گزارش پیاده سازی ماژول Hash\_Register در پروژه پیاده سازی SHA-256

مقدمه:

بخش نهایی ساخت یک Hash\_Register، پیاده سازی ماژول Hash Register می‌باشد. این ماژول در واقع نگهدارنده مقدار هش شده در طول روند اجرا الگوریتم می‌باشد که به مرور این مقادیر بروز می‌شوند و در نهایت مقدار نهایی سیگنال رمزشده در این رجیستر قرار خواهد داشت.

بررسی روند عملکرد:

این ماژول در واقع از دو بخش اصلی تشکیل شده است.  
۱- یک رجیستر جامع که از ۸ رجیستر ۳۲ بیتی (یک کلمه) تشکیل شده است.

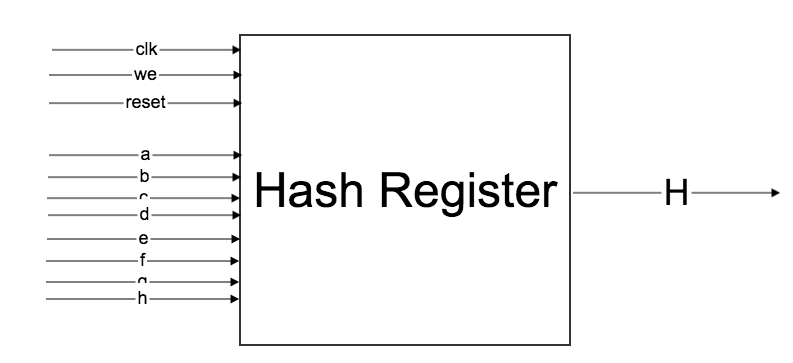
۲- یک جمع کننده ۳۲ بیتی

عملکرد آن نیز به این صورت است که در ورودی ۸ سیگنال ۳۲ بیتی دریافت می‌شود و اگر سیگنال نوشتن فعال باشد، در اولین لبه بالارونده کلاک، مقداری که در هر کدام از ۸ رجیستر وجود دارد با سیگنال متناسب خود جمع شده و دوباره در همان رجیستر ذخیره می‌گردد. کل این عملیات در یک کلاک صورت می‌گیرد.

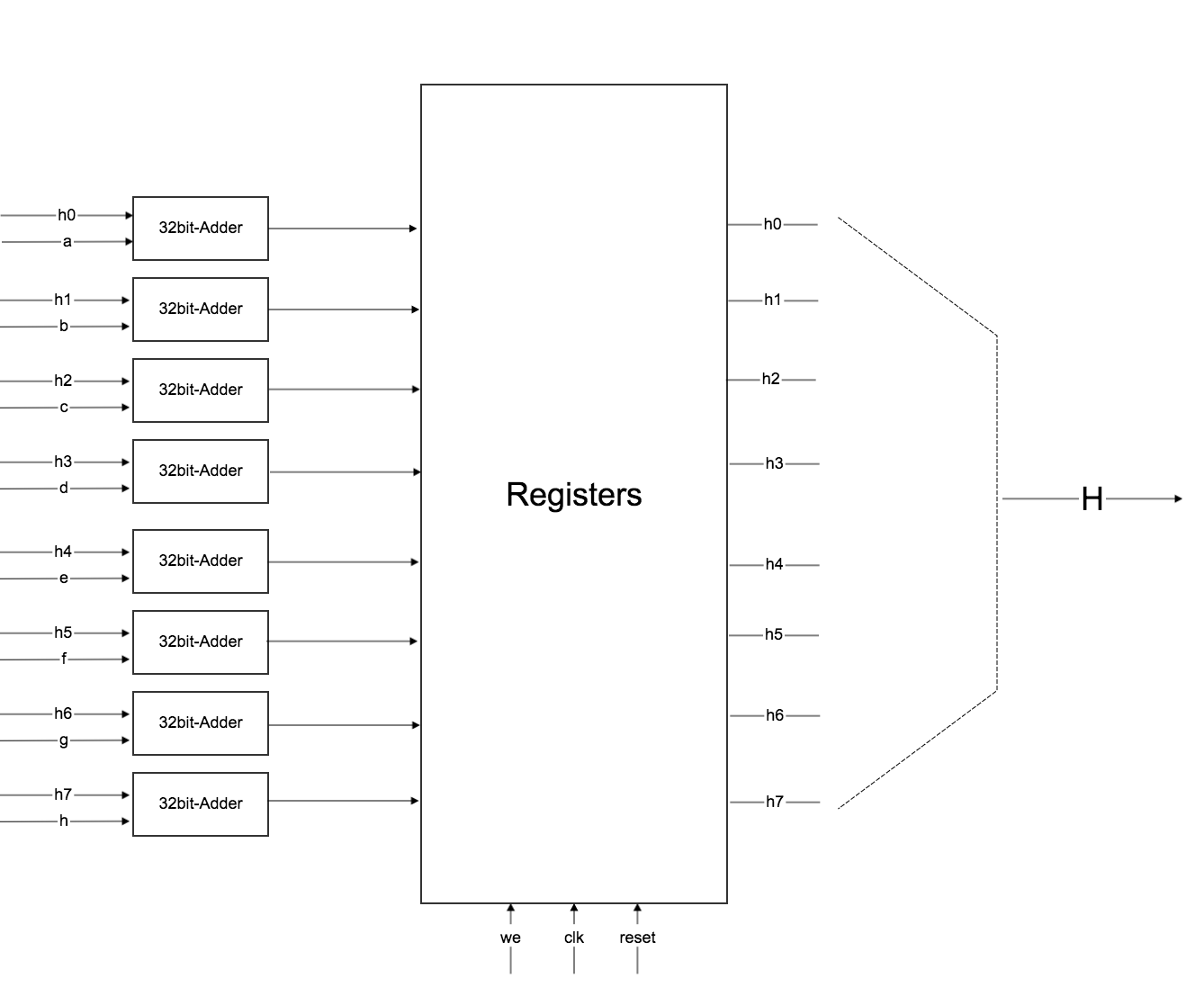
این ماژول همچنین از یک سیگنال ریست که به صورت سنکرون عمل می‌کند و در سطح بالا فعال می‌شود بهره می‌برد.

شکل شماتیک ماژول:

شکل شماتیک ماژول به همراه سیگنال‌های ورودی و خروجی آن را می‌توانید در شکل زیر مشاهده کنید:



همچنین شماتیک درون این ماژول و دو بخش اصلی آن را می‌توانید در شکل زیر مشاهده کنید:



نحوه ارتباط با سایر ماژول‌ها:

این ماژول چون ماژول نهایی پروژه اصلی می‌باشد، بعد از این ماژول، ماژول دیگری وجود نخواهد داشت که بخواهد با آن ارتباط داشته باشد و پاسخ سیگنال اولیه ورودی و رمزشده آن در این ماژول نهایی می‌شود. اما ماژول Round Registers با آن ارتباط خواهد داشت و سیگنال‌های ورودی این ماژول را آماده می‌کند. در واقع ماژول پیشین، پس از آماده سازی کامل ۸ سیگنال ۳۲ بیتی لازم برای این مرحله آن را به این ماژول به عنوان ورودی و در قالب سیگنال‌های a-h می‌دهد و سپس سیگنال نوشتن (we) را فعال می‌کند. سیگنال کلاک که بین ماژول‌ها مشترک است. همچنین پس از نهایی شدن عملیات رمزنگاری، برای شروع عملیات بعدی باید این ماژول ریست شود پس کنترل کننده اصلی این سیگنال را برای یک کلاک یک می‌کند.

پیاده سازی:

برای پیاده سازی این ماژول از زبان Verilog استفاده شده است که توسط نرم‌افزار Modelsim نوشته و اجرا شده است.

یکی از زیر ماژول‌های مورد استفاده در این ماژول یک جمع کننده ۳۲ بیتی بود که ما این جمع کننده را به صورت یک جمع کننده سری متشکل از ۳۱ جمع کننده کامل و ۱ یک نیم‌جمع‌کننده ساختیم. همچنین تعداد بیت‌های این جمع‌ کننده را به صورت پارامتریک در نظر گرفتیم تا قابل استفاده مجدد نیز باشد.

این کد را می‌توانید در زیر مشاهده کنید:

// This Module has written by Ali Zeynali

//32 bit (parametric) adder

module adder32(input1, input2, answer);

parameter N=32;

input [N-1:0] input1,input2;

output [N-1:0] answer;

wire carry\_out;

wire [N-1:0] carry;

genvar i;

generate

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

begin: generate\_N\_bit\_Adder

if(i==0)

half\_adder f(input1[0],input2[0],answer[0],carry[0]);

else

full\_adder f(input1[i],input2[i],carry[i-1],answer[i],carry[i]);

end

assign carry\_out = carry[N-1];

endgenerate

endmodule

//Half Adder code

module half\_adder(x,y,s,c);

input x,y;

output s,c;

assign s=x^y;

assign c=x&y;

endmodule

// Full Adder code

module full\_adder(x, y, c\_in, s, c\_out);

input x, y, c\_in;

output s, c\_out;

assign s = (x^y) ^ c\_in;

assign c\_out = (y&c\_in)| (x&y) | (x&c\_in);

endmodule

که درون این ماژول دو کد زیر بخش برای جمع‌کننده کامل و نیم‌جمع کننده وجود دارد.

برای پیاده سازی ماژول اصلی این بخش، از ۸ جمع‌کننده ۳۲ بیتی خود نمونه گرفتیم و ورودی‌ها را به ‌آن‌ها دادیم. همچنین از ۸ سیگنال کمکی h0 تا h7 کمک گرفتیم تا خروجی این جمع‌ها را داشته باشیم. سپس در لبه بالارونده کلاک بررسی کردیم که اگر ریست یک بود مقادیر رجیستر ریست شوند و در غیر این صورت مقدار h0 تا h7 جانشین آن‌ها شوند. در زیر کد این ماژول را مشاهده می‌کنید:

module Hash\_Register(H, clk, we, reset, a, b, c, d, e, f, g, h);

input clk, we, reset; // reset is synchronize and active high

input [31:0] a, b, c, d, e, f, g, h; // a is H[0], b is H[1], c is H[2] ... h is H[7]

output [32\*8-1:0] H; //Output hashed

reg [31:0] h0, h1, h2, h3, h4, h5, h6, h7; //registers which contain hash values

wire [31:0] w0, w1, w2, w3, w4, w5, w6, w7; // wires wich connect adder to register

initial

begin

// initial value of hash registers

// values are derived from follow resource:

// https://csrc.nist.gov/csrc/media/publications/fips/180/4/final/documents/fips180-4-draft-aug2014.pdf

h0 = 32'h6a09e667;

h1 = 32'hbb67ae85;

h2 = 32'h3c6ef372;

h3 = 32'ha54ff53a;

h4 = 32'h510e527f;

h5 = 32'h9b05688c;

h6 = 32'h1f83d9ab;

h7 = 32'h5be0cd19;

end

//assign output to saved registers

assign H = {h0, h1, h2, h3, h4, h5, h6, h7};

adder32 adder1(a, h0, w0);

adder32 adder2(b, h1, w1);

adder32 adder3(c, h2, w2);

adder32 adder4(d, h3, w3);

adder32 adder5(e, h4, w4);

adder32 adder6(f, h5, w5);

adder32 adder7(g, h6, w6);

adder32 adder8(h, h7, w7);

always @(posedge clk)

begin

if (reset)

begin

// restore registers to initial values

h0 = 32'h6a09e667;

h1 = 32'hbb67ae85;

h2 = 32'h3c6ef372;

h3 = 32'ha54ff53a;

h4 = 32'h510e527f;

h5 = 32'h9b05688c;

h6 = 32'h1f83d9ab;

h7 = 32'h5be0cd19;

end

else

begin

// if writing is enable:

if (we)

begin

//add new H values to recent values and save them in H again

h0 = w0;

h1 = w1;

h2 = w2;

h3 = w3;

h4 = w4;

h5 = w5;

h6 = w6;

h7 = w7;

end

end

end

endmodule

تست:

در بخش تست ابتدا ماژول جمع کننده را مورد تست قرار دادیم. این ماژول بصورت ترکیبی عمل می‌کند. برای تست این ماژول از یک کد وریلاگ زیر کمک گرفتیم که خودمان آن را نوشته‌ایم.

module adder32\_tb;

reg [31:0] input1, input2;

wire [31:0] result;

adder32 my\_adder\_test(input1, input2, result);

initial

begin

input1 = 32'd0;

input2 = 32'd0;

#5

input1 = 32'd1;

input2 = 32'd0;

#5

input1 = 32'd5;

input2 = 32'd10;

#5

input1 = 32'd124;

input2 = 32'd1237;

#5

input1 = 32'h10101010;

input2 = 32'h4ABFFFFF;

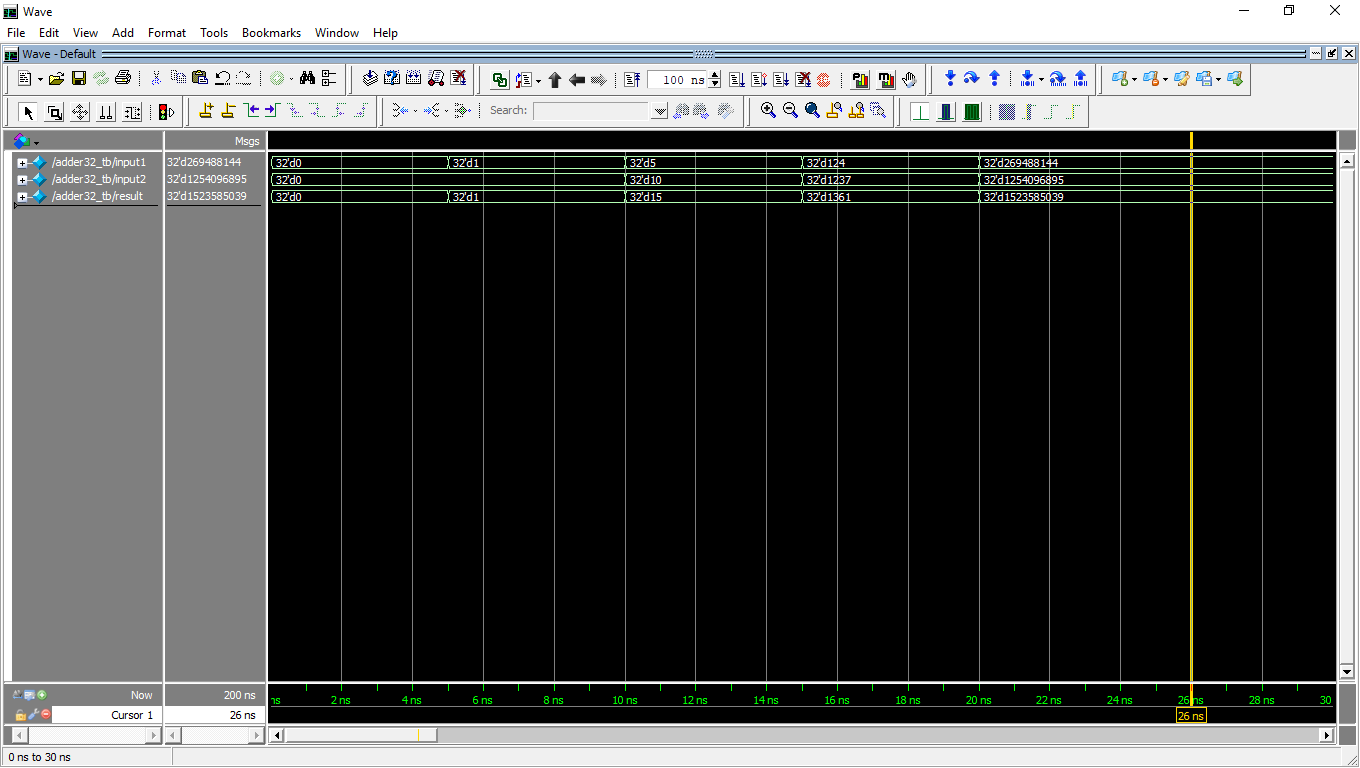
end

initial

$monitor($time, " , input1\_binary: %b, input2\_binary: %b, result\_binary: %b , input1\_decimal: %d, input2\_decimal: %d, result\_decimal: %d", input1,input2, result, input1, input2, result);

endmodule

که مقادیر ورودی را برای چند مقدار خاص و یا رندم مورد بررسی قرار می‌دهد. شکل سیگنال خروجی به صورت زیر بود که صحت کار را نشان می‌دهد:



سپس ماژول اصلی را مورد تست قرار دادیم. برای تست این ماژول نیز از کد زیر استفاده کردیم. چون این ماژول به مراتب از ماژول جمع‌کننده پیچیده‌تر است از تعداد تست‌های بیشتری برای بررسی آن استفاده کردیم. کد زیر این تست را نشان می‌دهد:

module Hash\_Register\_tb;

reg clk, we, reset;

reg [31:0] a,b,c,d,e,f,g,h;

wire [32\*8-1:0] H;

initial

begin

clk = 1'b0;

we = 1'b0;

reset = 1'b0;

a = 32'd0;

b = 32'd0;

c = 32'd0;

d = 32'd0;

e = 32'd0;

f = 32'd0;

g = 32'd0;

h = 32'd1;

#20

we = 1'b1;

#10

a = 32'd0;

b = 32'd1;

c = 32'd0;

d = 32'd0;

e = 32'd0;

f = 32'd0;

g = 32'd0;

h = 32'd0;

#10

a = 32'd0;

b = 32'd0;

c = 32'd1;

d = 32'd0;

e = 32'd0;

f = 32'd0;

g = 32'd0;

h = 32'd0;

#10

a = 32'd0;

b = 32'd0;

c = 32'd0;

d = 32'd1;

e = 32'd0;

f = 32'd0;

g = 32'd0;

h = 32'd0;

#10

a = 32'd0;

b = 32'd0;

c = 32'd0;

d = 32'd0;

e = 32'd1;

f = 32'd0;

g = 32'd0;

h = 32'd0;

#10

a = 32'd0;

b = 32'd0;

c = 32'd0;

d = 32'd0;

e = 32'd0;

f = 32'd1;

g = 32'd0;

h = 32'd0;

#10

a = 32'd0;

b = 32'd0;

c = 32'd0;

d = 32'd0;

e = 32'd0;

f = 32'd0;

g = 32'd1;

h = 32'd0;

#10

a = 32'd0;

b = 32'd0;

c = 32'd0;

d = 32'd0;

e = 32'd0;

f = 32'd5;

g = 32'd0;

h = 32'd10;

#10

a = 32'd10;

b = 32'd11;

c = 32'd12;

d = 32'd13;

e = 32'd14;

f = 32'd15;

g = 32'd16;

h = 32'd17;

#5

reset = 1'b1;

#20

a = 32'd0;

b = 32'd0;

c = 32'd0;

d = 32'd0;

e = 32'd0;

f = 32'd0;

g = 32'd0;

h = 32'd1;

end

always

#5 clk = ~clk;

Hash\_Register my\_hr (H, clk, we, reset, a, b, c, d, e, f, g, h);

endmodule

که مقدار کلاک ساخته شده را در هر ۵ واحد زمان تغییر دادیم تا سیگنال تولید شود. در ابتدا ورودی‌ها به ماژول داده می‌شود تا خروجی آن را در سیگنال ۲۵۶ بیتی نهایی مشاهده کنیم. همچنین در نهایت ماژول را ریست می‌کنیم تا بررسی عملکرد ریست آن نیز مورد آزمایش قرار بگیرد. در شکل زیر چند بخش کوچک از نمودار سیگنال‌های این تست را مشاهده می‌کنید:

