Dependable Real-Time Systems — EDA423 / DIT173

Homework assignment #1, 2023/24 Due no later than 23:59, April 23, 2024

Content:

The homework assignment consists of five problems divided into two parts: the *collaborative part* and the *non-collaborative part*. Problems 1-4 belong to the collaborative part (i.e., your group works with other groups in a meta-group) whereas Problem 5 belongs to the non-collaborative part (i.e., your group works alone).

The problems worth a total of 60 points as follows:

Problem 1: 10 points
Problem 2: 10 points
Problem 3: 10 points
Problem 4: 10 points
Problem 5: 20 points

Grading policy for this homework:

24–35 points ⇒ grade 3	24–43 points ⇒ grade G (GU)
36–47 points ⇒ grade 4	
48–60 points ⇒ grade 5	44–60 points ⇒ grade VG (GU)

Solution:

No solution provided.

IMPORTANT ISSUES — PLEASE READ THEM ALL BEFORE YOU START

---

1. Any group is welcome to work on HWA#1 from the laboratory room during any demonstration session even if that group has not booked a time slot for doing a demo during that session. If your examiner is there, you may discuss or ask for any clarification regarding HWA#1.
2. When the solution to the assignment problems is regarded as being complete, the meta-group should book a time with the course examiner for a demonstration of the solutions. The demonstration should take place during one of the scheduled demonstration sessions. Time slots for booking such a demonstration can be found in the April calender of the course in Canvas. All the meta-groups are listed below:

   Metagroup A: DRTS 1, DRTS 3, DRTS 5 and DRTS 13 (Splitters are with Arjun Kalamkar from Group 3)
   Metagroup B: DRTS 2, DRTS 7 and DRTS 14 (Splitters are with Andreas Karlsson from Group 14)
   Metagroup C: DRTS 4, DRTS 9, DRTS 10 and DRTS 15 (Splitters are with Daniel Rygaard from Group 9)
   Metagroup D: DRTS 6, DRTS 8, DRTS 11 and DRTS 12 (Splitters are with Simon Jansson from Group 6)

3. During the demonstration, the course examiner will assess and awards preliminary points for the functional correctness of the software code.
4. For some assignment problems, the assessment may require the examiner to download and run your software code on his own hardware. For this reason, your code must be compatible with the standard structure of the TinyTimber project.

5. Following the demonstration the course examiner will assess, and adjust awarded points for, the quality of the software design. To that end, you should submit the relevant parts of the software relating to the solution by the deadline. You will know your total points for HWA#1 during the oral examination of HWA#2 since the examiner may also ask you questions about HWA#1 during the oral examination of HWA#2.
6. Your group also needs to submit a report (in a PDF file) for each of the problems. In your report for each problem, your group needs to discuss the design of the solution, what worked well, what was challenging, and any experience that you want to share. There are also some specific instructions highlighted in **bold** text for each problem that you must discuss in your report. Note that the report is done by each group separately and no collaboration is allowed in writing the report.
7. The examiner during the demonstration will ask about the percentage of contribution by each member of a group. The percentage of contribution for each problem will be used to scale each student's points for that problem. Please agree on the percentage of contribution with your group member. In other words, the different members of the same group may not necessarily receive the same points. To that end, also state the percentage of contribution by each group member in the report for each of the problems.
8. It is STRICTLY FORBIDDEN to plagiarize other solutions. No group shall share any piece of code with any other group. Each group should construct their own solutions. However, you are allowed to discuss your design and protocol with other groups in your meta-group regarding the collaborative part. You are not allowed to discuss your solution with any other group regarding the non-collaborative part.
9. The dynamic behavior of your software is controlled via input from console. However, if you implemented the USER button and the LED functionality in the EDA223 course, you may decide to use that functionality for some part of the problems if you want (i.e., this is optional and you may instead choose to use input from the console to control the dynamic behavior of your software). Instructions for using the USER button and the LED functionality are explicitly given in this document.
10. **Demonstration:** One person on behalf of an entire meta-group needs to book exactly one time slot. The duration of each time slot is for 105 minutes and is used for demoing the software implemented by all the three/four groups in the meta-group depending on how many groups are there in your meta-group. All the members of all the three/four groups must be present during the scheduled demonstration time. The way the 105 minutes will be used during the demo is given below.

    During the first 45 minutes, all the three/four groups together will demo the software of the collaborative part. If the meta-group decides to make modification to their code, you will be allowed to do the modification and re-demo it during the already allotted 45 minutes of your collaborative part. In other words, the demo and possible re-demo of the collaborative part must be done during the first 45 minutes of the meta-group's scheduled demo time slot.

    The remaining 60 minutes of the time slot will be divided among the three/four groups to demo the software of the non-collaborative part as follows. If your meta group has three groups, each group gets 20 minutes and if your meta group has four groups, each group gets 15 minutes to demo it. If your group decides to make modification to your code, you will be allowed to do the modification and re-demo it during your already allotted 20 or 15 minutes. In other words, the demo and possible re-demo of the non-collaborative part must be done during your group's allotted 20 or 15 minutes.

Good Luck!

Collaborative part

Problem 1: Listen to the Round Robin

You should start working in collaborative part together with the other project groups in your meta-group appointed by the examiner. The first collaborative objective is to extend your melody player with *sequential form*, where the melody is played by the boards on the CAN bus in a "round robin" fashion. Among the members of your meta-group agree on a tune to play (yes, it is permitted to choose something else than the Brother John tune).

Start by preparing the CAN device driver to work in a more reliable mode, by removing the comments on line 21 in `canTinyTimber.c` (that is, enable the `#define __CAN_TxAck` statement). In this mode the CAN hardware will take advantage of an advanced handshake protocol that will allow each sender of a message to detect if the message has not been received correctly by the other boards. Then connect the primary CAN interface (CAN1) to the boards of your fellow groups by means of long cables and splitter modules. Each meta-group will be given splitters that are needed to connect the three/four boards of the three/four groups in your meta-group depending on how many groups are there in your meta group.

The initial requirements on the sequential form of the melody player 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 *conductor* mode. The other boards should run in *musician* mode.
3. The first board in rank order plays the first note of the melody, and then the responsibility to play changes for every note according to the (cyclic) rank order among the boards. For example, if there are three boards in the network:
   - the first board in rank order plays the first, fourth, seventh, ... notes.
   - the second board in rank order plays the second, fifth, eighth, ... notes.
   - the third board in rank order plays the third, sixth, ninth, ... notes.

   For example, if there are four boards in the network:
   - the first board in rank order plays the first, fifth, ninth, ... notes.
   - the second board in rank order plays the second, sixth, tenth, ... notes.
   - the third board in rank order plays the third, seventh, eleventh, ... notes.
   - the fourth board in rank order plays the fourth, eighth, twelfth, ... notes.

   Consequently, no board should play a note at the same time as any other board.

4. Before the playing of the melody is started the melody player should automatically detect how many boards are in the network. **In your report, please discuss how the melody player decides how many boards are there in the network.**

5. At the initialization, your code should also be able to configure any board either as a conductor or musician via user input from the keyboard such that there is at most one conductor at any time instant. After this initialization, it is not required that the boards shall be able to switch dynamically between conductor and musician mode, nor is it required that the melody player should detect how many boards are in the network after the melody has started playing.

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. **In your report, please discuss this common protocol.**

The special requirements on the conductor board are as follows:

1. The conductor board should be able to start or restart the melody from the beginning of the melody, and should also be able to stop the playing of the melody at any time.

2. The conductor board should be able to change the key and tempo dynamically while the melody is being played using input from the keyboard.

3. The conductor board should be able to reset the key and tempo to their default values when the character 'R' is entered from the keyboard. Alternatively, if your group implemented the USER button functionality in the EDA223 course, the conductor board should be able to reset the key and tempo to their default values by means of a 2-second PRESS-AND-HOLD activation of the USER button.

4. The conductor board should print the current tempo (bpm) every **T** seconds. You are free to choose the value of **T**, for example, 10 seconds. It should be possible to enable/disable this printing feature, by means of a command from the console keyboard. Alternatively, if your group implemented the LED functionality in the EDA223 course, the green LED on the conductor board should always blink at a frequency corresponding to the current tempo (bpm), fully synchronised with the melody.

The special requirements on the musician boards are as follows:

1. A musician board should have its speaker volume state (muted or unmuted) toggled when the character 'T' is entered from the keyboard. Alternatively, if your group implemented the USER button functionality in the EDA223 course, a musician board should have its speaker volume state (muted or unmuted) toggled for every valid activation of the board's USER button.

2. A musician board should print the message "MUTED" every **T** seconds when the volume state of that musician is muted. You are free to choose the value of **T**, for example, 5 seconds. It should be possible to enable/disable this printing feature, by means of a command from the console keyboard. Alternatively, if your group

implemented the LED functionality in the EDA223 course, the green LED on a musician board should be turned off ("unlit") if the board has its speaker muted, and turned on ("lit") if the board has its speaker unmuted.

Demonstration Setup: This problem should be demonstrated using the three/four boards depending on how may groups are there in your meta-group, with software from all the three/four different groups in your meta-group. This problem should also be able to demonstrate using only one board using the loop-back function with software only from your group. The initial state of the melody player should be that the melody is not playing. The default parameters for the melody player should be `key` = 0 and `tempo` = 120 bpm. Demonstrate that your collaborative group is able to play the melody in sequential form.

In your group report, discuss the design of your solution to this problem and also write the contribution by each member of your group as follows.

Contribution from group member 1 (name, %): ....

Contribution from group member 2 (name, %): ....

Contribution from group member 3 (name, %): ....

Documentation: Your demonstrated software code should be submitted in the following format. Create a '.zip' archive file, containing all your modified C files, and name the archive file `Problem1_DRTS_XX.zip` (where XX is your group number). Typically, only the `application.c` file needs to be included, but if you have created additional files you should include them too. In addition, also include the PDF of your report for this problem in the archive file. Alternatively, you can submit one '.zip' archive file for all the five problems in HWA#1 with all the source files marking each file clearly by problem number and one PDF report for all the five problems in HWA#1.

Maximum achievable score:	(10 points)
Amount of which that relates to functional correctness:	(5 points)
Amount of which that relates to software quality:	(3 points)
Amount of which that relates to your group report:	(2 points)

Problem 2: Embrace Your New Conductor

You should now extend your collaborative melody player to handle dynamic conductor change. That is, it should be possible for a musician board to claim, and obtain, conductorship while the melody continues to play.

The requirements on the dynamic conductor change functionality are as follows:

1. At any time it shall be possible for a musician board to claim conductorship, by means of a command from the console keyboard or a 5-second PRESS-AND-HOLD activation of the USER button.
2. Two or more simultaneous[1] conductorship claims from different boards should be resolved according to rank.
3. A conductor change means that the new conductor takes over the rights and responsibilities that were associated with the previous conductor. **In your report, discuss how a new musician takes the responsibility of a conductor and how the previous conductor is configured as a musician.**
4. During the conductor change the melody should continue playing using the key and tempo that were set prior to the conductor change.
5. A successful change of conductorship is indicated printing a message "Claimed Conductorship" in the new conductor's console. In addition, the previous conductor prints a message "Conductorship Void" in its console. Alternatively, if you implemented the LED functionality in EDA223 course, a successful change of conductorship is indicated by blinking the green LED on the new conductor board at a frequency corresponding to the current tempo (bpm), fully synchronised with the melody; and at the same time the previous conductor board has its green LED change from blinking to solid light as it now enters musician mode.

Demonstration Setup: This problem should be demonstrated using the boards of the groups in your meta-group, with software from all the different groups in your meta-group. This problem should be able to demonstrate using only one board using the loop-back function with software only from your group. The initial state of the collaborative melody player should be to have no default conductor. This means that one of the boards needs to claim conductorship before the melody can be started. Demonstrate that your collaborative group is able to handle a dynamic conductor change, that is, while the melody continues to play in sequential form.

In your group report, discuss the design of your solution to this problem and also write the contribution by each member of your group as follows.

[1] The exact interpretation of 'simultaneous' is delegated to the collaborative group.

Contribution from group member 1 (name, %): ....

Contribution from group member 2 (name, %): ....

Contribution from group member 3 (name, %): ....

Documentation: Your demonstrated software code should be submitted in the following format. Create a '.zip' archive file, containing all your modified C files, and name the archive file `Problem2_DRTS_XX.zip` (where XX is your group number). Typically, only the `application.c` file needs to be included, but if you have created additional files you should include them too. In addition, also include the PDF of your report for this problem in the archive file. Alternatively, you can submit one '.zip' archive file for all the five problems in HWA#1 with all the source files marking each file clearly by problem number and one PDF report for all the five problems in HWA#1.

Maximum achievable score:	(10 points)
Amount of which that relates to functional correctness:	(5 points)
Amount of which that relates to software quality:	(3 points)
Amount of which that relates to your group report:	(2 points)

Problem 3: The Sound of Silence

It is now time to add a dependable sequential form to your collaborative melody player, in the sense that it should be able to withstand silent failures of musician boards. From a listener's point of view this means that no gaps should appear in the music even if a musician is no longer able to play its assigned note(s). To achieve this the melody player must be able to detect a silent failure, and also be able to take suitable action to make sure a note assigned to the failed board will be played by another board.

The requirements on the dependable sequential form are as follows:

1. The loss of a board (silent failure) should be handled by means of a *passive backup*. That is, only when the fail silent board has been detected should another board take over the playing of the lost board's note(s).

2. It is not allowed to have an *active backup* mechanism, where there always is a board that simultaneously plays the same note as some other board.

3. It should be possible, on the conductor board's console output, to observe the current status of all boards. That is, to see at all times which boards are available for playing notes and which boards have entered silent failure state. When the character 'M' is entered in the conductor's console, the current status of all the boards should be printed. We choose the character 'M' to specify *membership*. **In your report, discuss how the conductor decides who are the members of the network. Explain in details the protocol you used in your design to keep track of the up-to-date membership information in the network.**

In order to avoid the hassle of resets and program downloads you should simulate board failure by an ability to instruct a board to enter silent failure state (stop playing) or leave silent failure state (seamlessly join sequential form again) as described below.

1. A board should enter and leave the silent failure state in the following three modes (you need to implement all three):

   - Failure Mode F1: A board enters silent failure state by means of a console command or a 2-second PRESS-AND-HOLD activation of the USER button. A board leaves silent failure state by means of one more console command or PRESS-AND-HOLD activation of the USER button.
   - Failure Mode F2: A board enters silent failure state by means of a console command or a 2-second PRESS-AND-HOLD activation of the USER button. In the absence of user interaction, a board leaves silent failure state automatically after a random delay in the range $[5 \ldots 10]$ seconds.

- Failure Mode F3: A board enters silent failure state by detecting that its connection to the CAN bus is lost[2]. Note that when there are exactly two functional boards and we now disconnect the CAN cable between these two nodes then none of the nodes knows which one the we want to be removed from the network. In such a case, your meta-group must keep the conductor board playing while the musician board is removed from the network. A board leaves silent failure state by detecting that its connection to the CAN bus has been restored when you physically connect the CAN cable to the board.

2. Print the message "Silent Failure" in the console if the board has entered the silent failure state. Alternatively, if you implemented the LED functionality in the EDA223 course, the green LED on a board should be turned off ("unlit") if the board has entered the silent failure state.

3. Print the message "Leave Silent Failure" in the console when the board joins the network again. Alternatively, if you implemented the LED functionality in the EDA223 course, the green LED on a board should be turned on ("lit") when the board joins the network again.

The special requirements on the collaborative melody player are as follows:

1. The melody player should be able to sustain up to two concurrent silent failures on separate musician boards if your meta group has three boards. The melody player should be able to sustain up to three concurrent silent failures on separate musician boards if your meta group has four boards.

2. The melody player's musician failure recovery time (the time that passes from when a musician board enters silent failure state until the corresponding passive backup is enabled) should not exceed one second. When a musician board joins back to the network, it should continue playing by synchronizing itself with the sequential form of the ongoing melody. **In your report, discuss how this synchronization is implemented, i.e., how the musician node that just joins the network knows which notes to play.**

3. The nodes in the network can exhibit failure modes F1, F2, and F3 in any arbitrary order. For example, the first node may fail with F1, the second with F3, and the third one with F2 and so on.

Demonstration Setup: This problem should be demonstrated using the boards of the groups in your meta-group, with software from all the different groups in your meta-group. This problem should be able to demonstrate using only one board using the loop-back function with software only from your group. Demonstrate that your collaborative group is able to play the melody in dependable sequential form, which should include handling silent failures of musician boards.

In your group report, discuss the design of your solution to this problem and also write the contribution by each member of your group as follows.

Contribution from group member 1 (name, %): ....

[2]A lost connection is achieved by physically removing the CAN cable from the board.

Contribution from group member 2 (name, %): ....

Contribution from group member 3 (name, %): ....

Documentation: Your demonstrated software code should be submitted in the following format. Create a '.zip' archive file, containing all your modified C files, and name the archive file `Problem3_DRTS_XX.zip` (where XX is your group number). Typically, only the `application.c` file needs to be included, but if you have created additional files you should include them too. In addition, also include the PDF of your report for this problem in the archive file. Alternatively, you can submit one '.zip' archive file for all the five problems in HWA#1 with all the source files marking each file clearly by problem number and one PDF report for all the five problems in HWA#1.

Maximum achievable score:	(10 points)
Amount of which that relates to functional correctness:	(5 points)
Amount of which that relates to software quality:	(3 points)
Amount of which that relates to your group report:	(2 points)

Problem 4: Silence that Conductor

Conclude the collaborative part by extending the dependable sequential form of your collaborative melody player to also withstand silent failure of the conductor board. Silent failure of the conductor board is initiated, and handled by the melody player, in the same three possible ways as the silent failure of a musician board (i.e., all three F1, F2, and F3 failure modes need to be supported in your code to simulate the failure of the conductor board), with the following exceptions:

1. Since silent failure of the conductor will leave the other boards without authority when it comes to tempo and key changes, a new conductor should be established automatically once the loss of the conductor is detected. **In your report, discuss how a musician board is elected as the new conductor when an existing conductor board fails.**

2. Automatic conductor change means that the new conductor takes over the rights and responsibilities that were associated with the previous conductor, including the function to be aware of, and display on the console, the current status of all boards when the character 'M' is entered.

3. During automatic conductor change the melody should continue playing using the key and tempo that were set prior to the silent failure of the conductor.

4. A successful automatic change of conductorship is indicated printing a message "I Am The New Conductor" in the new conductor's console. In addition, the previous conductor prints a console message "Conductorship Void Due To Failure". Alternatively, if you implemented the LED functionality in EDA223 course, a successful change of conductorship is indicated by blinking the green LED on the new conductor board at a frequency corresponding to the current tempo (bpm), fully synchronised with the melody; and at the same time the previous conductor board has its green LED change from blinking to unlit as it now enters silent failure state.

5. When the previous conductor leaves silent failure state it should not automatically claim to get conductorship back, but should instead assume musician mode and prints a message "I Now Join As A Musician" or have its green LED change to solid light.

6. The nodes in the network can exhibit failure modes F1, F2, and F3 in any arbitrary order. For example, the first node may fail with F1, the second with F3, and the third one with F2 and so on. The failure of the last board is simulated using F1 or F2 because simulating F3 is not possible without a network having at least two alive boards that are connected by a CAN cable.

7. If all the boards of the network fail silently one-by-one, there shall not be any melody (called, a dead network); and as soon as a node leaves the silent failure state from

a dead network, that very first node becomes the conductor and starts playing the melody from the beginning.

The special requirements on the collaborative melody player are as follows:

1. The melody player's conductor failure recovery time (the time that passes from when a conductor board enters silent failure state until a successful conductor change has taken place) should not exceed one second.

Demonstration Setup: This problem should be demonstrated using the boards of the groups in your meta-group, with software from all the different groups in your meta-group. This problem should be able to demonstrate using only one board using the loop-back function with software only from your group. Demonstrate that your collaborative group is able to handle a silent failure of the conductor or musician board, while the melody continues to play in sequential form until all the boards fail.

In your group report, discuss the design of your solution to this problem and also write the contribution by each member of your group as follows.

Contribution from group member 1 (name, %): ....

Contribution from group member 2 (name, %): ....

Contribution from group member 3 (name, %): ....

Documentation: Your demonstrated software code should be submitted in the following format. Create a '.zip' archive file, containing all your modified C files, and name the archive file `Problem4_DRTS_XX.zip` (where XX is your group number). Typically, only the `application.c` file needs to be included, but if you have created additional files you should include them too. In addition, also include the PDF of your report for this problem in the archive file. Alternatively, you can submit one '.zip' archive file for all the five problems in HWA#1 with all the source files marking each file clearly by problem number and one PDF report for all the five problems in HWA#1.

Maximum achievable score:	(10 points)
Amount of which that relates to functional correctness:	(5 points)
Amount of which that relates to software quality:	(3 points)
Amount of which that relates to your group report:	(2 points)

Non-Collaborative part

Problem 5: Rule that Regulator

You should now design a regulator for CAN messages, with the purpose of making sure that the inter-arrival time between two subsequent CAN messages is never shorter than a given bound, at the time when the messages are delivered to a receiving task (thus, preventing the occurrence of sporadic overruns). Note that the arrival of a message to a board and the delivery of the message are different events in the sense that if a message arrives too early, then the node may need to hold it for a certain amount of time before it is delivered to the application to respect the minimum inter-arrival time between any two consecutive messages. This problem does not require any melody player and you need to disable the melody player.

Your MD407 board should be used as the source for CAN messages, by means of the CAN loopback feature. To that end, add functionality to your program so that a CAN message is transmitted every time you enter the character 'O' in the console ('O' is for sending *one* message). Alternatively, if you have your USER button implemented in the EDA223 course, add code to the button call-back method so that a CAN message is transmitted for each valid activation of the USER button.

The actual data of the CAN message is irrelevant for this problem. The only relevant part of a message is that each message must have a valid sequence number that is stored in the `msgId` part of `CANmsg` message structure. Such a message without any actual data may be used, for example, to diagnose a network to know if transmitted messages with specific sequence numbers are received correctly or not.

The special requirements on the CAN message source board are as follows:

1. Each transmitted CAN message should be tagged with a unique sequence number (range 0–127, post-incremented), which should be stored in the `msgId` part of the `CANmsg` message structure.
2. The sequence number of each transmitted CAN message should be printed out. It should be possible to enable/disable[3] this feature, by means of a command from the console keyboard.
3. The board should be able to enter a special burst transmission mode if the user enters the character 'B' in the console. In such a burst transmission mode, the board should begin transmitting one CAN message every 0.5 seconds until the user enters the character 'X' in the console. Alternatively, if you have your USER button implemented in the EDA223 course, the board should be able to enter a special burst transmission mode by means of a PRESS-AND-HOLD activation of the USER button:

[3]During the demonstration of your software the CAN message source print-outs should be disabled.

if the button is continuously pressed for more than 2 seconds the board should begin transmitting one CAN message every 0.5 seconds until the button is released.

Your MD407 board should also run the regulator software. To that end, modify the code in the `receiver()` method so that the inter-arrival times of subsequent received CAN messages are monitored, and adjusted if necessary, before the messages are delivered. Indicate when a CAN message is ready for delivery, by printing out — the actual time of delivery relative to the start time of the program – the sequence number and delivery time (expressed in seconds) of the message.

The functionality of the regulator should be as follows. Let t_{in}^n indicate the arrival time of CAN message n. Let t_{out}^n indicate the delivery time of CAN message n. For all regulated subsequent CAN messages $n-1$ and n it should apply that $t_{out}^n - t_{out}^{n-1} = \Delta$, where Δ is the required minimum inter-arrival time after regulation. Note that the delivery time is NOT when the message arrives but it is the time when the message is made available to the application after any needed delay to ensure the minimum inter-arrival time between delivering two consecutive messages. No regulation is necessary in the case where $t_{in}^n - t_{out}^{n-1} \geq \Delta$, in which case CAN message n should be delivered immediately after arrival (i.e., $t_{out}^n = t_{in}^n$).

To handle bursts of CAN messages with inter-arrival times shorter than Δ, your regulator needs to implement a buffering function to prevent messages to be discarded. Having an efficient buffering mechanism is important to reduce overhead[4]. In many embedded systems, reducing the memory footprint is crucial. Devise a strategy for efficient buffering mechanism that will allow the program to not unnecessarily use extra memory for buffering. Your regulator should be able to handle bursts of at least B_{burst} messages. For message bursts larger than the capacity of the regulator, it may be necessary to discard messages.

Demonstration Setup: Demonstrate that your MD407 board's regulator software is able to guarantee that, for any arrival pattern of CAN messages generated by the message source board,the inter-arrival time of two subsequent delivered messages is never shorter than Δ second. The different values of Δ that your program should work for are {1sec, 2sec, 5sec} which can be selected by an user input. By default, we set $\Delta = 1$.

In the report, you need to add a performance analysis of your regulator software, as specified below:

- **The principle for calculating delays for messages that are buffered within the regulator. A timing diagram should be included as illustration.**
- **The implementation cost for any data structure you have added for the regulation, in terms of (a) memory size of the data structure and (b) time complexity of operations on the data structure. Each cost should be expressed as a function of the buffering capacity B_{burst}.**
- **An estimation of the smallest possible Δ (required minimum inter-arrival time after regulation) that can be handled by your regulator implementation, assuming a buffering capacity $B_{\text{burst}} = 10$.**

[4]Hint: In this problem it is only necessary to preserve the `msgId` from each received CAN message. Think about TinyTimber's supported features for managing reactiveness, concurrency, etc.

To be awarded full points on this problem, your regulator software should not only fulfill the requirements listed above, but should also make excellent use of TinyTimber and its support for concurrency, object-orientation, reactiveness and timing awareness. This means that fewer software quality points will be awarded to overly-complicated solutions. In your group report, also write the contribution by each member of your group as follows

Contribution from group member 1 (name, %): ....

Contribution from group member 2 (name, %): ....

Contribution from group member 3 (name, %): ....

Documentation: Your demonstrated software code should be submitted in the following format. Create a '.zip' archive file, containing all your modified C files, and name the archive file `Problem5_DRTS_XX.zip` (where XX is your group number). Typically, only the `application.c` file needs to be included, but if you have created additional files you should include them too. In addition, also include the PDF of your report for this problem in the archive file. Alternatively, you can submit one '.zip' archive file for all the five problems in HWA#1 with all the source files marking each file clearly by problem number and one PDF report for all the five problems in HWA#1.

Maximum achievable score: (20 points)

Amount of which that relates to functional correctness: (8 points)

Amount of which that relates to software quality: (8 points)

Amount of which that relates to performance analysis and report: (4 points)