Working With Blob and VIM32 using Modbus TCP and ICE2/3

Guide to acquire raw g values from VIM32PP-E7DC8-0RE-IO-1V1401 using Blob connected to ICE2/3 IO-link master and Modbus TCP

Contents

This document shows how to work with Blob data on our VIM32PP-E7DC8-0RE-IO-1V1401 sensor.

Blob Transfer consists of utilizing ISDU (index service data unit) messaging in IO-link that usually is used to get or set the IO-link device parameters.

ISDU with Modbus

Receive ISDU Response Register = **x100**(Base 0) and **x101**(Base 1), here x is the port number so in case the VIM is on port 4 it will be 4100 or 4101 respectively.

Transmit ISDU Request Register = **x300**(Base 0) and **x301**(Base 1), here x is the port number so in case the VIM is on port 4 it will be 4300 or 4301 respectively.

Blob ID

Blob ID the VIM uses different Blob ID to indicate how the blob capture is triggered as mentioned below. Also included is a snippet from spec giving more info on Blob ID if further reading is required.

BLOB_ID:

-4096 = F000 = 0xF0 0x00 = Parameter trigger

-4098 = PDO trigger via IO-Link Master

-4097 = SSC trigger on switching edge

0 = Idle, no BLOB transfer active

6.5.2 BLOB_ID

The parameter BLOB_ID is used to indicate the current BLOB whose transmission is in progress. Its data type is IntegerT (16) and read only. The Device provides its available BLOB_IDs via its IODD (see A.2) and the associated parameter value within this Index.

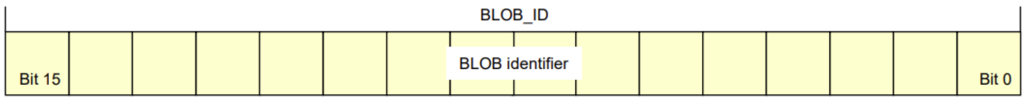


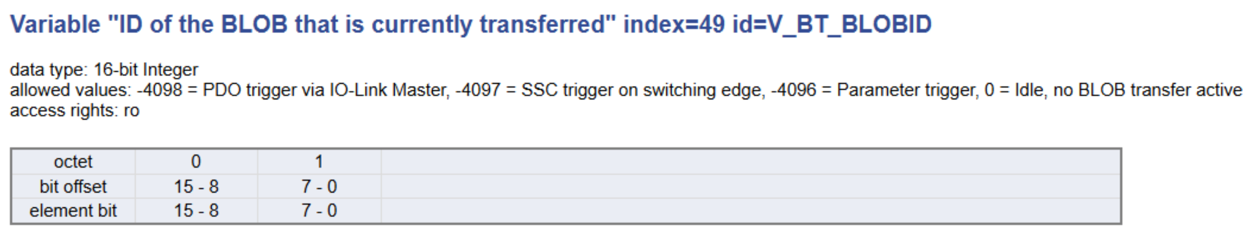
Figure 8 – Structure of BLOB_ID

Table 2 shows the coding of BLOB_ID, which implicitely indicates the read or write transmission direction.

Table 2 – Coding of BLOB_ID

Value (dec)	Definition
-32768	Not permitted
-32767 to -8193	Reserved
-8192 to -4096	Manufacturer specific (read)

The current Blob ID can be read using ISDU messaging on Index 49(0x31) sub-index 0(0x00)



Variable "ID of the BLOB that is currently transferred" index=49 id=V_BT_BLOBID

data type: 16-bit Integer
allowed values: -4098 = PDO trigger via IO-Link Master, -4097 = SSC trigger on switching edge, -4096 = Parameter trigger, 0 = Idle, no BLOB transfer active
access rights: ro

octet	0	1
bit offset	15 - 8	7 - 0
element bit	15 - 8	7 - 0

```python
def read_blob_id_status(port_number):
    write_register = port_number*1000 + 300
    read_register = port_number*1000 + 100
    client.write_multiple_register(write_register, [1, 49, 0, 2])
    client.write_multiple_register(write_register, [1, 49, 0, 2])
    response = client.read_register(read_register, 6)

    status = response[0]
    index = response[1]
    subindex = response[2]
    length = response[3]
    data = response[4]
    blob_id = data - 65536 if data > 32767 else data

    return {
        "status": status,
        "index": index,
        "subindex": subindex,
        "length": length,
        "data": data,
        "blob_id": blob_id
    }
```

To read the blob Id first a ISDU request is made by writing to the ISDU request register which is mentioned above.

The values written to the register are according to the ISDU structure of the ICE2/3 IO-link master, this might be different for other masters.

01 = operation is read

49 = index

00 = sub index

02 = data length

Once written to get the response the ISDU response register is read for 6 registers

The response is below

Read value: [8193, 49, 0, 2, 0, 0] from register: 4100

This can be translated as

8193 = Status bits

49 = index

0 = sub index

2 = data length

0 = Value of the parameter

In a case a Blob capture is triggered we might see something like

Read value: [8193, 49, 0, 2, 61440, 0] from register: 4100

Here 61440 has to converted to signed 61440-65536 = -4096 which represents that this is triggered via ISDU.

Trigger Blob Capture

Via ISDU messaging the Blob captured can be triggered by using the command 241(0xf1) following with Blob ID -4096(0xf0 0x00) or unsigned is 61440 which is 0xf1f0 for first register and 0x00 for second.

```
def trigger_Blob_capture(port_number):
    write_register = port_number*1000 + 300
    read_register = port_number*1000 + 100
    client.write_multiple_register(write_register, [2, 50, 0, 3, 0xf1f0,0x00])
```

02 = operation is a write

50 = index

00 = sub index

03 = data length (bytes)

Data is 0xf1f0 for first register and 0x00 for second register.

Blob Capture

```
def Blob_data_capture(port_number):
    write_register = port_number*1000 + 300
    read_register = port_number*1000 + 100
    client.write_multiple_register(write_register, [1, 50, 0, 201])
    time.sleep(0.2)
    response = client.read_register(read_register, 105)
    try:
        if response[4] == 4096:
            Total_data_size = (response[5] << 16) + response[6]
            print(f'The total data size is {Total_data_size} bytes')
            print(f'The total sample captured is {Total_data_size/1024} samples')
            input("Press Enter to continue...")
        else:
            Blob_data_parser(response[4:])
    except:
        pass
    return response[7]
```

To transfer the blob data captured by the VIM sensor, a read request has to be provided on index 50 sub-index 0

A slight delay is necessary between a ISDU request and response for it to perform. In the code its 200ms.

01 = operation is read

50 = index

00 = sub index

201 = data length (201 bytes)

Blob Data Parsing

The received Blob Data consists of raw g values and a counter and needs to be parsed to get the actual g value.

Initial Received Data

Once the transfer is started the first 200 bytes will have some information on the total capture.

[8193, 50, 0, 5, 4096, 171, 57344, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

8193 is status bit from the master on ISDU messaging.

50 is the index and 0 is the subindex of the request

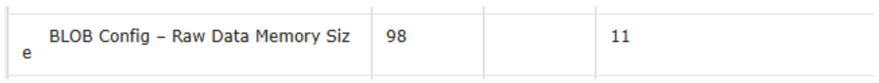
4096 converts to 0x10 0x00

Here 0x10 is the 1 = function (Read Info) and 0 = subfunction

0x00 combined with next 3 bytes give the total size of the data captured

0xAB 0xE0 0x00 = 11,264,000 bytes in total

This is equal to 11264000/1024 = 11,000 samples which is what we set in parameters for storage.



BLOB Config – Raw Data Memory Siz e	98		11

11 is 11,000

This is also used to calculate the size in the python function.

Consecutive Received Data

[8193, 50, 0, 201, 8192, 541, 47872, 542, 768, 542, 15616, 542, 30208, 542, 43008, 542, 57856, 543, 9216, 543, 32768, 543, 61952, 544, 25600, 544, 47360, 544, 56576, 544, 55552, 544, 44544, 544, 28928, 544, 10240, 543, 64000, 543, 59136, 543, 59648, 543, 57856, 543, 49920, 543, 36608, 543, 18176, 542, 60928, 542, 33536, 542, 7936, 541, 53248, 541, 39168, 541, 26368, 541, 16384, 541, 8704, 541, 4864, 541, 1536, 541, 256, 541, 4608, 541, 15872, 541, 34304, 541, 52992, 542, 6656, 542, 21504, 542, 35328, 542, 47872, 542, 61696, 543, 13056, 543, 35584, 543, 63744, 544, 24832, 544, 44544, 544, 52480, 544, 52480]

Here again 8193 is the ISDU status form the ICE2,

50 is the index and 0 is the subindex of the request

201 is the number of bytes received

8192 is 0x20 0x00

Here 0x20 is the counter number 32

0x00 combines with next 3 bytes gives the raw g value

541 = 0x21 0xD

47872 = 0xBB 0x00

Raw g = 0x00 0x21 0xD 0xBB = 138683 in decimal

actual_g = (raw # / 1969.3568) – 50

where 2^18 /133.11148 = 1969.3568

Thus actual_g = (138683/1969.3568) – 50 = 20.42045 m/s^2

The sample program captures, parses and converts the values to actual_g and stored to a CSV file for analysis.

Once all of the blob data is transferred the last value provided will be a checksum. The sample program gets out of the loop as the 4th value goes 0

Blob Finish Transmission

Once all the transmission is done, we need to send a finish command.

```
def Blob_finish_capture(port_number):
    write_register = port_number*1000 + 300
    read_register = port_number*1000 + 100
    client.write_multiple_register(write_register, [2, 50, 0, 2, 0x00f2])
```

02 = operation is a write

50 = index

00 = sub index

02 = data length (bytes)

Data is 0x00f2 for first register

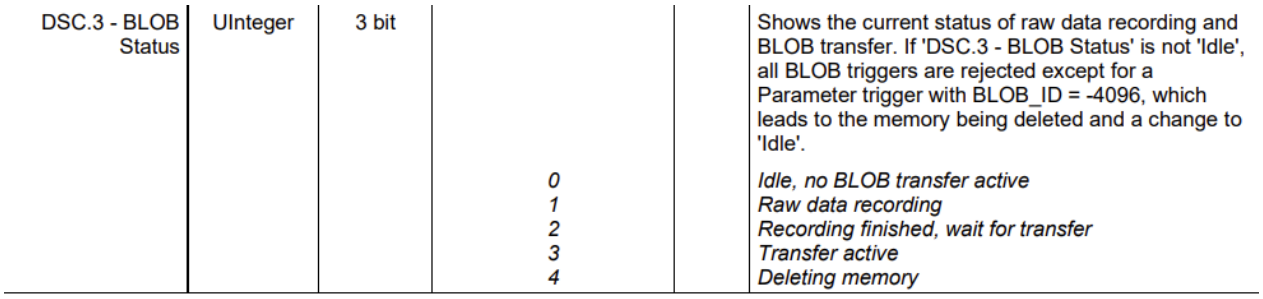
After this the whole process can be repeated.

Blob Status via PDI (Process Data Input)

Blob status can also be found from the PDI status 3 bit int value which is used to determine if recording has started and rest of the processes in the sample program.

```python
def Get_Blob_pdi_status(port_number):
    read_register = port_number*1000
    response = client.read_register(read_register, 15)

    if response:
        byte_value = response[11] & 0xFF  # Get the lower byte (00 part of fe00)
        bit_6_8_value = (byte_value >> 5) & 0x07  # Extract bits 6, 7, 8
        decimal_value = bit_6_8_value  # Convert to decimal number (0 to 3)
        #print(decimal_value)
        return decimal_value
    else:
        print("No response received")
```



DSC.3 - BLOB Status	UInteger	3 bit			Shows the current status of raw data recording and BLOB transfer. If 'DSC.3 - BLOB Status' is not 'Idle', all BLOB triggers are rejected except for a Parameter trigger with BLOB_ID = -4096, which leads to the memory being deleted and a change to 'Idle'.
			0		*Idle, no BLOB transfer active*
			1		*Raw data recording*
			2		*Recording finished, wait for transfer*
			3		*Transfer active*
			4		*Deleting memory*

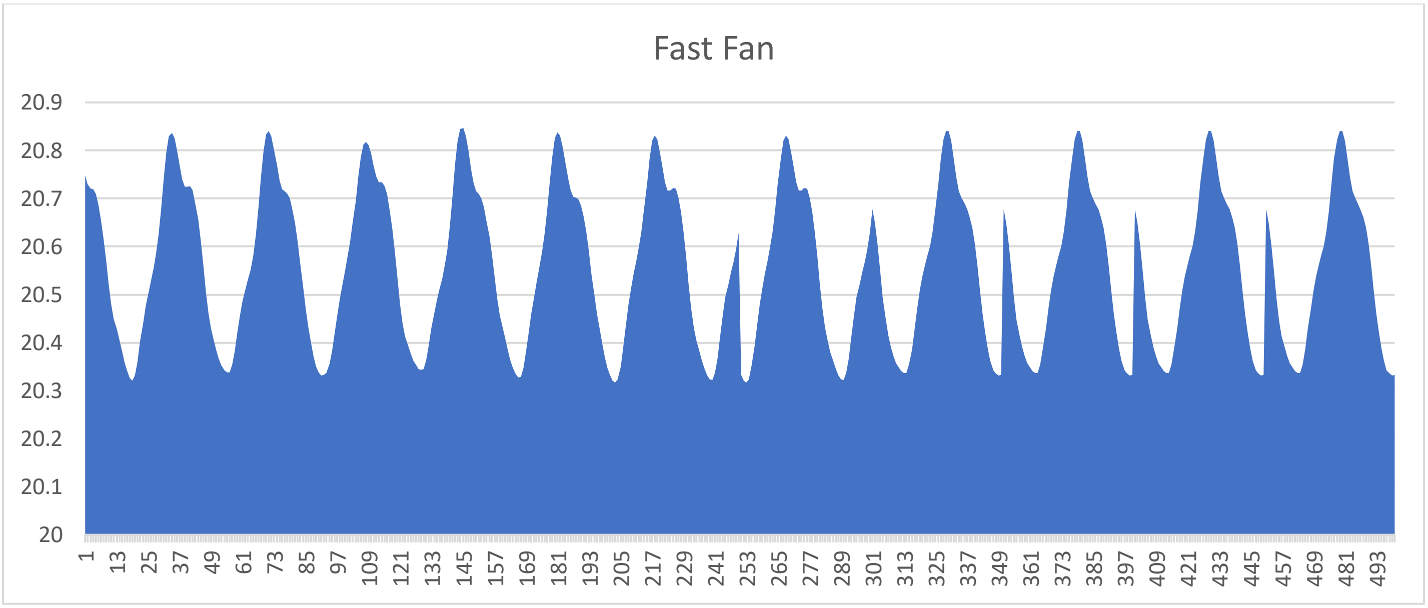
Results:

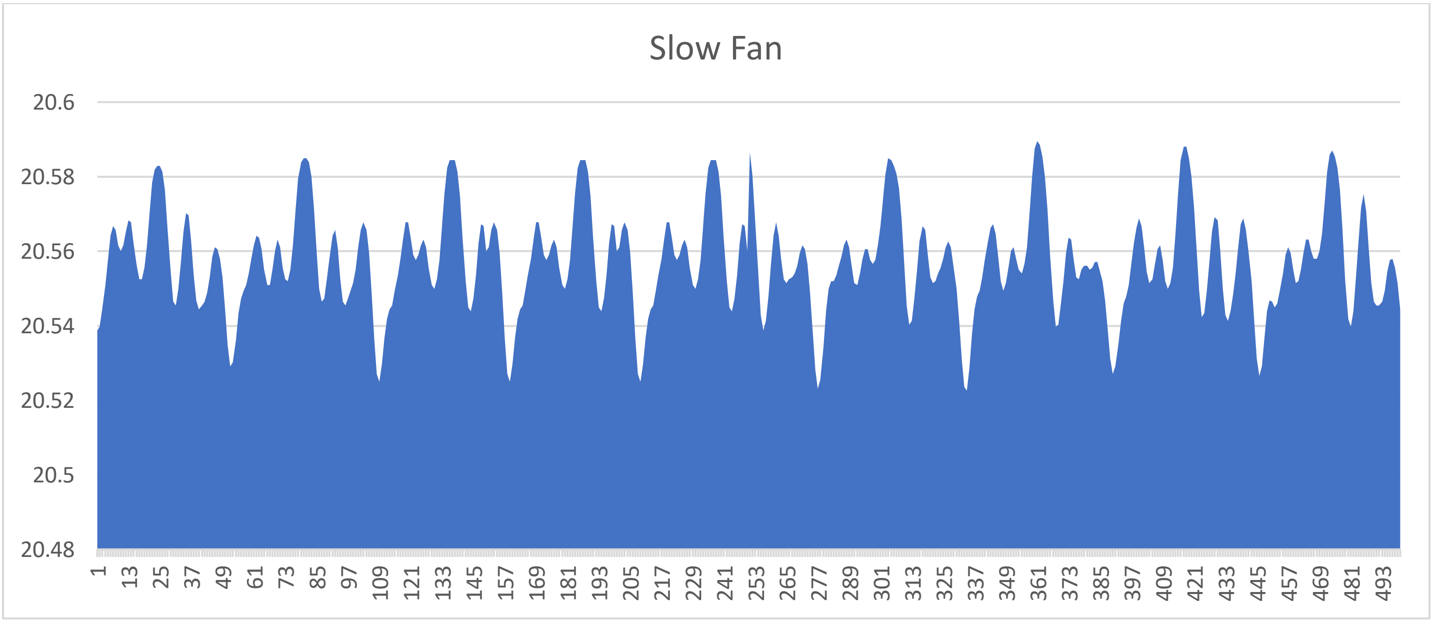
Few runs were made with VIM connected to a desktop fan.

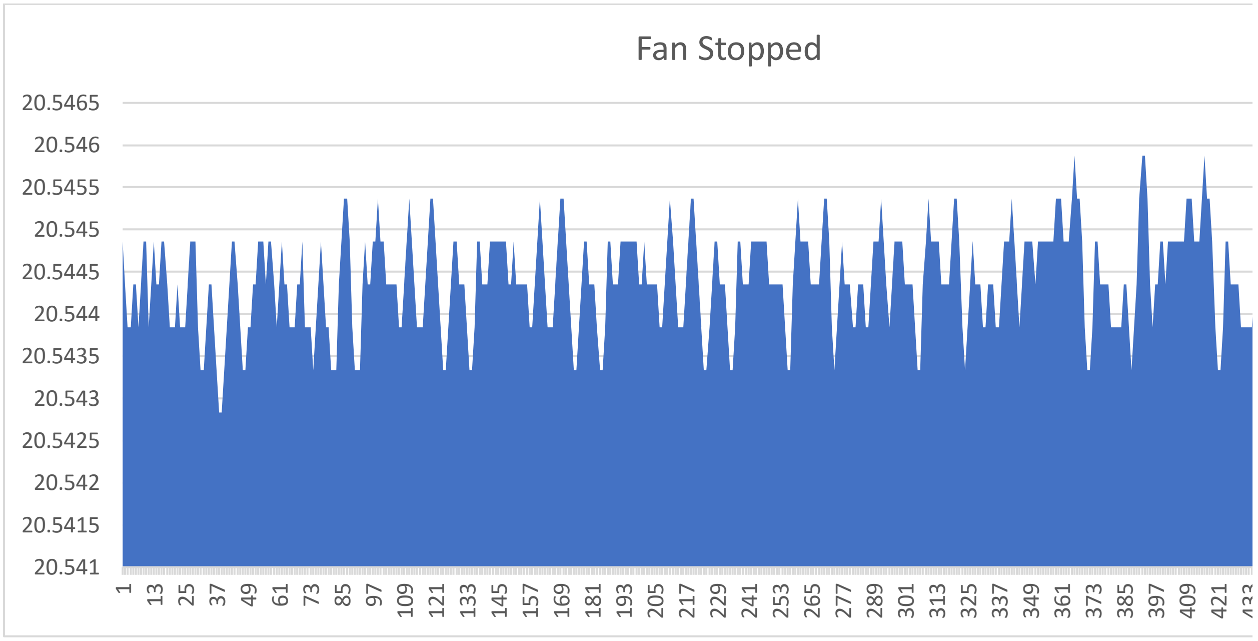
One with speed being very high

Other test with low speed

Last test with fan being turned off







If any questions please feel free to contact us.

Shahyan Bharucha
Field Application Specialist, Toronto
Product Specialist: IO-link, IIoT, Ethernet IO

Project Manager & Lead Application Developer – Smart Systems
Pepperl+Fuchs, Inc. Canada, Factory Automation
Phone: +1 330-486-0237
E-Mail: sbharucha@ca.pepperl-fuchs.com
www.pepperl-fuchs.com