|  |
| --- |
| the american university IN cairo |
| Digital Audio Processor |
| Digital Design I |
| Final Project Report  Farida Soliman  Hossam Samy  Maged Mekawy  Yasmin Sami  Ma |
|  |
| **22-May-15** |

|  |
| --- |
|  |

Tools:

* Nexsys 3 FPGA board
* PModMIC
* PModI2S

1. Clocks:

The clock frequencies were calculated according to the mic reference manual. The master clock is 12.288 MHz, LRclk (sampling frequency) is 32KHz and the serial clock has to be 48 times the LR clock according to the manual. The Pmod changes the clock in the board so we were not able to calculate the new clocks directly in the module and we had to use an IP core Gen clock generator which did not produce exactly what we needed so its output was manipulated to produce the right clock frequencies.

1. Echo:

The echo was implemented using an IP Core Gen FIFO. Its size was calculated using a sampling frequency of 32KHz and a delay of 200 ms. The echo module receives the sample from the mic and inputs it in FIFO serially and the output is sent to the I2S.

Inputs:

* Mic Data
* Serial input clock
* Reset

Output:

* I2S data.

1. Changing Pitch:

Changing pitch required calculating cosines which we tried to implement by writing a C++ code to generate the values of the cosine function and then attempted to store quarter the values on an IP Core Gen ROM, however, the memory on the board was not enough to store it. We then attempted to approximate the cosine directly in Verilog by using a Taylor Series and approximating the value of pi to 3. This method created an overflow in the output. The logic used was verified by implementing the same logic in C++ and using an input and the code worked and produced a higher pitch.

Inputs:

* Mic Data
* Serial input clock
* Reset

Output:

* I2S data

1. Low Pass Filter:

Our implementation of the low pass filter is based on averaging every 16 samples using shifting and then the averaged value is the new signal. So the module take a sample at each positive edge clock and stores 16 samples by removing the oldest sample and storing the new one. If the low pass filter switch is on the module will work. The end result is that low frequencies will be heard while high frequencies will decrease in volume. The ideal is to totally eliminate high frequencies but due to hardware restrictions, it wasn't possible due to a lot of averaging for constants.

Inputs:

* Mic Data
* Serial input clock
* Reset

Output:

* I2S data