DSP Audio Effects System

Detailed Description

Apart from Theremin this project includes the design of the audio effects system which is implemented in hardware. The effects implemented include: Distortion, Tremolo, Ring Modulation, Echo, Vibrato and Slapback. The hardware chosen is an FPGA board, Terasic DE1-SoC, which includes all the needed components for our purposes some of which are: Wolfson WM8731 audio codec, Line In, Line Out and Mic In 3.5 mm inputs and outputs, 10 switches, 6 seven segment displays, 4 buttons and, of course, the FPGA processor itself, Intel Cyclone 5 FPGA. An intuitive user interface was designed to give the user the ability to control the effects, change different parameters of every effect or turn off or on any effect when desired.

1. Software Environment: Intel Quartus Prime IDE

Absolute first step was to learn to use Quartus Prime. This IDE includes all the tools needed for the design such as the synthesis tool, ModelSim simulation tool, Signal Tap Logic Analyzer, schematic builder and many other essential tools for development. The language we chose is SystemVerilog.

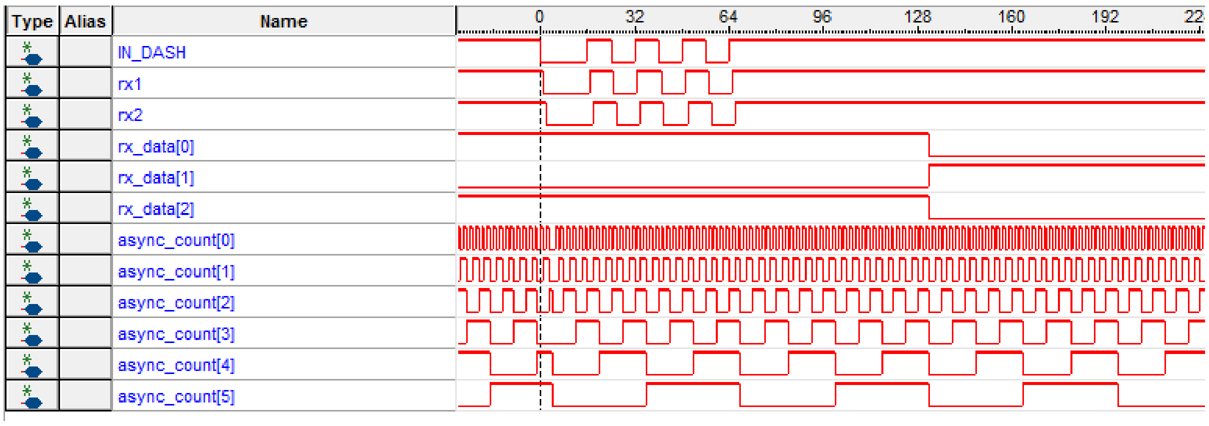
1.1 SystemVerilog HDL

When it comes to choosing the hardware description language there is not many options to choose from. It's either VHDL, Verilog or SystemVerilog. Even then, out of these 3 options, Verilog and SystemVerilog are basically the same HDLs, SystemVerilog being a more modern one with more features and additions. Our choice was SystemVerilog since it is a better Verilog which we learned in our previous courses. Knowledge of Verilog made it much easier even though we still considered VHDL as an option. SystemVerilog is a weakly typed, easy to use HDL which has lots of similarities with general purpose programming languages such as C++ or Java.

1.2 SignalTap Logic Analyzer

SignalTap Logic Analyzer is probably the most useful tool Quartus has. We think that without it our design would not be possible. Working on FPGA design is much more difficult than an ordinary programming due to the fact that FPGA designs are much harder to debug.

Debugging is usually done by writing a so called testbench which tests a written module for functionality. Another way to debug a design is to use a logic analyzer in this case SignalTap Logic Analyzer which is fitted inside the FPGA during the synthesis. STLA uses FPGA resources to implement a logic analyzer inside the FPGA which transmits user requested variables to the Quartus software using JTAG connection between the FPGA and PC. The information transmitted is later used to display to the user states of desired variables almost in



real time.

Figure 1.0 Sample STLA Output

In Figure 1.0 a sample output generated by STLA can be observed. Different variables of the left and corresponding values at different times can be seen on the diagram. Variables are chosen by the user and are usually ordinary variables of type logic defined in any module. Pin values of the FPGA can also be captured.

2. Interfacing the Audio Codec

After learning about essential software tools for the project the next step was to implement an interface for the audio codec. By interfacing we mean setting it up so that we are able to obtain digitized version of input analog signal coming from Line In 3.5mm input. The task is rather complicated since there are no obvious ways how to interface the codec and there are multiple approaches that can be taken. We first planned to use an Intellectual Property module also referred as IP core. These cores are like libraries in other languages such as C, they can be included into the project, but, unlike the libraries, most of them are distributed by Intel and all of them have a GUI which gives the ability to set up the IP core. We abandoned that idea and focused on raw FPGA approach which would only include pure HDL and nothing else.

We decided to use already written HDL code for the interface. The code belongs to Terasic and is part of one of the demonstration projects provided by Terasic to test different functionalities of the DE-1 board. This will be cited in the references.

Overall 5 modules implement the interface. Out of these modules, 3 which are “CLOCK_500”, “i2c” and “keytr” belong to Terasic. Other 2 which are “serializer” and “deserializer” are written by us.

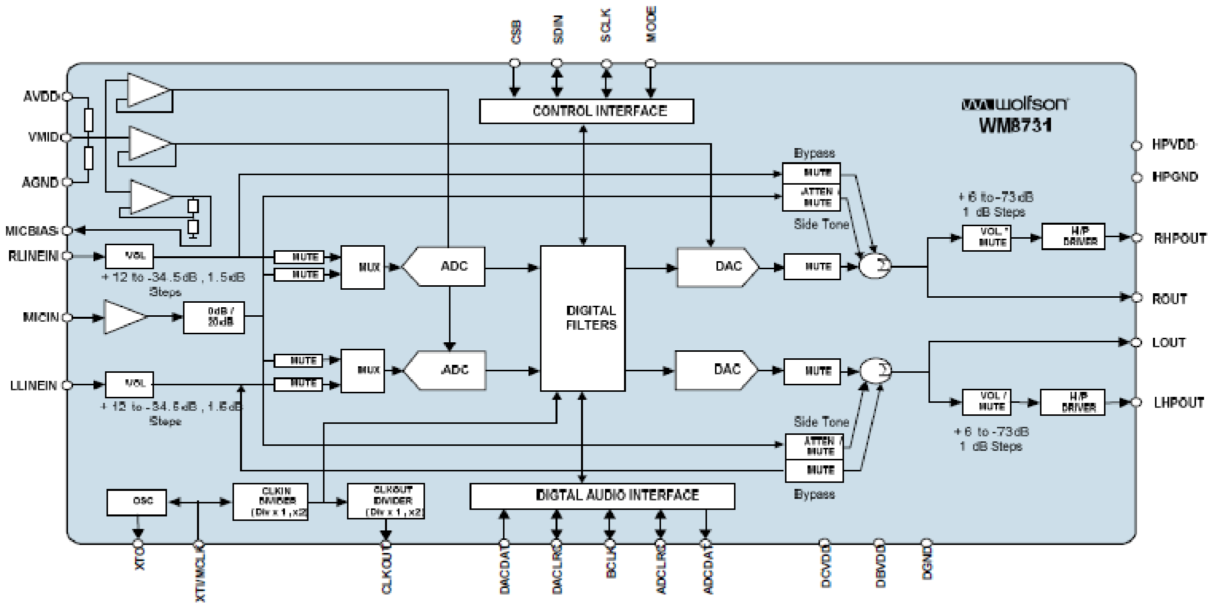


Figure 2.0 Block diagram for the audio codec

Figure 2.0 demonstrates the block diagram for the audio codec. Important parts to focus your attention on are Control Interface located at the top middle part of the diagram and Digital Audio Interface located at the bottom middle part of the diagram. We will come back to Digital Audio Interface later, for now we will discuss the implementation of Control Interface.

2.1 Control Interface

Modules “CLOCK_500” and “i2c” implement necessary I2C protocol to send different control settings. I2C is a popular protocol used for communication between various devices via only 2 lines which are clock SCL and data SDA lines. It implements master/slave hierarchy where master is the one that generates the clock and writes or reads the data from slave devices while the slave devices are the ones that respond to requests for write and read operations from master device.

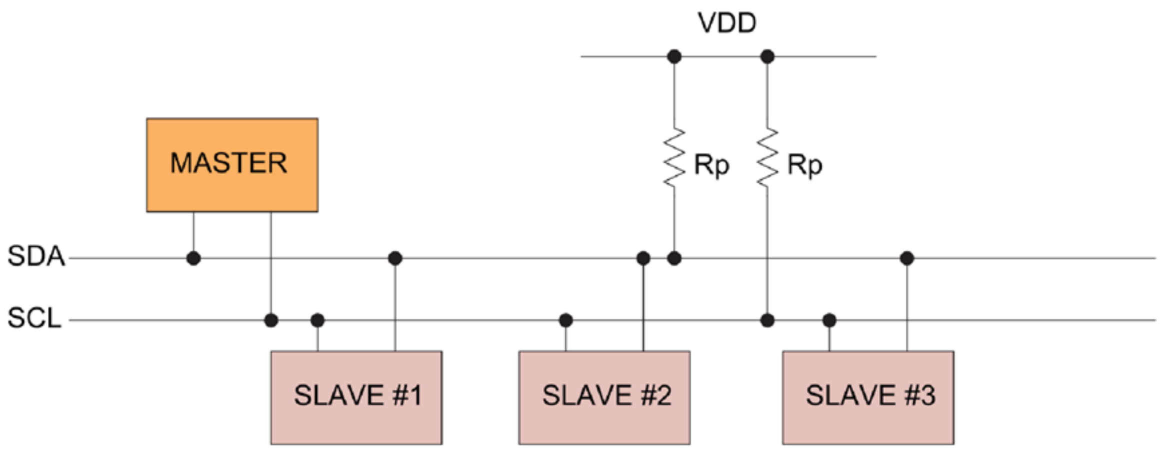


Figure 2.1 Interconnections in I2C protocol

The data that is being sent is arranged in 8-bit bytes comprising slave address, register number, and data to be transferred. The protocol initiates the communication with a start condition of SDA line when falling edge of SDA happens. The stop condition is the rising edge of SDA after all bits are received and counted and SDA stays high until next communication (same happens in other serial protocols like UART).

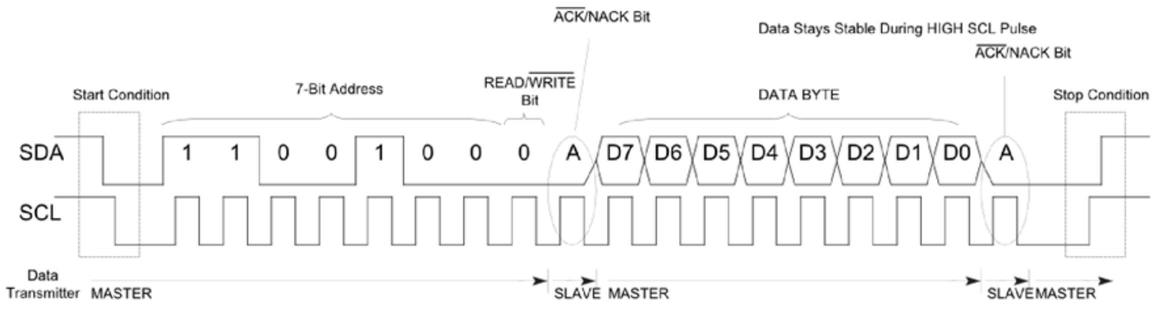


Figure 2.2 A single communication content of I2C

In Figure 2.2 it can be observed the start and stop conditions that were mentioned above. After the start condition comes 7-bit address which corresponds to the address of the slave device. Each slave device has its own unique address so for master to send the data to the correct slave device it needs its unique address which is transmitted in these 7-bits. After receiving these 7 bits each slave device compares if this address field matches its own address and discards the data if it doesn’t.

After 7-bit address comes 1-bit read/write bit which indicates if operation is read or write. After it comes an acknowledge bit. Since the SDA line is bidirectional which means that devices on both end can change its value that means that the slave device can also change it. So when it comes to the acknowledge bit the slave device is supposed to pull it low. If it is not set to low by the slave device the master terminates the communication or starts over (depends how the developer handles this situation).

If ACK is set to low then the communication continues and the actual data is being sent. It depends from configuration to configuration how many bits are sent. Afterwards another acknowledge is being checked and a stop condition occurs.

I2C is used by codec's control interface to transmit parameters of the codec on initialization (when the board is booted or the design is programmed). These parameters can be looked up in the documentation of the codec, but some of those are: the bit depth which is set to 16 bits in our case, the sampling rate which is set to 48 kHz, the input which is Line In, the mode of transmission of the samples (discussed later) and so on.

2.2 Digital Audio Interface

Modules "serializer" and "deserializer" implement the most important part which is to accept and transmit the samples from and to the audio codec. They use the Digital Audio Interface 1-bit pins which are "DACDAT", "DACLRC", "BCLK", "ADCLRC" and "ADCDAT". "ADCDAT" is the digitized sample coming from the codec, "DACDAT" is the digital sample we send to the codec to convert it to analog output, and others are clocks used during transmission of these data.

There are two samples coming from "ADCDAT", the left channel and the right channel samples. Each sample is 16-bit long, signed integer sampled in PCM format. Each sample comes bit by bit serially and need to be converted to parallel, just like in UART communication, for example. For this, modules "serializer" and "deserializer" were written.

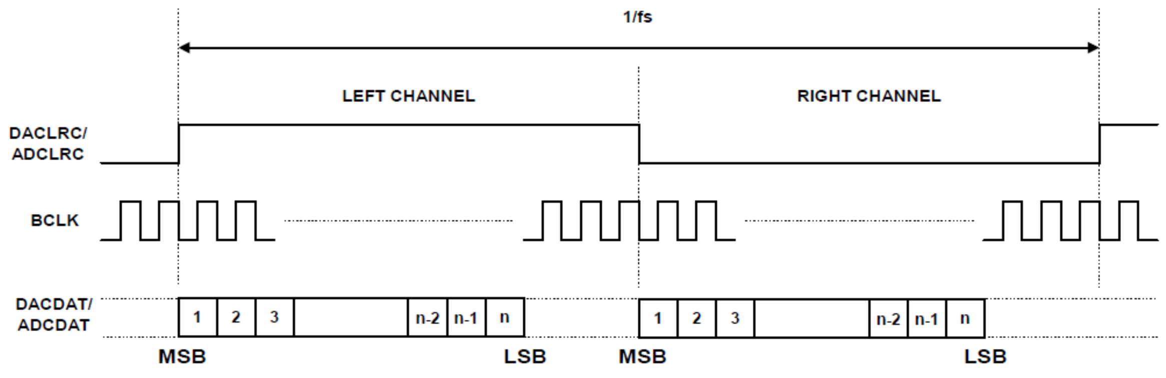


Figure 2.3 Left Justified Mode Transmission

Figure 2.3 describes how the transmission is handled. “BCLK” is an ordinary clock much slower when the base clock of the FPGA. “DACLRC” and “ADCLRC” are basically the same clock which dictate which sample is currently being transmitted. If they are high then the left sample is being transmitted, if low then the right one is transmitted. “DACDAT” and “ADCDAT” are being transmitted similarly, bit by bit from most significant bit to least significant. If we are receiving from the codec we capture these bits from “ADCDAT” pin in a 16-bit variable inside “deserializer” module and send this 16-bit sample to the effects modules. If we are transmitting a sample to the codec to convert it to analog we send the 16-bit sample bit by bit using “serializer” module.

With the sampling rate of 48 kHz we send and receive 48000 samples per second to and from the audio codec. Now that we are able to capture each sample we can proceed to implementing the effects.

3. Distortion

The term “distortion” describes different concepts depending on the field. In the field of Audio Signal Processing it often refers to an audio effect which manipulates an input audio signal adding extra harmonics to it. This effect can be achieved in different ways, but the approach that we chose is called Hard Clipping. It involves setting some arbitrary threshold in the range of minimum and maximum possible amplitudes of the signal which in this case with bit depth of 16 bits is from -32768 to 32767. Any sample that is outside of upper and lower thresholds is being clipped which means it is being set to the threshold value.

$$y[n] = \begin{cases} G * (-T), & x[n] < -T \\ G * x[t], & -T < x[n] < T \\ G * T, & x[n] > T \end{cases}$$

Where $x[n]$ is an input signal, $y[n]$ is an output signal, G is gain and T is the threshold. $y[n]$ is a non-linear system since it violates scaling property of linear systems. According to scaling property if an input is scaled by some value an output should also scale by proportionally, but this is violated in cases when an input signal exceed the threshold and is clipped. Since the system is non-linear it can’t be described using Fourier Transform or Z-Transform.

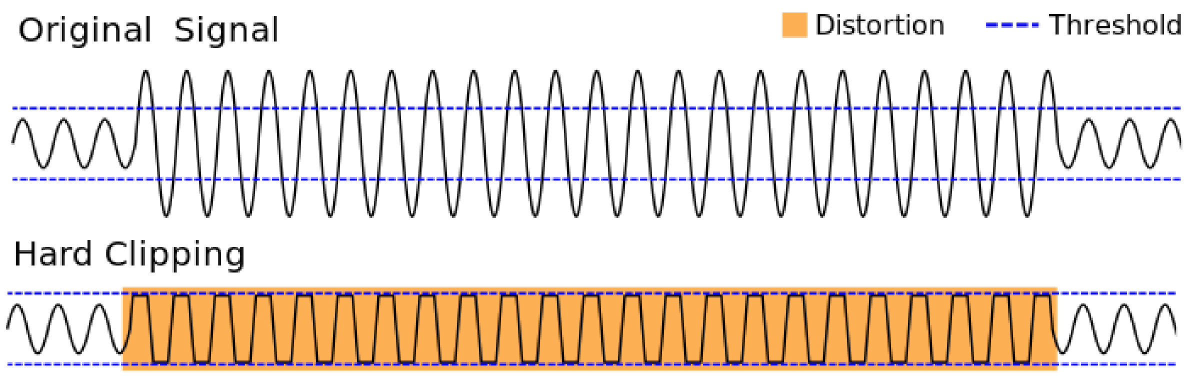


Figure 3.1 Result of Hard Clipping

Since we are cutting the signal in amplitude we are reducing it in volume. So if threshold is too low the audio volume will be low as well. To compensate that a gain needs to be added to the resulting signal to increase its resulting volume.

The distortion effect is being implemented by the “distortion” module. With simple SystemVerilog if/else statements we are able to compare input sample to the threshold and clip it in case the sample is outside the threshold. The module has two adjustable parameters which are the threshold value and the amount of gain. Both of these parameters are being controlled by another module “distortion_controller” which will be discussed later. The lower the threshold is set the harsher will be the effect and the lower the maximum amplitude. The higher the gain the louder the resulting signal will sound.

4. Tremolo

Tremolo is an audio effect which involves rapid linear increase and decrease of an input signal amplitude effectively producing “shuddering” effect. There are not many approaches how his can be done since it is pretty straightforward. First a linear triangle wave needs to be generated. It needs to oscillate from 0 to 1 and back from 1 to 0 linearly. The frequency of oscillation of this triangle wave for good quality shuddering effect needs to be in a range of 1 Hz to 10 Hz. This generated triangle needs to be multiplied by the input signal and the result will be the Tremolo effect.

$$y[n] = x[n] * x_{triangle}[n]$$

Where $y[n]$ is an output signal, $x[n]$ is an input signal and $x_{triangle}[n]$ is a triangle signal.

The system is not LTI system since it is time-varying because it involves the triangle wave which changes over time so applying the same input at different times can yield different results. Since it is not LTI system it can’t be described using either Fourier Transform or Z-Transform.

The module that implements Tremolo is “tremolo” module. It has a single adjustable parameter “frequency” which corresponds to the frequency of oscillation of the triangle wave. The parameter is controlled by another module “tremolo_controller” which will be discussed later. The triangle is generated inside the “tremolo” module using an always block which increments and decrements the triangle value from 0 to 128 and back. This triangle is then multiplied by the input signal and is divided by 128 effectively implementing fractional multiplication.

4. Ring Modulation

Similar to Tremolo effect, Ring Modulation involves multiplication of two signals, one being an input signal the other a generated high frequency sinusoid also called a carrier. Ring Modulation in a sense is an Amplitude Modulation which is used in AM radios to encode a signal by modulating it with a high frequency sinusoid (much higher than in Ring Modulation). Unlike AM radio modulation, Ring Modulation carrier frequency is in audible range. Human audible range is from 20 Hz to 20 kHz so the carrier should fall in that range.

$$y[n] = x[n] * \sin[2\pi f_c n]$$

Where $y[t]$ is an output signal, $x[t]$ is an input signal, f_c is the carrier frequency. The carrier frequency in out implementation can be varied from 80 Hz to 2000 Hz. The system is not LTI and can’t be described using Fourier Transform or Z-Transform since it is time varying because of the sinusoid.

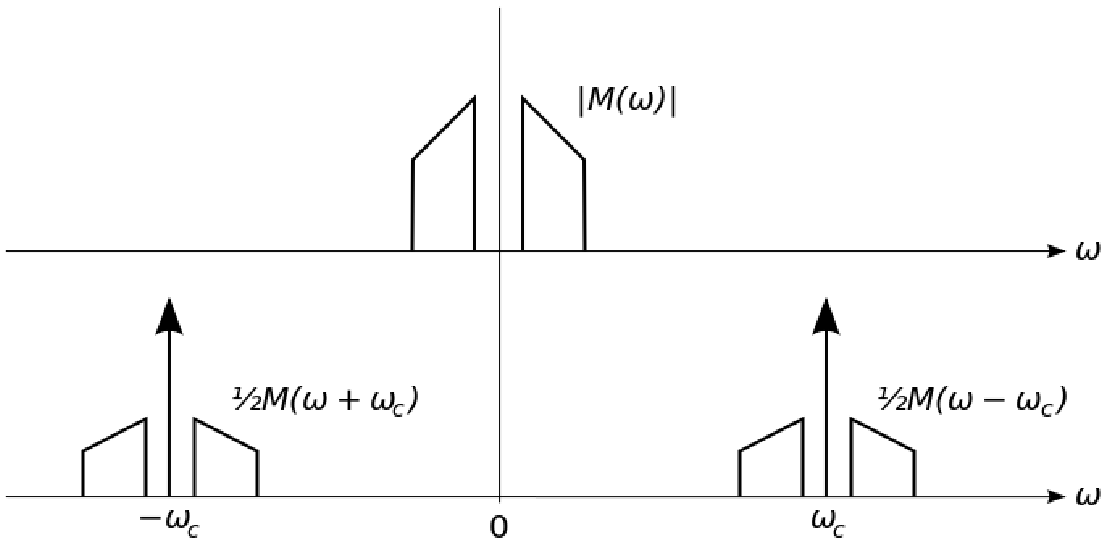
An interesting property of sinusoid multiplication should be noted:

$$\sin\alpha\cos\beta = \frac{\sin(\alpha+\beta)+\sin(\alpha-\beta)}{2}$$

$$\cos\alpha\cos\beta = \frac{\cos(\alpha+\beta)+\cos(\alpha-\beta)}{2}$$

$$\sin\alpha\sin\beta = \frac{\cos(\alpha-\beta)-\cos(\alpha+\beta)}{2}$$

Multiplication of sinusoids produces new sinusoids with sums and differences of frequencies of initial sinusoids. That property basically shifts a frequency component up and down on the frequency axis of the spectrum.



$|M(\omega)|$
ω
$\frac{1}{2}M(\omega + \omega_c)$
$\frac{1}{2}M(\omega - \omega_c)$
$-\omega_c$
0
ω_c
ω

Figure 4.1 Spectrum before and after modulation

In Figure 4.1 this shifting property can be observed. $M(w)$ is the spectrum before modulation, w is the angular frequency and w_c is the carrier's angular frequency. This property is heavily used even today to shift the band in spectrum for various needs. A modified signal after applying Ring Modulation in time domain looks like this:

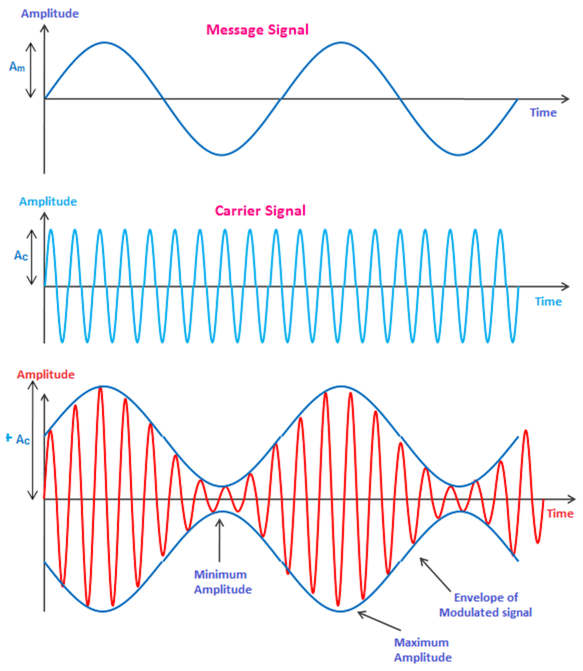


Figure 4.2 Input signal, carrier signal and modulated signal

The modules that implement Ring Modulation are “ring_modulation” and “sin_generator” module. The “ring_modulation” module receives a sinusoid signal generated in “sin_generator”. The sine wave is generated using lookup table. Sixteen samples of one period of a sine wave were recalculated and hardcoded into the module using basic case statements. An always block is used to continuously iterate over these 16 samples with a certain frequency effectively generating a sinusoid. The frequency of oscillation is a controllable parameter that can be changed by the user and is controlled by “ring_modulation_controller” module. The sine wave generated oscillates from -16 to 16 and is multiplied by an input sample and the result is divided by 16.

5. Echo

Echo represents an effect which adds delayed decaying repetitions of the input signal. In nature it is caused by the reflections of the sound from surfaces making replications of the initial sound to arrive with some delay.

5.1 Mathematical Representation

The following block diagram describes our Echo implementation:

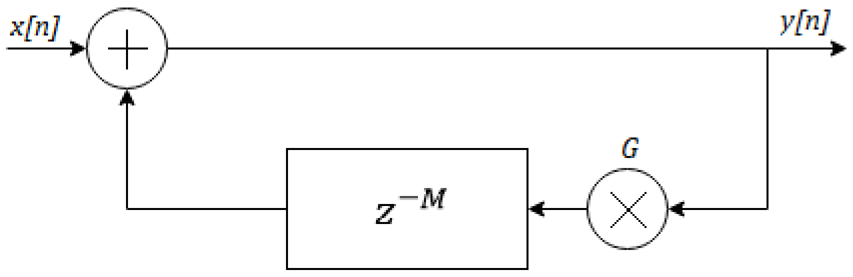


Figure 5.1 Block Diagram of the Echo effect

The following equation describes the block diagram in time domain:

$$y[n] = Gy[n - M] + x[n]$$

Where $y[n]$ is the output signal, $x[n]$ is the input signal, M is delay in samples and G is the gain.

Since the system is LTI it can be described using Z-Transform:

$$Y(z) = Gz^{-M}Y(z) + X(z)$$

A transfer function can be obtained:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - Gz^{-M}}$$

From the transfer function we can derive a single pole at G. Since the input sequence is right sided: ROC : $|z| > |G|$. For the system to be stable the ROC needs to include the unit circle thus: $0 < G < 1$.

The above conclusion can be reached by observation of the system equation in the time domain. If the gain is above 1 the output will keep summing up to infinity, in hardware it will result in repeating overflows since the maximum possible positive value of a sample is 32767. If the gain is less than 1 the output will decay with time. In our implementation gain is adjustable and can't be set to more than 1.

5.2 Delay Line implementation in hardware

From the equation of the Echo effect we can conclude that we need to delay an output signal by M samples and add it to the input signal. Since the system is real-time there's no direct way to access previous samples since they "come and go". For that purpose a specific hardware element called FIFO which refers to "First In First Out" needs to be implemented. FIFO is a type of buffer that can be implemented using RAM or Random Access Memory. Fortunately Cyclone 5 FPGA has internal memory blocks also referred as BRAM for the user to access and use. To access on FPGA memory a specific type of module needs to be written which follows strict template.

```
//SystemVerilog code for BRAM
module ram(
	output signed [15:0] Q,
	input signed [15:0] D,
	input [15:0] write_address,
	input [15:0] read_address,
	input W_E,
	input CLK
	);
```

```
      parameter WIDTH = 16;
      parameter DEPTH = 65536;

      logic signed [WIDTH-1:0] mem [DEPTH-1:0];

      always @(posedge CLK)begin
            if(W_E)begin
                  mem[write_address] <= D;
            end
            Q <= mem[read_address];
      end
endmodule
```

Components of the FPGA accessed in this way are called “Inferenced Components”. When this module is compiled the synthesizer will automatically instantiate the BRAM components in the FPGA.

From the above code we can conclude that the memory is synchronous because of the clock and means that a single data at a certain address can be accessed only once per clock cycle. The FIFO is implemented using read and write address pointers which on each new sample increment by 1. These pointers iterate over the range of 0 to max ram size which in this case is 65536. Since read happens before write in an always block we are able to read past samples before writing the new ones to the address. The clock used for read and write is the 48 kHz clock since there is no point reading and writing faster than the sampling rate.

5.3 Echo – Conclusion

The Echo effect is implemented using modules “echo” and “ram”. The amount of gain and delay time is adjustable and is controlled by another module “echo_controller”. The gain is exponential from 2^(-1) to 2^(-6). The delay time can be set from 2047 samples to 65535 samples. To calculate the delay in seconds we need to divide amount of delay samples by the sampling rate of 48k. So for 65535 value of M the delay in seconds will be 1.365 seconds.

6. Vibrato

The Vibrato audio effect is in implementation very similar to the Echo effect. It describes sinusoidal shift in frequency of the input signal spectrum. Having minor modifications to the Echo effect we can obtain the Vibrato effect. The equation that describes vibrato is the following:

$$y[n] = Gy\left[n - M * \frac{1}{2}(\sin[2\pi f_c n] + 1)\right] + x[n]$$

Where $y[n]$ is the output signal, $x[n]$ is the input signal, M is delay in samples, G is the gain and f_c is the carrier frequency. This equation describes the modulation of the delay line M by a sinusoid. This effectively creates time-varying delay line. Since we have a time-varying delay line the system is not LTI. The block diagram for this equation is:

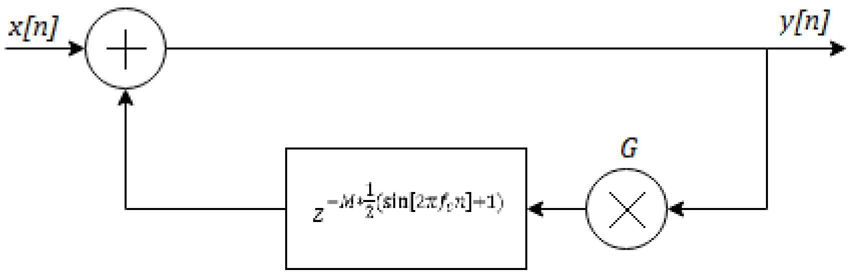


Figure 6.1 Block diagram for the Vibrato effect

Another FIFO similar to FIFO of Echo effect is used. The delay line is time-varying from 23 to 263 samples which in seconds is from 0.48 ms to 5.48 ms. The frequency of the sinusoid is adjustable from 1 Hz to 10 Hz and is controlled by another module "vibrato_controller". The Vibrato effect is implemented using 3 modules which are "vibrato", "ram" and "sin_generator_HD".

7. Slapback

The Slapback effect is another time delay effect which is yet again similar in implementation to the Echo effect. It manipulates the input signal by adding to it delayed version of the signal. Unlike Echo the delayed signal is added to the input signal only once.

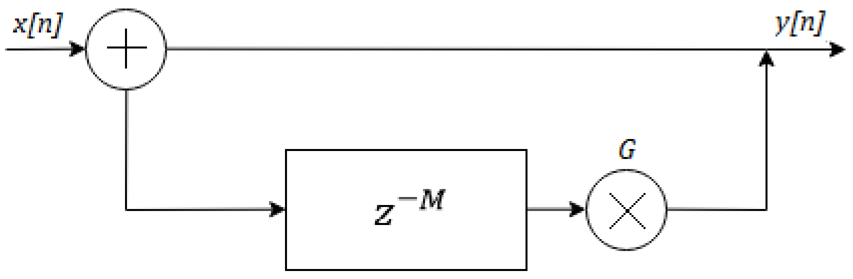


Figure 7.1 Block diagram for the Slapback effect

The following equation describes the Slapback effect:

$$y[n] = Gx[n - M] + x[n]$$

Where $y[n]$ is the output signal, $x[n]$ is the input signal, M is delay in samples and G is the gain. The key difference here from Echo effect is that we delay the input and not the feedback output. The system is LTI and can be described using Z-Transform:

$$Y(z) = Gz^{-M}X(z) + X(z)$$

The transfer function is:

$$H(z) = \frac{Y(z)}{X(z)} = Gz^{-M} + 1$$

There are no poles in the transfer function so ROC includes all values of z. That means that it also includes the unit circle and thus the system is stable for any value of G. Since our system is implemented in hardware it is better to restrict G to be always positive: $G > 0$.

Overall, the effect is implemented using 2 modules: “slapback” and “ram”. The effect has a single adjustable parameter of delay time M in samples. M can be varied from 512 to 16384 samples which in seconds is equivalent to 10 to 341 ms time delay. This parameter is controlled by another module “slapback_controller”.

8. User Interface

The project also includes an intuitive user interface to provide the user ability to control above mentioned parameters for each module. The UI makes usage of onboard 10 switches, 4 buttons, 6 seven segment displays and 10 LEDs. Overall 10 modules and a lot of wiring in the schematic takes care of the whole UI implementation.

Ten switches on the board allow to manipulate the effects applied to the signal. Six switches from switch 9 to switch 4 (including 4) enable or disable certain effect. The other 4 switches from switch 2 to 0 allow to choose the parameter to modify. Switch number 3 is not used. The two buttons, key 3 and 2 increase or decrease selected parameter value. Key 0 should be pressed once to increase the output volume after the board boot up or as soon as the design is programed into the FPGA.

The following table describes the effects and their corresponding enabling switch.

Effect Name	**Switch Value**
Distortion	ON: SW[9] = 1 OFF: SW[9] = 0
Tremolo	ON: SW[8] = 1 OFF: SW[8] = 0
Ring Modulation	ON: SW[7] = 1 OFF: SW[7] = 0
Echo	ON: SW[6] = 1 OFF: SW[6] = 0
Vibrato	ON: SW[5] = 1 OFF: SW[5] = 0
Slapback	ON: SW[4] = 1 OFF: SW[4] = 0

Table 8.1 Effects and their corresponding enabling switch values

The following table describes the possible adjustable parameters and corresponding switch values.

SW[2:0] Value Dec	**SW[2:0] Value Bin**	**Parameter Description**	**Parameter Range**
0	000	Distortion threshold	20 : 32767
1	001	Distortion gain	1 : 50
2	010	Tremolo triangle frequency	1 : 10 Hz
3	011	Ring Modulation carrier frequency	20 : 2000 Hz
4	100	Echo delay in samples	42 ms to 1.365 s
5	101	Echo gain	1 : 6
6	110	Vibrato sinusoid frequency	1 : 10 Hz
7	111	Slapback delay in samples	10 to 341 ms

Table 8.2 Parameters and their corresponding switch values and ranges

The UI is implemented by 9 modules. Modules implementing the above functionality are: “main_7seg_controller”, “dec7seg”, “distortion_controller”, “tremolo_controller”, “ring_modulation_controller”, “echo_controller”, “vibrato_controller”, “slapback_contoller”. Each effect modules has corresponding controller modules which receives inputs from switches and buttons and generates corresponding module parameter. Modules: “main_7seg_controller” and “dec7seg” take care of displaying the current selected parameter value on the 6 seven segment displays.

Special module “amplitude2leds” was written. It displays current output signal amplitude on the 10 LEDs. A 16-bit sample is mapped to 10-bit binary number which is later displayed through the LEDs. To map we divide a 16-bit sample by 2^6 which effectively reduces it to a proportional 10-bit number. The samples are displayed on LEDs in real-time so any change for output signal at any time can be observed on these LEDs.

9. Compilation Report

The following compilation report was generated by Quartus synthesizer:

Flow Status	Successful - Tue May 21 14:03:37 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	FPGA_Audio_Effects
Top-level Entity Name	main
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	13,514 / 32,070 (42 %)
Total registers	1080
Total pins	76 / 457 (17 %)
Total virtual pins	0
Total block memory bits	1,302,528 / 4,065,280 (32 %)
Total DSP Blocks	22 / 87 (25 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Table 9.1 Compilation Report

Time it took to compile the design was 12 minutes and 12 seconds.

10. Conclusion

To conclude, despite the difficulties we faced during the design of the Audio Effects part of this project we managed to build satisfying design which we planned from the beginning. The final design, as can be concluded from the compilation report, doesn’t require too many FPGA resources to be synthesized. The resulting design presents easy to use 6 adjustable audio effect with an intuitive user interface.

11. References

Zölzer, U. (2011). *DAFX: Digital Audio Effects*. John Wiley & Sons, Ltd.

Wang, A. (2010). *DE1_SoC_i2sound Demonstration Project.* Retrieved from https://www.terasic.com.tw/cgibin/page/archive.pl?Language=English&CategoryNo=205&No=836&PartNo=4

S. Disch. (1999). *Digital audio effects – modulation and delay lines.* Technical University Hamburg-Harburg.