기초 디지털 실험 – 결과레포트 13~15 주차 project

**1. Barnsley fern 개요**

이번 프로젝트에서 사용한 Barnsley fern 알고리즘을 소개한다. Barnsley fern 알고리즘은 정해진 4 가지 확률적으로 실행되는 규칙에 의해 생성된 점이 규칙에 의해 위치가 정해지고 그 점들의 집합이 고사리의 모양을 만드는 알고리즘이다. 4 가지 규칙은 다음과 같다.

1) 1%확률로 실행되는 규칙: xn = 0, yn = 0.16 * y

오른쪽 그림의 빨간색 줄기를 담당한다.

2) 85%확률로 실행되는 규칙: xn = 0.85 * x + 0.04 * y, yn = -0.04 * x + 0.85 * y + 1.5

아래쪽의 노랑, 보라, 빨간색 점들의 집합을 오른쪽, 위쪽으로 조금씩 움직이게 한다.

오른쪽 그림의 파란 점들이 2) 규칙에 해당한다.

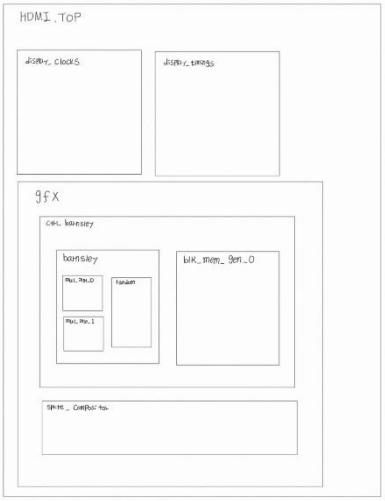
3) 7%확률로 실행되는 규칙: xn = 0.2 * x - 0.26 * y, yn = 0.23 * x + 0.22 * y + 1.5

오른쪽 그림의 노란 점들이 3) 규칙에 해당한다.

4) 7%확률로 실행되는 규칙: xn = -0.15 * x + 0.28 * y, yn = 0.26 * x + 0.24 * y

오른쪽 그림의 보라 점들이 4) 규칙에 해당한다.

**2. Barnsley fern 프로젝트 구현 개요**




이번 프로젝트의 목표는 Barnsley fern 을 LCD 화면 상에 구현 후 이 Barnsley fern 과 상호작용하는 구체를 LCD 화면에 생성하는 것이다.

이를 Verilog 를 이용하여 구현하려면 모듈을 TOP - DOWN 방식으로 HDMI 출력 ~ Barsley fern 알고리즘까지 오른쪽 그림과 같이 모듈을 연결하고 구성해야 한다.

이번 프로젝트에서의 구현 목표는 오른쪽 그림에서

barnsley.v, ctrl_barnsley, sprite_compositor, blk_mem_gen_0, gfx 5 개의 모듈을 나머지 모듈과 적절하게 상호작용할 수 있도록 하여 LCD 에 원하는 화면이 나오도록 만들어주는 것이다.

**3. barnsley.v 구현**

구현되어 있는 mul.v 모듈을 inst0, inst1 으로 instance 화 하여 2 개 이용하였다. 또한, random 모듈을 이용하여 0~$2^{31}$-1 의 난수들을 이용하여 Barnsley 알고리즘의 확률을 구현하였다.

1) 확률 구성

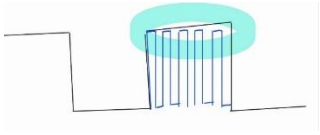
0~$2^{31}$-1 의 난수가 생성되므로 1%, 85%, 7%, 7%의 확률을 구성하기 위하여 0.01×($2^{31}$-1), 0.85×($2^{31}$-1), 0.07×($2^{31}$-1), 와 나머지로 if-else 문을 구성하여 확률에 따라 각각의 if 문이 실행되도록 하였다.

2) iteration

Iteration 은 간단하게 말하자면, 생성되는 x,y 좌표의 개수이다. 즉, for 문을 이용하여 최대 iteration 만큼 bernsley fern 알고리즘을 반복해주면 된다. Iteration 을 증가시킬 때는 Iteration 은 1000 으로 시작해 +3000 씩 3 번되어 최대 10000 의 iteration 을 달성해야 한다. 따라서, btn 의 입력을 감지할 때만 원래 iteration 값에 + 3000 을 하여 늘어난 +3000 만큼의 iteration 을 추가로 실행하도록 하였다. 코드 작성시 주의점은 btn = 1 일 때로 설정하면 아래 그림에서 FPGA 의 clock 과 모듈의 clock rate 가 다르기에 한번의 입력에 +3000 이 여러 번 실행되어 제대로 출력되지 않을 것을 주의하면 코드를 짰다.

3) x, y 좌표 연산

mul 모듈은 5bit integer 와 20bit fraction 의 부동소수점 규칙을 따르는 수의 곱셈기이다. 이 모듈은 두 수를 각각 mul_a, mul_b 에 받아서 곱셈결과를 mul_val 로 출력한다. 이 모듈을 활용하여 연산을 하기 위해서 현재 알고리즘에 있는 소수들을 부동소수점 형식으로 바꾸어 준다. 예를 들어, 0.16 = 25'b00000001010001111101011100 와 같이 표현된다.



먼저, 오른쪽 그림에서 검정색 clock 은 우리가 모듈에서 이용하는 input clock 신호를 표현한 것이다. 그리고, 파란색 clock 은 FPGA 자체의 clock 신호를 표현한 것이다. 즉, 우리가 input 으로 받는 clock 신호는 FPGA 의 clock 신호를 분기하여 얻어진 것이다.

mul 모듈의 코드를 보면, clock 신호를 input 으로 받지 않는다. 하지만, 연산은 clock cycle 을 무조건 소모한다. 이 연산은 hardware 자체의 clock cycle 을 소모하는 것이다. 우리는 4 주차 강의에서 FPGA 자체의 clock rate 가 125M Hz 정도인 것을 배웠다. 이 clock rate 를 소모하는 것이다.

또한, random.v 의 경우 clock 신호를 input 으로 받지만, output 이 나오기까지 4cycle 이 소모된다. 이점을 인지하여 xn = 0.85 * x + 0.04 * y, yn = -0.04 * x + 0.85 * y + 1.5 와 같은 연산의 cycle 소모를 4cycle 이 넘도록 설정하면, 연산이 끝났을 때 새로운 random 난수가 준비되어 있다. Random.v 의 첫 output 이 나오기 전 temp 값을 보면 이 값은 13 으로 설정되어 있으므로, 1% 확률의 연산이 첫 연산임도 고려하였다.

이제 각 x, y 좌표 연산을 한다. 여기서 주의해야 할 점은 모든 연산은 clock cycle 을 소모한다는 것이다. 즉, x 와 y 를 각각 mul_a, mul_b 에 할당하면 mul 모듈에서 어느 정도의 clock cycle 을 소모하고 연산결과가 나온다는 것이다. barnsley 모듈이 input 으로 받은 clock (위에서 검정색 )을 기준으로 움직이기에 연산결과는 다음 clock 에 나온다는 것이다.

본인의 경우는 연산의 clock 을 구분해주기 위하여 state machine 과 2 개의 mul_inst 를 사용하였다. 그리고 state 를 다음과 같이 구분하였다.

상태 1: xn 를 계산하기 위하여 mul_a_1, mul_b_1 에 x 값과 곱해지는 소수를 넣어주고, mul_a_2, mul_b_2 에 y 값과 곱해지는 소수를 넣어주는 state

상태 2: xn 에 관한 각각의 곱하기 연산결과를 더해주는 state

상태 3: yn 를 계산하기 위하여 mul_a_1, mul_b_1 에 x 값과 곱해지는 소수를 넣어주고, mul_a_2, mul_b_2 에 y 값과 곱해지는 소수를 넣어주는 state

상태 4: yn 에 관한 각각의 곱하기 연산결과를 더해주는 state

위와 같이 모듈을 구성하면 제대로 연산이 완료되어 xn, yn 좌표가 매 clock 마다 barnsley 모듈의 output 단을 통하여 나간다. 물론, 연산 중의 결과도 매 clock 마다 output 으로 출력된다.

**4. ctrl_barnsley 구현**

ctrl_barnsley 에서는 x,y 좌표가 연산 완료될 때마다 bram 의 적절한 address 에 write 를 해줘야 한다.

1) Bram 의 주소구성

BRAM 은 480000 개의 값을 저장할 수 있는 주소를 가지고 각 주소마다는 5bit 의 값을 write 하거나 read 를 할 수 있다. 800 X 600=48000 이므로 horizontal resolution 800, vertical resolution 600 으로 생각할 수 있다.

2) x, y 좌표 출력 범위

먼저, X 축과 Y 축의 최소 단위는 $2^{-6}$ 이다. 따라서, 계산된 25bit 의 x,y 좌표에서 $2^{-6}$ 을 나타내는 bit 까지 잘라 총 11bit 로 만든다. 앞의 5bit 는 integer, 뒤의 6bit 는 fraction 을 담당한다.

다음으로, 800, 600 을 11bit 이진수로 나타낸다. 이 이진수를 이제 11bit 부동소수점이라고 간주하고 다시 값을 계산한다.

800 = 11'b01100100000, 600 = 11'b01001011000 가 나오고, 이를 11bit 부동소수점으로 생각하고 계산하면, 각각 11'b01100100000 = 12.5 [decimal], 11'b01001011000 = 9.375 [decimal]이 나온다.

LCD 는 계산된 x,y 좌표를 빠짐없이 출력해야 한다. 우리가 사용하는 Barnsley fern 알고리즘에서 x, y 좌표의 범위를 계산해본다. 확률 변수이므로 어느 값에 수렴하는지 확인하면 된다. 기본적으로 원래 Barnsley fern 알고리즘은 대략 x = -2.182 to x = 2.6558 and y = 0 to y = 9.9983 의 범위를 가진다. 하지만, 우리가 사용하는 Barnsley fern 알고리즘은 수정되었다.

이를 계산해보면, 약 x = -2.0455 to x = 2.49 and y = 0 to y = 9.3734 의 범위를 가진다. 위에서 계산한 800 X 600 LCD 의 출력 범위는 x = 0 to x = 12.5 and y = 0 to y = 9.375 이었다. 따라서, 연산된 x 좌표만 적절히 평행이동 시켜주면 LCD 에 모든 좌표를 출력할 수 있다.

3) x,y 좌표 Bram address 에 할당 & write

x 좌표의 범위를 조정해주기 위하여 본인은 연산된 x 좌표가 LCD 의 horizontal_resolution 800 의 중간위치인 400 에 올 수 있도록 하였다. 따라서, 연산된 x 좌표에 +400 을 하였다. x 좌표는 11bit 부동소수점이고 400 은 decimal 이지만, 위에서 언급했듯이 애초에 800 을 부동소수점으로 생각하고 800 의 절반만큼 평행이동을 시키는 것을 판단한 것이기에 +400 을 부동소수점 형식으로 바꾸지 않고 그냥 +400 을 하여도 상관없다.

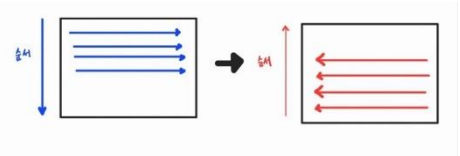
그리고 12 주차에서 hcnt + vcnt × (horizontal_resolution 크기)로 BRAM address 에 할당해주었다.

따라서, write 할 address 는 hcnt = x 좌표 + 400, vcnt = y 좌표 × 800 로 계산하고 address = hcnt+vcnt 를 하여 12 주차와 유사하게 구성해주었다.

Write 를 위해서는 BRAM 의 write port 를 enable 하게 해주어 write 를 진행한다. address 값이 계산된 곳에는 5'b11111 이 적히고, address 값이 계산되지 않은 곳에는 0 이 있다.

이제 생각해줘야 할 것은 목표 이미지가 LCD 의 중앙 기준으로 오른쪽으로 기울어져 있기에, 출력 시 x 축, y 축 반전하여 출력하는 것만 생각해주면 된다.

4) BRAM read



12 주차에서 사용하였던 방법과 유사하게 구성하였다. 하지만, 위에서 언급했듯이 address 를 0~480000 까지 차례로 읽으면 이미지가 LCD 에 x 축, y 축 반전되어 출력될 것이기 때문에 제대로 출력하기 위해 y 축 반전을 위해 48000~0 까지 읽고 x 축 반전을 위해 (-)기호를 붙여주어 아래 그림과 같이 read 하도록 하였다.

색 출력은 5'b11111 이 읽힌 address 에서는 pixel 이 초록색이 나오도록, 5'b00000 이 읽힌 address 에서는 pixel 이 검정색을 출력되도록 하였다. 구현방법은 12 주차와 유사하게 하였다.

**5. sprite_compositor 구현 & gfx 에 sprite_compositor 추가**

1) 구체 생성

구체 생성은 반지름이 RADIUS=80 인 원의 방정식을 사용하였다. Sprite 로 움직이어야 하므로, 원의 중심의 좌표를 (RADIUS + sprite_x, RADIUS + sprite_y)로 설정하여 sprite_x, sprite_y 의 값이 바뀌어 중심의 좌표가 변함에 따라, LCD 상에서 움직일 수 있게 하였다.

2) sprite 이미지의 범위 (sprite_hit_x, sprite_hit_y)

원의 지름의 160 픽셀이므로, sprite_x ~ sprite_x+160, sprite_y ~ sprite_y+160 와 같이 설정하였다.

3) sprite 의 이동 및 색 변환 및 초기화

FPGA 의 btn2 의 입력이 들어오면 sprite_x 와 sprite_y 의 좌표를 0 으로 초기화하여 LCD 의 왼쪽 상단으로 구체가 이동하도록 하였다.

이동은 12 주차에서 한 것과 같이 sprite_x_direction, sprite_y_direction 을 이용하여 방향을 설정하였고, 프레임의 끝에 (x 축: 0 & 800 – 160, y 축: 0 & 600-160 ) 도달하면 direction 신호를 0 에서 1 로 또는 1 에서 0 으로 바꿔주어 부딪힌 반대방향으로 움직이게 하였다.

색변경은 프레임 끝에 sprite 이미지가 도달하면 sprite_flip 신호를 토글해주는 형식으로 구현하였다. sprite_flip 신호가 0 이면 검정색이고 1 이면 빨간색이다. 하지만, 12 주차에서 예제로 주어진 코드를 그대로 사용하면 색 변경이 되지 않는다. 만약, sprite 이미지가 왼쪽 벽에 부딪혔다고 생각해보자. sprite_x 가 0 이 되어 x_direction 신호가 바뀌었지만, 코드에서는 non_blocking assignment 를 사용하므로 그와 동시에 sprite_x 가 -1 이 되므로 다음 clk 에서 sprite_x 의 값은 1 이 아니라 -1 이 된다. 따라서, 오류가 발생한다. 이러한 이유로 if-else 문을 사용하여 if, else if 문에는 프레임 끝에 도달하였을 때 구체가 바뀐 방향으로 1 픽셀만큼 움직이게 하는 코드를 작성하여 다음 clk 의 rising edge 에서 sprite_x 또는 sprite_y 의 값이 올바르게 연산되도록 하였고, else 문은 어떤 프레임 끝에도 도달하지 않았을 경우를 담당하므로 그 경우에는 direction 의 방향에 따라 sprite_x 와 sprite_y 가 direction 에 따라 +1 이 되거나 -1 이 되도록 하였다.

4) gfx 모듈에 sprite_compositor 인스턴스화

gfx 모듈에 sprite_compositor 인스턴스화하였고, sprite 이미지와 배경의 Barsley fern 이미지의 상호작용은 12 주차에서처럼 sprite_hit 신호를 이용하여 제어하였다.

5) sprite 이미지와 Barnsley fern 상호작용

두 개의 이미지가 겹치면 Barnsley fern 집합이 파란색을 나타내어야 한다. 여기서 Barnsley fern 집합은 원래 초록색을 나타내었다. 초록색을 나타낸다는 것은 red 를 나타내는 data = 0 이라는 의미이다. 따라서, if 문을 사용하여 12 주차에서 다루었던 sprite_hit 코드에서 sprite 이미지의 범위의 색상을 출력할 때 Barnsley fern 집합의 빨간 색상 data 가 0 인 좌표들에서는 gfx 모듈의 색상 output 으로 파란색을 출력하도록 하였다. 그리고 나머지 좌표에서는 sprite 이미지의 색상을 출력하도록 하였다. sprite 이미지 범위에 들지 않는 좌표에서는 원래 ctrl_barnsley 코드가 출력하는 색상을 출력하였다.

**6. Discussion**

이번 프로젝트를 통해 clock 에 대한 이해도와 hardware 설계, 즉. verilog 에서의 소수연산에 대한 이해도가 증진되었다. 더불어, verilog 에서의 testbench 를 사용한 디버깅 방법에 대하여 더 숙련되게 되었다. 또한, 강의안에서는 sprite 이미지를 Bitmap 형식으로만 구성하였는데, Bitmap 형식이 아닌 방식으로 sprite 이미지를 구현할 수 있음을 공부하게 되었다. 더불어, 2 의 보수로 음수를 계산하기에 음수일 경우 MSB 는 1 이어야 한다. 계산을 올바르게 하였어도, 배열의 크기를 데이터 크기보다 크게 하게 되면, MSB 는 0 이 되어 부호가 틀리게 된다. 따라서, 부호있는 수의 계산에서는 데이터 배열의 크기도 상당히 중요하다는 것을 이번 프로젝트로 몸소 느끼게 되었다.