

REPORT

전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: <http://www.ieee.org>)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

1. **공중의 안전, 건강 복리에 대한 책임:** 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.
2. **지위 남용 배제:** 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)
3. **정직성:** 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.
4. **뇌물 수수 금지:** 어떠한 형태의 뇌물도 거절한다.
5. **기술의 영향력 이해:** 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.
6. **자기계발 및 책무성:** 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.
7. **엔지니어로서의 자세:** 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.
8. **차별 안하기:** 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공정하게 대한다.
9. **도덕성:** 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.
10. **동료애:** 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

<설계 계획서>

학 부: 전자공학과

제출일: 2023.12.14

과목명: 논리회로실험

교수명: 이 교 범 교수님

분 반: C087-6

학 번: 202020962, 202020963, 202020889

성 명: 정준희, 안재형, 한현빈

설계 계획서

0. 목차

1. 설계 목표.....	2
2. 접근 전략.....	3
2.1 요구 사항에 대한 분석 및 접근 전략.....	3
2.2 설계 과정에서 예상되는 문제점 및 접근 전략.....	4
2.3 예제 프로젝트 분석과 접근 전략.....	4
3. 개발 환경 정의 및 이론적 배경.....	10
4. 프로젝트 동작 블록 다이어그램.....	18
6. 조원 역할.....	19

1. 설계 목표

1. FPGA Kit에서 7-segment 모듈과 Keypad 모듈을 이용하여 사칙연산 계산기를 만든다.

○ 하드웨어 사양

- FPGA : ALTERA Cyclone IV EP4CE22F17C6 FPGA “DE0-NANO“
- 4x4 tact switch :
- 7-segment : 최대 6자리까지 표현할 수 있으며 common cathode 방식의 FND(Flexible Numeric Display)이다. Dynamic 구동 방식으로 6자리 FND를 동작시킨다. FPGA 보드의 GPIO pin 14개를 이용하여 제어한다. 다음은 예제 코드에서 segment를 동작시키기 위해 이용한 출력이다.

fnd_selector: [5:0]fnd_s 출력을 6개의 segment 중 하나를 선택하여 dynamic 구동
fnd_digit : [7:0]fnd_d 출력을 이용해 segment의 anode에 전압 인가

○ 개발 환경

- IDE : intel quartus prime lite edition design software
- HDL: Verilog
- *simulation은 AMD xilinx Vivado design software 이용.

2. 각 기능을 구현하는 단위 module별로 나누어 설계

계층적 설계를 통해 프로젝트를 기능별로 나누고 간단히 설계한다. divide and conquer 전략을 이용한다. 또한 각 기능을 module로 나누어 설계할 시 프로젝트의 디버깅이 수월해지고 팀원간 협업이 쉬워진다는 장점이 있다.

2. 접근 전략

2.1 요구사항에 대한 분석 및 접근 전략

2.1.1 요구사항 분석

기본 요구사항

- 6자리 숫자를 사칙 연산할 수 있는 계산기 설계
- 입력된 숫자는 7-segment 모듈에 표시

필수 구현사항

- 숫자와 연산기호를 번갈아 가며 입력 후 '='를 입력하여 연산 마무리
(숫자: 0~999999, 연산기호: +, -, *, /(몫), %(나머지))

ex)

$$113 + 24 - 36875 + 421124 - 509 = 383877$$

$$1 + 2 = 3$$

$$1 - 2 = -1$$

- 연산자 우선순위 무시 ($1 + 2 * 3 - 2 = 7$)
- 초기화 버튼 구현
- 추가 기능 구현 - 2가지 (자유롭게)

예외처리 구현

- 0으로 나누었을 때 Error 출력
- Overflow(999999 초과) 시 Error 출력
- 최종 결과물은 Verilog로 구현

2.1.2 문제 접근 전략

1. 입력된 숫자는 7-segment에 표시

버튼이 눌렸을 때 버튼 눌림을 감지하는 변수를 이용하여 눌린 숫자에 대해 1의 자리부터 ~ 100000의 자리까지 저장하고 각 자리수에 해당하는 숫자를 segment로 표시한다.

2. 숫자와 연산기호를 번갈아 가며 입력 후 '='를 입력하여 연산 마무리

피 연산자와 연산자를 저장하기 위한 register를 각각 정의하여 저장하고 등호가 입력되면 입력된 숫자와 연산기호들이 적절하게 입력되었는지 판단한 후 적절하면 계산값을, 적절하지 않으면 Error가 7-segment에 출력되도록 설계한다.

3. 연산자 우선순위 무시

연산자 우선순위를 무시하므로 변수 하나를 선언하여 숫자, 연산기호, 숫자가 나올 시에 그 계산값을 변수에 저장하고 다음 연산기호와 숫자가 입력될 때 변수에 지정된 값을 피연산자로 하여 계산하도록 설계한다.

4. 초기화 버튼 구현

초기화 버튼을 구현하여 초기화 버튼이 눌리는 것을 감지하면 7-segment의 출력 뿐만 아니라 모든 변수에 저장되어 있는 변수들을 초기화 하도록 설계한다. 이때 초기화 입력은 비동기 입력으로 초기화를 눌렀을 때 항상 모든 값이 초기화 되도록 한다.

5. 0으로 나누었을 때 Error 출력

0으로 나누는 것은 연산기호중 / 연산기호가 입력되었을 때는 그 이후의 바로 입력되는 숫자가 0인 경우 Error가 출력되도록 구현한다.

6. Overflow 발생 시 Error 출력

계산한 결과에 대해 100000으로 나누었을 때 몫이 10이상이면 6자리를 초과하는 것으로 판단할 수 있으므로 이 때

Error를 7-segment로 출력한다.

2.2 설계 과정에서 예상되는 문제점 및 접근 전략

- overflow detect
연산 후 overflow detect에 대한 문제가 존재한다. overflow에 대해 확인하기 위해서는 우선 피연산자의 크기에 대해 논의해야 한다.
피 연산자에 입력할 수 있는 정수의 범위는 -99,999~999,999(10)이다. 이때 음수의 범위가 양수의 범위보다 작은 이유는 음수 표현에 있어 6자리 FND를 이용해서는 5자리 음수까지 밖에 표현할 수 없기 때문이다. 이를 2진수로 나타내면 다음과 같다. 레지스터의 크기는 byte 단위로 결정하였다.

	10진수	2진수		register size
양수 최대 표현 범위	999,999	0000 1111 0100 0010 0011 1111	→	24 bit
음수 최대 표현 범위	-99,999	1111 1110 0111 1001 0110 0001		24 bit

표 5 <필요 register 크기> 최대 표현가능 10진수와 대응되는 바이너리 값.

실제 피연산자를 저장하는 register는 기본 크기 4byte인 reg signed [31:0] 벡터로 정의.

다음은 연산 결과를 overflow 없이 저장하기 위해 필요한 register의 크기이다.

	피연산자1	피연산자2		연산 결과
adder	24 bit	24 bit		25 bit
multiplier	24 bit	24 bit	→	48 bit
divider	24 bit	24 bit		48 bit
modulo	24 bit	24 bit		24 bit

표 6 <연산자 별 필요 register> signed 연산에서 overflow 없이 연산 결과를 저장하기 위해 필요한 register 크기

따라서 연산 결과는 최대 48 bit를 넘어갈 수 없다. 각 연산기의 계산 결과를 48 bit로 출력한다. 이렇게 처리할 경우 연산에서 overflow가 발생하지 않는다. 이후 error detect module에서 최대 표현가능 범위를 초과하는 경우 Error를 출력한다.

*실제 연산 결과를 저장하는 register는 8byte크기 벡터 reg [63:0] resulte로 정의한다.

- move a segment when entering two numbers
 1. 매 자릿수마다 상태를 설정해준다.
 2. sw_s1, sw_s2, . . . , sw_s6을 설정해주고 현재 몇 자릿수인지 상태로 저장한다.
 3. sw_s1에서 숫자를 입력받으면 sw_s2로 설정한다.
 4. 수가 저장된 num값을 그에 맞게 바꾸어준다.
- divide by zero condition handling
나눗셈기로 error를 찾을 수 없으므로, 소프트웨어 인터럽트를 발생시켜 error 처리를 해야한다. 따라서 나눗셈기로 신호를 출력하기 전에 divide by zero를 확인하여 error 메시지를 출력해야 한다.

2.3 예제 프로젝트 분석과 접근 전략

예제 프로젝트에서는 board에서 기본 제공하는 clock_50m을 이용한다. 50MHz는 key pad 회로나 segment 회로에 이용되기에 너무 주파수가 크므로, 각각 $\frac{1}{2^{21}}$, $\frac{1}{2^{17}}$ 로 분주하여 23.84Hz, 381.47Hz로 분주하여 이용한다. 높은 주파수가 필요 없을뿐더러 circuit을 통해 흐르는 신호의 주파수가 커질수록 전력또한 많이 소모됨을 생각한다.

크게 세가지 파트로 나눌 수 있다.

- key pad encoder : 키패드로부터 입력받아 사용자 정의 code로 인코딩한다.
- password checker : 입력받은 데이터를 검사하여 password가 일치하는지 확인한다. 동시에 입력받은 문자를 위치에 맞춰 fnd decoder로 보낸다.
- fnd decoder : segment에 출력할 코드를 decode 하여 GPIO로 출력한다.

이때 key pad encoder는 password checker 내에서 함께 동작하며, sw_clk 상승애지마다 실행된다.

segment는 fnd decoder에 의해 fnd_clk 상승 애지마다 한자리씩 출력된다.

다음은 예제 코드에 대해 line by line으로 분석하고 주석을 단 코드이다. 보고서에 첨부되어있는 코드 중 불필요하게 반복되는 부분은 생략하고 주석으로 그 기능을 서술해 놓았다.

```
module switch_segment(clock_50m, pb, fnd_s, fnd_d);

    // input output.
    input clock_50m;           //보드 제공 clk
    input[15:0] pb;           //16bit key pad 입력
    output reg[3:0] fnd_s;     //segment select negative decoder 필요
    output reg[7:0] fnd_d;     //segment anode positive decoder

    // clock.
    reg[15:0] npb;
    reg[31:0] init_counter;    //50MHz counter
    reg sw_clk;                //2^(-21) 분주
    reg fnd_clk;               //2^(-17) 분주
    reg[1:0] fnd_cnt;          //segment selector
                                //4자리를 번갈아 출력하기 위해 fnd_clk counter

    // 7-segment.
    reg[4:0] set_no1;          //segment display char1
    reg[4:0] set_no2;          //char2
    reg[4:0] set_no3;          //char3
    reg[4:0] set_no4;          //char4
    reg[6:0] seg_1000;         //segment cathode corresponding to char1
    reg[6:0] seg_100;          //corresponding to char2
    reg[6:0] seg_10;           //corresponding to char3
    reg[6:0] seg_1;            //corresponding to char4

    // pass check.
    reg[4:0] save_no1;         //password parameter
    reg[4:0] save_no2;
    reg[4:0] save_no3;
    reg[4:0] save_no4;
    regpass_ok;                //password correct

    // switch(keypad) control.
    reg[15:0] pb_1st;          //key pad 입력 현재 상태
    reg[15:0] pb_2nd;          //key pad 입력 이전 상태
```

```

regsw_toggle;          //입력 모드

// sw_status.
reg[2:0] sw_status;
parametersw_idle =0;    //대기 상태
parametersw_start =1;   //idle 상태에서 'h1000 입력되었을 때 (enter)
parametersw_s1 =2;      //첫번째 문자 출력 완료
parametersw_s2 =3;      //두번째 문자 출력 완료
parametersw_s3 =4;      //세번째 문자 출력 완료
parametersw_s4 =5;      //네번째 문자 출력 완료
// parameter sw_save = 6;
// parameter sw_cancel = 7;

// initial.
initial begin
    sw_status <=sw_idle;  //sw_status 초기 설정 : idle 상태
    sw_toggle <=0;
    npb <='h0000;        //imverted key in
    pb_1st <='h0000;     //switch control???
    pb_2nd <='h0000;
    set_no1 <=18;        //set segment display character '-'
    set_no2 <=18;
    set_no3 <=18;
    set_no4 <=18;
    save_no1 <=2;        //set password
    save_no2 <=5;
    save_no3 <=8;
    save_no4 <=0;
    pass_ok <=1;
end

// input. clock divider.
always begin
    /* 입력값을 반전시켜 저장
    * 입력 스위치가 pull-up 되어있는 듯.
    * 반전했을 때 하나의 bit만 H상태
    * key 입력에 대해 encoding 필요
    * 한번에 여러개의 키가 눌렸을 경우 처리 필요.
    */
    npb <=~pb;
    /* [31:0]init_counter 2^(-21) divid -> 50Mhz/(2^21)
    * init_counter의 20번째 bit는 clock_50m이 2^20주기마다 값이 바뀜
    * 따라서 sw_clk의 주파수는 50Mhz/(2^21) = 23.84Hz
    * 주기는 41.94ms
    * sw_clk = 23.84Hz

```

```

*/
sw_clk <=init_counter[20];    // clock for keypad(switch)
/* 2^(-17) divide -> 50Mhz/(2^17) = 381.47Hz
* 주기는 2.6214ms
* fnd_clk = 381.47Hz
*/
fnd_clk <=init_counter[16];    // clock for 7-segment
end

// clock_50m. clock counter.
always@(posedgeclock_50m) begin
    //50MHz clock signal
init_counter <=init_counter +1;
end

// sw_clk = 23.84Hz. get two consecutive inputs to correct switch(keypad) error.
always@(posedgesw_clk) begin
    pb_2nd <=pb_1st;        //이전 입력 저장
    pb_1st <=npb;          //현재 key pad 입력 반전 값

    if(pb_2nd == 'h0000 &&pb_1st !=pb_2nd) begin
        //이전 입력이 없고(중요!), 이전 입력과 현재 입력이 다를때
        //입력이 새로 들어왔을 때 입력 모드
        //하나의 키가 눌린 상태에서 또다른 키가 눌렸을 경우 입력 무시
        sw_toggle <=1;
    end

    if(sw_toggle ==1 &&pb_1st ==pb_2nd) begin
        //입력 모드이고, 이전 입력과 현재 입력이 같을 때
        //키가 계속 눌러 있는 상태로 생각
        //입력 토글 끈다.
        sw_toggle <=0;

        case(pb_1st)
            //모든 키 입력 경우의 수에 대해 동작 설정.
            //4개 버튼으로 만들어지는 순열을
        endcase
    end
end

// 7-segment.
always@(set_no1) begin
    case(set_no1)
        0: seg_1000 <='b0011_1111;        //0
        1: seg_1000 <='b0000_0110;        //1

```

```

        2: seg_1000 <='b0101_1011;          //2
        3: seg_1000 <='b0100_1111;          //3
        4: seg_1000 <='b0110_0110;          //4
        5: seg_1000 <='b0110_1101;          //5
        6: seg_1000 <='b0111_1101;          //6
        7: seg_1000 <='b0000_0111;          //7
        8: seg_1000 <='b0111_1111;          //8
        9: seg_1000 <='b0110_0111;          //9
       10: seg_1000 <='b0111_1000;          //t
       11: seg_1000 <='b0111_0011;          //P
       12: seg_1000 <='b0111_0111;          //A
       13: seg_1000 <='b0111_1100;          //b
       14: seg_1000 <='b0011_1001;          //C
       15: seg_1000 <='b0101_1110;          //d
       16: seg_1000 <='b0111_1001;          //E
       17: seg_1000 <='b0101_0000;          //r
       18: seg_1000 <='b0100_0000;          //-
       19: seg_1000 <='b0101_0100;
      default: seg_1000 <='b0000_0000;
    endcase
end

    //각 segment에 대한 출력...statment

    // fnd_clk. output.
    always@(posedgefnd_clk) begin
fnd_cnt <=fnd_cnt +1;
        case(fnd_cnt)
            3: begin
fnd_d <=seg_1000;
fnd_s <='b0111;
                end
            2: begin
fnd_d <=seg_100;
fnd_s <='b1011;
                end
            1: begin
fnd_d <=seg_10;
fnd_s <='b1101;
                end
            0: begin
fnd_d <=seg_1;
fnd_s <='b1110;
                end
        endcase
    end

```



```
end
endmodule
```

예제 코드에서는 각 기능들을 case 구문 안에서 case마다 처리하여 확장에 한계가 있다. 또한 6개의 키 입력을 순서대로 받아와서, 모든 경우의 수에 대해 코딩해 주어야 하는 것은 프로젝트를 더욱 복잡하게 만든다. 따라서 계산기 프로젝트에서는 각 기능들을 독립적인 module로 구현하여 확장성과 이식성을 높이겠다. 또한 task문을 이용하여 반복적인 작업을 캡슐화 하는 것을 주안점으로 둔다. 이에 따라 본 프로젝트는 segment_driver, keypad_driver, interface, calculate, clock_divider 5가지 파트로 나뉜다.

<출력>

■ **segment_driver(clk, rst, data, fnd_anode, nfnd_sel_out)**

입력 레지스터 6개, 출력 레지스터 6개

segment 출력에 필요한 문자 parameter 정의 or `define 이용

- binary2BCD
- decoder ← 비동기 reset 기능 포함

<입력>

■ **keypad_driver(clk, key, EBCD)**

keypad에 입력되는 10진수와 연산자, 명령을 우리팀이 정의한 확장된 BCD code로 encoding하여 전송

<interface>

■ **main(clk, rst, EBCD, result, operand, operator, fndPrint)**

input EBCD

input reg signed [31:0] resulte //연산 결과

output reg signed [31:0] operand [0:1] //피연산자 두개

output reg [1:0] operand //연산자

output reg [31:0] fndPrint //출력 문자(-99,999~999,999 + 문자)

reg error //error flag

- input_buffer

레지스터 2개

실시간 입력 레지스터 4bit : decoder를 통해 입력되는 데이터 확인 후 버퍼에 저장

입력 버퍼 32bit → 세그먼트에 실시간 출력

- EEROR_detector(resutle)

계산 결과 overflow 또는 (10)6자리 넘어갈 시 error

error 시 main으로 flag 전송

<연산기>

연산자에 따라 deMUX를 통해 해당 연산기로 값 전달 및 출력

- case문으로 처리 demultiplexer(operator(selector), operand[2])

- adder : 덧셈 - 보수이용
- subtractor : 뺄셈 - 보수이용
- divider : 나눗셈

****나눗셈 명령어****

SDIV: sigend 나눗셈

UDIV: unsigned 나눗셈

****divide by zero**** 에러는 나눗셈기로는 (하드웨어로는) 찾을 수 없다. 이는 소프트웨어의 영역이다.

- multiplier : 곱셈

곱셈 명령어

MUL: 곱셈 결과의 오른쪽 64비트를 제공한다.

SMULH: sigend 곱셈의 왼쪽 64비트를 제공한다.

UMULH: unsigend 곱셈의 왼쪽 64비트를 제공한다.

하나의 레지스터가 가지고 있을 수 있는 최대 비트는 64비트이다. 그래서 곱셈의 결과가 64비트를 넘어가면 오버플로우이다. 이는 SMULH나 UMULH의 값으로 오버플로우를 확인할 수 있다. SMULH, UMULH의 값이 0이 아니라면 오버플로우가 발생한 거고 곱셈의 값을 표현하기 위해 2개 이상의 레지스터가 필요하다는 뜻이다.

- modulo : 나머지

- OR gate multiplexer(operator(selector), resulte)

<클럭 분주>

// 각각 $\frac{1}{2^{21}}$, $\frac{1}{2^{17}}$ 로 분주하여 23.84Hz, 381.47Hz로 분주하여 출력하는 모듈

■ clock_divider(clock_50m, rst, sw_clk, fnd_clk)

input clock_50m

input rst

output sw_clk

output fnd_clk

3. 개발 환경 정의 및 이론적 배경

3.1 개발 환경

○ 하드웨어 사양

- FPGA : ALTERA Cyclone IV EP4CE22F17C6 FPGA “DE0-NANO“
- 4x4 tact switch :
- 7-segment : 최대 6자리까지 표현할 수 있으며 common cathode 방식의 FND(Flexible Numeric Display)이다. Dynamic 구동 방식으로 6자리 FND를 동작시킨다. FPGA 보드의 GPIO pin 14개를 이용하여 제어한다. 다음은 예제 코드에서 segment를 동작시키기 위해 이용한 출력이다.

fnd_selector: [5:0]fnd_s 출력을 6개의 segment 중 하나를 선택하여 dynamic 구동
fnd_digit : [7:0]fnd_d 출력을 이용해 segment의 anode에 전압 인가

○ 개발 환경

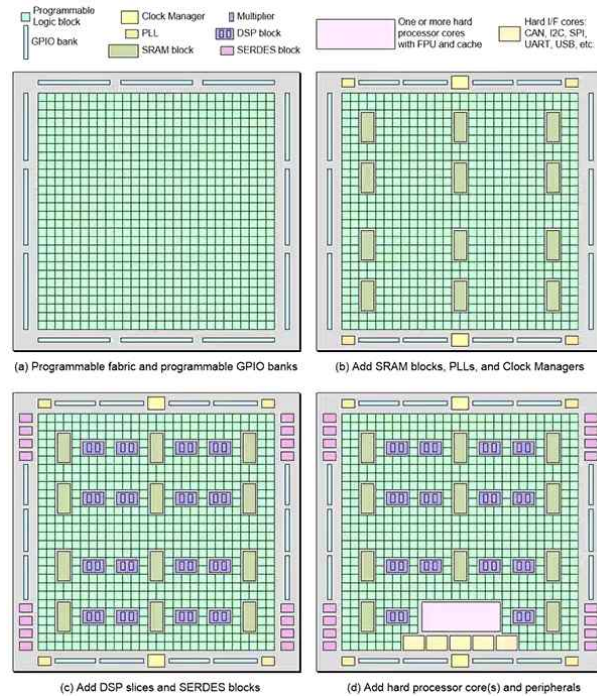
- IDE : intel quartus prime lite edition design software
- HDL: Verilog
- *simulation은 AMD xilinx Vivado design software 이용.

3.2 이론적 배경

3.2.1 FPGA

3.2.1-1 FPGA란?

FPGA는 프로그램 가능한 로직 블록과 다양한 기능 블록을 조합하여 다양한 디지털 회로를 구현할 수 있는 반도체 장치이다.



3.2.1-2 FPGA의 특성

-로직 블록 (Logic Block):

대부분의 FPGA는 4입력 LUT(룩업 테이블)를 기본으로 사용하며, 몇몇 FPGA는 6, 7, 또는 8입력 LUT를 제공할 수 있다. 이 LUT는 프로그램 가능한 논리 함수를 수행하며, 서로 상호 연결하여 복잡한 디지털 회로를 형성한다.

-레지스터와 논리 블록:

FPGA 내부의 논리 블록에서는 레지스터와 멀티플렉서를 사용하여 클럭 관리, 초기화 및 다양한 동작을 구현할 수 있다. 레지스터는 에지 트리거 플립플롭이나 레벨 감지 래치 등으로 구성될 수 있다.

-추가적인 기능 블록:

FPGA에는 블록 RAM(BRAM), 위상 고정 루프(PLL), 디지털 신호 처리(DSP) 블록, 고속 직렬 인터페이스(SERDES) 등 다양한 기능 블록이 포함될 수 있다.

-프로세서 코어:

일부 FPGA에는 소프트 코어나 하드 코어 프로세서가 내장되어 있다. 이는 FPGA에서 프로세서 기능을 구현할 수 있게 해주는데, 소프트 코어는 FPGA의 프로그래밍 가능 패브릭에서 동작하고, 하드 코어는 FPGA에 직접 구현된 것이다.

-주변 장치 인터페이스:

FPGA는 주변 장치 인터페이스 기능을 프로그래밍 가능한 패브릭에서 또는 하드 코어로 구현할 수 있다. 이러한 인터페이스는 CAN, I2C, SPI, UART, USB 등을 포함할 수 있다.

3.2.1-3 FPGA설계 및 사용법

-하드웨어 설명 언어(HDL) 사용:

FPGA 설계는 주로 Verilog 또는 VHDL과 같은 하드웨어 설명 언어를 사용한다.

-합성 도구 사용:

설명이 완료되면 합성 도구에 전달되어 FPGA에 프로그래밍할 수 있는 구성 파일을 생성한다. 각 FPGA 벤더는 고유한 합성 도구를 제공하거나 맞춤형 도구를 활용한다. 몇몇 FPGA 벤더는 소프트웨어 개발자가 FPGA를 사용하기 쉽도록 C, C++ 또는 OpenCL과 같은 상위 레벨 언어로 알고리즘을 설명할 수 있는 고급 합성 도구를 제공한다.

-평가 기판:

FPGA 설계자는 시작을 돕기 위해 다양한 개발 및 평가 기판을 사용한다. 이러한 기판은 다양한 FPGA를 포함하고 있으며, 예를 들어 Xilinx의 Zynq-7000 또는 Intel의 Cyclone V를 사용한 기판 등이 있다.


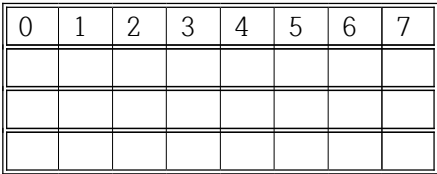
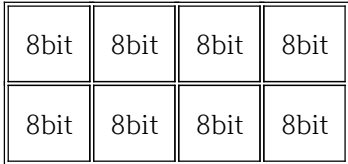
3.2.2 Verilog

3.2.2.1 Vector vs. array

Verilog에서 배열

배열은 연결이나 변수의 선언으로 스칼라나 벡터가 될 수 있다. 다차원 배열은 변수 명 다음에 표현된 범위에 대해 n 차원 공간으로 만들어질 수 있다. 배열은 Verilog에서 reg, wire, integer, real 데이터형에서 정의될 수 있다. 다음은 예시이다.

reg	y1 [11:0];	y1는 scalar reg 배열이며, depth = 12, each 1bit wide
wire	[0:7] y2 [3:0];	y2는 8bit 벡터 net으로 depth = 4
reg	[7:0] y3 [0:1][0:3];	y3는 2차원 배열로 2행, 4열을 가지고, each 8bit wide

reg y1 [11:0]	wire [0:7] y2 [3:0]	reg [7:0] y3 [0:1][0:3]
<p>scalar reg array, each 1bit</p> 	<p>8bit vector net with a depth of 4</p> 	<p>2D array, each 8bit wide</p> 

배열의 특정 요소에 접근하기 위해서 모든 차원의 배열 요소에 대한 index는 특정되어야 하며, index는 변수로 표현될 수 있다. 또한 Verilog에서 지원하는 어떠한 자료형에 대해서도 배열은 만들어질 수 있다.

1bit register n개를 저장하는 것과 n-bit vector를 저장하는 것은 같지 않다.

배열의 정의

y1 = 0;	// Illegal - All elements can't be assigned in a single go
y2[0] = 8'ha2;	// Assign 0xa2 to index=0
y2[2] = 8'h1c;	// Assign 0x1c to index=2
y3[1][2] = 8'hdd;	// Assign 0xdd to rows=1 cols=2
y3[0][0] = 8'haa;	// Assign 0xaa to rows=0 cols=0

배열 예시

module des ();	
reg [7:0] mem1;	// reg vector 8-bit wide
reg [7:0] mem2 [0:3];	// 8-bit wide vector array with depth=4
reg [15:0] mem3 [0:3][0:1];	// 16-bit wide vector 2D array with rows=4,cols=2
initial begin	
int i;	
mem1 = 8'ha9;	

```

$display ("mem1 = 0x%0h", mem1);

mem2[0] = 8'haa;
mem2[1] = 8'hbb;
mem2[2] = 8'hcc;
mem2[3] = 8'hdd;
for(i = 0; i < 4; i = i+1) begin
    $display("mem2[%0d] = 0x%0h", i, mem2[i]);
end

for(int i = 0; i < 4; i += 1) begin
    for(int j = 0; j < 2; j += 1) begin
        mem3[i][j] = i + j;
        $display("mem3[%0d][%0d] = 0x%0h", i, j, mem3[i][j]);
    end
end
end
endmodule

```

3.2.3 Hierarchica Project(계층적 프로젝트)

설계가 복잡해지면 기능 단위 블록으로 나누어 설계한 후, 상위 계층에서 통합하는 방법인 계층적 설계를 실시한다. 다음은 계층적으로 설계하는 과정의 간단한 예이다.

계층적으로 설계하는 과정은, 먼저 전체 설계를 기능적으로 분리한 다음에 각 기능들을 HDL로 설계하고 컴파일한 후, 시뮬레이션을 통해 동작에 이상이 없는지 확인하여 Block Symbol File을 생성한다.

상위 계층에서는 새로운 프로젝트를 생성하고 새로운 디자인 파일로 Block Diagram/Schematic파일을 연다. 이 파일에서는 schematic 설계가 가능하며, 앞에서 생성한 Block Symbol File 및 라이브러리에서 제공하는 다른 심볼과 함께 설계할 수도 있다.

이 예에서는 4x1 multiplexer(MUX41)를 HDL로 설계하고 BSF로 생성한 후 상위 계층에서 이용해 설계하는 과정을 보여준다. 또한 상위계층에서 Schematic 방법을 사용하지 않고 HDL을 사용해 계층적인 설계를 할 수도 있다.

Verilog의 구조적 표현(Structural Description)은 실제 회로를 구성하는 컴포넌트 간의 연결을 정의한다. Verilog에서는 하나의 module을 정의하고, 정의된 module을 다른 module에서 instantiation에 의해 컴포넌스 상호 간의 연결을 정의할 수 있다. module의 객체를 생성하는 형식은 다음과 같다.

```

module_name  instance_name1 (terminal, terminal ...);
              instance_name2 (terminal, terminal ...);

```

이때 객체에 값을 전달하기 위해 Referenced by Position, Referenced by Name 두가지 방식에 의해 전달할 수 있다.

- Referenced by Position(위치에 의한 참조)

```

module_name  instance_name1 (terminal, terminal ...);
              instance_name2 (terminal, terminal ...);

```

module에서 정의된 port의 순서에 따라 해당 위치에 값을 전달하는 방식이다.

- Referenced by Name(이름에 의한 참조)

```

module_name  instance_name1 (.prot(terminal), .port(terminal) ...);
              instance_name2 (.port(terminal), .port(terminal) ...);

```

module의 port 이름에 대응되는 값을 연결시켜 값을 전달하는 방식이다. module의 port 수가 늘어나, 순서대로 값을 전달하는 것이 비효율적일 경우 이름에 의한 참조로 값을 전달할 수 있다. 또한 module의 해당 위치에 어떤 포트가 위치하는지 일일이 확인할 필요 없이 port에 직접 값을 대응시키므로 가독성이 더욱 뛰어나다.

3.2.4 segment

3.2.7-1 segment란?

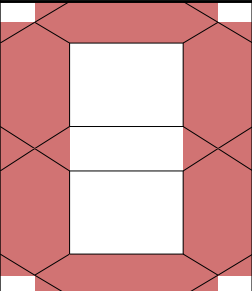
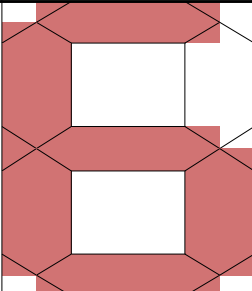
세그먼트는 프로그램에 정의된 특정 영역으로, 코드, 데이터, 그리고 스택(stack)으로 알려져 있는 것을 포함한다. 한 세그먼트는 패러그래프 경계(paragraph boundary), 즉 16또는 hex 10으로 나누어지는 위치에서 시작하며 세그먼트는 메모리의 거의 어느 곳이나 위치할 수 있다, 명령어가 세그먼트 레지스터에 세그먼트 주소를 적재할 때, 가장 오른쪽에 위치한 네 개의 0비트가 자동으로 오른쪽으로 이동하면서 제거된다.

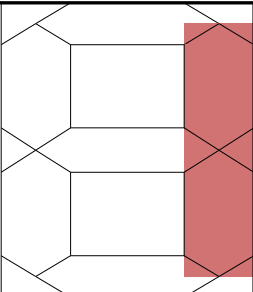
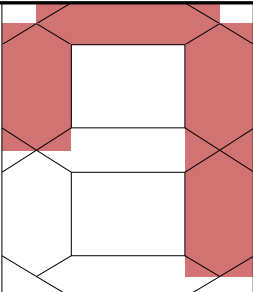
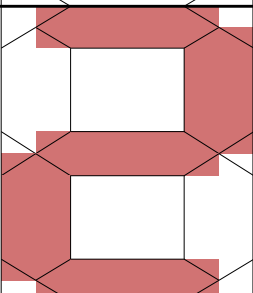
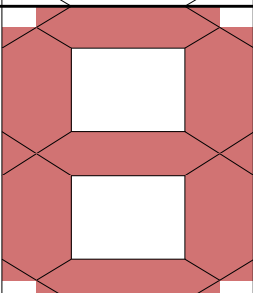
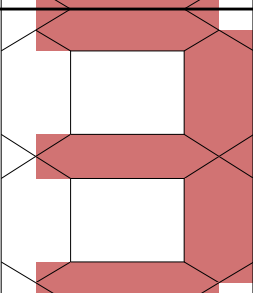
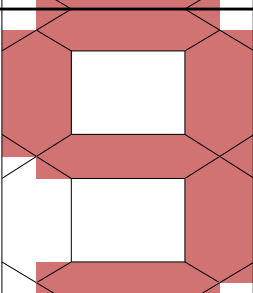
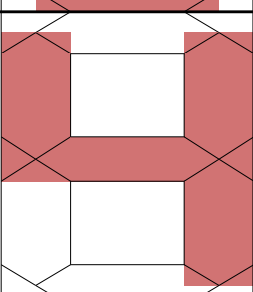
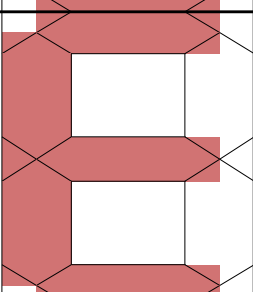
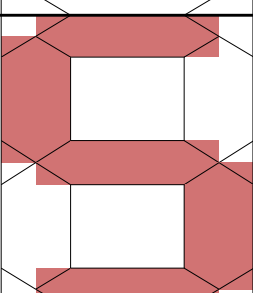
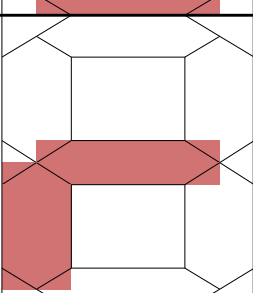
3.2.7-2. segment 특성

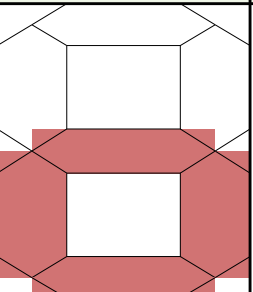
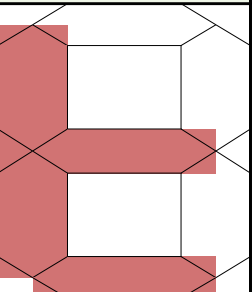
대응값	gfedcba	segment출력	대응값	gfedcba	segment출력
0	01111111	0	10	1111000	t
1	0000110	1	11	1110011	p
2	1011011	2	12	1110111	A
3	1001111	3	13	1111100	b
4	1100110	4	14	0111001	C
5	1101101	5	15	1011110	d
6	1111101	6	16	1111001	E
7	0000111	7	17	1010000	r
8	1111111	8	18	1000000	-
9	1100111	9	19	1010100	n
			else	000000	미출력

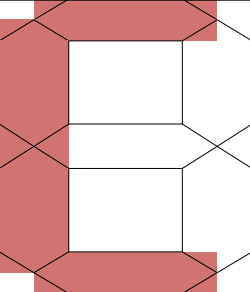
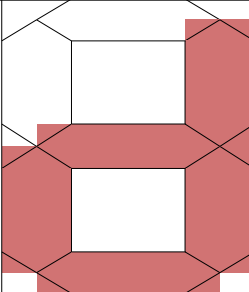
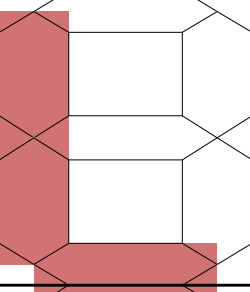
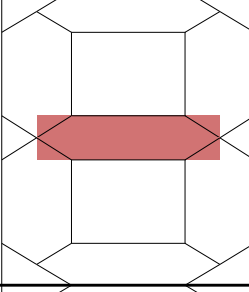
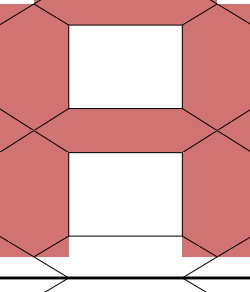
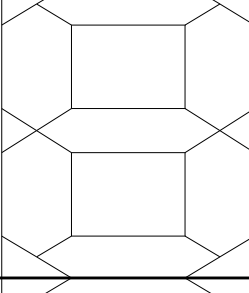
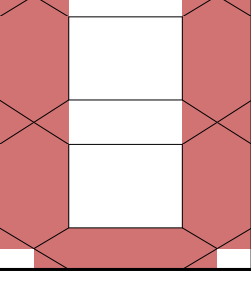
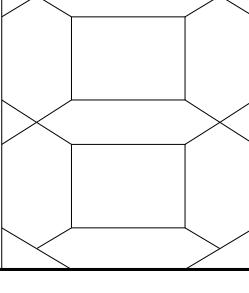
3.2.7-3. 각 segment로 표현 할 값 설정

Code는 "0000000" 순으로 "gfedcba" 와 대응

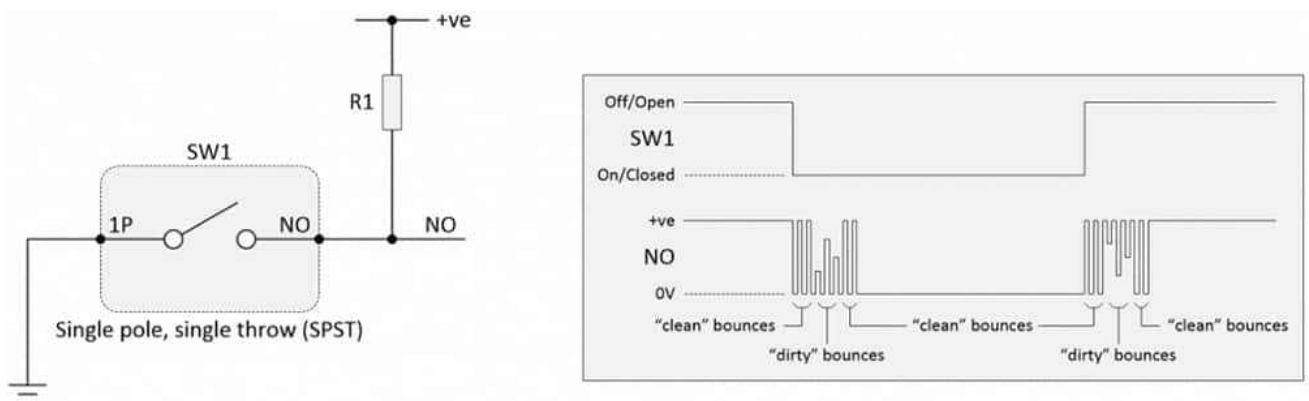
Letter	Mean	Code	Segment	Letter	Mean	Code	Segment
0	0	01111111		6	6	1111101	

1	1	000110		7	7	0100111	
2	2	1011011		8	8	1111111	
3	3	1001111		9	9	1101111	
4	4	1100110		10	E	1001111	
5	5	1101101		11	r	1010000	

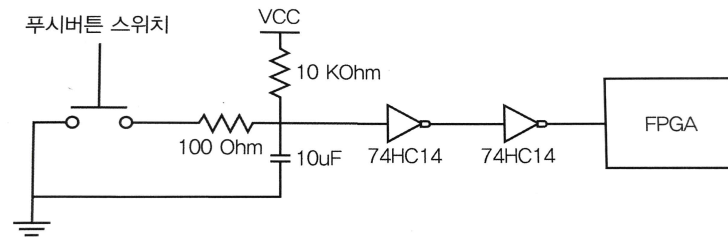
Letter	Mean	Code	Segment	Letter	Mean	Code	Segment
12	o	1011100		17	* times	1111000	

13	C	0111001		18	/ div	1011110	
14	L	0111000		19	- minus	1000000	
15	+ Add	1110111		20	Null	0000000	
16	- Sub	0111110		-	-	-	

3.2.5 Debounce circuit



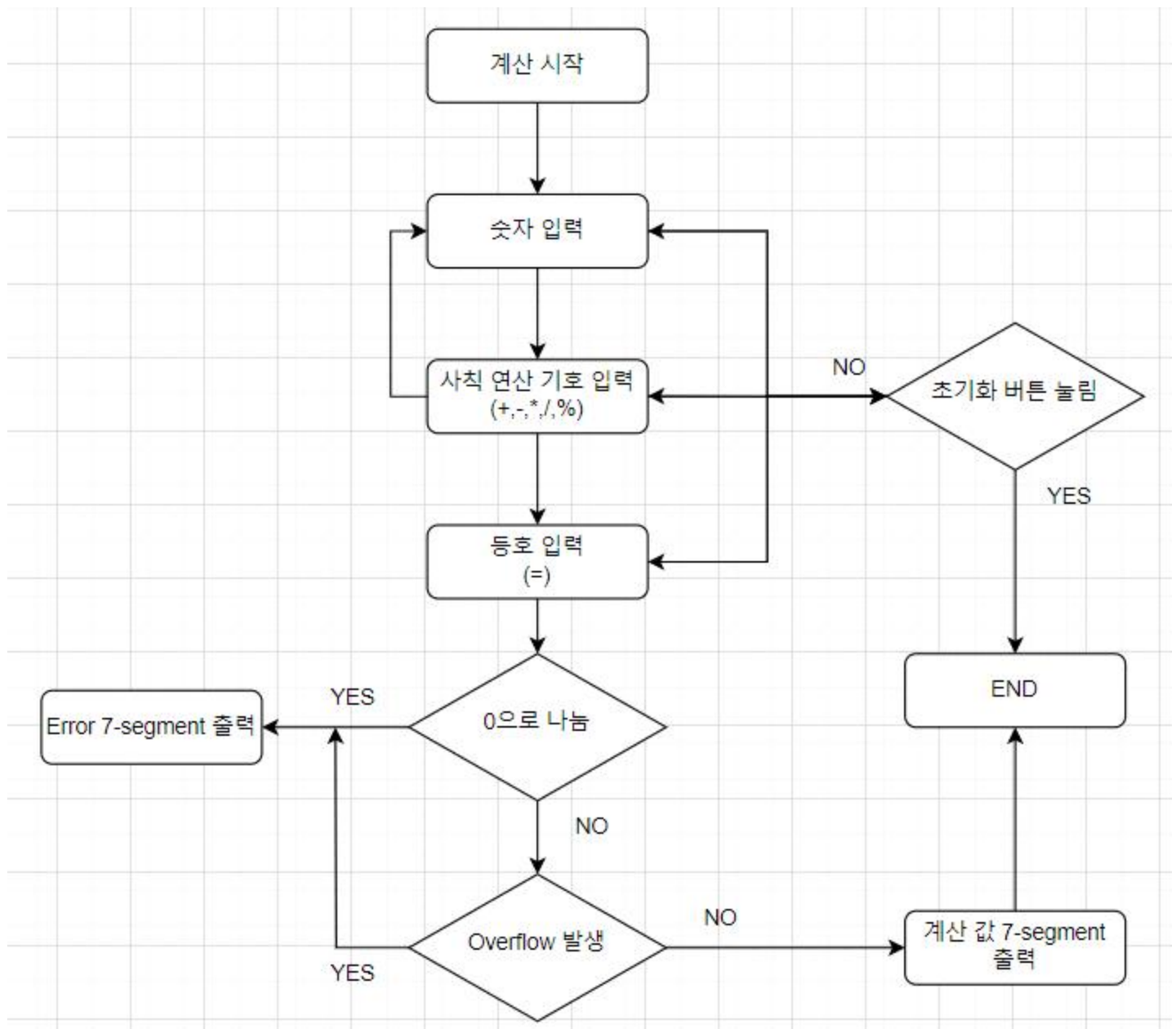
물리적 접점이 있는 push 스위치가 open 상태일 때에는 입력 pin이 pull-up 되어 있으므로 H상태에 놓여있게 된다. 하지만 버튼을 close 하면 0V로 값이 떨어지게 된다. 이때 스위치 출력단에서 chattering이 발생한다. 이때 생기는 ripple 신호는 debounce 회로에 의해 제거된다. 다음은 푸시버튼 스위치의 debounce 회로 예이다.



[그림 3-27] 푸시버튼 스위치의 회로도

총 16개의 푸시버튼 스위치가 있으므로 각 스위치에 'h0 ~ 'hF의 기 값을 할당하고, 각 스위치는 debounce 회로를 통해 FPGA의 GPIO에 연결된다.

4. 프로젝트 동작 블록 다이어그램



5. 조원 역할 분담

정준혁	안재형	한현빈
<ul style="list-style-type: none">- 블록다이어그램 작성- 요구사항 분석 및 해결방안 작성	<ul style="list-style-type: none">- 모듈 설계 기획 및 schematic- 블록다이어그램 수정 및 추가 설계	<ul style="list-style-type: none">- 설계과정 예상 문제 및 해결방안 작성- 개발환경 정의 및 이론적 배경

6. 참고문헌

segment: <https://lowsec.tistory.com/2> [On & On:티스토리]

fpga:<https://www.digikey.kr/ko/articles/fundamentals-of-fpgas-what-are-fpgas-and-why-are-they-needed>