DVB-T2 ROTATED CONSTELLATION DEMAPPING ON A GPU

Stefan Grönroos (Turku Centre for Computer Science TUCS/Åbo Akademi University, Turku, Finland; stefan.gronroos@abo.fi); Kristian Nybom (Åbo Akademi University, Turku, Finland; kristian.nybom@abo.fi); Jerker Björkqvist (Åbo Akademi University, Turku, Finland; jerker.bjorkqvist@abo.fi)

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 paper, 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 of 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.

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 [1, 2] has been developed to address this need. This standard offers significantly increased capacity (bitrate) 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 [3]. For channel coding, the standard specifies the use of LDPC (low-density parity-check) codes [4] 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 (SSD)[5, 6] and gives improved performance in some fading channels.

The authors of this paper 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) [7]. In a step towards creating a real-time capable software define radio (SDR) implementation of a DVB-T2 receiver on a general purpose computer, we focus on implementing fast rotated constellation demappers in this paper. 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 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 paper, we examine the implementations of several demapping algorithms, measuring their performance in terms of speed. 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.

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 [6, 8–11]. In addition to the maximum likelihood (ML) demapper as well as its max-log simplification [6, 8], 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 paper 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 results are presented in section 5 along with discussion regarding the obtained results. The paper 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[12] 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.

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 processors on a GPU. The GPU used for running the LDPC decoder implementation described in this paper was an NVIDIA GeForce GTX 570 [13, 14], which is based on the Fermi architecture [15], and features 15 streaming multiprocessors (SMs) containing 32 cores each. The scheduler schedules threads in groups of 32 threads, 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 same 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 if the memory locations accessed by the threads are scattered [12]. 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 Rotated Constellations

In Figure 1, a block diagram of the BICM (bit interleaved coding & modulation) module of a DVB-T2 modulator is shown. DVB-T2 [1, 8] supports the use of QPSK, 16-QAM, 64-QAM, and 256-QAM modulation. After gray mapping of bit sequences to constellations, the constellation diagram may optionally be rotated to provide SSD. If rotation is enabled, the Q component is also cyclically delayed by one OFDM (orthogonal frequency-division multiplexing) cell. As seen in Figure 1, a cell interleaver in which the mapped OFDM cells are further interleaved follows the Q-delay, thus separating the I and Q components further. 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.

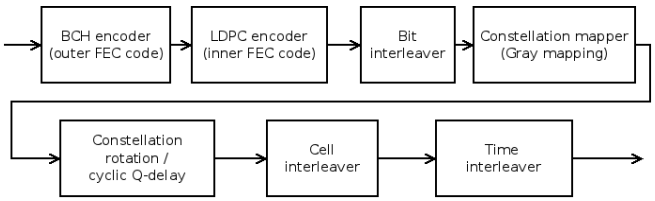

Figure 1: Block diagram of the BICM module of a DVB-T2 modulator.

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 [8]:

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

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 received OFDM cell, 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 equation 1.

To simplify equation 1, one can apply the approximation [8]:

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

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 equation 3 does remove some complexity, we still need to calculate two-dimensional distances to 2^m points in the case of 2^m-QAM modulation. The authors of [9] discuss the possibility of assigning the constellation points of a rotated constellation diagram into four overlapping subsets. This is illustrated in Figure 2, where one proposed subset is shown (within the shaded area) for a 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 Figure 2 was demonstrated [9] 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 [6] 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):

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

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 equation 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 equation 4 with lookup tables [10].

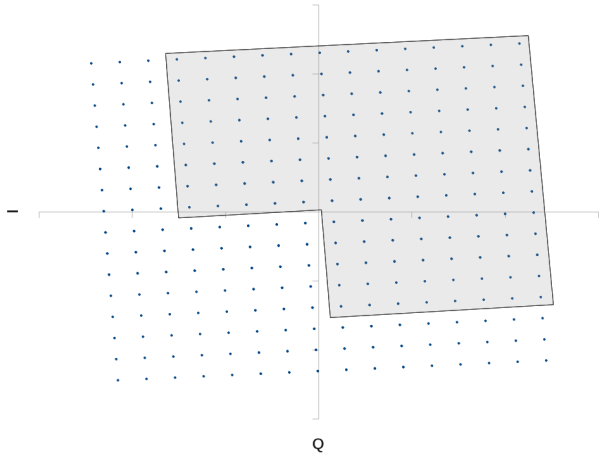

Figure 2: One of four overlapping subsets of a 256-QAM rotated constellation diagram, as proposed in [9].

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 eq. 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

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 was in the form of complex cell values where both components were 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 [7]. 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 [1].

In order to benefit from the advantages of memory access coalescing, 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 such that the data needed for 32 consecutive FEC frames are also consecutive in memory, gives good memory coalescence. 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 equation 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 equation 4, while the second implementation performs IC and calculates the LLRs for the bits conveyed by the worst channel according to equation 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 1-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. Within the DVB-T2 physical layer simulator used for testing the implementations, the OFDM cells were transfered to the GPU, after which the demapping of cells belonging to 128 FEC frames was performed. The output LLR values were not transfered 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 transfered back to the host for further processing. The bit-deinterleaver operation, which is normally performed after demapping (see Figure 1), 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 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 [7]), 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 [9] was also implemented and measured for 256-QAM, where each subset contained 144 of the 256 constellation points (the subset size shown in Figure 2). 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 [1, 8]. 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 [7], 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.

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.

While the penalties in terms of BER (bit error rate) at a certain SNR (signal-to-noise ratio) of the various simplifications of the demapper were explored to some extent in, for instance [6, 10], more thorough simulations should still be performed on the implementations presented in this paper. This would give further insight into the accuracy-throughput tradeoff of the various demappers. These simulations were not a focus of this paper due to incomplete models of fading channels in the DVB-T2 physical layer simulator used to test the implementations. In the future, we also hope to integrate the decoder implementations with other software defined signal processing blocks to build a completely software defined, real-time, receiver chain.

7. REFERENCES

[1] ETSI EN 302 755 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.

[2] L. Vangelista, N. Benvenuto, S. Tomasin, C. Nokes, J. Stott, A. Filippi, M. Vlot, V. Mignone, and A. Morello. Key technologies for next-generation terrestrial digital television standard DVB-T2. *Communications Magazine, IEEE*, 47(10):146–153, Oct. 2009.

[3] S. Grönroos, K. Nybom, and J. Björkqvist. 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.

[4] R. Gallager. *Low-Density Parity-Check Codes*. PhD thesis, M.I.T., 1963.

[5] J. Boutros and E. Viterbo. Signal space diversity: a power- and bandwidth-efficient diversity technique for the Rayleigh fading channel. *Information Theory, IEEE Transactions on*, 44(4):1453 –1467, jul 1998.

[6] Meng Li, C.A. Nour, C. Jego, and C. Douillard. Design of rotated QAM mapper/demapper for the DVB-T2 standard. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 18–23, Oct. 2009.

[7] S. Grönroos, K. Nybom, and J. Björkqvist. Efficient GPU and CPU-based LDPC decoders for long codewords. *Analog Integrated Circuits and Signal Processing*, 2012. DOI: 10.1007/s10470-012-9895-7.

[8] 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.

[9] D. Pérez-Calderón, V. Baena-Lecuyer, A.C. Oria, P. López, and J.G. Doblado. Rotated constellation demapper for DVB-T2. *Electronics Letters*, 47(1):31 –32, 6 2011.

[10] Kyeongyeon Kim, Kitaek Bae, and Ho Yang. One-dimensional soft-demapping using decorrelation with interference cancellation for rotated QAM constellations. In *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, pages 787 –791, jan. 2012.

[11] Kitaek Bae, Kyeongyeon Kim, and Ho Yang. Low complexity two-stage soft demapper for rotated constellation in DVB-T2. In *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, pages 618 –619, jan. 2012.

[12] NVIDIA. CUDA C Programming Guide v.4.0. http://www.nvidia.com, 2011.

[13] NVIDIA. NVIDIA GeForce GTX 570 GPU Datasheet. Datasheet, http://www.nvidia.com, 2010.

[14] NVIDIA. GeForce GTX 570. http://www.nvidia.com/object/product-geforce-gtx-570-us.html. Accessed June 2011.

[15] NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Fermi. Whitepaper, http://www.nvidia.com, 2009.