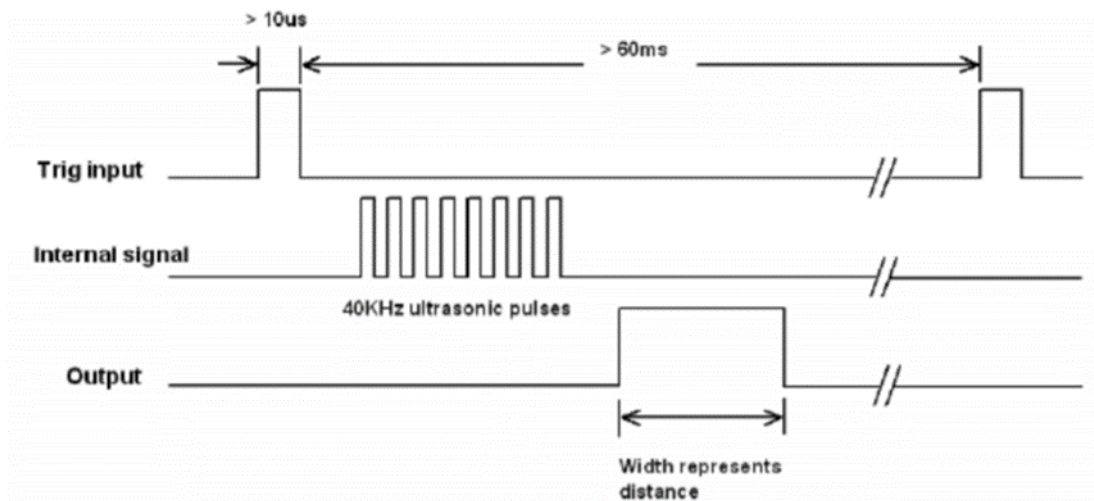


COMPTE - Rendu

1.2.1 Simulation de l'IP

« Reporter dans le compte-rendu de projet les captures d'écrans de la simulation de l'IP avec les explications sur son fonctionnement et les tests réalisés. »

Fonctionnement



Dans le fichier VHDL, nous avons conçu 3 processus :

➤ **count: process(rst, clk)**

```
count: process(rst, clk)
begin
    if rst = '0' then
        trig_counter <= (others => '0');
        wait_counter <= (others => '0');
    elsif clk'event and clk = '1' then
        if trig_counter <= To_unsigned(period1,32) then
            trig_counter <= trig_counter + 1;
        else
            if wait_counter <= To_unsigned(period2,32) then
                wait_counter <= wait_counter + 1;
            else
                trig_counter <= (others => '0');
                wait_counter <= (others => '0');
            end if;
        end if;
    end if;
end process;
```

Ce processus est utilisé pour le chronométrage. Nous savons que l'horloge système est

de 50 MHz. Si nous voulons que le signal « trig » dure 10us, alors nous utilisons le signal « trig_counter » pour compter 600 fronts montants de l'horloge système. De même, nous utilisons le signal « wait_counter » pour compter 3000000 fronts montants de l'horloge système pour garantir le signal « wait » dure 60 ms, et après que le signal « wait_counter » dépasse 3000000, le signal « wait_counter » et le signal « trig_counter » sont effacés de sorte que nous avons atteint un intervalle de temps pour une mesure.

➤ **output_cycle: process(rst,clk)**

```
output_cycle: process(rst,clk)
begin
    if rst = '0' then
        output_trig <= '0';
    elsif clk'event and clk = '1' then
        if trig_counter <= To_unsigned(period1,32) then
            output_trig <= '1';
        elsif trig_counter > To_unsigned(period1,32) then
            output_trig <= '0';
        end if;
    end if;
end process;
```

Ce processus met le signal « output_trig » à 1 lorsque le signal « trig_counter » <= 600, et met le signal « output_trig » à 0 lorsque le signal « trig_counter » > 600, afin que nous puissions réaliser le chronogramme ci-dessus.

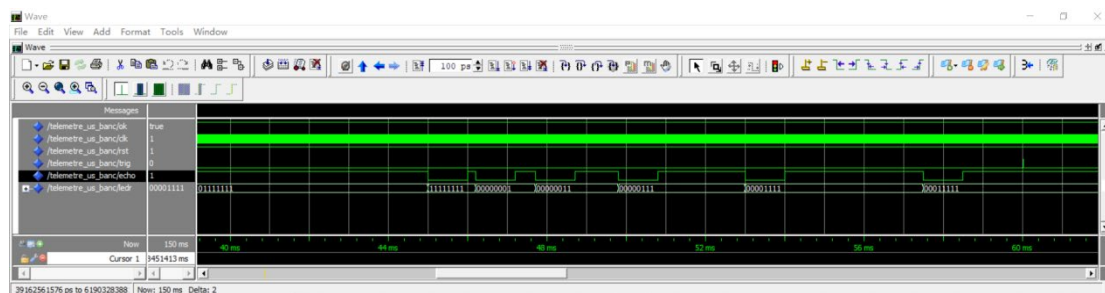
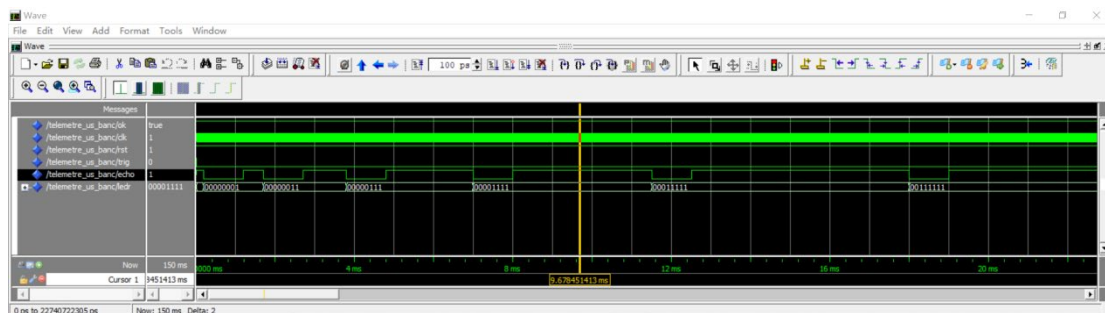
➤ **LED: process(rst, clk)**

```
LED: process(rst, clk)
begin
    if rst = '0' then
        output_LEDR <= (others => '0');
    elsif clk'event and clk = '1' then
        if echo = '0' then
            if state = '1' then
                if duree_counter <= x"00003a98" then output_LEDR <= "00000001";
                elsif duree_counter <= x"00007530" then output_LEDR <= "00000011";
                elsif duree_counter <= x"0000ea60" then output_LEDR <= "00000111";
                elsif duree_counter <= x"0001d4c0" then output_LEDR <= "00001111";
                elsif duree_counter <= x"0002bf20" then output_LEDR <= "00011111";
                elsif duree_counter <= x"00047c70" then output_LEDR <= "00111111";
                elsif duree_counter <= x"000900b0" then output_LEDR <= "01111111";
                else output_LEDR <= "11111111";
            end if;
            state <= '0';
            distance <= duree_counter;
        elsif state = '0' then
            duree_counter <= (others => '0');
        end if;
    elsif echo = '1' then
        state <= '1';
        duree_counter <= duree_counter + 1;
    end if;
end if;
end process;
```

Ce processus associe la durée du signal d'écho (proportionnelle à la distance) au nombre de lumières LED « output_LED ». Nous utilisons le signal « duree_counter » pour compter la longueur du signal d'écho, et le divisons en 8 parties en fonction de la taille de « duree_counter ». Les détails sont les suivants :

Distance / cm	Nb de LED	Max / cm	Durée d'écho / ms = $(2 * \text{Max}) / (340 \text{ m/s})$	Nb de clk = Durée / 20ns
0 ~ 5	1	5	0.3	15000 (0x 0000 3198)
5 ~ 10	2	10	0.6	30000 (0x 0000 7530)
10 ~ 20	3	20	1.2	60000 (0x 0000 ea60)
20 ~ 40	4	40	2.4	120000 (0x 0001 d4c0)
40 ~ 60	5	60	3.6	180000 (0x 0002 bf20)
60 ~ 100	6	100	5.88	294000 (0x 0004 7c70)
100 ~200	7	200	11.8	590000 (0x 0009 00b0)
> 200	8	∞		

Tests Réalisés

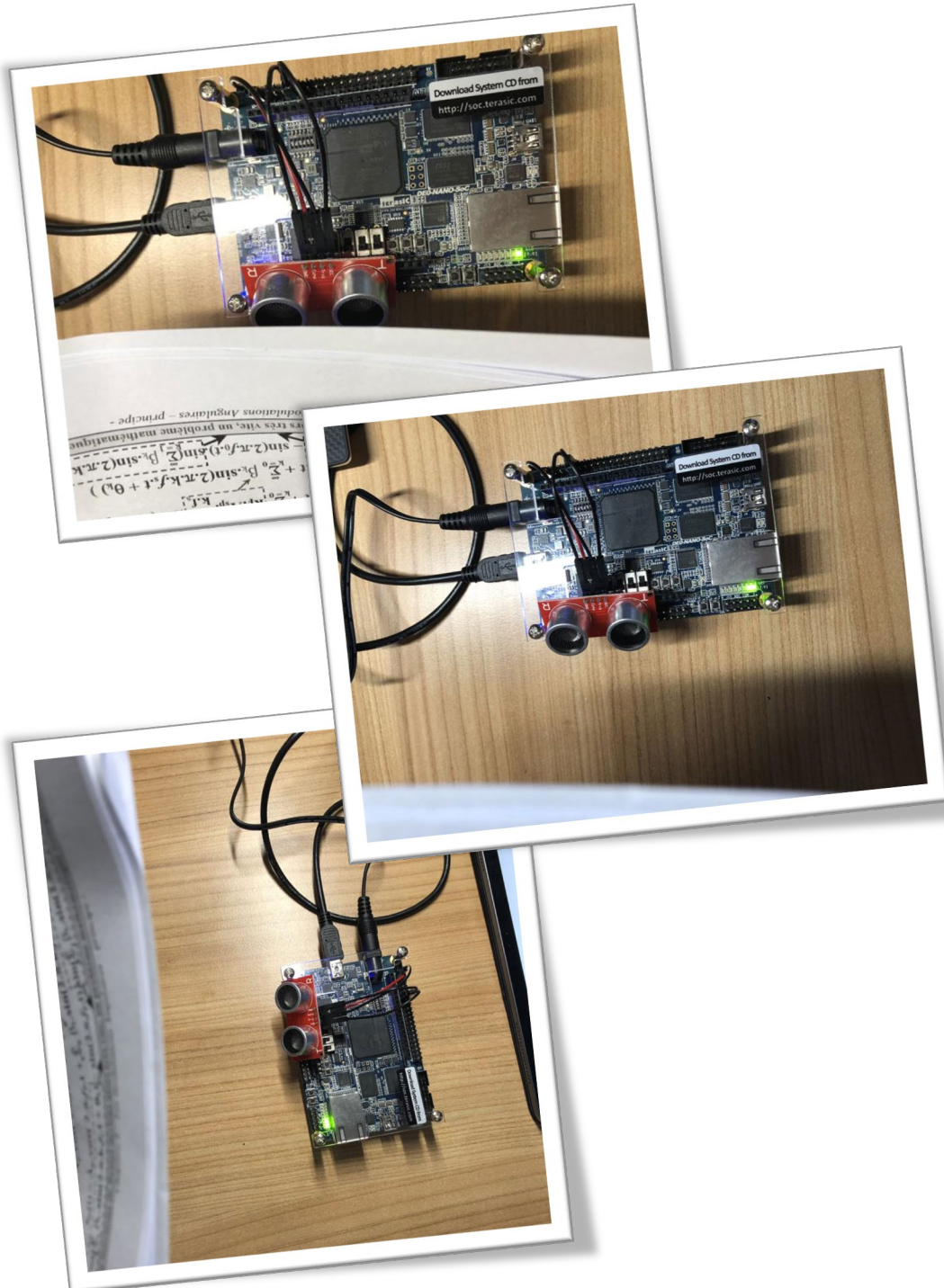


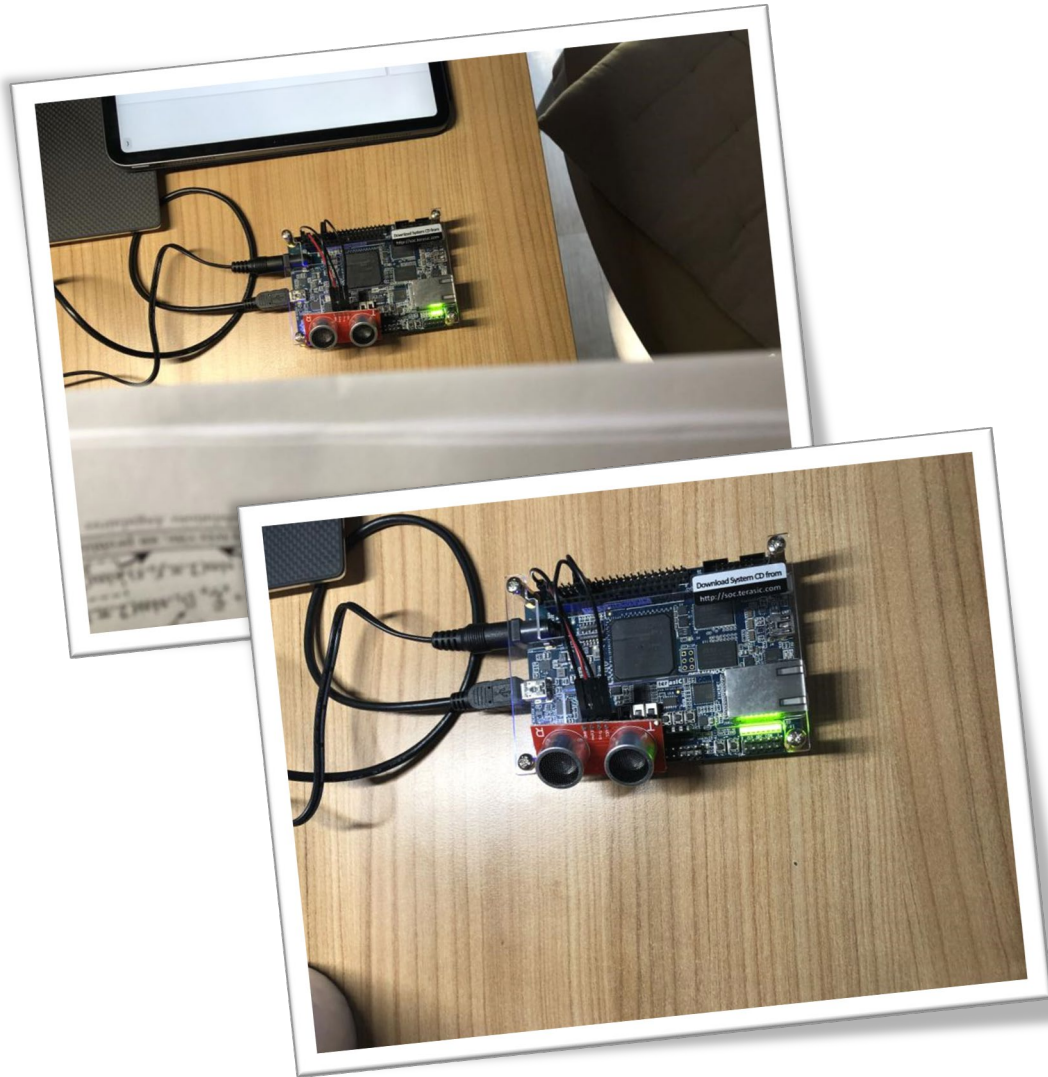
Nous pouvons voir à partir du diagramme de simulation que lorsque « echo » dure 0,2 ms, le bit le plus bas de « ledr » est 1 et les autres bits sont 0. Après cela, l'écho dure

respectivement 0,5 ms / 1,1 ms / 2,2 ms / 3,5 ms / 5,5 ms / 11 ms / 14ms, le nombre de 1 de la sortie ledr augmente progressivement. Par conséquent, la simulation de l'IP fonctionne normalement.

1.2.2 Test de l'IP sur carte

« Faites des photos à rajouter dans votre compte rendu pour montrer que les valeurs des LEDs évoluent en fonction de la distance d'un obstacle tenu devant le télémètre. »





Nous utilisons le papier comme obstacle, laissant le capteur à ultrasons de près et de loin, jusqu'à ce que finalement nous supprimons l'obstacle, nous pouvons voir sur les photos, le nombre de LED allumées augmente progressivement. Par conséquent, Test de l'IP sur carte est réussie.

1.2.3 Intégration de l'IP Télémètre dans Qsys

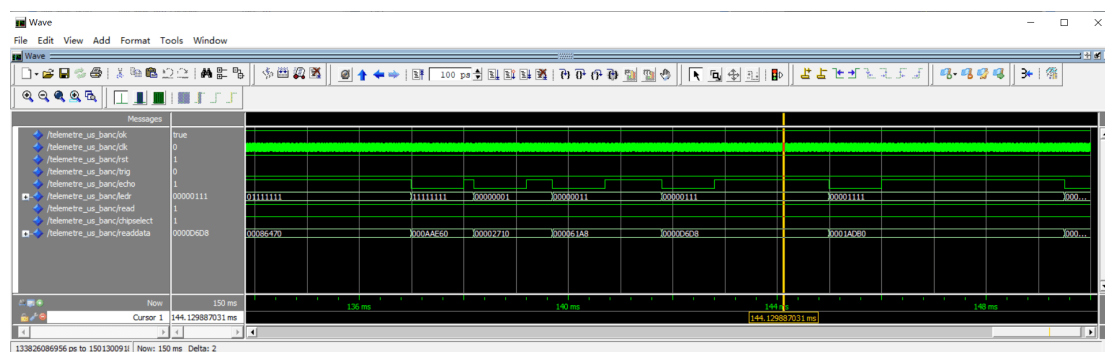
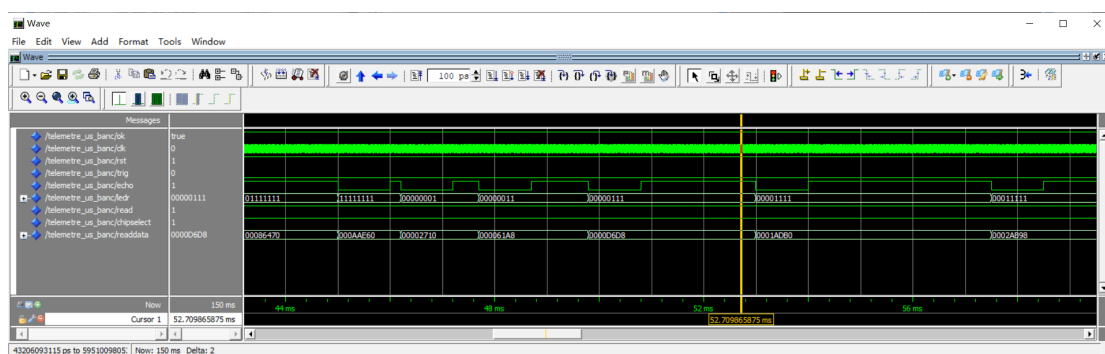
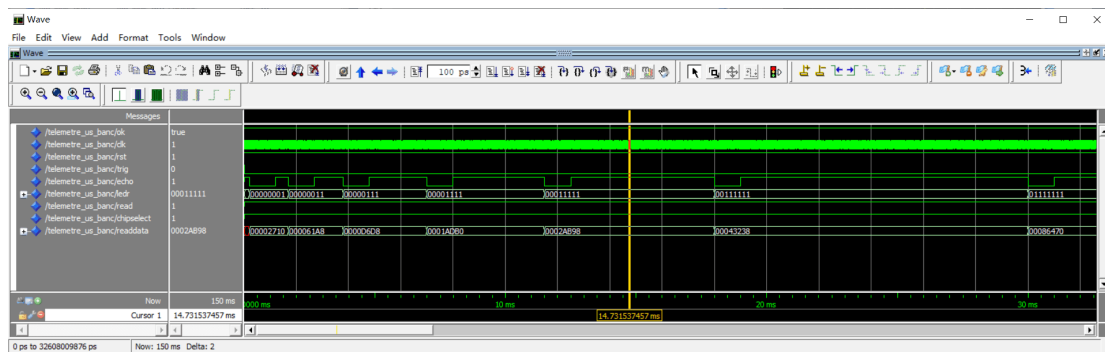
« Reporter dans le compte-rendu de projet les captures d'écrans de la simulation de l'IP avec les explications sur les tests réalisés. »

Tests Réalisés

Nous avons ajouté les trois interfaces read, chipselect et readdata au code VHDL, et ajouté l'instruction suivante :

```
readdata <= std_logic_vector (distance) when chipselect = '1' and read = '1' else (others
=> '0');
```

Lorsque chipselect et read valent tous les deux 1, la distance est affectée à readdata.



Après la simulation, on peut voir que l'interface Avalon fonctionne bien.

1.2.4 Programmation logicielle et test de l'IP

« Faites des photos à rajouter dans votre compte rendu pour montrer que cette partie fonctionne bien également. »

Dans le code de langage C, nous lisons le registre via le port IO, le multiplions par la vitesse du son, et obtenons enfin la distance qui peut être affichée à l'écran.

```

hello_world.c
+ * "Hello World" example.

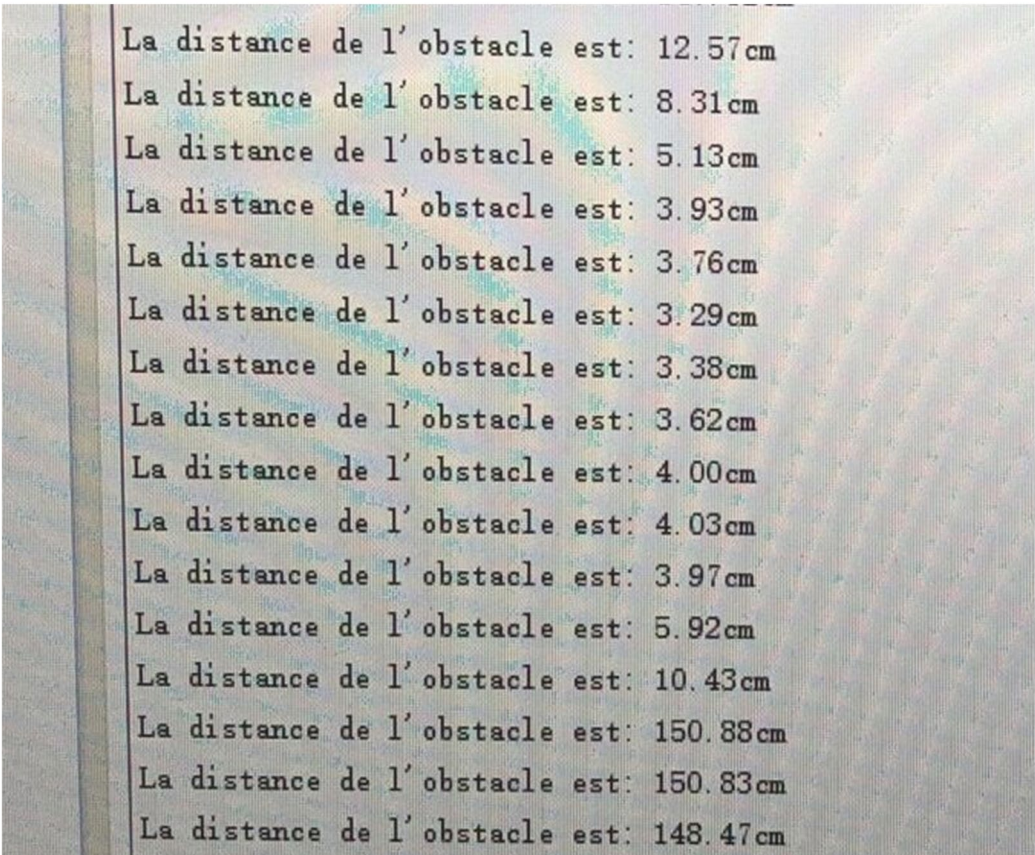
#include <stdio.h>
#include <unistd.h>
#include <io.h>
#include "system.h"

int main()
{
    int nb_clk;
    float distance;
    printf("Nous allons commencer a mesurer!\n");
    while(1) {
        usleep(100000);
        wait(3);
        nb_clk = IORD(TELEMETRE_US_INST_BASE,0);
        distance = nb_clk * 0.00034;
        printf("La distance de l'obstacle est: %.2fcm\n",distance);
    }

    return 0;
}

```

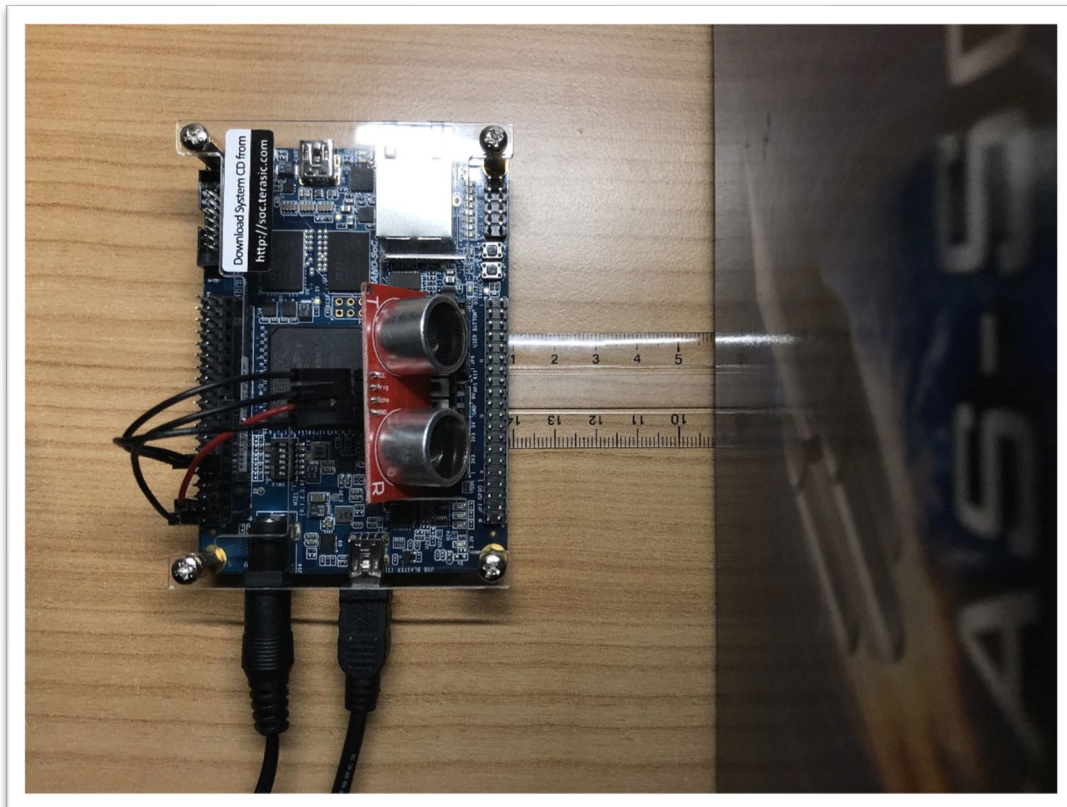
Voici la photo de l'exécution et les photos correspondes :



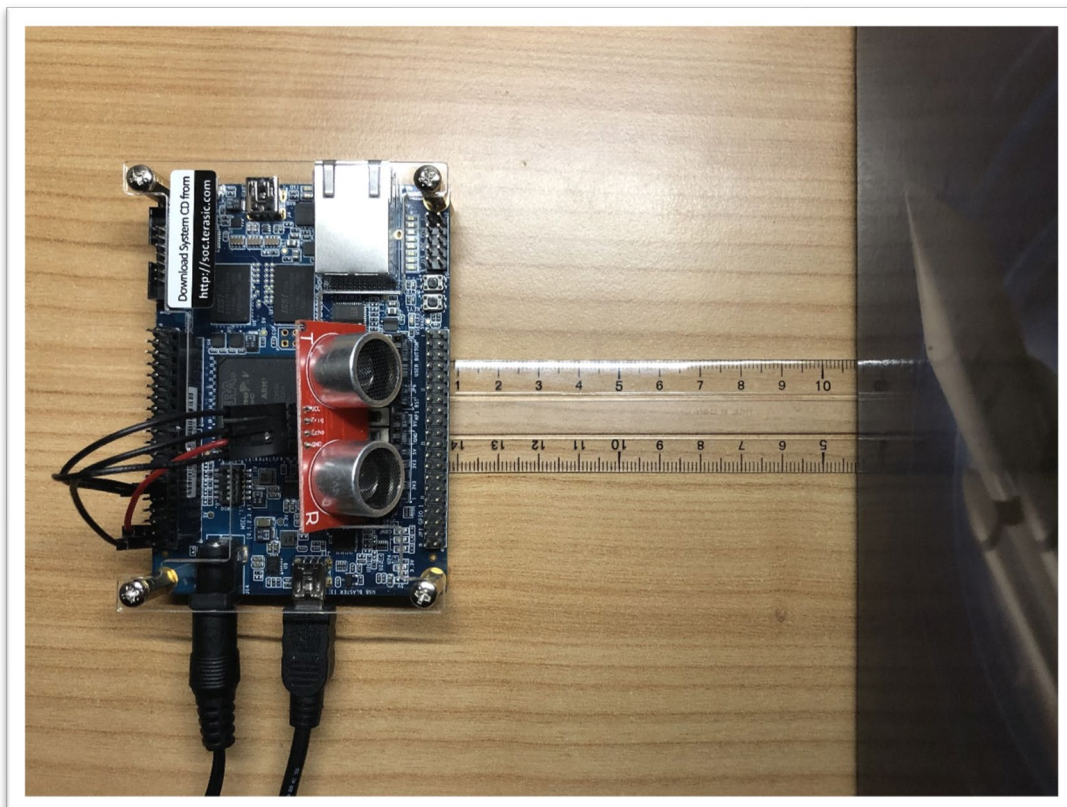
```

La distance de l'obstacle est: 12.57cm
La distance de l'obstacle est: 8.31cm
La distance de l'obstacle est: 5.13cm
La distance de l'obstacle est: 3.93cm
La distance de l'obstacle est: 3.76cm
La distance de l'obstacle est: 3.29cm
La distance de l'obstacle est: 3.38cm
La distance de l'obstacle est: 3.62cm
La distance de l'obstacle est: 4.00cm
La distance de l'obstacle est: 4.03cm
La distance de l'obstacle est: 3.97cm
La distance de l'obstacle est: 5.92cm
La distance de l'obstacle est: 10.43cm
La distance de l'obstacle est: 150.88cm
La distance de l'obstacle est: 150.83cm
La distance de l'obstacle est: 148.47cm

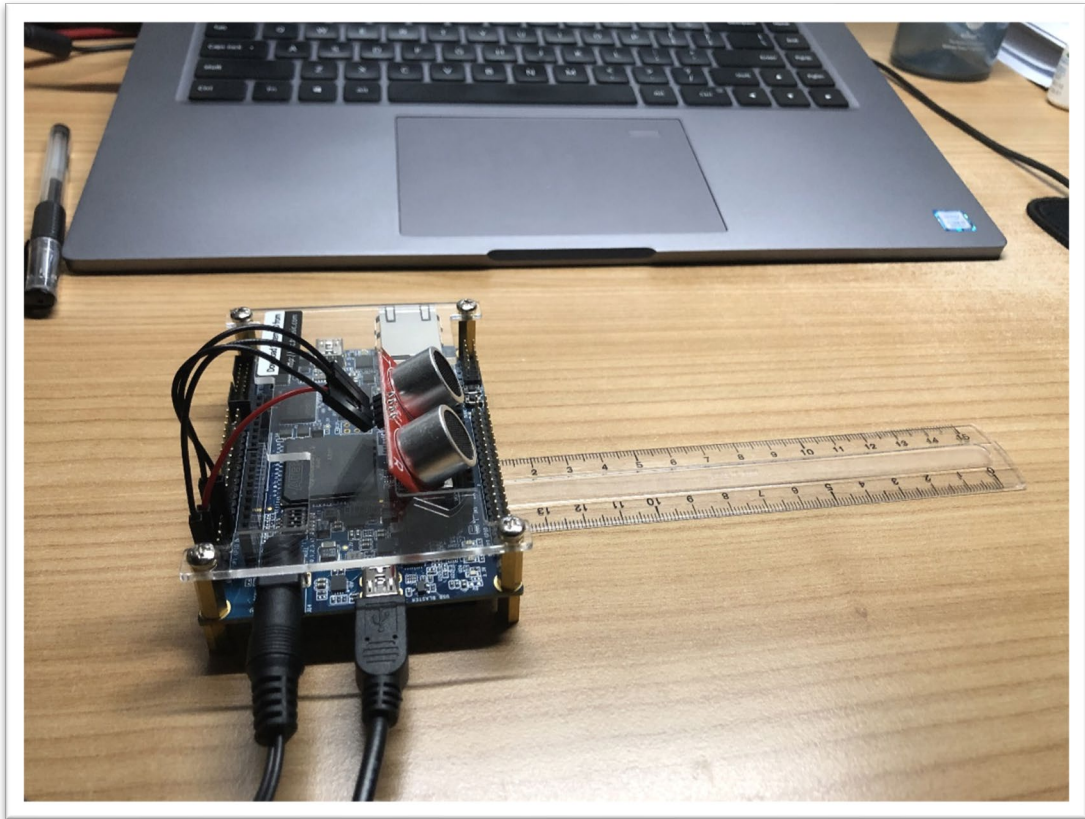
```

Correspond à distance = 5.92cm



Correspond à distance = 10.43cm



Correspond à distance = 150cm, pas d'obstacle

En conclusion, cette partie fonctionne bien également.