2IN70 - Track 2: Real-Time Architectures
Practical Training - Exercises 8 -

Today we will experiment with a networked system and gain insight into a distributed application comprised of several tasks communicating via the CAN bus. You will need to work together with another group. It is up to you to divide the work between your group members. Each joined group should hand in one set of solutions.

8.1 Connecting the boards via the CAN bus

In this exercise you will connect two boards via the CAN bus.

1. Make sure the switch `SW4` on both boards is in `STAND ALONE` mode.
2. Connect the two boards using the CAN cable.
3. Connect the light sensor to one board. *Double-check with your partner that the wires are correctly connected, to prevent destroying the board!*

Each board can be programmed independently, like it was done in previous exercises. For each board:

4. Connect the programmer to the board.
5. Connect the programmer to your computer via a USB port.
6. Connect the board to the power.

8.2 Distributed sensing and actuation

In this exercise you will gain insight into implementing precedence constraints between two tasks executing on two different ECUs connected via a CAN bus. In particular, `Task1` executing on one node will read the light sensor and communicate the value to `Task2` executing on another node, which will then turn on the leds accordingly. The CodeWarrior projects for this exercise are in the directories `exercise8_2_sender` and `exercise8_2_receiver`. Each of the two boards is meant to run one of the projects.

Note: make sure that only one project is open in CodeWarrior, otherwise it is difficult to see which code is being uploaded to the board.

1. **Study the task specification and implement the tasks. Make sure to initialize the CAN driver properly. Hand in the the two `main.c` files**

   Hint: The CAN protocol Use `CANId11()` to format the CAN frame id correctly when sending, e.g. `CANSendFrame(CANId11(LightId), ...)`.

   Hint: Study the type definition of `CAN_MSG` inside `CAN_driver.h` to understand how to extract data from a received CAN frame.

   Hint: Use the `OSQPend()` function to read the incoming message on the receiver side. It will return a pointer to a message of type `CAN_MSG`, which is defined in `CAN_driver.h`.

8.3 Distributed control

In this exercise you will reorganize the application: one board will contain all the sensors and actuators and the other will be responsible for the control. The CodeWarrior projects for this exercise are in the directories `exercise8_3_context` and `exercise8_3_controller`. Each of the two boards is meant to run one of the projects.

1. **Study the task specification and implement the tasks. Make sure to initialize the CAN driver properly. Hand in the the two `main.c` files**

   Hint: Use the `OSTimeDly()` function to delay the execution of `Task3`.