Imperial College London

Lecture 13

Echo Synthesizer & Challenges Explained

Peter Cheung
Department of Electrical & Electronic Engineering
Imperial College London

URL: www.ee.imperial.ac.uk/pcheung/teaching/E2_CAS/
E-mail: p.cheung@imperial.ac.uk

Lecture Objectives

- To revisit some of the issues that came up during the laboratory experiments
- To provide some guidelines on how to perform diagnosis when things don't work
- To provide explanations on Lab 6
- To explain how the ADC works
- To explain some of the major modules used in the experiment
- To explain the idea of offset binary vs 2's complement
- To explain the ALLPASS module and its use
- To explain how echo may be synthesized

This lecture is designed to complement and explain Lab 6 experiment.

How to minimize problems?

1. Top level module name and file name (i.e. *.v) must match. This rule only applies to top-level module connected to physical pins.
2. Always check each .v file for syntax error with **Processing > Start > Analyze and Elaborate**
3. Make sure that you have included ONLY the files used in your design with **Project > Add/Remove files in Project**
4. Make sure that you have specify the correct top-level entity by first open the top-level module file, and click **Project > Set as Top-level Entity**
5. Always check for correctness of your design with **Processing > Start > Start Analysis and Synthesize**, and fix any errors
6. Check that you have assigned top-level ports to physical pins (done by editing the **<project_name>.qsf** file).
7. Check that you have specified your device to be 10M50DAF484C7G
8. Edit .qsf file to add pin assignment immediately after creating the project

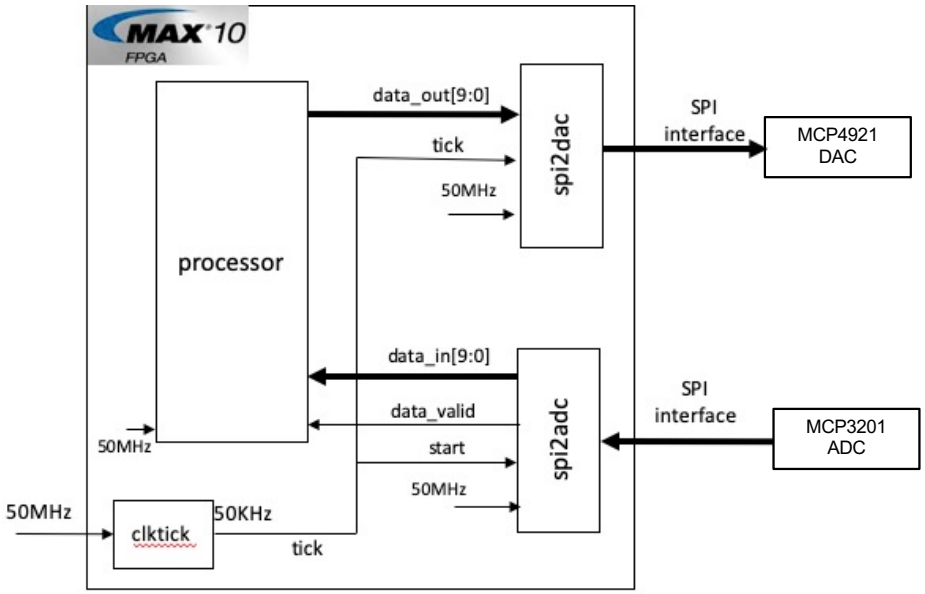
This slide is self explanatory. These are some steps you should take in order to minimize problems that you may encounter.

Common mistakes

1. Bad organisation of design folder – missing versions, files, folder etc.
2. Wrong case for signal names (all names are case sensitive)
3. Wrong number or wrong order of signals when instantiating a module
4. Different number of bits used in signals at top-level and lower modules
5. Missing pin assignments or use the wrong pin names
6. You may use multiple always_ff @ (posedge/negedge clk) blocks in the SAME module, but must not do assignment to the same signal more than once

Here is a list of common mistakes students had in the lab.

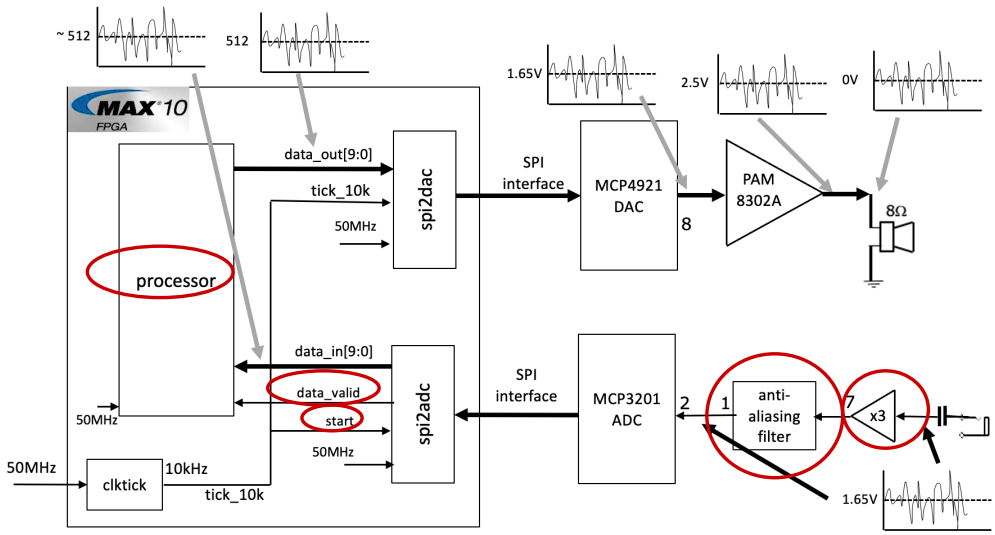
Lab 6 Task 3 Explained



This shows a "**processor**" module, which in Task 3 does an ALL PASS function. That is, it takes a sample from the ADC and pass this to the output and to the DAC. Therefore everything is simply passed from input to output. In Task 4, we create other "processor" module that perform other processing functions.

Combining analogue and digital systems

- X3 amplifier & anti-aliasing LP filter
- ADC produces a data_valid pulse at end of conversion



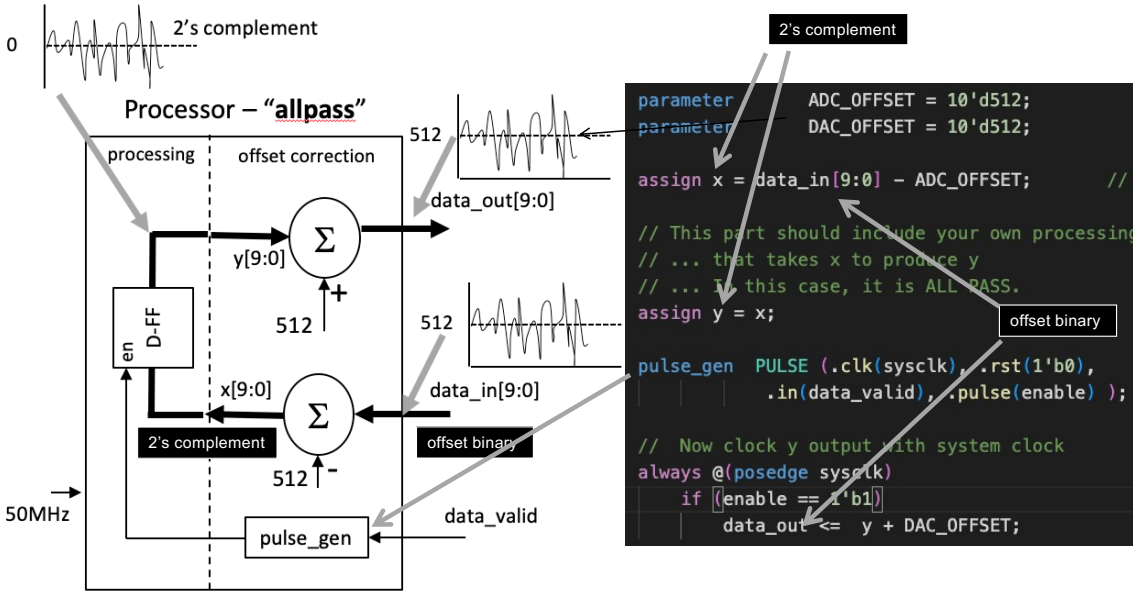
This is the block diagram of the basic framework used for Lab 6 Task 3 and 4.

The analogue part of the system includes a x3 amplifier which provides an audio signal for the full 3.3V range.

The Sallen-Key lowpass filter acts as an anti-aliasing filter (from Signals and Systems course) to avoid corrupting signal in the lower frequency band. This LP filter has a corner frequency of around 1kHz. Given that our sampling frequency is 50kHz, we only need to suppress signals beyond 25kHz. We could have used a LP filter with much higher corner frequency, e.g. 10kHz. This will work well for our system.

The two main modules on the FPGA are spi2dac.v and spi2adc.v. They provide SPI interface to the DAC and ADC respectively. The control circuit is simple – a clock tick circuit generating a 50 KHz sampling clock.

Lab 6 Task 3 – allpass.v (offset correction)

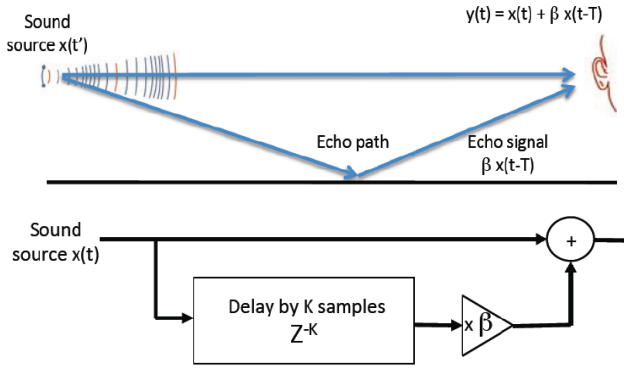


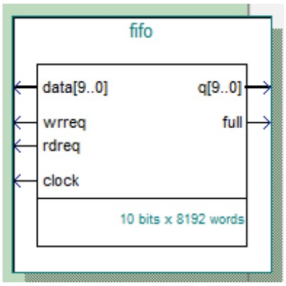
The ALL PASS module is slightly more complex than it may appear. Data_in[9:0] is used to represent the analogue signal input (which is bipolar) as offset binary. There is an offset of around 512 if the input is connect to zero (no signal). The output data_out[9:0] also has an offset. To get Vout = 0V, you need to send the binary number 512.

If you are to process the signal using normal arithmetic operators such as +, - and *, you need to use 2's complement number system. Therefor the ADC data is first offset correct by subtracting the offset 512 from the converted data to yield x[9:0]. The actual processing step is simply to store this data in a register in 2's complement form. Then the output y[9:0] is again converted back to offset binary for the DAC to output. This is done by adding 512 to y[9:0].

If allpass.v and lab6task3.v are both correctly specified, you can send in the ADC a recorded speech signal via the 3.5mm cable, and hear the same speech on the speaker.

Lab 6 Task 4 – single echo synthesizer





- Single echo of source signal
- Signal flow-graph is simple: a K samples delay block, a gain block and an adder
- Use First-in-First-out memory to store sample: need a status signal "full" to indicate FIFO full
- Sampling frequency = 50KHz, theref a 8192 word FIFO provides 0.1638 second delay

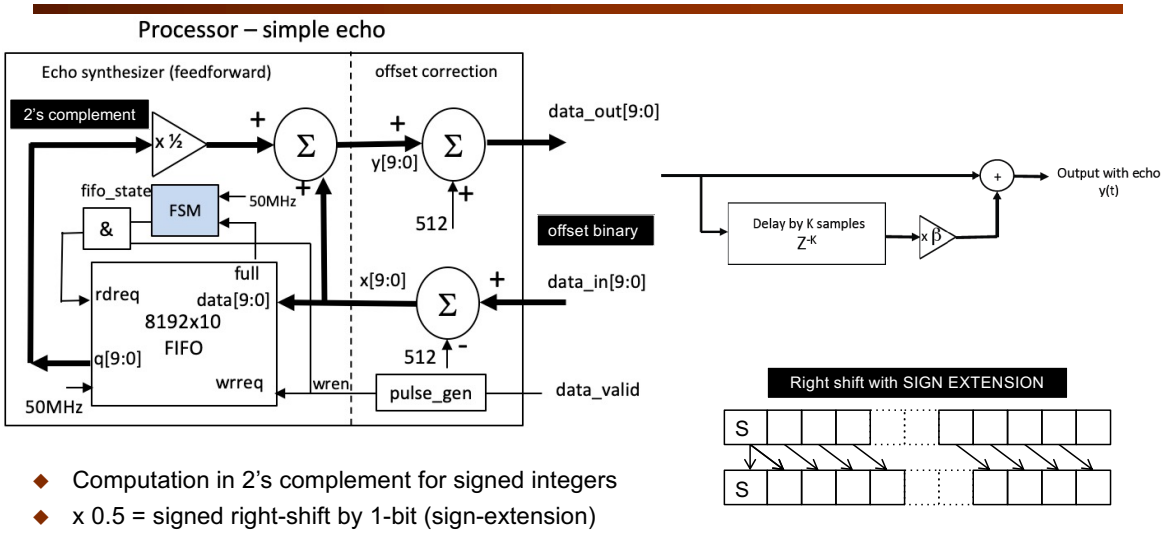
The final task is to create an echo synthesizer. The basic idea is simple: an echo is recreated when the listener receive the source signal via a direct path AND a delayed echo path as shown.

In order for this to work, we need a delay component in the FPGA system. The easiest way to achieve this is to use a first-in-first-out (FIFO). I will explain exactly what a FIFO is in a later lecture. For now it is sufficient for you to know that a FIFO block has data[9:0] as input, and q[9:0] as output. The first sample that goes in is the sample the first sample that comes out. There is a write request signal wrreq which is asserted when you want to write a word into the FIFO. Similar a rdreq signal is asserted when you want to read a word out from the FIFO. There is a synchronising clock signal.

Finally if the FIFO is full (in this case storing 8192 samples already), then the full signal goes high.

This FIFO will provide 0.1638 second delay if the sampling clock is 50KHz.

Lab 6 Task 4a – single echo synthesizer

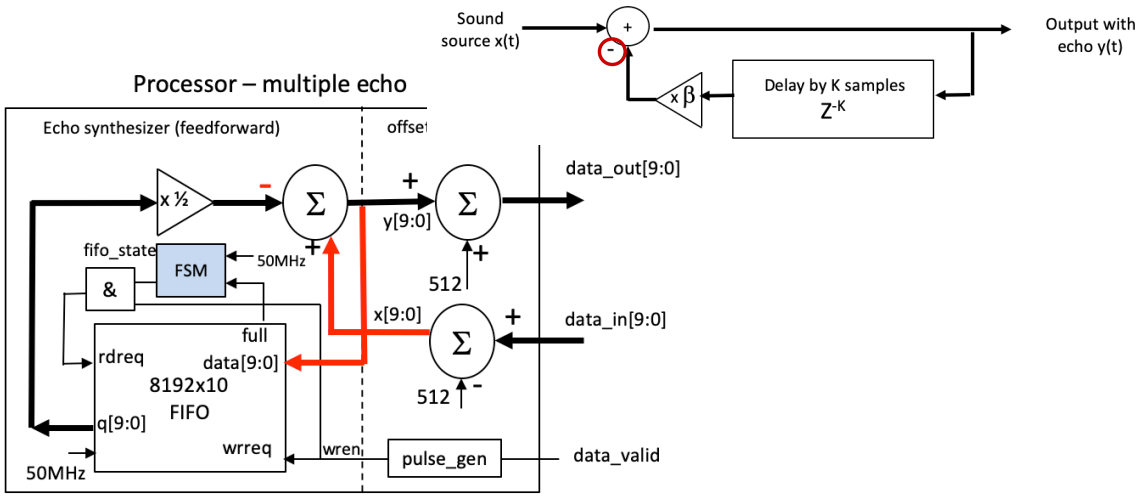


- Computation in 2's complement for signed integers
- x 0.5 = signed right-shift by 1-bit (sign-extension)
- Verilog: y[9:0] = x[9:0] + {q[9], q[9:1]};
- Additional signal to processor module: data_valid = a high pulse whenever there is a new data_in
- Need to fill to First-in-First-out memory before starting to read data off it – hence FSM

Here is the block diagram of the processor module for a single echo synthesizer. The FIFO control circuit is quite simple, the FSM and the AND gate ensure that at the start, the FIFO is not read until it is completely filled. The AND gate blocks the wren pulse from the pulse generator. Therefore for the first 8192 conversions, the FIFO is only written to, and nothing is taken off it.

When the FIFO is full, the FSM output goes high, and from now on, every data written into the FIFO, another data value 8192 samples earlier is taken off the FIFO as the echo signal. This is then scaled by a constant 0.5 (which is an arithmetic right shift with sign extension).

Lab 6 Task 4b – multiple echoes synthesizer



- Instead of feedforward only, this uses a feedback loop
- To avoid instability, you must SUBTRACT delayed echo signal instead of add
- FIFO now stores y[9:0] output, and NOT input

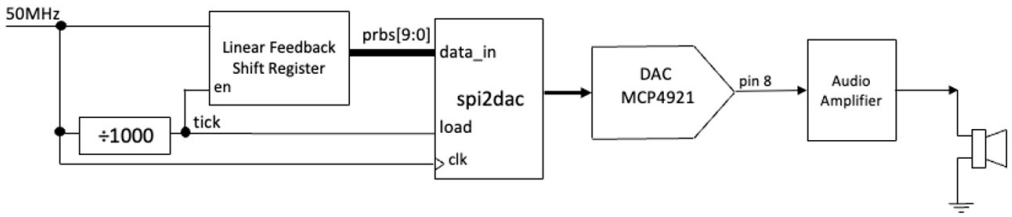
A slight modification create a mult-echo synthesizer. Here we put the delay element in a feedback path. Note that you MUST perform a subtract instead of an add, otherwise the system has positive feedback and will become unstable.

Challenges

- Lab 1 – 6: Teaching you by holding your hands, with a few “test yourself” tasks
- Challenges: Open-ended problems to challenge you. Give you a chance to “showoff” what you have learned
- No time to do more than one or two. Welcome to do them all if you want.
- Final Lab Oral – asked evidence of successful challenges (videos)
- Not completing any challenges will limit your final lab oral grade to at best a B (fair to others)
- All challenges are ranked in levels of difficulties (1 to 4)

Challenges are created to allow you to demonstrate you have attained the learning outcomes for this module. Therefore you are advised to complete all 6 Labs and at least one or more challenges.

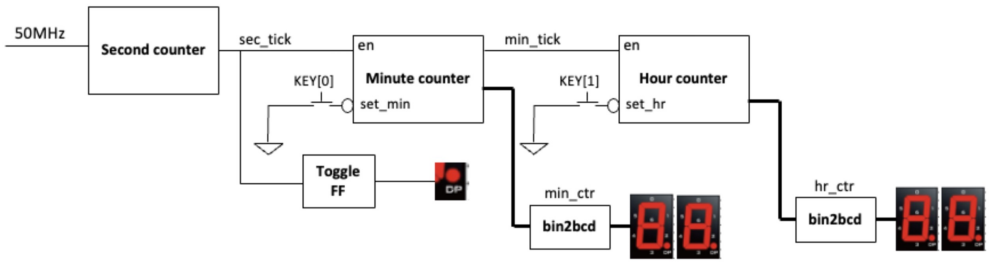
Challenge 1 – Noise Generator (level 1)



- Hint: If you use the 9-bit PRBS from Lab 5, remember that the DAC is 10-bits. Beware of the mismatch.
- Better to use a 10-bit PRBS implementing a primitive polynomial for 10-bits from the table provided in the lecture notes.

This should be very simple to do.

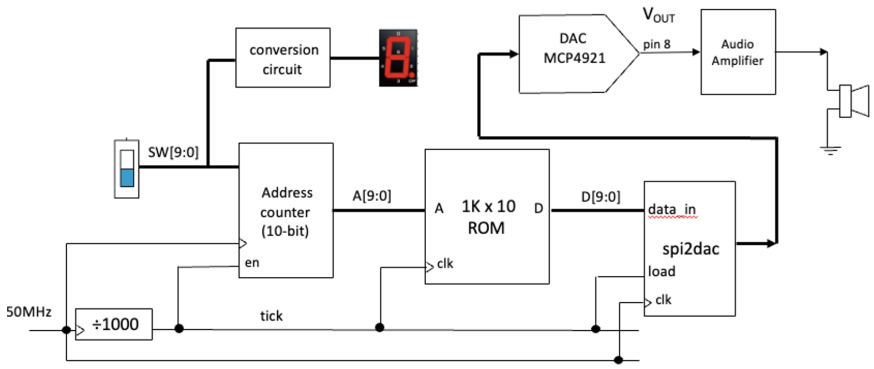
Challenge 2 – A real-time clock (level 2)



- This is a simple challenge because you only use counters and a few extra modules.
- The tricky part is to have a way of setting minutes and hours.

This challenge is easy to achieve but can be time consuming to finding an effective way of setting the time.

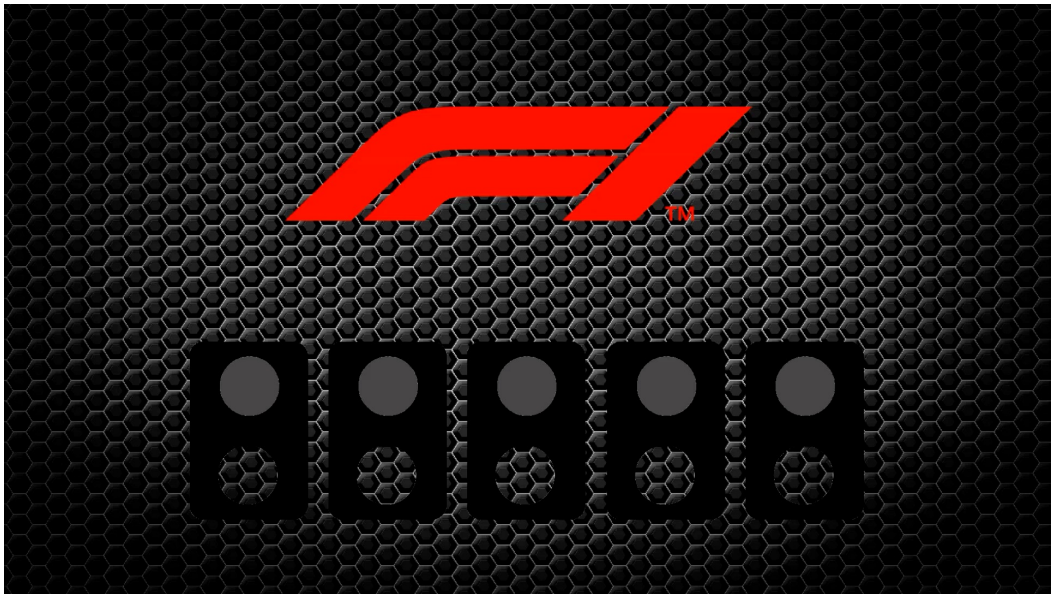
Challenge 3 – Variable sinewave generator (level 2 or 3)



- This challenge teaches you to design with a ROM as well. The level 3 attainment will involve the translation of SW[9:0] to the actual frequency of the sinewave produced.

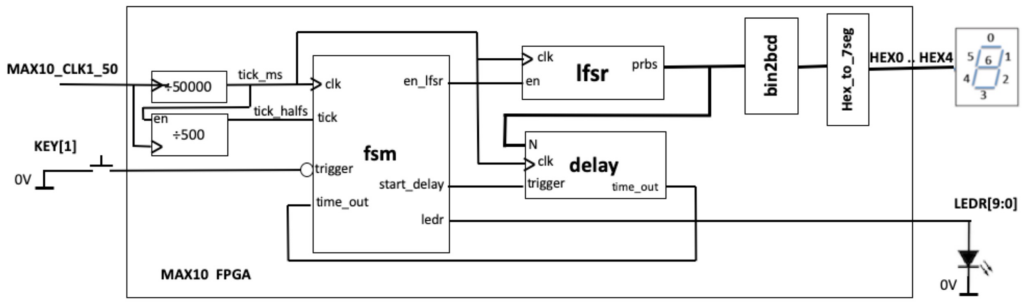
This is a nice challenge that produces very pure sinewave that you can hear. Try setting frequency to 440Hz – the tuning fork frequency of orchestras.

Challenge 4 – Formula 1 starting light (level 4)



This is a nice challenge that produces very pure sinewave that you can hear. Try setting frequency to 440Hz – the tuning fork frequency of orchestras.

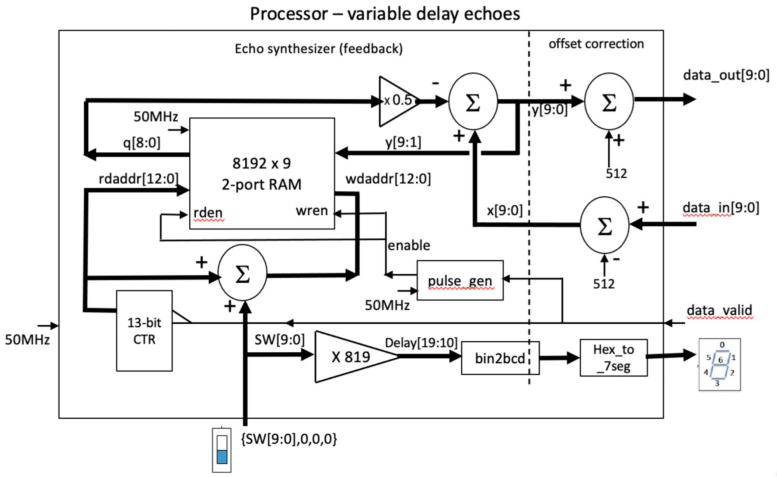
Challenge 4 – Formula 1 starting light (level 4)



- This challenge is level 4 because it really tests every aspects of the digital part of the module. It requires LFSR, FSM, counters, delay module, shift registers etc.
- Stretch goal allows you test your reaction time (in ms)

This is not an easy challenge, but you will learn a lot and the end result will be very satisfying.

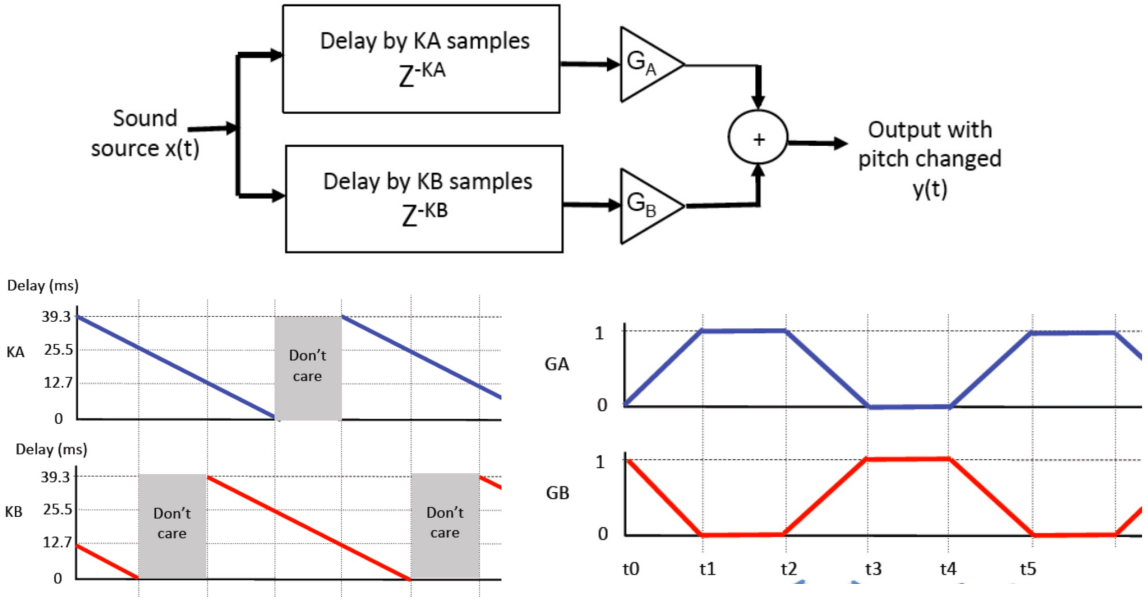
Challenge 5 – Variable Delay Echo Synthesizer (level 4)



- This challenge is based on an extension to Lab 6. Instead of FIFO, you use a RAM to implement a variable delay block. Note also that you need to compute the delay in ms and display this!

This is a great challenge which take what you did in Lab 6 further. The end result is most satisfying. Play the long audio book and change the delay value. You will hear the effect of echo very clearly.

Beyond the challenges – A Voice Changer



PYKC 26 Dec 2024 EE2 Circuits & Systems Lecture 13 Slide 18

For those who are bored with Christmas vacation, here is a challenge beyond all other challenges. You can construct a voice changer (one that changes the pitch of a voice without change in speed) by implementing a system shown in the slide. There are two delay blocks with variable delay changing in time as shown above. You then mix the two signal paths with a variable gain. Magically, the voice pitch will be changed.

I will demonstrate its effect during the lecture. It is quite impressive!