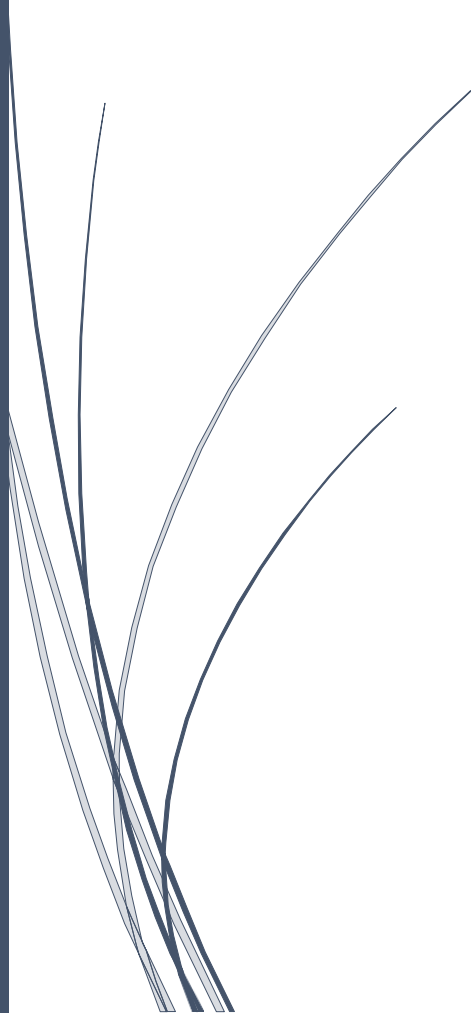
L3 E3A

Ultrasound Recoil Radar

Electronic studies



Xavier Goeman
Sebastien Girard
Guillaume Somonnian
Nabil El Fodil

Table of Materials

1) Introducing the project:

It is about designing, performing, and testing a device capable of detecting the presence of an obstacle using the propagation of ultra-sound waves.

The principle is to measure the time taken by an ultra-sound wave to make a round trip between the measuring device and the obstacle. This temps is then converted and, when the distance between the device and the object is too small, a enslavement signal is emitted.

This project is divided into three sub-parts:

- Generate an ultra-sound wave with a frequency of 4khz, trigger and stop counting.
- Amplification and filtering: the production of amplifiers for wave emission and reception.
- Counting: making a counter block, managing the clock, converting time/distance.

2) PCB creation and welding:

Once the operation of the ultrasonic transmitter/receiver and the PW0268 chip was understood, we made a diagram with all the necessary components. We followed the instructions of the datasheet of the PW0 and our reference professor to obtain the diagram in Appendix 2.1.

We then implemented this scheme under the Eagle Appendix 2.2 CAD. Unfortunately, we realized that Eagle's libraries, although very comprehensive, did not have all the components we needed. Indeed, we lacked the diagrams and fingerprints of the transformer, the PW0268 and the transmitters/receivers.

To overcome this problem, we had to create new bookstores, containing the so-called missing component, under Eagle. To do this, you had to create three files and then "bind" them: a symbol file containing the component schematic, another named package for its fingerprint and the last, device, to link the previous two.

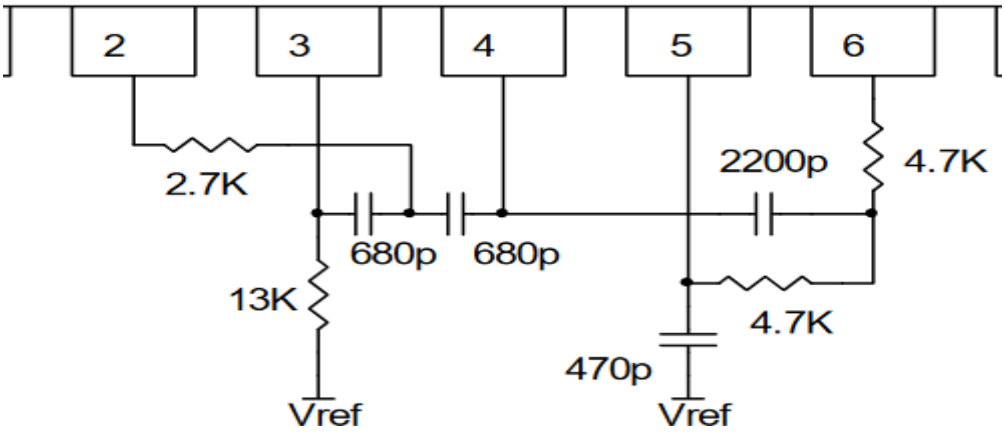
For the receiver transmitters, we have inquired directly with the datasheet provided on radio spear. We found the size of the components as well as the diameter of their pins and their spacing. For the transformer, we took one of those provided by our teacherand determined the spaces between the4 pins as well as the total surface area of the element.

Finally, for the PW0 chip, we took an imprint of a similar component but with 4 additional pins.

We will detail the different parts. First of all, we have the pine 1 that serves as a link between the FPGA card and the PCB. It is on this pine that the FPGA emits the pulse and receives the pulse back.

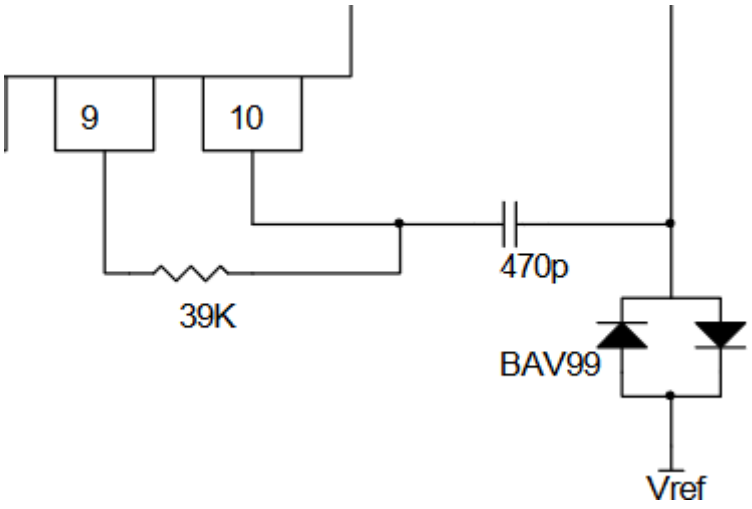
On pines 2 to 6 we have a filter pass band broken down into two distinct parts passes high, pines 2 and 3, and a low pass, pines 4 and 5. The frequency of our arousal wave (wave emitted) generated by the FPGA card is 40khz, so we try to keep the same order of magnitude for receiving the reflected wave (after contact with the obstacle). For this we will use a tape pass by adapting the values of its components to get a bandwidth from 33khz to 41khz. This will allow us to filter out noises and interferences at reception (Low pass cut frequency 31khz, and high pass frequency is 41khz)

The values of the components here are not the same as those we used.



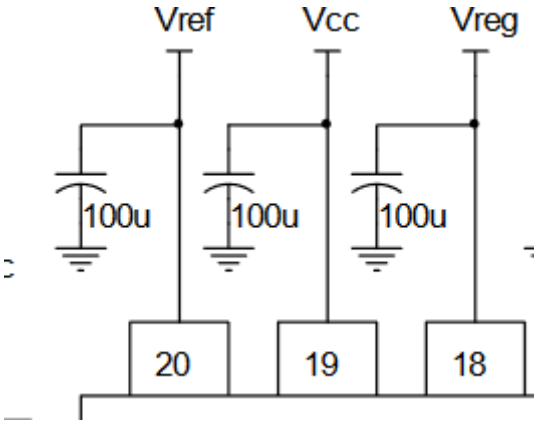
On pines 7 and 8 we have a binding capacity that only serves to block the polarization tensions (continuous tension) between the two pines, but which allows dynamic variations to pass through, i.e. signals.

On pines 9 and 10 we have a resistance of counter reaction, the gain of the pre-amplifier is adjusted to be able to accommodate the transmitter with several sensitivities, playing on the value of this resistance. (CF: datasheet)

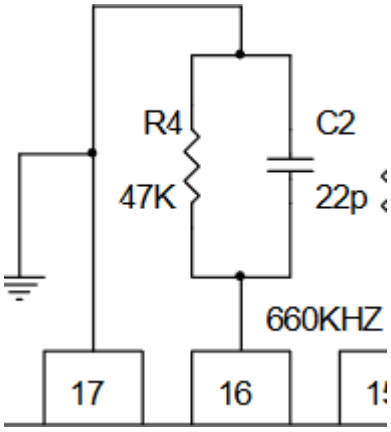


The capacity is then connected to the ultrasound receiver on the PCB.

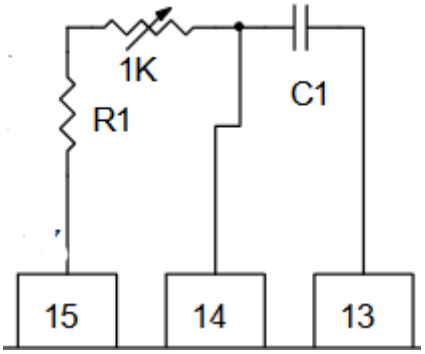
Pine 19 receives 7V power for the PW0268 chip, pines 20 and 18 are used to generate 2 other sources of tension Vref and Vreg. They are all three related to polarized decoupling capabilities. They aim to reduce the internal impedance of these foods.



As far as pine 16 is concerned, this is the System Clock. As the block diagram shows, if the System Clock corresponds to 660khz the components are therefore C2 - 22pF, R4-47k Ohm, then the maximum size input pulse is 0.6ms (in practice our pulsation lasts 0.4ms). Any pulse exceeding this time will be ignored. In practice our frequency is about 450 kHz. (CF: datasheet)



Pines 15 to 13 are used to implement the oscillator that will allow the transmitter to send the ultrasonic waves. In practice our oscillator has a frequency of 40kHz.



On pine 11, we have an elevator transformer. The wave created by the CArte FPGA has a fixed voltage of 7V, we use this lift transformer to increase the initial voltage around 30V output, so that we can have a fairly large range (in practical case we can visualize the distance Iuse to theceiling: 2089 mm).

From that moment on, we started the creation of the PCB, namely, to place our elements logically without them "disturbing" each other, that the connectors are accessible and that the ultrasonic receiver transmitters are well aligned, see Appendix 3.

3) Programming:

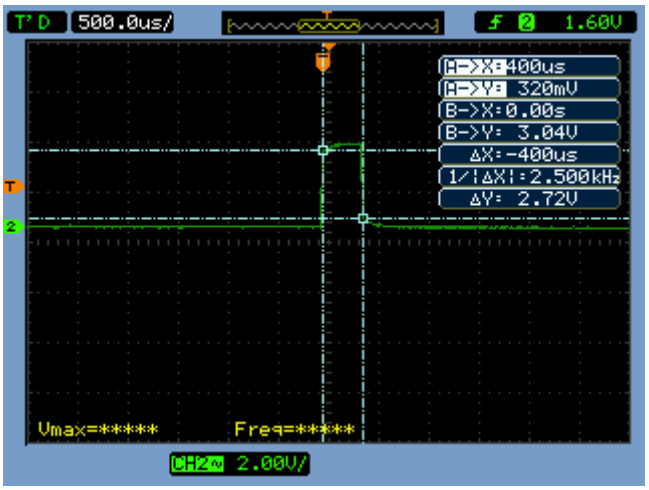
The programming part of the project is essentially done in two stages, the first will be used to send a pulse on the electronic card and then the second will translate the return signal received on the FPGA card from the electronic card

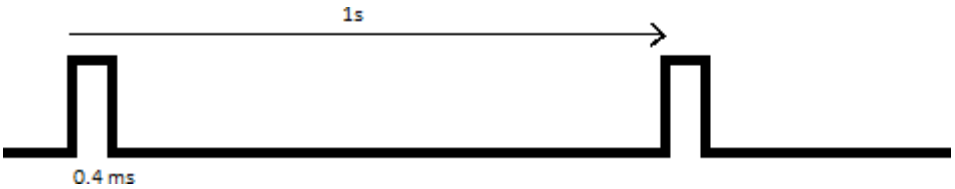
a) Pulse at 1 Hz:

In order for the card to generate the ultrasonic wave it is first necessary to impose a command, for this, using the FPGA card it is sent a square pulse of 3.3V with a frequency of 1Hz, the code that generates this pulse is relatively simple. (Annex 3-1)

We are working with a 50MHz clock, in order tosend the pulse of 1Hz, so we have to divide this frequency by 50,000,000. Finally, for the ultrasound transmitter to function properly, the duration of the pulse should not exceed 0.6ms, so we chose to take a duration of 0.4ms.

In addition, the signal must be reversed out of the block, which explains the presence of the inverter block in the block scheme.

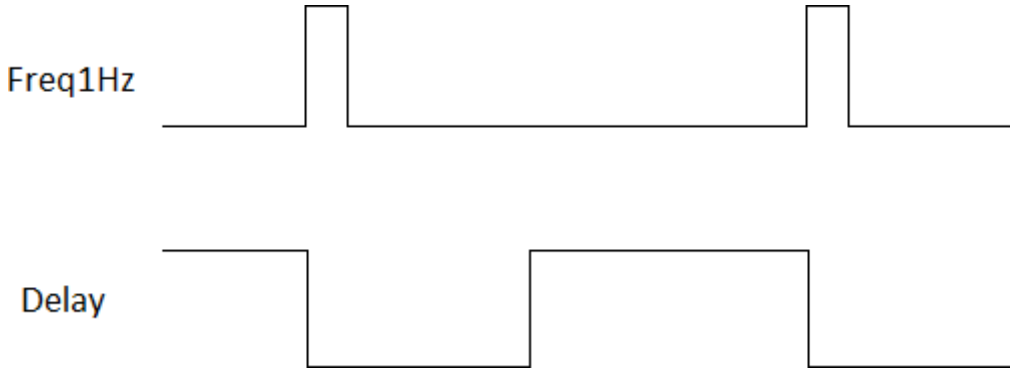




b) Simulation of the return signal:

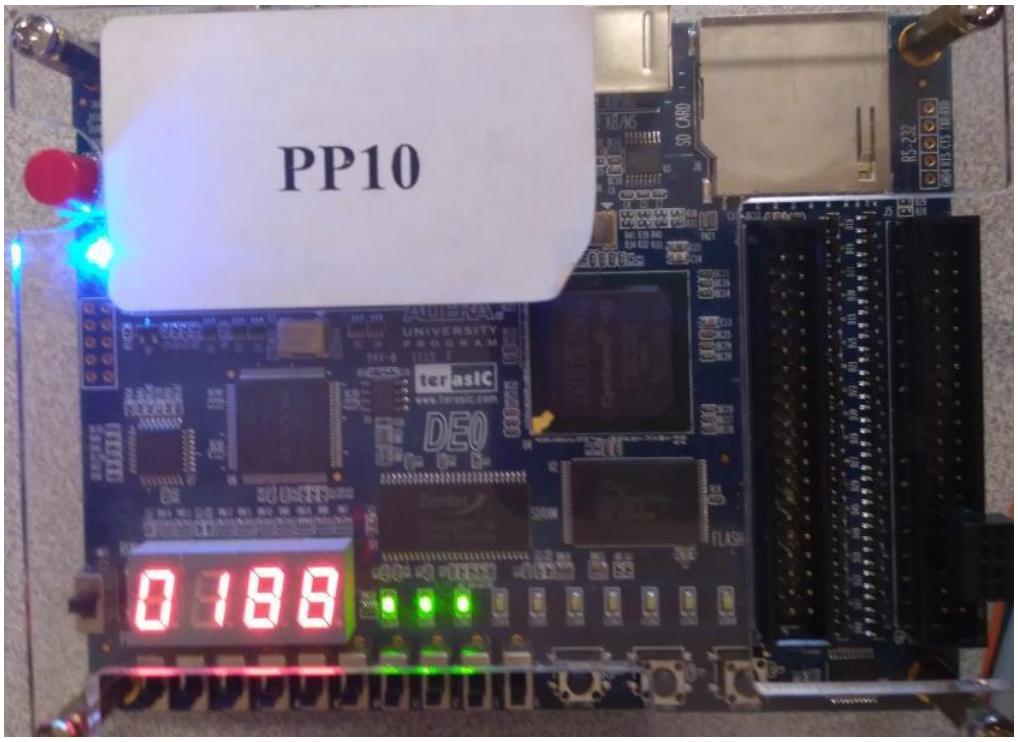
Before we could send the pulse on the electronic map we went through a phase of signal simulation back thanks to the "delay" function (Annex 3-2).

Thisc ode allows us to simulate the reception of a signal returned by an obstacle on the FPGA card, for this, the "delay" block is synchronized with the pulse sent to 1Hz and resets to the front of it to go back up after a given time.



c) Distance calculation:

With the version of the FPGA card we have, we can display a distance in millimeters.



The emitted wave is a sound wave, so it propagates through the air at a speed of 340 m/s, so we can know how long the wave will travel 1mm.

$$t(s)=\frac{l(m)}{v(m/s)}=\frac{0{,}001}{340}=2{,}94.10^{-6}s \Rightarrow f(hz)=\frac{1}{t(s)}=340.000\,Hz$$

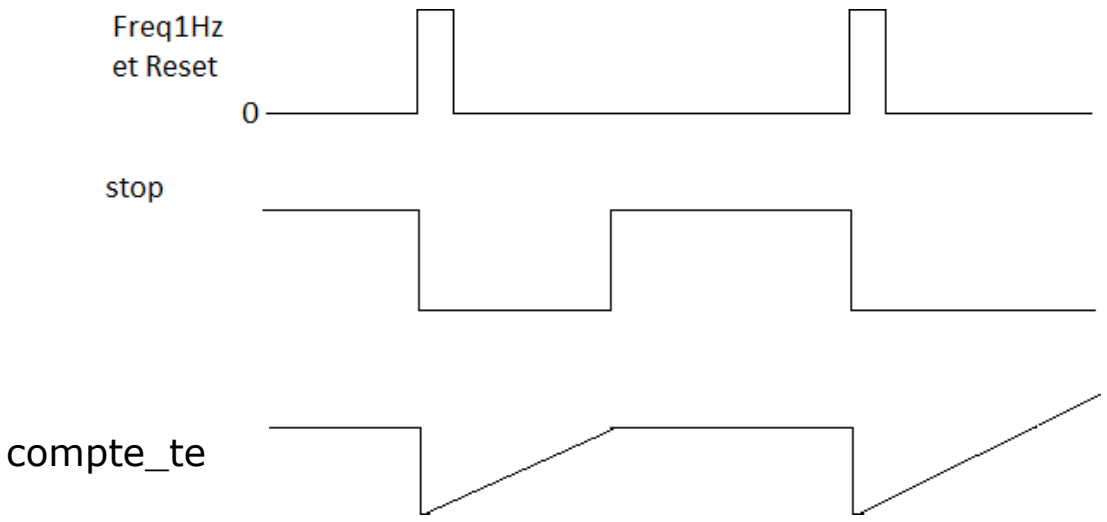
So we have to count all 3s (340kHz) in order to recover the distance of the reflected wave, however the wave makes a round trip if we count too fast. We will count both the distance back and forth and therefore have a factor 2 on the distance obtained. To overcome this fact we have to double this duration and therefore count all 6 (170kHz).

To be able to count at this speed we create the block "freq170kHz" (Annex 3-3), which is a clock that allows to count one millimeter with each stroke of the clock.

The result of the distance between the obstacle and the transmitter is made by the
'compte_temps', which inputs 'clock_50' to be synchronized with the rest of the program, 'the clock' at 170kHz that counts, and 'stop' that stops counting.

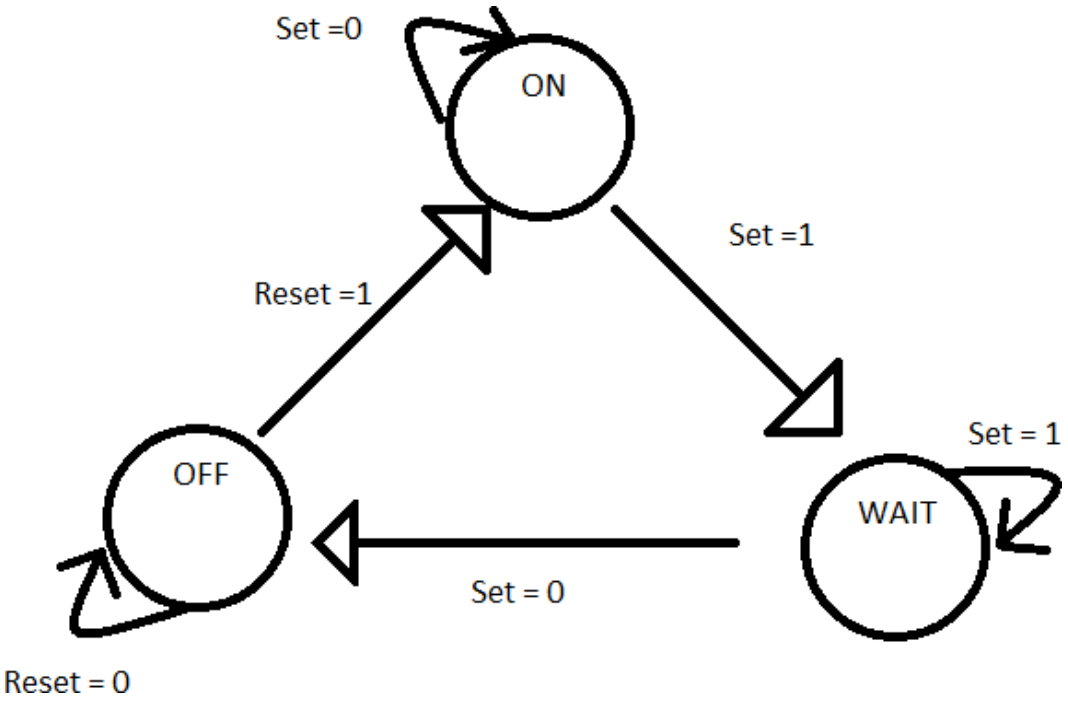
To view the result on the 7 segment displays, you have to rely directly on the bcd format. In addition, it sends a signal on the 10 LEDs of the FPGA card that gradually turns them on following different level (1 LED for 1500mm, 2 LEDs per 1000 mm...) (Annex 3-4)

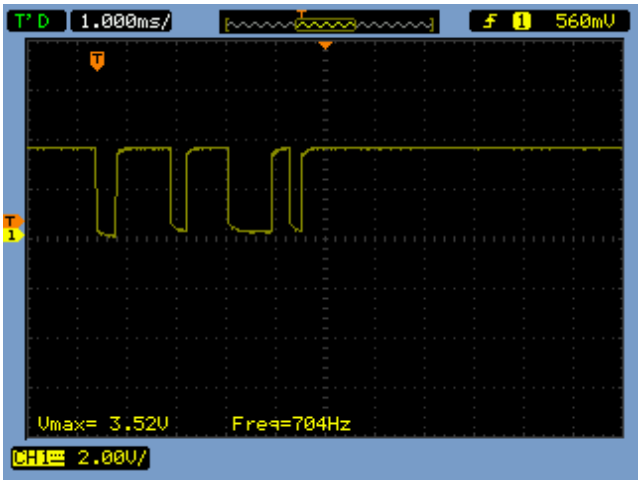
For Force the shutdown Of Block "compte_temps », We Place In Upstream A Block "Strength which is synchronized with the clock at 1Hz (so it recovers to 0 every second) and takes the exit of the "delay" block. This block will therefore send a logical signal 1 when there is a mount front"delay" function. (Annex 3-5)

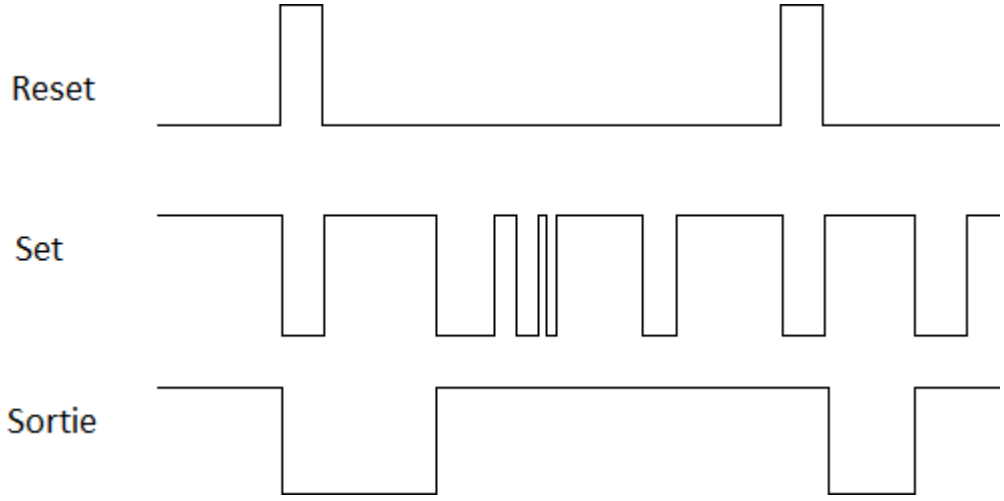


d) Real-life program:

When the card is connected to the PCB, the return signal is sent directly from it, so we no longer need the delay block, which was only useful for simulation. In addition, we change the "force" block to "force2" which is an automaton, which pulls out a logical signal 0 between the emission of the wave and its return (Annex 3-6).





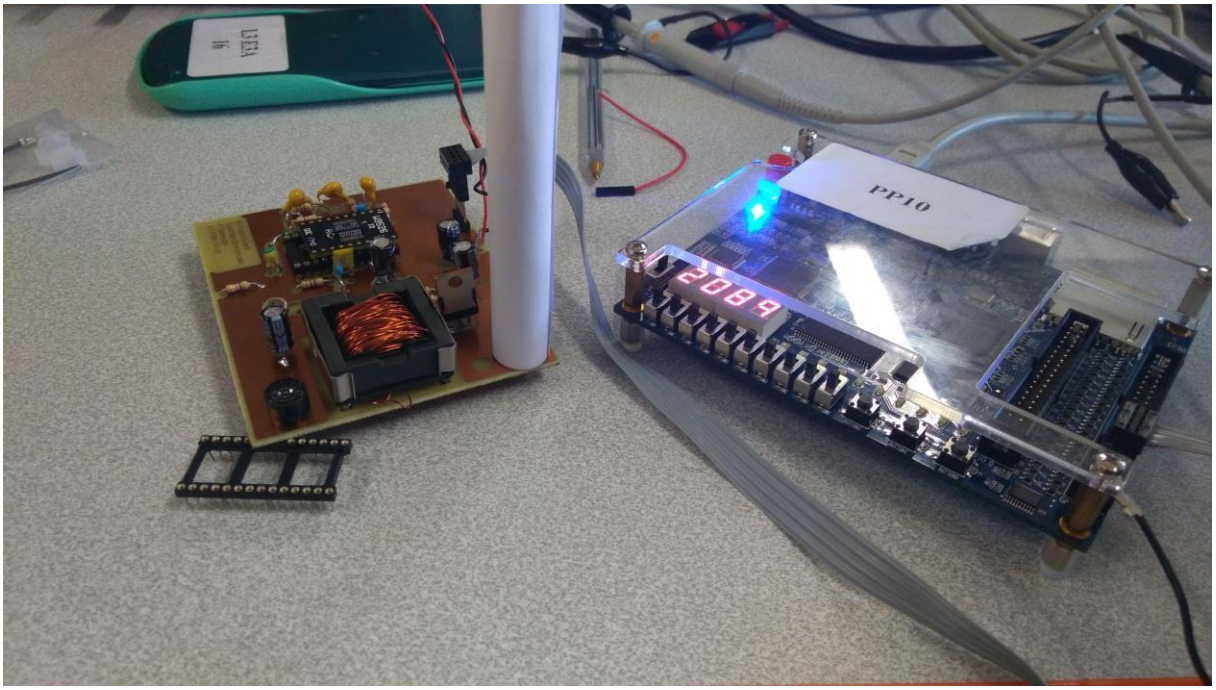


4) Conclusion:

This electronics project embodies all the concepts that we have been able to address this year.

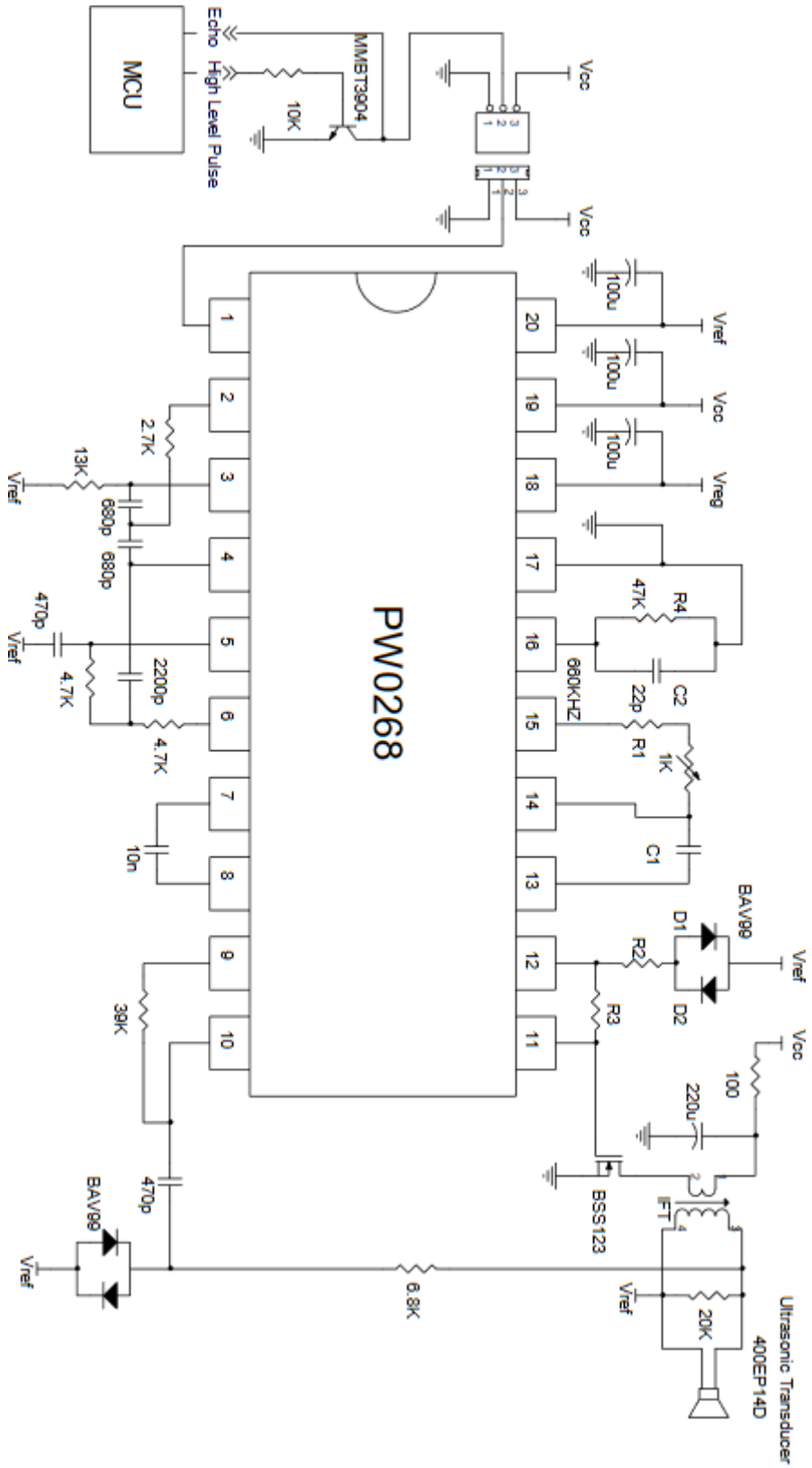
This technology is widely used in several industries, such as the automobile with parking assistance or blind spot radars, but alsoin years robotics.

The model we worked on obviously has its limits, it is only performing for short distances.

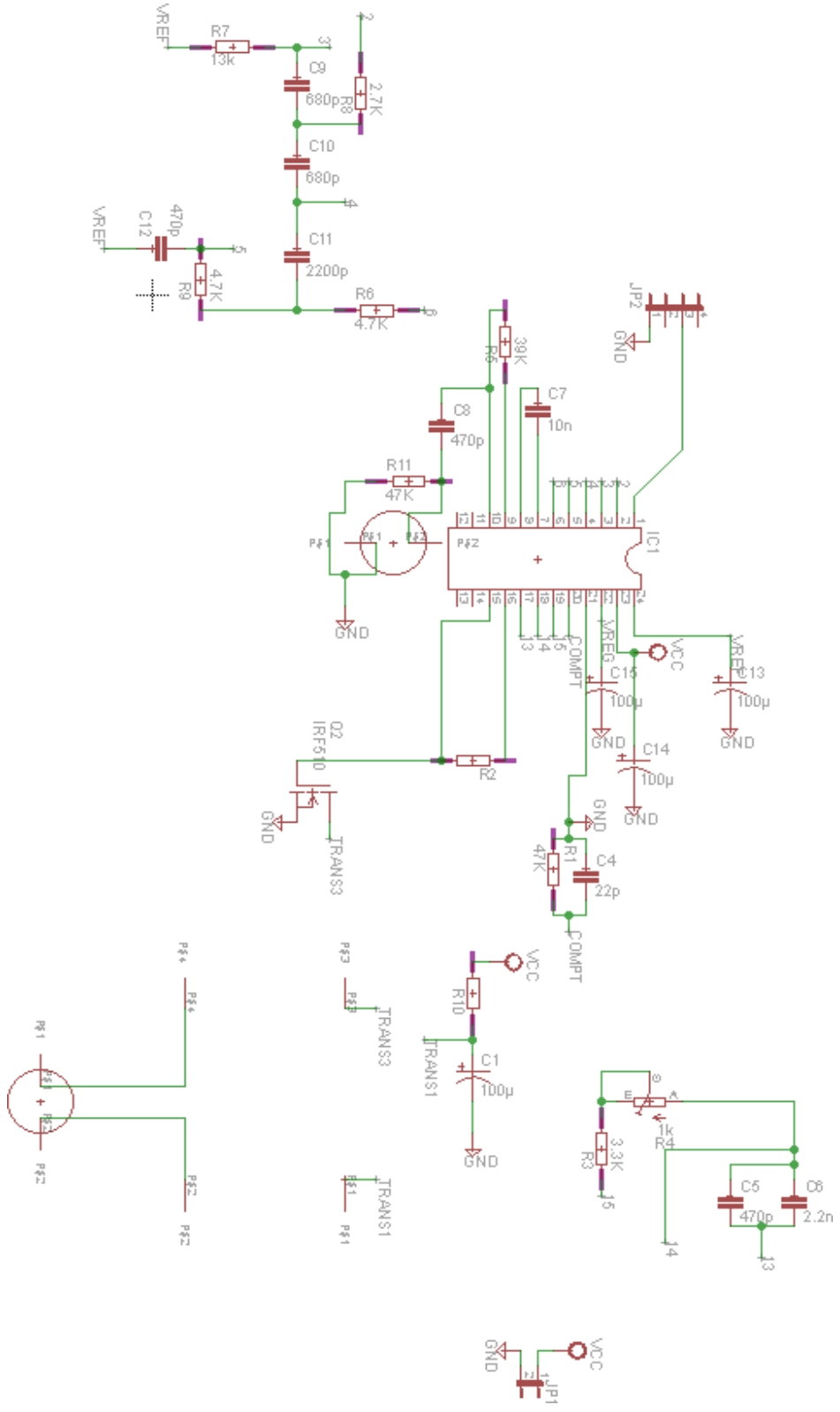


5) Annex

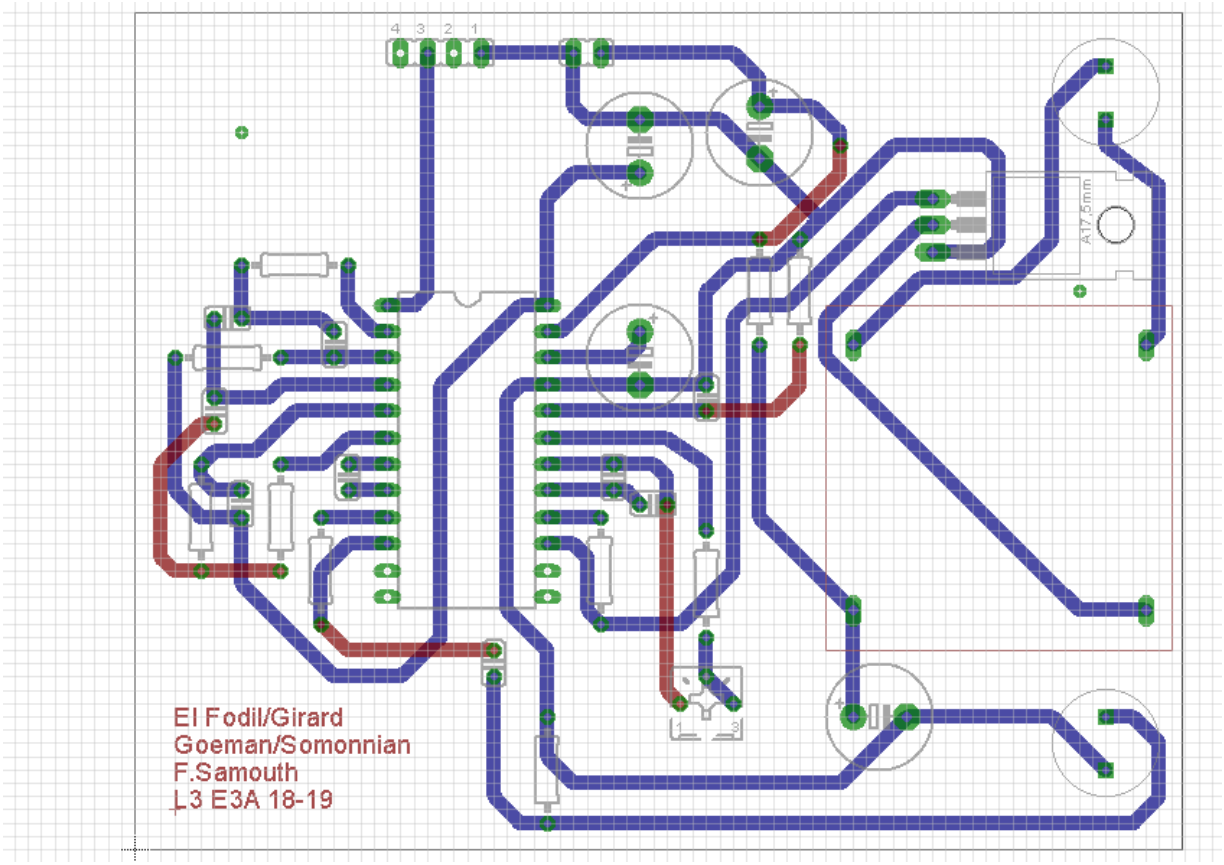
Appendix 2.1



Appendix 2.2


VREF
R7
13k
C9
680p
R8
2.7K
C10
680p
C11
2200p
C12
470p
R9
4.7K
R6
4.7K
JP2
GND
39K
C7
10n
C8
470p
R11
47K
IC1
VREG
COMPT
VCC
C15
100µ
C13
100µ
C14
100µ
GND
Q2
IRF510
R2
TRANS3
R1
47K
C4
22p
R10
C1
100µ
TRANS1
R3
3.3K
R4
1k
C5
470p
C6
2.2n
JP1

Appendix 2.3



Appendix 3-1: Freq1Hz

```
entity freqHz is

   port (clock: in std_logic;

                        freq: out std_logic);

end entity;

     divisive architecture of
                      freqHz is

account signal: std_logic_vector (25 downto 0):

         "000000000000000,000,000,000,000"; begin

                  process (clock,

                  compt) begins

                           If clock'event and clock -- '1' then --counts with each

                                    clock shot counts std_logic_vector (unsigned

                                    (compt)");

                           end if;
```

```
                    if compt - "10111110101111110001001000000000000" then -- reset
                    once reached 50,000,000
(freq - 1Hz)

                            "00000000,000,000,000,000,000,";

                    end if;

                    If count -- "00000000010011000100100100000000" then -- lets
                    choose the report
cyclical 1 and then 0

                            '1';

                    Else

                            '0';

                    end if;

        end process;

end architecture;
```

Appendix 3-2: Delay:

```
entity delay is
        port (clock: in std_logic;
                    reset: in
                    std_logic;
                    output: out
                    std_logic);
end entity;


architecture waiting of delay is
counter signal: std_logic_vector (25 downto 0):
"00000000000000000,000,000,000"; begin
        process (clock,
        reset) begins
                If clock'event and clock '1' then
                        counter std_logic_vector (unsigned"");
                end if;
                If reset - '1' then
                        "0000000,000,000,000,000,000,000";
                end if;
                If counter 'lt; "" and counter "" then
```

```
                    '1';
                Else
                    '0';
                end
        if; end
        process;
end architecture;
```

Appendix 3-3: Freq170kHz:

```
entity freq170kHz is
        port (clock: in std_logic;
                    freq: out std_logic);
end entity;



architecture divider of freq170kHz is
account signal: std_logic_vector (25 downto 0):
        "000000000000000,000,000,000,000"; begin
                process (clock,
                compt) begin
                    If clock'event and clock -- '1' then --counts with each
                        clock shot counts std_logic_vector (unsigned
                        (compt)");
                    end if;
                    If compt - "00000000000010010010110" then -- reset once
                    reached 50
000,000 (freq - 1Hz)
                        "000000000,000,000,000000000000000";
                    end if;
                    If count -- "00000000000000001001011" then -- allows
you to choose the cyclical ratio 1 then 0
                        '1';
                    Else
                        '0';
                    end if;
        end process;
```

```
end architecture;
```

Appendix 3-4: compte_temp

```
entity compte_temps5 is
        port (clock_50: in std_logic;
                        clock: in
                        std_logic;
                        stop: in
                        std_logic;
                        time: out std_logic_vector (31
                        downto 0); alert: out
                        std_logic_vector (9 downto 0));
end entity;

architecture compt of compte_temps5 is
        constant zero: unsigned(31 downto 0):
        to_unsigned (0.32); counter signal: unsigned(31
        downto 0): zero;
        m_clock signal:
        std_logic; begin
                process (clock_50, clock, stop) --
                counter begins
                If clock_50'event and
                        clock_50'1' then m_clock
                        clock;
                        If clock'1' and m_clock'0' then
                                if stop ''0' then--we count
                                        counter 'lt;
                                        counter'1;
                                        If counter (3 downto 0) - "1001"
                                                then counter (3 downto 0)
                                                'lt;'000;;
```

```
                                    counter (7 downto 4) counter (7 downto
                                    4) - 1; If counter (7 downto 4) - "1001"
                                    then
                                        counter (7 downto 4) '0000';
                                        counter (11 downto 8) counter (11
downto 8) - 1;
```

```
                                        If counter (11 downto 8) - "1001"
                                            then counter (11 downto 8)
                                            'lt;' '0000';
                                            counter (15 downto 12)
counter (15 downto  12) - 1;
                                        end if;
                                    end if;
                                end if;
                            Else
                                 if not (counter -zero) then --we recover the value
only if it is not zero
                                    time std_logic_vector
                                    (counter); zero counter;
                                end if;
                            end if;
                        end
                        if; If
                        you
                        do
                    end if;
                    end
process; end
architecture;
```

Appendix 3-5: Strength

```
entity force is
        port (reset: in std_logic;
                        set: in std_logic;
                        clock_50: in
                        std_logic; result:
                        out std_logic);
end entity;

architecture cycle of
force is signal tempo:
std_logic; state type is
(e_on, e_off);
etat_present signal,
etat_suivant: state; begin
        process (clock_50 )-
        -heart begins
                If clock_50 event and
                        clock_50'1' then
                        etat_present
                        etat_suivant;
                end
        if; end
        process;

        process (set, reset, etat_present) --
        transition begins
                case
                etat_present is
                when e_on
```

```
                If reset ' '1' then etat_suivant e_off; else etat_suivant
        e_on; endif; when e_off
                if set '1' then etat_suivant e_on; else etat_suivant e_off; end
                if;
```

```
            end
        case; end
        process;

        process (etat_present) -
        -exits begin
            case
            etat_present is
            when e_on
                result '1';
            when e_off
                result '0';
            end case;
        end process;




end architecture;
```

Appendix 3-6: Force II

```
entity force2 is
        port (reset: in std_logic;
                    set: in std_logic;
                    clock_50: in
                    std_logic; result:
                    out std_logic);
end entity;

architecture cycle of
force2 is signal tempo:
std_logic;
state type is (e_on, e_off.e_wait);
etat_present signal,
etat_suivant: state; begin
```

process(clock_50)--heart

```
begin
        If clock_50 event and
                clock_50'1' then
                etat_present
                etat_suivant;
        end
if; end
process;

process (set, reset, etat_present) --
transition begins
        case
        etat_present is
        when e_on
                if set '1' then etat_suivant e_wait; else etat_suivant
        e_on; end if; when e_off
                If reset - '1' then etat_suivant e_on; else etat_suivant
        e_off; end if;  when e_wait
                if set '0' then etat_suivant e_off; else etat_suivant
        e_wait; end if; end case;
end process;

process (etat_present) -
-exits begin
        case
        etat_present is
        when e_on
                result '0';
        when e_off
                result '1';
        when e_wait
                result '0';
        end case;
end process;
```

```
end architecture;
```