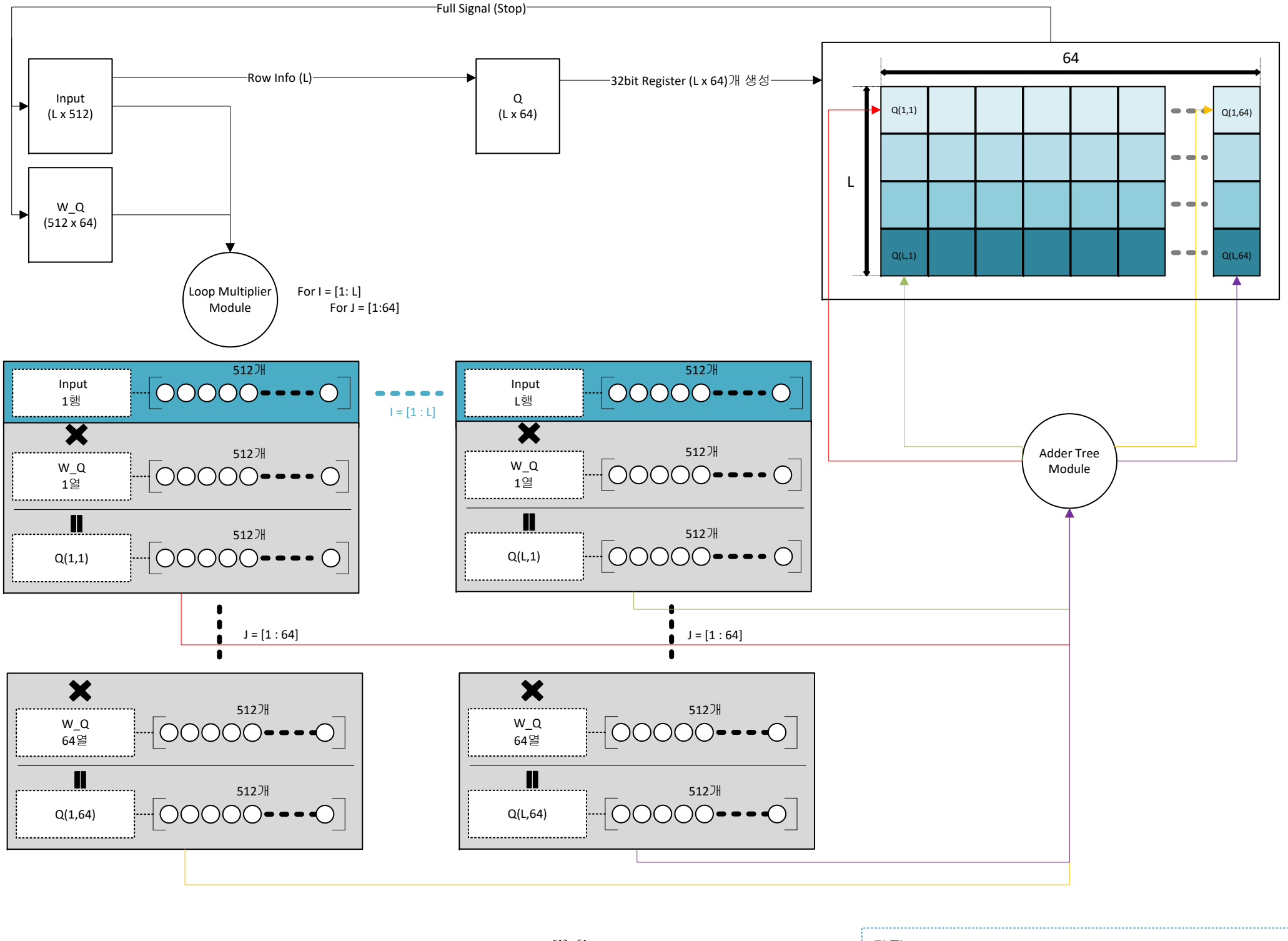
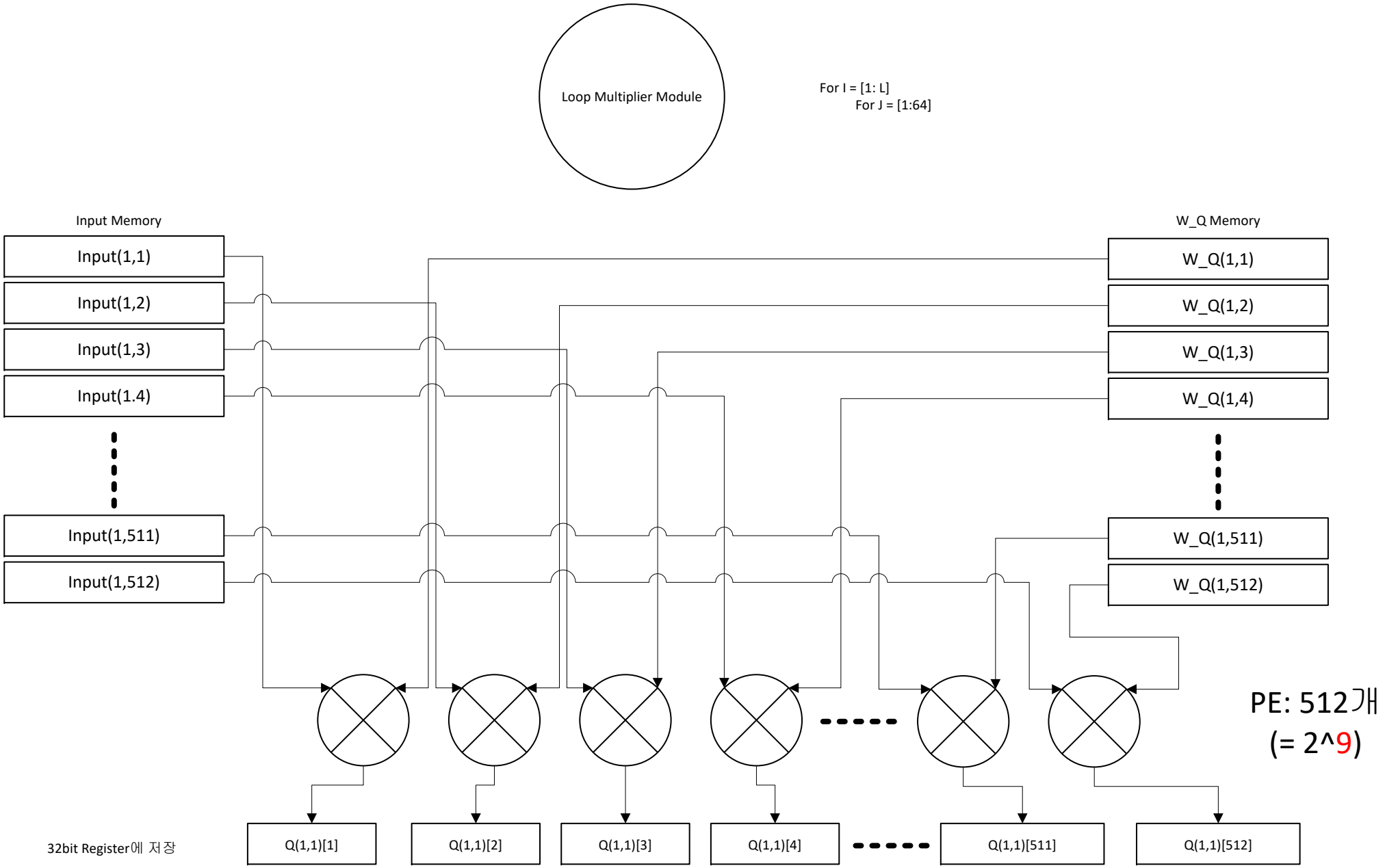

Full Signal (Stop)
Input
(L x 512)
W_Q
(512 x 64)
Row Info (L)
Q
(L x 64)
32bit Register (L x 64)개 생성
64
L
Q(1,1)
Q(1,64)
Q(L,1)
Q(L,64)
Loop Multiplier Module
For I = [1: L]
For J = [1:64]
Input
1행
512개
W_Q
1열
Q(1,1)
I = [1 : L]
Input
L행
W_Q
1열
Q(L,1)
Adder Tree Module
J = [1 : 64]
J = [1 : 64]
W_Q
64열
Q(1,64)
Q(L,64)

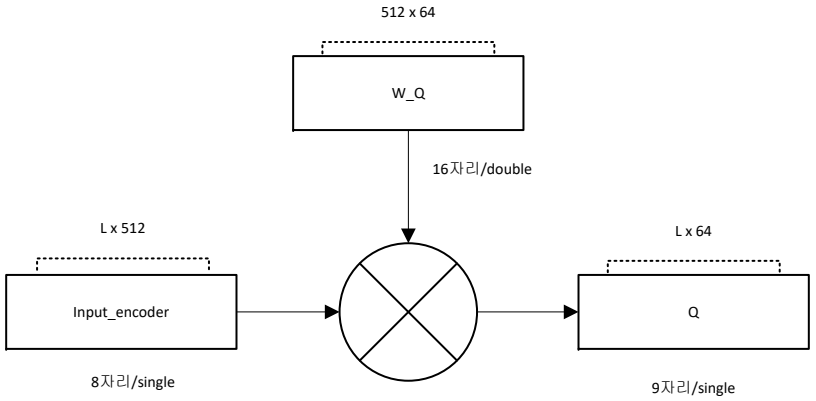


Min_n(PE) = (512 x 64) x 2 = 65k개
Max_n(PE) = (512 x 64) x 256 = 8388k개

Min(Register) = PE x 4B = 260kB
Max(Register) = PE x 4B= 33MB

Min(Clk) = (4clk x 64) x 2 = 512clk
Max(Clk) = (4clk x 64) x 256 = 6,5536clk

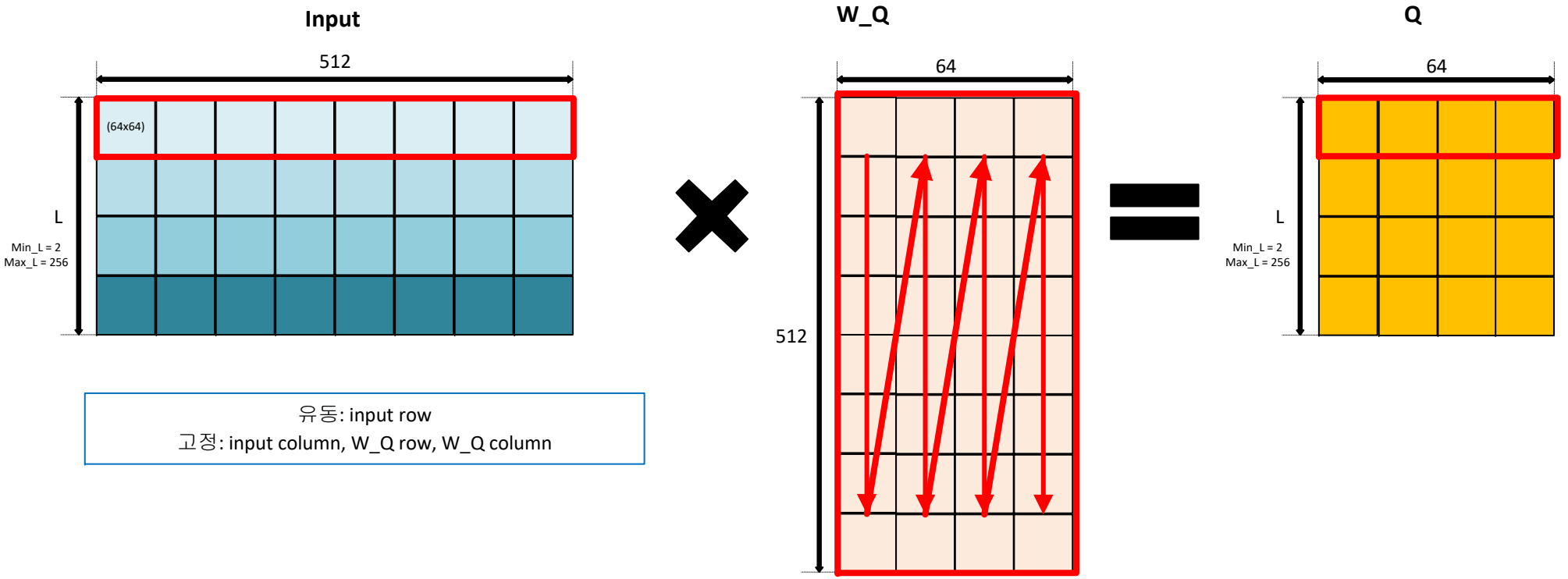
L(Sentence Length)
Min_L = 2
Max_L = 256
D_model = 512
N_heads = 8
D_model/n_heads = 64



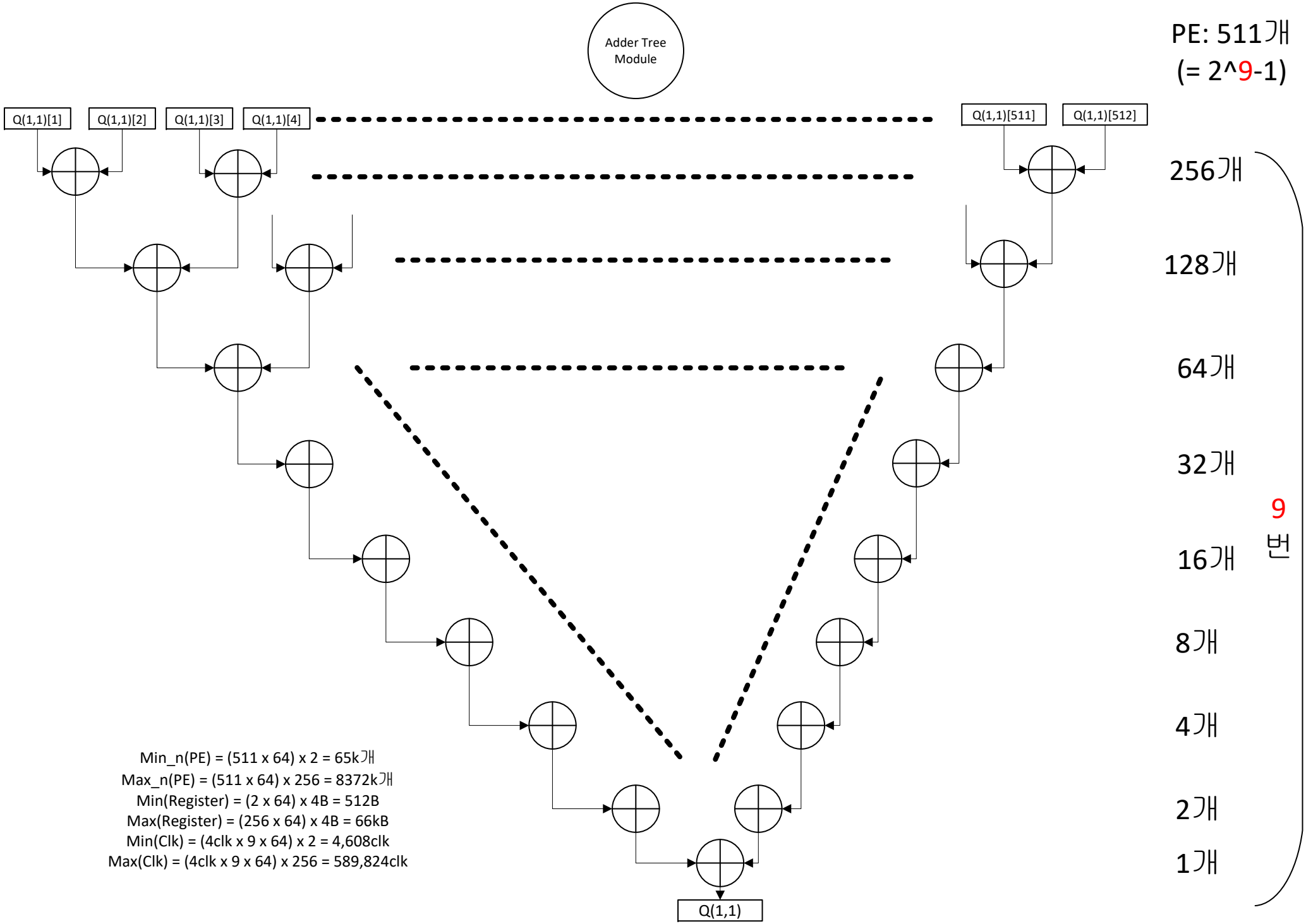
<단점>
1. 너무 병렬구조
2. 곱셈: 4clk / 덧셈: 36clk로 지연시간 차이 너무 큼
→ Critical Path: Adder Tree Module
3. 곱셈도 Adder구조 포함이므로 Adder 구조가 너무 많음
4. 곱셈에서 최대 33MB의 Register가 필요하므로 용량이 너무 큼
→ SRAM을 사용해야 하지만 그래도 무거움

<개선방안>
1. 곱셈과 덧셈연산에서 clk 비슷하게 맞추기
2. Input 행렬을 column 기준으로 잘라서 1cycle 연산이 끝났을 때 Q행렬의 모든 요소에 값이 하나씩 채워지게 하기
3. Q 행렬 요소에 adress를 부여하여 연산값이 주소를 보고 찾아갈 수 있게 하여 메모리 부담 줄이기
4. 덧셈은 한 번 더하는 구조에서 끝나게 설계
5. Input 행렬 불러오는 방식도 고안하기

Loop Multiplier: I = 1



유동: input row
고정: input column, W_Q row, W_Q column


Adder Tree Module
PE: 511개
(= 2^9-1)
Q(1,1)[1]
Q(1,1)[2]
Q(1,1)[3]
Q(1,1)[4]
Q(1,1)[511]
Q(1,1)[512]
256개
128개
64개
32개
16개
8개
4개
2개
1개
9
번
Q(1,1)
Min_n(PE) = (511 x 64) x 2 = 65k개
Max_n(PE) = (511 x 64) x 256 = 8372k개
Min(Register) = (2 x 64) x 4B = 512B
Max(Register) = (256 x 64) x 4B = 66kB
Min(Clk) = (4clk x 9 x 64) x 2 = 4,608clk
Max(Clk) = (4clk x 9 x 64) x 256 = 589,824clk