|  |  |
| --- | --- |
| **ITU – Telecommunications Standardization Sector**  STUDY GROUP 21 Question C/16 (ex-Q6/16)  **Video Coding Experts Group (VCEG)**  75th Meeting: 2-8 November 2024, Kemer, TR | Document VCEG-BW05-v1 |

|  |  |  |  |
| --- | --- | --- | --- |
| Question: | C/16 SG21 (VCEG) | | |
| Source: | **Jongmo Sung, Byeongho Jo, Sooyoung Park, Seungkwon Beack (ETRI)** | Email: | {jmseong, bhjo, sooyoung, skbeack}@etri.re.kr |
| Title: | **ETRI response to Call for Proposals for Recommendation H.BWC on the coding of biomedical waveform data** | | |
| Purpose: | Proposal | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This document presents ETRI’s response to the Call for Proposals (CfP) for Recommendation H.BWC on biomedical waveform coding.

# Introduction

This document provides technical descriptions, evaluation results, and complexity characteristics of the codec submitted by ETRI in response to the H.BWC Call for Proposals [1]. The proposed codec combines block-wise predictive coding with transform-based residual coding, as implemented based on Fraunhofer HHI’s Call for Evidence (CfE) response [2]. For each input block, a predicted block is estimated from the previously reconstructed blocks. The residual block, representing the difference between the input and predicted blocks, is then transformed, quantized, and entropy-coded.

We conducted evaluations of the proposed codec on electrocardiogram (ECG) datasets.

# Technical descriptions

Figure 2‑1 shows the encoding block diagram of the proposed codec.

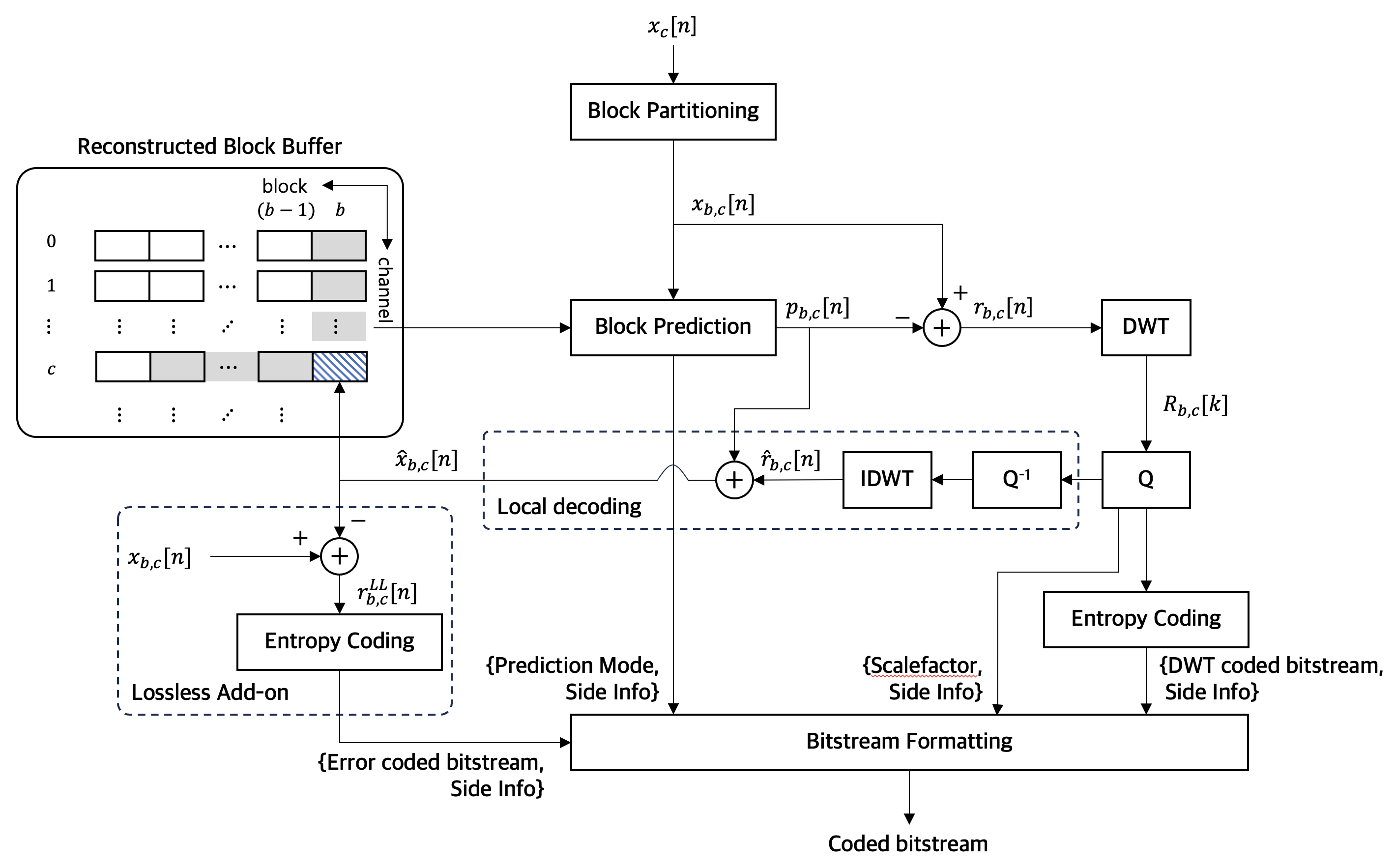


Figure 2‑1. Encoding block diagram

The multi-channel input signal is partitioned into blocks, serving as the basic coding units. For each block, a prediction process is performed based on previously reconstructed blocks. The difference signal between the original block and the predicted signal is transformed into wavelet coefficients using the discrete wavelet transform (DWT), then quantized and entropy-coded to produce a transform-coded bitstream. For subsequent block prediction, the quantized transform coefficients are locally decoded and added to the predicted block to reconstruct the quantized current block.

For lossless compression, the difference signal between the block reconstructed from the lossy compression and the original block is entropy-coded to generate an error-coded bitstream. The entropy-coded bitstream(s) along with the side information are formatted according to the bitstream syntax, resulting in the final bitstream output.

## Block partitioning

Biomedical waveform data generally consists of a bundle of one-dimensional signals, and the -th sample at the -th channel in the input signal can be represented as:

,

where is the number of channels and is the number of samples in each channel. In the proposed codec, the input signal is processed in fixed-length blocks of (= 512) samples without overlap between blocks. The signal block for each channel is defined as:

,

where is the number of blocks in a single channel. The input signal is processed sequentially in block-channel order.

## Block prediction

A prediction for the input block signal is performed by generating the predicted block signal from the previously reconstructed blocks using multiple prediction candidates. In the proposed codec, six prediction modes are used: zero prediction (ZP), DC prediction (DCP), two types of slope predictions (SPs), inter-channel prediction (ICP), and block copy prediction (BCP). The DCP, SPs, and ICP are performed in the original signal domain, while the BCP is performed in the linear predictive coding (LPC) filtered domain.

### Zero prediction

In the ZP mode, all samples of the predicted block are set to zero as:

.

### DC prediction

In DCP mode, all samples of the predicted block are set to the mean value of a portion of the previously reconstructed block:

,

,

where are the reconstructed block samples. Half of the previously reconstructed block is used to calculate the DC value.

### Slope predictions

In SP modes, the slope is calculated using two adjacent samples in the previously reconstructed block. Each sample of the predicted block is then extrapolated along a straight line derived from the scaled slope.

.

The following two scaling values are applied to each SP mode:

.

### Inter-channel prediction

In ICP mode, the most similar channel block to the current block for is searched within the ICP reference blocks, which are the previously reconstructed blocks for . Rather than directly copying one of the reference blocks, a linear mapping between each channel pair is applied. The linear mapping parameters, and are derived from the previously reconstructed block channel pair of and , where. The best channel index, is determined by minimizing the mean squared error (MSE) between the current block, and the linearly-mapped block, . The predicted block is computed as:

,

where and are the linear mapping parameters for the best reference channel. The best channel index, is transmitted to the decoder.

### Block copy prediction

In the BCP mode, the most similar block to the current block of is searched in the BCP reference blocks, which are the previously reconstructed blocks of . Unlike other prediction modes, BCP operates in the LPC-filtered domain. Figure 2‑2 illustrates the proposed BCP mode

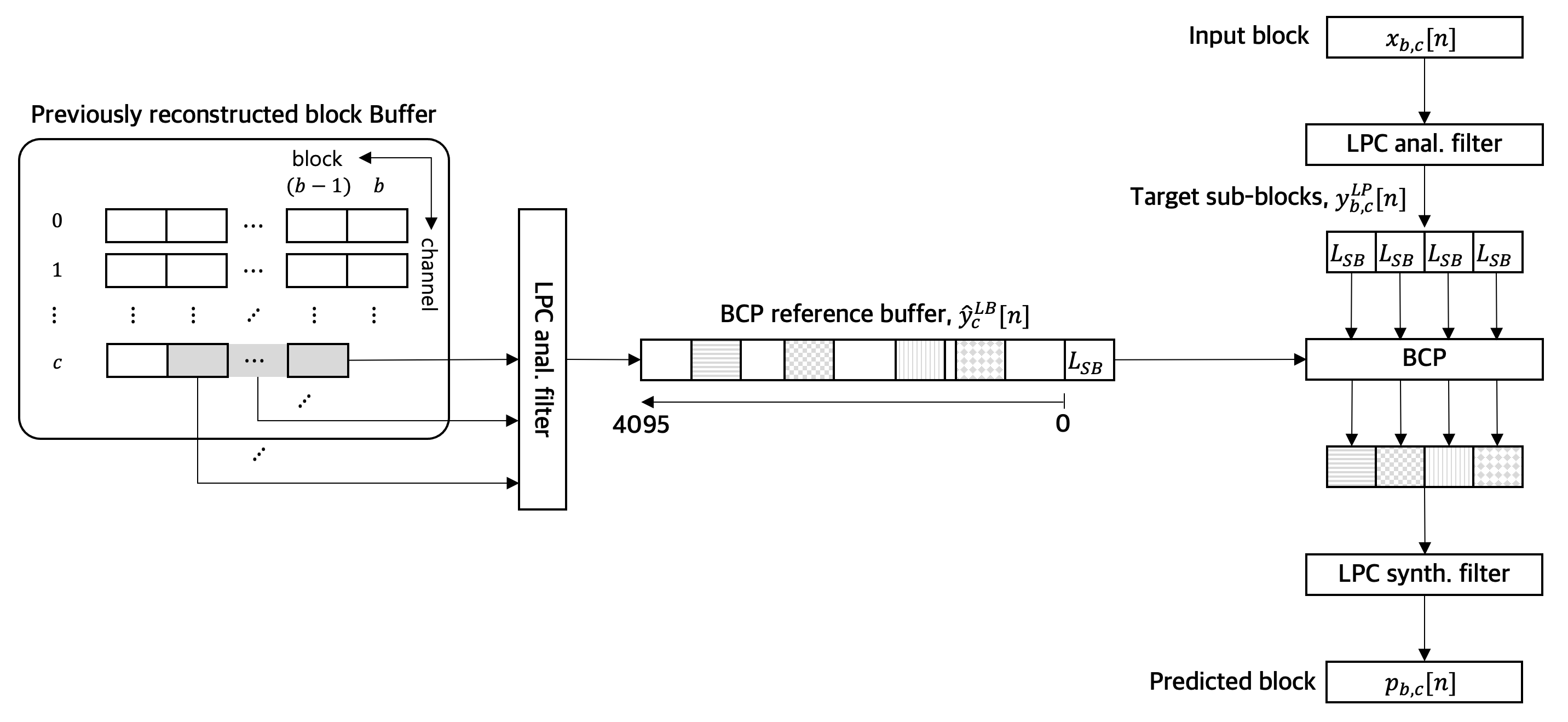


Figure 2‑2. Block copy prediction in the LPC filtered domain

The 4th-order LPC analysis is used to compute LPC filtered block signals for the reference blocks,

,

where are LPC coefficients for the reference block. The LPC filtered reference block, is stored in the BCP reference buffer, ,

.

Using the LPC coefficients of the ()-th reference block, the LPC filtered signal of current block, is obtained as follows:

.

The positional lag search is performed on sub-block basis, which is each of the (= 4) sub-blocks having (= 128) samples of the current LPC filtered block. A search range is [0, 4095], so the 9 reference blocks are used for BCP. The best positional lag for each sub-block, is determined by minimizing the MSE between a target sub-block and LPC filtered reference sub-blocks as follows:

,

,

.

Using the best positional lag for each sub-block, the predicted block in the LPC filtered domain can be constructed as follows:

​.

The final predicted block is obtained by applying LPC synthesis filtering to the predicted block in the LPC filtered domain:

.

Additionally, the half-pel interpolation is also applied using the same interpolation filter as [2]. The best positional lags for sub-blocks, and the half-pel flag are transmitted to the decoder.

## Transform coding

For the input block , the residual block, is computed by subtracting the prediction block, :

.

Before transforming the residual block, it is divided into 8 sub-blocks of 64 samples, and the sub-block with maximum energy is down-scaled by a factor of 1.1. The index of this sub-block is transmitted to the decoder. The scaled residual block, denoted as , is then processed using DWT, followed by quantization and entropy coding. At the decoder, the sub-block indicated by the signaled index is up-scaled.

### Transform and quantization

The one-dimensional Bior4.4 (Biorthogonal Wavelet Transform) was adopted to transform the scaled residual block, . The decomposition level of the Bior4.4 transform is set to five. The transformed coefficients, of the scaled residual block are decomposed into six sub-bands, consisting of one approximation coefficients, cA5, and five detail coefficients set, {cD5, cD4, cD3, cD2, cD1}. The sub-band decomposition is depicted in Figure 2‑3.

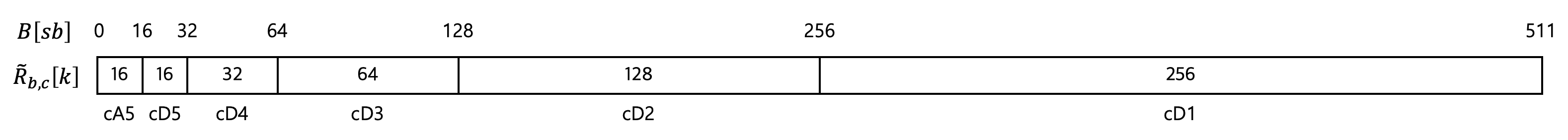


Figure 2‑3. Sub-band boundaries of Bior4.4 (level = 5)

The transformed coefficients, are scaled using a factor and then scalar-quantized using rounding:

.

Depending on the scale factor, for the transform coefficients, a step size of the quantizer is varied. The scale factor is common for every block in each file and is signaled to the decoder. The quantized transform indices, are entropy-coded and transmitted. At the decoder side, the quantized transform coefficients are recovered by multiplying the quantized transform indices by the signaled scale factor.

### Entropy coding of transform coefficient indices

The quantized transform indices, are decomposed into their absolute values and signs. Then, the absolute values and signs for the non-zero values are entropy-coded using arithmetic coding.

We designed 24 context models for the entropy-coding of absolute values based on magnitude and sub-band. As shown in Figure 2‑3, the absolute values of the quantized transform indices are grouped into six sub-bands. A 6-dimensional vector consisting of the maximum absolute value for each sub-band is calculated as follows:

,

where is the boundary index of the -th sub-band. Based on , one of four context levels is selected using a pre-defined mapping function, :

.

The selected context information is transmitted to the decoder. After the first context level is selected, a per-sub-band context model is determined using another mapping function,

.

Using the selected probability model depending on the sub-band, the absolute values in the sub-band are encoded using arithmetic coding. The non-zero signs are coded as 1 bit per quantized index. The entropy-coded bitstream of absolute values and non-zero sign bitstream are transmitted to the decoder along with their bitstream length.

At the decoder, the received bitstream is parsed into the coded bitstreams of absolute values with the selected context model and non-zero signs. The probability model for entropy decoding is retrieved based on the selected context model and sub-band index. Utilizing the probability model, the absolute values of the quantized transform indices are recovered. The combination of the absolute values and their signs produces the quantized transform coefficients.

## Lossless add-on

For lossless compression (WP0), the last-mile error block between the original input block, and the locally reconstructed block, in the lossy part is computed:

.

A BPS of 3.0 is set for the lossy codec in the ECG dataset. The last-mile error block is entropy-coded to produce an error-coded bitstream along with the corresponding side information. A pre-defined offset is added to the samples in the last-mile error block to make them non-negative integer. The resulting non-negative error block is arithmetically encoded using a single probability model. The length of error-coded bitstream is also transmitted as side information to the decoder.

## Bitstream formatting

All coded bitstreams and side information are formatted into overall bitstream. The syntax of the bitstream is shown in Table 2‑1 to Table 2‑10.

Table 2‑1. Syntax of BwcDecoderConfig()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| BwcDecoderConfig()  {  **signalLength;**  **numFrames;**  **numChannels;**  **dataType;** a  **workingPoint;** b  decoderConfig()  } | **32**  **16**  **16**  **2**  **4** | **uimsbf**  **uimsbf**  **uimsbf**  **uimsbf**  **uimsbf** |
| a dataType: EEG=0, ECG=1, EMG=2   |  | | --- | | b workingPoint: integer number defined in VCEG-BT07-v1, 5. | |  | | | |

Table 2‑2. Syntax of decoderConfig()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| decoderConfig()  {  **DWTScaleFactor;**  **coefficientLengthBitLength;**  sourceData ()  } | **64**  **4** | **uimsbf**  **uimsbf** |

Table 2‑3. Syntax of sourceData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| sourceData()  {  for (i=0 i<numFrames; i++){  for (j=0; j<numChannels; j++){  modeData()  }  }  } |  |  |

Table 2‑4. Syntax of modeData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| modeData(workingPoint)  {  **probabilityModelIndex;**  **modeIndex;**  switch (modeIndex){  case: ID\_ZP\_MODE a  zpData(workingPoint)  case: ID\_DCP\_MODE a  dcpData(workingPoint)  case: ID\_HSP\_MODE a  zpData(workingPoint)  case: ID\_QSP\_MODE a  qspData(workingPoint)  case: ID\_LPBCP\_MODE a  lpBcpData(workingPoint)  case: ID\_ICP\_MODE a  icpData(workingPoint)  }  } | **2**  **4** | **uimsbf**  **uimsbf** |
| a ID\_ZP\_MODE = 0, ID\_DCP\_MODE = 1, ID\_HSP\_MODE = 2, ID\_QSP\_MODE = 3, : ID\_LPBCP\_MODE = 6, ID\_ICP\_MODE = 7 | | |

Table 2‑5. Syntax of zpData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| zpData(workingPoint)  {  **scaledSegmentIndex;**  **nonzeroCoefficientBitLength;**  **signCoefficient;**  **absoluteCoefficientBitLength;**  **absoluteCoefficient;**  if (workingPoint == 0){  **residualBitLength;**  **residualOfCoefficient;**  **}**  } | **3**  **9**  **nonzeroCoefficientLength**  **coefficientLengthBitLength** a  **absoluteCoefficientBitLength**  **11**  **residualBitLength** | **uimsbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf** |
| a coefficientLengthBitLength is defined in syntax of decoderConfig() | | |

Table 2‑6. Syntax of dcpData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| dcpData(workingPoint)  {  **scaledSegmentIndex;**  **nonzeroCoefficientBitLength;**  **signCoefficient;**  **absoluteCoefficientBitLength;**  **absoluteCoefficient;**  if (workingPoint == 0){  **residualBitLength;**  **residualOfCoefficient;**  **}**  } | **3**  **9**  **nonzeroCoefficientLength**  **coefficientLengthBitLength** a  **absoluteCoefficientBitLength**  **11**  **residualBitLength** | **uimsbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf** |
| a coefficientLengthBitLength is defined in syntax of decoderConfig() | | |

Table 2‑7. Syntax of hspData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| hspData(workingPoint)  {  **scaledSegmentIndex;**  **nonzeroCoefficientBitLength;**  **signCoefficient;**  **absoluteCoefficientBitLength;**  **absoluteCoefficient;**  if (workingPoint == 0){  **residualBitLength;**  **residualOfCoefficient;**  **}**  } | **3**  **9**  **nonzeroCoefficientLength**  **coefficientLengthBitLength** a  **absoluteCoefficientBitLength**  **11**  **residualBitLength** | **uimsbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf** |
| a coefficientLengthBitLength is defined in syntax of decoderConfig() | | |

Table 2‑8. Syntax of qspData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| qspData(workingPoint)  {  **scaledSegmentIndex;**  **nonzeroCoefficientBitLength;**  **signCoefficient;**  **absoluteCoefficientBitLength;**  **absoluteCoefficient;**  if (workingPoint == 0){  **residualBitLength;**  **residualOfCoefficient;**  **}**  } | **3**  **9**  **nonzeroCoefficientLength**  **coefficientLengthBitLength** a  **absoluteCoefficientBitLength**  **11**  **residualBitLength** | **uimsbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf** |
| a coefficientLengthBitLength is defined in syntax of decoderConfig() | | |

Table 2‑9. Syntax of lpDcpData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| lpDcpData(workingPoint)  {  **scaledSegmentIndex;**  for (i=0; i<numSubFrame; i++){  **PredictionLag;**  **HalfPelFlag**  }  **nonzeroCoefficientBitLength;**  **signCoefficient;**  **absoluteCoefficientBitLength;**  **absoluteCoefficient;**  if (workingPoint == 0){  **residualBitLength;**  **residualOfCoefficient;**  }  } | **3**  **12**  **1**  **9**  **nonzeroCoefficientLength**  **coefficientLengthBitLength** a  **absoluteCoefficientBitLength**  **11**  **residualBitLength** | **uimsbf**  **uimsbf**  **uimsbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf** |
| a coefficientLengthBitLength is defined in syntax of decoderConfig() | | |

Table 2‑10. Syntax of icpData()

|  |  |  |
| --- | --- | --- |
| Syntax | No. of bits | Mnemonic |
| icpData(workingPoint, numChannels)  {  if (numChannels > 2){  **channelCopyIndex;**  }  **scaledSegmentIndex;**  **nonzeroCoefficientBitLength;**  **signCoefficient;**  **absoluteCoefficientBitLength;**  **absoluteCoefficient;**  if (workingPoint == 0){  **residualBitLength;**  **residualOfCoefficient;**  }  } | **3**  **3**  **9**  **nonzeroCoefficientLength**  **coefficientLengthBitLength** a  **absoluteCoefficientBitLength**  **11**  **residualBitLength** | **uimsbf**  **uimsbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf**  **uimsbf**  **vlclbf** |
| a coefficientLengthBitLength is defined in syntax of decoderConfig() | | |

# Evaluation test results

We evaluated the proposed codec for MIT ECG datasets using the three metrics: PRD (Percentage Root Mean Square Distortion), CPRD (Channel-normalized PRD) and PSNR (Peak Signal-to-Noise Ratio), as defined in [1]. MIT ECG dataset contains 48 files consisting of two-channel signals sampled at 360 Hz.

Metric values were measured at 8 lossy working points (WPs) using three metrics. Figure 3‑1 to Figure 3‑3 show graphs representing the average values for each WP vs. BPS (bits per sample) in the ECG category.

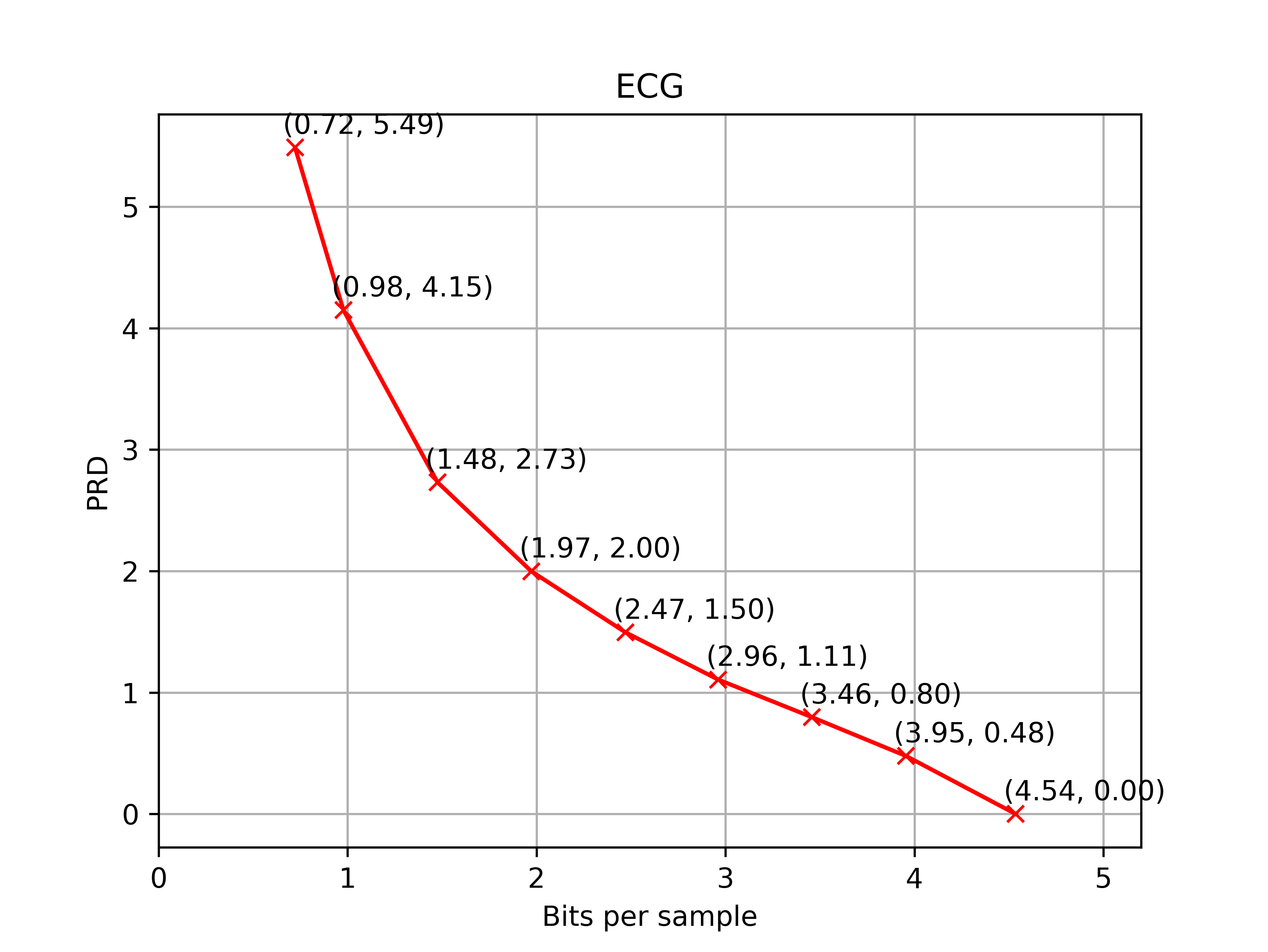


Figure 3‑1. PRD over BPS for the ECG dataset

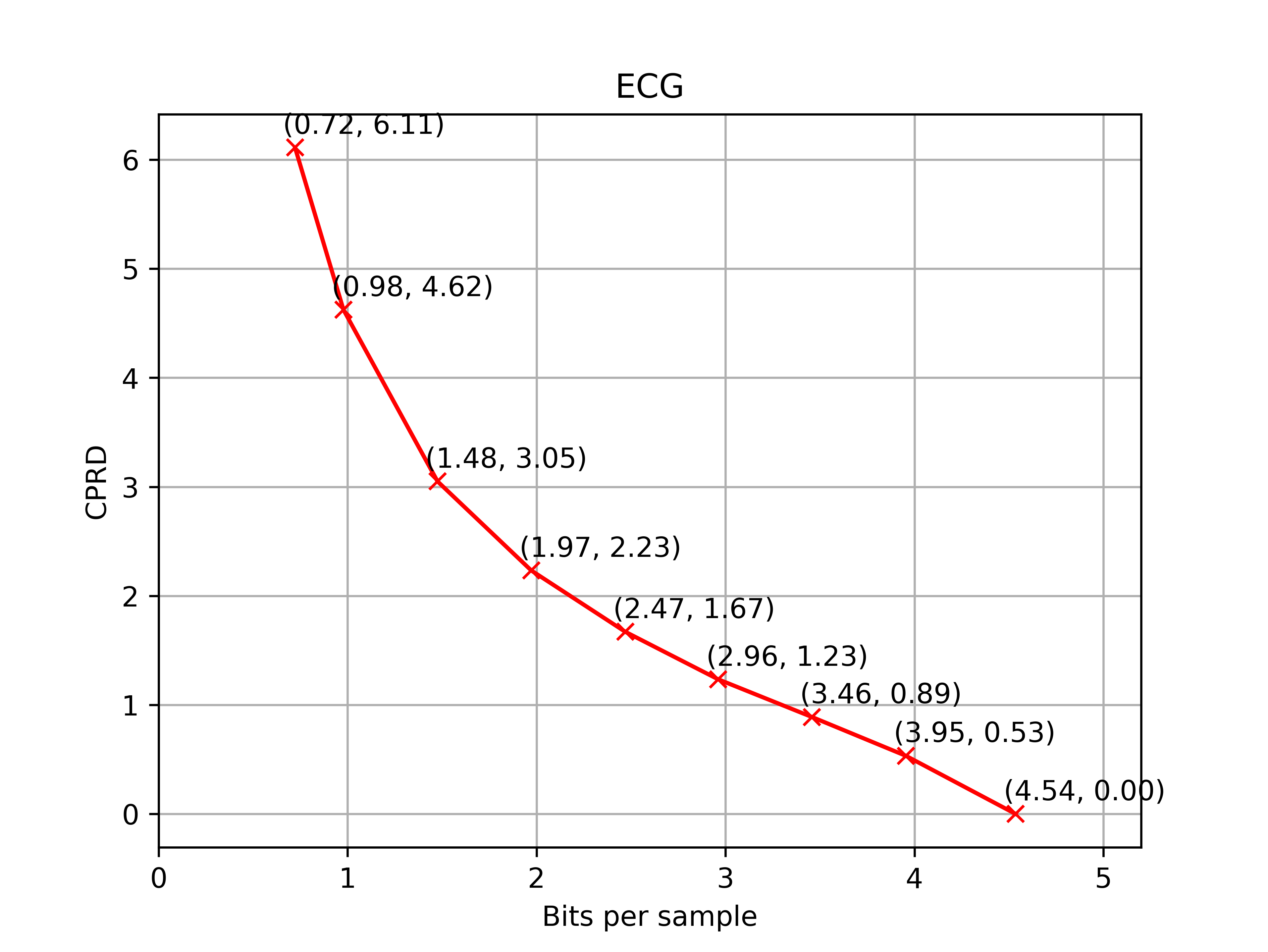


Figure 3‑2. CPRD over BPS for the ECG dataset

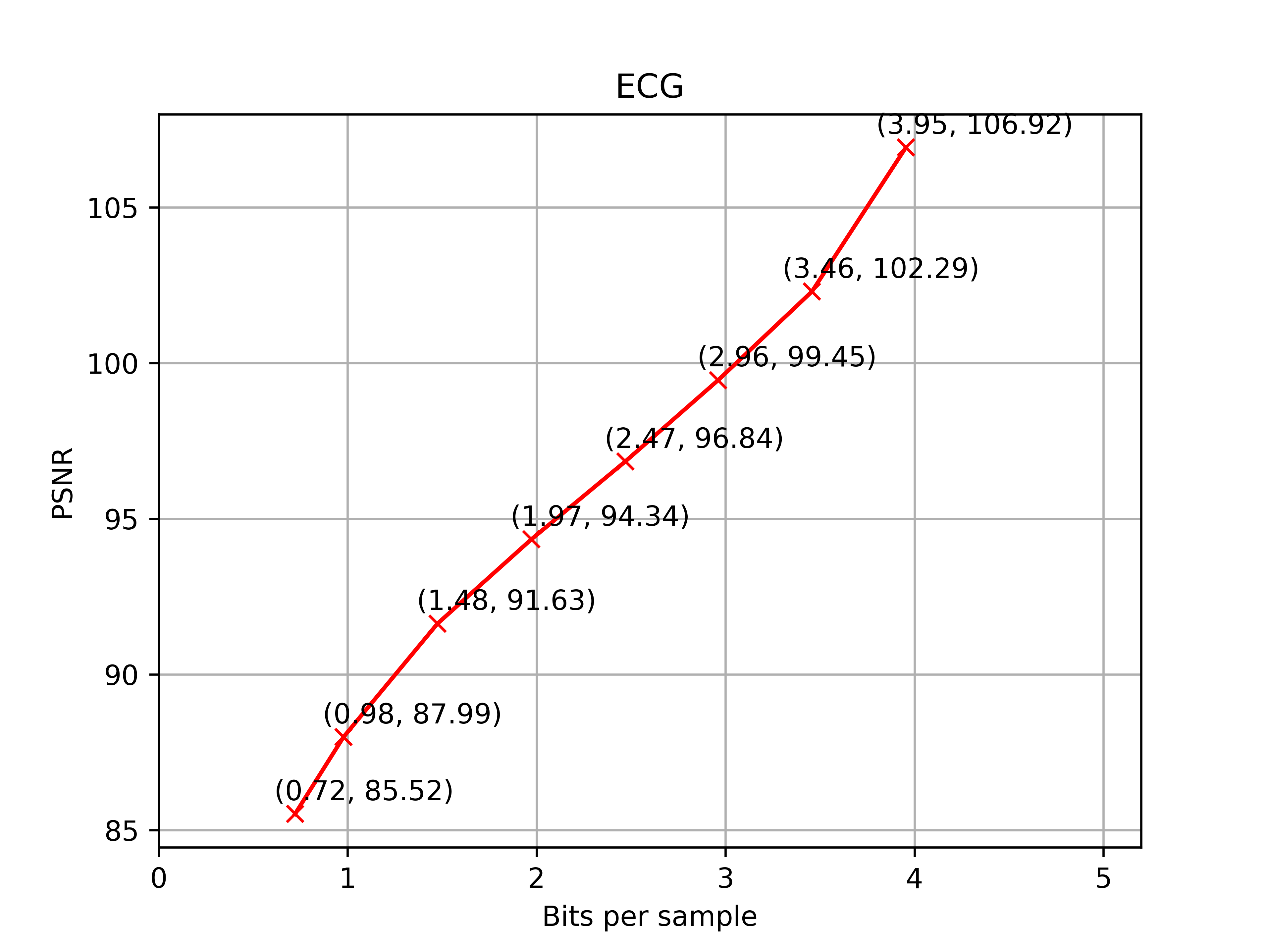


Figure 3‑3. PSNR over BPS for the ECG dataset

# Software implementation and running environments

The proposed codec was implemented using Python 3.11 language with the following major packages:

* *numpy*: numerical computing
* *scipy*: scientific computing
* *pywavelets*: wavelet transforms
* *pyedflib*: read/write EDF/EDF+ files
* *bitarray*: efficient arrays of Booleans

We confirmed that the decoder executable was working on Ubuntu 22.04.

# Complexity

The decoding complexity characteristics for implementation of the proposed codec were measured in terms of memory capacity for decoding program, fixed data tables, working data storage, and computational requirement, which are shown in Table 5‑1.

Table 5‑1. Complexity characteristics of the decoder

|  |  |
| --- | --- |
| Memory capacity for decoding program (in MegaBytes) | 57.56 |
| Fixed data tables (in MegaBytes) | 0.97 |
| Working data storage (in KiloBytes) | 726 |
| Computational requirement (RTF) | 0.016 |

The computational requirement was measured as processing time in real-time factor (RTF) rather than operational counts. The RTF is measured as follows:

RTF = (processing time) / (duration of an input data).

We measured the RTF by running the decoder executable using a single core on the following system:

* CPU: 13th Gen Intel Core i9-13900K (24 core)
* Memory: 128 GiB
* OS: Ubuntu 22.04.

The RTF values in Table 5‑1 are an RTF averaged across all files in the dataset. Note that every file in the ECG dataset has the same number of channels and sample length.

# Conclusion

ETRI submitted a response to the CfP for the ECG categories, and presented the technical description of the proposed codec. ETRI also evaluated the proposed codec using the three metrics and profiled the complexity characteristics of the decoder executable.

ETRI strongly supports the standardization works to develop a new Recommendation for biomedical waveform data coding and hopes to contribute to the collaborative work in the later stage.

# Patent rights declaration(s)

**ETRI may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

# References

1. G. Sullivan, Y. Ye and T. Wiegand, “Report of Question 6/16 visual, audio and signal coding,” Q.6/SG16, *SG16-TD242/WP3*, Rennes, Apr. 2024.
2. J. Pfaff and J. Halford, “Call for Evidence on the coding of biomedical waveform data,” Q.6/SG16, *doc.VCEG-BT07*, Hannover, Nov. 2023.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_