10 自定义数码管 IP 实验

10.1. 背景介绍

本次实验将数码管作为从机挂载在 AHB 总线上，Cortex-M3 作为主机控制数码管的显示。

10.1.1. 自定义 IP 设计框架介绍

本次实验以数码管为例，自定义 IP 的基本框架如下图 10. 1 所示：

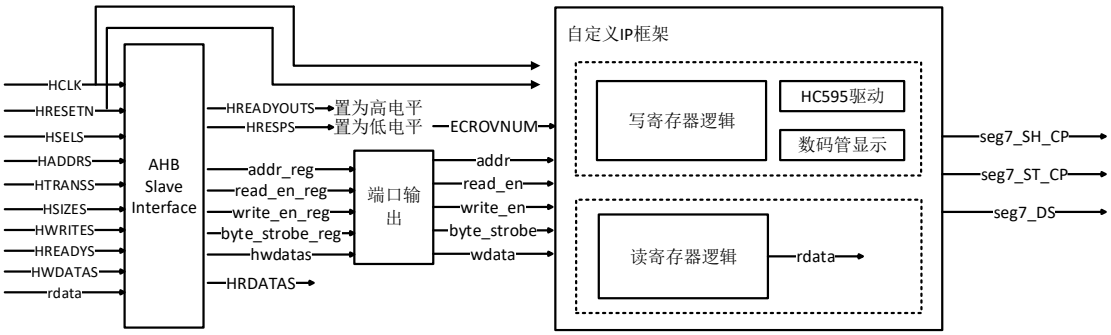

图 10. 1 自定义 IP 的基本框架

从上图可以看出，自定义 IP 的基本框架与第 9 个实验的 AHB Slave 设计框架基本一致，只是将 AHB Slave REG 模块中对于 RAM 的读写改为自己需要的控制信号。

本次实验以数码管为例，详细讲解如何自己设计所需要的 IP：

1. 确定端口信号

AHB Slave Interface 模块使用到的信号在第 9 章的 9.1.1 节中有过介绍，这里将不再赘述。AHB Slave Interface 模块的作用是将 AHB 总线协议转换成简单的寄存器读写协议。自定义 IP 框架部分的端口信号可以分为以下几类：

(1) 全局信号：时钟信号（HCLK）、复位信号（HRESETN）；

(2) AHB 总线信号：地址信号（addr）、读请求信号（read_en）、写请求信号（write_en）、写数据信号（wdata）、读数据信号（rdata）、处理器 AHB-Lite 读/写数据字节通道设定信号（byte_strobe）。

(3) 导出信号：导出到顶层或者需要连接到其它逻辑的信号，比如数码管需要到导出的信号为移位时钟（SH_CP），数据锁存时钟（ST_CP），移位串行数据（DS）。

2. 内部寄存器的定义

根据自己实际需求进行定义，比如数据寄存器、状态寄存器、控制寄存器、

中断屏蔽寄存器、用户自定义寄存器。就数码管的来说，需要使用的寄存器有：控制寄存器和数据寄存器。

3. AHB 总线对寄存器的读写

根据自己的实际需求编写对应的寄存器读写逻辑，比如本次实验中需要通过控制寄存器去控制数码管的显示使能，数据寄存器控制每一位数码管需要显示的值。

10.2. 实验介绍

本次实验将详细介绍如何自定义数码管 IP，主要分为两步进行开发：首先在 FPGA 侧用 Verilog 语言编写对应的寄存器逻辑及寄存器所需要控制的从机功能代码（本次实验对应的就是数码管显示代码及 HC595 的驱动代码，实验将不做详细介绍）；然后通过 MDK 软件对寄存器地址进行读写操作，从而驱动寄存器去实现需要实现的功能，本次实验最终需要实现的功能就是在数码管上显示数字“01234567”。

10.3. 建立 HQFPGA 工程

将实验八的工程复制至存放本次实验的工程文件夹下，然后修改对应的工程名和模块名，修改方式参考实验二中 2.3 节的内容。

10.4. 添加新的模块文件

找到本次实验例程中存放数码管显示代码和 HC595 驱动代码（CM3_System \SEG_7\rtl \SEG_7）的文件夹,将该文件夹复制至你自己的工程文件中，操作方式如下图 10. 2 所示。

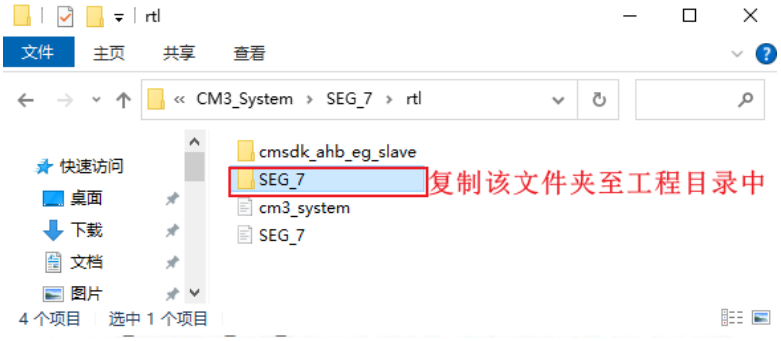

图 10. 2 复制需要的模块文件至工程文件夹之中

打开 HQ 软件，在工程中添加对应的工程代码，如下图 10. 3 所示。

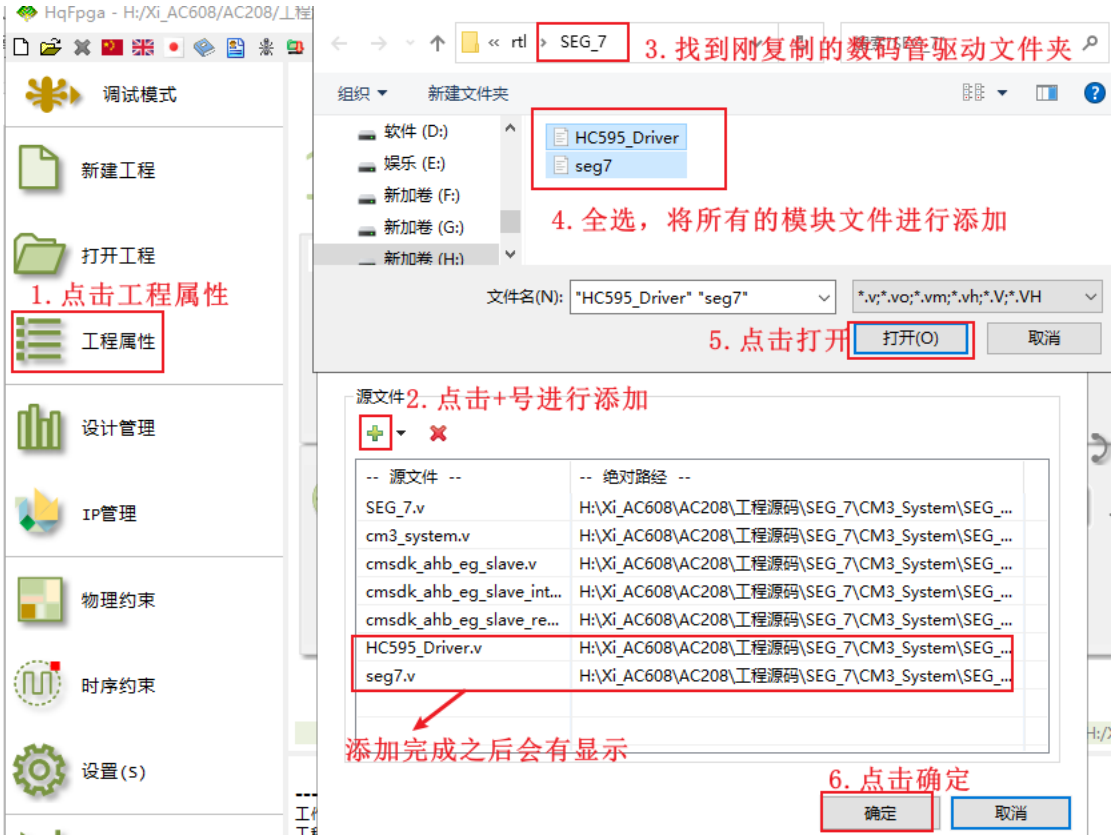

图 10. 3 在 HQ 软件中添加对应的模块文件

10.5. FPGA 侧代码编写

根据 10.1.1 一节的介绍，对第 9 个实验的“cmsdk_ahb_eg_slave_reg”文件中的代码进行修改。

10.5.1. 定义接口

根据实际需求进行编写对应的接口信号，用户在设计自己需要的 IP 文件时，全局信号和 AHB 总线的信号不需要修改，直接复制使用即可，需要修改的就是导出信号，数码管需要导出的信号：数码管位选（sel），数码管段选（seg），移位时钟（SH_CP），数据锁存时钟（ST_CP），移位串行数据（DS）。其中移位时钟（SH_CP），数据锁存时钟（ST_CP），移位串行数据（DS）需要导出至顶层文件；数码管位选（sel），数码管段选（seg）这两个信号是由数码管显示模块输出至 HC595 驱动模块中的（这两个模块都例化在 ahb_slave_reg 这个文件中），不需要导出至顶层模块。最终的端口信号定义如下所示：

```
module cmsdk_ahb_eg_slave_reg #(
  // parameter for address width
  parameter   ADDRWIDTH=12)
 (
```

```
//全局信号
input  wire                  hclk,       // clock
input  wire                  hresetn,    // reset
//AHB 总线信号
input  wire [ADDRWIDTH-1:0]   addr,
input  wire                  read_en,
input  wire                  write_en,
input  wire [3:0]            byte_strobe,
input  wire [31:0]           wdata,
input  wire [3:0]            ecorevnum,
output reg  [31:0]           rdata,
//导出信号
output wire        seg7_SH_CP,   //seg7.SH_CP
output wire        seg7_ST_CP,   //.ST_CP
output wire        seg7_DS,      //.DS
output [7:0]sel,
output [7:0]seg);
```

10.5.2. 内部寄存器定义

本次实验需要使用的寄存器：控制寄存器控制数码管的使能显示；数据寄存器的低 32 位，控制数码管的低 4 位数码管的显示；数据寄存器的高 32 位，控制数码管的高 4 位数码管的显示。定义的寄存器如下所示：

```
assign wr_sel[0] = ((addr[11:2]==10'b0000000000)&(write_en)) ? 1'b1: 1'b0; //控制寄存器 0x00
assign wr_sel[1] = ((addr[11:2]==10'b0000000001)&(write_en)) ? 1'b1: 1'b0; //数据寄存器低 32 位 0x04
assign wr_sel[2] = ((addr[11:2]==10'b0000000010)&(write_en)) ? 1'b1: 1'b0; //数据寄存器高 32 位 0x08
```

10.5.3. 内部寄存器的读写逻辑

1. 写寄存器逻辑

根据不同的寄存器的地址，写入对应的值。本次实验中，使用控制寄存器控制数码管的使能，利用数据寄存器对数码管的高 32 位和低 32 位进行写值，也就是对每一位数码管的段选写值，实现代码如下所示，这里以高 32 位的为例：

```
always @(posedge hclk or negedge hresetn)
begin
    if (~hresetn)
    begin
        dataH <= {32{1'b0}}; // reset data register to 0x00000000
    end
    else if (wr_sel[2])//数据寄存器的高 32 位
    begin
        if (byte_strobe[0])
            dataH[ 7: 0] <= wdata[ 7: 0];
```

```
        if (byte_strobe[1])
            dataH[15: 8] <= wdata[15: 8];
        if (byte_strobe[2])
            dataH[23:16] <= wdata[23:16];
        if (byte_strobe[3])
            dataH[31:24] <= wdata[31:24];
    end
end
```

2. 读寄存器逻辑

根据读回来的寄存器地址的不同，判断回读的数据属于哪一个寄存器中的值，代码如下所示：

```
always @ (read_en or addr or regControl or dataH or dataL)
begin
   case (read_en)
     1'b1:
     begin
       if (addr[11:5] == 8'h00) begin
         case(addr[4:2])
           3'b000: rdata =  regControl;    //0xA0000000
           3'b001: rdata =  dataL;   //0xA0000004
           3'b010: rdata =  dataH;      //0xA0000008
           default: rdata = {32{1'bx}};
         endcase
       end
       else begin
         rdata = {32'h00000000}; // default
       end
     end
   1'b0:
     begin
       rdata =  {32{1'b0}};
     end
   default:
     begin
       rdata =  {32{1'bx}};
     end
   endcase
end
```

10.5.4. 例化模块端口

10.5.4.1. 数码管显示模块（seg7）

数码管显示驱动模块的框图如下图 10. 4 所示。接口列表如下表 10- 1 所示。

图 10. 4 数码管模块框图

表 10- 1 数码管模块接口列表

信号名称	I/O	功能描述
CLK	I	系统时钟，本次实验是 25M
Rst_n	I	复位信号
control	I	数码管使能信号：1 使能，0 关闭
disp_data	I	8 个数码管待显示的数据，每四位组成一个 BCD 码
sel[7:0]	O	数码管位选，选择当前要显示的数码管
seg[6:0]	O	数码管段选，当前要显示的内容

从上表可以分析得出，该模块的 control 信号应该接入控制寄存器 regControl，然后通过 CM3 向控制寄存器中写入值来使能对应的数码管，disp_data 信号接入数据寄存器 dataH、dataL，通过向数据寄存器中写入值来控制数码管需要显示的值。模块例化代码如下所示：

```
seg7 seg7(
    .Clk(hclk),
    .Rst_n(hresetn),
    .control(regControl),
    .disp_data({dataH, dataL}),
    .sel(sel),
    .seg(seg)
);
```

10.5.4.2. 74HC595 驱动模块（HC595_Driver）

74HC595 驱动模块的框图如下图 10. 5 所示，该模块的接口功能描述如下表 10- 2 所示。

图 10. 5 74HC595 驱动模块框图

表 10- 2 74HC595 驱动模块接口功能描述

信号名称	I/O	功能描述
CLK	I	系统时钟，本次实验是 25M
Rst_n	I	复位信号
En	I	数码管使能信号：1 使能，0 关闭
Data	I	8 个数码管数码管的 SEG 和 SEL 数据
SH_CP	O	移位寄存器的时钟输出
ST_CP	O	存储寄存器的时钟输出
DS	O	串行数据输出

从上表可以看出，数码管显示模块输出的 sel 和 seg 信号应该接入该模块的 Data 信号上。模块的输出信号 SH_CP、ST_CP 和 DS 信号导出至顶层模块，最终分配给开发板上 74HC595 芯片对应的管脚。74HC595 驱动模块的例化代码如下所示：

```
HC595_Driver HC595_Driver(
    .Clk(hclk),
    .Rst_n(hresetn),
    .Data({seg,sel}),
    .S_EN(1'b1),
    .SH_CP(seg7_SH_CP),
    .ST_CP(seg7_ST_CP),
    .DS(seg7_DS)
);
```

10.5.4.3. cm3_system 文件中添加数码管 IP 模块端口

在 cm3_system 文件，添加数码管的输出信号接口，如下所示：

```
//数码管接口
output wire        seg7_SH_CP,   //seg7.SH_CP
output wire        seg7_ST_CP,   //.ST_CP
output wire        seg7_DS      //.DS
```

将数码管的 IP 模块例化至 cm3_system 文件中，例化代码如下所示（本次实验使用的 AHB 总线扩展端口 0）：

```
cmsdk_ahb_eg_slave #(32) ahb_slave(
  .HCLK      (PLL_OUT),// Clock
  .HRESETn   (MTXRSTN),  // Reset
  .ECOREVNUM (4'b0), // Engineering-change-order revision bits

  // AHB connection to master
  .HSELS     (TARGEXP0HSEL),
  .HADDRS    (TARGEXP0HADDR),
  .HTRANSS   (TARGEXP0HTRANS),
  .HSIZES    (TARGEXP0HSIZE),
  .HWRITES   (TARGEXP0HWRITE),
```

```
  .HREADYS   (TARGEXP0HREADYMUX),
  .HWDATAS   (TARGEXP0HWDATA),
  .HREADYOUTS(TARGEXP0HREADYOUT),
  .HRESPS    (TARGEXP0HRESP),
  .HRDATAS   (TARGEXP0HRDATA),

  //数码管
  .seg7_SH_CP(seg7_SH_CP),   //seg7.SH_CP
  .seg7_ST_CP(seg7_ST_CP),   //.ST_CP
  .seg7_DS(seg7_DS)      //.DS
);
```

10.5.5. 修改顶层模块

在顶层模块中添加数码管的输出端口，最终的顶层模块文件如下所示：

```
module SEG_7(
  input    wire                MAIN_CLK,        //CM3 时钟
  output   wire                CLKOUT,
  output   wire                PCLK,
  output   wire                MTX_CLK,
  input    wire                MTXRSTN,       //CM3 复位

  input    wire                swdclk,        //debug 接口
  inout    wire                swddio,        //debug 接口
  inout[31:0]                  GPIO,           //32 位通用 GPIO

  //数码管接口
  output wire        seg7_SH_CP,   //移位时钟
  output wire        seg7_ST_CP,   //数据锁存时钟
  output wire        seg7_DS      //移位串行数据
);
wire CLK100M;
wire PLL_Locked;
cm3_system cm3_system(
  .MAIN_CLK(MAIN_CLK),
  .CLKOUT(CLKOUT),
  .PCLK(PCLK),
  .MTX_CLK(MTX_CLK),
  .PLL_OUT(CLK100M),
  .MTXRSTN(MTXRSTN),
  .swdclk(swdclk),
  .swddio(swddio),
  .GPIO(GPIO),
  .seg7_SH_CP(seg7_SH_CP),   //seg7.SH_CP
```

```
  .seg7_ST_CP(seg7_ST_CP),   //.ST_CP
  .seg7_DS(seg7_DS)      //.DS
);
PLL PLL(
  .CLKI(MAIN_CLK),
  .CLKOP(CLK100M),
  .LOCK(PLL_Locked)
);
endmodule
```

10.6. 物理管脚约束

在设计管理器界面，点击设计文件中的“ac208.upc”文件，添加对于数码管的引脚分配，如下所示：

```
#----------------数码管接口-----------------
#-------------数码管移位时钟-----------------
phycst.pin.set {seg7_SH_CP} E4
#-------------数码管数据锁存时钟-------------
phycst.pin.set {seg7_ST_CP} G6
#-------------移位串行数据-------------------
phycst.pin.set {seg7_DS} H2
```

10.7. 编译设计

保存完修改后的顶层文件和物理约束文件之后，点击全部运行，生成本次实验需要的 FPGA 侧的 bin 文件，操作如图 10.6 所示。

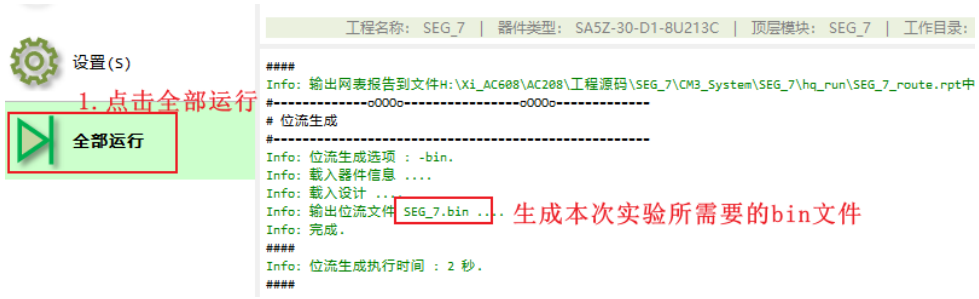

图 10.6 编译设计

10.8. 建立 MDK 工程

找到已有的 MDK 工程，复制至本次工程的 CM3_Softerware 文件夹下，对其进行修改方式参考实验二中 2.8 节中的内容。

10.9. 软件设计

本次实验需要添加的库文件为 seg7（在提供的例程的 HARDWARE 文件夹下进行复制添加），将不需要的库文件删除，节约代码空间。

10.9.1. SEG7

10.9.1.1. Write_Seg7Reg

向数码管寄存器地址中写入对应的值。函数输入：address:寄存器地址；data:写入寄存器中的数据，函数代码如下所示：

```
void Write_Seg7Reg(uint32_t *address,uint32_t data)
{
    *address = data;
}
```

对于寄存器的地址，定义在“seg.h”文件中，使用的时候直接调用即可：

```
//数码管对应的寄存器地址：基地址+偏移地址（使用的 AHB 总线扩展端口 0）
#define SEG_CONTROL CM3DS_MPS2_TARGEXP0_BASE+SEG_CONTROL_OFFSET
#define SEG_DATAH   CM3DS_MPS2_TARGEXP0_BASE+SEG_DATAH_OFFSET
#define SEG_DATAL   CM3DS_MPS2_TARGEXP0_BASE+SEG_DATAL_OFFSET
```

关于寄存器偏移地址的得来可以参看实验 8 中 8.7 节中的内容。

10.9.1.2. Read_Seg7Reg

读寄存器操作函数，函数输入：address:寄存器地址。函数代码如下所示：

```
uint32_t Read_Seg7Reg(uint32_t address)
{
    uint32_t *data;
    data = (unsigned int *)address;
    return *data;
}
```

10.9.1.3. SEG7_ENABLE

数码管使能函数，函数输入：segx：使能哪一位数码管，需要使用哪一位数码管就将该位至 1，比如需要使用到第二个数码管（从右往左第 2 个），segx=0x02(0000 0010)。函数代码如下所示：

```
void SEG7_ENABLE(uint8_t segx)
{
    Write_Seg7Reg((uint32_t *)SEG_CONTROL,segx);
}
```

10.9.1.4. Seg7_SetOneValue

数码管显示函数，函数输入：segx:哪一位数码管需要显示，取值范围 0~7。函数代码如下所示：

```
void Seg7_SetOneValue(uint8_t segx,uint8_t data)
{
    Write_Seg7Reg((uint32_t *)(SEG_DATAL+segx),data);
}
```

需要注意的是：segx=0，对应的是从左往右的第一个数码管，与数码管使能函数是相反的，为了防止用的时候弄混，对其需要使用的数码管定义在“seg.h”文件中。比如：如果只需要最低位数码管显示 1，代码举例如下：

```
SEG7_ENABLE(Enable_Seg_1);//使能数码管
Seg7_SetOneValue(seg_1,1);//是该位数码管显示值
```

10.9.2. 添加用户代码

实验功能：数码管上显示数字“01234567”。

实验代码：使能全部数码管，利用 for 循环实现数码管的显示。最终的用户代码如下所示：

```
#include "CM3DS_MPS2.h"
#include "seg7.h"

int main()
{
    int i = 0;
    SEG7_ENABLE(Enable_ALLSeg);//使能全部数码管
    for(i = 0;i < 8;i++)
        Seg7_SetOneValue(i,i);
}
```

10.10. 板级验证

10.10.1. 实验所需硬件

(1) AC208 开发板
(2) FPGA 下载器：XIST USB Cable
(3) CM3 仿真器：DAP Link
(4) 电源线一根

10.10.2. 硬件连接

硬件连接参考实验一。

10.10.3. 下载文件至目标板

下载文件的方式参考实验一。

10.10.4. 功能演示

将程序下载至开发板之后，数码管依次显示“01234567“，如下图 10. 7 所示：

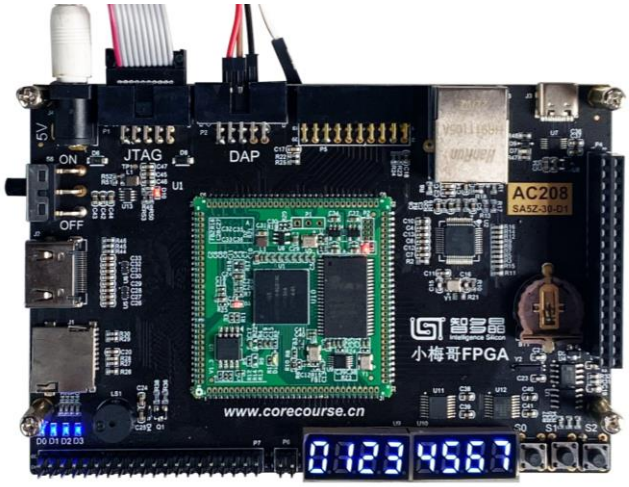

图 10. 7 数码管最终显示

10.11. 思考与总结

本次实验以数码管为例，讲解了自定义 IP 的基本框架及其代码实现，最终实现了对数码管显示的控制。用户可以根据本次实验的内容去定义自己需要的 IP，对于寄存器的定义和模块的例化都是基于“cmsdk_ahb_eg_slave_reg”这个文件中的代码进行修改的，用户使用的修改这个文件中的代码并将需要的端口导出至顶层即可。