

CCB (Car Control Brains)

Principal Designer: Petersen

Telemetry Information:

<i>RDB Type</i>	'M' (Motor)
<i>RDB Address(es)</i>	32 (0x20)
<i>Known Packets</i>	A, B, C, E, H, I, J, L, M, Q, R, T
<i>Scheduled Packet(s)</i>	I, J

References:

Datasheets: 25C040, LTC1286, LM2672-5.0, MAX3100, MAX3224, MAX483E, MAXC001, MAXL001, PIC12C509, PIC16C73, VHELCS-100

Source Code: CCB.SRC

Description:

CCB is the main interface layer between the motor controller, driver, car systems, and telemetry. It was developed to take advantage of the features provided by the New Generation Motors EV-C200-092 motor controller used on the car. CCB in terms of a system encompasses several devices:

- Motor controller

- Driver controls (throttle, regen, brake switch, enable, direction)

- Main controller (Brains)

- Display controller (Display), *described in separate section*

- Driver pushbuttons (Switches), *described in separate section*

- Brake light and turn signal controller (BLATS), *described in separate section*

The motor controller is directly connected to the throttle and regen pots and the enable, direction, and throttle enable switches. It is also connected to the “brains” via the provided RS-232 serial link. All other driver controls and switches go directly to the “brains”. The display controller (see *Display*) and BLATS controller (see *BLATS*) also receive and send information via the “brains”.

Circuit Description

The “brains” are composed of several main circuit sections...

Main Controller and RDB: The first section consists of the main PIC16C73 and the RDB communication peripherals. The RDB portion is the basic RDB block consisting of an 8-pin RDB PIC and a MAX483E line driver. RDB communication is handled via the asynchronous serial port on the 16C73. The rest of the ports on the 16C73 are used to communicate with the other sub-units that comprise CCB. Serial communication to these devices takes place using the PICs SPI port (clock rate = $F_{osc}/16 = 625\text{kHz}$) while discrete signals are sent and received through the remaining port pins.

Motor Controller Communication: Communication with the motor controller is provided with a MAX3100 UART and a MAX3224 line driver. The PIC communicates with the UART via the SPI bus. This UART provides a convenient link and buffer between the PIC and the motor controller. The serial I/O lines of the UART pass through the line driver and get converted to/from the appropriate RS-232 levels. From here, they go directly to the motor controller.

A/D Converter: Input current to the motor controller is measured with an Engineered Components Company VHELCS-100 hall effect current sensor. The output from the sensor is digitized with an LTC1286 12-bit A/D converter. The ADC is connected to the SPI bus. Several levels of filtering and careful layout have been included in the design of this section since this ADC, being a sampling type converter, can be sensitive to noise. The SPI bus is also driven at a slower clock rate ($F_{osc}/64 = 156.25 \text{ kHz}$) during ADC communication due to frequency limitations of the ADC.

Odometer EEPROM: The odometer reading for the car is stored in a 25C040 E²PROM. The main PIC counts the pulses from the motor controller's SPD_PULSE line. The counter overflows every 256 counts and the number of overflows is stored as three bytes in the E²PROM. This value, therefore, is actually proportional to the number of wheel revolutions. However, assuming a constant tire circumference, this value is also proportional to the distance.

Horn: The car's horn can be controlled with a pushbutton by the driver or remotely through telemetry. The output is intended to be a relay coil that receives either +5V or +12V on one lead while the other lead gets grounded to turn it on. The pushbutton directly grounds the relay while the PIC drives a transistor that grounds the relay.

Power Supply: The power supply for CCB is a stock National Semiconductor Simple Switcher design based on the LM2672 step-down regulator. Some component values were modified to adapt it to the particular situation and the availability of parts.

Junction Panel: Since the "brains" act as a central hub for most of the control wiring on the car, it is designed to be a junction panel for these signals. By routing all signals into and out of the board, some degree of modularity and adaptability is maintained.

Theory of Operation

CCB is, at the basic level, a simple, query driven controller. There are two main sets of queries executed:

The first set of queries are those concerned with reading the switch (left/right turn, cruise on/off, brake) statuses and dealing with the various housekeeping tasks required. The switches are read in the :CHECK_URGENT program section. If further action needs to be taken due to the switches or for some other reason, the :URGENT section is executed. In this code, the event is decoded and the appropriate action(s) taken.

When a switch is depressed, the corresponding flag for dealing with it in the :URGENT section is set. The PIC waits to take the corresponding action again until it finds that the

switch has been released and depressed again. This prevents multiple occurrences which is important for the turn signal switches as they act in a toggle fashion (press-on, press-off).

The :URGENT code section is where most of the real actions are taken. It is executed whenever the priority flag (FLAG_PRIORITY) is set. When a turn signal switch is pressed or the brake is activated, the display controller is instructed to turn on the corresponding LEDs on the switches and set up the corresponding signals for the BLATS controller. Cruise control is also set up and stopped here when the appropriate switches are activated. If the appropriate flags are set in the timing routine (see below) the A/D converter is read, information packets are sent, received RDB packets are decoded, and/or the odometer E²PROM is written. In all cases, the display controller is updated with the most recent information.

The other set of queries involves communication with the motor controller. These queries are executed in a sequential, continuous loop. After each individual query execution, the previous set of switch and interface queries are run once to check for various events that may have occurred. The bulk of the motor controller queries deal with obtaining telemetry information from the motor controller, converting this information, and storing the information for later transmission.

Two queries (:QUERY_CURRENT_MATH and :QUERY_SEND_CURRENT), though, are involved in the actual driving of the car. These routines basically duplicate the math that would be done in the motor controller if it was in discrete control mode. But since the controller is placed in serial input mode to take advantage of its speed control loop for cruise control, this math must be done remotely. The throttle and regen pots are digitized by the motor controller, read back (:QUERY_AM_THR, :QUERY_AM_RGN), converted into a phase current following the equations provided by NGM (:QUERY_CURRENT_MATH), and sent back to the motor controller (:QUERY_SEND_CURRENT). When in cruise control mode, the :QUERY_THROTTLE_OVERRIDE routine checks for and deals with the case when the driver overrides cruise control temporarily with the throttle.

In addition to the queries, CCB also has a timing loop and a variety of math, conversion, and communication routines. The timing loop is interrupt driven from TIMER1. It provides an interrupt every 0.1 sec. This is then postscaled to provide a timer for the automatic transmission of 'I' and 'J' packets and the resetting of the performance metric counters. Within the postscaler routine is embedded a set of secondary timing routines. These routines control the horn and A/D conversion timing.

The math and conversion routines included in the code perform 16x16 bit multiplication (:MULTIPLY_16x16) and a variety of necessary conversions: ASCII hex to binary (:ASCIHEX_TO_BINARY), binary to decimal digits (:DECIMAL_TO_DIGITS), and ASCII decimal to binary (:ASCII_TO_DECIMAL).

The communication routines deal with SPI communication (:SPI_TX), sending and receiving serial motor controller data (:MC_SEND_CHAR, :MC_RECV_STRING), and sending and receiving RDB packets (:RDB_RX).

Several of the other features provided in CCB are...

Telemetry Information: Information about the motor controller, motor, brake lights, turn signals, cruise control, and the performance of CCB are available in one of two telemetry information packets: I & J. Other packets are available for controlling the various aspects of CCB and its operation.

Remote Control of Car Systems: The horn and turn signals can all be remotely controlled via telemetry. Each item has a specific packet format (see CCB Packet Descriptions).

Motor Controller Communication: In addition to the standard queries sent to the motor controller, custom commands may be sent via telemetry. The telemetry response packet contains the information returned by the motor controller.

Watchdog Reset: The watchdog reset is enabled in order to reset the system should it become locked. Upon reset by the watchdog timer, all initialization routines are performed except that the display only does a display test, skipping the “moving segments” routine. This was done so that CCB could more quickly return to its normal duties which is especially important should the reset occur while driving. The display test remains as a means to inform the driver that a reset has occurred.

History & Design Considerations

Design of CCB began during the summer of 1998. It was known at this point that we would be using the NGM-EV-C200-092 motor controller. When the controller manual was released, it became clear that a microcontroller interface was necessary to take full advantage of the controller and the data available and act as an interface between the controller, telemetry, and the driver. CCB, then known as **Car Control Bus**, was intended to be a fast, serial sub-network on the car responsible for communicating critical information between various devices on the car such as the motor controller, speedometer, brake lights, turn signals, etc. As development proceeded, it became clear that the device responsible for communicating with the motor controller was going to be the most involved of all the CCB devices due to the amount of information available and the dependence of most other devices on the status of the motor controller. For this reason, this device became the main controller for the CCB.

As for the serial link between CCB devices, it was decided to eliminate the complex serial protocol and just use discrete lines between devices. The complexity of developing another communication protocol combined with the increased possibility of noise interfering with the communication were two of the main reasons for this decision. Also, there was not a significant difference in the number of wires required for either plan. For example, the brake light/turn signal system currently in place (see *BLATS*) uses 6 lines (power, ground, 3 signal lines, signal return) between the car and the shell whereas a serial link would use at minimum three (power, ground, signal) but more likely four or five for either bi-directional communication or differential signaling. A similar situation was the case between the CCB controller and the speedometer, though here an SPI bus was also used in addition to the discrete lines carrying switch information since the chosen display controller, a MAX7221 (see *Display*) is an SPI compatible device. Also, the spatial separation between the main

controller and the various buttons and switches was not going to be very great and thus more wires was not a major problem.

At this point, the main CCB controller became the central hub for car-critical devices. Thus, the idea of a specialized bus was gone and hence the name change to **Car Control Brains**. As the central hub and communicator with the motor controller, it was decided to route all of the signals associated with running the car through CCB, including the various enable switches, direction switch, regen pot and throttle pot. These signals pass through directly to the motor controller where they are interpreted. CCB's function at this point was to read information from the motor controller and relay it to telemetry and the speedometer display. It was also responsible for reading and interpreting the turn signal pushbuttons and the brake switches on the brake and regen pedals and instructing the BLATS controller (via the Display controller) in the shell to light the appropriate lights on the car. CCB would also enable and disable cruise control at the request of the driver by communicating the appropriate commands to the motor controller. Other than this, CCB was passive in terms of driving the car—it only queried for information and did not normally play an active (software) role.

Several issues arose during the implementation of cruise control. The first dealt with how to implement cruise control. The original thought (before connecting the motor controller to a motor and experimenting with it) was to run with the controller in discrete torque mode—the controller reads the pots and calculates the desired torque—and then switch to serial speed mode when cruise control was to be enabled. To accomplish this, the motor controller must be put into program mode to change the appropriate E²PROM settings. The problem discovered was that to enter program mode the controller must be disabled *and* stopped—not very useful for cruise control! After talking with NGM and discussing several solution scenarios, it was decided to use the serial control mode exclusively but continue using the controller to read the pots. CCB would then read a scaled pot value, do the appropriate conversions to current, and then send this current value back to the controller. In this way, the controller may be easily switched between speed and torque modes. One initial fear was that the car is now dependent upon the operation of CCB for propulsion. On the other hand, cruise control would not have been possible aside from creating our own speed control loop in the PIC which still would have required the PIC to take an active role. Also, since the controller is still connected directly to the pots, it is a simple matter to change it back to discrete mode and resume driving. For these reasons, having the PIC take an active role was beneficial.

One problem discovered with this implementation of cruise control was that, if the controller was applying torque in torque control mode and was switched to speed control mode, the controller would accelerate the motor to full speed and hold, never correcting for the difference in set speed and actual speed! To eliminate this from happening, a coast command is sent before switching to speed control mode in order to force the current to zero.

During development and testing of CCB, several glitches were noticed with the circuit...

Capacitor Lock-up: The first problem was a tendency for the PIC to lock up when one of the input pushbuttons was pressed. This was traced to a capacitor from the signal line to ground, originally included so as to filter (RC filter with the pull-up resistor) and debounce the input

line. Since the software polls the inputs anyway, debouncing is not necessary and thus the capacitor was removed to eliminate the problem.

SPI & MAX3100 UART: Since the SPI port on the PIC16C73A does not include the provision for defining both the active clock transition edge *and* the inactive state of the clock, some devices, the MAX3100 in particular, did not initially work since the configuration used was not compatible. The fix involved simply sending a blank data byte after bringing the chip select line high so that the MAX3100 saw the appropriate pulses. The :SPI_TX routine was modified to account for this fix.

Page Boundaries: Since the software for CCB fills more than half the available memory in the PIC, jumps and calls between page boundaries must be dealt with correctly. This involves setting and clearing the PCLATH.3 bit before any jump or call to a different page.

Cruise Control: In the original implementation of cruise control, if the controller was applying torque in torque control mode (cruise off) and was switched to speed control mode (cruise on), the controller would accelerate the motor to full speed and hold, never correcting for the difference in set speed and actual speed. To eliminate this from happening, a coast command is sent before switching to speed control mode in order to force the current to zero.

Water Sensitivity of Power Supply: During one of the testing runs, water got onto the top panel of the car and on the original prototyping board for CCB. This moisture caused the power supply to shut down. The symptoms included an output voltage of approximately 2.5V that was not very stable. One theory is that the on/off pin (pin 5) of the LM2672 became “shorted” to the adjacent ground pin (pin 6), thus shutting down the device. A very weak “short” is all that would be needed since the internal pull-up on the pin is very weak. The problem was solved before determining the exact cause by drying the board and removing the solder flux. Although not conclusive proof of the cause of this incident, it was found that a damp paper towel placed between the on/off pin and ground did cause the device to shut down.

Spontaneous Display Blanking: Randomly, the speedometer display (see *Display*) would blank out, requiring a display or CCB reset. This was traced to EMI problems caused by the power trackers, usually when the trackers were operating at low power levels and/or in discontinuous mode. To help alleviate this problem, the Display is software reconfigured every second. Although not a complete fix, this does guarantee that the Display will come back without telemetry intervention.

Spontaneous Odometer Increment: Due to EMI problems, mostly associated with the power trackers, the odometer would increment when the car was not moving and even with the motor controller turned off. When the motor controller was on, the problem occurred almost exclusively during the precharge time, and not while the controller was fully powered through the motor breaker. When scoping the SPD_PULSE line in CCB, large voltage transients were discovered, corresponding to the power tracker’s switching frequency. These transients are apparently large enough to be read as a pulse on the input line. To help alleviate this problem, the overflows on the odometer counter in the PIC (TIMER0) are not counted toward the total odometer reading when the motor controller is not present. A similar situation was also encountered during strong accelerations and regenerative braking. In these cases, the previous fix did no good as the controller was powered and communicating during these periods of time.

Notes

The printed circuit board for CCB requires two fixes. The pull-up resistors for the CCB_IRQ and SPD_PULSE lines are missing and must be added to the bottom of the board.