

# REPORT

## 보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,  
성·균·인으로서 나의 명예를 지킬 것을 약속합니다.

과 목 : 논리회로설계실험

과 제 명 : 프로젝트 보고서 6조

담당교수 : 정 일 섭

학 년 : 4

이 름 : 황 온 유

제 출 일 : 2020.12.13

## 1. 설계주제

음식점이나 카페에서 쓰는 순번대기표 기기를 설계하였다. 이러한 순번 대기표는 순번표를 뽑아 자기의 번호를 인식하고 현재 대기 시간이 얼마나 되는지 확인할 수 있다. 앞의 순번이 모두 호출되면 직원의 호출에 의해 자기의 번호가 호출된다.

본 기기에서는 순번 대기표와 기능적으로 유사하게 설계하였다. 손님이 번호표를 뽑고 정해진 순서가 오면 호출되는 큰 기능은 비슷하지만 일괄적으로 부여된 번호 대신 손님의 전화번호 뒷 두 자리를 적는 것으로 종이/온라인 번호표의 기능을 대체한다.

기계에는 총 3가지 모드가 있는데, 기본 모드(Push1), 손님 모드(Push2), 직원 모드(Push3)이다.

기본 모드에서는 비특정 사용자가 현재 몇 명이 기다리고 있는지, 다음 호출번호(4개)는 몇 번인지, 시간은 얼마나 걸리는지를 확인할 수 있는 모드이다.

손님 모드에서는 번호표를 받는다. 정확히는, 자신의 전화번호 뒷 두자리를 입력함으로써 순위표에 들어갈 수 있다. 또한, 지금 자신이 번호표를 받는 시점에서의 대기인원, 대기시간 또한 알 수 있다. 전화번호 뒷 두자리를 입력할 때에는 DIP 스위치를 이용한다.

직원 모드에서는 직원이 손님을 호출한다. 30초간 기다렸다가 나타나지 않으면 바로 다음 사람을 호출한다. 만약 나타난다면 push\_sw2 버튼을 누르고 push1 기본 모드로 돌아가서 대기하면 된다.

호출번호, 기다리는 명 수, 입력 번호의 정보는 LCD를 이용하여 출력하였고, 대기 시간과 호출시간 카운트는 7-SEGMENT를 이용하여 출력하였다.

## 1. VHDL Sources 및 설명

-> *Background :*

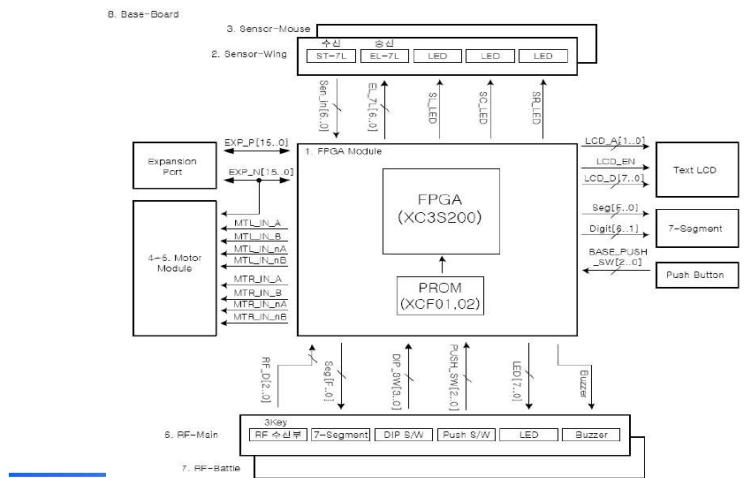
- Rov-Lab 3000 FPGA 기본 정보

Rov-Lab 3000은 FPGA디지안을 증명하기 위한 machine이다.

Input of Rov-Lab : Push switch(3), Dip switch(4 bit), reset switch

Display of Rov-Lab : character LCD, 7segment, LED

Base Board Block Diagram



## - Component

일반적인 프로그래밍을 할 때도, 반복되는 부분을 재사용함으로써 전체 코드를 간결하게 구조적으로 작성하기 위해 '함수'를 활용한다. VHDL에서도 마찬가지로 이러한 기능을 비슷하게 구현할 수 있다. 바로 'component'와 'port map'이다. 'component'는 '구성요소'라는 뜻처럼 이전에 작성한 VHDL모듈을 하나의 작은 단위로써 활용하는 방법이다. 이전에 작성된 component를 활용하기 위해서, 상위 설계 모듈에서 이를 호출해야 할 필요가 있다. 호출하기 위해 port map을 사용해야 하며, 이때 주의할 점은 원래 component의 입력과 출력을 같은 유형으로 일치시켜야 한다는 점이다.

```
-- 입력 ci는 연산 종류를 결정하는 m 입력과의 exclusive-or 연산을 하게 된다.
-- 내부 신호 c0는 위의 연산 결과에 따라 carry(가산), borrow(내림)이 된다.
-- 입력 신호 m을 처리한다는 점에서 위의 c1, c2, c3와 구분되어 별도 선언을 하였으나,
-- c1, c2, c3와 같이 instantiation에서 carry의 역할을 한다는 점에서 변수명 c0을 사용했다.

-- STD_LOGIC_VECTOR형 신호 b는 c0와 유사하게 연산에 따라 피연산자의 각 비트를 반전시키고 있
begin
    c0 <= m xor ci;

    -- 먼저 연산의 종류를 결정하는 입력과 carry in의 연산 결과에 따라
    -- ci는 Full_Adder 연산의 첫번째 피연산자인 c0가 된다.

    b(0) <= y(0) xor m;
    b(1) <= y(1) xor m;
    b(2) <= y(2) xor m;
    b(3) <= y(3) xor m;

    -- 가산연산인지 감산연산인지에 따라 피연산자 벡터 y의 각 비트는 반전된다.
    -- 이전의 ci와 m과의 연산 결과에 따라 음수로 y의 각 비트가 반전된 후에도
    -- 추가적인 1이 더해지게 되므로, 2's complement로서 동작하게 된다.

    f_adder0: Full_Adder port map (x(0), b(0), c0, s(0), c1);
    f_adder1: Full_Adder port map (x(1), b(1), c1, s(1), c2);
    f_adder2: Full_Adder port map (x(2), b(2), c2, s(2), c3);
    f_adder3: Full_Adder port map (x(3), b(3), c3, s(3), c0);

    -- 가산인지 연산인지는 이전의 연산들에 의해 처리된 것으로, 더하기를 통해
    -- 최종적으로 연산을 완료한다. 감산연산은 '음수를 더한다'는 개념으로
    -- 처리되고 있다.

end Behavioral;
```

### - Conditional statement

0과 1로 표현되는 디지털 논리회로에 의해 Conditional statement의 사용이 가능하다, 예를 들어 case ~is when 구문이나 if ~ then ~ else 구문, when ~ else 구문 등 많은 구문을 사용하여 data값을 할당할 수 있을 것이다

```
case data is
    when "0000" => --number 0
        seg <= "00111111";
    when "0001" => --number 1
        seg <= "00000110";
    when "0010" => --number 2
        seg <= "01011011";
    when "0011" => --number 3
        seg <= "01001111";
    when "0100" => --number 4
        seg <= "01100110";
    when "0101" => --number 5
        seg <= "01101101";
    when "0110" => --number 6
        seg <= "01111101";
    when "0111" => --number 7
        seg <= "00000111";
    when "1000" => --number 8
        seg <= "01111111";
    when "1001" => --number 9
        seg <= "01100111";
    when "1010" => --number 10 : 빈칸
        seg <= "00000000";
    when others => null;
end case;

process(FPGA_RSTB, CLK)
begin
    if (FPGA_RSTB = '0') then
        for i in 0 to 31 loop
            ram(i) <= "00000000"; -- 모두 0 삽입
        end loop;
    elsif (CLK='1' and CLK'event) then
        ram(conv_integer(cnt2)) <= numwhole;
    end if;
end process;
```

<case when구문>

<if then else 구문>

### - Process

실제로 설계되어야 하는 논리회로는 입력이 결정되면 바로 출력이 결정되는 형태의 '계산기'의 기능을 하는 조합논리 뿐만 아니라, 논리회로가 스스로 어떤 일을 수행할 수 있는 '유용한 장치'의 기능을 하게 하는 순차논리 역시 필요하다. 순차논리를 구현하기 위해 계속해서 논리회로가 특정한 상태를 가지고 있어야 하며, 이렇게 '특정한 상태'를 가지고 있는 메모리의 대표적인 소자로 D Flip-Flop이 있으며, 이를 사용해서 데이터를 저장하기 위해서는 지속적인 clock signal이 필요하다. VHDL 상에서, 이러한 소자들의 구동을 위해, process를 이용하여, clock이 들어오는 상황을 가정할 수 있고, 위의 conditional statement와 연계해서, Behavioral Model로 순차논리를 작성할 수 있게 된다.

```
process(FPGA_RSTB, CLK)
begin
    if (FPGA_RSTB = '0') then
        cnt <= (others => '0');
        data_out <= '0';
    elsif (CLK='1' and CLK'event) then
        if (w_enable = '1') then
            data <= reg_file(conv_integer(cnt));
            addr <= cnt;
            data_out <= '1';
            if cnt = X"1F" then -- 31
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        else
            data_out <= '0';
        end if;
    end if;
end process;
```

### - FSM : Finite State Machine

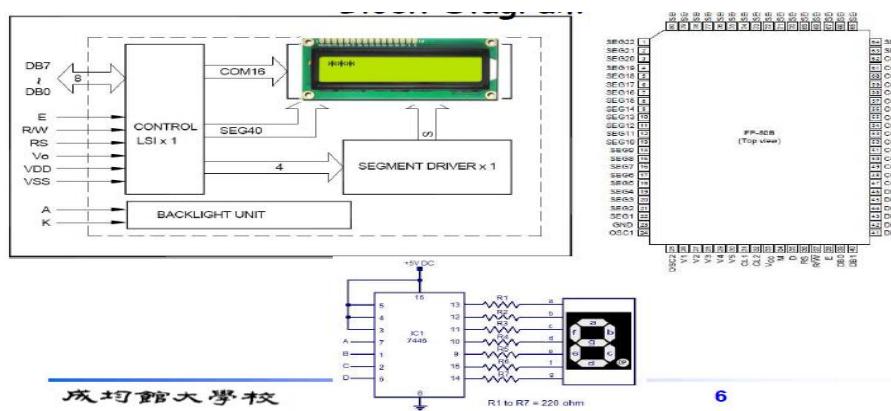
위에서 언급한, 특정한 입력을 받게 되면, 제한된 형태의 '스스로 동작하는' 유용한 장치를 구현을

위해 'FSM'에 대한 지식이 필요하다. FSM을 설계하기 위해서는 2)와 3)의 conditional statement와 process가 필요하다. FSM은 '상태'라는 정보를 저장하기 위해 메모리 소자가 필요하고, 이를 구동하기 위한 지속적인 clock signal을 공급해야 한다. Clock signal이 공급되면, 그 FSM이 '어떻게 동작하는지'를 설계해야 하는데, 이는 2) conditional statement의 '특정한 조건 하에서'를 '특정한 상태에서'로 해석하여 설계할 수 있다. State machine을 설계하기 위해, 모든 상태에서 모든 천이를 명시해야 하고, 이는 conditional statement를 여러 번 사용하여 구현된다. 본 코드에서는 LCD display시 각 instruction을 자동으로 다음으로 넘어가게 하는 곳에 사용한다. 즉, 상태 X"01"이 있으면 다음 rising clock 시 상태 X"02"로 넘어가는 식으로 작동한다.

### -Ram Infrastructure (Random Access Memory)

Final Project로 FSM을 설계하기 위한 배경지식은 위의 내용이 주를 이루지만, 실제로 우리가 설계할 Final Project는 논리회로 설계 프로그램인 Xilinx Design Suite 상에서만 동작하는 것이 아니라, 실제 하드웨어 상에서 동작하는지 검증을 마쳐야 한다. 프로그램 상에서의 설계를 이용해서 RoV-3000이라는 하드웨어의 각 부분을 동작 시켜야 한다. 하드웨어가 동작하는 원리는 특정 위치에서 1(혹은 0)이면 동작하고, 0(혹은 1)이면 동작하지 않는 것으로, 프로그램을 이용해서 지속적으로 하드웨어를 동작하기 위한 위치에 0과 1을 쓰고 지워야 한다. 다만, 계속해서 컴퓨터로 RAM에 데이터를 쓰고 지우는 것이 아닌, 한 번 프로그래밍한 후에는 별도의 추가 프로그래밍 없이도 동작해야 하므로, 이러한 RAM의 구조를 이용하여 FSM이 특정 상태에서 RAM의 데이터를 자동으로 읽거나 쓰도록 해야 한다. 이번 최종 프로젝트에서 각각 손님의 순서에 대해 손님의 번호를 ram에 저장되도록 하여 호출 시나 중간 계산시 값을 불러올 수 있도록 설계하였다.

### -LCD



LCD는 총 32자리의 문자나 숫자의 표시구간이 있으며, 6개의 Instruction code와 1개의 줄 바꿈으로 총 39가지의 입력 값을 설정하여야 한다.

### LCD Instruction Sets 부분

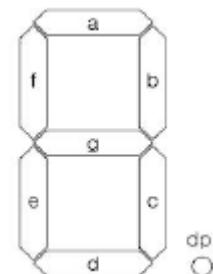
RS RW DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 으로 구성되며 RS RW 는 read 되거나 write 됨에 따라 "00","01","10","11" 로 구성이 된다. DB7 부터 DB0 까지는 각각 Initialize LCD 를 보여준다.

Instruction	Code											Bit name	Setting/Status
	RS	RW	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1			
Cursor home	0	0	0	0	0	0	0	0	1	-		I/D	0=dec cursor pos 1=inc cursor pos
Entry mode set	0	0	0	0	0	0	0	1	ID	S		S	0=No display shift 1=display shift
Display ON/OFF	0	0	0	0	0	0	1	D	C	B		D	0=display off 1=display on
Cursor display shift	0	0	0	0	0	1	S/C	R/L	-	-		C	0=Cursor off 1=Cursor On
Function set	0	0	0	0	1	DL	N	F	-	-		B	0=Cursor blink off 1=Cursor blink On
Set CGRAM address	0	0	0	1	CGRAM ADDRESS							S/C	0=Move cursor 1=Shift display
Set DDRAM address	0	0	1	DDRAM ADDRESS								R/L	0=Shift left 1=Shift right
Read busy flag and address counter	0	1	BF	CGRAM/DDRAM ADDRESS								DL	0=4bit interface 1=8bit interface
Write to CGRAM or DDRAM	1	0	WRITE ADDRESS									N	0=1 line 2=2 line
Read from DGRAM or DDRAM	1	1	READ ADDRESS									F	0=5x8 dots 2=5x10 dots
												BF	0=can accept inst 1= int op in prog

예를 들면 data length 가 8bit 이고 2 line-mode이며 display off 일 경우 Function set 은 "00001110\*\*"를 따르며 Display OFF 가 display off, cursor off, blink off 일 경우 "00001000"를 따르고, Entry mode set 에서 Increment mode, Entire shift off 일 경우 "00000110"을 만족시킨다. 이를 이용하여 read의 경우 Display OFF 는 [RS RW DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0] = [0 0 0 0 0 1 0 0 0]으로 LCD Instruction code 를 만들 수 있다. 본 코드에서는 LSB 2bit 는 LCD\_A 라는 다른 Vector 로 꺼내어 읽기, 쓰기를 조절하고 나머지 8 bit 는 instruction 이나 데이터를 전달하는 식으로 나누어 쓰인다.

### -7-Segment

지금까지의 지식을 활용해서 VHDL을 이용한 프로그래밍으로 논리회로 설계를 작성한 이후에는 구체적으로 하드웨어의 작동원리를 이해할 필요가 있다. 7-Segment LED는 5)에서 언급한 것처럼 특정한 위치에 0과 1을 쓰으므로써 동작하는데, 표현할 decimal number에 따라 다르다. 6자리의 display구간이 있으며 한자리마다 7개의 부분으로 나누어져 조건마다 각각 불빛이 들어와 값을 표시하도록 할 수 있다. 한 번에 6개의 숫자를 표시할 수 없다는 특징으로 인해 본 코드에서는 100us의 주기로 첫번째 숫자부터 마지막 숫자까지 순차적으로 표시한다.



## -> Design theory

편의를 위해 다음과 같이 스위치 이름 설정

Pin번호	스위치이름 (VHDL 코드 상 이름)	Pin번호	스위치이름 (VHDL 코드 상 이름)
P102	PUSH1 (load_b1)	P18	PUSH_SW0 (load_s1)
P101	PUSH2 (load_b2)	P19	PUSH_SW1 (load_s2)
P100	PUSH3 (load_b3)	P20	PUSH_SW2 (load_s3)

-Rov-3000의 Pin번호 P100, P101, P102의 PUSH버튼 3개를 이용하여 모드를 3가지로 조절한다.

그리고 손님의 번호를 Dip 스위치를 이용하여 4비트의 숫자를 두 번 받아서 두 자리의 숫자로 표현한다. Pin번호 P18, P19의 PUSH스위치는 각각 Dip스위치로부터 받은 번호를 첫째 자리와 둘째 자리를 각각 구별하기 위해 사용하고, P20의 PUSH3 버튼은 모드3에서 손님의 등장 여부를 확인하기 위해 사용된다.

### -Dip 스위치와 PUSH\_SW0 & PUSH\_SW1을 이용한 손님 번호 설정

Dip스위치로 4비트의 수를 입력한 후 PUSH\_SW0을 눌러 손님 번호의 첫째 자리를 받고 다시 Dip스위치로 4비트의 수를 입력한 후 PUSH\_SW1을 눌러 손님 번호의 둘째 자리를 받아 십진수 두 자리의 손님번호를 ram에 저장하도록 할 수 있다. 저장된 ram에서 손님의 번호를 호출할 때나 화면에 표시할 때 사용될 것이다.

### -cnt2와 cnt3을 이용한 입력과 호출

Ram에 저장되어 있는 data를 cnt2와 cnt3에 의해서 불러온다. cnt2를 이용하여 현재 번호를 입력하는 손님의 대기순서를 저장하고 cnt3을 이용하여 호출되는 손님의 순서를 알 수 있다. 아직 호출되지 않은 손님의 수를 cnt2 – cnt3 연산을 통해 알 수 있고, 총 대기시간또한 미호출 손님에 10분씩 곱하여 일괄적으로 구하였다.

### -mode1의 동작 (PUSH1)

PUSH1버튼을 누르게 되면 마지막 호출 받은 손님 이후 4명의 손님의 번호(N1, N2, N3, N4)를 LCD의 첫째 줄에 나타낼 것이며 총 대기중인 손님의 수(XX)를 LCD의 둘째 줄에 표시할 것이다.

<	N	1	>	<	N	2	>	<	N	3	>	<	N	4	>
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

T	O	T	A	L		N	U	M	B	E	R	:	X	X
---	---	---	---	---	--	---	---	---	---	---	---	---	---	---

그리고 7-Segment에는 앞으로 호출 받을 때까지 기다려야 하는 시간을 나타낼 것이다. 한 손님당 10분의 업무 기간으로 설정하고 6자리에 대기 시간이 출력된다. 해당 대기시간은 특정인에 대한 대기 시간이 아니라 현재 입장하는(또는 할 예정인) 사람의 대기시간(비특정인)이다. 즉, cnt2-cnt3를 한 값에 10분을 곱한 시간이다.

0	대기시간 시간 일의자리	대기시간 분의 십의자리	대기시간 분의일의자리	0	0
---	-----------------	-----------------	----------------	---	---

### -mode2의 동작(PUSH2)

PUSH2 버튼을 클릭하게 되면 mode2가 동작할 것이다. Mode2에서 LCD의 첫째 줄은 현재 입력된 손님의 번호를 ram값을 이용하여 출력할 것이다. 둘째 줄은 호출되고 있는 손님의 대기번호와 현재 입력된 손님의 대기번호의 차이를 이용하여 앞에 얼마나 많은 사람들이 대기하고 있는지를 나타낼 것이다.

Y	O	U	R	N	U	M	B	E	R	:	X1	X2		
W1	W2		P	E	O	P	L	E		A	H	E	A	D

그리고 7-Segment에는 mode1과 비슷하게 앞으로 호출 받을 때까지 기다려야 하는 시간을 나타낼 것이다. 한 손님당 10분의 업무 기간으로 설정하고 6자리에 대기 시간이 출력된다. 이때, 출력되는 시간은 현재 입력하고 있는 손님기준이다. 즉, cnt2-cnt3 데이터를 사용한다.

0	대기시간 시간 일의자리	대기시간 분의 십의자리	대기시간 분의일의자리	0	0
---	-----------------	-----------------	----------------	---	---

### -mode3의 동작(PUSH3)

PUSH3 스위치를 누르게 되면 mode3이 동작하게 될 것이다. Mode3에서는 손님의 호출번호를 현재 cnt3를 이용하여 호출한다. 그리고 30초 Countdown이 완료되기 전 손님의 등장 여부에 따라 PUSH\_SW2를 눌러 손님의 등장여부를 LCD에 표시할 것이다. 일단 mode3이 동작하게 되면 현재 호출되는 대기순위번호 cnt3손님의 번호(C1, C2)를 ram에 저장되어 있는 data를 통해 부를 것이다. 30초 안에 등장을 하지 않는다면 cnt3 <= cnt 3 + 1 연산이 진행되어 C1과 C2는 그 다음 번호로 자동으로 넘어간다.

C	A	L	L	I	N	G	:	C1	C2					

그리고 7-Segment에서 30초를 카운트할 것이다

0	0	0	0	초의 십의자리	초의 일의자리
---	---	---	---	------------	------------

손님이 30초 안에 등장을 할 경우 PUSH\_SW2버튼을 클릭하여 손님이 등장했다는 표시를 하기 위해  
해서 LCD에 ENTERED를 띄우게 될 것이다.

E	N	T	E	R	E	D								

### -사용할 Pin 번호

PORT	FPGA pin	PORT	FPGA pin
DIGIT1	P22	SEG_A	P31
DIGIT2	P24	SEG_B	P33
DIGIT3	P26	SEG_C	P34
DIGIT4	P27	SEG_D	P35
DIGIT5	P28	SEG_E	P36
DIGIT6	P29	SEG_F	P37
LCD_A[1]	P51	SEG_G	P39
LCD_A[0]	P50	SEG_DP	P48
CLK	P79	number[3]	P12
FPGA_RSTB	P205	number[2]	P13
LCD_EN	P52	number[1]	P15
LCD_D[7]	P67	number[0]	P16
LCD_D[6]	P65	load_b1	P102
LCD_D[5]	P64	load_b2	P101
LCD_D[4]	P63	load_b3	P100
LCD_D[3]	P62	load_s1	P18
LCD_D[2]	P61	load_s2	P19
LCD_D[1]	P58	load_s3	P20
LCD_D[0]	P57		

### → Sources code 와 코드 설명

#### 1) Waitinglist (Top module)

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity waitinglist is
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           LCD_A : out STD_LOGIC_VECTOR (1 downto 0);
           LCD_EN : out STD_LOGIC;
           LCD_D : out STD_LOGIC_VECTOR (7 downto 0);
           DIGIT : out STD_LOGIC_VECTOR (6 downto 1);-- 숫자 패널 이름
           SEG_A : out STD_LOGIC;
           SEG_B : out STD_LOGIC;
           SEG_C : out STD_LOGIC;
           SEG_D : out STD_LOGIC;
           SEG_E : out STD_LOGIC;
           SEG_F : out STD_LOGIC;
           SEG_G : out STD_LOGIC;
           SEG_DP : out STD_LOGIC;
           load_b1 : in STD_LOGIC;
           load_b2 : in STD_LOGIC;
           load_b3 : in STD_LOGIC;
           load_s1: in STD_LOGIC;
           load_s2 : in STD_LOGIC;
           load_s3 : in STD_LOGIC;
           number : in STD_LOGIC_VECTOR (3 downto 0));-- operand
end waitinglist;
architecture Behavioral of waitinglist is

component data
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           addr2 : out STD_LOGIC_VECTOR (4 downto 0); --mode 2에서 온 손님 번호
           addr3 : out STD_LOGIC_VECTOR (4 downto 0); -- mode 3에서 온 호출 번호
           data0, data1, data2, data3, data4 : out STD_LOGIC_VECTOR (7 downto 0);-- next 4개 번호
           mydata : out STD_LOGIC_VECTOR (7 downto 0);
           mode : out STD_LOGIC_VECTOR (2 downto 0);-- 이 모듈에서 모드 인식 후 다른 모듈로 전달.
           load_b1 : in STD_LOGIC; -- b1 switch, 모드1
           load_b2 : in STD_LOGIC; -- b2 switch, 모드2
           load_b3 : in STD_LOGIC; -- b3 switch, 모드3
           load_s1 : in STD_LOGIC; -- s1 switch, 첫째자리 넣을 때
           load_s2 : in STD_LOGIC; -- s2 switch, 둘째자리 넣을 때
           load_s3 : in STD_LOGIC; -- s3 switch, 해당 번호 도착 의미
           number : in STD_LOGIC_VECTOR (3 downto 0); --operand
           push_ent : in STD_LOGIC);
end component;

component mode
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           addr2, addr3 : in STD_LOGIC_VECTOR (4 downto 0); -- data0에서 가져옴
           data0, data1, data2, data3, data4 : in STD_LOGIC_VECTOR (7 downto 0);-- data에서 가져옴
           mydata : in STD_LOGIC_VECTOR (7 downto 0);--data에서 가져옴, 지금 데이터
           mode : in STD_LOGIC_VECTOR (2 downto 0); -- data에서 가져옴, big switch 번호
           w_enable : in STD_LOGIC; --lcd display에서 들어옴
           data_out : out STD_LOGIC; -- lcd display로 들어감. 데이터가 나갔는지 여부만 확인
           addr : out STD_LOGIC_VECTOR (4 downto 0); -- lcd display로 내보냄. 레지스터 번호
           data : out STD_LOGIC_VECTOR (7 downto 0)); -- lcd display로 내보냄. 해당 레지스터에 들어갈 값
end component;

component LCD_display
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           LCD_A : out STD_LOGIC_VECTOR (1 downto 0);
           LCD_EN : out STD_LOGIC;
           LCD_D : out STD_LOGIC_VECTOR (7 downto 0);
           data_out : in STD_LOGIC; -- mode에서 들어옴 값이 들어왔나만 표시
           addr : in STD_LOGIC_VECTOR (4 downto 0); -- mode에서 들어옴, 해당 디스플레이 위치 표시
           data : in STD_LOGIC_VECTOR (7 downto 0); -- mode에서 들어옴, 해당 디스플레이에 들어갈 코드 표시
           w_enable : out STD_LOGIC); --1이면 쓰고 있는거, 0이면 안 쓰는 거
end component;

```

```

component clock
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           DIGIT : out STD_LOGIC_VECTOR (6 downto 1);-- 숫자 패널 이름
           SEG_A : out STD_LOGIC;
           SEG_B : out STD_LOGIC;
           SEG_C : out STD_LOGIC;
           SEG_D : out STD_LOGIC;
           SEG_E : out STD_LOGIC;
           SEG_F : out STD_LOGIC;
           SEG_G : out STD_LOGIC;
           SEG_DP : out STD_LOGIC;
           addr2 : in STD_LOGIC_VECTOR (4 downto 0); --mode 2에서 온 손님 번호
           addr3 : in STD_LOGIC_VECTOR (4 downto 0); -- mode 3에서 온 호출 번호
           mode : in STD_LOGIC_VECTOR(2 downto 0);
           push_ent : out STD_LOGIC);
    end component;

signal data_reg, data0_reg, data1_reg, data2_reg, data3_reg, data4_reg : std_logic_vector (7 downto 0);
signal mode_reg : std_logic_vector (2 downto 0);
signal addr2_reg, addr3_reg : std_logic_vector (4 downto 0);
signal data_out_reg, w_enable_reg: std_logic;
signal addr_reg : std_logic_vector (4 downto 0);
signal mydata_reg : std_logic_vector(7 downto 0);
signal push_ent_reg : std_logic;

begin
    dataport : data port map(FPGA_RSTB, CLK, addr2_reg, addr3_reg, data0_reg,
                           data1_reg, data2_reg, data3_reg, data4_reg, mydata_reg, mode_reg,
                           load_b1, load_b2, load_b3, load_s1, load_s2, load_s3, number, push_ent_reg);

    modeport : mode port map(FPGA_RSTB, CLK, addr2_reg, addr3_reg, data0_reg, data1_reg,
                           data2_reg, data3_reg, data4_reg, mydata_reg, mode_reg,
                           w_enable_reg, data_out_reg, addr_reg, data_reg);

    LCD_displayport : LCD_display port map(FPGA_RSTB, CLK ,LCD_A, LCD_EN, LCD_D,
                                         data_out_reg, addr_reg, data_reg, w_enable_reg);

    clockport : clock port map(FPGA_RSTB, CLK, DIGIT, SEG_A, SEG_B, SEG_C, SEG_D, SEG_E,
                               SEG_F, SEG_G, SEG_DP, addr2_reg, addr3_reg, mode_reg, push_ent_reg);

end Behavioral;

```

Waitinglist 모듈은 top 모듈로써 아래에서 곧 살펴볼 data, mode, LCD\_display, clock 모듈을 component를 이용하여 함수 값을 불러오고 이 함수에 의해 입력 값에 대해 출력 값이 나오도록 하는 역할을 한다. 입력을 하는 부분은 Pin 번호를 이용하여 설정을 하며 출력을 하는 부분은 LCD 와 7-Segment 를 이용하여 출력을 할 것이다.

## 2) Data

VHDL 특성상 clock rising 을 썼다면 다른 신호에 대해 clock 관련 코드를 사용을 할 수 없다. 하지만 push 버튼을 clock 시간에 비해 길게 누르고 있다면 다중 신호가 입력될 수 있는 문제의 소지가 있기 때문에 해당 문제를 pretempcnt2, pretempcnt3, prepush\_ent 를 이용하여 우회적으로 해결하였다.

코드 주석 대분류 1 번에서 push 2 버튼을 누르면 tempcnt2 = 1 을 할당하고, push\_sw1(load\_s2) 버튼을 누르면 0 을 할당한다. 대분류 2 번에서는 tempcnt2 값을 pretempcnt2 에 할당하고, 둘의 값이 다르고 tempcnt2 = 1 일 때 cnt2 가 +1 하게 설계하였다.

즉, 기본 설정값이 0인 tempcnt2 가 push2 를 누르는 순간 1 이 되고 그때 한 명이 더 들어왔다고 카운트를 하는 구조이다. 그리고 사람 번호를 마지막으로 입력하는 순간(push\_sw1) 다시 0 으로 돌려보내기 때문에 다음 push 2 를 누르기 전까지는 카운트가 안되는 것이다.

해당 방법으로 push 버튼 누르는 시간으로 인한 중복 카운팅을 방지할 수 있었다. 동일한 방법을 pretempcnt2, pretempcnt3, prepush\_ent 에 사용하였다. pretempcnt3 와 tempcnt3 는 호출한 사람 수(cnt3)를 push3 버튼을 누를 때 1씩 증가하는데 사용하였고, prepush\_ent 는 호출한 사람이 30 초 안에 오지 않았을 경우 그 또한 cnt3 를 증가하는데 사용되었다.



```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity data is
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           addr2 : out STD_LOGIC_VECTOR (4 downto 0); -- mode 2에서 온 손님 번호
           addr3 : out STD_LOGIC_VECTOR (4 downto 0); -- mode 3에서 온 호출 번호
           data0, data1, data2, data3, data4 : out STD_LOGIC_VECTOR (7 downto 0);-- next 4개 번호
           mydata : out STD_LOGIC_VECTOR (7 downto 0);
           mode : out STD_LOGIC_VECTOR (2 downto 0);-- 이 모듈에서 모드 인식 후 다른 모듈로 전달.
           load_b1 : in STD_LOGIC; -- b1 switch, 모드1
           load_b2 : in STD_LOGIC; -- b2 switch, 모드2
           load_b3 : in STD_LOGIC; -- b3 switch, 모드3
           load_s1 : in STD_LOGIC; -- s1 switch, 첫째자리 넣을 때
           load_s2 : in STD_LOGIC; -- s2 switch, 둘째자리 넣을 때
           load_s3 : in STD_LOGIC; -- s3 switch, 해당 번호 도착 의미
           number : in STD_LOGIC_VECTOR (3 downto 0); -- operand
           push_ent : in STD_LOGIC);
end data;

architecture Behavioral of data is

type nums is array ( 0 to 31) of std_logic_vector(7 downto 0);
signal ram: nums;
-- 일부러 ram(0) 0으로 설정하고 비워놓았다. 만약에 cnt30이 31일 경우 다음 것이 존재하지 않으므로 그 경우에는 --ram(0)에 저장된 0값을 가져오도록 하였다.

signal numwhole : std_logic_vector (7 downto 0);-- 실제 2자리 번호
signal cnt2, cnt3 : std_logic_vector(4 downto 0); -- 카운트
signal temp_mode : std_logic_vector(2 downto 0); -- mode용 중간 시그널
signal cnt3_1, cnt3_2, cnt3_3, cnt3_4 : std_logic_vector(4 downto 0);
signal pretempcnt2, tempcnt2, pretempcnt3, tempcnt3, prepush_ent : std_logic;-- 중복 카운트 막기 위한 조치

```

```

begin

    addr2 <= cnt2;
    addr3 <= cnt3;
    mode <= temp_mode;
    mydata <= ram(conv_integer(cnt2));-- 현재 내가 입력하고 있는 데이터

    cnt3_1 <= "00000" when (cnt3 = "11111") else (cnt3 + 1);
    cnt3_2 <= "00000" when (cnt3 = "11110") else (cnt3 + 2);
    cnt3_3 <= "00000" when (cnt3 = "11101") else (cnt3 + 3);
    cnt3_4 <= "00000" when (cnt3 = "11100") else (cnt3 + 4);

    data0 <= ram(conv_integer(cnt3));
    data1 <= ram(conv_integer(cnt3_1));
    data2 <= ram(conv_integer(cnt3_2));
    data3 <= ram(conv_integer(cnt3_3));
    data4 <= ram(conv_integer(cnt3_4));

```

---

--1. 번호 수집 및 모드 설정

```

process(FPGA_RSTB, CLK)
begin
    if FPGA_RSTB = '0' then
        numwhole <= (others => '0');
        temp_mode <= "000";
        tempcnt2<='0';
        tempcnt3<='0';
    elsif (CLK='1' and CLK'event) then
        if load_s1 = '0' then
            numwhole(7 downto 4) <= number;

        elsif load_s2 = '0' then
            numwhole(3 downto 0) <= number;
            tempcnt2 <= '0';

        elsif load_s3 = '0' then
            tempcnt3 <= '0';
            temp_mode <= "100";

        elsif load_b1 = '0' then
            temp_mode <= "001";

        elsif load_b2 = '0' then
            temp_mode <= "010";
            tempcnt2 <= '1';

        elsif load_b3 = '0' then
            temp_mode <= "011";
            tempcnt3 <= '1';

        end if;
    end if;
end process;

```

```

--2. cnt2, cnt3 값 지정 : 각 버튼을 눌렀을 때만 증가

process(FPGA_RSTB, CLK)
begin
    if FPGA_RSTB = '0' then
        cnt2 <= (others => '0');
        cnt3 <= (others => '0');

    elsif (CLK='1' and CLK'event) then
        pretempcnt2 <= tempcnt2;
        pretempcnt3 <= tempcnt3;
        prepush_ent <= push_ent;

        if (pretempcnt2 /= tempcnt2 and tempcnt2 = '1') then
            -- 전 입력과 다르고, mode가 2번일 때
            if cnt2 > "1111" then
                cnt2 <= "00000";
            else
                cnt2 <= cnt2 + 1;
            end if;

        elsif (pretempcnt3 /= tempcnt3 and tempcnt3 = '1') or (prepush_ent /= push_ent and push_ent = '1') then
            -- 전 입력과 다르고, mode가 3번일 때
            if cnt3 > "1111" then
                cnt3 <= "00000";
            else
                cnt3 <= cnt3 + 1;
            end if;

        end if;
    end if;
end process;

```

---

--3. ram에 넣기 : ram값 입력

```

process(FPGA_RSTB, CLK)
begin
    if (FPGA_RSTB = '0') then
        for i in 0 to 31 loop
            ram(i) <= "00000000"; -- 모두 0 삽입
        end loop;
    elsif (CLK='1' and CLK'event) then
        ram(conv_integer(cnt2)) <= numwhole;
    end if;
end process;

```

end Behavioral;

하위모듈인 data 에서는 PUSH 버튼 6 개(PUSH1,2,3 PUSH\_SW0,1,2)가 눌러 질 때마다 어떤 모듈로 가서 동작을 하는지 cnt 는 어떤 상황에서 +1 을 하여 ram 에 저장이 되도록 하는지 등을 설계한다. PUSH1, 2, 3 이 눌러질 때마다 mode1,2,3 이 동작이 되도록 설정을 할 수 있다. 그리고 PUSH\_SW0,1 을 이용하여 DIP 스위치로부터 받은 손님의 번호 두 자리를 받을 수 있으며 이를 현재 cnt2 에 값에 맞게 ram 에 저장하도록 한다. 이는 나중에 호출할 때나 LCD 에 표시할 때 ram data 로부터 값을 불러오기 유용하도록 할 것이다. mode2 와 mode3 이 동작할 때마다 각각의 조건에서 cnt2 와 cnt3 이 +1 을 하여 다음 손님의 대기번호나 호출 번호를 cnt2 와 cnt3 으로서 간접적으로 저장할 수 있을 것이며 ram 에 저장된 data 와 cnt3 을 이용하여

호출되는 현재 대기번호에 해당하는 손님의 번호를 불러올 수 있으며 다음 4 명의 손님 번호도 불러오는것이 가능하도록 설계하였다.

### 3) Mode

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD.LOGIC_ARITH.ALL;
use IEEE.STD.LOGIC_UNSIGNED.ALL;

entity mode is
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           addr2, addr3 : in STD_LOGIC_VECTOR (4 downto 0); -- data에서 가져옴
           data0, data1, data2, data3, data4 : in STD_LOGIC_VECTOR (7 downto 0);-- data에서 가져옴 : 다음 4개의 번호
           mydata : in STD_LOGIC_VECTOR (7 downto 0);-- data에서 가져옴, 지금 데이터
           mode : in STD_LOGIC_VECTOR (2 downto 0); -- data에서 가져옴, big switch 번호
           w_enable : in STD_LOGIC; --lcd display에서 들어옴
           data_out : out STD_LOGIC; -- lcd display로 들어감. 데이터가 나갔는지 여부만 확인
           addr : out STD_LOGIC_VECTOR (4 downto 0); -- lcd display로 내보냄. 레지스터 번호
           data : out STD_LOGIC_VECTOR (7 downto 0)); -- lcd display로 내보냄. 해당 레지스터에 들어갈 값
end mode;

architecture Behavioral of mode is

    type reg is array ( 0 to 31) of std_logic_vector(7 downto 0);
    signal reg_file: reg;

    signal cnt : std_logic_vector(4 downto 0);--default
    signal tot_num : std_logic_vector(4 downto 0); -- 남은 사람수 확인 용
    signal num_1: std_logic_vector(4 downto 0); -- 남은 사람수의 1의 자리
    signal ahead : std_logic_vector(4 downto 0);
    --signal now_calling : std_logic_vector(7 downto 0); -- 지금 불러야 하는 번호

begin

begin
-----1. mode0에 따라 각자 넣기

    -- 들어갈 사람 몇 명 남았음?
    tot_num <= (addr2-addr3) when (addr2 > addr3) else "00000";
    -- addr2 가 버튼 2 누르는 순간 카운트 되기 때문에 그건 제외해야함
    ahead <= (addr2 - addr3 - 1) when (addr2 > addr3) else "00000";

    process(FPGA_RSTB, CLK)
    begin
        if (FPGA_RSTB = '0') or (mode = "00") then
            for i in 0 to 31 loop
                reg_file(i) <= X"20"; -- space 넣기
            end loop;
        elsif (CLK='1' and CLK'event) then
            -- mode 1
            if mode = "001" then
                reg_file(0) <= X"3C"; --<
                reg_file(1) <= data1(7 downto 4) + X"30";
                reg_file(2) <= data1(3 downto 0) + X"30";
                reg_file(3) <= X"3E"; -->

                reg_file(4) <= X"3C"; --<
                reg_file(5) <= data2(7 downto 4) + X"30";
                reg_file(6) <= data2(3 downto 0) + X"30";
                reg_file(7) <= X"3E"; -->

                reg_file(8) <= X"3C"; --<-
                reg_file(9) <= data3(7 downto 4) + X"30";
                reg_file(10) <= data3(3 downto 0) + X"30";
                reg_file(11) <= X"3E"; -->

                reg_file(12) <= X"3C"; --<
                reg_file(13) <= data4(7 downto 4) + X"30";
                reg_file(14) <= data4(3 downto 0) + X"30";
                reg_file(15) <= X"3E"; -->
            end if;
        end if;
    end process;
end;

```

```

reg_file(16) <= X"54"; --T
reg_file(17) <= X"4F"; --O
reg_file(18) <= X"54"; --t
reg_file(19) <= X"41"; --a
reg_file(20) <= X"4C"; --l
reg_file(21) <= X"20"; --space
reg_file(22) <= X"4E"; --n
reg_file(23) <= X"55"; --u
reg_file(24) <= X"4D"; --m
reg_file(25) <= X"42"; --b
reg_file(26) <= X"45"; --e
reg_file(27) <= X"52"; --r
reg_file(28) <= X"3A"; -- :

if tot_num > "11101" then --29 보다 크면
    reg_file(29) <= X"33"; -- 첫자리 3
    num_1 <= tot_num - "11110"; -- (-30)
    reg_file(30) <= num_1 + X"30";

elsif tot_num > "10011" then -- 19보다 클 때
    reg_file(29) <= X"32"; -- 첫자리 2
    num_1 <= tot_num - "10100"; -- (-20)
    reg_file(30) <= num_1 + X"30";

elsif tot_num > "1001" then-- 9보다 클 때
    reg_file(29) <= X"31"; -- 첫자리 2
    num_1 <= tot_num - "01010"; -- (-20)
    reg_file(30) <= num_1 + X"30";

else -- 9보다 작을 때
    reg_file(29) <= X"30";
    reg_file(30) <= tot_num + X"30";
end if;
reg_file(31) <= X"20"; --space

--mode 2
elsif mode = "010" then

    reg_file(0) <= X"59"; --Y
    reg_file(1) <= X"4F"; --O
    reg_file(2) <= X"55"; --U
    reg_file(3) <= X"52"; --R
    reg_file(4) <= X"4E"; --N
    reg_file(5) <= X"55"; --U
    reg_file(6) <= X"4D"; --M
    reg_file(7) <= X"42"; --B
    reg_file(8) <= X"45"; --E
    reg_file(9) <= X"52"; --R
    reg_file(10) <= X"3A"; --:
    reg_file(11) <= mydata(7 downto 4) + X"30";
    reg_file(12) <= mydata(3 downto 0) + X"30";
    reg_file(13) <= X"20"; --space
    reg_file(14) <= X"20"; --space
    reg_file(15) <= X"20"; --space

    if ahead > "11101" then --29 보다 크면
        reg_file(16) <= X"33"; -- 첫자리 3
        num_1 <= ahead - "11110"; -- (-30)
        reg_file(17) <= num_1 + X"30";

    elsif ahead > "10011" then -- 19보다 클 때
        reg_file(16) <= X"32"; -- 첫자리 2
        num_1 <= ahead - "10100"; -- (-20)
        reg_file(17) <= num_1 + X"30";

    elsif ahead > "1001" then-- 9보다 클 때
        reg_file(16) <= X"31"; -- 첫자리 2
        num_1 <= ahead - "01010"; -- (-20)
        reg_file(17) <= num_1 + X"30";

    else -- 9보다 작을 때
        reg_file(16) <= X"30";
        reg_file(17) <= ahead + X"30";
    end if;

```

```

reg_file(18) <= X"20"; --space
reg_file(19) <= X"50"; --p
reg_file(20) <= X"45"; --e
reg_file(21) <= X"4F"; --o
reg_file(22) <= X"50"; --p
reg_file(23) <= X"4C"; --l
reg_file(24) <= X"45"; --e
reg_file(25) <= X"20"; --space
reg_file(26) <= X"41"; --a
reg_file(27) <= X"48"; --h
reg_file(28) <= X"45"; --e
reg_file(29) <= X"41"; --a
reg_file(30) <= X"44"; --d
reg_file(31) <= X"20"; --space

-- 이때 mode3를 누르면 카운트가 되기 때문에 다음 데이터가 아닌 현재 데이터를 가져와야 한다.
elsif mode = "011" then

    reg_file(0) <= X"43"; --C
    reg_file(1) <= X"41"; --A
    reg_file(2) <= X"4C"; --L
    reg_file(3) <= X"4C"; --L
    reg_file(4) <= X"49"; --I
    reg_file(5) <= X"4E"; --N
    reg_file(6) <= X"47"; --G
    reg_file(7) <= X"3A"; --:
    reg_file(8) <= data0(7 downto 4) + X"30";
    reg_file(9) <= data0(3 downto 0) + X"30";
    reg_file(10) <= X"20";
    reg_file(11) <= X"20";
    reg_file(12) <= X"20";
    reg_file(13) <= X"20";
    reg_file(14) <= X"20";
    reg_file(15) <= X"20";
    reg_file(16) <= X"20";
    reg_file(17) <= X"20";
    reg_file(18) <= X"20";
    reg_file(19) <= X"20";
    reg_file(20) <= X"20"; --space
    reg_file(21) <= X"20";
    reg_file(22) <= X"20";
    reg_file(23) <= X"20";
    reg_file(24) <= X"20";
    reg_file(25) <= X"20";
    reg_file(26) <= X"20";
    reg_file(27) <= X"20";
    reg_file(28) <= X"20";
    reg_file(29) <= X"20";
    reg_file(30) <= X"20";
    reg_file(31) <= X"20";

```

```

        elsif mode = "100" then

            reg_file(0) <= X"45"; --E
            reg_file(1) <= X"4E"; --N
            reg_file(2) <= X"54"; --T
            reg_file(3) <= X"45"; --E
            reg_file(4) <= X"52"; --R
            reg_file(5) <= X"45"; --E
            reg_file(6) <= X"44"; --D
            reg_file(7) <= X"20";
            reg_file(8) <= X"20";
            reg_file(9) <= X"20";
            reg_file(10) <= X"20";
            reg_file(11) <= X"20";
            reg_file(12) <= X"20";
            reg_file(13) <= X"20";
            reg_file(14) <= X"20";
            reg_file(15) <= X"20";
            reg_file(16) <= X"20";
            reg_file(17) <= X"20";
            reg_file(18) <= X"20";
            reg_file(19) <= X"20";
            reg_file(20) <= X"20"; --space
            reg_file(21) <= X"20";
            reg_file(22) <= X"20";
            reg_file(23) <= X"20";
            reg_file(24) <= X"20";
            reg_file(25) <= X"20";
            reg_file(26) <= X"20";
            reg_file(27) <= X"20";
            reg_file(28) <= X"20";
            reg_file(29) <= X"20";
            reg_file(30) <= X"20";
            reg_file(31) <= X"20";
        end if;
    end if;
end process;

```

---

```

--2. 실제 모듈에 출력
process(FPGA_RSTB, CLK)
begin
    if (FPGA_RSTB = '0') then
        cnt <= (others => '0');
        data_out <= '0';

    elsif (CLK='1' and CLK'event) then
        if (w_enable = '1') then
            data <= reg_file(conv_integer(cnt));
            addr <= cnt;
            data_out <= '1';
            if cnt = X"1F" then -- 31
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        else
            data_out <= '0';
        end if;
    end if;
end process;

```

---

--3. mode 에서 다음 번호 출력

```
end Behavioral;
```

다음 하위 모듈인 mode 에서는 모드에 따라 data 모듈에서 값을 불러와서 LCD\_display 모듈을 이용하여 LCD 에 숫자 또는 문자가 표시되고 reg\_file()에 값이 저장할 수 있게 설계하는 모듈이다. mode1 일 경우 아래의 표와 같이 data 모듈에서 출력 값인 data1, data2, data3, data4 값을 이용하여 LCD 첫번째 줄에 호출된 손님 이후의 4 명의 손님의 번호가 나타나도록 하며 LCD 두번째 줄에는 호출된 손님의 대기순위번호와 대기하는 손님의 대기순위번호의 차이를 이용하여 총 대기 인원의 수를 나타낸다.

Mode2의 경우에는 data 모듈에서 cnt2를 이용한 ram에 저장된 정보를 이용하여 현재 손님의 번호를 LCD 첫번째 줄에 표시하도록 하며, 앞의 손님이 몇 명이 남았는지 cnt2와 cnt3을 이용하여 LCD 두 번째 줄에 출력되도록 한다.

Mode3의 경우에는 호출 번호를 cnt3을 이용한 ram 값에 저장된 정보를 이용하여 손님의 번호를 LCD에 띄우며 30초 안에 손님이 등장하였을 경우 PUSH\_SW2를 클릭하여 LCD에 손님이 등장하였다는 ENTERED의 표시가 되도록 설계하였다.

Mode1과 Mode2의 경우 십의 자리에 따라 ASCII 코드가 달라지기 때문에 조건문을 이용하여 나눠주는 과정이 들어간다.

#### 4) LCD\_display

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LCD_display is
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           LCD_A : out STD_LOGIC_VECTOR (1 downto 0);
           LCD_EN : out STD_LOGIC;
           LCD_D : out STD_LOGIC_VECTOR (7 downto 0);
           data_out : in STD_LOGIC; -- mode에서 들어옴 값이 들어왔나만 표시
           addr : in STD_LOGIC_VECTOR (4 downto 0); -- mode에서 들어옴, 해당 디스플레이 위치 표시
           data : in STD_LOGIC_VECTOR (7 downto 0); -- mode에서 들어옴, 해당 디스플레이에 들어갈 코드 표시
           w_enable : out STD_LOGIC); -- 1이면 쓰고 있는거, 0이면 안 쓰는 거
end LCD_display;

architecture Behavioral of LCD_display is

    type reg is array (0 to 31) of std_logic_vector ( 7 downto 0);
    signal reg_file : reg;
    signal w_enable_reg : std_logic;
    signal load_100k : std_logic;
    signal clk_100k : std_logic;
    signal cnt_100k : std_logic_vector(7 downto 0);
    signal load_50 : std_logic;
    signal clk_50 : std_logic;
    signal cnt_50 : std_logic_vector (11 downto 0);
    signal lcd_cnt : std_logic_vector (7 downto 0);
    signal lcd_state : std_logic_vector (7 downto 0);
    signal lcd_nstate : std_logic_vector (7 downto 0);
    signal lcd_db : std_logic_vector (7 downto 0);
```

```

begin
-----
--1. 100khz clock
process(FPGA_RSTB, CLK, load_100k, cnt_100k)
begin
    if (FPGA_RSTB = '0') then
        cnt_100k <= (others => '0');
        clk_100k <= '0';
    elsif (CLK = '1' and CLK'event) then
        if load_100k = '1' then
            cnt_100k <= (others => '0');
            clk_100k <= not clk_100k;
        else
            cnt_100k <= cnt_100k + 1 ;
        end if;
    end if;
end process;

load_100k <= '1' when (cnt_100k = X"13") else '0';

-----
--2. 50hz clock
process(FPGA_RSTB, clk_100k, load_50, cnt_50)
begin
    if (FPGA_RSTB = '0') then
        cnt_50 <= (others => '0');
        clk_50 <= '0';
    elsif (clk_100k = '1' and clk_100k'event) then
        if load_50 = '1' then
            cnt_50 <= (others => '0');
            clk_50 <= not clk_50;
        else
            cnt_50<= cnt_50 + 1 ;
        end if;
    end if;
end process;

load_50 <= '1' when (cnt_50 = X"3E7") else '0';

-----
--3. state machine
process(FPGA_RSTB, clk_50)
begin
    if (FPGA_RSTB = '0') then
        lcd_state <= (others => '0');

    elsif rising_edge(clk_50) then
        lcd_state <= lcd_nstate;

    end if;
end process;

w_enable_reg <= '0' when (lcd_state < X"06" or lcd_state = X"16")
else '1';

-----
--4. 지정 값 넣기
process (FPGA_RSTB, CLK)
begin
    if (FPGA_RSTB ='0') then
        for i in 0 to 31 loop
            reg_file(i) <= X"20";
        end loop;
    elsif (CLK='1' and CLK'event) then
        if (w_enable_reg = '1' and data_out = '1') then
            reg_file(conv_integer(addr)) <= data;
        end if;
    end if;
end process;

```

```
--5. 실제 값 넣기
process(FPGA_RSTB, lcd_state)
begin
    if (FPGA_RSTB = '0') then
        lcd_nstate <= X"00";
    else
        case lcd_state is
            when X"00" => lcd_db <= "00111000"; -- function set
                            lcd_nstate <= X"01";
            when X"01" => lcd_db <= "00001000"; -- display off
                            lcd_nstate <= X"02";
            when X"02" => lcd_db <= "00000001"; -- display clear
                            lcd_nstate <= X"03";
            when X"03" => lcd_db <= "00000110"; -- entry mode set
                            lcd_nstate <= X"04";
            when X"04" => lcd_db <= "00001100"; -- display on
                            lcd_nstate <= X"05";
            when X"05" => lcd_db <= "00000011"; -- return home
                            lcd_nstate <= X"06";
            when X"06" => lcd_db <= reg_file(0);
                            lcd_nstate <= X"07";
            when X"07" => lcd_db <= reg_file(1);
                            lcd_nstate <= X"08";
            when X"08" => lcd_db <= reg_file(2);
                            lcd_nstate <= X"09";
            when X"09" => lcd_db <= reg_file(3);
                            lcd_nstate <= X"0A";
            when X"0A" => lcd_db <= reg_file(4);
                            lcd_nstate <= X"0B";
            when X"0B" => lcd_db <= reg_file(5);
                            lcd_nstate <= X"0C";
            when X"0C" => lcd_db <= reg_file(6);
                            lcd_nstate <= X"0D";
            when X"0D" => lcd_db <= reg_file(7);
                            lcd_nstate <= X"0E";
            when X"0E" => lcd_db <= reg_file(8);
                            lcd_nstate <= X"0F";
            when X"0F" => lcd_db <= reg_file(9);
                            lcd_nstate <= X"10";
            when X"10" => lcd_db <= reg_file(10);
                            lcd_nstate <= X"11";
            when X"11" => lcd_db <= reg_file(11);
                            lcd_nstate <= X"12";
            when X"12" => lcd_db <= reg_file(12);
                            lcd_nstate <= X"13";
```

```

when X"13" => lcd_db <= reg_file(13);
                lcd_nstate <= X"14";
when X"14" => lcd_db <= reg_file(14);
                lcd_nstate <= X"15";
when X"15" => lcd_db <= reg_file(15);
                lcd_nstate <= X"16";
when X"16" => lcd_db <= X"C0"; -- change line
                lcd_nstate <= X"17";
when X"17" => lcd_db <= reg_file(16);
                lcd_nstate <= X"18";
when X"18" => lcd_db <= reg_file(17);
                lcd_nstate <= X"19";
when X"19" => lcd_db <= reg_file(18);
                lcd_nstate <= X"1A";
when X"1A" => lcd_db <= reg_file(19);
                lcd_nstate <= X"1B";
when X"1B" => lcd_db <= reg_file(20);
                lcd_nstate <= X"1C";
when X"1C" => lcd_db <= reg_file(21);
                lcd_nstate <= X"1D";
when X"1D" => lcd_db <= reg_file(22);
                lcd_nstate <= X"1E";
when X"1E" => lcd_db <= reg_file(23);
                lcd_nstate <= X"1F";
when X"1F" => lcd_db <= reg_file(24);
                lcd_nstate <= X"20";
when X"20" => lcd_db <= reg_file(25);
                lcd_nstate <= X"21";
when X"21" => lcd_db <= reg_file(26);
                lcd_nstate <= X"22";
when X"22" => lcd_db <= reg_file(27);
                lcd_nstate <= X"23";
when X"23" => lcd_db <= reg_file(28);
                lcd_nstate <= X"24";
when X"24" => lcd_db <= reg_file(29);
                lcd_nstate <= X"25";
when X"25" => lcd_db <= reg_file(30);
                lcd_nstate <= X"26";
when X"26" => lcd_db <= reg_file(31);
                lcd_nstate <= X"05";
when others => lcd_db <= (others => '0');
end case;
end if;
end process;

LCD_A(1)<= '0';
LCD_A(0)<= '0' when (lcd_state >= X"00" and lcd_state < X"06") or (lcd_state = X"16")
else '1';
LCD_EN <= clk_50;
LCD_D <= lcd_db;
w_enable <= w_enable_reg;

end Behavioral;

```

그 다음 모듈인 LCD\_display 모듈은 mode 모듈로부터 받은 reg\_file 의 정보를 가지고 LCD에 출력하는 역할을 하는 모듈이다. lcd\_state 가 00~05 까지는 instruction code 임을 뜻하고 본격적으로 X"06"부터 X"26"까지 LCD 에 reg 에서 data 를 불러와 출력한다. 중간 X"16"에서는 line change 를 하는 instruction 을 넣었다. Instruction 과 출력 관련 명령은 LCD\_D, LCD\_A 에 저장이 되도록 하며 이는 LCD 에 숫자 또는 문자가 display 되도록 한다. State Machine 을 이용하였다.

## 5) Clock

clock 모듈에서 data 모듈로 넘어가는 신호중에 cnt3(호출 인원 수)를 조정하는 신호가 있다. push\_ent 이다. 이 신호는 data 모듈에서 cnt3 를 셀 때, 손님이 30 초 안에 오지 않는다면 +1 을 하는 것을 도와주는 신호이다. data 모듈에서는 이 신호가 0 에서 1 로 넘어가는 순간에만 카운트를 하도록 설계했다. clock 모듈에서는 push\_ent 신호는 카운트다운하는 시간이 01 초일 때 1로 바뀌고, 다시 30 초가 되면 0 으로 바뀌는 식으로 설계를 하였다. 30 초 카운트 다운 시간이 다 되었을 때 자동으로 1로 바뀌고, 그걸 data 모듈에서 인식해서 cnt3 에 +1 을 해주면서 내보내야 할 호출 번호가 바뀐다. 그 바뀐 번호를 display module 에서 클락에 맞게 lcd 에 출력하는 구조이다.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clock is
    Port ( FPGA_RSTB : in STD_LOGIC;
           CLK : in STD_LOGIC;
           DIGIT : out STD_LOGIC_VECTOR (6 downto 1);-- 숫자 패널 이름
           SEG_A : out STD_LOGIC;
           SEG_B : out STD_LOGIC;
           SEG_C : out STD_LOGIC;
           SEG_D : out STD_LOGIC;
           SEG_E : out STD_LOGIC;
           SEG_F : out STD_LOGIC;
           SEG_G : out STD_LOGIC;
           SEG_DP : out STD_LOGIC;
           addr2 : in STD_LOGIC_VECTOR (4 downto 0); --mode 2에서 온 손님 번호
           addr3 : in STD_LOGIC_VECTOR (4 downto 0); -- mode 3에서 온 호출 번호
           mode : in STD_LOGIC_VECTOR(2 downto 0);
           push_ent : out STD_LOGIC);
end clock;

architecture Behavioral of clock is

signal s01_clk : STD_LOGIC; -- 60초 다 세면 넘어감
signal hr01_cnt, hr10_cnt : STD_LOGIC_VECTOR(3 downto 0);
signal min01_cnt, min10_cnt : STD_LOGIC_VECTOR(3 downto 0);
signal sec01_cnt, sec10_cnt : STD_LOGIC_VECTOR(3 downto 0);

signal sel : STD_LOGIC_VECTOR(2 downto 0);
signal data : STD_LOGIC_VECTOR(3 downto 0);
signal seg : STD_LOGIC_VECTOR(7 downto 0);

signal tot_num, tot_num_5bit : STD_LOGIC_VECTOR (4 downto 0);
signal push_ent_temp : std_logic;
```

```

begin
----- 1. 패널 번호 지정 및 각 패널 지정 숫자에 따른 출력값 지정

process(sel)
begin
    case sel is
        when "000" => -- 원쪽 첫번째 숫자 패널
            DIGIT <= "000001";
            data <= hr10_cnt;
        when "001" => -- 2번째 패널
            DIGIT <= "000010";
            data <= hr01_cnt;
        when "010" => -- 3번째 패널
            DIGIT <= "000100";
            data <= min10_cnt;
        when "011" => -- 4번째 패널
            DIGIT <= "001000";
            data <= min01_cnt;
        when "100" => -- 5번째 패널
            DIGIT <= "010000";
            data <= sec10_cnt;
        when "101" => -- 6번째 패널
            DIGIT <= "100000";
            data <= sec01_cnt;
        when others => null;
    end case;
end process;

----- 2. reset, 패널 띄우는 방식 설정

process(FPGA_RSTB,CLK)
    -- 변수 입력
    variable seg_clk_cnt : integer range 0 to 200;
begin
    if (FPGA_RSTB = '0') then -- reset 버튼 누르면 0으로 돌아가기
        sel <= "000";
        seg_clk_cnt := 0;
    elsif ( CLK'event and CLK = '1') then
        -- 200개 * 4MHz = 50us의 주기를 가지게 된다.

        if (seg_clk_cnt = 200) then
            seg_clk_cnt := 0; -- 새로운 clock 만들기
            -- clock0이 200개가 되었을 때만 sel 값 변화됨

            -- 한 번에 6개의 패널 표시가 불가능해서
            -- 하나씩 돌아가면서 표시중
            if (sel = "101") then
                -- 마지막 숫자패널까지 가면 sel은 다시 첫 패널로 돌아가기
                sel <= "000";
            else
                sel <= sel + 1;
            end if;

            else
                seg_clk_cnt := seg_clk_cnt + 1;
            end if;
        end if;
    end process;

```

```
-- 3. 각 숫자를 만들기 위한 조각들 구성

--dp g f e d c b a 순서
process(data)
begin
    case data is
        when "0000" => --number 0
            seg <= "00111111";
        when "0001" => --number 1
            seg <= "00000110";
        when "0010" => --number 2
            seg <= "01011011";
        when "0011" => --number 3
            seg <= "01001111";
        when "0100" => --number 4
            seg <= "01100110";
        when "0101" => --number 5
            seg <= "01101101";
        when "0110" => --number 6
            seg <= "01111101";
        when "0111" => --number 7
            seg <= "00000111";
        when "1000" => --number 8
            seg <= "01111111";
        when "1001" => --number 9
            seg <= "01100111";
        when "1010" => --number 10 : 빈칸
            seg <= "00000000";
        when others => null;
    end case;
end process;
```

```

-- 4. out signal 과 내부신호 연결

SEG_A <= seg(0);
SEG_B <= seg(1);
SEG_C <= seg(2);
SEG_D <= seg(3);
SEG_E <= seg(4);
SEG_F <= seg(5);
SEG_G <= seg(6);
SEG_DP <= seg(7);

-- 5. clock 만들기 : 4MHZ -> 0.5sec

process(FPGA_RSTB,CLK)
  -- 250ns = 1/4,000,000
  -- 250ns * 2,000,000 = 0.5
  variable count_clk : integer range 0 to 2000000;--2,000,000

begin
  if (FPGA_RSTB = '0') then
    s01_clk <= '1';
    count_clk := 0;

  elsif( CLK'event and CLK = '1') then
    if (count_clk = 2000000) then -- 2,000,000
      -- new clock reset
      count_clk := 0;
      -- 새로 만든 clock을 0이면 1로, 1이면 0으로 반전시킨다.

      s01_clk <= not s01_clk;

    else
      count_clk := count_clk + 1;
    end if;
  end if;
end process;

```

#### --6. mode 3에서 count 시작하기

```

tot_num <= (addr2>addr3) when (addr2 > addr3) else "00000";
--mode 2에서 온 손님 번호 addr2에서 mode 3에서 온 호출 번호 addr3를 뺀 결과값을 현재 대기중인 총 인원tot_num으로 할당한다

process(FPGA_RSTB, s01_clk)

variable h10_cnt, h01_cnt, m10_cnt, m01_cnt, s10_cnt, s01_cnt : STD_LOGIC_VECTOR (3 downto 0);

begin
  if (FPGA_RSTB = '0') then --reset : 00:00:00
    s01_cnt := "0000"; --6자리 중 제일 오른쪽 값을 "0000" 의 data값으로 할당
    s10_cnt := "0000"; --s01_cnt의 그 왼쪽 값을 "0000" 의 data값으로 할당
    m01_cnt := "0000"; --s10_cnt의 그 왼쪽 값을 "0000" 의 data값으로 할당
    m10_cnt := "0000"; --m01_cnt의 그 왼쪽 값을 "0000" 의 data값으로 할당
    h01_cnt := "0000"; --m10_cnt의 그 왼쪽 값을 "0000" 의 data값으로 할당
    h10_cnt := "0000"; --h01_cnt의 그 왼쪽 값을 "0000" 의 data값으로 할당
  elsif (s01_clk = '1' and s01_clk'event) then --새로 만든 clock주기 s01_clk0| rising edge일 경우
    if (mode="001" or mode = "010") then --mode1 과 mode2가 선택이 되었을 때
      s01_cnt := "0000";
      s10_cnt := "0000";
      m01_cnt := "0000";
      m10_cnt := "0000";
      --7segment의 6자리중 1,4,5,6의 자리에 "0000"의 data값 할당
    end if;
  end if;
end process;

```

```

--tot_num의 범위에 따라 7segment 6자리수 2,3번째 자리에 if조건문에 넣맞는 data값을 할당
--대기만원당 10분으로 계산을 하고 총 대기 시간을 계산. 결과값에는 시간의 일의자리와 분의 십의자리를 이용
if (tot_num < "00110") then --tot_num이 6보다 작은 경우(총 대기시간이 60분 미만)
    h01_cnt := "0000"; --0시간
    m10_cnt := tot_num(3 downto 0); --(tot_num X 10분) 의 십의자리
elsif (tot_num < "01100") then --tot_num이 60이상 12미만인 경우(총 대기시간이 2시간 미만)
    h01_cnt := "0001"; --1시간
    m10_cnt := tot_num(3 downto 0) - "0110"; --(tot_num X 10분 - 60분)의 십의자리
elsif (tot_num < "10010") then --tot_num이 120이상 18미만인 경우(총 대기시간이 3시간 미만)
    h01_cnt := "0010"; --2시간
    tot_num_5bit <= tot_num - "01100";
    m10_cnt := tot_num_5bit(3 downto 0); --(tot_num X 10분 - 120분)의 십의자리
elsif (tot_num < "11000") then --tot_num이 180이상 24미만인 경우(총 대기시간이 4시간미만)
    h01_cnt := "0011"; --3시간
    tot_num_5bit <= tot_num - "10010";
    m10_cnt := tot_num_5bit(3 downto 0); --(tot_num X 10분 - 180분)의 십의자리
else --모두 4시간
    h01_cnt := "0100"; --4시간
    m10_cnt := "0000"; --00분
end if;

elsif (mode=="011") then --mode3이 선택이 뇌었을 때
    m01_cnt := "0000";
    m10_cnt := "0000";
    h01_cnt := "0000";
    h10_cnt := "0000";
    --왼쪽 4자리는 모두 0으로 표시

    if (s01_cnt = "0000") then
        if (s10_cnt = "0011") then --30일 경우
            s01_cnt := "1001";
            s10_cnt := "0010"; --29로
            push_ent_temp <= '0';
        elsif (s10_cnt = "0010") then --20일 경우
            s01_cnt := "1001";
            s10_cnt := "0001"; --19로
        elsif (s10_cnt = "0001") then --10일 경우
            s01_cnt := "1001";
            s10_cnt := "0000"; --9로
        else
            s01_cnt := "0000";
            s10_cnt := "0011"; --나머지 경우 30으로
        end if;
    elsif (s01_cnt = "0001" and s10_cnt = "0000") then --그 외의 경우에는 cnt가 -1씩 감소하도록 설계
        push_ent_temp <= '1';
        s01_cnt := s01_cnt -1;
    else
        s01_cnt := s01_cnt -1;
    end if;
end if;
end if;

--지금까지 정리된 variable 변수 s01_cnt ~ h10_cnt가 signal 내부변수 seg01_cnt ~ hr10_cnt에 각각 할당 되도록 설계
sec01_cnt<=s01_cnt;
sec10_cnt<=s10_cnt;
min01_cnt<=m01_cnt;
min10_cnt<=m10_cnt;
hr01_cnt<=h01_cnt;
hr10_cnt<=h10_cnt;
--0) signal 내부 변수들은 data값에 저장되어 7-segment 각각에 표시되게 된다.

end process;
push_ent <= push_ent_temp;

end Behavioral;

```

5 번째 모듈인 clock 모듈에서는 모드에 따라 7-segment에 나타나는 값을 설계하는 모듈이다. mode1과 mode2에서는 대기 시간을 표시하도록 설계하였다. 한 명당 업무시간은 10분이며 8명의 대기인원이 있을 경우 80분 즉 1시간 20분이 표시되도록 설계하였다. 7-Segment의 6자리중 왼쪽 두 자리는 시간을 그 다음 두 자리는 분을 나머지 두 자리는 초를 나타내는데 여기서 초는 의미가 없으므로 모두 0으로 표시되도록 하였다. 총 대기 인원은 cnt3과 cnt2를 이용하여 현재 손님의 대기순위번호 addr2에서 현재 호출된 손님이 대기순위번호 addr3을 차감하여 표현하였고 이에 총 대기해야 하는 시간은 곱하기 10을 하였다. mode3의 경우에는 30초를 세는데 30,29,28,27..... 이런 식인 역순으로 셈했다. 이는 30초 동안 호출된 손님의 등장여부를 판단하는데 좋은 지표가 될 것이다.

주석 5 번까지는 1초를 세고 LED를 표시하는 코드이고 실제 6 번 코드가 위의 기능을 중심적으로 구사한 코드이다.

## 2. ISim Simulation Result

TestBench는 분석을 간단하기 위해 모든 clock scale을 1/25로 조정하였고, 동작 시 생성되는 waveform은 각 하위 모듈의 signal 값을 확인하지 않고 전체적인 입력에 따른 결과만 확인하였다. 상황은 다음과 같다.

번호가 11,22,33,44,55인 손님이 차례로 등장한다

- > 번호가 11인 손님을 호출할 것이고 손님이 30초 안에 등장한다.
- > 번호가 22인 손님을 호출할 것이고 손님이 30초 안에 등장한다.
- > 번호가 33인 손님을 호출할 것이고 손님이 30초 안에 등장한다.
- > 번호가 44인 손님을 호출할 것이고 손님이 30초 안에 등장하지 않는다.
- > 자동적으로 다음 손님인 번호가 55인 손님의 호출이 된다.
- > 손님이 30초 안에 등장한다. -> 모든 손님들의 업무가 끝난다.

다음과 같은 상황에서 **ISim Simulation Result**와 **Rov-lab 3000 kit Result**를 적용하여 동작 여부를 확인 할 것이다.

Test Bench 를 실행하기 위한 VHDL 코드는 다음과 같다.

```

1 |-----  

2 |---- Testbench의 Simulation 속도를 빠르게 하기 위해  

3 |---- clock은 10 ns를 사용하였다  

4 |-----  

5 |  

6 |  

7 LIBRARY ieee;  

8 USE ieee.std_logic_1164.ALL;  

9 |  

10 ENTITY waitinglistTB IS  

11 END waitinglistTB;  

12 |  

13 ARCHITECTURE behavior OF waitinglistTB IS  

14 |  

15 -- Component Declaration for the Unit Under Test (UUT)  

16 |  

17 COMPONENT waitinglist  

18 PORT(  

19     FPGA_RSTB : IN std_logic;  

20     CLK : IN std_logic;  

21     LCD_A : OUT std_logic_vector(1 downto 0);  

22     LCD_EN : OUT std_logic;  

23     LCD_D : OUT std_logic_vector(7 downto 0);  

24     DIGIT : OUT std_logic_vector(6 downto 1);  

25     SEG_A : OUT std_logic;  

26     SEG_B : OUT std_logic;  

27     SEG_C : OUT std_logic;  

28     SEG_D : OUT std_logic;  

29     SEG_E : OUT std_logic;  

30     SEG_F : OUT std_logic;  

31     SEG_G : OUT std_logic;  

32     SEG_DP : OUT std_logic;  

33     load_b1 : IN std_logic;  

34     load_b2 : IN std_logic;  

35     load_b3 : IN std_logic;  

36     load_s1 : IN std_logic;  

37     load_s2 : IN std_logic;  

38     load_s3 : IN std_logic;  

39     number : IN std_logic_vector(3 downto 0)  

40 );  

41 END COMPONENT;  

42 |  

43 |-----  

44 --Inputs  

45 signal FPGA_RSTB : std_logic := '0';  

46 signal CLK : std_logic := '0';  

47 signal load_b1 : std_logic := '0';  

48 signal load_b2 : std_logic := '0';  

49 signal load_b3 : std_logic := '0';  

50 signal load_s1 : std_logic := '0';  

51 signal load_s2 : std_logic := '0';  

52 signal load_s3 : std_logic := '0';  

53 signal number : std_logic_vector(3 downto 0) := (others => '0');  

54 |  

55 --Outputs  

56 signal LCD_A : std_logic_vector(1 downto 0);  

57 signal LCD_EN : std_logic;  

58 signal LCD_D : std_logic_vector(7 downto 0);  

59 signal DIGIT : std_logic_vector(6 downto 1);  

60 signal SEG_A : std_logic;  

61 signal SEG_B : std_logic;  

62 signal SEG_C : std_logic;  

63 signal SEG_D : std_logic;  

64 signal SEG_E : std_logic;  

65 signal SEG_F : std_logic;  

66 signal SEG_G : std_logic;  

67 signal SEG_DP : std_logic;  

68 |  

69 -- Clock period definitions  

70 constant CLK_period : time := 10 ns;  

71 |  

72 BEGIN  

73 |  

74 -- Instantiate the Unit Under Test (UUT)  

75 uut: waitinglist PORT MAP (  

76     FPGA_RSTB => FPGA_RSTB,  

77     CLK => CLK,  

78     LCD_A => LCD_A,  

79     LCD_EN => LCD_EN,  

80     LCD_D => LCD_D,  

81     DIGIT => DIGIT,  

82     SEG_A => SEG_A,  

83     SEG_B => SEG_B,  

84     SEG_C => SEG_C,  

85     SEG_D => SEG_D,  

86     SEG_E => SEG_E,  

87     SEG_F => SEG_F,  

88     SEG_G => SEG_G,  

89     SEG_DP => SEG_DP,  

90     load_b1 => load_b1,  

91     load_b2 => load_b2,  

92     load_b3 => load_b3,  

93     load_s1 => load_s1,  

94     load_s2 => load_s2,  

95     load_s3 => load_s3,  

96     number => number  

97 );  

98 |  

99 -- Clock process definitions  

100 CLK_process :process  

101 begin  

102     CLK <= '0';  

103     wait for CLK_period/2;  

104     CLK <= '1';  

105     wait for CLK_period/2;  

106 end process;  

107 |  

108 |  

109 -- Stimulus process  

110 stim_proc: process  

111 begin  

112     fpga_rstb <= '1'; -- 모든 switch는 active low 로 동작  

113     load_b1 <= '1';  

114     load_b2 <= '1';  

115     load_b3 <= '1';  

116     load_s1 <= '1';  

117     load_s2 <= '1';  

118     load_s3 <= '1';

```

```

119      load_s1 <= '1';
120      load_s2 <= '1';
121      load_s3 <= '1';
122
123      wait for 4 ms;
124
125      -----
126      load_b2 <= '0'; -- 2번 큰 스위치를 눌러서 입력 시작
127      wait for 10 ns;
128
129      load_b2 <= '1';
130      wait for 10 ns;
131
132
133
134      number <= "0001"; -- 앞자리 1
135      load_s1 <= '0'; -- 입력 시작
136      wait for 10 ns;
137
138      load_s1 <= '1'; -- 입력 끝
139      wait for 10 ns;
140
141      number <= "0001"; -- 뒷자리 1
142      load_s2 <= '0'; -- 입력 시작
143      wait for 10 ns;
144
145      load_s2 <= '1'; -- 입력 끝
146      wait for 10 ns;
147
148
149      -----
150      load_b1 <= '0'; -- 1번 큰 스위치를 눌러서 대기명단 확인
151      wait for 10 ns;
152
153      load_b1 <= '1';
154      wait for 10 ns;
155
156
157
158      wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
159
160
161
162      -----
163      load_b2 <= '0'; -- 2번 큰 스위치를 눌러서 입력 시작
164      wait for 10 ns;
165
166      load_b2 <= '1';
167      wait for 10 ns;
168
169
170      number <= "0010"; -- 앞자리 2
171      load_s1 <= '0';
172      wait for 10 ns;
173
174      load_s1 <= '1';
175      wait for 10 ns;
176
177      number <= "0010"; -- 뒷자리 2
178      load_s2 <= '0';
179      wait for 10 ns;
180
181      load_s2 <= '1';
182      wait for 10 ns;
183
184
185      -----
186      load_b1 <= '0'; -- 1번 큰 스위치를 눌러서 대기명단 확인
187      wait for 10 ns;
188
189      load_b1 <= '1';
190      wait for 10 ns;
191
192
193      wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
194
195
196
197      -----
198      load_b2 <= '0'; -- 2번 큰 스위치를 눌러서 입력 시작
199      wait for 10 ns;
200      load_b2 <= '1';

```

```

200     load_b2 <= '1';
201     wait for 10 ns;
202
203
204     number <= "0011"; -- 앞자리 3
205     load_s1 <= '0';
206     wait for 10 ns;
207
208     load_s1 <= '1';
209     wait for 10 ns;
210
211     number <= "0011"; -- 뒷자리 3
212     load_s2 <= '0';
213     wait for 10 ns;
214
215     load_s2 <= '1';
216     wait for 10 ns;
217
218
219     load_b1 <= '0'; -- 큰 스위치1을 눌러서 대기명단 확인
220     wait for 10 ns;
221
222     load_b1 <= '1';
223     wait for 10 ns;
224
225
226
227     wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
228
229
230
231     load_b2 <= '0'; -- 큰 스위치2를 눌러서 입력 시작
232     wait for 10 ns;
233
234     load_b2 <= '1';
235     wait for 10 ns;
236
237
238     number <= "0100"; -- 앞자리 4
239     load_s1 <= '0';
240     wait for 10 ns;
241
242     load_s1 <= '1';
243     wait for 10 ns;
244
245     number <= "0100"; -- 뒷자리 4
246     load_s2 <= '0';
247     wait for 10 ns;
248
249     load_s2 <= '1';
250     wait for 10 ns;
251
252
253     load_b1 <= '0';
254     wait for 10 ns;
255
256     load_b1 <= '1';
257     wait for 10 ns;
258
259
260
261     wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
262
263
264
265     load_b2 <= '0';
266     wait for 10 ns;
267
268     load_b2 <= '1';
269     wait for 10 ns;
270
271
272     number <= "0101"; -- 앞자리 5
273     load_s1 <= '0';
274     wait for 10 ns;
275
276     load_s1 <= '1';
277     wait for 10 ns;
278
279     number <= "0101"; -- 뒷자리 5
280     load_s2 <= '0';
281     wait for 10 ns;
282

```

```

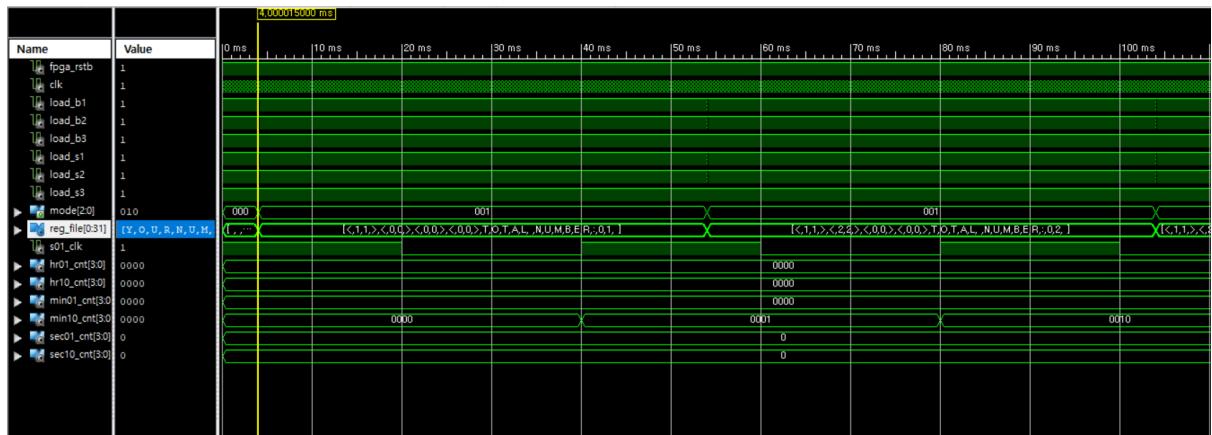
282     load_s2 <= '1';
283     wait for 10 ns;
285
286 -----[load_b1 <= '0'; -- 현재 대기명단 확인 시 <11> <22> <33> <44> 확인 가능
287     wait for 10 ns;
289
290     load_b1 <= '1';
291     wait for 10 ns;
292 -----
293
294 -----[wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
295
296 -----
297
298 -----[load_b3 <= '0'; -- 큰 스위치 3을 눌러서 대기 인원 호출 시작
299     wait for 10 ns;
301
302     load_b3 <= '1';
303     wait for 10 ns;
304 -----
305
306     wait for 560 ms; -- 충분한 시간을 기다린 후에 (250 : 14sec = 10 : 560 ms)
307
308 -----[load_s3 <= '0'; -- 작은 스위치 3을 눌러서 호출에 응답
309     wait for 10 ns;
311
312     load_s3 <= '1';
313     wait for 10 ns;
314 -----
315
316 -----[load_b1 <= '0'; -- 큰 스위치 1을 눌러서 현재 대기 인원 확인
317     wait for 10 ns; -- <22> <33> <44> <55> 확인 가능
319
320     load_b1 <= '1';
321     wait for 10 ns;
322 -----
323
324 -----[wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
325
326 -----
327
328 -----[load_b3 <= '0'; -- 큰 스위치 3을 눌러서 대기 인원 호출 시작
329     wait for 10 ns;
331
332     load_b3 <= '1';
333     wait for 10 ns;
334 -----
335
336 -----
337     wait for 960 ms; -- 충분한 시간을 기다린 후에 (250 : 24sec = 10 : 960 ms)
339
340 -----
341
342 -----[load_s3 <= '0'; -- 작은 스위치 3을 눌러서 호출에 응답
343     wait for 10 ns;
345
346     load_s3 <= '1';
347     wait for 10 ns;
348 -----
349
350 -----[load_b1 <= '0'; -- 큰 스위치 1을 눌러서 현재 대기 인원 확인
351     wait for 10 ns; -- <33> <44> <55> <00> 확인 가능
353
354     load_b1 <= '1';
355     wait for 10 ns;
356 -----
357
358 -----[wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
359
360 -----
361
362 -----[load_b3 <= '0'; -- 큰 스위치 3을 눌러서 대기 인원 호출 시작

```

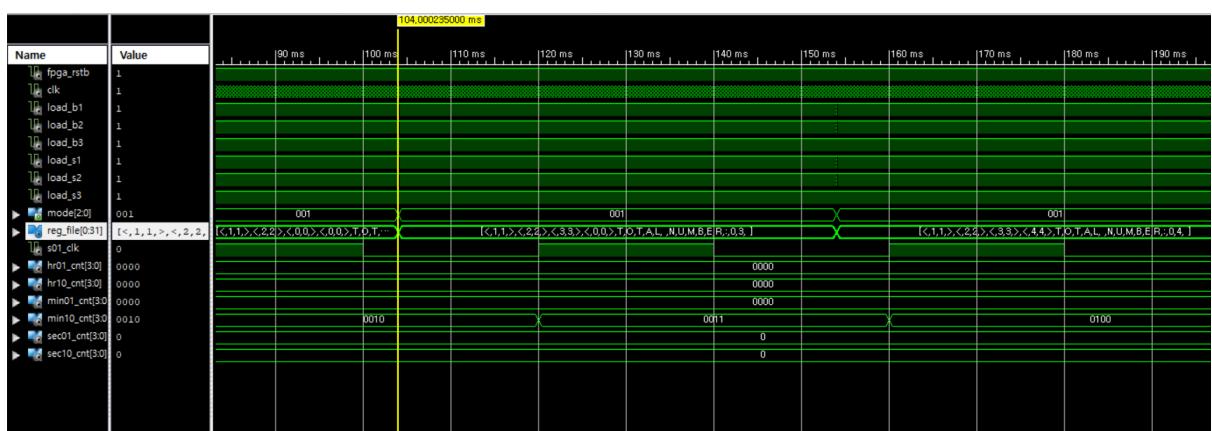
```

361
362     load_b3 <= '0'; -- 큰 스위치 3을 눌러서 대기 인원 호출 시작
363     wait for 10 ns;
364
365     load_b3 <= '1';
366     wait for 10 ns;
367
368 -----
369
370 -----
371     wait for 200 ms; -- 충분한 시간을 기다린 후에 (250 : 5sec = 10 : 200 ms)
372
373 -----
374
375 -----
376     load_s3 <= '0'; -- 작은 스위치 3을 눌러서 호출에 응답
377     wait for 10 ns;
378
379     load_s3 <= '1';
380     wait for 10 ns;
381
382 -----
383
384     load_b1 <= '0'; -- 큰 스위치 1을 눌러서 현재 대기 인원 확인
385     wait for 10 ns; -- <44> <55> <00> <00> 확인 가능
386
387     load_b1 <= '1';
388     wait for 10 ns;
389
390 -----
391
392     wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
393
394 -----
395
396     load_b3 <= '0'; -- 큰 스위치 3을 눌러서 대기 인원 호출 시작
397     wait for 10 ns;
398
399     load_b3 <= '1';
400     wait for 10 ns;
401
402 -----
403
404 -----
405     wait for 1400 ms; -- 충분한 시간을 기다린 후에 (250 : 5sec = 10 : 200 ms)
406
407 -----
408
409 -----
410     load_s3 <= '0'; -- 작은 스위치 3을 눌러서 호출에 응답
411     wait for 10 ns;
412
413     load_s3 <= '1';
414     wait for 10 ns;
415
416 -----
417
418     load_b1 <= '0'; -- 큰 스위치 1을 눌러서 현재 대기 인원 확인
419     wait for 10 ns; -- <00> <00> <00> <00> 확인 가능
420
421     load_b1 <= '1';
422     wait for 10 ns;
423
424 -----
425
426     wait for 50 ms; -- LCD에 표시가 완료될 때까지 충분한 시간을 기다린다
427
428 -----
429
430 -----
431
432     wait;
433 end process;
434
435 END;
436

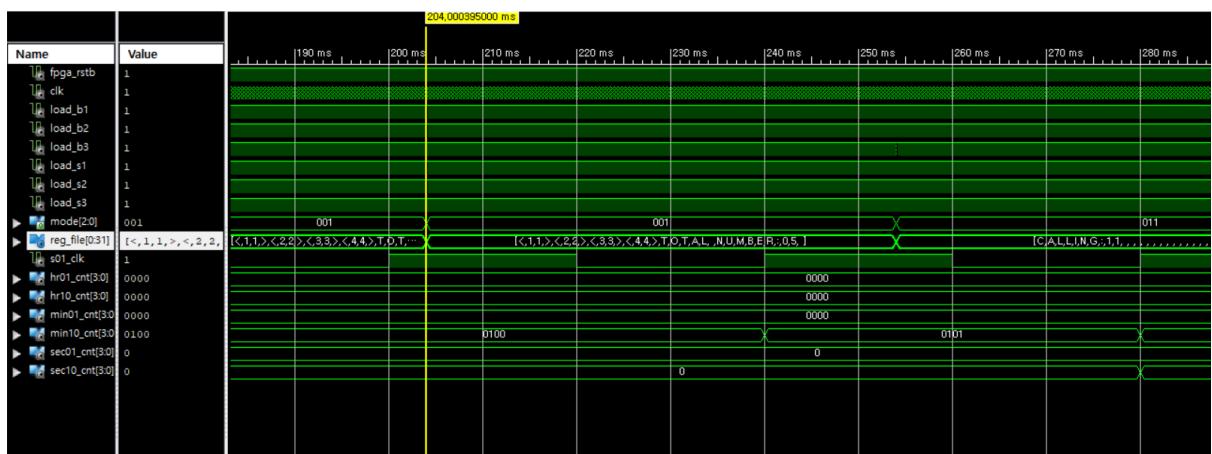
```



초기 상태는 mode 0 으로 어떠한 동작도 하지 않는 상태를 의미한다. Mode2로 전환해서 대기 리스트에 11을 추가한 뒤, mode1로 전환한 경우와 mode2로 전환해서 대기 리스트에 22를 추가한 뒤 mode1로 전환했을 때 LCD에 어떤 출력이 나타날지 reg\_file[0:31]을 통해 예측할 수 있다.



같은 방법으로 33과 44를 입력해서 추가했을 때의 출력 파형이다. Mode1로 다시 전환 시 33과 44가 추가된 것을 확인할 수 있다.



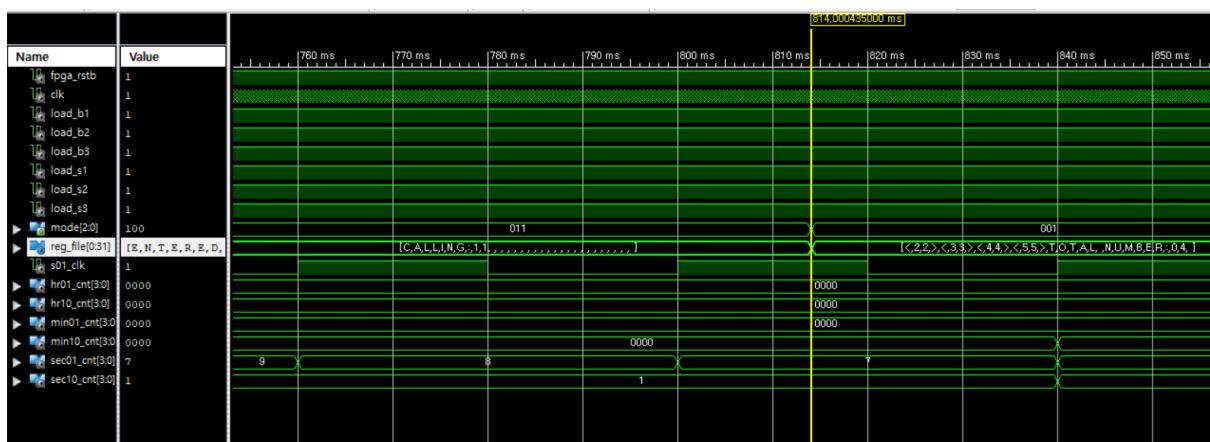
같은 방법으로 55가 추가되었을 때의 출력 파형이다. 이전과는 달리 55는 우선 순위가 낮기 때문에 LCD에 출력되지 않는 것을 확인할 수 있다.

위의 시뮬레이터 파형에서 나타나는 것처럼 테스트 벤치 상에서 입력된 데이터에 따라 정상적으로 mode1에서 정상적으로 현재 대기중인 인원의 명단이 LCD에 출력되는 것을 알 수 있다.

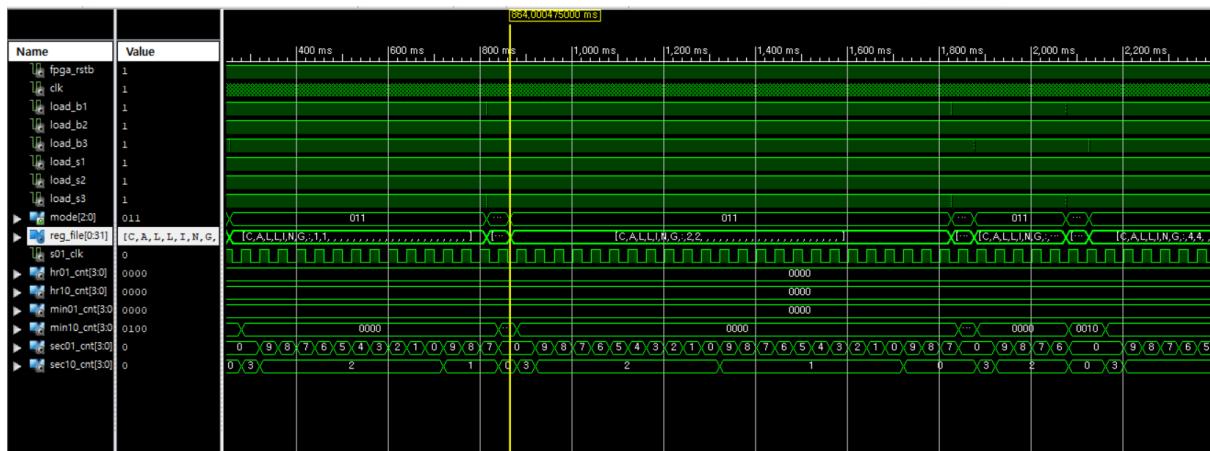
다음으로 검증할 기능은, 대기인원 명단의 초 세기 기능이다. 설계한 대기인원 리스트 기계를 이용해서 준비가 되었다면, 대기중인 인원들을 계속해서 호출할 수 있어야 한다. 이는 mode3에서 동작하는데, 그 출력 파형은 다음과 같다.



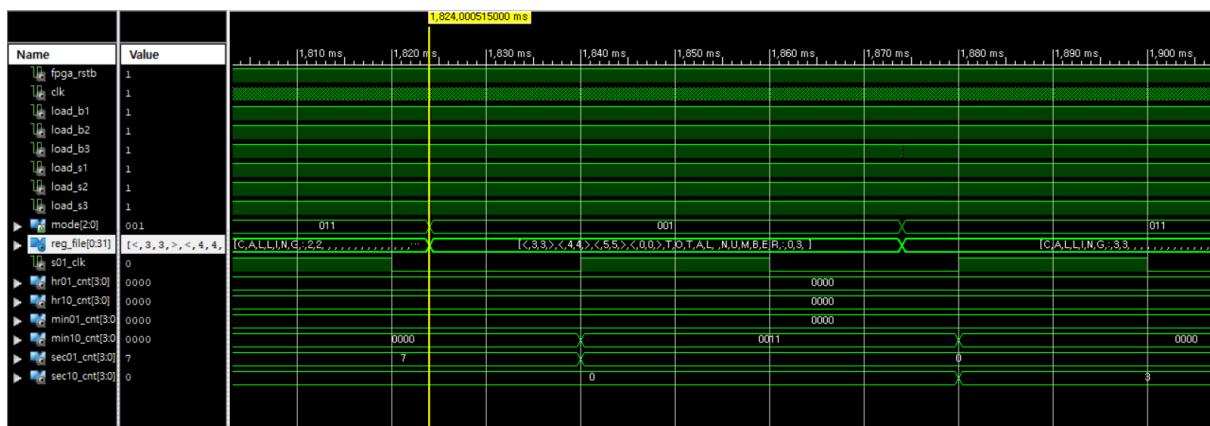
큰 스위치 버튼 3을 이용해서 대기 인원 호출 모드로 전환되었을 때의 출력 파형 결과이다. 초 세기 동작이 자릿수가 바뀌는 경우에도 정상적으로 동작하는지 위해 14초 이후에 입장하는 상황을 가정했다. 특별히 30초 초세기가 시작되는 부분과 모드3으로 전환되는 부분이 일치하지 않는 데, 이는 아래에서 좀 더 자세히 설명한다.



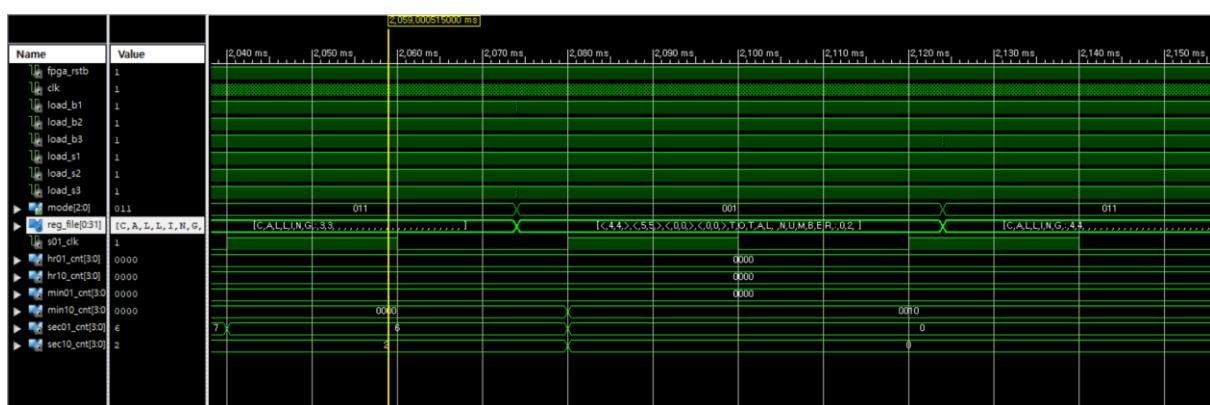
이후, 정상적으로 대기중이었던 인원이 입장하여 대기 인원 명단이 변화했는지 검증하기 위해 다시 대기중인 인원을 출력하는 mode1으로 전환한다. 대기 중이었던 11이 제거되고, 우선순위가 낮았던 55가 추가된 것을 확인할 수 있다.



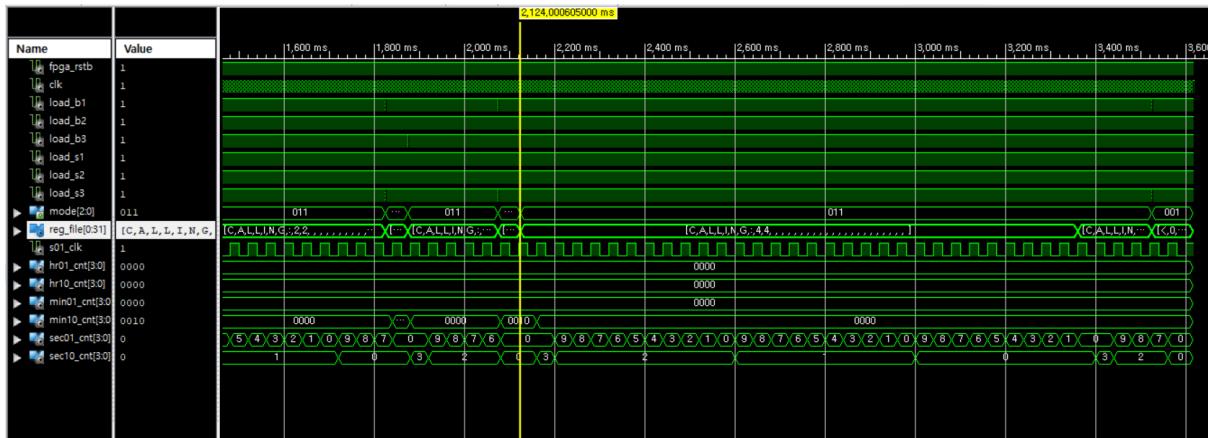
다음으로, 다시 모드3으로 전환하여 대기 우선순위가 가장 높은 22를 호출한다. 마찬가지로 자릿수 변화가 정상적으로 이루어지는지 확인하기 위해 24초 후에 입장하는 상황을 가정했다.



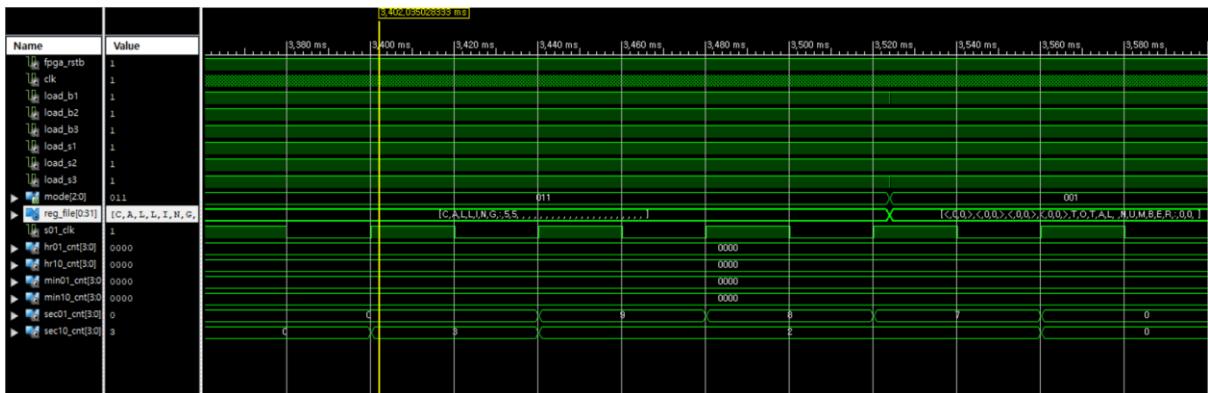
마찬가지로 다시 모드1로 전환하여 대기 명단에 변화가 있는지 확인한다. 22가 제거되고 55가 마지막으로 다음이 없으므로 00으로 채워지는 것을 확인할 수 있다.



33은 11과 22의 반복이기 때문에 5초 만에 입장하는 상황을 가정했고, 마찬가지로 다시 모드 전환 시 33이 제거된 것을 확인할 수 있다.



다음으로 대기 중인 인원이 호출에 응하지 않는 경우를 가정했다. 마찬가지로 30초 초 세기가 정상적으로 이루어지는 것을 확인할 수 있고, 30초 안에 입장이 되지 않는 경우 정상적으로 다음 번호인 55를 호출하는 것을 확인할 수 있다.



55번 역시 33번과 마찬가지로 동일한 경우의 반복이기 때문에 바로 입장처리를 한 후 모드를 전환하여 대기 인원 명단을 다시 확인 시, 44와 55 모두 제거되어 대기 인원 명단이 비어 있는 것을 확인할 수 있다.

정상적으로 mode3에서 현재 대기중인 인원들 중 가장 우선순위가 높은 인원을 화면에 띄우고 입장까지 남은 시간을 30초 초 세기에 들어간다. 대기 중인 인원은 이에 입장하면서 입장을 하는지 안 하는지 여부를 3번째 작은 스위치(push\_sw2)로 확인할 수 있고, 스위치를 눌러서 입장하는 경우, 그 인원이 정말 대기 명단에서 사라졌는지를 mode1로 전환해서 확인할 수 있다. 이와 같은 과정은 calling 22와 calling 33에서도 반복되고 있다. 특별히, calling 44의 경우는 대기 인원에는 있었지만 입장하지 않는 경우를 검증하기 위한 예로, 30초 초 세기가 끝날 때까지 작은 스위치 입력이 없을 때는 자동으로 다시 30초가 되고, 다음 대기 중인 인원을 호출하는 것을 확인할 수 있다.

특별히, 테스트 벤치 결과 출력 파형을 자세히 분석하면, 동작 모드를 결정하는 큰 스위치를 누른 순간과 30초 초 세기를 시작하는 시점이 약 0.8초 정도 일치하지 않는 것을 확인할 수 있는데,

이는 대기 인원을 호출할 때의 시간은 스위치를 눌러서 준비가 되었음을 알리는 업소 주인의 관점에서 계산하는 것이 아니라, 정확히 화면에 어떤 사람을 호출하는지를 나타낸 후에 시작하는 것이기 때문인 것을 알 수 있다.

### 3. Rov-lab 3000 kit Result



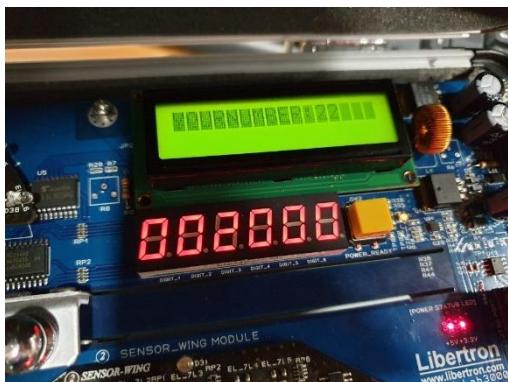
<PUSH1 을 눌러 아무도 등록되지 않은 대기 손님들의 번호가 표시>



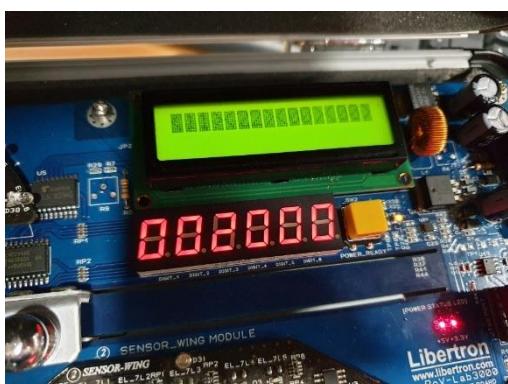
<11 번의 손님 등장 후 DIP 스위치를 이용하여 본인의 번호(11)을 입력 후 PUSH2 버튼을 눌러 본인의 번호와 대기시간 10 분 표시>



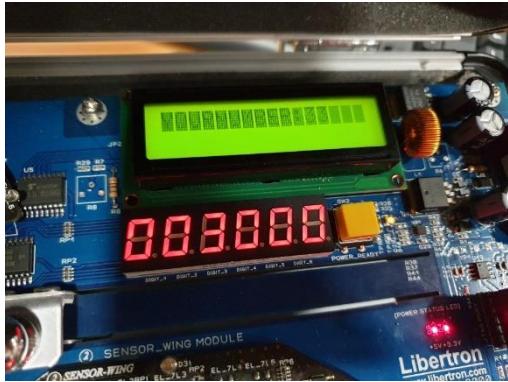
<PUSH1 버튼을 눌러 1 명의 대기자 번호를 출력 과 동시에 대기시간 표시 확인>



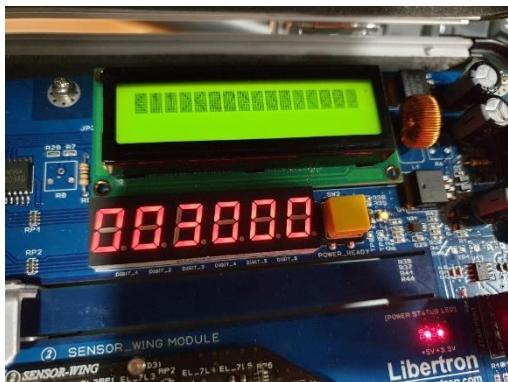
<22 번의 손님 등장 후 DIP 스위치를 이용하여 본인의 번호(22)을 입력 후 PUSH2 버튼을 눌러 본인의 번호와 대기시간 20 분 표시>



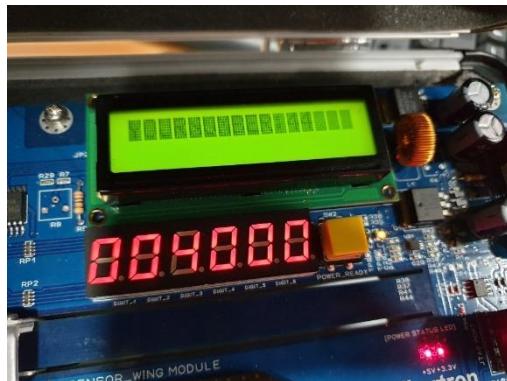
<PUSH1 버튼을 눌러 2 명의 대기자 번호를 출력과 동시에 대기시간 20 분 표시 확인>



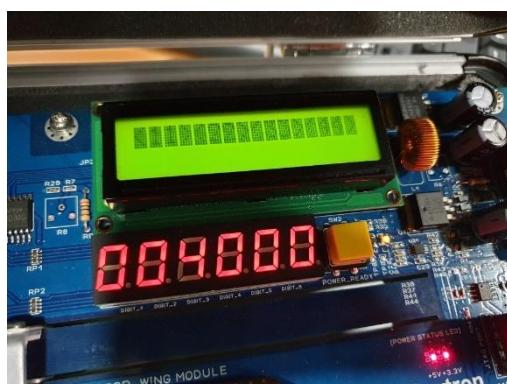
<33 번의 손님 등장 후 DIP 스위치를 이용하여 본인의 번호(33)을 입력 후 PUSH2 버튼을 눌러 본인의 번호와 대기시간 30 분 표시>



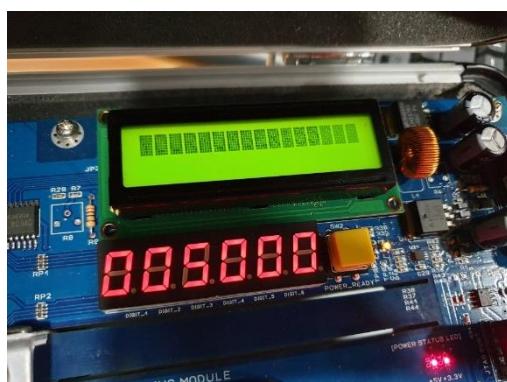
<PUSH1 버튼을 눌러 3 명의 대기자 번호를 출력과 동시에 대기시간 30 분 표시 확인>



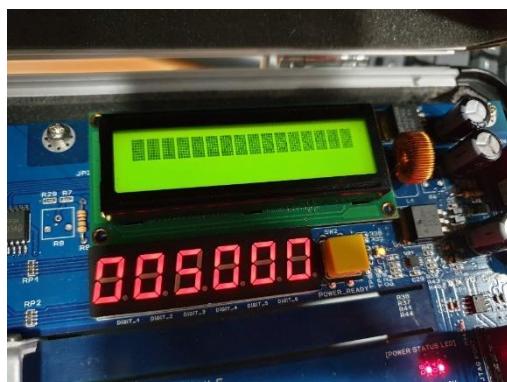
<44 번의 손님 등장 후 DIP 스위치를 이용하여 본인의 번호(44)을 입력 후 PUSH2 버튼을 눌러 본인의 번호와 대기시간 40 분 표시>



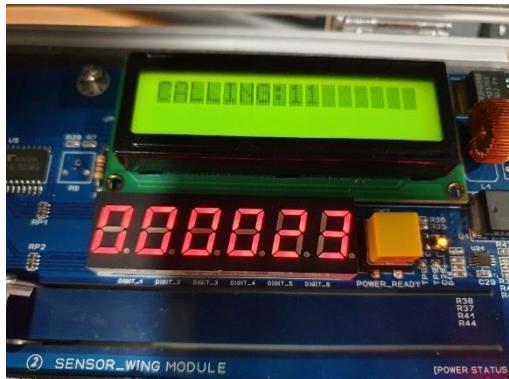
<PUSH1 버튼을 눌러 4 명의 대기자 번호를 출력과 동시에 대기시간 40 분 표시>



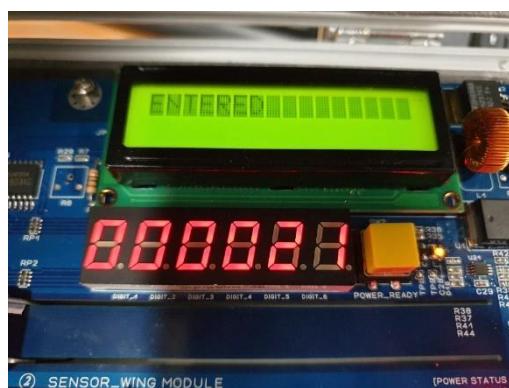
<55 번의 손님 등장 후 DIP 스위치를 이용하여 본인의 번호(55)을 입력 후 PUSH2 버튼을 눌러 본인의 번호와 대기시간 50 분 표시>



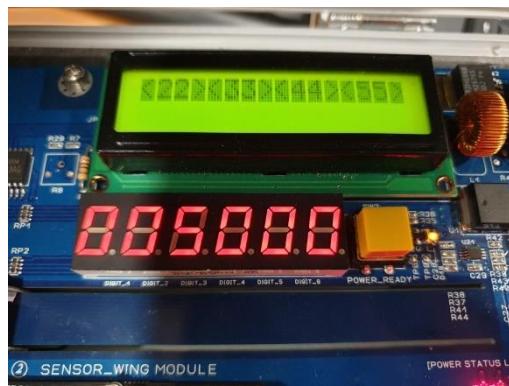
<PUSH1 버튼을 눌러 5 명중 먼저 온 4 명의 대기자 번호를 출력과 동시에 대기시간 50 분 표시>



<PUSH3 버튼을 눌러 처음의 손님 11 을 호출 하고  
30 초를 카운트하기 시작한다>



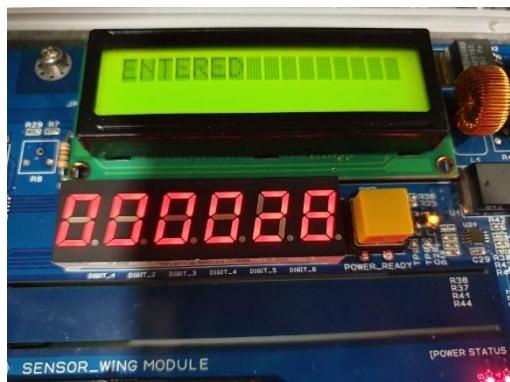
<9 초 뒤 손님이 도착하여 PUSH\_SW2(검은스위치 3 개  
중 마지막 스위치)을 눌러 등장확인 ENTERED 를  
LCD 에 출력한다>



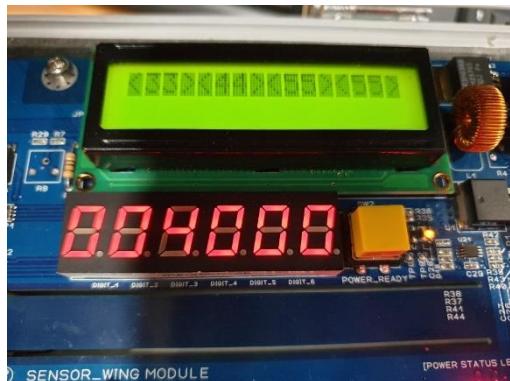
< 55 번의 손님의 본인의 업무를 오래 보기 위해 한번  
더 손님 등록을 한 후 PUSH1 버튼을 눌러 11 번의  
손님이 빠진 그 뒤의 4 명의 대기자 번호를 출력과  
동시에 대기시간 10 분\*3(22 번, 33 번, 44 번) +  
20 분(55 번) 총 50 분 표시 >



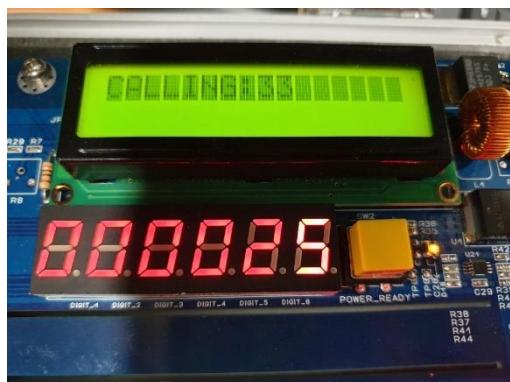
<PUSH3 버튼을 눌러 손님 22 를 호출하고 30 초를  
카운트하기 시작한다>



<2 초 뒤 손님이 도착하여 PUSH\_SW2를 눌러 등장확인 ENTERED를 LCD에 출력한다>



< PUSH1 버튼을 눌러 다음 3 명의 대기자(55 은 두 번의 업무) 번호를 출력과 동시에 대기시간 40 분 표시 >



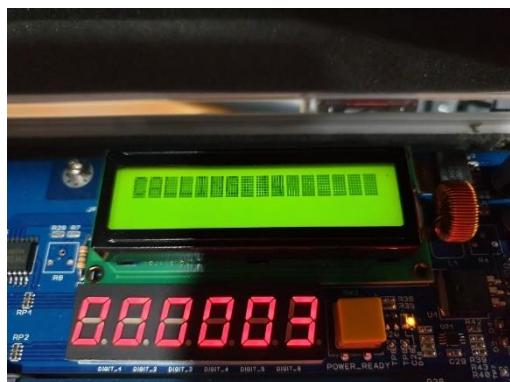
<PUSH3 버튼을 눌러 손님 33 를 호출 하고 30 초를 카운트하기 시작한다>



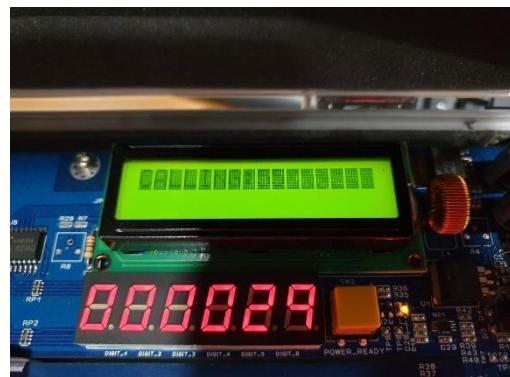
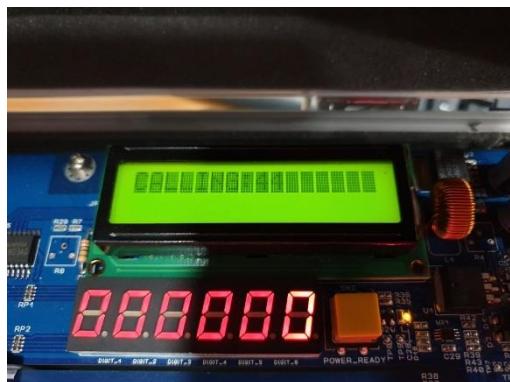
<17 초 뒤 손님이 도착하여 PUSH\_SW2를 눌러 등장확인 ENTERED를 LCD에 출력한다>



< PUSH1 버튼을 눌러 다음 2 명의 대기자(55 는 두 번의 업무) 번호를 출력과 동시에 대기시간 30 분 표시 >



<PUSH3 버튼을 눌러 손님 44 를 호출 하고 30 초를 카운트하기 시작한다>



<30 초 안에 44 번의 손님이 도착하지 않아 55 번의 손님이 호출 되며 30 초가 카운트 되기 시작한다>

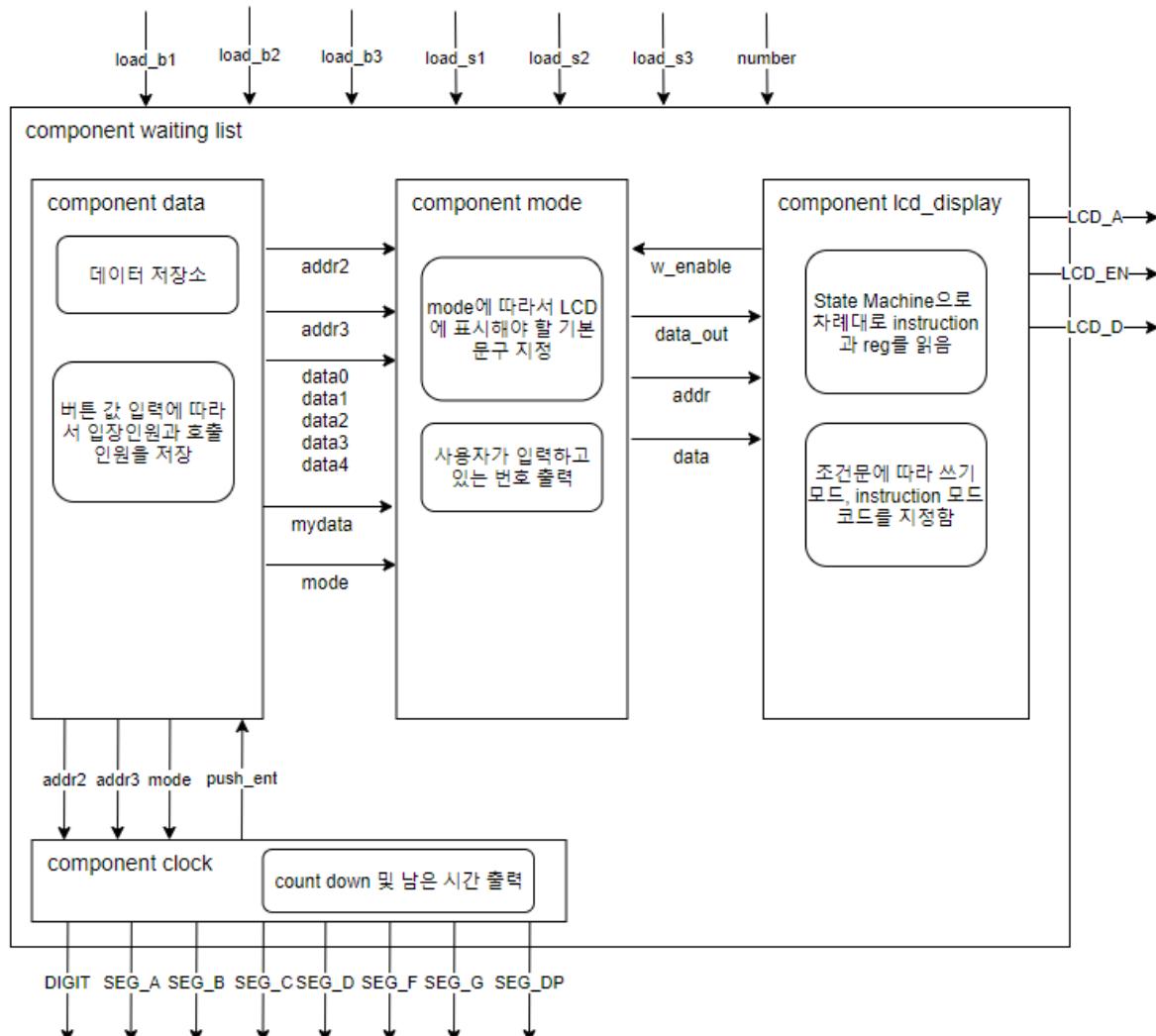


<12 초 뒤 손님이 도착하여 PUSH\_SW2 를 눌러 등장확인 ENTERED 를 LCD 에 출력한다>

이로써 모든 손님의 업무가 끝난다.

## 5. Result Analysis

### 1) 요약 및 총평



전반적인 입출력 도식도이다. reset 과 clock은 모든 component에서 사용되므로 생략하였다.

VHDL 코드 상 입출력 이름으로 지정하였다. load\_b1의 b는 big의 약자로, 편의상 큰 첫번째 스위치(push1)을 나타내기 위한 이름으로 지정하였다. load\_s1의 s는 small의 약자로 편의상 작은 검은색 스위치(push\_sw0)을 나타내기 위한 이름으로 지정하였다.

계획서에서 구상했던 내용을 대부분 완성하였고, 제안서 피드백에서 받은 내용(mode1, mode2에서 대기시간 출력) 내용또한 구현하였다. 아쉬운 점이 있다면 정확히 매뉴얼을 따라야 한다는 정 정도인 것 같다. 이에 대한 내용은 Discussion에서 다룰 것이다.

### 2) Discussion

전반적인 큰 기능들은 무리 없이 수행되었다. 그러나 몇 가지 특정한 입력을 하였을 때 문제가 생기는 기능이 있었다.

### 가) push 버튼 입력 문제

- 버튼 입력 기간이 clock 인식 기간보다 길어서 여러 번 인식되는 문제를 서로 다른 두 버튼의 입력 여부로 해결하였기 때문에 문제가 생길 때가 있었다. 예를 들면 cnt2(대기번호)의 같은 경우 push2(load\_b2)가 되는 순간을 인식하고 push\_sw1으로 reset 하는 기능을 하는데 만약 사용자가 등록하려고 push2를 누르고 실수로(아니면 번호표를 뽑지 않겠다고 생각해서) 번호를 입력하지 않고 push3를 누르게 되면 신호가 reset 이 되지 않고 cnt2 값에 오류가 생기는 상황이 발생했다.
- 실제로 해당 프로젝트에서는 버튼 순서에 오류가 생기면 카운트되는 것에 오류가 생기므로, 매뉴얼을 정확히 따라야 한다는 안내가 들어간다. 모든 입력은 순서대로 진행되어야 하고, 모든 입력을 한 후에는 push1(기본모드)로 돌아오면 오류가 생기지 않음을 발견했다.
- 이와 같은 상황을 근본적으로 해결하기 위해 고안한 방법은 몇 번이 입력되는 것과 상관없이 특정 입력이 들어오면 처음의 rising edge를 인식해서 clock 주기만큼만 신호를 유지하고 다시 초기값으로 돌아오는 방법이다. 하지만 실제로 실행에 옮기는 데 실패했다. 만약 가능하다면 간단하게 클락 상승 시점에 입력이 들어오면  $\text{cnt2} \leq \text{cnt2} + 1$  을 하는 방법으로 설계를 할 수 있을 것 같다.

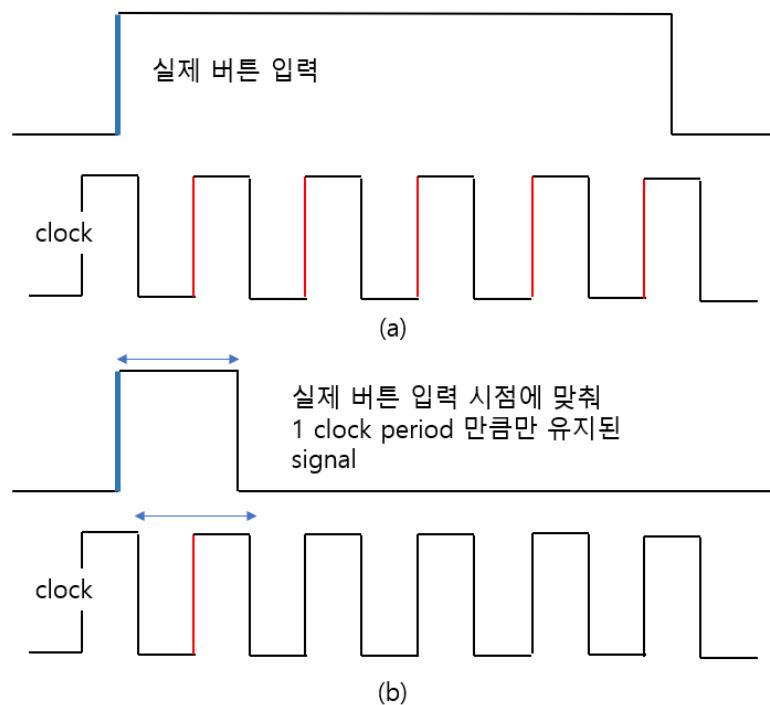


Figure 1 (a)에서는 총 5번의 clcok rising이 일어나고, (b)에서는 한 번 인식된다.

#### 나) LCD 아랫줄 출력 문제

- 처음에 LCD 아랫줄이 출력되지 않아 instruction에 오류가 생겼음을 발견. line change 있는 줄에서 쓰기 모드가 활성되어 있는 것을 읽기 모드로 바꾸고 실행하였을 때에는 실행이 되었다.
- 하지만 같은 코드로 후에 해 보니 될 때도 있고 안 될 때도 있어 원래 1 번째 줄에 있던 내용보다 2 번째 줄에 있던 내용이 중요하다면 위 아래 배치를 바꾸었다. 원인을 찾아 수정하고 싶었지만 이 또한 실패하였다.

## 6. 결론(역할 분담 내용 기재)

- 강상훈(10/10) : 초반 베타 버전의 코드를 작성함. 7-segment display 코드를 맡았으나 실행이 되지 않아 보고서 vhdl source 및 설명, isim simulation result, kit result 부분을 작성함.
- 윤채원(10/10) : 기본적인 코드 빠대 설계(waiting list, mode, lcd\_display, data), 후반에 최준우, 강상훈님이 설계하지 못한 mode 1, 2 대기시간 7-segment display 설계
- 최준우(8/10) : 초반에 component 구조 잡는 데 의견을 제시함. 7-segment display 코드를 맡았으나 실행이 되지 않아 보고서 test bench 실행을 맡음.
- 황온유(10/10) : waiting list, mode, lcd\_display, data 코드 계획서 내용으로 구현 및 FPGA 실행 확인. 윤채원님이 작성하신 코드에 주석 추가. 7-segment mode 3에서 카운트 다운 코드 설계

## 7. 참고문헌

논리회로설계실험 ppt(FSM, Memory, Digital\_clock, LCD, Calculator)

Charles H. Roth, J., & Kinney., L. L. (2008). *Fundamentals of Logic Design* (SEVENTH ED)(pp.16). Global Engineering: Tim Anderson.