

**Saab TransponderTech AB**

Appendices

Project designation

# R5 RIC

Document title

# R5 RIC STTv2 Video Protocol Specification

Distribution

**Saab TransponderTech AB**

# DOCUMENT VALIDATION

| | Name | Dept. Code/ Function | Date/ Signature |
|---|---|---|---|
| Prepared | Marcus Malm | | 2016-12-06 |
| Reviewed | | | |
| Approved | | | |
| Authorized | Christian Andersson | | 2016-12-06 |

# DOCUMENT CHANGE RECORD

| Issue | Change order | Date | Pages affected | Motive for and information on the change |
|---|---|---|---|---|
| A | | 16-12-06 | All | New document. |

# CONTENTS



# LIST OF FIGURES

**Saab TransponderTech AB**

# 1 OVERVIEW

R5 RIC (Radar interface card) is a Radar Extractor hardware and software solution that takes analogue radar video and converts to digital video that is sent out on broadcast UDP.

This document describes a digital radar video data protocol (STTv2) for digitized data sent from the R5 RIC.

## 2 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| ADC | Analogue to Digital Converter |
| Ethernet | Link layer in the internet protocol suite, just above the physical layer. |
| FPGA | Field Programmable Gate Array, Programmable logic. |
| IP | Internet Protocol. Internet layer in the internet protocol suite, above Ethernet. |
| IPv4 | Internet Protocol version 4, see Internet Protocol. |
| LSB | Least Significant Byte or Bit |
| MSB | Most Significant Byte or Bit |
| Msps | Mega samples per seconds, $1e^6$ samples / second. |
| R5 | Saab TransponderTech hardware and software platform |
| RIC | Radar Interface Computer |
| RLE | Run-Length Encoding |
| STT | Saab TransponderTech |
| UDP | User Datagram Protocol, Transport layer in the internet protocol suite, above IP. |

# 3 A NOTE ABOUT VIDEO DATA EXTRACTION

## 3.1 Down sampling the echo

Analogue video is digitized with a 16-bit ADC at 100 Msps. The digitized radar video is fed to the FPGA. The FPGA down samples the data in each echo to the requested data rate. Either the maximum sample or the last sample is picked during down sampling.

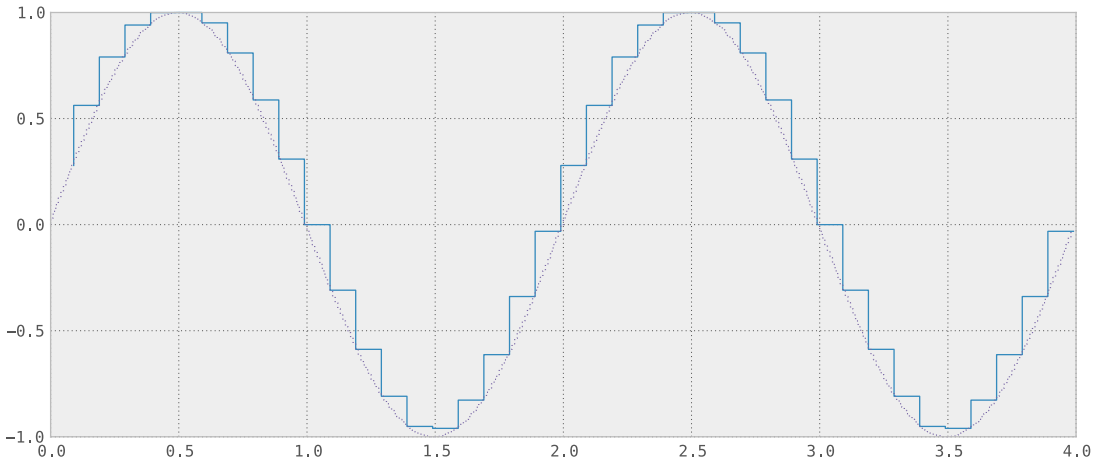


Figure 1: Down sampling at sample level

## 3.2 Resample a group of echoes into a sector

The output sector count is programmable with parameter ‘raiko.sector.downsample.count’. This is done by picking the maximum or latest sample at each sample in time/distance for each echo in a sector. This function is used to reduce the number of strobes per lap. This will also affect the azimuth counter.

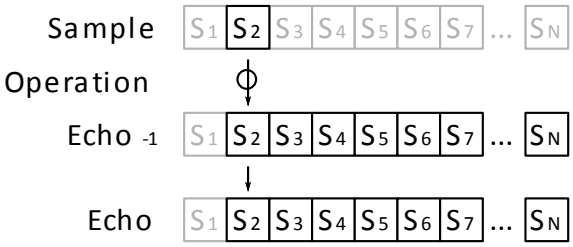


Figure 2: Echo resampling

## Saab TransponderTech AB

### 3.3 Grouping sectors into sector groups (Narrow)

To lessen the overhead of sending low resolution video (Narrow) there is an option to send multiple sectors per package. This means that multiple sectors are sent in order and fields in the header indicates which sectors are sent.

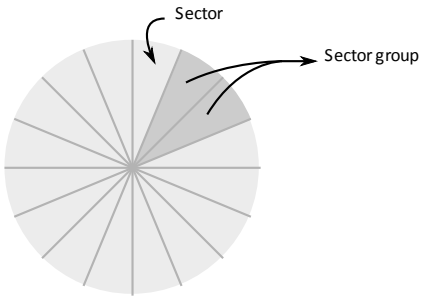


Figure 3: Sector groups

# 4 DIGITAL VIDEO PACKAGE

## 4.1 Overview

The digital video package carries the digital video data for a sector. This package also contains additional data about the sector. These additional metrics are called parameters.

This package is sent with UDP broadcast. All content is in network byte order (big endian).

## 4.2 Header

The header begins with the parameter count in a 32-bit field followed by the parameters. Each parameter has a 24-bit value and an 8-bit identifier. The parameter identifier is in the MSB.

| | 31:24 | 23:16 | 15:8 | 7:0 |
|---|---|---|---|---|
| 0 | Parameter count | | Version | Package mark |
| 32 | Parameter 1 Id | Parameter 1 Value | | |
| 64 | Parameter 2 Id | Parameter 2 Value | | |
| . | … | … | | |
| . | Parameter N Id | Parameter N Value | | |
| . | 255 | END_OF_HEADER (zero) | | |

The first field, Parameter count, describes the number of header fields including Parameter count and END_OF_HEADER.

### 4.2.1 Package mark

The package mark is zero for packages without fragmentation. For STTv2 Narrow the package mark is set to 0x56. If the package format is fragmented (described later) the most significant bit (0x80) is set.

### 4.2.2 Version

The first version was zero (0)

#### 4.2.2.1 Version 1

Following was added in version 1.

- 16-bit checksum
- Package fragmentation.

### 4.2.3 Parameters

Following parameters shall be supported. STT Extensions are appended after id 128 (0x80).

| Name | Id | Value / Unit | Comment |
|---|---|---|---|

| | | | |
|---|---|---|---|
| STREAM | 0 | 0-3 | Video Stream Name, configurable. |
| START_BEARING | 1 | 0-16384 | Bearing at the beginning of the first sector. |
| END_BEARING | 2 | 0-16384 | Bearing at the end of the last sector. |
| START_SECTOR | 3 | 0-($2^{16}$-1) | The first sector. The sector count has been extended to be configurable. |
| RANGE_SAMPLES | 4 | | Number of samples in the sector. |
| SWEEP_TIME | 5 | | Configurable (Always zero in RAW) |
| TRANSMISSION_MODE | 6 | | Configurable (Always zero in RAW) |
| VIDEO_BEAM_SELECT | 7 | | Configurable (Always zero in RAW) |
| VIDEO_TYPE_SELECT | 8 | | Configurable (Always zero in RAW) |
| TX_BEAM_SHAPE | 9 | | Configurable (Always zero in RAW) |
| ANTENNA ROTATION | 10 | [Rev/min] | Configurable |
| NO_VALID_VIDEO | 11 | | Always zero |
| BIN_SIZE | 12 | [100µm] | Length of a sample. |
| RX_BEAM_ELEVATION | 13 | | Configurable (Always zero in RAW) |
| DATE | 14 | 20YY-MM-DD | (Always zero in Narrow) |
| TIME | 15 | HH:MM:SS | (Always zero in Narrow) |
| MILLISECONDS | 16 | | Updated with approximately 4 Hz. (Always zero in Narrow) |
| BLOB_SIZE | 128 | 0-($2^{24}$-1) | Size in bytes of the binary block after the header. |
| SAMPLE_SIZE | 129 | 2,4,8,16 | Selector for 2, 4, 8 or 16-bit video samples. Default is 4. (16-bit only in RAW, 2-bit only in Narrow) |
| COMPRESSION | 130 | 0 = None,<br>1 = RLE | Compression algorithm used in the binary block. Default is None. |
| END_SECTOR | 131 | 0-($2^{24}$-1) | The last sector in this package. |
| PACKAGE_SEQUENCE | 132 | 0-($2^{24}$-1) | Package sequence number. |
| BYTE_OFFSET | 133 | 0-65535 | The byte offset of the binary block. |
| RANGE_OFFSET | 134 | [m] | Offset of the first sample. (Only used in Narrow) |
| DIGITAL_IO | 144 | 0-65535 | Bitmask for 16-bit digital in. (Only used in RAW) |
| ANALOG_IO_1 | 145 | 0-4095 | Value of analogue input 1. (Only used in RAW) |

| | | | |
|---|---|---|---|
| ANALOG_IO_2 | 146 | 0-4095 | Value of analogue input 2<br>(Only used in RAW) |
| ANALOG_IO_3 | 147 | 0-4095 | Value of analogue input 3<br>(Only used in RAW) |
| ANALOG_IO_4 | 148 | 0-4095 | Value of analogue input 4<br>(Only used in RAW) |
| END_OF_HEADER | 255 | - | Always zero |

### 4.2.4 Example headers

Following is an example Narrow header that contains 10 fields.

```
0000   00 0b 00 56 01 00 2b 84 02 00 36 5c 03 00 01 5c
0010   04 00 02 00 0c 12 4c 42 80 00 40 92 81 00 00 08
0020   82 00 00 01 83 00 01 b2 ff 00 00 00
```

| Name | Value |
|---|---|
| START_BEARING | 11140 |
| END_BEARING | 13916 |
| START_SECTOR | 348 |
| RANGE_SAMPLES | 512 |
| BIN_SIZE | 1199170 |
| BLOB_SIZE | 16530 |
| SAMPLE_SIZE | 8 |
| COMPRESSION | 1 |
| END_SECTOR | 434 |
| END_OF_HEADER | |

Following is an example RAW header that contains 17 fields.

```
0000   00 13 00 00 00 00 00 03 01 00 39 24 02 00 39 3c
0010   03 00 01 c9 04 00 04 00 05 00 00 01 06 00 00 02
0020   07 00 00 03 08 00 00 04 09 00 00 05 0a 00 00 06
0030   0b 00 00 00 0c 00 00 07 0d 00 00 08 0e 00 00 00
0040   0f 00 00 00 10 00 00 00 ff 00 00 00
```

- STREAM: 3
- START_BREARING - END_BEARING: 14628 – 14652
- START_SECTOR: 457
- …

### 4.3 Video data

After the video header a binary block is sent. The size of the binary block is BLOB_SIZE if COMRESSION is set otherwise the size is [RANGE_SAMPLES * SAMPLE_SIZE]. The binary block shall always be 16-bit word aligned in RAW. BLOB_SIZE must be set if COMPRESSION is set and is anything but None (0) in Narrow. If BLOB_SIZE is missing, the byte count can be calculated using the RANGE_SAMPLES, SAMPLE_SIZE, START_SECTOR and END_SECTOR fields. If END_SECTOR is missing the block is assumed to hold one echo. The binary block shall always be 16-bit word aligned.

The Narrow binary block consists of one or more radar echoes. Each echo is preceded with an echo header which consists of following data.

| Name | Size | Comment |
|---|---|---|
| Index | 16-bit | Sector index for the data |
| Size | 16-bit | Size of the data in bytes |

The sample size/compression shall be as described in the package header. Each echo data shall be 16-bit word aligned.

Following is an example with 31 8-bit samples.

```
0000   01 86 00 1f 8d 8e 90 91 93 95 97 98 9a 9c 9e a0
0010   a2 a4 a6 a9 ab ad af b2 b4 b6 b9 bb be c0 c3 c5
0020   c8 cb cd 00
```

| Name | Value |
|---|---|
| Index | 390 |
| Size | 31 |

Example of a RAW binary block with 4-bit samples.

| | 31:28 | 27:24 | 23:20 | 19:16 | 15:12 | 11:8 | 7:4 | 3:0 |
|---|---|---|---|---|---|---|---|---|
| 0 | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 |
| 32 | S9 | S10 | S11 | S12 | S13 | S14 | S15 | S16 |
| 64 | S17 | S18 | S19 | S20 | S21 | 0 | 0 | 0 |

Following is an example with 1024 4-bit samples.

```
0000   00 13 00 00 00 00 00 03 01 00 39 24 02 00 39 3c
0010   03 00 01 c9 04 00 04 00 05 00 00 01 06 00 00 02
0020   07 00 00 03 08 00 00 04 09 00 00 05 0a 00 00 06
```

```
0030   0b 00 00 00 0c 00 00 07 0d 00 00 08 0e 00 00 00
0040   0f 00 00 00 10 00 00 00 ff 00 00 00 88 88 88 88
0050   88 88 88 88 88 89 99 99 99 99 99 99 99 99 99 9a
0060   aa aa aa aa aa aa aa aa aa aa bb bb bb bb bb bb
0070   bb bb bb bb bb cc cc cc cc cc cc cc cc cc cc cc
0080   cc cd dd dd dd dd dd dd dd dd dd dd dd dd dd de
0090   ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee
00a0   ee ee ef ff ff ff ff ff ff ff ff ff ff ff ff ff
00b0   ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
00c0   ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
00d0   ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
00e0   ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
00f0   ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee ee
0100   ee ee ee dd dd dd dd dd dd dd dd dd dd dd dd dd
0110   dd cc cc cc cc cc cc cc cc cc cc cc cc cb bb bb
0120   bb bb bb bb bb bb bb bb ba aa aa aa aa aa aa aa
0130   aa aa aa 99 99 99 99 99 99 99 99 99 99 88 88 88
0140   88 88 88 88 88 88 88 77 77 77 77 77 77 77 77 77
0150   77 66 66 66 66 66 66 66 66 66 66 55 55 55 55 55
0160   55 55 55 55 55 54 44 44 44 44 44 44 44 44 44 44
0170   43 33 33 33 33 33 33 33 33 33 33 33 33 22 22 22
0180   22 22 22 22 22 22 22 22 22 22 22 21 11 11 11 11
0190   11 11 11 11 11 11 11 11 11 11 11 11 11 11 00 00
01a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01d0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01e0   00 00 00 00 00 00 00 00 01 11 11 11 11 11 11 11
01f0   11 11 11 11 11 11 11 11 11 11 11 22 22 22 22 22
0200   22 22 22 22 22 22 22 22 22 33 33 33 33 33 33 33
0210   33 33 33 33 33 34 44 44 44 44 44 44 44 44 44 44
0220   44 55 55 55 55 55 55 55 55 55 55 56 66 66 66 66
0230   66 66 66 66 66 67 77 77 77 77 77 77 77 77 77 78
0240   88 88 88 88 88 88 88 88 88 89 99 99
```

### 4.4 Checksum (RAW)

At the end of the binary block there is a 16-bit checksum. The checksum is calculated over all the package data. This checksum is calculated in the same way as IPv4 and UDP checksum.

## 4.5 Fragmentation (RAW)

The protocol may send the video data in fragments. When fragmentation is used and the package won't fit into an Ethernet frame (1500 bytes) the data will be sent in multiple packages. The first package has the user defined package fields while the subsequent packages have a static set of fields, package sequence and byte offset. Following depicts a fragmented package flow.

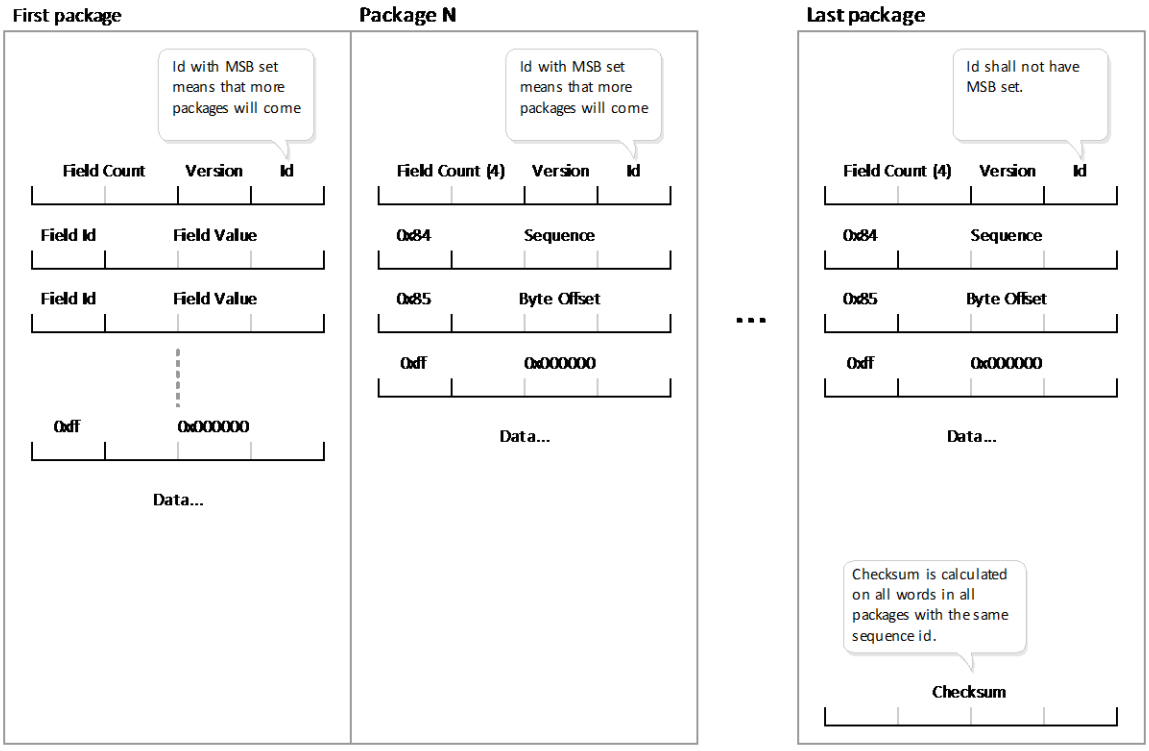


Figure 4: Fragmented packages

When the package fits into an Ethernet frame the package looks as before.

**Saab TransponderTech AB**

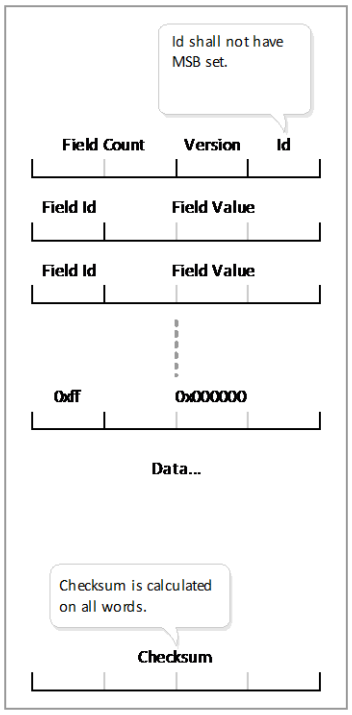


Figure 5: Single package

# 5 DIGITAL VIDEO COMPRESSION

## 5.1 Run Length Encoding

NOTE: This only applies to data with 8-bit samples.

This compression has the compression identity (COMPRESSION field) of one (1).

Run length encoding (RLE) encodes the data based on how many samples of the same value that are sent in sequence. If data block has 13 samples of value 63 in sequence this is encoded as 13 samples of 63.
This is encoded in a single byte where the MSB bit denotes if the byte is a counter or a sample. Counters are coded as (n-1) so a counter value of one (1) means that two (2) samples has been encoded. The above data block is encoded as the hexadecimal sequence `8c1f`. Since the most significant bit used the sample value is shifted down one bit and becomes a 7-bit value. When decoding an up-shift should be performed. If there are more than 128 samples of the same value in sequence the sequence is encoded into multiple resulting blocks. 200 values of 62 become `ff1fc71f`.

Here is an example of a hexadecimal sequence and the resulting encoded sequence.

```
01 02 03 03 02 00 00 00 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 04 3c 7f ff ff
```

Resulting encoded data block

```
00 83 01 82 00 90 02 1e 3f 81 7f
```