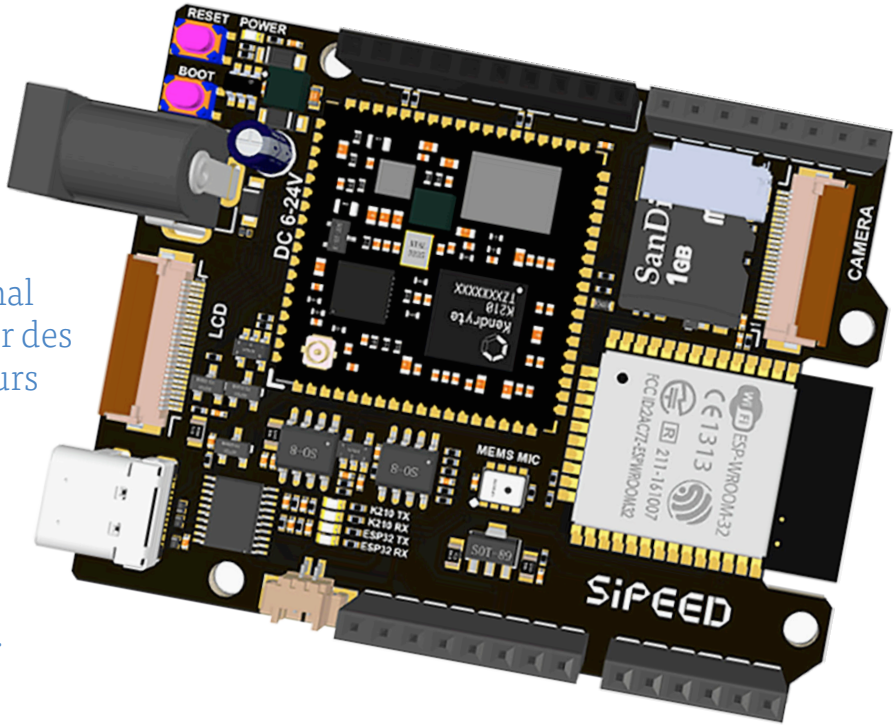
FFT avec Maixduino

Acquisition de spectres de fréquences

Tam Hanna (Slovénie)

La transformée rapide de Fourier est intéressante, mais pas seulement dans le domaine audio : pour l'analyse du bruit d'une turbine ou des niveaux de tension de véhicules mal amortis, la FFT permet souvent de tirer des conclusions sur les régimes des moteurs et autres. La carte Maixduino avec le K210-SoC disponible chez Elektor se prête non seulement à accompagner vos premiers pas dans l'univers des réseaux neuronaux, mais aussi pour un accès à la FFT rapide en temps réel.



Les progrès des logiciels et du matériel mettent à la disposition de monsieur Tout-le-Monde de techniques auparavant réservées aux milices privées et aux gouvernements.

Sipeed, spécialiste en semi-conducteurs connu jusqu'à présent surtout pour ses expériences avec le processeur RISC-V, propose une nouvelle carte avec le Maixduino. Celui-ci réunit un ESP32 pour la communication en réseau, et un SoC appelé *K210 Kendryte*, que le fabricant a l'intention d'utiliser dans les applications d'IA. Avec Google vous récolterez de nombreuses informations si vous recherchez «kendryte + problème».

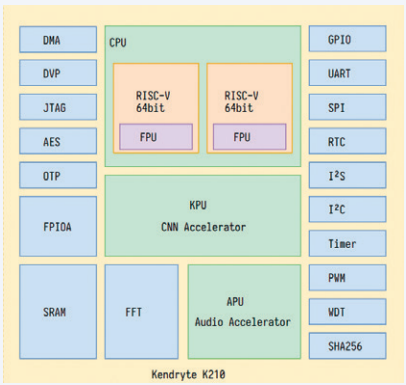


Figure 1. Le K210 est livré avec quelques extensions (source : Canaan Inc.).

Pour en savoir plus sur ce SoC, visitez le site [1] où il faut parfois faire preuve de patience. La fiche technique du K210 contient principalement une description des broches. Examinez le schéma (**fig. 1**), il est également intéressant.

Le processeur principal (CPU) adopte l'architecture RISC-V (avec les options I, M, A et C) à source ouverte, avec deux auxiliaires : d'abord la KPU, derrière laquelle se trouve un processeur de réseau neuronal (sans autre information dans la fiche technique), qui met en œuvre un réseau neuronal conservé dans le matériel, semblable au processeur *Tristar* de DPO utilisé autrefois par *Danaher* dans les TDS 5xxD et TDS 7xxD.

L'autre, un processeur audio appelé APU, traite les informations audio jusqu'à 192 kHz, stocke ces informations par DMA directement dans la mémoire principale et peut également appliquer automatiquement des traitements FFT. Nous utiliserons cet APU dans les étapes suivantes pour réaliser une petite FFT.

Mentionnons encore divers périphériques sur lesquels nous ne nous attarderons pas ici.

Une question d'outils

Ce qui importe pour nous, ce sont les outils de développement. Au niveau le plus bas se trouve le SDK autonome téléchargeable [2] – une bibliothèque (heureusement) en C, qui nous permet d'interagir directement avec les primitives du processeur.

Un niveau au-dessus, c'est ArduinoCore [3], une infrastructure pour l'EDI Arduino, également utilisable avec Platform.IO. Enfin, il y a MaixPy, un environnement de programmation dérivé de MicroPython, réservé

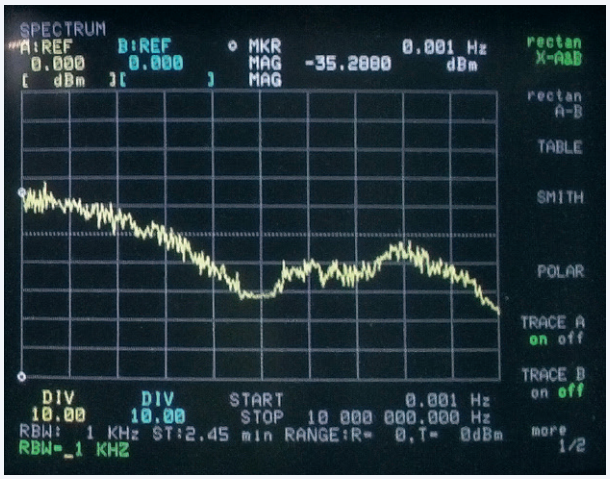


Figure 2. Les bandes étroites offrent une haute résolution mais imposent un temps de balayage plus long (Sweep Time)...

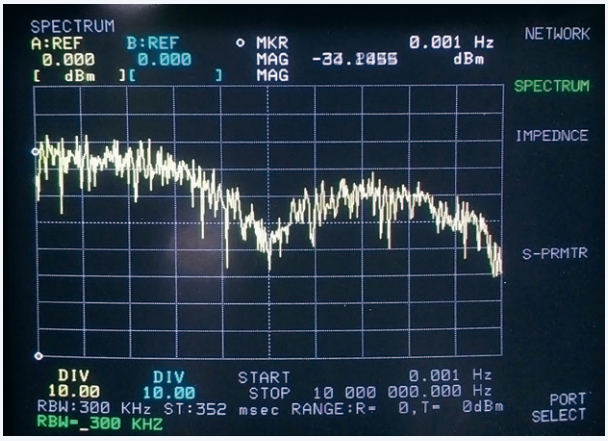


Figure 3. ...alors que des bandes plus larges et à balayage plus rapide fournissent des informations moins précises.

à ceux qui ont beaucoup d'expérience avec Python (lisez l'encadré !). L'étape suivante consiste à réfléchir à la manière de contruire les tampons à traiter. Dans la fiche technique déjà mentionnée, nous découvrons que le Kendryte ne fonctionne pas avec des tailles arbitraires. Pour utiliser l'accélération matérielle, il faut choisir l'une des tailles de tampon suivantes : 64, 128, 256 ou 512.
Lorsqu'on fait de la FFT, il est raisonnable d'avoir à portée de main un analyseur de spectre classique. Dans notre cas, la taille du *godet* est pertinente, car elle décrit la finesse des bandes de fréquences résultantes.

Les **figures 2** et **3** viennent du célèbre HP4195A et illustrent le principe. Alors que notre analyseur de spectre analogique affiche, pour cause de bruit, des bandes instables dans les deux cas, la courbe réelle est plus régulière avec la FFT. La résolution en fréquence découle du rapport *fréquence d'échantillonnage / taille de la FFT* ; si 1024 échantillons sont prélevés à 8192 Hz, la largeur de bande de résolution est RBW = 8 Hz. Comme c'est si souvent le cas, avec la FFT non plus, la maximisation n'est pas tout. La corrélation entre nombre de bandes de fréquences et nombre d'échantillons entrants est en principe le facteur 2 ; cependant, c'est plus difficile en réalité, car la complexité des algorithmes FFT augmente rapidement avec le nombre d'échantillons. La **figure 4** montre cette croissance problématique qui, dans le monde réel, implique une limitation de la précision de notre FFT.
À l'étape suivante, nous définirons un groupe de constantes. En pratique, spécifiez la longueur de la FFT au moyen d'un `#define` pour faciliter les ajustements ultérieurs entre précision et vitesse :

```
uint32_t rx_buf[1024];
#define FFT_N                512U
#define FFT_FORWARD_SHIFT    0x0U
#define SAMPLE_RATE          38640
#define FRAME_LEN            512
```

Le stockage effectif des informations fournies par le DMA nécessite un groupe de champs de mémoire supplémentaires :

```
 nt16_t  real[FFT_N];
int16_t  imag[FFT_N];
int    hard_power[FFT_N];
uint64_t fft_out_data[FFT_N / 2];
uint64_t buffer_input[FFT_N];
uint64_t buffer_output[FFT_N];
fft_data_t *output_data;
fft_data_t *input_data;
complex_hard_t data_hard[FFT_N] = {0};
```

Initialisation du matériel

Le fabricant du Maixduino nous gratifie d'un microphone MEMS MSM261S4030H0. Ces petits microphones sont intéressants (**fig. 5**). L'affectation des broches révèle que le composant s'occupe de l'adaptation du signal et fournit les données sur un bus série.
Le premier obstacle est que le logiciel n'initialise pas automatiquement les lignes de données nécessaires à la communication avec le microphone. Il est facile d'y remédier au début de la configuration :

```
void setup()
{
    lcd.begin(15000000, COLOR_RED);
    fpioa_set_function(20, FUNC_I2S0_IN_D0);
    fpioa_set_function(18, FUNC_I2S0_SCLK);
    fpioa_set_function(19, FUNC_I2S0_WS);
```

L'étape suivante consiste à configurer le matériel I2S. Le matériel de la carte détermine généralement le code ; un réglage crucial est le taux d'échantillonnage à définir avec la méthode `i2s_set_sample_rate`. Comme les options sont réduites du fait de la taille de la FFT, le taux

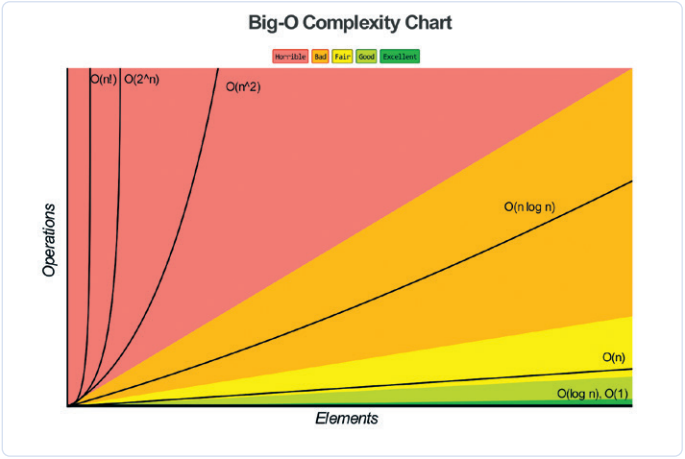
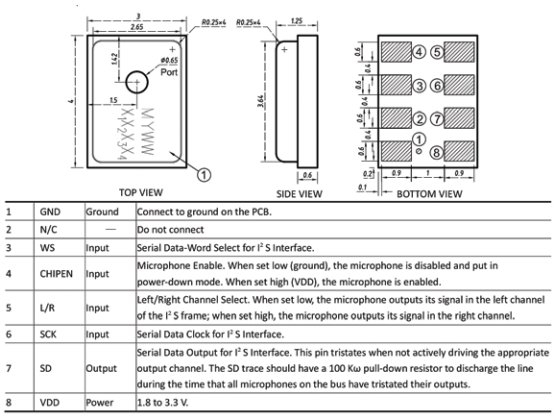


Figure 4. Pour obtenir des bandes de fréquences infiniment petites, vous avez besoin de beaucoup de puissance de calcul. (Photo : https://www.bigocheatsheet.com/).



1	GND	Ground	Connect to ground on the PCB.
2	N/C	—	Do not connect
3	WS	Input	Serial Data-Word Select for I²S Interface.
4	CHIPEN	Input	Microphone Enable. When set low (ground), the microphone is disabled and put in power-down mode. When set high (VDD), the microphone is enabled.
5	L/R	Input	Left/Right Channel Select. When set low, the microphone outputs its signal in the left channel of the I²S frame; when set high, the microphone outputs its signal in the right channel.
6	SCK	Input	Serial Data Clock for I²S Interface.
7	SD	Output	Serial Data Output for I²S Interface. This pin tristates when not actively driving the appropriate output channel. The SD trace should have a 100 KΩ pull-down resistor to discharge the line during the time that all microphones on the bus have tristated their outputs.
8	VDD	Power	1.8 to 3.3 V.

Figure 5. Avec MEMSensing, l'utilisateur du capteur a beaucoup moins de travail (source : MEMSensing Microsystems).

d'échantillonnage sera l'un des rares moyens de «traiter» la largeur de la bande de fréquences :

```
    i2s_init(I2S_DEVICE_0, I2S_RECEIVER, 0x3);
    i2s_set_sample_rate(I2S_DEVICE_0, SAMPLE_RATE );
    i2s_rx_channel_config(I2S_DEVICE_0, I2S_CHANNEL_0,
                    RESOLUTION_16_BIT, SCLK_CYCLES_32,
                    TRIGGER_LEVEL_4, STANDARD_MODE);
```

Il ne reste que quelques initialisations :

```
    dmac_init();
    sysctl_enable_irq();
}
```

La prochaine tâche est l'opération FFT proprement dite. Il faut s'assurer que les informations fournies par I2S sont stockées en mémoire :

```
void loop()
{
  int i, sampleID;

  i2s_receive_data_dma(I2S_DEVICE_0, &rx_buf[0],
   FRAME_LEN * 2, DMAC_CHANNEL3);
  dmac_wait_idle(DMAC_CHANNEL3);
```

En appelant `dmac_wait_idle`, nous imposons une pause au processeur jusqu'à ce que le canal DMA se libère. Cela garantit que chaque `boucle` fonctionne avec une base de données cohérente et nouvelle. Si (pour voir) vous supprimez (provisoirement) l'appel à `dmac_wait_idle`, vous constaterez d'étranges artefacts (*aliasing*) à l'écran. C'est dû probablement au fait que, selon la fiche technique, les modules FFT ne peuvent être utilisés que via le moteur DMA. En cas de collision, la catastrophe est imminente !
Lorsque vous travaillez avec les moteurs FFT ou des unités fonctionnelles fixes similaires, vous devez toujours tenir compte du fait que le matériel attend des informations dans un format de stockage prescrit. Dans le cas de `tag_fft_data`, il s'agit de la structure suivante, déclarée à plusieurs endroits dans le SDK :

```
typedef struct tag_fft_data
{
    int16_t  I1;
    int16_t  R1;
    int16_t  I2;
    int16_t  R2;
} fft_data_t
```

À ce stade, le lecteur attentif se demandera pourquoi on attend deux composantes appelées R et I. Rappelons que R signifie réel et I imaginaire.
La transformation de Fourier est définie en mathématiques classiques, c'est-à-dire symboliques, comme une opération basée sur des nombres complexes. Les algorithmes FFT attendent donc qu'on leur fournisse des composantes réelles et imaginaires, du moins dans la version générique, que les développeurs de matériel aiment intégrer dans le matériel pour obtenir une flexibilité maximale. Or, comme nos informations sonores comportent des éléments exclusivement réels, à ce stade nous mettons toujours à zéro les valeurs de I :

```
    for ( i = 0; i < FFT_N / 2; ++i)
    {
        input_data = (fft_data_t *)&buffer_input[i];
        input_data->R1 = rx_buf[2*i];
        input_data->I1 = 0;
        input_data->R2 = rx_buf[2*i+1];
        input_data->I2 = 0;
    }
```

C'est là que le «recyclage» du processus FFT devient pratique. Le classique *Numerical Recipies in C* [5], PDF disponible en ligne gratuitement ou à un prix modique, propose dans le douzième chapitre (de la deuxième édition) quelques formules intéressantes, mais qui requièrent de solides connaissances en maths. La récompense serait un processus FFT qui traiterait deux fonctions réelles en même temps. Il ne reste plus au développeur qu'à séparer les informations de sortie, ce qui n'est pas difficile.
La FFT est donc un de ces domaines où l'électronicien peut s'en donner à cœur joie – en fait, cette approche pratique est souvent plus efficace que l'épuisant effort d'apprentissage des maths.

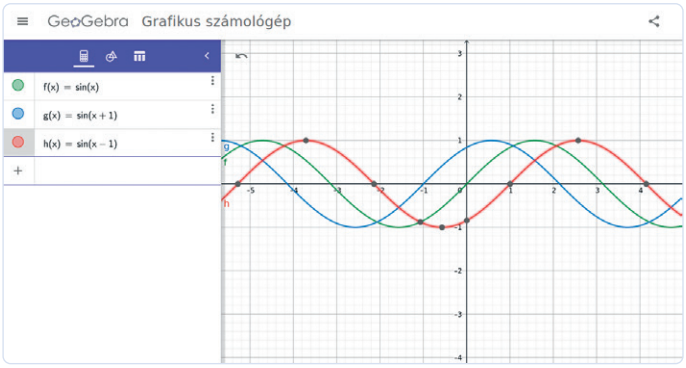


Figure 6. Malgré une fréquence et une amplitude identiques, ces oscillations ne sont pas identiques.

La prochaine étape consiste à exécuter les commandes FFT pour obtenir ensuite les résultats :

Notez ici le passage de l'opération `FFT_DIR_FORWARD`, qui informe le moteur matériel que nous passons du domaine temporel au domaine fréquentiel. En pratique, il y aura toujours des situations dans lesquelles vous aurez à transférer un signal du domaine des fréquences dans le domaine temporel. Dans ce cas, c'est la constante `FFT_DIR_BACKWARD` qui serait passée ici. Dans tous les cas, les données sont récoltées ensuite :

```
for ( i = 0; i < FFT_N / 2; i++)
{
    output_data = (fft_data_t*)&buffer_output[i];
    data_hard[2 * i].imag = output_data->I1 ;
    data_hard[2 * i].real = output_data->R1 ;
    data_hard[2 * i + 1].imag = output_data->I2 ;
    data_hard[2 * i + 1].real = output_data->R2 ;
}
```

Les informations renvoyées ont une partie réelle et une partie imaginaire. Pour comprendre ce problème, revenons encore une fois (brièvement) aux maths. La formule

$$S_f(t) = a_0/2 \Sigma [a_n \cos(n\omega t) + b_n \sin(n\omega t)]$$

montre l'une des nombreuses représentations de la série fondamentale de Fourier. L'utilisation de I ou J dans la formule nous montre que nous travaillons avec des unités complexes – puisque la FFT n'est en fait qu'une représentation numérique de la transformée de Fourier classique, elle doit suivre les règles.

Pourquoi complexe ?

L'échantillonnage effectué est certes réel, mais il ne fait pas le bonheur de la mathématicienne. L'une des raisons en est qu'il n'existe pas de déclencheur «défini» – du point de vue de la mathématicienne, les signaux sinusoïdaux de la **fig. 6** sont déphasés. L'opération FFT exprime cela par le fait que les valeurs renvoyées représentent des nombres complexes. Cela permet d'exprimer à la fois la puissance

et le déphasage de la composante respective – les deux sinusoïdes seraient également différentes dans le résultat.
L'électronicien préfère généralement ignorer les composantes complexes, que nous détruisons donc en calculant la valeur absolue :

```
float pmax=10;
for (i = 0; i < FFT_N; i++)
{
   hard_power[i] = sqrt(data_hard[i].real * data_
   hard[i].real + data_hard[i].imag * data_hard[i].
   imag);
```

Simplifions logarithmes et racines carrées, de façon à optimiser les calculs. Simplifier n'implique pas pour autant des procédures plus faciles à comprendre, au contraire. Par souci de clarté, nous renonçons à la simplification ici.
Si nous effectuons la combinaison déjà mentionnée en une opération FFT de deux fonctions différentes, il faudrait à ce stade effectuer une extraction et séparer les deux résultats (par un algorithme simple comparé à la FFT). La prochaine étape est une logarithmisation pour obtenir des valeurs de l'ordre du dB :

```
    hard_power[i] = 20*log10(2*hard_power[i]/FFT_N);
    if( hard_power[i]>pmax && i>5)
        pmax = hard_power[i];
}
```

Puis il faut afficher les valeurs. Le principal inconvénient est ici l'API d'affichage de l'Arduino, et les lourdes pertes de performance lors de la transmission de valeurs de coordonnées *invalides* :

```
    lcd.setRotation(0);
    lcd.fillScreen(COLOR_WHITE);
    for (int i=0;i<256;i++)
    {
      int dval = 240 - 240*(hard_power[i]/pmax);
      if (dval<2)dval=1;
      if (dval>238)dval=238;
      lcd.drawLine( i,  240, i, dval  , COLOR_RED);
    }
    delay(50);
}
```

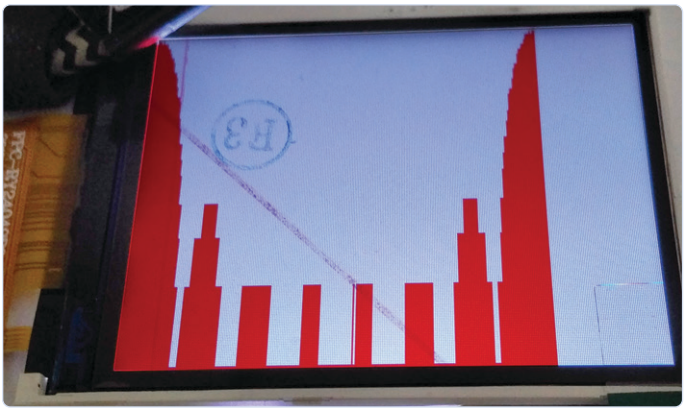


Figure 7. Cette «symétrie» est déroutante.

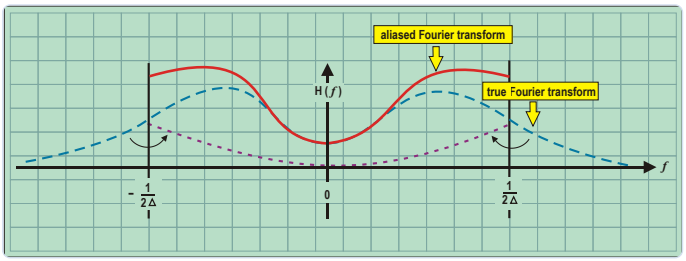


Figure 8. Les éléments spectraux «éliminés» de la fenêtre de Nyquist apparaissent comme des artefacts.

Nous voici prêts pour un premier essai (**fig. 7**). Ceux qui le souhaitent peuvent à ce stade ajouter au système des tonalités de test. La récompense de cet effort est la possibilité de déplacer sur l'écran la tonalité de test ou la ligne qui la représente.

De la réflexion

Le FFT est un sujet parmi les plus chronophages pour l'électronicien. Les observateurs attentifs remarqueront que le signal de test parcourt l'écran en partant aussi bien de gauche que de droite.
Plusieurs obstacles nous attendent. Notre FFT se répète donc dès le 128e champ, alors que, selon les théorèmes qui viennent d'être établis, cela devrait ne se produire qu'au 256e champ. La raison en

IL Y A PYTHON ET PYTHON

Dans le domaine de Python pour microcontrôleurs, deux concepts sont en compétition. L'un est MicroPython, qui met en œuvre un *runtime* complet. Le système cible contient un interpréteur Python complet, qui charge même des bibliothèques depuis l'internet, comme sur une station de travail. Cette approche a l'avantage d'une plus grande flexibilité, l'inconvénient de problèmes causés par des ressources limitées (RAM).

L'autre n'est actuellement représentée que par Zerynth, indisponible sur Maixduino. Le poste de travail crée un monolithe qui est transféré sur le système cible. À la flexibilité moindre s'oppose une meilleure stabilité – si les bibliothèques sont sur le poste de travail, les processus gourmands seront traités en dehors du microcontrôleur.

est un problème de livraison des échantillons via le bus I2S, un sujet que nous laisserons de côté maintenant. Vous trouverez sous [6] une version modifiée qui *atténue* ce problème.
Insistons sur le choix du verbe *atténuer*, car il y aura toujours une *réflexion* dans le cas d'une FFT avec des valeurs d'entrée entièrement réelles. Les valeurs de zéro à N/2 correspondent à celles de N/2+1 à N. La raison en est que les parties complexes de la FFT doivent s'annuler mutuellement.
Pour d'autres recours à la FFT, les domaines sont nombreux, par exemple l'introduction d'une fonction de fenêtre pour «découper» l'entrée. L'idée sous-jacente est que les parties situées à l'extérieur de la fenêtre de Nyquist peuvent être repliées ou réfléchies vers l'intérieur de part et d'autre, alors qu'une fonction de fenêtre supprimerait ce phénomène (**fig. 8**). Je recommande à ce sujet la lecture de *Some Windows with Very Good Sidelobe Behavior* [7], publié par Albert H. Nuttall en 1981.

Conclusion

Le moteur FFT matériel permet des expériences assez accessibles avec des algorithmes et des applications FFT. Ce ne sont pas les applications qui manquent. Vous ne devriez regretter ni le temps que vous aurez consacré ni l'argent que vous aurez investi.

200297-04 — VF : Denis Meyer

Produits

- **Sipeed MAix BiT Kit for RISC-V AI+IoT**
  www.elektor.fr/sipeed-maix-bit-kit-for-risc-v-ai-iot

LIENS

[1] Kendryte K210 : https://bit.ly/3fptbHu
[2] Standalone-SDK : https://github.com/kendryte/kendryte-standalone-sdk
[3] ArduinoCore-k210 : https://github.com/Seeed-Studio/ArduinoCore-k210
[4] fft_lcd : https://github.com/MrJBSwe/fft_lcd/blob/master/main.c
[5] Teukolsky, Press, Vetterling & Flannery: Numerical Recipes in C : http://www.numerical.recipes/
[6] stft_lcd : https://github.com/jpiat/stft_lcd
[7] Albert H. Nuttall : Some Windows with Very Good Sidelobe Behavior : https://bit.ly/3fTkCWb

LISTAGE 1. LISTAGE COMPLET DE MAIXDUINO.

```
#include <Sipeed_ST7789.h>
#include "i2s.h"
#include "fft.h"
SPIClass spi_(SPI0); // MUST be SPI0 for Maix series on board LCD
Sipeed_ST7789 lcd(320, 240, spi_);
uint32_t rx_buf[1024];
#define FFT_N               512U
#define FFT_FORWARD_SHIFT   0x0U
#define SAMPLE_RATE         38640
int16_t  real[FFT_N];
int16_t  imag[FFT_N];
int    hard_power[FFT_N];
uint64_t fft_out_data[FFT_N / 2];
uint64_t buffer_input[FFT_N];
uint64_t buffer_output[FFT_N];
fft_data_t *output_data;
fft_data_t *input_data;
complex_hard_t data_hard[FFT_N] = ;
#define FRAME_LEN           512

void setup()
 {
     lcd.begin(15000000, COLOR_RED);
```

suite au verso

```
    fpioa_set_function(20, FUNC_I2S0_IN_D0);
    fpioa_set_function(18, FUNC_I2S0_SCLK);
    fpioa_set_function(19, FUNC_I2S0_WS);

   i2s_init(I2S_DEVICE_0, I2S_RECEIVER, 0x3);
   i2s_set_sample_rate(I2S_DEVICE_0, SAMPLE_RATE );
   i2s_rx_channel_config(I2S_DEVICE_0, I2S_CHANNEL_0,
                         RESOLUTION_16_BIT, SCLK_CYCLES_32,
                         TRIGGER_LEVEL_4, STANDARD_MODE);

    dmac_init();
    sysctl_enable_irq();
}
void loop()
 {
   int i, sampleID;

   i2s_receive_data_dma(I2S_DEVICE_0, &rx_buf[0], FRAME_LEN * 2, DMAC_CHANNEL3);
    dmac_wait_idle(DMAC_CHANNEL3);//wait to finish recv

    for ( i = 0; i < FFT_N / 2; ++i)
    {
        input_data = (fft_data_t *)&buffer_input[i];
        input_data->R1 = rx_buf[2*i];   // data_hard[2 * i].real;
        input_data->I1 = 0;             // data_hard[2 * i].imag;
        input_data->R2 = rx_buf[2*i+1]; // data_hard[2 * i + 1].real;
        input_data->I2 = 0;             // data_hard[2 * i + 1].imag;
    }

   fft_complex_uint16_dma(DMAC_CHANNEL0, DMAC_CHANNEL1, FFT_FORWARD_SHIFT, FFT_DIR_FORWARD,
                                                        buffer_input, FFT_N, buffer_output);
   for ( i = 0; i < FFT_N / 2; i++)
    {
        output_data = (fft_data_t*)&buffer_output[i];
        data_hard[2 * i].imag = output_data->I1 ;
        data_hard[2 * i].real = output_data->R1 ;
        data_hard[2 * i + 1].imag = output_data->I2 ;
        data_hard[2 * i + 1].real = output_data->R2 ;
    }
   float pmax=10;
    for (i = 0; i < FFT_N; i++)
    {
        hard_power[i] = sqrt(data_hard[i].real * data_hard[i].real +
                                          data_hard[i].imag * data_hard[i].imag);
       //Convert to dBFS
        hard_power[i] = 20*log10(2*hard_power[i]/FFT_N);
       if( hard_power[i]>pmax && i>5)
            pmax = hard_power[i];
    }

    lcd.setRotation(0);
    lcd.fillScreen(COLOR_WHITE);
    for (int i=0;i<256;i++)
    {
      int dval = 240 - 240*(hard_power[i]/pmax);
      if (dval<2)dval=1;
      if (dval>238)dval=238;
      lcd.drawLine( i,  240, i, dval  , COLOR_RED);
    }
    delay(50);
}
```

embeddedworld2023

Exhibition&Conference

... it´s a smarter world

UNITED

JOIN THE EMBEDDED COMMUNITY

14–16.3.2023

Get your
free ticket now!

embedded-world.com/voucher

Use the voucher code **GG4ew23**

Media partners

Markt&Technik

Elektronik

SmarterWorld

DESIGN& ELEKTRONIK

Elektronik automotive

medical-design

computer & automation

elektroniknet.de

NÜRNBERG MESSE