High-Throughput Entropy Encoder Architecture for H.264/AVC

Mahamaya P. Chimurkar
ME 2nd year
Department of Computer Science
Amravati

ABSTRACT

This paper provides an advancement in earlier low and high efficient entropy encoder architecture for H.264/AVC which contains all three entropy encoding methods available in the H.264/AVC standard, by replacing simple CAVLC encoder used by high-throughput CAVLC encoder and CABAC encoder by real-time multi-bin CABAC encoder in order to further increase overall throughput of the system. The high-throughput CAVLC encoder that uses a dual-coefficient scanning phase which determines all the required data for encoding phase which helps in improving the speed of the encoding phase. And real-time multi-bin CABAC encoder implements the parallelism among the steps of CABAC encoding operation and thus achieves the increased throughput. The proposed architecture is expected to achieve higher throughput as compared to the present architecture but at the cost of increased complexity.

Keywords

H.264/AVC video compression, CABAC, CAVLC, syntax elements, etc.,

1. INTRODUCTION

The H.264/AVC is the recent video coding standard. The standard provides 50% reductions in bitrate compared to previous standards like MPEG-2, H.263, etc., however significantly increases the computational complexity when compared to MPEG-4 part 2[1, 6]. Entropy encoder hardware is a key challenge in the development of a complete H.264/AVC encoder architecture. The block diagram of H.264/AVC is as shown in Figure 1 which highlights the entropy encoder that receives data from all coding blocks used by the encoder. These information includes the block size, inter/intra decision, motion vectors, quantization parameter, residual information, etc. The information sent to the entropy encoder is divided into datum, each named a "Syntax Element" (SE). A SE is composed by a type and a value. The type informs the origin of the SE while the value is the data to be encoded. The decoder will receive the coded SE and if the semantics specification of the standard is given, it will guide the construction order of the syntax elements. This process has to be strictly followed to produce a bitstream which is compatible with the H.264/AVC standard.

The entropy is the block which is responsible for representing the syntax elements in a lossless and as compact as possible representation. The EE contributes to the overall compression rate In the H.264/AVC standard three entropy encoders are defined: Exponential Golomb (EXPG), Context-based Adaptive Variable Length Coding (CAVLC) and Context-based Adaptive Binary Arithmetic Coding (CABAC)

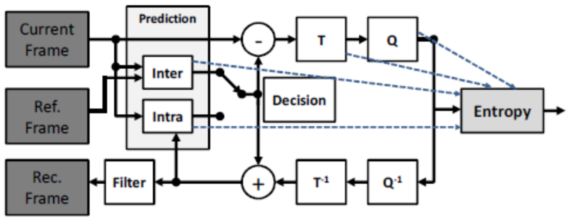


Figure 1. H.264/AVC Encoder block diagram[1]

The work in [1] presents an entropy encoder architecture with all three entropy encoding methods available in the H.264/AVC standard, while optimized to achieve high definition video (HD1080) with limited hardware and power resources. In order to further increase the throughput, work proposed in [1] is extended by replacing CAVLC and CABAC encoders by high throughput CAVLC[1, 9] and real time multi-bin CABAC encoders respectively.

2. BACKGROUND

Entropy encoder is a key challenge in development of H.264/AVC encoder architecture. The EE contributes to the overall compression rate and in turn throughput that need to be achieved. In order to obtain a good tradeoff between high throughput and FPGA resource usage for entropy encoder process, architecture that taking advantage of all the three encoders of H.264/AVC has been developed earlier. In order to further increase the throughput same work is extended with some advancements.

3. PREVIOUS WORK DONE

In [1] paper, author Cristiano C. Thiele contributes with an entropy encoder architecture with all three entropy encoding methods available in the H.264/AVC standard[7], while optimized to achieve high definition video (HD1080) with limited hardware and power resources in this process. Also a complete double-mode binary encoder for H.264/AVC is proposed with good tradeoffs between area and throughput; however it does not reach real time HD1080 capability.

In [2] paper, author Marc P. Hoffman investigated and realized a high-performance implementation of CAVLC. The design is based on previous architectures, but implements a modified scanning phase that allows for an improved encoding phase. This design is capable of encoding a 16 x 16 macroblock (MB) in a maximum of 258 clock cycles. At 200 MHz, the proposed implementation can encode one 1,080 p frame in at least 0.0105 s, or a minimum of 95 frames per second. This CAVLC design is intended for use by high-throughput H.264 architectures with wide-area surveillance applications. Its memory requirements are that of most high-throughput architectures, as a result of the arithmetic table elimination techniques use. The proposed architecture also displays relatively low power requirements, consistent with FPGA design.

In [3] paper, author P Jayakrishnan presented An optimized scheme for accessing the contexts and a Binary Arithmetic Encoder (BAE) capable of processing four bins per cycle. Hence parallelism and throughput is achieved by the means of reduction in syntax and bit level dependency. The allocation of bits in various Syntax Elements(SE) is analysed for the reduction in large amount of bit level operations involved and computational complexity involved.

In [4] paper, author G. D. Licciardo proposed a new context-adaptive variable length-coding encoder architecture particularly aimed to be implemented with field programmable logics (FPL) like FPGAs. The design implements different approaches in order to minimize the area cost as well as to speed up the coding efficiency, which allows real-time compression of 1080 p video streams coded in YCbCr 4:2:0 format. Priority cascading logics have been implemented in order to increase the parallelization degree of the pre-coding stage, whereas the employment of the arithmetic table elimination technique has allowed a large-area reduction of the encoder thanks to the elimination of 18 of the 38 tables needed for the encoding stage. The design achieves real time elaboration with an operation frequency of 63 MHz and occupies 2200 look-up table (LUT)s when implemented on a low-cost, low-end XILINX Spartan 3 FPGA, thus overcoming the most recent FPL implementation and making this encoder quite comparable both in terms of area and speed with some recently proposed ASIC implementations, so that it turns out to be a valid alternative also for application specific implementations.

In [5] paper, author Dieison Silveira presented the Reference Frame Context Adaptive Variable-Length Coder RFCAVLC), which is a low-complexity lossless solution to compress the reference data before storing them in the external memory. The proposed RFCAVLC reaches an average compression ratio superior to 32 % for the evaluated video sequences. The RFCAVLC design is able to reach real-time encoding for WQSXGA (3,200 X 2,048 pixels) at 33 fps. The RFCAVLC also achieves power savings related to external memory communication that exceeds 30 % when processing HD 1,080p videos at 30 fps.

4. EXISTING METHODOLOGY

The entropy encoder architecture proposed in [1] was based strictly on the features defined by the H.264/AVC standard. The block diagram of EE architecture is as shown in figure 4.

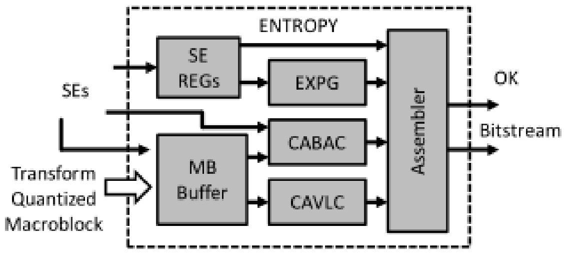


Figure 2 – Entropy encoder block diagram[1].

The inputs to the EE are all syntax elements that need to be assembled, and all added residual information as well. The outputs are 8-bit data and *ready* signal data. The SE REG is a set of registers that store the syntax elements used to assemble the NAL header, SPS, PPS Slice and MB headers. The MB Buffer is a memory that stores an entire macroblock. The final block is Assembler that works with four types of data: absolute syntax element, EXPG coded elements, CAVLC elements and CABAC binary code.

4.1 EXPG

As described in [1] the EXPG coder generates variable length codes with regular construction. The SE coded by EXPG are divided in four types: Unsigned, Signed, Mapped and Truncated. The EXPG code structure is presented based on ZeroPrefix and INFO as follows:

Code = [ZeroPrefix]1[INFO]

Where ZeroPrefix is extracted from $\mathrm{Code}_{\mathrm{Num}}$ as

$M = \log_2(\mathrm{Code}_{\mathrm{Num}} + 1)$, M indicates the number of zeros in ZeroPrefix.

And INFO = $\mathrm{Code}_{\mathrm{Num}} + 1 - 2^M$

4.2 CAVLC

There is number of literature available describing the working of CAVLC. As discussed in [1, 2] CAVLC codes the residual information of a block. The coder can work with 4x4 (16 or 15 coefficients) or 2x2 (4 coefficients) block sizes, according to the block type: Luma/Croma and DC/AC[1]. These blocks of data are represented as coefficient arrays in the CAVLC block diagram shown in Figure 2. CAVLC takes every 4 x 4 and 2 x 2 block of coefficients and reorders the data into a vector using a zigzag scan, These coefficient vectors are analyzed and converted into binary bit streams using a combination of look-up tables and simple arithmetic. CAVLC also predicts the number of coefficients in the current array using the context variables nA and nB. These two variables represent the number of non-zero coefficients in the neighboring 4 x 4 blocks, in particular the blocks that lie above (nA) and to the left (nB) of the current 4 x 4 block. The predicted number of non-zero coefficients in the current 4 x 4 block nC is equal to the average of nA and nB. If one of the two context variables are not available, then nC is simply set equal to the available context variable. If neither are available, nC is set equal to 0. Also, nC is automatically set to 0 when encoding DC data.

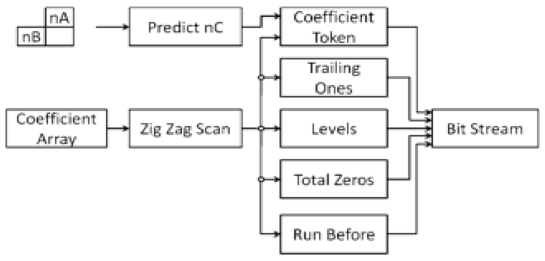


Figure 3. The CAVLC block diagram [2]

After CAVLC receives the coefficient vector and nC, it is ready to parse the data.CAVLC encodes each vector into the following five sections of data: coefficient token, trailing ones, levels, total zeros, and run before. The **coefficient token section** represents the number of non-zero coefficients and the number of trailing ones in the current vector. The **trailing ones section** is determined algorithmically and represents the sign of each trailing one. The **levels section** contains up to 16 VLC codewords, that contain the magnitude and sign of all the non-zero coefficients in the current vector, excluding trailing ones. The **total zeros section** of the CAVLC bit stream contains the number of zero coefficients that appear before the last non-zero coefficient in the current vector. The final section of the CAVLC bit stream for a 4 x 4 (or 2 x 2) block of data is the **run before section**. This section is made up of a varying number VLC codewords that represent the number of zeros that run before each non-zero coefficient.

4.3 CABAC

Also for describing the working of CABAC wide range of literature is available. As explained in [1, 3] the CABAC codes the SEs of macroblock prediction, quantization parameters and the residual information of the quantized transform macroblock. As in the CAVLC case, the coefficients are scanned by CABAC in zigzag order to take advantage of quantized transform block sparsity characteristics.

The syntax elements extracted from the block are: coded_block_flag(CBF),significant_coeff_flag,last_significant_coeff_flag, coeff_abs_level_minus1, coeff_sign_flag.

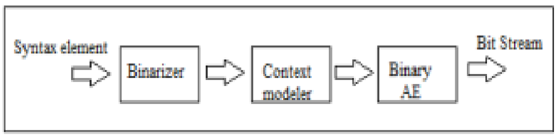


Figure 4. Block diagram of CABAC encoder[3]

The CABAC coding process involves the following steps:

4.3.1 Binarization: In this process the syntax element is converted into a binary code named binstring. There are four main methods used to make this: Unary (U), Truncated Unary (TU), Fixed Length (FL) and Parameterizable Exp-Golomb (EGk). Some syntax elements work with methods which are derived from these ones.

4.3.2 Context Model Selection: A Context Model is a probabilistic model with the statistical occurrence rate for each symbol, while each syntax element has a set of allowed Context Models. In order to generate the index for the more appropriate Context Model, the following data are taken into account: the syntax element itself, previous context model, and the bin value previously coded. There are more than hundred context models specified in the H.264 standard, given the diversity of image data sets. There are specific syntax elements which do not require a context model, due to the equiprobable occurrence of these symbols.

4.3.3 Arithmetic Encoding: The arithmetic encoder is responsible for generating the bitstream from the reading of the bins and their models of context, when the latter exist. This process begins by defining the division of an initial range R into sub-ranges based on Context Model. There is a symbol value defined for each sub-range. There are just two sub-ranges for each bin, associated with the possible values of ‘0’ and ‘1’. The bin is read and compared with the symbol value. The corresponding sub-range is selected and sets the new range R. This process is iterative.

5. ANALYSIS AND DISCUSSION

The CAVLC encoder is often seen as a bottleneck in the H.264 encoding engine. The encoder requires knowledge of previously encoded data to operate, which limits the number of instances of the CAVLC encoder to one per slice (a fixed portion of a coded video frame). With this limitation, encoding high-definition video (720 and 1,080 p) in real-time requires a high-performance design that is difficult to implement in software. Thus there is need of a high-performance implementation of CAVLC.

Also though CABAC is considered better entropy coding method, but it requires a significant amount of logic and increases the hardware implementation cost on a large scale. Due to the SE level dependency as well as bit-level dependency, it is also difficult to parallelize, thus the throughput of a single CABAC encoding engine is limited. This makes the design of CABAC encoder very challenging and crucial in the whole video encoding system. It can be seen from the trend that the designs are transferring from 1 bin/cycle to multi-bin/cycle to provide a higher throughput. The BAE logic is duplicated and cascaded sequentially to process multiple bins in one cycle.

6. PROPOSED METHODOLOGY

The entropy encoder architecture[1, 9, 10] proposed by Cristiano C. Thiele consists of Macroblock buffer, EXPG, CAVLC and CABAC encoder, and a final assembler. In this paper, CAVLC encoder is replaced with high-throughput CAVLC encoder and also CABAC encoder with real-time multi-bin CABAC encoder in order to increase the overall throughput of the architecture proposed in [5].

6.1 Proposed Methodology For CAVLC

The five sections that make up the CAVLC encoder can be performed in parallel with some pre-processing. This pre-processing is performed in the scanning phase of the CAVLC architecture, shown in Figure 5. To improve the throughput of the architecture, the suffix length and magnitude of each level, in addition to the number of zeros that run before each non-zero, are determined during the scanning phase. Having this data allows the encoding phase to be completed in a constant 3-clock cycles.

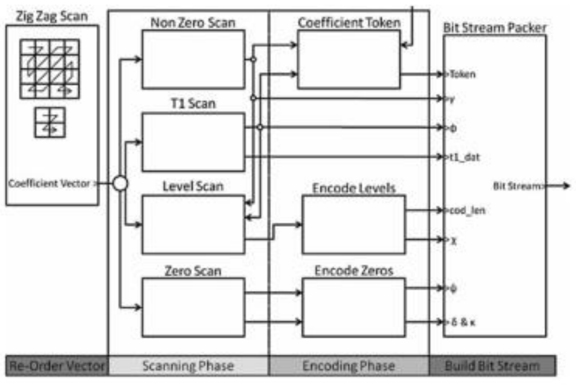


Figure 5. CAVLC hardware architecture[2]

Table 1. Coefficient Pairs [2]

Output	Coefficient pair							
	(0,0)	(0, ±1)	(0, level)	(±1,0)	(±1, ±1)	(±1, level)	(Level, 0)	(Level, level)
Level Buffer	–	–	Store Level	–	–	Store Level	Store Level	Store Levels
Zero Buffer	Store Zeros	Store Zero	Store Zero	Store Zero	–	–	Store Zero	–
t1_dat	–	Append T1 sign	–	Append T1 sign	Append T1 signs	Append T1 sign	–	–
γ	–	+1	+1	+1	+2	+2	+1	+2
ψ	+2	+1	+1	+1	–	–	+1	–
φ	–	+1	–	+1	+2	+1	–	–

6.1.1 The Scanning Phase

To reduce the number of clock cycles required to scan the vector, a modified method of the dual-coefficient scanning operation is implemented here.

The proposed scanning operation encounters 1 of the 14 unique pairings of coefficients that are shown in Table 1 during each of its 8 cycles. These pairs are made up of a combination of zeros, trailing ones, and levels. The data needed to encode each pair is saved so that the number of cycles required for the encoding phase is minimized. Table 1 contains all the outputs from the scanning phase as rows and the coefficient pairs as columns. Each output is updated according to the pair of coefficients that are currently being scanned. If the current pair has no effect on an output, then the table marks the appropriate location with a '–'.The level and zero buffers are the proposed additions to the scanning phase. These buffers contain the statistics that allow the encoding phase to determine all the level and run before code words in a single clock cycle.

Once every coefficient pair from the vector array has been scanned, the scanning phase is complete. The architecture then signals that all the registered values are ready for the encoding phase. All outputs of the scanning phase are registered while the encoding phase is active.

6.2 Proposed Methodology For CABAC

Also in place of CABAC the projected architecture of CABAC encoder performs both binarisation and context modeling in parallel and is a multi-bit parallel processing design and high throughput CABAC[8]. Hence the number of cycles needed for processing is reduced and throughput is increased. CABAC encoder consists of PIPO buffer, Binarisation and context modeling and Binary arithmetic encoder.

6.2.1 Binarization and context modeling (BCM)

To enhance the efficiency of CABAC, the design of CM should also provide a throughput equal to processing four bins each cycle on an average. 6 two-port SRAMs are used in our design to store the "460 contexts" to make context access efficient. The "460 contexts" are divided into 6 groups according to the feature of context access order. A simple forwarding scheme is used to avoid hazards such as read memory collision. The context model is updated and stored according to the different types of syntax elements. The Binarisation and Context Modeler (BCMODS) is processed in parallel until different data is accessed from the same bank.

*6.2.2 PIPO Buffer (14 * 14)*

The PIPO buffer is used to connect the output of the predecessor stage and to provide a stable output. The output of BCM i.e. Modes of BCMODS are given to PIPO.

6.2.3 Binary Arithmetic Encoder

The BAE is four stage pipelined architecture. The BAE processes four BCMODS in parallel obtained from the PIPO buffer. The update range and low stages are duplicated to obtain the throughput. The amount of complexity involved in the consequent division of interval subdivision and renormalization in order to process two element types in parallel without increasing the critical path. The pipeline architecture is used in our design of BAE to enlarge the throughput as much as possible. Four Update Range and four Update Low logic units are pipelined to achieve a throughput of 4 bin/cycle in BAE part. The first pipeline stage in BAE is to generate the four rlps values for each bin according to the value of pStateIdx from the context modeling module. One of these four values is selected out in Update Range stage and will be used as the rangelps (rlps) value in Table 2 for interval update. The Update Range stage will update the value of Range four times for the four bins in a cycle and generate the information needed for the update of Low. Then value of Low is updated in the next stage and the Bit Pack stage will pack the output bit of Update Low into the output stream.

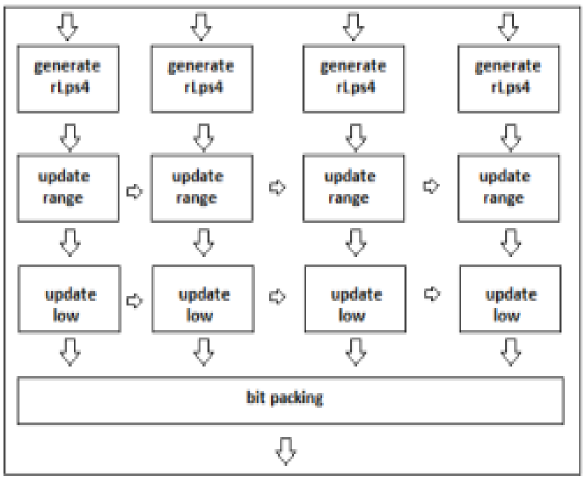
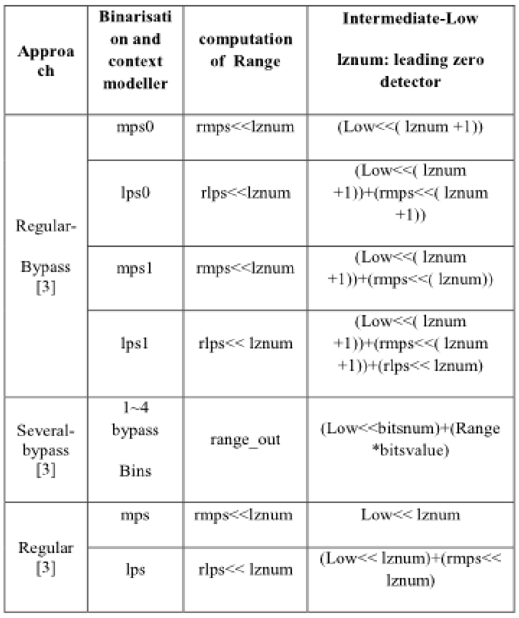


Figure 6. Architecture of BAE [3]

7. POSSIBLE OUTCOME AND RESULT

From the design of BAE it is clear that BAE will be capable of achieving maximum number of bin/cycle on an average, reduction in critical path and also will result in high speed. Also CAVLC will result in higher throughput. So, overall throughput of proposed architecture will be increased.

Table 2 Equations Of Range And Low Updation [3]



Approa ch	Binarisati on and context modeller	computation of Range	Intermediate-Low lznum: leading zero detector
Regular-Bypass [3]	mps0	rmps<<lznum	(Low<<(lznum +1))
	lps0	rlps<<lznum	(Low<<(lznum +1))+(rmps<<(lznum +1))
	mps1	rmps<<lznum	(Low<<(lznum +1))+(rmps<<(lznum))
	lps1	rlps<< lznum	(Low<<(lznum +1))+(rmps<<(lznum +1))+(rlps<< lznum)
Several-bypass [3]	1~4 bypass Bins	range_out	(Low<<bitsnum)+(Range *bitsvalue)
Regular [3]	mps	rmps<<lznum	Low<< lznum
	lps	rlps<< lznum	(Low<< lznum)+(rmps<< lznum)

8. CONCLUSION

The proposed architecture is expected to achieve increased throughput at the expense of increased complexity. The proposed architecture is unique as it includes all entropy encoders presented in the H.264/AVC specification. The proposed entropy encoder is capable to reach processing power sufficient to support full high definition video encoding for the H.264/AVC standard. Also proposed entropy encoder architecture for H.264/AVC provides a real-time full high definition (HD1080) resolution video coding.

9. FUTURE SCOPE

Future work can be done on overcoming the limitations of the proposed architecture. Further work can be done on making an encoder aware of environmental and communications conditions, capable of adjusting itself to meet channel, quality and energy constraints.

10. REFERENCES

[1] Cristiano C. Thiele, Bruno B. Vizzotto, Andre L. M. Martins, Vagner S. da Rosa, Sergio Bampi. : “A Low-Cost and High Efficiency Entropy Encoder Architecture for H.264/AVC”. IEEE, -1-4673-2658-2/12, 2012.

[2] Marc P. Hoffman, Eric J. Balster, William F. Turri. : “High-throughput CAVLC architecture for real-time H.264 coding using reconfigurable devices”. Springer-Verlag Berlin Heidelberg 2013, DOI 10.1007/s11554-013-0345-2, 2013.

[3] P Jayakrishnan, P V Anitha Lincy, R Mohamed Niyas. : “A Real Time Multi-bin CABAC Encoder for Ultra High Resolution Video”. IEEE, 978-1-4673-5090-7/13, pp. 402-405, 2013.

[4] G. D. Licciardo, L. Freda Albanese. :”Design of a context-adaptive variable length encoder for real-time video compression on reconfigurable platforms”. IET Image Processing, Vol. 6, Iss. 4, pp. 301-308, 2012.

[5] Dieison Silveira, Bruno Zatt, Luciano Agostini, Marcelo Porto. : “Reference frame context-adaptive variable-length coder: a real-time hardware-friendly approach for lossless external memory bandwidth reduction in current video-coding systems”. Springer-Verlag Berlin Heidelberg 2014, DOI 10.1007/s11554-014-0443-9, Published on-09 August 2014.

[6] I. Richardson “The H.264 Advanced Video Compression Standard”, 2nd ed, John Wiley & Sons, 2010.

[7] INTERNATIONAL TELECOMMUNICATION UNION. IYU-T Recommendation H.264(03/10); Advanced Video Coding for Generic Audiovisual Services, 2010.

[8] L. C. Wu, Y. L. Lin, “A high throughput CABAC encoder for ultra high resolution video”, ISCAS, 2009, pp 1048-1051.

[9] F. L. Ramos, A. Susin, S. Bampi, “High Throughput CAVLC Hardware Architecture with Parallrl Coefficients Processing for HDTV H.264/AVC Encoding”. Athenas, International Conference on Electronics, Circuits, and Systems (ICECS), 2010.

[10] L. F. Albanese, G. D. Licciardo, High speed CAVLC Encoder Suitable for Field Programmable Platforms, Polonia, International Conference on Signals and Electronic Systems(ICSES), 2010.