COMP225 - 디지털 설계 및 실험 Project 보고서

2019118024 백종인

Index

- State 의 변화로 코딩을 구성하지 않아 fsm은 생략하였습니다.

1. C++ Code Source Description

(1) length가 128 (512bit) 인 문자열을 입력으로 받습니다.

(2) 문자열을 하나씩 Char로 parsing하여 16진수로 변환후에 배열 M에 넣습니다.

- M은 32bit형의 배열입니다.

```
string str;

cin >> str;

uint32_t W[80];
uint32_t M[129];
for (int i = 0; i < 128; i++) {
    M[i] = convert(str[i]);
}
```

(3) 입력 값으로 채울 수 있는 W[0] ~ W[15] 값을 채웁니다.

```
//W[0] ~ W[15] 계산
for (int i = 0; i < str.size(); i += 8) {
    W[cnt++] = (M[i] << 28) | (M[i + 1] << 24) | (M[i + 2] << 20) | (M[i + 3] << 16) | (M[i + 4] << 12) |
        (M[i + 5] << 8) | (M[i + 6] << 4) | (M[i + 7]);
}
```

W는 배열마다 1개의 Word 로 되어있습니다. (32bit)

M에서 길이 8의 단위로 가져와 1Word를 만듭니다.

이때 각 숫자의 가중치가 다르기 때문에 shift를 해주어야 합니다.

//

Ex> 1011_0011-> 0001 0000 0001 0001 _ 0000 0000 0001 0001 입니다.

M의 저장되어 있는 16진수 값들은 32bit로 저장되어 있기에, 가장 앞의 1이라고 하면

0000 0000 0000 0000 _ 0000 0000 0000 0001 형태를 지니고 있습니다.

Left Shift 28 수행하여 0001 0000 0000 0000 _ 0000 0000 0000 0000 로 만들어 줍

니다.

//

가중치에 맞게 Left Shift를 해주고 마지막으로 OR연산을 진행해주어 1개의 Word를 완성시킵니다.

(4) 이후에 W[16] ~ W[79] 까지는 SHA1의 알고리즘에 따라 계산합니다.

단, 이때는 rotate_shift 라는 연산이 진행되는데 Define 함수로 지정하여 사용해줍니다.

```
for (int i = 16; i < 80; i++) {
    W[i] = (W[i - 3] ^ W[i - 8] ^ W[i - 14] ^ W[i - 16]);
    W[i] = rotate_shift(1, W[i]);
}
```

```
//bits : 옮길 amount , word : 변수
#define rotate_shift(bits,word) (((word) << (bits)) | ((word) >> (32-(bits))))
```

(5) 초기 a, b, c, d, e 값을 초기화 해줍니다. 값은 배열 h에 저장해 두었습니다

```
uint32_t a = h[0];
uint32_t b = h[1];
uint32_t c = h[2];
uint32_t d = h[3];
uint32_t e = h[4];
```

(6) 총 80번의 연산을 진행하면서 최종 a, b, c, d, e 값을 도출합니다.

연산은 기본 SHA1알고리즘을 따르며 연산 20번마다 지정된 k값을 설정해줍니다.

```c
//a, b, c, d, e 값 검사

for (int i = 0; i < 80; i++) {

    if (0 <= i && i <= 19) {
        f = (b & c) | ((~b) & d);
        k = 0x5A827999;
    }

    else if (20 <= i && i <= 39) {
        f = b ^ c ^ d;
        k = 0x6ED9EBA1;
    }


    else if (40 <= i && i <= 59) {
        f = ((b & c) | (b & d) | (c & d));
        k = 0x8F1BBCDC;
    }


    else if (60 <= i && i <= 79) {
        f = b ^ c ^ d;
        k = 0xCA62C1D6;
    }

    uint32_t temp = (rotate_shift(5, a) + f + e + k + W[i]);
    e = d;
    d = c;
    c = rotate_shift(30, b);
    b = a;
    a = temp;
```

(7) 최종적으로 계산된 a, b, c, d, e 값과 초기값을 더해줘서 최종 output을 만들어 냅니다.

C:\Windows\system32\cmd.exe

```
74657374800000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000020
Round 1 --> A: a594e7c5 B: d7e2bb59 C: 4a36181f D: fb609e49 E: 1a43078
Round 2 --> A: 80f549bf B: ed3e0d75 C: 6ccac64c D: 462ae26f E: f93a146
Round 3 --> A: 1387e9e4 B: 2a106003 C: d91959bb D: 6cf9fcdf E: 84d25bda
Round 4 --> A: 42056ce4 B: dce3f01d C: 83912b75 D: c35f9511 E: d45cd9e3
result : a94a8fe5ccb19ba61c4c0873d391e987982fbbd3계속하려면 아무 키나 누르십시오 . . .
```

- 디버깅 코드까지 포함하여 출력한 결과

2. Verilog Code Source Description

변수 및 벡터 설명

- Input
  - [511:0] SHA1IN : 512bit의 입력 값
  - CLK : 주기적으로 반복되는 clock (posedge)
  - START : Operation을 시작하는 신호 (1Cycle 유지)
  - nRST : Reset 신호 (negedge), 0이 들어오면 초기화 된다.

- Output
  - [159:0] SHA1OUT : 160bit의 출력 값
  - DONE : Operation이 끝났음을 알리는 신호 (1Cycle 유지)

- In sha1 module
  - [31:0] W [79:0] : 32bit의 워드를 저장할 수 있는 벡터 배열
  - [31:0] H [4:0] : 초기 A, B, C, D, E 에 들어가는 값 저장
  - State : START 신호가 들어왔었는지 저장용
  - [7:0] t : rotation shift용 reg

(1) Input으로 512bit의 SHA1IN을 입력 받습니다.

```
2
3  module sha1(SHA1IN, SHA1OUT, CLK, nRST, START, DONE);
4
5  //in module -> input : net
6  input [511:0]SHA1IN; // 512bit input
7  input CLK, nRST, START;
8
9  //in module -> output : reg or net
10 output [159:0] SHA1OUT;
11 output DONE;
12 reg [159:0] SHA1OUT;
13
14 reg DONE;
15 reg [31:0] W[79:0];
16 reg [31:0] H[4:0];
17 reg state;
18
```

(2) 1Word (32bit) 씩 Parsing하여 W[0] ~ W[15]를 채워 넣습니다.

```
36
37 initial
38 begin
39   W[0] = SHA1IN[511:480];
40   W[1] = SHA1IN[479:448];
41   W[2] = SHA1IN[447:416];
42   W[3] = SHA1IN[415:384];
43   W[4] = SHA1IN[383:352];
44   W[5] = SHA1IN[351:320];
45   W[6] = SHA1IN[319:288];
46   W[7] = SHA1IN[287:256];
47   W[8] = SHA1IN[255:224];
48   W[9] = SHA1IN[223:192];
49   W[10] = SHA1IN[191:160];
50   W[11] = SHA1IN[159:128];
51   W[12] = SHA1IN[127:96];
52   W[13] = SHA1IN[95:64];
53   W[14] = SHA1IN[63:32];
54   W[15] = SHA1IN[31:0];
55 end
56
```

(3) 이후 지정된 식으로 W[16] ~ W[79] 또한 계산합니다.

```
66    integer i;
67    always@(posedge CLK && START)
68
69    begin
70    for(i = 16; i<80; i = i+1)
71        begin
72            W[i] = W[i-3] ^ W[i-8] ^ W[i-14] ^ W[i-16];
73            W[i] = (W[i] << 1) | (W[i] >> (32 - 1) ); //rotate shift
74        end
75
76    end
```

(4) 초기 A, B, C, D, E 값을 설정해줍니다.

```
84    A = H[0];
85    B = H[1];
86    C = H[2];
87    D = H[3];
88    E = H[4];
```

- 초기값은 H벡터에 배열형태로 저장되어 있습니다.

```
29    begin
30        H[0] = 32'h67452301;
31        H[1] = 32'hEFCDAB89;
32        H[2] = 32'h98BADCFE;
33        H[3] = 32'h10325476;
34        H[4] = 32'hC3D2E1F0;
35    end
```

(5) START 신호가 들어오면 posedge CLK 일때마다 Operation 1개를 실행합니다.

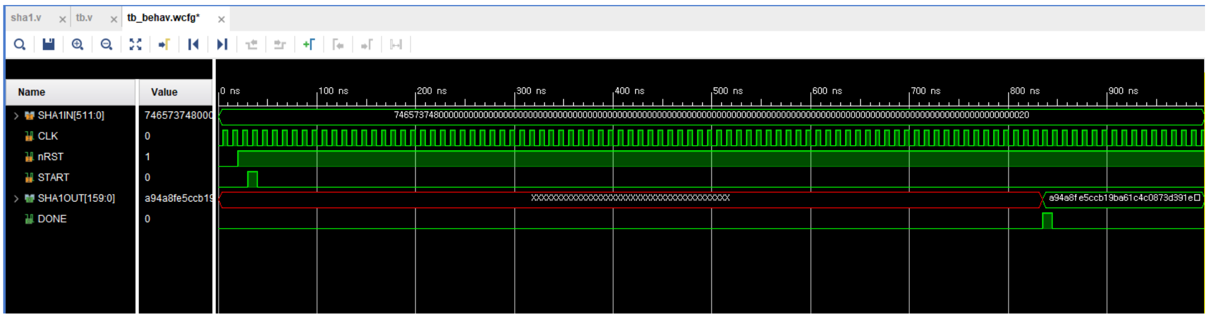
Operation을 수행할 때 마다 F값을 계산해주며 A, B, C, D, E 또한 갱신됩니다.

```
always@(posedge CLK && state) // excute 1operation when CLK is posedge
    begin
            if(0 <= t && t<=19)
                begin
                    F = (B & C) | ((~B) & D);
                    K = 32'h5A827999;
                end

            else if(20 <= t && t <= 39)
                begin
                    F = B ^ C ^ D;
                    K = 32'h6ED9EBA1;
                end

            else if(40 <= t && t<=59)
                begin
                    F = ((B & C) | (B & D) | (C & D));
                    K = 32'h8F1BBCDC;
                end

            else if(60<= t && t<=79)
                begin
                    F = (B ^ C ^ D);
                    K = 32'hCA62C1D6;
                end


            temp = (A << 5) | (A >> (32-5) );
            temp = temp + F + E + K + W[t];

            E = D;
            D = C;
            C = ( B << 30) | ( B >> (32-30) );
            B = A;
            A = temp;

            t = t+8'b0000_0001;

    end // end of Always
```

(6) 최종적으로 계산된 A ~ E 값과 초기 A ~ E 값을 각각 더하여서 SHA1OUT 값을 만들어 냅니다. 이때 최종 계산이 끝남과 동시에 DONE 신호를 띄웁니다.

```
always@(posedge CLK)
begin
if(t==80 && DONE == 0)
    begin
        H[0] = H[0] + A;
        H[1] = H[1] + B;
        H[2] = H[2] + C;
        H[3] = H[3] + D;
        H[4] = H[4] + E;

        DONE = 1;
        SHA1OUT = {H[0],H[1],H[2],H[3],H[4]};
    end
end
```

2 - 1 Result



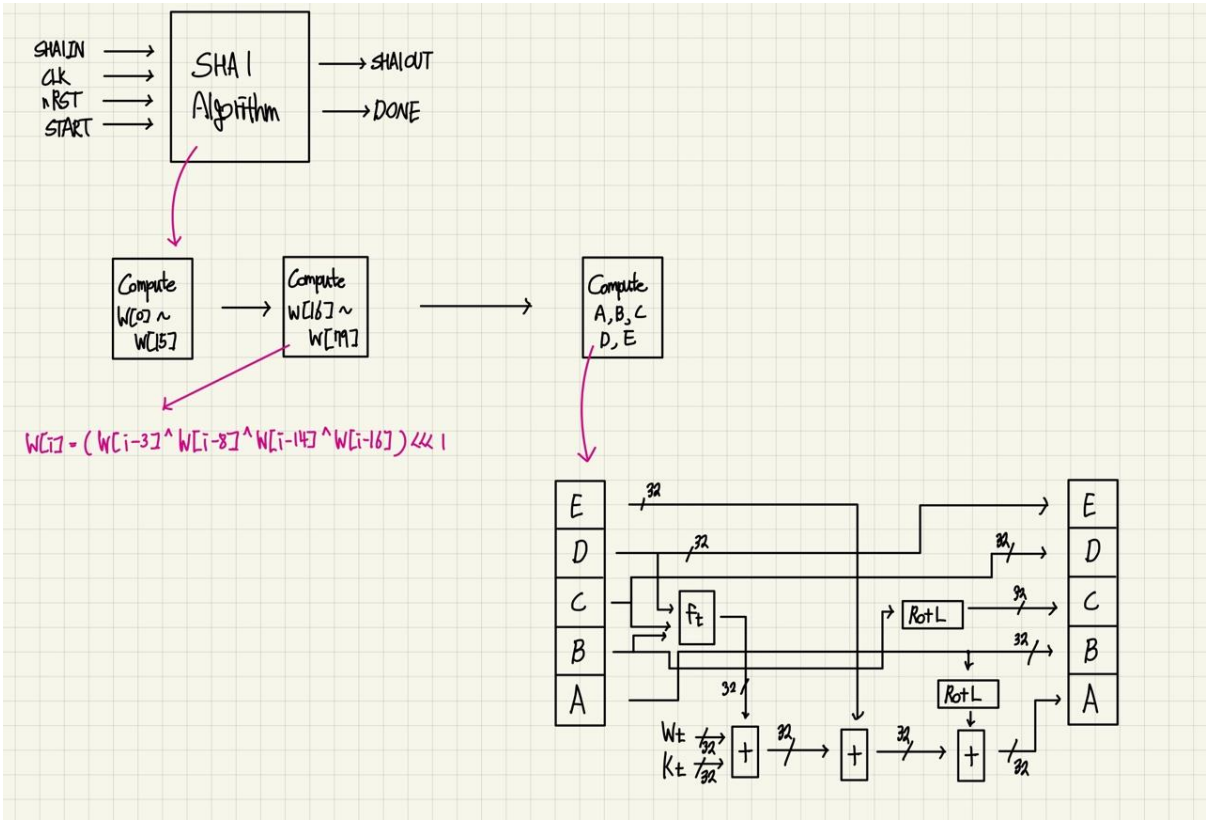
INPUT (512 bit)

7465737480000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
0000020

OUTPUT (160 bit)

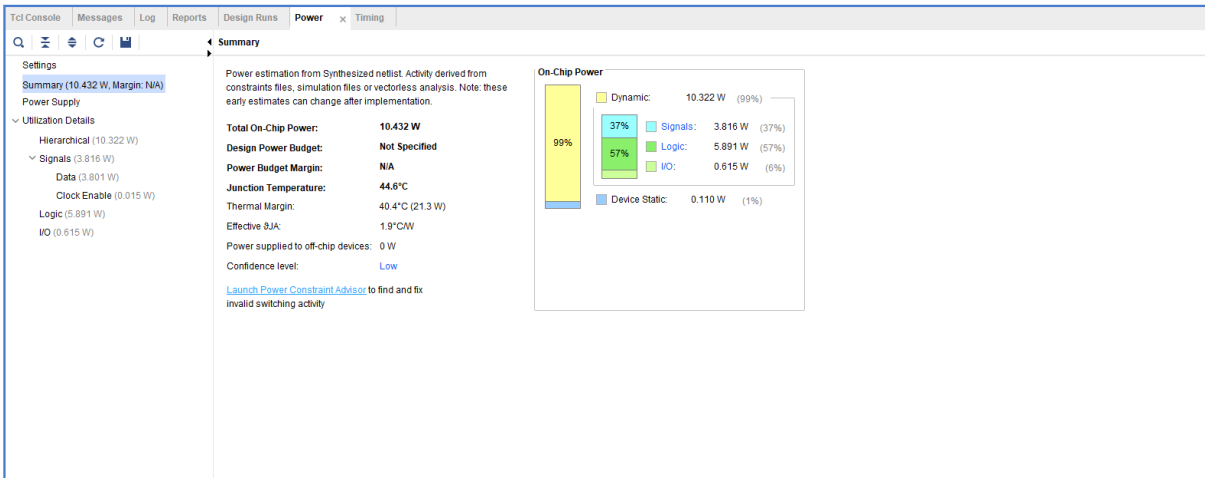
A94A8FE5 CCB19BA6 1C4C0873 D391E987 982FBBD3

2 - 2 Data Path

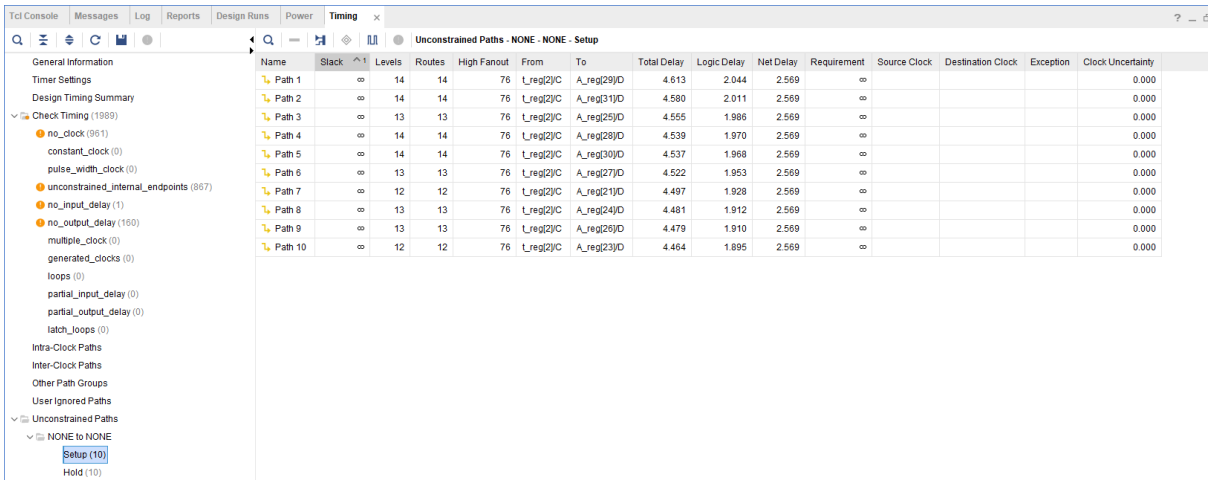


2 - 3 Synthesis

- Power



- Timing



Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	∞	14	14	76	t_reg[2]/C	A_reg[29]/D	4.613	2.044	2.569	∞				0.000
Path 2	∞	14	14	76	t_reg[2]/C	A_reg[31]/D	4.580	2.011	2.569	∞				0.000
Path 3	∞	13	13	76	t_reg[2]/C	A_reg[25]/D	4.555	1.986	2.569	∞				0.000
Path 4	∞	14	14	76	t_reg[2]/C	A_reg[28]/D	4.539	1.970	2.569	∞				0.000
Path 5	∞	14	14	76	t_reg[2]/C	A_reg[30]/D	4.537	1.968	2.569	∞				0.000
Path 6	∞	13	13	76	t_reg[2]/C	A_reg[27]/D	4.522	1.953	2.569	∞				0.000
Path 7	∞	12	12	76	t_reg[2]/C	A_reg[21]/D	4.497	1.928	2.569	∞				0.000
Path 8	∞	13	13	76	t_reg[2]/C	A_reg[24]/D	4.481	1.912	2.569	∞				0.000
Path 9	∞	13	13	76	t_reg[2]/C	A_reg[26]/D	4.479	1.910	2.569	∞				0.000
Path 10	∞	12	12	76	t_reg[2]/C	A_reg[23]/D	4.464	1.895	2.569	∞				0.000

- Schematic