VirtualEngineIO V1.7 Specification: A physical emulation of ECU commands for AVC

ES/Infotainment Division – ALTEN Sverige AB
Last Edit: 2019/03/04 – Ali Nakisai

VirtualEngineIO

For the development of a demo of the Instrument Cluster for the “ALTEN Virtual Cockpit” vehicle, we needed some sort of physical device, that could generate events and messages from an ECU. We came up with an Arduino Mega board that emulates some of the commands from an ECU. The piece of firmware is called **VirtualEngineIO**.

These commands have a simple customized format and are sent from the virtual ECU to the AGL system, using Serial port. These packets will be first received by a piece of software called the **VirtualEngine**. This software will convert the messages to equivalent CAN messages, while simulating/emulating the behavior of a real vehicle’s different parts, including the engine. It will generate a simulation of the “speed” and “RPM” values based on the status of the engine, e.g. brake and gas ‘pedals’, gears, etc.

This piece of documentation, describes the specifications that were in mind for the operation of the VirtualEngineIO.

What is emulated

The VirtualEngineIO firmware, emulates following categories of signals:

(A) Signals regarding activation/deactivation of warning lights in the instrument cluster, including: (1) Left Blinker (2) Engine State Light (3) Oil Warning (4) ABS Brake Warning (5) Battery On (6) Seatbelt Warning (7) Doors Open Warning (8) Front Projector Indicator (9) Hand Brake Warning (10) Right Blinker

(B) Acceleration and Brake ‘pedals’: There are two buttons defined for these two pedals.

(C) Analog Gauges: There are three gauges defined: the fuel, the oil and the temperature gauge. These three gauges are simulated by an analog potentiometer (pot). Their value is scaled in range of 0~100. Whenever a change occurs, the MCU has to inform the host system.

<NOTE> In order to switch between the gauges the pot is emulating, an auxiliary button is provided. The change’s in pot value with result in **differential** changes in the actual gauge value, e.g. if the fuel gauge is selected and is currently on 25, and the pot is increased 30 units, the fuel becomes 55. It is evident that the pot has a limit on turning in both clockwise

and counter clockwise directions. A rotary encoder can be employed instead to overcome this problem.

(D) Buttons for engine ignition and hazard lights. The logic is same as **(A)**.

(E) The gear box button. The button has a more complicated logic. Three different messages could be sent to the host. If it is clicked, the gear goes one up. If it is double clicked, the gear goes one down. If it is long-held, the gear switches between manual and automatic. The logic for gear switching is not implemented here. It is implemented in the VirtualEngine.

MCU Logic

The MCU has to process the status of all inputs and report changes to the host. The analog signals are shared through a single analog pot (as explained previously) and a switching button. The logic to provide three different analog signals is called ‘fake analog input’ in the source code.

The commands are defined in the following section. The commands are send over a UART link (serial port), which for the Arduino Mega that we have, is carried over an FTDI USB-Serial converter. Therefore, the baud rate is important, as there is no virtual USB-Serial converter. The baud rate is set to 115200 as default.

Hardware for the digital signal: Each digital signal represents a button. The I/O pin of Arduino is internally pulled high and the button connects the pin to system GND (0V) when pressed. Therefore the key is **Active Low**. A logic NOT is placed to reverse the logic, so whenever the key is pressed, the memory values are set to **1** (and vice versa). The pin signal must be low-pass filtered in order to avoid ‘button bouncing effect’.

Hardware for the analog signal: There is one common pot to emulate all 3 different analog signals. The pot’s center leg is connected to the MCU. Two side legs are connected to VCC (5V) and GND (0V) respectively. This allows generation of a variable voltage in range of 0~5V on the analog pin. Capacitors could be added for better noise immunity. The voltage is read in range of 0~1023 by the MCU and is converted to range of 0~100. There is a simple digital ‘averaging’ mechanism in place to overcome and reduce the noise effects.

The processing loop: The input processing routine (IPR) occurs at least every 1 millisecond. So the maximum number of times the IPR is run is 1000 per second. The IPR basically has very small overhead, but in some iterations it may take longer and the number of times the IPR is run per second maybe less than 1000 (because of reasons such as UART message composition overhead, or IRQ handling delays).

Digital Signals: Last ‘stable’ status of each digital signal is stored. Whenever a change is detected (e.g. transition from 0 to 1 or vice versa), the change

must be retain for a fixed amount of time called **Debouncing** time. The idea is to filter short-term noises or button bouncing effects. If the change is retained for the span of the **Debouncing** time, the change is recorded as a the new stable status of the key.

- For some keys, the transition of key state to 1 is reported (such as most of warning light keys).

- For some keys, transitions to 1 and 0 are both reported (such as pedals).

- For the gear box key, three circumstances are checked: (1) Single click (2) Double click (3) Long-holding. To process these three incidents, the MCU keeps the number of subsequent clicks and the amount of time the key is held or released.

An event is generated and sent over the UART link. For first type of digital signals, only transitions to 1 are reported (when the key is pressed).

All time constants are #defined in the source code.

<NOTE> The debouncing time is currently 20ms. It is evident that is it takes 20 IPRs or less before the change is considered stable. It is the actual time (in ms) that matters not the number of IPR iterations.

<NOTE> The maximum delay between two clicks to be counted as subsequent ones is 250ms. The amount to time to count as a long-hold is 750ms.

Analog Signals: First, three values for three ‘fake’ analog inputs are emulated. These values are updated by the ‘analog faker’ mechanism. The analog pot ‘rotates’ to be assigned each of the 3 channels, when the switch key is clicked.

Then, last status of each analog signal is stored (either fake or real channel). There is an averaging system in place to reduce the noise on analog signals. There is a global counter called the ‘filter counter’. On every IPR iteration, the value of each analog signal is read by the Analog-Digital converter (ADC) and added to an accumulating register. After a fixed number of iterations (**Averaging Iterations**), the accumulator is divided by the number of samples and a ‘low-pass filtered’ value is calculated. This value is in range of 0~1023 and will be converted linearly to range of 0~100. For every analog signal, if there is a change in comparison to last status, the change is reported.

Current value for **Averaging Iterations** is 256. This number allows for low-level arithmetic substitution of division operator to right shift operator.

Contrary to the digital signals, it is the number of IPRs that matters in counting the averaging iterations, not the time that actually passes. Therefore, the updates may occur at least every 256 x 1ms = 256ms or slightly higher.

<NOTE> When the MCU is reset, all values are reset to 0. The default behavior is that no event is generated on reset.

Command Format

For each category of the four aforementioned categories the command is listed as follows.

A) When a button related to one of the 10 warning lights is ‘pressed’:

```
ICON $N<\r\n>
```

Where $N is index of the icon (0~9) and <‘\r\n’> is the carriage return + the line feed (code 13=0x0D followed by code 10=0x0A).

The exception is warning light with index 7 (ICON 7 – the front light beam), which reports when the key is pressed or released:

```
ICON 7 $V<\r\n>
```

Where $V is the key state: ‘P’ for ‘pressed’ and ‘R’ for ‘released’.

B) When a button related to the acceleration/brake pedal is pressed.

```
ACCL $V<\r\n>

BREK $V<\r\n>
```

Where $V is a single character ‘P’ for ‘pressed’ and ‘R’ for ‘released’.

C) When an analog signal value is changed:

```
GAUG $C $V<\r\n>
```

Where $C is type of the gauge: ‘F’ for fuel, ‘O’ for oil and ‘T’ for temperature.

Also $V is the value from 0 to 100.

D) When the ignite or hazard light buttons are pressed.

```
IGNT<\r\n>

HZRD<\r\n>
```

E) There is a button to switch between analog channels (Switch Analog key). This key is used for the logic in analog faker. This key doesn’t generate any messages.

F) The gear button. The respective messages if the key is clicked, double clicked or long-held are as follows.

```
GEAR<\r\n>
GEAR S<\r\n>
GEAR L<\r\n>
```

<NOTE> Contrary to a usual user interface system, not both click and double click messages are reported at once. If the key is double clicked, there is no message for a single click.

Wiring Diagram

In current code, the pinout is as follows. The list of buttons and the pinout are now in a seperate C++ header file: InputList.h. The hardware for each type of pin is specified above and also in the following diagrams.

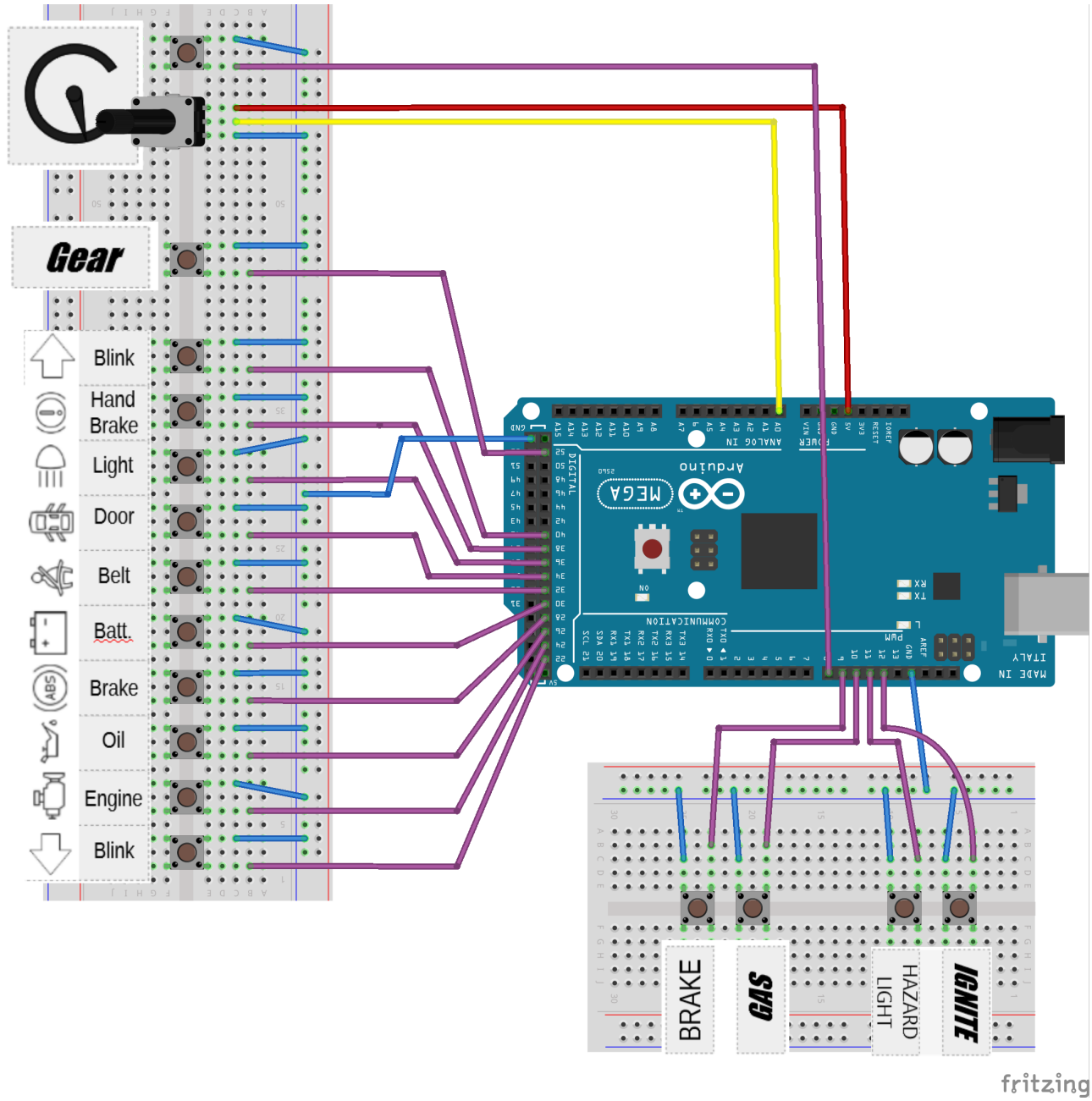
```
For digital pins which are connected to a button.

                                    /---\
  Arduino Digital Pin ---------------| O |-------------| GND
                                    \---/           -----
                                                     ---
                                                      -

For analog pins which are connected to a potentiometer.
                                                _
                                             ___|_|___
                                            |         |
  Arduino Analog Pin ------------           |_2__1__3_|
                                |             |   |   |
                                |-------------|   |   |---------| GND
                                              VCC           -----
                                                             ---
                                                              -
```

List of pins:

Name	Arduino Digital/Analog Pin	Event Name
Left Blinker	22	ICON 0
Engine Warning	24	ICON 1
Oil Warning	26	ICON 2
ABS Brake Warn.	28	ICON 3
Battery On	30	ICON 4
Seatbelt	32	ICON 5
Doors	34	ICON 6
Front Beam	36	ICON 7 $V
Hand Brake	38	ICON 8
Right Blinker	40	ICON 9
Accelerator	10	ACCL $V
Brake	9	BREK $V
Fuel Gauge	Fake #0	GAUG F
Oil Gauge	Fake #1	GAUG O
Thermo. Gauge	Fake #2	GAUG T
The pot	A0	N/A
Ignition Key	12	IGNT
Hazard Light Key	11	HZRD
Analog Switch	8	N/A
Gear Key	52	GEAR $X

<NOTE> The pot's resistance is not a specific value. Around 10KOhm should be Okay.

See next page for Fritzing's wiring diagram.



See ‘InputList.h’ for more information.