Lab #6 Shifter Design

Class : 00

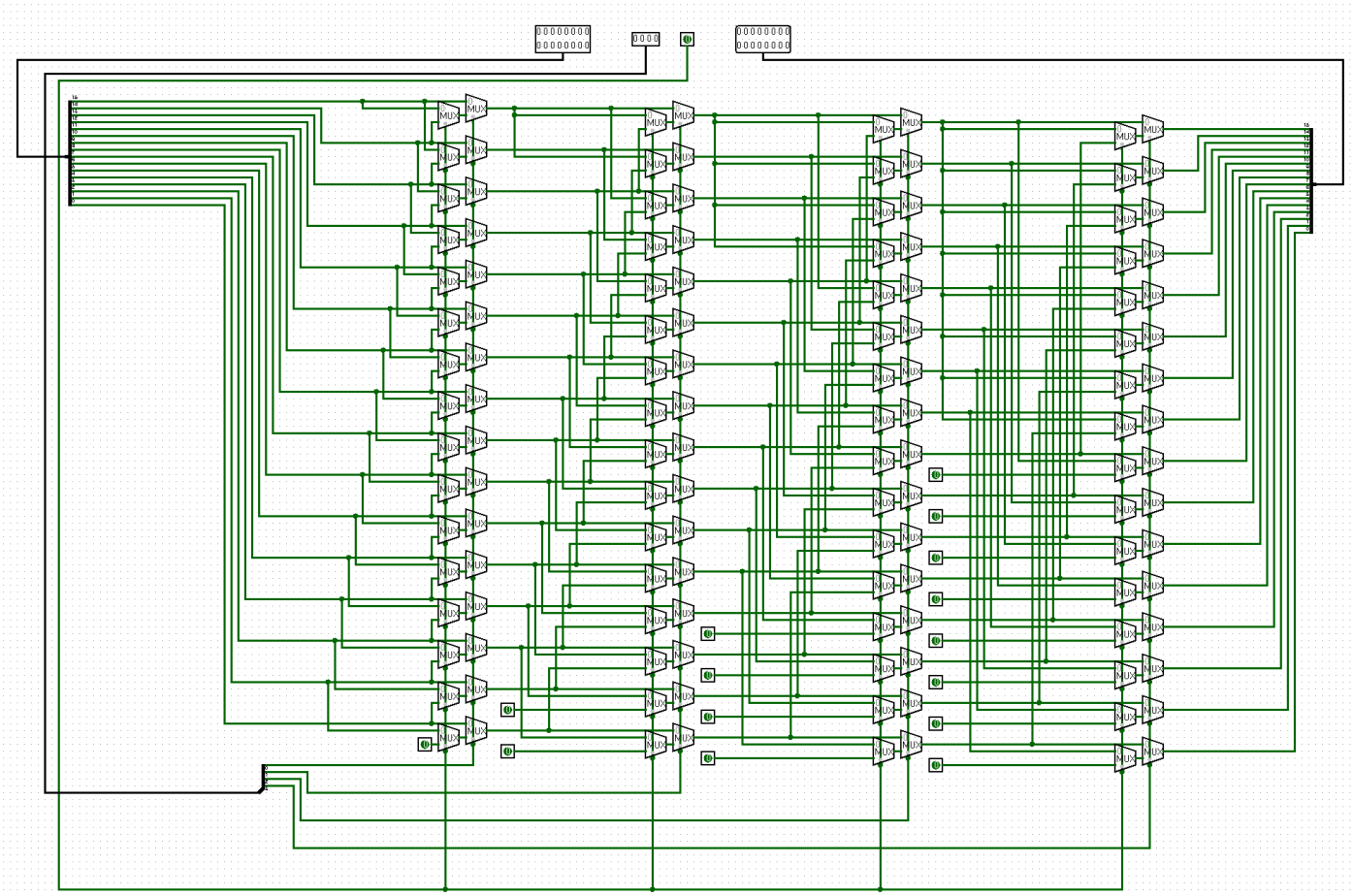
201602004 박태현

I. 실습 목적

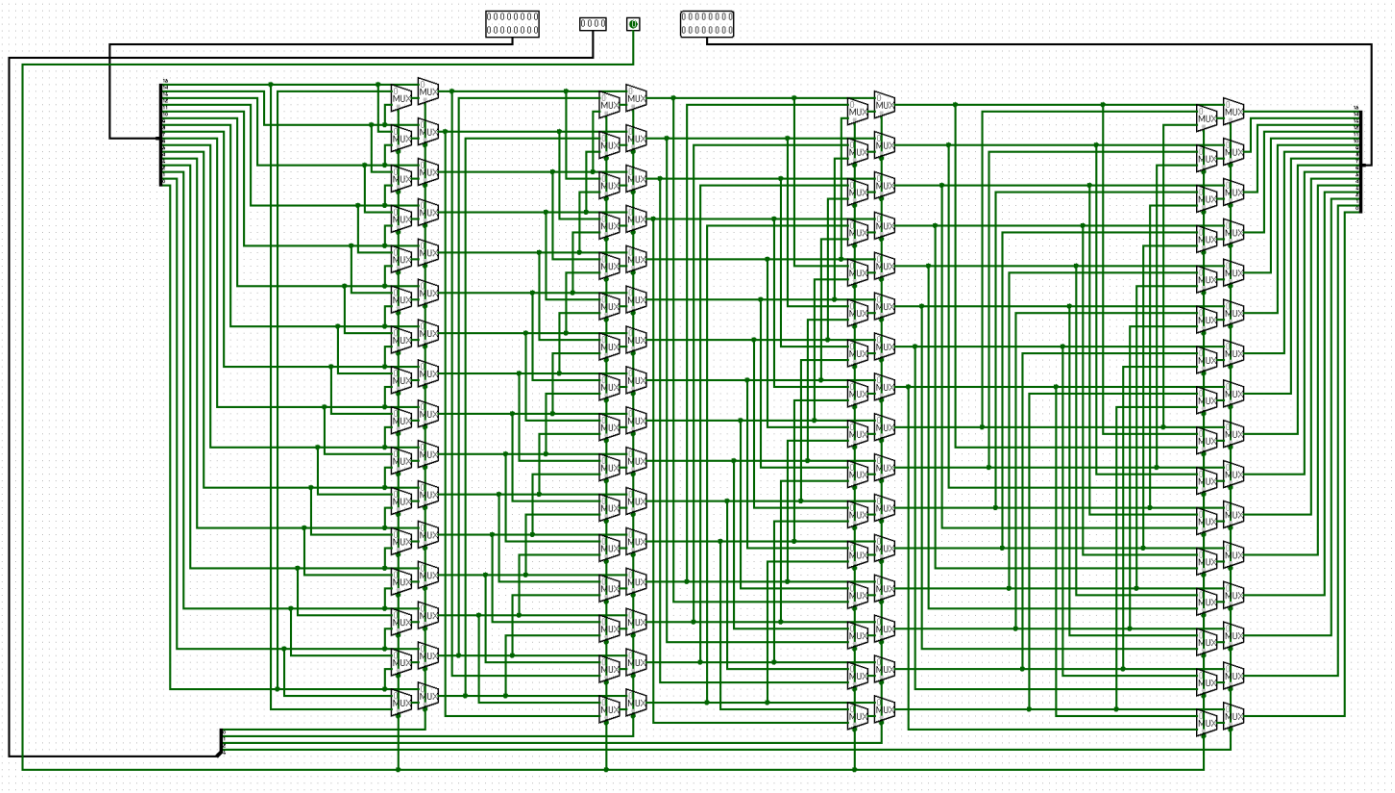
Shifter 와 Rotator 의 구현

II. Design procedure

16bit Shifter



16bit Rotator



III. Simulation

- 16bit shifter

```
module shifter(shift, lr, in, out);
input [3:0] shift;
input lr;
input [15:0] in;
output [15:0] out;

wire [15:0] ta;
wire [15:0] tb;
wire [15:0] tc;
wire [15:0] td;

assign ta = shift[0] ? ( lr ? {in[15],in[15:1]} : {in[14:0], 1'b0} ) : in;
assign tb = shift[1] ? ( lr ? {{2{ta[15]}}, ta[15:2]} : {ta[13:0],2'b0}) : ta;
assign tc = shift[2] ? ( lr ? {{4{tb[15]}}, tb[15:4]} : {tb[11:0],4'b0}) : tb;
assign td = shift[3] ? ( lr ? {{8{tc[15]}}, tc[15:8]} : {tc[7:0], 8'b0}) : tc;

assign out = td;

endmodule
```

시프트할 값 16비트와 시프트 크기를 결정할 4비트, 방향을 결정할 1비트를 입력으로 받습니다

우선 산술 시프트이므로, 오른쪽으로 시프트할 시 오른쪽 빈 공간을 기존의 MSB로 채웁니다

왼쪽으로 시프트할 시는 0으로 채웁니다

레벨A에서는 시프트의 0번째가 1인 경우 좌측 또는 우측으로 1비트씩 시프트합니다.

레벨B에서는 시프트의 1번째가 1인 경우 좌측 또는 우측으로 2비트씩 시프트합니다.

레벨C에서는 시프트의 2번째가 1인 경우 좌측 또는 우측으로 4비트씩 시프트합니다.

레벨D에서는 시프트의 3번째가 1인 경우 좌측 또는 우측으로 8비트씩 시프트합니다.

이렇게 시프트된 결과를 out에 연결하여 결과를 출력합니다

- Rotator

```
module rotator(shift, lr, in, out);
input [3:0] shift;
input lr;
input [15:0] in;
output [15:0] out;

wire [15:0] ta;
wire [15:0] tb;
wire [15:0] tc;
wire [15:0] td;

assign ta = shift[0] ? ( lr ? {in[0],in[15:1]} : {in[14:0], in[15]} ) : in;
assign tb = shift[1] ? ( lr ? {ta[1:0], ta[15:2]} : {ta[13:0],ta[15:14]}) : ta;
assign tc = shift[2] ? ( lr ? {tb[3:0], tb[15:4]} : {tb[11:0],tb[15:12]}) : tb;
assign td = shift[3] ? ( lr ? {tc[7:0], tc[15:8]} : {tc[7:0], tc[15:8]})  : tc;

assign out = td;

endmodule
```

시프트할 값 16비트와 시프트 크기를 결정할 4비트, 방향을 결정할 1비트를 입력으로 받습니다

Rotator이므로, 시프트한 경우, 시프트해서 범위 밖으로 나간 비트를 반대편으로 밀어넣습니다

레벨A에서는 시프트의 0번째가 1인 경우 좌측 또는 우측으로 1비트씩 시프트합니다.

이때 밀려난 1비트는 반대쪽에 저장합니다.

레벨B에서는 시프트의 1번째가 1인 경우 좌측 또는 우측으로 2비트씩 시프트합니다.

이때 밀려난 2비트는 반대쪽에 저장합니다

레벨C에서는 시프트의 2번째가 1인 경우 좌측 또는 우측으로 4비트씩 시프트합니다.

이때 밀려난 4비트는 반대쪽에 저장합니다.

레벨D에서는 시프트의 3번째가 1인 경우 좌측 또는 우측으로 8비트씩 시프트합니다.

이때 밀려난 8비트는 반대쪽에 저장합니다.

이렇게 시프트된 결과를 out에 연결하여 결과를 출력합니다

- Testbench

```
module shifter_tb;

reg [3:0] shift;
reg lr;
reg signed [15:0] in;
wire signed [15:0] out;
reg signed [15:0] ans;

integer i, count;
parameter iter = 1000;


shifter shifter0(shift, lr, in, out);

initial begin
count = 0;

for(i = 0;i < iter; i = i + 1) begin

   in = $random;
   lr = $random;
   shift = $random;

   if(lr == 1) ans = in >>> shift;
   else ans = in <<< shift;
   #10; // wait for calculate
   if({out} == {ans}) count = count + 1;
   else begin
      if(lr == 1) $display("%d >> %d = %d, but %d\n", in, shift, ans, out);
      else $display("%d << %d = %d, but %d\n", in, shift, ans, out);
   end

end
$display("accept count : %d\n",count);

end

endmodule
```

시프터 테스트. 입력값에 대해 부호가 있는 입력이라는 조건을 달아서 산술 연산 시프트를 테스트에서 가능하도록 만들었습니다.

Lr == 1인 경우 오른쪽으로 시프트합니다. 0인 경우 왼쪽으로 시프트합니다.

시프트 연산 결과가 나오기를 잠시 기다리고, out과 ans를 비교하여 일치하는지 확인합니다.

맞는 경우 카운트를 증가시킵니다.

```
module rotator_tb;

reg [3:0] shift;
reg lr;
reg [15:0] in;
wire [15:0] out;
reg [15:0] ans;

integer i, count;
parameter iter = 1000;

rotator rotator0(shift, lr, in, out);

initial begin
count = 0;

for(i = 0;i < iter; i = i + 1) begin
   lr = $random;
   shift = $random;
   in = $random;

   if(shift == 0) ans = in;
   else if(lr == 1) ans = (in >> shift) | (in << (16 - shift));
   else ans = (in << shift) | (in >> (16 - shift));
   #10; // wait for calculate
   if({out} == {ans}) count = count + 1;
   else begin
      if(lr == 1) $display("%d >> %d = %d, but %d\n", in, shift, ans, out);
      else $display("%d << %d = %d, but %d\n", in, shift, ans, out);
   end

end
$display("accept count : %d\n",count);

end

endmodule
```

Rotator 테스트.

Lr == 1인 경우 오른쪽으로 시프트합니다. 0인 경우 왼쪽으로 시프트합니다.

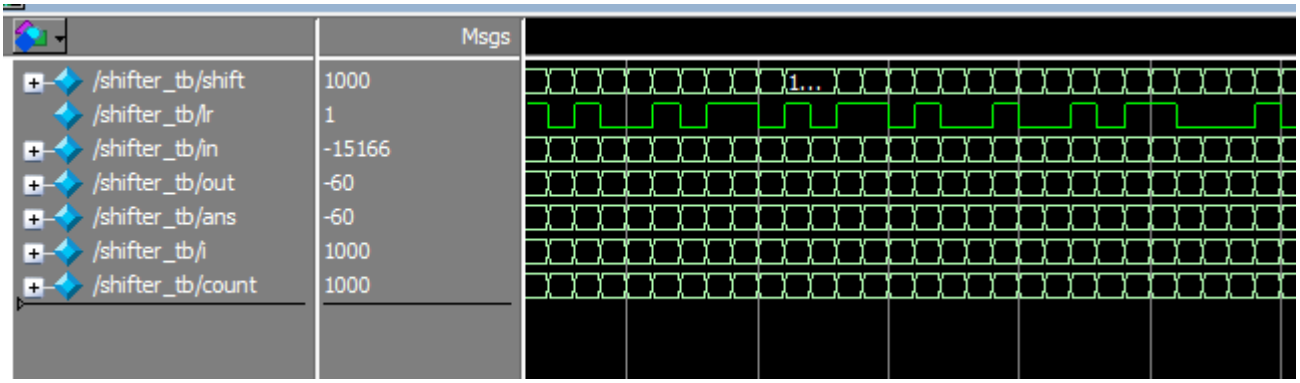
밀려난 비트에 대해서는 반대쪽으로 옮겨줍니다

시프트 연산 결과가 나오기를 잠시 기다리고, out과 ans를 비교하여 일치하는지 확인합니다.

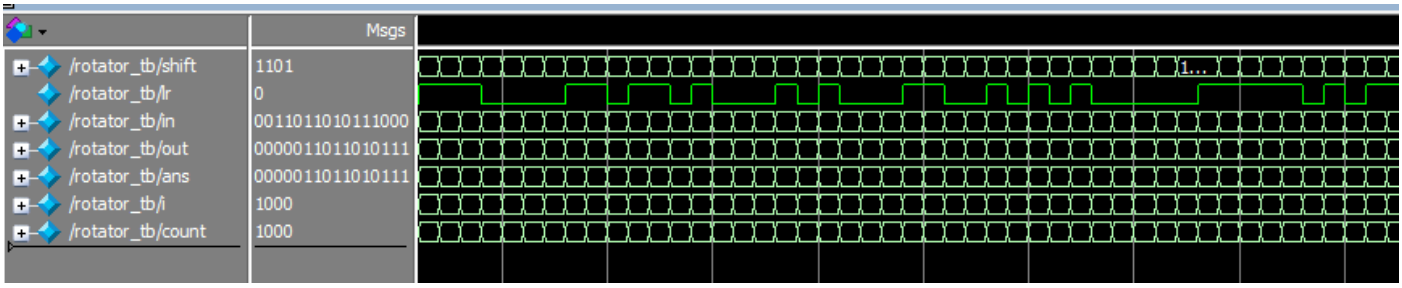
맞는 경우 카운트를 증가시킵니다.

- Waveform

Shifter



Rotator



IV. Evaluation

입력에 대한 결과가 일치하는지를 카운팅하여 확인하였습니다

Shifter

```
# run -all
# accept count :        1000
#
```

Rotator

```
# .main_pane.objects.interior.cs
# run -all
# accept count :        1000
#
```

V. Discussions

코드를 통해 작성하는 것이 직접 그려서 작성하는 것보다 훨씬 확실하고 깔끔하게 작성할 수 있다는 것을 느꼈습니다.

Rotator의 경우 마지막 shifting에서 절반만큼을 좌,우로 이동합니다. 여기서 lr 플래그가 필요없어 이에 대한 MUX를 제거한다면 더 효율적인 rotator를 만들 수 있을 것 같습니다.