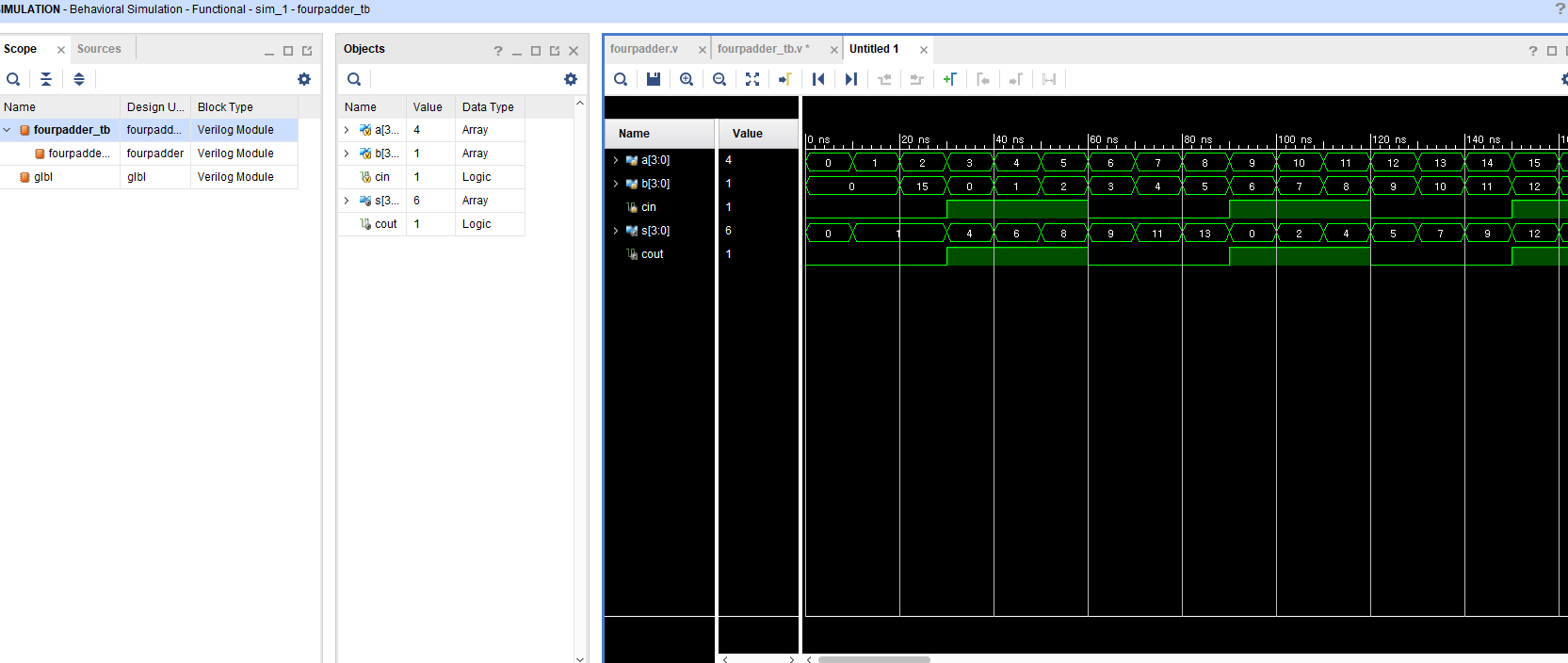
10주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20221549 이름: 김효림

**1.**

.................



4 bit Binary Parallel Adder의 simulation 결과

|  |  |
| --- | --- |
| Source code | Simulation code(tb) |
| module fourpadder(  input [3:0]a,  input [3:0]b,  input cin,  output [3:0]s,  output [3:0]c  );  assign s[0] = (a[0]^b[0])^cin;  assign c[0] = (a[0]&b[0]) | (cin&(a[0]^b[0]));  assign s[1] = (a[1]^b[1])^c[0];  assign c[1] = (a[1]&b[1]) | (c[0]&(a[1]^b[1]));  assign s[2] = (a[2]^b[2])^c[1];  assign c[2] = (a[2]&b[2]) | (c[1]&(a[2]^b[2]));  assign s[3] = (a[3]^b[3])^c[2];  assign c[3] = (a[3]&b[3]) | (c[2]&(a[3]^b[3]));  endmodule | module fourpadder\_tb;  reg [3:0] a,b;  reg cin;  wire [3:0] s;  wire cout;  fourpadder fourpadder\_l(a,b,cin,s,cout);  initial begin  a = 4'D0;  b = 4'D0;  cin = 1'b0;  end  always@(a or b or cin)begin  a<= #10 a + 4'D1;  b <= #20 a-4'D1;  cin <= #30 ~cin;  end  endmodule |

4 bit Binary Parallel Adder의 Verilog code

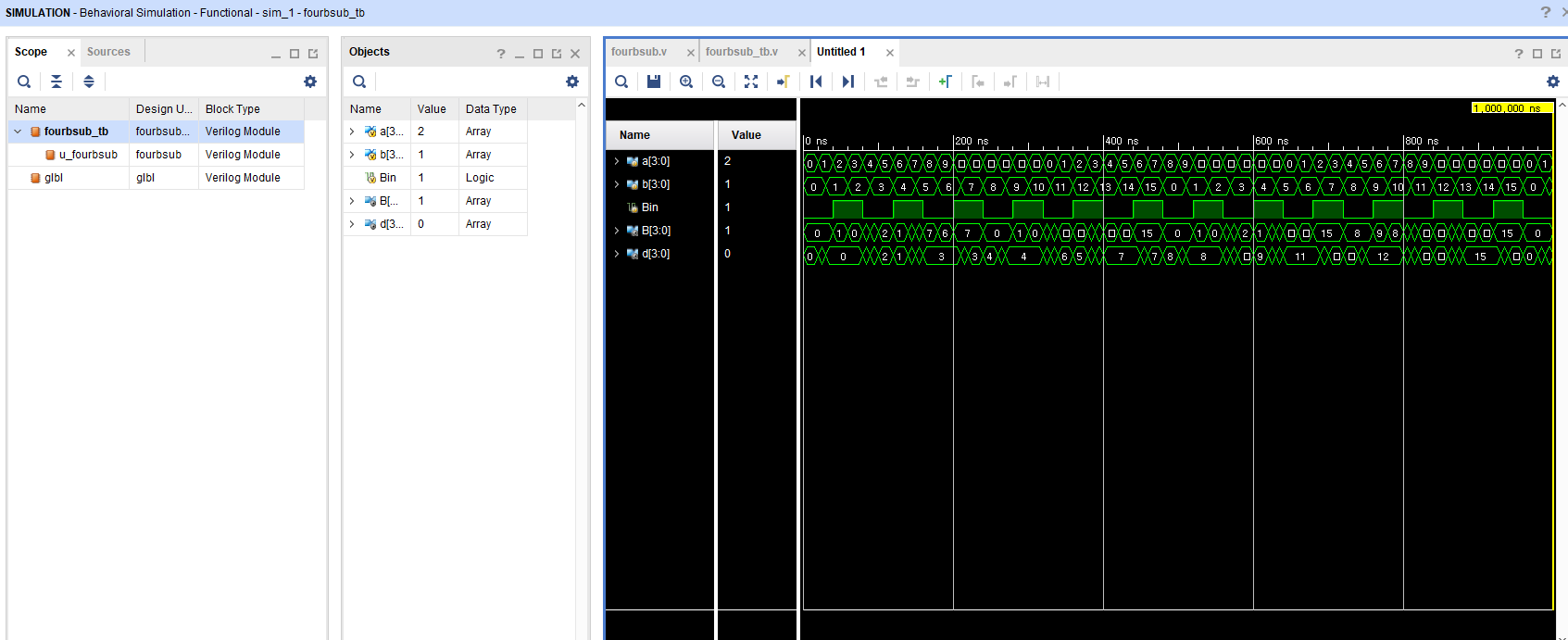
Verilog source code는 6주차 때 다뤘던 Full adder를 이용하여 만들 수 있었다. Full adder의 output 값들은 다음과 같은 수식을 가지고 있다. S = (A XOR B) XOR Cin, Cout = (Cin And (A XOR B)) OR (A AND B). 따라서 s배열 값은 s[i] = (a[i]^b[i])^c[i-1] (i = 0 to 3)와 같이 할당하고, c[i] = (a[i] ^ b[i]) | (c[i-1]&(a[i]^b[i]))와 같이 할당하여 구할 수 있었다. 이때 네 개의 가산기가 연결되어 있기 때문에, 0번 째를 제외한 나머지 Cin은 이전 단계에서 연산된 c[i-1]의 값을 이용했다.

만약 a가 0001이고 b가 0001, Cin이 0이었다면, 결과는 s는 0010, Cout은 0이 됐을 것이다.

................

**2.**

.......................



4 bit Binary Parallel Subtractor의 simulation 결과

|  |  |
| --- | --- |
| Source code | Simulation code(tb) |
| module fourbsub(  input [3:0] a,  input [3:0] b,  input Bin,  output [3:0] B,  output [3:0] d  );    assign d[0] = (a[0]^b[0])^Bin;  assign B[0] = (~a[0]&b[0]) | (Bin&~(a[0]^b[0]));  assign d[1] = (a[1]^b[1])^B[0];  assign B[1] = (~a[1]&b[1]) | (B[0]&~(a[1]^b[1]));  assign d[2] = (a[2]^b[2])^B[1];  assign B[2] = (~a[2]&b[2]) | (B[1]&~(a[2]^b[2]));  assign d[3] = (a[3]^b[3])^B[2];  assign B[3] = (~a[3]&b[3]) | (B[2]&~(a[3]^b[3]));  endmodule | module fourbsub\_tb;  reg [3:0] a, b;  reg Bin;  wire [3:0] B;  wire [3:0] d;  fourbsub u\_fourbsub(  .a(a),  .b(b),  .Bin(Bin),  .B(B),  .d(d)  );  initial begin  a = 4'b0000;  b = 4'b0000;  Bin = 1'b0;  end  always @(a or b) begin  a <= #20 a + 4'b0001;  b <= #30 b + 4'b0001;  Bin <= #40 ~Bin;  end  initial begin  #5120  $finish;  end  endmodule |

4 bit Binary Parallel Subtractor의 Verilog code

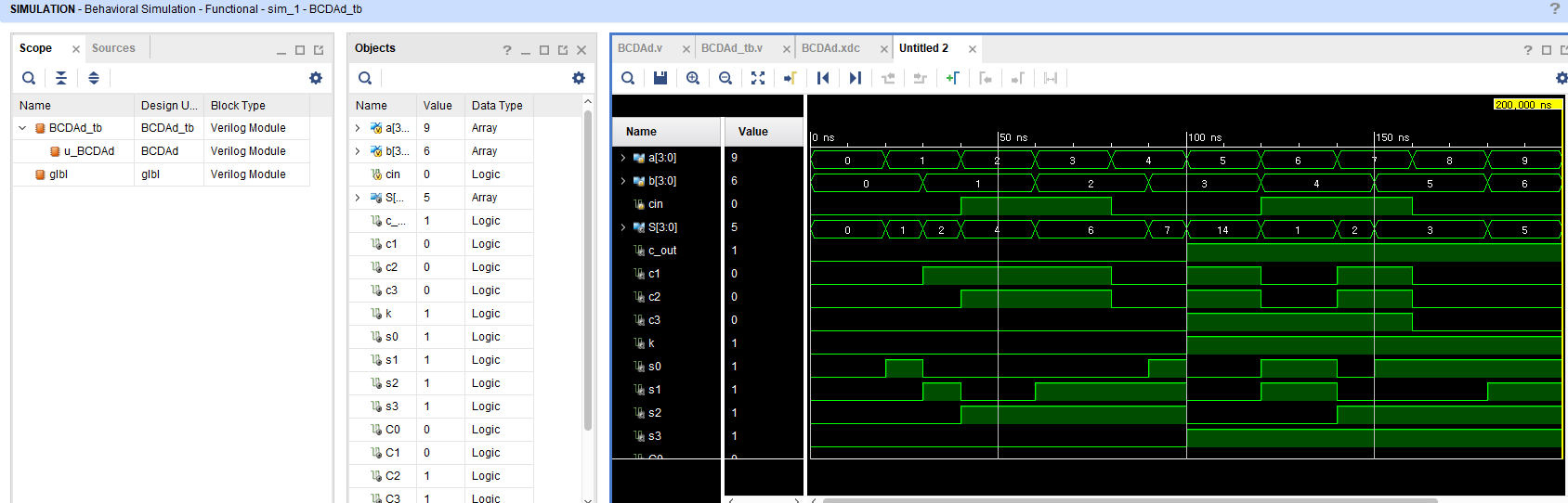
Verilog source code는 6주차 때 다뤘던 Full Subtractor를 이용하여 만들 수 있었다. Full Subtractor의 output 값들은 다음과 같은 수식을 가지고 있다. 이다. 따라서 d 배열은 d[i] = (a[i]^b[i])^B[i-1]와 같이 할당하고, B 배열은 B[i] = (~a[i]^b[i]) | (B[i-1] &~(a[i]^b[i])) 와 같이 할당하여 구할 수 있었다. ] (i = 0 to 3, i가 0일 때 Bin값 이용)

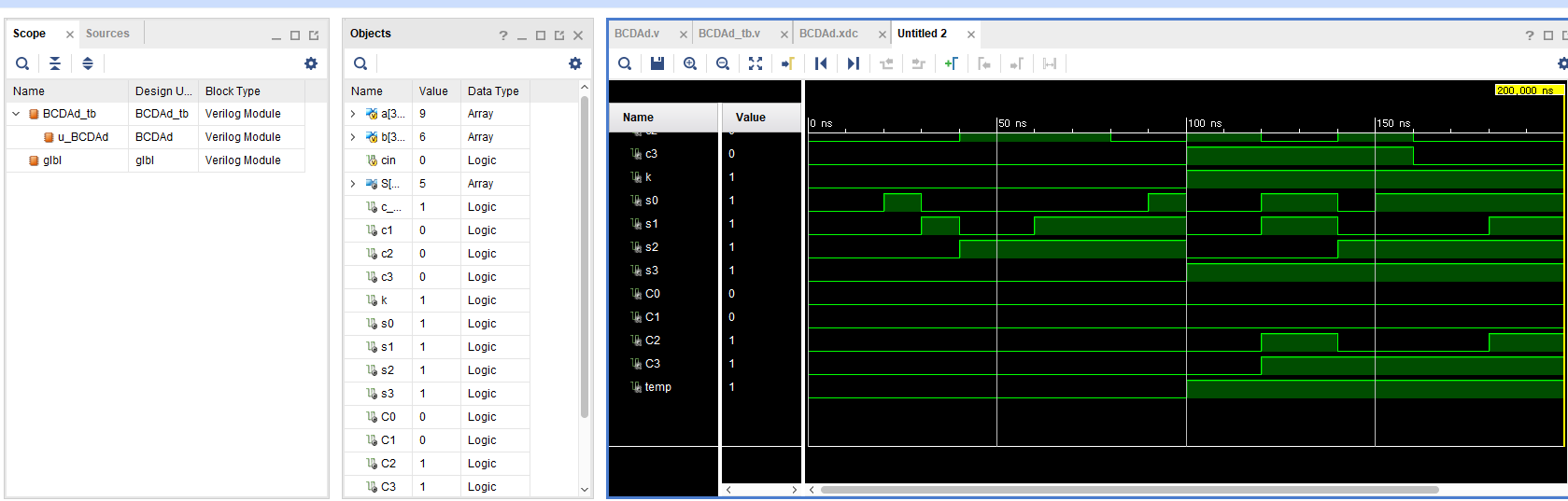
만약 a가 0010이고 b가 0001이고 Bin이 0이라면 차이 d는 0010이 되고, 빌림 B는 0이 될 것이다.

.........................

**3.**

.......................





BCD Adder의 simulation 결과

|  |  |
| --- | --- |
| Source code | Simulation code(tb) |
| module BCDAd(  input [3:0]a,b,  input cin,  output [3:0] S,  output c\_out  );    wire c1,c2,c3,k,s0,s1,s2,s3,C0,C1,C2,C3,temp;      assign c1 = (cin&(a[0]^b[0])) | (a[0]&b[0]);  assign c2 = (c1&(a[1]^b[1])) | (a[1]&b[1]);  assign c3 = (c2&(a[2]^b[2])) | (a[2]&b[2]);    assign k = c3 | (a[3] & b[2]) | (a[3] ^ b[3]);  assign s0 = a[0] ^ b[0] ^ cin;  assign s1 = a[1] ^ b[1] ^ c1;  assign s2 = a[2] ^ b[2] ^ c2;  assign s3 = a[3] ^ b[3] ^ c3;    assign temp = k | (s3 & s2) | (s3&s1);    assign C0 = 1'b0;  assign C1 = (C0 & (s0^1'b0)) | (s0&1'b0);  assign C2 = (C1 & (s1^temp)) | (s1&temp);  assign C3 = (C2 & (s2^temp)) | (s2&temp);  assign c\_out = temp;  assign S[0] = s0 ^ 1'b0 ^ C0;  assign S[1] = s1 ^ temp ^ C1;  assign S[2] = s2 ^ temp ^ C2;  assign S[3] = s3 ^ 1'b0 ^ C3;    endmodule | module BCDAd\_tb;  reg [3:0] a;  reg [3:0] b;  reg cin;  wire [3:0] S;  wire c\_out;  BCDAd u\_BCDAd(  .a(a),  .b(b),  .cin(cin),  .S(S),  .c\_out(c\_out)  );  initial begin  a = 4'b0000;  b = 4'b0000;  cin = 1'b0;  end  always @(a or b) begin  a <= #20 a + 4'b0001;  b <= #30 b + 4'b0001;  cin <= #40 ~cin;  end  initial begin  #200  $finish;  end  endmodule |

BCD Adder의 Verilog code

Source code에서 입력인 a와 b는 각각 4비트의 숫자를 의미하며, cin은 덧셈에서 올림 비트이다. BCD adder는 4 bit Binary Adder 두 개를 연결하여 만들 수 있다. 따라서 wire로 정의한 c1,c2,c3는 각 자릿수에서의 올림 비트를 저장하는 중간변수, k는 올림비트를 계산하는 중간 변수, [0:3]s는 각 자릿수에서의 합을 나타내는 중간변수를 의미한다. Temp는 최종 올림비트를 계산하는 중간변수이며, C0~C3는 각 자릿수에서의 올림 비트를 저장하는 레지스터이다. S[0]~S[3]은 각 자릿수에서의 BCD 합을 나타내는 출력벡터이고, c\_out은 최상위 비트에서의 올림 비트를 나타내는 출력이다.

각 c1~c3, s0~s4, C0~C3, S0~S3는 앞서 Full Adder에서 언급한 방식으로 할당을 하여 구할 수 있었다.

.........................

**4.**

.......................

4비트 가산기와 감산기는 1비트 전가산기와 전감산기를 4개 연결하여 구현할 수 있었다. BCD Adder를 보드로 확인하기 위해 I/O ports를 연결하는 부분에서 막혔었는데, wire로 선언한 변수를 테스트 벤치에서 삭제해 주니 해결이 되었다. Ports에 연결이 될 변수만 test bench에 넣었어야 했는데, wire형을 처음 source에 선언하며 문제가 발생했던 것 같다.

.........................

**5.**

.......................

베릴로그에서 vector의 표현은 input wire [3:0] a 같은 방식으로 사용할 수 있으며, 각각 [데이터 타입] [[최상위 비트 : 최하위 비트]] [변수명] 형태로 나타낼 수 있다. 베릴로그에서 배열은 변수명 뒤에 배열 범위를 지정하게 된다. [데이터 타입][변수명][배열 범위] 꼴로 선언하게 된다. 예를 들면, input wire[3:0] a[3:0]과 같이 선언되는 것이다. 숫자의 표현은 a = 4’b1111 (binary), b = 4’d15 (decimal), c = 4’hF (hexdecimal)로 표현할 수 있다. 즉 다음과 같이 숫자의 비트 수, 사용할 진수, 숫자를 입력하는 것이다. [비트 수]’[진수][숫자] 외에도 Concatenation을 사용할 수 있는데, 이는 흔히 하는 기능 그대로 두 수를 연결시켜 준다. 이와 비슷하게 Replication은 특정 숫자를 여러 번 반복해서 연결해 준다. {반복횟수{number}}과 같은 형태로 작성된다.

.........................