14주차 결과보고서

전공: 수학과 학년: 4학년 학번: 20171273 이름: 심현우

1. **Sequence Detector 1101 Moore machine 구현(Verilog source, simulation 결과(testbench 파일 작성시 output 제대로 나오도록 조정), 상태도, 상태표 작성)**

Sequence detector란 input에 대하여 주어진 sequence를 찾는 기계이다. Sequence detector 1101은 1101의 배열을 찾아야 한다. 예시로 보면 쉽게 이해할 수 있다. 만약 매 posedge clk마다 11011011101의 input이 들어온다고 가정하자. 그렇다면 output은 00010010001으로 출력되어야 한다. 이는 1101이 완성되는 부분에서 1의 출력이 나오는 것을 알 수 있다. Mealy와 moore의 차이는 이전 state를 이용하는지에 따라 나뉜다. 지난 실습에서 구현했던 mealy machine의 경우 이전 state를 이용하지 않고 구현한다. State table은 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **이전 상태** | **다음 상태** | | **출력** | |
| **Input 0** | **Input 1** | **Input 0** | **Input 1** |
| **S0(00)** | **S0** | **S1** | **0** | **0** |
| **S1(01)** | **S0** | **S2** | **0** | **0** |
| **S2(10)** | **S3** | **S2** | **0** | **0** |
| **S3(11)** | **S0** | **S1** | **0** | **1** |

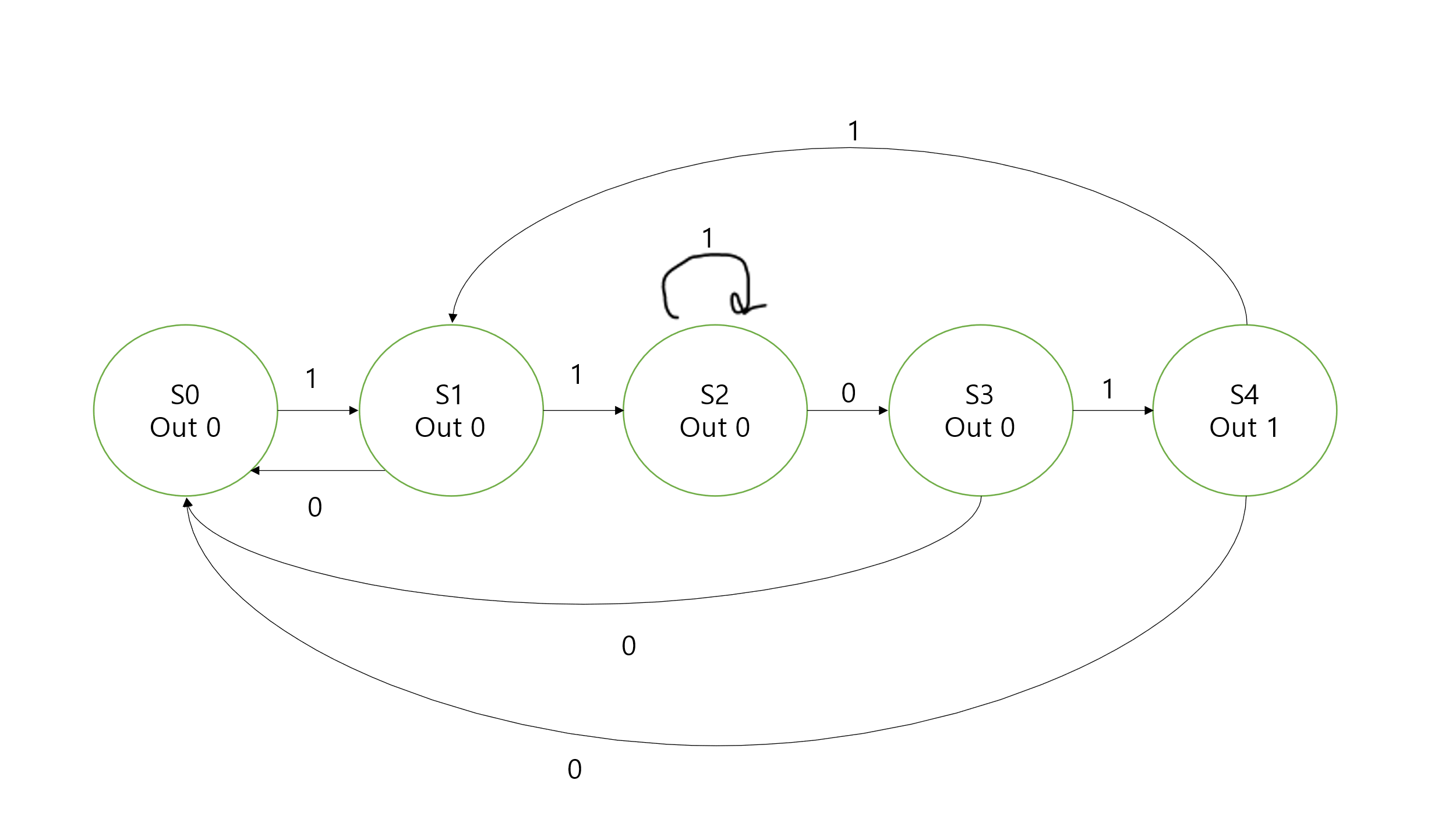
Mealy machine의 경우 입력에 따라 바로 반응하며 적은 state로 구성이 가능하다. 1101의 경우 4번의 input을 통해 detect할 수 있으므로 4개의 state로 구성할 수 있는 것이다.

Moore machine은 이전 상태와 input을 확인하며 출력한다. State table로 이해해보자.

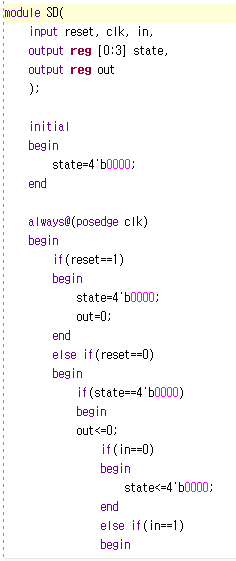
|  |  |  |  |
| --- | --- | --- | --- |
| **이전 상태** | **다음 상태** | | **출력** |
| **Input 0** | **Input 1** |
| **S0(0000)** | **S0** | **S1** | **0** |
| **S1(0001)** | **S0** | **S2** | **0** |
| **S2(0011)** | **S3** | **S2** | **0** |
| **S3(0110)** | **S0** | **S4** | **0** |
| **S4(1101)** | **S0** | **S1** | **1** |

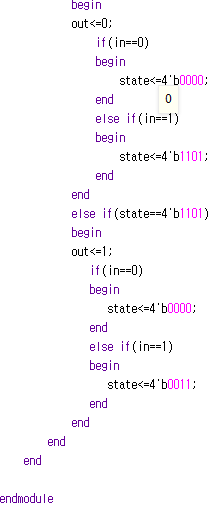
Input과 상태를 가지고 output을 출력한다. 상태에서 1101이 완료되면 input값에 상관없이 1을 출력한다. 그 이후 input에 따라 상태를 변화한다.

Moore 머신의 상태도는 다음과 같다.



이제 Verilog code로 작성하여 확인해보자.

텍스트이(가) 표시된 사진

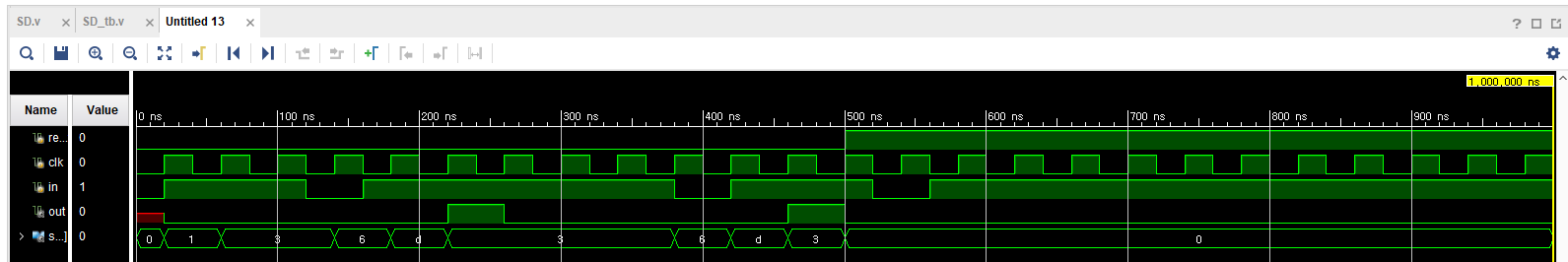
자동 생성된 설명

State table과 diagram에 의거하여 코드를 작성했다. Testbench는 다양한 input을 넣을 수 있도록 다음과 같이 작성했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

코드에 대한 simulation결과는 다음과 같다.



moore머신은 출력이 한칸씩 밀려서 나오므로 200ns에서 1101을 detect한 다음 220ns에서 out이 1로 출력되는 것을 확인할 수 있다. 그 뒤 1101에서도 1을 잘 출력하는 것을 확인할 수 있다.

1. **Sequence Detector 10101을 구현(mealy, moore machine 모두) (Verilog source, simulation 결과(testbench 파일 작성시 output 제대로 나오도록 조정), 상태도, 상태표 작성)**

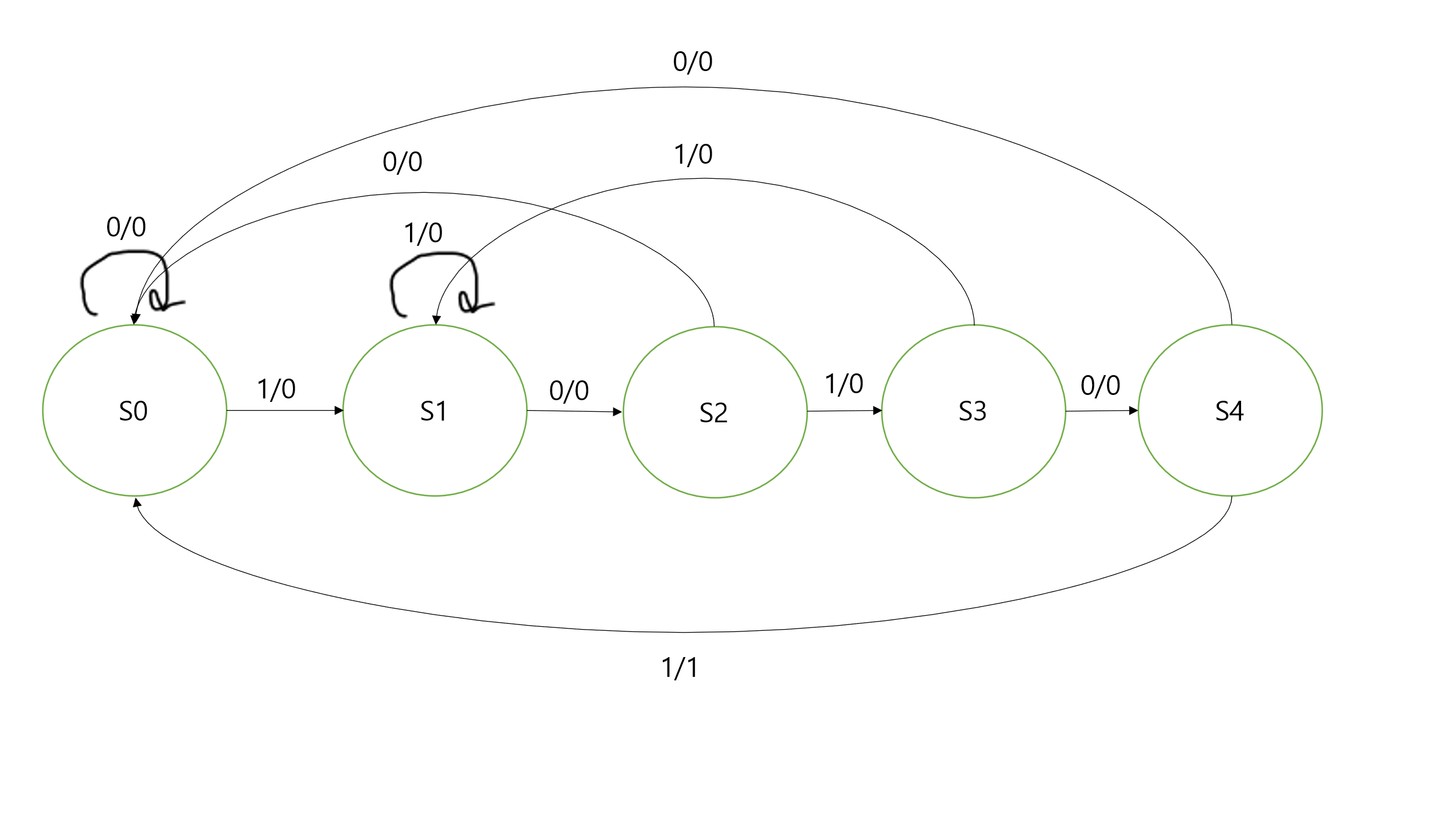
위의 설명처럼 10101도 구현해보자. 우선 적은 상태로 구현할 수 있는 mealy machine부터 보자.

State table은 다음과 같다.

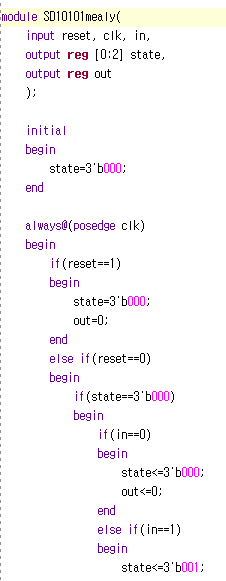
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **이전 상태** | **다음 상태** | | **출력** | |
| **Input 0** | **Input 1** | **Input 0** | **Input 1** |
| **S0(000)** | **S0** | **S1** | **0** | **0** |
| **S1(001)** | **S2** | **S1** | **0** | **0** |
| **S2(010)** | **S0** | **S3** | **0** | **0** |
| **S3(011)** | **S4** | **S1** | **0** | **0** |
| **S4(100)** | **S0** | **S0** | **0** | **1** |

이 역시 10101 총 다섯개의 bit를 찾아야 하므로 5개의 state를 통해 구성할 수 있다.

State diagram은 다음과 같다.



Table과 diagram에 맞춰 Verilog 코드를 작성해보자.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

Testbench도 10101을 포함하도록 작성했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

simulation결과는 다음과 같다.

텍스트, 실내, 스크린샷이(가) 표시된 사진

자동 생성된 설명

확인한 결과 10101에서 out이 1로 잘 출력되는 것을 확인할 수 있다.

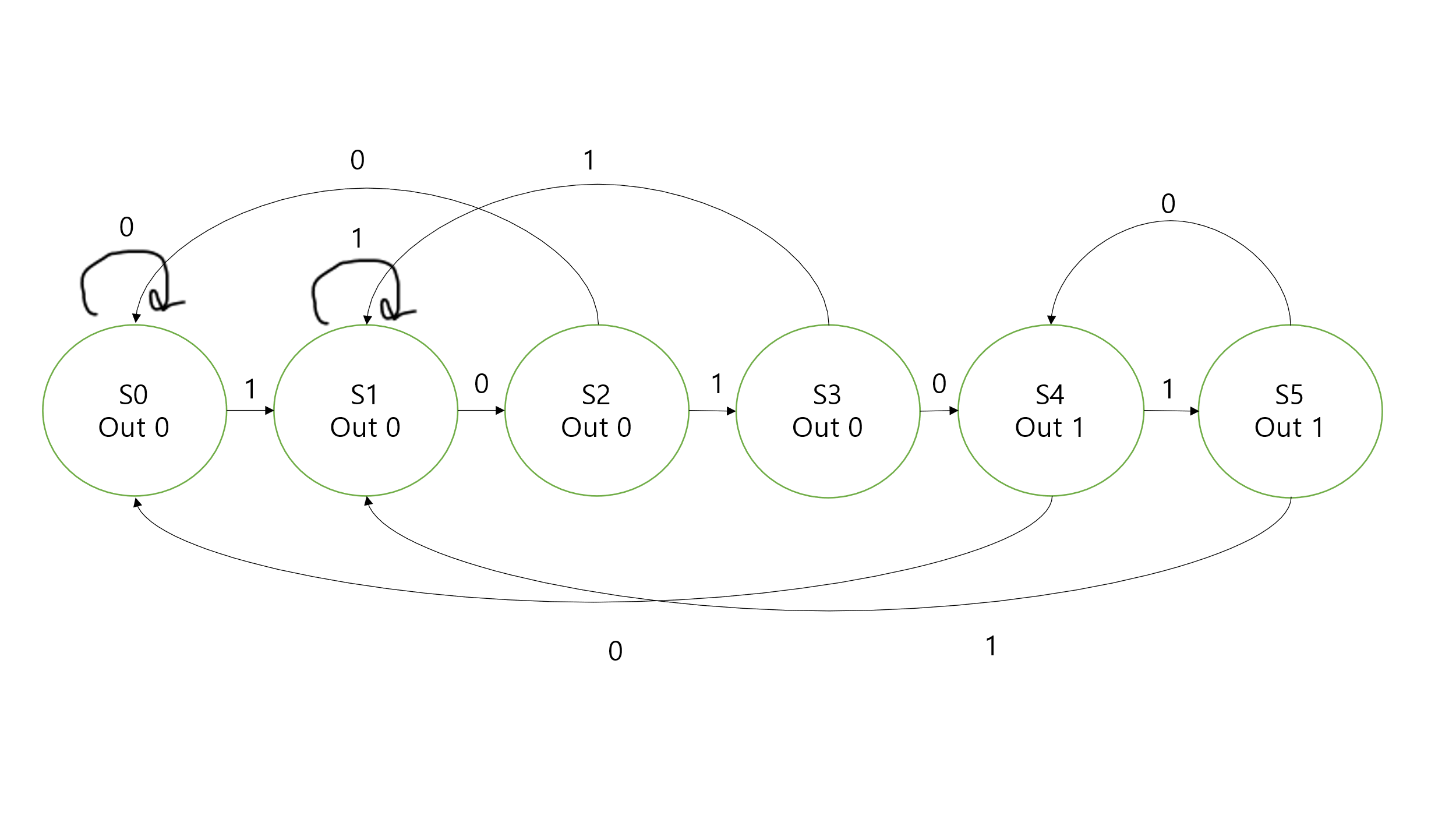
다음으로 moore machine도 살펴보자.

Moore machine의 state table은 다음과 같다.

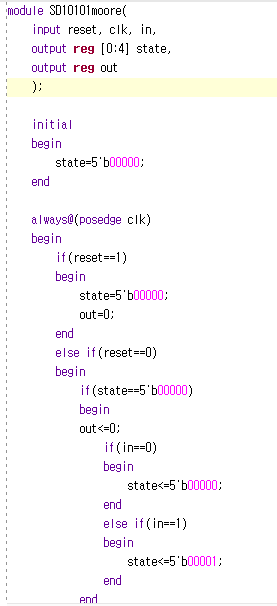
|  |  |  |  |
| --- | --- | --- | --- |
| **이전 상태** | **다음 상태** | | **출력** |
| **Input 0** | **Input 1** |
| **S0(00000)** | **S0** | **S1** | **0** |
| **S1(00001)** | **S2** | **S1** | **0** |
| **S2(00010)** | **S0** | **S3** | **0** |
| **S3(00101)** | **S4** | **S1** | **0** |
| **S4(01010)** | **S0** | **S5** | **0** |
| **S5(10101)** | **S4** | **S1** | **1** |

역시 mealy보다 state가 한 개 더 많은 것을 확인할 수 있다. 또한 state에 따라 출력을 결정하는 것도 확인할 수 있다.

State diagram은 다음과 같다.



이를 토대로 작성한 코드를 보자.

텍스트이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

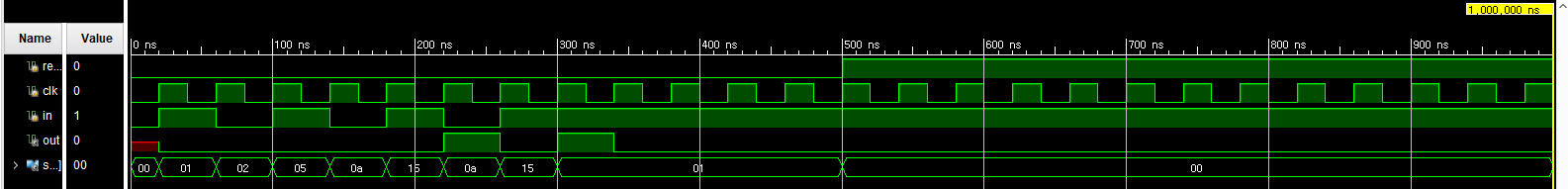
자동 생성된 설명

Testbench는 mealy machine과 동일하게 가져갔다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Simulation 결과는 다음과 같다.



역시 moore machine은 출력이 한 칸씩 밀려 180ns에서 10101이 관측되고 220ns에서 1이 출력되는 것을 확인할 수 있다.