|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 입력(NM) / 스탯 | 청결 | 포만 | 재미 | 상쾌 |
| 00 (목욕) | + | - | - | 0 |
| 01 (간식) | 0 | + | + | 0 |
| 10 (산책) | - | 0 | + | - |
| 11 (잠자기) | 0 | - | 0 | + |

**디지털 시스템 설계 최종 보고서**

**양준영, 박민혁, 김지민**

* **board.v**

먼저 사용자로부터 입력을 받고 그에 따른 led 출력이 어떻게 이루어졌는지에 관한 설명이다.

* 각 스탯은 0과 1로 이루어지며 +는 0에서 1로 (해당 스탯이 이미 1인 경우 그대로 유지), -는 1에서 0으로(해당 스탯이 이미 0인 경우 그대로 유지), 0은 스탯 유지를 의미한다. 만약 1100에서 00버튼을 누르게 되면 청결 스탯은 이미 1이므로 1을 그대로 유지, 포만 스탯은 1에서 0으로, 재미 스탯은 이미 0이므로 그대로 유지, 상쾌 스탯도 0으로 그대로 유지되어 1000으로 상태 변화가 된다.

이를 표로 나타내면 다음과 같다.

테이블이(가) 표시된 사진

자동 생성된 설명

state가 4비트로 표현되므로 JK FF 4개로 구현하고자 했다. 리셋 버튼을 누르면 0으로 초기화 되게끔 설계했다. 또한 positive edge로 설정하였는데, 이는 버튼이 눌리는 순간에 상태가 변화하도록 하기 위함이다. (뒤에 설명되겠지만, clk는 버튼이 눌려야 들어오도록 설정되었다.)

텍스트이(가) 표시된 사진

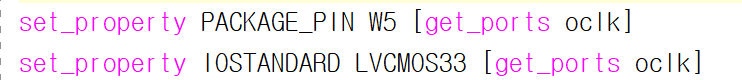
자동 생성된 설명

최상위 비트에 해당하는 부터 최하위 비트인 까지 식으로 표현하면 다음과 같다. (NM은 입력값)

텍스트, 전자기기이(가) 표시된 사진

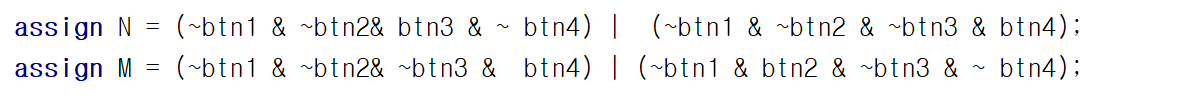
자동 생성된 설명

이제 회로를 보면, LED와 스위치 입력을 외부에서 받고 있으므로 아래와 같이 적절한 포트와 서로 연결해준다. 여기서 btn1,2,3,4는 각각 00,01,10,11입력값을 의미하며 led0,1,2,3은 청결, 포만, 재미, 상쾌 순으로 스탯을 의미한다.

테이블이(가) 표시된 사진

자동 생성된 설명

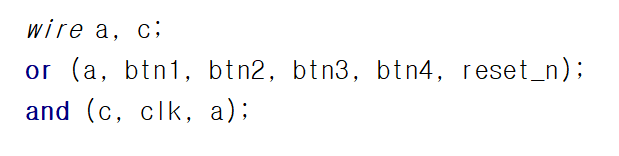
N과 M은 누르는 버튼에 따라 알맞은 값이 들어와야 하므로 (예를 들어 01에 해당하는 btn3를 눌렀을 때 N은 0, M은 1이 되어야 한다.)



이와 같이 식을 대입해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명 그리고 FF은 btn1,2,3,4와 reset 버튼이 눌릴 때에만 반응하도록 설계해야 한다.



왼쪽과 같이 같이 버튼 입력과 reset 버튼, 클록 신호를 묶은 뒤 오른쪽과 같이 클록 신호 입력 부분에 넣어준다. 이렇게 하면 버튼이 눌릴 때에만 클록 신호가 들어가서 안 눌릴 땐 현재 값을 유지하도록 한다.

* **oper.v**

이미지 출력을 설명하기 이전, 이미지 출력에 필요한 여러 기능들을 어떻게 구현했는지 설명하겠다.

1. JKFF

초기값은 0으로 설정하고, reset\_n 값이 0이 되면 0으로, 1이면 현재 값을 유지하는 JKFF이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. oJKFF

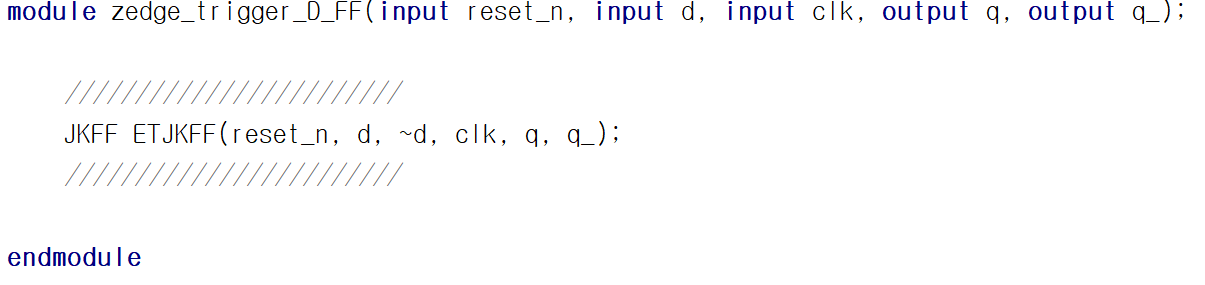
초기값이 1인 JKFF이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. zedge\_trigger\_D\_FF

초기값을 0으로 하는 FF이다. 1)JKFF을 활용한다.



1. edge\_trigger\_T\_FF

초기값을 0으로 하는 FF이다. 1)JKFF을 활용한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. lt10

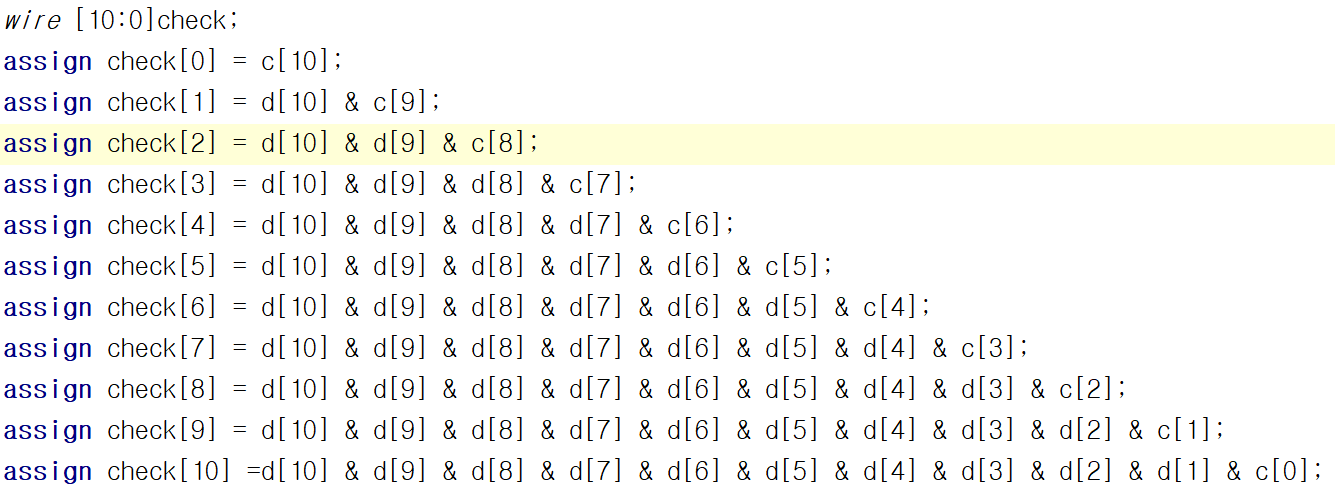
두 개의 11-bit input a, b에 대해 크기 비교를 하는 module이다. a보다 b  
(뒤에 들어온 input)이 더 큰 경우 output이 1이 된다.

텍스트, 테이블이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

우선 왼쪽과 같이 각 자릿수를 조사하여 a보다 b의 자릿수가 더 큰 경우에만 1이 저장되도록 한다. 그 다음 오른쪽과 같이 자릿수가 같은 경우에만 1이 저장되도록 한다.



최상위비트부터 차례대로 비교해가며 해당 비트 이전 비트가 모두 같은 숫자였으며 (즉, 해당 비트 이전에 해당하는 d값이 모두 1이어야 함.), 해당 비트의 값이 a보다 b가 더 크면(c[i]가 1) check[i]의 값이 1이 된다. check의 값이 하나라도 1이 되면 a가 b보다 작다는 것이므로 output이 1이 된다.

1. gt10

c[i]값을 할당할 때 a대신 b에 ~가 붙는다는 점 외에는 위 lt10과 과정이 동일하다. (즉, a가 b보다 큰 경우 output이 1이 된다.

1. mux2to1

2:1 멀티플렉서를 설계한 것이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

* **VGA 화면 출력**

1. 640X480, 60Hz로 화면을 출력하기 위해서는 clock을 25.175MHz로 만들어 주어야 하므로 들어오는 original clock을 clock divider를 통해 25.175MHz로 만들어준다.
2. Clock divider는 original clock이 2번 반복될 때 clock이 한 번 작동하게 만들어주며 이는 counter를 통해 2자리 비트값에서 최상위이 비트 값을 따르도록 한다.
3. 그 이후 H Count, V Count 값을 증가시키기 위해 Counter를 사용한다.
4. H Count와 V Count의 Counter를 synchronous 방식으로 만든다.
   1. H Count는 0부터 799까지 반복되도록 하며
   2. V Count는 0부터 525까지 반복되도록 구현한다.
5. Hsync는 한 line이 시작될 지점을 알려주는 것으로 H Count의 값이 96보다 작을 때 1이 되도록 구현한다. 이 때 구현해 놓은 2:1 MUX와 less than 함수를 이용한다
6. Vsync는 한 프레임이 시작할 때를 알려주는 것으로 V Count의 값이 2보다 작을 때 1이 되도록 구현한다. 구현 방법은 Hsync와 동일하다.
7. 화면이 출력되는 H Count의 영역은 143에서 784까지, V Count가 35에서 515까지일 때 RGB에 4bit를 전달해주어 해당 위치에 맞는 bit를 전달해준다

(이 때, 만들어 놓은 Less than, Greater than을 사용하여 크기 비교를 한다)

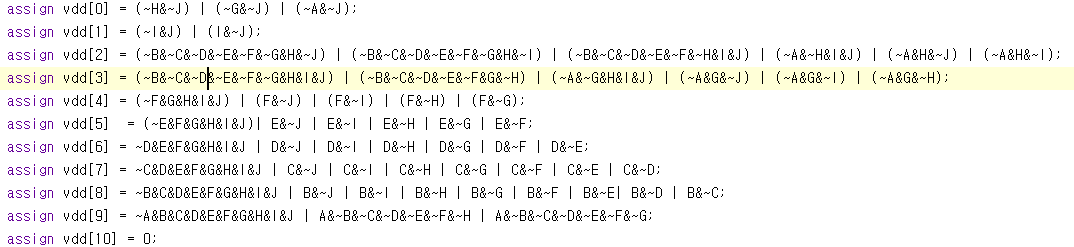
1. 기존 계획은 이미지를 출력하기 위해서 ROM을 통해 해당 위치에 맞는 RGB 값을 전달해주는 것이였으나 시간 상 등의 문제로 상태에 맞는 색을 화면에 출력하는 것으로 대체하였다. (시도한 방법은 밑에 기술)
2. 화면에 RGB 각 4bit는 구현해놓은 회로를 통해 얻은 상태 4bit에서 순서를 바꾸어 전달해주어 각 상태에 따라 여러 색이 나타날 수 있도록 한다.

* **Clock Divider**
* edge triggered D FF를 이용하여 2 bit Counter를 구현하고 최상위 비트를 따르도록 clock output를 구성한다.
* **H Counter**
* H Counter는 H Count를 0부터 799까지 증가시키는 동기 계수기로 구현하였으며 799가 되면 최대값에 도달했을 때 max라는 것을 전달해준다.
* 구현은 D FF로 하였으며 식은 다음과 같다. (첨부 코드 확인 요망)

텍스트이(가) 표시된 사진

자동 생성된 설명

* **V Counter**
* H Count가 Max라면 V Counter를 작동시킨다. 위처럼 0부터 525가 될 때까지 반복되도록 동기 계수기로 구현한다.
* 구현은 D FF로 하였으며 식은 다음과 같다.



* **Sync**
* 화면을 구현하기 위한 Vsync, Hsync는 위의 내용을 따라 값을 부여한다.
* **RGB 비트 할당**
* 화면 영역이 143에서 784, 35에서 515의 범위에서 RGB값을 맞는 값을 전달하고 이외의 범위에서는 0을 부여한다.
* 화면이 영역 안일 경우, 현 상태 led[3:0]의 값을 잘 할당하여 상태에 따라 다양한 화면 색을 출력할 수 있도록 한다.
* 색 할당은 밑의 코드대로 하였다

테이블이(가) 표시된 사진

자동 생성된 설명

* **이미지 출력을 하기 위해 시도한 내역**

먼저 총 16개의 state에 해당하는 포닉스의 그림을 그려서 640x480 사이즈로 만들었다. 이 파일들을 RGB 각각 4비트 출력을 갖는 BASYS3으로 이미지를 출력하기 위해 파이썬의 Pillow 라이브러리를 활용하여 16개의 이미지를 8비트의 RGB로 추출한 다음 16으로 나누어 다음과 같이 16진수로 텍스트 파일에 저장하였다.

F F F

F F F

F F F

:

F F F

그리고 이미지를 포함하는 데이터를 베릴로그 내에서 선언하기 위해 2차원의 reg배열을 필요한 만큼 선언하는 코드를 만들어주는 코드를 파이썬으로 구현하여 다음과 같은 파일을 만들어서 베릴로그에 선언하였다.

reg [2559:0] red[0:15][0:479];

assign red[0][0][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFFFF…

assign red[0][1][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFFFF…

:

assign red[0][479][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFF…

reg [2559:0] green[0:15][0:479];

assign green[0][0][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFFFF…

assign green[0][1][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFFFF…

:

assign green[0][479][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFF…

reg [2559:0] blue[0:15][0:479];

assign blue[0][0][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFFFF…

assign blue[0][1][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFFFF…

:

assign blue[0][479][2559:0] = 2560'hFFFFFFFFFFFFFFFFFFFFFFFFFF…

그러나 베릴로그 실행시 계속 선언부에서 syntax error가 발생하여서 선언부를

assign red[0][0] = 2560’hFFFFFF… 으로도 바꾸어봤는데 여전히 에러가 발생하였다.

따라서 시간관계로 이미지의 출력은 포기하고 각 state별로 다른 색상을 출력하는 코드를 다음과 같이 짰다.

이 부분은 출력하는 이미지가 범위 안에 있는지 체크해주는 코드이다. HMIN과 HMAX, VMIN과 VMAX를 통해 화면이 표시되는 범위를 정하였다. lt10과 gt10은 각각 10비트의 첫 번째 매개변수와 두 번째 매개변수의 크기를 비교해주는 모듈이다. 두 번째 매개변수가 첫 번째 매개변수보다 크거나 작은 조건을 만족하면 세 번째의 매개변수에 1을 전달한다. 네 개의 매개변수가 모두 1의 값을 가지면 현재 이미지 출력 대상 픽셀이 화면 안에 있다는 의미이다. 이후 rcolor, gcolor, bcolor에 led의 출력 상태를 MSB에 저장한 다음, 2:1 mux를 통하여 적절한 값을 Red, Green, Blue에 전달하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명테이블이(가) 표시된 사진

자동 생성된 설명

테이블이(가) 표시된 사진

자동 생성된 설명

Red, Green, Blue는 다음과 같이 constraint파일에서 BASYS3의 VGA output으로 출력되는 값이다.

텍스트이(가) 표시된 사진

자동 생성된 설명