Design of an Arduino based low-cost error generator for PROFIBUS DP

Philippe Saey  
KU Leuven – Faculty of Engineering  
Technology – E&A (Energy & Automation)  
Gebr. Desmetstraat 1  
9000 Gent, Belgium  
philippe.saey@kuleuven.be

Ward Hauspie  
KU Leuven – Faculty of Engineering  
Technology – E&A (Energy & Automation)  
Gebr. Desmetstraat 1  
9000 Gent, Belgium  
ward.hauspie@kuleuven.be

Hendrik Derre  
KU Leuven – Faculty of Engineering  
Technology – E&A (Energy & Automation)  
Gebr. Desmetstraat 1  
9000 Gent, Belgium  
hendrik.derre@kuleuven.be

Annemarie Kokosy  
ISEN-Lille, Service Robotics Team  
Boulevard Vauban 41  
59000 Lille, France  
& LAGIS, UMR CNRS 8219,  
Villeneuve d’Ascq  
annemarie.kokosy@isen-lille.fr

Jos Knockaert  
Ghent University – Department Industrial System and Product Design  
Graaf Karel de Goede Laan 5  
8500 Kortrijk, Belgium  
jos.knockaert@ugent.be

Abstract

This paper briefly discusses the design of a low-cost, versatile and configurable error generator for PROFIBUS DP. Using a PROFICORE ULTRA oscilloscope trigger pulse, a small circuit designed around an Arduino MEGA 2560 disrupts messages on the industrial network, without adding an extra slave.

1. Introduction

More advanced teaching of PROFIBUS DP [1] [2] network technology requires in-depth hands-on exercises. For exercises relevant for industry, a lot of attention should be on diagnostics and on tracing errors. Typical “classic” errors (e.g. short circuit, open circuit, power failure of or removing the connector of a slave, removing a module from a slave, …) are easy to simulate during lab exercises. These errors are – relative to the DP cycle time – of very long duration. Removing connectors or modules, switching terminating resistors, … put a heavy wear on the equipment.

Disruption of a single (or a determined number of) message(s), configured for type, address, number of disruptions, … is not easily done. Recently, a small commercial slave was introduced on the market [3] to emulate some errors.

In this paper, a low-cost easy to use error generator is presented; it uses the oscilloscope trigger pulse of a PROFICORE ULTRA [4], a tool readily available in most labs and in industry. No extra DP-slave is to be configured for the network under test. Only an Arduino MEGA 2560 and some extra components are needed to build a handy, configurable and low-cost error generator for PROFIBUS DP.

The remainder of this paper is organized as follows: Chapter 2 discusses the basic input (adaptation of the voltage levels from the PROFICORE trigger pulse) and output (disrupting telegrams on RS485) electronics. Chapter 3 briefly discusses the first design of the complete error generator, and some results are presented in Chapter 4.

2. Basic design

2.1. From PROFICORE trigger pulse to Arduino input

Configuration of the specific telegram(s) to be disrupted is done in the oscilloscope trigger set-up of PROFITRACE [4], Fig. 1. The actual voltage level of the oscilloscope trigger output is only around 2.3 V. As
we use the digital inputs of the Arduino MEGA 2560, this is not a logic “high” level. Fig. 2 shows a circuit with 2 NAND gates, to adapt the level. Fig. 3 shows the trigger and adapted trigger with a DP message (voltage levels and decoded UART characters).

2.2. Disrupting telegrams on the RS485 physical layer

Using a MAX3086E driver [5] allows disrupting of the voltage levels on the RS485 bus. Using only one driver does not disrupt enough (due to the differential working). Mounting 2 RS485 drivers in two different configurations (Fig. 4, refer to pins RE and DE) yields sufficient disruption of the original voltage levels of the message.

The driver used during this first design phase is limited to 10 Mbps; most tests were at 1.5 Mbps (typical for industry use), some at 3 or 6 Mbps. Overall timing however allows for use with bit rates up to 12 Mbps, if another RS485 driver is used.

3. The error generator

3.1. Basic properties

Using standard Arduino libraries, the microcontroller introduces a delay of about 6 µs, which is too long. Code generated by Matlab/Simulink [6] – providing a complete graphical design – was also too slow.

Using a mixture of C code and assembly code using operations on bits resulted in a delay of typically 640 ns [7].

Typical delay times in the circuit can be found in Table 1: the total time varies between 800 and a practical maximum of about 1200 ns, due to the processing time in the Arduino. This means that typically the 2nd bit after a rising edge of the PROFICORE trigger signal is already disrupted (1 bit is 667 ns at 1.5 Mbps).

Even with a larger (variation in the) time delay of the Arduino – to be expected when adding a small display for HMI in a later phase – this timing should prove to be adequate: most messages still contain a lot of bytes (7.3 µs per byte at 1.5 Mbps) after the PROFICORE trigger. Use at higher bit rates is also possible with this timing.

An example of a healthy and a disrupted SD1 message is shown in Fig. 5.

Table 1. Typical delay times in the error generator.
Figure 4. The overall electronic circuit.

In this test design 4 input switches determine the number of consecutive (selected) messages that will be disrupted (0 to 15). 1 input switch enables the cycle time, and 3 switches determine the cycle time (0 to 7, unit is 100 ms). As already mentioned, configuration of the message type, address, … to be disrupted is done in PROFITRACE (Fig. 1).

Fig. 4 provides the overall electronic scheme, Fig. 6 shows the final look of this first version. Practical connection is shown in Fig. 7.

Figure 5. Healthy (top) and disrupted (bottom) SD1 message at 6 Mbps. Channel 3 (purple) shows the PROFICORE trigger, Channel 4 (green) displays the disrupting signals from the error generator.

Figure 6. View of the first design.

Figure 7. Connection overview.
4. Experimental results

Fig. 5 shows oscilloscope images of a disrupted SD1 message at 6 Mbps; it also contains the UART decoding done by in this case a DPO 2024 oscilloscope [8].

Fig. 8 is an example forcing 7 repeats by the master, invoked by disrupting the slave answer 7 consecutive times; the 8th PROFICORE trigger (purple) is under a correct (undisrupted) message. On top the voltage levels on the 2 bus lines, the inserted disturbance signal is in green. The PROFITRACE logging interpreting the disrupted telegrams is also provided.

![Figure 8. Forcing 7 master repeats with the Arduino based error generator.](image)

Disrupting e.g. a SD4 message (Token Pass) is not possible, as the oscilloscope trigger by the PROFICORE comes too late for this purpose (Fig. 9). Idem for a “Short Acknowledge” message.

![Figure 9. Trigger pulse (purple) at the very end of a SD4 message.](image)

5. Conclusions

A low-cost, versatile and configurable error generator for PROFIBUS DP has been developed. It proves very useful for training purposes.

The error generator uses an Arduino MEGA 2560 to detect the oscilloscope trigger pulse from a PROFICORE ULTRA. Processing this pulse to disrupt the chosen DP message, results in a total reaction time of between 800 and 1200 ns.

No extra PROFIBUS DP-slave is to be configured for the network under test and no messages are added; only existing messages are disrupted.

One can select messages and addresses using all options of the oscilloscope trigger of the PROFICORE. The Arduino program lets one select single or multiple consecutive disruptions, or “on timer” disruptions (repetition rates are multiples of 100 ms).

Token Pass and Short Acknowledge messages cannot be disturbed this way, because of the timing of the PROFICORE trigger pulse and the circuit.

Total cost is less than € 60.00 (using vendors providing quick delivery of components, so a further decrease of the price is possible), without the mounting box.

Future work includes the use of Galileo [9] or other faster (future) Arduino compatible boards. Using a RS485 driver supporting 12 Mbps, and upgrading the HMI with a small text display is also planned.

6. Acknowledgement

Aurélien Duquesnoy and Martin Heumez (MSc students Catholic University of Lille - ISEN), the members of the i-MOCCA team in KU Leuven, and the i-MOCCA project partners for sharing knowledge and know-how.

This work was partially funded by the Interreg IVa 2 Seas project 07-022-BE i-MOCCA.

References