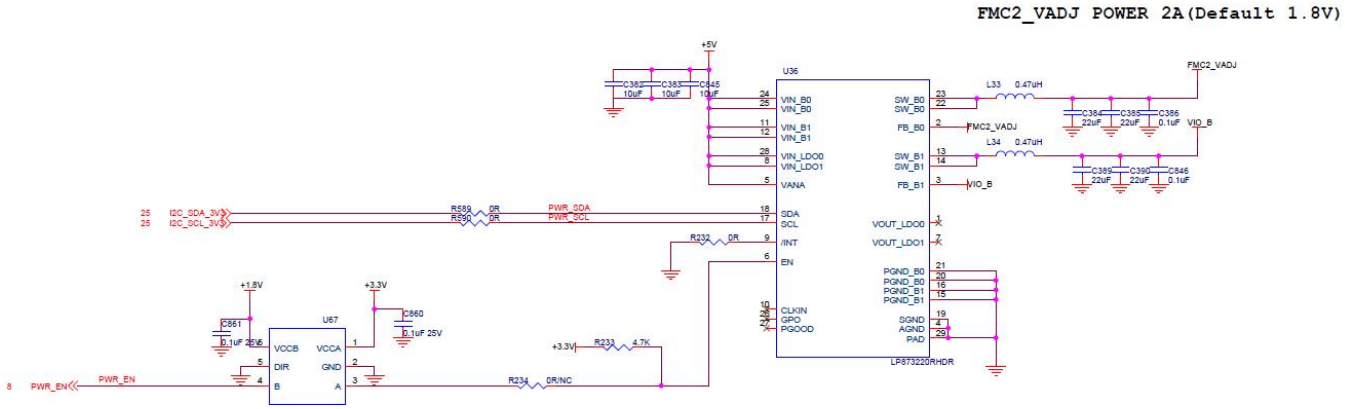
1 Experiment Introduction

There is a chip on AXKU040 FPGA Board that can adjust the io voltage, and the output voltage can be changed by adjusting the value of the internal register. The default configuration of bank 64 65 on the FPGA development board is 1.8v. LP873220 in the schematic



2 Experiment Principle

Here we have a simple understanding of the I2C protocol, and we will explain it in detail in the subsequent chapters:

2.1 I2C Physical Connection

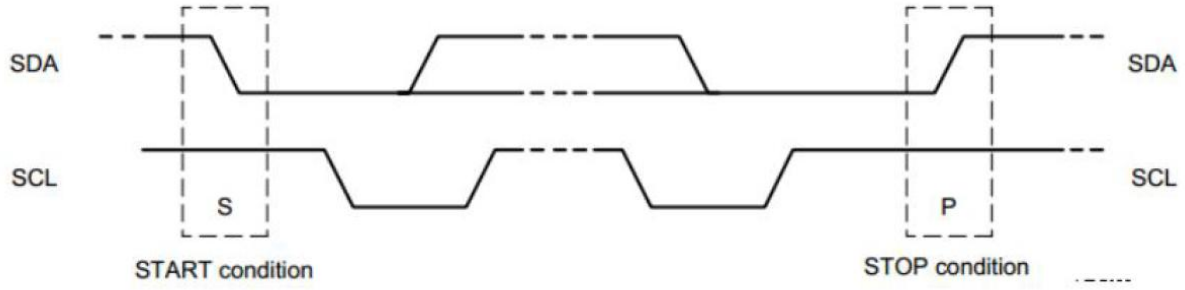
They are composed of serial data lines, serial clock lines, and pull-up resistors. The communication principle is to generate the signals required by the I2C bus protocol for data transmission by controlling the high and low timing of the SCL and SDA lines. When the bus is in idle state, these two lines are generally pulled high by the pull-up resistor connected above, and maintain a high level

2.2 Start Condition S and Stop Condition P

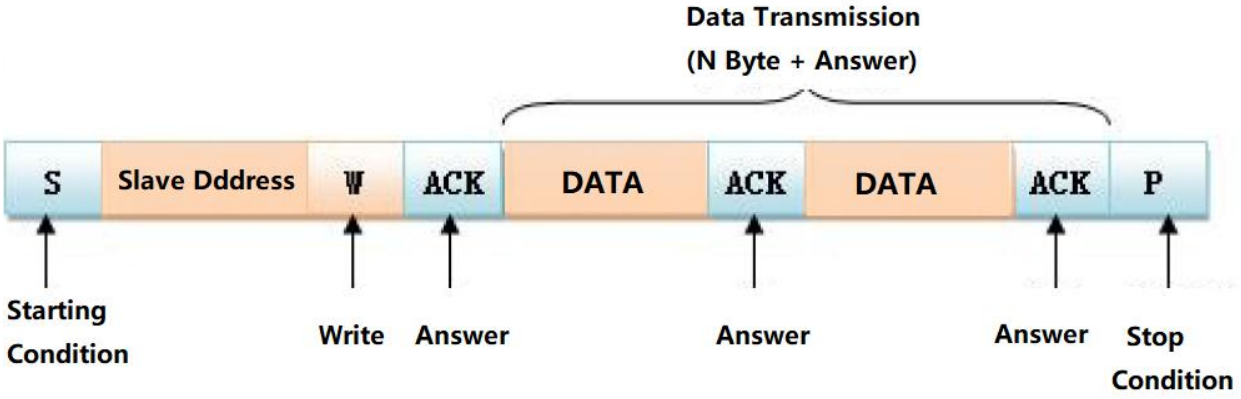
Start condition S: When SCL is high, SDA changes from high level to low level

Stop condition P: When SCL is high, SDA switches from low level to high level

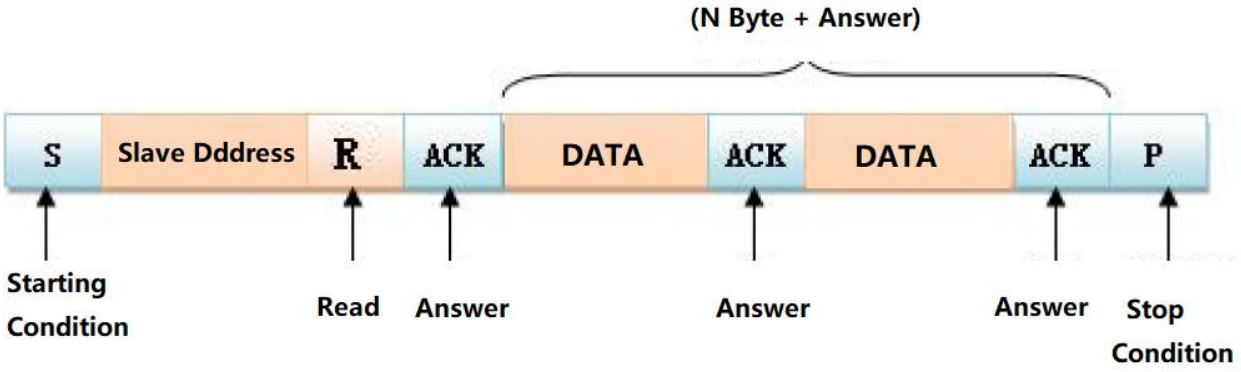
The start and stop conditions are generally generated by the host. The bus is in the busy state after the start condition, and after a certain period of time in the stop condition, the bus is in the idle state again. The figure 2-1 shows the timing diagram of the signal generation of the start and stop conditions.



The figure below is the process of i2c reading and writing data



The master device reads data from the slave device. The data transmission format is as follows:



Data transmission method:

1) As can be seen from the above figure, before reading and writing data, you must know the device address 0xc2i2c of the LP873220 mounted on the bus. The data on the bus is transmit in bytes, and the byte transmitting order is serial transmission from high to low. At this time, everyone may have doubts. There is only one SDA signal line for data transmission on our i2c bus. How does the host distinguish when reading and writing data from the slave device? There are 7 fixed device address

bits in the front of this program, and the last digit 0 means that the host transmits data to the slave, and 1 means that the host reads the data from the slave.

2) After transmitting the device address, the slave device needs to transmit a response signal (ack low level) to the host after sending the last bit, indicating that the response host address is correct, and then the data can be sent. Every time a byte of data is transmitted, an ack is returned to indicate that the byte data is received.

3) In the process of transmitting, each bit of data transmitted by i2c has a corresponding pulse (or synchronization control), that is, with the cooperation of the SCL serial clock, each bit of data is transmitted serially in SDA bit by bit. During the high level of SCL, the SDA level must remain stable. Changes can be sent during SCL low level, high level data is 1 and low level data is 0.

3 Programming

The program design is very simple. It is divided into three modules in total. Among them, reg_config.v generates the clock for configuring lp873220 and the data output corresponding to the voltage value sent to the slave. You can see the value of the corresponding voltage in the figure below from the chip data manual.

```
//register of lp873220
always@(reg_index)
begin
        case(reg_index)
            0:reg_data<=16'h06fc;       //BUCK0_VSET(addr=0x06), b2=1.8V, d4 =2.5V, fc=3.3V.
            1:reg_data<=16'h07fc;       //BUCK1_VSET(addr=0x07), b2=1.8V, d4 =2.5V, fc=3.3V
      default:reg_data<=16'hffff;
      endcase
end
```

i2c_com is the underlying control SDA bus to transmit data to lp873220 part of the code display:

```
case(cyc_count)
0:begin ack1<=1;ack2<=1;tr_end<=0;sclk<=1;reg_sdat<=1;end
1:reg_sdat<=0;                    //开始传输
2:sclk<=0;
3:reg_sdat<=i2c_data[23];
4:reg_sdat<=i2c_data[22];
5:reg_sdat<=i2c_data[21];
6:reg_sdat<=i2c_data[20];
7:reg_sdat<=i2c_data[19];
8:reg_sdat<=i2c_data[18];
9:reg_sdat<=i2c_data[17];
10:reg_sdat<=i2c_data[16];
11:reg_sdat<=1;                   //应答信号
12:begin reg_sdat<=i2c_data[15];ack1<=i2c_sdat;end
13:reg_sdat<=i2c_data[14];
14:reg_sdat<=i2c_data[13];
15:reg_sdat<=i2c_data[12];
16:reg_sdat<=i2c_data[11];
17:reg_sdat<=i2c_data[10];
18:reg_sdat<=i2c_data[9];
19:reg_sdat<=i2c_data[8];
20:reg_sdat<=1;                   //应答信号
21:begin reg_sdat<=i2c_data[7];ack2<=i2c_sdat;end
22:reg_sdat<=i2c_data[6];
23:reg_sdat<=i2c_data[5];
24:reg_sdat<=i2c_data[4];
25:reg_sdat<=i2c_data[3];
26:reg_sdat<=i2c_data[2];
27:reg_sdat<=i2c_data[1];
28:reg_sdat<=i2c_data[0];
29:reg_sdat<=1;                   //应答信号
30:begin ack3<=i2c_sdat;sclk<=0;reg_sdat<=0;end
31:sclk<=1;
32:begin reg_sdat<=1;tr_end<=1;end              //IIC传输结束
endcase
```

4 Experiment Result

After programming the program, find the u36 number on the FPGA development board, and use a universal meter to measure the changed voltage value of our No. 2 pin.