10주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231609 이름: 정희선

**1.**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input Cin,*    *input [3:0] A,*  *input [3:0] B,*  *output [4:0] C,*  *output [3:0] S*  *);*  *assign C[0] = Cin;*  *genvar i;*  *generate*  *for (i = 0; i < 4; i = i + 1) begin : adder*  *assign S[i] = (A[i] ^ B[i]) ^ C[i];*  *assign C[i + 1] = (C[i] & (A[i] ^ B[i])) | (A[i] & B[i]);*  *end*  *endgenerate*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg Cin;*  *reg [3:0] A;*  *reg [3:0] B;*  *wire [4:0] C;*  *wire [3:0] S;*  *boolean u\_boolean(*  *.Cin(Cin),*  *.A(A),*  *.B(B),*  *.C(C),*  *.S(S)*  *);*  *initial begin*  *Cin = 0;*  *A = 4'b0000;*  *B = 4'b0000;*  *end*  *always begin*  *#10 Cin = ~Cin;*  *#20 A = A + 1;*  *#40 B = B + 1;*  *end*  *initial begin*  *#200*  *$finish;*  *end*  *endmodule* |

스크린샷이(가) 표시된 사진

자동 생성된 설명

스크린샷, 다채로움이(가) 표시된 사진

자동 생성된 설명

4-bit binary parallel adder는 두 개의 4비트 이진수를 더하는 디지털 회로이다. 이 회로는 입력으로 Cin을 받고, 계산 결과로 Sum과 Carry-out을 출력한다. Generate 블록을 사용해 full adder의 동작을 반복적을 수행한다. 이 때 각 비트의 합과 다음 자리올림을 계산한다. 시뮬레이션 결과, Cin은 10ns마다 반전되며 변한다. 이는 덧셈 시작 시 자리올림의 여부를 테스트한다. 4비트 입력 값 A는 20ns마다 1씩 증가한다. B 역시 A와 독립적으로 동작하며 40ns마다 1씩 증가한다. 이 때 입력 값에 의해 자리올림이 발생하는 경우, 최종 자리올림인 C[4]에 그 결과가 반영된다.

**2.**

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input bin,*    *input [3:0] A,*  *input [3:0] B,*  *output [4:0] b,*  *output [3:0] D*  *);*  *assign b[0] = bin;*  *genvar i;*  *generate*  *for (i = 0; i < 4; i = i + 1) begin : substractor*  *assign D[i] = (A[i] ^ ~B[i]) ^ b[i];*  *assign b[i + 1] = (b[i] & (A[i] ^ ~B[i])) | (~A[i] & ~B[i]);*  *end*  *endgenerate*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg bin;*  *reg [3:0] A;*  *reg [3:0] B;*  *wire [4:0] b;*  *wire [3:0] D;*  *boolean u\_boolean(*  *.bin(bin),*  *.A(A),*  *.B(B),*  *.b(b),*  *.D(D)*  *);*  *initial begin*  *bin = 0;*  *A = 4'b0000;*  *B = 4'b0000;*  *end*  *always begin*  *#10 bin = ~bin;*  *#20 A = A + 1;*  *#40 B = B + 1;*  *end*  *initial begin*  *#200*  *$finish;*  *end*  *endmodule* |

스크린샷, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

4비트 parallel subtractor를 구현했다. 입력으로 4비트 값 A,B와 초기 빌림 값 bin을 받아 계산 결과로 D와 최종 빌림 값 b를 출력한다. 이는 full subtractor를 기반으로 설계되며, 4개의 full subtractor를 병렬로 연결하여 구상한다. Generate 블록을 사용해서 full subtractor의 동작을 반복적으로 수행한다. 시뮬레이션 결과, D는 각 자리의 차를 나타내고, b[4]는 최종 빌림 값을 나타낸다. 이 때 A<B인 경우 자리 빌림이 발생한다.

**3.**

|  |  |
| --- | --- |
| **fa module** | **Padd module** |
| *module fa(*  *input x, y, cin,*  *output s, cout*  *);*  *assign s = (x ^ y) ^ cin;*  *assign cout = (cin & (x ^ y)) | (x & y);*  *endmodule* | *module padd(*  *input[3:0] a, b,*  *input cin,*  *output[3:0] s,*  *output cout*  *);*    *wire c1, c2, c3;*    *fa fa0(a[0], b[0], cin, s[0], c1);*  *fa fa1(a[1], b[1], c1, s[1], c2);*  *fa fa2(a[2], b[2], c2, s[2], c3);*  *fa fa3(a[3], b[3], c3, s[3], cout);*  *endmodule* |

|  |  |
| --- | --- |
| **Design Source** | **Test Bench** |
| *`timescale 1ns / 1ps*  *module boolean(*  *input Cin,*    *input [3:0] A,*  *input [3:0] B,*  *output Cout,*  *output [3:0] Sum*  *);*    *wire k;*  *wire [3:0] z, S;*    *padd rca0(A, B, Cin, z, k);*  *assign Cout = (z[3]&z[2])|(z[3]&z[1])|k;*  *assign S = {1'b0, Cout, Cout, 1'b0};*  *padd rca1(z, S, 0, Sum);*  *endmodule* | *`timescale 1ns / 1ps*  *module boolean\_tb;*  *reg Cin;*  *reg [3:0] A;*  *reg [3:0] B;*  *wire Cout;*  *wire [3:0] Sum;*  *boolean u\_boolean(*  *.Cin(Cin),*  *.A(A),*  *.B(B),*  *.Cout(Cout),*  *.Sum(Sum)*  *);*  *initial begin*  *Cin = 0;*  *A = 4'b0000;*  *B = 4'b0000;*  *end*  *always begin*  *#10 Cin = ~Cin; // Cin을 10ns마다 토글*  *#30 A = A + 4; // A를 20ns마다 1 증가*  *#60 B = B + 6; // B를 40ns마다 1 증가*  *end*  *initial begin*  *#300*  *$finish;*  *end*  *endmodule* |

스크린샷, 텍스트, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

먼저 fa모듈은 1 bit Full adder를 구현한 코드이다. 따라서 입력으로 두 비트와 자리올림을 받아서 합과 자리올림을 출력한다. Padd 모듈은 4bit parallel adder를 구현한 코드이다. Fa 모듈 4개를 사용하여 4비트 덧셈을 처리한다. 따라서 4비트 입력 두개와 자리올림 입력을 받아 합과 최종 자리올림을 출력한다. 최종적으로 구현한 BCD adder는 두 개의 padd 모듈과 보정 논리를 추가하여 구현했다. 먼저 첫번째 padd 모듈에서는 A와 B의 덧셈을 수행하여 결과를 x와 자리올림 k로 가진다. 이후 BCD로 보정하는 과정에서 조건식 (z[3] & z[2]) | (z[3] & z[1]) | k을 통해 중간 덧셈 결과가 BCD 범위를 초과하는지 확인한다. 이때 보정이 필요한 경우 보정값 6을 더해준다. 두번째 padd 모듈에서 보정된 값을 덧셈하여 최종 결과 Sum과 자리올림 Cout을 생성한다. 시뮬레이션의 결과를 살펴보면, 10ns에 Cin = 0, A = 4'b0001, B = 4'b0001인 경우 Sum = 4'b0010, Cout = 0이 올바르게 출력된다. 30ns의 경우, Cin = 0, A = 4'b0101, B = 4'b0101이 되어 5 + 5 = 10이 출력되어야 하는데, 이는 BCD 범위를 초과하므로 보정 값 6(= 0110)이 더해져서 Sum = 4'b0000, Cout = 1이라는 결과가 올바르게 출력된다.

**4.**

4 bit binary parallel adder는 두 개의 4비트 이진수를 입력으로 받아 자리올림까지 계산하는 구조로, Cin 입력을 통해 이전 자리에서 발생한 자리올림을 추가적으로 계산할 수 있다. 시뮬레이션 결과, 두 입력의 덧셈 연산과 자리올림이 정확히 출력되었다. 다만 adder 설계에서 carry propagation delay가 시스템의 전체 처리 속도에 영향을 미칠 수 있다. 특히 더 큰 비트 크기의 adder를 설계하는 경우에는 ripple carry 구조의 지연 시간을 최소화하기 위해 carry lookahead adder와 같은 구조를 고려할 수 있다.

4 bit binary parallel subtractor는 입력된 두 4비트 이진수에서 뺄셈 연산을 수행하여 자리 내림을 계산하는 구조이다. 시뮬레이션 결과 뺄셈 연산과 borrow가 정확하게 동작하였다. 모든 경우에서 A-B의 결과가 출력되었고, 특히 A<B인 경우 borrow가 발생하여 음수 결과를 처리할 수 있었다. 이 경우도 마찬가지로 borrow propagation delay와 같은 문제가 발생할 수 있다. 다라서 carry save subtractor와 같은 구조를 검토할 수 있다.

BCD adder는 이진 덧셈 결과가 BCD 범위를 초과하는 경우 보정 값(6)을 더해 결과를 BCD 형식으로 유지하는 기능을 정확하게 수행하였다. 특히 보정 논리의 조건을 잘 정의하여 BCD 범위 내의 값으로 유지할 수 있었다. 다만 BCD adder는 보정논리가 추가된 구조이므로 연산 속도가 순수 binary adder보다 느릴 수 있다. 이를 개선하기 위한 동작 병렬화나 다양한 최적화를 고려할 필요가 있다.

**5.**

BCD 연산의 응용에 대해 알아볼 것이다. BCD 연산은 이진수와 달리 각 숫자를 10진수 단위로 처리하기 때문에 디지털 시스템에서 다양한 응용 분야에 사용된다. 먼저 디지털 디스플레이 시스템에 사용된다. 그 예시로는 계산기나 디지털 시계가 있다. 입력 값을 디스플레이에 출력할 때 BCD 연산을 통해 10진수를 유지할 수 있다. 이는 이진수를 다시 10진수로 변환하는 과정을 생략해도 되기 때문에 적합하다. 다음으로 상업 및 금융 시스템에서도 사용된다. 금융 시스템에서는 소수점 이하를 포함한 금액을 처리해야 하는데, BCD 연산은 10진수를 유지하며 연산을 수행하기 때문에 데이터의 정확성을 유지한다는 장점이 있어 금융 계산에서 널리 사용된다.