SOC Design

한양대학교 공과대학 컴퓨터소프트웨어학부

최가온(학번: 2019009261)

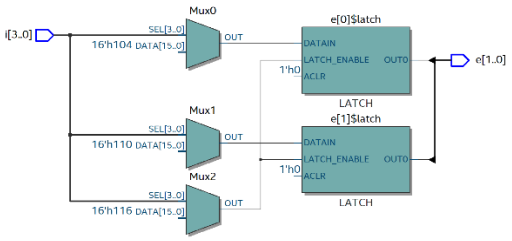
[Homework 1] Four different implementations of 4-to-2 Encoder

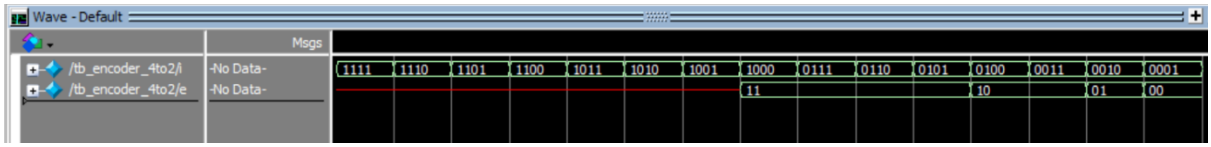
아래의 테스트벤치(testbench)를 기반으로 하여 총 4가지의 encoder 구현체를 테스트하였다. 각각에 대해 Netlist Viewer를 통한 구조와 테스트벤치를 적용한 waveform의 결과를 각각 나타내었다.

TestBench Code

https://github.com/Gaon-Choi/ITE4003/blob/main/Week05/encoder_4to2/tb_encoder_4to2.v

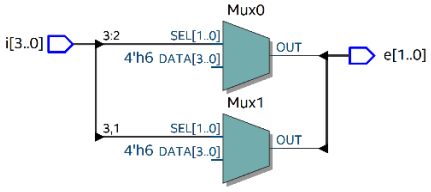
1) encoder_4to2_case

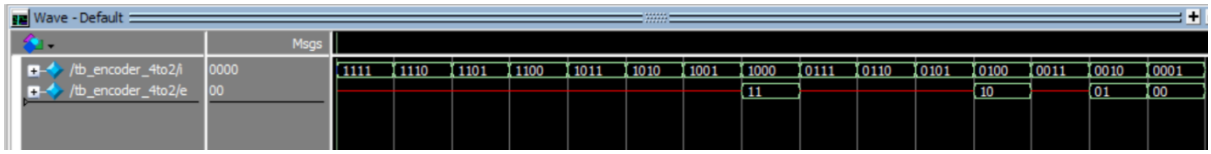




default가 없는 case문의 경우 명시되지 않은 값이 들어올 경우 unknown을 결과로 내보낸다. 0ps~ 7ps 사이에서, i의 값은 case문에 명시되지 않은 값이 입력으로 들어왔으므로 unknown 값임을 알 수 있다. 그러나, 8ps~9ps에서의 i값은 b'0111인데 이는 case문에 없는 값이다. 이를 통해 case문에 명시되지 않은 값이 들어올 경우 unknown을 출력하거나, 이전에 유효한 값이 있었을 경우 그 값을 일정하게 유지함을 알 수 있다. case문에 있는 값이 들어오는 즉시(그림에서는 b'0100), e의 값이 바뀌는 것을 확인할 수 있다.

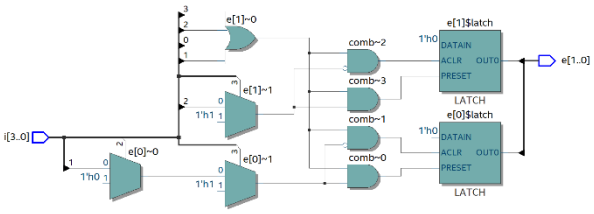
2) encoder_4to2_case_default

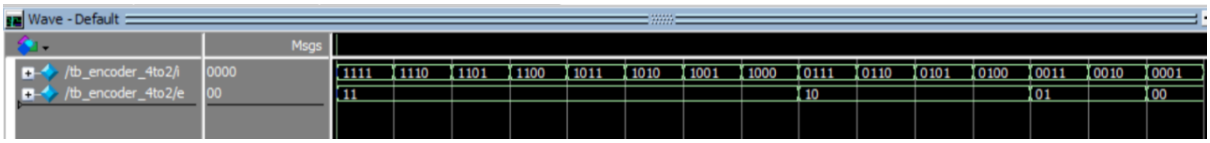




default문을 쓰지 않았을 때 8ps~11ps, 12ps~13ps 구간에서 이전의 e값으로 일정하게 유지되는 것과 차이가 두드러지게 나타난다. default문을 쓰지 않았을 때와는 달리, case문에 없는 값이 입력으로 들어올 경우 무조건적으로 unknown을 출력으로 내보내는 것을 알 수 있다. 추가적으로 casex문에서는 X와 Z 모두 don't care로 처리한다.

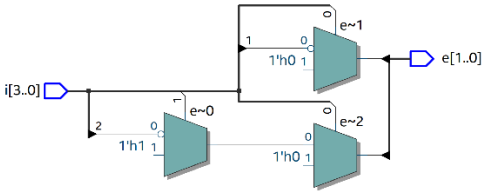
3) encoder_4to2_if

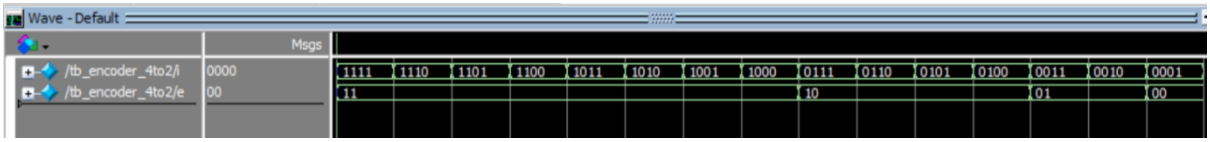




위 구현은 4개의 if문이 sequential form을 이루어 구성된 것을 볼 수 있다. 가장 위에 있는 if문부터 순차적으로 처리된다. 위에서부터 아래 방향으로 각각 i[0], i[1], i[2], i[3] 값을 검사한다. 4개의 if문 모두 e라는 하나의 값을 바꾸는 역할을 한다. 따라서, 가장 아래에 있는 if문에서 최종적인 e의 값을 결정하게 된다. 이를 i와 관련하여 설명하면, i의 most-significant bit 방향을 우선적으로 따라간다는 의미가 된다. 0ps ~ 8ps에서는 i[4]의 값이 1이기에 b'11을 출력하고, 8ps ~ 12ps에서는 i[3]의 값이 1이기에 b'10을 출력하고, 12ps ~ 14ps에서는 i[2]의 값이 1이기에 b'01을 출력하며, 14ps ~ 15ps에서는 i[1]의 값이 1이기에 b'00을 출력한다.

4) encoder_4to2_if_default





해당 구현 방식에서는 if – else if문을 사용하였다. if문을 직렬방식으로 쓴 것과 달리, 여기에서는 각각의 조건을 만족하지 않아야 남은 아래의 else if문을 체크하고, 만족하면 해당 부분에서 끝나기에 최종 값이 윗방향에 우선적으로 영향을 받는다. 마지막 else 부분까지 도달하려면 i의 4개 비트가 모두 0이어야 하는데 주어진 테스트벤치에는 i='b0000이 없으므로 전 구간에 출력값이 존재함을 알 수 있다. 결론적으로 i의 MSB부터 차례대로 보면서 1인 값이 가장 먼저 나오는 것끼리 묶이는 것을 확인할 수 있다.