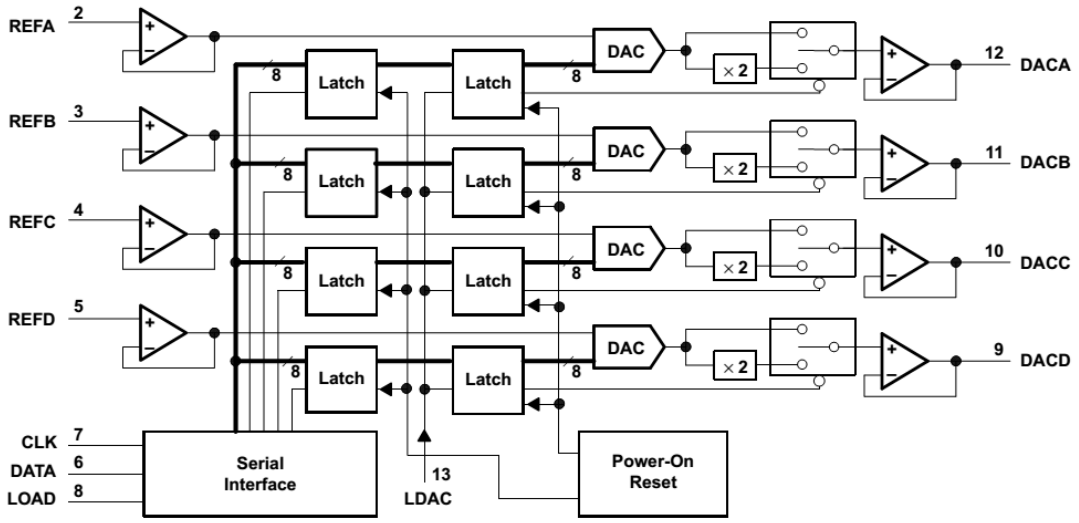
三、TLC5620 型 DAC 驱动设计

TLC5620 型 DAC 芯片概述：

- TLC5620C 是一个具有 4 个独立 8 位电压输出型 DAC 的数模转换器
- 单电源 5V 供电
- 采用串行接口时序
- 具备 4 个高阻抗参考电压输入端口（对应四个 DAC 输出通道）
- 可编程的电压倍增模式

TLC5620 是一个内部具备 4 个独立 8 位电压输出型数字——模拟转换器，每个 DAC 转换器都拥有一个带缓冲（高输入阻抗）的参考电压输入端口。每个 DAC 可以输出一倍或者两倍的参考电压与 GND 之间的电压值。

TLC5620 使用 CMOS 电平兼容的三线制串行总线与各种流行的处理器进行连接，TLC5620 接收控制器发送过来的 11 位的命令字，这 11 位的控制字被分为 3 个部分，包括 8 位的数据位，2 位的 DAC 选择位，1 位的电压倍增控制位。每个 DAC 的寄存器都采用双缓冲结构，这样，可以实现首先通过数据总线给所有的 DAC 传输需要更新的数据，然后通过控制信号 LDAC 将所有 DAC 的电压同步更新到输出上。



TLC5620 芯片内部框图

TLC5620 型 DAC 芯片引脚说明：

引脚名	编号	IO	功能描述
CLK	7	I	串行接口时钟，每个时钟的下降沿，输入数字总线上的数据被移入内部的接口寄存器中
DACA	12	O	DAC A 模拟输出端口
DACB	11	O	DAC B 模拟输出端口
DACC	10	O	DAC C 模拟输出端口
DACD	9	O	DAC D 模拟输出端口
DATA	6	I	串行接口的数字数据输入线，发送给 DAC 的数据是通过串行的方式传入 DAC 的寄存器的，每个数据位都在时钟的下降沿被移入内部寄存器中
GND	1	I	GND
LDAC	13	I	加载 DAC（更新 DAC 待输出数据），当该信号为高电平时，串行总线上传入的数据不会更新到 DAC 上去，只有当 LDAC 的电平由高电平变为低电平时，数据才会更新到 DAC 上去
LOAD	8	I	串行数据加载控制，当 LDAC 为低电平时，LOAD 的下降沿将带输出数据锁存到输出锁存器并立即产生输出电压。
REFA	2	I	DAC A 的参考电压，该电压决定了输出电压的范围，输出电压为 0~VREFA 或者 0~2*VREFA（2VREFA <= VDD）
REFB	3	I	DAC B 的参考电压，该电压决定了输出电压的范围，输出电压为 0~VREFB 或者 0~2*VREFB（2VREFB <= VDD）
REFC	4	I	DAC C 的参考电压，该电压决定了输出电压的范围，输出电压为 0~VREFC 或者 0~2*VREFC（2VREFC <= VDD）
REFD	5	I	DAC D 的参考电压，该电压决定了输出电压的范围，输出电压为 0~VREFD 或者 0~2*VREFD（2VREFD <= VDD）
VDD	14	I	正电源输入

TLC5620 型 DAC 芯片详细介绍：

TLC5620 是由四个电阻串式 DAC 组成的，每个 DAC 的核心是一个拥有 256 个节点（抽头）的电阻，对应了 256 中不同的组合，如下表所示，每个电阻串的一段连接到 GND，另一端来自参考输入缓存的输出。

每个 DAC 的输出都接有一个可配置增益的输出放大器，该放大器的增益可以配置为 1 或者 2。当芯片上电时，DAC 的值全部被复位到 0,。每个 DAC 通道的输出可由下列公式计算得出：

Vo（DAC A|B|C|D） = REF * CODE/256 *（1 + RNG bit value）

其中，Vo 为输出电压值，REF 为 DAC 的输出参考电压，CODE 为输出电压值的数字量化量，如 255 表示按照参考电压的满幅输出（关闭电压倍增模式），0 则 0V 输出，RNG bit value 表示电压倍增模式，为 0 则关闭输出电压倍增模式，为 1 则打开输出电压倍增模式。
当串行控制字中的数据部分为 0~255，RNG bit 为 0 或者 1 时，输出电压与数字量化值的关系如下表所示：

D7	D6	D5	D4	D3	D2	D1	D0	输出电压
0	0	0	0	0	0	0	0	GND
0	0	0	0	0	0	0	1	1/256 * REF(1+RNG)
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
0	1	1	1	1	1	1	1	127/256 * REF(1+RNG)
1	0	0	0	0	0	0	0	128/256 * REF(1+RNG)
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
1	1	1	1	1	1	1	1	255/256 * REF(1+RNG)

TLC 5620 型 DAC 接口时序：

控制器对 TLC5620 的单个 DAC 设置包括两个主要操作

1. 将数字量化值以及控制位发送到 TLC5620 中对应的寄存器中
2. 控制 DAC 将寄存器中接收到的数据值更新到 DAC 输出上

对于数据的传输，有连续传输（11 个连续的时钟周期传输 11 位的控制字）和 2 个 8 时钟周期传输方式（使用两次 8 时钟周期的传输来实现 11 位数据的传输）。

对于数据的更新，则使用 LOAD 和 LDAC 配合以实现。

当 LOAD 为高电平时，在每个 CLK 的下降沿，数据被移入 DAC 的移位寄存器中。当所有的数据位被移入完成后，LOAD 被拉低，以将数据从串行输入移位寄存器中转入选中的 DAC 中，如上图所示：

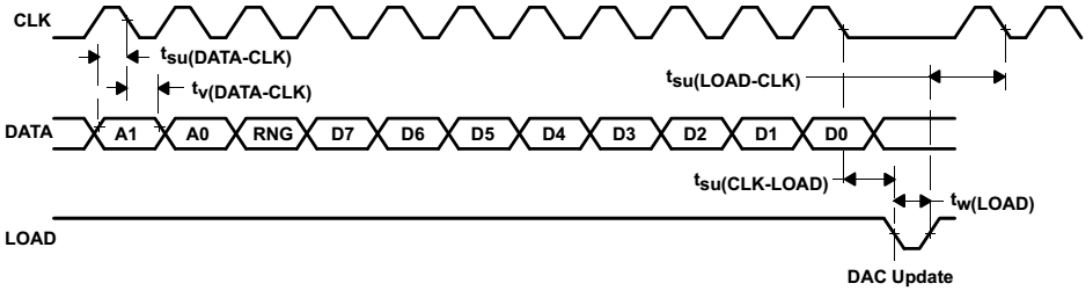
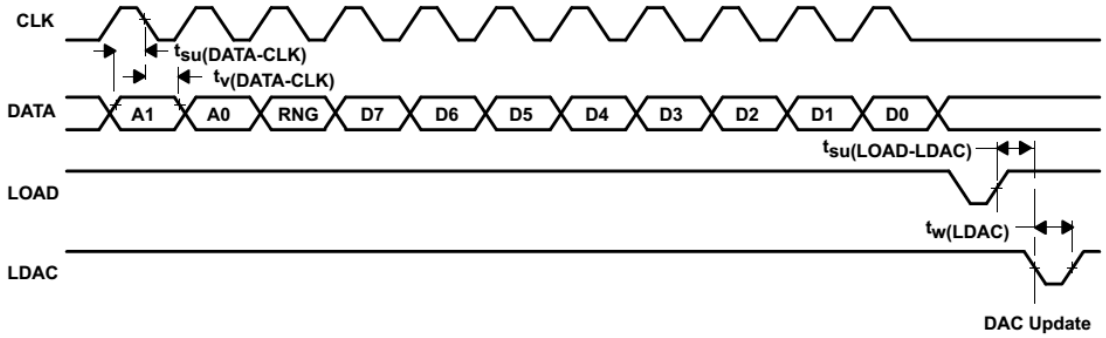


Figure 1. LOAD-Controlled Update (LDAC = Low)

当 LDAC 为低电平时，选中的 DAC 通道的输出电压在 LOAD 变为低电平时更新。

当 LDAC 在串行数据传输过程中为高电平时，新的数据值被存在器件中，该值可以在稍后将 LDAC 拉低时传入 DAC 的输出，如下图 2 所示。串行总线上传输数据时，高位在前，低位在后。



使用两个 8 时钟周期的传输数据（主要针对 8 位定长的 SPI 控制器）的时序图 3 和图 4 所示：

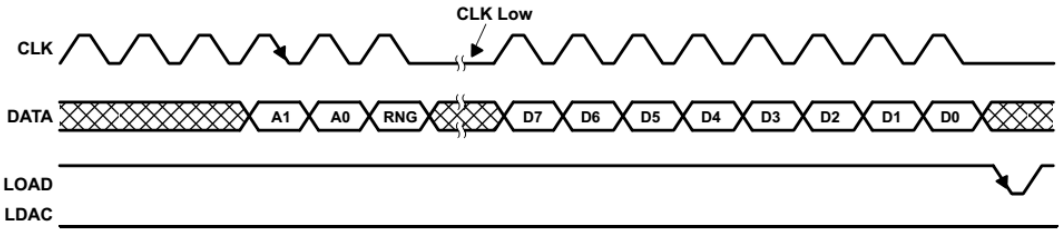


Figure 3. Load-Controlled Update Using 8-Bit Serial Word (LDAC = Low)

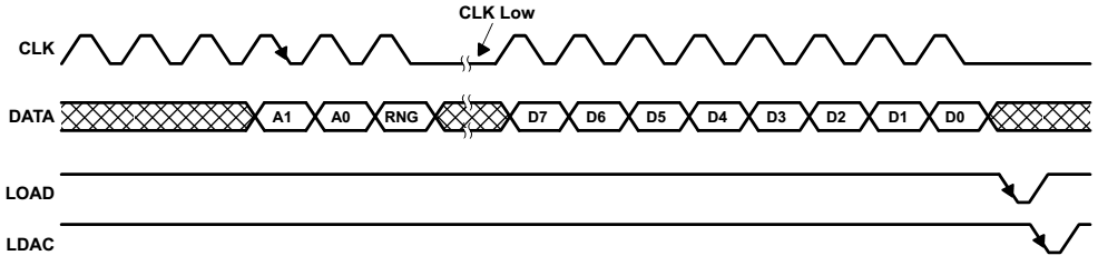


Figure 4. LDAC-Controlled Update Using 8-Bit Serial Word

在传输时序中，标为 A0 和 A1 的两位指定了需要设置输出的 DAC，具体 A0 和 A1 值与对应被选择更新的 DAC 如下表所示：

A1	A0	被更新的 DAC 通道
0	0	DACA
0	1	DACB
1	0	DACC
1	1	DACD

TLC5620 串行数字接口的关键时序参数：

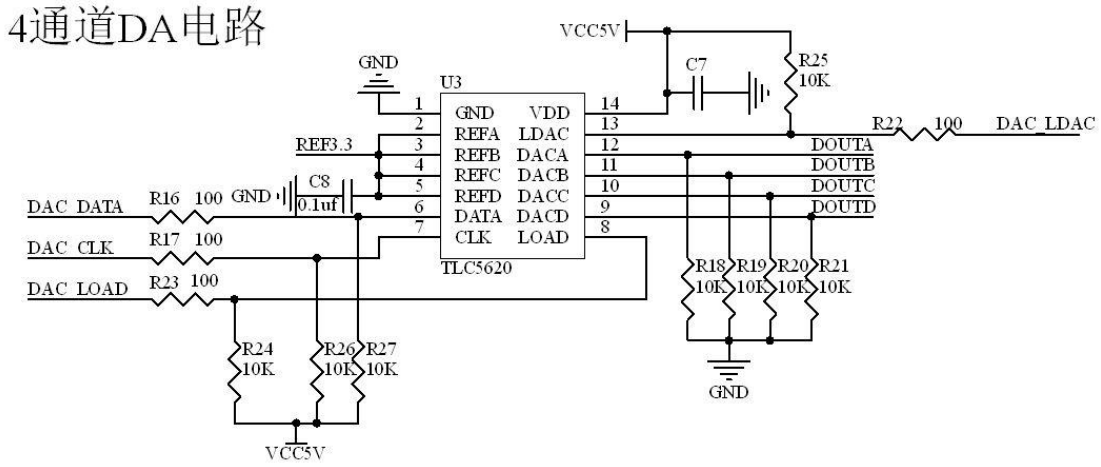
针对 TLC5620 的数字接口，其操作时序如下表所示。

		MIN	NOM	MAX	UNIT
Supply voltage, V_{DD}		4.75		5.25	V
High-level input voltage, V_{IH}		0.8 V_{DD}			V
Low-level input voltage, V_{IL}				0.8	V
Reference voltage, V_{ref} [A|B|C|D]				$V_{DD}-1.5$	V
Analog full-scale output voltage, R_L = 10 kΩ			3.5		V
Load resistance, R_L		10			kΩ
Setup time, data input, $t_{su(DATA-CLK)}$ (see Figures 1 and 2)		50			ns
Valid time, data input valid after CLK↓, $t_{v(DATA-CLK)}$ (see Figures 1 and 2)		50			ns
Setup time, CLK eleventh falling edge to LOAD, $t_{su(CLK-LOAD)}$ (see Figure 1)		50			ns
Setup time, LOAD↑ to CLK↓, $t_{su(LOAD-CLK)}$ (see Figure 1)		50			ns
Pulse duration, LOAD, $t_{w(LOAD)}$ (see Figure 1)		250			ns
Pulse duration, LDAC, $t_{w(LDAC)}$ (see Figure 2)		250			ns
Setup time, LOAD↑ to LDAC↓, $t_{su(LOAD-LDAC)}$ (see Figure 2)		0			ns
CLK frequency				1	MHz
Operating free-air temperature, T_A	TLC5620C	0		70	°C
	TLC5620I	−40		85	°C

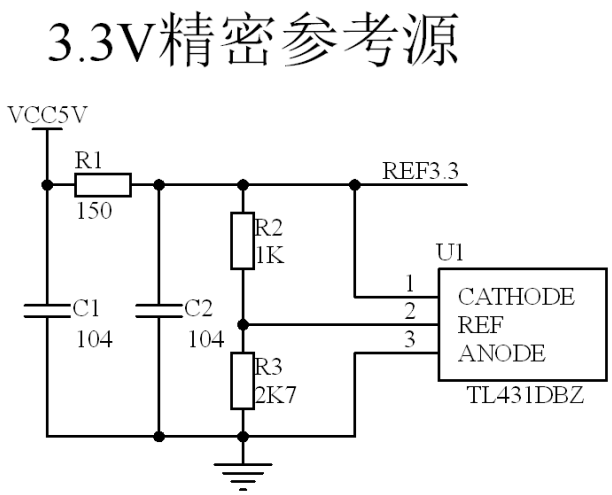
其中，从 $t_{SU(DATA-CLK)}$、开始一直到 CLK frequency 都是我们在设计接口时序时，需要重点关注的参数。我们设计的控制时序必须要严格满足表中各个时序参数，否则会导致数据传输或转换失败。

芯航线 ADDA 模块 TLC5620 电路介绍：

芯航线 FPGA 学习套件中，提供了一个多通道串行 ADDA 模块。其中，DA 部分所使用的芯片就是上文介绍的 TLC5620，TLC5620 部分电路图如下图所示：



为了给 DAC 的参考输入提供稳定的参考电压，这里使用专用精密参考源芯片 TL431 搭建了一个参考源电路，该电路如下图所示：



根据 5V 的输入电压和输出电压/电流设计电路，按照上图设置电路即可，其中 R2：R3=1：2.7 得到的输出最接近 3.3V（例如 R1 取值为 1k，R2 取值为 2.7k）

$$Vout = (R2+R3)*2.5/R3 = 3.7*2.5/2.7 = 3.42V$$

为了保证 TL431 1mA 的工作电流，R1 需要满足

$$1mA< (Vcc-Vout)/R1< 500mA$$

这里设置 R1 为 150 欧姆，则(Vcc-Vout)/R1 = 10.5mA，满足 TL431 工作要求。

因此，当确定一个输出电压时，就可以得到对应的 RNG 和 CODE 了，如下式所示：

$$\mathrm{CODE} = \frac{\mathrm{Vo} * 256}{\mathrm{REF} * (1 + \mathrm{RNG})} = \frac{\mathrm{Vo} * 256}{3.42 * (1 + \mathrm{RNG})}$$

然后，在我们控制 DAC 的输出时，只需根据所需输出的电压计算得到 CODE 和 RNG，然后将该值通过串行接口传入 TLC5620，再发出一个更新控制信号（LOAD + LDAC），就能实现控制 TLC5620 输出想要的电压了。

线性序列机设计思想与 TLC5620 接口时序设计：

这里以使用 LOAD 信号控制 DAC 更新的时序图来分析 TLC5620 的数字接口时序：

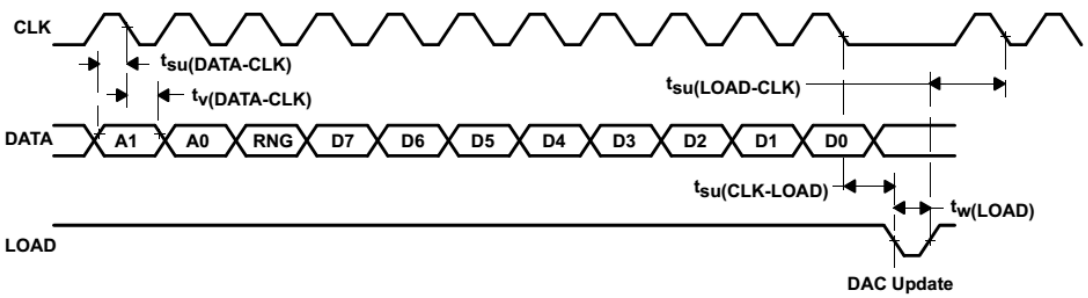
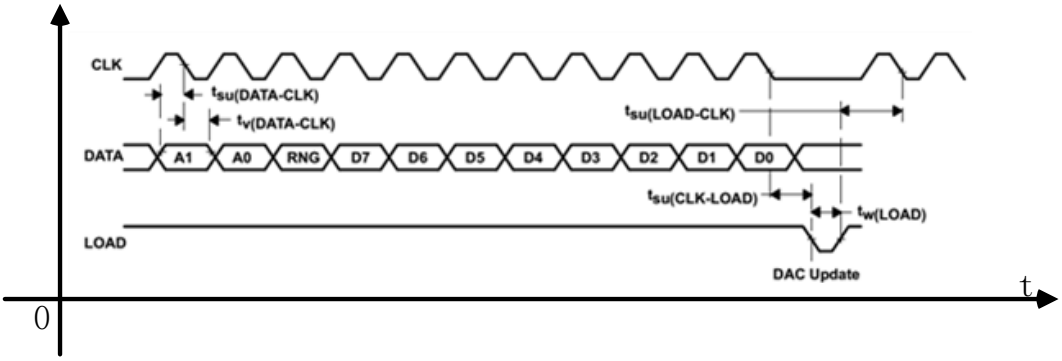


Figure 1. LOAD-Controlled Update (LDAC = Low)

从图中我们可以看到，该接口的时序是一个很有规律的序列，CLK 信号什么时候该由低变高，什么时候由高变低。DATA 信号什么时候该传输哪一位数据，LOAD 信号什么时候拉低，什么时候拉高，都是可以根据时序参数唯一确定下来的。

我们可以将该数据波形放到以时间为横轴的一个二维坐标系中，纵轴就是每个信号对应的状态：



因此我们只需要在逻辑中使用一个计数器来计数，然后每个计数值时就相当于在 t 轴上对应了一个相应的时间点，那么在这个时间点上，各个信号需要进行什么操作，直接赋值即可。

针对 TLC5620 的接口时序，在 FPGA 中，我们以时钟周期为 20ns 进行设计，由于 TLC5620 的数字接口工作时钟最高位 1MHz，周期为 1000ns，因此时钟的翻转时间最小为 500ns，为了给设计留有余量，因此本设计使用 1200ns 作为时钟周期，即时钟信号每 600ns 翻转一次。从而可以通过每个信号变化时的时间得到对应计数器的值：

每个时间点对应信号操作详表：

时间	计数器值	对应信号状态
00	00	CLK= 0； DATA = 0； LOAD = 1； LDAC = 0
200	10	DATA = A1； CLK= 1；
800	40	CLK= 0；
1400	70	DATA = A0； CLK= 1；
2000	100	CLK= 0；
2600	130	DATA = RNG； CLK= 1；
3200	160	CLK= 0；
3800	190	DATA = D7； CLK= 1；
4400	220	CLK= 0；
5000	250	DATA = D6； CLK= 1；
5600	280	CLK= 0；
6200	310	DATA = D5； CLK= 1；
6800	340	CLK= 0；
7400	370	DATA = D4； CLK= 1；
8000	400	CLK= 0；
8600	430	DATA = D3； CLK= 1；
9200	460	CLK= 0；
9800	490	DATA = D2； CLK= 1；

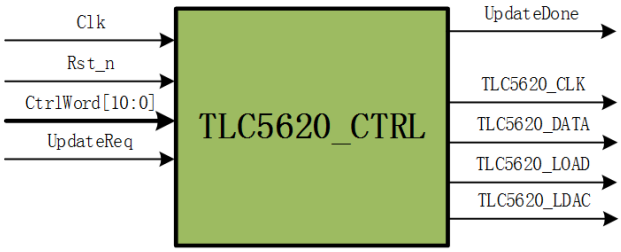
10400	520	CLK= 0；
11000	550	DATA = D1； CLK= 1；
11600	580	CLK= 0；
12200	610	DATA = D0； CLK= 1；
12800	640	CLK= 0；
13400	670	LOAD = 0；
16000	800	LOAD = 1；

每个时刻计数器操作：

Cnt	Cnt Operation
Cnt < 820	Cnt <= Cnt + 1'b1;
Cnt = 820	Cnt <= 0;

以上就是通过线性序列机设计接口时序的一个典型案例，可以看到，线性序列机可以大大简化我们的设计思路。线性序列机的设计思想就是使用一个计数器不断计数，由于每个计数值都会对应一个时间，那么当该时间符合我们需要操作信号的时刻时，就对该信号进行操作。这样，就能够轻松的设计出各种时序接口了。

有了这两张表，我们就可以进行 TLC5620 的接口逻辑的编写了。设计 TLC5620 接口逻辑的模块如下图所示：



其中，每个端口的作用如下所示：

端口名称	I/O	端口功能描述
Clk	I	为控制器的工作时钟，频率为 50MHz，
Rst_n	I	控制器复位，低电平复位
CtrlWord[10：0]	I	控制器控制字，组成为{A1，A0，RNG，DATA[7:0]}
UpdateReq	I	更新 DAC 输出请求信号，在该信号的高电平时，控制器寄存 CtrlWord 上的数据并按照 TLC5620 的接口时序将数据发送出去。每次高电平持续一个时钟周期
UpdateDone	O	更新 DAC 完成标志，每次完成更新产生一个高电平脉冲，脉冲宽度为 1 个时钟周期
TLC5620_CLK	O	TLC5620 的 CLK 接口
TLC5620_DATA	O	TLC5620 的 DATA 接口
TLC5620_LOAD	O	TLC5620 的 LOAD 接口
TLC5620_LDAC	O	TLC5620 的 LDAC 接口

有了这些之后，我们就可以开始进行控制器的具体逻辑设计了。具体逻辑设计过程请参看小梅哥 FPGA 设计思想与验证方法视频第 17 课。

视频教程中的工程源码：

```
/*=====================================================================
   *
   *  LOGIC CORE:          TLC5620_CTRL
   *  MODULE NAME:         TLC5620_CTRL()
   *  COMPANY:             芯航线电子工作室
   *                      http://xiaomeige.taobao.com
   *   author:                 小梅哥
   *   author QQ Group: 472607506
   *  REVISION HISTORY:
   *
   *    Revision 1.0  01/01/2016    Description: Initial Release.
   *
   *   FileType    : Testbench
   *
   *  FUNCTIONAL DESCRIPTION:
   *

=====================================================================*/
   //Vo（DAC A|B|C|D） = REF * CODE/256 *（1 + RNG bit value）
001 module TLC5620_CTRL(
002     Clk,
003     Rst_n,
004     UpdateReq,
005     CtrlWord,
006
007     UpdateDone,
008     TLC5620_CLK,
009     TLC5620_DATA,
010     TLC5620_LOAD,
011     TLC5620_LDAC
012 );
013
014
015     input Clk;
016     input Rst_n;
017     input UpdateReq;    //更新输出电压请求
018     input [10:0]CtrlWord;//ADDR[1:0];RNG bit;CODE[7:0]
019
020     output reg UpdateDone;      //更新输出电压完成标志信号
021     output reg TLC5620_CLK;     //TLC5620 接口时钟信号
```

```
022     output reg TLC5620_DATA;    //TLC5620 数据输入信号
023     output reg TLC5620_LOAD;    //
024     output reg TLC5620_LDAC;
025
026     reg [9:0] Cnt;  //线性序列机计数器
027
028     //线性序列机计数器计数进程
029     always@(posedge Clk or negedge Rst_n)
030     if(!Rst_n)
031         Cnt <= 10'd0;
032     else if(UpdateReq == 1 | (Cnt != 10'd0))begin
033         if(Cnt == 10'd820)
034             Cnt <= 10'd0;
035         else
036             Cnt <= Cnt + 10'd1;
037     end
038     else
039         Cnt <= 10'd0;
040
041
042 //线性序列机控制进程
043     always@(posedge Clk or negedge Rst_n)
044     if(!Rst_n)begin
045         TLC5620_CLK <= 1'b0;
046         TLC5620_DATA <= 1'b0;
047         TLC5620_LOAD <= 1'b0;
048         TLC5620_LDAC <= 1'b0;
049         UpdateDone <= 1'b0;
050     end
051     else begin
052         case(Cnt)
053             0:
054                 begin
055                     TLC5620_CLK <= 1'b0;
056                     TLC5620_DATA <= 1'b0;
057                     TLC5620_LOAD <= 1'b1;
058                     TLC5620_LDAC <= 1'b0;
059                     UpdateDone <= 1'b0;
060                 end
061             10:
062                 begin
063                     TLC5620_CLK <= 1'b1;
064                     TLC5620_DATA <= CtrlWord[10];
065                 end
```

```
066             40: TLC5620_CLK <= 1'b0;
067
068             70:
069                 begin
070                     TLC5620_CLK <= 1'b1;
071                     TLC5620_DATA <= CtrlWord[9];
072                 end
073
074             100: TLC5620_CLK <= 1'b0;
075             130:
076                 begin
077                     TLC5620_CLK <= 1'b1;
078                     TLC5620_DATA <= CtrlWord[8];
079                 end
080             160: TLC5620_CLK <= 1'b0;
081             190:
082                 begin
083                     TLC5620_CLK <= 1'b1;
084                     TLC5620_DATA <= CtrlWord[7];
085                 end
086             220: TLC5620_CLK <= 1'b0;
087             250:
088                 begin
089                     TLC5620_CLK <= 1'b1;
090                     TLC5620_DATA <= CtrlWord[6];
091                 end
092             280: TLC5620_CLK <= 1'b0;
093             310:
094                 begin
095                     TLC5620_CLK <= 1'b1;
096                     TLC5620_DATA <= CtrlWord[5];
097                 end
098             340: TLC5620_CLK <= 1'b0;
099             370:
100                 begin
101                     TLC5620_CLK <= 1'b1;
102                     TLC5620_DATA <= CtrlWord[4];
103                 end
104             400: TLC5620_CLK <= 1'b0;
105             430:
106                 begin
107                     TLC5620_CLK <= 1'b1;
108                     TLC5620_DATA <= CtrlWord[3];
109                 end
```

```
110             460: TLC5620_CLK <= 1'b0;
111             490:
112                 begin
113                     TLC5620_CLK <= 1'b1;
114                     TLC5620_DATA <= CtrlWord[2];
115                 end
116             520: TLC5620_CLK <= 1'b0;
117             550:
118                 begin
119                     TLC5620_CLK <= 1'b1;
120                     TLC5620_DATA <= CtrlWord[1];
121                 end
122             580: TLC5620_CLK <= 1'b0;
123             610:
124                 begin
125                     TLC5620_CLK <= 1'b1;
126                     TLC5620_DATA <= CtrlWord[0];
127                 end
128             640: TLC5620_CLK <= 1'b0;
129             670:TLC5620_LOAD <= 1'b0;
130             800:TLC5620_LOAD <= 1'b1;
131             820:UpdateDone <= 1'b1;
132             default:;
133         endcase
134     end
135
136 endmodule
137
```

视频教程中的测试文件源码：

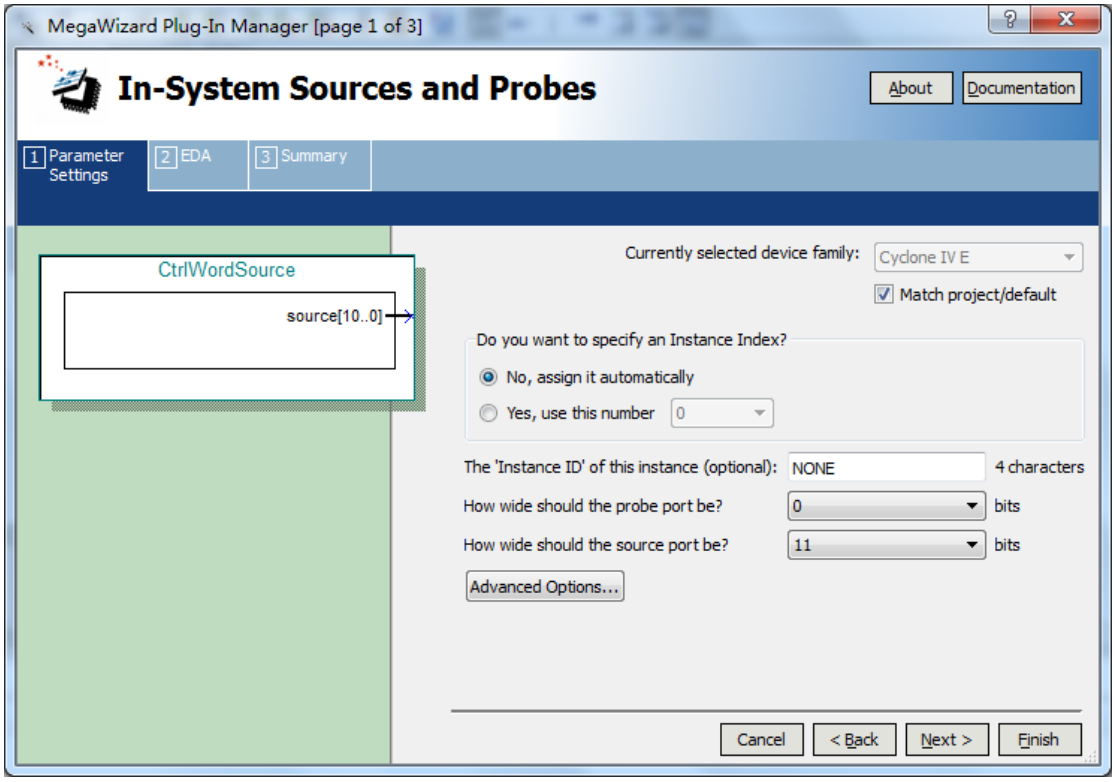
```
01  `timescale 1ns/1ns
02  `define clk_period 20
03  module TLC5620_CTRL_tb;
04
05      reg Clk;
06      reg Rst_n;
07      reg UpdateReq;
08      reg [10:0]CtrlWord;
09
10      wire UpdateDone;
```

```
11      wire TLC5620_CLK;
12      wire TLC5620_DATA;
13      wire TLC5620_LOAD;
14      wire TLC5620_LDAC;
15
16      TLC5620_CTRL TLC5620_CTRL0(
17          .Clk(Clk),
18          .Rst_n(Rst_n),
19          .UpdateReq(UpdateReq),
20          .CtrlWord(CtrlWord),
21
22          .UpdateDone(UpdateDone),
23          .TLC5620_CLK(TLC5620_CLK),
24          .TLC5620_DATA(TLC5620_DATA),
25          .TLC5620_LOAD(TLC5620_LOAD),
26          .TLC5620_LDAC(TLC5620_LDAC)
27      );
28
29      initial Clk = 1;
30      always #(`clk_period/2) Clk = ~Clk;
31
32      initial begin
33          Rst_n = 1'b0;
34          UpdateReq = 1'b0;
35          CtrlWord = 0;
36
37          #(`clk_period*100 + 1);
38          Rst_n = 1'b1;
39          #(`clk_period*20);
40          CtrlWord = {2'd0,1'b0,8'haa};
41          UpdateReq = 1'b1;
42          #(`clk_period);
43          UpdateReq = 1'b0;
44          @(posedge UpdateDone);
45          #(`clk_period*40);
46
47          CtrlWord = {2'd0,1'b0,8'h55};
48          UpdateReq = 1'b1;
49          #(`clk_period);
50          UpdateReq = 1'b0;
51          @(posedge UpdateDone);
52          #(`clk_period*20);
53          $stop;
54      end
```

```
55
56  endmodule
```

板级验证方法：

设计中使用了一个信号探针来通过电脑传递输出电压控制字给 DAC 控制逻辑，使用 In system sources and probes editor 工具，输入希望输出的电压值，则芯航线开发板上，FPGA 控制 TLC5620 芯片输出对应的电压值



顶层例化模块源码：

```
01  module TCL5620_TOP(
02      Clk,
03      Rst_n,
04
05      TLC5620_CLK,
06      TLC5620_DATA,
07      TLC5620_LOAD,
08      TLC5620_LDAC
09  );
10
```

```
11      input Clk;
12      input Rst_n;
13
14      output TLC5620_CLK;
15      output TLC5620_DATA;
16      output TLC5620_LOAD;
17      output TLC5620_LDAC;
18
19      wire [10:0]CtrlWord;
20
21      CtrlWordSource  CtrlWordSource(
22          .probe(),
23          .source(CtrlWord)
24      );
25
26      TLC5620_CTRL TLC5620_CTRL0(
27          .Clk(Clk),
28          .Rst_n(Rst_n),
29          .UpdateReq(1'b1),
30          .CtrlWord(CtrlWord),
31
32          .UpdateDone(),
33          .TLC5620_CLK(TLC5620_CLK),
34          .TLC5620_DATA(TLC5620_DATA),
35          .TLC5620_LOAD(TLC5620_LOAD),
36          .TLC5620_LDAC(TLC5620_LDAC)
37      );
38
39  endmodule
```

如果希望更加细致详细的学习本实验，请观看《小梅哥 FPGA 设计思想与验证方法视频教程》第 17 课。

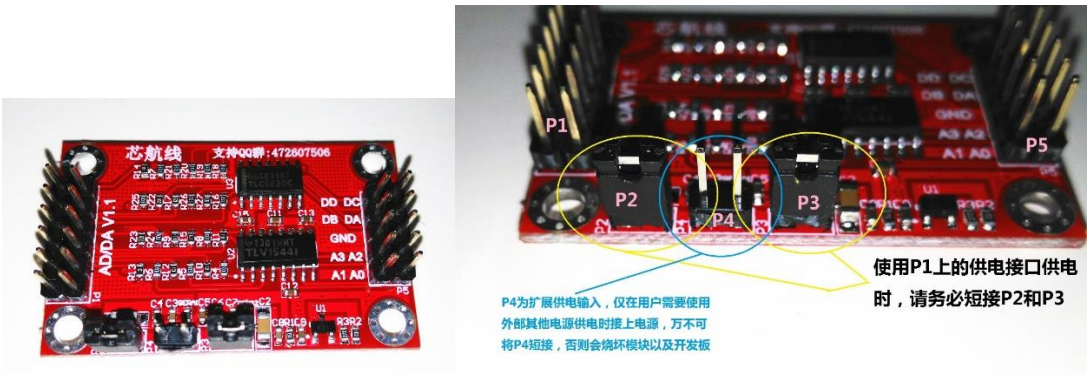
如有更多问题，欢迎加入芯航线 FPGA 技术支持群：472607506

小梅哥
芯航线电子工作室

附录 1：ADDA V1.1 模块使用说明

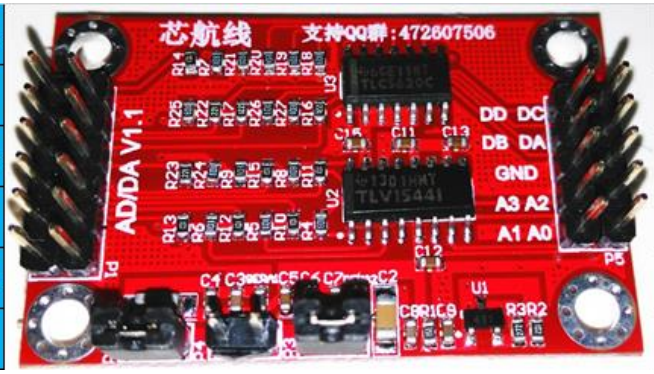
本模块提供两种连接方式，分为开放式接口和芯航线 FPGA 学习套件专用接口。开放式接口焊接普通排针，方便用于使用杜邦线与其他非芯航线 FPGA 学习套件的板卡相连。芯航线专用接口使用卧式排针，与芯航线 FPGA 学习套件的 CAMERA 接口相连。现分别进行介绍：

1、开放式接口：



本模块共有 5 个接插件：

P1 为与 FPGA 开发板连接的数字接口信号和电源信号，使用时使用杜邦线与芯航线开发板的 GPIO 0 中相应引脚相连，具体连接位置，可以参照我们提供的示例连接方法；其每个引脚的功能如下图所示：

使用排针引出直插版本，主要针对单买 ADDA 模块的用户，方便用杜邦线与其他板卡连接					
ADC_EOC	DAC_LDAC			DAC_CH_D	DAC_CH_C
DAC_DATA	DAC_CLK			DAC_CH_B	DAC_CH_A
ADC_FS	DAC_LOAD			GND	GND
ADC_nCS	ADC_SDO			ADC_CH_3	ADC_CH_2
ADC_SDI	ADC_SCLK			ADC_CH_1	ADC_CH_0
GND	VCC5V				

P2 为 VCC5V 的连通短接点，如果使用 P1 上的供电引脚为模块供电，则需要将 P2 短接；

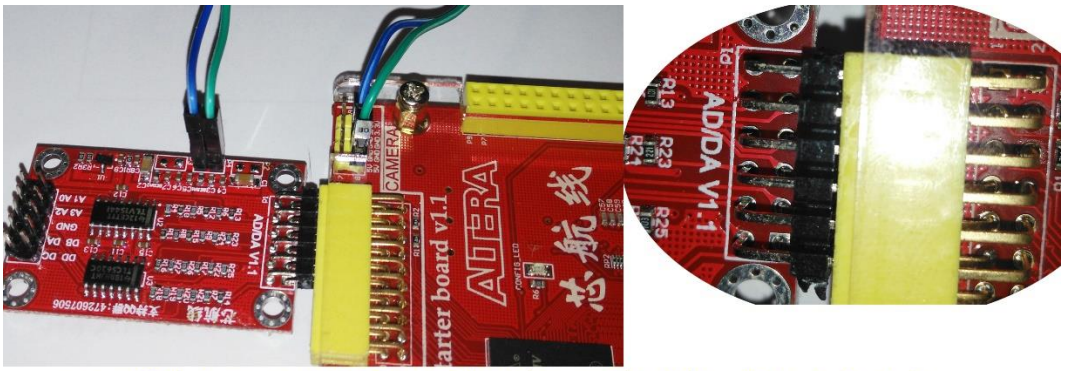
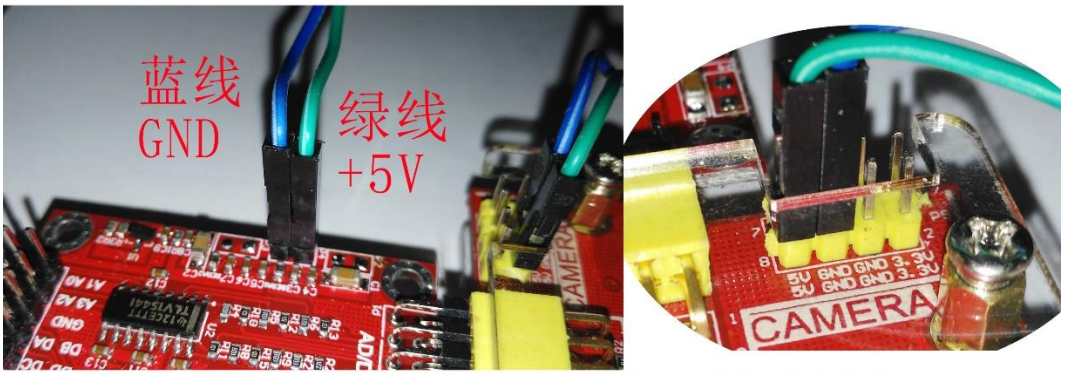
P3 为 GND 的连通短接点，如果使用 P1 上的 GND 引脚为 GND 连接到开发板，则需要将 P3 短接；

P4 为扩展供电引脚，由于芯航线核心板本身为高速数字电路，因此电源噪声相对较高，在一些需要精密测量的应用中，往往使用专用的独立供电电路为模块供电，因此可以直接将外部供电使用杜邦线接到 P4 上，其中 P4 靠近 P2 的针接 VCC，P4 靠近 P3 的针接 GND，P3 需要保持短接状态，P2 则需要断开。

注意，无论何时，请不要短接 P4，否则会烧毁开发板。

2、针对芯航线 FPGA 学习板套件的专用插接方式：

针对购买我们开发板的用户，我们直接焊接的卧式排针，可以直接插接到我们芯航线 FPGA 学习套件的 Camera 接口上使用，插接到 Camera 接口时，需要单独供电，供电接口从 P4 接口通过杜邦线与开发板的供电输出引脚相连。如下图所示：

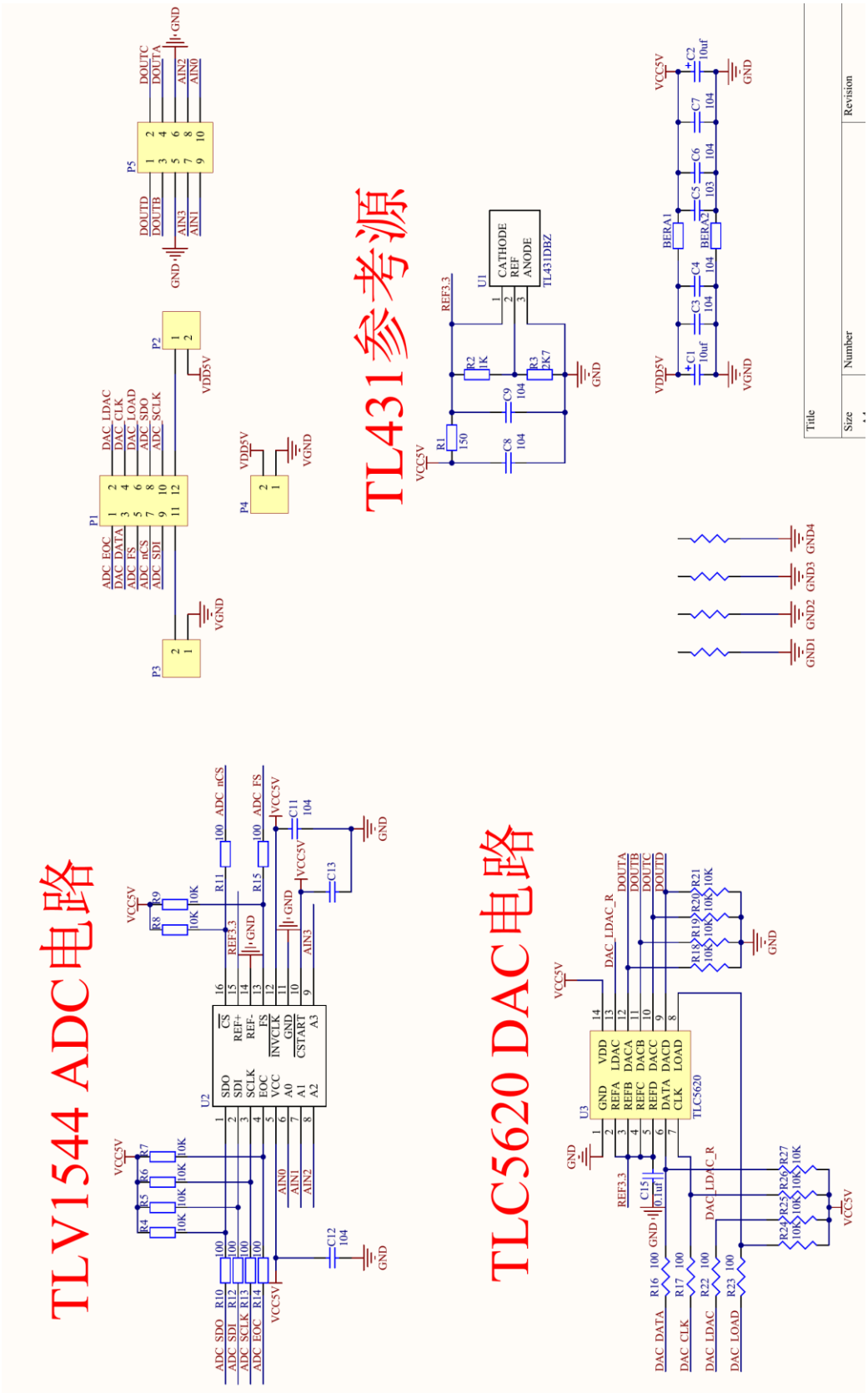
<table>
<tr><td>ADC_EOC</td><td>DAC_LDAC</td><td rowspan="6">  </td><td></td><td></td></tr>
<tr><td>DAC_DAT A</td><td>DAC_CLK</td><td>DAC_CH_ D</td><td>DAC_CH_C</td></tr>
<tr><td>ADC_FS</td><td>DAC_LOA D</td><td>DAC_CH_B</td><td>DAC_CH_ A</td></tr>
<tr><td>ADC_nCS</td><td>ADC_SDO</td><td>GND</td><td>GND</td></tr>
<tr><td>ADC_SDI</td><td>ADC_SCLK</td><td>ADC_CH_3</td><td>ADC_CH_2</td></tr>
<tr><td>GND</td><td>VCC5V</td><td>ADC_CH_1</td><td>ADC_CH_0</td></tr>
<tr><td colspan="2"></td><td>VCC5 V</td><td>GND</td><td></td></tr>
<tr><td colspan="5"> 
ADDA模块与FPGA主板通过摄像头接口直接连接,靠右上角对齐
 
ADDA模块供电从FPGA开发板单独取5V供电，具体线序见上图</td></tr>
</table>

使用此种插接方式时，对应引脚分配如下表所示：

插接在 CMOS 摄像头接口上的引脚分配		
ADC_EOC	CMOS_D4	PIN_B14
ADC_SDO	CMOS_VSYNC	PIN_F7
ADC_SCLK	CMOS_SCLK	PIN_C9
ADC_FS	CMOS_XCLK	PIN_C16

ADC_nCS	CMOS_HREF	PIN_D16
ADC_SDI	CMOS_SDA	PIN_E9
DAC_LDAC	CMOS_D5	PIN_A15
DAC_CLK	CMOS_D7	PIN_C15
DAC_LOAD	CMOS_PCLK	PIN_D15
DAC_DATA	CMOS_D6	PIN_B16

附录 2：ADDA V1.1 模块原理图



如有更多问题，欢迎加入芯航线 FPGA 技术支持群交流学习：472607506

小梅哥
芯航线电子工作室

关于学习资料，小梅哥系列所有能够开放的资料和更新（包括视频教程，程序代码，教程文档，工具软件，开发板资料）都会发布在我的云分享。（记得订阅）链接：http://yun.baidu.com/share/home?uk=402885837&view=share#category/type=0