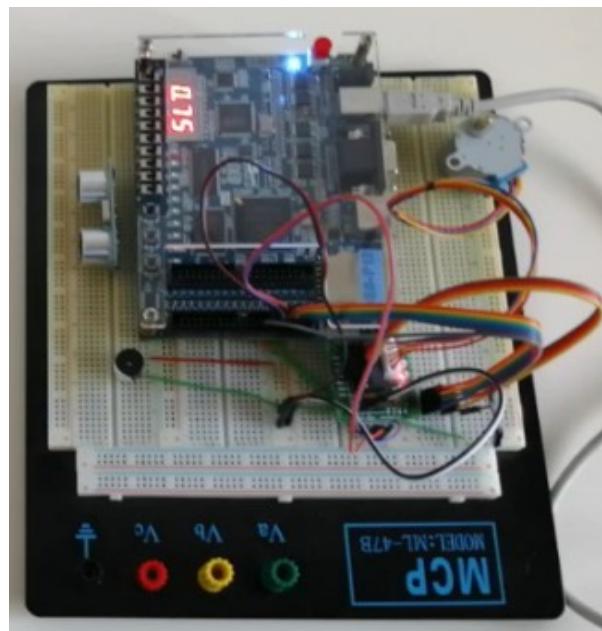


FPGA

mini-projet



Réalise par :

NIYIKIZA Rubagumya Arsene

Supervise par :

Isabelle Lajoie

Table des matières

. Mise en œuvre d'une électronique autour d'un FPGA capable de piloter un moteur pas à pas en fonction de la distance mesurée par un capteur ultrasonore.....	3
. Partie I : Capteur Ultrason.....	4
1. Génération du signal Trigger.....	5
2. Acquisition et traitement du signal Echo.....	6
3. Affichage de la distance.....	7
. Partie II : moteur pas à pas & Interfaçage du capteur US avec moteur pas à pas.....	8
. Partie III : Intégration du Buzzer.....	9
. Configuration des Pins.....	12
. Application et utilités.....	13
. Conclusion.....	14
. Sources et quelques liens.....	15

. Mise en œuvre d'une électronique autour d'un FPGA capable de piloter un moteur pas à pas en fonction de la distance mesurée par un capteur ultrasonore

Dans ce mini-projet on a pour objectif d'utiliser un capteur Ultrason HC-SR04 pour mesurer des distances et pouvoir contrôler un moteur pas à pas à partir des distance détecte. À la fin intégré un buzzer active pour signaler le niveau de proximité.

En m'inspirant d'une source internet, je commence par configurer le capteur, après le moteur pas à pas, à la suite j'intègre le moteur dans le circuit contenant le capteur pour pouvoir manipuler le moteur selon l'instruction issue du capteur et à la fin j'ai ajouter un buzzer qui donne un signal sonore à différentes fréquences et selon la proximité des objets.

Je me suis inspirée des systèmes de détection de proximité automobile Figure 1, pour améliorer et donner plus d'application dans la vie courante de mon projet.

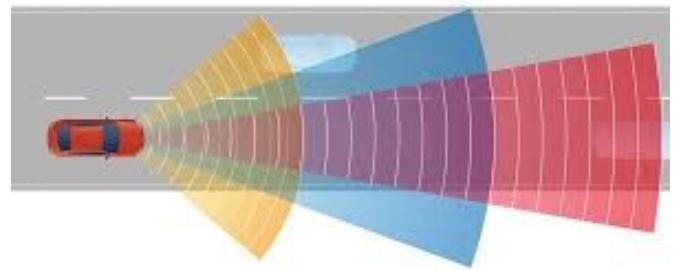


Figure 1: Exemple d'un système de détection de proximité automobile

L'ensemble de mon projet, a été effectué dans 3 étapes principaux :

Partie I : Capteur Ultrason

Partie II : moteur pas à pas & Interfaçage du capteur US avec moteur pas à pas

Partie III : Intégration du Buzzer

. Partie I : Capteur Ultrason

HC-SR04 est un module de capteur ultrason (Figure 2) avec qui permet de mesure la distance entrée un objet (capteur) et un obstacle.

Il a une portée distance allant de 2 à 400 cm sans contact avec une précision de télémétrie pouvant atteindre 3 mm.

Ce module est compose des émetteurs ultrasoniques, un récepteur, un circuit de commande et 4 broches des branchements :



Figure 2: Capteur HC-SR04

- 2 pins d'alimentation **3.3V/5V** et **Gnd** (dans notre on utilisera 5v fournie la carte FPGA)
- Pin **Trigger** : un signal de $10\mu s$ à l'état haut entrant dans le capteur
- Pin **Echo** : signal fournie par le capteur, lequel nous permet de déterminer la distance

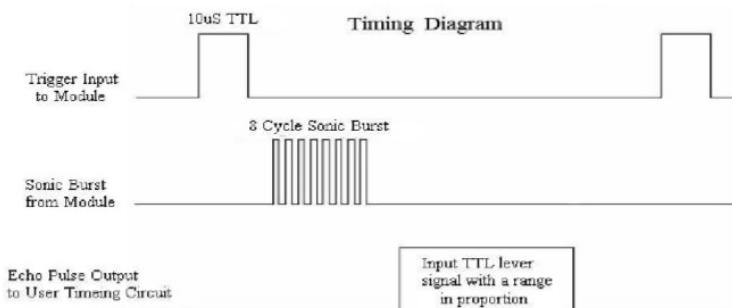


Figure 3: Signaux traitent dans le capteur

Le principe de fonctionnement :

1. On fournit un signal **Trigger** de $10\mu s$ à l'état haut au module comme illustre sur Figure 3 (*Trigger Input to module*)
2. Le module envoie automatiquement huit 40 kHz et détecte s'il y a un signal d'impulsion en retour (*Sonic Burst from Module*).
3. On obtient un signal de retour SI, à travers un niveau élevé, lequel est envoyé à travers la pin **Echo**. Le temps de la durée d'E / S de sortie élevée est le temps entre l'envoi des ultrasons et le retour. Et on calcule la distance = (temps de niveau élevé \times vitesse du son (340M / S) / 2

1. Génération du signal Trigger

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4     --declaration des variables, entrées et sorties
5 entity Trig_gen is
6     port ( clk : in std_logic;
7             Trigger : out std_logic);
8
9 end Trig_gen;
10
11 architecture Behavioral of Trig_gen is
12 component Counter is
13
14     generic ( n: POSITIVE := 10);
15     port (clk : in std_logic;
16             enable : in std_logic;
17             reset : in std_logic;          --Active low
18             Counter_output : out std_logic_vector(n-1 downto 0));
19
20 end component;
21
22 signal resetCounter : std_logic;
23 signal outputCounter : std_logic_vector(23 downto 0);
24
25 begin
26     trigg : Counter generic map (24) port map (clk, '1', resetCounter, outputCounter);
27     process(clk)
28
29 constant ms250: std_logic_vector(23 downto 0) := "101111101011110000100000"; --to generate 250ms
30 constant ms250and100us: std_logic_vector(23 downto 0) := "1011111011000000000001000";-- to generat
31 begin
32     if( outputCounter > ms250 and outputCounter < ms250and100us) then
33         Trigger <= '1';
34     else
35         Trigger <= '0';
36     end if;
37     if(outputCounter = ms250and100us or outputCounter = "XXXXXXXXXXXXXXXXXXXXXX") then
38         resetCounter <= '0';
39     else
40         resetCounter <= '1';
41     end if;
42 end process;
43 end Behavioral;
```

Figure 4: Code VHDL pour générer le signal Trigger

Figure 4 montre le code VHDL qui permet de générer le signal Trigger appeler **Trig_gen.vhd** dans le projet. Avec un signal **clk** (horloge) en entrée et **Trigger** en sortie comme on peut le voir sur Figure 5.

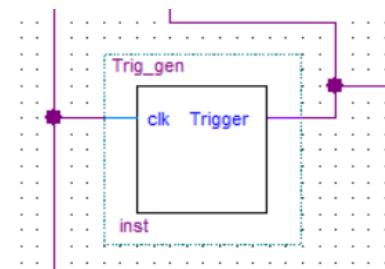
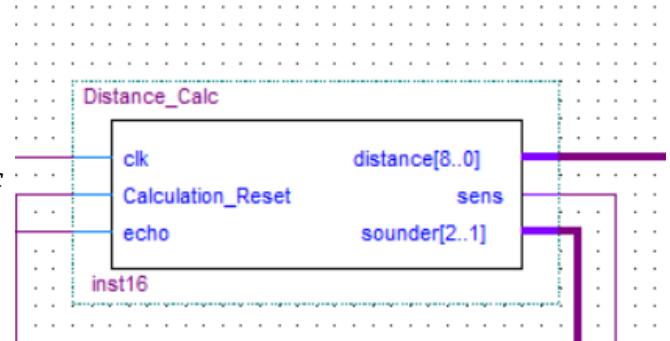


Figure 5: Symbole Trigger

2. Acquisition et traitement du signal Echo

On traite le signal Echo provenant du capteur qui est sous d'un échelon avec la distance équivalent à la durée de l'état haut du signal (**Input TTL** sur Figure 3).



Distance_calc représenter le projet pour le traitement de l'echo Figure 6. Dans **Distance_calc** j'acquis l'echo et calculer la distance représenter en sortie par **distance[8..0]** (sur Figure 6) un signal 9 bits.

Figure 6: Symbole de *Distance_calc*

En entrée et sortie de **Distance_calc**, on a :

- **clk** : horloge
- **Calculation_Reset** : signal Trigger, permettant de savoir quand commencer et terminer le calcul de la distance. En d'autre terme pour réinitialiser en fonction du Trigger
- **echo** provenant du capteur
- **distance[8..0]** : signal de 9 bits générer à la fin du traitement de l'echo
- **sens** : un signal permettant de commander le moteur pas à pas
- **sounder[2..1]** : un signal 2 bits permettant de commande le buzzer

```
35      variable Result : integer;
36      variable Multiplier : STD_LOGIC_VECTOR(23 downto 0);
37
38 begin
39     if(echo = '0') then
40         multiplier := echo_width * "11";
41         Result := to_integer(unsigned(Multiplier(23 downto 13)));
42
43
44         --sonor ranger
45         if(Result > 40) then sounder <="11";      --distance au dela de 40cm
46
47         else if(Result <40 and Result >30 ) then sounder <="10"; -- distance entre 30 et 40cm
48
49         else if(Result <30 and Result >10 ) then sounder <="01";           --distance entre 10 et 30cm
50
51             else if(Result <10) then sounder <="00"; --distance en dessus de 10cm
52             end if;
53         end if;
54     end if;
55
56         --calcul de la distance minimale
57         if(Result < 10) then sens <='1'; --on dit que la distance minimale est de 10cm
58         else sens <='0';
59     end if;
60
61         if(Result > 458) then
62             distance <= "111111111";
63
64             else
65                 distance <= STD_LOGIC_VECTOR(to_unsigned(Result,9));
66
67             end if;
68         end if;
69     end process Distance_calculation;
70 end Behavioral;
```

Figure 7: Extrait de code VHDL de *Distance_calc*

Fonctionnement du code Figure 7 :

- ligne 41 : on calcule la distance et on la déclare dans le variable **Result** en cm
- ligne 45 - 55 : la partie qui fournie le signal 2 bits **sounder[2..1]**
- ligne 57 – 59 : la partie qui fournie le signal **sens**. Avec **Result** calcule auparavant, je force sens à 1 quand la distance est < 10 cm et à 0 autrement.
- Ligne 61 – 67 : on met la distance calcule sous forme d'un signal 9 bits

3. Affichage de la distance

Pour affiche la distance, j'ai utilisé un bloc **BCD_Converter** qui converti le signal de son entrée en unités, dizaines et centaines et 3 afficheur 7segments **affichage** et aussi avec un point suivant le centaine Figure 8.

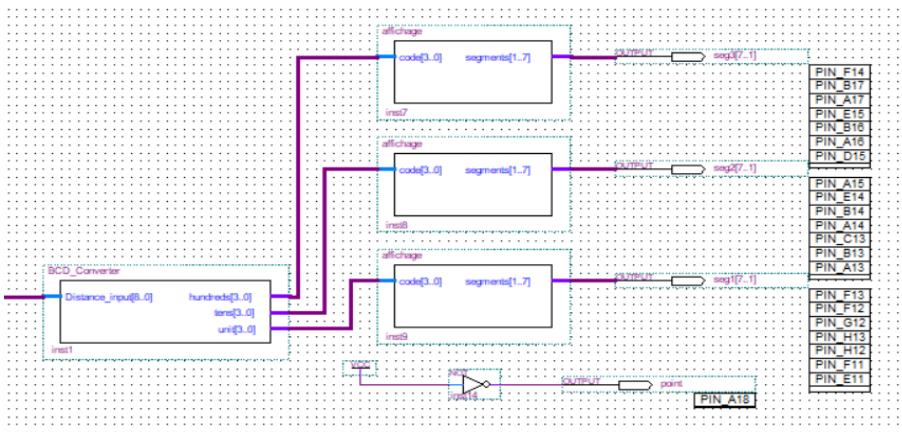


Figure 8: Affichage de la distance

```

10  architecture Behavioral of BCD_Converter is
11
12  begin
13      process(Distance_input)
14          variable i : integer := 0;
15          variable bcd : std_logic_vector(20 downto 0);
16
17
18      begin
19          --Initialisation phase: the space for the BCD representation is reserved
20          bcd := (others => '0');
21          bcd(8 downto 0) := Distance_input;
22          --"Double-Dabble Algorithm" is used to convert the Distance in binary into BCD
23
24      for i in 0 to 8 loop
25          --Left shift one bit
26          bcd(19 downto 0) := bcd(18 downto 0) & '0';
27          --check phase
28          if(i < 8 and bcd(12 downto 9) > "0100") then
29              bcd(12 downto 9) := bcd(12 downto 9) + "0011";
30          end if;
31          if(i < 8 and bcd(16 downto 13) > "0100") then
32              bcd(16 downto 13) := bcd(16 downto 13) + "0011";
33          end if;
34          if(i < 8 and bcd(20 downto 17) > "0100") then
35              bcd(20 downto 17) := bcd(20 downto 17) + "0011";
36          end if;
37
38      end loop;
39
40      --The three BCD numbers are put in output
41      hundreds <= bcd(20 downto 17);
42      tens <= bcd(16 downto 13);
43      unit <= bcd(12 downto 9);
44
45      end process;
46
47  end Behavioral;

```

Figure 9: Extrait du code VHDL du BCD_Converter

À note que pour le point j'ai utilisé un porte logique **NOT** comme les Leds de point sur le 7Segments sont branche en inverse, Figure 8.

. Partie II : moteur pas à pas & Interfaçage du capteur US avec moteur pas à pas

Appliquer dans les petits robots, modélisation ferroviaire, machines pour citer un peu, le **moteur pas à pas** Figure 10 est un moteur à 4 phases ou états, commander par un module faisant varier les états pour faire tourner le moteur. Dans notre cas un module **ULN2003**.

Le module ULN2003 est un circuit compose des plusieurs composants électronique et un circuit intégré. Sur ce module on retrouve :

- 2 pins d'alimentation **5V** et **Gnd**
- 4 pins IN1, IN2, IN3 et IN4 représentant les 4 états du moteur.
- Un socket le branchement du moteur



Figure 10: Moteur pas à pas et le module ULN2003

Le moteur est compose de 4 bobines, place de façon a ce que une rotation correspond à la mise de 2 bobines et l'arrêt de 2 autres restant, comme on peut le voir sur Figure 11. Il y a 5 files, un **5V** et les 4 autres sont des masses avec chacune correspondant à chaque bobine.

Dans notre cas, j'ai utilise un machine à état pour faire varier les différents états et faire tourne le moteur.

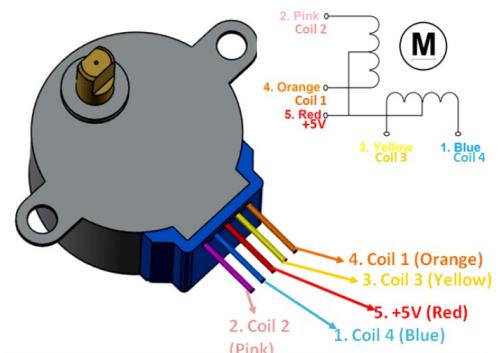


Figure 11: Schéma structureur du moteur pas à pas

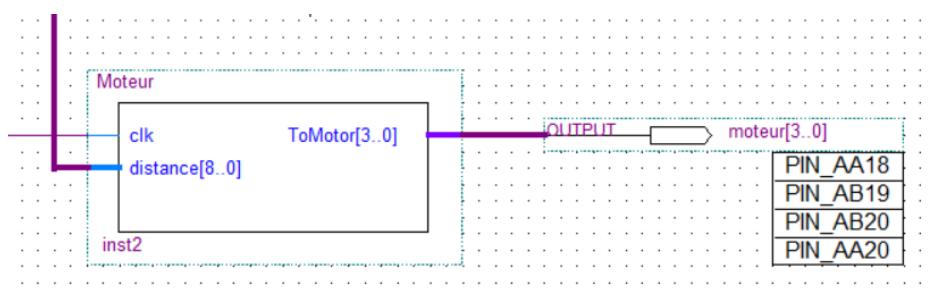


Figure 12: Schéma bloc du moteur

```

10
11  architecture Behavioral of Moteur is
12      type type_etat is (E1, E2, E3, E4);
13      signal x:type_etat;
14      signal sens : std_logic;
15
16  begin
17      process (clk)
18      begin
19
20          if clk'event and clk='1' then
21
22              if(distance >"000001111") then sens <= '1';
23              else sens <= '0';
24          end if;
25
26          case x is
27              when E1=> if sens = '1' then x<=E2;
28                  else x<=E4;
29              end if;
30              when E2=>
31                  if sens = '1' then x<=E3;
32                  else x<=E1;
33              end if;
34              when E3=>
35                  if sens = '1' then x<=E4;
36                  else x<=E2;
37              end if;
38              when others=> x<=E1;
39
40          end case;
41      end if;
42  end process;
43
44      with x select
45          ToMotor <=  "1001" when E1,
46                      "1010" when E2,
47                      "0110" when E3,
48                      "0101" when E4;
49  end Behavioral;

```

Figure 13: Extrait du code VHDL du moteur

Le moteur est représenté par **Moteur.vhd**

Partie III : Intégration du Buzzer

Pour le choix du Buzzer, j'ai utilisé un buzzer active Figure 14 au lieu d'un buzzer passive. C'est du au fait que le buzzer active n'a besoin seulement que d'une source de tension et puis il génère le son lui même, tandis pour le passive il faut lui fournir le signal sonore.

Sur ce projet j'ai du fournir des petits signaux sonore pour être amplifier d'avantage par le buzzer.



Figure 14: Buzzer active

Buzzer_sounder représter le symbole du code VHDL dans le projet faisant fonctionne le buzzer.

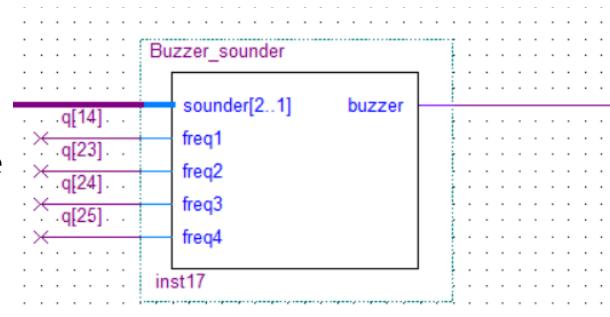


Figure 15: Schéma bloc du buzzer

En entrée et sortie du **Buzzer_sounder**, on a :

- **sounder[2..1]** : un signal 2 bits fournie par **Distance_calc Figure 7**
- **q[14], q[23], q[24] et q[25]** : 4 fréquences 3KHz, 23Hz, 3Hz et 1Hz
- **buzzer** : sortie du signal sonore

L'intégration du buzzer permet de générer des sons selon la distance de proximité. Avec un son trop aiguë (moins favorable à une haute fréquence) la proximité minimale (lequel on pourrais considérer comme collision) et un son le mieux favorable à une distance sûre.

En considérant 4 distances et leurs fréquences <10cm (distance minimale ou collision), 10-30cm (distance un peu sur), 30-40cm (distance sur) et >40cm (distance très sur) Tableau 1.

Tableau 1: Fréquences en fonction de la distance

Distances	Fréquences
<10 cm	3 KHz
10 – 30 cm	23 Hz
30 – 40 cm	3 Hz
> 40 cm	1 Hz

```

43
44          --sonor ranger
45      if(Result > 40) then sounder <="11";      --distance au dela de 40cm
46
47      else if(Result <40 and Result >30 ) then sounder <="10"; -- distan
48
49      else if(Result <30 and Result >10 ) then sounder <="01";
50
51      else if(Result <10) then sounder <="00"; --distanc
52          end if;
53      end if;
54      ...

```

Figure 16: Extrait du code VHDL de **Distance_calc**, **sounder[2..1]**

Le signal **sounder[2..1]** Figure 16 c'est un signal 2 bits (binaire) la ou j'associe 4 distances en 4 donnees binaire Tableau 2

Tableau 2: Binaire en fonction de la distance

Distances	Binaire
<10 cm	00
10 – 30 cm	01
30 – 40 cm	10
> 40 cm	11

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ENTITY Buzzer_sounder IS
4
5  PORT ( sounder : IN std_logic_vector (2 DOWNTO 1);
6        freq1, freq2, freq3, freq4 : IN std_logic;
7        buzzer : OUT std_logic);
8  END Buzzer_sounder;
9
10 ARCHITECTURE behavioral OF Buzzer_sounder IS
11   SIGNAL a,b,c,d,f : std_logic;
12 BEGIN
13   b <= transport freq4 after 200000us;
14   c <= transport freq3 after 80000us;
15   d <= transport freq2 after 30000us;
16   f <= transport freq1 after 20000us;
17
18   --c <= transport freq3 after 6000 ms;
19   --d <= transport freq2 after 5000 ms;
20   --f <= transport freq1 after 2000 ms;
21
22   WITH sounder (2 DOWNTO 1) SELECT
23     a <= b WHEN "11",
24               c WHEN "10",
25               d WHEN "01",
26               f WHEN "00";
27   buzzer <= a;
28
29 END behavioral;

```

Figure 17: Code VHDL Buzzer_sounder

Après avoir saisie les fréquences et distance associe (sous forme binaire), je crée 4 signaux sonore en ajoutant des délais temporaire a chaque fréquence (le choix de délais a été effectué par *trial and fail* pour pouvoir mieux déterminer les délais convenable enfin d'avoir des sons favorable).

D'après Figure 17 :

- ligne 13 – 16 : *b <= transport freq4 after 200000us;*
c <= transport freq3 after 80000us;
d <= transport freq2 after 30000us;
f <= transport freq1 after 20000us;

- ligne 23 – 26 : le choix du son à sortir en fonction du binaire de la distance

Le schéma bloc **div1hz** Figure 18 est un compteur qui a été utilisé enfin d'avoir plusieurs fréquences.

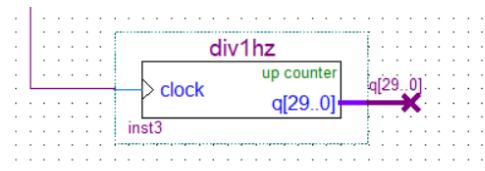


Figure 18: Compteur diviseur de fréquences

• Configuration des Pins

Après l'assemblage complet des toutes les symboles Figure 19 et la compilation, je suis passé à l'affectation des pins Tableau 3.

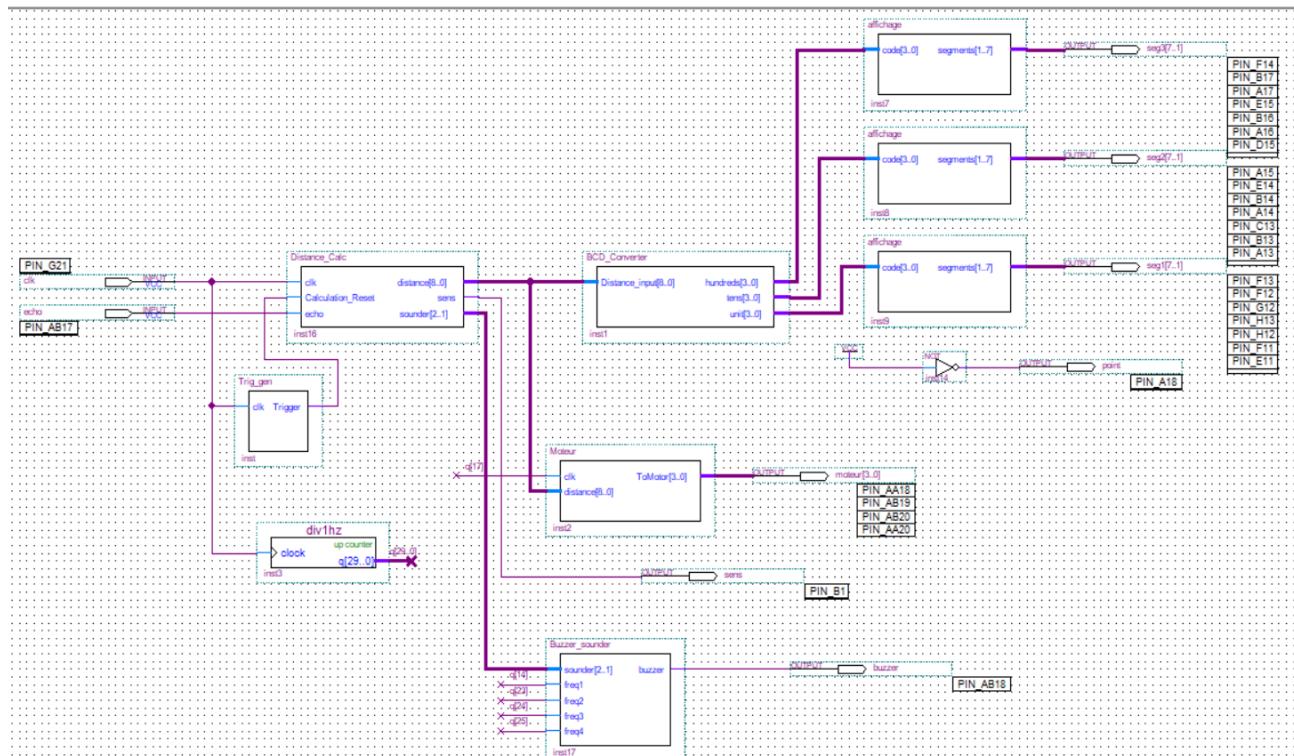
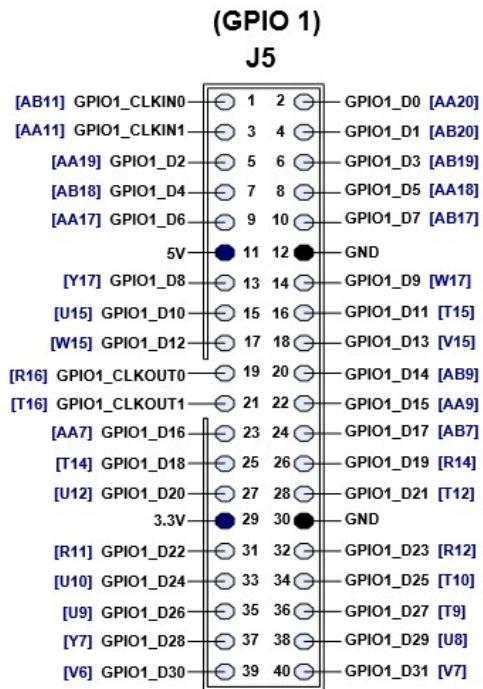


Figure 19: Schéma complet du projet

Tableau 3: Affectation des Pins

Broche	Pin	
clk	G21	
Trigger	AA17	
Echo	AB17	
Buzzer	AB18	
Moteur	AA18 AB19 AB20 AA20	
Afficheurs 7 segments	Les unités	F13 - E11
	Les dizaines	A15 - A13
	Les centaines	F14 - D15
Point	A18	



Application et utilités

Le mini-projet ci dessus peut être utiliser dans la détection de distance et signaler en cas forte danger. Quelques applications :

- Radar de proximité de voiture
- Pour les personnes avec des problèmes de vision
- Systèmes d'alarmes

. Conclusion

En général le mini-projet se dérouler avec succès. J'ai vu le capteur ultrason, moteur pas à pas et buzzer leurs fonctionnements, utilisations et configurations pour des fonctionnements souhaité.

J'ai aussi du faire face à quelques contraintes notamment le développements des codes vhdl faisant marche les composants. La configuration du moteur, la ou la mauvaise fréquence ou petite variance empêcher son mise en marche.

. Sources et quelques liens

- Extrait vidéo du fonctionnement :
[https://drive.google.com/file/d/1mEO9FwfFUUvbu6f2wGkQKZBKNcaEfqJl/view?
usp=sharing](https://drive.google.com/file/d/1mEO9FwfFUUvbu6f2wGkQKZBKNcaEfqJl/view?usp=sharing)
- DEO User Manual :
[http://moodle.univ-fcomte.fr/pluginfile.php/617231/mod_resource/content/1/
DE0_User_manual.pdf](http://moodle.univ-fcomte.fr/pluginfile.php/617231/mod_resource/content/1/DE0_User_manual.pdf)
- Énoncé du mini-projet (page 20-21) :
[http://moodle.univ-fcomte.fr/pluginfile.php/1181314/mod_resource/content/1/
Poly_TP_EP02_2019-2020.pdf](http://moodle.univ-fcomte.fr/pluginfile.php/1181314/mod_resource/content/1/Poly_TP_EP02_2019-2020.pdf)
- Datasheet Capteur Ultrason :
<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>
- Datasheet module ULN2003 :
<https://pdf1.alldatasheet.com/datasheet-pdf/view/125994/ETC1/ULN2003.html>
- Internet