SoC설계

Lab#10

Shifter

컴퓨터공학과

201402439

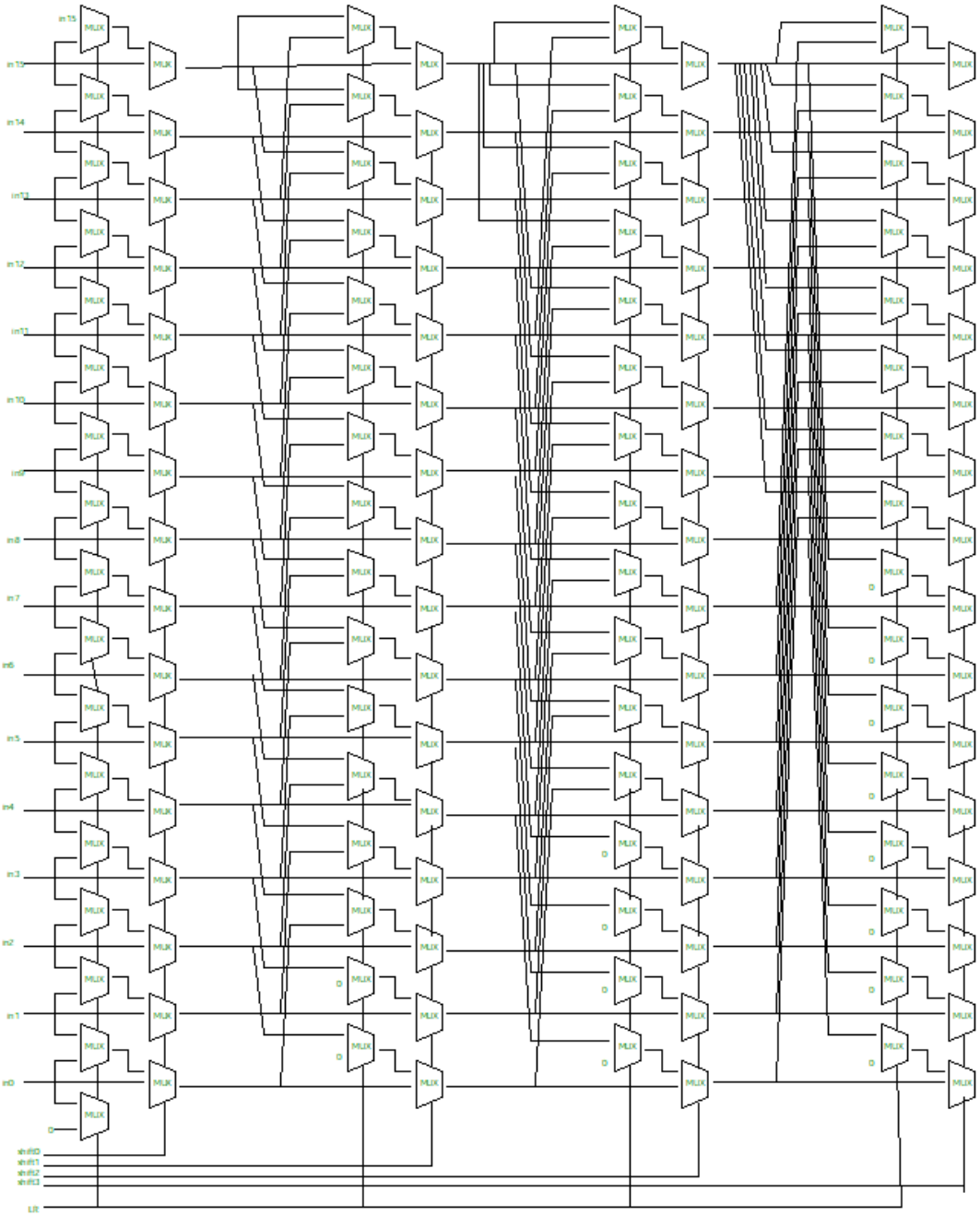
천원준

1. Purpose of the lab

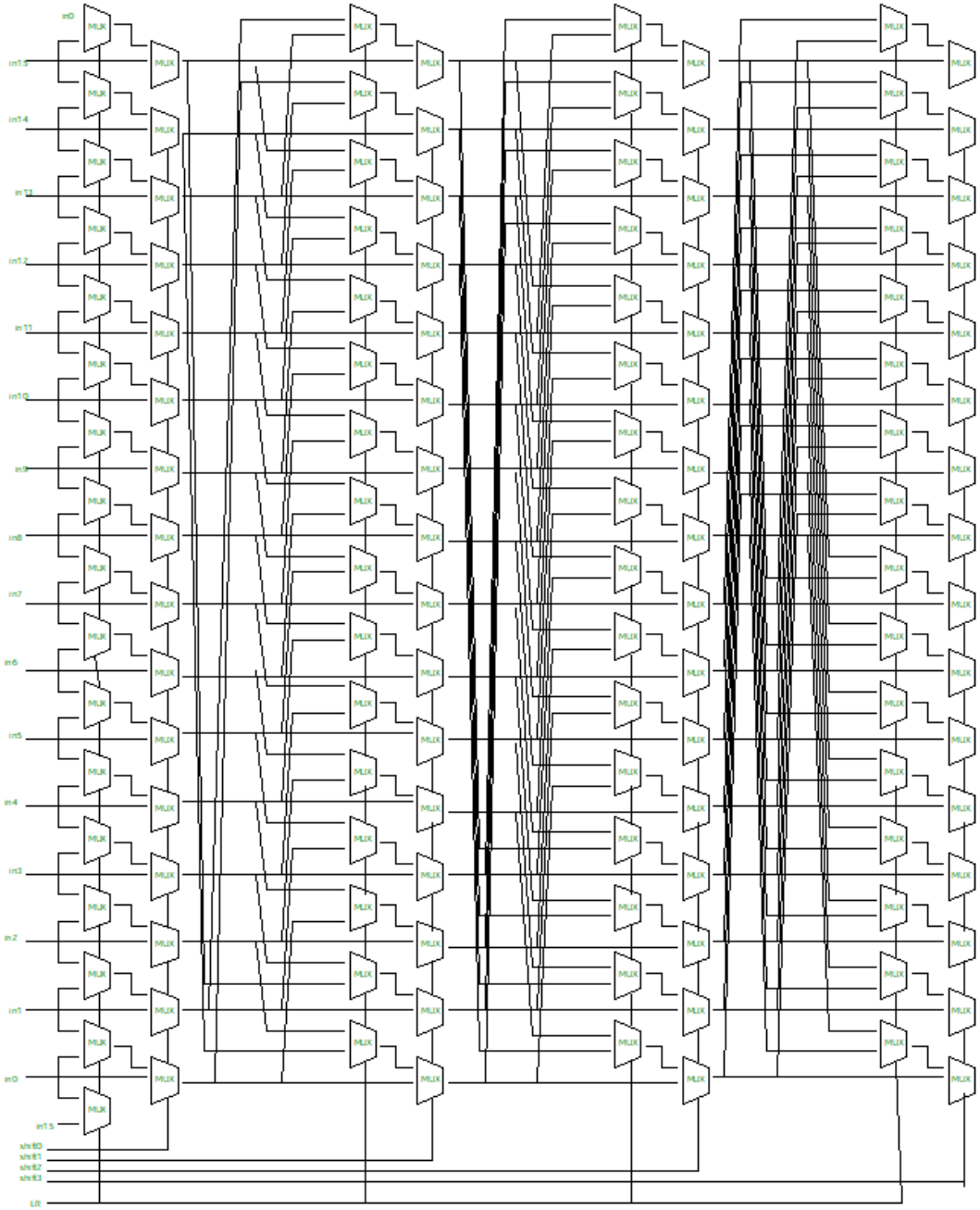
이번 실습의 목표는 16bit 산술 shift 연산이 가능한 Shifter와, 논리 shift 연산이 가능한 Rotator를 구현하는 것입니다.

2. Design Procedure

- Shifter Block Diagram



- Rotator Block Diagram



3. Simulation

- Shifter

```
module Shifter(shift, lr, in, out);
//lr? 1 == left, 0 == right

   input [3:0] shift;
   input lr;
   input [15:0] in;

   output [15:0] out;

   wire [15:0] st1, st2, st3;

   assign st1 = shift[0]? (lr? {in[14:0], 1'b0}:{in[15], in[15:1]}) : in[15:0];
   assign st2 = shift[1]? (lr? {st1[13:0], 2'b00}:{st1[15], st1[15], st1[15:2]}) : st1[15:0];
   assign st3 = shift[2]? (lr? {st2[11:0], 4'b0000}:{st2[15], st2[15], st2[15], st2[15], st2[15:4]}) : st2[15:0];
   assign out = shift[3]?
   (lr? {st3[7:0], 8'b00000000}:{st3[15], st3[15], st3[15], st3[15], st3[15], st3[15], st3[15], st3[15], st3[15:8]}) : st3[15:0];

endmodule
```

Shift 연산의 방향을 정하는 lr, 얼마나 옮길지 정하는 shift, 16bit 피연산자 in을 입력으로 받고, 16bit out을 출력합니다.

St1 에서는 shift의 첫번째 bit가 1이라면, 1비트만큼 lr 방향에 따라 이동 시켜줍니다. 이동 시, 왼쪽 방향 shift일 경우에는 이동하고 남은 자리를 0값으로, 오른쪽 방향 shift일 경우에는 남은 자리를 부호 비트로 채웁니다.

St2, st3, out의 경우 st1과 마찬가지 방식으로 각각 2, 4, 8비트씩 이동시켜줍니다.

- Rotator

```
module Rotator(shift, lr, in, out);
//lr? 1 == left, 0 == right

   input [3:0] shift;
   input lr;
   input [15:0] in;

   output [15:0] out;

   wire [15:0] st1, st2, st3;

   assign st1 = shift[0]? (lr? {in[14:0], in[15]}:{in[0], in[15:1]}) : in[15:0];
   assign st2 = shift[1]? (lr? {st1[13:0], st1[15:14]}:{st1[1:0], st1[15:2]}) : st1[15:0];
   assign st3 = shift[2]? (lr? {st2[11:0], st2[15:12]}:{st2[3:0], st2[15:4]}) : st2[15:0];
   assign out = shift[3]? (lr? {st3[7:0], st3[15:8]}:{st3[7:0], st3[15:8]}) : st3[15:0];

endmodule
```

논리 shift 연산을 수행하는 Rotator 모듈입니다.

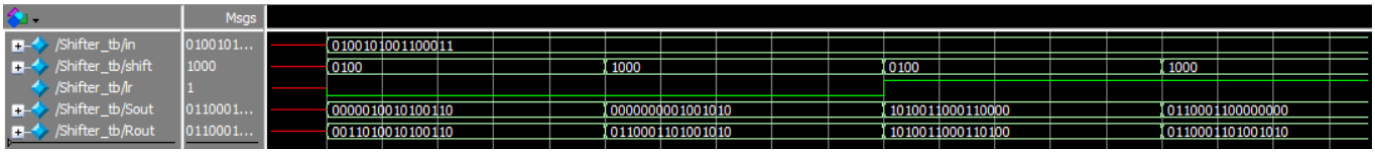
전체적인 동작은 shifter와 유사하지만, shift 연산 후 빈 자리는 shift 연산에 의해 밀려난 값들로 채운다는 점이 다릅니다.

- Shifter testbench

```
`timescale 1ns/100ps

module Shifter_tb;
//Shifter, Rotator testbench

   wire [15:0] Sout, Rout;
   reg [15:0] in;
   reg [3:0] shift;
   reg lr;

   Shifter sh(.shift(shift), .lr(lr), .in(in), .out(Sout));
   Rotator ro(.shift(shift), .lr(lr), .in(in), .out(Rout));

   initial begin
      #10;
      in = 16'b0100101001100011;
      lr = 1'b0;
      shift = 4'b0100;
      #10;
      shift = 4'b1000;
      #10
      lr = 1'b1;
      shift = 4'b0100;
      #10
      shift = 4'b1000;
   end
endmodule
```

앞서 설계한 Shifter와 Rotator를 test하는 testbench 모듈입니다.

오른쪽으로 4비트, 오른쪽으로 8비트, 왼쪽으로 4비트, 왼쪽으로 8비트 순으로 산술 shift연산과 논리 shift 연산을 수행합니다.



입력으로 0100101001100011을 입력하고 산술 shift 연산의 결과인 Sout과, 논리 shift 연산의 결과인 Rout을 확인해보았습니다. 예상한대로 연산이 수행되는 것을 확인할 수 있었습니다.

4. Evaluation

실행한 결과가 의도한 대로 나오는 것을 보아, 올바로 설계했음을 확인할 수 있습니다.

5. Discussion

이번 실습의 key는 shift 연산을 수행하고 난 후의 빈 자리를 산술 연산에서는 0 또는 부호 비트, 논리 연산에서는 shift연산으로 인해 밀려난 값들로 채우는 것입니다.

흥미로웠던 점은 블록 다이어그램이 복잡한 것에 비해, verilog 코드가 매우 단순하단 점이었습니다.

이 design을 향상시킬 방법은 없습니다.