



**Institut Supérieur de l'Electronique et du
Numérique**

20, RUE CUIRASSE BRETAGNE
CS 42807 29228 BREST CEDEX 2 FRANCE
TEL : 02.98.03.84.00

35 AVENUE DU CHAMP DE MANŒUVRE
44470 CARQUEFOU
TEL : 02 40 52 40 35

PROJET A3

OXYMETRIE DE POULS

-PARTIE INFORMATIQUE-

Version 9.2 mise à jour le lundi 30 mars 2020

Pierre-Jean BOUVET

TABLE DES MATIERES

1.	Introduction.....	4
2.	Lecture des champs	6
a)	Relevé USB	7
b)	Conversion	8
3.	Filtrage des signaux	11
a)	FIR	11
b)	IIR	14
4.	Mesures.....	17
a)	Calcul de SP02	17
b)	Calcul du rythme cardiaque	18
5.	Affichage	19
6.	Contraintes générales.....	20
7.	Evaluation.....	20
a)	Recette	20
b)	Qualimétrie	23
c)	QCM	26
8.	Annexe : fonctions imposées pour les tests automatiques..	26
a)	Fichier de définition « define.h »	27
b)	FIR	27
c)	IIR	27
d)	Mesure	28
e)	Affichage	28
f)	Intégration	29
g)	Conseils	30

9.	Annexe : Fichiers de tests	31
10.	Références	32

1. INTRODUCTION

Le projet A3 2019-2020 consiste à réaliser un équipement médical appelé « oxymètre de pouls » qui permet de mesurer la fréquence cardiaque et la saturation en oxygène du sang (SpO_2) d'un individu grâce à un capteur photo-électrique non invasif situé en général à l'extrémité du doigt du patient [1].

Comme le montre la Figure 1, la partie informatique consiste à développer un module permettant de calculer et d'afficher la fréquence cardiaque et le taux de SpO_2 à partir des mesures d'absorption de la lumière rouge et de l'infrarouge fournies par le module « interface capteur oxymètre ».

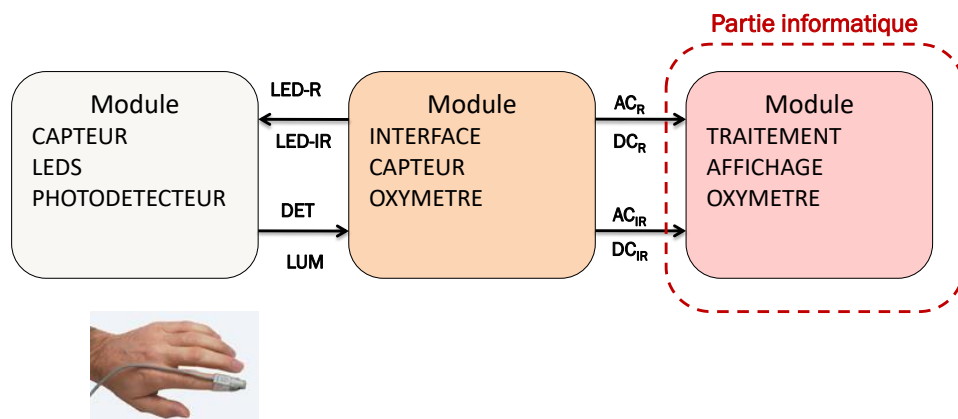


FIGURE 1 : SCHEMA GENERAL DU PROJET N3

Les mesures d'absorption fournies pour le module précédent sont les suivantes :

- AC_R : variation crête à crête de la lumière rouge mesurée (onde de pouls)
- DC_R : valeur moyenne de la lumière rouge mesurée (tissus)
- AC_{IR} : variation crête à crête de la lumière infrarouge mesurée (onde de pouls)
- DC_{IR} : valeur moyenne de la lumière infrarouge mesurée (tissus)

Ces 4 mesures sont transmises tous les 2 ms par liaison USB au module informatique (le format de trame sera décrit plus loin)

Le module informatique, intitulé « Traitement et affichage oxymètre » doit effectuer les opérations suivantes (voir Figure 2) :

- Lecture sur périphérique USB des signaux AC_R , DC_R , AC_{IR} et DC_{IR} (optionnel pour les CGSI3 / CBIO3 / CENT3)
- Filtrage numérique des signaux
- Mesure du SpO_2 et de la fréquence cardiaque

- Affichage des mesures via une interface JAVA fournie par l'équipe enseignante.

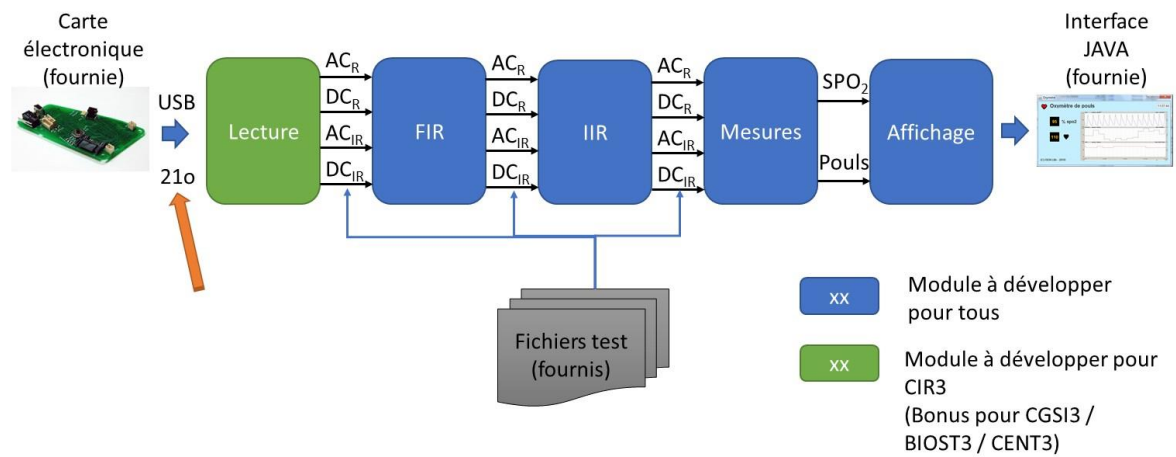


FIGURE 2 : SYNOTIQUE DE LA PARTIE INFORMATIQUE

2. LECTURE DES CHAMPS

Cette partie est optionnelle pour les cycles suivant :

- CGSI3
- CBIO3
- CENT3

Les mesures en provenance du module « interface capteur oxymètre » sont stockées sous le format suivant :

Taille (octets)	4	1	4	1	4	1	4	1	1
Champ	AC _R	,	DC _R	,	AC _{IR}	,	DC _{IR}	LF	CR

TABLEAU 1 : FORMAT DE LECTURE DES CHAMPS (21 OCTETS PAR LIGNE)

La carte électronique écrit toutes les 2 ms une ligne de mesure sur le port USB de la carte. Chaque champ est écrit sous format texte, i.e. sous la forme d'une série de caractères. Les valeurs numériques (AC_R, DC_R, AC_{IR} et DC_{IR}) quant à elles sont écrites chacune sous la forme de 4 caractères représentant des valeurs allant de « 0000 » à « 4095 ». On rappelle que tout caractère est représenté par son code ASCII dont on donne la table en Figure 3. On note ainsi que les champs « , », « LF » et « CR » sont des caractères auxquels sont associés des codes ASCII.

ASCII Character Set											
Hex	Dec	Character	Hex	Dec	Character	Hex	Dec	Character	Hex	Dec	Character
00	00	NUL	20	32	space	40	64	@	60	96	`
01	01	SOH	21	33	!	41	65	A	61	97	a
02	02	STX	22	34	"	42	66	B	62	98	b
03	03	ETX	23	35	#	43	67	C	63	99	c
04	04	EOT	24	36	\$	44	68	D	64	100	d
05	05	ENQ	25	37	%	45	69	E	65	101	e
06	06	ACK	26	38	&	46	70	F	66	102	f
07	07	BEL	27	39	'	47	71	G	67	103	g
08	08	BS	28	40	(48	72	H	68	104	h
09	09	HT	29	41)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	
1D	29	GS	3D	61	=	5D	93]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

ASCII Control Codes							
NUL	Null	HT	Horizontal Tab	DC2	Device Control 2	EM	End of Medium
SOH	Start of Heading	LF	Line Feed	DC3	Device Control 3	SUB	Substitute
STX	Start of Text	VT	Vertical Tab	DC4	Device Control 4	ESC	Escape
ETX	End of Text	FF	Form Feed	NAK	Negative Acknowledge	FS	File Separator
EOT	End of Transmission	CR	Carriage Return	SYN	Synchronous Idle	GS	Group Separator
ENQ	Enquiry	SO	Shift Out	ETB	End of Transmission Block	RS	Record Separator
ACK	Acknowledge	SI	Shift In	CAN	Cancel	US	Unit Separator
BEL	Bell	DLE	Data Link Escape			DEL	Delete
BS	Back Space	DC1	Device Control 1				

FIGURE 3 : TABLE DU CODE ASCII

Le Tableau 2 montre un exemple d'une ligne mesure.

Champ	ACr				,	DCr				,	ACiR				,	DCiR				LF	CR
Valeur	2	0	8	5	,	2	0	3	0	,	2	0	2	7	,	2	0	3	0	LF	CR
Code ASCII (hexa)	32	30	38	35	2C	32	30	33	30	2C	32	30	32	37	2C	32	30	33	30	0A	0D

TABLEAU 2 : EXEMPLE D'UNE LIGNE DE MESURE

Point sensible : Etablir la synchronisation afin de récupérer les valeurs.

a) RELEVÉ USB

Le relevé s'effectue via une liaison USB 2.0 entre la carte électronique et le micro-ordinateur :

- interface de transfert FPGA vers USB par le circuit FT245R de la société FTDI
- protocole décrit ci-dessus

Les drivers FTDI ainsi que la procédure d'installation (sous Windows) sont disponible sur l'intranet.

Du fait du confinement due à l'épidémie COVID-19, l'accès à la carte électronique est rendu impossible. La lecture sur le port USB sera simulée. A cet effet nous avons créé un fichier nommé record1_bin.dat contenant les données à lire sous format binaire. Pour lire un octet dans le fichier, vous pourrez utiliser la fonction fgetc().

b) CONVERSION

Chaque champ est représenté par une valeur entière comprise entre 0 et 4095. La Figure 4 et la Figure 5 montrent un exemple de mesures obtenues avec les composantes AC_R et DC_R .

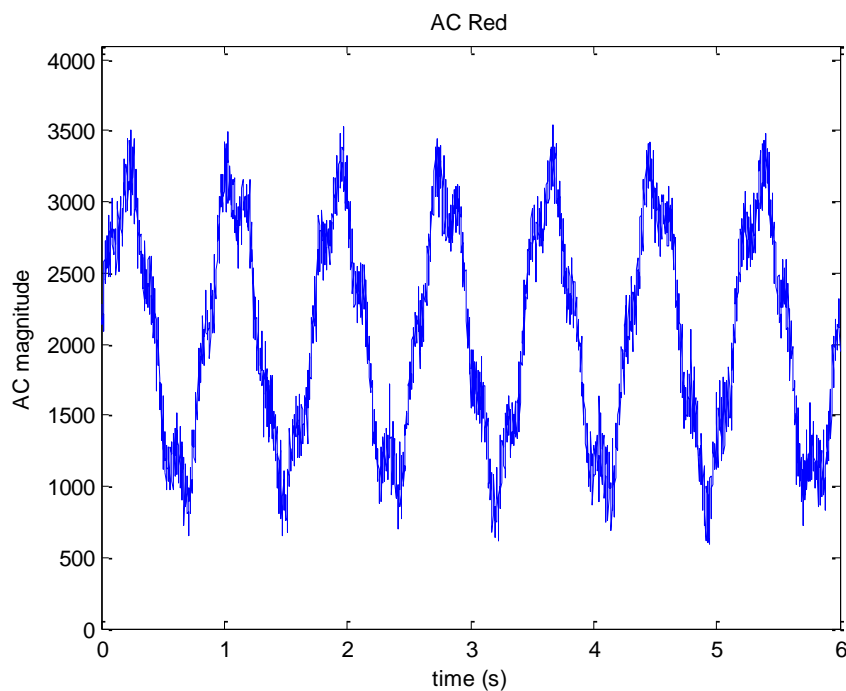


FIGURE 4 : EXEMPLE DE MESURE OBTENUE POUR LA COMPOSANTE AC_R

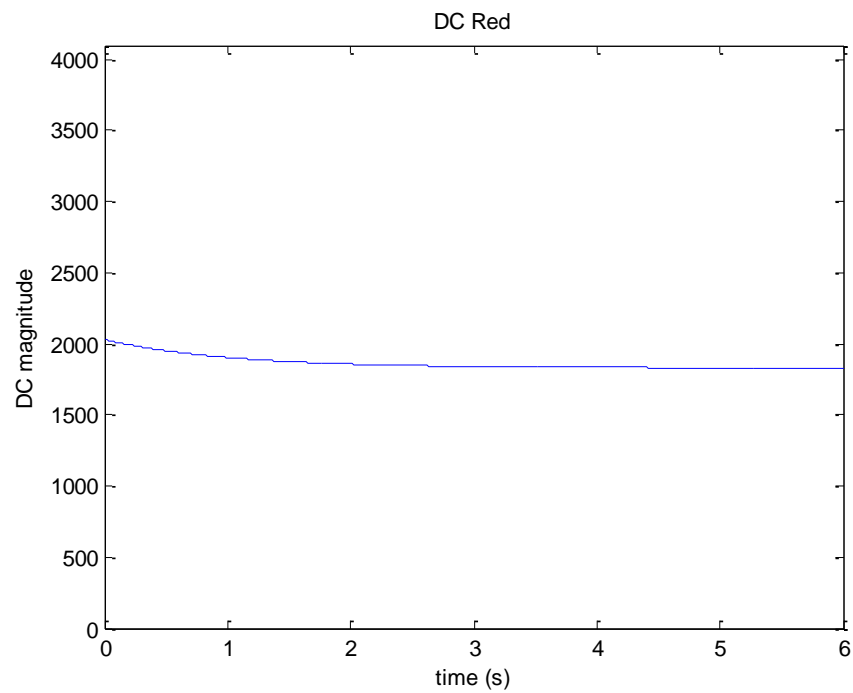


FIGURE 5 : EXEMPLE DE MESURE OBTENUE POUR LA COMPOSANTE DC_R

Afin de rendre possible les opérations du filtrage il convient de recadrer les 2 composantes alternatives (AC_R et AC_{IR}) autour de 0, comme le montre la Figure 6. Ce recadrage peut-être approximatif, il sert simplement à faciliter les équations des filtres.

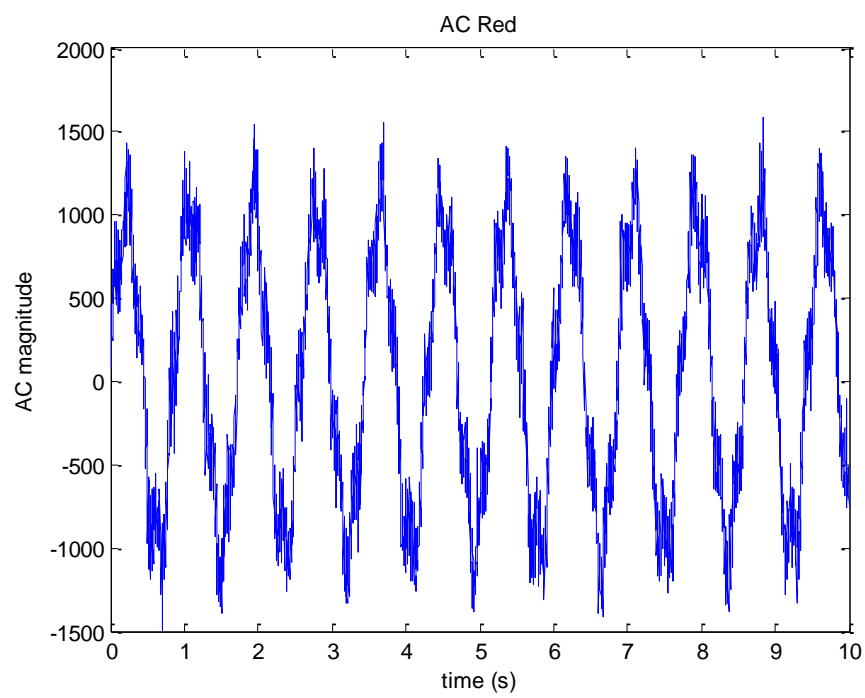


FIGURE 6 : EXEMPLE DE COMPOSANTE AC_R RECADREE AUTOUR DE 0

3. FILTRAGE DES SIGNAUX

a) FIR

Les composantes AC des mesures en rouge et infrarouge sont filtrées au moyen d'un filtre à réponse finie (FIR) [2]. Ce filtre a pour but d'éliminer les composantes hautes fréquences et ne garder que le signal utile qui est sensé se trouver autour d'une fréquence de 1 Hz.

On rappelle qu'un filtre FIR se représente sous la forme de sa transformée en Z de la façon suivante :

$$H(z) = \sum_{k=0}^{L-1} h_k \cdot z^{-k}$$

Où $(h_0, h_1, \dots, h_{L-1})$ sont les coefficients du filtre et L l'ordre du filtre. Par définition la relation entre le signal de sortie $y[n]$ et le signal d'entrée $x[n]$ s'exprime comme suit :

$$y[n] = \sum_{k=0}^{L-1} h_k \cdot x[n-k]$$

Pour le projet, nous choisirons un filtre de type fenêtre de Hamming d'ordre 51 dont les coefficients sont donnés dans le Tableau 3.

k	h_k	k	h_k	k	h_k
0	1.4774946e-019	17	3.1294938e-002	34	2.7892178e-002
1	1.6465231e-004	18	3.4578348e-002	35	2.4459630e-002
2	3.8503956e-004	19	3.7651889e-002	36	2.1082435e-002
3	7.0848037e-004	20	4.0427695e-002	37	1.7838135e-002
4	1.1840522e-003	21	4.2824111e-002	38	1.4793934e-002
5	1.8598621e-003	22	4.4769071e-002	39	1.2004510e-002
6	2.7802151e-003	23	4.6203098e-002	40	9.5104679e-003
7	3.9828263e-003	24	4.7081811e-002	41	7.3374938e-003
8	5.4962252e-003	25	4.7377805e-002	42	5.4962252e-003
9	7.3374938e-003	26	4.7081811e-002	43	3.9828263e-003
10	9.5104679e-003	27	4.6203098e-002	44	2.7802151e-003
11	1.2004510e-002	28	4.4769071e-002	45	1.8598621e-003
12	1.4793934e-002	29	4.2824111e-002	46	1.1840522e-003
13	1.7838135e-002	30	4.0427695e-002	47	7.0848037e-004
14	2.1082435e-002	31	3.7651889e-002	48	3.8503956e-004
15	2.4459630e-002	32	3.4578348e-002	49	1.6465231e-004
16	2.7892178e-002	33	3.1294938e-002	50	1.4774946e-019

TABLEAU 3 : COEFFICIENTS DU FILTRE FIR

En langage C, les coefficients du filtre sont stockés sous la forme d'un tableau. Pour ne pas avoir à recopier les valeurs, vous trouverez sur l'intranet dans la section « Ressources », un code en langage C déclarant les coefficients du filtre.

On montre que ce filtre est de type passe- base avec une fréquence de coupure de 10 Hz. Avec une fréquence d'échantillonnage de $F_s = 500$ Hz, on obtient les réponses en temps et en fréquence suivantes :

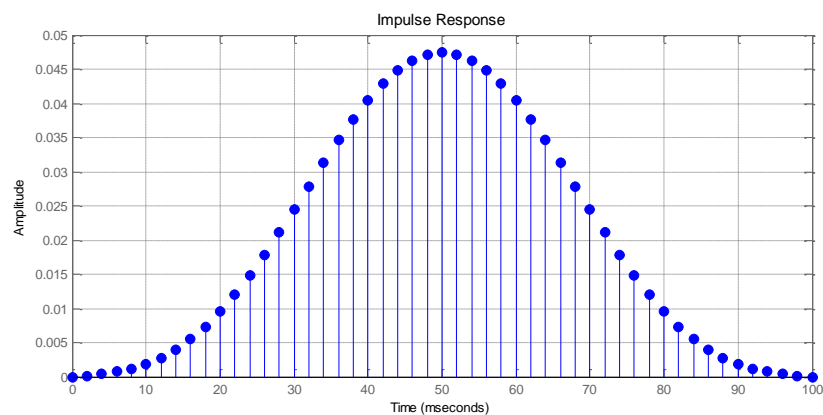


FIGURE 7: REPONSE IMPULSIONNELLE DU FILTRE FIR

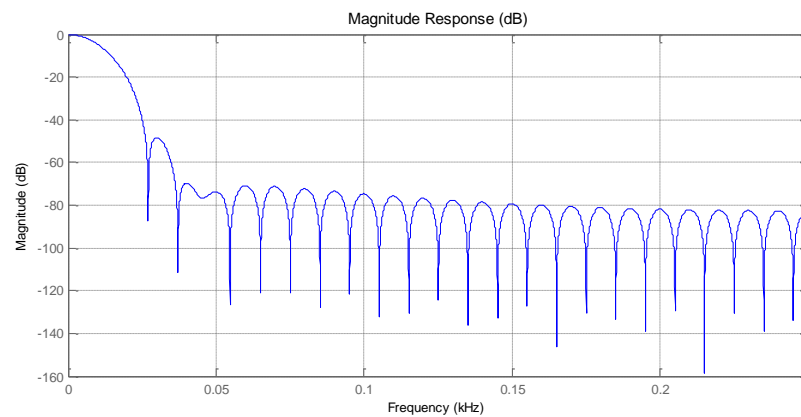


FIGURE 8 : REPONSE FREQUENTIELLE DU FILTRE FIR

On rappelle qu'il convient de filtrer seulement les composantes AC_R et AC_{IR} et de garder inchangés les composantes DC_R et DC_{IR} .

Pour réaliser ce type de filtre, il est d'usage d'utiliser un buffer permettant de mémoriser les $L-1$ derniers échantillons $x(k)$ ainsi que l'échantillon courant $x(n)$. Une fois la valeur $y(n)$ calculée, le buffer est décalé vers la droite afin d'insérer le nouvel échantillon entrant comme le montre la Figure 9.

Afin d'éviter de recopier manuellement les échantillons lors de chaque décalage on pourra utiliser le principe du buffer circulaire vu en cours (ce n'est pas une obligation !!!).

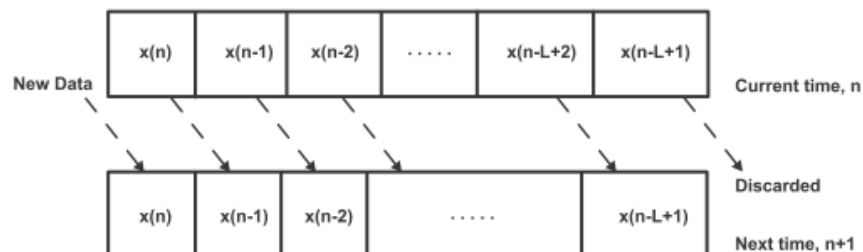
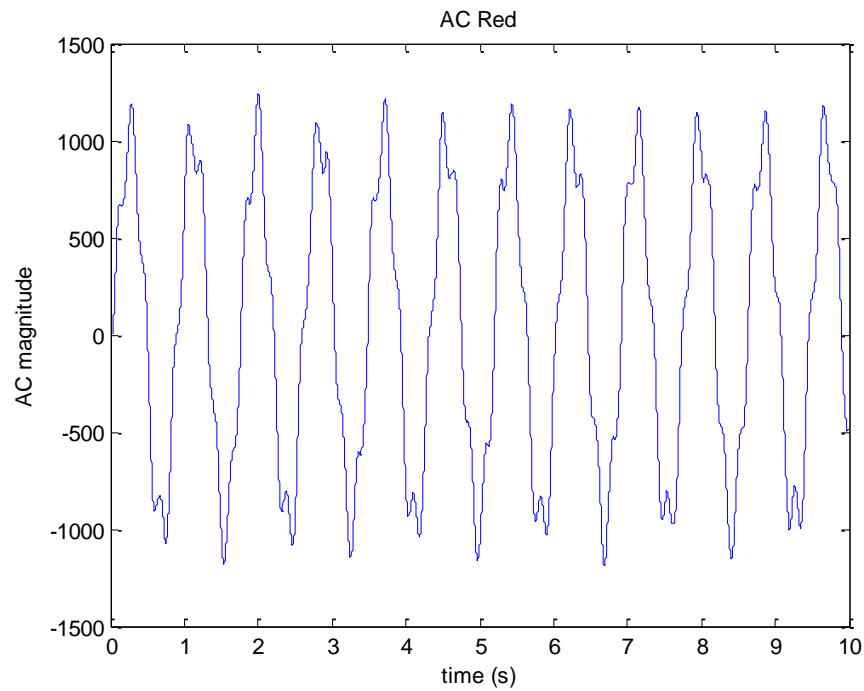


FIGURE 9 : BUFFER A DECALAGE

La Figure 10 montre un exemple de résultat de la composante AC_R obtenue après filtrage FIR.

FIGURE 10 : EXEMPLE DE COMPOSANTE AC_R APRES FILTRAGE FIR

b) IIR

Après filtrage FIR, les composantes AC résultantes sont ensuite traitées au moyen d'un filtre à réponse infinie (IIR) [3]. Ce filtre sert à supprimer la composante continue résultante. Dans le projet nous utiliserons le filtre de transformée en Z suivante :

$$H(z) = \frac{1 - z^{-1}}{1 - \alpha \cdot z^{-1}}$$

On rappelle que la transformée en Z précédente implique la relation suivante entre le signal de sortie $y[n]$ et le signal d'entrée $x[n]$:

$$y(n) = x[n] - x[n-1] + \alpha \cdot y[n-1]$$

Dans le projet nous choisirons une valeur de α égale à 0.992

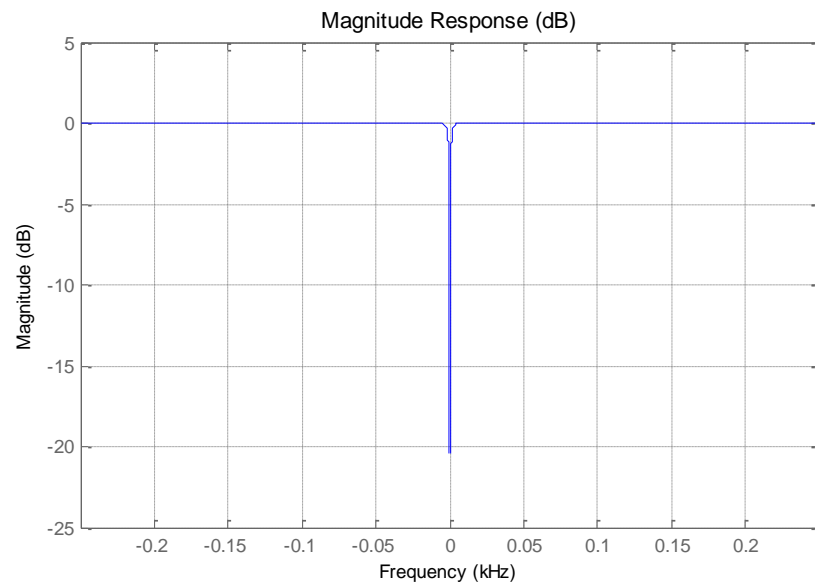


FIGURE 11 : REPONSE FREQUENTIELLE DU FILTRE IIR

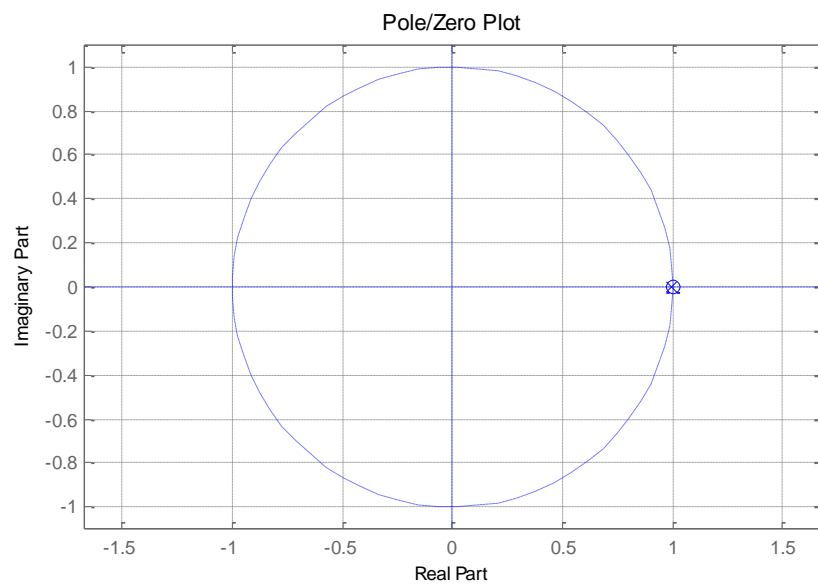


FIGURE 12 : EMBLEMENT DES POLES ET DES ZEROS DU FILTRE IIR

La Figure 13 montre un exemple de résultat obtenu sur la composante AC_R après filtrage IIR.

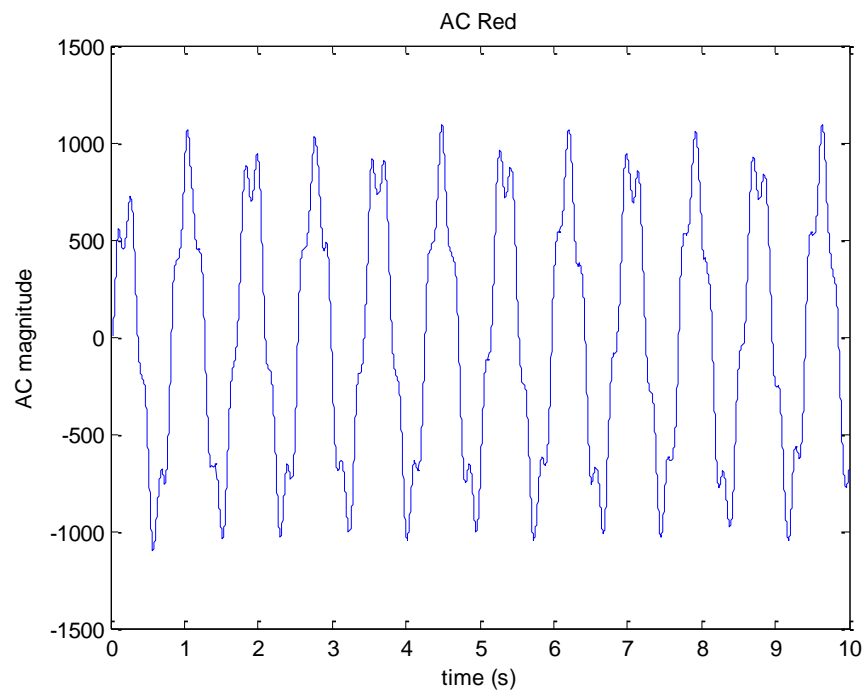


FIGURE 13 : EXEMPLE DE COMPOSANTE AC_R APRES FILTRAGE IIR

On rappelle qu'il convient de filtrer seulement les composantes AC_R et AC_{IR} et de garder inchangés les composantes DC_R et DC_{IR} .

4. MESURES

a) CALCUL DE SP02

Le taux de saturation de l'oxygène dans le sang s'obtient à partir des valeurs filtrées de AC_R et AC_{IR} ainsi que des valeurs DC_R et DC_{IR} en calculant la rapport suivant :

$$RsIR = \frac{\frac{PtP(AC_R)}{DC_R}}{\frac{PtP(AC_{IR})}{DC_{IR}}}$$

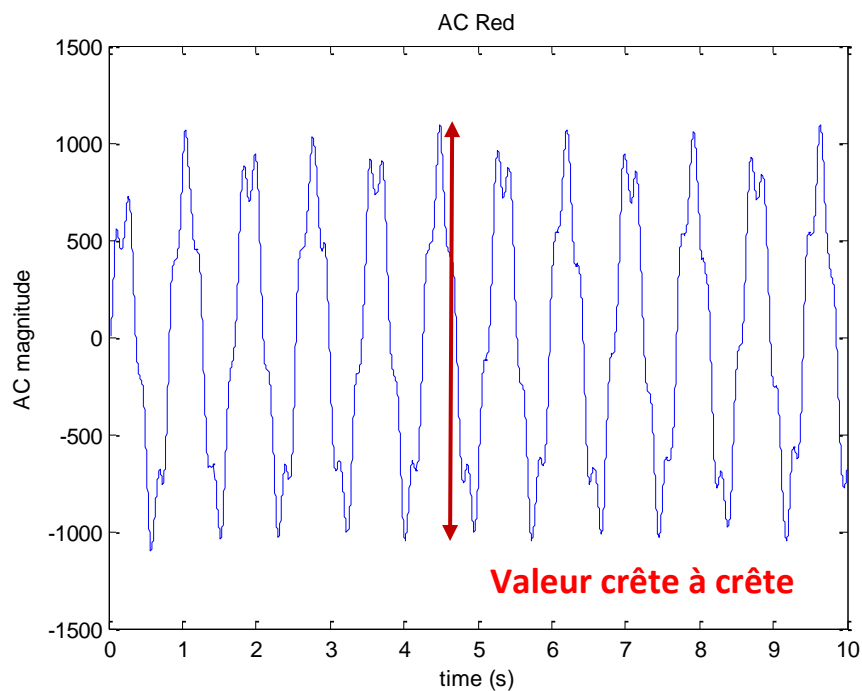


FIGURE 14 : CALCUL DE LA VALEUR CRÊTE À CRÊTE DE LA COMPOSANTE AC R

Où $PtP(x)$ correspond à la valeur crête à crête (peak-to-peak en anglais) du signal x .

On utilise ensuite la table de correspondance suivante :

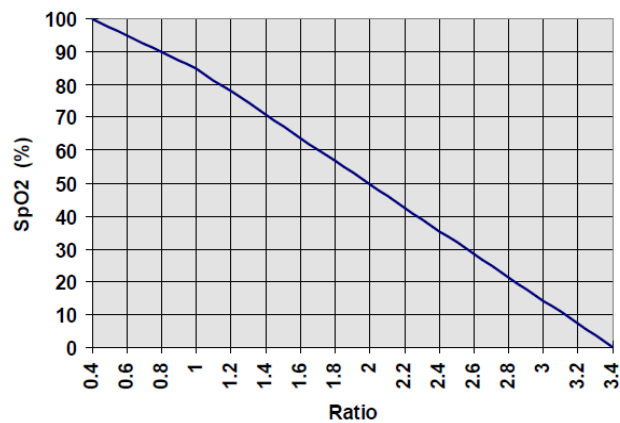


FIGURE 15 : CORRESPONDANCE ENTRE RSIR ET SPO2

b) CALCUL DU RYTHME CARDIAQUE

Le pouls se mesure en estimant la fréquence des signaux AC_R et AC_{IR} (qui doit être à peu de chose près identique).

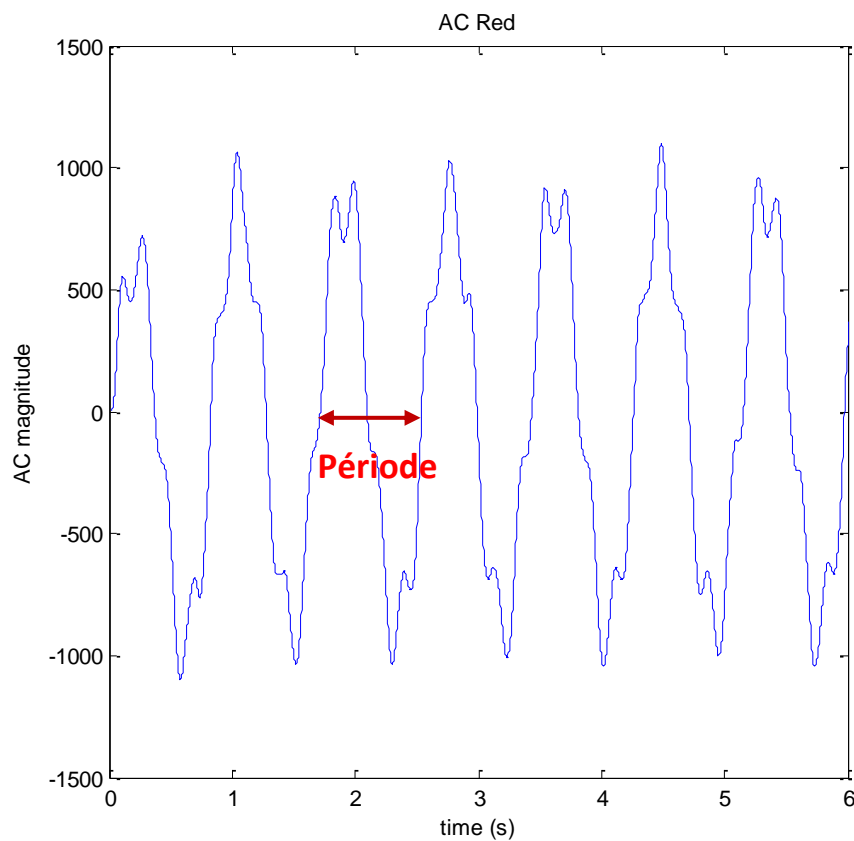


FIGURE 16 : CALCUL DU RYTHME CARDIAQUE SUR LA COMPOSANTE AC_R

5. AFFICHAGE

Une interface utilisateur a été développée en langage Java à votre intention :

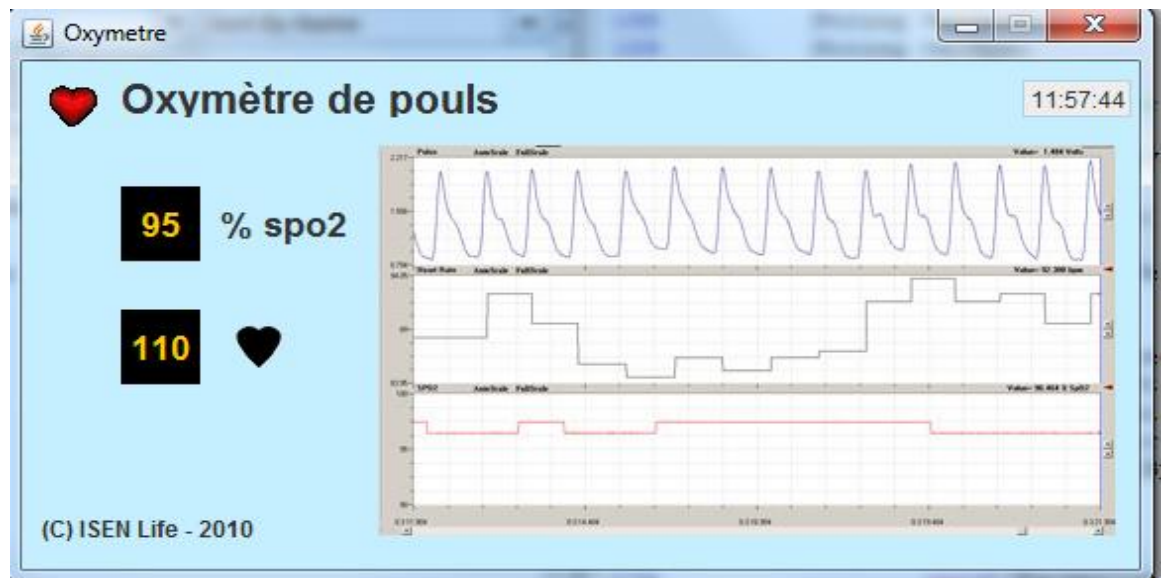


FIGURE 17 : INTERFACE UTILISATEUR FOURNIE

Les données qui apparaissent sur cette interface sont lues dans le fichier `Data.txt` (mise à jour toutes les secondes) :

- taux de saturation de l'oxygène dans le sang (SPO2) : type entier
- rythme cardiaque (BPM) : type entier

Afin d'être sûr de ne pas avoir de problèmes de concurrence entre les programmes, les accès en lecture ou écriture au fichier `Data.txt` sont protégés par un verrou. Il n'est pas possible de lire et écrire en même temps dans un fichier. C'est ce qu'on appelle l'exclusion mutuelle. Ce mécanisme de verrou est mis en œuvre le fichier [.verrouData](#). La présence du fichier verrou indique qu'une opération de lecture ou écriture est en cours sur le fichier de données correspondant. L'absence de verrou autorise la lecture ou l'écriture. Ne pas oublier la gestion du verrou !!!

6. CONTRAINTES GENERALES

Afin de pouvoir fonctionner en temps réel, on demande que votre programme soit capable de calculer un résultat pour chaque mesure d'oxymétrie donnée en entrée du programme. Autrement dit pour une mesure (AC_R , DC_R , AC_{IR} et DC_{IR}) votre programme doit sortir et afficher un résultat de SpO_2 et de rythme cardiaque (voir Figure 18). Il va de soi que les premiers résultats seront erronés du fait du manque de mesures prises en compte.

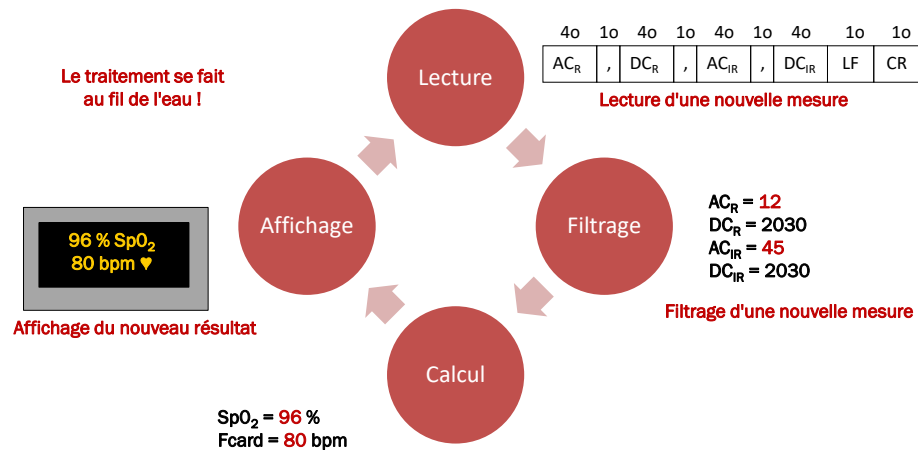


FIGURE 18 : ENCHAINEMENT DES BLOCS

7. EVALUATION

La barème prévisionnel du projet est décrit dans le Tableau 4.

Métrique	Coefficient
Recette programme	6
Qualimétrie code	2
QCM	2
Total	10

TABLEAU 4 : BAREME PREVISIONNEL DU PROJET

a) RECETTE

MODALITES

#	Nom du test	Type	Barème A3 hors CI3	Barème CIR3
1	FIR	Automatique	3,5	3

#	Nom du test	Type	Barème A3 hors CI3	Barème CIR3
2	IIR	Automatique	3	2,5
3	Mesure SPO2	Automatique	2,5	2
4	Mesure Pouls	Automatique	2,5	2
5	Affichage	Automatique	3	2,5
6	Intégration	Automatique	3,5	3
7	Programme global simulation	Manuel	2	2
8	Lecture	Automatique	Bonus	3
Total			20	20

TABLEAU 5 : BAREME PREVISIONNEL DE LA RECETTE

La barème de la recette de votre programme est indiqué dans le Tableau 5. La recette de votre programme sera effectuée le dernier jour entre 15:00 et 18:00. Selon le travail que vous aurez effectué, vous devrez rendre avant 15 :00 le jour de la recette deux archives distinctes contenant vos sources :

- Sources pour tests automatiques :
 - Attention, le contenu est imposé (voir ci-dessous)
 - Les prototypes des fonctions sont également imposées (voir annexe)

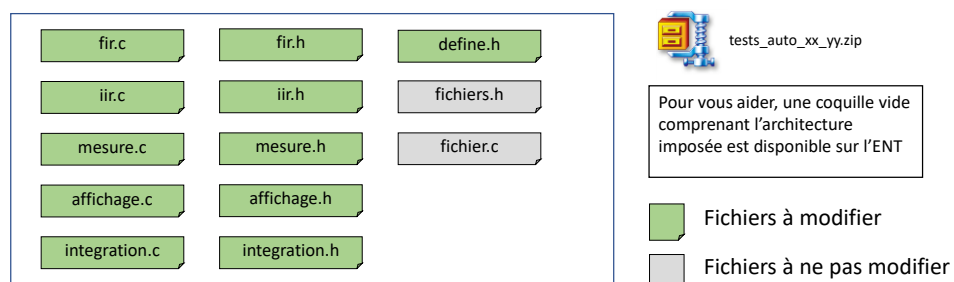


FIGURE 19 : ARCHIVE POUR TESTS AUTOMATIQUES

- Sources pour les tests manuels (simulation ou USB) : en plus des fichiers compris dans les sources pour les tests automatiques vous devez fournir les fichiers suivants :

- lecture.h (si USB)
- lecture.c (si USB)
- main.c
- Makefile

Tout retard lors de la livraison des sources sera systématiquement sanctionné (l'heure du réseau faisant foi).

Lors de la recette, un outil anti-plagiat sera utilisé pour vérifier qu'il n'y a pas eu de recopie entre étudiants ou avec du code sur internet. Attention, toute distance informationnelle inférieure à 0.5 entre 2 codes entraînera au minimum une pénalité de 50% pour chaque auteur.

Si lors de la recette, les tests automatiques ou manuels ne peuvent être effectués (erreur de compilation ou d'exécution), un repassage plus tard dans l'après-midi peut être proposé, une pénalité (substantielle) sera alors appliquée.

TESTS AUTOMATIQUES

Un environnement de tests automatiques est accessible sur l'intranet à l'onglet Formation/tests automatiques pendant toute la durée du projet. Pour les CGSI3, CENT3 et BIST3

- Etape 1 : Dans l'onglet « Console », choisir le projet vous concernant :
 - **Oxymétrie** : pour les promotions CGSI3, CBIO3 et CENT3
 - **Oxymétrie-CIR3** : pour les promotions CIR3
- Etape 2 : chargement de votre projet du fichier *.zip contenant vos sources

Tests Welcome Projects Console Results

Console

Control

Project
Oxymetrie

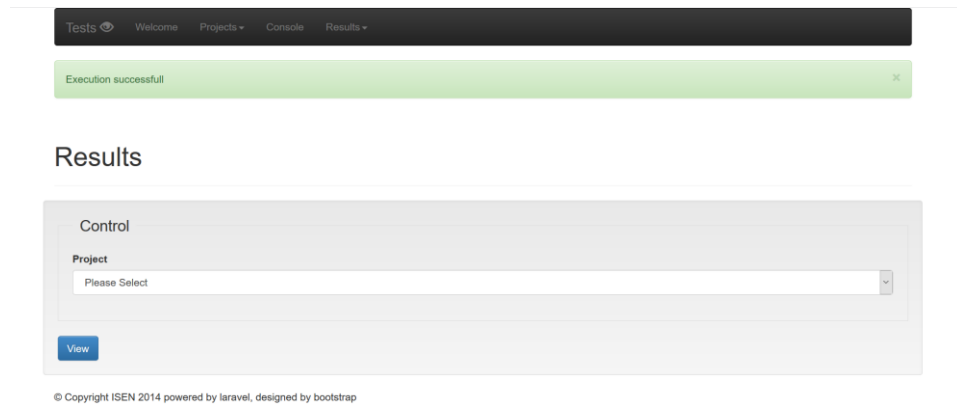
Source upload
Parcourir... Tests.zip

Student
eleven3 EleveN3

Launch

© Copyright ISEN 2014 powered by laravel, designed by bootstrap

- Etape 3 : compilation de votre projet sur le serveur



- Etape 4 : Dans l'onglet « Results », visualisation de vos résultats (vert : OK, rouge : KO, si rien ne s'affiche cela veut dire que tout est KO)

General informations

Project: Oxymetrie
Student: eleven3 Eleve

[Download Src](#)

Details

Function	Tests
affichage	0 / 2
fir	0 / 1
iir	0 / 1
intergration	0 / 2
lecture	3 / 3
mesure	0 / 2

b) QUALIMETRIE

PRESENTATION

SONAR est un logiciel libre permettant de mesurer la qualité du code source en continu et utilisé fréquemment dans le monde professionnel. Pour le projet, le logiciel SONAR sera utilisé par l'encadrant durant la phase de recette pour auditer le code livré par les étudiants.

NOTATION

- Commentaires
 - <5% du code → 0/20

- ...
 - >20% du code → 20/20
- Duplication de code
 - >20% du code → 0/20
 - ...
 - <5% du code → 20/20
- Respect des règles (mineures ou majeures)
 - 80% des règles respectées → 0/20
 - ...
 - 100 % des règles respectées → 20/20
- Respect des règles critiques
 - **Violation d'une règle → 0/20 pour la note totale de Qualimétrie**

REGLES

15 règles sont définies dans le logiciel, elles sont classées en 3 catégories :

- Mineure
- Majeure
- Critique

Elles sont définies comme suit :





<input checked="" type="checkbox"/> Major	<u>Avoid empty block</u> In most case, en empty block represents a missing implementation. Key: C.EmptyBlock
<input checked="" type="checkbox"/> Minor	<u>Avoid file with too many lines of code</u> Violations of this rule usually indicate that the file is doing too much. Try to reduce the file size by splitting it into several other ones. maximumFileLocThreshold: <input type="text" value="1000"/>  <input type="button" value="Update"/> The maximum authorized lines of code. Key: C.FileLoc
<input checked="" type="checkbox"/> Major	<u>Avoid function with too many lines of code</u> Violations of this rule usually indicate that the method is doing too much. Try to reduce the method size by extracting sub methods and removing any copy/pasted code. maximumFunctionLocThreshold: <input type="text" value="100"/>  <input type="button" value="Update"/> The maximum authorized lines of code. Key: C.FunctionLoc
<input checked="" type="checkbox"/> Major	<u>Avoid function with too many parameters</u> Long parameter lists can indicate that a new structure should be created to wrap the numerous parameters or that the function is doing to many things. maximumFunctionParameters: <input type="text" value="7"/>  <input type="button" value="Update"/> The maximum authorized number of parameters. Key: C.ExcessiveParameterList
<input checked="" type="checkbox"/> Major	<u>Avoid switch statement without a "default" clause</u> It's usually a good idea to introduce a default case in every switch statement. Even if the developer is sure that all currently possible cases are covered, this should be expressed in the default branch. This way the code is protected against later changes, e.g. introduction of new types. Key: C.SwitchStatementWithoutDefault
<input checked="" type="checkbox"/> Major	<u>Avoid too complex function</u> The cyclomatic complexity of a function should not exceed a defined threshold. Complex code can perform poorly and will in any case be difficult to understand and therefore to maintain. maximumFunctionComplexityThreshold: <input type="text" value="200"/>  <input type="button" value="Update"/> The maximum authorized complexity in function. Key: C.FunctionComplexity

FIGURE 20 : REGLES DE QUALIMETRIE (1 A 6)

☒ Critical

Avoid using 'continue' branching statement
 The use of the 'continue' branching statement increase the essential complexity of the source code and so prevent any refactoring of this source code to replace all well structured control structures with a single statement.

 For instance, in the following java program fragment, it's not possible to apply the 'extract method' refactoring pattern :


```

mylabel : for(int i = 0 ; i < 3; i++) {
  for (int j = 0; j < 4 ; j++) {
    doSomething();
    if (checkSomething()) {
      continue mylabel;
    }
  }
}

```

 Key: C.DoNotUseContinue

☒ Major

Boolean expression complexity
 Restricts nested boolean operators (&&, || and ^) to a specified depth.

 maximumNestedBooleanOperators: The maximum number of nested boolean operators that are authorized.
 Key: C.BooleanExpressionComplexity

☒ Critical

Break statement must be used only inside a switch block
 Avoid using keyword break outside a switch block.

 Key: C.DoNotUseBreak

☒ Minor

Collapsible if statements
 Several 'if' statements can be consolidated by separating their conditions with a boolean short-circuit operator.

 Key: C.CollapsibleIfStatement

☒ Minor

For loops must use braces
 Avoid using For loops without using curly braces.

 Key: C.ForLoopsWithoutBraces

☒ Critical

Goto statement must not be used
 Avoid using goto statement.

 Key: C.DoNotUseGoto

FIGURE 21 : REGLES DE QUALIMETRIE (7 A 12)

☒ Minor

If statement must use braces
 Avoid using if statements without using curly braces.

 Key: C.IfStatementWithoutBraces

☒ Major

Nested if depth
 Restricts nested if-else blocks to a specified depth.

 maximumNestedIfLevel: The maximum number of nested if that are authorized.
 Key: C.NestedIfDepth

☒ Minor

While and Do/While loops must use braces
 Avoid using While and Do/While loops without using curly braces.

 Key: C.WhileLoopsWithoutBraces

FIGURE 22 : REGLES DE QUALIMETRIE (12 A 15)

c) QCM

Lors du jour de la recette, une épreuve théorique de validation des connaissances acquises sera effectuée sous la forme d'un questionnaire à choix multiple (QCM) d'une durée de 30 min. Ce QCM sera effectué en monôme et en ligne sur l'intranet.

8. ANNEXE : FONCTIONS IMPOSEES POUR LES TESTS AUTOMATIQUES

a) FICHIER DE DEFINITION « DEFINE.H »

```

#ifndef DEFINE_H
#define DEFINE_H

typedef struct{
    float acr; /*!< AC R */
    float dcr; /*!< DC R */
    float acir; /*!< AC IR */
    float dcir; /*!< DC IR */
} absorp;
typedef struct{
    int spo2; /*!< SPO2 */
    int pouls; /*!< Pouls */
} oxy;

#endif

```

b) FIR

Le prototype de la fonction `FIR` n'est pas imposé, ceci dans le but de laisser au libre choix des étudiants les paramètres d'entrées et de sorties du bloc. En revanche, il est demandé de fournir une fonction de test de ce bloc FIR nommée « `firTest` » dont le prototype est imposé :

fir.h

```
absorp firTest(char* str);
```

fir.c

```
absorp firTest(char* str){
    absorp data;
    ...
    return data;
}
```

Cette fonction ouvre et lit les valeurs d'absorption contenues dans le fichier nommé `str` et filtre ces valeurs au moyen de la fonction `FIR`. Lorsque le fichier est fini d'être lu, la fonction retourne la dernière valeur d'absorption filtrée. Attention pour avoir le maximum de points il faut que la fonction `firTest` appelle itérativement la fonction `FIR`.

c) IIR

Le prototype de la fonction `IIR` n'est pas imposé, ceci dans le but de laisser au libre choix des étudiants les paramètres d'entrées et de sorties du bloc. En revanche, il est demandé de fournir une fonction de test de ce bloc IIR nommée « `iirTest` » dont le prototype est imposé :

`iir.h`

```
absorp iirTest(char* str);
```

`iir.c`

```
absorp iirTest(char* str) {  
    absorp data;  
    ...  
    return data;  
}
```

Cette fonction ouvre et lit les valeurs d'absorption contenues dans le fichier nommé `str` et filtre ces valeurs au moyen de la fonction `IIR`. Lorsque le fichier est fini d'être lu, la fonction retourne la dernière valeur d'absorption filtrée. Attention pour avoir le maximum de points il faut que la fonction `iirTest` appelle itérativement la fonction `IIR`.

d) MESURE

Le prototype de la fonction `mesure` n'est pas imposé, ceci dans le but de laisser au libre choix des étudiants les paramètres d'entrées et de sorties du bloc. En revanche, il est demandé de fournir une fonction de test de ce bloc Mesure nommée « `mesureTest` » dont le prototype est imposé :

`mesure.h`

```
oxy mesureTest(char* str);
```

`mesure.c`

```
oxy mesureTest(char* str) {  
    oxy myOxy;  
    ...  
    return myOxy;  
}
```

Cette fonction ouvre et lit les valeurs d'absorption contenues dans le fichier nommé `str` et calcule des valeurs d'oxymétrie au moyen de la fonction `mesure`. Lorsque le fichier est fini d'être lu, la fonction retourne la dernière valeur d'oxymétrie calculée. Attention pour avoir le maximum de points il faut que la fonction `mesureTest` appelle itérativement la fonction `mesure`.

e) AFFICHAGE

affichage.h

```
void affichage(oxy myOxy) ;
```

affichage.c

```
void affichage(oxy myOxy) {
    ...
}
```

Si le verrou est présent, la fonction ne modifie pas le fichier `Data.txt`.

f) INTEGRATION

Pour l'intégration, il est demandé d'écrire une fonction `integrationTest` comme suit :

integration.h

```
void integrationTest(char* str) ;
```

integration.c

```
void integrationTest(char* str) {
    ...
}
```

La fonction `integrationTest` réalise l'intégration de toutes les fonctions précédentes en ouvrant puis lisant les valeurs d'absorptions contenues dans le fichier nommé `str`. Les fonctions `FIR`, `IIR`, `mesure` et `affichage` sont ensuite appelées itérativement jusqu'à que le fichier soit fini d'être lu.

g) LECTURE (SIMULATION USB)

Pour le test de la fonction lecture (simulation de la lecture sur port USB) il est demandé d'écrire une fonction `lecture` comme suit :

lecture.h

```
absorp lecture(FILE* pf, int* etat) ;
```

lecture.c

```
absorp lecture(FILE* file_pf, int* etat) {
    absorp myAbsorp;
    *etat=EOF;

    return myAbsorp;
}
```

La fonction `lecture` lit une mesure d'absorption (ACR, DCR, ACIR et DCIR) selon le format défini dans Tableau 2 dans le fichier ouvert avec le pointeur `pf`. Quand on détecte la fin du fichier, la variable à l'adresse `etat` passe à EOF.

h) CONSEILS

Une archive contenant tous les fichiers demandés et les prototypes associés vous est fournie sur l'intranet sous le nom Minimal.zip (Minimal_CIR3.zip pour les CIR3)

Nous vous conseillons de partir de ces fichiers et de compléter au fur et à mesure les différentes fonctions.

Si vous n'avez pas réussi un bloc, commentez le code qui n'est pas fonctionnel mais laissez le squelette obligatoire de la fonction (coquille vide).

9. ANNEXE : FICHIERS DE TESTS

Des fichiers de test vous sont fournis afin de vous donner la possibilité de tester vos blocs de façon unitaire :

- record1.dat : en sortie de fonction lecture et après recentrage
- record1_fir.dat : en sortie de filtrage FIR
- record1_iir.dat : en sortie de filtrage IIR

La lecture des fichiers est possible en utilisant les fonctions disponibles dans fichiers.h/fichiers.c, dont voici un exemple d'utilisation :

```
absorp myAbsorp;  
int fileState = 0;  
FILE* pf=initFichier("record1.dat");  
while(fileState != EOF){  
    myAbsorp = lireFichier(pf,&fileState);  
}  
finFichier(pf);
```

Pour le fichier, record1_iir.dat, les valeurs calculées de SpO2 et de pouls sont les suivantes :

- SpO2 = 85 %
- Pouls = 69 bpm

10. REFERENCES

- [1] J. Corbel, « Présentation générale du projet A3 : oxymètre de Pouls », ISEN Ouest, Février 2020.
- [2] [« Finite impulse Response filter », Wikipedia, the free encyclopedia](#)
- [3] [« Infinite Impulse Response filter », Wikipedia, the free encyclopedia](#)
- [4] [Driver FTDI chip, Future Technology Devices International Ltd.](#)