Analog Integrated Circuits and Signal Processing manuscript No.
(will be inserted by the editor)

Implementation and Performance Analysis of DVB-T2 Rotated Constellation Demappers on a GPU

Stefan Grönroos · Kristian Nybom · Jerker Björkqvist

Received: date / Accepted: date

Abstract The DVB-T2 standard for digital terrestrial broadcasting supports the use of QAM (quadrature amplitude modulation) constellations where the constellation points are rotated in the I-Q plane. This combined with a cyclic delay of the Q component provides improved performance in some fading channels. The complexity of the optimal demapping process for rotated constellations is however significantly higher than for non-rotated constellations. This makes the DVB-T2 demapper one of the most computationally complex parts of a receiver. In this article, we examine possible simplifications of the demapping process suitable for implementation on a general purpose computer containing a modern GPU (graphics processing unit). Furthermore, we measure the performance in terms of throughput, as well as accuracy, of the implemented algorithms. The implementations are designed to interface efficiently to a previously implemented real-time capable GPU-based LDPC (low-density parity-check) channel decoder.

Keywords DVB-T2 · Demapper · QAM · SDR · CUDA

1 Introduction

The DVB-T (Digital Video Broadcast Terrestrial) system for digital television broadcasting is widely used for broadcasting around the world. As high bitrate High-Definition Television (HDTV) broadcasts become more prevalent, however, the need for a more spectrum efficient standard increases. The DVB-T2 standard [6, 17] has been developed to address this need. This standard offers significantly increased capacity when compared to DVB-T. The increased capacity comes at the cost of more complex signal processing components in the physical layer, however.

Two of the most complex parts of a DVB-T2 receiver are the channel decoder and QAM (quadrature amplitude modulation) demapper [8]. For channel coding, the standard specifies the use of LDPC (low-density parity-check) codes [7] with exceptionally long codeword lengths as the inner coding scheme, as well as an outer BCH (Bose-Chaudhuri-Hocquenghem) code. DVB-T2 features QPSK, 16-QAM, 64-QAM and 256-QAM modulation. Optionally, the QAM constellation diagram may be rotated in signal space (the angle of rotation is specified for each modulation scheme). This rotation, combined with the subsequent interleaving of the in-phase (I) and quadrature (Q) components of the signal, provides signal-space-diversity [2, 11] and gives improved performance in some fading channels.

The authors of this article have earlier presented a real-time capable decoder of DVB-T2 LDPC codes implemented on a GPU (graphics processing unit) using the NVIDIA CUDA (Compute Unified Device Architecture) [9]. In a step towards creating a real-time capable software defined radio (SDR) implementation of a DVB-T2 receiver on a general purpose computer, we focus on implementing fast rotated constellation demappers in this article. The proposed implementations will, like the LDPC decoder already in place, be implemented using the CUDA on a consumer-grade GPU. In a DVB-T2 receiver chain, the demapper generates log-likelihood

S. Grönroos · K. Nybom · J. Björkqvist
Åbo Akademi University, Joukahaisenkatu 3-5A, 20520, Turku, Finland
E-mail: stefan.gronroos@abo.fi
E-mail: kristian.nybom@abo.fi
E-mail: jerker.bjorkqvist@abo.fi

S. Grönroos
Turku Centre for Computer Science TUCS

ratio (LLR) values to be used as inputs to the LDPC decoder, with only a bit interleaver separating the two signal processing blocks. As both blocks are implemented on a GPU, we can avoid copying of data between the GPU and CPU between the demapper and LDPC decoder blocks.

In the article, we examine the implementations of several demapping algorithms and measure their performance in terms of both speed and accuracy. We also measure the combined throughput when performing both demapping and LDPC decoding on the GPU, and compare the achieved figures to the maximum throughputs required by the DVB-T2 standard. The impact of the various algorithmic simplifications on demapping accuracy is measured by simulating DVB-T2 transmissions within a DVB-T2 physical layer simulator.

While the demapping of traditional Gray-mapped non-rotated QAM is not very complex, due to the possibility of treating the I and Q components independently, rotation of the constellation diagram changes this, and complexity is significantly increased due to the I and Q axes being inter-dependent. Various algorithms for the demapping of rotated constellations have been discussed in [1,3,10,11,16]. In addition to the maximum likelihood (ML) demapper as well as its max-log simplification [3, 11], the algorithms based on MMSE (minimum mean squared error) decorrelation and IC (interference cancellation) described in [10], will also be implemented and measured on the GPU.

The article is laid out as follows. In section 2, we describe the target platform of our implementations, including the CUDA and the specific GPU which was used to test the implementations. In section 3, we describe the various demapping algorithms that were implemented and tested on the GPU, while in section 4, we describe the actual implementations of these algorithms. Benchmark and simulation results are presented in section 5 along with discussion regarding the obtained results. The article is finally concluded in section 6.

2 Target architecture

In this section we describe the NVIDIA CUDA, and the specific GPU for which the GPU-based implementations were developed. Other relevant components of the system used for benchmarking the implementations are also described.

The desktop computer system, of which the NVIDIA GeForce GPU — on which the CUDA implementations were tested — was one component, also contained an Intel Core i7-950 main CPU running at a 3.06 GHz clock frequency. 6 GB of DDR3 RAM (Double Data Rate 3 random access memory) with a clock frequency of 1666 MHz was also present in the system. The operating system was the Ubuntu Linux distribution for 64-bit architectures. Version 4.2 of the CUDA Toolkit was used.

2.1 CUDA

The NVIDIA CUDA [15] is used on modern NVIDIA GPUs. This architecture is well suited for data-parallel problems, i.e problems where the same operation can be executed on many data elements at once.

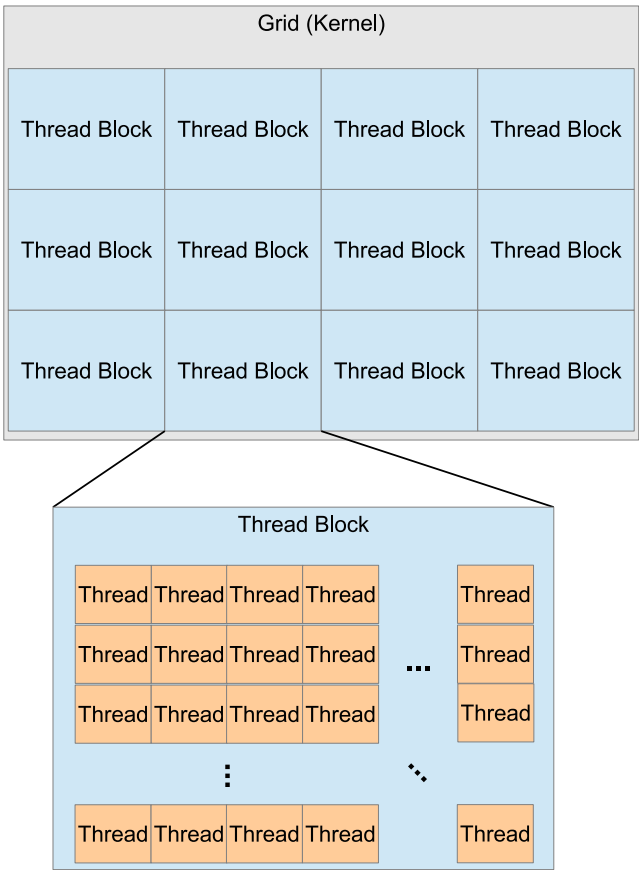


Fig. 1 As shown in [15], the threads running a CUDA kernel are divided into a grid consisting of a number of thread blocks, where each thread block is mapped onto one streaming multiprocessor. A group of 32 consecutive threads in a block are called a warp.

In the CUDA C programming model, we define kernels, which are functions that are run on the GPU by many threads in parallel. The threads executing one kernel are split up into thread blocks, where each thread block may execute independently, making it possible to execute different thread blocks on different pro-

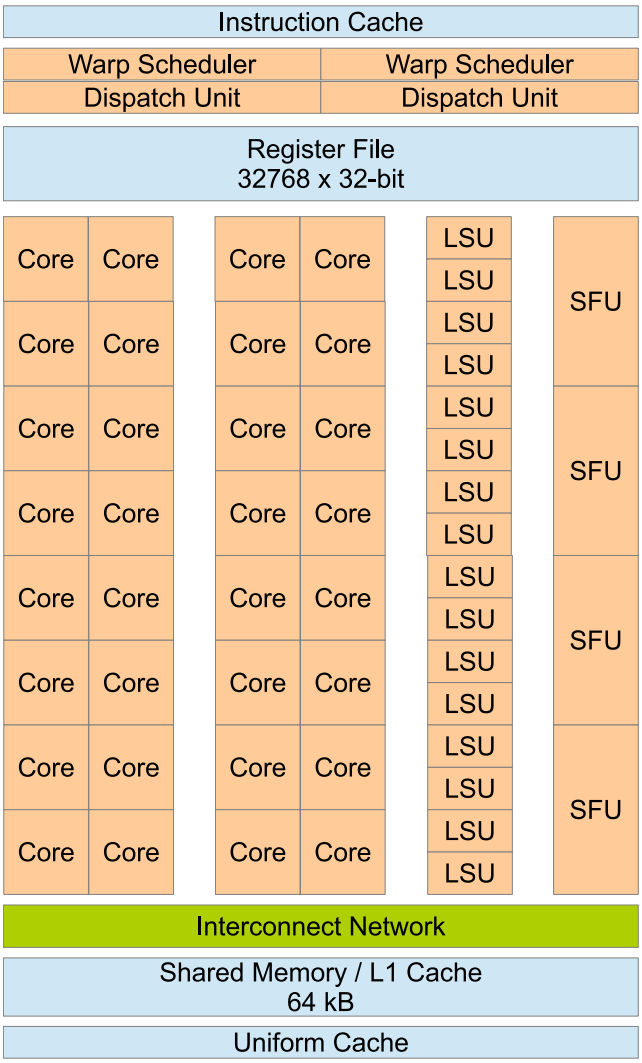


Fig. 2 A streaming multiprocessor (SM) in the Fermi architecture, as depicted in [13]. There are two warp schedulers per SM, which may schedule a group of 16 threads onto one of two groups of 16 cores. There are also 16 load-store units (LSU), and four special function units (SFU).

cessors on a GPU. This subdivision of threads into thread blocks is illustrated in Fig. 1. Both the thread blocks in a grid and threads in a block may be referenced by an up to three dimensional index in the kernel code (Fig. 1 illustrates a two-dimensional grid with two-dimensional thread blocks). The GPU used for running the LDPC decoder implementation described in this article was an NVIDIA GeForce GTX 570 [12, 14], which is based on the Fermi architecture [13], and features 15 streaming multiprocessors (SMs) containing 32 cores each. A block diagram of a streaming multiprocessor in the Fermi architecture is shown in Fig. 2. Threads are scheduled in groups of 32 threads of a thread block, called thread *warps*. The Fermi hardware architecture features two warp schedulers per SM, meaning the cores of a group of 16 cores on one SM execute the same instruction from the 16 threads of one half of a warp.

Each SM features 64 kB of fast on-chip memory that can be divided into 16 kB of L1 cache and 48 kB of shared memory ("scratchpad" memory) to be shared among all the threads of a thread block, or as 48 kB of L1 cache and 16 kB of shared memory. There is also a per-SM register file containing 32,768 32-bit registers. All SMs of the GPU share a common large amount of global RAM memory (1280 MB for the GTX 570), to which access is typically quite costly in terms of latency, as opposed to the on-chip shared memories.

The long latencies involved when accessing global GPU memory can limit performance in memory intensive applications. Memory accesses can be optimized by allowing the GPU to *coalesce* the accesses. When the 32 threads of one warp access a continuous portion of memory (with certain alignment limitations), only one memory fetch/store request might be needed in the best case, instead of 32 separate requests in the worst case if the memory locations accessed by the threads are very scattered [15]. In fact, if the L1 cache is activated (can be disabled at compile time by the programmer), all global memory accesses fetch a minimum of 128 bytes (aligned to 128 bytes in global memory) in order to fill an L1 cache line. Memory access latencies can also be effectively hidden if some warps on an SM are able to run arithmetic operations while other warps are blocked by memory accesses. As the registers as well as shared memories are split between all warps that are scheduled to run on an SM, the number of active warps can be maximized by minimizing the register and shared memory requirements of each thread.

3 Demapper algorithms

In this section, we describe various demapper algorithms that were implemented on the GPU. While non-rotated Gray-mapped QAM demappers may treat the I and Q components of the signal as two separate PAM (pulse amplitude modulation) signals, with rotated QAM constellations the two components become correlated and the demapper thus becomes more complex. In subsection 3.1, we describe the rotated QAM constellations used in DVB-T2, while in subsection 3.2, we describe various simplifications that can be made in the demapper to lower complexity.

3.1 DVB-T2 and Rotated Constellations

In this subsection, we briefly describe the Bit Interleaved Coding & Modulation (BICM) module of a DVB-

T2 modulator, as well as the rotated QAM constellations supported by the standard.

The input data streams to a DVB-T2 modulator [6], which are in the form of MPEG-2 Transport Streams or GSE (Generic Stream Encapsulation) encapsulated data are first split into one or more Physical Layer Pipes (PLPs), where each PLP may use different coding and modulation. The first module of the system is the Input Processing module. This module converts the input data streams into DVB-T2 baseband frames. After passing through the Input Processing module, each baseband frame is processed by the Bit Interleaved Coding & Modulation module. A block diagram of this module is shown in Fig. 3, and it contains the following stages (in order):

- FEC (Forward Error Correction) coding. DVB-T2 uses an outer BCH code, as well as an inner LDPC code. The resulting EFC blocks can be either 16200 (short code) or 64800 bits (long code) long. 6 different LDPC code rates are available.
- Bit Interleaver (not used for QPSK modulation). Consists of parity bit interleaving, followed by column twist interleaving.
- Constellation Mapper, which maps bits onto QAM constellations. DVB-T2 supports the use of QPSK, 16-QAM, 64-QAM, and 256-QAM modulation. This module outputs cells, i.e. coordinates in the complex I-Q plane, where each complex output value represents 2 (QPSK) to 8 (256-QAM) bits of data.
- Constellation Rotation, if rotated constellations are used. The cell values produced by the mapper are rotated in the complex plane (the angle depends on the modulation used), and the imaginary part is cyclically delayed by one cell.
- Cell Interleaver. Used to uniformly spread the cells of a FEC block.
- Time Interleaver. In this block, cells of groups of FEC blocks, making up TI-blocks – which in turn make up Interleaving Frames – are interleaved.

As seen above, after Gray mapping of bit sequences to constellations, the constellation diagram may optionally be rotated. The constellation diagrams for QPSK, 16-QAM, 64-QAM, and 256-QAM are rotated by 29.0, 16.8, 8.6, and $\arctan(\frac{1}{16}) \approx 3.6$ degrees, respectively. If rotation is enabled, the Q component is also cyclically delayed by one OFDM (orthogonal frequency-division multiplexing) cell. The cell interleaver in which the mapped OFDM cells are further interleaved follows the Q-delay, thus separating the I and Q components further, providing signal-space diversity.

The Bit Interleaved Coding & Modulation module is followed by the Frame Builder and OFDM generation modules [6], however these parts are not relevant for this article.

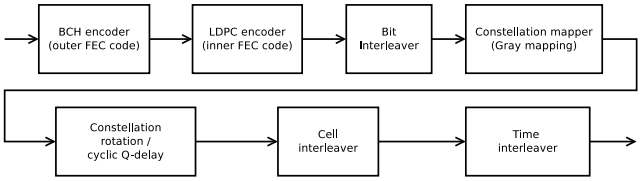


Fig. 3 Block diagram of the BICM module of a DVB-T2 modulator.

3.2 Algorithms

The optimal, although highly complex, maximum likelihood (ML) algorithm for calculating the LLR value for bit b_i ($i \in [1, m]$ if we use 2^m-QAM) can be expressed as follows [3]:

$$\begin{aligned}\mathbf{LLR}(b_i) &= \ln\left(\frac{Pr(b_i = 1|\mathbf{r}}{Pr(b_i = 0|\mathbf{r}}\right) \\ &= \ln\left(\frac{\sum_{\mathbf{x}\in C_i^1}(e^{-\frac{D(\mathbf{x})}{2\sigma^2}})}{\sum_{\mathbf{x}\in C_i^0}(e^{-\frac{D(\mathbf{x})}{2\sigma^2}})}\right),\end{aligned} \tag{1}$$

where

$$D(\mathbf{x}) = (r_I - \rho_I x_I)^2 + (r_Q - \rho_Q x_Q)^2.$$

Here $\mathbf{r} = \begin{bmatrix} r_I \\ r_Q \end{bmatrix}$ denotes the coordinate of the received OFDM cell in the two-dimensional I-Q plane, ρ_I and ρ_Q denote the amplitude fading factors of the channel, and σ^2 is noise variance. C_i^0 and C_i^1 denote the sets of rotated constellation points for which the i:th bit equals 0 and 1, respectively. Also note that $\mathbf{x} = \begin{bmatrix} x_I \\ x_Q \end{bmatrix}$ in (1).

To simplify (1), one can apply the approximation [3]:

$$\ln\left(\sum_{i\in[1,n]}(e^{a_i})\right) \approx \max_{i\in[1,n]}(a_i), \tag{2}$$

which yields the simplified max-log demapper equation:

$$\mathbf{LLR}(b_i) \approx \frac{1}{2\sigma^2}\left[\min_{\mathbf{x}\in C_i^0}\big(D(\mathbf{x})\big) - \min_{\mathbf{x}\in C_i^1}\big(D(\mathbf{x})\big)\right]. \tag{3}$$

While the max-log simplification of (3) does remove some complexity (at the cost of degraded demapping performance), we still need to calculate two-dimensional distances to 2^m points in the case of 2^m-QAM modulation. The authors of [16] discuss the possibility of

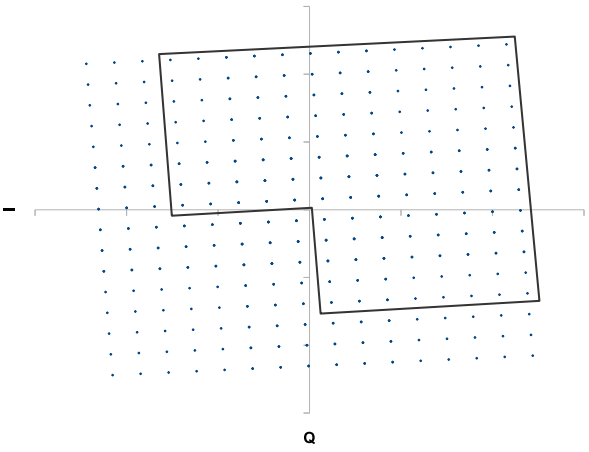


Fig. 4 One of four overlapping subsets of a 256-QAM rotated constellation diagram, as proposed in [16].

assigning the constellation points of a rotated constellation diagram into four overlapping subsets. This is illustrated in Fig. 4, where one proposed subset is shown for a rotated 256-QAM constellation diagram. A subset is chosen based on the signs of the received point's ($\mathbf{r}$) I and Q components, and only distances to points within that subset are calculated using the max-log demapper. Depending on the chosen size of the four subsets, complexity is reduced at the cost of some demapper accuracy. The subset size shown in Fig. 4 was demonstrated [16] to yield good demapper performance at a reduction of 44% in the number of distance calculations performed for each received OFDM cell. The authors of [11] also propose a similar division into subsets.

In [10], the authors propose performing an MMSE (minimum mean squared error) decorrelation followed by interference cancellation (IC) to decorrelate the I and Q components. This is similar to methods commonly used in MIMO (multiple input - multiple output) detectors. Based on the derotated and decorrelated I and Q components, we may perform reduced complexity demapping separately on the two components, similarly to traditional QAM demapping. In this case we have the channel matrix:

$$\mathbf{H} \doteq \mathbf{PQ} = \begin{bmatrix} \rho_I & 0 \\ 0 & \rho_Q \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix},$$

where θ is the rotation angle of the constellation diagram. The LLR values after decorrelation are given by (see [10] for detailed calculations):

$$\mathbf{LLR}(b_i) = \beta_k \left[\min_{\mathbf{a}\in C_i^0} \left| \frac{\hat{x}_{MMSE,k}}{\gamma_{kk}} - a_k \right|^2 - \min_{\mathbf{a}\in C_i^1} \left| \frac{\hat{x}_{MMSE,k}}{\gamma_{kk}} - a_k \right|^2 \right], \quad (4)$$

where $k = 2 - (i \mod 2)$, i.e. k equals 1 for odd values of i, and 2 for even values. This reflects the fact that odd bits are conveyed by the I component, and even bits by the Q component in the Gray mapped (non-rotated) constellation diagram. Also note that, in contrast to the ML and max-log algorithms, C here denotes the set of *non-rotated* constellation points. Furthermore,

$$\begin{aligned} \hat{\mathbf{x}}_{MMSE} &\doteq (\mathbf{H}^T\mathbf{H} + \sigma_n^2\mathbf{I})^{-1}\mathbf{H}^T\mathbf{r} \\ &= \mathbf{Q}^T(\mathbf{P}^2 + \sigma_n^2\mathbf{I})^{-1}\mathbf{P}^T\mathbf{r} \\ &= \mathbf{Q}^T \begin{bmatrix} \frac{\rho_I}{\rho_I^2+\sigma_n^2} r_I \\ \frac{\rho_Q}{\rho_Q^2+\sigma_n^2} r_Q \end{bmatrix}, \\ \mathbf{\Gamma} &\doteq (\mathbf{H}^T\mathbf{H} + \sigma_n^2\mathbf{I})^{-1}\mathbf{H}^T\mathbf{H} \\ &= \mathbf{Q}^T(\mathbf{P}^2 + \sigma_n^2\mathbf{I})^{-1}\mathbf{P}^2\mathbf{Q} \\ &= \mathbf{Q}^T \begin{bmatrix} \frac{\rho_I^2}{\rho_I^2+\sigma_n^2} & 0 \\ 0 & \frac{\rho_Q^2}{\rho_Q^2+\sigma_n^2} \end{bmatrix} \mathbf{Q}, \end{aligned}$$

and

$$\beta_k \doteq \frac{\gamma_{kk}}{1-\gamma_{kk}}$$

After calculating $\hat{\mathbf{x}}_{MMSE}$ and $\mathbf{\Gamma}$, the LLR calculation in (4) for a certain bit b_i is now only dependent on one axis of the non-rotated constellation diagram. To further decrease complexity, one may replace the minimum distance calculations in (4) with lookup tables [10].

Interference cancellation may be performed on the weakest channel (I or Q), which is determined by selecting the component corresponding to the smallest value of ρ_I and ρ_Q. To calculate the LLR for the weakest channel (strongest channel is calculated according to (4)):

$$\mathbf{LLR}(b_i) = \beta_{IC,k} \left[\min_{\mathbf{a}\in C_i^0} \left| \frac{\hat{x}_{IC,k}}{\gamma_{IC,k}} - a_k \right|^2 - \min_{\mathbf{a}\in C_i^1} \left| \frac{\hat{x}_{IC,k}}{\gamma_{IC,k}} - a_k \right|^2 \right], \quad (5)$$

where

$$\hat{x}_{IC,k} \doteq \frac{\mathbf{h}_k^T(\mathbf{r} - \mathbf{h}_j\bar{a}_j)}{\mathbf{h}_k^T\mathbf{h}_k + \sigma_n^2}, \gamma_{IC,k} \doteq \frac{\mathbf{h}_k^T\mathbf{h}_k}{\mathbf{h}_k^T\mathbf{h}_k + \sigma_n^2}, \text{ and}$$
$$\beta_{IC,k} \doteq \frac{\gamma_{IC,k}}{1-\gamma_{IC,k}},$$

where $j = 1 + (i \mod 2)$ (i.e. the opposite channel from k), and $\bar{a}_j$ is the value of a_j for which $\left| \frac{\hat{x}_{MMSE,j}}{\gamma_{jj}} - a_j \right|^2, \mathbf{a} \in C$ is minimized (C is the set of all constellation points, i.e. we choose the one-dimensional point on the axis corresponding to j which is closest to $\frac{\hat{x}_{MMSE,j}}{\gamma_{jj}}$).

The algorithms discussed in this section were implemented on the GPU mentioned in section 2 for further evaluation. These CUDA implementations are discussed further in the following section.

4 Implementation

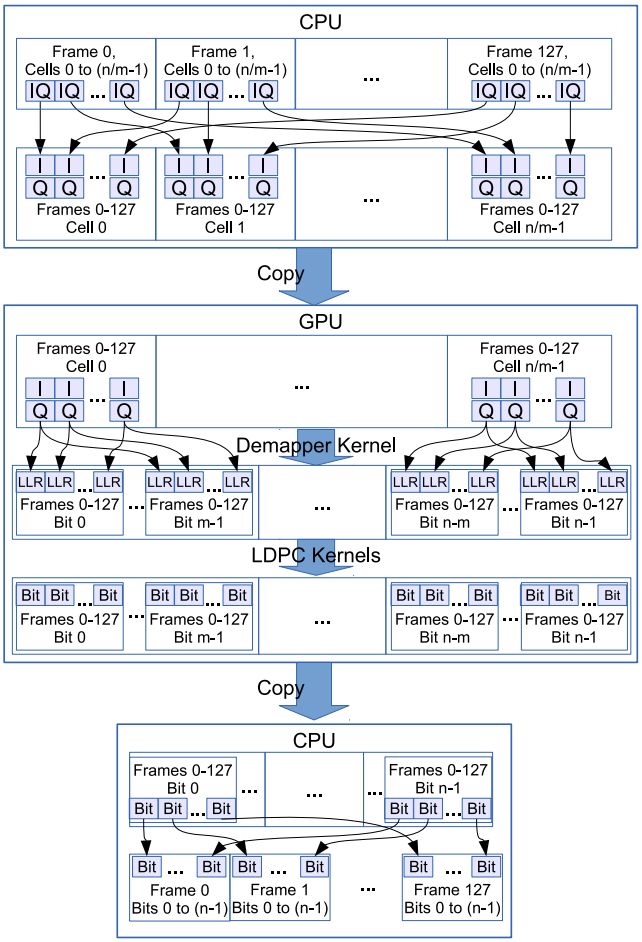


Fig. 5 Illustration of the arrangement of the main input samples in memory at different stages of running the demapper and LDPC decoder on the GPU. First, in the main host memory, input samples are in the form of complex I-Q samples in arrival order, after which they are interleaved and split into separate arrays for I and Q samples. These arrays are copied to the GPU memory. On the GPU, the demapper processes these samples and produces LLR values for the LDPC decoder. The LDPC decoder produces hard bits as its output. These bits are transferred back to the host memory, and deinterleaved.

As mentioned, our implementations of the demapper algorithms were realized on an NVIDIA CUDA-based GPU. The GPU kernels were written in the C language. The input to the demapper are in the form of complex cell values where both components are 32-bit floating point values. This precision is also retained within the GPU kernels. In these implementations, the output LLR values are however converted to 8-bit fixed point values, due to the fact that the GPU-based LDPC decoder uses this LLR format [9]. This conversion may not be necessary if another, high-precision, LDPC decoder is used. The GPU implementations operate on cells belonging to 128 DVB-T2 FEC (forward error correction) frames in parallel. Each FEC frame corresponds to 16200 or 64800 bits, depending on if short or long LDPC codewords are used [6].

Fig. 5 illustrates the arrangement of the main input values in memory at various stages of performing demapping and LDPC decoding on the GPU. Other values, such as estimated noise figures and channel fading factors are arranged similarly. As seen in the figure, the input I-Q samples, which are arranged in the order of arrival, are interleaved such that the first cell (sample) of all 128 FEC frames are arranged consecutively in memory, followed by the second cell of all codewords, etc. There are n/m cells per FEC frame if we use 2^m-QAM and a FEC frame length of n. The I and Q samples are also separated into separate arrays at this stage. After interleaving, the samples are transferred to the GPU memory. Running the demapper kernels produces LLR values for each of m bits per cell. The LDPC decoder produces hard bits as its output. After copying the bits back to host memory, we deinterleave the bits back into the original sample ordering.

GPU threads were created such that the 32 threads of a thread warp would operate on cells from 32 consecutive FEC frames. This combined with organizing data in memory, as illustrated in Fig. 5, such that the data needed for 32 consecutive FEC frames are also consecutive in memory, gives good memory coalescence. This arrangement of thread blocks and memory does not significantly affect demapper throughput (as compared to the input ordering), since the memory access patterns during demapping are the same between neighboring cells in the same FEC block as between the same cell in different FEC blocks. It does, however, greatly improve LDPC decoder throughput [9], where decoding the same cell of different FEC blocks results in very similar memory access patterns, while decoding neighboring cells results in scattered memory accesses with fewer opportunities for memory access coalescing.

Thread blocks were set to be 256 threads in size, i.e. one thread block contains threads demapping 2 cells from all 128 frames. The remainder of this section provides some implementation details for the algorithms presented in section 3.

4.1 ML Demapper and Max-Log Demapper

The implementation of the full maximum likelihood demapper described by (1) was implemented roughly as follows. First we loop over each constellation point $\mathbf{x} \in C$, where the expression $d := e^{-\frac{D(\mathbf{x})}{2\sigma^2}}$ is calculated. For 2^m-QAM, $2m$ sums, $\mathbf{S} = [s_{u,v}]_{m\times 2}$, were needed to store the two sums for each of the m LLRs. In each loop iteration, we then perform $s_{i,k} := s_{i,k} + d, \forall i \in [1, m]$ — where k is 1 if $\mathbf{x} \in C_i^0$, and 2 if $\mathbf{x} \in C_i^1$ — in an inner loop.

After this main loop, we calculate

$$LLR(b_i) := ln(s_{i,2}/s_{i,1}), \forall i \in [1, m],$$

where $LLR(b_i)$ is converted to a fixed point 8-bit value.

The implementation of the max-log demapper is quite similar. Here, for each constellation point, let $d := D(\mathbf{x})$, and we use $\mathbf{S}$ to store the smallest values of d instead of accumulating sums, i.e. $\forall i \in [1, m] : s_{i,k} := d$ iff $d < s_{i,k}$. In the final loop, we then calculate $LLR(b_i) := (s_{i,1} - s_{i,2})/2\sigma^2, \forall i \in [1, m]$.

4.2 MMSE and MMSE-IC Demapper

Two demappers based on MMSE decorrelation were implemented for comparison. The first implementation calculates all LLRs according to (4), while the second implementation performs IC and calculates the LLRs for the bits conveyed by the worst channel according to (5). Both of these implementations consist of two separate GPU kernels. In the MMSE-only case, one kernel calculates the LLRs corresponding to the I channel, and the other calculates LLRs corresponding to the Q channel. When IC is used, one kernel calculates LLRs corresponding to the strongest of the two channels, while the other calculates LLRs for the weaker channel. This makes code for each kernel shorter, and avoids branching instructions in the kernels.

Furthermore, for both implementations, two one-dimensional lookup tables were used for calculating the LLRs after MMSE decorrelation. The tables contain the value

$$\min_{\mathbf{a}\in C_i^0} \left| \frac{\hat{x}_{MMSE,k}}{\gamma_{kk}} - a_k \right|^2 - \min_{\mathbf{a}\in C_i^1} \left| \frac{\hat{x}_{MMSE,k}}{\gamma_{kk}} - a_k \right|^2$$

for 1024 (in our case) values of $\frac{\hat{x}_{MMSE,k}}{\gamma_{kk}}$, where one table contains the values for $k = 1$ and the other for $k = 2$. Each table thus contains $1024 * m/2$ entries when using 2^m-QAM. The large table size was chosen for precision, as smaller sizes were not found to affect demapping speed significantly.

5 Measurement results

In this section, measured throughputs of the implemented demapper algorithms are presented, along with simulation results that measure BER (bit error rate) performance at certain SNR (signal-to-noise ratio) values.

5.1 Throughput measurements

Within the DVB-T2 physical layer simulator used for testing the implementations, the OFDM cells were transferred to the GPU, after which the demapping of cells belonging to 128 FEC frames was performed. The output LLR values were not transferred back to the host, as the LDPC decoder was also implemented as GPU kernels. Thus, only the output hard-decision bits of the LDPC decoder were transferred back to the host for further processing, as illustrated in Fig. 5. The bit-deinterleaver operation, which is normally performed after demapping (see Fig. 3), was postponed until after the LDPC decoder, and was implemented using lookup tables on the host CPU. Postponing the bit-deinterleaver is possible if one makes the LDPC decoder operate on interleaved LLR values and bits by appropriately interleaving the columns of the parity-check matrix defining the LDPC code.

Table 1 Execution times (in seconds) of the various demapper algorithms on the GPU, given 16-QAM, 64-QAM, and 256-QAM modulation schemes.

Modulation	ML	Max-Log	MMSE	MMSE-IC
16-QAM	0.0101	0.0070	0.0028	0.0032
64-QAM	0.0255	0.0218	0.0022	0.0024
256-QAM	0.0832	0.0788	0.0019	0.0021

Table 2 Throughput (in Mbps) of the combined demapper and LDPC decoder operation on the GPU. This figure also includes copying of data between the host and GPU.

Modulation	ML	Max-Log	MMSE	MMSE-IC
16-QAM	70.8	72.5	75.4	75.0
64-QAM	64.1	66.0	78.2	78.1
256-QAM	44.9	45.9	80.0	80.0

Table 1 shows the measured execution times in seconds to process one batch of 128 FEC frames of the four implemented GPU-based demappers on the test setup. This measured time is the average over 10 blocks of 128 FEC frames. As the long LDPC codeword length was used, each FEC frame contains 64800 bits of data. The

LDPC decoder (as described in [9]), running 30 iterations of the message passing decoding algorithm, had a run time of approximately 0.09 seconds for each block of 128 FEC-frames. Table 2 shows the average throughput in Mbps when running both the demapper and LDPC decoder on the GPU. Included in these measurements are the demapper, the LDPC decoder, as well as copying the data between host and GPU. The throughput has been calculated as $(128 * 64800)/t$ bps, where t is the total execution time for 128 FEC frames.

We can see from Table 1 that the MMSE-based algorithms are roughly 40 times faster than the ML algorithm and its max-log approximation when using 256-QAM. This is expected, and is largely due to the fact that we can calculate distances in one dimension after MMSE decorrelation, as well as due to the use of lookup tables. Note that, as opposed to the ML and max-log algorithms, the MMSE implementations decrease slightly in speed with lower order modulations. This is most probably due to the need for running a larger amount of total threads with lower order constellations, due to each cell carrying fewer bits, which increases the complexity per bit of the MMSE decorrelation. We can also see that the advantage of the MMSE implementations decreases dramatically with lower modulation order. At 16-QAM, the fastest (i.e. non-IC) MMSE algorithm is only 2.5 times faster than the max-log implementation. This is also to be expected, as the number of distances calculated by the ML and max-log implementations are very low at this setting. Furthermore, a max-log demapper using subsets as proposed in [16] was also implemented and measured for 256-QAM, where each subset contained 144 of the 256 constellation points (the subset size shown in Fig. 4). This yielded an execution time of 0.0538s for the demapper, and an overall throughput of 53.3 Mbps.

Annex C of the DVB-T2 standard assumes that received cells can be read from a deinterleaver buffer at 7.6×10^6 OFDM cells per second [3,6]. At the 16-QAM, 64-QAM, and 256-QAM modulation settings, we can represent 4, 6, and 8 bits per cell respectively. This means that the demodulator should be able to perform at a maximum bitrate of up to 60.8 Mbps (Megabits per second) in the 256-QAM case, as well as 30.4 and 45.6 Mbps in the 16-QAM and 64-QAM cases, respectively. These throughput requirements are met with all demapper implementations when using 16-QAM or 64-QAM modulation. In the case of 256-QAM, however, the maximum required throughput was exceeded only using the fast MMSE-based implementations.

One may further affect the combined throughput by lowering or increasing the maximum amount of LDPC iterations used. In [9], however, it is shown that error correction performance is quite significantly deteriorated once we lower the amount of iterations below 30. In order to gain better error correction performance in difficult channel conditions, we may however wish to increase the amount of iterations in case we exceed the required throughput.

5.2 Simulated BER performance

In the previous subsection we showed that the simplified demapping algorithms do result in significantly higher throughputs when implemented on the GPU. The simplifications do also, however, have a negative impact on the BER performance. In order to analyze the throughput-accuracy tradeoff, we simulated the various GPU implementations in a DVB-T2 physical layer simulator.

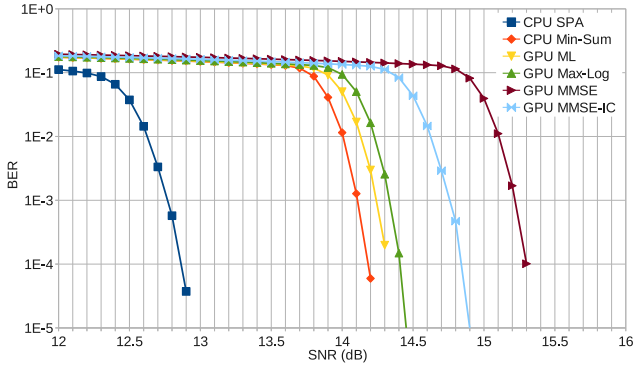
The simulator was set to use the Rayleigh-fading ("P1") channel model specified in [4, 5]. The bit error rate measured was the remaining BER after BCH decoding (i.e. after all stages of the demodulator). In addition to simulations for all GPU demapper implementations, we also include reference simulation results produced by CPU implementations that use the ML demapper.

While the LDPC decoder used in all GPU measurements is the min-sum decoder with 8-bit internal numeric precision as described in [9], we include reference CPU measurements using both the optimal sum-product algorithm (SPA), and the min-sum algorithm, both using 32-bit floating point numbers internally. In addition to higher numeric precision, the CPU implementations also run up to 50 LDPC decoder iterations, as opposed to a maximum of 30 for the GPU implementations (30 iterations was chosen in order to relate to the throughput measurement setup in the previous subsection). The CPU measurements using the SPA shows the performance when both the mapper and LDPC decoder are close to optimal, while the difference between the CPU measurements using min-sum and GPU measurements with the ML demapper give an indication of the impact of the limited numeric precision and limited number of iterations on BER performance. Since all GPU implementations use the same LDPC decoder, the difference between these are the most important as they show the difference between the various demapper implementations.

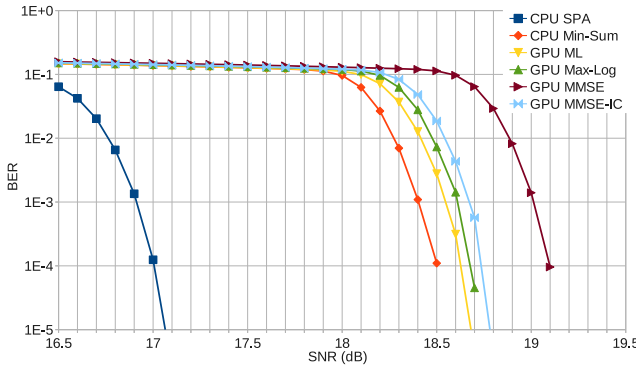
In Fig. 6(a), we present the measurement results for 64-QAM modulation at an LDPC code rate (ratio of information bits to total number of bits including parity bits) of 1/2 in the simulated Rayleigh channel. The CPU implementation using the SPA LDPC decoder and ML demapper is clearly the most accurate. We can also

see that switching to the min-sum LDPC decoder in the CPU implementation carries a penalty of roughly 1.3 dB at a BER level of 10^{-4}, while the penalty for using the lower precision GPU min-sum decoder instead of the floating point CPU min-sum decoder is less than 0.2 dB in this case. Furthermore, the GPU implementations of the ML and Max-Log algorithms perform within 0.1 dB of each other. The penalty for using MMSE-IC over ML is roughly 0.5 dB. Leaving out the interference cancellation in the MMSE approach appears to carry a relatively large performance penalty of around 0.5 dB.

The results for 256-QAM (also 1/2-rate), as presented in Fig. 6(b), differ from the 64-QAM case to some extent. The performance gap between the CPU implementations using the SPA and min-sum decoders is slightly larger, at around 1.5 dB, than in the 64-QAM case. In the 256-QAM case, however, the penalty for using MMSE-IC over ML is less than 0.2 dB, and the penalty for leaving out IC is slightly lower at less than 0.4 dB.



(a) 64-QAM, 1/2-rate



(b) 256-QAM, 1/2-rate

Fig. 6 Measurement results for 64-QAM and 256-QAM modulation with an LDPC code rate of 1/2. The two CPU implementations use the SPA and min-sum LDPC decoders (with a maximum of 50 iterations), respectively, together with the ML demapper. The GPU implementations use a lower precision min-sum LDPC decoder (with a maximum of 30 iterations) and the various demapper implementations discussed in section 4.

It is worth noting that both the reference CPU implementations are too slow to consider when real-time performance is required. Based on the measurements shown in Fig. 6, the MMSE-IC approach appears to be a good tradeoff between accuracy and speed, at least when using 256-QAM, as it yields almost double the combined LDPC and demapper throughput (see Table 2) of the ML and Max-Log implementations at an accuracy penalty below 0.5 dB SNR. Given the larger penalty when using 64-QAM compared to 256-QAM, the penalty might be even higher in lower modulation orders. This is however a minor issue, as the throughput of the ML and Max-Log implementations increase rapidly with lower modulation order. An SDR implementation could therefore choose the appropriate demapper depending on the channel parameters and conditions.

6 Conclusion

In this article, we have implemented and compared various demapping algorithms for rotated QAM constellations on a modern GPU. Benchmarks show that if a fast demapping algorithm is chosen, the demapper may share the GPU with an LDPC decoder while fulfilling the maximum required throughput requirements of the standard, even using the most complex 256-QAM mode. We have also shown that for up to 64-QAM, we may be able to run even the optimal, most complex, ML demapper, while still reaching the throughput target.

Furthermore, we measured the penalties in terms of BER of various simplifications of the demapper. These measurements did show a clear difference in accuracy between the ML or Max-Log implementations of the demapper and the MMSE-based implementations. The difference remained smaller than 0.5 dB in both the 64-QAM and 256-QAM cases if MMSE with interference cancellation was used, however.

In the future, we hope to integrate the decoder implementations with other software defined signal processing blocks to build a completely software defined, real-time, receiver chain.

References

1. Bae, K., Kim, K., Yang, H.: Low complexity two-stage soft demapper for rotated constellation in DVB-T2. In: Consumer Electronics (ICCE), 2012 IEEE International Conference on, pp. 618 –619 (2012). DOI 10.1109/ICCE.2012.6162044
2. Boutros, J., Viterbo, E.: Signal space diversity: a power- and bandwidth-efficient diversity technique for the Rayleigh fading channel. Information Theory, IEEE

Transactions on **44**(4), 1453 –1467 (1998). DOI 10.1109/18.681321
3. Draft ETSI TR 102 831 V0.10.4: Implementation guidelines for a second generation digital terrestrial television broadcasting system (DVB-T2). ETSI Technical Report (2010)
4. DVB BlueBook A133: Implementation guidelines for a second generation digital terrestrial television broadcasting system (DVB-T2). DVB Technical Report (2009)
5. ETSI EN 300 744 v1.6.1: Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television (DVB-T). ETSI Technical Report (2009)
6. ETSI EN 302755 v1.1.1: Digital Video Broadcasting (DVB); Frame Structure Channel Coding and Modulation for a Second Generation Digital Terrestrial Television Broadcasting System (DVB-T2). ETSI Technical Report (2009)
7. Gallager, R.: Low-Density Parity-Check Codes. Ph.D. thesis, M.I.T. (1963)
8. Grönroos, S., Nybom, K., Björkqvist, J.: Complexity analysis of software defined DVB-T2 physical layer. Analog Integrated Circuits and Signal Processing **69**, 131–142 (2011). DOI: 10.1007/s10470-011-9724-4
9. Grönroos, S., Nybom, K., Björkqvist, J.: Efficient GPU and CPU-Based LDPC Decoders for Long Codewords. . Analog Integrated Circuits and Signal Processing **73**, 583–595 (2012)
10. Kim, K., Bae, K., Yang, H.: One-dimensional soft-demapping using decorrelation with interference cancellation for rotated QAM constellations. In: Consumer Communications and Networking Conference (CCNC), 2012 IEEE, pp. 787 –791 (2012). DOI 10.1109/CCNC.2012.6181165
11. Li, M., Nour, C., Jego, C., Douillard, C.: Design of rotated QAM mapper/demapper for the DVB-T2 standard. In: Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on, pp. 18–23 (2009). DOI 10.1109/SIPS.2009.5336265
12. NVIDIA: GeForce GTX 570. http://www.nvidia.com/object/product-geforce-gtx-570-us.html. Accessed June 2011.
13. NVIDIA: NVIDIA's Next Generation CUDA Compute Architecture: Fermi. Whitepaper, http://www.nvidia.com (2009)
14. NVIDIA: NVIDIA GeForce GTX 570 GPU Datasheet. Datasheet, http://www.nvidia.com (2010)
15. NVIDIA: CUDA C Programming Guide v.4.0. http://www.nvidia.com (2011)
16. Pérez-Calderón, D., Baena-Lecuyer, V., Oria, A., López, P., Doblado, J.: Rotated constellation demapper for DVB-T2. Electronics Letters **47**(1), 31 –32 (2011). DOI 10.1049/el.2010.2682
17. Vangelista, L., Benvenuto, N., Tomasin, S., Nokes, C., Stott, J., Filippi, A., Vlot, M., Mignone, V., Morello, A.: Key technologies for next-generation terrestrial digital television standard DVB-T2. Communications Magazine, IEEE **47**(10), 146–153 (2009). DOI 10.1109/MCOM.2009.5273822