A la découverte d'Arduino

Par Stuart Cording, Directeur Technique des Ventes, Atmel Munich

Une petite carte bleue à circuit imprimé est à l'origine d'une mini révolution en électronique. Son nom : Arduino. Les développeurs chevronnés de systèmes embarqués commenceront peut être par s'en détourner, car cette carte minimaliste ressemble davantage à un jouet qu'à un outil destiné aux programmeurs professionnels. C'est exactement pour cette raison qu'elle est devenue si populaire et que sa notoriété ne cesse de croître. Depuis sa création, la famille de cartes Arduino s'est étendue et compte une douzaine de membres, de facteurs de forme variés et équipés de différents microcontrôleurs afin de répondre à la diversité des besoins. La plupart de ces cartes ont pour cerveau le microcontrôleur AVR d'Atmel, et les plus récentes sont équipées du microcontrôleur SAM3X d'Atmel à cœur ARM.

Avec la profusion de cartes et de plates-formes de développement sur le marché, pourquoi avoir besoin d'une autre encore ? Une réponse élégante est que les microcontrôleurs sont bien trop complexes. Naturellement, ils ne sont pas si compliqués pour ceux d'entre nous qui ont passé une décennie ou plus à développer autour de ces produits. Cependant, en dehors de notre industrie, les développeurs qui cherchent à ajouter une technologie de commande à leur produit ont besoin d'un outil dont l'assimilation ne soit pas aussi longue qu'avec la plupart des produit typiques pour microcontrôleurs.

Prenons pour exemple un USART. Un microcontrôleur AVR typique en possède plusieurs, et la première décision consiste à choisir celui à utiliser. Inévitablement, l'interface sélectionnée est associée à des broches multifonctions. Ainsi, avant de pouvoir entreprendre la moindre tâche, il nous faut lire le paragraphe de la fiche technique consacré à l'USART et la documentation expliquant comment sélectionner la fonction USART pour les broches choisies. Une fois cet travail accompli, il nous faut réfléchir à d'autres paramètres : comment calculer le débit ; quelle est la relation entre la mesure en bauds et la fréquence d'horloge de l'AVR ; comment activer les interruptions de l'USART ; comment écrire une routine d'interruption pour la chaîne de compilation choisie... La liste est longue et le temps est compté pour les débutants. Ceux d'entre nous qui travaillent à temps plein sur des microcontrôleurs savent qu'il faut des années pour acquérir l'expérience nécessaire à l'implémentation rapide et efficace d'une telle interface.

Les développeurs de la plate-forme Arduino en sont conscients et fournissent, dans leur environnement de développement, une API (interface de programmation d'applications) et des bibliothèques qui permettent de réaliser des fonctions complexes, comme les interfaces série, avec très peu de code et encore moins de connaissances sur ce qui se passe au cœur de la plate-forme. C'est ce qui ouvre le monde de la programmation embarquée et de l'électronique aux hobbyistes et aux étudiants, et le rend accessible à d'autres domaines comme l'art et la mode. En fait, la programmation embarquée et l'électronique peuvent aller partout où l'électronique est capable de s'associer à d'autres disciplines.

Pourquoi une nouvelle plate-forme ?

Un grand nombre de plates-formes de développement existent déjà pour presque la quasi-totalité des architectures de microcontrôleurs les plus connues. Elles tendent cependant à n'être qu'autant

de variations autour du même thème : fournir un microcontrôleur physique dans un format utilisable aisément dans soudure, et permettre un démarrage rapide en offrant toute la circuiterie externe nécessaire (quartz, émetteurs-récepteurs et alimentation). Les fabricants de semiconducteurs utilisent évidemment leurs outils de développement en complément de leur marketing pour promouvoir certains produits ou fonctions intégrées. Selon le site web d'Arduino[1], les points différentiateurs de sa plate-forme sont les suivants :

- Un faible prix : les modules pré-assemblés coûtent moins de 50 $, et les plus simples peuvent même être créés manuellement.
- Une plate-forme croisée : l'environnement de développement de code tourne sous Windows, Linux et Mac OS, une fonctionnalité qui semble encore problématique pour de nombreux fabricants de microcontrôleurs.
- Un environnement de programmation simple et clair, basé sur « Processing », un langage disposant d'un IDE (environnement de développement intégré) sobrement limité à quelques boutons et un éditeur de texte. Il est possible, si on le souhaite, d'ajouter du code C standard à un projet.
- Logiciel et matériel open source et extensible : selon la tendance d'aujourd'hui, tous les logiciels et toutes les conceptions matérielles sont publiés sous licence Creative Commons. Cela permet aux utilisateurs de modifier ce qui est disponible et de l'améliorer. Naturellement, le partage est vivement encouragé !

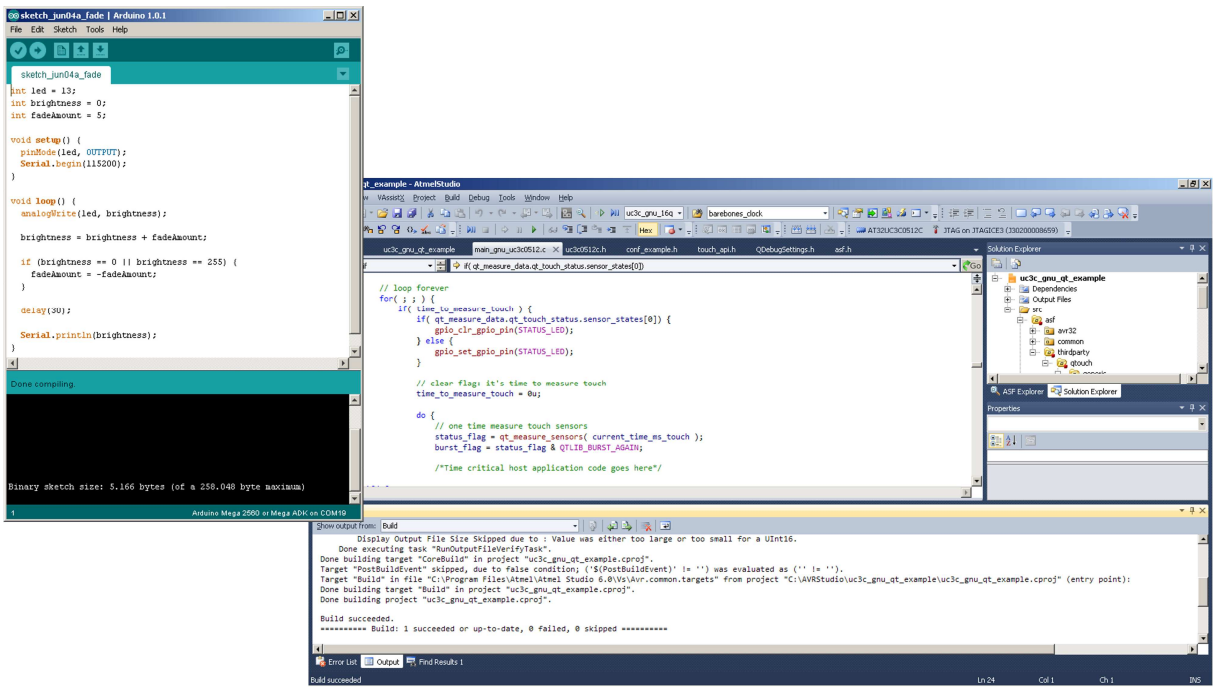


Figure 1 – L'IDE Arduino, vue côte à côte avec Atmel Studio 6, apparaît comme bien plus simple !

Ces avantages ont un prix. L'impact des inconvénients présentés ci-dessous dépend essentiellement de votre point de vue :

- Pas de débogage : l'environnement de développement Arduino ne supporte que la programmation de la carte cible. Pour pallier l'absence de support de débogage, une sortie

textuelle facile d'emploi est implémentée via le port USART, qui fournit des messages de débogage. De plus, l'IDE intègre une fonction moniteur série.

- Un cadre logiciel fixe : la fréquence de l'oscillateur, par exemple, est fixée, ce qui signifie que toutes les autres fonctions d'Arduino seront exécutées dans les limites de la fréquence choisie. De plus, certaines des bibliothèques offertes utilisent des fonctions hardware particulières, comme des timers, qui peuvent ne pas être disponibles pour notre application.

Que reçoit-on avec une carte Arduino?

A priori, pas grand' chose. Une carte Arduino comprend un régulateur d'alimentation ; un port USB qui supporte à la fois l'interface de programmation et l'interface série de classe CDC, et peut aussi servir d'alimentation ; le microcontrôleur AVR ; un bouton de reset ; et un ensemble de connecteurs. Ces connecteurs sont groupés selon une règle brute facile à comprendre. En premier lieu se trouve le connecteur d'alimentation pour les entrées Vin, 5 V, 3,3 V, GND et une broche IOREF de support des microcontrôleurs alimentant leurs E/S en 3,3 V plutôt qu'avec l'habituel 5 V. Le connecteur « ANALOG IN » permet des mesures analogiques via le convertisseur A/N du microcontrôleur. Il reste enfin le groupe de broches « DIGITAL » (numériques), utilisables en entrées ou sorties GPIO. Elles sont encore divisées entre sorties « PWM » et ports « COMMUNICATION », où se trouvent les interfaces série. Les seules fonctions matérielles sur la plupart des cartes sont une LED connectée à une broche du groupe DIGITAL, et l'USART relié à l'interface USB classe CDC sur la carte.

L'intention ici n'est pas d'offrir une plate-forme capable de tout faire. Au lieu de cela, l'équipe Arduino encourage les utilisateurs à fusionner leur carte avec des circuits électroniques de leur cru pour implémenter leurs propres créations. Arduino est un moyen et non une fin. Cela dit, certaines fonctions que nous aimerions implémenter peuvent poser problème, par exemple au niveau de la gestion des boîtiers de circuits intégrés ou de la complexité. Ainsi, l'implémentation d'une connexion Ethernet ou d'une liaison sans fil n'est pas une tâche facile, même pour des ingénieurs expérimentés. La conception mécanique des cartes Arduino permet très facilement, grâce au développement des « Shields », d'ajouter ces fonctionnalités à l'aide de modules standards partagés par la communauté.

Comme l'implantation mécanique des connecteurs de la plupart des cartes Arduino respecte une certaine cohérence, et comme les cartes les plus grandes sont des sur-ensembles de la carte Arduino Uno, il est possible de concevoir des modules standard, ou Shields, assurant une bonne compatibilité avec les cartes Arduino aussi bien qu'avec les autres Shields.

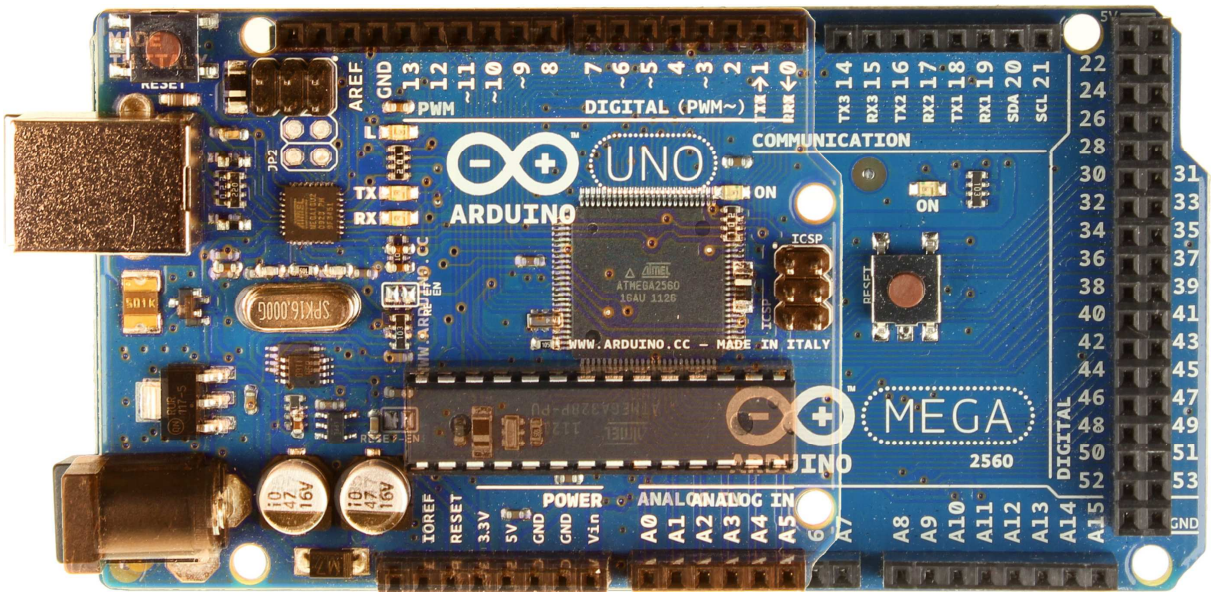


Figure 2 – Superposition de la carte Arduino UNO sur la carte Arduino MEGA 2560, montrant le principe de sur-ensembles qui régit toute la famille

Concevoir un Shield Arduino

Aussi brillantes que peuvent être les cartes et leur présentation sur son site web, Arduino conserve le côté « terre vierge » de tous les produits open source. Ce qui veut dire que les produits et services sont en général excellents, mais que la documentation reste quelque peu succincte. Il n'existe malheureusement aucune documentation officielle sur la manière de développer un Shield Arduino. Il n'existe pas non plus de canal officiel pour obtenir une reconnaissance officielle de la conformité d'un nouveau Shield. L'approche serait plutôt de lancer le produit (sur le marché) pour voir ce que ça donne, en espérant que, lorsqu'un Shield suscite assez d'intérêt, sa promotion se fera automatiquement à travers la communauté jusqu'au point où il deviendra trop important pour être ignoré.

Les défis à relever lors du développement d'un Shield Arduino sont entre autres :

- Les différentes tailles de cartes : nous l'avons vu, certaines cartes Arduino ont plus de connecteurs que d'autres. Bien évidemment, il n'est pas possible à chaque Shield de définir une ressource fixe à partir de la carte Arduino sous-jacente, puisque d'autres cartes de la pile peuvent aussi revendiquer cette ressource. Il est donc possible que différents Shields soient incompatibles entre eux lorsqu'ils sont empilés.
- Les différentes versions d'Arduino : sur les cartes Arduino de la révision 3, une broche IOREF permet aux Shields de savoir si le microcontrôleur utilise une alimentation d'entrée/ sortie de 3,3 V ou de 5 V. Les révisions précédentes de la plate-forme Arduino ne disposent pas de cette fonction ; les futurs Shields devraient pouvoir l'utiliser et, en même temps, rester rétrocompatibles.
- Les modèles PCB standard : il n'existe pas de modèles génériques ou « templates » standard de PCB pour les Shields, ce qui aiderait pourtant beaucoup à concevoir un tracé de circuit imprimé compatible avec un système donné.

- Pas de fournitures bien définies ; il est clair qu'un matériel livré sans exemples logiciels, pilotes logiciels ou documentation est beaucoup moins utile.

Bien sûr, comme il est d'usage dans les terres vierges de l'open source, il existe pour ceux qui les recherchent des suggestions, des solutions de rechange et des recommendations[2]. Si vous êtes le type de personne qui aime ajuster l'idée à la spécification, au lieu de créer la spécification à partir de l'idée, vous ne serez pas vraiment à l'aise.

Implémentation d'un Shield QTouch

Le processus du développement d'un élément hardware tel qu'un Shield Arduino ne diffère pas de celui de tout autre circuit. Pour conclure cet article, un Shield tactile capacitif basé sur la technologie QTouch® d'Atmel a été développé. Le projet appelé « TickleShield » est hébergé sur le nouveau site web collaboratif « Spaces » d'Atmel[3]. Son but est de créer une carte facile d'emploi apportant une interface tactile à tout projet nécessitant une commande simple ou une commande graphique par menus. Le choix s'est porté sur le contrôleur tactile capacitif AT42QT2120 qui fournit cinq boutons et un signal « GUARD » de blindage protégeant des lectures involontaires lorsque l'humidité ou des liquides sont essuyés sur la surface tactile. Le contrôleur offre de plus une entrée « détection de proximité » pouvant servir à allumer l'éclairage ambiant dès que l'approche d'une main est détectée. L'état de l'AT42QT2120 est lu sur six sorties logiques reliées à six entrées numériques de la carte Arduino à laquelle il est connecté. Pour utiliser le signal de blindage en configuration autonome, les sorties de l'AT42QT sont également envoyées vers un connecteur au pas standard de 0,1 pouce. Au cas ou le brochage standard du blindage ne convient pas à l'utilisateur, les liaisons de 0 Ohm peuvent être retirées et les sorties du contrôleur AT42QT reconfigurées comme désiré.

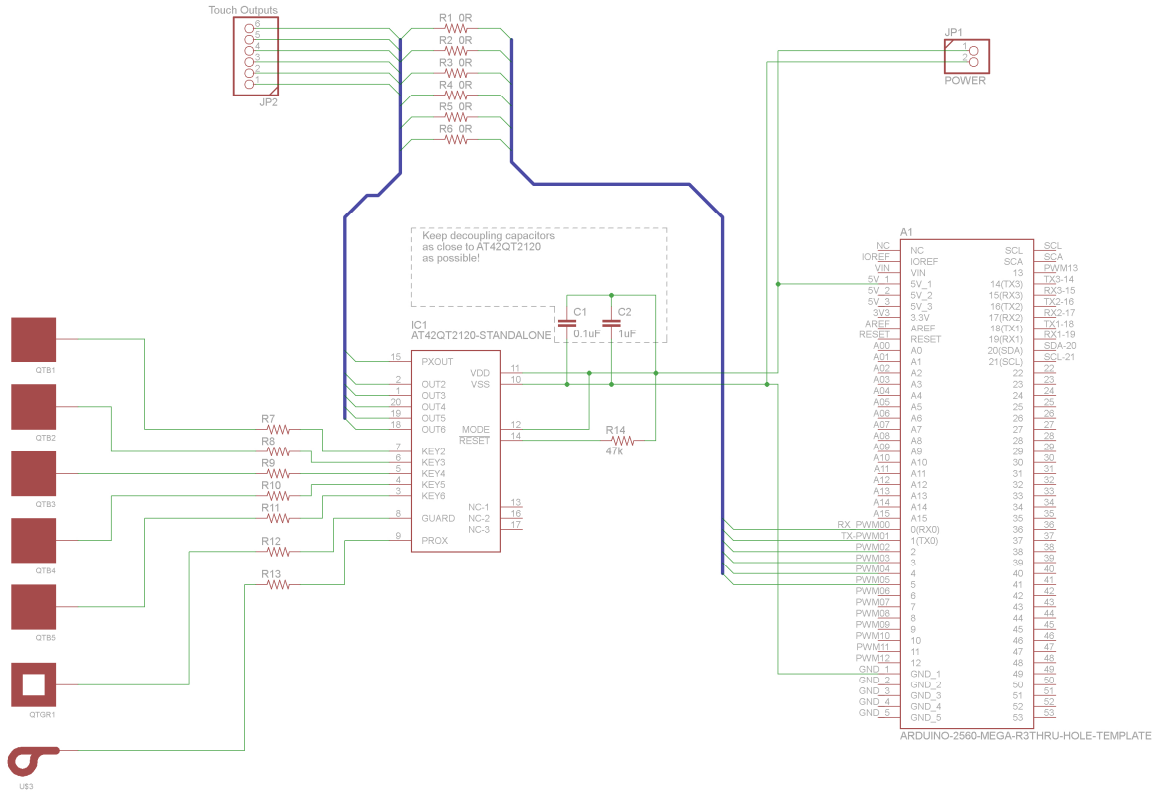


Figure 3 – Bloc-diagramme du contrôleur AT42QT2120

Le schéma du produit est relativement simple. La principale difficulté repose dans la conception du circuit imprimé. Pour obtenir une interface tactile capacitive robuste et fiable, il convient d'accorder beaucoup d'attention à la conception et au placement-routage des éléments bouton tactile, détection de proximité et anneau de garde sur la carte. La note d'application QTAN0079 d'Atmel (« Buttons, Sliders and Wheels Sensor Design Guide ») est une excellente source d'information. Pour éviter d'avoir à reconcevoir le circuit imprimé des plates-formes Arduino, il est possible d'utiliser les conceptions hardware open source, consistant en projets EagleCAD, comme base pour un Shield. Nous avons utilisé ici l'Arduino MEGA 2560, reprenant le placement-routage de la carte et enlevant tout, sauf les connecteurs traversants des bords de carte. Ce projet peut être réimporté dans EagleCAD en tant que nouveau composant, ce qui garantit l'exactitude du placement des broches et, si besoin, des trous. En exportant le projet à l'aide du plug-in open source EagleUp, il est possible de créer comme preuve de concept un tracé 3D dans l'outil SketchUp de Trimble. En exportant le résultat dans l'outil de rendering Maxwell, une vue photographique réaliste du design peut être créée à des fins de marketing.

Pour compléter le package de fournitures, il est important de fournir, au minimum, du logiciel pour la plate-forme Arduino, comme exemple de l'utilisation des fonctions du Shield. A partir de là, les étapes de création de bibliothèques ne sont pas très compliquées. Il existe un excellent tutorial[4] sur le site web d'Arduino qui décrit un processus simple permettant de passer d'un code applicatif opérationnel à une librairie complète intégrée dans l'IDE Arduino.

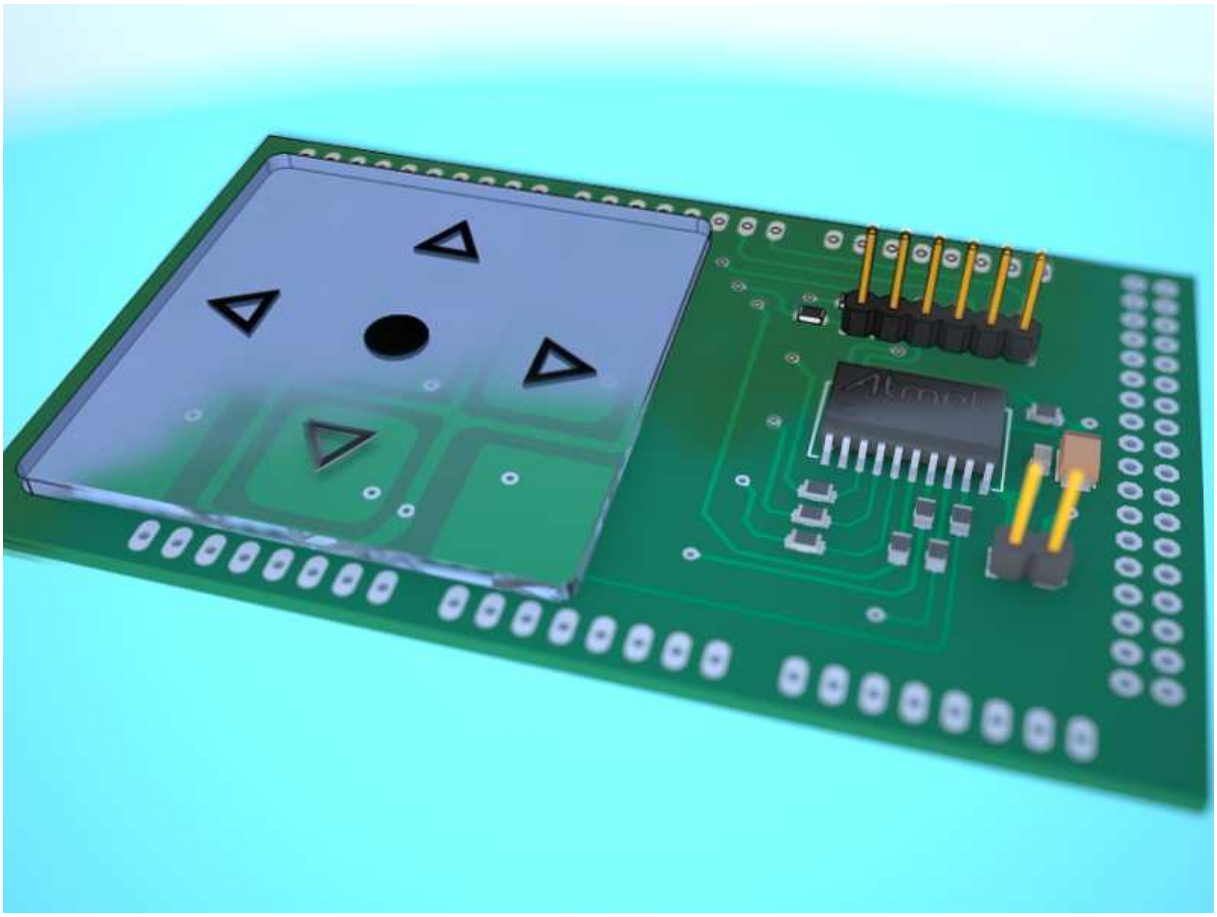


Figure 4 – Conception 3D du TickleShield avec superposition du modèle SketchUp au rendering Maxwell.

References

1) http://www.arduino.cc/en/Guide/Introduction
2) http://practicalmaker.com/blog/arduino-shield-design-standards
3) http://spaces.atmel.com/gf/project/tickleshield/
4) http://www.arduino.cc/en/Hacking/LibraryTutorial

L'auteur

Stuart Cording est directeur technique des ventes chez Atmel GmbH. Durant la dernière décennie, il a travaillé sur de nombreuses plates-formes DSP embarquées à microcontrôleur de différents fabricants. Il apporte avec dynamisme ses contributions à la communauté open source en utilisant github pour le partage de projets logiciels utiles dédiés aux systèmes embarqués. Originaire du Royaume-Uni, il est basé à Munich, où il vit avec sa famille.