REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

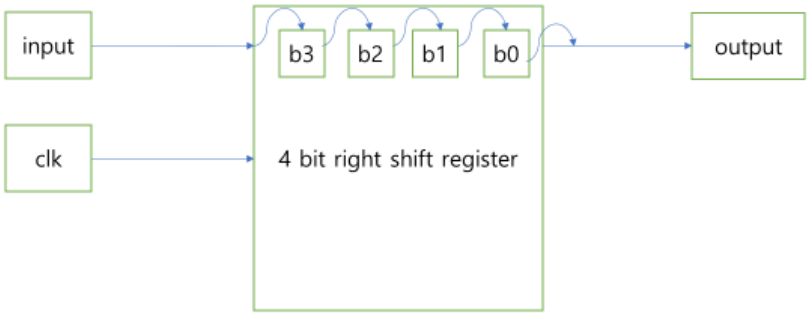
과　　목 : 디지털 시스템
과 제 명 : HW1
담당교수 : 민 형 복
학　　과 : 소프트웨어학과
학　　년 : 3
학　　번 : 2016312761
이　　름 : 여혁수
제 출 일 : 2020/10/12

1. Purpose

이번 실습에서 설계한 것은 synthesize 가능한 4 bit right shift register[1]와 이것이 잘 되는지 테스트하는 프로그램이다. 4 bit right shift register란 4비트를 담고 있는데 일정 주기마다 새롭게 들어갈 인풋 비트 값을 주면 4개의 비트 각각 1비트씩 오른쪽으로 위치를 옮기고 가장 왼쪽에 인풋이 들어가고 가장 오른쪽에 위치했던 비트는 아웃풋이 되는 프로그램이다.

2. Problem statement

1) Describe what is the problem.



이번 실습에서 구현할 것은 크게 2 가지다. 4 bit right shift register 와 그리고 이를 테스트하는 testbench 이다. 우선 4 bit right shift register 는 위 diagram 과 같이 clk 가 rising edge 일 때마다 input 비트를 레지스터에 전달하고 레지스터 안에 4 개의 비트는 1 비트씩 오른쪽으로 밀려 가장 오른쪽 비트는 아웃풋으로 빠져나가는 방식의 레지스터이다. 우선 올바른 이 프로그램에 대한 올바른 코드를 만들기 위해 Modelsim[2] 프로그램에서 내가 생각한 4 bit right shift

register 에 대한 logic 을 Verilog 코드화 하고 시뮬레이션했다[3-4]. 시뮬레이션하기 위해 이를 검증해줄 testbench 프로그램도 구현한다. 일단 테스트될 인풋 비트 배열과 그에 대한 올바른 결과를 예상한 비트 배열을 준비한다. Testbench 에서 준비한 인풋 비트 배열을 4 bit shift register 모듈에 입력해주어서 모듈에서 나온 아웃풋과 예상했던 값을 비교한다. 그래서 일치하는지 확인하고 불일치하다면 에러를 출력한다. 이 Design 에서는 아웃풋을 따로 저장하지는 않고 그냥 소멸된다.

2) Describe how do you solve the problem.

문제 해결을 위해 우선 shift register와 testbench의 Verilog 코드를 구현했다. Shift register는 수업을 통해 배운 left shift register를 참고[5-6]하여 구현했고, testbench 역시 강의자료pdf에서[7] 찾은 예시 코드를 참고하여 구현했다. 그 외에도 수업[8]에서 배운 continuous assignment를 shift register의 아웃풋을 할당해주는 데 활용하였고, case equal operator도 수업에서 알게 되어 testbench 에 예상 값과 실제 아웃풋을 비교하는 데에 사용하여 올바른 코드를 구현했다.
또한 Modelsim 프로그램에서 transcript에 커맨드를 입력하여 시뮬레이션 조건을 설정하는 것을 새롭게 배웠고, 그래서 시뮬레이션을 내가 하고 싶은 대로 할 수 있었고, 올바른 코드를 구현했다는 것을 알 수 있었다. 그리고 Verilog 문법의 구사방법을 익히는 계기가 되었는데 문법에 대해서 정리하자면 c언어와 매우 유사하지만 initial, always라는 프로그램 실행 시에 바로 한 번만 수행되는 부분과 프로그램 종료 전까지 계속 반복 실행되는 부분을 따로 정의할 수 있다는 것이 특별한 것 같다. 그리고 non-blocking assignment도 어떤 코드 블록 안에서 scheduling만 하고 같은 블록의 다른 코드들과 동시에 적용이 된다는 것이 특별했다. 마지막으로 synthesize 가능한지 알아보기 위해 Intel Quartus Prime[9] 프로그램에 코드를 넣어서 회로 모양을 확인하였다.

3. Sources & Results

1) Analysis of HDL Source Codes of Design

<rsr.v>

```
assign so = Q[0];
always @ (posedge clk) begin
    Q <= {si, Q[NBITS-1:1]};
end
```

여기가 right shift register에서 shifting과 아웃풋 할당이 이루어지는 부분이다. 처음에는 always block 안에 shifting과 아웃풋 할당을 모두 넣어 주었는데 그렇게 하면 shifting이 적용되기 전에 아웃풋 값을 scheduling을 이미 해놓아서 simulation했을 때 예상 아웃풋 타이밍보다 한 차례씩 밀려서 아웃풋이 들어왔다. 그래서 아웃풋 할당을 별도의 continuous assignment를 통해 해주었다.

2) Analysis of Testbench Source Codes

<rsr_tb.v>

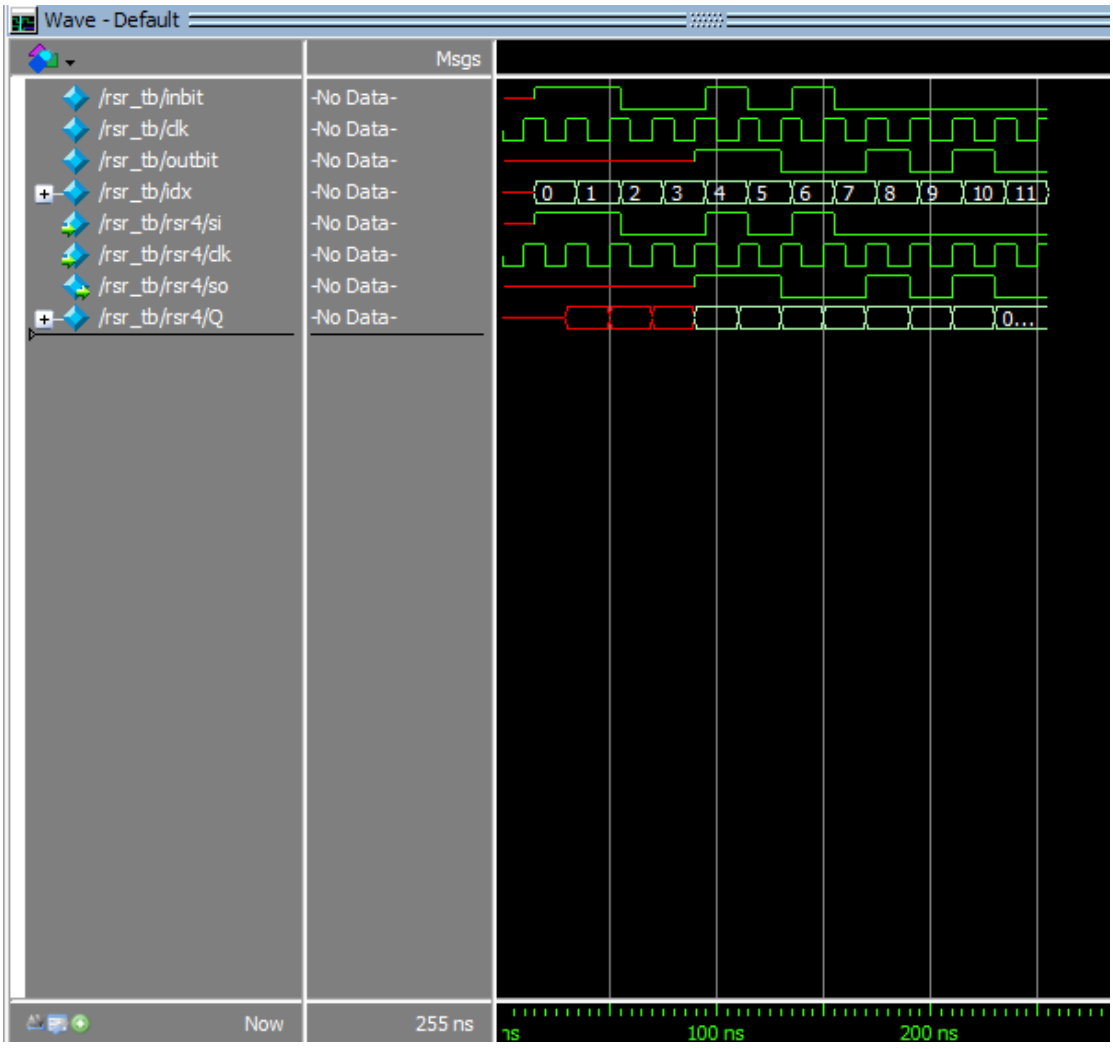
```
parameter [0:NUM_TEST_BITS-1] STIMULUS = 12'b11001010xxxx;
parameter [0:NUM_TEST_BITS-1] RESPONSE = 12'bxxxx11001010;

    for (idx = 0 ; idx < NUM_TEST_BITS ; idx = idx+1) begin
        if (STIMULUS[idx] !== 1'bx) begin
            inbit = STIMULUS[idx];
        end
       if (RESPONSE[idx] !== 1'bx) begin
            if (!(outbit === RESPONSE[idx])) begin
                $error("not matched with predicted result!");
            end else begin
                $display("Correct!");
            end
        end
```

```
        $display("expected bit: ", RESPONSE[idx], " output bit: ", outbit);
        #CLOCK_PERIOD;
    end
```

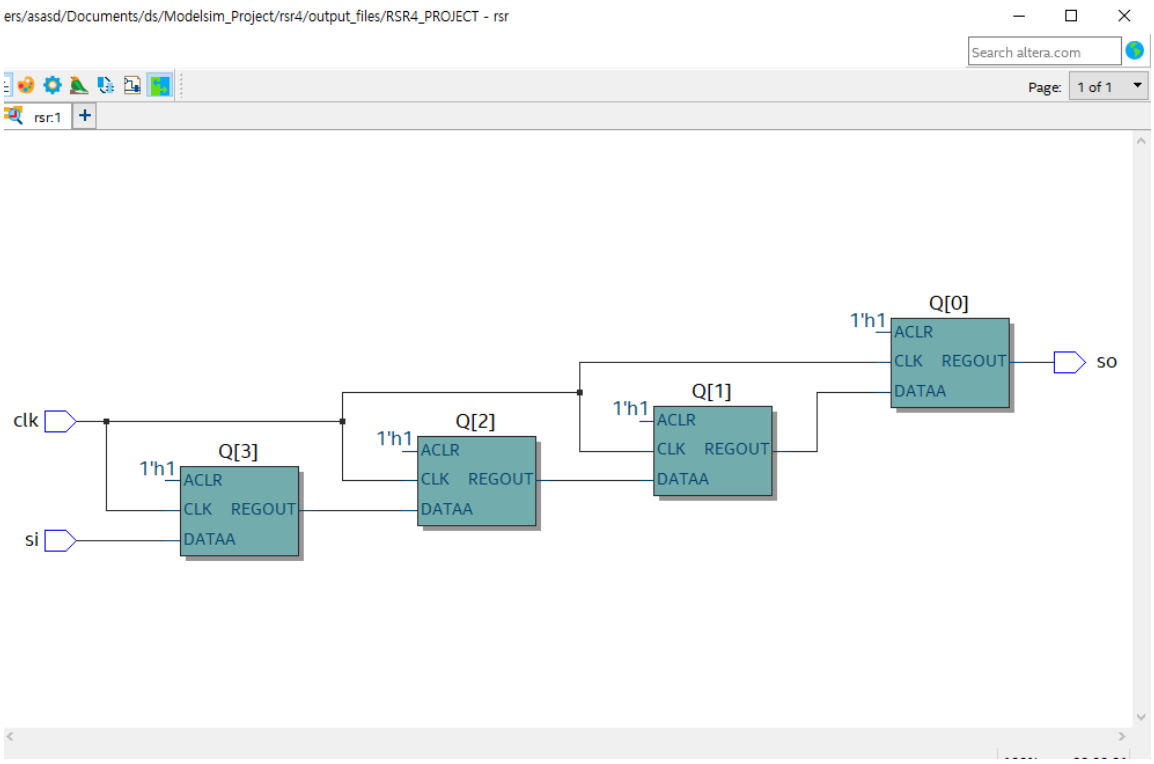
이것이 testbench의 핵심이라고 생각하는 실제 아웃풋과 예상 아웃풋 비교 부분이다. 배열 크기만큼 반복하면서 테스트 인풋 배열 값을 차례로 rsr 모듈의 인풋으로 주는데, 아웃풋은 인풋 4개가 들어온 후부터 생기기 때문에 인풋 배열과 같은 타이밍에 비교하면 안된다. 그래서 같은 반복문에서 아웃풋 비교를 하되 예상 아웃풋 배열에 첫 4비트는 unknown bit를 줌으로서 배열 값이 unknown bit일 경우는 아웃풋 비교를 하지 않도록 했다. 아웃풋 비교는 case equal operator를 사용하여 혹시나 비교하는 값 중에 하나가 unknown bit 이어도 같다고 판단할 수 있는 여지를 없앴다.

3) Simulation Waveforms

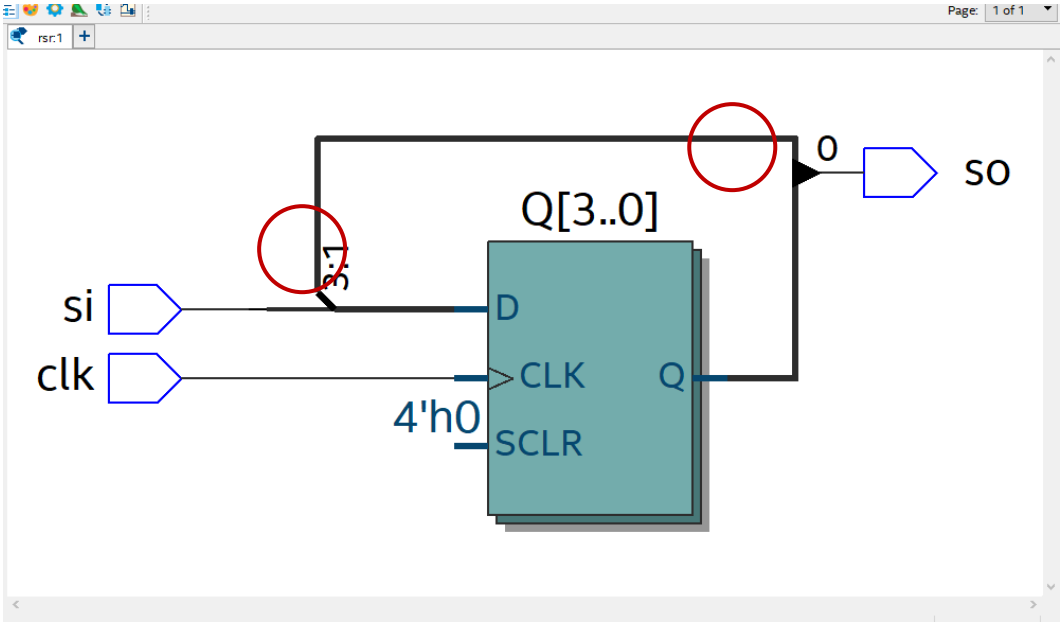


시뮬레이션 했을 때의 waveform이다. /rsr_tb/rsr4/에서 rsr4는 rsr모듈의 이름이다. rsr 모듈의 si라는 인풋이 클락마다 순서대로 1, 1, 0, 0을 받은 뒤에 so라는 아웃풋 비트가 클락의 rising edge에서 1로 반응하는 것을 볼 수 있다. 그리고 이때 testbench의 outbit라는 아웃풋 비트 또한 같이 업데이트 되는 것을 볼 수 있다. 이후로 인풋 비트의 파형을 아웃풋 비트의 파형이 80ns 정도 늦게 같은 파형으로 따라가는 것을 볼 수 있다. Q라는 배열은 인풋 비트가 4번 이상 들어오기 전까지는 다 채워지지 않았기 때문에 빨간색으로 표시되고 그 후로는 초록색인 것을 볼 수 있다.

4) Synthesis Results



Intel Quartus Prime 프로그램에서 MAX II device로 synthesis 했을 때 netlist 결과이다. 4개의 logic elements와 3개의 핀으로 synthesis 되었음을 알 수 있다. 각 logic element에 ACLR이라는 클리어 비트로 추정되는 것이 있는데 이것은 의도한 것이 아니기 때문에 문제라고 봐야할 것 같다.



이것은 RTL design이다. 하나 보이는 레지스터가 4 bit right shift register이다. rsr.v 코드가 이 레지스터에 해당된다. 아웃풋 Q가 나뉘는 것을 빨간 동그라미로 표시했는데, 4칸의 배열 전체가 아웃풋이 되어 0번째 비트는 아웃풋이 되고 나머지는 다시 그 배열의 인풋 데이터 핀과 같이 들어가는 것이 신기했다.

4. Conclusion

솔직히 Verilog 코드를 올바르게 구현하는 것은 어렵지 않았고 빠르게 끝났지만, 이것이 올바른 프로그램이라는 것을 증명하기 위해 여러 simulation 및 synthesis의 상황을 말로 쉽게 풀어내는 것이 어려웠다. 소프트웨어나 프로그램은 구현하는 것으로 끝이 아니라 내 프로그램에 대해 알고자 하는 사람들에게 쉽게 그리고 확실한 자료로 설명하는 능력이 필요하다는 것을 알게 되었다.

5. 참고 문헌

[1] 민형복, "vrsr.pdf," [Online]. Available: http://class.icc.skku.ac.kr/~min/di/. [Accessed Oct. 9, 2020].

[2] Modelsim. (FPGA), Intel. [Accessed Oct. 9, 2020]. Available: https://fpgasoftware.intel.com/20.1/?edition=lite

[3] rsr.do, 민형복. Accessed: Oct 10, 2020 Avaliable: http://class.icc.skku.ac.kr/~min/di/.

[4] modelsim.ini, Mentor Graphics Corporation. Accessed: Oct 10, 2020 Avaliable: http://class.icc.skku.ac.kr/~min/di/practice/verilog/ace/code.php?type=ini&dir=aux d&file=modelsim.ini.quartus.2020.1.ini
[5] 민형복, "DigitalSystem02.pdf," [Online]. Available: http://class.icc.skku.ac.kr/~min/di/. [Accessed Oct. 10, 2020].
[6] 민형복, "SynthesizableVerilog.pdf," [Online]. Available: http://class.icc.skku.ac.kr/~min/di/. [Accessed Oct. 10, 2020].
[7] 민형복, "vhints_pdf," [Online]. Available: http://class.icc.skku.ac.kr/~min/di/. [Accessed Oct. 10, 2020].
[8] 민형복, ds_2_verilog_part3.mp4, Topic: "Chapter 2: Introduction to Verilog (part 3)
[9] Intel Quartus Prime. (20.1), Intel. [Accessed Oct. 9, 2020]. Available: https://fpgasoftware.intel.com/20.1/?edition=lite