Department of Electrical & Electronic Engineering
Imperial College London

ELEC50001 Circuits & Systems

The Challenges

Introduction

Labs 1 to 6 were designed to teaching you something. In these Labs, you were told what to do, and often how to do them. Now you should be able to approach solving practical problems in circuits and systems by yourself.

This document is NOT a lab instruction *per se*, but it contains various suggestions on possible challenges that you might take on yourself. You are invited to attempt one of these challenges if you have time and want to go further than the basics. However, DO NOT do more than one or two – you will not have time. Pick the ones that you think you can complete within the time without stressing yourself. Don't forget, you have other modules to complete too!

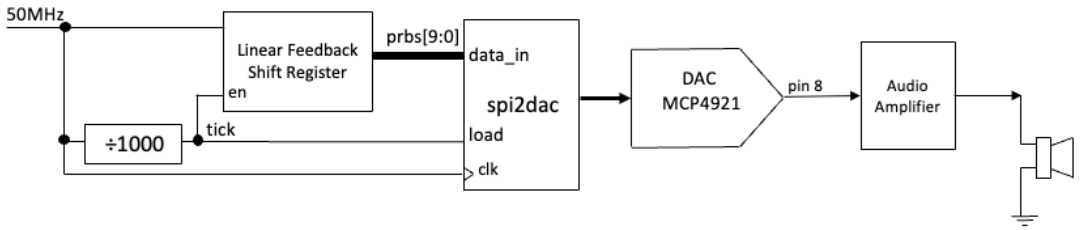
The Challenges are arranged in order of increasing levels of difficulties (Level 1 to 4). During the Lab Oral in the last week of the term, you will be asked by your assessor questions on Lab 4 – 6, and you will also have a chance to "show off" your Challenges if you have done any by showing your video recording of your Challenge result(s). Having completed one or two Challenges will of course gain you some credit during Oral, but they are not necessary to get a good grade.

Finally, since you will have the Lab-in-a-Box with you over Christmas, you are encouraged to explore some of the other challenges at your own leisure during the Christmas break between feasts and binge TV watching. You will not be doing this for credit, but for your own sheer enjoyment.

Have fun!

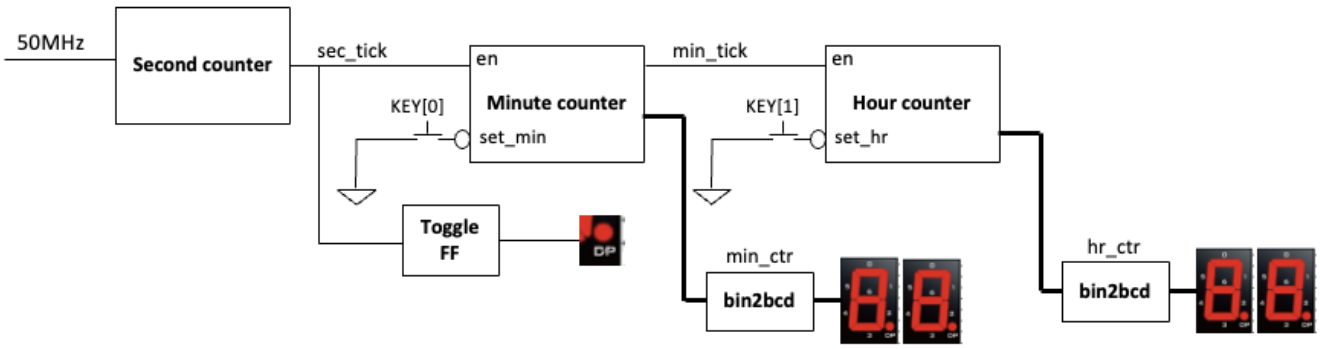
Challenge 1: Random Noise Generator (Level 1 – minimum level)

In Lab 4, you learned about the pseudo-random binary sequence circuit to generate pseudo random numbers. If you feed these sequence values to a DAC converter, you should now have to ability to produce random noise. Since the DAC accepts 10-bit input, I suggest that you should use a PRBS which is 10 bits or wider. Listen to this on the speaker. The scope can show the spectrum of a signal. Investigate the spectrum of the generated noise and see how "white" it is. (Look up the definition of white noise.)



Challenge 2: A Real-Time Clock (Level 2)

This challenge is not difficult but can be time consuming. The task is to combine everything you learned about counters and displays to create a 12 hours or 24 hours clock with the DE10 board. You need to create counters that count seconds, minutes, and hours, which should not be too difficult. I suggest that you only display hours and minutes, and seconds are shown as blinking decimal point of the relevant 7-segment display. The challenge here is also to include the ability to set the current time. (Note that all 7-segment LED has 8 input pins, e.g. HEX3 is from [7:0], where HEX3[7] is the decimal point DP.)



Challenge 3: Variable frequency Sine wave generator (Level 2 or 3)

In Lab 5, you produced variable frequency sawtooth wave generator. This was achieved by sending the output of a counter to the DAC, which produces an audible signal through the audio amplifier.

Instead of using the counter output values directly, you could use these as addresses to a ROM (Read-only Memory), which has a cycle of sine coefficient already stored. This will allow you to generate an extremely low distortion sine wave signal.

The following are some steps to help you to generate the ROM and initialize it content to store the sine wave.

Step 1: Generating the contents for a ROM

This part of the experiment leads you through the design of a 1K x 10 bit ROM, which stores a table of sine values suitable to drive our DAC. The relationship between the content of the ROM D[9:0] and its address A[9:0] is:

$$D[9:0] = \text{int}(511*\sin(A[9:0]*2*\pi/1024)+512) \qquad \text{for } 1023 \geq A[9:0] \geq 0$$

Since the DAC accepts an input range of 0 to 1023, we must add an offset of 512 in this equation. (This number representation is known as **off-set binary** code.)

Before generating the ROM in Quartus using the "**Memory Compiler**" tool, we need to first create a text file specifying the contents of the ROM. This can be done in different ways. Included on the course webpage are: 1) a Python script to do this; 2) a Matlab script to do the same thing; 3) a memory initialization file **rom_data.mif** created by either method. Download these files and examine them.

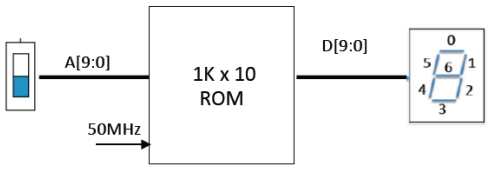
Step 2: Use IP Catalog to generate the ROM module

```
module ROM (
    address,
    clock,
    q);

    input   [9:0]  address;
    input     clock;
    output  [9:0]  q;
```

Click Tools > IP Catalog to bring up a tool which helps to create a 1-Port ROM. A catalog window will pop up. Select from the window **>Library >Basic Functions > Onchip Memory > ROM 1-Port.** Complete the on-screen form to create **ROM.v**. Note that IP Catalog only produces modules in Verilog, not SystemVerilog. You can mix them in a design.

To verify the ROM, create a design that uses the switches **SW[9:0]** to specify the address to the ROM, and display the contents stored at the specified location on the four 7-segment display. Once this is done and loaded onto the DE1, verify that the contents stored in the ROM matches those specified in the **rom_data.mif** file.



Step 3: Variable sine wave generator

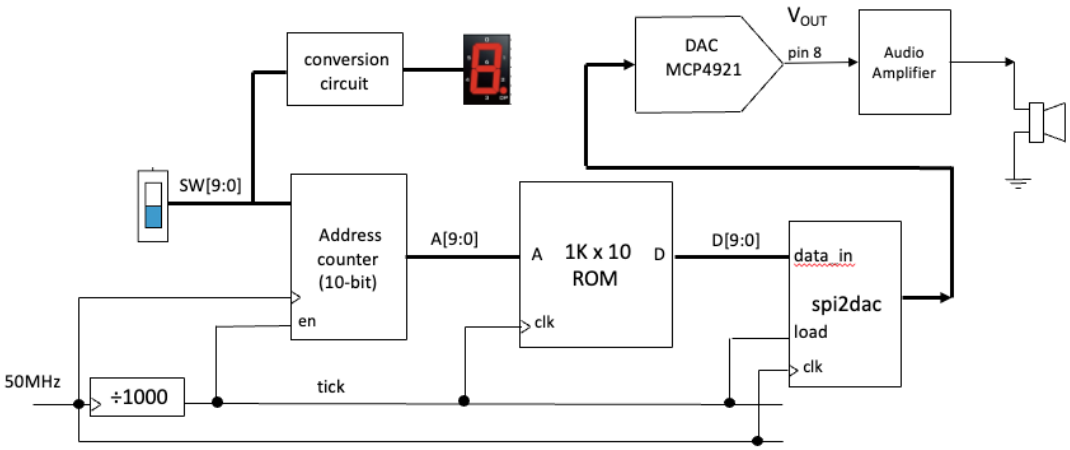
Modify the variable sawtooth wave generator in Lab 5 by inserting the ROM into your design at the appropriate place. You then can use the slide switch SW[9:0] to determine the frequency of the generated signal.

Use the scope to see how "pure" the generated sine wave is. (Note that the LM386 is not very linear. It is best to measure the spectrum of the sine wave at the output of the DAC directly, and not at the output of LM386.)

Step 4: Stretched Goal (Level 3)

Display on the 7-segment displays the frequency of the sinewave generated. You will have to work out how to convert SW[9:0] to frequency in hardware. (Harder than it first appears!)

The diagram below shows the entire system.



Challenge 4: Formula 1 starting light controller (Level 4)

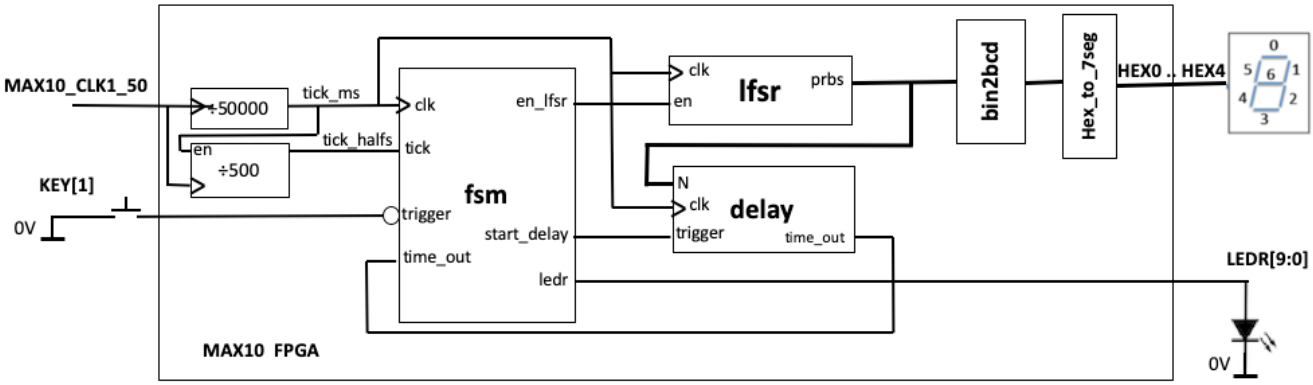
The goal in this Challenge is to design a Formula 1 style of race starting lights. The link below is to a YouTube video on a similar light with sound. (Sound is not required for this Challenge.

https://www.youtube.com/watch?v=FtT9IV1Q7Dg

The specification of your circuit is:

1. The circuit is triggered (or started) by pressing KEY[1] (don't forget KEY[1] is low active);
2. The 10 LEDs (above the slide switches) will then start lighting up from left to right at 0.5 second interval, until all LEDs are ON;
3. The circuit then waits for a random period of time between 0.25 and 16 seconds before all LEDs turn OFF;
4. You should also display the random delay period in milliseconds on five 7-segment displays.

To assist you in designing this circuit without spending too much time, the following overall block diagram of the circuit is provided.



The two clock divider circuits provide clock ticks once every 1ms and 0.5sec respectively. Each clock tick should be a positive pulse lasting one period of **CLOCK_50** (i.e. 20ns). The system then uses the **tick_ms** signal as the clock of the remaining circuit.

The **LFSR** module produces a pseudo-random binary sequence (**PRBS**), which is used to determine the random delay required. The **enable** signal to the **LFSR** allows this to cycle through many clock cycles before it is stopped at a random value.

The **delay** module is triggered after all 10 LEDs are lid, and then provides a delay of **N** clock cycles (at 1ms period) before asserting the **time_out** signal (for 1ms).

The delay value N is fed to the binary to BCD converter, which then drives the 7-segment displays.

There are several design decisions to be made:

1. How many bits LFSR is required?
2. How many bits should you use in the delay module?

The FSM module is the key module to the entire system. You must decide what are the states that are required, draw the state diagram, and then map that to SystemVerilog.

Challenge 4 Stretched Goal

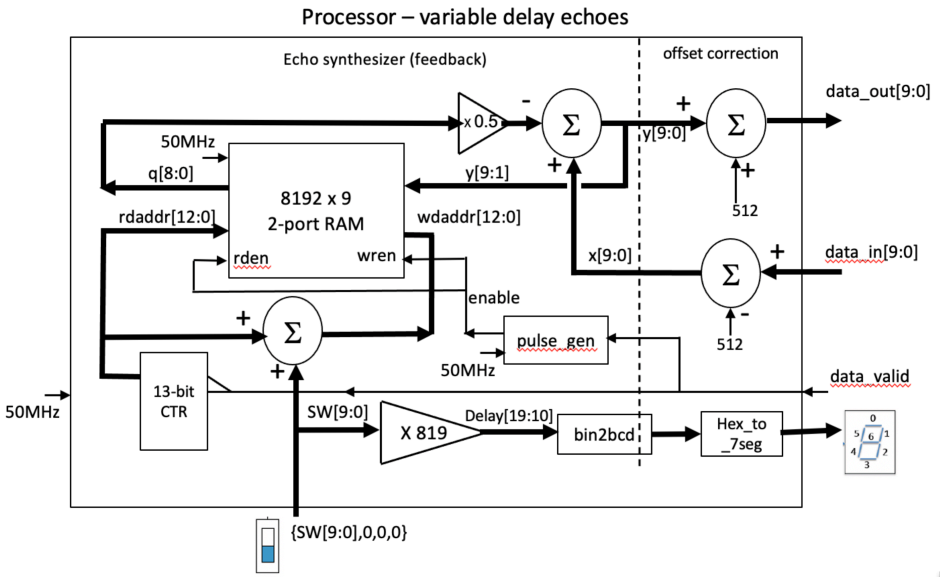
Extend your circuit by adding a reaction counter. This should count the time between all the LEDs turning OFF and you pressing **KEY[0]**. The reaction time should be displayed on the 7-segment displays in milliseconds in place of the random delay.

Challenge 5: Variable Delay Echo Synthesizer (Level 4)

This challenge is for you to modify the multiple echo synthesizer in Lab 6 so that you can vary the delay of the echo using SW[9:0]. The amount of echo 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. Since It consists of several 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. Due to the configuration of the embedded RAM block inside the MAX10 FPGA architecture, you can have 9-bit or 16-bit data width. Use 9 bits.
- In order that the 8192 word RAM is sufficient to store enough samples for a 0.8sec delay, we will use the sampling rate of 10kHz, so that each stored sample corresponds to a sampling period of 100us. Since we have an anti-aliasing filter with a corner frequency of 1kHz, 10kHz sampling rate is perfectly acceptable.
- Address Generator - An address 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 circuit computes the addresses used for the next read and write cycle ahead of the **rden** and **wren** pulses. The write address is generated from the read address by adding the value taken from **SW[9:0**]. Since the address is 13-bits wide, the 10-bit delay value is zero-padded in its lower 3 bits. Therefore, the delay between the read and write samples is: **SW[9:0]** x 8 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 to produce a read and write pulse at a rate of 10kHz.
- The write data value y[9:1] is 9-bit instead of 10-bit wide. This is because the embedded memory in the MAX10 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 RAM.
- 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 Lab 6.
- To display the delay value in milliseconds, the value of SW[8:0] is first multiplied by 819 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.

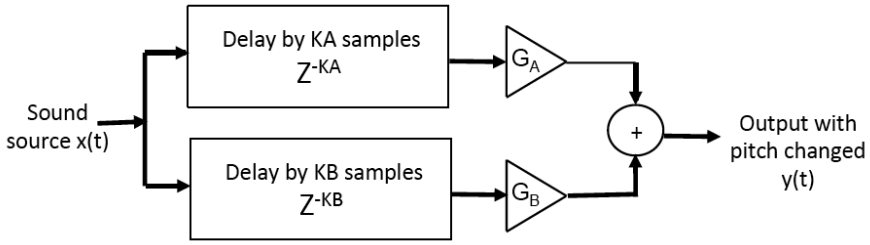


Challenge 6: Voice Corruptor (Bonus Christmas Challenge)

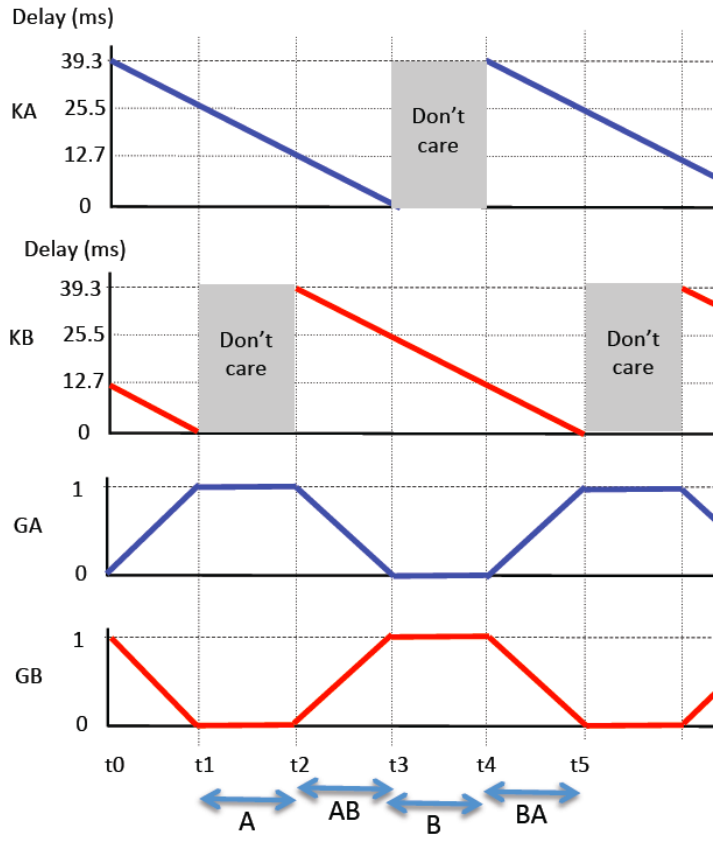
This Challenge is outside the scope of this module and the difficulty level is not specified. This challenge is designed to provide you with a difficult problem so that you can go beyond the expected learning outcomes of this module. For example, you might want to try this out over the Christmas break. If you intend to do this, you may keep the Lab-in-a-Box over Christmas. Do remember to bring it back in the New Year. Your final Lab Oral marksheet will not be released until I have confirmation that you have returned the Box.

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" it 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 echo synthesizer, 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. Unlike the echo synthesizer in Lab 6, I recommend that you reduce the sampling frequency from 50kHz to 10kHz to obtain the best effect.



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. In order 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 similar to Region A, but the behavior applies to channel B instead of channel A.
4. Regions BA (t4 to t5) – Similar to 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.