APPENDIX

DCT CORE IMPLEMENTATION

```
module DCT (d, clk, q);
        input d;
        input clk;
        output q;
flip [2:0} q1;
dff dff1(d,clk,q1[0]);
dff  dff1(q1[0],clk,q1[1]);
dff  dff1(q1[1],clk,q1[2]);
dff  dff1(q1[2],clk,q);
end module
module dff(d,clk,q);
        input d;
        input clk;
        output q;
reg q;
always @(posedge clk)
        q=d;
end module
```

2D-DCT ADDITION

```
module fa(a,b,cin,sum,cout);
        input [3:0]a;
        input [3:0]b;
        input cin;
        output [3:0] sum;
        output cout;
        fa[2:0] c1;
        fa fa1(a[0],b[0],cin, sum[0],c1[0]);
```

```
        fa fa2(a[1],b[1],c1[0], sum[1],c1[1]);
        fa fa3(a[2],b[2],c1[1], sum[2],c1[2]);
        fa fa4(a[3],b[3],c1[2], sum[3],cout);
        end module

module fa (a,b,cin,sum,cout);

        input a;
        input b;
        input cin;
        output sum;
        output cout;

fa s1,c1,c2;
xor xor1(s1,a,b);
xor xor2(sum,s1,cin);
and and1(c1,a,b);
and and2(c2,s1,cin);
or or1(cout,c1,c2);
end module
```

8*8 DCT Matrix Multiplications

```
module rom8*8(y,in);
        output [7:0] y;
        input [2:0] in;
reg [2:0] ROM [7:0];
        assign y=ROM[in];
        initial $readme mb("rom_data.txt",ROM,0,7);
end module
```

STM 32 ARM PROCESSOR

```
module fs32_test();
reg [2:0] testvectors[8:0];
reg clk;
reg [10:0]N,err;
reg a,b,c;
fs d_t,e_t;
fs32 DUT(a,b,c,d_t,e_t);
initial begin
        $readmemb(“testvectors_fs32.txt”,testvectors);
        N=0;err=0;
end

always begin
        clk=0;
        #50;
        clk=1;
        #50;
end

always @(posedge clk)begin
        a=testvectors[N][0];
        b=testvectors[N][1];
        c=testvectors[N][2];
end

always @(nedge clk)begin
        N=N+1;
        $display(“inputs  %b%b%b-Output %b%b”,a,b,c,d_t,e_t);
end

always @(N)begin
        if(N==100||testvectors[N}===3’bx)begin
        $display(“Completed %d tests with %d errors”,N,err);
        $finish;
```

```
        end
end
end module

module fs32(x,y,bin,diff,bout);
        input x;
        input y;
        input bin;
        output diff;
        output bout;

assign diff=x^y^bin;
assign bout=(y&bin)|((y|bin)&(~x));

end module
```