An Active Brake Light Check Control

John Firestone

22 June 2008

ABSTRACT

Automotive brake light failures are a widespread nuisance and hazard, and brake lights can fail in non-obvious ways. The automakers have long realized this and offered systems to check the brake lights, but not all and not on all models.

This project adds a brake light check control to a car that did not come with one. The unit continuously monitors a set of incandescent brake lights, the brake light switch and the brake light fuse. Should any of these fail, it flashes an instrument cluster check light with a code identifying the failure. The unit actively tests the brake lights, even when off, which as a boon, preheats their filaments and increases the speed and reliability of the brake light system.

The check control was designed for vehicles with European-style, yellow, rear turn signals, but may be adaptable to American cars that use the rear turn signals as brake lights. At the flip of a switch, the unit can accommodate and test a high brake light flasher, a promising safety feature Mercedes is adding to their cars[1].

The heart of the check control is an Atmel ATtiny13 microcontroller running at 128 khz that is frugal enough to draw what little power it needs off the check light it is driving. A multiplexed i/o scheme protects the chip in a harsh, automotive environment, compensates the inputs for 12V supply variations and provides easy communication with the hardware – using a short list of function codes and no bit twiddling.

The check control's C-language software was made simple, to fit in 1 Kbyte of flash, and portable, to speed cross-development. A superloop in *main()* runs an input – check – output cycle roughly 90 times a second. Several simple but fruitful expedients prevent "lamp sing" and allow the software to work with limited signals and hardware.

THE READER may wish to print out pages 4, 5 and 11 for easy, later reference.

INTRODUCTION

Automotive brake light failures have been a problem ever since yellow brake lights were introduced in 1915[2]. A series of voluntary, road side checks across the United States in 2005 found 13% of the inspected vehicles had faults in one or more brake lights[3]. As a keen student of automotive failure has observed[4], having just one brake light out can be just as bad as having no brake lights at all: on many American cars they double as turn signals. A single, bad brake light is also sufficient cause for being stopped and controlled by the police. Catching brake light failures – before they become a hazard or a nuisance – is probably not such a bad idea.

The automakers have long recognized the frequency and risks of faulty brake lights, and thought of solutions, some of them quite ingenious. Ford, for example, first offered a fiber optic tail light monitor on the 1968 Thunderbird that transmitted light from the rear tail lights to indicator jewels mounted on the rear parcel shelf. The driver could check his tail lights in the rear

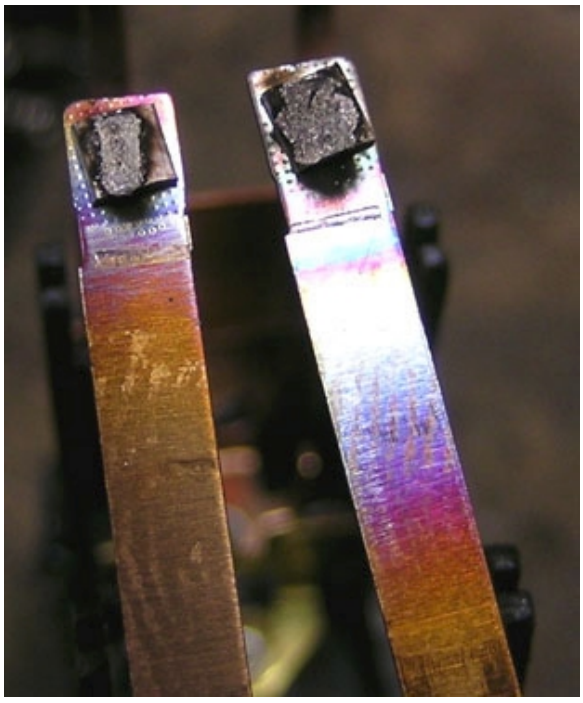

Fig. 1 *A well-burnt brake light switch (photograph by Randy Bernstein)*

view mirror and watch the car's sequential turn signals, which apparently was quite a treat[5]. More recently, the automakers have offered electrically-based systems that check the currents drawn by the individual brake light filaments.

Having bulbs burn out is not the only way a car's brake lights can degrade or fail. A set of cold, incandescent brake lights can have a combined inrush current of more than 50A. Such high currents stress all parts of the brake light system and cause sometimes, non-obvious failures. A search with Google for the phrase "brake light recall", yielded almost 100 000 hits worth, affecting almost every make, from the brake light switch on one end of the car to the tail light housings at the other. Figure 1 shows one such failure common to older BMW's (including the author's). Here, the repeated inrush surges have eroded away the more generously-sized, bifurcated contacts of an *improved* switch, and caused it to fail.

Many such failures could be prevented by converting a car to LED brake lights, but often that is not practical or legal.

Despite its appeal and usefulness, most cars do not have a system that checks the brake lights, often because it was not offered. This project adds one that continuously monitors the brake light fuse, switch and lights, and flashes a warning should any of them fail. Unlike many older designs, this one actively monitors the brake lights, which, as a boon, increases their life, speed and reliability.

HARDWARE

The Brake Light Check Control system (BLCC) consists of a small module placed between the brake light switch and brake lights, and an incandescent, instrument cluster check light (Fig. 2). The module has two independent sections:

- **a brake light decoder/driver** that draws power from the brake light fuse and drives the brake lights, as directed by the brake light switch and the check control, and

- **a check control section** that monitors the brake light power, switch, lights and decoder/driver, and lights the check light should any of these fail.

The module can service a traditional, single-section, high-current brake light switch (driving just the BRAKE_SW signal in Figure 2) or a two section switch with pairs of high- and low-current contacts (driving both the BRAKE_SW and TEST_SW signals). Some cars have and use the second pair to disable a cruise control or to test the switch. The decoder/driver uses it (if present) to increase the brake light system's reliability. The switch in the author's test vehicle has two pairs of bifurcated contacts which act as a *quadrifurcated* brake light switch when connected to the check control system.

The decoder/driver can light all three brake lights when the switch closes (high, left and right) or it can drive just the lower two and allow an optional controller (shown dashed) to light and flash the high brake light – and confirm that it does.

The module's two sections are independently powered and resistively coupled so that a power failure in one does not take out the other. For greater reliability, the check control section draws what little power it needs from the instrument cluster check lamp it is driving. As long as the cluster has power and the lamp is intact, the section should continue to function and warn about any failures.

Figure 3 is a schematic of the check control module. The dotted, horizontal line splits the upper decoder/driver section from the lower check control section. The line passes through resistors RN4A and RN4B which couple one section to the other.

Pull up resistor R7 and pull down resistor R13, at the middle left edge of the page, convert brake light switch closures to active low and high, test and brake switch signals. R13 draws a nominal, 12 ma "wetting current" through the high current switch section to clean away any contact oxidation. Resistor network RN5, to its right, and resistors R8 and R6, below, pass the signals to the two sections, and isolate one section from the other.

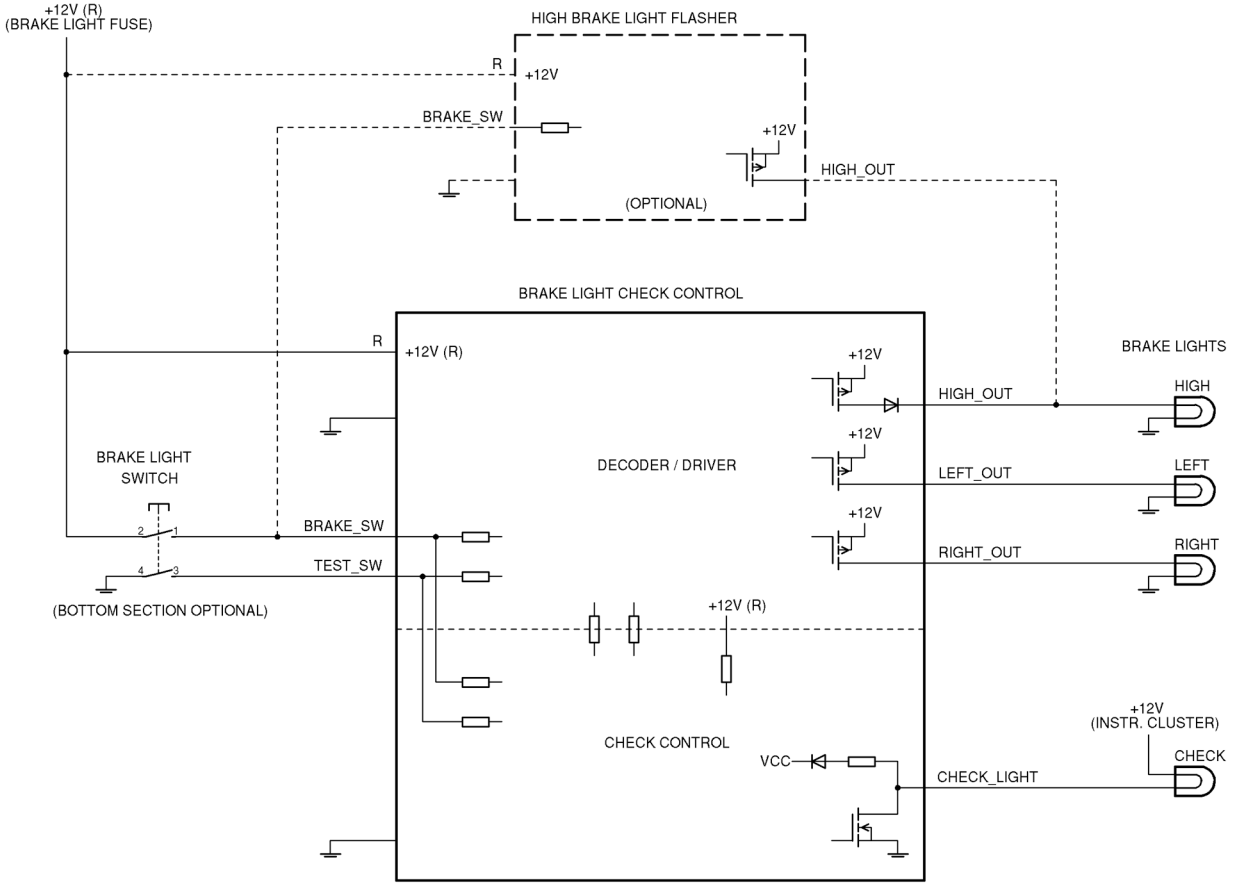

Fig. 2 *Block diagram of the check control module and brake light system*

THE BRAKE LIGHT DECODER/DRIVER

RN5 divides and limits the 12V brake and test switch signals to roughly 0–5V, for input to data selector U6. U6's 1Y and 3Y outputs go high when either signal is active which turns on high side drivers U1–U3 and the brake lights. Closing switch SW1 disables output 3Y and the output to the high brake light so that an external controller may flash it. Output 4Y goes high whenever U6 drives the high brake light, and communicates the switch setting, through RN4B, to the lower check control section.

The lower check control raises U6's G-strobe input, through RN4A, when it wishes to light the brake lights and measure their filament currents. This disables U6's active low outputs which go high and turn on U1–U3 and all three brake lights – even when SW1 is closed.

U6 is a piece of good, old, MSI ur-logic which might seem out of place in an age of low-cost, programmable parts. U6, however, does the job and is frugal, fast, robust, "uncrashable" and 100% bug free. The latter are good qualities for a safety-related system.

Resistors RN4C and R2, to the right of U6, protect it against transients from U1–U3.

Brake light drivers U1–U3 are standard, automotive, high side drivers that are protected against short circuits, overheating, spikes, surges, reverse battery connections and a few other electrical hazards. The automakers buy such drivers in huge numbers and the chipmakers churn them out like water. Consequently, they are impressively cheap for all that they do. These particular parts are a little quirky and have been applied following the manufacturer's suggestions. (Deviating from them didn't help.)

U3 drives the high brake light through schottky rectifier diode D1 so that the check control can monitor an external brake light flasher (*cf.* top of Fig. 2). The diode introduces a small voltage drop which greatly reduces U3's output current when an external flasher drives the high brake light.

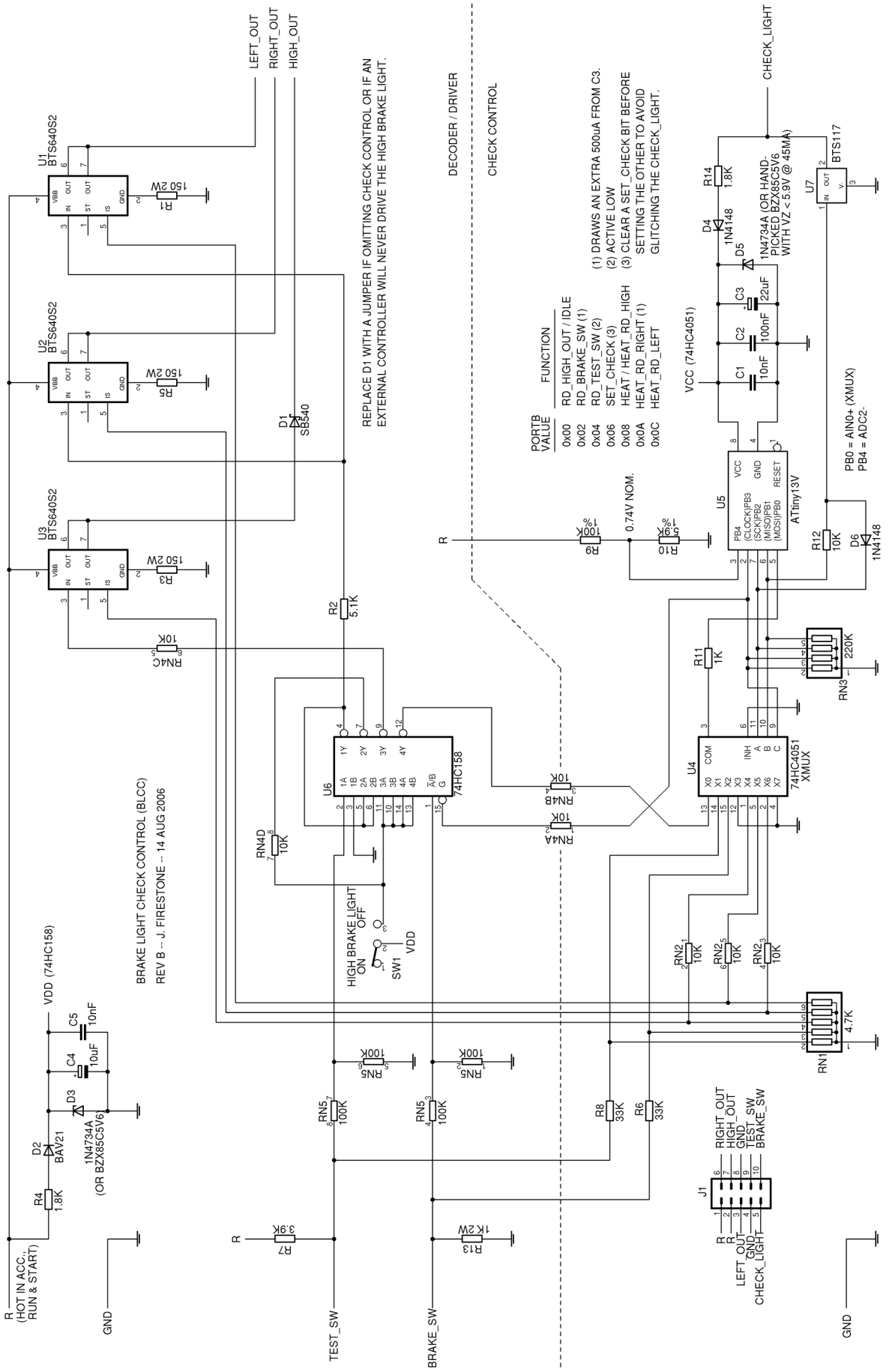

Fig. 3 Schematic of the Brake Light Check Control (BLCC) module

Each high side driver returns a small sense current, IS, on pin 5 that is proportional to the current into its load. The three sense currents from U1–U3 pass leftward and downward to the lower check control section which uses them to detect faulty brake lights, and a faulty flasher.

Some of the surges and transients within a car's electrical system can be quite fierce. Each automaker publishes an exacting list that the car's electronics should survive[6]. The International Standards Organization (ISO) and American Society of Automotive Engineers (SAE) have amalgamated these lists into conducted immunity standards[7,8] that have become increasingly popular.

Brake light drivers U1–U3 satisfy nearly all parts of test level III of the ISO 7637 standard, a high and quite demanding one, especially for aftermarket electronics. It requires, among other things, that it survive repeated, brief spikes of up to ±90V and a one-time 80V, 280 ms surge. The latter simulates a "load dump" which happens when the battery is disconnected with the engine and generator turning at high speed. Such a surge has a great deal of energy and is beyond what the drivers – or most of the electronics in a car – can handle. Fortunately, nearly all the automakers place a large, central suppressor across the generator outputs that clamps the surge to 40V or less and dissipates most of the energy. (U1–U3 are rated to at least 60V.)

The CMOS decoder/driver logic uses very little current. The simple and inexpensive, zener diode, shunt regulator in the upper left corner supplies U6 with filtered, 5V power. Zener diode D3 also suppresses any automotive surges and transients (to level III and beyond) that may enter the decoder/driver section through its supply *or* i/o lines. The signal polarities of SW1 and outputs 2Y and 4Y are arranged to reduce the section's worst case current drain and, thus, the maximum voltage drop from current limiting resistor R4.

THE CHECK CONTROL SECTION

The Atmel ATtiny13V microcontroller in the lower middle-right of Figure 3 serves as the heart of the check control. It monitors the decoder/driver inputs and outputs, pulses the brake lights to check them, and lights the check light when it detects a fault. The rest of the section is designed for and revolves around it – which should please its maker. The section's design tries to exploit its strengths while respecting its limits.

The 'tiny13 is "fused" to run off the same 128 khz oscillator that clocks its watchdog timer. This reduces its power consumption to less than 1 mW and allows it to run off its load, the check light. The check light uses an extra-bright, 2W incandescent bulb with one side tied to 12V (*cf.* lower right corner of Fig. 2). The 'tiny13 grounds the other side to light it (via low-side driver U7, lower right corner of Fig. 3), but only 75% of the time. This dims it to match the other, 1.2W instrument cluster bulbs. The 'tiny13 draws power to keep running while the bulb is off, the other 25% of the time, through the simple shunt regulator to its right. Reserve capacitor, C3 supplies power while the check light is on and smooths over brief, power outages that might otherwise reset the CPU. The 'tiny13's brownout level is fused at 1.8V to tolerate outages lasting hundreds of milliseconds.

The check control has to monitor and control some nine different inputs and outputs. Thus, it may seem odd that it uses a chip with, at most, six usable i/o pins. The check control uses a few other parts rather than a larger AVR because of the automotive surges and transients discussed earlier. Atmel recommend no more than a milliamp into an i/o pin, but allow higher, short term currents[9]. They do not characterize how much higher, however[10].

Rather than bet the design on a guessed, maximum, short term value, the check control adds the robust, CMOS multiplexer, U4, in front of the 'tiny13 and has U4 absorb the bulk of any surges and transients. The 'tiny13 outputs one of the listed PORTB function values to read any of six different signals that U4 selects and routes to pin PB0. Intermediate resistor R11 insures that the multiplexer takes the brunt of any overvoltage and PB0 less than a milliamp, even if the latter's ESD protection diodes conduct first.

The 'tiny13 can have U4 select:

- the decoder/driver logic, U6 (via RN4B), to see if U6 is driving the high brake light,
- the brake lights switches (via R8 and R6), to see if they are closed, and

- the brake light drivers, U1–U3 (via RN2), to see if they or a high brake light flasher are driving their brake lights.

The resistors protect U4 against transients that are beyond the severe requirements of ISO-7637, Level IV. The 'tiny13 indirectly software-filters the selected signals to eliminate parts.

It was not clear at the start just how many instructions it would need to read the signals. Indeed, the author ran out of both cycles and flash a couple times while developing its software. To consume less of both, the 'tiny13 uses its internal analog comparator to compare the selected input on pin PB0 against the divided brake light circuit voltage on PB4. The comparator signals an input is active when its attenuated signal on PB0 exceeds the nominal 0.74V on PB4.

RN1, at the lower left, sets or reduces the signals so that 0.74V appears on PB0 when they are at roughly half their normal, active levels. The voltage across RN1 should drop below 0.74V and a lamp circuit declared inactive when the corresponding load sense current returned by U1–U3 is less than 1.07A. At that current, a 32 MSCD brake light is about as bright as a fully-on, 3 MSCD tail light.

The threshold voltage on PB4 tracks the brake light supply voltage and drops, for example, while the engine is cranking. This automatically adjusts the active/inactive threshold for the switch inputs and keeps it at roughly half the supply voltage as the voltage changes. It somewhat overcompensates the threshold for the brake light load sense signals, however, since incandescent lamp currents vary sublinearly with the voltage. Since the threshold voltage adjusts all the way down to 0V, the 'tiny13 measures and checks the brake light voltage on PB4 with its A/D converter (ADC). Fortuitously, the nominal 0.74V on PB4 is a good fraction of the ADC's, 1.1V internal bandgap reference; comparing PB4 against PB0 also selects PB4 for the ADC.

Selecting any of the brake light drivers takes PB3 high. This turns on all three drivers and brake lights, through RN4A and U6, and lets the 'tiny13 check them even when the operator is not braking. R12 and D6 (below the 'tiny13) turn on U7 and the check light whenever the value 0x06 is output to PORTB. This "Mickey-Mouse" output decoding saves a little money and an i/o pin, but may require extra, intermediate output values to avoid glitching the check light.

Only occasionally, however. In return, R12, D6 and U4 create a function-code-driven, i/o interface that provides simple communication with the hardware. All output funnels through a single port, PORTB, and all input appears in a single, register bit, bit ACO from the analog comparator. The check control software can manipulate the outputs and test the inputs with just a few simple macros and a short list of i/o function values.

By contrast, substituting a larger AVR chip and connecting it more directly to the inputs would require reading, writing and bit twiddling over several different i/o ports. It would also require much higher value, input resistors (100K vs. 10K) to respect the maker's recommendations and protect the chip against surges and transients. The tenfold greater circuit impedance might significantly reduce the noise immunity of the brake light load sense signals when they are close to the 0.74V active/inactive threshold.

Finally, RN3 to the left of R12 and D6, pulls down and idles PORTB during power up and reset.

PRINTED CIRCUIT BOARD

The two check control prototypes were built with double-sided circuit boards and well-established, through-hole parts. The older technology did not significantly increase the project's total cost and makes the boards serviceable with inexpensive (non-SMT) equipment.

The circuit board in Figure 4 is a second spin using fewer parts inside the same form factor. Thus, it is generously sized. The board is mounted vertically and oriented as it is in the car, with the signal and power connector off to the right and the BLCC acronym across the top. The components are oriented and thermally zoned for bottom to top, convective cooling. The heat generat-

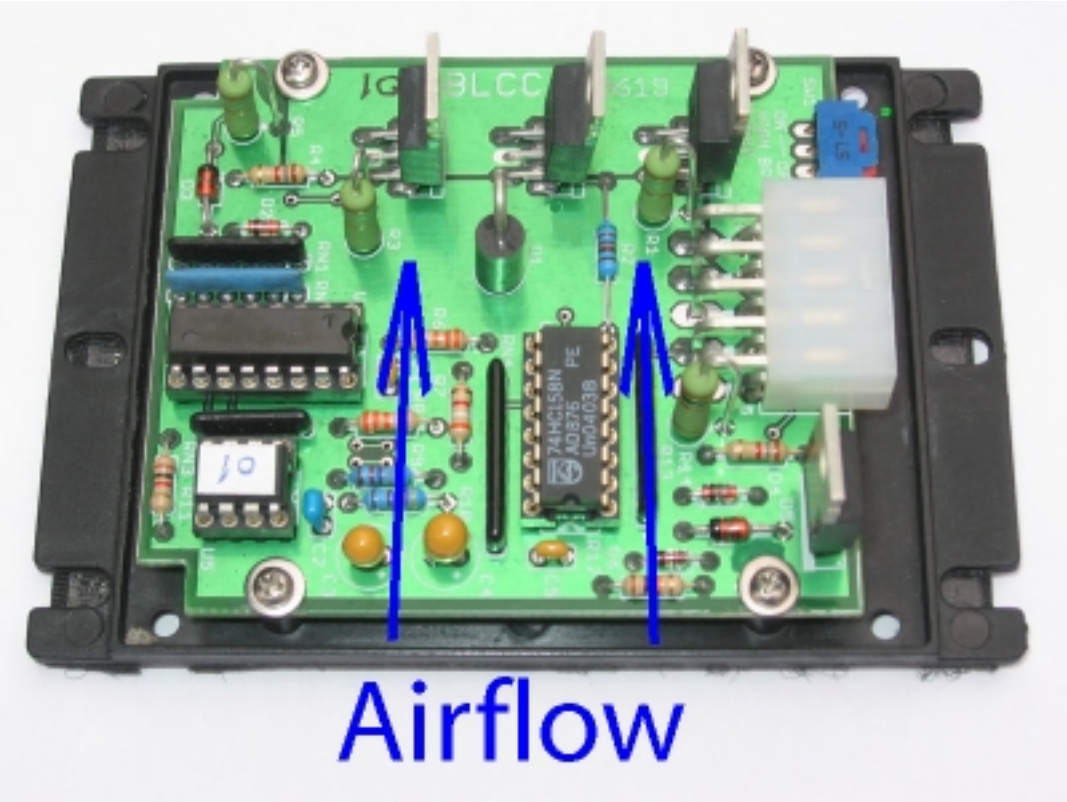

Fig. 4 The air flow over the printed circuit board

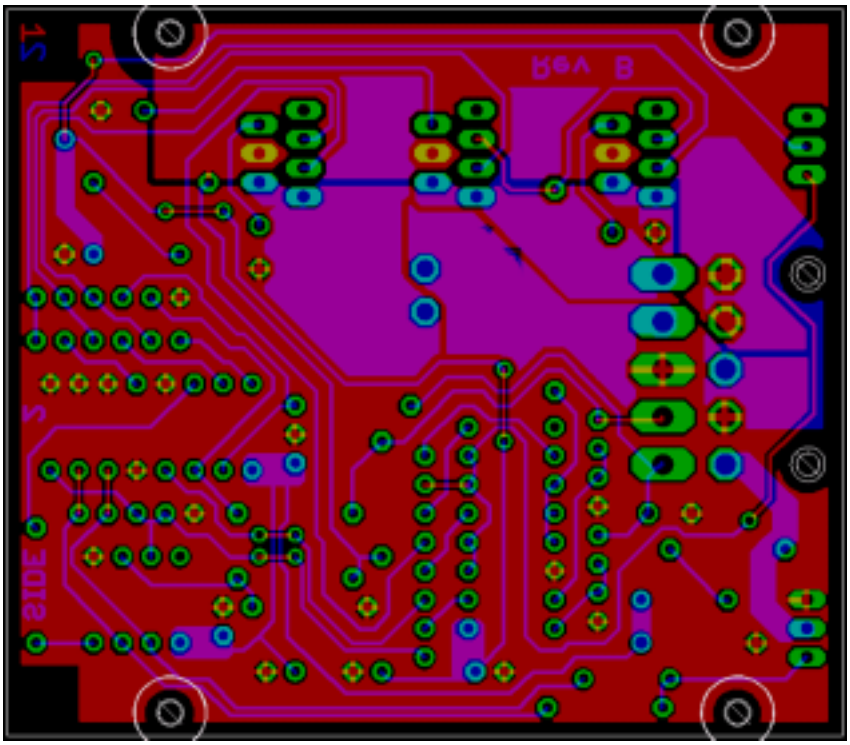

Fig. 5 The printed circuit board layers

ing parts are largely along the top; the two most heat sensitive components, C3 and C4, are at the bottom. The heat generators are mounted on end, for maximum cooling, and horizontally protrude into the clear air next to the vertical circuit board.

The prototypes used a few parts the author had on hand, including some that are more than sufficient for the job. Consequently, none should experience more than a 10–20 ° C temperature rise under normal conditions.

The heat generating parts could have been placed and packed much more casually. The circuit board's thermal design, however, should allow more aggressively rated, hotter running parts. The prototypes' 2W power resistors, for example, could be shaved to 1W , to cut costs.

Figure 5 shows the board's top and bottom layers. The red, top layer is largely used as a ground plane, with a minimum of inner cuts. The layer's top quarter carries 12V power to the three high side, brake light drivers. The power plane and traces are quite wide, as part of a 15A fused circuit. While they should never *have to* carry a current that blows the fuse, nevertheless, they should be *able to* in case of a short. Likewise, not one but two connector pins bring 12V power to the plane, through separate, parallel wires from the fuse: to split the current between the pins and increase their short term, overload capacity.

The output traces from the three high side drivers, in the upper half of the blue, bottom layer, were made as wide as possible even though they should be protected by the driver's built-in current limiting. The limiting should allow lighter wiring and make downstream fuses redundant. After some controller fires using similarly parts, however, that may not be entirely true.

The wide, blue traces to schottky diode D1 (the large pads just above the middle) also serve as heat spreaders.

OPERATION & USER INTERFACE

The check control begins testing the brake light system as soon as the driver turns on the ignition. If his foot is on the brake as he does and the brake lights are in order, the check control turns on the instrument cluster check light to confirm everything is working – including the light and the brake light switch. Thereafter, the check control continuously tests the system and reports any faults.

Up to 90 times a second, it samples the brake light circuit voltage, the brake light switch, the decoder/driver and the individual brake lights, and checks for discrepancies. If any appear and persist, it turns on and flashes the check light to announce and repeatedly identify the fault(s).

For less serious faults (Table 1), it turns on the check light, pauses and briefly dims it a certain number of times to identify the fault. It repeatedly pauses and winks out the faults for roughly two minutes and then turns off the check light so as to not annoy the driver at night. If another less serious fault appears, it turns on the light and winks out the new fault(s) for two more minutes, along with the earlier faults, as a helpful reminder.

The check control flashes the check light – fully on and off – 15 times if it detects a serious fault, one that suggests the car has no effective brake lights. It then winks out any Table 1 faults that might apply, for two more minutes, to help pinpoint the problem. After 15 flashes and perhaps 2 minutes of winking, the check control dims the check light and leaves it on until the driver switches off the ignition – again, to not annoy him at night, and to remind him that he can not count on his brake lights.

A warning device must walk a fine line between being overlooked and becoming a nuisance.

For a faulty:	Dim / wink the check light:
left brake light	1 time
high brake light	2 times
right brake light	3 times
brake light switch	4 times
output decoder/driver	5 times
external flasher	6 times

Table 1 *The check light flash codes*

The ATtiny13 runs the C-language check control program a nominal 90 times per second. This is often enough to avoid lamp flicker yet infrequent enough to allow a comfortable number of instructions per update cycle. Within each 90 hz update cycle, the program reads the module's inputs and outputs, checks them for faults and pulses the instrument cluster check light. To save power, the 1 Kbyte 'tiny13 runs off its 128 khz watchdog oscillator and idles when it has nothing to do.

The 'tiny13 can execute about 1400 instructions per update cycle and has fewer than 500 instructions in flash to do its job. This is not much. The software was made conceptually and functionally simple to stay within these limits, and to allow cross-development. This proved to be a good decision and made writing and testing the software a fast, agreeable and almost hassle-free project.

The software was kept portable and isolates target-specific definitions and code in two files, *target.c* and *target.h*. The software was first written and debugged on PowerPC computers running Mac OSX, with simple command line programs replacing the hardware. The hardware-driver code was then substituted and the result cross-compiled with *avr-gcc 3.0.3* and *3.4.3*, under both OSX and Windows XP (as a test). The final binary was then downloaded with *avrdud*e and tested in the actual hardware.

THE 90 HZ UPDATE CYCLE

A superloop scheduler in *main()* divides the 90 hz update cycle into six, 12.5%, single timeslices and a seventh, 25%, double timeslice. Figure 6 shows the tasks and principle data flow over a cycle. *Main()* calls one or two functions at the start of each timeslice, which return after consuming most of their timeslice.

Main() starts an update cycle (at the top left) by calling the *test_flash()* and *init_hardware()* functions. These test the program flash and configure the ATtiny13, respectively. On its very first call, *test_flash()* performs a Cyclic Redundancy Check on the entire flash and returns if it is correct (*i.e.* zero). If it isn't, it spin blocks until the watchdog times out and resets the controller. On subsequent calls, *test_flash()* slowly recalculates the flash's CRC value, dividing one flash byte per call, so that it continuously verifies the program code every 512 calls. (It is called again, later in the update cycle.)

Once per call, *init_hardware()* also reconfigures the 'tiny13 in case some spurious event clobbered a control register. The common Windows™ solution, cycling power to restart and restore, is not really an option, especially while the vehicle is moving.

Main() calls *sample()* and actually starts a new input – check – output cycle at 75% into the update cycle. Skipping to there, *sample()* triggers the A/D converter and reads and writes i/o PORTB: to measure the brake light voltage, sample the decoder/driver and start sampling the brake lights.

The brake light drivers, U1–U3, are really quite slow; *sample()* may have to pulse them for over 300 µs before they fully turn on and return stable filament currents. *Sample()* waits until the second half of the double timeslice: to have more time and to pulse the drivers and lights as quickly as possible.

At 87.5% into the cycle, *sample()* pulses the brake light drivers and checks their filament sense currents. It checks them twice: before the drivers have had a chance to respond, to see if the decoder/driver has already lit the brake lights, and after, to confirm the brake lights took its sampling pulse. This allows *sample()* to perform two checks with a single PORTB function value and saves a microcontroller i/o pin. *Sample()'s* long pulse also heats the brake light filaments and makes them glow very slightly. This preheating cuts their cold, inrush current by more than a factor of three, helps them light about 50 ms sooner and increases their reliability and resistance to shock.

Sample() returns the unfiltered switch, light and decoder/driver state in *io_state*, and *main()* copies the measured brake light voltage into *bl_voltage*. Although tempting, *main()* does not call a function to filter these raw values: that would be redundant. After *check()* (up next) infers faults from them, the result must be filtered,

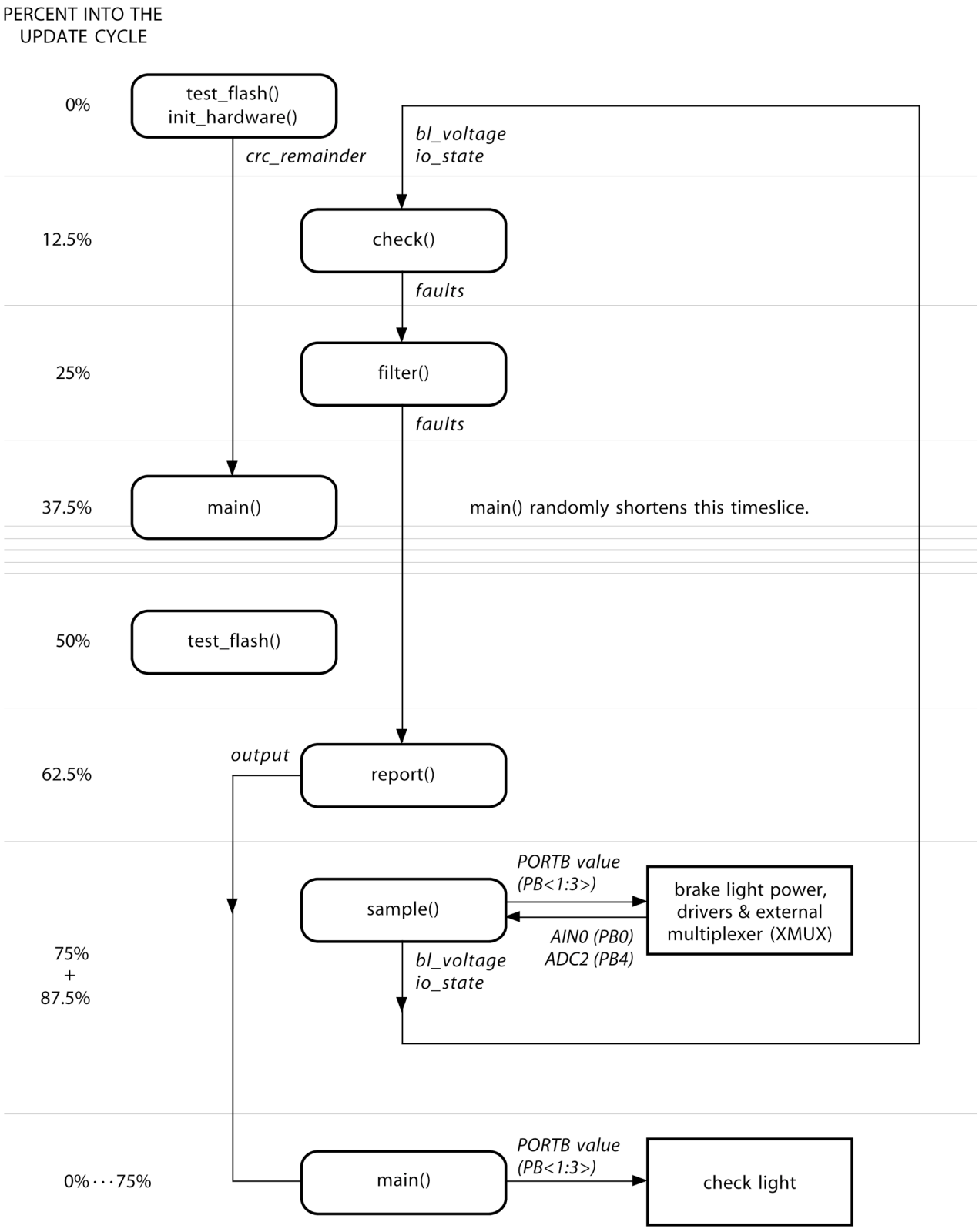

Fig. 6 The tasks and principle data flow over an update cycle

anyway, to ignore specious faults caused by input timing skew. The program might as well filter just once, and filter the result rather the input. The result is what counts.

At 12.5% into the next update cycle, *main()* calls the *check()* function with the *io_state* and the brake light voltage, *bl_voltage*. *Check()* examines both and returns a set of tentative *faults*. These may be no more than switching transients, so at 25% into the cycle, *main()* calls the *filter()* function to time-filter them: over 32 update cycles, if a fault is absent, or 64 cycles, if it is present.

The external brake light flasher in Figure 2 can either steadily drive the high brake light or rapidly flash it. As *check()* can't know which the flasher is doing (that would need another input), it assumes the flasher drives it steadily. To *check()*, that makes the flasher appear faulty, during its off cycle, whenever it is flashing. *Filter()* swallows this apparent complication through the above expedient: it time-filters the *faults* over a couple flash periods and counts the absence of a fault twice as quickly as its appearance, over 32 cycles rather than 64.

The 37.5% and 50% timeslices will be explained shortly.

At 62.5% into the update cycle, *main()* calls the *report()* function with the now, time-filtered faults. *Report()* notes any new ones and returns as *output* – over many successive calls and update cycles – a stream of PORTB function values that flash the check light and report the faults.

All the check control input and output flows through PORTB. *Main()* takes charge of the i/o port a couple timeslices later, at the start of the next update cycle (0% in). It writes out the latest *output* value, which may or may not light the check light, depending on what *report()* returned. At 50% into the cycle, *main()* idles the outputs and extinguishes the light if *report()* returned that it should be only dimly on.

At 75% into the cycle, *main()* idles the outputs. This extinguishes the check light if it was on, dims it to match the other indicator lights and allows the check control to draw power through the bulb to continue running. After idling PORTB, *main()* relinquishes it to the *sample()* function, so that it can access the controller's inputs and outputs for the rest of the update cycle.

Main() very briefly pulses the check light at 0 and 50% into the update cycle whenever the check light would otherwise be off, to preheat its filament and increase its reliability.

The *sample()* function briefly pulses the brake lights so that it can detect their filaments and tell if one has gone out. The brief pulses can excite resonances and make the filaments buzz. To reduce such "lamp sing", *init_hardware()* programs each timeslice to take slightly too long. Using the stream of CRC remainder values calculated by *test_flash()*, *main()* randomly shortens the fourth timeslice that starts 37.5% into the update cycle. This jitters the start of *sample()'s* timeslice, and thus the start of its pulses, and reduces the brake lights' lamp sing. *Main()* runs the *test_flash()* function a second time, at 50% into the update cycle, to make the stream of *crc_remainder* values more random.

"BUGS" & THE SOURCE CODE

The reader may consult the commented, C-language source code for the complete details of the check control software.

The reader should not be alarmed to see the words BUG or BUGS while reading the source. They really aren't. The author follows the UNIX tradition of prominently declaring as BUGS, the limitations or assumptions made in a piece of code to avoid some constraint. These should not be actual bugs, or even inconsequential bugs, rather, things that *could* become bugs and cause grief if a later change violates a limitation or assumption. To guard against that, the source code checks many of its assumptions and limits at compile time using the *CONFIRM()* macro defined in the file *common.h*.

While he tries to avoid them, the author had to resort to the very occasional hack for extreme want of free cycles or words. All such instances should be plainly marked HACK and checked with *CONFIRM()* statements.

CLOCK MEASUREMENT & FIX

ATtiny13V	Watchdog oscillator frequency
01	113 000 hz
02	123 500 hz
03	121 900 hz
04	124 700 hz
05	113 900 hz

Table 2 *The calculated watchdog oscillator frequencies for five ATtiny13 microcontrollers*

The ATtiny13's 128 khz, watchdog oscillator is about as stable as its internal, calibrated oscillator[9], but is uncalibrated. Its frequency needs to be measured before it can serve as the system clock: to fix the number of system clock cycles per timeslice. Each of the two check control modules was programmed assuming a 128 khz clock, connected just to power through a check light bulb, and allowed to flash and finally pulse width modulate it with a 50% duty cycle. The resulting pulses would repeat at 90 hz, once per update cycle, if the watchdog oscillator ran at 128 khz.

A scopemeter was used to measure the actual PWM frequency in the supply ripple across diode D5 (lower right hand corner of Fig. 3). Here, D5's somewhat weak zener knee helpfully strengthens the signal. The actual oscillator frequency was then calculated from the ratio to 90 hz.

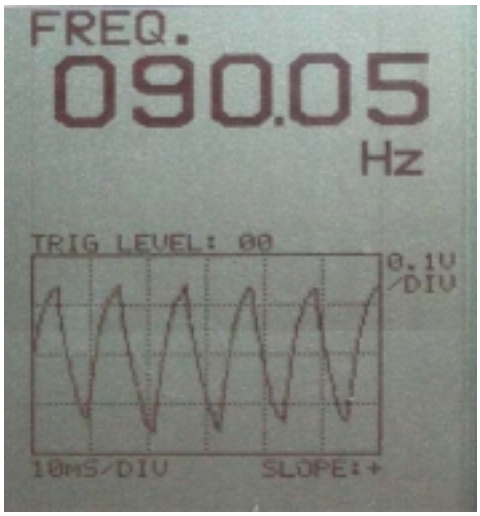

Fig. 7 *The second controller's output and update cycle frequency, after a clock fix*

Table 2 gives the frequencies determined for five different ATtiny13's. Each controller was reprogrammed with its oscillator's frequency and then checked to see that its output (and update cycle) repeat at 90 hz. Figure 7 shows the final result for the second controller, as seen in the ripple across D5.

INSTALLATION

Figure 8 shows the first installation of the brake light check control in a BMW 3-series. The unit is attached to the right rear wall with Velcro™ strips for easy access and removal without tools. According to their datasheets, the strips should hold three times the controller's mass in a 100g forward crash, after 1000 mate and unmate cycles.

The controller plugs into most of the car's brake light signals through a T-adapter inserted between the tail light assembly and its plug. In case of a serious malfunction, the adapter can be unplugged and the car's original brake lights restored.

A second controller is now being installed and tested in a Daimler/Chrysler Dodge Caravan with independent brake and turn signal circuits.

ACKNOWLEDGMENTS

Randy Bernstein kindly provided the image for Figure 1 which is reproduced with his permission. James Karaganis made helpful suggestions that improved the user interface. He also heroically volunteered his Dodge minivan to receive and test the second controller. Arne Greindl and Andrew Owen of Bobrink GmbH (Bremerhaven, Germany) help find the missing part for the T-adapter cable and provided further encouragement during the check control's development.

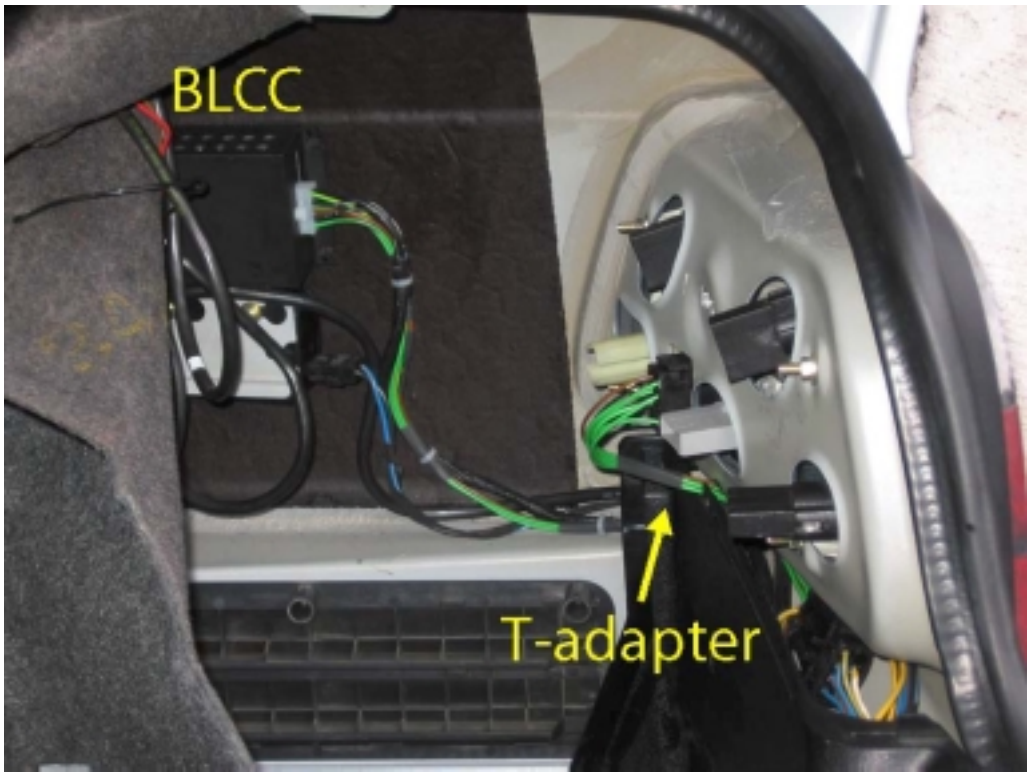

Fig. 8 *The first installation in a BMW 3-series*

REFERENCES

(1) "Mercedes-Benz to adopt flashing brake lights", Canadian Driver, 1 March 2005, http://www.canadiandriver.com/news/050301-1.htm

(2) "Exploring the Design and Implementation of Networked Vehicular Systems", Liviu Iftode, Cristian Borcea, Nishkam Ravi, and Tamer Nadeem, Rutgers Univ., Dept. of Computer Science Technical Report, DCS-TR-585.

(3) http://www.carcare.org/NCCM/National_Car_Care_Month.shtml.

(4) "What's all this Dead Car Stuff, Anyway", Electronic Design, 6 February 1992.

(5) http://automotivemileposts.com/thunderbird50.html.

(6) For example, General Motor's Worldwide Engineering Standard GMW3097, "General Specification for Electrical for Electrical/Electronic Components and Subsystems, Electromagnetic Compatibility (EMC)".

(7) International Standards Organization, ISO 7637-1 (2002), "Road vehicles – Electrical disturbances from conduction and coupling – Part 1: Definitions and general considerations"; ISO 7367-2 (2002), "Road vehicles – Electrical disturbances from conduction and coupling – Part 2: Electrical transient conduction along supply lines only"; ISO 7367-3 (1995) "Road vehicles – Electrical disturbance by conduction and coupling – Part 3: Vehicles with nominal 12 V or 24 V supply voltage – Electrical transient transmission by capacitive and inductive coupling via lines other than supply lines".

(8) Society of Automotive Engineers, J1113/1_199507, "Electromagnetic Compatibility Measurement Procedures and Limits for Vehicle Components (except aircraft) 60 Hz to 18 GHz"; J1113/2_199609: "Electromagnetic Compatibility Measurement Procedure for Vehicle Components (except aircraft) Conducted Immunity, 30 Hz to 250 kHz".

(9) Atmel application note, AVR182: "Zero Cross Detector".

(10) Atmel datasheet, 2535E-AVR-10/04: "ATtiny13 Preliminary Complete".