Automotive Networks 2019.2

Project Specification

Objective: Design and build a CAN ECU able to exchange CAN frames with other ECUs. Your ECU should be composed of a microcontroller, a CAN controller and a CAN transceiver. The microcontroller must be programmed in C/C++ and can be an Arduino or another platform. The CAN controller must be implemented in software using the microcontroller. The CAN transceiver will be provided, such that the ECU can be connected to a CAN bus. The CAN controller must be connected to a CAN bus through the CAN transceiver using the CAN RX and CAN TX pins. The controller has two main features: encode and decode CAN frames. The controller designed must be able to send and receive CAN2.0A and CAN2.0B frames. The correct functioning of your controller will be validated using a CAN bus shield. The CAN bus shield will send CAN frames that your controller must decode, and your controller must send CAN frames that the shield can decode. To design the CAN encoder and decoder you must comprehend the purpose of each bit and field of a CAN frame, what error conditions can occur, validate the CRC, transmit the ack bit, send an error frame if necessary, and synchronize to the bus. It is essential to spend time studying the CAN protocol to fully understand it. In a nutshell, your controller must be able to:

1) Send and receive data and remote frames checking the CRC and sending the ACK bit;

2) Identify the occurrence of error/overload frames;

3) Work at any CAN baud rate by making the CAN Bit Timing parameters configurable. Due to Arduino's limited processing speed, all teams must use a rate of 10 bps as reference during development. Just as an example, a 500 Kbps baud rate should be configured according to the following Bit Timing parameters:

a) fosc = 16MHz

b) Tq = 2/fosc

c) SJW = 1Tq

d) Prop Seg = 1Tq

e) Phase Seg 1 = 7Tq

f) Phase Seg 2 = 7Tq

Finally, each team will be designated a few CAN frames corresponding to specific signals that should be transmitted or received and interpreted in a particular way. Then, all ECUs designed will be connected to the same bus to exchange CAN frames among themselves and operate as they were the ECUs of an actual vehicle.

Platform and programming language: Arduino IDE or other equivalent and Processing IDE. The programming language must be C/C++. The use of Linux or any other High-Level OS is not allowed. Only microcontrollers and RTOS for microcontrollers are allowed.

Suggested platform: Arduino UNO.

Teams: 4 teams of 2 people.

Deadlines are for evaluating results and deliverables from students. Solve your questions during regular lessons, checkpoint lessons on demand and the classroom platform.

Deadlines and deliverables:

Deadline 1: 30/09/2019

Grade percentage: 10%

Deliverables: Block diagrams, state diagrams and flowcharts for the Bit Timing module needed for the CAN controller design.

Deadline 2: 14/10/2019

Grade percentage: 20%

Deliverables: Implementation for the Bit Timing module. Format: It must be possible to see the states in a plot using Arduino Serial Plotter. Also, there must be two buttons available: one to represent a soft sync and another to represent a hard sync **OR** one to represent a sync and another to enable hard sync.

Deadline 3: 21/10/2019

Grade percentage: 10%

Deliverables: Block diagrams and state diagrams for the CAN encoder/decoder modules needed for the CAN controller design.

Deadline 4: 04/11/2019

Grade percentage: 20%

Deliverables: Implementation of encoder and decoder modules in a simulation environment (PC). The teams are recommended to write their code in the programming language used in the target microcontroller to facilitate the next deadline. The program must be able to:

- Read a *txt* file with one or more frames;
- Print a status messages every iteration (current state, moment of execution: sample point or writing point, bit read / written, indicate if it is bit stuffing, error detected and other information that the team find to be important);
- Calculate and print the CRC at the CRC check;
- Encode a frame ready to send from a *struct* or *class* and output it to a *txt* file.

Deadline 5: 18/11/2019

Grade percentage: 15%

Deliverables: Implementation of encoder and decoder modules working along with the bit timing module already implemented. By now two ECUs must be able to read/send and save CAN frames talking to each other. Format: All the received bits must be printed at the end of the frame or when an Error occurs (the type of the error detected should be printed in the console). The fields of the received frame should be printed at the end. The fields of a frame to be transmitted should be printed before the beginning of transmission.

Deadline 6: 25/11/2019

Grade percentage: 15%

Deliverables: PowerPoint presentation in class, report, public exhibition and final project zip file. The presentation order will be defined randomly.

Deadline 7: 27/11/2019

Grade percentage: 10%

Deliverables: Integration of all ECUs working together in a single CAN bus. All teams will have specific messages to send and receive.

Message map:

Node A - Message ID: 0x301

Node B - Message ID: 0x10000000

Node C - Message ID: 0x302

Node D - Message ID: 0x10000001

Node X - Message ID: 0x10000003

All nodes must read from their predecessor and act according to the received data.
Data: 01 to turn LED off | 02 to turn LED on.

If a node receives a message while its LED is on it must send a message to turn off the LED of the next ECU and vice-versa.

All nodes must accept messages from Node X.