Objective: The purpose of Part 2 of the lab is to design a music player capable of performing the song Brother John. First, you will develop the simple tone generator of Part 1 into a small synthesizer/sequencer combo that can play the tune in isolation (Step 1 and Step 2). You will then further develop your player to become a network-aware application that may participate in coordinated Brother John performances together with all other like-minded MD407 boards it finds on the local network (Step 3 and Step 4).

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 sign your lab-PM and mark the corresponding examination objective as 'Passed' in Canvas. You can then continue with the next problem.

Step 1: The Brother John player – the design

Starting with the tone generator from Part 1, you should now extend your software design with the functionality of playing notes of varying frequency and length.

The frequency of a note is expressed in terms of a ***frequency index***, where a frequency index of 0 represents the ***base frequency***. Varying the frequency of a note means choosing among the tones in the Western 12-tone scale. Recall that, on a piano keyboard, there is a pattern of 12 keys (7 white and 5 black) that repeats itself over the entire keyboard. These 12 keys constitute an octave.

The relation between tones that are an octave apart on the keyboard is well defined: their frequency ratio is exactly 2. The frequencies of the remaining tones in the scale can be defined according one of the many temperament systems that are available. The temperament system used in this laboratory assignment is the equal-tempered 12-tone scale, which consists of a range of frequencies $p(i)$ (where i is the frequency index) such that the ratio $p(i+1)/p(i)$ always equals the twelfth root of 2. The default base frequency in this assignment is $p(0) = 440$ Hz.

The Brother John tune is a (cyclic) sequence of 32 notes, with the frequency of each note being defined by the following list of 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

The duration of a note is expressed in terms of the length of a ***beat***, which in turn is determined by the tune's ***tempo*** as measured in beats per minute (bpm). All notes in the Brother John tune are not of equal length, but exhibit the following (cyclic) pattern:

$$a\ a\ a\ a\ a\ a\ a\ a\ a\ a\ b\ a\ a\ b\ c\ c\ c\ c\ a\ a\ c\ c\ c\ c\ a\ a\ a\ a\ b\ a\ a\ b$$

Here, a represents a length of one beat, b represents a length of two beats (i.e., $2a$) and c represents a length of half a beat (i.e., $a/2$). The length of a beat is set by selecting a tempo for the tune. The default tempo in this assignment is 120 bpm, which means that the default length of a beat is 500 ms.

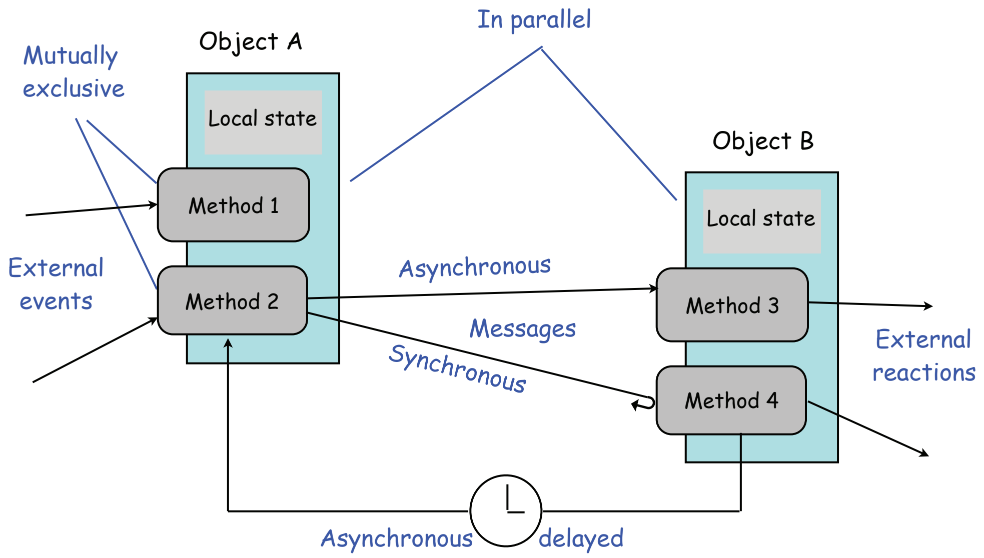

Figure 1: An example of an access graph

To give the tune some character a small gap of silence of suitable length should be inserted at the end of each note, by shortening the actual note duration by the same amount. A suitable length of the gap is 50–100 ms.

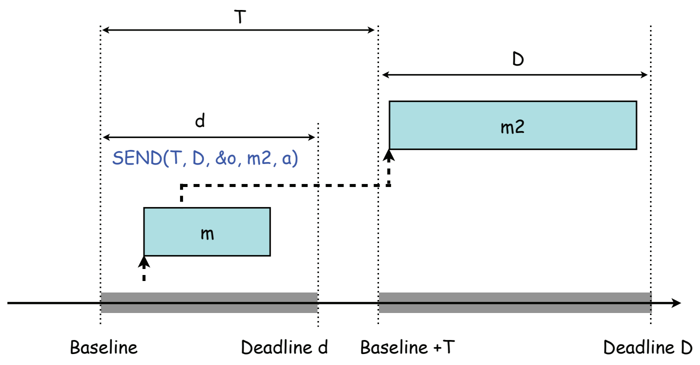

Figure 2: An example of a timing diagram

The performance of Brother John tune is controlled by the parameters `key` and `tempo`. The value of `key` is used as an offset to the frequency index of each note, thereby transposing the tune from the default key of A (parameter `key` = 0) to any other key within a given range. The value of `tempo` determines the length of a beat.

IMPORTANT! Before you begin writing any code you should get your design ideas approved by the teacher/assistant. To illustrate your design you should use access graphs and timing diagrams, similar to the ones shown in Figure 1 and Figure 2.

> **Problem 1.a:** Using access graphs illustrate (i) how the tone generator will be controlled to play a note of a certain length, including the small gap of silence, (ii) how the tone generator will be controlled to play the cyclic sequence of 32 notes constituting the Brother John tune, and (iii) how the key and tempo of the tune can be dynamically changed from the keyboard while the tune is being played.
>
> **Problem 1.b:** Using timing diagrams illustrate how the tone generator will be controlled to (i) play a note of a certain length, including the small gap of silence, and (ii) play the next note in the tune after the gap.
>
> *Assistant's approval:* ....

Step 2: The Brother John player – the code

You should now implement the player design that you developed in Step 1. Define suitable data structures for storing the frequency and length information[1], and write code for the methods shown in your access graphs and timing diagrams. Make sure that your `reader` function is able to dynamically supply the `key` and and `tempo` parameters to the relevant software parts. Your player should able to handle a `key` parameter in the range of $[-5 \dots 5]$ and a `tempo` parameter in the range of $[60 \dots 240]$ bpm. The `key` and and `tempo` parameters should be read from the keyboard in the form of integer numbers[2].

> **Problem 2:** Demonstrate that your software implementation is able to produce the notes that you recognize as the Brother John tune, and that the key and tempo can be dynamically changed from the keyboard while the tune is being played in a cyclic fashion.
>
> *Assistant's approval:* ....

IMPORTANT! After approval you should submit your software code in Canvas (under 'Modules'/'Laboratory Software'). Typically, you only need to submit the `application.c` file, but if you have multiple files you should submit a file archive.

[1]Reuse the array of pre-computed tone generator periods that you defined in Part 0.

[2]Reuse the software code for reading integer numbers that you wrote in Part 0.

Step 3: Yes, we CAN!

You will now enable your music-playing boards to communicate with each other using broadcast messages over the CAN bus. When you connect your board to the CAN bus your program must be prepared to run in either ***leader mode*** or in ***slave mode***. In leader mode your board's (and every other connected board's) music player will be controlled directly via your keyboard. That is, whenever you give a command via the keyboard a corresponding command should be sent out on the CAN bus. In slave mode your player will be controlled by another project group's board via the CAN bus.

You should first familiarize yourself with the CAN device driver. Locate the `receiver` method in the `application.c` file. Whenever a message is received from the CAN bus, the `can_interrupt` method (an interrupt handler) located in the `canTinyTimber.c` file is invoked and the message is read from the MD407 board's CAN controller. The interrupt handler then invokes the `receiver` method, within which you can acquire and decode the contents of the received CAN message.

To acquire the contents of the received CAN message, you should use the built-in macro `CAN_RECEIVE`. Correspondingly, to send a new CAN message you should use the built-in macro `CAN_SEND`. Locate the use of `CAN_RECEIVE` and `CAN_SEND` in the `application.c` file, and notice that they use a pointer to a variable of data type `CANMsg` as the second parameter. The data type `CANMsg`, defined in the `canTinyTimber.h` file, is used for storing only the user-relevant parts of the CAN message, such as the message identifier and the (maximum 8-byte) message payload. In TinyTimber's CAN device driver the 11-bit identifier is stored in two parts in the `CANMsg` type: one part consisting of the four least significant bits of the identifier (referred to as `nodeid`), and one part containing the 7 most significant bits of the identifier (referred to as `msgid`). This partitioning of the message identifier allows you and the other groups to enumerate your MD407 boards (via `nodeid`) and the transmitted messages (via `msgid`).

When designing your software it is important to know that <u>the CAN controller does not receive what it sends out</u>. You therefore need to make sure that your software does not rely on incoming copies of the messages you send. However, in order to test your software before you connect with any other boards, you will make temporary use of a ***loopback feature*** to verify that you can play in both leader mode and slave mode.

To enable the loopback feature of the CAN device driver for your MD407 board you should proceed as follows:

- Connect the primary CAN interface (CAN1) to the secondary CAN interface (CAN2) by means of the short white (loopback) cable provided by the laboratory assistant.
- Configure the CAN device driver by removing the comments on line 3 in `canTinyTimber.c` (that is, enable the `#define __CAN_LOOPBACK` statement).

As part of adapting your code to handle CAN messages you should add a *play function* to your program that allows you to start and stop the playing of the tune. When you run your program the tune should not be playing by default. A start command should initiate the playing from the beginning of the tune. Control the start/stop function with suitable keys on the keyboard. In addition, you should prepare the `receiver` method to print out the contents of messages received on the CAN bus regardless of which mode you are running in.

Problem 3.a: Run your program in leader mode, and make sure that you are able to control the tune (i.e. start, stop, change tempo and key) regardless of whether the loopback cable is connected or not. Observe the printouts of the contents of each received CAN message, and verify that the messages that you receive (when the cable is connected) contain the same information as the messages you send.

Assistant's approval: ....

Problem 3.b: Run your program in slave mode. Since you are not yet connected to any other board you should use a temporary solution and let your own keyboard send out CAN messages (just like you did in leader mode.) Connect the loopback cable and make sure that you can control the tune playback from the keyboard, and also observe the printouts of the received message contents. Disconnect the loopback cable and show that you no longer can control your tune.

Assistant's approval: ....

Step 4: Playing in Chorus and CANon

After testing the CAN message functionality in isolation using the loopback feature you will now enter cooperative mode and work with your fellow groups to jointly develop a communication protocol that facilitates synchronized playback of our favorite tune.

When demonstrating your software in cooperative mode there should be three boards connected to the CAN bus. If there are less than three groups collaborating, one of the groups should use their software on multiple boards.

Remove the short white cable, and connect the primary CAN interface (CAN1) to the boards of your fellow groups by means of the long black cable provided by the laboratory assistant. NOTE! Now is the time to disable the CAN loopback feature used in Step 3.

The initial requirements on the protocol are as follows:

1. Each board should have a *rank* that is unique. Assign among yourselves the ranks of the boards (using, for example, the `nodeid` part of the message identifier).

2. Decide among yourselves which board should run in leader mode. The other boards should run in slave mode. When the leader starts playing the tune the other boards should join in, without noticeable delay, in *chorus form* (i.e., all at the same time).

3. The leader also decides on the tempo (i.e., the length of a beat) and the key of the tune (i.e., the offset of the frequency index). At this point it is only required to set the tempo and key before starting to play the tune.

4. Playing should go on in a cyclic fashion until the leader chooses to quit, and stay synchronized (in tempo) no matter how long time this takes.

5. It is not required that the boards shall be able to switch dynamically between leader and slave mode. Separate software versions may be used.

IMPORTANT! Before you start adapting your code to the requirements above, be sure to meet with your fellow groups to discuss, design and agree upon a common protocol. Do not hesitate to ask your teacher or assistant for hints.

Problem 4.a: Demonstrate that your board, and your fellow groups' boards, are able to play the Brother John tune in chorus form, and in the key and tempo set initially by the leader board.

Assistant's approval: ....

Now add the following requirements on the protocol:

6. In addition to the chorus form the leader board shall be able to select *canon form*, in which the leader starts playing and the other boards join in (in rank order) delayed by n steps, where a step equals the currently chosen length of a beat, and n is a parameter of the canon form. A particularly nice value of n is 8. For a nightmare experience, try $n = 1$!
7. For either form (chorus or canon) the leader board shall be able to change the tempo and key dynamically while the tune is being played.
8. It is not required that the boards shall be able to switch dynamically between chorus and canon form. Separate software versions may be used.
9. It is only required that the boards shall be able to play the tune in canon form with parameters $n = 4$ and $n = 8$.

IMPORTANT! Before you start adapting your code to the requirements above, be sure to meet with your fellow groups to discuss, design and agree upon a common protocol. Do not hesitate to ask your teacher or assistant for hints.

Problem 4.b: Demonstrate that your board, and your fellow groups' boards, are able to <u>play the Brother John tune in canon form</u>, and in the key and tempo set initially by the leader board.

Problem 4.c: For either form (chorus or canon) show that the leader board can set an initial key and tempo, and then is able to change the tempo dynamically while the tune is being played.

Assistant's approval: ....

That's it for the laboratory assignment.
Thank You for the Music!