**컴퓨터공학 설계 및 실험Ⅱ**

10주차 결과보고서

서강대학교 공학부 컴퓨터공학 전공

20171646 박태윤

**1. 4bit Binary Parallel Adder의 결과 및 Simulation 과정에 대해서 설명하시오.**

4bit Binary Parallel Adder를 구현하기 전 1 bit Full Adder를 Verilog로 구현하였다.

|  |
| --- |
| **adder1bit** |
| module adder1bit(A, B, Ci, S, Co);  input A, B, Ci;  output S, Co;  assign S = A^B^Ci;  assign Co = A&B | Ci&(A^B);  endmodule |

입력으로 A, B, Ci를 받으며 S와 Co를 출력하는 1 bit Full Adder이다. A와 B는 덧셈을 하고자 하는 1bit들을 의미하며 Ci는 입력으로 들어오는 Carry를 뜻한다. 출력 S는 합을 나타내며 Co는 출력으로 나오는 자리올림수, Carry out을 의미한다.

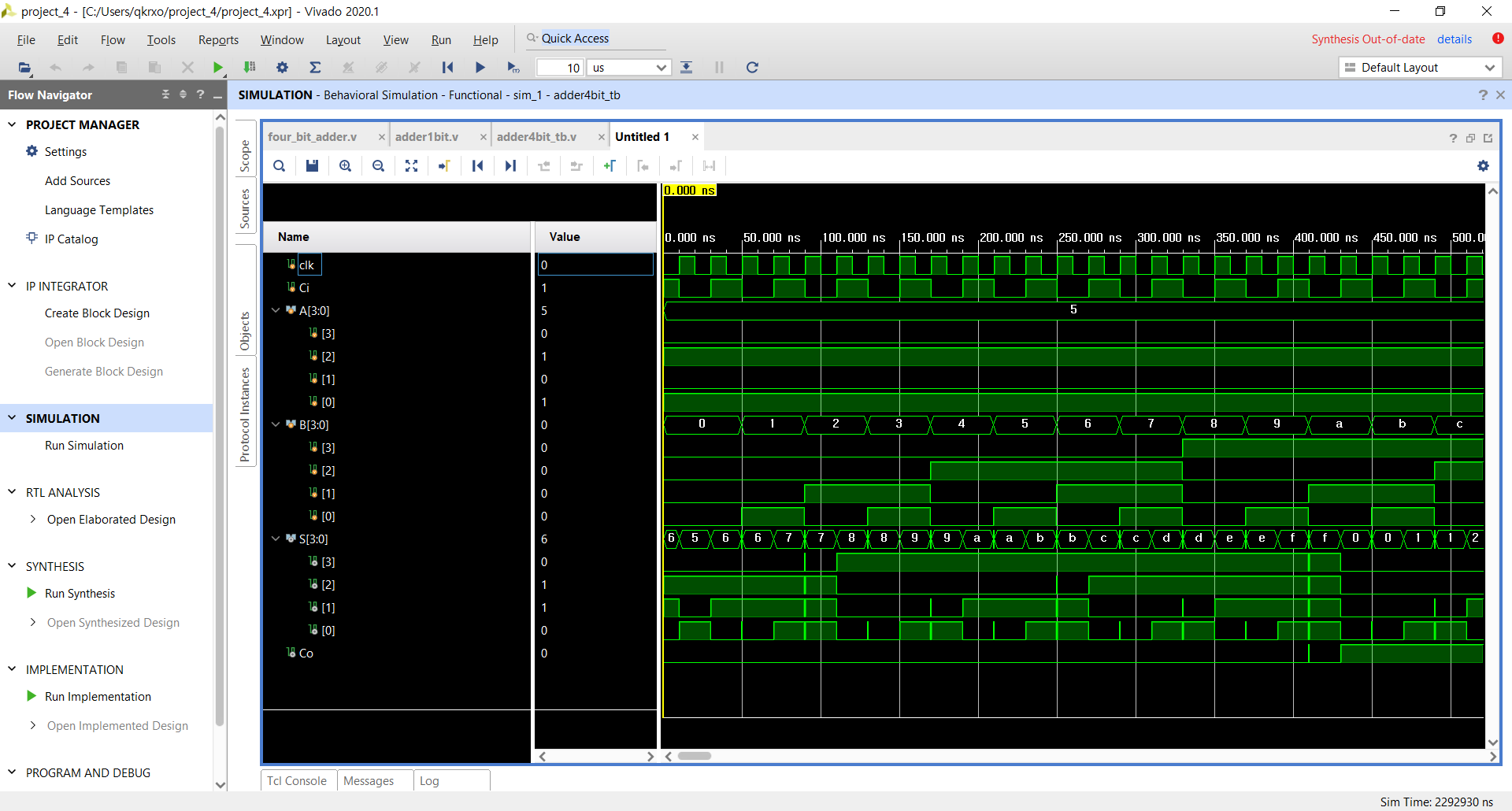
구현한 1 bit Full Adder를 이용하여 4bit Binary Parallel Adder를 다음과 같이 구현하였다.

|  |
| --- |
| **four\_bit\_adder** |
| `timescale 1ns / 1ps  module four\_bit\_adder(A, B, Ci, S, Co);  input [3:0] A, B; input Ci;  output [3:0] S; output Co;  wire [3:0] A, B, S; wire Ci, Co;  wire [2:0] C;  adder1bit add1(A[0], B[0], Ci, S[0], C[0]);  adder1bit add2(A[1], B[1], C[0], S[1], C[1]);  adder1bit add3(A[2], B[2], C[1], S[2], C[2]);  adder1bit add4(A[3], B[3], C[2], S[3], Co);  endmodule |
| **adder4bit\_tb** |
| `timescale 1ns / 1ps  module adder4bit\_tb();  reg clk, Ci; reg [3:0] A, B;  wire [3:0] S; wire Co;  four\_bit\_adder connect(A, B, Ci, S, Co);  initial begin  clk=0; A[3]=0; A[2]=1; A[1]=0; A[0]=1;  B[3]=0; B[2]=0; B[1]=0; B[0]=0; Ci=1; end  always clk = #10 ~clk;  always @(posedge clk) begin  Ci <= ~Ci; B[0] <= #39.999 ~B[0];  B[1] <= #79.999 ~B[1]; B[2] <= #159.999 ~B[2];  B[3] <= #319.999 ~B[3]; end    endmodule |



4비트의 두 수를 input [3:0] A, B와 같이 배열로 표현하였으며, 나오는 출력 S 또한 output [3:0] S으로 표현하였다. A(A[0], A[1], A[2], A[3])와 B(B[0], B[1], B[2], B[3])에 대하여 덧셈연산을 진행하는 역할을 하는 4 bit binary parallel adder를 다음과 같이 구현하였다. 마지막 비트 이전 모든 비트에서 나오는 Carry out이 다음 bit연산을 하는 full adder의 입력으로 들어가는 ripple carry adder이다. 출력으로 각 비트에서 덧셈 연산을 한 결과인 S(S[0], S[1], S[2], S[3])과 최종 Carry out인 Co가 나온다.

다음은 구현한 코드를 이용하여 Simulation을 한 결과이다.



테스트벤치 코드에서 A[3]=0; A[2]=1; A[1]=0; A[0]=1로 초기화하였기 때문에 A는 5를 나타낸다. B값과 Ci값이 변하면서 출력 S의 결과 또한 변화하는데, 이진수 덧셈이 제대로 이루어지는 것을 확인할 수 있다.

FPGA동작을 위해 다음과 같은 xdc코드를 추가하였다.

|  |
| --- |
| **four\_bit\_adder.xdc** |
| set\_property IOSTANDARD LVCMOS18 [get\_ports {S[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {S[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {S[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {S[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports Ci]  set\_property IOSTANDARD LVCMOS18 [get\_ports Co]  set\_property PACKAGE\_PIN F15 [get\_ports {S[0]}]  set\_property PACKAGE\_PIN F13 [get\_ports {S[1]}]  set\_property PACKAGE\_PIN F14 [get\_ports {S[2]}]  set\_property PACKAGE\_PIN F16 [get\_ports {S[3]}]  set\_property PACKAGE\_PIN J4 [get\_ports {A[0]}]  set\_property PACKAGE\_PIN L3 [get\_ports {A[1]}]  set\_property PACKAGE\_PIN K3 [get\_ports {A[2]}]  set\_property PACKAGE\_PIN M2 [get\_ports {A[3]}]  set\_property PACKAGE\_PIN K6 [get\_ports {B[0]}]  set\_property PACKAGE\_PIN J6 [get\_ports {B[1]}]  set\_property PACKAGE\_PIN L5 [get\_ports {B[2]}]  set\_property PACKAGE\_PIN L4 [get\_ports {B[3]}]  set\_property PACKAGE\_PIN Y16 [get\_ports Ci]  set\_property PACKAGE\_PIN E17 [get\_ports Co] |

이후 FPGA를 통해 결과를 확인하였다. FPGA동작을 위해서 다음과 같은 과정을 거쳤다.

Run Synthesis -> Run Implementation -> Constraints Wizard에서 포트 설정 -> Generate Bitstream -> Open Target, Auto Connect -> Program Device

FPGA에서 각 스위치와 LED가 나타내는 것을 다음과 같다 하자.

SW1=A[0], SW2=A[1], SW3=A[2], SW4=A[3]

SW5=B[0], SW6=B[1], SW7=B[2], SW8=B[3], SW9=Ci

LD1=S[0], LD2=S[1], LD3=S[2], LD4=S[3], LD5=Co

이때 예를 들어, 1001(2)+0011(2)+1=1101(2)와 같은 연산을 진행한다면

SW1=ON, SW2=OFF, SW3=OFF, SW4=ON, SW5=ON, SW6=ON, SW7=OFF, SW8=OFF, SW9=ON

와 같이 스위치를 조작을 한다. 결과로는

LD1=ON, LD2=OFF, LD3=ON, LD4=ON, LD5=OFF, LD(6~16)=OFF

을 얻을 수 있을 것이다.

**2. 4bit Binary Parallel Subtractor의 결과 및 Simulation과정에 대해서 설명하시오.**



위의 그림과 같이 4개의 1bit full subtractor를 연결하여 4bit binary parallel subtractor를 구현할 수 있다. 먼저 1bit full subtractor를 구현하였다.

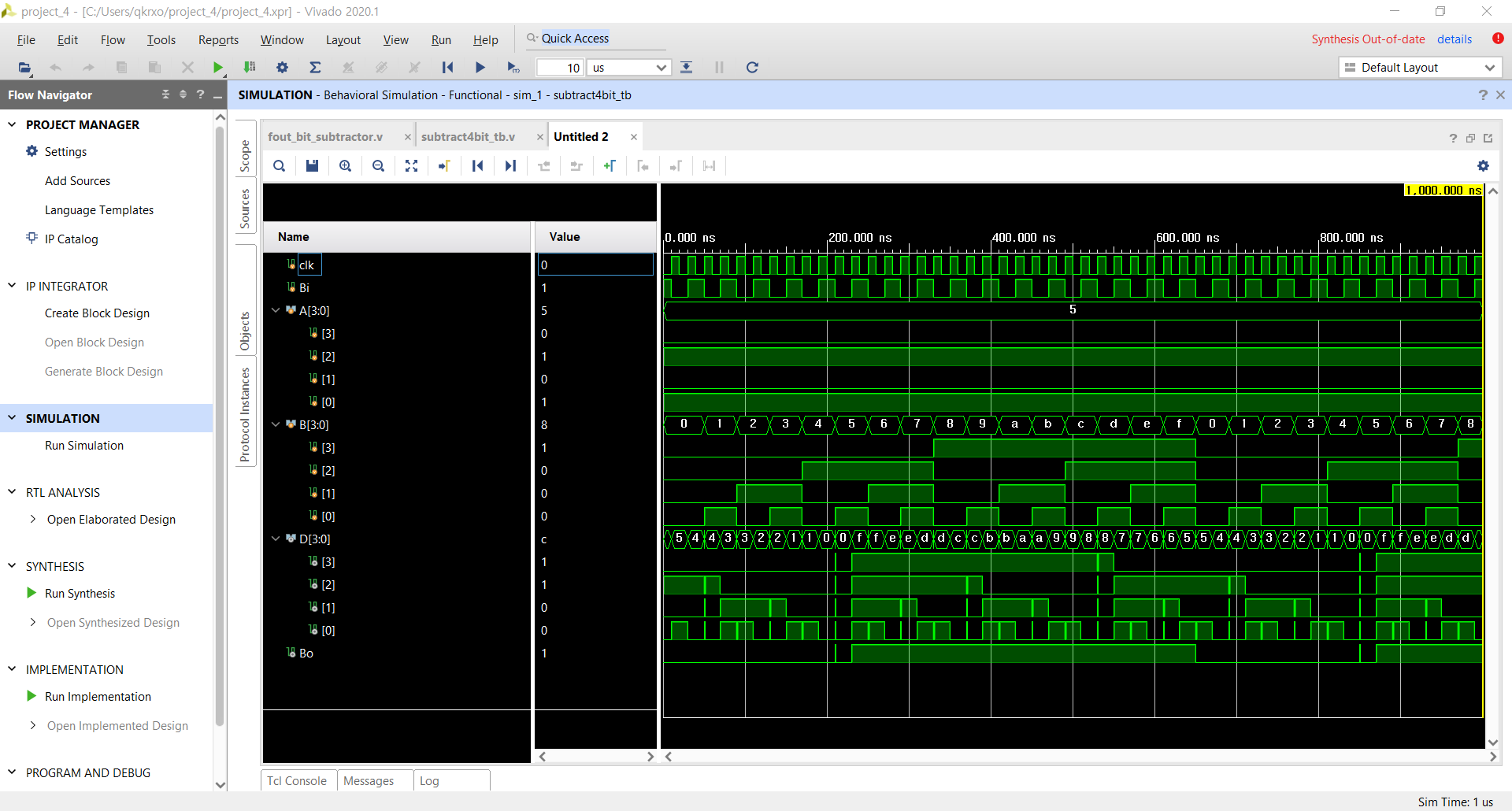
|  |
| --- |
| **subtract1bit** |
| `timescale 1ns / 1ps  module subtract1bit(A, B, Bi, D, Bo);  input A, B, Bi;  output D, Bo;  assign D = A^B^Bi;  assign Bo = ((~A)&Bi)|((~A)&B)|(B&Bi);  endmodule |

이를 이용하여 4bit subtractor를 구현하였다. 각 비트마다 sutract1bit를 사용하여 연산을 진행하였다.

|  |
| --- |
| **four\_bit\_subtractor** |
| `timescale 1ns / 1ps  module four\_bit\_subtractor(A, B, Bi, D, Bo);  input [3:0] A, B; input Bi;  output [3:0] D; output Bo;  wire [3:0] A, B, D; wire Bi, Bo;  wire [2:0] R;  subtract1bit sub1(A[0], B[0], Bi, D[0], R[0]);  subtract1bit sub2(A[1], B[1], R[0], D[1], R[1]);  subtract1bit sub3(A[2], B[2], R[1], D[2], R[2]);  subtract1bit sub4(A[3], B[3], R[2], D[3], Bo);  endmodule |
| **subtract4bit\_tb** |
| `timescale 1ns / 1ps  module subtract4bit\_tb();  reg clk, Bi; reg [3:0] A, B;  wire [3:0] S; wire Bo;  four\_bit\_subtractor connect(A, B, Bi, D, Bo);  initial begin  clk=0; A[3]=0; A[2]=1; A[1]=0; A[0]=1;  B[3]=0; B[2]=0; B[1]=0; B[0]=0; Bi=1; end  always clk = #10 ~clk;  always @(posedge clk) begin  Bi <= ~Bi; B[0] <= #39.999 ~B[0];  B[1] <= #79.999 ~B[1]; B[2] <= #159.999 ~B[2];  B[3] <= #319.999 ~B[3]; end  endmodule |

위의 코드는 4비트의 두 수 A(A[0], A[1], A[2], A[3])와 B(B[0], B[1], B[2], B[3])에 대하여 뺄셈연산을 진행하는 역할을 한다. 마지막 비트 이전 모든 비트에서 나오는 빌림수(R[0], R[1], R[2])가 다음 bit연산을 하는 full subtractor의 입력으로 들어간다. 출력으로 각 비트에서 뺄셈 연산을 한 결과인 D(D[0], D[1], D[2], D[3])과 최종 Borrow out인 Bo가 나온다.

다음은 구현한 코드를 이용하여 Simulation을 한 결과이다.



테스트벤치 코드에서 A[3]=0; A[2]=1; A[1]=0; A[0]=1로 초기화하였기 때문에 A는 5를 나타낸다. B값과 Bi(Borrow in)값이 변하면서 출력 D의 결과 또한 변화하는데, 이진수 뺄셈이 제대로 이루어지는 것을 확인할 수 있다.

FPGA동작을 위해 다음과 같은 xdc코드를 추가하였다.

|  |
| --- |
| **four\_bit\_subtractor.xdc** |
| set\_property IOSTANDARD LVCMOS18 [get\_ports {D[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {D[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {D[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {D[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports Bi]  set\_property IOSTANDARD LVCMOS18 [get\_ports Bo]  set\_property PACKAGE\_PIN F15 [get\_ports {D[0]}]  set\_property PACKAGE\_PIN F13 [get\_ports {D[1]}]  set\_property PACKAGE\_PIN F14 [get\_ports {D[2]}]  set\_property PACKAGE\_PIN F16 [get\_ports {D[3]}]  set\_property PACKAGE\_PIN J4 [get\_ports {A[0]}]  set\_property PACKAGE\_PIN L3 [get\_ports {A[1]}]  set\_property PACKAGE\_PIN K3 [get\_ports {A[2]}]  set\_property PACKAGE\_PIN M2 [get\_ports {A[3]}]  set\_property PACKAGE\_PIN K6 [get\_ports {B[0]}]  set\_property PACKAGE\_PIN J6 [get\_ports {B[1]}]  set\_property PACKAGE\_PIN L5 [get\_ports {B[2]}]  set\_property PACKAGE\_PIN L4 [get\_ports {B[3]}]  set\_property PACKAGE\_PIN Y16 [get\_ports Bi]  set\_property PACKAGE\_PIN E17 [get\_ports Bo] |

FPGA에서 각 스위치와 LED가 나타내는 것을 다음과 같다 하자.

SW1=A[0], SW2=A[1], SW3=A[2], SW4=A[3]

SW5=B[0], SW6=B[1], SW7=B[2], SW8=B[3], SW9=Bi

LD1=D[0], LD2=D[1], LD3=D[2], LD4=D[3], LD5=Bo

이때 예를 들어, 1101(2)-0011(2)-0=1010(2)와 같은 연산을 진행한다면

SW1=ON, SW2=OFF, SW3=ON, SW4=ON, SW5=ON, SW6=ON, SW7=OFF, SW8=OFF, SW9=OFF

와 같이 스위치를 조작을 한다. 결과로는

LD1=OFF, LD2=ON, LD3=OFF, LD4=ON, LD5=OFF, LD(6~16)=OFF

을 얻을 수 있을 것이다.

**3. BCD Adder의 결과 및 Simulation과정에 대해서 설명하시오.**



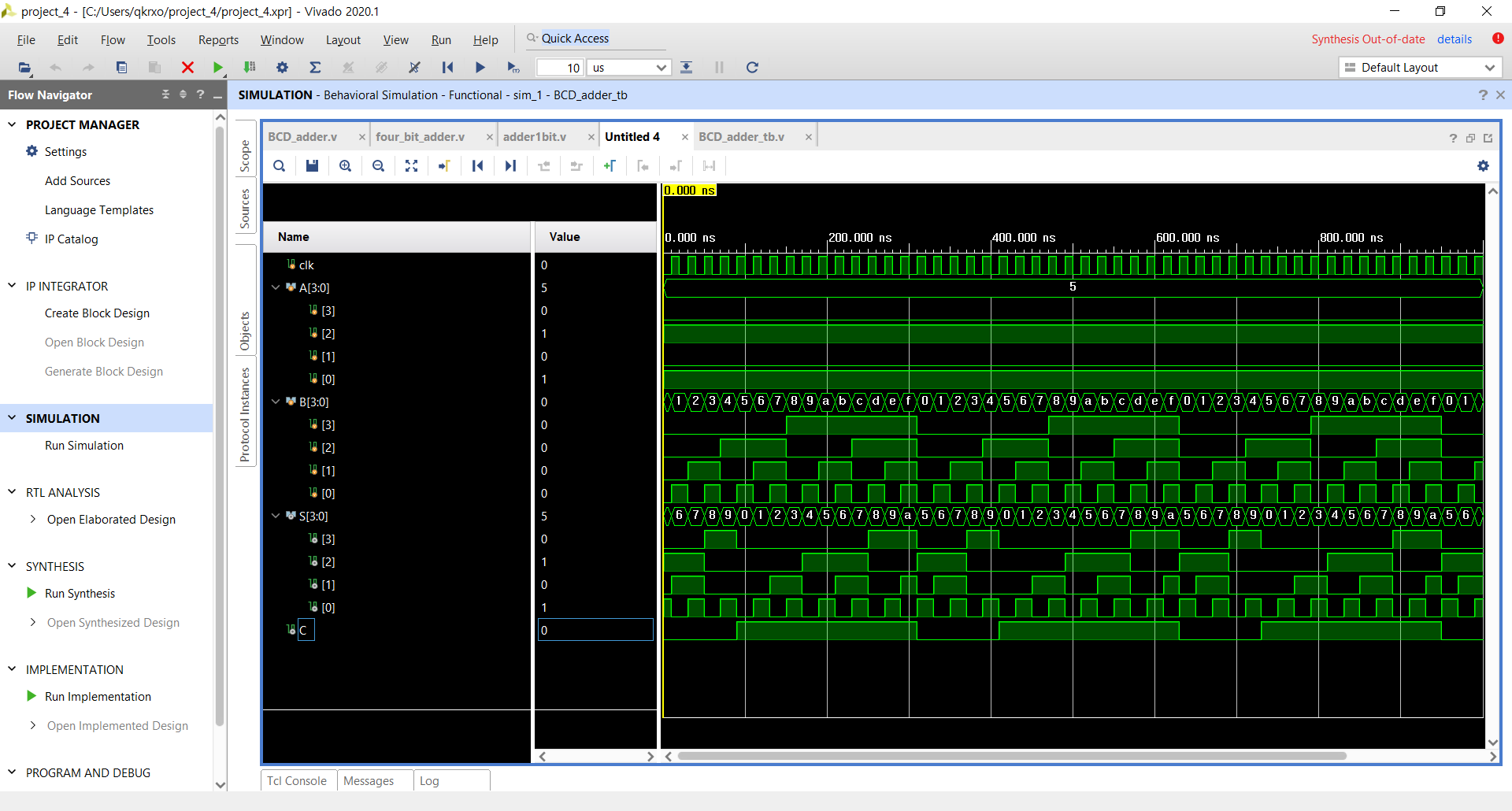
BCD Adder는 위의 그림과 같이 앞에서 구현한 4-bit Binary Adder를 활용하여 구현한다. 처음에 입력 A(A[0], A[1], A[2], A[3])와 B(B[0], B[1], B[2], B[3])을 대상으로 4-bit 덧셈 연산을 진행한다. 이 연산을 통해 생긴 Carry out과 (S[3] \* S[2]), (S[3] \* S[1]) 총 3개를 묶어 OR게이트의 결과를 받는데, 이는 C(Output Carry)이며 BCD에서 10의 자릿수를 나타낸다. 나머지 출력은 또 다시 4-bit Binary Adder를 사용하여 계산하는데, 입력으로 (0, C, C, 0)와 S(S[0], S[1], S[2], S[3])을 받는다. 따라서 최종 출력은 (C // S[3], S[2], S[1], S[0])과 같은 형태가 될 것이다.

BCD Adder는 다음과 같이 구현하였다.

|  |
| --- |
| **BCD\_Adder** |
| `timescale 1ns / 1ps  module BCD\_adder(A,B,S,C);  input [3:0] A, B;  output [3:0] S; output C;  wire [3:0] B2, A2; wire Cout;  four\_bit\_adder add1(A, B, 0, B2, Cout);  assign C = B2[3]&B2[1] | B2[3]&B2[2] | Cout;  assign A2 = 4'b0110 \* C;  four\_bit\_adder add2(A2, B2, 0, S);  endmodule |
| **BCD\_adder\_tb** |
| `timescale 1ns / 1ps  module BCD\_adder\_tb();  reg clk; reg [3:0] A, B;  wire [3:0] S; wire C;  BCD\_adder connect(A, B, S, C);  initial begin  clk=0; A=4'd5;  B=4'd0;  end;  always clk = #10 ~clk;  always @(posedge clk) begin  B <= B+4'd1;  end  endmodule |

4비트의 두 수 A(A[0], A[1], A[2], A[3])와 B(B[0], B[1], B[2], B[3])를 BCD코드로 보고 BCD 덧셈 연산을 진행한다. 앞에서 구현한 4bit adder를 이용하여 구현하였는데, 회로도와 마찬가지로 4bit adder를 두 번 사용하기 위해 두 번 호출한 것을 확인할 수 있으며 이 때 처음 4-Bit Adder에서 나오는 출력은 B2로 표현하여 다음 4-Bit Adder로 넘겨주었다. 또한 assign C = B2[3]&B2[1] | B2[3]&B2[2] | Cout; 와 같이 처음 4-Bit Adder에서 나온 Carry Out(Cout)와 B2를 이용하여 최종 출력 중 하나이자 다음 4-Bit Adder의 입력에 영향을 주는 Output Carry(C)를 계산하였다. 최종적으로 각 비트에서 BCD 덧셈 연산을 한 결과인 S(S[0], S[1], S[2], S[3])와 두 개의 4비트 수를 연산하여 10~18인 수가 나올 때 1을 나타내는 C를 출력으로 내보낸다.

다음은 구현한 코드를 이용하여 Simulation을 한 결과이다.



테스트벤치 코드에서 A=4’d5로 초기화하였기 때문에 A는 5를 나타낸다. B값이 변하면서 출력 S의 결과 또한 변화하는데, BCD 덧셈이 제대로 이루어지는 것을 확인할 수 있다.

FPGA동작을 위해 다음과 같은 xdc코드를 추가하였다.

|  |
| --- |
| **four\_bit\_subtractor.xdc** |
| set\_property IOSTANDARD LVCMOS18 [get\_ports {S[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {S[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {S[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {S[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {A[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[0]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[1]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[2]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports {B[3]}]  set\_property IOSTANDARD LVCMOS18 [get\_ports C]  set\_property PACKAGE\_PIN F15 [get\_ports {S[0]}]  set\_property PACKAGE\_PIN F13 [get\_ports {S[1]}]  set\_property PACKAGE\_PIN F14 [get\_ports {S[2]}]  set\_property PACKAGE\_PIN F16 [get\_ports {S[3]}]  set\_property PACKAGE\_PIN J4 [get\_ports {A[0]}]  set\_property PACKAGE\_PIN L3 [get\_ports {A[1]}]  set\_property PACKAGE\_PIN K3 [get\_ports {A[2]}]  set\_property PACKAGE\_PIN M2 [get\_ports {A[3]}]  set\_property PACKAGE\_PIN K6 [get\_ports {B[0]}]  set\_property PACKAGE\_PIN J6 [get\_ports {B[1]}]  set\_property PACKAGE\_PIN L5 [get\_ports {B[2]}]  set\_property PACKAGE\_PIN L4 [get\_ports {B[3]}]  set\_property PACKAGE\_PIN E17 [get\_ports C] |

FPGA에서 각 스위치와 LED가 나타내는 것을 다음과 같다 하자.

SW1=A[0], SW2=A[1], SW3=A[2], SW4=A[3]

SW5=B[0], SW6=B[1], SW7=B[2], SW8=B[3]

LD1=S[0], LD2=S[1], LD3=S[2], LD4=S[3], LD5=C

이때 예를 들어, 1001+0011=1 0010와 같은 연산을 진행한다면

SW1=ON, SW2=OFF, SW3=OFF, SW4=ON, SW5=OFF, SW6=ON, SW7=OFF, SW8=OFF

와 같이 스위치를 조작을 한다. 결과로는

LD1=OFF, LD2=ON, LD3=OFF, LD4=OFF, LD5=ON, LD(6~16)=OFF

을 얻을 수 있을 것이다.

**4. 결과 검토 및 논의사항.**

4 bit Binary Parallel Adder / Subtractor, BCD Adder를 살펴보았으며 Verilog로 구현하여 Simulation을하였고 FPGA를 통해 동작을 살펴보았다. 이번 주차 실습을 통해 FPGA동작법을 다시 한번 제대로 확인하였다. 또한 4bit Adder(Subtractor)에서는 구현된 1bit Adder(Subtractor)가, BCD Adder에서는 구현된 4bit Adder가 마치 design source안에서 함수처럼 동작할 수 있다는 것 또한 알게 되었다.

**5. 추가 이론 조사 및 작성**

집적회로란 Intrgration Circuit의 약자로 IC라고도 부르며 반도체 물질로 이루어진 칩에서 전자회로들의 집합이라고 정의할 수 있다. 컴퓨터, 핸드폰 등 전자기기의 핵심부품인데, 칩에 들어가는 트랜지스터의 개수에 따라 SSI, MSI, LSI, VLSI로 구분된다.

SSI는 Small-Scale Integeration의 약자로 소규모 집적회로라고 한다. 한 칩에 수십개의 트랜지스터가 집적된 회로를 의미한다. MSI는 Medium-Scale Integration의 약자로 중규모 집적회로라고 한다. 한 칩에 수백개의 트랜지스터가 집적되어 있으며 병렬 가산회로 등에 사용된다. LSI는 Large-Scale Integration의 약자로, 대규모 집적회로라고 하며 한 집에 수 만개의 레지스터가 집적되어 있다. CPU등에 사용이 된다. 마지막으로 VLSI는 Very-Large-Scale-Integration의 약자로, 초고밀도 집적회로라고 하며 현재 수십만개부터 수십억개 까지의 트랜지스터가 집적된 회로를 의미한다.