H.264 Video Decoder Implemented on FPGAs using 3x3 and 2x2 Networks-on-Chip

Ian J. Barge
Expedition Technology Inc.
Dulles VA, USA
Email: ian.barge@marquette.edu

Cristinel Ababei
Dept. of Electrical and Computer Engineering
Marquette University, Milwaukee WI, USA
Email: cristinel.ababei@marquette.edu

***Abstract*—In this paper, we present the design and verification of the H.264 video decoder algorithm on FPGAs. The primary difference compared to previously reported designs is that the communication between the decoder modules is done via a network-on-chip in our case. The proposed design is a complete system level hardware design described in VHDL and Verilog. We report experimental results for two different implementations. The first implementation uses a 3x3 network-on-chip and is validated on the DE4 development board, which uses Altera's Stratix IV GX FPGA chip. The second implementation uses a 2x2 network-on-chip and is validated on the Cyclone V SoC FPGA, which is a smaller FPGA chip available on the DE1-SoC board. Both implementations will be released to the public domain with the hope that they will foster further research, in addition to facilitating replication and comparison to our results.**

***Index Terms*—H.264 video decoder; network-on-chip; FPGA implementation**

I. Introduction

The popularity of field programmable gate arrays (FPGAs) has continuously increased not only because of the reduced performance gap between FPGAs and application specific integrated circuits (ASICs) but also because of the great flexibility that reconfiguration offers when it comes to product development, maintenance and updates. Today, FPGAs are used in virtually all application domains. Examples of such applications include various algorithms used in image and video processing, computer vision, robotics, including video compression and decompression. An important development that has been taking place in the last decade or so is the emergence of networks-on-chip (NoCs) [1]. NoCs replace design-specific global on-chip wires with a generic on-chip interconnection network implemented by specialized routers that connect processing elements (PEs) to the network and facilitate communications or links between them. NoCs are predicted to become the primary communication paradigm for integrated circuits with increasingly large number of cores, such as multiprocessor systems-on-chip (MPSoCs) and chip multiprocessors (CMPs). The benefits of the NoC based SoC design approach include scalability, predictability, and higher bandwidth with support for concurrent communications.

In this paper, we present a complete implementation in VHDL of the H.264 video decoder algorithm, which we validate on two different FPGA chips. The novelty of our implementation is that the communication between the decoder modules is done using a network-on-chip. This makes our design scalable and easily integrated within larger future NoC based MPSoCs, where the same hardware platform would host other algorithms or applications such as compression, filtering, etc. To the best of our knowledge, our implementation of the H.264 decoder algorithm is the first such implementation validated on real FPGA hardware that is fully disclosed and made publicly available.

II. Related Work

Prior work on FPGA based implementations of the H.264 video decoder can be classified into three categories: 1) studies that target FPGA implementations, but stop at the simulation level (i.e., synthesis, place, and route steps performed using CAD tools such as Quartus II or Vivado), without actual validation on real hardware, 2) studies that do implement and validate portions of the H.264 decoder algorithm on FPGA hardware, and 3) simulation level only studies of NoC based H.264 decoder solutions.

Due to lack of space we do not review here studies in the first category. Examples of prior studies in the second category include [2]–[5]. The study in [2] presented implementations of IQIT, intra prediction, inter prediction, and deblocking modules. However, they did not report implementations of the entropy decoder or frame buffer. The solution in [3] provided optimization techniques for the entropy decoder and intra prediction modules. The study in [4] reported a pipelined design of the intra prediction module while the study in [5] reported a design solution for the CALVC decoder, which is a component of the entropy decoder module.

Examples of previous work in the third category, with focus on simulations of NoC based H.264 decoders, include [6]–[8]. The work in [6] studied area occupied by various NoC components on FPGAs. While [6] reported simulations on a 3x4 mesh NoC topology, the study in [7] focused on a network topology consisting of 2 star networks connected by a 3x3 NoC. A comparison between a generic NoC architecture based decoder and an NoC architecture tailored to the H.264 decoder is reported in [8]. The study in [9] reported synthesis results for an NoC based H.264 decoder targeting a Virtex 4 FPGA implementation. The work in [10] further discusses the H.264 implementation on an FPGA but reports only HDL level simulations. The study in [11] proposes a unified software and hardware architecture for video decoding where the communication infrastructure is implemented with an arrays of modified NoC routers. The processing elements are light weight processor tiles that enable software and hardware implementations to coexist, while a programmable interconnect enables dynamic interconnection of the tiles. While they reported an FPGA prototype, the source codes are not publicly available.

III. The H.264 Video Decoder Algorithm

The top level block diagram of the H.264 video decoder algorithm is shown in Fig.1. The input to the decoding algorithm consists of network abstraction layer (NAL) units, which can be organized either as packets or as a stream of bytes [12]. The primary modules of the H.264 video decoder include: entropy decoder, inverse quantization and inverse transform (IQIT), intra prediction, inter prediction, and deblocking filter. Due to space limitations, we cannot describe the functionality of each of these modules in this paper. For a complete description and details please see references [12]–[15]. Noteworthy is that for the entropy decoder we use the context adaptive variable length coding (CAVLC) approach [16].

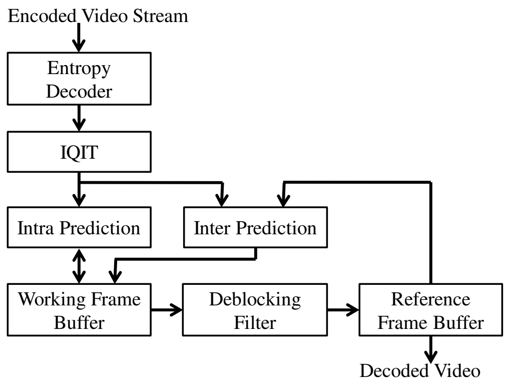

Fig. 1. The top level block diagram of the H.264 video decoder algorithm.

IV. Partitioning of the H.264 Algorithm

Before mapping the H.264 algorithm to a network-on-chip, the algorithm must first be partitioned into several components. Then, these components will be implemented as individual processing elements (PEs) attached to the routers of a regular mesh NoC architecture. Note that there may not be a one-to-one correspondence between the blocks in the diagram from Fig. 1 and these partitions, as it will become clear below. In this paper, we use a partitioning into seven primary PEs. A separate PE is dedicated to each of these seven functions: IQIT, intra prediction, sub-pixel luma motion compensation (i.e., intra prediction), sub-pixel chroma motion compensation, frame buffers control and integer motion compensation, deblocking filter, and display driver (i.e., controller). In addition, we dedicate one more PE for NAL parsing and entropy decoding. Brief descriptions of these PEs are given below.

1. NAL Parsing and Entropy Decoding PE: The task of this PE is to parse the incoming NAL stream and perform necessary entropy decoding functions on the data. These 2 functions are grouped together because the entropy decoder is only required by the NAL parser. This PE interacts with two other PEs during normal operation. It sends transform domain residuals to the IQIT PE, which then processes them and forwards the results to the buffer node. The parser PE also sends information directly to the buffer PE. This information includes prediction modes, parameters for those prediction modes, and the coordinates of the macroblock to those predictions should be performed on. Additionally, the parser PE also sends commands to start a new frame and to start the video sequence.

2. IQIT PE: This PE receives transform domain residuals data from the parser PE. It then performs the inverse quantization and inverse transform procedures on the received data. After the IQIT process is completed, the residuals are sent to the buffer PE to be added to the prediction results.

3. Intra Prediction PE: The intra prediction PE processes one block of up to 16x16 pixels at a time. The reference pixels for this prediction are provided by the buffer PE, while the parameters for this prediction are provided by the parser PE, but are routed through the buffer node before arriving at the intra prediction PE. Because of this, the intra prediction PE only communicates directly with the buffer. Therefore, when the manual mapping of these PEs to the NoC will be performed (discussed later), these two PEs should be assigned to routers that are close to each other.

4. Deblocking Filter PE: The deblocking filter PE is used right before displaying the completed frame. This PE receives data of pixels near a macroblock boundary from the buffer PE, it performs the deblocking procedure on them, and then, it sends the results back to the buffer PE.

5. Luma Motion Compensation PE: The luma motion compensation algorithm is divided between two processing elements. Integer luma motion compensation occurs on the same PE as the reference buffer because this portion of inter prediction is bound exclusively by frame buffer access time. Sub-pixel motion is a separate PE because it is computationally intensive. In addition, it has good potential for parallelization, which is easier done if this procedure is separated from other computations. Sub-pixel motion compensation uses up to two successive six tap FIR filters to interpolate pixel values. Additionally, a third two point FIR filter may be used when quarter pixel accuracy is required. The luma motion compensation PE performs interpolation for eight luma samples at a time. By interpolating eight samples at a time, the luma motion compensation PE is able to match the filter output to the throughput of the network. Although some profiles of the H.264 standard use multiple reference frames for inter prediction, our implementation uses only a single reference frame for simplicity.

6. Chroma Motion Compensation PE: The chroma motion compensation PE implements the sub-pixel motion compensation algorithm for chroma samples. Similarly to the luma motion compensation PE, this PE also performs interpolation for eight samples at a time. In this case, the eight samples are comprised of 2x2 blocks, one for each chroma channel.

By implementing the chroma and luma motion compensation algorithms on different PEs allows for both of these algorithms to run in parallel, and thus, to take advantage of the parallel communication provided by the NoC.

7. Buffer Control PE: The buffer control PE controls access to both the current frame buffer and the reference frame buffer. This PE receives parameters from the parser PE, which trigger intra prediction, inter prediction, deblocking, and display events. This PE also receives residuals from the IQIT PE that are added to the working frame buffer at the specified location. When a packet containing a command to perform a prediction action is received, the buffer PE packages up any relevant information for that prediction and sends it to the respective node or nodes. Similarly, if residuals are received from the IQIT PE, the residuals are added to the working frame buffer at the specified location. Because the parser can actually overwhelm the rest of the network during certain algorithms, the buffer PE also controls the rate at which the parser PE sends commands using one flit acknowledgment packets.

8. Display Driver PE: The display driver PE is responsible for reading decoded frames from the decoded frame buffer and for converting them into signals to control an external display. The display driver PE receives eight bit LCbCr pixels and converts these values into six bit RGB values. The RGB data is stored in the RAM local to the display PE and used to drive the hardware VGA driver.

V. Mapping to a Regular Mesh Network-on-Chip

A. *3x3 Network-on-Chip*

In the final actual VHDL description of the top level design entity, we must specify the location of each of the PEs inside the regular mesh NoC architecture. Because we use eight PEs in total, initially, the NoC architecture is a 3x3 regular mesh topology. This NoC is small enough to fit on the target FPGA chip for this implementation, which is a Stratix IV FPGA, available on the DE4 development board.

When we specify the location of the PEs inside the mesh topology (i.e., specify the NoC router to which a particular PE is assigned), we effectively solve what is commonly called a *network-on-chip mapping problem*. Thus, each of the PEs described in the previous section are mapped to the routers of the NoC. Because solving the NoC mapping problem is not the main focus of this paper, we use manual mapping, which is done intelligently based on information about the internal structure of the decoding algorithm. Noteworthy, when mapping the PEs onto the NoC, it is crucial to place PEs which share large amounts of communication close to each other. This helps to reduce the number of routers a packet must travel between source and destination through the NoC. In addition, based on our study of the decoding algorithm, we noticed that the Entropy Decoding PE sends most of its packets to the IQIT. Also, the Parser PE communicates almost exclusively with the IQIT and frame buffer control PEs. Hence, these PEs are placed as close to each other as possible. In addition, based on the partitioning used in our design, we know that both inter and intra prediction should be close to the reference frame controller. The deblocking filter and both prediction nodes are expected to communicate extensively with the reference frame controller. After taking into consideration the above observations, we arrived to the manual mapping solution shown in Fig. 2.

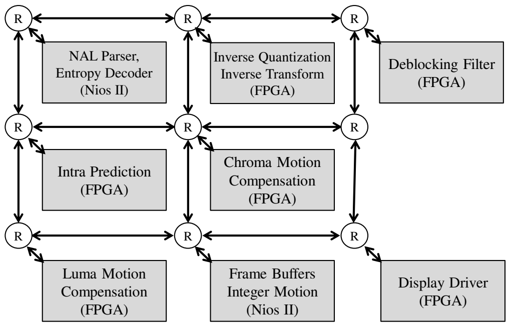
R
NAL Parser, Entropy Decoder (Nios II)
Inverse Quantization Inverse Transform (FPGA)
Deblocking Filter (FPGA)
Intra Prediction (FPGA)
Chroma Motion Compensation (FPGA)
Luma Motion Compensation (FPGA)
Frame Buffers Integer Motion (Nios II)
Display Driver (FPGA)

Fig. 2. The proposed mapping of the H.264 decoder PEs to a 3x3 regular mesh NoC. This mapping was derived manually.

Our VHDL implementation is modular and follows the architecture illustrated in Fig. 2. We have developed VHDL descriptions for each of the PEs shown in this diagram. The PEs implemented on Nios II soft cores execute portions of the decoding algorithm written in C and compiled using the tool chain available from Altera. The Verilog description of the network-on-chip was generated with the CONNECT tool from Carnegie Mellon University [17] and integrated within our project. The entire design was synthesized, placed, and routed with the Quartus II 16.1 Standard [18]. The design was tested and verified on the DE4 development board [19], which uses Altera's Stratix IV GX FPGA chip [20]. A summary of the resource utilization report is given in Table I.

TABLE I
Resource utilization of the 3x3 NoC based H.264 decoder.

Item	Report
FPGA Device Family	Stratix IV GX
Device	EP4SGX230C2
Logic Utilization	135,953 / 182,400 (75 %)
Total Combinational Functions	87002
Total Registers	65437
Dedicated Logic Registers	64,161 / 182.400 (35 %)
Total Pins	292 / 888 (33 %)
Total Block Memory Bits	1,886,567 / 14,625,792 (13 %)
DSP Block 18-bit Elements	224 / 1,288 (17 %)
Total PLLs	3 / 8 (38 %)

B. *2x2 Network-on-Chip*

To study the scalability of our design, we also implemented a scaled down version of the decoder to target an SoC style FPGA from the Cyclone V SoC family, available on the DE1-SoC development board. This is a smaller FPGA chip. However, the SoC FPGA has a built-in hard processor system (HPS), which contains two ARM cores that will be used to implement the parser and buffer PEs. Three major changes were made to the initial design in order to be able to fit it on the smaller FPGA. First, the built-in ARM cores instead of Nios II soft cores are used to implement the buffer and parser PEs. Second, the intra prediction functionality was moved to the buffer PE, which is performed in software (i.e., executed on the HPS). Finally, the NoC topology was changed to a 2x2 topology. Because of that, routers were changed to include more ports in order to be able to accommodate all the PEs in the design.

The intra prediction was moved into software (i.e., added into the buffer PE) because, based on profiling results, it is not utilized as much as the inter prediction, but it still uses a large amount of area on the FPGA. The choice to reduce the number of routers, while increasing the number of ports per router was made because it saves on the number of resources greatly and was not expected to greatly degrade performance. Additionally, the software which controls the network interface needed to be modified to support the IO interface of the target processor. The final architecture for this design implementation is shown in Fig. 3, which also shows the manual mapping, which was done under the same considerations as before.

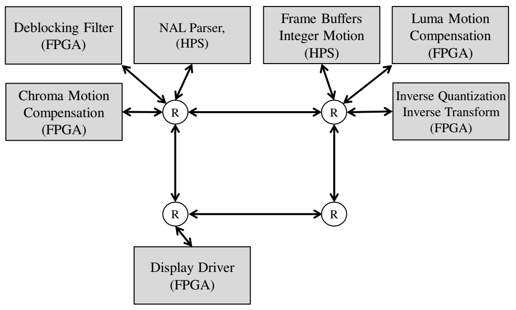
Deblocking Filter (FPGA)
NAL Parser, (HPS)
Frame Buffers Integer Motion (HPS)
Luma Motion Compensation (FPGA)
Chroma Motion Compensation (FPGA)
R
R
Inverse Quantization Inverse Transform (FPGA)
R
R
Display Driver (FPGA)

Fig. 3. Mapping of the H.264 decoder PEs to a 2x2 regular mesh NoC.

The scaled down version of the video decoder described in this section was also implemented using the Quartus II Prime tool. Testing and verification was done on the DE1-SoC development board [21], which uses Altera's Cyclone V SoC FPGA chip [22]. The resource utilization report is given in Table II.

TABLE II
RESOURCE UTILIZATION OF THE 2X2 NOC BASED H.264 DECODER.

Item	**Report**
FPGA Device Family	Cyclone V
Device	5CSEMA5F31C6
Logic Utilization	24,708 / 32,070 (77 %)
Total Registers	30978
Dedicated Logic Registers	31,102
Total Pins	159 / 457 (35 %)
Total Block Memory Bits	1,152,000/4,065,280 (28 %)
Total DSP Blocks	87 / 87 (100 %)
Total PLLs	1 / 6 (17 %)

VI. Experimental Results

In this section, we present the results of the performance testing and profiling performed on both versions of the NoC based H.264 decoder. To perform this profiling several counters were added to the Nios II PEs. These counters are read by the software running on the Nios II cores at a variety of locations in order to determine what the limiting factors of system performance are. Additionally, the timer is also used to determine the overall system performance in order to be able to compare against an all-software decoder implementation.

A. Test Videos

For testing purposes, we use five test videos (*akiyo*, *foreman*, *highway*, *hall*, and *paris*) from the online repository of YUV encoded video files available at [23]. The first three tests use the QCIF format with a resolution of 176x144. The remaining two tests use the CIF format, which has twice the horizontal and vertical resolution of the QCIF videos. An important note here is that, although the video decoder itself can decode CIF as well as higher resolution videos, the display PE only contains enough RAM to display 320x200 videos. Therefore, for the CIF videos, the entire video is decoded, but, only the top left 320x200 pixels are displayed from each frame. Each of the five test videos are encoded using the JM reference encoder available at [24]. The encoding settings use a modified baseline profile that uses a single reference image and a periodic intra prediction update to avoid accumulated errors in the output video stream. Selected frames of these test videos being decoded by each of our two implementations are shown in Fig. 4 and Fig. 5.

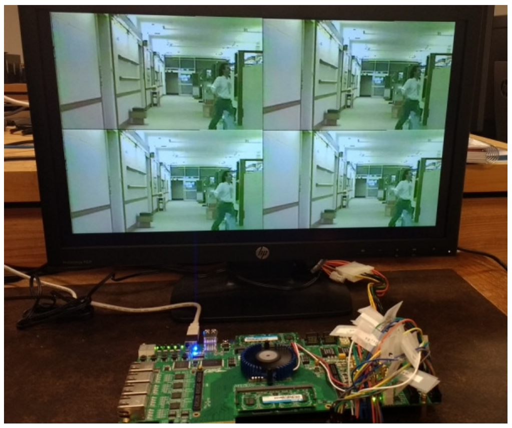

Fig. 4. Operation of the 3x3 implementation on the *hall* video.

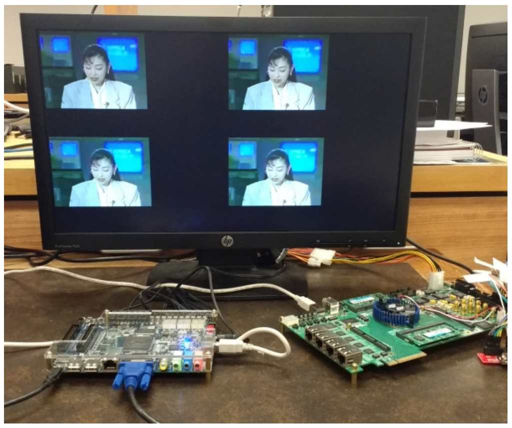

Fig. 5. Operation of the 2x2 implementation on the *akiyo* video.

B. Buffer PE Profiling

Because the buffer PE controls access to resources used by nearly every algorithm in the system, the profiling measurements are taken at this PE. The block diagram that illustrates the profiling of different components of the system at the buffer PE level is shown in Fig. 6.

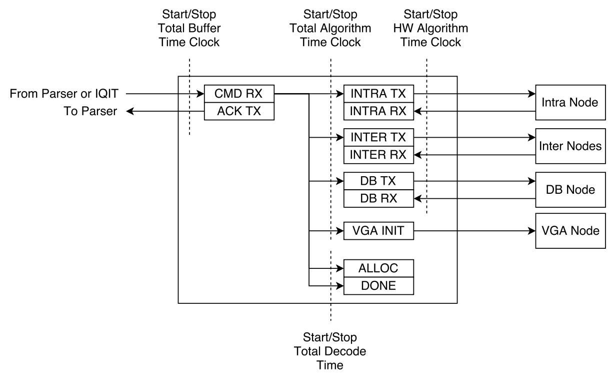

Fig. 6. Simplified diagram of the buffer PE software to illustrate the timer start/stop positions used for profiling.

A total of nine counters are included and monitored for profiling purposes. The first counter starts when the buffer PE receives an allocate frame command and stops when the buffer PE receives a special packet indicating the video stream is done. The purpose of this counter is to keep track of the total time it takes to decode the video sequence. Counters for deblocking, intra prediction, and inter prediction are included in order to keep track of the total time taken to perform each of these algorithms. An additional counter for each of these three algorithms is included to keep track of the amount of time the buffer PE idles while waiting for a response from the PE associated with each of these algorithms. A counter is also used to keep track of the duration of a write to the display PE. Another counter is used to determine the total time the buffer PE spends idling after completing a command before it receives another one. The IQIT algorithm is not profiled because this algorithm is essentially done by the time it reaches the buffer PE.

The results of the profiling for both the 3x3 and 2x2 decoder implementations are summarized in Tables III and IV. All times in these tables are in units of seconds The average distribution for each decoder is shown in Fig. 7 for the 3x3 decoder and in Fig. 8 for the 2x2 decoder.

TABLE III
PROFILING RESULTS OF THE 3X3 NOC DECODER.

Video	**akiyo**	**highway**	**foreman**	**paris**	**hall**
Format	qcif	qcif	qcif	cif	cif
Frames	300	2000	300	1060	300
Total Decode Time	37.18	310.00	55.07	579.68	168.65
Total Intra Time	1.61	11.46	1.96	23.68	6.08
Total Inter Time	8.08	83.26	16.36	135.85	38.65
Total Deblock Time	16.14	107.62	16.14	242.31	68.26
Total Display Time	3.99	26.60	3.99	35.95	10.13
Intra Idle Time	0.54	3.80	0.65	7.92	2.02
Inter Idle Time	0.47	12.97	3.25	11.39	3.52
Deblock Idle Time	2.57	17.12	2.57	38.51	10.85
Command Wait Time	6.23	68.75	14.14	121.27	38.63

C. Discussion of Profiling Results

The profiling results indicate that a large portion of total time is spent performing inter prediction and deblocking

TABLE IV
PROFILING RESULTS OF THE 2X2 NOC DECODER.

Video	**akiyo**	**highway**	**foreman**	**paris**	**hall**
Format	qcif	qcif	qcif	cif	cif
Frames	300	2000	300	1060	300
Total Decode Time	43.05	430.34	83.79	615.21	184.04
Total Intra Time	0.04	0.27	0.04	0.60	0.16
Total Inter Time	6.26	157.34	39.00	143.15	43.96
Total Deblock Time	13.30	88.71	13.31	199.47	56.19
Total Display Time	16.45	109.63	16.45	147.52	41.55
Intra Idle Time	0.00	0.00	0.00	0.00	0.00
Inter Idle Time	1.65	46.00	11.56	39.88	12.36
Deblock Idle Time	8.95	59.69	8.95	134.20	37.80
Command Wait Time	6.28	67.29	13.48	112.80	38.14

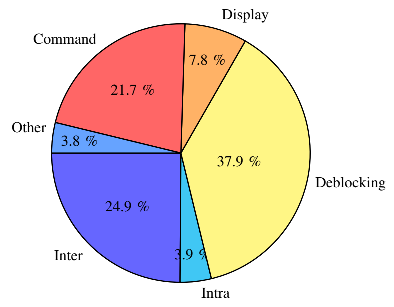

Fig. 7. Average time spent in each section of the buffer PE code for the 3x3 NoC based decoder.

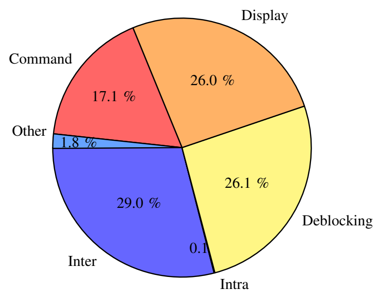

Fig. 8. Average time spent in each section of the buffer PE code for the 2x2 NoC based decoder.

relative to time the buffer PE idles waiting for these algorithms PEs to respond. This indicates that the buffer PE is currently incapable of fully utilizing these PEs. Because the code on the buffer PE for dispatching either of these algorithms consists almost entirely of reading from memory and writing to the NoC through the NoC interface, architectural changes for improving performance in the future could focus on several aspects in order to improve the design. First, one could improve the total bandwidth to the memory if possible. Second, one should reduce the amount of activity or interaction required by the buffer PE to perform NoC reads and writes. Also, given that the buffer PE is relatively simple, one could implement its functionality completely in hardware. Finally, the parser PE could be a good target for improvement because the command wait time takes up a significant amount of the buffer PE time.

VII. PERFORMANCE COMPARISONS

In this section, we conduct performance comparisons between our FPGA implementations and an open source all-software decoder implemented in C [25]. This decoder was executed separately on both the Nios II soft core and the HPS core. To do that, the decoder was modified to use the VGA display PE for video output. Additionally, the decoder was modified to use the available hardware timer to measure the total decoding time. Because reading from the counters has a performance impact, all of the profiling counters were removed from the NoC based decoders, except for the total decoding time timer. The test results are shown in Table V, where all reported numbers are in units of frames per second.

TABLE V
COMPARISON OF THE NOC BASED DECODERS WITH THE OPEN SOURCE SOFTWARE BASED DECODER, WHICH IS EXECUTED ON NIOS II OR HPS.

	3x3 Decoder	2x2 Decoder	Nios II SW	HPS SW
akiyo (qcif)	8.26	7.12	4.86	11.57
highway (qcif)	6.74	4.73	3.55	11.03
foreman (qcif)	5.76	3.64	2.81	10.79
paris (cif)	1.96	1.76	0.82	4.23
hall (cif)	1.91	1.66	1.02	4.22
average fps (cif)	1.95	1.74	0.85	4.23
average fps (qcif)	6.75	4.75	3.56	11.06

A. Discussion of Performance Comparisons

As expected, the NoC based implementation outperforms the Nios II soft core processor running the full software decoder. An unexpected result is the performance of the software decoder running on the ARM core. One important difference between the software decoders and the NoC based decoders is that the software decoders do not implement a deblocking filter. However, even after accounting for this, a large performance discrepancy exists. Another noteworthy point is that the 2x2 decoder is slower than the 3x3 decoder, despite the fact that the software decoder is over three times faster on the ARM core compared to the Nios II soft core, and intra prediction is not a heavily utilized function. Based on the profiling results, this appears to stem from the fact that the communication between the HPS and the FPGA on the SoC style FPGA is slower than the the communication between the Nios II soft cores and the rest of the FPGA on the large scale design. Evidence of this can be seen in the relative time spent writing to the display PE in the case of each decoder. The 2x2 decoder spends about 26% of its time writing to the display PE, while the 3x3 decoder only spends about 8% of its time for the same activity. This indicates a large difference in the communication overhead between the two designs. Based on this information, efforts seeking to improve the NoC decoder implementations against the ARM core should focus on improving the processor to FPGA communication or removing it altogether by creating a hardware only implementation of the buffer PE.

VIII. CONCLUSION

We presented the design of an H.264 video decoder prototyped on the Altera's Stratix IV GX and Cyclone V SoC FPGA chips. Our main contributions can be summarized as follows: 1) The communication between the decoder modules is done via a network-on-chip, which makes our implementation suitable for integration into larger MPSoC solutions, 2) We provide a complete system level design solution that includes a partitioning of the decoding algorithm into eight processing elements and manual mappings of these PEs onto 3x3 and 2x2 NoC topologies, and 3) The entire project implementation will be made publicly available.

ACKNOWLEDGMENT

This work was supported by the Department of Electrical and Computer Engr. at Marquette University.

REFERENCES

[1] R. Marculescu, U.Y. Ogras, L.-S. Peh, N.E. Jerger, and Y. Hoskote, "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives," *IEEE TCAD*, 2009.

[2] V.S. Rosa et al., "FPGA prototyping strategy for a H.264/AVC video decoder," *IEEE/IFIP International Workshop on Rapid System Prototyping*, 2007.

[3] J. Ru, Y. Yang, and Y. Yang, "Design of H.264 video decoding IP core on FPGA," *Int. Conf. on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*, 2014.

[4] L.-G. Wu, D.-L. Zhang, G.-M. Du, Y.-K. Song, and M.-L. Gao, "A 4x4 pipelined intra frame decoder for H.264," *Int. Conf. on Anti-counterfeiting, Security, and Identification in Communication*, 2009.

[5] T.G. George and N. Malmurugan, "A new fast architecture for HD H.264 CAVLCm multi-syntax decoder and its FPGA implementation," *Int. Conf. on Computational Intelligence and Multimedia Appl.*, 2007.

[6] A. Agarwal, C.-D. Iskander, H. Kalva, and R. Shankar, "System-level modeling of a NoC-based H.264 decoder," *IEEE Systems Conf.*, 2008.

[7] J.-Y. Chang, W.-J. Kim, Y.-H. Bae, M.-Y. Lee, J.-Y. Kim, and H.-J. Cho, "Star-Mesh NoC based multi-channel H.264 decoder design," *Int. SoC Design Conference*, 2008.

[8] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "H. 264 HDTV decoder using application-specific networks-on-chip," *Int. Conf. on Multimedia and Expo (ICME)*, 2005.

[9] A. Luczak, P. Garstecki, O. Stankiewicz, and M. Stepniewska, "Network-on-chip based architecture of H.264 video decoder," *Int. Conf. on Signals and Electronic Systems*, 2008.

[10] M. Stepniewska, Advanced video codecs implementation using Networks-on-Chip in FPGA devices, Poznan University, 2012.

[11] A. Rao, S.K. Nandy, H. Nikolov, and E.F. Deprettere, "USHA: unified software and hardware architecture for video decoding," *IEEE Symp. on Application Specific Processors (SASP)*, 2011.

[12] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 3, no. 7, pp. 560-576, July 2003.

[13] H. Kalva, "The H.264 video coding standard," *IEEE MultiMedia*, vol. 13, no. 4, pp. 86-90, Oct.-Dec. 2006.

[14] S. Nargundmath and A. Nandibewoor, "Entropy coding of H.264/AVC using Exp-Golomb coding and CAVLC coding," *Int. Conf. on Advanced Nanomaterials and Emerging Engineering Technologies*, 2013.

[15] I. Richardson, "H.264/AVC context adaptive variable length coding," *White Paper*, Vcodex, 2002.

[16] International Telecommunication Union, Advanced video coding for generic audiovisual services, 2014. [Online]. Available: https://www.itu.int/rec/T-REC-H.264-201402-S/en.

[17] M.K. Papamichael and J.C. Hoe, "The CONNECT network-on-chip IP generator," *IEEE Computer*, vol. 48, no. 12, Dec. 2015.

[18] Quartus II Software, 2016. [Online]. Available: http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html.

[19] Altera DE4 Board, Terasic, 2017. [Online]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=138&No=501.

[20] Stratix IV FPGAs, 2017. [Online]. Available: https://www.altera.com/products/fpga/stratix-series/stratix-iv/overview.html.

[21] DE1-SoC Board, Terasic, 2017. [Online]. Available: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836.

[22] Cyclone V SoC FPGAs, 2017. [Online]. Available: https://www.altera.com/products/soc/portfolio/cyclone-v-soc/overview.html.

[23] M. Reisslein, L.J. Karam, P. Seeling, F.H.P. Fitzek, and T.K. Madsen, YUV Video Sequences, 2017. http://trace.eas.asu.edu/yuv/.

[24] Karsten Suehring, H.264/AVC Software Coordination, http://iphome.hhi.de/suehring/tml/.

[25] Martin Feidler, Implementation of a basic H.264/AVC decoder, 2017. https://keyj.emphy.de/projects/studies/.