# ELEX 7660: Digital System Design

# Voice Shifting


## *Abstract*

This report documents our process in designing a voice shifting system that takes user voice input and shifts it to a different frequency, and outputs a new voice that has been modified to sound like a man, a woman, or a teen. In addition, this report will explains how we implement and connect each module together. All codes and diagrams that was used to implement this system will also be included.



| Prepared by: | Date |
|---|---|
| *Brendon Soberano* | April 12, 2018 |
| *Siek Meng Teng* | |

# Table of Contents

1 Introduction ..... 3
2 Background ..... 3
3 List of Modules ..... 3
3.1 Analog to Digital Converter ..... 3
3.2 Shifter ..... 4
3.3 Pulse Width Modulation ..... 4
3.4 Low Pass Reconstruction Filter ..... 5
3.5 Control Toggle ..... 5
4 Diagrams ..... 5
5 Conclusion ..... 6
6 Code ..... 7
6.1 PWM ..... 7
6.2 LPF ..... 7
6.3 Keypad ..... 10
6.4 ADC ..... 11
7 Bibliography ..... 13

# Table of Figures

Figure 1 - Diagram of ADC pins ..... 3
Figure 2 - State Diagram of count ..... 4
Figure 3 - State Diagram of dataout ..... 4
Figure 4 - Layout of Voice Shifter Design ..... 5

# 1 Introduction

This project is prepared for the ELEX 7660 - Digital System Design course and is presented to Dr. Eduardo Casas. This project is designed to create a voice shifter that converts an ordinary voice into a changed and distorted voice. We used the DE0-Nano FPGA and used System Verilog as our choice of programming language. Hardware included in this project are a keypad push button, an electret microphone and an electric speaker. There are three settings to allow the user to voice shift:

- Deep man voice
- High-pitched female voice, and
- Teen voice

The push buttons allow to freely change to one of the three voice shifting choices. The speaker output will create a sound that will be a changed version of the user's voice.

# 2 Background

Voice shifting is a way to change the tone, speed, or distortion of a waveform. One can use audio programs to change its voice settings. The waveform of the song or voice has either its points stretched out from each other, some squeezed in removing points in between, or add in noise or shift the data points to a different value.

# 3 List of Modules

## 3.1 Analog to Digital Converter

The ADC module converts analog inputs (voice) into a digital signal in order to easily modify the signal to change the pitch of the voice. The ADC is implemented by using the built in ADC pins located on the FPGA. The DE0-Nano FPGA contains a low power, eight-channel CMOS 12-bit ADC.

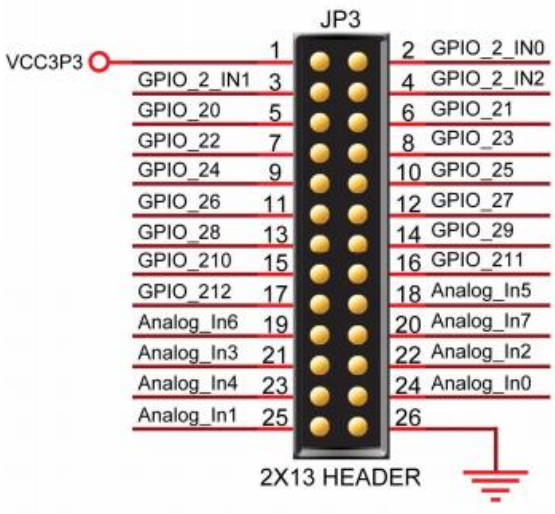


*Figure 1 - Diagram of ADC pins*

## 3.2 Shifter

The Shifter module will take the audio signals from the microphone. This audio signal is then stretched by a factor of n, which is the specified voice shifting factor. There are three pre-set voice shifting factor, which correspond to the push buttons that were assigned for each effect. Each of the shifting factor will be decided based on three different type of voices, a deep man voice, a high female voice, and a teen voice. For instance, when time stretching the original audio signal by a factor of n, if the deep man voice setting is selected, the new audio signal will be less than one, which means the pitch is decreased. By time stretching a signal first and resampled it at higher speed instead of increasing the play speed, we are able to achieve a shift in pitch while maintaining the original duration of the signal.

## 3.3 Pulse Width Modulation

After our signal has been shifted, we need to convert that signal from digital signal back to analog signal before it could be outputted through the speaker. To implement this ADC, we use pulse width modulation to modulate the signal and run it through a low pass reconstruction filter to create an analog signal. Pulse width modulation is an input control pulse width. To implement this, we use counter to count from 2^16 = 65536 to 0. When the counter equal to the input signal, it outputs a zero. And when the counter reaches zero, it will output a one again.

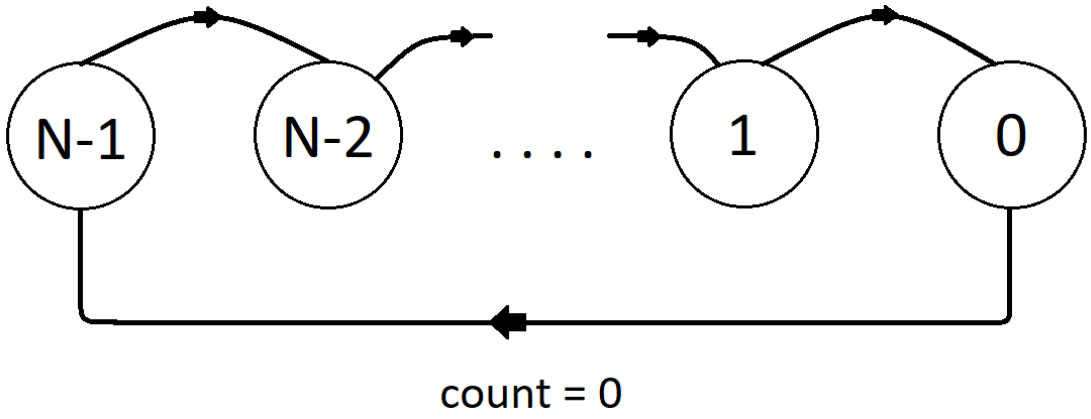


*Figure 2 - State Diagram of count*

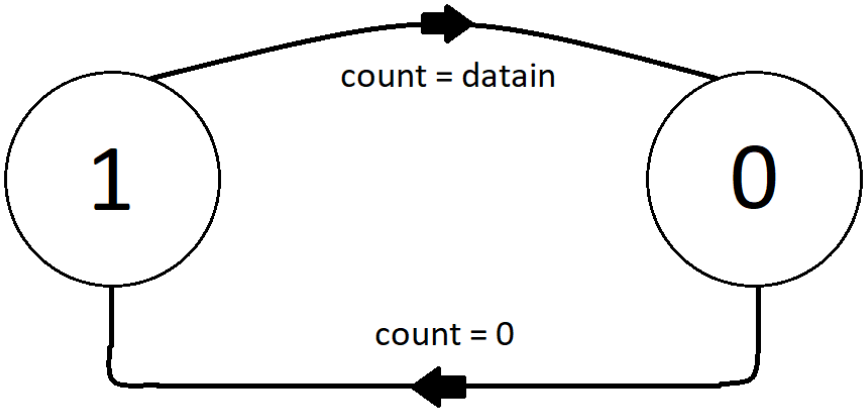


*Figure 3 - State Diagram of dataout*

## 3.4 Low Pass Reconstruction Filter

A low pass reconstruction filter is needed in order to convert the pulse width modulation signal to analog signal. To implement this low pass filter, we uses the built-in FIR filter from Altera Quartus IP catalog.

## 3.5 Control Toggle

The control module takes the input from the keypad to select the corresponding voice shifting setting for the voice shifting module. When the keypad is pressed, the corresponding coefficient is set and sent to the voice shifting module, so that the input voice signal can be time stretched to achieve desired effect.

# 4 Diagrams

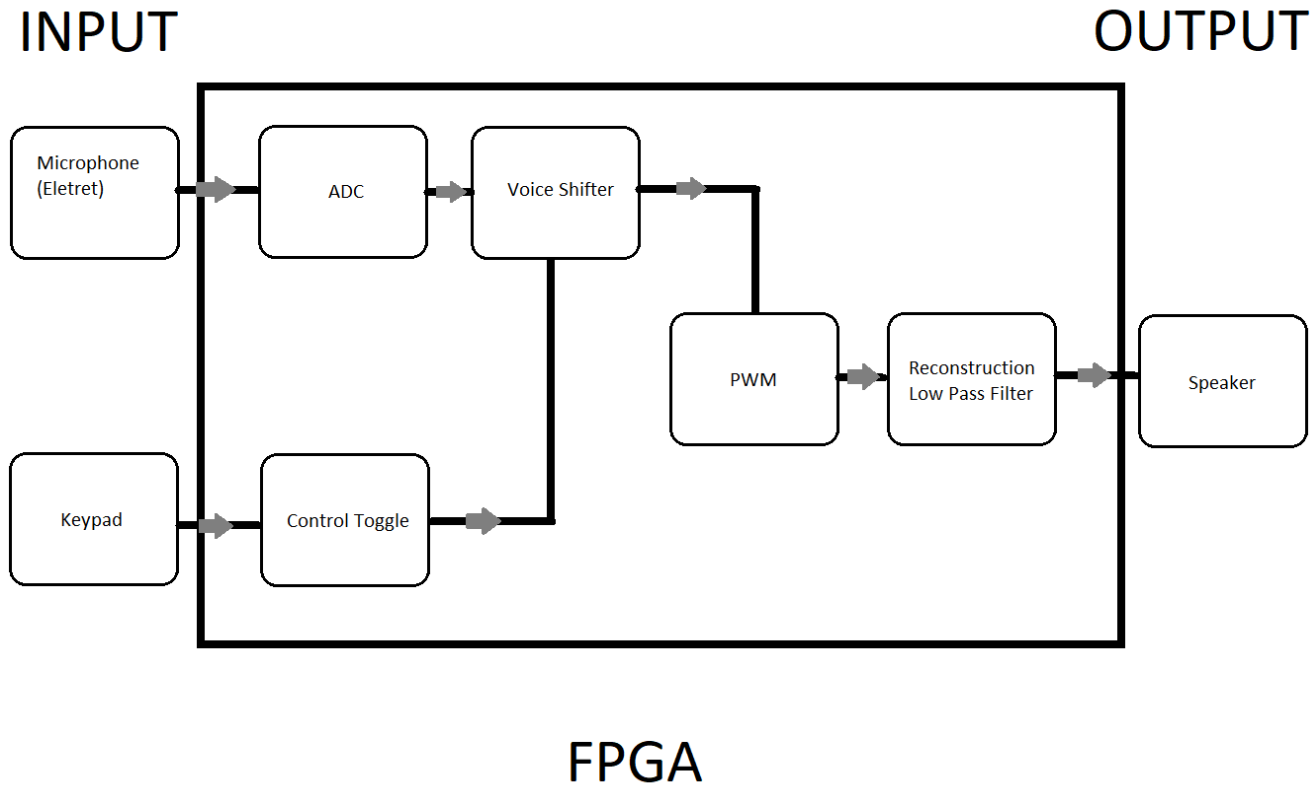


*Figure 4 - Layout of Voice Shifter Design*

This diagram is the layout of the design. The modules will be set out so that the speaker output lines up. While the layout of the microphone to speaker is designed for the voice shifter, the keypad provides changing the voice modulation.

[Image of Platform Designer Schematic]

This diagram shows the top-level block diagram for this project.

# 5 Conclusion

This report shows the software design required to build and create a voice shifter. While the project is simple to implement, there are additions that could be made in the future. One would be adding an indicator to visibly show that the user is using this voice shift option. Also, an analog freeform shifter could be implemented. This uses an analog controller, such as a joystick or knob, to freely change the shifter to the user's own custom voice shift.

# 6 Code

## 6.1 PWM

```
module pwm
                  #( N = 65536)
                  (input logic reset, clk,
                   input logic [16:0] datain,
                   output logic dataout);

      logic [$clog2(N):0] count = N;

      logic [$clog2(N):0] count_next;

      always_comb begin

            count_next = count;

            if (reset) begin
                  count_next = N - 1;
                  dataout = 1;
            end

            else begin

                  if (count_next == 0) begin
                        dataout = 1;
                        count_next = N - 1;
                  end

                  else if (count_next == datain) begin
                        dataout = 0;
                        count_next = count - 1;
                  end

                  else begin
                  count_next = count - 1;
                  end

            end
      end

      always @(posedge clk)
            count <= count_next;

endmodule
```

## 6.2 LPF

```
// megafunction wizard: %FIR II v17.1%
// GENERATION: XML
// LowPass.v

// Generated using ACDS version 17.1 590
```

```
`timescale 1 ps / 1 ps
module LowPass (
            input  wire          clk,              //                      clk.clk
            input  wire          reset_n,          //
rst.reset_n
            input  wire [999:0]  ast_sink_data,    //   avalon_streaming_sink.data
            input  wire          ast_sink_valid,   //
.valid
            input  wire [1:0]    ast_sink_error,   //
.error
            output wire [2399:0] ast_source_data,  // avalon_streaming_source.data
            output wire          ast_source_valid, //
.valid
            output wire [1:0]    ast_source_error  //
.error
      );

      LowPass_0002 lowpass_inst (
            .clk              (clk),              //                      clk.clk
            .reset_n          (reset_n),          //
rst.reset_n
            .ast_sink_data    (ast_sink_data),    //   avalon_streaming_sink.data
            .ast_sink_valid   (ast_sink_valid),   //                           .valid
            .ast_sink_error   (ast_sink_error),   //                           .error
            .ast_source_data  (ast_source_data),  // avalon_streaming_source.data
            .ast_source_valid (ast_source_valid), //                           .valid
            .ast_source_error (ast_source_error)  //                           .error
      );

endmodule
// Retrieval info: <?xml version="1.0"?>
//<!--
//    Generated by Altera MegaWizard Launcher Utility version 1.0
//    ************************************************************
//    THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//    ************************************************************
//    Copyright (C) 1991-2018 Altera Corporation
//    Any megafunction design, and related net list (encrypted or decrypted),
//    support information, device programming or simulation file, and any other
//    associated documentation or information provided by Altera or a partner
//    under Altera's Megafunction Partnership Program may be used only to
//    program PLD devices (but not masked PLD devices) from Altera.  Any other
//    use of such megafunction design, net list, support information, device
//    programming or simulation file, or any other related documentation or
//    information is prohibited for any other purpose, including, but not
//    limited to modification, reverse engineering, de-compiling, or use with
//    any other silicon devices, unless such use is explicitly licensed under
//    a separate agreement with Altera or a megafunction partner.  Title to
//    the intellectual property, including patents, copyrights, trademarks,
//    trade secrets, or maskworks, embodied in any such megafunction design,
//    net list, support information, device programming or simulation file, or
//    any other related documentation or information provided by Altera or a
//    megafunction partner, remains with Altera, the megafunction partner, or
//    their respective licensors.  No other licenses, including any licenses
//    needed under any third party's intellectual property, are provided herein.
//-->
// Retrieval info: <instance entity-name="altera_fir_compiler_ii" version="17.1" >
// Retrieval info:      <generic name="filterType" value="single" />
// Retrieval info:      <generic name="interpFactor" value="1" />
```

```
// Retrieval info:      <generic name="decimFactor" value="1" />
// Retrieval info:      <generic name="symmetryMode" value="nsym" />
// Retrieval info:      <generic name="L_bandsFilter" value="1" />
// Retrieval info:      <generic name="inputChannelNum" value="1" />
// Retrieval info:      <generic name="clockRate" value="1" />
// Retrieval info:      <generic name="clockSlack" value="0" />
// Retrieval info:      <generic name="inputRate" value="100" />
// Retrieval info:      <generic name="coeffReload" value="false" />
// Retrieval info:      <generic name="baseAddress" value="0" />
// Retrieval info:      <generic name="readWriteMode" value="read_write" />
// Retrieval info:      <generic name="backPressure" value="false" />
// Retrieval info:      <generic name="deviceFamily" value="Cyclone V" />
// Retrieval info:      <generic name="speedGrade" value="medium" />
// Retrieval info:      <generic name="delayRAMBlockThreshold" value="20" />
// Retrieval info:      <generic name="dualMemDistRAMThreshold" value="1280" />
// Retrieval info:      <generic name="mRAMThreshold" value="1000000" />
// Retrieval info:      <generic name="hardMultiplierThreshold" value="-1" />
// Retrieval info:      <generic name="reconfigurable" value="false" />
// Retrieval info:      <generic name="num_modes" value="2" />
// Retrieval info:      <generic name="reconfigurable_list" value="0" />
// Retrieval info:      <generic name="MODE_STRING" value="None Set" />
// Retrieval info:      <generic name="channelModes" value="0,1,2,3" />
// Retrieval info:      <generic name="inputType" value="int" />
// Retrieval info:      <generic name="inputBitWidth" value="10" />
// Retrieval info:      <generic name="inputFracBitWidth" value="0" />
// Retrieval info:      <generic name="coeffSetRealValue"
value="0.0176663,0.013227,0.0,-0.0149911,-0.0227152,-
0.0172976,0.0,0.0204448,0.0318046,0.0249882,0.0,-0.0321283,-0.0530093,-
0.04498,0.0,0.0749693,0.159034,0.224907,0.249809,0.224907,0.159034,0.0749693,0.0,-
0.04498,-0.0530093,-0.0321283,0.0,0.0249882,0.0318046,0.0204448,0.0,-0.0172976,-
0.0227152,-0.0149911,0.0,0.013227,0.0176663" />
// Retrieval info:      <generic name="coeffSetRealValueImag" value="0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0530093, -0.04498, 0.0,
0.0749693, 0.159034, 0.224907, 0.249809, 0.224907, 0.159034, 0.0749693, 0.0, -
0.04498, -0.0530093, -0.0321283, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0" />
// Retrieval info:      <generic name="coeffScaling" value="auto" />
// Retrieval info:      <generic name="coeffType" value="int" />
// Retrieval info:      <generic name="coeffBitWidth" value="8" />
// Retrieval info:      <generic name="coeffFracBitWidth" value="0" />
// Retrieval info:      <generic name="coeffComplex" value="false" />
// Retrieval info:      <generic name="karatsuba" value="false" />
// Retrieval info:      <generic name="outType" value="int" />
// Retrieval info:      <generic name="outMSBRound" value="trunc" />
// Retrieval info:      <generic name="outMsbBitRem" value="0" />
// Retrieval info:      <generic name="outLSBRound" value="trunc" />
// Retrieval info:      <generic name="outLsbBitRem" value="0" />
// Retrieval info:      <generic name="bankCount" value="1" />
// Retrieval info:      <generic name="bankDisplay" value="0" />
// Retrieval info: </instance>
// IPFS_FILES : LowPass.vo
// RELATED_FILES: LowPass.v, dspba_library_package.vhd, dspba_library.vhd,
auk_dspip_math_pkg_hpfir.vhd, auk_dspip_lib_pkg_hpfir.vhd,
auk_dspip_avalon_streaming_controller_hpfir.vhd,
auk_dspip_avalon_streaming_sink_hpfir.vhd,
auk_dspip_avalon_streaming_source_hpfir.vhd, auk_dspip_roundsat_hpfir.vhd,
altera_avalon_sc_fifo.v, LowPass_0002_rtl_core.vhd, LowPass_0002_ast.vhd,
LowPass_0002.vhd
```

## 6.3 Keypad

```
module kpdecode (input logic [3:0] kpc, kpr,
                 input logic clk,
                 output logic [3:0] num,
                 output logic kphit) ;

      logic [3:0] num_next;
      logic kphit_next;


      always_comb begin
      kphit_next = kphit;
      num_next = num;

            //Check if kphit is pressed

            if (kphit) begin
            kphit_next = 1;

                  //Case by case method for each kpc bit
                  if (kpr == 4'b0111)
                        case(kpc)
                        4'b1110: num_next = 4'ha;
                        4'b1101: num_next = 4'h3;
                        4'b1011: num_next = 4'h2;
                        4'b0111: num_next = 4'h1;
                        default: num_next = 4'h0;
                        endcase
                  else if (kpr == 4'b1011)
                        case(kpc)
                        4'b1110: num_next = 4'hb;
                        4'b1101: num_next = 4'h6;
                        4'b1011: num_next = 4'h5;
                        4'b0111: num_next = 4'h4;
                        default: num_next = 4'h0;
                        endcase
                  else if (kpr == 4'b1101)
                        case(kpc)
                        4'b1110: num_next = 4'hc;
                        4'b1101: num_next = 4'h9;
                        4'b1011: num_next = 4'h8;
                        4'b0111: num_next = 4'h7;
                        default: num_next = 4'h0;
                        endcase
                  else if (kpr == 4'b1110)
                        case(kpc)
                        4'b1110: num_next = 4'hd;
                        4'b1101: num_next = 4'hf;
                        4'b1011: num_next = 4'h0;
                        4'b0111: num_next = 4'he;
                        default: num_next = 4'h0;
                        endcase
                  else
                        num_next = num;
                  end

            else if (!kphit) begin
```

```
                    num_next = num;
                    kphit_next = 1;
              end

        end

        always_ff @(posedge clk) begin
              num <= num_next;
              kphit <= kphit_next;
        end

endmodule
```

## 6.4 ADC

```
// Code implemented by Ed.Casas 2017-2-14

// Instatiate an SPI master for the DE0-Nano ADC. Reads from
   // ADC SPI pins, outputs to a ready/valid interface that feeds
      // 'myfifo'.

   logic ready ;
   logic valid ;
   logic [31:0] data ;
   logic reset_n, clk ;

   assign clk = CLOCK_50 ;
   assign reset_n = KEY[0] ;

   adcspi a0
     (
      .sclk(ADC_SCLK), .mosi(ADC_SADDR), .ssn(ADC_CS_N), // SPI master
      .miso(ADC_SDAT),

      .ready(ready),
      .valid(valid),          // data out
      .data(data),

      .clk(clk), .reset(~reset_n) ) ;

   // copy MS ADC bits to LEDs for debug
   assign LED = { data[27:24], data[11:8] } ;

endmodule

// -- start of adcspi.sv ---


// reads channels 0 and 1
// sclk is clk is divided by 16
// output is 16-bit samples from channels 0 and 1
// samples packed into 32 bits (ch 0 in MS byte)

// ADC128S0022 interface:
```

```
// 16 bit transfers

// mosi and cs* change on falling edge of sclk
// mosi bits 13:11 are (next) channel number

// miso sampled on rising edge of sclk
// miso data is on ls 12 bits of miso

// sample rate is sclk rate / 16
// sample rate must be 50 to 200 kHz
// sclk rate must be 800 kHz to 3.2 MHz
// e.g. 50 MHz / 32 = 1.5625 MHz sclk, ~98kHz sampling

// mosi timing relative to rising edge of sclk:
// setup is >10ns, hold  >10ns

// miso timing is relative to falling edge of sclk:
// access is <27ns, hold ~4ns

module adcspi
   (
    output logic sclk, mosi, ssn, // SPI master
    input logic miso,

    input logic ready,          // ready/valid data out
    output logic valid,
    output logic [31:0] data,

    input logic clk, reset ) ;

   parameter MISO = {5'b00001,27'b0} ;

   // clock/bit counter
   struct packed {
      logic wordcnt ;
      logic [3:0] bitcnt ;
      logic sclk ;
      logic [3:0] clkcnt ; } cnt, cnt_next ;

   logic [31:0] sr ;            // shift register

   logic rising, falling, done ;

   assign sclk = cnt.sclk ;

   // done all bits
   assign done = cnt ==? '{'1,'1,'1,'1} ;

   // clock/bit counter
   assign cnt_next = ( reset || done ) ? '0 : cnt+1'b1 ;
   always@(posedge clk)
     cnt <= cnt_next ;

   assign rising  =  cnt_next.sclk && ~cnt.sclk ;
   assign falling = ~cnt_next.sclk &&  cnt.sclk ;

   always@(posedge clk) begin

      if ( falling )            // shift mosi out
        mosi <= sr[31] ;
```

```
      if ( rising )              // shift miso in
        sr <= {sr[30:0],miso} ;

      if ( done ) begin
         data <= sr ;            // copy to parallel out
         sr <= MISO ;            // channel select serial out
         mosi <= MISO[31] ;
         valid <= '1 ;           // data ready
      end

      if ( ready && valid )      // data was read
        valid <= '0 ;

   end

   always@(posedge clk)          // run continously
     ssn <= reset ;

endmodule
```

# 7 Bibliography

[1] "Reconfigurable Digital Systems Research Group," [Online]. Available: http://www.ent.mrt.ac.lk/rds/index.php/curriculum-support/dsd-projects/30-batch-07-dsd-projects/47-real-time-pitch-shifting-on-an-fpga. [Accessed 23 March 2018].

[2] D. Pyatkov and D. Hughes, "Real-time FPGA Pitch Shifter," 2007. [Online]. Available: https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2007/dah64_dp239/index.htm. [Accessed 3 March 2018].