WATERFORD INSTITUTE OF TECHNOLOGY

EMBEDDED PROJECT APPLICATION

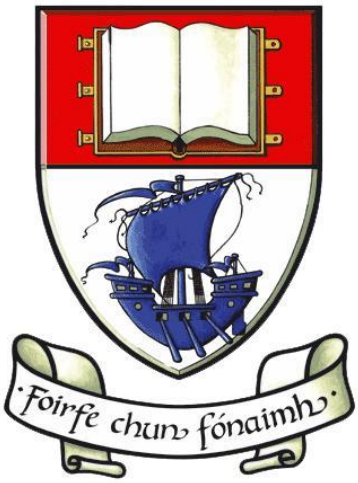
SOUND EFFECTS SYSTEM (SES)

Carlos Durango

Student ID: 20069162

Beng Electronic Engineering

2015



INDEX

1. INTRODUCTION

This document is a report for the project “Sound Effects Systems” developed as an embedded project on the module “Embedded Project Application” studied in the third year of the Bachelor in Electronic Engineering at Waterford Institute of Technology.

The subject uses a real project already built to improve it and creates new applications every year. Each one of the students has a different embedded project to work on in order to make them all work together by the end of the semester. All the projects have been developed individually by the students helped by the lecturers Jason T. Berry, PJ Walsh and Paddy Murray.

In this case, the project “Sound Effects Systems” has been developed by Carlos Durango Labrador during the second semester of the course 2014/2015. Basic concepts about Can Bus Networking and how the used base boards work have been learnt on different subjects as “Embedded Software & RTOS”, so this project works as a final application of the knowledge studied in other subjects.

All the base boards working on the project have been connected by Can Bus Networking. Can Bus network is one of the best ways to interact between different parts of the same project. BENGiE is an embedded project in which several different small projects have to interact sending and receiving messages between each other through a Can Bus Network.

1.1. Sound Effects System Concept

The Sound Effects System is a new application for the embedded project on BENGiE Robot. The basic concept for this project has always been to provide BENGiE with can-activated sound effects, so the final appearance of the robot could be friendlier and the sounds would give a more realistic experience on the interaction with it on its many different functions.

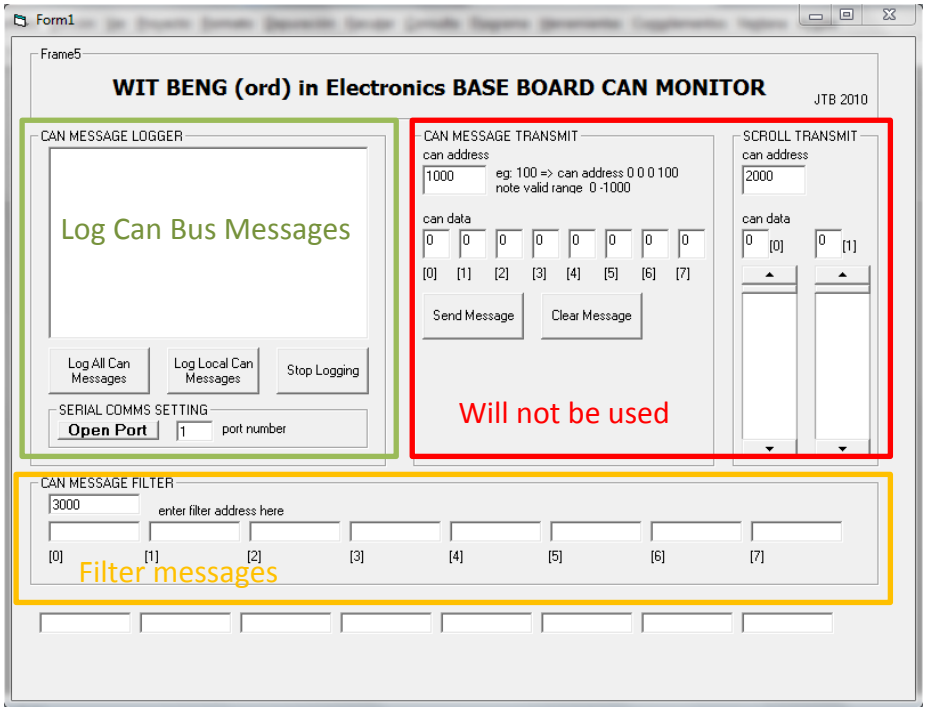
In other words, the Sound Server is able to read all the messages through the can bus Network on the robot, filter them, and play different sounds depending on the received messages. Each one of the connected boards can enable and disable the several operation modes on that sound server and make it play what it wants to play.

1.2. Developing the Project

As it will be shown in the following sections of the report, the project is based on an interface already programmed in Visual Basic 6 language. Many changes through the last months programming have given the final result to the interface, but also two different versions of the project that will be explained below.

2. Review of the Existing Software

The Sound Server is based on an interface already programmed at WIT. The Base Board Can Monitor allows the user to see all the messages going through the network and filter them. Once the user has configured the port which the board is attached to, all the messages travelling around the network will be available on the "Can Message Logger" text-box.



The filter boxes below will filter the messages to the address we are interested on check, the sound server address in this case, so it will use that messages to play the sounds. That means the sound server could play [6x255=] 1530 different sounds –the last byte is used as a space to set up a timer in operation mode2 or to stop the message in mode1, explained in following sections.

The rest of the interface of that Can Monitor will not be used. It does not need to transmit any messages, just need to read from the network and play the sounds that the rest of the system asks for playing. We can define the Sound Server Software as a "*slave*" software that executes orders from others.

The basic logic for allowing the connections of that interface to the network was already written. That is a huge advantage because it gives the programmer more time to improve the concepts on his real project; allow the robot to play sounds.

3. First Steps

3.1. Understanding Can Monitor

3.1.1. Variable Declaration

The variables used by this interface are declared at the beginning of the code, on a general purpose space. Also special functions are declared here on this section of the code, as we will see on following explanations.

The screenshot below contains all the declarations as integers or strings used on the first version of the software.

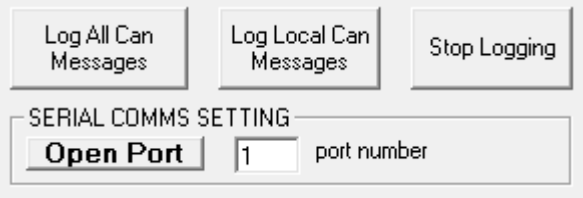
```
'_________________________Variable declaration_____________________________'
Dim serial_received As String
Dim can_address_string As String
Dim portnumber As Integer

Dim can_address_number0 As Integer
Dim can_address_number1 As Integer
Dim can_address_number2 As Integer
Dim can_address_number3 As Integer

Dim can_data_string As String
Dim can_data_number0 As Integer
Dim can_data_number1 As Integer
Dim can_data_number2 As Integer
Dim can_data_number3 As Integer
Dim can_data_number4 As Integer
Dim can_data_number5 As Integer
Dim can_data_number6 As Integer
Dim can_data_number7 As Integer

Dim message_counter As Integer
Dim time_stamp_instr As String

Dim decsec As Integer 'miliseconds
Dim counter As Integer 'Counter
Dim readingcounter As Integer 'Counter while reading
Dim timer3control1 As Integer 'First byte controls the timer in mode 3
Dim timer3control2 As Integer 'Second byte controls the timer in mode 3
Dim soundOff As Integer       'MODE 3: TRUE if sound is not being played
```

Most of them were already used by the can bus monitor, although the last ones written on the code are used to programme the counters in different modes of operation that will be explained later.

All of the declarations create variable that will be changing continuously as the can bus monitor receives new messages from the network. They will store values of the messages, addresses or essential information for the connection. Also a counter is declared to work on a loop.

3.1.2. Can Bus Monitor

The Sound Effects System is completely based on this software already programmed, although it does not use all its functions. Indeed, it only uses the functions that allow the computer to read the messages sent through the can bus network on which the base board is attached to.



One of the main functions written in this piece of software is able to open the COM-PORT needed in each occasion. By pressing the button "*Open Port"* after entering the number of the port in which the base board has been connected, the program will start reading the messages available in that port. The picture below shows the code attached to the command button and the operation working when is pressed.

```
Private Sub Command1_Click()
'open comm port selected in port
'///////////////
MSComm1.CommPort = Val(Text30.Text)
MSComm1.PortOpen = True
End Sub
```

Next interesting function is the one that allow us to show all the messages in the text box on the left of the screen. That button simple send to the board the message "*crx:2*" that we already known for being the code to start reading messages. The command "*+ Chr$(13)*" adds "*enter*" on the message to send it.

```
Private Sub Command3_Click()
'enable logging all messages
MSComm1.Output = "crx:2" + Chr$(13)
End Sub
```

We can also stop messages as we do on *HyperTerminal* using "*crx:1"* or end the logging with the command *"crx:0"*. There are to buttons allowing that functions.

```
Private Sub Command4_Click()
'enable logging local message
MSComm1.Output = "crx:1" + Chr$(13)
End Sub
Private Sub Command5_Click()
'Disable logging
MSComm1.Output = "crx:0" + Chr$(13)
End Sub
```

Once the connection has been established through the serial port, the program uses a timer that check if any messages have been received every 10ms. The timer uses several conditions to know if the message has arrived to the port so it would save it in different variables.

```
Private Sub Timer1_Timer()
'open comm port
'///////////////
If MSComm1.InBufferCount > 0 Then 'if data available
 serial_received = MSComm1.Input
  If serial_received = "c" Then
   serial_received = MSComm1.Input
    If serial_received = "a" Then
     serial_received = MSComm1.Input
      If serial_received = "d" Then
       serial_received = MSComm1.Input
        If serial_received = ":" Then
          serial_received = MSComm1.Input
           If serial_received = " " Then
              'valid header cad: read can address byte 0
              can_address_string = MSComm1.Input + MSComm1.Input + MSComm1.Input
              can_address_number0 = Val(can_address_string)
              can_address_string = MSComm1.Input + MSComm1.Input + MSComm1.Input + MSComm1.Input
              can_address_number1 = Val(can_address_string)
              can_address_string = MSComm1.Input + MSComm1.Input + MSComm1.Input + MSComm1.Input
              can_address_number2 = Val(can_address_string)
              can_address_string = MSComm1.Input + MSComm1.Input + MSComm1.Input + MSComm1.Input
              can_address_number3 = Val(can_address_string)
```

The picture above checks the address of the receiving board and saves. Then it continues checking the message. As shown in the picture below, it will look into each one of the eight bytes of the message, save its value and also so it in the text boxes on the bottom of the interface –only data 0 is shown on this picture.

```
'read in cmd:
serial_received = MSComm1.Input + MSComm1.Input + MSComm1.Input + MSComm1.Input + MSComm1.Input
'List1.AddItem (serial_received)
If serial_received = " cmd:" Then
  'read in can data 0
  can_data_string = MSComm1.Input + MSComm1.Input + MSComm1.Input + MSComm1.Input
  can_data_number0 = Val(can_data_string)
  Text1.Text = can_data_number0
```

The last condition on this timer is used to display all the messages checked by the interface in the big text box showed on the screen.

```
'display data
 If message_counter = 13 Then
  List1.Clear
  message_counter = 1
  time_stamp_instr = "CAD: " & can_address_number0 & ":" & " CMD " _
  & Text1 & " " & Text2 & " " & Text3 & " " & Text4 & " " & Text5 & " " _
  & Text6 & " " & Text7 & " " & Text8 & " " & "    ::" & Hour(Time) & ":" _
  & Minute(Now) & ":" & Second(Now)
  List1.AddItem time_stamp_instr, 0
 Else
  time_stamp_instr = "CAD: " & can_address_number0 & ":" & " CMD " _
  & Text1 & " " & Text2 & " " & Text3 & " " & Text4 & " " & Text5 & " " _
  & Text6 & " " & Text7 & " " & Text8 & " " & "    ::" & Hour(Time) & ":" _
  & Minute(Now) & ":" & Second(Now)
  List1.AddItem time_stamp_instr, 0
  message_counter = message_counter + 1
 End If
```

Besides watching all the messages on the big text box, one of the most useful functions for the Sound Effects System is the possibility of filtering the messages sent to a specific address. That is how the new system will play the sounds sent to its own address. This filter is also set up on the timer, so every time a message arrives, the system checks if it is a message for the address we are interested on.

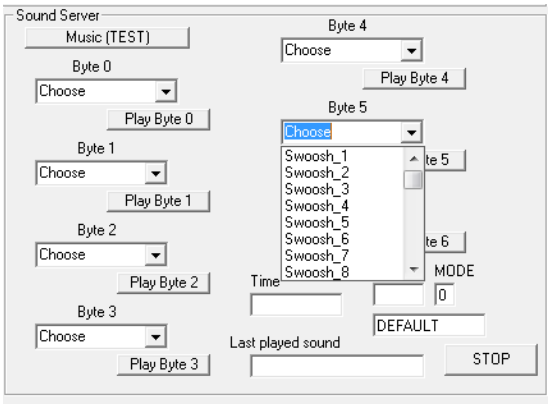
```
'_____________________________FILTER ADDRESS_____________________________'
'filter code check current address against filter address in text box 12
    If can_address_number0 = Val(Text21.Text) Then
      Text50.Text = can_data_number0
      Text51.Text = can_data_number1
      Text52.Text = can_data_number2
      Text53.Text = can_data_number3
      Text54.Text = can_data_number4
      Text55.Text = can_data_number5
      Text56.Text = can_data_number6
      Text57.Text = can_data_number7
    End If
```

This filter allows the user to programme several different sounds depending on the received message. In the following sections it will be explained how the storage and selection of the sounds work, as well as the precautions we must take to add new sounds.

3.2. Sound Storage

The interface gives the user the chance to try all the sounds available on the system. Although the next section will explain how these sounds are played, now we will explain how those sounds are stored and how to show all of them on the interface.

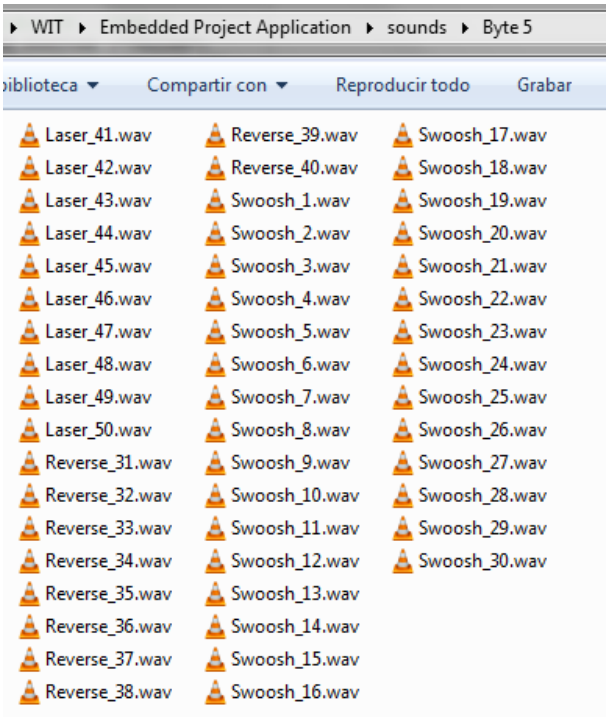
First of all, the picture below shows the design of the interface for the first version of the system –this design is slightly different on the next version, as it will be explained later. As the picture shows we can choose different sounds in 7 "*list-boxes*" and play each one of them. We can store up to 255 sounds in each section (highest message that one byte can receive), and the name of the sound on the folder have to match with the name on theses boxes. We will take *"Byte 5*" as an example to explain the rest of them.



All the names showed on the different *list-boxes* must be defined on the "*Form_Load*" code (general purpose section of the code). The easiest way to store all the sounds and make easier to play them is creating "groups" with similar sounds. In this case, the same byte has 3 different kinds of sounds; "*Swoosh*" from message equal 1 to 30, "*Reverse*" from message equal 31 to 40 and "*Laser*" from message equal 41 to 50.

The picture on the right is a screenshot of the folder that stores all the sounds related with this byte. The names match with the names we see in the interface, so it is taken as an advantage to play them.

```
'ComboBox #6 - BYTE 5
Combo6.AddItem ("Choose")
Combo6.AddItem ("Swoosh_1")
Combo6.AddItem ("Swoosh_2")
Combo6.AddItem ("Swoosh_3")
Combo6.AddItem ("Swoosh_4")
Combo6.AddItem ("Swoosh_5")
Combo6.AddItem ("Swoosh_6")
_
_
_
Combo6.AddItem ("Reverse_31")
Combo6.AddItem ("Reverse_32")
Combo6.AddItem ("Reverse_33")
Combo6.AddItem ("Reverse_34")
Combo6.AddItem ("Reverse_35")
Combo6.AddItem ("Reverse_36")
Combo6.AddItem ("Reverse_37")
Combo6.AddItem ("Reverse_38")
Combo6.AddItem ("Reverse_39")
Combo6.AddItem ("Reverse_40")
Combo6.AddItem ("Laser_41")
Combo6.AddItem ("Laser_42")
Combo6.AddItem ("Laser_43")
Combo6.AddItem ("Laser_44")
Combo6.AddItem ("Laser_45")
Combo6.AddItem ("Laser_46")
Combo6.AddItem ("Laser_47")
Combo6.AddItem ("Laser_48")
Combo6.AddItem ("Laser_49")
Combo6.AddItem ("Laser_50")
```



If in any case it is needed to add more sounds there are four things to keep in mind:

- **Location of the sound**: it has to be stored on the folder of the byte that will receive the message to play it.
- **Name of the sound**: the name must be the same on the list-box of the interface (defined on the *Form_Load* code) and on the file at the folder.
- **Format of the sound**: the only extension able to be played with VB6 is "*.wav*", but also the bit-rate cannot be too big.
- **Condition on the code**: as it will be explained, when a message arrived to a box on the filter, it will check the received number and filter it through some conditions to play the right one. We must add more conditions if we add more sounds.

3.3. How to Play Sounds in VB6

A new function declaration is needed in order to play the sounds in the code. There are actually two different ways to play sounds on VB6; the first one simply adds a *WindowsMediaPlayer* component, while the second one uses a function to play, stop, and modify sounds. The first mode is that simple that can't stop the sound; neither does more things than play it. Instead of using this simple component, the following function found on the internet also allows to handle the sounds while playing.

```
'____________________PlaySound Function Declaration_____________________'
'                                                                       '
'How it works:                                                          '
' Declare the main needed function to play sounds and music,as well as  '
' different options about it.                                           '
' This is only needed if we use the 'long method' for playing sounds    '
'                                                                       '
' Option Explicit
Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" _
    (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

Const SND_SYNC = &H0
Const SND_ASYNC = &H1
Const SND_NODEFAULT = &H2
Const SND_LOOP = &H8
Const SND_NOSTOP = &H10
```

Each one of the *List-boxes* where we can find the different sounds to play has a button close to it that says "*Play Byte #*". When we choose a sound of the list and press that button we can test the sound and listen to it. Using Byte 5 as example again, the following code shows how this buttons work and what they do:

```
'_____________________________BYTE 5 SOUNDS_____________________________'

Private Sub Command15_Click()
        'FileAddress = "G:\sounds\Byte 5\" + Combo6.Text + ".wav"
        FileAddress = "C:\                    \sounds\Byte 5\" + Combo6.Text + ".wav"
        soundfile$ = FileAddress
        wFlags% = SND_ASYNC Or SND_NODEFAULT
        sound = sndPlaySound(soundfile$, wFlags%)
        Text31.Text = Combo6.Text

        If Text23.Text = 2 Then
            counter = 0
            Timer2.Enabled = True
        End If
End Sub
```

The red box will be changed for the address of the folder where the sounds of this byte are located. This address on BENGiE is:

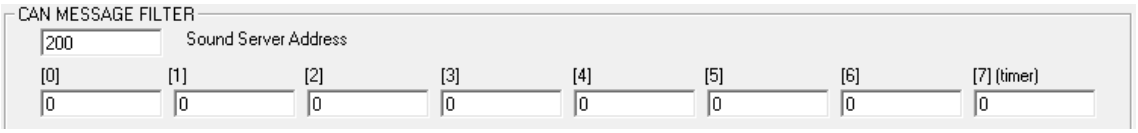
```
FileAddress = "C:\Users\Electronics\Desktop\SOUND SERVER\sounds\Byte 5\" + Combo6.Text + ".wav"
```

The last portion of the line adds to the address the name of the file and the extension. That is why naming alike the file and the text on the list is very important in order to complete the address on the simplest way possible.

Also that section of the code changes the text box where we can play the last sound played on the program. The last lines on that section are used for a special mode of operation which will be explained in following sections.

3.4. Filter the Message and Play the Sound

The filter will be used with the address 200 (Sounds Server Address) most of the times, but the address to be filtered can be changed. Each one of the boxes below has its own code that will run in case its message changes.



The code stablishes conditions depending on the number of the message received, so it creates the name of the sound to play depending on that number. Taking the case for the Byte 5 –one more time-, it has 3 different kinds of sounds, so it creates conditions for them and complete the names with the numbers. Then it changes the name on the *list-box* for that byte and virtually presses the button to play that sound.

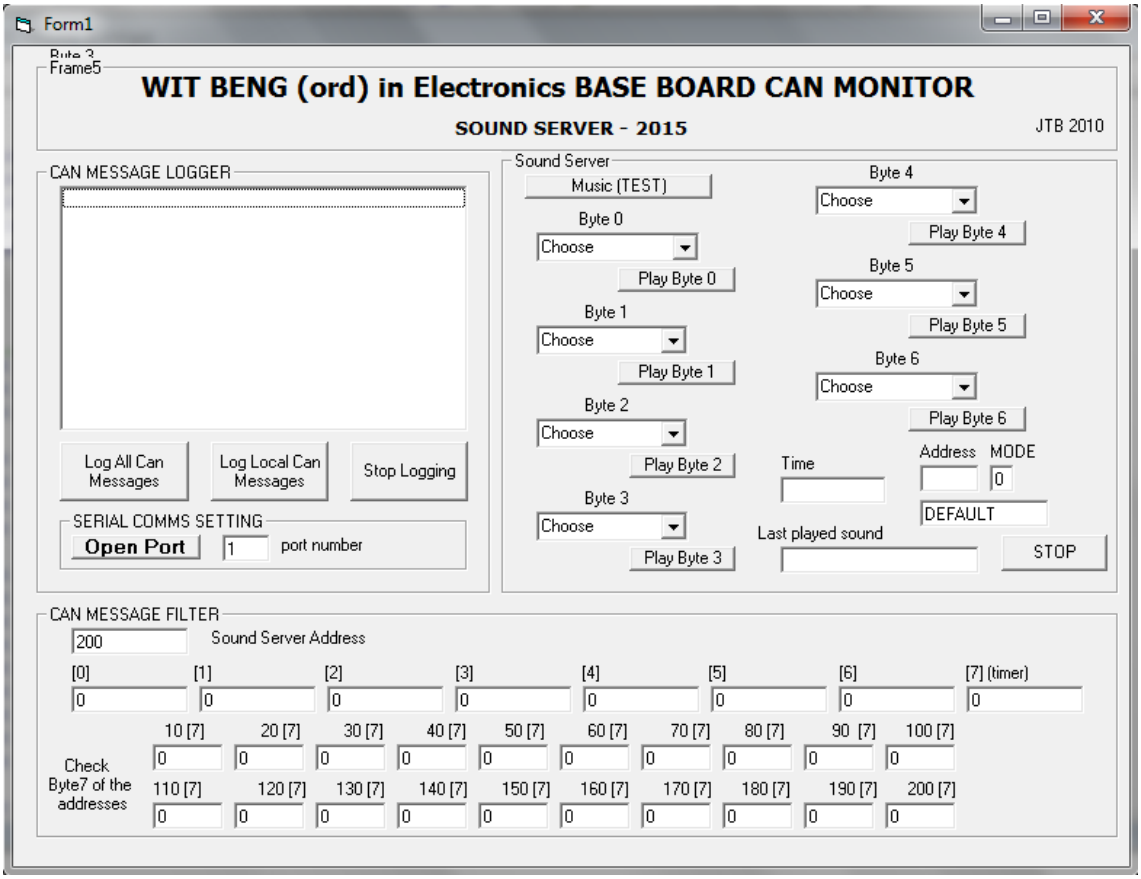
```
'_____________________________MESSAGE DATA BYTE 5_____________________________'

Private Sub Text55_Change()
    If Text21.Text = 200 Then
        If Text55.Text <> 0 Then
            If (Text55.Text >= 1 And Text55.Text <= 30) Then
                Combo6.Text = "Swoosh_" + Text55.Text
                Command15 = True
            ElseIf (Text55.Text >= 31 And Text55.Text <= 40) Then
                Combo6.Text = "Reverse_" + Text55.Text
                Command15 = True
            ElseIf (Text55.Text >= 41 And Text55.Text <= 50) Then
                Combo6.Text = "Laser_" + Text55.Text
                Command15 = True
            End If
            Else: Combo6.Text = "Choose"
        End If
    End If
End Sub
```

As we saw before, the same byte has 3 different kinds of sounds; "*Swoosh*" from message equal 1 to 30, "*Reverse*" from message equal 31 to 40 and "*Laser*" from message equal 41 to 50. So the conditions create a string with the name of the kind of sound and the number received, changes the value on the combo list box and presses the button attached to that byte.

When the button is virtually pressed, it will work as when we test the sounds, following the code shown on the section above this one.

3.5. Basic Interface

Now on we know how sounds are played and all the basic functions this interface provides to the user. Also we know how to add more sounds and the changes and conditions needed on the code in that case. All these settings brought the implementation of the first design for the interface –shown on the picture below. This interface adds the operation modes that will be studied on the following sections.



Once we have chosen the com port and logged all the messages the filter will do its function and show the received bytes. Each one of the changes on these bytes will change the names shown on the combo-list boxes on the right and play the sounds, depending on the number received.

The sound being played is shown in the box by "*Last played sound*". It can always be stopped pressing the bottom "*STOP*" that would run the following code:

```
'_____________________Stop playing the sound_____________________'

Private Sub Command6_Click()
    StopTheSoundNOW = sndPlaySound(soundfile$, wFlags%)
End Sub
```

Two sounds can´t be played simultaneously. If we play one while other is being played, it would stop it.

4. Development of the Operation Modes

The essential function of this software was already done at this point. The Sounds Effect System is able to play sounds when it receives a certain message through the can bus network, also to stop it when it is necessary. Nevertheless, this way of operation makes the rest of project send messages to this server constantly; therefore it would be really interesting to develop different modes of operation to make this server interact in better ways with the rest of embedded projects.

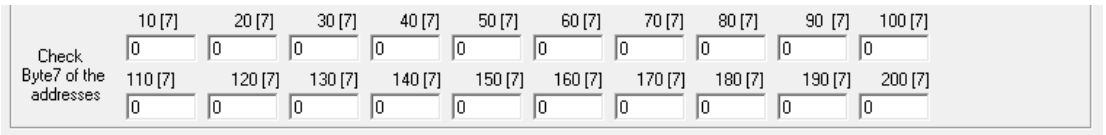
Three modes of operation have been developed in this first version of the software –also the second version that will be explained is based on these modes. All of them provide the user and the rest of the projects some ways to interact with the Sound Effects Server.

4.1. Choose the Operation Mode

Each one of the embedded projects uses a base board to interact with the rest of projects connected to the can bus network. That means each one of them has a different address on this network and the information located on their own address can be used for the rest of projects.

The idea for changing the operation mode on the Sounds Effects System is checking the byte 7 of each one of the addresses searching for a change. The bottom of the interface filters the seventh byte on all the projects; in the case that one of them changes its value to 1-2or3, it will change the mode of operation for the Sounds Server.

Mode 1 is the basic mode already studied, named as “Stop Message”; mode 2 is named “Time Out”; while the last mode, mode 3, is “Observer” mode. The following picture shows this filter where each box-text has some code to change the operation mode –as it will be explained below.



Also all these boxes can be hidden or deleted in order to show just the addresses that actually exist on the system or the addresses that would interact with the Sound Effects System.

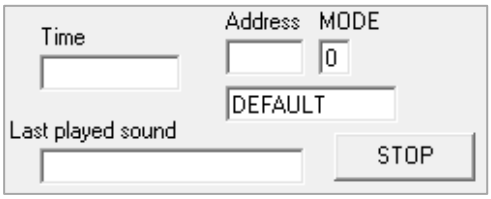
The piece of code in all the text-boxes in charge of looking for changes on the byte 7 of the different addresses is always the same. The screenshot below shows the code for the address 140 (head's address), that will be useful for explaining "*observer*" mode in the following sections.

```
Private Sub Text14_Change()
    If (Text14 >= 1 And Text14 <= 3) Then
        Text25.Text = "140"
        Text23.Text = Text14
    End If
End Sub
```

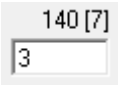
As we see, it is based on a simple condition; in case the number filtered is in between 1 and 3, it will change the value of two other different boxes that will notify the user and the system that the operation mode has changed.

These changes will be shown on the part of the interface that is displayed on the screenshot below. For example, if a project related with the head wants to use mode 3, it will change byte 7 on the address 140. This change will be filtered and displayed on the boxes of the picture:

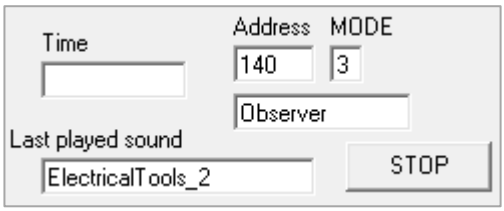
System waiting for changes



Address attached to the head changed byte 7



Text box changes operation mode and notifies it to the system

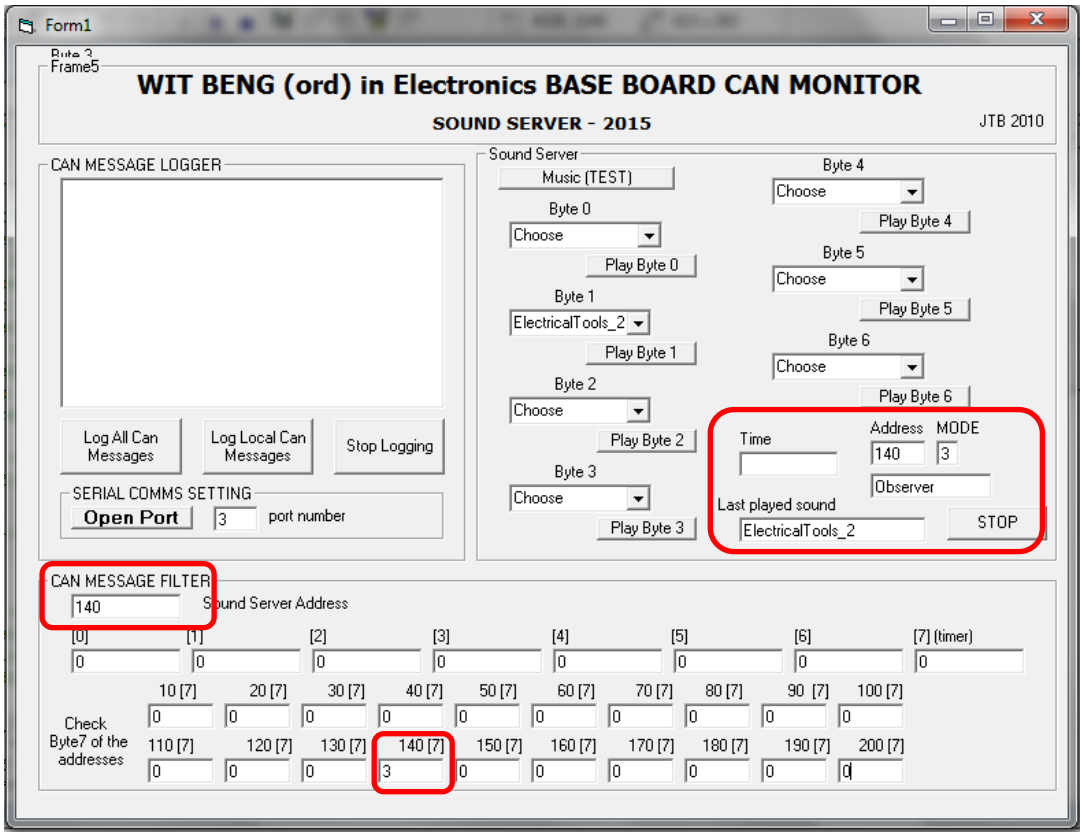


The sections below explain all the operation modes, how they work and how they interact with the rest of embedded projects working on BENGiE.

When one of the text boxes on the bottom of the interface filters a message that changes the mode of operation, it changes the value on the box shown above. The code for that piece of interface simply changes the letters on the textbox below it to notify the name of the mode working and also the value of the address that the system must filter now –if necessary. This last case is only used in mode 3, the rest of them use address 200 to receive the messages and play the sounds –the address of the Sounds Effects Server.

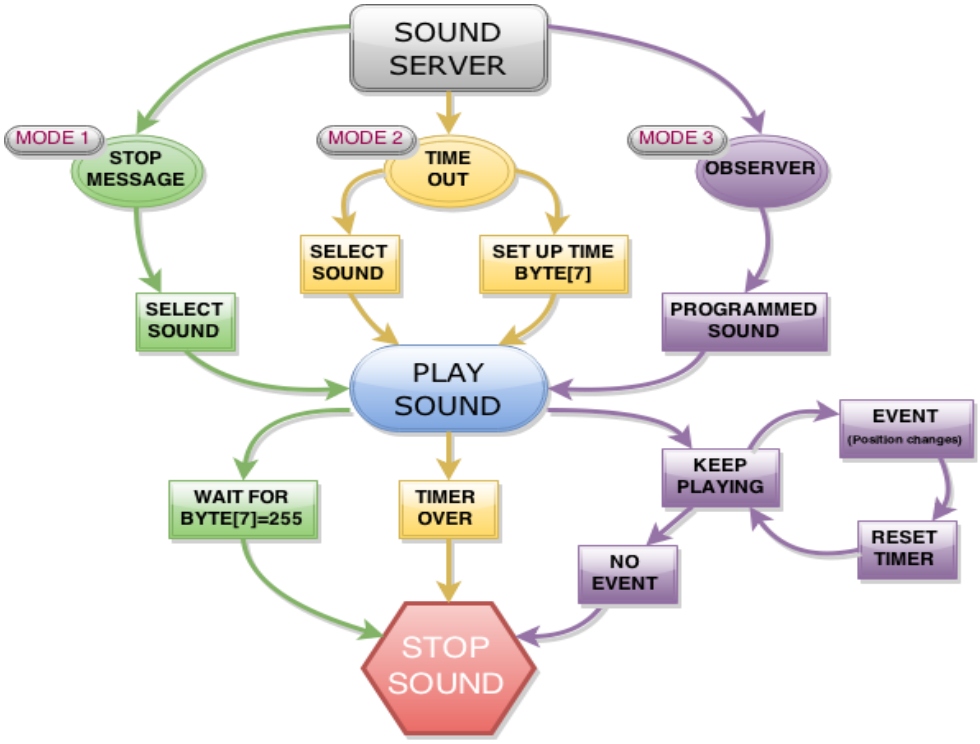
```
'_____________________Set up MODE of running_____________________'

Private Sub Text23_Change()
    If Text23.Text = 0 Then
        Text24.Text = "DEFAULT"
        Text21.Text = 200
    ElseIf (Text23.Text = 1) Then
        Text24.Text = "StopMessage"
        Text21.Text = 200
    ElseIf (Text23.Text = 2) Then
        Text24.Text = "TIMEOUT"
        Text21.Text = 200
    ElseIf (Text23.Text = 3) Then
        Text24.Text = "Observer"
        Text21.Text = Text25
    End If
End Sub
```

As an example, the picture below shows how the interface looks operation on mode 3; "*Observer*" mode.



4.2. Definition of the Modes and Flow Chart

The flow chart below shows how each one of the operation modes works related with the rest of the system. It depends on the message received, the common node for all of them it is that all will play a sound, although they are activated and stopped in different ways.



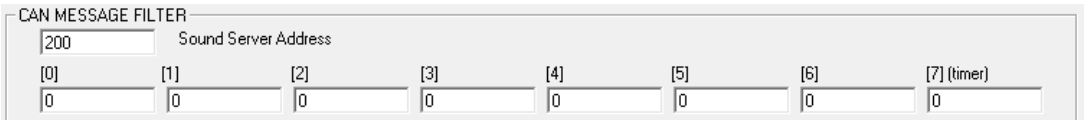
The modes available on the first version of the software are:

- MODE 1:
  The filter takes all the messages to address 200. These messages contain the information about the selected sound, so the Server will play it until the Sound Server receives a message 255 in its last byte. The last byte in this case is used to stop the sound.
- MODE 2:
  The filter receives a message with numbers in two of its bytes. The first one selects the sound that will be played while the other one will select the time that the server will be playing the sound.
- MODE 3:
  The *Observer* mode keeps looking how the information on a different address changes and will stop playing the sound when this bytes stop changing.

4.3. Operation Modes

4.3.1. MODE 1: Stop Message

The "*Stop Message*" mode works exactly as the basic function of the system explained before. The filter already programmed is configured to filter all the messages sent to the address of the Sounds Effects System (Address 200). This operation is also available as the first mode of the system or "*Default*" mode.



When a message is received on this address, it will be shown on the interface and will change one of the text boxes of the filter. That change will activate the code on programmed on that text box and play the sound. The code in all the text boxes of the can message filter is pretty similar, except for the first and the second byte; both of them add an extra function for the operation mode 3 –which will be explained later.

The fifth byte will be used as an example, as we have used it in the last sections.

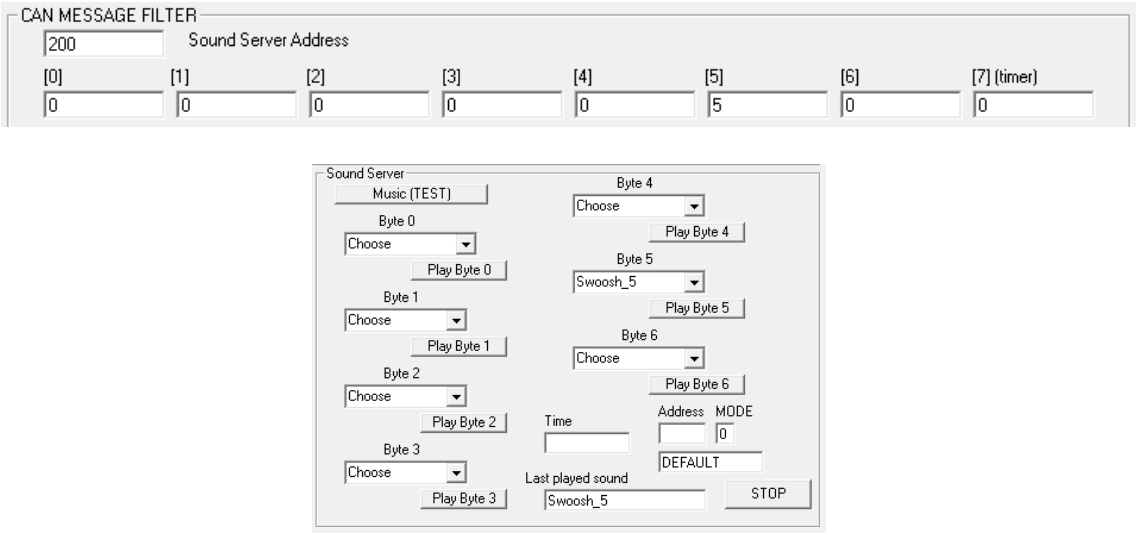
```
'_____________________________MESSAGE DATA BYTE 5_____________________________'

Private Sub Text55_Change()
    If Text21.Text = 200 Then
        If Text55.Text <> 0 Then
            If (Text55.Text >= 1 And Text55.Text <= 30) Then
                Combo6.Text = "Swoosh_" + Text55.Text
                Command15 = True
            ElseIf (Text55.Text >= 31 And Text55.Text <= 40) Then
                Combo6.Text = "Reverse_" + Text55.Text
                Command15 = True
            ElseIf (Text55.Text >= 41 And Text55.Text <= 50) Then
                Combo6.Text = "Laser_" + Text55.Text
                Command15 = True
            End If
            Else: Combo6.Text = "Choose"
        End If
    End If
End Sub
```

```
'                                                                             '
'_____________________________MESSAGE DATA BYTE 6_____________________________'

Private Sub Text56_Change()
    If Text21.Text = 200 Then
        If Text56.Text <> 0 Then
            If (Text56.Text >= 1 And Text56.Text <= 3) Then
                Combo7.Text = "Heartbeat_" + Text56.Text
                Command16 = True
            End If
            Else: Combo7.Text = "Choose"
        End If
    End If
End Sub
```

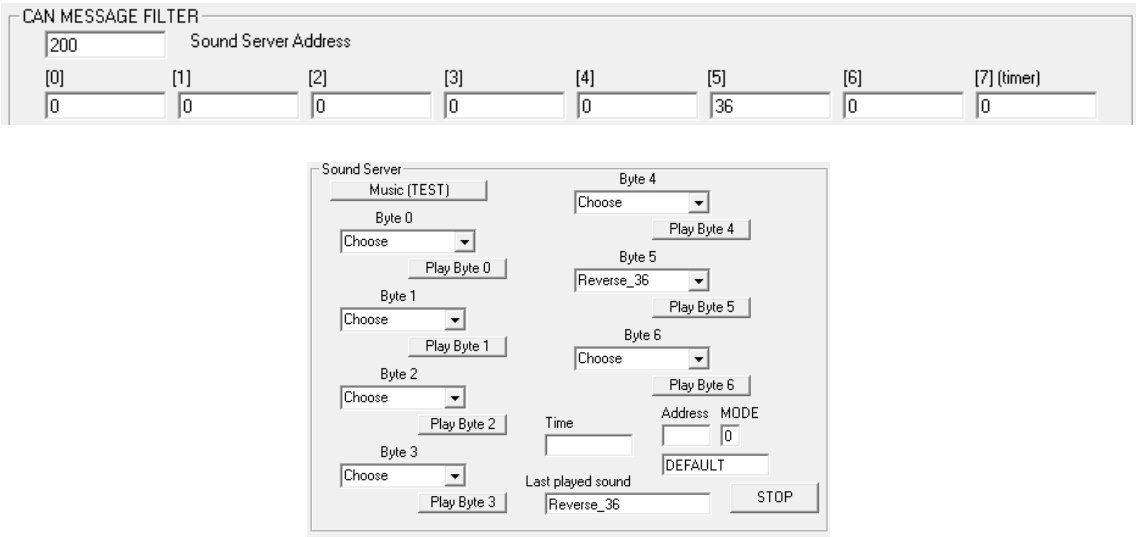
When the byte 5 receives a message (an external event changes its value), it starts certain conditions to know which sound it has to play. First of all, it checks that the address is 200, it means mode 1 or 2 are working. The second condition creates the name of the sound it has to play checking in between which two numbers is located the received one. That way matches with the explanation given at the beginning about how to locate and name the sounds on the folders.

Once it has created this name, it changes the value of its respective combo text box (the one for byte 5 in this case) and virtually presses it "*Play Sound"* button attached to it.

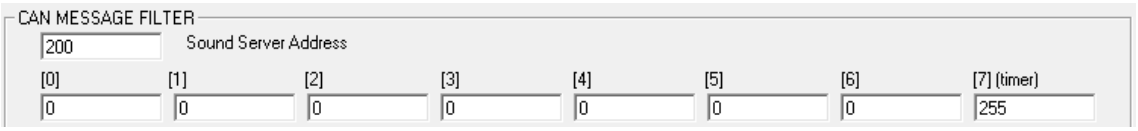
As an example it will be pretended to receive the number 5 so it would play the sound "Swoosh_5" located on the folder "C:/…./sounds/byte 5".



In case the received number is over 30, it would create a different string and play a different kind of sound. Let's so an example with the number 36:



In case a different user wants to STOP the sound being played, it just has to send another message to the same address with the number 255 on the last byte –byte 7.



This command will automatically presses the STOP button of the system –already explained-, and will stop the sound.

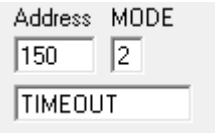
```
'_____________________________MESSAGE DATA BYTE 7_____________________________'

Private Sub Text57_Change()
    If Text21.Text = 200 Then
        If (Text57.Text > 0 And Text57.Text < 254) Then
            'Set the time out
            Text22.Text = Text57.Text
        'If Byte 7 is 255 it will stop the sound and mode STOPmessage is ON
        ElseIf (Text57.Text = 255 And Text23.Text = 2) Then
            Text22.Text = "0"
            Command6 = True
        End If
    End If
End Sub
```

The first condition of the code for this text box is programmed for the mode 2 of operation and will be explained on the next section. What the second part does is just virtually press the STOP button.

4.3.2. MODE 2: Time Out

The *"Time Out"* mode follows the same basic logic as the mode 1 explained before. The different appears in the fact that this mode only needs one message to know when the system has to stop the sound being played.

To start using this mode, the text box that shows the mode in which the system is operating every time must be changed:

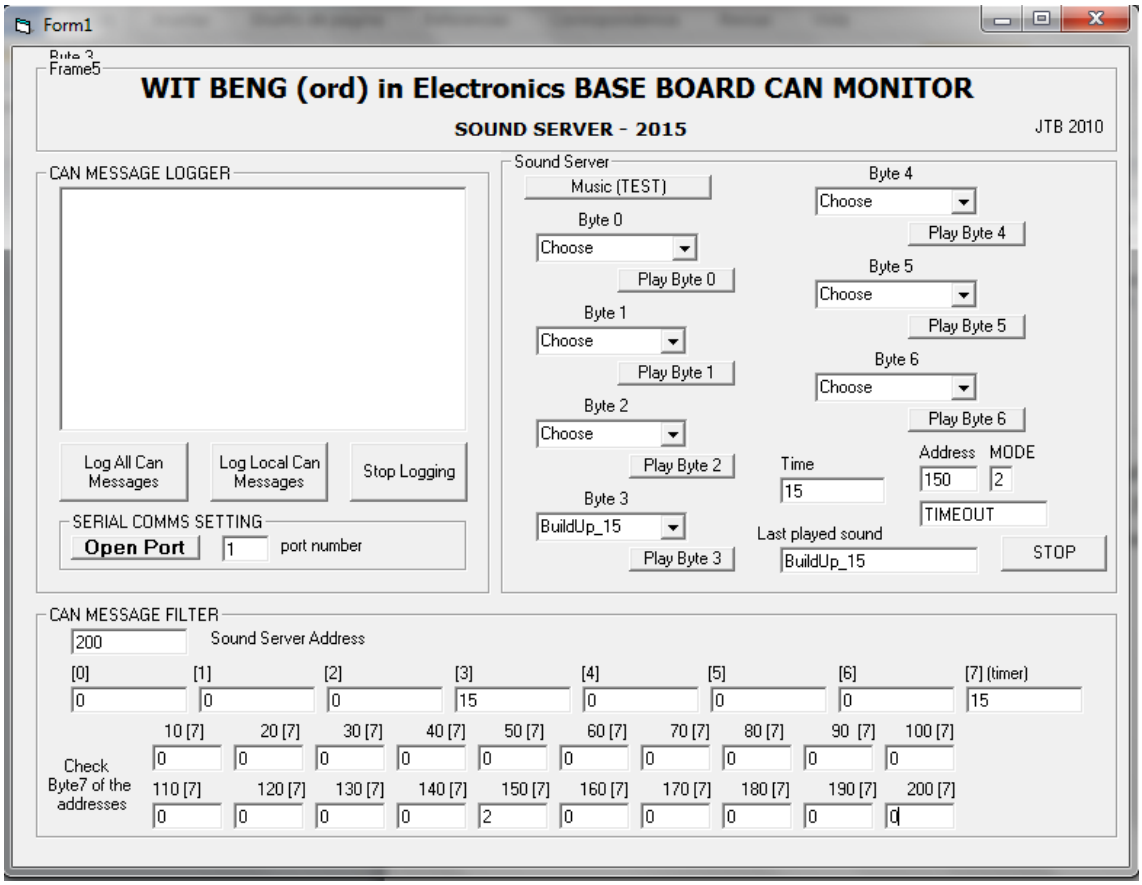


The filter is still focus on the address of the Sound Server, although this time it will receive a message with data different to zero in two of its bytes. The first one will be in a byte in between the first and the sixth; the second number will be received on the last byte of the address (byte 7) and will indicate the time the sound will be played. The value of this time must be sent in values of tenth of a second.

The code used on the text boxes of the filter in this case is exactly the same as when we use it on the first mode. As an example, the code for the byte 3 is shown on the screenshot below:

```
'_____________________________MESSAGE DATA BYTE 3_____________________________'

Private Sub Text53_Change()
    If Text21.Text = 200 Then
        If Text53.Text <> 0 Then
            If (Text53.Text >= 1 And Text53.Text <= 20) Then
                Combo4.Text = "BuildUp_" + Text53.Text
                Command13 = True
            ElseIf (Text53.Text >= 21 And Text53.Text <= 27) Then
                Combo4.Text = "Cymbals_" + Text53.Text
                Command13 = True
            ElseIf (Text53.Text >= 28 And Text53.Text <= 36) Then
                Combo4.Text = "Drums_" + Text53.Text
                Command13 = True
            End If
            Else: Combo4.Text = "Choose"
        End If
    End If
End Sub
```

Now, one of these sounds will be played for one and a half seconds; that means the value sent to the last byte of this address should be 15 (1.5 seconds). The picture shows how the system reacts to this order that receives on mode 2.



How the STOP button is pressed this time is slightly different to the last case. A timer is programmed to start counting when the system starts playing the sound, so when the counter is over, the sound will stop. Obviously, the counter rises to the number entered by the user through the can bus message.

See the difference on the "*Play Sound*" button attached to byte 3 (used as an example this time). The string with the name of the sound is created and displayed on the combo text box with the same name of the file to be played before pressing the button. Once it is pressed, the timer is active when the mode 2 is running.

```
'_____________________________BYTE 3 SOUNDS_____________________________'

Private Sub Command13_Click()
        FileAddress = "C:\Users\20069162\Desktop\sounds\Byte 3\" + Combo4.Text + ".wav"
        'Address in Bengie:
        'FileAddress = "C:\Users\Electronics\Desktop\SOUND SERVER\sounds\Byte 3\" + Combo4.Text + ".wav"
        soundfile$ = FileAddress
        wFlags% = SND_ASYNC Or SND_NODEFAULT
        sound = sndPlaySound(soundfile$, wFlags%)
        Text31.Text = Combo4.Text

        If Text23.Text = 2 Then
            counter = 0
            Timer2.Enabled = True
        End If
End Sub
```

The condition on the bottom of the code for this button checks that the mode 2 is the selected one, reset the counter of the timer and starts the timer.

The value of the last byte automatically change the value of the text box 22 where is always displayed for how long the sound will be played.

```
'_____________________________MESSAGE DATA BYTE 7_____________________________'

Private Sub Text57_Change()
    If Text21.Text = 200 Then
        If (Text57.Text > 0 And Text57.Text < 254) Then
            'Set the time out
            Text22.Text = Text57.Text
        'If Byte 7 is 255 it will stop the sound and mode STOPmessage is ON
        ElseIf (Text57.Text = 255 And Text23.Text = 2) Then
            Text22.Text = "0"
            Command6 = True
        End If
    End If
End Sub
```

The code for this text box uses for the time is attached below. That text box equals the value of the variable "*decsec*" to its own value, so this variable can be used by the timer anytime.

```
'                       TIMER SETUP                                          '
'____________________________________________________________________________'
' Timer will count in intervals of 0.1 seconds                               '

Private Sub Text22_Change()
    decsec = Text22.Text
End Sub
```

```
Private Sub Timer2_Timer()
    counter = counter + 1
    If counter = decsec Then
        Command6 = True
    End If
End Sub
```

The timer works as an interrupt that run every 100 milliseconds -0.1 seconds-, increment a variable and check its value with a condition. When the value of the variable "counter" reaches the value of the variable "*decsec*" –variable equal to the total time-, the program pressed the button "STOP", so the sound will be stopped.

This mode of operation will be used as the essential code to programme the improvements added on the last version of the interface, which will be shown in following sections of the report.

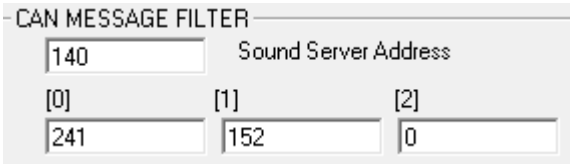
4.3.3. MODE 3: Observer

When a different project wants the Sounds Server to follow its movements and play sounds according to when something is done, it would change its byte 7, sending that way the order to start the "*observer*" mode. For now this mode has only been programmed for the address 140, which is the one that sends messages to the head to move up-down (first byte) or right-left (second byte).

Once the byte 7 on the base board attached to the head is changed, the mode of operation will change. Also the box that says which mode is working will change the address of the filter and the Sound Server will start "*observing*" the changes of the two first bytes of this address to all the messages sent to the servo-motors in charge of moving the head.

Address 140 changes to mode 3

The system starts observing and reading the changes on the bytes for the address of the base board that controls the head.



When the observer mode has been stablished, the text box that stores the value of the address that the system is filtering, choose the sound that will be played while the system is working on this mode with this address. In this case, the chosen sound is "*ElectricalTools_2*" on the sounds from byte 1, therefore the button used to play it is the *Command11*.

```
'_____________________SOUND DESIGNED WHILE OBSERVER MODE_____________________'
'                                                                             '
' Depending on the value of the address, it will play a different sound       '
' in order to find the most suitable one for each function                    '

Private Sub Text21_Change()
    If Text21.Text = 140 Then                'Address 140: NECK and HEAD
        Combo2.Text = "ElectricalTools_2"    'Pick sound for that element (Head movement)
        Command11 = True                     'Button that plays that sound
    End If
End Sub
```

When mode 3 is operating and Command11 is pressed (button attached to byte 1 sound), the timer 3 will start working. The picture below shows the code for the byte 1 "*Play Sound*" button:

```
'_____________________________BYTE 1 SOUNDS_____________________________'
Private Sub Command11_Click()

        FileAddress = "C:\Users\20069162\Desktop\sounds\Byte 1\" + Combo2.Text + ".wav"
        'Address in Bengie:
        'FileAddress = "C:\Users\Electronics\Desktop\SOUND SERVER\sounds\Byte 1\" + Combo2.Text + ".wav"
        soundfile$ = FileAddress
        wFlags% = SND_ASYNC Or SND_NODEFAULT
        sound = sndPlaySound(soundfile$, wFlags%)
        Text31.Text = Combo2.Text

        If Text23.Text = 2 Then
            counter = 0
            Timer2.Enabled = True
        ElseIf Text23.Text = 3 Then
            Timer3.Enabled = True
        End If

End Sub
```

To understand how this mode works we have to at the code of the boxes for the two first bytes of the can bus message filter. The first part of these codes is the interesting part for this mode, because the code on the bottom of both of them has been studied before on the other modes. As we see, the first part stablish a condition to see when mode 3 is working, in that case it goes to the following commands.

```
'____________________________MESSAGE DATA BYTE 0_____________________________'

Private Sub Text50_Change()
    If Text23.Text = 3 Then
        If soundOff = 1 Then    'If there is no sound being played
            soundOff = 0        'Says there is a sound being played
            Command11 = True    'Start the sound
            timer3control1 = 1  'Let the timer know it has changed
        Else: 'If there's a sound being played
            timer3control1 = 1  'Let know it has changed again to continue playing the sound
        End If
    Else:
    If Text21.Text = 200 Then
        If Text50.Text <> 0 Then
            If (Text50.Text >= 1 And Text50.Text <= 9) Then
                Combo1.Text = "Servo_" + Text50.Text
                Command10 = True
            ElseIf (Text50.Text >= 10 And Text50.Text <= 21) Then
                Combo1.Text = "Ambience_" + Text50.Text
                Command10 = True
            End If
        Else: Combo1.Text = "Choose"
        End If
    End If
    End If
End Sub
```

Also the code on the second byte has the same lines.

```
'_____________________________MESSAGE DATA BYTE 1_____________________________'

Private Sub Text51_Change()
    If Text23.Text = 3 Then
        If soundOff = 1 Then
            soundOff = 0
            Command11 = True
            timer3control2 = 1
        Else:
            timer3control2 = 1
        End If
    Else:
        If Text21.Text = 200 Then
            If Text51.Text <> 0 Then
                If (Text51.Text >= 1 And Text51.Text <= 22) Then
                    Combo2.Text = "ElectricalTools_" + Text51.Text
                    Command11 = True
                ElseIf (Text51.Text >= 101 And Text51.Text <= 108) Then
                    Combo2.Text = "R2D2_" + Text51.Text
                    Command11 = True
                ElseIf (Text51.Text >= 110 And Text51.Text <= 122) Then
                    Combo2.Text = "SCIFI_" + Text51.Text
                    Command11 = True
                End If
            Else: Combo2.Text = "Choose"
            End If
        End If
    End If
End Sub
```

This section uses two global variables that will be changed by the text boxes when their values change, or by the timer that will control when to stop the sound. These variables are "*soundOff*" and *"Timer3control*" –this last one will be different for the first and the second byte.

For example, what the code for the first byte does, basically, is check if there is no sounds being played, that means soundOff=1, in that case press the button to play the sound, equals Timer3Control1=1 and soundOff=0 –to let know a sound is being played. Every time the message to the head changes, which means that the head keeps moving, this variable Timer3Control will store the value 1. Those values will be used by the timer shown below.

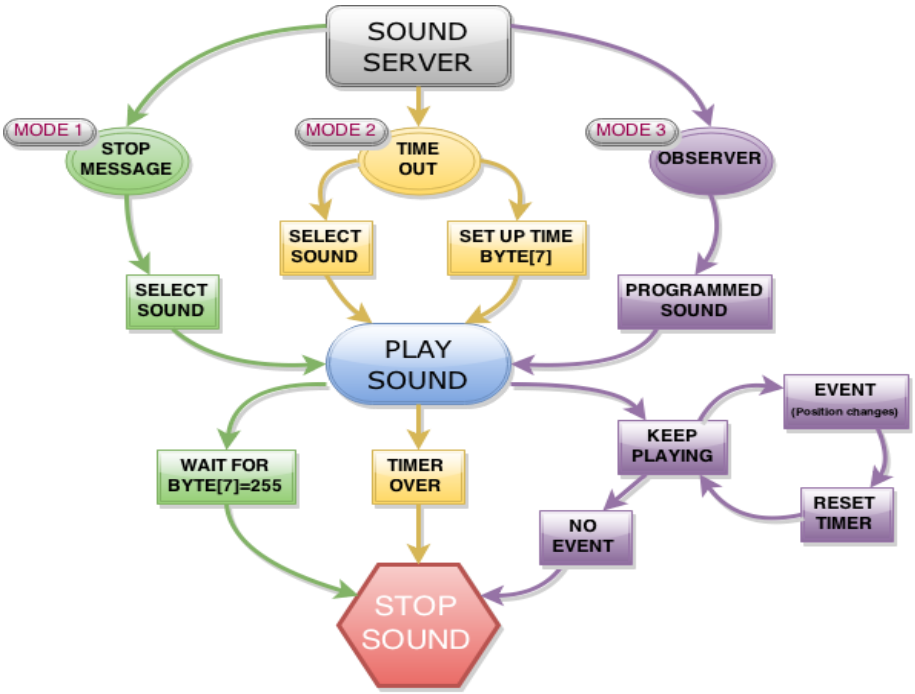
```
'                      READING TIMER SETUP                                    '
'_____________________________________________________________________________'
' Timer will count in intervals of 0.1 seconds                                 '

Private Sub Timer3_Timer()
    'If one of the two bytes that the computer is reading has changed,
    'the counter turns to 0, as well as the controllers.
    If (timer3control1 = 1 Or timer3control2 = 1) Then
        readingcounter = 0
        timer3control1 = 0
        timer3control2 = 0
    'Else if noone of the read bytes has changed, the timer continues working
    'until it rises to the top of the count. It stops the sound there and
    'changes the value of the cariable soundOff to let know that it has stopped
    Else:
        readingcounter = readingcounter + 1
        If readingcounter = 3 Then 'if there are no changes in 0.3secs, it will stop playing the sound
            Command6 = True
            soundOff = 1
        End If
    End If
End Sub
```

Both Timer3Control1 and Timer3Control2 variables will be checked on the timer that will work as an interruption every tenth a second (0.1 seconds). If one of them is equal to 1, it means the head is still moving so the counter will be reset and also these two variables, so they can notify when it has been another change on the bytes.

In the case no one of the bytes have changed to move the head, the "*readingcounter*" variable used by the timer will start counting until it reach to 3. In that point, the condition will be TRUE and so the STOP button will be pressed to stop the sound. Also, when the sound stops, the variable *soundOff* goes back to the value 1. In this case, it will allow the system know that it has to start playing the again because there no one being played.

In other words, it will play a sound while the time in between two messages to this address is less than 0.3 seconds, which is an amount of time that human ear can hardly notice.

This operation mode could be implemented on different addresses, so different embedded project, as the arms of the LEDs strips. The result actually works as a project to give a more interesting appearance to BENGiE, so the sound server only plays sounds for the head when it is moving and stops playing the sound when the head stops.



5. Improvements of the Sounds Effect System

At this point of development of the project, the software has achieved the basic steps proposed at the beginning of the semester and it provides different modes of operation. As a "slave" system, other projects must make it work as they want, so it will wait for orders.

Although it works properly, it forces the rest of programmers to send messages continuously every time they want to play a sound. They only mode that does not need to receive a message on the address 200 is "*observer*" mode. That would obligate all the programmers to modify their codes and send orders to the sound server at the same time they do to its respective projects.

In order to create an easiest way to interact with other projects, the last weeks of the semester have been used to implement a new interface that uses "*Time Out*" mode to configure loops of sounds that matches with different programmed modes of different projects.

For now, it has been implemented and tested working with the *"Knight Rider"* mode of operation available on the base board attached to the LED strips. The following sections explain and show how this new improvement work and the ways to define the final effect that it provides.

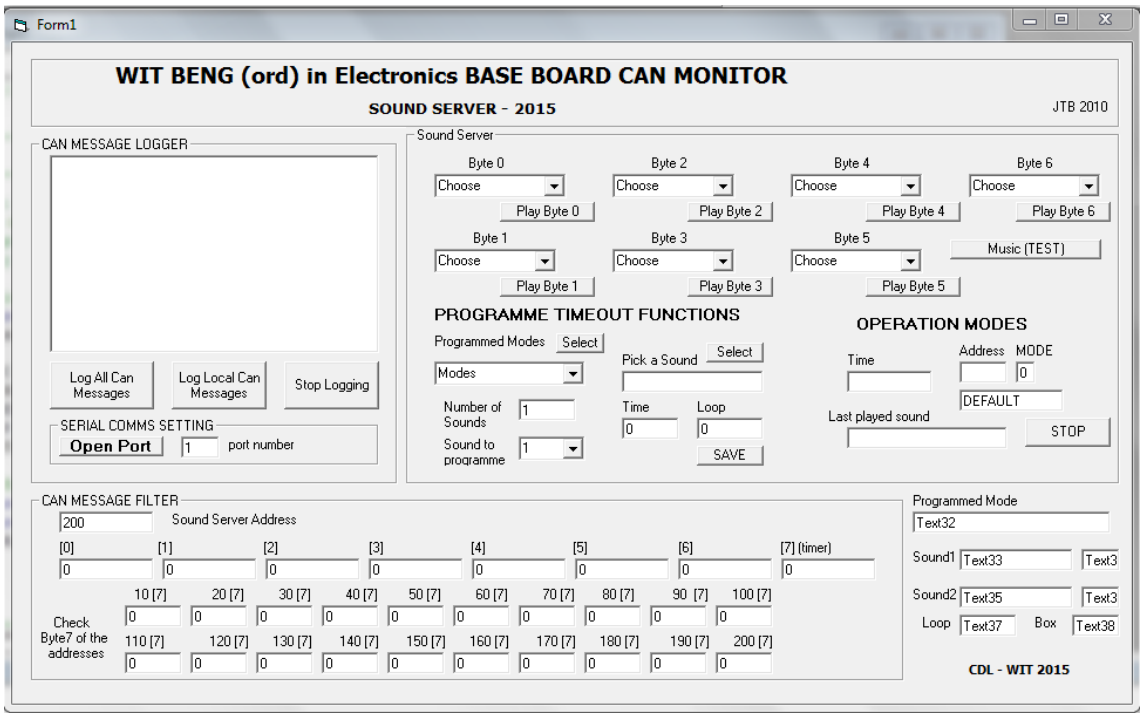
5.1. New Interface

This new design tries to solve problems of interaction with the rest of embedded systems by given the user a new interface in which loops of sounds can be programmed to be played when a certain message is read on the network.

The interface was created using the same design as in the last version but generating new parts for this new application. The user can choose up to two sounds – for now- to be played when a different action is executed on another project. It also defines the time of each one of the sounds and the times that these sounds will be played as a loop.

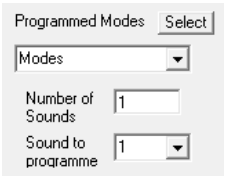
This method matches the repetitive movements programmed in other parts of the project as the head, arms or LEDs strips and creates the illusion of the sounds being part of a different project.

The image shown below is the new design –slightly different to the first one- used for this new method of operation.

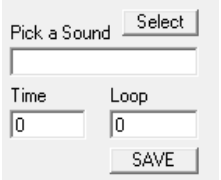


As we see on the picture above, there are three new sections on the interface for the Sound Effect System:

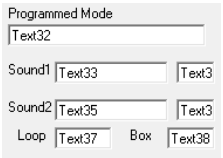
- **Programme Timeout Functions**
  This section shows the user which modes on different projects can be used to programme sounds to match with.



- **Pick a sound**
  This space is used to select the sounds that will create the loop. Also the number of loops and the time of each sound are chosen here.



- **Programmed Mode**
  Those boxes show the selected mode whose activation message the project will look for on the network and the settings programmed for the loop.



5.2. Creating Loops of Sounds

Nowadays, the only working programmable mode is the one that matches the mode *"Knight Rider"* operating on the LEDs strips. This mode will be used as example and explanation for future developments.

5.2.1. Step 1. Choose mode

First of all, we must choose which mode we want to operate with. In this case, and as the only one available for now, "*Knight Rider*" mode from the LEDs strips is selected from the list, then "SELECT" button is pressed.

As it is shown in the picture, the text box "*Number of Sounds*" changes its value. That gives the user information about the number of sounds that will complete the loop in this mode. It also changes the information available on the zone "*Programmed Mode*" where the user will see all the settings picked for the loop.

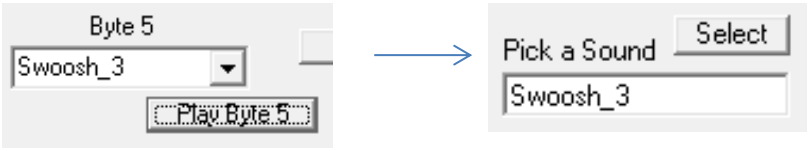
The code for the button "*Select"* is in charge of changing that text boxes as the following screenshot shows.

```
'SELECT buttom for the available programming modes

Private Sub Command9_Click()
    If Combo8.Text = "KNIGHT RIDER" Then
        Text29.Text = "2"
        Text32.Text = "KNIGHTRIDER"
    ElseIf Combo8.Text = "BRIGHTENING" Then
        Text29.Text = "2"
        Text32.Text = "BRIGHTENING"
    ElseIf Combo8.Text = "HEAD: yes" Then
        Text29.Text = "2"
        Text32.Text = "BRIGHTENING"
    End If
End Sub
```

5.2.2. Step 2. Selection of the first sound

The text box "*Sound to programme"* already shows the number 1, which means the following changes will be saved as the first sound. What we have to do to pick the first sound is play it on the trial boxes programmed for the first interface, choose the time that the system will be playing it and enter the number of loops it will play.

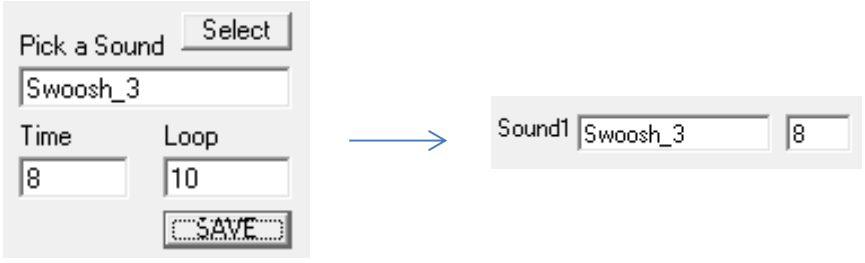
After we have played the sound, we must press the button "*Select"* close to the text box named as *"Pick a sound"* and the name of the last played sound will appear on that text box.



The code that runs when this button is pressed just changes the name of the text box for the string displayed on one of the combo list boxes where the sounds are located.

```
'SELECT buttom for the sounds

Private Sub Command8_Click()
    If Combo1.Text <> "Choose" Then
        Text26.Text = Combo1.Text
    ElseIf Combo2.Text <> "Choose" Then
        Text26.Text = Combo2.Text
    ElseIf Combo3.Text <> "Choose" Then
        Text26.Text = Combo3.Text
    ElseIf Combo4.Text <> "Choose" Then
        Text26.Text = Combo4.Text
    ElseIf Combo5.Text <> "Choose" Then
        Text26.Text = Combo5.Text
    ElseIf Combo6.Text <> "Choose" Then
        Text26.Text = Combo6.Text
    ElseIf Combo7.Text <> "Choose" Then
        Text26.Text = Combo7.Text
    End If
End Sub
```

Once the first sound has been selected, the time for this sound and the number of loops must be setup. After choosing them, the button "*SAVE*" is pressed to save the changes. The timer used in this case is the same one used in "*Time Out"* mode, so the time will be selected in tenths of second. Also, the mode of operation running will playing this kind of loops must be mode 2 –or "*Time Out*" mode.



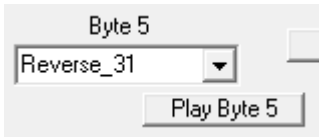
The code for the save button is created with a condition to differ the first and the second sound and change the correct box on the "*Programmed mode"* zone of the interface.

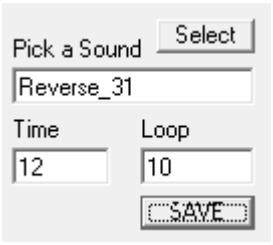
```
' SAVE buttom

Private Sub Command7_Click()
        If Combo9.Text = "1" Then
            Text33.Text = Text26
            Text34.Text = Text28
            Text37.Text = Text27
        ElseIf Combo9.Text = "2" Then
            Text35.Text = Text26
            Text36.Text = Text28
            Text37.Text = Text27
        End If
End Sub
```

5.2.3. Selection of the second mode

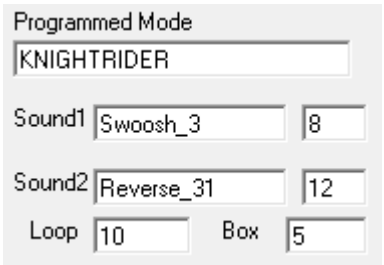
The box close to "*Sound to programme"* letters must be changed to select the second sound that will be part of the loop. Then we can follow the same steps as selecting the first sound and save the settings.

The sound to be selected must be played and then the button "*Select*" has to be pressed. Also the time for the second sound must be chosen. The number of loops is the total number that the first and second sound will be played, so there is no need to change it.





After we have saved the settings for the second sound, the loop is already created and ready for be played when the LEDs strips receive the message for playing the *"Knight Rider*" mode. The final appearance of the "*Programmed mode"* part of the interface is shown on the screenshot below.



5.3. Playing the loop

Once the sounds we want to match with the "*Knight Rider"* mode as a sound loop played will the lights go up and down on the LEDs strips, the only way to check when the message is sent to the strips is looking at all the messages through the network.

The following conditions that play the sounds loop are located on the code for the timer that runs as an interruption receiving, reading and filtering all the messages. That means that every 0.01 seconds a message will probably be received and it will be checked in case it is the message that carries the instruction for *"Knight Rider*".

This message is delivered on the first byte of the address 23 with the information "2". In other words, if the address 23 receives a message with a number 2 on its first byte, and also the system is running in mode 2 ("Time Out").

```
'       DEFINITION OF MODES IN TIMEOUT MODE. (Programmed by the user)     '

'_____________Check KNIGHTRIDER                                           '
'If the address 24 receive the message 4 in its byte7
  If (can_address_number0 = 23 And can_data_number0 = 2 And Text23.Text = 2) Then
            For Loopcounter = 1 To Loopfinal 'Text30.Text
                If Text38.Text = 0 Then
                    Combo1.Text = Text33
                    decsec = Val(Text34.Text)
                    Command10 = True
                    Pause (decsec)
                    Combo1.Text = Text35
                    decsec = Val(Text36.Text)
                    Command10 = True
                    Pause (decsec)
                ElseIf Text38.Text = 1 Then
                    Combo2.Text = Text33
                    decsec = Val(Text34.Text)
                    Command11 = True
                    Pause (decsec)
                    Combo2.Text = Text35
                    decsec = Val(Text36.Text)
                    Command11 = True
                    Pause (decsec)
                ElseIf Text38.Text = 2 Then
                                    ↓
                ElseIf Text38.Text = 6 Then
                    Combo7.Text = Text33
                    decsec = Val(Text34.Text)
                    Command16 = True
                    Pause (decsec)
                    Combo7.Text = Text35
                    decsec = Val(Text36.Text)
                    Command16 = True
                    Pause (decsec)
                End If
            Next
  End If
```

When all the settings are chosen, the number of the byte related with the picked sounds is also saved. In the example above, the sounds used for the loop were chosen on the folder for the byte 5 where “swoosh” and “reverse” sounds are located, so the number of this byte is also saved.

Once the message for the *“knight rider”* mode is received, the code runs the conditions programmed that are shown above. Inside this condition, a loop with the chosen number of replays is created and also inside these loops the number of the folder where the picked sounds is checked. The code for each condition is the same, except for the combo text box where the loop sends the name of the sound and the “*Play button*” that is played.

The variable “*Loopfinal*” contains the value of the selected number of loops. In the example this code is the one that would run.

```
For Loopcounter = 1 To Loopfinal 'Text30.Text
        '....
        ElseIf Text38.Text = 5 Then
            Combo6.Text = Text33
            decsec = Val(Text34.Text)
            Command15 = True
            Pause (decsec)
            Combo6.Text = Text35
            decsec = Val(Text36.Text)
            Command15 = True
            Pause (decsec)
            Text39.Text = Loopcounter
        '.....
 Next
```

While a sound is being played the code continues running, so it had to be found a way to stop it for the time each sound is being played. The function “*PAUSE*” was programmed with that purpose.

```
'TIMER FOR THE PAUSE_____________________________________
Sub Pause(interval)
    countersleep = 0
    Timer4.Enabled = True
    Do While countersleep < Val(interval)
        DoEvents
    Loop
End Sub
```

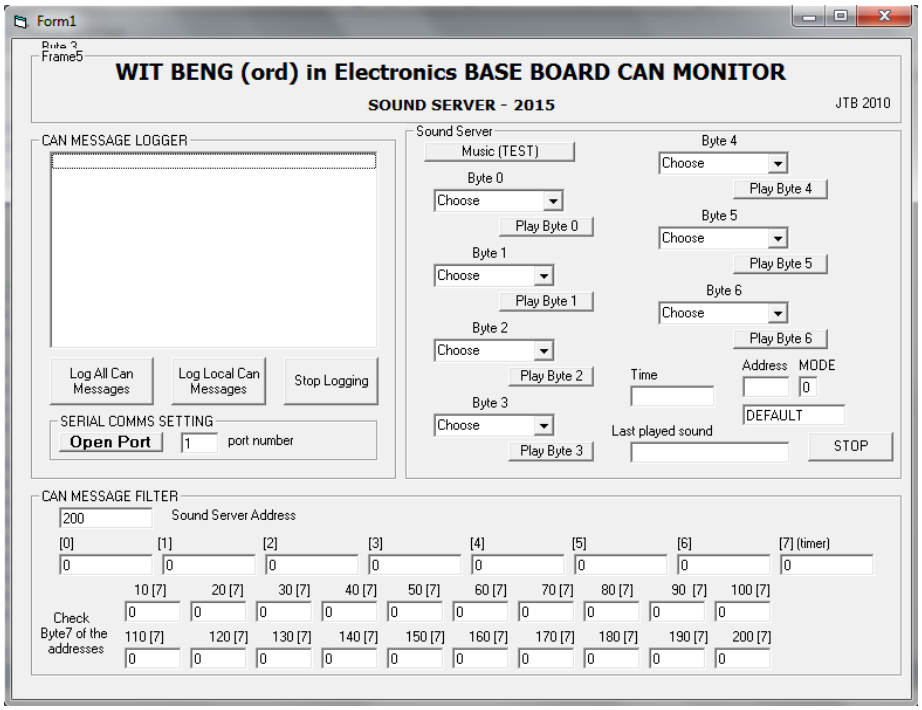
That function starts a new timer (*timer 4*) that will increase the value of a global variable, and then it will enter an infinite loop using “*Do while*” and the function *“DoEvents*” that will be running until the value of the global variable “*countersleep*” is equal to the entered value “*interval*” –the value that will be sent from the code of the sounds loop, and it is programmed by the user.

```
Private Sub Timer4_Timer()
    countersleep = countersleep + 1
End Sub
```

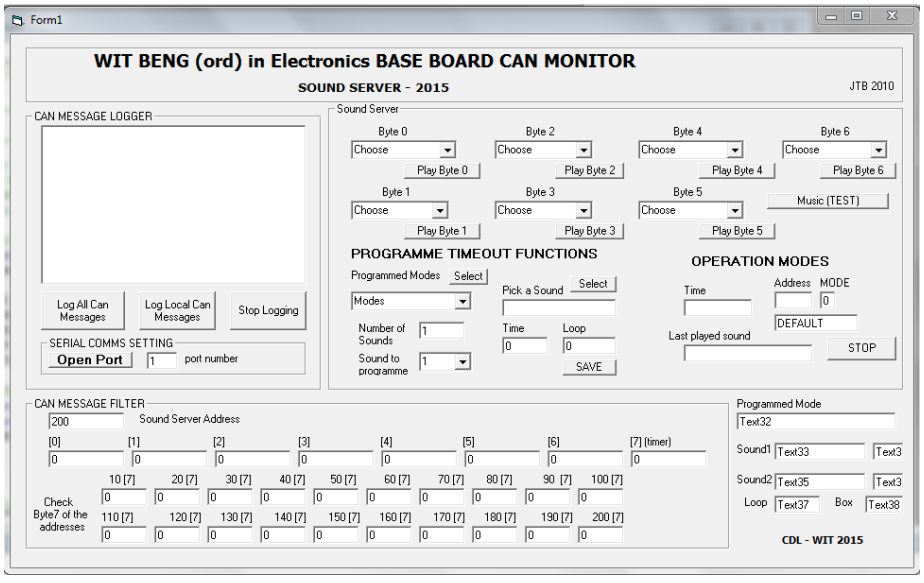
6. Final Results

As it was explained before, there are two different designs for the Sound Effects Server. The second one uses exactly the same code of the first one and adds the new function to match loops of sounds with different functions in other projects. The interfaces of both versions are slightly different, as the new one needed more options, but the operation modes are the same. In the pictures below both designs can be compared.

Sounds Effects System. Design 1



Sounds Effects System. Design 2



To sum up, both of them present the same basic operation modes; “Stop message”, “Time Out” and “observer”. The working functions of “Stop Message” and “Time Out” are basically the same except for how they stop the played sound. Both of them receive a message to the address of the sound server on one of its first six bytes and play the sound attached to this message. Then, the first one need another message to stop the sound (sent to the last byte of the address), while the second mode need the value of the time that will be playing the sound displayed on the last byte of the address at the same time as it receives the order of the chosen sound.

The third operation mode “tracks” the messages sent to a different address in order to know when an operation or movement is taking place, so it can play or stop the sound depending on the status of a different project.

Those modes are also available –working on the same way- in the second design. This new design was developed on the necessity of improvement. It provides a new way to play sounds base on the operation mode “*Time Out*” that allow the user to create loops with sounds for several different operation modes that can be designed in other projects.

6.1. Future improvements

Facing the future of this software from the code already programmed can be an interesting option for future students. More sounds can be added, but also the code can be refined and improved. For now, there is only one mode sequence which we can match the sound loops on the last interface, so it can also be interested to create the necessary conditions for allowing create loops in more sequences.

All the instructions about how to add more sounds, or options, are given on the different sections of this report, but also some explanations about how the code works were added on the actual code as comments. The complete code will be uploaded on moodle platform.

Now that the code was tested and all the methods were able to operate with the rest of projects, the next step would be make everything work together and get to the point in which all the projects know about each other and can cooperate sending and receiving messages through the network.

7. Conclusion

This project has developed a Sounds Effects System to work as an embedded project on BENGiE robot. BENGiE is now the result of many years of different students working together to create more and more functions giving it better conditions every year. In particular, this project has been developed almost from scratch based on the already programmed Can Bus Monitor used to read and filter the messages sent through the can bus network that connects all the base boards.

It has been a great experience to work with a magnificent group of students on a common project as if we were facing a real engineering experience in a company. Everyone helped each other in so many different levels and it was actually rewarding to see the finished projects working on BENGiE.

The beginning was –probably- the tougher part of the creation for the Sound Server. Organize and create the server based on the Can Bus Monitor software was relatively easy, although creating the different modes of operation and developing how they would work was not that simple.

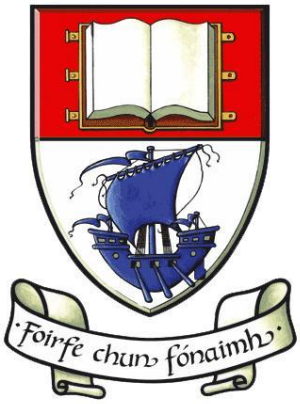
Visual Basic 6 was a language of programming which I had never used before, so it has been a challenge in every moment to understand how it works and how it can be used for different purposes. It sometimes delimits the options of the programmer and can be hard to understand. Thus, it could be a good option for following developers to try to create a similar interface running with a different language.

In general, it has been a wonderful experience as a student to have the opportunity of learning about electronic engineering abroad. Studying at Waterford Institute of Technology for the last months has been a challenge in many moments especially for the first month, when I was trying to get used to the language and the new college as soon as possible. But I found no problems in any moment thanks to all the opportunities this school of engineering has offered and the facilities that the lecturers have given to the Erasmus students to adapt and get into the rhythm of the classes easily.

This subject "*Embedded Project Application*" is the clearest example. I could not be happier about have chosen this subject in which everyone has worked on his own project but always with a common goal under the point of view of a team of engineers led by the lecturers Jason T. Berry, PJ Walsh and Paddy Murray.

SOUND EFFECTS SYSTEM (SES)

EMBEDDED PROJECT APPLICATION



Carlos Durango Labrador

Erasmus Student

2015