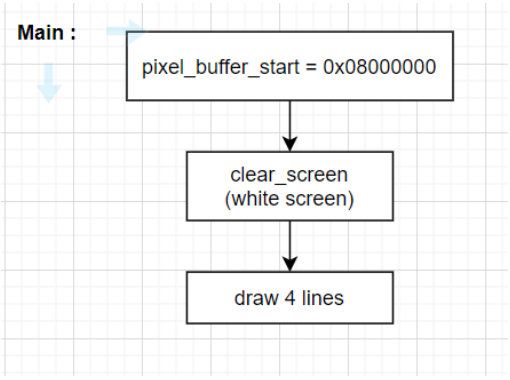
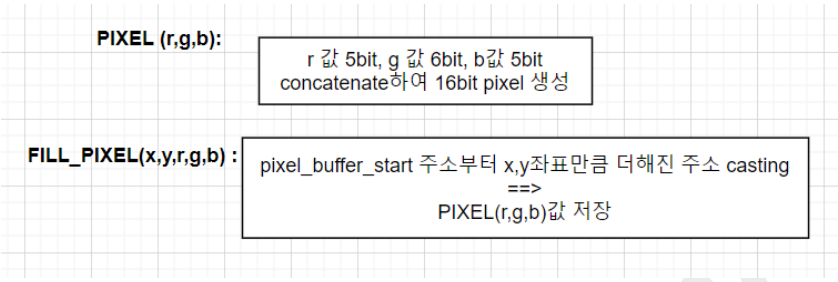
전자 HW 설계 – 실습 보고서

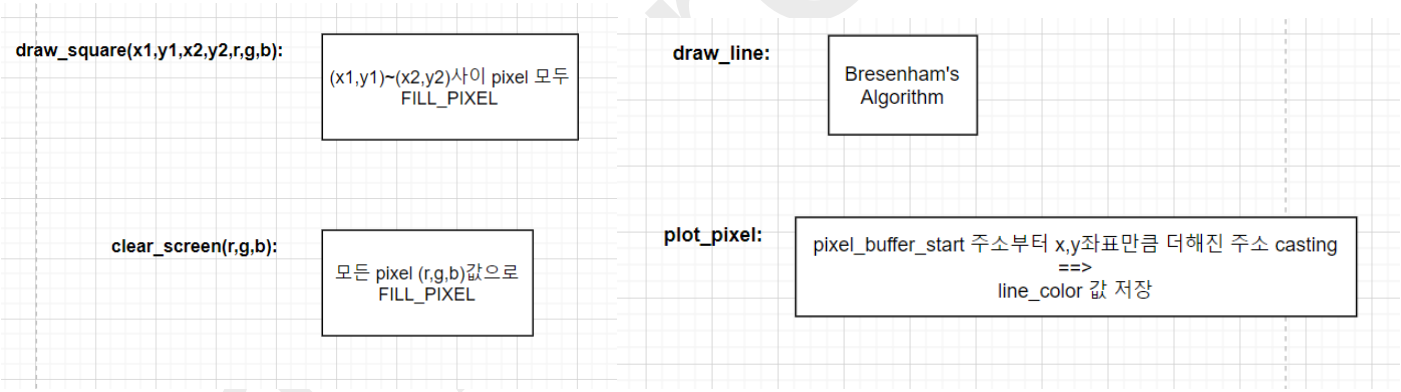
이름: 박관영 (2016124099)

✓ Part I

동작 원리







Bresenham's algorithm을 사용하여 직선 4개를 그리는 프로그램이다. 해당 프로그램은 video가 아닌 image를 출력하기 때문에 double buffering이나 vsync를 사용하지 않아도 된다. Pixel_buffer_start로 default buffer 주소를 저장한 뒤, 원하는 좌표와 색깔을 입력하여 line을 그릴 수 있게 된다.

< PIXEL(r,g,b) >

5bit의 r 값과 6 bit의 g 값, 5 bit의 b값을 concatenate하여 16bit의 pixel value를 만드는 매크로이다.

< FILL_PIXEL(x,y,r,g,b) >

전역변수로 정의된 pixel_buffer_start 주소부터 x, y의 좌표 값 만큼 간 주소에 PIXEL(r,g,b)의 값을 저장하는 매크로이다. 이때 해당 프로그램에서 x, y는 default width(9, 8)를 사용하게 되고 각 pixel은 16bit 값을 사용하게 된다. 즉 2byte의 크기를 가지게 되므로 y와 x를 concatenate할 때 2byte씩 이동해야 한다. 따라서 y<<10과 x<<1된 bit를 더한 주소에 pixel값을 저장한다.

<draw_square>

(x1, y1)~(x2, y2)사이의 pixel을 (r,g,b)값의 pixel로 채우는 함수이다. FILL_PIXEL 매크로를 (x2-x1)*(y2-y1)만큼 반복하여 구현할 수 있다.

<clear_screen>

설정된 SCREEN_WIDTH와 SCREEN_HEIGHT 만큼 draw_square를 수행하는 함수이다. 즉 모든 화면을 (r, g, b) pixel로 채우는 함수이다.

<draw_line>

Bresenham's algorithm을 사용하여 draw_line을 구현한다.

<plot_pixel>

FILL_PIXEL과 같은 방법으로 구현되었으나 r,g,b값을 따로 받지 않고 short int 형태로 받아 수행되는 함수이다.

구현 코드 설명

```
#include"address_map_nios2.h"
#include<math.h>

#define PIXEL(r, g, b) ₩
    (short int)((((r)&0x1f)<<11)|((((g)&0x3f)<<5)|(((b)&0x1f)))          // 16bit pixel 생성

#define FILL_PIXEL(x,y,r,g,b)₩
    *(short int *)(pixel_buffer_start + (((y)&0xff)<<10) + (((x)&0x1ff)<<1))=PIXEL(r,g,b)   // buffer+(x,y)에 pixel value 저장

#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 240               // default resolution

volatile int pixel_buffer_start;
volatile int* pixel_ctrl_ptr;           // memory mapped register 주소이므로 volatile 형태로 선언한다.

void swap(int*, int*);
void clear_screen(int, int, int);
void draw_square(int, int, int, int, int, int, int);
void draw_line(int, int, int, int, short int);
void plot_pixel(int, int, short int);                              // 사용될 함수들 선언.

int main(void) {
        pixel_ctrl_ptr = (int*)PIXEL_BUF_CTRL_BASE;       // pixel buffer controller의 buffer reg 주소 int* 로 casting
        pixel_buffer_start = *pixel_ctrl_ptr;             // 실제로 pixel이 저장될 buffer 주소

        clear_screen(0xFF, 0xFF, 0xFF);             // 화면 흰색으로 initializing
```

```
	draw_line(0, 0, 150, 150, 0x001F);
	draw_line(150, 150, 319, 0, 0x07E0);
	draw_line(0, 239, 319, 239, 0xF800);
	draw_line(319, 0, 0, 239, 0xF81F);		// 그리고자 하는 직선의 색깔과 좌표 전달하여 4개의 직선 그린다.
}

void swap(int* x, int* y) {		// x, y 값 교환. Pointer를 통해 구현해야 호출한 함수에서도 값이 바뀐다.
	int temp;
	temp = *x;
	*x = *y;
	*y = temp;
}

void clear_screen(int r, int g, int b) {	// screen 전체를 (r,g,b)로 채우는 함수.
	draw_square(0, 0, SCREEN_WIDTH - 1, SCREEN_HEIGHT - 1, r, g, b);
}

void draw_square(int x1, int y1, int x2, int y2, int r, int g, int b) {	// (x1, y1)~(x2,y2)까지 (r,g,b)로 채우는 함수
	int x, y;
	for (x = x1; x <= x2; x++) {
		for (y = y1; y <= y2; y++) {
			FILL_PIXEL(x, y, r, g, b);
		}
	}
}

void draw_line(int x0, int y0, int x1, int y1, short int line_color) {	// Bresenham's algorithm
	int is_steep;
	(abs(y1 - y0) > abs(x1 - x0)) ? 1 : 0;
	is_steep = (abs(y1 - y0) > abs(x1 - x0));
	if (is_steep) {
		swap(&x0, &y0);
		swap(&x1, &y1);
	}
	if (x0 > x1) {
		swap(&x0, &x1);
		swap(&y0, &y1);
	}
	int deltax = x1 - x0;
	int deltay = abs(y1 - y0);
	int error = -(deltax / 2);
	int y = y0;
	int y_step;
	if (y0 < y1)   y_step = 1;
	else   y_step = -1;
```

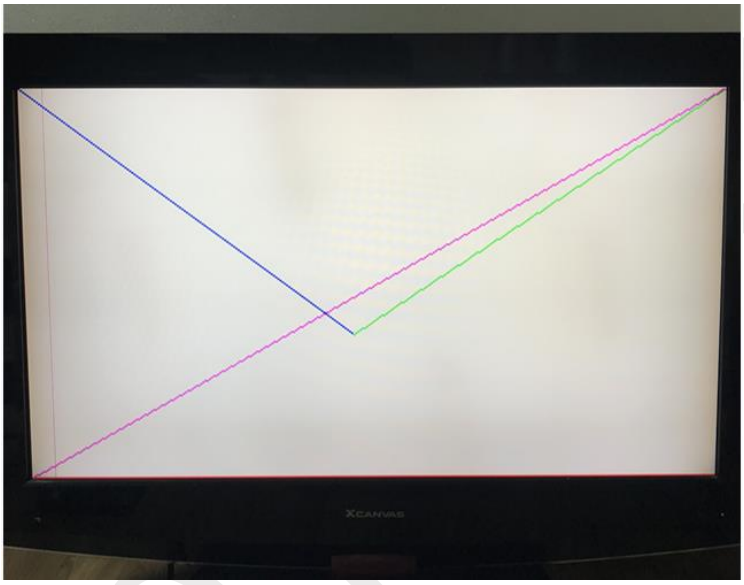
```
		for (int x = x0; x <= x1; x++) {
			if (is_steep) plot_pixel(y, x, line_color);
			else plot_pixel(x, y, line_color);

			error += deltay;

			if (error >= 0) {
				y = y + y_step;
				error = error - deltax;
			}
		}
}


void plot_pixel(int x, int y, short int line_color) {		// line_color의 pixel 값을 x,y에 저장하는 함수
		*(short int*)(pixel_buffer_start + (y << 10) + (x << 1)) = line_color;
}
```

결과 및 토의

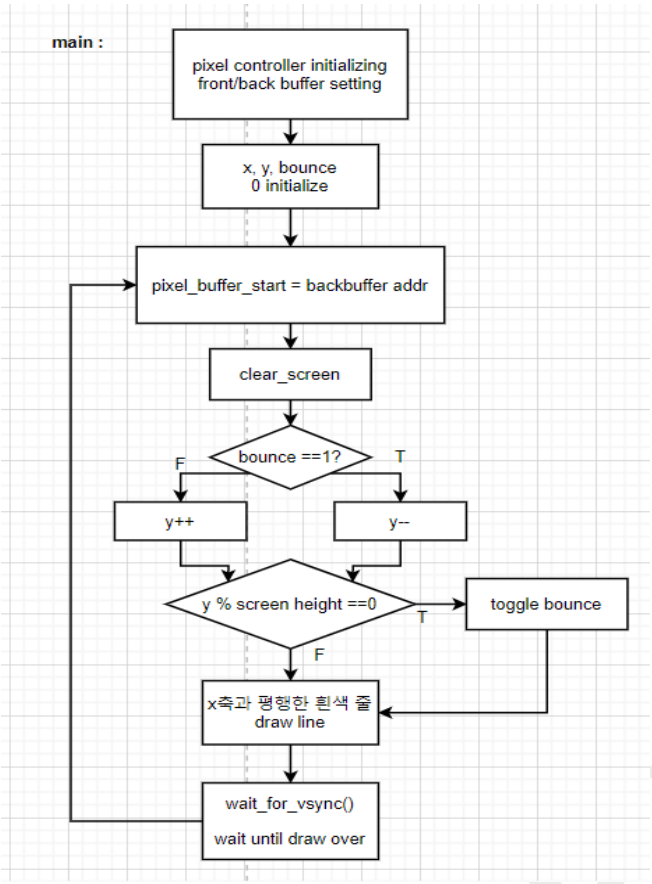


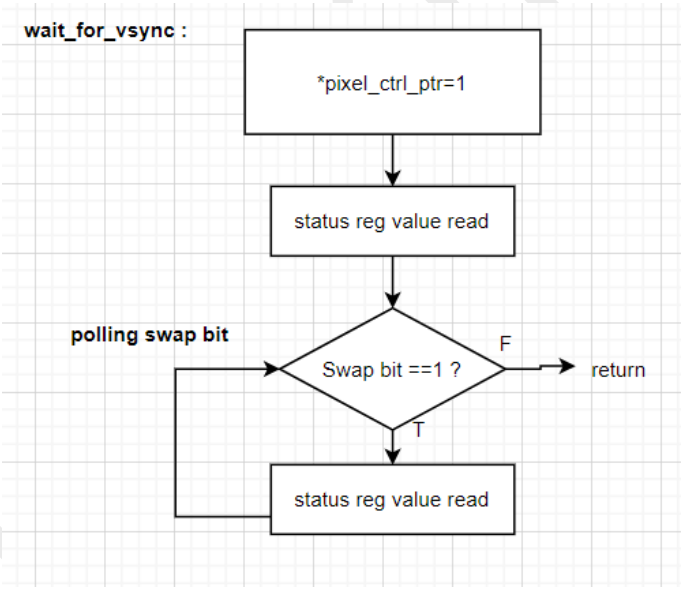
De1-soc board의 해상도와 출력 모니터 해상도 차이에 의해 발생하는 약간의 오차 이외에 문제없이 잘 동작하는 것을 확인할 수 있었습니다.

전자 HW 설계 – 실습 보고서

✓ Part II

동작 원리





다음 프로그램은 x축에 평행한 직선이 y축에 수직한 방향으로 움직이면서 y축의 끝, 즉 y=0 혹은 y=SCREEN_HEIGHT 가 되면 bounce를 하여 y축으로 움직이는 방향이 바뀌는 프로그램이다. Part1과 다르게 video를 출력하는 프로그램이기 때문에 double buffering 과 vsync를 사용한다. Pixel buffer controller를 통해 화면에 pixel을 display하는 기본적인 함수는 part1과 같은 함수를 사용하였다.

<wait_for_vsync>

Video를 display할 때 tearing artifact가 없게 하는 기법이다. Processor가 frame을 만드는 속도와 board가 vga를 통해 frame을 draw하는 속도를 일치시켜 한 화면에 두 종류의 frame 겹쳐서 display되는 것을 방지할 수 있게 된다. Vsync event를 detect하는 방법으로는 status register의 swap bit를 polling하는 방식을 사용하였다. Pixel controller의 buffer reg에 1을 써넣어서 vsync event triggering 을 기다릴 수 있게 하는 control mechanism 을 사용하였고, swap bit가 0이 되는 순간 vsync event가 발생했다고 볼 수 있으므로 이 때 함수를 return 하여 vsync event를 detect 할 수 있다.

<main>

Global array를 통해 프로그램에서 사용할 front buffer와 back buffer 공간을 선언한다. Back buffer reg에 front buffer addr을 입력한 뒤 vsync event를 통해 swap이 일어나기를 기다린다. 그 후 back buffer reg에 back buffer addr을 입력함으로써 pixel controller를 초기화 한다. 이 후 pixel_buffer_start를 back buffer addr로 설정하여 processor에서 만드는 frame을 back buffer를 통해 만들게 된다. 반복문을 통해서 draw_line을 수행하여 선이 움직이는 것을 표현할 수 있다. 이 때

bounce 값을 flag로 하여 y축을 기준으로 위로 움직일지, 아래로 움직일지를 선택하게 된다. 만약 y좌표의 값이 0이거나 screen_height가 된다면 bounce 값을 toggle하여 움직이는 방향을 바꾸게 됨으로써 bouncing line을 구현할 수 있었다. X, y 좌표를 설정한 뒤 화면에 draw될 때까지 기다린 후, 즉 vsync event가 trigger되는 것을 기다린 후 다시 반복문을 수행하게 된다.

구현 코드 설명

```
#include"address_map_nios2.h"
#include<math.h>

#define PIXEL(r, g, b) ₩
    (short int)((((r)&0x1f)<<11)|((((g)&0x3f)<<5)|(((b)&0x1f)))

#define FILL_PIXEL(x,y,r,g,b)₩
    *(short int *)(pixel_buffer_start + (((y)&0xff)<<10) + (((x)&0x1ff)<<1))=PIXEL(r,g,b)

#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 240

volatile int pixel_buffer_start;
volatile int* pixel_ctrl_ptr;

short int front_buffer[512 * 256];
short int back_buffer[512 * 256];                    // front/back_buffer 공간 선언

void swap(int*, int*);
void clear_screen(int, int, int);
void draw_square(int, int, int, int, int, int, int);
void draw_line(int, int, int, int, short int);
void plot_pixel(int, int, short int);
void wait_for_vsync(void);

void wait_for_vsync() {
        register int status;
        *pixel_ctrl_ptr = 1;        // wait for vsync event triggering control mechanism

        status = *(pixel_ctrl_ptr + 3);
        while ((status & 0x01) != 0)                // status reg swap bit polling
                status = *(pixel_ctrl_ptr + 3);
}



void clear_screen(int r, int g, int b) {
        draw_square(0, 0, SCREEN_WIDTH - 1, SCREEN_HEIGHT - 1, r, g, b);
```

```
}

void draw_square(int x1, int y1, int x2, int y2, int r, int g, int b) {
	int x, y;
	for (x = x1; x <= x2; x++) {
		for (y = y1; y <= y2; y++) {
			FILL_PIXEL(x, y, r, g, b);
		}
	}
}

void swap(int* x, int* y) {
	int temp;
	temp = *x;
	*x = *y;
	*y = temp;

}

void plot_pixel(int x, int y, short int line_color) {
	*(short int*)(pixel_buffer_start + (y << 10) + (x << 1)) = line_color;
}

void draw_line(int x0, int y0, int x1, int y1, short int line_color) {
	int is_steep;
	(abs(y1 - y0) > abs(x1 - x0)) ? 1 : 0;
	is_steep = (abs(y1 - y0) > abs(x1 - x0));
	if (is_steep) {
		swap(&x0, &y0);
		swap(&x1, &y1);
	}
	if (x0 > x1) {
		swap(&x0, &x1);
		swap(&y0, &y1);
	}
	int deltax = x1 - x0;
	int deltay = abs(y1 - y0);
	int error = -(deltax / 2);
	int y = y0;
	int y_step;
	if (y0 < y1)   y_step = 1;
	else   y_step = -1;
	for (int x = x0; x <= x1; x++) {
		if (is_steep) plot_pixel(y, x, line_color);
		else plot_pixel(x, y, line_color);
```

```
                error += deltay;

                if (error >= 0) {
                        y = y + y_step;
                        error = error - deltax;
                }
        }
}


int main(void) {
        pixel_ctrl_ptr = (int*)PIXEL_BUF_CTRL_BASE;
        pixel_buffer_start = *pixel_ctrl_ptr;

        *(pixel_ctrl_ptr + 1) = front_buffer;
        wait_for_vsync();                       // swap buffer/ back buffer reg value ➔ front buffer addr store at buffer reg

        pixel_buffer_start = *pixel_ctrl_ptr;
        clear_screen(0x00, 0x00, 0x00);         // clear screen with front buffer
        *(pixel_ctrl_ptr + 1) = back_buffer;    // back buffer addr store

        int x = 0, y = 0, bounce = 0;           // x, y coordinate, bounce flag initialize
        while (1) {
                pixel_buffer_start = *(pixel_ctrl_ptr + 1);     // back buffer addr를 start addr로 설정
                clear_screen(0x00, 0x00, 0x00);         // clear screen to black screen
                if (bounce) y--;            // bounce=1 ➔ 아래로 이동
                else y++;                   // bounce=0 ➔ 위로 이동
                if (y % SCREEN_HEIGHT == 0) bounce ^= 1;        // toggle bounce

                draw_line(x + 100, y, x + 150, y, 0xffff);              // x축과 평행한 흰색 줄 draw_line
                wait_for_vsync();                       // wait for drawing over
        }
}
```

결과 및 토의

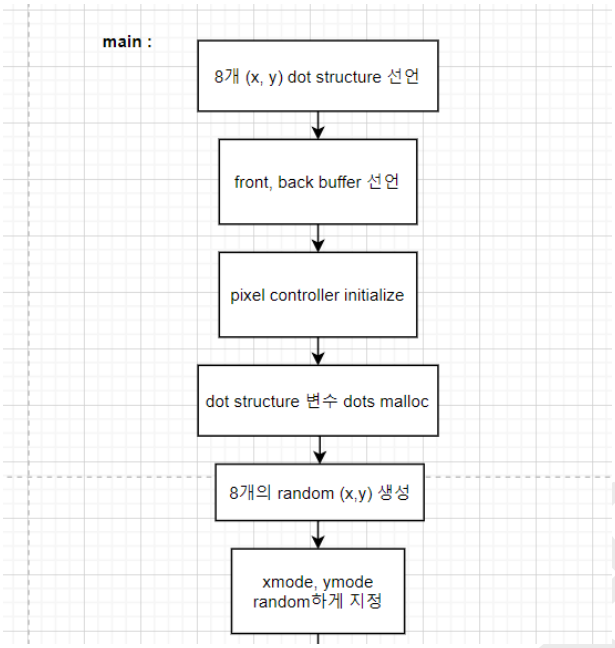
https://youtu.be/lgOFfuMayWg

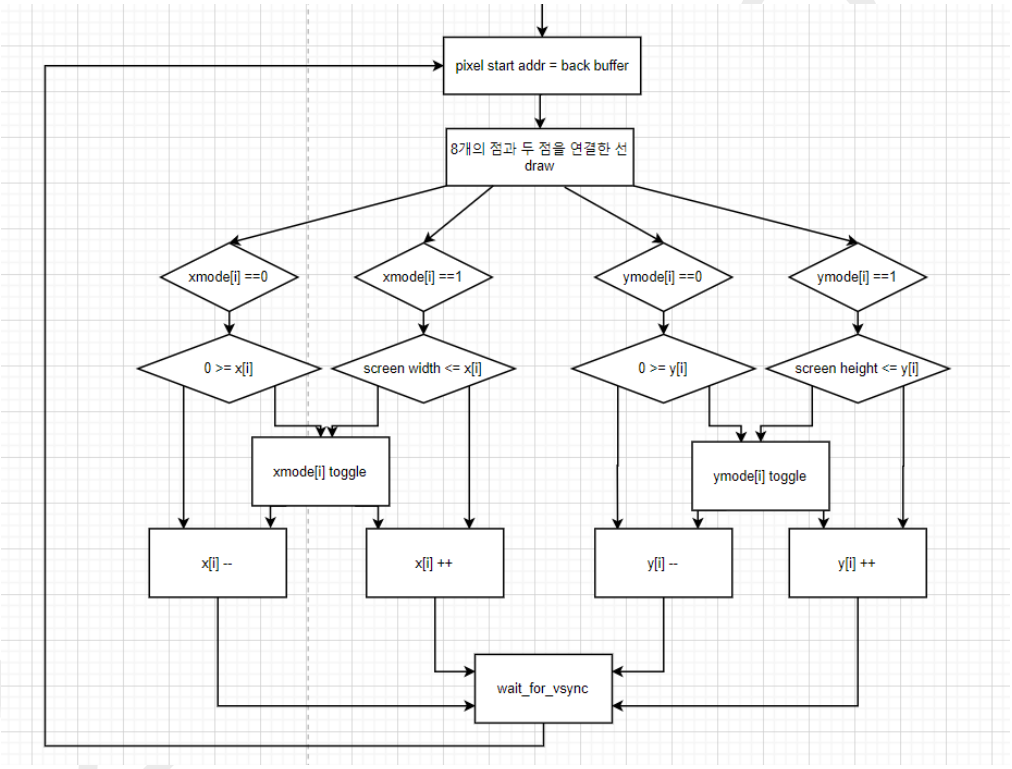
>>> 실행 결과 영상 첨부합니다.

Tearing artifact와 flickering artifact가 발생하지 않는 것을 확인할 수 있었습니다.

✓ Part III

동작 원리





8개의 pixel과 각 pixel의 이웃한 점을 연결하여 만든 도형이 대각선 방향으로 이동하며 화면 밖에 나가지 않도록 bouncing하는 animation을 display하는 프로그램이다. Part1,2의 기본함수들을 동일하게 사용하였고, 8개의 pixel의 좌표를 구조체를 통해 선언하였다. Part2와 마찬가지로 animation을 display하므로 double buffering과 vsync를 사용하였다. 또한 pixel이 bounce 하는 과정도 8개의 pixel에 대해서 수행하기 위해 xmode[i]와 ymode[i]라는 배열의 원소를 flag로 하여 각 pixel이 bounce하는 것을 detect하였다.

<main>

Part2와 마찬가지로 pixel buffer controller를 initialize하여 double buffering을 수행하게 하고, 사용할 구조체를 malloc한다. random하게 구조체의 (x,y)좌표를 설정한 뒤, xmode[i]와 ymode[i]도 임의의 0,1 값으로 초기화 한다. 이후 while 무한반복문을 통해 back buffer에 8개의 pixel과 각 이웃한 pixel을 연결하는 loop를 수행한다. 각 pixel은 모드에 따라 x, y좌표가 더해지거나 빼지면서 대각선으로 이동하게 된다. 또한 화면의 끝에 닫게 되면 각 xmode[i], ymode[i]를 toggle한다. 다음과 같이 x, y좌표와 xmode, ymode를 설정한 뒤 화면에 draw 되기까지, 즉 vsync event가 trigger될 때까지 기다린 후 다시 반복문을 수행하게 된다.

구현 코드 설명

```
#include"address_map_nios2.h"
#include<math.h>
#include<stdlib.h>

#define PIXEL(r, g, b) ₩
    (short int)((((r)&0x1f)<<11)|((((g)&0x3f)<<5)|(((b)&0x1f)))

#define FILL_PIXEL(x,y,r,g,b)₩
    *(short int *)(pixel_buffer_start + (((y)&0xff)<<10) + (((x)&0x1ff)<<1))=PIXEL(r,g,b)

#define SCREEN_WIDTH 320
#define SCREEN_HEIGHT 240

typedef struct coordinates {
        int x[8];
        int y[8];
}dot;                   // 8개의 pixel의 x, y좌표를 저장하고 있는 두 배열을 포함하는 구조체 선언

volatile int pixel_buffer_start;
volatile int* pixel_ctrl_ptr;

short int front_buffer[512 * 256];
short int back_buffer[512 * 256];           // double buffering을 위한 front/back buffer 선언

void swap(int*, int*);
void clear_screen(int, int, int);
void draw_square(int, int, int, int, int, int, int);
void draw_line(int, int, int, int, short int);
void plot_pixel(int, int, short int);
void wait_for_vsync(void);

void wait_for_vsync() {                     // vsync event detect
        register int status;
```

```
	*pixel_ctrl_ptr = 1;

	status = *(pixel_ctrl_ptr + 3);
	while ((status & 0x01) != 0)
		status = *(pixel_ctrl_ptr + 3);
}


void clear_screen(int r, int g, int b) {
	draw_square(0, 0, SCREEN_WIDTH - 1, SCREEN_HEIGHT - 1, r, g, b);
}

void draw_square(int x1, int y1, int x2, int y2, int r, int g, int b) {
	int x, y;
	for (x = x1; x <= x2; x++) {
		for (y = y1; y <= y2; y++) {
			FILL_PIXEL(x, y, r, g, b);
		}
	}
}

void swap(int* x, int* y) {
	int temp;
	temp = *x;
	*x = *y;
	*y = temp;

}

void plot_pixel(int x, int y, short int line_color) {
	*(short int*)(pixel_buffer_start + (y << 10) + (x << 1)) = line_color;
}

void draw_line(int x0, int y0, int x1, int y1, short int line_color) {
	int is_steep;
	(abs(y1 - y0) > abs(x1 - x0)) ? 1 : 0;
	is_steep = (abs(y1 - y0) > abs(x1 - x0));
	if (is_steep) {
		swap(&x0, &y0);
		swap(&x1, &y1);
	}
	if (x0 > x1) {
		swap(&x0, &x1);
		swap(&y0, &y1);
	}
```

```
		int deltax = x1 - x0;
		int deltay = abs(y1 - y0);
		int error = -(deltax / 2);
		int y = y0;
		int y_step;
		if (y0 < y1)   y_step = 1;
		else   y_step = -1;
		for (int x = x0; x <= x1; x++) {
			if (is_steep) plot_pixel(y, x, line_color);
			else plot_pixel(x, y, line_color);

			error += deltay;

			if (error >= 0) {
				y = y + y_step;
				error = error - deltax;
			}
		}
}



int main(void) {
		pixel_ctrl_ptr = (int*)PIXEL_BUF_CTRL_BASE;
		pixel_buffer_start = *pixel_ctrl_ptr;

		*(pixel_ctrl_ptr + 1) = front_buffer;
		wait_for_vsync();

		pixel_buffer_start = *pixel_ctrl_ptr;
		clear_screen(0x00, 0x00, 0x00);
		*(pixel_ctrl_ptr + 1) = back_buffer;
		dot* dots = (dot*)malloc(sizeof(dot));            // 구조체 변수 선언
		int i;
		for (i = 0; i < 8; i++) {                         // random한 8개의 점 구조체에 저장
			dots->x[i] = (rand() % SCREEN_WIDTH);
			dots->y[i] = (rand() % SCREEN_HEIGHT);
		}
		int xmode[8] = { 1,1,1,0,0,0,0,1 };
		int ymode[8] = { 0,1,0.0,0,1,1,1 };                       // xmode, ymode 임의로 지정

		while (1) {
			pixel_buffer_start = *(pixel_ctrl_ptr + 1);     // back buffer에 frame make
			clear_screen(0x00, 0x00, 0x00);
```

```
                for (i = 0; i < 8; i++) {          // 8개의 pixel plot하고, 이웃한 line draw, 모두 흰색으로 display
                        plot_pixel(dots->x[i], dots->y[i], 0xffff);
                        plot_pixel(dots->x[(i + 1) % 8], dots->y[(i + 1) % 8], 0xffff);
                        draw_line(dots->x[i], dots->y[i], dots->x[(i + 1) % 8], dots->y[(i + 1) % 8], 0xffff);
                }
                for (int i = 0; i < 8; i++) {.
                        if (xmode[i]) {          // x[i], y[i]가 xmode[i], ymode[i]에 따라 더해질지 빼질지 결정됨
                                if (dots->x[i] >= SCREEN_WIDTH) {xmode[i]^= 1;}  // 화면 끝에 닿으면 mode toggle
                                else dots->x[i]+=1;
                        }
                        else {
                                if (dots->x[i] <= 0) {xmode[i] ^= 1; }
                                else dots->x[i]-=1;
                        }

                        if (ymode[i]) {
                                if (dots->y[i] >= SCREEN_HEIGHT) {ymode[i]^= 1;}
                                else dots->y[i]+=1;
                        }

                        else {
                                if (dots->y[i] <= 0) {ymode[i] ^= 1;}
                                else dots->y[i]-=1;
                        }
                }

                wait_for_vsync();            // frame draw될 때까지 wait
        }
}
```

결과 및 토의

https://youtu.be/6fEn7V9avAw

>>> 실험 결과 영상 첨부합니다.

De1-soc board의 해상도와 출력 모니터 해상도 차이에 의해 발생하는 약간의 오차 이외에 문제없이 잘 동작하는 것을 확인할 수 있었습니다.