**CAN Bus Tutorial**

Introduction to Controller Area Network

<http://www.youtube.com/watch?v=sw3ADKPo1Uo>

<http://www.youtube.com/watch?v=RlZGwP0WyFY>

Texas Instruments’ detailed CANbus architecture (read sections 1 up to 5, excluding 5.1+)

<http://www.ti.com/lit/an/sloa101a/sloa101a.pdf>

Watch all 4 parts of the following video series:

<http://www.youtube.com/watch?v=m8HTVlEC7Lg>

DigiKey’s take on CANbus video (watch this then attempt the next link, **your goal is to understand it, not finish it**):

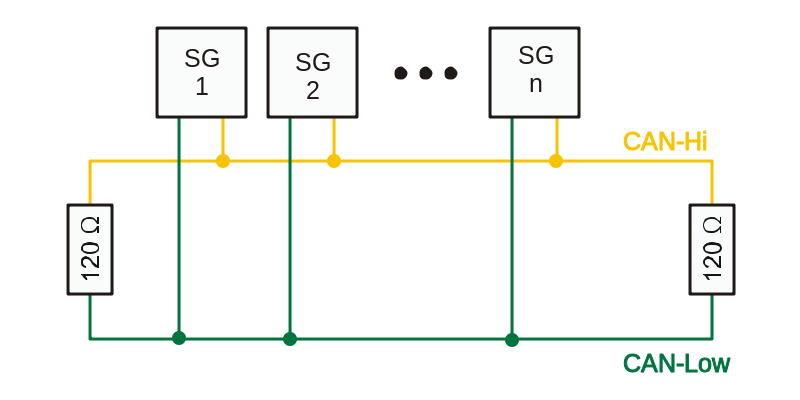
<http://www.youtube.com/watch?v=6KQylhGLvHo>

Digikey’s CAN bus tutorial:

<http://dkc1.digikey.com/us/en/tod/Renesas/CANBasicsPart-1_NoAudio/CANBasicsPart-1_NoAudio.html>

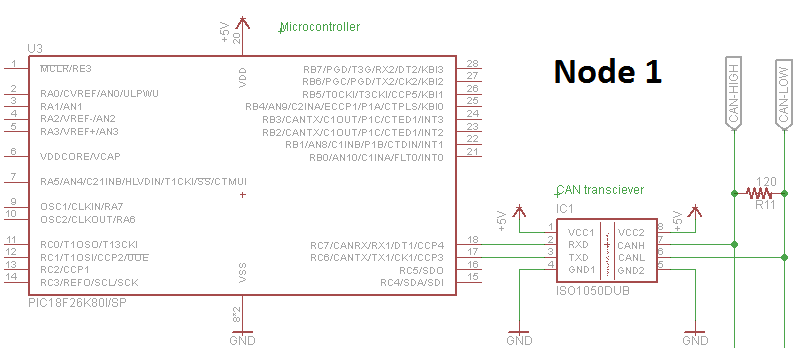
***You are now an expert in CAN bus theory!***

**CAN Bus Hardware general layout:**



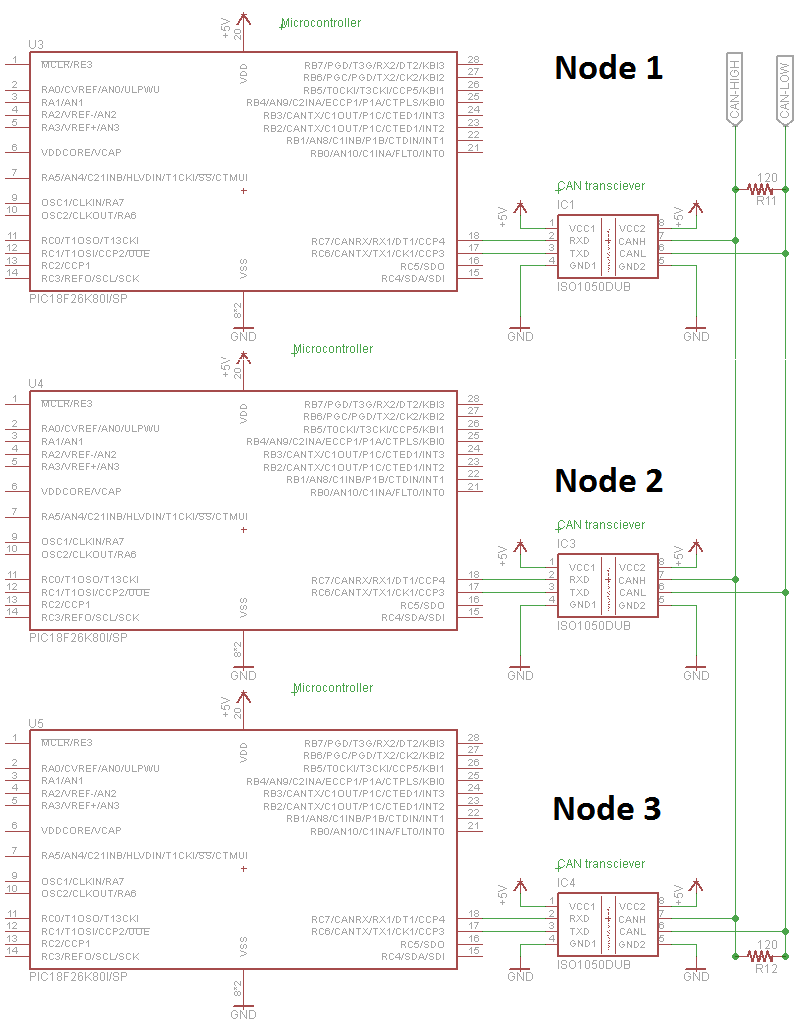
* Where SG1, SG2… SGn are the nodes attached on the Bus lines (CAN-HIGH and CAN-LOW)
* 120 Ohm termination resistors are used at the ends of the bus for minimizing reflections

**Hardware Design (EagleCAD schematic)**

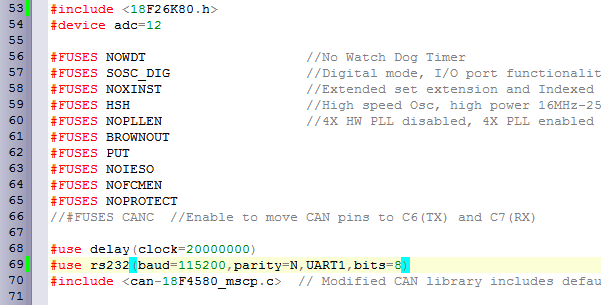


* This is the typical CAN bus hardware design we use in the solar car circuits
* We use the pic18f26k80 simply for reducing chip usage and space (the 26k80 has its own built in CAN interface, which directly connects to the CAN transceiver: Texas Instruments’ **ISO1050** isolated 5V CAN transceivers, and these are sample-able aka we get them for free)
* If you don’t use the microcontrollers that have built in CAN interfacing (for CAN bus required circuits), then you would have to use SPI to communicate to a separate CAN interface chip (**MCP2515** CAN interface chip), then the interface chip connects to the transceiver, which then connects to the Bus (CANH and CANL). Basically “*Microcontroller 🡪 MCP2515 🡪 ISO1050 🡪 Bus*” instead of “*Microcontroller 🡪 ISO1050 🡪 Bus*”. In total, you are wasting 4 pins on the microcontroller (SPI requires SDI, SDO, SCL, SS) and space for the CAN interface chip. Remember, surface area = money in circuit design world, so minimizing is ideal. So make sure you use the 26k80 MCU if your circuit requires CAN bus.
* For the actual bus itself, we use a CAT5-B type wire and two RJ-45 connectors on each board. Two connectors are required because all the boards are daisy-chained.
* Make sure to include a ¼ W 120 Ohm termination resistor for all the boards in the schematic and Board files. Do not solder the resistors on the boards. This will be decided once all the boards are installed into spitfire and the first/last boards are chosen based on minimizing cat5 wiring.
* The CAT5 wires will contain the following (in order): 12V(1), 5V(2), GND(3), CANH(1), CANL(1). The 12V will be accessed by the MPPTs and the Motor controller, the rest of the nodes will access the 5V line.

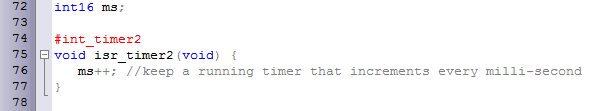
**CAN Bus system (simple example)**



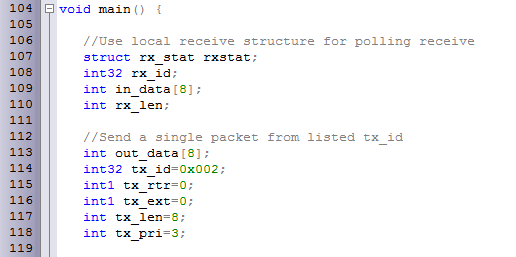
**CAN bus software implementation (accessible from n002.c in Dropbox\Battery Protection Share\CCS\n002.c)**



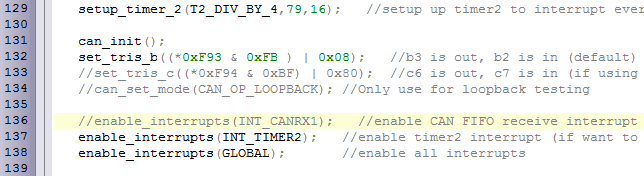
* In the beginning of the code, make sure to include the header files for the MCU you are using, for this example we are using the 18f26k80 microcontroller
* The #FUSES are already included when creating a new source file
* CAN bus requires high frequency operation, all CAN bus circuits require a minimum of a 20 MHz external clock, make sure to include it in the #use delay(clock=…)
* The <can-18F4580\_mscp.c> is a modified can library which contains timing settings that match the drivetek V3 MPPTs we are using in Spitfire



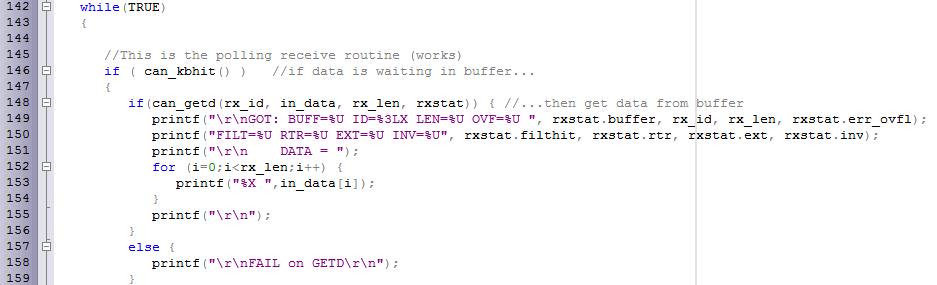
* This is a timer interrupt initialization for counting milliseconds



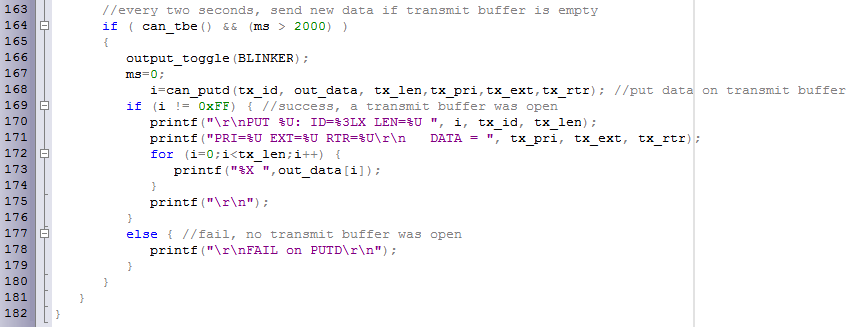
* Rx\_stat, id, data and length are all the frames which are received by this specific node from the CAN bus, hence why their values are not specified
* Out\_data = output data, each element of this array is a single byte for a total of 8 bytes
* Tx\_id is the transmitting ID, in other words this is the ID number of this specific node. The Lower the ID number, the higher priority for that specific node. So when programming your circuits, assign reasonable ID numbers to maintain message priority for boards such as motor controller, BPS and MPPTs



* Setup timer starts the timer interrupt which we had prepared to count milliseconds earlier
* Can\_init(); starts the CAN module
* Set\_tris\_b sets the CAN Rx and Tx for receiving and transmitting on the specified CAN pins of the microcontroller
* Then enable interupts



* Now during every cycle of the microcontroller, you want to check the BUS for information, this is done by can\_kbhit() which checks for data in the buffer
* If data is waiting in the buffer, then access it using the function can\_getd, and assign the data to the following fields: rx\_id, in\_data, rx\_len, rxstat
* In this example, the data is printed out to the screen, for the purpose of your circuits, you must check which node did this data come from by using a lookup table that compares the ID number to the ID numbers that you are interested in. for example: lets say Motor controller has an ID = 4, and you are coding the Driver display which needs to access the RPM, then you would check if the data received is from ID = 4, from there you would access in\_data and process the information to display it to the driver display circuit



* Can\_tbe() checks if the transmit buffer is empty
* This if statement checks if the timer interrupt has passed the 2 second mark AND if the transmit buffer is empty prior to proceeding
* If both transmit buffer is empty and timer > 2 seconds, then you can load data to the transmit buffer using can\_putd() with the input arguments representing the ID number, 8 bytes of data, length… etc
* The function can\_putd returns 0xFF during a fail, so !0xFF = returned success, for this example it prints out the transmitted frames during a success, for your circuits, you won’t need to print anything, but rather send valuable information (depending on what your circuit does)

I hope this has been a good walkthrough of setting up CAN bus. If anything is not clear, please ask me questions as I am happy to help, point out if any part is confusing and I will try to make it more straightforward.

Mhamad Salih