An exploration on hardware/software co-design H.264 decoder

Yiyao Hu, Rui Chen
Department of Electrical Engineering, UCLA

Abstract— For the H.264/MPEG-4 AVC, there are purely hardware codec and solely software solutions. They are known to have either the best performance in terms of energy and speed or the flexibility. This project aims at examine the possible partitions between HW/SW and evaluate its tradeoffs between the flexibility and performance.

Keywords-H.264, bitstream, deblocking filter

I. INTRODUCTION

H.264/MPEG-4 AVC has becoming a popular codec standard since its first release. Compare to its predecessors, H.264 has higher performance and requires a more complex codec structure. Architecture based on ASIC are known as the most power efficient and possess the highest performance in terms of speed. However, there are also alternative solutions which based on CPU controlled architecture. The CPU controlled topology has a higher flexibility in encoding, and can also supports more features on container format, but usually requires hardware acceleration. However, [1] indicates that as the development of the RISC, some solely software decoders are also available.

In this projects, two reference designs were majorly examined. One is the Nova decoder designed by Xu[2], which is an open-source H.264 baseline decoder, and another is the software JM decoder version 17.2[3]. We proposed one configuration that can take the advantage from both the hardware and software sides of a system. Then, some optimization methods are discussed.

II. SYSTEM DESIGN

Generally, the H.264 decoder design can be decomposed into three main parts. The first part is the bitstream parser, and reconstruction path goes subsequently. The bitstream parser takes care of the syntax and extract the parameters. The reconstruction path basically doing the Inverse Transform/Inverse Quantization, Inter-prediction and Intra prediction. The final stage is the deblocking filter. The decoder as a whole accepts the bitstream information from the channel or memory and sends the decoded data to a display controller.

As the bitstream parser is dealing with context-adaptive and highly correlated data, it requires basically a sequentially flow. It does not have high demand on computational power or memory access.[2] Therefore, it is considered that the bitstream parser could be done based on software implementation. What is really needed to be implemented in hardware is the reconstruction datapath. This part of the decoder requires a lot of computation and in order to make it real-time, hardware acceleration is a necessary.

The proposed H.264 decoder is shown in the following graph.

As shown in the fig.1, the essential parts which are shown blue here, are to be implemented in hardware whereas the bitstream parser and display controller are to be implemented in software. The developed board that we are using is the XUP Virtex II Pro. This board share the common feature that many FPGA development board, which contains one or more RISC embedded. This RISC can be programmed to achieve various customized tasks. Therefore, this whole system could be implemented and tested in a recent FPGA development environment and achieve a realtime H.264.

In this project, we will use open source code to implement our design. a low power design methodology for video decoding is proposed and applied on a H.264/AVC baseline decoder. At the algorithm level, the computational complexity will be optimized. At the architecture level, pipeline and parallel are widely adopted to reduce operating frequency; hierarchial memory organization moves the majority of data access from larger external memories to smaller ones; resource sharing reduces total switching capacitance by function reconfiguration. At the circuit level data dependent signal-gating and clock-gating are introduced which are dynamic techniques for power reduction; multiplications, which account for large chip area and switching power, are reduced to minimum through proper transformations.

Fig.2 provides an overview of the proposed video decoding system architecture. Generally it can be divided into three parts, a bitstream RAM, a video decoder, and a display.

Fig.1

Fig.2
controller with LCD display. We will consider our time schedule to implement parts of them.

### III. ARCHITECTURAL OPTIMIZATION

The mapping from the algorithm to the proper hardware architecture is highly related to design specifications, such as function, cost, performance, area, and power. Owing to the tough real-time constraint, pipelining and parallelism are the two most widely adopted techniques in video codec design. In addition, due to the demanding memory access and the associated huge accessing power, memory organization and access pattern should be intelligently devised. Hardware cost is also an important issue for video codec design. Both parallelism and pipelining would result higher area cost. Simple duplicating multiple processing elements or inserting pipeline registers without limitation may not be an optimal way. The optimized goal should be a just enough parallelism and pipelining with as higher as possible hardware utilization.

#### A. Hybrid Pipeline Control

Pipelining is a very common technique used in digital design in order to push the throughput. However, unlike the General Purpose Processor (GPP) or the Digital Signal Processor (DSP) which usually have fixed pipeline, the H.264 decoder is more flexible. The work done by each pipeline stage in a video decoder is highly depend on the video type and its characteristic. Xu[2] has proposed a scalable pipeline control technique that uses handshake communications and a controller to control the flow of each pipeline stage. However, it should be noted that this does not mean that the frame can be transmitted to the output as soon as the processing is done, because the frame has to be outputted at a fixed rate, e.g. 30FPS, otherwise, fastened or freezed pictures would be observed. What can be shared in time domain is the macroblock based pipelining. Since all the macroblocks in one frame are to be output at the same time. Therefore, their processing time can be shared and borrow one from another.

#### B. Parallel Architecture

For a complete IQIT decoder, generally there are three steps required to be carried out: inverse transform (IT), rescaling (IQ, inverse quantization) and final rounding (not the rounding inside IQ). The inverse transform can be further decomposed into two 1D transforms (1D-IDCT and 2D-IDCT, respectively) and a transpose memory. For DC coefficients, the rescaling runs after 2D-IDCT and there is no rounding operation; for AC coefficients, the rescaling comes before 1D-IDCT, and the rounding follows 2D-IDCT. For the entire IQ/IT decoding process for a 16×16 macroblock is illustrated in Figure 3.

#### C. Throughput Comparison

The table below shows the architecture, memory, throughput, gate counts comparison among the proposed deblocking filter and other reported designs. This work achieves highest throughput due to smallest filtering cycles/MB and maximum operating frequency.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>DMA[99]</th>
<th>HIECLOS</th>
<th>HIECOS2</th>
<th>HIOY34</th>
<th>ECIFLOS</th>
<th>LELES[96]</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing</td>
<td>hybrid</td>
<td>hybrid</td>
<td>hybrid</td>
<td>hybrid</td>
<td>hybrid</td>
<td>hybrid</td>
<td>hybrid</td>
</tr>
<tr>
<td>order</td>
<td>non-pipe</td>
<td>non-pipe</td>
<td>non-pipe</td>
<td>non-pipe</td>
<td>non-pipe</td>
<td>single-pipe</td>
<td>single-pipe</td>
</tr>
<tr>
<td>RAM type</td>
<td>dual-port</td>
<td>two-port</td>
<td>two-port</td>
<td>two-port</td>
<td>two-port</td>
<td>single-port</td>
<td>single-port</td>
</tr>
<tr>
<td>RAM size</td>
<td>1x10632X2</td>
<td>2x80X32</td>
<td>1x64X32</td>
<td>2x80X32</td>
<td>1x632</td>
<td>1x10632X2</td>
<td>2x80X32</td>
</tr>
<tr>
<td>Transpose buffer</td>
<td>8072</td>
<td>256</td>
<td>812</td>
<td>n/a</td>
<td>256</td>
<td>512</td>
<td>896</td>
</tr>
<tr>
<td>Technology</td>
<td>FPGA</td>
<td>0.25um</td>
<td>0.25um</td>
<td>0.25um</td>
<td>0.25um</td>
<td>0.25um</td>
<td>0.25um</td>
</tr>
<tr>
<td>Gate count</td>
<td>18K</td>
<td>206K</td>
<td>2K</td>
<td>13K</td>
<td>19K</td>
<td>218K</td>
<td></td>
</tr>
<tr>
<td>Processing</td>
<td>614</td>
<td>614</td>
<td>446</td>
<td>300</td>
<td>250</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>72MHz</td>
<td>100MHz</td>
<td>100MHz</td>
<td>100MHz</td>
<td>100MHz</td>
<td>200MHz</td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>11.7k</td>
<td>15.5k</td>
<td>16k</td>
<td>224k</td>
<td>333k</td>
<td>400k</td>
<td>998k</td>
</tr>
</tbody>
</table>

#### IV. IQ/IT

### A. LUT based divider and modulator

As a common technique, sometimes a system can bypass a full mathematical core by implementing it by lookup tables. This design uses lookup tables to do modulation and division, in which cases have the constant divider and modulator number. Since the design uses only fixed value division and modulation, as well as the dividend or the number to be modded does not have large wordlength (5 bits in this case), the number of LUTs are not quite big. The result shows that the lookup-table based design requires more flip-flops while the word-length optimized divider and modulator requires much more 4-input LUTs in the FPGA. This technique can be used to conserve the hardware without losing the accuracy if designed carefully. The result is confirmed in the simulation of these cores.

### B. Reuse of hardware

Three types of transforms are utilized in H.264 based on the nature of residual data being coded. In Intra_16x16 mode, a 4x4 Hadamard transform is used to processing the luma DC coefficients. A 2x2 Hadamard transform for the chroma array DC coefficients, and a DCT-based transform for all other 4x4 blocks in the residual data. The inverse Hadamard transform is shown as follows:

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 \\
1 & -1 & -1 & 1
\end{bmatrix}
\begin{bmatrix}
c_{00} & c_{01} & c_{02} & c_{03} \\
c_{10} & c_{11} & c_{12} & c_{13} \\
c_{20} & c_{21} & c_{22} & c_{23} \\
c_{30} & c_{31} & c_{32} & c_{33}
\end{bmatrix}
= \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 \\
1 & -1 & -1 & 1
\end{bmatrix}
\]

**Fig. 3**

The inverse Hadamard transform for the 2x2 chroma DC transform coefficients is defined as:
And the inverse transform for $4 \times 4$ residual transform coefficients is defined as:

$$f = \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 2 & 1 & -1 \\
1 & 2 & 2 & -1 \\
1 & -1 & 1 & \frac{1}{2} \\
\end{bmatrix} \begin{bmatrix}
C_{00} & C_{01} & C_{10} & C_{11} \\
C_{20} & C_{21} & C_{22} & C_{23} \\
C_{30} & C_{31} & C_{32} & C_{33} \\
\end{bmatrix} \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & 2 & 1 & -1 \\
1 & 2 & 2 & -1 \\
1 & -1 & 1 & \frac{1}{2} \\
\end{bmatrix}$$

Fig.4.

These kind of calculation usually requires complex calculations [2] and Malvar [4] has proposed fast algorithms for the calculations. Its schematics for the above three calculations are attached as follows.

Fig.5.

In the combined butterfly architecture, one multiplexers are used to select between the different configuration of $4 \times 4$ inverse Hadamard transform or the $4 \times 4$ inverse residual transform. For the $2 \times 2$ Hadamard transform, the two stage butterfly can be combined in one.

Fig.6

Fig.7

These fast algorithms can be further combined into a single structure, so that one butterfly can be reused. This can contribute to the reduce in cost of hardware and power.

Fig.8

These kind of calculation usually requires complex calculations [2] and Malvar [4] has proposed fast algorithms for the calculations. Its schematics for the above three calculations are attached as follows.

Fig.9

Verification

There are slight differences of RTL code for FPGA verification and AISC verification:

1) Clock gating in ASIC, which is used to reduce the power consumption on clock network and propagation combinational logic, is revised to clock enable due to the limitation of FPGA clock network as it can not perfectly balance the clock propagation delay.

2) Instead of instantiating foundry provided memory IP in ASIC, Xilinx FPGA utilizes “core generator” to produce RAM IP provided by Virtex2 device.

At the very beginning, it was expected that the verification can be done purely on the FPGA. According to some previous hand-on experience of FPGA, the FPGA educational board is capable of managing both software and hardware. However, after some time of exploration on the Xilinx Virtex II Pro development board and plenty of trial-and-error, some difficulties were found. Firstly, several compatible issues were encountered during tool-setup. It was finally found out that the latest version that supports Virtex-II is the ISE 10.1. Also, this version is not officially compatible with Windows version higher than Windows XP. Secondly, the software decoder successfully compiled in MS Visual Studio has some win32 libraries needed which makes it nontrivial to immigrate the original software decoder to the FPGA platform. Another issue is that our Virtex 2 FPGA board which integrates PowerPC RISC has a maximum of 64-kb on-chip memory, which might not be enough for the entire decoder. The decoder executables in Visual Studio platform is over 600kb. Technically, it is possible to add other peripherals such as on-board CF card or use some communication lines such as RS-232 or parallel port, but these techniques may need further drivers for the CF card or RS-232 driver on both the FPGA and PC side. We considered that this would be getting too far from the purpose of the project and finally we decided to save time by leave the software part of the decoder in Visual Studio. On the RTL side, after the profiling of a reference core, it was concluded by the synthesis report that the Virtex II FPGA does not have enough 4-input LUTs.
The synthesis summary is attached.

**Fig. 10**

It can be seen that the LUT required is beyond the capability of Virtex II and the whole design can not be mapped into the FPGA.

**A. IQIT**

The divider and modular block implemented by LUTs and full math cores are synthesized and compared. The following summary shows the total area of the IQIT block.

Simulations on the math-core divider and modulator were done to guarantee that they have the same behaviors to the LUT-based divider and modulator. The simulations for the math-cores are attached.

**Fig. 11**

The RTL for the LUTs are attached. The divider and modular block implemented by LUTs and full math cores are synthesized and compared. The following summary shows the total area of the IQIT block.

**VI Conclusion**

The system architecture, including pipelining, parallelism, and memory organization, is carefully explored. Different pipelining granularities, 16x16, 8x8, and 4x4, are evaluated. A hybrid pipeline architecture is proposed to balance the requirements of power, area, and throughput for the target application. Various parallelism combinations of 1pix/cycle, 4pix/cycle, and 16pix/cycle, are evaluated individually. To compromise between area and throughput, the 4pix/cycle parallel processing is chosen. Two 4pix/cycle processing sequences, 1x4 column and 4x1 row, are compared in terms of hardware cost, memory access, and throughput for various building blocks. Each building block is then designed with its most efficient pixel organization. To save the memory access power, a three-level memory hierarchy is proposed which trades additional small on-chip SRAM power for large savings on I/O and external memory access power.

The hardware/software architectures has many applications. Firstly, in the modern handheld devices which has RISC cores, the combined structure can save some chip areas without losing the real-time decoding capability. Secondly, since the bitstream decoding is implemented in software, which is more flexible to update and accepts various wrapping syntax. Also, the FPGA based hardware/software codesign provides the benefits in the decoder development since other parts of the decoder can be modeled in software while examining one or more specific part of a design in RTL.

**References**


[2] XU, Ke, Chinese University of Hong Kong. 2007
