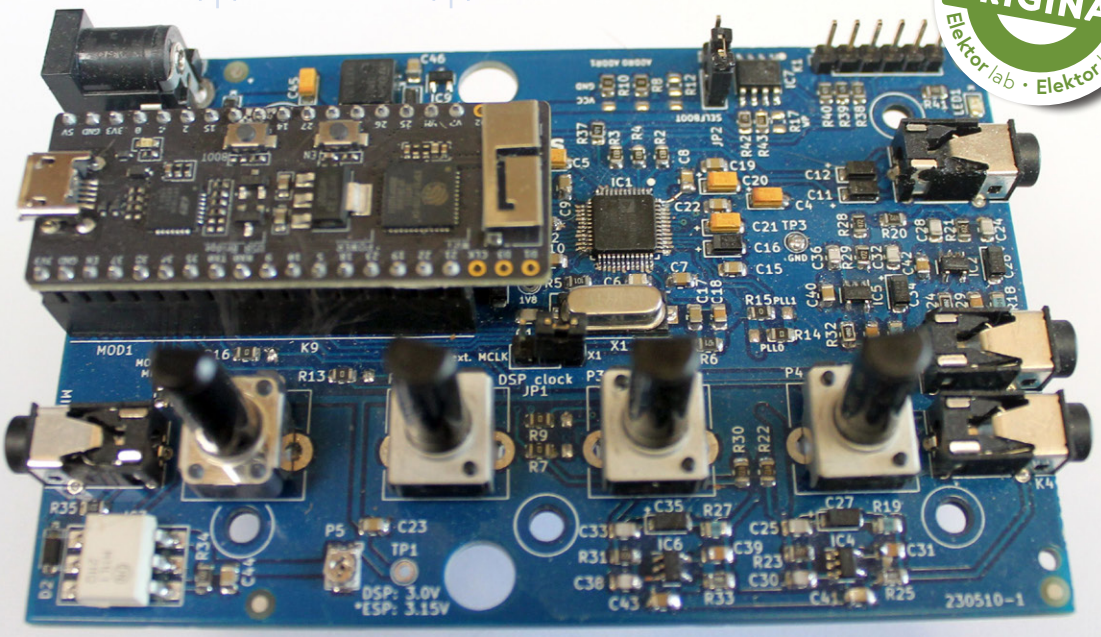


carte processeur
Audio DSP FX

partie 1 : caractéristiques et conception

Clemens Valens (Elektor)

La carte Elektor Audio DSP FX Processor est en réalité un microcontrôleur ESP32 complété par des entrées et sorties audio de haute qualité. Ce qui distingue néanmoins cette carte des autres cartes, apparemment similaires, est son interface audio qui intègre un DSP apte à traiter l'audio tout seul. Cette fonctionnalité pratique rend la carte non seulement puissante, mais aussi flexible et polyvalente. Dans ce premier article d'une série de deux, nous examinerons la conception de la carte et ses caractéristiques.

L'Audio DSP FX Processor combine un microcontrôleur Espressif ESP32 et un DSP audio ADAU1701 d'Analog Devices. La famille Audio DSP est une série de processeurs de signaux numériques (DSP) de haute performance, optimisés pour le traitement audio. Les programmes pour ces DSP sont créés avec l'outil de programmation graphique gratuit SigmaStudio en glissant et déposant des blocs d'algorithmes prédéfinis sur un canevas.

L'ADAU1701 utilisé ici n'est pas le dispositif le plus récent, mais il est le plus accessible. Outre un cœur DSP programmable par l'utilisateur, l'ADAU1701 intègre des convertisseurs analogique-numérique et numérique-analogique de haute qualité et dispose d'un port I²S. Il convient donc parfaitement comme interface audio pour l'ESP32.

Avant d'explorer les différents cas d'utilisation de la carte Audio DSP FX Processor, examinons d'abord son schéma. On dit qu'une image vaut mieux qu'un long discours. Jetez également un coup d'œil aux encadrés **Spécifications** et **Applications**. Cela vous aidera à comprendre la description du circuit qui suit. Une fois ceci fait, nous pouvons entrer dans le vif du sujet.

Aperçu du circuit

La **figure 1** présente un schéma fonctionnel de la carte Audio DSP FX Processor. On y voit deux blocs principaux, l'ESP32 et l'ADAU1701, connectés l'un à l'autre par plusieurs bus. Au centre, on trouve le bus I²S pour le transport des signaux audio entre les deux processeurs. Si le DSP se charge de tout le traitement audio, ce qu'il peut faire, ce bus n'est pas utilisé et l'ESP32 peut servir à d'autres tâches.

Spécifications

- ADAU1701 Processeur audio numérique 28/56 bits, 50 MIPS prenant en charge des taux d'échantillonnage allant jusqu'à 192 kHz
- ESP32 Microcontrôleur double cœur 32 bits avec Wi-Fi 802.11b/g/n et Bluetooth 4.2 BR/EDR et BLE
- 2× entrées audio 24 bits (2 VRMS, 20 kΩ)
- 4× sorties audio 24 bits (0,9 VRMS, 600 Ω)
- 4× potentiomètre de réglage
- Entrée et sortie MIDI
- Port d'expansion I²C
- Fonctionnement multimode
- Alimentation : 5 VCC USB ou 7,5 VDC à 12 VDC (jack, la broche centrale est GND)
- Consommation de courant (moyenne) : 200 mA

Applications

- Récepteur Bluetooth/Wi-Fi audio (par ex. haut-parleur) et émetteur
- Pédale d'effet guitare (pédale de percussion)
- Synthétiseur musical
- Générateur de sons/fonctions
- Filtre cross-over programmable pour haut-parleurs
- Processeur d'effets audios avancé (réverbération, chorus, transposition de tonalité, etc.)
- Appareil audio connecté à Internet
- Plate-forme d'expérimentation DSP
- MIDI sans fil
- Convertisseur MIDI vers CV
- et bien d'autres choses encore !

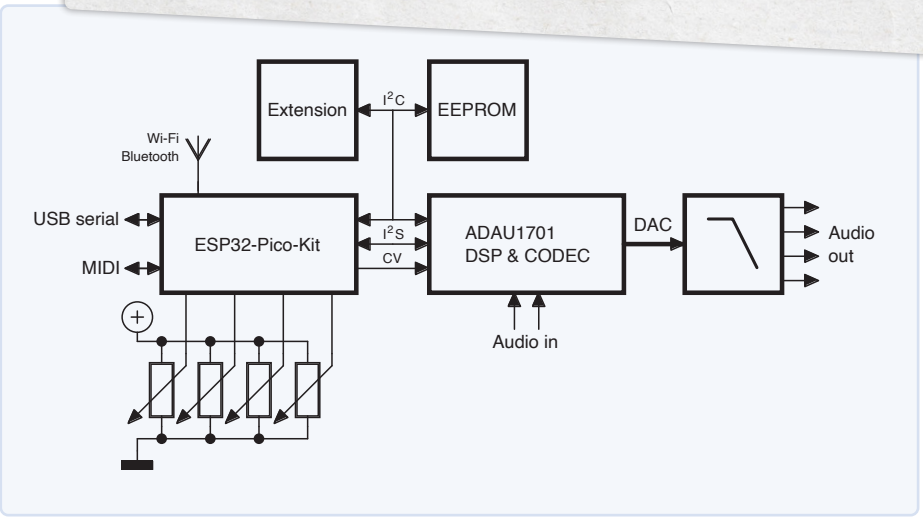


Figure 1. Le schéma fonctionnel du processeur Audio DSP FX contient deux blocs principaux : un processeur ESP32 et un DSP.

Le bus I²C permet de commander le DSP à partir d'un autre processeur comme l'ESP32 ou un Arduino UNO connecté au port d'extension. Le DSP peut également utiliser le bus SPI, mais nous avons choisi le bus I²C parce qu'il est plus facile à étendre qu'un bus SPI (aucun signal de sélection de cible n'est nécessaire).

Ce bus est également disponible pour lire ou écrire dans l'EEPROM. Par exemple, lorsque l'ADAU1701 est en mode *Selfboot*, il s'attend à ce que son programme soit stocké dans cette EEPROM. L'ESP32 peut charger l'EEPROM avec un programme DSP reçu d'un autre ordinateur, sans fil ou par USB.

Le quatrième bus va de l'ESP32 au DSP et est destiné aux signaux analogiques qui servent au réglage des paramètres des blocs algorithmiques au sein du DSP. Ce bus possède quatre canaux. Nous avons fait en sorte de pouvoir ajuster ces tensions analogiques par des potentiomètres (qui peuvent aussi servir à d'autres fins) ou par un micrologiciel fonctionnant sur l'ESP32.

Le bus d'E/S audio (tout à droite) se compose de deux canaux d'entrée et de quatre canaux de sortie. Les sorties sont filtrées par des filtres passifs ou actifs (voir ci-dessous), les entrées sont filtrées dans le DSP. Outre les signaux mentionnés jusqu'à présent, l'ESP32 dispose également d'un convertisseur USB-série, d'une entrée MIDI (optiquement isolée) et d'une sortie MIDI.

Enfin, l'ESP32 dispose d'une radio pour les communications Wi-Fi et Bluetooth, représentée par un symbole d'antenne.

Circuit

La **figure 2** présente le schéma détaillé de la carte Audio DSP FX Processor. IC1 est le DSP audio ADAU1701. Le circuit qui l'entoure est pratiquement un copier-coller de la fiche technique du DSP, à l'exception du cavalier JP1 qui permet de sélectionner la source d'horloge pour le DSP.

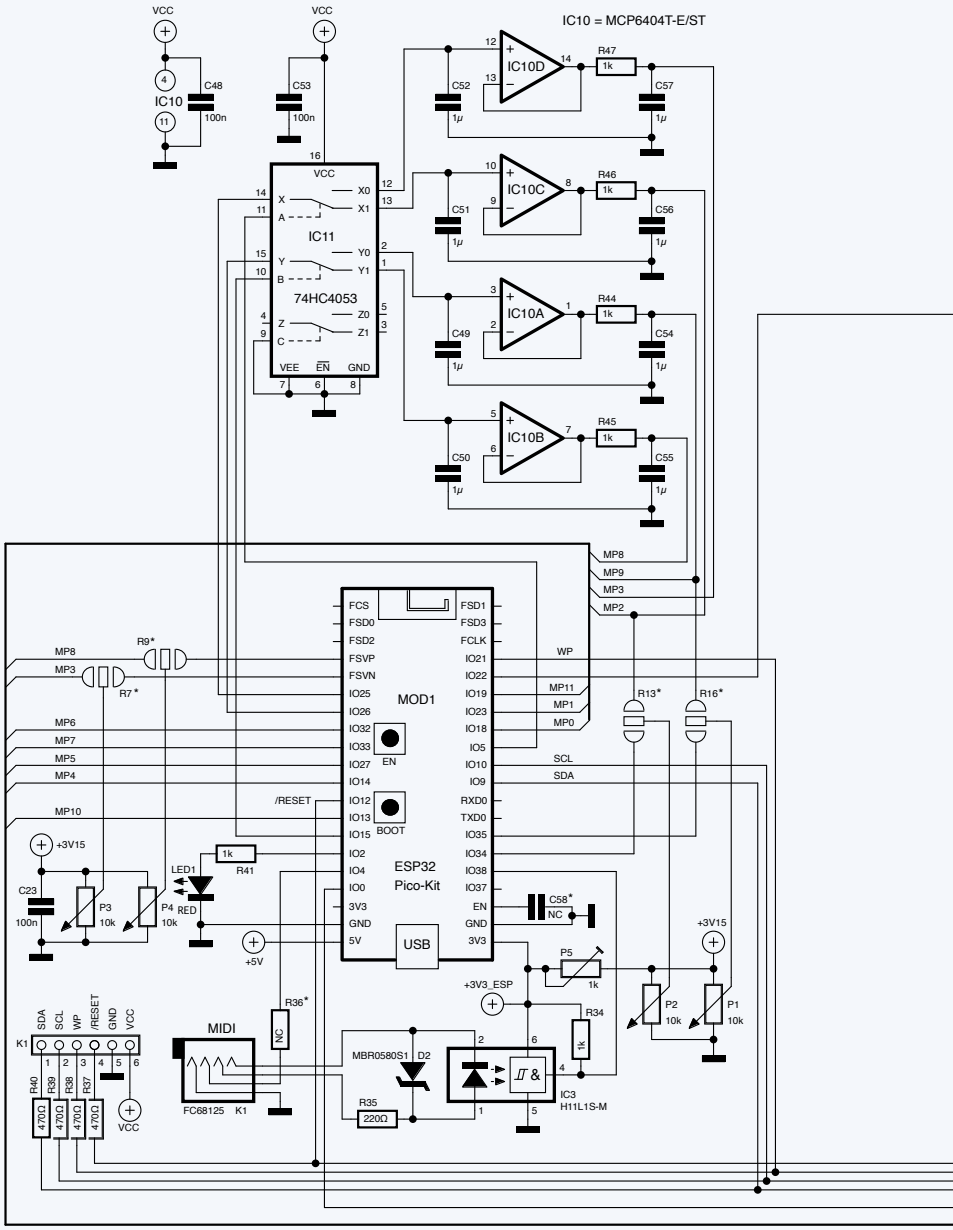
Les filtres de sortie audio (IC2, IC4, IC5 et IC6, voir **figure 3**) sont implémentés comme quatre canaux complètement séparés sans aucun partage d'ampli-op entre les canaux. Non seulement cela simplifie le dessin du circuit imprimé, mais cela permet également de ne monter que les composants nécessaires pour une configuration de sortie audio donnée (c'est-à-dire 2 canaux avec des filtres passifs ou 4 canaux complets avec des filtres actifs. Notez que la différence de prix est minime).

Les quatre signaux analogiques qui servent au réglage des paramètres à l'intérieur du DSP sont connectés à MP9, MP2, MP3 et MP8 (ADC0 à ADC3, dans cet ordre). Selon la position d'un cavalier à souder (R16, R13, R7 et R9), le signal peut être un potentiomètre ou provenir du multiplexeur analogique (IC10 et IC11). Ce dernier est nécessaire car l'ESP32 n'a que deux sorties analogiques. Si un potentiomètre est acheminé vers le DSP, veillez à supprimer la résistance en série du multiplexeur correspondant (R44, R46, R47, R45). Dans ce cas, réglez également P5 de sorte que le côté chaud des potentiomètres soit à 3 V. Lorsque tous les potentiomètres sont connectés à l'ESP32, réglez P5 pour obtenir 3,15 V, afin d'améliorer légèrement leur plage dynamique au sein de l'ESP32.

Le bus I²S peut être un peu un casse-tête jusqu'à ce que vous le compreniez. L'ADAU1701 prend en charge les modes meneur et suiveur I²S, tout comme l'ESP32. Alors, qui sera meneur et qui sera suiveur ? Au moment du développement de ce projet, on n'avait pas le choix car les bibliothèques logicielles de l'ESP32 ne prenaient pas (encore) en charge le mode I²S suiveur. L'ESP32 devait donc être le meneur. Problème résolu ? Presque.

Un détail subtil (non documenté ?) de l'ADAU1701 est qu'il exige que les signaux d'horloge I²S soient synchronisés avec son horloge principale.

Si ce n'est pas le cas, il coupe les sorties audios, ce qui n'est pas du tout souhaitable. La seule façon d'assurer la synchronisation lorsque le DSP est suiveur I²S, est de le cadencer à partir de l'ESP32. C'est là que JP1 entre en jeu. Il permet de choisir la source d'horloge du DSP : le cristal X1 ou le signal MCLK de l'ESP32. Par conséquent, lorsque le bus I²S doit être utilisé, veillez à court-circuiter les broches 2 et 3 de JP1. Si le bus I²S n'est pas nécessaire, court-circuiter les broches 1 et 2 de JP1.
L'ESP32 se présente sous la forme d'un ESP32-PICO-KIT, un module facile à utiliser qui intègre un convertisseur USB-série et une alimentation de 3,3 V. L'utilisation d'un module au lieu de monter l'ESP32 directement sur la carte principale permet d'éviter de nombreux problèmes d'adaptation d'antenne et d'autres problèmes de radiofréquence, et permet de ne monter le module qu'en cas de besoin. C58 n'est nécessaire que dans les rares cas où le module ESP32 ne démarre pas lorsqu'il n'est pas connecté à l'USB. S'il vous le faut, c'est que vous utilisez probablement un très vieux module.
Le DSP est alimenté par IC9 qui, à son tour, est alimenté soit par le port USB, soit par IC8. Le lecteur attentif se demandera pourquoi il n'y a pas de diode de protection comme D3 dans l'alimentation 5 V provenant du module ESP32. C'est parce que cette diode est déjà présente sur le module ESP32.
Concernant l'alimentation, T1 est nécessaire pour le régulateur de tension interne de 1,8 V du DSP. Comme la fiche technique du DSP suggère de l'utiliser, nous l'avons fait.
Les programmes pour le DSP peuvent être stockés dans IC7, l'EEPROM (appelée *E2Prom* dans SigmaStudio). Elle fournit beaucoup plus d'espace mémoire que ce qui est nécessaire pour un programme DSP, de sorte que l'ESP32 peut également l'utiliser pour stocker d'autres données. Elle est également accessible depuis le connecteur K1 via le bus I²C, et peut donc être lue et écrite par un dispositif externe tel qu'un Arduino UNO ou un Raspberry Pi Pico.
Un mot sur l'interface MIDI. Il s'agit d'une entrée isolée optiquement et d'une sortie non isolée, mais connectée à une prise jack 3,5 mm à 4 voies (TRRS, *tip-ring-ring-sleeve*) en raison des contraintes d'espace de la carte. La plupart des personnes intéressées par le MIDI n'auront besoin que de l'entrée et utiliseront un câble TRS. Un tel câble court-circuitera la sortie MIDI (broche R2 de K5) à la masse, c'est pourquoi R36 n'est pas montée par défaut. Montez R36 (82 Ω) pour activer la sortie et assurez-vous d'utiliser un câble approprié.
Programmable par l'utilisateur, la LED1 peut servir, entre autres, au débogage de logiciels ou comme indicateur de mode.
La consommation de courant dépend bien sûr fortement de l'application, mais elle est en moyenne de l'ordre de 200 mA pour une tension d'alimentation de 9 V_{CC}.



Une carte multifonction

Maintenant que vous avez une vague idée de ce qu'est la carte Audio DSP FX Processor, explorons plus en détail la spécification de fonctionnement multimode. Comme son nom l'indique, cela signifie que la carte peut être utilisée dans différentes configurations :

- DSP uniquement
- DSP avec programmateur EEPROM intégré
- DSP avec OTA
- DSP comme CNA I²S
- Synthétiseur musical / Générateur de sons
- Processeur d'effets audio (FX) avancé
- Autres

DSP uniquement

Dans cette configuration (**figure 4)**, la carte est identique à la carte

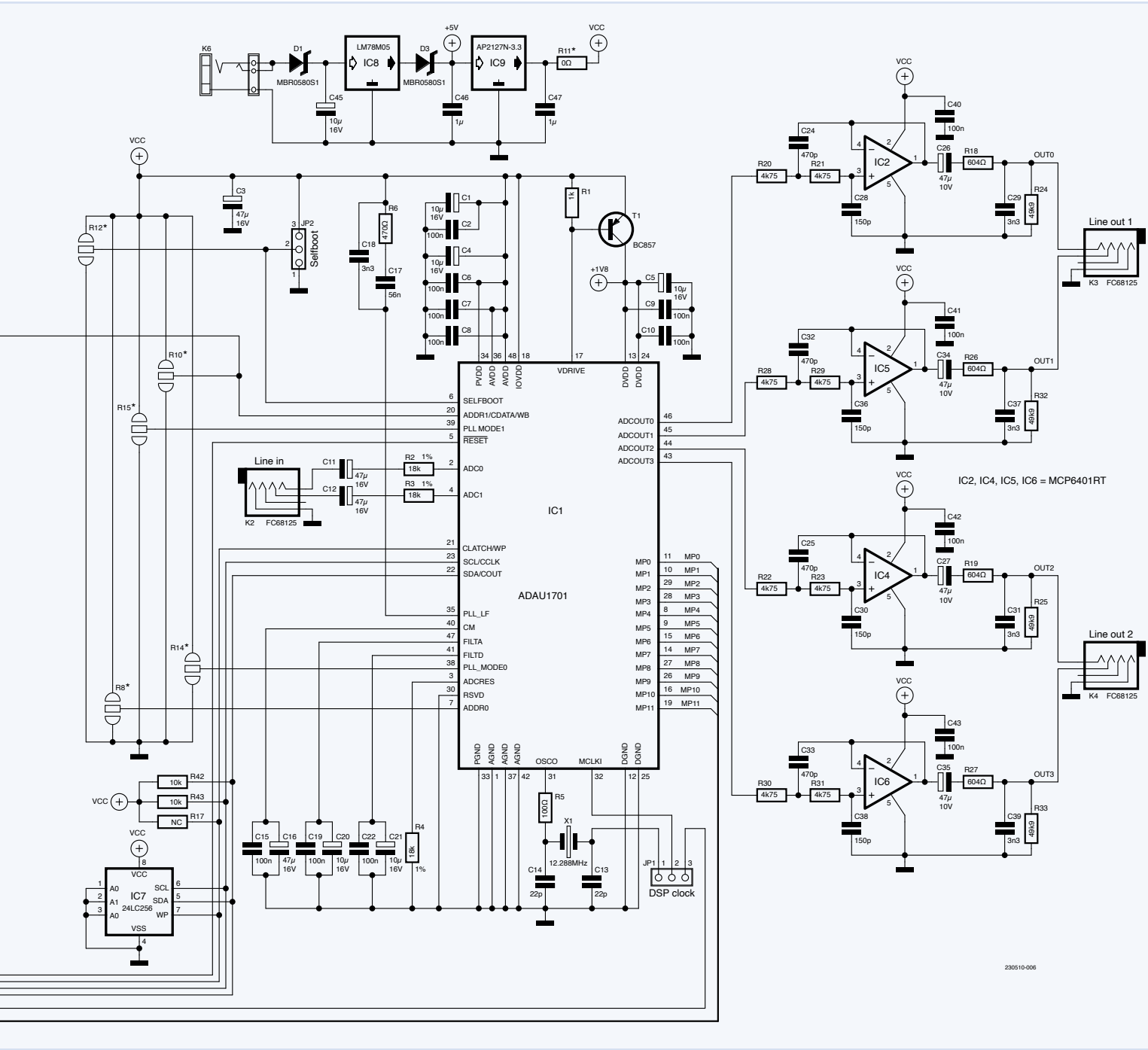


Figure 2. Schéma du processeur audio DSP FX. Le tout tient sur une carte d'environ 102 mm par 66 mm.

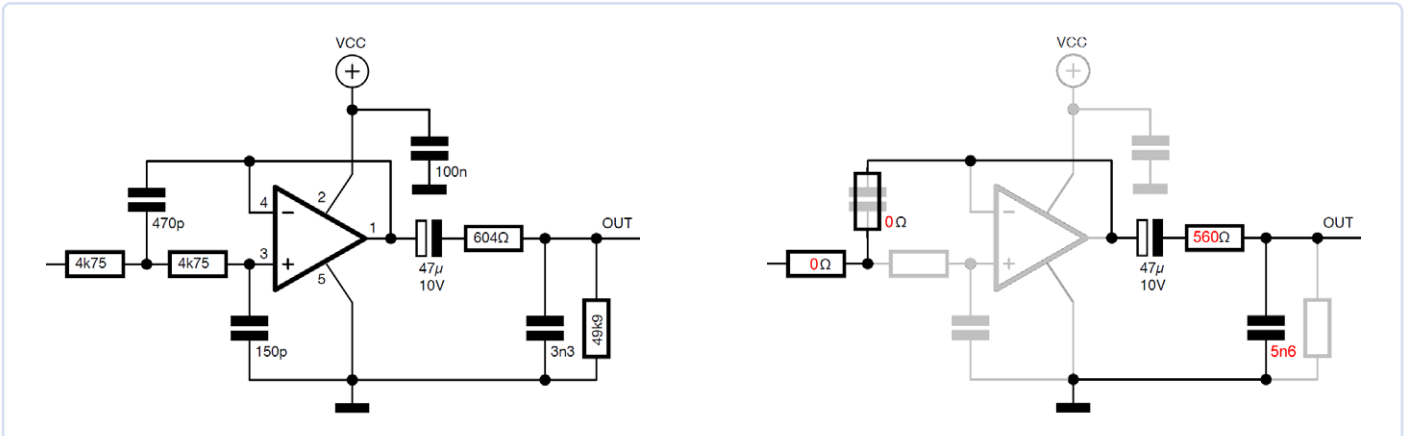


Figure 3. Les filtres de sortie actifs et passifs sont pris en charge par le processeur audio DSP FX.

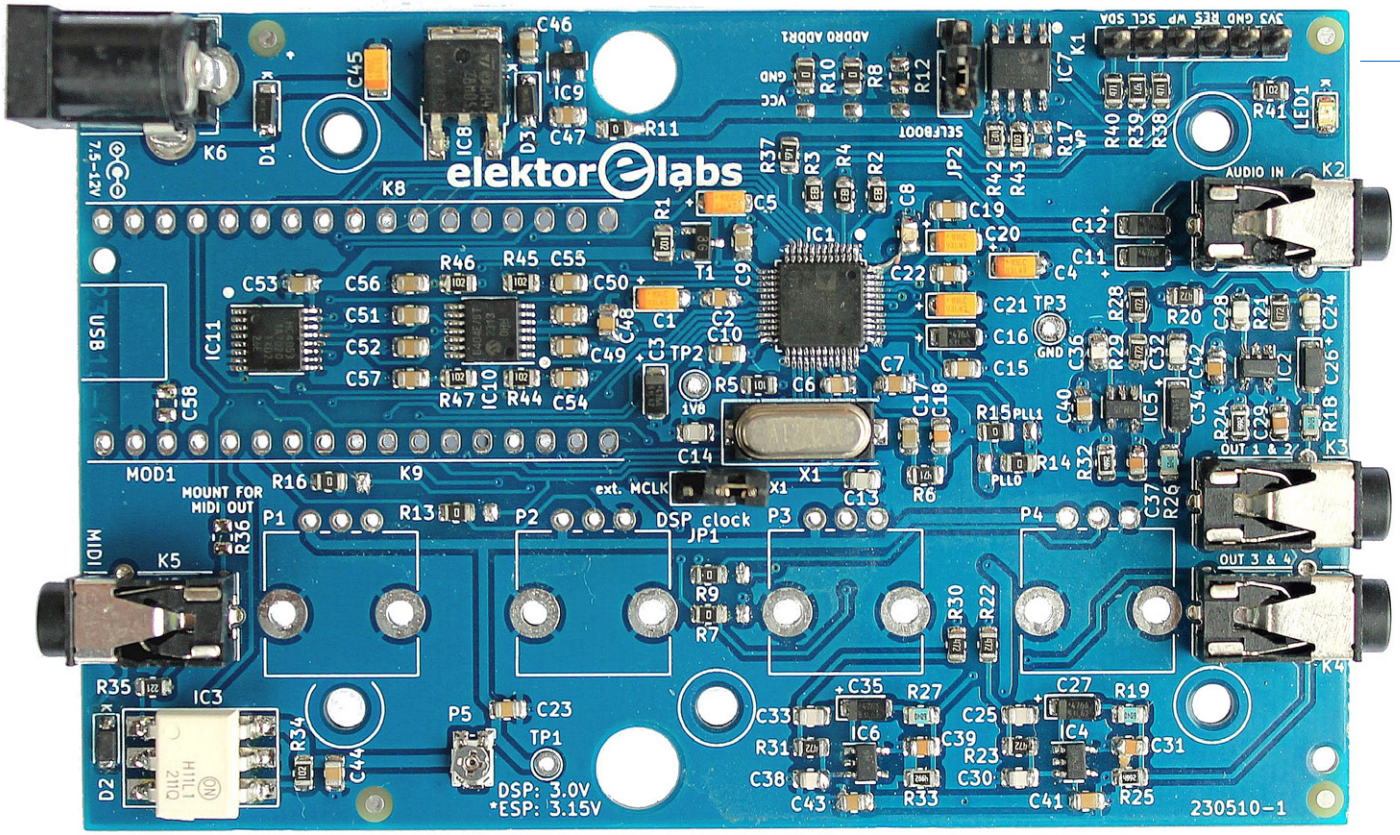


Figure 4. La carte configurée pour un fonctionnement autonome du DSP sans potentiomètres.

ADAU1701 Universal Audio DSP publiée en janvier 2014 (projet Elektor 130232). Les programmes pour le DSP sont créés dans SigmaStudio 4.7, exportés sous forme de fichier HEX, et chargés dans l'EEPROM via le connecteur K1 avec par exemple un Arduino UNO [1]. Placez le cavalier JP1 en position *X1* (sur les broches 1 et 2, l'horloge du DSP est fournie par le cristal X1) et JP2 en position *Selfboot* (broches 2 et 3, indiquant au DSP de charger son programme à partir de l'EEPROM). Les cavaliers R7, R9, R13, R16 doivent être positionnés à droite, c'est-à-dire à l'opposé de leur nom sérigraphié sur la carte, pour permettre la commande du potentiomètre. Cette configuration permet d'ajouter des effets audio (réverbération, filtrage, etc.) au signal d'entrée. Les applications les plus courantes sont les filtres croisés pour haut-parleurs, rendus possibles par les quatre sorties audios indépendantes, et les effets pour guitare.

DSP avec programmateur EEPROM intégré

Comme le DSP seul, mais avec l'ESP32-Pico-Kit ajouté à la carte pour simplifier la programmation de l'EEPROM. Les programmes DSP peuvent maintenant être téléchargés par USB vers l'ESP32, qui les grave dans l'EEPROM. Outre la simplification de la programmation de l'EEPROM, cette configuration permet également de stocker plus d'un programme DSP. Par exemple, un serveur web fonctionnant sur l'ESP32 peut fournir une interface utilisateur permettant de choisir entre différents algorithmes DSP.

DSP avec OTA

Comme pour le DSP avec l'ESP32-Pico-Kit, mais dans ce cas, l'ESP32 fournit une connexion Wi-Fi à SigmaStudio pour télécharger les algorithmes du DSP *Over the Air* (OTA), sans exporter de fichiers HEX. C'est la façon la plus simple de programmer le DSP dans SigmaStudio, mais les programmes ne sont pas stockés dans l'EEPROM. Cette configuration est pratique pour développer et tester de nouveaux algorithmes SigmaStudio à la volée. Placez JP2 sur les broches 1 et 2 pour indiquer au DSP de ne pas charger son programme à partir de l'EEPROM.

DSP comme DAC I²S

Dans cette configuration, l'ESP32-Pico-Kit est la source audio tandis que le DSP agit comme un convertisseur numérique-analogique (CNA) de haute qualité. Les données audios sont envoyées par I²S. Placer le cavalier JP1 en position *ext. MCLK* (sur les broches 2 et 3, l'horloge du DSP est fournie par l'ESP32). Pour plus de simplicité, placez JP2 en position *Selfboot* (sur les broches 2 et 3, le DSP charge son programme depuis l'EEPROM) et programmez le DSP avec un simple programme de transfert par I²S (inclus dans les téléchargements). Cette configuration convient aux lecteurs audios sans fil (Wi-Fi ou Bluetooth).

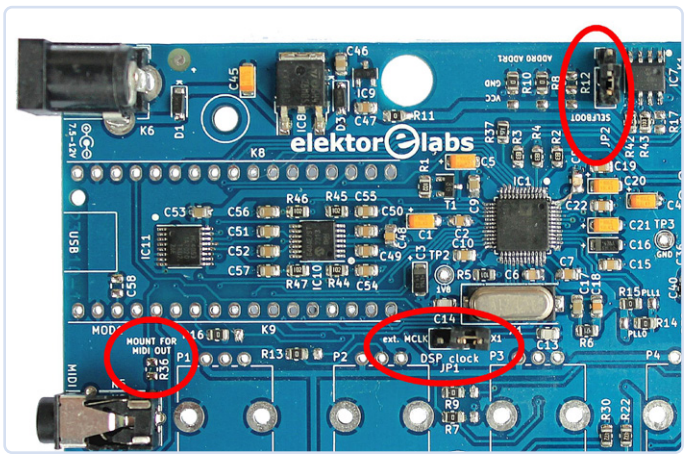


Figure 5. L'emplacement des cavaliers JP1 (au milieu) et JP2 (en haut). Dans le coin inférieur gauche, vous trouverez R36 qui active la sortie MIDI.

Liste des composants

Résistances (0805, 5%, 0.125 W)

R7, R8, R9, R10, R11, R13, R14, R15, R16 = 0 Ω
R1, R34, R41, R44, R45, R46, R47 = 1 kΩ 1%
R2, R3, R4 = 18 kΩ 1%
R5 = 100 Ω
R6, R37, R38, R39, R40 = 470 Ω
R18, R19, R26, R27 = 604 Ω 1%
R20, R21, R22, R23, R28, R29, R30, R31 = 4.75 kΩ 1%
R24, R25, R32, R33 = 49.9 kΩ 1%
R35 = 220 Ω
R42, R43 = 10 kΩ
P1, P2, P3, P4 = potentiomètres linéaires verticaux 10 kΩ
P5 = potentiomètre 1 kΩ
R12, R17, R36 = NC

Condensateurs (0805)

C1, C4, C5, C20, C21, C45 = 10 µF 16 V
C2, C6, C7, C8, C9, C10, C15, C19, C22, C23, C40, C41, C42, C43, C44, C48, C53 = 100 nF
C3, C11, C12, C16, C26, C27, C34, C35 = 47 µF 10 V
C13, C14 = 22 pF
C17 = 56 nF
C18, C29, C31, C37, C39 = 3.3 nF
C24, C25, C32, C33 = 470 pF
C28, C30, C36, C38 = 150 pF
C46, C47, C49, C50, C51, C52, C54, C55, C56, C57 = 1 µF
C58 = NC (100 nF si vous en avez vraiment besoin)

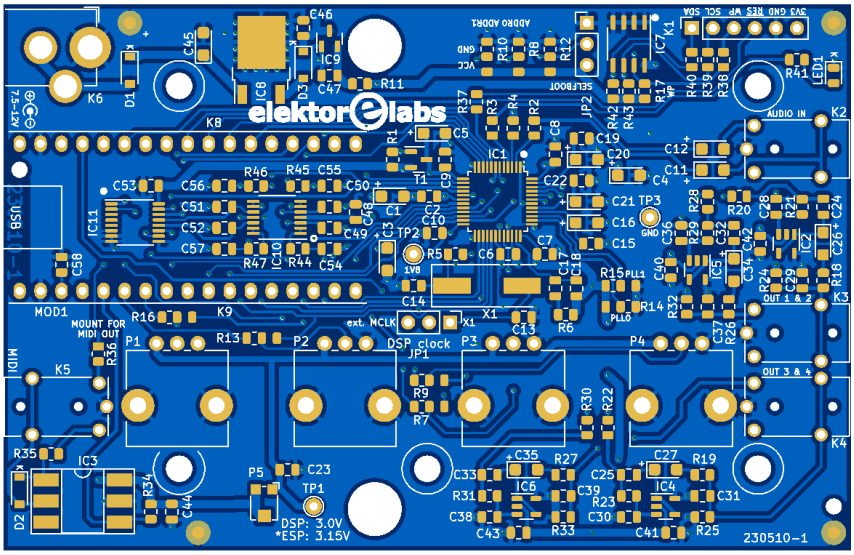
Semi-conducteurs

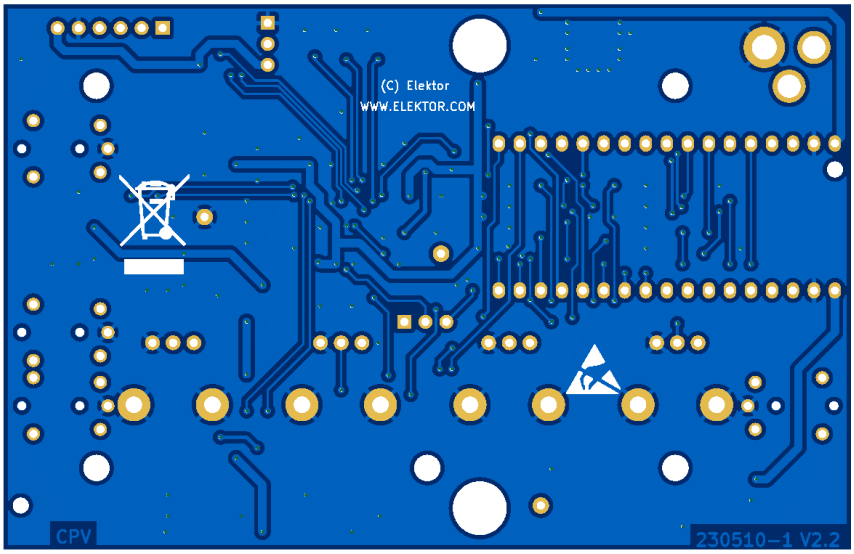
D1, D2, D3 = MBR0580S1
IC1 = ADAU1701JSTZ
IC2, IC4, IC5, IC6 = MCP6401RT-E/OT
IC3 = H11L1S-M
IC7 = 24LC256
IC8 = LM78M05
IC9 = AP2127N-3.3
IC10 = MCP6404T-E/ST
IC11 = 74HC4053PW
LED1 = LED, rouge, 0805
T1 = BC857
MOD1 = Espressif ESP32-PICO-KIT

Divers

JP1, JP2 = barrettes à 3 voies, pas de 0,1» + cavalier
K1 = barrette à 6 voies, pas de 0,1»
K2, K3, K4, K5 = Jack audio, 4 pôles (TRRS)
K6 = Jack
K8, K9 = embase à 17 voies, pas de 0,1»
X1 = 12,288 MHz, 20 pF

Circuit imprimé = Elektor 230510-1





Des effets sonores comme la réverbération ou la transposition de tonalité peuvent être ajoutés facilement en créant des algorithmes appropriés dans SigmaStudio et en les chargeant dans l'EEPROM du DSP. Les quatre potentiomètres sont disponibles pour le réglage des paramètres (veillez à ce que les résistances de pontage soient positionnées correctement). L'entrée audio analogique peut, bien entendu, être utilisée pour des applications de chant, karaoké ou autres mixages audio.

Synthétiseur musical / Générateur de sons

Cette configuration est pratiquement la même que la précédente, sauf que l'ESP32 se charge de l'essentiel (voire de la totalité) du traitement du signal ; le DSP fonctionne comme un CNA audio I²S fantaisiste. L'ESP32 est le moteur (sonore) qui crée des sons à partir de rien en utilisant des algorithmes définis par l'utilisateur. Plusieurs bibliothèques de synthèse sonore sont disponibles en ligne, ce qui en fait un domaine intéressant à explorer. L'entrée MIDI vous permet de jouer le synthétiseur avec un clavier MIDI et de le commander avec, par exemple, des messages MIDI CC (*Control Change* ou *Continuous Controller*). Bien entendu, la commande sans fil est également possible, tout comme

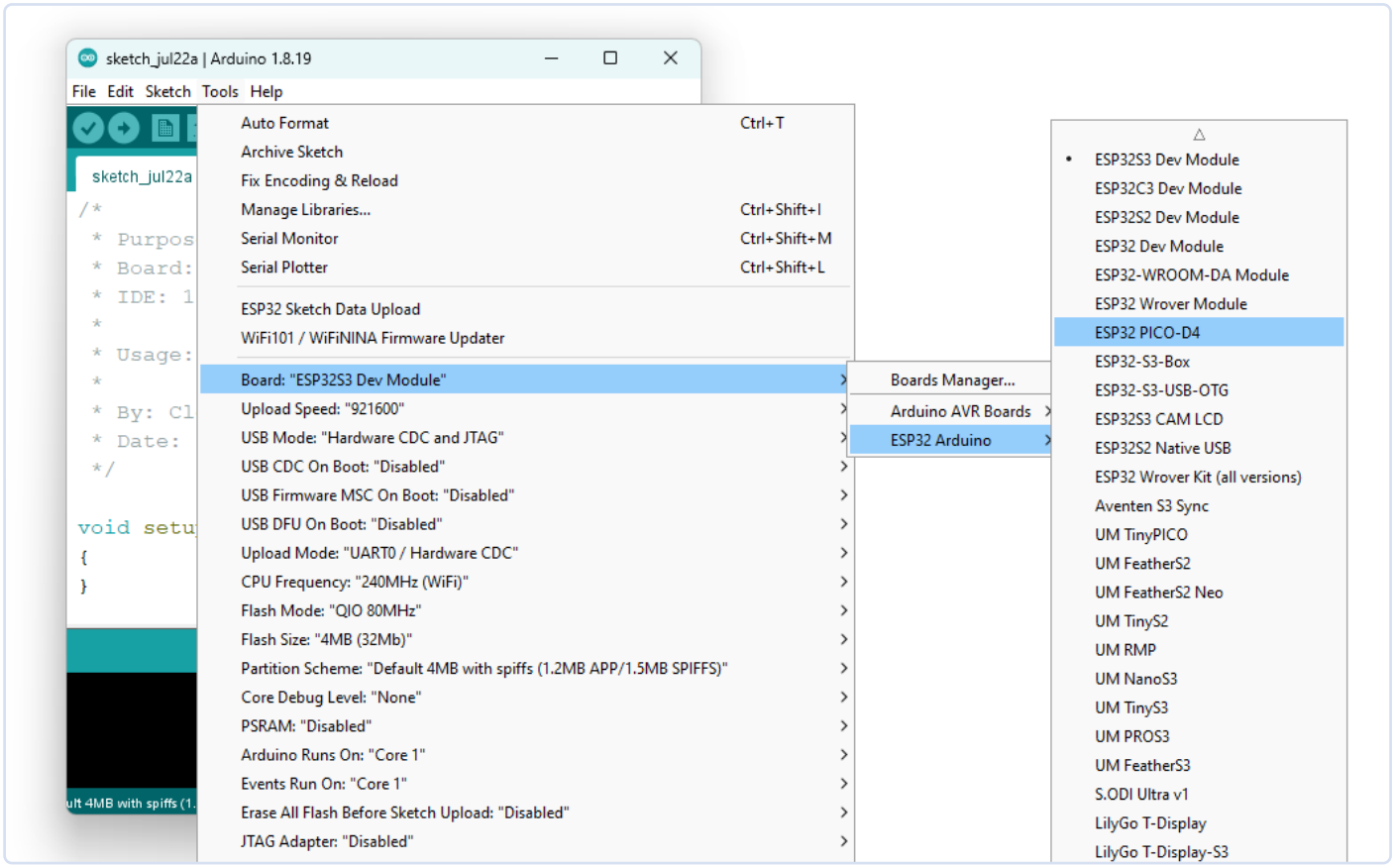


Figure 6. Le processeur audio DSP FX utilise l'ESP32 Pico-D4 comme «carte» dans l'EDI Arduino.

par liaison série. Les quatre potentiomètres offrent encore plus de possibilités de commande. Pour s'amuser encore plus, le DSP peut ajouter ses propres effets sonores.

Processeur d'effets audio (FX) avancé

Comme précédemment, mais au lieu d'utiliser l'ESP32 comme source sonore, il est maintenant utilisé comme processeur de signal numérique. L'ADAU1701 fournit les entrées et sorties audios. Il envoie le signal audio numérique par I²S à l'ESP32 où il est traité. Le résultat est renvoyé à l'ADAU1701, toujours par I²S. L'ADAU1701 se charge de la conversion numérique-analogique et émet jusqu'à quatre signaux audio analogiques. L'ADAU1701 peut bien sûr ajouter son propre (pré)traitement. Les paramètres des effets peuvent être commandés sans fil, en série, par MIDI et par les potentiomètres, et même par I²C si on le souhaite.

Autres

Cette configuration est destinée aux personnes qui ont acheté la carte mais sans avoir besoin de traitement audio. Placez JP1 en position *ext. MCLK* (broches 2 et 3) et JP2 sur les broches 1 et 2 (pas *Selfboot*) pour que le DSP ne consomme pas beaucoup d'énergie. Le module ESP32 peut être utilisé comme un serveur web, un pont MIDI USB-série, un pont USB-série I²C, un convertisseur MIDI-I²C, un convertisseur MIDI-analogique et vice-versa, un convertisseur analogique-numérique à 4 canaux, un capteur IoT ou un clignotant à LED. On peut dire que seule votre propre créativité vous limite.

Réglage des paramètres

Les effets audios comportent généralement un ou plusieurs paramètres réglables par l'utilisateur pour ajuster le son ou l'effet en temps réel. Le processeur audio DSP FX propose plusieurs méthodes pour ce faire, dont la plupart peuvent être utilisées simultanément.

I²C

Quelle que soit la configuration utilisée, le DSP peut toujours être commandé par I²C, soit par K1, soit par l'ESP32. Ce dernier étant connecté au même bus, il peut également être commandé par I²C si vous le souhaitez.

Potentiomètres

On peut équiper le processeur audio DSP FX de quatre potentiomètres. Ceux-ci peuvent être connectés soit au DSP, soit à l'ESP32. Lorsqu'ils sont connectés au DSP, ils peuvent être lus par les CAN auxiliaires de SigmaStudio (ADC1 à ADC4 dans le *bloc GPIO* de l'onglet *Register Control* de l'onglet *Hardware Configuration d'*un projet). Dans ce cas, enlevez R44 à R47.

Une méthode préférable, plus souple, consiste toutefois à connecter les potentiomètres à l'ESP32. L'appareil possède deux sorties analogiques qui sont transformées en quatre par multiplexage (IC10 et IC11) et qui sont connectées aux CAN auxiliaires du DSP (broches MP9, MP2, MP3 et MP8). Cela permet à l'ESP32 de commander les CAN auxiliaires du DSP par les potentiomètres ou d'une autre manière (par exemple,

messages CC MIDI ou sans fil).
Notez que les sorties analogiques multiplexées peuvent également être utilisées comme sorties numériques (lentes), qu'on peut traiter par un algorithme DSP comme des interrupteurs ou des boutons-poussoirs.

MIDI

L'entrée optiquement isolée de la carte (IC3) est destinée à l'entrée MIDI (mais on peut bien sûr aussi l'utiliser pour autre chose). Pour des raisons d'encombrement, le connecteur est un jack de 3,5 mm (connu dans le monde du MIDI sous le nom de TRS, pour *Tip, Ring, Sleeve*) et non un connecteur DIN à 5 voies. L'entrée MIDI vous permet de commander l'ESP32 et/ou le DSP avec un clavier ou une surface de contrôle.
Comme déjà dit, une sortie MIDI est également disponible. Montez R36 (82 Ω) pour activer la sortie (**figure 5**). Faites attention à ce que vous connectez à K5 en raison de la faible valeur de R36. Il doit s'agir d'un jack 3,5 mm à 4 voies (TRRS) et non d'un type à 2 voies (mono ou TS) ou à 3 voies (stéréo ou TRS). Comme la sortie MIDI ne sera probablement pas utilisée par beaucoup de monde, R36 n'est pas monté par défaut pour éviter d'endommager l'ESP32.

USB série

Le connecteur USB de l'ESP32-Pico-Kit est un port USB-série. Dans Arduino, il est disponible en tant que port série par défaut (alias *Serial*). Un ordinateur peut utiliser ce port pour communiquer avec l'ESP32, qui peut relayer des messages au DSP via I²C ou via les ports de réglage analogiques (MP2, MP3, MP8 et MP9) ou d'autres broches MP. Le port série peut également être utilisé pour fournir un pont MIDI ou I²C (sur K1).

Sans fil

L'un des principaux attraits de l'ESP32 est, bien sûr, ses capacités sans fil. L'ESP32-Pico-Kit prend en charge le Wi-Fi 802.11b/g/n, le Bluetooth v4.2 BR/EDR et le Bluetooth LE. Pour les applications audios, la fonctionnalité Bluetooth est particulièrement intéressante car elle permet d'utiliser la carte comme un haut-parleur Bluetooth (émetteur ou récepteur), mais il est également possible de l'utiliser comme une source audio Bluetooth.

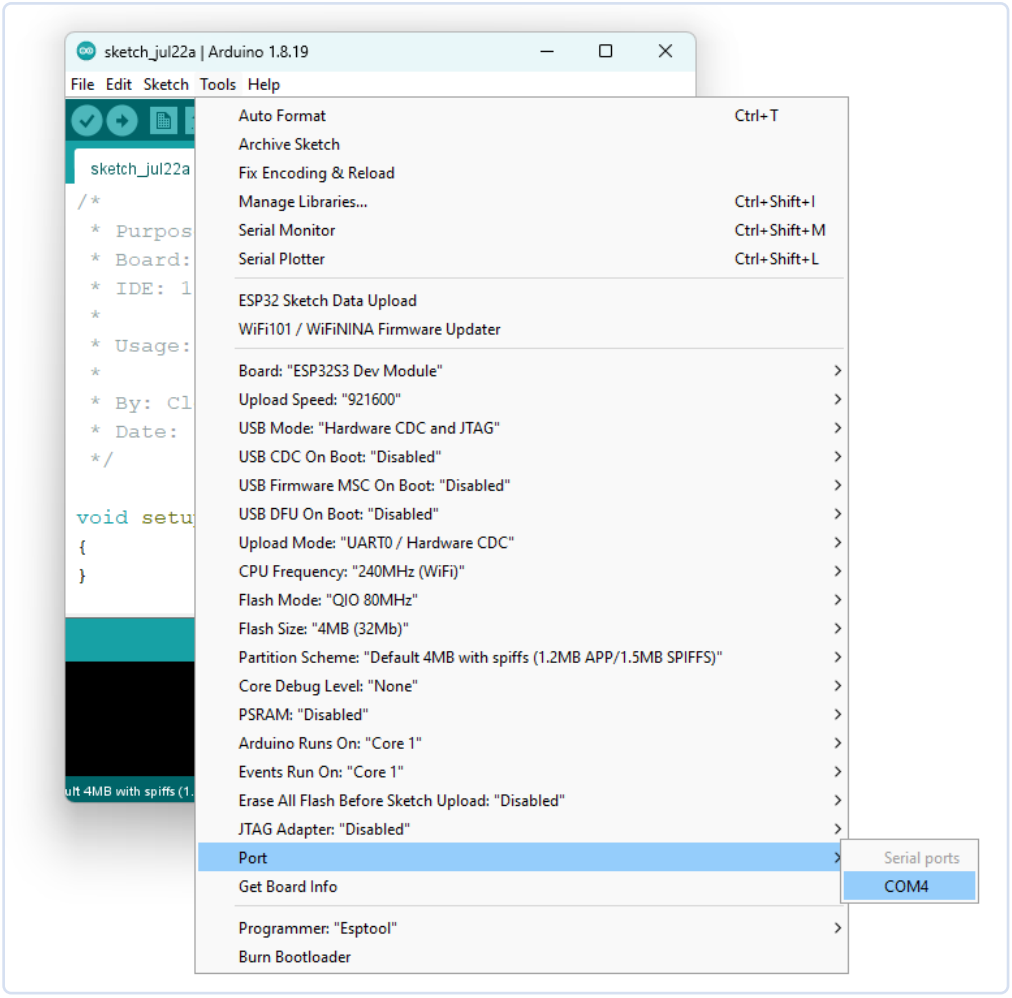


Figure 7. Choisissez le port série correspondant au module ESP32.

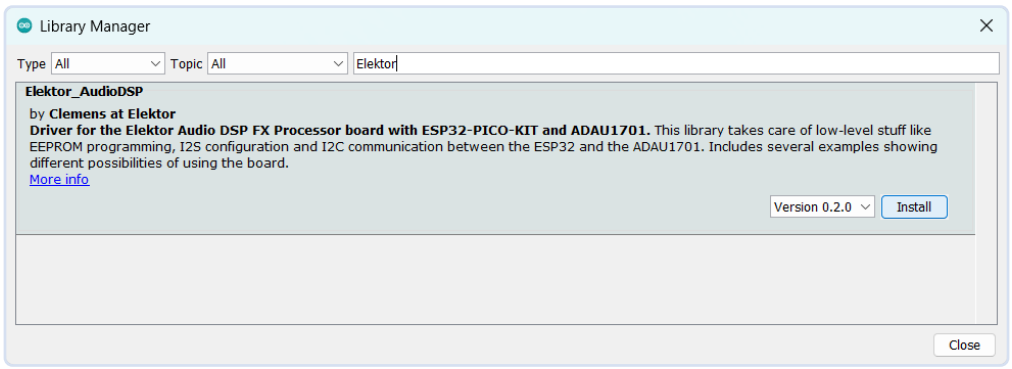


Figure 8. Installer la bibliothèque Elektor_AudioDSP à l'aide du gestionnaire de bibliothèque intégré à l'EDI Arduino.

Pour commencer

Au déballage, avec une EEPROM vide, la carte ne fait rien. La première étape consiste donc à charger un programme dans l'EEPROM. Il y a plusieurs façons de le faire, mais la plus simple est d'utiliser l'ESP32-Pico-Kit. Suivez les étapes ci-dessous.

Préparer la carte

- Branchez le module ESP32-Pico-Kit sur la carte principale en orientant le connecteur USB vers le côté et l'antenne vers le centre.
- Placez un cavalier sur les broches 2 et 3 de JP2 (*Selfboot*).
- Un cavalier doit être placé sur JP1 (*DSP Clock*), mais sa position dépend de l'application. Les broches 1 et 2 (*X1*) sélectionnent le

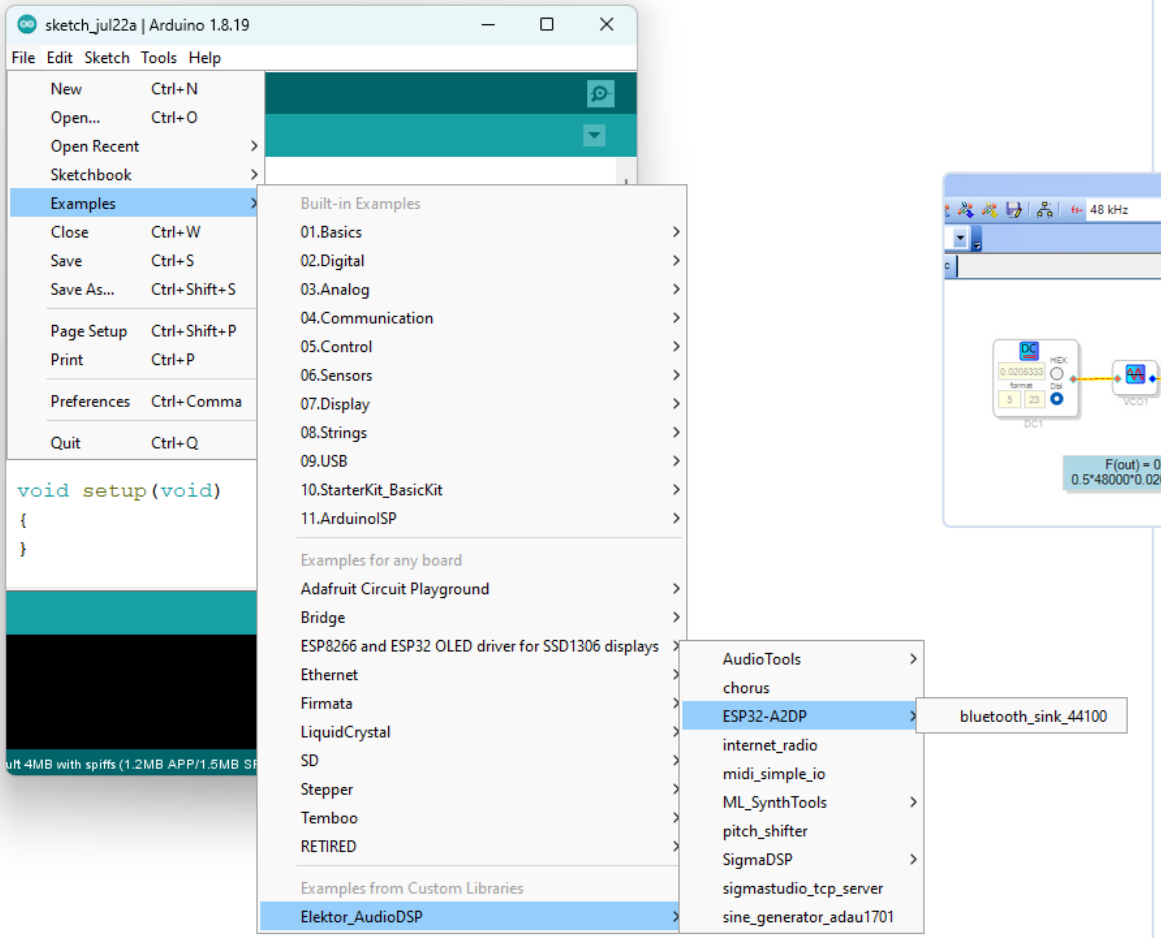


Figure 10. Un programme SigmaStudio [5] est en fait un diagramme de flux.

Figure 9. C'est ainsi que vous accédez aux exemples inclus dans la bibliothèque Elektor_AudioDSP.

quartz X1. Utilisez cette position pour les applications qui n'ont pas besoin de l'ESP32 pour cadencer le DSP (à priori tout ce qui n'utilise pas la communication I²S entre le DSP et l'ESP32). Les positions 2 et 3 de JP1 (*ext. MCLK*) sélectionnent le signal MCLK de l'ESP32 comme horloge pour le DSP. C'est la position requise lors de l'utilisation de la communication I²S entre l'ESP32 et le DSP.

- Connectez le port micro-USB de l'ESP32-Pico-Kit à un port USB libre sur un ordinateur. La LED d'alimentation de l'ESP32-Pico-Kit doit s'allumer (rouge) et l'ordinateur doit détecter un nouveau port série. Si le port n'est pas trouvé, installez le pilote du convertisseur USB-série CP2101 monté sur l'ESP32-Pico-Kit. Pour plus de détails, voir [3].

Préparez votre ordinateur

- Lancer l'EDI Arduino (ce projet a été développé en utilisant la version 1.8.19 de l'EDI).
- Dans le menu *Outils*, sélectionnez la carte *ESP32 PICO-D4* (**figure 6**). Si l'ESP32 PICO-D4 ne figure pas dans la liste des cartes, installez d'abord le package de cartes ESP32. Pour plus de détails sur la manière de procéder, veuillez-vous référer au tutoriel [2]. Notez qu'il est fortement recommandé d'installer une version antérieure à 3.0.0 (par exemple 2.0.17) car certains pilotes ont été complètement révisés dans la version 3.0.0 (par exemple I²S, ADC, et DAC). Il faudra un certain temps avant que les autres bibliothèques utilisant ces périphériques n'intègrent les modifications. Nous avons également observé une expansion du code avec des applications basées sur la version 3.0.x. qui ne tiendront plus dans la mémoire du module ESP32.
- Une fois installé le package de cartes, vous devriez pouvoir sélectionner l'*ESP32 PICO-D4*.
- Dans le menu *Outils*, sélectionnez le port approprié (**figure 7**).
- Dans l'EDI Arduino, ouvrez le gestionnaire de bibliothèques (*menu Outils → Gérer les bibliothèques...*). Recherchez *Elektor*. Vous devriez trouver une bibliothèque nommée *Elektor_AudioDSP* (**figure 8**). Installez-la. Par défaut, la version la plus récente sera installée. Normalement, il n'y a aucune raison d'installer une version plus ancienne, à moins que vous ne vouliez plus de bogues et moins d'options. La bibliothèque peut également être téléchargée depuis GitHub [4] et installée manuellement.

Exemples d'application

Vous êtes maintenant prêt à créer et à charger des programmes dans le processeur Audio DSP FX. Pour vous aider à démarrer, la bibliothèque *Elektor_AudioDSP* est accompagnée d'une collection d'exemples. Ouvrez un exemple en cliquant sur *Fichier → Exemples* et en faisant défiler la page jusqu'à la section *Exemples dans bibliothèques personnalisées*. Vous devriez y trouver une entrée pour la bibliothèque *Elektor_AudioDSP*. Cliquez dessus et choisissez un exemple (**figure 9**). Notez que les entrées avec une flèche vers la droite (par exemple *AudioTools*) requièrent une bibliothèque tierce portant le nom de l'entrée. Cela signifie que les exemples du sous-menu *AudioTools* nécessitent la bibliothèque *AudioTools*. Certaines de ces bibliothèques peuvent être installées via le gestionnaire de bibliothèques de l'EDI comme décrit ci-dessus, mais pas toutes. Le lien vers la bibliothèque est inclus dans le croquis.

Un exemple consiste en un croquis Arduino et, dans la plupart des cas, en un exécutable pour le DSP à charger dans l'EEPROM de la carte.

Les commentaires en tête du croquis vous indiquent comment régler les cavaliers de la carte et comment utiliser l'exemple. L'exemple s'occupe de tout. Il suffit de charger un croquis et de l'envoyer sur la carte. Si l'EEPROM a besoin d'être programmée, le croquis le fera automatiquement.

Important : Veuillez lire les commentaires en haut d'un croquis d'exemple car ils contiennent des informations pour la configuration de la carte !

Essayons un exemple

Pour illustrer l'utilisation d'un exemple, chargeons *sine_generator_adau1701*. Après avoir téléchargé ce croquis sur la carte, il programmera l'EEPROM (alias « E2Prom ») avec l'algorithme DSP *i2s_pass_through_48000*. Son nom ne l'indique pas, mais cet algorithme transforme le DSP en un générateur d'ondes sinusoïdales commandé en tension, dont la sortie se trouve sur le connecteur K4 (et non K3). La fréquence de l'oscillateur est réglée par P1, connecté à l'ESP32. Dans la boucle principale, les quatre potentiomètres sont lus et leurs valeurs sont imprimées sur le port série. Ces valeurs sont également envoyées sous forme de tension aux entrées de commande du DSP. Après avoir programmé l'EEPROM (cela prend environ une seconde), la LED de la carte commence à clignoter à une fréquence de 1 Hz. Si vous réinitialisez le croquis, l'EEPROM ne sera pas programmée à nouveau (à moins que son contenu ne soit différent), et la LED commence à clignoter presque immédiatement après la réinitialisation.

Voici les instructions (tirées du croquis) pour que tout fonctionne :

- Régler JP1 (*horloge DSP*) sur la position *X1* (broches 1 et 2)
- Régler JP2 en position *Selfboot* (broches 2 et 3)
- Utiliser P5 pour ajuster TP1 à 3,15 V
- Charger le croquis sur l'ESP32
- Attendre que la LED1 commence à clignoter à une fréquence de 1 Hz
- Branchez un casque ou un amplificateur sur K4 (pas sur K3) et profitez-en.
- P1 règle la fréquence

Exemples de SigmaStudio

Les exemples qui utilisent un programme DSP comprennent le projet SigmaStudio [5] (**figure 10**). Après avoir installé la bibliothèque *Elektor_AudioDSP*, vous trouverez les exemples dans :

```
[path-to-your-Arduino-folder]\sketchbook\libraries\
Elektor_AudioDSP\examples\
```

Une collection d'exécutables DSP précompilés se trouve dans le fichier *adau1701_e2prom_collection.h* inclus dans la bibliothèque *Elektor_AudioDSP*. Les fichiers de projet SigmaStudio utilisés pour générer ces exécutables se trouvent dans le dossier *extra* de la bibliothèque. Sont inclus deux exécutables génériques qui transmettent simplement les données audios de l'ESP32 à l'ADAU1701 en utilisant I²S :

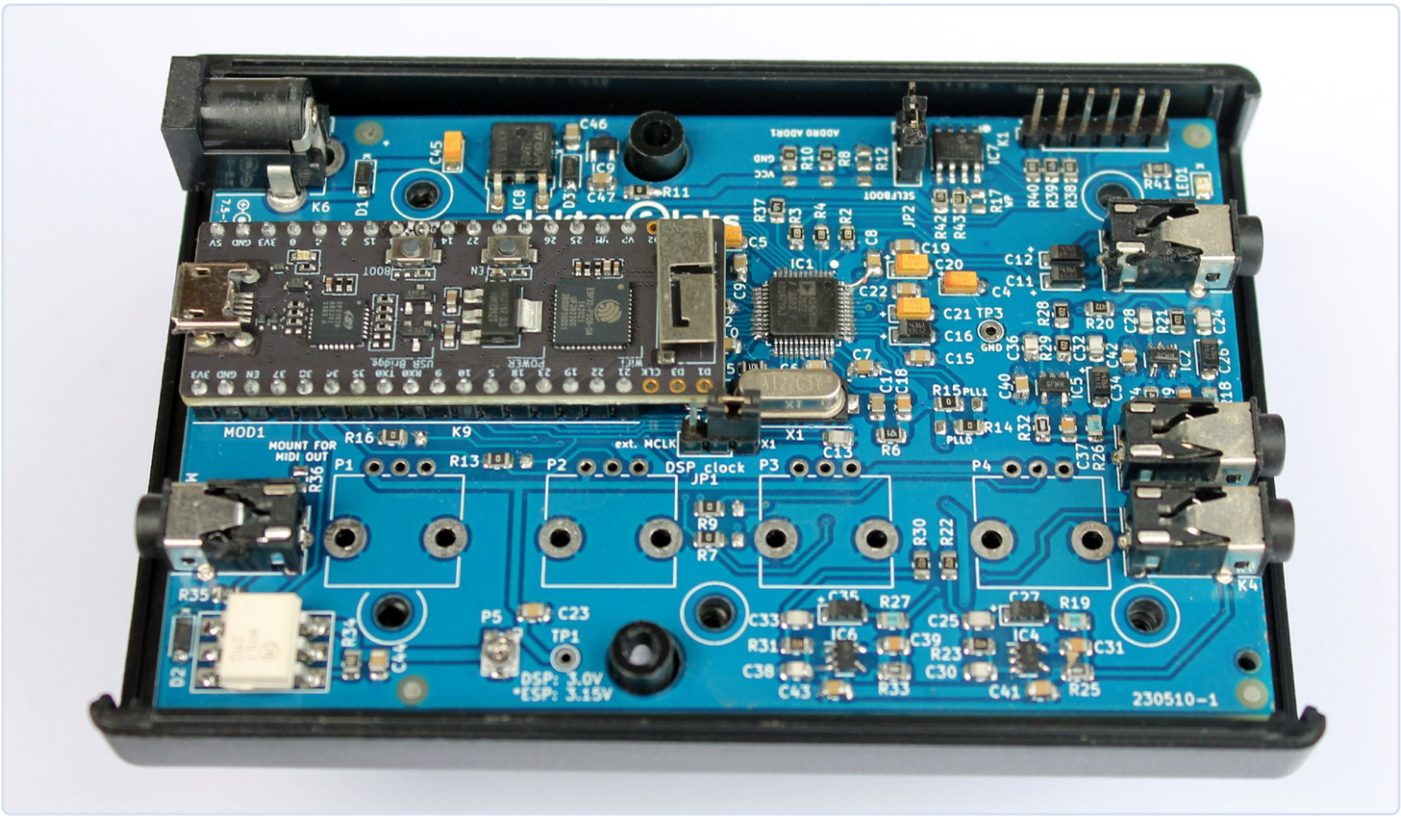


Figure 11. Le processeur audio DSP FX s'insère dans un boîtier de type Hammond 1593N. Pour permettre la fermeture du couvercle, le module ESP32 doit être monté sans barrettes.

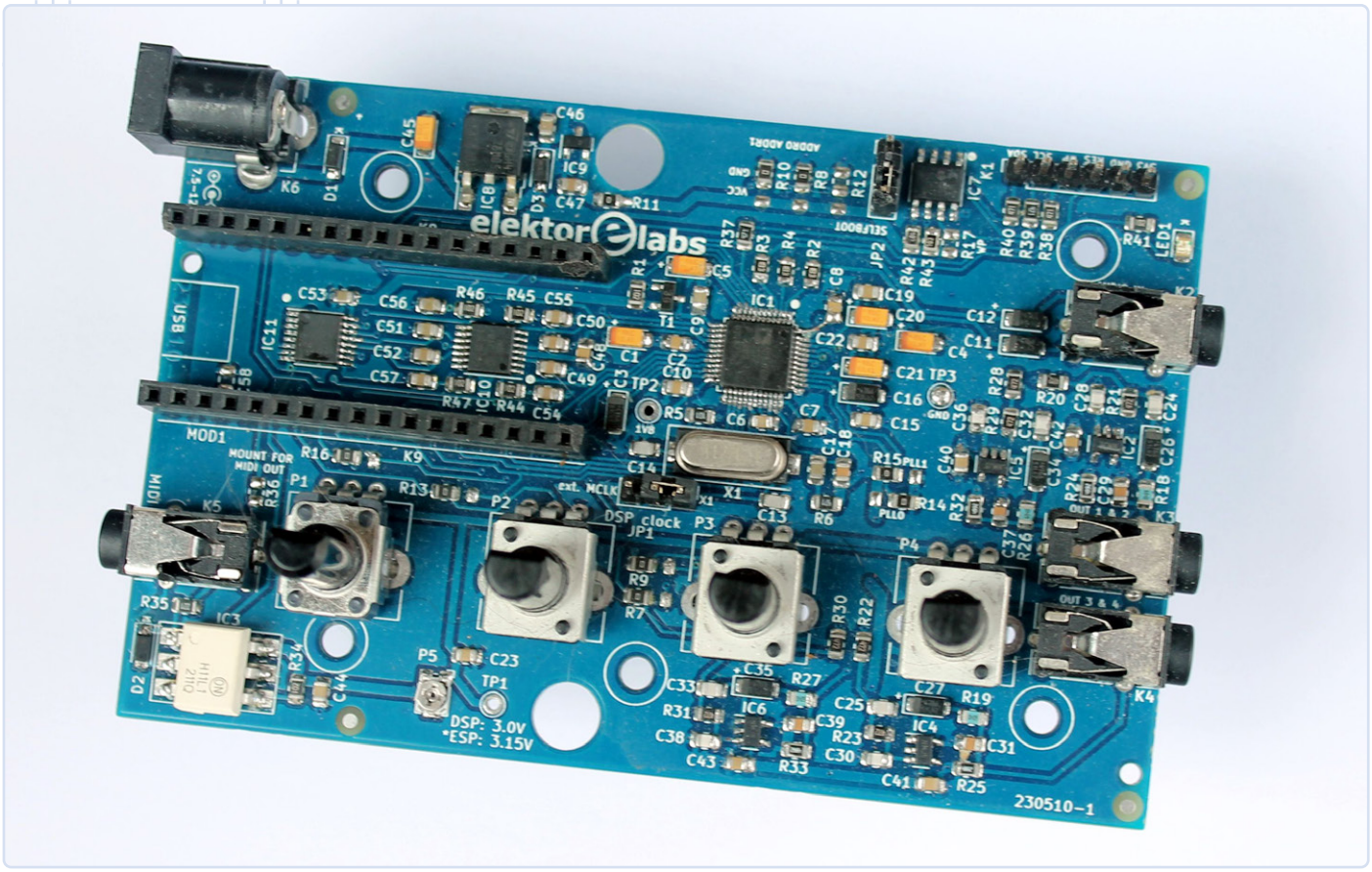


Figure 12. La carte avec les connecteurs et les potentiomètres montés.

i2s_pass_through_44.1khz et *i2s_pass_through_48khz*. La seule différence entre eux est la fréquence d'échantillonnage. Ces deux exécutables transforment le DSP en une sortie audio stéréo sur le connecteur K3 (avec un générateur de sinus sur le connecteur K4). JP1 doit être en position *ext. MCLK*.

À suivre...

C'est tout pour l'instant. Cet article a présenté beaucoup d'informations et, nous l'espérons, suffisamment pour vous permettre de démarrer. Dans le prochain article, nous irons plus loin dans la création de vos propres applications pour la carte Audio DSP FX Processor.

VF : Denis Lafourcade — 230510-04

Questions ou commentaires ?

Envoyez un courriel à l'auteur (clemens.valens@elektor.com), ou contactez Elektor (redaction@elektor.fr).

À propos de l'auteur

Clemens Valens a commencé à travailler pour Elektor en 2008 et a occupé plusieurs postes depuis. Il fait actuellement partie de l'équipe de développement de produits. Ses principaux centres d'intérêt en électronique sont le traitement du signal (numérique) et ses applications dans la production musicale et la synthèse sonore.

Produits

- **ESP32-PICO-KIT**
  www.elektor.com/18423
- **Elektor Audio DSP FX Processor**
  www.elektor.fr/20895
- **Dogan & Ahmet Ibrahim, *Practical Audio DSP Projects with the ESP32*, (Elektor, 2023)**
  www.elektor.fr/20558

LIENS

[1] Ramkumar Ramaswamy, «carte DSP audio universelle à ADAU1701» Elektor 1-2/2014 : https://www.elektormagazine.fr/magazine/elektor-201401/24299
[2] ESP32 FAQ, How-to & Getting Started: https://www.elektormagazine.fr/esp32-faq
[3] Pilotes VCP CP210x USB to UART Bridge : https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers
[4] The Elektor_AudioDSP library at GitHub: https://github.com/ClemensAtElektor/Elektor_AudioDSP
[5] Obtenir SigmaStudio (pas SigmaStudio +) : https://www.analog.com/sigmastudio
[6] Audio DSP FX Processor sur Elektor Labs : https://www.elektormagazine.fr/labs/audio-dsp-fx-processor

SPECTRAN® V6 XPR

REAL-TIME SPECTRUM ANALYZER

Mastering **Microwave** Measurements

WR08
90 - 140 GHz

WR10
75 - 110 GHz

WR12
60 - 90 GHz

WR15
50 - 75 GHz

RTBW	Sweep Speed	ADC	DANL*
490 MHz	**3 THz/s**	**16-Bit**	**-170 dBm/Hz**

- **World's first** USB real-time 140 GHz spectrum analyzer
- WR08 | WR10 | WR12 | WR15 waveguide connectors
- Analyze important standards like **5G or radar**
- **Record-breaking** USB real-time bandwidth of 490 MHz
- **24/7 recording** and analyzing of IQ-data
- **16-Bit 2 GSPS ADC**
- **Single USB-C** connection incl. power
- **Windows and Linux** software included

*Depending on frequency

MADE IN GERMANY

www.aaronia.com
mail@aaronia.de
+49 6556 900 310

Aaronia AG
Aaroniaweg 1
D-54597 Strickscheid

AARONIA AG®
WWW.AARONIA.DE