

디지털 논리 회로  
과제 보고서  
Verilog를 사용한RCA제작

학과: 컴퓨터정보공학부

학번: 2024402047

이름: 하성준

담당 교수: 유지현

## 1. 문제 정의

### A. 목적

해당 과제는 베릴로그를 사용하여 16bit ,40bit ripple carry adder를 제작하고 제작한 RCA를 기반으로 80bit RCA를 제작후 두개의 80bit RCA의 성능을 비교하는데 목적이 있다.

## 2. 입·출력 신호

### A. 16bit RCA

#### i. 입력

변수	길이	설명
$A$	16	피가산수
$B$	16	가산수
$C_{in}$	1	이전단계의 carry값

#### ii. 출력

변수	길이	설명
$S$	16	덧셈 결과
$C_{out}$	1	최종 carry출력

### B. 40bit RCA

#### i. 입력

변수	길이	설명
$A$	40	피가산수
$B$	40	가산수
$C_{in}$	1	이전단계의 carry값

#### ii. 출력

변수	길이	설명
$S$	40	덧셈 결과
$C_{out}$	1	최종 carry출력

### C. 80bit RCA(16bit 직렬,40bit 직렬 동일)

#### i. 입력

변수	길이	설명
$A$	80	피가산수
$B$	80	가산수
$C_{in}$	1	이전단계의 carry값

#### ii. 출력

변수	길이	설명
$S$	80	덧셈 결과
$C_{out}$	1	최종 carry출력

### 3. 기본 게이트 수준 설계 및 Full Adder 도식화

#### A. K-map

##### i. S

$C_{in} \backslash AB$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + AB\bar{C}_{in} + A\bar{B}C_{in}$$

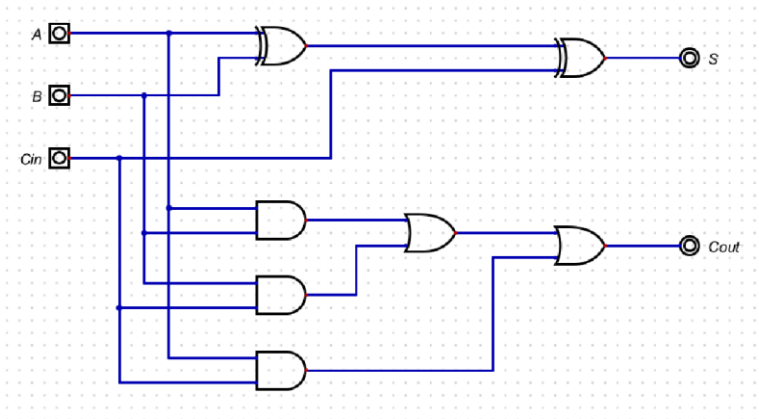
$$= A \oplus B \oplus C_{in}$$

##### ii. $C_{in}$

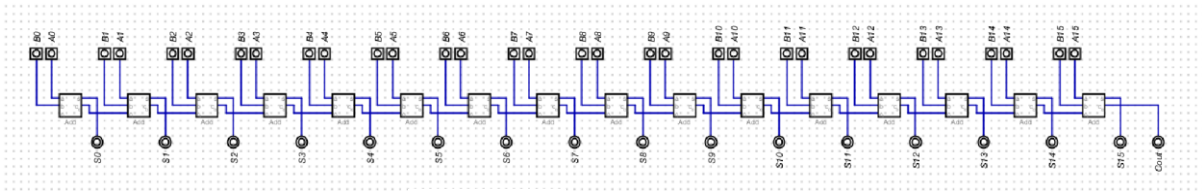
$C_{in} \backslash AB$	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C_{out} = AB + BC + CA$$

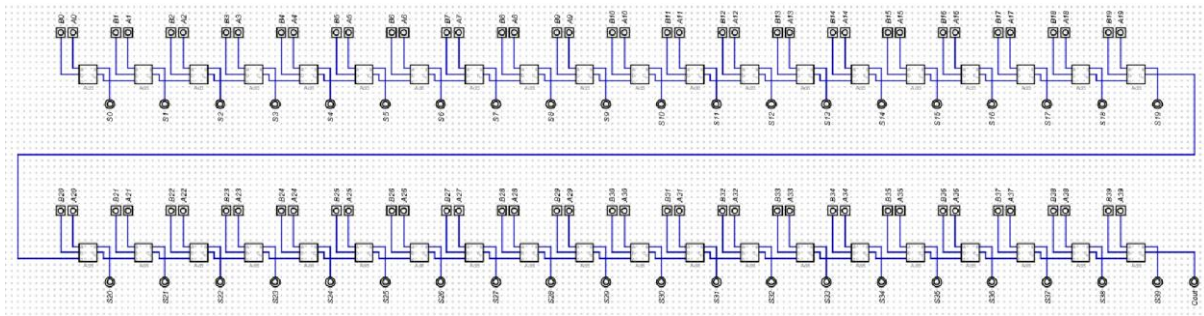
B. 회로도



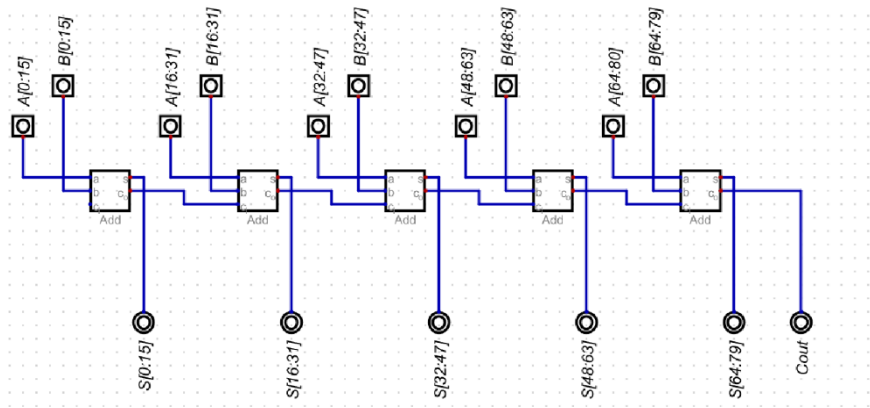
4. 블록다이어그램



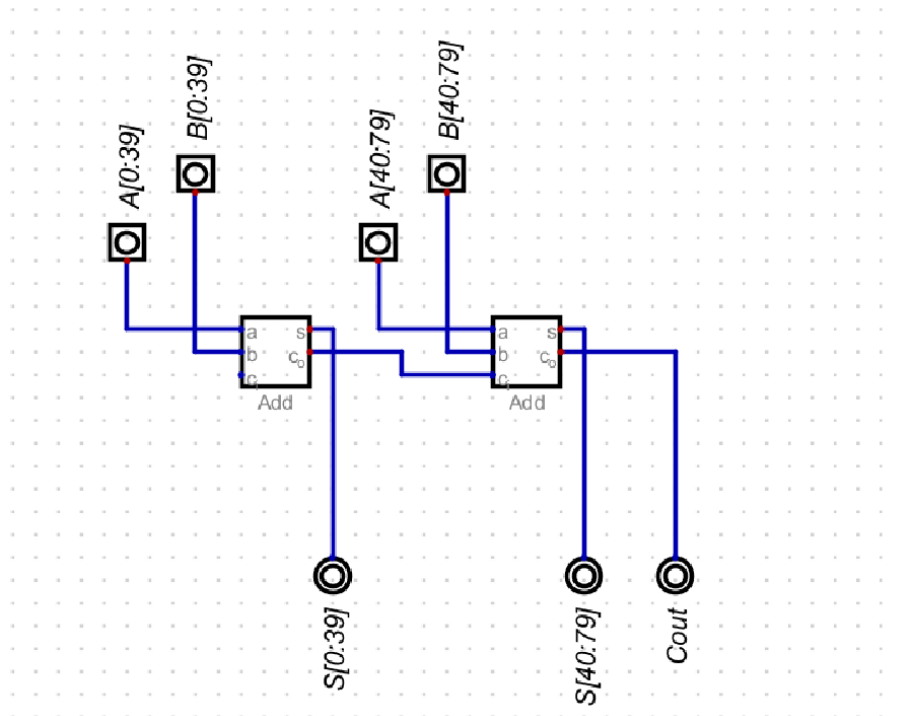
16bit-RCA



40bit-RCA



80bit-RCA by 16bit-RCA



80bit-RCA by 40bit-RCA

## 5. RCA 동작 설명 및 예시

RCA는  $n$ 비트 이진수  $A$ ,  $B$ 를 더하기 위해  $n$ 개의 Full Adder를 사용한다.

### A. 기본구조

i. Full adder의 다음과 같은 3개의 입력을 받는다.

$A[i]$ :  $A$ 의  $i$ 번째 비트

$B[i]$ :  $B$ 의  $i$ 번째 비트

$Cin[i]$ : 이전 자리에서 넘어온 올림수(Carry-in)

ii. 출력은 다음과 같다.

$Sum[i]$ :  $i$ 번째 자리의 합 결과

$Cout[i]$ : 다음 Full Adder로 전달되는 Carry-out

## B. 동작 원리

i번째 비트에서, 입력 A[i]와 B[i], 그리고 이전 자리(i-1)의 캐리(Carry-in)를 더하여 합(S[i])을 출력하며, 연산 결과로 발생한 캐리(Carry-out)는 Cout[i]로 출력된다.

### i. Cin은 왜 필요한가?

Cin의 필요성은 다음 이진수 덧셈 예제를 통해 명확히 알 수 있다.

$$\begin{aligned} 01011_2 + 01101_2 &= 11000_2 (24_{10}) \text{ (Cin 이 존재하는 경우)} \\ &= 00110_2 (6_{10}) \text{ (Cin 이 없는 경우)} \end{aligned}$$

위 결과에서 알 수 있듯이, Carry-in (Cin)이 없을 경우 이진 덧셈 과정에서 올바른 자리올림(Carry Propagation)이 이루어지지 않아, 정확한 연산 결과를 얻지 못하는 오류가 발생하게 된다. 따라서 전가산기(Full Adder) 구조에서 Cin은 필수적인 입력이며, 누적 자리올림 정보를 전달하여 정확한 이진 덧셈 수행을 가능하게 하는 역할을 한다.

### ii. Cin은 어떤 값을 가지는가?

RCA에서는 첫 번째 비트(LSB)의 입력 자리올림(Cin)은 항상 0으로 설정된다. 그 이후 각 비트의 Cin은 바로 이전 Full Adder에서 발생한 자리올림 출력(Cout) 값을 입력으로 받는다.

### iii. 예시

$$1. 01011_2 + 00101_2 = 10000_2$$

인덱스	A[i]	B[i]	Cin[i]	S[i]	Cout[i]
0	1	1	0	0	1
1	1	0	1	0	1
2	0	1	1	0	1
3	1	0	1	0	1
4	0	0	1	1	0

위에서 보는 바와 같이 carry는 오른쪽에서 왼쪽으로 연쇄작용하는 것을 알 수 있다.

## 6. 코드

### A. rca16.v

```
// 1비트 Full Adder 모듈 정의
module full_adder (
    input wire A,      // 피가산수 A (1비트)
    input wire B,      // 가산수 B (1비트)
    input wire Cin,    // 이전 자리에서 전달된 캐리 입력 (Carry-in)
    output wire Sum,    // 현재 자리의 합 결과 (1비트)
    output wire Cout   // 다음 자리로 전달할 캐리 출력 (Carry-out)
);

// 1비트 덧셈: 합은 XOR 세 개로 계산됨
assign Sum = A ^ B ^ Cin;

// 캐리는 두 입력이 모두 1이거나, 하나가 1이고 Carry-in이 1일 때 발생
assign Cout = (A & B) | (B & Cin) | (A & Cin);

endmodule

// 16비트 Ripple Carry Adder (RCA) 모듈 정의
module rca16 (
    input wire [15:0] A,    // 피가산수 A (16비트)
    input wire [15:0] B,    // 가산수 B (16비트)
    input wire Cin,        // 처음 자리의 캐리 입력
    output wire [15:0] Sum, // 합 결과 (16비트)
    output wire Cout       // 최종 캐리 출력
);

// 내부 캐리 신호를 저장할 16비트 와이어 배열
wire [15:0] carry;

// 첫 번째 비트(LSB)에 대해 Full Adder 수행
full_adder FA0 (
    .A    (A[0]),          // A의 0번째 비트
    .B    (B[0]),          // B의 0번째 비트
    .Cin   (Cin),          // 입력 캐리
    .Sum   (Sum[0]),       // 결과의 0번째 비트
    .Cout  (carry[0])      // 다음 비트로 전달될 캐리
);
```

```

// 나머지 15개의 Full Adder를 generate 문을 사용해 반복 생성

genvar i;

generate

    for (i = 1; i < 16; i = i + 1) begin : adder_loop

        full_adder FA (

            .A    (A[i]),          // A의 i번째 비트

            .B    (B[i]),          // B의 i번째 비트

            .Cin  (carry[i-1]),    // 이전 비트에서 전달된 캐리 입력

            .Sum   (Sum[i]),        // 결과의 i번째 비트

            .Cout (carry[i])       // 다음 비트로 전달될 캐리 출력

        );

    end

endgenerate

// 마지막 자리에서 발생한 캐리를 최종 캐리 출력 Cout에 연결

assign Cout = carry[15];

endmodule

```



## B. tb\_rca16.v

```
`timescale 1ns/1ps // 시뮬레이션 시간 단위 설정: 1ns 단위, 1ps 정밀도
// 테스트벤치 모듈 정의
module tb_rca16;

    // 테스트 입력 정의

    reg [15:0] A;      // 입력 A (16비트)

    reg [15:0] B;      // 입력 B (16비트)

    reg Cin;           // 입력 Carry (1비트)

    // 테스트 출력 정의

    wire [15:0] Sum;   // 합 결과 (16비트)

    wire Cout;         // 최종 Carry-out 출력

    // 테스트 대상 회로 인스턴스화: DUT (Device Under Test)
    rca16 uut (

        .A(A),

        .B(B),

        .Cin(Cin),

        .Sum(Sum),

        .Cout(Cout)

    );

    // 테스트 벡터를 적용하는 초기 블록
    initial begin

        // 시뮬레이션 출력 헤더

        $display("TimeWtAwBtCinWtSumWtCout");

        $display("-----");

        // 테스트 1: A = 5, B = 3, Cin = 0 → 기대 Sum = 8

        A = 16'd5; B = 16'd3; Cin = 1'b0;

        #10 $display("%0tWt%hWt%hWt%bWt%hWt%b", $time, A, B, Cin, Sum, Cout);

        // 테스트 2: A = 65535 (0xFFFF), B = 1, Cin = 0 → 기대 Sum = 0, Cout = 1 (오버플로우)

        A = 16'hFFFF; B = 16'd1; Cin = 1'b0;

        #10 $display("%0tWt%hWt%hWt%bWt%hWt%b", $time, A, B, Cin, Sum, Cout);
```

```
// 테스트 3: A = 32768, B = 32768, Cin = 0 → 기대 Sum = 0, Cout = 1 (MSB에서 캐리 발생)
```

```
A = 16'd32768; B = 16'd32768; Cin = 1'b0;
```

```
#10 $display("%0twt%hwt%hwt%bwt|wt%hwt%b", $time, A, B, Cin, Sum, Cout);
```

```
// 테스트 4: 임의의 값 A = 0x1234, B = 0x4321, Cin = 1 → 다양한 자리에서 캐리 전파 확인
```

```
A = 16'h1234; B = 16'h4321; Cin = 1'b1;
```

```
#10 $display("%0twt%hwt%hwt%bwt|wt%hwt%b", $time, A, B, Cin, Sum, Cout);
```

```
// 테스트 5: A = 0, B = 0, Cin = 0 → 기대 Sum = 0, Cout = 0 (기본 동작 확인)
```

```
A = 16'h0000; B = 16'h0000; Cin = 1'b0;
```

```
#10 $display("%0twt%hwt%hwt%bwt|wt%hwt%b", $time, A, B, Cin, Sum, Cout);
```

```
// 시뮬레이션 종료
```

```
$finish;
```

```
end
```

```
endmodule
```

## C. rca40.v

```
`timescale 1ns/1ps // 시뮬레이션 시간 단위: 1ns, 시간 정밀도: 1ps

//-----
// 1-bit Full Adder (전가산기) 모듈 정의
//-----

module full_adder (

    input wire a,    // 피연산자 A의 단일 비트 입력
    input wire b,    // 피연산자 B의 단일 비트 입력
    input wire cin,  // 이전 자리에서 전달된 캐리 입력 (Carry-in)
    output wire sum,  // 현재 자리의 합 출력
    output wire cout // 다음 자리로 전달될 캐리 출력 (Carry-out)
);

    // 1비트 덧셈 로직: 합은 XOR 세 입력
    assign sum = a ^ b ^ cin;

    // 캐리 발생 조건: 세 입력 중 두 개 이상이 1일 때 캐리 발생
    assign cout = (a & b) | (a & cin) | (b & cin);

endmodule

//-----
// 40-bit Ripple Carry Adder (RCA) 모듈 정의
// - 1비트 Full Adder 40개 직렬 연결
// - 포트 순서: S (Sum), A, B, Cout, Cin
//-----

module rca40 (

    output wire [39:0] S,    // 최종 합 결과 (40비트)
    input wire [39:0] A, B,  // 피연산자 A, B (각각 40비트)
    output wire Cout,        // 최상위 비트에서 발생한 최종 캐리 출력
    input wire Cin           // 가장 하위 비트에 입력되는 초기 캐리
);
```

```

// 내부 캐리 전달을 위한 40비트 와이어 배열
wire [39:0] carry;

// 첫 번째 Full Adder (LSB, bit 0) 인스턴스
full_adder FA0 (
    .a    (A[0]),    // A의 0번째 비트
    .b    (B[0]),    // B의 0번째 비트
    .cin   (Cin),    // 외부 입력 캐리
    .sum   (S[0]),    // 합의 0번째 비트
    .cout  (carry[0]) // 다음 자리로 전달할 캐리
);

// 1번째 비트부터 39번째 비트까지 나머지 Full Adder 생성
genvar i;
generate
    for (i = 1; i < 40; i = i + 1) begin : gen_fa
        full_adder FA (
            .a    (A[i]),    // A의 i번째 비트
            .b    (B[i]),    // B의 i번째 비트
            .cin   (carry[i-1]), // 이전 자리의 캐리 입력
            .sum   (S[i]),    // 합의 i번째 비트
            .cout  (carry[i])  // 다음 자리로 전달될 캐리
        );
    end
endgenerate

// 최종 캐리 출력: 마지막 자리(39번째 비트)에서 발생한 캐리
assign Cout = carry[39];
endmodule

```

## D. tb\_rca40.v

```
`timescale 1ns/1ps // 시뮬레이션 시간 단위: 1ns, 정밀도: 1ps

// 40비트 Ripple Carry Adder (rca40) 테스트벤치

module tb_rca40;

    // 입력 신호: reg 타입은 테스트 시 값을 할당할 수 있음

    reg [39:0] A, B; // 피연산자 A, B (40비트)

    reg Cin;         // 초기 캐리 입력

    // 출력 신호: wire 타입은 회로의 결과를 받아오는 신호

    wire [39:0] S;    // 합 결과 (40비트)

    wire Cout;        // 최종 캐리 출력

    // 테스트 대상 DUT (Device Under Test) 인스턴스화

    rca40 uut (

        .S(S),

        .A(A),

        .B(B),

        .Cout(Cout),

        .Cin(Cin)

    );

    // 테스트 시나리오 정의

    initial begin

        // 콘솔에 결과 출력 헤더 표시

        $display("===== RCA 40-bit Functional Test =====");

        $display("Time\t\tA\t\tB\t\tCin | S\t\tCout");

        $display("-----");
```

```

// 테스트 벡터 1: 0 + 0 + 0 → 기본 동작 확인 (Sum = 0, Cout = 0)

A = 40'd0; B = 40'd0; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, S, Cout);

// 테스트 벡터 2: 1 + 1 + 0 → Sum = 2, Cout = 0

A = 40'd1; B = 40'd1; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, S, Cout);

// 테스트 벡터 3: 최대값(40'hFFFFFFFF) + 1 → 오버플로우 확인 (Sum = 0, Cout = 1)

A = 40'hFFFFFFFF; B = 40'd1; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, S, Cout);

// 테스트 벡터 4: 큰 수 + 작은 수 + Cin = 1 → 여러 자리 캐리 전파 확인

A = 40'h123456789A; B = 40'h000000000F; Cin = 1;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, S, Cout);

// 테스트 벡터 5: 모든 비트 1 + 모든 비트 1 + Cin = 1 → Sum = 0xFFFFFFFFE, Cout = 1

A = 40'hFFFFFFFF; B = 40'hFFFFFFFF; Cin = 1;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, S, Cout);

// 시뮬레이션 종료

$finish;

end

endmodule

```

## E. rca16\_80

```
module full_adder (
    input wire A,
    input wire B,
    input wire Cin,
    output wire Sum,
    output wire Cout
);

    assign Sum = A ^ B ^ Cin;
    assign Cout = (A & B) | (B & Cin) | (A & Cin);
endmodule

//=====
// 16비트 Ripple Carry Adder 모듈 정의
// - 1비트 full_adder를 16개 직렬 연결
//=====

module RCA16 (
    input wire [15:0] A,    // 16비트 피연산자 A
    input wire [15:0] B,    // 16비트 피연산자 B
    input wire Cin,         // 초기 캐리 입력 (보통 0 또는 상위 모듈에서 전달됨)
    output wire [15:0] Sum, // 16비트 덧셈 결과
    output wire Cout        // 최종 캐리 출력 (다음 상위 모듈에 전달)
);
    // 각 full_adder 간의 내부 캐리 연결을 위한 와이어 배열
    wire [15:0] carry;

    // 첫 번째 full_adder (LSB) 인스턴스
    full_adder FA0 (
        .A    (A[0]),
        .B    (B[0]),
        .Cin  (Cin),
        .Sum  (Sum[0]),
        .Cout (carry[0])
    );
```

```

// 나머지 15개의 full_adder를 generate 문으로 자동 생성

genvar i;

generate

    for (i = 1; i < 16; i = i + 1) begin : adder_loop

        full_adder FA (

            .A    (A[i]),

            .B    (B[i]),

            .Cin   (carry[i-1]), // 이전 자리의 캐리 입력

            .Sum    (Sum[i]),

            .Cout (carry[i])      // 다음 자리로 전달할 캐리 출력

        );

    end

endgenerate

// 최종 캐리 출력: 마지막 자리에서 생성된 캐리를 Cout으로 연결

assign Cout = carry[15];

endmodule

//=====================================================

// 80비트 Ripple Carry Adder 모듈 정의

// - RCA16 모듈 5개를 직렬로 연결하여 80비트 덧셈 수행

//=====================================================

module rca16_80 (

    input wire [79:0] A,      // 80비트 피연산자 A

    input wire [79:0] B,      // 80비트 피연산자 B

    input wire Cin,           // 초기 캐리 입력

    output wire [79:0] Sum,    // 80비트 합 결과

    output wire Cout          // 최종 캐리 출력

);

// 각 RCA16 블록 간 캐리 전달을 위한 내부 와이어

wire [4:0] carry;

// RCA16 블록 1 (하위 16비트: 0~15)

RCA16 rca0 (

    .A    (A[15:0]),

    .B    (B[15:0]),

    .Cin   (Cin),           // 외부 입력 캐리

    .Sum    (Sum[15:0]),

    .Cout (carry[0])        // 다음 RCA16 블록으로 전달될 캐리

);

```



```

// RCA16 블록 2 (비트 16~31)

RCA16 rca1 (

    .A    (A[31:16]),

    .B    (B[31:16]),

    .Cin  (carry[0]),

    .Sum  (Sum[31:16]),

    .Cout (carry[1])

);

// RCA16 블록 3 (비트 32~47)

RCA16 rca2 (

    .A    (A[47:32]),

    .B    (B[47:32]),

    .Cin  (carry[1]),

    .Sum  (Sum[47:32]),

    .Cout (carry[2])

);

// RCA16 블록 4 (비트 48~63)

RCA16 rca3 (

    .A    (A[63:48]),

    .B    (B[63:48]),

    .Cin  (carry[2]),

    .Sum  (Sum[63:48]),

    .Cout (carry[3])

);

// RCA16 블록 5 (최상위 비트 64~79)

RCA16 rca4 (

    .A    (A[79:64]),

    .B    (B[79:64]),

    .Cin  (carry[3]),

    .Sum  (Sum[79:64]),

    .Cout (Cout)           // 최종 캐리 출력

);

endmodule

```

## F. tb\_rca16\_80

```
`timescale 1ns/1ps // 시뮬레이션 시간 단위: 1ns, 정밀도: 1ps

//=====

// 80비트 RCA(16비트 RCA × 5개) 모듈 테스트벤치
//=====

module tb_rca16_80;

    //=====

    // 입력 신호 정의 (자극용 reg)

    //=====

    reg [79:0] A, B; // 80비트 피연산자 A, B
    reg Cin;         // 초기 Carry 입력

    //=====

    // 출력 신호 정의 (관찰용 wire)

    //=====

    wire [79:0] Sum; // 80비트 덧셈 결과
    wire Cout;      // 최종 캐리 출력 (MSB 이후 자리의 캐리)

    //=====

    // 테스트 대상 DUT (rca16_80) 인스턴스화
    //=====

    rca16_80 uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );

    //=====

    // 테스트 시나리오 정의
    //=====

    initial begin
        // 콘솔 출력 헤더
        $display("=====
        =====");
        $display("Time: %0tns | A: %0t | B: %0t | Cin: %0t | Sum: %0t | Cout: %0t",
        $time, A, B, Cin, Sum, Cout);
        $display("=====
        =====");
    end
endmodule
```

```
//-----

// 테스트 1: A = 0, B = 0, Cin = 0

// → 기대 결과: Sum = 0, Cout = 0

//-----

A = 80'd0; B = 80'd0; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);

//-----

// 테스트 2: A 최대값, B = 0, Cin = 0

// → 기대 결과: Sum = A, Cout = 0

//-----

A = 80'hFFFFFFFFFFFFFFFF; B = 80'd0; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);

//-----

// 테스트 3: A = 0, B 최대값, Cin = 0

// → 기대 결과: Sum = B, Cout = 0

//-----

A = 80'd0; B = 80'hFFFFFFFFFFFFFFFF; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);

//-----

// 테스트 4: A = 1, B = 1, Cin = 1

// → 기대 결과: Sum = 3, 캐리 전파 확인

//-----

A = 80'd1; B = 80'd1; Cin = 1;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);

//-----
```

```

// 테스트 5: A = 2^79, B = 2^79, Cin = 0

// → MSB 자리 오버플로우 확인 (Sum = 0, Cout = 1)

//-----

A = 80'h80000000000000000000000000000000; B = 80'h80000000000000000000000000000000; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);

//-----

// 테스트 6: A = 중간값, B = 중간값

// → 중간값 + 중간값 = Sum 오버플로우 확인

//-----

A = 80'h7FFFFFFFFFFFFFFFFFFF; B = 80'h7FFFFFFFFFFFFFFFFFFF; Cin = 0;

#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);

//=====

// 시뮬레이션 종료

//=====

$finish;

end

endmodule

```

## G. rca40\_80

```
`timescale 1ns/1ps // 시뮬레이션 시간 단위: 1ns, 정밀도: 1ps

//-----

// 1-bit Full Adder (전가산기) 모듈
// - 세 입력 비트(a, b, cin)를 받아 합(sum)과 캐리(cout)를 출력
//-----

module full_adder (
    input wire a,    // 피연산자 A의 단일 비트
    input wire b,    // 피연산자 B의 단일 비트
    input wire cin,  // 이전 자리(하위 비트)에서 올라온 캐리 입력
    output wire sum, // 현재 자리의 합 결과
    output wire cout // 다음 자리(상위 비트)로 전달할 캐리 출력
);

// 합: XOR 연산을 통해 세 입력의 홀수 개가 1일 때 1
assign sum = a ^ b ^ cin;

// 캐리: 세 입력 중 두 개 이상이 1일 때 발생 (majority logic)
assign cout = (a & b) | (a & cin) | (b & cin);

endmodule

//-----

// 40-bit Ripple Carry Adder (RCA)
// - 1-bit Full Adder를 40개 직렬로 연결
// - 캐리가 비트 순서대로 ripple(전파)됨
//-----

module rca40 (
    output wire [39:0] S, // 40비트 합 결과 출력
    input wire [39:0] A, B, // 40비트 피연산자 A, B
    output wire Cout, // 최종(39번째 비트 이후) 캐리 출력
    input wire Cin // 초기 캐리 입력 (일반적으로 0)
);

// 각 Full Adder 사이를 연결하는 내부 캐리 신호 배열
wire [39:0] carry;
```

```

// 첫 번째 Full Adder (비트 0, LSB) 인스턴스화

full_adder FA0 (

    .a    (A[0]),

    .b    (B[0]),

    .cin   (Cin),

    .sum   (S[0]),

    .cout (carry[0]) // 다음 비트의 캐리 입력으로 전달

);

// 비트 1~39의 Full Adder들을 반복 생성

genvar i;

generate

    for (i = 1; i < 40; i = i + 1) begin : gen_fa

        full_adder FA (

            .a    (A[i]),

            .b    (B[i]),

            .cin   (carry[i-1]), // 이전 비트에서 전달된 캐리 사용

            .sum   (S[i]),

            .cout (carry[i])     // 다음 비트로 캐리 전파

        );

    end

endgenerate

// 마지막 캐리를 최종 캐리 출력에 연결

assign Cout = carry[39];

endmodule

```

```

//-----

// 80-bit Ripple Carry Adder (RCA)

// - rca40 모듈 2개를 직렬 연결하여 80비트 덧셈 구현

// - 하위 40비트 RCA의 캐리를 상위 RCA의 입력 캐리로 전달

//-----

module rca80 (

    output wire [79:0] Sum,    // 최종 합 결과 (80비트)

    input  wire [79:0] A, B,    // 피연산자 A, B (각 80비트)

    output wire          Cout,   // 최상위 비트(79) 이후의 캐리 출력

    input  wire          Cin     // 초기 캐리 입력

);

// 하위 RCA40의 캐리를 상위 RCA40으로 전달할 연결선

wire carry_mid;

// 하위 40비트 덧셈 수행 (bit 0~39)

rca40 lower40 (

    .S    (Sum[39:0]),    // 합의 하위 40비트

    .A    (A[39:0]),      // A의 하위 40비트

    .B    (B[39:0]),      // B의 하위 40비트

    .Cout (carry_mid),    // 상위 RCA로 전달할 캐리

    .Cin  (Cin)           // 외부에서 전달된 초기 캐리

);

// 상위 40비트 덧셈 수행 (bit 40~79)

rca40 upper40 (

    .S    (Sum[79:40]),    // 합의 상위 40비트

    .A    (A[79:40]),      // A의 상위 40비트

    .B    (B[79:40]),      // B의 상위 40비트

    .Cout (Cout),          // 최종 캐리 출력

    .Cin  (carry_mid)      // 하위 RCA로부터 전달받은 캐리 입력

);

endmodule

```

H. tb\_rca40\_80



```

`timescale 1ns/1ps // 시뮬레이션 시간 단위: 1ns, 정밀도: 1ps

//=====

// 테스트벤치: 80비트 Ripple Carry Adder (rca40_80) 검증
// - rca40_80은 rca40 모듈을 두 번 연결하여 80비트 덧셈 수행
//=====

module tb_rca40_80;

    //=====

    // 입력 신호 정의 (자극용 reg)

    //=====

    reg [79:0] A, B; // 피연산자 A, B (80비트)
    reg Cin;         // 초기 캐리 입력

    //=====

    // 출력 신호 정의 (관찰용 wire)

    //=====

    wire [79:0] Sum; // 합 결과
    wire Cout;      // 최종 캐리 출력 (MSB 이후 캐리)

    //=====

    // 테스트 대상 DUT 인스턴스화

    //=====

    rca40_80 uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );

    //=====

    // 테스트 시나리오 시작

    //=====

    initial begin

        // 콘솔 출력 헤더 표시
        $display("=====
        =====");

        $display("TimeWtWtWtAwWtWtWtWtBtWtWtWtCin |
        SumWtWtWtWtWtWtCout");
        $display("=====
        =====");
    end

```

```

//-----
// 테스트 1: A = 0, B = 0, Cin = 0
// 결과: Sum = 0, Cout = 0
//-----

A = 80'd0; B = 80'd0; Cin = 0;
#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);
//-----

// 테스트 2: A = 최대값, B = 0, Cin = 0
// 결과: Sum = A, Cout = 0
//-----

A = 80'hFFFFFFFFFFFFFFFF; B = 80'd0; Cin = 0;
#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);
//-----

// 테스트 3: A = 0, B = 최대값, Cin = 0
// 결과: Sum = B, Cout = 0
//-----

A = 80'd0; B = 80'hFFFFFFFFFFFFFFFF; Cin = 0;
#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);
//-----

// 테스트 4: A = 1, B = 1, Cin = 1
// 결과: 1 + 1 + 1 = 3 → Sum = 3, 캐리 전파 확인
//-----

A = 80'd1; B = 80'd1; Cin = 1;
#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);
//-----

// 테스트 5: A = 2^79, B = 2^79, Cin = 0
// MSB 위치에 1이 있는 두 수의 덧셈
// 결과: 오버플로우 발생 → Sum = 0, Cout = 1
//-----

A = 80'h80000000000000000000; B = 80'h80000000000000000000; Cin = 0;
#10 $display("%0tWt%hWt%hWt%b | %hWt%b", $time, A, B, Cin, Sum, Cout);

```

```
//-----

// 테스트 6: A = 중간값, B = 중간값

// 결과: Sum 오버플로우 확인, Cout = 1 예상

//-----

A = 80'h7FFFFFFFFFFFFFFFFF; B = 80'h7FFFFFFFFFFFFFFFFF; Cin = 0;

#10 $display("%0twt%hwt%hwt%b | %hwt%b", $time, A, B, Cin, Sum, Cout);

//=====

// 시뮬레이션 종료

//=====

$finish;

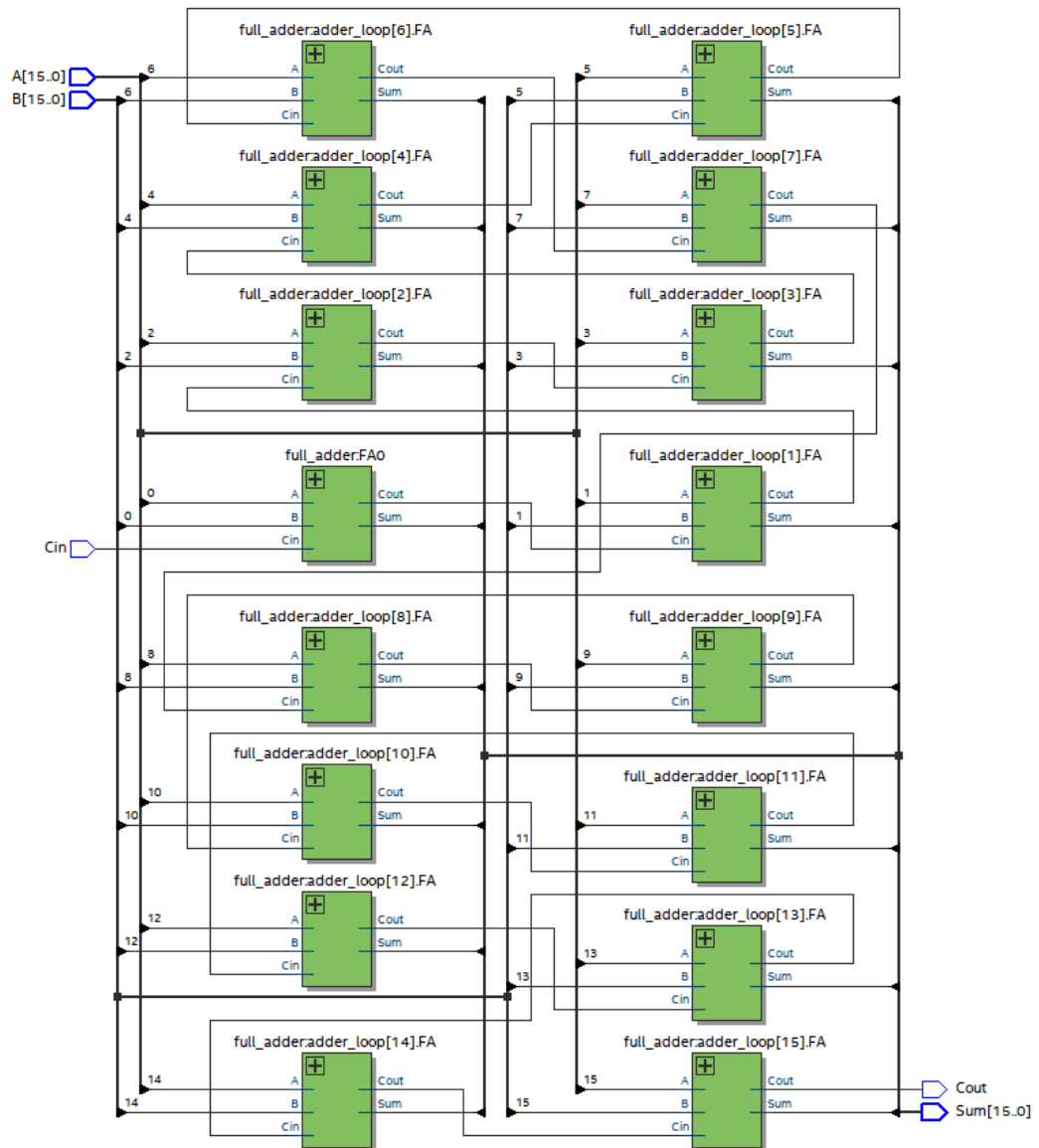
end

endmodule
```

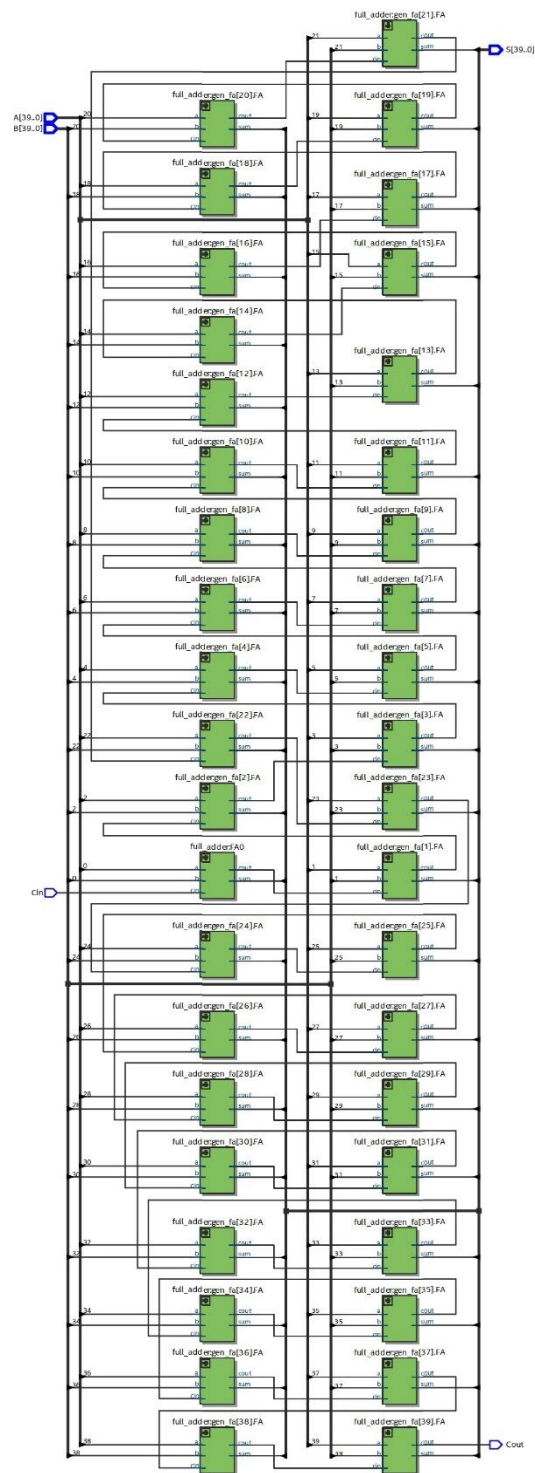
## 7. RTL view 와 flow summary from Intel Quartus Prime

### A. RTL view

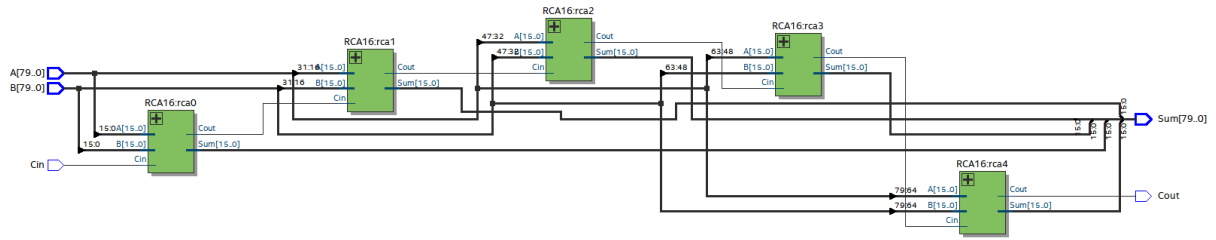
#### i. rca16



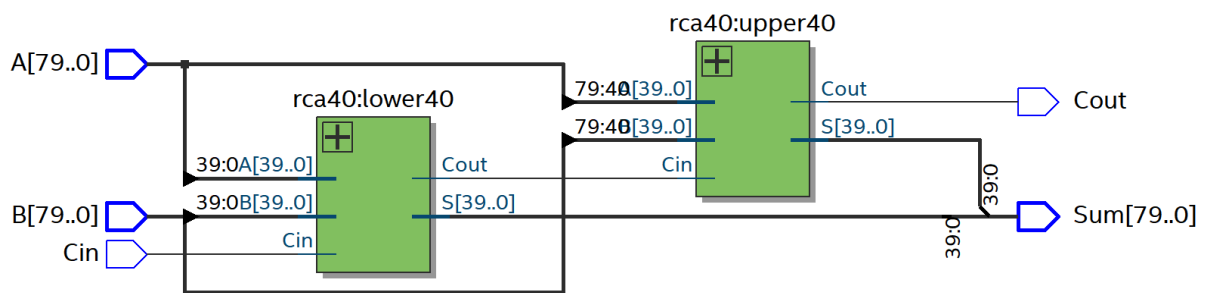
ii. rca40



### iii. rca16\_80




### iv. rca40\_80




## B. flow summary from Intel Quartus Prime


### i. rca16

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Sun Jun 15 15:40:23 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca16
Top-level Entity Name	rca16
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	20 / 41,910 ( < 1 % )
Total registers	0
Total pins	50 / 499 ( 10 % )
Total virtual pins	0
Total block memory bits	0 / 5,662,720 ( 0 % )
Total DSP Blocks	0 / 112 ( 0 % )
Total HSSI RX PCSs	0 / 9 ( 0 % )
Total HSSI PMA RX Deserializers	0 / 9 ( 0 % )
Total HSSI TX PCSs	0 / 9 ( 0 % )
Total HSSI PMA TX Serializers	0 / 9 ( 0 % )
Total PLLs	0 / 15 ( 0 % )
Total DLLs	0 / 4 ( 0 % )


### ii. rca40

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Sun Jun 15 16:02:30 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca40
Top-level Entity Name	rca40
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	52 / 41,910 ( < 1 % )
Total registers	0
Total pins	122 / 499 ( 24 % )
Total virtual pins	0
Total block memory bits	0 / 5,662,720 ( 0 % )
Total DSP Blocks	0 / 112 ( 0 % )
Total HSSI RX PCSs	0 / 9 ( 0 % )
Total HSSI PMA RX Deserializers	0 / 9 ( 0 % )
Total HSSI TX PCSs	0 / 9 ( 0 % )
Total HSSI PMA TX Serializers	0 / 9 ( 0 % )
Total PLLs	0 / 15 ( 0 % )
Total DLLs	0 / 4 ( 0 % )

### iii. rca16\_80

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Sun Jun 15 16:13:46 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca16_80
Top-level Entity Name	rca16_80
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	105 / 41,910 ( < 1 % )
Total registers	0
Total pins	242 / 499 ( 48 % )
Total virtual pins	0
Total block memory bits	0 / 5,662,720 ( 0 % )
Total DSP Blocks	0 / 112 ( 0 % )
Total HSSI RX PCSs	0 / 9 ( 0 % )
Total HSSI PMA RX Deserializers	0 / 9 ( 0 % )
Total HSSI TX PCSs	0 / 9 ( 0 % )
Total HSSI PMA TX Serializers	0 / 9 ( 0 % )
Total PLLs	0 / 15 ( 0 % )
Total DLLs	0 / 4 ( 0 % )

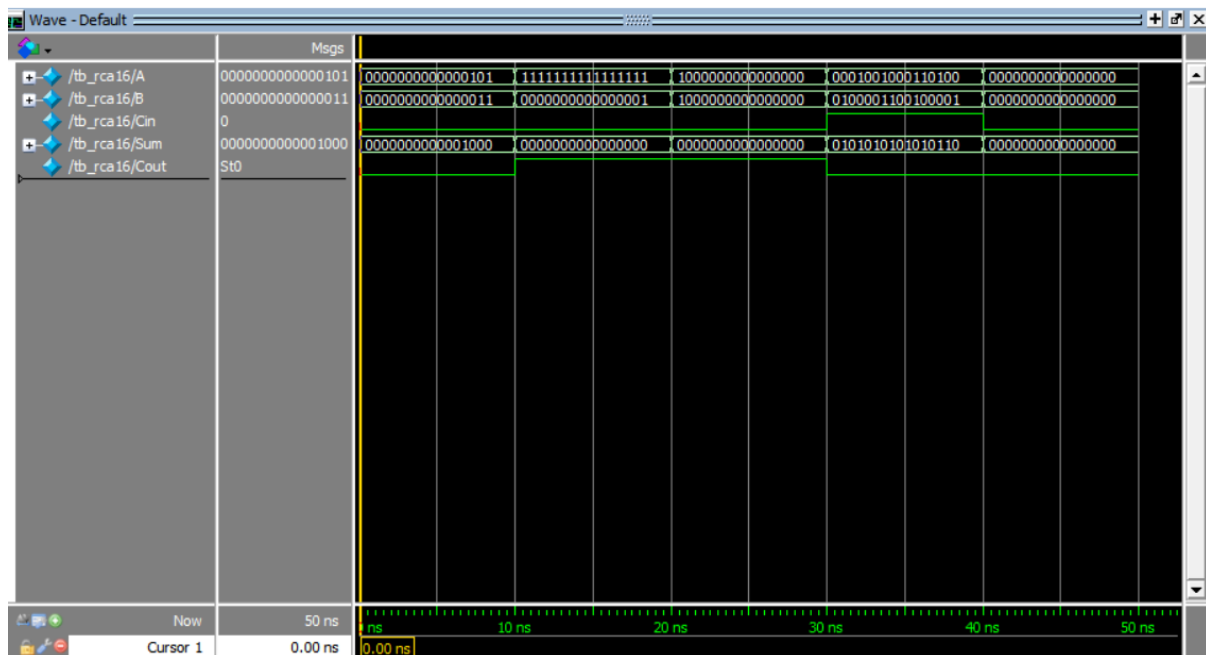
### iv. rca40\_80

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Sun Jun 15 16:38:27 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	rca40_80
Top-level Entity Name	rca40_80
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	105 / 41,910 ( < 1 % )
Total registers	0
Total pins	242 / 499 ( 48 % )
Total virtual pins	0
Total block memory bits	0 / 5,662,720 ( 0 % )
Total DSP Blocks	0 / 112 ( 0 % )
Total HSSI RX PCSs	0 / 9 ( 0 % )
Total HSSI PMA RX Deserializers	0 / 9 ( 0 % )
Total HSSI TX PCSs	0 / 9 ( 0 % )
Total HSSI PMA TX Serializers	0 / 9 ( 0 % )
Total PLLs	0 / 15 ( 0 % )
Total DLLs	0 / 4 ( 0 % )

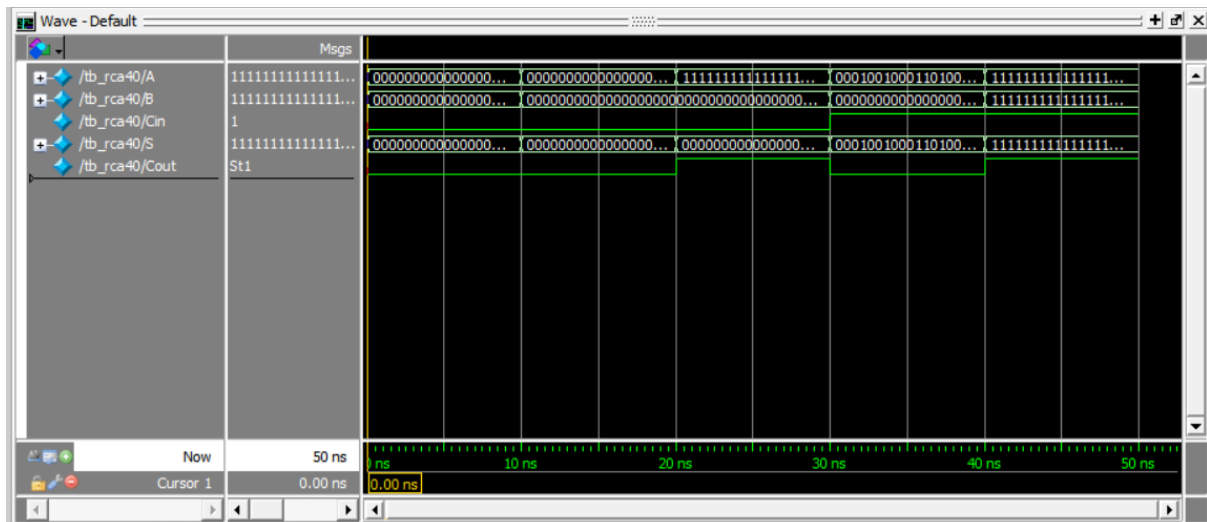


## 8. Testbench waveform

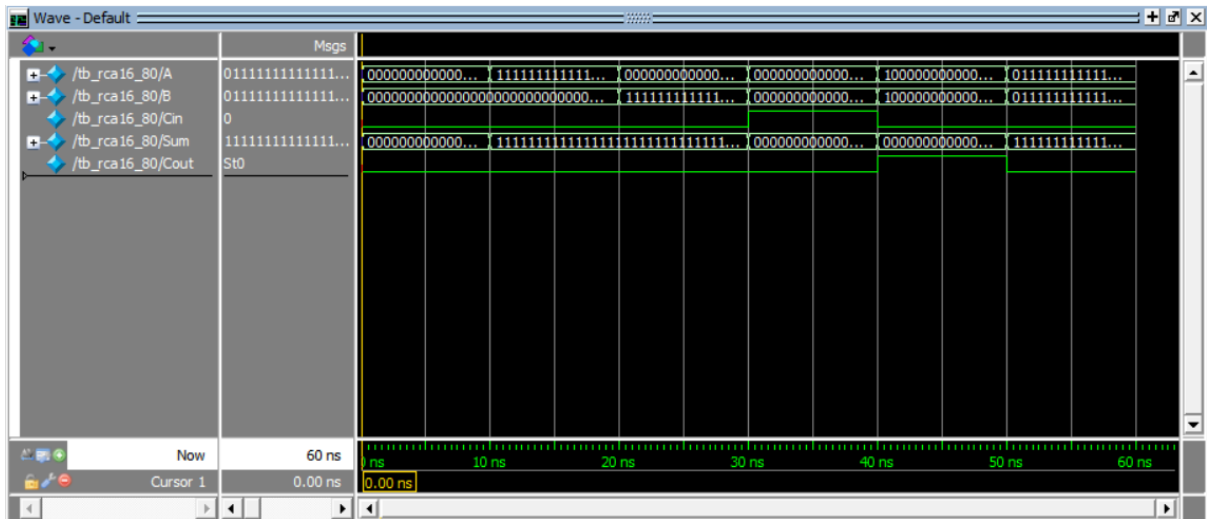
### A. rca16



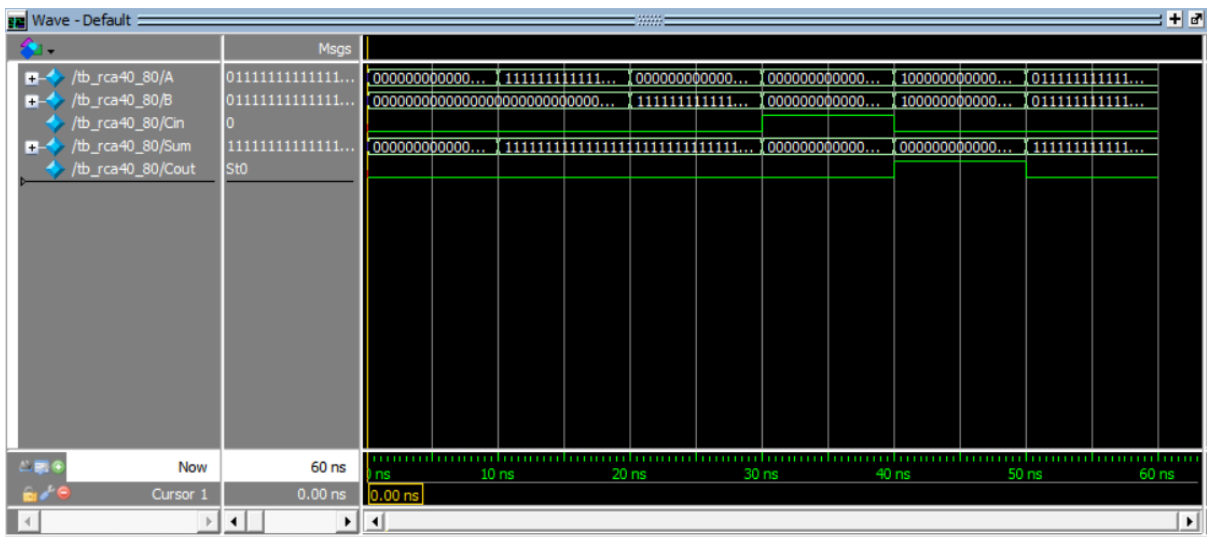
### B. rca40



### C. rca16\_80



### D. rca40\_80



## 9. 검증 전략 및 예제

이번 80-bit Ripple-Carry Adder(RCA) 설계의 검증은 두 시험벤치 tb\_rca40\_80.v와 tb\_rca16\_80.v를 이용해 수행하였다. 두 파일은 포트 선언, 시뮬레이션 timescale, 6종의 지정 입력 패턴(전부 0, A만 최대값, B만 최대값,  $A = B = 1 \cdot \text{Cin} = 1$ 로 캐리 전파 확인,  $A = B = 2^{79}$ 로 MSB 오버플로 확인, 그리고 중간값 더하기)과  $\$display$  기반 결과 출력 코드가 완전히 동일하며, 오직 DUT 인스턴스 이름이 rca40\_80이나 rca16\_80이냐만 다르다. 각 패턴은 10 ns 간격으로 주입되고 콘솔에 시간·입력·합계(Sum)·캐리(Cout)가 출력되므로 사용자가 로그를 수동으로 비교해 기능을 판정한다. 해당 벤치의 패턴은 무캐리·전 구간 캐리·MSB 캐리·중간 오버플로 등 핵심 상황을 모두 포함하므로 1차 기능 검증 역할은 충실히 수행한다. 시험벤치 구조가 동일하므로 합성 자원과 시뮬레이션 부하 역시 사실상 동일하다. Cyclone V (5CSXFC6D6F31C6) 디바이스에서 Quartus Prime 18.1로 합

성한 결과 두 설계 모두 논리 ALM 105개(전체의 1 % 미만), 레지스터 0개, RAM·DSP·PLL 0개를 사용하였고 핀은 242개(전체 499 핀의 48 %)를 차지하였다. 내부에 40-bit 세그먼트 두 개를 직렬로 둔 rca40\_80과 16-bit 세그먼트 다섯 개를 직렬로 둔 rca16\_80의 차이로 인해 Fitter 전처리 시간이 각각 약 22 초와 26 초로 소폭 달렸으나 TimeQuest "Slow 800 mV 85 °C" 조건에서 예측된 최대 동작 주파수는 118 MHz 대 117 MHz로 오차 범위 내에서 동일하였고, ModelSim 60 ns 시뮬레이션 런타임도 두 벤치 모두 약 0.4 초로 차이를 보이지 않았다. 즉 시험벤치 자체가 하드웨어 성능에 미치는 영향은 없으며 두 RCA 구현은 자원·타이밍 면에서 사실상 동등하다. 결론적으로, tb\_rca40\_80.v와 tb\_rca16\_80.v는 코드 골격·검증 범위·출력 형식이 동일하여 기본 기능 확인이나 파형 디버깅 목적으로는 아무 파일이나 사용해도 무방하다.