UNIVERSITÄT PADERBORN

Die Universität der Informationsgesellschaft

Faculty for Electrical Engineering, Computer Science and Mathematics
Department of Computer Science
PG Custom Computing for Video Codecs

PG Custom Computing for Video Codecs

Project Plan

May 12, 2013

Members
Farhan Ahmed
Andreas Kohlos
Vignesh Makeswaran
Robert Mittendorf
Manuel Peuster
Henning Schmitz
and
SeydEsmaeil SeydAshrafi

Contents

1 Introduction

1.1 Introduction

This is the project plan of a project group in the master's program of computer science at the University of Paderborn. The name of the project group is "Custom Computing for Video Codecs" or in short "PG CC Codecs". Main goal of this project group is to evaluate the DataFlowEngine (DFE) technology developed by Maxeler Technologies in the context of video encoding. A DFE is basically an FPGA, but the architecture of the implemented hardware is described on a higher level than VHDL. More details about the DFE will be presented in Section 1.3.

The project group started in October 2012 and will end in September 2013. Before the main project of this group, which is about to be described in this document, there was a tutorial session where the participants got introduced to the DFE technology. Furthermore a seminar took place, where every student investigated a paper of a promising approach in porting parts of a video encoding process to FPGAs. In parallel a JPEG Encoder based on the libjpeg from the Independent JPEG Group (IJG) was partly ported to a Maxeler DFE.

About video codecs

Given outline of the project group was the acceleration of a state-of-the-art video codec using Maxeler's DFE technology. A video codec stores visual information, captured by a camera or created by a computer, in a bitstream. A codec consists of an encoder, which produces a compress video stream, and decoder used to retrieve the pictures out of the video stream, e.g. for playback. Since modern cameras and renderings are done in high resolutions, hundreds of megabytes are produced for every single second of video. To overcome bottlenecks in transmission and storage of data, efficient compression techniques are needed. Since lossless compression is highly limited, the limited capabilities of the human eye are exploited to reduce the actual amount of data which needs to be stored. Furthermore local and especially temporal redundancy of most video streams are exploited to reduce the amount of data even more.

Full HD is the common resolution in livingrooms nowadays, but higher resolutions like Ultra HD (also called 4K HD) are upcoming and already used by professionals. Furthermore, there are several techniques which increase the amount of data even more. 3D and 48 fps recordings boost the size of the raw data even more, as well as extended color spaces do. Yet those encodings are supposed to be as fast as possible. For scenarios like live television broadcasts realtime encoding is needed, for scenarios like postproduction or mastering even faster encoding is desirable. On the other hand, a good quality of the encoded material is mandatory and low power-consumption is another worthwhile goal. High performance and low power consumption can be achieved by hardware builds,

which are optimized for their task. Compared to general purpose CPUs, hardware builds can improve the system on these metrics by orders of magnitude.

The decision for H.264

There are many video codecs available, but only a few of them are widely used and utilize advanced means of compression. We are aware of the fact that H.265/HEVC, the successor of H.264, was standardized on January 25th 2013. There are several reasons why we decided to work on H.264 instead. First of all there is a well documented open source reference implementation of H.264, the JM Software [1]. Using this software, we can focus on the actual acceleration and do not need to care about details like the order of bits in a video file. Furthermore all sub-standards will be supported by our implementation, even though some sub-standards may be not or not fully accelerated. In addition there have been a lot of approaches of accelerating the encoding according to the H.264 standard using FPGA architectures. Those approaches may be adoptable to our DFE design.

Further sections and chapters

In the remainder of this introduction, H.264 and the DataFlowEngine technology will be introduced to the reader. These information are important to understand further chapters of this project plan, but are not part of actual project plan in a narrow sense. In Chapter 2 the overall goals will be determined. After describing related work about H.264 on FPGAs in Chapter 3 we will describe our organizational structure in Chapter 4 and finally the heart of this document, our actual workplan, in Chapter 5.

1.2 Introduction to H.264

This section is an introduction to basic strategies used in the H.264/AVC codec, which will be referred to as H.264. Many ideas in H.264 are inherited concepts from earlier standards like MPEG, MPEG 2 and MPEG 4, but since the history of video codecs is beyond the scope of our project group, we will not go to much in detail about this. The introduction starts with the general partition used for video sequences and will afterwards present the means of motion estimation, rate-distortion optimization and encoding used in H.264. A rough overview of a typical encoder is shown in Figure 1.1.

1.2.1 Partitioning of a video sequence

A video sequence consists of Groups of Pictures (GoPs) containing single pictures, which are called frame. Such a frame consists of two fields. The top field contains the even numbered rows (starting with the first line, because of the zero-based index), the bottom field contains the odd-numbered rows. If these fields of a single frame contain the information captured by the camera at one point in time, this is called progressive video format. If

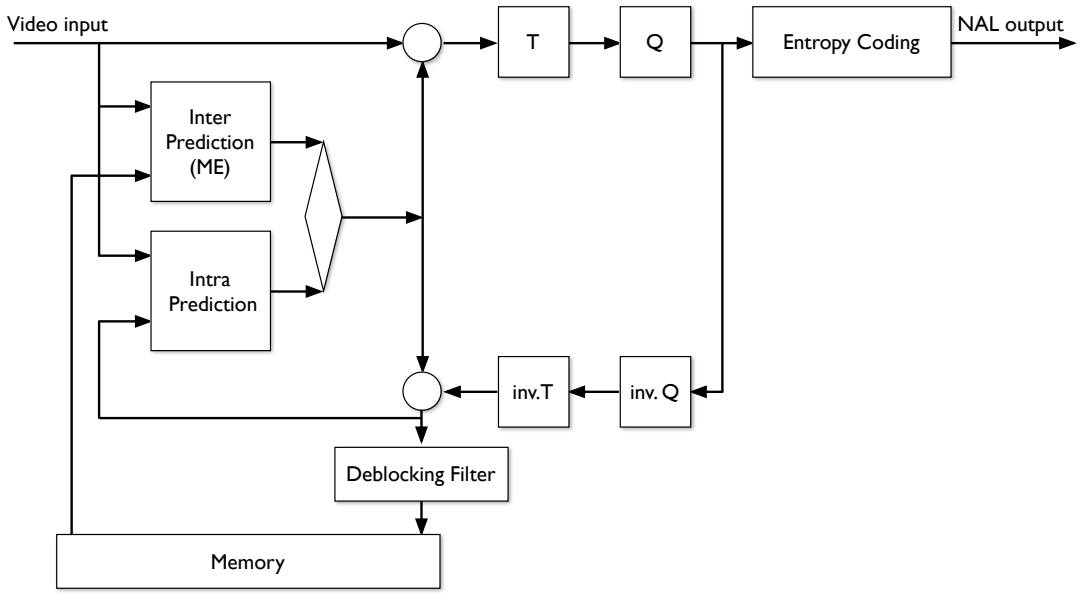

Figure 1.1: Overview about a typical H.264 encoder structure

there is a small shift in time between the recording of those fields, this is called interlaced video format.

The frames (and fields alike) can be partitioned into so called slices, which are again partitioned into macroblocks (blocks of 16 x 16 pixels). These slices correspond to frames in older video codecs, which did not allow a level of partition between the picture level and the macroblock level. Consequently, there are I- Slices, P-Slices and B-Slices. New types have been introduced with H.264, but they are used for fast skipping or error recovery only and therefore not covered here. Every encoding is either done in inter mode(using information from another element - like an other slice) or in intra mode (using information from the same element only). The word "mode" is usually omitted, so "the macroblock is inter" means that it is encoded in the inter encoding mode. An I-Slice can be decoded independently, because it uses intra coding only, which is similar to the techniques used in the JPEG-Standard. P- and B-Slices cannot be decoded independently, because they use an inter coding. P-Slices may refer to information from a previous picture. Different blocks can reference to different pictures, but always to exactly one picture. B-Slices may refer to information from two pictures. For the ease of read, we will talk about pictures from now on, which in fact means a slice, since different slices of one frame/field are independent from each other. As a remark we want to mention, that the order of pictures can be changed. In fact, the references need to point to a previously *decoded* picture, which is not necessarily a prior picture in the original video stream. This is another abstraction from details made for the ease of read.

1.2.2 Motion vectors and motion compensation

Motion vectors are stored with every (macro)block of a picture. The amount of vectors per macroblock differs, depending on the number of blocks it consist of - more about this later on. These vectors hold the information, where the content represented by this block was located on a previous slice. So this is a reference to previous slices, not a prediction for upcoming ones. Using the blocks of a slice and their vector(s), a motion compensated picture (MCP) can be calculated. This MCP is usually very close to the current picture, since the motion vectors have been chosen by the encoder using the minimal difference as criterion. The difference between the MCP and the current picture is usually much smaller than between the pictures themselves, so these differences can be encoded with much less bits than the current picture itself. The information about the differences are called residuals or residual picture.

The reason why there can be more than one motion vector per macroblock is, that this macroblock can be sub-partitioned into blocks. If a block is smaller than 16x16, there are more motion vectors, because every block has its own motion vector. Obviously it is a waste of memory/space to store the motion vectors of neighboring blocks, if they are the same. If these neighboring blocks are from a different macroblock, this encoding cannot be optimized, because the macroblock structure is needed - otherwise the Encoder (and the Decoder) would have to constantly change the partitioning of the picture, and errors in the bitstream might effect much larger parts of the video sequence when decoding. On the other hand this type of partitioning would also be much harder to handle for encoders. If these neighboring blocks are from the same macroblock, the subpartition of the macroblock can be optimized by storing those blocks together. Not every subpartition of a macroblock is possible, but quite a lot of them.

Advanced means for motion compensation

The idea of motion compensation can be further optimized by using so called fractional-pel motion vectors. Using them, a motion vector can not only point to a specific pixel (of a previous picture), but also to a position defined by fractions of pixels. A pixel usually represents a sample recorded by the video sensor, thats where "- pel" stems from. In fact, only the brightness (luma) of a pixel is always the representation of one single sample, the information of the color (chroma) maybe not. The discussion about color spaces like RGB, YC_bC_r, CMYK, HSV and so on is beyond the scope of this paper, as well as down-sampling. Even though it is rather counter-intuitive, using these fractional-pel motion vectors which are based on a (standardized) interpolated version of the pictures themselves really improves the quality of the prediction by the motion compensated pictures - measured by the number of differences between the MCP and current picture itself.

In some cases, for example alternating sequences of pictures, the picture directly before the current one may not be suited as a good reference for the prediction. A picture prior to this one maybe more useful. Therefore it is possible to use the information from pictures prior to the previous one. Since these prior pictures are needed for decoding the video sequence, they need to be stored in a buffer in RAM for fast access - also for decoding. Because of this, the number of prior pictures under consideration has to be limited. This

limit is defined by the coding standard. In H.264, this limit is 16 or less - depending on different parameters like the video resolution.

How motion vectors are found

The interesting question is, how motion vectors are found, which result in a MCP which is as close as possible to the current frame. Furthermore, how a partition of the macroblock can be determined, such that the number of partitions is as small as possible. This is the task of the encoder, and it is the most computation intense part of the encoding process. It is also the reason why encoding lasts much longer than decoding.

The beauty of H.264 is, that the encoder can freely decide how to obtain these motion vectors, as only decoding is standardized. The drawback is, that with a given bitrate a video stream encoded in H.264 can look very good or rather bad. Without knowing the encoder (and the specific encoding mode) the format H.264 does not tell anything about the quality of the encoded video stream.

If the Encoder wants to guarantee the best possible representation (an thereby the best possible compression), it has to check the difference between every possible block - resulting from the partition of the macroblock - and every single position in every single reference picture. This brute-force or full search-approach costs a lot of encoding time. Even though this maybe an option in some scenarios like encoding a Hollywood movie for cinema, DVD or Blu-ray Disc, in the common case this amount of time is usually either not possible (especially in realtime-situations like live-broadcasting shows this would take to long) or not available (e.g. thinking about the resulting costs).

1.2.3 Rate-distortion optimization and encoding

In principle, a macroblock can be stored lossless using any motion vector by storing all information of the residual picture. The encoder has to trade off between the quality of the decoded picture and the size of the encoded information. The loss of quality or distortion is usually measured as Peak Signal to Noise Ratio (PSNR). The size of the encoded information is measured in bits per second (bps). The residuals are transformed to a spectral representation and later on quantized using quantization coefficients. The larger those coefficients become, the higher the quality loss and by trend the lower the size of the encoded information will be. An example obtained by JPEG compressions of a picture with different quantization coefficients can be found in Figure 1.2.

The rate-distortion optimization (RDO) tweaks the quantization coefficients, usually given a quality parameter or a desired bitrate. Depending on the specific encoder, the RDO may also select between different motion vector candidates determined by the motion estimation beforehand. To measure the PSNR and the (bit)rate, the residuals are usually transformed, quantized and encoded during the RDO, before the picture is reconstructed like a decoder would do. This means the encoded information are backward quantized, backward transformed and combined with the MCP to obtain the picture the decoder would output. Now the PSNR can be determined, while the bitrate can already be determined directly after encoding.

Figure 1.2: Results of a JPEG compression with different quality settings [2]

The encoding process is rather advanced. For different information, different optimized encoding techniques are used. On the other hand the encoding is not as efficient as it could be, because for the playback it is desirable to have a robust stream. This means, that an error in the bitstream due to transmission errors or storage failures should not destroy all pictures beyond the point in the stream the damage took place on. Another eligible feature is fast forward. If a playback is supposed to start at a position other than the beginning, it should not be necessary to process all the data of the previous frames, as this might be gigabytes of information. Random access to each I-Frame in the encoded bitstream is the optimal solution for skipping in a video stream.

1.3 Maxeler DataFlowEngine

As the name suggests, the DataFlowEngine (DFE) is designed to operate on data flows. These data flows or data streams are supposed to be as large as possible, since the data is transfered back and forth via PCI-Express (PCIe) to a DFE card. The computation starts, as soon as enough data is available to start with. By this strategy, a share of the transfer time can be hidden behind the computation time, if the stream is large enough. CPUs need to fetch instructions and data. Even though several clever strategies like caching are used in modern CPUs, there is still a lot of overhead caused by these steps. On DFEs the data access can be optimized by using the data flows and instructions are completely obsolete.
For many applications, there is a large locality in operations on the data streams. These applications are suited perfectly for acceleration by Maxeler DFE technology. Moreover, the operations on DFE cards are realized on FPGA-chips (field programmable gate array) from Xilinx. Therefore the compilation includes a hardware build and thereby takes quite some time, but afterwards this pays off with operations being orders of magnitude faster than computations on a common CPU. Several instances of a FPGA design, a so called kernel, can be used to accelerate the computation even more by parallelizing the computations.

What distinguishes the DFE and a common FPGA design

The interesting idea in Maxeler's approach is to allow the design of hardware in high-level programming languages. The programmers do not need to learn a hardware description language like VHDL. The code for the host site is written in C or Fortran, the kernel itself is written in Java. Since the kernel is preprocessed by the default Java bytecode compiler, The Java-API can be used to calculate values, that can be determined at compile time.
Nevertheless, programming is very different from typical Java programming. Loops create a lot of hardware which processes all instances of the loop body in parallel, for example. This way one can very quickly construct kernels, which exceed the resources available on the FPGA. The same holds, if the access pattern to the data flow includes large offsets, as they are realized with buffers and the architecture of the DFE is simply not designed for those computations.
Since the basis of the DFEs is a FPGA, a real hardware build, a so called synthesis, costs a lot of time, easily reaching several hours or even days. Therefore Maxeler provides a simulator, which can be used to verify correctness of the kernels without the synthesis of hardware. The simulation does not only decrease the performance compared to a hardware build, also the resource limit of the hardware is not taken into account by the simulation. It is hard to predict the exact amount of hardware components needed, as the synthesis optimizes the hardware usage and is very computationally intense, as mentioned earlier.

Maxeler DFE for video encoding

Video encoding seems like a task well suited for being accelerated by the Maxeler DFEs. Video encoding is highly parallelizable by assigning different GoPs to different workers, in this case different kernels. Video encoding nowadays enforces a lot of computations on a huge amount of data. For Full HD content, about 180 MBytes are produced for every single second of the video, if no compression is used. So we have a high throughput and computationally intense task, which is rather easy to parallelize - exactly that kind of task the DFEs are designed for.

2 Goals

Basically we want to accelerate the H.264/AVC video coding standard by using Maxeler's DFE technology. For this purpose we will use JM Software, which is an open source reference implementation of H.264/AVC. JM Software [1] has very well organized Doxygen-based software manual and Release Notes, which seems to be very helping for us in order to get into the JM Software to analyze the features which JM Software supports and we can easily decide which features we will be working on. We have planned to work on the acceleration of the ME (motion estimation), SAD (some of absolute difference), RDO (rate distortion optimization), as well as transformation and quantization.

H.264/AVC supports three different profiles, which are sets of capabilities of H.264 that are used in order to handle high quality coding of video content at very low bit rate. We have planned to support the two most used profiles: Baseline Profile (BP) and Main Profile (MP), which incorporate the main features which H.264 provides.

The Baseline Profile supports all the features of H.264 except of B slices, weighted prediction, CABAC, field coding, and picture or macroblock adaptive switching between frame and field coding, but these features are supported in the Main Profile. The Baseline Profile does also not support SP (*Switching P*)/SI (*Switching I*) slices and slice data portioning. The Main Profile does not support FMO (*Flexible Macroblock Ordering*), ASO (*Arbitrary Slice Ordering*), and redundant picture features which are supported by Baseline Profile.

H.264 uses the block-based motion compensation and transform coding model. We will support the default motion estimation techniques which H.264 offers: Multi-picture inter-picture prediction. In which previously encoded pictures as a reference are used, the standard allows up to 16 reference frames and in case of interlaced encoding limit is extended up to 32 reference fields.

In order to fine tune the displacement of moving areas we will support feature provided by H.264 that is called quarter-sample-accurate motion compensation that is obtained through linear interpolation of the half-pel values. We will also incorporate another efficient technique regarding motion vectors, which is to point the motion vectors over the picture boundaries. Whereas in earlier standards motion vectors pointed only to the areas within previously decoded reference pictures. Enhanced entropy coding CABAC and CAVALC will be brought under consideration, but we are still not sure how encoding can be ported to DFE in a proper way.

2.1 Coding performance

In order to obtain an overview of how much time is spent during the coding process on motion estimation, transformation and quantization, as well as the actual encoding of the residual data, the JM reference software was profiled using the gprof tool [3]. For the

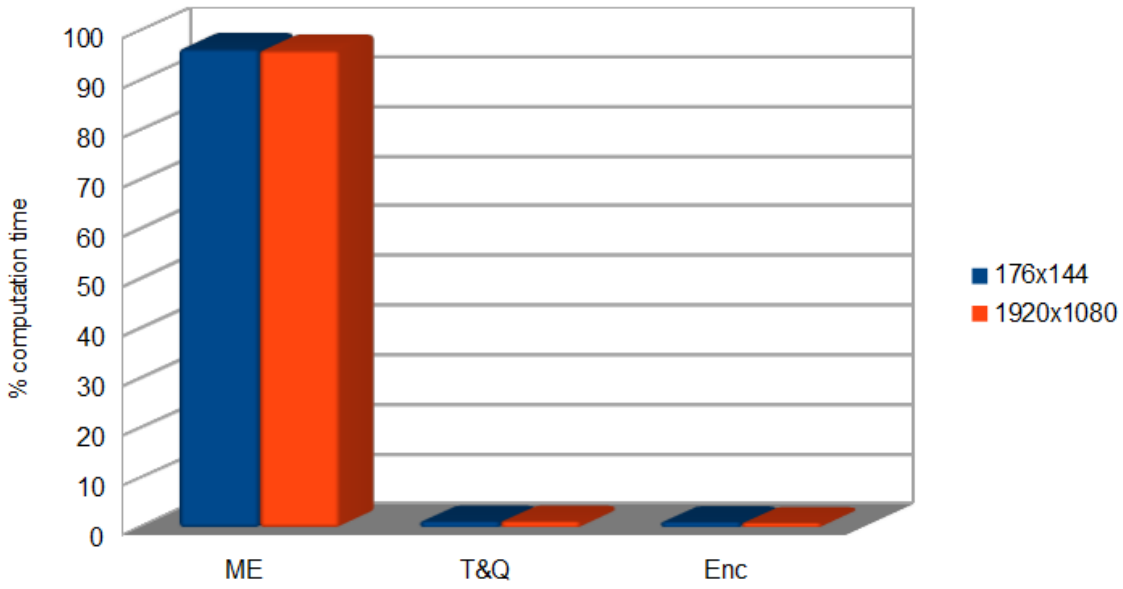

Figure 2.1: Portion of motion estimation (ME), transformation and quantization (T&Q), as well as encoding (Enc) of total computation time for JM reference software at different video resolutions.

remainder of this chapter the complete process will be called "coding" of a video and the last operation, where the actual bits of the output file are determined, will be referred to as *en*coding. The settings for the coder were left at default with Fast Full Search as the algorithm used for motion estimation. This rather simple algorithm was chosen, because it would most likely be easier to implement on the Maxeler hardware, though other more complex motion estimation algorithms will be analyzed as well. The profiling results, as represented in Figure 2.1 showed that more than 96% of the computation time was used on motion estimation, around 1.2% on transformation and quantization and about 1% on encoding. In related work, like [4] and [5], the motion estimation was measured to just take up about 80% of the overall computation, which is due to faster and more efficient motion estimation algorithms than the Fast Full Search being used. The first video sequence used had a resolution of just 176x144, but even a second test sequence in Full HD resolution yielded the same results. So the resolution, at least for the tested settings, has no discernible impact on the distribution of the overall computational effort during the coding process. This means, that any speedup achieved for the motion estimation will significantly improve the overall performance of the coder as well.

As a first step to enhance performance, a kernel for acceleration of the Fast Full Search algorithm is being implemented, though more complex algorithms will be reviewed and implemented in later stages of development. The kernel will compute the numerous sums of absolute differences (SADs) that are part of the Fast Full Search algorithm. The computation of these values alone takes up to 63% of the entire coding process.

To illustrate the potential for possible speedups we compare the computation time of the current JM reference software to the estimated computation time of the SAD kernel. For that, we let the JM software encode a Full HD video stream, using the Fast Full

Search. The algorithm takes a given search range to determine a fixed search window for each reference block. For each position in the search window 16 SAD values are computed, corresponding to the 16 different 4x4 blocks in a macroblock. The software itself measures the coding time for each frame individually. According to those measurements, the JM reference software requires about 130s to completely encode a frame. Considering that about 63% are used for computation of the SAD values, that would be about 82s for computations, that would be done by the kernel on the FPGA.

The kernel being implemented uses two inputs to stream in a reference block and its search window respectively. For the initial implementation, as well as the following performance model, the data is being streamed in via the PCIe directly from the host. Another option would be to send the pixel data for one frame to the DRAM on board of the FPGA and stream in the search windows from there. The maximum bandwidth of the DRAM exceeds the transfer speed of the PCIe significantly, but is greatly dependent on the access pattern of the necessary data. Nevertheless, this approach will be explored for future milestones.

In general, since the time for initializing the FPGA, setting up streams etc. is assumed to be dominated by the computation time, the time for execution can be modeled as

$$T_{exec} = max(T_{compute}, T_{DRAM}, T_{PCIE}, T_{MaxRing}). \tag{2.1}$$

Where $T_{compute}$ is the time the kernel is running, and $T_{DRAM}, T_{PCIE}, T_{MaxRing}$ are the times that are necessary to send the required data over DRAM, the PCIe and the MaxRing respectively. Since, for this model, we only use the PCIe to send and receive data from the FPGA, the DRAM and MaxRing can be discarded.

That leaves the computation time of the kernel and the transfer time of the input and output data over the PCIe, last of which can be calculated by

$$\begin{aligned} T_{PCIE} &= \frac{Bytes\_In_{PCIE}}{BW\_In_{PCIE}} + \frac{Bytes\_Out_{PCIE}}{BW\_Out_{PCIE}} \\ &= \frac{(80^2 * 2B * 8100 + 1920 * 1080 * 2B) + (65^2 * 16 * 4B * 8100)}{2GB/s} \\ &= 1.06819s. \end{aligned} \tag{2.2}$$

Since the PCIe is bi-directional, it does not share bandwidth between send and received data, so that $BW\_In_{PCIE} = BW\_Out_{PCIE}$. The Max3 card currently in use by our Maxeler computer is capable of 2GB/s of bandwidth. As for the necessary amount of input and output data, the Fast Full Search algorithm uses a fixed search window to find the most likely position of a given block. That search window is given by a search range, which is usually set to 32, so that a block is searched ± 32 pixels in horizontal, as well as vertical direction. That results to $32 * 2 + 1 = 65$ search positions in as many lines of a frame. Since we also need 15 additional values in each line, as well as 15 additional lines, to be able to calculate all 16 SAD values at a location at the right border of the search window, the number of values for a search window to be streamed to the kernel is 80^2. Since the JM reference software uses 16 Byte values for each pixel that number has to multiplied by 2 and again by 8100 which is the number of blocks in a Full HD frame, because the search window for each block to be searched is streamed in separately. In

addition the reference blocks which are searched have to be streamed in as well, which accounts as an additional $1920 * 1080 * 2$ Byte. For the output, for each of the 65^2 search positions the kernel produces 16 SAD values which are stored as 4 Byte integer. Again this amount of data is calculated for each of the 8100 macroblocks in the reference frame.

The runtime of the kernel itself can simply be computed by taking into account the number of cycles that the kernel runs over the clock frequency, which is set at compile time.

$$\begin{aligned} T_{compute} &= \frac{Cycles}{Frequency} \\ &= \frac{80^2 * 8100}{100MHz} \\ &= 0.5184s \end{aligned} \tag{2.3}$$

It takes the kernel 80^2 cycles to iterate over all the values in the input stream with the data of a search window, to complete the search for one of the 8100 blocks. The data for the reference blocks is read in parallel to the other input and does not impact the computation time. At 100MHz that leaves a runtime of almost half of the time necessary to transfer the data to and from the FPGA. Therefore the overall execution time is governed by the transfer time of the PCIe.

As mentioned earlier, the computation of the SAD values takes about 82s for the reference software, whereas the kernel would theoretically need just over 1s. Maybe even less, if the DRAM could be integrated as well. Of course this speedup can only be achieved if the data can be streamed in efficiently, without any redundant data transfer, which will be one of our immediate goals. However the potential for significant speedups is clearly indicated.

Unfortunately even if we could achieve computation times comparable to this estimate, it would not be fast enough to encode an HD video in real time, where 25 to 30 frames have to be encoded each second. To achieve this, it will be necessary to analyze different motion estimation algorithms, that may be easily adoptable to the Maxeler hardware, to gain further speedup of the encoder.

2.2 Measuring our results

In order to measure speedups that we will achieve during our project we will use a continuous integration environment to periodically track the speed of our implementation. This is done with our automated test suite, which we have developed during our first project on the JPEG encoder. This test suite is basically a Python wrapper that pulls the latest software version from git, compiles it and executes it with different runtime options and input files.

This in combination with a cron-job that executes the test suite in different time intervals, builds a continuous integration (CI) environment. On the one hand it guarantees that the code base is checked periodically to find bugs or errors, and on the other hand we can track the speed of execution for different test sequences. It also allows us to keep track of the current hardware utilization of our kernels. For this it is planned to execute normal

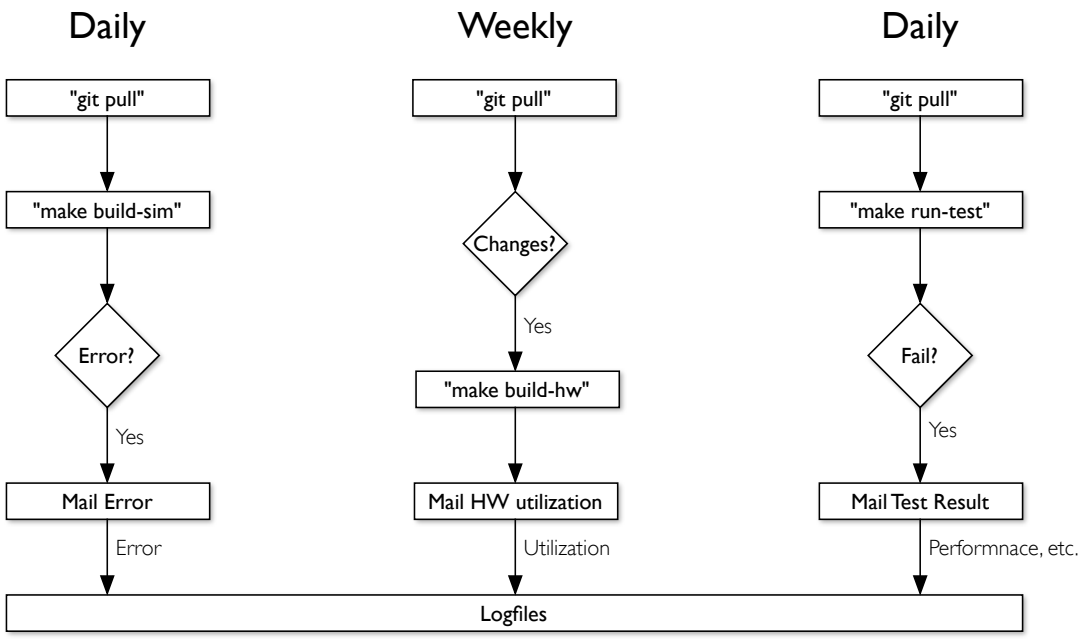

Figure 2.2: Actions performed by the CI environment

simulation builds each night and a complete hardware build once a week, which is only executed if the kernel files have changed, to avoid unnecessary load on the Maxeler machine. The action flow of the CI can be seen in Figure 2.2. It shows that on the one hand an email is send to our mailing list if something goes wrong and it shows that all results and outputs are stored in different log files for reuse. The daily test run with real input sequences is always done with the previously created hardware build, since performance measurements with the simulation environment make no sense.

Beside the execution times, also other values are tracked. For example the peek-signal-to-noise-ration (PSNR) as an indicator for quality losses of different implementation approaches. For this the very detailed output of the JM encoder can be used which is shown in Listing 2.1. It can be seen that there are also different timing outputs and especially the time for motion estimation is measured separately.

All these outputs are stored and can later be reused to evaluate our results, especially to create graphs which show the different speedups achieved by our implementation at different points in time during the runtime of this project group.

```
-------------------------------------------------------------------------
Frame   Bit/pic  QP SnrY   SnrU   SnrV Time(ms) MET(ms) Frm/Fld Ref
-------------------------------------------------------------------------
00000(NVB)   320
00000(IDR) 22288 28 37.648 41.349 42.955     86       0    FRM    3
00002( P )  9944 28 37.280 39.667 40.658    800     634    FRM    2
00001( B )  2184 30 36.770 37.856 39.624   2363    2161    FRM    0
-------------------------------------------------------------------------
 Total Frames:  3
 Leaky BucketRateFile does not have valid entries.
 Using rate calculated from avg. rate
 Number Leaky Buckets: 8
     Rmin     Bmin     Fmin
   344160    22288    22288
   430200    22288    22288
   516240    22288    22288
   602280    22288    22288
   688320    22288    22288
   774360    22288    22288
   860400    22288    22288
   946440    22288    22288
------------------ Average data all frames  -----------------------

 Total encoding time for the seq.  :   3.249 sec (0.92 fps)
 Total ME time for sequence        :   2.795 sec

 Y { PSNR (dB), cSNR (dB), MSE }   : {  37.233,  37.218,  12.33970 }
 U { PSNR (dB), cSNR (dB), MSE }   : {  39.624,  39.392,   7.48038 }
 V { PSNR (dB), cSNR (dB), MSE }   : {  41.079,  40.869,   5.32365 }

 Total bits                        : 34736 (I 22288, P 9944, B 2184
    NVB 320)
 Bit rate (kbit/s)  @ 30.00 Hz     : 347.36
 Bits to avoid Startcode Emulation : 20
 Bits for parameter sets           : 320
 Bits for filler data              : 0

-------------------------------------------------------------------------
Exit JM 18 (FRExt) encoder ver 18.4
```

Listing 2.1: Output of the JM encoder running on 3 frames

3 Related Work

H.264/AVC is one of the most used video coding standards today. Because of this many researchers and companies are working on performance improvements of the video encoding process. Therefore they do not only invent faster and more efficient algorithms for e.g. motion estimation, but also implement video encoders on different hardware platforms with different optimization goals e.g. throughput or energy efficiency. In this section we will show and discuss different solutions together with some problems which can appear while implementing H.264 completely (or partly) on hardware like FPGAs or ASICs.

Section 3.1 shows related work focusing on general problems occurring when H.264 should be implemented in hardware and some available products are presented. Section 3.2 discusses techniques and architectures which are used to accelerate sub-parts of H.264 and Section 3.3 marks out how a implementation on Maxeler DFEs differs from other FPGA prototypes.

3.1 Accelerate a H.264 encoder with hardware

Video encoding uses a big amount of computational power when done in software on a general purpose CPU. But, on the one hand there is a growing need of encoding videos on small mobile devices with limited processing power and small energy budgets. To meet this challenge, H.264 encoders are needed that are implemented as dedicated and energy efficient hardware. On the other hand, companies like television networks need high performance encoders which are able to process high resolution video data as fast as possible.

Therefore different possibilities exist to build encoders which use specialized hardware for better performance or energy efficiency. There are basically three possibilities used for this. The first one is using co-processors on e.g. graphics adapter (GPU) which can accelerate software implementations optimized for parallelism. The second and the third options are to design specialized hardware which is then implemented as ASIC or on a FPGA chip. These two options can be used in normal computers as encoder accelerators e.g. PCIe video encoder cards or as dedicated chips in energy efficient embedded devices.

However, implementing a complex video codec like H.264 in hardware is still a challenging task. On the one hand, not all parts of the encoder are well suited for hardware acceleration and one should try to find and focus on computational hot-spots which can easily be accelerated with hardware, on the other hand do software implementations, like the H.264 reference software (JM [1]) do not provide a hardware optimized functional structure or data flow.

These problems are discussed by Shafique *et al.* in [6]. They started their work with profiling the software implementation of H.264 to find computational hot-spots to be ac-

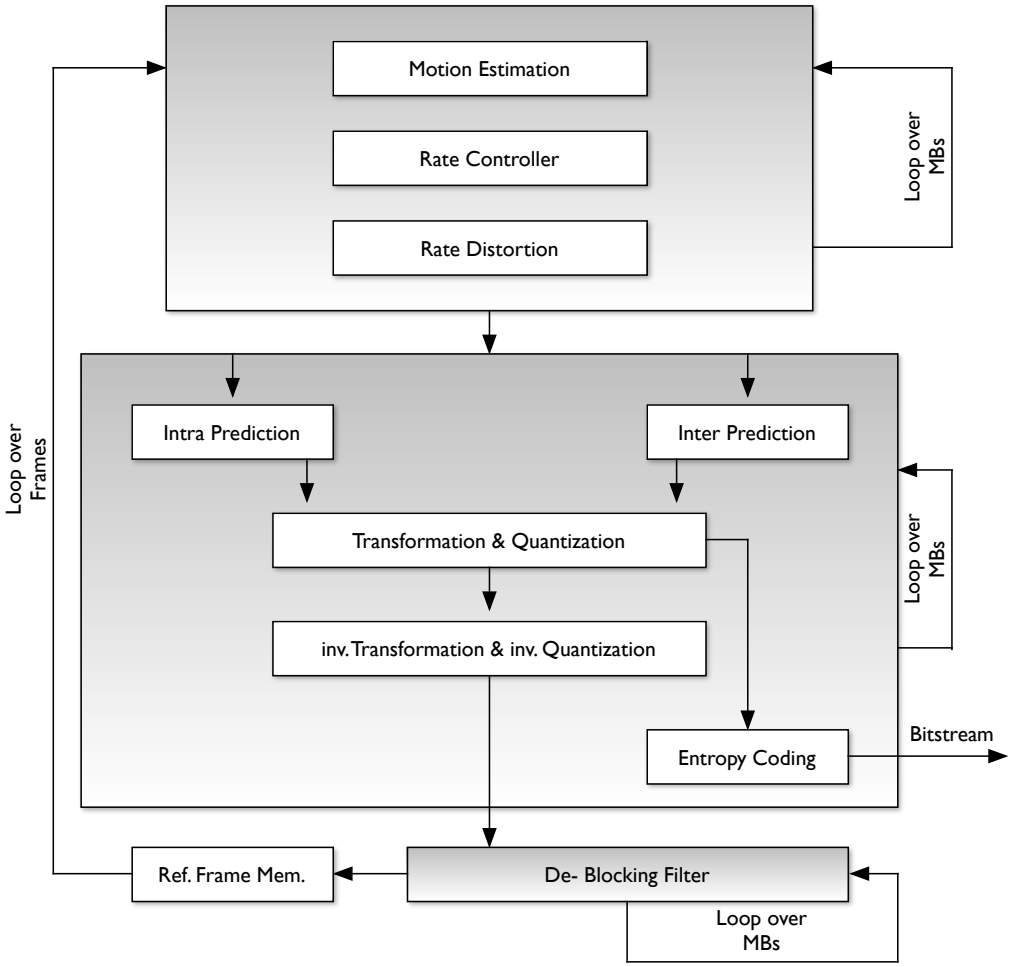

Figure 3.1: Simplified encoder structure [6]

celerated. Then they have redefined the structure and data-path so that the different parts of the encoder become more independent. With this, they have built FPGA/ASIC based prototypes to evaluate their architecture. They do not go much into detail about their concrete hardware implementations, but their separation of different functionalities provide many helpful hints for our project.

One of the main points in their redesigned encoder structure is the separation of the motion estimation loop and the encoding loop, creating the possibility to design independent hardware accelerators for each of these parts. Giving the possibility to perform the complete motion estimation of one complete frame before starting the encoding loop. This is done by storing all resulting motion vectors for one frame and its references in memory and reuse them later in the encoding loop. A simplified block diagram of their architecture is shown in Figure 3.1.

Available products

There are already a couple of video encoder acceleration products that implement

H.264 available on the market. These can be divided into two categories. The first one consists of (consumer-ready) hardware products which can be connected to a normal computer and work together with specialized software. Examples for these products are the Turbo.264 HD USB video converter [7] and the Matrox CompressHD PCIe acceleration card [8]. Both can be seen in Figure 3.2. However, even if there are already working accelerators available, none of these solutions was developed for the acceleration platform provided by Maxeler Technologies. This separates our project from existing solutions and it will be interesting to compare our novel acceleration prototype with existing solutions.

Figure 3.2: Two different commercial acceleration products [7][8]

3.2 Different acceleration approaches

There are in general different approaches to design a hardware accelerator for H.264. The first one mentioned by Lehtoranta et. al. in [9] uses a so called soft-core implementation which is basically working like multiple normal CPUs. These soft-cores can be replicated and placed several times on one FPGA or ASIC. With this, the hardware accelerator works like a co-processor and gains speedups through a higher level of parallelism.

The second approach is using specialized hardware which can only compute its encoding related tasks. However, this can be done very fast and energy efficient by such a customized hardware. This second approach is widely used and presented for example in [10], [4] and [11] for motion estimation as well as for transformation and quantization computations.

The H.264 encoding process is dividable in three main parts: Motion estimation, transformation and quantization, and entropy coding. Each of these parts contains some computational hot spots which should be accelerated as mentioned in [6]. Nevertheless, the motion estimation part has got the biggest acceleration potential, since it normally consumes up to 80% of the overall encoding time [4] and [5]. It is of course possible to implement all ports of the H.264 in one piece of hardware, but especially for our project on Maxeler DFEs, a separation of the three parts is needed due to the different data flows of these parts. In the following three sections further optimization and hardware acceleration approaches are presented. They are categorized into motion estimation, transformation and quantization, and encoding.

3.2.1 Motion estimation

Motion estimation is one of the main techniques used in state of the art video encoding standards, but it is also the most time consuming part as mentioned in [4], [5] and [12]. This is, because parts (macro blocks) of the current frame must be found that also occur in one of the previous frames. An example for this is given in Figure 3.3 where blocks from the last four frames are reused in the current frame.

To find matching blocks in two different frames, an algorithm is needed which computes the similarity between two blocks. This is often done with the sum of absolute differences (SAD), which is calculated by summing up the absolute differences between each pixel of two blocks, resulting in one single value per tested block pair. These results can then easily be compared and the block with the minimum SAD value is that one, that is most similar to the original block.

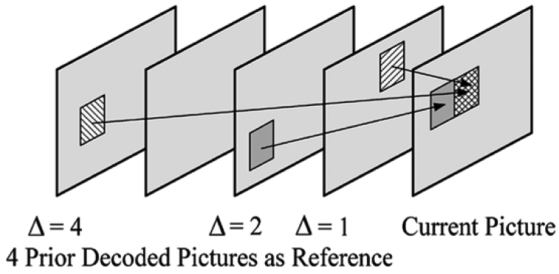

Figure 3.3: Multiframe motion estimation [12]

To accelerate the process of motion estimation different kinds of search algorithms have been proposed. Some of them use simple search patterns to reduce the amount of needed comparisons. Examples for these are Three-Step-Search (TSS) [13], Four-Step-Search (4SS) [14], or Diamond Search (DS) [15] and [16]. Other algorithms try to achieve better results by implementing more complicated search strategies. Examples are Enhanced Predictive Zonal Search (EPZS) [17], Cross-Diamond-Hexagonal search [18] and Simplified Unified Multi-Hexagon Search (SUMH) [19] and [20].

Implementing these search algorithms in hardware can be very difficult, especially implementations of the algorithms with complicated control structures. So previously special hardware oriented algorithms have been proposed which have the design goal to be implementable on hardware with a small amount of hardware usage (e.g. FPGA utilization). Examples for these algorithms are presented by Ndili *et al.* in [4] and Rahman *et al.* in [21]. Beside this, some work has been done in designing hardware based motion estimation in general, especially architectures to compute SAD values faster and more efficient presented in [22] and [11]. We have also already worked on such hardware based motion estimation approaches proposed by others during our project group seminar [23] and [24].

For this project will at first focus on a simple Fast Full Search implementation on Maxeler DFEs during the first milestone, resulting in a first prototype which can be seen as a proof-of-concept which can be used for comparison and validation. In our further work we will try to implement better suited algorithms to produce a faster encoder. However, all the hardware optimized algorithms which are mentioned here, can just be used as

hints and inspiration for our designs, since all of them use memory based communication approaches which do not directly fit to the communication flow of Maxeler DFEs.

3.2.2 Transformation and quantization

Transforming (T) to frequency domain and quantization afterwards are necessary in H.264 (also exist in other video and image coding standards); To exploit the less perceptual sensitivity of human vision in observing certain regions of the picture (domains with high frequency variation in brightness), and in order to provide an efficient spatial correlation between pixels (e.g. a lot of values of 1) in a block for further entropy coding to compress the data, every block undergoes transformation at first and down-scaling—quantization (Q), at next. In predictive video coding particularly (e.g. H.26*), the part of the pixel block that can not be constructed based on prediction references, i.e. the regions/pixels in the original block of the frame to be encoded, whose values are different from predicted block, which is called residue—the prediction error, is subjected to TQ (Transform and Quantize).

H.264 improved this part considerably by providing new features such as: Reducing the size of processing block (performed on 4x4 instead of 8x8 as in previous versions of the standard and in JPEG); Since H.264 is heavily based on prediction, even in Intra mode. That means there might be multiple references employed for a single MB prediction and they may differ in size (e.g the use of 4x4 motion segmentation). Therefore, the spatial correlation, can not be observed effectively in larger areas (e.g. 8x8 blocks), and thus, TQ in H.264 is performed on 4x4 blocks. Another advantage of this processing size reduction is the less computations and processing word length. i.e. 16 bit is required instead of 32 bit in previous versions of the standard. In addition, the so-called ringing artifacts present as a noise on the edge areas of a block, is reduced using this approach.

A new transformation approach, Integer Cosine Transform (ICT), whose resulting coefficients are no worse than DCT, yet providing an approximation to DCT with a multiplication free integer function; Multiplications are replaced by bit shifting and additions, improving efficiency on implementation. This feature can be seen as the hallmark of new TQ approach in H.264, since it avoids drifting effect on reconstructed block between encoder and decoder that existed in former versions of the standard. Drifting arises from the fact of different floating point calculations on inverse transformation between different encoders and decoders.

An additional transformation, Hadamard, following the DCT-like transform which is applied in Intra mode to the luma component of 16x16 MBs and the chroma component of all other Intra MBs; 2x2 block sizes are processed for chroma component. The Hadamard transform, improves compression performance for very flat areas (regions with least variation in pixel values) the quantization Parameter (QP) is introduced to control the compression ratio by varying among 52 values. Each step provides 12% of compression.

And by defining possible transform of suitably correlated chroma components of 8x8 blocks—so that to exploit correlation in a larger area where appropriate, besides transforming 2x2 blocks which form adjacent 4x4 blocks, the standard constitutes a hierarchical model of transformation [25], [12], and [26].

Agostini *et al.* [27] provided an architecture which embeds all three types of transfor-

mation in one single unit (and as well for the inverse part in another unit, again altogether 4x4 ICT, 4x4 Hadamard, and 2x2 Hadamard), making use of a pipeline and synchronizing the result of three different transform types (since not all samples are conducted through all three cores, syncing is indispensable and implies different processing time for different TQ types), their architecture produces one output (and consumes one input) sample per clock cycle, achieving a high throughput capable of supporting HDTV streams. Despite there is no implementation detail provided, their idea is to maximize the throughput, is interesting to our project since it shows some successes already obtained on similar design fashion to our platform: Maximizing throughput, pipelining, and also interestingly using no memory for storing intermediate calculation results; This last one could be extremely considerable as in our solution there exists limited amount of memory.

In the work of Ben Atitallah *et al.* [28], the optimization on HW solution aims at sharing resources for different kinds of prediction mode blocks. Again here, the pipelining approach is exploited, as well as parallelization (clear features possible to invest on our solution). The architecture provides a high throughput of 4x4 blocks through 16 input lines (each input for one 4x4 block), performs the 2-D integer transforming and process them in parallel by sixteen replicated units for quantization. In case of Intra mode, the data flow branches before quantization (and after ICT) to undergo Hadamard transform, and then quantization. Interestingly, to reduce the complexity of 2-D transform, it is spread to four replicated 1-D units performing on rows/columns of 4x4 blocks; This seems very promising to be implemented in our solution and gain performance, although the performance is not single clock cycle production.

Two other approaches focusing on highly parallel implementations of TQ are presented in [29] and [30]. Nevertheless, taking into account the considerations of our HW/SW solution (i.e the resource availability with respect to other units of encoder ported to HW, and the design paradigm of Max. Tech.), our specific implementation would be rather a transformed version of functions presented on JM source and aggregated ideas from other related works, where useful.

3.2.3 Entropy coding

H.264 profiles support different kinds of entropy coding algorithms. In [31] and [32] a sophisticated overview of run-length encoding (RLE) is provided, also variable length coding (VLC), context-adaptive variable length coding (CAVLC) or context-adaptive binary arithmetic coding (CABAC) and their roles in the entropy coding process are discussed explained. VLC is often referred to as Exponential-Golomb coding in the field of H.264. To give a short context overview of these, CAVLC and VLC are adaptive, enhanced versions of RLE while CABAC serves as an even more complex binary coder for the higher quality videos. In H.264, simple VLC is applied to syntax symbols which are not considered by CAVLC for the baseline profiles or by CABAC for the higher profiles. However, CAVLC and CABAC cover the main amount of entropy coding. When a codec architect needs to decide which coding algorithms he wants to support, he also needs to consider the additional processing requirements of CABAC in comparison to CAVLC.

Baseline coding in H.264 is actually a mixture of CAVLC and VLC. Silva *et al.*[33] describe a hardware architecture for these two blocks in VHDL. They ported their results

to an Altera Stratix-II FPGA in order to verify that the CAVLC and Exp-Golomb modules reached an throughput of 15.9 million samples per second for Exp-Golomb, and 103.8 million samples per second for the CAVLC coder. These results indicated that their design is capable to process HDTV frames in real time.

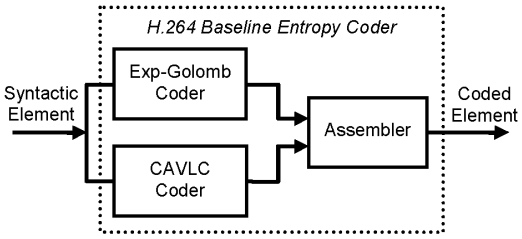

Figure 3.4: Block diagram of baseline profile entropy coder [33]

Figure 3.4 shows how the two main blocks allow parallel execution for a high throughput in the H.264 baseline entropy coder.

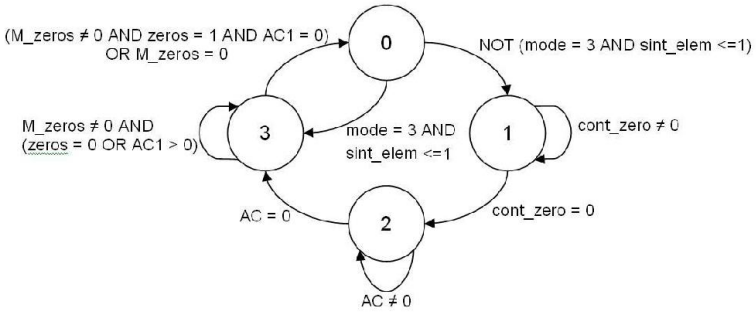

Figure 3.5: Finite state machine for the CAVLC output code generation [33]

Silva *et al.*[33] implemented the modules through a finite state machine design. Figure 3.5 depicts the state machine model of the CAVLC module. One can observe that four states with few state change dependencies were sufficient to model the binary code generation for the CAVLC coder part.

When it comes to higher quality video sequences, CABAC reveals improved encoding results compared to baseline coding approaches as stated before. This observation is directly drawn from the fact that CABACs trade-off between additional processing requirements and compression efficiency is better suited for higher amounts of data. Here, CABAC output is actually around 20-30% smaller than its CAVLC pendant. Osorio and Javier D. Bruguera [34] investigated an architecture for arithmetic coding adapted to the characteristics of CABAC, including optimized use of memory and context managing. They were are able to encode more than two symbols per cycle at a maximum speed of 185 MHz and tested their design on a Virtex-II 2000 FPGA device with a maximum speed of 92 MHz. However, the do also state that their approach may be suitable for broadcasting and video recording applications only if RDO is disabled.

Pastuszak presented in [35] an approach where the FPGA encoder supports CAVLC and CABAC. He specifies an architecture which saves hardware resources by sharing the same logic and storage elements for both coding modes. The implementation also enables the codec to code symbols in parallel for higher efficiency. More specifically, Pastuszak divided the coding algorithms into different stages and combined the parts which do map each other in their functionality. The resulting architecture is presented in VHDL, exposing the pipelines and components in a very detailed manner. Additionally he investigates the advantage of the parallel symbol encoding in the CABAC mode. His testing results show an improved throughput of 40% with active parallel symbol encoding.

For our project, the stated approaches can all work as examples for our own implementation on the Maxeler DFEs. All three can be taken as reference to get to an efficient hardware implementation of the entropy coding part. However, the presented architecture of the CABAC approach can raise additional hardware resource requirements due to the streaming architecture of the DFEs. Moreover, DFEs normally need to know its own output data size. For the presented entropy coding techniques it is hard to distinguish their output which could become problematic with DFEs. We need to analyze these issues further before starting to implement. Anyways, before even starting to work on CABAC, we first will focus on CAVLC and baseline profile, but once we have both, we could look into further improvements by increasing parallel computations as shown by Grzegorz Pastuszak presented in [35].

To set the entropy coding into the overall computational context of H.264, profiling results (cf. Chapter 2) revealed that it demands a small amount of computations in comparison to motion estimation/compensation. Just like as this does also apply to the other parts, transformation and quantization, entropy coding represents a smaller priority when we decide which features we want to include. Accordingly, motion estimation is still the most important part.

3.3 Distinction between our project and others

Most of the mentioned hardware acceleration prototypes were implemented on standard FPGA development boards and designed in VHDL. In our project we try to implement an accelerated H.264 video encoder on Maxeler's data flow acceleration platform. Our prototype will be a novel approach, since we can not directly apply solutions which were presented in this chapter.

The main difference is the data handling: We noticed, in our previous work with the Maxeler technology, that the streaming approach of the DFEs often raises additional challenges when working on two dimensional image or video data. It is much easier to have real random access memory for these tasks. However, DFEs offer different specialized memory approaches, like LMem. These approaches have to be tested and evaluated during our project to find the best working solution. The main benefit of the Maxeler DFEs is the simplified abstract design methodology using the Java programming language. This speeds up the implementation and prototyping workflow, compared to traditional hardware design processes.

4 Organization

The general project organization follows the agile software development method with grassroots democracy process. In this process there is no leader, all the members of the project have equal responsibility. Decisions are taken by discussions among all the group members and the majority member's preference is the chosen decision. The agile software development method uses iterations. In this project, each iteration has a unique milestone and achieving those milestone is the goal in each iteration.

4.1 Agile software development

Agile software development is followed in this project for the following reasons:

- Testing is an integrated part of the whole development process, which improves the quality of the project.
- Incremental development of the features by the iterative nature of agile software development.
- Regular release of a working project code at the end of each iteration.
- Easy adaptation for any change in circumstances.
- Self organizing development process provides more team member involvement and friendly environment.
- Less end to end project overhead is required.

4.2 Meetings

Project meetings are conducted twice a week. A mandatory team meeting on every Thursday morning of the week and another non-mandatory meeting on every Tuesday morning.

The mandatory meeting is attended by all the members of the group. In this meeting the current status of all the tasks and the whole project is discussed. Then the further tasks for next week is developed by discussion among the team members. The members also track the progress towards the approaching milestone on each of this mandatory meeting. Tasks are created and the members are assigned to the tasks in this meeting. Tasks are assigned to the members either by voluntarily or by asking a member. Any member with knowledge about the task by previous experience and seminar topic is assigned to that task. If any of the member fails to attend this mandatory meeting without any prior

excuse, the member has to pay the penalty of buying cookies for the other members in the next mandatory meeting.

Though the Tuesday meeting is not mandatory, all the project members are expected to attend the meeting. Failing to attend the meeting without any prior excuses has no penalty but for work ethics the members are expected to inform their absence in advance. The tasks allocated in the mandatory meeting mostly requires interaction between other team members, these related team members of a task use this Tuesday meeting for developing the task, integrating their work and discuss on any problems facing.

Additionally, a Wednesday afternoon time slot is reserved for each week. In a normal week, the Wednesday time slot is open for all the team members to work together and experience team work, which helps to motivate each other. In addition, the Wednesday time slot will also be used for weekly team meeting if continuous cancellation of weekly team meeting happens due to holidays.

4.3 Tools

For project organization GitHub and Redmine are used. All the project code, documents and presentations are shared using a GitHub repository. GitHub wiki is used to update the minutes of meeting conducted on Tuesdays and Thursdays. GitHub wiki is also used for tracking the members attending each meeting and tracking on the penalty for the absentee of the meetings. Informations that are discussed in the meeting are also updated in GitHub wiki. In addition, individuals who missed deadline for tasks are tracked in GitHub wiki. GitHub wiki also contains the private contact details of all the team members.

Redmine is used to represent the whole project plan with each milestones. Tasks under each milestones are created and are assigned to the team members according to the meeting outcomes. Each task has a deadline and the work progress of each member is tracked through Redmine. If any bug or new task identified by a team member, the member creates the task or bug in Redmine and the details are immediately informed to all the other team members via email. Redmine is also used to track the overall project progress for each milestone and the tasks that are pending. Skype is also used by all the team members to have real-time conversations between team members throughout the week. A mailing list which includes all the project members email address is used for email communication between members.

4.4 Member role

Each team member in the project is responsible for both the project development and the project organization. Each member's project development responsibilities include planning, analyzing, coding, documenting, testing, discussing and evaluating. The project organization role of each member involves frequent interaction between all members, using the project organization tools efficiently and update of current status wherever possible. Each member updates his error-less compilable code to GitHub repository as frequent as possible, even if the code change is very little. While committing the error-less code, the member also provides a brief information about the code change in the commit message.

The same commit rule is applied for any information that is committed to the repository. When a member has not attended a team meeting with or without excuse, the member goes through the minutes of meeting in GitHub wiki mostly within the day of the meeting conducted. It is the whole responsibility of the member to identify the task allocated in a team meeting. If any confusion arise about the task allocated, the member clarifies with the team members through email or Skype immediately.

From the task assigned information in the team meeting, the member creates the task in Redmine if not already created by others and assign it to himself. Each member of a task updates the task detail in Redmine about the problems faced, unique experiences and the solution identified, if more than one solution is identified then the reason is provided for the chosen solution. These information provided by the member are used by the team when a future similar task is identified and also helps in documentation. The members also update the status of their tasks in Redmine, hence the status of all the tasks is known to the whole team in the team meeting even in the absence of other members and helps to decide about the project progress in the meeting. If a member identifies any bug while testing or coding, it is immediately reported in Redmine as a new bug, which will email all the team members about the new bug. When a team member faces problem in the allocated task, the member discusses the issue with other team members through Skype or email within the next day. Issues faced while the task progress are not be carried till the end of the task deadline without the knowledge of the other team members.

4.5 Milestone presentation

At the end of each iteration during the project development, a milestone presentation is presented by the team members to the project mentors. This presentation has two parts. In the first part of the milestone presentation, the plan for the completed milestone and its current status will be described in detailed. If any of the planned feature is not completed with in the milestone, the reason for failure to complete the feature and current status are explained in detail. Also the members will propose a revised plan for the incomplete feature in the new milestone work plan, in part two of the current presentation. Demos and screen shots of results from the milestone features will be provided wherever it is possible in the presentation. In the second part of the milestone presentation, the next new milestone's work plan is described in detail which also includes any previously incomplete feature's revised work plan.

The project mentors provide their feedback, for the completed milestone and about the features in the next milestone. This ensures that the project does not deviate from its main goal, no features are missed and the quality satisfaction is obtained. A small on spot discussion in held between the team members and the project mentors at the end of the milestone presentation. During this discussion any disagreement of the proposed next milestone work plan by the project mentors will be discussed. If the project mentors request some additional features to be included for the next milestone, the team members will analyze the possibility of including and will propose a new work plan with in the next two days.

4.6 Limitations

The grassroots democracy process of organization of the project has few limitations. Chances of no clear majority in any decision taking is possible, in such cases the discussion is continued until a clear majority is arrived. Since everyones goal is to achieve the milestone, the repeated discussions have provided fruitful results till now. If a team member does not follow the rules or have not progressed in tasks, the issue is discussed with the team member and motivated to work harder for the progress of the project. No leader roles are used for such discussions, all discussions are made in the team meeting involving all the team members. Tasks are just shared between the members in the most possible satisfactory way, though at times some tasks are assigned to the members with no other choice. These limitations in decision making, task allocating and member commitment are well known by project members, till now members are able to work around them. If in case this approach fails in the future, a total organizational change like a project leader approach will be considered.

4.7 Members specialization

Member Name	Specialization
Andreas Kohlos	Redmine, Gprof
Farhan Ahmed	Kernels, Transformation
Henning Schmitz	Kernels, RD
Manuel Peuster	Compiling, Hostcode (incl. SLiC), ME
Robert Mittendorf	Hostcode, ME
SeydEsmaeil SeydAshrafi	Kernels, Entropy coding
Vignesh Makeswaran	Redmine, Kernels

Table 4.1: Specializations of the team members

5 Workplan

5.1 Plan overview

The work plan is divided into several milestones. The work plan to achieve each milestone will be 4 to 6 weeks long depending upon the number of features planned to achieve in the milestone. Detailed features and tasks for each milestone are discussed and fixed in the first team meeting of the milestone. The features and tasks depend upon the knowledge attained, problems faced, work experience and work progressed from the previous milestone. At the end of work plan for each milestone, will have a working implementation with additional features or performance or both compared to the previous results. Internal presentation about the project status and new results at the end of each milestone will be presented to the project mentors. A final project presentation will be presented at the end of the project to the whole department work group.

5.2 Identified tasks

The following is a list of possible and optional tasks identified during initial project analysis. The implementation of these tasks completely depends upon the progress of the project.

- Performance analyze JM reference software using Gprof tool, to obtain the processing time on motion estimation, transformation & quantization and entropy coding.
- Implement complex motion estimation algorithm before the encoding loop.
- Acceleration of motion estimation, transformation & quantization and entropy coding.
- SAD kernel implementation with optimal search algorithm.
- 4x4 TQ kernel implementation with possible parallelization.
- Analyze entropy coding especially CABAC for possible implementation.
- Entropy coding implementation beginning with CAVLC and Baseline profile.
- Automated weekly performance testing.
- Performance comparison with similar solutions.

5.3 Milestones

Currently five milestones are planned including a backup milestone for buffer work when any of the milestone plan exceeded during implementation.

5.3.1 Experiments Milestone

The first milestone of the project involves the analysis and basic implementations. The work plan to achieve this milestone is scheduled from the mid of March 2013 to the beginning of May 2013.

Experiments Milestone Tasks

- Analyze the JM reference software
- Initial project host code and kernel code structuring
- Basic SAD kernel implementation for motion estimation
- Basic TQ kernel implementation for transformation and quantization

5.3.2 Fledge Milestone

The second milestone of the project involves the evolution to complex implementation of features over the basic code developed in the experiments milestone. The work plan to achieve this milestone is scheduled from the beginning of May 2013 to the mid of June 2013.

Fledge Milestone Tasks

- Upgrade SAD kernel as the following:
  - Reduce number of kernel calls
  - Reduce redundancy in input data
  - Optimize search strategy
  - Combine 4x4-results
- Complete the TQ kernel and implement TQ 4x4 kernel.
- Optimize the implementation to become faster than JM implementation.
- Analysis and conclusion of final codec features to be accelerated (including documentation).
- Implementation of complex motion estimation algorithm.
- Do performance measurements, comparisons & document the results.
- Final setup up of continuous integration (CI) environment.

5.3.3 Grownup Milestone

The third milestone of the project involves the final feature implementations and concentrated on improving the performance. The work plan to achieve this milestone is scheduled from the mid of June 2013 to the end of July 2013.

Grownup Milestone Tasks

- Attaining the best possible performance
- Final trade-off decisions and implementations
- Performance evaluation and comparison

5.3.4 Polishing Milestone

The fourth milestone of the project involves the final touch on the code and final project documentation including the software manual. The work plan to achieve this milestone is scheduled from the end of July 2013 to the end of August 2013.

Polishing Milestone Tasks

- Final project documentation
- Clean up of project code
- Clean up of project related documents and informations in GitHub and Redmine

5.3.5 Backup Milestone

The last milestone of the project is a buffer milestone to accommodate any work exceed the planned amount of time. The work plan to achieve this milestone is scheduled from the beginning of the September 2013 till the end of the September 2013. There is no rigid plan for this milestone as this milestone might not be required if the work plan is successful without any major unexpected work plan breach.

5.4 H.264 Milestones gantt chart

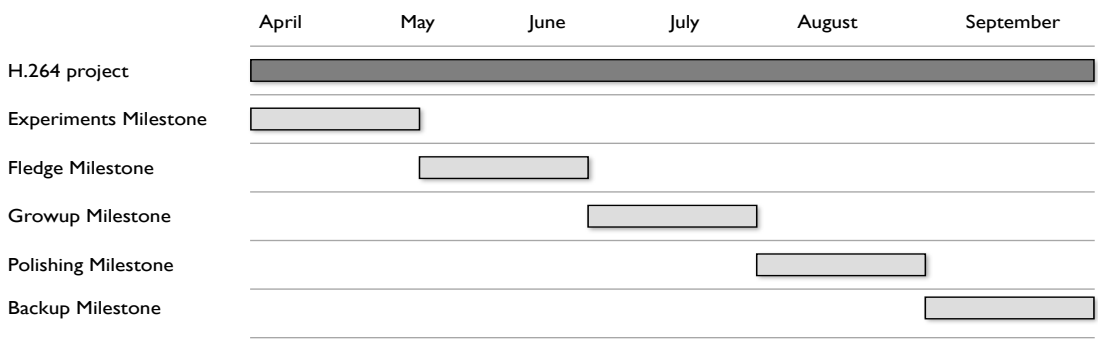

Figure 5.1: H.264 Milestone gantt chart

Bibliography

[1] The H.264/AVC reference implementation JM Software. http://iphome.hhi.de/suehring/tml/.

[2] JPEG compression artifacts Jens Niebuhr. https://commons.wikimedia.org/wiki/File:Jpegartefakt90-20.jpg.

[3] gprof tool. http://www.gnu.org/software/binutils/.

[4] O. Ndili and T. Ogunfunmi. Algorithm and architecture co-design of hardware-oriented, modified diamond search for fast motion estimation in H.264/AVC. *Circuits and Systems for Video Technology, IEEE Transactions on*, 21(9):1214–1227, 2011.

[5] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video coding with H.264/AVC: tools, performance, and complexity. *Circuits and Systems Magazine, IEEE*, 4(1):7–28, 2004.

[6] M. Shafique, L. Bauer, and J. Henkel. Optimizing the H.264/AVC video encoder application structure for reconfigurable and application-specific platforms. *Journal of Signal Processing Systems*, 60(2):183–210, 2010.

[7] Elgato. http://www.elgato.com/video/turbo-264-hd.

[8] Matrox: Digital Video Solutions. http://www.matrox.com/video/en/products/compresshd/.

[9] I. Lehtoranta, E. Salminen, A. Kulmala, M. Hannikainen, and T. D. Hamalainen. A parallel MPEG-4 encoder for FPGA based multiprocessor SoC. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 380–385, 2005.

[10] I. Amer, W. Badawy, and G. Jullien. A design flow for an H.264 embedded video encoder. In *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on*, pages 505–513, 2005.

[11] S. Yalcin, H. F. Ates, and I. Hamzaoglu. A high performance hardware architecture for an SAD reuse based hierarchical motion estimation algorithm for H.264 video coding. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 509–514, 2005.

[12] G. Sullivan and T. Wiegand. Video compression-from concepts to the H. 264/AVC standard. *Proceedings of the IEEE*, 93:18–31, 2005.

[13] R. Li, B. Zeng, and M. Liou. A new three-step search algorithm for block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 4:438–442, 1994.

[14] L. Po and W. Ma. A novel four-step search algorithm for fast block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(3):313 –317, 1996.

[15] J. Tham, S. Ranganath, M. Ranganath, and A. Kassim. A novel unrestricted center-biased diamond search algorithm for block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 8(4):369 –377, 1998.

[16] S. Zhu and K.-K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *Image Processing, IEEE Transactions on*, 9:287–290, 2000.

[17] A. M. Tourapis. Enhanced predictive zonal search for single and multiple frame motion estimation. *Proceedings of SPIE*, 4671:1069–1079, 2002.

[18] C. Cheung and L. Po. A novel cross-diamond search algorithm for fast block motion estimation. *Circuits and Systems for Video Technology, IEEE Transactions on*, 12(12):1168 – 1177, 2002.

[19] O. Ndili and T. Ogunfunmi. FPSoC-based architecture for a fast motion estimation algorithm in H.264/AVC. *EURASIP Journal on Embedded Systems*, 2009:893–897, 2009.

[20] X. Yi and N. Ling. Improved normalized partial distortion search with dual-halfway-stop for rapid block motion estimation. *Multimedia, IEEE Transactions on*, 9:995–1003, 2007.

[21] C. A. Rahman and W. Badawy. UMHexagonS algorithm based motion estimation architecture for H.264/AVC. In *System-on-Chip for Real-Time Applications, 2005. Proceedings. Fifth International Workshop on*, pages 207–210, 2005.

[22] R. Chen and J. Fan. Complexity reduction for SOPC-based H.264/AVC coder via sum of absolute difference. In *ASIC, 2007. ASICON '07. 7th International Conference on*, pages 1277–1280, 2007.

[23] R. Mittendorf. Evaluation of an adaptive computationally-scalable motion estimation for the hardware H.264/AVC encoder. In *UPB, 2013. Seminar PG CC for video codecs*, 2013.

[24] M. Peuster. Motion estimation with hardware oriented modified diamond search. In *UPB, 2013. Seminar PG CC for video codecs*, 2013.

[25] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Trans. Cir. and Sys. for Video Technol.*, 13(7):560–576, July 2003.

[26] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. Low-complexity transform and quantization in H.264/AVC. *IEEE Trans. Cir. and Sys. for Video Technol.*, 13(7):598–603, 2003.

[27] L.V. Agostini and S. Bampi. FPGA based architectures for H.264/AVC video compression standard. In *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pages 1–2, 2006.

[28] A. Ben Atitallah, H. Loukil, and N. Masmoudi. HW/SW TQ/IQT design for H.264/AVC. In *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*, pages 13–16, 2011.

[29] L. Ling-zhi, Q. Lin, R. Meng-tian, and J. Li. A 2-d forward/inverse integer transform processor of H.264 based on highly-parallel architecture. In *System-on-Chip for Real-Time Applications, 2004.Proceedings. 4th IEEE International Workshop on*, pages 158–161, 2004.

[30] Y Li, Y. He, and S. Mei. A highly parallel joint VLSI architecture for transforms in H.264/AVC. *Journal of Signal Processing Systems*, 50:19–32, 2008.

[31] S. SeydAshrafi. Context-based adaptive encoding with hardware acceleration. In *UPB, 2013. Seminar PG CC for video codecs*, 2013.

[32] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):620–636, 2003.

[33] T. Silva, J. Vortmann, L. Agostini, S. Bampi, and A. Susin. FPGA based design of CAVLC and Exp-Golomb Coders for H.264/AVC baseline entropy coding. In *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, pages 161–166, 2007.

[34] R. R. Osorio and J. D. Bruguera. High-throughput architecture for H.264/AVC CABAC compression system. *Circuits and Systems for Video Technology, IEEE Transactions on*, 16(11):1376–1384, 2006.

[35] G. Pastuszak. A high-performance architecture of the double-mode binary coder for H.264/AVC. *IEEE Trans. Cir. and Sys. for Video Technol.*, 18(7):949–960, July 2008.

List of Figures