DATALOGGER (THE BLACK BOX) OVERVIEW

"THE RANTINGS OF A DUCK"

Summary

Datalogger does exactly what the name implies - it logs data, much like aircraft black boxes. More specifically, it reads CAN frames from the car CANbus and stores it to a SD Card for later analysis. This is primarily intended to be used as a debugging tool.
There is an Appendix at the end which defines some terms used.

Hardware Overview

A microcontroller serves as the "brain" of the board - it interfaces with the CAN network and the SD Card. There is also other supporting circuitry performing a variety of functions.

Microcontroller (P8X32A or PIC24HJ128GP502/dsPIC33FJ128MC802)

Currently, the board support both Propeller and PIC microcontrollers, however, only one is used at a time. Both use an external crystal for clocking.
Active development is done on the PIC since programming can be done in portable, industry-standard C.
A standard BRAIN was considered, however, initial testing with the Roland-Riegel shows it did not have the sufficient performance. Additionally, lack of two SPI lines or an onboard CAN transceiver could make code complicated. These problems may possible be remedied in software in the future, although no guarantees are made.

CAN Interface (MCP2515 and MCP2551)

The PIC directly interfaces with the CAN transceiver since it has an onboard CAN module.
The Propeller interfaces with a CAN module (which then interfaces with a CAN transceiver) through SPI.
The advantage of an onboard CAN module is that it typically has a larger buffer and there is no communications overhead with retrieving data from an external module.
There is circuitry to lower the signal voltage from the +5v CAN transceiver to the +3.3v the CAN module operates at.

SD Card Interface (SD Card Holder, HS_DM1AA-SF-PEJ(21))

The microcontroller interfaces with a SD Card through SPI. The microcontroller also reads two additional pins from the SD Card holder, a write-protect indicator and card detect indicator through the IO Expander.

12v Measuring Equipment

Resistor divider to drop the +12v CAN supply voltage to a level acceptable by microcontroller inputs.
This may be used to periodically log the voltage on the +12v CAN supply, however, precision resistors would be required.
This is also used to detect whether the unit is operating on stored power (see Supercapacitor Bank section) or external power.

PIC Programming Header (6-pin 0.1" header)

6-pin header to program the PIC microcontroller using a PICkit 2 or equivalent device.
Note that the Propeller microcontroller is programmed through UART and FT232.

EEPROM (24LC256)

A 256kBit I^2C EEPROM chip is used storing data between power cycles.
For the Propeller chip, this is where the program is stored.

UART to USB (FT232)
A FT232 chip is used to allow the microcontroller UART to be accessed from a computer with USB. This is mainly used for debugging purposes - `printf` equivalent.
For the Propeller chip, this is how the program is downloaded to the device. This also includes the necessary reset circuitry.
IO Expander (MCP23017)
Since the PIC does not have sufficient pins to do all IO, an external I^2C IO controller is used to set LED outputs and read switch inputs.
Supercapacitor Bank
Supercapacitors are used to sustain power for ~5 seconds after power down to log any remaining CAN data and neatly dismount the SD Card.
There is circuitry to limit inrush current - the resistor limits charging current, while the diode allows discharge at whatever current necessary for board operation.
Likely, we would only need to populate one supercapacitor on the actual board to meet specifications.
Dedicated External Data Bus
Provisions for interfacing to anything required in the future. Includes a 4-wire interface (meant for SPI), 2-wire interface to the internal I^2C data bus, and a 2-wire interface containing a protected analog input and ground.
One possible use for the analog input was to implement a CVR.
Power Supply
The +12v CAN supply voltage is reduced to +5v and then +3.3v (both required for operation of certain chips) through linear regulators.
External Bench Power
Fancy term for 12V barrel jack interface. Includes a bridge rectifier so polarity of the power supply is not a concern.
LEDs
Blinking LEDs to indicate what the device is doing. Functions and proposed functions are:

LED	**Off**	**Blinking**	**On**
FT232 RX/TX LEDs	Governed by the FT232 chip.		
SD	SD Card Dismounted	SD Read/Write Activity	SD Card Mounted & Idle
CAN	CAN not connected	CAN Activity	Idle CANbus
Active	Fault	Normal operation	Fault
Fault	No fault condition	Undefined	Fault User attention required

(Possible) Front-Panel Interface
2-wire (signal and ground) wire-to-board connections for any possible front panel components - like panel-mount switches or LEDs.
Miscellanea
A rubber duck with an internal LED SHALL be included somewhere.

<u>Firmware</u>
The software basically reads data from the CAN network, and writes it to a SD Card using the FAT32 file system.
Main modules are listed below:
"Glue Logic"
Reads CAN data and writes it to the SD Card. Also handles error conditions and user I/O (for example, blinking LEDs, sending data to the driver's screen, or initiating actions based on pressed switches).

CAN Library
Implements CAN communications. With the PIC microcontroler, DMA is used to read CAN in the background and buffer it to memory. With any other microcontroller, an external CAN module must be polled periodically or interrupts must be used.
Datalogger File Format
Exact standards are to-be-decided. Goals include:
Extensibility - so that more things than CAN data can be stored, such as audio data from the CVR, voltage data, or external sensor data not coming from the CAN network.
Sequential - should only write one file at a time, and write that file sequentially. This is for performance reasons - to avoid alternating between writing the file table and data.
Error recovery - should be designed so that the rest of the file is recoverable if certain blocks are corrupt or skipped due to errors.
Human readability - it would be nice to encode data in ASCII, although this increases the size and data rate requirement.
FAT32 Library
Implements the FAT32 file system, so that a standard PC without specialized software can read the data contents. This may be optimized for Datalogger's single-file continuous sequential write operation. This is layered on top of the SD Card Library.
Relevant parts of the file table may be cached on the microcontroller to assist in fast block allocation. Additionally, there should be provisions to allocate large chunks for a file quickly to minimize the number of overhead writes required.
SD Card Library
Implements communication with the SD Card, including card read, card write, and card information retrieval. On microcontrollers with DMA, reads and writes may occur in the background so user code can execute concurrently and do data processing in parallel. On other microcontrollers, it may be possible to allow the programming phase of the SD Card write to occur in the background, although this has not been tested.
Likely, if background operations are involved, the write buffer will be double-buffered. That is, there are two buffers, and while one is being shifted out of the SPI module to the SD Card, the other is being filled with data.
RFC 1149 / 2549 / 6214
There are currently no plans to implement IPoAC.

Firmware Coding Conventions
This is a proposed set of coding rules. This is NOT binding, and is definitely open to revision.
Note that current code does not completely conform to this, although we're working on that!

Physical layer here is defined as code which interfaces with low-level hardware features, such as the SPI module or IO ports. Typically, this is highly dependent on the microcontroller used and not portable.

Documentation
All functions MUST have a short Doxygen-comparible description of what they do and the parameters they take / return. If there are special meanings to certain parameter values (like a pointer[out] of value `NULL` means don't write that data field), those MUST be included in the documentation.
All global `const` variables or `#define`s MUST include a short Doxygen-compatible description of what they mean. An exceptions is made for related lists of these variables (such as register addresses on an external chip) provided that there is an overview of the list at the top of the list.
Complicated code segments SHOULD include comments to increase readability.
Reliability
Code MUST be written according to team-wide reliability specifications (whenever those come out)
Constants SHOULD use `const` instead of `#define`, and functions SHOULD use `inline` instead of `#define`. This allows type-checking at compile time. One exception to this rule would be physical layer code, where constants (such as UART baud and CAN bit rate) need to be calculated at compile time. Currently, `const` variables cannot be sanity-checked at compile time, so we will continue using `#define` for those.
Any function call which may return an error value MUST either handle the error or propagate it upward. Silent failures MUST NOT be allowed.
All parameters to functions SHOULD be checked for sanity. Pointers SHOULD be checked against null.
Any potentially infinite loops SHOULD include a time-out counter to avoid triggering the watchdog timer forcing a reset and corrupting SD Card data.
Portability
The non-physical-layer portions of the FAT32 and SD Card library implementation SHOULD be portable - that is, hardware independent, and access the physical layer through well-defined interfaces.
Native C data types (`int`, `unsigned int`, etc) SHOULD be avoided in portable code. It is RECOMMENDED that those be replaced with `typedef`'ed types based on the variable size to avoid complier implementation dependence.

Desktop Software
There will likely be a desktop software component to parse the data recorded on the SD Card.
Proposed functionality:
Automatic anomaly detection and error flagging in logged CAN messages.
Ability to generate graphs of various parameters over time - such as +12v voltage (with glitch flagging), motor output / speed, or position (if GPS is hooked into the CAN network). This data may be relevant to other teams such as race strategy.
Cross-platform / platform-independent. Java comes to mind.
Easy-to-use, attractive, GUI.

Appendix

BRAIN - The standard microcontroller for most CalSol circuit boards. Contains an Atmel ATMega324 with a FT232 chip for USB communications and onboard CAN through a MCP2515 CAN module and MCP2551 CAN transceiver.
CAN / CANbus - Controller Area Network, a data bus standard primary used in vehicles, and used for communicating between different circuit boards in the solar car. Wikipedia has all the gory low-level details including physical layer specifications and frame format and contents.
CAN Transceiver - An integrated circuit which changes a logic-level CAN signal to the voltages necessary for transmission on the CAN network.
CAN Module - A device which generates logic-level CAN signals based on input data. These may be integrated on the microcontroller, or may be on an external integrated circuit.
CVR - Cockpit Voice Recorder - One of two components on aircraft black boxes, the other being the Flight Data Recorder. This device is self-explanatory - it records cockpit audio. Both pilot conversations and background noises may help in accident investigations.
DMA - Direct Memory Access - A feature on some microcontrollers which allows peripheral memory accesses to take place in the background. For example, while the processor is executing user code (possibly accessing memory), CAN data can be retrieved and stored in a memory buffer, or SD Card write data can be shifted out onto the SPI bus.
I^2C - Inter-Integrated Circuit, also known as "2-wire serial" - Basically, a data bus standard for communicating between chips. Multiple devices can be attached to the same two lines.
SPI - Serial Peripheral Interface, also known as "4-wire serial" - Basically, a data bus standard for communicating between chips.
UART - Universal Asynchronous Receiver/Transmitter - A simple standard for transmitting serial data. Commonly used to output text to a computer screen.

Additional Reading

SD Card Simplified Physical Layer Spec - contains details on interfacing with the SD card through SPI.
MISRA C Coding Standards - contains guidelines on producing safety-critical code. We don't have it and we won't use it since it's $20, but it's listed for completeness. If you have a physical copy (or know where we could borrow one), it might be worth a read. If not, **don't** buy it, since we won't use it.
JSF Air Vehicle C++ Coding Standards - contains guidelines on producing safety-critical code. Some of it is derived from the MISRA C Coding standards. It's quite restrictive and contains details about C++ which we wouldn't use, so this is more of stuff to consider rather than absolute coding rules.
Datasheets for various components - self-explanatory.