Term Project
Digital Alarm Clock

2017103780 전자공학과 권용환
2018104121 전자공학과 황지준

1. 요구사항 및 성능개선

첫 번째 요구사항인 Reset의 추가의 경우 다음과 같이 해결하였다. Digital_Clock에서 출력하기 위한 시간 data들은 time_blk에서 결정된다. 따라서 time_blk에 리셋 입력을 추가하였다. 동기식 리셋 방식으로 v_hour, v_min, v_sec, s_hour, s_min을 모두 0으로 초기화하도록 하였다. 또한, Decrease 기능을 추가하였는데 Digital_Clock에서 decrease, dec_hour, dec_min을 추가하여 Increase와 같은 방식으로 구현하였다. 마지막으로, 시간의 경우 23, 분의 경우 59를 플로팅하지 못하는 문제를 해결하는 과제가 주어졌는데 이는 지난 실험 11에서 해결하여 레포트를 제출하였기 때문에 여기서는 성능 개선에서 간단하게 다루도록 한다.

2. 성능개선

이번 프로젝트에서는 DE2 보드를 이용하여 디지털 시계를 설계하고, 교재에서 제시한 디지털 시계 코드의 단점을 보완하고, 추가적인 기능을 추가하였다. 먼저, 교재에 제시된 시계가 가지고 있는 문제점은 다음과 같다. Increase, Decrease 버튼을 눌렀을 때, 너무 빠른 속도로 시간이 변한다. 이 경우, 변화 속도를 제어하기 위해 Increase 신호를 만들어 내는 카운터의 bit를 늘리는 것이 가장 단순한 해결 방법이다. 그러나, 이번 프로젝트에서는 정확하게 한 번, 버튼이 눌리면 한 번만 올라갈 수 있도록 버튼 입력에 대한 pulse 출력을 만드는 pulse_FSM을 이용하여 설계하였다. pulse_FSM은 실험 7에서 자세히 다루었기 때문에 이번 레포트에서는 특별히 기술하지 않는다. pulse_FSM 모듈은 입력의 길이에 관계없기 입력에 대하여 한 클럭의 pulse 출력을 발생시킨다. 이 pulse 출력을 Increase, Decrease로 한다. 이때, 입력은 각각 KEY3와 KEY2이다. 또한, 시간 설정 모드와 클럭 모드일 때, 두 모드를 Transition하면서 발생할 수 있는 시간 차이 문제가 존재할 수 있다. 실험 11에서 다룬 것처럼 시간 설정 모드일 때와 분 설정 모드일 때, 서로의 v_hour, v_min과 s_hour, s_min 의 data를 교환하는 방식을 통해 둘의 차이를 제거하였다. 실험 11의 결과보고서에 최종적인 출력인 hour, min에 v_hour, v_min을 사용할 수도 있

다고 기술하였으나 v_hour와 v_min은 1hz 클럭을 사용하고, s_hour와 s_min은 1Mhz 클럭을 사용하기 때문에 시간 설정 모드일 때 변화시킨 값이 v_hour와 v_min에 셋업 되는데 시간이 필요하다. 따라서 s_hour와 s_min을 hour와 min으로 출력하도록 하여야 한다.

3. 추가기능

이번에 추가한 기능은 알람 기능이다. mode의 경우 2bit로 2'b11은 사용하지 않았다. 2'b11일 때를 알람 설정 모드로 한다. 알람 설정 모드에서 시간과 분을 선택한다. 알람 설정 모드를 벗어나면 현재 시간이 출력된다. 버튼이 부족하기 때문에 스위치를 이용하여 increase와 decrease를 선택할 수 있도록 하였다. 또한, 스위치를 이용하여 Alarm 리셋 기능을 추가하였다. Alarm은 59초 동안 지속된다. 알람이 울리는 동안 LEDR[17]에 불이 들어온다. 또한, 알람 설정을 위해 Altera 사가 제공하는 IP인 Altpll과 Rom을 이용하여 WM8731로 출력하기 위한 XCK와 BCLK을 생성하고, 음원 파일을 샘플링하여 16bit word size로 Rom에 저장하였다. I2C protocol과 WM8731에 대한 소개는 각각 Section 4.와 Section 5.에서 다루며 코드 구현에 대한 부분은 Section 6.에서 다룬다.

4. I2C protocol

I2C는 통신 프로토콜의 일종이다. Master와 Slave를 오가며 data를 주고 받는다. 이때, 주고 받는 data에 의해서 이번 프로젝트에서는 WM8731의 모드를 선택하도록 할 것이다. I2C는 다음과 같이 규칙을 따른다.

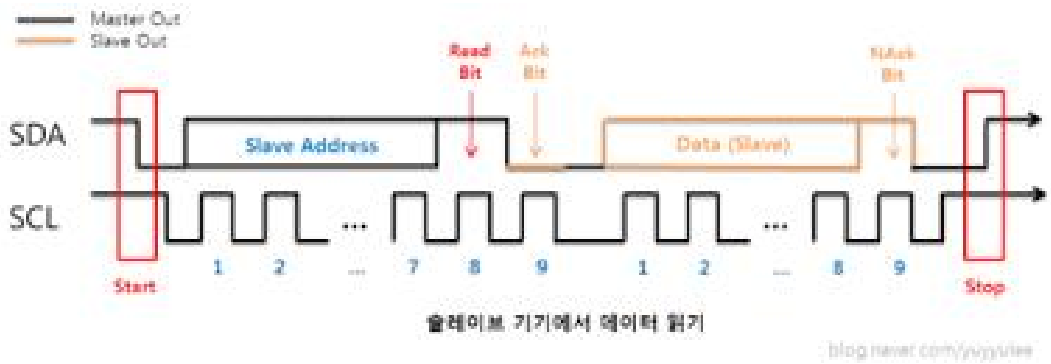


그림 1

그림 1에 제시된 것처럼 SDA가 1에서 0으로 내려간 후에 SCL이 1에서 0으로 내려가면서 Start 가 된다. 이때, SDA는 7bit의 Slave Address, 1bit의 Read/Write bit, 1bit의 ACK bit, 8bit의 Register bit, 1bit의 ACK bit, 8bit의 data, 1bit의 ACK bit를 전송한다. 그 후, SCL이 1로 올라간 후 SDA가 1로 올라가면 통신은 끝나게 된다. DE2의 경우 SDA는 SDAT로, SCL은 SCLK으로 명명하고 있다. 또한, 데이

터의 교신은 SCLK이 0일 때 이루어진다. SDAT의 default 상태는 High z이며, Master에서 Slave로 7bit의 Slave Address(시작 주소)를 전송한 후, 해당 Address를 가진 Slave가 응답하면 SDAT 0이 Slave에서 Master로 전송된다. 8bit의 Register Adress의 경우 Slave내부의 데이터 영역을 선택하기 위해 Slave로 전송되며, 해당 Register가 사용 가능한 경우 SDAT 0을 Slave에서 Master로 전송한다. 마지막으로, 8bit의 data를 Slave에 전송한다. 이 8bit의 data가 Slave에 정상적으로 전송되면 Slave에서 Master로 SDAT 0이 전송된다. 그림 2에 동작 시퀀스가 비교적 잘 나타나 있다.

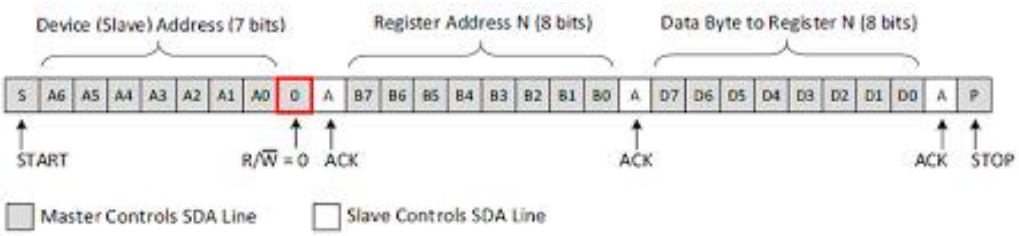


그림 2

I2C는 UART처럼 여러 Device에서 사용된다. Master 하나가 동시에 127개의 Slave와 교신할 수 있으며, UART와 다르게 보드레이트를 설계자가 직접 선택할 수 있다. UART보다 조금 컨트롤 하는 것이 어렵지만 위의 내용을 숙지하여, 각 Slave마다 어떤 동작을 할지 datasheet에 자세히 기술되어 있기 때문에 Slave의 시작 주소 값을 설정하고 통신 속도를 조절하여 효율적으로 사용할 수 있다.

5. WM8731

DE2의 탑재된 WM8731은 8khz ~ 96khz 사이에서 sampling rate을 갖는다. 기기의 동작 주파수는 12.5Mhz로 DE2의 경우 XCK로 명명한다. BCLK과 DACLRCLK는 sampling frequency와 관련되어 있는데 그림 3의 waveform table을 통해서 설명하도록 한다. 48.8khz의 sampling frequency를 기준으로 하는 waveform table이다. sampling 된 data는 기본적으로 DAC이든, ADC이든 관계없이 16bit를 갖는다. Left data와 Right data가 있다. BCLK의 negedge마다 data는 1bit씩 전달된다. 따라서 그림 3처럼 48.8khz의 sampling frequency를 갖는 경우 DACLRCLK는 48.8khz이며, DACLRCLK의 한 주기 동안 BCLK은 총 64번 진동한다. 그림 3의 Wave form table은 Petter K¨allstr¨om, Mario Garrido 의 TSIU03: Lab 4 – Audio Codec에서 인용하였다. 해당 자료에서는 Left data와 Right data 사이는 쓰지 않는 영역으로 하였다. 이번 프로젝트의 경우 data가 남는 부분을 없애고자 BCLK이 32번 진동했을 때, DACLRCLK이 한 주기를 갖도록 하여 97.6khz의 속도를 갖도록 한 결과 Alarm은 2배속으로 출력되었다. 다시, 48.8khz로 낮추어 빈 부분에 Left data와

Right data를 한 번 더 쓰도록 한 결과 컴퓨터를 통해 확인한 음원과 크게 차이가 없어 그대로 사용하였다. 그러나, 해당 자료에서는 빈 부분의 data는 사용하면 안 된다고 한다. 우리는 mono 모드로 하였기 때문에 큰 차이가 없었지만 stereo 모드인 경우는 data가 비는 부분을 만들어 줄 필요가 있는 것으로 보인다. DE2의 램의 경우 8Mbyte의 매우 작은 저장 공간을 가지고 있기 때문에 높은 sample rate로 출력한 음원 파일은 사용할 수 없었다. 따라서 8khz에서 샘플링을 하였으며, Matlab을 통해 구현하였다.

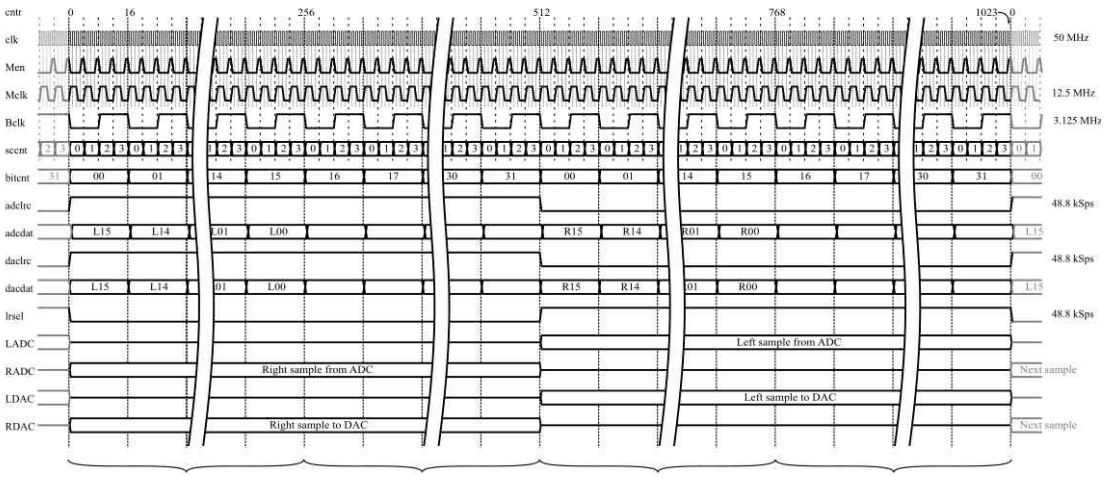


그림 3

다음은 Matlab을 통해 진행한 Sampling 과정이다.

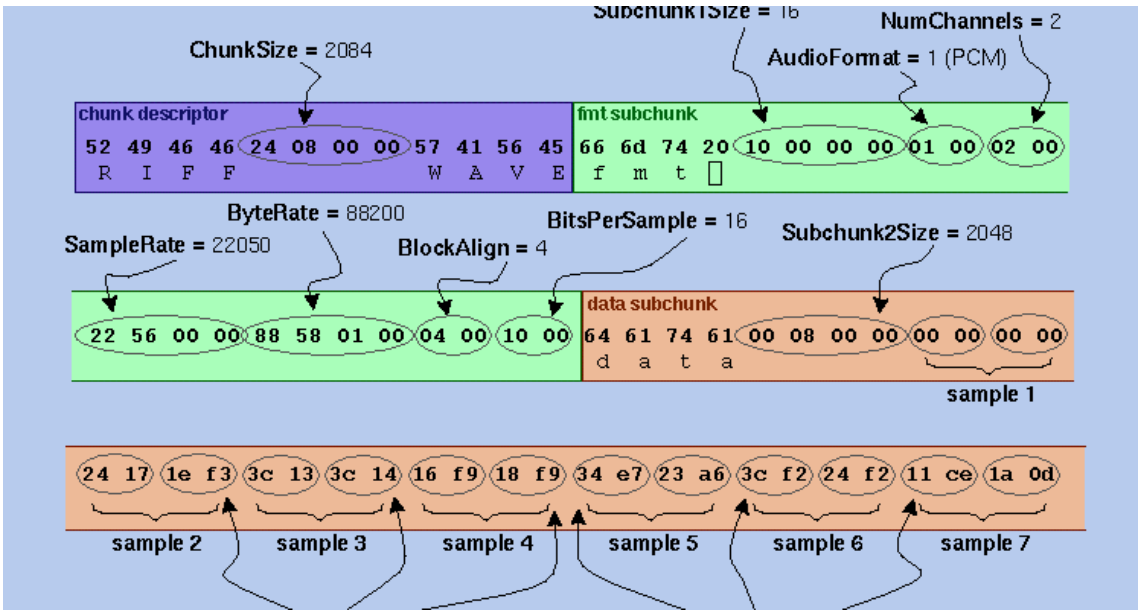


그림 1. Wav file format

Rom에 음원 데이터를 적재시키기 위해서는 우선 음원 파일의 format을 확인해야 한다. 사용할 음원의 확장자는 wav이며, wav 파일의 data는 상단의 그림에서 붉은색 화살표로 표기된 부분에서부터 시작한다. 앞의 bit들은 해당 파일이 어떤 파일인지 알려주는 헤더 파일이다. RIFF, fmt, sample rate, ADC 방식 등 다양한 정보가 담겨있지만, 이번 실험에서는 필요없는 데이터들이므로 해당 헤더 파일은 전부 지우고 데이터 스트림만 가져오도록 한다.

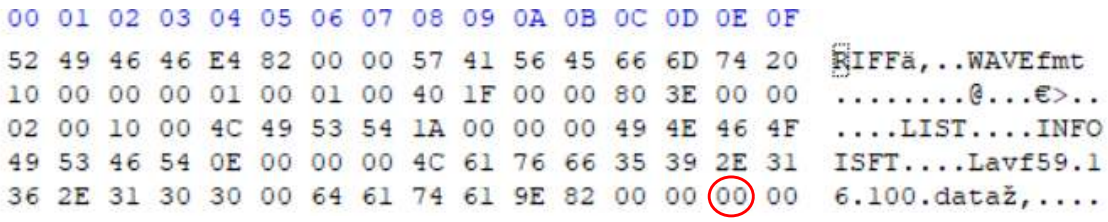


그림 2 HEX Editor

Wav 파일을 HEX editor로 열게 되면 다음과 같이 Wav 파일에 담긴 데이터 스트림은 붉은 색 원으로 표기된 부분부터이다. 이전 부분의 정보들은 모두 지운 뒤, hex 파일로 저장한다. 앞서 수정한 Hex 파일을 MATLAB으로 불러와 ROM에 적재할 수 있도록 바꿔주는 코드를 그림 6에 첨부한다.

```
%%% ver 2.
clear all;
clc;
close all;
[baseName, folder] = uigetfile('*.*');
fullFileName = fullfile(folder, baseName);
fid= fopen(fullFileName);
fid2 = fopen( 'mem_8K_mod.mif', 'wt' );
A=fread(fid,inf, 'uint16','ieee-le'); % little endian data

L = length(A);

fprintf(fid2,'%s%u%s','DEPTH = ',L,';');
fprintf(fid2,'%c%n','');
fprintf(fid2,'%s%u%s','WIDTH = 16;');
fprintf(fid2,'%c%n','');
fprintf(fid2,'%s','ADDRESS_RADIX = HEX;');
fprintf(fid2,'%c%n','');
fprintf(fid2,'%s','DATA_RADIX = HEX;');
fprintf(fid2,'%c%n','');
fprintf(fid2,'%s','CONTENT');
fprintf(fid2,'%c%n','');
fprintf(fid2,'%s%n','BEGIN');
fprintf(fid2,'%c%n','');

for i=1:L
 fprintf(fid2,'%X', (i-1));
 fprintf(fid2,'%s', ' : ');
 fprintf(fid2,'%04X', A(i));
 fprintf(fid2,'%s%n', ';');
end

 fprintf(fid2,'%s%n', '');
 fprintf(fid2,'%s', 'END;');

 fclose(fid);
 fclose(fid2);
```

그림 6 MATLAB 코드

6. Verilog 구현

베릴로그로 구현하기 위해 몇가지 작업이 필요하다. WM8731의 동작 주파수인 12.5Mhz의 경우 Quartus II의 Megawizard function을 이용하여 만들어 내는 것이 가능하다. BCLK의 경우 우리가 선택한 sampling frequency인 8khz의 64배 이므로 512khz가 되어야 한다. 그러나, Megawizard function에서는 이를 만들 수 없다. 따라서 다음과 같이 Megawizard function으로 만든 코드를 수정하여 510khz의 BCLK을 생성하였다.

```
36  // synopsys translate_off
37  `timescale 1 ps / 1 ps
38  // synopsys translate_on
39  module testCLK (
40      inclk0,
41      c0,
42      c1);
43
44      input     inclk0;
45      output    c0; // 12.5Mhz
46      output    c1; // 510khz
47      reg c1 = 1'b0;
48      reg [5:0] cnt_c0 = 6'd0;
49      always @(posedge inclk0) begin
50          if(cnt_c0 == 48) begin
51          c1 = ~c1;
52          cnt_c0 = 0;
53          end
54          else cnt_c0 = cnt_c0 + 1;
55      end
```

그림 7

6bit의 카운터를 이용하여, cnt_c0가 48이 될 때마다, c1을 진동시킨다. 이 경우 inclk0의 frequency를 98배 낮춘다. 따라서 대략 510khz가 만들어진다.

mem_8K_mod.mif

```
1   DEPTH = 14969;
2   WIDTH = 16;
3   ADDRESS_RADIX = HEX;
4   DATA_RADIX = HEX;
5   CONTENT
6   BEGIN
7   0 : 0000;
8   1 : 0000;
9   2 : 0000;
10  3 : 0000;
```

그림 8

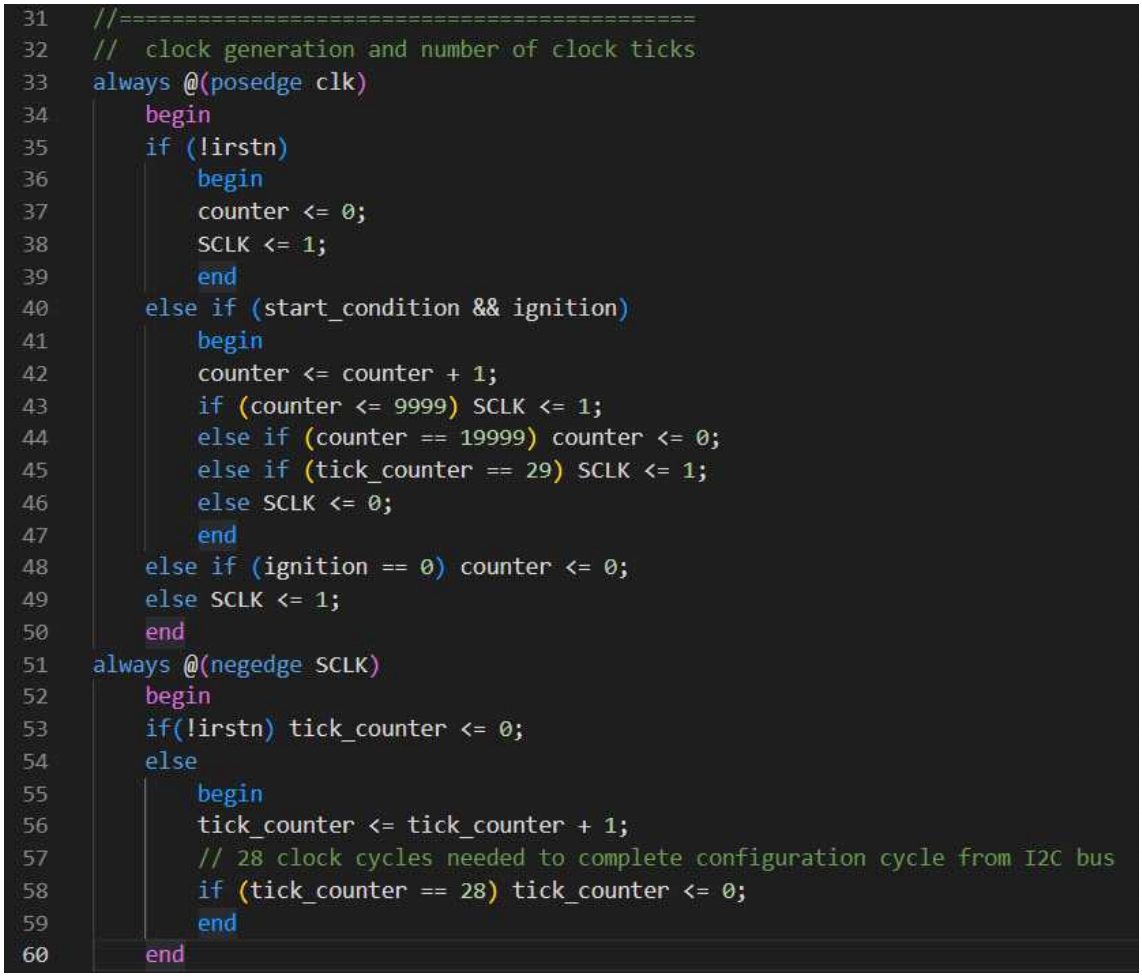
우리가 사용할 데이터 샘플의 DEPTH는 14969이다. 이 부분에 대한 언급은 잠시 후에 기술하도록 한다. 먼저, WM8731을 동작시키기 위한 Audio_codec의 설명에 앞서 I2C protocol 모듈에 대해 먼저 다루도록 한다.

I2C_Protocol.v

```
1   module I2C_Protocol(
2       input clk,
3       input irstn,
4       input ignition,
5       input [15:0] MUX_input,
6       inout SDAT,
7       output reg finish_flag,
8       output reg [2:0] ACK,
9       output reg SCLK
10      );
11  //=========================================
12  // parameter
13  //to generate 2.5kHz clock
14  reg [13:0] counter;
15  //to measure number of clock ticks
16  reg [4:0] tick_counter;
17  reg serial_data;
18  reg start_condition;
19  reg out;
20  assign SDAT = (out)? serial_data: 1'bz;
```

그림 9

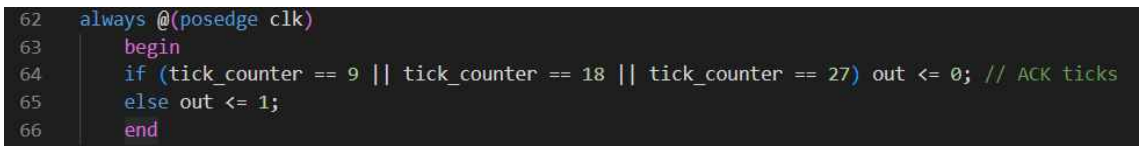
ignition은 SCLK을 생성하기 위한 trigger 신호다. MUX_input은 Register address와 data를 담은 16비트 입력 신호다. SDAT를 통해 Master와 Slave는 신호를 주고받는다. finish_flag의 경우 통신이 끝날 때마다 1을 출력한다. ACK는 Slave의 Address가 확인되거나 Slave의 data의 영역이 선택되거나 Slave에 data가 정상적으로 들어갈 때마다 1을 출력하며, 20번 라인에서 알 수 있듯 SDAT의 default상태는 high z로 open된 상태다. 나머지 신호는 잠시 후에 기술한다.



```
31  //=============================================
32  //  clock generation and number of clock ticks
33  always @(posedge clk)
34      begin
35      if (!irstn)
36          begin
37          counter <= 0;
38          SCLK <= 1;
39          end
40      else if (start_condition && ignition)
41          begin
42          counter <= counter + 1;
43          if (counter <= 9999) SCLK <= 1;
44          else if (counter == 19999) counter <= 0;
45          else if (tick_counter == 29) SCLK <= 1;
46          else SCLK <= 0;
47          end
48      else if (ignition == 0) counter <= 0;
49      else SCLK <= 1;
50      end
51  always @(negedge SCLK)
52      begin
53      if(!irstn) tick_counter <= 0;
54      else
55          begin
56          tick_counter <= tick_counter + 1;
57          // 28 clock cycles needed to complete configuration cycle from I2C bus
58          if (tick_counter == 28) tick_counter <= 0;
59          end
60      end
```

그림 10

start_condition과 ignition에 1이 들어오면, SCLK을 2.5khz로 생성한다. tick_counter는 SCLK의 negedge마다 1씩 증가하여 28까지 증가한다. tick_counter에 따라서 Master와 Slave는 1bit씩 data를 주고 받는다.



```
62  always @(posedge clk)
63      begin
64      if (tick_counter == 9 || tick_counter == 18 || tick_counter == 27) out <= 0; // ACK ticks
65      else out <= 1;
66      end
```

그림 11

tick_counter가 9, 18, 27일 때, out이 0이 되어 SDAT는 high z가 된다. high z 상태인 동안 Slave에서 Master로 1의 입력 pulse가 들어왔다 나갈 것이고 이 부분은 후에 기술하도록 한다.

```
68  always @(posedge clk)
69      begin
70          case(tick_counter)
71          0://SDAT will be high for 60us then low until SCLK goes high
72              begin
73              ACK <= 3'b000;
74              finish_flag <= 0;
75              start_condition <= 1;
76              serial_data <= 1;   //SDAT idle condition
77              if (counter > 3000) serial_data <= 0; // was 3000
78              end
79          1://push 0x34 address through I2C data bus, 0x34 = 00110100
80            //push 0
81              begin
82              serial_data <= 0;
83              end
84          2://push 0
85              begin
86              serial_data <= 0;
87              end
88          3://push 1
89              begin
90              serial_data <= 1;
91              end
92          4://push 1
93              begin
94              serial_data <= 1;
95              end
96          5://push 0
97              begin
98              serial_data <= 0;
99              end
100         6://push 1
101             begin
102             serial_data <= 1;
103             end
```

그림 12

앞서 서술한 것처럼 tick_counter의 상태에 따라서 data를 송수신할 것이다. tick_counter가 0일 때, ACK, finish_flag, start_condition, serial_data를 초기화한다. serial_data가 0으로 떨어지는데 out이 1일 때는 serial_data가 SDAT가 되기 때문에 SDAT가 1에서 0으로 떨어지고, SCLK이 1에서 0으로 떨어지면 START를 하기 위한 것이다. WM8731의 시작 주소와 Read signal인 1'b0을 합치면 0x34이다. 따라서 8'b0011_0100을 Slave로 보낸다.

```
112         9://wait for ACK
113             begin
114             ACK[0] <= !SDAT; //was SDAT
115             end
116         10://start sending register address, 7 bits each address
117             //push MSB first (MUX_input[15])
118             begin
119             serial_data <= MUX_input[15];
120             end
121         11://push MUX_input[14]
122             begin
123             serial_data <= MUX_input[14];
124             end
125         12://push MUX_input[13]
126             begin
127             serial_data <= MUX_input[13];
128             end
129         13://push MUX_input[12]
130             begin
131             serial_data <= MUX_input[12];
132             end
133         14://push MUX_input[11]
134             begin
135             serial_data <= MUX_input[11];
136             end
137         15://push MUX_input[10]
138             begin
139             serial_data <= MUX_input[10];
140             end
141         16://push MUX_input[9]
142             begin
143             serial_data <= MUX_input[9];
144             end
```

그림 13

tick_counter가 9가 되면 out은 0이 되고, SDAT는 high z가 된다. 이때, Slave에서 Master로 SDAT를 0으로 보내기 때문에 ACK[0]가 !SDAT를 받고 있으므로 1이 된다. ACK[1]과 ACK[2]도 같은 방식이다. 따라서 정상적으로 수신이 되어 통신이 끝났다면 ACK는 3'b111이 되어야 한다.

```
186         27://wait for ACK
187             begin
188             ACK[2] <= !SDAT;
189             end
190         28://bring serial_data to low and stop condition
191             begin
192             serial_data <= 0;
193             finish_flag <= 1;
194             if (counter > 3000 && SCLK == 1 /*|| counter == 0*/) serial_data <= 1;
195             end
196         endcase
197     end
```

그림 14

190라인을 보면, serial_data를 0으로 초기화하여 SDAT를 0으로 유지하다가 SCLK이 1이 된 후에 serial_data가 1이 되도록 하여 STOP상태를 만든다. 이때, finish_flag가 1이 되어 출력 되는데 현재 always는 50Mhz클럭을 기준으로 하고 잇기 때문에 매우 빠르게 finish_flag가 출력된다. 이제 Audio_Codec 부분을 볼 것이다. 다만, 이번 프로젝트에서는 Audio_codec 모듈을 따로 두지 않고 Digital_Clock 안에서 만들었다.

```
92      // Audiocodec
93      //=============================================
94  // parameter
95  reg [3:0] counter; // transition mode
96  reg counting_state; // I2C, Read ROM
97  reg ignition; // Read ROM -> ignition : 0
98  reg read_enable;
99  reg [15:0] MUX_input;
100 reg [17:0] read_counter;
101 reg [3:0] ROM_output_mux_counter;
102 reg [5:0] DAC_LR_CLK_counter;
103 wire [15:0]ROM_out;
104 wire finish_flag;
105 assign DAC_DATA = (trg_alarm) ? ROM_out[15-ROM_output_mux_counter]: 0;
```

그림 15

counter는 I2C 통신이 끝난 후 WM8731로 ROM의 data를 넣어주기 위해 구분한 STATE transition에 이용된다. ignition의 경우 I2C 통신이 끝나면 바로 0이 되어 SCLK이 동작하지 않도록 한다. read_enable의 경우 I2C 통신이 끝나면 1이 되어 ROM의 clkenable신호로 이용된다. read_counter의 경우 ROM의 address로 사용된다. ROM_output_mux_counter는 BCLK의 negedge마다 0~15를 오가며, ROM에 저장된 16비트의 data를 WM8731로 보내기 위해 사용된다. DAC_LR_CLK의 sampling frequency를 나타낸다. DAT_DATA는 trg_alarm이라는 trigger신호가 들

어올 때마다 ROM의 16bit의 신호를 DAC_DATA에 실어 보내기 위해 사용되는데 trg_alarm의 경우 time_blk에서 보내주는 신호로 후에 기술한다.

```
107   // Instantiation section
108   I2C_Protocol I2C(
109       .clk(clk50),
110       .irstn(rstn),
111       .ignition(ignition),
112       .MUX_input(MUX_input),
113       .ACK(ACK_LEDR),
114       .SDAT(SDAT),
115       .finish_flag(finish_flag),
116       .SCLK(SCLK)
117       );
118   testCLK testCLK_inst(
119       .inclk0(clk50),
120       .c0(XCLK),
121       .c1(BCLK)
122       );
123   testROM testROM_inst(
124       .address(read_counter),
125       .clken(read_enable),
126       .clock(DAC_LR_CLK),
127       .q(ROM_out)
128       );
```

그림 16

지금까지 설명한 것처럼 initiate되어 있는 것을 알 수 있다.

```
129  //=========================================
130  // Switch registers
131  always @(posedge finish_flag) counter <= counter + 1;
132  always @(posedge SCLK) begin
133          case(counter)
134          0: MUX_input <= 16'h0460;
135          //1: MUX_input <= 16'h0660;
136          //MUX_input[15:9] register address, MUX_input[8:0] register data
137          // MUX_input <= 16'h1201; // activate interface
138          // MUX_input <= 16'h0460; // left headphone out
139          // MUX_input <= 16'h0C00; // power down control
140          // MUX_input <= 16'h0812; // analog audio path control
141          // MUX_input <= 16'h0A00; // digital audio path control
142          // MUX_input <= 16'h102F; // sampling control
143          // MUX_input <= 16'h0E23; // digital audio interface format
144          // MUX_input <= 16'h0660; // right headphone out
145          // MUX_input <= 16'h1E00; // reset device
146          endcase
147      end
```

그림 17

finish flag가 뜰 때마다 counter가 1씩 증가하여 SCLK의 posedge에 MUX_input을 바꾸는데 앞서 I2C에서 tick_counter는 SCLK의 negedge의 값을 바꿨기 때문에 MUX_input이 먼저 결정되는 것을 알 수 있다. 기존에 소개되어 있는 코드에는 Left headphone out과 Right headphone out을 구분하였지만 우리는 mono로 비는 데이터 없이 전송할 것이기 때문에 Left, Right를 구분하지 않는다.

finish_flag가 뜨면, counter는 1로 바뀐다. 그림 18을 보면 counter가 1이 되자마자 counting_state가 1로 transition된다. 50Mhz클럭을 기준으로 tarnsition하기 때문에 2.5khz의 속도를 사용하는 tick_counter는 28에서 다시 0으로 돌아오기 전에 transition된다. 이때, counting_state가 1이 되고 바로 read_enable이 1이 되고, ignition이 0이 되기 때문에 SCLK은 더 이상 동작하지 않는다. 따라서 tick_counter가 0이 되어 초기화되지 않는다. 만약, tick_counter가 0이 되어 초기화 되면 ACK를 LEDR[2..0]에 mapping할 것이기 때문에 불이 들어오면 안 된다.

```
148  //=========================================
149  // readenable & couting_state
150  always @(posedge clk50) begin
151      if(!rstn) begin
152          counting_state <= 0;
153          read_enable <= 0;
154          end
155      else begin
156          case(counting_state)
157          0:
158              begin
159              ignition <= 1;
160              read_enable <= 0;
161              if(counter == 1) counting_state <= 1;
162              end
163          1:
164              begin
165              read_enable <= 1;
166              ignition <= 0;
167              end
168          endcase
169      end
170  end
```

그림 18

```
171  //=========================================
172  // ROM_output_mux_counter & DACLRCLK
173  always @(posedge BCLK) begin
174      if(read_enable) DAC_LR_CLK_counter <= DAC_LR_CLK_counter + 1;
175  end
176  always @(posedge BCLK) begin
177      if(read_enable) begin
178          ROM_output_mux_counter <= ROM_output_mux_counter + 1;
179          if (DAC_LR_CLK_counter == 63) DAC_LR_CLK <= 1; // 8khz
180          else DAC_LR_CLK <= 0;
181      end
182  end
```

그림 19

그림 19를 보면 BCLK에 맞추어 read_enable이 1이 되어 정상적으로 WM8731에 데이터를 전송할 준비가 되면 DAC_LR_CLK_counter가 1씩 upcounting을 한다. 63에

도달할 때마다 DAC_LR_CLK을 1로 하고, 그렇지 않은 경우에는 0으로 하여 BCLK을 64배로 낮추어 8khz의 sampling frequency를 만들어 낸다. 이제 Audio_Codec은 정상동작한다. time_blk와 Digital_Clock에서 알람 설정을 위해 수정한 부분을 보도록 하자.

```
42      // mode selection
43      always @(posedge KEY_mode) begin
44              if (mode == 2'b00) mode <= 2'b01; // set hour selection mode
45              else if (mode == 2'b01) mode <= 2'b10; // set minute selection mode
46              else if (mode == 2'b10) mode <= 2'b11; // Alaram selection mode
47              else if (mode == 2'b11) mode <= 2'b00; // set clock mode
48              else mode <= 2'b00;
49          end
```

그림 20

Digital_Clock의 경우 알람 설정 모드가 추가 되었다.
time_blk는 그림 19를 보면 SWITCH와 AlarmRESET이 추가 되었는데 각각 SW[0]와 SW[1]에 맵핑하여 동작하며, DE2의 경우 설계자의 의사와 관계없이 reg의 defaut값들이 0이 되어 의도치 않게 알람이 울리기 때문에 보드에 코드를 업로딩 한 후에 AlarmRESET스위치를 한 번 올렸다 내려 Alarm으로 설정된 시간을 한 번 초기화 해줘야 한다. 출력으로 trg_alarm이 추가된 것을 알 수 있는데, DAC_DATA로 출력할 데이터의 경우 알람으로 설정한 시간과 현재 시간이 같아지면 trg_alarm을 1로 올려 Rom에서 WM8731로 DAC_DATA를 전송한다. 추가된 t_hour와 t_min은 각각 설정된 알람 정보를 기억하는 register이다.

time_blk.v

```
1   module time_blk (
2       input clk,
3       input rstn,
4       input [1:0] mode,
5       input SWITCH,
6       input alramRESET,
7       input increase,
8       input decrease,
9       output reg [5:0] hour,
10      output reg [5:0] min,
11      output reg [5:0] sec,
12      output reg trg_alarm
13  );
14      reg inc_hour;
15      reg dec_hour;
16      reg inc_min;
17      reg dec_min;
18      reg [5:0] v_sec;
19      reg [5:0] v_min;
20      reg [5:0] v_hour;
21      reg [5:0] s_sec;
22      reg [5:0] s_min;
23      reg [5:0] s_hour;
24      reg [5:0] t_min;
25      reg [5:0] t_hour;
```

그림 21

```
51          2'b11 :
52          begin
53              if(increase) begin
54                  if(SWITCH) begin inc_min <= 1; inc_hour <= 0; end
55                  else begin inc_hour <= 1; inc_min <= 0; end
56              end
57              else begin
58                  inc_min <= 0;
59                  inc_hour <= 0;
60              end
61              if(decrease) begin
62                  if(SWITCH) begin dec_min <= 1; dec_hour <= 0; end
63                  else begin dec_hour <= 1; dec_min <= 0; end
64              end
65              else begin
66                  dec_min <= 0;
67                  dec_hour <= 0;
68              end
69          end
70          default: begin
71              inc_hour <= 0;
72              dec_hour <= 0;
73              inc_min <= 0;
74              dec_min <= 0;
75          end
76      endcase
```

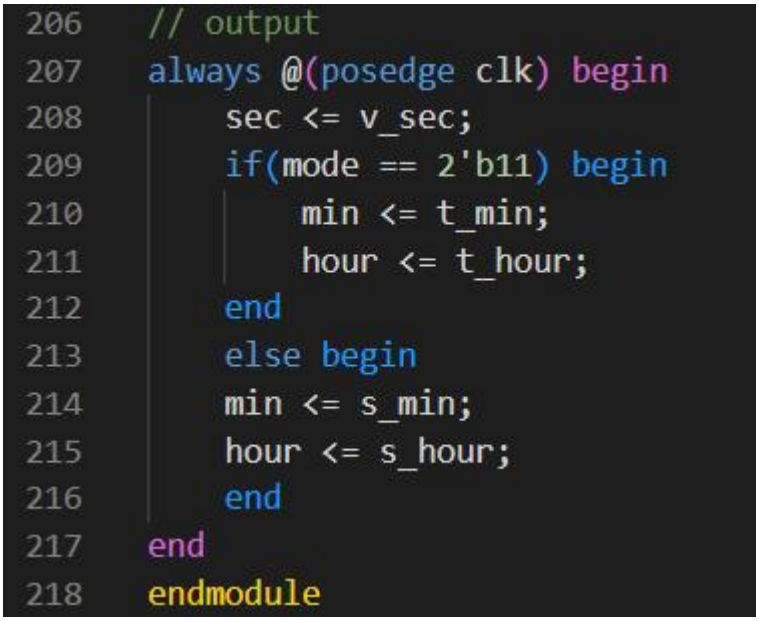
그림 22

알람설정 모드일 때, inc_min, inc_hour, dec_min, dec_hour의 경우 SWITCH를 통해 어떤 신호를 1로 올리고 내릴지 결정한다.

그림 23을 보면 Alarm설정은 50Mhz클럭을 기준으로 s_hour, s_min을 설정했던 것과 비슷하게 설정한다. 단, 리셋되었을 때, 초기값을 63으로 하여 의도치 않게 00시 00분에 알람이 울리는 것을 방지한다. trg_alarm은 설정한 시간과 현재 시간이 같아지면 1이 되는데 v_sec가 59가 되면 꺼지도록 한다. 따라서 59초 동안 Alarm이 울리게 된다.

```
167  // Alarm selection
168  always @(posedge clk) begin
169      if(alramRESET) begin
170          t_hour <= 63;
171          t_min <= 63;
172          trg_alarm <= 0;
173      end
174      else begin
175          if(mode == 2'b11) begin
176              if(inc_hour) begin
177                  if(t_hour >= 5'd23) t_hour <= 5'd0;
178                  else t_hour <= t_hour + 1;
179              end
180              else if(dec_hour) begin
181                  if(t_hour == 0) t_hour <= 5'd23;
182                  else t_hour <= t_hour - 1;
183              end
184              if(inc_min) begin
185                  if (t_min >= 6'd59) t_min <= 6'd0;
186                  else t_min <= t_min + 1;
187              end
188              else if (dec_min == 1) begin
189                  if (t_min == 0) t_min <= 6'd59;
190                  else t_min <= t_min - 1;
191          end
192          end
193          else begin
194              if(t_hour == v_hour && t_min == v_min) begin
195                  trg_alarm <= 1;
196                  if(v_sec == 59) begin
197                      t_hour <= 63;
198                      t_min <= 63;
199                  end
200              end
201              else trg_alarm <= 0;
202          end
203      end
204  end
```

그림 23



```
206   // output
207   always @(posedge clk) begin
208       sec <= v_sec;
209       if(mode == 2'b11) begin
210           min <= t_min;
211           hour <= t_hour;
212       end
213       else begin
214       min <= s_min;
215       hour <= s_hour;
216       end
217   end
218   endmodule
```

그림 24

마지막으로 출력의 경우 알람 설정 모드일 때는 설정한 시간을 볼 수 있도록 하고, 그 외의 경우에는 현재 시간을 보여준다. 보드의 동작은 영상에서 소개하며, 레포트에 서술한 자세한 내용을 사용설명서에 간략하게 기술하도록 한다.