Group - 31

- Hemanth Kumar J
- J Pradeeshwar
- Yacha Venkata Rakesh
- Deepak Abhiram Inagandle
- Dometti Sree Gnanesh

DCT IN VERILOG HDL

25.11.2019

Abstract

DCT (Discrete Cosine Transform), a widely used lossy image compression technique, implemented in Verilog, a Hardware Description Language, mostly using in designing for FPGA s. The data retainment is approximately 99.95% for this implementation. A python script has been used to complement this process by converting an image into its RGB matrices format that is being fed into the verilog program. No processes related to DCT/IDCT has been implemented in the python script and thus can be modified as per the user's requirement.

DCT can be implemented in 2 ways, a direct approach using a DCT equation and a matrix approach which deals with the manipulation of matrices to achieve similar results. This project follows the latter approach to attain more accuracy in verilog's perspective. All the matrix operations are defined for 8 x 8 matrix as DCT is highly compatible for this specification. Inverse DCT(IDCT)has also been implemented in the similar fashion.

Usage

1. We have chosen to implement the algorithm on a fixed image size of 256*256 pixels which is automatically resized by the python script and ordered correctly according to the required specifications.

2. The verilog code is run for 3.72 µs with each 8 x 8 matrix operated per ns(or 16 integers in the case of IDCT).

Algorithm

1. The image is given as input in 256x256x3 format with each 8x8 per instantiation.
2. T * 10^3 ,Q matrices are attained by a generator written in python and T_ * 10^3(transpose of T) is found in verilog
3. 128 is removed from each integer in the matrix to preserve a mean value of 0.
4. A set of operations are applied as mentioned below.
5. The result is encoded upto 32 integers.
    - Z = T M T_
    - P = Z / Q (element wise division)

Note: Q is multiplied by 10^6 to preserve a floating point precision upto 3 decimals which is negated during the division process with rounding off as per the usual trend.

Modules

1. mat_mul(c,a,b)

    Matrix multiplication of a and b each of port size 4096 bits.

2. mat_transpose(b,a)

    Transpose for T matrix of port size 4096

3. division(out,inp,q) and multiply(out,inp,q)

    Element wise division/multiplication of inp and q each of port size 4096 bits. Rounding off has been implemented in division module.

4. encode(zig,W,x) and decode(X,C,d)

    Encoding in a zig-zag pattern as required for DCT algorithm and Decoding on a similar fashion keeping rest 0. Port size of W,X being 4096 bits, zig and C with 2048 bits, x and d with 192 bits(carrying a sequence to initiate the pattern which is also generated by the python code)

5. dct_test1(C,M,T,T_,Q,x) and idct_test1(N,C,T,T_,ss,Q)

   Main module controlling the operations to be done as per the algorithm. Port size of M,N,T,T_,Q being 4096 bits, C being 2048 bits and x and ss being 192 bits.

6. dct_tb_final and idct_tb_final

   Testbench module which handles input/output matrices from “out_dct.txt”/”out_idct.txt” and constants from “Matrix.txt”

Source Code

```
//###################### Code consisting the algorithm#####################

module mat_mul(c,a,b);


input [4095:0]a;

input [4095:0]b;

output reg [4095:0]c;


integer  i,j,k,x,y,z,sum;

always @ (a or b)

begin

 for(i=0;i<8;i=i+1)

  begin

for(j=0;j<8;j=j+1)

begin

x =  8*i+j;

sum = 0;

                for(k=0;k<8;k=k+1)

                begin

y = 8*i+k;

z = 8*k+j;
```

```
 sum = sum+a[y*64 +:64]*b[z*64 +:64];
end
c[x*64 +:64] = sum;
  end
end
end
endmodule


module mat_transpose(b,a);
output reg [4095:0]b;
input [4095:0]a;
integer i,j,x,y;
always @ a
begin
for(i=0;i<8;i=i+1)
                begin
                for(j=0;j<8;j=j+1)
                begin
                                x = 8*i+j;
                                y = 8*j+i;
                b[x*64 +:64] = a[y*64 +:64];
                end
end
end
endmodule
```

```
module division(out,inp,q);

input  [4095:0] inp;

input [4095:0]q;

output reg [4095:0] out ;

integer zz;

integer a,b;

integer i;


always@(inp) begin

for(i=0;i<=63;i=i+1)

begin

if(inp[63+64*i]!=1) begin

 a=inp[i*64 +: 64];

 b=q[i*64 +: 64];

 zz=a/b + ((a%b>=b/2)?1:0);

 out[i*64 +: 64]=zz;

end

else begin

 a=~(inp[i*64 +: 64]-1 );

 b=q[i*64 +: 64];

 zz= a/b + ((a%b>=b/2)?1:0);

 out[i*64 +: 64]=~zz+1;

end


end
```

```
end

endmodule


module multiply(out,inp,q);

input [4095:0] inp;

input [4095:0]q;

output reg[4095:0] out ;

integer i;



always@(inp) begin


for(i=0;i<=63;i=i+1)

 out[i*64 +: 64]=inp[i*64 +: 64]*q[i*64 +: 64];


end

endmodule


module encode(zig,W,x);

input [4095:0] W;

output reg [2047:0]zig;


integer i;

integer j,k;

input [191:0]x;
```

```
always@(W)
begin
for(i=0;i<=31;i=i+1)
begin

j=x[i*6 +:6];
zig[i*64 +:64]=W[j*64 +:64];

end
end

endmodule

module decode(X,C,d);

input [2047:0]C;

output reg [4095:0]X;
input [191:0]d;
integer i,j;
always@(C)
begin
X=4095'b0;

for(i=0;i<=31;i=i+1)
begin
```

```
j=d[i*6 +:6];
X[j*64 +:64]=C[i*64 +:64];
end

end


endmodule

module dct_test1(C,M,T,T_,Q,x);
output [2047:0]C;
input [4095:0]M;
input [4095:0]T,T_;
input [4095:0]Q;

input [191:0]x;

reg [4095:0]X;
wire [4095:0]Y;
wire [4095:0]Z;
wire  [4095:0]W;
wire [4095:0]P;
reg [4095:0]D;

integer i;
```

```
always@(M)
begin
for(i = 0;i <=63;i = i+1)
begin
        D[i*64 +: 64] = 64'd1000000;
        X[i*64 +: 64] = M[i*64 +: 64]-128 ;
end
end

mat_mul mat_mul1(Y,T,X);
mat_mul mat_mul2(Z,Y,T_);

multiply multiply_1(W,D,Q);

division division_1(P,Z,W);

encode encode_1(C,P,x);

endmodule

module idct_test1(N,C,T,T_,ss,Q);

output reg[4095:0]N;
input [2047:0]C;
input [4095:0]T_,T;
input [4095:0]Q;
```

```
input [191:0]ss;


integer i;

wire [4095:0]X;

reg [4095:0]Y;

wire [4095:0]Z;

wire [4095:0]W;

wire [4095:0]R;

wire [4095:0]P;


decode decode_1(X,C,ss);

multiply multiply_1(R,X,Q);


mat_mul mat_mul_1(Z,T_,R);

mat_mul mat_mul_2(W,Z,T);


initial

begin

for(i=0;i<=63;i=i+1)

begin


Y[i*64 +: 64] =64'd1000000;


end


end
```

```
division division_1(P,W,Y);


always @( P )

begin

for(i=0;i<=63;i=i+1)

begin


N[i*64 +: 64]=P[i*64 +: 64]+128;


end

end

endmodule
```

###################### End of source code ######################

###################### Start of Testbench (input/output) code ######################

```
module dct_tb_final;

integer  M,i,j,Matrix,image,out,x,y;

reg  [4095:0]T;

wire  [4095:0]T_trans;

reg [4095:0]Q;

reg [4095:0]original;

reg [191:0]ss;

integer zz,k,l;

wire [2047:0]C;
```

```
wire [4095:0]P;


dct_test1 dct(C,original,T,T_trans,Q,ss);

mat_transpose transpose(T_trans,T);

initial

begin


        Matrix=$fopen("Matrix.txt","r");

        image=$fopen("image.txt","r");

        out=$fopen("out_dct.txt","w");

        for(i=0;i<64;i=i+1)

        begin

                x=$fscanf(Matrix,"%d",y);

                T[i*64 +: 64]=y;


        end


        for(i=0;i<64;i=i+1)

        begin

                x=$fscanf(Matrix,"%d",y);

                Q[i*64 +:64]=y;


        end

        for(i=0;i<=31;i=i+1)

        begin

                x=$fscanf(Matrix,"%d",y);
```

```
                ss[i*6 +:6]=y;


        end
for(k=0;k<3072;k=k+1) begin
        for(i=0;i<64;i=i+1)
        begin
                x=$fscanf(image,"%d",y);
                original[i*64 +:64]=y;
        end
        #1
        for(j=0;j<=31;j=j+1)
        begin
                $display("%0d # %0d $ %0d ,",$signed(C[64*j +:64]),Q[j*64 +:64],T[j*64 +:64]);
        end
        for(l=0;l<=31&&k<=3072;l = l+1) begin

        $fwrite(out,"%0d  ",$signed(C[l*64 +:64]));
        end
        $fwrite(out,"\n");
end

end



endmodule
```

```
module idct_tb_final;

integer M,i,j,Matrix,image,out,x,y,k,l;

reg [4095:0]T;

wire [4095:0]T_trans;

reg [4095:0]Q;

wire [4095:0]original;

reg [2047:0]C;

reg [191:0]ss;


idct_test1 idct(original,C,T,T_trans,ss,Q);

mat_transpose transpose(T_trans,T);

initial begin

        Matrix=$fopen("Matrix.txt","r");

        image=$fopen("out_dct.txt","r");

        out=$fopen("out_idct.txt","w");

        for(i=0;i<64;i=i+1)

        begin

                x=$fscanf(Matrix,"%d",y);

                T[i*64 +:64]=y;

        end

        for(i=0;i<64;i=i+1)

        begin

                x=$fscanf(Matrix,"%d",y);

                Q[i*64 +:64]=y;


        end
```

```
        for(i=0;i<=31;i=i+1)
        begin
                x=$fscanf(Matrix,"%d",y);
                ss[i*6 +:6]=y;
        end
for(k=0;k<3072;k=k+1) begin
        for(i=0;i<=31;i=i+1)
        begin
                x=$fscanf(image,"%d",y);
                C[i*64 +:64]=y;
        end #1
        for(j=0;j<=63;j=j+1)
        begin
                $display("%0d # %0d $ %0d ,",$signed(original[64*j +:64]),Q[j*64 +:64],T[j*64 +:64]);
        end
        for(l=0;l<=63&&k<=3072;l = l+1) begin

        $fwrite(out,"%0d ",$signed(original[l*64 +:64]));
        end
        $fwrite(out,"\n");
end
end
endmodule
```

#################### End of Testbench ####################

References

- https://www.math.cuhk.edu.hk/~lmlui/dct.pdf
- https://www.geeksforgeeks.org/discrete-cosine-transform-algorithm-program/
- PPT for reference: https://docs.google.com/presentation/d/1BwLgxYBSefvEc3VN8k9dOJzzW3v2QyhgxcZhMWfgLJs/edit#slide=id.p