Objective: The purpose of Part 0 of the lab is to get you familiarized with the I/O capabilities of the MD407 card and learn the background for Part 1 and Part 2 of the laboratory assignment. First, you will learn how to compile the template code using the cross compiler for the Cortex-M4 microcontroller and upload the executable on the target hardware (Step 0). Second, you will learn how to take input from the keyboard of your workstation to the application hosted on the target microcontroller via the USB serial port of the MD407 card and how to generate output to the console program on your workstation (Step 2). Third, you will compute the period of a task that generates the waveform for a 1 kHz tone in Part 1 of the laboratory assignment (Step 3). Finally, you will learn how to compute the periods of the task that generates the waveform for the tones of the Brother John melody, and create the data structures that you need for Part 2 of the laboratory assignment (Steps 4–6).

Approval: When you see the text "*Assistant's approval: .....*" below a problem description you should show your solutions to the laboratory assistant. If the solutions are found to be satisfactory the laboratory assistant will mark the corresponding examination objective as 'Passed' in Canvas. You can then continue with the next problem.

Step 0: Hello, hello...

A paper copy of the 'Loan Agreement' form will be available in the lab room at your loan equipment pick-up session. Sign the document, to certify that you agree on the stated terms for using and returning the equipment, and then hand over the signed document to the course examiner.

Download the `CodeLite-TinyTimber quickstart tutorial` from Canvas[1]. Follow the guidelines given in the tutorial to compile the template code and upload it on the target hardware. If you run the template code successfully, you will see an output similar to the following on the console:

```
TinyTimber v2.06 (2018-02-05)

Hello, hello...
```

Step 1: Rules of Conduct

A paper copy of the 'Rules of Conduct' document will be available in the lab room at your first laboratory session. Sign the document, to certify that you have read and understood its contents, and then hand over the signed document to the course examiner.

[1]The document is available on the 'Resources' page in Canvas, under the 'Software' tab.

Step 2: I/O Using the serial port

During this step, you will practice input and output using the serial port to read data from the workstation keyboard and generate output on the console. You will work with the code in the `application.c` file.

Locate the `reader` method in the `application.c` file. Whenever you press a key on the keyboard, the `sci_interrupt` method (an interrupt handler) located in the `sciTinyTimber.c` file is invoked and the entered character is read from the serial port. This interrupt handler then invokes the `reader` method and provides to it the character read as a parameter. In summary, the `reader` method is executed whenever you press a key on the keyboard. It also prints the character to the console. For example, if you type character 'R', you will see `Rcv: 'R'` printed on the console.

Only *one* character can be read *at a time* from the serial port. Now you will learn how to take *integer* input. In order to read an integer, we have to type each character that constitutes the integer. In addition, you also need to type a special *delimiter* character, for example, 'e' to specify the end of the integer input. For example, reading integer -53 from the keyboard requires you to type '-' (the sign), '5' (the first digit), '3' (the second digit), and finally, 'e' (the delimiter).

Since the `reader` method is called each time a character is typed, we have to *store* all the previous characters of the integer number being typed in the keyboard before reading the next character. Therefore, in order to store every character that is entered until the delimiter is hit, you need to declare a string, that is, a character array of a **fixed** size. Every time a character is hit, it is compared with the delimiter ('e'). If the character typed is the delimiter, then you need to store the null character `'\0'` in the string to specify the end of the input integer[2]. Now the string can be converted to its integer representation using the `atoi` function. An example declaration of a fixed-size string and use of the `atoi` function is the following:

```
int num;
char buf[20];
buf[0]='-';
buf[1]='5';
buf[2]='3';
buf[3]='\0';
num = atoi(buf);   //variable num is equal to integer -53
```

Now you will learn how to output to the console. You can only print a character or a string on the console using the operations `SCI_WRITECHAR()` or `SCI_WRITE()`, respectively. Locate at least one use of `SCI_WRITECHAR()` and one use of `SCI_WRITE()` in the `application.c` file. In order to write the value of an integer variable in console, you first have to convert the integer into a string, and then print that string using `SCI_WRITE()`. For the conversion, the `snprintf` function can be used.

[2]Make sure that the character array is large enough to store any number you could possible use, including an initial '-' and the trailing `'\0'`. Unintended writes of data outside the limits of a character array can cause many hard-to-solve problems in your software!

Problem 2: Implement a function that reads a sequence of integers, and always keeps the last three integers (the "3-history"). Each time a new integer is entered your function should print the sum and median of the 3-history.[3] Typing 'F' should erase the 3-history and wait for new integers to be entered.

An example output produced when entering the sequence `13 -7 20 -1`:

```
Rcv: 'F'
The 3-history has been erased
Rcv: '1'
Rcv: '3'
Rcv: 'e'
Entered integer 13: sum = 13, median = 13
Rcv: '-'
Rcv: '7'
Rcv: 'e'
Entered integer -7: sum =  6, median =  3
Rcv: '2'
Rcv: '0'
Rcv: 'e'
Entered integer 20: sum = 26, median = 13
Rcv: '-'
Rcv: '1'
Rcv: 'e'
Entered integer -1: sum = 12, median = -1
```

Note that the 3-history as well as the character array used for parsing the integers are state variables, i.e. their contents need to be stored between each call to the `reader` method. You should therefore place these state variables (and any other data relating to them) inside a suitable object.

Assistant's approval: .....

IMPORTANT! After approval you should submit your software code in Canvas (under 'Modules'/'Laboratory Software'). The code should be submitted as is, that is, in the shape it was at the time of approval (without modifications).

Create a '.zip' archive file, containing all your modified C files, and name the archive file '`Part0_2_PGID.zip`' (where `PGID` is your project group name). Typically, you only need to include the `application.c` file in the archive file, but if you have created additional files you should include them too.

A short `User's Guide`, describing how to control the software with the key commands, should be included (by means of C program comments or console print-outs) in `application.c`.

[3] If there are only two integers, a_1 and a_2, in the 3-history the median is the average, $(a_1 + a_2)/2$, (integer part only, fractions truncated.) If there is only one integer, a_1, in the 3-history the median is a_1.

Step 3: Preliminaries for Part 1 – task period

You already listened to the Brother John melody at the first exercise class. A **melody** is a sequence of several **tones**. Brother John has 32 tones. The Brother John melody is generated by playing its tones one by one: the first tone is played, then the second tone is played, etc. After the 32^{nd} tone finishes, the sequence of tones is repeated starting from the first tone. In others words, the Brother John melody is a cyclic sequence of 32 tones.

You need to implement program code to generate one particular tone in **Part 1** of the laboratory assignment. Before that, in this part, you will learn the background regarding how to generate a tone, particularly, how to compute the period of the task that produces the waveform of the tone. A particular tone is generated by *periodically* writing alternating '1's and '0's to the DAC (digital-to-analog converter) of the MD407 card. In other words, a *tone-generating task* periodically writes the digits of the following sequence to the DAC:

$$1, \quad 0, \quad 1, \quad 0, \quad 1, \quad 0, \ldots$$

The time between two consecutive writes is the ***period*** of the tone-generating task. The period of the task is computed from the ***frequency*** of the tone. If the required frequency of the tone is f Hz, then the corresponding period of the task must be $\frac{1}{2f}$ sec.

Problem 3.a: Consider that alternating '1's and '0's are written to the DAC so that a tone of frequency 200 Hz is produced. The period of the tone-generating task is equal to one of the following. Tick the correct one. [Hint: 1 sec = 10^3 ms =$10^6 \mu$s]

Answer: (a) 5000 μs (b) 2500 μs (c) 2000 μs

Problem 3.b: Consider that the DAC is written with alternating '1's and '0's so that a tone of frequency 1 kHz is produced. Find the period of the tone-generating task.

Answer: ..... μs

Assistant's approval: .....

Step 4: Preliminaries for Part 2 – frequency index

At this point, you know how to compute the period of the tone-generating task for a *given* frequency. Therefore, if the frequency for each of the 32 tones of the Brother John melody is known, you can compute the corresponding task period for each of the frequencies. For practical reasons, **Part 2** of the laboratory assignment will refer to a ***frequency index***, rather than the frequency itself, for each of the 32 tones.

A frequency index i is an integer, e.g., $i = -3$. For a given frequency index i, the corresponding frequency is denoted by $p(i)$. Now you will learn how to compute the frequency corresponding to any given frequency index.

The frequency index $i = 0$ is called the *base frequency index*. The frequency for the base index is called the **base frequency**, which in our case is $p(0) = 440$ Hz (see footnote[4]).

> **Problem 4.a:** Consider that the DAC is written with alternating '1's and '0's so that a tone with base frequency $p(0)$ is produced. Find the period of the tone-generating task.
>
> **Answer:** ..... μs

Given the base frequency $p(0)$, frequencies for other indices can be computed based on one of the various *temperament standards* used for musical instruments. In Part 2 of the laboratory assignment, one such standard called equal-tempered 12-tone scale is used. According to this standard, the ratio of $\frac{p(i+1)}{p(i)}$ always equals the twelfth root of 2. In other words, given any two consecutive frequency indices i and $(i+1)$, the frequencies $p(i)$ and $p(i+1)$ are related as follows:

$$\frac{p(i+1)}{p(i)} = 2^{\frac{1}{12}}$$

Simple arithmetic shows that the frequency $p(k)$ and $p(i)$ for frequency indices k and i are related as follows:

$$\ldots = \frac{p(i+1)}{p(i)} = \frac{p(i+2)}{p(i+1)} = \ldots = \frac{p(k-1)}{p(k-2)} = \frac{p(k)}{p(k-1)} = \ldots = 2^{\frac{1}{12}}$$

which yields

$$\frac{p(k)}{p(i)} = 2^{\frac{k-i}{12}} \tag{1}$$

> **Problem 4.b:** Given that $p(0) = 440$ Hz, the frequency $p(k)$ for index k can be computed using Equation (1). Complete the following expression for frequency $p(k)$ as a function of frequency index k:
>
> **Answer:** $p(k) = 440 \times$ .....

[4]This is the frequency of the musical note A, which is a commonly-used audio frequency reference for calibration of acoustic equipment and tuning of musical instruments.

Problem 4.c: As mentioned before the period of the tone-generating task must be $\frac{1}{2f}$ sec in order to produce a tone of frequency f Hz. What should the period of the tone-generating task be when (a) the frequency index is −7, and (b) the frequency index is 9?

Answer: (a) ..... μs (b) ..... μs

Now you know how to compute the period of the tone-generating task that corresponds to a given frequency index k, assuming base frequency $p(0)$. Since the base frequency $p(0) = 440$ Hz is known, we can compute the task periods corresponding to the tones in any given melody if all frequency indices for the tones are available. The Brother John melody consists of the following 32 tones expressed in the form of frequency indices:

0	2	4	0	0	2	4	0	4	5	7	4	5	7	7	9	7	5	4	0	7	9	7	5	4	0	0	-5	0	0	-5	0

Table 1: The 32 frequency indices for the Brother John melody

Problem 4.d: Add a one-dimensional integer array of size 32 to your application code and copy into it the 32 frequency indices for the Brother John melody, in the order given by Table 1.

Problem 4.e: Inspect the 32 frequency indices given in Table 1 and find the maximum and minimum frequency indices.

Answer: Maximum is ..... Minimum is .....

Assistant's approval: .....

Step 5: Preliminaries for Part 2 – transposition

The magnitude (value) of all the frequency indices in Table 1 can be increased or decreased using a *transpose* parameter called `key`. The value of `key` can be in range $[-5 \ldots 5]$. The frequency indices in Table 1 are for `key` = 0, which is the default[5] set of frequencies for the Brother John melody. The value of `key` is added to each of the 32 frequency indices in Table 1, thereby producing new frequency indices for the given `key`. For example, if `key` = −2, then an offset of -2 is added to each of the 32 frequency indices given in Table 1 and a new[6] set of frequencies will be used when playing the melody.

[5]That is, playing the melody in the default key of A.
[6]That is, transposing the melody to the key of G.

Problem 5.a: What are the first 10 frequency indices of the Brother John melody for `key` = −5?

Answer: .....

Problem 5.b: Inspect the 32 frequency indices given for `key` = 0. If the `key` can be within the range of [−5 . . . 5], what is the maximum and minimum frequency index of the Brother John melody for any given `key`?

Answer: Maximum is ..... Minimum is .....

Let `max_index` and `min_index` be the maximum and minimum indices, respectively, obtained in the solution of Problem 5.b.

Problem 5.c: Calculate the periods of the tone-generation task corresponding to all frequency indices in the range [`min_index` . . . `max_index`]. The period values should be pre-computed (see footnote[7]), meaning that they should not be calculated at run-time while the software is running time-critical tasks.

Add another one-dimensional integer[8] array, named `period`, to your application code and copy the pre-computed period values into the array. The array should have a size of (`max_index` − `min_index` + 1) elements.

Problem 5.d: In the C programming language, array indices cannot be negative. Consequently, the indices for the new array will not correspond directly to the frequency indices, but will have to be translated. To that end, when the frequency index is k (where `min_index` $\leq k \leq$ `max_index`), the period array element `period[`x`]` needs to be accessed (where $x \geq 0$).

Express the period array index x as a function of the frequency index k:

Answer: $x(k)$ = ...

Assistant's approval: .....

[7] Hint: It is recommended that you use a tool to do the manual computation of periods, for example, you can use a spreadsheet program like Excel, a programming language like Python, or even MATLAB.

[8] If a computed period is not an integer number, use only the integer part of the result. There is no need to involve floating-point numbers at run-time as the effects of using integer values will not be perceived by the human ear.

Step 6: Preliminaries for Part 2 – period lookup

You stored frequency indices of Brother John melody in an array based on Table 1, and now also have an array of all required periods for the tone-generating task.

> **Problem 6:** Add a function to your application code that takes the `key` as input from the keyboard (in the form of an integer number) and prints the periods corresponding to the 32 frequency indices of the Brother John melody for the input `key`.
>
> Print out the periods as follows: First print "Key: " followed by the chosen key value and a line break. Then print the 32 period values on one line, each value separated by a space character, and terminate with a line break.

Assistant's approval: .....

IMPORTANT! Do not dispose of your program code for Part 0. Several pieces of code will be reused in the later parts of the laboratory assignment, for example your code for parsing integer numbers from Problem 2 and the array of pre-computed tone generator periods that your created in Problem 5.c.

That's it for Part 0.
Good luck with Part 1 and Part 2!