Project Design

The design includes the following parts.

1. Square wave output

2. Key input

3. OLED display

FPGA drives the square wave generation, and the MCU drives the key and OLED, and the two communicate directly through the AHB bus.

Flow: OLED displays the frequency and performs key operation when a key operation is monitored, and sends the set frequency to the FPGA through AHB bus.

MCU Program Design

Here the address 0xA0000000 is used to store the frequency to be sent to the FPGA (the data is the accumulated value converted according to the 32-bit overflow and the 27MHz clock).

1. Judge the state of the three keys. When the key is pressed, it will first de-jitter and then wait for the key to be stable. The functions includes: move the cursor, decrease the value of the frequency where the cursor is located, and increase the value of the frequency where the cursor is located.

2. Write the converted frequency to the 0xA0000000 address (frequency*2^32/27000000).

3. Modify the display on the screen

```
uint32_t freq = 0; //frequency
uint8_t place = 0; //select

SystemInit();
GPIOInit();

```

```
OLED_Init();                          //initialize OLED
OLED_Clear();

while(1)
{
        if((GPIO_ReadBits(GPIO0)&GPIO_Pin_2) == 0)
        {
                delay_ms(20);
                while((GPIO_ReadBits(GPIO0)&GPIO_Pin_2) == 0){}

                freq += reduce_freq[place];

        }
        if((GPIO_ReadBits(GPIO0)&GPIO_Pin_3) == 0)
        {
                delay_ms(20);
                while((GPIO_ReadBits(GPIO0)&GPIO_Pin_3) == 0){}
                if(freq > reduce_freq[place])
                        freq -= reduce_freq[place];
                else
                        freq = 0;

        }
        if((GPIO_ReadBits(GPIO0)&GPIO_Pin_4) == 0)
        {
                delay_ms(20);
                while((GPIO_ReadBits(GPIO0)&GPIO_Pin_4) == 0);

                place++;
                if(place > 7)
                        place = 0;
        }

         *((uint32_t *)0xA0000000) = freq*159.0728628148148148;  // set the frequency to fpga

```

```
        {
                OLED_ShowChar(16+8*0, 3, '0'+freq/10000000%10, place==7?1:0);
                OLED_ShowChar(16+8*1, 3, '0'+freq/1000000%10, place==6?1:0);
                OLED_ShowChar(16+8*2, 3, ',',0);
                OLED_ShowChar(16+8*3, 3, '0'+freq/100000%10, place==5?1:0);
                OLED_ShowChar(16+8*4, 3, '0'+freq/10000%10, place==4?1:0);
                OLED_ShowChar(16+8*5, 3, '0'+freq/1000%10, place==3?1:0);
                OLED_ShowChar(16+8*6, 3, ',',0);
                OLED_ShowChar(16+8*7, 3, '0'+freq/100%10, place==2?1:0);
                OLED_ShowChar(16+8*8, 3, '0'+freq/10%10, place==1?1:0);
                OLED_ShowChar(16+8*9, 3, '0'+freq/1%10, place==0?1:0);
                OLED_ShowChar(16+8*10, 3 ,'H',0);
                OLED_ShowChar(16+8*11, 3 ,'z',0);
        }
}
```

FPGA Program Design
It includes top, hardcore, AHB parsing, and square wave generation.
Top: instantiate each module.
Hardcore: use gpio and AHB2 Master, and the ip cores are very easy to configure via software.
AHB parsing: At position 0 of AHB, a 32 bit reg_freq register used to receive the frequency of the square wave sent from MCU.

```
`timescale 100 ps/100 ps

module Gowin_AHB_Multiple
(
output [31:0] freq,
        output        wire        [31:0]        AHB_HRDATA,
        output        wire                      AHB_HREADY,
        output        wire        [ 1:0]        AHB_HRESP,
        input        wire        [ 1:0]  AHB_HTRANS,
          input        wire        [ 2:0]  AHB_Hburst,
        input        wire        [ 3:0]  AHB_HPROT,
```

```
        input        wire        [ 2:0]        AHB_HSIZE,
        input        wire                      AHB_HWRITE,
        input        wire                      AHB_HMASTLOCK,
        input        wire        [ 3:0]        AHB_HMASTER,
        input        wire        [31:0]        AHB_HADDR,
        input        wire        [31:0]  AHB_HWDATA,
        input        wire                      AHB_HSEL,
        input        wire                      AHB_HCLK,
        input        wire                      AHB_HRESETn
);

//The AHB BUS is always ready
assign AHB_HREADY = 1'b1; //ready signal, slave to MCU master
//Response OKAY
assign AHB_HRESP  = 2'b0;//response signal, slave to MCU master

//Define Reg for AHB BUS
reg [31:0]  ahb_address;
reg                 ahb_control;
reg         ahb_sel;
reg         ahb_htrans;

always @(posedge AHB_HCLK or negedge AHB_HRESETn)
begin
        if(~AHB_HRESETn)
        begin
                ahb_address  <= 32'b0;
                ahb_control  <= 1'b0;
        ahb_sel      <= 1'b0;
        ahb_htrans   <= 1'b0;
        end
        else              //Select The AHB Device
        begin                           //Get the Address of reg
                ahb_address  <= AHB_HADDR;
                ahb_control  <= AHB_HWRITE;
        ahb_sel      <= AHB_HSEL;
```

```
        ahb_htrans   <= AHB_HTRANS[1];
        end
end

wire write_enable = ahb_htrans & ahb_control    & ahb_sel;
wire read_enable  = ahb_htrans & (!ahb_control) & ahb_sel;

//The register of Multiple AHB bus
reg [32:0] reg_freq;

//write data to AHB bus
always @(posedge AHB_HCLK or negedge AHB_HRESETn)
begin
        if(~AHB_HRESETn)
        begin
                 reg_freq   <= 32'b0;
        end
        else if(write_enable)
        begin
                case (ahb_address[15:0])
                16'h0000: reg_freq         <= AHB_HWDATA[31:0];
                endcase
        end
end

//register address
reg [31:0] ahb_rdata;

always @(*)
begin
        if(read_enable)  //read cmd
        begin
                case (ahb_address[15:0])
                32'h0000:  ahb_rdata = reg_freq;
                default:ahb_rdata = 32'hFFFFFFFF;
                endcase
```

```
        end
    else
    begin
        ahb_rdata = 32'hFFFFFFFF;
    end
end

assign AHB_HRDATA = ahb_rdata;

assign freq = reg_freq;

Endmodule
```

Square wave generation: 32-bit accumulator, by modifying the size of the single accumulation to control the period of the waveform.

```
module wave (
    input sys_clk,
    input reset_n,
    input [31:0] freq,
    output wave_out
);

reg [31:0] phase_acc;
always @(posedge sys_clk) phase_acc <= phase_acc + freq;
wire [31:0] phase = phase_acc;
assign wave_out = phase[31];

endmodule
```

Resource Utilization

MCU:

```
1. ===============================================================================
2.
3.
4.       Code (inc. data)   RO Data    RW Data    ZI Data      debug
5.
6.        2880         142        2324          16        1632       38415    Grand
   Totals
7.       2880        142        2324          16        1632      38415   ELF Image
   Totals
8.       2880        142        2324          16           0          0   ROM Totals
9.
10. ================================================================================
   ==
11.
12.     Total RO  Size (Code + RO Data)                 5204 (   5.08kB)
13.     Total RW  Size (RW Data + ZI Data)              1648 (   1.61kB)
14.     Total ROM Size (Code + RO Data + RW Data)       5220 (   5.10kB)
15.
16. ================================================================================
   ==
17.
```