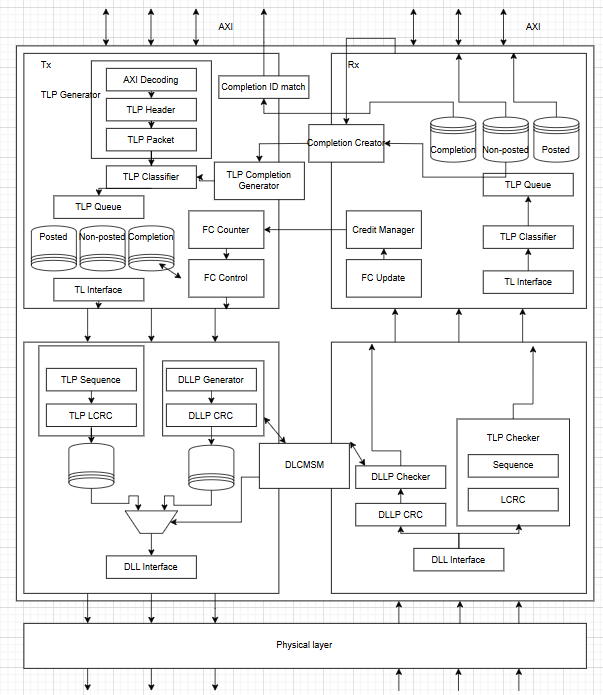
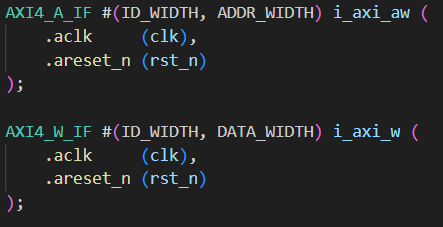
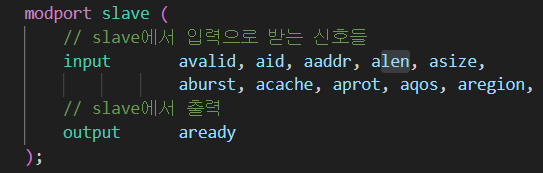
AXI4\_DECODING 모듈 설계 과정

<기본 아키텍쳐>

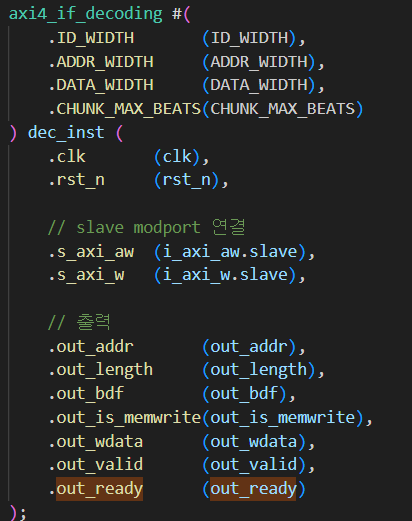


1. AXI DECODING 모듈 설계

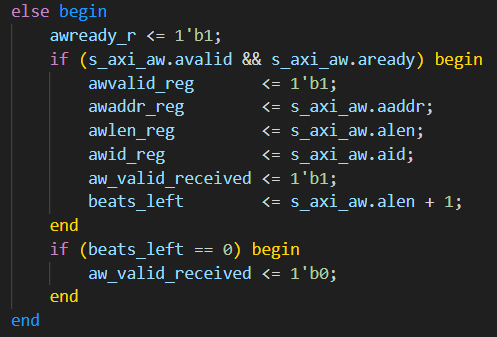
AXI4 인터페이스를 통해서 master에서 값을 받아 TLP로 처리를 하기 위한 AXI4\_DECODING 모듈 설계를 맡아서 하였다. 가장 처음 정리한 것은, AXI 인터페이스 측의 신호 중 사용할 신호와 사용하지 않을 신호를 구분 짓는 과정을 진행하였다. AXI4\_DECODING 측은 axi\_a slave로 다음과 같은 신호들이 인가된다.



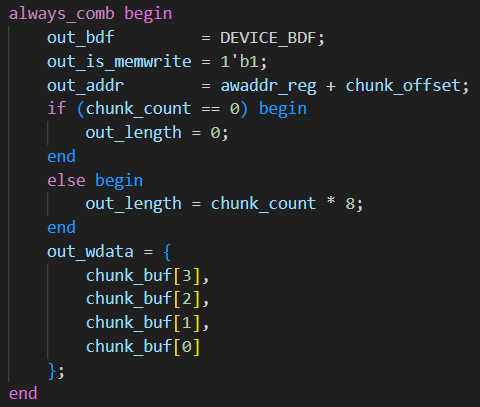
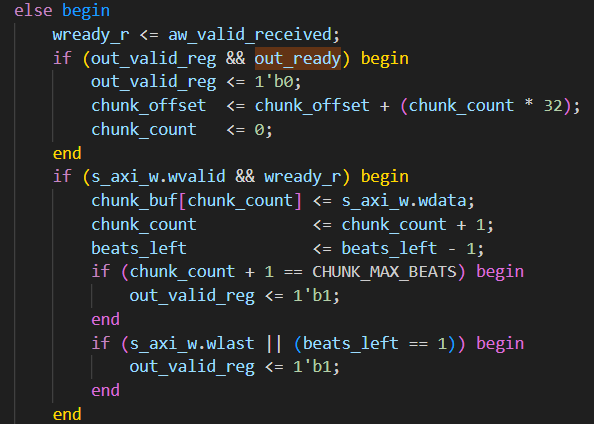
하지만 설계하고자 하는 PCIe에서는 availd부터 aburst까지의 신호만 사용하면 된다고 판단하여 해당 신호 기반으로 우선 top모듈을 작성하였다. 우측과 같이 top 모듈에서 interface를 선언해주고, 해당 interface를 활용하여 I/O신호를 다른 모듈에서 따로 신호를 인가해주지 않아도 ~~~



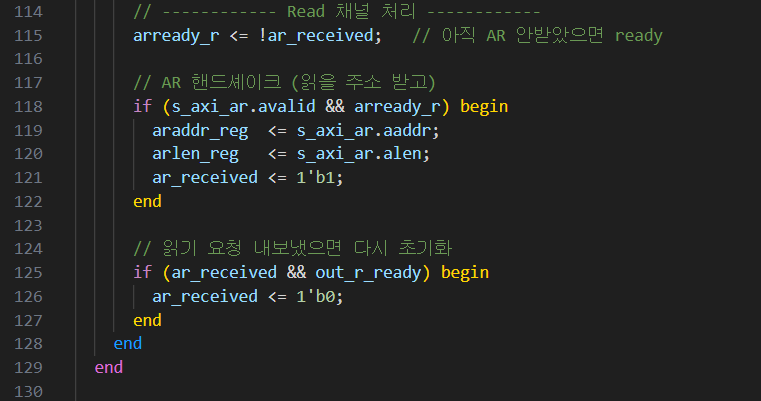
AW측과 W측을 통해 인가되는 output은 다음과 같이 구성을 하였고, 이를 이제 TLP 패킷화 모듈에서 받아서 TLP를 진행하게 설계하였다. TLP 설계단에서의 번거로움을 줄이기 위해 decoding 모듈에서 데이터를 최대한 잘 정리해서 보내주는 것이 매우 중요하다고 생각하여, PCIe specification에 부합하도록 data를 전송해주었다.



우선 AW 쪽 설계 흐름은 vaild와 aready를 통해 handshake를 진행하였고, beats\_left 변수를 통해 들어올 wdata의 크기를 미리 할당을 해주고, 값이 들어올 때마다 해당 값을 줄여나감으로써 wdata가 최종적으로 다 들어온 경우 이를 인지하고 데이터를 전송할 수 있도록 설계하였다.

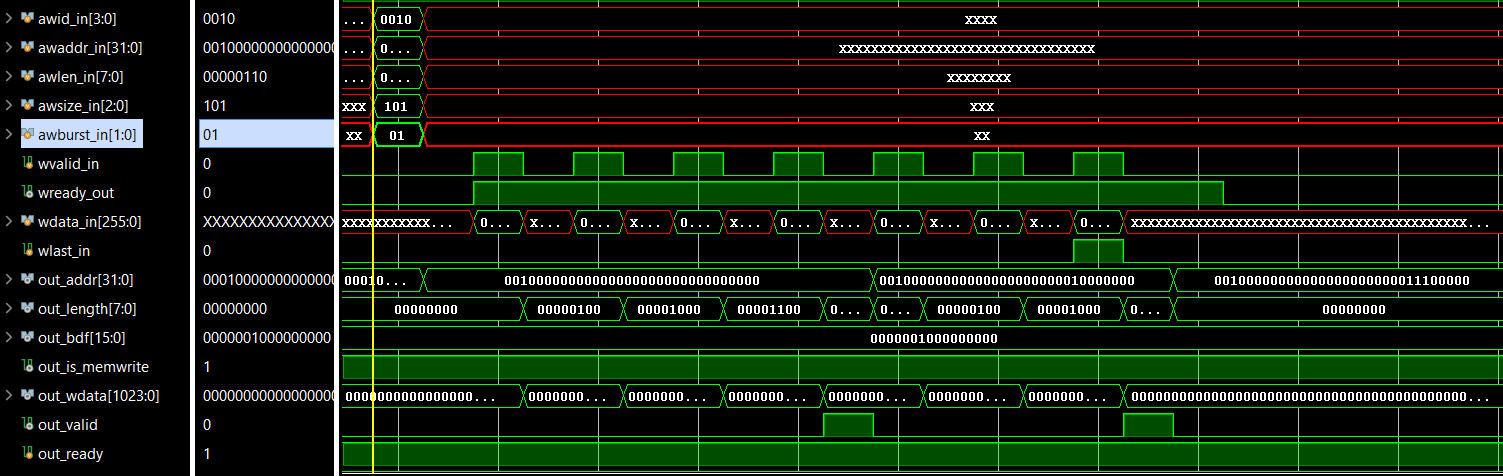


W측의 설계 흐름은 역시 handshake를 통해 연산의 여부를 결정하였고, AXI 측으로 wdata는 32B로 고정된 상태로 들어오지만 PCIe 측으로는 max\_payload\_size가 128B로 결정되어있으므로 미리 decoding 측에서 128B씩 묶어서 wdata를 전송해주었다. 또한, Burst type이 increase이므로 chunk가 가득찬 상태이며 wlast가 아직 들어오지 않은 상태라면 주소 자체를 증가시켜서 미리 보내준다면 TLP를 만드는 과정에서 추가적인 연산이 필요하지 않다고 판단되어, 주소 역시 TLP에 곧바로 받아서 사용할 수 있는 형태로 보내고자 하였다. 따라서 chunk\_offset를 통해 만약 128B가 가득찬 상태로 데이터가 전달이 되는 경우 주소를 128만큼 증가시켜서 address를 전달해주었다. 이때 두 개의 always 문에 같은 reg가 선언되는 경우 오류가 발생하여 하나의 always로 묶어서 이를 해결하였다.



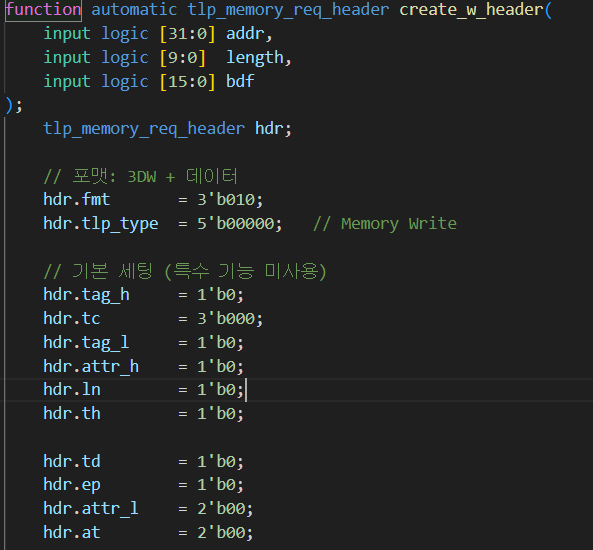
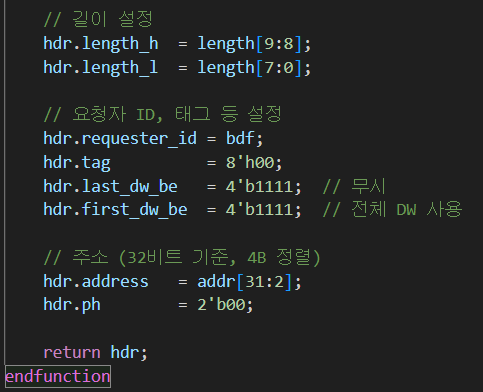
<read측 신호 decoding 모듈 핵심 코드>

Write와 무관한 채널을 사용하여 axi를 통해 address와 필요한 신호들을 tlp\_gen에 보내주도록 설계하였다.

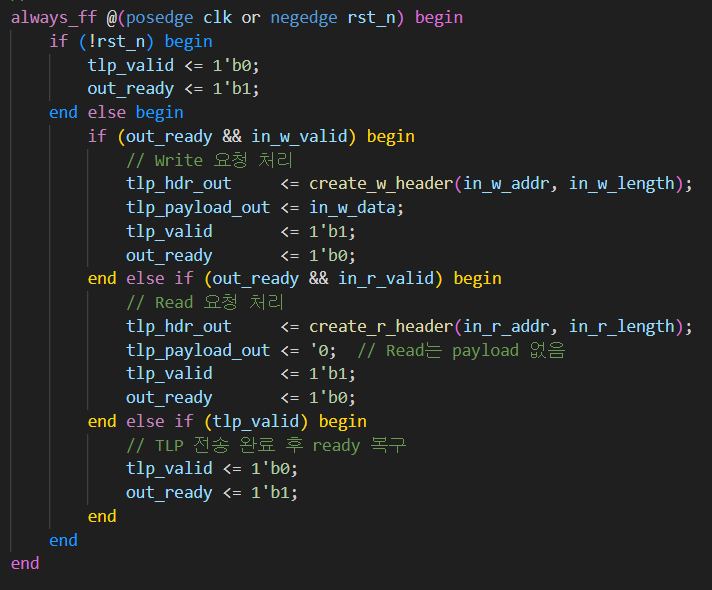


결과적으로 write resquest에 대해서 wdata와 address를 올바르게 decoding하여 출력하는 것을 확인할 수 있었다.

<TLP Header / TLP Packet 설계>

<TLP 헤더 함수> : read측과 write측을 type의 차이를 둔 채 각각 따로 작성.



<TLP generator> 핵심 코드

지금 작성한 코드는 write를 먼저 처리하고 read를 처리할 수 있도록 작성하였으나, 버퍼에 넣는 과정에서는 각각의 버퍼에 따로 저장하므로 해당 코드를 수정하여 곧바로 tlp를 버퍼에 넣을 수 있도록 수정할 예정.

Testbench 설계 과정

완성된 TLP generator 모듈을 테스트하는 testbench 코드를 구현하려 하였다. 완성된 모듈이 당시에는 컴파일 오류가 있어서 dummy DUT를 만들어 testbench를 시험해 보았다.

<Generator, driver, env, monitor, test, scoreboard를 이용한 testbench 설계>

텍스트, 스크린샷, 도표, 직사각형이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Generator, driver, env, monitor, test, scoreboard 모듈을 이용한 설계를 먼저 진행하였다. SystemVerilog tutorial의 과정을 따라가보려 하였다. 원래라면 generator에서 random 신호 생성 -> driver가 interface를 통해 DUT에 구동 -> 결과를 monitor를 통해 scoreboard로 보내고 예상 값과 비교하는 과정을 거친다.

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

<generator> -> 값을 random으로 넣어주기

Burst로 값을 넣어주는 상황을 생각해서 alen을 통해 burst 횟수를 정해주고, 그 만큼 wdata를 생성하기로 하였다. Transaction object라는 것을 활용하여 item을 mailbox로 driver에 전달하도록 하였다. PCIE\_PKG를 item 안에 포함하였더니 코드 수정을 많이 해야 한다고 느껴서 많은 수정이 필요하다.

텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Driver를 통해 값이 들어가는 부분만 보이게 하고 monitor와 scoreboard는 사용하지 않았다. Scoreboard로 비교하려면 tlp를 만들어내는 DUT를 설계해야 하는게 그게 어려워 일단 전체적인 틀만 완성하였다.

1. Testbench 설계

기존의 방식이 복잡하여 새로 testbench를 만들어 마찬가지로 dummy DUT로 시험해보았다.

이번에는 testbench를 하나로 만들어 그 안에 random하게 값을 넣어주는 task를 추가하였다. 그리고 testbench 상의 결과와 DUT를 통해 나온 결과를 비교하는 코드를 짜보았다. 이번에도 간단한 DUT를 설계하였는데 FSM을 이용하여 실제 TLP를 생성하여서 비교하는 것을 출력할 수 있었다. 다만 모든 신호를 사용하지 않고 간추려서 작동할 수 있도록 하였다. 추후 DUT 모듈이 완성되면 TB도 완성할 수 있을 것이다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

3. TL interface (구상)

TL과 DLL을 연결하기 위한 인터페이스이다. 관련 신호를 구상해보았다. 우리쪽 디바이스에서 tlp를 전송하고자 하는 경우 tl에서 dll로는 tlp 전송 요청 및 소모 크레딧(헤더, payload)를 보내고 dll에서 tl로는 현재 유효 크레딧에 따라 TLP 전송을 보류할지 결정한다.

추가적으로 디바이스의 RX 부분(receiver인 경우)에서는 수신한 tlp를 상위 인터페이스로 보내면 버퍼의 크레딧을 회복하고, 이를 dll에 전달하여 FC dllp가 생성가능하도록 해야 할 것으로 보인다

