

# 명지 공학인 윤리 서약서

## 보고서 및 논문 윤리 서약

1. 나는 보고서 및 논문의 내용을 조작하지 않겠습니다.
2. 나는 다른 사람의 보고서 및 논문의 내용을 내 것처럼 무단으로 복사하지 않겠습니다.
3. 나는 다른 사람의 보고서 및 논문의 내용을 참고하거나 인용할 시 참고 및 인용 형식을 갖추고 출처를 반드시 밝히겠습니다.
4. 나는 보고서 및 논문을 대신하여 작성하도록 청탁하지도 청탁받지도 않겠습니다.  
나는 보고서 및 논문 작성 시 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

## 시험 윤리 서약

1. 나는 대리시험을 청탁하거나 청탁받지 않겠습니다.
2. 나는 허용되지 않은 교과서, 노트 및 타학생의 답안지 등을 보고 답안지를 작성하지 않겠습니다.
3. 나는 타인에게 답안지를 보여주지 않겠습니다.
4. 나는 감독관의 지시와 명령에 따라 시험 과정에 참여하겠습니다.  
나는 시험에 위법 행위를 하지 않고, 명지인으로서 또한 공학인으로서 나의 양심과 명예를 지킬 것을 약속합니다.

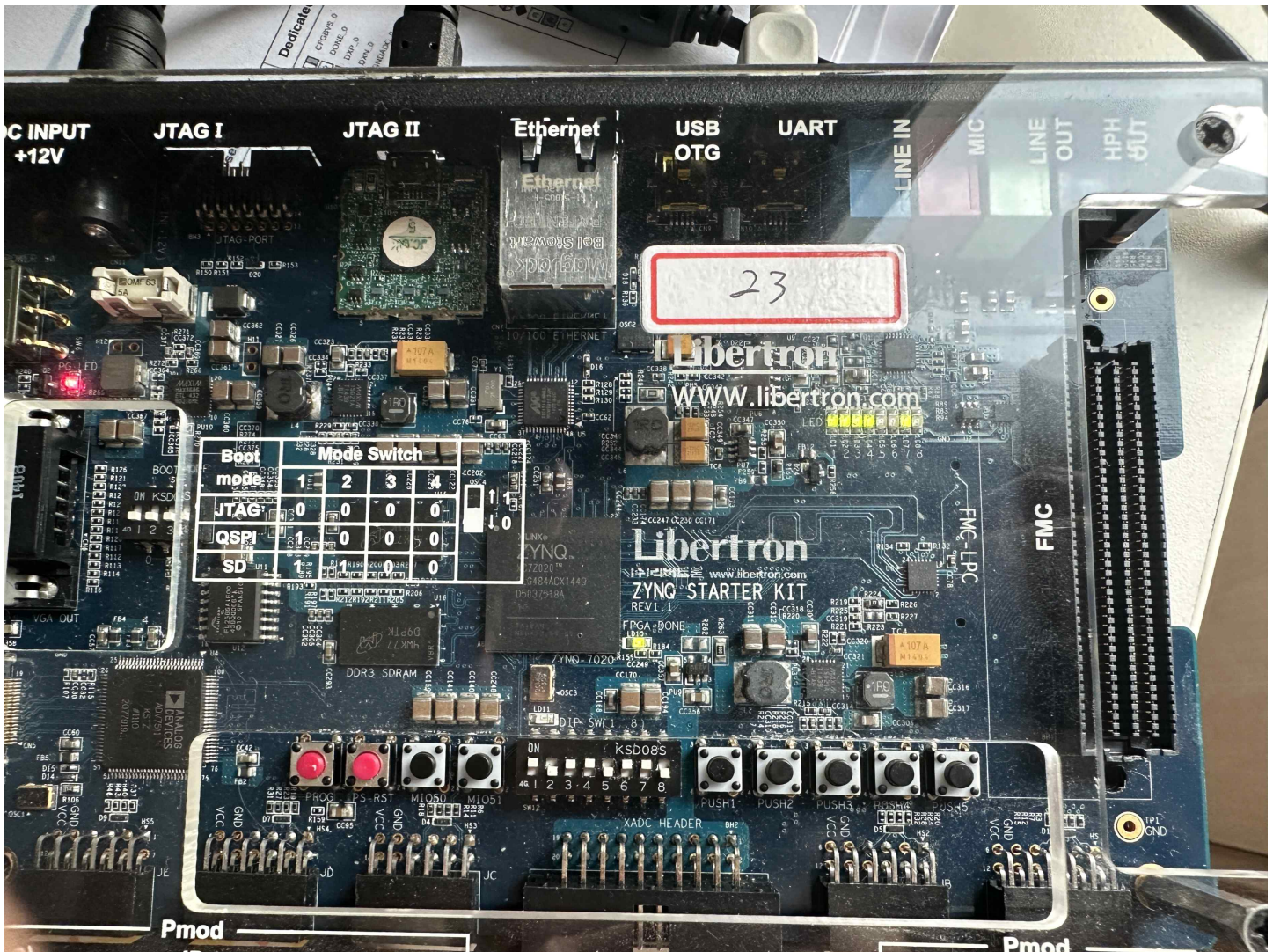
2023년 03월 25일

서약자

(학번) 60171878

(성명) 허무혁 (인)

HW1\_ALU  
60171878 허무혁



목차

1. Vivado System Blcok Diagram
2. SDK 소스코드
3. 구현모습
4. Discussion

```
* HW1_ALU.c
*
* This application configures UART 16550 to baud rate 9600.
* PS7 UART (Zynq) is not initialized by this application, since
* bootrom/bsp configures it to baud rate 115200
*
* -----
* | UART TYPE      BAUD RATE |
* -----
*   uartns550      9600
*   uartlite        Configurable only in HW design
```

```

*   ps7_uart    115200 (configured by bootrom/bsp)
*/

/*
* PL의 dip switch와 button을 입력으로 받아
* PS에서 ALU를 구현하고
* PL의 LED로 출력하는 프로그램이다.
* dip 스위치의 앞 4bit는 opcode로 사용한다.
* dip 스위치의 뒷 4bit는 operand로 사용한다.
* 버튼을 누르면 R값이 0000_0001로 초기화 한다.
* 연산 후 결과값을 led를 통해 출력한다.
*/
#include <stdio.h>
#include <unistd.h>
#include "platform.h"
#include "xparameters.h"
#include "xgpio.h"

//overflow 를 검사하는 코드
u8 add_with_overflow(u8 a, u8 b, u8 *overflow){
    u16 result = (u16)a + (u16)b;
    if(result > 0xff)
        *overflow = 1;
    else
        *overflow = 0;
    return (u8)result;
}

u8 multi_with_overflow(u8 a, u8 b, u8 *overflow){
    u16 result = (u16)a * (u16)b;
    if(result > 0xff)
        *overflow = 1;
    else
        *overflow = 0;
    return (u8)result;
}

//underflow 를 검사하는 코드
u8 sub_with_underflow(u8 a, u8 b, u8 *underflow){
    if(a < b)
        *underflow = 1;
    else
        *underflow = 0;
    return (u8)(a-b);
}

// rotation left 하는 코드
u8 rotate_left(u8 result, int shift){
    shift %= 8;
    return (result << shift) | (result >> (8 - shift));
}

// rotation right 하는 코드
u8 rotate_right(u8 result, int shift){
    shift %= 8;
    return (result >> shift) | (result << (8 - shift));
}

int main (void)
{
    static XGpio SW;           //디프스위치의 InstancePtr
    static XGpio BTN;          //버튼의 InstancePtr
    static XGpio LED;          //LED의 InstancePtr
    u8 SW_status;              //디프스위치의 상태를 저장할 변수
    u8 BTN_status;             //버튼의 상태를 저장할 변수
    u8 opcode;
    u8 stepsize;

    u8 overflow;
    u8 underflow;

    u8 Result = 0x01;          //ALU의 값을 저장할 변수 unsigned로 선언
    /*PL GPIO 상태 체크 및 입출력 설정*/
    int status;
    status = XGpio_Initialize(&SW, XPAR_SW_DEVICE_ID);
    //int XGpio_Initialize(XGpio *InstancePtr, u16 DeviceId);

```



```

//XPAR_SW_DEVICE_ID는 xparameters.h에 정의
if (status != XST_SUCCESS){
    return XST_FAILURE;
}

XGpio_SetDataDirection(&SW, 1, 0xff); //1번 채널, 입력으로 지정.
/*
 * void XGpio_SetDataDirection (XGpio *      InstancePtr,
 *                               unsigned      Channel,
 *                               u32          DirectionMask
 *                               )
 * InstancePtr is a pointer to an XGpio instance to be worked on.
 * Channel contains the channel of the GPIO (1 or 2) to operate on.
 * DirectionMask is a bitmask specifying which discretes are input and which are output. Bits set
to 0 are output and bits set to 1 are input.
 */

status = XGpio_Initialize(&BTN, XPAR_BTN_DEVICE_ID);
if (status != XST_SUCCESS)
return XST_FAILURE;
XGpio_SetDataDirection(&BTN, 1, 0xff); //1번 채널, 입력으로 지정.

status = XGpio_Initialize(&LED, XPAR_LED_DEVICE_ID);
if (status != XST_SUCCESS)
return XST_FAILURE;
XGpio_SetDataDirection(&LED, 1, 0x00); //1번 채널, 출력으로 지정.
////////////////////

while (1) // infinite loop
{
    BTN_status = XGpio_DiscreteRead(&BTN, 1);
    //u32 XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel);
    if(BTN_status == 1){
        Result = 0x01;           //버튼 입력이 1이면 값 0x01로 초기화
        xil_printf("BTN Reset = %u \n\r", Result);
    }

    SW_status = XGpio_DiscreteRead(&SW, 1);
    opcode = (SW_status >> 4) & 0xF; //SW_status의 앞 4bit 추출
    stepsize = SW_status & 0xF;      //SW_status의 뒷 4bit 추출

    switch(opcode){
        //덧셈 연산
        case 0b0000:
            Result = add_with_overflow(Result, stepsize, &overflow);
            if(overflow){
                xil_printf("overflow! Reset R = 0x01 \n\r");
                Result = 0x01;
            }
            else{
                xil_printf("ADD operation R = R + %u \n\r", stepsize);
                xil_printf("Result = %u \n\r", Result);
            }
            break;
        //뺄셈 연산
        case 0b0001:
            Result = sub_with_underflow(Result, stepsize, &underflow);
            if(underflow){
                xil_printf("underflow! Reset R = 0x01 \n\r");
                Result = 0x01;
            }
            else{
                xil_printf("SUB operation R = R - %u \n\r", stepsize);
                xil_printf("Result = %u \n\r", Result);
            }
            break;
        //곱셈연산
        case 0b0010:
            Result = multi_with_overflow(Result, stepsize, &overflow);
            if(overflow){
                xil_printf("overflow! Reset R = 0x01 \n\r");
                Result = 0x01;
            }

```

```

    }
    else{
        xil_printf("Multiplication R = R * %u \n\r", stepsize);
        xil_printf("Result = %u \n\r", Result);
    }
    break;
//왼쪽으로 이동
case 0b0011:
    Result = rotate_left(Result, stepsize);
    xil_printf("rotate_left stepsize = %u \n\r", stepsize);
    xil_printf("Result = %u \n\r", Result);
    break;
//오른쪽으로 이동
case 0b0100:
    Result = rotate_right(Result, stepsize);
    xil_printf("rotate_right stepsize = %u \n\r", stepsize);
    xil_printf("Result = %u \n\r", Result);
    break;
//비트 and 연산
case 0b0101:
    Result = Result & stepsize; //bitwise and
    xil_printf("bitwise and /stepsize = %u \n\r", stepsize);
    xil_printf("Result = %u %6 \n\r", Result);
    break;
//비트 or 연산
case 0b0111:
    Result = Result | stepsize; //bitwise or
    xil_printf("bitwise or /stepsize = %u \n\r", stepsize);
    xil_printf("Result = %u \n\r", Result);
    break;
//비트 xor 연산
case 0b1000:
    Result = Result ^ stepsize; //bitwise xor
    xil_printf("bitwise xor /stepsize = %u \n\r", stepsize);
    xil_printf("Result = %u \n\r", Result);
    break;
}

XGpio_DiscreteWrite(&LED, 1, Result); //ALU연산 결과값을 LED로 출력함.
//void XGpio_DiscreteWrite(XGpio *InstancePtr, unsigned Channel, u32 Mask);

sleep(1); //1 초 지연
}
}

```

### 3. 구현모습

```

COM4 - Tera Term VT
메뉴(F)  수정(E)  설정(S)  제어(O)  창(W)  도움말(H)
Result = 6
ADD operation R = R + 1
Result = 7
ADD operation R = R + 1
Result = 8
ADD operation R = R + 1
Result = 9
ADD operation R = R + 3
Result = 12
ADD operation R = R + 2
Result = 14
ADD operation R = R + 7
Result = 21
ADD operation R = R + 7
Result = 28
ADD operation R = R + 5
Result = 33
ADD operation R = R + 15
Result = 48
ADD operation R = R + 11
Result = 59
ADD operation R = R + 9
Result = 68
ADD operation R = R + 8

```

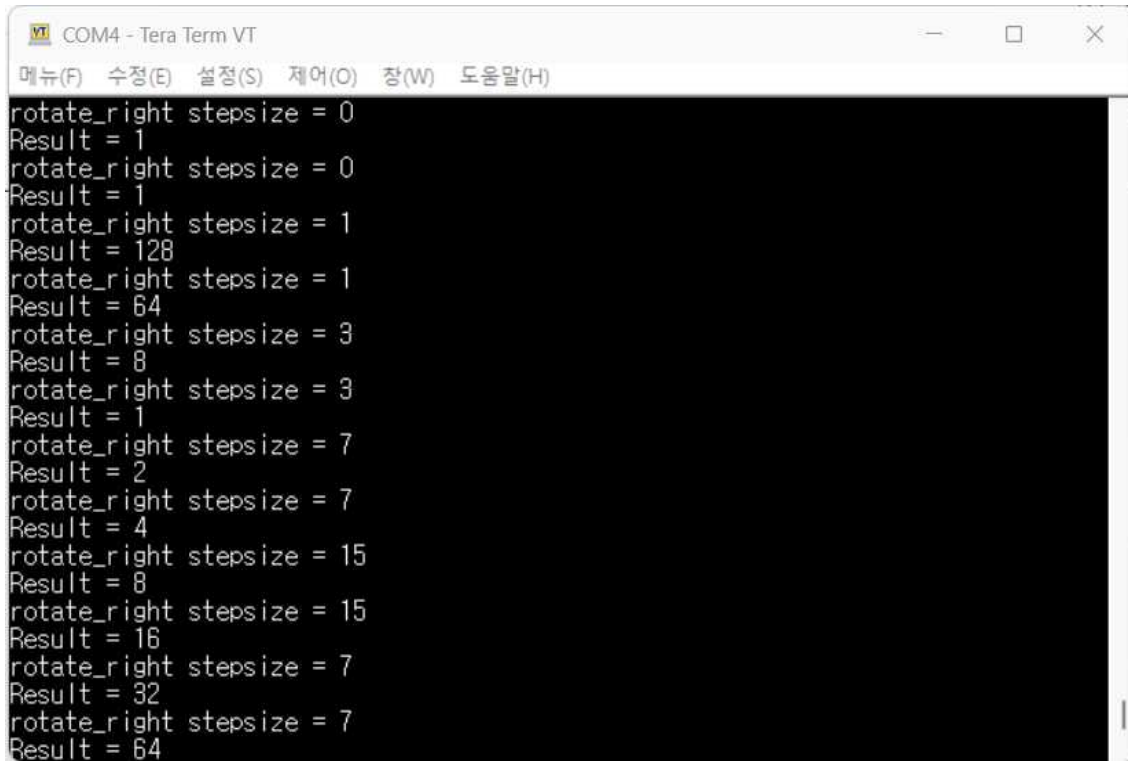
그림 4 ADD operation

```
COM4 - Tera Term VT
메뉴(F)  수정(E)  설정(S)  제어(O)  창(W)  도움말(H)
SUB operation R = R - 7
Result = 196
SUB operation R = R - 7
Result = 189
SUB operation R = R - 7
Result = 182
SUB operation R = R - 7
Result = 175
SUB operation R = R - 3
Result = 172
SUB operation R = R - 7
Result = 165
SUB operation R = R - 15
Result = 150
SUB operation R = R - 15
Result = 135
SUB operation R = R - 15
Result = 120
SUB operation R = R - 15
Result = 105
SUB operation R = R - 15
Result = 90
SUB operation R = R - 15
Result = 75
```

그림 5 Sub operation

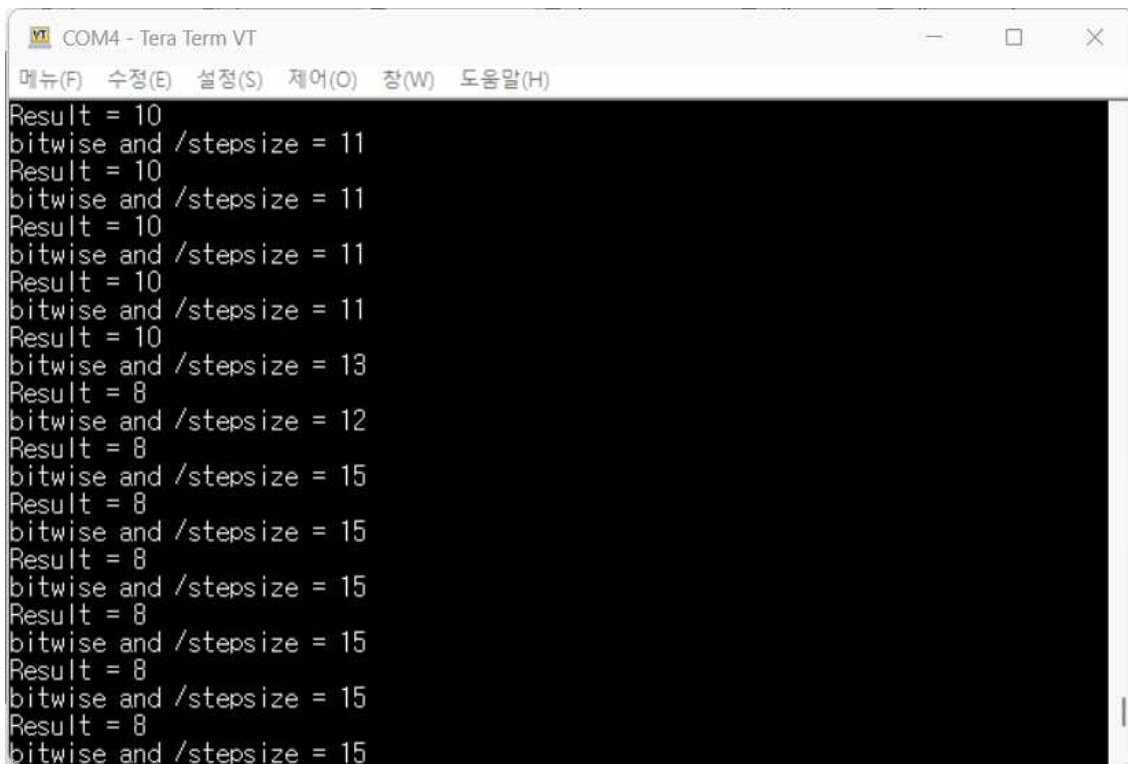
```
COM4 - Tera Term VT
메뉴(F)  수정(E)  설정(S)  제어(O)  창(W)  도움말(H)
Multiplication R = R * 1
Result = 1
Multiplication R = R * 1
Result = 1
Multiplication R = R * 3
Result = 3
Multiplication R = R * 3
Result = 9
Multiplication R = R * 3
Result = 27
Multiplication R = R * 7
Result = 189
overflow! Reset R = 0x01
Multiplication R = R * 7
Result = 7
Multiplication R = R * 3
Result = 21
Multiplication R = R * 3
Result = 63
Multiplication R = R * 3
Result = 189
overflow! Reset R = 0x01
Multiplication R = R * 11
Result = 11
```

그림 6 Multiplication operation



```
COM4 - Tera Term VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
rotate_right stepsize = 0
Result = 1
rotate_right stepsize = 0
Result = 1
rotate_right stepsize = 1
Result = 128
rotate_right stepsize = 1
Result = 64
rotate_right stepsize = 3
Result = 8
rotate_right stepsize = 3
Result = 1
rotate_right stepsize = 7
Result = 2
rotate_right stepsize = 7
Result = 4
rotate_right stepsize = 15
Result = 8
rotate_right stepsize = 15
Result = 16
rotate_right stepsize = 7
Result = 32
rotate_right stepsize = 7
Result = 64
```

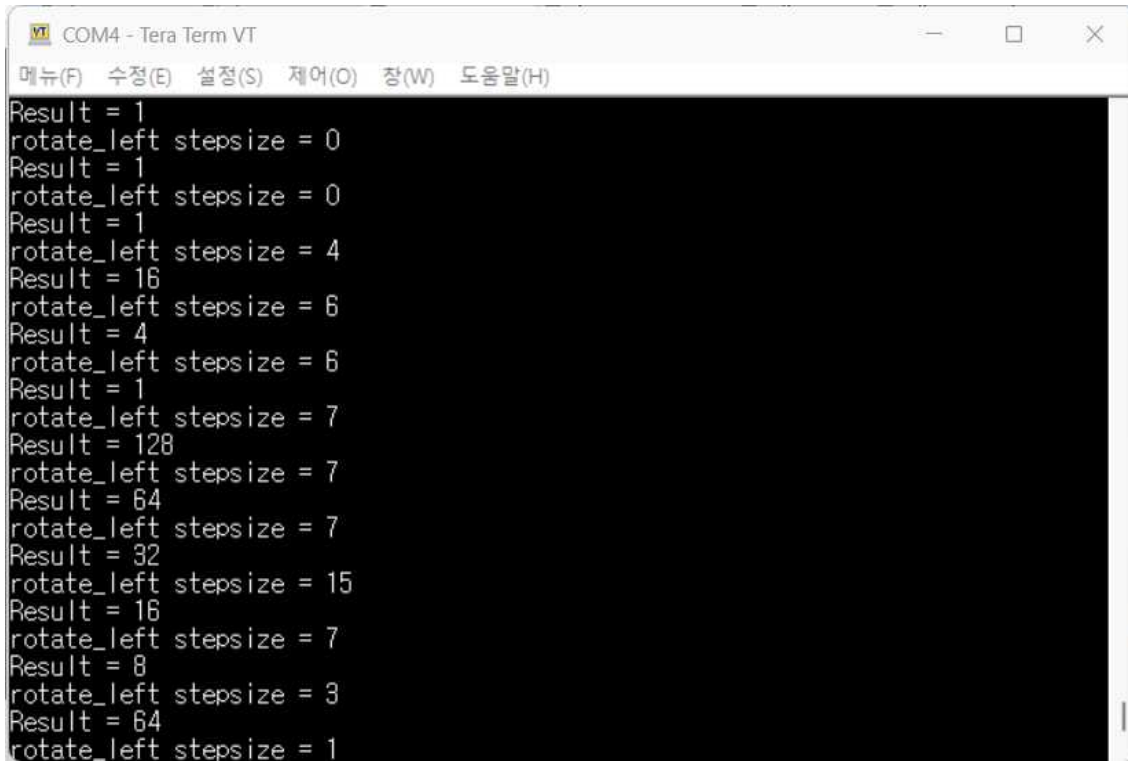
그림 7 rotate\_right 연산



```
COM4 - Tera Term VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
Result = 10
bitwise and /stepsize = 11
Result = 10
bitwise and /stepsize = 11
Result = 10
bitwise and /stepsize = 11
Result = 10
bitwise and /stepsize = 11
Result = 10
bitwise and /stepsize = 13
Result = 8
bitwise and /stepsize = 12
Result = 8
bitwise and /stepsize = 15
Result = 8
bitwise and /stepsize = 15
Result = 8
bitwise and /stepsize = 15
Result = 8
bitwise and /stepsize = 15
Result = 8
bitwise and /stepsize = 15
Result = 8
bitwise and /stepsize = 15
```

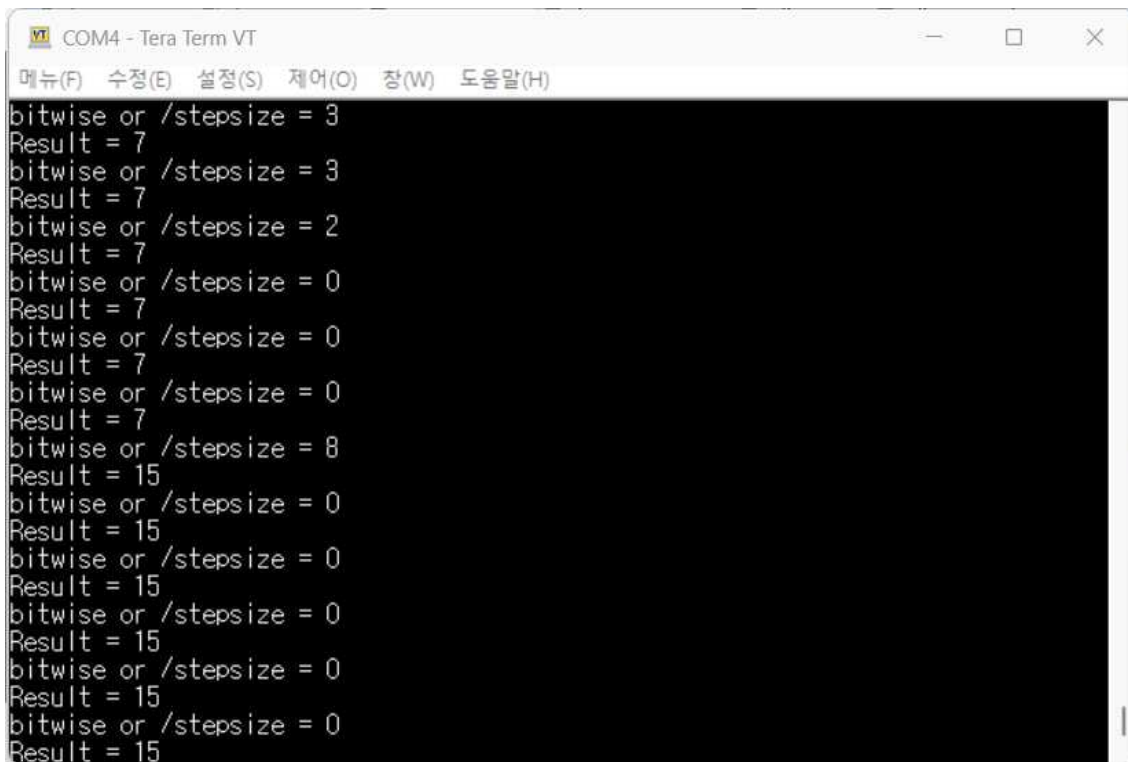
그림 8 bitwise and 연산





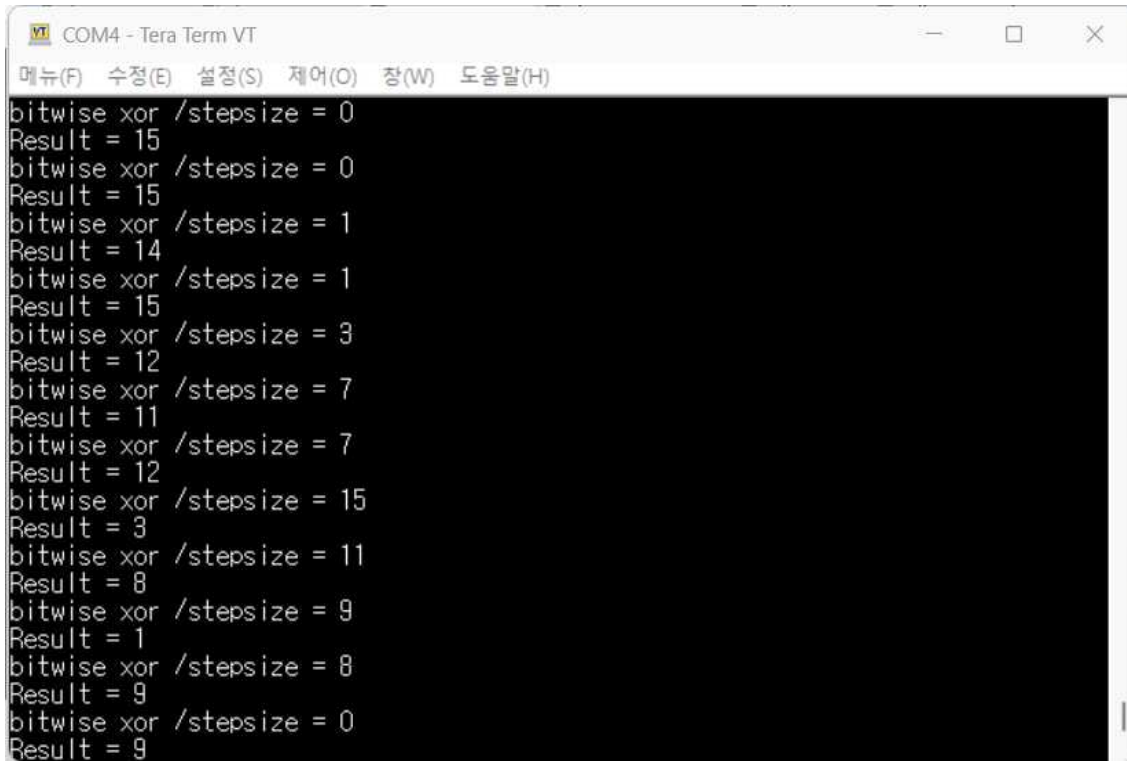
```
COM4 - Tera Term VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
Result = 1
rotate_left stepsize = 0
Result = 1
rotate_left stepsize = 0
Result = 1
rotate_left stepsize = 4
Result = 16
rotate_left stepsize = 6
Result = 4
rotate_left stepsize = 6
Result = 1
rotate_left stepsize = 7
Result = 128
rotate_left stepsize = 7
Result = 64
rotate_left stepsize = 7
Result = 32
rotate_left stepsize = 15
Result = 16
rotate_left stepsize = 7
Result = 8
rotate_left stepsize = 3
Result = 64
rotate_left stepsize = 1
```

그림 9 rotate\_left 연산



```
COM4 - Tera Term VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
bitwise or /stepsize = 3
Result = 7
bitwise or /stepsize = 3
Result = 7
bitwise or /stepsize = 2
Result = 7
bitwise or /stepsize = 0
Result = 7
bitwise or /stepsize = 0
Result = 7
bitwise or /stepsize = 0
Result = 7
bitwise or /stepsize = 8
Result = 15
bitwise or /stepsize = 0
Result = 15
bitwise or /stepsize = 0
Result = 15
bitwise or /stepsize = 0
Result = 15
bitwise or /stepsize = 0
Result = 15
bitwise or /stepsize = 0
Result = 15
```

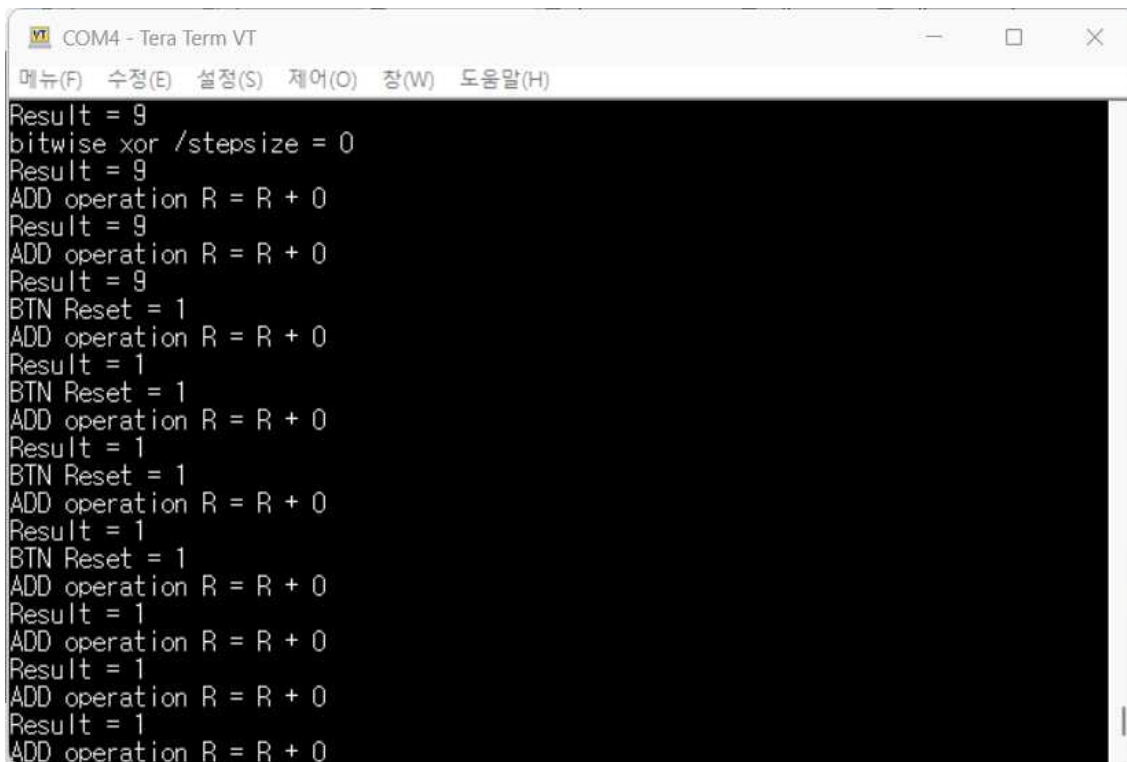
그림 10 bitwise or 연산



A screenshot of a Tera Term VT window titled "COM4 - Tera Term VT". The window has a menu bar with "메뉴(F)", "수정(E)", "설정(S)", "제어(O)", "창(W)", and "도움말(H)". The main area displays a series of bitwise xor operations and their results:

```
bitwise xor /stepsize = 0
Result = 15
bitwise xor /stepsize = 0
Result = 15
bitwise xor /stepsize = 1
Result = 14
bitwise xor /stepsize = 1
Result = 15
bitwise xor /stepsize = 3
Result = 12
bitwise xor /stepsize = 7
Result = 11
bitwise xor /stepsize = 7
Result = 12
bitwise xor /stepsize = 15
Result = 3
bitwise xor /stepsize = 11
Result = 8
bitwise xor /stepsize = 9
Result = 1
bitwise xor /stepsize = 8
Result = 9
bitwise xor /stepsize = 0
Result = 9
```

그림 11 bitwise xor 연산



A screenshot of a Tera Term VT window titled "COM4 - Tera Term VT". The window has a menu bar with "메뉴(F)", "수정(E)", "설정(S)", "제어(O)", "창(W)", and "도움말(H)". The main area displays a sequence of operations and their results:

```
Result = 9
bitwise xor /stepsize = 0
Result = 9
ADD operation R = R + 0
Result = 9
ADD operation R = R + 0
Result = 9
BTN Reset = 1
ADD operation R = R + 0
Result = 1
BTN Reset = 1
ADD operation R = R + 0
Result = 1
BTN Reset = 1
ADD operation R = R + 0
Result = 1
BTN Reset = 1
ADD operation R = R + 0
Result = 1
ADD operation R = R + 0
Result = 1
ADD operation R = R + 0
Result = 1
ADD operation R = R + 0
```

그림 12 버튼 눌러 초기화

```
COM4 - Tera Term VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
Result = 135
SUB operation R = R - 15
Result = 120
SUB operation R = R - 15
Result = 105
SUB operation R = R - 15
Result = 90
SUB operation R = R - 15
Result = 75
SUB operation R = R - 15
Result = 60
SUB operation R = R - 15
Result = 45
SUB operation R = R - 15
Result = 30
SUB operation R = R - 15
Result = 15
SUB operation R = R - 15
Result = 0
underflow! Reset R = 0x01
underflow! Reset R = 0x01
underflow! Reset R = 0x01
underflow! Reset R = 0x01
```

그림 13 언더플로우 발생 경고 후 초기화

```
COM4 - Tera Term VT
메뉴(F) 수정(E) 설정(S) 제어(O) 창(W) 도움말(H)
Multiplication R = R * 0
Result = 0
Multiplication R = R * 12
Result = 0
Multiplication R = R * 14
Result = 0
Multiplication R = R * 15
Result = 0
BTN Reset = 1
Multiplication R = R * 15
Result = 15
Multiplication R = R * 15
Result = 225
overflow! Reset R = 0x01
Multiplication R = R * 15
Result = 15
Multiplication R = R * 15
Result = 225
overflow! Reset R = 0x01
Multiplication R = R * 15
Result = 15
Multiplication R = R * 15
Result = 225
```

그림 14 오버플로우 발생 경고 후 초기화

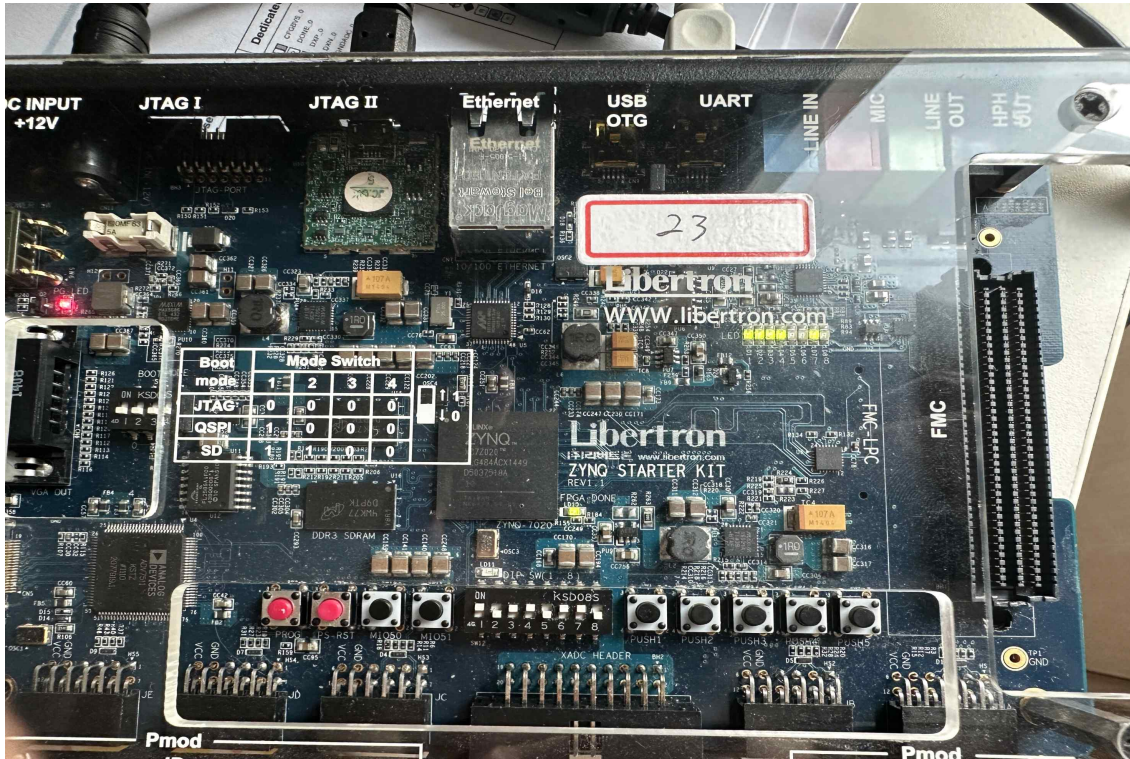


그림 15 결과값 LED로 출력

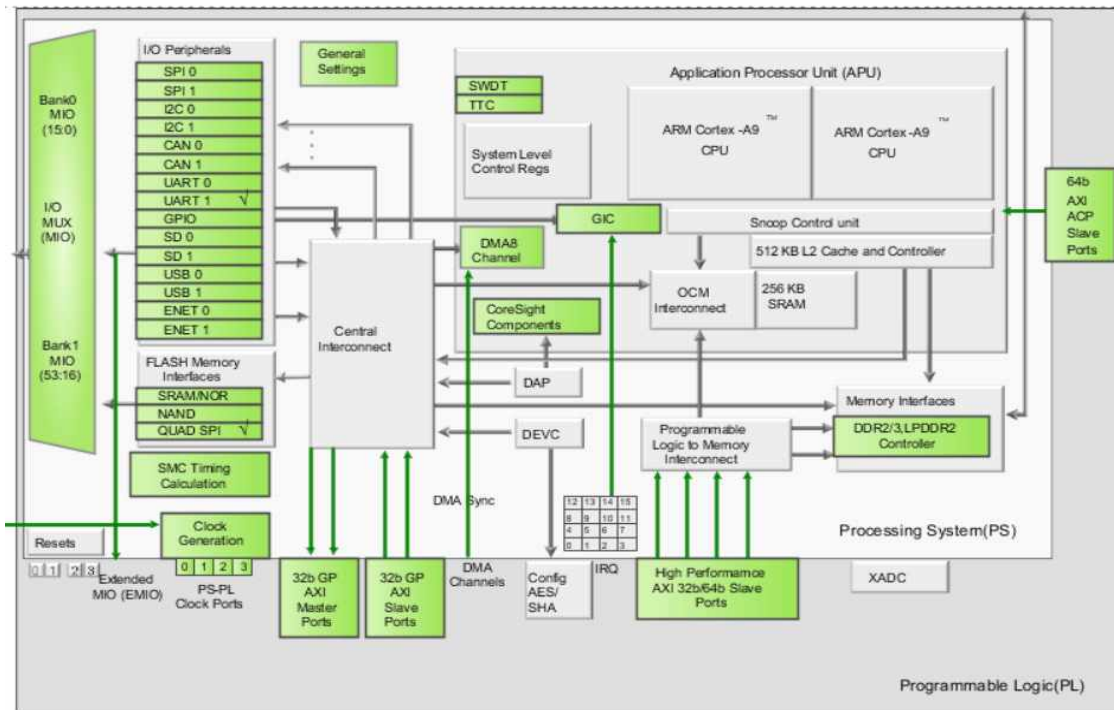
## 4. Discussion

### 1. 설계 방법 설명.

#### 1-1) 설계 조건.

ALU 설계. PL의 peripheral인 DIP Switch (OPCODE, STEPSIZE), 버튼(초기화)에서 입력을 받아 PS에서 ALU 연산을 수행한 후 PL의 peripheral인 LED로 출력한다.

Processing System IP와 AXI Interconnect IP, AXI GPIO IP를 이용해야 한다.

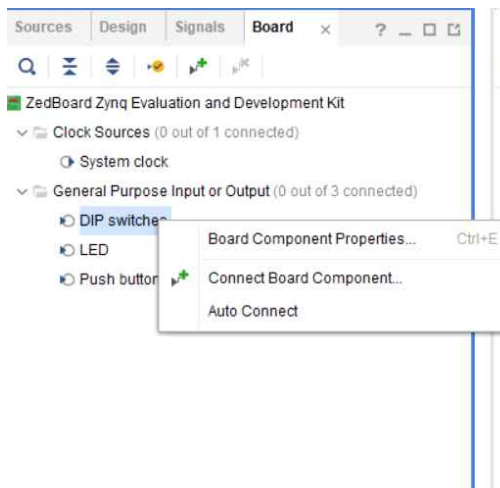


### 1-2) Zynq Block design

PS(Zynq block)에서 TERATERM 통신을 위한 UART1과 PS-PL간 통신을 위한 AXI(Advanced eXtensible Interface)를 활성화시켜 주었다.

### 1-3)PL peripheral 활성화.

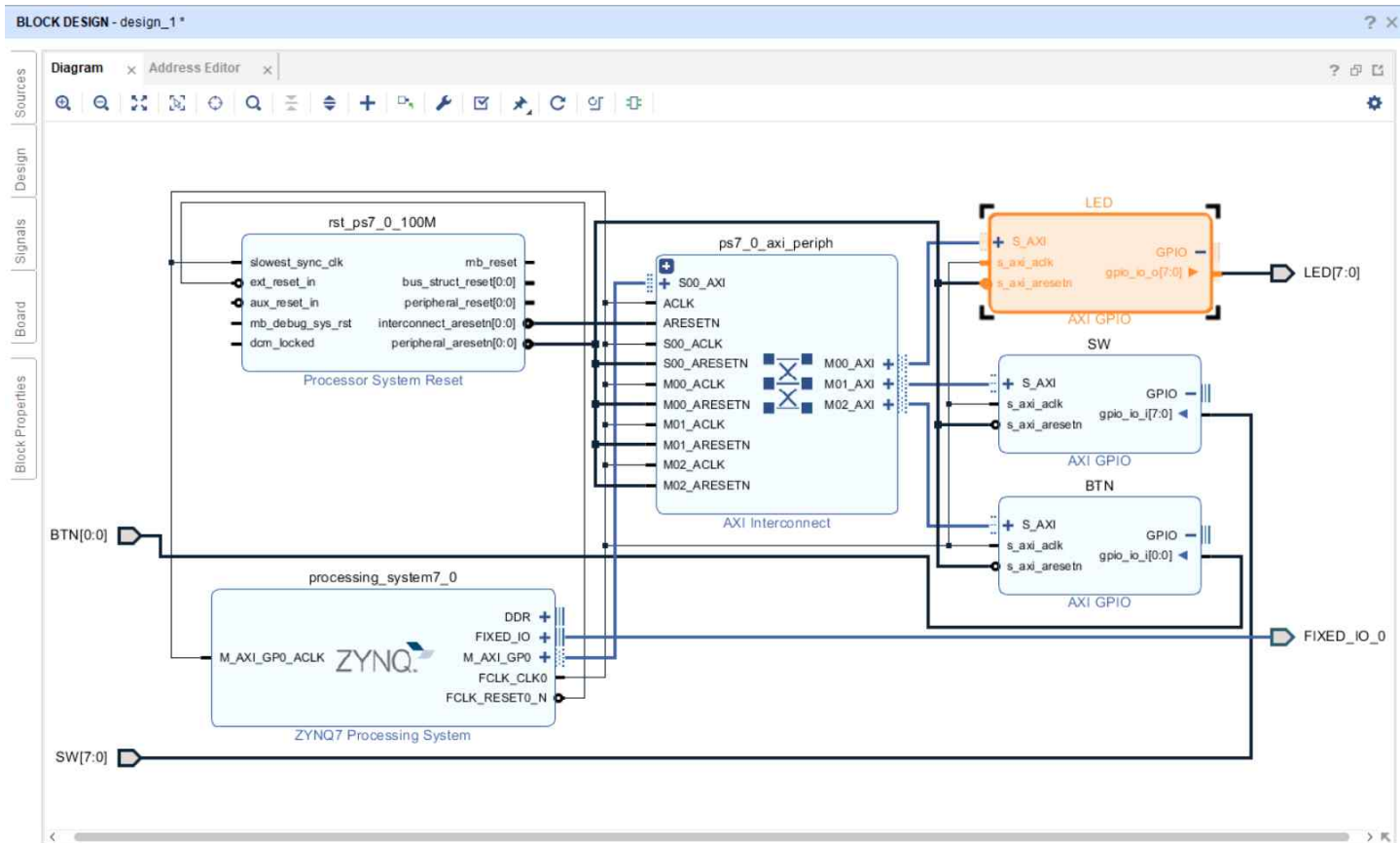
PL단에서 DIP switch와 버튼 LED의 블록을 생성한다. 생성하는 방식에는 두가지가 있다.



첫 번째 방법은 Board 탭에서 제공하는 GPIO목록에서 DIP switches, LED, Push button을 선택하고 Auto connect 혹은 Connect Board Component 버튼을 눌러 자동으로 AXI Interface를 생성하고 PS와 연결하는 방법이다. 이 방법을 사용할 때 xdc 파일에서 버튼의 width를 수정하고 이름을 포트이름과 일치 시켜주어야 한다.

두 번째 방법은 직접 axi gpio블록을 불러와 IP Configuration 탭에서 input, output과 width 비트폭을 조정한 후 port를 생성하여 LED, DIP SWITCH, BUTTON을 직접 만드는 방법이다. 이때 DIP switch와 Button 은 input으로 LED는 output임을 유의해야 한다.





Vivado의 connection Automation기능으로 PS와 AXI GPIO를 연결한 모습이다. 이후 블록 디자인을 Generate Output Products 기능을 사용해 PS 부분의 합성을 완료하고 Create HDL wrapper를 실행해 HDL TOP Module을 생성한다.

#### 1-4) constraints 생성

##### < User LED Pin-List >

Signal Name	Zynq Pin	I/O	Note
LED<1>	T22	Output	User LED [8..1] (Active 'H') Bank33
LED<2>	T21		
LED<3>	U22		
LED<4>	U21		
LED<5>	V22		
LED<6>	W22		
LED<7>	U19		
LED<8>	U14		

##### < User PUSH Switch Pin-List >

Signal Name	Zynq Pin	I/O	Note
<b>PUSH_SW&lt;1&gt;</b>	<b>T18</b>	Input	User PUSH Switch [5..1] (Active 'L') Bank34
PUSH_SW<2>	P16		
PUSH_SW<3>	N15		
PUSH_SW<4>	R18		
PUSH_SW<5>	R16		

```
set_propertyPACKAGE_PIN T22 [get_ports{LED[0]}]
set_propertyPACKAGE_PIN T21 [get_ports{LED[1]}]
```

```
set_propertyPACKAGE_PIN U22 [get_ports{LED[2]}]
set_propertyPACKAGE_PIN U21 [get_ports{LED[3]}]
set_propertyPACKAGE_PIN V22 [get_ports{LED[4]}]
set_propertyPACKAGE_PIN W22 [get_ports{LED[5]}]
set_propertyPACKAGE_PIN U19 [get_ports{LED[6]}]
set_propertyPACKAGE_PIN U14 [get_ports{LED[7]}]
```

```
set_propertyPACKAGE_PIN F22 [get_ports{SW[7]}]
set_propertyPACKAGE_PIN G22 [get_ports{SW[6]}]
set_propertyPACKAGE_PIN H22 [get_ports{SW[5]}]
set_propertyPACKAGE_PIN F21 [get_ports{SW[4]}]
set_propertyPACKAGE_PIN H19 [get_ports{SW[3]}]
set_propertyPACKAGE_PIN H18 [get_ports{SW[2]}]
set_propertyPACKAGE_PIN H17 [get_ports{SW[1]}]
set_propertyPACKAGE_PIN M15 [get_ports{SW[0]}]
```

```
set_propertyPACKAGE_PIN T18 [get_ports{BTN[0]}]
```

```
set_propertyIOSTANDARD LVCMOS25 [get_ports{LED[*]}]
set_propertyIOSTANDARD LVCMOS25 [get_ports{SW[*]}]
set_propertyIOSTANDARD LVCMOS25 [get_ports{BTN[0]}]
```

다음 표를 보고 Zynq pin을 매핑하고 포트의 핀과 연결한다.

```
set_propertyPACKAGE_PIN T22 [get_ports{LED[0]}]
```

다음 명령에서 get\_ports{LED[0]}는 “LED”라는 이름의 0번 pin을 의미하며 T22는 패키지 내의 물리적인 핀 이름이다. 다음 명령을 통해 led 0번 핀은 T22와 연결된다.

```
set_propertyIOSTANDARD LVCMOS25 [get_ports{LED[*]}]
```

다음 명령에서 get\_ports{LED[\*]}는 “LED”라는 이름의 모든 pin을 의미하며 해당 핀의 전기적 특성으로 LVCMOS25(2.5V)를 사용하겠다는 것을 나타낸다.

1-5)비트스트림 생성 후 sdk에서 c파일 작성.

constraints 파일 작성 후 비트스트림을 생성한후 export hardware한 후 SDK툴에서 C파일 작성을 시작한다. 코드 작성 시 system.bss 파일에서 예제 코드를 따왔으며 사용한 함수는 xgpio.h 헤더파일에서 찾아 사용하였다. XPAR\_SW\_DEVICE\_ID 같은 장치 주소값은 xparameter.h에서 찾아 사용하였다.

코드 알고리즘은 SDK 코드 설명란에 주석을 달아 설명하였으며 덧셈, 뺄셈, 곱셈 연산을 작성하였고 overflow, underflow를 검사하는 함수를 작성하여 언더 오버플로우를 방지하였다. 오른쪽, 왼쪽 시프트 연산을 구현하였으며 비트의 mst, lst에서 시프트되면 회전하는 rotate\_shift 연산이다. 추가적인 ALU기능 구현을 위하여 Bitwise and, or, xor 연산을 구현하였다.

2. 발생한 문제점과 해결방안.

2-1) LED 제어을 MIO 7번으로 해야 하는가?

Lab4 gpio ps 제어 예제에서는 SDK에서 LED를 MIO 핀맵 번호를 알아내어 제어하였다. 이번 프로젝트에서도 SDK 개발 도구에서 MIO 인터페이스를 이용하여 LED와 버튼, DIP 스위치등을 제어할 수 있지만 데이터 시트에서 MIO 번호를 알아내어 제어하는 것이 불편하고, Vivado Block 디자인을 사용하여 AXI GPIO와

Port 이름을 생성 후 constraints 파일을 작성하는 것이 더 직관적이다. 따라서 프로젝트 수행 전 MIO 핀 번호를 알아낸다. BLOCK 설계로 방향을 틀어 진행하게 되었다.

2-2) AXI GPIO블록 생성 시 LED, DIP Switch, Button 각각 3개의 블록을 생성할 까? 블록 하나에 두 개 이상의 포트를 만들까?

블록 하나에 두 개 이상의 포트를 연결하여 사용해도 프로젝트 수행에는 문제가 없지만, SDK에서 코드 작성시 직관성이 떨어지는 것 같다. 따라서 3개의 블록을 생성하여 각각 포트를 생성하였으며 블록의 이름도 GPIO\_0 등의 이름에서 LED등의 직관성 있는 이름으로 교체하였다. 그로 인한 이점은 다음과 같다.

xparameter.h

```
/* Canonical definitions for peripheral BTN */
#define XPAR_GPIO_0_BASEADDR 0x41220000
#define XPAR_GPIO_0_HIGHADDR 0x4122FFFF
#define XPAR_GPIO_0_DEVICE_ID XPAR_BTN_DEVICE_ID
#define XPAR_GPIO_0_INTERRUPT_PRESENT 0
#define XPAR_GPIO_0_IS_DUAL 0

/* Canonical definitions for peripheral LED */
#define XPAR_GPIO_1_BASEADDR 0x41200000
#define XPAR_GPIO_1_HIGHADDR 0x4120FFFF
#define XPAR_GPIO_1_DEVICE_ID XPAR_LED_DEVICE_ID
#define XPAR_GPIO_1_INTERRUPT_PRESENT 0
#define XPAR_GPIO_1_IS_DUAL 0

/* Canonical definitions for peripheral SW */
#define XPAR_GPIO_2_BASEADDR 0x41210000
#define XPAR_GPIO_2_HIGHADDR 0x4121FFFF
#define XPAR_GPIO_2_DEVICE_ID XPAR_SW_DEVICE_ID
#define XPAR_GPIO_2_INTERRUPT_PRESENT 0
#define XPAR_GPIO_2_IS_DUAL 0
```

LED, BTN, SW의 peripheral 정보가 더 직관적으로 변하였고 device id 이름이 xpar\_axi\_gpio\_0\_device 등의 이름에서 XPAR\_LED\_DEVICE\_ID같은 이름으로 뚜렷해졌다.

단점으로 채널을 하나만 사용하기 때문에 그만큼 공간과 성능이 낭비되지만 해당 프로젝트에서 요구하는 기능은 많지 않기에 이렇게 진행하기로 하였다.

2-3) ALU 연산시 overflow, underflow 문제

해당 프로젝트는 8bit unsigned integer를 사용하며 표현가능한 수의 범위는 0에서 255까지이며 음수 표현은 사용하지 않는다. 따라서 255를 넘어갈 때 overflow가 발생하며 음수가 되면 underflow문제가 발생하게 되지만 실제로 키트 구동 시 자동으로 255에서 +1을 더하면 되면 1으로 0에서 -1을 더하면 255가 된다. 따라서 사람이 오버플로우, 언더플로우를 인식할 수 없다. 따라서 C언어 라이브러리 함수 중 **add\_with\_overflow**와 **sub\_with\_underflow** 함수를 참조하여 오버플로우, 언더플로우 발생 시 경고를 내보내는 수식을 구성하였다.

2-4) print()함수에서는 %d, %u, %f등의 기능을 이용할 수 없다.

ALU연산 모니터링을 위하여 teraterm 어플리케이션을 이용하는데 print()에서는 %d, %u 등의 기능을 지원하지 않아 결과값을 확인할 수 없다. 따라서 xil\_printf()함수를 이용하면 결과값을 확인할 수 있다. xil\_printf()함수는 xil\_printf.h 헤더안에 정의되어 있으며, xgpio.h 헤더 안에 이미 선언되어 있으므로 따로 메인 c파일에서 헤더선언을 해줄 필요는 없다.

## 2-5) 연산 속도 문제

ALU연산 테스트 시 clk이 너무 빨라 값을 검증하기에 힘들다. 따라서 1초에 한번씩만 값이 바뀌도록 설정하였으며 sleep() 함수를 이용했다. sleep 함수는 n초동안 연산을 대기하게 하는 코드이며 sleep 함수는 unistd.h 안에 정의되어 있다.

## 2-6) PS에서 ALU연산이 올바른가?

ALU는 하드웨어적인 구성 요소이기 때문에 C언어를 사용하여 직접 ALU를 구현하는 것은 어렵다. 반면 Verilog는 하드웨어 설계언어이며 디지털 회로를 모델링하고 설계하는데 사용되기에 Verilog를 통하여 ALU를 구현하는 것이 적합하다. ALU는 이진수로 표현된 두 개의 입력 값에 대한 산술 및 논리 연산을 수행하는 하드웨어 장치로, Verilog의 연산자 및 모듈화 기능을 사용하여 구현하는 것이 적합하다.

따라서 Verilog를 사용하여 FPGA안에서 ALU를 구현하는 것이 적합하지만, 본 프로젝트는 PS와 PL를 연결하여 AXI라는 ARM의 AMBA 버스를 스펙을 이용해보고, 디바이스 및 모듈 간의 데이터 전송 인터페이스를 구현하는 것이 목적인 것 같기에 PS에서 ALU 연산을 구현하게 되었다.