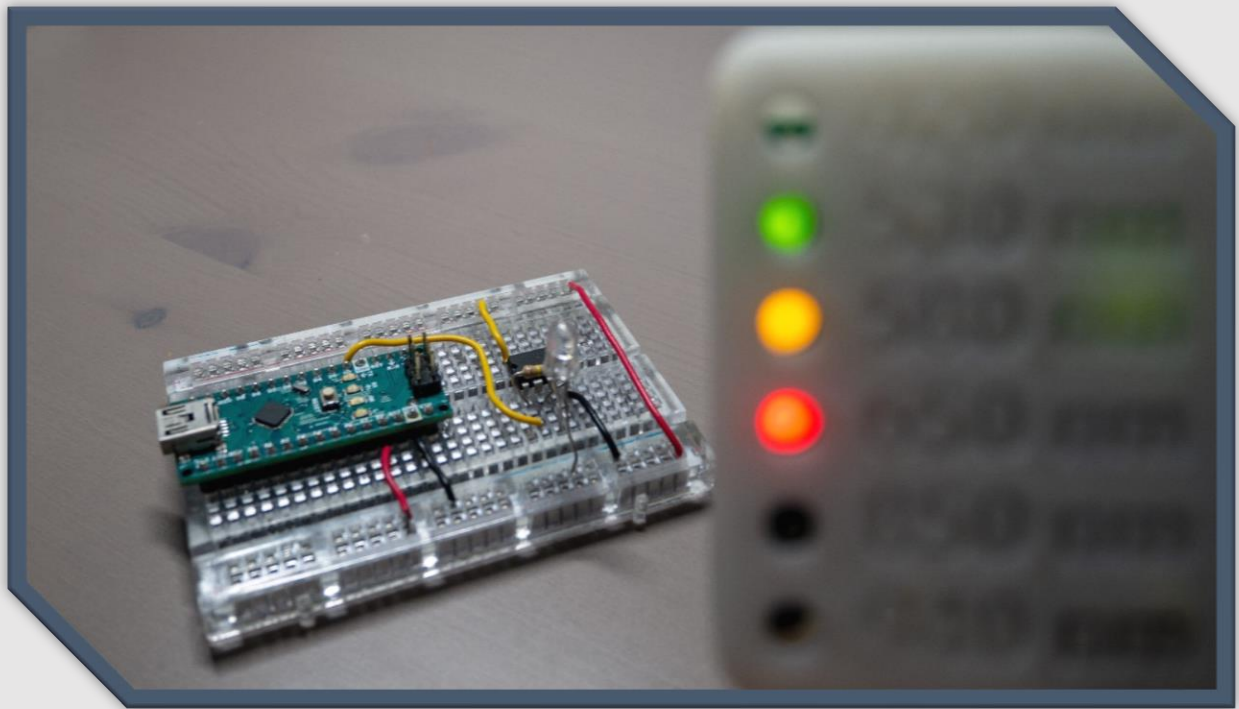


Du capteur à la mesure

ASSIGNMENT 2



Aymard Victor
Ikka Daniel
OCRES Gr.2

Sommaire

INFORMATIONS COMPLEMENTAIRES SUR LES COMPOSANTS	2
LE MICROCONTROLEUR : L'ARDUINO NANO	2
1. Les ports :	2
Ports analogiques :	2
Ports digitaux :	2
RAPPEL DE LA CHAINE DE MESURE	2
TESTS, MISE EN PRATIQUE ET CONCEPTION DU MONTAGE	3
BATTERIE DE TESTS	3
2. Intensité du courant généré par l'excitation de la photodiode	3
3. Choix de la résistance pour l'amplification	3
4. Phase de tests	4
CONCEPTION DE LA SOLUTION FINALE	7
BRANCHEMENTS	7
ANALYSE DU SIGNAL REÇU	8
DU SIGNAL AU MESSAGE FINAL	9
PHASE DE TEST	9
SOLUTION CHOISIE : MASQUE BINAIRE	10
CODE FINAL	11
1. Explication du code	12
DECODAGE DU MESSAGE	13

Table des Illustrations

Figure 1 : Schéma de la chaine de mesure actuelle	2
Figure 2 Schéma explicatif d'une amplification	3
Figure 3 : Montage type transimpédance	4
Figure 4 schéma du montage	7
Figure 6 affichage des différentes durées de signal up	9
Figure 7 constantes et setup du programme	11
Figure 8 fonction loop du programme	11
Figure 9 autres fonctions	12
Figure 10 affichage du message en décimal	13
Figure 11 table ASCII	13
Figure 12 affichage du message final	14
Photo 1 : Émetteur IR de l'iPhone X Photo 2 : envoi du message IR sur la photodiode....	5
Photo 3 : visualisation de la réception d'un signal en provenance de FaceID de l'iPone X ..	5
Photo 4 : Réception du message espacé de 200ms	6
Photo 5 : Zoom sur un octet	6
Photo 6 photographie du montage	7

Informations complémentaires sur les composants

Le microcontrôleur : L'Arduino nano

D'après la datasheet de celui-ci, la clock speed est de 16MHz. On pourra donc différencier les différentes fréquences de 3KHz et 5KHz sans problème.

1. Les ports :

Durant nos recherches, nous nous sommes questionnés pour savoir quel type de port serait le plus adéquat pour notre problème, à savoir, un port analogique, ou un port digital.

Ports analogiques :

Un port analogique comprend un convertisseur analogique numérique. Comme nous avons un signal analogique, cela nous permet d'avoir directement une retranscription fidèle du signal sans avoir recours à un CAN supplémentaire dans le montage. Ces ports ont une résolution de 10 bits (à savoir 1024 valeurs possibles).

Ports digitaux :

Ils peuvent recevoir un max de 40mA. Or, nous avons une intensité d'environ 10mA. Ainsi, nous pourrions utiliser un port digital sans problème. L'intérêt du port digital selon nous est qu'il servira de circuit comparateur, envoyant uniquement des 0 et des 1. Il devrait pouvoir transmettre un 1 lorsque la tension atteindra 5V et un 0 pour une tension faible (environ 200mV dans notre cas).

Rappel de la chaine de mesure

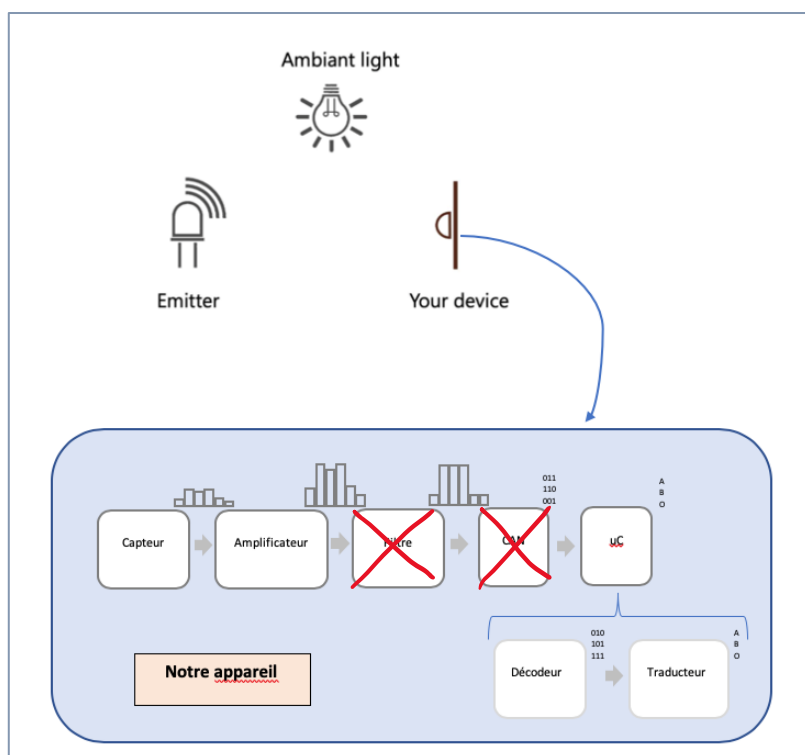


Figure 1 : Schéma de la chaîne de mesure actuelle

Contrairement à ce à quoi nous avons pensé précédemment, nous pensons à présent ne pas avoir besoin de filtre (car les différences de tensions sont notables : 200mV contre 5V), ni de convertisseur analogique numérique, pour les mêmes raisons.

Nous sommes en possession de la photodiode SFH213. Cette dernière est sensible sur une plage relativement importante de longueur d'onde (400nm à 1100nm). Cette sensibilité est toutefois notablement que dans la partie infrarouge avec un pique à 850nm. Ce capteur génère en moyenne une intensité électrique d'environ 100μA (en fonction de l'intensité lumineuse ainsi que de la longueur d'onde).

L'idée est d'utiliser un montage transimpédance, à la fois pour convertir le courant généré par l'excitation de la photodiode et pour par la même occasion amplifier la tension de sortie.

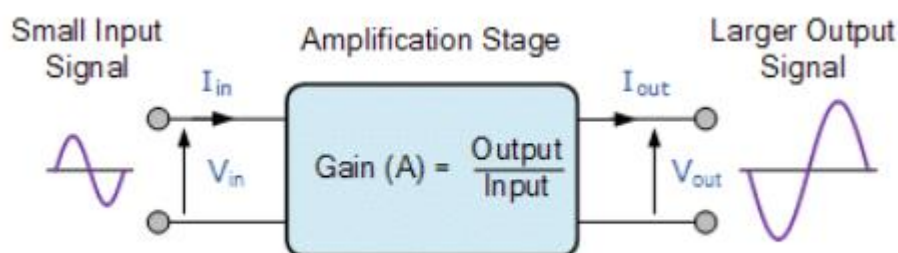


Figure 2 Schéma explicatif d'une amplification

Par la suite nous pensions avoir besoin de filtrer les basses tensions, relatives au courant généré par le spectre du visible, mais nous pensons finalement pouvoir nous en passer (car les différences de tensions sont notables : 200mV contre 5V). Il en est de même pour le convertisseur analogique numérique.

Tests, mise en pratique et conception du montage

Batterie de tests

2. Intensité du courant généré par l'excitation de la photodiode

Nous avons tout d'abord vérifié l'intensité du courant généré par la lumière ambiante. Cette dernière est de l'ordre de la dizaine de micro-ampères. Ensuite sous l'excitation du signal lumineux infrarouge nous atteignons facilement en fonction de la direction et de la distance du faisceau des intensités de l'ordre de 100 μA.

3. Choix de la résistance pour l'amplification

Les intensités étant très petites nous avons choisi d'utiliser de façon arbitraire une « grosse » résistance, 470 kOhms. D'après le calcul suivant :

$$V_s = -I_e \cdot R_f$$

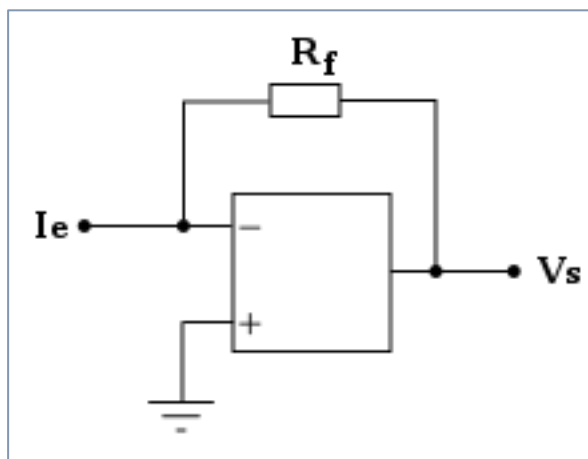


Figure 3 : Montage type transimpédance

Ainsi, l'amplification serait d'un facteur 5.10^5 nous permettant de travailler avec des valeurs de l'ordre du Volt.

Cette résistance élevée nous permet d'avoir un signal presque « carré », là où une résistance trop faible nous aurait donné un signal en dents de scie.

4. Phase de tests

a. Sondage de la lumière ambiante

Au cours de nos différents tests, nous avons tout d'abord essayé de capter la tension de sortie du uniquement à la lumière ambiante. Les valeurs observées sur l'oscilloscope tournaient en fonction de la salle entre 1V et 2V, ainsi totalement cohérents avec la partie théorique.

b. Test sans l'émetteur adéquat

Les tests suivants ont ensuite été réalisés avec des variations d'intensité lumineuse à l'aide d'un flash de téléphone (lumière visible). Bien que beaucoup moins sensible à ces longueurs d'onde l'intensité lumineuse et la forte amplification suffisent rapidement à saturer la tension de sortie à 5V.

L'idée nous est venue de tester l'éventuelle réception d'un message infrarouge avec une télécommande, mais en vain, puis avec l'émetteur IR d'un iPhone X (permettant le fonctionnement de FaceID).



Photo 1 : Émetteur IR de l'iPhone X

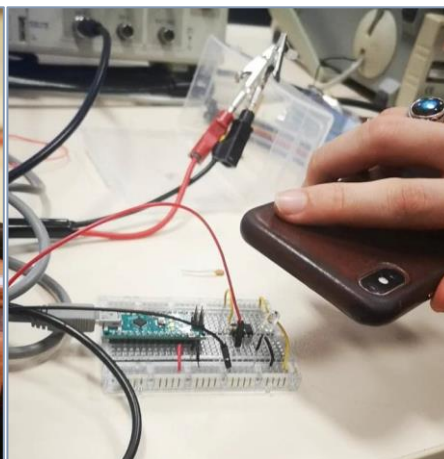


Photo 2 : envoi du message IR sur la photodiode

À notre grande surprise, nous avons eu des résultats notables à l'oscilloscope.

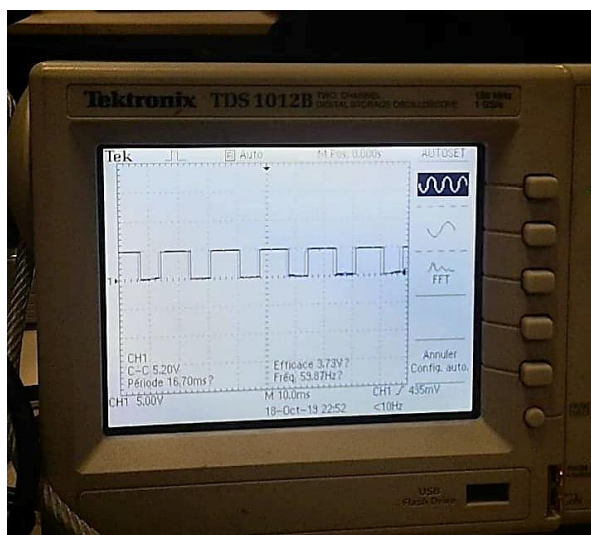


Photo 3 : visualisation de la réception d'un signal en provenance de FaceID de l'iPhone X

Nous observons deux choses, la première étant que le message perçu est bien distinct perceptible. La sensibilité au IR est nettement remarquable. Enfin, les variations sont à présent entre 5V (saturation de l'AOP dut à la forte amplification imposée par la grosse résistance) et 0-300mV. Nous avons déduit que sous l'excitation d'une longueur d'onde, à laquelle le capteur avait une forte sensibilité, ce dernier se trouvait comme « ébloui » et pendant un court laps de temps (comme nous pouvons le voir sur l'oscilloscope, les rayons sont envoyés à intervalle régulier de 10ms) la lumière ambiante devient négligeable totalement ou quasi totalement.

c. Tests avec l'émetteur adéquat

À présent munis de l'émetteur du signal à récupérer nous observons les mêmes interprétations de saturation à 5V et « d'éblouissement ».

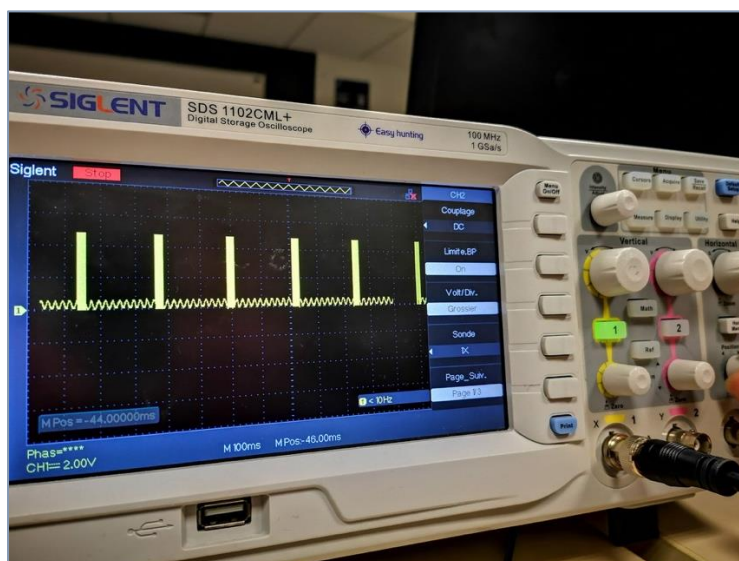


Photo 4 : Réception du message espacé de 200ms

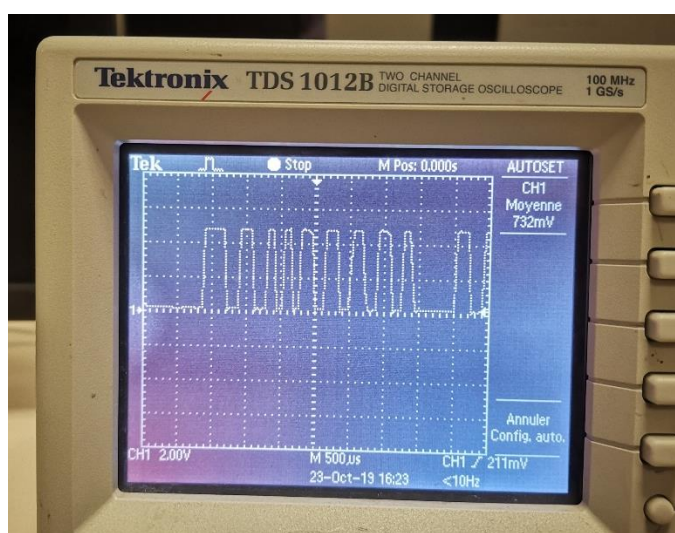


Photo 5 : Zoom sur un octet

Voici donc des illustrations bien encourageantes avec un discernement de l'émission des messages, espacé chacun d'un blanc de 200ms (deux div au balayage horizontal 100ms). Enfin sur lorsque l'on zoom sur le message nous interprétons tout aussi bien les bit 0 et 1 ainsi que le bit de start et l'espacement de 0.5ms séparant deux bits.

On arrive ici à distinguer 8bits précédés du signal de début de transmission. De plus, en zoomant davantage, on peut lire le premier octet : 10011110. Il ne nous reste donc plus qu'à transmettre ces informations à l'Arduino pour qu'il analyse et décode le message.

Conception de la solution finale

Branchements

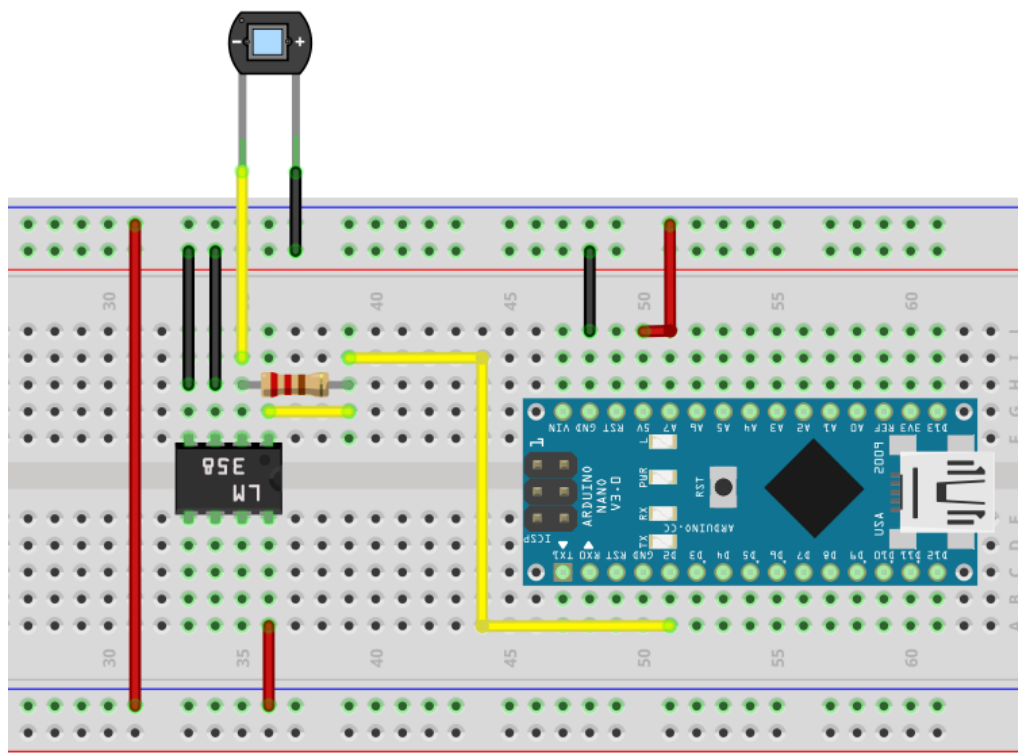


Figure 4 schéma du montage

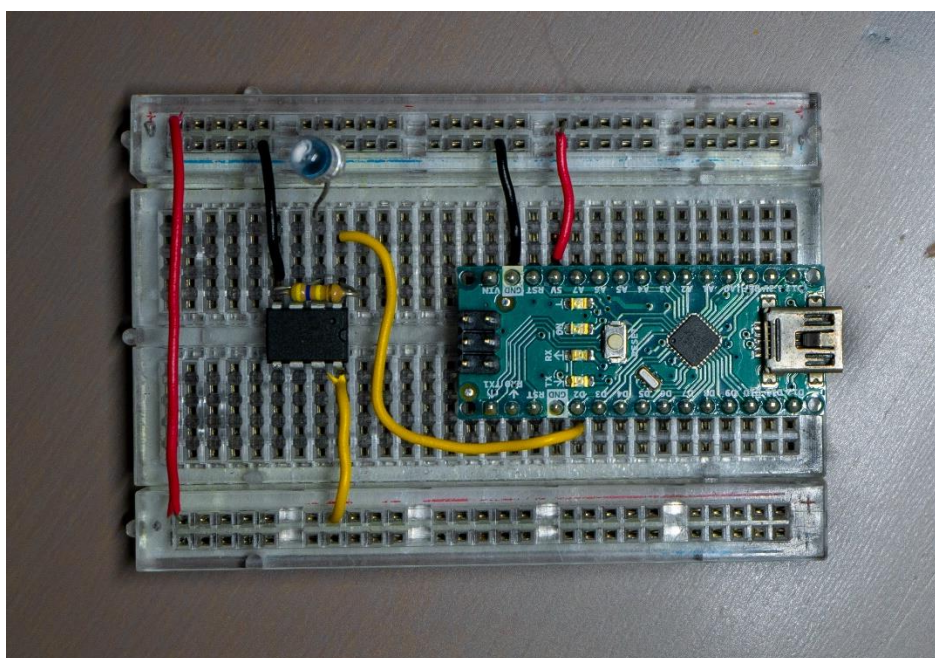


Photo 6 photographie du montage

Du fait de la saturation à 5V et du phénomène « d'éblouissement » constaté, il ne nous paraît pas nécessaire de mettre en œuvre une solution de « filtre » de tension. Si cette dernière est à ajouter, nous avons pensé à l'utilisation du second AOP présent dans le composant MCP6242 (*LM 356 sur le schéma, non fidèle à 100%*). Ce dernier serait utilisé comme comparateur de tension sortant alors 0V ou 5V.

Analyse du signal reçu

N'ayant pas eu accès à l'émetteur assez longtemps pour pousser la phase de développement aussi loin, cette partie restera théorique. Le signal sera donc directement perçu et envoyé sur un PIN digital de l'Arduino. Ce dernier est déclenché en théorie à 5V, mais dans la pratique, cette valeur est à revoir à la baisse autour de 2.7V ou 3V. Par la suite des fonctions déjà implémentées devraient retourner les différentes fréquences perçues. Encore une fois cette dernière partie reste à murir et demande réflexion et documentation plus poussée.

Question éventuelle à élucider au prochain cours :

- Avons-nous réellement besoin d'un comparateur de tension où le Pin digital de l'Arduino fera office tout simplement de « comparateur ».

Du signal au message final

Pour cette dernière partie, nous nous sommes intéressés à la partie décodage du projet, et donc à l'Arduino.

Le masque binaire

Phase de test

En observant ce que notre Arduino affichait lorsque nous en approchions l'émetteur, nous nous sommes rendu compte que trois groupes de valeurs se distinguaient, tout comme l'énoncé le précisait (avec une légère différence comparée aux valeurs théoriques). La première valeur : le bit de start avait une durée d'environ 320 ms. Pour les valeurs représentant les 0 et les 1, nous avons un intervalle de valeur avec environ 20-35% la valeur théorique de 1 et de 0. Pour être certain de ne pas perdre une valeur à cause d'un dysfonctionnement du capteur, nous avons donc séparé les valeurs entrantes par rapport à la durée de 150 ms. Ainsi, pour une valeur d'une durée de supérieure à 150ms nous lui attribuons la valeur 0 alors qu'un front montant inférieur à 150ms, correspond à un 1.

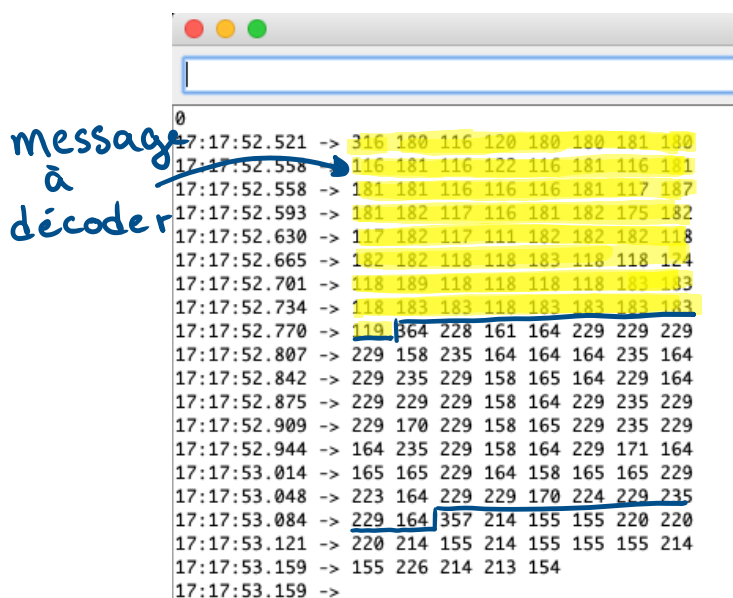
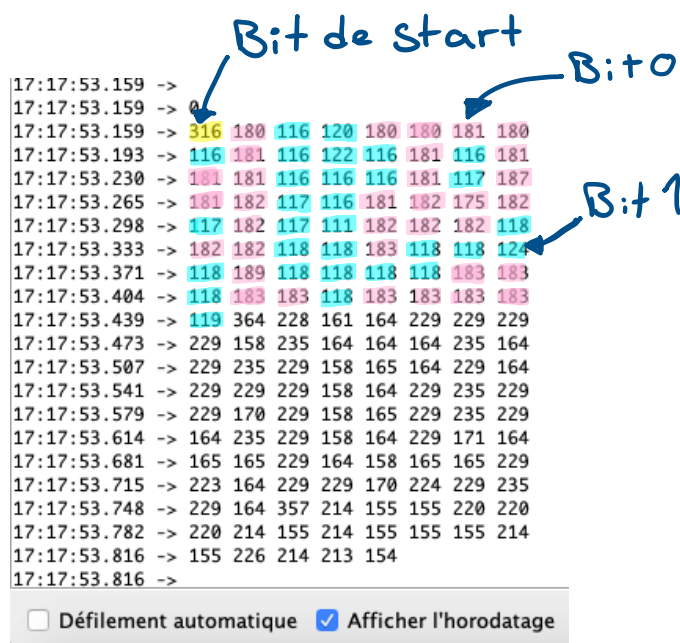


Figure 5 affichage des différentes durées de signal up



Solution choisie : Masque binaire

Afin d'optimiser au maximum les ressources de notre Arduino nous avons fait le choix du masque binaire sans mémoire tampon. Nous traitons donc directement l'information entrante plutôt que de la stocker dans un tableau tampon et d'analyser ensuite les valeurs.

Chaque valeur est alors rangée dans un char (ayant une taille d'un byte, soit 8 bits).

Connaissant la longueur du message à recevoir (64 bits), nous avons donc décidé d'utiliser un tableau de 8 char (portant le nom de message dans notre code). Ces chars sont initialisés par défaut à 0 (0b0000.0000). Lorsque l'on perçoit la durée correspondant à un 0, on décrémente notre rang de bit, ainsi aucune opération binaire n'est effectuée, mais le bit est bien pris en compte. Lorsqu'on a un 1 (qui est mesuré au front montant), on va le placer dans notre char de 8 bits. Pour ce faire, on effectue l'opération binaire OU $1 \ll n$ (où n est le rang du bit à traiter). L'opération « OU \Rightarrow » permet de conserver les bits qui ont déjà été mis à 1, tout en mettant notre nouveau bit de rang n à 1.

Par exemple, pour un bit de 1 présent à la troisième position, on va se décaler de 3 bits avant de procéder au OU $=$.

0 1 1 0 0 0 0 0



Décalage de 3 bits (avec le $1 \ll 3$)

On fait le $|=$ et on place notre 1

0 1 1 0 0 0 1 0 0

Une fois passé 8 fois dans la boucle de remplissage de notre char, on passe au char suivant, jusqu'à avoir parcouru notre tableau de 8 bytes.

Code final

```

const int digital_pin = 2;

const int one_direction = 300/2;      // in microsecondes 50% pwm
const int start_duration = 500*0.6;  // in microsecondes 60% pwm 500us

unsigned long duration = 0;

bool hasStarted = false;

//initialize all bytes to 0;
char message[8] = {0,0,0,0,0,0,0,0};
short int bit_rank = 7;
short int byte_rank = 0;

void setup() {
    Serial.begin(9600);
    pinMode(digital_pin, INPUT);
}

```

Figure 6 constantes et setup du programme

```

void loop() {
    // Get the duration of next high level
    duration = pulseIn(digital_pin, HIGH);

    if(hasStarted) {
        // it's one // one_direction = 150
        if (duration < one_direction) {
            message[byte_rank] |= (1<<bit_rank);
        }

        // Even 0 or 1
        bit_rank--;

        if(bit_rank < 0) {
            bit_rank = 7;
            byte_rank++;
        }

        if(byte_rank > 7) {
            restart(); // and display
        }
    } else {
        hasStarted = mustStarted();
    }
}

```

Figure 7 fonction loop du programme

```

bool mustStarted() {
    return ((duration > start_duration*0.9) && (duration < start_duration*1.1));
}

void restart() {
    byte_rank = 0;
    hasStarted = false;
    display();
    for(int i=0; i<8 ; i++) {
        message[i] = 0;
    }
}

void display() {
    for(int i=0 ; i<8 ; i++) {
        Serial.print(message[i]);
    }
    Serial.println("");
}

```

Figure 8 autres fonctions

1. Explication du code

À l'entrée de la fonction `loop()`, on regarde si le message est en train d'être lu. Si oui, la variable `hasStarted` vaut `true` ; dans l'autre cas, on appelle la fonction `mustStarted` qui va chercher le bit de start, et renvoi `true` dès qu'elle le trouve.

Par la suite, si on a une durée correspondant à un 1, on va l'inclure dans le char qui est étudié, puis on décrémente le `byte_rank`. Le char étant par défaut à 0, il ne nous est pas nécessaire de faire une opération lorsque l'on détecte un 0. On va ensuite finir de compléter notre byte pour passer au suivant.

Lorsque notre tableau de 8 char est rempli, on appelle la fonction `restart()` qui va remettre les variables à leur état d'origine et afficher notre message avec la fonction `display()`.

Décodage du message

Pour ce qui est du décryptage, celui-ci se fait de manière immédiate étant donné que les données ont été rentrées sous forme de char.

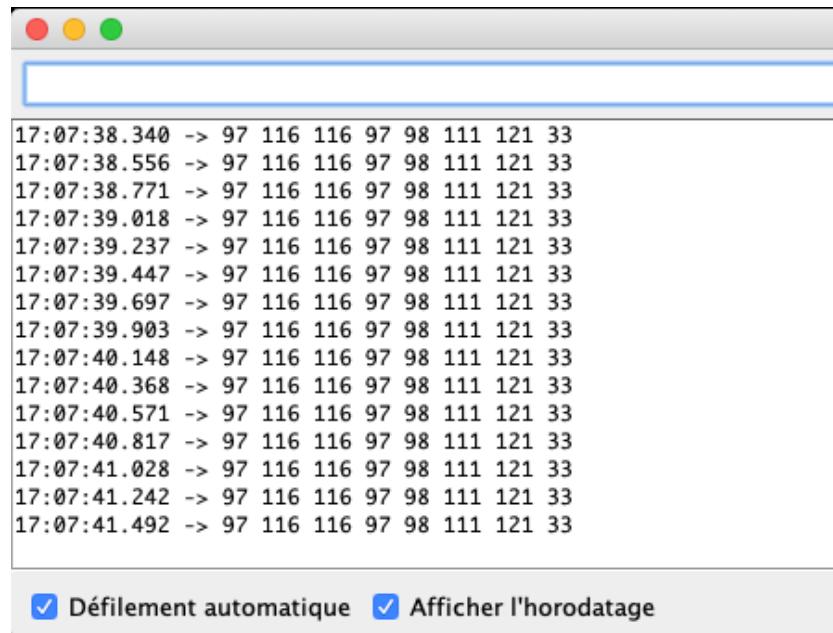


Figure 9 affichage du message en décimal

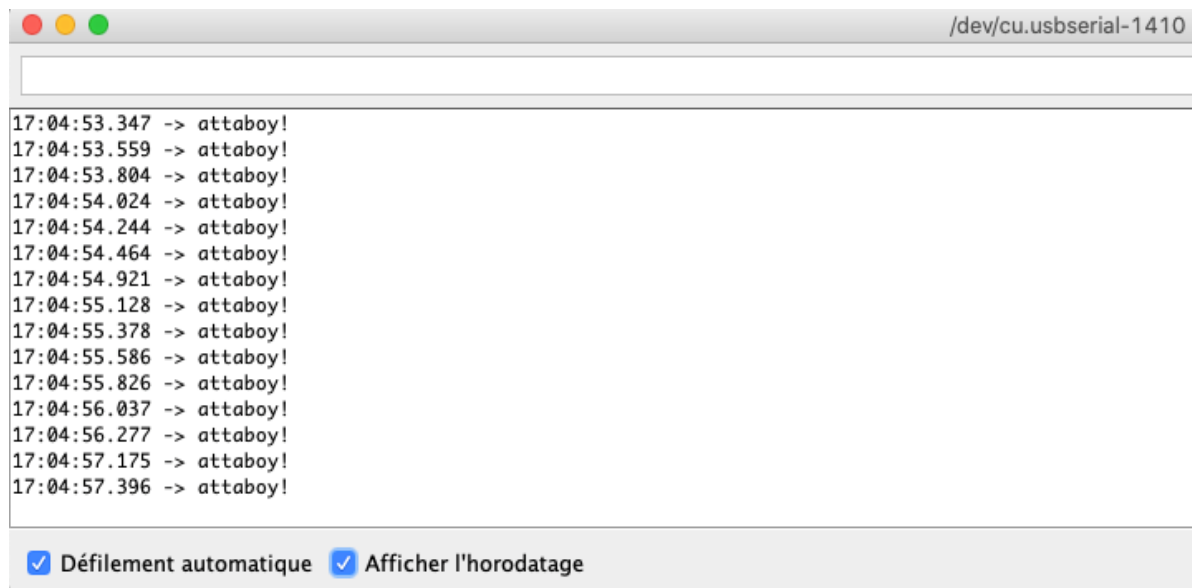
La correspondance avec la table ascii nous permet donc de décoder très simplement le message.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Figure 10 table ASCII

Et ainsi, nous avons pu découvrir ce mystérieux message qui nous a tant intrigués depuis le début de ce semestre : attaboy!



The image shows a terminal window with a title bar containing three colored buttons (red, yellow, green) and the text "/dev/cu.usbserial-1410". The terminal displays a list of messages, each preceded by a timestamp and a right-pointing arrow. The messages are all "attaboy!". At the bottom of the window, there are two checkboxes, both of which are checked: "Défilement automatique" and "Afficher l'horodatage".

```
17:04:53.347 -> attaboy!  
17:04:53.559 -> attaboy!  
17:04:53.804 -> attaboy!  
17:04:54.024 -> attaboy!  
17:04:54.244 -> attaboy!  
17:04:54.464 -> attaboy!  
17:04:54.921 -> attaboy!  
17:04:55.128 -> attaboy!  
17:04:55.378 -> attaboy!  
17:04:55.586 -> attaboy!  
17:04:55.826 -> attaboy!  
17:04:56.037 -> attaboy!  
17:04:56.277 -> attaboy!  
17:04:57.175 -> attaboy!  
17:04:57.396 -> attaboy!
```

☒ Défilement automatique ☒ Afficher l'horodatage

Figure 11 affichage du message final