

Waveform Template

This appendix contains the Waveform Template which describes the contents of the Waveform Descriptor that is produced by the commands WF? DESC and WF? ALL. After the template come explanations of the construction of floating point numbers from bytes in the descriptor, followed by program fragments which show a method of performing the calculations.

Waveform Template

This template is the oscilloscope's response to a TMPL? query:

```
/00
000000          LECROY_2_3:  TEMPLATE
                8 66 111

;
; Explanation of the formats of waveforms and their descriptors on the
; LeCroy Digital Oscilloscopes,
;   Software Release 8.1.0, 98/09/29.
;
; A descriptor and/or a waveform consists of one or several logical data blocks
; whose formats are explained below.
; Usually, complete waveforms are read: at the minimum they consist of
;   the basic descriptor block WAVEDESC
;   a data array block.
; Some more complex waveforms, e.g. Extrema data or the results of a Fourier
; transform, may contain several data array blocks.
; When there are more blocks, they are in the following sequence:
;   the basic descriptor block WAVEDESC
;   the history text descriptor block USERTEXT (may or may not be present)
;   the time array block (for RIS and sequence acquisitions only)
;   data array block
;   auxiliary or second data array block
;
```

APPENDIX II: *Waveform Template*

; In the following explanation, every element of a block is described by a
; single line in the form

```
;
; <byte position>    <variable name>: <variable type> ; <comment>
;
; where
;
; <byte position> = position in bytes (decimal offset) of the variable,
;                  relative to the beginning of the block.
;
; <variable name> = name of the variable.
;
; <variable type> = string          up to 16-character name
;                               terminated with a null byte
;                               byte      08-bit signed data value
;                               word     16-bit signed data value
;                               long    32-bit signed data value
;                               float   32-bit IEEE floating point value
; with the format shown below
;
;                               31 30 .. 23   22 ... 0   bit position
;                               s  exponent  fraction
;                               where
;                               s = sign of the fraction
;                               exponent = 8 bit exponent e
;                               fraction = 23 bit fraction f
;                               and the final value is
;                                $(-1)^s * 2^{(e-127)} * 1.f$ 
;
;                               double   64-bit IEEE floating point value
;                               with the format shown below
;                               63 62 .. 52   51 ... 0   bit position
;                               s  exponent  fraction
;                               where
;                               s = sign of the fraction
;                               exponent = 11 bit exponent e
;                               fraction = 52 bit fraction f
;                               and the final value is
;                                $(-1)^s * 2^{(e-1023)} * 1.f$ 
;
;                               enum      enumerated value in the range 0 to N
;                               represented as a 16-bit data value.
;                               The list of values follows immediately.
;                               The integer is preceded by an _.
```

```
;          time_stamp          double precision floating point number,
;                               for the number of seconds and some bytes
;                               for minutes, hours, days, months and year.
;
;                               double   seconds      (0 to 59)
;                               byte     minutes      (0 to 59)
;                               byte     hours        (0 to 23)
;                               byte     days         (1 to 31)
;                               byte     months       (1 to 12)
;                               word     year         (0 to 16000)
;                               word     unused
;
;                               There are 16 bytes in a time field.
;          data                byte, word or float, depending on the
;                               read-out mode reflected by the WAVEDESC
;                               variable COMM_TYPE, modifiable via the
;                               remote command COMM_FORMAT.
;          text                arbitrary length text string
;                               (maximum 160)
;          unit_definition      a unit definition consists of a 48 character
;                               ASCII string terminated with a null byte
;                               for the unit name.
;
;=====
;
```

WAVEDESC: BLOCK

```
;
; Explanation of the wave descriptor block WAVEDESC;
;
;
< 0>          DESCRIPTOR_NAME: string ; the first 8 chars are always WAVEDESC
;
< 16>         TEMPLATE_NAME: string
;
< 32>         COMM_TYPE: enum          ; chosen by remote command COMM_FORMAT
;          _0      byte
;          _1      word
;          endenum
;
< 34>         COMM_ORDER: enum
;          _0      HIFIRST
;          _1      LOFIRST
;          endenum
;
;
```

APPENDIX II: *Waveform Template*

```
; The following variables of this basic wave descriptor block specify
; the block lengths of all blocks of which the entire waveform (as it is
; currently being read) is composed. If a block length is zero, this
; block is (currently) not present.
;
; Blocks and arrays that are present will be found in the same order
; as their descriptions below.
;
;BLOCKS :
;
< 36>          WAVE_DESCRIPTOR: long      ; length in bytes of block WAVEDESC
< 40>          USER_TEXT: long           ; length in bytes of block USERTEXT
< 44>          RES_DESC1: long           ;
;
;ARRAYS :
;
< 48>          TRIGTIME_ARRAY: long       ; length in bytes of TRIGTIME array
;
< 52>          RIS_TIME_ARRAY: long       ; length in bytes of RIS_TIME array
;
< 56>          RES_ARRAY1: long           ; an expansion entry is reserved
;
< 60>          WAVE_ARRAY_1: long         ; length in bytes of 1st simple
;                                         ; data array. In transmitted waveform,
;                                         ; represent the number of transmitted
;                                         ; bytes in accordance with the NP
;                                         ; parameter of the WFSU remote command
;                                         ; and the used format (see COMM_TYPE).
;
< 64>          WAVE_ARRAY_2: long         ; length in bytes of 2nd simple
;                                         ; data array
;
< 68>          RES_ARRAY2: long
< 72>          RES_ARRAY3: long           ; 2 expansion entries are reserved
;
; The following variables identify the instrument
;
< 76>          INSTRUMENT_NAME: string
;
< 92>          INSTRUMENT_NUMBER: long
;
< 96>          TRACE_LABEL: string        ; identifies the waveform.
;
<112>          RESERVED1: word
<114>          RESERVED2: word           ; 2 expansion entries
;
```

```
; The following variables describe the waveform and the time at
; which the waveform was generated.
;
<116>          WAVE_ARRAY_COUNT: long      ; number of data points in the data
                                           ; array. If there are two data
                                           ; arrays (FFT or Extrema), this number
                                           ; applies to each array separately.
;
<120>          PNTS_PER_SCREEN: long      ; nominal number of data points
                                           ; on the screen
;
<124>          FIRST_VALID_PNT: long      ; count of number of points to skip
                                           ; before first good point
                                           ; FIRST_VALID_POINT = 0
                                           ; for normal waveforms.
;
<128>          LAST_VALID_PNT: long      ; index of last good data point
                                           ; in record before padding (blanking)
                                           ; was started.
                                           ; LAST_VALID_POINT = WAVE_ARRAY_COUNT-1
                                           ; except for aborted sequence
                                           ; and rollmode acquisitions
;
<132>          FIRST_POINT: long          ; for input and output, indicates
                                           ; the offset relative to the
                                           ; beginning of the trace buffer.
                                           ; Value is the same as the FP parameter
                                           ; of the WFSU remote command.
;
<136>          SPARSING_FACTOR: long      ; for input and output, indicates
                                           ; the sparsing into the transmitted
                                           ; data block.
                                           ; Value is the same as the SP parameter
                                           ; of the WFSU remote command.
;
<140>          SEGMENT_INDEX: long        ; for input and output, indicates the
                                           ; index of the transmitted segment.
                                           ; Value is the same as the SN parameter
                                           ; of the WFSU remote command.
;
<144>          SUBARRAY_COUNT: long      ; for Sequence, acquired segment count,
                                           ; between 0 and NOM_SUBARRAY_COUNT
;
<148>          SWEEPS_PER_ACQ: long      ; for Average or Extrema,
                                           ; number of sweeps accumulated
                                           ; else 1
;
```

APPENDIX II: *Waveform Template*

```
<152>          POINTS_PER_PAIR: word      ; for Peak Detect waveforms (which
always                                                ; include data points in DATA_ARRAY_1
and                                                ; min/max pairs in DATA_ARRAY_2).
                                                ; Value is the number of data points
for                                                ; each min/max pair.
;
<154>          PAIR_OFFSET: word           ; for Peak Detect waveforms only
                                                ; Value is the number of data points by
                                                ; which the first min/max pair in
                                                ; DATA_ARRAY_2 is offset relative to
the                                                ; first data value in DATA_ARRAY_1.
;
<156>          VERTICAL_GAIN: float
;
<160>          VERTICAL_OFFSET: float     ; to get floating values from raw data
;                                                ; VERTICAL_GAIN * data -
VERTICAL_OFFSET
;
<164>          MAX_VALUE: float            ; maximum allowed value. It corresponds
                                                ; to the upper edge of the grid.
;
<168>          MIN_VALUE: float            ; minimum allowed value. It corresponds
                                                ; to the lower edge of the grid.
;
<172>          NOMINAL_BITS: word          ; a measure of the intrinsic precision
                                                ; of the observation: ADC data is 8 bit
                                                ;   averaged data is 10-12 bit, etc.
;
<174>          NOM_SUBARRAY_COUNT: word    ; for Sequence, nominal segment count
                                                ; else 1
;
<176>          HORIZ_INTERVAL: float       ; sampling interval for time domain
                                                ; waveforms
;
<180>          HORIZ_OFFSET: double        ; trigger offset for the first sweep of
                                                ; the trigger, seconds between the
                                                ; trigger and the first data point
;
<188>          PIXEL_OFFSET: double        ; needed to know how to display the
                                                ; waveform
;
<196>          VERTUNIT: unit_definition   ; units of the vertical axis
;
<244>          HORUNIT: unit_definition    ; units of the horizontal axis
;
```

```
<292>          HORIZ_UNCERTAINTY: float ; uncertainty from one acquisition to the
                                           ; next, of the horizontal offset in seconds
;
<296>          TRIGGER_TIME: time_stamp ; time of the trigger
;
<312>          ACQ_DURATION: float          ; duration of the acquisition (in sec)
                                           ; in multi-trigger waveforms.
                                           ; (e.g. sequence, RIS, or averaging)
;
<316>          RECORD_TYPE: enum
              _0      single_sweep
              _1      interleaved
              _2      histogram
              _3      graph
              _4      filter_coefficient
              _5      complex
              _6      extrema
              _7      sequence_obsolete
              _8      centered_RIS
              _9      peak_detect
              endenum
;
<318>          PROCESSING_DONE: enum
              _0      no_processing
              _1      fir_filter
              _2      interpolated
              _3      sparsed
              _4      autoscaled
              _5      no_result
              _6      rolling
              _7      cumulative
              endenum
;
<320>          RESERVED5: word              ; expansion entry
;
<322>          RIS_SWEEPS: word              ; for RIS, the number of sweeps
                                           ; else 1
;
; The following variables describe the basic acquisition
; conditions used when the waveform was acquired
;
```

APPENDIX II: *Waveform Template*

```
<324>      TIMEBASE: enum
        _0      1_ps/div
        _1      2_ps/div
        _2      5_ps/div
        _3      10_ps/div
        _4      20_ps/div
        _5      50_ps/div
        _6      100_ps/div
        _7      200_ps/div
        _8      500_ps/div
        _9      1_ns/div
        _10     2_ns/div
        _11     5_ns/div
        _12     10_ns/div
        _13     20_ns/div
        _14     50_ns/div
        _15     100_ns/div
        _16     200_ns/div
        _17     500_ns/div
        _18     1_us/div
        _19     2_us/div
        _20     5_us/div
        _21     10_us/div
        _22     20_us/div
        _23     50_us/div
        _24     100_us/div
        _25     200_us/div
        _26     500_us/div
        _27     1_ms/div
        _28     2_ms/div
        _29     5_ms/div
        _30     10_ms/div
        _31     20_ms/div
        _32     50_ms/div
        _33     100_ms/div
        _34     200_ms/div
        _35     500_ms/div
        _36     1_s/div
        _37     2_s/div
        _38     5_s/div
        _39     10_s/div
        _40     20_s/div
        _41     50_s/div
        _42     100_s/div
        _43     200_s/div
        _44     500_s/div
        _45     1_ks/div
        _46     2_ks/div
        _47     5_ks/div
        _100    EXTERNAL
```

```
endenum
;
<326>      VERT_COUPLING: enum
          _0      DC_50_Ohms
          _1      ground
          _2      DC_1MOhm
          _3      ground
          _4      AC,_1MOhm
endenum
;
<328>      PROBE_ATT: float
;
<332>      FIXED_VERT_GAIN: enum
          _0      1_uV/div
          _1      2_uV/div
          _2      5_uV/div
          _3      10_uV/div
          _4      20_uV/div
          _5      50_uV/div
          _6      100_uV/div
          _7      200_uV/div
          _8      500_uV/div
          _9      1_mV/div
          _10     2_mV/div
          _11     5_mV/div
          _12     10_mV/div
          _13     20_mV/div
          _14     50_mV/div
          _15     100_mV/div
          _16     200_mV/div
          _17     500_mV/div
          _18     1_V/div
          _19     2_V/div
          _20     5_V/div
          _21     10_V/div
          _22     20_V/div
          _23     50_V/div
          _24     100_V/div
          _25     200_V/div
          _26     500_V/div
          _27     1_kV/div
endenum
;
```

APPENDIX II: *Waveform Template*

```
<334>          BANDWIDTH_LIMIT: enum
              _0          off
              _1          on
              endenum
;
<336>          VERTICAL_VERNIER: float
;
<340>          ACQ_VERT_OFFSET: float
;
<344>          WAVE_SOURCE: enum
              _0          CHANNEL_1
              _1          CHANNEL_2
              _2          CHANNEL_3
              _3          CHANNEL_4
              _9          UNKNOWN
              endenum
;
/00          ENDBLOCK
;
;=====
;
USERTEXT: BLOCK
;
; Explanation of the descriptor block USERTEXT at most 160 bytes long.
;
;
< 0>          TEXT: text          ; a list of ASCII characters
;
/00          ENDBLOCK
;
;=====
;
TRIGTIME: ARRAY
;
; Explanation of the trigger time array TRIGTIME.
; This optional time array is only present with SEQNCE waveforms.
; The following data block is repeated for each segment which makes up
; the acquired sequence record.
;
< 0>          TRIGGER_TIME: double      ; for sequence acquisitions,
                                          ; time in seconds from first
                                          ; trigger to this one
;
< 8>          TRIGGER_OFFSET: double    ; the trigger offset is in seconds
                                          ; from trigger to zeroth data point
;
/00          ENDARRAY
;
;=====
;
```

```
RISTIME: ARRAY
;
; Explanation of the random-interleaved-sampling (RIS) time array RISTIME.
; This optional time array is only present with RIS waveforms.
; This data block is repeated for each sweep which makes up the RIS record
;
< 0>          RIS_OFFSET: double          ; seconds from trigger to zeroth
                                           ; point of segment

;
/00          ENDARRAY
;
;=====
;
DATA_ARRAY_1: ARRAY
;
; Explanation of the data array DATA_ARRAY_1.
; This main data array is always present. It is the only data array for
; most waveforms.
; The data item is repeated for each acquired or computed data point
; of the first data array of any waveform.
;
< 0>          MEASUREMENT: data            ; the actual format of a data is
                                           ; given in the WAVEDESC descriptor
                                           ; by the COMM_TYPE variable.

;
/00          ENDARRAY
;
;=====
;
DATA_ARRAY_2: ARRAY
;
; Explanation of the data array DATA_ARRAY_2.
; This is an optional secondary data array for special types of waveforms:
;   Complex FFT      imaginary part      (real part in DATA_ARRAY_1)
;   Extrema          floor trace          (roof trace in DATA_ARRAY_1)
;   Peak Detect      min/max pairs        (data values in DATA_ARRAY_1)
; In the first 2 cases, there is exactly one data item in DATA_ARRAY_2 for
; each data item in DATA_ARRAY_1.
; In Peak Detect waveforms, there may be fewer data values in DATA_ARRAY_2,
; as described by the variable POINTS_PER_PAIR.
;
< 0>          MEASUREMENT: data            ; the actual format of a data is
                                           ; given in the WAVEDESC descriptor
                                           ; by the COMM_TYPE variable.

;
/00          ENDARRAY
;
;=====
;
SIMPLE: ARRAY
```

APPENDIX II: *Waveform Template*

```
;
; Explanation of the data array SIMPLE.
; This data array is identical to DATA_ARRAY_1. SIMPLE is an accepted
; alias name for DATA_ARRAY_1.
;
< 0>          MEASUREMENT: data          ; the actual format of a data is
                                           ; given in the WAVEDESC descriptor
                                           ; by the COMM_TYPE variable.

;
/00          ENDARRAY
;
;=====
;
DUAL: ARRAY
;
; Explanation of the DUAL array.
; This data array is identical to DATA_ARRAY_1, followed by DATA_ARRAY_2.
; DUAL is an accepted alias name for the combined arrays DATA_ARRAY_1 and
; DATA_ARRAY_2 (e.g. real and imaginary parts of an FFT).
;
< 0>          MEASUREMENT_1: data        ; data in DATA_ARRAY_1.
;
< 0>          MEASUREMENT_2: data        ; data in DATA_ARRAY_2.
;
/00          ENDARRAY
;
;
00          ENDTEMPLATE
```

DECODING FLOATING POINT NUMBERS

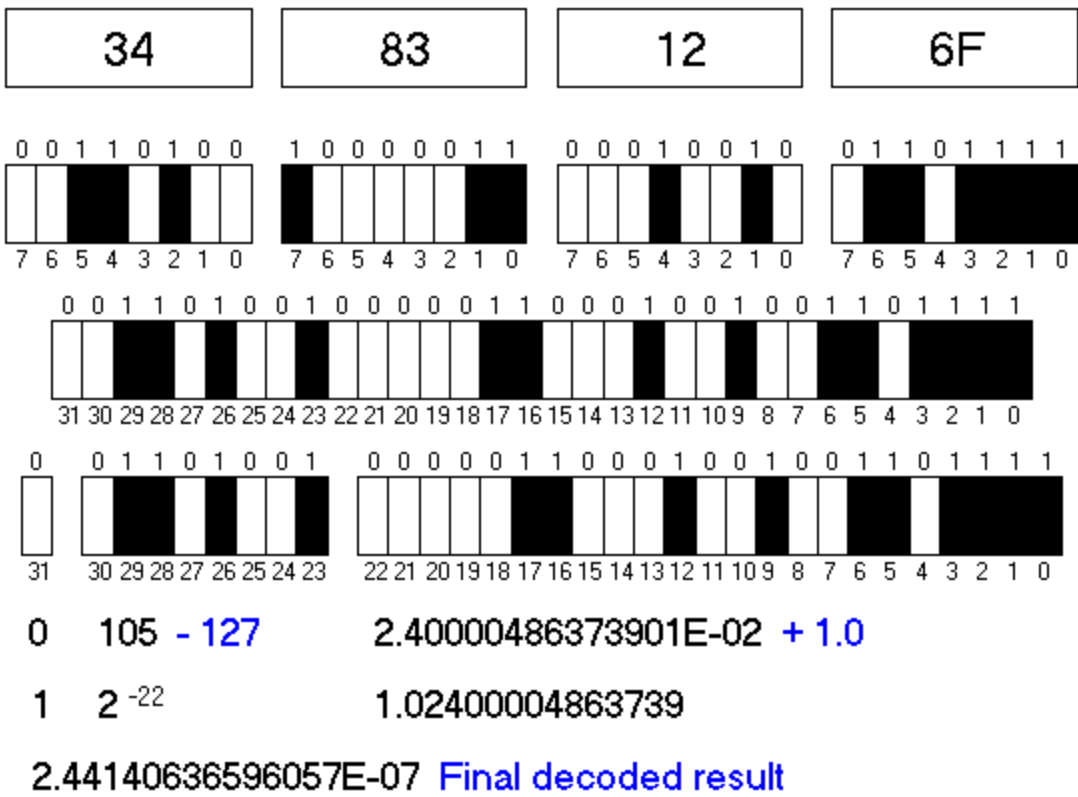
Single precision values are held in four bytes. If these are arranged in decreasing order of value we get the following bits – bit 31, bit 30, bit 29, bit 28 bit 3, bit 2, bit 1, bit 0.

We must remember that if the byte order command CORD has been set for low byte first, the bytes as received in a waveform descriptor will be received in the reverse order. But within a byte, the bits keep their order, highest at the left as expected.

From these bits we are to construct three numbers that are to be multiplied together: S x E x F. These in turn are constructed as follows:

$S = (-1)^s$ $E = 2^{(e - 127)}$ $F = 1 + f$

and it is **s**, **e**, and **f** that are calculated directly from the 32 bits. The diagram below illustrates the calculation of the vertical gain example of Chapter 4.



APPENDIX II: *Waveform Template*

The bits are to be segregated as follows, in a way that does not follow the byte boundaries:

31	30, 29 . . . 24, 23	22, 21 . . . 2, 1, 0
sign bit	exponent bits	fractional bits
		0.5, 0.25, 0.125 . . .

The sign bit **s** is 1 for a negative number and 0 for a positive number, so it is easy to construct the sign from this:

$$S = (-1)^s$$

The 8 exponent bits have the following values:

bit 23 is worth 1, bit 24 is worth 2 . . . bit 29 \rightarrow 64, bit 30 \rightarrow 128, so the resulting number can range from 0 to $2^8 - 1$, which is 255.

127 is then subtracted from this value **e** creating a range from -127 to $+128$. This is then used as an exponent to raise two to a power that is 2^e , to create a value **E**.

Then we have to create the multiplying number. The values of the 23 bits are as follows:

Bit 22 is worth 0.5, 21 is worth 0.25, 20 is worth 0.125, 19 is worth 0.0625

When all the bits are added together, we obtain a positive number **f** that can be very close to one, differing from it only by the value of the smallest bit, if all the bits are ones. (Generally the value will be much less than one.) Then we add one to the result, obtaining $1 + f = F$. The use of the added one extends the dynamic range of the data.

Another way of calculating **f** is to take the 23-bit number at face value, and divide it by 2^{24} .

Finally we multiply together the sign, the value **E**, and the value **F** to create the final result:

$$\text{Result} = (-1)^s \times 2^{(e-127)} \times (1 + f) = S \times E \times F$$

Example

In Chapter 4, one of the examples, Vertical Gain, states that the floating point number 34 83 12 6F leads to the decimal value 2.44141E-07. Let's see how this is done.

The bytes 34 83 12 and 6F can be written in binary as follows:

0011 0100 1000 0011 0001 0010 0110 1111.

This string of bits is to be split up as follows:

0 01101001 00000110001001001101111.

The first bit, 0, makes the sign of the number **S**, using the formula $S = (-1)^s = 1$.

The next eight bits make the exponent **e** as follows:

$$0 \times 128 + 1 \times 64 + 1 \times 32 + 0 \times 16 + 1 \times 8 + 0 \times 4 + 0 \times 2 + 1 \times 1 = 105, \text{ from which we subtract } 127, \\ \text{giving } -22.$$

So the factor **E** is $2^{(e-127)} = 2^{-22}$, which is 2.3842E-7.

Finally, we need to make the multiplier **F**. The remaining bits are given the values 0.5, 0.25, 0.125, 0.0625, 0.03125, etc. The first bits that are not zero are the 6th and 7th bits, whose values are 0.015625 and 0.078125, respectively. To get a rough value, we will take just these two bits, since the next three are zero, giving 0.0234375. We have to add 1 to this, giving 1.023 as a rough value for **F**.

The final result is therefore $S \times E \times F = 1 \times 2.3842\text{E-}7 \times 1.023 = 2.439$, which is a little smaller than the correct value because we did not use all the bits to calculate the value of **F**.

Double precision values are held in eight bytes. If these are arranged in decreasing order of value we get the following bits:

$$63, 62, 61, 60, \dots, 3, 2, 1, 0.$$

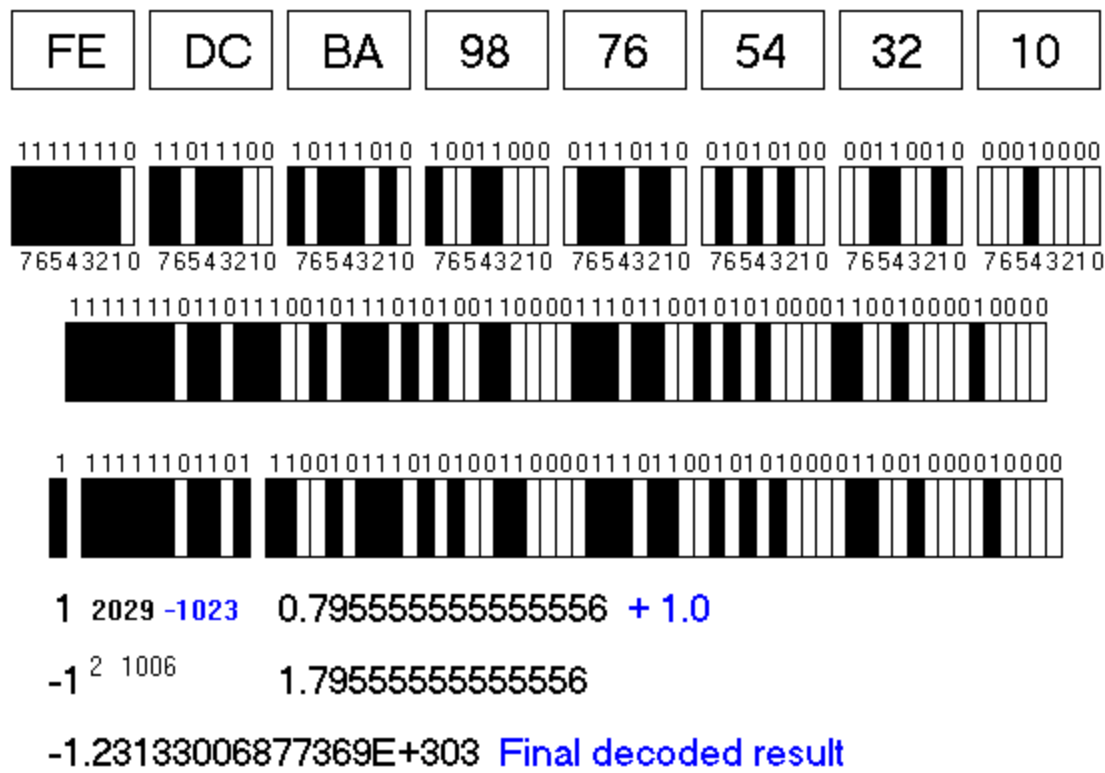
We must remember that if the byte order command CORD has been set for low byte first, the bytes as received in a waveform descriptor will be received in the reverse order. But within a byte, the bits keep their order: highest at the left, as expected.

From these bits we are to construct three numbers that are to be multiplied together: $S \times E \times F$. These in turn are constructed as follows:

$$S = (-1)^s \quad E = 2^{(e - 1023)} \quad F = 1 + f$$

and it is **s**, **e**, and **f** that are calculated directly from the 32 bits. The following diagram illustrates the calculation of an example.

APPENDIX II: *Waveform Template*



The bits are to be segregated as follows, in a way that does not follow the byte boundaries –

63	62, 61 53, 52	51, 50 2, 1, 0
sign	11 exponent bits	52 fractional bits
bit		0.5, 0.25, 0.125 . . .

The sign bit is 1 for a negative number and 0 for a positive number, so it is easy to construct the sign from this: $S = (-1)^s$.

The 11 exponent bits have the following values:

$$52 \rightarrow 1, 53 \rightarrow 2 \dots 61 \rightarrow 512, 62 \rightarrow 1024$$

so the resulting number can range from 0 to $2^{12} - 1$, which is 2047. 1023 is then subtracted from this value, creating a range from -1023 to +1024. This is then used as a power of two to create a value **E**.

Then we have to create the multiplying number. The values of the 52 bits are as follows:

$$51 \rightarrow 0.5, 50 \rightarrow 0.25, 49 \rightarrow 0.125, 48 \rightarrow 0.0625 \dots$$

When all the bits are added together, we obtain a positive number **f** that can be very close to one, differing from it only by the value of the smallest bit, if all the bits are ones. Generally the value will be much less than one. Then we add one to the result, obtaining $1 + f = F$. The use of the added one extends the dynamic range of the data.

Alternatively, we can take the 52-bit number at face value, and divide it by 2^{53}

Finally we multiply together the sign, the value **E**, and the value **F**, to create the final result:

$$\text{Result} = S \times E \times F$$

APPENDIX II: *Waveform Template*

HOW TO CONSTRUCT A FLOATING POINT NUMBER FROM FOUR BYTES

```
' Routine to construct a floating point number from four bytes.

Function GetFloat(DescPoint as Integer)

' DescPoint is the address of the byte in the waveform descriptor
' where the data begin.
' The data are assumed to be in an array called Desc (0 to 350).

' For example, to calculate VERTICAL_GAIN, DescPoint = 156.

' Constants needed by GetFloat
Mult2 = 1 / 128
Mult3 = Mult2 / 256
Mult4 = Mult3 / 256

' Comm_Order is the variable which provides information
' about the order of the bytes in the descriptor and.
' in the waveform data. Comm_Order is the byte at position
' 34 in the descriptor.

' Set ByteOrd = 1 when Comm_Order = 0 for high byte first.
' Set ByteOrd = -1 when Comm_Order = 1 for low byte first.
' Set ByteOrd3 = 3 * Comm_Order.
'


---


ByteOrd = 1 - 2 * Comm_Order
ByteOrd3 = 3 * Comm_Order


---



FByte = ByteOrd3                                ' Sign started

FDigit = Desc(DescPoint + FByte)
FSign = (FDigit And 128) \ 128
FSign = 1 - 2 * FSign                            ' Sign completed


---



FExponent = FDigit And 127                        ' Exponent started
FExponent = 2 * FExponent

FByte = ByteOrd3 + ByteOrd
FDigit = Desc(DescPoint + FByte)
FExpBit = FDigit And 128

If FExpBit = 128 Then FExpBit = 1

FExponent = FExponent + FExpBit - 127            ' Exponent completed
```

```
'  
_____  
FFraction = CDb1(FDigit And 127)          ' Fraction started  
FFraction = FFraction * Mult2  
  
FByte = ByteOrd3 + 2 * ByteOrd  
FDigit = Desc(DescPoint + FByte)  
FFraction = FFraction + CDb1(FDigit) * Mult3  
  
FByte = ByteOrd3 + 3 * ByteOrd  
FDigit = Desc(DescPoint + FByte)  
FFraction = FFraction + CDb1(FDigit) * Mult4    ' Fraction completed  
  
_____  
FVariable = 2 ^ FExponent  
GetFloat = FVariable * FSign * (1 + FFraction)  ' Conversion completed  
  
End  
  
' End of GetFloat _____
```

This example is taken from program TranWM, which is included in the CD-ROM shipped with the instrument. The same calculations are available in VBA / Excel in program TraceLook.

APPENDIX II: *Waveform Template*

HOW TO CONSTRUCT A FLOATING POINT NUMBER FROM FOUR BYTES

' Routine to construct a double precision floating point number from eight bytes.

Function GetDoubleFloat (DescPoint as Integer)

' DescPoint is the address of the byte in the waveform descriptor
' where the data begin.
' The data are assumed to be in an array called Desc (0 to 350).

' For example, to calculate HORIZONTAL_OFFSET, DescPoint = 180.

' Constants needed by GetDoubleFloat
DMult2 = 1 / 16
DMult3 = DMult2 / 256

' Comm_Order is the variable which provides information
' about the order of the bytes in the descriptor and.
' in the waveform data. Comm_Order is the byte at position
' 34 in the descriptor.

' Set ByteOrd = 1 when Comm_Order = 0 for high byte first.
' Set ByteOrd = -1 when Comm_Order = 1 for low byte first.
' Set ByteOrd7 = 7 * Comm_Order.
,

ByteOrd = 1 - 2 * Comm_Order
ByteOrd7 = 7 * Comm_Order

DMult3 = DMult2 / 256

FByte = ByteOrd7
FDigit = Desc(DescPoint + FByte)
FSign = (FDigit And 128) \ 128
FSign = 1 - 2 * FSign

' Sign started

' Sign completed

FExponent = FDigit And 127
FExponent = 16 * FExponent

' Exponent started

FByte = ByteOrd7 + ByteOrd
FDigit = Desc(DescPoint + FByte)
FExponent = (FExponent + Cdbl((FDigit And 240) \ 16)) - 1023
' Exponent completed

FFraction = Cdbl((FDigit And 15)) * DMult2

' Fraction started

```
For I = 2 To
FByte = ByteOrd7 + I * ByteOrd
FDigit = Desc(DescPoint + FByte)
FFraction = FFraction + CDb1(FDigit) * DMult3
DMult3 = DMult3 / 256
Next I                                     ' Fraction completed
'
FVariable = 2 ^ FExponent

GetDoubleFloat = FVariable * FSign * (1 + FFraction)

End

' End of GetDoubleFloat _____
```

This example is taken from Visual Basic program TranWM, which is included in the CD-ROM shipped with the WaveMaster. The same calculations are available in VBA / Excel in program TraceLook.

\$\$\$