Proceedings of the 5th WSEAS Int. Conf. on Signal Processing, Computational Geometry & Artificial Vision, Malta, September 15-17, 2005 (pp94-101)

Optimisation techniques for the system on chip implementation of JPEG encoder

V. AMUDHA , B. VENKATARAMANI AND G. SEETHARAMAN
Department of Electronics and Communication Engineering
National Institute of Technology, Tiruchirappalli, INDIA

Abstract
Systems on chips have both general-purpose microprocessors and custom blocks optimized for specific functions. In this paper, Altera APEX 20KE200 based SOC kit with Nios soft-core processor is considered for the implementation of JPEG encoder. For an 8X8 matrix of image pixels, JPEG encoder is implemented in hardware as a custom block and its computation complexity is compared with that implemented using high level language. A number of optimization schemes are proposed for minimizing the computation time in the custom block. This includes employing the *13 multiplier Algorithm Architecture Transform (AAT)* for 2D DCT computation, *internal clock generation scheme which increases the speed of the custom instructions by 50%* and use of *memory read and write operations at different rates*. A scheme for *concurrent execution* of the operations in the custom block and data transfer as well as other operations by the Nios core is also proposed. From the implementation results, it is observed that for sub image of size 8X8, the hardware custom block is faster by a factor of *twenty six* compared to software implementation. The optimization schemes proposed in this paper are also applicable for the computation of other image transforms such as 2D DWT and encoders such as ITU H.263.

Keywords: Clock, Concurrent execution, DCT, FPGA, JPEG, SOC

1 Introduction

The ability to design, fabricate and test ICs with gate counts of the order of a few million has led to the development of complex embedded systems on chip. Design of System on chip (SOC) may be defined as the design of a complex IC, which integrates the major functional elements of a complete end product into a single chip or chip set. Hardware components in a SOC may include one or more processors, memories, and dedicated components for accelerating critical tasks, and interfaces to various peripherals. It may also include analog, RF, Micro-Electro-Mechanical Systems (MEMS) and optical input/output interfaces.

One of the approaches for SOC design is the platform based approach. For example, the platform FPGAs such as Xilinx Virtex II Pro and Altera Excalibur include custom designed fixed programmable processor cores together with millions of gates of reconfigurable logic devices.[1] In addition to this, the development of IP cores for the FPGAs for a variety of standard functions including processors, enables a multi million gate FPGA to be configured to contain all the components of a platform based FPGA. Development tools such as the Altera SOPC builder enable the integration of IP cores and the user designed custom blocks with the soft-core processors such as the Nios processor. ([2], [3]) Soft-core processors are far more flexible than the hard-core processors and they can be enhanced with custom hardware to optimize them for specific application. The technique for efficient implementation of the various blocks of JPEG encoder on the Altera APEX 20KE200 based SOPC kit is presented in this paper.

2 An overview of JPEG encoder and its various blocks

The block diagram of the JPEG encoder is given in Fig.1. Brief description about the technique used for the Discrete Cosine Transform (DCT) transform is given in this section. The details of the other blocks of the encoder may be found from [4],[5] [6]. The DCT and quantization are the most computation intensive tasks in the JPEG encoder and are desirable to be implemented in hardware. The zigzag scanning can be carried out using look up table and hence it is found to be efficient in hardware implementation by just swapping the memory contents. The rest of the blocks in the encoder may be implemented either in software or in hardware.

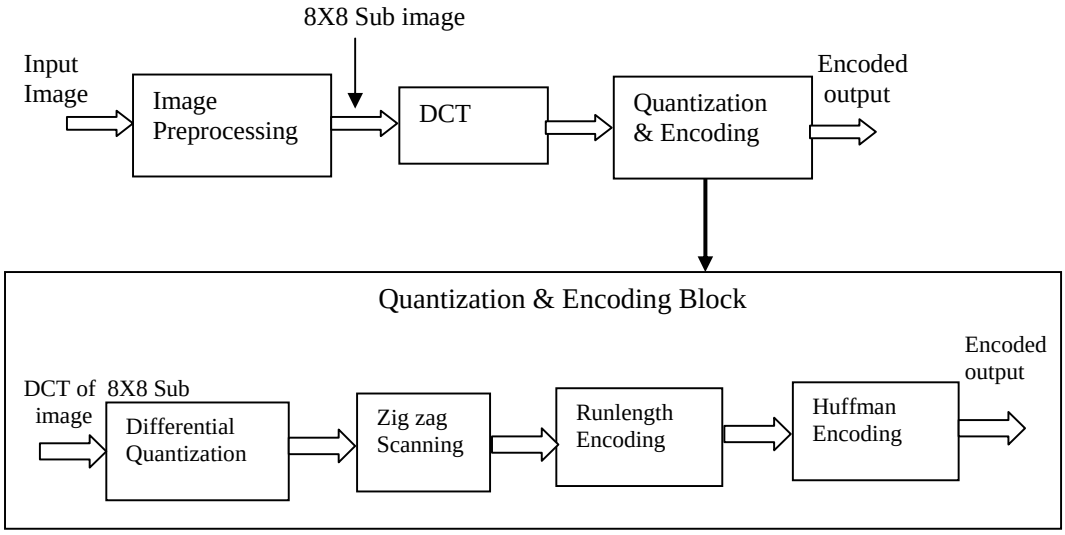

Fig. 1. Block diagram of the JPEG encoder

2.1 Discrete cosine transform

Discrete Cosine Transform (DCT) transforms the information in space or time domain to the frequency domain and has a number of desirable properties such as real transform coefficients, good energy compaction and availability of fast computation algorithms [7]. In view of these, it has been adopted in many compression and decompression standards such as JPEG, MPEG and H.263. In the JPEG encoder, the image to be compressed is split into 8X8 sub images and passed to the DCT block. X(k), the k^{th} transform coefficient of one dimensional DCT of a sequence x(n) of length M is given by

$$X(k) = \alpha(k) \sum_{n=0}^{M-1} x(n) \cos[(2n+1)k\pi/2M \quad (1)$$

The inverse transform coefficient x(n), the n^{th} data in the sampled sequence, can be expressed in terms of the DCT coefficients as follows:

$$x(n) = (1/M) \sum_{k=0}^{M-1} \alpha(k)\, X(k) \cos[(2n+1)k\pi/2M] \quad \ldots(2)$$

where $\alpha(k) = 1/\sqrt{2}$ if $k = 0$

$= 1$ $k \neq 0$

The 2D DCT of the 8X8 image block is computed, first by computing the 1D DCT of each row of the matrix and then computing the 1D DCT of the resulting matrix along the column. VLSI implementation of discrete cosine transform has been studied in a number of previous works. [7],[8],[9]. In this paper, DCT computation using Algorithm-Architecture Transform (AAT) structure shown in Fig.2, is considered [8]. *The 8-point DCT architecture derived using this technique, reduces the number of multiplications from 56 to 13.*

3. System design using SOC kit

3.1 An overview of the SOC kit

System on chip (SOC) kit used for the implementation of 2D DCT is based on Altera® APEX20K200E device and is pre-loaded with a 32-bit Nios embedded processor system reference design. The Nios system module (CPU and peripherals) typically occupies between 25% to 35% of the logic elements on this device and the remaining logic elements can be used for the implementation of user defined custom blocks. Nios development software supplied with this kit includes a Quartus® II project directory containing reference design examples.

The reference design and software are preloaded in flash memory, and boot on power-up. The reference design software includes a monitor that can be used to download and debug programs. With SOPC builder, designers can select and parameterize intellectual property (IP) block from an extensive list of communication, digital signal processing (DSP), microprocessor, interface cores, and common microprocessor peripherals.

SOPC builder integrates complex system components such as IP blocks, memories, microprocessors, and interfaces to off-chip devices including ASSPs and ASICs on Altera's high-density FPGAs. SOPC builder also saves designer's time by automatically generating software to match the target hardware

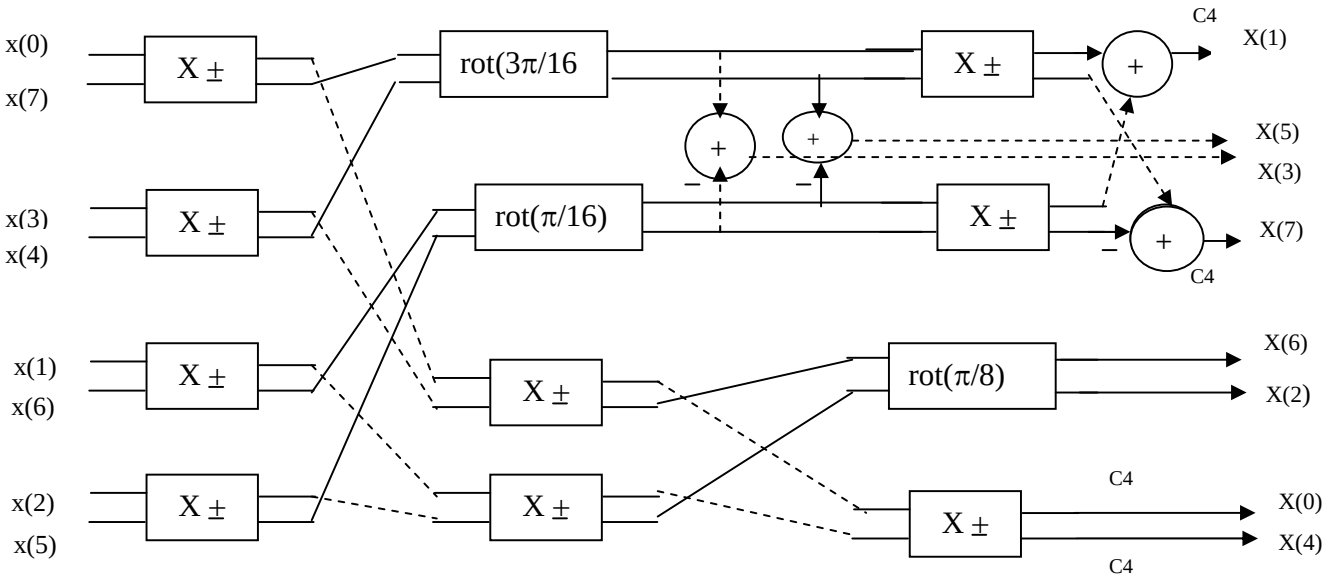

Fig.2.a Algorithm-Architecture Transform approach using 13 multipliers for 8X1 DCT

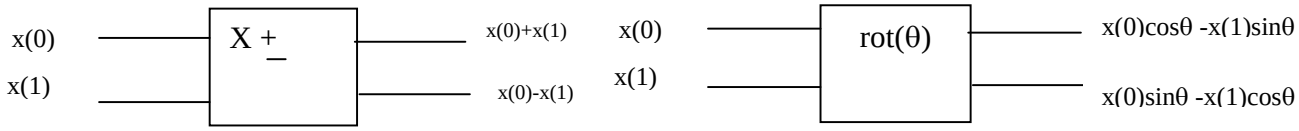

Fig.2.b Processing elements for DCT

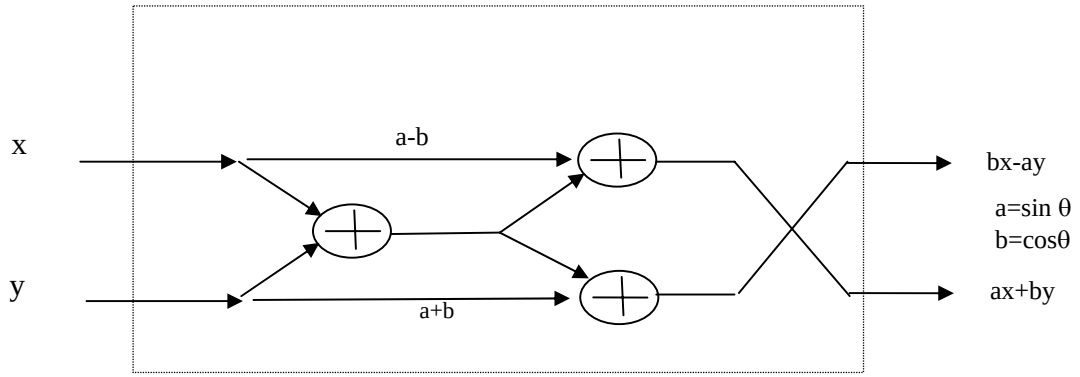

Fig. 2.c. Internal Blocks of rot(θ)

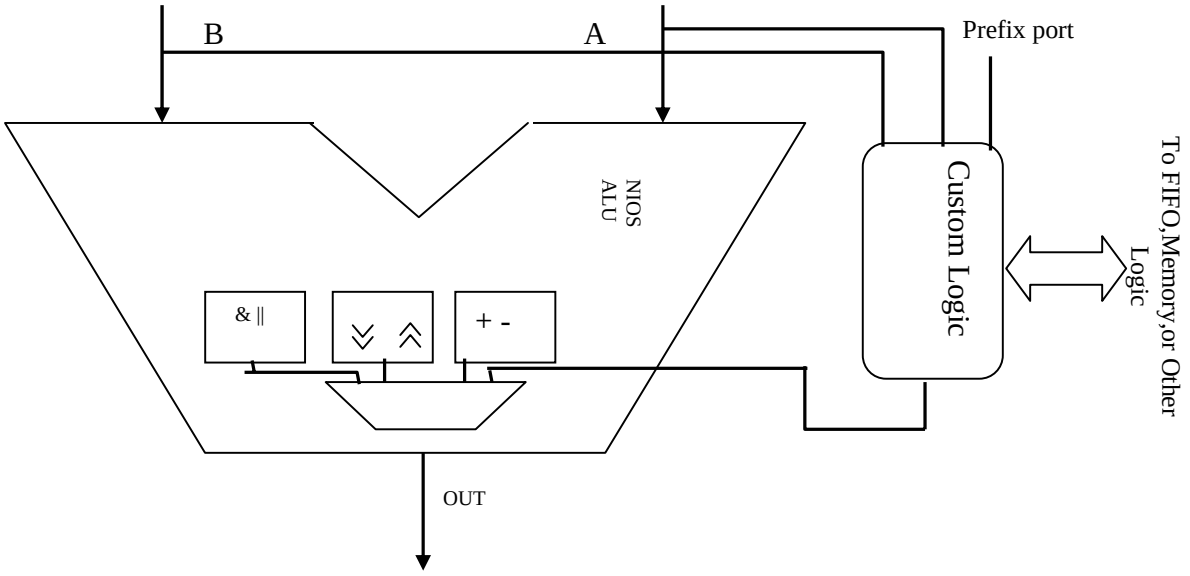

Fig. 3. Adding Custom Logic to the Nios ALU

3.2 Custom instructions of NIOS processor

The functions which cannot be executed efficiently using the in-built IP blocks may be implemented using custom logic blocks which are designed using Quartus® II design software. The interfacing details of the custom logic with the Nios ALU is shown in Fig.3. The performance of the Nios processor is enhanced by integrating these blocks with the Nios core using the SOPC builder system development tool. The program to be executed by the Nios core is written in C/C++ and the custom instructions executed by the custom block are defined as software macro and invoked in C/C++ program.

Custom instructions consist of two essential elements, a *Custom logic block*, the hardware that performs the user-defined operation, and a *Software macro, which* allows the system designer to access the custom logic through software code.

The task performed by the custom block may be defined either as single-cycle combinatorial operation or as multi-cycle sequential operation. In both cases, two 32-bit operands may be passed to custom block and a 32-bit result is returned. The CPU clock is made available to the custom block only when it is defined to be sequential.

An 11 bit prefix code may be sent to the custom block along with the other operand(s) through the prefix port. This may be used to specify the type of operation to be performed in the custom block.

3.3 Study of the custom instructions and some observations

The custom instructions for add, multiply and convolution are added to the Nios core and studied by defining them as both combinatorial and sequential blocks. The sequential blocks require more than two clock cycles for correct operation. (The number of clock periods is chosen to be the number of clock cycles required for the sequential block +1). The combinatorial blocks require 1 clock cycle. However, due to the overheads involved, for every call to a custom block, the Nios CPU spends at least 7 CPU cycles. For less computation intensive tasks, it would be preferable to make the Nios CPU to wait during every call to the custom block. For highly computation intensive tasks, it would be desirable to make the CPU concurrently working.

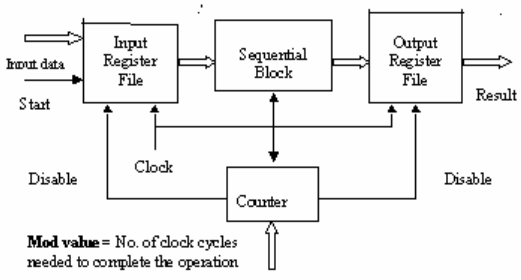

Fig.4 Scheme for concurrent execution of custom logic with Nios core

3.4 Custom instructions with concurrent execution

In this paper, we propose a scheme shown in Fig.4 to ensure the concurrency between the Nios processor and custom logic.

The execution of the custom instruction is split into a number of phases: write operand and operand number, issue start or reset signal, and read the results. Each phase of the instruction requires only 2-3 clock cycles. When a custom instruction is to be executed, first, the input data is fed into the input register file one after another. Next, a reset signal and start signal are issued to start the computation. The counter stops the computation process after a predetermined number of clock cycles. Using the read phase, the processor can read the result.

While adding custom instructions in SOPC builder, the number of cycle counts after which we need to process the result of custom block, is given as either 2 or 3.

3.5 Study of the clock generation schemes

3.5.a A new scheme for internal clock generation

When the custom block is integrated to the Nios core using the SOPC builder, the CPU clock (of frequency 33.33 MHz) is available as one of the inputs to the custom block.

Sequential circuits may use this clock as the basic clock from which other clock signals are derived. However, if the custom block requires a large number of sequential operations, it would be preferable to use a higher clock frequency to minimize the computation time. This may be achieved by generating the clock internally in the custom block. Fig.5 gives the circuit diagram of a clock generation scheme, which consists of a delay block and an inverter.

For example, to perform 8X8 DCT, a few clock cycles may be used to write the operands. Then the reset and start signals are issued. Assuming that a

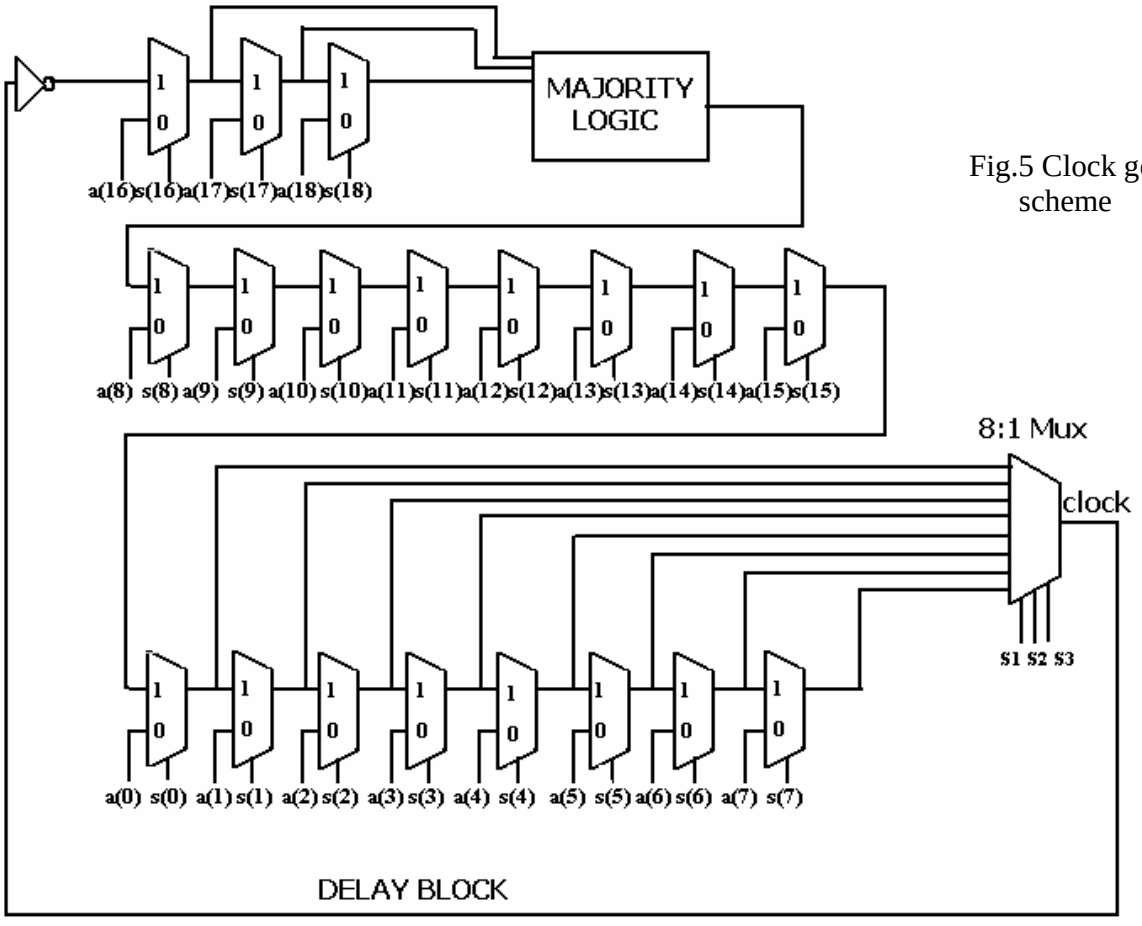

Fig.5 Clock generation scheme

4-stage pipeline is used for computing DCT, the counter stops the operation after 4N clock cycles. The processor can read the result now and restart the entire process. Pipelining also enables a batch of data to be processed one behind the other and the termination count for the counter should be appropriately modified. As the input register file and output register file are the integral part of the custom logic there is no need for separate arbitration for writing the data and reading the result of the custom logic. But the overhead here is additional memory needed to store the input data and output data in register files.

In APEX 20KE200, LUT delay is of the order of 1ns. Each 2 input MUX requires a 4 input LUT for the implementation. Hence when 5 multiplexers are in cascade, the clock period is of the order 10ns. The actual clock period depends on the interconnect delay. The select inputs s(0)-s(18) are connected to the lower order 18 bits of dataa port and the data inputs a(0)-a(18) are connected to the datab port of the Nios CPU. When prefix code is 0, the clock is reset to 0. And it keeps toggling periodically for other prefix codes. Clocks with periods in the range of 6.076 ns - 13.704 ns are obtained when 1 to 8 multiplexers are in cascade. In order to measure the clock frequency, the scheme given in Fig.6 is used. In this scheme, the CPU clock of period 30ns is divided by 40 to generate the count enable signal of duration 1.2μs. The Cycle counter is initially cleared and is applied with the internal clock. The count reached by this counter when the count enable is high, gives an estimate of the frequency of the internal clock.

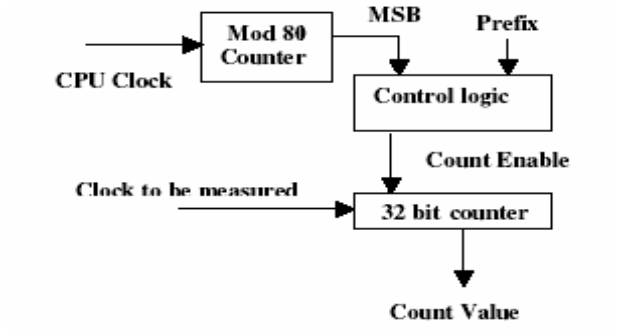

Fig.6 User clock frequency measurement scheme

3.5.b Internal clock generation using PLL

For the custom block declared as sequential, the internal clock of higher/lower frequency may be obtained by multiplying the processor clock using the PLL in APEX devices. For this purpose, the megacore function *altclkclock* may be used. Fig.7 gives the ClockLock and ClockBoost circuitry in APEX 20K devices.

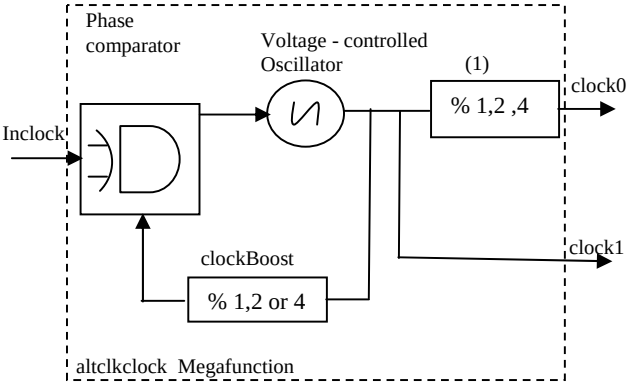

Fig.7. Clock Lock and Clock Boost circuitry in APEX 20K devices

Either single output clock of 1X, 2X, and 4X can be used or combination of both can be used. For the DCT, the *altclkclock* is used to generate the clocks for memory read and write operations. The scaling multiplication of APEX20KE PLLs allows a wide range of user-defined multiplication and division ratios that are not possible with DLLs. Hence it is possible to achieve non-integral multiples of the CPU clock frequency.

The clock multiplication factor to be realized using mega core function *altclkclock* has to be specified at the synthesis time and hence the clock frequency cannot be dynamically altered as in the proposed new scheme.

4 Implementation results

The various blocks of the JPEG encoder given in Fig.1, are implemented in SOC kit and the results are presented in this section.

All the blocks are implemented in the Nios core using C programs and the number of clock cycles required for execution of each of these blocks are listed in Table 1. Two dimensional DCT, quantization and zig - zag scanning operations are also implemented in hardware and integrated to Nios core as custom block. In order to enable these operations to be carried out concurrently with run length coding and Huffman coding in Nios core, prefix codes are used for different operations such as loading the 2D data into block RAM, reading the processed results, initialize the parameters and start the computations.

Table 1. Computation Time of various blocks of JPEG encoder for 8X8 Sub image

Function	No. of CPU clock cycles for software approach (C)	No. of CPU clock cycles for hard-ware approach (Custom block)
2D- DCT	14,331	216
Quantization	11,318	73
Zigzag Scanning	2,178	320
Run length coding	2,924	128
Huffman Encoding	2,452	128
Data transfer for 128 bytes	0	400
Total	33,203	1,265

Table 2. Resource utilization of various blocks of JPEG encoder for 8X8 sub-image

Function	No. of logical elements	No. Memory bits
2D- DCT	3,832	1,536
Quantization & zigzag	263	2,240
Run length coding & Huffman Encoding	407	8,448
CPU core	2,672	26,496
Available	8,320	1,06,496
% of usage	**86.23%**	**36.36%**

For the 2D DCT and quantization, more number of memory read and write operations are required. In order to reduce the computation time, the *CPU clock is multiplied by two* and it is used as the basic clock for memory operations. Moreover two clock cycles are used for memory read but only one cycle is found to be adequate for the memory write. The computation time required for each of these operations in the custom block is given in Table 1. The area required for each of these blocks is given in

Table 2.For the purpose of comparison, the huff man encoder and the run length encoder are also implemented in hardware as custom blocks and the cycle counts are also given in Table 1. From Table 1 & 2, it may be concluded that the custom block enhances the speed of the JPEG encoder by a factor of 26.25 **.**

5 Conclusions

The optimization schemes proposed in this paper for increasing the speed of SOC based JPEG encoder have been validated. Scheme proposed for concurrent execution of custom block and Nios CPU is found to give satisfactory results. Extension of this work for the implementation of H.263 encoder is under progress. The internal clock generation scheme proposed in this paper is useful for the study of self tuned wave-pipelined circuits.

References

[1] Martin, G.Chang, H, System-on-Chip design, Proc. of Intl. conf. on ASIC, 2001, pp 12 – 17

[2] Altera documentation library- 2003, Altera corporation, USA

[3] W. Tuttlebee, Software defined radio, John Wiley & Sons ltd. USA, 2004

[4] Borko Furht, Stephen W. Smoliar, Hongjiang Zhang, " Video and Image Processing in multi-media Systems", Kluwer Academic Publishers, 1995.

[5] Fred Halshal, "Multimedia Communications Applications, Networks, Protocols and Standards", Pearson Education Asia, 2001.

[6] Khalid Sayood, "Introduction to Data Compression", Pearson Education, Asia, 2003.

[7] Bruce A. D., J. Ross B., Böhm A. P. W., Charles R., and Monica C., "Accelerated Image Processing on FPGAs", IEEE Transactions On Image Processing, VOL. 12, NO. 12, DEC 2003 1543-1551

[8] K. K. Parhi, "VLSI signal processing systems", John Wiley & Sons, 1999

[9] August, N.J. and Dong Sam Ha, "Low power design of DCT and IDCT for low bit rate video codecs", Multimedia, IEEE Transactions on Volume 6, Issue 3, June 2004 Page(s):414 - 422

[10] Yonghong Zeng, Lizhi Cheng, Guoan Bi and Kot. A.C., "Integer DCTs and fast algorithms" IEEE Transactions on Signal Processing Volume 49, Issue 11, Nov. 2001 pp 2774 - 2782

[11] Che-Hong Chen; Bin-Da Liu; Jar-Ferr Yang; Jiun-Lung Wang; "Efficient recursive structures for forward and inverse discrete cosine transform", IEEE Transactions on Signal Processing, Volume 52, Issue 9, Sept. 2004 Page(s):2665 - 2669