รายการอ้างอิง

ภาษาอังกฤษ

Bellanger, Maurice. Digital Processing Of Signals Theory And Practice. John Wiley & Sons, 1980

Burrus, C.S., and Parks, T.W. DFT/FFT and Convolution Algorithms Theory and Implementation. John Wiley & Sons, 1985

Despain, A.M. Fourier Transform Computes Using CORDIC Iterations. IEEE Transactions on Signal Processing. (July 1973): 993-1001.

Fettweis, G., Chiu, J., and Fraenkel, B. A Low-complexity bit-serial DCT/IDCT Architecture. IEEE International Conference on Communications, (February 1993): 217-221.

Haberman, Werner. HDTV Standards. IEEE Communications Magazine, (August 1991): 10-12.

Hopkins, Robert. Digital HDTV Broadcasting. IEEE Transactions on Broadcasting, 37(December 1991): 123-127.

Karathanasis, H. C. A Low ROM Distributed Arithmetic Implementation of the Forward/Inverse DCT/IDCT using rotation. IEEE Transactions on Consumer Electronics, (May 1995): 263-271.

Kim, C. S., Song, S. W., Kim, M. Y., Han, Y. T., Kang, S. A., and Lee, B. W. 200 Mega Pixel Rate IDCT Processor for HDTV Applications. IEEE International Symposium on Circuits and Systems, (June 1993): 2003-2006.

Lim, Jae S. Two-Dimensional Signal and Image Processing. Prentice-Hall International Edition, 1991

Pennebaker, W. B., and Mitchell, J. L. JPEG Still Image Data Compression Standard, New York: Van Nostrand Reinhold. 1991

Perle, M. D. Cordic Technique Reduces Trigonometric Function Look-Up. Computer Design, (June 1971): 72-77.

Van Den Enden, A.W.M., and Ver Hoeckx, N.A.M. Discrete-Time Signal Processing. Prentice-Hall International Edition, 1989

Vetterli, M., and Ligtenberg, A. A Discrete Fourier-Cosine Transform Chip IEEE J. Select. Areas Communication, (Jan 1986): 245-254.

Zhou, F., and Kornerup, P. High Speed DCT/IDCT Using a Pipeline CORDIC Algorithm. IEEE Transactions on Signal Processing, (1995): 180-187.

ภาคผนวก

ภาคผนวก ก

การใช้งาน Verilog HDL

การใช้ภาษาที่อธิบายโครงสร้างของฮาร์ดแวร์สำหรับการออกแบบวงจรตรรก ได้รับความนิยมเพิ่มมากขึ้นเรื่อยๆ ในระยะเวลาหลายปีที่ผ่านมา นักออกแบบมักออกแบบตามหน้าที่การทำงานหรือตามคุณสมบัติซึ่งในขั้นตอนนี้ยังไม่ต้องสนใจในรายละเอียด การออกแบบในแนวนี้จะทำให้เห็นถึงภาพรวม และเห็นข้อบกพร่องต่างๆ ก่อนที่จะทำการออกแบบในรายละเอียด เหตุผลที่ควรใช้ภาษาที่อธิบายโครงสร้างของฮาร์ดแวร์ เนื่องจากพัฒนาการทางด้านเทคโนโลยีทั้งความจุของชิปและความซับซ้อนในการออกแบบมีมากขึ้นทุกทีเกินกว่าจะทำด้วยมือ การออกแบบในระดับสูงจะซ่อนในส่วนรายละเอียดเพื่อมองในภาพรวม ด้วยเหตุผลเดียวกันที่ปัจจุบันในระบบใหญ่ๆ ภาษาระดับสูงจึงได้เข้ามาแทนที่ภาษาแอสเซมบลีซึ่งเป็นภาษาในระดับล่าง

อุตสาหกรรมอิเล็กทรอนิกส์ในปัจจุบันได้เติบโตและมีการแข่งขันกันสูง จึงต้องมีการเพิ่มประสิทธิภาพในการออกแบบ ลดค่าใช้จ่ายในการออกแบบ และสิ่งที่สำคัญคือเรื่องเวลา ด้วยโปรแกรมจำลองจะช่วยตรวจจับความผิดพลาดของการออกแบบก่อนที่ทำการผลิต ซึ่งเป็นการประหยัดเวลาและช่วยลดขั้นตอนการออกแบบและการทดลอง

การออกแบบด้วย Verilog HDL

คำว่า "Verilog" หมายถึงทั้ง ภาษา (Language) และ ตัวจำลอง (Simulator) ซึ่งใช้อธิบายแบบจำลองดิจิตอล Verilog HDL (Hardware Description Language) คิดค้นโดย Gateway Design Automation และต่อมาบริษัท Cadence Design System ได้นำมาเผยแพร่ เพื่อเป็นการสนับสนุนให้ Verilog เกิดเป็นภาษามาตรฐานสำหรับการออกแบบ แบบจำลองของ Verilog คล้ายกับการโปรแกรม การกำหนดและแบ่งออกเป็นส่วยย่อยๆเรียกว่า "มอดูล" มอดูลคล้ายกับรูทีนย่อย ซึ่งสามารถที่จะเขียนขึ้นมาครั้งเดียว แล้วเรียกใช้จากหลายๆที่ เราสามารถที่จะประกอบมอดูลเป็นลำดับชั้น มอดูลที่อยู่ในระดับต่ำจะมีส่วนของอินพุตและเอาต์พุต ที่ทำหน้าที่เหมือนกับเป็นพารามิเตอร์ของกระบวนคำสั่ง มอดูลในระดับสูงจะประกอบขึ้นจากมอดูลต่ำเหล่านี้และเชื่อมต่ออินพุตและเอาต์พุตด้วยสายสัญญาณ มอดูลอาจเป็นเพียงแค่เกต วงจรตรวจนับ ระบบหน่วยความจำ ระบบคอมพิวเตอร์ หรือเครือข่ายคอมพิวเตอร์ พอร์ตของมอดูลอาจเป็นข้อมูลบิตเดียวหรือหลายบิตก็ได้ แต่ละพอร์ตอาจจะเป็นอินพุต เอาต์พุตหรือเป็นทั้งอินพุตและเอาต์พุต เราสามารถที่จะเขียนมอดูลได้หลายแบบคือ

1) Structural เป็นการอธิบายถึงองค์ประกอบ หรือการประกอบกันของเกตพื้นฐาน
2) Data Flow อธิบายวิธีที่แบบจำลองประกอบขึ้นมา ถ้าเปรียบกับต้นไม้ ส่วนรากคือเอาต์พุต เมื่ออินพุตมีการเปลี่ยนแปลง จะมีผลให้เอาต์พุตเปลี่ยนไปด้วย

```
module AND2 (out, in1, in2);
        input    in1, in2;
        output   out;
        wire     in1, in2, out;
        and      u1(out, in1, in2);
endmodule
```

ตัวอย่างที่ ก.1 การเขียนมอดูลแบบ Stuctural

```
module AND2 (out, in1, in2);
        input    in1, in2;
        output   out;
        wire     in1, in2, out;
        assign   out = in1 & in2;
endmodule
```

ตัวอย่างที่ ก.2 การเขียนมอดูลแบบ Data Flow

3) Behavioral เป็นวิธีการอธิบายด้วยภาษาระดับสูง ซึ่งบอกถึงคณลักษณะหรือหน้าที่การทำงาน แต่ไม่ได้ระบุในส่วนของรายละเอียด

```
module AND2 (out, in1, in2);
        input    in1, in2;
        output   out;
        wire     in1, in2;
        reg      out;
        always @(n1 or in2)
                 out = in1 & in2;
endmodule
```

ตัวอย่างที่ ก.3 การเขียนมอดูลแบบ Behavioral

มอดูลของ Verilog ประกอบไปด้วยคำรหัส ชื่อ หมายเหตุ และคำสั่งต่างๆ แต่ละมอดูลจะเริ่มต้นด้วยคำรหัส *module* ตามด้วยชื่อมอดูล และส่วนของอินพุตและเอาต์พุต ส่วนท้ายของมอดูลจะปิดด้วยคำว่า *endmodule* เราสามารถกำหนดชนิดของอินพุตและเอาต์พุตว่าเป็น *wire* หรือ *reg* ถ้าไม่มีการระบุจะถือว่าเป็น wire ซึ่ง wire หมายถึงสายสัญญาณ แต่ถ้าเป็น reg จะสามารถเก็บค่าไว้ได้จนกว่าจะมีการบรรจุค่าใหม่เข้าไป

การทดสอบการทำงาน

การเขียนมอดูล Verilog คล้ายการเขียนโปรแกรม ที่เป็นส่วนเล็กๆของขั้นตอนการออกแบบ การออกแบบต้องมีการทดสอบและการพิสูจน์ โดยการป้อนอินพุตและดูเอาต์พุตเพื่อเปรียบเทียบกับค่าที่ควรจะเป็น มอดูลที่ใช้ทดสอบจะเป็นมอดูลระดับสูงที่ไม่มีอินพุตและเอาต์พุตดังตัวอย่างโปรแกรมในตัวอย่างที่ ก.4

```
module test_and2;
        reg     i1, i2;
        wire    o;
        AND2 u2(o, i1, i2);
        initial begin
                i1 = 0; i2 = 0;
                #1 $display("i1 = %b, i2 = %b o = %b",i1, i2, 0);
                i1 = 0; i2 = 1;
                #1 $display("i1 = %b, i2 = %b o = %b" ,i1, i2, 0);
                i1 = 1; i2 = 0;
                #1 $display("i1 = %b, i2 = %b o = %b" ,i1, i2, 0);
                i1 = 1; i2 = 1;
                #1 $display("i1 = %b, i2 = %b o = %b" ,i1, i2, 0);
        end
endmodule
```

ตัวอย่างที่ ก.4 โปรแกรมทดสอบมอดูล AND

คำสั่ง $display จะมีผลให้มีการแสดงค่าตัวแปร i1 i2 และ o ในระบบเลขฐานสอง (%b) ซึ่งผลลัพธ์จากโปรแกรมทดสอบนี้ได้แสดงไว้ในตัวอย่างที่ ก.5

```
I1 = 0, i2 = 0, o = 0
I1 = 0, i2 = 1, o = 0
I1 = 1, i2 = 0, o = 0
I1 = 1, i2 = 1, o = 1
```

ตัวอย่างที่ ก.5 ผลลัพธ์จากมอดูลทดสอบ

<u>ชนิดของข้อมูล</u>

เช่นเดียวกับโปรแกรมภาษาโดยทั่วไป Verilog สนับสนุนค่าคงตัวที่เก็บค่าคงที่ และตัวแปรซึ่งจะถูกใช้งานในระหว่างการจำลอง ค่าคงตัวอาจจะเป็นเลขฐานสิบ เลขฐานสิบหก เลขฐานแปด หรือเลขฐานสอง และมีรูปแบบเป็นดังนี้

<width>'<radix><value>

โดย width คือขนาดของค่าคงตัวเป็นเลขจำนวนเต็ม ส่วน radix อาจจะเป็น b หรือ B หมายถึงเลขฐานสอง d หรือ D หมายถึงเลขฐานสิบ o หรือ O หมายถึงเลขฐานแปด และ h หรือ H หมายถึงเลขฐานสิบหก ถ้าไม่ระบุ radix จะถือเป็นเลขฐานสิบ value เป็นค่าแอสกีขึ้นกับว่า radix จะเป็นค่าอะไร ถ้าเป็น b หรือ B value อาจจะเป็น 0 1 x X z หรือ Z ถ้า radix เป็น o หรือ O radix สามารถเป็น 2 3 4 5 6 หรือ 7 ได้อีก ถ้า radix เป็น h หรือ H radix สามารถเป็น 8 9 a A b B c C d D e E f หรือ F แต่ถ้า radix เป็น d หรือ D radix เป็นได้เพียงเลข 0-9 เท่านั้น ดังตัวอย่าง

```
15          ( เลข 15 ฐานสิบ )
'h15        ( เลข 21 ฐานสิบ เลข 15 ฐานสิบหก)
5'b1001     ( เลข 19 ฐานสิบ เลข 1001 ฐานสอง)
12'h01f     ( เลข 31 ฐานสิบ เลข 01F ฐานสิบหก)
'b01x       ( ไม่มีฐานสิบ เลข 01x ฐานสอง)
```

<u>การประกาศ</u>

ตัวแปรจะต้องประกาศก่อนการใช้ การประกาศเป็นการบอกถึงชนิดและขนาดของตัวแปร แถวลำดับเป็นโครงสร้างข้อมูลแบบเดียวที่ใช้ใน Verilog ตัวอย่างการประกาศ

```
integer     I, j;           // เลขจำนวนเต็มสองตัว
real        f, d;           // เลขจำนวนจริงสองตัว
```

```
wire      [7:0]    bus;                  // บัสมีความกว้าง 8 บิต
reg       [0:15]   word;                 // แถวลำดับทั้ง 16 ของรีจิสเตอร์
event              trigger, clock_high;  // เหตุการณ์สองเหตุการณ์
parameter          width=8;              // กำหนดค่า 8 ให้กับตัวแปร width
```

ตัวดำเนินการ

+ - * /	เครื่องหมายคำนวณ
>= < <=	ความสัมพันธ์
! && ||	การดำเนินการเชิงตรรก
== !=	ความเท่ากันเชิงตรรก
?:	เงื่อนไข
{ }	ต่อกัน
%	มอดูลัส
=== !==	เปรียบเทียบ
~ & |	ปฏิบัติการบิต
<< >>	การเลื่อนบิต

ข้อความสั่งกระบวนงาน

นิพจน์สามารถใช้ในการคำนวณ แต่ไม่สามารถเขียนขึ้นมาลอยๆ จะต้องเขียนเป็นส่วนหนึ่งของข้อความสั่งหรือโครงสร้างอื่น คำสั่งที่ง่ายๆ อาจจะเป็นข้อความสั่งกำหนดค่า (assignment statement) หรือการควบคุมสายงาน (control flow) พวกกลุ่มของคำสั่งจะอยู่ระหว่างคำรหัส *begin* และ *end* หรือ *fork* และ *join* ซึ่งการทำงานจะเรียงตามลำดับสำหรับ *begin-end* แต่จะทำงานไปพร้อมๆกันทุกคำสั่งสำหรับข้อความสั่ง *fork-join*

ข้อความสั่งกำหนดค่า

จะเป็นหนึ่งในสามของรูปแบบต่อไปนี้

```
lhs-expression = expression;
lhs-expression = #delay expression;
lhs-expression = @event expression;
```

ฝั่งซ้ายจะเป็นชื่อตัวแปร หรืออาจจะเป็นช่วงข้อมูลบิตที่ถูกเลือก ค่าทางขวาจะถูกกำหนดให้กับค่าทางฝั่งซ้ายในกรณีที่หนึ่งทันที กรณีที่สองรอ delay กรณีที่สามรอจนกว่าเหตุการณ์ที่ระบุจะเกิดขึ้น

ตัวอย่างข้อความสั่งของการทำซ้ำ

```
module for_loop;
        integer  I;
        for(I = 0; I < 4; I  I + 1) begin
                $display ("I = %0d (%b binary)", I, I));
        end
endmodule
```

ตัวอย่างที่ ก.6 A for loop

```
I = 0 ( 0 binary)
I = 1 ( 1 binary)
I = 2 ( 10 binary)
I = 3 ( 11 binary)
```

ตัวอย่างที่ ก.7 ผลลัพธ์จากข้อความสั่ง for loop

```
module while_loop;
        integer  I;
        initial begin
                I = 0;
        while ( I < 4)  begin
                $display ("I = %0d (%b binary)", I, I));
                I = I + 1;
        end
endmodule
```

ตัวอย่างที่ ก.9 A while loop

```
module case_statement;
        integer  I;
        initial    I = 0;
        always begin
                $display ("I = %0d (%b binary)", I, I));
                case ( I )
                        0: I = I + 2;
                        1: I = I + 7;
                        2: I = I - 1;
                        default: $stop;
                endcase
        end
endmodule
```

ตัวอย่างที่ ก.10 A case statement

```
I = 0
I = 2
I = 1
I = 8
```

ตัวอย่างที่ ก.11 ผลลัพธ์จากข้อความสั่ง case

```
module repeat_loop (clock);
        input    clock;
        initial begin
                repeat ( 5 )
                @ (posedge clock);
                $stop;
        end
endmodule
```

ตัวอย่างที่ ก.12 A repeat loop

```
module forever_statement (a,b,c);
        input    a,b,c;
        initial forever begin
                @( a or b or c)
                if (a + b == c) begin
                        $display ("a(%d)+b(%d) = c(%d)",a,b,c);
                        $stop;
                end
        end
endmodule
```

ตัวอย่างที่ ก.13 A forever Loop

<u>การควบคุมเวลาและเหตุการณ์</u>

กระบวนการของ Verilog (เช่น บล็อกของ initial หรือ always) สามารถลำดับใหม่โดยรูปแบบการควบคุมด้วยเวลา 3 รูปแบบต่อไปนี้

#expression	หยุดรอเวลาหน่วยค่าคงที่
@even-expression	หยุดรอจนกว่าเหตุการณ์ที่ระบุจะเกิดขึ้น
wait (expression)	หยุดรอจนกว่านิพจน์นั้นเป็นจริง

ภาคผนวก ข

เทคนิคคำนวณ 1-D DCT อย่างเร็ว โดย Vetterli และ Ligtenberg

จากสมการคำนวณ DCT แบบ 1 มิติ

$$S(u) = \frac{C(u)}{2}\sum_{x=0}^{7} s(x)\cos\left(\frac{(2x+1)u\pi}{16}\right) \quad \text{.....(1)}$$

ถ้าคำนวณตามสูตรของ DCT เราต้องการใช้การคูณ 64 ครั้ง และใช้การบวก 56 ครั้ง สำหรับทุกๆ 1-D DCT ขนาด 8 จุด โดยที่สัมประสิทธิ์โคไซน์จะถูกคำนวณเอาไว้ก่อน เทคนิคการคำนวณของ Vetterli และ Ligtenberg ใช้ข้อดีของความสมดุลสมการ DCT ทำให้สามารถลดจำนวนของการปฏิบัติการลงได้ สมการสามารถแสดงในรูปผลรวมและผลต่างของ samples โดยกำหนดค่าดังต่อไปนี้

$C_k = \cos\left(\frac{k\pi}{16}\right)$

$S_k = \sin\left(\frac{k\pi}{16}\right)$

$C_1 = S_7 = 0.9808$

$C_2 = S_6 = 0.9239$

$C_3 = S_5 = 0.8315$

$C_4 = S_4 = 0.7071$

$C_5 = S_3 = 0.5556$

$C_6 = S_2 = 0.3827$

$C_7 = S_1 = 0.1951$

Sums: $s_{jk} = s(j)+s(k)$

$s_{07} = s(0)+s(7)$

$s_{16} = s(1)+s(6)$

$s_{25} = s(2)+s(5)$

$s_{34} = s(3)+s(4)$

$s_{0734} = s_{07}+s_{34}$

$s_{1625} = s_{16}+s_{25}$

Differences:

$d_{jk} = s(j)-s(k)$

$d_{07} = s(0)-s(7)$

$d_{16} = s(1)-s(6)$

$d_{25} = s(2)-s(5)$

$d_{34} = s(3)-s(4)$

$d_{0734} = s_{07}-s_{34}$

$d_{1625} = s_{16}-s_{25}$

สมการ DCT สามารถแสดงได้ดังนี้

$2S(0) = C_4(s_{0734}+s_{1625})$ ..... (2)

$2S(1) = C_1d_{07}+C_3d_{16}+C_5d_{25}+C_7d_{34}$ ..... (3)

$2S(2) = C_2d_{0734}+C_6d_{1625}$ ..... (4)

$2S(3) = C_3d_{07}-C_7d_{16}-C_1d_{25}-C_5d_{34}$ ..... (5)

$2S(4) = C_4(s_{0734}-s_{1625})$ ..... (6)

$2S(5) = C_5d_{07}-C_1d_{16}+C_7d_{25}+C_3d_{34}$ ..... (7)

$2S(6) = C_6d_{0734}-C_3d_{1625}$ ..... (8)

$2S(7) = C_7d_{07}-C_5d_{16}+C_3d_{25}-C_1d_{34}$ ..... (9)

สมการเหล่านี้เป็นการจัดเขียนขึ้นใหม่เพื่อแสดงถึง การรวมหรือผลต่างระหว่างค่าตัวอย่าง ที่เกิดขึ้นมากว่า 1 ครั้ง จำนวนปฏิบัติการทั้งหมดที่นับได้เท่ากับการคูณ 22 ครั้ง กับการบวก 28 ครั้ง

Ligtenberg and Vetterli ได้เสนอขั้นตอนวิธีการคำนวณ 1-D DCT อย่างเร็ว ที่ใช้จำนวนปฏิบัติการน้อยกว่านี้ โดยขั้นแรกเขียนสมการเดิมแต่มีการการจัดกลุ่มใหม่

$2S(0) = C_4((s_{07}+s_{12})-(s_{34}+s_{56}))$ ..... (10)

$2S(1) = C_1d_{07}+C_3d_{16}+C_5d_{25}+C_7d_{34}$ ..... (11)

$2S(2) = C_2(s_{07}-s_{34})+C_6(d_{12}-d_{56})$ ..... (12)

$2S(3) = C_3d_{07}-(C_5d_{34}+C_7d_{16}+C_1d_{25})$ ..... (13)

$2S(4) = C_4((s_{07}+s_{34})-(s_{12}+s_{56}))$ ..... (14)

$2S(5) = C_5d_{07}+C_7d_{25}+C_3d_{34}-C_1d_{16}$ ..... (15)

$2S(6) = C_6(s_{07}-s_{34})-C_3(s_{25}-s_{16})$ ..... (16)

$2S(7) = C_7d_{07}+C_3d_{25}-(C_5d_{16}+C_1d_{34})$ ..... (17)

ขั้นตอนวิธีใช้ปฏิบัติการการพื้นฐาน เรียกว่าการหมุน (rotation) โดยการหมุนของอินพุต x,y ไปเป็นมุม Q จะเกิดเอาท์พุต X ,Y โดยค่าของ X และ Y เป็นไปตามสมการ

$X = x\cos(Q) + y\sin(Q)$

$Y = -x\cos(Q) + y\cos(Q)$

ถ้ามุมที่หมุนเป็นมุม $\left(\frac{k\pi}{16}\right)$ เราได้

$X \quad = C_k x + S_k y$

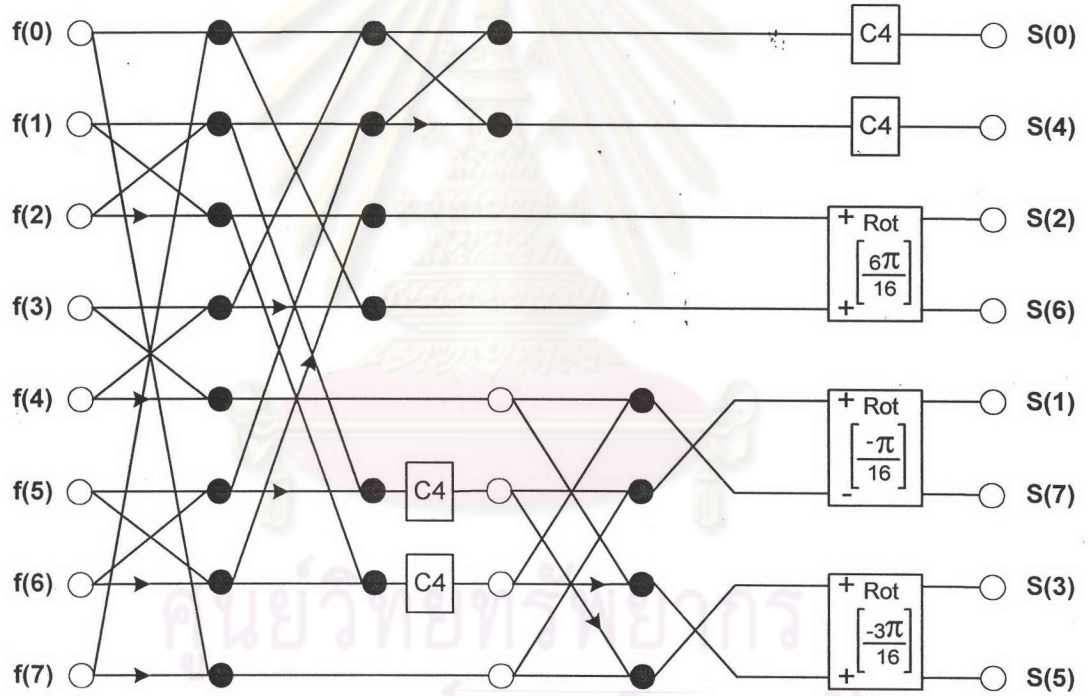
$Y \quad = -S_k x + C_k y$

สำหรับการหมุนเราต้องการการคูณ 4 ครั้งและการบวกอีก 2 ครั้ง แต่เนื่องจาก วงจรการคูณมีขนาดใหญ่กว่าการบวก เราต้องปรับปรุงโดย

$X \quad = C_k x + S_k y + C_k y - C_k y \quad = C_k(x+y) + (S_k - C_k)y$

$Y \quad = -S_k x + C_k y + C_k x - C_k x \quad = -(S_k + C_k)x + C_k(x+y)$

เราสามารถลดปฏิบัติการการคูณเหลือเพียง 3 ครั้งและการบวกเพิ่มเป็น 3 ครั้ง โดย $(S_k - C_k)$ และ $(S_k + C_k)$ เป็นค่าคงที่ การหมุนอาจจะเกิดขึ้นไปในทิศทางลบหรือบวกก็ได้



รูปที่ ข.1 ผังการคำนวณ 1-D DCT อย่างเร็วของ Vetterli และ Ligtenberg

รูปที่ ข.1 แสดงถึงผังการคำนวณสำหรับ 1-D DCT อย่างเร็ว ของ Ligtenberg และ Vetterli จากกราฟปฏิบัติการจะเริ่มจากซ้ายไปขวา และเส้นที่โยงระหว่างข้อหมายถึงการรวม แต่ถ้ามีเครื่องหมายลูกศรอยู่ในเส้นหมายถึง มีการเปลี่ยนเครื่องหมายเป็นลบก่อนการรวม การคูณของสัญญาณเช่น บล็อกของ C_4 ในแผนผังบล็อกการหมุนการกระทำที่บล็อกจะเขียนว่า “Rot” ถ้าเครื่องหมายเป็น “+” หมายถึง การหมุนตามเข็มนาฬิกา ถ้าเป็น “-” คือ การหมุนทวนเข็มนาฬิกา

คราวนี้เรามาพิจารณาว่าเส้นทางของกราฟเดินทางอย่างไร สมการ S(0) และ S(4) ใช้ตรงๆ

$2S(0) = C_4((s_{07}+s_{12})+(s_{34}+s_{56}))$ ..... (16)

$2S(4) = C_4((s_{07}+s_{34})-(s_{12}+s_{56}))$ ..... (17)

มีเพียงแค่ผลรวม และคูณกับค่า C_4 (การคูณด้วย 2 ไม่ได้กล่าวถึงในแผนผังบล็อก)

$2S(2) = C_2(s_{07}-s_{34})+C_6(d_{12}-d_{56})$ ..... (18)

$2S(6) = C_6(s_{07}-s_{34})-C_3(s_{25}-s_{16})$ ..... (19)

เนื่องจาก $s_{25}-s_{16} = -(d_{12}-d_{56})$

$2S(2) = C_6(d_{12}-d_{56})+S_6(s_{07}-s_{34})$ ..... (20)

$2S(6) = -S_6(d_{12}-d_{56})+S_6(s_{07}-s_{34})$ ..... (21)

นั่นคือมีการหมุนของอินพุต x,y ไป $\left(\frac{6\pi}{16}\right)$ โดย

$x = d_{12}-d_{56}$

$y = s_{07}-s_{34}$

จากสมการของ S(3) และ S(5)

$2S(3) = C_3d_{07}-(C_5d_{34}+C_7d_{16}+C_1d_{25})$ ..... (22)

$2S(5) = C_5d_{07}+C_7d_{25}+C_3d_{34}-C_1d_{16}$ ..... (23)

เขียนใหม่เป็น

$2S(3) = [C_3d_{07}-S_3d_{34}] - [S_1d_{16}+C_1d_{25}]$ ..... (24)

ต้องการเปลี่ยนนิพจน์ที่ 2 ให้อยู่ในรูปของ s_{12} d_{12} s_{56} และ d_{56} โดยกำหนดสมการและติดค่าสัมประสิทธิ์ A B C และ D ดังสมการ

$S_1d_{16}+C_1d_{25} = As_{12}+Bd_{12}+Cs_{56}+Dd_{56}$ ..... (25)

เราแก้สมการโดยเขียนให้อยู่ในนิพจน์ของ s(1) s(2) s(5) และ s(6) ดังนี้

For s(1): $S_1 = A + B$

For s(2): $C_1 = A - B$

For s(6): $-S_1 = C - D$

For s(5): $-C_1 = C + D$

จาก s(1) และ s(2)

$S_1 - B = C_1 + B$

$2B = S_1 - C_1$

$= C_7 - C_1$ (cos A - cos B = -2 sin (sum) sin (diff))

$= -2S_4S_3$

$B = -S_4S_3 = -C_4S_3$

$A = C_4C_3$

$C = -C_4C_3$

$D = -C_4S_3$ (A C และ D หาได้ในทำนองเดียวกันกับ B)

จากสมการ (25) เมื่อแทนค่า A B C และ D ลงไป

$S_1d_{16}+C_1d_{25} = C_4[C_3(s_{12}- s_{56})- S_3(d_{12}+d_{56})]$ ..... (26)

กระทำในแนวเดิมกับ S(5)

$2S(5) = C_5d_{07}+C_7d_{25}+C_3d_{34}-C_1d_{16}$ ..... (27)

$2S(5) = [S_3d_{07}+C_3d_{34}]+[S_1d_{25}-C_1d_{16}]$ ..... (28)

$S_1d_{25}-C_1d_{16} = As_{12}+Bd_{12}+Cs_{56}+Dd_{56}$ ..... (29)

$A = -C_4S_3$

$B = -C_4C_3$

$C = C_4S_3$

$D = -C_4C_3$

$S_1d_{25}+C_1d_{16} = C_4[-S_3(s_{12}- s_{56})- C_3(d_{12}+d_{56})]$ ..... (30)

สมการสำหรับ S(3) และ S(5) จัดได้ใหม่เป็นดังนี้

$2S(3) = C_3[d_{07}-C_4(s_{12}- s_{56})] - S_3[d_{34}-C_4(d_{12}+d_{56})]$ ..... (31)

$2S(5) = S_3[d_{07}-C_4(s_{12}- s_{56})] + C_3[d_{34}-C_4(d_{12}+d_{56})]$ ..... (32)

หมายถึงมีการหมุนเป็นมุม - $\left(\frac{3\pi}{16}\right)$

$x = d_{07}-C_4(s_{12}- s_{56})$

$y = d_{34}-C_4(d_{12}+d_{56})$

เช่นเดียวกับ S(1) และ S(7) ซึ่งเราจะข้ามขั้นตอนการคำนวณไป เราได้

$$2S(1) = C_1 [d_{07}+C_4 (s_{12}- s_{56})] - S_1 [-d_{34} -C_4 (d_{12}+d_{56})] \quad \text{..... (33)}$$

$$2S(7) = S_1 [d_{07}+C_4 (s_{12}- s_{56})] - C_1 [-d_{34} -C_4 (d_{12}+d_{56})] \quad \text{..... (34)}$$

หมายถึงมีการหมุนเป็นมุม $-\left(\frac{\pi}{16}\right)$

$x = d_{07}+C_4 (s_{12}- s_{56})$

$y = -[d_{34} +C_4 (d_{12}+d_{56})]$

จากแผนผังดังรูป ข.1 นี้เมื่อนับจำนวนการปฏิบัติการ แล้วจะได้การคูณ 13 ครั้ง กับการบวกเพียง 29 ครั้ง เท่ากับว่าการคำนวณใช้ปฏิบัติการเพียง 30% ของการคำนวณแบบปกติ

ภาคผนวก ค

คำอธิบายคำย่อ

ATM	Asynchronous Transfer Mode
ATV	Advanced Television
B-Picture	Bidirectionally Predictive Coded Picture (ภาพเข้ารหัสโดยอาศัยข้อมูลจากภาพในอดีตและภาพในอนาคต)
CLA	Carry Look Ahead (การทดมองล่วงหน้า)
CSA	Carry Save Adder (วงจรบวกประหยัดตัวทด)
DCT	Discrete Cosine Transform
FFT	Fast Fourier Transform
FHT	Fast Hartley Transform
GOP	Group of Pictures (กลุ่มของภาพอยู่ในลำดับที่สองของโครงสร้างกระแสข้อมูล MPEG)
HDTV	High Definition Television
HSV	Hue Saturation Value (ระบบการแสดงสีแบบหนึ่ง)
IDCT	Inverse Discrete Cosine Transform (ส่วนแปลงกลับของ DCT)
IEC	International Electrotechnical Commission
I-Picture	Intra Coded Picture (ภาพที่เข้ารหัสภายใน โดยไม่ต้องอาศัยข้อมูลจากภาพอื่น)
ISDN	Integrated Services Digital Network (โครงข่ายดิจิตอลบริการรวม)
ISO	International Standards Organization (องค์การมาตรฐานระหว่างประเทศ)
JPEG	Joint Pictures Expert Group
NTSC	National Television Standards
MAC	Multiply Accumulator (วงจรสะสมผลคูณ)
MPEG	Moving Pictures Expert Group
MUX	Multiplexer (ตัวมัลติเพลกซ์)
P-Picture	Predictive Coded Picture (ภาพเข้ารหัสโดยอาศัยข้อมูลจากภาพในอดีต)
PP	Partial Product (ผลคูณย่อย)
PPS	Partial Product Sum (ผลรวมผลคูณย่อย)
VLC	Variable Length Coding (การเข้ารหัสความยาวแปรได้)
VLSI	Very Large Scale Integration (วงจรรวมความจุสูงมาก)

ประวัติผู้เขียน

นางสาว นวพร วรรณวิมลศรี เกิดวันที่ 7 ธันวาคม 2513 จังหวัดกรุงเทพมหานคร เข้าศึกษาในระดับปริญญาตรี ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง เมื่อปีการศึกษา 2532 สำเร็จการศึกษาปริญญาวิศวกรรมศาสตร์บัณฑิต เกียรตินิยมอันดับ 2 เมื่อปีการศึกษา 2536 และเข้าทำงานเป็นเจ้าหน้าที่บริหารข้อมูลให้กับ บริษัท อินฟอร์เมชั่น ฟาซิลิตี้ เซอร์วิส จำกัด และในปี พ.ศ. 2537 ได้เข้าทำงานในตำแหน่งวิศวกรระบบ ในบริษัท พาราแอดวานซ์อินโฟเทค จำกัด เข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2538