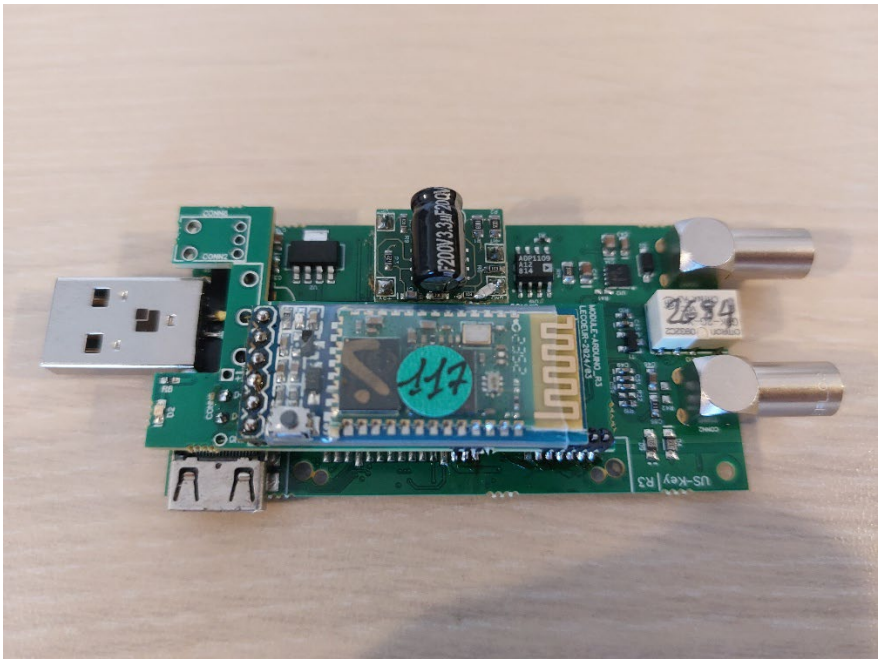
LECOEUR ELECTRONIQUE
300 Chemin des comtois
45220 CHUELLES - FRANCE -
Tel : +33 (0)2 38 94 28 30
Fax : +33 (0)2 38 94 29 67

mailto:info@lecoeur-electronique.com
http://www.lecoeur-electronique.net

Rédacteur	*M.COUPARD / S. BERTOLOTTO / JM.LECOEUR*
Date	*11/07/2025*

US-BUILDER

Révision 0.0 le 11/07/2025



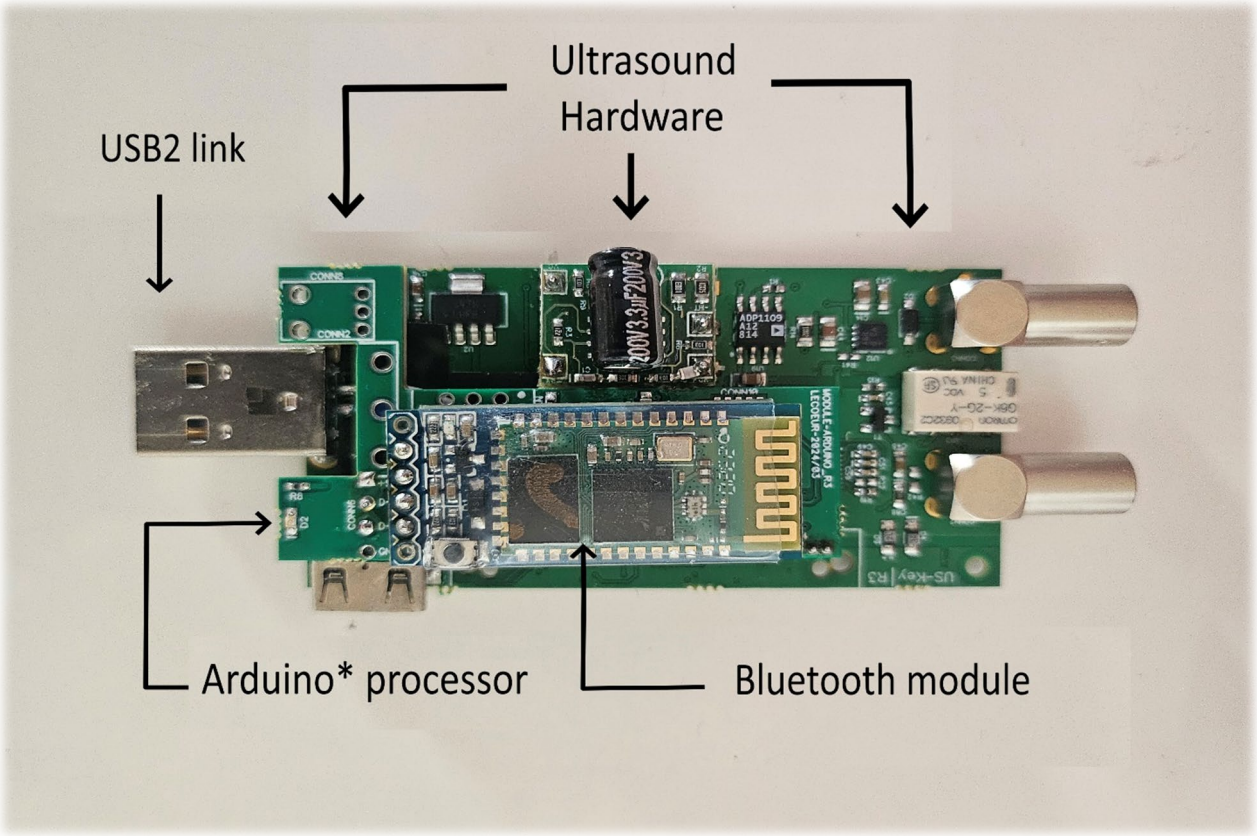
1 DESCRIPTION

US-BUILDER is an OPEN-SOURCE platform dedicated to designers. It is delivered with many software examples that can be modified or rewritten to fit your needs. It is supported by main OS: Windows / Android / Linux. Due to the Arduino processor, that control ultrasound hardware, the user can also write real time application in Arduino language (.ino).

US-BUILDER is the more recent concept and technology in ultrasound control. All electronics are on the board. (Transmitter, Amplifier, Digitizer, Dac, USB2 and Bluetooth link).

The transmitter can generate pulses with a voltage level and a width programmed by the user. A low noise preamplifier combined to a VGA gives a gain range between 0 and 80 dB, a DAC curve is also available. A 12 bits analog digital converter with a sampling frequency of 80 MHz is used to digitize ultrasound signals. The USB2 or Bluetooth link transfers, in real time, ultrasonic waveforms to computer (lap-top, tablet, portable pc)

2 ***HARDWARE***



Connect the USB2 link or the Bluetooth at your computer.
The device is see as a serial port of communication.

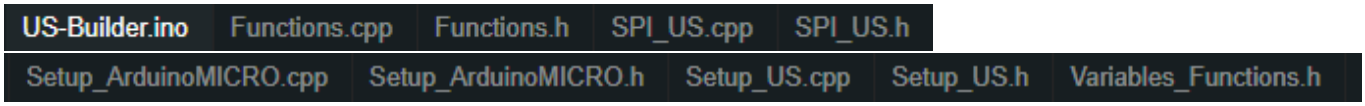
The Arduino processor make the communication between the computer and the Ultrasound Hardware.

You can use the Ultrasound Hardware with one sensor if you are in Pulse-Echo mode, or with two sensors fi you are in Pitch & Catch mode.
In Pulse-Echo : connect the sensor at the Lemo 00 on Top or down (see the picture on the top).
In Pitch & Catch : connect one sensor by Lemo 00, the transmitter is on a top and the receiver is on down (see the picture on the top).

The mini hdmi connector on the Ultrasound Hardware is useless.

3 ARDUINO FIRMWARE

The folder with the firmware has 10 files.



The main is « US-Builder.ino », open it with ARDUINO IDE, you can download this software on « https://www.arduino.cc/en/software/ »

3.1 US-Builder.ino

```
#include <SPI.h>
#include "Variables_Functions.h"
#include "Functions.h"
#include "Setup_ArduinoMICRO.h"
#include "Setup_US.h"
#include "SPI_US.h"

byte versionFW=1;


void setup()
  {
    Setup_ArduinoMICRO();
    delay(1000);
    Setup_US();
  }



void loop()
  {
    MANAGE_SPI();
  }

```

The of the firmware is very short.
In first, all library is call by « include » and we program the number of the version.

« void setup() » is the initialisation of the Arduino when it start :

- « Setup_ArduinoMICRO() » program all parameters of the Arduino to work. You can find the details of this function in « Setup_ArduinoMICRO.cpp » library.
- « delay(1000) » The Arduino waits 1000 milliseconds, the time for the ultrasound card to finish initializing.
- « Setup_US() » program the Ultrasound board with a default configuration. You can find the details of this function in « Setup_US.cpp » library.

« void loop() » is the operating loop of the device :

- « MANAGE_SPI() » is the main function that allows communication between the PC and the ultrasound board. This function is in « Functions.cpp ».

3.2 *Variables_Functions.h*

In this file is define all numbers of pin-out or SPI functions.

```
#define E_MOSI 2      // pin D2 for Enable_MOSI
#define E_MISO 3      // pin D3 for Enable_MISO

#define mabt 5        // pin D5 for Enable Bluetooth
#define analogPin A0  // Battery level

#define ADDR_Gain                   0
#define ADDR_Mode                   1
#define ADDR_Start                  2
#define ADDR_Compres                3
#define ADDR_NbSamples              4
#define ADDR_Delay                  5
#define ADDR_Voltage                6
#define ADDR_Width                  7
```

3.3 *Setup_ArduinoMICRO*

You can find the file in « .h » and « .cpp ».
The « .h » file resume the name of functions in « .cpp » file.
The « .cpp » file is the program contained in each of the functions summarized in the « .h » file.

« .h » file :

```

void Setup_ArduinoMICRO();

```

« .cpp » file :

```
#include <SPI.h>
#include "Variables_Functions.h"

void Setup_ArduinoMICRO()
{
Serial.begin(2000000);  // Serial at 2.000.000 bauds
pinMode(E_MOSI,OUTPUT); // Digital Port D2 used for Enable_MOSI
pinMode(E_MISO,OUTPUT); // Digital Port D3 used for Enable_MISO
pinMode(mabt,OUTPUT);   // Digital Port D5 used for Enable Bluetooth

digitalWrite(E_MOSI,1); // 1=Disable => See SPI_Mode1 => Enable_MOSI is active at 0 level
digitalWrite(E_MISO,1); // 1=Disable => See SPI_Mode1 => Enable_MISO is active at 0 level

digitalWrite(mabt,0);   // Bluetooth mode

SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE1));   // SCLK=1MHz ///// MSBF
SPI.begin();
Serial1.begin(115200);  // Serial at 115.200 bauds for BT
}
```

The « Setup_ArduinoMICRO » function initializes the serial port for USB communication, serial port 1 for Bluetooth communication, the SPI port for communication with the ultrasound board, and the Pin-Out for the SPI write and read Enable as well as the pin allowing to power or not the Bluetooth module (variable mabt).

3.4 Setup_US

You can find the file in « .h » and « .cpp ».
The « .h » file resume the name of functions in « .cpp » file.
The « .cpp » file is the program contained in each of the functions summarized in the « .h » file.

« .h » file :

```
1
2   void Setup_US();
3
```

« .cpp » file :

```
1   #include <SPI.h>
2   #include "Variables_Functions.h"
3   #include "SPI_US.h"
4
5   void Setup_US()
6   {
7     SEND_GAIN_dB(40.0);                          // Gain=40dB
8     SEND_COMPRESS_bin(0);                        // 0=> Compressor OFF
9     SEND_DELAY_us(0);                            // Delay=0us
10    SEND_VOLTAGE_Volt(50.0);                     // Volt=50
11    SEND_WIDTH_ns(55.5);                         // =SEND_WIDTH_bin(6)
12    SEND_PRF_kHz(0.1);                           // PRF=100Hz
13    SEND_MODE(0);                                // MODE=Pulse Echo
14    SEND_SCALE_us(51.0);                         // Scale=51us
15    SEND_FILTER_bin(BROADBAND);                  // Filter=BroadBand=No filter
16    SEND_SAMPLINGFREQ_bin(F_80MHz);              // Sampling frequency = 80MHz => resolution=12.5ns
17
18    SEND_ECHOSTARTPOSWIDTH_us(0,3.5);            // Echostart => Pos=0us / Width=3.5us
19    SEND_ECHOSTARTPOLARITY_pourcent(50);         // Echostart => Polarity +50%
20
21    SEND_GATE1POSWIDTH_us(7.1,3.5);              // Gate1 => Pos=7.1us / Width=3.5us
22    SEND_GATE2POSWIDTH_us(15.1,3.5);             // Gate2 => Pos=15.1us / Width=3.5us
23    SEND_GATE3POSWIDTH_us(23.5,3.5);             // Gate3 => Pos=23.5us / Width=3.5us
24    SEND_GATE1THRESHOLDALFILTER_pourcent(0,80.7); // Gate1 => AlFilter[7..0]=0 ! / Threshold=80.7%
25    SEND_GATE2THRESHOLDALFILTER_pourcent(0,60.7); // Gate2 => AlFilter[7..0]=0 ! / Threshold=60.7%
26    SEND_GATE3THRESHOLDALFILTER_pourcent(0,40.7); // Gate3 => AlFilter[7..0]=0 ! / Threshold=40.7%
27    SEND_ANALOGSETPOLARITY_bin(ANALOG_TOTAL,ANALOG_TOTAL,ANALOG_TOTAL,BOTH_WAVES);    // ANALOG1 = OFF /
28    SEND_ALDELAY_bin(0,0);              // Duration = 0 / All Alarms = 0 => On Appears
29
30    SEND_NBSAMPLES_bin(4090);          // Only used for AUTOMATIC A-scan storage => Unused, here
31    SEND_ONOFFDAC(1);                  // DAC OFF - Mode Read
32    SEND_READPORT(ASCAN_8bit);
33  }
34
```

This function programme all functions of the Ultrasound Board.
It's a default configuration. If you want, you can change the parameters to try or to match a configuration that suits you when starting the device.
You can find the value of « BROADBAND », « F_80MHZ », ect …, in « Variables_Functions.h » file, there are other variables in this file to change.

3.5 SPI_US

You can find the file in « .h » and « .cpp ».
The « .h » file resume the name of functions in « .cpp » file.
The « .cpp » file is the program contained in each of the functions summarized in the « .h » file.

« .h » file :

```
1
2   void SEND_BYTES(byte Len);
3   void ReadSamples(int passe,int Nb, byte type);
4
5   void SEND_GAIN_dB(double Gain_dB);
6   void SEND_GAIN_bin(unsigned int Gain_bin);
7
8   void SEND_COMPRESS_bin(byte Compress_bin);
9
10  void SEND_DELAY_us(double Delay_us);
11  void SEND_DELAY_bin(unsigned int Delay_bin);
12
13  void SEND_VOLTAGE_Volt(double Voltage_V);
14  void SEND_VOLTAGE_bin(byte Voltage_bin);
```

All communication functions with the Ultrasound Board are functions with the word « _bin » the data sent must be in binary.
For some functions, such as gain or scale, we have developed the function in the displayed value for easier programming (in dB, µs, Volts, etc…), and this function transforms this value into binary to send it to its function in « _bin » version to program the Ultrasound Board.

« .cpp » file :

```
53  void SEND_GAIN_bin(unsigned int Gain_bin)
54  {
55    byte Gain_LSB,Gain_MSB;
56
57    Gain_LSB=Gain_bin%256;
58    Gain_MSB=(Gain_bin-Gain_LSB)/256;
59    //Serial.print(String(Gain_dB)+"="+String(Gain_bin)+"="+String(Gain_LSB)+"="+String(Gain_MSB)+" / ");
60
61    digitalWrite(E_MOSI,0);
62    SPI.transfer(0);
63    SPI.transfer(Gain_MSB);
64    SPI.transfer(Gain_LSB);
65    SPI.transfer(ADDR_Gain);
66    digitalWrite(E_MOSI,1);
67  }
68
69
70  void SEND_GAIN_dB(double Gain_dB)
71  {
72      unsigned int Gain_bin;
73
74    Gain_bin=(10.108*Gain_dB)+65;
75    SEND_GAIN_bin(Gain_bin);
76  }
```

Above is the example of the « SEND_GAIN_dB » function which transforms the value into binary and transmits it to « SEND_GAIN_bin » which cuts the MSB and the LSB of the binary word then sends it to the Ultrasound Board by SPI.

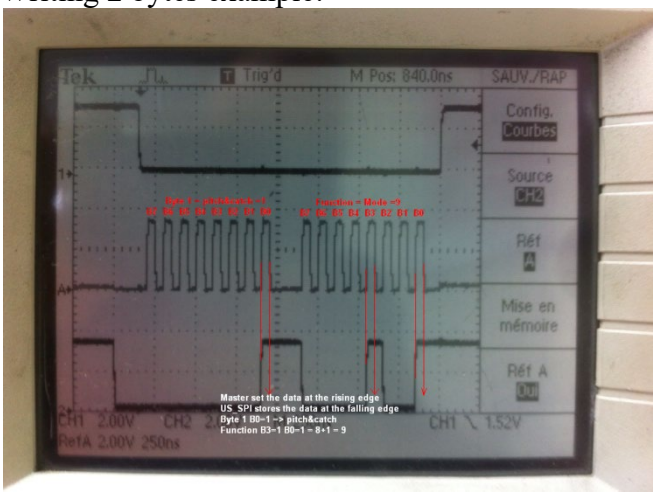
3.5.1 *SPI Configuration of Ultrasound Board*

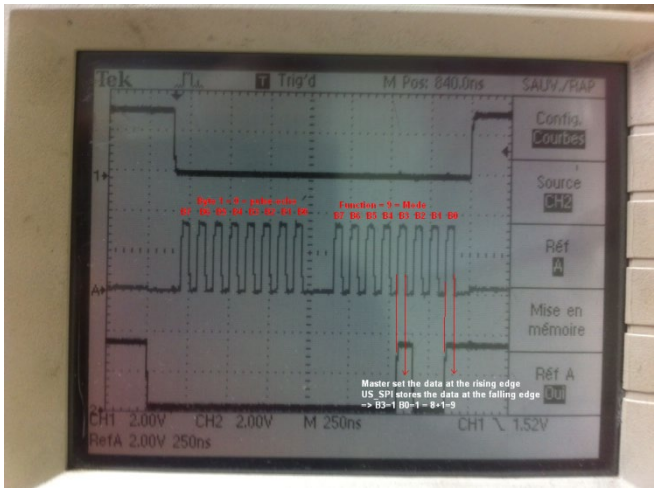
US_SPI uses 2 SPI lines: 1 for Writing and 1 for reading.
So, the number of wires necessary is 5:

- 2 for Writing (EnableMOSI/, DataMOSI)
- 2 for Reading (EnableMISO/, DataMISO)
- SCLK is common for the 2 ports at 10MHz.

Sleeping state for SCLK=0
Sleeping state for both Enable=1
DataMISO = Hi-Z when EnableMISO/=1

Writing 2 bytes example:





This example shows the function 9 because it needs only 1 byte DATA (0/1) and allows to ear the relay clicking when the program is ok

3.5.2 ***SPI Functions***

To program US-Key SPI it's necessary to send up to 5 byte.
1 Byte (the last one to send) is a function number that you want to program (CMDSPI).
Depends of this function, between 1 to 4 other byte should be sent.

CMDSPI		Byte 1	Byte 2	Byte 3	Byte 4
Value	Function				
0	Gain	Gain [7..0]	Gain[9..8]	Address[7..0]	-
1	-	-	-	-	-
2	Sampling request	-	-	-	-
3	Compression	Factor[7..0]	-	-	-
4	Automatic Sampling request	X LSB	X MSB	-	-
5	Scale delay	LSB	MSB	-	-
6	High voltage	Voltage [7..0]	-	-	-
7	Pulse width	Width[7..0]	-	-	-
8	PRF	PRF[7..0]	PRF[15..8]	PRF[19..16]	-
9	Mode	0/1	-	-	-
10	Scale	Scale [7..0]	Scale[15..8]	-	-
11	ONOFF DAC	Bit 0 : 0=ON /1=OFF!! Bit 1 : 0=Read/1=Write	-	-	-
12	Echo-start	Position[7..0]	Position[15..8]	Width[7..0]	Width[15..8]
13	Echo-start	Polarity [7..0]	-	-	-
14	Filter	Filter[2..0]	-	-	-
15	PosWidth G1	Position[7..0]	Position[15..8]	Width[7..0]	Width[15..8]
16	PosWidth G2	Position[7..0]	Position[15..8]	Width[7..0]	Width[15..8]
17	PosWidth G3	Position[7..0]	Position[15..8]	Width[7..0]	Width[15..8]
18	ThresholdAlFilter G1	AlFilter[7..0]	Threshold[7..0]	-	-
19	ThresholdAlFilter G2	AlFilter[7..0]	Threshold[7..0]	-	-
20	ThresholdAlFilter G3	AlFilter[7..0]	Threshold[7..0]	-	-
21	AlDelay	Duration[7..0]	Duration[15..8]	AlarmSet[2..0]	-
22	AnalogOutput	Analog1Set[1..0]	Analog2Set[1..0]	Analog3Set[1..0]	Polarity[1..0]
23	ReadingPortFunction	Port[2..0]	-	-	-
24	Sampling Freq	Sampling[1..0]	-	-	-

3.5.3 SPI Detailed Functions

Fct.0 Gain :

The Gain value is only on 10bit → Gain[9..0]
You must convert Gain in dB to digital value with the formula: Gain[9..0] = 10*Gain(dB) + 65
For example 0 dB = 65 and 80dB=865

When DAC is OFF, Address[7..0]=0

See below DAC function description (§5.11)

Fct.1 Not used

Fct.2 Sampling Request :

When you send CMDSPI=2, the US-Key will store the current A-scan inside its FIFO (4k*12bit)

Fct.3 Compression :

No compression=0
Compression=1, the number of samples is divided by 2 because US_SPI return the tallest echo between 2
Compression=2, the number of samples is divided by 3 because US_SPI return the tallest echo between 3
Compression=3, the number of samples is divided by 4 because US_SPI return the tallest echo between 4
Etc.

Note:

- Compression is available only if Filter=4=NoFilter AND SamplingFreq=1=80MHz

Fct.4 Automatic Sampling Request =Nb samples:

When you will read the Xth samples, the US-Key will store AUTOMATICALLY the current A-scan inside its FIFO. It's faster than recall Fct2 and read again.

Fct.5 Scale Delay :

Scale delay in step of 25ns !

Fct.6 Voltage :

Voltage[7..0]=(98/180)*X+81.777 230V<X<10V

Fct.7 Pulse width :

Pulse width by step of 6.25ns. If value=0 then width=18ns up to 18+(255*6.25)=1600ns

Fct.8 Pulse Repetition Frequency :

PRF=Pulse Repetition Frequency → Period in step of 25ns !

Fct.9 Mode :

Mode 0=Pulse Echo / 1 = pitch & catch

Fct.10 Scale :

Scale of the A-scan in step of 25ns !
Note: To reduce electronic noise, the high voltage converter is switch off during Scale and restart after

*Fct.11 **DAC :***

The DAC function is a memory of 256 Gain values read every 650ns.
When the DAC is ON (=0!) this memory is read and the gain is modified from the emission or the interface echo when Echo-start is ON, to 256*650ns=166µs

When the DAC is OFF (=1!), it's not necessary to program Address value in function Gain (Fct 0)

To Program the DAC function :

1- Send Fct 11 = 3 to switch the DAC off and set the memory mode to WRITE
   (3 = Bit0=1=DAC OFF, Bit1=1=Mode Write)
2- Send Fct 0 with 256 Gain[9..0] values and 0<=Address[7..0]<=255 (For loop)
3- Send Fct 11 = 0 to switch the DAC on and set the memory mode to READ
   (0 = Bit0=0=DAC ON, Bit1=0=Mode Read)

To stop the DAC function:

1- Send Fct 11 = 1 to switch the DAC off and let the memory mode to READ
   (1 = Bit0=1=DAC OFF, Bit1=0=Mode Read)
1- RECALL Fct 0 to reprogram the desired global Gain

Note:

- When the DAC is ON, the US_SPI read the complete 256 Gain values memory, so you must program these 256 gain values
- When you already programmed the 256 values, it's possible to reprogram only a part of them

*Fct.12 **Echo-Start :***

Position[15..0] and width[15..0]by step of 25ns !

*Fct.13 **Echo-Start :***

Polarity [7..0] !!! = Echo-start threshold on 8bit.
You must convert Polarity in % to digital value with the formula: Polarity[7..0] = 1.27*Polarity(%) + 128
For example you want set the threshold at -40% → 77
For example you want set the threshold at +50% → 192

Note:
When you have a water path, it's possible to resynchronize the entire time base and gate measurement on the moving interface echo by using Echo-start function.
In this case, program an area where the interface echo should be detected. The beginning of this area = Echo-start position and the duration = Echo-start width.
If Echo-start position<>0 the function is On
If Echo-start position==0 the function is Off

When it's on, the 1st echo over the threshold will resynchronize the complete time base, scale, delay AND gates positions.

Fct.14 Filters :

Filter → 0=1.25MHz, 1=2.5MHz, 2=5MHz, 3=10MHz, 4=No filter

Note:

- Filters are only available at 80MHz sampling frequency
- They are based on FIR

Fct.15 Gate 1 position and width :
Fct.16 Gate 2 position and width :
Fct.17 Gate 2 position and width :

Hardware Gate measurement step of 25ns: Position[15..0]/Width[15..0]

Fct.18 Gate 1 threshold alarm filter :
Fct.19 Gate 2 threshold and alarm filter :
Fct.20 Gate 3 threshold and alarm filter :

Hardware Gate measurement: Threshold on 8bit=0..255=0..100% AlFilter[7..0]=0 !

Fct.21 AlDelay :

Duration[15..0] step of 800ns for Alarm and Analog output duration
AlarmSet[0]=AlG1, AlarmSet[1]=AlG2, AlarmSet[2]=AlG3 : 0=Alarm on appearance 1=Alarm on Disappearence

Fct.22 AnalogOutput :

AnalogχSet 0=OFF 1=TotalAmplitude 2=AmplitudeOverThreshold
Polarity[1..0] 0=Positive&Negative 1=Negative 2=Positive

Fct.23 ReadingPortFucntion :

ReadingPortFucntion[2..0] : 0=Ascan 8bit / 1=HardwareGateMeasurement / 2=AscanLSB

Note:

- For 8bit A-scan, the 3 1st byte are 10,10,1 else the A-scan is wrong
- HardwareGateMeasurement is a 16bytes frame : AmplG1, AmplG2, AmplG3, DistPeakG1LSB, DistPeakG1MSB, DistPeakG2LSB, DistPeakG2MSB, DistPeakG3LSB, DistPeakG3MSB, Alarm [2..0] , DistEdgeG1LSB, DistEdgeG1MSB, DistEdgeG2LSB, DistEdgeG2MSB, DistEdgeG3LSB, DistEdgeG3MSB

- Alarm[0]=G1, Alarm[1]=G2, Alarm[2]=G3, Alarm[3]=Echo-start

- Distance = step of current sampling frequency

- If ReadingPortFucntion[2..0]=0 The master will read an 8bit Ascan
- To Read an 12bit A-scan :
    - Set ReadingPortFucntion[2..0]=0 to read A-scan[11..4] !!!!
    - Set ReadingPortFucntion[2..0]=2 to read A-scan[3..0] !!!!

See 'A-scan 12Bit reading

Fct.24 SamplingFreq :

Sampling Freq: 0=160MHz / 1=80MHz / 2=40MHz/ 3=20MHz

Note:

- Sampling Frequencies are available only if Filter=4=NoFilter AND compression=0

3.6 Functions

You can find the file in « .h » and « .cpp ».
The « .h » file resume the name of functions in « .cpp » file.
The « .cpp » file is the program contained in each of the functions summarized in the « .h » file.

« .h » file :

```

void MANAGE_SPI();

```

« .cpp » file :

```
#include <SPI.h>
#include "Variables_Functions.h"
#include "SPI_US.h"

extern byte buffer[10];
extern byte versionFW;
byte BlueTooth;

void MANAGE_SPI()
{

  byte tab[12];
  byte tab1[512];

  if (Serial.available()) // --------------- USB
    {
      digitalWrite(mabt,1);
      BlueTooth=0;
      for (int i=0;i<12;i++)
        {
          tab[i]=Serial.read();
        }
      if (tab [0] ==140 and tab [1] ==140 and tab [2] ==140 and tab [3] ==140 and tab [4] ==140)
       {
        if (tab[5]==1)                          //ecriture
            {
              digitalWrite(E_MOSI,0);
              for (int i=0;i<tab[6];i++)
              {
              SPI.transfer(tab[i+7]);
              }
              digitalWrite(E_MOSI,1);
            }
        if (tab[5]==0)                        //lecture
            {
```

The « MANAGE_SPI » function allows communication between the PC, via USB or Bluetooth, and the Ultrasound Board, via SPI.
A header is provided for decoding.

Communication is as follows :

An array of 12 data points.

-tab[0 to 4]: security header => 140/140/140/140/140/
-tab[5]: Arduino functions => 0 or 1 or 3

***If tab[5]=3:**

The Arduino firmware version is requested. It will respond with its current version, which is 1, and the Arduino ignores the values in the rest of the array.

***If tab[5]=1:**

The Arduino will program the ultrasonic functions according to the following values in the table. Full details of each function are in the firmware file "SPI_US.cpp."
-tab[6]: Number of parameters to be sent. See chapter "3.5 SPI_US " for details
-tab[7 to 12]: Values to be programmed and function number.

*Example when programming the filter:

-tab[0 to 4]=140/140/140/140/140 *header*
-tab[5]=1 *ultrasound programming mode*
-tab[6]=2 *filter value to program + filter function number*
-tab[7]=4 *Broadband mode*
-tab[8]=14 *filter function number*

***If tab[5]=0:**

The Arduino will request the ultrasound board to read the ASCAN. The request must be completed with the number of points to be read in 8-bit words. Therefore, it must be split in half to include the most significant word (MSB) and the least significant word (LSB).
-tab[6]=MSB
-tab[7]=LSB
The Arduino then retrieves the ASCAN points and sends them directly to the PC.

Special case:

If you make a read request but have an error in one of the header values, the device will return an array of the requested number of points, all points will have the value « 50 ».

4 EXAMPLES

4.1 Python examples:

All examples work with Python v3.
To run the scripts, you need the following libraries: matplotlib, pyqt, pyserial and numpy.

There are 3 examples :

- « US-Builder A-scan V3.py » : Requests an 8-bit A-scan from US-Builder and then displays the result in a graph.
- « US-Builder A-scan12bit V0.py » : Requests an 12-bit A-scan from US-Builder and then displays the result in a graph.
- « US-Builder Complete V1.py » : Presentation of a complete programming of the device with an ASCAN query and display of the result in a graph.

Start of the « US-Builder A-scan V3.py » script :

```
from pylab import *
from serial import *
from ctypes import *
from matplotlib import pyplot as plt
import ctypes
import time
import pyqtgraph as pg

Port="COM6"
Header=[140,140,140,140,140]        # Lenght=5
Write=ushort(1)
Read=ushort(0)
FirmVer=ushort(3)
#                  [R/W , NbByte for SPI, 5 bytes for SPI part
SerialFrame=Header+[ 0  ,       0       , 0,0,0,0,0]  # SerialFrame[5]=R/W [6]=NbByte [7/8/9/10/11]=SPI values


with Serial(port=Port, baudrate=9600, timeout=1, writeTimeout=1) as US_Builder:
    print ("Serial ",Port," Open")
    print()

    SerialFrame[5]=FirmVer
    SerialFrame[6]=ushort(0)
    SerialFrame[7]=ushort(1)
    print("Firm.Ver. Reading : *** Serial Frame=",SerialFrame[0:8])
    US_Builder.write(bytes(SerialFrame[0:8]))
    print ("Arduino Firm. Ver. :",int(US_Builder.read(1).hex()))
    print()

    SerialFrame[5]=Write
    SerialFrame[6]=ushort(2) # NbByte to write for SPI values
    SerialFrame[7]=ushort(0) # 0=8bit A-scan 1=measurement 2=12bit A-scan (LSB)
    SerialFrame[8]=ushort(23) ############################## Fct 23 = ReadingPort
    US_Builder.write(bytes(SerialFrame[0:9]))
```

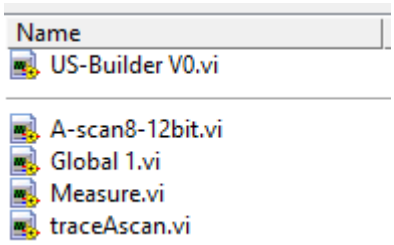
Before start the script, you must define the Variable « Port » in the script !

```
Port="COM4"
Header=[140,140,140,140,140]        # Lenght=5
Write=ushort(1)
Read=ushort(0)
```

If you are on Windows, it's « COM? ».
If you are on Linux, it's « /dev/ttyACM? »
Ect …
? = The number of the port to which US-Builder is connected.

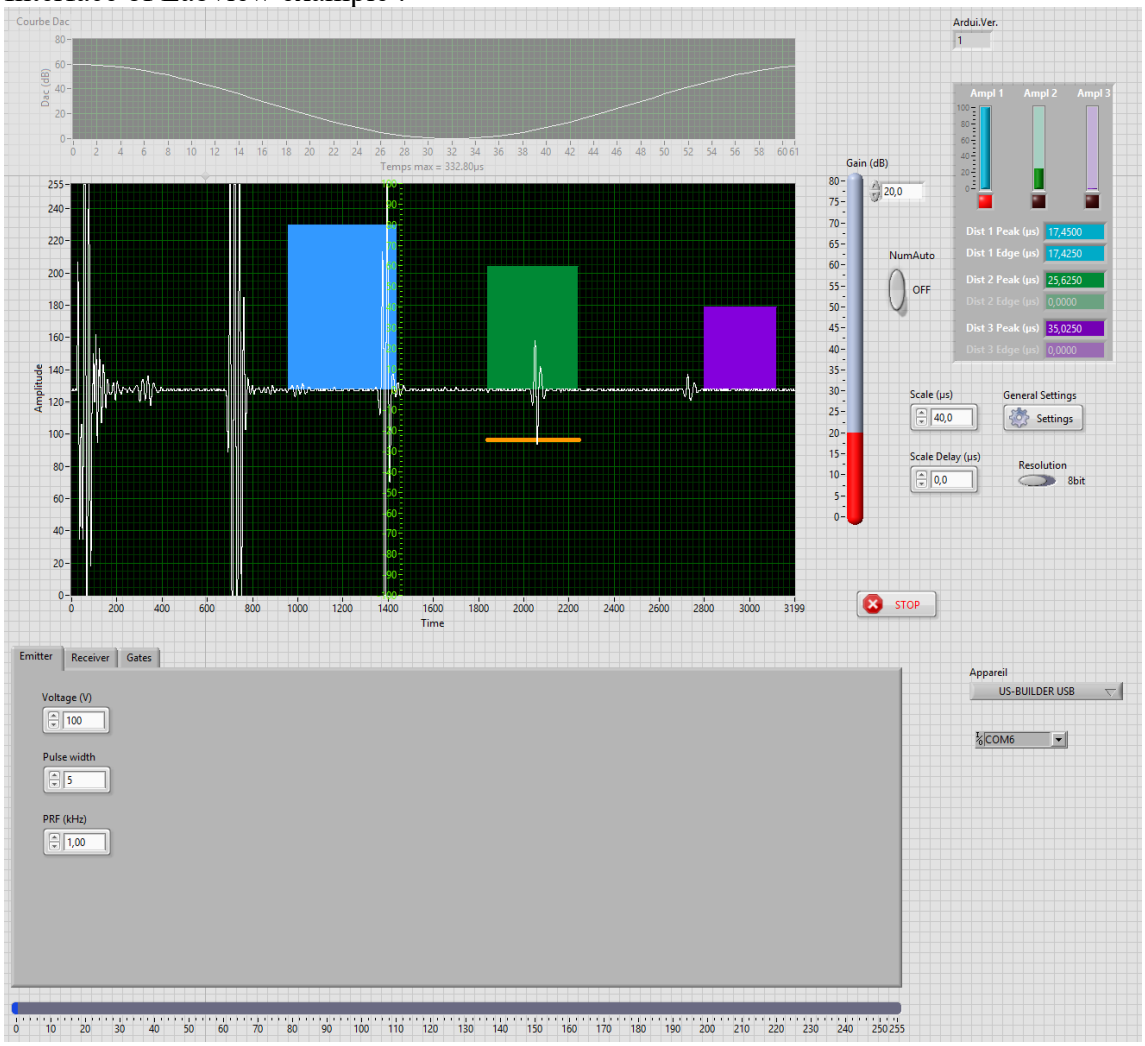
4.2 Labview example :

This example is create on Labview 2011 and all VI is in « US-Builder.llb ».
In this llb there is 5 VI, one of which is the main one.

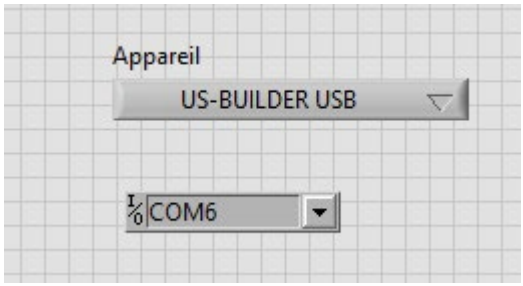


- « US-Builder V0.vi » is the main operating VI.
- « A-scan8-12bit.vi » allows you to manage the ASCAN request and retrieval if the option is requested in the « US-Builder VO.vi ».
- « Global 1.vi » is a vi that stores only the beginnin of the code table to be US-Builder : header + write or read bit.
- « Measure.vi » uses function 23 « ReadingPortFunction », It requests the value « 1 » so that the ultrasound board sends all measured values to the measurement ports according to the settings made in the port.
- « traceAscan.vi » after the ASCAN points are retrivied, this vi formats the table to display it in the graph and then in the main vi « US-Builder.llb ».
-

Interface of Labview example :



Before start the script, you must define the Variable « Port » in the script !



*Details about the graph :
-« Time » is the number of points of the ascan read, the maximum number of readable points is 4000.
-« Amplitude » is the value of each of the points.
=> Amplitude of 8-bit ASCAN is 0 to 255 and 128 is the middle point
=> Amplitude of 12-bit ASCAN is 0 to 4095 and 2048 is the middle point

The time beween two points depent of the Sampling Frequency
-160MHz=0.625ns
-80MHz=1.25ns
-40MHz=2.5ns
-20MHz=5ns

Scale delay is the total scale displayed in the graph, it depends on the number of points requested (knowing that a maximum of 4000 can be read) and on Sampling Frequency.
-(number of points) x (sampling frequency) = scale delay
-4000x1.25ns=50µs

When the DAC is « ON », the default curve apply is the curve on the top of the Ascan graph, it's a Gain curve.
When the DAC is « OFF », the Gain is the same for all points.

*Details about the gates :

-Amplitude : This is the percentage value of the highest peak present in the measurement gate. The green percentage graduation (from -100% to +100%) is present in the middle of the graph to help you find your way. The red light under each amplitude is the alarm, it is activated according to the parameter you have chosen, by default it is in « ONn appears ».

-Dist. Peak : measurement in µs of the emission up to the highest measurement point present in the measurement gate.

-Dist. Edge : measurement in µs from the emission to the first edge passing the gate measurement threshold.