
LIVRET DE TRAVAUX PRATIQUES
CALCUL EMBARQUÉ



Département Électronique
ECE – Programme Grande École d'Ingénieurs

Version du 15 avril 2025

Programme des séances de TP

— Arduino DUE et SAM3X8E —



- TP 0** - Fondamentaux de l'Arduino Due p. 3
- TP 1** - Numérisation rapide sur Arduino Due p. 16

— Méthodes de Calcul Numérique —



- TP 2** - Racines d'équations p. 20
- TP 3** - Optimisation p. 24
- TP 4** - Équations différentielles p. 27

— Intelligence Artificielle —



- TP 5** - Outils mathématiques pour le Machine Learning p. 30
- TP 6** - Introduction à l'apprentissage machine p. 33
- TP 7** - Les réseaux de neurones p. 35
- TP 8** - Deep Learning et Réseau de Neurones Convolutionnel p. 38
- TP 9** - FFT et MFCC p. 42
- TP 10** - Filtre numérique RIF p. 47

TP n° 0

Fondamentaux de l'Arduino Due

Motivation

L'objectif de ce TP n°0 est de rappeler les acquis du **Cours 1 : Acquisition sur microcontrôleur**. Le traitement numérique du signal exige la maîtrise du temps d'exécution et de la fréquence d'échantillonnage lors de l'acquisition de signaux analogiques. Il est nécessaire de passer au niveau des registres pour configurer les différents périphériques de la carte Arduino Due. Ainsi, nous verrons qu'il existe des méthodes optimisées afin de remplacer la fonction **analogRead()** ou encore pour optimiser les précieux cycles d'horloges de notre microcontrôleur.

I. Arduino Due - Arm Cortex-M3 SAM3X8E

L'Arduino Due est une carte de développement disposant d'un microcontrôleur 32-bit Arm Cortex-M3 SAM3X8E. La datasheet de ce microcontrôleur est disponible ici : [Datasheet SAM3X](#). Des références à des pages précises de la datasheet seront données dans les prochaines sections afin de vous guider parmi les 1459 pages de cette dernière. La datasheet est à consommer sans modération.

La carte Arduino Due est présentée ci-dessous. La programmation de la carte Arduino Due s'effectuera par le port **Programming** à côté de la prise DC Jack 9V. La vitesse maximale de la communication série est de **460800 baud**.

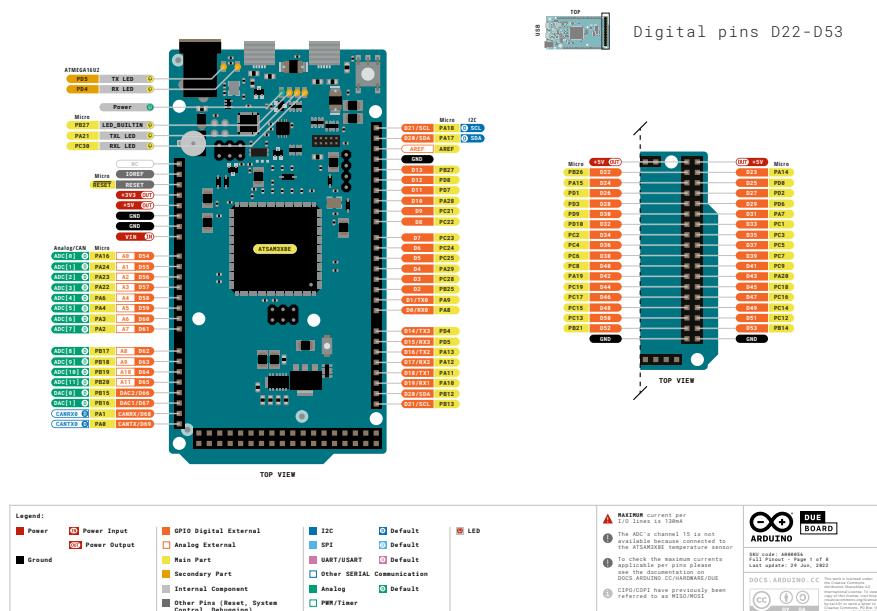


FIGURE 1 – Pinout de la carte Arduino Due

A. Activation des périphériques et interruptions

Le Arm Cortex-M3 SAM3X possède un **Peripheral Management Controller** ou PMC. Il permet d'activer ou non les différents périphériques du microcontrôleur. Certains périphériques, comme ceux que nous allons voir dans les prochaines sections, sont désactivés par défaut. La liste de l'identifiant correspondant au périphérique souhaité est disponible à la [page 38](#) de la datasheet. Dans la majorité des travaux pratiques, nous utiliserons l'ADC (PID37), le DAC (PID38), le contrôleur DMA (PID39) et des timers/counters (PID27 à PID35).

Le microcontrôleur est sur 32 bits. Ainsi, deux registres PMC sont nécessaires pour contrôler les 44 périphériques (1 bit par périphérique) : **PMC_PCER0** (p.542) et **PMC_PCER1** (p.563).

28.15.4 PMC Peripheral Clock Enable Register 0

Name: **PMC_PCER0**
Address: 0x400E0610
Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| PID31 | PID30 | PID29 | PID28 | PID27 | PID26 | PID25 | PID24 |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| PID23 | PID22 | PID21 | PID20 | PID19 | PID18 | PID17 | PID16 |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| PID15 | PID14 | PID13 | PID12 | PID11 | PID10 | PID9 | PID8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PID7 | PID6 | PID5 | PID4 | PID3 | PID2 | — | — |

This register can only be written if the WPEN bit is cleared in "PMC Write Protect Mode Register".

- **PIIdx: Peripheral Clock x Enable**

0 = No effect.

1 = Enables the corresponding peripheral clock.

Note: To get PIIdx refer to identifiers as defined in the section "Peripheral Identifiers" in the product datasheet. Other peripherals can be enabled in PMC_PCER1 (Section 28.15.23 "PMC Peripheral Clock Enable Register 1").

Note: Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

(a) Registre PMC_PCER0

28.15.23 PMC Peripheral Clock Enable Register 1

Name: **PMC_PCER1**
Address: 0x400E0700
Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|-------|-------|-------|-------|-------|-------|-------|-------|
| — | — | — | — | — | — | — | — |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| — | — | — | — | — | — | — | — |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| — | — | — | — | PID44 | PID43 | PID42 | PID41 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PID39 | PID38 | PID37 | PID36 | PID35 | PID34 | PID33 | PID32 |

This register can only be written if the WPEN bit is cleared in "PMC Write Protect Mode Register".

- **PIIdx: Peripheral Clock x Enable**

0 = No effect.

1 = Enables the corresponding peripheral clock.

Notes: 1. To get PIIdx refer to identifiers as defined in the section "Peripheral Identifiers" in the product datasheet.

2. Programming the control bits of the Peripheral ID that are not implemented has no effect on the behavior of the PMC.

(b) Registre PMC_PCER1

FIGURE 2 – Registres de contrôle des périphériques

Le Arm Cortex-M3 SAM3X possède un **Nested Vectored Interrupt Controller** ou NVIC. Le NVIC permet de gérer de 1 à 30 interruptions matérielles de l'Arduino Due avec 16 niveaux de priorités. Une interruption matérielle déclenchera une interruption de l'exécution en cours afin d'exécuter l'Interrupt Service Routine (ISR) correspondante (souvent nommée `<Periph>_Handler()`).

Une fois l'ISR terminée, le programme reprend exactement là où il en était. Il est néanmoins **nécessaire de clear** (mettre à 0) le drapeau d'interruption avant de quitter l'interruption. Dans le cas contraire, le programme ne pourra pas retourner à l'endroit où il en était avant l'interruption. La page 164 de la datasheet nous indique comment programmer ce contrôleur NVIC en C/C++.

10.20.10.1 NVIC programming hints

Software uses the CPSIE I and CPSID I instructions to enable and disable interrupts. The CMSIS provides the following intrinsic functions for these instructions:

```
void __disable_irq(void) // Disable Interrupts
void __enable_irq(void) // Enable Interrupts
```

In addition, the CMSIS provides a number of functions for NVIC control, including:

Table 10-29. CMSIS functions for NVIC control

| CMSIS interrupt control function | Description |
|---|---|
| void NVIC_SetPriorityGrouping(uint32_t priority_grouping) | Set the priority grouping |
| void NVIC_EnableIRQ(IRQn_t IRQn) | Enable IRQn |
| void NVIC_DisableIRQ(IRQn_t IRQn) | Disable IRQn |
| uint32_t NVIC_GetPendingIRQ (IRQn_t IRQn) | Return true if IRQn is pending |
| void NVIC_SetPendingIRQ (IRQn_t IRQn) | Set IRQn pending |
| void NVIC_ClearPendingIRQ (IRQn_t IRQn) | Clear IRQn pending status |
| uint32_t NVIC_GetActive (IRQn_t IRQn) | Return the IRQ number of the active interrupt |
| void NVIC_SetPriority (IRQn_t IRQn, uint32_t priority) | Set priority for IRQn |
| uint32_t NVIC_GetPriority (IRQn_t IRQn) | Read priority of IRQn |
| void NVIC_SystemReset (void) | Reset the system |

For more information about these functions see the CMSIS documentation.

FIGURE 3 – Programmation en C du contrôleur NVIC sur Arduino Due

B. Configuration des Timers/Counters

Le Arm Cortex-M3 SAM3X8E possède 9 timers/counters 32-bits soit $2^{31} - 1$ pour la valeur maximale d'un timer. Le timer s'incrémenté dans le registre **TC_CV** correspondant selon la clock (interne ou externe) et son front (montant ou descendant) sélectionné. Lorsque le timer atteint sa valeur maximale, le drapeau d'interruption **TC_SR** est mis à l'état haut. Ainsi, les timers peuvent générer une interruption matérielle si l'on active l'interruption à partir du NVIC (**TCx_IRQn**). Toutes ces informations générales sont disponibles à la page 860 de la datasheet.

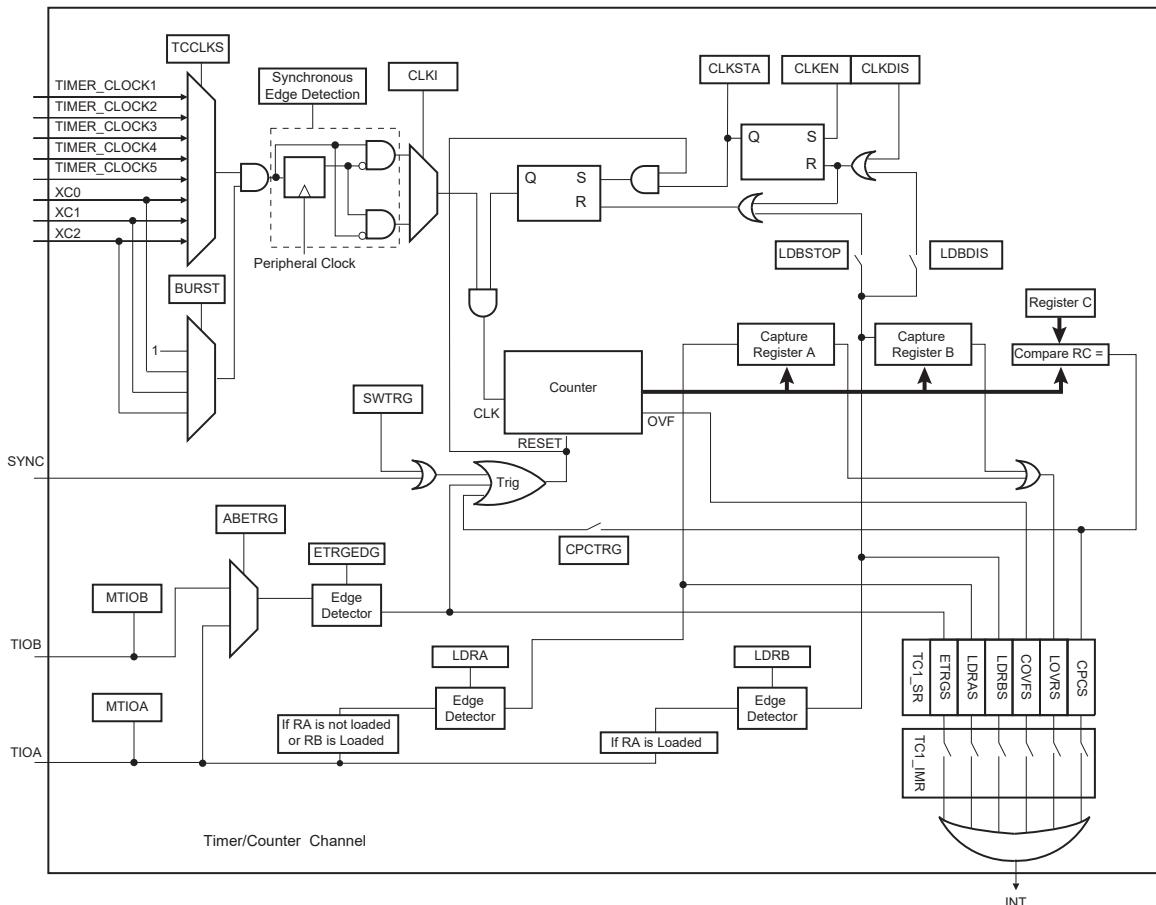


FIGURE 4 – Block diagram du Timer/Counter en Capture Mode p.864

Il est important d'avoir connaissance de la fréquence de la Master Clock (MCK) de la carte de développement que vous utilisez puisque les timers dépendent de cette source. La Master Clock de l'Arduino Due est à 84 MHz. La configuration du timer s'effectue par le registre **TC_CMRx**.

36.7.2 TC Channel Mode Register: Capture Mode

Name: TC_CMRx [x=0..2] (CAPTURE_MODE)
Address: 0x40080004 (0)[0], 0x40080044 (0)[1], 0x40080084 (0)[2], 0x40084004 (1)[0], 0x40084044 (1)[1],
0x40084084 (1)[2], 0x40088004 (2)[0], 0x40088044 (2)[1], 0x40088084 (2)[2]

Access: Read/Write

| | | | | | | | |
|--------|---------|-------|------|--------|--------|---------|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| - | - | - | - | - | - | - | - |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| - | - | - | - | LDRB | LDRA | - | - |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| WAVE | CPCTRG | - | - | - | ABETRG | ETRGEDG | - |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| LDBDIS | LDBSTOP | BURST | CLKI | TCCLKS | | | |

This register can only be written if the WPEN bit is cleared in the [TC Write Protection Mode Register](#)

FIGURE 5 – Registre TC_CMRx p.881

Le bit **WAVE** (bit n°15) de ce registre permet de configurer le timer en mode Capture ou Waveform.

- Mode Capture (bit = 0) : Mesurer des signaux internes ou externes (nombre de fronts montants ou descendants)
- Mode Waveform (bit = 1) : Générer des signaux PWM

Les bits **TCCLKS**, dans le capture mode (bits n°0 à 2) permettent de sélectionner la clock d'entrée.

- Timer_Clock1 (000) : internal MCK/2 (Master Clock)
- Timer_Clock2 (001) : internal MCK/8 (Master Clock)
- Timer_Clock3 (010) : internal MCK/32 (Master Clock)
- Timer_Clock4 (011) : internal MCK/128 (Master Clock)
- Timer_Clock5 (100) : internal SLCK (Slow clock)
- XC0 (101)
- XC1 (110)
- XC2 (111)

Le bit **CPCTRG** (n°14) permet d'activer le trigger selon la comparaison avec RC Compare. Ainsi, une fois la valeur du registre RC atteinte, le timer est reset.

- 0 : désactivé
- 1 : activé

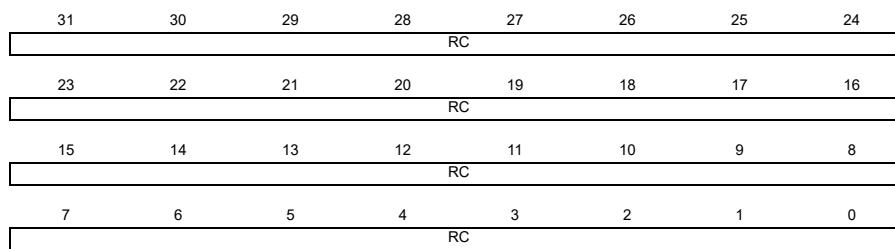
La valeur du registre RC est modifiable par l'utilisateur dans le registre **TC_RC** suivant. Il s'agit toujours d'un registre de 32 bits.

36.7.8 TC Register C

Name: TC_RCx [x=0..2]

Address: 0x4008001C (0)[0], 0x4008005C (0)[1], 0x4008009C (0)[2], 0x4008401C (1)[0], 0x4008405C (1)[1],
 0x4008409C (1)[2], 0x4008801C (2)[0], 0x4008805C (2)[1], 0x4008809C (2)[2]

Access: Read/Write



This register can only be written if the WPEN bit is cleared in the [TC Write Protection Mode Register](#).

- **RC: Register C**

RC contains the Register C value in real time.

FIGURE 6 – Registre TC_RC p.891

Sur la Figure 4, on observe que le bit **CPCS** doit être activé pour générer une interruption sur RC Compare. Le registre **TC_IERx** permet d'activer ce bit et donc l'interruption en question. Ce registre comporte 8 modes d'interruptions possibles. Nous nous servirons exclusivement de **CPCS**. Afin de fixer une fréquence pour un timer en utilisant RC Compare en tant qu'interruption, la formule pour calculer RC est la suivante avec F la fréquence désirée et le prescaler mis pour les bits **TCCLKS** :

$$RC = \frac{\left(\frac{MCK}{\text{Prescaler}}\right)}{F} \quad (1)$$

La lecture du registre **TC_SR** permet de reset le flag d'interruption et ainsi de permettre une nouvelle interruption d'avoir lieu.

36.7.10 TC Interrupt Enable Register

Name: TC_IERx [x=0..2]
Address: 0x40080024 (0)[0], 0x40080064 (0)[1], 0x400800A4 (0)[2], 0x40084024 (1)[0], 0x40084064 (1)[1],
0x400840A4 (1)[2], 0x40088024 (2)[0], 0x40088064 (2)[1], 0x400880A4 (2)[2]

Access: Write-only

| | | | | | | | |
|-------|-------|-------|------|------|------|-------|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| - | - | - | - | - | - | - | - |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| - | - | - | - | - | - | - | - |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| - | - | - | - | - | - | - | - |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ETRGS | LDRBS | LDRAS | CPCS | CPBS | CPAS | LOVRS | COVFS |

- **CPCS:** RC Compare

0: No effect.

1: Enables the RC Compare Interrupt.

FIGURE 7 – Registre TC_IERx p894

Enfin, le registre **TC_CCR** permet de contrôler la clock et de démarrer le timer à l'aide d'un software reset. Il s'agit de la dernière commande à lancer lors du setup. Il faudra alors activer la clock **CLKEN** et **SWTRG** pour démarrer le timer.

36.7.1 TC Channel Control Register

Name: TC_CCRx [x=0..2]
Address: 0x40080000 (0)[0], 0x40080040 (0)[1], 0x40080080 (0)[2], 0x40084000 (1)[0], 0x40084040 (1)[1],
0x40084080 (1)[2], 0x40088000 (2)[0], 0x40088040 (2)[1], 0x40088080 (2)[2]

Access: Write-only

| | | | | | | | |
|----|----|----|----|----|-------|--------|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| - | - | - | - | - | - | - | - |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| - | - | - | - | - | - | - | - |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| - | - | - | - | - | - | - | - |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| - | - | - | - | - | SWTRG | CLKDIS | CLKEN |

- **CLKEN:** Counter Clock Enable Command

0: No effect.

1: Enables the clock if CLKDIS is not 1.

- **CLKDIS:** Counter Clock Disable Command

0: No effect.

1: Disables the clock.

- **SWTRG:** Software Trigger Command

0: No effect.

1: A software trigger is performed: the counter is reset and the clock is started.

FIGURE 8 – Registre TC_CCR p.880

Le reste des registres ne nous sera pas utile pour la suite des Timers/Counters. Si toutefois la curiosité vous envahit, la datasheet reste votre meilleur compagnon.

C. Configuration de l'ADC

Lors de vos travaux pratiques précédents, vous avez travaillé avec la fonction **analogRead()**. Il s'agissait d'une boîte noire magique sans aucun contrôle sur les différents paramètres de configuration de l'ADC. Nous allons voir comment est-il possible de configurer votre ADC comme bon vous semble.

Comme discuté en cours, l'ADC dans le Cortex-M3 SAM3X8E est un ADC **12-bit** pouvant fonctionner à **1 MHz**. Son numéro de périphérique est le 37. Il s'agira de l'activer afin d'activer l'ADC.

Attention, l'ADC de l'Arduino Due a une tension maximale de 3,3 V (ADVref). Dépasser cette valeur pourrait endommager de manière irréversible la carte.

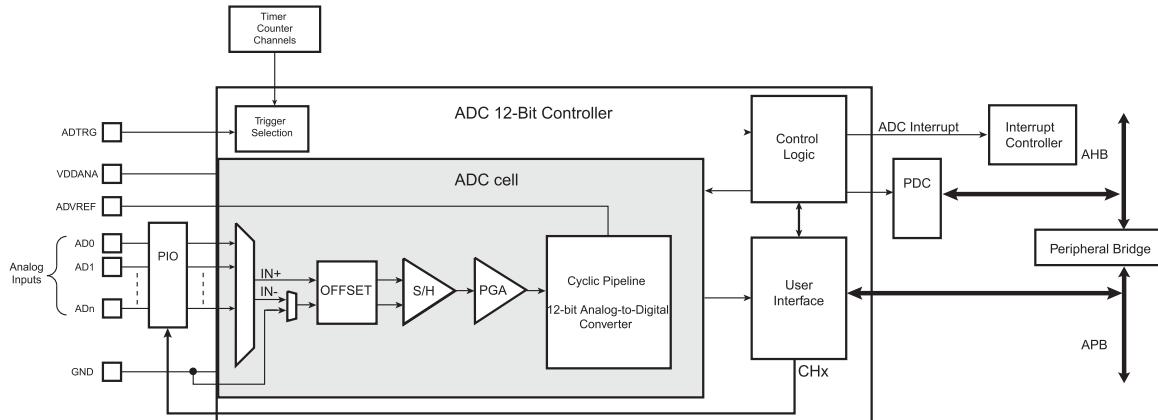


FIGURE 9 – Block diagram de l'ADC sur SAM3X, p.1318

Sur le block diagram, du côté des entrées, nous observons **Trigger Selection** qui prend ADTRG et Timer/Counter en entrée. Ce sous-module nous permet de lancer les conversions selon des événements extérieurs. Il serait effectivement pratique de pouvoir contrôler la fréquence d'échantillonnage afin de respecter le théorème de Nyquist-Shannon.

Le registre **ADC_CR** permet de lancer manuellement une conversion en mettant à 1 le bit **START**. Une fois la conversion terminée, l'ADC change le status d'un drapeau d'interruption dans le registre **ADC_ISR**.

43.7.1 ADC Control Register

Name: ADC_CR

Address: 0x400C0000

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|----|----|----|----|----|----|-------|-------|
| - | - | - | - | - | - | - | - |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| - | - | - | - | - | - | - | - |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| - | - | - | - | - | - | - | - |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| - | - | - | - | - | - | START | SWRST |

- **SWRST: Software Reset**

0 = No effect.

1 = Resets the ADC simulating a hardware reset.

- **START: Start Conversion**

0 = No effect.

1 = Begins analog-to-digital conversion.

FIGURE 10 – Registre ADC Control p.1332

On observe également que les entrées analogiques sont derrières un multiplexeur. Il n'est pas possible d'effectuer la conversion de toutes les entrées simultanément. **Un cours délai doit être conservé**

entre le changement de channel et le démarrage de la conversion analogique (ex : A2 à A7). Le non respect de ce délai peut provoquer des erreurs de conversions. Le registre **ADC_MR** ou ADC Mode Register permet de configurer l'ADC. Par défaut, tous les bits sont mis à 0.

43.7.2 ADC Mode Register

Name: ADC_MR

Address: 0x400C0004

Access: Read-write

| | | | | | | | |
|---------|------|----------|--------|----|----------|----|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| USEQ | - | TRANSFER | | | TRACKTIM | | |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| ANACH | - | SETTLING | | | STARTUP | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| PRESCAL | | | | | | | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FREERUN | FWUP | SLEEP | LOWRES | | TRGSEL | | TRGEN |

This register can only be written if the WPEN bit is cleared in "ADC Write Protect Mode Register" on page 1353

FIGURE 11 – Registre ADC Mode p.1333

Le bit **TRGEN** (n°0) permet d'activer le déclenchement de la conversion de l'ADC selon un évènement matériel à configurer dans les prochains bits **TRGSEL**. Conserver le bit à 0 permet de déclencher la conversion manuellement par la mise à 1 du bit START du registre **ADC_CR**. Ce détail vous sera utile pour déclencher manuellement le début de la conversion à chaque interruption de votre Timer responsable de maintenir la fréquence d'échantillonnage.

Les bits **TRGSEL** (n°1 à 3) permettent de configurer l'évènement matériel source du déclenchement du début de la conversion de l'ADC uniquement si le bit **TRGEN** est mis à 1.

Le bit **LOWRES** (n°4) permet de configurer la résolution de l'ADC en 10 bits (1) ou 12 bits (0).

Le bit **FREERUN** (n°7) permet de configurer l'ADC pour qu'il fonctionne à pleine vitesse sans attendre un déclenchement automatique par interruption ou manuel. Le bit doit être à 1 pour le mettre en mode Free run.

Les bits **PRESCAL** (n°8 à 15) permettent d'appliquer un prescaler sur la clock de l'ADC selon l'équation suivante :

$$ADCClock = \frac{MCK}{(PRESCAL + 1) \times 2} \quad (2)$$

Les bits **STARTUP** (n°16 à 19) permettent d'appliquer un temps de démarrage d'un certains nombre de périodes de l'ADC Clock. Le détail des différentes valeurs possibles sont à la page 1334 de la datasheet. Pour modifier les valeurs des bits rapidement, on choisit la valeur en mettant SUTX avec X le nombre de périodes de l'ADC Clock.

Les bits **SETTLING** (n°20 et 21) permettent d'appliquer un temps de stabilisation de l'ADC. 4 valeurs sont possibles : AST3, AST5, AST9 et AST17.

Les bits **TRACKTIM** permettent d'appliquer un délai entre chaque basculement de l'entrée analogique de l'ADC. L'équation suivante permet de trouver la valeur du tracking time :

$$TrackingTime = (TRACKTIM + 1) \cdot ADC_{Clock_Periods} \quad (3)$$

Le reste des registres ne nous serons pas utile pour la suite de la configuration mais la datasheet fournie

l'ensemble des informations que vous désirez.

Nous avons vu que l'ensemble des entrées analogiques étaient derrière un multiplexeur. Le registre **ADC_CHER** permet d'activer les entrées analogiques à convertir. Les données converties seront enregistrées dans le registre **ADC_CDR** du channel correspondant.

43.7.5 ADC Channel Enable Register

Name: ADC_CHER

Address: 0x400C0010

Access: Write-only

| | | | | | | | |
|------|------|------|------|------|------|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
| - | - | - | - | - | - | - | - |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| - | - | - | - | - | - | - | - |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| CH15 | CH14 | CH13 | CH12 | CH11 | CH10 | CH9 | CH8 |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |

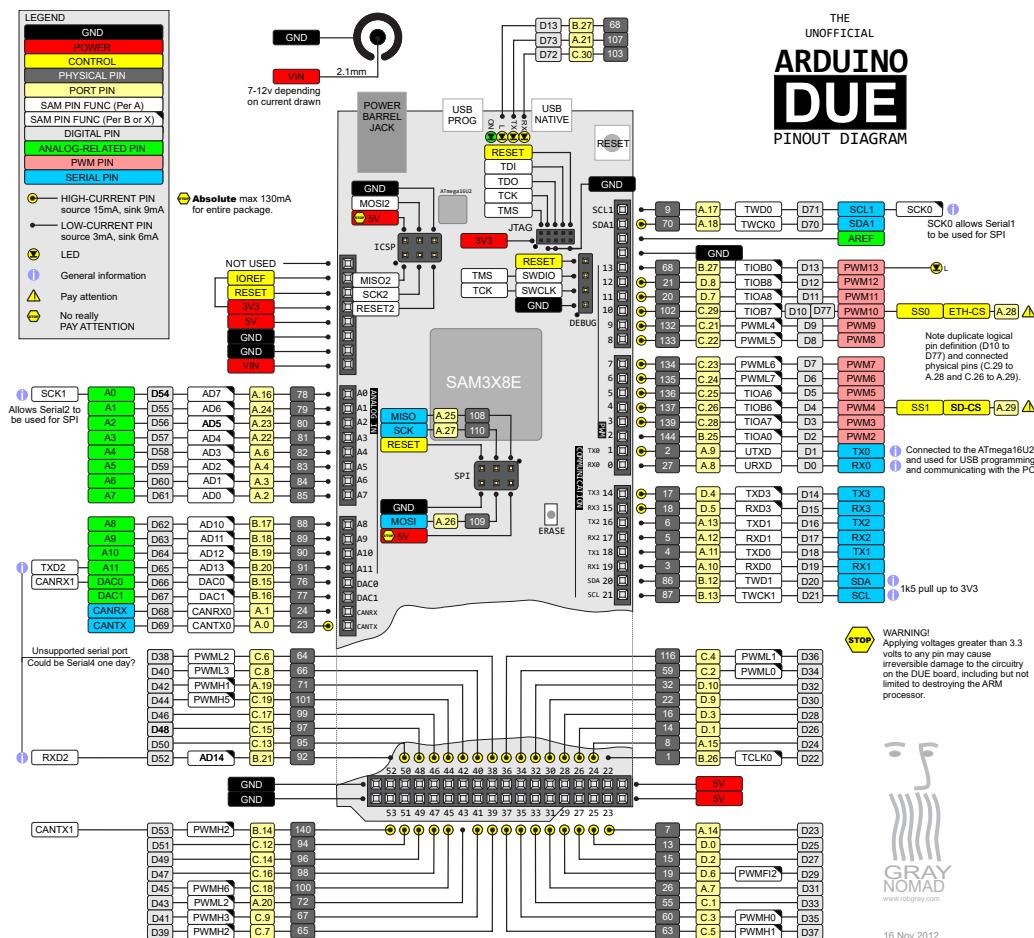
This register can only be written if the WPEN bit is cleared in "ADC Write Protect Mode Register" on page 1353.

- CHx: Channel x Enable

0 = No effect.

1 = Enables the corresponding channel.

FIGURE 12 – Registre ADC Enable p.1338



d'une utilisation sans DMA, nous pouvons utiliser le drapeau EOC pour End Of Conversion Interrupt Enable correspondant au channel que nous voulons lire (**ex : EOC7 pour le channel 7 de l'ADC au pin A0**). Avec la DMA, nous devons utiliser le drapeau d'interruption ENDRX pour End Of RX Buffer. Le détail est donné à la page 1345 de la datasheet.

D. Configuration du DAC

Le Cortex-M3 SAM3X8E inclut deux sorties digital-analogiques (DAC) avec une résolution de 12 bits. La carte Arduino Due est ainsi capable de prendre un signal digital en entrée et de le convertir en un signal analogique entre 0 et 3.3V avec 2^{12} niveaux différents.

Dans notre cas, la configuration du DAC s'effectue en assignant les bits des registres **DACC_MR**, **DACC_CHER** et **DACC_IER**. D'autres registres permettent une configuration du DAC que nous n'utiliserons pas ici. Le numéro de périphérique du DAC est le n°38.

Le registre **DACC_MR** sert à configurer le fonctionnement du DAC général. Ci-dessous les bits que nous utiliserons. La liste de l'ensemble des bits est disponibles à la page 1364 de la datasheet.

- **TRGEN** à 0 pour mettre le DAC en free running mode
- **WORD en Half** car notre micro-controller est en 32 bits mais nous n'envoyons que les 16 premiers bits à convertir.
- **USER_SEL** à 1 pour sélectionner le DAC n°1 de la carte Arduino Due
- **STARTUP** à 8 DAC_CLOCK
- **MAXS** speed mode activé
- **REFRESH** period la plus faible possible (1)

Le registre **DACC_CHER** est utilisé pour activer le(s) channel(s) souhaité(s). Un seul bit est utilisé dans l'ensemble du registre 32-bits.

- **CH0** (bit n°0) : si 1 alors activé sinon désactivé
- **CH1** (bit n°1) : si 1 alors activé sinon désactivé

Le registre **DACC_IER** permet d'activer les interruptions au niveau du DAC. Il est tout de même nécessaire d'activer l'interruption au niveau du NVIC.

- **TXRDY** : Transmit Buf Interrupt Enable
- **EOC** : End Of Conversion Interrupt Enable
- **ENDTX** : End of Transmit Buffer
- **TXBUIFE** : Transmit Buffer Empty Interrupt Enable

Nous utilisons ici le drapeau d'interruption TXRDY. Le drapeau d'interruption TXRDY nous indique la fin de l'envoi du buffer pour le DAC et ainsi la possibilité de passer à la conversion suivante.

E. Fonctionnement de la DMA

La DMA, ou « Direct Memory Access » est un dispositif permettant de transférer directement des données entre des périphériques d'un microcontrôleur et sa mémoire. L'utilisation de la DMA permet d'économiser des cycles d'horloges pouvant être utilisés pour effectuer d'autres tâches.

Pour le mettre en œuvre, le microcontrôleur a besoin d'un contrôleur spécifique au niveau hardware. Dans le cas du ARM Cortex-M3 SAM3X8E, il s'agit du PDC « Peripheral DMA Controller » et il possède également un contrôleur DMA central à 6 canaux.

L'architecture globale du microcontrôleur est la suivante :

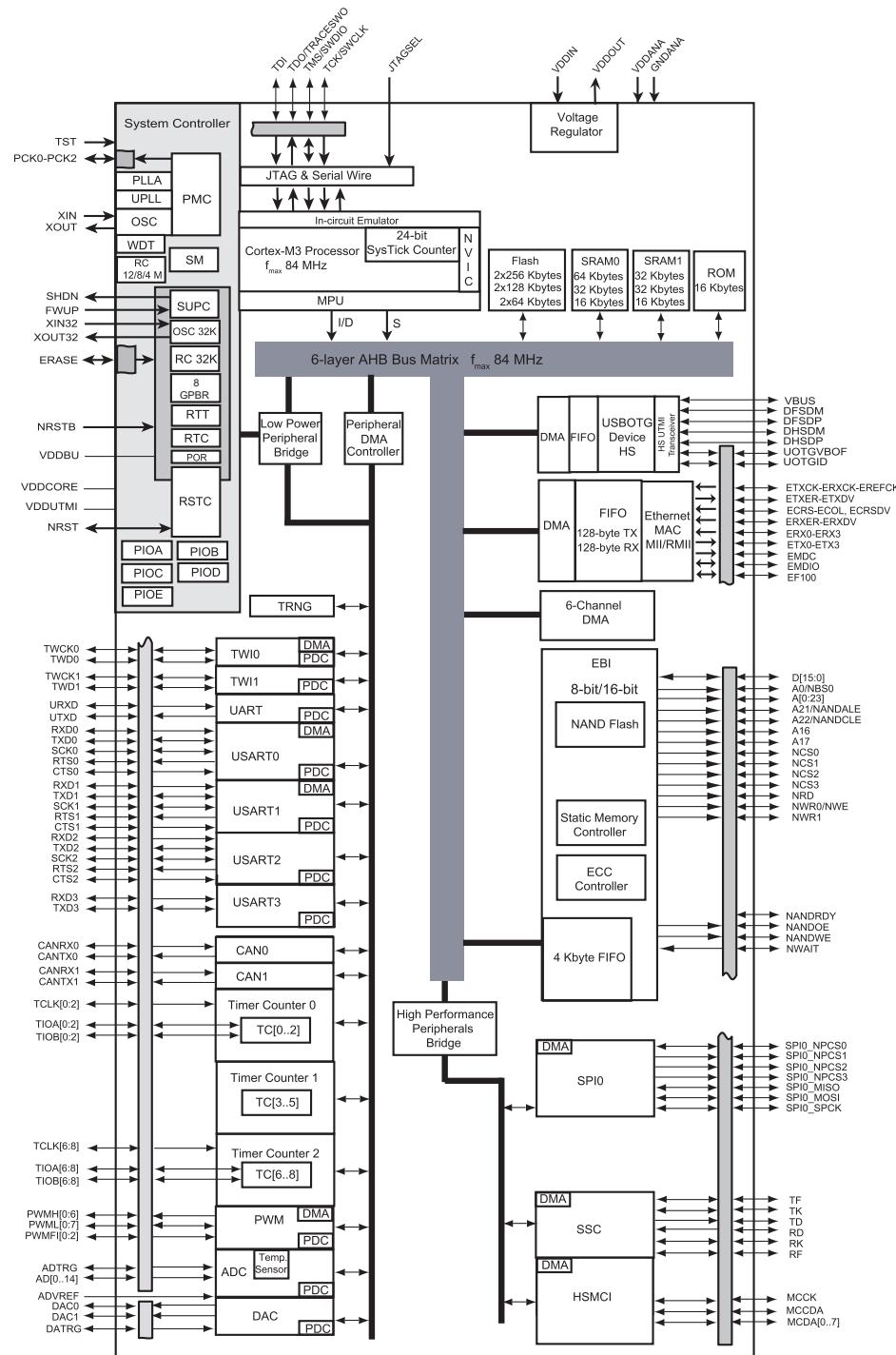


FIGURE 14 – Block diagram du SAM3X8E

Les channels du PDC sont définis à la page 504 de la datasheet :

| Instance Name | Channel T/R | 217 Pins ⁽¹⁾ | 144 Pins | 100 Pins |
|---------------|-------------|-------------------------|----------|----------|
| DAC | Transmit | X | X | X |
| PWM | Transmit | X | X | X |
| TWI1 | Transmit | X | X | X |
| TWI0 | Transmit | X | X | X |
| USART3 | Transmit | X | X | N/A |
| USART2 | Transmit | X | X | X |
| USART1 | Transmit | X | X | X |
| USART0 | Transmit | X | X | X |
| UART | Transmit | X | X | X |
| ADC | Receive | X | X | X |
| TWI1 | Receive | X | X | X |
| TWI0 | Receive | X | X | X |
| USART3 | Receive | X | X | N/A |
| USART2 | Receive | X | X | X |
| USART1 | Receive | X | X | X |
| USART0 | Receive | X | X | X |
| UART | Receive | X | X | X |

Note: 1. Corresponds to 217-pin SAM3X8H. This device is not commercially available. Mounted only on the SAM3X-EK evaluation kit.

FIGURE 15 – Tableau des périphériques connectés au PDC p.504

Chaque périphérique contient une interface utilisateur avec des pointeurs RCR, RNCR, TCR et TCNR puis RPR, RNPR, TPR et TNPR (p.506).

- La lettre R signifie Receive et la lettre T signifie Transmit. Le tableau des périphériques permet de connaître si le périphérique est en mode Receive ou Transmit
- La lettre P correspond au pointeur de la valeur.
- La lettre C correspond au compteur (buffersize)
- La lettre N correspond à Next et permet de pointer sur le prochain espace mémoire du buffer
- La dernière lettre R signifie register

Le transfert entre deux périphériques et la mémoire est contrôlé à l'aide du registre **PERIPH_PTCR** à la page 517 de la datasheet.

26.5.9 Transfer Control Register

Name: PERIPH_PTCR

Access: Write-only

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |
|----|----|----|----|----|----|--------|-------|
| - | - | - | - | - | - | - | - |
| 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| - | - | - | - | - | - | - | - |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| - | - | - | - | - | - | TXTDIS | TXTEN |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| - | - | - | - | - | - | RXTDIS | RXTEN |

- **RXTEN: Receiver Transfer Enable**

0: No effect.

1: Enables PDC receiver channel requests if RXTDIS is not set.

When a half duplex peripheral is connected to the PDC, enabling the receiver channel requests automatically disables the transmitter channel requests. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **RXTDIS: Receiver Transfer Disable**

0: No effect.

1: Disables the PDC receiver channel requests.

When a half duplex peripheral is connected to the PDC, disabling the receiver channel requests also disables the transmitter channel requests.

- **TXTEN: Transmitter Transfer Enable**

0: No effect.

1: Enables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, it enables the transmitter channel requests only if RXTEN is not set. It is forbidden to set both TXTEN and RXTEN for a half duplex peripheral.

- **TXTDIS: Transmitter Transfer Disable**

0: No effect.

1: Disables the PDC transmitter channel requests.

When a half duplex peripheral is connected to the PDC, disabling the transmitter channel requests disables the receiver channel requests.

FIGURE 16 – Transfer Control Register

II. Optimisation mémoire et rappels de C

La programmation embarqué se déroule en langage C/C++ puisqu'il s'agit de langages de programmations dit de « bas niveau ». Lors de la modification des registres, il est nécessaire d'utiliser des opérations bit-à-bit comme suivant :

- AND : &
- OR : |
- XOR : ^
- NOT : ~
- Left shift : <<
- Right shift : >>

L'opération binaire OR nous permet de set des registres sans modifier les autres.

Par exemple : PMC->PMC_PCER0 |= PMC_PCER0_PID27;

On peut également set plusieurs registres en même temps.

Par exemple : DACC->DACC_MR = DACC_MR_TRGEN_DIS | DACC_MR_USER_SEL_CHANNEL1 | DACC_MR_WORD_HALF;

Pour configurer les registres avec plusieurs périphériques, on peut utiliser la syntaxe suivante :

TCX->TC_CHANNEL[X].TC_<nom du registre> = <nom du bits>.

III. Visualisation et écoute de votre signal analogique

Lors du projet d'électronique, il vous sera utile de vérifier le signal analogique reçu à partir du microphone. Un tutoriel sur Audacity est disponible sur la page de la Toolbox sur Boostcamp.

Numérisation rapide sur Arduino Due

Motivation

L'objectif de ce TP est de comprendre le fonctionnement des périphériques tel que l'ADC ou le DAC qui seront des éléments cruciaux pour la suite et surtout pour le projet. Il s'agit du TP le plus important à maîtriser !

Au cours de ce TP, nous allons voir comment **maîtriser la fréquence d'échantillonnage de l'ADC de l'Arduino Due, gérer le transfert DMA et enfin, sa sortie sur le DAC**. La validation du bon fonctionnement de votre système ce fera à l'aide d'un oscilloscope et d'un GBF.

Le code est disponible sur le Github du Pôle Electronique :

<https://github.com/ElectroniqueECE/ECE-ELEC-TNS>

I. Maîtrise du temps

Dans cette partie, nous allons mettre en place un timer simple sur Arduino Due directement à partir des registres de l'Arm Cortex-M3 SAM3X8E. Le timer nous servira dans un second temps à gérer la fréquence d'échantillonage de notre acquisition de notre ADC.

T1. Quel est l'identifiant du périphérique timer TC0 ?

E1. Quels sont les différents registres à utiliser pour le contrôle du timer en utilisant une interruption sur RC Compare ? Expliquer votre raisonnement à l'aide du bloc diagramme Timer/Counter de la datasheet et créer votre code.

E2. Comment démarrer le timer TC0 ?

E3. Comment peut-on activer et récupérer l'interruption du Timer TC0 ?

E4. Est-il nécessaire de clear (mettre à 0) le registre **TC_SR** avant de quitter votre interruption timer ? Si oui, expliquer pourquoi.

E5. Quelles doivent être les valeurs du registre de comparaison et du prescaler afin d'obtenir un timer de 8kHz, 16 kHz, 32 kHz, 44 kHz et 48 kHz ?

E6. A l'aide d'un **oscilloscope**, valider les résultats trouvés précédemment pour au moins 3 fréquences. Astuce : changer l'état d'une broche dans votre fonction d'interruption pour l'analyser sur l'oscilloscope.

T2. Comment peut-on utiliser notre timer en tant que fréquence d'échantillonage ?

Créer votre fonction **setupTimer()** afin de l'utiliser plus facilement pour les prochains TPs et le projet.

II. Maîtrise de l'ADC et du DAC

Une fois la fréquence du timer définie, nous allons pouvoir déclencher l'ADC en fonction du timer à une fréquence fixe. Ce sera donc notre fréquence d'échantillonage.

T3. Quel est l'identifiant du périphérique ADC du microcontrôleur ?

E7. Comment doit-on configurer le registre **ADC_MR** pour avoir les caractéristiques suivantes :

- Déclenchement manuel du début de la conversion analogique-digital
- Précision de l'ADC à 12 bits ;
- Prescaler à 3 ;
- Temps de démarrage à 64 périodes d'ADC_CLK ;
- Délai de 16 périodes d'ADC_CLK entre chaque basculement de l'entrée analogique de l'ADC ;
- Temps de stabilisation le plus faible possible ;

Les bits non utilisés du registre **ADC_MR** doivent rester à '0'.

E8. Comment peut-on sélectionner l'entrée de l'ADC à convertir ?

E9. Comment peut-on lancer la conversion de l'ADC et exécuter un code uniquement lorsque le drapeau d'interruption de fin de conversion est mis à 1 ? La fonction Handler de l'interruption de l'ADC est-elle nécessaire ?

E10. En connectant un potentiomètre à l'entrée activée de l'ADC, afficher les données converties de l'ADC sur le moniteur/traceur série. Comparer les valeurs dans le cas d'une configuration de l'ADC en 10 bits puis en 12 bits. Astuce : mettre le baud rate du moniteur série au maximum des capacités de l'Arduino Due.

E11. A l'aide de la fonction DAC donnée sur le **Github**, sortir les valeurs de l'ADC (avec une Fe à 16kHz) sur le DAC de l'Arduino Due.

1. Relier en entrée de l'ADC, un signal sinusoïdal généré par un **GBF** d'une fréquence 1kHz et d'amplitude 1V (**attention à ne pas dépasser 3.3V d'amplitude crête à crête !**)
2. En utilisant deux channels de l'oscilloscope, afficher simultanément la tension en entrée de l'ADC ainsi que la tension de sortie du DAC.

Qu'observez-vous ? Quel est le temps de la conversion (temps entre deux échantillons) de l'ADC et du DAC ?

E12. En utilisant le même protocole de test, modifier la fréquence d'échantillonage à 8KHz puis à 32KHz de l'ADC. Comparer vos résultats et conclure.

Créer votre fonction **setupADC()** et **setupDAC()** afin de les utiliser plus facilement pour les prochains TP et le projet.

III. Maîtrise de la DMA

La fréquence d'échantillonnage et la conversion analogique-numérique et numérique-analogique fonctionnant, nous pouvons explorer des pistes d'optimisations matérielles : l'utilisation de la DMA ou Direct Memory Access. La DMA nous permet de gagner du temps d'exécution afin de passer les valeurs d'un périphérique à l'autre rapidement (ADC vers DAC).

E13. Implémenter l'utilisation de la DMA pour le passage des données entre l'ADC et le DAC. A l'aide d'un oscilloscope, comparer vos résultats à ceux obtenus précédemment et conclure sur l'utilité de la DMA dans ce cas d'usage.

TP n° 2

Racines d'équations

Répertoire Github : <https://github.com/ElectroniqueECE/ECE-ELEC-CE>

Expérience 1 : Méthode de Newton-Raphson et amélioration de cette méthode en calculant l'ordre de la racine.

T1. Rappeler, pour la méthode de Newton-Raphson, la définition de la suite définie par récurrence qui converge vers une racine d'ordre p d'une fonction $f(x)$.

T2. Appliquer, en supposant tout d'abord $p = 1$, à la fonction :

$$f(x) = x^3 - 3x^2 + 3x - 1$$

T3. Écrire l'algorithme correspondant, qui devra retourner une racine et le nombre d'itérations exécutées.

E1. Écrire un code Arduino correspondant (aucun code ne doit être écrit dans la fonction `loop()`, mais seulement dans la fonction `setup()`) ; l'exécuter pour $x_0 = 0$.

E2. Que se passe-t-il si on l'exécute avec maintenant $x_0 = 10000$?

E3. On désire accélérer la convergence de la méthode de Newton-Raphson, en calculant l'ordre p de la racine « à la volée » et en tenant compte après 4 itérations par exemple. On rappelle que l'ordre p d'une racine peut être évalué par la formule, d'autant plus fiable que l'on est proche de la racine :

$$p = E\left(\frac{1}{1 - \frac{f''(x)f(x)}{f'(x)^2}}\right)$$

où $E(r)$ représente la partie entière du réel r .

Écrire alors le nouvel algorithme et le code Arduino correspondant, et le tester pour $x_0 = 0$ et $x_0 = 10000$.

E4. Qu'observe-t-on ?

E5. Conclure.

Expérience 2 : Méthode de Halley.

- T4.** Rappeler, pour la méthode de Halley, la définition de la suite définie par récurrence qui converge vers une racine d'une fonction $f(x)$.
- T5.** Écrire l'algorithme correspondant, qui devra retourner une racine et le nombre d'itérations exécutées.
- E6.** Écrire un code Arduino correspondant (aucun code ne doit être écrit dans la fonction `loop()`, mais seulement dans la fonction `setup()`) ; l'exécuter pour $f(x) = x^3 - 3x^2 + 3x - 1$ et $x_0 = 0$, puis encore pour $x_0 = 10000$.
- E7.** Qu'observe-t-on ?
- E8.** Conclure.

Expérience 3 : Comparaison de différentes méthodes de recherche de racine.

Nous allons déterminer le nombre d'or par différentes méthodes de recherche de racine. Le nombre d'or Φ est la racine positive d'ordre $p = 1$ de la fonction polynôme $f(x) = x^2 - x - 1$.

- T6.** Donner une expression exacte de Φ et en donner à l'aide d'une calculatrice ou autre une approximation avec 14 décimales maximum.
- E9.** En recherchant une même précision $\epsilon = 10^{-14}$, remplir le tableau suivant (on partira de $x_0 = 1$) :

| | Newton-Raphson | Halley | Laguerre |
|------------|----------------|--------|----------|
| x | • | • | • |
| $f(x)$ | • | • | • |
| N | • | • | • |
| m | • | • | • |
| θ | • | • | • |
| E | • | • | • |
| Δt | • | • | • |

x : Racine obtenue

N : Nombre d'itérations

m : Ordre de la méthode

θ : Travail par itération

$E = m^{\frac{1}{\theta}}$: Efficacité de la méthode

Δt : Temps d'exécution

Remarque : On rappelle la méthode de Laguerre :

$$x_{k+1} = x_k - \frac{nf(x_k)}{f'(x_k) + \text{signe}(f'(x_k)) \sqrt{(\frac{n-p}{p})((n-1)f'^2(x_k) - nf(x_k)f''(x_k))}}$$

p : ordre de la racine ; n : degré du polynôme. Cette méthode est d'ordre 3 pour une racine simple, et d'ordre 1 pour une racine multiple.

On donne aussi l'ordre de la méthode de Halley : 3.

Conclure. Comparer notamment l'efficacité de la méthode au temps d'exécution. Discuter des différents paramètres en jeu. Trouver une formule simple pour définir un indice de performance à partir de E et Δt . Donner la méthode la plus performante.

Expérience 4 : Étude d'une diode réelle.

Nous allons déterminer la caractéristique d'une diode qui se trouve dans votre kit d'électronique, la diode 1N4007. C'est une diode qui n'est pas de qualité « signal », mais sert essentiellement en pratique au redressement de tension. Ensuite, en plus de notre étude expérimentale, nous utiliserons une méthode de calcul embarqué pour déterminer le point de fonctionnement d'un montage avec cette diode, de façon automatique. On considère le montage d'étude ci-dessous. L'OPAMP est à sélectionner selon votre kit LM358 ou LM741.

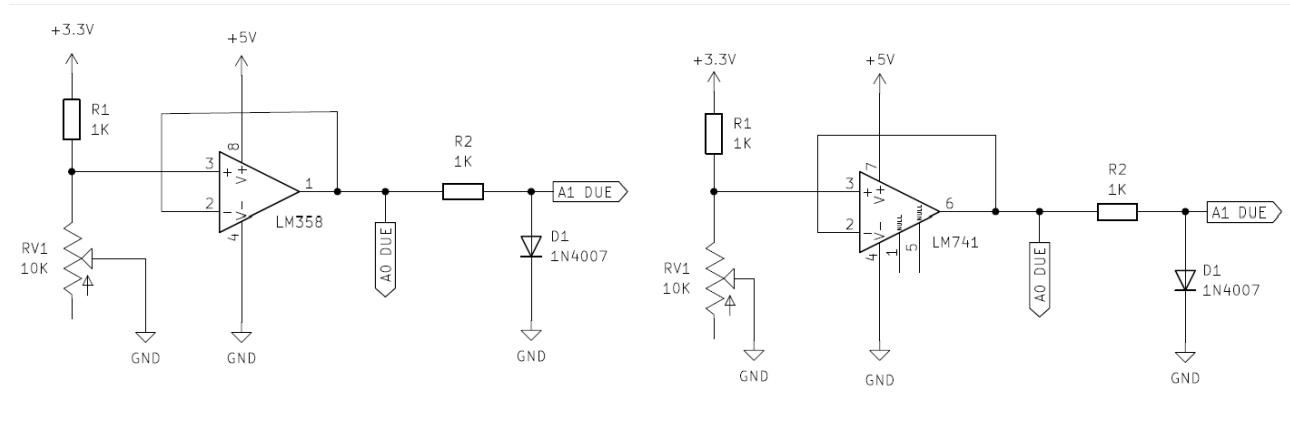


FIGURE 2.1 – Schéma électrique à connecter à l'Arduino DUE

Les pinouts des OPAMPS sont ci-dessous. Vérifiez bien le modèle : LM358 ou LM741. Les branchements ne sont pas les mêmes !

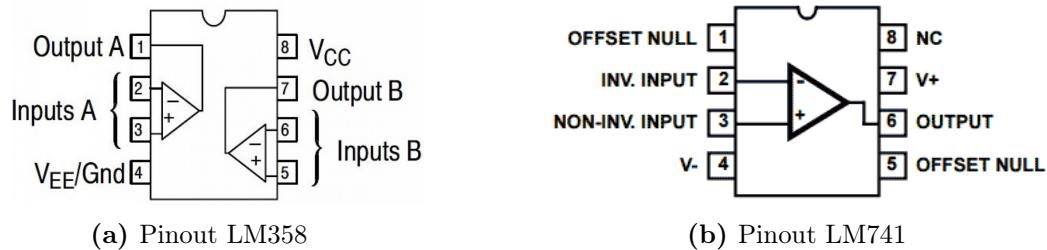


FIGURE 2.2 – Pinout des OPAMPS LM358 et LM741

- E10.** Quel rôle joue la partie gauche du montage ? La partie centrale ? La partie droite ? Justifier vos réponses.
- E11.** Exprimer la tension analogique U_e lue sur l'entrée A0. On introduira dans ce calcul la résistance utilisée RV_1 du potentiomètre. Noter que le CAN des PIN analogiques est codé sur 10 bit, et qu'il peut accepter au maximum 3,3 V ! Donner l'intervalle de variation de U_e lorsqu'on tourne le bouton du potentiomètre, conclure.
- E12.** Que représente la tension analogique U_d lue sur l'entrée A1 ?
- E13.** Comment calculer l'intensité du courant I_d qui traverse la diode, en fonction de U_e , U_d et R_2 ?
- E14.** Télécharger et utiliser le code Arduino disponible sur **Github** pour faire vos mesures : (vous remarquerez dans le code que le CAN de l'Arduino DUE est codé sur 10 bit encore, de tension maximale 3,3 V).
- E15.** Remplir alors le tableau de 14 mesures à bien répartir suivant :

| | | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|---|
| U_e | • | • | • | • | • | • | • | • | • |
| I_d | • | • | • | • | • | • | • | • | • |
| U_d | • | • | • | • | • | • | • | • | • |

E16. On utilise ici un tableur du genre Excel. Saisir en colonnes les données précédentes puis tracer la courbe I_d en fonction de U_d et chercher à l'aide du modèle :

$$I_d = a \exp(b U_d)$$

la valeur des coefficients a et b introduits dans le modèle. On sait en effet depuis le cours d'Electronique Fondamentale que pour une diode réelle :

$$I_d = I_{ds} \left(\exp\left(\frac{e U_d}{\eta k_B T}\right) - 1 \right)$$

avec : I_{ds} courant de saturation inverse (très faible)

$e = 1,6.10^{-19}$ C charge absolue de l'électron

$k_B = 1,38.10^{-23}$ SI constante de Boltzmann

$T(K) = 273 + t(C)$ température en degré Kelvin

η coefficient d'efficacité de la diode (compris entre 1 pour les meilleures et environ 2)

On peut définir la tension thermique $U_T \equiv \frac{k_B T}{e} \simeq 26$ mV à température ambiante. Cependant, comme les tensions U_d de nos mesures sont largement supérieures à U_T , on en déduit que la loi régissant notre diode réelle peut s'écrire :

$$I_d = I_{ds} \exp\left(\frac{e U_d}{\eta k_B T}\right)$$

ce qui explique l'origine de notre modèle dans le tableur.

E17. A la vue de la courbe, en déduire si la diode 1N4007 est une diode au germanium ou au silicium. Justifier.

E18. Évaluer alors le coefficient d'efficacité de la diode 1N4007. En déduire la qualité de la diode (d'autant meilleure que cette valeur est proche de 1).

Évaluer aussi I_{ds} .

E19. On veut maintenant, toujours pour le même montage, déterminer directement I_d et U_d à partir de la connaissance seule de U_e en entrée, à l'aide d'un calcul d'algorithme de calcul embarqué. En posant $x = U_d$, montrer que x est une racine de la fonction :

$$f(x) = R_2 I_{ds} \exp(b.x) + x - U_e$$

E20. Appliquer alors l'algorithme de Newton-Raphson pour obtenir x , puis exprimer aussi I_d en fonction de x .

E21. Ajouter alors au code Arduino utilisé plus haut, le code permettant de comparer en permanence les valeurs de U_d et I_d mesurées à celles calculées par la question précédente. Évaluer au passage l'erreur relative en pourcentage.

E22. Conclusion.

TP n° 3

Optimisation

Répertoire Github : <https://github.com/ElectroniqueECE/ECE-ELEC-CE>

Expérience 1 : Recherchons un extremum

Soit la fonction dont on cherche les extrema :

$$f(x) = 2x + \frac{3}{x}$$

- E1. Commencer par appliquer une méthode de dichotomie ou une analyse graphique pour cerner un intervalle initial $[a, b]$ que l'on précisera, où se situe un extremum.
- E2. Appliquer ensuite une méthode de Newton-Raphson pour déterminer l'extremum d'abscisse positive.
- E3. Est-ce un maximum ou un minimum ? Global ou local ?
- E4. Montrer que la valeur pouvait être obtenue ici par le calcul ; comparer vos résultats.

Expérience 2 : Trajectoire d'une balle

La trajectoire d'une balle peut être calculée selon l'équation :

$$y = (\tan\theta_0)x - \frac{g}{2v_0^2 \cos^2\theta_0}x^2 + y_0$$

où y est la hauteur, θ_0 l'angle initial, v_0 la vitesse initiale et g l'intensité de la pesanteur.

- E5. Déterminer la hauteur maximale atteinte.

Données : $g = 9,81$ SI, $y_0=2$ m, $v_0=24$ m/s, $\theta_0=52$ degrés.

Expérience 3 : Recherchons un extremum à l'aide de la méthode d'interpolation parabolique (calculs à réaliser à l'aide de l'Arduino DUE).

Soit la fonction :

$$f(x) = -1,6x^6 - 3x^4 + 10x$$

On rappelle la formule d'interpolation parabolique :

$$x_3 = \frac{f(x_0)(x_1^2 - x_2^2) + f(x_1)(x_2^2 - x_0^2) + f(x_2)(x_0^2 - x_1^2)}{2f(x_0)(x_1 - x_2) + 2f(x_1)(x_2 - x_0) + 2f(x_2)(x_0 - x_1)}$$

puis la permutation proposée :

$$x_0 \leftarrow x_{1\text{precedent}}$$

$$x_1 \leftarrow x_{2\text{precedent}}$$

$$x_2 \leftarrow x_{3\text{precedent}}$$

- E6.** Rechercher la position x de l'extremum de $f(x)$ et sa valeur à l'aide de la méthode d'interpolation parabolique. On partira de $x_0 = 0$, $x_1 = 1$ et $x_2 = 2$ et on réalisera trois itérations minimum. Il peut être utile de remplir un tableau rassemblant les résultats obtenus.
- E7.** Comparer vos résultats par rapport à une méthode de Newton. Conclure (et donner quelques comparaisons, avantages/inconvénients).

Expérience 4 : Modèle de Hopfield et recherche d'extremum (calculs à réaliser à l'aide de l'Arduino DUE).

On considère un montage simple qui permet d'afficher des données sur un écran OLED monochrome à partir d'une Arduino DUE : on rappelle que le branchement se fait simplement comme décrit ci-dessous :

- * ARDUINO DUE : PIN GND = OLED : GND
- * ARDUINO DUE : PIN 5V = OLED : VCC
- * ARDUINO DUE : PIN SCL 21 = OLED : SCL
- * ARDUINO DUE : PIN SDA 20 = OLED : SDA

Le modèle de Hopfield, vu en cours, permet de créer une mémoire associative. On rappelle que si les neurones sont notés V_i (valeurs -1 ou +1, ou encore 0 ou 1) et les poids des connexions (règle d'apprentissage) entre le neurone i et le neurone j sont notés :

$$T_{ij} = \sum_s V_i^s V_j^s$$

s étant l'indice indexant les motifs à apprendre, alors la dynamique du réseau sera :

$$\begin{aligned} V_i(t+1) &= +1 \text{ si } \sum_j T_{ij} V_j(t) > 0 \\ V_i(t+1) &= -1 \text{ si } \sum_j T_{ij} V_j(t) \leq 0 \end{aligned}$$

et cette dynamique du réseau fera converger à priori le réseau vers un motif appris (vers le plus ressemblant).

- E8.** Réaliser le montage.

E9. Télécharger le code HOPFIELD.ino depuis **Github** et le téléverser sur l'Arduino DUE. Ce code affiche d'abord les lettres à apprendre au réseau de neurones, puis les apprend, et ensuite on présente au réseau la lettre « T » bruitée et on regarde vers quelle lettre la dynamique du réseau fait converger le réseau.

E10. Qu'observe-ton sur l'écran OLED ?

E11. Le réseau converge-t-il toujours vers la lettre « T » ?

E12. Analysons le code fourni HOPFIELD.ino :

- (a) Quelles sont les dimensions n et m de l'image qui s'affiche ?
- (b) Combien de neurones y a-t-il dans ce réseau de neurones ?
- (c) Combien de lettres (motifs) notre réseau peut-il donc apprendre d'après le cours (environ 0, 14.*NombreDeNeurones*) ?
- (d) Faire varier la lettre de départ (ligne de code 139 : modifier le numéro du motif, entre 0 et 6) et le niveau de bruit (ligne de code 169 : modifier le seuil) ; et observer vers quelle lettre converge le réseau.
- (e) Conclusion : la reconnaissance de caractère avec un modèle de Hopfield fonctionne-t-elle et est-elle robuste au bruit ? Existe-t-il des états « poubelles » ? Comment pourrait-t-on l'améliorer (en jouant sur quels paramètres) ? Pour répondre à cette question, vous pouvez aussi supprimer le bruit et/ou modifier le motif de départ en amorçant le réseau avec une partie de lettre commune à plusieurs lettres apprises et en regardant vers quel motif converge le réseau (cela nécessite un peu de codage supplémentaire simple).

E13. Maintenant, on peut exploiter la capacité d'un modèle de Hopfield à minimiser une forme d'énergie. Par exemple pour le problème du voyageur de commerce : comment minimiser sa distance totale de parcours de N villes et choisir donc l'ordre de visite des villes, les distances entre chaque villes étant connues. On peut montrer qu'il suffit de changer la règle d'apprentissage de notre réseau de neurones de Hopfield et ce problème très complexe peut à priori être résolu ou au moins approcher la solution optimale. Nous verrons cela dans le cours.

TP n° 4

Équations différentielles

Exercice 1 : Polynômes de Hermite

On étudie une certaine classe de polynômes de Hermite $H_k(x)$, définis selon la relation de récurrence :

$$H_{k+1}(x) = 2xH_k(x) - 2kH_{k-1}(x)$$

avec :

$$H_0(x) = 1$$

$$H_1(x) = 2x$$

On rappelle la méthode de Clenshaw qui permet d'évaluer la valeur de n'importe quelle fonction de la forme :

$$y(x) = \sum_{k=0}^n c_k \varphi_k(x)$$

selon la condition :

$$\varphi_{k+1}(x) + a_k(x)\varphi_k(x) + b_k(x)\varphi_{k-1}(x) = 0$$

On définit alors la suite u_k selon le schéma itératif :

$$u_{n+1} = u_{n+2} = 0$$

$$u_k + a_k(x)u_{k+1} + b_{k+1}(x)u_{k+2} = c_k$$

Le théorème de Clenshaw nous fournit alors le résultat final :

$$y(x) = \varphi_0(x)u_0 + (\varphi_1(x) + a_0(x)\varphi_0(x))u_1$$

T1. À l'aide de la relation de récurrence sur $H_k(x)$, exprimer $H_2(x)$ et $H_3(x)$.

T2. Exprimer $a_k(x)$ et $b_k(x)$.

T3. Écrire alors la relation détaillée qui permet de calculer les u_k pour $k=n, n-1, \dots, 0$.

T4. Exprimer alors le résultat final $y(x)$.

E1. Coder le programme correspondant en langage Python (appeler ce fichier `exo1_q5.py`), qui permet de tracer la courbe de $y(x)$ sur l'intervalle $[-2, 2]$ (avec 200 points) en prenant comme coefficients c_k les valeurs suivantes ($n=5$) :*

| | | | | | | |
|-------|---|---|---|----|---|---|
| k | 0 | 1 | 2 | 3 | 4 | 5 |
| c_k | 3 | 1 | 0 | -1 | 1 | 2 |

Fournir la capture d'écran.

E2. Utiliser et modifier le programme précédent pour afficher les polynômes $H_7(x)$ puis $H_8(x)$. Fournir les captures d'écran.

E3. Application : En physique quantique, un oscillateur harmonique (masse m accrochée à l'extrémité d'un ressort) possède des états d'énergie E_n de fonction d'onde spatiale de la forme $\Psi_n(x) = K_n H_n(\lambda x) \exp(-\frac{\lambda^2 x^2}{2})$ avec x la position d'écartement par rapport à l'équilibre et où λ et K_n sont des constantes qu'on prendra ici égales à 1.

Tracer la courbe de la densité de probabilité de présence $|\Psi(x)|^2$ de la masse m en fonction de la position x dans l'état $n=5$ (fournir la capture d'écran) sur l'intervalle $[-10, 10]$ (1000 points par exemple)

En déduire combien de positions x sont fortement probables ? La position $x = 0$ est-elle probable ? Quelles sont les positions les plus probables ? Interpréter.

Exercice 2 : Résolution d'une équation différentielle par Euler et Runge Kutta 2

On étudie l'équation différentielle suivante :

$$y'(x) + 8y(x) = \exp(-4x)$$

obéissant à la condition initiale $y(0) = 0$.

T5. Donner la solution de cette équation différentielle.

T6. Si on écrit cette équation différentielle sous la forme :

$$y'(x) = f(x, y)$$

expliciter alors la fonction $f(x, y)$.

T7. Rappeler la méthode de Euler.

T8. On désire utiliser aussi la méthode RK2. On rappelle qu'il est établi que dans cette méthode :

$$y_{n+1} - y_n = h\Phi(x_n, y_n, h)$$

avec :

$$\Phi(x, y, h) = (1 - \beta)f(x, y) + \beta f\left(x + \frac{h}{2\beta}, y + \frac{h}{2\beta}f(x, y)\right)$$

Expliciter alors cette méthode RK2 :

1. dans le cas où $\beta = 1$
2. dans le cas où $\beta = \frac{1}{5}$

E4. Coder les méthodes de Euler et RK2($\beta = \frac{1}{5}$) dans un même code python (partir du fichier `exo2.py` fourni, le compléter et le rendre), et comparer les 2 solutions obtenues avec la solution exacte (tracer donc 3 courbes, fournir la capture d'écran). On prendra l'intervalle d'étude $[0, 2]$ avec un pas $h = 0.01$.

E5. Conclure.

Exercice 3 : Résolution d'une équation différentielle par Runge Kutta 4

On étudie l'équation différentielle suivante :

$$y'(x) + \exp(-4x)y(x) = \exp(-x)$$

obéissant à la condition initiale $y(0) = 0$. On rappelle la méthode de Runge Kutta 4 :

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right) \\ k_3 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right) \\ k_4 &= f(x_n + h, y_n + hk_3) \\ y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ x_{n+1} &= x_n + h \end{aligned}$$

E6. Résoudre par RK4 l'équation différentielle sur l'intervalle $[0, 5]$ avec un pas $h = 0.01$. Fournir le code créé appelé exo3.py et une copie d'écran de la courbe.

E7. Que vaut $y'(0)$ d'après l'équation différentielle ? Vérifier ce résultat sur la courbe.

E8. Que vaut $\lim_{x \rightarrow +\infty} y(x)$? Vérifie-t-on ce résultat sur la courbe ?

Outils mathématiques pour le Machine Learning

I. Convolution 2D d'une image

On considère une image déjà vue en cours CM5 sur laquelle on a déjà appliqué des filtres (kernel) afin d'en extraire de l'information à l'aide de scripts MATLAB.



FIGURE 5.1 – Image sur laquelle appliquer les filtres

Retrouvons le même genre de résultats à l'aide du langage support des scientifiques Python. On peut, pour exécuter nos codes Python, utiliser un codeur Python en ligne, ou installer sur son ordinateur personnel l'IDE Python Spyder ou encore (mieux) installer Anaconda qui contient déjà Spyder aussi, mais qui permet surtout une installation sans soucis des librairies Python ; ou tout autre moyen. Il faudra en effet installer pour cet exercice la librairie OpenCV (cv2) (dans anaconda, aller dans Environnement et installez opencv).

E1. Exécuter le code Python « convolution2D.py » sur le Github
<https://github.com/ElectroniqueECE/ECE-ELEC-CE>.

1. Qu'observe-t-on ? Faire une copie d'écran.
2. Où se trouve l'image de départ sur l'écran ?
3. Que représente à priori l'autre image et où se trouve-t-elle ?
4. Confirmer ce résultat en analysant le code.
5. Quel kernel a été choisi ? Écrire cette matrice. Quels types de caractéristiques est-il censé extraire ? Ce résultat est-il atteint ?

E2. Modifier maintenant le kernel dans le code Python pour observer maintenant un floutage de type gaussien. Écrire la matrice de ce kernel. Le résultat escompté est-il bien obtenu ? Faire une copie d'écran.

E3. On désire maintenant réaliser l'inverse du floutage de cette image, c'est-à-dire appliquer un filtre « piqué ».

1. Quel kernel simple faut-il choisir ?
2. Réaliser ce filtrage, qu'observe-ton ? Faire une copie d'écran.

E4. On désire maintenant réaliser l'extraction des lignes verticales.

1. Quel kernel simple faut-il choisir ?
2. Réaliser ce filtrage, qu'observe-ton ? Faire une copie d'écran.

E5. On désire maintenant réaliser l'extraction des lignes horizontales.

1. Quel kernel simple faut-il choisir ?
2. Réaliser ce filtrage, qu'observe-ton ? Faire une copie d'écran.

E6. Maintenant on désire extraire les lignes diagonales à 45 degrés vers la droite.

1. Ecrire le kernel recherché et l'appliquer ; obtient-on un bon résultat ? Faire une copie d'écran.
2. Remplacer alors les zéros de ce kernel par des -1 et changer les 2 en 3, et reobserver. Faire une copie d'écran.
3. Conclure. Trouver le kernel optimal pour cette opération.

E7. Maintenant on désire extraire les lignes diagonales à 45 degrés vers la gauche.

1. Ecrire le kernel recherché et l'appliquer ; obtient-on un bon résultat ? Faire une copie d'écran.
2. Remplacer alors les zéros de ce kernel par des -1 et changer les 2 en 3, reobserver. Faire une copie d'écran.
3. Conclure. Trouver le kernel optimal pour cette opération.

Si vous avez le temps, choisir une autre image et recommencer ce travail...

II. Régression linéaire et descente de gradient à 2 dimensions

On considère 3 points non alignés : $A_1 = (0 ; 5,5)$, $A_2 = (1 ; 6,5)$ et $A_3 = (2 ; 9,2)$

E8. Écrire la fonction erreur $E(a, b)$ en fonction de a et b .

E9. Écrire le gradient de la fonction d'erreur en fonction de a et b .

E10. Écrire les deux suites définies par récurrence de la descente de gradient permettant de trouver le minimum de la fonction erreur.

E11. Calculer à l'aide d'une feuille de calcul Excel ce système.

E12. En déduire a et b .

E13. Tracer la droite de régression et les points de départ sur un même graphique.

E14. Conclure.

Introduction à l'apprentissage machine

Motivation

L'objectif de ce TP est d'introduire des notions de Machine Learning appliquée à l'embarqué sur la carte Arduino Due. Nous verrons la régression linéaire ainsi que la régression logistique.

I. Régression linéaire

Dans cette partie, nous étudierons l'algorithme de régression linéaire. Le dataset pour cette étude représente l'analyse de quatre moteurs identiques théoriquement mais différents en pratique en raison de la tolérance élevée de sortie d'usine. Il s'agira de trouver une équation unique liant les quatre moteurs ensemble et ainsi d'établir la vitesse de rotation en fonction du rapport cyclique envoyé par le microcontrôleur. Des mesures ont été volontairement enlevés du dataset final afin de justifier l'analyse par régression linéaire.

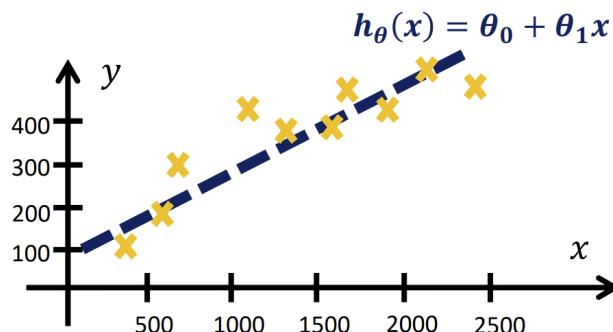


FIGURE 6.1 – Exemple de régression linéaire

T1. À partir du dataset donné dans le code, représenter graphiquement l'hypothèse $h_\theta(x)$ et expliquer le choix de la régression linéaire.

T2. Écrire le critère J et rappeler l'algorithme de descente de gradient pour calculer les paramètres $(\theta_1; \theta_0)$ ainsi que la définition des termes dans le cas de la régression linéaire.

E1. À partir du code donné sur [Github](#), compléter la fonction d'entraînement par votre résultat trouvé précédemment, la fonction de prédiction linéaire, la fonction de prédiction de la vitesse et la fonction de prédiction du rapport cyclique. Un algorigramme est à réaliser.

E2. Afficher le résultat d'une prédiction à 20 %, 60 %, $12\,000 \text{ tr} \cdot \text{min}^{-1}$ puis à $25\,000 \text{ tr} \cdot \text{min}^{-1}$ et commenter vos résultats.

E3. Modifier le taux d'apprentissage et l'objectif d'erreur puis analyser vos résultats en cycle d'entraînement et en précision en traçant l'équation donnée et conclure.

II. Régression logistique

Dans cette partie, nous étudierons l'algorithme de régression logistique. Le dataset pour cette étude représente les données reçues d'une caméra pour le suivi d'une ligne continue par un véhicule autonome pouvant être développé par l'association ARECE ou lors de la majeure VCA en ING4. Plus la caméra voit la ligne à droite, plus les valeurs tendent vers 255 et respectivement -255 pour la ligne à gauche. La prédiction indique la nouvelle direction du véhicule à prendre.

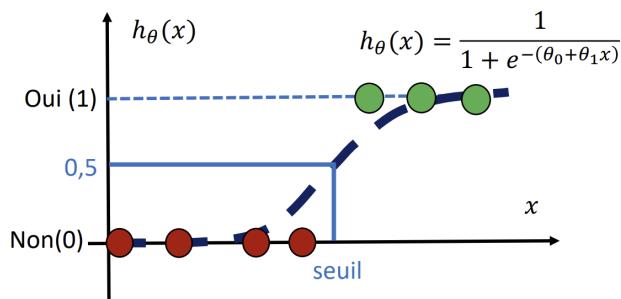


FIGURE 6.2 – Exemple de régression logistique

T3. À partir du dataset donné dans le code, représenter graphiquement l'hypothèse $h_\theta(x)$ et expliquer le choix de la régression logistique.

T4. Écrire le critère J et rappeler l'algorithme de descente de gradient pour calculer les paramètres $(\theta_1 ; \theta_0)$ dans le cas de la régression logistique.

E4. À partir du code donné sur [Github](#), compléter la fonction d'entraînement par votre résultat trouvé précédemment, la fonction $h_\theta(x)$ et la fonction de prédiction de la direction. Un alorigramme est à réaliser.

E5. Afficher le résultat d'une prédiction quand la caméra voit -125 puis 40 et commenter vos résultats.

E6. Modifier le taux d'apprentissage et l'objectif d'erreur puis analyser vos résultats en cycle d'entraînement et en précision en traçant l'équation donnée. Quel serait l'impact sur la direction si le modèle était trop entraîné ?

Les réseaux de neurones

Motivation

L'objectif de ce TP va être de créer et d'entraîner des réseaux de neurones simples et de se familiariser avec les concepts liés aux réseaux de neurones : couches d'entrées, poids, fonction de transfert, fonction d'activation, couches de sorties.

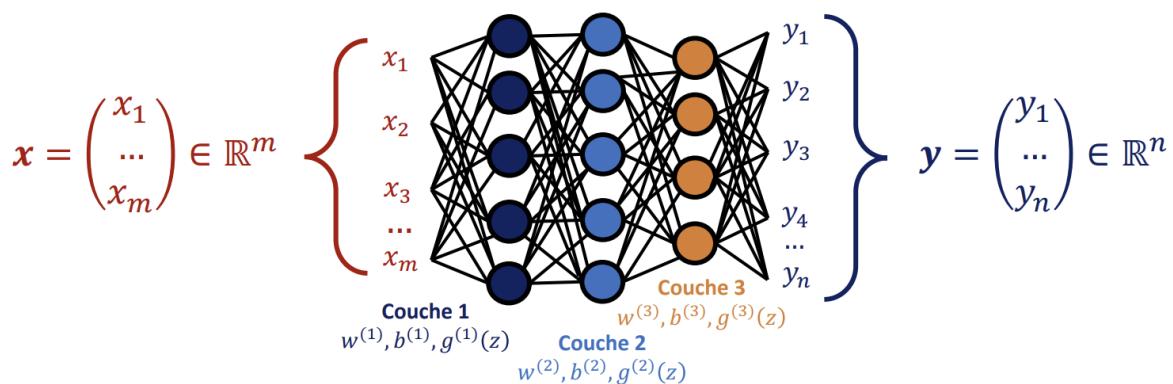


FIGURE 7.1 – Exemple d'un réseau de neurones multicouche

Nous utiliserons la bibliothèque **NeuralNetwork** de George Chousos afin d'implémenter simplement nos différents réseaux de neurones que nous développerons au cours de ce TP sur Arduino Due. Il est possible de l'installer directement depuis Arduino IDE en passant par le **Library Manager > NeuralNetwork**.

Le Github de la bibliothèque liste les différentes macros possibles pour optimiser l'entraînement ainsi que les fonctions d'activation disponibles : <https://github.com/GiorgosXou/NeuralNetworks>

I. Réseau simple de neurones : Porte XOR

Afin de s'initier à des réseaux de neurones simple, nous allons prendre l'exemple fourni par la bibliothèque NeuralNetwork « **Backpropagation_Single_Xor** ».

(Fichier > Exemples > NeuralNetwork > Basic > Backpropagation_Single_Xor).

Téléversez le code sur votre Arduino Due depuis le port **Programming**. Ouvrez le moniteur série. Félicitations ! Vous avez entraîné votre premier réseau de neurones !

T1. Comment sont initialisés les poids du réseau de neurones ?

T2. En vos mots, expliquer le concept de feedforward, de backpropagation et l'utilité de la fonction d'activation dans un réseau de neurones (10~20 lignes).

T3. Quelle est le lien entre les valeurs d'entrées/sortie attendue et le nombre de neurones ?

T4. Les poids du réseau de neurones sont-ils modifiés au cours de l'entraînement ? Si oui, justifier par un algorithme en introduisant les notions vues à la question T2 et T3.

T5. Comment la Mean Square Error (MSE) du réseau de neurones est-elle calculée ? Pourquoi est-ce un paramètre important à prendre en compte dans l'entraînement d'un réseau de neurones ?

E1. Dans le code d'exemple XOR, expliquer le choix de l'auteur pour les données d'entraînement et leur lien avec le nombre de couches et de neurones.

E2. Une fois l'entraînement terminé, de quelle manière de nouvelles données peuvent être passées à travers le réseau de neurones pour obtenir une prédiction finale ?

T6. Pourquoi est-il nécessaire de vérifier l'entraînement de votre réseau de neurones avec des données de test ?

E3. Modifier le seuil d'erreur à 0,1 puis à 0,0001 et tracer la courbe de la MSE au fil des itérations à l'aide du moniteur série ou de l'outil « log » de Putty. Quel serait, selon vous, le seuil d'erreur optimal pour réduire le temps d'entraînement tout en conservant un résultat correct ? Un résultat correct peut se traduire par une différence faible entre les données de tests et la sortie attendue.

E4. Modifier les couches cachées de votre réseau de neurones (plus de neurones, moins de neurones, augmenter le nombre de couches cachées). Comment l'entraînement est-il affecté ? Discuter de l'optimisation autour de la dimension d'un réseau de neurones.

E5. Modifier la fonction d'activation du réseau de neurones. Comment l'entraînement est-il affecté ? Quelle fonction vous semble plus optimisée ? Que se passe-t-il si l'on modifie aussi les couches du réseaux de neurones ?

II. Réseau avancé de neurones : détection de formes

Le réseau de neurones que nous voulons créer doit pouvoir déterminer si la forme détectée venant d'un capteur est un cercle ou une croix. Les données d'entraînement et de test sont fournies sur le [Github du Pôle Électronique](#).

À partir des données d'entraînement et des informations sur ces dernières, créer un réseau de neurones capable de reconnaître un cercle ou une croix avec un objectif de MSE de 0,001.

E6. À quoi correspondent les données d'entraînement ? A quoi correspond la sortie attendue (0 et 1) ?

E7. Combien de neurones d'entrée et de sortie votre réseau de neurones doit-il avoir ?

E8. Calculer et comparer le temps d'entraînement afin de trouver la meilleure fonction d'activation et le nombre de couches cachées ainsi que leur nombre de neurones respectifs. Le modèle doit pouvoir reconnaître correctement les quatre premières formes du vecteur testData décrite ci-dessous afin d'être validé.

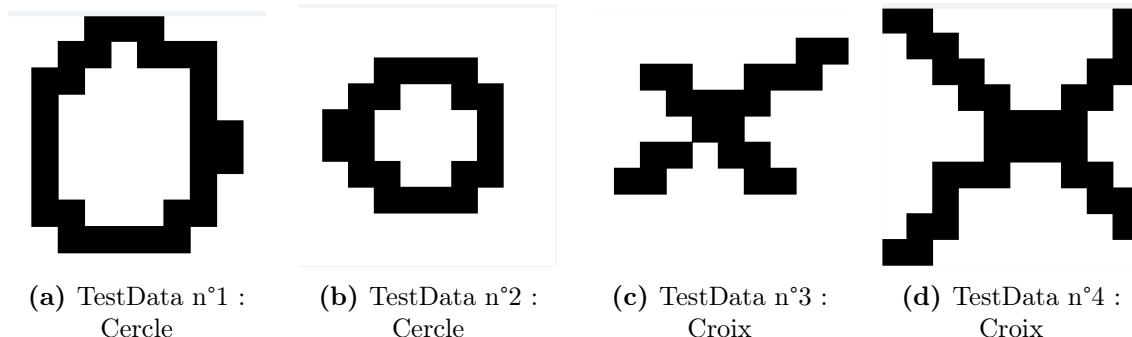


FIGURE 7.2 – Les quatre premières formes

E9. Une fois le modèle correctement entraîné, quelles sont les quatre dernières formes du vecteur testData ?

E10. À l'aide du script Python donné, dessiner vos propres cercles ou croix afin de tester les limites de votre réseau de neurones ? Quel type de réseau de neurones peut-être utilisé et qui serait plus optimisé pour le traitement d'images ?

TP n° 8

Deep Learning et Réseau de Neurones Convolutionnel

Motivation

Dans le TP précédent, nous avons vu différentes implémentations simples d'un réseau de neurones. Cependant, un réseau de neurones classique n'est pas toujours optimal pour résoudre l'ensemble des problématiques existantes. Des variantes ont dû être développées pour palier à ce manque. En effet, la reconnaissance d'images s'effectue principalement sous la forme d'un réseau de neurones convolutif (CNN), extrêmement efficace dans l'analyse d'images.

L'objectif de ce TP va être de créer et d'entraîner un CNN sur Arduino Due afin classifier des 1 et des 8 à partir de la base de données d'écriture Manuscrite MNIST.

MNIST, Mixed National Institute of Standards and Technology, est une base de données de chiffres écrit à la main très utilisée afin de tester des modèles de reconnaissances, comme par exemple un CNN. Il s'agit d'une base de données de plus de 60 000 images au format suivant :

- une image représente un chiffre écrit à la main entre 0 et 9 ;
- chaque image est composée de 28×28 pixels en noir et blanc, soit une matrice de 784 valeurs comprises entre 0 et 255.



FIGURE 8.1 – Exemple de données d'écriture manuscrite MNIST

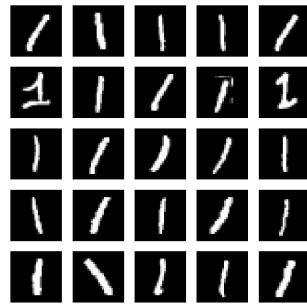
L'entraînement des meilleurs réseaux de neurones pour avoir une catégorisation la plus précise possible des caractère du MNIST nécessite des machines puissantes. L'entraînement de notre CNN doit s'effectuer sur la carte Arduino Due contraignant la taille du modèle possible. Ainsi, nous allons nous concentrer sur la reocnnaissance des caractères 1 et 8 et de leur classification respective.

I. Résolution sur NN Classique

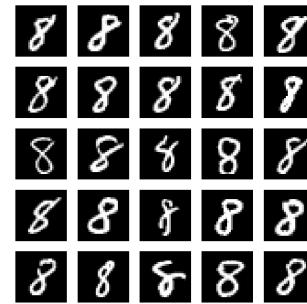
Dans un premier temps, il est nécessaire de déterminer les performances d'un réseau de neurones classique pour résoudre cette tâche. Il sera ainsi possible de comparer l'optimisation en temps et en précision obtenue avec la création de votre réseau de neurones convolutif.

T1. Quels sont les avantages et les inconvénients d'avoir un très grand jeu de données pour l'entraînement d'un réseau de neurones ?

Le Github du Pôle Électronique met à votre disposition les données nécessaires à l'entraînement de votre réseau de neurones classique. Il s'agit d'un extrait de la MNIST avec 20 images de chaque chiffre à classifier. Les données à tester sont également disponibles sous la forme d'un tableau de 30 données. Il est à noter que ces données serviront pour votre réseau de neurones classique ainsi que pour votre réseau de neurones convolutif.



(a) Images d'entraînement 1



(b) Images d'entraînement 8

Deux modifications ont été apportées par rapport aux données brutes issues de la base de données :

- Chaque valeur a été normalisée entre 0 et 1. Elles sont normalement entre 0 et 255.
Attention, il s'agit d'une étape très importante pour que la bibliothèque utilisée puisse lire les données (attention à bien effectuer cette étape dans des projets futurs).
- Les données ont été formatées sous la forme de vecteurs à deux dimensions en langage C. Il est possible d'importer les images au format texte dans votre IDE dans un tableau.

T2. À quoi servent les données de test ? Pourquoi ne pas réutiliser les données d'entraînement ?

De la même manière que dans le TP précédent, créer un réseau de neurones avec les données d'entraînement et de test du Github. L'architecture proposée pour les couches est la suivante : {X, 25, 7, 1}. La fonction d'activation à utiliser est la fonction tangente hyperbolique. La fonction FlattenToVector vous permettra de transformer le tableau en 2 dimensions en un tableau à une dimension.

E1. Quelle doit être la taille de la première couche pour votre réseau de neurones ? Déterminer le nombre de poids qui devront être ajustés à chaque propagation.

E2. Quelles contraintes de l'Arduino Due nécessite l'implémentation de la fonction FlattenToVector dans la boucle for de l'entraînement ? Justifier votre réponse de manière chiffrée.

E3. Combien de temps votre réseau prend-t-il pour converger avec une MSE de 0,03 ? Quelle est la précision sur les données de test (MSE à calculer) ?

II. Conception du CNN

Une fois la performance de référence établie avec un réseau de neurones classique, l'étude des optimisations apportées par un réseau de neurones convolutif est à effectuer. L'architecture de votre réseau de neurones convolutif est décrite ci-dessous. La disposition des couches est décrite comme suivante : $\{X, 3, 3, 1\}$

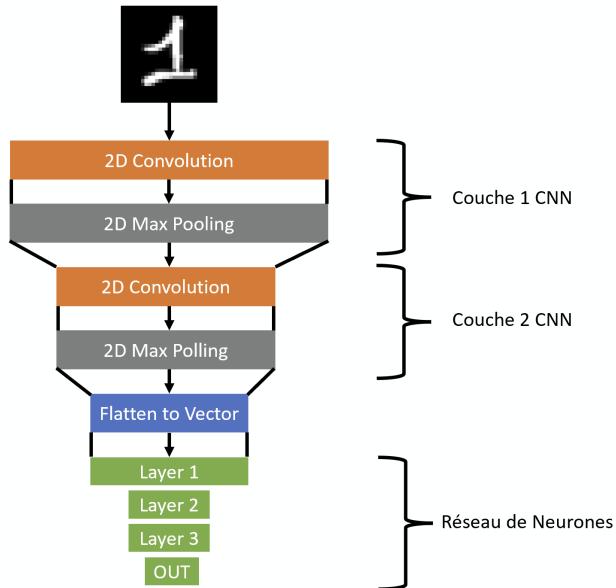


FIGURE 8.3 – Architecture de votre CNN

T3. En vos mots, expliquer les concepts implémentés par les fonctions de convolution, de maxpolling (5~10 lignes). Quelle est l'utilité du padding dans la convolution ?

À l'aide du code fourni, implémenter les fonctions de maxPolling et de convolution afin de transformer le réseau de neurones classique en réseau de neurones convolutif. Il est nécessaire d'adapter la fonction flattenToVector. Attention à bien modifier la première couche d'entrée !

Pour l'écriture de votre fonction de convolution 2D, veillez à ajouter un padding à votre matrice directement dans la fonction et appliquez la convolution avec le kernel.

E4. Pourquoi n'a-t-on pas besoin d'inverser le kernel ?

E5. Expliquer le choix du Kernel à partir du cours ? Le choix qui a été fait dans le programme est il pertinent par rapport au résultat souhaité ?

III. Analyse des Résultats

Maintenant que l'architecture est définie et que les couches de convolutions ont été implémentées, nous allons pouvoir déterminer les performances de votre CNN et les comparer à celle du réseau de neurones classique afin de mettre en évidence les avantages de la convolution.

E6. Déterminer le temps de convergence de votre réseaux de neurones successivement pour une MSE de 0,03, 0,001 et 0,000 02. Quelle est la précision sur les données de test (MSE à calculer) ? Comparer avec le réseau de neurones classique.

E7. Combien d'images d'entraînement pourrait-on rajouter ? Quels seraient les impacts sur les résultats obtenus précédemment ?

E8. Modifier la disposition des couches en suivant ces deux configurations :

- {X, 4, 1}
- {X, 30, 20, 1}

Citer dans les deux cas les phénomènes qui se produisent et décrivez les à l'aide de vos résultats.

E9. En s'inspirant du code d'exemple qui est proposé sur le Github de la library NeuralNetwork, modifier votre code pour pouvoir inscrire les poids calculés à l'issu de l'entraînement directement dans votre code.

(Fichier > Exemples > NeuralNetwork > Other > FeedForward_Individual_MNIST_PROGMEM).

T4. En quoi ce fonctionnement est-il plus pertinent pour de l'embarqué ? Peut-on imaginer une manière de refaire l'entraînement, mais cette fois-ci pour reconnaître tous les caractères ? La reconnaissance étant toujours faite sur Arduino Due.

FFT et MFCC

Motivation

L'objectif de ce TP est de comprendre le fonctionnement d'un algorithme de reconnaissance vocale et de l'implémenter sur Arduino Due à l'aide d'un microphone.

L'algorithme MFCC, Mel-Frequency Cepstral Coefficients, est utilisé dans des applications de reconnaissance vocale. L'objectif de cet algorithme est de réduire au minimum le nombre de coefficients qui permettent de qualifier un enregistrement audio. Ensuite, d'autres algorithmes pourront qualifier les coefficients obtenus pour y détecter des sons ou des mots.

Afin d'extraire les caractéristiques de chaque enregistrement, nommées **les features**, l'algorithme MFCC applique des transformations sur l'enregistrement d'origine afin de passer du domaine temporel au domaine fréquentiel (FFT) puis à nouveau une transformation pour repasser dans le domaine temporel (iFFT) afin de récupérer les coefficients MFCC.

Dans un premier temps, l'objectif est de découvrir le microphone qui sera utilisé pour enregistrer des sons. Ensuite, l'étude les différentes étapes de la transformation du signal vers les MFCC et de son implémentation sur l'Arduino Due sera à réaliser.

I. Microphone MAX9814

L'acquisition de votre signal phonatoire s'effectuera à l'aide du module MAX9814 fourni dans votre kit électronique. Le module MAX9814 est idéal car il comporte un amplificateur intégré et donc une plage de tension en sortie directement exploitable par l'Arduino Due. Sa tension d'alimentation est de +3.3V. Les broches du module sont détaillées ci-dessous.

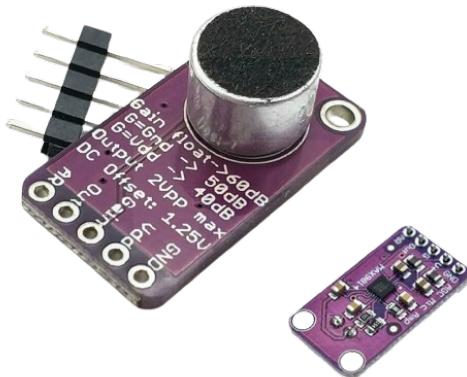


FIGURE 9.1 – MAX9814 : Microphone et carte d'amplification

- **VCC et GND** : Le microphone contient un circuit d'amplification actif, il doit donc être alimenté avec une tension de +3.3V.
- **GAIN** : La broche de GAIN permet de contrôler le gain de l'amplification à la sortie du microphone. En fonction de l'état du point le gain est modifié :

Nous utiliserons le gain le plus fort, en laissant la broche à l'état flottant. (non connecté).

- **A/R** : Cette broche permet de contrôler le rapport d'attaque et de relâchement du contrôle automatique du gain.

Nous n'aurons pas à utiliser cette broche pour le TP, nous pourrons donc la laisser non connecté.

- **OUT** : Sortie analogique du microphone après amplification par la carte.

La première étape afin d'utiliser le microphone est de le connecter à l'Arduino Due qui pourra convertir par la suite le signal analogique en un signal numérique à l'aide de son ADC. La broche OUT du microphone devra être connectée à une des broches analogiques (ex : A0) de la carte Arduino Due.

E1. À l'aide de la fonction **AnalogRead** et du **traceur série**, afficher la sortie du module MAX9814. Quelle est la valeur moyenne du signal ?

E2. En vous appuyant sur la [datasheet du composant](#), donner les valeurs moyenne, maximale et minimale de la tension de la broche OUT du MAX9814. Vos mesures précédentes sont-elles cohérentes par rapport à la datasheet ?

II. Fréquence d'échantillonnage de l'ADC

Afin de pouvoir faire des calculs numériques sur le signal, nous allons devoir le convertir. Comme vu au TP n°1, nous allons mettre en œuvre l'échantillonnage à une fréquence donnée à l'aide des registres de l'Arduino pour appliquer des transformations (temporelles et fréquentielles) au signal numérique. Cet échantillonnage doit exploiter toute les capacités de l'ADC de l'Arm Cortex-M3 SAM3X8E et donc se faire sur 12 bits.

E3. En cherchant sur internet, expliquer pourquoi une fréquence d'acquisition de 8 kHz est pertinente pour un enregistrement de la voix humaine (attention à bien prendre en compte la condition de Shannon).

E4. Proposer une configuration des registres de l'Arduino Due permettant un échantillonnage du signal audio en respectant les contraintes énoncées ci-dessus.

Astuce : Un tutoriel pour lire votre enregistrement à l'aide d'Audacity est disponible sur La **Toolbox > Audacity**. Ce point est optionnel mais cela vous permettra de valider votre système. De manière générale, toute initiative de validation du fonctionnement de votre système sera valorisé dans les rapports de travaux pratiques.

III. Transformation du signal par Fourier (FFT)

Dans cette partie du TP, nous allons enregistrer des signaux sonores et analyser leurs composantes fréquentielles. Afin d'effectuer l'analyse de fréquence l'algorithme de la transformée de Fourier rapide (FFT). Cet algorithme convertit un signal temporel échantillonné en sa transformée de Fourier. La transformée de Fourier correspond au domaine fréquentiel du signal.

La forme la plus simple et la plus courante de l'algorithme FFT est appelée l'algorithme de Cooley-Tukey dont nous allons voir l'application

E5. Compléter le code fourni sur le **Github** pour afficher le spectre de votre signal audio sur votre écran OLED. La bibliothèque Arduino FFT vous sera utile.

E6. À quoi correspond le nombre de points de la FFT ? Expliquer l'influence du nombre de points sur le résultat final, et donner la valeur de la résolution spectrale.

E7. Quelle transformation la fonction **DCRemoval()** applique au signal audio ?

E8. Quels sont les impacts des fonctions **Windowing()** et **ComplexToMagnitude()** sur le signal audio ?

E9. En utilisant le spectre audio généré et affiché sur l'écran OLED, vérifier que votre fréquence d'échantillonnage est correcte avec une fréquence audio de 1 kHz. De même avec 5 kHz, expliquer le résultat.

A. Mel-Frequency Cepstral Coefficients

Nous allons pouvoir utiliser l'algorithme MFCC afin de calculer les coefficients d'un enregistrement. L'objectif de cette partie va d'être de calculer les coefficients et de les afficher sur l'écran OLED et/ou sur le moniteur série. L'implémentation sur Arduino Due est fourni sur le [Github du Pole Electronique](#) (installer le .zip puis Sketch/Include Library/Add .ZIP Library). Attention à bien prendre la version 2.0.0 !

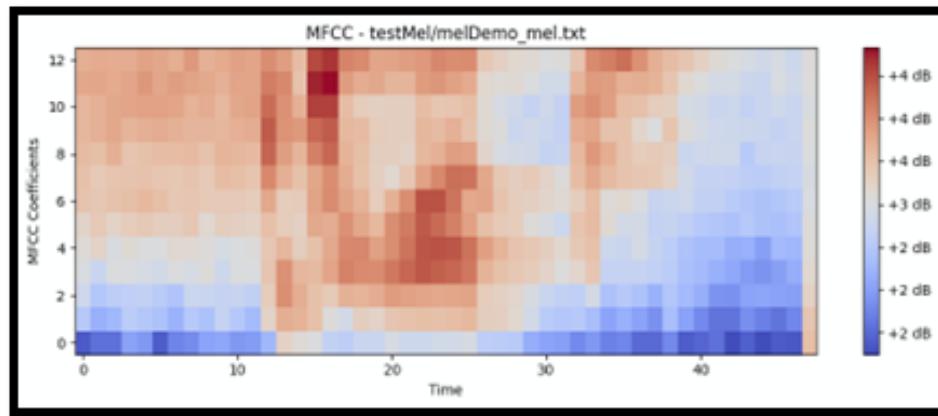


FIGURE 9.2 – Représentation graphique des coefficients MFCC

Dans cet exemple, on peut voir une représentation graphique des coefficients calculés. Il s'agit d'un tableau de 50 frames de 12 coefficients chacun. Les deux dimensions du tableau sont issues de la caractérisation temporelle et fréquentielle que les coefficients font du signal.

- Les coefficients sont ordonnés par fréquence, de la fréquence la plus basse à la plus haute en ordonnées.
- La première étape de l'algorithme consiste en la séparation de l'enregistrement en fragment, dit frame, de 256 échantillons chacune. Les coefficients associés à chacune de ces frames sont organisés de manière temporelle selon l'axe des abscisses. Ainsi, on peut voir l'évolution temporelle discrète des coefficients, et ainsi par exemple caractériser les différentes syllabes d'un mot.
- La couleur de chaque enregistrement correspond à l'énergie de la fréquence en question, c'est-à-dire du coefficient dans l'enregistrement.

ATTENTION ! Vous trouverez une bibliothèque qui a été développée pour l'implémentations sur Arduino Due de cet algorithme. Pour ce TP, nous ne traiterons que des enregistrements de 256 valeurs maximums, afin de n'avoir qu'une seule frame. Cependant, vous pourrez rencontrer dans d'autres applications, des enregistrements plus longs qui nécessiteront plusieurs frames afin de les traiter. Il faudra alors faire du recouvrement entre les frames avant le traitement MFCC. Dans ce tp, nous n'allons pas pouvoir traiter du caractère temporel du signal, même si nous allons nous en approcher en affichant les frames en décalé. Il est important de noter que l'algorithme s'exécute toujours sur un buffer d'une taille donnée et permet de caractériser uniquement les données de ce buffer.

E10. Après avoir téléversé le code d'exemple de la bibliothèque MFCC, expliquer à quoi correspond le résultat affiché sur la console. Pourquoi les méthodes `create_hamming_window()` et `create_mel_filter_bank()` sont-elles dans `setup` et non dans `loop` ?

E11. Expliquer, à l'aide du code source de la bibliothèque **ArduinoMFCC**, l'implémentation de chaque étape de l'algorithme MFCC permettant d'obtenir les coefficients finaux.

Réutiliser le code d'échantillonage du signal audio développé précédemment afin de pouvoir enregistrer votre propre frame à 8 kHz, à la place du signal audio écrit en dur dans le code d'exemple.

E12. Déterminer la durée de votre frame. En prenant en compte la taille de la mémoire flash de l'Arduino Due, quelle est la durée maximale d'un enregistrement à 8 kHz ?

E13. Afficher sur l'écran OLED les coefficients calculés à l'aide de la fonction disponible sur le [Github](#). Expliquer l'affichage avec un signal sonore de 3 kHz .

E14. Pour l'instant, nous n'avons utilisé que la fonction **compute**. La bibliothèque MFCC contient une autre fonction **computeDCT**. Faites une recherche documentaire sur l'algorithme DCT et les usages communs qui en sont faits, puis modifier le code pour afficher les coefficients MFCC après passage de la DCT. Conclure par la comparaison avec et sans l'algorithme.

TP n° 10

Filtre numérique RIF

Motivation

L'objectif du TP est de mettre en œuvre une fonction de filtrage RIF afin d'extraire certaines composantes spectrales d'un signal en utilisant le tampon de données (buffer) pour un traitement plus efficace et rapide. Les filtres RIF sont représentés par une combinaison linéaire des valeurs du signal d'entrée et peuvent être décrits par des équations de convolution discrètes. Ils sont naturellement stables car leur réponse est une somme finie de valeurs et peuvent être réalisés à l'aide d'éléments de base tels que le gain, la sommation et le retard unitaire. Différentes structures de filtre RIF peuvent être utilisées pour concevoir des systèmes de filtrage RIF. Dans le cadre de ce TP, nous utiliserons le microphone MAX9814 pour enregistrer un signal sonore, le filtrer à l'aide des modèles proposés dans la suite de ce TP et afficher la sortie à l'aide du DAC de l'Arduino Due. Enfin, il sera demandé de suivre les étapes pour réaliser un spectromètre du signal phonatoire, en tirant parti de ces technologies avancées pour améliorer les performances globales du système.

Afin de calculer les coefficients nécessaires pour implémenter les filtres RIF, il est possible d'utiliser de nombreux outils différents, comme MATLAB, ou des logiciels spécialisés. Pour ce TP, nous avons choisi d'utiliser le site suivant : <http://t-filter.engineerjs.com/>

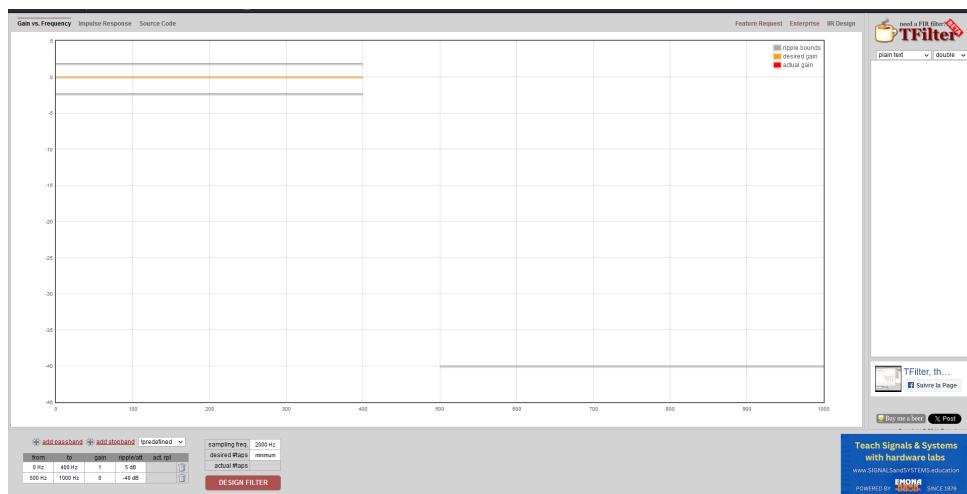


FIGURE 10.1 – Outil WEB pour générer les coefficients pour filtrage RIF

Il est possible de spécifier la bande de fréquence qui doit être atténuée en bas à gauche de la fenêtre (gain 0 pour la bande à supprimer et gain 1 pour la bande qui ne doit pas être filtrée). Il est important de bien connaître la fréquence d'échantillonnage et de la renseigner afin d'obtenir les coefficients adaptés. Voici un exemple de configuration pour un filtre passe-haut d'un signal échantillonné à 8 kHz :

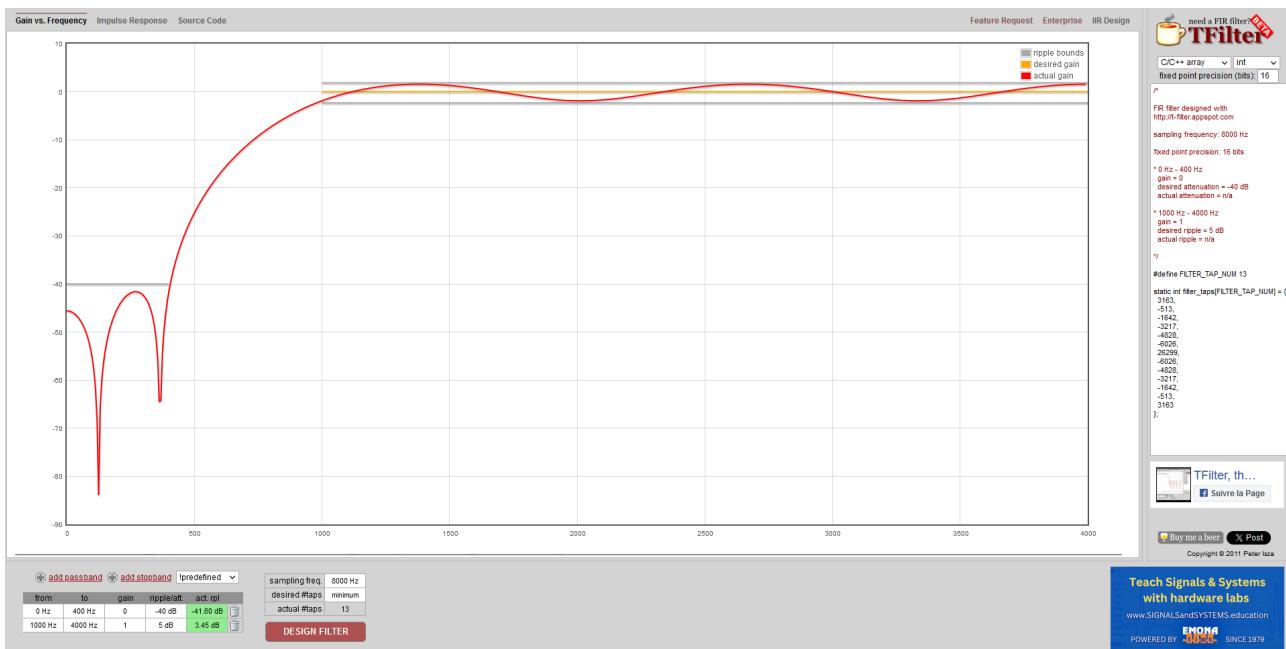


FIGURE 10.2 – Exemple de configuration d'un filtre numérique RIF à 8 kHz

Il est possible de récupérer les coefficients sous la forme d'un tableau en sélectionnant : « C/C+ array » puis le type de données souhaité (int_8, int_16 ou double) sur le panneau de droite. **Attention à la bande de transition de votre filtre :** plus cette dernière sera faible, plus il y aura de coefficients en sortie et donc plus votre temps de calcul sera grand.

A. Implémentation d'un Filtre RIF

(a) Filtre RIF

Pour ce TP, nous utiliserons le montage du microphone vu lors du TP2. L'objectif de cette partie est d'échantillonner un signal sonore afin de le filtrer numériquement à l'aide de l'Arduino Due.

E1. Configurer la fonction `setupADC()` en prenant en compte les spécifications suivantes :

- Activer le périphérique ADC et le configurer les paramètres de l'ADC sur 12 bits.
- Activer le canal 7 (A0) pour la lecture du signal analogique
- Configurer un Timer Counter pour générer une interruption à la fréquence d'échantillonnage de 44 kHz.
- Créer un buffer de données en sortie permettant d'enregistrer les échantillons.

Afin de pouvoir comparer l'audio filtré par rapport à l'audio d'origine, implémenter aussi la fonction DAC de l'Arduino Due dans votre programme.

On souhaite implémenter dans la chaîne d'acquisition de votre carte un filtre RIF classique suivant cette fonction :

$$Y[n] = x[n] - 0.9 \cdot x[n - 1]$$

E2. Modifier la taille de votre buffer de données en fonction de l'ordre de ce filtre. Expliquer votre raisonnement.

E3. Implémenter une fonction **filterSignal()** dans votre code puis réaliser un schéma bloc expliquant le fonctionnement du filtre.

(b) Filtre RIF d'ordre n avec Buffer FIFO

Nous allons maintenant voir comment nous pouvons implémenter un filtre de plus grand ordre. Plus nous voulons de la précision dans la fréquence de coupure, plus la zone de transition doit être faible et donc plus l'ordre du filtre doit être grand.

Pour pouvoir filtrer le signal en continu lors de l'acquisition, nous pouvons utiliser différents types de buffer, comme le buffer FIFO :

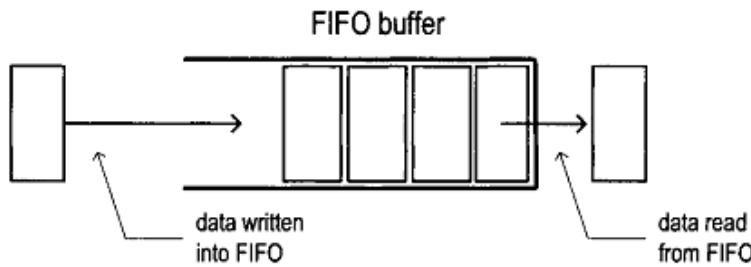


FIGURE 10.3 – Fonctionnement d'un buffer FIFO

Attention ! Pour optimiser l'impact sur la mémoire des échantillons, il est possible de les enregistrer sous la forme de `uint16_t`. Cependant, cela va poser un problème d'espace lors du calcul du filtre car les coefficients sont sous la forme de `uint16_t`. En effet, une multiplication de deux nombres sur 16 bits doit être enregistrée sur 32 bits.

Afin de contourner le problème en ayant toujours une gestion optimale de la mémoire, il donc nécessaire d'accumuler le résultat de la convolution des coefficients du filtre sur une variable de 32 bits, puis de reconvertis les données sur 16 bits afin d'optimiser la mémoire en sortie. Pour reconvertis les données, il suffit de faire faire un décalage de 15 bits au buffer de 32 bits à l'aide de l'opérateur bit shift pour récupérer les valeurs en `uint16_t`

E4. Implémenter un filtre RIF suivant les contraintes suivantes à l'aide du code fourni sur **Github** :

- $F_c = 2 \text{ kHz}$, Zone de transition = 1 kHz
- Le filtre doit être implanté avec une gestion de la mémoire à l'aide d'un mémoire de type FIFO. Vous devez dimensionner le buffer afin que le décalage de phase (temps entre l'acquisition ADC et sortie sur le DAC des données filtrées) soit le plus faible possible.
- Vous conserverez la fréquence d'échantillonage de 44 kHz des questions précédentes. Les coefficients du filtre seront à trouver à l'aide du site web [tfILTER](#).

(c) Filtre RIF d'ordre n avec buffer circulaire

Nous allons maintenant voir un nouveau type de gestion de données afin de gérer le vieillissement des données en mémoire. Le buffer circulaire permet de gérer les données sans avoir à les déplacer en mémoire. Cela permet de réaliser moins d'opération à chaque filtrage.

L'implémentation du buffer circulaire est donnée sur le Github, et suit le schéma suivant :

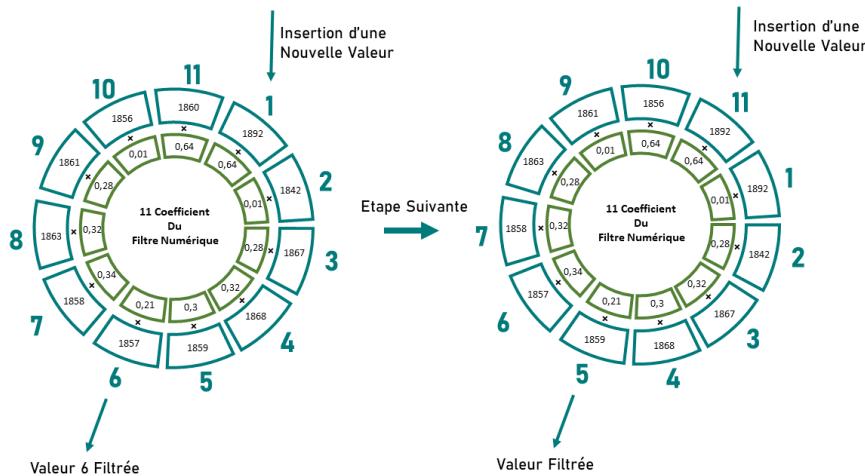


FIGURE 10.4 – Buffer circulaire

E5. Réaliser l'algorigramme du code du buffer circulaire fourni sur Github. Expliquer les différentes étapes et en quoi il permet de gérer le vieillissement des données en mémoire.

E6. A l'aide du buffer circulaire, implémenter un filtre RIF dont la fréquence de coupure se situe à 10 kHz et qui permet de filtrer le signal en moins de 22 µs avec une fréquence d'échantillonnage à 44 kHz. Vous devrez faire varier le nombre de coefficients, et donc la taille de la bande de fréquence.

E7. Comparer les performances entre les deux méthodes de gestion du buffer avec un filtre (réponse impulsionnelle à l'aide de l'oscilloscope en envoyant les données filtrées directement sur le DAC de l'Arduino Due) :

$$F_c = 15 \text{ kHz}, \quad F_e = 44 \text{ kHz}, \quad \text{Nombre de coefficients} = 35$$