

Document NumberRevisionAPN00**37**1.0

Prepared By NGB Approved By NB Page 1 of 22

Title Connecting to a KPV200 Vibration Sensor

CONNECTING TO A KPV200 VIBRATION SENSOR

1. Introduction

The KPV200 vibration sensor provides precise vibration monitoring and on-board signal processing for industrial and commercial applications. Equipped with integrated Fast Fourier Transform (FFT) capabilities, the sensor performs real-time frequency domain analysis directly within the device, eliminating the need for external processing hardware.

This advanced functionality allows for the early detection of mechanical issues such as imbalance, misalignment, or bearing wear by identifying characteristic vibration signatures. The sensor features a broad frequency response range, high sensitivity, and a rugged design suitable for demanding environments.

This application note describes how to interface a KPV200 sensor to a Senquip device to enable remote vibration monitoring. The end application will:

- read the current vibration profile,
- provide an option to enable the learning of a typical profile,
- show the current profile and alarm profile on a bar chart,
- show the current alarm status,
- allow alarms to be cleared.



Figure 1- KPV200 Vibration Sensor

Extensive use of the Senquip scripting language will be used in this application note. Further details on the Senquip scripting language can be found in the <u>Device Scripting Guide</u>.



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title	Page		
Connecting to a KPV200 Vibration Sensor			2 of 22

2. Connection the KPV200 to a Senquip Device

A Senquip QUAD-C2 was used in this application.

An RS485 to USB converter was used to connect the KPV200 to a PC. We also connected the RS485 network to the QUAD. Having the PC on the network allowed us to run the sensor supplier software and then later run a terminal program to monitor traffic on the bus.

A vibration motor with an offset weight was attached to the vibration sensor, and the frequency was tuned to approximately 70Hz by changing the supply voltage.



_

Figure 2 - Vibration Motor

The Senquip QUAD was supplied with 24V, and the QUAD internal voltage generator was used to provide 12V to the vibration sensor through IO5. The voltage and current supplied to the sensor on IO5 were monitored.



Figure 3 – Vibration Sensor Connection to Senquip QUAD

The Senquip QUAD was configured with a *Base Interval* of 10 seconds and the serial port was set to RS485, with a baud rate of 115200, 8 bits, even parity, and 1 stop bit. The serial port *Mode* was set as scripted as the serial port will be completely controlled within a script.



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV200 Vibration Sensor			3 of 22

Serial 1		0
Name	Serial 1	
Interval	1	
Туре	RS485	~
Termination Resistor	Enabled	
Mode	Scripted	~
Baud Rate	115200	
Settings	8E1	
Powered by Output 1	□ Enabled	

Figure 4 - Serial Port Settings with Even Parity

IO5 (KPV200 Power)			0			
Name	KPV200 Power					
Interval	1	1				
Output						
Default State	VSET		~			
Measurement State	No Change		~			
Measurement Time	0		Seconds			
V mA	Hz	Duty	Pulse			

Figure 5 - IO5 Providing 12V and Monitoring Voltage and Current



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV200 Vibration Sensor			4 of 22

3. Getting to Know the KPV200 Sensor

_

Kjaerulf Pedersen, the sensor suppler, provide LabVIEW based software that shows live data and allows for settings updates.



Figure 6 - Supplier Provided Software Showing Live Vibration Data

Using the software, the sensor was configured as shown in Figure 7. A sampling frequency of 400Hz was chosen to allow a maximum frequency of 200Hz to be measured. For more information on sampling frequencies, see this article on <u>Nyquist frequency</u>. Note the even parity.

Direction EEPROM Z direction ▽	Sensor Address	Use data from file.	Edit Setup file
SampleRate EEPROM	PC Com port]	
G Rate EEPROM ±2G ▽	baud rate	Change Baud	
Change AlarmDelay	parity	Change Parity	
Change Alarmoffset	New Sensor Address	Change Address	
Start Learning Stop Learning Learning state active	Read Sensor Address	Sensor Node Address	





Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title		Page	
Connecting to a KPV200 Vibration Sensor			5 of 22

To test the functionality of the sensor, we started the learning function and ran it for 10 minutes. After that time, we could see that an alarm profile had been built that matched the envelope of vibration and had a 10mG offset as configured. We noticed that once cleared, the alarm would stay off for a short while and then turned back on. Because of this, we left the sensor in learning mode for an hour. After an hour, the alarm could be cleared, and it would stay cleared.



To better understand the interface to the sensor, we then moved to requesting Modbus data from the KPV200 using <u>Realterm</u>, a serial monitor program. The <u>KPV200 User Guide</u> contains a Modbus register map for the sensor which is duplicated in



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV200 Vibration Sensor			6 of 22

Appendix II – KPV200 Register Description.

As a sanity check, the baud rate was read back from the sensor.

	Device Address	Function Code	Register Add	dress	Number of F	Registers	Checksum	
	1 byte	1 byte	2 bytes		2 bytes		2 bytes	
Request	0xF7 (247)	0x03	0x01 (MSB)	0xF9 (LSB)	0x00 (MSB)	0x01 (LSB)	0x41 (MSB)	0x51 (LSB)

	Device	Function	Bytes to Follow	Data		Checksum	
	Address	Code					
	1 byte	1 byte	1 byte	2 bytes		2 bytes	
Response	0xF7 (247)	0x03	0x02	0x04	0x80	0x73	0x31
				(MSB)	(LSB)	(MSB)	(LSB)

In RealTerm, the green text shows the command, and the yellow is the response. Looking at the response data 0x0480 = 1152 or a baud rate of 115200 as expected.

📲 RealTerm: Serial Capture Program 2.0.0.57	_		Х
F7 03 01 F9 00 01 41 51 F7 03 02 04 80 73 31			
Display Port Capture Pins Send Echo Port 12C 12C-2 12CMisc Misc Misc 0xF7 0x03 0x01 0x41 0x51 Send Send Ascli +CR +CR +CR Send Numbers Send Numbers Send Ascli +CR +CR +CR +CR 0 ^C LF Rgpeats 1 Literal Strip Spaces +CR SMBUS 8 Dump File to Port Send Ejle X Stop Delays 0	n <u>Clear</u>	Freeze	> (2) > (2) > (3) ; (8) > (1) 3 (6) g (9)
	Port: 31	BRE BRE	AK r None

Figure 8- Baud Rate Read in RealTerm

3.1. Reading Vibration Data

Current vibration data is held in 128 input (function code 4) registers of type unsigned int.



7 of 22

Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page

Connecting to a KPV200 Vibration Sensor

Data Address	Description	Data Type	Value range
0	Vibration data first bin	Unsigned int	0-255
127	Vibration data last bin	Unsigned int	0-255

The Modbus RTU standard allows for a maximum message length of 256 bytes. Given that each register is 2 bytes, not all registers can be read in a single read. Either two reads must be done, or some of the registers must be forfeited, resulting in a lower maximum frequency. We will do two reads to retrieve all 128 register values.

The read commands for reading the first 64 and second 64 registers are shown below.

	Device Address	Function Code	Register Ad	dress	Number of F	Registers	Checksum	
	1 byte	1 byte	2 bytes		2 bytes		2 bytes	
First 64 bins	0xF7 (247)	0x04	0x00	0x00	0x00	0x40	0xE5	0x6C
			(MSB)	(LSB)	(MSB)	(LSB)	(MSB)	(LSB)
Last 64 bins	0xF7 (247)	0x04	0x00	0x40	0x00	0x40	0xE4	0xB8
			(MSB)	(LSB)	(MSB)	(LSB)	(MSB)	(LSB)

The read requests were tested in RealTerm and resulted in 64 registers (128 bytes) being returned for each request.

RealTerm: Serial Capture Program 2.0.0.57	_	o x
F7 04 00 00 40 E5 6C F7 04 80 00 80 00 06 00 04 00 02 00 04 00 02 00 04 00 02 <td< th=""><th>34 00 31 00 32 00 30 00 31 00 32 00 34 00 34 00 34 00 30 00 30 00 30 01 30 01 30 01</th><th></th></td<>	34 00 31 00 32 00 30 00 31 00 32 00 34 00 34 00 34 00 30 00 30 00 30 01 30 01 30 01	
Display Port Capture Pins Send Echo Port I2C I2C-2 I2CMisc Misc Misc 0xF7 0x00 0x00 0x00 0x40 0xE5 0x6C Send Numbers Send ASCII +CR +CR +LF 0xF7 0x00 0x40 0x24 0x88 Send Numbers Send ASCII +CR +LF 0 ^C LF Repeats 1 Literal Strip Spaces +crc SMBUS 8 0 ^C LF Repeats Send File X Stop Delays 0 0 0 C Literal Strip Stop Delays 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 <td>n <u>Clear</u></td> <td>Freeze</td>	n <u>Clear</u>	Freeze
Char Count:532 CPS:0	Port: 3 11	15200 8E1 None



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV2	00 Vibration Sensor		8 of 22

Similarly, the learned frequency profile, referred to as the Known Curve in the documentation, and the alarm profile, can be read as below. The Modbus CRC can be calculated using this online <u>CRC Calculator</u>. Note that the CRC bytes need to be flipped as the endianness of the calculator is different to the Modbus standard. In the final application, the CRC's will be calculated in the script on the Senquip device.

	Device	Function	Register Ad	dress	Number of I	Registers	Checksum	
	Address	Code						
	1 byte	1 byte	2 bytes		2 bytes		2 bytes	
Learned Profile	0xF7 (247)	0x03	0x00	0x00	0x00	0x40	0x50	0xAC
First 64 bins			(MSB)	(LSB)	(MSB)	(LSB)	(MSB)	(LSB)
Learned Profile	0xF7 (247)	0x03	0x00	0x40	0x00	0x40	0x51	0x78
Last 64 bins			(MSB)	(LSB)	(MSB)	(LSB)	(MSB)	(LSB)
Alarm Profile	0xF7 (247)	0x03	0x00	0x80	0x00	0x40	0x51	0x44
First 64 bins			(MSB)	(LSB)	(MSB)	(LSB)	(MSB)	(LSB)
Alarm Profile	0xF7 (247)	0x03	0x00	0xC0	0x00	0x40	0x50	0x90
Last 64 bins			(MSB)	(LSB)	(MSB)	(LSB)	(MSB)	(LSB)

3.2. Writing to the Sensor

To start and stop the learning function, and to clear alarm data, we need to be able to write to registers on the KPV200 sensor. Writing to the sensor is achieved with the write multiple function code (dec 16, hex 0x10).

	Device	Function	Registe	er	Number	of	Number	Data Bytes	5	Checksum	
	Address	Code	Addres	s	Register	s	of Bytes				
	1 byte	1 byte	2 bytes		2 bytes		1 byte	2 bytes		2 bytes	
Start	0xF7	0x10	0x02	OxBE	0x00	0x01	0x02	0x00	0x01	0x70	0xEA
Learning	(247)		(MSB)	(LSB)	(MSB)	(LSB)		(MSB)	(LSB)	(MSB)	(LSB)
Stop	0xF7	0x10	0x02	OxBF	0x00	0x01	0x02	0x00	0x00	0xB0	0xFB
Learning	(247)		(MSB)	(LSB)	(MSB)	(LSB)		(MSB)	(LSB)	(MSB)	(LSB)
Clear	0xF7	0x10	0x02	0xBC	0x00	0x01	0x02	0x00	0x01	0x71	0x08
Alarm	(247)		(MSB)	(LSB)	(MSB)	(LSB)		(MSB)	(LSB)	(MSB)	(LSB)

4. The Scripted Application

The scripted application will

- read the current vibration profile,
- allow an alarm profile to be learned,
- show the current profile and alarm profile on a bar chart,
- show the current alarm status,
- allow alarms to be cleared.

The script starts by issuing a Modbus command to the sensor. Based on the response to this command, further commands are issued. The process of issuing a command and receiving the feedback is asynchronous. Once the Modbus command is issued, the script continues on. This allows the Senquip device to continue with other functions while it is waiting for the sensor to respond and prevents the device freezing if the sensor never responds. A new callback function, *modparse_cb*, triggers when a valid Modbus response is detected on the serial port. The



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV2	00 Vibration Sensor		9 of 22

modparse_cb function checks the length of the message and CRC to ensure valid Modbus data before being executed.

Looking at the script, firstly, libraries are included, and variables are declared to store vibration, alarm data, and indicators as to whether the sensor is in a learning state and an alarm state.

Since the alarm profile does not change much over time, to save on transmitted data, it will only be sent periodically. The variables *cycle* and *dispatchAlarmProfile* are used to track when the alarm profile should be sent.

A debug function that sends data over Wi-Fi via UDP was used to assist in debugging. The UDP endpoint needs to be configured in the *Endpoint Settings* and a UDP client used to view the data. For more information on debugging with UDP, see Application Note <u>AN002 – Connecting a Senguip Device to a UDP Server over Wi-Fi</u>.

```
load('senquip.js');
load('api_config.js');
load('api_serial.js');
load('api_timer.js');
load('api_endpoint.js');
let vibration = []; // vibration profile
let alarm = []; // goal profile
let command = ''; // commands to be sent to the sensor
let learnstate = 0; // alarm status of sensor
let alarmstate = 0; // alarm state of sensor
let cycle = 0;
let dispatchAlarmProfile = false;
function debug(s) {UDP.send(s);}
```

_

A structure that contains all the Modbus requests that will be used is created. The requests are sent to the sensor using the *modbusSend* function that receives the command, adds a CRC, and sends it via the serial port. Note the additional *SERIAL.LOOPBACK* flag that allows transmitted serial data to be routed to the *modparse_cb* callback function to allow it to then look for a valid Modbus response.

A helper function, *nextRead*, is defined that uses a timer to schedule the next Modbus read after an interval. The function is used to schedule the next Modbus command based on the result of the latest response. For instance, when we have finished reading the first 64 registers of the current vibration data, we can schedule the read of the next 64 registers after a delay that allows for the processing of the first 64 to be completed and the processing function to be exited.



Document Number APN00 37	Revision 1.0	Prepared By NGB	Approved By NB
Title Connecting to a KPV2(00 Vibration Sensor		Page 10 of 22
<pre>let modbusCommand = 'value0' : '\xF7\x 'value1' : '\xF7\x 'alarm0' : '\xF7\x 'alarm1' : '\xF7\x 'alarmstate' : '\x 'startlearn': '\xF 'stoplearn' : '\xF 'clearalarm': '\xF };</pre>	{ 04\x00\x00\x00\x40', // rea 04\x00\x40\x00\x40', // rea 03\x00\x80\x00\x40', // rea 03\x00\x20\x00\x40', // rea 03\x00\x20\x00\x40', // rea 03\x00\x20\x00\x40', // rea 7\x04\x00\x80\x00\x01', // 7\x10\x02\x8E\x00\x01\x02\x0 '7\x10\x02\x8F\x00\x01\x02\x0 '7\x10\x02\x8C\x00\x01\x02\x0 '7\x10\x02\x8C\x00\x01\x02\x0	ad first 64 vibration values ad second 64 vibration values ad first 64 alarm values ad second 64 alarm values read the current alarm state 00\x01', // start learning the 00\x00', // stop learning the v 00\x01' // clear alarms	vibration profile ibration profile
<pre>function modbusSend(let crc = SQ.crc(c let crc_str = SQ.e let modbus_str = c SERIAL.write(1, mo }</pre>	<pre>cmd_str) { md_str); ncode(crc, -SQ.U16); md_str + crc_str; dbus_str, modbus_str.length,</pre>	SERIAL.LOOPBACK);	
<pre>function nextRead(ne // Pass index to t Timer.set(delay_ms</pre>	<pre>xt_cmd, delay_ms) { he timer function as the use , 0, function(next_cmd) { t_cmd);</pre>	erdata parameter	

The *data_handler* runs at the end of every base interval. It first checks if any requests have been issued by the user to start or stop learning, or to clear alarms. If a command has been issued, the Modbus command to execute this request is sent to the sensor. If no commands are pending, the first register to be requested from the sensor is the alarm status. Once the first Modbus command has been issued, the script continues to dispatch state information and exits. When a response to the Modbus request is received, the *modparse_cb* callback function will be triggered.

```
SQ.set data handler(function()
  if (command === 'startlearn') {
    modbusSend(modbusCommand.startlearn);
  else if (command === 'stoplearn') {
    modbusSend(modbusCommand.stoplearn);
  else if (command === 'clearalarm') {
    modbusSend(modbusCommand.clearalarm);
  else {
    dispatchAlarmProfile = cycle % 10 === 0;
    modbusSend(modbusCommand.alarmstate);
    cycle++;
  if (learnstate === 'Learning profile') {alarmstate = 'Monitoring off during learning';}
  SQ.dispatch(3,learnstate);
  SQ.dispatch(4, alarmstate);
}, null);
SQ.set trigger handler(function(tp) {
 if (tp === 1) { command = 'startlearn'; } // start learning
if (tp === 2) { command = 'stoplearn'; } // stop learnig
if (tp === 3) { command = 'clearalarm'; } // clear the alarm flag
  }, null);
```



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV20	00 Vibration Sensor		11 of 22

The *modparse_cb* function is configured and starts to monitor serial traffic on serial channel 1. In this instance, the function is configured to trigger when Modbus data is detected that completes a previous Modbus request. If no response is received after 350msec, the function will stop looking for a valid response.

```
// Possible Modbus parsing modes:
// 0 = Disabled
// 1 = Callback triggers for all requests and responses
// 2 = Callback triggers for only requests
// 3 = Callback triggers for valid responses that complete a request (sniffer bus data)
let mode = 3;
let timeout_ms = 350;
let serial_ch = 1;
SERIAL.set_modparse(serial_ch, mode, modparse_cb, timeout_ms, null);
```

The parsing of Modbus responses, and the issuing of subsequent Modbus requests is performed in the *modparse_cb* callback function. When triggered, the slave address, function code, register address, length of data, and the actual data are passed to the function. The function code and register address are used to identify which Modbus request is being responded to, and based on this, the data is parsed, and the next Modbus request is made. For instance, if a response to the alarm state request has been received, the alarm status is set, and the Modbus request for the first 64 vibration values is requested. If the first 64 vibration values response has been received, the vibration data is loaded into an array and the next 64 values are requested, and so on.



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV20	00 Vibration Sensor		12 of 22

```
// This callback fires when a valid Modbus request or response is detected (depending on the mode)// The
CRC, function code and length are used to check the message is valid
function modparse cb(slave addr, func, reg addr, data len, data) {
  let s = '';
  if (data !== null) {
    s = mkstr(data, data len);
    debug(JSON.stringify ({addr: slave addr, f: func, reg: reg addr, l: data len}));
    if (func === 16 && reg_addr === 0x02BE) { // start learning
      learnstate = 'Learning profile';
      command = '';
     nextRead(modbusCommand.value0, 400); // read first 64 registers of current value
    else if (func === 16 && reg addr === 0x02BF) { // stop learning
     learnstate = 'Learning complete';
      command = '';
      dispatchAlarmProfile = true; // get the learned profile on this cycle
      nextRead(modbusCommand.alarmstate, 400); // read first 64 registers of current value
    else if (func === 16 && reg addr === 0x02BC) { // clear alarm
      command = '';
      nextRead(modbusCommand.alarmstate, 400); // read first 64 registers of current value
    else if (func === 4 && reg addr === 0x80) { // alarm state
     SQ.dispatch(5,s);
if (s === '\x01\x00') {alarmstate = 'Alarm';} else {alarmstate = '0k';}
      nextRead (modbusCommand.value0, 400); // read first 64 registers of current value
    else if (func === 4 && reg addr === 0x00) {
     vibration = [];
      for (let i = 0; i < 64; i++) {</pre>
        vibration[i] = data[i*2+1];
      3
     nextRead(modbusCommand.value1, 400); // read second 64 registers of current value
    else if (func === 4 && reg_addr === 0x40) {
      for (let i = 0; i < 64; i++) {
       vibration[i+64] = data[i*2+1];
     SQ.dispatch(1, JSON.stringify(vibration));
      if (dispatchAlarmProfile) {
          nextRead(modbusCommand.alarm0,400); // read first 64 registers of alarm value;
      }
    else if (func === 3 && reg addr === 0x80) {
     alarm = [];
      for (let i = 0; i < 64; i++) {
        alarm[i] = data[i*2+1];
     nextRead(modbusCommand.alarm1,400); // read second 64 registers of alarm value;
    else if (func === 3 && reg_addr === 0xC0) {
      for (let i = 0; i < 64; i++) {
        alarm[i+64] = data[i*2+1];
     SQ.dispatch(2, JSON.stringify(alarm));
    }
 }
}
```



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV20	00 Vibration Sensor		13 of 22

5. Setting Up the Display

Vibration data is dispatched to the Senquip Portal as a stringified JSON array of numbers. By default, the when the Portal detects data formatted in this way, it will display it as a bar chart in a standard widget.

Vibra	tion
120 90 2 60 30 0	
08-Jan-25	14:58:14 [cp1]

Figure 9 - Default Display for a Stringified Array of Numbers

To see the actual data being sent, use the *eye* icon at the bottom right of the widget to change the display format to HEX.

Vibration	
5B 31 31 37 2C 35 2C 35 2C 34 2C 34 2C 32 2C 33 2C 34 2C 34 2C 32 2C 34 2C 33 2C	32
2C 32 2C 32 2C 33 2C 32 2C 32 2C 31 2C 32 2C 32 2C 32 2C 32 2C 33 2C 31 2C 32 2C	32
2C 32 2C 32 2C 32 2C 32 2C 32 2C 32 2C 31 2C 31 2C 32 2C 31 2C 32 2C 32 2C 32 2C 32 2C 31 2C	32
2C 31 2C 32 2C 31 2C 30 2C 31 2C 31 2C 31 2C 32 2C 32 2C 31 2C 31 2C 32 2C 32 2C	31
2C 32 2C 32 2C 31 2C 32 2C 31 2C 31 2C 31 2C 32 2C 32 2C 32 2C 31 2C 32 2C 31 2C 31 2C	31
2C 30 2C 31 2C 32 2C 31 2C 30 2C 32 2C 30 2C 30 2C 30 2C 30 2C 30 2C 31 2C 31 2C	31
2C 31 2C 30 2C 30 2C 30 2C 31 2C 30 2C 31 2C 31 2C 30 2C 30 2C 30 2C 32 2C 32 2C	32
2C 31 2C 30 2C 31 2C 30 2C 30 2C 31 2C 30 2C 30 2C 31 2C 31 2C 31 2C 32 2C 31 2C 31 2C	31
2C 30 2C 31 2C 30 2C 30 2C 30 2C 31 2C 30 2C 32 2C 30 2C 31 2C 32 2C 30 2C 31 2C	30
2C 31 2C 31 2C 32 5D	
08-Jan-25 15:00:14 [cp1]	ø

Figure 10 - Stringified Array of Numbers Shown as Hex

5.1. Displaying the Vibration and Alarm Data on a Chart

Use the Add Custom Chart option in the device display settings menu to create a new bar chart.

In this example, a Bar chart of width 2 and height 2 is chosen. The x-axis is named Frequency, and the axis is scaled so that the 128 data points map to an x-axis scale of 0-200.

A y-axis called mG is added and the range is set between 0 and 200mG.

Copyright © 2023 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV200 Vibration Sensor			14 of 22

The Vibration Data and Alarm data are added as series on the chart. The Alarm data is set as being a *Goal* series. The *Goal* series will always appear in front of the other series, and it only needs to be refreshed every 600 cycles. This is done to save the amount of data that has to be transmitted on every cycle.



_

The device page will now show the vibration data as a list of numbers, and the vibration and alarm values on a bar chart.





Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title		Page	
Connecting to a KPV20	15 of 22		

6. Results

_

With the sensor and vibration motor connected and data being received, the *Start Learning* function was initiated.



After a few minutes, the learning was terminated. The vibration data was seen to be below the alarm threshold in all cases.



After a short while, the Alarm Status widget change to "Alarm".



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page
Connecting to a KPV20	0 Vibration Sensor		16 of 22
-			
Alarm Status			



Alarm

[cp4] 🐵

08-Jan-25 15:16:44



Learning was initiated for a while longer and after having cancelled the alarm, it did not re-appear until the frequency of the motor was changed.



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title	Page		
Connecting to a KPV200 Vibration Sensor			17 of 22

7. Conclusions

_

The KPV200 vibration sensor was integrated with a Senquip QUAD to enable the following features:

- read the current vibration profile,
- provide an option to enable the learning of a typical profile,
- show the current profile and alarm profile on a bar chart,
- show the current alarm status,
- allow alarms to be cleared.

The integration was simple, with the sensor being powered by an output from the Senquip device. The new *modparse_cb* callback function allowed an asynchronous application to be written that allows the Senquip device to continue operating even when the volume of Modbus data being transferred is high.





Document Number Revision APN00**37** 1.0

Prepared By NGB Approved By

Title

Connecting to a KPV200 Vibration Sensor

Page 18 of 22

8. Appendix I – Example Script

```
load('senquip.js');
load('api_config.js');
load('api_serial.js');
load('api timer.js');
load('api endpoint.js');
let vibration = []; // vibration profile
                  // goal profile
// goal profile
let alarm = [];
let command = '';
                      // commands to be sent to the sensor
let learnstate = '';
                           // learn status of sensor
let alarmstate = 0;
                           // alarm state of sensor
let cvcle = 0;
let dispatchAlarmProfile = false;
function debug(s) {UDP.send(s);}
let modbusCommand = {
  'value0' : '\xF7\x04\x00\x00\x00\x40', // read first 64 vibration values
'value1' : '\xF7\x04\x00\x40\x00\x40', // read second 64 vibration values
'alarm0' : '\xF7\x03\x00\x80\x00\x40', // read first 64 alarm values
'alarm1' : '\xF7\x03\x00\xC0\x00\x40', // read second 64 alarm values
  'alarm1' : '\xF7\x03\x00\xC0\x00\x40',
                                                // read second 64 alarm values
  <code>'alarmstate' : '\xF7\x04\x00\x80\x00\x01', // read the current alarm state</code>
  'startlearn': '\xF7\x10\x02\xBE\x00\x01\x02\x00\x01', // start learning the vibration profile
'stoplearn': '\xF7\x10\x02\xBF\x00\x01\x02\x00\x00', // stop learning the vibration profile
  'clearalarm': '\xF7\x10\x02\xBC\x00\x01\x02\x00\x01' // clear alarms
1:
function modbusSend(cmd str) {
  let crc = SQ.crc(cmd str);
  let crc str = SQ.encode(crc, -SQ.U16);
  let modbus str = cmd str + crc str;
  SERIAL.write(1, modbus str, modbus str.length, SERIAL.LOOPBACK);
function nextRead(next_cmd, delay_ms) {
  // Pass index to the timer function as the userdata parameter
  Timer.set(delay ms, 0, function(next cmd) {
      modbusSend(next cmd);
  }, next cmd);
}
// This callback fires when a valid Modbus request or response is detected (depending on the mode)// The CRC,
function code and length are used to check the message is valid
// slave_addr: (int) the slave address
// func: (int) the function code
// reg addr: (int) the register address
// data_len: (int) the length of the register data in bytes
// data: (void*) the register data from the request or response
function modparse cb(slave addr, func, reg addr, data len, data) {
  let s = '';
  if (data !== null) {
    s = mkstr(data, data len);
    debug(JSON.stringify (addr: slave_addr, f: func, reg: reg_addr, l: data_len));
    if (func === 16 && reg addr === 0x02BE) { // start learning
      learnstate = 'Learning profile';
      command = '':
      nextRead(modbusCommand.value0, 400); // read first 64 registers of current value
    else if (func === 16 && reg addr === 0x02BF) { // stop learning
      learnstate = 'Learning complete';
       command = '';
       dispatchAlarmProfile = true; // get the learned profile on this cycle
```



Document Number APN00 37	Revision 1.0	Prepared By NGB	Approved By NB	
Title			Page	
Connecting to a KPV20	00 Vibration Sensor		19 of 22	
nextRead(modbus(Command.alarmstate, 400)	; // read first 64 registe	ers of current value	
<pre>else if (func === command = ''; nextRead(modbus)</pre>	16 && reg_addr === 0x02 Command.alarmstate, 400)	<pre>2BC) { // clear alarm ; // read first 64 registe</pre>	ers of current value	
<pre>} else if (func === SQ.dispatch(5,s)</pre>	4 && reg_addr === 0x80) ;	{ // alarm state		
<pre>if (s === '\x01' nextRead(modbus(}</pre>	<pre>\x00') {alarmstate = 'Al Command.value0, 400); //</pre>	<pre>Larm';} else {alarmstate = / read first 64 registers of </pre>	'Ok';} of current value	
<pre>else if (func === vibration = []; for (let i = 0; vibration[i] =</pre>	4 && reg_addr === 0x00) i < 64; i++) { = data[i*2+1];	(
} nextRead(modbus(Command.value1, 400); //	/ read second 64 registers	of current value	
<pre>else if (func === for (let i = 0; vibration[i+64</pre>	4 && reg_addr === 0x40) i < 64; i++) { 4] = data[i*2+1];	(
} SQ.dispatch(1,JS if (dispatchAlar nextRead(mod	SON. stringify (vibration) rmProfile) { dbusCommand.alarm0,400);); : // read first 64 register	cs of alarm value;	
<pre>} else if (func ==== alarm = []; for (let i = 0; alarm[i] = dat } nextBead(modbus(</pre>	<pre>3 && reg_addr === 0x80) i < 64; i++) { ta[i*2+1]; Command_alarm1.400); //</pre>	read second 64 registers (of alarm value:	
<pre>else if (func === for (let i = 0;</pre>	3 && reg_addr === 0xC0) i < 64; i++) { data[i*2+1];	{		
SQ.dispatch(2,JS } }	<pre>SON.stringify(alarm));</pre>			
<pre>/ Possible Modbus par / 0 = Disabled / 1 = Callback trigge / 2 = Callback trigge / 3 = Callback trigge et mode = 3;</pre>	rsing modes: ers for all requests and ers for only requests ers for valid responses	d responses that complete a request (s	sniffer bus data)	
<pre>et timeout_ms = 350; et serial_ch = 1; ERIAL.set_modparse(set)</pre>	erial_ch, mode, modparse	e_cb, timeout_ms, null);		
Q.set_data_handler(f u	unction()			
<pre>if (command === 'sta modbusSend(modbusC) } else if (command === modbusSend(modbusC) } else if (command ===</pre>	artlearn'){ Command.startlearn); = 'stoplearn'){ Command.stoplearn); = 'clearalarm'){			
<pre>modbusSend(modbus(} else { dispatchAlarmProfi </pre>	<pre>Command.clearalarm); ile = cycle % 10 === 0;</pre>			



Document Number APN00 37	Revision 1.0	Prepared By NGB	Approved By NB
Title			Page
Connecting to a KPV20	00 Vibration Sensor		20 of 22
<pre>modbusSend(modbusC cycle++; }</pre>	Command.alarmstate);		
<pre>if (learnstate === ' SQ.dispatch(3,learns SQ.dispatch(4,alarms)</pre>	<pre>Learning profile') {alarmsta state); state);</pre>	te = 'Monitoring off during le	arning';}
<pre>}, null);</pre>			
<pre>SQ.set_trigger_handler if (tp === 1) { comm if (tp === 2) { comm if (tp === 3) { comm }, null);</pre>	<pre>(function(tp) { and = 'startlearn'; } // st and = 'stoplearn'; } // st and = 'clearalarm'; } // cl</pre>	art learning op learnig ear the alarm flag	



21 of 22

Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Title			Page

_

Connecting to a KPV200 Vibration Sensor

9. Appendix II – KPV200 Register Description

FUNCTION CODE 03 - READ Holding REGISTERS								
Register	Data Address	Description	Data Type	Value range	System Value	Factory settings	Destination	Read write mode.
30001	0	Known curve First FFT BIN	Unsigned int	0-255		127	Eeprom	Read
30128	127	Known curve Last FFT BIN	Unsigned int	0-255		127	Eeprom	Read
30129	128	Alarm curve First FFT BIN	Unsigned int	0-255		137	Eeprom	Read
30256	255	Alarm curve Last FFT BIN	Unsigned int	0-255		1.0	Eprom	

FUNCTION C	ODE 03 - REA		EGISTERS (Ho	lding registers, Sett	ings) Eepro	om registers		
Register	Data		Data Type	Value range	System	Factory	Destination	Read write
Register	Address	Description	Data Type	value range	Value	settings	Desunation	mode
30501	500	Spare			Value	Jettings		mode.
30502	501	Spare						
30503	502	Spare						
30504	503	Spare						
30505	504	Firmware				1.0		
		version						
30506	505	Baudrate	Unsigned int	Value= baudrate 96 = 9600 192 = 19200 384 = 38400 576 = 57600 1152 = 115200		115200	Eeprom	Read / write
		Address						
30508	507	Parity	Unsigned int	0 = No parity 1 = Odd parity 2 = Even parity		Even	Eeprom	Read / write
30509	508	Spare				-		
30510	509	Spare				-		
30511	510	Direction. Choose X, Y or Z direction.	Unsigned int	0 = X 1 = Y 2 = Z		Z direction.	Eeprom	Read / write
30512	511	Samplerate	Unsigned int	10 = 100 Hz 11 = 200 Hz 12 = 400 Hz 13 = 800 Hz 14 = 1600 Hz 15 = 3200 Hz		3200Hz	Eeprom	Read / write
30513	512	G-Rate	Unsigned int	0 = ±2G 1 = ±4G 2 = ±8G 3 = ±16G		±2G	Eeprom	Read / write
30514	513	AlarmOffset	Unsigned int	0 - 255		10	Eeprom	Read / write
30515	514	AlarmDelay	Unsigned int	0 - 65535		0	Eeprom	Read / write



Document Number	Revision	Prepared By	Approved By
APN00 37	1.0	NGB	NB
Titlo			Ρασο

Title

Connecting to a KPV200 Vibration Sensor

FUNCTION CODE 03 – READ HOLDING REGISTERS (Holding registers, Settings) Dynamic registers FUNCTION CODE 16 – WRITE MULTIPLE REGISTERS (Holding registers, Settings)								
Register	Data Address	Description	Data Type	Value range	System Value	Factory settings	Destination	Read write mode.
30701	700	Clear Alarm	Unsigned int	1	0	0	Code	Write
30702	701	Reset to stored settings	Unsigned int	1	0	0	Code	Write
30703	702	Start Learning	Unsigned int	1 = Start Learning		0	Code	write
30704	703	Stop Learning	Unsigned int	0 = Stop Learning		0	Code	write
30705	704	Change Direction	Unsigned int	0 = X 1 = Y 2 = Z		Z direction.	Code	Read / write
30706	705	Change Samplerate	Unsigned int	10 = 100 Hz 11 = 200 Hz 12 = 400 Hz 13 = 800 Hz 14 = 1600 Hz 15 = 3200 Hz		3200Hz	Code	Read / write
30707	706	Change G- rate.	Unsigned int	0 = ±2G 1 = ±4G 2 = ±8G 3 = ±16G		±2G	Code	Read / write
1			1					

FUNCTION CODE 04 – READ INPUT REGISTERS								
Register	Data	Description	Data Type	Value range	System	Factory	Destination	Read write
	Address				Value	settings		mode.
40001	0	Vibration	Unsigned int	0-255	0	0	Code	read
		Data First						
		Bin						
40128	127	Vibration	Unsigned int	0-255	0	0	Code	read
		Data Last						
		Bin						
40129	128	Alarm Flag	Unsigned int	0 to 1	0	0	Code	read
		, and the set	energine a nit					····