Verilog Bit Pattern Checker description

Minjae Kim
Data-Intensive Computing Systems Laboratory(DataLab), DGIST

목적

- **12-bit * 512 (6144-bit) 크기의 vector 하나가 input으로 들어오면, 12-bit pattern 4종류에 대해 일치하는 12-bit group(편의상 page라고 칭함)들을 빠르게 찾는 것이 목적**
  - pattern 4종류 중 무엇과 일치하는지는 상관없으며, pattern 4종류 중 하나와 일치하는 page들을 모두 찾아야 한다.
- **원래 C code에서는 pattern과 일치하는 page들을 하나하나 bit연산을 통해 비교해야 했는데, 하드웨어 가속기를 만들어 더 빠르게 pattern과 일치하는 page들을 찾고자 하는 것**
- **모든 page를 한번에 병렬적으로 검사할 수 있으면 좋겠지만, 그렇게 하려면 너무 많은 하드웨어 자원을 필요할 것으로 예상되어 여러 번에 나눠서 검사(6144-bit 정도는 무리없이 한번에 할 수 있다면 상관없음)**
  - 여러 번에 나눠서 검사할 때, 한번에 검사하는 page들을 편의상 'block' 이라고 칭함
- **Input은 12-bit * n 크기의 vector와 pattern(12-bit vector) 4개**
- **Output은 pattern과 일치하는 page number(index)의 배열 형태**
  - Page 3번과 12번이 pattern과 일치하는 True page라면, Verilog output은 [3 12]의 reg array이다.

목적

- 가령, input이 6144-bit가 아니라 288-bit여서 12-bit group(page) 24개로 구성되며 page 8개마다 block 1개로 묶여 있다면, block과 page의 구성은 다음 그림처럼 표현될 수 있음
  - Verilog 상에서 들어오는 input array가 다음과 같다면,
  - 288'h111_234_567_890_abc_222_333_012___123_234_111_345_444_678_abc_111___666_777_888_111_222_666_000_fff
  - Page 0은 fff, Page 1은 000, Page 2는 666, … Page 23은 111이다.
  - Block 단위로 검사하는 모듈을 하나 만들어서, 그 모듈이 Block 개수(3개)만큼 작동하면 된다.

Page 'n' : 12-bit

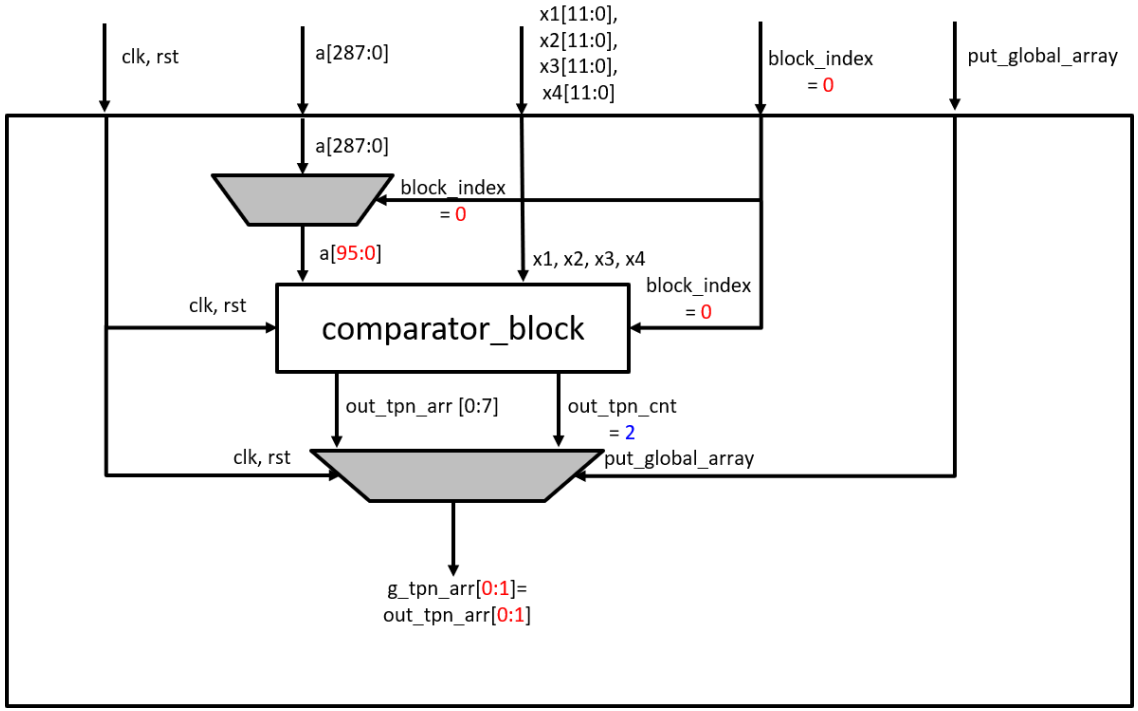
2 clock cycle 동안 Block 하나에 들어있는 8개의 page들을 4가지의 Pattern과 비교

Pattern x1, x2, x3, x4

Block 2				Block 1				Block 0			
Page 23	Page 22	Page 21	Page 20	Page 15	Page 14	Page 13	Page 12	Page 7	Page 6	Page 5	Page 4
Page 19	Page 18	Page 17	Page 16	Page 11	Page 10	Page 9	Page 8	Page 3	Page 2	Page 1	Page 0

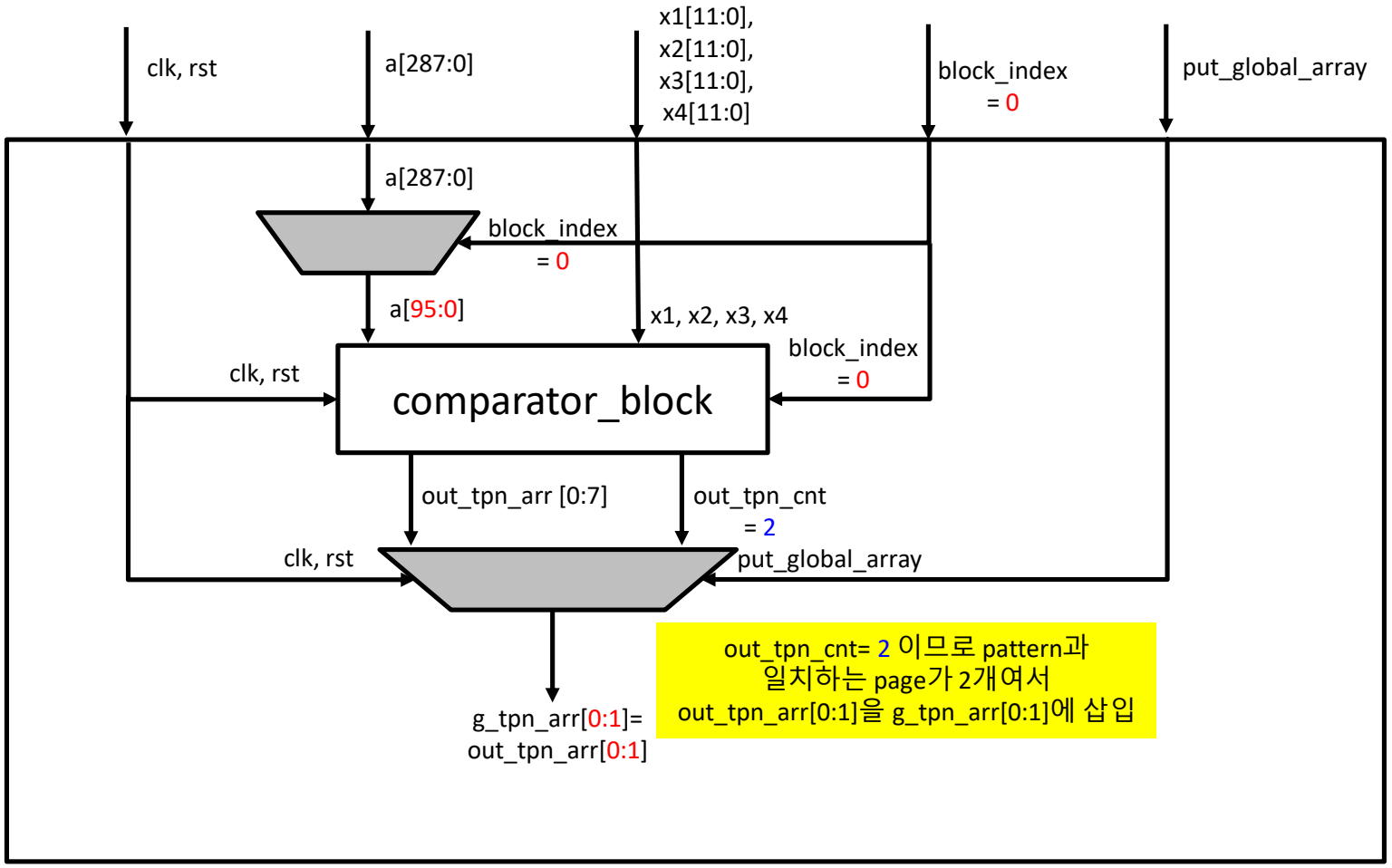
구조 설명

- a: input array
- x1, x2, x3, x4: pattern
- block_index: 어떤 block을 검사할 지 0부터 순서대로 알려주는 신호(input으로 하지 않고 design.sv 내에서 자체적으로 카운팅되는 것이 더 바람직하겠지만 빠른 구현을 위해 testbench에서 넣어주는 상황)
- put_global_array: comparator_block에서 나오는 out 중 홀수 번째 clock cycle(2번 당 1번)에 나오는 것만 올바른 out이므로, 홀수 번째 posedge에만 g_tpn_arr을 채우기 위한 신호
- out_tpn_cnt: 현재 block_index에 대응되는 block 하나를 검사할 때, 그 block에 포함된 page 중 pattern과 일치하는 true page 개수
- out_tpn_arr: true page들의 number(index)를 담은 array
- g_tpn_arr: 최종 output으로, out_tpn_arr를 FIFO 방식으로 모아놓은 형태의 array이다.



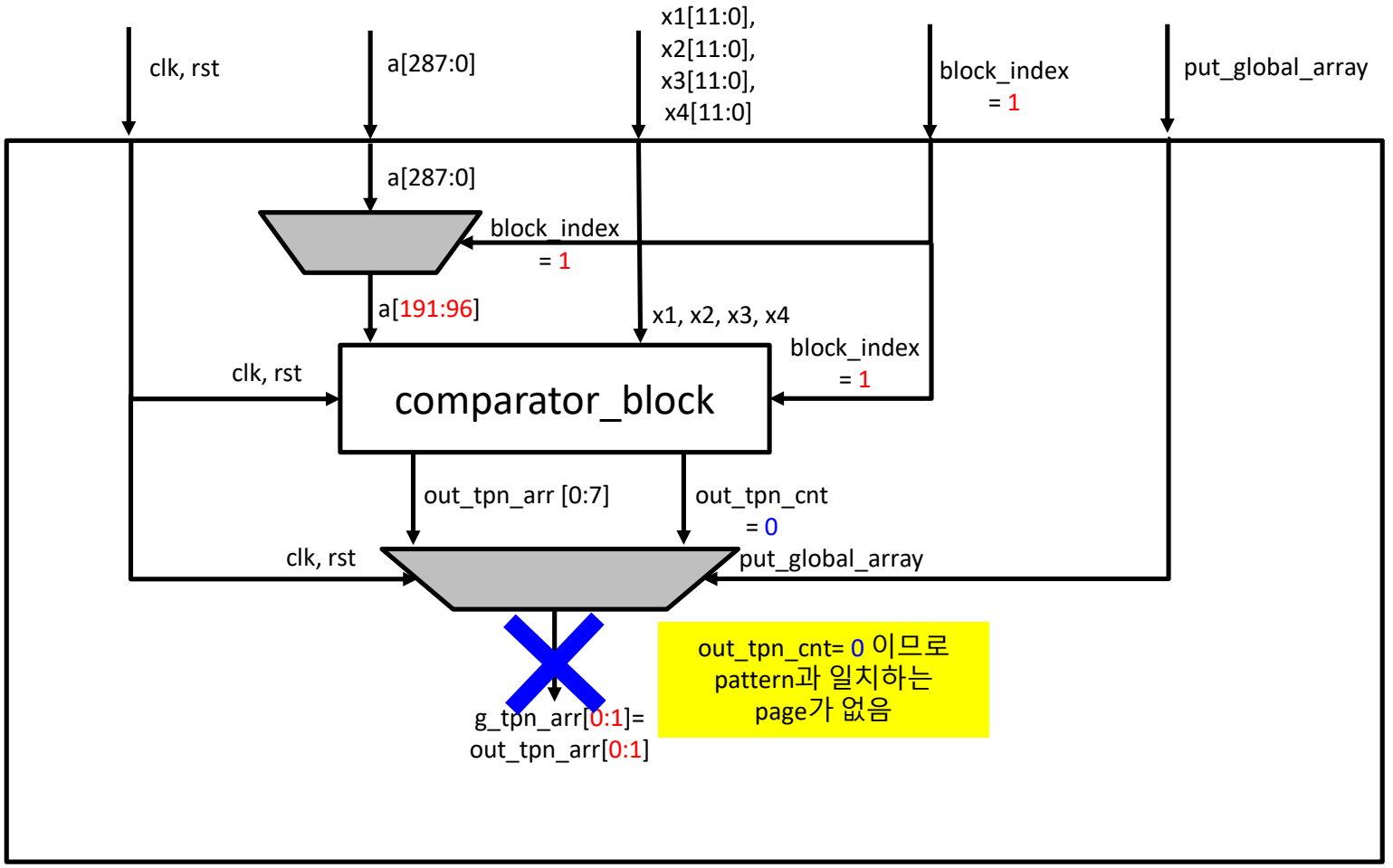
동작 설명

- block_index = 0일 때 동작 과정 (첫번째 block 검사)
  - 빨간색: block_index에 의해 결정되는 수 / 파란색: comparator 결과(input)에 따라 결정되는 수



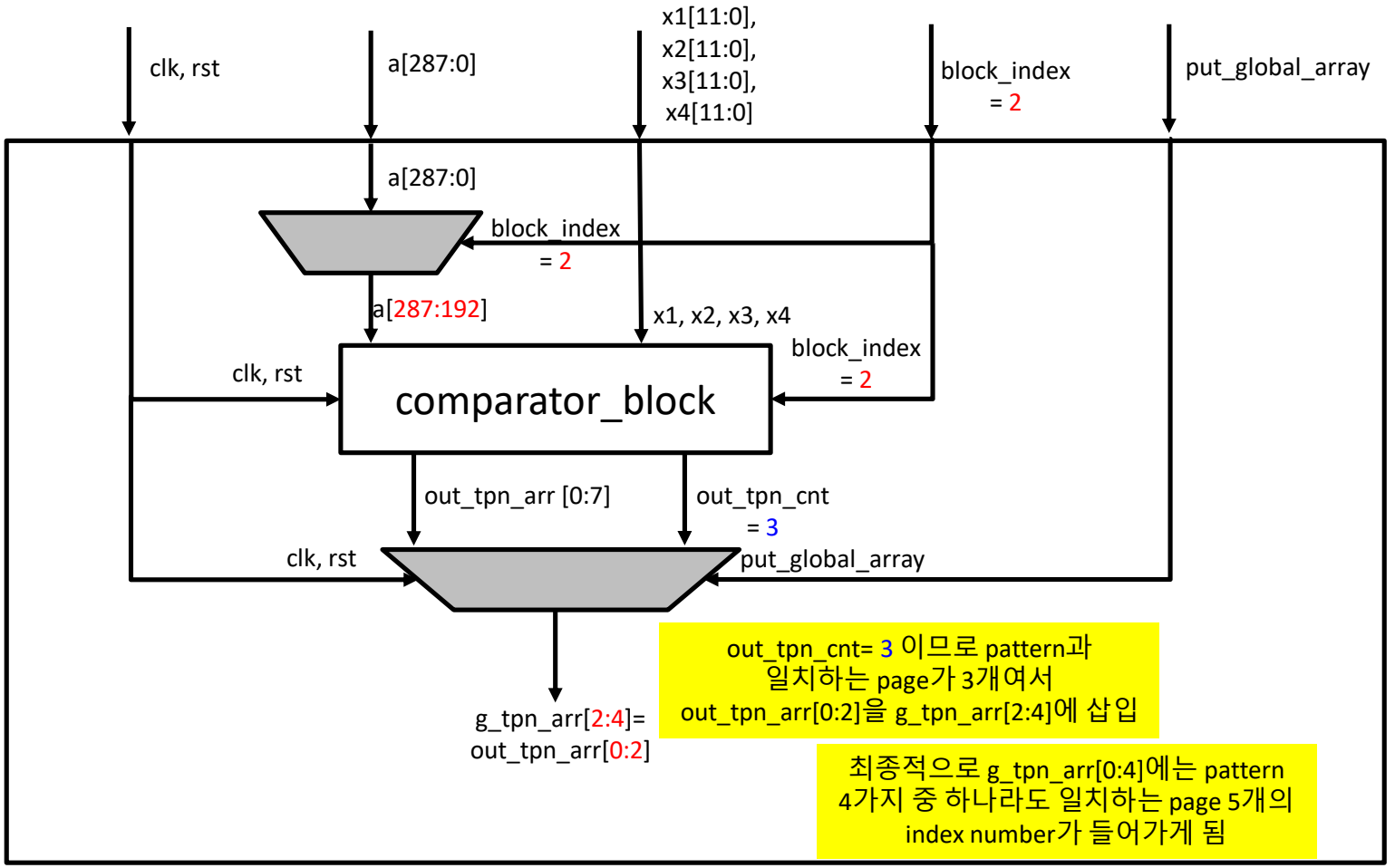
동작 설명

- block_index = 1일 때 동작 과정 (두번째 block 검사)
  - 빨간색: block_index에 의해 결정되는 수 / 파란색: comparator 결과(input)에 따라 결정되는 수



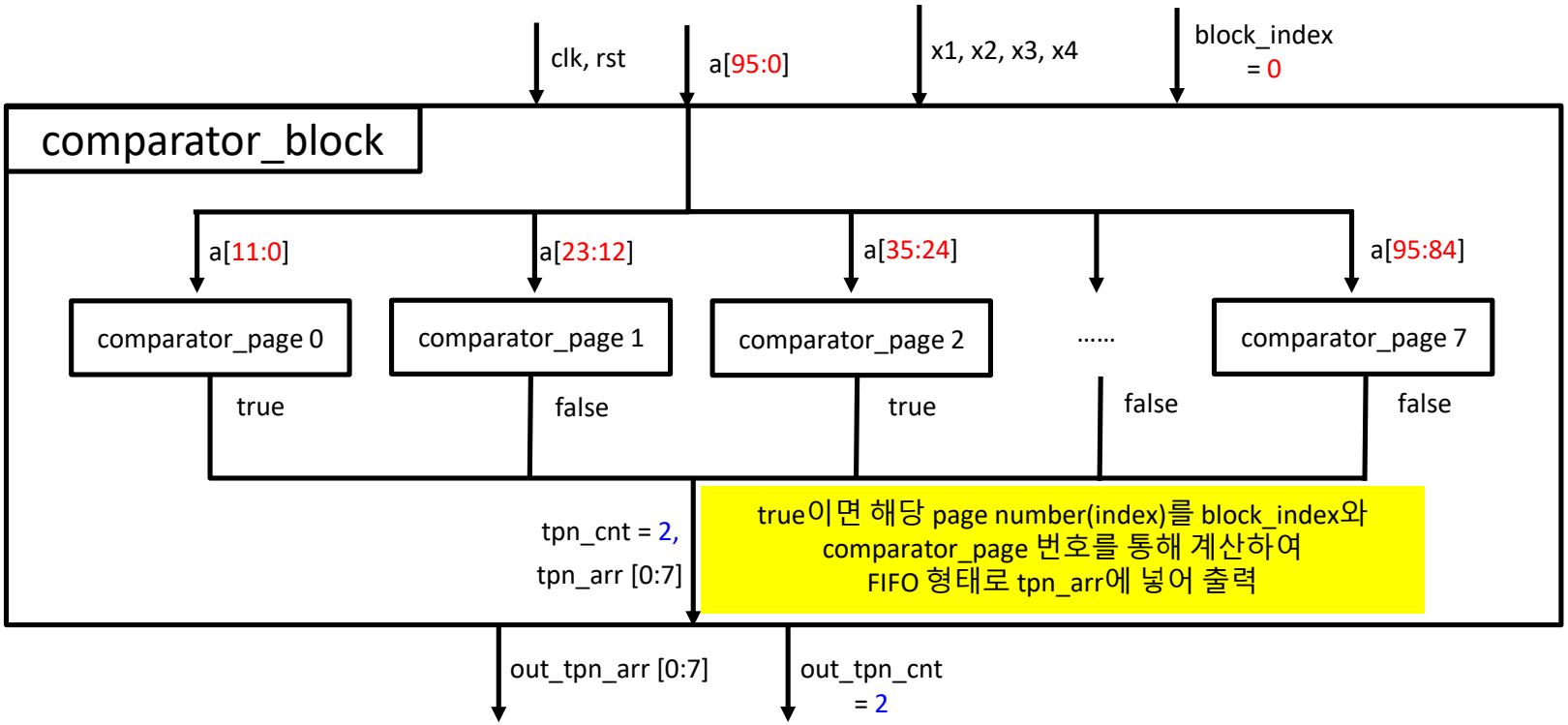
동작 설명

- block_index = 2일 때 동작 과정 (세번째 block 검사)
  - 빨간색: block_index에 의해 결정되는 수 / 파란색: comparator 결과(input)에 따라 결정되는 수



동작 설명

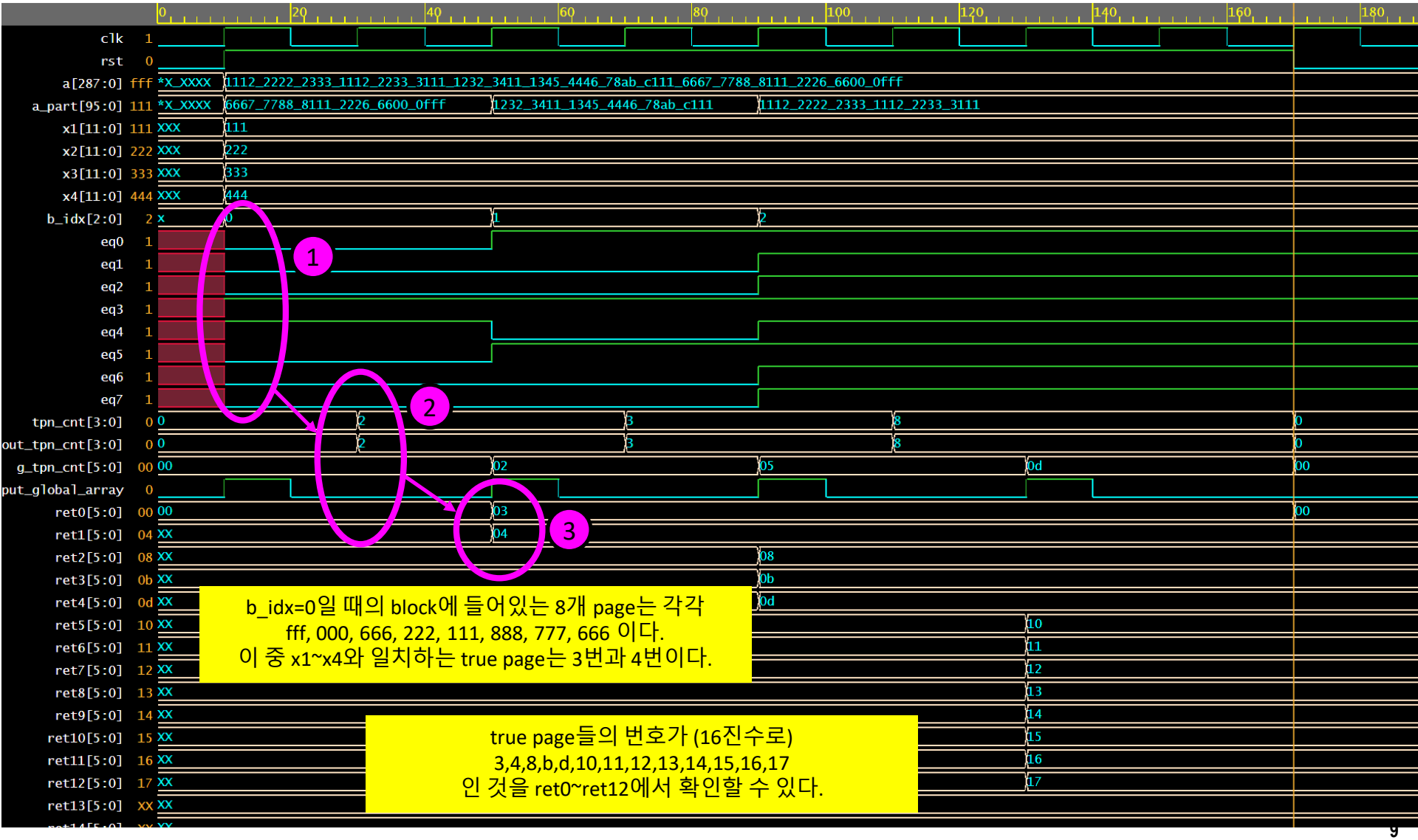
- comparator_block 모듈 설명
  - comparator_page는 12-bit input을 4가지 pattern과 비교해서 true/false를 출력
    - 내부는 4-bit comparator 12개(한 pattern당 3개)로 구성됨



결과 설명

- Waveform

[Block 0의 동작 과정]
1. page 8개 받아서 검사
2. true page의 수와 번호 추출
3. 최종 output array에 삽입



b_idx=0일 때의 block에 들어있는 8개 page는 각각
fff, 000, 666, 222, 111, 888, 777, 666 이다.
이 중 x1~x4와 일치하는 true page는 3번과 4번이다.

true page들의 번호가 (16진수로)
3,4,8,b,d,10,11,12,13,14,15,16,17
인 것을 ret0~ret12에서 확인할 수 있다.

기타 질문사항들

- **Verilog** 초심자이기에 좋은 방식으로 구현되어 있지 않음

- **Output**을 **reg array** 형태로 구현했는데, 그냥 **output reg** 하나에만 넣게 해서 **true page number**를 찾을 때마다 **output reg**가 **update**되게만 해도 되는 것인지**?**
- 여러 번에 나눠서**(block** 단위로**)** 검사하게 했는데, 하드웨어 자원 입장에서 이렇게 하지 않고 한번에 해도 충분한지**?**
- **Testbench.sv**에서 **block_index, put_global** 신호를 수동으로 설정하도록 구현되어 있는데, 이렇게 해도 되는 것인지 **/ Design.sv** 안에서 자체적으로 관리해야 한다면 어떻게 해야 하는지**?**