cyclique. C'est donc "plus facile" au moins sur le plan de l'électronique interne du microcontrôleur, on en discutera plus loin.

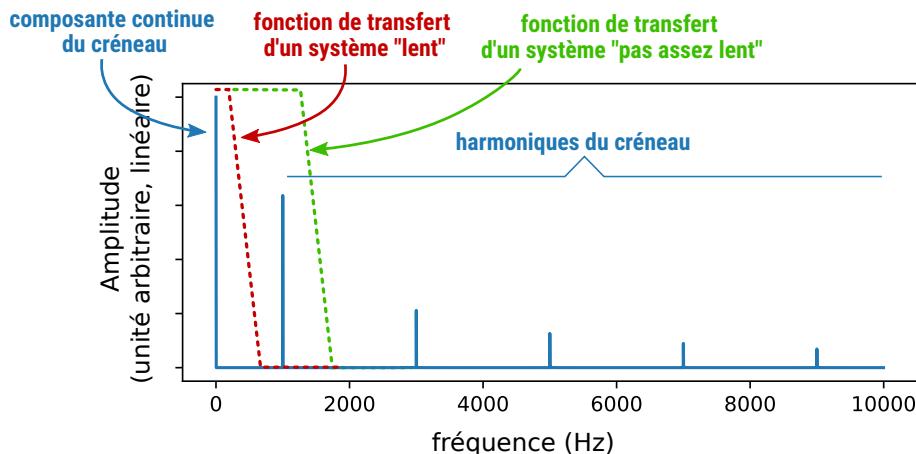
Mais intuitivement, cette idée qu'un signal rapide peut être perçu comme du continu, c'est quelque chose que vous savez déjà et dont vous faites l'expérience :

- Si on prend le cas de l'oeil : Vous savez que le principe du cinéma est de fournir plus de 20 images par seconde. Ainsi tout changement plus rapide que ça n'est pas visible par l'oeil. Ainsi, un signal PWM peut être un moyen de changer l'intensité lumineuse d'une LED, il suffit que la fréquence PWM soit bien plus rapide que 20 Hz, par exemple 2kHz.
- Cela concerne beaucoup d'autres systèmes, par exemple piloter des résistances chauffantes : la chaleur est un phénomène lent qui se prête très bien aux signaux PWM. Votre radiateur électrique, chez vous, fonctionne plus ou moins sur ce principe : il ne fonctionne pas en injectant un signal continu constant dans le radiateur, il travaille en s'activant au maximum quand la température est trop basse, et en s'éteignant quand la température est trop élevée. Et "pour vous" ça ne change presque rien, vous n'avez pas la sensation que la pièce change de température, parce qu'il y a une très grande inertie de la part du volume de la pièce. Ça n'est pas précisément du PWM mais ça montre que des variations importantes peuvent passer inaperçues si elles sont imposées à un système qui a une grande inertie.

Mais au final, "c'est quoi" un "système lent" ? Un système lent, c'est un système qui ne réagit pas aux variations rapides, autrement dit qui filtre les variations rapides. Et au sens de Fourier, les variations rapides sont représentées par les hautes fréquences. Autrement dit : un "système lent" est un système dont la fonction de transfert s'apparente à celle d'un filtre passe-bas. C'est bien sûr approxitatif mais ça suffit largement pour réfléchir, et réfléchir avec un simple premier ordre suffit également.

Alors, ce que l'on peut dire, c'est qu'un système "très lent" est un système qui filtre "tout" sauf la composante continue, c'est à dire la composante de Fourier obtenue pour  $f = 0$ . Et donc, en PWM, on utilise cette propriété qui est qu'un système lent peut se modéliser simplement par un filtre passe-bas dont la fréquence de coupure est "suffisamment proche" de  $f = 0$ . Ainsi, la correspondance entre valeur

moyenne et rapport cyclique, que l'on a observé plus haut, est tout à fait prévisible. Voici le spectre d'un signal créneau superposé avec la fonction de transfert de deux systèmes, un "lent" et un "pas assez lent" :



On voit bien que si le système n'est pas "assez lent", sa fonction de transfert est alors assez large en fréquence pour laisser passer des harmoniques du créneau, et donc le signal que l'on obtient n'est pas un signal continu. Si le système est assez lent, on voit bien qu'il n'y a "que" la composante continue dans le gabarit de la fonction de transfert, et donc on aura un signal de sortie quasi continu.

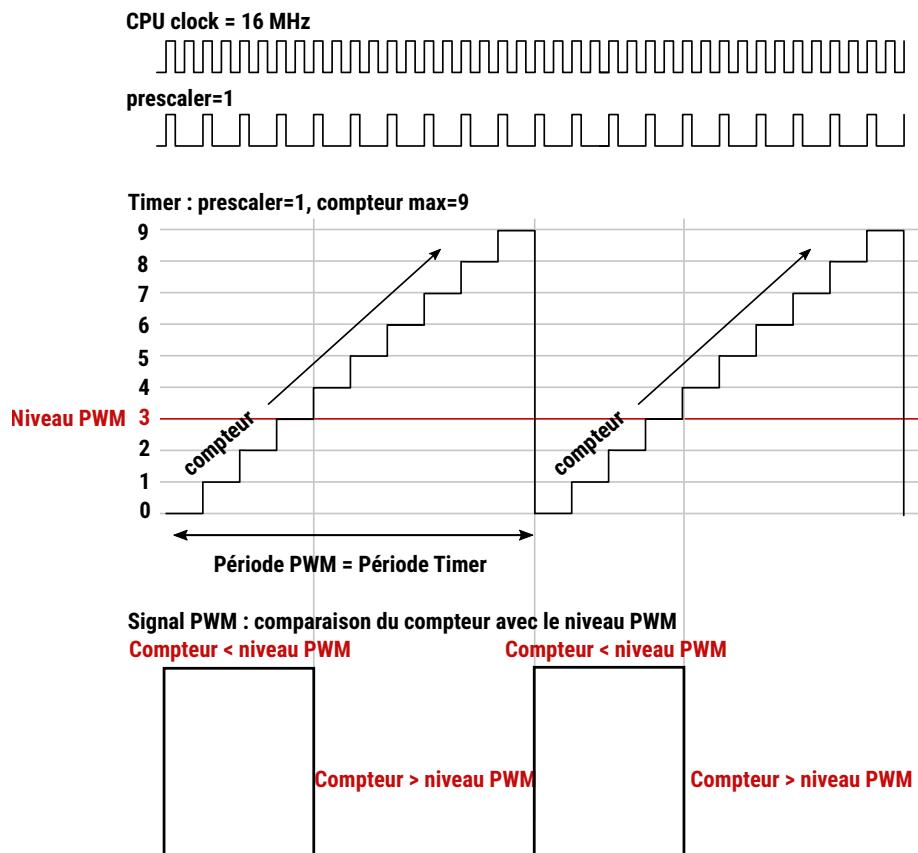
Or, la composante continue de la série de Fourier d'un signal créneau s'écrit :

$$\text{continu} = V_{max} \times \text{RapportCyclique}$$

qui exprime bien la proportionnalité entre le continu et le rapport cyclique.

### Générer des signaux PWM avec un STM32L152

Les signaux PWM utilisent le système de compteur du microcontrôleur. En effet, reprenons le système d'horloges et de Timer dont nous avons déjà parlé :



On a représenté aussi comment se passe la génération du signal PWM. La génération du signal PWM utilise simplement la comparaison avec le compteur : lorsque l'on utilise le PWM, on dispose d'un nouveau registre, nommé "pulse". La valeur de ce registre est utilisée comme suit : le signal est d'abord généré à l'état haut. Ensuite, dès que le compteur dépasse la valeur de "pulse", le signal repasse à 0. On comprend bien que cette comparaison permet d'obtenir la largeur d'impulsion que l'on veut. Notez que ce travail de comparaison n'est pas à la charge du programmeur : c'est une fonctionnalité du microcontrôleur.

En matière de code, il faut :

- Créer un timer adapté à ce que l'on veut faire, en paramétrant prescaler et compteur pour disposer de la fréquence que l'on veut. Pour avoir des rapports cycliques les plus précis possible, il faut que dans ce calcul, le compteur soit le plus grand possible, et donc réduire la valeur du prescaler.
- Paramétrier un comparateur, appelé "comparateur de Sortie" ("OC" pour "Output Comparator"), qui va permettre de définir la valeur de "pulse"
- Paramétrier un GPIO capable de générer un signal PWM en mode "Alternate Function"
- Faire la liaison entre le GPIO et le Output Comparator

Cela est documenté comme une option du bloc "Timer" et une option du bloc "GPIO". Voici un code simple qui permet de moduler la luminosité d'une LED en utilisant le signal PWM. On utilise le OC2 du TIM4 et la pin PB7 :

```

1 #include "stm32l1xx.h"
2
3 void TIM4_PWM_Config();
4 void TIM4_PWM_Set(uint16_t pulseWidth);
5
6
7 int main(void)
8 {
9     TIM4_PWM_Config();
10
11     float duty = 0;
12     float step;
13     while(1)
14     {
15         duty *= step;
16         if (duty > 32000) {
17             duty = 32000;
18             step = 0.99;
19         }
20         if (duty < 0.001*32000) {
21             duty = 0.001*32000;
22             step = 1.01;
23         }
24
25         TIM4_PWM_Set(duty);
26
27         int k;
28         for(k=0; k<3000; k++) {
29         }
30     }
31 }
32
33 // ## PWM via TIM4 sur PB7
34 // Configuration
35 void TIM4_PWM_Config()
36 {
37     /*Activer TIM4 sur APB1 */

```

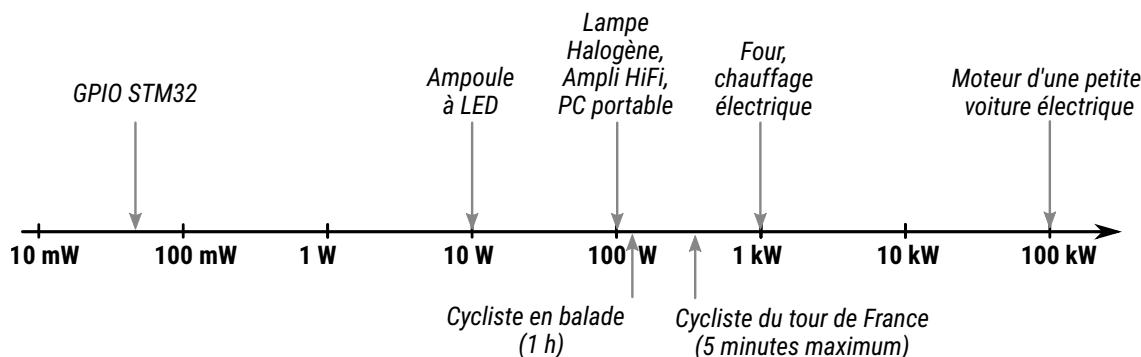
```

38     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4 ,ENABLE);
39     /* Configurer TIM4 a 20 ms */
40     TIM_TimeBaseInitTypeDef timer_4;
41     TIM_TimeBaseStructInit(&timer_4);
42     timer_4.TIM_Period = 32000;
43     timer_4.TIM_Prescaler = 1;
44     TIM_TimeBaseInit(TIM4 ,&timer_4);
45     TIM_Cmd(TIM4, ENABLE);
46
47     /* Configurer le comparateur de sortie OC2 de TIM4 */
48     /* pour faire du PWM */
49     TIM_OCInitTypeDef timer_4_oc_2;
50     TIM_OCStructInit(&timer_4_oc_2);
51     timer_4_oc_2.TIM_OCMode =           TIM_OCMode_PWM1;
52     timer_4_oc_2.TIM_Pulse = 1000;
53     timer_4_oc_2.TIM_OutputState = TIM_OutputState_Enable;
54     TIM_OC2Init(TIM4,&timer_4_oc_2);
55
56     /*Activer GPIOB sur AHB */
57     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB ,ENABLE);
58     /* Configurer PB7 comme "Alternative Function" pour preparer son
       utilisation en PWM */
59     GPIO_InitTypeDef gpio_b;
60     GPIO_StructInit(&gpio_b);
61     gpio_b.GPIO_Mode = GPIO_Mode_AF;
62     gpio_b.GPIO_Pin = GPIO_Pin_7;
63     gpio_b.GPIO_Speed = GPIO_Speed_10MHz;
64     GPIO_Init(GPIOB, &gpio_b);
65
66     /* relier PB7 a TIM4/OC2 */
67     GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_TIM4);
68 }
69
70 // Changer rapport cyclique
71 void TIM4_PWM_Set(uint16_t pulseWidth)
72 {
73     TIM_SetCompare2(TIM4 ,pulseWidth);
74 }
```

 [code/code-VI-3-11.c]

## VI.4 Comparaison DAC et PWM

Le DAC reste la solution la plus élégante, mais aussi la plus chère. C'est déjà vrai à l'intérieur du microcontrôleur : la gestion d'un signal PWM nécessite beaucoup moins d'électronique qu'un DAC. Mais c'est tout aussi vrai hors du microcontrôleur. En effet, il est bien clair qu'il faut souvent amplifier le signal qui sort d'une pin PWM, puisqu'il s'agit de piloter des éléments qui produisent des actions physiques. Le microcontrôleur en effet peut "sortir"  $3V \times 20mA$  environ, soit  $60mW$ . Comparez par exemple à la puissance du moindre four, radiateur, moteur, amplificateur audio, ou même simplement lampe de plafond, on est en Watt, en dizaines de Watt et même en kWatt !



Or, amplifier un signal "vraiment analogique", tel qu'il est fourni par un ADC, nécessite, pour le faire bien, que l'amplificateur remplisse une amplification sans déformation, et ça c'est très difficile en électronique : *c'est la raison pour laquelle les amplificateurs opérationnels existent!* et *c'est la raison pour laquelle ils contiennent des dizaines de transistors et utilisent des contre-réactions, pour être le plus linéaire possible.*

Avec un signal PWM, l'amplificateur n'a que 2 niveaux précis à gérer, et c'est donc bien plus facile, il suffit dans la plupart des applications de faire fonctionner *un seul transistor* en régime tout-ou rien. La mise en oeuvre est montrée en annexe E.

Malgré tout on ne peut pas toujours utiliser du PWM, comme on va le voir plus loin

## ❖ En résumé ...

Nous avons vu comment faire l'acquisition et la génération de signaux analogiques, ce qui est extrêmement utile.

- Pour l'acquisition il y a essentiellement un type de sous-ensemble, le "ADC" pour "Analog-To-Digital" (ou encore "CAN" pour "Convertisseur Analogique Numérique" en français)
- Pour la génération de signaux analogiques on a deux possibilités : une qui génère un signal analogique "normal", et qui donc suppose une chaîne d'instruments en régime linéaire à la suite. Une autre qui génère un signal formé d'impulsions dont le rapport cyclique change dans le temps, et que l'on appelle "PWM". Cette solution fonctionne bien si le système que l'on pilote est lent par rapport à la période des impulsions. C'est une solution très populaire et préférable quand elle est acceptable pour l'application visée, parce qu'elle a l'avantage d'être plus simple sur un plan de l'électronique à l'intérieur du microcontrôleur, mais aussi à l'extérieur car on peut travailler en tout-ou-rien.

# Chapitre VII – Sous-ensemble 4 : Interruptions Externes et Périodiques

## ⌚ Motivations et objectifs

Jusqu'ici nous avons toujours utilisé du code qui fonctionnait dans le programme principal, au sein d'une boucle. Cette approche est limitée pour plusieurs raisons :

- Elle peut conduire à "rater" des modifications qui apparaîtraient sur certaines entrées du microcontrôleur (on va voir pourquoi)
- Elle n'est pas pratique si le microcontrôleur doit faire plusieurs petites tâches en parallèle

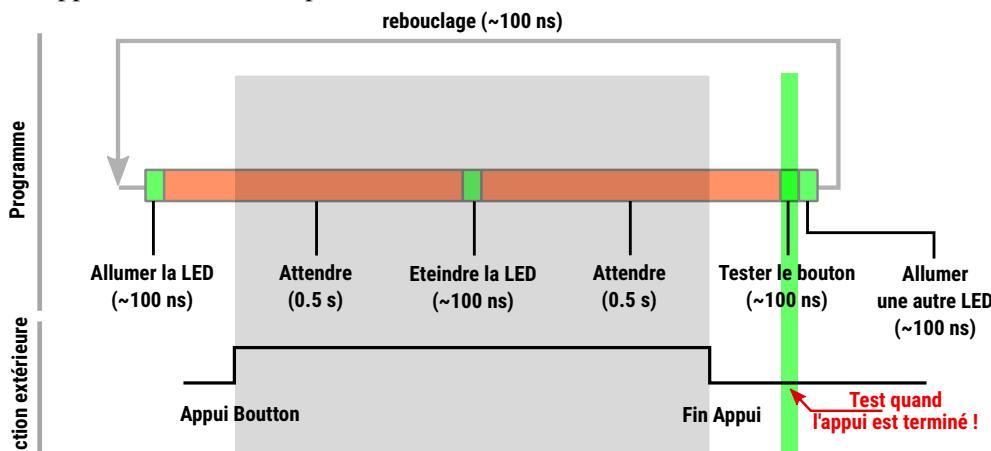
La solution à ces problèmes exploite le concept "d'interruption", une notion qui n'est pas spécifique aux microcontrôleurs mais que l'on retrouve aussi sur les microprocesseurs de PC aussi. C'est un sous-ensemble fondamental pour nous, même si ça peut sembler un sujet un peu difficile, parce que cela conduit à programmer dans un état d'esprit un peu différent.

## VII.1 Le problème à résoudre

Jusqu'ici, nos programmes comportaient deux parties :

- Une partie "initialisation" des blocks que l'on veut utiliser,
- Une partie "utilisation" qui contient essentiellement une boucle principale qui exécute de façon routinière la tâche que l'on demande au microcontrôleur.

Mais dans beaucoup de situations ça n'est pas satisfaisant. Voici un exemple très simple. Imaginons que l'on fasse clignoter une LED dans un programme principal, et que l'on veuille, en même temps, réagir au fait que l'on appuie sur un bouton poussoir :



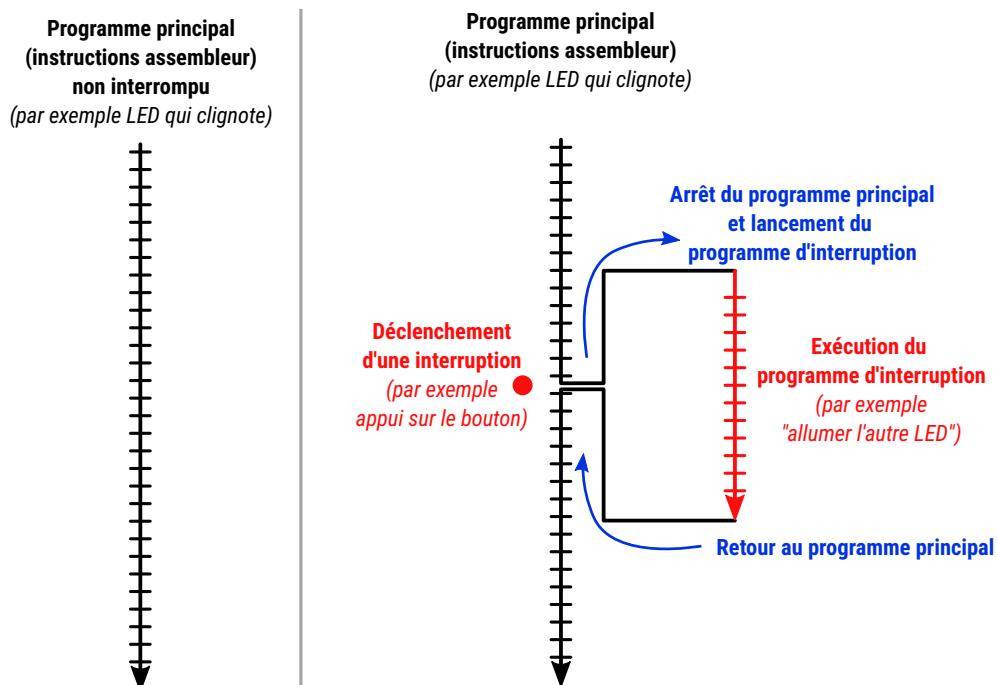
Dans la mesure où "on n'a pas eu la chance" que le bouton soit actif au moment où le test se réalise, on a loupé l'action de l'utilisateur. Si le clignottement de la DEL est rapide ça a des chances de marcher parce qu'on teste le bouton plus souvent, si le clignottement est lent c'est juste raté.

En résumé, si la boucle principale est "très occupée", même par quelque chose de simple, on a des difficultés à détecter des actions extérieures. Alors ? Il faut 2 microcontrôleurs pour faire ça ? Evidemment

non ! Sur un PC avec un système d'exploitation, on pourrait utiliser 2 processus ou 2 programmes qui fonctionnent "en même temps", mais là on n'a pas de système d'exploitation.

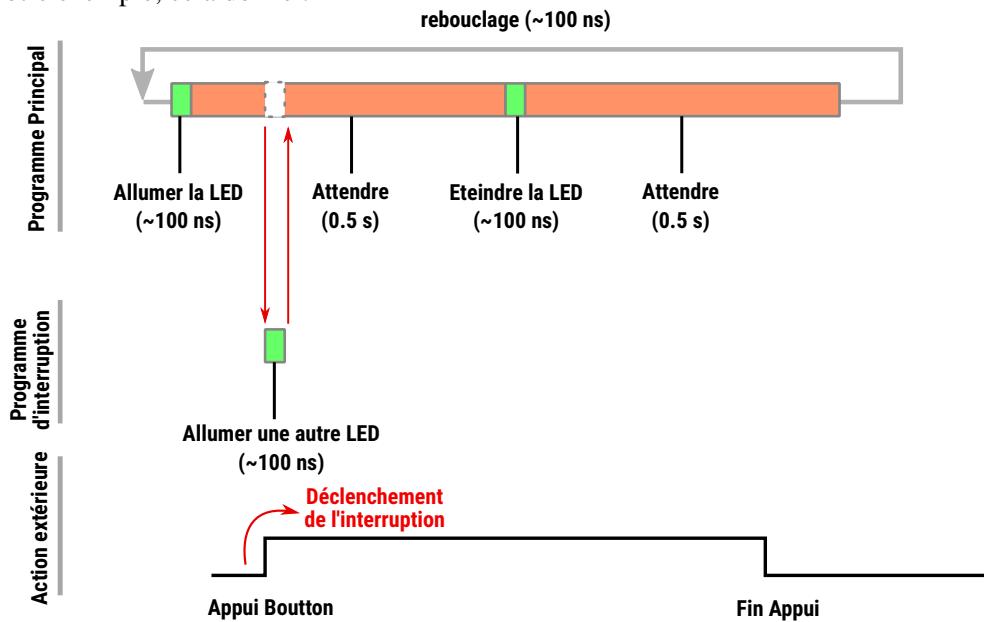
## VII.2 La bonne solution avec un microcontrôleur: "Interruption externe"

Pour résoudre ce genre de problème, les microcontrôleurs mettent à notre disposition le concept "d'interruption". Une interruption porte exactement un nom lié à sa fonction : c'est une façon "d'interrompre" le programme principal. Voici comment cela se déroule :



De cette façon, on peut demander à un microcontrôleur de réaliser, en "parallèle", un ensemble de tâches simples, qui vont pouvoir s'interrompre.

Sur notre exemple, cela donne :



### VII.3 Réalisation sur le STM32L152

On va réaliser l'allumage d'une LED par appui sur un switch en utilisant cette fois l'interruption : En matière de code, c'est un peu plus compliqué que ce qu'on a fait jusqu'ici. Il faut :

- Il faut prendre en compte les contraintes du microcontrôleur : le STM32L152 permet 7 interruptions externes et pas sur n'importe quelle pin. On le justifiera plus tard mais nous avons choisi ici de travailler avec PA0.
- Une fois ce choix fait, il faut évidemment configurer le GPIO du bouton et de la LED comme on le fait habituellement, sans modification,
- Ensuite, le fait d'avoir choisi PA0 fait que nous travaillons avec l'interruption externe EXTI0\_IRQn, qui exécute, lorsqu'elle est déclenchée, le code contenu dans la fonction EXTI0\_IRQHandler(). Cette fonction est symbolisée comme "ligne 0" dans la bibliothèque (c'est comme ça). Il nous faudra configurer, pour cette "ligne 0", sur quel critère l'interruption est exécutée : front montant, front descendant, etc.
- Ensuite, il faudra aussi l'activer au niveau du gestionnaire de "vecteurs d'interruption", appelé NVIC. Les "vecteurs d'interruptions" sont une notion qui intervient au niveau du mapping, il s'agit simplement du lieu où sont stockées les adresses des fonctions d'interruption. Mais ce travail est déjà fait par la bibliothèque et nous n'avons pas à nous en soucier. Et il est donc normal que cette déclaration auprès du NVIC semble artificielle au niveau de la bibliothèque, mais elle est nécessaire.
- Et pour finir, la raison pour laquelle on fait tout ça : remplir la fonction d'interruption, c'est à dire EXTI0\_IRQHandler(), pour qu'elle fasse ce que l'on souhaite, ici allumer une LED quand l'interrupteur est activé !

Commençons par justifier le choix de PA0. D'abord, l'interrupteur cablé sur la carte est le PA0. Ensuite, comme le montre le tableau ci-dessous, la gestion des interruptions est moins ambiguë (plus simple) pour les interruptions entre 0 et 4, et donc pour les pin dont le numéro est entre 0 et 4, peu importe leur port :

Nom de l'interruption	Fonction C associée	"Lignes" associées	exemples de pins associées
EXTI0_IRQn	EXTI0_IRQHandler()	Ligne 0	PA0, PB0, PC0, ...
EXTI1_IRQn	EXTI1_IRQHandler()	Ligne 1	PA1, PB1, PC1, ...
EXTI2_IRQn	EXTI2_IRQHandler()	Ligne 2	PA2, PB2, PC2, ...
EXTI3_IRQn	EXTI3_IRQHandler()	Ligne 3	PA3, PB3, PC3, ...
EXTI4_IRQn	EXTI4_IRQHandler()	Ligne 4	PA4, PB4, PC4, ...
EXTI9_5_IRQn	EXTI9_5_IRQHandler()	Lignes 5 à 9	PA5, PB5, PA8, ...
EXTI15_10_IRQn	EXTI15_10_IRQHandler()	Lignes 10 à 15	PA10, PB12, PC15, ...

#### Remarque

Vous voyez un peu partout la terminologie "IRQ" apparaître. IRQ est un acronyme anglophone qui signifie "Interrupt ReQuest" pour "demande d'interruption". C'est le terme technique habituel pour parler d'interruptions.

Et donc voici le code qui correspond :

```

1 #include "stm32l1xx.h"
2
3 void IRQ_EXTI0_Config();
4
5 int main(void)
6 {
7     // # LED sur PB7
8     /* Activer GPIOB sur AHB */
9     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
10    /* Configurer PB7 comme sortie tout-ou-rien */

```

```
11     GPIO_InitTypeDef gpio_b;
12     GPIO_StructInit(&gpio_b);
13     gpio_b.GPIO_Mode = GPIO_Mode_OUT;
14     gpio_b.GPIO_Pin = GPIO_Pin_7;
15     GPIO_Init(GPIOB,&gpio_b);
16     /* allumer la LED*/
17     GPIO_SetBits(GPIOB,GPIO_Pin_7);
18
19     // configuration de l'interruption
20     IRQ_EXTI0_Config();
21
22     while(1) {
23
24     }
25 }
26
27 // callback pour l'interruption externe EXTI0_IRQ
28 void EXTI0_IRQHandler(void)
29 {
30     if(EXTI_GetITStatus(EXTI_Line0) != RESET)
31     {
32         /* Clear the EXTI line 0 pending bit */
33         EXTI_ClearITPendingBit(EXTI_Line0);
34         GPIO_ToggleBits(GPIOB,GPIO_Pin_7);
35     }
36 }
37
38 // ### EXTI0 sur PA0
39 // Configuration
40 void IRQ_EXTI0_Config()
41 {
42     // # Interrupteur
43     /* Activer GPIOA sur AHB */
44     RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE);
45     /* Configurer PB7 comme entree tout-ou-rien */
46     GPIO_InitTypeDef gpio_a;
47     GPIO_StructInit(&gpio_a);
48     gpio_a.GPIO_Mode = GPIO_Mode_IN;
49     gpio_a.GPIO_Pin = GPIO_Pin_0;
50     GPIO_Init(GPIOB,&gpio_a);
51
52     /* Activer SYSCFG sur APB2
53      * pour permettre l'utilisation des interruptions externes */
54     RCC_APB2PeriphClockCmd (RCC_APB2Periph_SYSCFG,ENABLE);
55     /* Declarer PA0 comme source d'interruption */
56     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,EXTI_PinSource0);
57     /* Param. des signaux qui declencheront l'appel de EXTI0_IRQHandler()
58      * autrement dit on parametre les signaux associes a la "ligne 0"
59      * ici : sur front montant ("Trigger_Rising")
60      */
61     EXTI_InitTypeDef EXTI0_params;
62     EXTI_StructInit(&EXTI0_params);
63     EXTI0_params.EXTI_Line = EXTI_Line0;
64     EXTI0_params.EXTI_LineCmd = ENABLE;
65     EXTI0_params.EXTI_Trigger = EXTI_Trigger_Rising;
66     EXTI_Init(&EXTI0_params);
67
68     /* Activer l'interruption dans le NVIC */
```

```

69     NVIC_InitTypeDef nvic;
70     NVIC_Init(&nvic);
71     nvic.NVIC IRQChannel = EXTI0 IRQn;
72     nvic.NVIC IRQChannelCmd = ENABLE;
73     NVIC_Init(&nvic);
74 }
```

 [code/code-VII-3-12.c]

Soyons honnêtes, contrairement à ce que nous avons écrit jusqu'ici, il est difficile de générer cette page de code en se basant seulement sur la documentation. Ce code écrit ici s'inspire de codes d'exemples fournis avec la documentation.

Dans ce code, comme on le voit, le programme principal ne fait rien une fois les pins configurées : il fait juste un **while(1)** vide. Et donc tout se fait au niveau de l'interruption, dans laquelle, à part quelques précautions qui seront rediscutées en TP, le code ne fait que basculer l'état de PB7.

## VII.4 Interruptions périodiques

Comme on l'a fait remarquer, on peut créer des interruptions périodiques. Et c'est très classique en électronique. On va commencer par utiliser une interruption périodique pour simplement faire clignoter une LED, mais on verra qu'on peut faire beaucoup plus.

Pour cela il faut faire exactement ce que l'on a fait pour faire le "délai" dans un code précédent, c'est à dire paramétrer un timer (ici on va prendre le Timer 2 mais on aurait pu prendre n'importe lequel), et lui associer une interruption. Voici le code :

```

1 #include "stm32l1xx.h"
2
3 void TIM2_IRQHandler();
4
5 int main(void)
6 {
7     TIM2_IRQHandler();
8
9     // # LED sur PB7
10    /* Activer GPIOB sur AHB */
11    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB,ENABLE);
12    /* Configurer PB7 comme sortie tout-ou-rien */
13    GPIO_InitTypeDef gpio_b;
14    GPIO_StructInit(&gpio_b);
15    gpio_b.GPIO_Mode = GPIO_Mode_OUT;
16    gpio_b.GPIO_Pin = GPIO_Pin_7;
17    GPIO_Init(GPIOB,&gpio_b);
18
19    while(1) {
20
21    }
22}
23
24 // callback pour l'interruption périodique associée à TIM2
25 void TIM2_IRQHandler() {
26     if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
27     {
28         TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
29         GPIO_ToggleBits(GPIOB, GPIO_Pin_7);
30     }
31}
32
33 // ### TIMER 2 + IRQ à 500 ms
34 // Configuration Timer 2 à 500 ms
```

```

35 // avec émission d'IRQ : execute périodiquement TIM2_IRQHandler()
36 void TIM2_IRQHandler()
37 {
38     /*Activer TIM2 sur APB1 */
39     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);
40     /* Configurer TIM2 à 500 ms */
41     TIM_TimeBaseInitTypeDef timer_2;
42     TIM_TimeBaseStructInit(&timer_2);
43     timer_2.TIM_Prescaler = 16000-1;
44     timer_2.TIM_Period = 500;
45     TIM_TimeBaseInit(TIM2,&timer_2);
46     TIM_SetCounter(TIM2,0);
47     TIM_Cmd(TIM2, ENABLE);
48
49     /* Associer une interruption à TIM2 */
50     TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
51
52     NVIC_InitTypeDef nvic;
53     /* Configuration de l'interruption */
54     nvic.NVIC_IRQChannel = TIM2_IRQn;
55     nvic.NVIC_IRQChannelPreemptionPriority = 0;
56     nvic.NVIC_IRQChannelSubPriority = 1;
57     nvic.NVIC_IRQChannelCmd = ENABLE;
58     NVIC_Init(&nvic);
59 }
```

 [code/code-VII-4-13.c]

## VII.5 Les interruptions sont partout!

On n'en n'a pas parlé depuis le début de ce cours, mais on peut associer des interruptions à presque tout ce que l'on fait dans un microcontrôleur.

Par exemple, avec l'ADC, on avait écrit une boucle qui attendait la fin de la conversion. Au lieu de faire cette attente par une boucle, on pourrait attribuer une interruption à la fin de la conversion pour n'agir qu'à ce moment là. Sur des acquisitions lentes ça pourrait avoir du sens. Autre exemple que l'on utilise beaucoup et que l'on vient de voir : les timers. On peut associer une interruption à un timer, l'interruption sera déclenchée à chaque période par exemple. Même chose avec du PWM, forcément, etc.

C'est une façon de pouvoir gérer tout un tas de choses en parallèle, en évitant d'avoir à faire des attentes avec des boucles. Ces deux façons de programmer portent un nom :

- Quand on fait des boucles pour attendre que quelque chose change, on parle de "polling"
- Si on utilise des interruptions on parle de programmation par interruption

### Remarque

Les interruptions ne sont pas une caractéristique spécifique des microcontrôleurs. C'est une notion qui provient des CPU au départ. Sur un PC, le couplage entre un timer et les interruptions est ce qui permet de mettre en œuvre le multitâche : à chaque interruption périodique, le processus en cours d'exécution est interrompu, pour donner place à un programme interne au système, appelé "ordonnanceur", qui va avoir pour rôle de choisir le prochain processus auquel il faudra consacrer du temps. Puis ce processus sera à son tour interrompu par l'interruption, et ainsi de suite.

### En résumé ...

C'est un autre sous-ensemble fondamental des microcontrôleurs que nous avons regardé ici. Il permet notamment de réaliser un ensemble de petites "tâches" en parallèle et parfaitement synchronisées avec l'événement qui les concerne, qu'il soit à l'extérieur du microcontrôleur ou déclenché à l'intérieur du microcontrôleur lui-même.

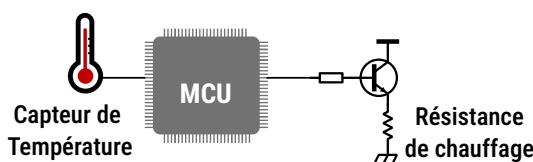
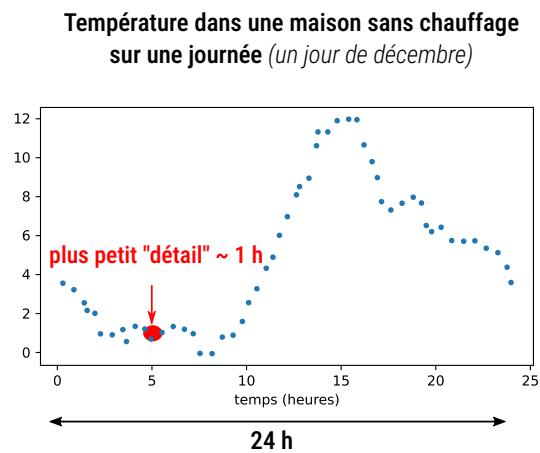
# Chapitre VIII – Chaine complète : ADC ▶ traitement ▶ DAC/PWM

## ② Motivations et objectifs

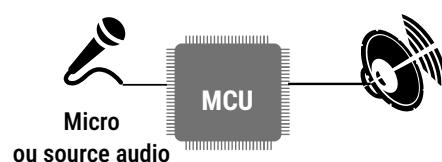
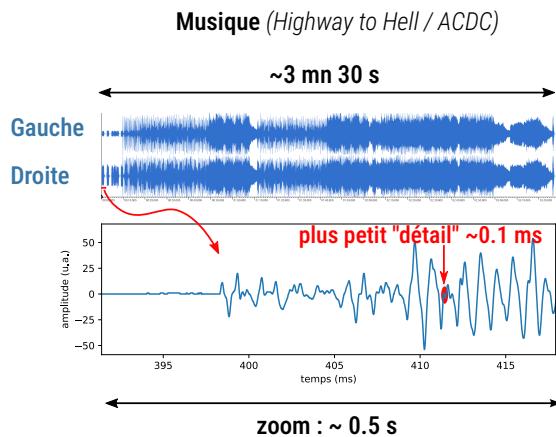
On fait ici une synthèse de l'utilisation des sous-ensembles que nous avons vu précédemment (TIM, IRQ, ADC, DAC, PWM) pour créer une chaîne complète d'acquisition et génération de signaux, autrement dit une plateforme pour le traitement du signal, en temps réel.

### VIII.1 Contexte

Jusqu'ici on a surtout parlé des signaux continus. Mais on a souvent besoin de traiter des signaux qui varient en fonction du temps. Prenons deux exemples.



Température plus élevée et constante  
= "régulation de température" ou "Thermostat"



Effets sur le son : equalizer, echo,  
compresseur de dynamique, ...

### Exemple de la température d'une maison

D'abord, regardons l'évolution de la température dans une maison sans chauffage l'hiver sur une durée de 24 h. Si on fait un relevé, on se rend compte que la température évolue au global assez lentement, et c'est normal parce que le volume d'une maison est quelque chose d'important, et l'ensemble a donc une grande inertie. Sans que ce soit extrêmement "fiable", on peut voir que les variations temporelles les plus rapides, que l'on nomme ici "plus petit détail", est de l'ordre de 1h. Pour le dire précisément, il faudrait faire l'analyse de Fourier de ce signal et regarder jusqu'à quelle fréquence vont les harmoniques, mais pour ce que l'on fait là on peut se contenter de ça.

Ainsi, si on voulait regarder l'évolution de la température, il serait raisonnable de se dire que l'on va prendre des points "un peu plus souvent" que une fois par heure, histoire d'être sûr de "ne rien rater".

Si maintenant on prend un microcontrôleur pour piloter une résistance chauffante qui va chauffer la maison, on peut écrire un programme pour le microcontrôleur qui mesure la température de la maison et sait dire "de combien" il faut modifier le courant dans la résistance pour chauffer la maison pour atteindre "assez vite" la température demandée. On appelle cela une "régulation de température". Les microcontrôleurs ne sont pas les seuls à pouvoir faire cela, on peut aussi le faire en analogique, c'est le sujet d'un autre cours.



### Important

Hormis le contexte, l'information qui nous intéresse ici est qu'il est inutile, avec le microcontrôleur, de mesurer la température toutes les millisecondes par exemple, d'abord parce que la température de la maison ne change pas aussi vite que ça spontanément, et ensuite parce que la résistance chauffante ne peut pas produire un effet aussi rapide sur la maison. ■

### Exemple du son

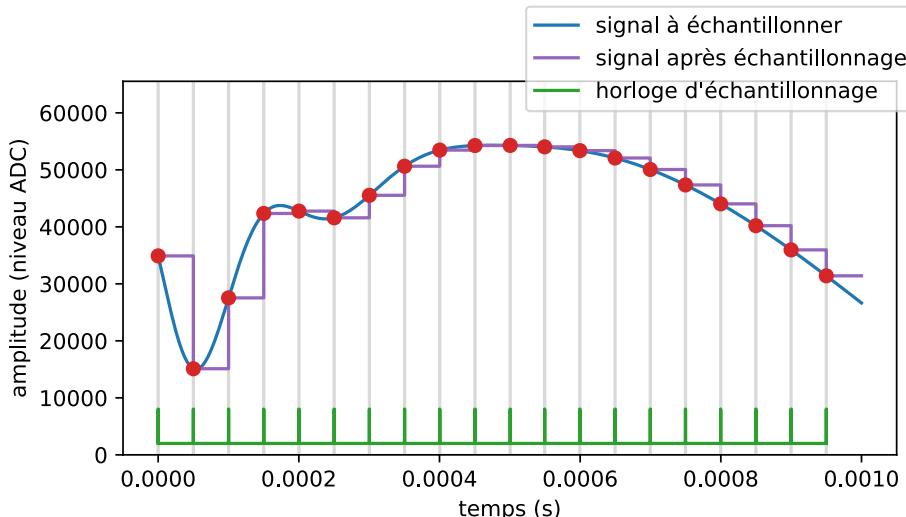
Regardons maintenant quelque chose de plus exigeant : le son, la musique. On a représenté en haut la représentation des deux pistes (oreille gauche et oreille droite) de son qui correspondent à "Highway To Hell" de AC/DC. Ces courbes, dont la durée complète est environ 3 mn 30 s, représentent les variations de tension analogique que l'on pourrait percevoir à la sortie d'un microphone d'enregistrement.

Si maintenant on zoomé sur des temps beaucoup plus courts, on se rend compte que les "détails" font de l'ordre de 0,1 ms. En réalité, notre oreille peut entendre des variations de 0,05 ms, mais de telles variations n'apparaissent pas dans le morceau qui est montré ici. Mais l'ordre de grandeur reste correct. Et donc, il est bien clair que si l'on regarde, comme dans le cas de la température de la maison, le signal "toutes les heures" environ, on va perdre énormément d'information ! Ici c'est bien entendu caricatural, mais il s'agit d'illustrer qu'il existe, pour tout type de signal que l'on veut regarder, un temps caractéristique de mesure qui est pertinent. Ce temps est appelé "temps d'échantillonnage".

En récupérant, à la bonne vitesse, l'amplitude du son sur un microcontrôleur, on peut appliquer par exemple "des effets" sur un son : equalizer, echo, compresseur de dynamique, etc. Ces effets sont possibles en analogique bien entendu, mais pour certains ils sont complexes à obtenir, et pas "souples" : si on veut changer un paramètre, il faut changer des composants. Avec un microcontrôleur, il suffit de réinjecter un nouveau code. C'est son grand avantage.

## VIII.2 Signaux échantillonés

On a donc vu que le temps du plus petit détail perceptible est de l'ordre de 1h pour la température de la maison, et de l'ordre de 0.05 ms pour du son. On peut montrer en traitement du signal, que pour "bien faire les choses", il faut mesurer le signal 2 fois plus vite que le temps du détail le plus court. Ainsi, pour la température, ce serait toutes les 30 mn, et pour le son ce serait toutes les 0.025 ms. Comme on va mesurer régulièrement, cette mesure est périodique, et on appelle ces temps des "temps d'échantillonnage" :  $T_{temperature} = 30\text{mn}$ , et  $T_{audio} = 0.025\text{ms}$ . Le signal qui résulte de cette mesure est appelé "signal échantillonné" :



### VIII.3 Gestion avec un microcontrôleur

Comme on a besoin de faire des mesures à intervalle de temps régulier, puisqu'on a associé une constante de temps, on va utiliser un **timer** pour cela. Ce timer donnera lieu à des **interruptions**. À chaque interruption, il faudra faire l'aquisition du signal avec un ADC, réaliser le traitement nécessaire, puis générer le signal en sortie, avec un DAC mais préférentiellement en PWM.

Programme principal

Interruption par Timer



On comprend bien qu'en faisant ça on a une chaîne d'instrumentation complète, qui est un besoin assez classique en électronique.

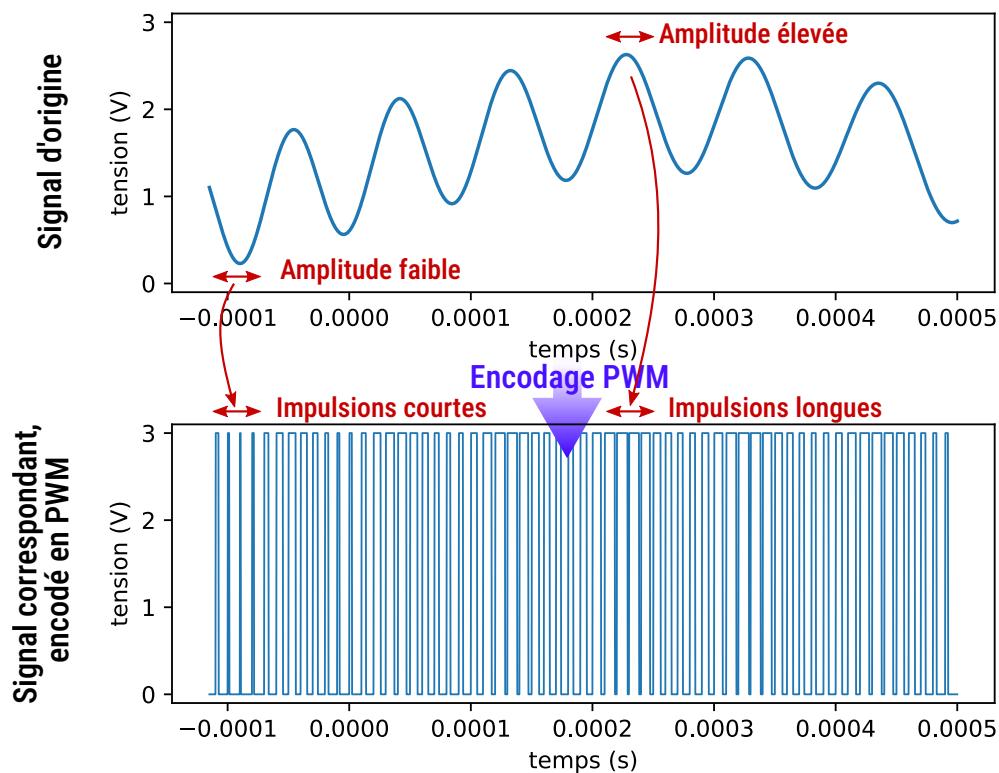
On note aussi qu'il faut absolument que le code que l'on écrit dans l'interruption, tout compris, prenne un temps inférieur à la période d'échantillonnage.

#### Remarque

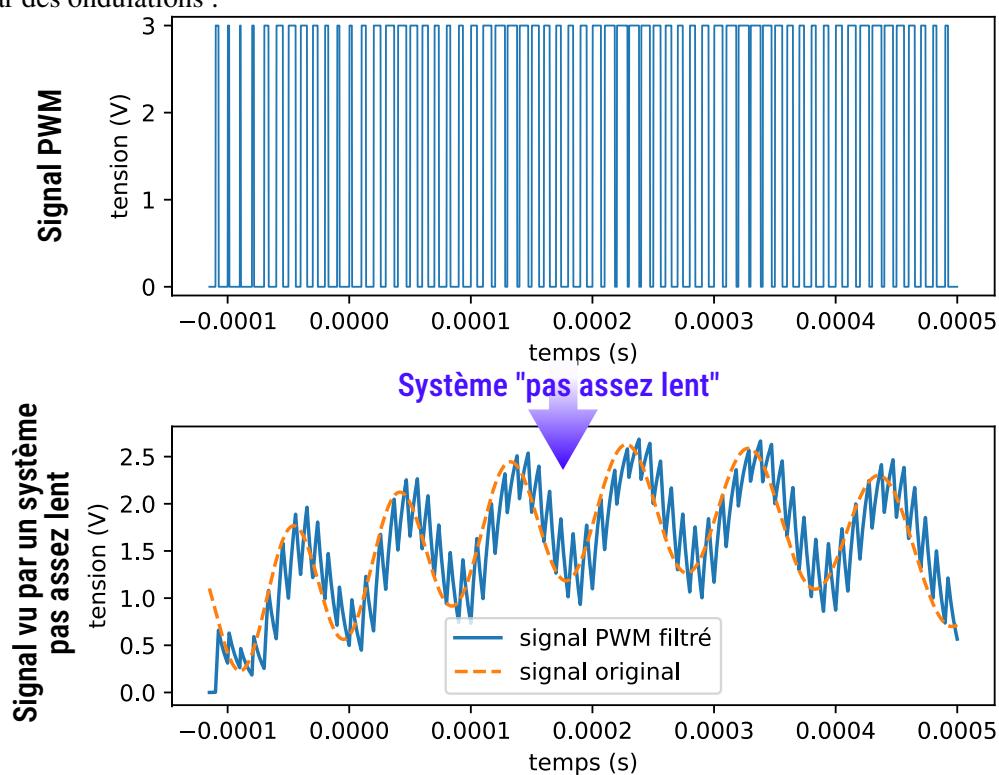
On perçoit bien que, en fonction du temps d'échantillonnage nécessaire pour les signaux et la puissance de calcul du microcontrôleur, les choses peuvent se passer plus ou moins bien. Notez qu'il est difficile, sans expérience, d'avoir une idée concrète de la puissance de calcul nécessaire pour réaliser un traitement donné. Cela dépend autant de votre capacité à programmer "proprement" qu'à la possibilité absolue de le faire. C'est un arbitrage difficile, même pour un professionnel, et cela se base souvent, en pratique, sur une forme d'empirisme : il faut "essayer".

### VIII.4 Codage PWM de signaux variant dans le temps

Jusqu'ici on a surtout traité le codage PWM de signaux continus : on a obtenu un codage PWM avec un rapport cyclique constant. Mais on comprend bien que si on fait varier lentement le rapport cyclique en fonction du temps, ce que l'on appelait jusque là "composante continue" va varier aussi, et donc on n'a plus à faire à un signal continu. Autrement dit : on peut représenter par un codage PWM des signaux variant dans le temps. On représente cela ci-dessous sur un exemple :



On observe très bien que les impulsions sont courtes lorsque l'amplitude est faible, et que les impulsions sont longues lorsque l'amplitude est forte. Maintenant, il faut essayer de comprendre ce qui se passe quand ce signal est vu à travers un système "assez lent". L'idée intuitive est de dire ici qu'un système est "assez lent", au regard du système PWM, si on fait disparaître le plus possible les impulsions, pour retrouver le signal lisse qui est le signal d'origine. Si on ne dispose pas d'un système assez lent (ou, c'est la même chose, si on a choisi une fréquence PWM trop lente), on va retrouver le signal d'origine abîmé par des ondulations :



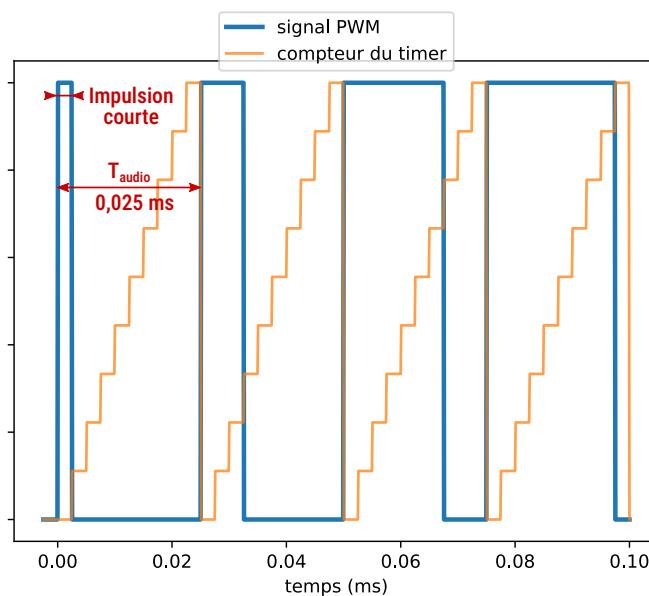
## VIII.5 DAC ou PWM : la suite!

Avec des signaux qui varient dans le temps, le codage en PWM n'est pas toujours possible, ou nécessiterait des ressources complexes. On va traiter les deux exemples que l'on a déjà regardé, tour à tour, pour montrer les limites d'une part, et ce qui se prête bien d'autre part.

### Cas d'un système audio HiFi

Avec un système audio, on a établi qu'il fallait une période d'échantillonnage  $T_{audio} = 0,025ms$ . Si on veut qu'il s'agisse de son HiFi, les limites de l'oreille conduisent au fait qu'il faut 16 bits par amplitude, ce qui fait des valeurs situées entre 0 et  $2^{16} - 1 = 65535$ . Ca, c'est le cahier des charges.

Maintenant, supposons que l'on veuille générer un tel signal avec du PWM. En PWM, si on besoin de coder des valeurs entre 0 et 65535, alors on a besoin d'un compteur dont la valeur maximale vaudra 65535. Et donc, sur cette échelle, on va générer des impulsions dont la largeur seront entre 1 et 65535. Regardons sur un exemple simplifié :



Intéressons nous à l'impulsion la plus courte. Il est bien clair que cette impulsion fait 1/65535<sup>eme</sup> de la période  $T_{audio}$ . Si on calcule, ça donne :

$$T_{pulseCourt} = \frac{0.025 \times 10^{-3}}{65535} \approx 0.38ns$$

Cela signifie qu'il faut une horloge de  $(0.38 \times 10^{-9})^{-1} \approx 2.9GHz$ !! C'est à dire une fréquence très élevée, par rapport au petit 16MHz de la clock de notre microcontrôleur! Donc, juste en utilisant le microcontrôleur et ses capacités, ce n'est pas envisageable, et on ne pourra pas retenir la solution PWM ici, on utilisera le DAC.

### Cas d'un système en température

Compte tenu des variations que l'on a vu, si on décide que l'on a besoin de mesurer la température toutes les minutes, on est déjà très confortable par rapport au besoin. Alors prenons ça,  $T = 60s$ . Ensuite, supposons que l'on mesure la température entre 0 et 50°C : c'est beaucoup plus que les besoins pour faire un chauffage (par exemple on ne va pas chauffer une pièce qui est déjà à 50°C). Mais supposons ça, on est encore large. Supposons aussi que l'on veut une précision sur la température de 0.5°C, c'est déjà très bien. Cela fait que l'on a besoin d'une précision relative de  $0.5/(50 - 0) = 1\%$ . Cette précision nécessite des nombres codés sur  $\log_2(1\%) \approx 7$  bits. "Arrondissons" à 8 bits, ca correspond à ce que peuvent faire beaucoup convertisseurs, ainsi qu'à la quantité de mémoire de base en informatique. Avec 8bits, on code

des valeurs entre 0 et 255, et donc en faisant le même raisonnement que précédemment, on aboutit au fait que l'impulsion la plus courte va durer :

$$T_{pulseCourt} = \frac{60}{256} \approx 0.2\text{s}$$

Ce qui est très facile à atteindre avec le microcontrôleur, comme on la déjà vu sur les Timers. Ainsi, les systèmes liés à la température sont assez lents pour être faciles d'utilisation avec le PWM.

### ❖ En résumé ...

On a vu qu'une chaîne complète de traitement du signal utilise la totalité des sous-ensembles que l'on a vu jusqu'ici. On a comparé, pour le cas de signaux variant dans le temps, les possibilités de faire du PWM et les cas qui nous conduisent à utiliser un DAC. Comme on l'a vu, avant même de parler de qualité de signal, le premier critère qui permet de trancher est de calculer la fréquence nécessaire pour générer l'impulsion la plus courte utile, en prenant en compte le temps d'échantillonnage et la précision que l'on veut, et la comparer à la fréquence de clock du microcontrôleur, sur laquelle se basent tous les Timers. ■

## Chapitre IX – Sous-ensemble 5 : Communication

### ⌚ Motivations et objectifs

RS232, I2C, SPI, USB, si y'a du temps

### ❖ En résumé ...

[Progress bar: A blue bar with a white square at the end, indicating progress or completion.]

## *Annexes*

---



## Annexe A – Documentation de la carte STM32L152C-Dicover

*Extrait du PDF UM1079*

*Pages 72 à 109*



# UM1079

## User manual

### Discovery kits with STM32L152RCT6 and STM32L152RBT6 MCUs

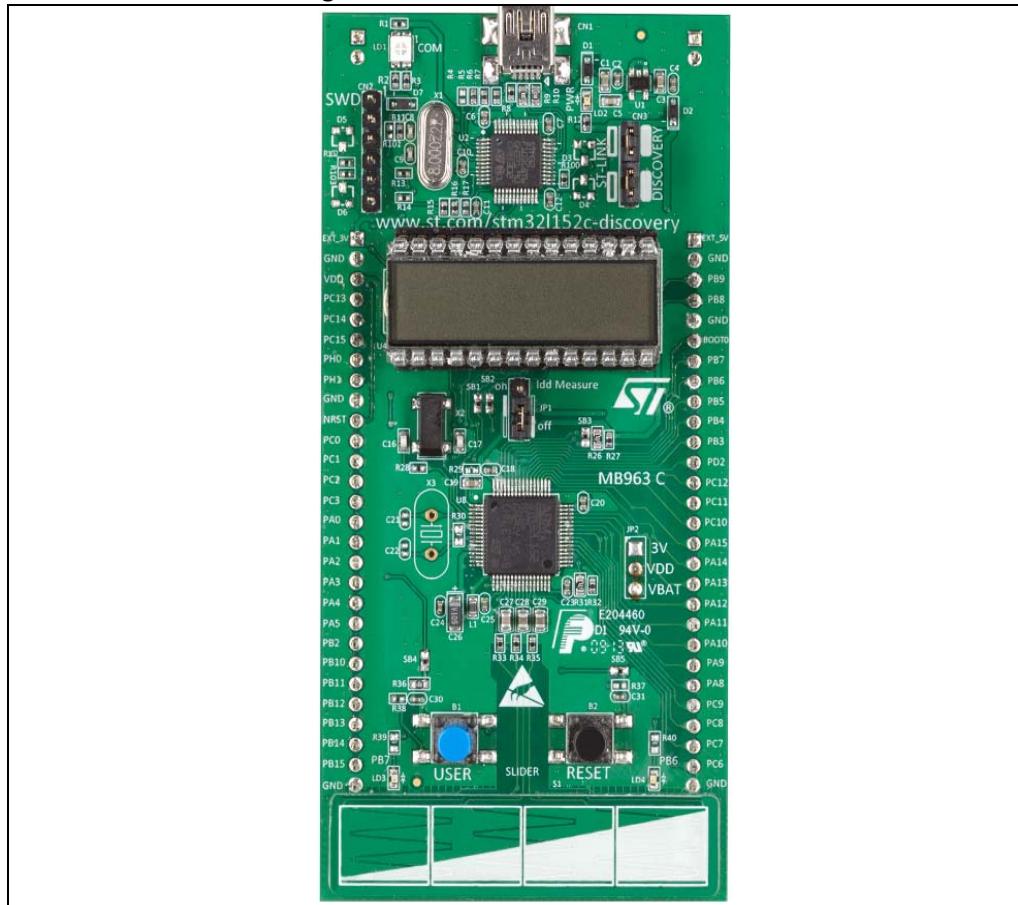
## Introduction

The STM32L152RCT6 Discovery kit (32L152CDISCOVERY) and the STM32L152RBT6 (STM32L-DISCOVERY) allow to develop applications based on the STM32L1 Series and to benefit from the ultra-low-power features of these microcontrollers.

The 32L152CDISCOVERY is based on an STM32L152RCT6 (256 Kbytes of Flash memory). The STM32L-DISCOVERY is based on an STM32L152RBT6 (128 Kbytes of Flash memory).

These discovery kits include the ST-LINK/V2 in-circuit debugger, one LCD (24 segments, 4 commons), four LEDs, two pushbuttons, one linear touch sensor and four touchkeys.

**Figure 1. 32L152CDISCOVERY board**



1. Picture is not contractual.

## **Contents**

<b>1</b>	<b>Ordering information</b>	<b>6</b>
<b>2</b>	<b>Conventions</b>	<b>6</b>
2.1	Quick start	6
2.2	Getting started	6
2.3	System requirements	7
2.4	Development toolchain supporting the 32L152CDISCOVERY	7
2.5	Demonstration software	8
<b>3</b>	<b>Features</b>	<b>8</b>
<b>4</b>	<b>Hardware and layout</b>	<b>9</b>
4.1	STM32L152RCT6 microcontroller	11
4.2	Embedded ST-LINK/V2	14
4.2.1	Using the ST-LINK/V2 to program/debug the microcontroller on board	14
4.2.2	Using the ST-LINK/V2 to program/debug an external application	15
4.3	Power supply and power selection	16
4.4	LEDs	17
4.5	Pushbuttons	17
4.6	Linear touch sensor / touchkeys	17
4.7	Built-in IDD measurement circuit	18
4.7.1	High $I_{DD}$ range mode	19
4.7.2	Low $I_{DD}$ range mode	19
4.7.3	$I_{BIAS}$ current measurement procedure	20
4.8	Solder bridges	21
4.9	LCD (24 segments, 4 commons)	23
<b>5</b>	<b>Extension connectors</b>	<b>25</b>
<b>6</b>	<b>Mechanical drawing</b>	<b>30</b>
<b>7</b>	<b>Electrical schematics</b>	<b>31</b>

<b>UM1079</b>	<b>Contents</b>
<b>8        Revision history .....</b>	<b>37</b>

---

## List of tables

Table 1.	Ordering information . . . . .	6
Table 2.	ON/OFF conventions . . . . .	6
Table 3.	Functions executed when clicking B1 button . . . . .	7
Table 4.	Jumper states . . . . .	14
Table 5.	Debug connector CN2 (SWD) . . . . .	15
Table 6.	Solder bridges. . . . .	21
Table 7.	LCD connections . . . . .	24
Table 8.	MCU pin description versus board function . . . . .	25
Table 9.	Document revision history . . . . .	37

## List of figures

Figure 1.	32L152CDISCOVERY board .....	1
Figure 2.	Hardware block diagram.....	9
Figure 3.	Top layout.....	10
Figure 4.	Bottom layout .....	11
Figure 5.	STM32L152RCT6 package .....	12
Figure 6.	STM32L152RCT6 block diagram .....	13
Figure 7.	Typical configuration.....	14
Figure 8.	32L152CDISCOVERY connections .....	15
Figure 9.	ST-Link connections .....	16
Figure 10.	$I_{DD}$ measurement circuit .....	19
Figure 11.	Low $I_{DD}$ range measurement timing diagram .....	20
Figure 12.	LCD segment mapping .....	23
Figure 13.	Mechanical drawing .....	30
Figure 14.	32L152CDISCOVERY .....	31
Figure 15.	ST-LINK/V2 (SWD only).....	32
Figure 16.	MCU .....	33
Figure 17.	LCD .....	34
Figure 18.	$I_{DD}$ measurement .....	35
Figure 19.	Linear touch sensor/touchkeys.....	36

**Ordering information****UM1079**

## 1 Ordering information

To order the 32L152CDISCOVERY ultra-low-power discovery board, refer to [Table 1](#).

**Table 1. Ordering information**

Part number	Order code	Description
32L152CDISCOVERY	STM32L152C-DISCO	Discovery kit based on STM32L152RCT6
STM32L-DISCOVERY	STM32L-DISCOVERY <sup>(1)</sup>	Discovery kit based on STM32L152RBT6

1. STM32L-DISCOVERY is replaced by STM32L152C-DISCO.

## 2 Conventions

[Table 2](#) provides some definitions used in this user manual.

**Table 2. ON/OFF conventions**

Convention	Definition
Jumper JP1 ON	Jumper placed between pin 2 and 3
Jumper JP1 OFF	Jumper placed between pin 1 and 2
Solder bridge SBx ON	SBx connections closed by solder
Solder bridge SBx OFF	SBx connections left open

The following sections of this user manual are also applicable to the STM32L-DISCOVERY except specific features of the STM32L152RBT6 microcontroller (128 Kbyte Flash memory, 16 Kbyte RAM, 4 Kbyte data EEPROM).

### 2.1 Quick start

Before using the discovery kit, please accept the Evaluation product license agreement available on the 32L152CDISCOVERY page of the [www.st.com/mcu](http://www.st.com/mcu) web site.

### 2.2 Getting started

The following sequence allows to configure the 32L152CDISCOVERY and to launch the discovery application:

- Check jumper positions on the board: JP1 and CN3 must be ON (discovery selected) (see [Figure 3](#)).
- Connect the 32L152CDISCOVERY to a computer with an USB cable to power the board. The red LEDs LD2 (PWR) and LD1 (COM) are lit up. The Function 1 is executed.
- Click on user button B1 to change the executed function as described in [Table 3](#). The 4-LED bar shows the function being performed (1 to 4 bars can be switched ON).

**UM1079****Conventions**

Depending on the function selected, the voltage value, the linear touch sensor position, the touchkeys status or the STM32L152RCT6 current consumption is displayed on the LCD.

**Table 3. Functions executed when clicking B1 button**

Function	LED LD3/4	Bar status	Value displayed on LCD	Main function
1	LD3 and LD4 blink		Measured STM32L152RCT6 VDD voltage	Voltage measurement
2	LD3 ON		Linear touch sensor position from 0 to 100%	Touch sensing
3	LD4 ON		Status of the four touchkeys	
4	LD3 and LD4 OFF		STM32L152RCT6 consumption measured in Run mode (4 MHz)	STM32L152RCT6 current consumption measurement
			STM32L152RCT6 consumption measured in Sleep mode (4 MHz)	
			STM32L152RCT6 consumption measured in Run mode (32 KHz)	
			STM32L152RCT6 consumption measured in Low-power sleep mode (32 KHz)	
			STM32L152RCT6 consumption measured in Stop mode, RTC ON	
			STM32L152RCT6 consumption measured in Stop mode, RTC OFF	
			STM32L152RCT6 consumption measured in Standby mode	

Please refer to the [www.st.com/mcu](http://www.st.com/mcu) web site for more details on the discovery project and the STM32L152RCT6 features.

## 2.3 System requirements

- Windows PC (XP, Vista, 7)
- USB type A to Mini-B USB cable

## 2.4 Development toolchain supporting the 32L152CDISCOVERY

- Altium TASKING® VX-Toolset
- Atollic® TrueSTUDIO®
- IAR™ EWARM
- Keil™ MDK-ARM

## 2.5 Demonstration software

The demonstration software, preloaded in the board Flash memory, uses the built-in  $I_{DD}$  measurement feature to automatically measure and display the MCU consumption on the LCD (in Run and Low-power modes). This software also allows to demonstrate touch sensing functionalities such as linear touch sensor or touchkeys.

The latest version of this demonstration source code and associated documentation can be downloaded from [www.st.com/mcu](http://www.st.com/mcu)

## 3 Features

The 32L152CDISCOVERY offers the following features:

- An STM32L152RCT6 microcontroller (256 Kbyte Flash memory, 32 Kbyte RAM, 8 Kbyte data EEPROM) in a 64-pin LQFP package
- On-board ST-LINK/V2 with selection mode switch to use the kit as a standalone ST-LINK/V2 (with SWD connector for programming and debugging)
- Board power supply: through USB bus or from an external 3.3 or 5 V supply voltage
- External application power supply: 3 V and 5 V
- $I_{DD}$  current measurement
- LCD
  - DIP28 package
  - 24 segments, 4 commons
- Four LEDs:
  - LD1 (red/green) indicating USB communication
  - LD2 (red) indicating that 3.3 V power supply is ON
  - Two user LEDs, LD3 (green) and LD4 (blue)
- Two pushbuttons (user and reset)
- One linear touch sensor and four touchkeys
- An extension header for LQFP64 I/Os for quick connection to prototyping board and easy probing

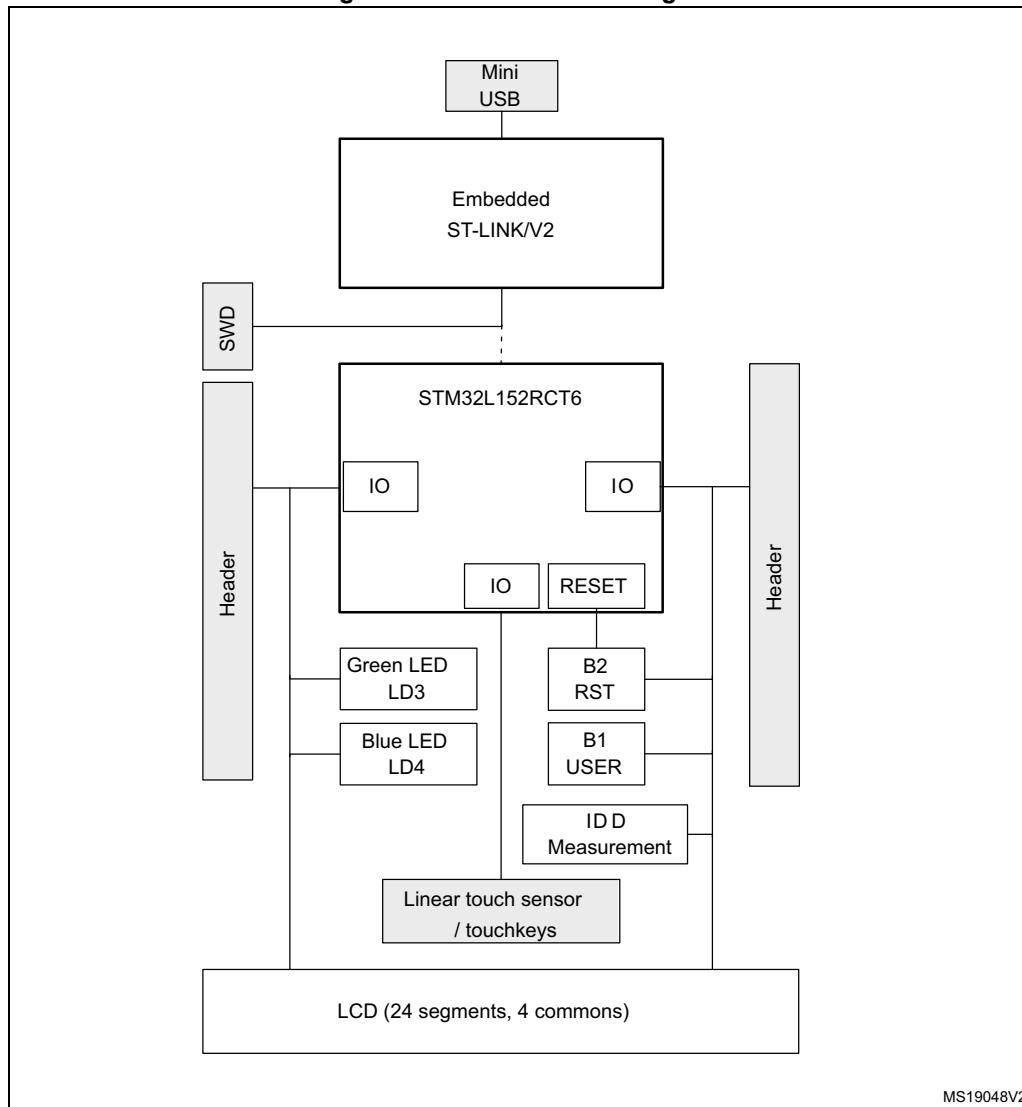
The STM32L-DISCOVERY offers the same features except an STM32L152RBT6 microcontroller (128 Kbyte Flash memory, 16 Kbyte RAM, 4 Kbyte data EEPROM) in a 64-pin LQFP package.

## 4 Hardware and layout

The 32L152CDISCOVERY is designed around one STM32L152RCT6 packaged in an LQFP64.

*Figure 2* illustrates the connections between the STM32L152RCT6 microcontroller and its peripherals (ST-LINK/V2, pushbuttons, LEDs, LCD, linear touch sensor, touchkeys, and connectors). These connections are the same for the STM32L-DISCOVERY.

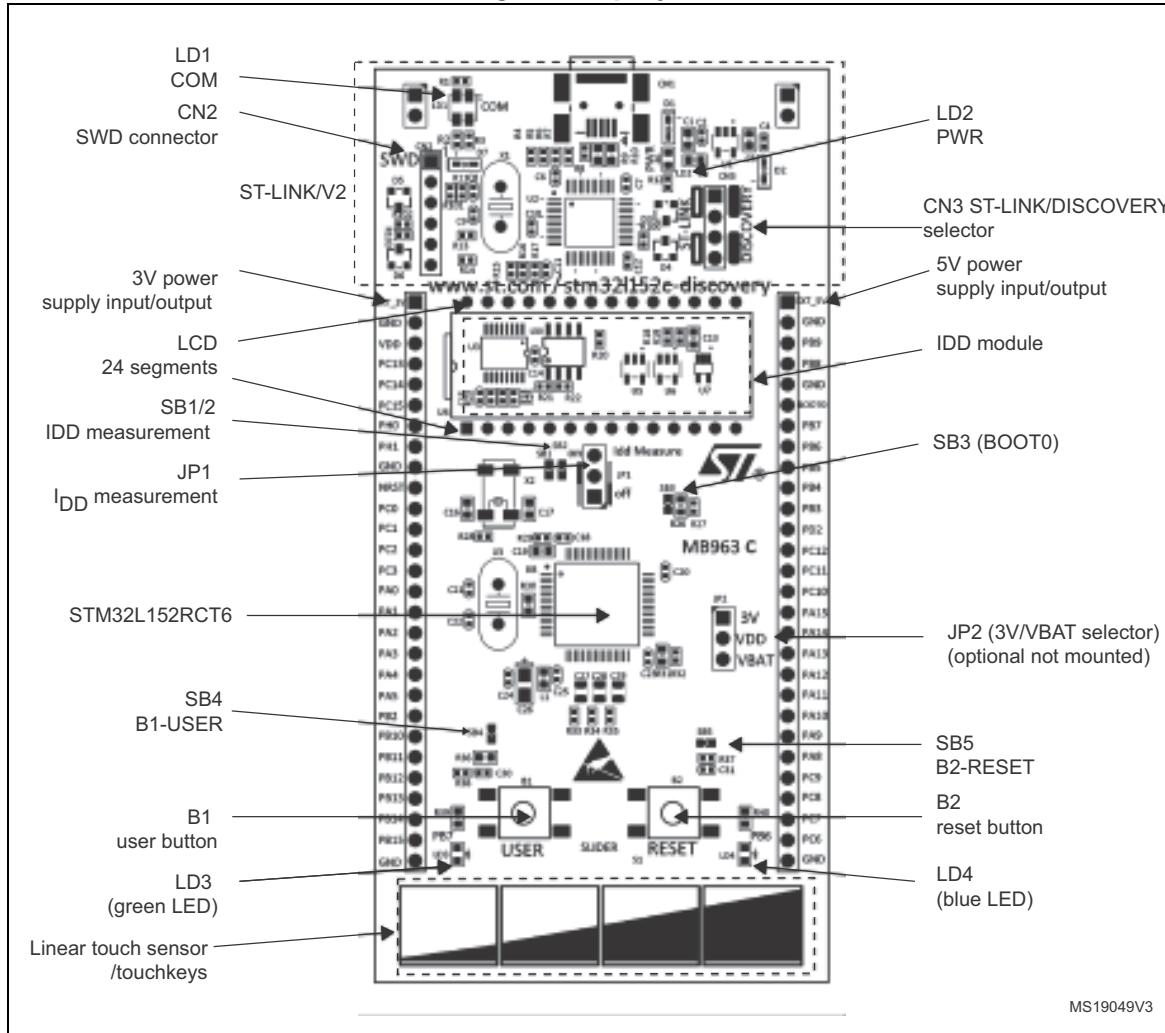
**Figure 2. Hardware block diagram**



*Figure 3* and *Figure 4* allow to locate these features on the board.

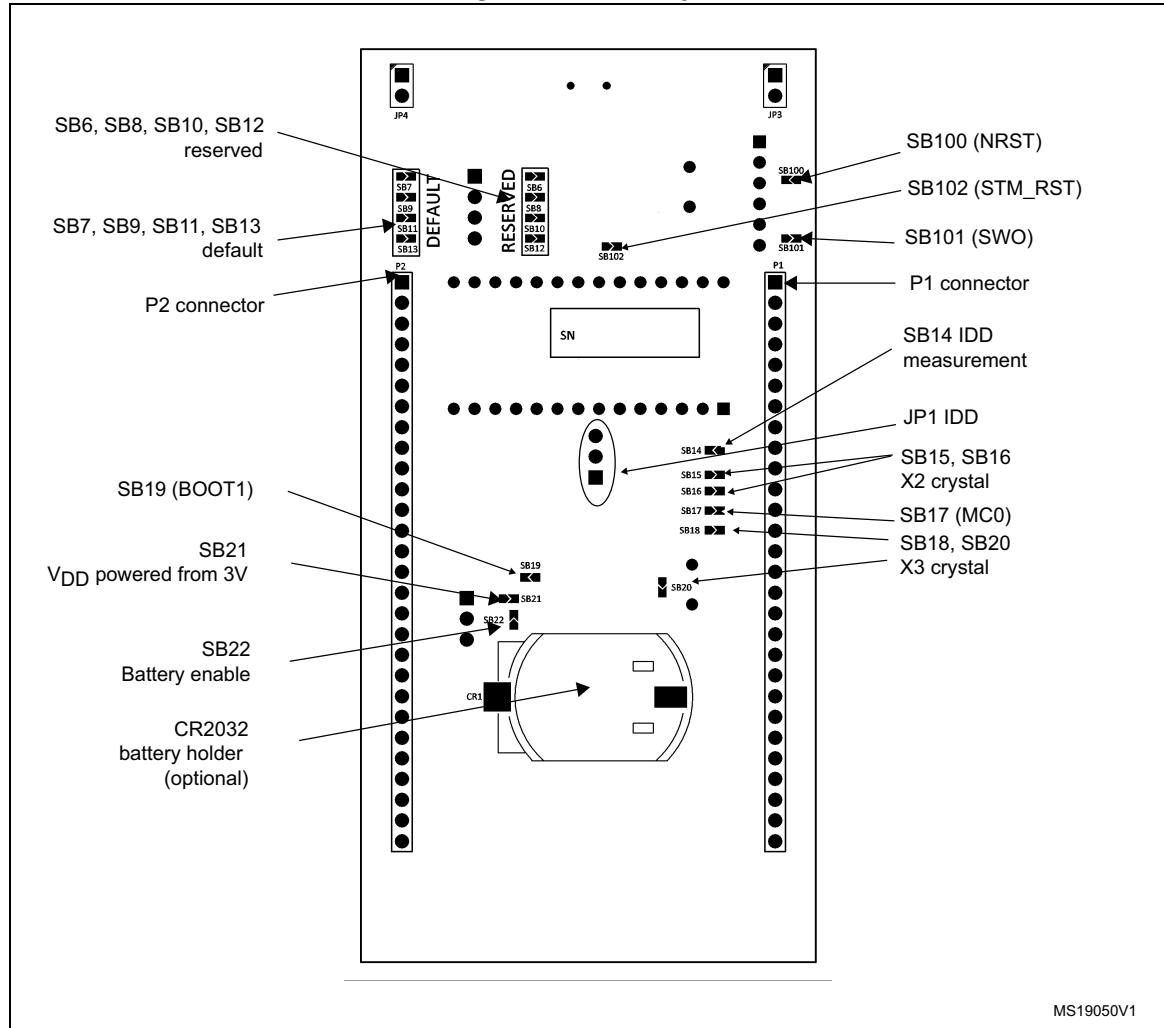
**Hardware and layout**

UM1079

**Figure 3. Top layout**

1. Pin 1 of CN1, CN2, P1 and P2 connectors are identified by a square.

Figure 4. Bottom layout



1. Pin 1 of CN1, CN2, P1 and P2 connectors are identified by a square.

## 4.1 STM32L152RCT6 microcontroller

The STM32L152RCT6 features 256 Kbytes of Flash memory, 32 Kbytes of RAM and 8 Kbytes data of EEPROM.

This microcontroller embeds RTC, LCD, timers, USART, I2C, SPI, ADC, DAC, and comparators.

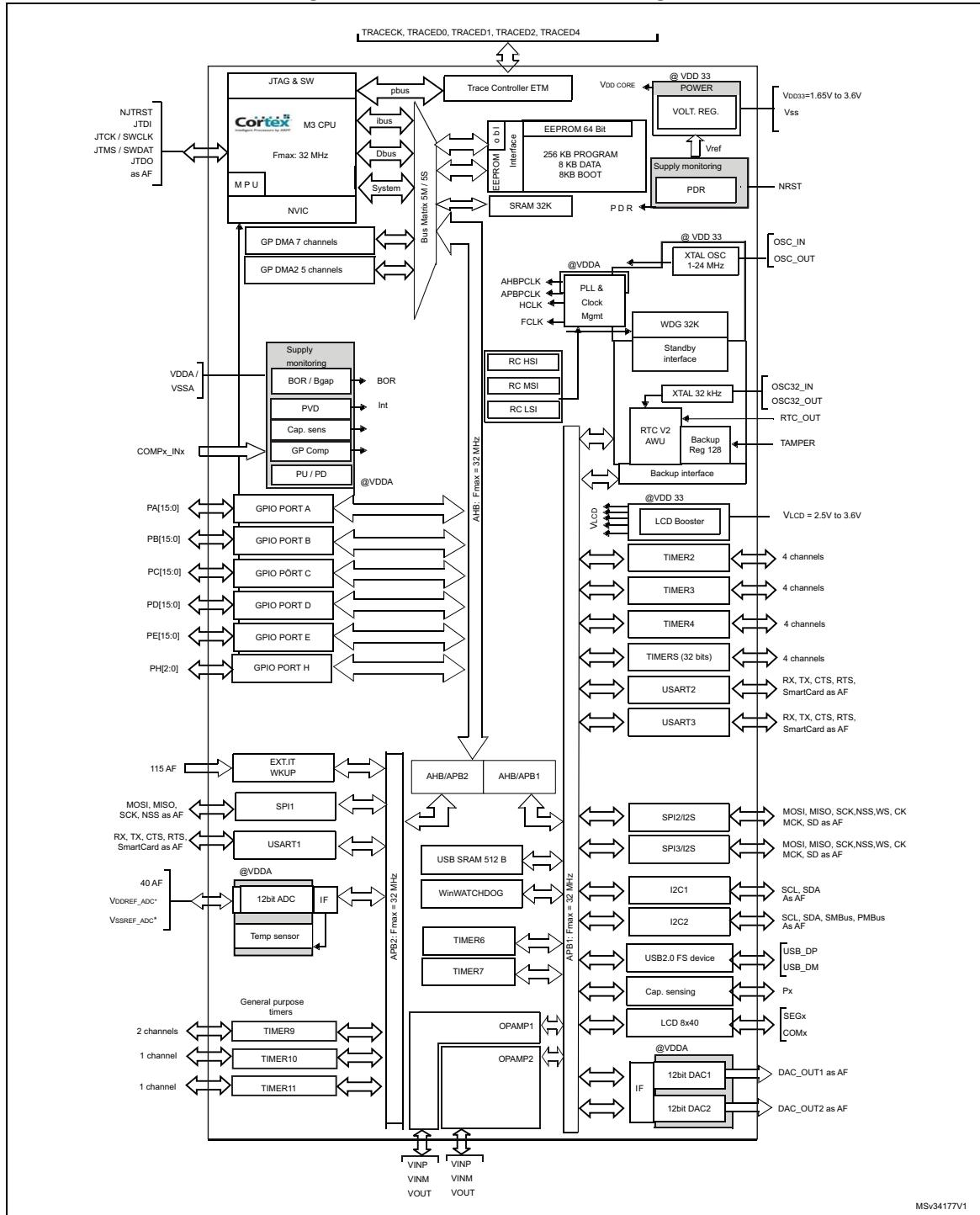
**Hardware and layout****UM1079****Figure 5. STM32L152RCT6 package**

The STM32L152RCT6 provides the following benefits:

- Ultra low power proprietary 130 nm technology: speed and power consumption independent of MCU power supply, and ultra low leakage
- Ultra Low power design (clock gating, low-power Flash with power-off capability): reduced overall Run and Wait mode current consumption by turning off clocks of unused peripherals or Flash
- Sub 1  $\mu$ A hardware RTC and AWU system unit:  
Ultra-low-power modes for applications requesting regular wake up
- Up to 6 Low-power modes: suitable for many applications from complete switch off to continuous monitoring at ultra low frequency
- Advanced and flexible clock system (multiple internal and external clock sources): switch and adjust frequency and clock sources on the fly depending on application needs
- Direct memory access on board (up to 12 DMA channels): autonomy for peripherals, independent from the core; can switch off Flash memory and CPU (large current consumption contributors) while keeping peripherals active
- Ultra Low power and ultrasafe features (POR, PDR, BOR, PVD) allowing integrated application safety and security
- Unique identifier to enhance user data confidentiality/reliability
- Ultrafast wakeup from lowest consumption low-power mode allowing fast switching from static and dynamic power modes
- Analog functional down to 1.8 V, and programming down to 1.65 V
- Full functionality over the complete  $V_{DD}$  range

For more information, refer to STM32L152RCT6 datasheet available on ST website.

Figure 6. STM32L152RCT6 block diagram



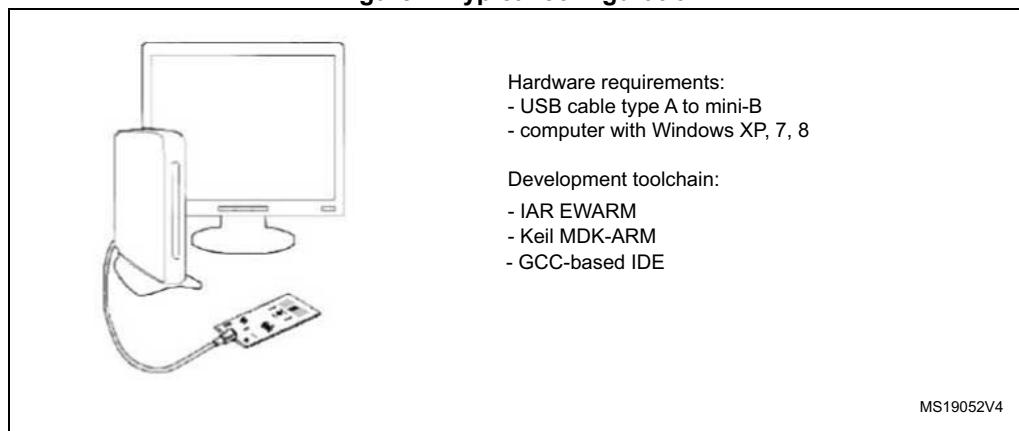
## 4.2 Embedded ST-LINK/V2

The ST-LINK/V2 programming and debugging tool is integrated on the 32L152CDISCOVERY. The embedded ST-LINK/V2 can be used in 2 different ways according to the jumper states (see [Table 4](#)):

- Program/debug the MCU on board
- Program/debug an MCU in an external application board using a cable connected to SWD connector CN2

The embedded ST-LINK/V2 supports only SWD for STM32 devices. For information about debugging and programming features, refer to the user manual *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32* (UM1075).

**Figure 7. Typical configuration**



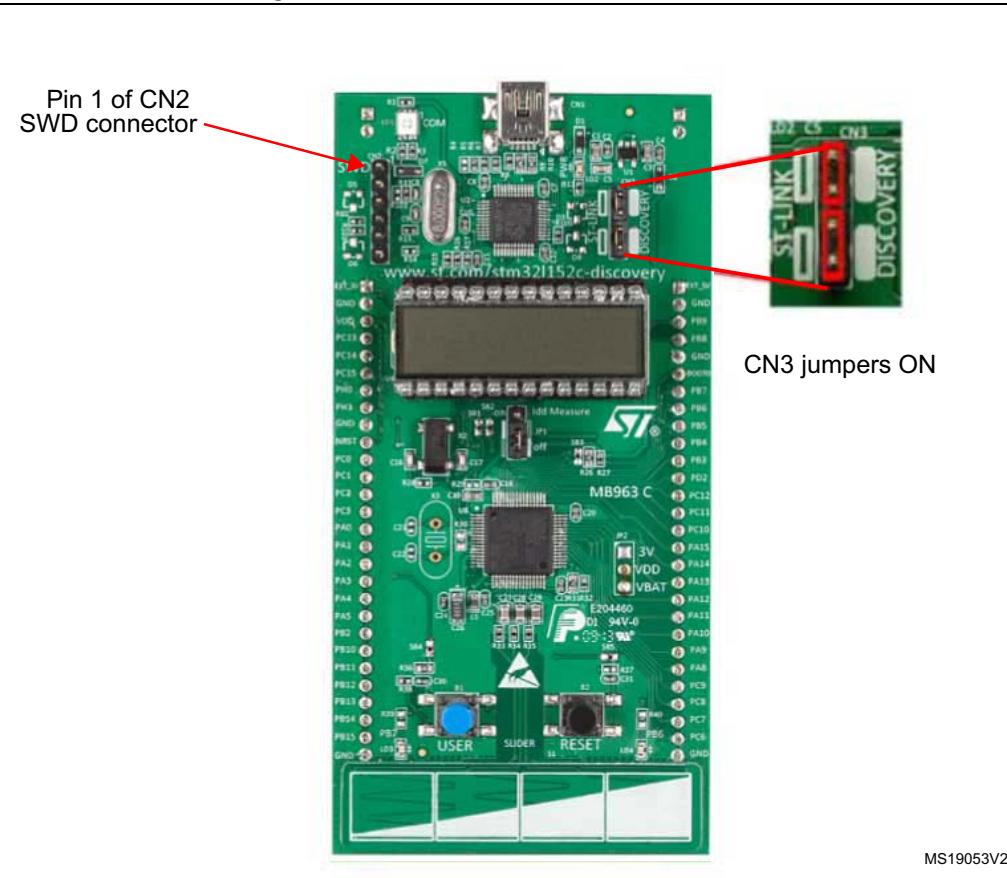
**Table 4. Jumper states**

Jumper state	Description
Both CN3 jumpers ON	ST-LINK/V2 functions enabled for on board programming (default)
Both CN3 jumpers OFF	ST-LINK/V2 functions enabled for external application through CN2 connector (SWD supported).

### 4.2.1 Using the ST-LINK/V2 to program/debug the microcontroller on board

[Figure 8](#) shows how to plug the two jumpers on CN3 to program the STM32L152RCT6 on the board. The usage of CN2 is forbidden as it could disturb communication with the microcontroller.

Figure 8. 32L152CDISCOVERY connections



#### 4.2.2 Using the ST-LINK/V2 to program/debug an external application

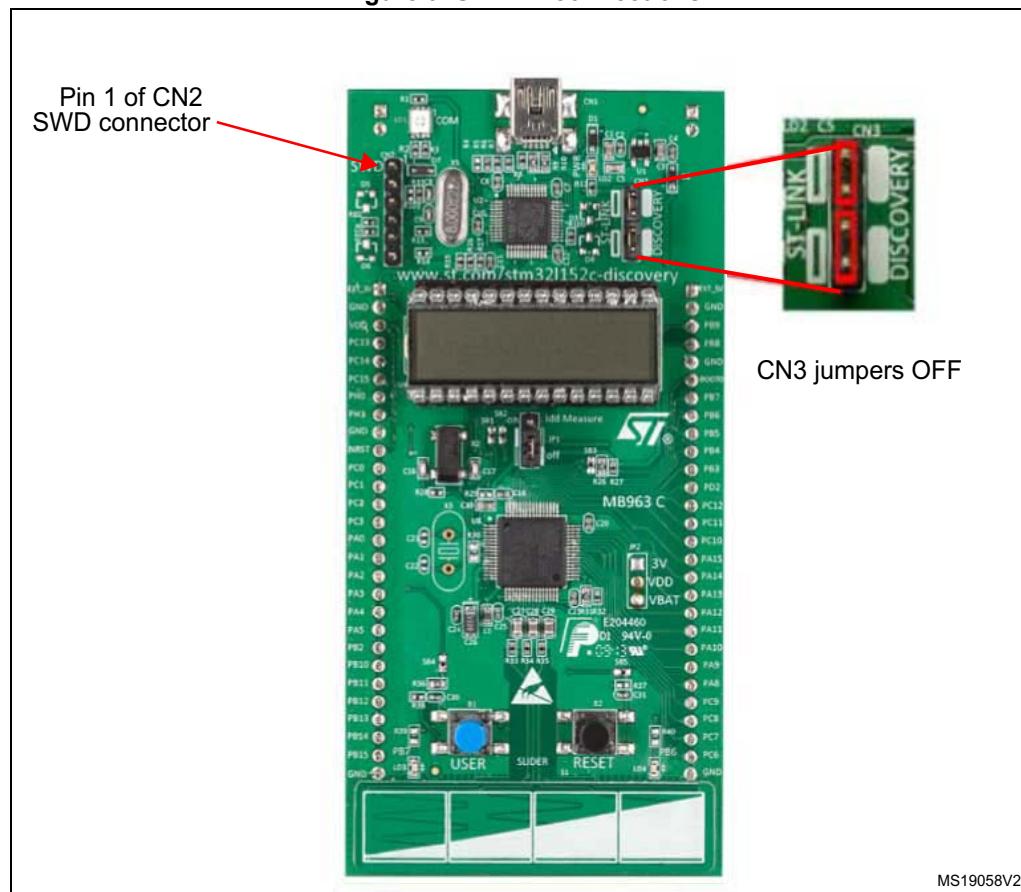
The ST-LINK/V2 allows also to program an STM32 device on an external application. [Figure 9](#) shows how to remove the 2 jumpers from CN3 and to connect the external application to the CN2 debug connector according to instructions in [Table 5](#).

*Note:* SB100 must be OFF if you the CN2 pin 5 is used in the external application.

Table 5. Debug connector CN2 (SWD)

Pin	CN2	Designation
1	VDD_TARGET	VDD from application
2	SWCLK	SWD clock
3	GND	Ground
4	SWDIO	SWD data input/output
5	NRST	RESET of target MCU
6	SWO	Reserved

Figure 9. ST-Link connections



## 4.3 Power supply and power selection

The power supply is provided either by the host computer through the USB cable, or by an external 5 V or 3.3 V power supply.

The D1 and D2 protection diodes allow to use the EXT\_5V and EXT\_3V pins independently as input or output power supplies (see [Figure 3](#)):

- EXT\_5V and EXT\_3V can be used as output power supplies when the application board is connected to pins P1 and P2. In this case, the EXT\_5V and EXT\_3V pins deliver a 5 V or 3 V power supply and power consumption must be lower than 100 mA.
- EXT\_5V and EXT\_3V can also be used as input power supplies when the USB connector is not connected to the computer. In this case, the power of the board must be provided by a power supply unit or by an auxiliary equipment complying with standard EN-60950-1: 2006+A11/2009. This power source must be Safety Extra Low Voltage (SELV) with limited power capability.

### Battery powered (optional)

The 32L152CDISCOVERY board has been designed to run from a CR2032 standalone battery (no connection with USB or other power supply is required).

---

UM1079	Hardware and layout
--------	---------------------

By default, no battery holder is mounted on the board and SB21 and SB22 are configured in their default state (see [Table 6: Solder bridges on page 21](#)).

Follow the procedure below to power the 32L152CDISCOVERY from the battery:

- Solder a B7410AP2L battery holder from LOTES on CR1
- Configure SB100 OFF
- Remove both jumpers from CN3 (see [Figure 9: ST-Link connections on page 16](#))
- Select the battery as power supply. Two solutions are possible:
  - Solder bridge: configure SB21 OFF, and SB22 ON. No header is required on JP2.
  - Jumper: configure SB21 and SB22 OFF. Solder a header on JP2, identical to JP1 on the top side. Set a jumper between VDD and VBAT to power the STM32L152RCT6 of the board

**Note:** *In this configuration, it is possible to power the STM32L152RCT6 from the 3 V supply voltage of the board by setting a jumper between VDD and 3V.*

- Plug the CR2032 battery into CR1 holder.

The demonstration is now ready to run.

---

**Warning: Wrong solder bridge configuration can damage board components.**

---

## 4.4 LEDs

- LD1 COM: LD1 default status is red. LD1 turns to green to indicate that communications are in progress between the computer and the ST-LINK/V2.
- LD2 PWR: red LED indicates that the board is powered.
- User LD3: user green LED connected to the I/O PB7 of the STM32L152RCT6.
- User LD4: user blue LED connected to the I/O PB6 of the STM32L152RCT6.

## 4.5 Pushbuttons

- B1 USER: User pushbutton connected to the I/O PA0 of the STM32L152RCT6.
- B2 RESET: Pushbutton is used to RESET the STM32L152RCT6.

## 4.6 Linear touch sensor / touchkeys

To demonstrate touch sensing capabilities, the 32L152CDISCOVERY includes a linear touch sensor which can be used either as a 3-position linear touch sensor or as 4 touchkeys. Both functionalities are illustrated in the demonstration software (see [Table 3: Functions executed when clicking B1 button on page 7](#)).

**Hardware and layout****UM1079**

Three pairs of I/O ports are assigned to the linear touch sensor / touchkeys. Each pair must belong to the same analog switch group:

- PA6, PA7 (group 2)
- PC4, PC5 (group 9)
- PB0, PB1 (group 3)

To minimize the noise, these pairs are dedicated to the linear touch sensor / touchkeys and are not connected to external headers.

To design a touch sensing application, refer to the following documentation and firmware:

- For details concerning I/O ports, refer to the STM32L152RCT6 datasheet.
- For information on software development, see discovery application software on [www.st.com/mcu](http://www.st.com/mcu).
- For more detail concerning touch sensing application design and layout, refer to *Guidelines for designing touch sensing applications with surface sensors* (AN4312).
- STM32 touch sensing library available from [www.st.com/mcu](http://www.st.com/mcu)

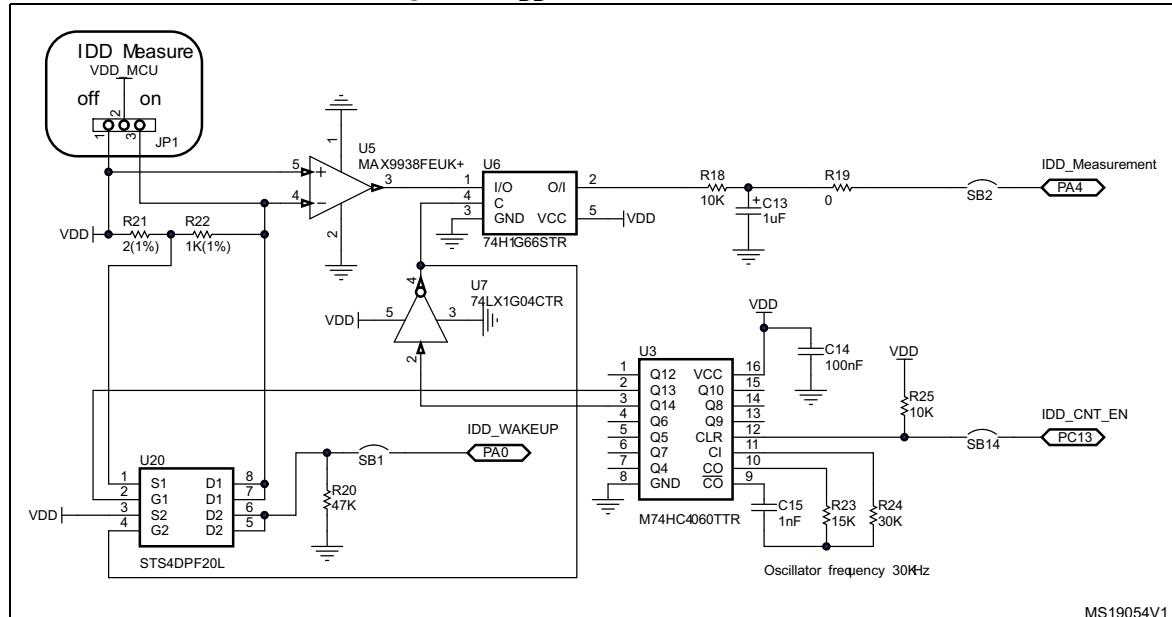
## 4.7 Built-in $I_{DD}$ measurement circuit

The 32L152CDISCOVERY built-in  $I_{DD}$  measurement circuit allows to measure the consumption of the STM32L152RCT6 and to display the value on the LCD glass while the MCU is in Run or Low-power modes.

- JP1 ON: the STM32L152RCT6 is powered through the  $I_{DD}$  measurement circuit (default).
- JP1 OFF: the STM32L152RCT6 is directly powered. The  $I_{DD}$  measurement circuit is bypassed.

**Note:** When jumper JP1 is removed, the current consumption of the STM32L152RCT6 can be measured by connecting an ammeter between jumper pin 1 and pin 2 of JP1.

To perform the  $I_{DD}$  measurement by the MCU itself, the circuit shown in [Figure 10](#) is implemented on the 32L152CDISCOVERY. The solder bridges SB1, SB2 and SB14 must be closed and JP1 must be ON. The low IDD range procedure (see [Section 4.7.2](#)) is recommended when the MCU is in Low-power mode and the IDD current does not exceed 60  $\mu$ A. The high IDD range procedure (see [Section 4.7.1](#)) is applicable when the MCU operates in Run mode and can sink up to 30 mA.

Figure 10.  $I_{DD}$  measurement circuit

#### 4.7.1 High $I_{DD}$ range mode

In high  $I_{DD}$  range mode, the  $I_{DD}$  current is measured using the operational amplifier MAX9938FEUK+ (U5) connected to the  $2\Omega$  shunt resistor (R21). In this case IDD\_CNT\_EN remains high during the measurement. R22 remains in short-circuit during the measurement because the FET transistor 1 of U20 remains ON permanently.

#### 4.7.2 Low $I_{DD}$ range mode

In low  $I_{DD}$  range mode, the operational amplifier MAX9938FEUK+ (U5) is connected to the  $1\text{K}\Omega$  shunt resistor (R22), controlled by the FET transistor 1 of U20. In this case the counter 74HC4060 (U3) enabled by IDD\_CNT\_EN manages the measurement timing according to [Figure 11](#).

##### Low $I_{DD}$ range measurement principle

The principle used to measure the consumption current when the STM32L152RCT6 is in low  $I_{DD}$  range mode is as follows:

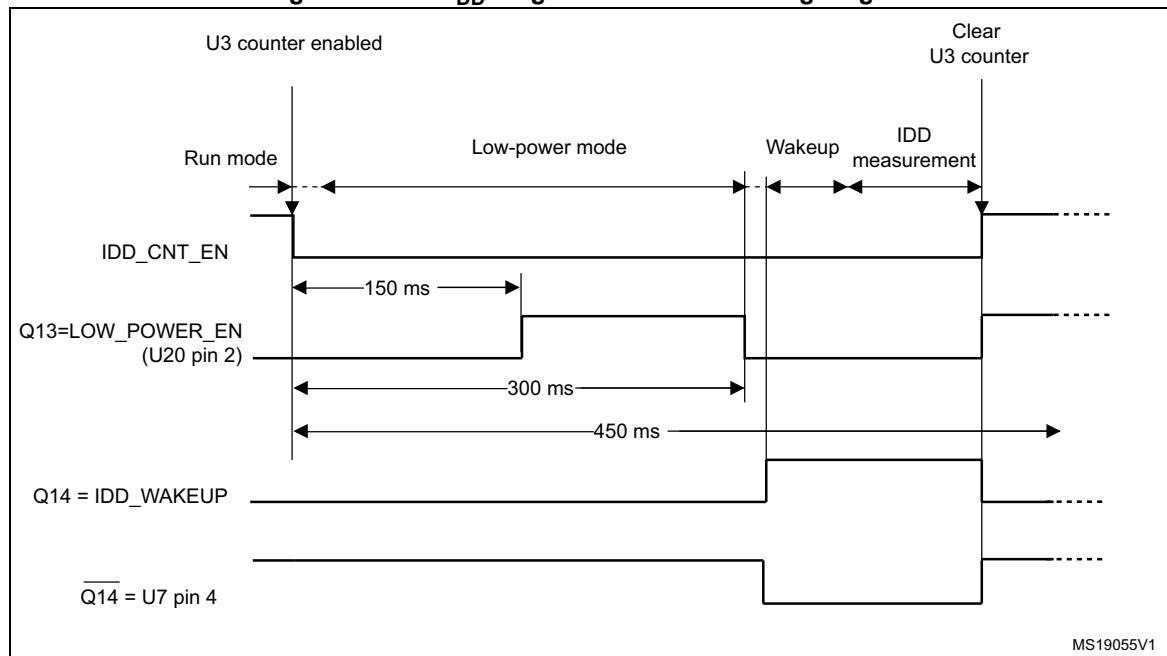
1. Configure ADC to measure voltage on the IDD\_Measurement pin.
2. Configure PA0 to serve as wakeup pin.
3. Enter low  $I_{DD}$  range mode after setting IDD\_CNT\_EN (PC13) signal low.
4. IDD\_WAKEUP rising edge wakes up the MCU after around 300 ms.
5. Start ADC conversion as soon as possible after wakeup in order to measure the voltage corresponding to Low-power mode on capacitor C13.
6. Reset the counter by programming IDD\_CNT\_EN high (in less than 150 ms after the wakeup) to avoid the R22  $1\text{K}\Omega$  resistor being connected later in Run mode.

The measurement timing is given in [Figure 11](#). In low  $I_{DD}$  range mode, the  $1\text{K}\Omega$  resistor is connected when the FET transistor 1 of U20 goes OFF, after entering low  $I_{DD}$  range mode.

**Hardware and layout****UM1079**

The Q13 output of the counter allows connecting the 1 KW resistor when the current  $I_{DD}$  becomes very low.

*Figure 11* shows how the counter and the FET transistor 1 of U20 ensure that, 150 ms after  $IDD\_CNT\_EN$  falling edge, the shunt resistor R22 is connected between  $VDD\_MCU$  and the power supply to reduce the measurement range to 60  $\mu A$  for the full scale. Then after another 150 ms required for current stabilization, R22 is shorted, the  $I_{DD}$  measurement is stored in C13, and the MCU is woken up. After wakeup, the MCU measures the  $I_{DD}$  current corresponding to the Low-power mode stored in C13.

**Figure 11. Low  $I_{DD}$  range measurement timing diagram****4.7.3  $I_{BIAS}$  current measurement procedure**

In low  $I_{DD}$  range mode, the bias current of the operational amplifier input (U5 pin 4) is not negligible compared to  $I_{DD}$  current (typical  $I_{BIAS}$  is ~240 nA). To obtain a reliable  $I_{DD}$  measurement, it is mandatory to subtract the bias current from the low  $I_{DD}$  current value since this current is not sunk by the MCU.  $I_{BIAS}$  is measured during production test and stored in the MCU data EEPROM. The discovery demonstration software uses this value to display the correct  $I_{DD}$ .

The procedure for  $I_{BIAS}$  measurement implemented in the demonstration software is:

1. Power off the board (disconnect the USB cable).
2. Set JP1 OFF.
3. Push down B1 (USER button), power on the board from the USB.
4. Wait at least 1 second before releasing B1. The LCD displays the  $I_{BIAS}$  measurement.
5. Power off the board (disconnect the USB cable).
6. Set JP1 ON. The  $I_{BIAS}$  value is stored in data EEPROM. The bias current is then subtracted from the  $I_{DD}$  measured in  $I_{DD}$  range mode.

## 4.8 Solder bridges

Table 6. Solder bridges

Bridge	State <sup>(1)</sup>	Description
SB18,20 (X3 crystal) <sup>(2)</sup>	ON	PH0, PH1 are connected to P1 (X3, C21, C22, R30 must not be fitted).
	OFF	X3, C21, C22 and R30 provide a clock as shown in <a href="#">Section 7: Electrical schematics</a> . PH0, PH1 are disconnected from P1.
SB7,9,11,13 (DEFAULT)	ON	Reserved, do not modify.
SB6,8,10,12 (RESERVED)	OFF	Reserved, do not modify.
SB1,2,14 (IDD_Measurement)	ON	PA0, PA4, PC13 are used by the I <sub>DD</sub> measurement. JP1 ON.
	OFF	PA0, PA4, PC13 are available and IDD module cannot be used JP1 OFF.
SB15,16 (X2 crystal)	OFF	X2, C16, C17 and R28 deliver a 32 KHz clock. PC14, PC15 are not connected to P1.
	ON	PC14, PC15 are only connected to P1. Do not remove X2, C16, C17, R28.
SB5 (B2-RESET)	ON	B2 Pushbutton is connected to the NRST pin of the STM32L152 MCU.
	OFF	B2 Pushbutton is not connected the NRST pin of the STM32L152 MCU.
SB4 (B1-USER)	ON	B1 Pushbutton is connected to PA0.
	OFF	B1 Pushbutton is not connected to PA0.
SB21 (VDD powered from 3 V)	ON	V <sub>DD</sub> is powered from 3 V, SB22 must be OFF.
	OFF	V <sub>DD</sub> is not powered from 3 V, SB22 must be ON.
SB22 (Battery enable)	OFF	V <sub>DD</sub> is not powered by the CR2032 battery, SB21 must be ON.
	ON	V <sub>DD</sub> is powered by the CR2032 battery, SB21 must be OFF.
SB100 (NRST)	ON	The NRST signal of the CN2 connector is connected to the NRST pin of the STM32L152RCT6.
	OFF	The NRST signal of the CN2 connector is not connected to the NRST pin of the STM32L152RCT6.
SB101 (SWO)	ON	The SWO signal of the CN2 connector is connected to PB3.
	OFF	The SWO signal is not connected.
SB102 (STM_RST)	OFF	No incidence on STM32L152RCT6 NRST signal.
	ON	STM32L152RCT6 NRST signal is connected to GND.

**Table 6. Solder bridges (continued)**

Bridge	State <sup>(1)</sup>	Description
SB3 (BOOT0)	<b>ON</b>	The BOOT0 signal of the STM32L152RCT6 is held low through a 510 Ω pull-down resistor.
	OFF	The BOOT0 signal of the STM32L152RCT6 is held high through a 10 KΩ pull-up resistor.
SB19 (BOOT1)	<b>OFF</b>	The BOOT1 signal of the STM32L152RCT6 is held high through a 10 KΩ pull-up resistor.
	ON	The BOOT1 signal of the STM32L152RCT6 is held low through a 510 Ω pull-down resistor.
SB17 (MCO) <sup>(2)</sup>	<b>OFF</b>	STM32L152RCT6 MCO clock signal is not used.
	ON	STM32L152RCT6 MCO clock signal is connected to OSC_IN of the STM32L152RCT6

1. Default SBx state is shown in bold.
2. SB17 and SB20 are OFF to allow the user to choose between MCO and X3 crystal for clock source.

## 4.9 LCD (24 segments, 4 commons)

This LCD allows the STM32L152RCT6 to display any information on six 14-segment digits and 4 bars, using all COMs. (See the LCD segment mapping in [Figure 17](#) and pin connections in [Table 7](#).)

*Note:*

*This LCD also supports six 8-segment digits by only using COM0 and COM1.*

*This configuration allows COM2 and COM3 to be used as I/O ports. In this case the 2 LCD pins must not be plugged into the LCD socket. To proceed with this configuration, remove the LCD carefully, slightly open the COM2 and COM3 pins (pin 13 and pin 14) of the LCD, then replug it in the socket.*

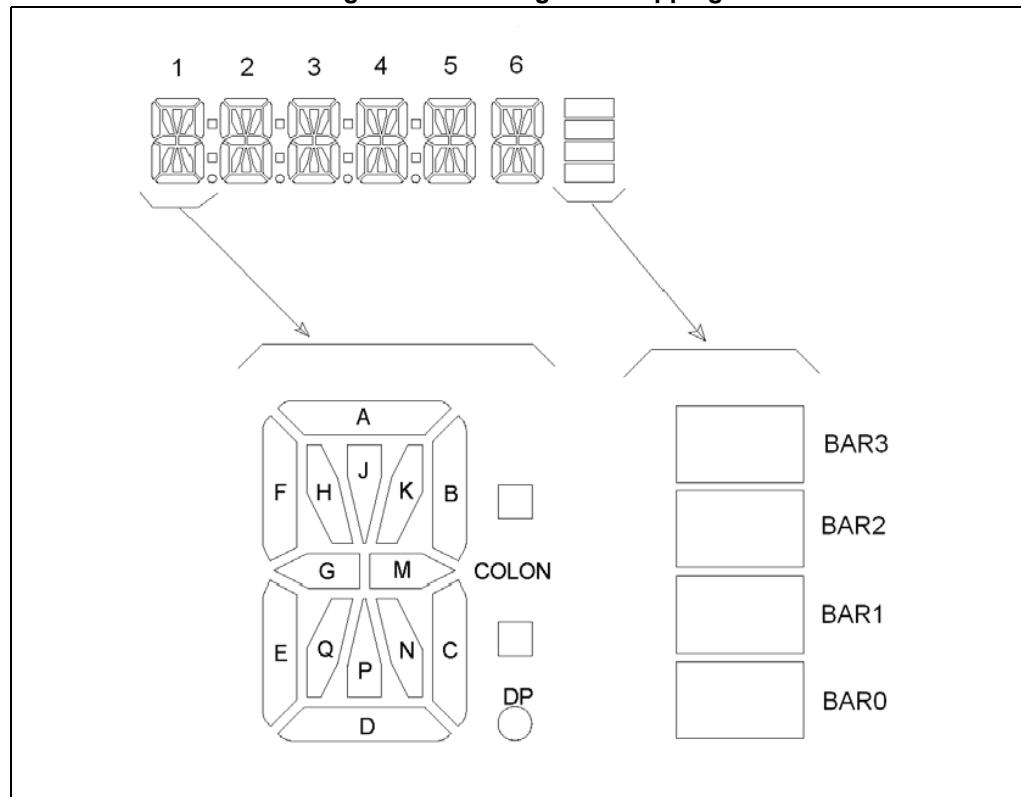
Characteristics overview:

- 24 segments and 4 commons
- Drive method: multiplexed 1/4 duty, 1/3 bias
- Operating voltage: 3 V
- Operating temperature: 0 to 50°C
- Connector: 28-pin DIL 2.54 mm pitch

*Note:*

*When the LCD is plugged, all I/O ports listed in [Table 7](#) are unavailable. To use one of these as I/O, you must remove the LCD.*

**Figure 12. LCD segment mapping**



**Hardware and layout****UM1079****Table 7. LCD connections**

<b>STM32L152RCT6</b>	<b>LCD</b>					
<b>GPIO Name</b>	<b>Pin</b>	<b>COM3</b>	<b>COM2</b>	<b>COM1</b>	<b>COM0</b>	<b>Name</b>
PA1	1	1N	1P	1D	1E	LCDSEG0
PA2	2	1DP	1COLON	1C	1M	LCDSEG1
PA3	3	2N	2P	2D	2E	LCDSEG2
PB3	4	2DP	2COLON	2C	2M	LCDSEG3
PB4	5	3N	3P	3D	3E	LCDSEG4
PB5	6	3DP	3COLON	3C	3M	LCDSEG5
PB10	7	4N	4P	4D	4E	LCDSEG6
PB11	8	4DP	4COLON	4C	4M	LCDSEG7
PB12	9	5N	5P	5D	5E	LCDSEG8
PB13	10	BAR2	BAR3	5C	5M	LCDSEG9
PB14	11	6N	6P	6D	6E	LCDSEG10
PB15	12	BAR0	BAR1	6C	6M	LCDSEG11
PB9	13	COM3	-	-	-	LCDCOM3
PA10	14	-	COM2	-	-	LCDCOM2
PA9	15	-	-	COM1	-	LCDCOM1
PA8	16	-	-	-	COM0	LCDCOM0
PA15	17	6J	6K	6A	6B	LCDSEG12
PB8	18	6H	6Q	6F	6G	LCDSEG13
PC0	19	5J	5K	5A	5B	LCDSEG14
PC1	20	5H	5Q	5F	5G	LCDSEG15
PC2	21	4J	4K	4A	4B	LCDSEG16
PC3	22	4H	4Q	4F	4G	LCDSEG17
PC6	23	3J	3K	3A	3B	LCDSEG18
PC7	24	3H	3Q	3F	3G	LCDSEG19
PC8	25	2J	2K	2A	2B	LCDSEG20
PC9	26	2H	2Q	2F	2G	LCDSEG21
PC10	27	1J	1K	1A	1B	LCDSEG22
PC11	28	1H	1Q	1F	1G	LCDSEG23

## 5 Extension connectors

The male headers P1 and P2 can connect the 32L152CDISCOVERY to a standard prototyping/wrapping board. The STM32L152RCT6 GPIOs are available on these connectors. P1 and P2 can also be probed by an oscilloscope, a logical analyzer or a voltmeter.

**Table 8. MCU pin description versus board function**

MCU pin			Board function											
Main function	Alternate functions	LQF P64 pin num	LCD glass	Linear Touch Sensor	Push butt on	I <sub>DD</sub>	LED	SWD	OSC	Free I/O	Power supply	P1	P2	
-	-	-	-	-	-	-	-	-	-	-	EXT_3V	1	-	
-	-	-	-	-	-	-	-	-	-	-	EXT_5V		1	
BOOT0	-	60	-	-	-	-	-	-	-	-	-	-	-	6
NRST	-	7	-	-	-	-	-	-	NRS T	-	-	10	-	
PA0	WKUP1/USART2_CTS/ ADC_IN0/TIM2_CH1_ETR/COMP1_INP	14	-	-	PA0	WAKE UP	-	-	-	-	-	15	-	
PA1	USART2_RTS/ADC_IN1/ TIM2_CH2/LCD SEG0/COMP1_INP	15	SEG 0	-	-	-	-	-	-	-	-	16	-	
PA2	USART2_TX/ADC_IN2/ TIM2_CH3/TIM9_CH1/ LCD_SEG1/COMP1_INP	16	SEG 1	-	-	-	-	-	-	-	-	17	-	
PA3	USART2_RX/ADC_IN3/ TIM2_CH4/TIM9_CH2/ LCD_SEG2/COMP1_INP	17	SEG 2	-	-	-	-	-	-	-	-	18	-	
PA4	SPI1_NSS/USART2_CK/ ADC_IN4/DAC_OUT1/COMP1_INP	20	-	-	-	Measur ement	-	-	-	-	-	19	-	

## Extension connectors

UM1079

Table 8. MCU pin description versus board function (continued)

MCU pin			Board function											
Main function	Alternate functions	LQF P64 pin num	LCD glass	Linear Touch Sensor	Push button	I <sub>DD</sub>	LED	SWD	OSC	Free I/O	Power supply	P1	P2	
PA5	SPI1_SCK/ADC_IN5/DAC_OUT2/TIM2_CH1_ETR/COMP1_INP	21	-	-	-	-	-	-	-	X	-	20	-	
PA6	SPI1_MISO/ADC_IN6/TIM3_CH1/TIM1_BKIN/LCD_SEG3/TIM10_CH1/COMP1_INP	22	-	PA6	-	-	-	-	-	-	-	-	-	
PA7	SPI1_MOSI/ADC_IN7/TIM3_CH2/TIM1_CH1N/LCD_SEG4/TIM11_CH1/COMP1_INP	23	-	PA7	-	-	-	-	-	-	-	-	-	
PA8	USART1_CK/MCO/LCD_COM0	41	COM0	-	-	-	-	-	-	-	-	-	23	
PA9	USART1_TX/LCD_COM1	42	COM1	-	-	-	-	-	-	-	-	-	22	
PA10	USART1_RX/LCD_COM2	43	COM2	-	-	-	-	-	-	-	-	-	21	
PA11	USART1_CTS/USBDM/SPI1_MISO	44	-	-	-	-	-	-	-	X	-	-	20	
PA12	USART1_RTS/USBDP/SPI1_MOSI	45	-	-	-	-	-	-	-	X	-	-	19	
JTMS/SWDIO	PA13	46	-	-	-	-	-	SWDIO	-	-	-	-	18	
JTCK/SWCLK	PA14	49	-	-	-	-	-	SWCLK	-	-	-	-	17	
JTDI	TIM2_CH1_ETR/PA15/SPI1_NSS/LCD SEG17	50	SEG12	-	-	-	-	-	-	-	-	-	16	
PB0	ADC_IN8/TIM3_CH3/LCD_SEG5/COMP1_INP/VREF_OUT	26	-	PB0	-	-	-	-	-	-	-	-	-	

UM1079

Extension connectors

Table 8. MCU pin description versus board function (continued)

MCU pin			Board function											
Main function	Alternate functions	LQF P64 pin num	LCD glass	Linear Touch Sensor	Push butt on	I <sub>DD</sub>	LED	SWD	OSC	Free I/O	Power supply	P1	P2	
PB1	ADC_IN9/TIM3_CH4/ LCD_SEG6/COMP1_INP/VREF_OUT	27	-	PB1	-	-	-	-	-	-	-	-	-	-
PB2/BOOT1	-	28	-	-	-	-	-	-	-	-	-	21	-	-
JTDO	TIM2_CH2/PB3/TRACESWO/SPI1_SCK/COMP2_INM/LCD_SEG7	55	SEG3	-	-	-	-	SWO	-	-	-	-	-	11
JNTRST	TIM3_CH1/PB4/SP11_MISO/COMP2_INP/LCD_SEG8	56	SEG4	-	-	-	-	-	-	-	-	-	-	10
PB5	I2C1_SMBAI/TIM3_CH2/ SPI1_MOSI/COMP2_INP/LCD_SEG9	57	SEG5	-	-	-	-	-	-	-	-	-	-	9
PB6	I2C1_SCL/TIM4_CH1/ USART1_TX/LCD_SEG8	58	-	-	-	-	Blue	-	-	-	-	-	-	8
PB7	I2C1_SDA/TIM4_CH2/ USART1_RX/PVD_IN	59	-	-	-	-	Green	-	-	-	-	-	-	7
PB8	TIM4_CH3/I2C1_SCL/ LCD_SEG16/TIM10_CH1	61	SEG13	-	-	-	-	-	-	-	-	-	-	4
PB9	TIM4_CH4/I2C1_SDA/ LCD_COM3/TIM11_CH1	62	COM3	-	-	-	-	-	-	-	-	-	-	3
PB10	I2C2_SCL/USART3_TX/ TIM2_CH3/LCD SEG10	29	SEG6	-	-	-	-	-	-	-	-	-	22	-
PB11	I2C2_SDA/USART3_RX/ TIM2_CH4/LCD SEG11	30	SEG7	-	-	-	-	-	-	-	-	-	23	-

## Extension connectors

UM1079

Table 8. MCU pin description versus board function (continued)

MCU pin			Board function											
Main function	Alternate functions	LQF P64 pin num	LCD glass	Linear Touch Sensor	Push button	I <sub>DD</sub>	LED	SWD	OSC	Free I/O	Power supply	P1	P2	
PB12	SPI2_NSS/I2C2_S MBA/ USART3_CK/LCD_ SEG12/ADC_IN18/ COMP1_INP/ TIM10_CH1	33	SEG 8	-	-	-	-	-	-	-	-	24	-	
PB13	SPI2_SCK/USART 3_CTS/ LCD SEG13/ADC_ IN19/ COMP1_INP/TIM9 _CH1	34	SEG 9	-	-	-	-	-	-	-	-	25	-	
PB14	SPI2_MISO/USAR T3_RTS/LCD_SEG 14/ADC_IN20/ COMP1_INP/TIM9 _CH2	35	SEG 10	-	-	-	-	-	-	-	-	26	-	
PB15	SPI2_MOSI/TIM1_ CH3N/ LCD SEG15/ADC_ IN21/ COMP1_INP/TIM1 _1_CH1/ RTC_50_60Hz	36	SEG 11	-	-	-	-	-	-	-	-	27	-	
PC0	ADC_IN10/LCD_S EG18/ COMP1_INP	8	SEG 14	-	-	-	-	-	-	-	-	11	-	
PC1	ADC_IN11/LCD_S EG19/ COMP1_INP	9	SEG 15	-	-	-	-	-	-	-	-	12	-	
PC2	ADC_IN12/LCD_S EG20/ COMP1_INP	10	SEG 16	-	-	-	-	-	-	-	-	13	-	
PC3	ADC_IN13/LCD_S EG21/ COMP1_INP	11	SEG 17	-	-	-	-	-	-	-	-	14	-	
PC4	ADC_IN14/LCD_S EG22/ COMP1_INP	24	-	PC4	-	-	-	-	-	-	-	-	-	
PC5	ADC_IN15/LCD_S EG23/ COMP1_INP	25	-	PC5	-	-	-	-	-	-	-	-	-	

UM1079

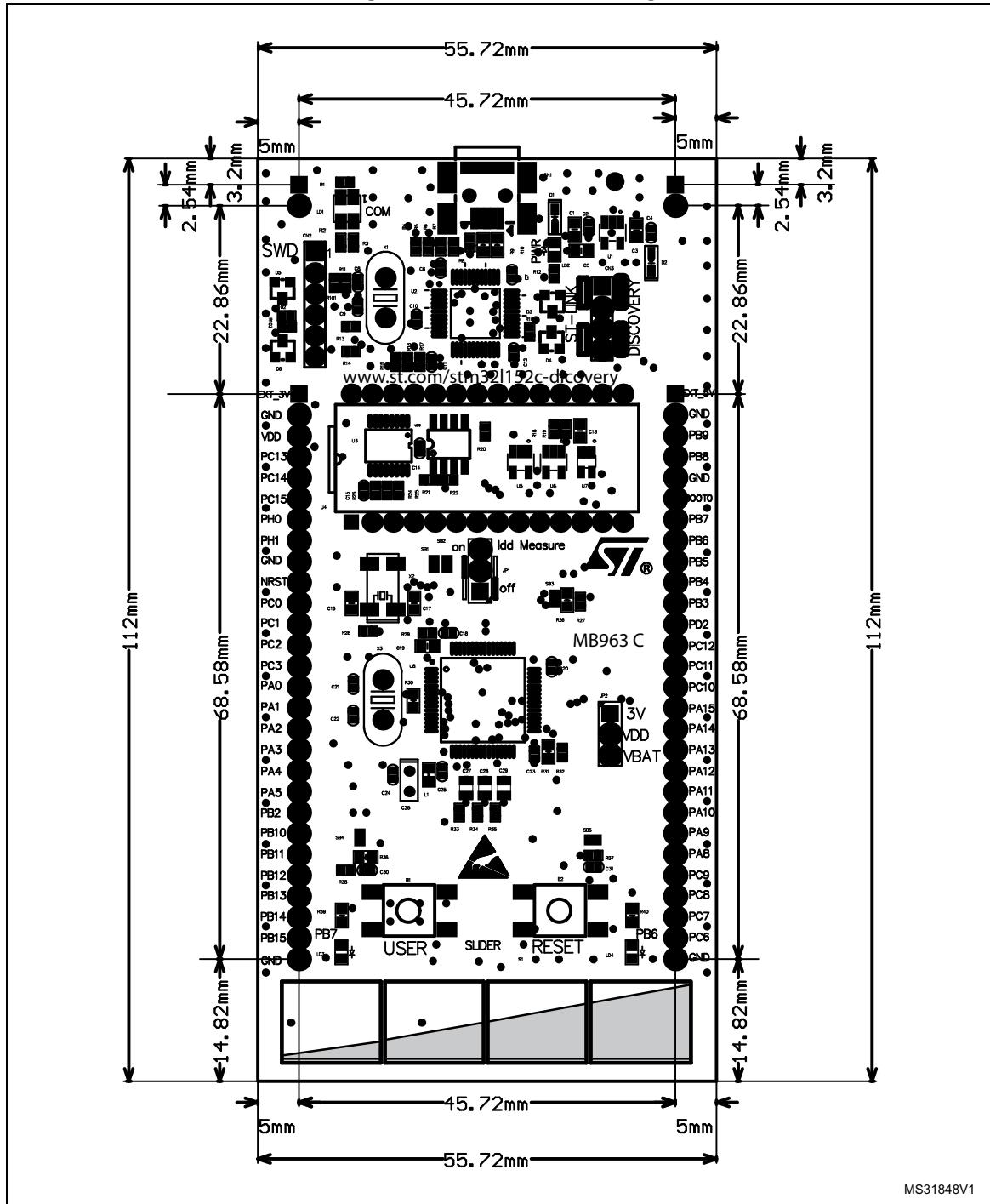
Extension connectors

Table 8. MCU pin description versus board function (continued)

MCU pin			Board function											
Main function	Alternate functions	LQF P64 pin num	LCD glass	Linear Touch Sensor	Push butt on	I <sub>DD</sub>	LED	SWD	OSC	Free I/O	Power supply	P1	P2	
PC6	TIM3_CH1/LCD SEG24	37	SEG 18	-	-	-	-	-	-	-	-	-	-	27
PC7	TIM3_CH2/LCD SEG25	38	SEG 19	-	-	-	-	-	-	-	-	-	-	26
PC8	TIM3_CH3/LCD SEG26	39	SEG 20	-	-	-	-	-	-	-	-	-	-	25
PC9	TIM3_CH4/LCD SEG27	40	SEG 21	-	-	-	-	-	-	-	-	-	-	24
PC10	USART3_TX/LCD SEG28/LCD_SEG4 0/LCD_COM4	51	SEG 22	-	-	-	-	-	-	-	-	-	-	15
PC11	USART3_RX/LCD SEG29/LCD_SEG4 1/LCD_COM5	52	SEG 23	-	-	-	-	-	-	-	-	-	-	14
PC12	USART3_CK/LCD SEG30/LCD_SEG4 2/LCD_COM6	53	-	-	-	-	-	-	-	X	-	-	-	13
PC13	RTC_AF1/WKUP2	2	-	-	-	CNT EN	-	-	-	-	-	-	4	-
PC14	OSC32_IN	3	-	-	-	-	-	-	OSC 32_IN	-	-	5	-	
PC15	OSC32_OUT	4	-	-	-	-	-	-	OSC 32_OUT	-	-	6	-	
PD2	TIM3_ETR/LCD SEG31/LCD_SEG43/LCD_COM7	54	-	-	-	-	-	-	-	X	-	-	-	12
OSC_IN	PH0	5	-	-	-	-	-	-	OSC _IN	-	-	7	-	
OSC_O_UT	PH1	6	-	-	-	-	-	-	OSC _OUT	-	-	8	-	
-	-	-	-	-	-	-	-	-	-	-	GND	2	2	
-	-	-	-	-	-	-	-	-	-	-	GND	9	5	
-	-	-	-	-	-	-	-	-	-	-	GND	28	28	
-	-	-	-	-	-	-	-	-	-	-	VDD	3	-	

## 6 Mechanical drawing

Figure 13. Mechanical drawing

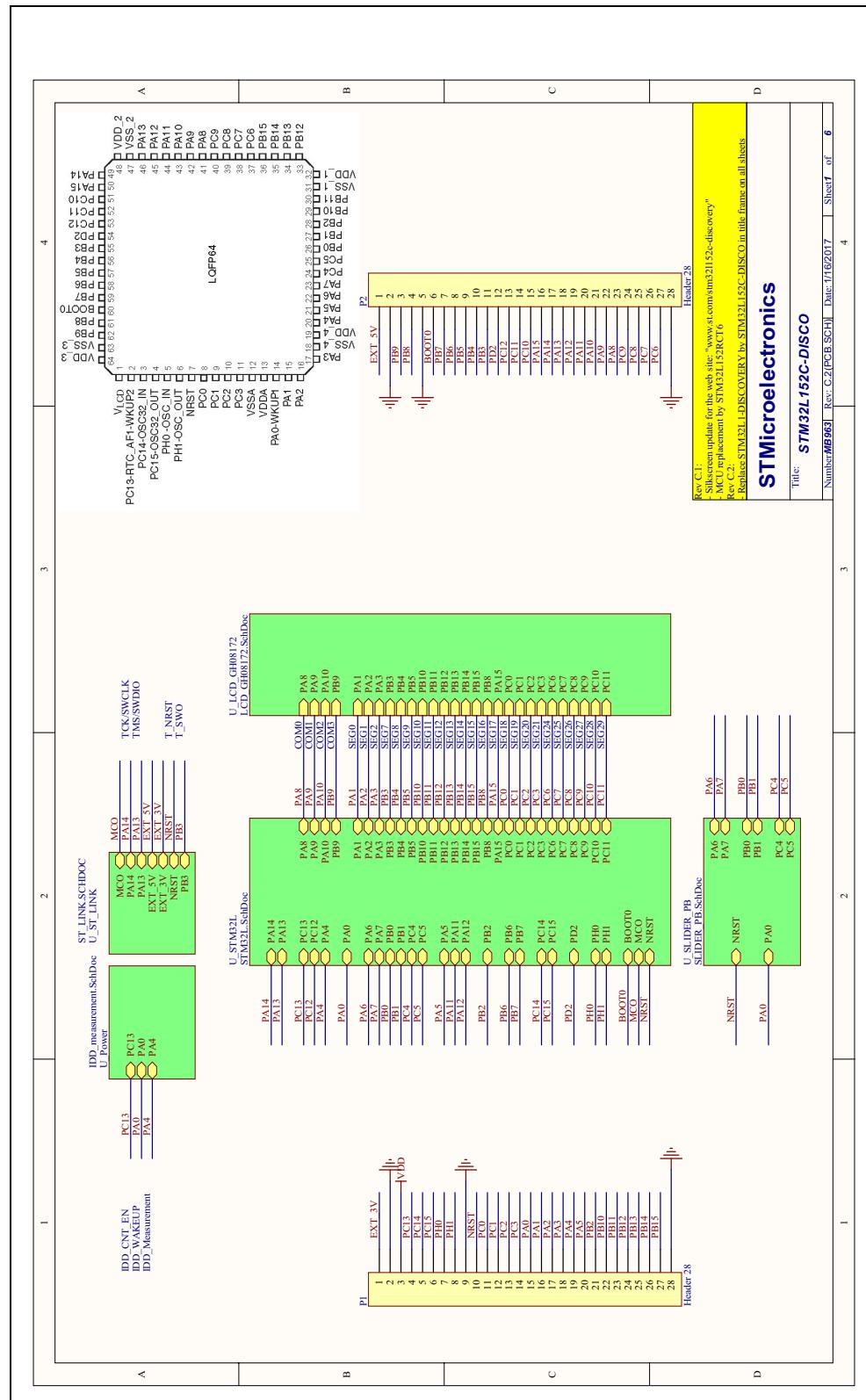


## 7 Electrical schematics

**UM1079**

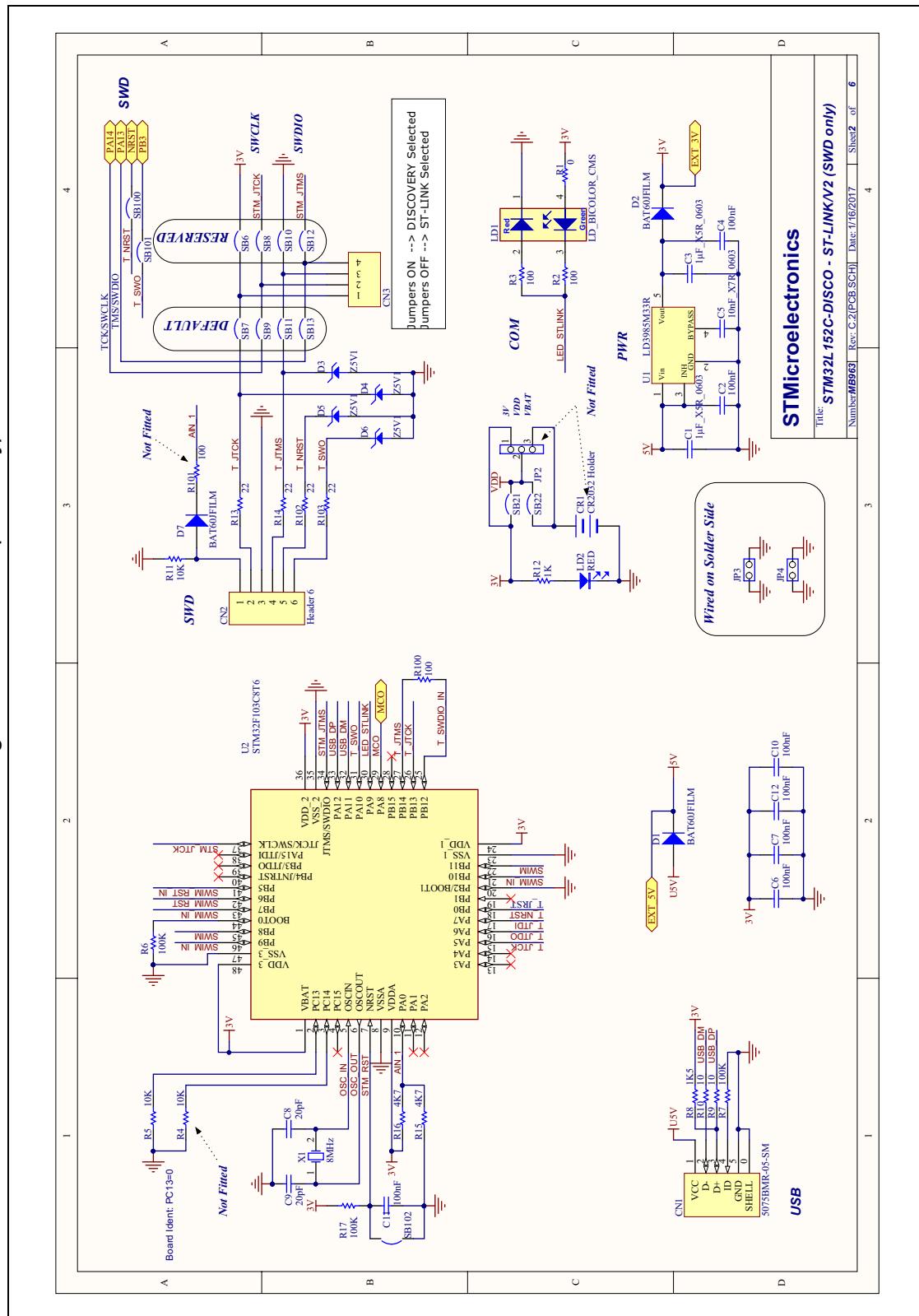
**Electrical schematics**

**Figure 14. 32L152CDISCOVERY**



## Electrical schematics

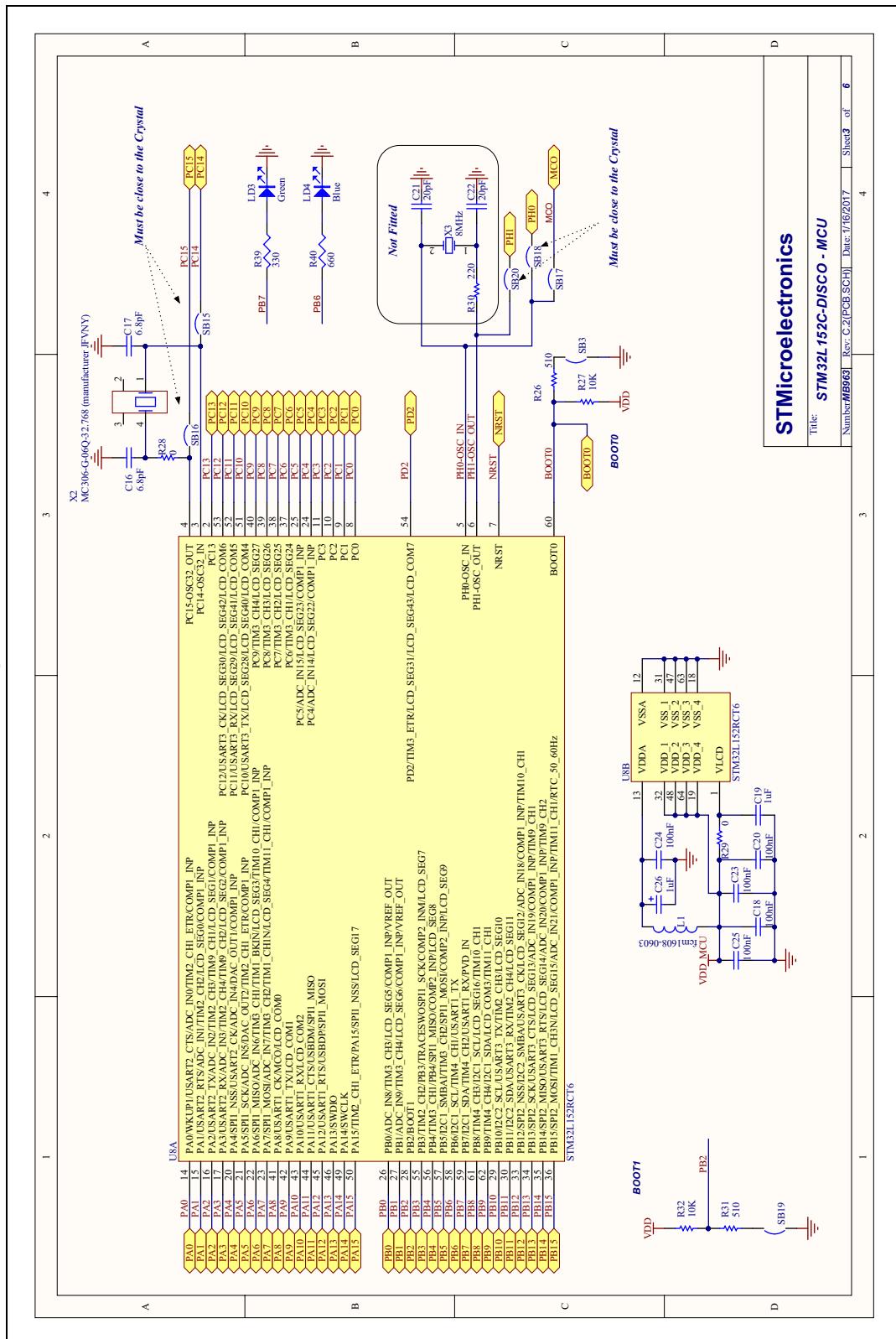
UM1079



## UM1079

## Electrical schematics

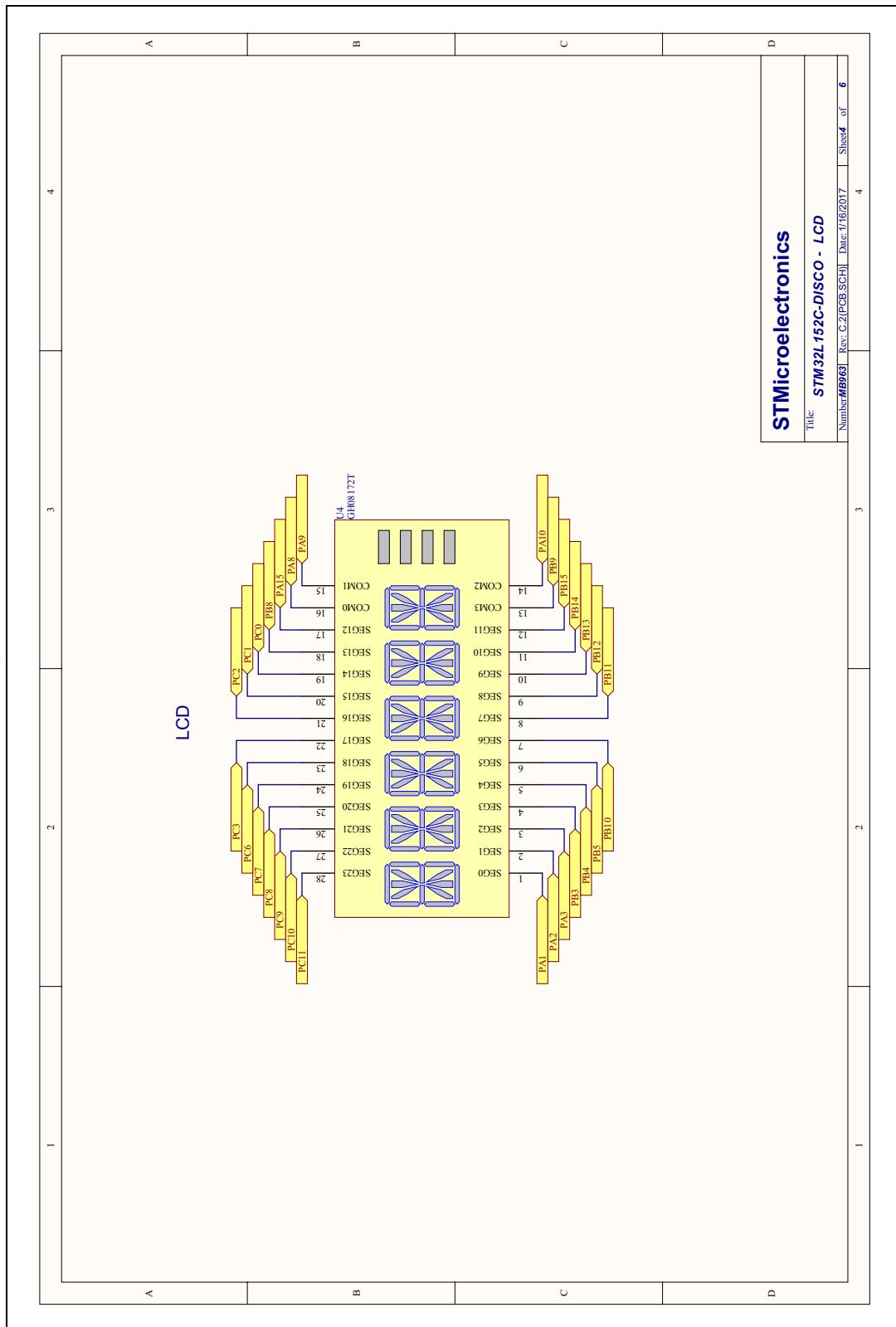
Figure 16. MCU



## Electrical schematics

UM1079

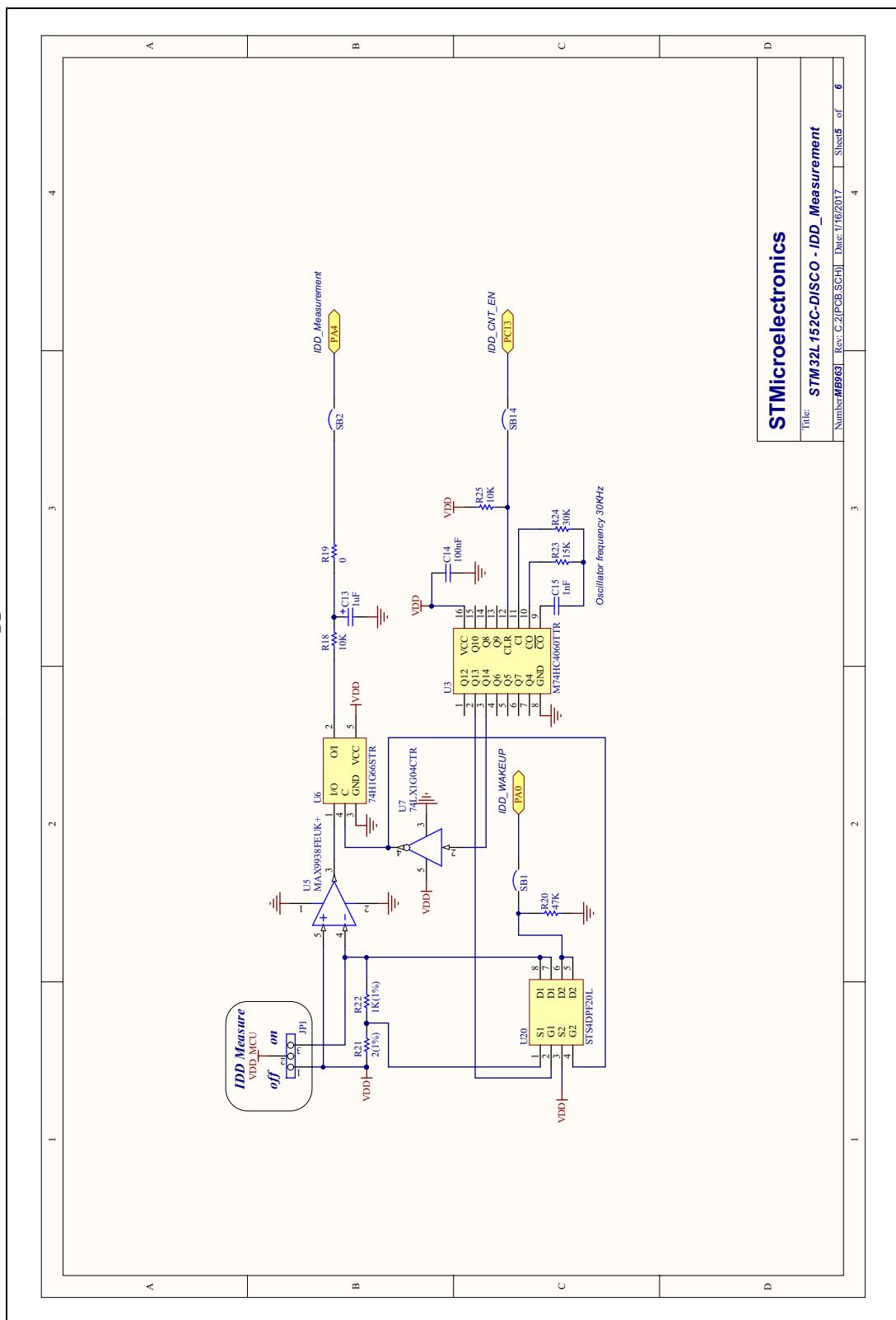
Figure 17. LCD



UM1079

Electrical schematics

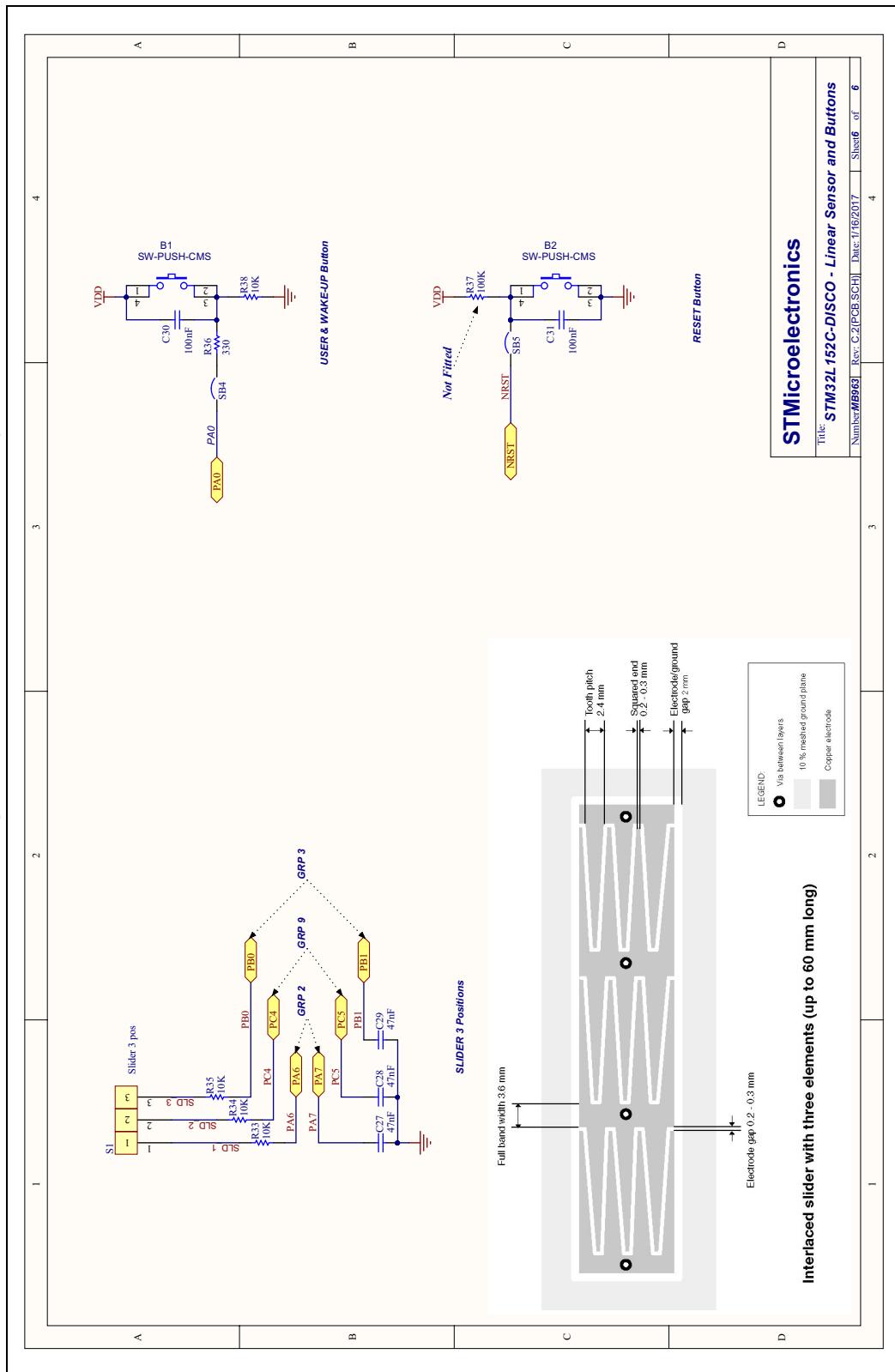
Figure 18. IDD measurement



## Electrical schematics

UM1079

Figure 19. Linear touch sensor/touchkeys



## 8 Revision history

Table 9. Document revision history

Date	Revision	Changes
10-May-2011	1	Initial release.
24-June-2011	2	Added <i>Chapter 6: Mechanical drawing</i> . Modified <i>Chapter 4.3: Power supply and power selection</i> .
19-Apr-2013	3	Added 32L152CDISCOVERY, related features. Updated STM32L-DISCOVERY url. Modified <i>Section 2.2: System requirements</i> , <i>Section 2.5: Order codes</i> , <i>Section 4.1: STM32L152RBT6 or STM32L152RCT6 microcontroller</i> , <i>Section 4.2.1: Using the ST-LINK/V2 to program/debug the STM32L on board</i> , and <i>Section 4.2.2: Using the ST-LINK/V2 to program/debug an external STM32L application</i> Updated <i>Figure 1: STM32L1 discovery board</i> , <i>Figure 2: Hardware block diagram</i> , <i>Figure 3: Top layout</i> , <i>Figure 6: STM32L152RBT6 block diagram</i> , <i>Figure 13: LCD segment mapping</i> and all schematics in <i>Section 7</i> .
23-Jan-2017	4	<ul style="list-style-type: none"><li>– Updated title.</li><li>– Updated <i>Section 4.6: Linear touch sensor / touchkeys</i>: AN2869 replaced by AN4312.</li><li>– Updated all schematics in <i>Section 7</i>.</li></ul>

**IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved



## Annexe B – Datasheet du microcontrôleur STM32L152RCT6

*Extrait du PDF [DS8890](#)*

*Pages [111 à 246](#)*



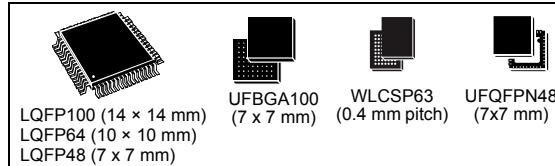
# STM32L15xCC STM32L15xRC STM32L15xUC STM32L15xVC

Ultra-low-power 32-bit MCU ARM®-based Cortex®-M3,  
256KB Flash, 32KB SRAM, 8KB EEPROM, LCD, USB, ADC, DAC

Datasheet - production data

## Features

- Ultra-low-power platform
  - 1.65 V to 3.6 V power supply
  - **-40 °C to 105 °C** temperature range
  - 0.29µA Standby mode (3 wakeup pins)
  - **1.15 µA Standby mode + RTC**
  - 0.44 µA Stop mode (16 wakeup lines)
  - 1.4 µA Stop mode + RTC
  - 8.6 µA Low-power run mode
  - 185 µA/MHz Run mode
  - 10 nA ultra-low I/O leakage
  - 8 µs wakeup time
- Core: ARM® Cortex®-M3 32-bit CPU
  - From 32 kHz up to 32 MHz max
  - 1.25 DMIPS/MHz (Dhrystone 2.1)
  - Memory protection unit
- Reset and supply management
  - Low-power, ultrasafe BOR (brownout reset) with 5 selectable thresholds
  - Ultra-low-power POR/PDR
  - Programmable voltage detector (PVD)
- Clock sources
  - 1 to 24 MHz crystal oscillator
  - 32 kHz oscillator for RTC with calibration
  - High Speed Internal 16 MHz factory-trimmed RC (+/- 1%)
  - Internal Low-power 37 kHz RC
  - Internal multispeed low-power 65 kHz to 4.2 MHz PLL for CPU clock and USB (48 MHz)
- Pre-programmed bootloader
  - USB and USART supported
- Development support
  - Serial wire debug supported
  - JTAG and trace supported
- Up to 83 fast I/Os (70 I/Os 5V tolerant), all mappable on 16 external interrupt vectors



- Memories
  - 256 Kbytes of Flash memory with ECC
  - 32 Kbytes of RAM
  - 8 Kbytes of true EEPROM with ECC
  - 128-byte backup register
- LCD Driver (except STM32L151xC devices) up to 8x40 segments, contrast adjustment, blinking mode, step-up converter
- Rich analog peripherals (down to 1.8 V)
  - 2x operational amplifiers
  - 12-bit ADC 1Msps up to 25 channels
  - 12-bit DAC 2 channels with output buffers
  - 2x ultra-low-power-comparators (window mode and wake up capability)
- DMA controller 12x channels
- 9x peripheral communication interfaces
  - 1x USB 2.0 (internal 48 MHz PLL)
  - 3x USARTs
  - Up to 8x SPIs (2x I2S, 3x 16 Mbit/s)
  - 2x I2Cs (SMBus/PMBus)
- 11x timers: 1x 32-bit, 6x 16-bit with up to 4 IC/OC/PWM channels, 2x 16-bit basic timers, 2x watchdog timers (independent and window)
- Up to 23 capacitive sensing channels
- CRC calculation unit, 96-bit unique ID

**Table 1. Device summary**

Reference	Part number
STM32L151CC	STM32L151CCT6, STM32L151CCU6
STM32L151RC <sup>(1)</sup>	STM32L151RCT6
STM32L151UC	STM32L151UCY6
STM32L151VC <sup>(1)</sup>	STM32L151VCT6, STM32L151VCH6
STM32L152CC	STM32L152CCT6, STM32L152CCU6
STM32L152RC <sup>(1)</sup>	STM32L152RCT6
STM32L152UC	STM32L152UCY6
STM32L152VC <sup>(1)</sup>	STM32L152VCT6, STM32L152VCH6

1. For sales types ending with "A" and STM32L15xxC products in WLCSP64 package, please refer to STM32L15xxC/C-A datasheet.

**Contents****STM32L151xC STM32L152xC****Contents**

<b>1</b>	<b>Introduction</b>	<b>9</b>
<b>2</b>	<b>Description</b>	<b>10</b>
2.1	Device overview	11
2.2	Ultra-low-power device continuum	12
2.2.1	Performance	12
2.2.2	Shared peripherals	12
2.2.3	Common system strategy	12
2.2.4	Features	12
<b>3</b>	<b>Functional overview</b>	<b>13</b>
3.1	Low-power modes	14
3.2	ARM® Cortex®-M3 core with MPU	18
3.3	Reset and supply management	19
3.3.1	Power supply schemes	19
3.3.2	Power supply supervisor	19
3.3.3	Voltage regulator	20
3.3.4	Boot modes	20
3.4	Clock management	21
3.5	Low-power real-time clock and backup registers	23
3.6	GPIOs (general-purpose inputs/outputs)	23
3.7	Memories	24
3.8	DMA (direct memory access)	24
3.9	LCD (liquid crystal display)	25
3.10	ADC (analog-to-digital converter)	25
3.10.1	Temperature sensor	26
3.10.2	Internal voltage reference ( $V_{REFINT}$ )	26
3.11	DAC (digital-to-analog converter)	26
3.12	Operational amplifier	27
3.13	Ultra-low-power comparators and reference voltage	27
3.14	System configuration controller and routing interface	27
3.15	Touch sensing	27

**STM32L151xC STM32L152xC****Contents**

3.16	Timers and watchdogs . . . . .	28
3.16.1	General-purpose timers (TIM2, TIM3, TIM4, TIM5, TIM9, TIM10 and TIM11) . . . . .	28
3.16.2	Basic timers (TIM6 and TIM7) . . . . .	29
3.16.3	SysTick timer . . . . .	29
3.16.4	Independent watchdog (IWDG) . . . . .	29
3.16.5	Window watchdog (WWDG) . . . . .	29
3.17	Communication interfaces . . . . .	29
3.17.1	I <sup>2</sup> C bus . . . . .	29
3.17.2	Universal synchronous/asynchronous receiver transmitter (USART) . . . . .	30
3.17.3	Serial peripheral interface (SPI) . . . . .	30
3.17.4	Inter-integrated sound (I2S) . . . . .	30
3.17.5	Universal serial bus (USB) . . . . .	30
3.18	CRC (cyclic redundancy check) calculation unit . . . . .	30
3.19	Development support . . . . .	31
3.19.1	Serial wire JTAG debug port (SWJ-DP) . . . . .	31
3.19.2	Embedded Trace Macrocell™ . . . . .	31
<b>4</b>	<b>Pin descriptions . . . . .</b>	<b>32</b>
<b>5</b>	<b>Memory mapping . . . . .</b>	<b>51</b>
<b>6</b>	<b>Electrical characteristics . . . . .</b>	<b>52</b>
6.1	Parameter conditions . . . . .	52
6.1.1	Minimum and maximum values . . . . .	52
6.1.2	Typical values . . . . .	52
6.1.3	Typical curves . . . . .	52
6.1.4	Loading capacitor . . . . .	52
6.1.5	Pin input voltage . . . . .	52
6.1.6	Power supply scheme . . . . .	53
6.1.7	Optional LCD power supply scheme . . . . .	54
6.1.8	Current consumption measurement . . . . .	54
6.2	Absolute maximum ratings . . . . .	55
6.3	Operating conditions . . . . .	56
6.3.1	General operating conditions . . . . .	56
6.3.2	Embedded reset and power control block characteristics . . . . .	57
6.3.3	Embedded internal reference voltage . . . . .	59

**Contents****STM32L151xC STM32L152xC**

6.3.4	Supply current characteristics . . . . .	60
6.3.5	Wakeup time from low-power mode . . . . .	71
6.3.6	External clock source characteristics . . . . .	72
6.3.7	Internal clock source characteristics . . . . .	77
6.3.8	PLL characteristics . . . . .	80
6.3.9	Memory characteristics . . . . .	80
6.3.10	EMC characteristics . . . . .	82
6.3.11	Electrical sensitivity characteristics . . . . .	83
6.3.12	I/O current injection characteristics . . . . .	84
6.3.13	I/O port characteristics . . . . .	85
6.3.14	NRST pin characteristics . . . . .	88
6.3.15	TIM timer characteristics . . . . .	89
6.3.16	Communications interfaces . . . . .	90
6.3.17	12-bit ADC characteristics . . . . .	98
6.3.18	DAC electrical specifications . . . . .	103
6.3.19	Operational amplifier characteristics . . . . .	105
6.3.20	Temperature sensor characteristics . . . . .	107
6.3.21	Comparator . . . . .	107
6.3.22	LCD controller . . . . .	109
<b>7</b>	<b>Package information . . . . .</b>	<b>110</b>
7.1	LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package information . . . . .	110
7.2	LQFP64, 10 x 10 mm, 64-pin low-profile quad flat package information . . . . .	113
7.3	LQFP48, 7 x 7 mm, 48-pin low-profile quad flat package information . . . . .	116
7.4	UFQFPN48 7 x 7 mm, 0.5 mm pitch, package information . . . . .	119
7.5	UFBGA100, 7 x 7 mm, 100-ball ultra thin, fine pitch ball grid array package information . . . . .	122
7.6	WL CSP63, 0.400 mm pitch wafer level chip size package information . . . . .	125
7.7	Thermal characteristics . . . . .	128
7.7.1	Reference document . . . . .	129
<b>8</b>	<b>Part numbering . . . . .</b>	<b>130</b>
<b>9</b>	<b>Revision History . . . . .</b>	<b>131</b>

## List of tables

Table 1.	Device summary . . . . .	1
Table 2.	Ultra-low-power STM32L151xC and STM32L152xC device features and peripheral counts . . . . .	11
Table 3.	Functionalities depending on the operating power supply range . . . . .	15
Table 4.	CPU frequency range depending on dynamic voltage scaling . . . . .	16
Table 5.	Functionalities depending on the working mode (from Run/active down to standby) . . . . .	17
Table 6.	$V_{LCD}$ rail decoupling . . . . .	25
Table 7.	Timer feature comparison . . . . .	28
Table 8.	Legend/abbreviations used in the pinout table . . . . .	38
Table 9.	STM32L151xC and STM32L152xC pin definitions . . . . .	38
Table 10.	Alternate function input/output . . . . .	46
Table 11.	Voltage characteristics . . . . .	55
Table 12.	Current characteristics . . . . .	55
Table 13.	Thermal characteristics . . . . .	56
Table 14.	General operating conditions . . . . .	56
Table 15.	Embedded reset and power control block characteristics . . . . .	57
Table 16.	Embedded internal reference voltage calibration values . . . . .	59
Table 17.	Embedded internal reference voltage . . . . .	59
Table 18.	Current consumption in Run mode, code with data processing running from Flash . . . . .	61
Table 19.	Current consumption in Run mode, code with data processing running from RAM . . . . .	62
Table 20.	Current consumption in Sleep mode . . . . .	63
Table 21.	Current consumption in Low-power run mode . . . . .	64
Table 22.	Current consumption in Low-power sleep mode . . . . .	65
Table 23.	Typical and maximum current consumptions in Stop mode . . . . .	66
Table 24.	Typical and maximum current consumptions in Standby mode . . . . .	68
Table 25.	Peripheral current consumption . . . . .	69
Table 26.	Low-power mode wakeup timings . . . . .	71
Table 27.	High-speed external user clock characteristics . . . . .	72
Table 28.	Low-speed external user clock characteristics . . . . .	73
Table 29.	HSE oscillator characteristics . . . . .	74
Table 30.	LSE oscillator characteristics ( $f_{LSE} = 32.768$ kHz) . . . . .	75
Table 31.	HSI oscillator characteristics . . . . .	77
Table 32.	LSI oscillator characteristics . . . . .	77
Table 33.	MSI oscillator characteristics . . . . .	78
Table 34.	PLL characteristics . . . . .	80
Table 35.	RAM and hardware registers . . . . .	80
Table 36.	Flash memory and data EEPROM characteristics . . . . .	81
Table 37.	Flash memory and data EEPROM endurance and retention . . . . .	81
Table 38.	EMS characteristics . . . . .	82
Table 39.	EMI characteristics . . . . .	83
Table 40.	ESD absolute maximum ratings . . . . .	83
Table 41.	Electrical sensitivities . . . . .	84
Table 42.	I/O current injection susceptibility . . . . .	84
Table 43.	I/O static characteristics . . . . .	85
Table 44.	Output voltage characteristics . . . . .	86
Table 45.	I/O AC characteristics . . . . .	87
Table 46.	NRST pin characteristics . . . . .	88

**List of tables****STM32L151xC STM32L152xC**

Table 47.	TIMx characteristics . . . . .	89
Table 48.	I <sup>2</sup> C characteristics . . . . .	90
Table 49.	SCL frequency ( $f_{PCLK1} = 32$ MHz, $V_{DD} = V_{DD\_I2C} = 3.3$ V) . . . . .	91
Table 50.	SPI characteristics . . . . .	92
Table 51.	USB startup time . . . . .	95
Table 52.	USB DC electrical characteristics . . . . .	95
Table 53.	USB: full speed electrical characteristics . . . . .	95
Table 54.	I2S characteristics . . . . .	96
Table 55.	ADC clock frequency . . . . .	98
Table 56.	ADC characteristics . . . . .	98
Table 57.	ADC accuracy . . . . .	100
Table 58.	Maximum source impedance $R_{AIN}$ max . . . . .	102
Table 59.	DAC characteristics . . . . .	103
Table 60.	Operational amplifier characteristics . . . . .	105
Table 61.	Temperature sensor calibration values . . . . .	107
Table 62.	Temperature sensor characteristics . . . . .	107
Table 63.	Comparator 1 characteristics . . . . .	107
Table 64.	Comparator 2 characteristics . . . . .	108
Table 65.	LCD controller characteristics . . . . .	109
Table 66.	LQPF100, 14 x 14 mm, 100-pin low-profile quad flat package mechanical data . . . . .	110
Table 67.	LQFP64, 10 x 10 mm 64-pin low-profile quad flat package mechanical data . . . . .	113
Table 68.	LQFP48, 7 x 7 mm, 48-pin low-profile quad flat package mechanical data . . . . .	117
Table 69.	UFQFPN48 – ultra thin fine pitch quad flat pack no-lead 7 x 7 mm, 0.5 mm pitch package mechanical data . . . . .	120
Table 70.	UFBGA100, 7 x 7 mm, 0.5 mm pitch package mechanical data . . . . .	122
Table 71.	UFBGA100, 7 x 7 mm, 0.50 mm pitch, recommended PCB design rules . . . . .	123
Table 72.	WLCSP63, 0.400 mm pitch wafer level chip size package mechanical data . . . . .	126
Table 73.	Thermal characteristics . . . . .	128
Table 74.	STM32L151xC and STM32L152xC ordering information scheme . . . . .	130
Table 75.	Document revision history . . . . .	131

## List of figures

Figure 1.	Ultra-low-power STM32L151xC and STM32L152xC block diagram . . . . .	13
Figure 2.	Clock tree . . . . .	22
Figure 3.	STM32L15xVC UFBGA100 ballout . . . . .	32
Figure 4.	STM32L15xVC LQFP100 pinout . . . . .	33
Figure 5.	STM32L15xRC LQFP64 pinout . . . . .	34
Figure 6.	STM32L15xUC WLCSP63 ballout . . . . .	35
Figure 7.	STM32L15xCC UFQFPN48 pinout . . . . .	36
Figure 8.	STM32L15xCC LQFP48 pinout . . . . .	37
Figure 9.	Memory map . . . . .	51
Figure 10.	Pin loading conditions . . . . .	52
Figure 11.	Pin input voltage . . . . .	52
Figure 12.	Power supply scheme . . . . .	53
Figure 13.	Optional LCD power supply scheme . . . . .	54
Figure 14.	Current consumption measurement scheme . . . . .	54
Figure 15.	High-speed external clock source AC timing diagram . . . . .	72
Figure 16.	Low-speed external clock source AC timing diagram . . . . .	73
Figure 17.	HSE oscillator circuit diagram . . . . .	75
Figure 18.	Typical application with a 32.768 kHz crystal . . . . .	76
Figure 19.	I/O AC characteristics definition . . . . .	88
Figure 20.	Recommended NRST pin protection . . . . .	89
Figure 21.	I <sup>2</sup> C bus AC waveforms and measurement circuit . . . . .	91
Figure 22.	SPI timing diagram - slave mode and CPHA = 0 . . . . .	93
Figure 23.	SPI timing diagram - slave mode and CPHA = 1 <sup>(1)</sup> . . . . .	93
Figure 24.	SPI timing diagram - master mode <sup>(1)</sup> . . . . .	94
Figure 25.	USB timings: definition of data signal rise and fall time . . . . .	95
Figure 26.	I <sup>2</sup> S slave timing diagram (Philips protocol) <sup>(1)</sup> . . . . .	97
Figure 27.	I <sup>2</sup> S master timing diagram (Philips protocol) <sup>(1)</sup> . . . . .	97
Figure 28.	ADC accuracy characteristics . . . . .	101
Figure 29.	Typical connection diagram using the ADC . . . . .	101
Figure 30.	Maximum dynamic current consumption on V <sub>REF+</sub> supply pin during ADC conversion . . . . .	102
Figure 31.	12-bit buffered /non-buffered DAC . . . . .	105
Figure 32.	LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package outline . . . . .	110
Figure 33.	LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package recommended footprint . . . . .	111
Figure 34.	LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package top view example . . . . .	112
Figure 35.	LQFP64, 10 x 10 mm, 64-pin low-profile quad flat package outline . . . . .	113
Figure 36.	LQFP64, 10 x 10 mm, 64-pin low-profile quad flat package recommended footprint . . . . .	114
Figure 37.	LQFP64 10 x 10 mm, 64-pin low-profile quad flat package top view example . . . . .	115
Figure 38.	LQFP48, 7 x 7 mm, 48-pin low-profile quad flat package outline . . . . .	116
Figure 39.	LQFP48 recommended footprint . . . . .	117
Figure 40.	LQFP48 package top view example . . . . .	118
Figure 41.	UFQFPN48 7 x 7 mm, 0.5 mm pitch, package outline . . . . .	119
Figure 42.	UFQFPN48 recommended footprint . . . . .	120
Figure 43.	UFQFPN48 package top view example . . . . .	121
Figure 44.	UFBGA100, 7 x 7 mm, 0.5 mm pitch package outline . . . . .	122
Figure 45.	UFBGA100, 7 x 7 mm, 0.5 mm pitch, package recommended footprint . . . . .	123

**List of figures****STM32L151xC STM32L152xC**

---

Figure 46. UFBGA100, 7 x 7 mm, 0.5 mm pitch, package top view example . . . . .	124
Figure 47. WLCSP63, 0.400 mm pitch wafer level chip size package outline . . . . .	125
Figure 48. WLCSP63 device marking example . . . . .	127
Figure 49. Thermal resistance suffix 6 . . . . .	129
Figure 50. Thermal resistance suffix 7 . . . . .	129

## 1 Introduction

This datasheet provides the ordering information and mechanical device characteristics of the STM32L151xC and STM32L152xC ultra-low-power ARM® Cortex®-M3 based microcontroller product line with a Flash memory of 256 Kbytes.

The ultra-low-power STM32L151xC and STM32L152xC family includes devices in 6 different package types: from 48 pins to 100 pins. Depending on the device chosen, different sets of peripherals are included, the description below gives an overview of the complete range of peripherals proposed in this family.

These features make the ultra-low-power STM32L151xC and STM32L152xC microcontroller family suitable for a wide range of applications:

- Medical and handheld equipment
- Application control and user interface
- PC peripherals, gaming, GPS and sport equipment
- Alarm systems, wired and wireless sensors, video intercom
- Utility metering

This STM32L151xC and STM32L152xC datasheet should be read in conjunction with the STM32L1xxxx reference manual (RM0038). The application note “Getting started with STM32L1xxxx hardware development” (AN3216) gives a hardware implementation overview. Both documents are available from the STMicroelectronics website [www.st.com](http://www.st.com).

For information on the ARM® Cortex®-M3 core please refer to the ARM® Cortex®-M3 technical reference manual, available from the [www.arm.com](http://www.arm.com) website. *Figure 1* shows the general block diagram of the device family.

**Description****STM32L151xC STM32L152xC**

## 2 Description

The ultra-low-power STM32L151xC and STM32L152xC devices incorporate the connectivity power of the universal serial bus (USB) with the high-performance ARM® Cortex®-M3 32-bit RISC core operating at a frequency of 32 MHz (33.3 DMIPS), a memory protection unit (MPU), high-speed embedded memories (Flash memory up to 256 Kbytes and RAM up to 32 Kbytes) and an extensive range of enhanced I/Os and peripherals connected to two APB buses.

The STM32L151xC and STM32L152xC devices offer two operational amplifiers, one 12-bit ADC, two DACs, two ultra-low-power comparators, one general-purpose 32-bit timer, six general-purpose 16-bit timers and two basic timers, which can be used as time bases.

Moreover, the STM32L151xC and STM32L152xC devices contain standard and advanced communication interfaces: up to two I2Cs, three SPIs, two I2S, three USARTs and an USB. The STM32L151xC and STM32L152xC devices offer up to 23 capacitive sensing channels to simply add a touch sensing functionality to any application.

They also include a real-time clock and a set of backup registers that remain powered in Standby mode.

Finally, the integrated LCD controller (except STM32L151xC devices) has a built-in LCD voltage generator that allows to drive up to 8 multiplexed LCDs with the contrast independent of the supply voltage.

The ultra-low-power STM32L151xC and STM32L152xC devices operate from a 1.8 to 3.6 V power supply (down to 1.65 V at power down) with BOR and from a 1.65 to 3.6 V power supply without BOR option. They are available in the -40 to +85 °C and -40 to +105 °C temperature ranges. A comprehensive set of power-saving modes allows the design of low-power applications.



STM32L151xC STM32L152xC	Description
-------------------------	-------------

## 2.1 Device overview

Table 2. Ultra-low-power STM32L151xC and STM32L152xC device features and peripheral counts

Peripheral	STM32L15xCC	STM32L15xUC STM32L15xRC	STM32L15xVC
<b>Flash (Kbytes)</b>	256		
<b>Data EEPROM (Kbytes)</b>	8		
<b>RAM (Kbytes)</b>	32		
<b>Timers</b>	<b>32 bit</b>	1	
	<b>General-purpose</b>	6	
	<b>Basic</b>	2	
<b>Communication interfaces</b>	<b>SPI</b>	8(3) <sup>(1)</sup>	
	<b>I<sup>2</sup>S</b>	2	
	<b>I<sup>2</sup>C</b>	2	
	<b>USART</b>	3	
	<b>USB</b>	1	
<b>GPIOs</b>	37	51	83
<b>Operation amplifiers</b>		2	
<b>12-bit synchronized ADC</b> <b>Number of channels</b>	1 14	1 21	1 25
<b>12-bit DAC</b> <b>Number of channels</b>		2 2	
<b>LCD<sup>(2)</sup></b> <b>COM x SEG</b>	1 4x18	1 4x32 or 8x28	1 4x44 or 8x40
<b>Comparators</b>		2	
<b>Capacitive sensing channels</b>	16		23
<b>Max. CPU frequency</b>		32 MHz	
<b>Operating voltage</b>	1.8 V to 3.6 V (down to 1.65 V at power-down) with BOR option 1.65 V to 3.6 V without BOR option		
<b>Operating temperatures</b>	Ambient operating temperature: -40 °C to 85 °C / -40 °C to 105 °C Junction temperature: -40 to + 110 °C		
<b>Packages</b>	LQFP48, UFQFPN48	LQFP64, WLCSP63	LQFP100, UFBGA100

1. 5 SPIs are USART configured in synchronous mode emulating SPI master.

2. STM32L152xx devices only.

Description	STM32L151xC STM32L152xC
-------------	-------------------------

## 2.2 Ultra-low-power device continuum

The ultra-low-power family offers a large choice of cores and features. From proprietary 8-bit to up to Cortex-M3, including the Cortex-M0+, the STM32Lx series are the best choice to answer the user needs, in terms of ultra-low-power features. The STM32 ultra-low-power series are the best fit, for instance, for gas/water meter, keyboard/mouse or fitness and healthcare, wearable applications. Several built-in features like LCD drivers, dual-bank memory, Low-power run mode, op-amp, AES 128-bit, DAC, USB crystal-less and many others will clearly allow to build very cost-optimized applications by reducing BOM.

**Note:** *STMicroelectronics as a reliable and long-term manufacturer ensures as much as possible the pin-to-pin compatibility between any STM8Lxxxx and STM32Lxxxx devices and between any of the STM32Lx and STM32Fx series. Thanks to this unprecedented scalability, the old applications can be upgraded to respond to the latest market features and efficiency demand.*

### 2.2.1 Performance

All the families incorporate highly energy-efficient cores with both Harvard architecture and pipelined execution: advanced STM8 core for STM8L families and ARM Cortex-M3 core for STM32L family. In addition specific care for the design architecture has been taken to optimize the mA/DMIPS and mA/MHz ratios.

This allows the ultra-low-power performance to range from 5 up to 33.3 DMIPS.

### 2.2.2 Shared peripherals

STM8L15xx, STM32L15xxx and STM32L162xx share identical peripherals which ensure a very easy migration from one family to another:

- Analog peripherals: ADC, DAC and comparators
- Digital peripherals: RTC and some communication interfaces

### 2.2.3 Common system strategy.

To offer flexibility and optimize performance, the STM8L15xxx, STM32L15xxx and STM32L162xx family uses a common architecture:

- Same power supply range from 1.65 V to 3.6 V
- Architecture optimized to reach ultra-low consumption both in low-power modes and Run mode
- Fast startup strategy from low-power modes
- Flexible system clock
- Ultrasafe reset: same reset strategy including power-on reset, power-down reset, brownout reset and programmable voltage detector

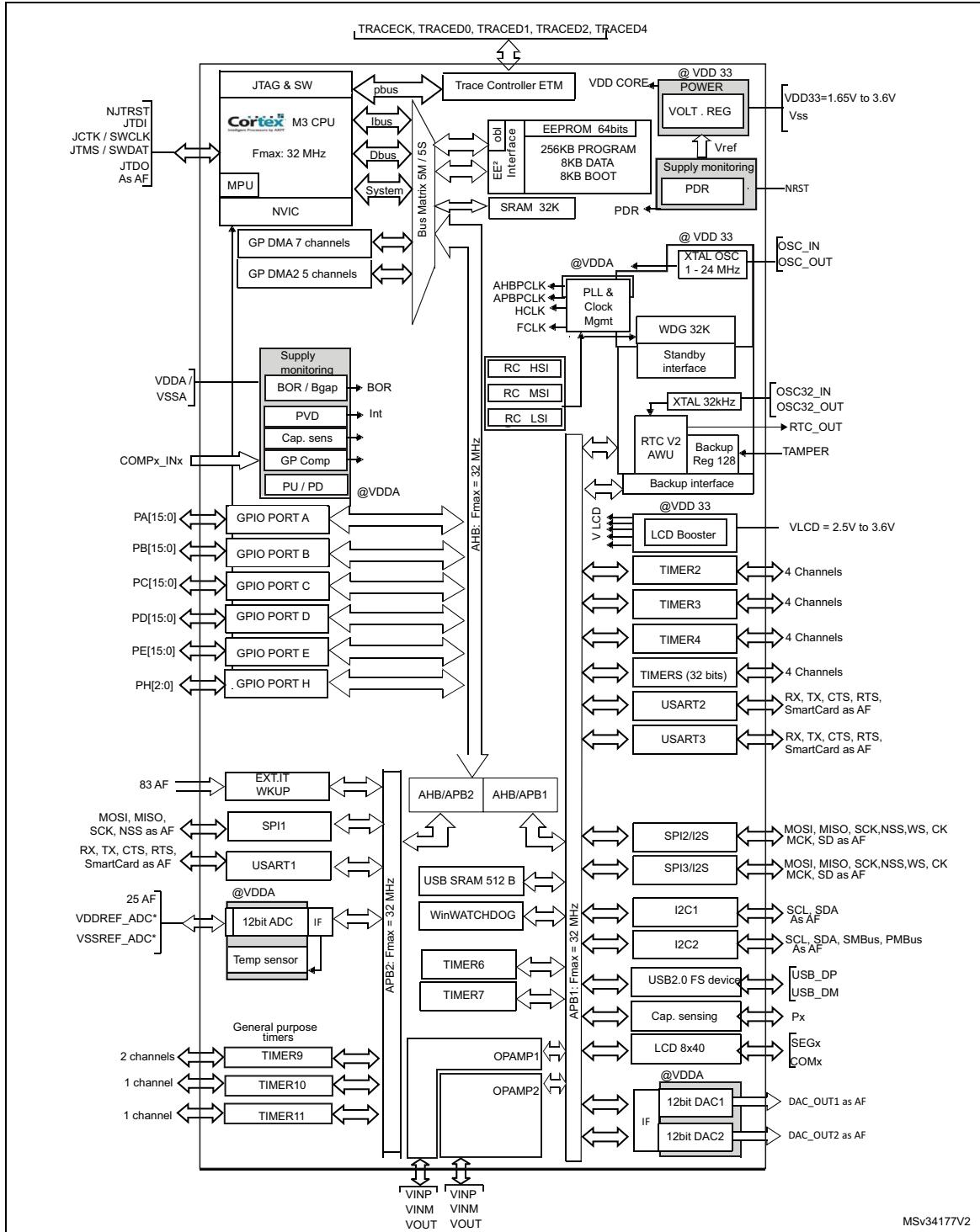
### 2.2.4 Features

ST ultra-low-power continuum also lies in feature compatibility:

- More than 15 packages with pin count from 20 to 144 pins and size down to 3 x 3 mm
- Memory density ranging from 2 to 512 Kbytes

### 3 Functional overview

**Figure 1. Ultra-low-power STM32L151xC and STM32L152xC block diagram**



---

**Functional overview****STM32L151xC STM32L152xC**

---

### 3.1 Low-power modes

The ultra-low-power STM32L151xC and STM32L152xC devices support dynamic voltage scaling to optimize its power consumption in run mode. The voltage from the internal low-drop regulator that supplies the logic can be adjusted according to the system's maximum operating frequency and the external voltage supply.

There are three power consumption ranges:

- Range 1 ( $V_{DD}$  range limited to 1.71 V - 3.6 V), with the CPU running at up to 32 MHz
- Range 2 (full  $V_{DD}$  range), with a maximum CPU frequency of 16 MHz
- Range 3 (full  $V_{DD}$  range), with a maximum CPU frequency limited to 4 MHz (generated only with the multispeed internal RC oscillator clock source)

Seven low-power modes are provided to achieve the best compromise between low-power consumption, short startup time and available wakeup sources:

- **Sleep mode**

In Sleep mode, only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs. Sleep mode power consumption at 16 MHz is about 1 mA with all peripherals off.

- **Low-power run mode**

This mode is achieved with the multispeed internal (MSI) RC oscillator set to the MSI range 0 or MSI range 1 clock range (maximum 131 kHz), execution from SRAM or Flash memory, and internal regulator in low-power mode to minimize the regulator's operating current. In low-power run mode, the clock frequency and the number of enabled peripherals are both limited.

- **Low-power sleep mode**

This mode is achieved by entering Sleep mode with the internal voltage regulator in Low-power mode to minimize the regulator's operating current. In Low-power sleep mode, both the clock frequency and the number of enabled peripherals are limited; a typical example would be to have a timer running at 32 kHz.

When wakeup is triggered by an event or an interrupt, the system reverts to the run mode with the regulator on.

- **Stop mode with RTC**

Stop mode achieves the lowest power consumption while retaining the RAM and register contents and real time clock. All clocks in the  $V_{CORE}$  domain are stopped, the PLL, MSI RC, HSI RC and HSE crystal oscillators are disabled. The LSE or LSI is still running. The voltage regulator is in the low-power mode.

The device can be woken up from Stop mode by any of the EXTI line, in 8  $\mu$ s. The EXTI line source can be one of the 16 external lines. It can be the PVD output, the Comparator 1 event or Comparator 2 event (if internal reference voltage is on), it can be the RTC alarm(s), the USB wakeup, the RTC tamper events, the RTC timestamp event or the RTC wakeup.

**STM32L151xC STM32L152xC****Functional overview**

- **Stop mode without RTC**

Stop mode achieves the lowest power consumption while retaining the RAM and register contents. All clocks are stopped, the PLL, MSI RC, HSI and LSI RC, LSE and HSE crystal oscillators are disabled. The voltage regulator is in the low-power mode. The device can be woken up from Stop mode by any of the EXTI line, in 8 µs. The EXTI line source can be one of the 16 external lines. It can be the PVD output, the Comparator 1 event or Comparator 2 event (if internal reference voltage is on). It can also be wakened by the USB wakeup.

- **Standby mode with RTC**

Standby mode is used to achieve the lowest power consumption and real time clock. The internal voltage regulator is switched off so that the entire  $V_{CORE}$  domain is powered off. The PLL, MSI RC, HSI RC and HSE crystal oscillators are also switched off. The LSE or LSI is still running. After entering Standby mode, the RAM and register contents are lost except for registers in the Standby circuitry (wakeup logic, IWDG, RTC, LSI, LSE Crystal 32K osc, RCC\_CSR).

The device exits Standby mode in 60 µs when an external reset (NRST pin), an IWDG reset, a rising edge on one of the three WKUP pins, RTC alarm (Alarm A or Alarm B), RTC tamper event, RTC timestamp event or RTC Wakeup event occurs.

- **Standby mode without RTC**

Standby mode is used to achieve the lowest power consumption. The internal voltage regulator is switched off so that the entire  $V_{CORE}$  domain is powered off. The PLL, MSI RC, HSI and LSI RC, HSE and LSE crystal oscillators are also switched off. After entering Standby mode, the RAM and register contents are lost except for registers in the Standby circuitry (wakeup logic, IWDG, RTC, LSI, LSE Crystal 32K osc, RCC\_CSR).

The device exits Standby mode in 60 µs when an external reset (NRST pin) or a rising edge on one of the three WKUP pin occurs.

*Note:* The RTC, the IWDG, and the corresponding clock sources are not stopped automatically by entering Stop or Standby mode.

**Table 3. Functionalities depending on the operating power supply range**

Operating power supply range	Functionalities depending on the operating power supply range <sup>(1)</sup>		
	DAC and ADC operation	USB	Dynamic voltage scaling range
$V_{DD} = V_{DDA} = 1.65$ to 1.71 V	Not functional	Not functional	Range 2 or Range 3
$V_{DD} = V_{DDA} = 1.71$ to 1.8 V <sup>(2)</sup>	Not functional	Not functional	Range 1, Range 2 or Range 3
$V_{DD} = V_{DDA} = 1.8$ to 2.0 V <sup>(2)</sup>	Conversion time up to 500 Ksps	Not functional	Range 1, Range 2 or Range 3

**Functional overview****STM32L151xC STM32L152xC****Table 3. Functionalities depending on the operating power supply range (continued)**

Functionalities depending on the operating power supply range <sup>(1)</sup>			
Operating power supply range	DAC and ADC operation	USB	Dynamic voltage scaling range
$V_{DD}=V_{DDA} = 2.0 \text{ to } 2.4 \text{ V}$	Conversion time up to 500 Ksps	Functional <sup>(3)</sup>	Range 1, Range 2 or Range 3
$V_{DD}=V_{DDA} = 2.4 \text{ to } 3.6 \text{ V}$	Conversion time up to 1 Msps	Functional <sup>(3)</sup>	Range 1, Range 2 or Range 3

1. The GPIO speed also depends from VDD voltage and the user has to refer to [Table 45: I/O AC characteristics](#) for more information about I/O speed.
2. CPU frequency changes from initial to final must respect " $F_{CPU \text{ initial}} < 4 * F_{CPU \text{ final}}$ " to limit  $V_{CORE}$  drop due to current consumption peak when frequency increases. It must also respect 5  $\mu\text{s}$  delay between two changes. For example to switch from 4.2 MHz to 32 MHz, the user can switch from 4.2 MHz to 16 MHz, wait 5  $\mu\text{s}$ , then switch from 16 MHz to 32 MHz.
3. Should be USB compliant from I/O voltage standpoint, the minimum  $V_{DD}$  is 3.0 V.

**Table 4. CPU frequency range depending on dynamic voltage scaling**

CPU frequency range	Dynamic voltage scaling range
16 MHz to 32 MHz (1ws) 32 kHz to 16 MHz (0ws)	Range 1
8 MHz to 16 MHz (1ws) 32 kHz to 8 MHz (0ws)	Range 2
2.1MHz to 4.2 MHz (1ws) 32 kHz to 2.1 MHz (0ws)	Range 3

## STM32L151xC STM32L152xC

## Functional overview

**Table 5. Functionalities depending on the working mode (from Run/active down to standby)**

Ips	Run/Active	Sleep	Low-power Run	Low-power Sleep	Stop		Standby	Wakeup capability
						Wakeup capability		
CPU	Y	--	Y	--	--	--	--	--
Flash	Y	Y	Y	Y	--	--	--	--
RAM	Y	Y	Y	Y	Y	--	--	--
Backup Registers	Y	Y	Y	Y	Y	--	Y	--
EEPROM	Y	Y	Y	Y	Y	--	--	--
Brown-out rest (BOR)	Y	Y	Y	Y	Y	Y	Y	--
DMA	Y	Y	Y	Y	--	--	--	--
Programmable Voltage Detector (PVD)	Y	Y	Y	Y	Y	Y	Y	--
Power On Reset (POR)	Y	Y	Y	Y	Y	Y	Y	--
Power Down Rest (PDR)	Y	Y	Y	Y	Y	--	Y	--
High Speed Internal (HSI)	Y	Y	--	--	--	--	--	--
High Speed External (HSE)	Y	Y	--	--	--	--	--	--
Low Speed Internal (LSI)	Y	Y	Y	Y	Y	--	Y	--
Low Speed External (LSE)	Y	Y	Y	Y	Y	--	Y	--
Multi-Speed Internal (MSI)	Y	Y	Y	Y	--	--	--	--
Inter-Connect Controller	Y	Y	Y	Y	--	--	--	--
RTC	Y	Y	Y	Y	Y	Y	Y	--
RTC Tamper	Y	Y	Y	Y	Y	Y	Y	Y
Auto WakeUp (AWU)	Y	Y	Y	Y	Y	Y	Y	Y
LCD	Y	Y	Y	Y	Y	--	--	--
USB	Y	Y	--	--	--	Y	--	--
USART	Y	Y	Y	Y	Y	(1)	--	--
SPI	Y	Y	Y	Y	--	--	--	--
I2C	Y	Y	--	--	--	(1)	--	--

## Functional overview

## STM32L151xC STM32L152xC

**Table 5. Functionalities depending on the working mode (from Run/active down to standby) (continued)**

Ips	Run/Active	Sleep	Low-power Run	Low-power Sleep	Stop		Standby
					Wakeup capability	Wakeup capability	
ADC	Y	Y	--	--	--	--	--
DAC	Y	Y	Y	Y	Y	--	--
Tempsensor	Y	Y	Y	Y	Y	--	--
OP amp	Y	Y	Y	Y	Y	--	--
Comparators	Y	Y	Y	Y	Y	--	--
16-bit and 32-bit Timers	Y	Y	Y	Y	--	--	--
IWDG	Y	Y	Y	Y	Y	Y	Y
WWDG	Y	Y	Y	Y	--	--	--
Touch sensing	Y	Y	--	--	--	--	--
Systic Timer	Y	Y	Y	Y		--	--
GPIOs	Y	Y	Y	Y	Y	Y	-- 3 pins
Wakeup time to Run mode	0 µs	0.4 µs	3 µs	46 µs	< 8 µs	58 µs	
Consumption V <sub>DD</sub> =1.8 to 3.6 V (Typ)	Down to 185 µA/MHz (from Flash)	Down to 34.5 µA/MHz (from Flash)	Down to 8.6 µA	Down to 4.4 µA	0.43 µA (no RTC) V <sub>DD</sub> =1.8V	0.29 µA (no RTC) V <sub>DD</sub> =1.8V	
					1.15 µA (with RTC) V <sub>DD</sub> =1.8V	0.9 µA (with RTC) V <sub>DD</sub> =1.8V	
					0.44 µA (no RTC) V <sub>DD</sub> =3.0V	0.29 µA (no RTC) V <sub>DD</sub> =3.0V	
					1.4 µA (with RTC) V <sub>DD</sub> =3.0V	1.15 µA (with RTC) V <sub>DD</sub> =3.0V	

1. The startup on communication line wakes the CPU which was made possible by an EXTI, this induces a delay before entering run mode.

### 3.2 ARM® Cortex®-M3 core with MPU

The ARM® Cortex®-M3 processor is the industry leading processor for embedded systems. It has been developed to provide a low-cost platform that meets the needs of MCU implementation, with a reduced pin count and low-power consumption, while delivering outstanding computational performance and an advanced system response to interrupts.

The ARM® Cortex®-M3 32-bit RISC processor features exceptional code-efficiency, delivering the high-performance expected from an ARM core in the memory size usually associated with 8- and 16-bit devices.

**STM32L151xC STM32L152xC****Functional overview**

The memory protection unit (MPU) improves system reliability by defining the memory attributes (such as read/write access permissions) for different memory regions. It provides up to eight different regions and an optional predefined background region.

Owing to its embedded ARM core, the STM32L151xC and STM32L152xC devices are compatible with all ARM tools and software.

#### **Nested vectored interrupt controller (NVIC)**

The ultra-low-power STM32L151xC and STM32L152xC devices embed a nested vectored interrupt controller able to handle up to 53 maskable interrupt channels (not including the 16 interrupt lines of ARM® Cortex®-M3) and 16 priority levels.

- Closely coupled NVIC gives low-latency interrupt processing
- Interrupt entry vector table address passed directly to the core
- Closely coupled NVIC core interface
- Allows early processing of interrupts
- Processing of *late arriving*, higher-priority interrupts
- Support for tail-chaining
- Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead

This hardware block provides flexible interrupt management features with minimal interrupt latency.

### **3.3 Reset and supply management**

#### **3.3.1 Power supply schemes**

- $V_{DD} = 1.65$  to  $3.6$  V: external power supply for I/Os and the internal regulator. Provided externally through  $V_{DD}$  pins.
- $V_{SSA}, V_{DDA} = 1.65$  to  $3.6$  V: external analog power supplies for ADC, reset blocks, RCs and PLL (minimum voltage to be applied to  $V_{DDA}$  is  $1.8$  V when the ADC is used).  $V_{DDA}$  and  $V_{SSA}$  must be connected to  $V_{DD}$  and  $V_{SS}$ , respectively.

#### **3.3.2 Power supply supervisor**

The device has an integrated ZEROPOWER power-on reset (POR)/power-down reset (PDR) that can be coupled with a brownout reset (BOR) circuitry.

The device exists in two versions:

- The version with BOR activated at power-on operates between  $1.8$  V and  $3.6$  V.
- The other version without BOR operates between  $1.65$  V and  $3.6$  V.

After the  $V_{DD}$  threshold is reached ( $1.65$  V or  $1.8$  V depending on the BOR which is active or not at power-on), the option byte loading process starts, either to confirm or modify default thresholds, or to disable the BOR permanently: in this case, the  $V_{DD}$  min value becomes  $1.65$  V (whatever the version, BOR active or not, at power-on).

When BOR is active at power-on, it ensures proper operation starting from  $1.8$  V whatever the power ramp-up phase before it reaches  $1.8$  V. When BOR is not active at power-up, the power ramp-up should guarantee that  $1.65$  V is reached on  $V_{DD}$  at least  $1$  ms after it exits the POR area.

**Functional overview****STM32L151xC STM32L152xC**

Five BOR thresholds are available through option bytes, starting from 1.8 V to 3 V. To reduce the power consumption in Stop mode, it is possible to automatically switch off the internal reference voltage ( $V_{REFINT}$ ) in Stop mode. The device remains in reset mode when  $V_{DD}$  is below a specified threshold,  $V_{POR/PDR}$  or  $V_{BOR}$ , without the need for any external reset circuit.

*Note:* *The start-up time at power-on is typically 3.3 ms when BOR is active at power-up, the start-up time at power-on can be decreased down to 1 ms typically for devices with BOR inactive at power-up.*

The device features an embedded programmable voltage detector (PVD) that monitors the  $V_{DD}/V_{DDA}$  power supply and compares it to the  $V_{PVD}$  threshold. This PVD offers 7 different levels between 1.85 V and 3.05 V, chosen by software, with a step around 200 mV. An interrupt can be generated when  $V_{DD}/V_{DDA}$  drops below the  $V_{PVD}$  threshold and/or when  $V_{DD}/V_{DDA}$  is higher than the  $V_{PVD}$  threshold. The interrupt service routine can then generate a warning message and/or put the MCU into a safe state. The PVD is enabled by software.

### 3.3.3 Voltage regulator

The regulator has three operation modes: main (MR), low-power (LPR) and power down.

- MR is used in Run mode (nominal regulation)
- LPR is used in the Low-power run, Low-power sleep and Stop modes
- Power down is used in Standby mode. The regulator output is high impedance, the kernel circuitry is powered down, inducing zero consumption but the contents of the registers and RAM are lost except for the standby circuitry (wakeup logic, IWDG, RTC, LSI, LSE crystal 32K osc, RCC\_CSR).

### 3.3.4 Boot modes

At startup, boot pins are used to select one of three boot options:

- Boot from Flash memory
- Boot from System memory
- Boot from embedded RAM

The boot loader is located in System memory. It is used to reprogram the Flash memory by using USART1, USART2 or USB. See Application note “STM32 microcontroller system memory boot mode” (AN2606) for details.

### 3.4 Clock management

The clock controller distributes the clocks coming from different oscillators to the core and the peripherals. It also manages clock gating for low-power modes and ensures clock robustness. It features:

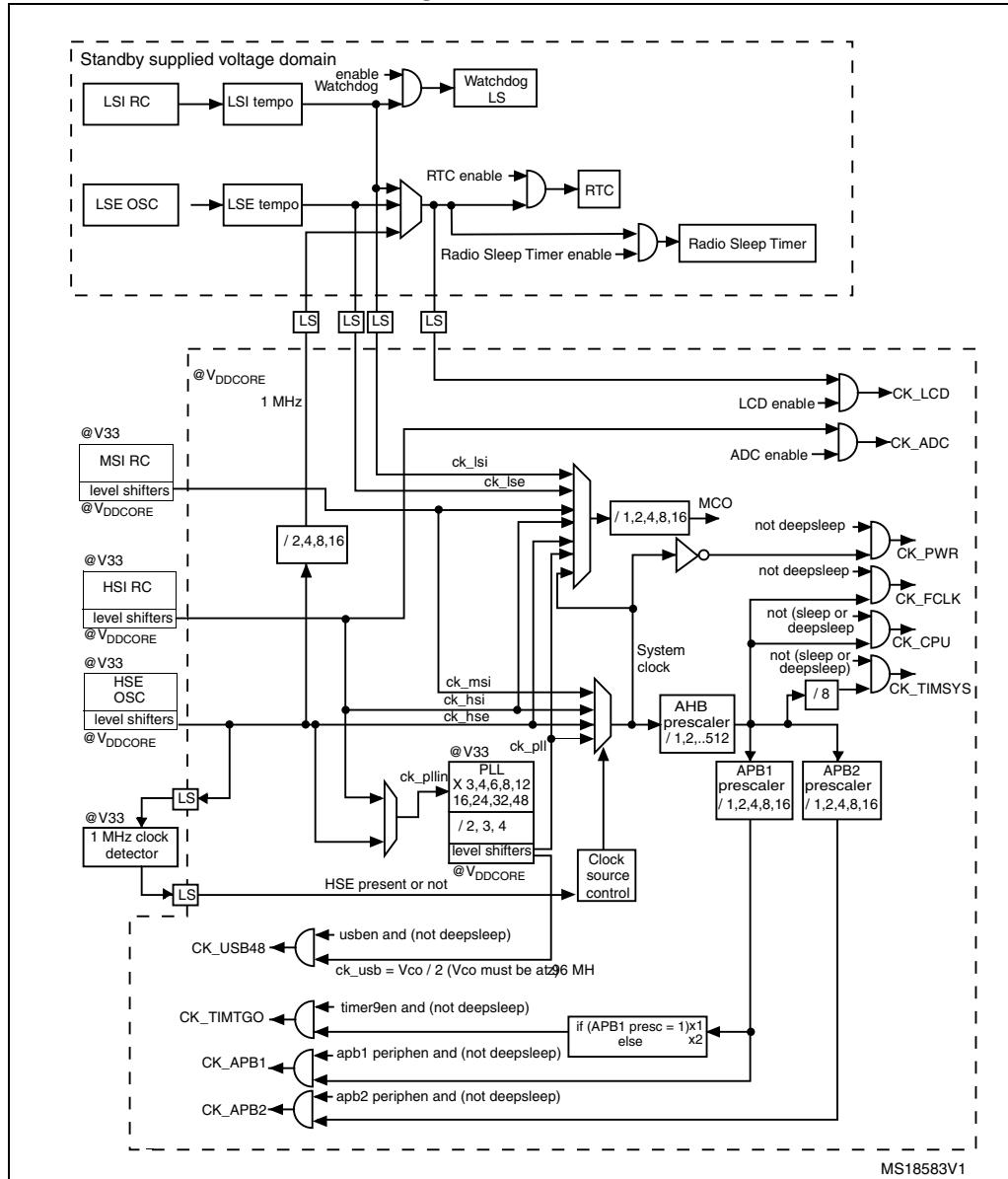
- **Clock prescaler:** to get the best trade-off between speed and current consumption, the clock frequency to the CPU and peripherals can be adjusted by a programmable prescaler.
- **Safe clock switching:** clock sources can be changed safely on the fly in run mode through a configuration register.
- **Clock management:** to reduce power consumption, the clock controller can stop the clock to the core, individual peripherals or memory.
- **System clock source:** three different clock sources can be used to drive the master clock SYSCLK:
  - 1-24 MHz high-speed external crystal (HSE), that can supply a PLL
  - 16 MHz high-speed internal RC oscillator (HSI), trimmable by software, that can supply a PLL
  - Multispeed internal RC oscillator (MSI), trimmable by software, able to generate 7 frequencies (65 kHz, 131 kHz, 262 kHz, 524 kHz, 1.05 MHz, 2.1 MHz, 4.2 MHz). When a 32.768 kHz clock source is available in the system (LSE), the MSI frequency can be trimmed by software down to a  $\pm 0.5\%$  accuracy.
- **Auxiliary clock source:** two ultra-low-power clock sources that can be used to drive the LCD controller and the real-time clock:
  - 32.768 kHz low-speed external crystal (LSE)
  - 37 kHz low-speed internal RC (LSI), also used to drive the independent watchdog. The LSI clock can be measured using the high-speed internal RC oscillator for greater precision.
- **RTC and LCD clock sources:** the LSI, LSE or HSE sources can be chosen to clock the RTC and the LCD, whatever the system clock.
- **USB clock source:** the embedded PLL has a dedicated 48 MHz clock output to supply the USB interface.
- **Startup clock:** after reset, the microcontroller restarts by default with an internal 2 MHz clock (MSI). The prescaler ratio and clock source can be changed by the application program as soon as the code execution starts.
- **Clock security system (CSS):** this feature can be enabled by software. If a HSE clock failure occurs, the master clock is automatically switched to HSI and a software interrupt is generated if enabled.
- **Clock-out capability (MCO: microcontroller clock output):** it outputs one of the internal clocks for external use by the application.

Several prescalers allow the configuration of the AHB frequency, each APB (APB1 and APB2) domains. The maximum frequency of the AHB and the APB domains is 32 MHz. See [Figure 2](#) for details on the clock tree.

## Functional overview

## STM32L151xC STM32L152xC

Figure 2. Clock tree



MS18583V1

### 3.5 Low-power real-time clock and backup registers

The real-time clock (RTC) is an independent BCD timer/counter. Dedicated registers contain the sub-second, second, minute, hour (12/24 hour), week day, date, month, year, in BCD (binary-coded decimal) format. Correction for 28, 29 (leap year), 30, and 31 day of the month are made automatically. The RTC provides two programmable alarms and programmable periodic interrupts with wakeup from Stop and Standby modes.

The programmable wakeup time ranges from 120 µs to 36 hours.

The RTC can be calibrated with an external 512 Hz output, and a digital compensation circuit helps reduce drift due to crystal deviation.

The RTC can also be automatically corrected with a 50/60Hz stable powerline.

The RTC calendar can be updated on the fly down to sub second precision, which enables network system synchronization.

A time stamp can record an external event occurrence, and generates an interrupt.

There are thirty-two 32-bit backup registers provided to store 128 bytes of user application data. They are cleared in case of tamper detection.

Three pins can be used to detect tamper events. A change on one of these pins can reset backup register and generate an interrupt. To prevent false tamper event, like ESD event, these three tamper inputs can be digitally filtered.

### 3.6 GPIOs (general-purpose inputs/outputs)

Each of the GPIO pins can be configured by software as output (push-pull or open-drain), as input (with or without pull-up or pull-down) or as peripheral alternate function. Most of the GPIO pins are shared with digital or analog alternate functions, and can be individually remapped using dedicated AFIO registers. All GPIOs are high current capable. The alternate function configuration of I/Os can be locked if needed following a specific sequence in order to avoid spurious writing to the I/O registers. The I/O controller is connected to the AHB with a toggling speed of up to 16 MHz.

#### External interrupt/event controller (EXTI)

The external interrupt/event controller consists of 24 edge detector lines used to generate interrupt/event requests. Each line can be individually configured to select the trigger event (rising edge, falling edge, both) and can be masked independently. A pending register maintains the status of the interrupt requests. The EXTI can detect an external line with a pulse width shorter than the Internal APB2 clock period. Up to 83 GPIOs can be connected to the 16 external interrupt lines. The 8 other lines are connected to RTC, PVD, USB, comparator events or capacitive sensing acquisition.

**Functional overview****STM32L151xC STM32L152xC**

### 3.7 Memories

The STM32L151xC and STM32L152xC devices have the following features:

- 32 Kbytes of embedded RAM accessed (read/write) at CPU clock speed with 0 wait states. With the enhanced bus matrix, operating the RAM does not lead to any performance penalty during accesses to the system bus (AHB and APB buses).
- The non-volatile memory is divided into three arrays:
  - 256 Kbytes of embedded Flash program memory
  - 8 Kbytes of data EEPROM
  - Options bytes

The options bytes are used to write-protect or read-out protect the memory (with 4 Kbytes granularity) and/or readout-protect the whole memory with the following options:

- Level 0: no readout protection
- Level 1: memory readout protection, the Flash memory cannot be read from or written to if either debug features are connected or boot in RAM is selected
- Level 2: chip readout protection, debug features (ARM Cortex-M3 JTAG and serial wire) and boot in RAM selection disabled (JTAG fuse)

The whole non-volatile memory embeds the error correction code (ECC) feature.

The user area of the Flash memory can be protected against Dbus read access by PCROP feature (see RM0038 for details).

### 3.8 DMA (direct memory access)

The flexible 12-channel, general-purpose DMA is able to manage memory-to-memory, peripheral-to-memory and memory-to-peripheral transfers. The DMA controller supports circular buffer management, avoiding the generation of interrupts when the controller reaches the end of the buffer.

Each channel is connected to dedicated hardware DMA requests, with software trigger support for each channel. Configuration is done by software and transfer sizes between source and destination are independent.

The DMA can be used with the main peripherals: SPI, I<sup>2</sup>C, USART, general-purpose timers, DAC and ADC.

### 3.9 LCD (liquid crystal display)

The LCD drives up to 8 common terminals and 44 segment terminals to drive up to 320 pixels.

- Internal step-up converter to guarantee functionality and contrast control irrespective of  $V_{DD}$ . This converter can be deactivated, in which case the  $V_{LCD}$  pin is used to provide the voltage to the LCD
- Supports static, 1/2, 1/3, 1/4 and 1/8 duty
- Supports static, 1/2, 1/3 and 1/4 bias
- Phase inversion to reduce power consumption and EMI
- Up to 8 pixels can be programmed to blink
- Unneeded segments and common pins can be used as general I/O pins
- LCD RAM can be updated at any time owing to a double-buffer
- The LCD controller can operate in Stop mode
- $V_{LCD}$  rail decoupling capability

**Table 6.  $V_{LCD}$  rail decoupling**

	Bias			Pin	
	1/2	1/3	1/4		
$V_{LCDRAIL1}$	$1/2 V_{LCD}$	$2/3 V_{LCD}$	$1/2 V_{LCD}$	PB2	
$V_{LCDRAIL2}$	N/A	$1/3 V_{LCD}$	$1/4 V_{LCD}$	PB12	PE11
$V_{LCDRAIL3}$	N/A	N/A	$3/4 V_{LCD}$	PB0	PE12

### 3.10 ADC (analog-to-digital converter)

A 12-bit analog-to-digital converters is embedded into STM32L151xC and STM32L152xC devices with up to 25 external channels, performing conversions in single-shot or scan mode. In scan mode, automatic conversion is performed on a selected group of analog inputs with up to 24 external channels in a group.

The ADC can be served by the DMA controller.

An analog watchdog feature allows very precise monitoring of the converted voltage of one, some or all scanned channels. An interrupt is generated when the converted voltage is outside the programmed thresholds.

The events generated by the general-purpose timers (TIMx) can be internally connected to the ADC start triggers, to allow the application to synchronize A/D conversions and timers. An injection mode allows high priority conversions to be done by interrupting a scan mode which runs in as a background task.

The ADC includes a specific low-power mode. The converter is able to operate at maximum speed even if the CPU is operating at a very low frequency and has an auto-shutdown function. The ADC's runtime and analog front-end current consumption are thus minimized whatever the MCU operating mode.

**Functional overview****STM32L151xC STM32L152xC****3.10.1 Temperature sensor**

The temperature sensor (TS) generates a voltage  $V_{SENSE}$  that varies linearly with temperature.

The temperature sensor is internally connected to the ADC\_IN16 input channel which is used to convert the sensor output voltage into a digital value.

The sensor provides good linearity but it has to be calibrated to obtain good overall accuracy of the temperature measurement. As the offset of the temperature sensor varies from chip to chip due to process variation, the uncalibrated internal temperature sensor is suitable for applications that detect temperature changes only.

To improve the accuracy of the temperature sensor measurement, each device is individually factory-calibrated by ST. The temperature sensor factory calibration data are stored by ST in the system memory area, accessible in read-only mode. See [Table 61: Temperature sensor calibration values](#).

**3.10.2 Internal voltage reference ( $V_{REFINT}$ )**

The internal voltage reference ( $V_{REFINT}$ ) provides a stable (bandgap) voltage output for the ADC and Comparators.  $V_{REFINT}$  is internally connected to the ADC\_IN17 input channel. It enables accurate monitoring of the  $V_{DD}$  value (when no external voltage,  $V_{REF+}$ , is available for ADC). The precise voltage of  $V_{REFINT}$  is individually measured for each part by ST during production test and stored in the system memory area. It is accessible in read-only mode. See [Table 16: Embedded internal reference voltage calibration values](#).

**3.11 DAC (digital-to-analog converter)**

The two 12-bit buffered DAC channels can be used to convert two digital signals into two analog voltage signal outputs. The chosen design structure is composed of integrated resistor strings and an amplifier in non-inverting configuration.

This dual digital Interface supports the following features:

- Two DAC converters: one for each output channel
- 8-bit or 12-bit monotonic output
- Left or right data alignment in 12-bit mode
- Synchronized update capability
- Noise-wave generation
- Triangular-wave generation
- Dual DAC channels, independent or simultaneous conversions
- DMA capability for each channel (including the underrun interrupt)
- External triggers for conversion
- Input reference voltage  $V_{REF+}$

Eight DAC trigger inputs are used in the STM32L151xC and STM32L152xC devices. The DAC channels are triggered through the timer update outputs that are also connected to different DMA channels.

### 3.12 Operational amplifier

The STM32L151xC and STM32L152xC devices embed two operational amplifiers with external or internal follower routing capability (or even amplifier and filter capability with external components). When one operational amplifier is selected, one external ADC channel is used to enable output measurement.

The operational amplifiers feature:

- Low input bias current
- Low offset voltage
- Low-power mode
- Rail-to-rail input

### 3.13 Ultra-low-power comparators and reference voltage

The STM32L151xC and STM32L152xC devices embed two comparators sharing the same current bias and reference voltage. The reference voltage can be internal or external (coming from an I/O).

- One comparator with fixed threshold
- One comparator with rail-to-rail inputs, fast or slow mode. The threshold can be one of the following:
  - DAC output
  - External I/O
  - Internal reference voltage ( $V_{REFINT}$ ) or a sub-multiple (1/4, 1/2, 3/4)

Both comparators can wake up from Stop mode, and be combined into a window comparator.

The internal reference voltage is available externally via a low-power / low-current output buffer (driving current capability of 1  $\mu$ A typical).

### 3.14 System configuration controller and routing interface

The system configuration controller provides the capability to remap some alternate functions on different I/O ports.

The highly flexible routing interface allows the application firmware to control the routing of different I/Os to the TIM2, TIM3 and TIM4 timer input captures. It also controls the routing of internal analog signals to ADC1, COMP1 and COMP2 and the internal reference voltage  $V_{REFINT}$ .

### 3.15 Touch sensing

The STM32L151xC and STM32L152xC devices provide a simple solution for adding capacitive sensing functionality to any application. These devices offer up to 23 capacitive sensing channels distributed over 10 analog I/O groups. Both software and timer capacitive sensing acquisition modes are supported.

Capacitive sensing technology is able to detect the presence of a finger near a sensor which is protected from direct touch by a dielectric (glass, plastic...). The capacitive variation

**Functional overview****STM32L151xC STM32L152xC**

introduced by the finger (or any conductive object) is measured using a proven implementation based on a surface charge transfer acquisition principle. It consists of charging the sensor capacitance and then transferring a part of the accumulated charges into a sampling capacitor until the voltage across this capacitor has reached a specific threshold. The capacitive sensing acquisition only requires few external components to operate. This acquisition is managed directly by the GPIOs, timers and analog I/O groups (see [Section 3.14: System configuration controller and routing interface](#)).

Reliable touch sensing functionality can be quickly and easily implemented using the free STM32L1xx STMTouch touch sensing firmware library.

### **3.16 Timers and watchdogs**

The ultra-low-power STM32L151xC and STM32L152xC devices include seven general-purpose timers, two basic timers, and two watchdog timers.

[Table 7](#) compares the features of the general-purpose and basic timers.

**Table 7. Timer feature comparison**

Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary outputs
TIM2, TIM3, TIM4	16-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No
TIM5	32-bit	Up, down, up/down	Any integer between 1 and 65536	Yes	4	No
TIM9	16-bit	Up, down, up/down	Any integer between 1 and 65536	No	2	No
TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No
TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No

#### **3.16.1 General-purpose timers (TIM2, TIM3, TIM4, TIM5, TIM9, TIM10 and TIM11)**

There are seven synchronizable general-purpose timers embedded in the STM32L151xC and STM32L152xC devices (see [Table 7](#) for differences).

##### **TIM2, TIM3, TIM4, TIM5**

TIM2, TIM3, TIM4 are based on 16-bit auto-reload up/down counter. TIM5 is based on a 32-bit auto-reload up/down counter. They include a 16-bit prescaler. They feature four independent channels each for input capture/output compare, PWM or one-pulse mode output. This gives up to 16 input captures/output compares/PWMs on the largest packages.

TIM2, TIM3, TIM4, TIM5 general-purpose timers can work together or with the TIM10, TIM11 and TIM9 general-purpose timers via the Timer Link feature for synchronization or event chaining. Their counter can be frozen in debug mode. Any of the general-purpose timers can be used to generate PWM outputs.

**STM32L151xC STM32L152xC****Functional overview**

TIM2, TIM3, TIM4, TIM5 all have independent DMA request generation.

These timers are capable of handling quadrature (incremental) encoder signals and the digital outputs from 1 to 3 hall-effect sensors.

**TIM10, TIM11 and TIM9**

TIM10 and TIM11 are based on a 16-bit auto-reload upcounter. TIM9 is based on a 16-bit auto-reload up/down counter. They include a 16-bit prescaler. TIM10 and TIM11 feature one independent channel, whereas TIM9 has two independent channels for input capture/output compare, PWM or one-pulse mode output. They can be synchronized with the TIM2, TIM3, TIM4, TIM5 full-featured general-purpose timers.

They can also be used as simple time bases and be clocked by the LSE clock source (32.768 kHz) to provide time bases independent from the main CPU clock.

**3.16.2 Basic timers (TIM6 and TIM7)**

These timers are mainly used for DAC trigger generation. They can also be used as generic 16-bit time bases.

**3.16.3 SysTick timer**

This timer is dedicated to the OS, but could also be used as a standard downcounter. It is based on a 24-bit downcounter with autoreload capability and a programmable clock source. It features a maskable system interrupt generation when the counter reaches 0.

**3.16.4 Independent watchdog (IWDG)**

The independent watchdog is based on a 12-bit downcounter and 8-bit prescaler. It is clocked from an independent 37 kHz internal RC and, as it operates independently of the main clock, it can operate in Stop and Standby modes. It can be used either as a watchdog to reset the device when a problem occurs, or as a free-running timer for application timeout management. It is hardware- or software-configurable through the option bytes. The counter can be frozen in debug mode.

**3.16.5 Window watchdog (WWDG)**

The window watchdog is based on a 7-bit downcounter that can be set as free-running. It can be used as a watchdog to reset the device when a problem occurs. It is clocked from the main clock. It has an early warning interrupt capability and the counter can be frozen in debug mode.

**3.17 Communication interfaces****3.17.1 I<sup>2</sup>C bus**

Up to two I<sup>2</sup>C bus interfaces can operate in multimaster and slave modes. They can support standard and fast modes.

They support dual slave addressing (7-bit only) and both 7- and 10-bit addressing in master mode. A hardware CRC generation/verification is embedded.

They can be served by DMA and they support SM Bus 2.0/PM Bus.

**Functional overview****STM32L151xC STM32L152xC****3.17.2 Universal synchronous/asynchronous receiver transmitter (USART)**

The three USART interfaces are able to communicate at speeds of up to 4 Mbit/s. They support IrDA SIR ENDEC and have LIN Master/Slave capability. The three USARTs provide hardware management of the CTS and RTS signals and are ISO 7816 compliant.

All USART interfaces can be served by the DMA controller.

**3.17.3 Serial peripheral interface (SPI)**

Up to three SPIs are able to communicate at up to 16 Mbits/s in slave and master modes in full-duplex and half-duplex communication modes. The 3-bit prescaler gives 8 master mode frequencies and the frame is configurable to 8 bits or 16 bits. The hardware CRC generation/verification supports basic SD Card/MMC modes.

The SPIs can be served by the DMA controller.

**3.17.4 Inter-integrated sound (I<sup>2</sup>S)**

Two standard I2S interfaces (multiplexed with SPI2 and SPI3) are available. They can operate in master or slave mode, and can be configured to operate with a 16-/32-bit resolution as input or output channels. Audio sampling frequencies from 8 kHz up to 192 kHz are supported. When either or both of the I2S interfaces is/are configured in master mode, the master clock can be output to the external DAC/CODEC at 256 times the sampling frequency.

The I2Ss can be served by the DMA controller.

**3.17.5 Universal serial bus (USB)**

The STM32L151xC and STM32L152xC devices embed a USB device peripheral compatible with the USB full-speed 12 Mbit/s. The USB interface implements a full-speed (12 Mbit/s) function interface. It has software-configurable endpoint setting and supports suspend/resume. The dedicated 48 MHz clock is generated from the internal main PLL (the clock source must use a HSE crystal oscillator).

**3.18 CRC (cyclic redundancy check) calculation unit**

The CRC (cyclic redundancy check) calculation unit is used to get a CRC code from a 32-bit data word and a fixed generator polynomial.

Among other applications, CRC-based techniques are used to verify data transmission or storage integrity. In the scope of the EN/IEC 60335-1 standard, they offer a means of verifying the Flash memory integrity. The CRC calculation unit helps compute a signature of the software during runtime, to be compared with a reference signature generated at link-time and stored at a given memory location.

## 3.19 Development support

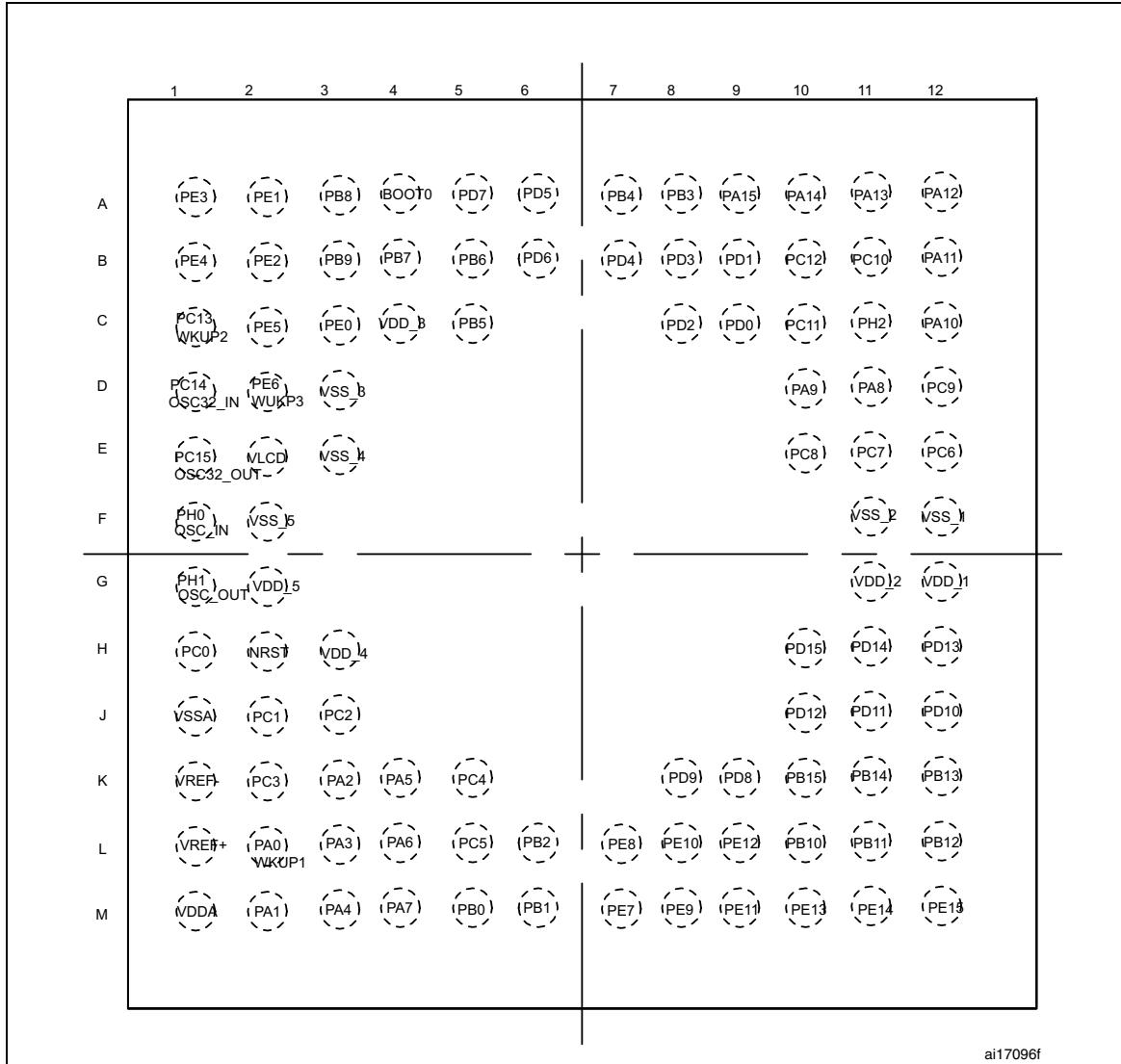
### 3.19.1 Serial wire JTAG debug port (SWJ-DP)

The ARM SWJ-DP interface is embedded, and is a combined JTAG and serial wire debug port that enables either a serial wire debug or a JTAG probe to be connected to the target. The JTAG JTMS and JTCK pins are shared with SWDAT and SWCLK, respectively, and a specific sequence on the JTMS pin is used to switch between JTAG-DP and SW-DP.

The JTAG port can be permanently disabled with a JTAG fuse.

### 3.19.2 Embedded Trace Macrocell™

The ARM® Embedded Trace Macrocell provides a greater visibility of the instruction and data flow inside the CPU core by streaming compressed data at a very high rate from the STM32L151xC and STM32L152xC device through a small number of ETM pins to an external hardware trace port analyzer (TPA) device. The TPA is connected to a host computer using USB, Ethernet, or any other high-speed channel. Real-time instruction and data flow activity can be recorded and then formatted for display on the host computer running debugger software. TPA hardware is commercially available from common development tool vendors. It operates with third party debugger software tools.

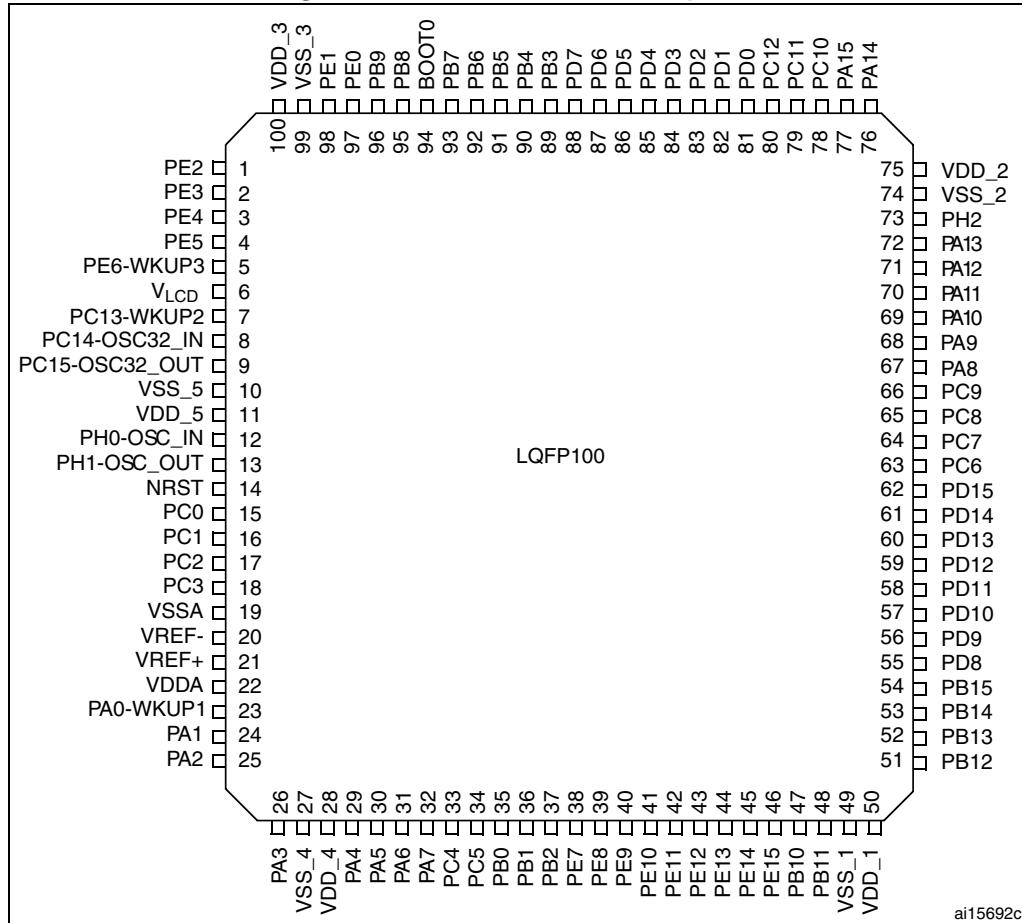
**Pin descriptions****STM32L151xC STM32L152xC****4 Pin descriptions****Figure 3. STM32L15xVC UFBGA100 ballout**

1. This figure shows the package top view.

## STM32L151xC STM32L152xC

## Pin descriptions

Figure 4. STM32L15xVC LQFP100 pinout

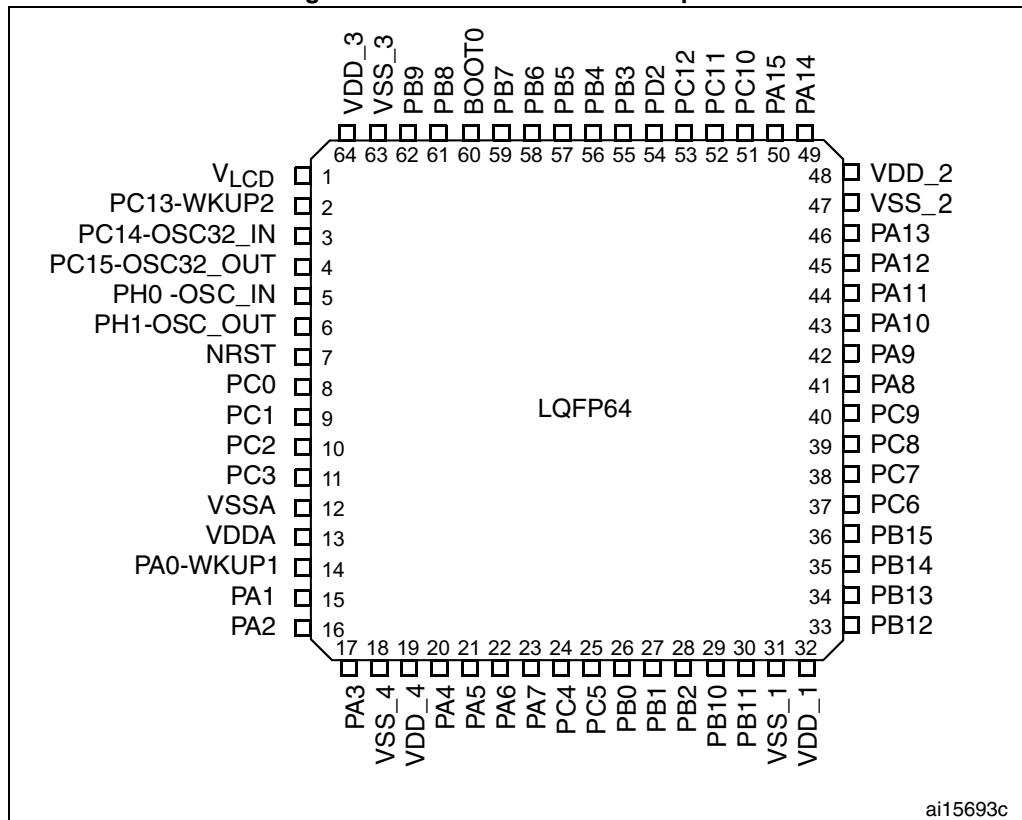


1. This figure shows the package top view.

## Pin descriptions

## STM32L151xC STM32L152xC

Figure 5. STM32L15xRC LQFP64 pinout

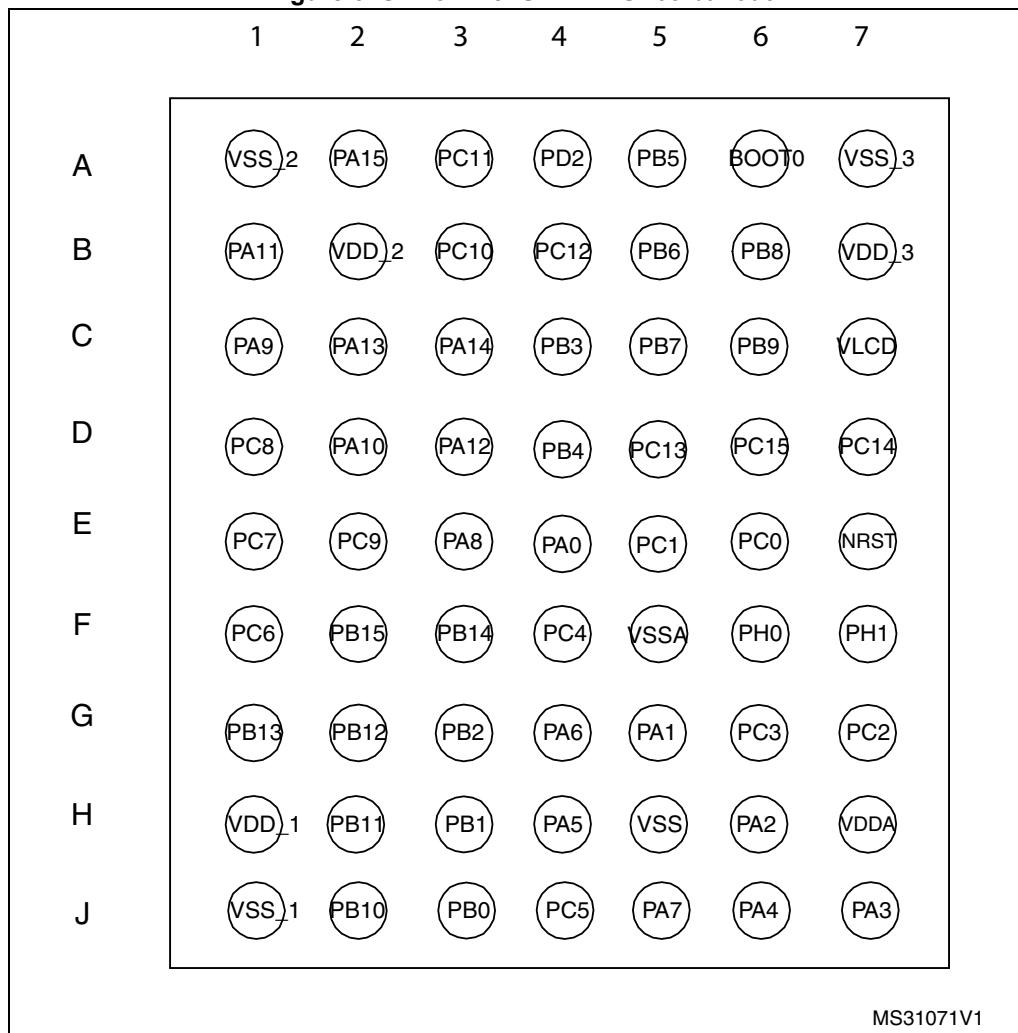


1. This figure shows the package top view.

## STM32L151xC STM32L152xC

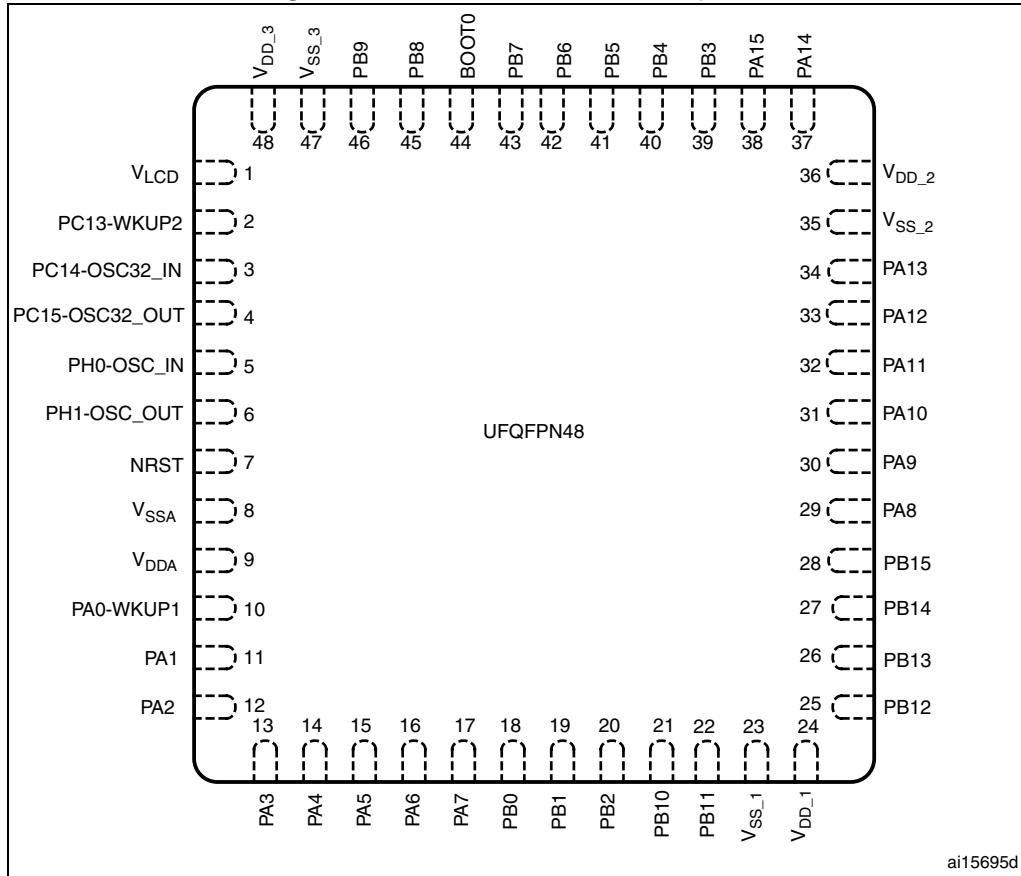
## Pin descriptions

Figure 6. STM32L15xUC WLCSP63 ballout



1. This figure shows the package top view.

MS31071V1

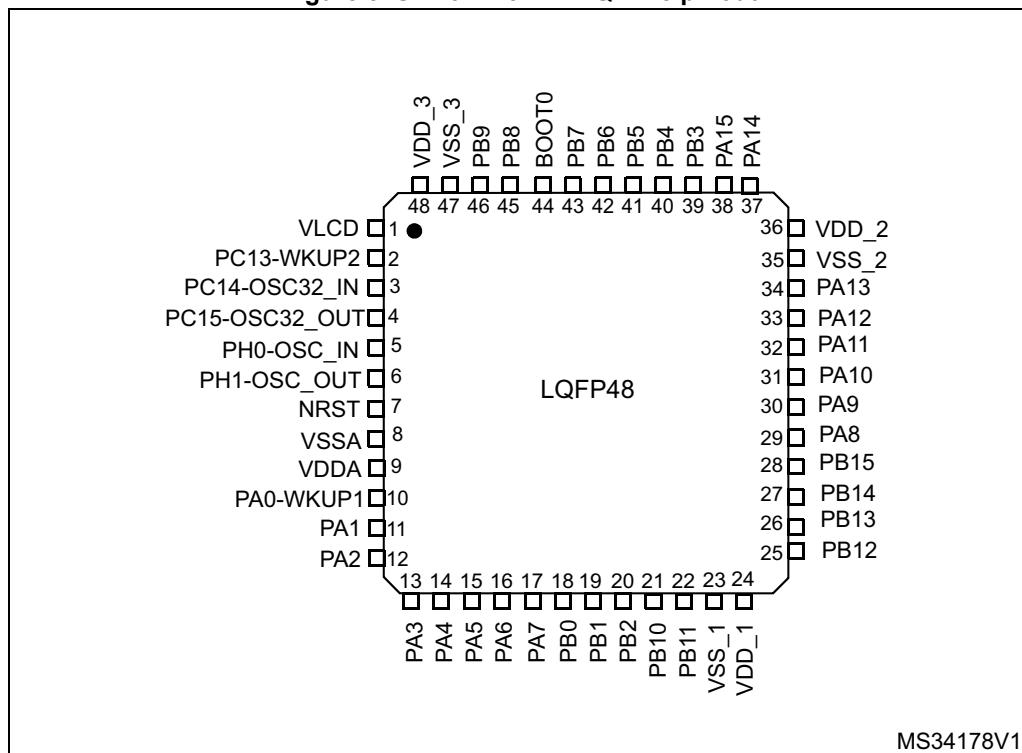
**Pin descriptions****STM32L151xC STM32L152xC****Figure 7. STM32L15xCC UFQFPN48 pinout**

1. This figure shows the package top view.

## STM32L151xC STM32L152xC

## Pin descriptions

Figure 8. STM32L15xCC LQFP48 pinout



1. This figure shows the package top view.

## Pin descriptions

## STM32L151xC STM32L152xC

Table 8. Legend/abbreviations used in the pinout table

Name	Abbreviation	Definition
Pin name		Unless otherwise specified in brackets below the pin name, the pin function during and after reset is the same as the actual pin name
Pin type	S	Supply pin
	I	Input only pin
	I/O	Input / output pin
I/O structure	FT	5 V tolerant I/O
	TC	Standard 3.3 V I/O
	B	Dedicated BOOT0 pin
	RST	Bidirectional reset pin with embedded weak pull-up resistor
Notes		Unless otherwise specified by a note, all I/Os are set as floating inputs during and after reset
Pin functions	Alternate functions	Functions selected through GPIOx_AFR registers
	Additional functions	Functions directly selected/enabled through peripheral registers

Table 9. STM32L151xC and STM32L152xC pin definitions

Pins						Pin name	Pin type <sup>(1)</sup>	I / O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48						Alternate functions	Additional functions
B2	1	-	-	-	PE2	I/O	FT	PE2	TIM3_ETR/LCD_SEG38 /TRACECLK	-	
A1	2	-	-	-	PE3	I/O	FT	PE3	TIM3_CH1/LCD_SEG39 /TRACED0	-	
B1	3	-	-	-	PE4	I/O	FT	PE4	TIM3_CH2/TRACED1	-	
C2	4	-	-	-	PE5	I/O	FT	PE5	TIM9_CH1/TRACED2	-	
D2	5	-	-	-	PE6-WKUP3	I/O	FT	PE6	TIM9_CH2/ TRACED3	WKUP3/ RTC_TAMP3	
E2	6	1	C7	1	V <sub>LCD</sub> <sup>(3)</sup>	S	-	V <sub>LCD</sub>	-	-	

## STM32L151xC STM32L152xC

## Pin descriptions

Table 9. STM32L151xC and STM32L152xC pin definitions (continued)

Pins					Pin name	Pin type <sup>(1)</sup>	I/O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48					Alternate functions	Additional functions
C1	7	2	D5	2	PC13-WKUP2	I/O	FT	PC13	-	WKUP2/ RTC_TAMP1/ RTC_TS/RTC_OUT
D1	8	3	D7	3	PC14-OSC32_IN <sup>(4)</sup>	I/O	TC	PC14	-	OSC32_IN
E1	9	4	D6	4	PC15-OSC32_OUT	I/O	TC	PC15	-	OSC32_OUT
F2	10	-	-	-	V <sub>SS_5</sub>	S	-	V <sub>SS_5</sub>	-	-
G2	11	-	-	-	V <sub>DD_5</sub>	S	-	V <sub>DD_5</sub>	-	-
F1	12	5	F6	5	PH0-OSC_IN <sup>(5)</sup>	I/O	TC	PH0	-	OSC_IN
G1	13	6	F7	6	PH1-OSC_OUT <sup>(5)</sup>	I/O	TC	PH1	-	OSC_OUT
H2	14	7	E7	7	NRST	I/O	RST	NRST	-	-
H1	15	8	E6	-	PC0	I/O	FT	PC0	LCD_SEG18	ADC_IN10/ COMP1_INP
J2	16	9	E5	-	PC1	I/O	FT	PC1	LCD_SEG19	ADC_IN11/ COMP1_INP
J3	17	10	G7	-	PC2	I/O	FT	PC2	LCD_SEG20	ADC_IN12/ COMP1_INP
K2	18	11	G6	-	PC3	I/O	TC	PC3	LCD_SEG21	ADC_IN13/ COMP1_INP
J1	19	12	F5	8	V <sub>SSA</sub>	S	-	V <sub>SSA</sub>	-	-
K1	20	-	-	-	V <sub>REF-</sub>	S	-	V <sub>REF-</sub>	-	-
L1	21	-	-	-	V <sub>REF+</sub>	S	-	V <sub>REF+</sub>	-	-
M1	22	13	H7	9	V <sub>DDA</sub>	S	-	V <sub>DDA</sub>	-	-
L2	23	14	E4	10	PA0-WKUP1	I/O	FT	PA0	TIM2_CH1_ETR/ TIM5_CH1/ USART2_CTS	WKUP1/ RTC_TAMP2/ ADC_IN0/ COMP1_INP

## Pin descriptions

## STM32L151xC STM32L152xC

Table 9. STM32L151xC and STM32L152xC pin definitions (continued)

Pins					Pin name	Pin type <sup>(1)</sup>	I/O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48					Alternate functions	Additional functions
M2	24	15	G5	11	PA1	I/O	FT	PA1	TIM2_CH2/TIM5_CH2/ USART2_RTS/ LCD_SEG0	ADC_IN1/ COMP1_INP/ OPAMP1_VINP
K3	25	16	H6	12	PA2	I/O	FT	PA2	TIM2_CH3/TIM5_CH3/ TIM9_CH1/USART2_TX/ LCD_SEG1	ADC_IN2/ COMP1_INP/ OPAMP1_VINM
L3	26	17	J7	13	PA3	I/O	TC	PA3	TIM2_CH4/TIM5_CH4/ TIM9_CH2/USART2_RX/ LCD_SEG2	ADC_IN3/ COMP1_INP/ OPAMP1_VOUT
E3	27	18	-	-	V <sub>SS_4</sub>	S	-	V <sub>SS_4</sub>	-	-
H3	28	19	-	-	V <sub>DD_4</sub>	S	-	V <sub>DD_4</sub>	-	-
M3	29	20	J6	14	PA4	I/O	TC	PA4	SPI1_NSS/SPI3_NSS/ I2S3_WS/ USART2_CK	ADC_IN4/ DAC_OUT1/ COMP1_INP
K4	30	21	H4	15	PA5	I/O	TC	PA5	TIM2_CH1_ETR/ SPI1_SCK	ADC_IN5/ DAC_OUT2/ COMP1_INP
L4	31	22	G4	16	PA6	I/O	FT	PA6	TIM3_CH1/TIM10_CH1/ SPI1_MISO/LCD_SEG3	ADC_IN6/ COMP1_INP/ OPAMP2_VINP
M4	32	23	J5	17	PA7	I/O	FT	PA7	TIM3_CH2/TIM11_CH1/ SPI1_MOSI/LCD_SEG4	ADC_IN7/ COMP1_INP/ OPAMP2_VINM
K5	33	24	F4	-	PC4	I/O	FT	PC4	LCD_SEG22	ADC_IN14/ COMP1_INP
L5	34	25	J4	-	PC5	I/O	FT	PC5	LCD_SEG23	ADC_IN15/ COMP1_INP
M5	35	26	J3	18	PB0	I/O	TC	PB0	TIM3_CH3/LCD_SEG5	ADC_IN8/ COMP1_INP/ OPAMP2_VOUT/ VLCDRAIL3/ VREF_OUT

## STM32L151xC STM32L152xC

## Pin descriptions

Table 9. STM32L151xC and STM32L152xC pin definitions (continued)

Pins					Pin name	Pin type <sup>(1)</sup>	I/O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48					Alternate functions	Additional functions
M6	36	27	H3	19	PB1	I/O	FT	PB1	TIM3_CH4/LCD_SEG6	ADC_IN9/ COMP1_INP/ VREF_OUT
L6	37	28	G3	20	PB2	I/O	FT	PB2 /BOOT1	BOOT1	VLCDRAIL1/ ADCIN0b
M7	38	-	-	-	PE7	I/O	TC	PE7	-	ADC_IN22/ COMP1_INP
L7	39	-	-	-	PE8	I/O	TC	PE8	-	ADC_IN23/ COMP1_INP
M8	40	-	-	-	PE9	-	TC	PE9	TIM2_CH1_ETR/ TIM5_ETR	ADC_IN24/ COMP1_INP
L8	41	-	-	-	PE10	I/O	TC	PE10	TIM2_CH2	ADC_IN25/ COMP1_INP
M9	42	-	-	-	PE11	I/O	FT	PE11	TIM2_CH3	VLCDRAIL2
L9	43	-	-	-	PE12	I/O	FT	PE12	TIM2_CH4/SPI1_NSS	VLCDRAIL3
M10	44	-	-	-	PE13	I/O	FT	PE13	SPI1_SCK	-
M11	45	-	-	-	PE14	I/O	FT	PE14	SPI1_MISO	-
M12	46	-	-	-	PE15	I/O	FT	PE15	SPI1_MOSI	-
L10	47	29	J2	21	PB10	I/O	FT	PB10	TIM2_CH3/I2C2_SCL/ USART3_TX/ LCD_SEG10	-
L11	48	30	H2	22	PB11	I/O	FT	PB11	TIM2_CH4/I2C2_SDA/ USART3_RX/ LCD_SEG11	-
-	-	-	H5	-	V <sub>SS</sub>	S	-	V <sub>SS</sub>	-	-
F12	49	31	J1	23	V <sub>SS_1</sub>	S	-	V <sub>SS_1</sub>	-	-
G12	50	32	H1	24	V <sub>DD_1</sub>	S	-	V <sub>DD_1</sub>	-	-

## Pin descriptions

## STM32L151xC STM32L152xC

Table 9. STM32L151xC and STM32L152xC pin definitions (continued)

Pins					Pin name	Pin type <sup>(1)</sup>	I/O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48					Alternate functions	Additional functions
L12	51	33	G2	25	PB12	I/O	FT	PB12	TIM10_CH1 /I2C2_SMBA/ SPI2_NSS/I2S2_WS/ USART3_CK/ LCD_SEG12	ADC_IN18/ COMP1_INP/ VLCDRAIL2
K12	52	34	G1	26	PB13	I/O	FT	PB13	TIM9_CH1/SPI2_SCK/ I2S2_CK/ USART3_CTS/ LCD_SEG13	ADC_IN19/ COMP1_INP
K11	53	35	F3	27	PB14	I/O	FT	PB14	TIM9_CH2/SPI2_MISO/ USART3_RTS/ LCD_SEG14	ADC_IN20/ COMP1_INP
K10	54	36	F2	28	PB15	I/O	FT	PB15	TIM11_CH1/SPI2_MOSI /I2S2_SD/LCD_SEG15	ADC_IN21/ COMP1_INP/ RTC_REFIN
K9	55	-	-	-	PD8	I/O	FT	PD8	USART3_TX/ LCD_SEG28	-
K8	56	-	-	-	PD9	I/O	FT	PD9	USART3_RX/ LCD_SEG29	-
J12	57	-	-	-	PD10	I/O	FT	PD10	USART3_CK/ LCD_SEG30	-
J11	58	-	-	-	PD11	I/O	FT	PD11	USART3_CTS/ LCD_SEG31	-
J10	59	-	-	-	PD12	I/O	FT	PD12	TIM4_CH1/ USART3_RTS/ LCD_SEG32	-
H12	60	-	-	-	PD13	I/O	FT	PD13	TIM4_CH2/LCD_SEG33	-
H11	61	-	-	-	PD14	I/O	FT	PD14	TIM4_CH3/LCD_SEG34	-
H10	62	-	-	-	PD15	I/O	FT	PD15	TIM4_CH4/LCD_SEG35	-
E12	63	37	F1	-	PC6	I/O	FT	PC6	TIM3_CH1/I2S2_MCK/ LCD_SEG24	-

## STM32L151xC STM32L152xC

## Pin descriptions

Table 9. STM32L151xC and STM32L152xC pin definitions (continued)

Pins					Pin name	Pin type <sup>(1)</sup>	I/O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48					Alternate functions	Additional functions
E11	64	38	E1	-	PC7	I/O	FT	PC7	TIM3_CH2/I2S3_MCK/ LCD_SEG25	-
E10	65	39	D1	-	PC8	I/O	FT	PC8	TIM3_CH3/LCD_SEG26	-
D12	66	40	E2	-	PC9	I/O	FT	PC9	TIM3_CH4/LCD_SEG27	-
D11	67	41	E3	29	PA8	I/O	FT	PA8	USART1_CK/MCO/ LCD_COM0	-
D10	68	42	C1	30	PA9	I/O	FT	PA9	USART1_TX/ LCD_COM1	-
C12	69	43	D2	31	PA10	I/O	FT	PA10	USART1_RX/ LCD_COM2	-
B12	70	44	B1	32	PA11	I/O	FT	PA11	USART1_CTS/ SPI1_MISO	USB_DM
A12	71	45	D3	33	PA12	I/O	FT	PA12	USART1_RTS/ SPI1_MOSI	USB_DP
A11	72	46	C2	34	PA13	I/O	FT	JTMS-SWDIO	JTMS-SWDIO	-
C11	73	-	-	-	PH2	I/O	FT	PH2	-	-
F11	74	47	A1	35	V <sub>SS_2</sub>	S	-	V <sub>SS_2</sub>	-	-
G11	75	48	B2	36	V <sub>DD_2</sub>	S	-	V <sub>DD_2</sub>	-	-
A10	76	49	C3	37	PA14	I/O	FT	JTCK-SWCLK	JTCK-SWCLK	-
A9	77	50	A2	38	PA15	I/O	FT	JTDI	TIM2_CH1_ETR/ SPI1_NSS/ SPI3_NSS/I2S3_WS/ LCD_SEG17/JTDI	-
B11	78	51	B3	-	PC10	I/O	FT	PC10	SPI3_SCK/I2S3_CK/ USART3_TX/ LCD_SEG28/ LCD_SEG40/ LCD_COM4	-

## Pin descriptions

## STM32L151xC STM32L152xC

Table 9. STM32L151xC and STM32L152xC pin definitions (continued)

Pins					Pin name	Pin type <sup>(1)</sup>	I/O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48					Alternate functions	Additional functions
C10	79	52	A3	-	PC11	I/O	FT	PC11	SPI3_MISO/ USART3_RX/ LCD_SEG29/ LCD_SEG41/ LCD_COM5	-
B10	80	53	B4	-	PC12	I/O	FT	PC12	SPI3_MOSI/I2S3_SD/ USART3_CK/ LCD_SEG30/ LCD_SEG42/ LCD_COM6	-
C9	81	-	-	-	PD0	I/O	FT	PD0	TIM9_CH1/SPI2_NSS/ I2S2_WS	-
B9	82	-	-	-	PD1	I/O	FT	PD1	SPI2_SCK/I2S2_CK	-
C8	83	54	A4	-	PD2	I/O	FT	PD2	TIM3_ETR/LCD_SEG31 /LCD_SEG43/ LCD_COM7	-
B8	84	-	-	-	PD3	I/O	FT	PD3	SPI2_MISO/ USART2_CTS	-
B7	85	-	-	-	PD4	I/O	FT	PD4	SPI2_MOSI/I2S2_SD/ USART2_RTS	-
A6	86	-	-	-	PD5	I/O	FT	PD5	USART2_TX	-
B6	87	-	-	-	PD6	I/O	FT	PD6	USART2_RX	-
A5	88	-	-	-	PD7	I/O	FT	PD7	TIM9_CH2/USART2_CK	-
A8	89	55	C4	39	PB3	I/O	FT	JTDO	TIM2_CH2/SPI1_SCK/ SPI3_SCK/I2S3_CK/ LCD_SEG7/JTDO	COMP2_INM
A7	90	56	D4	40	PB4	I/O	FT	NJTRST	TIM3_CH1/SPI1_MISO/ SPI3_MISO/LCD_SEG8 /NJTRST	COMP2_INP
C5	91	57	A5	41	PB5	I/O	FT	PB5	TIM3_CH2/I2C1_SMBA/ SPI1_MOSI/SPI3_MOSI /I2S3_SD/LCD_SEG9	COMP2_INP

## STM32L151xC STM32L152xC

## Pin descriptions

Table 9. STM32L151xC and STM32L152xC pin definitions (continued)

Pins					Pin name	Pin type <sup>(1)</sup>	I/O Structure	Main function <sup>(2)</sup> (after reset)	Pin functions	
UFBGA100	LQFP100	LQFP64	WL CSP63	LQFP48 or UFQFPN48					Alternate functions	Additional functions
B5	92	58	B5	42	PB6	I/O	FT	PB6	TIM4_CH1/I2C1_SCL/ USART1_TX	COMP2_INP
B4	93	59	C5	43	PB7	I/O	FT	PB7	TIM4_CH2/I2C1_SDA/ USART1_RX	COMP2_INP/PVD_IN
A4	94	60	A6	44	BOOT0	I	B	BOOT0	-	-
A3	95	61	B6	45	PB8	I/O	FT	PB8	TIM4_CH3/TIM10_CH1/ I2C1_SCL/LCD_SEG16	-
B3	96	62	C6	46	PB9	I/O	FT	PB9	TIM4_CH4/TIM11_CH1/ I2C1_SDA/LCD_COM3	-
C3	97	-	-	-	PE0	I/O	FT	PE0	TIM4_ETR/TIM10_CH1/ LCD_SEG36	-
A2	98	-	-	-	PE1	I/O	FT	PE1	TIM11_CH1/ LCD_SEG37	-
D3	99	63	A7	47	V <sub>SS_3</sub>	S	-	V <sub>SS_3</sub>	-	-
C4	100	64	B7	48	V <sub>DD_3</sub>	S	-	V <sub>DD_3</sub>	-	-

1. I = input, O = output, S = supply.

2. Function availability depends on the chosen device.

3. Applicable to STM32L152xC devices only. In STM32L151xC devices, this pin should be connected to V<sub>DD</sub>.

4. The PC14 and PC15 I/Os are only configured as OSC32\_IN/OSC32\_OUT when the LSE oscillator is ON (by setting the LSEON bit in the RCC\_CSR register). The LSE oscillator pins OSC32\_IN/OSC32\_OUT can be used as general-purpose PH0/PH1 I/Os, respectively, when the LSE oscillator is off (after reset, the LSE oscillator is off). The LSE has priority over the GPIO function. For more details, refer to Using the OSC32\_IN/OSC32\_OUT pins as GPIO PC14/PC15 port pins section in the STM32L151xx, STM32L152xx and STM32L162xx reference manual (RM0038).

5. The PH0 and PH1 I/Os are only configured as OSC\_IN/OSC\_OUT when the HSE oscillator is ON (by setting the HSEON bit in the RCC\_CR register). The HSE oscillator pins OSC\_IN/OSC\_OUT can be used as general-purpose PH0/PH1 I/Os, respectively, when the HSE oscillator is off (after reset, the HSE oscillator is off). The HSE has priority over the GPIO function.

**Alternate functions****Table 10. Alternate function input/output**

Port name	Digital alternate function number							
	AFIO0	AFIO1	AFIO2	AFIO3	AFIO4	AFIO5	AFIO6	AFIO7
Alternate function								
SYSTEM	TIM2	TIM3/4/5	10/11	TIM9/12	SPI1/2	SPI3	USART1/2/3	LCD
BOOT0	-	-	-	-	-	-	-	-
NRST	-	-	-	-	-	-	-	-
PA0_WKUP1	-	TIM2_CH1_ETR	TIM5_CH1	-	-	-	USART2_CTS	-
PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	USART2 RTS	SEG0
PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	USART2_TX	SEG1
PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	USART2_RX	SEG2
PA4	-	-	-	-	SPI1_NSS	SPI3_NSS_I2S3_WS	USART2_CK	-
PA5	-	TIM2_CH1_ETR	-	-	SPI1_SCK	-	-	TIMx_IC3
PA6	-	TIM3_CH1	TIM10_CH1	-	SPI1_MISO	-	-	TIMx_IC4
PA7	-	TIM3_CH2	TIM11_CH1	-	SPI1_MOSI	-	-	TIMx_IC1
PA8_MCO	-	-	-	-	-	-	-	TIMx_IC2
PA9	-	-	-	-	-	-	SEG3	TIMx_IC3
PA10	-	-	-	-	-	-	SEG4	TIMx_IC4
PA11	-	-	-	-	SPI1_MISO	USART1_CK	COM0	TIMx_IC1
PA12	-	-	-	-	-	USART1_TX	COM1	TIMx_IC2
PA13_JTMS_SWDIO	-	-	-	-	SPI1_MOSI	USART1_RX	COM2	TIMx_IC3
PA14_JTCK_SWCLK	-	-	-	-	-	USART1_CTS	-	TIMx_IC4
PA15_JTDI	TIM2_CH1_ETR	-	-	-	SPI1 NSS_I2S3_WS	-	SEG17	TIMx_IC1
								EVEN TOUT

## STM32L151xC STM32L152xC

## Pin descriptions

Table 10. Alternate function input/output (continued)

Port name	Digital alternate function number									
	AFIO0	AFIO1	AFIO2	AFIO3	AFIO4	AFIO5	AFIO6	AFIO7	AFIO11	AFIO14
Alternate function										
SYSTEM	TIM2	TIM3/4/5	TIM9/10/11	I2C1/2	SPI1/2	SPI3	USART1/2/3	LCD	CPRI	SYSTEM
PB0	-	-	TIM3_CH3	-	-	-	-	SEG5	-	EVEN TOUT
PB1	-	-	TIM3_CH4	-	-	-	-	SEG6	-	EVENT OUT
PB2	BOOT1	-	-	-	-	-	-	-	-	EVENT OUT
PB3	JTDO	TIM2_CH2	-	-	SPI1_SCK	SPI3_SCK I2S3_CK	-	SEG7	-	EVENT OUT
PB4	NJTRST	-	TIM3_CH1	-	SPI1_MISO	SPI3_MISO	-	SEG8	-	EVENT OUT
PB5	-	-	TIM3_CH2	-	I2C1_SMBA	SPI1_MOSI	SPI3_MOSI I2S3_SD	-	SEG9	-
PB6	-	-	TIM4_CH1	-	I2C1_SCL	-	USART1_TX	-	-	EVENT OUT
PB7	-	-	TIM4_CH2	-	I2C1_SDA	-	USART1_RX	-	-	EVENT OUT
PB8	-	-	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	-	SEG16	-	EVENT OUT
PB9	-	-	TIM4_CH4	TIM11_CH1	I2C1_SDA	-	-	COM3	-	EVENT OUT
PB10	-	-	TIM2_CH3	-	I2C2_SCL	-	USART3_TX	SEG10	-	EVENT OUT
PB11	-	-	TIM2_CH4	-	I2C2_SDA	-	USART3_RX	SEG11	-	EVENT OUT
PB12	-	-	-	TIM10_CH1	I2C2_SMBA	SPI2_NSS I2S2_WS	-	SEG12	-	EVENT OUT
PB13	-	-	-	TIM9_CH1	-	SPI2_SCK I2S2_CK	-	SEG13	-	EVENT OUT
PB14	-	-	-	TIM9_CH2	-	SPI2_MISO	-	USART3_RTS	SEG14	-
PB15	-	-	-	TIM11_CH1	-	I2S2_SD	-	-	SEG15	-
PC0	-	-	-	-	-	-	-	SEG18	Timx_IC1	EVENT OUT
PC1	-	-	-	-	-	-	-	SEG19	Timx_IC2	EVENT OUT
PC2	-	-	-	-	-	-	-	SEG20	Timx_IC3	EVENT OUT
PC3	-	-	-	-	-	-	-	SEG21	Timx_IC4	EVENT OUT

## Pin descriptions

## STM32L151xC STM32L152xC

Table 10. Alternate function input/output (continued)

Port name	Digital alternate function number									
	AFIO0	AFIO1	AFIO2	AFIO3	AFIO4	AFIO5	AFIO6	AFIO7	AFIO11	AFIO14
Alternate function										
SYSTEM	TIM2	TIM3/4/5	TIM9/10/11	I2C1/2	SPI1/2	SPI3	USART1/2/3	LCD	CPRI	SYSTEM
PC4	-	-	-	-	-	-	-	SEG22	TIMx_IC1	EVENT OUT
PC5	-	-	-	-	-	-	-	SEG23	TIMx_IC2	EVENT OUT
PC6	-	-	TIM3_CH1	-	I2S2_MCK	-	-	SEG24	TIMx_IC3	EVENT OUT
PC7	-	-	TIM3_CH2	-	I2S3_MCK	-	-	SEG25	TIMx_IC4	EVENT OUT
PC8	-	-	TIM3_CH3	-	-	-	-	SEG26	TIMx_IC1	EVENT OUT
PC9	-	-	TIM3_CH4	-	-	-	-	SEG27	TIMx_IC2	EVENT OUT
PC10	-	-	-	-	SPI3_SCK I2S3_CK	USART3_TX	COM4/ SEG28/ SEG40	TIMx_IC3	EVENT OUT	
PC11	-	-	-	-	-	SPI3_MISO	USART3_RX	COM5/ SEG29/ SEG41	TIMx_IC4	EVENT OUT
PC12	-	-	-	-	-	SPI3_MOSI I2S3_SD	USART3_CK	COM6/ SEG30/ SEG42	TIMx_IC1	EVENT OUT
PC13-WKUP2	-	-	-	-	-	-	-	-	TIMx_IC2	EVENT OUT
PC14-OSC32_IN	-	-	-	-	-	-	-	-	TIMx_IC3	EVENT OUT
PC15-OSC32_OUT	-	-	-	-	-	-	-	-	TIMx_IC4	EVENT OUT
PD0	-	-	-	TIM9_CH1	-	SPI2_NSS I2S2_WS	-	-	TIMx_IC1	EVENT OUT
PD1	-	-	-	-	SPI2_SCK I2S2_CK	-	-	-	TIMx_IC2	EVENT OUT
PD2	-	-	TIM3_ETR	-	-	-	COM7/ SEG31/ SEG43	TIMx_IC3	EVENT OUT	

Table 10. Alternate function input/output (continued)

Port name	Digital alternate function number									
	AFIO0	AFIO1	AFIO2	AFIO3	AFIO4	AFIO5	AFIO6	AFIO7	AFIO11	AFIO14
Alternate function										
SYSTEM	TIM2	TIM3/4/5	TIM9/10/11	I2C1/2	SPI1/2	SPI3	USART1/2/3	LCD	CPRI	SYSTEM
PD3	-	-	-	-	SPI2_MISO I2S2_SD	-	USART2_CTS	-	TIMx_IC4	EVENT OUT
PD4	-	-	-	-	-	-	USART2_RTS	-	TIMx_IC1	EVENT OUT
PD5	-	-	-	-	-	-	USART2_TX	-	TIMx_IC2	EVENT OUT
PD6	-	-	-	-	-	-	USART2_RX	-	TIMx_IC3	EVENT OUT
PD7	-	-	-	TIM9_CH2	-	-	USART2_CK	-	TIMx_IC4	EVENT OUT
PD8	-	-	-	-	-	-	USART3_TX	SEG28	TIMx_IC1	EVENT OUT
PD9	-	-	-	-	-	-	USART3_RX	SEG29	TIMx_IC2	EVENT OUT
PD10	-	-	-	-	-	-	USART3_CK	SEG30	TIMx_IC3	EVENT OUT
PD11	-	-	-	-	-	-	USART3_CTS	SEG31	TIMx_IC4	EVENT OUT
PD12	-	-	TIM4_CH1	-	-	-	USART3_RTS	SEG32	-	-
PD13	-	-	TIM4_CH2	-	-	-	-	SEG33	TIMx_IC2	EVENT OUT
PD14	-	-	TIM4_CH3	-	-	-	-	SEG34	TIMx_IC3	EVENT OUT
PD15	-	-	TIM4_CH4	-	-	-	-	SEG35	TIMx_IC4	EVENT OUT
PE0	-	-	TIM4_ETR	TIM10_CH1	-	-	-	SEG36	TIMx_IC1	EVENT OUT
PE1	-	-	-	TIM11_CH1	-	-	-	SEG37	TIMx_IC2	EVENT OUT
PE2	TRACECK	-	TIM3_ETR	-	-	-	-	SEG38	TIMx_IC3	EVENT OUT
PE3	TRACED0	-	TIM3_CH1	-	-	-	-	SEG39	TIMx_IC4	EVENT OUT
PE4	TRACED1	-	TIM3_CH2	-	-	-	-	-	TIMx_IC1	EVENT OUT
PE5	TRACED2	-	-	TIM9_CH1	-	-	-	-	TIMx_IC2	EVENT OUT
PE6-WKUP3	TRACED3	-	-	TIM9_CH2	-	-	-	-	TIMx_IC3	EVENT OUT
PE7	-	-	-	-	-	-	-	-	TIMx_IC4	EVENT OUT

## Pin descriptions

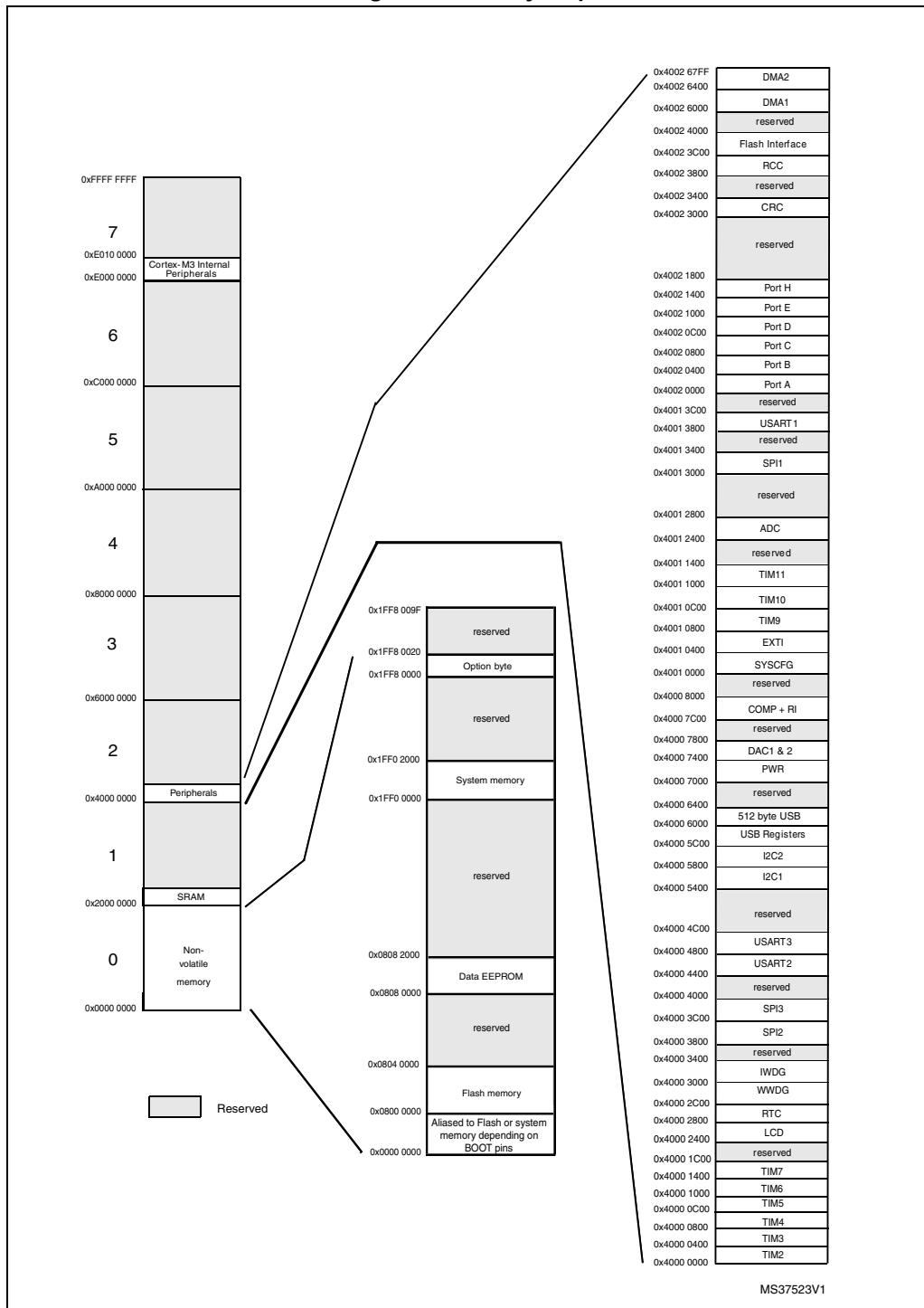
## STM32L151xC STM32L152xC

Table 10. Alternate function input/output (continued)

Port name	Digital alternate function number									
	AFIO0	AFIO1	AFIO2	AFIO3	AFIO4	AFIO5	AFIO6	AFIO7	AFIO11	AFIO14
<i>Alternate function</i>										
SYSTEM	TIM2	TIM3/4/5	TIM9/10/11	I2C1/2	SPI1/2	SPI3	USART1/2/3	LCD	CPRI	SYSTEM
PE8	-	-	-	-	-	-	-	-	-	TIMx_IC1 EVENT OUT
PE9	-	TIM2_CH1_ETR	TIM5_ETR	-	-	-	-	-	-	TIMx_IC2 EVENT OUT
PE10	-	TIM2_CH2	-	-	-	-	-	-	-	TIMx_IC3 EVENT OUT
PE11	-	TIM2_CH3	-	-	-	-	-	-	-	TIMx_IC4 EVENT OUT
PE12	-	TIM2_CH4	-	-	SPI1_NSS	-	-	-	-	TIMx_IC1 EVENT OUT
PE13	-	-	-	-	SPI1_SCK	-	-	-	-	TIMx_IC2 EVENT OUT
PE14	-	-	-	-	SPI1_MISO	-	-	-	-	TIMx_IC3 EVENT OUT
PE15	-	-	-	-	SPI1_MOSI	-	-	-	-	TIMx_IC4 EVENT OUT
PH0OSC_IN	-	-	-	-	-	-	-	-	-	-
PH1OSC_OUT	-	-	-	-	-	-	-	-	-	-
PH2	-	-	-	-	-	-	-	-	-	-

## 5 Memory mapping

Figure 9. Memory map



**Electrical characteristics****STM32L151xC STM32L152xC**

## 6 Electrical characteristics

### 6.1 Parameter conditions

Unless otherwise specified, all voltages are referenced to V<sub>SS</sub>.

#### 6.1.1 Minimum and maximum values

Unless otherwise specified the minimum and maximum values are guaranteed in the worst conditions of ambient temperature, supply voltage and frequencies by tests in production on 100% of the devices with an ambient temperature at T<sub>A</sub> = 25 °C and T<sub>A</sub> = T<sub>Amax</sub> (given by the selected temperature range).

Data based on characterization results, design simulation and/or technology characteristics are indicated in the table footnotes. Based on characterization, the minimum and maximum values refer to sample tests and represent the mean value plus or minus three times the standard deviation (mean  $\pm 3\sigma$ ).

#### 6.1.2 Typical values

Unless otherwise specified, typical data are based on T<sub>A</sub> = 25 °C, V<sub>DD</sub> = 3.6 V (for the 1.65 V  $\leq$  V<sub>DD</sub>  $\leq$  3.6 V voltage range). They are given only as design guidelines and are not tested.

Typical ADC accuracy values are determined by characterization of a batch of samples from a standard diffusion lot over the full temperature range, where 95% of the devices have an error less than or equal to the value indicated (mean  $\pm 2\sigma$ ).

#### 6.1.3 Typical curves

Unless otherwise specified, all typical curves are given only as design guidelines and are not tested.

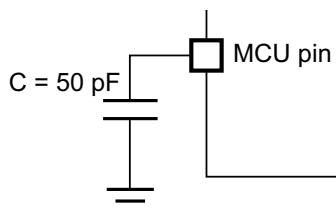
#### 6.1.4 Loading capacitor

The loading conditions used for pin parameter measurement are shown in [Figure 10](#).

#### 6.1.5 Pin input voltage

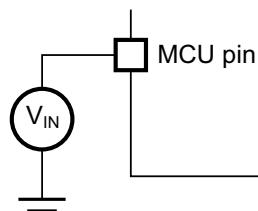
The input voltage measurement on a pin of the device is described in [Figure 11](#).

**Figure 10. Pin loading conditions**



ai17851c

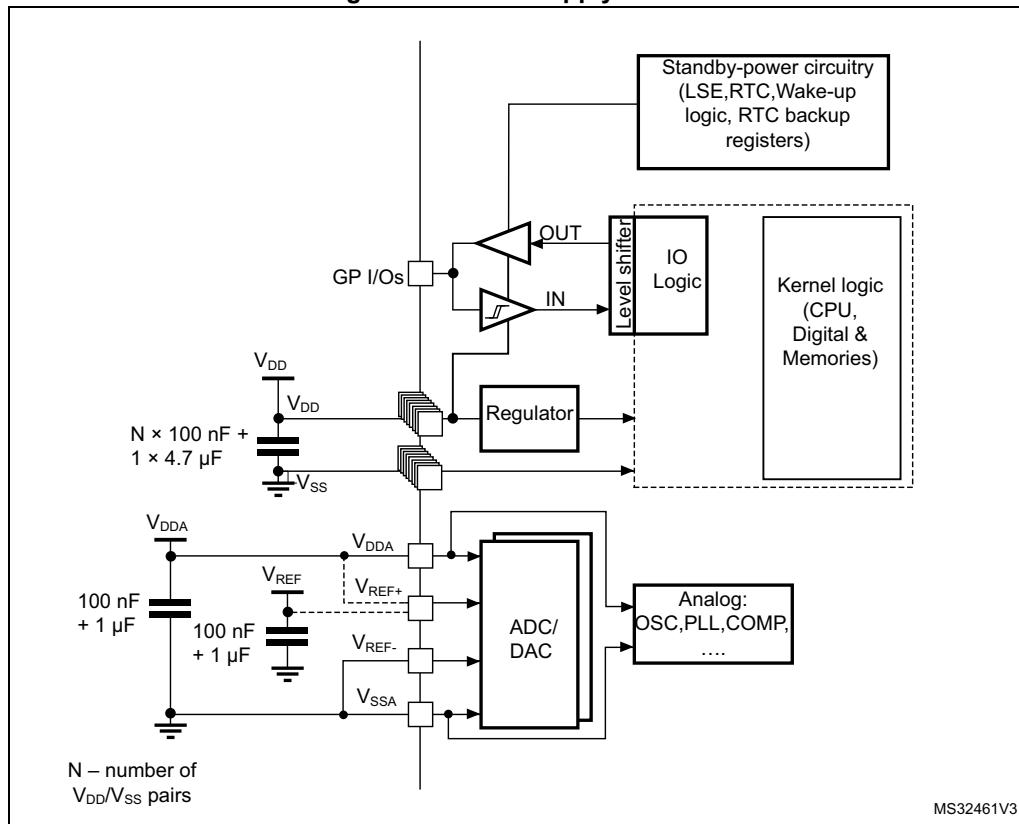
**Figure 11. Pin input voltage**

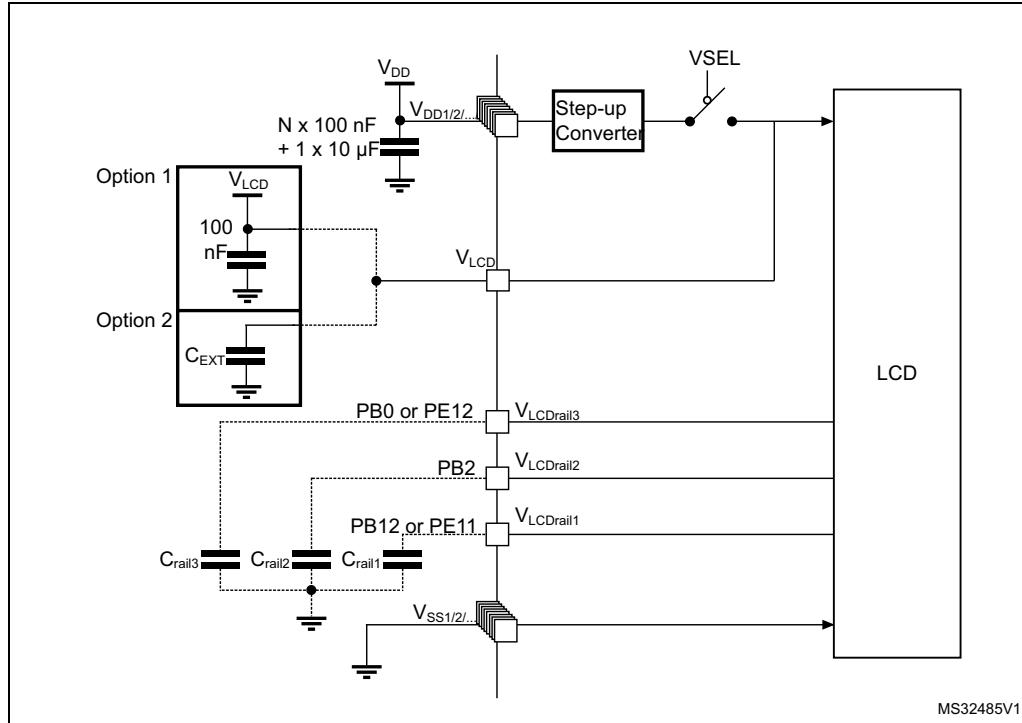


ai17852d

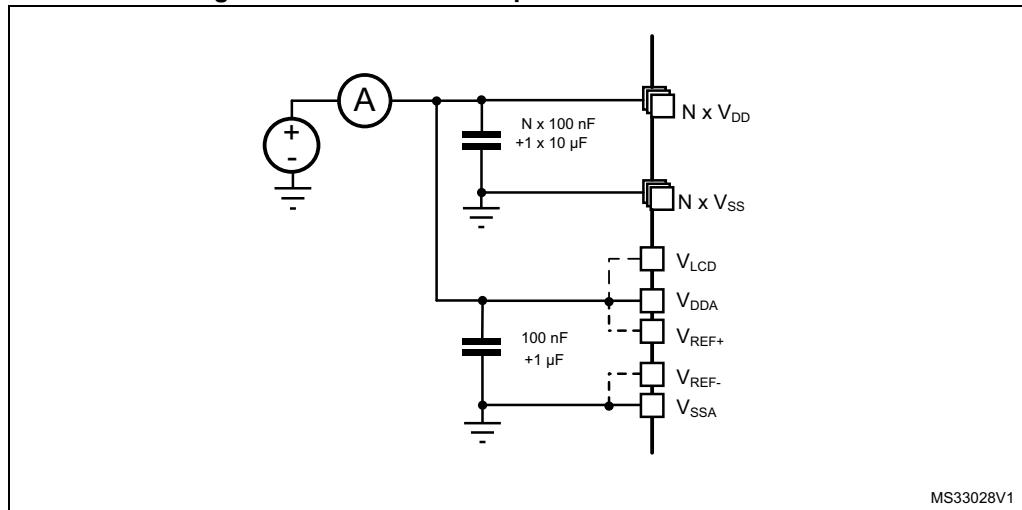
### 6.1.6 Power supply scheme

Figure 12. Power supply scheme



**Electrical characteristics****STM32L151xC STM32L152xC****6.1.7 Optional LCD power supply scheme****Figure 13. Optional LCD power supply scheme**

1. Option 1: LCD power supply is provided by a dedicated VLCD supply source, VSEL switch is open.
2. Option 2: LCD power supply is provided by the internal step-up converter, VSEL switch is closed, an external capacitance is needed for correct behavior of this converter.

**6.1.8 Current consumption measurement****Figure 14. Current consumption measurement scheme**

**STM32L151xC STM32L152xC****Electrical characteristics****6.2 Absolute maximum ratings**

Stresses above the absolute maximum ratings listed in [Table 11: Voltage characteristics](#), [Table 12: Current characteristics](#), and [Table 13: Thermal characteristics](#) may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

**Table 11. Voltage characteristics**

Symbol	Ratings	Min	Max	Unit
$V_{DD}-V_{SS}$	External main supply voltage (including $V_{DDA}$ and $V_{DD}$ ) <sup>(1)</sup>	-0.3	4.0	V
$V_{IN}^{(2)}$	Input voltage on five-volt tolerant pin	$V_{SS}-0.3$	$V_{DD}+4.0$	
	Input voltage on any other pin	$V_{SS}-0.3$	4.0	
$ \Delta V_{DDx} $	Variations between different $V_{DD}$ power pins	-	50	mV
$ V_{SSX}-V_{SSL} $	Variations between all different ground pins <sup>(3)</sup>	-	50	
$V_{REF+}-V_{DDA}$	Allowed voltage difference for $V_{REF+} > V_{DDA}$	-	0.4	
$V_{ESD(HBM)}$	Electrostatic discharge voltage (human body model)	see <a href="#">Section 6.3.11</a>		

1. All main power ( $V_{DD}$ ,  $V_{DDA}$ ) and ground ( $V_{SS}$ ,  $V_{SSA}$ ) pins must always be connected to the external power supply, in the permitted range.
2.  $V_{IN}$  maximum must always be respected. Refer to [Table 12](#) for maximum allowed injected current values.
3. Include  $V_{REF-}$  pin.

**Table 12. Current characteristics**

Symbol	Ratings	Max.	Unit
$I_{VDD(\Sigma)}$	Total current into sum of all $V_{DD\_x}$ power lines (source) <sup>(1)</sup>	100	mA
$I_{VSS(\Sigma)}^{(2)}$	Total current out of sum of all $V_{SS\_x}$ ground lines (sink) <sup>(1)</sup>	100	
$I_{VDD(PIN)}$	Maximum current into each $V_{DD\_x}$ power pin (source) <sup>(1)</sup>	70	
$I_{VSS(PIN)}$	Maximum current out of each $V_{SS\_x}$ ground pin (sink) <sup>(1)</sup>	-70	
$I_{IO}$	Output current sunk by any I/O and control pin	25	
	Output current sourced by any I/O and control pin	-25	
$\Sigma I_{IO(PIN)}$	Total output current sunk by sum of all IOs and control pins <sup>(2)</sup>	60	
	Total output current sourced by sum of all IOs and control pins <sup>(2)</sup>	-60	
$I_{INJ(PIN)}^{(3)}$	Injected current on five-volt tolerant I/O <sup>(4)</sup> , RST and B pins	-5/+0	
	Injected current on any other pin <sup>(5)</sup>	$\pm 5$	
$\Sigma I_{INJ(PIN)}$	Total injected current (sum of all I/O and control pins) <sup>(6)</sup>	$\pm 25$	

1. All main power ( $V_{DD}$ ,  $V_{DDA}$ ) and ground ( $V_{SS}$ ,  $V_{SSA}$ ) pins must always be connected to the external power supply, in the permitted range.
2. This current consumption must be correctly distributed over all IOs and control pins. The total output current must not be sunk/sourced between two consecutive power supply pins referring to high pin count LQFP packages.
3. Negative injection disturbs the analog performance of the device. See note in [Section 6.3.17](#).

**Electrical characteristics****STM32L151xC STM32L152xC**

4. Positive current injection is not possible on these I/Os. A negative injection is induced by  $V_{IN} < V_{SS}$ .  $I_{INJ(PIN)}$  must never be exceeded. Refer to [Table 11](#) for maximum allowed input voltage values.
5. A positive injection is induced by  $V_{IN} > V_{DD}$  while a negative injection is induced by  $V_{IN} < V_{SS}$ .  $I_{INJ(PIN)}$  must never be exceeded. Refer to [Table 11: Voltage characteristics](#) for the maximum allowed input voltage values.
6. When several inputs are submitted to a current injection, the maximum  $\Sigma I_{INJ(PIN)}$  is the absolute sum of the positive and negative injected currents (instantaneous values).

**Table 13. Thermal characteristics**

Symbol	Ratings	Value	Unit
$T_{STG}$	Storage temperature range	-65 to +150	°C
$T_J$	Maximum junction temperature	150	°C

## 6.3 Operating conditions

### 6.3.1 General operating conditions

**Table 14. General operating conditions**

Symbol	Parameter	Conditions	Min	Max	Unit
$f_{HCLK}$	Internal AHB clock frequency	-	0	32	MHz
$f_{PCLK1}$	Internal APB1 clock frequency		0	32	
$f_{PCLK2}$	Internal APB2 clock frequency		0	32	
$V_{DD}$	Standard operating voltage	BOR detector disabled	1.65	3.6	V
		BOR detector enabled, at power on	1.8	3.6	
		BOR detector disabled, after power on	1.65	3.6	
$V_{DDA}^{(1)}$	Analog operating voltage (ADC and DAC not used)	Must be the same voltage as $V_{DD}^{(2)}$	1.65	3.6	V
	Analog operating voltage (ADC or DAC used)		1.8	3.6	
$V_{IN}$	I/O input voltage	FT pins; $2.0 \text{ V} \leq V_{DD}$	-0.3	5.5 <sup>(3)</sup>	V
		FT pins; $V_{DD} < 2.0 \text{ V}$	-0.3	5.25 <sup>(3)</sup>	
		BOOT0 pin	0	5.5	
		Any other pin	-0.3	$V_{DD}+0.3$	
$P_D$	Power dissipation at $TA = 85 \text{ }^\circ\text{C}$ for suffix 6 or $TA = 105 \text{ }^\circ\text{C}$ for suffix 7 <sup>(4)</sup>	LQFP48 package	-	364	mW
		LQFP100 package	-	465	
		LQFP64 package	-	435	
		UFQFPN48 package	-	625	
		UFBGA100	-	339	
		WLCSP63 package	-	408	

**STM32L151xC STM32L152xC****Electrical characteristics****Table 14. General operating conditions (continued)**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Max</b>	<b>Unit</b>
$T_A$	Ambient temperature for 6 suffix version	Maximum power dissipation <sup>(5)</sup>	-40	85	$^{\circ}\text{C}$
	Ambient temperature for 7 suffix version	Maximum power dissipation	-40	105	
$T_J$	Junction temperature range	6 suffix version	-40	105	$^{\circ}\text{C}$
		7 suffix version	-40	110	

1. When the ADC is used, refer to [Table 56: ADC characteristics](#).
2. It is recommended to power  $V_{DD}$  and  $V_{DDA}$  from the same source. A maximum difference of 300 mV between  $V_{DD}$  and  $V_{DDA}$  can be tolerated during power-up and up to 140 mV in operation.
3. To sustain a voltage higher than  $VDD+0.3\text{V}$ , the internal pull-up/pull-down resistors must be disabled.
4. If  $T_A$  is lower, higher  $P_D$  values are allowed as long as  $T_J$  does not exceed  $T_J$  max (see [Table 73: Thermal characteristics on page 128](#)).
5. In low-power dissipation state,  $T_A$  can be extended to -40°C to 105°C temperature range as long as  $T_J$  does not exceed  $T_J$  max (see [Table 73: Thermal characteristics on page 128](#)).

**6.3.2 Embedded reset and power control block characteristics**

The parameters given in the following table are derived from the tests performed under the conditions summarized in [Table 14](#).

**Table 15. Embedded reset and power control block characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Typ</b>	<b>Max</b>	<b>Unit</b>
$t_{VDD}^{(1)}$	$V_{DD}$ rise time rate	BOR detector enabled	0	-	$\infty$	$\mu\text{s}/\text{V}$
		BOR detector disabled	0	-	1000	
	$V_{DD}$ fall time rate	BOR detector enabled	20	-	$\infty$	
		BOR detector disabled	0	-	1000	
$T_{RSTTEMPO}^{(1)}$	Reset temporization	$V_{DD}$ rising, BOR enabled	-	2	3.3	$\text{ms}$
		$V_{DD}$ rising, BOR disabled <sup>(2)</sup>	0.4	0.7	1.6	
$V_{POR/PDR}$	Power on/power down reset threshold	Falling edge	1	1.5	1.65	$\text{V}$
		Rising edge	1.3	1.5	1.65	
$V_{BOR0}$	Brown-out reset threshold 0	Falling edge	1.67	1.7	1.74	
		Rising edge	1.69	1.76	1.8	
$V_{BOR1}$	Brown-out reset threshold 1	Falling edge	1.87	1.93	1.97	
		Rising edge	1.96	2.03	2.07	
$V_{BOR2}$	Brown-out reset threshold 2	Falling edge	2.22	2.30	2.35	
		Rising edge	2.31	2.41	2.44	

**Electrical characteristics****STM32L151xC STM32L152xC****Table 15. Embedded reset and power control block characteristics (continued)**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Typ</b>	<b>Max</b>	<b>Unit</b>
$V_{BOR3}$	Brown-out reset threshold 3	Falling edge	2.45	2.55	2.6	V
		Rising edge	2.54	2.66	2.7	
$V_{BOR4}$	Brown-out reset threshold 4	Falling edge	2.68	2.8	2.85	V
		Rising edge	2.78	2.9	2.95	
$V_{PVD0}$	Programmable voltage detector threshold 0	Falling edge	1.8	1.85	1.88	mV
		Rising edge	1.88	1.94	1.99	
$V_{PVD1}$	PVD threshold 1	Falling edge	1.98	2.04	2.09	mV
		Rising edge	2.08	2.14	2.18	
$V_{PVD2}$	PVD threshold 2	Falling edge	2.20	2.24	2.28	mV
		Rising edge	2.28	2.34	2.38	
$V_{PVD3}$	PVD threshold 3	Falling edge	2.39	2.44	2.48	mV
		Rising edge	2.47	2.54	2.58	
$V_{PVD4}$	PVD threshold 4	Falling edge	2.57	2.64	2.69	mV
		Rising edge	2.68	2.74	2.79	
$V_{PVD5}$	PVD threshold 5	Falling edge	2.77	2.83	2.88	mV
		Rising edge	2.87	2.94	2.99	
$V_{PVD6}$	PVD threshold 6	Falling edge	2.97	3.05	3.09	mV
		Rising edge	3.08	3.15	3.20	
$V_{hyst}$	Hysteresis voltage	BOR0 threshold	-	40	-	mV
		All BOR and PVD thresholds excepting BOR0	-	100	-	

1. Guaranteed by characterization results.

2. Valid for device version without BOR at power up. Please see option "D" in Ordering information scheme for more details.

**STM32L151xC STM32L152xC****Electrical characteristics****6.3.3 Embedded internal reference voltage**

The parameters given in [Table 17](#) are based on characterization results, unless otherwise specified.

**Table 16. Embedded internal reference voltage calibration values**

Calibration value name	Description	Memory address
VREFINT_CAL	Raw data acquired at temperature of $30^{\circ}\text{C} \pm 5^{\circ}\text{C}$ $V_{DDA} = 3\text{ V} \pm 10\text{ mV}$	0x1FF8 00F8 - 0x1FF8 00F9

**Table 17. Embedded internal reference voltage**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{\text{REFINT out}}^{(1)}$	Internal reference voltage	$-40^{\circ}\text{C} < T_J < +110^{\circ}\text{C}$	1.202	1.224	1.242	V
$I_{\text{REFINT}}$	Internal reference current consumption	-	-	1.4	2.3	$\mu\text{A}$
$T_{\text{VREFINT}}$	Internal reference startup time	-	-	2	3	ms
$V_{\text{VREF_MEAS}}$	$V_{DDA}$ and $V_{\text{REF+}}$ voltage during $V_{\text{REFINT}}$ factory measure	-	2.99	3	3.01	V
$A_{\text{VREF_MEAS}}$	Accuracy of factory-measured $V_{\text{REF}}$ value <sup>(2)</sup>	Including uncertainties due to ADC and $V_{DDA}/V_{\text{REF+}}$ values	-	-	$\pm 5$	mV
$T_{\text{Coeff}}^{(3)}$	Temperature coefficient	$-40^{\circ}\text{C} < T_J < +110^{\circ}\text{C}$	-	25	100	$\text{ppm}/^{\circ}\text{C}$
$A_{\text{Coeff}}^{(3)}$	Long-term stability	1000 hours, $T = 25^{\circ}\text{C}$	-	-	1000	ppm
$V_{\text{DDCoeff}}^{(3)}$	Voltage coefficient	$3.0\text{ V} < V_{DDA} < 3.6\text{ V}$	-	-	2000	ppm/V
$T_{S\_vrefint}^{(3)}$	ADC sampling time when reading the internal reference voltage	-	4	-	-	$\mu\text{s}$
$T_{\text{ADC_BUF}}^{(3)(4)}$	Startup time of reference voltage buffer for ADC	-	-	-	10	$\mu\text{s}$
$I_{\text{BUF_ADC}}^{(3)}$	Consumption of reference voltage buffer for ADC	-	-	13.5	25	$\mu\text{A}$
$I_{\text{VREF_OUT}}^{(3)}$	$V_{\text{REF\_OUT}}$ output current <sup>(5)</sup>	-	-	-	1	$\mu\text{A}$
$C_{\text{VREF_OUT}}^{(3)}$	$V_{\text{REF\_OUT}}$ output load	-	-	-	50	pF
$I_{\text{LPBUF}}^{(3)}$	Consumption of reference voltage buffer for $V_{\text{REF\_OUT}}$ and COMP	-	-	730	1200	nA
$V_{\text{REFINT_DIV1}}^{(3)}$	1/4 reference voltage	-	24	25	26	% $V_{\text{REFIN}}$ T
$V_{\text{REFINT_DIV2}}^{(3)}$	1/2 reference voltage	-	49	50	51	
$V_{\text{REFINT_DIV3}}^{(3)}$	3/4 reference voltage	-	74	75	76	

1. Guaranteed by test in production.
2. The internal  $V_{\text{REF}}$  value is individually measured in production and stored in dedicated EEPROM bytes.
3. Guaranteed by characterization results.
4. Shortest sampling time can be determined in the application by multiple iterations.

**Electrical characteristics****STM32L151xC STM32L152xC**

5. To guarantee less than 1% VREF\_OUT deviation.

### 6.3.4 Supply current characteristics

The current consumption is a function of several parameters and factors such as the operating voltage, temperature, I/O pin loading, device software configuration, operating frequencies, I/O pin switching rate, program location in memory and executed binary code. The current consumption is measured as described in [Figure 14: Current consumption measurement scheme](#).

All Run-mode current consumption measurements given in this section are performed with a reduced code that gives a consumption equivalent to the Dhrystone 2.1 code, unless otherwise specified. The current consumption values are derived from tests performed under ambient temperature  $T_A = 25^\circ\text{C}$  and  $V_{DD}$  supply voltage conditions summarized in [Table 14: General operating conditions](#), unless otherwise specified.

The MCU is placed under the following conditions:

- All I/O pins are configured in analog input mode
- All peripherals are disabled except when explicitly mentioned.
- The Flash memory access time, 64-bit access and prefetch is adjusted depending on  $f_{HCLK}$  frequency and voltage range to provide the best CPU performance.
- When the peripherals are enabled  $f_{APB1} = f_{APB2} = f_{AHB}$ .
- When PLL is ON, the PLL inputs are equal to HSI = 16 MHz (if internal clock is used) or HSE = 16 MHz (if HSE bypass mode is used).
- The HSE user clock applied to OSCI\_IN input follows the characteristic specified in [Table 27: High-speed external user clock characteristics](#).
- For maximum current consumption  $V_{DD} = V_{DDA} = 3.6\text{ V}$  is applied to all supply pins.
- For typical current consumption  $V_{DD} = V_{DDA} = 3.0\text{ V}$  is applied to all supply pins if not specified otherwise.

## STM32L151xC STM32L152xC

## Electrical characteristics

**Table 18. Current consumption in Run mode, code with data processing running from Flash**

Symbol	Parameter	Conditions	f <sub>HCLK</sub>	Typ	Max <sup>(1)</sup>	Unit	
I <sub>DD</sub> (Run from Flash)	Supply current in Run mode, code executed from Flash	f <sub>HSE</sub> = f <sub>HCLK</sub> up to 16 MHz included, f <sub>HSE</sub> = f <sub>HCLK</sub> /2 above 16 MHz (PLL ON) <sup>(2)</sup>	Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	1 MHz	215	400	µA
			Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	2 MHz	400	600	
			Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	4 MHz	725	960	
		HSI clock source (16 MHz)	Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	8 MHz	0.915	1.1	mA
			Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	16 MHz	1.75	2.1	
			Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	32 MHz	3.4	3.9	
		MSI clock, 65 kHz	Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	8 MHz	4.2	4.9	
			Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	16 MHz	8.2	9.6	
			Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	32 MHz	40.5	110	
		MSI clock, 524 kHz	524 kHz	4.2 MHz	125	190	µA
		MSI clock, 4.2 MHz	4.2 MHz	775	900	900	

1. Guaranteed by characterization results, unless otherwise specified.

2. Oscillator bypassed (HSEBYP = 1 in RCC\_CR register).

**Electrical characteristics****STM32L151xC STM32L152xC****Table 19. Current consumption in Run mode, code with data processing running from RAM**

Symbol	Parameter	Conditions	f <sub>HCLK</sub>	Typ	Max <sup>(1)</sup>	Unit	
I <sub>DD</sub> (Run from RAM)	Supply current in Run mode, code executed from RAM, Flash switched off	f <sub>HSE</sub> = f <sub>HCLK</sub> up to 16 MHz, included f <sub>HSE</sub> = f <sub>HCLK</sub> /2 above 16 MHz (PLL ON) <sup>(2)</sup>	Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	1 MHz	185	240	μA
			2 MHz	345	410		
			4 MHz	645	880 <sup>(3)</sup>		
		Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	4 MHz	0.755	1.4	mA	
			8 MHz	1.5	2.1		
			16 MHz	3	3.5		
		Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	8 MHz	1.8	2.8		
			16 MHz	3.6	4.1		
			32 MHz	7.15	8.3		
		HSI clock source (16 MHz)	Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	16 MHz	2.95	3.5	
			Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	32 MHz	7.15	8.4	
		MSI clock, 65 kHz	Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	65 kHz	38.5	85	μA
		MSI clock, 524 kHz		524 kHz	110	160	
		MSI clock, 4.2 MHz		4.2 MHz	690	810	

1. Guaranteed by characterization results, unless otherwise specified.

2. Oscillator bypassed (HSEBYP = 1 in RCC\_CR register).

3. Guaranteed by test in production.

## STM32L151xC STM32L152xC

## Electrical characteristics

Table 20. Current consumption in Sleep mode

Symbol	Parameter	Conditions	f <sub>HCLK</sub>	Typ	Max <sup>(1)</sup>	Unit	
I <sub>DD</sub> (Sleep)	Supply current in Sleep mode, Flash OFF	$f_{HSE} = f_{HCLK}$ up to 16 MHz included, $f_{HSE} = f_{HCLK}/2$ above 16 MHz (PLL ON) <sup>(2)</sup>	Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	1 MHz	50	130	
				2 MHz	78.5	195	
				4 MHz	140	310	
			Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	4 MHz	165	310	
				8 MHz	310	440	
				16 MHz	590	830	
			Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	8 MHz	350	550	
				16 MHz	680	990	
				32 MHz	1600	2100	
	HSI clock source (16 MHz)		Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	16 MHz	640	890	
			Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	32 MHz	1600	2200	
			MSI clock, 65 kHz	65 kHz	19	60	
	MSI clock, 524 kHz	$f_{HSE} = f_{HCLK}$ up to 16 MHz included, $f_{HSE} = f_{HCLK}/2$ above 16 MHz (PLL ON) <sup>(2)</sup>	Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	524 kHz	33	99	
			4.2 MHz	145	210		
			MSI clock, 4.2 MHz	19	60		
	Supply current in Sleep mode, Flash ON	$f_{HSE} = f_{HCLK}$ up to 16 MHz included, $f_{HSE} = f_{HCLK}/2$ above 16 MHz (PLL ON) <sup>(2)</sup>	Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	1 MHz	60.5	130	
				2 MHz	89.5	190	
				4 MHz	150	320	
			Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	4 MHz	180	320	
				8 MHz	320	460	
				16 MHz	605	840	
			Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	8 MHz	380	540	
				16 MHz	695	1000	
				32 MHz	1600	2100	
	HSI clock source (16 MHz)		Range 2, V <sub>CORE</sub> =1.5 V VOS[1:0] = 10	16 MHz	650	910	
			Range 1, V <sub>CORE</sub> =1.8 V VOS[1:0] = 01	32 MHz	1600	2200	
			MSI clock, 65 kHz	65 kHz	30	90	
	Supply current in Sleep mode, Flash ON	$f_{HSE} = f_{HCLK}$ up to 16 MHz included, $f_{HSE} = f_{HCLK}/2$ above 16 MHz (PLL ON) <sup>(2)</sup>	Range 3, V <sub>CORE</sub> =1.2 V VOS[1:0] = 11	524 kHz	44	96	
			4.2 MHz	155	220		
			MSI clock, 4.2 MHz	4.2 MHz	90		

1. Guaranteed by characterization results, unless otherwise specified.

2. Oscillator bypassed (HSEBYP = 1 in RCC\_CR register)

**Electrical characteristics****STM32L151xC STM32L152xC****Table 21. Current consumption in Low-power run mode**

Symbol	Parameter	Conditions			Typ	Max <sup>(1)</sup>	Unit
$I_{DD}$ (LP Run)	Supply current in Low-power run mode	All peripherals OFF, code executed from RAM, Flash switched OFF, $V_{DD}$ from 1.65 V to 3.6 V	MSI clock, 65 kHz $f_{HCLK} = 32$ kHz	$T_A = -40$ °C to 25 °C	8.6	12	μA
				$T_A = 85$ °C	19	25	
				$T_A = 105$ °C	35	47	
		MSI clock, 65 kHz $f_{HCLK} = 65$ kHz	MSI clock, 65 kHz $f_{HCLK} = 65$ kHz	$T_A = -40$ °C to 25 °C	14	16	
				$T_A = 85$ °C	24	29	
				$T_A = 105$ °C	40	51	
		MSI clock, 131 kHz $f_{HCLK} = 131$ kHz	MSI clock, 131 kHz $f_{HCLK} = 131$ kHz	$T_A = -40$ °C to 25 °C	26	29	
				$T_A = 55$ °C	28	31	
				$T_A = 85$ °C	36	42	
				$T_A = 105$ °C	52	64	
		All peripherals OFF, code executed from Flash, $V_{DD}$ from 1.65 V to 3.6 V	MSI clock, 65 kHz $f_{HCLK} = 32$ kHz	$T_A = -40$ °C to 25 °C	20	24	
				$T_A = 85$ °C	32	37	
				$T_A = 105$ °C	49	61	
		MSI clock, 65 kHz $f_{HCLK} = 65$ kHz	MSI clock, 65 kHz $f_{HCLK} = 65$ kHz	$T_A = -40$ °C to 25 °C	26	30	
				$T_A = 85$ °C	38	44	
				$T_A = 105$ °C	55	67	
		MSI clock, 131 kHz $f_{HCLK} = 131$ kHz	MSI clock, 131 kHz $f_{HCLK} = 131$ kHz	$T_A = -40$ °C to 25 °C	41	46	
				$T_A = 55$ °C	44	50	
				$T_A = 85$ °C	56	87	
				$T_A = 105$ °C	73	110	
$I_{DD}$ max (LP Run)	Max allowed current in Low-power run mode	$V_{DD}$ from 1.65 V to 3.6 V	-	-	-	200	

1. Guaranteed by characterization results, unless otherwise specified.

## STM32L151xC STM32L152xC

## Electrical characteristics

Table 22. Current consumption in Low-power sleep mode

Symbol	Parameter	Conditions		Typ	Max <sup>(1)</sup>	Unit
$I_{DD}$ (LP Sleep)	Supply current in Low-power sleep mode	All peripherals OFF, $V_{DD}$ from 1.65 V to 3.6 V  MSI clock, 65 kHz $f_{HCLK} = 32$ kHz Flash OFF	$T_A = -40$ °C to 25 °C	4.4	-	µA
			$T_A = -40$ °C to 25 °C	14	16	
			$T_A = 85$ °C	19	23	
			$T_A = 105$ °C	27	33	
			$T_A = -40$ °C to 25 °C	15	17	
			$T_A = 85$ °C	20	23	
			$T_A = 105$ °C	28	33	
		MSI clock, 131 kHz $f_{HCLK} = 131$ kHz, Flash ON	$T_A = -40$ °C to 25 °C	17	19	
			$T_A = 55$ °C	18	21	
			$T_A = 85$ °C	22	25	
			$T_A = 105$ °C	30	35	
			$T_A = -40$ °C to 25 °C	14	16	
		MSI clock, 65 kHz $f_{HCLK} = 32$ kHz	$T_A = 85$ °C	19	22	
			$T_A = 105$ °C	27	32	
			$T_A = -40$ °C to 25 °C	15	17	
		MSI clock, 65 kHz $f_{HCLK} = 65$ kHz	$T_A = 85$ °C	20	23	
			$T_A = 105$ °C	28	33	
			$T_A = -40$ °C to 25 °C	17	19	
		MSI clock, 131 kHz $f_{HCLK} = 131$ kHz	$T_A = 55$ °C	18	21	
			$T_A = 85$ °C	22	25	
			$T_A = 105$ °C	30	36	
			-	-	-	
$I_{DD \max}$ (LP Sleep)	Max allowed current in Low-power sleep mode	$V_{DD}$ from 1.65 V to 3.6 V	-	-	-	200

1. Guaranteed by characterization results, unless otherwise specified.

## Electrical characteristics

## STM32L151xC STM32L152xC

Table 23. Typical and maximum current consumptions in Stop mode

Symbol	Parameter	Conditions		Typ	Max <sup>(1)</sup>	Unit
$I_{DD}$ (Stop with RTC)	Supply current in Stop mode with RTC enabled	RTC clocked by LSI or LSE external clock (32.768kHz), regulator in LP mode, HSI and HSE OFF (no independent watchdog)	LCD OFF	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$ $V_{DD} = 1.8 \text{ V}$	1.15	-
				$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	1.4	-
				$T_A = 55^\circ\text{C}$	2	-
				$T_A = 85^\circ\text{C}$	3.4	10
				$T_A = 105^\circ\text{C}$	6.35	23
			LCD ON (static duty) <sup>(2)</sup>	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	1.55	6
				$T_A = 55^\circ\text{C}$	2.15	7
				$T_A = 85^\circ\text{C}$	3.55	12
				$T_A = 105^\circ\text{C}$	6.3	27
			LCD ON (1/8 duty) <sup>(3)</sup>	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	3.9	10
				$T_A = 55^\circ\text{C}$	4.65	11
				$T_A = 85^\circ\text{C}$	6.25	16
				$T_A = 105^\circ\text{C}$	9.1	44
			LCD OFF	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	1.5	-
				$T_A = 55^\circ\text{C}$	2.15	-
				$T_A = 85^\circ\text{C}$	3.7	-
				$T_A = 105^\circ\text{C}$	6.75	-
			LCD ON (static duty) <sup>(2)</sup>	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	1.6	-
				$T_A = 55^\circ\text{C}$	2.3	-
				$T_A = 85^\circ\text{C}$	3.8	-
				$T_A = 105^\circ\text{C}$	6.85	-
			LCD ON (1/8 duty) <sup>(3)</sup>	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	4	-
				$T_A = 55^\circ\text{C}$	4.85	-
				$T_A = 85^\circ\text{C}$	6.5	-
				$T_A = 105^\circ\text{C}$	9.1	-
			LCD OFF	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$ $V_{DD} = 1.8\text{V}$	1.2	-
				$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$ $V_{DD} = 3.0\text{V}$	1.5	-
				$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$ $V_{DD} = 3.6\text{V}$	1.75	-

**STM32L151xC STM32L152xC****Electrical characteristics****Table 23. Typical and maximum current consumptions in Stop mode (continued)**

Symbol	Parameter	Conditions	Typ	Max <sup>(1)</sup>	Unit
$I_{DD}$ (Stop)	Supply current in Stop mode (RTC disabled)	Regulator in LP mode, HSI and HSE OFF, independent watchdog and LSI enabled	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	1.8	2.2
		Regulator in LP mode, LSI, HSI and HSE OFF (no independent watchdog)	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	0.435	1
			$T_A = 55^\circ\text{C}$	0.99	3
			$T_A = 85^\circ\text{C}$	2.4	9
			$T_A = 105^\circ\text{C}$	5.5	22 <sup>(5)</sup>
$I_{DD}$ (WU from Stop)	Supply current during wakeup from Stop mode	MSI = 4.2 MHz	$T_A = -40^\circ\text{C} \text{ to } 25^\circ\text{C}$	2	-
		MSI = 1.05 MHz		1.45	-
		MSI = 65 kHz <sup>(6)</sup>		1.45	-

1. Guaranteed by characterization results, unless otherwise specified.
2. LCD enabled with external VLCD, static duty, division ratio = 256, all pixels active, no LCD connected.
3. LCD enabled with external VLCD, 1/8 duty, 1/3 bias, division ratio = 64, all pixels active, no LCD connected.
4. Based on characterization done with a 32.768 kHz crystal (MC306-G-06Q-32.768, manufacturer JFVNY) with two 6.8 pF loading capacitors.
5. Guaranteed by test in production.
6. When MSI = 64 kHz, the RMS current is measured over the first 15  $\mu\text{s}$  following the wakeup event. For the remaining part of the wakeup period, the current corresponds the Run mode current.

**Electrical characteristics****STM32L151xC STM32L152xC****Table 24. Typical and maximum current consumptions in Standby mode**

Symbol	Parameter	Conditions	Typ	Max <sup>(1)</sup>	Unit	
$I_{DD}$ (Standby with RTC)	Supply current in Standby mode with RTC enabled	RTC clocked by LSI (no independent watchdog)	$T_A = -40^\circ\text{C}$ to $25^\circ\text{C}$ $V_{DD} = 1.8 \text{ V}$	0.905	-	
			$T_A = -40^\circ\text{C}$ to $25^\circ\text{C}$	1.15	1.9	
			$T_A = 55^\circ\text{C}$	1.5	2.2	
			$T_A = 85^\circ\text{C}$	1.75	4	
			$T_A = 105^\circ\text{C}$	2.1	8.3 <sup>(2)</sup>	
		RTC clocked by LSE external quartz (no independent watchdog) <sup>(3)</sup>	$T_A = -40^\circ\text{C}$ to $25^\circ\text{C}$ $V_{DD} = 1.8 \text{ V}$	0.98	-	
			$T_A = -40^\circ\text{C}$ to $25^\circ\text{C}$	1.3	-	
			$T_A = 55^\circ\text{C}$	1.7	-	
			$T_A = 85^\circ\text{C}$	2.05	-	
			$T_A = 105^\circ\text{C}$	2.45	-	
$I_{DD}$ (Standby)	Supply current in Standby mode (RTC disabled)	Independent watchdog and LSI enabled	$T_A = -40^\circ\text{C}$ to $25^\circ\text{C}$	1	1.7	
		Independent watchdog and LSI OFF	$T_A = -40^\circ\text{C}$ to $25^\circ\text{C}$	0.29	0.6	
			$T_A = 55^\circ\text{C}$	0.345	0.9	
			$T_A = 85^\circ\text{C}$	0.575	2.75	
			$T_A = 105^\circ\text{C}$	1.45	7 <sup>(2)</sup>	
$I_{DD}$ (WU from Standby)	Supply current during wakeup time from Standby mode	-	$T_A = -40^\circ\text{C}$ to $25^\circ\text{C}$	1	-	mA

1. Guaranteed by characterization results, unless otherwise specified.

2. Guaranteed by test in production.

3. Based on characterization done with a 32.768 kHz crystal (MC306-G-06Q-32.768, manufacturer JFVNY) with two 6.8pF loading capacitors.

**On-chip peripheral current consumption**

The current consumption of the on-chip peripherals is given in the following table. The MCU is placed under the following conditions:

- all I/O pins are in input mode with a static value at  $V_{DD}$  or  $V_{SS}$  (no load)
- all peripherals are disabled unless otherwise mentioned
- the given value is calculated by measuring the current consumption
  - with all peripherals clocked off
  - with only one peripheral clocked on

## STM32L151xC STM32L152xC

## Electrical characteristics

Table 25. Peripheral current consumption<sup>(1)</sup>

Peripheral	Typical consumption, $V_{DD} = 3.0$ V, $T_A = 25$ °C				Unit
	Range 1, $V_{CORE} = 1.8$ V $VOS[1:0] = 01$	Range 2, $V_{CORE} = 1.5$ V $VOS[1:0] = 10$	Range 3, $V_{CORE} = 1.2$ V $VOS[1:0] = 11$	Low-power sleep and run	
APB1	TIM2	11.2	8.9	7.0	8.9
	TIM3	11.2	9.0	7.1	9.0
	TIM4	12.9	10.4	8.2	10.4
	TIM5	14.4	11.5	9.0	11.5
	TIM6	4.0	3.1	2.4	3.1
	TIM7	3.8	3.0	2.3	3.0
	LCD	5.8	4.6	3.6	4.6
	WWDG	2.9	2.3	1.8	2.3
	SPI2	6.5	5.2	4.1	5.2
	SPI3	5.9	4.6	3.6	4.6
	USART2	8.8	7.0	5.5	7.0
	USART3	8.4	6.8	5.3	6.8
	I2C1	7.3	5.8	4.6	5.8
	I2C2	7.9	6.3	5.0	6.3
	USB	13.3	10.6	8.3	10.6
	PWR	2.8	2.2	1.8	2.2
	DAC	6.1	4.9	3.9	4.9
	COMP	4.8	3.8	3.0	3.8

## Electrical characteristics

## STM32L151xC STM32L152xC

Table 25. Peripheral current consumption<sup>(1)</sup> (continued)

Peripheral	Typical consumption, $V_{DD} = 3.0\text{ V}$ , $T_A = 25^\circ\text{C}$				Unit
	Range 1, $V_{CORE} = 1.8\text{ V}$ $VOS[1:0] = 01$	Range 2, $V_{CORE} = 1.5\text{ V}$ $VOS[1:0] = 10$	Range 3, $V_{CORE} = 1.2\text{ V}$ $VOS[1:0] = 11$	Low-power sleep and run	
APB2	SYSCFG & RI	2.6	2.0	1.6	2.0
	TIM9	7.9	6.4	5.0	6.4
	TIM10	5.9	4.7	3.8	4.7
	TIM11	5.9	4.6	3.7	4.6
	ADC <sup>(2)</sup>	10.5	8.3	6.6	8.3
	SPI1	4.3	3.4	2.8	3.4
	USART1	8.8	7.1	5.6	7.1
AHB	GPIOA	4.3	3.3	2.6	3.3
	GPIOB	4.3	3.5	2.8	3.5
	GPIOC	4.0	3.2	2.5	3.2
	GPIOD	4.1	3.3	2.5	3.3
	GPIOE	4.2	3.4	2.7	3.4
	GPIOH	3.7	3.0	2.3	3.0
	CRC	0.8	0.6	0.5	0.6
	FLASH	11.1	9.4	8	<sup>(3)</sup>
	DMA1	15.6	12.7	10	12.7
	DMA2	16.3	13.4	10.5	13.4
All enabled	187	154	120	144.6	
$I_{DD}$ (RTC)			0.4		$\mu\text{A}$
$I_{DD}$ (LCD)			3.1		
$I_{DD}$ (ADC) <sup>(4)</sup>			1450		
$I_{DD}$ (DAC) <sup>(5)</sup>			340		
$I_{DD}$ (COMP1)			0.16		
$I_{DD}$ (COMP2)	Slow mode		2		
	Fast mode		5		
$I_{DD}$ (PVD / BOR) <sup>(6)</sup>			2.6		
$I_{DD}$ (IWDG)			0.25		

1. Data based on differential  $I_{DD}$  measurement between all peripherals OFF and one peripheral with clock enabled, in the following conditions:  $f_{HCLK} = 32\text{ MHz}$  (range 1),  $f_{HCLK} = 16\text{ MHz}$  (range 2),  $f_{HCLK} = 4\text{ MHz}$  (range 3),  $f_{HCLK} = 64\text{ kHz}$  (Low-power run/sleep),  $f_{APB1} = f_{HCLK}$ ;  $f_{APB2} = f_{HCLK}$ ; default prescaler value for each peripheral. The CPU is in Sleep mode in both cases. No I/O pins toggling.

2. HSI oscillator is OFF for this measure.

**STM32L151xC STM32L152xC****Electrical characteristics**

3. In Low-power sleep and run mode, the Flash memory must always be in power-down mode.
4. Data based on a differential I<sub>DD</sub> measurement between ADC in reset configuration and continuous ADC conversion (HSI consumption not included).
5. Data based on a differential I<sub>DD</sub> measurement between DAC in reset configuration and continuous DAC conversion of V<sub>DD</sub>/2. DAC is in buffered mode, output is left floating.
6. Including supply current of internal reference voltage.

**6.3.5 Wakeup time from low-power mode**

The wakeup times given in the following table are measured with the MSI RC oscillator. The clock source used to wake up the device depends on the current operating mode:

- Sleep mode: the clock source is the clock that was set before entering Sleep mode
- Stop mode: the clock source is the MSI oscillator in the range configured before entering Stop mode
- Standby mode: the clock source is the MSI oscillator running at 2.1 MHz

All timings are derived from tests performed under the conditions summarized in [Table 14](#).

**Table 26. Low-power mode wakeup timings**

Symbol	Parameter	Conditions	Typ	Max <sup>(1)</sup>	Unit
t <sub>WUSLEEP</sub>	Wakeup from Sleep mode	f <sub>HCLK</sub> = 32 MHz	0.4	-	
t <sub>WUSLEEP_LP</sub>	Wakeup from Low-power sleep mode, f <sub>HCLK</sub> = 262 kHz	f <sub>HCLK</sub> = 262 kHz Flash enabled	46	-	μs
		f <sub>HCLK</sub> = 262 kHz Flash switched OFF	46	-	
t <sub>WUSTOP</sub>	Wakeup from Stop mode, regulator in Run mode ULP bit = 1 and FWU bit = 1	f <sub>HCLK</sub> = f <sub>MSI</sub> = 4.2 MHz	8.2	-	μs
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 4.2 MHz Voltage range 1 and 2	7.7	8.9	
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 4.2 MHz Voltage range 3	8.2	13.1	
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 2.1 MHz	10.2	13.4	
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 1.05 MHz	16	20	
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 524 kHz	31	37	
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 262 kHz	57	66	
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 131 kHz	112	123	
		f <sub>HCLK</sub> = f <sub>MSI</sub> = 65 kHz	221	236	
t <sub>WUSTDBY</sub>	Wakeup from Standby mode ULP bit = 1 and FWU bit = 1	f <sub>HCLK</sub> = MSI = 2.1 MHz	58	104	ms
	Wakeup from Standby mode FWU bit = 0	f <sub>HCLK</sub> = MSI = 2.1 MHz	2.6	3.25	

1. Guaranteed by characterization, unless otherwise specified

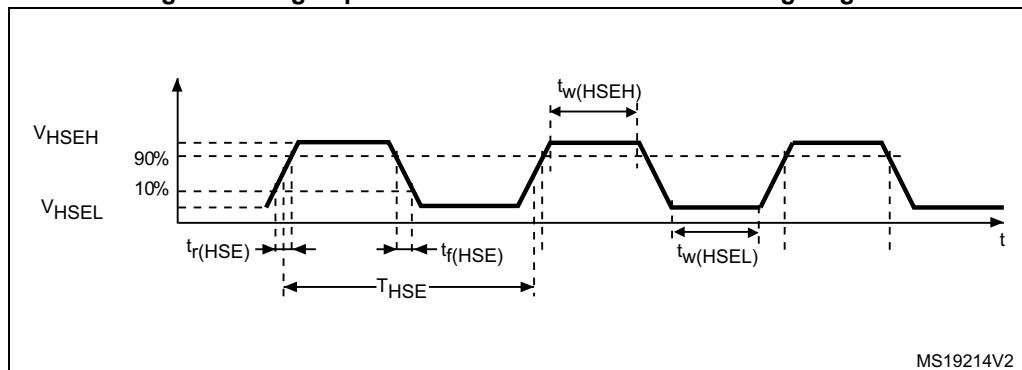
**Electrical characteristics****STM32L151xC STM32L152xC****6.3.6 External clock source characteristics****High-speed external user clock generated from an external source**

In bypass mode the HSE oscillator is switched off and the input pin is a standard GPIO. The external clock signal has to respect the I/O characteristics in [Section 6.3.12](#). However, the recommended clock input waveform is shown in [Figure 15](#).

**Table 27. High-speed external user clock characteristics<sup>(1)</sup>**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$f_{HSE\_ext}$	User external clock source frequency	CSS is on or PLL is used	1	8	32	MHz
		CSS is off, PLL not used	0	8	32	MHz
$V_{HSEH}$	OSC_IN input pin high level voltage	-	0.7V <sub>DD</sub>	-	$V_{DD}$	V
$V_{HSEL}$	OSC_IN input pin low level voltage		$V_{SS}$	-	0.3V <sub>DD</sub>	
$t_w(HSEH)$	OSC_IN high or low time		12	-	-	ns
$t_r(HSE)$	OSC_IN rise or fall time		-	-	20	
$C_{in(HSE)}$	OSC_IN input capacitance		-	2.6	-	pF

1. Guaranteed by design.

**Figure 15. High-speed external clock source AC timing diagram**

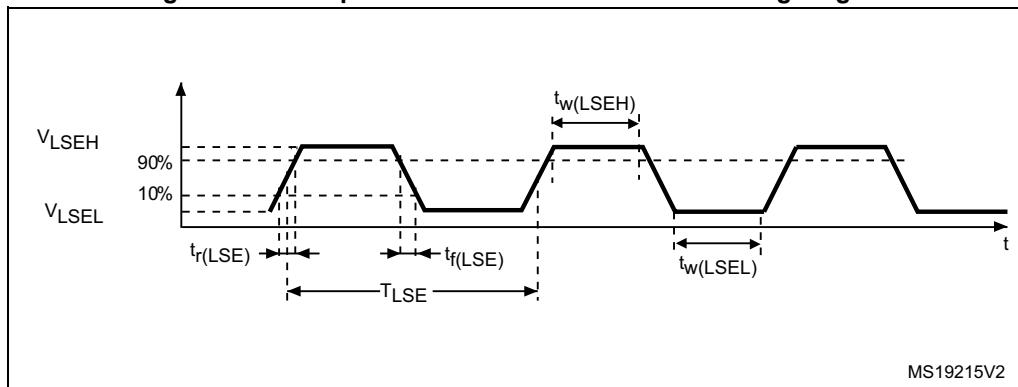
**STM32L151xC STM32L152xC****Electrical characteristics****Low-speed external user clock generated from an external source**

The characteristics given in the following table result from tests performed using a low-speed external clock source, and under the conditions summarized in [Table 14](#).

**Table 28. Low-speed external user clock characteristics<sup>(1)</sup>**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$f_{LSE\_ext}$	User external clock source frequency	-	1	32.768	1000	kHz
$V_{LSEH}$	OSC32_IN input pin high level voltage		0.7 $V_{DD}$	-	$V_{DD}$	V
$V_{LSEL}$	OSC32_IN input pin low level voltage		$V_{SS}$	-	0.3 $V_{DD}$	
$t_w(LSEH)$ $t_w(LSEL)$	OSC32_IN high or low time		465	-	-	ns
$t_r(LSE)$ $t_f(LSE)$	OSC32_IN rise or fall time		-	-	10	
$C_{IN(LSE)}$	OSC32_IN input capacitance		-	-	0.6	pF

1. Guaranteed by design.

**Figure 16. Low-speed external clock source AC timing diagram****High-speed external clock generated from a crystal/ceramic resonator**

The high-speed external (HSE) clock can be supplied with a 1 to 24 MHz crystal/ceramic resonator oscillator. All the information given in this paragraph are based on characterization results obtained with typical external components specified in [Table 29](#). In the application, the resonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and startup stabilization time. Refer to the crystal resonator manufacturer for more details on the resonator characteristics (frequency, package, accuracy).

**Electrical characteristics****STM32L151xC STM32L152xC****Table 29. HSE oscillator characteristics<sup>(1)(2)</sup>**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$f_{OSC\_IN}$	Oscillator frequency	-	1		24	MHz
$R_F$	Feedback resistor	-	-	200	-	kΩ
C	Recommended load capacitance versus equivalent serial resistance of the crystal ( $R_S$ ) <sup>(3)</sup>	$R_S = 30 \Omega$	-	20	-	pF
$I_{HSE}$	HSE driving current	$V_{DD} = 3.3 \text{ V}$ , $V_{IN} = V_{SS}$ with 30 pF load	-	-	3	mA
$I_{DD(HSE)}$	HSE oscillator power consumption	$C = 20 \text{ pF}$ $f_{OSC} = 16 \text{ MHz}$	-	-	2.5 (startup) 0.7 (stabilized)	mA
		$C = 10 \text{ pF}$ $f_{OSC} = 16 \text{ MHz}$	-	-	2.5 (startup) 0.46 (stabilized)	
$g_m$	Oscillator transconductance	Startup	3.5	-	-	mA / V
$t_{SU(HSE)}^{(4)}$	Startup time	$V_{DD}$ is stabilized	-	1	-	ms

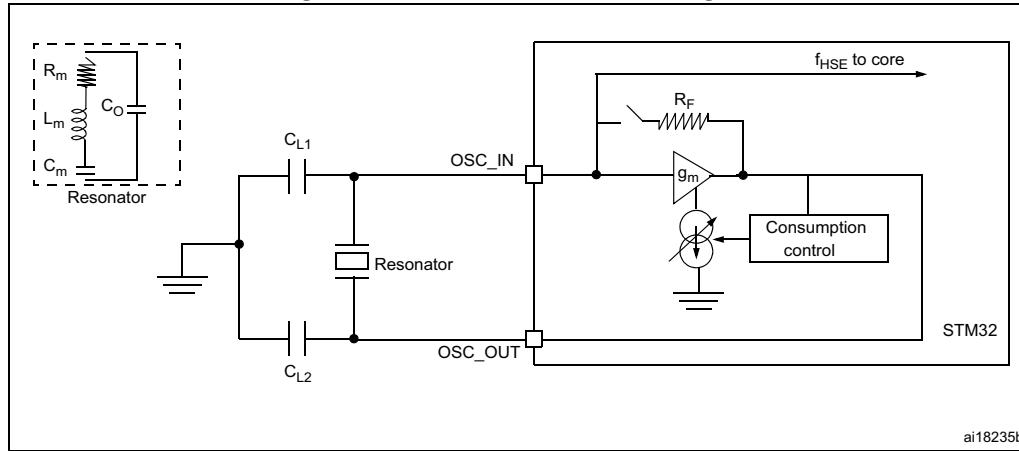
1. Resonator characteristics given by the crystal/ceramic resonator manufacturer.
2. Guaranteed by characterization results.
3. The relatively low value of the RF resistor offers a good protection against issues resulting from use in a humid environment, due to the induced leakage and the bias condition change. However, it is recommended to take this point into account if the MCU is used in tough humidity conditions.
4.  $t_{SU(HSE)}$  is the startup time measured from the moment it is enabled (by software) to a stabilized 8 MHz oscillation is reached. This value is measured for a standard crystal resonator and it can vary significantly with the crystal manufacturer.

For  $C_{L1}$  and  $C_{L2}$ , it is recommended to use high-quality external ceramic capacitors in the 5 pF to 25 pF range (typ.), designed for high-frequency applications, and selected to match the requirements of the crystal or resonator (see [Figure 17](#)).  $C_{L1}$  and  $C_{L2}$  are usually the same size. The crystal manufacturer typically specifies a load capacitance which is the series combination of  $C_{L1}$  and  $C_{L2}$ . PCB and MCU pin capacitance must be included (10 pF can be used as a rough estimate of the combined pin and board capacitance) when sizing  $C_{L1}$  and  $C_{L2}$ . Refer to the application note AN2867 “Oscillator design guide for ST microcontrollers” available from the ST website [www.st.com](http://www.st.com).

## STM32L151xC STM32L152xC

## Electrical characteristics

Figure 17. HSE oscillator circuit diagram



1.  $R_{EXT}$  value depends on the crystal characteristics.

### Low-speed external clock generated from a crystal/ceramic resonator

The low-speed external (LSE) clock can be supplied with a 32.768 kHz crystal/ceramic resonator oscillator. All the information given in this paragraph are based on characterization results obtained with typical external components specified in [Table 30](#). In the application, the resonator and the load capacitors have to be placed as close as possible to the oscillator pins in order to minimize output distortion and startup stabilization time. Refer to the crystal resonator manufacturer for more details on the resonator characteristics (frequency, package, accuracy).

Table 30. LSE oscillator characteristics ( $f_{LSE} = 32.768 \text{ kHz}$ )<sup>(1)</sup>

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$f_{LSE}$	Low speed external oscillator frequency	-	-	32.768	-	kHz
$R_F$	Feedback resistor	-	-	1.2	-	MΩ
$C^{(2)}$	Recommended load capacitance versus equivalent serial resistance of the crystal ( $R_S$ ) <sup>(3)</sup>	$R_S = 30 \text{ k}\Omega$	-	8	-	pF
$I_{LSE}$	LSE driving current	$V_{DD} = 3.3 \text{ V}, V_{IN} = V_{SS}$	-	-	1.1	μA
$I_{DD(LSE)}$	LSE oscillator current consumption	$V_{DD} = 1.8 \text{ V}$	-	450	-	nA
		$V_{DD} = 3.0 \text{ V}$	-	600	-	
		$V_{DD} = 3.6 \text{ V}$	-	750	-	
$g_m$	Oscillator transconductance	-	3	-	-	μA/V
$t_{SU(LSE)}^{(4)}$	Startup time	$V_{DD}$ is stabilized	-	1	-	s

- Guaranteed by characterization results.
- Refer to the note and caution paragraphs below the table, and to the application note AN2867 "Oscillator design guide for ST microcontrollers".
- The oscillator selection can be optimized in terms of supply current using an high quality resonator with small  $R_S$  value for example MSIV-TIN32.768kHz. Refer to crystal manufacturer for more details.

**Electrical characteristics****STM32L151xC STM32L152xC**

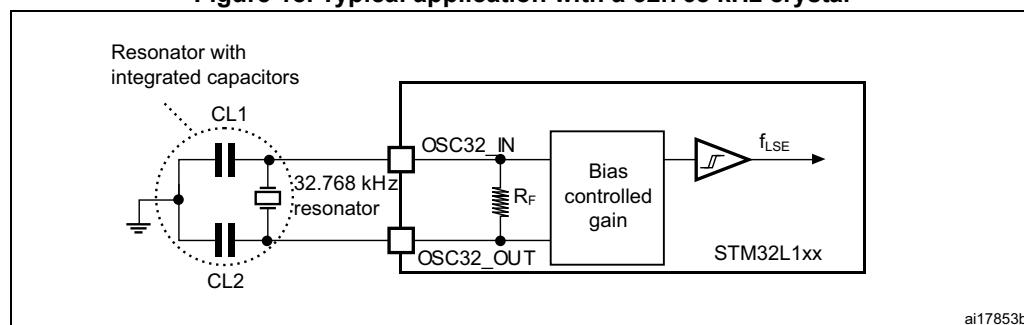
4.  $t_{SU(LSE)}$  is the startup time measured from the moment it is enabled (by software) to a stabilized 32.768 kHz oscillation is reached. This value is measured for a standard crystal resonator and it can vary significantly with the crystal manufacturer.

**Note:** For  $C_{L1}$  and  $C_{L2}$ , it is recommended to use high-quality ceramic capacitors in the 5 pF to 15 pF range selected to match the requirements of the crystal or resonator (see Figure 18).  $C_{L1}$  and  $C_{L2}$ , are usually the same size. The crystal manufacturer typically specifies a load capacitance which is the series combination of  $C_{L1}$  and  $C_{L2}$ . Load capacitance  $C_L$  has the following formula:  $C_L = C_{L1} \times C_{L2} / (C_{L1} + C_{L2}) + C_{stray}$  where  $C_{stray}$  is the pin capacitance and board or trace PCB-related capacitance. Typically, it is between 2 pF and 7 pF.

**Caution:** To avoid exceeding the maximum value of  $C_{L1}$  and  $C_{L2}$  (15 pF) it is strongly recommended to use a resonator with a load capacitance  $C_L \leq 7$  pF. Never use a resonator with a load capacitance of 12.5 pF.

**Example:** if the user chooses a resonator with a load capacitance of  $C_L = 6$  pF and  $C_{stray} = 2$  pF, then  $C_{L1} = C_{L2} = 8$  pF.

**Figure 18. Typical application with a 32.768 kHz crystal**



**STM32L151xC STM32L152xC****Electrical characteristics****6.3.7 Internal clock source characteristics**

The parameters given in [Table 31](#) are derived from tests performed under the conditions summarized in [Table 14](#).

**High-speed internal (HSI) RC oscillator****Table 31. HSI oscillator characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Typ</b>	<b>Max</b>	<b>Unit</b>
$f_{HSI}$	Frequency	$V_{DD} = 3.0 \text{ V}$	-	16	-	MHz
TRIM <sup>(1)(2)</sup>	HSI user-trimmed resolution	Trimming code is not a multiple of 16	-	$\pm 0.4$	0.7	%
		Trimming code is a multiple of 16	-	-	$\pm 1.5$	%
ACC <sub>HSI</sub> <sup>(2)</sup>	Accuracy of the factory-calibrated HSI oscillator	$V_{DDA} = 3.0 \text{ V}, T_A = 25 \text{ }^\circ\text{C}$	-1 <sup>(3)</sup>	-	1 <sup>(3)</sup>	%
		$V_{DDA} = 3.0 \text{ V}, T_A = 0 \text{ to } 55 \text{ }^\circ\text{C}$	-1.5	-	1.5	%
		$V_{DDA} = 3.0 \text{ V}, T_A = -10 \text{ to } 70 \text{ }^\circ\text{C}$	-2	-	2	%
		$V_{DDA} = 3.0 \text{ V}, T_A = -10 \text{ to } 85 \text{ }^\circ\text{C}$	-2.5	-	2	%
		$V_{DDA} = 3.0 \text{ V}, T_A = -10 \text{ to } 105 \text{ }^\circ\text{C}$	-4	-	2	%
		$V_{DDA} = 1.65 \text{ V to } 3.6 \text{ V}$ $T_A = -40 \text{ to } 105 \text{ }^\circ\text{C}$	-4	-	3	%
$t_{SU(HSI)}$ <sup>(2)</sup>	HSI oscillator startup time	-	-	3.7	6	$\mu\text{s}$
$I_{DD(HSI)}$ <sup>(2)</sup>	HSI oscillator power consumption	-	-	100	140	$\mu\text{A}$

1. The trimming step differs depending on the trimming code. It is usually negative on the codes which are multiples of 16 (0x00, 0x10, 0x20, 0x30...0xE0).

2. Guaranteed by characterization results.

3. Guaranteed by test in production.

**Low-speed internal (LSI) RC oscillator****Table 32. LSI oscillator characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Min</b>	<b>Typ</b>	<b>Max</b>	<b>Unit</b>
$f_{LSI}$ <sup>(1)</sup>	LSI frequency	26	38	56	KHz
$D_{LSI}$ <sup>(2)</sup>	LSI oscillator frequency drift $0^\circ\text{C} \leq T_A \leq 105^\circ\text{C}$	-10	-	4	%
$t_{su(LSI)}$ <sup>(3)</sup>	LSI oscillator startup time	-	-	200	$\mu\text{s}$
$I_{DD(LSI)}$ <sup>(3)</sup>	LSI oscillator power consumption	-	400	510	nA

1. Guaranteed by test in production.

2. This is a deviation for an individual part, once the initial frequency has been measured.

3. Guaranteed by design.

**Electrical characteristics****STM32L151xC STM32L152xC****Multi-speed internal (MSI) RC oscillator****Table 33. MSI oscillator characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Condition</b>	<b>Typ</b>	<b>Max</b>	<b>Unit</b>
$f_{\text{MSI}}$	Frequency after factory calibration, done at $V_{\text{DD}} = 3.3 \text{ V}$ and $T_A = 25^\circ\text{C}$	MSI range 0	65.5	-	kHz
		MSI range 1	131	-	
		MSI range 2	262	-	
		MSI range 3	524	-	
		MSI range 4	1.05	-	MHz
		MSI range 5	2.1	-	
		MSI range 6	4.2	-	
$\text{ACC}_{\text{MSI}}$	Frequency error after factory calibration	-	$\pm 0.5$	-	%
$D_{\text{TEMP(}MSI\text{)}}^{(1)}$	MSI oscillator frequency drift $0^\circ\text{C} \leq T_A \leq 105^\circ\text{C}$	-	$\pm 3$	-	%
$D_{\text{VOLT(}MSI\text{)}}^{(1)}$	MSI oscillator frequency drift $1.65 \text{ V} \leq V_{\text{DD}} \leq 3.6 \text{ V}, T_A = 25^\circ\text{C}$	-	-	2.5	%/V
$I_{\text{DD(}MSI\text{)}}^{(2)}$	MSI oscillator power consumption	MSI range 0	0.75	-	$\mu\text{A}$
		MSI range 1	1	-	
		MSI range 2	1.5	-	
		MSI range 3	2.5	-	
		MSI range 4	4.5	-	
		MSI range 5	8	-	
		MSI range 6	15	-	
$t_{\text{SU(}MSI\text{)}}$	MSI oscillator startup time	MSI range 0	30	-	$\mu\text{s}$
		MSI range 1	20	-	
		MSI range 2	15	-	
		MSI range 3	10	-	
		MSI range 4	6	-	
		MSI range 5	5	-	
		MSI range 6, Voltage range 1 and 2	3.5	-	
		MSI range 6, Voltage range 3	5	-	

**STM32L151xC STM32L152xC****Electrical characteristics****Table 33. MSI oscillator characteristics (continued)**

Symbol	Parameter	Condition	Typ	Max	Unit
$t_{STAB(MSI)}^{(2)}$	MSI oscillator stabilization time	MSI range 0	-	40	$\mu s$
		MSI range 1	-	20	
		MSI range 2	-	10	
		MSI range 3	-	4	
		MSI range 4	-	2.5	
		MSI range 5	-	2	
		MSI range 6, Voltage range 1 and 2	-	2	
		MSI range 3, Voltage range 3	-	3	
$f_{OVER(MSI)}$	MSI oscillator frequency overshoot	Any range to range 5	-	4	MHz
		Any range to range 6	-	6	

1. This is a deviation for an individual part, once the initial frequency has been measured.

2. Guaranteed by characterization results.

**Electrical characteristics****STM32L151xC STM32L152xC****6.3.8 PLL characteristics**

The parameters given in [Table 34](#) are derived from tests performed under the conditions summarized in [Table 14](#).

**Table 34. PLL characteristics**

Symbol	Parameter	Value			Unit
		Min	Typ	Max <sup>(1)</sup>	
$f_{PLL\_IN}$	PLL input clock <sup>(2)</sup>	2	-	24	MHz
	PLL input clock duty cycle	45	-	55	%
$f_{PLL\_OUT}$	PLL output clock	2	-	32	MHz
$t_{LOCK}$	PLL lock time PLL input = 16 MHz PLL VCO = 96 MHz	-	115	160	μs
Jitter	Cycle-to-cycle jitter	-	-	±600	ps
$I_{DDA}(PLL)$	Current consumption on $V_{DDA}$	-	220	450	μA
$I_{DD}(PLL)$	Current consumption on $V_{DD}$	-	120	150	

1. Guaranteed by characterization results.

2. Take care of using the appropriate multiplier factors so as to have PLL input clock values compatible with the range defined by  $f_{PLL\_OUT}$ .

**6.3.9 Memory characteristics**

The characteristics are given at  $T_A = -40$  to  $105$  °C unless otherwise specified.

**RAM memory****Table 35. RAM and hardware registers**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
VRM	Data retention mode <sup>(1)</sup>	STOP mode (or RESET)	1.65	-	-	V

1. Minimum supply voltage without losing data stored in RAM (in Stop mode or under Reset) or in hardware registers (only in Stop mode).

**STM32L151xC STM32L152xC****Electrical characteristics****Flash memory and data EEPROM****Table 36. Flash memory and data EEPROM characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Typ</b>	<b>Max<sup>(1)</sup></b>	<b>Unit</b>
$V_{DD}$	Operating voltage Read / Write / Erase	-	1.65	-	3.6	V
$t_{prog}$	Programming/ erasing time for byte / word / double word / half-page	Erasing	-	3.28	3.94	ms
		Programming	-	3.28	3.94	
$I_{DD}$	Average current during the whole programming / erase operation	$T_A = 25^\circ\text{C}, V_{DD} = 3.6 \text{ V}$	-	600	900	$\mu\text{A}$
	Maximum current (peak) during the whole programming / erase operation		-	1.5	2.5	mA

1. Guaranteed by design.

**Table 37. Flash memory and data EEPROM endurance and retention**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Value</b>			<b>Unit</b>
			<b>Min<sup>(1)</sup></b>	<b>Typ</b>	<b>Max</b>	
$N_{CYC}^{(2)}$	Cycling (erase / write) Program memory	$T_A = -40^\circ\text{C} \text{ to } 105^\circ\text{C}$	10	-	-	kcycles
	Cycling (erase / write) EEPROM data memory		300	-	-	
$t_{RET}^{(2)}$	Data retention (program memory) after 10 kcycles at $T_A = 85^\circ\text{C}$	$T_{RET} = +85^\circ\text{C}$	30	-	-	years
	Data retention (EEPROM data memory) after 300 kcycles at $T_A = 85^\circ\text{C}$		30	-	-	
	Data retention (program memory) after 10 kcycles at $T_A = 105^\circ\text{C}$	$T_{RET} = +105^\circ\text{C}$	10	-	-	
	Data retention (EEPROM data memory) after 300 kcycles at $T_A = 105^\circ\text{C}$		10	-	-	

1. Guaranteed by characterization results.

2. Characterization is done according to JEDEC JESD22-A117.

**Electrical characteristics****STM32L151xC STM32L152xC****6.3.10 EMC characteristics**

Susceptibility tests are performed on a sample basis during device characterization.

**Functional EMS (electromagnetic susceptibility)**

While a simple application is executed on the device (toggling 2 LEDs through I/O ports), the device is stressed by two electromagnetic events until a failure occurs. The failure is indicated by the LEDs:

- **Electrostatic discharge (ESD)** (positive and negative) is applied to all device pins until a functional disturbance occurs. This test is compliant with the IEC 61000-4-2 standard.
- **FTB:** A Burst of Fast Transient voltage (positive and negative) is applied to  $V_{DD}$  and  $V_{SS}$  through a 100 pF capacitor, until a functional disturbance occurs. This test is compliant with the IEC 61000-4-4 standard.

A device reset allows normal operations to be resumed.

The test results are given in *Table 38*. They are based on the EMS levels and classes defined in application note AN1709.

**Table 38. EMS characteristics**

Symbol	Parameter	Conditions	Level/ Class
$V_{FESD}$	Voltage limits to be applied on any I/O pin to induce a functional disturbance	$V_{DD} = 3.3 \text{ V}$ , LQFP100, $T_A = +25^\circ\text{C}$ , $f_{HCLK} = 32 \text{ MHz}$ conforms to IEC 61000-4-2	2B
$V_{EFTB}$	Fast transient voltage burst limits to be applied through 100 pF on $V_{DD}$ and $V_{SS}$ pins to induce a functional disturbance	$V_{DD} = 3.3 \text{ V}$ , LQFP100, $T_A = +25^\circ\text{C}$ , $f_{HCLK} = 32 \text{ MHz}$ conforms to IEC 61000-4-4	4A

**Designing hardened software to avoid noise problems**

EMC characterization and optimization are performed at component level with a typical application environment and simplified MCU software. It should be noted that good EMC performance is highly dependent on the user application and the software in particular.

Therefore it is recommended that the user applies EMC software optimization and prequalification tests in relation with the EMC level requested for his application.

**Software recommendations**

The software flowchart must include the management of runaway conditions such as:

- Corrupted program counter
- Unexpected reset
- Critical data corruption (control registers...)

**Prequalification trials**

Most of the common failures (unexpected reset and program counter corruption) can be reproduced by manually forcing a low state on the NRST pin or the oscillator pins for 1 second.

**STM32L151xC STM32L152xC****Electrical characteristics**

To complete these trials, ESD stress can be applied directly on the device, over the range of specification values. When unexpected behavior is detected, the software can be hardened to prevent unrecoverable errors occurring (see application note AN1015).

**Electromagnetic Interference (EMI)**

The electromagnetic field emitted by the device are monitored while a simple application is executed (toggling 2 LEDs through the I/O ports). This emission test is compliant with IEC 61967-2 standard which specifies the test board and the pin loading.

**Table 39. EMI characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Monitored frequency band</b>	<b>Max vs. frequency range</b>			<b>Unit</b>
				<b>4 MHz voltage range 3</b>	<b>16 MHz voltage range 2</b>	<b>32 MHz voltage range 1</b>	
$S_{\text{EMI}}$	Peak level	$V_{\text{DD}} = 3.3 \text{ V}$ , $T_A = 25^\circ\text{C}$ , LQFP100 package compliant with IEC 61967-2	0.1 to 30 MHz	3	-6	-5	dB $\mu$ V
			30 to 130 MHz	18	4	-7	
			130 MHz to 1GHz	15	5	-7	
			SAE EMI Level	2.5	2	1	

**6.3.11 Electrical sensitivity characteristics**

Based on three different tests (ESD, LU) using specific measurement methods, the device is stressed in order to determine its performance in terms of electrical sensitivity.

**Electrostatic discharge (ESD)**

Electrostatic discharges (a positive then a negative pulse separated by 1 second) are applied to the pins of each sample according to each pin combination. The sample size depends on the number of supply pins in the device (3 parts  $\times$  (n+1) supply pins). This test conforms to the JESD22-A114, ANSI/ESD STM5.3.1. standard.

**Table 40. ESD absolute maximum ratings**

<b>Symbol</b>	<b>Ratings</b>	<b>Conditions</b>	<b>Class</b>	<b>Maximum value<sup>(1)</sup></b>	<b>Unit</b>
$V_{\text{ESD(HBM)}}$	Electrostatic discharge voltage (human body model)	$T_A = +25^\circ\text{C}$ , conforming to JESD22-A114	2	2000	V
$V_{\text{ESD(CDM)}}$	Electrostatic discharge voltage (charge device model)	$T_A = +25^\circ\text{C}$ , conforming to ANSI/ESD STM5.3.1.	C4	500	V

1. Guaranteed by characterization results.

**Electrical characteristics****STM32L151xC STM32L152xC****Static latch-up**

Two complementary static tests are required on six parts to assess the latch-up performance:

- A supply overvoltage is applied to each power supply pin
- A current injection is applied to each input, output and configurable I/O pin

These tests are compliant with EIA/JESD 78A IC latch-up standard.

**Table 41. Electrical sensitivities**

Symbol	Parameter	Conditions	Class
LU	Static latch-up class	T <sub>A</sub> = +105 °C conforming to JESD78A	II level A

**6.3.12 I/O current injection characteristics**

As a general rule, current injection to the I/O pins, due to external voltage below V<sub>SS</sub> or above V<sub>DD</sub> (for standard pins) should be avoided during normal product operation.

However, in order to give an indication of the robustness of the microcontroller in cases when abnormal injection accidentally happens, susceptibility tests are performed on a sample basis during device characterization.

**Functional susceptibility to I/O current injection**

While a simple application is executed on the device, the device is stressed by injecting current into the I/O pins programmed in floating input mode. While current is injected into the I/O pin, one at a time, the device is checked for functional failures.

The failure is indicated by an out of range parameter: ADC error above a certain limit (higher than 5 LSB TUE), out of conventional limits of induced leakage current on adjacent pins (out of -5 µA/+0 µA range), or other functional failure (for example reset occurrence oscillator frequency deviation, LCD levels).

The test results are given in the [Table 42](#).

**Table 42. I/O current injection susceptibility**

Symbol	Description	Functional susceptibility		Unit
		Negative injection	Positive injection	
I <sub>INJ</sub>	Injected current on all 5 V tolerant (FT) pins	-5 <sup>(1)</sup>	NA <sup>(2)</sup>	mA
	Injected current on BOOT0	-0	NA <sup>(2)</sup>	
	Injected current on any other pin	-5 <sup>(1)</sup>	+5	

1. It is recommended to add a Schottky diode (pin to ground) to analog pins which may potentially inject negative currents.

2. Injection is not possible.

**STM32L151xC STM32L152xC****Electrical characteristics****6.3.13 I/O port characteristics****General input/output characteristics**

Unless otherwise specified, the parameters given in [Table 49](#) are derived from tests performed under the conditions summarized in [Table 14](#). All I/Os are CMOS and TTL compliant.

**Table 43. I/O static characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Typ</b>	<b>Max</b>	<b>Unit</b>
$V_{IL}$	Input low level voltage	TC and FT I/O	-	-	$0.3 V_{DD}^{(1)(2)}$	V
		BOOT0	-	-	$0.14 V_{DD}^{(2)}$	
$V_{IH}$	Input high level voltage	TC I/O	$0.45 V_{DD} + 0.38^{(2)}$	-	-	V
		FT I/O	$0.39 V_{DD} + 0.59^{(2)}$	-	-	
		BOOT0	$0.15 V_{DD} + 0.56^{(2)}$	-	-	
$V_{hys}$	I/O Schmitt trigger voltage hysteresis <sup>(2)</sup>	TC and FT I/O	-	$10\% V_{DD}^{(3)}$	-	nA
		BOOT0	-	0.01	-	
$I_{lkg}$	Input leakage current <sup>(4)</sup>	$V_{SS} \leq V_{IN} \leq V_{DD}$ I/Os with LCD	-	-	$\pm 50$	nA
		$V_{SS} \leq V_{IN} \leq V_{DD}$ I/Os with analog switches	-	-	$\pm 50$	
		$V_{SS} \leq V_{IN} \leq V_{DD}$ I/Os with analog switches and LCD	-	-	$\pm 50$	
		$V_{SS} \leq V_{IN} \leq V_{DD}$ I/Os with USB	-	-	$\pm 250$	
		$V_{SS} \leq V_{IN} \leq V_{DD}$ TC and FT I/Os	-	-	$\pm 50$	
		FT I/O $V_{DD} \leq V_{IN} \leq 5V$	-	-	$\pm 10$	$\mu A$
$R_{PU}$	Weak pull-up equivalent resistor <sup>(5)(1)</sup>	$V_{IN} = V_{SS}$	25	45	65	k $\Omega$
$R_{PD}$	Weak pull-down equivalent resistor <sup>(5)</sup>	$V_{IN} = V_{DD}$	25	45	65	k $\Omega$
$C_{IO}$	I/O pin capacitance	-	-	5	-	pF

1. Guaranteed by test in production.
2. Guaranteed by design.
3. With a minimum of 200 mV.
4. The max. value may be exceeded if negative current is injected on adjacent pins.
5. Pull-up and pull-down resistors are designed with a true resistance in series with a switchable PMOS/NMOS. This MOS/NMOS contribution to the series resistance is minimum (~10% order).

**Electrical characteristics****STM32L151xC STM32L152xC****Output driving current**

The GPIOs (general purpose input/outputs) can sink or source up to  $\pm 8$  mA, and sink or source up to  $\pm 20$  mA with the non-standard  $V_{OL}/V_{OH}$  specifications given in [Table 44](#).

In the user application, the number of I/O pins which can drive current must be limited to respect the absolute maximum rating specified in [Section 6.2](#):

- The sum of the currents sourced by all the I/Os on  $V_{DD}$ , plus the maximum Run consumption of the MCU sourced on  $V_{DD}$ , cannot exceed the absolute maximum rating  $I_{VDD(\Sigma)}$  (see [Table 12](#)).
- The sum of the currents sunk by all the I/Os on  $V_{SS}$  plus the maximum Run consumption of the MCU sunk on  $V_{SS}$  cannot exceed the absolute maximum rating  $I_{VSS(\Sigma)}$  (see [Table 12](#)).

**Output voltage levels**

Unless otherwise specified, the parameters given in [Table 44](#) are derived from tests performed under the conditions summarized in [Table 14](#). All I/Os are CMOS and TTL compliant.

**Table 44. Output voltage characteristics**

Symbol	Parameter	Conditions	Min	Max	Unit
$V_{OL}^{(1)(2)}$	Output low level voltage for an I/O pin	$I_{IO} = 8$ mA $2.7 \text{ V} < V_{DD} < 3.6 \text{ V}$	-	0.4	V
$V_{OH}^{(2)(3)}$	Output high level voltage for an I/O pin		$V_{DD}-0.4$	-	
$V_{OL}^{(3)(4)}$	Output low level voltage for an I/O pin	$I_{IO} = 4$ mA $1.65 \text{ V} < V_{DD} < 3.6 \text{ V}$	-	0.45	V
$V_{OH}^{(3)(4)}$	Output high level voltage for an I/O pin		$V_{DD}-0.45$	-	
$V_{OL}^{(1)(4)}$	Output low level voltage for an I/O pin	$I_{IO} = 20$ mA $2.7 \text{ V} < V_{DD} < 3.6 \text{ V}$	-	1.3	V
$V_{OH}^{(3)(4)}$	Output high level voltage for an I/O pin		$V_{DD}-1.3$	-	

1. The  $I_{IO}$  current sunk by the device must always respect the absolute maximum rating specified in [Table 12](#) and the sum of  $I_{IO}$  (I/O ports and control pins) must not exceed  $I_{VSS}$ .
2. Guaranteed by test in production.
3. The  $I_{IO}$  current sourced by the device must always respect the absolute maximum rating specified in [Table 12](#) and the sum of  $I_{IO}$  (I/O ports and control pins) must not exceed  $I_{VDD}$ .
4. Guaranteed by characterization results.

**STM32L151xC STM32L152xC****Electrical characteristics****Input/output AC characteristics**

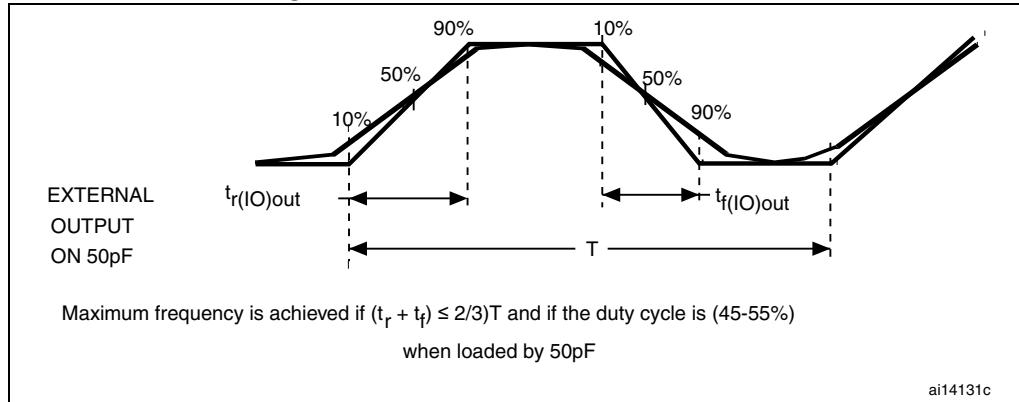
The definition and values of input/output AC characteristics are given in [Figure 19](#) and [Table 45](#), respectively.

Unless otherwise specified, the parameters given in [Table 45](#) are derived from tests performed under the conditions summarized in [Table 14](#).

**Table 45. I/O AC characteristics<sup>(1)</sup>**

<b>OSPEEDRx [1:0] bit value<sup>(1)</sup></b>	<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Max<sup>(2)</sup></b>	<b>Unit</b>
00	$f_{max(IO)out}$	Maximum frequency <sup>(3)</sup>	$C_L = 50 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	400	kHz
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	400	
	$t_f(IO)out$ $t_r(IO)out$	Output rise and fall time	$C_L = 50 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	625	ns
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	625	
01	$f_{max(IO)out}$	Maximum frequency <sup>(3)</sup>	$C_L = 50 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	2	MHz
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	1	
	$t_f(IO)out$ $t_r(IO)out$	Output rise and fall time	$C_L = 50 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	125	ns
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	250	
10	$F_{max(IO)out}$	Maximum frequency <sup>(3)</sup>	$C_L = 50 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	10	MHz
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	2	
	$t_f(IO)out$ $t_r(IO)out$	Output rise and fall time	$C_L = 50 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	25	ns
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	125	
11	$F_{max(IO)out}$	Maximum frequency <sup>(3)</sup>	$C_L = 30 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	50	MHz
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	8	
	$t_f(IO)out$ $t_r(IO)out$	Output rise and fall time	$C_L = 30 \text{ pF}, V_{DD} = 2.7 \text{ V to } 3.6 \text{ V}$	-	5	ns
			$C_L = 50 \text{ pF}, V_{DD} = 1.65 \text{ V to } 2.7 \text{ V}$	-	30	
-	$t_{EXTIpw}$	Pulse width of external signals detected by the EXTI controller	-	8	-	

1. The I/O speed is configured using the OSPEEDRx[1:0] bits. Refer to the STM32L151xx, STM32L152xx and STM32L162xx reference manual for a description of GPIO Port configuration register.
2. Guaranteed by design.
3. The maximum frequency is defined in [Figure 19](#).

**Electrical characteristics****STM32L151xC STM32L152xC****Figure 19. I/O AC characteristics definition****6.3.14 NRST pin characteristics**

The NRST pin input driver uses CMOS technology. It is connected to a permanent pull-up resistor,  $R_{PU}$  (see [Table 46](#))

Unless otherwise specified, the parameters given in [Table 46](#) are derived from tests performed under the conditions summarized in [Table 14](#).

**Table 46. NRST pin characteristics**

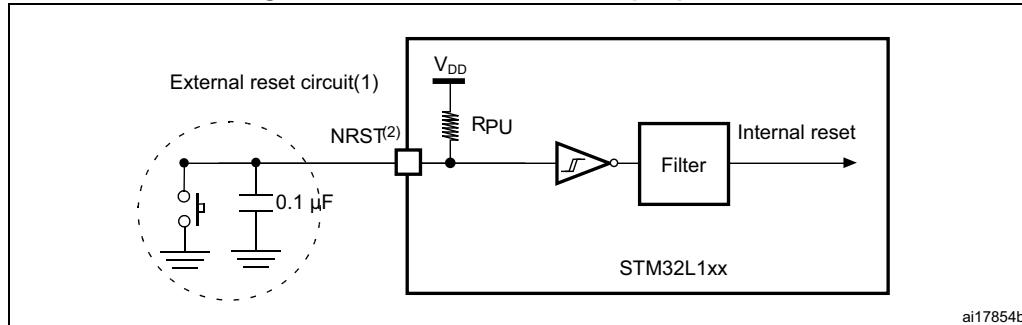
Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{IL(\text{NRST})}^{(1)}$	NRST input low level voltage	-	-	-	0.3 $V_{DD}$	
$V_{IH(\text{NRST})}^{(1)}$	NRST input high level voltage	-	$0.39V_{DD}+0.59$	-	-	
$V_{OL(\text{NRST})}^{(1)}$	NRST output low level voltage	$I_{OL} = 2 \text{ mA}$ $2.7 \text{ V} < V_{DD} < 3.6 \text{ V}$	-	-	0.4	V
		$I_{OL} = 1.5 \text{ mA}$ $1.65 \text{ V} < V_{DD} < 2.7 \text{ V}$	-	-		
$V_{hys(\text{NRST})}^{(1)}$	NRST Schmitt trigger voltage hysteresis	-	-	$10\%V_{DD}^{(2)}$	-	mV
$R_{PU}$	Weak pull-up equivalent resistor <sup>(3)</sup>	$V_{IN} = V_{SS}$	25	45	65	k $\Omega$
$V_{F(\text{NRST})}^{(1)}$	NRST input filtered pulse	-	-	-	50	ns
$V_{NF(\text{NRST})}^{(3)}$	NRST input not filtered pulse	-	350	-	-	ns

1. Guaranteed by design.
2. With a minimum of 200 mV.
3. The pull-up is designed with a true resistance in series with a switchable PMOS. This PMOS contribution to the series resistance is around 10%.

## STM32L151xC STM32L152xC

## Electrical characteristics

Figure 20. Recommended NRST pin protection



ai17854b

1. The reset network protects the device against parasitic resets. 0.1 uF capacitor must be placed as close as possible to the chip.
2. The user must ensure that the level on the NRST pin can go below the V<sub>L(NRST)</sub> max level specified in [Table 46](#). Otherwise the reset will not be taken into account by the device.

## 6.3.15 TIM timer characteristics

The parameters given in the [Table 47](#) are guaranteed by design.

Refer to [Section 6.3.13: I/O port characteristics](#) for details on the input/output characteristics (output compare, input capture, external clock, PWM output).

Table 47. TIMx<sup>(1)</sup> characteristics

Symbol	Parameter	Conditions	Min	Max	Unit
$t_{\text{res}}(\text{TIM})$	Timer resolution time	-	1	-	$t_{\text{TIMxCLK}}$
		$f_{\text{TIMxCLK}} = 32 \text{ MHz}$	31.25	-	ns
$f_{\text{EXT}}$	Timer external clock frequency on CH1 to CH4	-	0	$f_{\text{TIMxCLK}}/2$	MHz
		$f_{\text{TIMxCLK}} = 32 \text{ MHz}$	0	16	MHz
$\text{Res}_{\text{TIM}}$	Timer resolution	-		16	bit
$t_{\text{COUNTER}}$	16-bit counter clock period when internal clock is selected (timer's prescaler disabled)	-	1	65536	$t_{\text{TIMxCLK}}$
		$f_{\text{TIMxCLK}} = 32 \text{ MHz}$	0.0312	2048	μs
$t_{\text{MAX\_COUNT}}$	Maximum possible count	-	-	$65536 \times 65536$	$t_{\text{TIMxCLK}}$
		$f_{\text{TIMxCLK}} = 32 \text{ MHz}$	-	134.2	s

1. TIMx is used as a general term to refer to the TIM2, TIM3 and TIM4 timers.

**Electrical characteristics****STM32L151xC STM32L152xC****6.3.16 Communications interfaces****I<sup>2</sup>C interface characteristics**

The device I<sup>2</sup>C interface meets the requirements of the standard I<sup>2</sup>C communication protocol with the following restrictions: SDA and SCL are not “true” open-drain I/O pins. When configured as open-drain, the PMOS connected between the I/O pin and V<sub>DD</sub> is disabled, but is still present.

The I<sup>2</sup>C characteristics are described in [Table 48](#). Refer also to [Section 6.3.13: I/O port characteristics](#) for more details on the input/output characteristics (SDA and SCL).

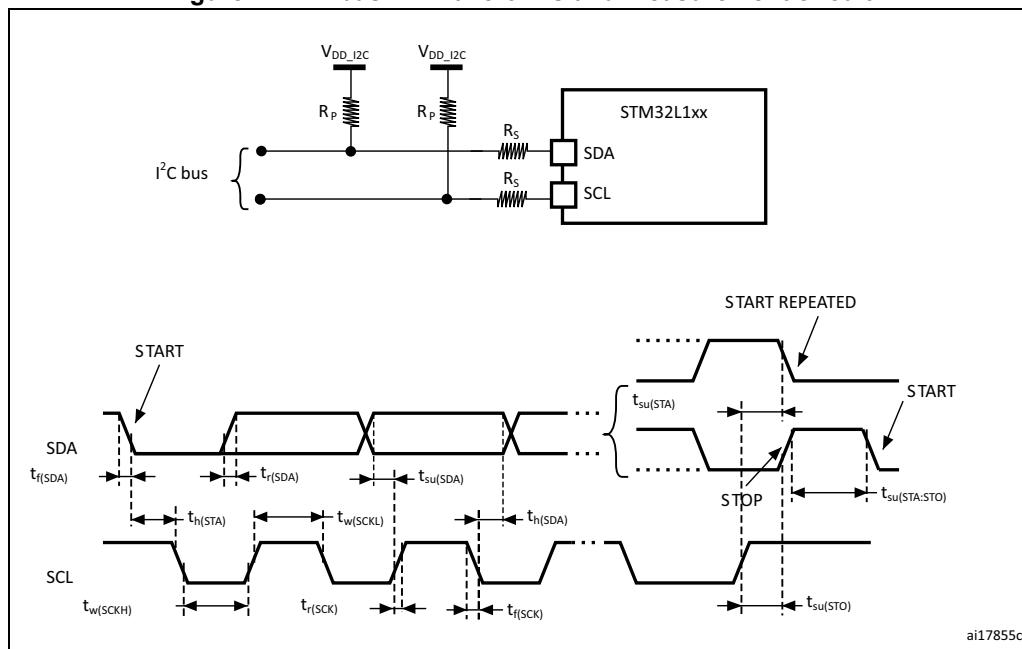
**Table 48. I<sup>2</sup>C characteristics**

<b>Symbol</b>	<b>Parameter</b>	<b>Standard mode I<sup>2</sup>C<sup>(1)(2)</sup></b>		<b>Fast mode I<sup>2</sup>C<sup>(1)(2)</sup></b>		<b>Unit</b>
		<b>Min</b>	<b>Max</b>	<b>Min</b>	<b>Max</b>	
t <sub>w(SCLL)</sub>	SCL clock low time	4.7	-	1.3	-	μs
t <sub>w(SCLH)</sub>	SCL clock high time	4.0	-	0.6	-	
t <sub>su(SDA)</sub>	SDA setup time	250	-	100	-	ns
t <sub>h(SDA)</sub>	SDA data hold time	-	3450 <sup>(3)</sup>	-	900 <sup>(3)</sup>	
t <sub>r(SDA)</sub> t <sub>r(SCL)</sub>	SDA and SCL rise time	-	1000	-	300	ns
t <sub>f(SDA)</sub> t <sub>f(SCL)</sub>	SDA and SCL fall time	-	300	-	300	
t <sub>h(STA)</sub>	Start condition hold time	4.0	-	0.6	-	μs
t <sub>su(STA)</sub>	Repeated Start condition setup time	4.7	-	0.6	-	
t <sub>su(STO)</sub>	Stop condition setup time	4.0	-	0.6	-	μs
t <sub>w(STO:STA)</sub>	Stop to Start condition time (bus free)	4.7	-	1.3	-	μs
C <sub>b</sub>	Capacitive load for each bus line	-	400	-	400	pF
t <sub>SP</sub>	Pulse width of spikes that are suppressed by the analog filter	0	50 <sup>(4)</sup>	0	50 <sup>(4)</sup>	ns

1. Guaranteed by design.
2. f<sub>PCLK1</sub> must be at least 2 MHz to achieve standard mode I<sup>2</sup>C frequencies. It must be at least 4 MHz to achieve fast mode I<sup>2</sup>C frequencies. It must be a multiple of 10 MHz to reach the 400 kHz maximum I<sup>2</sup>C fast mode clock.
3. The maximum Data hold time has only to be met if the interface does not stretch the low period of SCL signal.
4. The minimum width of the spikes filtered by the analog filter is above t<sub>SP(max)</sub>.

## STM32L151xC STM32L152xC

## Electrical characteristics

Figure 21. I<sup>2</sup>C bus AC waveforms and measurement circuit

1.  $R_S$  = series protection resistor.
2.  $R_P$  = external pull-up resistor.
3.  $V_{DD\_I2C}$  is the I<sup>2</sup>C bus power supply.
4. Measurement points are done at CMOS levels:  $0.3V_{DD}$  and  $0.7V_{DD}$ .

Table 49. SCL frequency ( $f_{PCLK1} = 32$  MHz,  $V_{DD} = V_{DD\_I2C} = 3.3$  V)<sup>(1)(2)</sup>

$f_{SCL}$ (kHz)	I <sup>2</sup> C_CCR value
	$R_P = 4.7$ kΩ
400	0x801B
300	0x8024
200	0x8035
100	0x00A0
50	0x0140
20	0x0320

1.  $R_P$  = External pull-up resistance,  $f_{SCL}$  = I<sup>2</sup>C speed.
2. For speeds around 200 kHz, the tolerance on the achieved speed is of  $\pm 5\%$ . For other speed ranges, the tolerance on the achieved speed is  $\pm 2\%$ . These variations depend on the accuracy of the external components used to design the application.

**Electrical characteristics****STM32L151xC STM32L152xC****SPI characteristics**

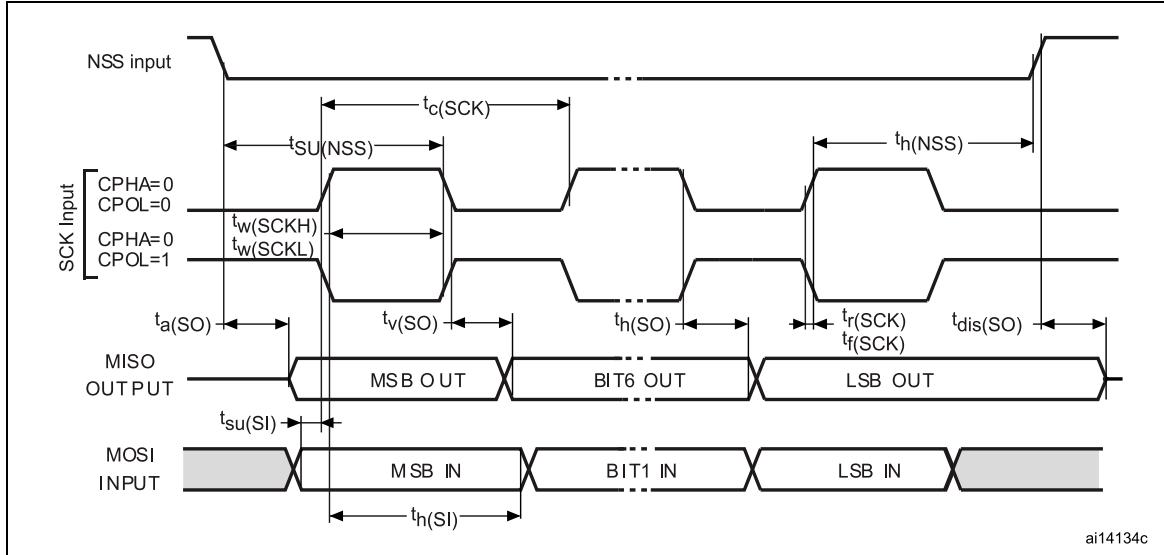
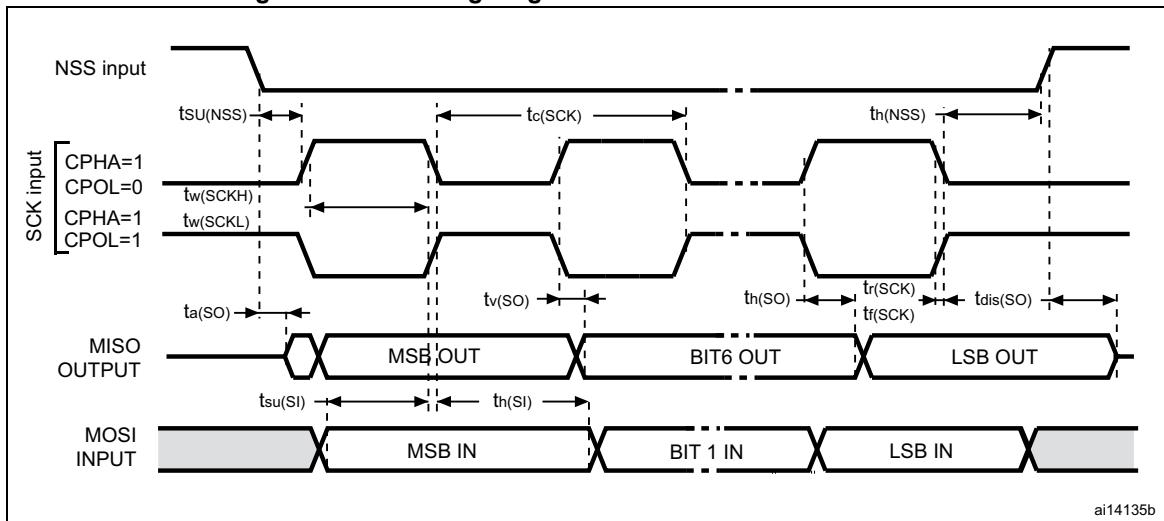
Unless otherwise specified, the parameters given in the following table are derived from tests performed under the conditions summarized in [Table 14](#).

Refer to [Section 6.3.12: I/O current injection characteristics](#) for more details on the input/output alternate function characteristics (NSS, SCK, MOSI, MISO).

**Table 50. SPI characteristics<sup>(1)</sup>**

<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Max<sup>(2)</sup></b>	<b>Unit</b>
$f_{SCK}$ $1/t_c(SCK)$	SPI clock frequency	Master mode	-	16	MHz
		Slave mode	-	16	
		Slave transmitter	-	12 <sup>(3)</sup>	
$t_{r(SCK)}^{(2)}$ $t_{f(SCK)}^{(2)}$	SPI clock rise and fall time	Capacitive load: C = 30 pF	-	6	ns
DuCy(SCK)	SPI slave input clock duty cycle	Slave mode	30	70	%
$t_{su(NSS)}$	NSS setup time	Slave mode	$4t_{HCLK}$	-	ns
$t_h(NSS)$	NSS hold time	Slave mode	$2t_{HCLK}$	-	
$t_w(SCKH)^{(2)}$ $t_w(SCKL)^{(2)}$	SCK high and low time	Master mode	$t_{SCK}/2-5$	$t_{SCK}/2+3$	
$t_{su(MI)}^{(2)}$	Data input setup time	Master mode	5	-	
$t_{su(SI)}^{(2)}$		Slave mode	6	-	
$t_{h(MI)}^{(2)}$	Data input hold time	Master mode	5	-	
$t_{h(SI)}^{(2)}$		Slave mode	5	-	
$t_a(SO)^{(4)}$	Data output access time	Slave mode	0	$3t_{HCLK}$	
$t_v(SO)^{(2)}$	Data output valid time	Slave mode	-	33	
$t_v(MO)^{(2)}$	Data output valid time	Master mode	-	6.5	
$t_h(SO)^{(2)}$	Data output hold time	Slave mode	17	-	
$t_h(MO)^{(2)}$		Master mode	0.5	-	

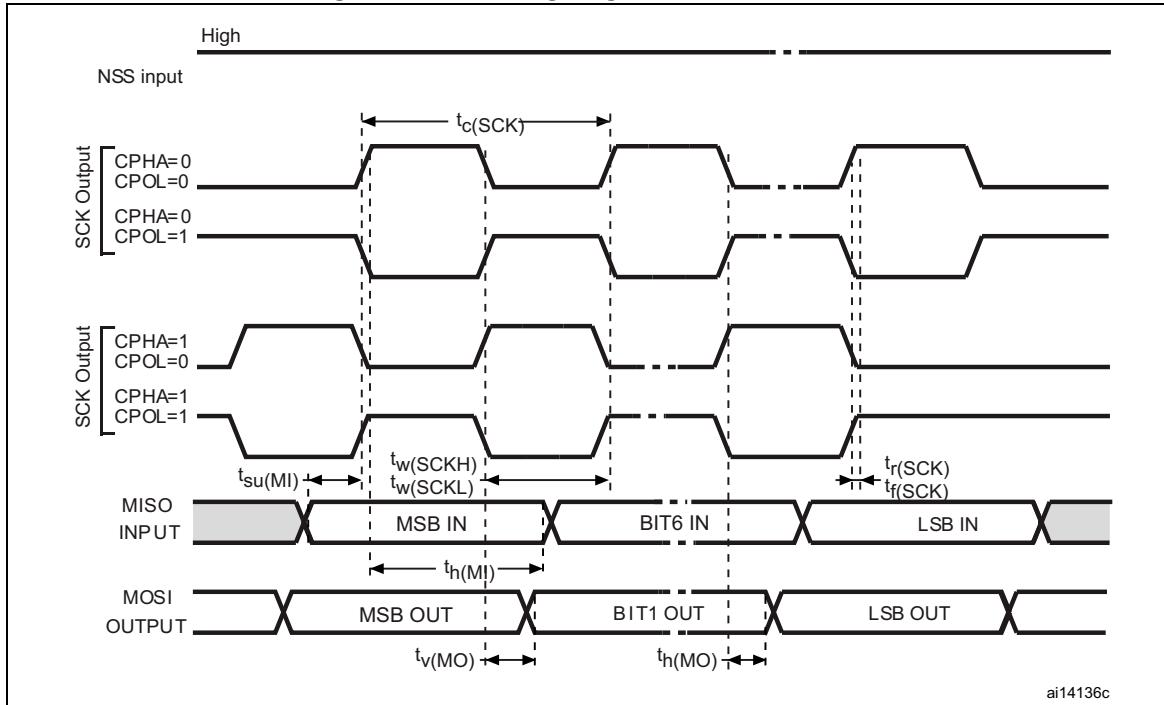
1. The characteristics above are given for voltage range 1.
2. Guaranteed by characterization results.
3. The maximum SPI clock frequency in slave transmitter mode is given for an SPI slave input clock duty cycle (DuCy(SCK)) ranging between 40 to 60%.
4. Min time is for the minimum time to drive the output and max time is for the maximum time to validate the data.

**STM32L151xC STM32L152xC****Electrical characteristics****Figure 22. SPI timing diagram - slave mode and CPHA = 0****Figure 23. SPI timing diagram - slave mode and CPHA = 1<sup>(1)</sup>**

1. Measurement points are done at CMOS levels:  $0.3V_{DD}$  and  $0.7V_{DD}$ .

## Electrical characteristics

## STM32L151xC STM32L152xC

Figure 24. SPI timing diagram - master mode<sup>(1)</sup>

1. Measurement points are done at CMOS levels:  $0.3V_{DD}$  and  $0.7V_{DD}$ .

**STM32L151xC STM32L152xC****Electrical characteristics****USB characteristics**

The USB interface is USB-IF certified (full speed).

**Table 51. USB startup time**

Symbol	Parameter	Max	Unit
$t_{STARTUP}^{(1)}$	USB transceiver startup time	1	$\mu s$

1. Guaranteed by design.

**Table 52. USB DC electrical characteristics**

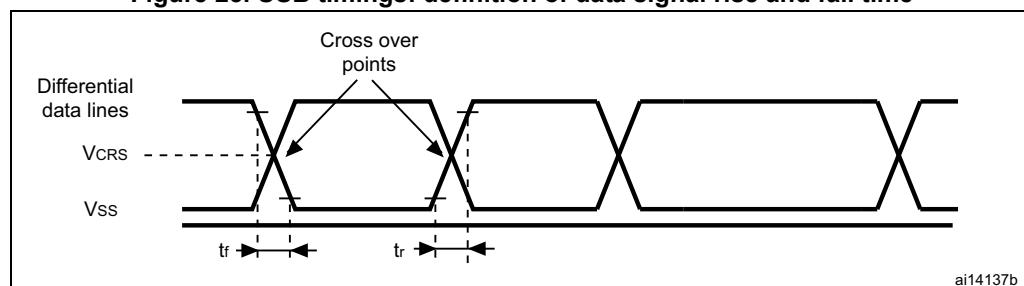
Symbol	Parameter	Conditions	Min. <sup>(1)</sup>	Max. <sup>(1)</sup>	Unit
<b>Input levels</b>					
$V_{DD}$	USB operating voltage	-	3.0	3.6	V
$V_{DI}^{(2)}$	Differential input sensitivity	$I(USB\_DP, USB\_DM)$	0.2	-	V
$V_{CM}^{(2)}$	Differential common mode range	Includes $V_{DI}$ range	0.8	2.5	
$V_{SE}^{(2)}$	Single ended receiver threshold	-	1.3	2.0	
<b>Output levels</b>					
$V_{OL}^{(3)}$	Static output level low	$R_L$ of 1.5 k $\Omega$ to 3.6 V <sup>(4)</sup>	-	0.3	V
$V_{OH}^{(3)}$	Static output level high	$R_L$ of 15 k $\Omega$ to $V_{SS}^{(4)}$	2.8	3.6	

1. All the voltages are measured from the local ground potential.

2. Guaranteed by characterization results.

3. Guaranteed by test in production.

4.  $R_L$  is the load connected on the USB drivers.

**Figure 25. USB timings: definition of data signal rise and fall time**

ai14137b

**Table 53. USB: full speed electrical characteristics**

Driver characteristics <sup>(1)</sup>					
Symbol	Parameter	Conditions	Min	Max	Unit
$t_r$	Rise time <sup>(2)</sup>	$C_L = 50 \text{ pF}$	4	20	ns
$t_f$	Fall Time <sup>(2)</sup>	$C_L = 50 \text{ pF}$	4	20	ns
$t_{rfm}$	Rise/ fall time matching	$t_r/t_f$	90	110	%
$V_{CRS}$	Output signal crossover voltage		1.3	2.0	V

**Electrical characteristics****STM32L151xC STM32L152xC**

1. Guaranteed by design.
2. Measured from 10% to 90% of the data signal. For more detailed informations, please refer to USB Specification - Chapter 7 (version 2.0).

**I2S characteristics****Table 54. I2S characteristics**

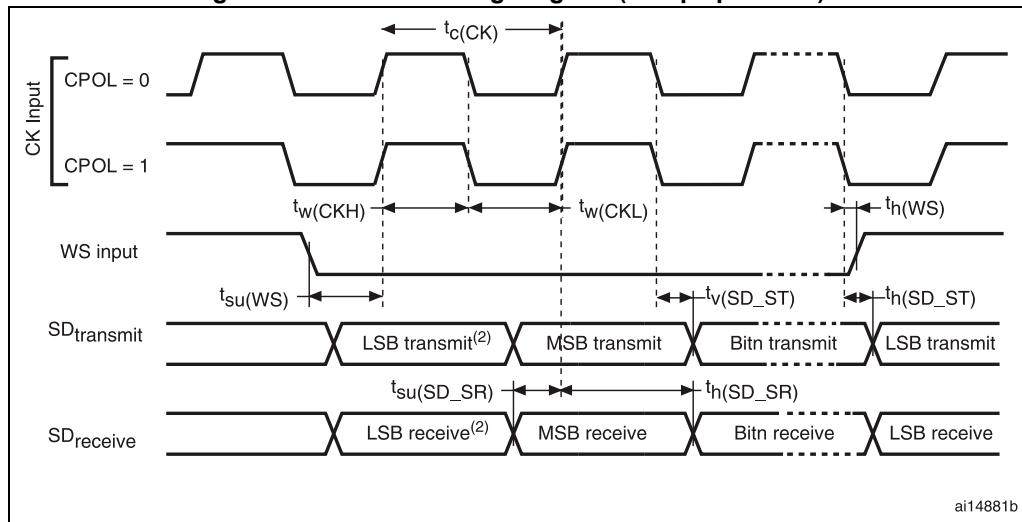
<b>Symbol</b>	<b>Parameter</b>	<b>Conditions</b>	<b>Min</b>	<b>Max</b>	<b>Unit</b>
$f_{MCK}$	I2S Main Clock Output		256 x 8K	256x $F_s$ <sup>(1)</sup>	MHz
$f_{CK}$	I2S clock frequency	Master data: 32 bits	-	64x $F_s$	MHz
		Slave data: 32 bits	-	64x $F_s$	
$D_{CK}$	I2S clock frequency duty cycle	Slave receiver, 48KHz	30	70	%
$t_{r(CK)}$	I2S clock rise time	Capacitive load CL=30pF	-	8	ns
$t_{f(CK)}$	I2S clock fall time			8	
$t_{v(WS)}$	WS valid time	Master mode	4	24	
$t_{h(WS)}$	WS hold time	Master mode	0	-	
$t_{su(WS)}$	WS setup time	Slave mode	15	-	
$t_{h(WS)}$	WS hold time	Slave mode	0	-	
$t_{su(SD\_MR)}$	Data input setup time	Master receiver	8	-	
$t_{su(SD\_SR)}$	Data input setup time	Slave receiver	9	-	
$t_{h(SD\_MR)}$	Data input hold time	Master receiver	5	-	
$t_{h(SD\_SR)}$		Slave receiver	4	-	
$t_{v(SD\_ST)}$	Data output valid time	Slave transmitter (after enable edge)	-	64	
$t_{h(SD\_ST)}$	Data output hold time	Slave transmitter (after enable edge)	22	-	
$t_{v(SD\_MT)}$	Data output valid time	Master transmitter (after enable edge)	-	12	
$t_{h(SD\_MT)}$	Data output hold time	Master transmitter (after enable edge)	8	-	

1. The maximum for 256x $F_s$  is 8 MHz

**Note:** Refer to the I2S section of the product reference manual for more details about the sampling frequency ( $F_s$ ),  $f_{MCK}$ ,  $f_{CK}$  and  $D_{CK}$  values. These values reflect only the digital peripheral behavior, source clock precision might slightly change them.  $D_{CK}$  depends mainly on the ODD bit value, digital contribution leads to a min of  $(I2SDIV/(2*I2SDIV+ODD))$  and a max of  $(I2SDIV+ODD)/(2*I2SDIV+ODD)$ .  $F_s$  max is supported for each mode/condition.

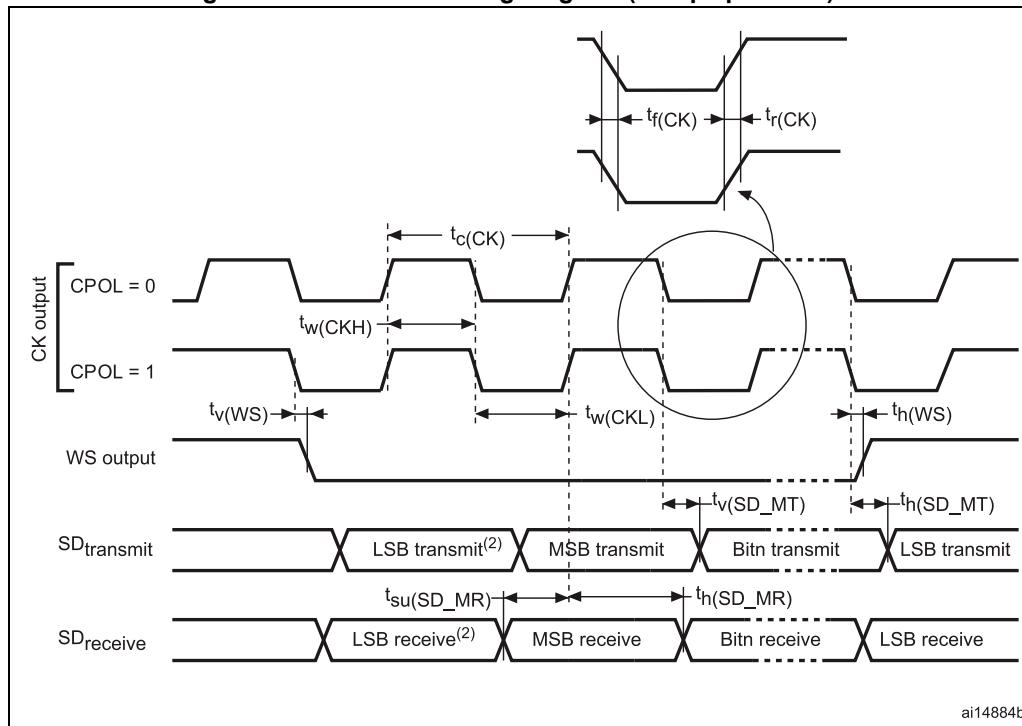
## STM32L151xC STM32L152xC

## Electrical characteristics

Figure 26. I<sup>2</sup>S slave timing diagram (Philips protocol)<sup>(1)</sup>

ai14881b

1. Measurement points are done at CMOS levels:  $0.3 \times V_{DD}$  and  $0.7 \times V_{DD}$ .
2. LSB transmit/receive of the previously transmitted byte. No LSB transmit/receive is sent before the first byte.

Figure 27. I<sup>2</sup>S master timing diagram (Philips protocol)<sup>(1)</sup>

ai14884b

1. Guaranteed by characterization results.
2. LSB transmit/receive of the previously transmitted byte. No LSB transmit/receive is sent before the first byte.

**Electrical characteristics****STM32L151xC STM32L152xC****6.3.17 12-bit ADC characteristics**

Unless otherwise specified, the parameters given in *Table 56* are guaranteed by design.

**Table 55. ADC clock frequency**

Symbol	Parameter	Conditions			Min	Max	Unit
$f_{ADC}$	ADC clock frequency	Voltage range 1 & 2	2.4 V $\leq V_{DDA} \leq$ 3.6 V	$V_{REF+} = V_{DDA}$	0.480	16	MHz
				$V_{REF+} < V_{DDA}$ $V_{REF+} > 2.4$ V		8	
				$V_{REF+} < V_{DDA}$ $V_{REF+} \leq 2.4$ V		4	
		1.8 V $\leq V_{DDA} \leq$ 2.4 V		$V_{REF+} = V_{DDA}$		8	
				$V_{REF+} < V_{DDA}$		4	
		Voltage range 3				4	

**Table 56. ADC characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$V_{DDA}$	Power supply	-	1.8	-	3.6	V
$V_{REF+}$	Positive reference voltage	-	1.8 <sup>(1)</sup>	-	$V_{DDA}$	
$V_{REF-}$	Negative reference voltage	-	-	$V_{SSA}$	-	$\mu A$
$I_{VDDA}$	Current on the $V_{DDA}$ input pin	-	-	1000	1450	
$I_{VREF}^{(2)}$	Current on the $V_{REF}$ input pin	Peak	-	400	700	$\mu A$
		Average	-		450	
$V_{AIN}$	Conversion voltage range <sup>(3)</sup>	-	0 <sup>(4)</sup>	-	$V_{REF+}$	V
$f_S$	12-bit sampling rate	Direct channels	-	-	1	Msps
		Multiplexed channels	-	-	0.76	
	10-bit sampling rate	Direct channels	-	-	1.07	Msps
		Multiplexed channels	-	-	0.8	
	8-bit sampling rate	Direct channels	-	-	1.23	Msps
		Multiplexed channels	-	-	0.89	
	6-bit sampling rate	Direct channels	-	-	1.45	Msps
		Multiplexed channels	-	-	1	

## STM32L151xC STM32L152xC

## Electrical characteristics

Table 56. ADC characteristics (continued)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
$t_S^{(5)}$	Sampling time	Direct channels $2.4 \text{ V} \leq V_{DDA} \leq 3.6 \text{ V}$	0.25	-	-	$\mu\text{s}$
		Multiplexed channels $2.4 \text{ V} \leq V_{DDA} \leq 3.6 \text{ V}$	0.56	-	-	
		Direct channels $1.8 \text{ V} \leq V_{DDA} \leq 2.4 \text{ V}$	0.56	-	-	
		Multiplexed channels $1.8 \text{ V} \leq V_{DDA} \leq 2.4 \text{ V}$	1	-	-	
		-	4	-	384	$1/f_{ADC}$
$t_{CONV}$	Total conversion time (including sampling time)	$f_{ADC} = 16 \text{ MHz}$	1	-	24.75	$\mu\text{s}$
		-	4 to 384 (sampling phase) +12 (successive approximation)			$1/f_{ADC}$
$C_{ADC}$	Internal sample and hold capacitor	Direct channels	-	16	-	$\text{pF}$
		Multiplexed channels	-		-	
$f_{TRIG}$	External trigger frequency Regular sequencer	12-bit conversions	-	-	$T_{conv+1}$	$1/f_{ADC}$
		6/8/10-bit conversions	-	-	$T_{conv}$	$1/f_{ADC}$
$f_{TRIG}$	External trigger frequency Injected sequencer	12-bit conversions	-	-	$T_{conv+2}$	$1/f_{ADC}$
		6/8/10-bit conversions	-	-	$T_{conv+1}$	$1/f_{ADC}$
$R_{AIN}^{(6)}$	Signal source impedance		-	-	50	$\text{k}\Omega$
$t_{lat}$	Injection trigger conversion latency	$f_{ADC} = 16 \text{ MHz}$	219	-	281	$\text{ns}$
		-	3.5	-	4.5	$1/f_{ADC}$
$t_{latr}$	Regular trigger conversion latency	$f_{ADC} = 16 \text{ MHz}$	156	-	219	$\text{ns}$
		-	2.5	-	3.5	$1/f_{ADC}$
$t_{STAB}$	Power-up time	-	-	-	3.5	$\mu\text{s}$

1. The  $V_{ref+}$  input can be grounded if neither the ADC nor the DAC are used (this allows to shut down an external voltage reference).
2. The current consumption through  $V_{REF}$  is composed of two parameters:
  - one constant (max 300  $\mu\text{A}$ )
  - one variable (max 400  $\mu\text{A}$ ), only during sampling time + 2 first conversion pulses
 So, peak consumption is  $300+400 = 700 \mu\text{A}$  and average consumption is  $300 + [(4 \text{ sampling} + 2) / 16] \times 400 = 450 \mu\text{A}$  at 1Msps
3.  $V_{REF+}$  can be internally connected to  $V_{DDA}$  and  $V_{REF-}$  can be internally connected to  $V_{SSA}$ , depending on the package. Refer to [Section 4: Pin descriptions](#) for further details.
4.  $V_{SSA}$  or  $V_{REF-}$  must be tied to ground.
5. Minimum sampling time is reached for an external input impedance limited to a value as defined in [Table 58: Maximum source impedance RAIN max.](#)
6. External impedance has another high value limitation when using short sampling time as defined in [Table 58: Maximum source impedance RAIN max.](#)

**Electrical characteristics****STM32L151xC STM32L152xC****Table 57. ADC accuracy<sup>(1)(2)</sup>**

Symbol	Parameter	Test conditions	Min <sup>(3)</sup>	Typ	Max <sup>(3)</sup>	Unit
ET	Total unadjusted error	$2.4 \text{ V} \leq V_{DDA} \leq 3.6 \text{ V}$ $2.4 \text{ V} \leq V_{REF+} \leq 3.6 \text{ V}$ $f_{ADC} = 8 \text{ MHz}, R_{AIN} = 50 \Omega$ $T_A = -40 \text{ to } 105^\circ\text{C}$	-	2	4	LSB
EO	Offset error		-	1	2	
EG	Gain error		-	1.5	3.5	
ED	Differential linearity error		-	1	2	
EL	Integral linearity error		-	1.7	3	
ENOB	Effective number of bits	$2.4 \text{ V} \leq V_{DDA} \leq 3.6 \text{ V}$ $V_{DDA} = V_{REF+}$ $f_{ADC} = 16 \text{ MHz}, R_{AIN} = 50 \Omega$ $T_A = -40 \text{ to } 105^\circ\text{C}$ $F_{input}=10\text{kHz}$	9.2	10	-	bits
SINAD	Signal-to-noise and distortion ratio		57.5	62	-	dB
SNR	Signal-to-noise ratio		57.5	62	-	
THD	Total harmonic distortion		-	-70	-65	
ENOB	Effective number of bits	$1.8 \text{ V} \leq V_{DDA} \leq 2.4 \text{ V}$ $V_{DDA} = V_{REF+}$ $f_{ADC} = 8 \text{ MHz or } 4 \text{ MHz}, R_{AIN} = 50 \Omega$ $T_A = -40 \text{ to } 105^\circ\text{C}$ $F_{input}=10\text{kHz}$	9.2	10	-	bits
SINAD	Signal-to-noise and distortion ratio		57.5	62	-	dB
SNR	Signal-to-noise ratio		57.5	62	-	
THD	Total harmonic distortion		-	-70	-65	
ET	Total unadjusted error	$2.4 \text{ V} \leq V_{DDA} \leq 3.6 \text{ V}$ $1.8 \text{ V} \leq V_{REF+} \leq 2.4 \text{ V}$ $f_{ADC} = 4 \text{ MHz}, R_{AIN} = 50 \Omega$ $T_A = -40 \text{ to } 105^\circ\text{C}$	-	4	6.5	LSB
EO	Offset error		-	2	4	
EG	Gain error		-	4	6	
ED	Differential linearity error		-	1	2	
EL	Integral linearity error		-	1.5	3	
ET	Total unadjusted error	$1.8 \text{ V} \leq V_{DDA} \leq 2.4 \text{ V}$ $1.8 \text{ V} \leq V_{REF+} \leq 2.4 \text{ V}$ $f_{ADC} = 4 \text{ MHz}, R_{AIN} = 50 \Omega$ $T_A = -40 \text{ to } 105^\circ\text{C}$	-	2	3	LSB
EO	Offset error		-	1	1.5	
EG	Gain error		-	1.5	2	
ED	Differential linearity error		-	1	2	
EL	Integral linearity error		-	1	1.5	

1. ADC DC accuracy values are measured after internal calibration.
2. ADC accuracy vs. negative injection current: Injecting a negative current on any analog input pins should be avoided as this significantly reduces the accuracy of the conversion being performed on another analog input. It is recommended to add a Schottky diode (pin to ground) to analog pins which may potentially inject negative currents.  
Any positive injection current within the limits specified for  $I_{INJ(PIN)}$  and  $\Sigma I_{INJ(PIN)}$  in [Section 6.3.12](#) does not affect the ADC accuracy.
3. Guaranteed by characterization results.

## STM32L151xC STM32L152xC

## Electrical characteristics

Figure 28. ADC accuracy characteristics

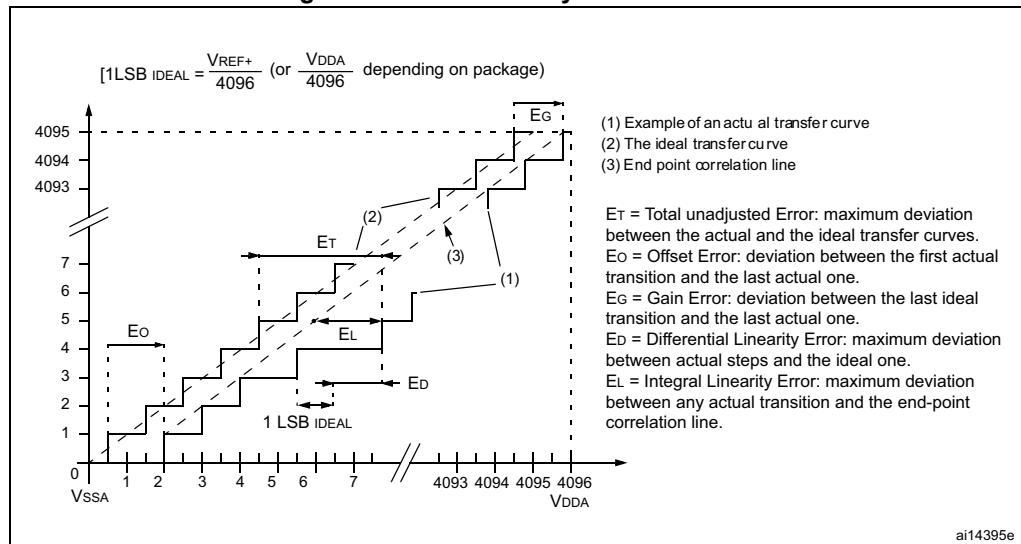
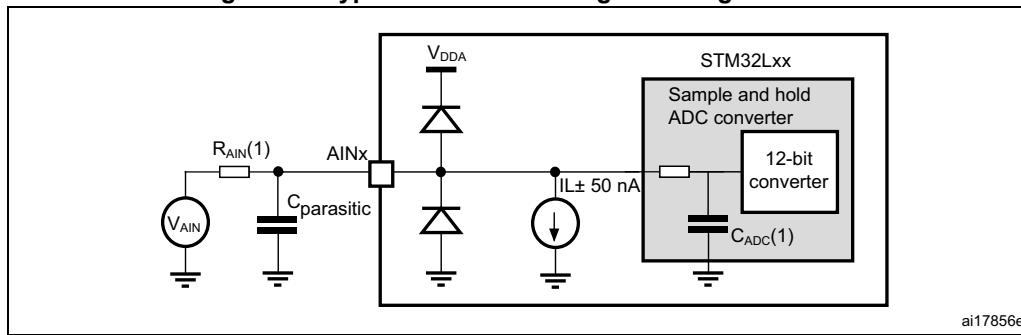


Figure 29. Typical connection diagram using the ADC

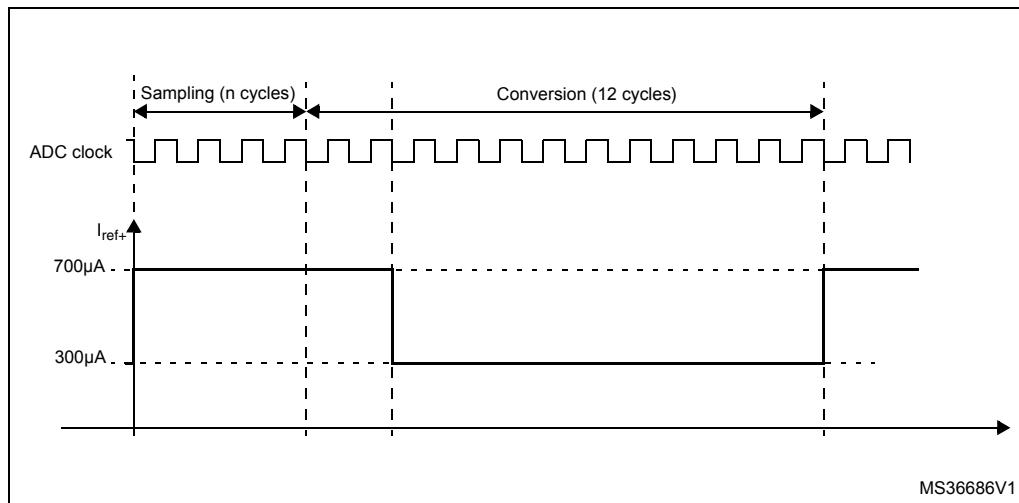


1. Refer to [Table 58: Maximum source impedance R<sub>A</sub><sub>IN</sub> max](#) for the value of  $R_{AIN}$  and [Table 56: ADC characteristics](#) for the value of  $C_{ADC}$ .
2.  $C_{parasitic}$  represents the capacitance of the PCB (dependent on soldering and PCB layout quality) plus the pad capacitance (roughly 7 pF). A high  $C_{parasitic}$  value will downgrade conversion accuracy. To remedy this,  $f_{ADC}$  should be reduced.

## Electrical characteristics

## STM32L151xC STM32L152xC

**Figure 30. Maximum dynamic current consumption on  $V_{REF+}$  supply pin during ADC conversion**



**Table 58. Maximum source impedance  $R_{AIN}$  max<sup>(1)</sup>**

Ts (μs)	$R_{AIN}$ max (kΩ)				Ts (cycles) $f_{ADC}=16$ MHz <sup>(2)</sup>	
	Multiplexed channels		Direct channels			
	2.4 V < $V_{DDA}$ < 3.6 V	1.8 V < $V_{DDA}$ < 2.4 V	2.4 V < $V_{DDA}$ < 3.6 V	1.8 V < $V_{DDA}$ < 2.4 V		
0.25	Not allowed	Not allowed	0.7	Not allowed	4	
0.5625	0.8	Not allowed	2.0	1.0	9	
1	2.0	0.8	4.0	3.0	16	
1.5	3.0	1.8	6.0	4.5	24	
3	6.8	4.0	15.0	10.0	48	
6	15.0	10.0	30.0	20.0	96	
12	32.0	25.0	50.0	40.0	192	
24	50.0	50.0	50.0	50.0	384	

1. Guaranteed by design.

2. Number of samples calculated for  $f_{ADC} = 16$  MHz. For  $f_{ADC} = 8$  and 4 MHz the number of sampling cycles can be reduced with respect to the minimum sampling time  $T_s$  (μs).

### General PCB design guidelines

Power supply decoupling should be performed as shown in [Figure 12](#). The applicable procedure depends on whether  $V_{REF+}$  is connected to  $V_{DDA}$  or not. The 100 nF capacitors should be ceramic (good quality). They should be placed as close as possible to the chip.

**STM32L151xC STM32L152xC****Electrical characteristics****6.3.18 DAC electrical specifications**

Data guaranteed by design, unless otherwise specified.

**Table 59. DAC characteristics**

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
$V_{DDA}$	Analog supply voltage	-		1.8	-	3.6	V
$V_{REF+}$	Reference supply voltage	$V_{REF+}$ must always be below $V_{DDA}$		1.8	-	3.6	
$V_{REF-}$	Lower reference voltage	-		$V_{SSA}$			
$I_{DDVREF+}^{(1)}$	Current consumption on $V_{REF+}$ supply $V_{REF+} = 3.3$ V	No load, middle code (0x800)		-	130	220	$\mu A$
		No load, worst code (0x000)		-	220	350	
$I_{DDA}^{(1)}$	Current consumption on $V_{DDA}$ supply $V_{DDA} = 3.3$ V	No load, middle code (0x800)		-	210	320	$\mu A$
		No load, worst code (0xF1C)		-	320	520	
$R_L$	Resistive load	DAC output buffer ON	Connected to $V_{SSA}$	5	-	-	$k\Omega$
			Connected to $V_{DDA}$	25	-	-	
$C_L^{(2)}$	Capacitive load	DAC output buffer ON	-	-	-	50	pF
$R_o$	Output impedance	DAC output buffer OFF	12	16	20	$k\Omega$	
$V_{DAC\_OUT}$	Voltage on DAC_OUT output	DAC output buffer ON		0.2	-	$V_{DDA} - 0.2$	V
		DAC output buffer OFF		0.5	-	$V_{REF+} - 1LSB$	mV
$DNL^{(1)}$	Differential non linearity <sup>(3)</sup>	$C_L \leq 50$ pF, $R_L \geq 5$ k $\Omega$ DAC output buffer ON		-	1.5	3	LSB
		No $R_L$ , $C_L \leq 50$ pF DAC output buffer OFF		-	1.5	3	
$INL^{(1)}$	Integral non linearity <sup>(4)</sup>	$C_L \leq 50$ pF, $R_L \geq 5$ k $\Omega$ DAC output buffer ON		-	2	4	
		No $R_L$ , $C_L \leq 50$ pF DAC output buffer OFF		-	2	4	
Offset <sup>(1)</sup>	Offset error at code 0x800 <sup>(5)</sup>	$C_L \leq 50$ pF, $R_L \geq 5$ k $\Omega$ DAC output buffer ON		-	$\pm 10$	$\pm 25$	LSB
		No $R_L$ , $C_L \leq 50$ pF DAC output buffer OFF		-	$\pm 5$	$\pm 8$	
Offset1 <sup>(1)</sup>	Offset error at code 0x001 <sup>(6)</sup>	No $R_L$ , $C_L \leq 50$ pF DAC output buffer OFF		-	$\pm 1.5$	$\pm 5$	

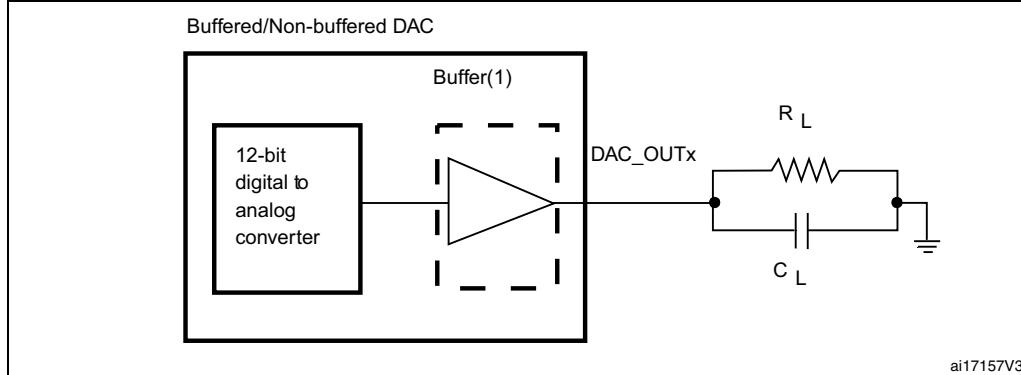
**Electrical characteristics****STM32L151xC STM32L152xC****Table 59. DAC characteristics (continued)**

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
dOffset/dT <sup>(1)</sup>	Offset error temperature coefficient (code 0x800)	V <sub>DDA</sub> = 3.3V V <sub>REF+</sub> = 3.0V T <sub>A</sub> = 0 to 50 °C DAC output buffer OFF	-20	-10	0	µV/°C
		V <sub>DDA</sub> = 3.3V V <sub>REF+</sub> = 3.0V T <sub>A</sub> = 0 to 50 °C DAC output buffer ON	0	20	50	
Gain <sup>(1)</sup>	Gain error <sup>(7)</sup>	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 5 kΩ DAC output buffer ON	-	+0.1 / -0.2%	+0.2 / -0.5%	%
		No R <sub>L</sub> , C <sub>L</sub> ≤ 50 pF DAC output buffer OFF	-	+0 / -0.2%	+0 / -0.4%	
dGain/dT <sup>(1)</sup>	Gain error temperature coefficient	V <sub>DDA</sub> = 3.3V V <sub>REF+</sub> = 3.0V T <sub>A</sub> = 0 to 50 °C DAC output buffer OFF	-10	-2	0	µV/°C
		V <sub>DDA</sub> = 3.3V V <sub>REF+</sub> = 3.0V T <sub>A</sub> = 0 to 50 °C DAC output buffer ON	-40	-8	0	
TUE <sup>(1)</sup>	Total unadjusted error	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 5 kΩ DAC output buffer ON	-	12	30	LSB
		No R <sub>L</sub> , C <sub>L</sub> ≤ 50 pF DAC output buffer OFF	-	8	12	
tSETTLING	Settling time (full scale: for a 12-bit code transition between the lowest and the highest input codes till DAC_OUT reaches final value ±1LSB)	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 5 kΩ	-	7	12	µs
Update rate	Max frequency for a correct DAC_OUT change (95% of final value) with 1 LSB variation in the input code	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 5 kΩ	-	-	1	MspS
tWAKEUP	Wakeup time from off state (setting the ENx bit in the DAC Control register) <sup>(8)</sup>	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 5 kΩ	-	9	15	µs
PSRR+	V <sub>DDA</sub> supply rejection ratio (static DC measurement)	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 5 kΩ	-	-60	-35	dB

1. Data based on characterization results.
2. Connected between DAC\_OUT and VSSA.
3. Difference between two consecutive codes - 1 LSB.

**STM32L151xC STM32L152xC****Electrical characteristics**

4. Difference between measured value at Code i and the value at Code i on a line drawn between Code 0 and last Code 4095.
5. Difference between the value measured at Code (0x800) and the ideal value =  $V_{REF+}/2$ .
6. Difference between the value measured at Code (0x001) and the ideal value.
7. Difference between ideal slope of the transfer function and measured slope computed from code 0x000 and 0xFFFF when buffer is OFF, and from code giving 0.2 V and  $(V_{DDA} - 0.2)$  V when buffer is ON.
8. In buffered mode, the output can overshoot above the final value for low input code (starting from min value).

**Figure 31. 12-bit buffered /non-buffered DAC**

1. The DAC integrates an output buffer that can be used to reduce the output impedance and to drive external loads directly without the use of an external operational amplifier. The buffer can be bypassed by configuring the BOFFx bit in the DAC\_CR register.

**6.3.19 Operational amplifier characteristics****Table 60. Operational amplifier characteristics**

Symbol	Parameter		Condition <sup>(1)</sup>	Min <sup>(2)</sup>	Typ	Max <sup>(2)</sup>	Unit
CMIR	Common mode input range		-	0	-	$V_{DD}$	
$V_{I_{OFFSET}}$	Input offset voltage	Maximum calibration range	-	-	-	$\pm 15$	mV
		After offset calibration	-	-	-	$\pm 1.5$	
$\Delta V_{I_{OFFSET}}$	Input offset voltage drift	Normal mode	-	-	-	$\pm 40$	$\mu V/^\circ C$
		Low-power mode	-	-	-	$\pm 80$	
$I_{IB}$	Input current bias	Dedicated input	75 °C	-	-	1	nA
		General purpose input		-	-	10	
$I_{LOAD}$	Drive current	Normal mode	-	-	-	500	$\mu A$
		Low-power mode	-	-	-	100	
$I_{DD}$	Consumption	Normal mode	No load, quiescent mode	-	100	220	$\mu A$
		Low-power mode		-	30	60	
CMRR	Common mode rejection ration	Normal mode	-	-	-85	-	dB
		Low-power mode	-	-	-90	-	

## Electrical characteristics

## STM32L151xC STM32L152xC

Table 60. Operational amplifier characteristics (continued)

Symbol	Parameter		Condition <sup>(1)</sup>	Min <sup>(2)</sup>	Typ	Max <sup>(2)</sup>	Unit
PSRR	Power supply rejection ratio	Normal mode	DC	-	-85	-	dB
		Low-power mode		-	-90	-	
GBW	Bandwidth	Normal mode	$V_{DD} > 2.4 \text{ V}$	400	1000	3000	kHz
		Low-power mode		150	300	800	
		Normal mode	$V_{DD} < 2.4 \text{ V}$	200	500	2200	
		Low-power mode		70	150	800	
SR	Slew rate	Normal mode	$V_{DD} > 2.4 \text{ V}$ (between 0.1 V and $V_{DD}-0.1 \text{ V}$ )	-	700	-	V/ms
		Low-power mode	$V_{DD} > 2.4 \text{ V}$	-	100	-	
		Normal mode	$V_{DD} < 2.4 \text{ V}$	-	300	-	
		Low-power mode		-	50	-	
AO	Open loop gain	Normal mode		55	100	-	dB
		Low-power mode		65	110	-	
R <sub>L</sub>	Resistive load	Normal mode	$V_{DD} < 2.4 \text{ V}$	4	-	-	kΩ
		Low-power mode		20	-	-	
C <sub>L</sub>	Capacitive load		-	-	-	50	pF
V <sub>OH_SAT</sub>	High saturation voltage	Normal mode	I <sub>LOAD</sub> = max or R <sub>L</sub> = min	$V_{DD}-100$	-	-	mV
		Low-power mode		$V_{DD}-50$	-	-	
V <sub>OL_SAT</sub>	Low saturation voltage	Normal mode		-	-	100	
		Low-power mode		-	-	50	
φm	Phase margin		-	-	60	-	°
GM	Gain margin		-	-	-12	-	dB
t <sub>OFFTRIM</sub>	Offset trim time: during calibration, minimum time needed between two steps to have 1 mV accuracy		-	-	1	-	ms
t <sub>WAKEUP</sub>	Wakeup time	Normal mode	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 4 kΩ	-	10	-	μs
		Low-power mode	C <sub>L</sub> ≤ 50 pF, R <sub>L</sub> ≥ 20 kΩ	-	30	-	

1. Operating conditions are limited to junction temperature (0 °C to 105 °C) when  $V_{DD}$  is below 2 V. Otherwise to the full ambient temperature range (-40 °C to 85 °C, -40 °C to 105 °C).

2. Guaranteed by characterization results.

**STM32L151xC STM32L152xC****Electrical characteristics****6.3.20 Temperature sensor characteristics****Table 61. Temperature sensor calibration values**

Calibration value name	Description	Memory address
TS_CAL1	TS ADC raw data acquired at temperature of 30 °C $\pm$ 5 °C $V_{DDA} = 3 \text{ V} \pm 10 \text{ mV}$	0x1FF8 00FA - 0x1FF8 00FB
TS_CAL2	TS ADC raw data acquired at temperature of 110 °C $\pm$ 5 °C $V_{DDA} = 3 \text{ V} \pm 10 \text{ mV}$	0x1FF8 00FE - 0x1FF8 00FF

**Table 62. Temperature sensor characteristics**

Symbol	Parameter	Min	Typ	Max	Unit
$T_L^{(1)}$	$V_{SENSE}$ linearity with temperature	-	$\pm 1$	$\pm 2$	°C
Avg_Slope <sup>(1)</sup>	Average slope	1.48	1.61	1.75	mV/°C
$V_{110}$	Voltage at 110°C $\pm$ 5°C <sup>(2)</sup>	612	626.8	641.5	mV
$I_{DDA(TEMP)}^{(3)}$	Current consumption	-	3.4	6	µA
$t_{START}^{(3)}$	Startup time	-	-	10	µs
$T_{S\_temp}^{(3)}$	ADC sampling time when reading the temperature	4	-	-	

1. Guaranteed by characterization results.

2. Measured at  $V_{DD} = 3 \text{ V} \pm 10 \text{ mV}$ .  $V_{110}$  ADC conversion result is stored in the TS\_CAL2 byte.

3. Guaranteed by design.

**6.3.21 Comparator****Table 63. Comparator 1 characteristics**

Symbol	Parameter	Conditions	Min <sup>(1)</sup>	Typ	Max <sup>(1)</sup>	Unit
$V_{DDA}$	Analog supply voltage	-	1.65		3.6	V
$R_{400K}$	$R_{400K}$ value	-	-	400	-	kΩ
$R_{10K}$	$R_{10K}$ value	-	-	10	-	
$V_{IN}$	Comparator 1 input voltage range	-	0.6	-	$V_{DDA}$	V
$t_{START}$	Comparator startup time	-	-	7	10	µs
$t_d$	Propagation delay <sup>(2)</sup>	-	-	3	10	
$V_{offset}$	Comparator offset	-	-	$\pm 3$	$\pm 10$	mV
$dV_{offset}/dt$	Comparator offset variation in worst voltage stress conditions	$V_{DDA} = 3.6 \text{ V}$ $V_{IN+} = 0 \text{ V}$ $V_{IN-} = V_{REFINT}$ $T_A = 25 \text{ }^{\circ}\text{C}$	0	1.5	10	$\text{mV}/1000 \text{ h}$
$I_{COMP1}$	Current consumption <sup>(3)</sup>	-	-	160	260	nA

**Electrical characteristics****STM32L151xC STM32L152xC**

1. Guaranteed by characterization results.
2. The delay is characterized for 100 mV input step with 10 mV overdrive on the inverting input, the non-inverting input set to the reference.
3. Comparator consumption only. Internal reference voltage not included.

**Table 64. Comparator 2 characteristics**

Symbol	Parameter	Conditions	Min	Typ	Max <sup>(1)</sup>	Unit
$V_{DDA}$	Analog supply voltage	-	1.65	-	3.6	V
$V_{IN}$	Comparator 2 input voltage range	-	0	-	$V_{DDA}$	V
$t_{START}$	Comparator startup time	Fast mode	-	15	20	$\mu s$
		Slow mode	-	20	25	
$t_d$ slow	Propagation delay <sup>(2)</sup> in slow mode	$1.65 \text{ V} \leq V_{DDA} \leq 2.7 \text{ V}$	-	1.8	3.5	$\mu s$
		$2.7 \text{ V} \leq V_{DDA} \leq 3.6 \text{ V}$	-	2.5	6	
$t_d$ fast	Propagation delay <sup>(2)</sup> in fast mode	$1.65 \text{ V} \leq V_{DDA} \leq 2.7 \text{ V}$	-	0.8	2	
		$2.7 \text{ V} \leq V_{DDA} \leq 3.6 \text{ V}$	-	1.2	4	
$V_{offset}$	Comparator offset error		-	$\pm 4$	$\pm 20$	mV
$d\text{Threshold}/dt$	Threshold voltage temperature coefficient	$V_{DDA} = 3.3 \text{ V}$ $T_A = 0 \text{ to } 50^\circ\text{C}$ $V^- = V_{REFINT} \cdot 3/4$ $V^- = V_{REFINT} \cdot 1/2$ $V^- = V_{REFINT} \cdot 1/4$	-	15	100	ppm/ $^\circ\text{C}$
$I_{COMP2}$	Current consumption <sup>(3)</sup>	Fast mode	-	3.5	5	$\mu A$
		Slow mode	-	0.5	2	

1. Guaranteed by characterization results.
2. The delay is characterized for 100 mV input step with 10 mV overdrive on the inverting input, the non-inverting input set to the reference.
3. Comparator consumption only. Internal reference voltage (necessary for comparator operation) is not included.

**STM32L151xC STM32L152xC****Electrical characteristics****6.3.22 LCD controller**

The device embeds a built-in step-up converter to provide a constant LCD reference voltage independently from the  $V_{DD}$  voltage. An external capacitor  $C_{ext}$  must be connected to the  $V_{LCD}$  pin to decouple this converter.

**Table 65. LCD controller characteristics**

Symbol	Parameter	Min	Typ	Max	Unit
$V_{LCD}$	LCD external voltage	-	-	3.6	V
$V_{LCD0}$	LCD internal reference voltage 0	-	2.6	-	
$V_{LCD1}$	LCD internal reference voltage 1	-	2.73	-	
$V_{LCD2}$	LCD internal reference voltage 2	-	2.86	-	
$V_{LCD3}$	LCD internal reference voltage 3	-	2.98	-	
$V_{LCD4}$	LCD internal reference voltage 4	-	3.12	-	
$V_{LCD5}$	LCD internal reference voltage 5	-	3.26	-	
$V_{LCD6}$	LCD internal reference voltage 6	-	3.4	-	
$V_{LCD7}$	LCD internal reference voltage 7	-	3.55	-	
$C_{ext}$	$V_{LCD}$ external capacitance	0.1	-	2	$\mu F$
$I_{LCD}^{(1)}$	Supply current at $V_{DD} = 2.2$ V	-	3.3	-	$\mu A$
	Supply current at $V_{DD} = 3.0$ V	-	3.1	-	
$R_{Htot}^{(2)}$	Low drive resistive network overall value	5.28	6.6	7.92	$M\Omega$
$R_L^{(2)}$	High drive resistive network total value	192	240	288	$k\Omega$
$V_{44}$	Segment/Common highest level voltage	-	-	$V_{LCD}$	V
$V_{34}$	Segment/Common 3/4 level voltage	-	3/4 $V_{LCD}$	-	V
$V_{23}$	Segment/Common 2/3 level voltage	-	2/3 $V_{LCD}$	-	
$V_{12}$	Segment/Common 1/2 level voltage	-	1/2 $V_{LCD}$	-	
$V_{13}$	Segment/Common 1/3 level voltage	-	1/3 $V_{LCD}$	-	
$V_{14}$	Segment/Common 1/4 level voltage	-	1/4 $V_{LCD}$	-	
$V_0$	Segment/Common lowest level voltage	0	-	-	
$\Delta V_{xx}^{(3)}$	Segment/Common level voltage error $T_A = -40$ to $105$ °C	-	-	± 50	mV

1. LCD enabled with 3 V internal step-up active, 1/8 duty, 1/4 bias, division ratio= 64, all pixels active, no LCD connected.
2. Guaranteed by design.
3. Guaranteed by characterization results.

## Package information

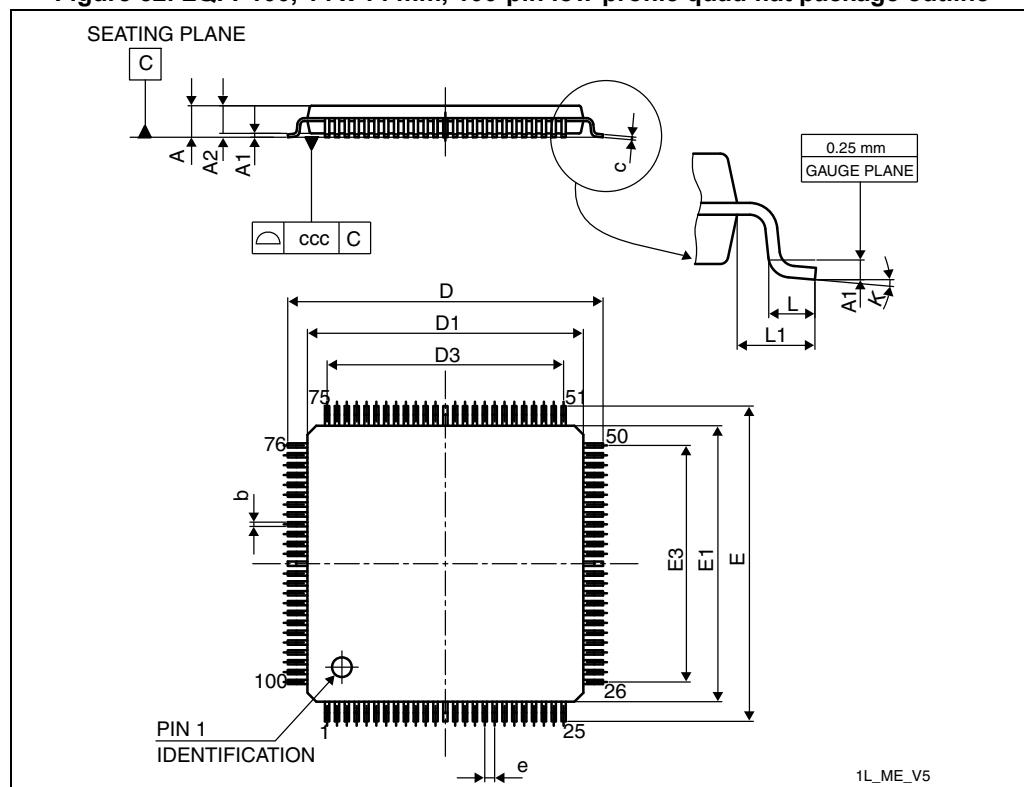
## STM32L151xC STM32L152xC

## 7 Package information

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK® specifications, grade definitions and product status are available at: [www.st.com](http://www.st.com).  
ECOPACK® is an ST trademark.

### 7.1 LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package information

**Figure 32. LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package outline**



1. Drawing is not to scale.

**Table 66. LQPF100, 14 x 14 mm, 100-pin low-profile quad flat package mechanical data**

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
A	-	-	1.600	-	-	0.0630
A1	0.050	-	0.150	0.0020	-	0.0059
A2	1.350	1.400	1.450	0.0531	0.0551	0.0571

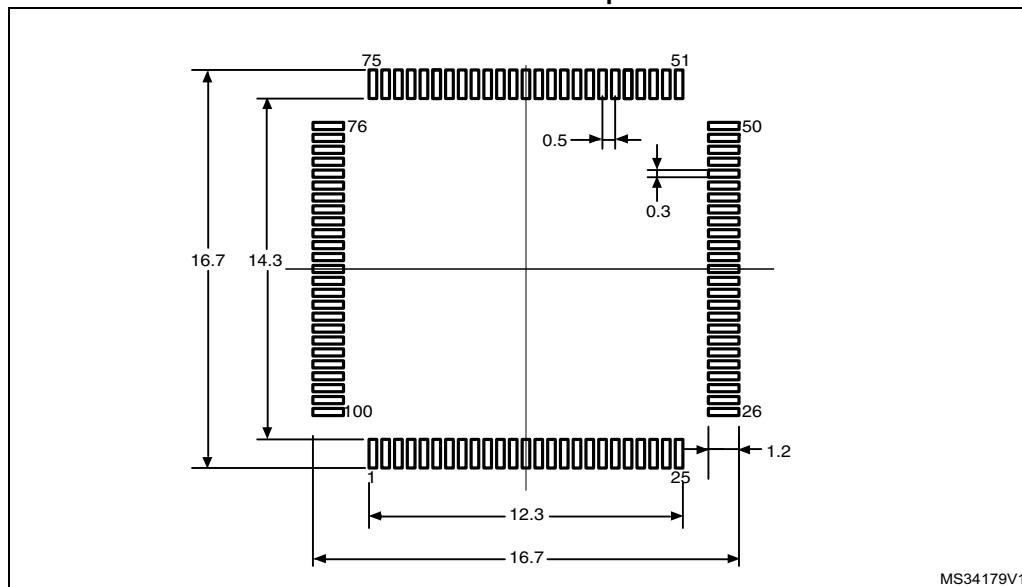
## STM32L151xC STM32L152xC

## Package information

**Table 66. LQPF100, 14 x 14 mm, 100-pin low-profile quad flat package mechanical data (continued)**

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
b	0.170	0.220	0.270	0.0067	0.0087	0.0106
c	0.090	-	0.200	0.0035	-	0.0079
D	15.800	16.000	16.200	0.6220	0.6299	0.6378
D1	13.800	14.000	14.200	0.5433	0.5512	0.5591
D3	-	12.000	-	-	0.4724	-
E	15.800	16.000	16.200	0.6220	0.6299	0.6378
E1	13.800	14.000	14.200	0.5433	0.5512	0.5591
E3	-	12.000	-	-	0.4724	-
e	-	0.500	-	-	0.0197	-
L	0.450	0.600	0.750	0.0177	0.0236	0.0295
L1	-	1.000	-	-	0.0394	-
k	0.0°	3.5°	7.0°	0.0°	3.5°	7.0°
ccc	-	-	0.080	-	-	0.0031

1. Values in inches are converted from mm and rounded to 4 decimal digits.

**Figure 33. LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package recommended footprint**

1. Dimensions are in millimeters.

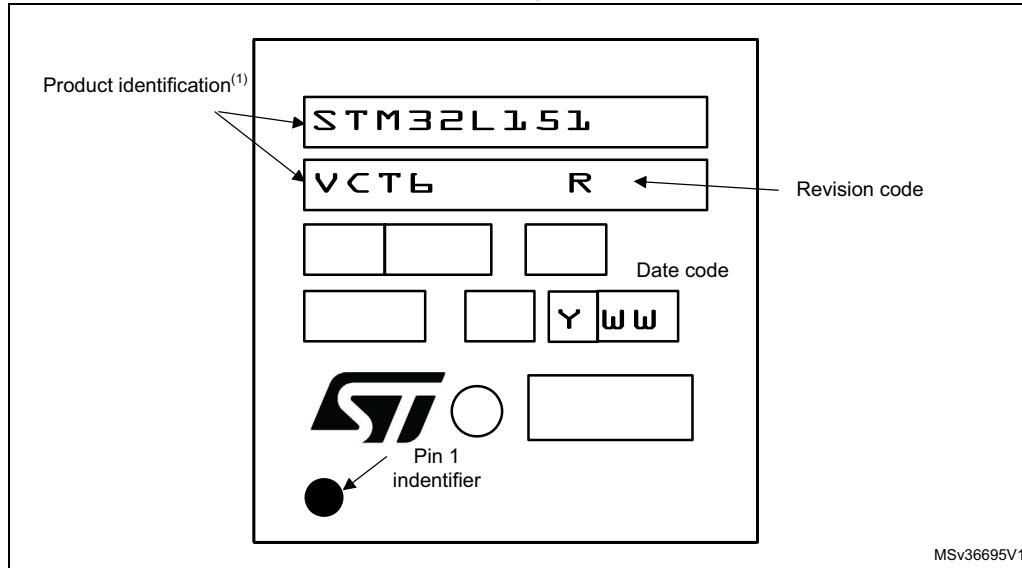
MS34179V1

**Package information****STM32L151xC STM32L152xC****LQFP100 device marking**

The following figure gives an example of topside marking orientation versus pin 1 identifier location.

Other optional marking or inset/upset marks, which identify the parts throughout supply chain operations, are not indicated below.

**Figure 34. LQFP100, 14 x 14 mm, 100-pin low-profile quad flat package top view example**

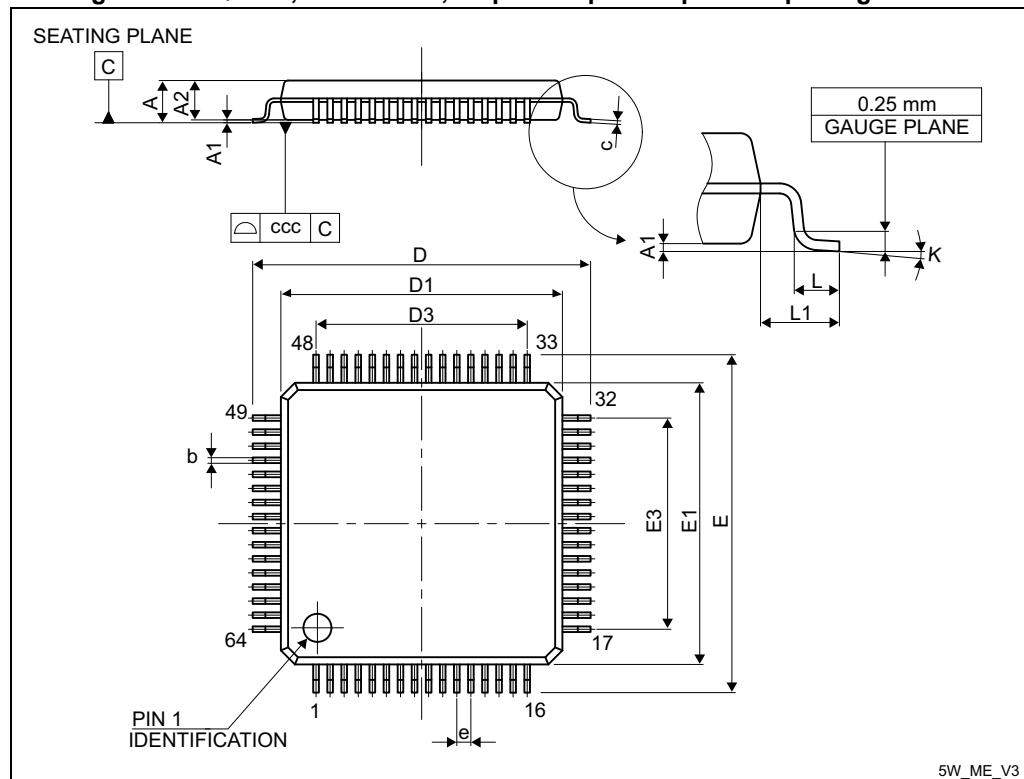


MSv36695V1

1. Parts marked as ES or E or accompanied by an engineering sample notification letter are not yet qualified and therefore not approved for use in production. ST is not responsible for any consequences resulting from such use. In no event will ST be liable for the customer using any of these engineering samples in production. ST's Quality department must be contacted prior to any decision to use these engineering samples to run a qualification activity.

## 7.2 LQFP64, 10 x 10 mm, 64-pin low-profile quad flat package information

Figure 35. LQFP64, 10 x 10 mm, 64-pin low-profile quad flat package outline



1. Drawing is not to scale.

Table 67. LQFP64, 10 x 10 mm 64-pin low-profile quad flat package mechanical data

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
A	-	-	1.600	-	-	0.0630
A1	0.050	-	0.150	0.0020	-	0.0059
A2	1.350	1.400	1.450	0.0531	0.0551	0.0571
b	0.170	0.220	0.270	0.0067	0.0087	0.0106
c	0.090	-	0.200	0.0035	-	0.0079
D	-	12.000	-	-	0.4724	-
D1	-	10.000	-	-	0.3937	-
D3	-	7.500	-	-	0.2953	-
E	-	12.000	-	-	0.4724	-
E1	-	10.000	-	-	0.3937	-

## Package information

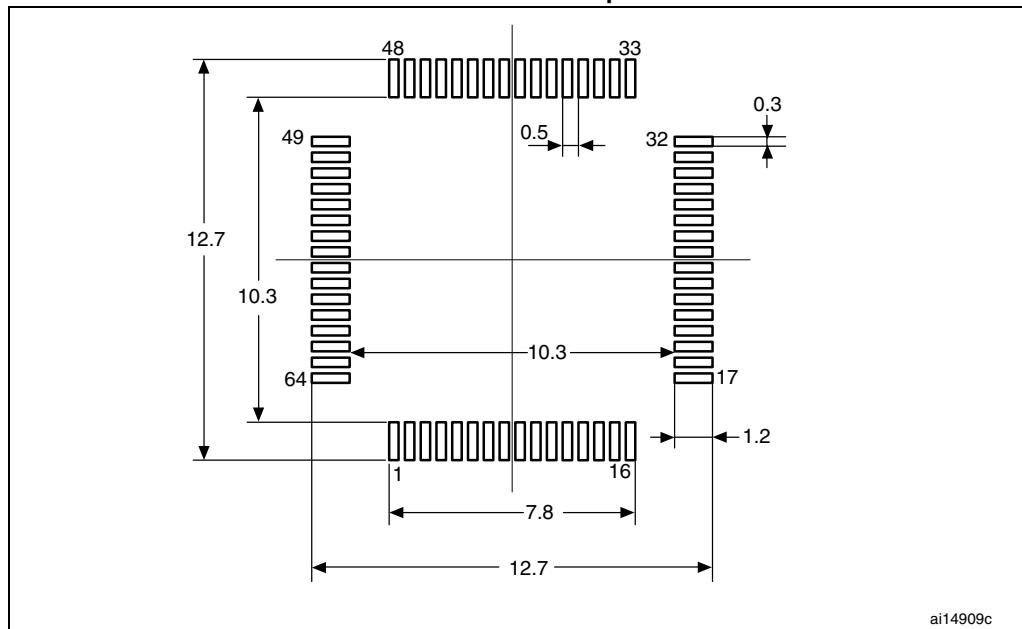
## STM32L151xC STM32L152xC

**Table 67. LQFP64, 10 x 10 mm 64-pin low-profile quad flat package mechanical data (continued)**

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
E3	-	7.500	-	-	0.2953	-
e	-	0.500	-	-	0.0197	-
K	0°	3.5°	7°	0°	3.5°	7°
L	0.450	0.600	0.750	0.0177	0.0236	0.0295
L1	-	1.000	-	-	0.0394	-
ccc	-	-	0.080	-	-	0.0031

1. Values in inches are converted from mm and rounded to 4 decimal digits.

**Figure 36. LQFP64, 10 x 10 mm, 64-pin low-profile quad flat package recommended footprint**



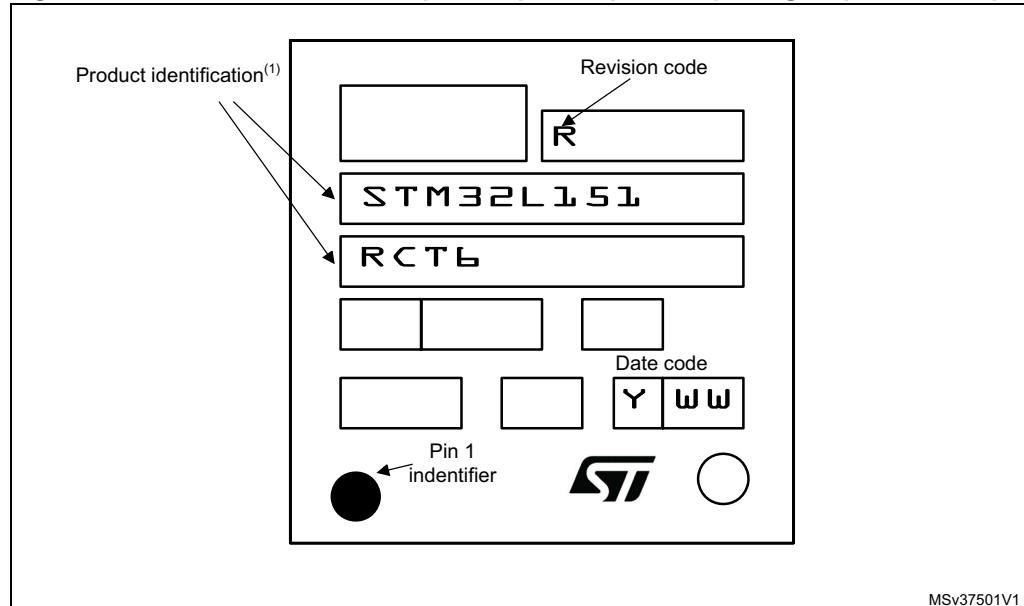
1. Dimensions are in millimeters.

**STM32L151xC STM32L152xC****Package information****LQFP64 device marking**

The following figure gives an example of topside marking orientation versus pin 1 identifier location.

Other optional marking or inset/upset marks, which identify the parts throughout supply chain operations, are not indicated below.

**Figure 37. LQFP64 10 x 10 mm, 64-pin low-profile quad flat package top view example**



MSv37501V1

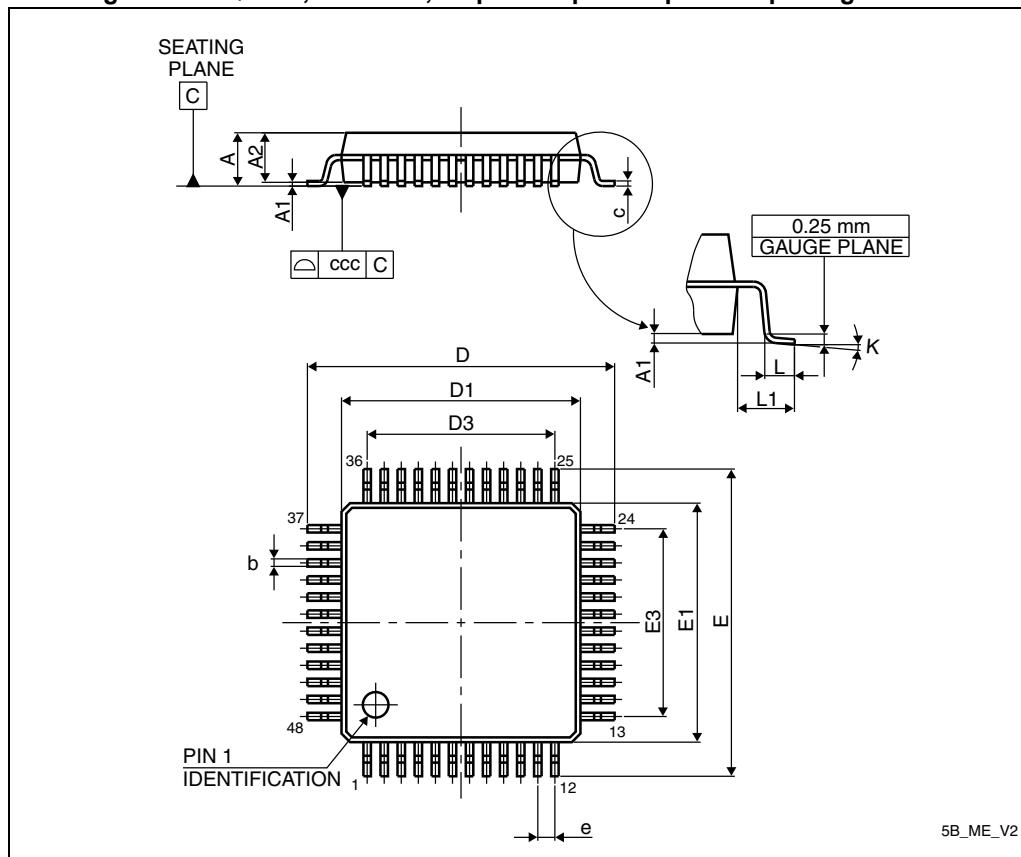
1. Parts marked as ES or E or accompanied by an engineering sample notification letter are not yet qualified and therefore not approved for use in production. ST is not responsible for any consequences resulting from such use. In no event will ST be liable for the customer using any of these engineering samples in production. ST's Quality department must be contacted prior to any decision to use these engineering samples to run a qualification activity.

## Package information

STM32L151xC STM32L152xC

### 7.3 LQFP48, 7 x 7 mm, 48-pin low-profile quad flat package information

Figure 38. LQFP48, 7 x 7 mm, 48-pin low-profile quad flat package outline



1. Drawing is not to scale.

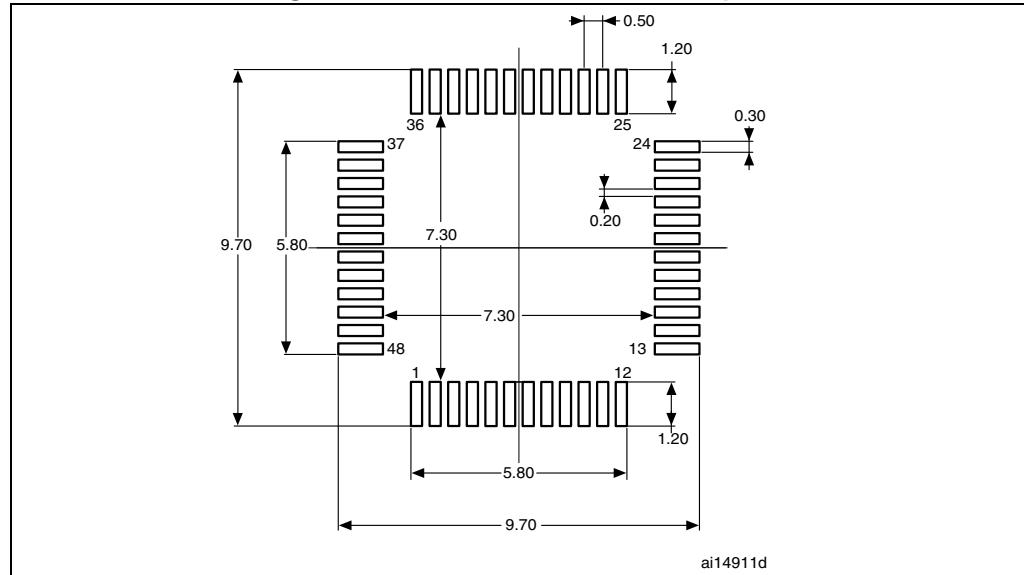
## STM32L151xC STM32L152xC

## Package information

**Table 68. LQFP48, 7 x 7 mm, 48-pin low-profile quad flat package mechanical data**

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
A	-	-	1.600	-	-	0.0630
A1	0.050	-	0.150	0.0020	-	0.0059
A2	1.350	1.400	1.450	0.0531	0.0551	0.0571
b	0.170	0.220	0.270	0.0067	0.0087	0.0106
c	0.090	-	0.200	0.0035	-	0.0079
D	8.800	9.000	9.200	0.3465	0.3543	0.3622
D1	6.800	7.000	7.200	0.2677	0.2756	0.2835
D3	-	5.500	-	-	0.2165	-
E	8.800	9.000	9.200	0.3465	0.3543	0.3622
E1	6.800	7.000	7.200	0.2677	0.2756	0.2835
E3	-	5.500	-	-	0.2165	-
e	-	0.500	-	-	0.0197	-
L	0.450	0.600	0.750	0.0177	0.0236	0.0295
L1	-	1.000	-	-	0.0394	-
k	0°	3.5°	7°	0°	3.5°	7°
ccc	-	-	0.080	-	-	0.0031

1. Values in inches are converted from mm and rounded to 4 decimal digits.

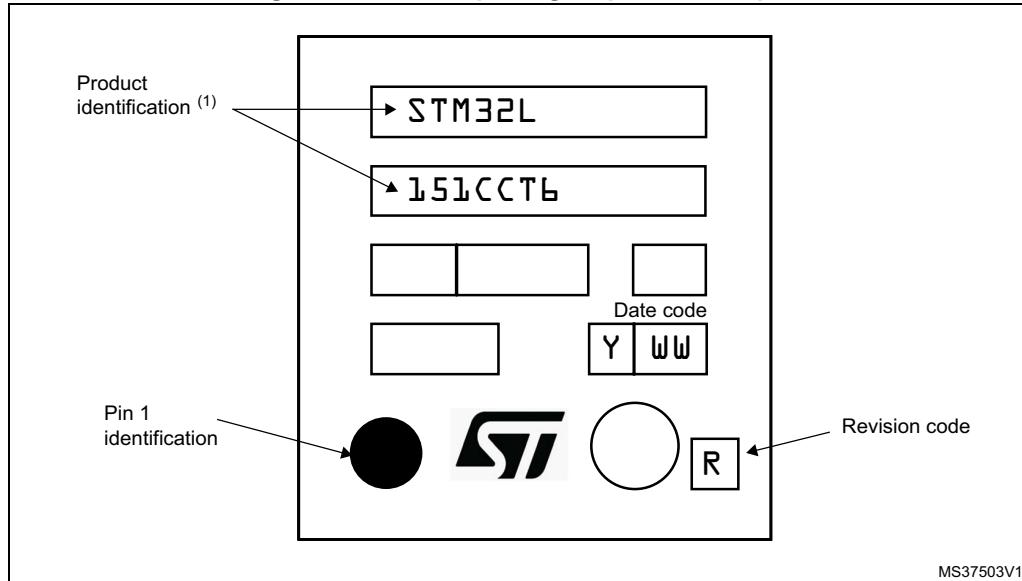
**Figure 39. LQFP48 recommended footprint**

1. Dimensions are in millimeters.

**Package information****STM32L151xC STM32L152xC****LQFP48 device marking**

The following figure gives an example of topside marking orientation versus pin 1 identifier location.

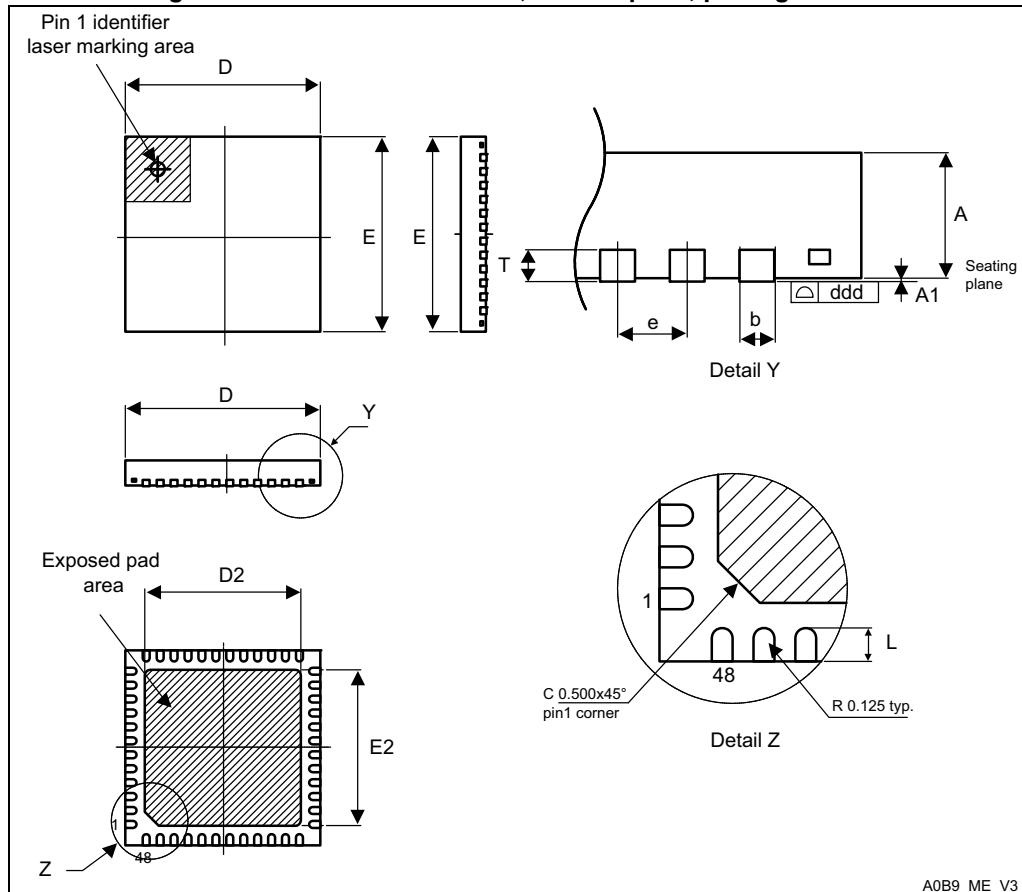
Other optional marking or inset/upset marks, which identify the parts throughout supply chain operations, are not indicated below.

**Figure 40. LQFP48 package top view example**

1. Parts marked as ES or E or accompanied by an engineering sample notification letter are not yet qualified and therefore not approved for use in production. ST is not responsible for any consequences resulting from such use. In no event will ST be liable for the customer using any of these engineering samples in production. ST's Quality department must be contacted prior to any decision to use these engineering samples to run a qualification activity.

## 7.4 UFQFPN48 7 x 7 mm, 0.5 mm pitch, package information

Figure 41. UFQFPN48 7 x 7 mm, 0.5 mm pitch, package outline



1. Drawing is not to scale.
2. All leads/pads should also be soldered to the PCB to improve the lead/pad solder joint life.
3. There is an exposed die pad on the underside of the UFQFPN package. It is recommended to connect and solder this back-side pad to PCB ground.

## Package information

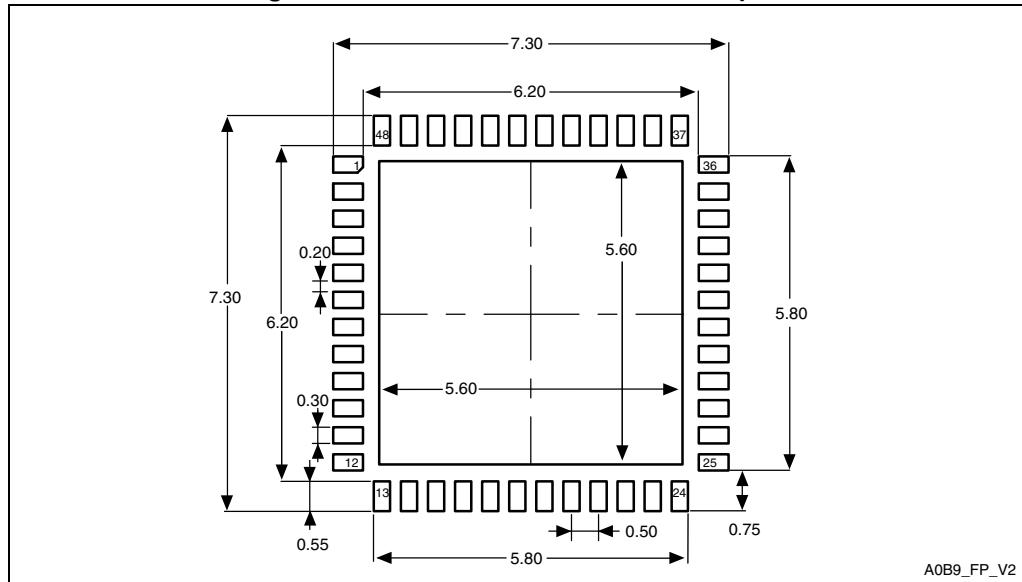
## STM32L151xC STM32L152xC

**Table 69. UFQFPN48 – ultra thin fine pitch quad flat pack no-lead 7 × 7 mm,  
0.5 mm pitch package mechanical data**

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
A	0.500	0.550	0.600	0.0197	0.0217	0.0236
A1	0.000	0.020	0.050	0.0000	0.0008	0.0020
D	6.900	7.000	7.100	0.2717	0.2756	0.2795
E	6.900	7.000	7.100	0.2717	0.2756	0.2795
D2	5.500	5.600	5.700	0.2165	0.2205	0.2244
E2	5.500	5.600	5.700	0.2165	0.2205	0.2244
L	0.300	0.400	0.500	0.0118	0.0157	0.0197
T	-	0.152	-	-	0.0060	-
b	0.200	0.250	0.300	0.0079	0.0098	0.0118
e	-	0.500	-	-	0.0197	-
ddd	-	-	0.080	-	-	0.0031

1. Values in inches are converted from mm and rounded to 4 decimal digits.

**Figure 42. UFQFPN48 recommended footprint**



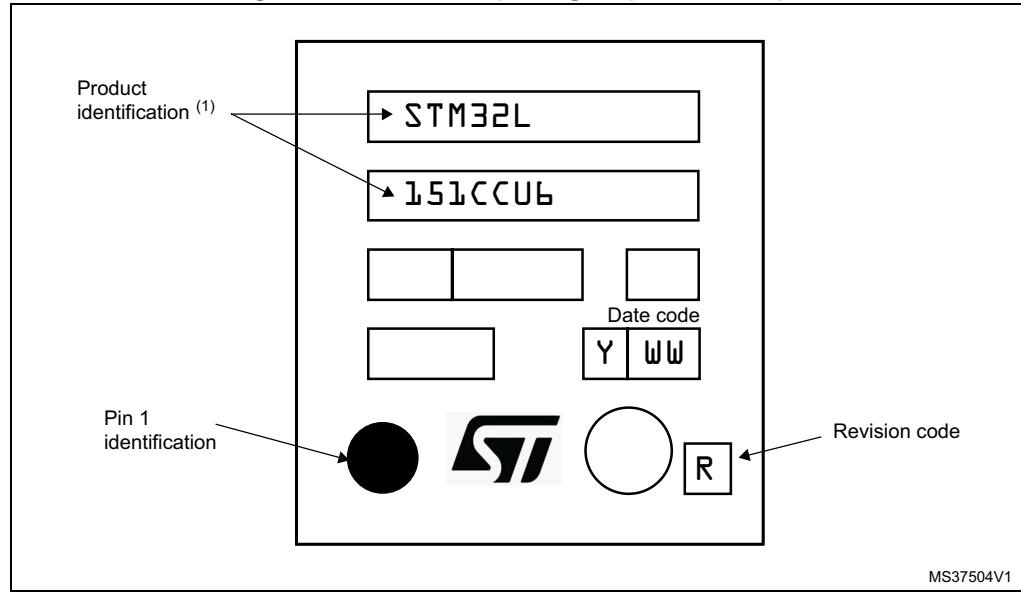
1. Dimensions are in millimeters.

**STM32L151xC STM32L152xC****Package information****UFQFPN48 device marking**

The following figure gives an example of topside marking orientation versus pin 1 identifier location.

Other optional marking or inset/upset marks, which identify the parts throughout supply chain operations, are not indicated below.

**Figure 43. UFQFPN48 package top view example**



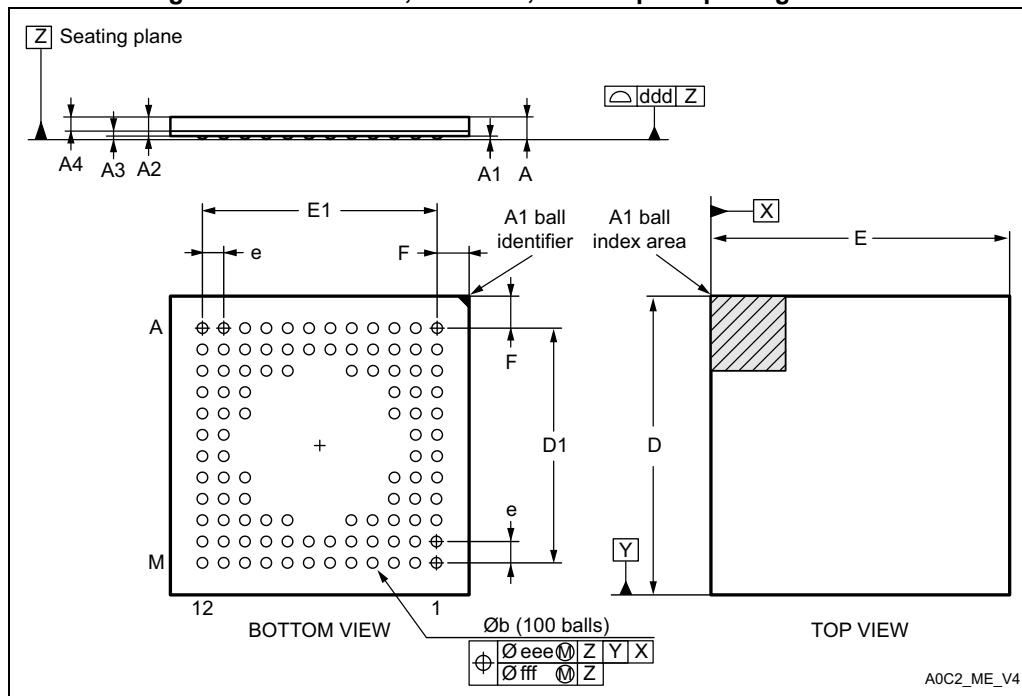
1. Parts marked as ES or E or accompanied by an engineering sample notification letter are not yet qualified and therefore not approved for use in production. ST is not responsible for any consequences resulting from such use. In no event will ST be liable for the customer using any of these engineering samples in production. ST's Quality department must be contacted prior to any decision to use these engineering samples to run a qualification activity.

## Package information

## STM32L151xC STM32L152xC

## 7.5 UFBGA100, 7 x 7 mm, 100-ball ultra thin, fine pitch ball grid array package information

Figure 44. UFBGA100, 7 x 7 mm, 0.5 mm pitch package outline



1. Drawing is not to scale.

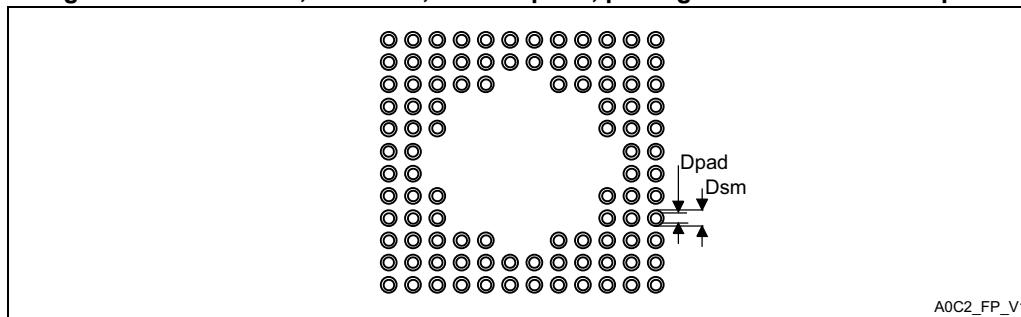
Table 70. UFBGA100, 7 x 7 mm, 0.5 mm pitch package mechanical data

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
A	0.460	0.530	0.600	0.0181	0.0209	0.0236
A1	0.050	0.080	0.110	0.0020	0.0031	0.0043
A2	0.400	0.450	0.500	0.0157	0.0177	0.0197
A3	0.080	0.130	0.180	0.0031	0.0051	0.0071
A4	0.270	0.320	0.370	0.0106	0.0126	0.0146
b	0.200	0.250	0.300	0.0079	0.0098	0.0118
D	6.950	7.000	7.050	0.2736	0.2756	0.2776
D1	5.450	5.500	5.550	0.2146	0.2165	0.2185
E	6.950	7.000	7.050	0.2736	0.2756	0.2776
E1	5.450	5.500	5.550	0.2146	0.2165	0.2185
e	-	0.500	-	-	0.0197	-
F	0.700	0.750	0.800	0.0276	0.0295	0.0315
ddd	-	-	0.100	-	-	0.0039

**STM32L151xC STM32L152xC****Package information****Table 70. UFBGA100, 7 x 7 mm, 0.5 mm pitch package mechanical data (continued)**

<b>Symbol</b>	<b>millimeters</b>			<b>inches<sup>(1)</sup></b>		
	<b>Min</b>	<b>Typ</b>	<b>Max</b>	<b>Min</b>	<b>Typ</b>	<b>Max</b>
eee	-	-	0.150	-	-	0.0059
fff	-	-	0.050	-	-	0.0020

1. Values in inches are converted from mm and rounded to 4 decimal digits.

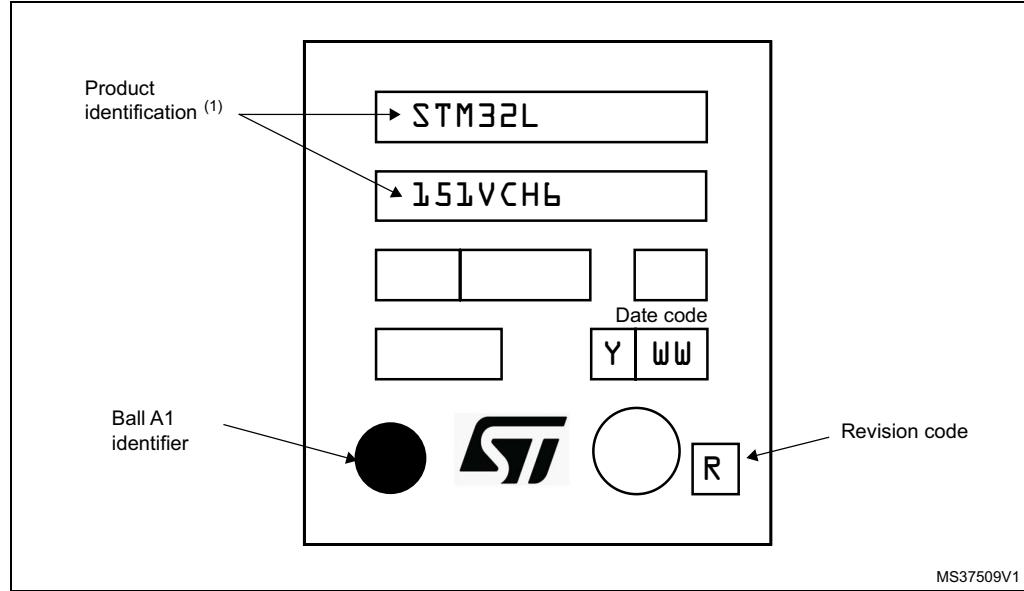
**Figure 45. UFBGA100, 7 x 7 mm, 0.5 mm pitch, package recommended footprint****Table 71. UFBGA100, 7 x 7 mm, 0.50 mm pitch, recommended PCB design rules**

<b>Dimension</b>	<b>Recommended values</b>
Pitch	0.5
Dpad	0.280 mm
Dsm	0.370 mm typ. (depends on the soldermask registration tolerance)
Stencil opening	0.280 mm
Stencil thickness	Between 0.100 mm and 0.125 mm

**Package information****STM32L151xC STM32L152xC****UFBGA100 device marking**

The following figure gives an example of topside marking orientation versus ball A1 identifier location.

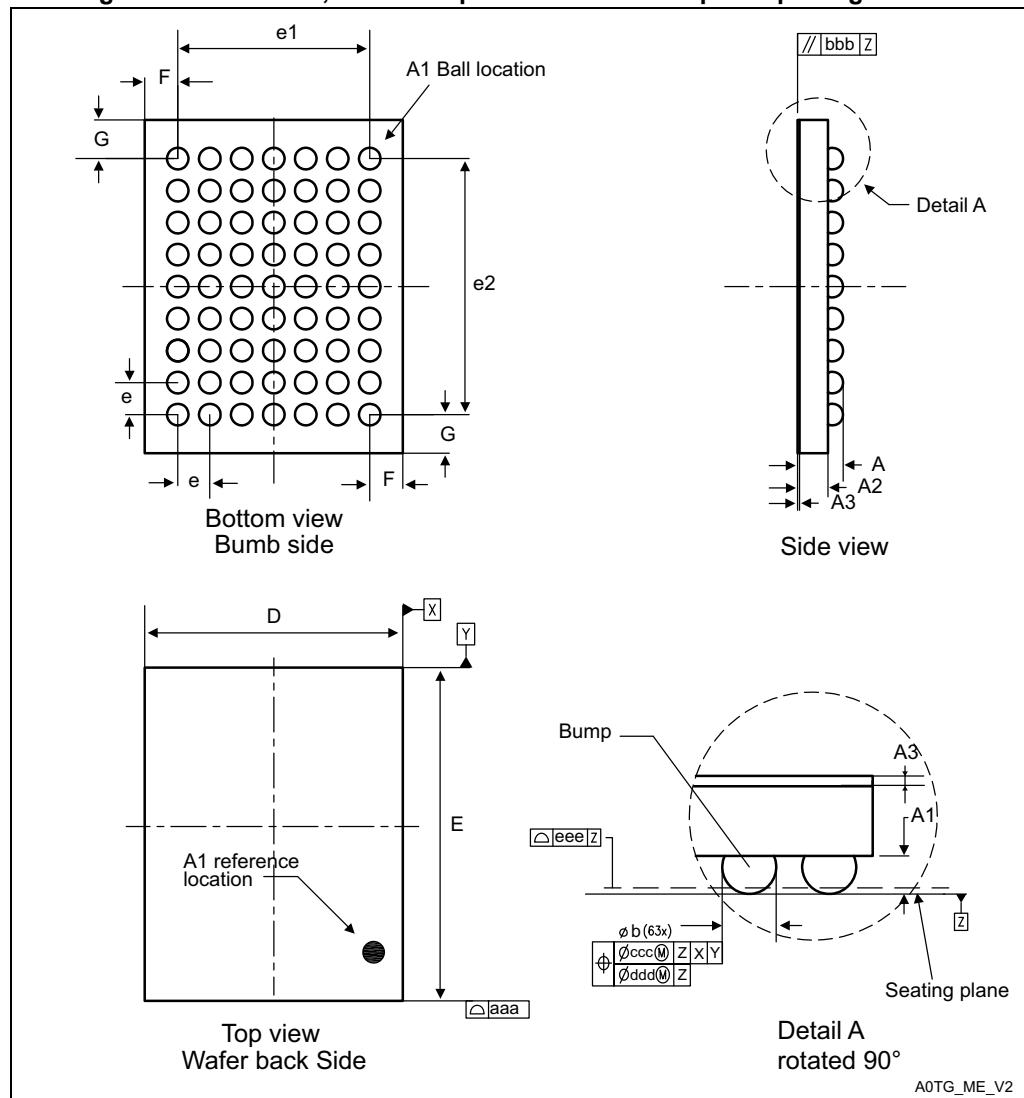
Other optional marking or inset/upset marks, which identify the parts throughout supply chain operations, are not indicated below.

**Figure 46. UFBGA100, 7 x 7 mm, 0.5 mm pitch, package top view example**

1. Parts marked as ES or E or accompanied by an engineering sample notification letter are not yet qualified and therefore not approved for use in production. ST is not responsible for any consequences resulting from such use. In no event will ST be liable for the customer using any of these engineering samples in production. ST's Quality department must be contacted prior to any decision to use these engineering samples to run a qualification activity.

## 7.6 WLCSP63, 0.400 mm pitch wafer level chip size package information

Figure 47. WLCSP63, 0.400 mm pitch wafer level chip size package outline



1. Drawing is not to scale.

**Package information****STM32L151xC STM32L152xC****Table 72. WLCSP63, 0.400 mm pitch wafer level chip size package mechanical data**

Symbol	millimeters			inches <sup>(1)</sup>		
	Min	Typ	Max	Min	Typ	Max
A	0.540	0.570	0.600	0.0213	0.0224	0.0236
A1	-	0.190	-	-	0.0075	-
A2	-	0.380	-	-	0.0150	-
A3	-	0.025	-	-	0.0010	-
Øb	0.240	0.270	0.300	0.0094	0.0106	0.0118
D	3.193	3.228	3.263	0.1257	0.1271	0.1285
E	4.129	4.164	4.199	0.1626	0.1639	0.1653
e	-	0.400	-	-	0.0157	-
e1	-	2.400	-	-	0.0945	-
e2	-	3.200	-	-	0.1260	-
F	-	0.414	-	-	0.0163	-
G	-	0.482	-		0.0190	-
aaa	-	-	0.100	-	-	0.0039
bbb	-	-	0.100	-	-	0.0039
ccc	-	-	0.100	-	-	0.0039
ddd	-	-	0.050	-	-	0.0020
eee	-	-	0.050	-	-	0.0020

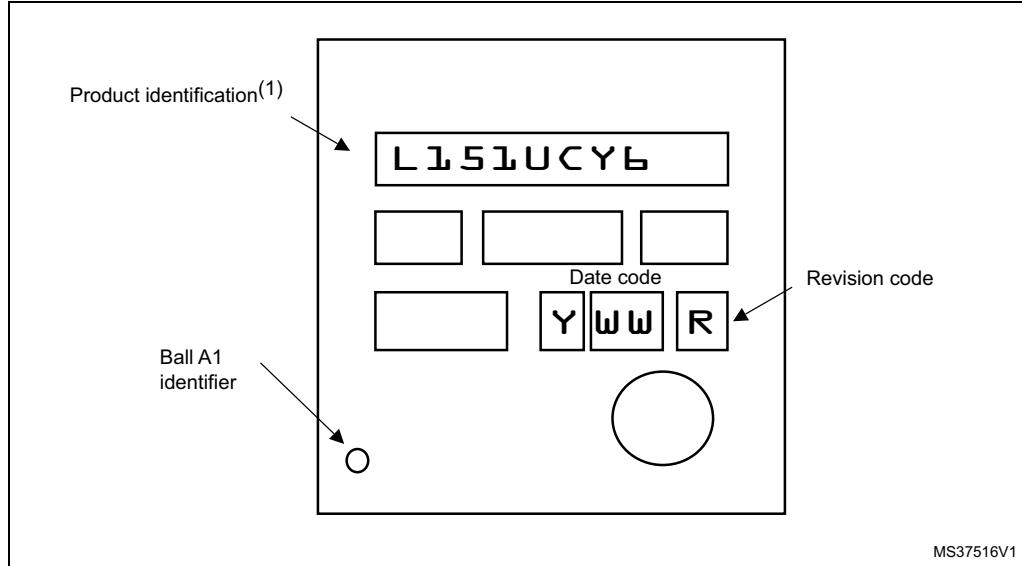
1. Values in inches are converted from mm and rounded to 4 decimal digits.

**STM32L151xC STM32L152xC****Package information****WLCSP63 device marking**

The following figure gives an example of topside marking orientation versus ball A1 identifier location.

Other optional marking or inset/upset marks, which identify the parts throughout supply chain operations, are not indicated below.

**Figure 48. WLCSP63 device marking example**



1. Parts marked as ES or E or accompanied by an engineering sample notification letter are not yet qualified and therefore not approved for use in production. ST is not responsible for any consequences resulting from such use. In no event will ST be liable for the customer using any of these engineering samples in production. ST's Quality department must be contacted prior to any decision to use these engineering samples to run a qualification activity.

**Package information****STM32L151xC STM32L152xC**

## 7.7 Thermal characteristics

The maximum chip-junction temperature,  $T_J$  max, in degrees Celsius, may be calculated using the following equation:

$$T_J \text{ max} = T_A \text{ max} + (P_D \text{ max} \times \Theta_{JA})$$

Where:

- $T_A$  max is the maximum ambient temperature in °C,
- $\Theta_{JA}$  is the package junction-to-ambient thermal resistance, in °C/W,
- $P_D$  max is the sum of  $P_{INT}$  max and  $P_{I/O}$  max ( $P_D$  max =  $P_{INT}$  max +  $P_{I/O}$  max),
- $P_{INT}$  max is the product of  $I_{DD}$  and  $V_{DD}$ , expressed in Watts. This is the maximum chip internal power.

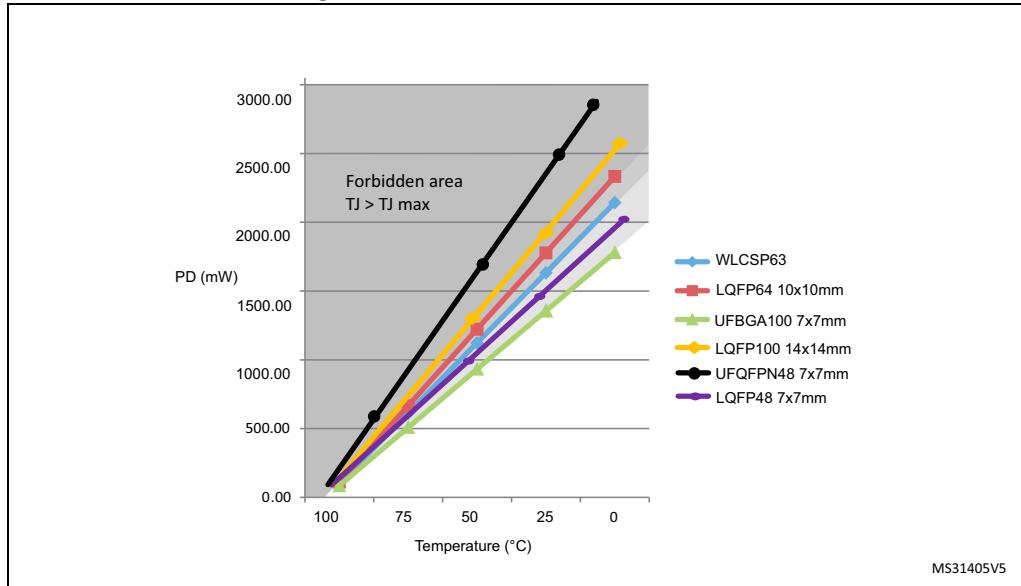
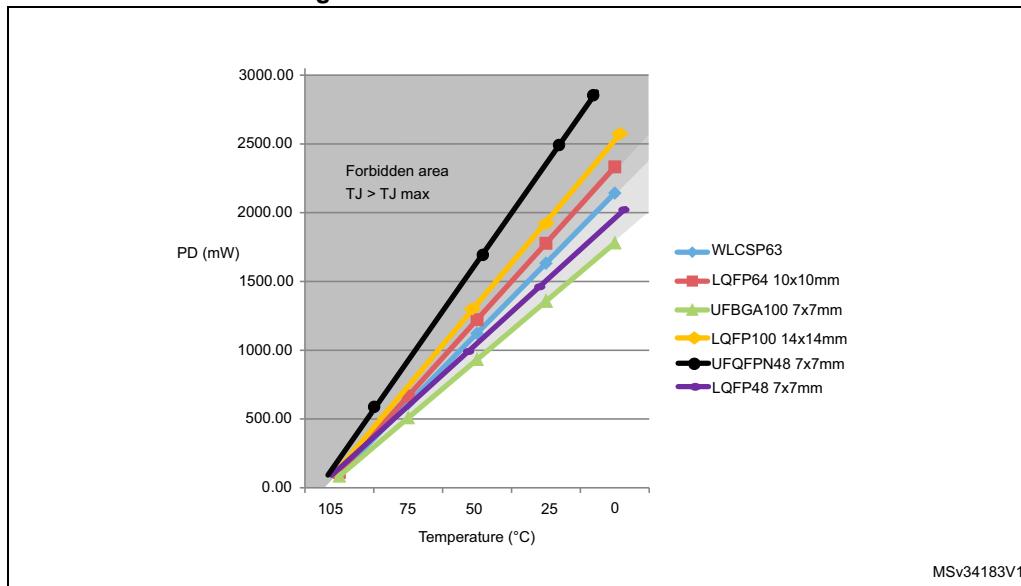
$P_{I/O}$  max represents the maximum power dissipation on output pins where:

$$P_{I/O} \text{ max} = \sum (V_{OL} \times I_{OL}) + \sum ((V_{DD} - V_{OH}) \times I_{OH}),$$

taking into account the actual  $V_{OL}$  /  $I_{OL}$  and  $V_{OH}$  /  $I_{OH}$  of the I/Os at low and high level in the application.

**Table 73. Thermal characteristics**

Symbol	Parameter	Value	Unit
$\Theta_{JA}$	<b>Thermal resistance junction-ambient</b> UFBGA100 - 7 x 7 mm	59	°C/W
	<b>Thermal resistance junction-ambient</b> LQFP100 - 14 x 14 mm / 0.5 mm pitch	43	
	<b>Thermal resistance junction-ambient</b> LQFP64 - 10 x 10 mm / 0.5 mm pitch	46	
	<b>Thermal resistance junction-ambient</b> WLCSP63 - 0.400 mm pitch	49	
	<b>Thermal resistance junction-ambient</b> LQFP48 - 7 x 7 mm / 0.5 mm pitch	55	
	<b>Thermal resistance junction-ambient</b> UFQFPN48 - 7 x 7 mm / 0.5 mm pitch	33	

**STM32L151xC STM32L152xC****Package information****Figure 49. Thermal resistance suffix 6****Figure 50. Thermal resistance suffix 7****7.7.1 Reference document**

JESD51-2 Integrated Circuits Thermal Test Method Environment Conditions - Natural Convection (Still Air). Available from [www.jedec.org](http://www.jedec.org).

**Part numbering****STM32L151xC STM32L152xC**

## 8 Part numbering

**Table 74. STM32L151xC and STM32L152xC ordering information scheme**

Example:	STM32	L	151	R	C	T	6	D	TR
<b>Device family</b>	STM32								
STM32 = ARM-based 32-bit microcontroller									
<b>Product type</b>		L							
L = Low-power									
<b>Device subfamily</b>			151						
151: Devices without LCD									
152: Devices with LCD									
<b>Pin count</b>									
C = 48 pins									
U = 63 pins									
R = 64 pins									
V = 100 pins									
<b>Flash memory size</b>									
C = 256 Kbytes of Flash memory									
<b>Package</b>									
H = BGA									
T = LQFP									
Y = WLCSP									
U = UFQFPN									
<b>Temperature range</b>									
6 = Industrial temperature range, -40 to 85 °C									
7 = Industrial temperature range, -40 to 105 °C									
<b>Options</b>									
No character = $V_{DD}$ range: 1.8 to 3.6 V and BOR enabled									
D = $V_{DD}$ range: 1.65 to 3.6 V and BOR disabled									
<b>Packing</b>									
TR = tape and reel									
No character = tray or tube									

For a list of available options (speed, package, etc.) or for further information on any aspect of this device, please contact the nearest ST sales office.

## 9 Revision History

**Table 75. Document revision history**

Date	Revision	Changes
21-Feb-2012	1	<p>Initial release.</p>
12-Oct-2012	2	<p>Added WLCSP63 package.</p> <p>Updated <a href="#">Figure 1: Ultra-low-power STM32L162xC block diagram</a>.</p> <p>Changed maximum number of touch sensing channels to 34, and updated <a href="#">Table 2: Ultralow power STM32L15xxC device features and peripheral counts</a>.</p> <p>Added <a href="#">Table 4: Functionalities depending on the working mode (from Run/active down to standby)</a>, and <a href="#">Table 3: Range depending on dynamic voltage scaling</a>.</p> <p>Updated <a href="#">Section 3.10: ADC (analog-to-digital converter)</a> to add <a href="#">Section 3.10.1: Temperature sensor</a> and <a href="#">Section 3.10.2: Internal voltage reference (VREFINT)</a>.</p> <p>Updated <a href="#">Figure 3: STM32L162VC LQFP100 pinout</a>.</p> <p><a href="#">Table 10: STM32L15xxC pin definitions</a>: updated name of reference manual in footnote 5.</p> <p>Changed I2C1_SMBAI into I2C1_SMBA in <a href="#">Table 10: STM32L15xxC pin definitions</a>.</p> <p>Modified PB10/11/12 for AFIO4 alternate function, and replaced LBAR by NADV for AFIO12 in <a href="#">Table 10: Alternate function input/output</a>.</p> <p>Removed caution note below <a href="#">Figure 8: Power supply scheme</a>.</p> <p>Added <a href="#">Note 2 in Table 15: Embedded reset and power control block characteristics</a>.</p> <p>Updated <a href="#">Table 14: General operating conditions</a>.</p> <p>Updated <a href="#">Table 22: Typical and maximum current consumptions in Stop mode</a> and added <a href="#">Note 6</a>. Updated <a href="#">Table 23: Typical and maximum current consumptions in Standby mode</a>. Updated <math>t_{WUSTOP}</math> in <a href="#">Table 10: STM32L15xxC pin definitions</a>.</p> <p>Updated <a href="#">Table 26: Peripheral current consumption</a>.</p> <p>Updated <a href="#">Table 60: SPI characteristics</a>, added <a href="#">Note 1</a> and <a href="#">Note 3</a>, and applied <a href="#">Note 2</a> to <math>t_r(SCK)</math>, <math>t_f(SCK)</math>, <math>t_w(SCKH)</math>, <math>t_w(SCKL)</math>, <math>t_{su(MI)}</math>, <math>t_{su(SI)}</math>, <math>t_h(MI)</math>, and <math>t_h(SI)</math>.</p> <p>Added <a href="#">Table 61: I2S characteristics</a>, <a href="#">Figure 29: I2S slave timing diagram (Philips protocol)(1)</a> and <a href="#">Figure 30: I2S master timing diagram (Philips protocol)(1)</a>.</p> <p>Updated <a href="#">Table 72: Temperature sensor characteristics</a>.</p> <p>Added <a href="#">Figure 40: Thermal resistance</a>.</p>

## Revision History

## STM32L151xC STM32L152xC

**Table 75. Document revision history (continued)**

Date	Revision	Changes
01-Feb-2013	3	<p>Removed AHB1/AHB2 and corrected typo on APB1/APB2 in:<a href="#">Figure 1: Ultra-low-power STM32L162xC block diagram-low-power STM32L162xC block diagram</a></p> <p>Updated "OP amp" line in <a href="#">Table 4: Functionalities depending on the working mode (from Run/active down to standby)</a></p> <p>Added IWDG and WWWDG rows in <a href="#">Table 4: Functionalities depending on the working mode (from Run/active down to standby)</a></p> <p>Updated address range in <a href="#">Table 7: Internal voltage reference measured values</a></p> <p>The comment "HSE = 16 MHz(2) (PLL ON for fHCLK above 16 MHz)" replaced by "fHSE = fHCLK up to 16 MHz included, fHSE = fHCLK/2 above 16 MHz (PLL ON)(2)" in table <a href="#">Table 27: Current consumption in Sleep mode</a></p> <p>replaced pin names D7,C7,C6,C8,B8,A8 respectively by D11,D10,C12,B12,A12,A11 in column UFBGA100 of <a href="#">Table 9: STM32L15xxC pin definitions</a></p> <p>Added more alternate functions supported on pin K3 and M4 for UFBGA100 package in <a href="#">Table 9: STM32L15xxC pin definitions</a></p> <p>Added part number STM32L151CC in <a href="#">Table 1: Device summary</a></p> <p>Updated Stop mode current to 1.5 <math>\mu</math>A in <a href="#">Ultra-low-power platform</a></p> <p>Updated entire <a href="#">Section 7: Package information</a></p>
02-Sep-2013	4	<p>Removed UFBGA132 and LQFP144 packages</p> <p>Removed first sentence in <a href="#">Section : I2C interface characteristics</a></p> <p>Added <a href="#">Section Table 5.: <math>V_{LCD}</math> rail decoupling</a></p> <p>Added VRAIL functions in <a href="#">Table 9: STM32L15xxC pin definitions</a></p> <p>Updated PH0-OSC_IN and PH1-OSC_OUT type in <a href="#">Table 9: STM32L15xxC pin definitions.</a></p> <p>Added <a href="#">Table 6.1.7: Optional LCD power supply scheme.</a></p> <p>Updated consumption data in <a href="#">Table 6.3.4: Supply current characteristics</a></p> <p>Updated <a href="#">Table 7: Pin loading conditions</a></p> <p>Updated <a href="#">Table 8: Pin input voltage</a> Updated <a href="#">Table 15: Typical application with a 32.768 kHz crystal</a></p> <p>Updated <a href="#">Table 25: Recommended NRST pin protection</a></p> <p><a href="#">Table 26: I<sup>2</sup>C bus AC waveforms and measurement circuit</a> Updated <a href="#">Table 35: Typical connection diagram using the ADC</a> and definition of symbol "RAIN" in <a href="#">Table 77: ADC characteristics</a></p> <p>Updated dThreshold/dt conditions in <a href="#">Table 85: Comparator 2 characteristics.</a></p> <p>Updated <a href="#">Table 49: Thermal resistance suffix 6.</a></p> <p>Added D2 and E2 in <a href="#">Table 69: UFQFPN48 – ultra thin fine pitch quad flat pack no-lead 7 x 7 mm, 0.5 mm pitch package mechanical data</a></p> <p>Fixed columns inversion in <a href="#">Table 67: LQFP64, 10 x 10 mm 64-pin low-profile quad flat package mechanical data</a> and <a href="#">Table 70: UFBGA100, 7 x 7 mm, 0.5 mm pitch package mechanical data</a></p>

## STM32L151xC STM32L152xC

## Revision History

**Table 75. Document revision history (continued)**

Date	Revision	Changes
12-Nov-2013	5	<p>Updated <a href="#">Section 3.15: Touch sensing</a>.</p> <p>Added <math>V_{DD} = 1.71</math> to 1.8 V operating power supply range in <a href="#">Table 4: Functionalities depending on the working mode (from Run/active down to standby)</a></p> <p>Renamed "I/O Level" to "I/O structure" in <a href="#">Table 9: STM32L15xxC pin definitions</a>, added the I/O structure for PC14, PC15, PC3, PH0, PH1, PA3, PA4, PA5, PB0, PE7, PE8, PE9, PE10, NRST and BOOT0</p> <p>Updated <a href="#">Table 10: Voltage characteristics</a> added row</p> <p>Updated <a href="#">Table 11: Current characteristics</a> replaced with the one inside STM32L15xxBxxA datasheet.</p> <p>Updated <a href="#">Table 13: General operating conditions</a>, footnote and added row.</p> <p>Updated <a href="#">Table 15: Embedded internal reference voltage calibration values</a> and moved inside <a href="#">Section 6.3.3: Embedded internal reference voltage</a></p> <p>Updated <a href="#">Section 6.3.4: Supply current characteristics</a>.</p> <p>Updated <a href="#">Table 19: Current consumption in Run mode, code with data processing running from Flash</a>.</p> <p>Updated <a href="#">Table 22: Current consumption in Run mode, code with data processing running from RAM</a>.</p> <p>Created <a href="#">Section 6.3.5: Wakeup time from low-power mode..</a></p> <p>Updated <a href="#">Table 38: High-speed external user clock characteristics</a>.</p> <p>Moved <a href="#">Figure 12: High-speed external clock source AC timing diagram</a> after <a href="#">Table 38: High-speed external user clock characteristics</a>.</p> <p>Updated <a href="#">Table 40: HSE oscillator characteristics</a>.</p> <p>Updated <a href="#">Section 6.3.12: Electrical sensitivity characteristics</a> (title).</p> <p>Updated <a href="#">Section 6.3.13: I/O current injection characteristics</a>.</p> <p>Updated <a href="#">Table 61: I/O current injection susceptibility</a> and added footnote.</p> <p>Updated <a href="#">Table 63: I/O static characteristics</a></p> <p>Updated <a href="#">Section 6.3.15: NRST pin characteristics</a>.</p> <p>Updated <a href="#">Table 77: ADC characteristics</a>.</p> <p>Added footnote<sup>(5)</sup> and <sup>(6)</sup> in <a href="#">Table 77: ADC characteristics</a></p> <p>Updated THD values and added 4 more rows ENOB, SINAD, SNR, THD in <a href="#">Table 78: ADC accuracy</a></p> <p>Updated "SDA data hold time" and "SDA and SCL rise time" values and added "Pulse width of spikes that are suppressed by the analog filter" row in <a href="#">Table 68: I<sup>2</sup>C characteristics</a></p> <p>Updated direct channels VDDA range in <a href="#">Table 79: R<sub>Ain</sub> max for f<sub>ADC</sub> = 16 MHz</a></p> <p>Moved <a href="#">Table 82: Temperature sensor calibration values</a> and moved inside <a href="#">Section 6.3.23: Temperature sensor characteristics</a></p> <p>Updated I<sub>DD</sub> (WU from Standby) unit in <a href="#">Table 31: Typical and maximum current consumptions in Standby mode</a>.</p> <p>Updated <a href="#">Table 67: LQFP64, 10 x 10 mm 64-pin low-profile quad flat package mechanical data</a></p> <p>Updated <a href="#">Chapter 8: Part numbering</a> (title).</p>

## Revision History

## STM32L151xC STM32L152xC

**Table 75. Document revision history (continued)**

Date	Revision	Changes
09-Dec-2013	6	<p>Apply footnote 1 also to VDD= 1.8 to 2.0 V in <a href="#">Table 2: Functionalities depending on the operating power supply range</a>.</p> <p>Updated <math>I_{inj}</math> pin in <a href="#">Table 11: Current characteristics</a>.</p> <p>Added Input Voltage in <a href="#">Table 13: General operating conditions</a>.</p> <p>Updated Input leakage current conditions in <a href="#">Table 63: I/O static characteristics</a></p> <p>Removed minimum value for <math>f_{SIN}</math> in <a href="#">Table 77: ADC characteristics</a>.</p> <p>Removed <math>F_{input}</math> for ENOB,SINAD,SNR,THD in <a href="#">Table 78: ADC accuracy</a>.</p> <p>Added tolerance for TS_CAL1 and TS_CAL2 in <a href="#">Table 82: Temperature sensor calibration values</a>.</p>
13-Mar-2014	7	<p>Updated <a href="#">Section 3.7: Memories</a>, <a href="#">Table 33: Peripheral current consumption</a> : updated Flash value, <a href="#">Table 61: I/O current injection susceptibility</a>, <a href="#">Table 63: I/O static characteristics</a>:added BOOT0 pin <a href="#">Table 66: NRST pin characteristics</a>, <a href="#">Chapter 2.2: Ultra-low-power device continuum</a>. removed figures “Power supply and reference decoupling (<math>V_{REF+}</math> not connected to <math>V_{DDA}</math>) and “Power supply and reference decoupling(<math>V_{REF+}</math> connected to <math>V_{DDA}</math>). Updated <a href="#">Table 19: Current consumption in Run mode, code with data processing running from Flash</a></p> <p>Updated <a href="#">Section 6.3.1: General operating conditions</a>.</p> <p>Updated <a href="#">Table 80: DAC characteristics</a></p> <p>Added marking for LQFP48/UFQFPN48 packages</p> <p>Updated <a href="#">Table 66: NRST pin characteristics</a></p> <p>Updated <a href="#">Table 63: I/O static characteristics</a></p>
16-May-2014	8	<p>Updated <math>I_{IO}</math> in <a href="#">Table 12: Current characteristics</a>.</p> <p>Updated conditions in <a href="#">Table 44: Output voltage characteristics</a>.</p> <p>Removed note 4 in <a href="#">Table 62: Temperature sensor characteristics</a></p> <p>Updated the conditions in <a href="#">Table 26: Low-power mode wakeup timings</a>.</p> <p>Removed ambiguity of “ambient temperature” in the electrical characteristics description.</p>
13-Oct-2014	9	<p>Updated <a href="#">Section 3.17: Communication interfaces</a> putting I2S characteristics inside.</p> <p>Updated DMIPS features in cover page and <a href="#">Section 2: Description</a>.</p> <p>Updated max temperature at 105°C instead of 85°C in the whole datasheet.</p> <p>Updated current consumption in <a href="#">Table 20: Current consumption in Sleep mode</a>.</p> <p>Updated <a href="#">Table 25: Peripheral current consumption</a> with new measured current values.</p> <p>Updated <a href="#">Table 58: Maximum source impedance RAIN max</a> adding note 2.</p>
06-Mar-2015	10	<p>Updated <a href="#">Section 7: Package information</a> with new package device marking.</p> <p>Updated <a href="#">Figure 9: Memory map</a>.</p>

## STM32L151xC STM32L152xC

## Revision History

**Table 75. Document revision history (continued)**

Date	Revision	Changes
20-Aug-2015	11	<p>Updated <a href="#">Table 17: Embedded internal reference voltage</a> temperature coefficient at 100ppm/°C and table footnote 3: “guaranteed by design” changed by “guaranteed by characterization results”.</p> <p>Updated <a href="#">Table 64: Comparator 2 characteristics</a> new maximum threshold voltage temperature coefficient at 100ppm/°C.</p>
10-Mar-2016	12	<p>Updated cover page putting eight SPIs in the peripheral communication interface list.</p> <p>Updated <a href="#">Table 2: Ultra-low-power STM32L151xC and STM32L152xC device features and peripheral counts</a> SPI and I2S lines.</p> <p>Updated <a href="#">Table 40: ESD absolute maximum ratings</a> CDM class.</p> <p>Updated all the notes, removing ‘not tested in production’.</p> <p>Updated thermal resistance for UFQFPN48 to value of 33 °C/W.</p> <p>Updated <a href="#">Table 11: Voltage characteristics</a> adding note about <math>V_{REF}</math>- pin.</p> <p>Updated <a href="#">Table 5: Functionalities depending on the working mode (from Run/active down to standby)</a> LSI and LSE functionalities putting “Y” in Standby mode.</p> <p>Removed note 1 below <a href="#">Figure 2: Clock tree</a>.</p>
25-Aug-2017	13	<p>Updated <a href="#">Table 43: I/O static characteristics</a> pull-up and pull-down values.</p> <p>Updated <a href="#">Table 46: NRST pin characteristics</a> pull-up values.</p> <p>Updated <a href="#">Section 7: Package information</a> adding information about other optional marking or inset/upset marks.</p> <p>Updated note 1 below all the package device marking figures.</p> <p>Updated <a href="#">Section 7: Package information</a> replacing “Marking of engineering samples” by “device marking”.</p> <p>Updated <a href="#">Nested vectored interrupt controller (NVIC)</a> in <a href="#">Section 3.2: ARM® Cortex®-M3 core with MPU</a> about process state automatically saved.</p> <p>Updated <a href="#">Table 3: Functionalities depending on the operating power supply range</a> removing I/O operation column and adding note about GPIO speed.</p> <p>Updated <a href="#">Table 42: I/O current injection susceptibility</a> note by ‘injection is not possible’.</p> <p>Updated <a href="#">Figure 20: Recommended NRST pin protection</a> note about the 0.1uF capacitor.</p> <p>Updated <a href="#">Table 59: DAC characteristics</a> resistive load.</p> <p>Updated <a href="#">Section 3.1: Low-power modes</a> Low-power run mode (MSI) RC oscillator clock.</p> <p>Updated <a href="#">Table 5: Functionalities depending on the working mode (from Run/active down to standby)</a> disabling I2C functionality in Low-power Run and Low-power Sleep modes.</p>

**STM32L151xC STM32L152xC****IMPORTANT NOTICE – PLEASE READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved

## Annexe C – Extraits du "Reference Manual" du STM32L152RCT6

Aussi surprenant que cela paraisse, la documentation de presque 150 pages de l'annexe B est ... un résumé ! La documentation complète est un document appelé "document de référence" qui fait lui plus de 900 pages. Il n'a donc pas été ajouté en intégralité à ce document, mais il est disponible [ici](#) ([PDF RM0038](#)).

On a compilé dans cette annexe les extraits qui nous seront les plus utiles pour ce cours.

System architecture .....	249
Register Boundary Addresses .....	253
AHB peripheral clock enable register (RCC_AHBENR) .....	255
RCC register map .....	257
General Purpose I/Os (GPIO) .....	260
GPIO register map .....	278

## 2 System architecture and memory overview

### 2.1 System architecture

The main system consists of a 32-bit multilayer AHB bus matrix that interconnects:

- Up to five masters:
  - Cortex®-M3 I-bus, D-bus and S-bus
  - DMA1 and DMA2
- Up to five slaves:
  - Internal Flash memory ICode
  - Internal Flash memory DCode
  - Internal SRAM
  - AHB to APBx (APB1 or APB2), which connect all the APB peripherals
  - Flexible Static Memory Controller

These are interconnected using the multilayer AHB bus architecture shown in [Figure 1](#):

**Figure 1. System architecture (Cat.1 and Cat.2 devices)**

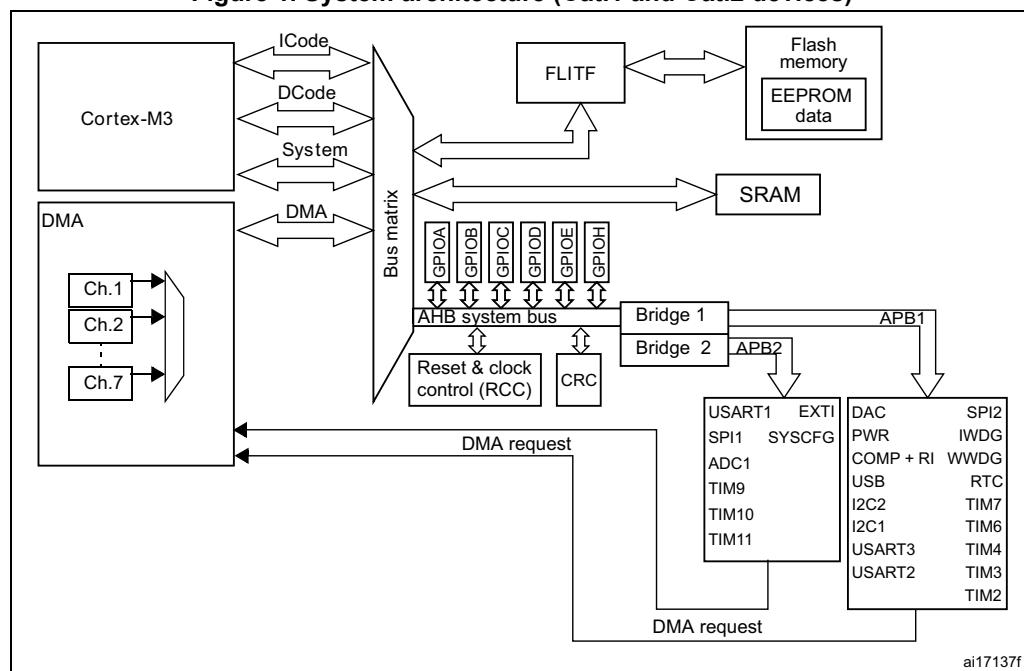
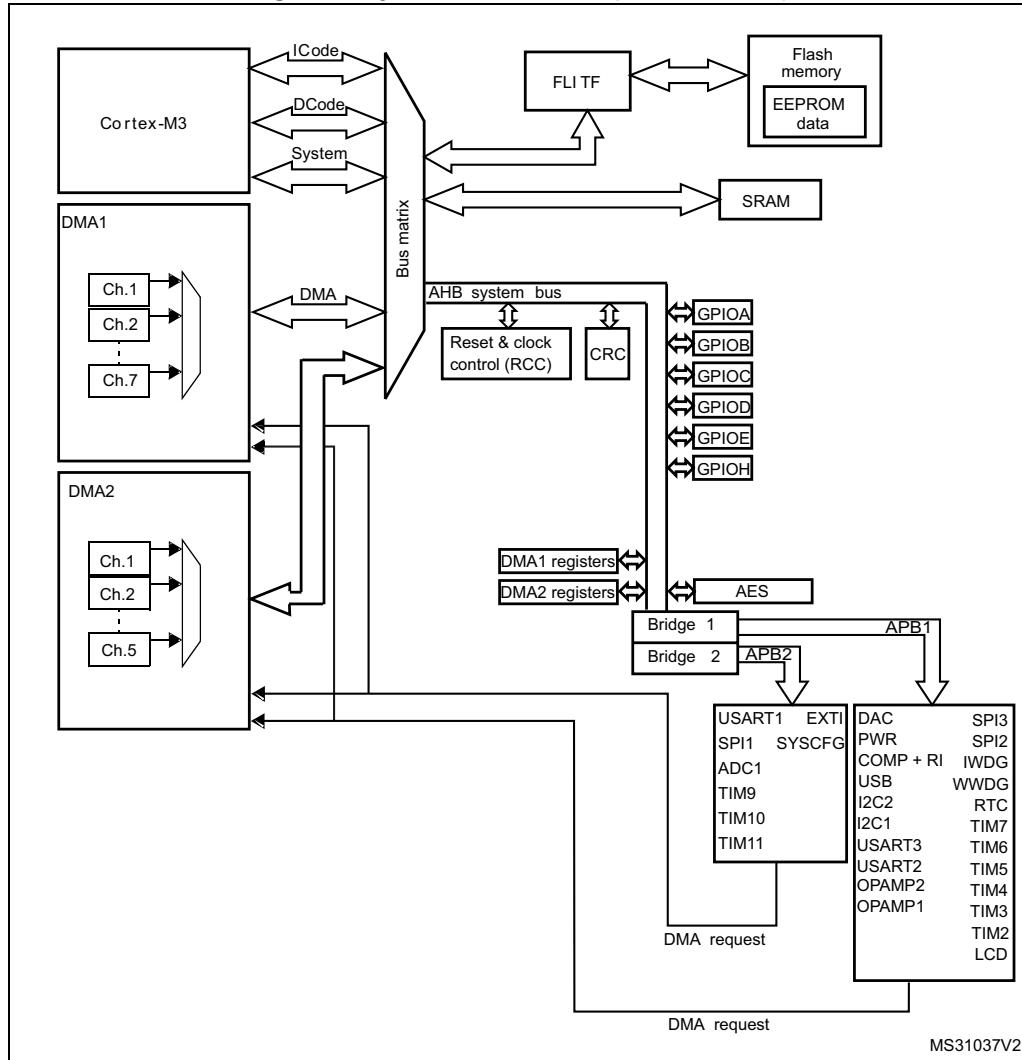


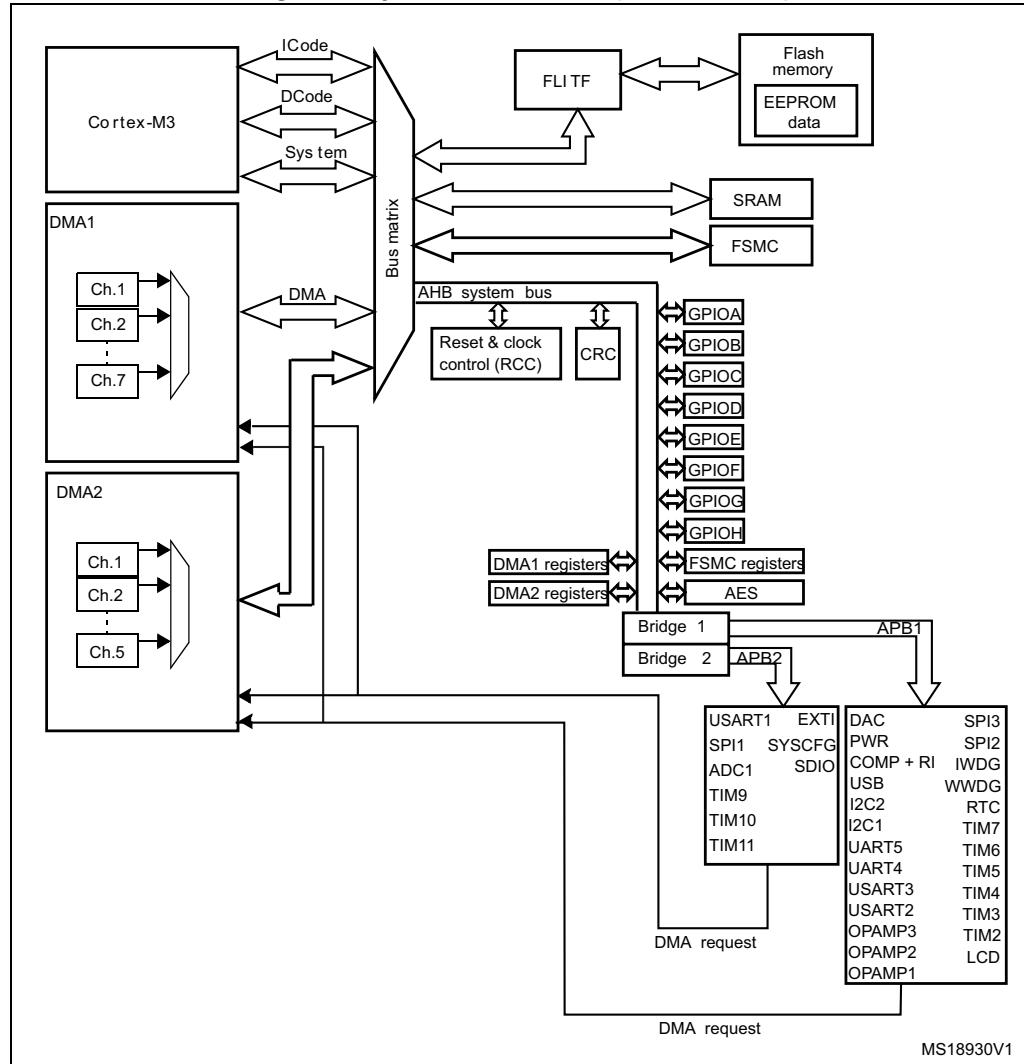
Figure 2. System architecture (Cat.3 devices)



## System architecture and memory overview

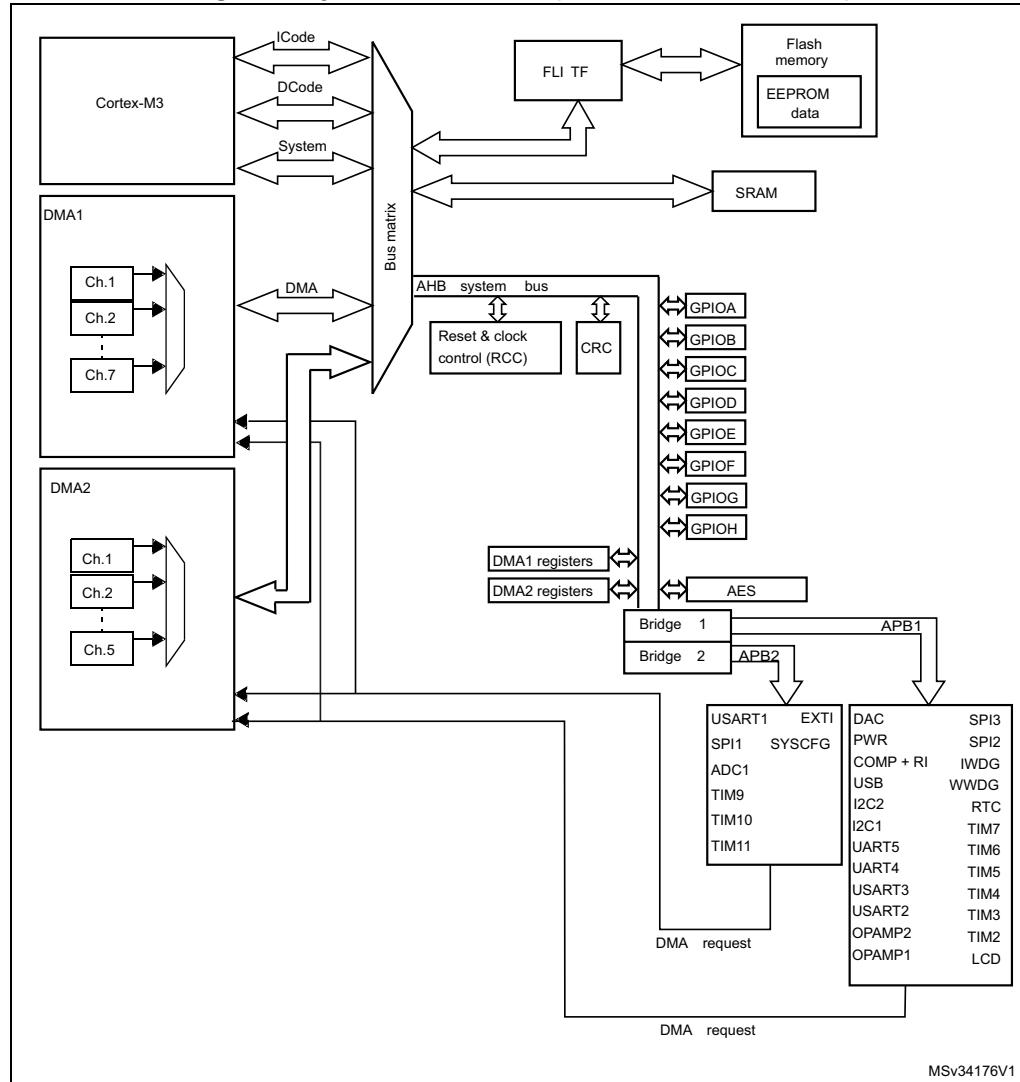
RM0038

Figure 3. System architecture (Cat.4 devices)



MS18930V1

Figure 4. System architecture (Cat.5 and Cat.6 devices)



### ICode bus

This bus connects the Instruction bus of the Cortex<sup>®</sup>-M3 core to the BusMatrix. This bus is used by the core to fetch instructions. The target of this bus is a memory containing code (internal Flash memory or SRAM).

### DCode bus

This bus connects the databus of the Cortex<sup>®</sup>-M3 to the BusMatrix. This bus is used by the core for literal load and debug access. The target of this bus is a memory containing code or data (internal Flash memory or SRAM).

### System bus

This bus connects the system bus of the Cortex<sup>®</sup>-M3 core to a BusMatrix. This bus is used to access data located in a peripheral or in SRAM. Instructions may also be fetched on this bus (less efficient than ICode). The targets of this bus are the internal SRAM and the AHB/APB bridges.

---

**System architecture and memory overview****RM0038**

---

**DMA bus**

This bus connects the AHB master interface of the DMA to the bus matrix which manages the access of the CPU DCode and DMA to the SRAM, Flash memory and peripherals.

**Bus matrix**

The bus matrix manages the access arbitration between the core system bus and the DMA master bus. The arbitration uses a round robin algorithm. The bus matrix is composed of five masters (ICode, DCode, System bus, DMA1 bus, DMA2 bus) and five slaves (Flash ICode interface, Flash DCode interface, SRAM, FSMC, and AHB2APB bridges).

AHB peripherals are connected on the system bus through the bus matrix to allow DMA access.

**AHB/APB bridges (APB)**

The two AHB/APB bridges provide full synchronous connections between the AHB and the 2 APB buses. The two APB buses operates at full speed (up to 32 MHz).

Refer to [Table 5 on page 47](#) for the address mapping of the AHB and APB peripherals.

After each device reset, all peripheral clocks are disabled (except for the SRAM and Flash interface). Before using a peripheral, its clock should be enabled in the RCC\_AHBENR, RCC\_APB1ENR or RCC\_APB2ENR register.

*Note:* When a 16- or 8-bit access is performed on an APB register, the access is transformed into a 32-bit access: the bridge duplicates the 16- or 8-bit data to feed the 32-bit vector.

## 2.2 Memory organization

SRAM, NVM, registers and I/O ports are organized within the same linear 4 Gbyte address space.

The bytes are coded in memory in little endian format. The lowest numbered byte in a word is considered the word's least significant byte and the highest numbered byte, the most significant.

For the detailed mapping of peripheral registers, refer to the related sections.

The addressable memory space is divided into 8 main blocks, each of 512 Mbytes.

All the memory areas that are not allocated to on-chip memories and peripherals are considered "Reserved". Refer to the memory map figure in the STM32L1xxxx datasheet.

## 2.3 Memory map

See the STM32L1xxxx datasheet for a comprehensive diagram of the memory map. [Table 5](#) gives the boundary addresses of the peripherals available in STM32L1xxxx devices.

**Table 5. Register boundary addresses**

Boundary address	Peripheral	Bus	Register map	
0xA000 0000 - 0xA000 0FFF	FSMC	AHB	<a href="#">Section 25.5.7: FSMC register map on page 658</a>	
0x5006 0000 - 0x5006 03FF	AES		<a href="#">Section 23.12.13: AES register map on page 585</a>	
0x4002 6400 - 0x4002 67FF	DMA2		<a href="#">Section 11.4.7: DMA register map on page 262</a>	
0x4002 6000 - 0x4002 63FF	DMA1		<a href="#">Section 11.4.7: DMA register map on page 262</a>	
0x4002 3C00 - 0x4002 3FFF	FLASH		<a href="#">Section 3.9.10: Register map on page 91</a>	
0x4002 3800 - 0x4002 3BFF	RCC		<a href="#">Section 6.3.15: RCC register map on page 168</a>	
0x4002 3000 - 0x4002 33FF	CRC		<a href="#">Section 4.4.4: CRC register map on page 96</a>	
0x4002 1C00 - 0x4002 1FFF	GPIOG		<a href="#">Section 7.4.12: GPIO register map on page 189</a>	
0x4002 1800 - 0x4002 1BFF	GPIOF			
0x4002 1400 - 0x4002 17FF	GPIOH			
0x4002 1000 - 0x4002 13FF	GPIOE			
0x4002 0C00 - 0x4002 0FFF	GPIOD			
0x4002 0800 - 0x4002 0BFF	GPIOC			
0x4002 0400 - 0x4002 07FF	GPIOB			
0x4002 0000 - 0x4002 03FF	GPIOA			
0x4001 3800 - 0x4001 3BFF	USART1	APB2	<a href="#">Section 27.6.8: USART register map on page 743</a>	
0x4001 3000 - 0x4001 33FF	SPI1		<a href="#">Section 28.5.10: SPI register map on page 796</a>	
0x4001 2C00 - 0x4001 2FFF	SDIO		<a href="#">Section 29.9.16: SDIO register map on page 852</a>	
0x4001 2400 - 0x4001 27FF	ADC		<a href="#">Section 12.15.21: ADC register map on page 308</a>	
0x4001 1000 - 0x4001 13FF	TIM11		<a href="#">Section 14.4.17: TIMx register map on page 368</a>	
0x4001 0C00 - 0x4001 0FFF	TIM10		<a href="#">Section 14.4.17: TIMx register map on page 368</a>	
0x4001 0800 - 0x4001 0BFF	TIM9		<a href="#">Section 14.4.17: TIMx register map on page 368</a>	
0x4001 0400 - 0x4001 07FF	EXTI		<a href="#">Section 10.3.7: EXTI register map on page 244</a>	
0x4001 0000 - 0x4001 03FF	SYSCFG		<a href="#">Section 8.5.7: SYSCFG register map on page 222</a>	

**System architecture and memory overview****RM0038****Table 5. Register boundary addresses (continued)**

<b>Boundary address</b>	<b>Peripheral</b>	<b>Bus</b>	<b>Register map</b>
0x4000 7C00 - 0x4000 7C03	COMP	APB1	<a href="#">Section 14.9.2: COMP register map on page 343</a>
0x4000 7C04 - 0x4000 7C5B	RI		<a href="#">Section 8.5.7: SYSCFG register map on page 222</a>
0x4000 7C5C - 0x4000 7FFF	OPAMP		<a href="#">Section 15.4.4: OPAMP register map on page 353</a>
0x4000 7400 - 0x4000 77FF	DAC		<a href="#">Section 13.5.15: DAC register map on page 331</a>
0x4000 7000 - 0x4000 73FF	PWR		<a href="#">Section 5.4.3: PWR register map on page 124</a>
0x4000 6000 - 0x4000 63FF	USB device FS SRAM 512 bytes		<a href="#">Section 24.5.4: USB register map on page 616</a>
0x4000 5C00 - 0x4000 5FFF	USB device FS		<a href="#">Section 26.6.10: I2C register map on page 691</a>
0x4000 5800 - 0x4000 5BFF	I2C2		<a href="#">Section 27.6.8: USART register map on page 743</a>
0x4000 5400 - 0x4000 57FF	I2C1		<a href="#">Section 28.5.10: SPI register map on page 796</a>
0x4000 5000 - 0x4000 53FF	USART5		<a href="#">Section 21.4.5: IWDG register map on page 553</a>
0x4000 4C00 - 0x4000 4FFF	USART4		<a href="#">Section 22.6.4: WWDG register map on page 560</a>
0x4000 4800 - 0x4000 4BFF	USART3		<a href="#">Section 20.6.21: RTC register map on page 546</a>
0x4000 4400 - 0x4000 47FF	USART2		<a href="#">Section 16.5.6: LCD register map on page 379</a>
0x4000 3C00 - 0x4000 3FFF	SPI3		<a href="#">Section 19.4.9: TIM6 and TIM7 register map on page 506</a>
0x4000 3800 - 0x4000 3BFF	SPI2		<a href="#">Section 17.4.21: TIMx register map on page 440</a>
0x4000 3000 - 0x4000 33FF	IWDG		
0x4000 2C00 - 0x4000 2FFF	WWDG		
0x4000 2800 - 0x4000 2BFF	RTC		
0x4000 2400 - 0x4000 27FF	LCD		
0x4000 1400 - 0x4000 17FF	TIM7		
0x4000 1000 - 0x4000 13FF	TIM6		
0x4000 0C00 - 0x4000 0FFF	TIM5 (32-bits)		
0x4000 0800 - 0x4000 0BFF	TIM4		
0x4000 0400 - 0x4000 07FF	TIM3		
0x4000 0000 - 0x4000 03FF	TIM2		

### 6.3.8 AHB peripheral clock enable register (RCC\_AHBENR)

Address offset: 0x1C

Reset value: 0x0000 8000

Access: no wait state, word, half-word and byte access

**Note:** When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.		FSMC EN	Reserved	AES EN	Res.	DMA2E N	DMA1EN									
	rw			rw		rw	rw									
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FLITF EN	Reserved	CRCEN	Reserved			GPIOG EN	GPIOF EN	GPIOH EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN			
rw		rw				rw										

Bit 31 Reserved, must be kept at reset value.

Bit 30 **FSMCEN:** FSMC clock enable

This bit is set and cleared by software.

0: FSMC clock disabled

1: FSMC clock enabled

*Note: This bit is available in Cat.4 devices only.*

Bits 29:28 Reserved, must be kept at reset value.

Bit 27 **AESEN:** AES clock enable

This bit is set and cleared by software.

0: AES clock disabled

1: AES clock enabled

*Note: This bit is available in STM32L16x devices only.*

Bit 26 Reserved, must be kept at reset value.

Bit 25 **DMA2EN:** DMA2 clock enable

This bit is set and cleared by software.

0: DMA2 clock disabled

1: DMA2 clock enabled

*Note: This bit is available in Cat.3, Cat.4, Cat.5 and Cat.6 devices only.*

Bit 24 **DMA1EN:** DMA1 clock enable

This bit is set and cleared by software.

0: DMA1 clock disabled

1: DMA1 clock enabled

Bits 23:16 Reserved, must be kept at reset value.

Bit 15 **FLITFEN:** FLITF clock enable

This bit can be written only when the Flash memory is in power down mode.

0: FLITF clock disabled

1: FLITF clock enabled

Bits 14:13 Reserved, must be kept at reset value.

**Reset and clock control (RCC)**

RM0038

**Bit 12 CRCEN: CRC clock enable**

This bit is set and cleared by software.

- 0: CRC clock disabled
- 1: CRC clock enabled

Bits 11:6 Reserved, must be kept at reset value.

**Bit 7 GPIOGEN: IO port G clock enable**

This bit is set and cleared by software.

- 0: IO port G clock disabled
- 1: IO port G clock enabled

*Note: This bit is available in Cat.4, Cat.5 and Cat.6 devices only.***Bit 6 GPIOFEN: IO port F clock enable**

This bit is set and cleared by software.

- 0: IO port F clock disabled
- 1: IO port F clock enabled

*Note: This bit is available in Cat.4, Cat.5 and Cat.6 devices only.***Bit 5 GPIOHEN: IO port H clock enable**

This bit is set and cleared by software.

- 0: IO port H clock disabled
- 1: IO port H clock enabled

**Bit 4 GPIOEEN: IO port E clock enable**

This bit is set and cleared by software.

- 0: IO port E clock disabled
- 1: IO port E clock enabled

**Bit 3 GPIODEN: IO port D clock enable**

Set and cleared by software.

- 0: IO port D clock disabled
- 1: IO port D clock enabled

**Bit 2 GPIOCEN: IO port C clock enable**

This bit is set and cleared by software.

- 0: IO port C clock disabled
- 1: IO port C clock enabled

**Bit 1 GPIOBEN: IO port B clock enable**

This bit is set and cleared by software.

- 0: IO port B clock disabled
- 1: IO port B clock enabled

**Bit 0 GPIOAEN: IO port A clock enable**

This bit is set and cleared by software.

- 0: IO port A clock disabled
- 1: IO port A clock enabled

**Reset and clock control (RCC)**

RM0038

**Bit 9 LSERDY: External low-speed oscillator ready**

This bit is set and cleared by hardware to indicate when the LSE oscillator is stable. After the LSEON bit is cleared, LSERDY goes low after 6 LSE oscillator clock cycles.

It is reset by setting the RTCRST bit or by a POR.

0: External 32 kHz oscillator not ready

1: External 32 kHz oscillator ready

**Bit 8 LSEON: External low-speed oscillator enable**

This bit is set and cleared by software.

It is reset by setting the RTCRST bit or by a POR.

0: LSE oscillator OFF

1:LSE oscillator ON

Bits 7:2 Reserved, must be kept at reset value.

**Bit 1 LSIRDY: Internal low-speed oscillator ready**

This bit is set and cleared by hardware to indicate when the LSI oscillator is stable. After the LSION bit is cleared, LSIRDY goes low after 3 LSI oscillator clock cycles. This bit is kept set if IWDG is activated.

This bit is reset by system reset.

0: LSI oscillator not ready

1: LSI oscillator ready

**Bit 0 LSION: Internal low-speed oscillator enable**

This bit is set and cleared by software.

It is reset by system reset.

0: LSI oscillator OFF

1: LSI oscillator ON

**6.3.15 RCC register map**

The following table gives the RCC register map and the reset values. The reserved memory areas are highlighted in gray in the table.

**Table 36. RCC register map and reset values**

Off-set	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	RCC_CR	Reserved	RTCPRE1	RTCPRE0	CSSON	Reserved	PLL RDY	PLL ON	Reserved				HSEBYP	HSERDY	Reserved				MISRDY	Reserved				HSIRDY	Reserved				HSION	Reserved			
		0	0	0	0	0	0	0	Reserved				x	0	0	1	1	1	1	0	0	0	0	x	0	0	0	0	0	0	0		
0x04	RCC_ICSCR	MSITRIM[7:0]						MSICAL[7:0]						MSIRAN GE[2:0]		HSITRIM[4:0]				HSICAL[7:0]							Reserved						
		0	0	0	0	0	0	0	x	x	x	x	x	x	x	1	0	1	1	1	0	0	0	0	x	x	x	x	x	x	x	x	
0x08	RCC_CFGR	Reserved	MCOPRE [2:0]		Reserved	MCOSEL [2:0]		PLL DIV [1:0]	PLL MUL[3:0 ]			Reserved	PLLSRC	Reserved	PPRE2 [2:0]		PPRE1 [2:0]		HPRE[3:0]			SWS [1:0]	SW [1:0]		Reserved				Reserved				
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 36. RCC register map and reset values (continued)

Offset	Register		
0x0C	RCC_CIR		
		Reset value	Reserved
0x10	RCC_AHBRSTR	0	FSMCRST
		Reset value	Reserved
0x14	RCC_APB2RSTR	AESRST	AESRST
0x18	RCC_APB1RSTR	DMA2RST	DMA2RST
0x1C	RCC_AHBNR	DMA1RST	DMA1RST
0x20	RCC_APB2ENR	USBEN	USBEN
0x24	RCC_APB1ENR	I2C2EN	I2C2EN
0x28	RCC_AHBLENR	I2C1EN	I2C1EN
0x2C	RCC_AHBLENR	USART5EN	USART5EN
0x30	RCC_APB2ENR	USART4EN	USART4EN
0x34	RCC_APB1ENR	USART3EN	USART3EN
0x38	RCC_APB1ENR	USART2EN	USART2EN
0x3C	RCC_APB1ENR	FLITFEN	FLITFEN
0x40	RCC_APB1ENR	SPI3EN	SPI3EN
0x44	RCC_APB1ENR	SPI2EN	SPI2EN
0x48	RCC_APB1ENR	USART1EN	USART1EN
0x4C	RCC_APB1ENR	SPIEN	SPIEN
0x50	RCC_APB1ENR	SDIOEN	SDIOEN
0x54	RCC_APB1ENR	ADC1EN	ADC1EN
0x58	RCC_APB1ENR	CRCLEN	CRCLEN
0x5C	RCC_APB1ENR	WWDRST	WWDRST
0x60	RCC_APB1ENR	SPIORST	SPIORST
0x64	RCC_APB1ENR	SDIORST	SDIORST
0x68	RCC_APB1ENR	ADC1RST	ADC1RST
0x70	RCC_APB1ENR	Res.	Res.
0x74	RCC_APB1ENR	GPIOGEN	GPIOGEN
0x78	RCC_APB1ENR	GPIOHEN	GPIOHEN
0x7C	RCC_APB1ENR	TIM7RST	TIM7RST
0x80	RCC_APB1ENR	GPIOPEEN	GPIOPEEN
0x84	RCC_APB1ENR	TIM6RST	TIM6RST
0x88	RCC_APB1ENR	TIM5RST	TIM5RST
0x8C	RCC_APB1ENR	TIM4RST	TIM4RST
0x90	RCC_APB1ENR	TIM3RST	TIM3RST
0x94	RCC_APB1ENR	TIM2RST	TIM2RST
0x98	RCC_APB1ENR	SYSCFGEN	SYSCFGEN
0xA0	RCC_APB1ENR	GPIOAEN	GPIOAEN
0xA4	RCC_APB1ENR	TIM2EN	TIM2EN
0xA8	RCC_APB1ENR	SYSCFG	SYSCFG
0xB0	RCC_APB1ENR	LSIRDYF	LSIRDYF
0xB4	RCC_APB1ENR	HSIRDYF	HSIRDYF
0xB8	RCC_APB1ENR	LSERDYF	LSERDYF
0xC0	RCC_APB1ENR	HSERDYF	HSERDYF
0xC4	RCC_APB1ENR	PLLRDYF	PLLRDYF
0xC8	RCC_APB1ENR	CSSF	CSSF
0xD0	RCC_APB1ENR	GPIORST	GPIORST
0xD4	RCC_APB1ENR	GPIOFRST	GPIOFRST
0xD8	RCC_APB1ENR	GPIOHFRST	GPIOHFRST
0xE0	RCC_APB1ENR	MSIRDYF	MSIRDYF

## Reset and clock control (RCC)

RM0038

Table 36. RCC register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
0x2C	RCC_APB2LPEN																																								
		Reset value																																							
0x30	RCC_APB1LPEN	1	COM1LPEN																																						
		Reset value																																							
0x034	RCC_CSR	1	LPWRSTF	WWDG_RSTF	IWDG_RSTF	SFRSTF	PORRSTF	PNRSTF	OBLRSTF	RMVF	RTC_RST	RTCEN	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
		Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Refer to [Table 5 on page 47](#) for the register boundary addresses.

## 7 General-purpose I/Os (GPIO)

This section applies to the whole STM32L1xxxx family, unless otherwise specified.

### 7.1 GPIO introduction

Each general-purpose I/O port has four 32-bit configuration registers (GPIOx\_MODER, GPIOx\_OTYPER, GPIOx\_OSPEEDR and GPIOx\_PUPDR), two 32-bit data registers (GPIOx\_IDR and GPIOx\_ODR), a 32-bit set/reset register (GPIOx\_BSRR), a 32-bit locking register (GPIOx\_LCKR) and two 32-bit alternate function selection register (GPIOx\_AFRH and GPIOx\_AFRL).

### 7.2 GPIO main features

- Up to 16 I/Os under control
- Output states: push-pull or open drain + pull-up/down
- Output data from output data register (GPIOx\_ODR) or peripheral (alternate function output)
- Speed selection for each I/O
- Input states: floating, pull-up/down, analog
- Input data to input data register (GPIOx\_IDR) or peripheral (alternate function input)
- Bit set and reset register (GPIOx\_BSRR) for bitwise write access to GPIOx\_ODR
- Locking mechanism (GPIOx\_LCKR) provided to freeze the I/O configuration
- Analog function
- Alternate function input/output selection registers (at most 16 AFs per I/O)
- Fast toggle capable of changing every two clock cycles
- Highly flexible pin multiplexing allows the use of I/O pins as GPIOs or as one of several peripheral functions

### 7.3 GPIO functional description

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

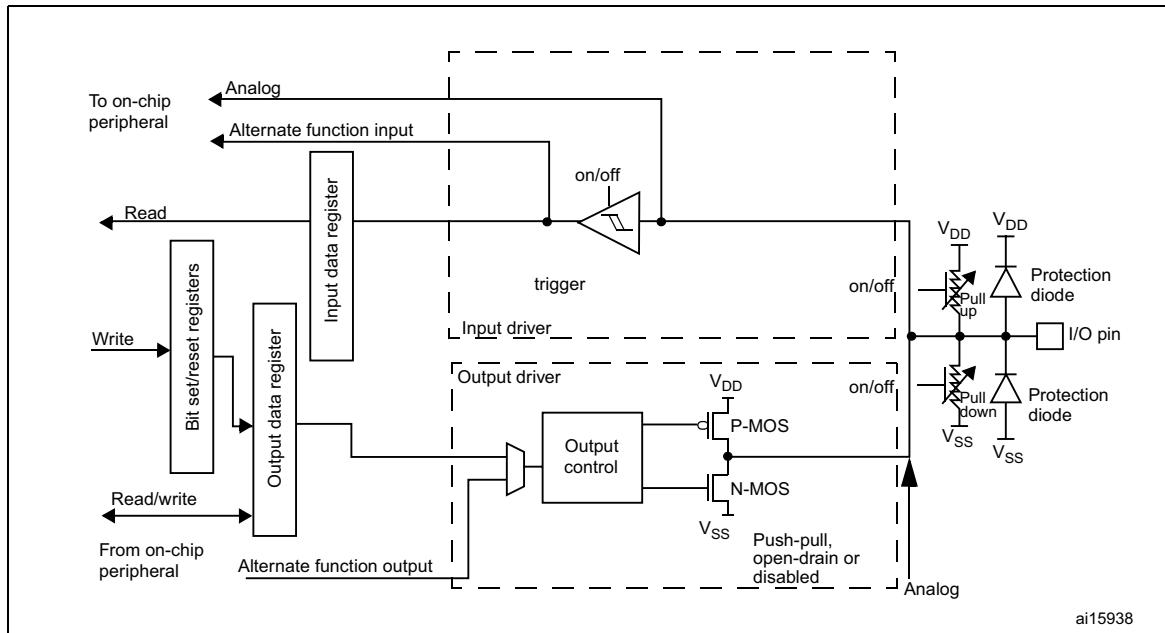
- Input floating
- Input pull-up
- Input-pull-down
- Analog
- Output open-drain with pull-up or pull-down capability
- Output push-pull with pull-up or pull-down capability
- Alternate function push-pull with pull-up or pull-down capability
- Alternate function open-drain with pull-up or pull-down capability

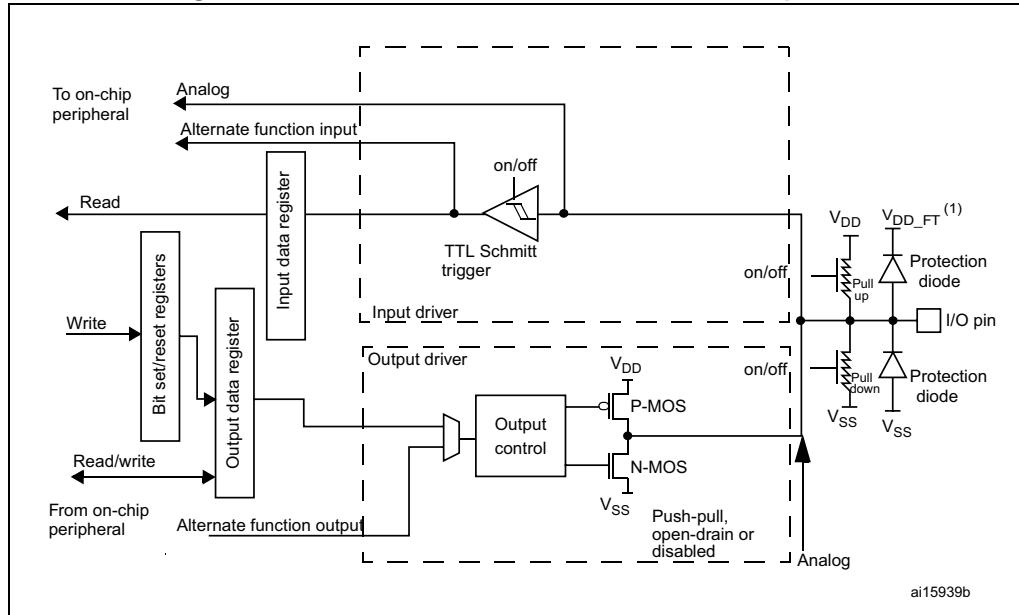
**General-purpose I/Os (GPIO)**

RM0038

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words, half-words or bytes. The purpose of the GPIOx\_BSRR register is to allow atomic read/modify accesses to any of the GPIO registers. In this way, there is no risk of an IRQ occurring between the read and the modify access.

*Figure 18* and *Figure 19* show the basic structures of a standard and a 5 V tolerant I/O port bit, respectively. *Table 40* gives the possible port bit configurations.

**Figure 18. Basic structure of a standard I/O port bit**

**Figure 19. Basic structure of a five-volt tolerant I/O port bit**

1.  $V_{DD\_FT}$  is a potential specific to five-volt tolerant I/Os and different from  $V_{DD}$ .

**Table 37. Port bit configuration table<sup>(1)</sup>**

MODER(i) [1:0]	OTYPER(i)	OSPEEDR(i) [B:A]	PUPDR(i) [1:0]	I/O configuration	
01	0	SPEED [B:A]	0	0	GP output
	0		0	1	GP output
	0		1	0	GP output
	0		1	1	Reserved
	1		0	0	GP output
	1		0	1	GP output
	1		1	0	GP output
	1		1	1	Reserved (GP output OD)
10	0	SPEED [B:A]	0	0	AF
	0		0	1	AF
	0		1	0	AF
	0		1	1	Reserved
	1		0	0	AF
	1		0	1	AF
	1		1	0	AF
	1		1	1	Reserved

**General-purpose I/Os (GPIO)**

RM0038

**Table 37. Port bit configuration table<sup>(1)</sup> (continued)**

<b>MODER(i) [1:0]</b>	<b>OTYPER(i)</b>	<b>OSPEEDR(i) [B:A]</b>		<b>PUPDR(i) [1:0]</b>		<b>I/O configuration</b>	
00	x	x	x	0	0	Input	Floating
	x	x	x	0	1	Input	PU
	x	x	x	1	0	Input	PD
	x	x	x	1	1	Reserved (input floating)	
11	x	x	x	0	0	Input/output	Analog
	x	x	x	0	1	Reserved	
	x	x	x	1	0		
	x	x	x	1	1		

1. GP = general-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

**7.3.1 General-purpose I/O (GPIO)**

During and just after reset, the alternate functions are not active and the I/O ports are configured in input floating mode.

The debug pins are in AF pull-up/pull-down after reset:

- PA15: JTDI in pull-up
- PA14: JTCK/SWCLK in pull-down
- PA13: JTMS/SWDAT in pull-up
- PB4: NJTRST in pull-up
- PB3: JTDO in floating state

When the pin is configured as output, the value written to the output data register (GPIOx\_ODR) is output on the I/O pin. It is possible to use the output driver in push-pull mode or open-drain mode (only the N-MOS is activated when 0 is output).

The input data register (GPIOx\_IDR) captures the data present on the I/O pin at every AHB clock cycle.

All GPIO pins have weak internal pull-up and pull-down resistors, which can be activated or not depending on the value in the GPIOx\_PUPDR register.

**7.3.2 I/O pin multiplexer and mapping**

The microcontroller I/O pins are connected to onboard peripherals/modules through a multiplexer that allows only one peripheral's alternate function (AF) connected to an I/O pin at a time. In this way, there can be no conflict between peripherals sharing the same I/O pin.

Each I/O pin has a multiplexer with sixteen alternate function inputs (AF0 to AF15) that can be configured through the GPIOx\_AFRL (for pin 0 to 7) and GPIOx\_AFRH (for pin 8 to 15) registers:

- After reset all I/Os are connected to the system's alternate function 0 (AF0)
- The peripherals' alternate functions are mapped from AF1 to AF14
- Cortex®-M3 EVENTOUT is mapped on AF15

This structure is shown in [Figure 20](#) below.

In addition to this flexible I/O multiplexing architecture, each peripheral has alternate functions mapped onto different I/O pins to optimize the number of peripherals available in smaller packages.

To use an I/O in a given configuration, proceed as follows:

- **System function**

Connect the I/O to AF0 and configure it depending on the function used:

- JTAG/SWD, after each device reset these pins are assigned as dedicated pins immediately usable by the debugger host (not controlled by the GPIO controller)
- RTC\_AF1: refer to [Table 39: RTC\\_AF1 pin](#) for more details about this pin configuration
- RTC\_50Hz: this pin should be configured in Input floating mode
- MCO: this pin has to be configured in alternate function mode.

**Note:** *The user can disable some or all of the JTAG/SWD pins and so release the associated pins for GPIO usage (released pins highlighted in gray in the table).*

For more details refer to [Section 6.2.13: Clock-out capability](#).

**Table 38. Flexible SWJ-DP pin assignment**

Available debug ports	SWJ I/O pin assigned				
	PA13 / JTMS/ SWDIO	PA14 / JTCK/ SWCLK	PA15 / JTDI	PB3 / JTDO	PB4/ NJTRST
Full SWJ (JTAG-DP + SW-DP) - Reset state	X	X	X	X	X
Full SWJ (JTAG-DP + SW-DP) but without NJTRST	X	X	X	X	
JTAG-DP Disabled and SW-DP Enabled	X	X			
JTAG-DP Disabled and SW-DP Disabled					Released

- **GPIO**

Configure the desired I/O as output, input or analog in the GPIOx\_MODER register.

- **Peripheral alternate function**

For the ADC and DAC, configure the desired I/O as analog in the GPIOx\_MODER register.

For other peripherals:

- Configure the desired I/O as an alternate function in the GPIOx\_MODER register
- Select the type, pull-up/pull-down and output speed via the GPIOx\_OTYPER, GPIOx\_PUPDR and GPIOx\_OSPEEDR registers, respectively
- Connect the I/O to the desired AFx in the GPIOx\_AFRL or GPIOx\_AFRH register

- **EVENTOUT**

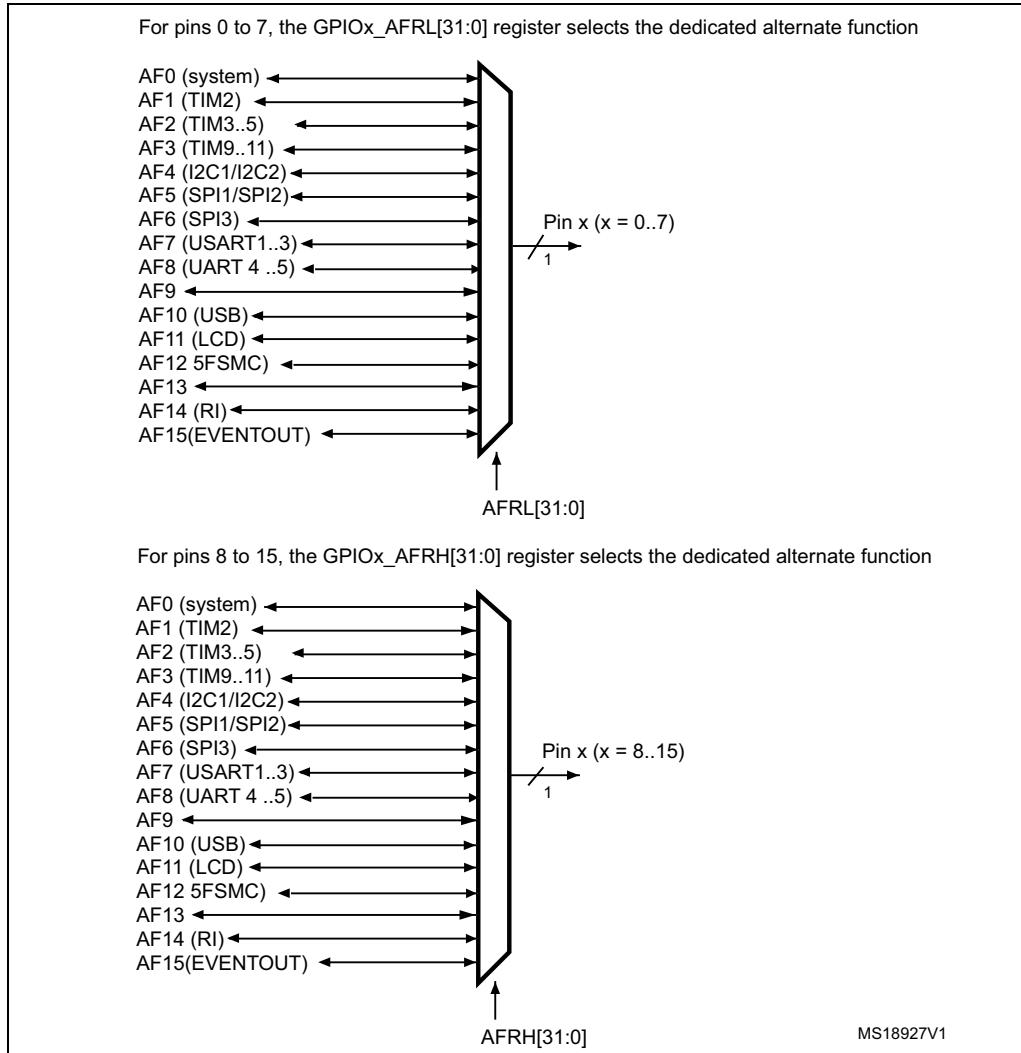
Configure the I/O pin used to output the Cortex®-M3 EVENTOUT signal by connecting it to AF15

**Note:** *EVENTOUT is not mapped onto the following I/O pins: PH0, PH1 and PH2.*

**General-purpose I/Os (GPIO)**

RM0038

Refer to the “Alternate function mapping” table in the datasheets for the detailed mapping of the system and peripherals’ alternate function I/O pins.

**Figure 20. Selecting an alternate function**

### 7.3.3 I/O port control registers

Each of the GPIOs has four 32-bit memory-mapped control registers (GPIO<sub>x</sub>\_MODER, GPIO<sub>x</sub>\_OTYPER, GPIO<sub>x</sub>\_OSPEEDR, GPIO<sub>x</sub>\_PUPDR) to configure up to 16 I/Os.

The GPIO<sub>x</sub>\_MODER register is used to select the I/O direction (input, output, AF, analog). The GPIO<sub>x</sub>\_OTYPER and GPIO<sub>x</sub>\_OSPEEDR registers are used to select the output type (push-pull or open-drain) and speed (the I/O speed pins are directly connected to the corresponding GPIO<sub>x</sub>\_OSPEEDR register bits whatever the I/O direction). The GPIO<sub>x</sub>\_PUPDR register is used to select the pull-up/pull-down whatever the I/O direction.

### 7.3.4 I/O port data registers

Each GPIO has two 16-bit memory-mapped data registers: input and output data registers (GPIO<sub>x</sub>\_IDR and GPIO<sub>x</sub>\_ODR). GPIO<sub>x</sub>\_ODR stores the data to be output, it is read/write accessible. The data input through the I/O are stored into the input data register (GPIO<sub>x</sub>\_IDR), a read-only register.

See [Section 7.4.5: GPIO port input data register \(GPIO<sub>x</sub>\\_IDR\) \(x = A..H\)](#) and [Section 7.4.6: GPIO port output data register \(GPIO<sub>x</sub>\\_ODR\) \(x = A..H\)](#) for the register descriptions.

### 7.3.5 I/O data bitwise handling

The bit set reset register (GPIO<sub>x</sub>\_BSRR) is a 32-bit register which allows the application to set and reset each individual bit in the output data register (GPIO<sub>x</sub>\_ODR). The bit set reset register has twice the size of GPIO<sub>x</sub>\_ODR.

To each bit in GPIO<sub>x</sub>\_ODR, correspond two control bits in GPIO<sub>x</sub>\_BSRR: BSRR(i) and BSRR(i+SIZE). When written to 1, bit BSRR(i) sets the corresponding ODR(i) bit. When written to 1, bit BSRR(i+SIZE) resets the ODR(i) corresponding bit.

Writing any bit to 0 in GPIO<sub>x</sub>\_BSRR does not have any effect on the corresponding bit in GPIO<sub>x</sub>\_ODR. If there is an attempt to both set and reset a bit in GPIO<sub>x</sub>\_BSRR, the set action takes priority.

Using the GPIO<sub>x</sub>\_BSRR register to change the values of individual bits in GPIO<sub>x</sub>\_ODR is a "one-shot" effect that does not lock the GPIO<sub>x</sub>\_ODR bits. The GPIO<sub>x</sub>\_ODR bits can always be accessed directly. The GPIO<sub>x</sub>\_BSRR register provides a way of performing atomic bitwise handling.

There is no need for the software to disable interrupts when programming the GPIO<sub>x</sub>\_ODR at bit level: it is possible to modify one or more bits in a single atomic AHB write access.

### 7.3.6 GPIO locking mechanism

It is possible to freeze the GPIO control registers by applying a specific write sequence to the GPIO<sub>x</sub>\_LCKR register. The frozen registers are GPIO<sub>x</sub>\_MODER, GPIO<sub>x</sub>\_OTYPER, GPIO<sub>x</sub>\_OSPEEDR, GPIO<sub>x</sub>\_PUPDR, GPIO<sub>x</sub>\_AFRL and GPIO<sub>x</sub>\_AFRH.

To write the GPIO<sub>x</sub>\_LCKR register, a specific write / read sequence has to be applied. When the right LOCK sequence is applied to bit 16 in this register, the value of LCKR[15:0] is used to lock the configuration of the I/Os (during the write sequence the LCKR[15:0] value must be the same). When the LOCK sequence has been applied to a port bit, the value of the port bit can no longer be modified until the next MCU or peripheral reset. Each GPIO<sub>x</sub>\_LCKR bit freezes the corresponding bit in the control registers (GPIO<sub>x</sub>\_MODER, GPIO<sub>x</sub>\_OTYPER, GPIO<sub>x</sub>\_OSPEEDR, GPIO<sub>x</sub>\_PUPDR, GPIO<sub>x</sub>\_AFRL and GPIO<sub>x</sub>\_AFRH).

**General-purpose I/Os (GPIO)****RM0038**

The LOCK sequence (refer to [Section 7.4.8: GPIO port configuration lock register \(GPIOx\\_LCKR\) \(x = A..H\)](#)) can only be performed using a word (32-bit long) access to the GPIOx\_LCKR register due to the fact that GPIOx\_LCKR bit 16 has to be set at the same time as the [15:0] bits.

For more details refer to LCKR register description in [Section 7.4.8: GPIO port configuration lock register \(GPIOx\\_LCKR\) \(x = A..H\)](#).

### 7.3.7 I/O alternate function input/output

Two registers are provided to select one out of the sixteen alternate function inputs/outputs available for each I/O. With these registers, you can connect an alternate function to some other pin as required by your application.

This means that a number of possible peripheral functions are multiplexed on each GPIO using the GPIOx\_AFRL and GPIOx\_AFRH alternate function registers. The application can thus select any one of the possible functions for each I/O. The AF selection signal being common to the alternate function input and alternate function output, a single channel is selected for the alternate function input/output of one I/O.

To know which functions are multiplexed on each GPIO pin, refer to the datasheets.

**Note:** *The application is allowed to select one of the possible peripheral functions for each I/O at a time.*

### 7.3.8 External interrupt/wakeup lines

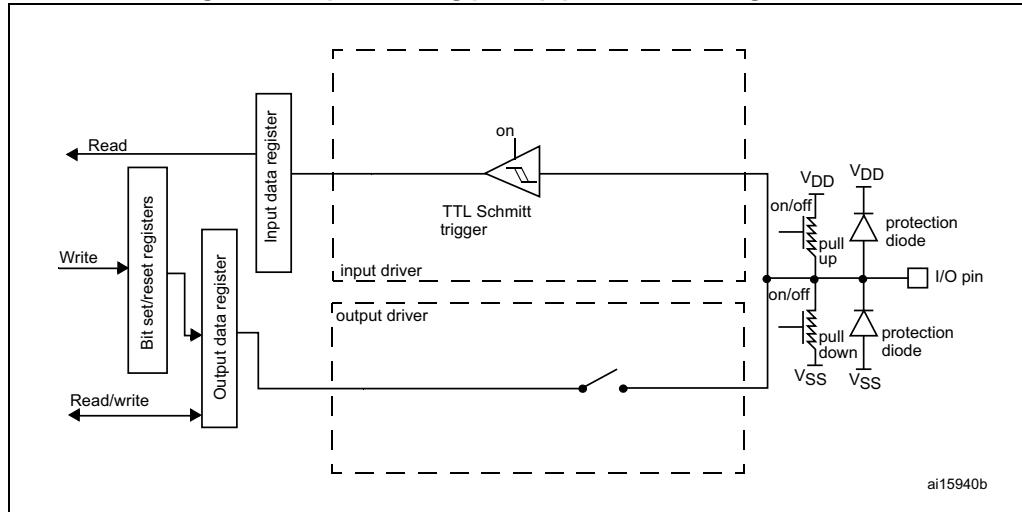
All ports have external interrupt capability. To use external interrupt lines, the port must be configured in input mode, refer to [Section 10.2: External interrupt/event controller \(EXTI\)](#) and [Section 10.2.3: Wakeup event management](#).

### 7.3.9 Input configuration

When the I/O port is programmed as Input:

- the output buffer is disabled
- the Schmitt trigger input is activated
- the pull-up and pull-down resistors are activated depending on the value in the GPIOx\_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register provides the I/O State

[Figure 21](#) shows the input configuration of the I/O port bit.

**Figure 21. Input floating/pull up/pull down configurations**

### 7.3.10 Output configuration

When the I/O port is programmed as output:

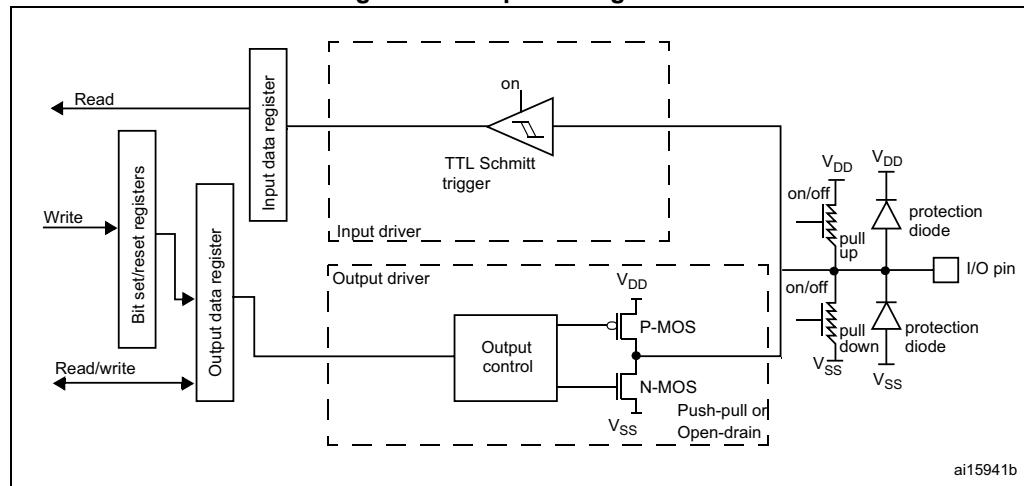
- The output buffer is enabled:
  - Open drain mode: A “0” in the Output register activates the N-MOS whereas a “1” in the Output register leaves the port in Hi-Z (the P-MOS is never activated)
  - Push-pull mode: A “0” in the Output register activates the N-MOS whereas a “1” in the Output register activates the P-MOS
- The Schmitt trigger input is activated
- The weak pull-up and pull-down resistors are activated or not depending on the value in the GPIOx\_PUPDR register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state
- A read access to the output data register gets the last written value

*Figure 22* shows the output configuration of the I/O port bit.

## General-purpose I/Os (GPIO)

RM0038

Figure 22. Output configuration



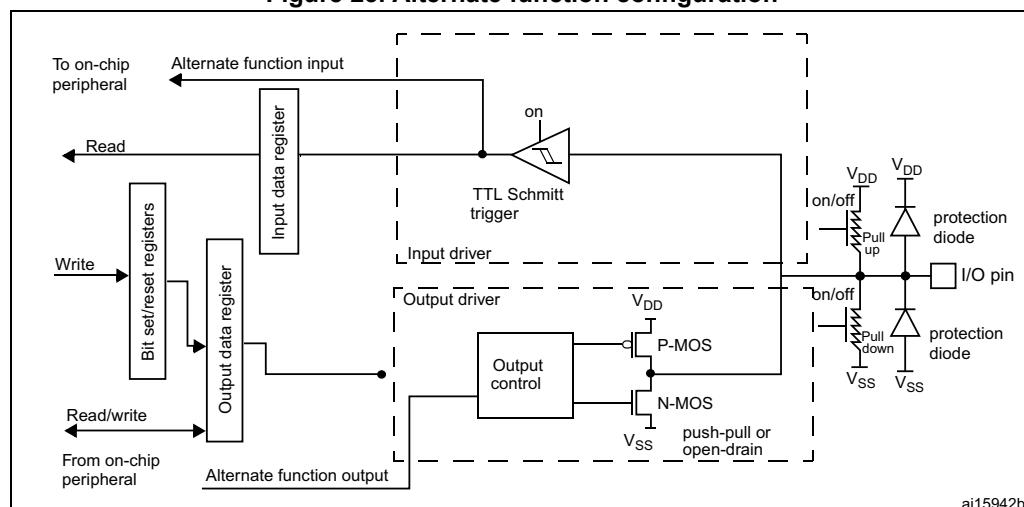
## 7.3.11 Alternate function configuration

When the I/O port is programmed as alternate function:

- The output buffer can be configured as open-drain or push-pull
- The output buffer is driven by the signal coming from the peripheral (transmitter enable and data)
- The Schmitt trigger input is activated
- The weak pull-up and pull-down resistors are activated or not depending on the value in the `GPIOx_PUPDR` register
- The data present on the I/O pin are sampled into the input data register every AHB clock cycle
- A read access to the input data register gets the I/O state

*Figure 23* shows the Alternate function configuration of the I/O port bit.

Figure 23. Alternate function configuration



### 7.3.12 Analog configuration

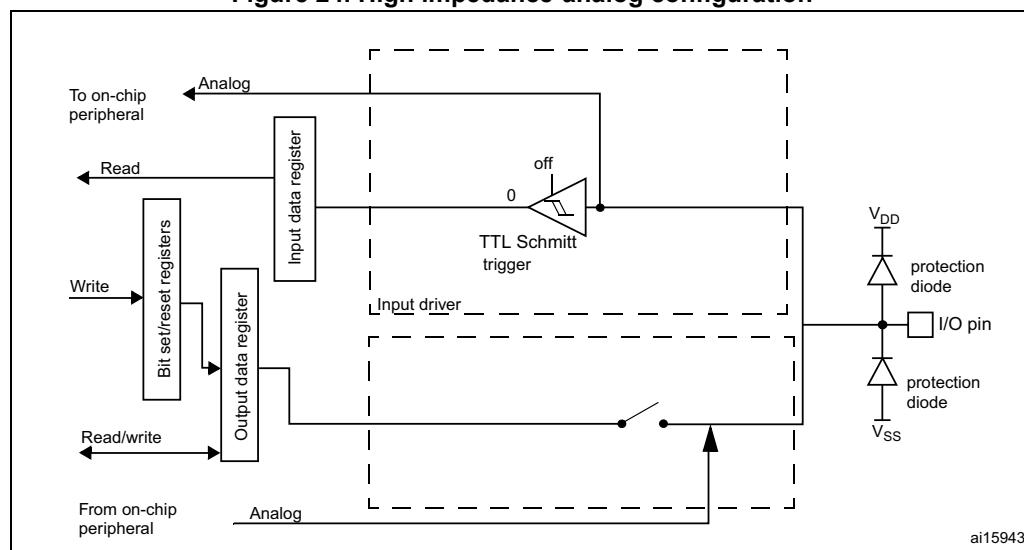
When the I/O port is programmed as analog configuration:

- The output buffer is disabled
- The Schmitt trigger input is deactivated, providing zero consumption for every analog value of the I/O pin. The output of the Schmitt trigger is forced to a constant value (0).
- The weak pull-up and pull-down resistors are disabled
- Read access to the input data register gets the value "0"

**Note:** *The alternate function configuration described above is not applied when the selected alternate function is a LCD function. In case, the I/O, programmed as an alternate function output, is configured as described in the analog configuration.*

*Figure 24* shows the high-impedance, analog-input configuration of the I/O port bit.

**Figure 24. High impedance-analog configuration**



### 7.3.13 Using the OSC32\_IN/OSC32\_OUT pins as GPIO PC14/PC15 port pins

The LSE oscillator pins OSC32\_IN and OSC32\_OUT can be used as general-purpose PC14 and PC15 I/Os, respectively, when the LSE oscillator is off. The PC14 and PC15 I/Os are only configured as LSE oscillator pins OSC32\_IN and OSC32\_OUT when the LSE oscillator is ON. This is done by setting the LSEON bit in the RCC\_BDCR register. The LSE has priority over the GPIO function.

**Note:** *The PC14/PC15 GPIO functionality is lost when the V<sub>CORE</sub> domain is powered off (by the device entering the standby mode). In this case the I/Os are set in analog input mode.*

### 7.3.14 Using the OSC\_IN/OSC\_OUT pins as GPIO PH0/PH1 port pins

The HSE oscillator pins OSC\_IN/OSC\_OUT can be used as general-purpose PH0/PH1 I/Os, respectively, when the HSE oscillator is OFF. (after reset, the HSE oscillator is off). The PH0/PH1 I/Os are only configured as OSC\_IN/OSC\_OUT HSE oscillator pins when the

**General-purpose I/Os (GPIO)****RM0038**

HSE oscillator is ON. This is done by setting the HSEON bit in the RCC\_CR register. The HSE has priority over the GPIO function.

**7.3.15 Selection of RTC\_AF1 alternate functions**

The STM32L1xxxx features:

- Two GPIO pins, which can be used as wakeup pins (WKUP1 and WKUP3).
- One GPIO pin, which can be used as a wakeup pin (WKUP2), for the detection of a tamper or time-stamp event, or to output RTC AFO\_ALARM or AFO\_CALIB.

The RTC\_AF1 pin (PC13) can be used for the following purposes:

- Wakeup pin 2 (WKUP2): this feature is enabled by setting the EWUP2 in the PWR\_CSR register.
- RTC AFO\_ALARM output: this output can be RTC Alarm A, RTC Alarm B or RTC Wakeup depending on the OSEL[1:0] bits in the RTC\_CR register.
- RTC AFO\_CALIB output: this feature is enabled by setting the COE[23] bit in the RTC\_CR register.
- RTC AF1\_TAMPER1: Tamper event detection
- Time-stamp event detection

The selection of the RTC AFO\_ALARM output is performed through the RTC\_TAFCR register as follows: ALARMOUTTYPE is used to select whether the RTC AFO\_ALARM output is configured in push-pull or open-drain mode.

The output mechanism follows the priority order shown in [Table 39](#).

**Table 39. RTC\_AF1 pin<sup>(1)</sup>**

Pin configuration and function	AFO_ALARM enabled	AFO_CALIB enabled	Tamper enabled	Time-stamp enabled	EWUP2 enabled	ALARMOUTTYPE AFO_ALARM configuration
Alarm out output OD	1	0	Don't care	Don't care	Don't care	0
Alarm out output PP	1	0	Don't care	Don't care	Don't care	1
Calibration out output PP	0	1	Don't care	Don't care	Don't care	Don't care
TAMPER input floating	0	0	1	0	Don't care	Don't care
TIMESTAMP and TAMPER input floating	0	0	1	1	Don't care	Don't care
TIMESTAMP input floating	0	0	0	1	Don't care	Don't care
Wakeup Pin 2	0	0	0	0	1	Don't care
Standard GPIO	0	0	0	0	0	Don't care

1. OD: open drain; PP: push-pull.

## 7.4 GPIO registers

This section gives a detailed description of the GPIO registers.

For a summary of register bits, register address offsets and reset values, refer to [Table 40](#).

The GPIO registers can be accessed by byte (8 bits), half-words (16 bits) or words (32 bits).

### 7.4.1 GPIO port mode register (GPIOx\_MODER) (x = A..H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

### 7.4.2 GPIO port output type register (GPIOx\_OTYPER) (x = A..H)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

**General-purpose I/Os (GPIO)**

RM0038

### 7.4.3 GPIO port output speed register (GPIO<sub>x</sub>\_OSPEEDR) (x = A..H)

Address offset: 0x08

Reset values:

- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw										

Bits 2y:2y+1 **OSPEEDR<sub>y</sub>[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

- 00: Low speed  
01: Medium speed  
10: High speed  
11: Very high speed

*Note:* Refer to the product datasheets for the values of OSPEEDR<sub>y</sub> bits versus  $V_{DD}$  range and external load.

### 7.4.4 GPIO port pull-up/pull-down register (GPIO<sub>x</sub>\_PUPDR) (x = A..H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw										

Bits 2y:2y+1 **PUPDR<sub>y</sub>[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

- 00: No pull-up, pull-down  
01: Pull-up  
10: Pull-down  
11: Reserved

#### 7.4.5 GPIO port input data register (GPIOx\_IDR) (x = A..H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

#### 7.4.6 GPIO port output data register (GPIOx\_ODR) (x = A..H)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

*Note:* For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx\_BSRR register (x = A..H).

#### 7.4.7 GPIO port bit set/reset register (GPIOx\_BSRR) (x = A..H)

Address offset: 0x18

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

**General-purpose I/Os (GPIO)**

RM0038

**Bits 31:16 BRy:** Port x reset bit y (y = 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Resets the corresponding ODRx bit

*Note: If both BSx and BRx are set, BSx has priority.*

**Bits 15:0 BSy:** Port x set bit y (y= 0..15)

These bits are write-only and can be accessed in word, half-word or byte mode. A read to these bits returns the value 0x0000.

0: No action on the corresponding ODRx bit

1: Sets the corresponding ODRx bit

#### 7.4.8 GPIO port configuration lock register (GPIOx\_LCKR) (x = A..H)

This register is used to lock the configuration of the port bits when a correct write sequence is applied to bit 16 (LCKK). The value of bits [15:0] is used to lock the configuration of the GPIO. During the write sequence, the value of LCKR[15:0] must not change. When the LOCK sequence has been applied on a port bit, the value of this port bit can no longer be modified until the next MCU or peripheral reset.

*Note: A specific write sequence is used to write to the GPIOx\_LCKR register. Only word access (32-bit long) is allowed during this write sequence.*

Each lock bit freezes a specific configuration register (control and alternate function registers).

Address offset: 0x1C

Reset value: 0x0000 0000

Access: 32-bit word only, read/write register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	LCKK
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	rw
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:17 Reserved, must be kept at reset value.

**Bit 16 LCKK[16]: Lock key**

This bit can be read any time. It can only be modified using the lock key write sequence.

0: Port configuration lock key not active

1: Port configuration lock key active. The GPIOx\_LCKR register is locked until an MCU reset or a peripheral reset occurs.

LOCK key write sequence:

WR LCKR[16] = '1' + LCKR[15:0]

WR LCKR[16] = '0' + LCKR[15:0]

WR LCKR[16] = '1' + LCKR[15:0]

RD LCKR

RD LCKR[16] = '1' (this read operation is optional but it confirms that the lock is active)

*Note: During the LOCK key write sequence, the value of LCK[15:0] must not change.*

*Any error in the lock sequence aborts the lock.*

*After the first lock sequence on any bit of the port, any read access on the LCKK bit returns '1' until the next CPU reset.*

**Bits 15:0 LCKy: Port x lock bit y (y = 0..15)**

These bits are read/write but can only be written when the LCKK bit is '0'.

0: Port configuration not locked

1: Port configuration locked

#### 7.4.9 GPIO alternate function low register (GPIOx\_AFRL) (x = A..H)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw												

**Bits 31:0 AFRLy: Alternate function selection for port x bit y (y = 0..7)**

These bits are written by software to configure alternate function I/Os

AFRLy selection:

0000: AF0                            1000: AF8

0001: AF1                            1001: AF9

0010: AF2                            1010: AF10

0011: AF3                            1011: AF11

0100: AF4                            1100: AF12

0101: AF5                            1101: AF13

0110: AF6                            1110: AF14

0111: AF7                            1111: AF15

**General-purpose I/Os (GPIO)****RM0038**

### 7.4.10 GPIO alternate function high register (GPIOx\_AFRH) ( $x = A..H$ )

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw												
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw												

Bits 31:0 **AFRH<sub>y</sub>**: Alternate function selection for port x bit y ( $y = 8..15$ )

These bits are written by software to configure alternate function I/Os

AFRH<sub>y</sub> selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

### 7.4.11 GPIO bit reset register (GPIOx\_BRR) ( $x = A..H$ )

These registers are available on *Cat.4, Cat.5 and Cat.6* products only.

Address offset: 0x28

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

Bits 31:16 Reserved

Bits 15:0 **BR<sub>y</sub>**: Port x Reset bit y ( $y = 0 .. 15$ )

These bits are write-only. A read to these bits returns the value 0x0000

0: No action on the corresponding ODR<sub>x</sub> bit1: Reset the corresponding ODR<sub>x</sub> bit

### 7.4.12 GPIO register map

The following table gives the GPIO register map and the reset values. The reserved memory areas are highlighted in gray in the table.

**Table 40. GPIO register map and reset values**

Offset	Register	Reset value	31
0x00	<b>GPIOA_MODER</b>	0 MODER15[1:0]	0 MODER15[1:0]
	Reset value	1 0	1 0
0x00	<b>GPIOB_MODER</b>	0 MODER14[1:0]	0 MODER14[1:0]
	Reset value	1 0	1 0
0x00	<b>GPIOx_MODER</b> (where x = C..H)	0 MODER13[1:0]	0 MODER13[1:0]
	Reset value	0 0	0 0
0x04	<b>GPIOx_OTYPER</b> (where x = A..H)	0 MODER12[1:0]	0 MODER12[1:0]
	Reset value	0 0	0 0
0x08	<b>GPIOx_OSPEEDR</b> (where x = A..H)	0 OSPEEDR15[1:0]	0 OSPEEDR15[1:0]
	Reset value	0 0	0 0
0x08	<b>GPIOB_OSPEEDR</b>	0 OSPEEDR14[1:0]	0 OSPEEDR14[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR15[1:0]	0 PUPDR15[1:0]
	Reset value	0 1	0 1
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR14[1:0]	0 PUPDR14[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR13[1:0]	0 PUPDR13[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR12[1:0]	0 PUPDR12[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR11[1:0]	0 PUPDR11[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR10[1:0]	0 PUPDR10[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR9[1:0]	0 PUPDR9[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR8[1:0]	0 PUPDR8[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR7[1:0]	0 PUPDR7[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR6[1:0]	0 PUPDR6[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR5[1:0]	0 PUPDR5[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR4[1:0]	0 PUPDR4[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR3[1:0]	0 PUPDR3[1:0]
	Reset value	0 1	0 1
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR2[1:0]	0 PUPDR2[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOA_PUPDR</b>	0 PUPDR1[1:0]	0 PUPDR1[1:0]
	Reset value	0 0	0 0
0x0C	<b>GPIOB_PUPDR</b>	0 PUPDR0[1:0]	0 PUPDR0[1:0]
	Reset value	0 0	0 0

## General-purpose I/Os (GPIO)

RM0038

Table 40. GPIO register map and reset values (continued)

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x0C	GPIOx_PUPDR (where x = C..H)	PUPDR15[1:0]																																
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x10	GPIOx_IDR (where x = A..H)																																	
	Reset value																																	
0x14	GPIOx_ODR (where x = A..H)																																	
	Reset value																																	
0x18	GPIOx_BSRR (where x = A..H)	BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BS0	ODR15	x	IDR15	0	PUPDR7[1:0]												
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x1C	GPIOx_LCKR (where x = A..H)																																	
	Reset value																																	
0x20	GPIOx_AFRL (where x = A..H)	AFRL7[3:0]	AFRL6[3:0]	AFRL5[3:0]	AFRL4[3:0]	AFRL3[3:0]	AFRL2[3:0]	AFRL1[3:0]	AFRL0[3:0]																									
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x24	GPIOx_AFRH (where x = A..H)	AFRH15[3:0]	AFRH14[3:0]	AFRH13[3:0]	AFRH12[3:0]	AFRH11[3:0]	AFRH10[3:0]	AFRH9[3:0]	AFRH8[3:0]																									
	Reset value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0x28	GPIOx_BRR (where x = A..H)																																	
	Reset value																																	

Refer to [Section 2.3: Memory map](#) for the register boundary addresses.



## Annexe D – Documentation d'autres composants

Led verte Everlight .....	281
Led bleue Liteon .....	291
Led rouge Multicomp .....	301



## Technical Data Sheet

### 0603 Package Chip LED (0.8 mm Height)

**19-21SYGC/S530-E2/TR8**

#### Features

- Package in 8mm tape on 7" diameter reel.
- Compatible with automatic placement equipment.
- Compatible with infrared and vapor phase reflow solder process.
- Mono-color type.
- Pb-free.
- The product itself will remain within ROHS complaint version.



#### Descriptions

- The 19-21 SMD LED is much smaller than lead frame type components, thus enable smaller board size, higher packing density, reduced storage space and finally smaller equipment to be obtained.
- Besides, lightweight makes them ideal for miniature applications. etc.

#### Applications

- Backlighting in dashboard and switch.
- Telecommunication: indicator and backlighting in telephone and fax.
- Flat backlight for LCD, switch and symbol.
- General use.
- Indoor signboard use.

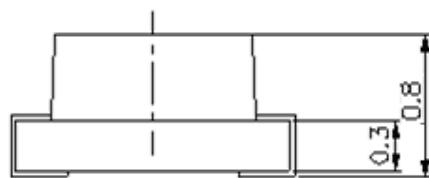
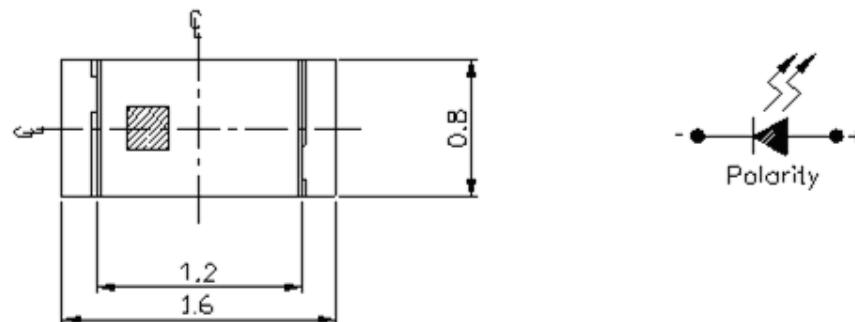
#### Device Selection Guide

Part No.	Chip	Emitted Color	Resin Color
	Material		
19-21SYGC/S530-E2/TR8	AlGaInP	Brilliant Yellow Green	Water Clear

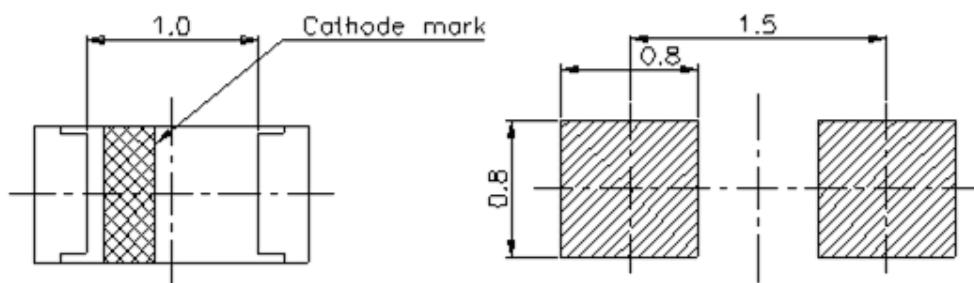


19-21SYGC/S530-E2/TR8

### Package Outline Dimensions



For reflow soldering



**Note:** The tolerances unless mentioned is  $\pm 0.1\text{mm}$ , Unit = mm

**19-21SYGC/S530-E2/TR8****Absolute Maximum Ratings (Ta=25°C)**

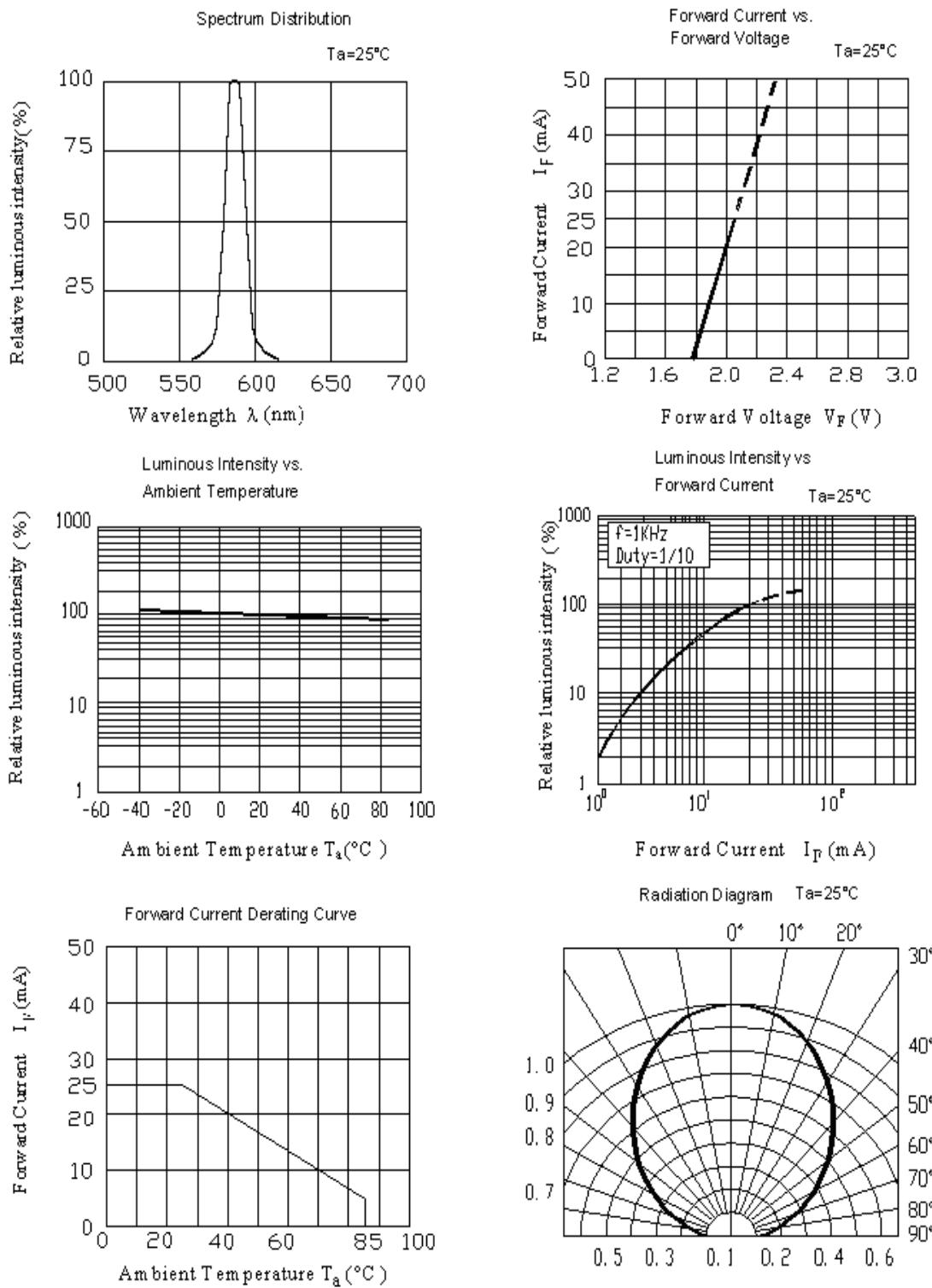
Parameter	Symbol	Rating	Unit
Reverse Voltage	V <sub>R</sub>	5	V
Forward Current	I <sub>F</sub>	25	mA
Peak Forward Current (Duty 1/10 @ 1KHz)	I <sub>FP</sub>	60	mA
Power Dissipation	P <sub>d</sub>	60	mW
Electrostatic Discharge(HBM)	ESD	2000	V
Operating Temperature	Topr	-40 ~ +85	°C
Storage Temperature	Tstg	-40 ~ +90	°C
Soldering Temperature	Tsol	Reflow Soldering : 260 °C for 10 sec. Hand Soldering : 350 °C for 3 sec.	

**Electro-Optical Characteristics (Ta=25°C)**

Parameter	Symbol	Min.	Typ.	Max.	Unit	Condition
Luminous Intensity	I <sub>v</sub>	16	21	-----	mcd	I <sub>F</sub> =20 mA
Viewing Angle	2θ 1/2	-----	100	-----	deg	
Peak Wavelength	λ <sub>p</sub>	-----	575	-----	nm	
Dominant Wavelength	λ <sub>d</sub>	-----	573	-----	nm	
Spectrum Radiation Bandwidth	△λ	-----	20	-----	nm	
Forward Voltage	V <sub>F</sub>	1.7	2.0	2.4	V	
Reverse Current	I <sub>R</sub>	-----	-----	10	μA	

19-21SYGC/S530-E2/TR8

## Typical Electro-Optical Characteristics Curves





19-21SYGC/S530-E2/TR8

### Label explanation

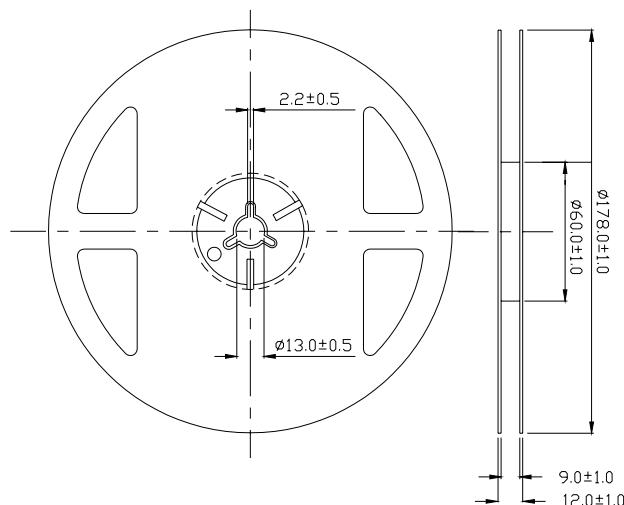
**CAT:** Luminous Intensity Rank

**HUE:** Dom. Wavelength Rank

**REF:** Forward Voltage Rank



### Reel Dimensions

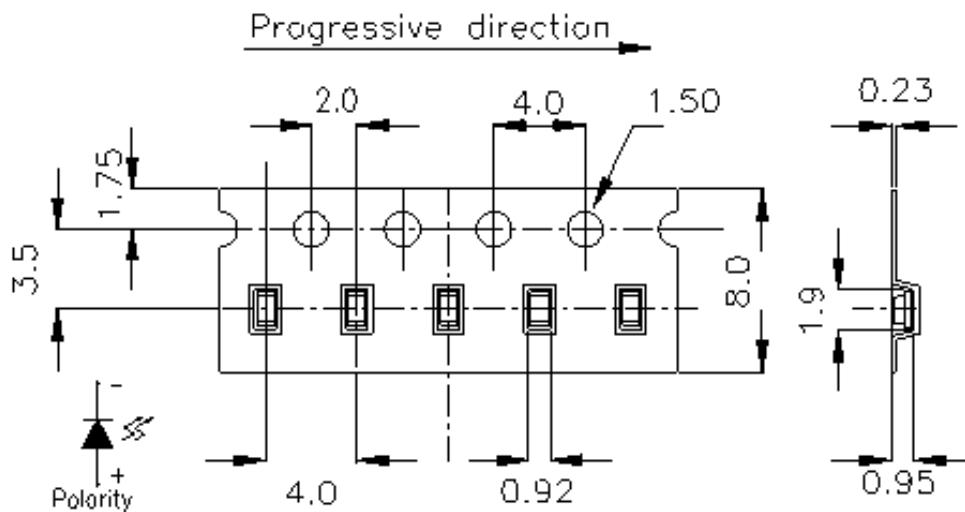


**Note:** The tolerances unless mentioned is  $\pm 0.1\text{mm}$ , Unit = mm



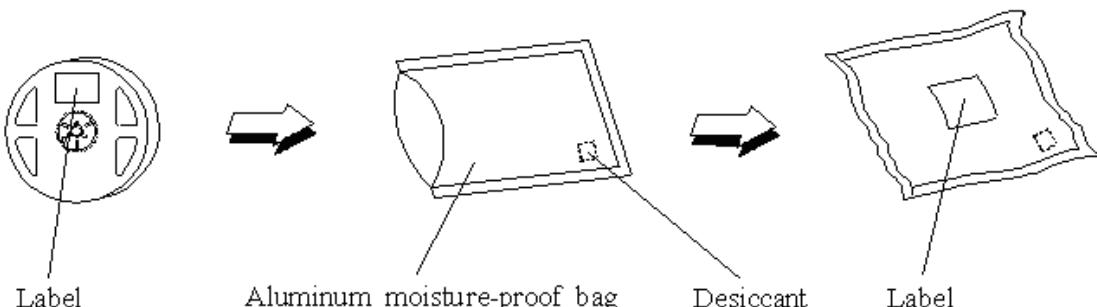
**19-21SYGC/S530-E2/TR8**

**Carrier Tape Dimensions: Loaded quantity 3000 PCS per reel**



**Note:** The tolerances unless mentioned is  $\pm 0.1\text{mm}$ , Unit = mm

**Moisture Resistant Packaging**





**19-21SYGC/S530-E2/TR8**

### **Reliability Test Items And Conditions**

The reliability of products shall be satisfied with items listed below.

Confidence level : 90%

LTPD : 10%

No.	Items	Test Condition	Test Hours/Cycles	Sample Size	Ac/Re
1	Reflow Soldering	Temp. : $260^{\circ}\text{C} \pm 5^{\circ}\text{C}$ Min. 5sec.	6 Min.	22 PCS.	0/1
2	Temperature Cycle	H : $+100^{\circ}\text{C}$ 15min ↓ 5 min L : $-40^{\circ}\text{C}$ 15min	300 Cycles	22 PCS.	0/1
3	Thermal Shock	H : $+100^{\circ}\text{C}$ 5min ↓ 10 sec L : $-10^{\circ}\text{C}$ 5min	300 Cycles	22 PCS.	0/1
4	High Temperature Storage	Temp. : $100^{\circ}\text{C}$	1000 Hrs.	22 PCS.	0/1
5	Low Temperature Storage	Temp. : $-40^{\circ}\text{C}$	1000 Hrs.	22 PCS.	0/1
6	DC Operating Life	$I_F = 20 \text{ mA}$	1000 Hrs.	22 PCS.	0/1
7	High Temperature / High Humidity	$85^{\circ}\text{C} / 85\% \text{RH}$	1000 Hrs.	22 PCS.	0/1



**19-21SYGC/S530-E2/TR8**

## Precautions For Use

### 1. Over-current-proof

Customer must apply resistors for protection , otherwise slight voltage shift will cause big current change ( Burn out will happen ).

### 2. Storage

2.1 Do not open moisture proof bag before the products are ready to use.

2.2 Before opening the package, the LEDs should be kept at 30°C or less and 90%RH or less.

2.3 After opening the package: The LED's floor life is 1 year under 30°C or less and 60% RH or less.

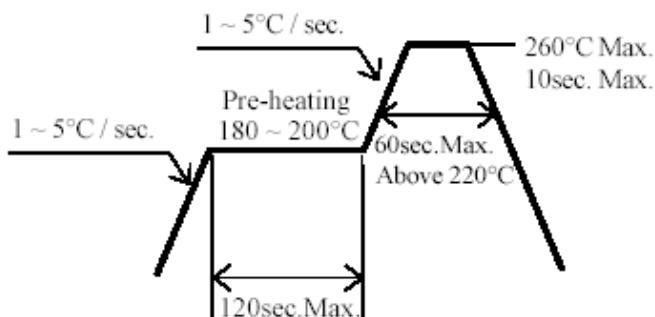
If unused LEDs remain, it should be stored in moisture proof packages.

2.4 If the moisture absorbent material (silica gel) has faded away or the LEDs have exceeded the storage time, baking treatment should be performed using the following conditions.

Baking treatment :  $60 \pm 5^\circ\text{C}$  for 24 hours.

### 3. Soldering Condition

#### 3.1 Pb-free solder temperature profile



3.2 Reflow soldering should not be done more than two times.

3.3 When soldering, do not put stress on the LEDs during heating.

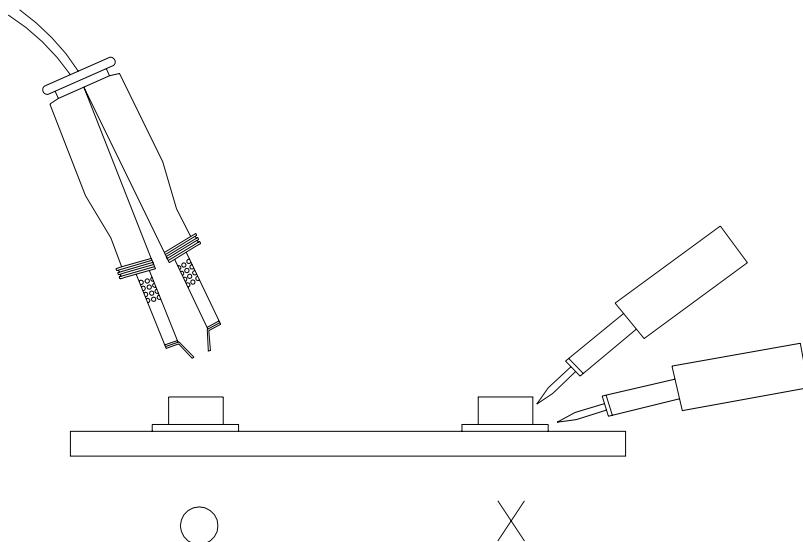
3.4 After soldering, do not warp the circuit board.

**19-21SYGC/S530-E2/TR8****4.Soldering Iron**

Each terminal is to go to the tip of soldering iron temperature less than 350°C for 3 seconds within once in less than the soldering iron capacity 25W. Leave two seconds and more intervals, and do soldering of each terminal. Be careful because the damage of the product is often started at the time of the hand solder.

**5.Repairing**

Repair should not be done after the LEDs have been soldered. When repairing is unavoidable, a double-head soldering iron should be used (as below figure). It should be confirmed beforehand whether the characteristics of the LEDs will or will not be damaged by repairing.



**EVERLIGHT ELECTRONICS CO., LTD.**  
Office: No 25, Lane 76, Sec 3, Chung Yang Rd,  
Tucheng, Taipei 236, Taiwan, R.O.C

Tel: 886-2-2267-2000, 2267-9936  
Fax: 886-2267-6244, 2267-6189, 2267-6306  
<http://www.everlight.com>

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Everlight:](#)

[19-21SYGC/S530-E2/TR8](#)

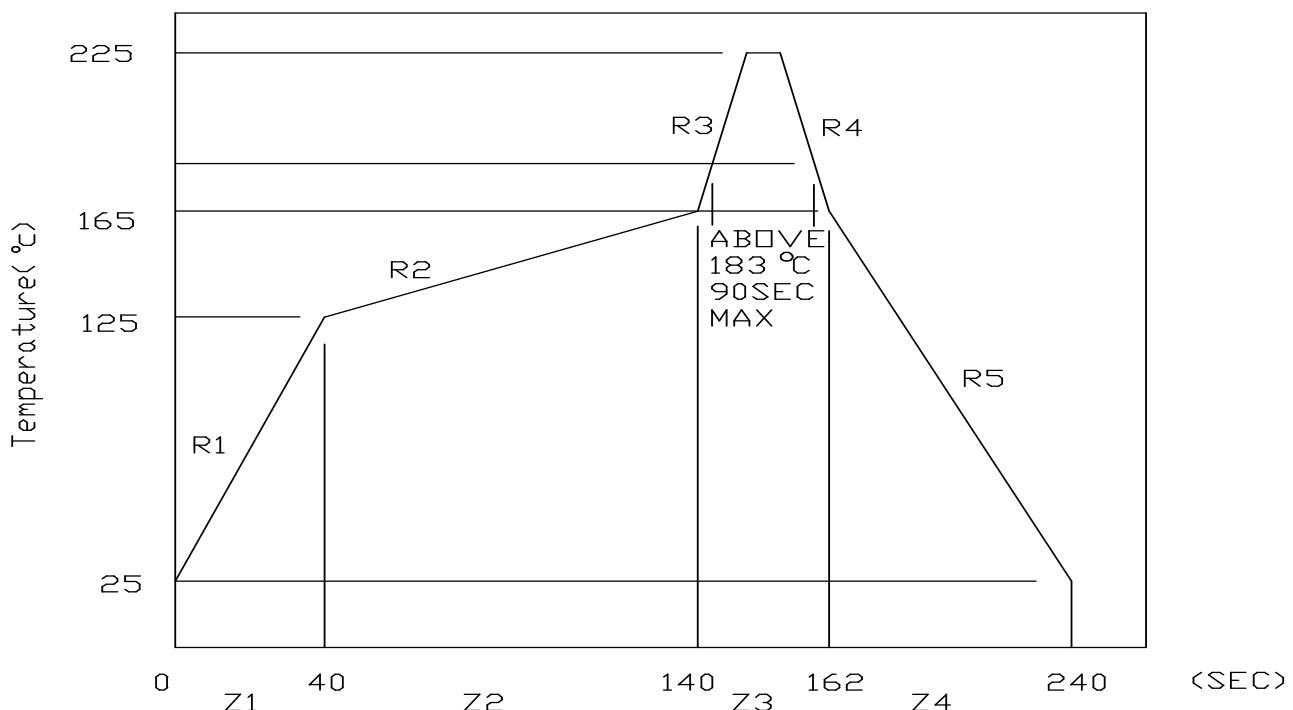
Absolute Maximum Ratings At Ta=25°C

Parameter	LTST-C193TBKT-5A	Unit
Power Dissipation	76	mW
Peak Forward Current (1/10 Duty Cycle, 0.1ms Pulse Width)	100	mA
DC Forward Current	20	mA
Derating Linear From 25°C	0.25	mA/°C
Reverse Voltage <sup>Note A</sup>	5	V
Operating Temperature Range	-20°C to + 80°C	
Storage Temperature Range	-30°C to + 100°C	
Wave Soldering Condition	260°C For 5 Seconds	
Infrared Soldering Condition	260°C For 5 Seconds	
Vapor Phase Soldering Condition	215°C For 3 Minutes	

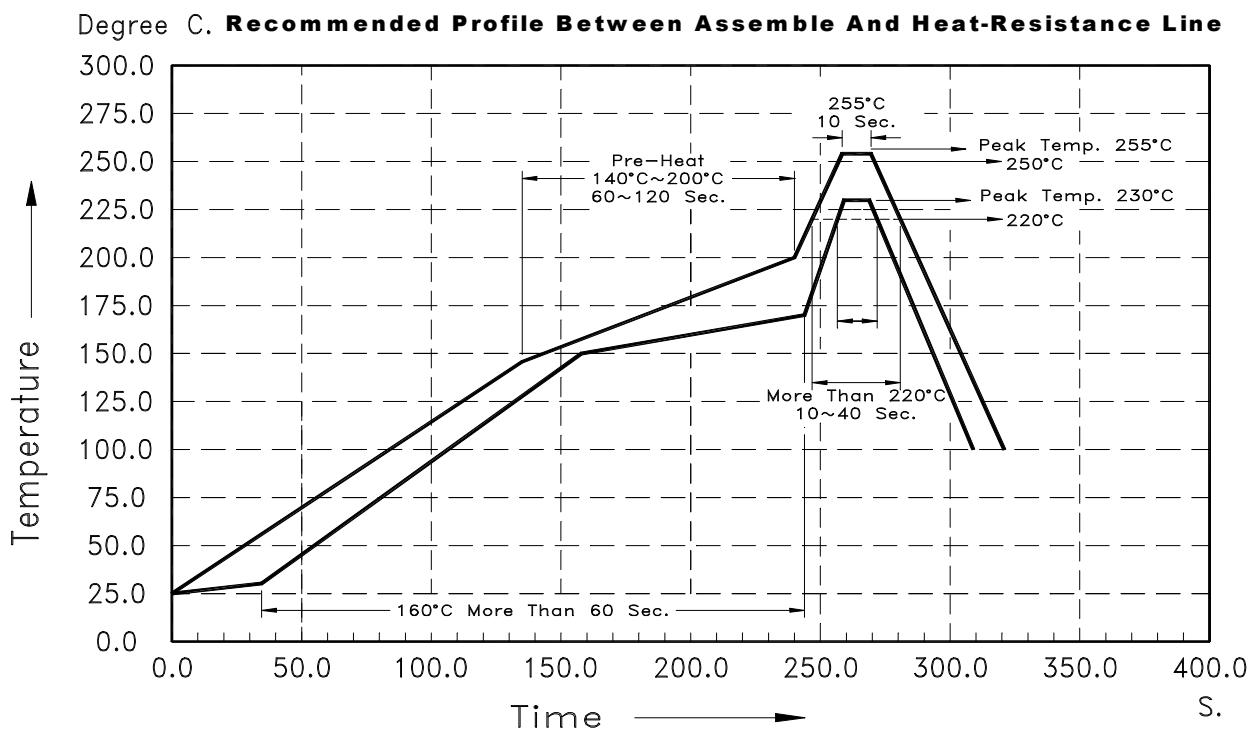
Note A: Reverse Voltage can't be continued operating

**Suggestion Profile:**

## (1) Suggestion IR Reflow Profile For Normal Process



## (2) Suggestion IR Reflow Profile For Pb Free Process



The Profile is available that must to use SnAg ( $\alpha=3.3 \sim 3.8$ ) Cu ( $\alpha=0.2 \sim 0.7$ ) solder paste

**Electrical Optical Characteristics At Ta=25°C**

Parameter	Symbol	Part No. LTST-	Min.	Typ.	Max.	Unit	Test Condition
Luminous Intensity	IV	C193TBKT-5A	11.2	15.0	45.0	mcd	IF = 5mA Note 1
Viewing Angle	$2\theta_{1/2}$	C193TBKT-5A		130		deg	Note 2 (Fig.6)
Peak Emission Wavelength	$\lambda_P$	C193TBKT-5A		468		nm	Measurement @Peak (Fig.1)
Dominant Wavelength	$\lambda_d$	C193TBKT-5A	465.0	-	475.0	nm	IF = 5mA Note 3
Spectral Line Half-Width	$\Delta\lambda$	C193TBKT-5A		25		nm	
Forward Voltage	VF	C193TBKT-5A	2.65	2.80	3.05	V	IF = 5mA
Reverse Current	IR	C193TBKT-5A			10	$\mu A$	VR = 5V

Notes: 1. Luminous intensity is measured with a light sensor and filter combination that approximates the

CIE eye-response curve.

2.  $\theta_{1/2}$  is the off-axis angle at which the luminous intensity is half the axial luminous intensity.

3. The dominant wavelength,  $\lambda_d$  is derived from the CIE chromaticity diagram and represents the single wavelength which defines the color of the device.

4. Caution in ESD:

Static Electricity and surge damages the LED. It is recommend to use a wrist band or anti-electrostatic glove when handling the LED. All devices, equipment and machinery must be properly grounded.

### Bin Code List

Forward Voltage		Unit: V @5mA
Bin Code	Min.	Max.
1	2.65	2.75
2	2.75	2.85
3	2.85	2.95
4	2.95	3.05

Tolerance on each Forward Voltage bin is +/-0.1 volt

Luminous Intensity		Unit : mcd @5mA
Bin Code	Min.	Max.
L1	11.2	14.0
L2	14.0	18.0
M1	18.0	22.4
M2	22.4	28.0
N1	28.0	35.5
N2	35.5	45.0

Tolerance on each Intensity bin is +/-15%

Dominant Wavelength		Unit : nm @5mA
Bin Code	Min.	Max.
AC	465.0	470.0
AD	470.0	475.0

Tolerance for each Dominate Wavelength bin is +/- 1nm

**Typical Electrical / Optical Characteristics Curves**

(25 °C Ambient Temperature Unless Otherwise Noted)

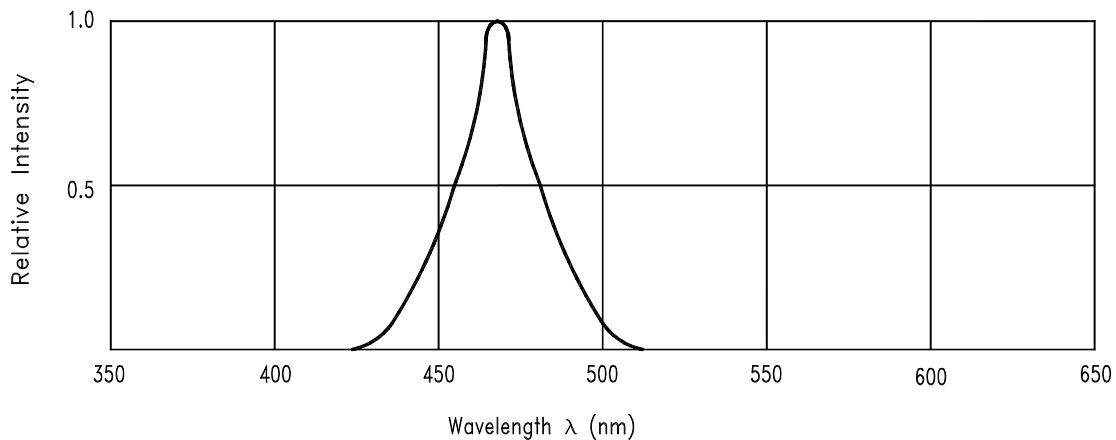


Fig.1 Relative Intensity vs. Wavelength

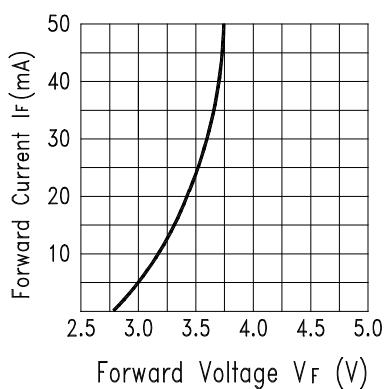


Fig.2 Forward Current vs.  
Forward Voltage

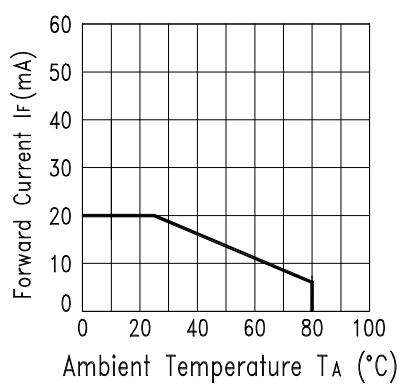


Fig.3 Forward Current  
Derating Curve

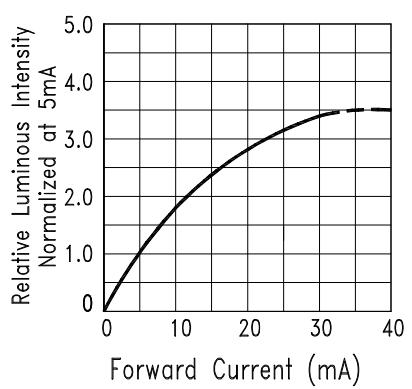


Fig.4 Relative Luminous Intensity  
vs. Forward Current

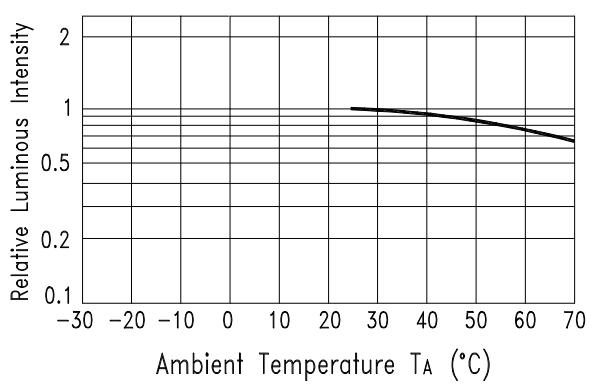


Fig.5 Luminous Intensity vs.  
Ambient Temperature

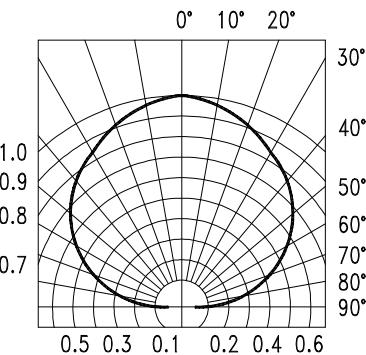
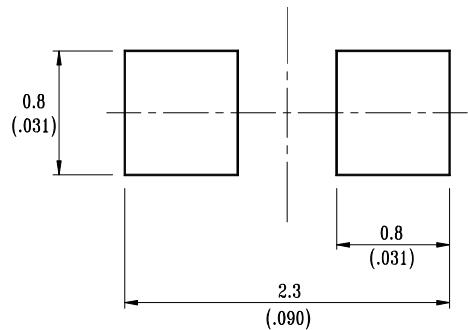


Fig.6 Spatial Distribution

### Cleaning

Do not use unspecified chemical liquid to clean LED they could harm the package.  
 If clean is necessary, immerse the LED in ethyl alcohol or in isopropyl alcohol at normal temperature for less one minute.

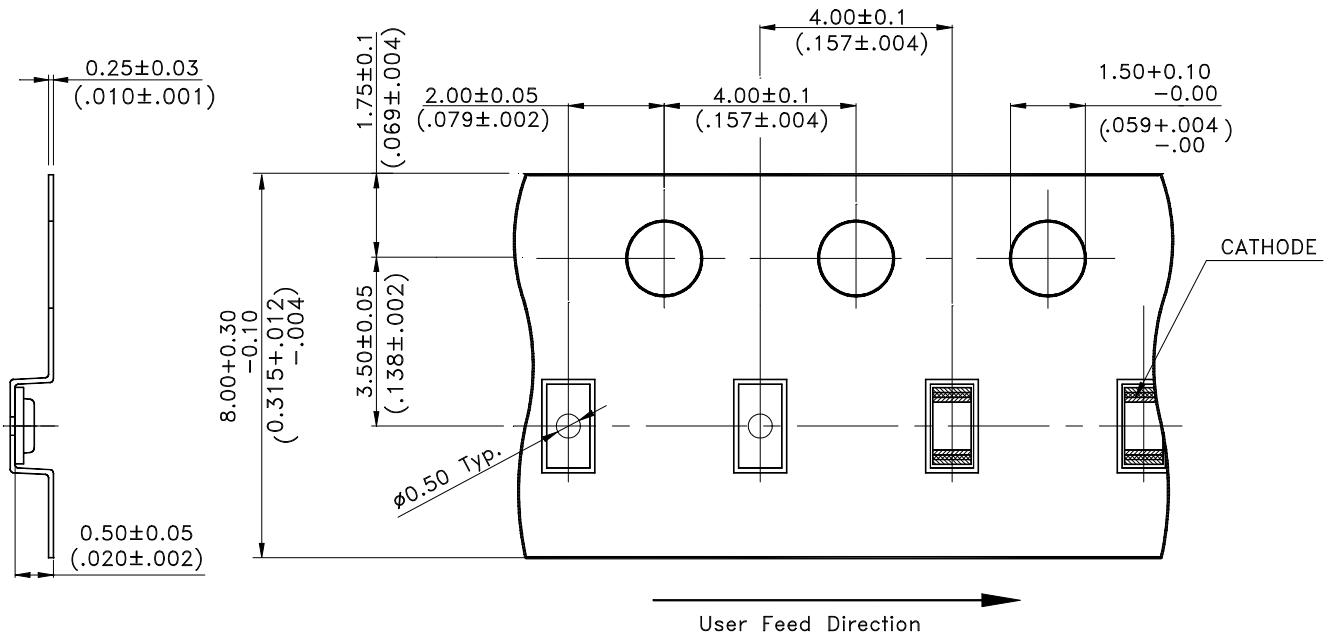
### Suggest Soldering Pad Dimensions



Notes:

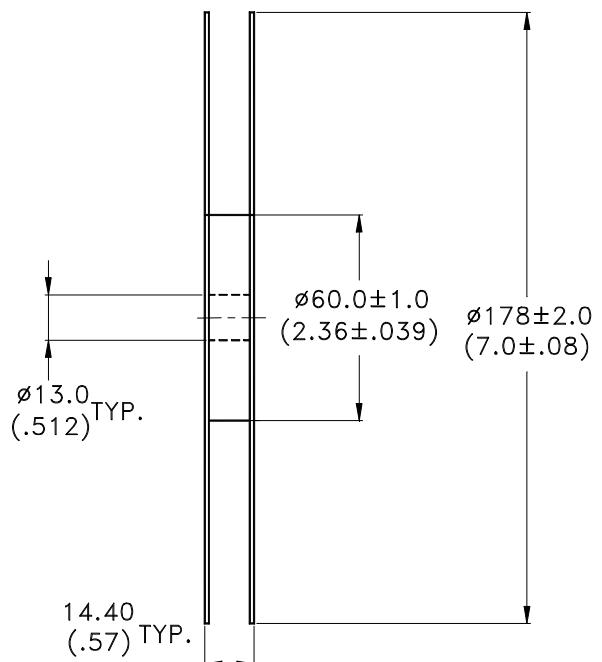
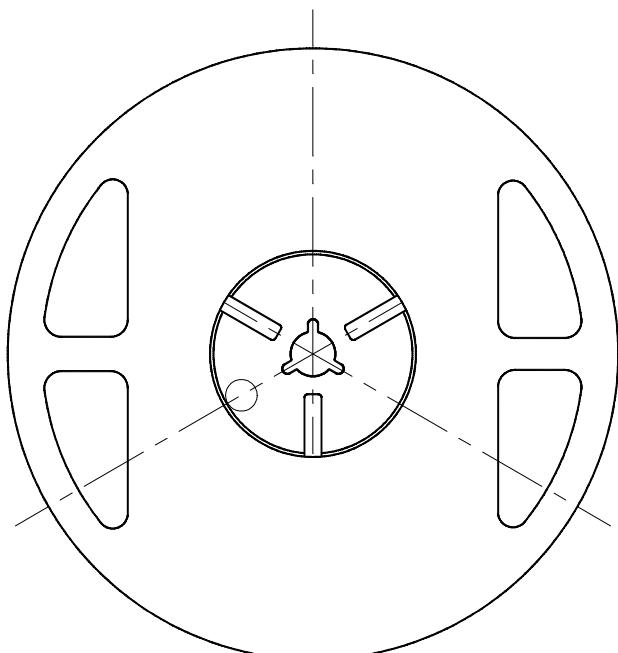
1. Suggest stencil thickness is maximum 0.10mm.

### Package Dimensions Of Tape And Reel



Notes:

1. All dimensions are in millimeters (inches).



Notes:

1. Empty component pockets sealed with top cover tape.
2. 7 inch reel-3000 pieces per reel.
3. Minimum packing quantity is 500 pcs for remainders.
4. The maximum number of consecutive missing lamps is two.
5. In accordance with ANSI/EIA 481-1-A-1994 specifications.

## CAUTIONS

### 1. Application

The LEDs described here are intended to be used for ordinary electronic equipment (such as office equipment, communication equipment and household applications). Consult Liteon's Sales in advance for information on applications in which exceptional reliability is required, particularly when the failure or malfunction of the LEDs may directly jeopardize life or health (such as in aviation, transportation, traffic control equipment, medical and life support systems and safety devices).

### 2. Storage

The storage ambient for the LEDs should not exceed 30°C temperature or 70% relative humidity.

It is recommended that LEDs out of their original packaging are IR-reflowed within one week.

For extended storage out of their original packaging, it is recommended that the LEDs be stored in a sealed container with appropriate desiccant, or in a desiccators with nitrogen ambient.

LEDs stored out of their original packaging for more than a week should be baked at about 60 deg C for at least 24 hours before solder assembly.

### 3. Cleaning

Use alcohol-based cleaning solvents such as isopropyl alcohol to clean the LED if necessary.

### 4. Soldering

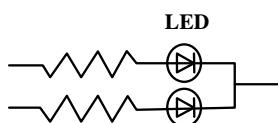
Recommended soldering conditions:

Reflow soldering		Wave Soldering		Soldering iron	
Pre-heat	120~150°C	Pre-heat	100°C Max.	Temperature	300°C Max.
Pre-heat time	120 sec. Max.	Pre-heat time	60 sec. Max.	Soldering time	3 sec. Max.
Peak temperature	260°C Max.	Solder wave	260°C Max.		(one time only)
Soldering time	5 sec. Max.	Soldering time	10 sec. Max.		

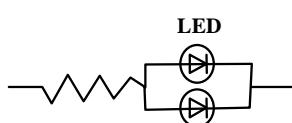
### 5. Drive Method

An LED is a current-operated device. In order to ensure intensity uniformity on multiple LEDs connected in parallel in an application, it is recommended that a current limiting resistor be incorporated in the drive circuit, in series with each LED as shown in Circuit A below.

**Circuit model A**



**Circuit model B**



(A) Recommended circuit.

(B) The brightness of each LED might appear different due to the differences in the I-V characteristics of those LEDs.

### 6. ESD (Electrostatic Discharge)

Static Electricity or power surge will damage the LED.

Suggestions to prevent ESD damage:

- Use of a conductive wrist band or anti-electrostatic glove when handling these LEDs.
- All devices, equipment, and machinery must be properly grounded.
- Work tables, storage racks, etc. should be properly grounded.
- Use ion blower to neutralize the static charge which might have built up on surface of the LED's plastic lens as a result of friction between LEDs during storage and handling.

**Property of Lite-On Only**

ESD-damaged LEDs will exhibit abnormal characteristics such as high reverse leakage current, low forward voltage, or “ no lightup ” at low currents.

To verify for ESD damage, check for “ lightup ” and Vf of the suspect LEDs at low currents.

The Vf of “ good ” LEDs should be >2.0V@0.1mA for InGaN product and >1.4V@0.1mA for AlInGaP product.

## 7. Reliability Test

Classification	Test Item	Test Condition	Reference Standard
Endurance Test	Operation Life	Ta= Under Room Temperature As Per Data Sheet Maximum Rating *Test Time= 1000HRS (-24HRS,+72HRS)	MIL-STD-750D:1026 MIL-STD-883D:1005 JIS C 7021:B-1
	High Temperature High Humidity Storage	IR-Reflow In-Board, 2 Times Ta= $65 \pm 5^\circ\text{C}$ , RH= 90~95% *Test Time= 240HRS±2HRS	MIL-STD-202F:103B JIS C 7021:B-11
	High Temperature Storage	Ta= $105 \pm 5^\circ\text{C}$ *Test Time= 1000HRS (-24HRS,+72HRS)	MIL-STD-883D:1008 JIS C 7021:B-10
	Low Temperature Storage	Ta= $-55 \pm 5^\circ\text{C}$ *Test Time=1000HRS (-24HRS,+72HRS)	JIS C 7021:B-12
Environmental Test	Temperature Cycling	105°C ~ 25°C ~ -55°C ~ 25°C 30mins 5mins 30mins 5mins 10 Cycles	MIL-STD-202F:107D MIL-STD-750D:1051 MIL-STD-883D:1010 JIS C 7021:A-4
	Thermal Shock	IR-Reflow In-Board, 2 Times $85 \pm 5^\circ\text{C}$ ~ $-40^\circ\text{C} \pm 5^\circ\text{C}$ 10mins 10mins 10 Cycles	MIL-STD-202F:107D MIL-STD-750D:1051 MIL-STD-883D:1011
	Solder Resistance	T.sol= $260 \pm 5^\circ\text{C}$ Dwell Time= $10 \pm 1$ secs	MIL-STD-202F:210A MIL-STD-750D:2031 JIS C 7021:A-1
	IR-Reflow Normal Process	Ramp-up rate( $183^\circ\text{C}$ to Peak) $+3^\circ\text{C}$ / second max Temp. maintain at $125(\pm 25)^\circ\text{C}$ 120 seconds max Temp. maintain above $183^\circ\text{C}$ 60-150 seconds Peak temperature range $235^\circ\text{C} +5/-0^\circ\text{C}$ Time within $5^\circ\text{C}$ of actual Peak Temperature (tp) 10-30 seconds Ramp-down rate $+6^\circ\text{C}$ /second max	MIL-STD-750D:2031.2 J-STD-020C
	IR-Reflow Pb Free Process	Ramp-up rate( $217^\circ\text{C}$ to Peak) $+3^\circ\text{C}$ / second max Temp. maintain at $175(\pm 25)^\circ\text{C}$ 180 seconds max Temp. maintain above $217^\circ\text{C}$ 60-150 seconds Peak temperature range $260^\circ\text{C} +0/-5^\circ\text{C}$ Time within $5^\circ\text{C}$ of actual Peak Temperature (tp) 20-40 seconds Ramp-down rate $+6^\circ\text{C}$ /second max	MIL-STD-750D:2031.2 J-STD-020C
	Solderability	T.sol= $235 \pm 5^\circ\text{C}$ Immersion time $2 \pm 0.5$ sec Immersion rate $25 \pm 2.5$ mm/sec Coverage $\geq 95\%$ of the dipped surface	MIL-STD-202F:208D MIL-STD-750D:2026 MIL-STD-883D:2003 IEC 68 Part 2-20 JIS C 7021:A-2

## 8. Others

The appearance and specifications of the product may be modified for improvement without prior notice.

## 9. Suggested Checking List

### Training and Certification

1. Everyone working in a static-safe area is ESD-certified?
2. Training records kept and re-certification dates monitored?

### Static-Safe Workstation & Work Areas

1. Static-safe workstation or work-areas have ESD signs?
2. All surfaces and objects at all static-safe workstation and within 1 ft measure less than 100V?
3. All ionizer activated, positioned towards the units?
4. Each work surface mats grounding is good?

### Personnel Grounding

1. Every person (including visitors) handling ESD sensitive (ESDS) items wears wrist strap, heel strap or conductive shoes with conductive flooring?
2. If conductive footwear used, conductive flooring also present where operator stand or walk?
3. Garments, hairs or anything closer than 1 ft to ESD items measure less than 100V\*?
4. Every wrist strap or heel strap/conductive shoes checked daily and result recorded for all DLs?
5. All wrist strap or heel strap checkers calibration up to date?

Note: \*50V for Blue LED.

### Device Handling

1. Every ESDS items identified by EIA-471 labels on item or packaging?
2. All ESDS items completely inside properly closed static-shielding containers when not at static-safe workstation?
3. No static charge generators (e.g. plastics) inside shielding containers with ESDS items?
4. All flexible conductive and dissipative package materials inspected before reuse or recycles?

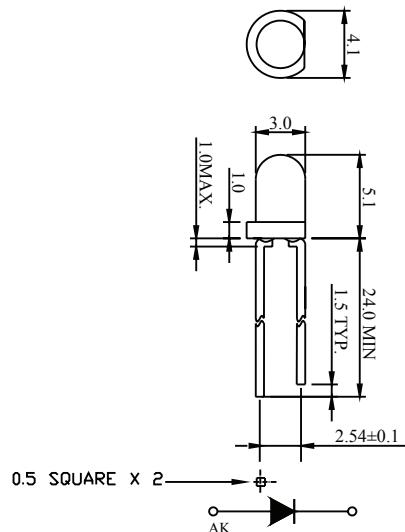
### Others

1. Audit result reported to entity ESD control coordinator?
2. Corrective action from previous audits completed?
3. Are audit records complete and on file?

# 3mm Round LED Lamp



## Package Dimensions:



## Features:

- High red lamp
- Made with AlGaAs / AlGaAs chip and red clear epoxy resin

All dimensions are in mm  
Tolerance: ±0.25mm

## Absolute Maximum Ratings at Ta=25°C

Parameter	Symbol	Rating	Unit
Power Dissipation	P <sub>D</sub>	72	mW
Reverse Voltage	V <sub>R</sub>	4	V
D.C. Forward Current	I <sub>f</sub>	30	mA
Reverse (Leakage) Current	I <sub>r</sub>	100	µA
Peak Current (1 / 10 Duty Cycle, 0.1ms Pulse Width)	I <sub>f</sub> (Peak)	100	mA
Operating Temperature Range	T <sub>opr</sub>	-25 to + 85	°C
Storage Temperature Range	T <sub>stg</sub>	-40 to +100	°C
Soldering Temperature (1.6mm from body)	T <sub>sol</sub>	Dip Soldering: 260°C for 5sec. Hand Soldering: 350°C for 3sec.	

# 3mm Round LED Lamp

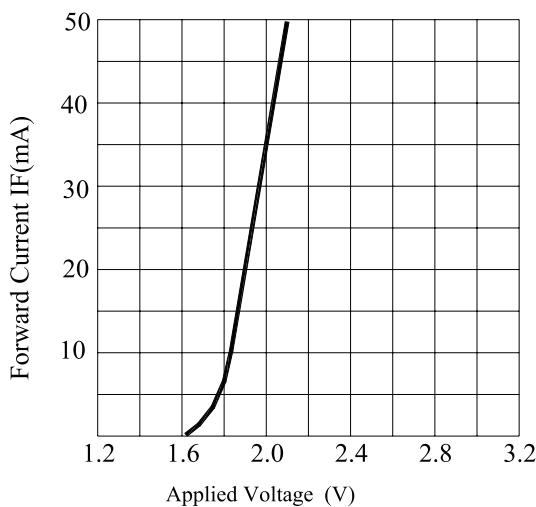
## Electrical & Optical Characteristics:

Parameter	Symbol	Condition	Min.	Typ.	Max.	Unit
Luminous Intensity	I <sub>v</sub>	I <sub>f</sub> = 20mA	110	300		mcd
Forward Voltage	V <sub>f</sub>	I <sub>f</sub> = 20mA		1.9	2.4	V
Peak Wavelength	λ <sub>p</sub>	I <sub>f</sub> = 20mA		660		nm
Dominant Wavelength	λ <sub>d</sub>	I <sub>f</sub> = 20mA		643		nm
Reverse (Leakage) Current	I <sub>r</sub>	V <sub>r</sub> = 5V			100	μA
Viewing Angle	2θ½	I <sub>f</sub> = 20mA		35		deg
Spectrum Line Halfwidth	Δλ	I <sub>f</sub> = 20mA		20		nm

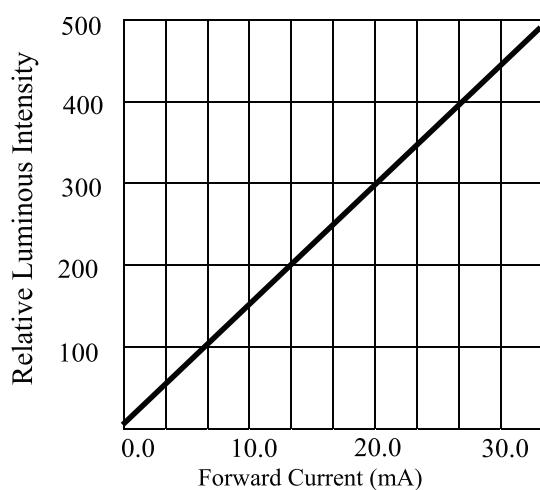
Notes: 1. The data is tested by IS tester.

2. Customer's special requirements are also welcome.

## Typical Electrical & Optical Characteristics Curves:

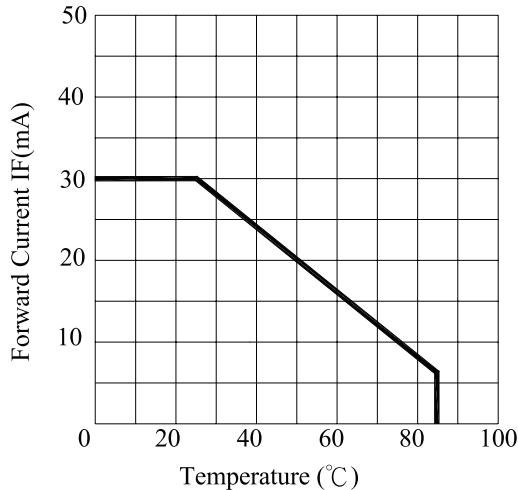
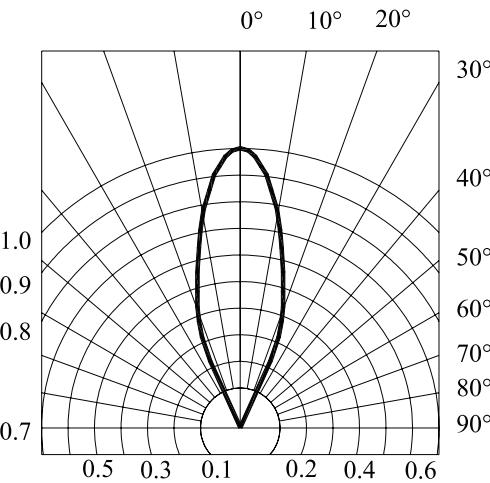


FORWARD CURRENT VS.APPLIED VOLTAGE



FORWARD CURRENT VS. LUMINOUS INTENSITY

# 3mm Round LED Lamp

FORWARD CURRENT VS. AMBIENT TEMPERATURERADIATION DIAGRAM

## Part Number Table

LED Chip		Lens Colour	Part Number
Material	Emitting Colour		
AlGaAs / AlGaAs	High red	Red clear	703-0091

**Important Notice :** This data sheet and its contents (the "Information") belong to the members of the Premier Farnell group of companies (the "Group") or are licensed to it. No licence is granted for the use of it other than for information purposes in connection with the products to which it relates. No licence of any intellectual property rights is granted. The Information is subject to change without notice and replaces all data sheets previously supplied. The Information supplied is believed to be accurate but the Group assumes no responsibility for its accuracy or completeness, any error in or omission from it or for any use made of it. Users of this data sheet should check for themselves the Information and the suitability of the products for their purpose and not make any assumptions based on information included or omitted. Liability for loss or damage resulting from any reliance on the Information or use of it (including liability resulting from negligence or where the Group was aware of the possibility of such loss or damage arising) is excluded. This will not operate to limit or restrict the Group's liability for death or personal injury resulting from its negligence. Multicomp is the registered trademark of the Group. © Premier Farnell plc 2012.

## Annexe E – Signaux tout-ourien : pas assez de tension ou de courant ?

### E.1 Introduction

On l'a vu, les "sources de tension" équivalentes aux pins des ports ne sont pas de très haute qualité, ce qui signifie qu'elles ne sont pas faites pour délivrer des courants très élevés. Et puis aussi la tension est limitée au mieux à 3 V et on comprend bien que c'est aussi quelque chose de limitant. Alors comment faire si on veut piloter quelque chose qui demande de la puissance ? Par exemple : simplement une LED blanche de puissance ? Une résistance chauffante ? Un ventilateur ? un moteur ? etc. La solution est le plus souvent d'ajouter au circuit "**un transistor**". Mais il faut savoir le faire, et c'est ce que nous montrons ici.

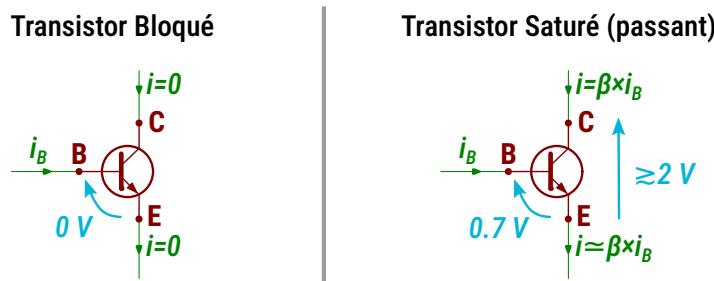
#### Important

On ne va traiter ici que le cas des signaux "tout-ourien". Traiter l'amplification de signaux plus généraux tels que les signaux analogiques est un sujet bien plus complexe, parce que l'on cherche alors de fonctionner en régime "linéaire", ce qui est immensément plus difficile que ce que l'on envisage ici. Mais dans énormément de cas, le travail avec les microcontrôleurs fait que l'on peut se contenter de fonctionner avec des signaux tout-ourien, y compris pour traiter des signaux analogiques (voir le codage "PWM" dans le cours). Ce sujet est traité dans d'autres modules en EEA. ■

### E.2 Le transistor en mode "tout-ourien" (saturé-bloqué)

Nous n'allons pas rentrer ici dans un cours complet sur les transistors, ça n'est pas la vocation du cours. Mais nous allons donner quelques points de repères pour utiliser les transistors avec des sorties "tout-ourien". Nous ne traiterons que le transistor **bipolaire npn**, parce qu'il est très courant dans les situations que nous décrivons.

Le transistor npn est donc un composant à 3 pattes (base, émetteur, collecteur) qui, en configuration tout-ourien, s'utilise presque toujours de la même façon :



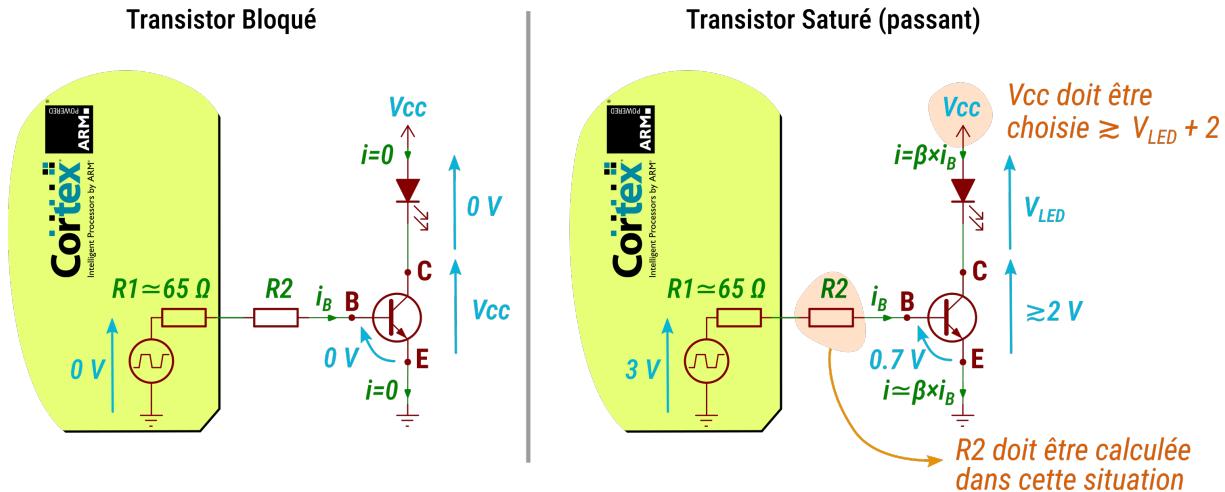
On pilote le transistor en imposant une tension  $v_{BE}$  :

- à  $v_{BE} = 0V$  pour imposer un courant nul au niveau collecteur-base,
- en imposant une tension  $v_{BE} \approx 0,7V$  pour permettre un courant important de passer au niveau collecteur-base. Si cette condition est vérifiée, alors on a :

$$i_E \approx i_C = \beta i_B \quad \beta \approx 60 \text{ à } 400$$

Pour que cela soit vrai, il faut aussi que la tension collecteur-émetteur  $v_{CE}$  soit "suffisante" pour le transistor, typiquement au moins 2V.

Il faut bien comprendre que les valeurs de  $v_{BE}$ ,  $i_B$  et  $v_{CE}$  sont imposées par les composants à l'extérieur du circuit, et que l'objectif est d'obtenir une valeur de  $i_C$  suffisante pour faire fonctionner le composant à faire fonctionner. Concrètement, cela nous amène à la situation ci-dessous :



Plusieurs remarques :

- Quand par programmation, on impose un état bas sur la pin voulue, alors la source de tension équivalente vaut 0V, et forcément la tension  $v_{BE}$  vaut aussi 0V, et donc le transistor est bloqué et aucun courant ne peut circuler au niveau du collecteur : la LED est donc éteinte.
- Quand par programmation on impose un état haut, il faut considérer deux choses : il faut prendre en compte le courant nécessaire pour la LED et en déduire le courant de base utile :

$$i_B = i_C / \beta$$

et vérifier que la tension  $V_{CC}$  sera suffisante pour la tension utile pour la LED et pour le  $v_{CE}$  du transistor.

### Remarque

On préfère placer le composant à piloter dans la branche du collecteur plutôt que dans la branche de l'émetteur parce que cela permet de piloter la tension  $v_{BE}$  de façon plus intuitive (= moins de calculs compliqués). En effet, comme l'émetteur est à la masse la tension de base sera toujours de 0.7 volts par rapport à la masse, quel que soit le composant à piloter.

Sur un exemple, si on veut piloter une LED de 60mA nécessitant 2,9V, avec un transistor disposant d'un  $\beta = 200$  :

- Calcul de  $R_2$  : on aura besoin d'un courant de base  $i_B = i_C / \beta = 0,3\text{mA}$ . Regardons maintenant ce qui se passe au niveau de la partie base-émetteur du transistor. La loi des mailles permet d'écrire que :

$$3 - 65 \times i_B - R_2 \times i_B - 0.7 = 0$$

d'où  $R_2 = 2280\Omega$ .

Nous aurons besoin de  $V_{CC} > 2,9 + 2$  soit  $V_{CC} > 4,9\text{V}$ . Ainsi, en exagérant, n'importe quelle source de tension supérieure à 4,9V conviendra. En réalité il faut prendre en compte la puissance dissipée par le transistor : si on augmente  $V_{CC}$ ,  $v_{CE}$  augmentera d'autant, puisque  $V_{LED}$  est fixé à  $i_C$  constant. Et donc la limite est donnée par le produit  $\approx v_{CE} \times i_C$ . Ici donc, on peut très bien choisir 5V ou 9V ou encore 12V sans problème pour beaucoup de transistors.

### Remarque

**A.** On ne demande plus que  $0,3\text{ mA}$  au microcontrôleur pour piloter la LED, en comparaison avec plusieurs milliampères ou dizaines de milliampères précédemment. La source de tension qui correspond à la pin sera donc utilisée dans de meilleures conditions, et le microcontrôleur chauffera moins (mais le transistor chauffera à sa place).

**B.** Retenez bien que  $v_{CE}$  va s'ajuster automatiquement si  $V_{CC}$  est trop importante, ce n'est pas "tellelement" un problème, mis à part la puissance à dissiper. Si on n'a pas le choix, par exemple parce qu'on a une tension d'alimentation qui ne peut pas changer, on peut ajouter une résistance en série avec la LED pour éviter que le transistor ne chauffe trop (on répartit alors la dissipation de puissance entre la résistance et le transistor).

## ❖ En résumé ...

Utiliser un transistor en mode "saturé-bloqué", n'est en général pas compliqué. Les **règles de design** sont les suivantes :

1. On part d'une configuration où l'émetteur est à la masse,
2. On place le composant à piloter entre le collecteur et la tension d'alimentation. A ce stade, il est important de se souvenir que cette tension d'alimentation n'est pas forcément la même que celle du microcontrôleur, elle sera souvent supérieure.
3. On se renseigne sur le "point de fonctionnement" du composant en question : courant  $i_{composant}$  et tension  $v_{composant}$  nécessaires.
4. On calcule la tension d'alimentation avec une règle simple :

$$v_{CC} = v_{composant} + 2V$$

Ce calcul peut être affiné en cherchant, dans le datasheet du transistor précis que l'on va utiliser, quelle valeur de  $v_{CE}$  convient vraiment pour le transistor choisi.

5. On calcule le courant de base  $i_B$  nécessaire. Il faut :

$$i_B = i_{composant} / \beta$$

où  $\beta$  est caractéristique du transistor et dépend donc du datasheet.

6. On relie la pin du microcontrôleur, configurée (par code) pour être une "sortie", à la base, à travers une résistance. Cette résistance est calculée garantir  $v_{BE} = 0.7V$  lorsque la pin sera activée (à 1) et pour le courant de base  $i_B$  calculé précédemment.

Enfin, il faut veiller à un ensemble de **précautions quant au choix du transistor** :

- il faut bien sur qu'il puisse délivrer un courant de collecteur  $i_C$  au moins égal à  $i_{composant}$
- et il faut faire attention à ce que la puissance qu'il consomme, c'est à dire :

$$P \approx V_{CE} \times i_C$$

soit inférieur à ce qu'il peut dissiper. Cette information est disponible normalement dans le datasheet du transistor.

- Enfin, si jamais la tension d'alimentation était imposée pour d'autres raisons et si le transistor ne peut pas dissiper la puissance, on peut ajouter une résistance de faible valeur (mais de puissance) en série avec le composant à piloter, et la calculer pour diminuer  $v_{CE}$  dans les limites de ce que le transistor peut accepter.