HIGH PERFORMANCE IDCT REALIZATION USING COMPLEX ARITHMETIC

Kar-Lik Wong[1] *and Nigel Topham*

Siroyan Ltd
200 Brook Drive, Green Park
Reading RG2 6UB, U.K.
http://www.siroyan.com

ABSTRACT

In this paper we describe a high performance IDCT realization using complex arithmetic. The algorithm is based on novel factorization of the IDCT designed to exploit the complex multiplication capability provided by the OneDSP processor. We show a very efficient loop schedule implementing an 8-point IDCT in 9 cycles in each cluster. A single cluster OneDSP processor running at 300MHz is capable of decoding MPEG2 bit-streams at ATSC resolutions. Error analysis of the algorithm based on IEEE 1180 compliance testing is presented.

1. INTRODUCTION

The 2D 8x8 inverse discrete cosine transform (IDCT) [1] is a major component of the decompression engine in most video coding standards. A lot of research effort has been devoted to investigate efficient software and hardware realizations of this important operation. A popular approach is to use the row-column method to decompose a 2D 8x8 IDCT into 16 8-point 1D IDCT. A fast algorithm is then used to compute the 1D IDCTs. This often results in simple and regular computational structures while keeping the number of operations comparable to fast algorithms directly computing a 2D transform. A common goal among various fast IDCT algorithms is to minimize the number of arithmetic operations, especially multiplications. To date, 11 multiplications is the reported minimum required to compute an 8-point 1D IDCT [2]. However, it has been pointed out, for example in [3], that such algorithms might not be optimized for modern processor architectures designed to exploit sub-word SIMD parallelism. In particular, significant overhead is often incurred when re-ordering data to enable such parallelism to be exploited.

In this paper, we describe a simple fast 8-point IDCT algorithm that is designed to be realized on a OneDSP processor extremely efficiently. In section 2 we derive the fast algorithm. In section 3 we briefly describe the OneDSP architecture and the software realization that computes an 8-point IDCT in 9 cycles on a single cluster. Section 4 presents an error analysis of this IDCT realization. Section 5 concludes the paper with a summary.

[1] Now with ARC International.

2. FAST 8-POINT IDCT USING COMPLEX MULTIPLY

An 8-point 1D IDCT is defined as

$$x_n = \frac{1}{2}\sum_{k=0}^{7} u_k y_k \cos\frac{(2n+1)k\pi}{16} \qquad \text{for n = 0,..,7}$$

where $u_k = \begin{cases} 1/\sqrt{2} & \text{if } k = 0 \\ 1 & \text{otherwise} \end{cases}$

This can also be described in the matrix-vector multiplication form

$$\mathbf{x} = \mathbf{Cy}$$

where

$$\mathbf{C} = \begin{bmatrix} a & l & b & g & a & u & d & v \\ a & g & d & -v & -a & -l & -b & -u \\ a & u & -d & -l & -a & v & b & g \\ a & v & -b & -u & a & g & -d & -l \\ a & -v & -b & u & a & -g & -d & l \\ a & -u & -d & l & -a & -v & b & -g \\ a & -g & d & v & -a & l & -b & u \\ a & -l & b & -g & a & -u & d & -v \end{bmatrix}$$

with

$$a = 0.5\cos(\pi/4) = 1/2\sqrt{2}$$
$$b = 0.5\cos(\pi/8)$$
$$d = 0.5\sin(\pi/8)$$
$$l = 0.5\cos(\pi/16)$$
$$v = 0.5\sin(\pi/16)$$

$$g = 0.5\cos(3\pi/16)$$
$$u = 0.5\sin(3\pi/16)$$

and $$\mathbf{x} = \left| x_0\, x_1\, x_2\, x_3\, x_4\, x_5\, x_6\, x_7 \right|^T$$
$$\mathbf{y} = \left| y_0\, y_1\, y_2\, y_3\, y_4\, y_5\, y_6\, y_7 \right|^T$$

Using a variant of the odd-even decomposition technique, the transform matrix **C** can be factorized as

$$\mathbf{C} = \mathbf{AMP}$$

where

$$\mathbf{A} = \begin{vmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 1 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$\mathbf{M} = \begin{vmatrix} a & -a & 0 & 0 & 0 & 0 & 0 & 0 \\ a & a & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b & -d & 0 & 0 & 0 & 0 \\ 0 & 0 & d & b & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & v & -l & 0 & 0 \\ 0 & 0 & 0 & 0 & l & v & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g & -u \\ 0 & 0 & 0 & 0 & 0 & 0 & u & g \\ 0 & 0 & 0 & 0 & g & -u & 0 & 0 \\ 0 & 0 & 0 & 0 & u & g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & l & v \\ 0 & 0 & 0 & 0 & 0 & 0 & -v & l \end{vmatrix}$$

$$\mathbf{P} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Unlike most fast IDCT algorithms, minimizing the number of operation, especially multiplications, is not the only goal in this factorization. Equally important to our efficient IDCT realization are the regular structures of the above matrices. Firstly, we found the input data re-shuffling required by the matrix **P** is fairly simple to implement, as explained in the next section. Secondly, we observed that the matrix-vector multiplication

$$\begin{vmatrix} p \\ q \end{vmatrix} = \begin{vmatrix} g & -u \\ u & g \end{vmatrix} \begin{vmatrix} r \\ s \end{vmatrix}$$

is equivalent to the complex multiplication

$$p + iq = (g + iu)(r + is)$$

in the sense that both produce

$$p = gr - us$$
$$q = ur + gs$$

Therefore, multiplication by **M** can be implemented using 6 complex multiplications. Lastly, the matrix **A** requires only additions and subtractions to complete the IDCT. These are relatively inexpensive to implement in hardware and, whenever feasible, should be done in parallel to minimize the overhead in storing and fetching intermediate results. In the next section, we describe how we exploit these facts in our highly efficient realization of the IDCT on a OneDSP processor.

3. EFFICIENT REALIZATION ON ONEDSP

The OneDSP [4] is a scalable clustered VLIW processor architecture that delivers very high DSP performance while retaining the versatility of traditional RISC processors. It is designed to be a soft IP core to ease integration into system-on-chip applications. Figure 1 is a top-level diagram of the OneDSP architecture. In a OneDSP processor, multiple clusters work in lock-step as a highly parallel VLIW machine. Being a soft core, the number of clusters in a processor can be chosen by the system designer to match the performance requirement. Each cluster has its own local memory and register files to ensure that aggregate data bandwidth increases with the number of cluster. Within each cluster, two instructions are issued in parallel to two functional units. The address generation unit (AGU) performs all load/store operations and some simple arithmetic operations. The execution unit (EXU) is the main computation unit and is capable of sub-word SIMD processing. The EXU has a fully pipelined fixed-point multiplier that can compute in every cycle one 32x32 integer or fractional multiplication or two 16x16 integer or fractional multiplications. It can also compute a complex multiplication by treating the upper 16-bit half-word of a 32-bit data word as the real part and the lower

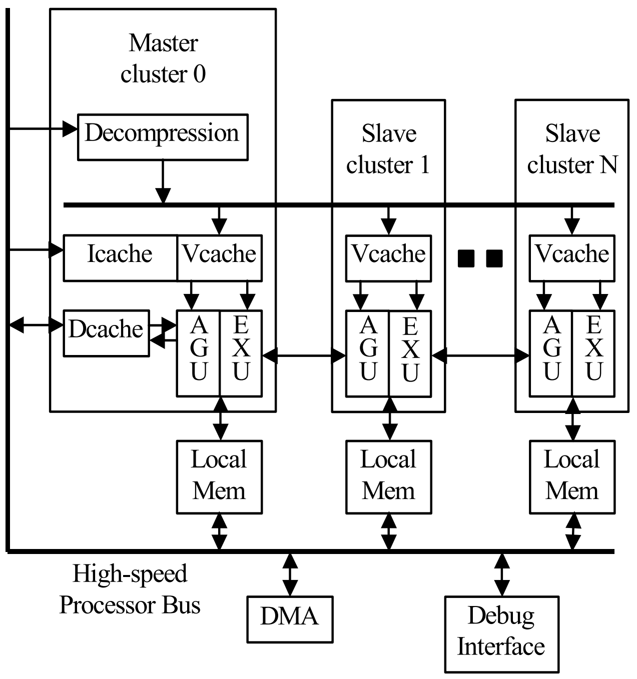

Figure 1 OneDSP Architecture.

half-word as the imaginary part. There are up to 4 accumulators that can add or subtract the multiplier output from their current values. Such datapath elements are ideally suited to realize the fast algorithm described in section 2.

To compute an 8-point IDCT, we assume the inputs y_k are each 16-bit wide and are arranged in memory in ascending order. When loaded as 32-bit words into 4 registers, they can be treated as complex numbers ($y_1 + iy_0$), ($y_3 + iy_2$), ($y_5 + iy_4$) and ($y_7 + iy_6$). The OneDSP instruction `exch` takes values from two 32-bit registers, say ($y_1 + iy_0$) and ($y_5 + iy_4$), and writes back two 32-bit registers the values ($y_1 + iy_5$) and ($y_0 + iy_4$). The data re-ordering described by **P** is achieved using 3 `exch` to re-shuffle the data as follows

$$[(y_1 + iy_5),(y_0 + iy_4)] \leftarrow [(y_1 + iy_0),(y_5 + iy_4)]$$
$$[(y_7 + iy_3),(y_6 + iy_2)] \leftarrow [(y_7 + iy_6),(y_3 + iy_2)]$$
$$[(y_1 + iy_7),(y_5 + iy_3)] \leftarrow [(y_1 + iy_5),(y_7 + iy_3)]$$

To create the matrix **M** requires the following complex multiplications to be performed

$$t_0 + it_1 = (a + ia)(y_0 + iy_4)$$
$$t_2 + it_3 = (b + id)(y_6 + iy_2)$$
$$t_4 + it_5 = (v + il)(y_1 + iy_7)$$
$$t_6 + it_7 = (g + iu)(y_5 + iy_3)$$
$$t_8 + it_9 = (g + iu)(y_1 + iy_7)$$
$$t_{10} + it_{11} = (l - iv)(y_5 + iy_3)$$

Subsequently, the following additions or subtractions described in **A** are required to complete the IDCT

$$\begin{array}{lc|c|c|c|c|c|c|}
x_0 = & t_1 & +t_3 & +t_5 & +t_7 & & \\
x_1 = & t_0 & -t_2 & & & +t_8 & -t_{10} \\
x_2 = & t_0 & +t_2 & & & +t_9 & -t_{11} \\
x_3 = & t_1 & -t_3 & +t_4 & +t_6 & & \\
x_4 = & t_1 & -t_3 & -t_4 & -t_6 & & \\
x_5 = & t_0 & +t_2 & & & -t_9 & +t_{11} \\
x_6 = & t_0 & -t_2 & & & -t_8 & +t_{10} \\
x_7 = & t_1 & +t_3 & -t_5 & -t_7 & &
\end{array}$$

These operations are divided into 6 groups, as shown by the dashed boxes above, with each group representing the results of one complex multiplication. The sign of each t_i indicates whether it is to be added to or subtracted from the corresponding x_k. OneDSP has 6 application specific instructions that can be optionally included in an implementation to improve IDCT performance. These are `mtx8f_0`, `mtx8f_1`, `mtx8f_2`, `mtx8f_3`, `mtx8f_4` and `mtx8f_5`. Each of these instructions performs a fractional complex multiply and then optionally loads, adds or subtracts the product to the 4 accumulators as described above. In these instructions, the upper and lower halves of the accumulators operate independently. Each half-accumulator is 22-bits wide, although this IDCT never requires more than 16-bit significant bits during accumulation. This is due in part to the unbiased rounding of each component of the complex result after summation of the individual fractional products. This is a particularly useful aspect of the OneDSP complex multiplier which helps to keep the accumulated errors under control.

On completion of the six transform instructions, the IDCT outputs are held in pairs in the accumulators as ($x_1 + ix_0$), ($x_3 + ix_2$), ($x_5 + ix_4$) and ($x_7 + ix_6$). These can be stored as 32-bit words to memory with the IDCT outputs arranged in the normal ascending order.

Figure 2 is the listing of an optimized software realization of the IDCT in a single cluster using these instructions. The transform is computed in a tight loop with a 3-stage software pipeline. The first line is the loop instruction that starts a zero-overhead loop with the rest of the listing being the loop body. Inside the loop body, each line is a packet of 2 instructions executed in parallel. The one on the left is an AGU instruction while the right one

```
 :    loop $ic, L, EIC
 :$p4 sw ($s++),$x10; $p3 exch $y15,$y10,$y54
 :$p4 sw ($s++),$x32; $p3 exch $y73,$y76,$y32
 :$p4 sw ($s++),$x54; $p3 exch $y17,$y15,$y73
 :$p4 sw ($s++),$x76; $p3 mtx8f_0 $aa, $y04
 :$p2 lw $y10,($r++); $p3 mtx8f_1 $bd, $y62
 :$p2 lw $y32,($r++); $p3 mtx8f_2 $vl, $y17
 :$p2 lw $y54,($r++); $p3 mtx8f_3 $gu, $y53
 :$p2 lw $y76,($r++); $p3 mtx8f_4 $gu, $y17
L:    nop          ; $p3 mtx8f_5 $l_v, $y53
```

Figure 2 Highly optimized IDCT implementation.

is an EXU instruction. Every instruction is guarded by a shifting predicate to allow the epilogue and the prologue phases of the software pipeline to be embedded into the loop body. The first stage of the software pipeline, as guarded by `$p2`, consists of 4 load word instructions loading the input data into registers. Symbolic names are used with the convention that, for example, `$y10` is the register holding the value $(y_1 + iy_0)$, `$aa` is the register holding the value $(a + ia)$, `$l_v` is the register holding the value $(l - iv)$, etc. The 2nd pipeline stage, as guarded by `$p3`, consists of the `exch` and transform instructions computing the IDCT. Note that the destination of `exch`, for example, `$y15` actually specifies a register pair written with the values $(y_1 + iy_5)$ and $(y_0 + iy_4)$. The transform instructions, `mtx8f_0` to `mtx8f_5`, only specify the source operands with the destination accumulators specified implicitly. The 3rd pipeline stage, as guarded by `$p4`, contains the store word instructions storing the IDCT results in the accumulators into memory. The loop body consists of 9 packets with one packet issued per cycle. Hence, an IDCT is computed every 9 cycles in every cluster.

4. ERROR ANALYSIS

This algorithm has been implemented using 16-bit input coefficients presented in a 12Q4 format. The complex multiply operation performs two 16x16 fractional multiplications for each of the real and imaginary components. The two products are added in 32-bit precision and then rounded to 16-bit values, retaining the most significant bits. The 16-bit real and imaginary components are then added or subtracted from nominated half-accumulators, which retain 22-bit precision. The following table summarizes the output from the IEEE1180 IDCT error compliance test.

5. SUMMARY

In this paper we have shown how an IEEE IDCT can be implemented using a novel factorization of the IDCT designed to exploit the complex multiplication capability provided by the OneDSP processor. This has been implemented using the OneDSP C compiler, augmented with compiler intrinsic functions to provide access to the non-C operators such as complex multiplication and accumulation. An optimal software pipeline schedule generated by the OneDSP C compiler was used to run the IEEE compliance tests. The results were well within the specifications, due partly to the provision of unbiased rounding as part of the fractional complex multiplication instruction. The 9-cycle loop schedule that we have presented for the IDCT contains only 18 instructions but performs a total of 72 individual arithmetic operations or memory references. This represents an exceptionally efficient encoding.

A single-cluster OneDSP synthesizable processor core implemented in 0.13um CMOS technology occupies less than 2 sq.mm of silicon and has power consumption of less than 0.2 mW per MHz, including cache memories. When operating at 300 MHz such a processor core is capable of decoding MPEG2 bit-streams at ATSC resolution.

6. REFERENCES

[1] *IEEE Standard Specifications for the implementation of 8x8 inverse discrete cosine transform*, IEEE Std. 1180-1990.

[2] Loeffler, C., A. Lightenberg and G.S. Moschytz, "Practical fast 1-D DCT Algorithms with 11 multiplications" *Proc. ICASSP 1989*, pp. 988-991, 1989.

[3] "Using streaming SIMD extensions in a fast DCT algorithm for MPEG encoding" *Application Note Ap-817*, Intel, 1999.

[4] Topham, N., "An Integrated DSP Architecture", *Microprocessor Forum*, October, 2001. (http://www.siroyan.com/pdf/SR_MPF01.pdf)

Test range	Sign	Worst peak error	PMSE		Mean error	
			Worst	Overall	Worst	Overall
-256..+255	+	1	0.0145	0.010486	0.0026	-0.000233
-5..+5	+	1	0.0127	0.010580	0.0022	-0.000039
-300..+300	+	1	0.0116	0.009100	0.0022	0.000009
-256..+255	-	1	0.0145	0.010473	0.0026	0.000223
-5..+5	-	1	0.0127	0.010580	0.0022	0.000039
-300..+300	-	1	0.0116	0.009100	0.0022	-0.000025

Table 1. IEEE 1180 error analysis of OneDSP IDCT