Transactions Briefs

Interframe Bus Encoding Technique and Architecture for MPEG-4 AVC/H.264 Video Compression

Asral Bahari, Tughrul Arslan, and Ahmet T. Erdogan

***Abstract*—In this paper, we propose an implementation of a data encoder to reduce the switched capacitance on a system bus. Our technique focuses on transferring raw video data for multiple reference frames between off- and on-chip memories in an MPEG-4 AVC/H.264 encoder. This technique is based on entropy coding to minimize bus transition. Existing techniques exploit the correlation between neighboring pixels. In our proposed technique, we exploit pixel correlation between two consecutive frames. Our method achieves a 58% power saving compared to an unencoded bus when transferring pixels on a 32-b off-chip bus with a 15-pF capacitance per wire.**

***Index Terms*—Data buses, encoding, integrated circuit design, video coding.**

I. Introduction

Video application has become increasingly popular in today's wireless communications. However, video processing is computing intensive and dissipates a significant amount of power. For MPEG-4 video compression, raw video data (pixels) dominate data transfer [1]. During the compression of 5 min of video (CIF@15 fps), at least 4500 frames are transferred from the memory to the video compressor. These values increase for higher frame rates and frame resolutions.

This high data transfer translates into high power dissipation on the memory-processor busses. This is severe for systems with off-chip memory where the bus load is several orders of magnitude higher than the on-chip bus with a typical value of around 15 pF on each bus wire [2]. It has been reported that the off-chip bus consumes 10%–80% of overall power [3].

In this paper, we present a data encoding technique to minimize power dissipation during multiple-frame transfer between the MPEG-4 AVC/H.264 video compressor and the external reference frame memory using an off-chip system bus, as shown in Fig. 1. Power reduction is achieved by utilizing bus encoding to reduce switching activity on the bus. Bus encoding transforms the original data such that two consecutive encoded data have lower switching activity than the unencoded one.

References [4]–[6] address the implementation of the bus encoding on address busses. The proposed methods exploit highly correlated bus addresses to reduce switching activity. Compared with address busses, data busses show more random characteristics. Bus invert [7], codebook [8], and exact algorithm [9] were proposed for this type of data.

Existing techniques exploit the correlation between neighboring pixels for video data. However, pixel correlation between frames has not been fully exploited to reduce bus transition in the literature. In [10], we have proposed an interframe bus encoding technique where we utilized the pixel correlation between two consecutive frames without full system consideration. In this paper, we extend the technique to a complete H.264 system.

The rest of this paper is organized as follows. Section II reviews the existing intraframe techniques for bus encoding. Section III discusses our approach to reducing the transition activity during memory data transfer. Section IV discusses the proposed implementation of the interframe bus encoding technique for the H.264 system. This is followed by the results and performance benchmarking of our method in Section V. Finally, Section VI concludes the paper.

Manuscript received June 23, 2008; revised November 19, 2008. First published May 19, 2009; current version published April 23, 2010.

A. Bahari is with the School of Microelectronic Engineering, University Malaysia Perlis, Arau 02600, Perlis, Malaysia (e-mail: asral@unimap.edu.my).

T. Arslan and A. T. Erdogan are with the School of Engineering and Electronics, University of Edinburgh, EH9 3JL Edinburgh, U.K. (e-mail: tughrul.arslan@ed.ac.uk; ahmet.erdogan@ed.ac.uk).

Digital Object Identifier 10.1109/TVLSI.2009.2015324

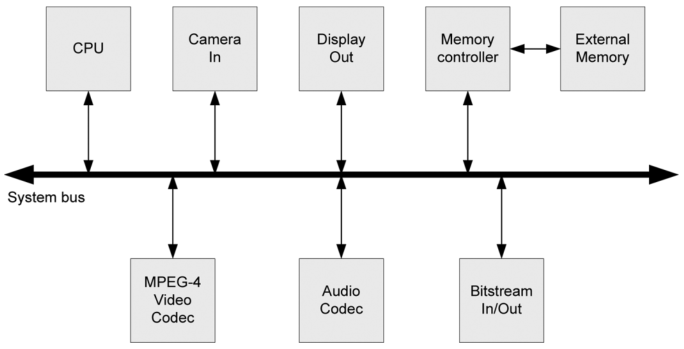


Fig. 1. Typical video communication system.

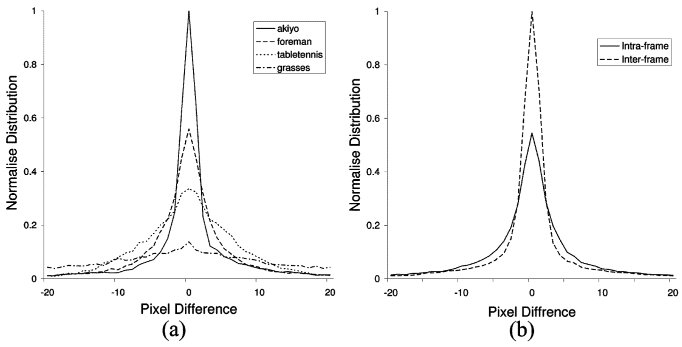


Fig. 2. Pixel decorrelation using (a) adjacent pixel. (b) Interframe versus intraframe decorrelation.

II. Intraframe Decorrelation

The technique discussed in this paper is based on the combination of difference-base-mapped and value-base-mapped (dbm–vbm) techniques, as discussed in [11]. We adopt this method because it allows us to exploit the pixel correlations widely available in video data.

Fig. 2(a) shows the distribution of two adjacent pixels' difference. Four different quarter common intermediate format (QCIF) video sequences (Akiyo, Foreman, Table Tennis, and Grasses), which represent various motion types fromlow to high, are evaluated. Five frames from each sequence are evaluated, which consists of 190 080 pixels. The graph shows that, for highly correlated data, the difference between two consecutive pixels with a smaller magnitude has higher probability than that of consecutive pixels with a larger magnitude. DBM–VBM utilizes this characteristic to minimize the bus transition.

Fig. 3 shows the block diagram describing the dbm–vbm operation. It consists of a decorrelator (dbm) and entropy coder (vbm). The

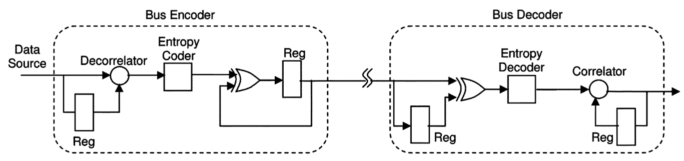


Fig. 3. DBM–VBM bus encoder and decoder.

dbm–vbm technique is summarized as follows. First, two adjacent pixels (intraframe) are decorrelated using dbm. DBM calculates the relative difference between the two pixels. VBM maps the values to patterns that have different weights (i.e., total number of 1s). To reduce the overall transition, it maps the low-magnitude value to a pattern that has the fewest 1s, whereas higher magnitude values are mapped to patterns that have more 1s. At the output, the XOR translates 1s as transition and 0s as transitionless.

The average number of transitions for the dbm–vbm method depends on its source word, i.e., the decorrelator output. The more the graph is skewed toward zero, the more patterns are assigned with less 1 s. Thus, one way to improve the transition reduction is by improving the decorrelator.

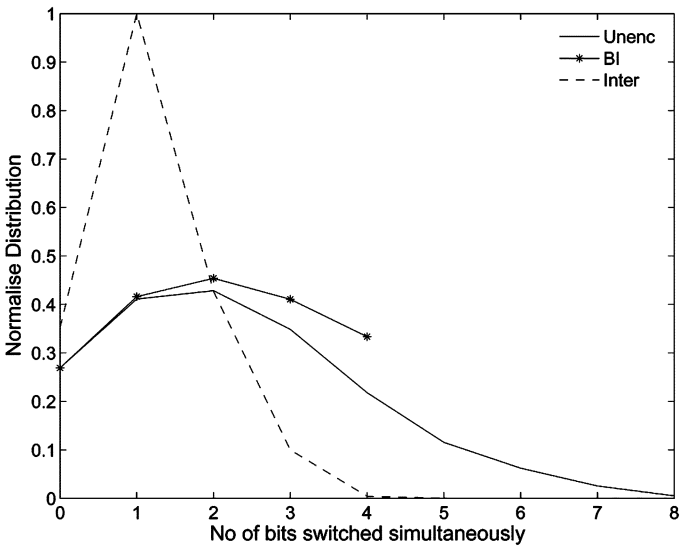


Fig. 4. Simultaneous bit-switching comparison between different bus encoding methods: Unencoded bus, bus invert, and the proposed interframe method.

III. Interframe Decorrelation

Video sequences consist of both spatial and temporal redundancy. The existing bus encoding techniques utilize spatial redundancy within frames. However, the temporal redundancy is not fully exploited to reduce bus transition.

In [10], we have proposed decorrelating the pixels using two consecutive frames (interframe). This method is based on the observation that two consecutive frames are highly correlated. Often, the background of a scene is stationary. Furthermore, for a moving object, the differences between successive frames are very small. Fig. 2(b) shows the pixel decorrelation using the intraframe and interframe methods for five Foreman sequences. The figure shows that decorrelating the pixels using interframe improves the graph skewness toward zero. This will translate to higher transition saving since more patterns will be assigned with less 1s.

The results in [10] show that the interframe method provides higher transition reduction compared with both bus invert and intraframe implementations. On average, our method reduces up to 65% of the transition over an unencoded bus. This is equivalent to 1.5 and 2.6 times more transition saving over intraframe and clustered bus invert, respectively.

Fig. 4 shows the normalized distribution for the total number of bits that are switched simultaneously when transferring the pixels. The higher the number of bits that are switched simultaneously, the higher the peak power of the bus. As shown in the figure, interframe bus encoding results in a much lower number of bits switching simultaneously compared with the bus invert method. This shows that not only does it reduce the off-chip average power but the interframe method also reduces the peak power of the bus.

IV. Interframe Bus Encoding Implementation Into H.264 System

As shown in Fig. 1, the external memory is connected with the on-chip video compressor through an off-chip bus. In typical implementations, the same bus is used to send or receive the data from the external memory. In [10], we assumed that the pixel from the two frames is transmitted in two different busses. In order to realize the interframe bus encoding into hardware implementation, modification has to be made to this setup. This is to take into account the limited availability of the off-chip bus used in the actual system.

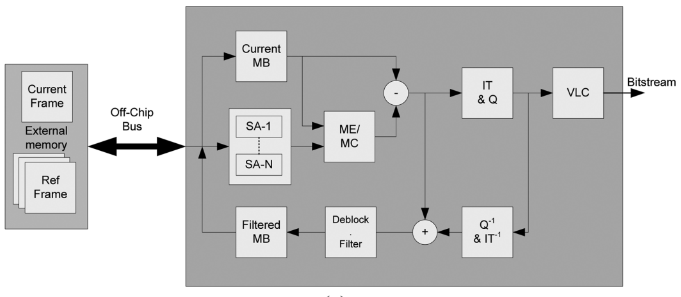
In this section, the operation of the H.264 video compression system is first described to illustrate the interaction between the video compressor and the off-chip memory. Then, the proposed architecture for implementing the interframe bus encoding for the H.264 system is discussed in depth to minimize the off-chip bus power.

A. H.264 System

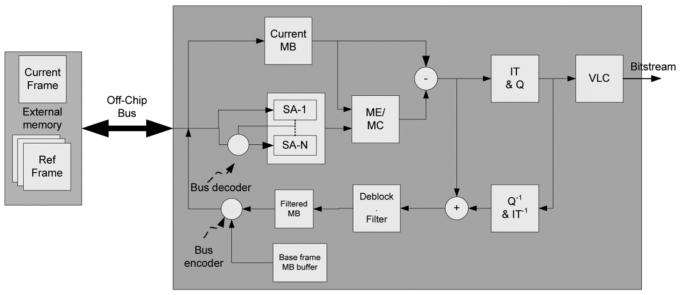
Fig. 5(a) shows the main functional block of the H.264 encoder and the interaction among its main modules. The encoder consists of motion estimation (ME), motion compensation (MC), integer transform (IT), quantizer (Q), variable length coder (VLC), and deblocking filter (DFIR). In addition, the system requires search-area (SA) buffers to store SA pixels temporarily from the reference frame memory and filtered macroblock (MB) buffers to keep the intermediate data during the encoding process. The main function of the encoder is to encode the current MB pixels into a compact bitstream so that it can be decoded by the decoder.

In H.264, the encoder first loads the SA and the current MB pixels from the external frame memory through the off-chip bus. Then, the current MB is predicted using ME. This process is repeated until all SAs from multiple reference frames are evaluated. The predicted MB that results in the lowest cost is selected. The residue of the predicted MB is then calculated by subtracting the current MB and the predicted MB before it is transformed by the IT. The transform coefficient and motion vector are coded using the VLC to generate a compact bitstream. The encoder also reconstructs the current MB using information from transform coefficients. The reconstructed MB is filtered by the DFIR before storing the MB in the external reference frame memory for future frame prediction.

The system requires 2000 clock cycles to process one MB by dividing the encoder operation into three pipeline stages: ME/MC, IT/Q/IQ/IT, and DFIR/VLC. For QCIF frame size at 30 frames per second, the system is set to 6 MHz to achieve real-time operation. Based on the United Microelectronics Corporation 0.13-μm CMOS technology, the system requires a total core area of 4.8 mm^2.



(a)



(b)

Fig. 5. (a) Interaction between the external reference frame memory and H.264 modules. (b) Modified H.264 system that includes an interframe bus coder.

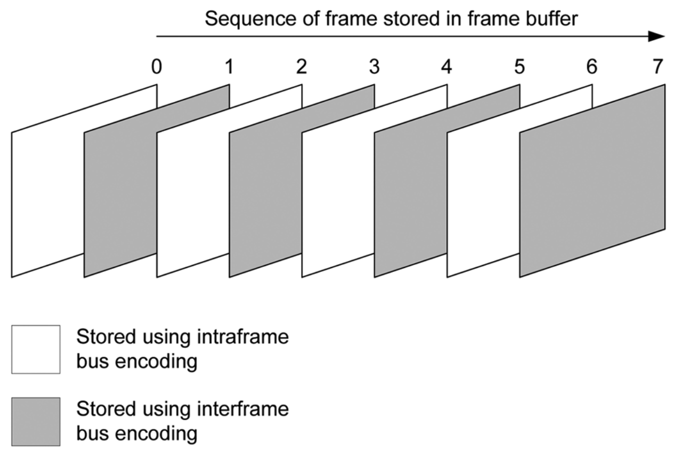

Sequence of frame stored in frame buffer
0
1
2
3
4
5
6
7
Stored using intraframe bus encoding
Stored using interframe bus encoding

Fig. 6. Reference frame arrangement of external buffer and the type of bus encoding applied to it.

B. Integration of Interframe Bus Encoding Into H.264

As shown in Fig. 5(a), an interaction between the H.264 modules and the external reference frame buffer occurs on two occasions: 1) when the pixel is stored into the external reference frame buffer after filtering the MB through the DFIR and 2) during the loading of the SA pixels from the external reference frame buffer into the SA RAM. Thus, the bus encoder and decoder should be implemented in 1) and 2), respectively.

Fig. 5(b) shows the modified H.264 system to include the proposed interframe bus encoding method. Since reference frames are accessed in series between the video processor and the external frame memory, some modification is made on the interframe bus encoding circuit proposed in [10]. To allow interframe decorrelation during bus encoding, reference frames stored in the external memory are arranged in sequence, as shown in Fig. 6. Since the interframe technique requires other frames, the bus coding is alternated between intraframe and interframe.

A base frame (BF) refers to the frame that is stored as intraframe by the bus encoder, since it does not require any other frames. A dependence frame (DF) is a frame that is stored as interframe by the bus

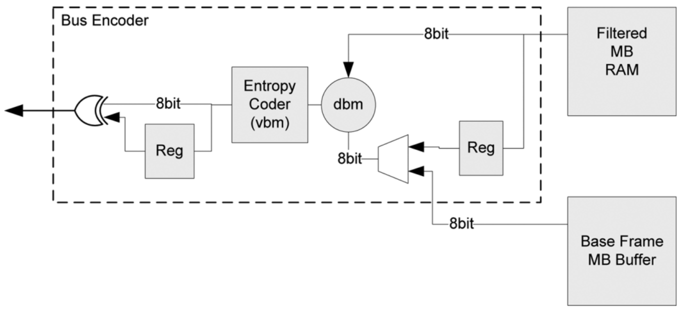


Fig. 7. Interframe–intraframe encoder for H.264 hardware.

encoder, since it requires other frames to decode the pixel values. DF is always stored after its BF for ease of decoding later on.

In order to allow interframe decorrelation, pixels from the BF have to exist for the bus encoder. The bus decoder has a similar requirement. For the bus decoder, since the SA is accessed from a multiple number of frames, the proposed solution is to load the BF first, followed by the DF. When the SA from the DF is loaded, the SA pixel from the BF is accessed as well to perform the interframe decorrelation.

In contrast to the bus decoder, since the fully filtered MB is only written into the external frame buffer, no BF is available at the bus encoder. The solution proposed for this is to store the required BF MB in a buffer. The BF MB can be stored when loading the BF SA into the SA memory. Since the DFIR operation is located in the third stage of the pipeline buffer, an additional MB buffer is required to store the last three MBs of the BF. This buffer will be used during interframe decorrelation when writing the pixels into the reference frame buffers. Using this approach, it is possible to perform interframe decorrelation at the bus encoder.

The detailed hardware implementations for the bus encoder and decoder are shown in Figs. 7 and 8, respectively. For this implementation, four pixels at a time are transferred to/from the external memory on a 32-b bus (8 b per pixel). Thus, four encoders and decoders are used to encode and decode the bus. Due to space limitations, only one encoder and one decoder are shown in Figs. 7 and 8. The architectures are able to perform either interframe or intraframe bus encoding. When intraframe is selected, the bus encoder calculates the dbm decorrelation using the filtered pixel and the previous pixel stored in the register (Reg). Similar input is used to calculate the dbm^{-1} at the bus decoder.

During the interframe bus encoding, the encoder calculates the dbm using filtered pixels and pixels stored in the buffer MB, as shown in Fig. 7. Since the buffer MB stores the corresponding BF MB, this is equivalent to decorrelating a pixel from two frames. To decode the pixel, the SA from the BF is first loaded into the SA buffer. When the SA from the DF is loaded, the loaded SA from the BF is read in parallel to correlate the SA pixel before storing it in the SA RAM. This is shown by the dotted line with an arrow in Fig. 8.

V. Results and Discussion

The design was synthesized using the UMC 0.13-μm CMOS library. Verilog-XL and Power Compiler were used to perform functional simulation and power analysis using the extracted layout data, respectively. Actual video data were used to verify the hardware and to obtain the estimated power consumption.

Tables I and II illustrate the resources required to implement the proposed interframe bus encoder and decoder in the H.264 system. For this implementation, four pixels at a time are transferred to/from the external memory on a 32-b bus. Thus, four encoders and decoders are used to encode and decode the bus. The tables show that the maximum delay and total area overhead required to implement the hardware is

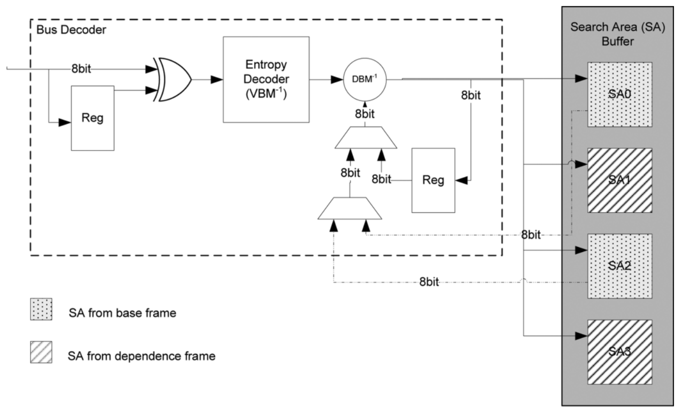


Fig. 8. Interframe–intraframe decoder for H.264 hardware.

TABLE I
BUS ENCODING AREA AND POWER OVERHEAD REQUIRED TO IMPLEMENT THE INTERFRAME–INTRAFRAME BUS CODING ON A 32-b BUS AT 6 MHz

Encoder Modules	Area (mm^2)	Power (mW)		Delay (ns)
		Interframe mode	Intraframe mode	
Logic	0.03	0.178	0.176	4.4
Memory	0.10	0.39	-	
Total	0.13	0.568	0.176	

TABLE II
BUS DECODING AREA AND POWER OVERHEAD REQUIRED TO IMPLEMENT THE INTERFRAME–INTRAFRAME BUS CODING AT 6 MHz

Decoder Modules	Area (mm^2)	Power (mW)		Delay (ns)
		Interframe mode	Intraframe mode	
Logic	0.03	0.119	0.060	4.6
Memory	-	0.352	-	
Total	0.03	0.470	0.060	

4.6 ns and 0.16 mm^2, respectively. This is equivalent to 3% of the total area in the conventional H.264 as discussed in Section IV-A.

Tables I and II also show the power evaluation result of the proposed architectures. From the table, it can be seen that the proposed encoder and decoder circuits consume 0.568 and 0.470 mW of power, respectively, during interframe bus coding mode with the memory dominating the circuit power overhead. Since the intraframe decorrelation does not require any memory access, the total circuit power during intraframe bus encoding and decoding modes is much lower at 0.176 and 0.060 mW, respectively.

Figs. 9 and 10 show the total power consumption during interframe bus encoding and decoding, respectively, as compared with an unencoded bus, bus coded using cluster bus invert, and intraframe. The total power consumption P_T is calculated as $P_T = P_{\mathrm{Circuit}} + P_{\mathrm{CL}}$, where P_{Circuit} represents the power consumption due to bus encoder or decoder circuits. P_{CL} is the total bus power consumption estimated by $P_{\mathrm{CL}} = (1/2)C_L V_{\mathrm{DD}}^2 f\alpha$, where C_L is capacitance load, V_{DD} is the operating voltage, f is the operating frequency, and α is the switching activity. The slope of the graphs in Figs. 9 and 10 is proportional to α, which is dependent on the type of bus encoding used.

For a typical off-chip wire capacitance of 15 pF, the total bus encoder power consumption during interframe mode is 3.39 mW, while it is 4.8 mW during intraframe mode. These are equivalent to 58% and 40% power savings compared with an unencoded bus. The greater power saving achieved by the interframe mode is due to the much lower transitions occurring on the busses. A smaller slope, as shown in the graphs in Figs. 9 and 10, reflects this.

Tables III and IV show the total energy consumed by the 32-b bus when accessing the external reference frame buffer. Since the loading

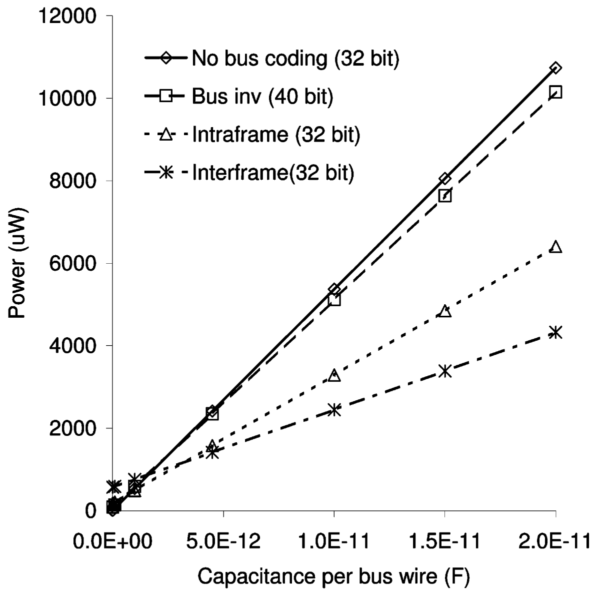


Fig. 9. Power consumption of 32-b bus at 6 MHz when transferring pixels into the external reference frame memory.

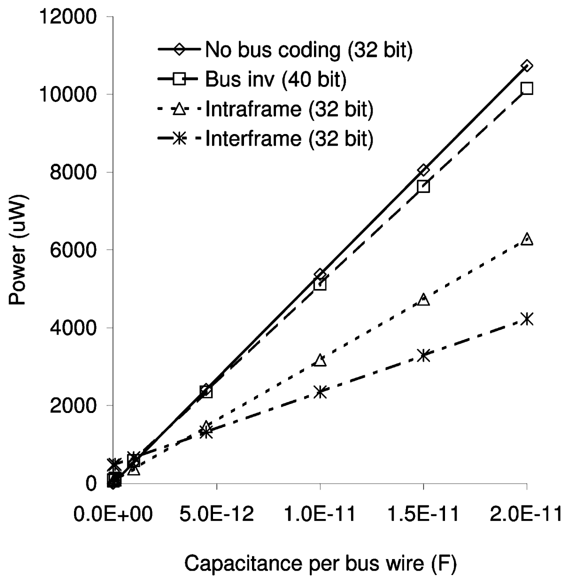


Fig. 10. Power consumption of 32-b bus at 6 MHz when loading pixels from the external reference frame memory.

TABLE III
TOTAL ENERGY CONSUMPTION ON 32-b BUS AT 6 MHz WITH 15 pF PER BUS WIRE WHEN SENDING ONE FILTERED MB TO EXTERNAL FRAME BUFFER

Bus encoder	Power (μW)	Clock	Energy (nJ)	Saving (%)
Unencoded	8053.16	64	85.90	-
Bus Inv	7633.83	64	81.43	5
Intraframe	4846.83	64	51.70	40
Interframe	3386.60	64	36.13	58

TABLE IV
TOTAL ENERGY CONSUMPTION ON 32-b BUS AT 6 MHz WITH 15 pF PER BUS WIRE WHEN RECEIVING ONE SA FROM EXTERNAL FRAME BUFFER FOR ONE REFERENCE FRAME

Bus decoder	Power (μW)	Clock	Energy (nJ)	Saving (%)
Unencoded	8053.16	128	171.80	0
Bus Inv	7633.83	128	162.85	5
Intraframe	4730.83	128	100.92	40
Interframe	3288.60	128	70.16	58

of one SA transfers more data (512 pixels) than storing one MB data into the external reference frame buffer (256 pixels), more energy is consumed during the loading of the SA pixel. In addition, the data loaded from the external reference frame buffer is propotional to the number of reference frames used during ME. From the tables, for both cases, the interframe bus encoding saves 58% of total energy as compared with an unencoded bus.

VI. SUMMARY

We have presented an interframe bus encoding technique for applications where multiple frames are transferred between off-chip memories, such as in MPEG-4 AVC/H.264 applications. The proposed interframe bus encoding technique results in a 65% transition reduction over the unencoded bus. This is equivalent to a 58% power saving compared to an unencoded bus when transmitting four pixels at a time over a 32-b bus with a 15-pF capacitance per wire.

REFERENCES

[1] C.-H. Lin, C.-M. Chen, and C.-W. Jen, “Low power design for MPEG-2 video decoder,” *IEEE Trans. Consum. Electron.*, vol. 42, no. 3, pp. 513–521, Aug. 1996.

[2] W.-C. Cheng and M. Pedram, “Chromatic encoding: A low power encoding technique for digital visual interface,” *IEEE Trans. Consum. Electron.*, vol. 50, no. 1, pp. 320–328, Feb. 2004.

[3] T. Givargis and F. Vahid, “Interface exploration for reduced power in core-based systems,” in *Proc. 11th Int. Symp. Syst. Synthesis*, 1998, pp. 117–122.

[4] H. Mehta, R. Owens, and M. Irwin, “Some issues in gray code addressing,” in *Proc. 6th Great Lakes Symp. VLSI*, 1996, pp. 178–181.

[5] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, “Asymptotic zero-transition activity encoding for address busses in low-power microprocessor-based systems,” in *Proc. Great Lakes Symp. VLSI*, 1997, pp. 77–82.

[6] W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano, “Power optimization of system-level address buses based on software profiling,” in *Proc. 8th Int. Workshop Hardw./Softw. Codes. (CODES)*, 2000, pp. 29–33.

[7] M. Stan and W. Burleson, “Bus-invert coding for low-power I/O,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 1, pp. 49–58, Mar. 1995.

[8] S. Komatsu, M. Ikeda, and K. Asada, “Low power chip interface based on bus data encoding with adaptive code-book method,” in *Proc. 9th Great Lakes Symp. VLSI*, 1999, pp. 368–371.

[9] L. Benini, A. Macii, M. Poncino, and R. Scarsi, “Architectures and synthesis algorithms for power-efficient bus interfaces,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 9, pp. 969–980, Sep. 2000.

[10] A. Bahari, T. Arslan, and A. Erdogan, “Interframe bus encoding technique for low power video compression,” in *Proc. 20th Int. Conf. VLSI Des.*, 2007, pp. 691–698.

[11] S. Ramprasad, N. Shanbhag, and I. Hajj, “A coding framework for low-power address and data busses,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 7, no. 2, pp. 212–221, Jun. 1999.

Power Estimation of Embedded Multiplier Blocks in FPGAs

Ruzica Jevtic and Carlos Carreras

***Abstract*—The use of embedded multiplier blocks has become a norm in DSP applications due to their high performance and low power consumption. However, as their implementation details in commercial field-programmable gate arrays are not available to users, and the power estimates given by the tested low-level tool are not accurate enough to validate high-level models, the work on power estimation of these blocks is very limited. We present a dynamic power estimation methodology for the embedded multipliers in Xilinx Virtex-II Pro chips. The methodology is an adaptation of an existing power estimation method for lookup-table-based components and uses information about the type of architecture of the embedded block. The power model is characterized and verified by on-board measurements and is ready for integration with high-level power optimization techniques. The experimental results show that the average accuracy of the model is higher than the average accuracy of the low-level commercial tool.**

***Index Terms*—Embedded multipliers, field-programmable gate array (FPGA), power estimation.**

I. INTRODUCTION

There is an extensive ongoing research work about the underlying field-programmable gate-array (FPGA) architecture. The number of lookup tables (LUTs) per cluster, the number of clusters per configurable logic block [1], the most efficient routing structures [2], and many other parameters are being explored in order to find the best tradeoff between design area, performance, and power consumption. Modern FPGA architectures also include special-purpose blocks, such as embedded multipliers and digital signal processing (DSP) blocks, that are used to accelerate arithmetic-intensive applications. They are not built from standard programmable FPGA fabric. Instead, their design corresponds to that of an application-specific integrated circuit (ASIC), as they are specialized for some chosen arithmetic functions and optimized to achieve the highest performance. Since only the required transistors and routing resources are used for the implementation of the embedded blocks, their power is optimized as well. Still, power estimation models are needed in order to be integrated into power optimization techniques. This is particularly important at higher levels of abstraction, where reliable information on power increase/decrease in each optimization step is necessary so as to avoid time-consuming design implementations.

The power estimation of ASICs has been studied in depth, and numerous techniques have been developed. However, they all need detailed information of the target circuit and/or proprietary technology, which is not available to the common user when the embedded multipliers in FPGAs are considered. Also, the noncommercial tool VPR that is often used to study FPGA architectures does not support the power estimation of these blocks. When a commercial tool, such as XPower (XPwr), is used for the power estimation of the embedded blocks, large estimation errors are detected, as it will be shown later.

Manuscript received August 01, 2008; revised December 23, 2008. First published May 29, 2009; current version published April 23, 2010. This work was supported in part by the Spanish Ministry of Education and Science under Project TEC2006-13067-C03-03.

The authors are with the Departamento de Ingenieria Electrónica, ETSI Telecomunicacion, Universidad Politécnica de Madrid, 28040 Madrid, Spain (e-mail: ruzica@die.upm.es; carreras@die.upm.es).

Digital Object Identifier 10.1109/TVLSI.2009.2015326