FLAG: A Framework for <u>F</u>PGA-based <u>LoA</u>d <u>G</u>eneration in Profinet Communication

Ahmad Khaliq[1], Sangeet Saha[1], Bina Bhatt[1], Dongbing Gu[1], Gareth Howells[2], and Klaus McDonald-Maier[1]

Abstract—Like other automated system technologies, PROFINET, a real-time Industrial Ethernet Standard has shown increasing level of integration into the present IT Infrastructure. Such vast use of PROFINET can expose the controllers and I/O devices to operate in critical failures when traffic goes unexpectedly higher than normal. Rigorous testing of the running devices then becomes essential and therefore, in this paper, we prototype and design an FPGA based load Generating solution called FLAG (FPGA-based LoAd Generator) for PROFINET based traffic at the desired load configurations such as, bits per second, the number and size of the packets with their Ethertypes and MAC addresses. We have employed, a Zynq-7000 FPGA as our implementation platform for the proposed FLAG framework. The system can easily be deployed and accessed via the web or command line interface for successful load generation. Extensive experiments have been conducted to verify the effectiveness of our proposed solution and the results confirm that the proposed framework is capable to generate precise load at Fast/Gigabit line rate with a defined number of packets.

Index Terms—Profinet, FPGA, NetJury, load generation,

I. INTRODUCTION

The existence of Ethernet in the world of industrial automation and traditional communication systems [1] is inevitable. Traditional industrial fieldbuses (such as, Profibus) are relatively slow and due to their limited bandwidth and data sizes, these are now being replaced with real-time industrial Ethernet standard, PROFINET [2]. PROFINET can either be a Component Based Automation, known as PROFINET CBA or PROFINET IO. PROFINET CBA deals with machine to machine communication between distributed automation systems whereas in PROFINET IO, Ethernet based distributed field devices exchange data. This paper is focused on PROFINET IO.

In industrial environment, IO devices and PLC controllers communicate with strict response times [3]. Thus, network bandwidth plays an vital role to fulfill such stringent real-time demands. Performance evaluation of the running automation devices is desirable and requires a scrupulous testing scheme that can help to analyse the performance under various network condition for instance bandwidth [4]. Such scheme utmost requires a framework which can generate network traffic
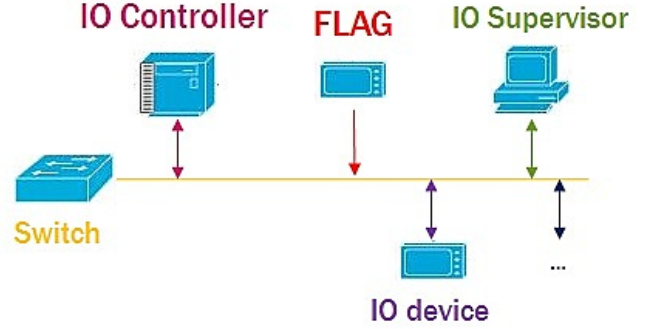
Fig. 1. FLAG in Industrial Profinet Communication.

in the form of synthesized packet with real-time attributes and timings. To evaluate the performance of the network and associated devices, this synthesized packets generation procedure can be termed as *load generation* or *net load* [5].

Network load generation is a system which sends synthesized user defined packets into the network while maintaining the desired throughput level. For real-time traffic inspection, common software-based solutions that either run on Windows or Linux have been typically used by the research community. For instance, scapy [6] and pyshark [7] are python-based software libraries commonly used for packet parsing, live packet capturing and for inspection purposes. Other open-source solutions (such as, netmap [8]) can also be employed and developed for sending user defined traffic while specifying each packet header fields.

However, such software-based solutions do not take care of the network available bandwidth and also, due to the involvement of the OS Kernel [8] in allocating memory and handling interrupts for the applications running in the user space, the throughput cannot reach the actual calculated level. In such circumstances, hardware-based solution, such as Field Programmable Gateway Arrays (FPGA) [9] can be an optimal solution to achieve the real-time measured throughput. Xilinx [10] being an innovator of hardware based programmable devices, such as FPGAs and SoC, enables the research community to rapidly innovate the ideas with low time cost and faster intelligent computing.

The recent boom of FPGA has encouraged the research community to shift from software to hardware-based solutions. Similar trends have been observed in the industrial automation systems [11] [12]. In [1], the authors analyzed the performance of PROFINET in motion control application. In [4], the authors present a case-study of PROFINET IO in industrial automation systems. Furthermore, [13] employed FPGAs for PROFINET

software IP cores but utilized the embedded processor (Nios) for the industrial communication. Authors in [14] for the first time taking advantage of the re-configurable feature of FPGA for PROFINET. [15] also shown that the FPGA-based implementation of Media Access Control (MAC) and Physical layer system (PHY) for industrial automation systems is fairly efficient.

With the advent of recent SoC-based solutions that combines programmable logic units (FPGA) with the Processing system consists of Dual-core ARM Cortex-A9 [16] for providing Linux based environments, it becomes relatively easier and efficient to perform time-dependent tasks. Such combination of hardware and software allows the research community to implement the low time cost operations in the Programmable logic unit (PL) while making use of PS for triggering those operations with the support of programming language including Python, C/C++, VHDL etc. Taking precedence of the re-configurable nature coupled with hardware and software subsystems of Xilinx NetJury [17], we have employed it as our implementation platform and proposed a hardware-based solution "FLAG" for PROFINET based traffic generation, as shown in Figure 1. Several experimentation shows its effectiveness in generating desired amount of the load with compact time limitations in PROFINET communication. The main contribution of this paper can be stated as:

- Proposed a framework, "FLAG" for generating network traffic with desired throughput/load in PROFINET communication.
- Integrated the framework with re-configurable FPGA-based NetJury device for user defined load generation on the fly.
- Precise experimental validation utilizing Hilscher netAN-ALYZER and tektronix DPO4054B oscilloscope-based analysis reveals the efficiency of our proposed framework in real-time industrial communication.

The remainder of the paper is organized as follow. Section II illustrates the brief overview of the FLAG framework. Section III presents an explanation of the implementation platform followed by the experimentation validation carried out in Section IV. The conclusion is presented in Section V.

II. FLAG Framework

This section will discuss the functionality of the proposed FLAG framework. In-short, our proposed architecture sends user defined packets into the network while achieving a desired throughput, also named as network load. We will first discuss the fundamental structure of a packet, followed by the discussion on the load generation scheme.

A. Fundamental of a Network Packet

A network packet is a formatted unit of data carried by a packet-switched network [18]. A packet is generally composed of control information and user data, often referred as header and payload. Delivering the payload depends upon the header information which includes source and destination network

or MAC address along with other sequencing information. A basic fields of a packet is shown in Table I.

Field	Length	Description
Preamble p	7 bytes	Synchronizes comminication
Start of Frame d	1 byte	Signals the start of a valid frame
MAC Destination M_d	6 bytes	Destination MAC address
MAC Source M_s	6 bytes	Source MAC address
802.1Q tag v	4 bytes	Optional VLAN tag
Ethertype or length E	2 bytes	Payload type or frame size
Payload P	42-1500 bytes	Data payload
CRC f	4 bytes	Frame error check
Interframe Gap i	12 bytes	Required idle Period between frames

Packet fields including Payload P with other fixed sized MAC addresses $M \in \{M_d, M_s\}$, Ethertype E, preamble p, frame start/delimiter d, frame check/crc f and optional v vLAN tag collectively represent a F frame with size S shown in equation (1).

$$S = \{p, d, M, v, E, P, f\} \qquad (1)$$

In this paper, by varying payload P, the overall untagged vLAN frame size $S \in \{72, 136, 268, 524, 1036, 1526\}$. However, if the frame is vLAN tagged, an extra 4 byte will be further added in S.

B. Load Generation strategy

Considering modern industrial automation devices, our proposed FLAG should be able to maintain the desired throughput, while allowing normal operation of the associated devices. Therefore, the desired throughput or network load should be less than the minimum bandwidth required by the network and devices. To maintain a L % of network load, the FLAG will send F number of frames. To make sure industrial controllers and IO devices keep operating normally, we have devised a strategy to send frames with gap I_L, called *Load Gap*. This I_L can be calculated as:

$$I_L = 12 + (12 + S) \times (\frac{1 - L}{L}) \qquad (2)$$

All F frames will have I_L gap in bytes. In equation 2, 12 represents the minimum gap, L denotes the desired load percentage where S be the frame size computed using (1). At $L = 100\%$, I_L will be 12 and all the network bandwidth will be going to be consumed by FLAG, therefore, it is important to set the L.

In real-time PROFINET communication, generating L load into the network with large number of F frames might take an infinite time. Therefore, it will be much more convenient if the user can be informed about the amount of time T within which the packets/load will be transmitted. This T time can be calculated using equation (3).

$$T = \frac{(S + 12) \times 8 \times F}{L \times R}, \ R \in \{100, 1000\} \qquad (3)$$

In equation 3, R denotes the total network bandwidth which can either be 100 for Fast Ethernet or 1000 for Gigabit

Ethernet. A minimum interframe gap of 12 is added into the S frame size and converted into bits by multiplying it with 8. Thus, T will be the actual time in seconds for FLAG to generate L % load by sending F number of frames into the industrial communication illustrated in Figure 2.
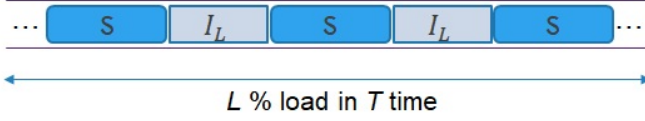

Fig. 2. S sized F frames with I_L load gap for L % load in T time.

III. FLAG ON ZYNQ-7000 FPGA SERIES

In this section, we will first go deep into the FPGA-based "NetJury" architecture employed for the implementation of the proposed FLAG. Later, we will discuss how the proposed FLAG framework mapped inside the NetJury.

A. NetJury background and its functionality

For the execution of the proposed FLAG framework, we have employed Xilinx Zynq FPGA based NetJury device [17] by Netmodule [19] in collaboration with Xilinx [17]. It is based on Zynq All Programmable SoC integrated with a Dual-core ARM Cortex-A9 processing subsystem [16]. The PL (FPGA) is designed to process real-time data and the Linux-based processing subsystem is used to execute the desired functionality in the PL unit. To program the PL subsystem, NetJury supports a proprietary scripting language, named as NetJury Scripting Language (NSL). A block diagram of the NetJury is shown in Figure 3.
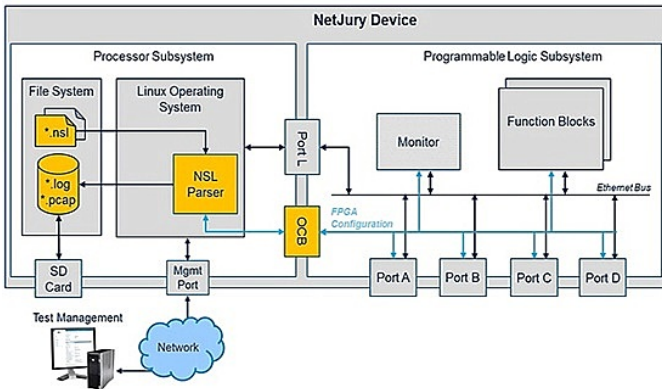

Fig. 3. Architectural view of the NetJury [17].

In Figure 3, the programming logic (PL) unit is on the right side and the left side is the Linux based dual-core ARM processor. The processor Subsystem (PS) can be accessed via Putty [20] or Winscp [21] utilizing the console port or through the Network via management port. Four Ethernet ports running at Fast/Gigabit bandwidth are provided for real-time Ethernet testing. Other components of the PL subsystem contain a Monitor module and hidden blocks containing registers. For each Ethernet port, packet generation with manipulation up-to the application layer can be performed via NSL scipt. The Port L (eth1) being a bridge port, communicate information between both the subsystems. With user defined NSL script(s), the PS can trigger the desired functionality in PL subsystem. Standard high-level programming languages, including Python, can be employed for the generation and execution of the NSL scripts. Upon execution of any test initiate by the user (if defined in the NSL) a log file and pcap file [22] generated in the PS block. The pcap file contains the actual packets being send/received on any of the FPGA ports. Within the Linux OS, pre-installed standard packet processing tools including scapy, iperf [23] and communication protocol stacks like IEC 61850 [24] and PROFINET are provided for packets analysis and generation.

1) Netjury Scripting Language (NSL): Typical NSL commands are shown in Table II. The NSL script is generally partitioned into three main phases (please see Figure 4):

- Setup Phase: Initialization of the Ethernet frame generators and analyser.
- Execution Phase: Ethernet frames are generated, and the analyser is enabled - the components run with high determinism and fulfill real-time requirements.
- Reporting Phase: The results of the execution phase are collected from each frame analyser instance and written to the log file in PS.

TABLE II
BASIC NSL COMMANDS [17]

Command	Meaning
REPORT	Description of the script and test case
DEFINE	Definition of constants and registers
REF	Test case reference
OCBM_WRITE	Setup of Generators and Analysers
ETH_TXRX_START	Start of execution phase
LOOP	Start of loop
WAIT_FOR	Delay in time or cycles
EXITONCHECK	Conditional exit
END LOOP	End of loop
OCBM _CHECK	Analyse results
ETH_TXRX_STOP	End of Execution phase

Using the commands in the Table II, parameters including the addresses of the registers, clockcycle, etc should be written in the NSL script. A pythonic NetJury library for NSL generation and execution is written for fast development and testing purpose.

2) Modes of the NetJury: There are three main modes of the NetJury; 1) Transparent mode, a default mode in which the traffic between all FPGA Ethernet ports can be accessed at Port L with no FPGA programming access. 2) Switching mode creates a bridge between Port A and Port B of the NetJury i.e. traffic received at Port A send out of Port B and vice-versa. 3) Scripting mode allows the user to program the PL unit for traffic generation and manipulation (if specified).

3) Packet Generator and Analyser: Each Ethernet port consists of a packet generator that triggers the packet sending process and an analyser which allows the packet manipulation. Both the generator and analyser are controlled via NSL (with

```
                        Setup
REPORT Names used in this script:
DEFINE CLOCKCYCLE 20 ns        -- system clock cycle in nanoseconds
DEFINE OCB_CYCLE 100 ns
REPORT Names used for master test controller
         .....
DEFINE TXA              0x40030000 -- base address of packet generator module
DEFINE TXB              0x40040000
DEFINE TXC              0x40050000
DEFINE TXD              0x40060000
DEFINE RXA              0x40034000  -- base address of packet analyzer module
DEFINE RXB              0x40044000
DEFINE RXC              0x40054000
DEFINE RXD              0x40064000
DEFINE TR_CTRL          0x2800
DEFINE TR_STAT          0x2804
         .....
DEFINE START_DELAY      0x2808     -- use with TX* only
DEFINE INTERFRAME_GAP   0x280C     -- use with TX* only
DEFINE NUMBER_OF_FRAMES 0x2810     -- use with TX* only
DEFINE HEADER_SIZE      0x2814
         .....
REPORT === Setup of generator B ===
OCBM_WRITE TXB START_DELAY       10        --bytes
OCBM_WRITE TXB INTERFRAME_GAP    12      --bytes
OCBM_WRITE TXB NUMBER_OF_FRAMES 10
         .....
REPORT .. Configuration of Header
OCBM_WRITE RXA ETHDST 0xff 0xff 0xff 0xff 0xff 0xff    -- setup of dst mac
OCBM_WRITE RXA ETHSRC 0x12 0x34 0x56 0x78 0x90 0xab    -- setup of src mac
OCBM_WRITE RXA HDR_AFTER_MAC 0x08 0x00           --field protocol/length : IP
OCBM_WRITE RXA HEADER_SIZE 14     -- bytes
OCBM_WRITE RXA PAYLOAD_SIZE 50
OCBM_WRITE RXA 0x0 0x10 0xf 0x88 0x92
         .....
                       Execution
OCBM_WRITE TXB TR_CTRL 1                     -- enable generator B
         .....
ETH_TXRX_START STARTSTOP_ALL    -- start all enabled devices
LOOP 50                         -- start of loop with about 5 ms timeout
WAIT_FOR 100 us
EXITONCHECKM RXC TR_STAT 0x2 0x2   -- exit loop if bit 2 is 1 (== done, all
others bits are masked out)
ENDLOOP
         .....
```

Fig. 4.   Sample NSL script with Setup and Execution phase.
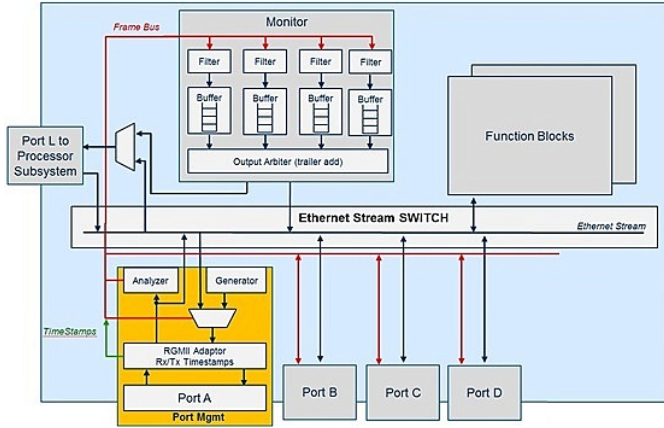

Fig. 5.   Structure of PL subsystem [17].

an NSL parser to decode it) which setup the PL registers through an On-Chip Bus (OCB). A detailed structural view of the Programming Subsystem is shown in Figure 5. A frame received at any of the Ethernet port is timestamped, computed at the arrival of start frame delimiter in the PHY clock domain with the compensation of the buffer latency. Its precision depends upon the dedicated clock of $8ns$ (system clock is $125MHz$). Traffic received at any of the Ethernet port can take multiple paths; under the switching mode, the Ethernet Stream divert it to other ports. Under the scripting mode, if enabled, the analyser inspects each packet up-to application layer, matches the user defined patterns, trigger the frames counters and append the frame with some event specific code tag (if tagging is enabled). Finally, it generates the statistics of the received traffic for automatic test evaluation and report generation. The Ethernet frame analyser attributes with their

description are shown in Table III.

TABLE III
FRAME ANALYSER ATTRIBUTES [17]

Attribute	Description
TRANSMITTER_CONTROL	Disable or Enable
TRASMITTER_STATE	Disable or Receiving — Hold
FRAMES_EXP	Total number of frames to be received until the Analyser will be automatically stopped
FRAMES_EXP_OK	Number of expected frames to be received until the Analyser will be automatically stopped
NUMBER_OF_RECV_OK	Number of matching frames received
NUMBER_OF_RECV_NOK	Number of frames received with mismatch
ERROR_CODE	Error code to be inserted into trailer frame is forwarded

Any frame which goes out of the NetJury's Ethernet port either comes from the generator under scripting mode or Ethernet stream path under switching mode. Under the scripting mode, the user needs to define the packet header parameters inside the NSL script. Features like calling specific registers to dynamically modify next frame data or loops for deterministic frame generation at wire-rate, all should be predefined inside the NSL. The Ethernet frame generator attributes are shown in Table IV.

TABLE IV
FRAME GENERATOR ATTRIBUTES [17]

Attribute	Description
TRANSMITTER_CONTROL	Disable or Enable
TRASMITTER_STATE	Disable or Transmitting or Done
INTERFRAME_GAP	Gap between transmitted packet
START_DELAY	Delay from reference point
NUMBER_OF_FRAMES	Number of frames transmitted with INTERFAME_GAP

4) Monitor: A monitor module is designed for accessing the traffic at port L either send by the generators or received by the analysers. There are two main components inside the monitor; 1) Filter that can remove undesired frames by specifying the source and destination MAC addresses and Ethertypes. When receiving traffic from multiple ports, at Port L, no frame usually has port information. In filters, enabling the port tag field feature, the user can append the packets with port information, 2) Buffer is the another component within which the incoming frames are stored. The monitor module is currently restricted to store 4 frames only, thus, we cannot expect to receive 100% traffic for analysis purpose.

B. FLAG on NetJury

To employ the NetJury device for FLAG framework, we go deep into the proprietary NSL script and develop it to send user defined number of frames at desired load configurations. A sample image of the NSL script is shown in Figure 4. Inside the setup block, the user needs to define the addresses of the registers including generators and analysers of the ports. Such information will be same for all the NSL scripts. Following these, the packet header fields such as destination and source MAC addresses, Ethertype and Payload size are

then defined. The execution phase includes triggering the registers to start and stop sending packets. For instance, setting the *TR_CTRL* to 1 amkes the specific port ready to send the frames. *ETH_TXRX_START* start sending the frames with *LOOP* and *WAIT_FOR* control the delay. *EXITONCHECKM* is for conditional exit but *ETH_TXRX_STOP* can also be used to stop sending frames otherwise. Since, we are interested in generating traffic only with no intention of analysing the response packets, therefore, we have not employed the analyser module (monitor) for load generation. Therefore, the user requires to provide the following fields to NetJury to be able to generate the desired load.

- *Source MAC address*
- *Destination MAC address*
- *Ethertype*
- *Payload size*
- *Number of frames*
- *Interframe gap*

IV. EXPERIMENTATION VALIDATION

This section elaborates on the performance efficiency of our *FLAG* framework in real-time traffic generation. In the PS side, packets are synthesized by using pythonic modules including Scapy and converted into NSL syntax (see Figure 4) for load generation. Furthermore, Hilscher netANALYZER [25] and a tektronix DPO4054B Oscilloscope [26] are employed for accurate measurements.

1) Command Line Interface (CLI) for Load generation:
To generate packets while maintaining load at desired %, we designed a console-based interface which can either run on Linux terminal or Putty running on Windows. The user needs to specify the percentage of load, packet headers' fields, packet size and number of frames to be send. In Figure 6, the user specified 100% load with packet size 1514 and 33510 frames. Therefore, it makes $S = 1526$, $I_L = 12$ and $T = 4.12370s$, also shown in Figure 6. Once, the FLAG generates the load, Figure 7 illustrates the number of total packets captured using Hilscher netANALYZER. It is quite evident that the last frame 33510th is captured at $4.1229s$ which is equal to the expected T.

```
=================================================================
 : load generator
-----------------------------------------------------------------
Please enter a load: 100
Load value is fine 100.000000
Please enter a source MAC address: 12:EB:74:06:55:6B
Source MAC address value is fine 12-EB-74-06-55-6B
Please enter a destination MAC address: 12:EB:74:CF:66:F9
Destination MAC address value is fine 12-EB-74-CF-66-F9
Please enter the Ethertype: 0x8892
Ethernet tag value is fine :0x8892
Please enter the packet size (60,124,252,508,1020,1514): 1514
S: 1526, InterframeGap : 12
Please enter the number of frames you want to send: 33510
Sending 33510 frames in 4.123070 Seconds
Load generated
```

Fig. 6. 100% load with 33510 frames

Furthermore, MATLAB decoding of the packets captured using Oscilloscope is performed. Figure 8 illustrates the deviation delay measured between the captured packets. With overall packet size of 1538 bytes, time per frame comes around $T_f = 123.04\mu s$ using equation (4). With oscilloscope, $F = 624$ packets are captured during a 80ms measurement with start time, $t_0 : 0s$ and end time, $t_1 : 0.076653s$, the overall load can be determined using equation (5), comes around 100% load.

$$T_f = \frac{(S + 12) \times 8}{R} \tag{4}$$

$$Load = \frac{(S + 12) \times 8 \times F}{t1 - t0} \tag{5}$$

No.	Time	Source	Destination	Protoc	Length	Info
33499	4.121630720	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33500	4.121753760	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33501	4.121876810	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33502	4.121999850	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33503	4.122122890	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33504	4.122245930	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33505	4.122368970	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33506	4.122492010	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33507	4.122615050	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33508	4.122738090	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33509	4.122861130	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)
33510	4.122984180	12:eb:74:06:55:6b	Dell_cf:66:f9	PNIO	1522	RTC1, ID:0x8000, Len:1494, Cycle: 1024 (Valid,Primary,Ok,Run)

Fig. 7. Hilscher netANALYZER measurement for 100% load with 33510 frames.
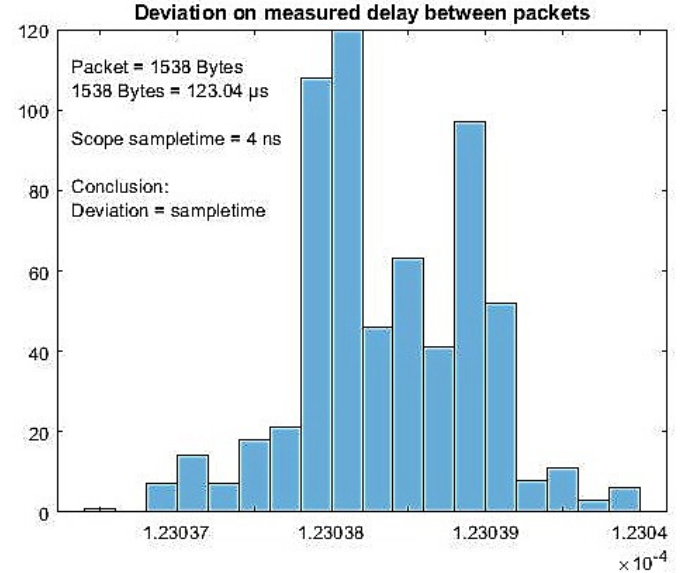

Fig. 8. Oscilloscope measurement for 100% load with 33510 frames.

2) Human Machine Interface (HMI) for Load generation:
For ease of access, we designed a Human Machine Interface (HMI) for load generation shown, in Figure 9, where the user can enter the desired load configurations. In this experiment shown, the user specified 50% load with 1514 packet and 31702 frames. Again, it makes $S = 1526$, $I_L = 1550$ and $T = 7.8012s$. Figure 10 confirms that the experimental time duration is equal to the calculated T. The last 31702th frame captured at $7.801s$, thus, $t_1 - t_0 = 7.801s$ and $F = 31702$, the load from equation (5) approximately comes around 50% load.

From the above test scenarios, it is evident that FLAG is efficient in generating real-time Profinet traffic. The load % can vary between 0 and 100. Also, vLAN tagged packets can

FPGA Based LoAd Generator (FLAG)

Fig. 9. 50% load with 31702 frames.

Fig. 10. Hilscher netANALYZER measurement for 50% load with 31702 frames.

also be generated with the proposed framework by changing the *HDR_AFTER_MAC* and *PAYLOAD_SIZE*. Our proposed FLAG framework can furthermore be used for generating load other than PROFINET by changing the Ethernet type / *HDR_AFTER_MAC*.

V. CONCLUSION

This paper presents a prototype and the design a hardware based solution "FLAG" for thorough testing of Industrial devices in real-time PROFINET communication. As an implementation platform, we integrate "NetJury" Zynq-7000 FPGA which is Zynq All Programmable SoC and proposed an architecture that can generates load by sending packets into the industrial network. For precise and accurate measurements, we have used Hilscher netANALYZER and tektronix DPO4054B Oscilloscope. Several experiments have shown that the proposed FLAG framework is capable to generate a constant load with defined number of packets. FLAG can be accessed either via terminal of Linux or Windows. Alongside, an HMI interface is also developed for ease of use.

ACKNOWLEDGMENT

REFERENCES

[1] A. L. Dias, G. S. Sestito, and D. Brandao, "Performance analysis of profibus dp and profinet in a motion control application," *Journal of Control, Automation and Electrical Systems*, vol. 28, no. 1, pp. 86–93, 2017.

[2] J. Feld, "Profinet-scalable factory communication for all applications," in *IEEE International Workshop on Factory Communication Systems, 2004. Proceedings.*, pp. 33–38, IEEE, 2004.

[3] S. Höme, S. Palis, and C. Diedrich, "Design of communication systems for networked control system running on profinet," in *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*, pp. 1–8, IEEE, 2014.

[4] P. Ferrari, A. Flammini, F. Venturini, and A. Augelli, "Large profinet io rt networks for factory automation: A case study," in *ETFA2011*, pp. 1–4, IEEE, 2011.

[5] "Netload." https://de.profibus.com/downloads/profinet-security-level-1-netload/.

[6] "Scapy." https://scapy.readthedocs.io/en/latest/.

[7] "pyshark." https://pypi.org/project/pyshark/.

[8] L. Rizzo, "Netmap: a novel framework for fast packet i/o," in *21st USENIX Security Symposium (USENIX Security 12)*, pp. 101–112, 2012.

[9] E. Monmasson and M. N. Cirstea, "Fpga design methodology for industrial control systemsa review," *IEEE transactions on industrial electronics*, vol. 54, no. 4, pp. 1824–1842, 2007.

[10] "Xilinx." https://www.xilinx.com/.

[11] M. Antolovic, K. Acton, N. Kalappa, S. Mantri, J. Parrott, J. Luntz, J. Moyne, and D. Tilbury, "Plc communication using profinet: experimental results and analysis," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*, pp. 1–4, IEEE, 2006.

[12] E. Monmasson, "Fpgas: Fundamentals, advanced features, and applications in industrial electronics [book news]," *IEEE Industrial Electronics Magazine*, vol. 11, no. 2, pp. 73–74, 2017.

[13] L. Dürkop, H. Trsek, J. Jasperneite, and L. Wisniewski, "Towards autoconfiguration of industrial automation systems: A case study using profinet io," in *Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, pp. 1–8, IEEE, 2012.

[14] H. Flatt, S. Schriegel, T. Neugarth, and J. Jasperneite, "An fpga based hsr architecture for seamless profinet redundancy," in *2012 9th IEEE International Workshop on Factory Communication Systems*, pp. 137–140, IEEE, 2012.

[15] T. T. T. Nguyen, Y. Nagao, T. Uwai, N. Sutisna, M. Kurosaki, H. Ochi, and B. Sai, "Fpga implementation of wireless lan system for factory automation," in *2018 International Conference on Advanced Technologies for Communications (ATC)*, pp. 78–83, IEEE, 2018.

[16] W. Wang and T. Dey, "A survey on arm cortex a processors," *Retrieved March*, 2011.

[17] "Netjury website." https://www.xilinx.com/products/boards-and-kits/1-4z9mkv.html.

[18] J. Akerberg and M. Bjorkman, "Exploring security in profinet io," in *2009 33rd Annual IEEE International Computer Software and Applications Conference*, vol. 1, pp. 406–412, IEEE, 2009.

[19] "Netmodule." http://www.netmodule.com/netmodule-home.html.

[20] R. Pearson, "Putty application tool," July 2 1974. US Patent 3,821,828.

[21] S. WinSCP-Free, "Scp and ftp client for windows."

[22] L. Deri *et al.*, "Improving passive packet capture: Beyond device polling," in *Proceedings of SANE*, vol. 2004, pp. 85–93, Amsterdam, Netherlands, 2004.

[23] P. Ferrari, A. Flammini, D. Marioli, S. Rinaldi, and A. Taroni, "Testing coexistence of different rte protocols in the same network," in *2008 IEEE International Workshop on Factory Communication Systems*, pp. 179–187, IEEE, 2008.

[24] R. E. Mackiewicz, "Overview of iec 61850 and benefits," in *2006 IEEE PES Power Systems Conference and Exposition*, pp. 623–630, IEEE, 2006.

[25] "Hilscher netanalyzer." www.de.hilscher.com.

[26] "tektronix dpo4054b." http://www.tektronix.com.