Laboratory Experiment – Mastering Digital Design (Part IV)

(webpage: https://github.com/Mastering-Digital-Design/Lab-Module)

PART 4 – Real-time Audio Signal Processing

1.0 Putting everything together

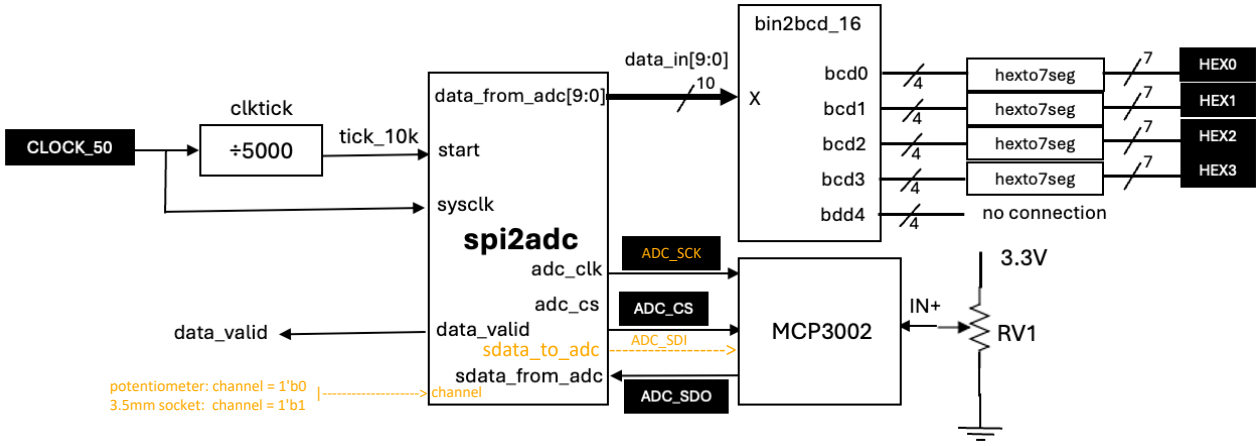
In this part of the experiment, you will learn to combine the ADC with the DAC on the Add-on card, and use the DE1 to perform some simple audio processing.

The goal of the final week's laboratory session is to implement a **speech echo effect synthesizer**. You need to bring your earphone to the lab to listen to the audio output.

2.0 Experiment 15: Testing the A-to-D converter

In this experiment, you will learn to use A-to-D converter MCP3002 on the add-on board to convert analogue voltages to digital signals. Download from the webpage the **spi2adc.v** module, which is already written for you to use.

Create the top-level design ex15_top.v which implements the circuit shown below:



Vary the potentiometer RV1 on the add-on card to verify that the ADC is working properly.

OPTIONAL CHALLENGE:

Modify **ex15_top.v** such that the display is the actual voltage in mV. For example, if the full-scale voltage is 3.312V, the display should show four digits with a value close to 3.312. Here are some useful hints:

1. All 7-segment displays have an eighth "segment" which is the decimal point (DP). For example, HEX3 is actually HEX3[7:0] where HEX3[6:0[are the 7 segments, and HEX3[7] is the decimal point DP. If HEX3[7] = 1'b0, DP will be lid (i.e. low active).
2. You may assume that a converted digital value of 1000 corresponds to a voltage of 3.3V.
3. The multiply operator "*" can be used in SystemVerilog for unsigned multiplication.

3.0 Experiment 16: An audio in-and-out (all pass) loop

Download from the Experiment webpage the file **ex16_proto.zip**, which contains the prototype folder for this experiment.

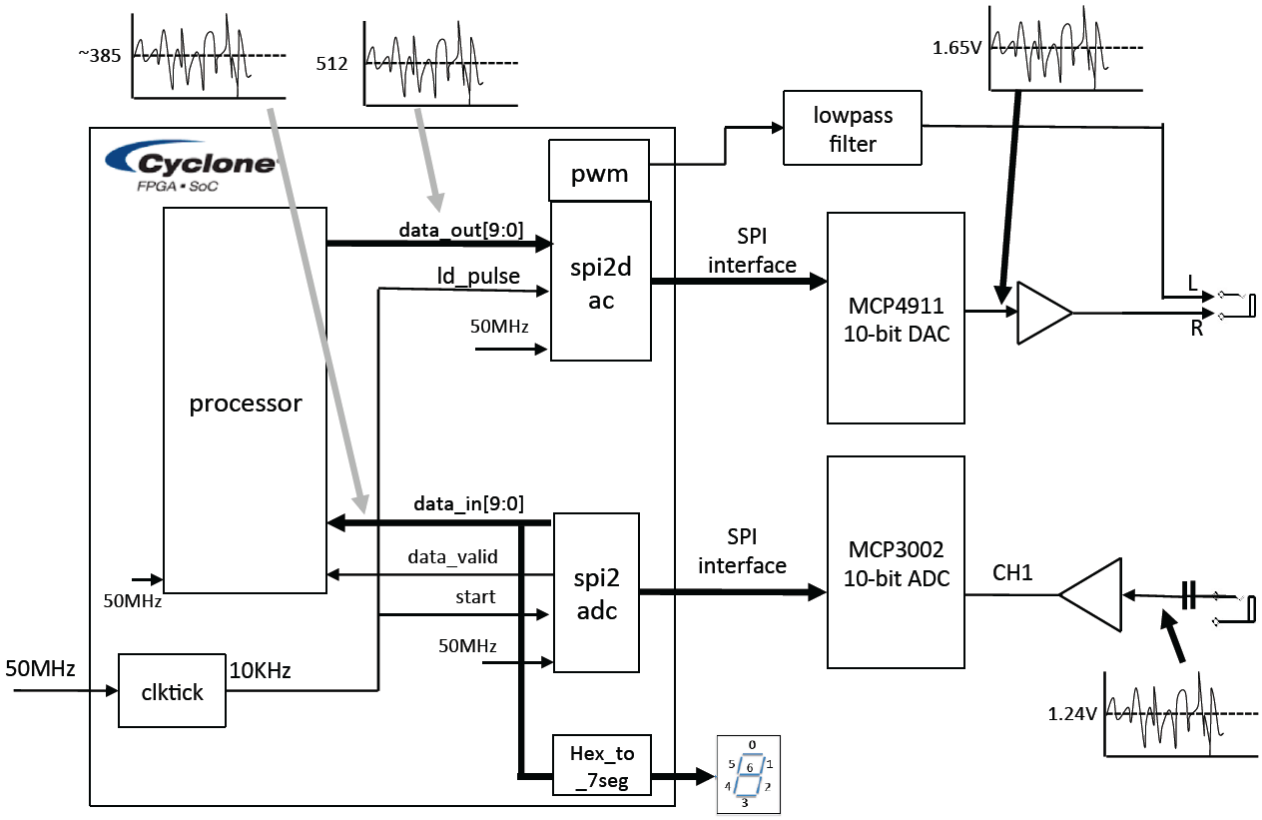
- Examine the contents within this folder. You should find the following Verilog files:

Module	Function
ex16_top.sv	Top-level design; interface to pins
spi2dac.sv	SPI interface circuit to DAC from Part 3
spi2adc.sv	SPI interface circuit to ADC
pwm.sv	Pulse-width modulation DAC from Part 3
clktick.sv	Clock divider to generate sampling clock ticks at 10kHz from Part 2
pulse_gen.sv	Generate a one-cycle pulse on rising edge of a trigger signal (new)
hexto7seg.sv	Hex to 7-segment decoder from Part 1
allpass.sv	"**processor**" module – this performs processing, which is passing input to output for now.

- Study **ex16_top.sv**. This specifies a system as shown in the following diagram (the part inside the Cyclone V). Make sure you understand how this works.
- Note how the **spi2adc.sv** module is used. Explicitly associating the signal names INSIDE the module to OUTSIDE allow connections to be defined independent of the order. This is a more verbose but is much safer way in making connects to modules.

```
spi2adc SPI_ADC (
    .sysclk (CLOCK_50),
    .channel (1'b1),
    .start (tick_10k),
    .data_from_adc (data_in),
    .data_valid (data_valid),
    .sdata_to_adc (ADC_SDI),
    .adc_cs (ADC_CS),
    .adc_sck (ADC_SCK),
    .sdata_from_adc (ADC_SDO));
```

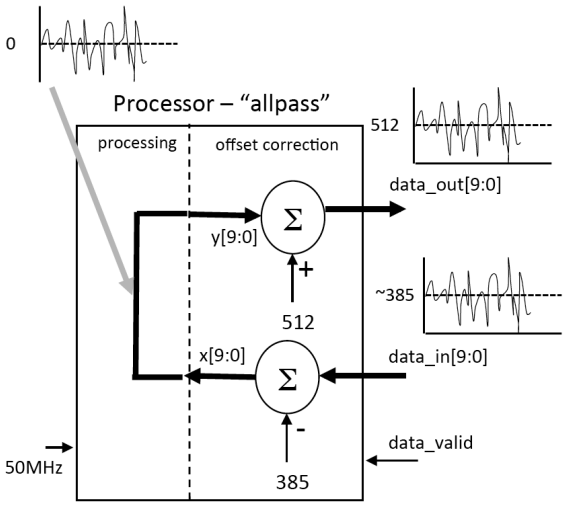
- The ADC has two analogue input channels: CH0 and CH1. They connected to the potentiometer and to the 3.5mm socket respectively. We always use CH1 for **ex16**.
- Now examine the module **allpass.v**. The name of this module is "**processor**" and is different from the name of the SV file. There is no need to use the same name except that normally it is more convenient to do so. However, in this case, we have deliberately used the filename "**allpass**" to describe its function, while using a more universal name for the module. You can choose "**allpass.sv**" as the source of the module "**processor**" now. Later, you can have a different Verilog file to define a different "**processor**". Which version of "**processor**" you use in your design is specified in **Project > Add/Remove File in Project**.



- Make sure that you understand fully what the Verilog file "**allpass.sv**" does. It actually does very little. It:

    1. Corrects the ADC converter data (which uses offset binary with 0V represented by a value of ~385), but subtracting the offset from data_out[9:0] to obtain a 2's complement value x[9:0].
    2. Connects X to Y, i.e. does nothing and hence "allpass".
    3. Converts the Y value from 2's complement to offset binary for the DAC. The offset now is at 512 as shown below.

- Build your design for testing on the DE1 Board. To do this, you should:
    1. Open each .v file, and use **Processing > Analyze Current File** on each of the Verilog file to ensure that there is not syntax error.
    2. Use **Project > Add/Remove** File in Project to include all the .v files you need. Here we select **allpass.sv** to supply the "**processor**" module. In the future, you could substitute **allpass.sv** with another file for a different processor.
    3. While **ex16_top.sv** is the current file in the editor window, use **Project > Set as Top-level Entity** to define top is the top-level module.
    4. Use **Project > Start > Analysis and Synthesize** … to check for errors and warnings without compiling everything.



5. Check that Device, Pin and **TimeQuest** clock period are all assigned correctly.
6. Compile the whole design and download the bit-stream file "**ex16_top.sof**" to DE1.
7. Test that it is working properly. You can use the PC to play some music or speech files (downloadable from Experiment webpage), and use an earphone to listen to the DAC output. When no signal is sent to the DE1 board, the display should show a hex value of 181 to 188.

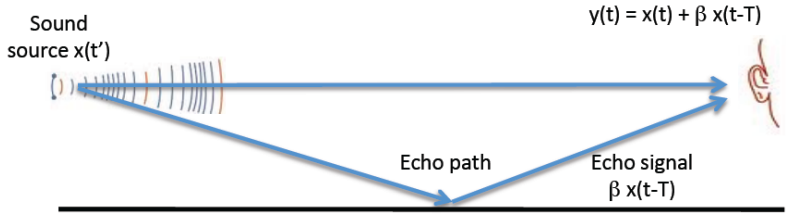
When you get to this part, the experiment framework is shown to be working. It takes audio samples at 10kHz from the ADC, passes it through a processor module and output the processed sample to the DAC.

Test yourself

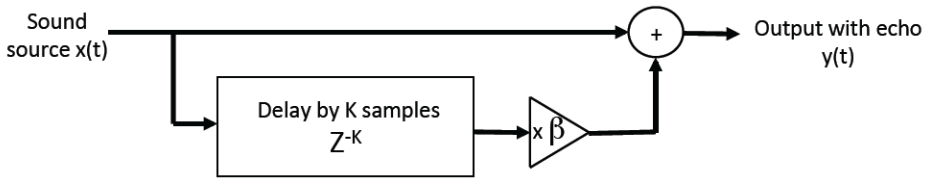
Now create a new SV file **mult4.sv** which is a processor module (i.e. module name is still "processor"), that amplifies the input by a factor of four. Test that this is working (i.e. the signal to the earphone should be louder or distorted).

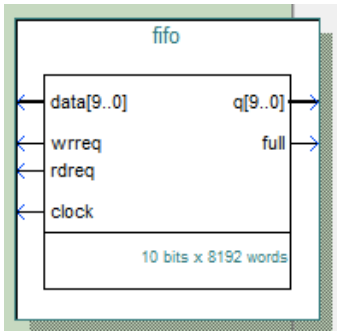
3.0 Experiment 17: Echo Synthesizer with fixed delay

In this part of the experiment, you will design, implement, and test a circuit that simulates the effect of simple echo. The diagram below shows two components of a sound source reaching its listener: the direct path signal x(t) and the echo signal β x(t-T) which is a weaker version of x(t) attenuated by a factor β, bounced off the floor. The echo signal is also delayed by T relative to the direct-path signal x(t).



Such simple echo can be implemented as signal flow graph as shown below. This involves three components: a delay block that delays x(t) by K sample periods; a gain block which multiplies the delayed signal by the factor β; and the adder.





The delay block can be implemented with a first-in-first-out (FIFO) buffer. A FIFO is found in all forms of digital systems. The rule is simple: receiving data are stored in sequence in such a way that they can be retrieved in the order that they arrive. When a new data item arrives and the FIFO is not full, it is written to the FIFO. As a stored data item is retrieved, it is removed from the FIFO. This allows the send and retrieve rates to be different in the short term. If the send rate is higher than retrieve rate, eventually the buffer will get full. If the buffer is full, it should not receive any more data (otherwise existing store data would be corrupted). A "full" status signal is asserted to tell the sender does not send any more

data. Similarly, if the buffer is empty, it cannot provide any data for retrieval. An "empty" status signal is used to indicate that the FIFO has no more data to provide.

Create a new project using the files from Experiment 16 as your prototype. With IP Catalog tool, generate a FIFO component of size 8192 x 10-bit as shown here. You only need to provide only the "full" status signal. This FIFO is used to store the most recent 8192 samples, hence providing a delay of 0.8192msec since the sampling frequency is 10KHz. Before the echo simulation circuit starts to provide the echo, the FIFO must first be filled (i.e. wait until the "full" signal is asserted). Thereafter, the writing of the ADC sample and DAC sample is synchronous, and the FIFO remains full. The read data is always the write data delayed by 8192 sample period.

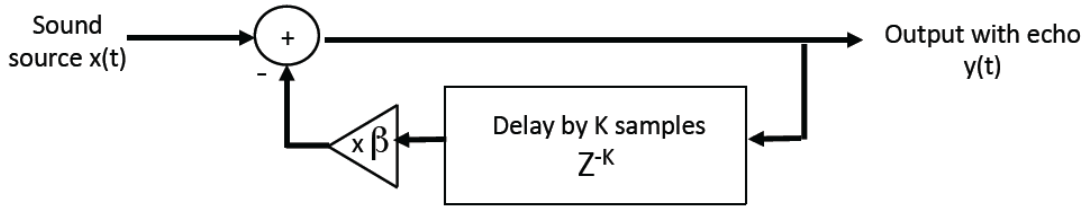
The attenuation factor β should be ½ or ¼, which can easily be implemented with a simply binary shift.

Deliverable

Implement the simple echo simulator and test that it works. For the purpose of test, download three different sound files: clapping.mp3, hello.mp3 and h2g2_5min.mp3, and play them on the PC or phone in a loop. Use your earphone to listen to the effect of the echo synthesizer.

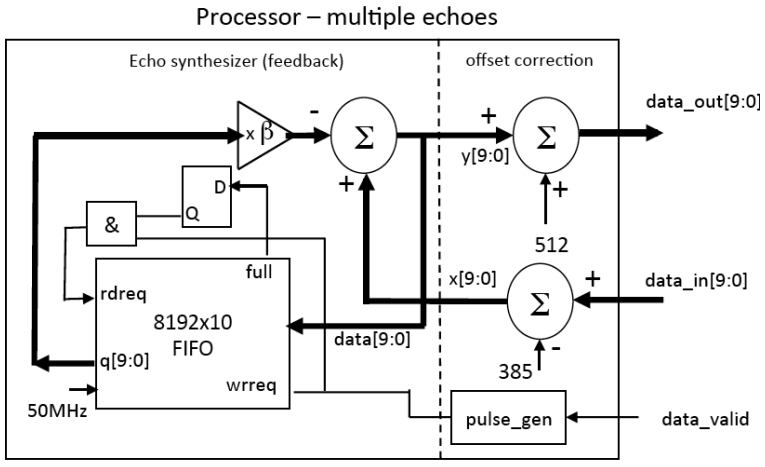
4.0 Experiment 18: Multiple echoes

The design in Experiment 17 produces a single echo. The signal flow graph only has feedforward paths. Multiple echoes can be produced with a slight modification of the signal flow graph to the one shown below.



The delay block now stores the output sample y(t) instead of the input sample x(t). The attenuated and delayed y(t) is SUBTRACTED from x(t) to produce the next output. (Why must this be a subtract and not an add?)

Provide a design to implement this architecture and test it.

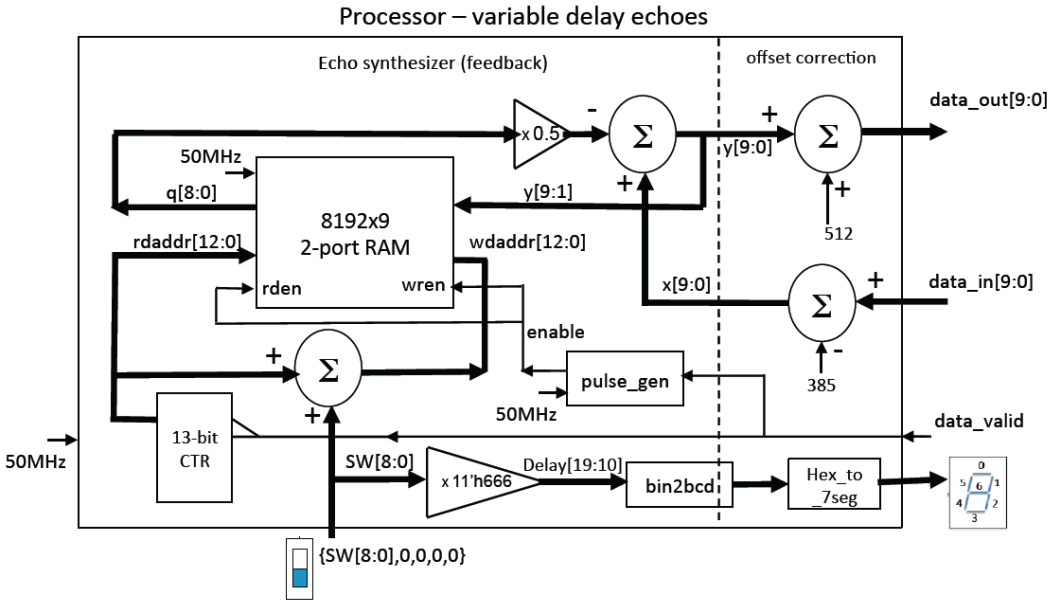


5.0 Experiment 19: Echo Synthesizer with Variable delay

In this experiment, you will design, implement and test a system with variable delay. A bit-stream (**echo.sof**) that implements a solution can be downloaded from the Experiment webpage. You also need to download the three MP3 test files. Connect the audio input to the speaker of the PC and play the audio files in a loop. Program DE1 with echo.sof and listen to the output with your earphone. Change the delay of the echo with SW[8:0]. The amount of delay in millisecond is displayed on the 7-segment displays as a decimal number.

The design of this experiment is shown in the block diagram below. It consists of a number of modules:

- RAM Delay Bock - In place of the FIFO to implement the delay block, it uses a 2-port RAM block (8192 x 9-bit) – one write port (to store the ADC samples) and one read port.
- Address Generator - A 13-bit counter is used to generate the read address to the RAM. (Why 13-bits?) The counter value is incremented on the negative edge of the data_valid signal at a frequency of 10KHz. In this way, the address generator computes the address used on the next read and write cycle. The write address is generated from the read address by adding the value taken from SW[8:0]. Since the address is 13-bits wide, the 9-bit delay value is zero-padded in its lower 4 bits. Therefore, the delay between the read and write samples is: SW[8:0] x 16 x 0.1 msec.
- The read and write enable signals are common, and it is generated from the data_valid signal with the pulse_gen module.
- The write data value y[9:1] is 9-bit instead of 10-bit wide. This is because the embedded memory in the Cyclone III FPGA is configurable as 9-bit in data width, but not 10-bit. Therefore, the output data value is truncated to 9-bit before storing in the delay block.
- The read data value is of course also 9-bit wide. Therefore the x0.5 can easily be implemented by sign-extending the 9-bit value to 10-bit: {q[8],q[8:0]}.
- The implementation of the feedback loop to generate the echo effect is identical to that from the previous experiment.
- To display the delay value in milliseconds, the value of SW[8:0] is first multiplied by 1638 (why) with a constant multiplier. This gives a 20-bit product, the most significant 10-bits of which is the delay in milliseconds. (Why?) This is then converted from binary to BCD and decoded for display on the 7-segment displays.

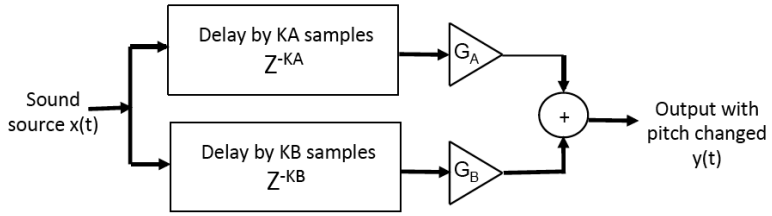


6.0 Experiment 20: Voice Corruptor (A Grand Challenge for the more able)

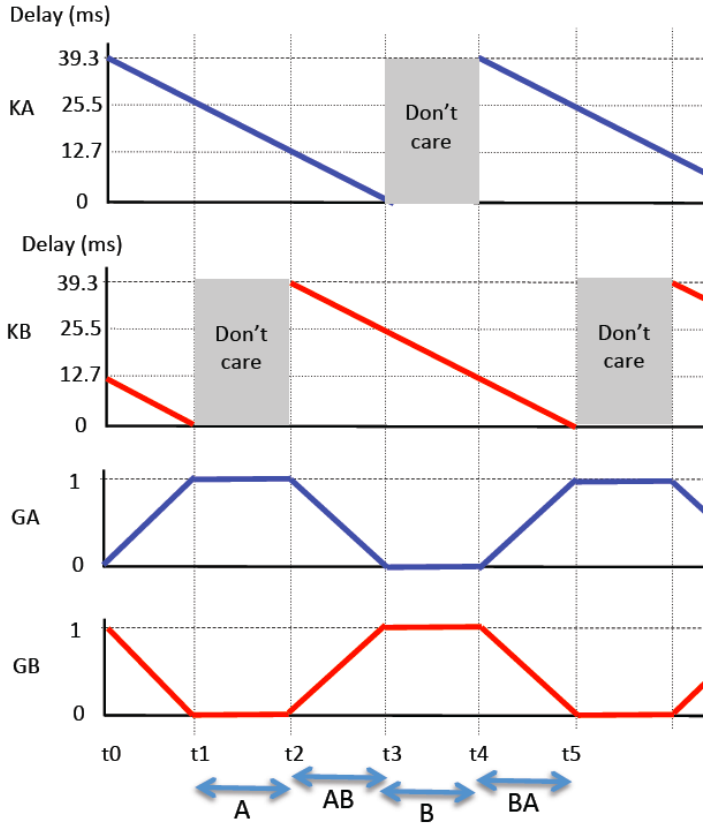
This part of the experiment is outside the scope of the experiment. It is designed to provide you with an open problem so that you can explore designing digital systems and implementing digital circuits using the DE1 and the add-on card at your own leisure. You need to check out a set of kits to take home from stores. For example, you might want to try this out over the Christmas break.

You are now equipped with all the tools and knowledge to design a reasonably complex audio processing system. The idea here is to design something that will take a human speech signal and then "corrupt" in a way that the identity of the speaker is masked while the speech remains intelligible.

One way to do this is to change the pitch of the speaker (e.g. make it sounds like Donald Duck). There are many ways to perform pitch change of speech. One method, which is linked to the previous experiments, is to employ a technique based on cross fading (i.e. combining) of two separately delayed version of the speech signal. The technique is depicted in the block diagram below.



The sound source is delayed through two separate blocks, providing KA and KB sample delays, which vary with time. The delayed signals are then attenuated by GA and GB, and combined with the adder. To minimize the artifacts and discontinuities in the output signal and to maintain a constant volume, the gain values GA and GB are designed to cross fade with each other – i.e. when one is ramping up (from 0 to 1), the other is ramping down. A plot of the four parameters, KA, KB, GA and GB, vs time is shown below.



There are four regions.

1. Region A (t1 to t2) - Only channel A is contributing to the output. The delay KA is gradually decreasing linearly from 25.5ms to 12.7ms (255 to 127 x 100μs). The gain GA is constant at 1.
2. Region AB (t2 to t3) – Both channels contribute to the output with A decreasing and B increasing their respective contributions. The two channels are cross faded before GA drops from 1 to 0 while GB increases in the other direction.
3. Region B (t3 to t4) – This is like Region A, but the behavior applies to channel B instead of channel A.
4. Regions BA (t4 to t5) – Like Region AB, but the two channels are reversed.

The pattern repeats itself indefinitely. Note that the "don't care" portion of KA and KB is because during this period, the gain GA or GB is zero.

Hints:

- Initially, try the delay ramping gradient of 0.5, i.e. the delay is dropped by k over time 2*k.
- You can use a 9-bit down counter to define both the delay KA and the four regions.
- You can derive all other values: KB, GA and GB, from the counter values.
- You can design a four-state synchronous state machine to control the corruptor circuit.
- Instead of delay varying ramping high to 0, you can reverse the direction of the ramping. Alternative you can design the delay to vary up and then down.
- In addition to pitch changes, you may explore other audio effects.