ESP32forth
Synthétiseur

Marc PETREMANN

Version 1.0 - 26/01/26

Table des matières

Préambule

Ce manuel est destiné à maîtriser le module Synthétiseur pour ESP32forth.

Synthèse polyphonique réelle : Contrairement à beaucoup de petites librairies, gère très bien plusieurs voix simultanément.

- **Architecture I2S native :** Elle est conçue pour piloter directement des DAC externes (comme le **PCM5102** ou le **MAX98357A**). Cela donne un son **16-bit propre**, bien supérieur au DAC interne de l'ESP32.
- **Simplicité d'API :** Elle utilise des commandes très intuitives comme `noteOn(midi_note, velocity)` et `noteOff()`, ce qui la rend parfaite pour créer un synthétiseur piloté par un clavier MIDI.

La synthèse sonore

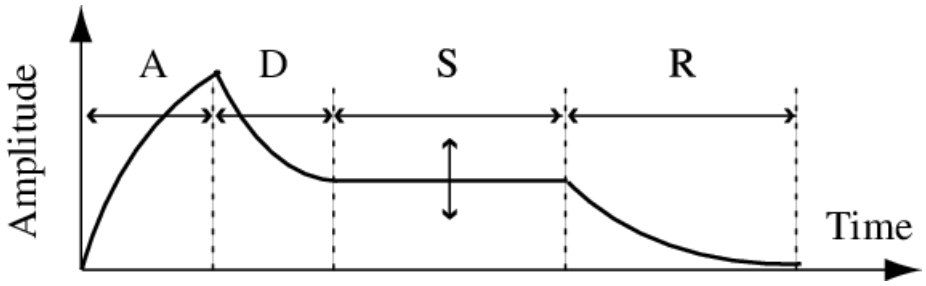
Les notes musicales

La courbe ADSR

Pour comprendre comment un synthétiseur sculpte le son, il faut regarder l'enveloppe **ADSR**. C'est l'outil qui détermine l'évolution du volume (ou d'autres paramètres) d'une note dans le temps, de l'instant où tu appuies sur la touche jusqu'à celui où le silence revient.

Sans ADSR, un synthé sonnerait de manière très binaire : "on/off". Grâce à elle, tu peux transformer un bip monotone en un piano délicat ou en un violon expressif.

Les 4 étapes de la courbe ADSR



L'acronyme **ADSR** signifie **Attack, Decay, Sustain, Release**.

Attack (Attaque)

C'est le temps que met le son pour atteindre son **volume maximum** après le déclenchement de la note.

- **Attaque courte :** Un son percutant (comme une batterie ou un piano).
- **Attaque longue :** Un son qui monte progressivement (comme des nappes de cordes ou d'ambiance).

Decay (Déclin)

Une fois le pic atteint, le son redescend vers un niveau de croisière. Le *Decay* définit la durée de cette chute.

- Si le Decay est très court, on obtient un effet de pincement (comme une guitare).

Sustain (Maintien)

Contrairement aux trois autres, le Sustain n'est pas une valeur de temps, mais un **niveau de volume**. C'est le volume qui reste constant tant qu’une est touche enfoncée.

- Si elle est réglée à zéro, le son s'éteindra même si on reste appuyé.

Release (Relâchement)

C'est le temps que met le son pour disparaître complètement après **relâchement de touche**.

- **Release court :** Le son s'arrête net.
- **Release long :** Le son résonne et s'estompe lentement (comme dans une cathédrale).

Montage du synthétiseur

Utilisation du DAC interne

Dans ce mode particulier (`SMODE_DAC`), l'ESP32 est limité à deux broches spécifiques car elles sont reliées physiquement aux convertisseurs analogiques internes :

- **GPIO 25** (Canal Droit / Right)
- **GPIO 26** (Canal Gauche / Left)

Dans ce mode, le paramètre `clkPin` est souvent ignoré par la librairie en interne, mais vous devez quand même lui passer une valeur (mettez 0 ou 26 par défaut).

Utilisation d’un DAC externe

Si vous utilisez un DAC externe (ex: MAX98357A, PCM5102), voici le câblage le plus courant pour le mode **I2S** :

Fonction	Paramètre dans le code	Broche suggérée (GPIO)
Bit Clock	clkPin (ou bck)	**GPIO 26**
Word Select	ws (ou lrc)	**GPIO 25**
Data Out	data	**GPIO 22**

Attention aux "Input Only".

Lors du choix de vos broches, évitez absolument les broches **34, 35, 36 et 39**. Ce sont des broches d'entrée uniquement ("Input Only"), elles ne peuvent pas envoyer de signal d'horloge ou de données.

Exemple de branchement type (MAX98357A) :

1. **LRC** (WS) → GPIO 25
2. **BCLK** (CLK) → GPIO 26
3. **DIN** (DATA) → GPIO 22
4. **GND** → GND
5. **Vin** → 5V (ou 3.3V)

Voici une configuration optimisée pour exploiter aussi un écran OLED sur une carte ESP 30 broches :

Fonction	Broche GPIO recommandée	Commentaire
OLED SDA	**GPIO 21**	Standard I2C
OLED SCL	**GPIO 22**	Standard I2C
I2S clkPin (BCK)	**GPIO 26**	Audio
I2S ws (LRC)	**GPIO 25**	Audio
I2S data (DIN)	**GPIO 19**	**Choix sûr** (évite le conflit avec le 22)

Les mots de gestion synthétiseur

Mots natifs depuis la librairie ESP32Synth.h

synth.begin (dataPin SynthOutputMode clkPin wsPin i2sDepth – fl)

Le mot `synth.begin` est le point de départ indispensable pour initialiser le moteur sonore. Il configure la manière dont ESP32Forth va générer et transmettre le signal audio.

Liste des paramètres :

- **dataPin** : port GPIO pour les données
- **SynthOutputMode** :
  - **SMODE_PDM** : (Pulse Density Modulation)
    Ce mode est utilisé pour les microphones numériques ou certains haut-parleurs très simples. Le signal est codé par la densité des impulsions. Sur ESP32, cela permet d'obtenir une sortie audio correcte sur une broche GPIO numérique avec un simple petit filtre (résistance + condensateur).
  - **SMODE_I2S** : (Inter-IC Sound)
    C'est le mode de haute fidélité. Il est destiné à être utilisé avec un composant externe appelé un **DAC I2S** (comme le MAX98357A ou le PCM5102). Le son est envoyé sous forme numérique pure à une puce spécialisée qui le transforme en un son de haute qualité.
  - **SMODE_DAC** : (Digital to Analog Converter)
    Ce mode utilise les convertisseurs numérique-analogique **intégrés** à l'ESP32 (disponibles sur les broches GPIO 25 et 26).
    - **Avantage :** Pas besoin de matériel externe, vous branchez directement un petit haut-parleur ou un casque (avec une protection).
    - **Inconvénient :** La qualité est limitée à 8 bits, ce qui produit un son avec un peu de souffle ou de "bruit de fond".
- **ClkPin** : GPIO associé à ce signal. Dans le contexte de l'audio numérique et de la bibliothèque **ESP32Synth**, le paramètre **`clkPin`** (souvent confondu ou associé au `BCK` dans le protocole I2S) correspond au **Bit Clock Pin**.
  Sans ce signal, le composant qui reçoit les données ne saurait pas à quel moment précis lire le bit présent sur la broche de données (`data`). Il verrait une suite de 0 et de 1 illisible.
- **WsPin** : GPIO associé à ce signal WS/LRCK.
- **I2sDepth** : Profondeur en bits :
  - **I2S_8BIT** : son basse définition
  - **I2S_16BIT** : son standard
  - **I2S_32BIT** : son Hi-Fi

Exemple pour DAC interne :

```
25 SMODE_DAC -1 -1 I2S_8BIT synth.begin
```

Paramètre	Valeur	Explication
dataPin	25 (ou 26)	Broche de sortie audio analogique.
mode	SMODE_DAC	Indique à la librairie d'utiliser les convertisseurs internes.

Paramètre	Valeur	Explication
clkPin	0 (ou -1)	Inutilisé en mode DAC, on met une valeur neutre.
wsPin	0 (ou -1)	Inutilisé en mode DAC.
i2sDepth	I2S_Depth_16Bit	Même si le DAC interne est en 8-bit, la librairie gère souvent la conversion depuis le 16-bit.

Exemple pour DAC externe :

```
19 SMODE_I2S 26 25 I2S_16BIT synth.begin
```

Paramètre	Valeur	Explication
dataPin	19	Envoie les données vers le pin DIN du module.
mode	SMODE_I2S	Active la sortie numérique série.
clkPin	26	Envoie le rythme (BCLK) au module.
wsPin	25	Indique le canal Gauche/Droit (LRC) au module.
i2sDepth	I2S_Depth_16Bit	Format standard pour une haute fidélité.

synth.noteOn (voice freq vol --)

`noteOn(voice, freq, vol)`: Abre o "gate". Isso dispara o envelope, que entra na fase de **Attack**. O som começa a subir de volume. Após o ataque, ele passa para o **Decay** até atingir o nível de **Sustain**.

synth.noteOff (voice --)

Le mot `synth.noteOff` indique au synthétiseur d'arrêter de jouer une note. Mais attention, contrairement à ce qu'on pourrait penser, ça ne coupe pas le son instantanément dans la plupart des cas.

Lorsque vous appelez `noteOff`, le synthétiseur quitte la phase de maintien (**Sustain**) et entre dans la phase de relâchement (**Release**).

Paramètre :

- **`voice`** : L'index de la voix que vous voulez arrêter (0, 1, 2, etc.).

synth.setEnv (voice a d s r --)

Ce mot permet de configurer manuellement une enveloppe **ADSR** (Attack, Decay, Sustain, Release) simplifiée pour une voix spécifique, sans avoir à définir une structure `Instrument` complète avec des tableaux.

C'est la méthode classique utilisée dans la plupart des synthétiseurs pour sculpter l'évolution du volume d'une note.

Détail des paramètres :

1. **`voice`** : L'index de la voix (le canal du synthétiseur) que vous voulez modifier. Si votre synthétiseur est polyphonique, vous pouvez régler chaque voix différemment.

2. **a (Attack)** : Le temps que met le son pour passer du silence au volume maximum après avoir pressé une touche :
    - *Valeur courte* : Impact immédiat (ex: piano, batterie).
    - *Valeur longue* : Apparition progressive (ex: violon, nappe de synthé).
3. **d (Decay)** : Le temps que met le son pour redescendre du pic de l'attaque vers le niveau de maintien (`s`).
4. **s (Sustain)** : Le **niveau** de volume (0 à 255) qui reste constant tant que la touche est maintenue enfoncée. *Note : Contrairement aux autres, c'est un niveau de volume, pas une durée.*
5. **r (Release)** : Le temps que met le son pour disparaître complètement une fois que la touche est relâchée (`noteOff`).

Pourquoi est-ce utile par rapport à la structure Instrument ?

Alors que la structure `Instrument` utilise des tableaux pour créer des enveloppes complexes et changeantes, `synth.setEnv` applique un modèle mathématique linéaire standard. C'est beaucoup plus simple à manipuler en direct depuis votre terminal Forth.

Exemple :

```
0 0 500 50 300 synth.setEnv          \ piano like sound
0 50 20 255 50 synth.setEnv          \ flûte ou orgue
0 1500 1000 150 2000 synth.setEnv    \ nappe de synthé – pad spatial
0 5 200 0 50  synth.setEnv           \ percussion
```

Les structures

Instrument

Codage :

```
struct Instrument
     8 field ->seqVolumes
    16 field ->seqWaves
     8 field ->seqLen
    16 field ->seqSpeedMs
     8 field ->susVol
    16 field ->susWave
     8 field ->relVolumes
    16 field ->relWaves
     8 field ->relLen
    16 field ->relSpeedMs
     8 field →smoothMorph
```

Voici l'usage de chaque élément :

La Phase d'Attaque (Sequence)

Cette phase démarre dès que vous appelez `noteOn`.

- **`seqVolumes`** : Pointeur vers un tableau d'octets. Chaque valeur (0-255) définit le volume à une étape précise. Cela permet de créer des impacts forts ou des montées progressives.
- **`seqWaves`** : Pointeur vers un tableau de formes d'ondes. Le synthétiseur peut changer de timbre *pendant* l'attaque.
- **`seqLen`** : Le nombre d'éléments dans les tableaux ci-dessus. Si vous avez 10 valeurs de volume, `seqLen` doit être 10.
- **`seqSpeedMs`** : La durée de chaque étape de la séquence en millisecondes. Si vous mettez 10ms et que `seqLen` est 50, l'attaque durera 500ms.

La Phase de Maintien (Sustain)

C'est l'état du son tant que la touche reste enfoncée (après la séquence d'attaque).

- **`susVol`** : Le volume de maintien (0-255). C'est le niveau constant du son pendant que vous jouez.
- **`susWave`** : La forme d'onde utilisée pendant le maintien (généralement une onde fixe comme un carré ou une dent de scie).

La Phase de Relâchement (Release)

Cette phase démarre dès que vous appelez `noteOff`.

- **`relVolumes`** : Pointeur vers un tableau définissant comment le son s'éteint. On commence généralement à la valeur du `susVol` pour descendre vers 0.
- **`relWaves`** : Permet de changer de timbre pendant que le son s'estompe (utile pour simuler la résonance d'une corde).
- **`relLen`** : Nombre d'étapes dans la séquence de relâchement.
- **`relSpeedMs`** : Vitesse de lecture de la séquence de fin.

Le lissage (Interpolation)

- **`bool smoothMorph`** :

- **Si `true` (1)** : Le synthétiseur calcule des valeurs intermédiaires entre les étapes des séquences. Le son change de manière fluide (glissando de timbre).
- **Si `false` (0)** : Le son saute brusquement d'une valeur à l'autre. Cela donne un aspect "lo-fi" ou "8-bit" plus marqué.