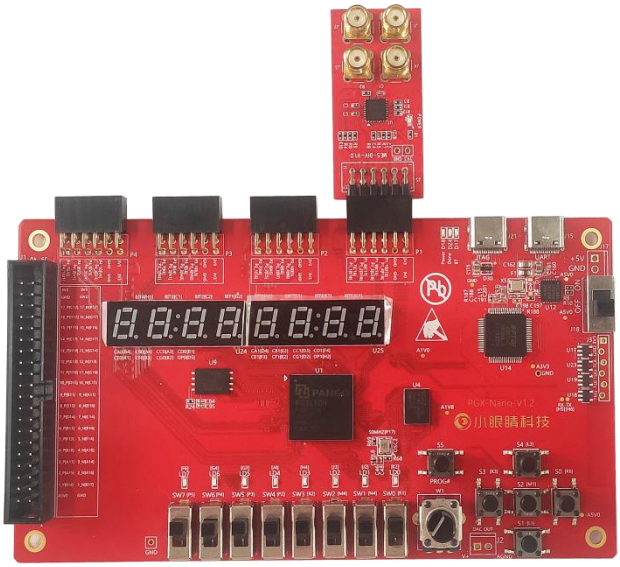
27. 分频器实验例程说明

27.1. 实验简介

实验使用“小眼睛科技”FPGA 开发板与分频器模块，时钟频率的二分频操作。

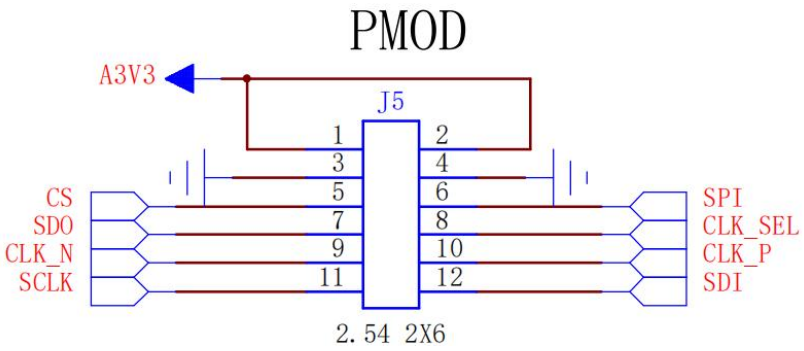


27.2. 实验目的

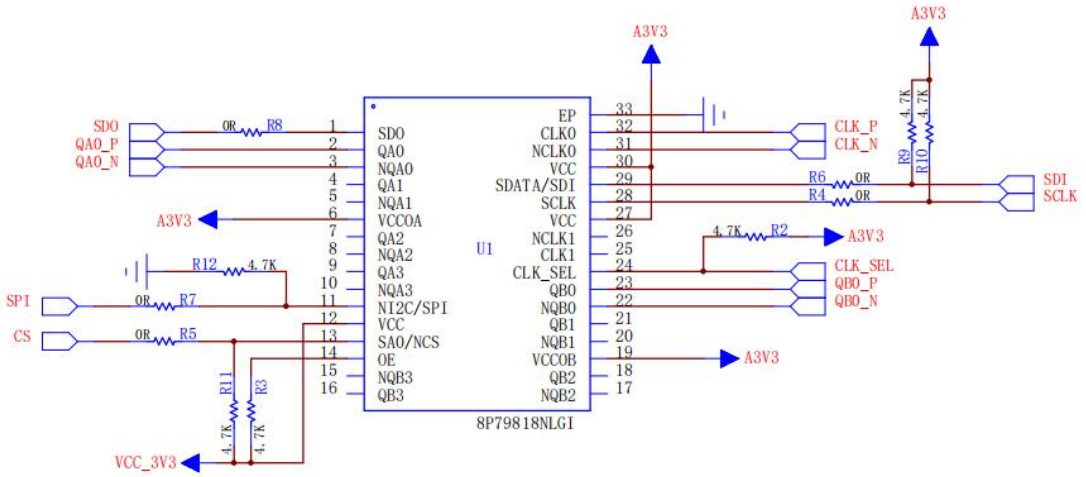
使用分频器完成二分频操作。

27.3. 实验使用模块介绍

实验使用“小眼睛科技”的分频器模块，模块搭载芯片型号为：8P79818，可使用 SPI 或 I2C 对分频器芯片进行寄存器配置，模块接口部分原理图如下图所示：



模块芯片连接原理图如下图所示，模块时钟输入连接到分频器芯片 CLK0，NCLK0 接口，模块分频输出连接到芯片输出信号 QA0、NQA0、QB0、NQBO，芯片内部共两个 bank，分别为 bankA、bankB，相同 bank 分频出的时钟频率是一致的，因此模块分别保留了 bankA 的一对输出 QA0、NQA0，bankB 的一对输出 QB0、NQBO，作为模块时钟输出。



通过配置分频器芯片寄存器，可以控制输出使能、分频系数、输出时钟信号类型灯参数。配置寄存器可使用 SPI 或 I2C 通信协议。

模块接口部分原理图中的 SPI 信号为高电平时，为 SPI 协议配置寄存器模式；为低电平时，为 I2C 协议配置寄存器模式。

使用 SPI 协议配置寄存器时，CS 为片选信号，SDI 为分频芯片 SPI 输入信号，SDO 为分频芯片 SPI 输出信号，SCLK 为时钟输入。

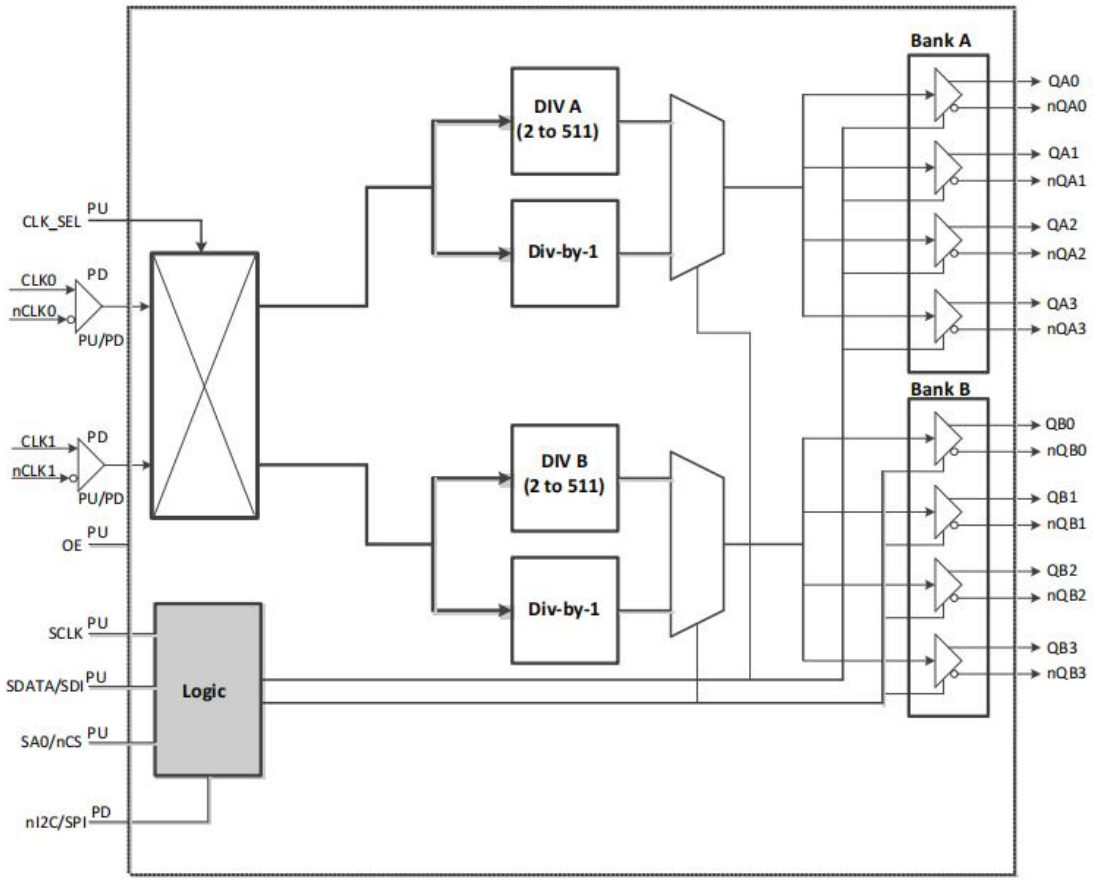
使用 I2C 协议配置寄存器时，器件地址为 110110x，其中 x 为由 CS 信号决

定，CS 为高电平时，器件地址为 1101101，CS 为低电平时，器件地址为 1101100；此时 SDO 信号不使用，SDI 信号为 I2C 数据线，SCLK 信号为 I2C 时钟线。

CLK_N 与 CLK_P 信号为模块的输入时钟信号，分频器模块将在此时钟基础上进行分频操作，分频范围为 2~511。

另外，在寄存器配置时，需要选择参考时钟控制方式，即通过寄存器控制选择 bankA 与 bankB 的参考时钟 CLK0、NCLK0 还是 CLK1、NCLK1，还是通过输入引脚 CLK_SEL 控制选择 bankA 与 bankB 的参考时钟 CLK0、NCLK0 还是 CLK1、NCLK1。由于在模块的硬件设计上，将模块的输入时钟连接在分频器芯片 CLK0、NCLK0 接口上，因此使用寄存器作为参考时钟控制方式，不再使用 CLK_SEL。

模块搭载分频器芯片 8P79818，芯片框架如下图所示：



分频器芯片共有两个 bank，每个 bank 对应一个分频，每个 bank 有 4 对时钟输出，且输出频率相同。因此模块仅连接分频器芯片 bankA、bankB 的各一对输出。

OE 信号为高电平时，输出由寄存器配置，OE 为低电平时，输出为高阻抗状态，因此模块对 OE 信号再硬件上进行上拉。

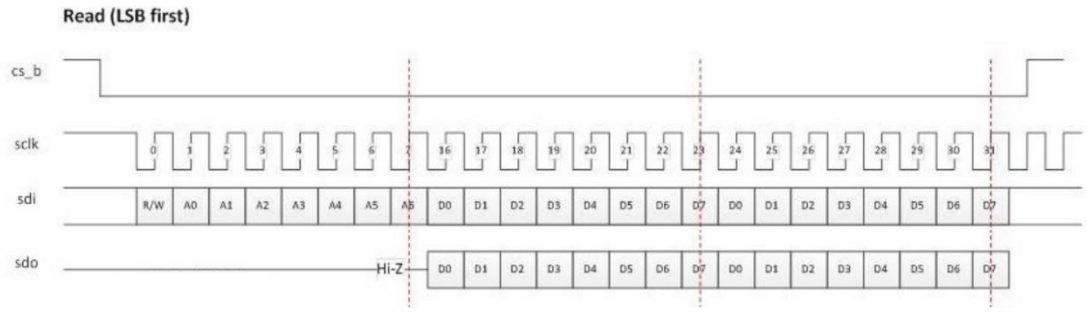
27.4. 模块寄存器配置过程简介

模块搭载的分频器芯片对应的寄存器地址范围为 0~F，每个寄存器地址对应 8bit 寄存器数据。寄存器配置详情请参考 *8P79818 数据手册*。

27.4.1. 使用 SPI 配置寄存器：

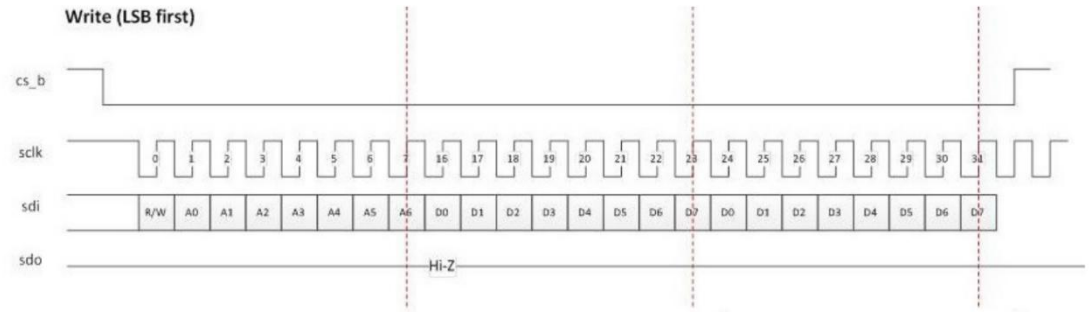
使用 SPI 配置寄存器时，需首先将模块 SPI 信号置 1，发送数据时，发送的第 1bit 数据为读写控制位，置 1 时，芯片执行读操作，置 0 时芯片执行写操作。在读写控制位后输入一个地址，数据会依此地址位起点，自动填入后面的寄存器地址或从寄存器中取出数据。

读操作顺序如下图所示，注意低位在前：

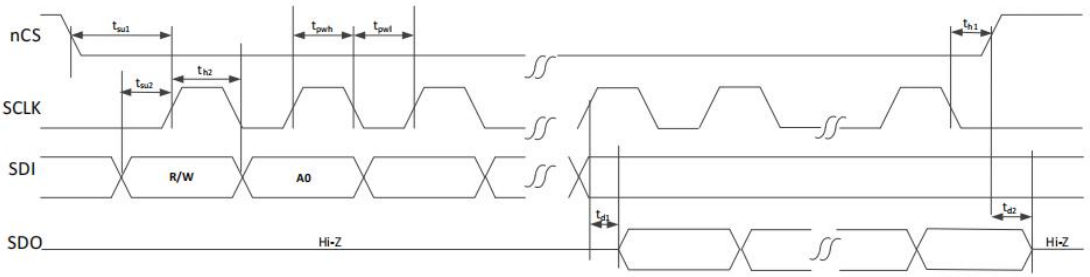


写操作顺序如下图所示，注意低位在前：

During SPI Write operations, the user may continue to hold nCS low and provide further bytes of data for up to a total of 16 bytes in a single block write. Data is written directly into the appropriate register as it is received.



读写操作时序如下所示：



Symbol	Parameter	Min	Typ	Max	Unit
t_{pw}	SCLK Period	20			ns
t_{pw1}	SCLK Pulse Width Low	8			ns
t_{pw2}	SCLK Pulse Width High	8			ns
t_{su1}	Valid nCS to SCLK Rising Setup Time	10			ns
t_{h1}	Valid nCS After Valid SCLK Hold Time (CLKE = 0/1)	10			ns
t_{su2}	Valid SDI to SCLK Rising Setup Time	5			ns
t_{h2}	Valid SDI after valid SCLK Hold Time	5			ns
t_{d1}	SCLK falling (rising in CLKE = 1 case) to Valid Data Delay Time			5	ns
t_{d2}	nCS rising edge to SDO High Impedance Delay Time			10	ns
t_{csh}	Time between Consecutive Read-Read or Read-Write Accesses (nCS rising edge to nCS falling edge)	20			ns

使用 SPI 配置此芯片时，支持的最大时钟频率为 50MHz。

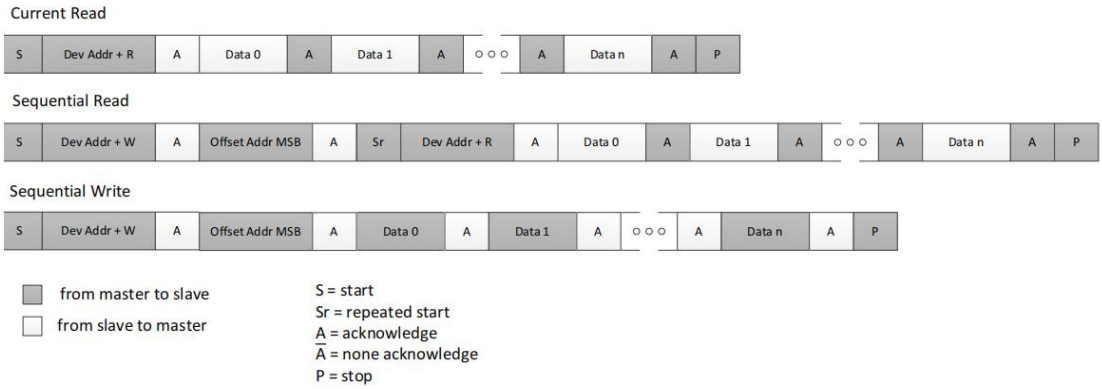
27.4.2. 使用 I2C 配置寄存器：

使用 I2C 配置寄存器时，需首先将模块 SPI 信号置 0，芯片作为 I2C 从机，其器件地址为 110110x，其中 x 由 CS 信号决定，CS 信号为高电平时，器件地址为 1101101，CS 信号为低电平时，器件地址为 1101100；使用的 I2C 时钟频率为 100KHz 或 400KHz。

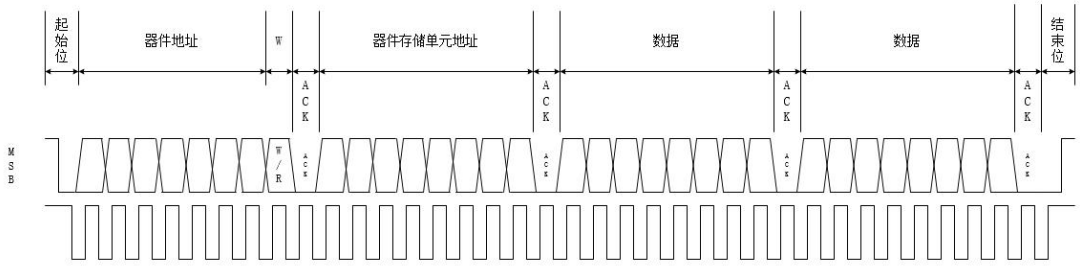
传输寄存器数据时，寄存器地址为写入或者读取的第一个寄存器地址，第一

字节数据对应写入或者读取第一个寄存器地址，第二字节数据对应写入或者读取第一个寄存器地址的下一个寄存器地址，第三字节数据对应写入或者读取第一个寄存器地址的下下一个寄存器地址，依次类推，即仅写入一个地址，数据会依此地址位起点，自动填入后面的寄存器地址或从寄存器中取出数据。

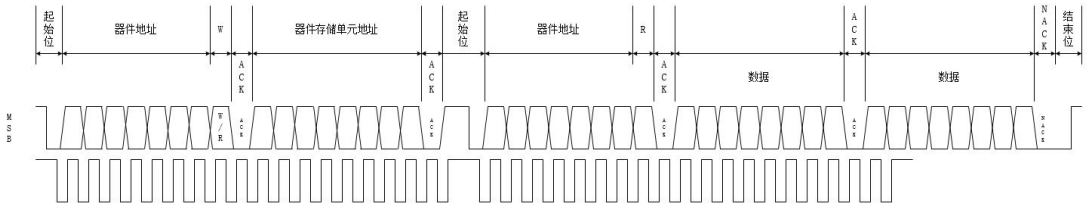
相应的读写时序如下图所示：



I2C 写时序详细框图如下所示（每次传输 8bit 数据，其中 8bit 中高位先发）：



I2C 读时序详细框图如下所示（每次传输 8bit 数据，其中 8bit 中高位先发）：



27.4.3. 模块寄存器简介

模块搭载的分频器芯片对应的寄存器地址范围为 0~F，每个寄存器地址对应

8bit 寄存器数据。寄存器配置详情请参考 *8P79818 数据手册*。

寄存器地址 0-1：

器件控制寄存器，配置为 0：8'h10，1：8'h00 即可。

寄存器地址 2-5：

bankA 控制，控制 bankA 信号输出模式等信息。

Address (Hex)	D7	D6	D5	D4	D3	D2	D1	D0
2	Rsvd			TERM_A	QA_POL3	QA_POL2	QA_POL1	QA_POL0
3	Rsvd		MODE_QA3[2:0]			MODE_QA2[2:0]		
4	Rsvd		MODE_QA1[2:0]			MODE_QA0[2:0]		
5	DIS_QA3	DIS_QA2	DIS_QA1	DIS_QA0	Rsvd			

寄存器地址 2~5

模块 bankA 输出保留的接口为 QA0、NQA0，因此控制寄存器地址 4 的数据第 2~0 位，设置 bankA 输出信号 QA0、NQA0 的输出模式，控制寄存器地址 5 的数据第 4 位为 0 使能 QA0、NQA0 输出即可，具体模式如下所示：

Output driver mode of operation for output pair QAm, nQAm:
000 = high-impedance
001 = LVPECL
010 = LVDS (default)
011 = LVCMOS
100 = HCSL
101 = CML 400mV swing
110 = CML 800mV swing
111 = Reserved

其他寄存器数据保持默认即可。

Bit Field Name	Field Type	Default Value	Description
TERM_A	R/W	0b	Indicates termination used on Bank A outputs when HCSL mode is selected: 0 = 33Ω/ 50Ω 1 = 50Ω
QA_POLm	R/W	0h	Output polarity selection for output pair nQAm, QAm in LVCMOS mode: 0 = nQAm pin is inverted relative to QAm pin when in LVCMOS mode 1 = nQAm and QAm pins are in-phase when in LVCMOS mode
MODE_QAm[2:0]	R/W	010b	Output driver mode of operation for output pair QAm, nQAm: 000 = high-impedance 001 = LVPECL 010 = LVDS (default) 011 = LVCMOS 100 = HCSL 101 = CML 400mV swing 110 = CML 800mV swing 111 = Reserved
DIS_QAm	R/W	0b	Disable output pair QAm, nQAm: 0 = Output pair QAm, nQAm is enabled (disable output bank using SYNC_DISA to prevent runt pulses when enabling) 1 = Output pair QAm, nQAm is powered-down
Rsvd	R/W	–	Reserved. Always write 0 to this bit location. Read values are not defined.

寄存器 2~5 数据说明

寄存器地址 6-9：

bankB 控制，控制 bankB 信号输出模式等信息。

Address (Hex)	D7	D6	D5	D4	D3	D2	D1	D0
6	Rsvd			TERM_B	Rsvd	Rsvd	Rsvd	Rsvd
7	Rsvd		MODE_QB3[2:0]			MODE_QB2[2:0]		
8	Rsvd		MODE_QB1[2:0]			MODE_QB0[2:0]		
9	DIS_QB3	DIS_QB2	DIS_QB1	DIS_QB0	Rsvd			

寄存器 6~9

模块 bankB 输出保留的接口为 QB0、NQB0，因此控制寄存器地址 8 的数据第 2~0 位，设置 bankB 输出信号 QB0、NQB0 的输出模式，控制寄存器地址 9 的数据第 4 位为 0 使能 QB0、NQB0 输出即可，具体模式如下所示

Bit Field Name	Field Type	Default Value	Description
TERM_B	R/W	0b	Indicates termination used on Bank B outputs when HCSL mode is selected: 0 = 33Ω/ 50Ω 1 = 50Ω
MODE_QBm[2:0]	R/W	010b	Output driver mode of operation for output pair QBm, nQBm: 000 = high-impedance 001 = LVPECL 010 = LVDS (default) 011 = Rsvd 100 = HCSL 101 = CML 400mV swing 110 = CML 800mV swing 111 = Reserved
DIS_QBm	R/W	0b	Disable output pair QBm, nQBm: 0 = Output pair QBm, nQBm is enabled (disable output bank using SYNC_DISB to prevent runt pulses when enabling) 1 = Output pair QBm, nQBm is powered-down
Rsvd	R/W	–	Reserved. Always write 0 to this bit location. Read values are not defined.

寄存器 6~9 数据说明

寄存器地址 A-B：

该寄存器地址 A~B 保留。

寄存器地址 C-F：

控制分频器输出信号的分频比。

Address (Hex)	D7	D6	D5	D4	D3	D2	D1	D0
C	DIVA[7:0]							
D	DIVB[7:0]							
E	Rsvd						DIVB[8]	DIVA[8]
F	Rsvd							

Bit Field Name	Field Type	Default Value	Description
DIVA[8:0]	R/W	000h	Divider ratio for Bank A outputs: 00h – 01h = Bypass divider and pass reference clock directly to the Bank A outputs 02h – 1FFh = ratio to be used by the A divider is value written here. For example writing a 4 in this field will results in a divide ratio of 4 being used.
DIVB[8:0]	R/W	000h	Divider ratio for Bank B outputs: 00h – 01h = Bypass divider and pass reference clock directly to the Bank B outputs 02h – 1FFh = ratio to be used by the B divider is value written here. For example writing a 4 in this field will results in a divide ratio of 4 being used.
Rsvd	R/W	–	Reserved. Always write 0 to this bit location. Read values are not defined.

寄存器 C~F 及数据说明

寄存器地址 C 的数据第 7~0 位、寄存器地址 E 的数据第 0 位控制 bankA 的输出

分频比。

寄存器地址 D 的数据第 7~0 位、寄存器地址 E 的数据第 1 位控制 bankB 的输出分频比。

27.5. 实验代码设计

实验通过 I2C 协议配置寄存器来控制模块分频，其中寄存器地址 C~F 控制 bankA、bankB 输出时钟的分频比率，寄存器地址 2~5、6~9 控制 bankA、bankB 的输出信号类型以及使能。具体寄存器内容请参考 *8P79818 数据手册*。

代码两对输出配置为 LVDS 信号，分屏比为 2，bankA 与 bankB 对应分频参考时钟均为 CLK0、NCLK0 输入时钟。

顶层模块如下所示：

```
module div_test(
   input       clk         ,
   input       rst_n       ,
   input       sdo         ,
   output      cs          /* synthesis PAP_MARK_DEBUG="true" */,
   output      scl         /* synthesis PAP_MARK_DEBUG="true" */,
   inout       sdi         /* synthesis PAP_MARK_DEBUG="true" */,
   output       mode        /* synthesis PAP_MARK_DEBUG="true" */,
   output      clk_sel     /* synthesis PAP_MARK_DEBUG="true" */,
   output      clk_n       /* synthesis PAP_MARK_DEBUG="true" */,
   output      clk_p
   );

parameter sys_clk_fre = 26'd50_000_000 ;
parameter time_10ms = sys_clk_fre/100 ;
```

```
reg [15:0]data_init /* synthesis PAP_MARK_DEBUG="true" */;
reg [4:0]cnt_init /* synthesis PAP_MARK_DEBUG="true" */;
reg start /* synthesis PAP_MARK_DEBUG="true" */;
reg [25:0]cnt_delay /* synthesis PAP_MARK_DEBUG="true" */;
```

```
wire rst_delay /* synthesis PAP_MARK_DEBUG="true" */;
reg  rst_delay_r1 ;
reg  rst_delay_r2 ;
wire byte_over /* synthesis PAP_MARK_DEBUG="true" */;
wire [7:0]data_out  /* synthesis PAP_MARK_DEBUG="true" */;
reg reg_byte_over /* synthesis PAP_MARK_DEBUG="true" */;
//----------------------OUTPUT SIGNAL----------------------------//

assign mode = 1'b0 ;
assign clk_sel = 1'b0 ;
assign cs = 1'b0 ;
assign clk_n = ~clk ;
assign clk_p = clk ;

//----------------------TIME DELAY-----------------------------//

always @(posedge clk ) begin
   if (~rst_n)
      cnt_delay <= 26'd0 ;
   else if (cnt_delay >= time_10ms)
      cnt_delay <= cnt_delay ;
   else
      cnt_delay <= cnt_delay + 26'd1 ;
end

assign rst_delay = (cnt_delay >= time_10ms - 26'd10) ? 1'b1 : 1'b0 ;

always @(posedge clk ) begin
   if (~rst_n) begin
      rst_delay_r1 <= 1'b0 ;
      rst_delay_r2 <= 1'b0 ;
      reg_byte_over <= 1'b0 ;
   end
   else begin
      rst_delay_r1 <= rst_delay ;
      rst_delay_r2 <= rst_delay_r1 ;
      reg_byte_over <= byte_over ;
   end
end

//------------------------CONFIG IIC-----------------------//
iic_dri #(
    .CLK_FRE      (27'd50_000_000    ),
    .IIC_FREQ     (20'd400_000       ),
```

```
    .T_WR         (10'd5            ),
    .DEVICE_ID    (8'b1101_1000           ),
    .ADDR_BYTE    (2'd1             ),
    .LEN_WIDTH    (8'd4             ),
    .DATA_BYTE    (2'd1             )
)iic_dri(
    .clk          ( clk             ),
    .rstn         ( rst_delay       ),
    .pluse        ( start           ),
    .w_r          ( 1               ),
    .byte_len     ( 5'd16           ),
    .addr         ( data_init[15:8]  ),
    .data_in      ( data_init[7:0]   ),
    .busy         (                 ),
    .byte_over    ( byte_over       ),
    .data_out     ( data_out        ),
    .scl          ( scl             ),
    .sda_in       ( sda_in          ),
    .sda_out      ( sda_out         ),
    .sda_out_en   ( sda_out_en      )
);
     assign sdi = sda_out_en ? sda_out : 1'bz;
     assign sda_in = sdi;
//-----------------REGISTER CONFIG----------------//
always @(posedge clk ) begin
   case (cnt_init)
      0  :  data_init <= 16'h0010 ;
      1  :  data_init <= 16'h0100 ;
      2  :  data_init <= 16'h0200 ;
      3  :  data_init <= 16'h0312 ;
      4  :  data_init <= 16'h0412 ;
      5  :  data_init <= 16'h05E0 ;
      6  :  data_init <= 16'h0600 ;
      7  :  data_init <= 16'h0712 ;
      8  :  data_init <= 16'h0812 ;
      9  :  data_init <= 16'h09E0 ;
      10 :  data_init <= 16'h0A00 ;
      11 :  data_init <= 16'h0B00 ;
      12 :  data_init <= 16'h0C02 ;
      13 :  data_init <= 16'h0D02 ;
      14 :  data_init <= 16'h0E00 ;
      15 :  data_init <= 16'h0F00 ;
      default: data_init <= 16'h0000 ;
   endcase
```

```
end

always @(posedge clk ) begin
   if (~rst_delay)
      cnt_init <= 5'd0 ;
   else if (cnt_init == 5'd16)
      cnt_init <= cnt_init ;
   else if (byte_over)
      cnt_init <= cnt_init + 5'd1 ;
end

always @(posedge clk ) begin
   if (~rst_delay)
      start <= 1'b0 ;
   else if ((rst_delay_r1)&&(~rst_delay_r2))
      start <= 1'b1 ;
//   else if ((reg_byte_over)&&(cnt_init <= 5'd15))
//      start <= 1'b1 ;
   else
      start <= 1'b0 ;
end

endmodule
```

I2C 通信模块如下所示：

```
`timescale 1ns / 1ps
`define UD #1
module iic_dri #(
    parameter           CLK_FRE = 27'd50_000_000,  //system clock
frequency
    parameter           IIC_FREQ = 20'd400_000,    //I2c clock frequency
    parameter           T_WR = 10'd5,              //I2c transmit delay ms
    parameter           DEVICE_ID = 8'hA0,         //I2C slave port device
ID
    parameter           ADDR_BYTE = 2'd1,          //I2C addr byte number
    parameter           LEN_WIDTH = 8'd3,          //I2C transmit byte
width
    parameter           DATA_BYTE = 2'd1           //I2C data byte number
)(
    input               clk,
    input               rstn,
    input               pluse,                    //I2C transmit trigger
    input               w_r,                      //I2C transmit
direction 1:send  0:receive
```

```
    input  [LEN_WIDTH:0] byte_len,                  //I2C transmit data
byte length of once trigger

    input  [7:0]         addr,                      //I2C transmit addr
    input  [7:0]         data_in,                   //I2C send data

    output reg           busy=0,                    //I2C bus status
    output reg           byte_over=0,               //I2C byte transmit
over flag

    output [7:0]      data_out,                  //I2C receive data

    output               scl,
    input                sda_in,
    output   reg         sda_out=1'b1,
    output               sda_out_en
);

    localparam CLK_DIV =
CLK_FRE/IIC_FREQ;  //������������ ������ʱ����
    localparam ID_ADDR_BYTE = ADDR_BYTE + 1'b1;//�� ��deviceID� ���
    localparam DATA_SET = CLK_DIV>>2;//���data�沿λ��
    localparam T_WR_DELAY = T_WR*CLK_FRE/1000;

    //  iic clock time counter
    reg [20:0] fre_cnt;//=21'd0;
    always @(posedge clk)
    begin
        if(!rstn)
            fre_cnt <= `UD 21'd0;
        else if(fre_cnt == CLK_DIV - 1'b1)
            fre_cnt <= `UD 21'd0;
        else
            fre_cnt <= `UD fre_cnt + 1'b1;
    end

    wire  full_cycle;
    wire  half_cycle;
    assign full_cycle = (fre_cnt == CLK_DIV - 1'b1) ? 1'b1 :
1'b0;  //SCL�����ڵ1��SCL����λ��
    assign half_cycle = (fre_cnt == (CLK_DIV>>1'b1) - 1'b1) ? 1'b1 :
1'b0;//SCL��½��ڵ1/2��SCL����λ��

    wire start_h;
```

```
    wire dsu;
    assign start_h = (fre_cnt == DATA_SET - 1'b1) ? 1'b1 : 1'b0;
//��'º�� SDA�� λ��  ��1/4��SCL����λ��
    assign dsu = (fre_cnt == (CLK_DIV>>1'b1) + DATA_SET - 1'b1) ? 1'b1 : 1'b0;
//��  ������SDA�抄λ�ã3/4��SCL����λ��

    //================================================================
==========
    //pluse trige the iic bus transmit start
    wire   start;
    reg    start_en;
    reg    pluse_1d,pluse_2d,pluse_3d;
    always @(posedge clk)
    begin
        if(!rstn)
        begin
            pluse_1d <= `UD 1'b0;
            pluse_2d <= `UD 1'b0;
            pluse_3d <= `UD 1'b0;
        end
        else
        begin
            pluse_1d <= `UD pluse   ;
            pluse_2d <= `UD pluse_1d;//ٍ��
            pluse_3d <= `UD pluse_2d;//d̨�����
        end
    end

    always @ (posedge clk)
    begin
        if(start || (!rstn))//��'��''��ź����
            start_en <= `UD 1'b0;
        else if(~pluse_3d & pluse_2d)//����
            start_en <= `UD 1'b1;
        else
            start_en <= `UD start_en;
    end

    assign start = (start_en & full_cycle) ? 1'b1 : 1'b0;

    reg w_r_1d=1'b0,w_r_2d=1'b0;
    always @(posedge clk)
    begin
        if(!rstn)
```

```
        begin
            w_r_1d <= `UD 1'b0;
            w_r_2d <= `UD 1'b0;
        end
        else
        begin
            w_r_1d <= `UD w_r;
            w_r_2d <= `UD w_r_1d;
        end
    end

    //=================================================================
========
    //     IIC FSM STATE
    //=================================================================
========
    localparam IDLE   = 3'd0;
    localparam S_START= 3'd1;
    localparam SEND   = 3'd2;
    localparam S_ACK  = 3'd3;
    localparam RECEIV = 3'd4;
    localparam R_ACK  = 3'd5;
    localparam STOP   = 3'd6;
    reg [2:0] state;
    reg [2:0] state_n;
    reg [2:0] trans_bit = 3'd0;

    reg [LEN_WIDTH :0] trans_byte = 5'd0;
    reg [LEN_WIDTH :0] trans_byte_max = 5'd0;
    reg       restart = 1'b0;
    reg [7:0] send_data=8'd0;
    reg [7:0] receiv_data=8'd0;
    reg       trans_en=0;
    reg       trans_over=0;
    reg       scl_out= 1'b1;

//    assign sda = sda_out_en ? sda_out : 1'bz;
//    assign sda_in = sda;
    assign scl = scl_out;

    //=================================================================
==========
    //transmit status
    always @ (posedge clk)
```

```
    begin
        if(start)//□□□□□□□□□'□□
            trans_en <= `UD 1'b1;
        else if(state == STOP && start_h)//□□□□STOP□□ □□□□□□
            trans_en <= `UD 1'b0;
        else
            trans_en <= `UD trans_en;
    end

//    always @(posedge clk)//□□□□
//    begin
//        if(start)
//            trans_over <= `UD 1'b0;
//        else if(state == STOP && start_h)
//            trans_over <= `UD 1'b1;
//        else
//            trans_over <= `UD trans_over;
//    end

    //================================================================
=========
    // IIC Bus status
    reg           twr_en=0;
    reg  [26:0]   twr_cnt=0;
    always @(posedge clk)
    begin
        if(state == STOP && dsu)//STOP □□'□□ʰ
            twr_en <= `UD 1'b1;
        else if(twr_cnt == T_WR_DELAY)//□□ʰ□□5ms
            twr_en <= `UD 1'b0;
        else
            twr_en <= `UD twr_en;
    end

    always @(posedge clk)
    begin
        if(twr_en)
        begin
            if(twr_cnt == T_WR_DELAY)//□□ʰ□□5ms
                twr_cnt <= `UD 1'b0;
            else
                twr_cnt <= `UD twr_cnt + 1'b1;
        end
        else
```

```
            twr_cnt <= `UD twr_cnt;
    end

    always @(posedge clk)
    begin
        if(start_en)  //���ա' ����busy
            busy <= `UD 1'b1;
        else if(twr_cnt == T_WR_DELAY)//busy��ʱ���
            busy <= `UD 1'b0;
        else
            busy <= `UD busy;
    end

    //================================================================
==========
    //iic bus controller
    always @(posedge clk)
    begin
        if(trans_en)
        begin
            if(half_cycle || full_cycle)
                scl_out <= ~scl_out;
            else
                scl_out <= scl_out;
        end
        else
            scl_out <= `UD 1'b1;
    end

    assign sda_out_en = ((state == S_ACK) || (state == RECEIV)) ? 1'b0 : 1'b1;

    //tx data control
    always @(posedge clk)
    begin
        if(start)//��'����ʱ��ʽ ����ʱ���ֽ ID+∂��
            send_data <= `UD {DEVICE_ID[7:1],1'b0};//{ֽ
ID����∂�� }   ������λ��ʽ
        else if(state == S_ACK && full_cycle)
//��SACK ʧ��ʱ������ʽ ������
        begin
            case(trans_byte)//������ 抄
                5'd0 : send_data <= `UD {DEVICE_ID[7:1],1'b0};
                5'd1 : send_data <= `UD addr[7:0];
```

```
                5'd2 : send_data <= `UD (w_r_2d) ? data_in :
{DEVICE_ID[7:1],1'b1};
                default: send_data <= `UD data_in;
            endcase
        end
        else
            send_data <= `UD send_data;
    end

    //transmit byte number,contain device ID��ADDR��DATA
    always @(posedge clk)
    begin
        if(start)
        begin
            if(w_r_2d)
                trans_byte_max <= `UD ADDR_BYTE + byte_len + 2'd1;
            else
                trans_byte_max <= `UD ADDR_BYTE + byte_len + 2'd2;
        end
        else
            trans_byte_max <= `UD trans_byte_max;
    end

    //sda out control
    always @(posedge clk)
    begin
        case(state)
            IDLE  ://����
            begin
                sda_out <= `UD 1'b1;
            end
            S_START ://��'
            begin
                if(start_h)//��'�� ����
                    sda_out <= `UD 1'b0;
                else if(dsu)//������SCL������ǰ����
                    sda_out <= `UD send_data[7-trans_bit];//��λ��
                else
                    sda_out <= `UD sda_out;
            end
            SEND  :
            begin
                sda_out <= `UD send_data[7-trans_bit];//�仯
��  ���λ��
```

```
            end
            S_ACK :
            begin
                if(trans_byte == ID_ADDR_BYTE && dsu
&& !w_r_2d)//�� ׷µ׷ ׷׷׷׷׷ƴ׷ ׷׷׷׷׷ȷ׷׷S_START
                    sda_out <= `UD 1'b1;
                else
                    sda_out <= `UD 1'h0;
            end
            R_ACK :
            begin
                if(trans_byte < trans_byte_max)//׷׷׷׷׷ɖ׷ь׷ACK
                    sda_out <= `UD 1'b0;
                else
                begin
                    if(dsu)//׷׷׷׷ACK׷׷׷׷׷׷׷׷׷׷STOP ׷׷׷׷ǰ׷
׷SDA׷׷׷׷
                        sda_out <= `UD 1'b0;
                    else//׷׷׷׷׷׷ۍ׷׷׷׷׷׷׷׷ʾ׷л׷׷׷׷ӷ׷׷
                        sda_out <= `UD 1'b1;
                end
            end
            STOP  :
            begin
                if(start_h)//ª׷׷ ׷׷
                    sda_out <= `UD 1'b1;
                else
                    sda_out <= `UD sda_out;
            end
            default: sda_out <= `UD 1'b1;
        endcase
    end

    // iic read data
    always @(posedge clk)
    begin
        if(state == RECEIV)
        begin
            if(full_cycle)//׷׷׷׷׷λ׷ý׷׷׷׷׷׷׷׷׷׷׷
                receiv_data <= `UD {receiv_data[6:0],sda_in};
            else
                receiv_data <= `UD receiv_data;
        end
        else
```

```
            receiv_data <= `UD 8'd0;
    end
    reg [7:0]data_out_reg=8'd0;
    always @(posedge clk)
    begin
        if(!rstn)
            data_out_reg <= `UD 8'd0;
        else if(state == RECEIV && trans_bit == 3'd7 &&
half_cycle)//����1byte���  ����  ���
            data_out_reg <= `UD receiv_data;
        else
            data_out_reg <= `UD data_out_reg;
    end
    assign data_out=data_out_reg;
    //one byte data transmit over flag
    always @(posedge clk)
    begin
        if(w_r_2d)
        begin
            if(trans_byte > ID_ADDR_BYTE - 1'b1 && dsu && trans_bit ==
3'd7)//∂��  ��Ա��ˣ����  ��б�ʶ
                byte_over <= `UD 1'b1;
            else
                byte_over <= `UD 1'b0;
        end
        else
        begin
            if(trans_byte > ID_ADDR_BYTE && dsu && trans_bit ==
3'd7)//∂��ڱ���ID���ˣ����  ��б�ʶ
                byte_over <= `UD 1'b1;
            else
                byte_over <= `UD 1'b0;
        end
    end

    always @(posedge clk)
    begin
        if(state == SEND || state == RECEIV)
        begin
            if(dsu)
                trans_bit <= `UD trans_bit + 1'b1;
            else
                trans_bit <= `UD trans_bit;
        end
```

```
        else
            trans_bit <= `UD 3'd0;
    end

    always @(posedge clk)
    begin
        if(start)//ÿ□o□'□□□□ʰ□□□□byte□□□□
            trans_byte <= `UD 5'd0;
        else if(state == SEND || state == RECEIV)//□□∂ ,□□Ç□□□□
        begin
            if(dsu && trans_bit == 3'd7)//□□□□□□1byteʰ□□□□□□1
                trans_byte <= `UD trans_byte + 1'b1;
            else//□□□□ʰ□ □ ˬ
                trans_byte <= `UD trans_byte;
        end
        else//□□□□ʰ□ □ ˬ
            trans_byte <= `UD trans_byte;
    end

    //================================================================
========
    //     IIC FSM STATE CHANGE
    //================================================================
========
    always @(posedge clk)
    begin
        if(!rstn)
            state <= `UD IDLE;
        else
            state <= `UD state_n;
    end

    // next state set
    always @(*)
    begin
        state_n = state;
        case(state)
            IDLE  :
            begin
                if(start)
                    state_n = S_START;
                else
                    state_n = state;
            end
```

```
            S_START :
            begin
//                if(full_cycle && trans_byte == 3'd0)
//��∂������ʰ��
//                    state_n = SEND;
//                else if(!w_r_2d && trans_byte == ID_ADDR_BYTE &&
dsu)//������ȷ���
//                    state_n = SEND;
                if(dsu)
                    state_n = SEND;
                else
                    state_n = state;
            end
            SEND  :
            begin
                if(trans_bit == 3'd7 & dsu)
                    state_n = S_ACK;
                else
                    state_n = state;
            end
            S_ACK :
            begin
                if(dsu)// & sda_in)
                begin
                    if(w_r_2d)
                    begin
                        if(trans_byte < ID_ADDR_BYTE)//∂��ID+ADDR
                            state_n = SEND;
                        else if(trans_byte < trans_byte_max)//∂������
                            state_n = SEND;
                        else//∂�������
                            state_n = STOP;
                    end
                    else
                    begin
                        if(trans_byte < ID_ADDR_BYTE)//∂��ID+ADDR
                            state_n = SEND;
                        else if(trans_byte ==
ID_ADDR_BYTE)//���¿'��∂��ID'И��  λ
                            state_n = S_START;
                        else//����  ˇ
                            state_n = RECEIV;
                    end
                end
```

```
                else
                    state_n = state;
            end
            RECEIV:
            begin
                if(trans_bit == 3'd7 & dsu)
                    state_n = R_ACK;
                else
                    state_n = state;
            end
            R_ACK :
            begin
                if(dsu)
                begin
                    if(trans_byte < trans_byte_max)
                        state_n = RECEIV;
                    else
                        state_n = STOP;
                end
                else
                    state_n = state;
            end
            STOP  :
            begin
                if(dsu)
                    state_n = IDLE;
                else
                    state_n = state;
            end
            default: state_n = IDLE;
        endcase
    end

    //=============================================================//
endmodule
```

27.6. 实验现象

实验实现 50MHz 时钟频率二分频，可观察到分频器模块输出 25MHz 差分时钟。实验现象使用示波器观察。