COMPLEXITY ANALYSIS OF SOFTWARE DEFINED DVB-T2 PHYSICAL LAYER

Stefan Grönroos (Åbo Akademi University, Turku, Finland; stefan.gronroos@abo.fi);
Kristian Nybom (Åbo Akademi University, Turku, Finland; kristian.nybom@abo.fi); and
Jerker Björkqvist (Åbo Akademi University, Turku, Finland; jerker.bjorkqvist@abo.fi)

ABSTRACT

The second generation terrestrial TV broadcasting standard from the Digital Video Broadcasting (DVB) project, DVB-T2, has recently been standardized. In this paper we will perform a complexity analysis of our GNU Radio based implementation of the modulator/demodulator parts of a DVB-T2 transmitter and receiver. First we describe the various stages of a T2 modulator and demodulator, as well as how they have been implemented in our system. We then perform an analysis of the computational complexity of each signal processing block. The complexity analysis is performed in order to identify the blocks that are not feasible to run in realtime on a general purpose processor. Furthermore, we discuss possibilities of implementing these computationally heavy blocks on other architectures, such as GPUs (Graphics Processing Units) and FPGAs (Field-Programmable Gate Arrays), that would still allow them to be implemented in software and thus be easily reconfigurable.

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.

Compared to its predecessor, DVB-T2 has a more efficient physical layer using state-of-the-art technologies to achieve close to optimal performance in terms of true bit-rate in quasi error free conditions: concatenated LDPC (Low-Density Parity-Check) and BCH (Bose-Chaudhuri-Hocquenghem) coding, rotated high-order QAM constellations, MISO (Multiple Input Single Output) antenna reception, efficient time and frequency interleaving, large FFT sizes, etc. All in all, DVB-T2 is expected to give an increase in capacity (bit rate) of at least 30% as compared to DVB-T, and for some configurations up to 70%[2]. The upcoming next generation mobile TV broadcasting system, DVB-NGH (Next Generation Handheld), is also expected to be based on DVB-T2.

In this paper, we will present a work-in-progress software defined DVB-T2 modulator and demodulator using the GNU Radio framework [3]. In addition to other benefits of a fully software defined implementation of DVB-T2, the reconfigurability of such an implementation can be very beneficial in developing future standards such as DVB-NGH. As mentioned, the GNU Radio based project is incomplete, and a number of parts have not yet been finished. However, the implementation is directly based on a DVB-T2 simulator, which is more complete.

We will examine the computational complexity of the various parts of a DVB-T2 modulator and demodulator using benchmarks performed on the already implemented GNU Radio signal processing blocks, as well as on the simulator blocks. We are not comparing the results to ASIC (Application-Specific Integrated Circuit) implementations, as our main motivation for this work is to analyze the applicability of a DVB-T2 system on a generic SDR (Software Defined Radio) platform.

We will also discuss alternative implementations of the most computationally complex blocks on platforms such as FPGAs (Field-Programmable Gate Arrays) and GPUs (Graphics Processing Units), which may allow us to reach realtime performance, while still retaining the reconfigurability of a software defined implementation.

Related work can be found in [4], where a GNU Radio implementation of a DVB-T modulator is described. An SDR implementation of a DVB-T2 receiver is described in [5], where most of the system has been realized using FPGAs and DSPs (Digital Signal Processors). In contrast, our implementation aims at keeping most functionality, if possible, on general purpose, commodity hardware.

The paper is laid out as follows. In section 2 we will describe the various parts of a DVB-T2 system. In section 3 we introduce the experimental setup on which measurements were performed. Section 4 contains the results of the measurements, while section 5 is reserved for discussion of the viability of the measured blocks for

realtime performance. Finally, we conclude the paper in section 6.

2. A DVB-T2 SYSTEM

In this section, we will describe the main building blocks of a DVB-T2 modulator, as defined in [1]. These are the blocks that were benchmarked for this paper.

The input data streams, which are in the form of MPEG2 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. In this paper, we will only consider a single-PLP system. The first module of such a system is the Input Processing module. This module converts the input data streams into DVB-T2 baseband frames. This module will not be discussed further in this paper, however. After passing through the input processing module, each baseband frame is processed by the Bit Interleaved Coding & Modulation (BICM) module, which 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 FEC 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.
- Mapper, which maps bits onto constellations. Produces cell words.
- 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 cell words of a FEC block.
- Time Interleaver. The cells of groups of FEC blocks, making up TI-blocks – which in turn make up Interleaving Frames – are interleaved.

The BICM module is followed by the Frame Builder, the task of which is to assemble so-called T2 frames from the Interleaving Frames of each PLP, as well as various signalling data. The modulated cells that are going to be included in one OFDM (Orthogonal Frequency-Division Multiplexing) symbol are grouped together in this module. The Frame Builder module also includes frequency interleaving, where the cells belonging to an OFDM symbol are interleaved, providing interleaving in the frequency domain.

The frames produced by the Frame Builder are sent to the OFDM Generation module for further processing. The OFDM Generation module includes the following parts:

- MISO processing. This is optional, and allows for the generation of two slightly different output signals for transmission from two groups of transmitters.
- Pilot Insertion. Cells containing reference information are inserted at, to the receiver, known points in the transmitted signal. Pilots can be, among other uses, used to aid in synchronization and channel estimation at the receiver.
- IFFT (Inverse Fast Fourier Transform). This is where the OFDM symbols are modulated. FFT sizes of 1K, 2K, 4K, 8K, 16K, and 32K are supported.
- PAPR reduction. This optional part allows us to reduce the Peak-to-Average Power Ratio (PAPR) of the transmitted signal.
- Guard interval insertion. This is where we insert guard intervals, which are a cyclic continuation of the useful part of an OFDM symbol.
- P1 symbol insertion. P1 symbols are special 1K OFDM symbols that are used mainly to aid the receiver in recognizing and tuning in to the T2 signal.

The output of the OFDM module is a signal ready for transmission. In the following section, we will discuss how the partial implementation in GNU Radio and the T2 simulator were used to benchmark the various functional blocks of a DVB-T2 system.

3. EXPERIMENTAL SETUP

As mentioned in the introduction, we have implemented some of the functional blocks discussed in section 2 within the GNU Radio environment. The actual functionality is written in C++ and is directly based on the building blocks of a DVB-T2 simulator developed at Åbo Akademi University. The blocks that had not yet been implemented in GNU Radio were run in the simulator instead. The simulator was not 100% complete, and lacked support for some configurations such as multiple PLPs, some pilot patterns etc. Neither the simulator nor the GNU Radio implementation included the necessary support for tuning in to a T2 channel, nor for various forms of receiver synchronization at the time when this paper was written.

The GNU Radio implementation included the full BICM module. The time interleaver, however, was benchmarked using the simulator due to some memory related issues with that block in the GNU Radio implementation. The Frame Builder and OFDM Generation modules, in addition to the time interleaver, were benchmarked within the simulator.

The benchmarking was performed on an Ubuntu Linux operating system using the Linux 2.6.32 kernel. The laptop computer in question was equipped with an Intel Core 2 Duo T7100 dual core CPU running at 1.8 GHz. During benchmarking, multithreading was not used within the measured functionality (and thus only one CPU core was

exploited). 3GB of DDR2 RAM at 666 MHz was available in the system.

The used technology was already a few years old when this paper was written. Thus, the benchmarks in the following section would quite certainly be significantly improved on top-of-the-line equipment.

The measurements were made by using the clock_gettime function to return the time before and after execution of what was considered to be the core functionality of a functional block, such as a main loop. The impact of additional memory transfers between blocks and similar setup operations were mostly ignored.

The efficiency of the code has been considered when the blocks were written. The code does however not contain low level optimizations, such as optimized inline assembly code. Some blocks could also possibly benefit from the use of lookup tables instead of calling mathematical functions.

The possibility for further optimizations combined with the somewhat dated hardware used, means that the benchmarks presented in the following section should be seen mainly as indications of the relative complexity of the involved functional blocks, as well as indications of the feasibility of running the blocks on general purpose CPUs. In the following section, we present the results of benchmarking the main functionality of the implemented parts of a T2 system.

4. BENCHMARK RESULTS

In this section the results of the benchmarked algorithms are presented. As DVB-T2 offers quite many customization possibilities, we fixed most parameters in the configurations used for the benchmarks. We used pilot pattern 1, and a guard interval of ¼ throughout the benchmark tests. Also only the 8K FFT mode was considered.

Tables 1 and 2 show the throughputs of the various blocks in the modulator and demodulator, respectively. The throughput was measured by timing the core functionality of the block, and dividing the time used for processing one FEC block by the size of the FEC block (16200 bits for short code length, and 64800 bits for long code). Thus, the throughput measure does not give the actual useful bitrate, but rather the bitrate including parity data. To gain an approximate useful bitrate, the throughput figure must be multiplied by the code rate.

It is worth noting that the BCH and LDPC encoder and decoder functionality depends on code rate. The throughputs in tables 1 and 2 are measured for code rate 1/2. It was found that the BCH encoder's throughput was roughly halved from the lowest code rate, 1/2, to the highest, 5/6. The LDPC encoder was found to not vary significantly in performance between code rates. A BCH decoder had not been completely implemented at the time of writing, which is the reason for the missing measurements in table 2.

In order to gain a clearer view of which blocks are suitable for realtime operation, it can be mentioned that using the 8K FFT mode with the extended carrier mode enabled, each OFDM symbol contains 6296 data cells, with each cell representing 2, 4, 6, or 8 bits for QPSK, 16-QAM, 64-QAM, and 256-QAM modulation, respectively. The maximum amount of OFDM symbols in a frame using the 8K mode, and a guard interval of 1/4 (as in the experiments), is 223. The duration of such a DVB-T2 frame is 250ms. From the above information, we can calculate that in order to fully process such frames, we will require throughputs of roughly 22.5 and 45 Mbps for 16-QAM and 256-QAM, respectively. The throughputs of the modulator and demodulator (as presented in tables 1 and 2) divided by these "target" throughputs for realtime performance, expressed as percentage values on a logarithmic scale, are presented in figures 1 and 2. In the figures, a percentage value at or above 100% thus means that a block has a throughput at or above the required realtime throughput.

The following section is dedicated to the discussion of these results, as well as optimization strategies and alternative implementation platforms for the most computationally costly blocks.

5. DISCUSSION

In this section we discuss the results presented in the last section, and identify which blocks might not be suitable for execution on general purpose processors.

It is worth noting that the benchmarked throughputs indicate the throughput that the main functionality of the block is capable of if it is run repeatedly with no other blocks competing for CPU time. In an actual system, these blocks will need to share the computing resources available. At the time of writing, it was however possible to purchase server grade hardware using dual 6-core CPUs, with each CPU core running at a higher clock frequency than the CPU of the test hardware. In such a system, each block could, if necessary, be given exclusive access to at least one CPU core.

From figure 1, we can see that the BCH and LDPC blocks were the slowest blocks of the modulator with less than 20% of the target bitrate for BCH and less than 25% for LDPC in the slowest case, i.e. 256-QAM with long code length. With 16-QAM and short code length, these blocks performed at about 40% and 80% of the target, respectively. While these figures are quite low, the combination of further code optimizations and faster processors might still be able to make these blocks capable of realtime performance. The other modulator blocks showed at least close to realtime performance, where most blocks surpassed realtime performance.

It is on the demodulator side that larger problems emerged. Here also, most blocks operated at close to or

above realtime performance, except for the constellation demapper and LDPC decoder. These blocks performed orders of magnitude below the realtime target. The demapper was only capable of a throughput of around 100 kbps at 256-QAM, and about 600 kbps at 16-QAM. The block would require a speedup of roughly 500 times in order to be able to handle 256-QAM in a realtime system.

Further profiling of the demapper revealed that the Log-Likelihood Ratio (LLR) estimation is responsible for the high complexity here. The LLR estimation algorithm used in our current implementation – where each bit of each received cell gets assigned a likelihood value based on calculated distances to each point in the constellation diagram – involves several computationally intensive operations per bit (such as logarithms and exponents, as well as multiplications and divisions).

Table 1: Modulator throughput (Mbps)

	Short code		Long code	
	16-QAM	**256-QAM**	**16-QAM**	**256-QAM**
BCH	9.53	9.53	7.45	7.45
LDPC	18.00	18.00	10.13	10.13
Bit Interleaver	36.00	36.00	36.00	34.11
Mapper	95.29	108.00	72.00	81.00
Rotation	1800.00	3240.00	1296.00	3240.00
Cell Interleaver	540.00	1620.00	324.00	648.00
Time Interleaver	53.05	96.61	51.36	95.98
Frame Builder	98.34	190.28	96.08	196.67
Frequency Interleaver	295.01	590.87	293.31	590.02
Pilot Insertion	41.83	82.55	39.81	80.07
FFT	56.05	112.27	55.73	112.10

It might perhaps be possible to significantly reduce the computational complexity of the block by using lookup tables instead of calling mathematical functions. However, this has not been attempted by the authors. A proposed methodology for exploiting the large amounts of memory typically present on general purpose computing systems in order to accelerate SDR systems, can be found in [6]. This method, or similar methods, could perhaps be used in the demapper, as well as in other parts of the system. An FPGA implementation of a DVB-T2 demapper, together with some simplifications of the algorithm, is discussed in [7].

The LDPC decoder performance in the benchmark results is the performance assuming 10 iterations of the iterative Belief Propagation (BP) decoder. The amount of iterations necessary might however vary with the quality of the received signal, making the LDPC decoder complexity highly variable.

The currently used LDPC decoder algorithm is that of the sum-product decoder [8,9]. This decoder gives good error correction performance, but is quite computationally intensive, involving for example hyperbolic tangent evaluations. The min-sum approximation [8,9] simplifies this algorithm, trading precision for speed, and involves mostly additions and comparisons. The long code lengths involved in DVB-T2 might still make realtime decoding (with acceptable error correction performance) quite hard to perform on general purpose processors.

Table 2: Demodulator throughput (Mbps)

	Short code		Long code	
	16-QAM	**256-QAM**	**16-QAM**	**256-QAM**
LDPC	0.12	0.12	0.10	0.10
Bit Interleaver	33.06	33.06	32.40	32.40
Mapper	0.54	0.10	0.59	0.09
Rotation	704.35	952.94	648.00	1620.00
Cell Interleaver	101.25	180.00	108.00	216.00
Time Interleaver	24.54	49.11	18.27	53.10
Frame Builder	140.13	280.67	139.32	280.26
Frequency Interleaver	31.14	59.09	29.33	59.00
Pilot Insertion	62.28	112.27	37.15	112.10
FFT	70.07	140.33	61.92	124.56

A Xilinx Virtex-5 LX110 FPGA was used for handling the realtime LDPC decoding in the DVB-T2 SDR system described in [5]. Also, in [10] a parallel architecture for LDPC decoding in a DVB-S2 system (only some minor differences exist between LDPC codes used for DVB-T2 and DVB-S2) on an FPGA is described.

LDPC decoding on Graphics Processing Units (GPUs), and other multicore architectures, is discussed in [11]. While the benchmark results in [11] do show that GPUs can achieve rather high throughputs for certain codes, the results

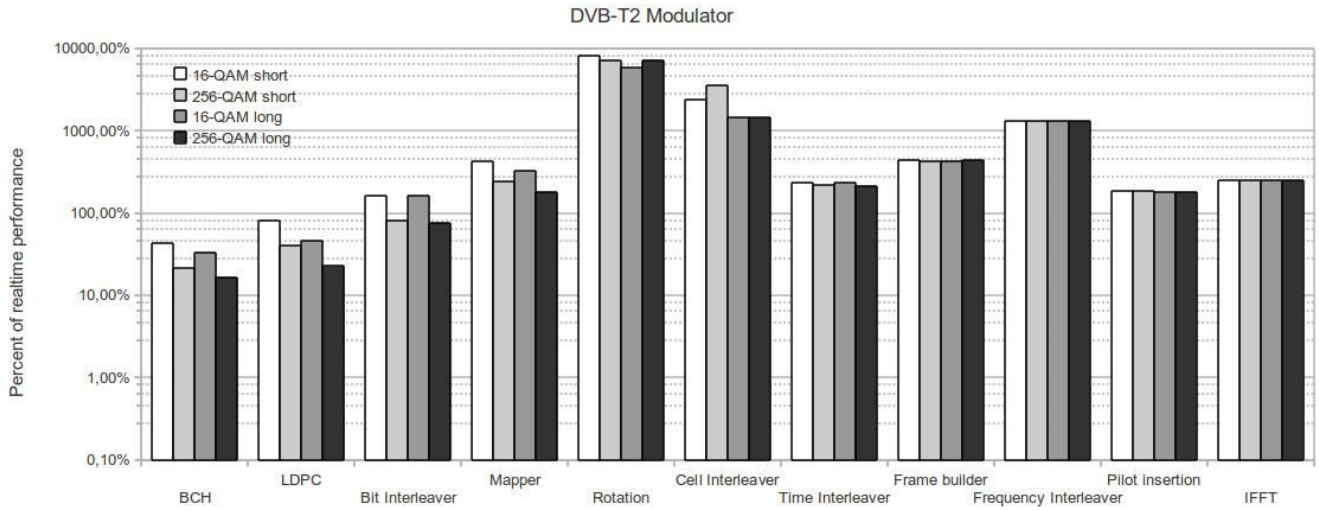

Figure 1: Modulator block throughput relative to required throughput for realtime performance (100% is realtime)

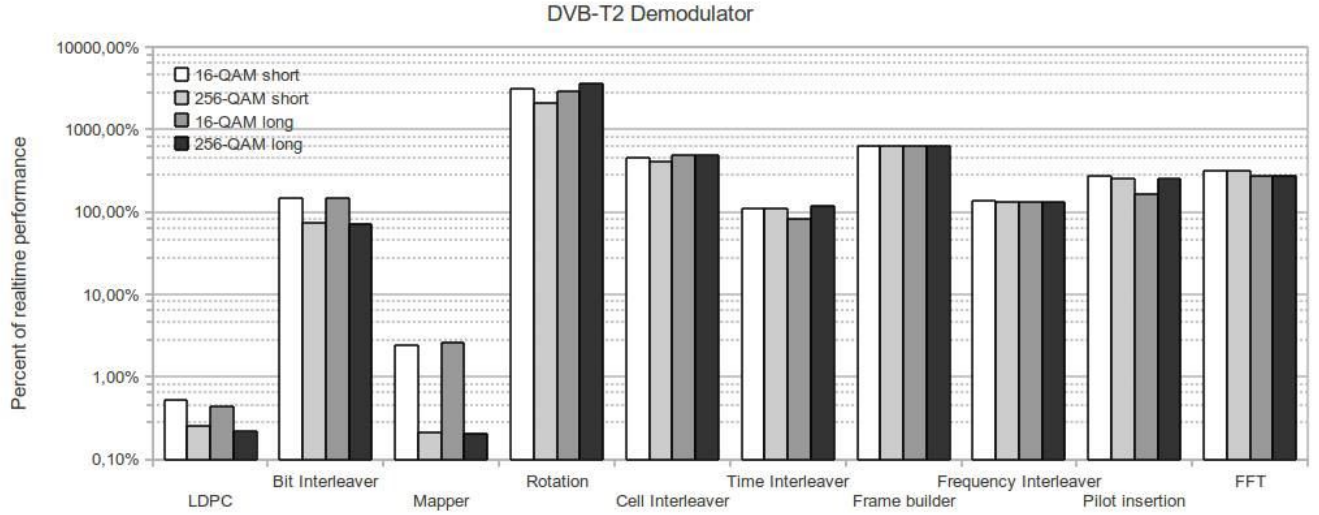

Figure 2: Demodulator block throughput relative to required throughput for realtime performance

suggest that performance may not be very good for the long code lengths used in DVB-T2. The authors of [11] point out that for long codes, the memory bandwidth between host memory and GPU RAM might become a bottleneck, as well as the fact that data structures for large codes do not fit very well into the fast constant memory on a typical GPU.

We benchmarked our system only using the 8K FFT size (because the other sizes were largely untested), while DVB-T2 supports up to 32K FFTs. Use of the larger FFT sizes will likely be quite detrimental to the throughputs of the FFT blocks in the modulator and demodulator. The FFTW [12] library is used for FFT calculations. FFTW should however compute discrete Fourier transforms in O(n log n) time for FFT length n [12], which would seem to not make the use of larger FFT sizes on general purpose processors infeasible.

As mentioned earlier, some parts of a complete DVB-T2 system – such as synchronization functionality in the receiver – have not been benchmarked here, as they have not been implemented yet. Thus it is unclear to the authors, at this point, how much extra complexity these parts would bring to the system.

6. SUMMARY

In this paper, we have measured the performance of our software implementations of most of the various signal processing blocks of a DVB-T2 modulator and

demodulator. The results were presented as throughputs based on timing the core functionality of each block.

The results indicate that the modulator should be possible to realize entirely in software on general purpose computing systems, given some further optimization of the algorithms involved. The demodulator, however, might not be suitable for running exclusively on general purpose processors. Results indicate that the most computationally heavy parts of the demodulator are the FEC decoding and demapper functional blocks. We have discussed alternative architectures, specifically GPUs and FPGAs, and their suitability for executing these tasks. These architectures would retain the reconfigurability that is a major benefit of SDR systems.

As the GNU Radio implementation is not finished, the next step would be to implement all of the remaining blocks in the GNU Radio environment, as well as to apply some of the techniques discussed here to improve the performance of blocks where necessary.

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)," 2009.

[2] L. Vangelista, et al., "Key technologies for next-generation terrestrial digital television standard DVB-T2," *Communications Magazine, IEEE*, vol.47, no.10, pp.146-153, October 2009.

[3] E. Blossom, "Exploring GNU Radio," Revision v1.1, Nov. 2004, Available online at: http://www.gnu.org/software /gnuradio/doc/exploring-gnuradio.html

[4] V. Pellegrini, G. Bacci, and M. Luise, "Soft-DVB, a Fully Software, GNURadio Based ETSI DVB-T Modulator," *5th Karlsruhe Workshop on Software Radios*, 2008.

[5] A. Viessmann, et al., "A DVB-T2 receiver realization based on a software-defined radio concept," *Communications, Control and Signal Processing (ISCCSP), 2010 4th International Symposium on*, pp.1-4, 3-5 March 2010.

[6] V. Pellegrini, L. Rose, and M. Di Dio, "On Memory Accelerated Signal Processing within Software Defined Radios", *Technical Report arXiv:1004.0263*, April 2010.

[7] Li Meng, C.A. Nour, C. Jego, and C. Douillard, "Design of rotated QAM mapper/demapper for the DVB-T2 standard," *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pp.018-023, 7-9 Oct. 2009.

[8] N. Wiberg. "Codes and Decoding on General Graphs," PhD Thesis, Linköping University, 1996.

[9] Chen, et al., "Reduced-Complexity Decoding of LDPC Codes," *Communications, IEEE Transactions on*, vol.53, no.7, pp. 1232- 1232, July 2005

[10] M. Gomes et al., "Flexible Parallel Architecture for DVB-S2 LDPC Decoders," *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, pp.3265-3269, 26-30 Nov. 2007.

[11] G. Falcao, L. Sousa, and V. Silva, "Massively LDPC Decoding on Multicore Architectures," *Parallel and Distributed Systems, IEEE Transactions on*, no.99, pp.1-1, 2010.

[12] M. Frigo, and S.G. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE* , vol.93, no.2, pp.216-231, Feb. 2005.