Wavelet Image Two-Line Coder for Wireless Sensor Node with extremely little RAM*

Stephan Rein, Stephan Lehmann, Clemens Gühmann
Wavelet Application Group, Dept. of Energy and Automation Technology
Technische Universität Berlin
[stephan.rein|clemens.guehmann]@tu-berlin.de uni@stephanlehmann.net

Abstract

This paper gives a novel wavelet image two-line (Wi2l) coder that is designed to fulfill the memory constraints of a typical wireless sensor node. The algorithm operates line-wisely on picture data stored on the sensor's flash memory card while it requires approximatively 1.5 kByte RAM to compress a monochrome picture with the size of 256x256 Bytes. The achieved data compression rates are the same as with the set partitioning in hierarchical trees (Spiht) algorithm. The coder works recursively on two lines of a wavelet subband while intermediate data of these lines is stored to backward encode the wavelet trees. Thus it does not need any list but three small buffers with a fixed dimension. The compression performance is evaluated by a PC-implementation in C, while time measurements are conducted on a typical wireless sensor node using a modified version of the PC-code.

1: Introduction and Motivation

A wireless sensor network consists of small devices that typically are equipped with a low-complexity microcontroller. These systems can gather and transfer environmental information, e.g., temperature, acceleration, and sound. Pictures are rarely taken by these low-cost systems, as the controllers are considered to be of too low complexity to process picture data and the network itself is designed for very small data rates. A low-complexity picture compression technique that gives good quality for high compression rates may solve these problems in that it prevents network congestion and reduces transmission times. The compression can also save energy within the network, as the coding energy is lower than the transmission energy.

State of the art compression performance for high compression rates is achieved with wavelet based techniques like *set partitioning in hierarchical trees* (Spiht). However, due to the high memory requirements of the transform and the coding system, even recently developed versions of wavelet tree coding systems are rather applicable to field programmable gate arrays (FPGA) or digital signal processors (DSP) than to the typical sensor node platforms that are subject of the current sensor network research activities. A wavelet-based picture compression system would have to get along with extremely little memory as the employed microcontroller's RAM and program memory is generally exhausted by the wireless communication protocols. In this work, such a system - the wavelet image two-line (Wi2l) coder, is detailed, which uses a backward algorithm for coding of wavelet trees. The coder only allocates 1.5 kByte of RAM for a monochrome picture with 256x256 8-Bit pixels. A typical sensor node can thus be extended to a camera sensor node by a software update, a camera module, and a flash memory card for storage of the pictures. As the cost for flash memory has fallen dramatically while size, power consumption, and access times were improved [7], flash memory like a multimedia card (MMC) may be a standard component of future sensor nodes to store the gathered data and to allow for more complex applications while relieving the wireless network. To demonstrate the proposed algorithm

* accepted for publication at the IEEE Data Compression Conference'09, Snowbird, UT, March 2009

a typical sensor node is extended in this way. The node consists of the *Microchip dsPIC30F4013* controller with 2 kBytes of RAM, the C328-7640 camera module (see *http://www.comedia.com.hk*), and a standard 64 MByte MMC card. The MMC card data is accessed through the library given in [14]. Camera and flash card can be connected to any controller with universal asynchronous receiver/transmitter (UART) interface and serial port interface (SPI).

The paper is organized as follows. In the next subsection related literature is reviewed. A recent coding system that backwards codes wavelet trees is taken as a starting point for the own research investigations. In section 2 the low-memory wavelet transform algorithm introduced in [10] is shortly reviewed, which is used in this work to calculate the picture wavelet coefficients with fixed-point arithmetic using 16 Bit integer numbers. Section 3 describes the novel recursive wavelet coder including the complete pseudo-source code that exactly reflects our own C-code program. A performance evaluation is given in section 4, where the system is demonstrated to give the same compression as Spiht, and time measurements are conducted for the own sensor architecture. The last section summarizes the paper and reflects our findings.

1.1: Related Work

There exists a large amount of literature in the field of low-memory wavelet-based image coding. We here review work that intends to reduce the memory requirements of Spiht, which is a standard technique for image coding that gives very competitive compression while it is conceptually simple. We categorize the work into two classes, the first one reducing the requirements for storage of intermediate data, and the second one concerning the maintenance of the picture data itself. Examples of the first class are given in [16, 15, 1, 12, 8, 13, 9], where the list of insignificant pixels (LIP), the list of significant pixels (LSP), and the list of insignificant sets (LIS) are reduced or excluded. As these lists are essential in Spiht and can require several MBytes of RAM, the reported memory savings are given in respect to the lists.

In the second class of low-memory coders the memory requirements of the transformed picture itself are addressed. Whereas the first class assumes free access on each pixel, this class only considers a subset of all pixels to be accessible by the coding algorithm. The memory needed for this subset is added to the RAM requirements. This class is discussed in the following to be more relevant for this paper, as only a small buffer of coefficients is available for direct coding activities.

The coders in [6, 5] both are designed for FPGA platforms and require 15 kByte of RAM for a picture dimension of 128x128, and 0.5 MByte of RAM for a picture dimension of 512x512, respectively. The work in [2] reports that there are at minimum eight memory lines needed by proposing a novel tree structure, which would require 256x8x2 bytes RAM for a picture dimension of 256x256 using 16 Bits per coefficient. Results are given using a Matlab simulation. While these works require specific hardware or were not yet verified using low-level implementations, the memory requirements are still too high. We consider the backward coding wavelet tree (Bcwt) algorithm in [3, 17] to give the most potential for a low-complexity wavelet coder and resume its idea of backwards coding, where the encoder starts at the lowest level. The coded coefficients can here be discarded as a map keeps track of the most significant levels of a tree. It is actually a backward version of Spiht with same compression rates with the cost that the progressive feature is distorted. If w denotes the picture width in pixels, Bcwt requires $42 \cdot w$ coefficients (16 Bit) and $3 \cdot w$ quantization levels (8 Bit) for the map, resulting in more than 20 kBytes of RAM for a 256x256 picture. This is due to the fact that coding one Bcwt unit requires data from several

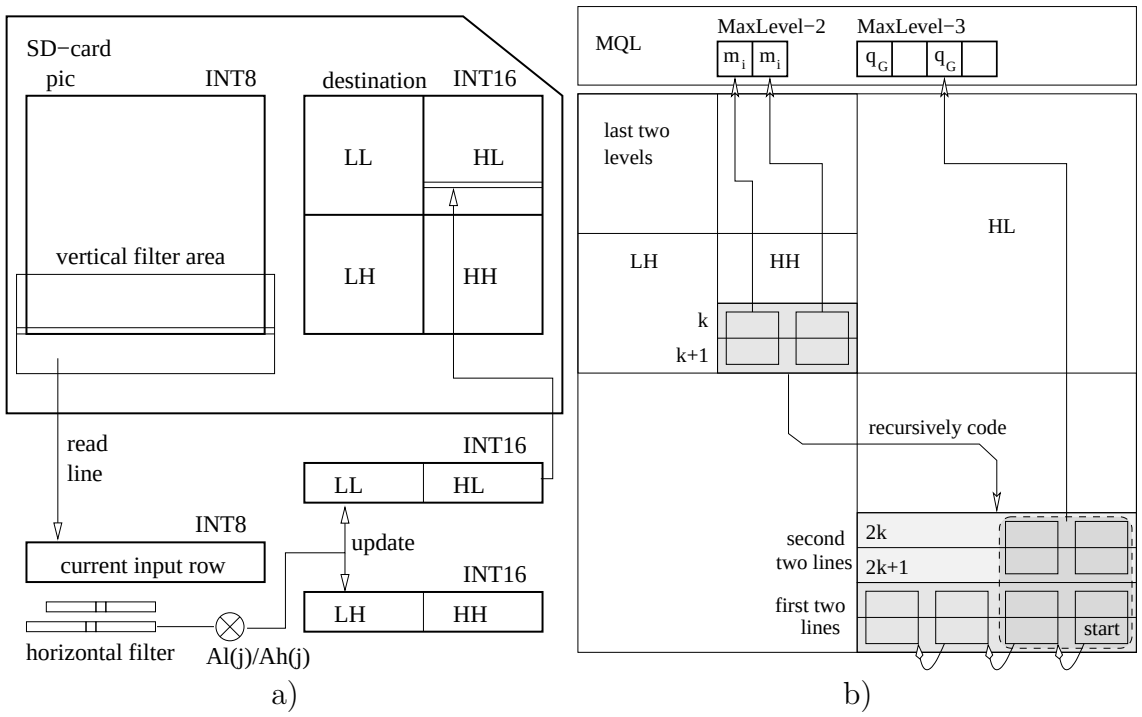

Figure 1. Fig.a) shows the system for the picture wavelet transform. The horizontal wavelet coefficients are computed on the fly and employed to compute the fractional wavelet coefficients that update the subband buffers. Fig.b) illustrates how Wi2l recursively codes two lines from right to left using a small buffer with level information of the lines. Such a buffer exists for each wavelet subband from the third highest level.

lines from different wavelet levels. A Bcwt unit codes a block of 16 coefficients while data of the parent coefficients is required. A unit generally also needs tree level data, which is stored in a list that concerns a complete subband.

This paper introduces the novel backward Wi2l coder that in contrast to Bcwt operates on two lines of a subband and only stores intermediate results concerning these lines in a small maximum quantization level (MQL) buffer. Thus the memory requirements are approximatively 1.5 kByte for the reference picture dimension of 256x256 pixels. To our best knowledge such a low-requirement system has not yet been proposed in the literature.

2: Low-Memory Fixed-Point Wavelet Transform

In this section we give a short outline of the low-complexity wavelet transform technique that is introduced in [10] and employed in this work before the coding algorithm starts. The same limitations than for the encoder hold true for the transform: 1)RAM memory requirements should be very low, as the total RAM of the controller is smaller than 2 kByte, and 2) picture data can be read line by line from external memory. This is due to the flash memory card, which only can read or write blocks of 512 Bytes. The transform uses the Daubechies 9/7 filter coefficients and real numbers are computed with fixed-point arithmetic on 16 Bit integers. Thus the controller only performs integer calculations, where we use *INT8* and *INT16* to denote the integer data format with 8 or 16 Bits. The system is illustrated in figure 1 a) and works as follows for the first wavelet level. The picture is located on the flash

a) Code2Lines(band,level,k)

```
//Code line k and line (k+1):
1) if level > 1
     Code2Lines(band, level − 1, (2k + 2))
     Code2Lines(band, level − 1, (2k))
2) Read lines k and (k + 1) to buffer
3) DimMql=DimPic/2^(level+1)
  for TwoSets=DimMql−1:−2:1{
3a) for i=1:−1:0
      set(i)=TwoSets+i−1
   i) if level > 1 get q_Gi(level − 1, 2set(i))
        from buffer
      else q_Gi = −1
  ii) m_i = max{q_Gi, max_{∀j∈set(i)}{q_j}}
 iii) if m_i ≥ q_min
    A) if level > 1
         encode q_Gi(level − 1) using m_i
    B) ∀ j ∈ set(i) do encode c_j
         using m_i
3b) if k/2==odd {Write m_{0,1} to
      buffer(TwoSets, TwoSets − 1)}
    else //second two lines
   i) Get m_{2,3} of previous two lines
      from buffer
  ii) q_G = max{m_0, m_1, m_2, m_3}
 iii) if q_G ≥ q_min
       encode m_{0...3} using q_G
  iv) Write q_G to buffer(TwoSets − 1)
 }//loop 3)
```

b) Decode2Lines(band,level,k)

```
//decode line (k+1) and line k:
1) Init coefficients in
   line buffer with 0
2) DimMql=DimPic/2^(level+1)
  for TwoSets=1:2:DimMql−1 {
2a) if k/2==even
   i) Get q_G from buffer
      at position(TwoSets − 1)
  ii) if q_G ≥ q_min
       decode m_3, m_2, m_1, m_0 using q_G
      else init these levels with −1
 iii) Put m_2, m_3 to buffer at
      position (TwoSets − 1), TwoSets
    else { // k/2 is odd:
     Get m_0, m_1 from buffer at
     position (TwoSets − 1) and TwoSets
    }
2b) for i=0:1
   i) if m_i ≥ q_min decode set i of
      coefficients using m_i and
      write to line buffer
  ii) if level > 1
    A) if m_i ≥ q_min{decode
       q_Gi(level − 1) using m_i}
      else q_Gi(level − 1) = −1
    B) Put q_Gi(level − 1, 2 · (TwoSets + i − 1))
       to buffer
  }// loop 2)
3) Write lines (k + 1) and k from line
   buffer to destination
4) If level>1
   Decode2Lines(band, level − 1, (2k))
   Decode2Lines(band, level − 1, (2k + 2))
```

Figure 2. Core function of the novel Wi2l coder (fig.a)) and decoder (fig.b)). Even if two lines are coded recursively, the input line buffer and the MQL buffer are the same for all instances. Coding of levels and MQL buffer activities vary with coding the first or the second two lines.

memory in the INT8 data format. Three buffers have to be allocated in the RAM memory, one for the input line and two destination buffers that are updated by the wavelet algorithm until they keep the final coefficients. The buffers need $N/level \cdot 5$ Bytes for a picture with the dimension NxN and a transform with *level* levels. So for a 256x256 dimension, 1280 Bytes are needed. The first picture line is read into the input line buffer. Two wavelet coefficients one for the lowpass filter Al and the other one for the highpass filter Ah are computed on the fly. These two values are used to compute four fractions of four coefficients each of them referring to one of the wavelet subbands LL, HL, LH, HH, where HL for instance denotes the subband resulting from horizontal highpass filter operation followed by a vertical lowpass filter operation. This operation of computing four fractions is repeated until Al and Ah have been slided over the whole input line and the two destination buffers are filled. Now the same procedure is repeated for the next eight picture input lines, as the larger vertical filter has nine coefficients. For each input line the fractional coefficients in the destination buffers are updated. After the last update operation four wavelet subband lines are located in the destination buffers and can be written to the flash card. The other subband lines are computed as well. For the next level the LL subband on the card is used as the input for the input buffer. The input buffer must now be of the type INT16 and keep $N/2$ elements.

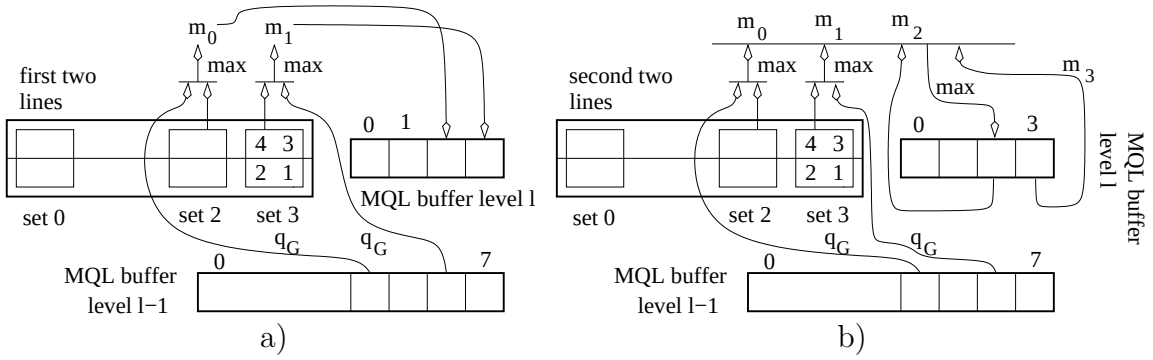

Figure 3. Fig.a) illustrates how the Wi2l algorithm acts on the first two lines. The m_i levels are computed retrieving the q_G levels through the MQL buffer of the previous level. The result is stored to the current level MQL buffer. In fig. b) the m_i levels for the second two lines are computed similarly. The m_i levels of the first two lines are retrieved from the current level MQL buffer to compute the q_G levels for the next wavelet level.

The first level coefficients are in the Texas Instrument's Q10.5 data format, the second level coefficients in the Q11.4 format, and so on. We typically use 6 levels. The filter coefficients are in the Q15 data format. In our implementation, each wavelet level is stored on a separate location on the flash card. Thus it is possible to access two wavelet subband lines by one read operation.

3: A Novel Coding Algorithm

3.1: Binary Notation and Quantization Levels

In this subsection the binary output when coding coefficients or levels is described. A coefficient c is coded using a minimum quantization level q_{min} and a maximum quantization level q_{max}. q_{min} is constant throughout the coding procedure and is thus omitted in the algorithm description. If $\mathrm{B}(c)|_a^b$ denotes the binary sequence of a coefficient c starting at the ath and ending at the bth right most bit, c is coded as $\mathrm{B}(c)|_{q_{min}}^{q_{max}}$, where q_{max} denotes the maximum quantization level relevant for the current set of coefficients. As c can be negative, a sign bit precedes this binary sequence. Coding a level q is done using q_{max} by writing the binary output $\mathrm{B}(2^q)|_{\max\{q,q_{min}\}}^{q_{max}}$. The work in [3] gives examples using this notation. Note that the quantization level for a coefficient c is given as $\lfloor \log_2 |c| \rfloor$ if $|c| \geq 1$ and as -1 otherwise. Levels are implemented with the INT8 data type.

This paper also uses the typical Spiht notation [11] in that a coefficient $c(i,j)$ has four children $C(pos(.,.))$ at the positions $(2i,2j),(2i,2j+1),(2i+1,2j)$, and $(2i+1,2j+1)$. The level m_i denotes the maximum quantization level of set i of four coefficients and all the descendant coefficients (including the children, children of the children, and so on). A quantization level q_{Gi} is the maximum quantization level of a set of 16 coefficients (that are the children of the coefficients in set i) and all their descendants.

3.2: Coding of Two Lines

The core of the novel algorithm is a recursive function that codes two lines of a wavelet subband. Three buffers - the line buffer, the maximum quantization level

a) CodePic():

```
1) // Code 3 subbands:
   list = {HL, LH, HH}
   for i=1:3 {
1a) Code2Lines(list(i),
      level = MaxLevel − 2, k = 2)
1b) q_G(i)=Code2Lines(list(i),
      level = MaxLevel − 2, k = 0)
   }
2) CodeLast2Levels(q_G(i = 1, 2, 3))
```

b) DecodePic():

```
1) q_G(i = 1, 2, 3)=DecodeLast2Levels()
2) // Decode the 3 subbands:
   list={HL, LH, HH}
   for i=3:1 {
2a) Write q_G(i) to appropriate buffer
      position
2b) Decode2Lines(list(i),
      level = MaxLevel − 2, k = 0)
2c) Decode2Lines(list(i),
      level = MaxLevel − 2, k = 2)
   }
```

Figure 4. Main loop of the Wi2l coder for low-memory encoding (fig. a)) and decoding (fig. b)). It is sufficient to call the recursive core function two times from the third highest level to completely code one of the three subbands from this level.

(MQL) buffer, and the input/output buffer for binary data, are employed within the function. The line buffer is loaded with two lines from a wavelet subband located on the flash memory. Its dimension is given as $N \cdot 2$ Bytes, where $N \cdot N$ is the picture dimension. The MQL buffer maintains maximum quantization levels of the coefficients. Its dimension is given as $\sum_{i=1}^{log_2(N)-2} N/2^{i+1}$. A function $q_{Gi}(level, n)$ retrieves the nth element of the buffer for a given level. The third buffer reads or writes the binary compressed data and is set to 512 Bytes. As each recursive function call puts variables onto the stack, we also calculate 256 Bytes for stack content. Thus we get a total memory of 512+512+126+256=1406 Bytes for $N = 256$.

We now go through this function given in figure 2 a). The function is called with parameters specifying the two lines to be coded. These include the subband given as HL, LH or HH, a wavelet level, and the line index k, which results into coding blocks of 4 coefficients within the lines k and $k + 1$.
In **step 1)** the four child lines of lines k and $k + 1$ are coded. This makes sure that the q_G levels of all child coefficients of lines k and $k + 1$ are computed. One of these MQL levels refers to 16 child coefficients.
In **step 2)** the two lines of coefficients to be coded are read into the line buffer. There only exists one line buffer for all function calls. Even if there exist recursive calls, the buffer is relieved when the function returns.
In **step 3)** the main loop for the function is started. This loop goes through all sets of eight adjacent coefficients in the two lines from right to left. Such a set is denoted by the variable *TwoSets*.
In **3a)** this set is divided into two subsets $set(i = 0, 1)$ each of them having 4 coefficients. For each $set(i)$ the function goes through three steps: In i) q_{Gi} is retrieved from the buffer and used in ii) together with the coefficient quantization levels $q_j, j \in set(i)$, to compute the level m_i. In iii) q_{Gi} and $set(i)$ of coefficients are coded.
In **3b)** the function detects if the first two or the second two lines are coded. Actions for each of these situations are different, as a set of eight coefficients denoted by $TwoSets$ in two lines forms a unit with the eight corresponding coefficients in the next two lines. Such a unit is coded as a whole in Bcwt and Spiht, but is broken here into several parts that are intermitted by parts of other units and quantization levels. If the function concerns the first two lines, then the two quantization levels m_0 and m_1 are written to the MQL buffer at the positions $(TwoSets, TwoSets - 1)$ of the current level. The MQL buffer then serves to store intermediate results. When the function is called for the next two lines, these two quantization levels will be retrieved in 3b) i) and here denoted as m_2, m_3. Then they are used in ii) to compute

a) CodeLast2Levels($q_G(i = 1, 2, 3)$):

```
1) Read 4x4 matrix of last two
   levels into input line buffer
2) //Denote positions of H and H_\LL
   pos[4]={0,1,4,5}
3) for i=1:3
     m(i) = max{q_G(i), max_{j∈C(pos(i))}{q_j}}
4) q_G(0) = max_{i=1,2,3}{m(i)}
5) for i=1:3 {
5a) Encode q_G(i) using m(i)
5b) //Encode coefficients of the
     //MaxLevel-1 bands HL,LH,HH
     if m(i) ≥ q_min do ∀j ∈ C(pos(i))
       encode c_j using m(i)
5c) Encode m(i) using q_G(0)
   }
6) m(0) = max{q_G(0), max_{i=0...3}{q(pos(i))}}
7) Encode q_G(0) using m(0)
8) //Encode coefficients of H:
   for i=0:3 Encode coefficient
     at pos(i) using m(0)
9) Encode m(0) using q_max
```

b) DecodeLast2Levels():

```
//a 4x4 matrix is decoded
1) Decode m(0) using q_max
2) pos[4]={0,1,4,5}
3) for i=3:-1:0 Decode coefficient
     at pos(i) using m(0)
4) Decode q_G(0) using m(0)
5) for i=3:-1:1 {
5a) Decode m(i) using q_G(0)
5b) if m(i) ≥ q_min do ∀j ∈ C(pos(i))
      decode c_j using m(i)
5c) Decode q_G(i) using m(i)
   }
6) Write decoded coefficients
     from line buffer (4x4 matrix)
     to destination matrix
7) Return q_G(i = 1,2,3) to decoder
   main function
```

Figure 5. The Wi2l coder uses a standard technique related to Spiht for coding (fig.a)) and decoding (fig.b)) the last two wavelet levels. It makes sure that the enclosed coefficients and the quantization levels of the connected trees are coded.

the quantization level q_G. In iii) the levels $m_{0,1,2,3}$ are coded and in iv) q_G is written to the MQL buffer at position $(TwoSets - 1)$ for the current level.

As can be concluded now, four adjacent lines have to be coded in step 1) to compute the corresponding q_G levels, which are then written to the MQL buffer at position $TwoSets - 1$ to be accessed by function calls at the next higher level. The value in the MQL buffer at position $TwoSets$ is not needed any more. For completeness the decoding algorithm is listed in figure 2 b). The encoding procedure and the difference between coding the first and the second two lines is illustrated in figure 3. Fig. a) refers to the coding of the first two lines. Note that coding a set i of coefficients is interluded by coding the corresponding q_{Gi} level. When the second two lines of a subband are processed (fig. b)), coefficients and q_G levels are coded similarly. However, the coding of two sets is interluded by encoding the four m_i levels for the current two sets and the associated two sets of the previous two lines.

3.3: Main Loop and Coding of the Last Levels

The main loop for the coding process is illustrated in figure 4. For each of the three possible wavelet subbands the recursive function *Code2Lines()* has to be called two times, one time for the first two lines in the level $MaxLevel - 2$, and a second time for the second two lines of this level. As illustrated in figure 1 b) the coder then goes recursively through all the levels of the given subband, starting to encode the lines from the lowest level. The q_G level of the $MaxLevel - 2$ subband is passed to the function *CodeLast2Levels()*, which is given in figure 5.

Coding of the last two levels is performed through a separate function, as the last two wavelet levels are given as a 4x4 matrix and the typical q_G levels for sets of 16 coefficients do not exist any more. In step 1) the coefficients are read into the line buffer. In step 2) the set H of coefficients, which refers to the highest wavelet subbands, is denoted by the position array pos[]. When the first position p[0] is left, the array denotes the set $H_{\setminus LL}$, which is the set H excluding the LL subband. Step

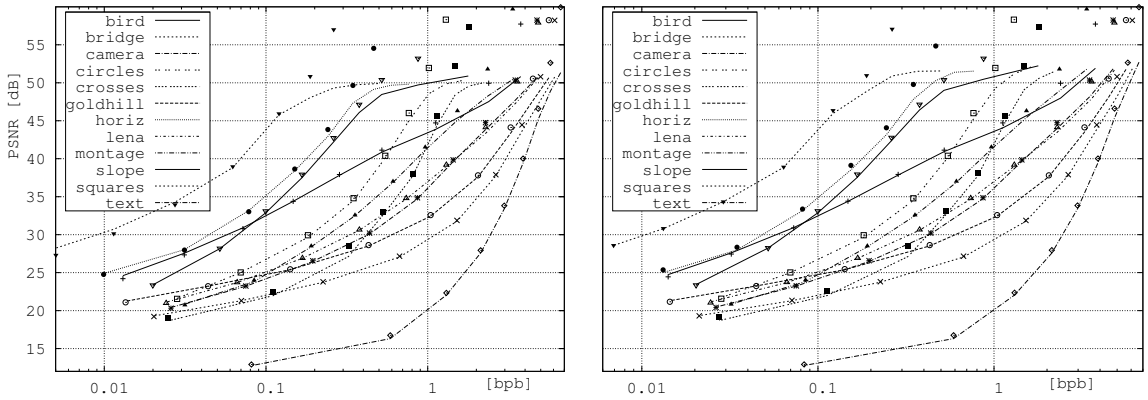

Figure 6. Fig. a) compares the achieved PSNR values for 9 different bits per byte rates resulting from the Wi2l minimum quantization levels 8,7,..,0, where the dots refer to Spiht and the lines to Wi2l. For the lower bpb rates, the achieved picture quality is nearly the same. As Wi2l uses a lossy, fixed-point wavelet transform, Spiht goes beyond Wi2l from roughly 50 dB. In Fig. b) wavelet level 5 is used for Wi2l.

3) computes the m_i levels $i = 1, 2, 3$ for the children of the set $H_{\setminus LL}$. In step 4) $q_G(0)$ is defined as the maximum of these m_i. In step 5) $q_G(i = 1, 2, 3)$ of the $MaxLevel - 2$ subbands and the coefficients with their m_i levels for the $MaxLevel - 1$ subbands are coded. In step 6) $m(0)$ is defined as the maximum quantization level of all coefficients. In the steps 7) and 8) $q_G(0)$ and the coefficients of set H are coded. In the last step $m(0)$ is coded using a predefined maximum quantization level q_{max}, which is set in the source code in accordance to the word length.

4: Performance Evaluation

4.1: Compression Results

For the quality evaluation the *GreySet1* test images from the *Waterloo Repertoire* (available at `http://links.uwaterloo.ca/bragzone.base.html`) were employed. This set contains twelve 256x256x8 greyscale images in the graphics interchange format (GIF). The pictures were converted to the portable network graphics (PNG) format using the *convert* command of the software suite *ImageMagick*. Finally, the data was converted to plain text with unsigned char numbers (INT8) using the software *Octave*. The data was then transformed with 6 levels and coded using an own C-implementation of the fractional wavelet filter and the Wi2l coder, and then retransformed in a similar way. The reconstructed picture was compared to the original by computing the peak signal-to-noise ratio (PSNR). The measurements were repeated for the minimum quantization levels $q_{min} = 0 \ldots 8$. The same PSNR computation was done using the Spiht executable programs *fastcode* and *fastdecd* from `http://www.cipr.rpi.edu/research/SPIHT/spiht3.html`, where the previously achieved bits per byte (bpb) rate from Wi2l for the different q_{min}s were taken as an input parameter for *fastcode*. The resulting comparison is illustrated in figure 6 a) for wavelet level 6. In figure b) level 5 is used for the Wi2l coder. For picture qualities lower than 45 dB, the performance of Spiht and Wi2l is nearly identical. For the range of higher picture qualities, the PSNR values of Spiht go beyond the values

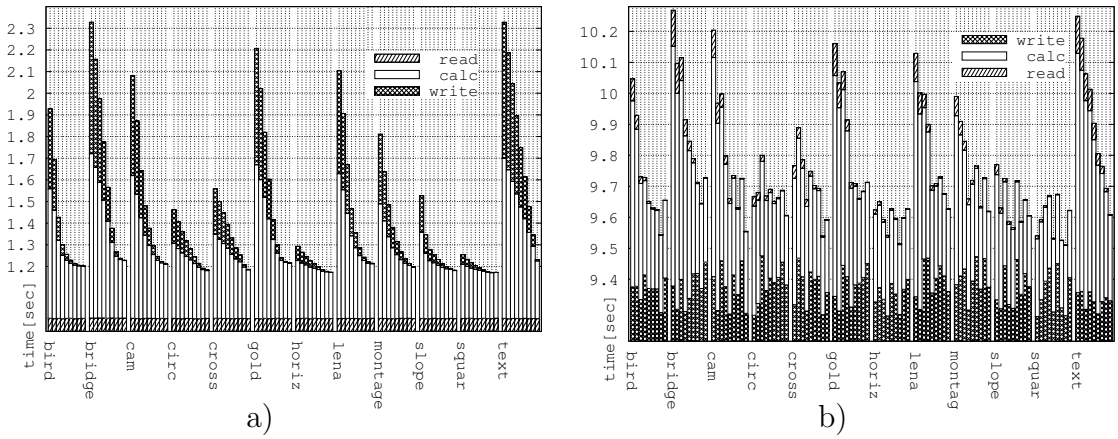

Figure 7. Coding (fig.a)) and decoding (fig.b)) time for a 256x256x8 picture on a real controller (dsPIC4013) with 29 MIPS. For each picture 9 measurements are visible for q_{min}=0..8. For each measurement the time for coding, read and write access is given.

of the Wi2l coder. This is because Wi2l uses a lossy, fixed-point wavelet transform for sensor nodes. There is not much difference between level 6 (fig.a)) and level 5 (fig.b)) in the figures. For the lower compression rates, level 5 performs slightly better.

4.2: Coding Time on Sensor

For the coding time measurements a real wireless sensor platform with the controller dsPIC30F4013 was employed, where the speed was set to 29.491 million instructions per second (MIPS). The PC-code was modified in that an MMC card is used for data exchange. The data can be wavelet coefficients or the compressed binary data stream, for which a write buffer of 512 Bytes was allocated. Figure 7 shows the results for a) coding and b) decoding times. For each picture, q_{min} was varied from $0 \ldots 8$. The lower the quantization levels, the longer the coding time. For the encoder the read access times stay almost constant at 0.96 seconds and take the largest proportion, as each coefficient has to be read. For $q_{min} = 4$ most pictures give coding times shorter than 1.3 seconds. For the decoder the write access time take 9.3 to 9.5 seconds, thus total decoding time is approximatively 10 seconds. This is because the decoder's write access is not sequentially as it is for the binary stream of the encoder. As detailed in [14], random write access on the MMC card is very slow and not stable.

5: Conclusion and Future Work

In this paper, the Wi2l coder for wavelet image compression was introduced. Wi2l is designed to fulfill the constraints of low-complexity sensor networks, which in the past were considered to be not sufficient for image processing. The coder reads two lines of a wavelet subband and codes the image backwards. For the image transform, a fixed-point wavelet filter is employed so that the complete system only performs integer calculations. As the coder works recursively, there only exists a small buffer for each wavelet level where intermediate results for the two lines are stored. Thus the memory requirements are extremely low: As it is demonstrated in this paper, a 256x256x8 Bit monochrome picture can be compressed using the dsPIC30F4013, a

low-cost controller with a total of 2 kByte RAM. As Wi2l is a backward version of Spiht, it achieves the same compression rates. Due to the employed lossy wavelet transform, the system is not suitable for lossless image compression. However, the loss of picture quality is barely visible for PSNR values higher than 40 dB, and lossless compression may be a minor issue regarding the application of sensor networks. Similarly, the long decoding times may be of secondary interest, as the pictures may be decoded outside of the sensor network.

Future work on Wi2l may integrate a progressive feature, as a bit-rate control is not yet possible. (For Bcwt such a feature was addressed in [4]). To speed up the algorithm, the three wavelet subbands can be coded as a whole, so that two lines contain two lines of each of the HL,LH, and the HH subbands. Another issue may concern the interconnection of the transform and the Wi2l coding system to minimize flash memory access.

References

[1] H. Arora, P. Singh, E. Khan, and F. Ghani. Memory efficient set partitionning in hierarchical tree (Mesh) for wavelet image compression. In *Proc.of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '05)*, volume 2, pages 385–388, March 2005.

[2] L.W. Chew, L.-M. Ang, and K.P. Seng. New virtual spiht tree structures for very low memory strip-based image compression. In *IEEE Signal Processing Letters*, volume 15, pages 389–392, 2008.

[3] J. Guo, S. Mitra, B. Nutter, and T. Karp. A fast and low complexity image codec based on backward coding of wavelet trees. In *Proc. of the Data Compression Conference (dcc'06)*, March 2006.

[4] J. Guo, B. Nutter S. Mitra, and T. Karp. Backward coding of wavelet trees with fine-grained bitrate control. In *Journal of Computers*, volume 1, pages 1–7, July 2006.

[5] A. Kumarayapa, X.-F. Zhang, and Y. Zhang. Simplifying Spiht for more memory efficient onboard machine-vision codec and the parallel processing architecture. In *Proc. International Conference on Machine Learning and Cybernetics*, volume 3, pages 1482–1486, Aug. 2007.

[6] M. Lanuzza, S. Perri, P. Corsonello, and G. Cocorullo. An efficient wavelet image encoder for FPGA-based designs. In *Proc. of IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 652–656, Nov. 2005.

[7] A. Leventhal. Flash storage memory. In *Communications of the ACM*, volume 51, July 2008.

[8] J. Lian, K. Wang, and J. Yang. Listless zerotree image compression algorithm. In *Proc.of 8th International Conference on Signal Processing*, volume 2, 2006.

[9] Hong Pan, W.C. Siu, and N.F. Law. Efficient and low-complexity image coding with the lifting scheme and modified Spiht. In *Proc. of IEEE International Joint Conference on Neural Networks (IJCNN '08)*, pages 1959–1963, July 2008.

[10] S. Rein, S. Lehmann, and C.Gühmann. Fractional wavelet filter for camera sensor node with external flash and extremely little RAM. In *Proc. of the ACM Mobile Multimedia Communications Conference (Mobimedia '08)*, July 2008.

[11] A. Said and W.A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 6, June 1996.

[12] M. Sakalli, W.A. Pearlman, and M. Farshchian. Spiht algorithms using depth first search algorithm with minimum memory usage. In *Proc. of 40th Annual Conference on Information Sciences and Systems*, pages 1158–1163, March 2006.

[13] Y.K. Singh. ISpiht-improved Spiht: A simplified and efficient subband coding scheme. In *Proc. of International Conference on Computing: Theory and Applications (ICCTA'07)*, pages 468–474, March 2007.

[14] S.Lehmann, S.Rein, and C.Gühmann. External flash filesystem for sensor nodes with sparse resources. In *Proc. of the ACM Mobile Multimedia Communications Conference (Mobimedia '08)*, July 2008.

[15] C.-Y. Su and B.-F. Wu. A low memory zerotree coding for arbitrarily shaped objects. In *IEEE Transactions on Image Processing*, volume 12, pages 271–282, March 2003.

[16] Y. Sun, H. Zhang, and G. Hu. Real-time implementation of a new low-memory Spiht image coding algorithm using DSP chip. In *IEEE Transactions on Image Processing*, volume 11, pages 1112–1116, Sep. 2002.

[17] L. Ye, J. Guo, B. Nutter, and S. Mitra. Memory-efficient image codec using line-based backward coding of wavelet trees. In *Proc. of the Data Compression Conference (dcc'07)*, 2007.