prise en main du microcontrôleur ESP32-C3 RISC-V

Mathias Claußen (Elektor)

L'ESP32-C3 d'Espressif est arrivé. Alternative économique et monocœur à l'ESP8266, l'ESP32-C3 utilise l'architecture ouverte du jeu d'instructions RISC-V. Faisons le point…

Figure 1. Carte ESP32-C3 DevKitC-02.

L'ESP32-C3 d'Espressif était très attendu. Son fonctionnement n'est cadencé que par un seul cœur, contre deux habituellement pour les contrôleurs basés sur le SoC ESP32. Ce cœur utilise l'architecture libre et ouverte du jeu d'instructions RISC-V, en concurrence avec les contrôleurs à base d'ARM, largement utilisés pour les applications IoT. J'ai récemment jeté un coup d'œil à l'ESP32-C3 et à l'ESP32-C3-DevKitC-02 pour voir comment ce microcontrôleur se comportait en laboratoire. Passons en revue ce que j'ai pu découvrir.

Ceux qui me suivent sur Twitter (@ElektorMathias) [1] ont peut-être remarqué la présence de deux DevKits ESP32-C3 sur mon bureau. Les visiteurs qui ont consulté nos notes de laboratoire du mois de juin [2] savent également que la prise en charge logicielle de l'ESP32-C3 est toujours en cours. Le kit de développement objet de cet article utilise la révision 2 de la puce ESP32-C3. Ces versions préliminaires sont accompagnées d'une feuille d'errata A4 (imprimée recto verso) énumérant les problèmes identifiés à ce jour.

Entre autres, la révision 2 souffre d'« insomnie », c'est-à-dire qu'elle consomme beaucoup d'énergie en mode veille profonde, et l'adaptateur série USB/JTAG intégré à la puce ne fonctionne pas. La version plus récente de la révision 3 devrait avoir résolu ces problèmes, comme je l'ai noté sur Twitter [3].

Le référentiel ESP-IDF d'Espressif associé à l'ESP32-C3 est également en cours de mise en place et peut contenir des bogues. Étant donné que la prise en charge d'Arduino est basée sur cet environnement, les bogues éventuels migreront également sur cette plateforme. Au moment où j'écris cet article, la version de développement du package de prise en charge d'Arduino devra être installée dans l'EDI Arduino que vous utilisez. Pour ce faire, entrez le lien suivant sous *Préférences -> URL de gestionnaire de cartes supplémentaires :* `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json`, ce qui permet d'installer la version 2.0.0-rc1.

Les acheteurs de kits de développement basés sur l'ESP32-C3 devraient recevoir la dernière version (rev.3) de la puce, comme on peut le lire sur le forum d'ESP32.com [4]. La version préliminaire rev.2 problématique n'a été installée que sur un nombre limité des premiers kits de développement fournis.

Carte ESP32-C3 DevKitC-02

Tout d'abord, intéressons-nous à la puce ESP32-C3 et à la carte DevKitC-02 (**fig. 1**). La puce ESP32-C3 succède effectivement à l'ESP8266 d'Espressif. Comme l'ESP8266, elle utilise un processeur monocœur qui peut être cadencé jusqu'à 160 MHz et possède une puce de communication Wi-Fi BGN à 2,4 GHz avec mise en œuvre complète de la pile TCP/IP. Mais les similitudes avec l'ESP8266 s'arrêtent là. L'ESP32-C3 comprend beaucoup plus de périphériques associés à

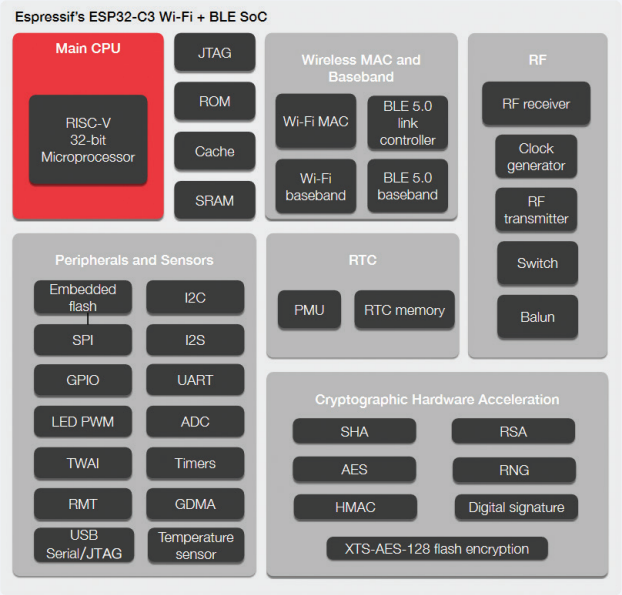

Figure 2. Schéma de principe de la carte ESP32-C3.

- CPU RISC-V cadencée à 160 MHz
- 400 Ko de SRAM (16 Ko de cache Flash)
- Wi-Fi (BGN) 2,4 GHz intégré
- Bluetooth LE 5.0
- Accélérateur matériel de chiffrage
- 22 GPIO programmables
- 2× CA/N SAR à 12 bits
- 3× SPI (supporte SPI, Dual SPI, Quad SPI et QPI)
- 2×x UART (supporte RS232, RS485 IrDA jusqu'à 5 MBd)
- 1× I²C (jusqu'à 800 kbit/s)
- 1× I²S RMT (commande à distance de périphériques)
- TWAI (compatible CAN 2.0 b / ISO 11898-1)
- PWM
- Adaptateur interne USB/JTAG

Tableau 1. Spécifications de l'ESP32-C3.

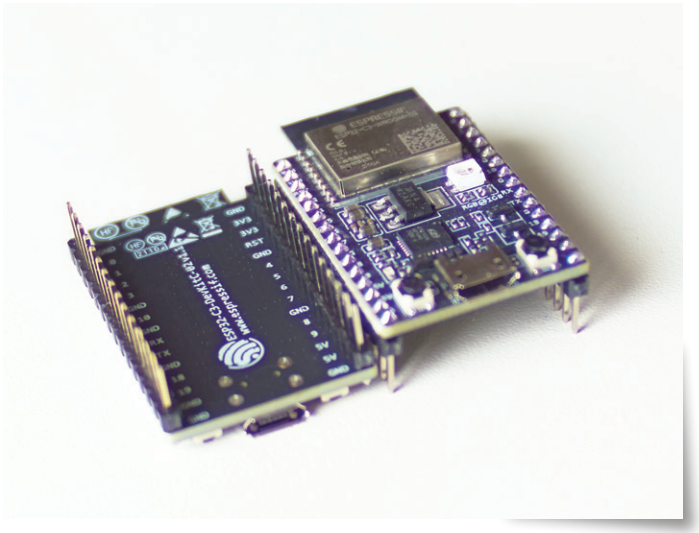

Figure 3. Les deux faces de la carte ESP32-C3-DevKitC-02.

l'ESP32, qui est le successeur de l'ESP8266 chez Espressif. En plus du Wi-Fi, l'ESP32-C3 inclut les communications BLE 5.0 et Bluetooth Mesh. Il utilise également la matrice GPIO particulièrement efficace, de sorte que presque toutes les fonctions peuvent être assignées à quasiment toutes les broches. La **figure 2** présente le schéma fonctionnel de l'ESP32-C3, qui comprend un adaptateur USB série/JTAG.

Un schéma de principe des fonctions du contrôleur est représenté sur la **figure 2** et montre clairement ce qu'il doit à l'ESP32, avec un aperçu des spécifications dans le **tableau 1**. Avec ses 384 Ko de mémoire RAM, l'ESP32-C3 offre presque cinq fois plus d'espace mémoire que l'ESP8266 (80 Ko). Le principal composant qui distingue l'ESP32-C3 de toutes les autres puces ESP32 ou ESP8266 est le cœur du processeur. Alors que ses prédécesseurs utilisent le processeur RISC Tensilica L106 ou LX6/LX7 [5], l'ESP32-C3 dispose d'une unité centrale RISC-V. Cela signifie que les compilateurs et autres programmes de la chaîne d'outils RISC-V peuvent être utilisés avec ce noyau. Les améliorations apportées à ces compilateurs et aux outils correspondants profiteront donc aux utilisateurs de l'ESP32-C3. Il n'est pas nécessaire que la communauté se lance dans la création d'une chaîne d'outils, comme ce fut le cas lors de l'introduction de l'ESP8266.

La carte DevKitC-02 comprend un convertisseur USB-série ainsi qu'une LED RVB compatible WS2812. Un schéma détaillé du circuit de la carte figure [6]. La **figure 3** montre les deux faces de la carte.

Autre routine de clignotement

La LED RVB compatible WS2812 installée sur la carte est commandée à l'aide d'un protocole de communication série. Nous ne pourrons donc pas nous en sortir simplement en configurant une broche d'E/S pour allumer la LED. Fort heureusement, la bibliothèque NeoPixel d'Adafruit contient des routines qui fonctionnent avec ce type de LED. Si elle n'est pas déjà installée, vous pouvez l'ajouter à votre EDI Arduino de la manière habituelle. Le code du **listage 1** fait clignoter la LED RVB en rouge. Vous n'avez pas besoin d'apporter de modifications particulières au code pour qu'il fonctionne sur l'ESP32-C3. Le téléchargement du code est tout aussi facile qu'avec un ESP32. Grâce à la matrice d'E/S, les LED WS2812 peuvent également être utilisées avec d'autres broches.

Lecture du type et de la version de la puce

Il est possible de lire les caractéristiques stockées dans les puces ESP32 elles-mêmes. Le **listage 2** affiche les informations lues via l'interface série à 115200 bauds (pour un débit à 9600 bauds, avec la révision 2 de la puce et le convertisseur USB-série intégré, les caractères sont brouillés). Vous pourrez y lire la version de la puce qui équipe votre kit ESP32-C3. Lorsque vous verrez la liste des bogues connus associés à la révision 2, vous serez heureux de pouvoir confirmer que la version de la puce installée n'est pas affectée.

Listage 1. Code du clignotement.

```
/* This sample needs the Adafruit NeoPixel library */
#include <Adafruit_NeoPixel.h>
 #define PIN        8
 #define NUMPIXELS  1
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);

void setup() {
     pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)
     pixels.clear(); // Set all pixel colors to 'off'
 }

void loop() {
     delay(1000);                          // wait for a second
         pixels.setPixelColor(0, pixels.Color(255, 0, 0));
         pixels.show();
     delay(1000);                          // wait for a second
         pixels.setPixelColor(0, pixels.Color(0, 0, 0));
         pixels.show();
 }
```

Portage de projets ESP32

Parmi les caractéristiques les plus utiles de l'ESP32-C3 figurent ses capacités de communication Wi-Fi et BLE intégrées. Associées aux systèmes de fichiers SPIFFS ou LittleFS pour gérer les pages web et d'autres données sur l'ESP32, ces fonctions de communication font de cette plateforme une solution idéale pour un large éventail d'applications Wi-Fi. Ceux qui travaillent avec une ancienne version 1.X de l'EDI Arduino devront installer un *plug-in* corrigé [7] pour télécharger des fichiers sur le système de fichiers de l'ESP32. La version originale de *me-no-dev* [8] ne fonctionne pas avec l'ESP32-C3.

À titre d'exemple, nous avons utilisé le code destiné à construire un serveur mini-NTP ESP32 [9] [10]. Il peut fournir l'heure via NTP dans son propre réseau et est écrit pour le *framework* Arduino. Après quelques ajustements dans l'affectation des broches, le code peut être compilé sans problème et téléchargé sur l'ESP32-C3. Cet exemple élémentaire montre qu'une grande partie du code existant et des connaissances de l'ESP32 sera directement portable sur l'ESP32-C3, de sorte que même les débutants en matière d'environnements RISC-V ne devraient pas trouver la transition trop ardue. La plupart des exemples ESP32 existants seront également opérationnels sur l'ESP32-C3, et comme FreeRTOS fonctionne en arrière-plan ici aussi, vous retrouverez tous ses avantages et inconvénients, comme avec un ESP32.

Comme l'ESP32-C3 n'est actuellement compatible qu'avec une branche de développement des puces ESP32 dans le *framework* Arduino, tous les EDI ne prennent pas encore pleinement en charge la puce. Au fur et à mesure de l'évolution du *framework*, il est probable que la prise en charge s'améliorera.

ESP32-C3 : une solution monocœur

L'ESP32-C3 est une alternative économique à l'ESP8266 et possède un grand nombre des périphériques de l'ESP32. L'adaptateur USB/série et JTAG intégré permet d'échanger facilement des fichiers et des données par un port USB. Vous pouvez même commencer à déboguer le code

LIENS

[1] @ElektorMathias sur Twitter : https://twitter.com/ElektorMathias/status/1386623477604626433
[2] Notes du labo d'Elektor, 06/2021 : https://www.elektormagazine.fr/news/compte-rendu-elektor-lab-juin-2021
[3] Tweet sur l'adaptateur USB-série / JTAG de l'ESP32-C3 : https://twitter.com/ElektorMathias/status/1392475706647584776
[4] Discussions du forum sur les révisions de l'ESP32-C3 : https://esp32.com/viewtopic.php?t=21040
[5] Page Wikipédia (en anglais) sur Cadence Tensilica : https://en.wikipedia.org/wiki/Tensilica
[6] Schémas de l'ESP32 C3 DevKitC-02 : https://dl.espressif.com/dl/schematics/SCH_ESP32-C3-DEVKITC-02_V1_1_20210126A.pdf
[7] Chargeur de fichiers dans ESP32 pour EDI Arduino (version corrigée) : https://github.com/lorol/arduino-esp32fs-plugin/releases
[8] Chargeur de fichiers dans ESP32 pour EDI Arduino : https://github.com/me-no-dev/arduino-esp32fs-plugin
[9] Dépôt Github – serveur mini-NTP avec GPS : https://github.com/ElektorLabs/180662-mini-NTP-ESP32
[10] Serveur mini-NTP avec GPS, Elektor Labs : https://www.elektormagazine.fr/labs/mini-ntp-server-with-gps
[11] Dépôt GitHub de cet article : https://bit.ly/3CD6iNs

(tant que rien d'imprévu ne se produit avec la révision 3 de la puce).

Grâce au *framework* Arduino, le code existant peut être réutilisé avec l'ESP32-C3 et le volume généreux de mémoire RAM et Flash permet d'envisager des projets plus conséquents. Concernant l'avenir, nous sommes sûrs de voir toute une série de cartes ESP32-C3 tierces commencer à apparaître dans les points de vente européens. Il sera intéressant de voir quelles nouvelles fonctions ces cartes offriront. Les possesseurs d'une carte ESP32-DevKitC-02 dans leur labo personnel seront bien placés pour commencer à écrire et tester immédiatement du code pour cet environnement. Les listages des croquis Arduino utilisés ici sont accessibles sur notre page GitHub [11].

210466-04

Listage 2. Lecture des informations relatives à la puce.

```
/* From the IDF documentation at https://github.com/espressif/esp-idf/components/esp_hw_support/include/
esp_chip_info.h */
 /*
  typedef enum {
     CHIP_ESP32  = 1, //!< ESP32
     CHIP_ESP32S2 = 2, //!< ESP32-S2
     CHIP_ESP32S3 = 4, //!< ESP32-S3
     CHIP_ESP32C3 = 5, //!< ESP32-C3
     CHIP_ESP32H2 = 6, //!< ESP32-H2
 } esp_chip_model_t;
// Chip feature flags, used in esp_chip_info_t
 #define CHIP_FEATURE_EMB_FLASH      BIT(0)      //!< Chip has embedded flash memory
 #define CHIP_FEATURE_WIFI_BGN       BIT(1)      //!< Chip has 2.4GHz WiFi
 #define CHIP_FEATURE_BLE            BIT(4)      //!< Chip has Bluetooth LE
 #define CHIP_FEATURE_BT             BIT(5)      //!< Chip has Bluetooth Classic
 #define CHIP_FEATURE_IEEE802154     BIT(6)      //!< Chip has IEEE 802.15.4
typedef struct {
     esp_chip_model_t model;  //!< chip model, one of esp_chip_model_t
     uint32_t features;       //!< bit mask of CHIP_FEATURE_x feature flags
     uint8_t cores;           //!< number of CPU cores
     uint8_t revision;        //!< chip revision number
 } esp_chip_info_t;
*/

void setup() {
     Serial.begin(115200);
 }

void loop() {
     delay(5000);
     Serial.println("esp_chip_info()");
     Serial.println("----------------------------------------------------------------");
     esp_chip_info_t info;
     esp_chip_info(&info);
     Serial.print("Chip Model: ");
     switch(info.model){
         case 1:{
             Serial.println("ESP32");
         }break;
        case 2:{
             Serial.println("ESP32-S2");
         }break;
```

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (mathias.claussen@elektor.com) ou contactez Elektor (redaction@elektor.fr).

Contributeurs

Texte et illustrations : **Mathias Claußen**
Rédaction : **Jens Nickel, C. J. Abate**
Mise en page : **Giel Dols**
Traduction : **Pascal Godart**

PRODUITS

- **Kit de développement ESP-C3-12F avec 4 Mo de mémoire Flash**
  www.elektor.fr/19855

```
        case 4:{
             Serial.println("ESP32-S3");
         }break;
        case 5:{
             Serial.println("ESP32-C3");
         }break;
        case 6:{
             Serial.println("ESP32-H2");
         } break;
        default:{
             Serial.print("Unknown Chipmodel");
             Serial.println(info.model);
         }
     }

    Serial.print("Featues :");
     if(info.features&CHIP_FEATURE_EMB_FLASH){
         Serial.print(" Embedded Flash ");
     }
     if(info.features&CHIP_FEATURE_WIFI_BGN){
         Serial.print(" WiFi(BGN) ");
     }
     if(info.features&CHIP_FEATURE_BLE){
         Serial.print(" BLE ");
     }

     if(info.features&CHIP_FEATURE_BT){
         Serial.print(" BT CLassic ");
     }
    if(info.features&CHIP_FEATURE_IEEE802154){
         Serial.print(" IEEE802.155.4 ");
     }
     Serial.println("");
     Serial.print("Cores: ");
     Serial.println(info.cores);
     Serial.print("Chip Revision: ");
     Serial.println(info.revision);
     Serial.println("-----------------------------------------------------------------");
     Serial.println();
}
```