SOPC 技术实验教程

EDA 实验室

目 录

第一章 TD-EDA 实验系统概述

随着大规模可编程器件的快速发展，基于 EDA 技术的 CPLD/FPGA 可编程器件的设计成为电子系统设计的主流。纵观近年来各高校毕业设计及大学生电子设计竞赛，不难发现越来越多的题目需要采用 EDA 技术，使用大规模可编程器件 CPLD/FPGA 来实现。作为培养专业人才的高等院校纷纷开展了相关课程的教学及实验内容。EDA 技术的教学内容可以分为三个层次：

第一 基础数字系统的设计：此部分教学通过对 CPLD/FPGA 芯片及相应的开发工具的学习，使学生对于 EDA 技术及其相关概念有所了解；结合学生对数字逻辑电路知识的掌握，通过具体采用自顶向下的设计方法并应用图形输入、硬件描述语言等设计手段，来设计、实现一些简单的数字逻辑及数字系统应用电路，使学生能初步掌握 CPLD/FPGA 芯片的结构、功能以及 EDA 技术的应用，为以后的学习打下一个基础。

第二 综合数字系统的设计：此部分教学通过一些实例将 CPLD/FPGA 应用于接口、控制、通信等领域。通过这部分的学习使学生掌握 EDA 技术的软、硬件开发环境及开发方法；通过使用 CPLD/FPGA 芯片进行外围接口控制和 IP 模块的使用，使学生能独立设计一些在毕业设计、电子竞赛中可能用到的模块，为更高层次的应用打下基础。这部分内容需要有系统结构、接口技术、通讯原理等基础课程的知识，适用于高年级本科生和研究生。

第三 基于 FPGA 的嵌入式系统设计。随着嵌入式处理器、专用数字器件和 DSP 算法以 IP 核的形式嵌入到 FPGA 中，以单片 FPGA 完成整个嵌入式系统数字部分的设计已经成为现实。由 Altera、Xilinx 等可编程逻辑器件厂商倡导的 SOPC(System On a Programmable Chip-可编程片上系统)技术已在通信、工业控制等领域得到实际应用。国内高校对于 SOPC 技术的发展及应用投入了极大的热情和关注。此部分教学通过学习 Altera 公司提供的 Nios II 软核处理器，了解 SOPC 的基本概念和基于 FPGA 的嵌入式系统的开发方法，掌握 SOPC 硬件开发工具，软件调试工具的使用。

实验设备厂家应根据教学内容的要求，根据不同层次的需求开发出有利于教学和实验的设备。实验设备应该为用户的学习及验证提供一个环境，为用户进行二次开发提供一个平台。为了更好的为高校实验教学服务，体现教学效果，满足实验教学、毕业设计、电子设计竞赛的要求，我公司采用新技术，使用新型可编程器件及设计开发环境，精心设计推出了“TD-EDA 教学实验系统”，该系统全面支持 EDA 技术教学中含盖的各层次的应用，完全满足 EDA 技术教学的需要。

1.1 TD-EDA 系统构成及特点

1.1.1 系统功能及特点

1. 合理的、灵活的系统结构

系统采用开发板＋基本实验平台的结构，采用此结构的优点有:

(1) 体现了实验系统的灵活性，便于用户根据教学的要求，选择不同的开发板，与基本实验平台配合满足不同层次 EDA 教学的要求。

(2) 体现了实验系统的开放性，满足用户二次开发的需要，可以在毕业设计、课题研究中独立作为开发板使用。

(3) 体现了实验系统的易升级性。随着可编程器件的快速发展，功能完善、性价比更高的芯片层出不穷，用户只需要更换、升级开发板，就可以实现设备的更新换代。

2. 充分体现新技术、紧随 EDA 技术的发展趋势

目前可编程逻辑器件成为电子领域发展最快的技术之一，伴随器件的快速发展相应的芯片结构、器件特点、设计思想、开发手段都发生了翻天覆地的变化。TD-EDA 实验系统紧随 EDA 技术的发展趋势，采用新型、集成度更高、性价比更好的 CPLD/FPGA 芯片以及最新的集成开发环境 Quartus II，在 EDA 教学中可以满足不同层次的教学要求。

3. 实验单元模块化，充分考虑了系统稳定性和开放性

实验系统采用单元化电路结构，实验单元丰富，可满足多层次 EDA 技术教学及应用的需要。我公司也根据具体应用的需要，可配套提供多种功能实用的扩展板，为用户提供了灵活多样的选择。

4. 实验内容丰富，层次分明

TD-EDA 实验系统，根据 EDA 技术实际的教学情况选择开发板，以满足不同用户的需求，实验单元丰富，实验内容层次分明，具有教学性、通用性、趣味性。为学生以后的发展打下坚实的基础。

1.1.2 系统构成

TD-EDA 实验系统采用“基本实验平台＋开发板”的结构，用户可选择满足需要的 CPLD/FPGA 开发板。TD-EDA 基本实验平台系统的构成如表 1-1-1 所示。

表 1-1-1 TD-EDA 基本实验平台的构成

单元名称	主要电路内容
CPLD/FPGA 开发板单元	基于 Cyclone 系列 EP1C6 芯片的 SOPC 开发板
信号源单元	方波信号源：提供 20M、10M、1M、100K、10K、1K、100Hz、10Hz、1Hz 频率的方波信号； 正弦波信号源：频率、幅值连续可调的正弦波信号。频率范围：200Hz～80KH ，幅值范围：500mV～10V
输入电路单元	16 组电平开关、4×4 键盘阵列单元、2 组消抖单脉冲
输出显示单元	16 位发光二极管显示、交通灯显示、8 位七段数码管显示、液晶显示单元、16×16 点阵单元
单片机及串口通讯单元	基于 ISP 技术的 51 单片机最小系统单元
接口单元	串口通信单元、VGA 接口单元、PSII 接口单元
数据采集单元	使用 8 位、20MSPS 的高速 AD 转换器和 32K 高速 SRAM 构成一个基本的高速数据采集通道
信号发生单元	使用 32K Flash 存储器和 8 位高速 DA 转换器及滤波电路构成一个基本的信号发生器单元
扩展单元	进口面包板、扩展总线、40 脚编程座
系统电源	+5V/2A,±12V/0.2A,3.3V/0.5A

1.2 TD-EDA 系统的使用方法

TD-EDA 实验系统采用单元化电路，整个系统包括十八个单元。各单元的功能如下所述：

1. CPLD/FPGA 开发板

该单元提供了两个 PC104-40 插座，将用户选择的开发板扣到该单元，便构成了 EDA 教学实验系统。开发板提供的引脚以排针形式引出到 I/O 接口单元。实验时与相应单元连线即可完成实验电路的连接。开发板单元 PC104-40 插座对应的引脚参看第 1.3 节“开发板的使用方法”。

2. I/O 接口单元

该单元将开发板的引脚引出，I/O 接口单元信号线说明如表 1-2-1 所示。

表 1-2-1 I/O 接口单元信号线说明

信号线	说明
IO1～IO66	IO 接口扩展引脚，实验时可作为输入/输出引脚
IN1～IN16 *	输入引脚，5V 设备需要作为输入时使用
CLK0、CLK1、CLK、CLK3	全局时钟引脚，其中 CLK 引脚的频率为开发板上提供的晶振，CLK0、CLK1、CLK3 为开放的时钟引脚
RST	全局复位引脚

***说明**：开发板上使用的可编程器件，均为低功耗、低电压芯片，外接 5V 器件作为输入时不能直接使用。需要串接限流电阻，并且在 Quartus II 软件中打开可编程器件内部的钳位二极管才可作为输入引脚与 5V 器件连接。如图 1-2-1 所示，I/O 接口单元的 IN1～IN16 对应于开发板上 IO51～IO66 引脚经过限流电阻后的引脚。在使用中需要 5V 器件作为输入时，常常连接到 IN1～IN16 引脚。

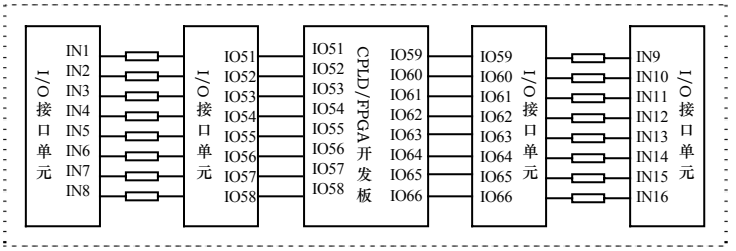


图 1-2-1 5V 器件作为输入引脚

I/O 接口单元的丝印与开发板对应的引脚参看第 1.3 节“开发板的使用方法”。

3. 方波信号源

方波信号源单元提供 20M 到 1Hz 之间 9 种频率脉冲，用于数字系统设计中提供时钟信号，其原理如图 1-2-2 所示。

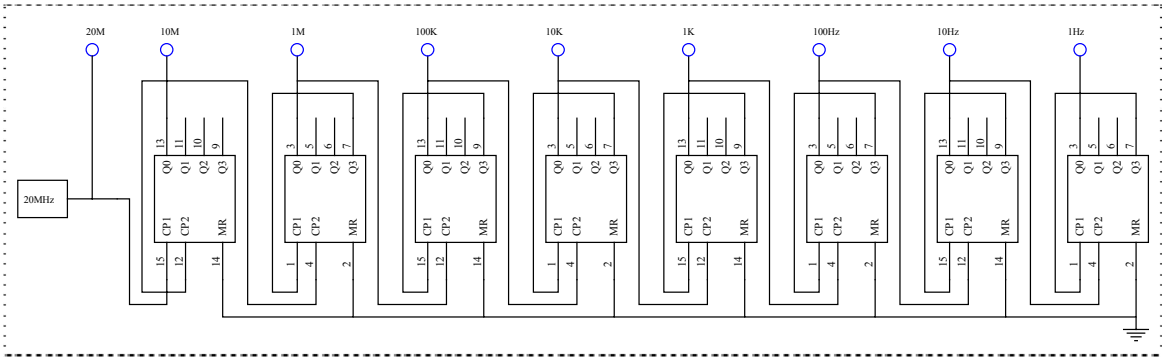


图 1-2-2　方波信号源原理图

4.　正弦波信号源

正弦波信号源单元是采用 DDS 技术实现的数字信号源，经滤波电路后产生幅值和频率可调的正弦波信号。通过档位开关可选择信号的频率范围，频率范围为 200Hz～20K、20K～80K，信号的幅值范围为 500mV～10V。用户在使用时首先通过调零电位器将波形调零。根据需要进行幅值调节和频率调节。其原理框图如图 1-2-3 所示。

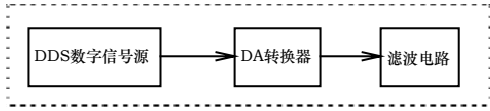


图 1-2-3　正弦波信号源原理图图

5.　七段数码管显示单元

七段数码管显示单元使用了八个共阴极七段数码管，作为输出显示。其原理如图 1-2-4 所示。

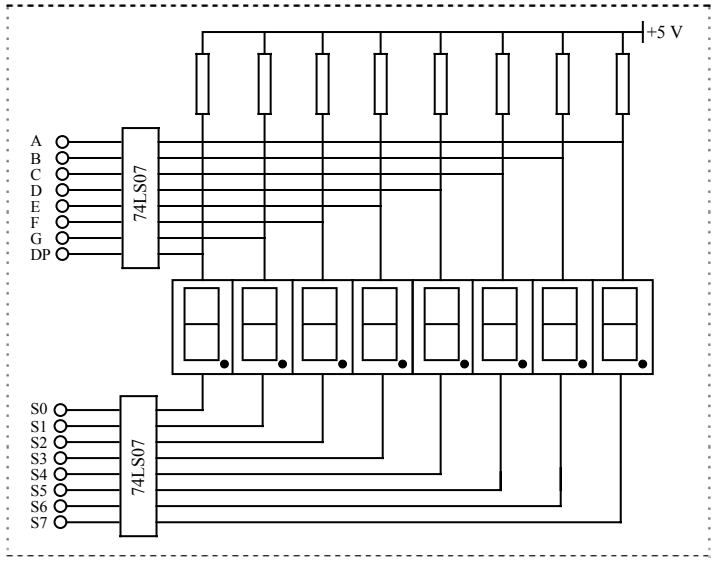


图 1-2-4　七段数码管显示单元原理图

6. 交通灯单元

交通灯单元模拟了一个十字路口的交通灯。采用正逻辑，其原理与 LED 显示单元相同。

7. 点阵单元

点阵单元由四个 8×8 点阵器件构成一个 16×16 的点阵。其中 R1~R16 为行控制信号线，L1~L16 为列控制信号。给某行低电平、某列高电平，则对应的 LED 点亮，如使 R1 为'0'，L1 为'1'，则左上角的 LED 点亮。其原理如图 1-2-5 所示。

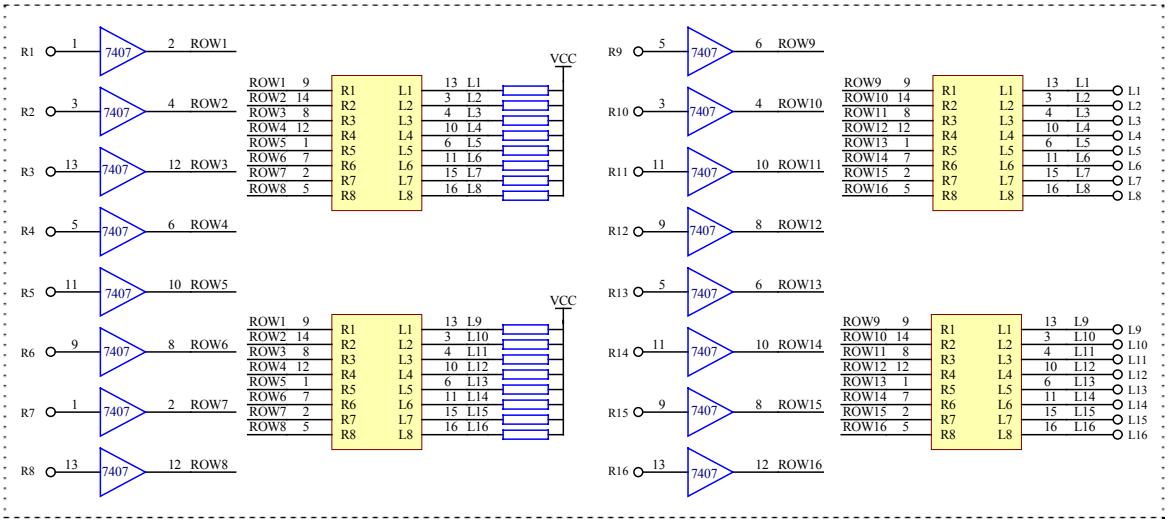


图 1-2-5 点阵单元原理图

8. LED 显示单元

LED 显示单元提供了 16 个显示灯，采用正逻辑，指示逻辑电平。其原理如图 1-2-6 所示。

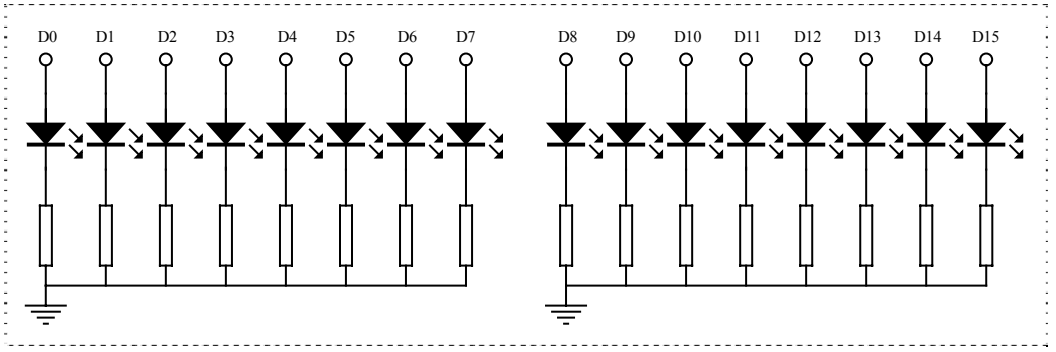


图 1-2-6 LED 显示单元原理图

9. 开关单元

开关单元提供了 16 组拨动开关及显示灯，开关拨上为"1"，显示灯亮。开关拨下为"0"，显示灯灭。其原理如图 1-2-7 所示。

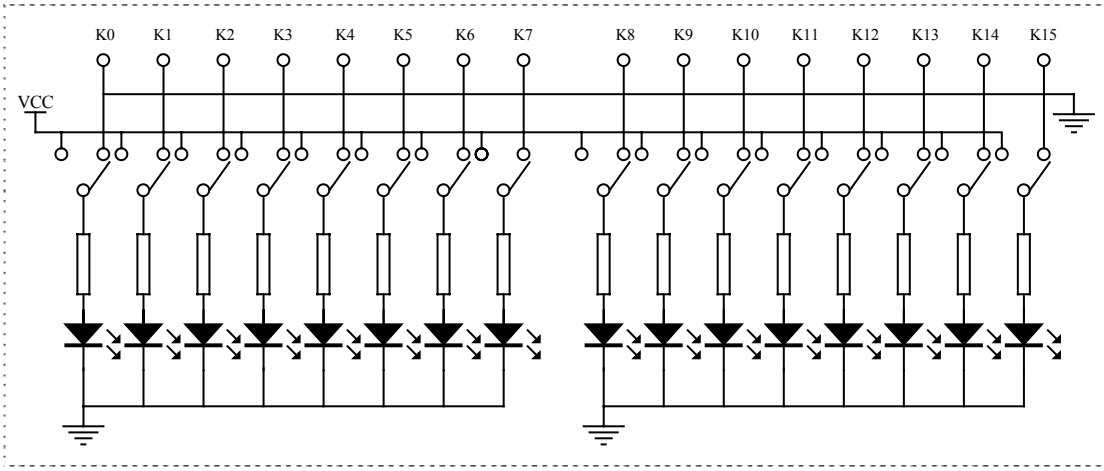


图 1-2-7　开关单元原理图

10.　键盘单元

键盘单元由 16 个按键构成一个 4×4 键盘扫描阵列。其原理如图 1-2-8 所示。

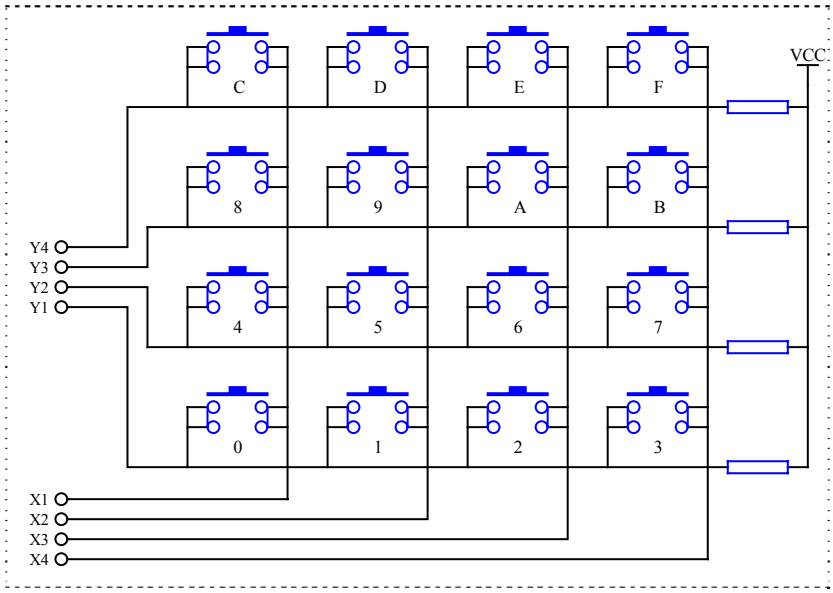


图 1-2-8　键盘单元原理图

11.　单脉冲单元

单脉冲单元提供两个单脉冲触发器，由与非门和微动开关等构成两路 R-S 触发器。输出分为上升沿和下降沿，分别以”+”和”-”表示。其原理如图 1-2-9 所示。

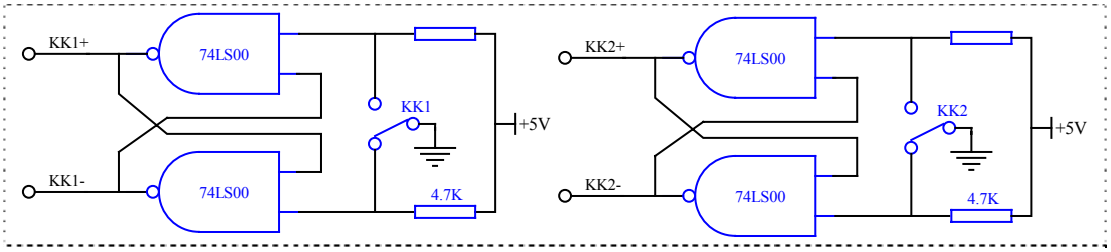


图 1-2-9 单脉冲单元原理图

12. 单片机最小系统单元

单片机最小系统单元由一片基于 ISP 技术的 51 单片机 SST89E554RC 和一片 MAX232 芯片构成了一个单片机最小系统。该单元满足“CPLD+单片机”开发的需要。通过短路块 JP2 的切换，可以提供一个串口通讯的电平转换。其原理如图 1-2-10 所示。

如果要使用单片机最小系统需要将 JP1 短路块短接到 EA＝1、JP2 短路块短接到 T/MC 、R/MC 处。如果要使用独立的串口需要将 JP2 短路块短接到 T/CP 、R/CP 处。

注意：系统默认将 JP2 短路块接到 T/MC、R/MC。将 JP3 短路块接到 EA＝1。

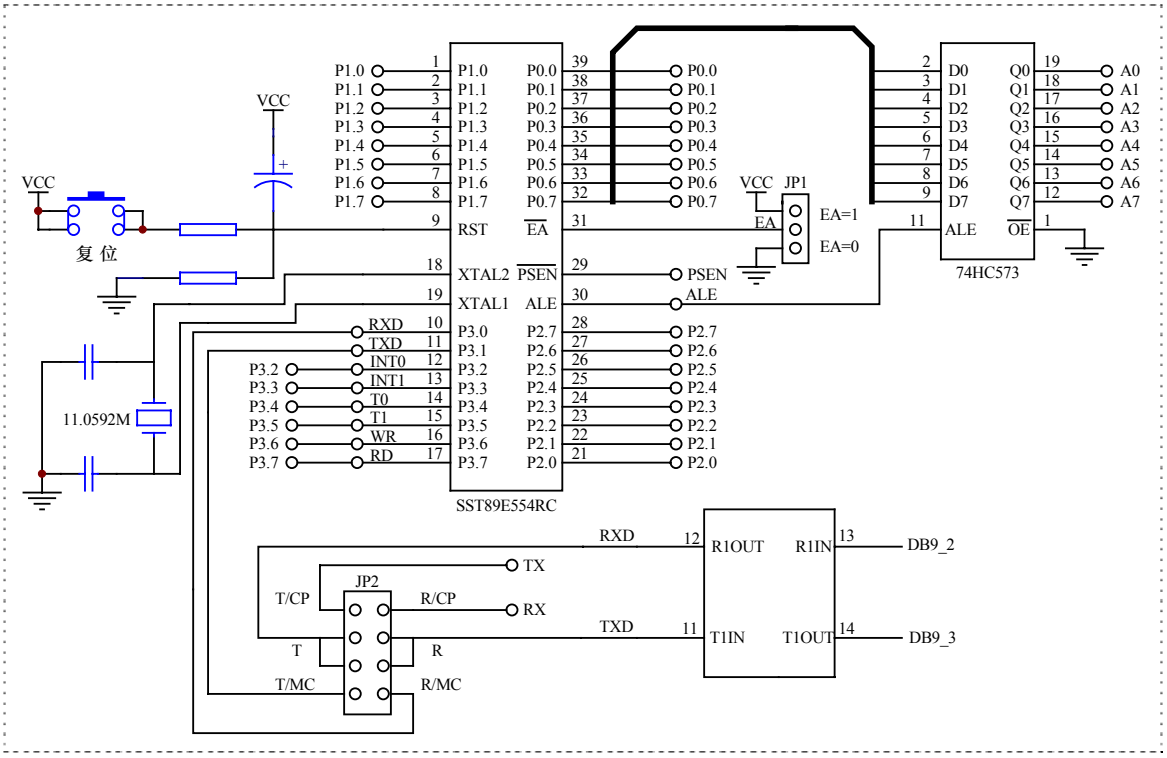


图 1-2-10 单片机最小系统单元原理图

13. 数据采集单元

数据采集单元由一片 8 位高速 AD、一片 32K 的高速 SRAM 存储器构成。该单元可以构成一个数据采集通道，并且将 AD 和存储器完全开放，用户进行二次开发时可以单独使用。通过 JP3 的短路块用户可以选择是否开放数据、地址总线。其原理如图 1-2-11 所示。

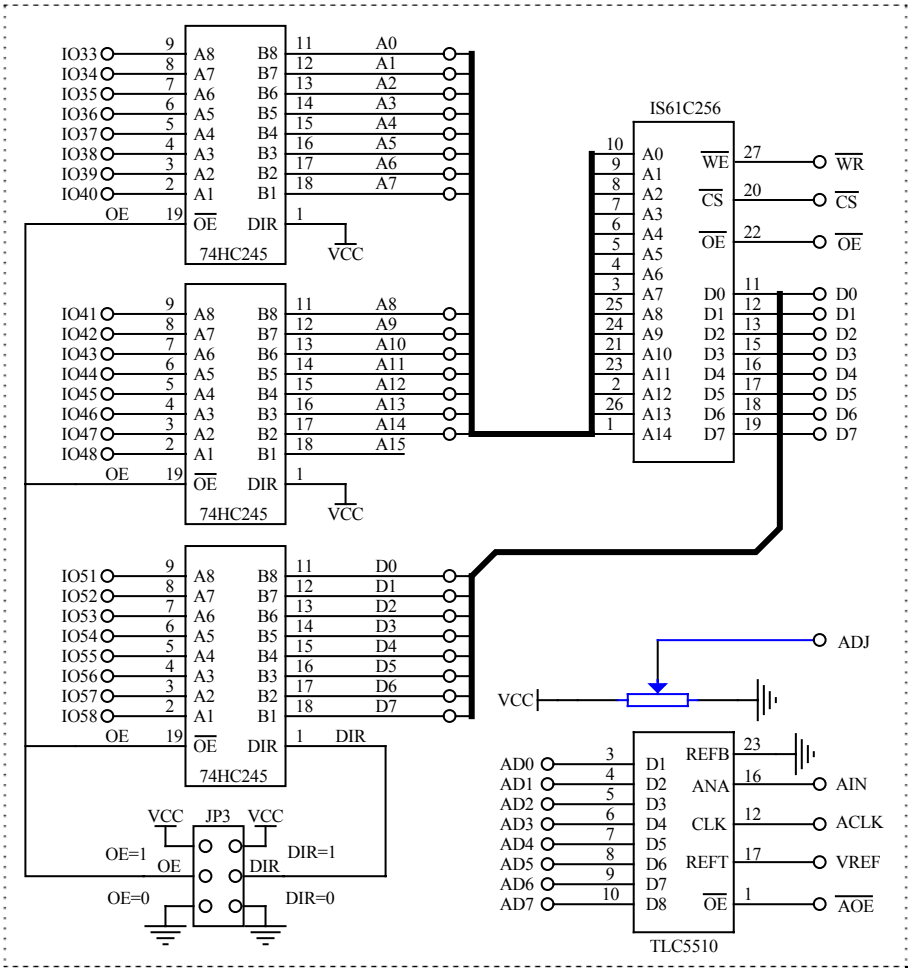


图 1-2-11　数据采集单元原理图

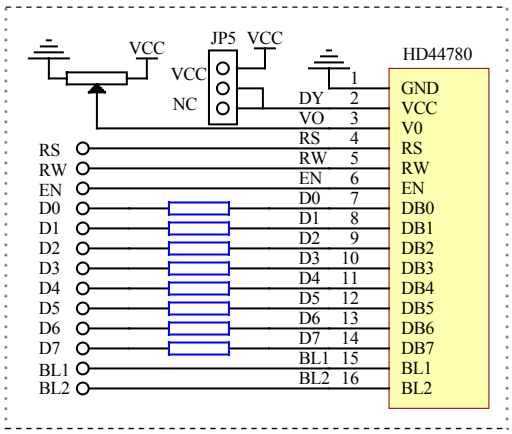


图 1-2-12　液晶显示单元原理图

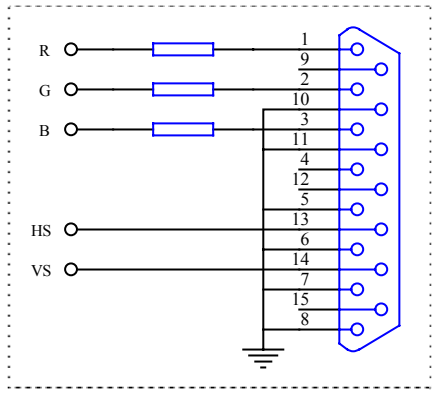


图 1-2-13　VGA 单元原理图

14. 液晶显示单元

液晶显示单元使用的是 16×2 字符型液晶显示器，控制器为 HD44780。其原理如图 1-2-12 所示。如果要使用液晶显示单元需要将 JP5 短路块接到 VCC 处给液晶显示器供电。

注意：系统默认将 JP5 短路块接到 NC。

15. VGA 单元

VGA 单元提供 EDA 实验系统与 VGA 显示器之间通信控制功能。VGA 彩色显示器使用 640×480 的分辨率，60Hz 的刷新率。其原理如图 1-2-13 所示。

16. 信号发生单元

信号发生单元由一片 8 位高速 DA、一片 32K 的 Flash 存储器及一片运放构成。该单元符合 DDS 实验的要求，并且将 DA 和存储器完全开放，用户进行二次开发时可以单独使用。通过 JP3 的短路块用户可以选择是否开放数据、地址总线。其原理如图 1-2-14 所示。

由上图可知当 JP3 短路块接到 OE＝1、DIR＝1 时数据线和地址线关闭，此时 IO33～IO58 可以作为 I/O 引脚使用，DA 和存储器完全开放。当 JP3 短路块接到 OE＝0、DIR＝0 时数据线和地址线接通，地址线和数据线与 IO33～IO58 引脚进行连接。

注意：系统默认将 JP3 短路块接到 OE＝1、DIR＝1。

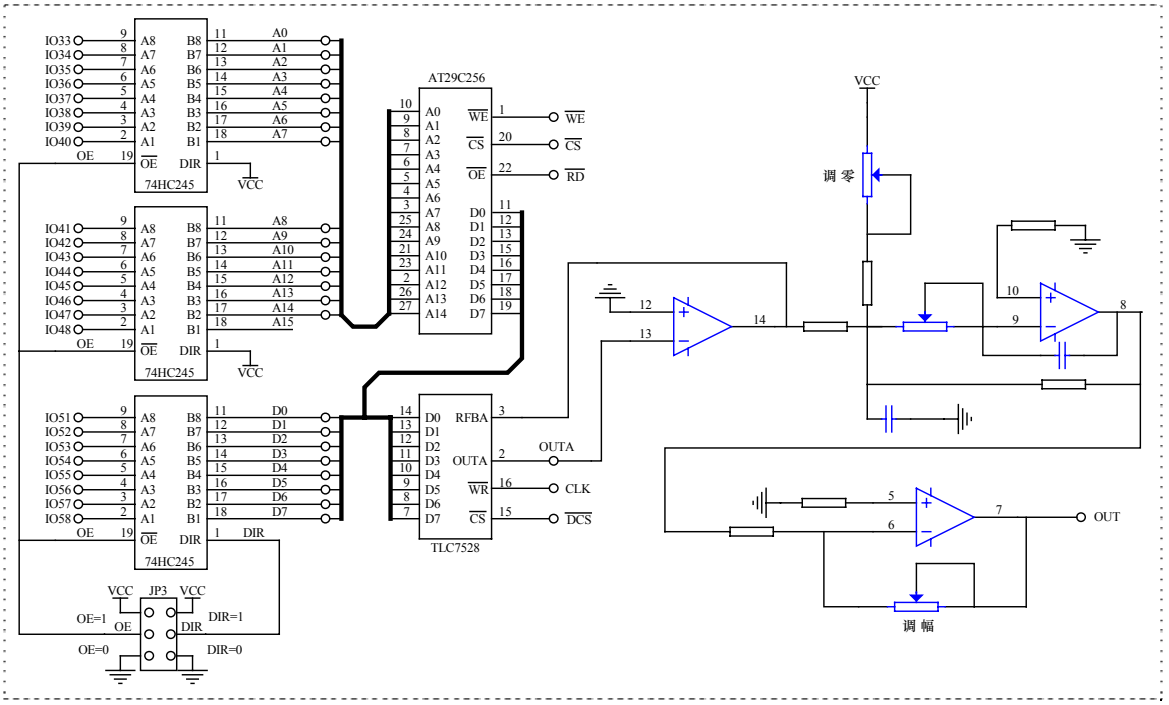


图 1-2-14 信号发生单元原理图

17. PSII 单元

PSII 单元提供 EDA 实验系统与计算机的 PC104 键盘或鼠标进行连接的接口，实现串行通信的协议转换。其原理如图 1-2-15 所示。如果使用需要将 JP4 短路块接到 3.3V 处。

注意：系统默认将 JP4 短路块接到 3.3V。

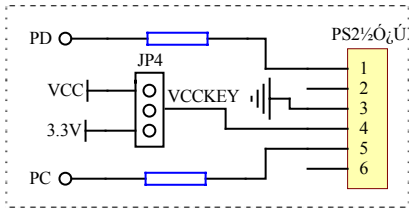


图 1-2-15 PSII 单元原理图

18. 扩展单元

该单元提供一块面包板和若干组转接排线及一个 40 引脚的编程座。便于用户进行二次开发，搭建一些简单的电路。

1.3 SOPC 开发板的使用方法

SOPC 开发板引脚分配如图 1-3-1、1-3-2、1-3-3 所示。

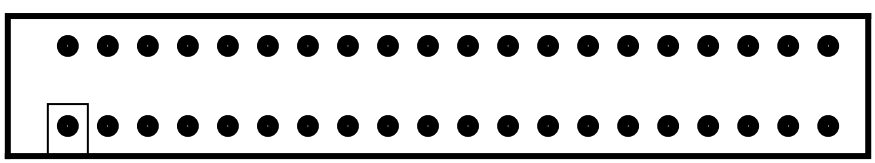
FPGA引脚	名称	序号	JP1接口	序号	名称	FPGA引脚
NC	VCC	1		2	GND	NC
2	IO1	3		4	IO2	3
4	IO3	5		6	IO4	5
6	IO5	7		8	IO6	7
8	IO7	9		10	IO8	11
12	IO9	11		12	IO10	13
14	IO11	13		14	IO12	15
16	IO13	15		16	IO14	17
18	IO15	17		18	IO16	19
20	IO17	19		20	IO18	21
23	IO19	21		22	IO20	41
28	CLK0	23		24	CLK1	29
42	IO21	25		26	IO22	43
44	IO23	27		28	IO24	45
46	IO25	29		30	IO26	47
48	IO27	31		32	IO28	49
50	IO29	33		34	IO30	53
54	IO31	35		36	IO32	55
56	IO33	37		38	IO34	57
NC	VCC	39		40	GND	NC

图 1-3-1 JP1 接口引脚分配图

FPGA引脚	名称	序号	JP2接口	序号	名称	FPGA引脚
NC	GND	1		2	VCC	NC
194	IO66	3		4	IO65	193
188	IO64	5		6	IO63	187
186	IO62	7		8	IO61	185
184	IO60	9		10	IO59	183
182	IO58	11		12	IO57	181
180	IO56	13		14	IO55	179
178	IO54	15		16	IO53	177
176	IO52	17		18	IO51	175
174	IO50	19		20	IO49	173
170	IO48	21		22	IO47	169
168	IO46	23		24	IO45	167
166	IO44	25		26	IO43	165
164	IO42	27		28	IO41	163
162	IO40	29		30	IO39	161
160	IO38	31		32	IO37	159
153	CLK(50M)	33		34	IO36	158
156	IO35	35		36	CLK3	152
131	RST	37		38	FLASH_RYBY	136
NC	GND	39		40	VCC	NC

图 1-3-2 JP2 接口引脚分配图

FPGA引脚	NC	240	144	39	237	235	233	227	226	224	222	218	216	214	208	206	202	200	196	NC
名称	VCC1.5	CLR	SCLK	PLL_N	IO95	IO93	IO91	IO89	IO88	IO86	IO84	IO82	IO80	IO78	IO76	IO74	IO72	IO70	IO68	GND
序号	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40



序号	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39
名称	VCC3.3	OE	INIT	PLL_P	IO96	IO94	IO92	IO90	IO87	IO85	IO83	IO81	IO79	IO77	IO75	IO73	IO71	IO69	IO67	VCC
FPGA引脚	NC	239	1	38	238	236	234	228	225	223	219	217	215	213	207	203	201	197	195	NC

图 1-3-3 扩展接口引脚分配图

SOPC 开发板与 TD-EDA 基本实验平台丝印的对应关系如表 1-3-1 所示。

表 1-3-1 SOPC 开发板引脚对应关系

FPGA 引脚	功能	备注	FPGA 引脚	功能	备注
1	**INIT_DONE**		53	IO30	
2	IO1		54	IO31	
3	IO2		55	IO32	
4	IO3		56	IO33	
5	IO4		57	IO34	
6	IO5		58	DA24	
7	IO6		59	DA25	
8	IO7		60	DA26	
11	IO8		61	DA27	
12	IO9		62	DA28	
13	IO10		63	DA29	
14	IO11		64	DA30	
15	IO12		65	DA31	
16	IO13		66	DA16	
17	IO14		67	DA17	
18	IO15		68	DA18	
19	IO16		73	DA19	
20	IO17		74	DA20	
21	IO18		75	DA21	
23	IO19		76	DA22	
28	**CLK0**		77	DA23	
29	**CLK1**		78	DA8	
38	PLL1_OUTP		79	DA9	
39	PLL1_OUTN		82	DA10	
41	IO20		83	DA11	
42	IO21		84	DA12	
43	IO22		85	DA13	
44	IO23		86	DA14	
45	IO24		87	DA15	
46	IO25		88	EA11	
47	IO26		93	EA10	
48	IO27		94	EA9	
49	IO28		95	EA8	
50	IO29		98	EA7	

续表 1-3-1 SOPC 开发板引脚对应关系

FPGA 引脚	功能	备注	FPGA 引脚	功能	备注
99	**SRAM_WE**		144	SDCLK2	
100	DA7		152	CLK3	
101	DA6		**153**	**CLK(50M)**	
104	DA5		156	IO35	
105	DA4		158	IO36	
106	EA12		159	IO37	
107	EA13		160	IO38	
108	EA14		161	IO39	
113	EA16		162	IO40	
114	**SRAM_BE2**		163	IO41	
115	**SRAM_BE3**		164	IO42	
116	**SRAM_BE0**		165	IO43	
117	**SRAM_BE1**		166	IO44	
118	**SRAM_OE**		167	IO45	
119	DA3		168	IO46	
120	DA2		169	IO47	
121	DA1		170	IO48	
122	DA0		173	IO49	
123	**SRAM_CE#**		174	IO50	
124	EA6		175	IO51	IN1
125	EA5		176	IO52	IN2
126	EA4		177	IO53	IN3
127	EA3		178	IO54	IN4
128	EA2		179	IO55	IN5
131	**RST**		180	IO56	IN6
132	EA17		181	IO57	IN7
133	EA18		182	IO58	IN8
134	EA19		183	IO59	IN9
135	EA20		184	IO60	IN10
136	**FLASH_RYBY**		185	IO61	IN11
137	**FLASH_WE**		186	IO62	IN12
138	**FLASH_CS**		187	IO63	IN13
139	**FLASH_OE**		188	IO64	IN14
140	**FLASH_BYTE**		193	IO65	IN15
141	EA1		194	IO66	IN16
143	EA15		195	IO67	

续表 1-3-1 SOPC 开发板引脚对应关系

FPGA 引脚	功能	备注	FPGA 引脚	功能	备注
196	IO68		222	IO84	
197	IO69		223	IO85	
200	IO70		224	IO86	
201	IO71		225	IO87	
202	IO72		226	IO88	
203	IO73		227	IO89	
206	IO74		228	IO90	
207	IO75		233	IO91	
208	IO76		234	IO92	
213	IO77		235	IO93	
214	IO78		236	IO94	
215	IO79		237	IO95	
216	IO80		238	IO96	
217	IO81		**239**	**OE**	
218	IO82		**240**	**CLR**	
219	IO83				

SOPC 开发板将+5V 电压变换为 3.3V、1.5V 电压，为 FPGA 提供工作电压。SOPC 开发板提供了一个 50M 晶振与 Cyclone 系列 EP1C6Q240 芯片的 153 脚(CLK2)连接作为时钟信号，。为了进行 SOPC 的开发 SOPC 开发板上提供了两片 SRAM 和一片 FLASH 存储器。

SOPC 开发板提供了两片高速异步 SRAM: ISSI IS61LV25616AL-10T，每片的容量为 256K×16Bit，通过并联实现 32Bit 数据宽度。两片 SRAM 的地址线和控制线连接到一起，字节使能信号独立，可以对每一个 Byte 进行操作。SRAM 存储器的引脚如图 1-3-4 所示，SRAM 地址总线的分配如表 1-3-2 所示，SRAM 数据总线的分配如表 1-3-3 所示，SRAM 控制信号的分配如表 1-3-4 所示

表 1-3-2 SRAM 地址总线信号

SRAM 引脚	FPGA 引脚	总线名称	SRAM 引脚	FPGA 引脚	总线名称
A0	PIN_128	SHA_A [2]	A9	PIN_88	SHA_A [11]
A1	PIN_127	SHA_A [3]	A10	PIN_106	SHA_A [12]
A2	PIN_126	SHA_A [4]	A11	PIN_107	SHA_A [13]
A3	PIN_125	SHA_A [5]	A12	PIN_108	SHA_A [14]
A4	PIN_124	SHA_A [6]	A13	PIN_143	SHA_A [15]
A5	PIN_98	SHA_A [7]	A14	PIN_113	SHA_A [16]
A6	PIN_95	SHA_A [8]	A15	PIN_132	SHA_A [17]
A7	PIN_94	SHA_A [9]	A16	PIN_133	SHA_A [18]
A8	PIN_93	SHA_A [10]	A17	PIN_134	SHA_A [19]

表 1-3-3 SRAM 数据总线信号

SRAM(U2)引脚	FPGA 引脚	总线名称	SRAM(U3)引脚	FPGA 引脚	总线名称
D0	PIN_122	SHA_D[0]	D0	PIN_66	SHA _D[16]
D1	PIN_121	SHA_D[1]	D1	PIN_67	SHA _D[17]
D2	PIN_120	SHA_D[2]	D2	PIN_68	SHA _D[18]
D3	PIN_119	SHA_D[3]	D3	PIN_73	SHA _D[19]
D4	PIN_105	SHA_D[4]	D4	PIN_74	SHA _D[20]
D5	PIN_104	SHA_D[5]	D5	PIN_75	SHA _D[21]
D6	PIN_101	SHA_D[6]	D6	PIN_76	SHA _D[22]
D7	PIN_100	SHA_D[7]	D7	PIN_77	SHA _D[23]
D8	PIN_78	SHA_D[8]	D8	PIN_58	SHA _D[24]
D9	PIN_79	SHA_D[9]	D9	PIN_59	SHA _D[25]
D10	PIN_82	SHA_D[10]	D10	PIN_60	SHA _D[26]
D11	PIN_83	SHA_D[11]	D11	PIN_61	SHA _D[27]
D12	PIN_84	SHA_D[12]	D12	PIN_62	SHA _D[28]
D13	PIN_85	SHA_D[13]	D13	PIN_63	SHA _D[29]
D14	PIN_86	SHA_D [14]	D14	PIN_64	SHA _D[30]
D15	PIN_87	SHA_D[15]	D15	PIN_65	SHA _D[31]

表 1-3-4 SRAM 控制信号

SRAM 引脚	FPGA 引脚	SRAM 引脚	FPGA 引脚
CS＃	PIN_123	BE0	PIN_116
WE＃	PIN_99	BE1	PIN_117
OE＃	PIN_118	BE2	PIN_114
		BE3	PIN_115

SOPC 开发板提供了一片大容量快速 Flash: AM29LV160DT-90EC，存储容量为 1M×16Bit。Flash 存储器的引脚如图 1-3-5 所示，FLASH 数据总线的分配如表 1-3-5 所示，FLASH 地址总线的分配如表 1-3-6 所示，FLASH 控制信号的分配如表 1-3-7 所示

表 1-3-5 FLASH 数据总线信号

FLASH 引脚	FPGA 引脚	总线名称	FLASH 引脚	FPGA 引脚	总线名称
DQ0	PIN_122	SHA _D[0]	DQ8	PIN_78	SHA _D[8]
DQ1	PIN_121	SHA _D[1]	DQ9	PIN_79	SHA _D[9]
DQ2	PIN_120	SHA _D[2]	DQ10	PIN_82	SHA _D[10]
DQ3	PIN_119	SHA _D[3]	DQ11	PIN_83	SHA _D[11]
DQ4	PIN_105	SHA _D[4]	DQ12	PIN_84	SHA _D[12]
DQ5	PIN_104	SHA _D[5]	DQ13	PIN_85	SHA _D[13]
DQ6	PIN_101	SHA _D[6]	DQ14	PIN_86	SHA _D[14]
DQ7	PIN_100	SHA _D[7]	DQ15/A-1	PIN_87	SHA _D[15]

表 1-3-6 FLASH 地址总线信号

FLASH 引脚	FPGA 引脚	总线名称	FLASH 引脚	FPGA 引脚	总线名称
A0	PIN_141	SHA_A[1]	A10	PIN_88	SHA _A[11]
A1	PIN_128	SHA _A[2]	A11	PIN_106	SHA _A[12]
A2	PIN_127	SHA _A[3]	A12	PIN_107	SHA _A[13]
A3	PIN_126	SHA _A[4]	A13	PIN_108	SHA _A[14]
A4	PIN_125	SHA _A[5]	A14	PIN_143	SHA _A[15]
A5	PIN_124	SHA _A[6]	A15	PIN_113	SHA _A[16]
A6	PIN_98	SHA _A[7]	A16	PIN_132	SHA _A[17]
A7	PIN_95	SHA _A[8]	A17	PIN_133	SHA _A[18]
A8	PIN_94	SHA _A[9]	A18	PIN_134	SHA _A[19]
A9	PIN_93	SHA _A[10]	A19	PIN_135	SHA _A[20]

表 1-3-7 FLASH 控制信号

FLASH 引脚	FPGA 引脚	FLASH 引脚	FPGA 引脚
CS＃	PIN_138	BYTE#	PIN_140
WE＃	PIN_137	RY/BY#	PIN_136
OE＃	PIN_139	RST	PIN_131

说明：板子上配置了 16Bit 的 Flash，可以通过配置，设置为 8Bit 操作。BYTE＃为低电平时，为 8Bit 模式，DQ15 变成了 A-1。

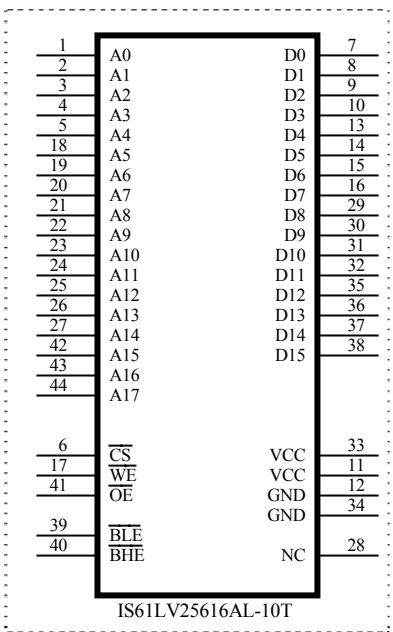


图 1-3-4 SRAM 存储器引脚图

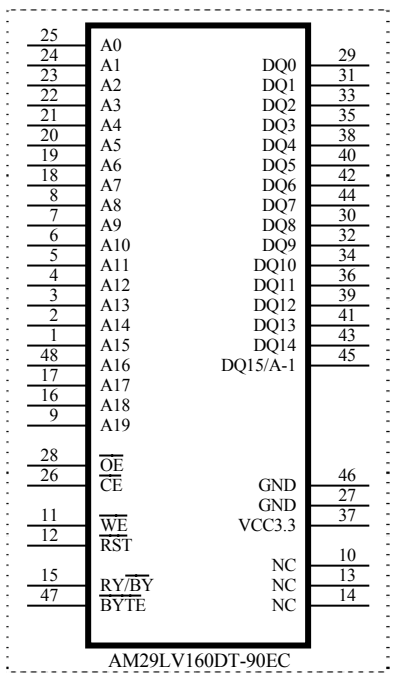


图 1-3-5 Flash 存储器引脚图

第二章 Quartus II 开发软件简介

Altera 公司的 EDA 开发工具有 Max＋Plus II 和 Quartus II 两种。

Max＋Plus II 界面友好使用便捷，被誉为业界最易学易用的 EDA 软件。它提供了与结构无关的设计环境，支持 MAX 系列、FLEX 系列及 ACEX1K 系列等 CPLD/FPGA 器件。可以使用原理图输入和硬件描述语言输入等多种描述方式。

随着技术的发展，用户对开发工具的要求越来越高，Altera 公司适时推出了新的开发工具 Quartus II 软件。Quartus II 集成环境可以实现系统级设计、嵌入式系统的软件开发、可编程逻辑器件(PLD)设计、综合，布局和布线、验证和仿真。

Quartus II 集成环境除支持 MAX 系列、FLEX 系列、ACEX1K 系列器件外，还支持 Altera 公司的 APEX20K 系列、Stratix 系列、Cyclone 系列和 MAX II 系列等新型 CPLD/FPGA 器件。

Quartus II 软件提供原理图输入和硬件描述语言输入等多种描述方式。使用它进行基于 CPLD/FPGA 的数字系统开发的过程与 Max＋Plus II 软件相似，只是在支持的器件种类、编译选项、优化选项、IP 模块、适配速度及准确性方面得到了极大的提高。

Quartus II 集成环境在保持 Max+Plus II 软件特点的基础上，提供了可编程片上系统(SOPC--System On a Programmable Chip)设计的一个综合开发环境，是进行 SOPC 设计的基础，从而形成了一个完整的可编程、可重构的 SOPC 设计环境。

Quartus II 设计软件根据设计者的需要提供了一个完整的多平台开发环境，它包含整个 FPGA 和 CPLD 设计阶段的解决方案。Quartus II 软件的开发流程如图 2-1 所示。

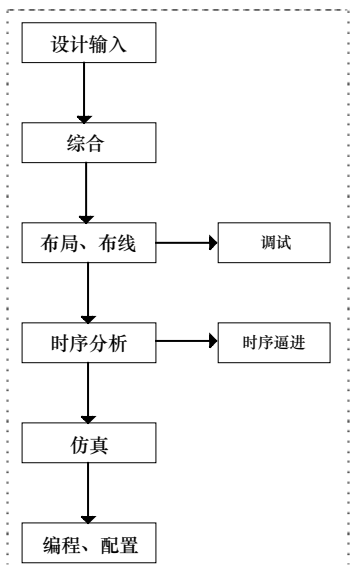


图 2-1 QuartusII 软件的开发流程

2.1 Quartus II 软件的安装

TD-EDA 实验系统使用 Quartus II 集成开发环境 Quartus II 4.2 版本，为了使 Quartus II 软件的性能达到最佳，建议计算机的最低配置如下：

(1) 奔腾 II 400MHz、512MB 以上系统内存；

(2) 大于 1G 的安装 Quartus II 软件所需的最小硬盘空间；

(3) Microsoft Windows NT4.0(Service Pack 4 以上)、Windows2000 或 Windows XP 操作系统。

注意：Quartus II 软件不支持 Windows98 操作系统。

(4) 用于 ByteBlaster II 或 ByteBlaster MV 下载电缆的并行口(LPT 口)；

(5) Microsoft IE5.0 以上浏览器；

(6) TCP/IP 网络协议；

(7) 网卡一块。

注意：Quartus II 软件必须在安装有网卡的 PC 上使用。

用户在确保计算机满足上述配置后就可以安装 Quartus II 软件，下面简单介绍 Quartus II 4.2 版软件的安装过程：

1. 将 Quartus II 设计软件的光盘放入计算机的光驱，从资源管理器进入光盘驱动器，双击 Quartus II 目录下的 install.exe 文件，出现如图 2-1-1 所示的 Quartus II 安装界面。

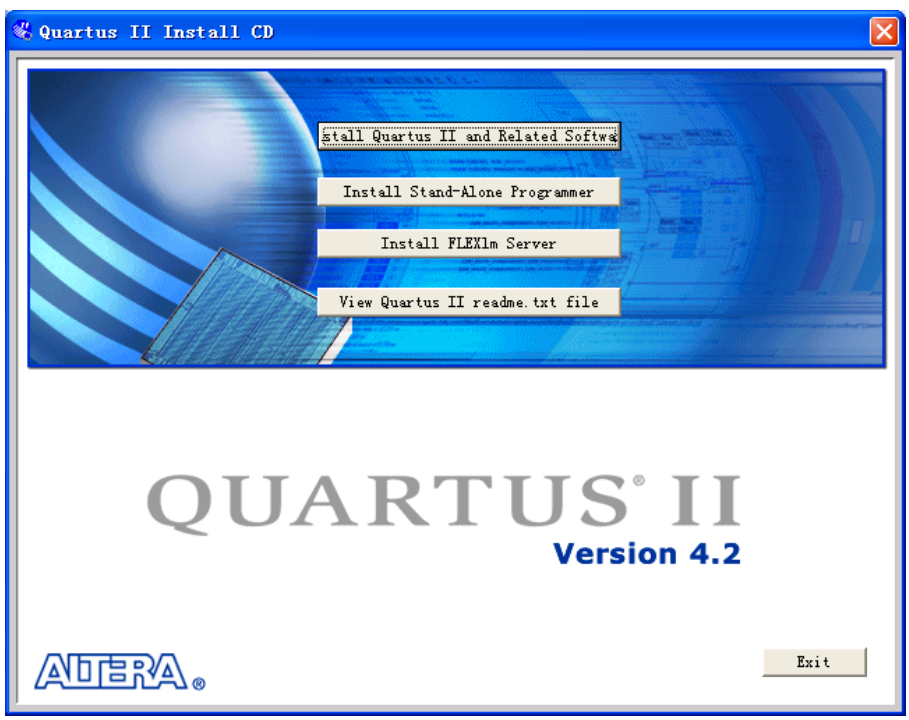


图 2-1-1 Quartus II 安装界面

2. 点击”Install Quartus II and Related Software”按钮进入如图 2-1-2 所示的安装 Quartus II 软件的安装向导界面。在这个安装向导界面中，可以选择安装”ModelSim-Altera”、 ”Nios II Embedded Processor,Evaluation Edition”等软件，在此我们选择安装”Quartus II 4.2”软件。

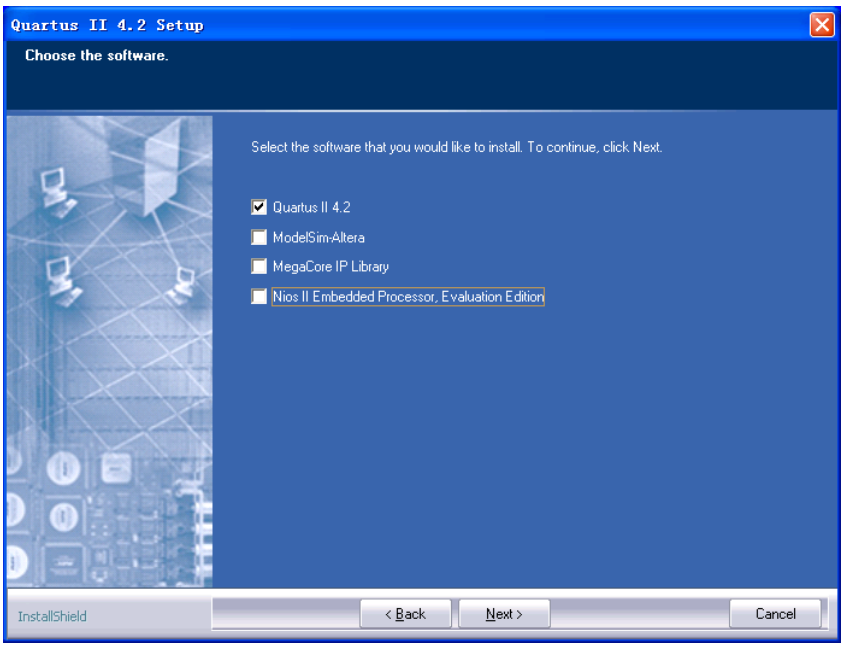


图 2-1-2 Quartus II 软件的安装向导界面

3. 按照安装向导的提示操作，经过一系列确认后，出现如图 2-1-3 所示的安装路径选择界面，及工程路径选择界面用户可以指定软件安装的路径，建议使用 Quartus II 默认的路径。

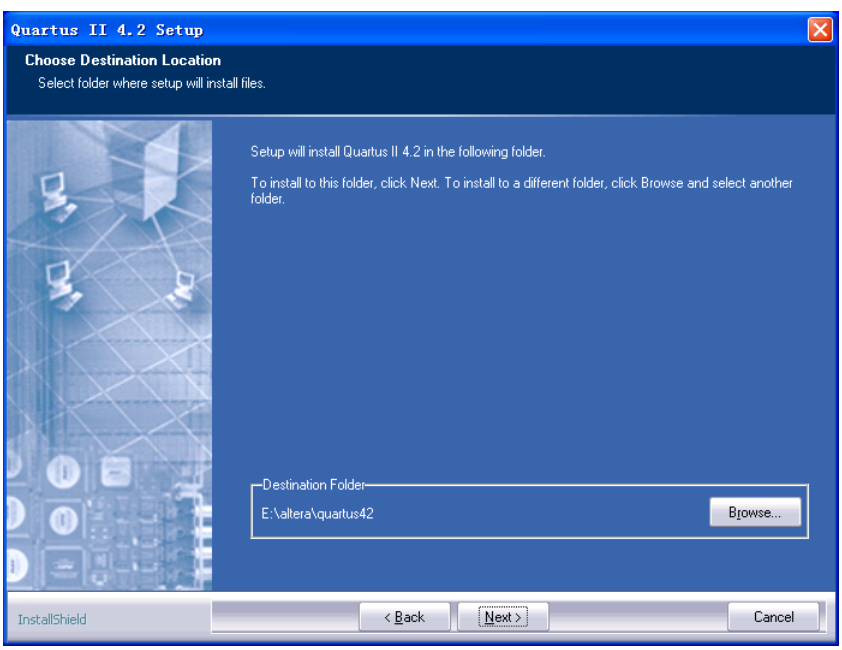


图 2-1-3 安装路径选择界面

4. 随后出现如图 2-1-4 所示的安装类型选择界面，用户可以选择“完全安装模式”或“用户定义安装模式”。我们选择“完全安装模式”。

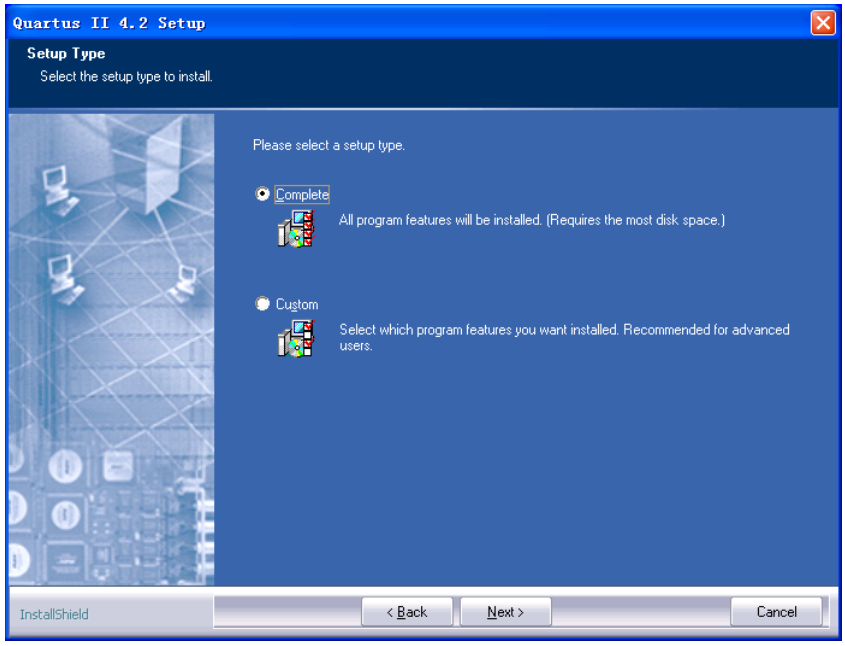


图 2-1-4　安装类型选择界面

5. 按照安装向导的提示操作，经过一系列确认后，出现如图 2-1-5 所示的 Quartus II 软件安装进程界面。

图 2-1-5　QuartusII 软件安装进程界面

6. 当 Quartus II 软件安装完成后，将出现提示界面，并显示安装成功与否的信息。至此已经顺利的完成了 QuartusII 4.2 版软件的安装。

7. 安装完 Quartus II 4.2 版软件之后，还要安装 Service Pack1 补丁，双击软件安装目录中的 QuartusII4.2_ServicePack1.EXE 文件，出现如图 2-1-6 所示的安装界面。

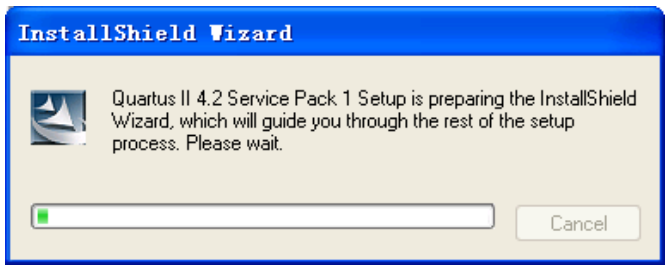


图 2-1-6 QuartusII 4.2 版 ServicePack1 的安装界面

8. 按照系统默认的路径一直点击”Next”按钮，出现如图 2-1-7 所示的安装进度界面。直到安装完成。

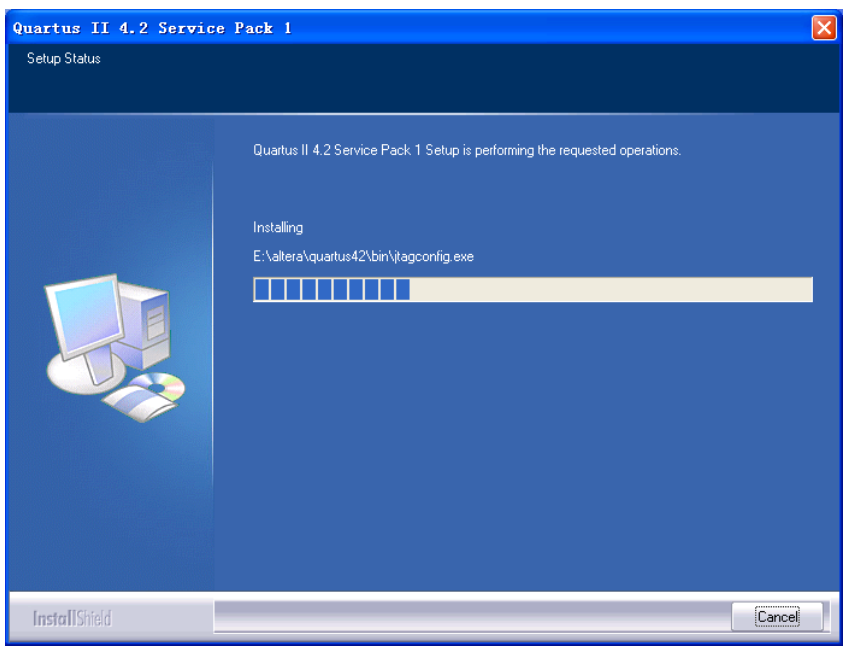


图 2-1-7 QuartusII 4.2 版 ServicePack1 的安装进度界面

9. 至此 Quartus II 4.2 版软件已经安装到用户的计算机上。但是要正常使用 Quartus II 软件还需进行一些设置。

10. 在首次运行之前，必须要有 Altera 公司提供的授权文件(License.Dat)，将授权文件(License.Dat)拷贝到 Quartus II 的安装目录下，并且用安装光盘中的 sys_cpt.dll 文件替换 Quartus II

软件安装目录下：\altera\quartus42\bin 文件夹下的源文件。

11. 点击“开始”→“程序”→“附件”→“命令提示符”，输入 IPCONFIG/ALL 命令出现如图 2-1-8 所示的命令提示符界面。在此界面中显示了网卡的物理地址，用“记事本”程序打开 License.Dat，将此文件中”HOSTID=**********”语句的*用网卡的物理地址替代。

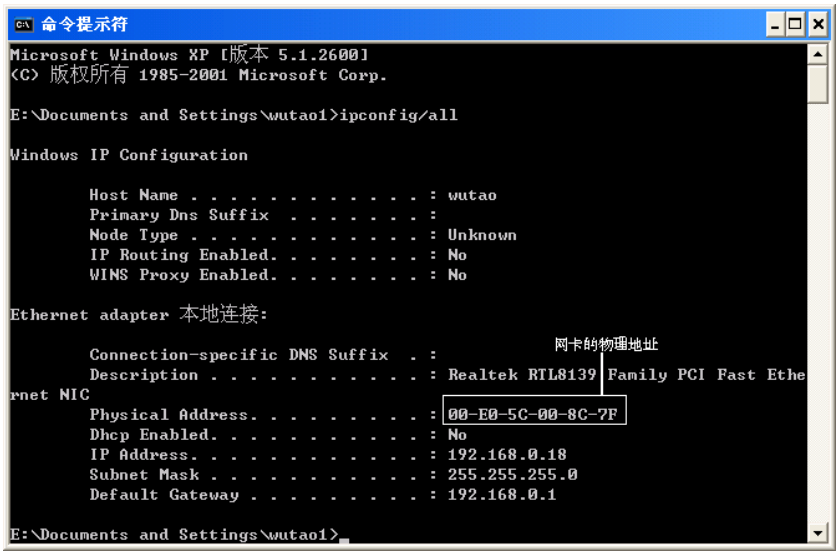


图 2-1-8 命令提示符界面

12. 启动 Quartus II 软件，出现如图 2-1-9 所示的授权文件指定界面，选择”Specify valid license file”选项。

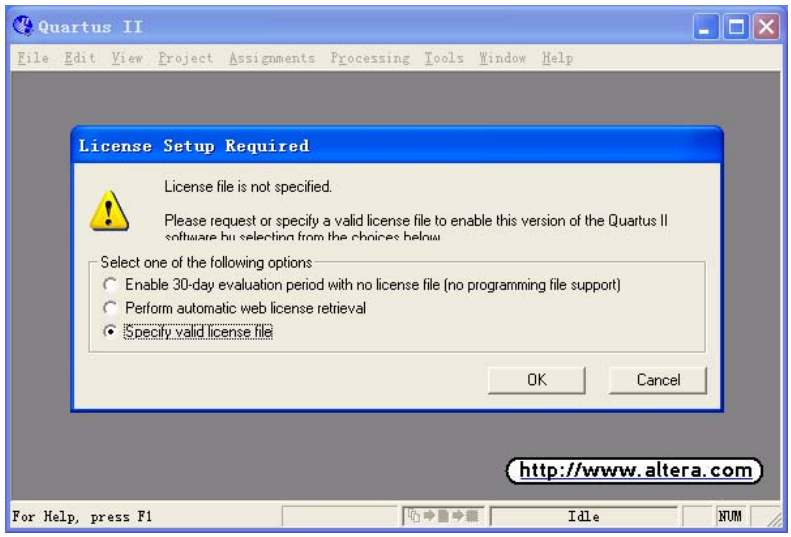


图 2-1-9 授权文件指定界面

13. 在如图 2-1-10 所示的设置 License 界面中，点击”License file”地址栏中指定 License.Dat 文件所在的路径，点击”OK”按钮就完成了 License.Dat 文件的设置。至此 Quartus II 软件就可以正常使用了。

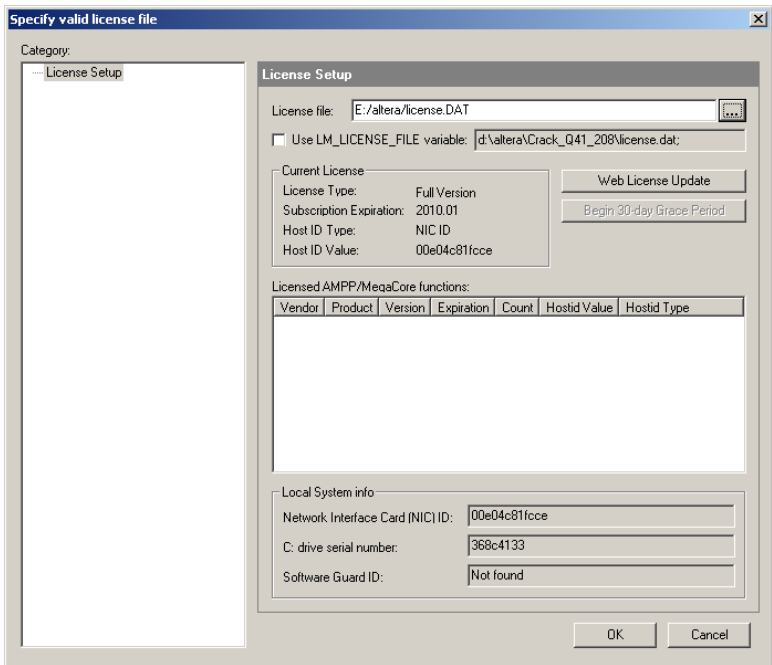


图 2-1-10 Options 对话框的 License Setup 界面

2.2 ByteBlaster II 下载电缆驱动程序的安装

使用 QuartusII 软件可以对 Altera 器件进行编程或配置，QuartusII 编译器的 Assembler 模块自动将适配过程的器件、逻辑单元和引脚分配信息转换为器件的编程图象，并将这些图象以目标器件的编程器对象文件(.POF)或 SRAM 对象文件(.SOF)的形式保存为编程文件，QuartusII 软件的编程器(Programmer)使用该文件对器件进行编程配置。

TD-EDA 实验系统的 CPLD/FPGA 开发板提供了 Altera 公司的 ByteBlaster II 下载电缆，为了使用此电缆对器件进行编程配置，需要手动安装 Altera 编程器硬件驱动程序。这里我们分别介绍在 Windows2000 操作系统及 Windows XP 操作系统中安装驱动程序的步骤。

2.2.1 Windows2000 操作系统中安装驱动的步骤

1. 打开控制面板（“开始”→ “设置”→ “控制面板”），双击“添加/删除硬件”图标，启动添加/删除硬件向导，点击“下一步”；

2. 在如图 2-2-1 所示的添加/排除设备故障界面中选择“添加/排除设备故障”，点击“下一步”。Windows2000 将在新的硬件检测窗口里搜索新的即插即用设备；

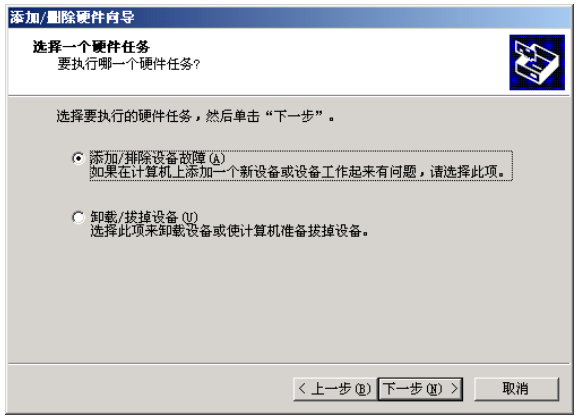


图 2-2-1 添加/排除设备故障界面

3. 在如图 2-2-2 所示的选择一个硬件设备界面中，选择“添加新设备”，点击“下一步”；

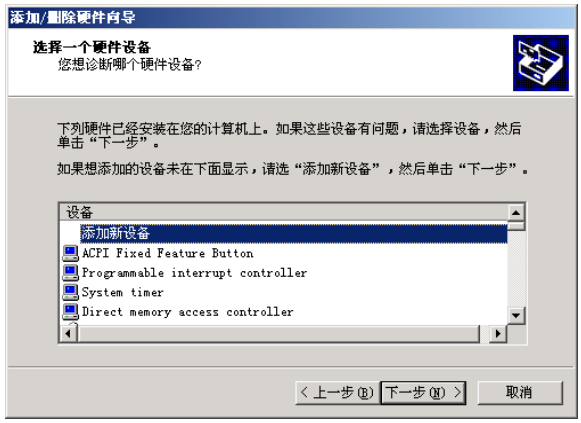


图 2-2-2 选择一个硬件设备界面

4. 在如图 2-2-3 所示查找新硬件界面中，选择“否，我想从列表选择硬件”，点击“下一步”；

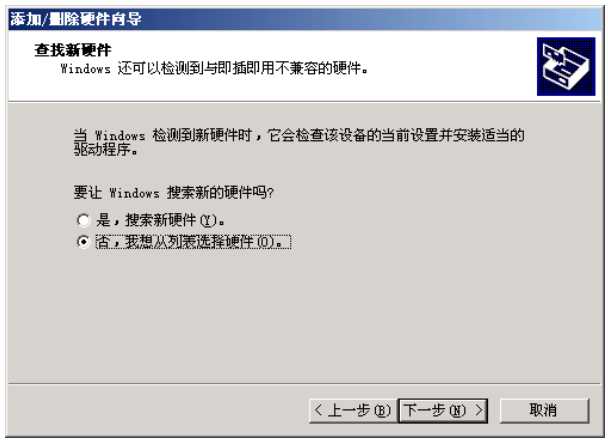


图 2-2-3　查找新硬件界面

5. 在如图 2-2-4 所示的硬件类型界面中，选择“声音、视频和游戏控制器”，点击“下一步”；

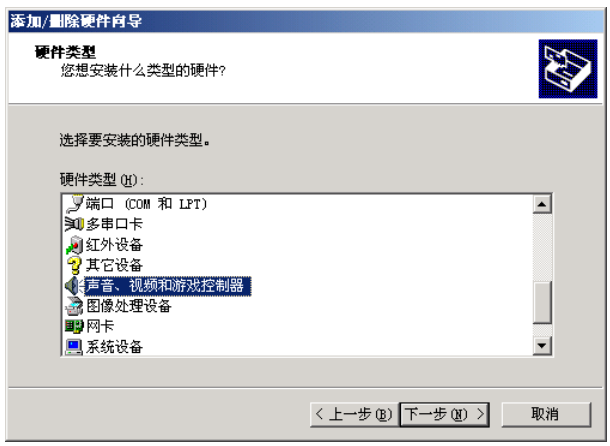


图 2-2-4　硬件类型界面

6. 在如图 2-2-5 所示的选择一个设备驱动程序界面中，点击“从磁盘安装”按钮；

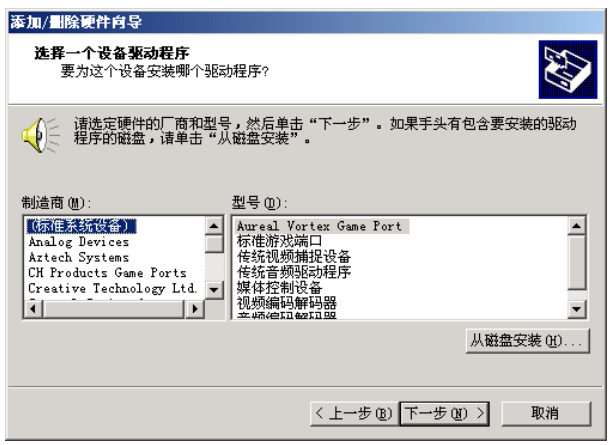


图 2-2-5　选择一个设备驱动程序界面

7. 在如图 2-2-6 所示的指定文件路径界面中，指定 win2000.inf 文件的完整路径(如 Quartus II 安装目录>\drivers\win2000)，点击“确定”按钮，在图 2-3-5 中点击“下一步”；

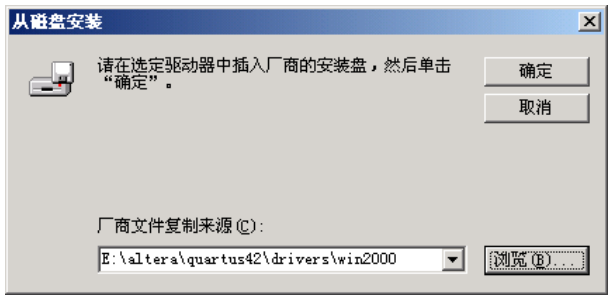


图 2-2-6 指定文件路径界面

8. 在如图 2-2-7 所示的选择一个设备驱动程序界面中，选择”Altera ByteBlaster”安装 ByteBlaster II 下载电缆的驱动程序，点击“下一步”；

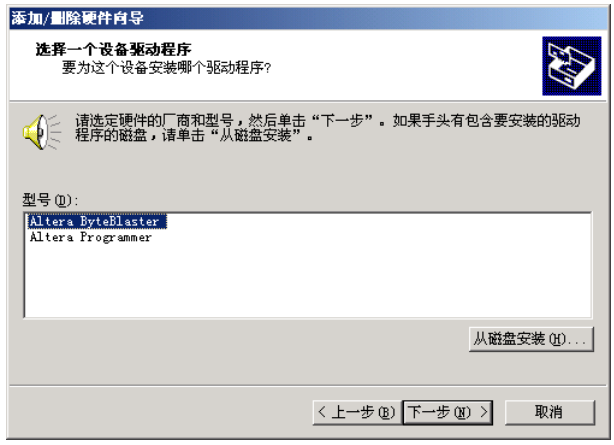


图 2-2-7 选择一个设备驱动程序界面

9. 在“开始硬件安装”窗口，点击“下一步”；在如图 2-2-8 所示的完成添加/删除硬件向导界面中，点击“完成”按钮 ；

图 2-2-8 完成添加/删除硬件向导界面

10. 在弹出的系统对话框中选择重新启动计算机。计算机在重新启动后驱动程序安装完成，用户可以在如图 2-2-9 所示的设备管理器界面中找到安装的驱动程序。

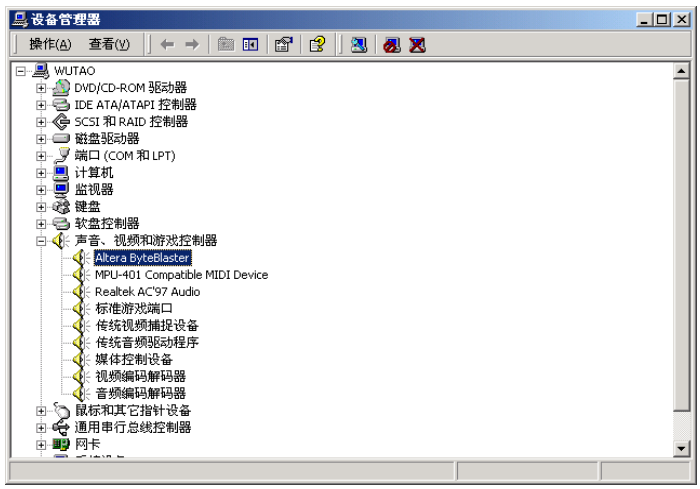


图 2-2-9 设备管理器界面

2.2.2 Windows XP 操作系统中安装驱动的步骤

1. 打开控制面板(“开始”→ “控制面板”)，双击“添加硬件”图标，启动添加硬件向导，点击“下一步”；

2. 在如图 2-2-10 所示的硬件是否已连接界面中选择“是，硬件已连接好”，点击“下一步”；

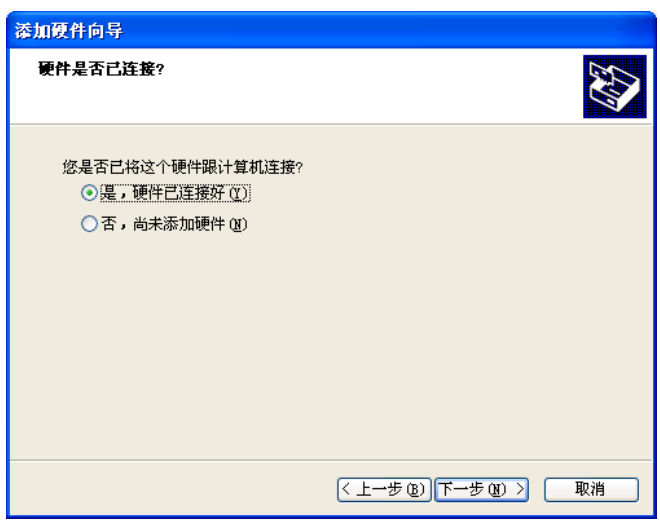


图 2-2-10 硬件是否已连接界面

3. 在如图 2-3-11 所示的选择一个硬件设备界面中，选择“添加新的硬件设备”，点击“下一步”；

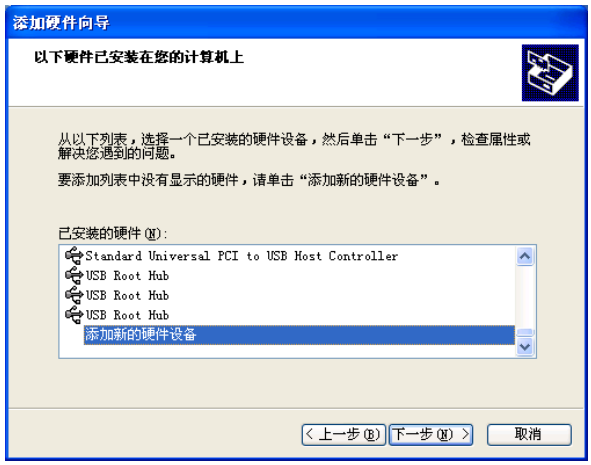


图 2-2-11　选择一个硬件设备界面

4. 在如图 2-2-12 所示的查找新硬件界面中，选择“安装我手动从列表选择的硬件(高级)”，点击“下一步”；

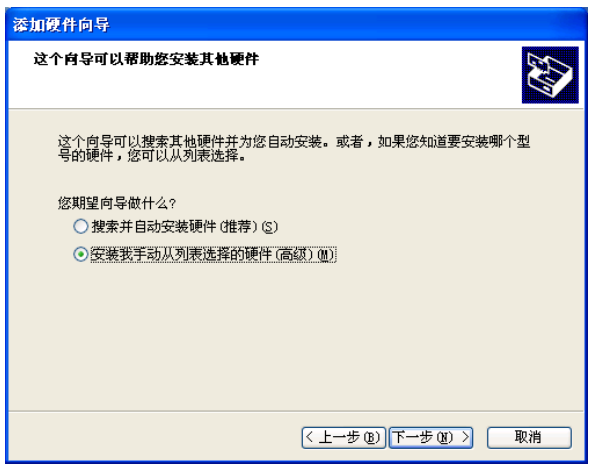


图 2-2-12　查找新硬件界面

图 2-2-13　硬件类型界面

5. 在如图 2-2-13 所示的硬件类型界面中，选择“声音、视频和游戏控制器”，点击“下一步”；

6. 在如图 2-2-14 所示的选择一个设备驱动程序界面中，点击“从磁盘安装”按钮；

图 2-2-14　选择一个设备驱动程序界面

7. 在如图 2-2-15 所示的指定文件路径界面中，指定 win2000.inf 文件的完整路径(如 Quartus II 安装目录>\drivers\win2000)，点击“确定”按钮，在图 2-2-14 中点击“下一步”；

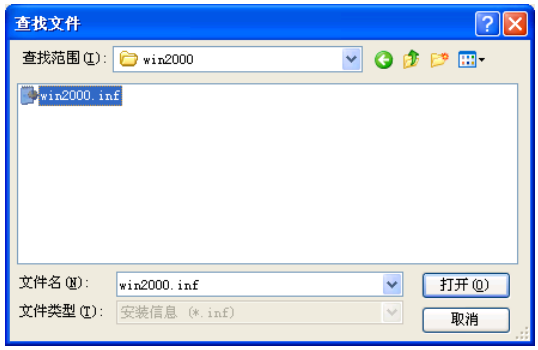


图 2-2-15　指定文件路径界面

图 2-2-16　选择一个设备驱动程序界面

8. 在如图 2-2-16 所示的选择一个设备驱动程序界面中，选择”Altera ByteBlaster”安装 ByteBlaster II 下载电缆的驱动程序，点击“下一步”；

9. 点击“下一步”出现“完成添加/删除硬件向导”界面，点击“完成”按钮。计算机在重新启动后驱动程序有效，用户可以在如图 2-2-17 所示的设备管理器界面中找到安装的驱动程序。

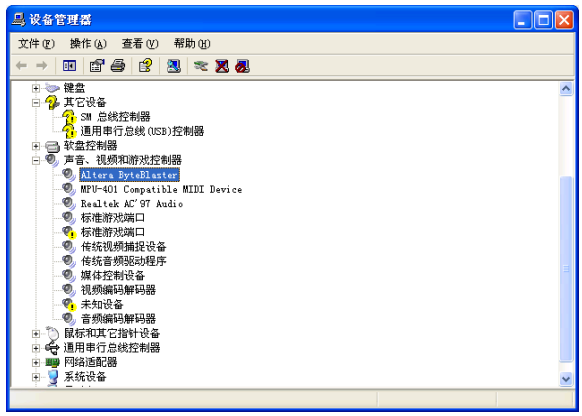


图 2-2-17 设备管理器界面

2.2.3 Quartus II 软件中设置下载电缆

安装完驱动程序后，在 Quartus II 软件中进行设置，以选择 ByteBlaster II 下载电缆对器件进行配置，具体步骤如下：

1. 打开 QuartusII 软件，选择 Tools→Programmer 命令，如图 2-2-18 所示的编程器窗口自动打开一个*.CDF 的新链式描述文件，点击编程硬件设置”Hardware Setup”按钮；

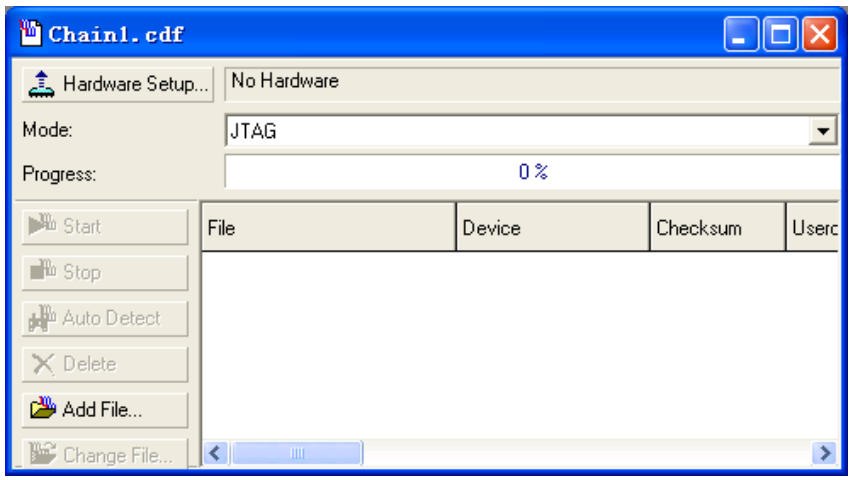


图 2-2-18 编程器窗口

2. 弹出如图 2-2-19 所示硬件设置对话框，点击”Add Hardware”按钮，出现如图 2-2-20 所示的添加硬件对话框。

3. 在如图 2-2-20 所示的添加硬件对话框中，从 Hardware type 列表中选择 ByteBlasterMV or ByteBlashter II，从 Port 列表中选择 LPT1，点击”OK”按钮返回 Hardware Setup 对话框，此时硬件设置对话框如图 2-2-21 所示。

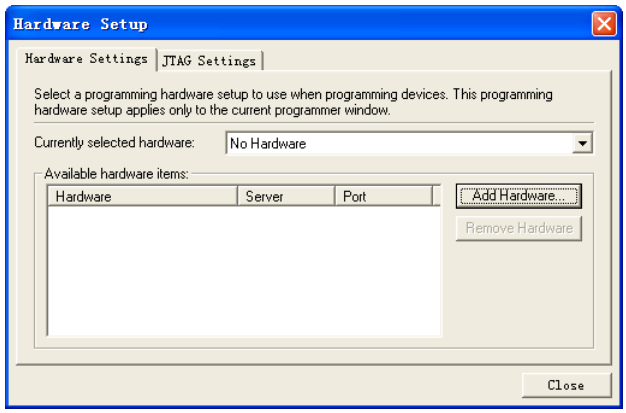


图 2-2-19 硬件设置对话框

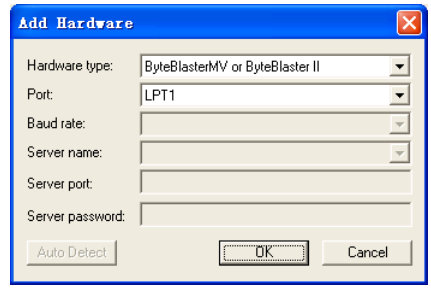


图 2-2-20 添加硬件对话框

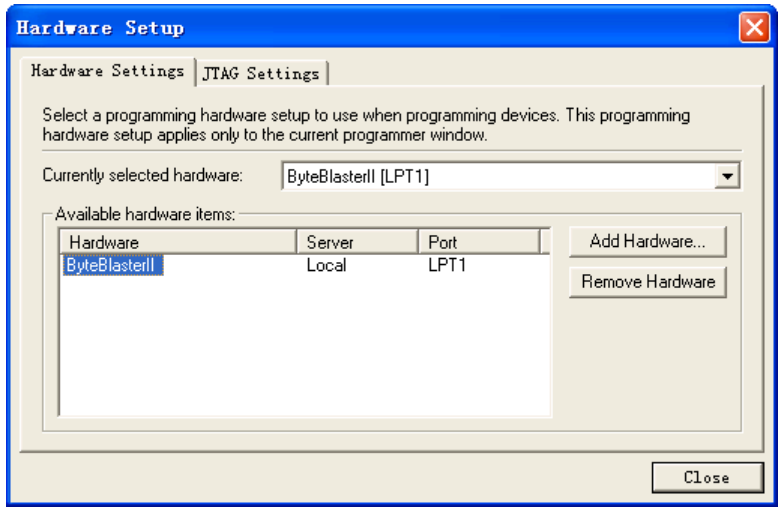


图 2-2-21 硬件设置对话框

4. 在图 2-2-21 所示硬件设置对话框中双击 ByteBlaster II，点击”Close”按钮。返回如图 2-2-22 所示设置完成的编程器窗口。

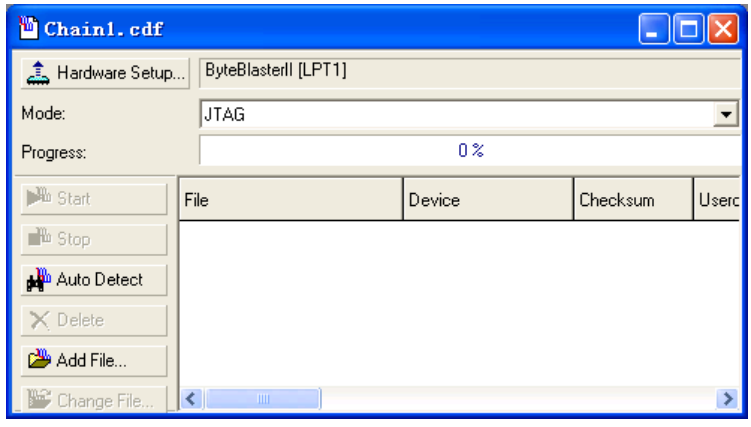


图 2-2-22 设置完成的编程器窗口

2.3 NiosII 软件的安装

SOPC 开发板在提供数字系统设计实验基础上还可以完成基于 Nios II 处理器的 SOPC 开发实验，在 SOPC 开发前需要安装 Nios II 嵌入式系统的软件。下面简单介绍 Nios II1.1 版软件的安装过程：

1. 将 Nios II 软件的光盘放入计算机的光驱，从资源管理器进入光盘驱动器，双击 launcher.exe 文件，出现如图 2-3-1 所示的 Nios II 安装界面。

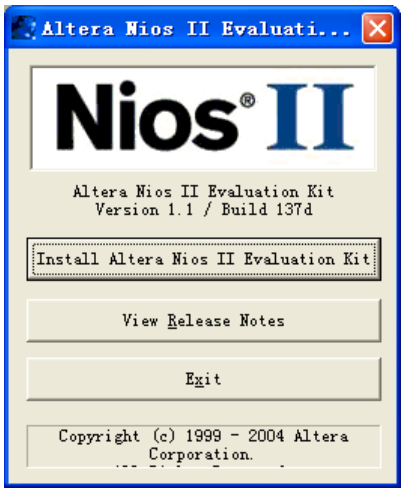


图 2-3-1 Nios II 软件安装界面

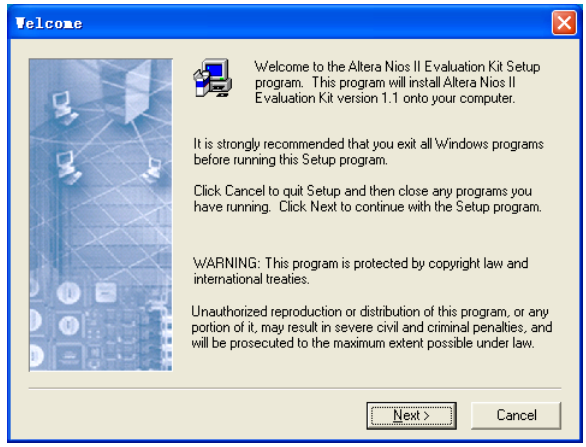


图 2-3-2 Nios II 软件的安装向导界面

2. 点击”Install Altera NiosII Evaluation Kit”按钮进入如图 2-3-2 所示的安装 Nios II 软件的安装向导界面。

3. 按照安装向导的提示操作，经过一系列确认后，出现如图 2-3-3 所示的安装路径选择界面,建议使用 NiosII 默认的路径。

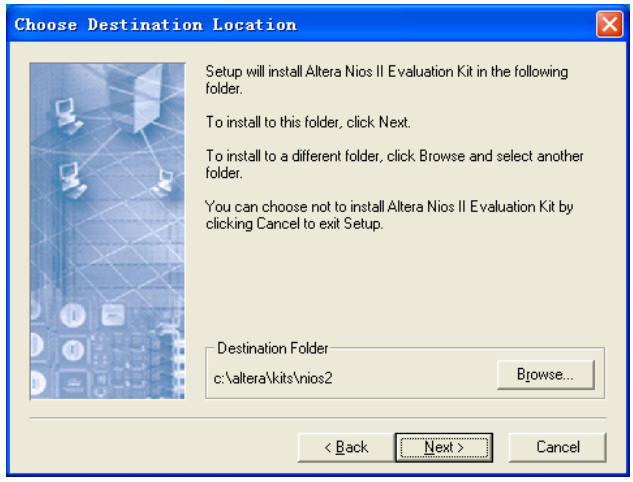


图 2-3-3 安装路径选择界面

5. 按照安装向导的提示操作，经过一系列确认后，出现如图 2-3-4 所示的 NiosII 软件安装进程界面。

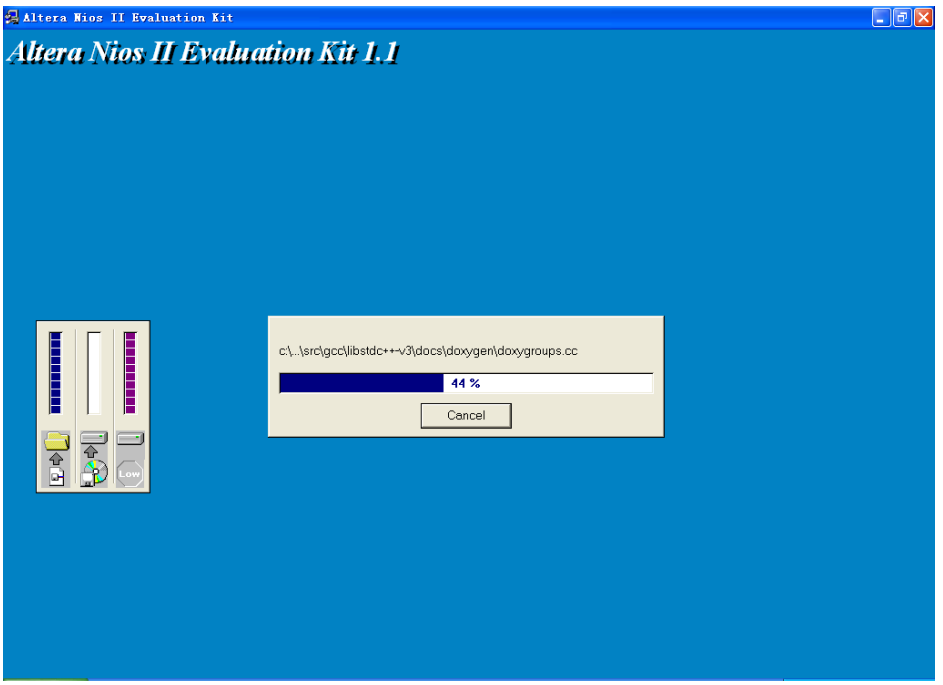


图 2-3-4 Nios II 软件安装进程界面

6. 当 NiosII 软件安装完成后，将出现提示界面，显示安装成功的信息。至此已经顺利的完成了 NiosII 软件的安装。

7. 安装完 NiosII 软件之后，需要将 Nios II 的 License 拷贝到 QuartusII 软件的 License.dat 文件后，并且将”HOSTID=**********”语句的*用网卡的物理地址替代。

8. 将光盘中的 TD_SOPC_Board 文件夹拷贝到 Quartus II 安装目录下的 altera\kits\nios2\components 文件夹中。

第三章 数字系统设计基础实验

本章提供了 10 个实验项目作为数字系统设计的基础实验，通过使用 Quartus II 编辑环境使用户对于图形输入、VHDL 硬件描述语言输入等设计手段，自顶向下的 EDA 技术设计方法，仿真、综合等 EDA 分析方法有所掌握。

通过使用 TD-EDA 实验平台及 SOPC 开发板上的 EP1C6 可编程逻辑器件使用户对芯片的结构、功能、引脚分配、系统配置的方法有所掌握。加强用户对数字逻辑电路知识的掌握，了解 EDA 技术及现代数字系统设计方法，培养复杂数字系统分析、设计的能力，为以后更高层次的应用打下坚实的基础。

3.1 基本门电路实验

3.1.1 实验目的

1. 学习 TD-EDA 实验平台及 SOPC 开发板的使用方法；
2. 熟悉 Quartus II 集成环境的使用方法；
3. 学习使用图形输入设计的方法。

3.1.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.1.3 实验内容

本实验使用图形输入方法设计输入非门、与门、或门及异或门逻辑，对电路进行仿真、引脚分配并下载到电路板进行逻辑功能验证。

3.1.4 实验步骤

1. 如图 3-1-1 所示，运行 Quartus II 软件。
2. 选择菜单 File→New Project Wizard，如图 3-1-2 所示，建立一个新工程。
3. 点击”Next”出现如图 3-1-3 所示 New Project Wizard 对话框界面。
4. 点击”Next”出现如图 3-1-4 所示的器件设置对话框界面，选择 SOPC 开发板使用的 Cyclone 系列 EP1C6Q240C8 芯片，一直点击”Next”按钮，完成新工程的建立。
5. 建立新工程后，选择菜单 File→New，弹出如图 3-1-5 所示的新建设计文件选择窗口。创建图形设计文件，选择图 3-1-5 所示对话框中的”Device Design Files”页下的”Block Diagram/Schematic File”；如若创建 VHDL 描述语言设计文件则选择图 3-1-5 所示对话框中”Device Design Files”页下的”VHDL File”。选择好所需要的设计输入方式后点击”OK”按钮，打开图形编辑器界面。

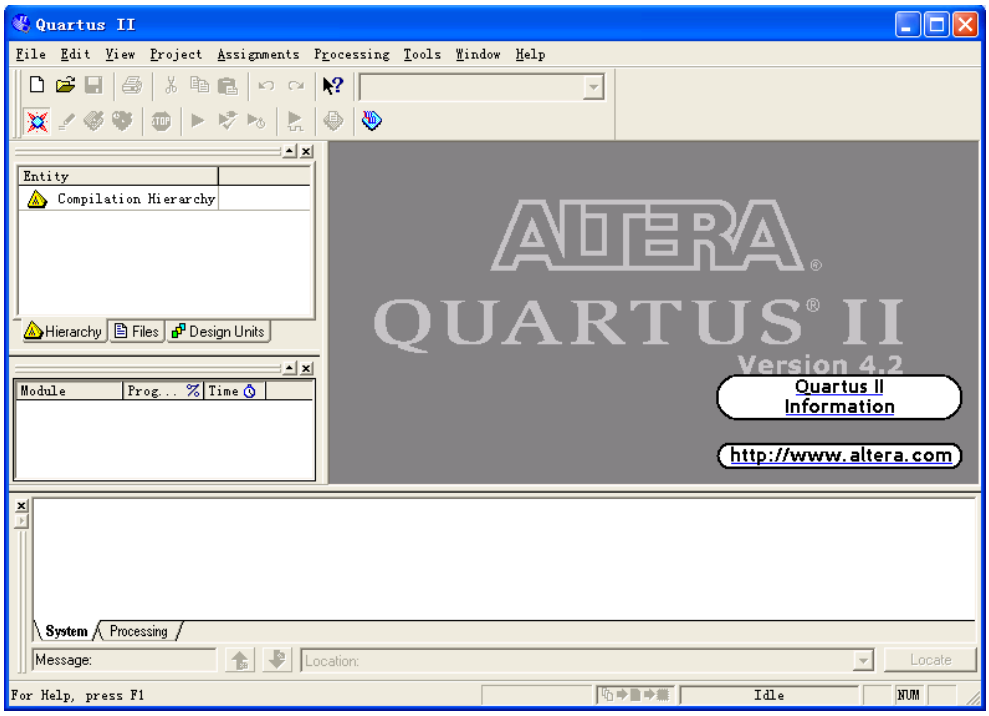


图 3-1-1　运行 QuartusII 软件界面

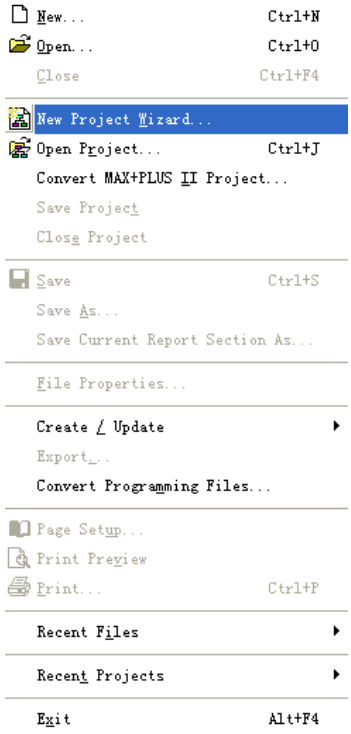


图 3-1-2　建立新工程向导

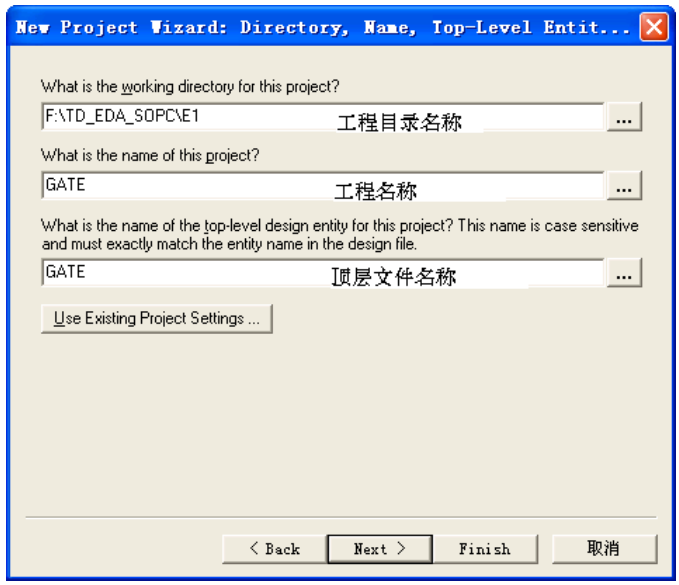


图 3-1-3　New Project Wizard 对话框界面

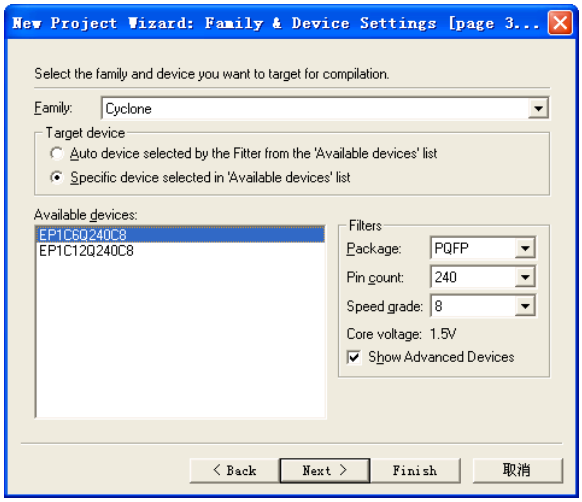


图 3-1-4 器件设置对话框界面

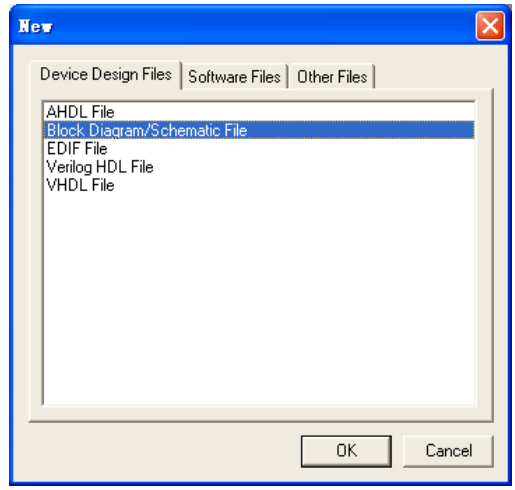


图 3-1-5 新建设计文件选择窗口

6. 选择 File→Save As 菜单，在如图 3-1-6 所示的文件保存对话框中，将创建的图形设计文件的名称保存为工程顶层文件名称。

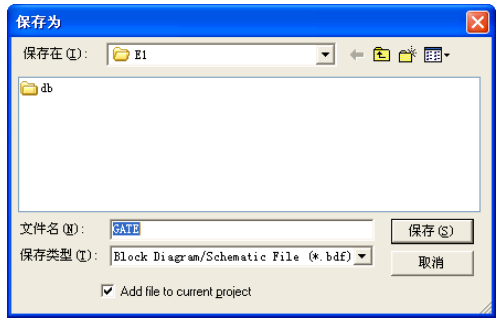


图 3-1-6 文件保存对话框

7. 在图形编辑器窗口中双击鼠标左键或选择菜单"Edit→Insert Symbol"，弹出如图 3-1-7 所示的 Symbol 对话框界面。

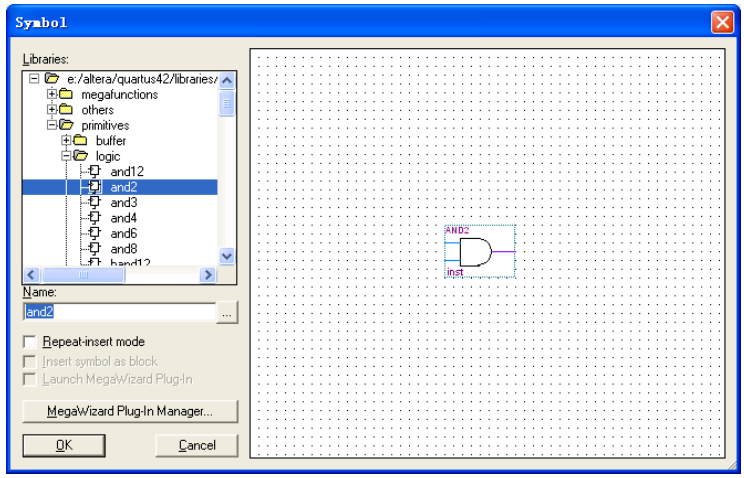


图 3-1-7 Symbol 对话框界面

8. 在 Name 栏中输入 AND2，所选择符号出现在 Symbol 对话框的右边，点击”OK”按钮，选中该符号在合适的位置点击鼠标左键放置符号。重复上述两步，在图形编辑工作区域中分别放置 NOT、AND2、OR2、XOR、INPUT、OUTPUT 等符号。
9. 将所需符号放置完成后，利用连线工具，如图 3-1-8 所示进行连接，并将 INPUT 与 OUTPUT 更改名称。

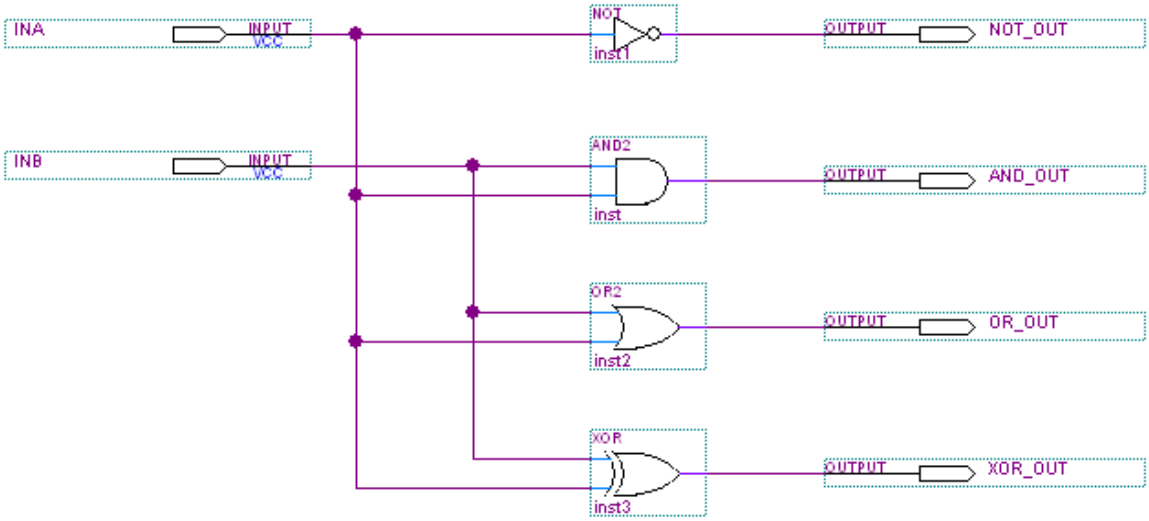


图 3-1-8 门电路实验图形

10. 设计完成后，选择 File→Save As 菜单，将创建的图形文件名称保存为工程顶层文件名 GATE.BDF。选择 Tools→Compiler Tool 菜单，出现如图 3-1-9 所示的编辑工具界面。点击”Start”按钮开始对此工程进行逻辑分析、综合适配、时序分析等。

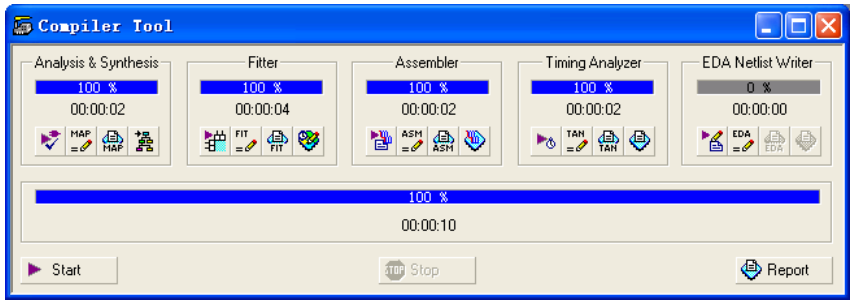


图 3-1-9 编辑工具界面

11. 如果设计正确则如图 3-1-9 所示完全通过各种编译，如果有错误则返回图形编辑工作区域进行修改，直至完全通过编译为止。
12. 编译完成后进行仿真，选择 File→New 菜单在如图 3-1-10 所示新建设计文件窗口中选择”Other Files”页下的”Vector Waveform File”选项，新建一个新的仿真波形文件。
13. 选择 File→Save As 菜单，在文件保存对话框中，将创建的仿真波形文件保存。在如图 3-1-11 所示的波形编辑器窗口的 Name 栏中双击鼠标，弹出如图 3-1-12 所示的增加总线及结点对话框。

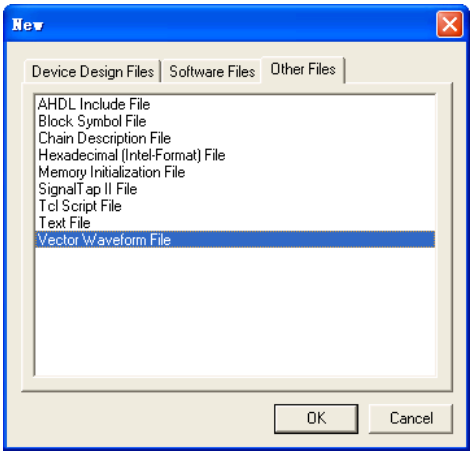


图 3-1-10 新建仿真文件窗口

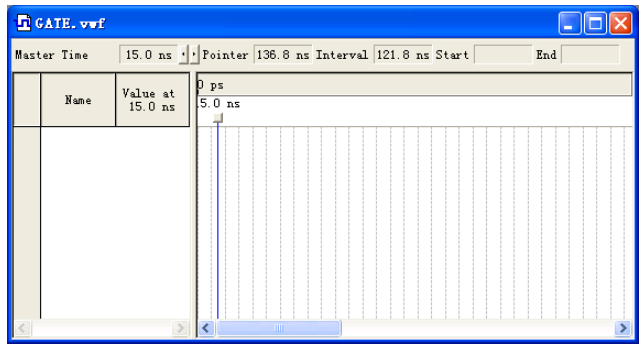


图 3-1-11 波形编辑器窗口

14. 在图 3-1-12 所示对话框中点击”Node Finder”按钮弹出如图 3-1-13 所示的寻找结点对话框。在对话框的 Filter：下拉框中选择 Pins:all，点击”List”按钮，将在 Nodes Found 中列出项目中使用的输入、输出引脚。

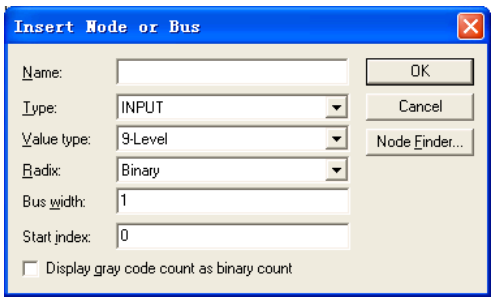


图 3-1-12 增加总线及结点对话框

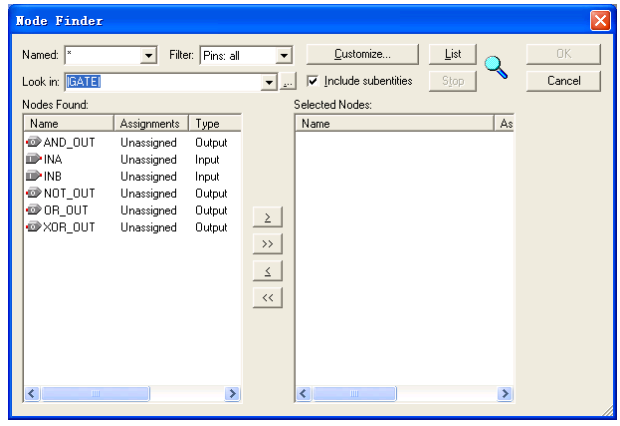


图 3-1-13 寻找结点对话框

15. 在”Node Finder”窗口中选择所需要仿真的引脚(此实验中全部选择)，点击 > 或 >> 按钮，将选择的结点选中到”Selected Nodes”窗口中，点击”OK”按钮。

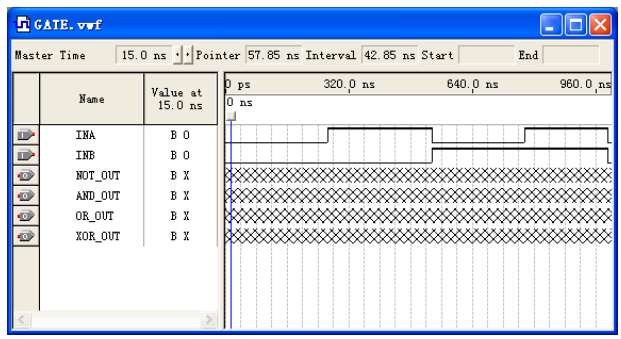


图 3-1-14 波形编辑器窗口

16. 在如图 3-1-14 所示的波形编辑器窗口中，编辑输入引脚的逻辑关系，输入完成后保存仿真波形文件。
17. 选择 Tools→Simulator Tool 菜单，在如图 3-1-15 所示的仿真工具窗口中可以选择时序仿真或功能仿真，指定仿真波形文件的位置等操作。如图选择完成后点击”Start”按钮开始仿真。仿真完成后点击”Report”按钮打开如图 3-1-16 所示的仿真波形窗口。

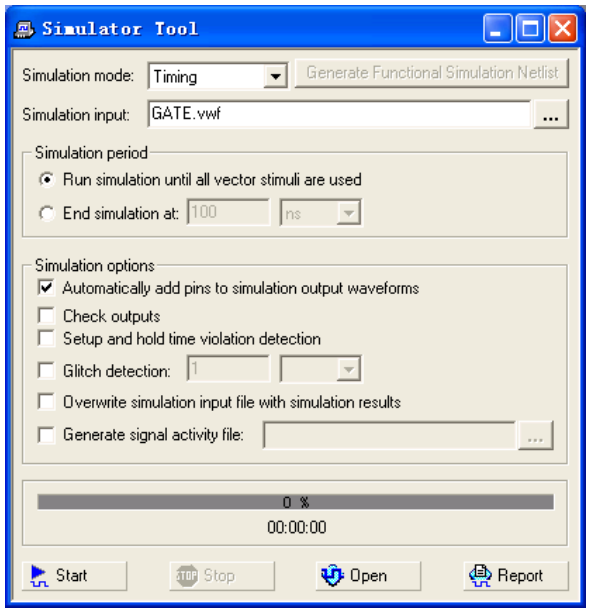


图 3-1-15　仿真工具窗口

18. 分析图 3-1-16 所示波形，证明其逻辑功能是否正确。如果正确就可以进行引脚分配并将设计结果配置到芯片中进行验证，如果不正确则要返回前述步骤进行修改。

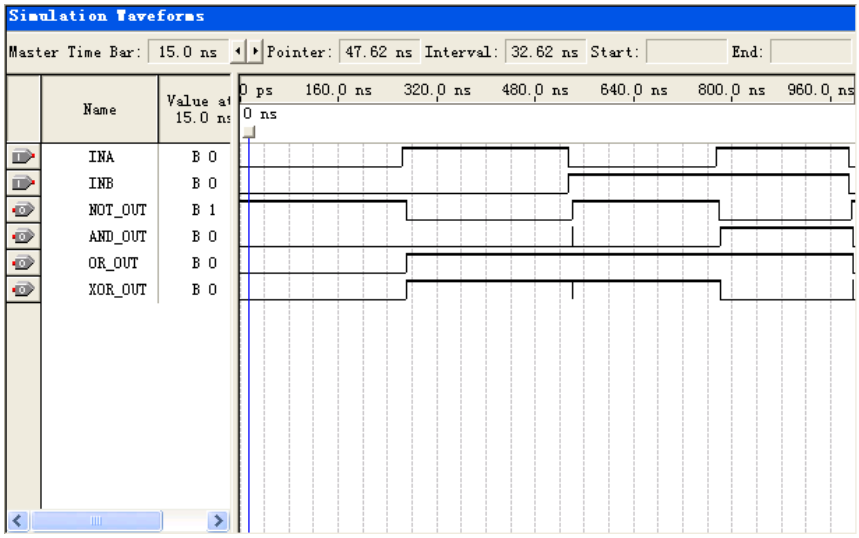


图 3-1-16　仿真波形窗口

19. 仿真正确后选择 Assignments→Assignment Editor 菜单，在如图 3-1-17 所示的 Assignment Editor 窗口中选择 Pin 标签页，在 Edit 中选择输入/输出引脚及对应的 FPGA 引脚。

	To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved	SignalProbe S
1	INB	PIN_193	2	LVTTL	Column I/O	DPCLK3/DQS0T		
2	OR_OUT	PIN_4	1	LVTTL	Row I/O	LVDS13n		
3	AND_OUT	PIN_3	1	LVTTL	Row I/O	LVDS13p/CLKUSR		
4	NOT_OUT	PIN_2	1	LVTTL	Row I/O	LVDS14n		
5	XOR_OUT	PIN_5	1	LVTTL	Row I/O	VREF0B1		
6	INA	PIN_194	2	LVTTL	Column I/O	VREF0B2		
7	<<new>>	<<new>>						

图 3-1-17 Assignment Editor 窗口

20. 引脚分配完成后，选择 Tools→Compiler Tool 菜单，在如图 3-1-9 所示的编辑工具界面中点击”Start”按钮，对此工程进行逻辑分析、综合适配、时序分析等。完成后可选择 Assignments→Timing Closure Floorplan 菜单，观察引脚分配的结果。经过编辑后会生成可以配置到 FPGA 的 SOF 文件及可以配置到外部存储器的 POF 文件。此时就可以将设计配置到芯片中。
21. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-1-18 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口中。仔细检查确保接线无误后打开电源。

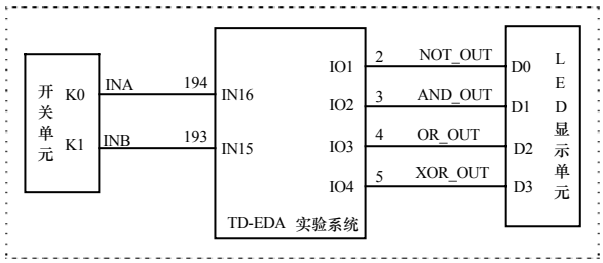


图 3-1-18 门电路实验接线图

22. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，出现如图 3-1-19 所示的编程配置界面。在”Mode”中选择 JTAG，点击”Add File”按钮添加需要配置的 SOF 文件，选中 Program/Configure，点击”Start”按钮就可以对芯片进行配置。

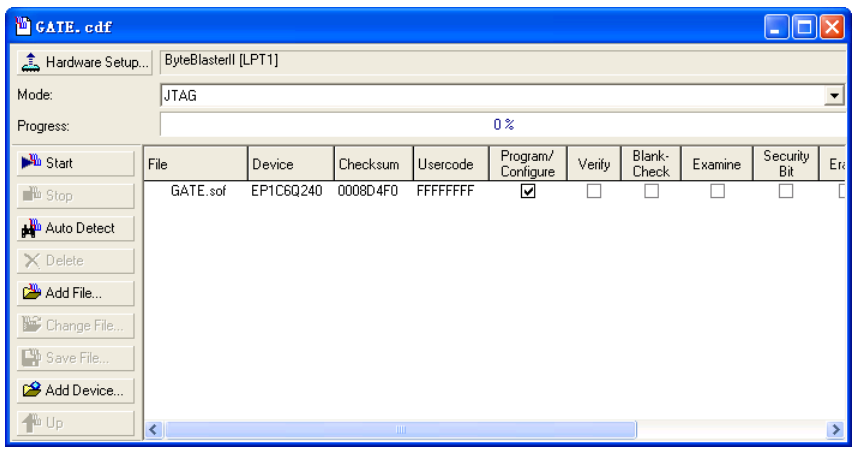


图 3-1-19 编程配置界面

23. 配置完成后拨动开关单元，改变 INA、INB 的输入观察输出结果证明设计是否正确。

3.2 编码器实验

3.2.1 实验目的

1. 学习 TD-EDA 实验平台及 SOPC 开发板的使用方法；
2. 熟悉 Quartus II 集成环境的使用方法；
3. 掌握编码器的工作原理，学习使用 VHDL 语言设计的方法。

3.2.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.2.3 实验内容

本实验使用 VHDL 语言设计一个 8 线-3 线优先编码器，进行仿真、引脚分配并下载到电路板进行功能验证。

3.2.4 实验步骤

1. 运行 Quartus II 软件，选择菜单 File→New Project Wizard，工程名称及顶层文件名称为 CODER，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，在如图 3-2-1 所示的新建设计文件选择窗口中选择创建 VHDL 描述语言设计文件，点击”OK”按钮，打开文本编辑器界面。

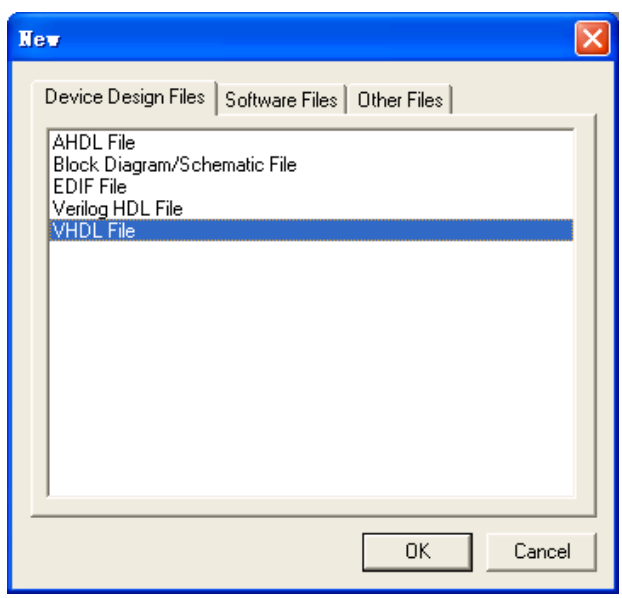


图 3-2-1 新建设计文件选择窗口

3. 选择 File→Save As 菜单，将创建的 VHDL 设计文件名称保存为工程顶层文件名 CODER.VHD。

4. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
-- 8 线-3 线优先编码器的设计：CODER.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY CODER IS
    PORT( DATAIN:   IN STD_LOGIC_VECTOR(1 TO 8);
          DOUT    : OUT STD_LOGIC_VECTOR(0 TO 2));
END ENTITY CODER;
ARCHITECTURE BEHAV OF CODER IS
  SIGNAL SINT : STD_LOGIC_VECTOR(4 DOWNTO 0);
    BEGIN
      PROCESS(DATAIN)
        BEGIN
            IF(DATAIN(8)='1') THEN DOUT<="111";
          ELSIF(DATAIN(7)='1') THEN DOUT<="011";
          ELSIF(DATAIN(6)='1') THEN DOUT<="101";
          ELSIF(DATAIN(5)='1') THEN DOUT<="001";
          ELSIF(DATAIN(4)='1') THEN DOUT<="110";
          ELSIF(DATAIN(3)='1') THEN DOUT<="010";
          ELSIF(DATAIN(2)='1') THEN DOUT<="100";
            ELSE                    DOUT<="000";
            END IF;
      END PROCESS;
  END   ARCHITECTURE BEHAV;
```

5. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮编译源文件。编译无误后建立仿真波形文件 CODER.VWF，选择 Tools→Simulator Tool 菜单进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-2-1 所示

表 3-2-1 编码器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
DATAIN[1]	PIN_175	DATAIN[7]	PIN_181
DATAIN[2]	PIN_176	DATAIN[8]	PIN_182
DATAIN[3]	PIN_177	DOUT[0]	PIN_2
DATAIN[4]	PIN_178	DOUT[1]	PIN_3
DATAIN[5]	PIN_179	DOUT[2]	PIN_4
DATAIN[6]	PIN_180		

7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮重新对此工程进行编译，生成可配置的 SOF 文件。
8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-2-2 所示进行接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口中。仔细检查确保接线无误后打开电源。

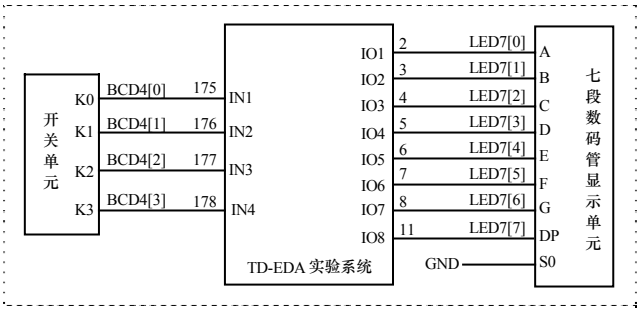


图 3-2-2 编码器实验接线图

9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后拨动逻辑电平开关，观察输出结果证明编码器设计的是否正确。

3.3 译码器实验

3.3.1 实验目的

1. 熟悉 Quartus II 集成环境的使用方法；
2. 掌握译码器的工作原理，了解 BCD/七段译码器的原理；
3. 学习使用 VHDL 语言设计译码器的方法。

3.3.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.3.3 实验内容

本实验使用 VHDL 语言设计一个 BCD/七段译码器，进行仿真、引脚分配并下载到电路板进行功能验证。由四个逻辑电平开关作为 BCD4[3 … 0]的四位输入，输出信号对应数码管的显示段，为高电平时对应的段发亮。

3.3.4 实验步骤

1. 运行 Quartus II 软件，选择菜单 File→New Project Wizard，工程名称及顶层文件名称为 DECODER7，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
3. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--BCD/七段译码器的设计：DECODER7.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY DECODER7 IS
    PORT   ( BCD4 : IN STD_LOGIC_VECTOR(3 downto 0);--四位 BCD 码输入
               LED7 : OUT STD_LOGIC_VECTOR(7 downto 0));--七位输出
END ENTITY DECODER7;
ARCHITECTURE BCD_LED OF DECODER7 IS
BEGIN
  PROCESS( BCD4)
   BEGIN
        CASE BCD4 IS                --gfedcba
          WHEN "0000"   => LED7<="00111111" ;
          WHEN "0001"   => LED7<="00000110" ;
          WHEN "0010"   => LED7<="01011011" ;
          WHEN "0011"   => LED7<="01001111" ;
```

```
            WHEN "0100"    => LED7<="01100110" ;
            WHEN "0101"    => LED7<="01101101" ;
            WHEN "0110"    => LED7<="01111101" ;
            WHEN "0111"    => LED7<="00000111" ;
            WHEN "1000"    => LED7<="01111111" ;
            WHEN "1001"    => LED7<="01101111" ;
            WHEN OTHERS    => LED7<="10000000" ;
        END CASE;
    END PROCESS;
END ARCHITECTURE BCD_LED;
```

4. 保存 VHDL 源程序为工程的顶层文件名 DECODER7.VHD。
5. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 DECODER7.VWF，选择 Tools→Simulator Tool 菜单进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-3-1 所示

表 3-3-1 译码器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
BCD4[0]	PIN_175	LED7 [2]	PIN_4
BCD4 [1]	PIN_176	LED7 [3]	PIN_5
BCD4 [2]	PIN_177	LED7 [4]	PIN_6
BCD4 [3]	PIN_178	LED7 [5]	PIN_7
LED7 [0]	PIN_2	LED7 [6]	PIN_8
LED7 [1]	PIN_3	LED7 [7]	PIN_11

7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮重新对此工程进行编译，生成可配置的 POF 文件。

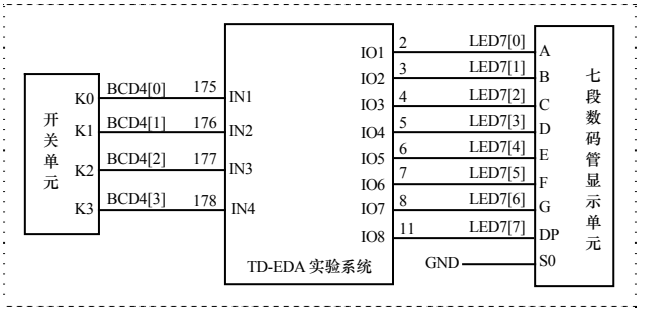


图 3-3-1 译码器实验接线图

8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-3-1 所示进行接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。
9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后拨动逻辑电平开关，观察显示的数字，证明译码器设计的是否正确。

3.4 加法器实验

3.4.1 实验目的

1. 熟悉 Quartus II 集成环境的使用方法，了解自顶向下的设计方法；
2. 掌握加法器的工作原理，使用图形输入的方法设计一个全加器；
3. 学习使用 VHDL 语言设计一个算术逻辑单元。

3.4.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.4.3 实验内容

本实验首先使用图形输入的方法由逻辑门电路设计一个全加器，然后使用 VHDL 语言设计一个具有加法、减法、相等、不相等比较运算的算术逻辑运算单元。分别进行仿真、引脚分配并下载到电路板进行功能验证。

3.4.4 实验步骤

1. 全加器实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 H_ADDER，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建图形设计文件，在图形编辑器界面中完成如图 3-4-1 所示的由与门、非门、异或门构成的半加器电路图。

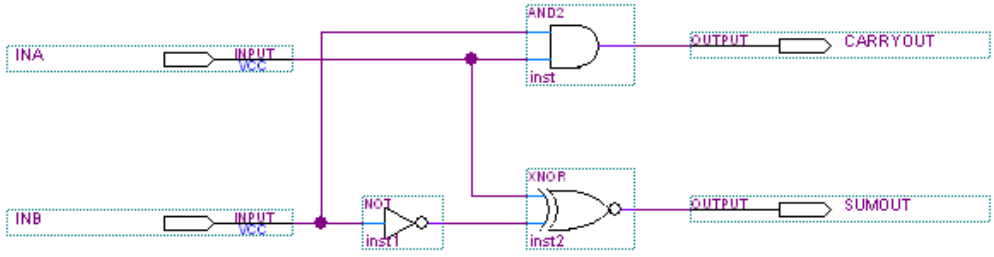


图 3-4-1 半加器电路图

3. 编译源文件，无误后进行仿真，验证半加器的逻辑功能。证明其正确后，选择 File→Create/Update →Create Symbol File for Current File 菜单，为当前工程生成一个符号文件 H_ADDER.BSF 文件。
4. 选择 File→Close Project 菜单关闭工程 H_ADDER。
5. 选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 ADDER，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
6. 将 H_ADDER 工程目录下的文件 H_ADDER.BSF 和 H_ADDER.BDF 拷贝到 ADDER 工

程目录下。

7. 选择 File→New 菜单，创建图形设计文件 ADDER.BDF，在图形编辑器窗口中双击或选择菜单”Edit→Insert Symbol”，在如图 3-4-2 所示的 Symbol 对话框界面中可以找到 H_ADDER 的符号。

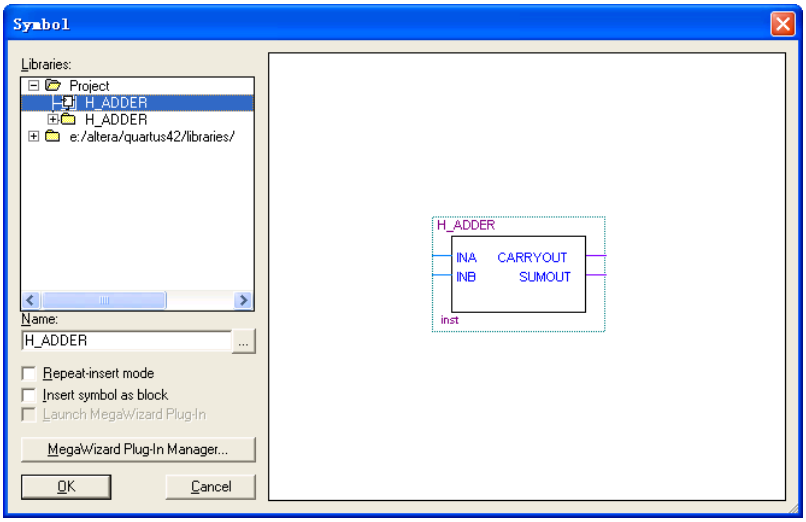


图 3-4-2 半加器符号

8. 在图形编辑器界面中完成如图 3-4-3 所示的全加器电路图。

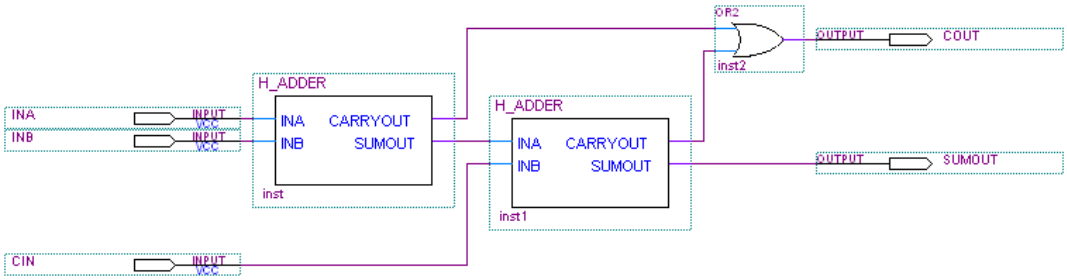


图 3-4-3 全加器电路图

9. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 ADDER.VWF。选择 Tools→Simulator Tool 菜单进行仿真。
10. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-4-1 所示。

表 3-4-1 全加器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
INA	PIN_175	SUMOUT	PIN_2
INB	PIN_176	COUT	PIN_3
CIN	PIN_182		

11. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
12. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-4-4 所示进行接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

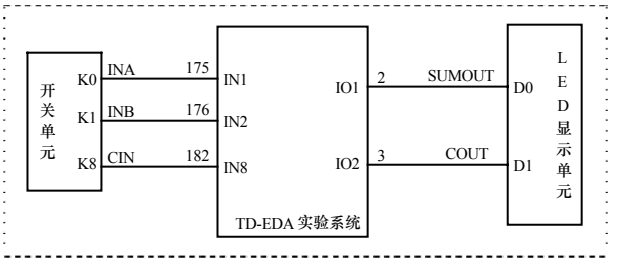


图 3-4-4 全加器实验接线图

13. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
14. 配置完成后拨动逻辑电平开关，证明全加器设计的是否正确。

2．算术逻辑运算单元实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 ALU，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
3. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--算术逻辑单元的 VHDL 设计：ALU.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ALU IS
    PORT( A,B : IN    STD_LOGIC_VECTOR(3 DOWNTO 0);
          OPCODE: IN    STD_LOGIC_VECTOR(1 DOWNTO 0);
          RESULT: OUT STD_LOGIC_VECTOR(4 DOWNTO 0));
END ENTITY ALU;

ARCHITECTURE BEHAVE OF ALU IS
    CONSTANT PLUS : STD_LOGIC_VECTOR(1 DOWNTO 0) :=B"00";
    CONSTANT MINUS : STD_LOGIC_VECTOR(1 DOWNTO 0) :=B"01";
    CONSTANT EQUAL : STD_LOGIC_VECTOR(1 DOWNTO 0) :=B"10";
    CONSTANT NOT_EQUAL: STD_LOGIC_VECTOR(1 DOWNTO 0) :=B"11";
    SIGNAL     INA,INB : STD_LOGIC_VECTOR(4 DOWNTO 0);
```

```
BEGIN
  INA<='0'&A;
  INB<='0'&B;
  PROCESS(OPCODE , A,B)
    BEGIN
      CASE OPCODE IS                                          --这两句要加载 LPM_ADD_SUB
        WHEN PLUS   => RESULT <= INA+INB;                     --OPCODE="00"进行加法运算
        WHEN MINUS => RESULT <= INA+NOT(INB)+1;--OPCODE="01"进行减法运算
        WHEN EQUAL =>                                         --OPCODE="10"进行相等比较
            IF(A=B)   THEN RESULT <=B"00001";
                      ELSE RESULT <=B"11111";
            END IF;
        WHEN NOT_EQUAL =>                                     --OPCODE="11"进行不相等比较
            IF(A/=B) THEN RESULT<=B"00001";
                      ELSE RESULT<=B"11111";
            END IF;
      END CASE;
  END PROCESS;
END ARCHITECTURE BEHAVE;
```

4. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为工程顶层文件名 ALU.VHD。
5. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 ALU.VWF。选择 Tools→Simulator Tool 菜单进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-4-2 所示。

表 3-4-2 算术逻辑运算单元实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
A [0]	PIN_175	OPCODE [0]	PIN_193
A [1]	PIN_176	OPCODE [1]	PIN_194
A [2]	PIN_177	RESULT [0]	PIN_2
A [3]	PIN_178	RESULT [1]	PIN_3
B [0]	PIN_179	RESULT [2]	PIN_4
B [1]	PIN_180	RESULT [3]	PIN_5
B [2]	PIN_181	RESULT [4]	PIN_6
B [3]	PIN_182		

7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。

8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-4-5 所示进行接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

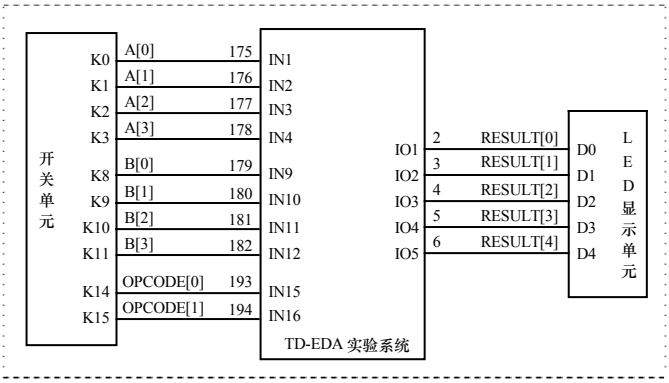


图 3-4-5 算术逻辑运算单元实验接线图

9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后拨动逻辑电平开关，验证算术逻辑单元的正确性。

3.5 乘法器实验

3.5.1 实验目的

1. 熟悉 Quartus II 集成环境提供的 LPM 宏功能模块及其使用方法；
2. 掌握乘法器的工作原理，使用 LPM 宏功能模块设计一个四位乘法器。

3.5.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.5.3 实验内容

Altera 公司为设计人员提供了多种功能的 LPM(Library of Parameterized Modules 即参数可设置模块库)宏模块，从功能上可以把它们分为时序电路宏模块、运算电路宏模块和存储器宏模块三大类。其中时序电路宏模块包括触发器、锁存器、计数器、分频器、多路复用器和移位寄存器；运算电路宏模块包括加法器、减法器、乘法器、除法器、绝对值运算器、数值比较器、编译码器和奇偶校验发生器；存储器宏模块包括 RAM、FIFO 和 ROM。

本实验使用 Quartus 软件中提供的 LPM_MULT 宏模块设计一个四位数据的乘法器，进行仿真、引脚分配并下载到电路板进行功能验证。通过此实验使用户了解利用 LPM 进行设计的方法。

3.5.4 实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 4BITMULT，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。

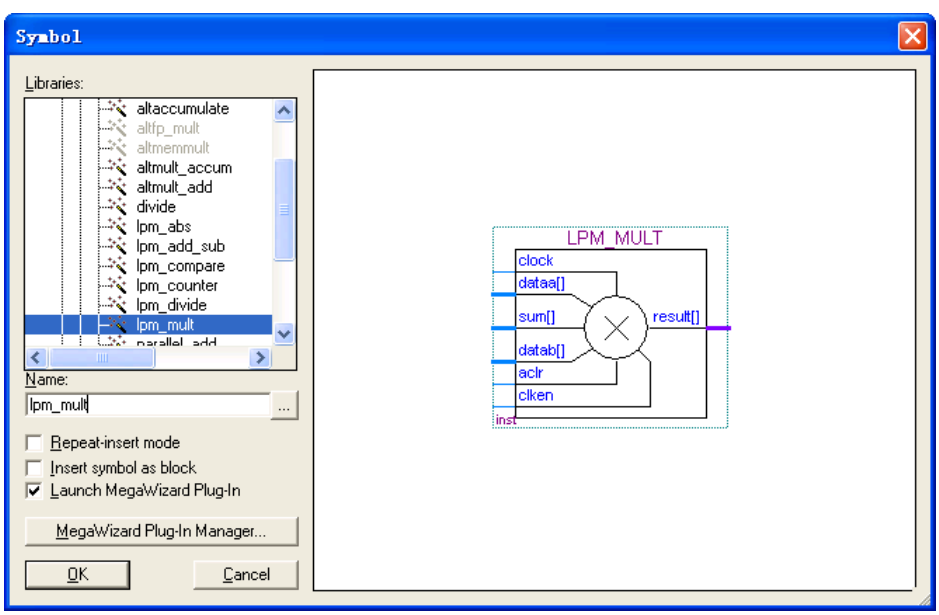


图 3-5-1 LPM_MULT 宏模块的符号

2. 选择 File→New 菜单，创建图形设计文件，在图形编辑器窗口中双击鼠标或选择菜单”Edit→Insert Symbol”，在如图 3-5-1 所示的 Symbol 对话框界面中可以找到 LPM_MULT 宏模块，点击”OK”按钮，在出现的界面中选择 VHDL，点击”NEXT”按钮。
3. 在如图 3-5-2 所示的 LPM_MULT 宏模块的设置对话框中选择输入数据的宽度为 4 位。点击”NEXT”按钮。在随后的图形中一直点击”NEXT”按钮。
4. 将设计好的 LPM_MULT 模块放置到图形编辑器界面中，在图形编辑器界面中完成如图 3-5-3 所示的乘法器电路图。

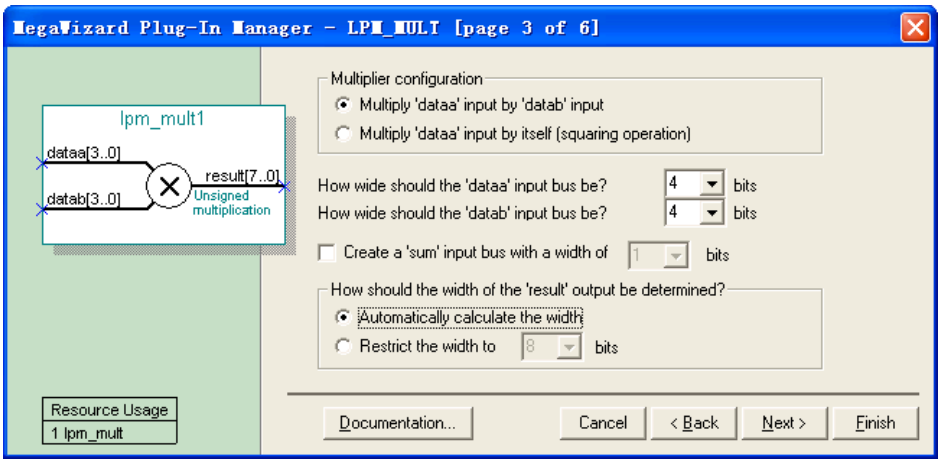


图 3-5-2 LPM_MULT 宏模块的设置

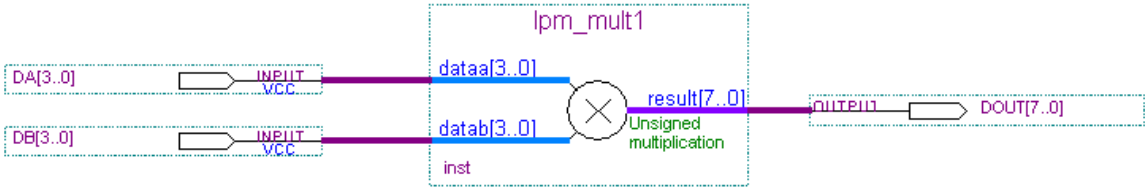


图 3-5-3 乘法器电路图

表 3-5-1 乘法器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
DA [0]	PIN_175	DOUT [0]	PIN_2
DA [1]	PIN_176	DOUT [1]	PIN_3
DA [2]	PIN_177	DOUT [2]	PIN_4
DA [3]	PIN_178	DOUT [3]	PIN_5
DB [0]	PIN_183	DOUT [4]	PIN_6
DB [1]	PIN_184	DOUT [5]	PIN_7
DB [2]	PIN_185	DOUT [6]	PIN_8
DB [3]	PIN_186	DOUT [7]	PIN_11

5. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 4BITMULT.VWF。选择 Tools→Simulator Tool 菜单进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-5-1 所示。
7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-5-4 所示进行接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

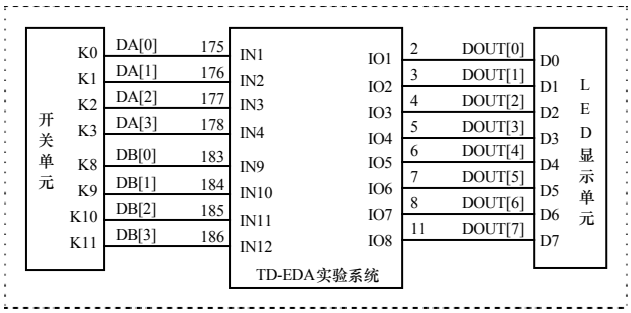


图 3-5-4 乘法器实验接线图

9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后拨动逻辑电平开关，验证乘法器的正确性。

3.6 寄存器实验

3.6.1 实验目的

1. 了解寄存器的分类方法，掌握各种寄存器的工作原理；
2. 学习使用 VHDL 语言设计两种类型的寄存器。

3.6.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.6.3 实验内容

寄存器中二进制数的位可以用两种方式移入或移出寄存器。第一种方法是以串行的方式将数据每次移动一位，这种方法称之为串行移位(Serial Shifting)，线路较少，但耗费时间较多。第二种方法是以并行的方式将数据同时移动，这种方法称之为并行移位(Parallel Shifting)，线路较为复杂，但是数据传送的速度较快。

因此，按照数据进出移位寄存器的方式，可以将移位寄存器分为四种类型：串行输入串行输出移位寄存器(Serial In- Serial Out)、串行输入并行输出移位寄存器(Serial In- Parallel Out)、并行输入串行输出移位寄存器(Parallel In- Serial Out)、并行输入并行输出移位寄存器(Parallel In- Parallel Out)。

本实验使用 VHDL 语言设计一个八位并行输入串行输出右移移位寄存器(Parallel In- Serial Out)和一个八位串行输入并行输出寄存器(Serial In- Parallel Out)，分别进行仿真、引脚分配并下载到电路板进行功能验证。

3.6.4 实验步骤

1．并行输入串行输出移位寄存器实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 SHIFT8R，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
3. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--带有同步并行预置功能的 8 位右移移位寄存器:SHIFT8R.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY SHIFT8R IS
   PORT( CLK , LOAD : IN STD_LOGIC ;   --CLK 是移位时钟信号、LOAD 是并行数据预置使能信号
         DIN    : IN STD_LOGIC_VECTOR(7 DOWNTO 0); --DIN 是 8 位并行预置数据端口
         QB     : OUT STD_LOGIC ); --QB 是串行输出端口
END ENTITY SHIFT8R;
```

```
ARCHITECTURE BEHAV OF SHIFT8R IS
  BEGIN
    PROCESS(CLK,LOAD)
      VARIABLE REG8 : STD_LOGIC_VECTOR( 7 DOWNTO 0);
        BEGIN
          IF CLK'EVENT AND CLK='1' THEN
              IF LOAD='1' THEN
                  REG8 := DIN;          --装载新数据
              ELSE
                  REG8(6 DOWNTO 0):= REG8(7 DOWNTO 1);
                  REG8(7):='0';
             END IF;
          END IF;
        QB <= REG8(0);                  -- 输出最低位
    END PROCESS;
END ARCHITECTURE BEHAV;
```

4. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为工程顶层文件名 SHIFT8R.VHD。
5. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 SHIFT8R.VWF。选择 Tools→Simulator Tool 菜单进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-6-1 所示

表 3-6-1 并入串出移位寄存器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
DIN [0]	PIN_175	DIN [6]	PIN_181
DIN [1]	PIN_176	DIN [7]	PIN_182
DIN [2]	PIN_177	LOAD	PIN_194
DIN [3]	PIN_178	CLK	PIN_28
DIN [4]	PIN_179	QB	PIN_2
DIN [5]	PIN_180		

7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-6-1 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。
9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后验证移位寄存器的正确性。

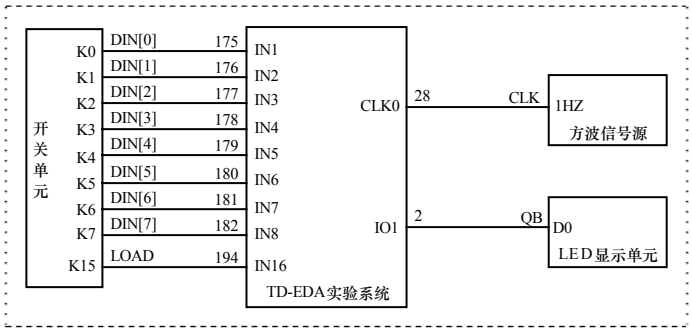


图 3-6-1 并入串出移位寄存器实验接线图

2．串行输入并行输出寄存器实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 SHIFT8，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
3. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--串行输入并行输出寄存器:SHIFT8.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;

ENTITY SHIFT8 IS
   PORT( DI ,CLK : IN    STD_LOGIC;
         DOUT    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ENTITY SHIFT8;

ARCHITECTURE BEHA OF SHIFT8 IS
    SIGNAL TMP : STD_LOGIC_VECTOR(7 DOWNTO 0);
        BEGIN
          PROCESS(CLK)
              BEGIN
                IF(CLK'EVENT AND CLK='1') THEN
                    TMP(7)<=DI;
                     FOR I IN 1 TO 7 LOOP
                          TMP(7-I)<=TMP(8-I);
                     END LOOP;
                END IF;
            END PROCESS;
       DOUT<=TMP;
END ARCHITECTURE BEHA;
```

4. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为工程顶层文件名 SHIFT8.VHD。
5. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 SHIFT8.VWF。选择 Tools→Simulator Tool 菜单进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-6-2 所示

表 3-6-2 串入并出寄存器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	DOUT [3]	PIN_5
DI	PIN_175	DOUT [4]	PIN_6
DOUT [0]	PIN_2	DOUT [5]	PIN_7
DOUT [1]	PIN_3	DOUT [6]	PIN_8
DOUT [2]	PIN_4	DOUT [7]	PIN_11

7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-6-2 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

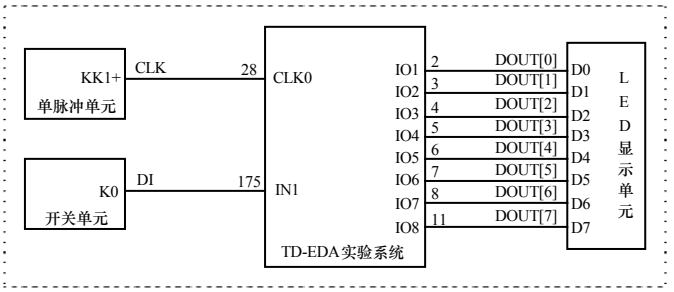


图 3-6-2 串入并出寄存器实验接线图

9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后验证串入并出寄存器的正确性。

3.7 计数器实验

3.7.1 实验目的

1. 掌握计数器的工作原理；
2. 学习使用 VHDL 语言设计一个十进制计数器。

3.7.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.7.3 实验内容

本实验使用 VHDL 语言设计十进制加法计数器，用发光二极管显示计数值。计数允许 EN 及复位 RST 信号由逻辑电平给出，计数时钟由实验箱上连续脉冲单元的 1Hz 给出。分别进行仿真、引脚分配并下载到电路板进行功能验证。

3.7.4 实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 COUNT10，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
3. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--此例是一个带有异步复位和同步时钟使能的十进制加法计数器:COUNT10.VHD
  LIBRARY IEEE;
  USE IEEE.STD_LOGIC_1164.ALL;
  USE IEEE.STD_LOGIC_UNSIGNED.ALL;

  ENTITY COUNT10 IS
    PORT( CLK,RST,EN :   IN   STD_LOGIC;
                  CQ      :   OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
  END ENTITY COUNT10;
  ARCHITECTURE BEH OF COUNT10 IS
    BEGIN
      PROCESS(CLK,RST,EN)
    VARIABLE CQI   : STD_LOGIC_VECTOR(3 DOWNTO 0);
      BEGIN
        IF RST='0' THEN CQI:=(OTHERS=>'0'); --计数器复位，这种操作独立于时钟，因而称为异步
        ELSIF CLK'EVENT AND CLK='1'   THEN      --检测时钟上升沿
          IF EN='1' THEN                        --检测是否允许计数 此语句构成了二输入的与门
            IF CQI < "1010" THEN CQI:= CQI + 1;   --允许计数构成了比较器
```

```
            ELSE CQI:=(OTHERS =>'0');     --大于 9 计数值清零 此语句构成二选一的多路选择器
            END IF;
        END IF;
    END IF;
            IF CQI="1010" THEN CQI:="0000"; --计数等于 9，输出进位信号
            END IF;
            CQ<=CQI;                              --计数值向端口输出
     END PROCESS;
END ARCHITECTURE BEH;
```

4. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为工程顶层文件名 COUNT10.VHD。
5. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 COUNT10.VWF。选择 Tools→Simulator Tool 菜单进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-7-1 所示

表 3-7-1 计数器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	CQ [0]	PIN_2
EN	PIN_175	CQ [1]	PIN_3
RST	PIN_131	CQ [2]	PIN_4
		CQ [3]	PIN_5

7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-7-1 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

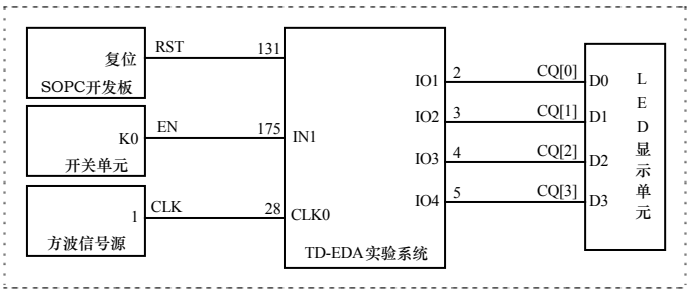


图 3-7-1 计数器实验接线图

9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后验证计数器的正确性。

3.8 分频器实验

3.8.1 实验目的

1. 掌握分频器的工作原理，了解半整数分频器的工作原理；
2. 学习使用 VHDL 语言设计一个可以设置分频系数的半整数分频器；
3. 掌握在 Quartus II 软件中使用层次化设计的方法。

3.8.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块，示波器或逻辑分析仪一台。

3.8.3 实验内容

分频器通常用于对某个给定频率进行分频，得到所需的频率。整数分频器的实现比较简单，通常采用标准的计数器。但是在某些场合系统时钟源与所需的频率不成整数倍关系，此时可以采用小数分频器进行分频。例如有一个 1MHz 的时钟源，但电路中需要一个 400Hz 的时钟信号，由于分频比为 2.5，此时整数分频器将不能胜任。

利用可编程逻辑器件进行小数分频的基本原理是：采用脉冲吞吐计数和锁相环技术，设计两个不同分频比的整数分频器，通过控制单位时间内两种分频比出现的不同次数，从而获得所需要的小数分频值。例如设计分频系数为 10.1 的分频器，可以将分频器设计成 9 次 10 分频 1 次 11 分频，这样总的分频值为：F=(9×10＋1×11)/（9＋1）=10.1。

从这种实现方法的特点可以看出，由于分频器的分频值在不断改变，因此分频后得到的信号抖动较大。当分频系数为 N-0.5(N 为整数)时，可控制扣除脉冲的时间，使输出为一个稳定的脉冲频率，而不是一次 N 分频，一次 N-1 分频。一个 N-0.5(N 为整数)的分频器电路的原理如图 3-8-1 所示：

图 3-8-1 半整数分频器的原理图

本实验采用层次化的设计方法，顶层的原理图输入调用了半整数分频器符号，半整数分频器的输出经过一个 D 触发器输出方波。底层的半整数分频器使用 VHDL 语言设计一个可预置系数的实现 N=1~15 的半整数分频器，并且在此程序中调用子模块 D_HEX。子模块 D_HEX 使用 VHDL 语言设计了将输入信号经过译码后驱动两个 LED 进行显示。

分频器的预置输入、CS 使能信号由逻辑电平给出，计数时钟由实验箱上连续脉冲单元的 1MHz 信号提供，输出信号驱动 LED 数码管，用于显示分频的模 N，用虚拟逻辑分析仪或示波器可观察到输出的分频信号频率随模 N 的变换。分别进行仿真、引脚分配并下载到电路板进行功能验证。

3.8.4 实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 DECOUNT，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
3. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--一个可预置系数的半整数分频器实现 N=1~15 的分频：DECOUNT.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY DECOUNT IS
  PORT(CS          : IN   STD_LOGIC ;--使能
       INCLK       : IN   STD_LOGIC; --时钟输入
       PRESET      : IN   STD_LOGIC_VECTOR(3 DOWNTO 0);--预置分频值
       LED         : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);--LED 数码管数据信号
       OUTCLK      : BUFFER STD_LOGIC); --时钟输出
END ENTITY DECOUNT;
ARCHITECTURE BEHAV OF DECOUNT IS
  SIGNAL CLK ,DIVIDE2 : STD_LOGIC;
  SIGNAL COUNT            : STD_LOGIC_VECTOR(3 DOWNTO 0);
    COMPONENT D_HEX     --子模块例化语句
      PORT(   CS      : IN   STD_LOGIC;
              DATA    : IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
              HEX_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
    END COMPONENT;
    BEGIN
      CLK<= INCLK XOR DIVIDE2;
        PROCESS(CLK,CS)
          BEGIN
           IF(CS='0') THEN
             IF(CLK'EVENT AND CLK='1')THEN
                IF(COUNT="0000")THEN
                      COUNT<=PRESET-1;
                      OUTCLK<='1';
                ELSE
                      COUNT<=COUNT-1;
                      OUTCLK<='0';
                END IF;
```

```
                END IF;
            END IF;
        END PROCESS;
    PROCESS(OUTCLK)
        BEGIN
            IF(OUTCLK'EVENT AND OUTCLK='1') THEN
              DIVIDE2<=NOT DIVIDE2;
            END IF;
    END PROCESS;
DISPLAY1: D_HEX        --调用子模块
  PORT MAP(CS=>CS,DATA=>PRESET,
    HEX_OUT =>LED);
 END ARCHITECTURE BEHAV;
```

4. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为 DECOUNT.VHD。
5. 因为 DECOUNT.VHD 程序中调用了子模块 D_HEX，所以还应该编写 D_HEX 子程序。选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
6. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--LED 显示子模块：D_HEX.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY D_HEX IS
  PORT(CS          : IN STD_LOGIC;
       DATA        : IN   STD_LOGIC_VECTOR(3 DOWNTO 0);
       HEX_OUT : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
       SEG         : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END ENTITY D_HEX;
ARCHITECTURE BEHAV OF D_HEX IS
 SIGNAL COM       : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
   COM<=DATA;
   PROCESS(COM,CS)
      BEGIN
     IF(CS='0') THEN
         CASE COM IS       --GFEDCBA
          WHEN "0000"    => HEX_OUT <="00111111" ; --0
          WHEN "0001"    => HEX_OUT <="00000110" ; --1
          WHEN "0010"    => HEX_OUT <="01011011" ; --2
```

```
                WHEN "0011"    => HEX_OUT <="01001111" ;--3
                WHEN "0100"    => HEX_OUT <="01100110" ;--4
                WHEN "0101"    => HEX_OUT <="01101101" ;--5
                WHEN "0110"    => HEX_OUT <="01111101" ;--6
                WHEN "0111"    => HEX_OUT <="00000111" ;--7
                WHEN "1000"    => HEX_OUT <="01111111" ;--8
                WHEN "1001"    => HEX_OUT <="01101111" ;--9
                WHEN "1010"    => HEX_OUT <="01110111" ;--A
                WHEN "1011"    => HEX_OUT <="01111100" ;--B
                WHEN "1100"    => HEX_OUT <="00111001" ;--C
                WHEN "1101"    => HEX_OUT <="01011110" ;--D
                WHEN "1110"    => HEX_OUT <="01111001" ;--E
                WHEN "1111"    => HEX_OUT <="01110001" ;--F
                WHEN OTHERS    => HEX_OUT <="10000000" ;
    END CASE;
END IF;
 IF(CS='1')    THEN
    CASE COM IS
            WHEN OTHERS=>HEX_OUT<="10000000" ;
    END CASE;
 END IF;
END PROCESS;
END ARCHITECTURE BEHAV;
```

7. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为 D_HEX.VHD。
8. 选择 Tools→Compiler Tool 菜单，编译 DECOUNT.VHD 源文件。编译无误后建立仿真波形文件 DECOUNT.VWF。选择 Tools→Simulator Tool 菜单进行仿真。证明其正确后，选择 File→Create/Update →Create Symbol File for Current File 菜单，为当前工程生成一个符号文件 DECOUNT.BSF 文件。选择 File→Close Project 菜单关闭工程 DECOUNT。
9. 选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 ODD_DEV_F，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
10. 将 DECOUNT 工程目录下的 D_HEX.VHD、DECOUNT.VHD 和 DECOUNT.BSF 文件拷贝到 ODD_DEV_F 工程目录下。
11. 选择 File→New 菜单，创建图形设计文件，在图形编辑器窗口中双击鼠标或选择菜单“Edit→Insert Symbol”，在 Symbol 对话框界面中可以找到 DECOUNT 的符号。在图形编辑器界面中完成如图 3-8-2 所示的分频器电路图。
12. 选择 File→Save As 菜单，将设计的图形文件保存为工程顶层文件名 ODD_DEV_F.BDF。
13. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 ODD_DEV_F.VWF。选择 Tools→Simulator Tool 菜单进行仿真。

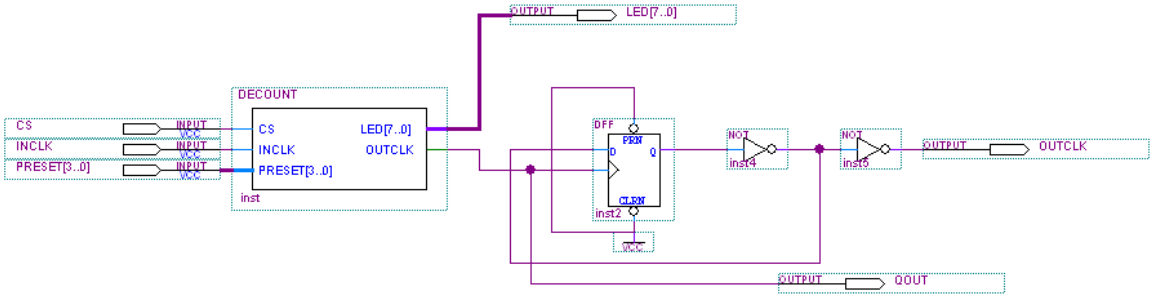


图 3-8-2　分频器电路图

14. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-8-1 所示。

表 3-8-1　分频器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
LED[0]	PIN_2	INCLK	PIN_28
LED[1]	PIN_3	PRESET[0]	PIN_175
LED[2]	PIN_4	PRESET[1]	PIN_176
LED[3]	PIN_5	PRESET[2]	PIN_177
LED[4]	PIN_6	PRESET[3]	PIN_178
LED[5]	PIN_7	CS	PIN_182
LED[6]	PIN_8	OUTCLK	PIN_12
LED[7]	PIN_11	QOUT	PIN_13

15. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。

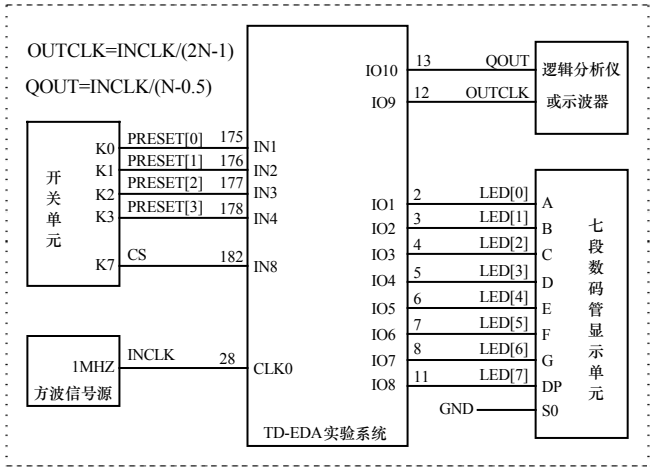


图 3-8-3　分频器实验接线图

16. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-8-3 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。
17. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
18. 配置完成后，验证分频器的正确性。

3.8.5 实验结果

实验结果如表 3-8-2 所示，OUTCLK 的输出是 QOUT 输出波形经过二分频的方波。

表 3-8-2 分频器实验结果

分频模 N 值	QOUT 引脚输出频率(KHz)	分频模 N 值	QOUT 引脚输出频率(KHz)
N＝2	F=1MHz/(2-0.5)=667	N＝3	F=1MHz/(3-0.5)=400
N＝4	F=1MHz/(4-0.5)=286	N＝5	F=1MHz/(5-0.5)=222
N＝6	F=1MHz/(6-0.5)=182	N＝7	F=1MHz/(7-0.5)=154
N＝8	F=1MHz/(8-0.5)=133	N＝9	F=1MHz/(9-0.5)=118
N＝10	F=1MHz/(10-0.5)=105	N＝11	F=1MHz/(11-0.5)=95
N＝12	F=1MHz/(12-0.5)=87	N＝13	F=1MHz/(13-0.5)=80
N＝14	F=1MHz/(14-0.5)=74	N＝15	F=1MHz/(15-0.5)=69

3.9 存储器实验

3.9.1 实验目的

1. 掌握存储器的结构及工作原理；
2. 掌握 SRAM 的工作原理，使用 RAM 宏模块设计一个数据存储器；
3. 掌握 FIFO 的原理，使用 FIFO 宏模块设计一个 FIFO 存储器。

3.9.2 实验设备

PC 微机一台。

3.9.3 实验内容

存储器是数字系统的重要组成部分，数据处理单元的结果需要存储，许多处理单元的初始化也需要存放在存储器中。存储器还可以完成一些特殊的功能，如多路复用、数值计算、脉冲形成、特殊序列产生以及数字频率合成等。

Quartus II 软件提供了 RAM、ROM 和 FIFO 等宏模块，Altera 公司的许多 CPLD/FPGA 器件内部都有存储器模块，适合于存储器的设计。设计者可以很方便的设计各种类型的存储器。

RAM(Random Access Memory,随机存取存储器)可以随时在任一指定地址写入或读取数据，它的最大优点是可以方便读出/写入数据，但是 RAM 存在易失性的缺点，掉电后所存数据便会丢失。RAM 的应用十分广泛，它是计算机系统的重要组成部分，在数字信号处理中 RAM 作为数据存储单元是必不可少的。Quartus II 软件提供了多种 RAM 宏模块，这些宏模块的功能如表 3-9-1 所示。

表 3-9-1 RAM 宏模块功能描述

宏模块名称	功能描述
csdpram	参数化循环共享双端口 RAM
lpm_ram_dp	参数化双端口 RAM
lpm_ram_dq	参数化 RAM，输入/输出端口分离
lpm_ram_io	参数化 RAM，输入/输出共用一个端口

FIFO(First In First Out，先进先出)是一种特殊功能的存储器，数据以到达 FIFO 输入端口的先后顺序依次存储在存储器中，并以相同的顺序从 FIFO 的输出端口送出，所以 FIFO 内数据的写入和读取只受读/写时钟和读/写请求信号的控制，不需要读/写地址线。FIFO 分为同步和异步，同步 FIFO 是指数据输入输出的时钟频率相同，异步 FIFO 是指数据输入输出的时钟频率可以不相同。FIFO 在数字系统中有着十分广泛的应用，可以作为并行数据延迟线，数据缓冲存储器以及速率变换器等。Quartus II 软件提供了多种 FIFO 宏模块，这些宏模块的功能如表 3-9-2 所示。

本实验利用 Quartus II 软件提供的 RAM 模块”lpm_ram_dq”设计一个数据存储器，使用 Quartus II 软件进行仿真。”lpm_ram_dq”宏模块的逻辑参数如表 3-9-3 所示。

表 3-9-2 FIFO 宏模块功能描述

宏模块名称	功能描述
csfifo	参数化循环共享 FIFO
dcfifo	参数化双时钟 FIFO
scfifo	参数化单时钟 FIFO
lpm_fifo	参数化单时钟 FIFO
lpm_fifo_dc	参数化单时钟 FIFO

表 3-9-3 ”lpm_ram_dq”宏模块的基本逻辑参数

	端口名称	功能描述
输入端口	data[ ]	输入数据
	Address[ ]	地址端口
	We	写使能端口，高电平时向 RAM 写入数据
	inclock	同步写入时钟
	outclock	同步读取时钟
输出端口	q[ ]	数据输出端口
参数设置	LPM_WIDTH	data[ ]和 q[ ]端口的数据线宽度
	LPM_WIDTHAD	address[ ]端口宽度
	LPM_NUMWORDS	RAM 中存储单元的数目
	USE_EAB	选择使用”EAB”或“逻辑单元”实现 RAM 功能，”ON”为使用 EAB，“OFF”为使用逻辑单元

利用 Quartus II 软件提供的”lpm_fifo_dc”模块设计一个数据速率变换电路。”lpm_fifo_dc”模块是参数化双时钟 FIFO 模块，可以用来构成异步 FIFO，它的逻辑参数如表 3-9-4 所示。

表 3-9-4 ”lpm_fifo_dc”宏模块的基本逻辑参数

	端口名称	功能描述
输入端口	data[ ]	输入数据
	rdclock	同步读取时钟，上升沿触发
	wrclock	同步写入时钟，上升沿触发
	wrreq	写请求控制，wrfull=1 时，写禁止
	rdreq	读请求控制，rdempty=1 时，读禁止
输出端口	q[ ]	数据输出端口
参数设置	LPM_WIDTH	data[ ]和 q[ ]端口的数据线宽度
	LPM_WIDTHU	rdusedw[ ]和 wrusedw[ ]端口宽度，取值一般为 CEIL
	LPM_NUMWORDS	FIFO 中存储单元的数目
	USE_EAB	选择使用”EAB”或“逻辑单元”实现 FIFO 功能，”ON”为使用 EAB，“OFF”为使用逻辑单元

3.9.4 实验步骤

1．RAM 数据存储器实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 RAM，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建图形设计文件，在图形编辑器窗口中双击或选择菜单“Edit→Insert Symbol”,在如图 3-9-1 所示的 Symbol 对话框中可以找到 LPM_RAM_DQ 宏模块符号，点击”OK”按钮，在出现的界面中选择 VHDL，点击”NEXT”按钮。

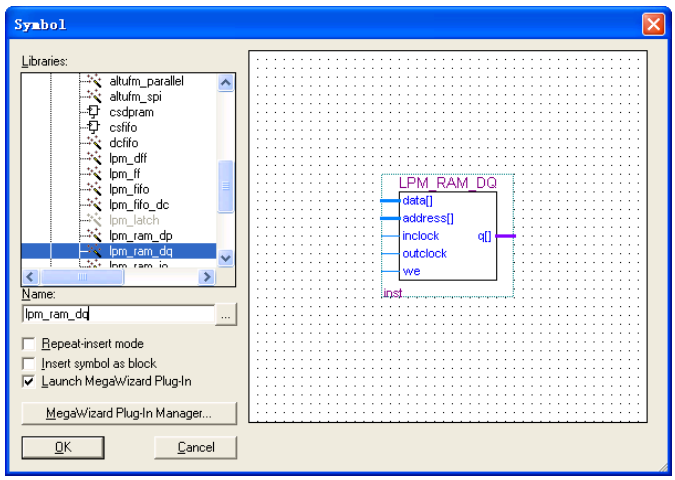


图 3-9-1 Symbol 对话框界面

3. 在如图 3-9-2 所示的 LPM_RAM_DQ 宏模块中如图设置，点击”NEXT”按钮。

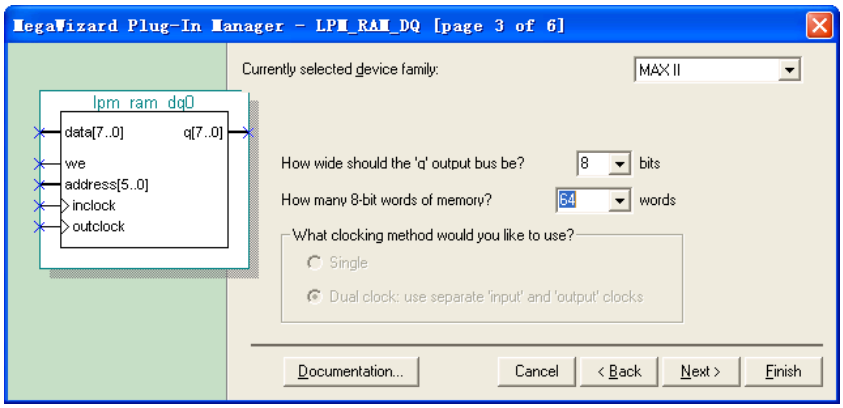


图 3-9-2 LPM_RAM_DQ 宏模块的设置(一)

4. 在如图 3-9-3 所示的 LPM_RAM_DQ 宏模块中如图设置，点击”NEXT”按钮。

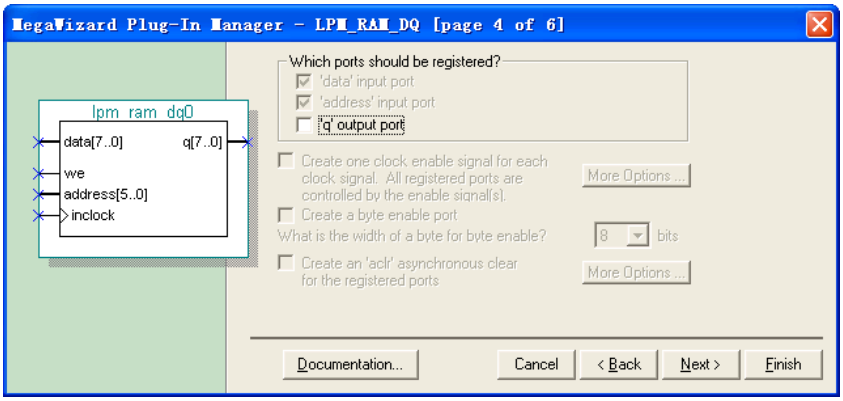


图 3-9-3 LPM_RAM_DQ 宏模块的设置(二)

5. 在如图 3-9-4 所示的 LPM_RAM_DQ 宏模块中如图设置，点击”NEXT”按钮。

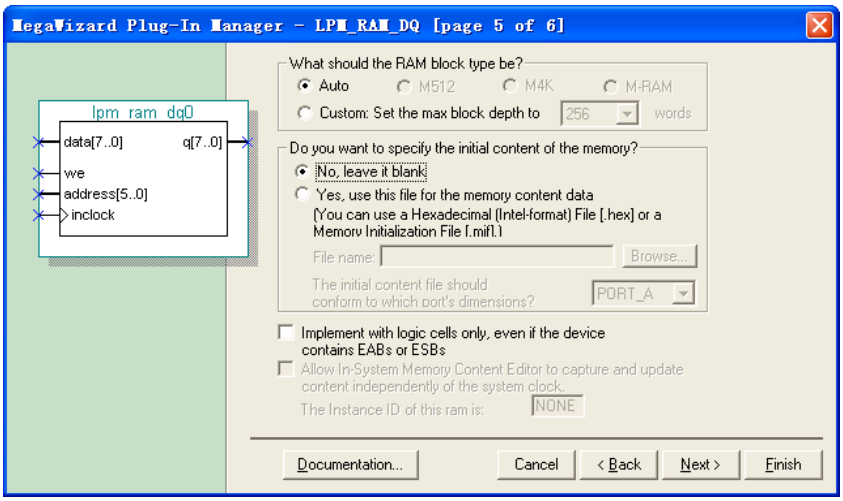


图 3-9-4 LPM_RAM_DQ 宏模块的设置(三)

6. 在 LPM_RAM_DQ 宏模块设置窗口点击”Finish”按钮，将设计好的 LPM_RAM_DQ 模块放置到图形编辑器界面中，完成如图 3-9-5 所示的存储器电路图。

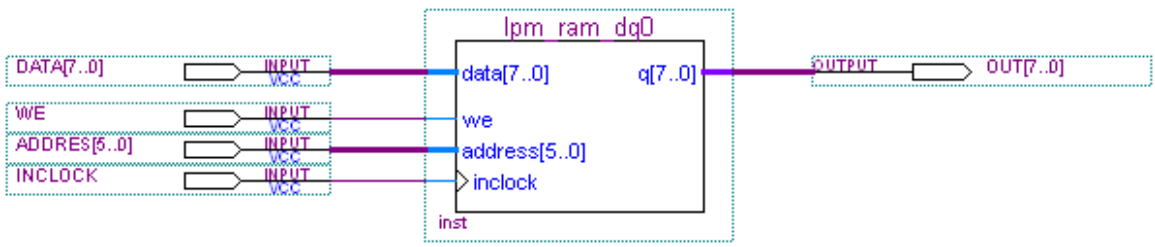


图 3-9-5 存储器电路图

7. 选择 Tools→Compiler Tool 菜单，编译源文件。
8. 编译无误后建立仿真波形文件 RAM.VWF。选择 Tools→Simulator Tool 菜单进行仿真。仿真结果如图 3-9-6 所示，分析仿真结果，验证存储器设计的正确性。

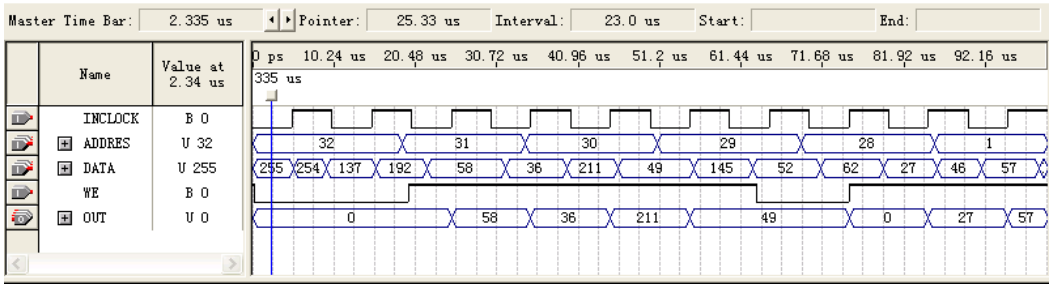


图 3-9-6 存储器仿真结果

2．FIFO 存储器实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 FIFO，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建图形设计文件，在图形编辑器窗口中双击或选择菜单“Edit→Insert Symbol”，在如图 3-9-7 所示的”Symbol 对话框”界面的 Name 中输入”lmp_fifo_dc”。

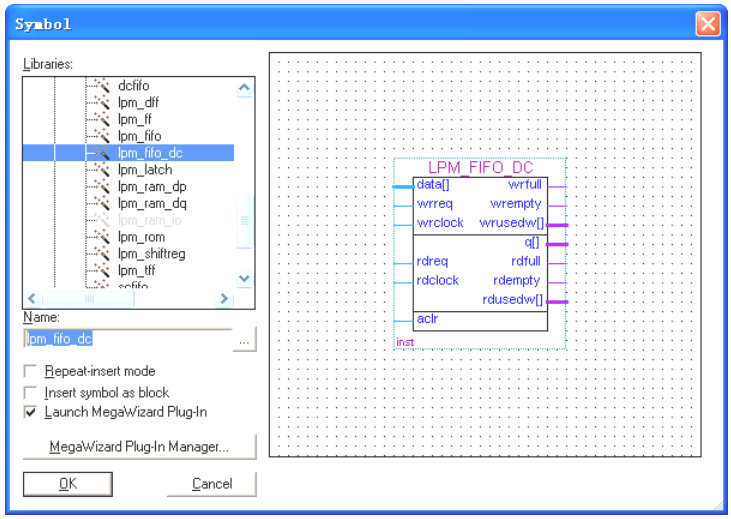


图 3-9-7 Symbol 对话框

3. 在如图 3-9-8 所示的”MegaWizard Plug-In Manager”界面中，点击”NEXT”按钮。

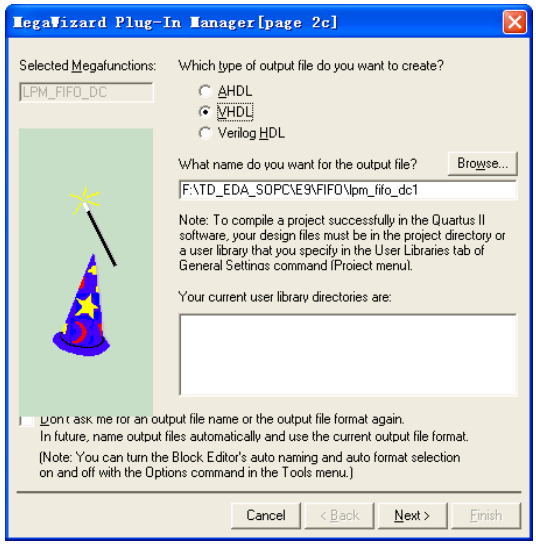


图 3-9-8　MegaWizard Plug-In Manager 界面

4. 在如图 3-9-9 所示的 FIFO 宏模块中如图设置，点击”NEXT”按钮。

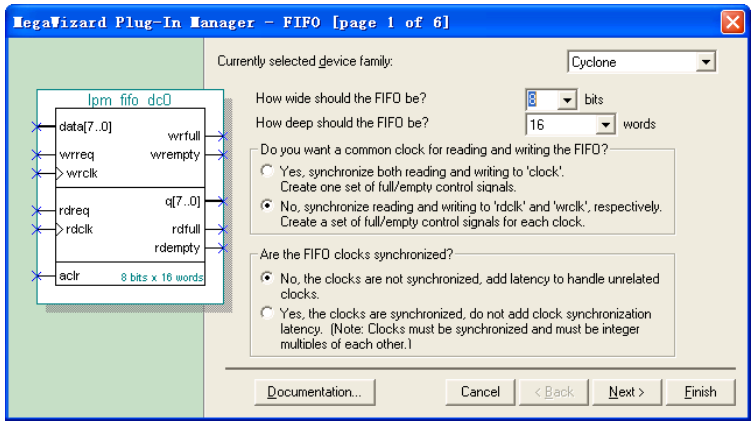


图 3-9-9　FIFO 宏模块的设置(一)

5. 在如图 3-9-10 所示的 FIFO 宏模块中如图设置。

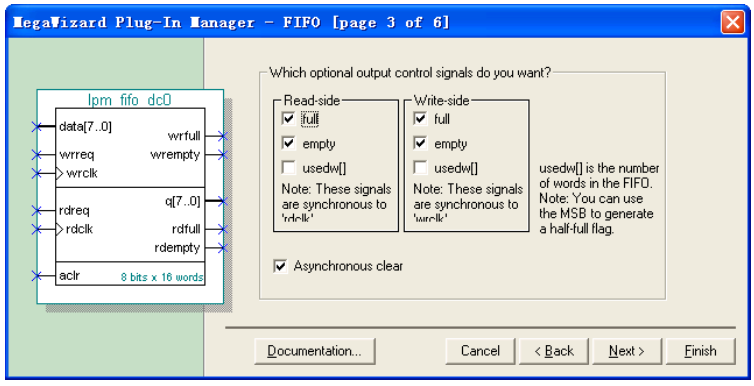


图 3-9-10　FIFO 宏模块的设置(二)

6. 一直点击”NEXT”按钮，最终在 FIFO 宏模块中点击”Finish”按钮。
7. 将 lmp_fifo_dc 模块放置到图形编辑器界面中，完成如图 3-9-11 所示的存储器电路图。

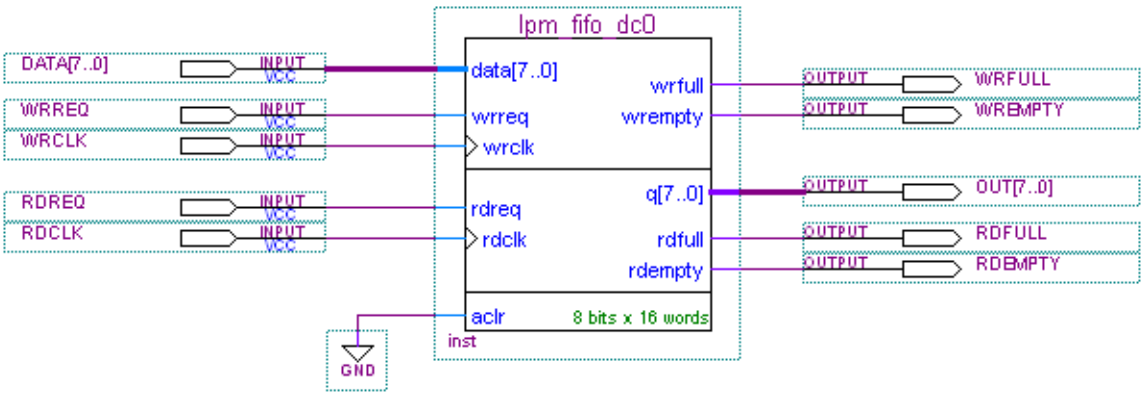


图 3-9-11 存储器电路图

8. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 FIFO.VWF。
9. 选择 Tools→Simulator Tool 菜单进行仿真。仿真结果如图 3-9-12 所示，分析仿真结果，说明 FIFO 存储器设计的正确性。

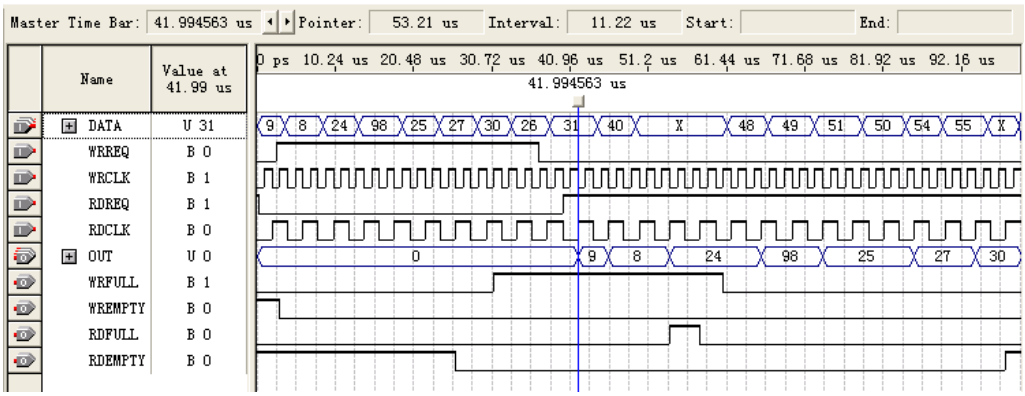


图 3-9-12 FIFO 存储器仿真结果

3.10 多路彩灯控制器实验

3.10.1 实验目的

1. 学习使用 VHDL 语言进行层次化设计的方法；
4. 掌握彩灯控制器的原理及设计方法。

3.10.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

3.10.3 实验内容

本实验的彩灯控制器 CDKZHQ 分为两大部分：时序控制电路 SHXKZH 和显示控制电路 XSKZH。时序控制电路 SHXKZH 根据输入信号 CLK_IN，CLR，CHOICE_KEY 产生符合一定要求的、供显示控制电路 XSHKZH 使用的时钟控制信号。显示控制电路 XSHKZH 则根据时序控制电路 SHXKZH 输入的时钟控制信号，输出各种花型变化的十六路彩灯工作的彩灯工作控制信号。

3.10.4 实验步骤

1. 运行 Quartus II 软件，选择 File→New Project Wizard 菜单，工程名称及顶层文件名称为 CDKZHQ，器件设置对话框中选择 Cyclone 系列 EP1C6Q240C8 芯片，建立新工程。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
3. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--时序控制器电路:SHXKZH.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY SHXKZH IS
  PORT( CHOICE_KEY : IN   STD_LOGIC;    --时钟快慢控制信号
        CLK_IN     : IN   STD_LOGIC;    --标准时钟输入
        CLR        : IN   STD_LOGIC;    --系统清零
        CLK        : OUT STD_LOGIC);    --时钟输出
END ENTITY SHXKZH ;
ARCHITECTURE   BEHAV OF SHXKZH IS
   SIGNAL   CLLK: STD_LOGIC;
   BEGIN
     PROCESS(CLK_IN,CLR,CHOICE_KEY) IS
       VARIABLE TEMP:STD_LOGIC_VECTOR(2 DOWNTO 0);
```

```
BEGIN
    IF CLR='1'THEN     --当 CLR='1'时清零，否则正常工作
        CLLK<='0';
        TEMP:="000";
    ELSIF RISING_EDGE(CLK_IN) THEN
        IF CHOICE_KEY='1' THEN
            IF TEMP="001" THEN
                TEMP:="000";
                CLLK<=NOT CLLK;
            ELSE
                TEMP:=TEMP+'1';
            END IF;
--当 CHOICE_KEY='1'时产生基准时钟频率的二分之一的时钟信号，
--否则产生基准时钟频率的八分之一的时钟信号。
        ELSE
            IF TEMP="111" THEN
                TEMP:="000";
                CLLK<=NOT CLLK;
            ELSE
                TEMP:=TEMP+'1';
            END IF;
        END IF;
    END IF;
END PROCESS;
CLK<=CLLK;
END ARCHITECTURE BEHAV;
```

4. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为 SHXKZH.VHD。
5. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
6. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--显示控制电路: XSHKZH.VHD
LIBRARY   IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY XSHKZH IS
    PORT( CLK : IN STD_LOGIC;
          CLR : IN STD_LOGIC;
          LED : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY XSHKZH;
```

```
ARCHITECTURE BEHAV OF XSHKZH IS
    TYPE STATE IS(S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,
                  S12,S13,S14,S15,S16,S17,S18,S19,S20,S21,
                  S22,S23,S24,S25,S26,S27,S28,S29,S30,S31);
    SIGNAL CURRENT_STATE,NEXT_STATE:STATE;
    SIGNAL DATA:STD_LOGIC_VECTOR(15 DOWNTO 0);
    BEGIN
    SWITCH_ORDER:PROCESS(CLK,CLR)   --主控时序进程：负责状态机的运转及在时钟驱动
                                    --下负责状态转换的进程
      BEGIN
        IF CLR='1' THEN CURRENT_STATE<=S0;
        ELSIF RISING_EDGE(CLK) THEN CURRENT_STATE<=NEXT_STATE;
        END IF;
    END PROCESS;
    CONTROL_S: PROCESS(CURRENT_STATE,DATA) --主控组合进程：根据外部输入的控制信号
     --或（和）当前状态的状态值确定下一状态的取向。即 NEXT_STATE 的取值内容以及
     --确定对外输出或对内部其它组合和时序进程输出控制信号的内容。
        BEGIN
            CASE CURRENT_STATE IS
                WHEN S0=> DATA<="1000000000000001";NEXT_STATE<=S1;
                WHEN S1=> DATA<="1100000000000011";NEXT_STATE<=S2;
                WHEN S2=> DATA<="1110000000000111";NEXT_STATE<=S3;
                WHEN S3=> DATA<="1111000000001111";NEXT_STATE<=S4;
                WHEN S4=> DATA<="1111100000011111";NEXT_STATE<=S5;
                WHEN S5=> DATA<="1111110000111111";NEXT_STATE<=S6;
                WHEN S6=> DATA<="1111111001111111";NEXT_STATE<=S7;
                WHEN S7=> DATA<="1111111111111111";NEXT_STATE<=S8;
                WHEN S8=> DATA<="0111111111111110";NEXT_STATE<=S9;
                WHEN S9=> DATA<="0011111111111100";NEXT_STATE<=S10;
                WHEN S10=>DATA<="0001111111111000";NEXT_STATE<=S11;
                WHEN S11=>DATA<="0000111111110000";NEXT_STATE<=S12;
                WHEN S12=>DATA<="0000011111100000";NEXT_STATE<=S13;
                WHEN S13=>DATA<="0000001111000000";NEXT_STATE<=S14;
                WHEN S14=>DATA<="0000000110000000";NEXT_STATE<=S15;
                WHEN S15=>DATA<="0000000000000000";NEXT_STATE<=S16;
                WHEN S16=>DATA<="0000000110000000";NEXT_STATE<=S17;
                WHEN S17=>DATA<="0000001111000000";NEXT_STATE<=S18;
                WHEN S18=>DATA<="0000011111100000";NEXT_STATE<=S19;
```

```
                WHEN S19=>DATA<="0000111111110000";NEXT_STATE<=S20;
                WHEN S20=>DATA<="0001111111111000";NEXT_STATE<=S21;
                WHEN S21=>DATA<="0011111111111100";NEXT_STATE<=S22;
                WHEN S22=>DATA<="0111111111111110";NEXT_STATE<=S23;
                WHEN S23=>DATA<="1111111111111111";NEXT_STATE<=S24;
                WHEN S24=>DATA<="1111111001111111";NEXT_STATE<=S25;
                WHEN S25=>DATA<="1111110000111111";NEXT_STATE<=S26;
                WHEN S26=>DATA<="1111100000011111";NEXT_STATE<=S27;
                WHEN S27=>DATA<="1111000000001111";NEXT_STATE<=S28;
                WHEN S28=>DATA<="1110000000000111";NEXT_STATE<=S29;
                WHEN S29=>DATA<="1100000000000011";NEXT_STATE<=S30;
                WHEN S30=>DATA<="1000000000000001";NEXT_STATE<=S31;
                WHEN S31=>DATA<="0000000000000000";NEXT_STATE<=S0;
                WHEN OTHERS=>DATA<="0000000011111111";NEXT_STATE<=S0;
            END CASE;
        END PROCESS;
        LED<=DATA;
    END ARCHITECTURE BEHAV;
```

7. 选择 File→Save As 菜单，将创建的 VHDL 设计文件保存为 XSHKZH.VHD。
8. 此实验的彩灯控制器 CDKZHQ 分为时序控制电路 SHXKZH 和显示控制电路 XSHKZH 两部分。现在需要设计顶层程序对这两个模块进行控制
9. 选择 File→New 菜单，创建 VHDL 描述语言设计文件，打开文本编辑器界面。
10. 在文本编辑器界面中编写 VHDL 程序，源程序如下：

```
--彩灯控制器 CDKZHQ.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
  ENTITY CDKZHQ IS
    PORT( CLK_IN: IN STD_LOGIC;
          CLR     : IN STD_LOGIC;
          CHOICE_KEY: IN STD_LOGIC;
          LED : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
  END ENTITY CDKZHQ;

  ARCHITECTURE BEHAV OF CDKZHQ IS
    COMPONENT SHXKZH IS
      PORT(CHOICE_KEY : IN STD_LOGIC;
           CLK_IN    : IN STD_LOGIC;
```

```
            CLR    : IN STD_LOGIC;
            CLK    : OUT STD_LOGIC);
    END COMPONENT SHXKZH;

    COMPONENT XSHKZH IS
        PORT(CLK : IN STD_LOGIC;
            CLR : IN STD_LOGIC;
            LED : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
    END COMPONENT XSHKZH;
    SIGNAL C1:STD_LOGIC;
    BEGIN
      U1:SHXKZH PORT MAP(CHOICE_KEY,CLK_IN,CLR,C1);
      U2: XSHKZH PORT MAP(C1,CLR,LED);
  END ARCHITECTURE BEHAV;
```

11. 将创建的 VHDL 设计文件保存为 CDKZHQ.VHD。
12. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件 CDKZHQ.VWF。选择 Tools→Simulator Tool 菜单进行仿真。
13. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 3-10-1 所示。

表 3-10-1 彩灯控制器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK_IN	PIN_28	LED [7]	PIN_11
CHOICE_KEY	PIN_175	LED [8]	PIN_12
CLR	PIN_182	LED [9]	PIN_13
LED [0]	PIN_2	LED [10]	PIN_14
LED [1]	PIN_3	LED [11]	PIN_15
LED [2]	PIN_4	LED [12]	PIN_16
LED [3]	PIN_5	LED [13]	PIN_17
LED [4]	PIN_6	LED [14]	PIN_18
LED [5]	PIN_7	LED [15]	PIN_19
LED [6]	PIN_8		

14. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
15. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 3-10-1 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

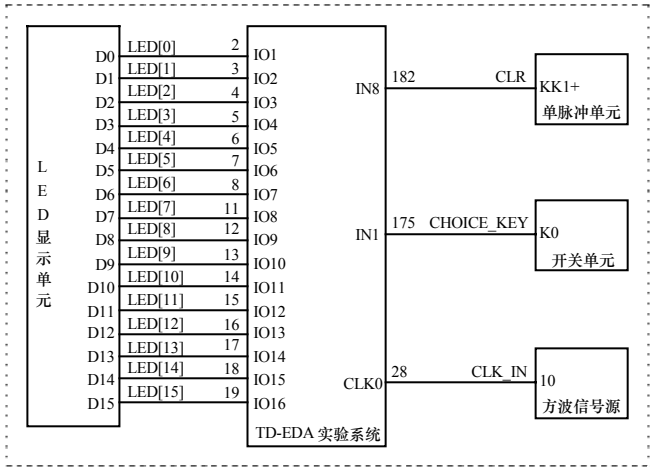


图 3-10-1 彩灯控制器实验接线图

16. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
17. 配置完成后，验证彩灯控制器的正确性。

第四章 数字系统设计综合实验

本章提供了 10 个实验项目作为数字系统设计的综合实验，通过这些实验使学生掌握复杂数字系统分析、设计的方法。本章提供的实验内容具有一定的综合性，用户需要 EDA 技术和 VHDL 开发的基础，还需要熟悉液晶显示器接口、PSII 键盘接口、VGA 显示接口、单片机开发技术，高速 AD、DA 的使用方法、存储器接口技术的知识，本章的实验内容可以作为课程设计或毕业设计的原形题目。通过本章的学习举一反三、触类旁通，用户可以利用 TD-EDA 实验系统及 SOPC 开发板提供的扩展功能完成更多复杂数字系统的开发。

4.1 键盘扫描输入实验

4.1.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 掌握矩阵式键盘输入阵列的设计方法。

4.1.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

4.1.3 实验内容

在电子、控制、信息处理等各种系统中，操作人员经常需要向系统输入数据和命令，以实现人机通信。实现人机通信最常用的输入设备是键盘。在 EDA 技术的综合应用设计中，常用的键盘输入电路有独立式键盘输入电路、矩阵式键盘输入电路和“虚拟式”键盘输入电路。

所谓矩阵式键盘输入电路，就是将水平键盘扫描线和垂直输出译码线的交叉处通过一个按键来连通，再通过一个键盘输入译码电路，将各种键盘扫描线和垂直输出译码线信号的不同组合编码转换成一个特定的输入信号值或输入信号编码。利用这种行列矩阵结构的键盘，只需 N 个行线和 M 个列线即可组成 N×M 按键。矩阵式键盘输入电路的优点是当需要键数较多时，可以节省 I/O 口线；缺点是编程相对困难。

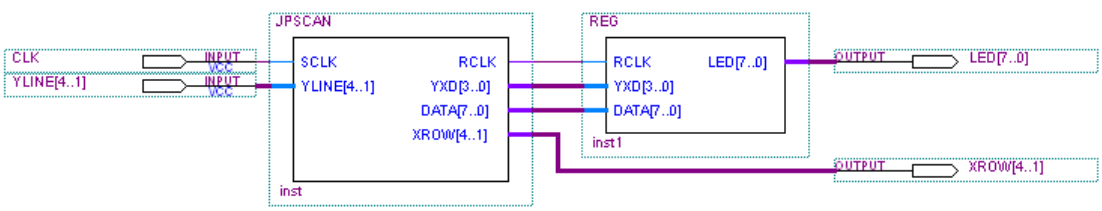


图 4-1-1 键盘扫描输入电路顶层设计电路图

本实验使用 TD-EDA 实验系统的键盘单元设计一个 4×4 矩阵键盘的扫描译码电路。此设计包括键盘扫描模块和扫描码锁存模块，原理如图 4-1-1 所示。每按下键盘阵列的一个按键立即在七段数码管上显示相应的数值。

4.1.4 实验步骤

1. 运行 Quartus II 软件，分别建立新工程，选择 File→New 菜单，创建 VHDL 描述语言设计文件，分别编写 JPSCAN.VHD、REG.VHD。
2. 扫描码锁存模块 REG 的 VHDL 源程序如下：

```
--输出锁存器 VHDL 源程序：REG.VHDL
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY REG IS
  PORT( RCLK : IN    STD_LOGIC;                                --扫描时钟
        YXD : IN    STD_LOGIC_VECTOR(3 DOWNTO 0); --Y 列消抖输入
       DATA : IN    STD_LOGIC_VECTOR(7 DOWNTO 0); --输入数据
        LED : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));--锁存数据输出
END ENTITY REG;
ARCHITECTURE BEHV OF REG IS
    SIGNAL RST : STD_LOGIC;                                    --锁存器复位清零
    SIGNAL OLDDATA : STD_LOGIC_VECTOR(7 DOWNTO 0);--锁存器旧数据
    SIGNAL NEWDATA : STD_LOGIC_VECTOR(7 DOWNTO 0);--锁存器新数据
      BEGIN
      PROCESS(RCLK)
         BEGIN
       IF RCLK'EVENT AND RCLK='1' THEN
          RST<=YXD(3) AND YXD(2) AND YXD(1) AND YXD(0);--判断是否有按键按下
       END IF;
      END PROCESS;
    PROCESS(RST) IS
      BEGIN
        IF(RST='1') THEN                          --RST=1 没有按键按下
          NEWDATA<=OLDDATA;
        ELSE
          OLDDATA<=DATA;                          --RST=0 有按键按下打入新数据
        END IF;
       LED<=NEWDATA;
    END PROCESS;
END ARCHITECTURE BEHV;
```

3. 键盘扫描模块 JPSCAN 的 VHDL 源程序如下：

```
--键盘扫描电路的 VHDL 源程序：JPSCAN.VHD
 LIBRARY  IEEE;
 USE IEEE.STD_LOGIC_1164.ALL;
 USE IEEE.STD_LOGIC_ARITH.ALL;
 USE IEEE.STD_LOGIC_UNSIGNED.ALL;

 ENTITY JPSCAN IS
 PORT(SCLK  :  IN  STD_LOGIC;                              --系统时钟:1KHz
      YLINE :  IN  STD_LOGIC_VECTOR(4 DOWNTO 1); --Y 列按键输入
      RCLK  :  OUT STD_LOGIC;                              --X 行键盘扫描时钟
      YXD   :  OUT STD_LOGIC_VECTOR(3 DOWNTO 0); --Y 列消抖输出
      DATA  :  OUT STD_LOGIC_VECTOR(7 DOWNTO 0); --数字输出
      XROW  :  OUT STD_LOGIC_VECTOR(4 DOWNTO 1));--X 行键盘扫描
 END ENTITY JPSCAN;

 ARCHITECTURE BEHV OF JPSCAN IS
   COMPONENT JPXD IS                                  --键盘消抖模块，实现键盘输入消抖
    PORT(D_IN : IN STD_LOGIC;
         CLK  : IN STD_LOGIC;
         D_OUT:OUT STD_LOGIC);
   END COMPONENT JPXD;
  SIGNAL CLK:STD_LOGIC;                                  --控制电路工作时钟:512Hz
  SIGNAL KEY_SCAN: STD_LOGIC_VECTOR(1 DOWNTO 0); --键盘扫描时钟信号
                                                         --"00-01-10-11"
  SIGNAL CLK_JPXD : STD_LOGIC;                          --去抖电路工作时钟:
  SIGNAL Y_XD : STD_LOGIC_VECTOR(3 DOWNTO 0);       --键盘列输入去抖后的寄存器
  SIGNAL X_SCAN : STD_LOGIC_VECTOR(3 DOWNTO 0);    --键盘行扫描输出寄存器
                                                         --1110-1101-1011-0111
  SIGNAL VALUE : STD_LOGIC_VECTOR(7 DOWNTO 0);     --按键译码数值寄存器
 BEGIN
   DATA<=VALUE;
   COUNTER: BLOCK IS                                    --信号扫描发生器
      SIGNAL Q     : STD_LOGIC_VECTOR(6 DOWNTO 0); --计数器实现分频
    BEGIN
      PROCESS(SCLK) IS
         BEGIN
           IF SCLK'EVENT AND SCLK='1' THEN
              Q<=Q+1;
```

```
        END IF;
        CLK<=Q(0);                          --控制电路工作时钟:512Hz 系统时钟的二分频
        CLK_JPXD<=Q(2);                     --去抖时钟信号，大约 128Hz
        KEY_SCAN<=Q(6 DOWNTO 5);    --产生键盘扫描信号 00-01-10-11,大约 16Hz
  END PROCESS;
   X_SCAN<="1110" WHEN KEY_SCAN="00" ELSE
                "1101" WHEN KEY_SCAN="01" ELSE
                "1011" WHEN KEY_SCAN="10" ELSE
                "0111" WHEN KEY_SCAN="11" ELSE
                "1111";
     XROW(4 DOWNTO 1)<=X_SCAN(3 DOWNTO 0);    --X 行扫描
END BLOCK COUNTER;
 JPXDMK: BLOCK IS                           --键盘去抖模块
     BEGIN
        U1 : JPXD PORT MAP(D_IN=>YLINE(1),D_OUT=>Y_XD(0),CLK=>CLK_JPXD);
        U2 : JPXD PORT MAP(D_IN=>YLINE(2),D_OUT=>Y_XD(1),CLK=>CLK_JPXD);
        U3 : JPXD PORT MAP(D_IN=>YLINE(3),D_OUT=>Y_XD(2),CLK=>CLK_JPXD);
        U4 : JPXD PORT MAP(D_IN=>YLINE(4),D_OUT=>Y_XD(3),CLK=>CLK_JPXD);
        YXD(3 DOWNTO 0)<=Y_XD(3 DOWNTO 0);
       RCLK<=CLK;                           --键盘扫描时钟等于控制电路工作时钟:512Hz
 END BLOCK JPXDMK;
 KEY_DECODER : BLOCK    IS                  --键盘译码模块
     SIGNAL Z:STD_LOGIC_VECTOR(5 DOWNTO 0);
 BEGIN
    PROCESS(CLK)
        BEGIN
          Z<=KEY_SCAN&Y_XD;
            IF CLK'EVENT AND CLK='1' THEN
              CASE Z IS
                 WHEN "001110" => VALUE<="00111111";     --0
                 WHEN "011110" => VALUE<="00000110" ;    --1
                 WHEN "101110" => VALUE<="01011011" ;    --2
                 WHEN "111110" => VALUE<="01001111" ;    --3
                 WHEN "001101" => VALUE<="01100110";     --4
                WHEN "011101" => VALUE<="01101101";     --5
                 WHEN "101101" => VALUE<="01111101";     --6
                WHEN "111101" => VALUE<="00000111";     --7
                 WHEN "001011" => VALUE<="01111111";     --8
                WHEN "011011" => VALUE<="01101111";     --9
```

```
                WHEN "101011" => VALUE<="01110111";   --A
                WHEN "111011" => VALUE<="01111100";   --B
                WHEN "000111" => VALUE<="00111001";   --C
                WHEN "010111" => VALUE<="01011110";   --D
                WHEN "100111" => VALUE<="01111001";   --E
                WHEN "110111" => VALUE<="01110001";   --F
                WHEN OTHERS   =>   VALUE<="00000000";   --OTHER
            END CASE;
        END IF;
    END PROCESS;
  END BLOCK KEY_DECODER;
END ARCHITECTURE BEHV;
```

4. 上述程序中键盘消抖模块 JPXD 的 VHDL 源程序如下:

--键盘输入消抖电路的 VHDL 源程序。 JPXD.VHD

```
LIBRARY   IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY JPXD IS
  PORT(D_IN : IN STD_LOGIC;
       CLK   : IN STD_LOGIC;
       D_OUT:OUT STD_LOGIC);
END ENTITY JPXD ;

ARCHITECTURE BEHV OF JPXD IS

  COMPONENT DCFQ IS
    PORT(CLK: IN STD_LOGIC;     --时钟信号
         CLRN: IN STD_LOGIC;      --清零信号
         PRN: IN STD_LOGIC;       --预置信号
         D: IN STD_LOGIC;
         Q: OUT STD_LOGIC);
  END COMPONENT DCFQ;

    SIGNAL VCC,INV_D:STD_LOGIC;
    SIGNAL Q0,Q1:STD_LOGIC;
    SIGNAL D1,D0:STD_LOGIC;
    BEGIN
        VCC<='1';
```

```
    INV_D<= NOT D_IN;
  UMN1:DCFQ PORT MAP(CLK=>CLK,CLRN=>INV_D,PRN=>VCC,D=>VCC,Q=>Q0);
  UMN2:DCFQ PORT MAP(CLK=>CLK,CLRN=>Q0,PRN=>VCC,D=>VCC,Q=>Q1);
    PROCESS(CLK)
      BEGIN
        IF CLK'EVENT AND CLK='1' THEN
            D0<=NOT Q1;
            D1<=D0;
        END IF;
   END PROCESS;
     D_OUT <= NOT(D1 AND NOT D0);
END ARCHITECTURE BEHV;
```

5. 编译源文件，编译无误后分别对上述 VHDL 文件进行仿真，验证其逻辑功能正确后，选择 File→Create/Update →Create Symbol File for Current File 菜单，分别生成符号文件.BSF。
6. 建立新工程，工程名称及顶层文件名称为 JIANPAN，选择 File→New 菜单，创建图形设计文件。如图 4-1-1 所示，完成键盘扫描输入电路的设计。将创建的图形设计文件保存为 JIANPAN.BDF 作为整个设计的顶层文件。
7. 选择 Tools→Compiler Tool 菜单，编译文件。
8. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-1-1 所示。

表 4-1-1 键盘扫描输入电路实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	XROW [1]	PIN_12
LED [0]	PIN_2	XROW [2]	PIN_13
LED [1]	PIN_3	XROW [3]	PIN_14
LED [2]	PIN_4	XROW [4]	PIN_15
LED [3]	PIN_5	YLINE [1]	PIN_175
LED [4]	PIN_6	YLINE [2]	PIN_176
LED [5]	PIN_7	YLINE [3]	PIN_177
LED [6]	PIN_8	YLINE [4]	PIN_178
LED [7]	PIN_11		

9. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 JIANPAN.SOF 文件。
10. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-1-2 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

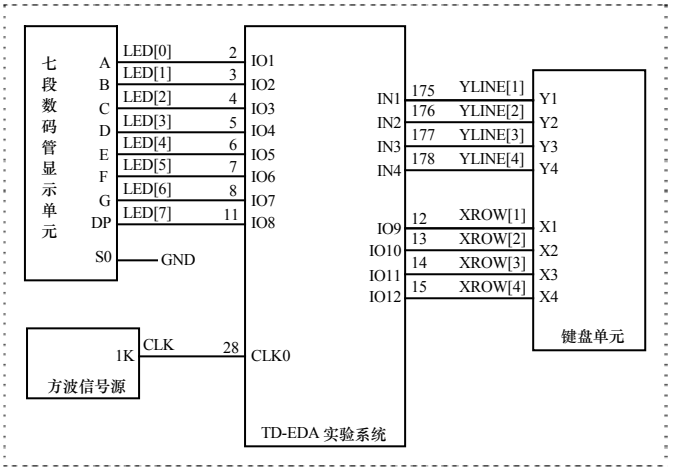


图 4-1-2 键盘扫描输入实验接线图

11. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
12. 配置完成后，观察实验现象验证设计的正确性。

4.2 扫描数码显示器实验

4.2.1 实验目的

3. 学习状态机的原理及使用 VHDL 语言设计的方法；
4. 学习复杂数字系统的设计方法；
5. 掌握动态扫描数码显示器的设计方法。

4.2.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

4.2.3 实验内容

状态机是一类很重要的时序电路，是许多数字电路的核心部件。根据状态机的输出方式可以分为 Mealy 型和 Moore 型两种状态机。输出与状态有关而与输入无关的状态机类型称为 Moore 型状态机。输出与状态及输入皆有关系的状态机称为 Mealy 型状态机。状态机通常包含：说明部分、主控时序进程、主控组合进程、辅助进程几个部分。利用状态机进行设计的步骤如下：

1. 分析设计要求，列出状态机所有可能的状态，并对每一个状态进行状态编码；
2. 根据状态转移关系和输出函数画出状态转移图；
3. 由状态转移图，用状态机语句描述状态机。

使数码管动态扫描显示的方式主要是为了节省 I/O 管脚和内部逻辑资源，它利用人的视觉暂留现象，将 6 位数码管分别循环选通，配合传送相应的要显示的数据，只要扫描的速度足够快，就可以使人的视觉感到好像是 6 位数码管在同时显示。一般扫描频率使用 1KHz 就可以了。

本实验设计一个可以使 6 位数码管动态刷新显示的扫描电路。分析系统的要求可知此设计需要包括 6 进制计数器、BCD 译码器、数据选择多路开关等多个小单元模块。实验需要设计一个模块来为 6 个数码块提供要显示的数据，设计一个六位数 123456 从左向右移动的方式，直到最高一位移出最右边数码块后，最低位 6 再从最左边数码块移进，从而实现循环移动。

4.2.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 SCANLED。

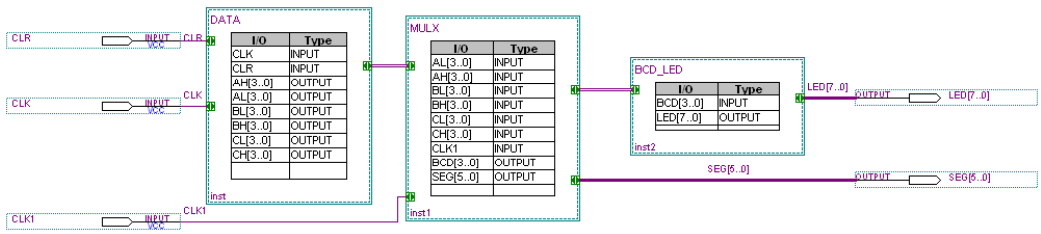


图 4-2-1 数码扫描显示器顶层设计电路图

2. 选择 File→New 菜单，创建图形设计文件，在图形编辑器界面中选择 Block Tool 工具按钮，分别新建 DATA、MULX、BCD_LED 子模块，完成模块的定义及模块之间的连接，完成如图 4-2-1 所示的数码扫描显示器顶层设计电路图。
3. 将创建的图形设计文件保存为 SCANLED.BDF 作为整个设计的顶层文件。
4. 右键单击 DATA 模块，在弹出的菜单中选择“Create Design File from Selected Block”，生成名称为 DATA.VHD 的 VHDL 设计文件。
5. 在文本编辑器界面中编写程序，文件保存为 DATA.VHD。源程序如下：

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY DATA IS
  PORT(CLK : IN STD_LOGIC;
       CLR : IN STD_LOGIC;
       AH : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       AL : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       BL : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       BH : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       CL : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
       CH : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END DATA;
ARCHITECTURE DATA_BEHAV OF DATA IS
   TYPE STATE IS(ST0,ST1,ST2,ST3,ST4,ST5,ST6,
                 ST7,ST8,ST9,ST10,ST11);
   SIGNAL CURRENT_STATE,NEXT_STATE:STATE;
   SIGNAL DAL,DAH,DBL,DBH,DCL,DCH:STD_LOGIC_VECTOR(3 DOWNTO 0);
 BEGIN
    REG: PROCESS(CLR,CLK)
      BEGIN
        IF CLR='1' THEN CURRENT_STATE<=ST0;
        ELSIF CLK='1' AND CLK'EVENT THEN
              CURRENT_STATE<=NEXT_STATE;
        END IF;
    END PROCESS;
    COM: PROCESS(CURRENT_STATE,DAL,DAH,DBL,DBH,DCL,DCH)
       BEGIN
         CASE CURRENT_STATE IS
           WHEN ST0   => DAL<="0110"; DAH<="0101"; DBL<="0100"; DBH<="0011";
                         DCL<="0010"; DCH<="0001"; NEXT_STATE<=ST1;
           WHEN ST1   => DAL<="0101"; DAH<="0100"; DBL<="0011"; DBH<="0010";
                         DCL<="0001"; DCH<="1111"; NEXT_STATE<=ST2;
```

```
                WHEN ST2   => DAL<="0100"; DAH<="0011"; DBL<="0010"; DBH<="0001";
                              DCL<="1111"; DCH<="1111"; NEXT_STATE<=ST3;
                WHEN ST3   => DAL<="0011"; DAH<="0010"; DBL<="0001";DBH<="1111";
                              DCL<="1111"; DCH<="1111"; NEXT_STATE<=ST4;
                WHEN ST4   => DAL<="0010"; DAH<="0001"; DBL<="1111"; DBH<="1111";
                              DCL<="1111"; DCH<="1111"; NEXT_STATE<=ST5;
                WHEN ST5   => DAL<="0001"; DAH<="1111"; DBL<="1111"; DBH<="1111";
                              DCL<="1111"; DCH<="1111"; NEXT_STATE<=ST6;
                WHEN ST6   => DAL<="1111"; DAH<="1111"; DBL<="1111"; DBH<="1111";
                              DCL<="1111"; DCH<="1111"; NEXT_STATE<=ST7;
                WHEN ST7   => DAL<="1111"; DAH<="1111"; DBL<="1111"; DBH<="1111";
                              DCL<="1111"; DCH<="0110"; NEXT_STATE<=ST8;
                WHEN ST8   => DAL<="1111"; DAH<="1111"; DBL<="1111"; DBH<="1111";
                              DCL<="0110"; DCH<="0101"; NEXT_STATE<=ST9;
                WHEN ST9   => DAL<="1111"; DAH<="1111"; DBL<="1111"; DBH<="0110";
                              DCL<="0101"; DCH<="0100"; NEXT_STATE<=ST10;
                WHEN ST10 => DAL<="1111"; DAH<="1111"; DBL<="0110"; DBH<="0101";
                              DCL<="0100"; DCH<="0011"; NEXT_STATE<=ST11;
                WHEN ST11   => DAL<="1111"; DAH<="0110"; DBL<="0101"; DBH<="0100";
                              DCL<="0011"; DCH<="0010"; NEXT_STATE<=ST0;
                WHEN OTHERS => DAL<="0000"; DAH<="0000"; DBL<="0000"; DBH<="0000";
                              DCL<="0000"; DCH<="0000"; NEXT_STATE<=ST0;
            END CASE;
        END PROCESS;
         AL<=DAL; AH<=DAH;
         BL<=DBL; BH<=DBH;
         CL<=DCL; CH<=DCH;
    END DATA_BEHAV ;
```

6. 右键单击 MULX 模块，在弹出的菜单中选择“Create Design File from Selected Block”，生成名称为 MULX.VHD 的 VHDL 设计文件。
7. 在文本编辑器界面中编写程序，文件保存为 MULX.VHD。源程序如下：

```
LIBRARY   IEEE;
USE   IEEE.STD_LOGIC_1164.ALL;
ENTITY MULX IS
   PORT(      AL : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
              AH : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
              BL : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
              BH : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
                CL : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                CH : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                CLK1 : IN STD_LOGIC;
                BCD : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                SEG : OUT STD_LOGIC_VECTOR(5 DOWNTO 0));
END MULX;
ARCHITECTURE MULX_BEHAV OF MULX IS
    TYPE STATE IS (ST0,ST1,ST2,ST3,ST4,ST5);
    SIGNAL CURRENT_STATE,NEXT_STATE: STATE;
  BEGIN
    SS1: PROCESS(CLK1)
      BEGIN
        IF(CLK1='1' AND CLK1'EVENT) THEN
            CURRENT_STATE<=NEXT_STATE;
        END IF;
    END PROCESS;
    SS2: PROCESS(CURRENT_STATE)
       BEGIN
          CASE CURRENT_STATE IS
             WHEN ST0 => BCD<=AL;SEG<="011111";NEXT_STATE<=ST1;
             WHEN ST1 => BCD<=AH;SEG<="101111";NEXT_STATE<=ST2;
             WHEN ST2 => BCD<=BL;SEG<="110111";NEXT_STATE<=ST3;
             WHEN ST3 => BCD<=BH;SEG<="111011";NEXT_STATE<=ST4;
             WHEN ST4 => BCD<=CL;SEG<="111101";NEXT_STATE<=ST5;
             WHEN ST5 => BCD<=CH;SEG<="111110";NEXT_STATE<=ST0;
          END CASE;
    END PROCESS;
END MULX_BEHAV ;
```

8. 右键单击 BCD_LED 模块，在弹出的菜单中选择“Create Design File from Selected Block”，生成名称为 BCD_LED.VHD 的 VHDL 设计文件。
9. 在文本编辑器界面中编写程序，文件保存为 BCD_LED.VHD。源程序如下：

```
LIBRARY  IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
ENTITY BCD_LED IS
   PORT
   (    BCD : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        LED : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END BCD_LED;
```

```
ARCHITECTURE BCD_LED_BEHV OF BCD_LED IS
BEGIN
   PROCESS(BCD)
      BEGIN
         CASE BCD IS              --Dpgfedcba
           WHEN "0000"   => LED<="00111111";
           WHEN "0001"   => LED<="00000110";
           WHEN "0010"   => LED<="01011011";
           WHEN "0011"   => LED<="01001111";
           WHEN "0100"   => LED<="01100110";
           WHEN "0101"   => LED<="01101101";
           WHEN "0110"   => LED<="01111101";
           WHEN "0111"   => LED<="00000111";
           WHEN "1000"   => LED<="01111111";
           WHEN "1001"   => LED<="01101111";
           WHEN OTHERS   => LED<="00000000";
         END CASE;
   END PROCESS;
END BCD_LED_BEHV;
```

10. 选择 Tools→Compiler Tool 菜单，编译 SCANLED.BDF 源文件。编译无误后建立仿真波形文件 SCANLED.VWF。
11. 选择 Tools→Simulator Tool 菜单进行仿真。分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-2-1 所示。
12. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。

表 4-2-1 数码扫描显示器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	LED[7]	PIN_11
CLK1	PIN_29	SEG[0]	PIN_12
LED[0]	PIN_2	SEG[1]	PIN_13
LED[1]	PIN_3	SEG[2]	PIN_14
LED[2]	PIN_4	SEG[3]	PIN_15
LED[3]	PIN_5	SEG[4]	PIN_16
LED[4]	PIN_6	SEG[5]	PIN_17
LED[5]	PIN_7	CLR	PIN_194
LED[6]	PIN_8		

13. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-2-2 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

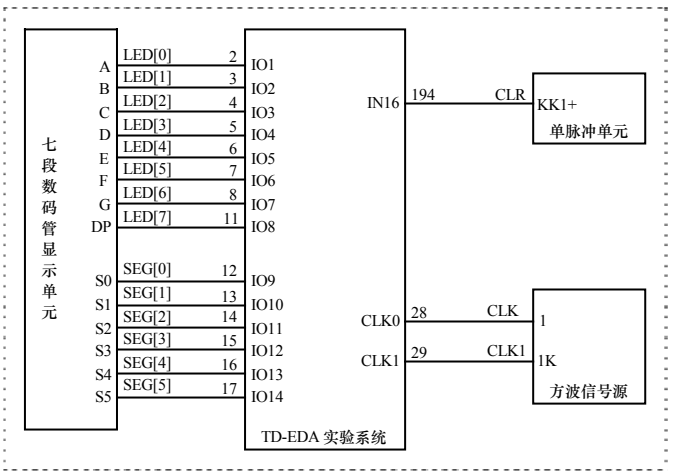


图 4-2-2 数码扫描显示器实验接线图

14. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
15. 配置完成后，验证设计的正确性。

4.2.5 实验扩展

本实验实现了六个七段数码管显示器的动态扫描，设计实现八个七段数码管显示器的动态扫描。

4.3 点阵显示实验

4.3.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 学习点阵 LED 的基本结构；
6. 掌握点阵显示控制的设计方法。

4.3.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

4.3.3 实验内容

点阵单元由四个 8×8 点阵器件构成 16×16 的 LED 点阵。其中 R1~R16 为行控制信号线，L1~L16 为列控制信号。给某行低电平、某列高电平，则对应的 LED 点亮，如使 R1 为'0'，L1 为'1'，则左上角的 LED 点亮。本实验使 16×16LED 点阵由大到小循环显示符号“口”。整个系统由四个单元电路组成：16 进制计数器单元 COUNT16：对输入时钟进行 16 进制计数，产生扫描状态信号。分频器单元 FENPIN：对输入时钟进行分频，实现扫描信号时钟。行控制单元 ROWCON：根据计数器的输出，产生行控制扫描状态。列控制单元 LINECON：产生列控制信号，显示不同的字符。原理图如图 4-3-1 所示。

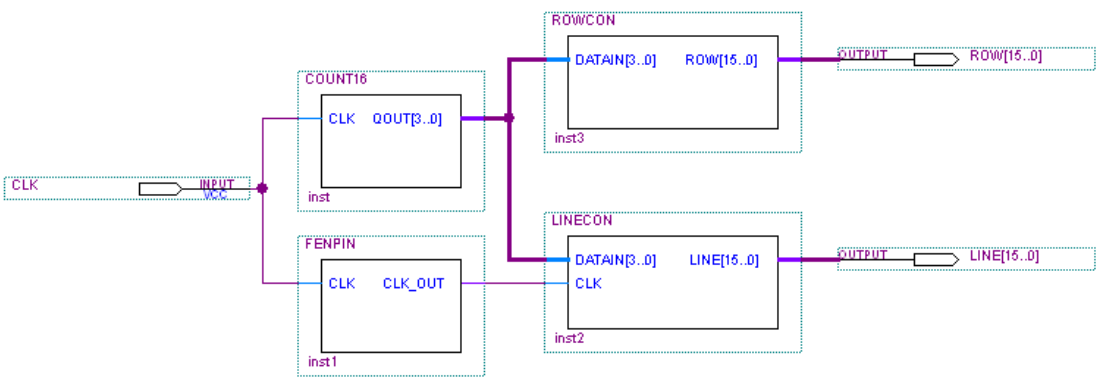


图 4-3-1 点阵显示实验顶层设计电路图

4.3.4 实验步骤

1. 运行 Quartus II 软件，分别建立新工程，选择 File→New 菜单，创建 VHDL 描述语言设计文件，分别编写 COUNT16.VHD、FENPIN.VHD、ROWCON.VHD、LINECON.VHD。根据前述实验内容自行设计 COUNT16 单元，完成其功能。
2. 分频器单元 FENPIN 的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY FENPIN IS
  PORT(CLK : IN STD_LOGIC;
       CLK_OUT : OUT STD_LOGIC);
END ENTITY FENPIN;

ARCHITECTURE BEHV OF FENPIN IS
  CONSTANT DIVIDE_PERIOD : INTEGER:=2048;
  --将 1KHz 的信号变为 0.5Hz，分频常数为 1K/2048=0.5Hz
  BEGIN
  DIVIDE_CLK:PROCESS(CLK) IS
    VARIABLE CNT:INTEGER RANGE 0 TO 2047;
   BEGIN
     IF(CLK'EVENT AND CLK='1') THEN
        IF(CNT<(DIVIDE_PERIOD/2)) THEN
          CLK_OUT<='1'; --前 1024 个周期输出为高电平
          CNT:=CNT+1;
        ELSIF(CNT<(DIVIDE_PERIOD-1)) THEN
          CLK_OUT<='0';--后 1024 个周期输出为低电平
          CNT:=CNT+1;
        ELSE
          CNT:=0;
        END IF;
     END IF;
  END PROCESS DIVIDE_CLK;
END ARCHITECTURE BEHV;
```

3. 行控制单元 ROWCON 的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ROWCON IS
PORT(DATAIN : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     ROW : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY ROWCON;
```

```
ARCHITECTURE BEHV OF ROWCON IS
 SIGNAL HANG : STD_LOGIC_VECTOR(15 DOWNTO 0);
 BEGIN
  PROCESS(DATAIN)
    BEGIN
      CASE DATAIN IS
        WHEN "0000"=> ROW<="0111111111111111";
        WHEN "0001"=> ROW<="1011111111111111";
        WHEN "0010"=> ROW<="1101111111111111";
        WHEN "0011"=> ROW<="1110111111111111";
        WHEN "0100"=> ROW<="1111011111111111";
        WHEN "0101"=> ROW<="1111101111111111";
        WHEN "0110"=> ROW<="1111110111111111";
        WHEN "0111"=> ROW<="1111111011111111";
        WHEN "1000"=> ROW<="1111111101111111";
        WHEN "1001"=> ROW<="1111111110111111";
        WHEN "1010"=> ROW<="1111111111011111";
        WHEN "1011"=> ROW<="1111111111101111";
        WHEN "1100"=> ROW<="1111111111110111";
        WHEN "1101"=> ROW<="1111111111111011";
        WHEN "1110"=> ROW<="1111111111111101";
        WHEN "1111"=> ROW<="1111111111111110";
        WHEN OTHERS=> ROW<="1111111111111111";
      END CASE;
   END PROCESS;
END ARCHITECTURE BEHV;
```

4. 列控制单元 LINECON 的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY LINECON IS
  PORT(DATAIN : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
       CLK : IN STD_LOGIC;
       LINE   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0));
END ENTITY LINECON;
```

```
ARCHITECTURE BEHV OF LINECON IS
TYPE STATE IS (ST0,ST1,ST2,ST3,ST4,ST5,ST6,ST7);
        SIGNAL CURRENT_STATE,NEXT_STATE:STATE;
   BEGIN
 REG:PROCESS(CLK)
     BEGIN
      IF CLK'EVENT AND CLK='1' THEN
        CURRENT_STATE<=NEXT_STATE;
      END IF;
     END PROCESS;
 COM: PROCESS(DATAIN,CURRENT_STATE)
      BEGIN
       CASE CURRENT_STATE IS
          WHEN ST0=>
          IF      DATAIN="0000" THEN LINE<="1111111111111111";NEXT_STATE<=ST1;
          ELSIF DATAIN="0001" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="0010" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="0011" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="0100" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="0101" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="0110" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="0111" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1000" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1001" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1010" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1011" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1100" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1101" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1110" THEN LINE<="1000000000000001";NEXT_STATE<=ST1;
          ELSIF DATAIN="1111" THEN LINE<="1111111111111111";NEXT_STATE<=ST1;
          END IF;
         WHEN ST1=>
          IF      DATAIN="0000" THEN LINE<="0000000000000000";NEXT_STATE<=ST2;
          ELSIF DATAIN="0001" THEN LINE<="0111111111111110";NEXT_STATE<=ST2;
          ELSIF DATAIN="0010" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
          ELSIF DATAIN="0011" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
          ELSIF DATAIN="0100" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
          ELSIF DATAIN="0101" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
          ELSIF DATAIN="0110" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
```

```
ELSIF DATAIN="0111" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
ELSIF DATAIN="1000" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
ELSIF DATAIN="1001" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
ELSIF DATAIN="1010" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
ELSIF DATAIN="1011" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
ELSIF DATAIN="1100" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
ELSIF DATAIN="1101" THEN LINE<="0100000000000010";NEXT_STATE<=ST2;
ELSIF DATAIN="1110" THEN LINE<="0111111111111110"; NEXT_STATE<=ST2;
ELSIF DATAIN="1111" THEN LINE<="0000000000000000";NEXT_STATE<=ST2;
END IF;
WHEN ST2=>
IF      DATAIN="0000" THEN LINE<="0000000000000000";NEXT_STATE<=ST3;
ELSIF DATAIN="0001" THEN LINE<="0000000000000000";NEXT_STATE<=ST3;
ELSIF DATAIN="0010" THEN LINE<="0011111111111100";NEXT_STATE<=ST3;
ELSIF DATAIN="0011" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="0100" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="0101" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="0110" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="0111" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="1000" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="1001" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="1010" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="1011" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="1100" THEN LINE<="0010000000000100";NEXT_STATE<=ST3;
ELSIF DATAIN="1101" THEN LINE<="0011111111111100";NEXT_STATE<=ST3;
ELSIF DATAIN="1110" THEN LINE<="0000000000000000";NEXT_STATE<=ST3;
ELSIF DATAIN="1111" THEN LINE<="0000000000000000";NEXT_STATE<=ST3;
END IF;
WHEN ST3=>
IF      DATAIN="0000" THEN LINE<="0000000000000000";NEXT_STATE<=ST4;
ELSIF DATAIN="0001" THEN LINE<="0000000000000000";NEXT_STATE<=ST4;
ELSIF DATAIN="0010" THEN LINE<="0000000000000000";NEXT_STATE<=ST4;
ELSIF DATAIN="0011" THEN LINE<="0001111111111000";NEXT_STATE<=ST4;
ELSIF DATAIN="0100" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
ELSIF DATAIN="0101" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
ELSIF DATAIN="0110" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
ELSIF DATAIN="0111" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
ELSIF DATAIN="1000" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
ELSIF DATAIN="1001" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
```

```
ELSIF DATAIN="1010" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
ELSIF DATAIN="1011" THEN LINE<="0001000000001000";NEXT_STATE<=ST4;
ELSIF DATAIN="1100" THEN LINE<="0001111111111000";NEXT_STATE<=ST4;
ELSIF DATAIN="1101" THEN LINE<="0000000000000000";NEXT_STATE<=ST4;
ELSIF DATAIN="1110" THEN LINE<="0000000000000000";NEXT_STATE<=ST4;
ELSIF DATAIN="1111" THEN LINE<="0000000000000000";NEXT_STATE<=ST4;
END IF;
WHEN ST4=>
IF      DATAIN="0000" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
ELSIF DATAIN="0001" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
ELSIF DATAIN="0010" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
ELSIF DATAIN="0011" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
ELSIF DATAIN="0100" THEN LINE<="0000111111110000";NEXT_STATE<=ST5;
ELSIF DATAIN="0101" THEN LINE<="0000100000010000";NEXT_STATE<=ST5;
ELSIF DATAIN="0110" THEN LINE<="0000100000010000";NEXT_STATE<=ST5;
ELSIF DATAIN="0111" THEN LINE<="0000100000010000";NEXT_STATE<=ST5;
ELSIF DATAIN="1000" THEN LINE<="0000100000010000";NEXT_STATE<=ST5;
ELSIF DATAIN="1001" THEN LINE<="0000100000010000";NEXT_STATE<=ST5;
ELSIF DATAIN="1010" THEN LINE<="0000100000010000";NEXT_STATE<=ST5;
ELSIF DATAIN="1011" THEN LINE<="0000111111110000";NEXT_STATE<=ST5;
ELSIF DATAIN="1100" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
ELSIF DATAIN="1101" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
ELSIF DATAIN="1110" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
ELSIF DATAIN="1111" THEN LINE<="0000000000000000";NEXT_STATE<=ST5;
END IF;
WHEN ST5=>
IF      DATAIN="0000" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="0001" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="0010" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="0011" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="0100" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="0101" THEN LINE<="0000011111100000";NEXT_STATE<=ST6;
ELSIF DATAIN="0110" THEN LINE<="0000010000100000";NEXT_STATE<=ST6;
ELSIF DATAIN="0111" THEN LINE<="0000010000100000";NEXT_STATE<=ST6;
ELSIF DATAIN="1000" THEN LINE<="0000010000100000";NEXT_STATE<=ST6;
ELSIF DATAIN="1001" THEN LINE<="0000010000100000";NEXT_STATE<=ST6;
ELSIF DATAIN="1010" THEN LINE<="0000011111100000";NEXT_STATE<=ST6;
ELSIF DATAIN="1011" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="1100" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
```

```
ELSIF DATAIN="1101" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="1110" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
ELSIF DATAIN="1111" THEN LINE<="0000000000000000";NEXT_STATE<=ST6;
END IF;
WHEN ST6=>
IF      DATAIN="0000" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="0001" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="0010" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="0011" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="0100" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="0101" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="0110" THEN LINE<="0000001111000000";NEXT_STATE<=ST7;
ELSIF DATAIN="0111" THEN LINE<="0000001001000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1000" THEN LINE<="0000001001000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1001" THEN LINE<="0000001111000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1010" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1011" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1100" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1101" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1110" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
ELSIF DATAIN="1111" THEN LINE<="0000000000000000";NEXT_STATE<=ST7;
END IF;
WHEN ST7=>
IF      DATAIN="0000" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="0001" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="0010" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="0011" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="0100" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="0101" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="0110" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="0111" THEN LINE<="0000000110000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1000" THEN LINE<="0000000110000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1001" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1010" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1011" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1100" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1101" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1110" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
ELSIF DATAIN="1111" THEN LINE<="0000000000000000";NEXT_STATE<=ST0;
```

```
        END IF;
      END CASE;
    END PROCESS;
END ARCHITECTURE BEHV;
```

5. 编译源文件，编译无误后分别对上述 VHDL 文件进行仿真，验证其逻辑功能正确后，选择 File→Create/Update →Create Symbol File for Current File 菜单，分别生成符号文件.BSF。
6. 建立新工程，工程名称及顶层文件名称为 DIANZHEN，选择 File→New 菜单，创建图形设计文件。如图 4-3-1 所示，完成点阵显示的设计。将创建的图形设计文件保存为 DIANZHEN.BDF 作为整个设计的顶层文件。
7. 选择 Tools→Compiler Tool 菜单，编译 DIANZHEN.BDF 文件。
8. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-3-1 所示。

表 4-3-1 点阵显示实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	ROW [0]	PIN_20
LINE [0]	PIN_2	ROW [1]	PIN_21
LINE [1]	PIN_3	ROW [2]	PIN_23
LINE [2]	PIN_4	ROW [3]	PIN_41
LINE [3]	PIN_5	ROW [4]	PIN_42
LINE [4]	PIN_6	ROW [5]	PIN_43
LINE [5]	PIN_7	ROW [6]	PIN_44
LINE [6]	PIN_8	ROW [7]	PIN_45
LINE [7]	PIN_11	ROW [8]	PIN_46
LINE [8]	PIN_12	ROW [9]	PIN_47
LINE [9]	PIN_13	ROW [10]	PIN_48
LINE [10]	PIN_14	ROW [11]	PIN_49
LINE [11]	PIN_15	ROW [12]	PIN_50
LINE [12]	PIN_16	ROW [13]	PIN_53
LINE [13]	PIN_17	ROW [14]	PIN_54
LINE [14]	PIN_18	ROW [15]	PIN_55
LINE [15]	PIN_19		

9. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 DIANZHEN.SOF 文件。
10. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-3-2 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

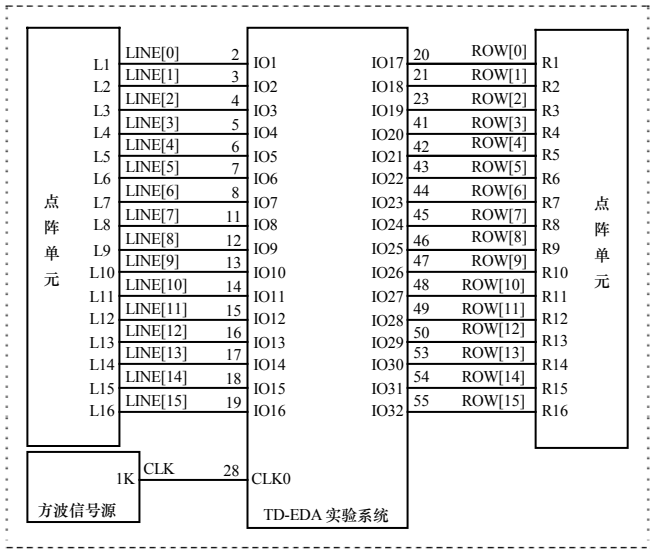


图 4-3-2 点阵显示实验接线图

11. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
12. 配置完成后，观察实验现象验证设计的正确性。

4.3.5 实验扩展

本实验实现了 16×16 点阵的显示。为了满足显示汉字的需要，设计一个显示汉字的点阵显示控制电路。

4.4 交通灯控制实验

4.4.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 掌握交通灯的原理及设计方法。

4.4.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

4.4.3 实验内容

本实验要求设计一个由一条主干道和一条支干道的汇合点形成的十字路口的交通灯控制器，具体要求如下：

(1) 主、支干道各设有一个绿、黄、红指示灯，两个显示数码管。

(2) 主干道处于长允许通行状态，而支干道有车来时才允许通行。当主干道允许通行亮绿灯时，支干道亮红灯。而支干道允许通行亮绿灯时，主干道亮红灯。

(3) 当主干道、支干道均有车时，两者交替允许通行，主干道每次通行 45 秒，支干道每次通行 25 秒，在每次由绿灯向红灯转换的过程中，要亮 5 秒的黄灯作为过渡，并进行减计时显示。

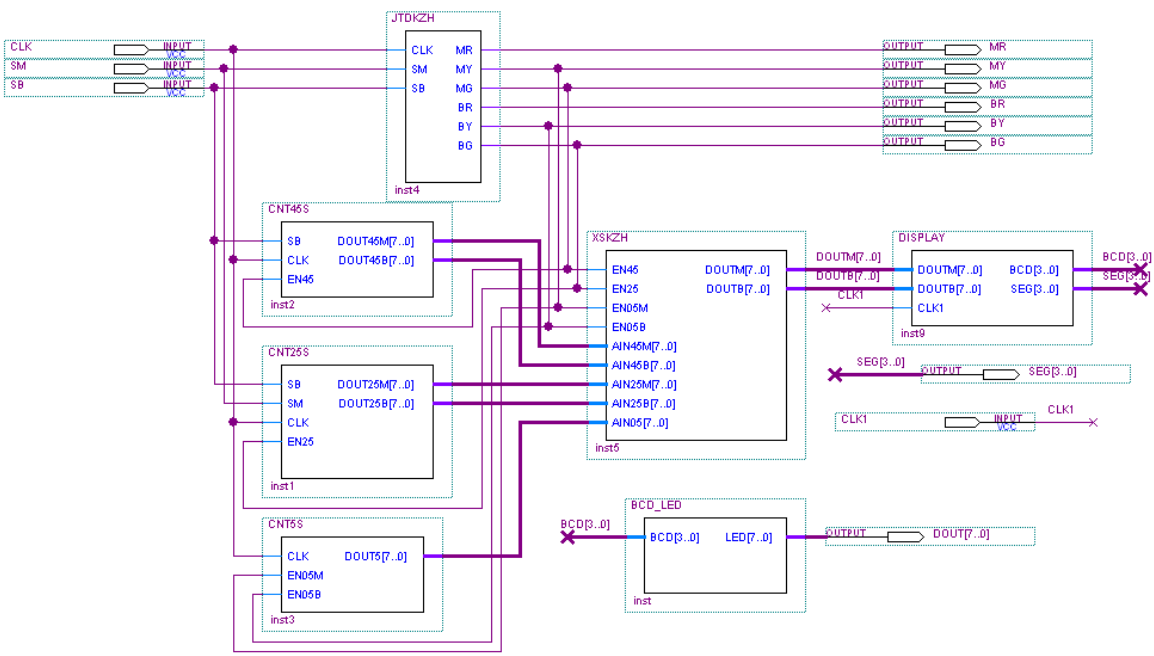


图 4-4-1 交通灯控制顶层设计电路图

交通灯控制器由单片 CPLD/FPGA 来实现，分析设计要求，整个系统由七个单元电路组成，顶层原理图如图 4-4-1 所示，图中的七个单元电路分别为：

交通灯控制器单元 JTDKZH：根据主干道、支干道输入信号 SM、SB 及时钟信号 CLK，发出主、支干道指示灯的控制信号，同时向各个定时单元 CNT45S、CNT25S、CNT5S，显示控制单元 XSKZH 发出使能控制信号 EN45、EN25、EN05M、EN05B。

定时单元 CNT45S、CNT25S、CNT5S 分别实现 45 秒、25 秒、5 秒钟的定时，根据 SM、SB、CLK 及 JTDKZH 单元发出的相关使能控制信号 EN45、EN25、EN05M、EN05B 按要求进行定时，并将其输出传送到显示控制单元 XSKZH。

显示控制单元 XSKZH：根据交通灯控制器单元 JTDKZH 发出的有关使能控制信号 EN45、EN25、EN05M、EN05B 选择定时单元 CNT45S、CNT25S、CNT5S 的输出传送到显示单元 DISPLAY。

显示单元 DISPLAY：根据显示控制单元 XSKZH 发出的数据，把需要显示的数据传送给相应七段数码管的段选和位选信号。段选信号动态扫描相应的数码管，位选信号输出到译码单元 BCD_LED。

译码单元 BCD_LED：将显示单元 DISPLAY 发出的位选信号进行七段译码，用于在数码管上显示正确的数据。

4.4.4 实验步骤

1. 运行 Quartus II 软件，分别建立新工程，选择 File→New 菜单，创建 VHDL 描述语言设计文件，分别编写 JTDKZH.VHD、CNT45S.VHD、CNT25S.VHD、CNT5S.VHD、XSKZH.VHD、DISPLAY.VHD 及 BCD_LED.VHD 源程序。根据前述实验内容自行设计 CNT45S、CNT25S、CNT5S、DISPLAY 、BCD_LED 单元，完成其功能。
2. 交通灯控制 JTDKZH 单元的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY JTDKZH IS
    PORT(CLK,SM,SB : IN STD_LOGIC;   --SM、SB 为主干道、支干道指示灯控制信号
         MR,MY,MG,BR,BY,BG : OUT STD_LOGIC); --MR:主干道红灯；MY:主干道黄灯；
          --MG:主干道绿灯；BR:从干道红灯；BY:从干道黄灯；BG:从干道绿灯；
END ENTITY JTDKZH;

ARCHITECTURE BEHV OF JTDKZH IS
    TYPE STATE_TYPE IS(A,B,C,D);
      SIGNAL STATE:STATE_TYPE;
        BEGIN
       CNT: PROCESS(CLK) IS
            VARIABLE S: INTEGER RANGE 0 TO 45;
            VARIABLE CLR,EN:BIT;
```

```
BEGIN
 IF(CLK'EVENT AND CLK='1') THEN
    IF CLR='0' THEN     --CLR=0 时 S=0
       S:=0;
    ELSIF EN='0' THEN
       S:=S;            --CLR=1、EN=0 时 S=S
    ELSE                --CLR=1、EN=1 时 S 自加 1
       S:=S+1;
    END IF;
CASE STATE IS
   WHEN A=>MR<='0';MY<='0';MG<='1';BR<='1';BY<='0';BG<='0';
                       --A 状态：主干道绿灯亮、从干道红灯亮
     IF(SB AND SM)='1' THEN
       IF S=45 THEN                  --判断 S 是否等于 45?
         STATE<=B;CLR:='0';EN:='0';  --S＝45 则跳转到 B 状态
       ELSE
         STATE<=A;CLR:='1';EN:='1';  --S/=45 保持 A 状态
                                     --CLR=1、EN=1 时 S 自加 1
       END IF;
     ELSIF(SB AND(NOT SM))='1' THEN   --SB=1,SM=0 说明
         STATE<=B;CLR:='0';EN:='0';
       ELSE
         STATE<=A;CLR:='1';EN:='1';
       END IF;
   WHEN B=>MR<='0';MY<='1';MG<='0';BR<='1';BY<='0';BG<='0';
                       --B 状态：主干道黄灯亮、从干道红灯亮
       IF S=5 THEN
         STATE<=C;CLR:='0';EN:='0';  --S＝5 则跳转到 C 状态
       ELSE
         STATE<=B;CLR:='1';EN:='1';  --S/=5 保持 B 状态
                                     --CLR=1、EN=1 时 S 自加 1
       END IF;
   WHEN C=>MR<='1';MY<='0';MG<='0';BR<='0';BY<='0';BG<='1';
                       --C 状态：主干道红灯亮、从干道绿灯亮
     IF(SM AND SB)='1' THEN
       IF S=25 THEN                  --S＝25 则跳转到 D 状态
         STATE<=D;CLR:='0';EN:='0';
       ELSE
         STATE<=C;CLR:='1';EN:='1';   --S/=25 保持 C 状态
```

```
                                                    --CLR=1、EN=1 时 S 自加 1
                    END IF;
                ELSIF SB ='0' THEN
                    STATE<=D;CLR:='0';EN:='0';
                ELSE
                    STATE<=C;CLR:='1';EN:='1';
                END IF;
            WHEN D=>MR<='1';MY<='0';MG<='0';BR<='0';BY<='1';BG<='0';
                                    --D 状态：主干道红灯亮、从干道黄灯亮
                IF S=5 THEN
                    STATE<=A;CLR:='0';EN:='0';
                ELSE
                    STATE<=D;CLR:='1';EN:='1';
                END IF;
            END CASE;
        END IF;
    END PROCESS CNT;
END ARCHITECTURE BEHV;
```

3. 显示控制 XSKZH 单元的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY XSKZH IS
    PORT(EN45,EN25,EN05M,EN05B : IN STD_LOGIC;
         AIN45M,AIN45B : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         AIN25M,AIN25B,AIN05 : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         DOUTM,DOUTB : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ENTITY XSKZH;

ARCHITECTURE BEHV OF XSKZH IS
    BEGIN
      PROCESS(EN45,EN25,EN05M,EN05B) IS
        BEGIN
          IF EN45='1' THEN
            DOUTM<=AIN45M(7 DOWNTO 0);DOUTB<=AIN45B(7 DOWNTO 0);
          ELSIF EN05M='1' THEN
            DOUTM<=AIN05(7 DOWNTO 0);DOUTB<=AIN05(7 DOWNTO 0);
```

```
        ELSIF EN25='1'   THEN
          DOUTM<=AIN25M(7 DOWNTO 0);DOUTB<=AIN25B(7 DOWNTO 0);
        ELSIF EN05B='1' THEN
          DOUTM<=AIN05(7 DOWNTO 0);DOUTB<=AIN05(7 DOWNTO 0);
        END IF;
    END PROCESS;
END ARCHITECTURE BEHV;
```

4. 编译源文件，无误后分别对上述 VHDL 文件进行仿真，验证其逻辑功能正确后，选择 File→Create/Update →Create Symbol File for Current File 菜单，分别生成符号文件.BSF。
5. 建立新工程，工程名称及顶层文件名称为 JTDKZHQ，选择 File→New 菜单，创建图形设计文件。
6. 如图 4-4-1 所示，完成交通灯控制器的设计。将创建的图形设计文件保存为 JTDKZHQ.BDF 作为整个设计的顶层文件。
7. 选择 Tools→Compiler Tool 菜单，编译 JTDKZHQ.BDF 文件。编译无误后建立仿真波形文件 JTDKZHQ.VWF。
8. 选择 Tools→Simulator Tool 菜单进行仿真。分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-4-1 所示。

表 4-4-1 交通灯控制实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	SEG[1]	PIN_20
CLK1	PIN_29	SEG[2]	PIN_44
LED[0]	PIN_12	SEG[3]	PIN_45
LED[1]	PIN_13	SM	PIN_194
LED[2]	PIN_14	SB	PIN_193
LED[3]	PIN_ 15	BG	PIN_2
LED[4]	PIN_16	BY	PIN_3
LED[5]	PIN_17	BR	PIN_4
LED[6]	PIN_18	MG	PIN_11
LED[7]	PIN_19	MY	PIN_8
SEG[0]	PIN_21	MR	PIN_7

9. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
10. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-4-2 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

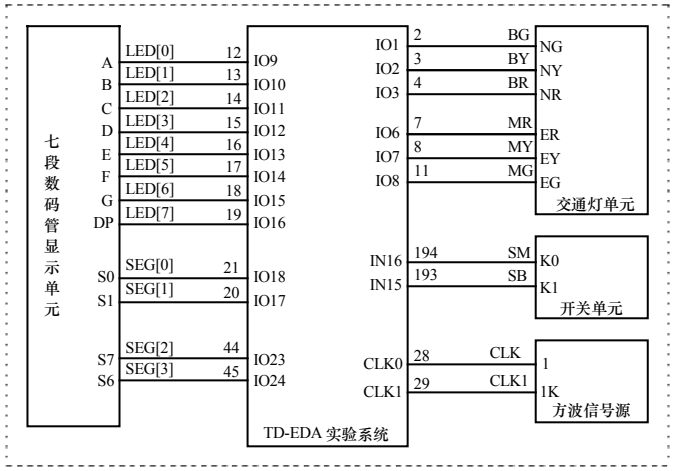


图 4-4-2 交通灯控制实验接线图

11. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
12. 配置完成后，观察实验现象验证设计的正确性。

4.4.5 实验扩展

本实验实现了一个由一条主干道和一条支干道的汇合点形成的十字交叉路口的交通灯控制器。为了满足主、支干道通行时间变化的需要，可以设计一个可配置主、支干道通行时间的交通灯控制器。

4.5 数字钟实验

4.5.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 掌握数字钟的原理及设计方法。

4.5.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

4.5.3 实验内容

钟表是我们生活中的必备工具，通常的时钟具备显示时、分、秒的功能，有些以 24 小时循环计数、有些以 12 小时循环计数。从结构上看数字钟可以分为三个部分：计时电路、显示电路和调整控制电路。在计时电路中秒针部分、分针部分由 60 进制计数器组成，时针部分可以由 12 进制计数器或者 24 进制计数器组成。显示电路部分由 6 个数码管构成，对小时、分钟、秒进行显示。调整控制电路则用于调整数字钟的参数。

本实验设计一个综合的数字钟，要求能实现时、分、秒的计时功能，将计时结果显示在六个七段数码管上，并且可通过两个设置键对数字钟的时间进行调整。

4.5.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 CLOCK。
2. 选择 File→New 菜单，创建图形设计文件，在图形编辑器界面中分别新建 TZHKZH、CNT60、CNT24、MULX、BCD_LED 子模块，完成模块的定义及模块之间的连接，完成如图 4-5-1 所示的数字钟实验设计原理图。

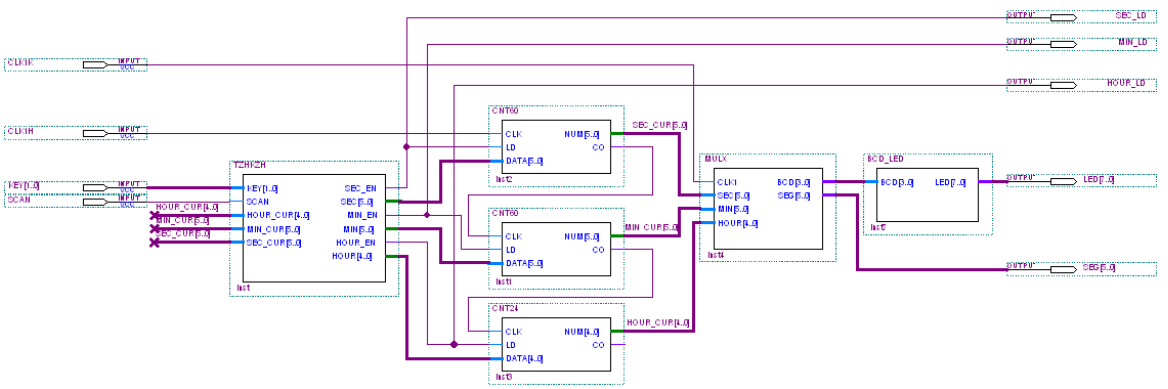


图 4-5-1 数字钟实验顶层设计原理图

3. 选择 File→Save As 菜单，将创建的图形设计文件保存为 BLOCK.BDF 作为顶层文件。
4. 分别右键点击 TZHKZH 模块、CNT60 模块、CNT24 模块、MULX 模块、BCD_LED 模块，在弹出的菜单中选择”Create Design File from Selected Block”，生成名称为 TZHKZH.VHD、CNT60.VHD、CNT24.VHD、MULX.VHD、BCD_LED.VHD 的 VHDL 设计文件。
5. 在文本编辑器界面中编写 TZHKZH.VHD、CNT60.VHD、CNT24.VHD、MULX.VHD、BCD_LED.VHD 程序。
6. TZHKZH 模块 VHDL 源程序如下：

```
--时间调整控制电路 TZHKZH 的 VHDL 源程序：TZHKZH.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY TZHKZH IS
    PORT( KEY : IN STD_LOGIC_VECTOR(1 DOWNTO 0); --按键信号
          SCAN : IN STD_LOGIC;   --键盘扫描信号
          HOUR_CUR : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
          MIN_CUR,SEC_CUR : IN STD_LOGIC_VECTOR(5 DOWNTO 0);
          SEC_EN:OUT STD_LOGIC;
          SEC : BUFFER STD_LOGIC_VECTOR(5 DOWNTO 0);
          MIN_EN:OUT STD_LOGIC;
          MIN : BUFFER STD_LOGIC_VECTOR(5 DOWNTO 0);
          HOUR_EN:OUT STD_LOGIC;
          HOUR : BUFFER STD_LOGIC_VECTOR(4 DOWNTO 0));
END ENTITY TZHKZH;

ARCHITECTURE BEHV OF TZHKZH IS
    TYPE STATE IS(NORMAL,SEC_SET,MIN_SET,HOUR_SET);
    SIGNAL MODE:STATE;
    BEGIN
      PROCESS(KEY,SCAN)
          BEGIN
            IF(SCAN'EVENT AND SCAN='1') THEN
                IF      KEY="01" THEN   --如果按下设置键，切换到下一个状态
                    SEC_EN<='1';MIN_EN<='1';HOUR_EN<='1';
                    CASE MODE IS
 WHEN NORMAL   => MODE<=SEC_SET; SEC<=SEC_CUR;SEC_EN<='0';
 WHEN SEC_SET   => MODE<=MIN_SET;MIN<=MIN_CUR;MIN_EN<='0';SEC_EN<='1';
 WHEN MIN_SET   => MODE<=HOUR_SET;HOUR<=HOUR_CUR;HOUR_EN<='0';MIN_EN<='1';
```

```
WHEN HOUR_SET =>MODE<=NORMAL;
                END CASE;
            ELSIF KEY="10" THEN     --如果按下调整键，则自加
                CASE MODE IS
                    WHEN SEC_SET => SEC_EN<='0'; --异步并行置位使能有效
                        IF SEC="111011" THEN SEC<="000000";
                                --如果秒计数到 59，返回到 0 重新计数
                        ELSE SEC<=SEC+1; --否则继续计数
                        END IF;

                    WHEN MIN_SET => MIN_EN<='0'; --异步并行置位使能有效
                        IF MIN="111011" THEN MIN<="000000";
                                --如果分计数到 59，返回到 0 重新计数
                        ELSE MIN<=MIN+1; --否则继续计数
                        END IF;

                    WHEN HOUR_SET => HOUR_EN<='0'; --异步并行置位使能有效
                        IF HOUR="10111" THEN HOUR<="00000";
                                --如果时计数到 23，返回到 0 重新计数
                        ELSE HOUR<=HOUR+1; --否则继续计数
                        END IF;

                    WHEN OTHERS     =>NULL;
                END CASE;
            END IF;
        END IF;
    END PROCESS;
END ARCHITECTURE BEHV;
```

7. 编写其它模块的 VHDL 程序，选择 File→Save As 菜单，将创建的 VHDL 设计文件保存。
8. 选择 Tools→Compiler Tool 菜单，编译 BLOCK.BDF 源文件。编译无误后建立仿真波形文件 BLOCK.VWF，选择 Tools→Simulator Tool 菜单进行仿真。
9. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-5-1 所示。
10. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。

表 4-5-1 数字钟实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK1H	PIN_28	LED [0]	PIN_2
CLK1K	PIN_29	LED [1]	PIN_3
KEY [0]	PIN_175	LED [2]	PIN_4
KEY [1]	PIN_176	LED [3]	PIN_5
SCAN	PIN_194	LED [4]	PIN_6
SEG [0]	PIN_12	LED [5]	PIN_7
SEG [1]	PIN_13	LED [6]	PIN_8
SEG [2]	PIN_14	LED [7]	PIN_11
SEG [3]	PIN_15	SEC_LD	PIN_20
SEG [4]	PIN_16	MIN_LD	PIN_21
SEG [5]	PIN_17	HOUR_LD	PIN_23

11. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-5-2 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

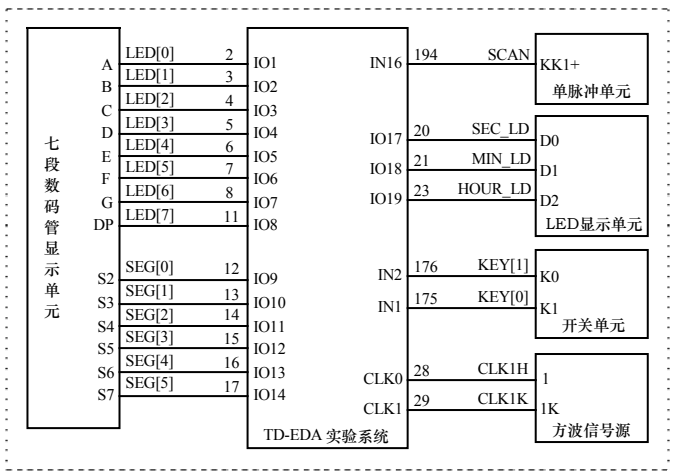


图 4-5-2 数字钟实验接线图

12. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
13. 配置完成后，验证设计的正确性。

4.5.5 实验扩展

本实验实现了一个 24 进制时间可调的实用数字钟。为了体现更多的功能，设计一个可以分别显示日期和时间，并且日期、时间均可调的数字钟。

4.6 液晶显示实验

4.6.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 学习 16×2 字符型液晶显示器的基本原理；
3. 掌握液晶显示控制器的设计方法。

4.6.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

4.6.3 实验内容

液晶显示器(LCD)由于体积小、质量轻、功耗小等优点，已成为各种便携式电子产品的理想显示器。液晶显示器从其显示的内容可分为字段型、点阵字符型和点阵图形式三种。点阵字符型液晶显示器是专门用于显示数字、字母、图形符号及少量自定义符号的显示器。这类显示器把 LCD 控制器、点阵驱动器全部做在一块印刷电路板上，构成便于应用的液晶显示模块。

字符型液晶显示模块在国际上已经规范化，控制器主要有 KS0066（三星公司产品）、HD44780（日立公司产品）、SED1278（EPSON 公司产品）。EDA 实验系统使用的液晶模块使用的是 HD44780 控制器。HD44780 控制器的指令如表 4-6-1 所示。

表 4-6-1 HD44780 控制器指令表

指令名称	控制信号		控制代码								运行时间
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	250KHz
清屏	0	0	0	0	0	0	0	0	0	1	1.64ms
归 HOME 位	0	0	0	0	0	0	0	0	1	*	1.64ms
输入方式设置	0	0	0	0	0	0	0	1	I/D	S	40us
显示状态设置	0	0	0	0	0	0	1	D	C	B	40us
光标画面滚动	0	0	0	0	0	1	S/C	R/L	*	*	40us
工作方式设置	0	0	0	0	1	DL	N	F	*	*	40us
CGRAM 地址设置	0	0	0	1	A5	A4	A3	A2	A1	A0	40us
DDRAM 地址设置	0	0	1	A6	A5	A4	A3	A2	A1	A0	40us
读 BF 和 AC 值	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	0us
写数据	1	0	数 据								40us
读数据	1	1	数 据								40us

注:*表示任意值,在实际应用时一般认为是 0

表 4-6-1 中的指令功能如下所述：

1. 清屏(Clear Display)指令

格式：

0	0	0	0	0	0	0	1

代码：01H

功能：将空码(20H)写入 DDRAM 的全部 80 个单元，将地址指针计数器 AC 清零，光标或闪烁归 HOME 位，设置输入方式参数 I/D=1。该指令多用于上电时或更新全屏显示内容时。在使用该指令前要确认 DDRAM 的当前内容是否有用。

2. 归 HOME 位(Return Home)

格式：

0	0	0	0	0	0	1	0

代码：02H

功能：该指令将地址指针计数器 AC 清零。执行该指令的效果有：将光标或闪烁位返回到显示屏的左上第一字符位上，即 DDRAM 地址 00H 单元位置，这是因为光标和闪烁位都是以地址指针计数器 AC 当前值定位的。如果画面已滚动，则撤消滚动效果，将画面拉回到 HOME 位。

3. 输入方式设置(Enter Mode Set)

格式：

0	0	0	0	0	1	I/D	S

代码：04H～07H

功能：该指令设置了显示字符的输入方式，即在计算机读/写 DDRAM 或 CGRAM 后，地址指针计数器 AC 的修改方式，反映在显示效果上，当写入一个字符后画面或光标的移动。该指令的两个参数位 I/D 和 S 确定了字符的输入方式。

I/D 表示当计算机读/写 DDRAM 或 CGRAM 的数据后，地址指针计数器 AC 的修改方式，由于光标位置也是由 AC 值确定，所以也是光标移动的方式。

I/D=0 AC 为减一计数器，光标左移一个字符位。

I/D=1 AC 为加一计数器，光标右移一个字符位。

S 表示在写入字符时，是否允许显示画面的滚动

S=0 禁止滚动

S=1 允许滚动

S=1 且 I/D=0 显示画面向右滚动一个字符

S=1 且 I/D=1 显示画面向左滚动一个字符

综合而论，该指令可以实现四种字符的输入方式，如表 4-6-2 所示。

表 4-6-2 输入方式设置指令功能表

输入方式	指令代码	参数状态
画面不动光标左移	04H	I/D= S= 0
画面右滚动	05H	I/D= 0 S= 1
画面不动光标右移	06H	I/D= 1 S= 0
画面左滚动	07H	I/D= S= 1

4. 显示状态设置(Display on/off Control)

格式:

0	0	0	0	1	D	C	B

代码：08H～0FH

功能：该指令控制着画面，光标及闪烁的开与关，该指令有三个状态位 D、C、B，这三个状态位分别控制着画面，光标和闪烁的显示状态。

D 画面显示状态位。

当 D＝1 时为开显示，当 D＝0 时为关显示，而 DDRAM 内容不变。这与清屏指令截然不同。

C 光标显示状态位。

当 C＝1 时为光标显示，当 C＝0 时为光标消失。光标为底线形式（5×1 点阵），出现在第八行或第十一行上。光标的位置由地址指针计数器 AC 确定，并随其变动而移动。当 AC 值超出了画面的显示范围，光标将随之消失。

B 闪烁显示状态位

当 B＝1 时为闪烁启用，当 B＝0 时为闪烁禁止。闪烁是指一个字符位交替进行正常显示态和全亮显示态。闪烁频率在控制器工作频率为 250KHz 时为 2.4Hz。闪烁位置同光标一样受地址指针计数器 AC 的控制。

闪烁出现在有字符或光标显示的字符位时，正常显示态为当前字符或光标的显示；全亮显示态为该字符位所有点全显示。若出现在无字符或光标显示的字符位时，正常显示态为无显示。全亮显示态为该字符位所有点全显示。这种闪烁方式可以设计成块光标，如同计算机 CRT 上块状光标闪烁提示符的效果。该指令代码的功能如表 4-6-3 所示。

表 4-6-3 显示状态设置指令功能表

指令代码	状态位			功能
	D	C	B	
08H～0BH	0	*	*	关显示
0CH	1	0	0	画面显示
0DH	1	0	1	画面，闪烁显示
0EH	1	1	0	画面，光标显示
0FH	1	1	1	画面，光标，闪烁显示

5. 光标或画面滚动(Cursor or Display Shift)

格式:

0	0	0	1	S/C	R/L	0	0

功能：该指令将产生画面或光标向左或向右滚动一个字符位。如果定时间隔地执行该指令将产生画面或光标地平滑滚动。画面的滚动是在一行内连续循环进行的，也就是说一行的第一单元与最后一个单元连接起来，形成了闭环式的滚动。当未开光标显示时，执行画面滚动指令

时不修改地址指针计数器 AC 值；当有光标显示时，由于执行任意一条滚动指令时都将使光标产生位移，所以地址指针计数器 AC 都需要被修改。光标的滚动是在 DDRAM 内全程进行的，它不分是一行显示还是两行显示。

S/C 滚动对象的选择

S/C＝1 画面滚动

S/C＝0 光标滚动

R/L 滚动方向的选择

R/L＝1 向右滚动

R/L＝0 向左滚动

该指令代码的功能如表 4-6-4 所示。

表 4-6-4 光标或画面滚动指令功能表

指令代码	状态位		功能
	S/C	R/L	
10H	0	0	光标左滚动，AC 自动减 1
14H	0	1	光标右滚动，AC 自动加 1
18H	1	0	画面左滚动
1CH	1	1	画面右滚动

该指令与输入方式设置指令都可以产生光标或画面的滚动，区别在于该指令专用于滚动功能，执行一次，显示呈现一次滚动效果；而输入方式设置指令仅是完成了一种字符输入方式的设置，仅在计算机对 DDRAM 等进行操作时才能产生滚动的效果。

6. 工作方式设置(Function Set)

格式:

0	0	1	DL	N	F	0	0

功能：该指令设置了控制器的工作方式，包括有控制器与计算机的接口形式和控制器显示驱动的占空比系数等。该指令有三个参数 DL，N 和 F。它们的作用是:

DL 设置控制器与计算机的接口形式。接口形式体现在数据总线长度上。

DL＝1 设置数据总线为 8 位长度，即 DB7～DB0 有效

DL＝0 设置数据总线为 4 位长度，即 DB7～DB4 有效。该方式下 8 位指令代码和数据将按先高 4 位后低 4 位的顺序分两次传送

N 设置显示的字符行数

N＝0 为一行字符行

N＝1 为两行字符行

F 设置显示字符的字体

F＝0 为 5×7 点阵字符体

F＝1 为 5×10 点阵字符体

N 和 F 的组合设置了控制器的确定占空比系数。如表 4-6-5 所示。

表 4-6-5 工作方式设置指令功能表

N	F	字符行数	字符体形式	占空比系数
0	0	1	5×7	1/8
0	1	1	5×10	1/11
1	0	2	5×7	1/16

该指令是字符型液晶显示控制器的初始化设置指令，也是唯一的软件复位指令。HD44780 虽然具有复位电路，但为了可靠的工作，HD44780 要求计算机在操作时首先对其进行软件复位。也就是说在控制字符型液晶显示模块工作时首先要进行的软件复位。

7. CGRAM 地址设置(Set CG RAM Address)

格式：

0	1	A5	A4	A3	A2	A1	A0

功能：该指令将 6 位的 CGRAM 地址写入地址指针计数器 AC 内，随后计算机对数据的操作是对 CGRAM 的读/写操作。

8. DDRAM 地址设置(Set DD RAM Address)

格式：

1	A6	A5	A4	A3	A2	A1	A0

功能：该指令将 7 位 DDRAM 地址写入地址指针计数器 AC 内，随后计算机对数据的操作是对 DDRAM 的读/写操作。

9. 读“忙”标志和地址指针值(Read Busy Flag and Address)

格式：

BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0

功能：计算机对指令寄存器通道读操作（RS＝0，R/W＝1）时，将读出此格式的“忙”标志 BF 值和 7 位地址指针计数器 AC 的当前值。计算机随时都可以对 HD44780 读“忙”操作。BF 值反映 HD44780 的接口状态。计算机在对 HD44780 每次操作时首先都要读 BF 值判断 HD44780 的当前接口状态，仅有在 BF＝0 时计算机才可以向 HD44780 写指令代码或显示数据和从 HD44780 读出显示数据。计算机读出的地址指针计数器 AC 当前值可能是 DDRAM 地址也可能是 CGRAM 的地址，这取决于最近一次计算机向 AC 写入的是那类地址。

10. 写数据(Write Data to CG or DD RAM)

计算机向数据寄存器通道写入数据，HD44780 根据当前地址指针计数器 AC 值的属性及数值将该数据送入相应的存储器内的 AC 所指的单元里。如果 AC 值为 DDRAM 地址指针，则认为写入的数据为字符代码并送入 DDRAM 内 AC 所指的单元里；如果 AC 值为 CGRAM 的地址指针，则认为写入的数据是自定义字符的字模数据并送入 CGRAM 内 AC 所指的单元里。所以计算机在写数据操作之前要先设置地址指针或人为的确认地址指针的属性及数值。在写入数据后地址指针计数器 AC 将根据最近设置的输入方式自动修改。由此可知，计算机在写数据操作之前要作两项工作，其一是设置或确认地址计数器 AC 值的属性及数值，以保证所写数据能够正确到位；其二是设置或确认输入方式，以保证连续写入数据时 AC 值的修改方式符号要求。

11. 读数据(Read Data to CG or DD RAM)

在 HD44780 的内部运行时序的操作下，地址指针计数器 AC 的每一次修改，包括新的 AC 值的写入，光标滚动位移所引起的 AC 值的修改或由计算机读写数据操作后所产生的 AC 值的修改，HD44780 都会把当前 AC 所指单元的内容送到接口部数据输出寄存器内，供计算机读取。如果 AC 值为 DDRAM 地址指针，则认为接口部数据输出寄存器的数据为 DDRAM 内 AC 所指单元的字符代码；如果 AC 值为 CGRAM 的地址指针，则认为数据输出寄存器的数据是 CGRAM 内 AC 所指单元的自定义字符的字模数据。

计算机的读数据是从数据寄存器通道中数据输出寄存器读取当前所存放的数据。所以计算机在首次读数据操作之前需要重新设置一次地址指针 AC 值，或用光标滚动指令将地址指针计数器 AC 值修改到所需的地址上，然后进行的读数据操作将能获得所需的数据。在读取数据后地址指针计数器 AC 将根据最近设置的输入方式自动修改。由此可知，计算机在读数据操作之前要作两项工作，其一是设置或确认地址计数器 AC 值的属性及数值，以保证所读数据的正确性；其二是设置或确认输入方式，以保证连续读取数据时 AC 值的修改方式符号要求。

HD44780 控制器的接口时序为 M6800 时序，其特点是读/写操作时序是由使能信号 E 完成。E 信号是正脉冲信号，不操作时为低电平状态，操作时产生一个正脉冲。在读操作时 E 信号在高电平时，控制器将所需数据送入数据总线上，供计算机读取；在写操作时，E 信号的下降沿处将数据总线上的数据写入控制器接口部的寄存器内。

HD44780 控制器对读/写操作的识别是判断 R/W 信号端上的电平状态，R/W＝1 为读操作选择，R/W＝0 为写操作选择。R/W 信号的宽度要大于 E 信号的宽度才能保证计算机的操作正确。

RS 信号是 HD44780 控制器识别数据总线上的数据上属于指令代码还是属于显示数据。RS＝0 选通指令寄存器通道，数据总线传输的是指令代码或标志位；RS＝1 选通数据寄存器通道，数据总线传输的是显示数据或自定义字符的字模数据。

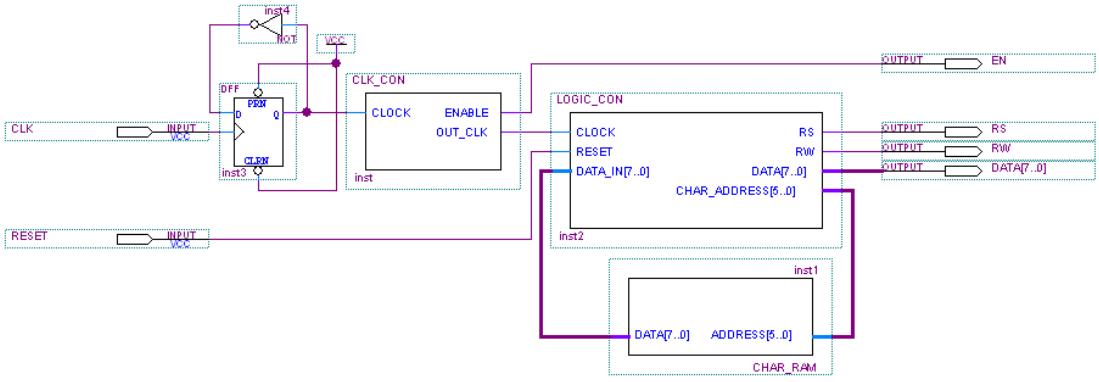


图 4-6-1 液晶显示实验顶层设计电路图

本实验使用 TD-EDA 实验系统的液晶单元，设计一个液晶接口模块，控制 LCD 动态左循环显示字符串“Welcome To Use TD-EDA”。液晶接口模块由三个单元组成：时钟控制模块 CLK_CON 主要为系统提供时钟源和产生使能信号 EN，考虑到动态左移显示时人眼能够方便的观察，时钟控制模块的输入时钟频率宜采用 50～100Hz；字符数据模块 CHAR_RAM 为系统提

供显示字符。限于篇幅，书中省略了部分字符，读者可参照 ASCII 字符表；逻辑控制模块 CON_LOGIC 实现 LCD 的状态控制、读/写操作。根据 LCD 的控制命令，采用状态机的描述方式实现 LCD 的控制功能。其顶层原理图如图 4-6-1 所示。

4.6.4 实验步骤

1. 运行 Quartus II 软件，分别建立新工程，选择 File→New 菜单，创建 VHDL 描述语言设计文件，分别编写 CLK_CON.VHD、LOGIC_CON.VHD、CHAR_RAM.VHD。
2. 时钟控制模块 CLK_CON 的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY CLK_CON IS
  PORT(CLOCK : IN    STD_LOGIC;
       ENABLE: OUT STD_LOGIC; --CLOCK 信号下降沿的二分频信号
       OUT_CLK:OUT STD_LOGIC    );   --CLOCK 信号上升沿的二分频信号
END ENTITY CLK_CON;

ARCHITECTURE BEHV OF CLK_CON IS
  COMPONENT DFF
    PORT(D : IN    STD_LOGIC;
         CLK : IN STD_LOGIC;
         Q : OUT STD_LOGIC);
  END COMPONENT;
  SIGNAL SIG1,SIG2,SIG3:STD_LOGIC;
    BEGIN
      SIG1<=NOT SIG2;
      SIG3<=NOT CLOCK;
      OUT_CLK<=SIG2;
  MYDFF1:
    DFF PORT MAP(D=>SIG1,CLK=>CLOCK,Q=>SIG2);
  MYDFF2:
    DFF PORT MAP(D=>SIG2,CLK=>SIG3,Q=>ENABLE);
END ARCHITECTURE BEHV;
```

3. 字符数据模块 CHAR_RAM 的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
```

```
ENTITY CHAR_RAM IS
   PORT(ADDRESS: IN    STD_lOGIC_VECTOR(5 DOWNTO 0);
          DATA     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END ENTITY CHAR_RAM;

ARCHITECTURE BEHV OF CHAR_RAM IS
    FUNCTION CHAR_TO_INTEGER(INDATA : CHARACTER) RETURN INTEGER IS
    VARIABLE RESULT : INTEGER RANGE 0 TO 16#7F#;--数据的范围为 0～7FH
      BEGIN
        CASE INDATA IS
          WHEN ' '=> RESULT:=16#20#;
          WHEN '!'=> RESULT:=16#21#;
          WHEN '#'=> RESULT:=16#23#;
          WHEN '$'=> RESULT:=16#24#;
          WHEN '%'=> RESULT:=16#25#;
          WHEN '&'=> RESULT:=16#26#;
          WHEN '''=> RESULT:=16#27#;
          WHEN '('=> RESULT:=16#28#;
          WHEN ')'=> RESULT:=16#29#;

          WHEN '.'=> RESULT:=16#2E#;

          WHEN 'A'=> RESULT:=16#41#;
          WHEN 'B'=> RESULT:=16#42#;
          WHEN 'C'=> RESULT:=16#43#;
          WHEN 'D'=> RESULT:=16#44#;
          WHEN 'E'=> RESULT:=16#45#;

          WHEN 'T'=> RESULT:=16#54#;
          WHEN 'U'=> RESULT:=16#55#;
          WHEN 'V'=> RESULT:=16#56#;
          WHEN 'W'=> RESULT:=16#57#;
          WHEN 'X'=> RESULT:=16#58#;
          WHEN 'Y'=> RESULT:=16#59#;
          WHEN 'Z'=> RESULT:=16#5A#;

          WHEN 'a'=> RESULT:=16#61#;
          WHEN 'b'=> RESULT:=16#62#;
          WHEN 'c'=> RESULT:=16#63#;
```

```
            WHEN 'd'=> RESULT:=16#64#;
            WHEN 'e'=> RESULT:=16#65#;

            WHEN 'l'=> RESULT:=16#6C#;
            WHEN 'm'=> RESULT:=16#6D#;
            WHEN 'n'=> RESULT:=16#6E#;
            WHEN 'o'=> RESULT:=16#6F#;
            WHEN 'p'=> RESULT:=16#70#;
            WHEN 'q'=> RESULT:=16#71#;
            WHEN 'r'=> RESULT:=16#72#;
            WHEN 's'=> RESULT:=16#73#;

            WHEN OTHERS => RESULT:=16#20#;
        END CASE;
      RETURN RESULT;
    END FUNCTION;
     BEGIN
        PROCESS(ADDRESS)
          BEGIN
           CASE ADDRESS IS
        WHEN "000000" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('W'),8);
        WHEN "000001" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('e'),8);
        WHEN "000010" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('l'),8);
        WHEN "000011" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('c'),8);
        WHEN "000100" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('o'),8);
        WHEN "000101" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('m'),8);
        WHEN "000110" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('e'),8);
        WHEN "000111" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER(' '),8);
        WHEN "001000" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('T'),8);
        WHEN "001001" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('o'),8);
        WHEN "001010" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER(' '),8);
        WHEN "001011" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('U'),8);
        WHEN "001100" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('s'),8);
        WHEN "001101" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('e'),8);
        WHEN "001110" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER(' '),8);
        WHEN "001111" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('T'),8);
        WHEN "010000" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('D'),8);
        WHEN "010001" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('-'),8);
        WHEN "010010" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('E'),8);
```

```
        WHEN "010011" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('D'),8);
        WHEN "010100" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('A'),8);
        WHEN "010101" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER('.'),8);
        WHEN "010110" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER(' '),8);
        WHEN "010111" =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER(' '),8);
        WHEN OTHERS   =>DATA<=CONV_STD_LOGIC_VECTOR(CHAR_TO_INTEGER(' '),8);
      END CASE;
    END PROCESS;
END ARCHITECTURE BEHV;
```

4. 逻辑控制模块 LOGIC_CON 的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

PACKAGE CONTROL IS
  CONSTANT IDLE            : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00000000000";
  CONSTANT CLEAR           : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00000000001";
  CONSTANT RETURNCURSOR : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00000000010";
  CONSTANT SETMODE         : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00000000100";
  CONSTANT SWITCHMODE      : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00000001000";
  CONSTANT SHIFT           : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00000010000";
  CONSTANT SETFUNCTION     : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00000100000";
  CONSTANT SETCGRAM        : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00001000000";
  CONSTANT SETDDRAM        : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00010000000";
  CONSTANT READFLAG        : STD_LOGIC_VECTOR(10 DOWNTO 0) :="00100000000";
  CONSTANT WRITERAM        : STD_LOGIC_VECTOR(10 DOWNTO 0) :="01000000000";
  CONSTANT READRAM         : STD_LOGIC_VECTOR(10 DOWNTO 0) :="11000000000";
  CONSTANT CUR_INC         : STD_LOGIC :='1';
  CONSTANT CUR_DEC         : STD_LOGIC :='0';
  CONSTANT CUR_SHIFT       : STD_LOGIC :='1';
  CONSTANT CUR_NOSHIFT     : STD_LOGIC :='0';
  CONSTANT OPEN_DISPLAY    : STD_LOGIC :='1';
  CONSTANT CLOSE_DISPLAY : STD_LOGIC :='0';
  CONSTANT OPEN_CUR        : STD_LOGIC :='1';
  CONSTANT CLOSE_CUR       : STD_LOGIC :='0';
  CONSTANT RAY_CUR         : STD_LOGIC :='1';
  CONSTANT BLANK_CUR       : STD_LOGIC :='0';
  CONSTANT SHIFT_DISPLAY : STD_LOGIC :='1';
  CONSTANT SHIFT_CUR       : STD_LOGIC :='0';
```

```
    CONSTANT RIGHT_SHIFT     : STD_LOGIC :='1';
    CONSTANT LEFT_SHIFT      : STD_LOGIC :='0';
    CONSTANT DATAWIDTH8      : STD_LOGIC :='1';
    CONSTANT DATAWIDTH4      : STD_LOGIC :='0';
    CONSTANT TWOLINE         : STD_LOGIC :='1';
    CONSTANT ONELINE         : STD_LOGIC :='0';
    CONSTANT FONT5_10        : STD_LOGIC :='1';
    CONSTANT FONT5_7         : STD_LOGIC :='0';
END CONTROL;

    LIBRARY IEEE;
    USE IEEE.STD_LOGIC_1164.ALL;
    USE IEEE.STD_LOGIC_ARITH.ALL;
    LIBRARY WORK;
    USE WORK.CONTROL.ALL;
    ENTITY   LOGIC_CON IS
      PORT(CLOCK : IN   STD_LOGIC;
           RESET : IN   STD_LOGIC;
           DATA_IN:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
           RS      : OUT STD_LOGIC;
           RW      : OUT STD_LOGIC;
           DATA    : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
           CHAR_ADDRESS : OUT STD_LOGIC_VECTOR(5 DOWNTO 0));
    CONSTANT DIVSS : INTEGER :=15;
    END ENTITY LOGIC_CON;
    ARCHITECTURE BEHV OF LOGIC_CON IS
      SIGNAL STATE:STD_LOGIC_VECTOR(10 DOWNTO 0);
      SIGNAL COUNTER : INTEGER RANGE 0 TO 127;
      SIGNAL DIV_COUNTER : INTEGER RANGE 0 TO 15;
      SIGNAL FLAG : STD_LOGIC;
      BEGIN
  RS<='1' WHEN STATE = WRITERAM OR STATE=READRAM ELSE
              '0';      --RS 信号在读、写数据时为 1，其他时间为 0。
  RW<='0' WHEN STATE = CLEAR OR STATE = RETURNCURSOR OR STATE = SETMODE OR
  STATE = SWITCHMODE OR STATE = SHIFT OR STATE = SETFUNCTION OR
  STATE = SETCGRAM OR STATE = SETDDRAM OR STATE = WRITERAM ELSE
               '1' ;   --R/W 信号在读 BF 值和读数据时为 1，其他时间为 0
  DATA<= "00000001" WHEN STATE = CLEAR ELSE
              "00000010" WHEN STATE = RETURNCURSOR ELSE
```

```
                    "000001"&CUR_INC&CUR_NOSHIFT WHEN STATE=SETMODE ELSE
                     "00001"&OPEN_DISPLAY&CLOSE_CUR&BLANK_CUR
                WHEN STATE=SWITCHMODE ELSE
"0001"    &SHIFT_DISPLAY&LEFT_SHIFT&"00" WHEN STATE=SHIFT ELSE
"001" & DATAWIDTH8 & TWOLINE & FONT5_7 & "00" WHEN STATE = SETFUNCTION ELSE
"01000000" WHEN STATE = SETCGRAM ELSE
"10000000" WHEN STATE = SETDDRAM AND COUNTER=0    ELSE
"11000000" WHEN STATE = SETDDRAM AND COUNTER/=0 ELSE
DATA_IN     WHEN STATE = WRITERAM ELSE
                    "ZZZZZZZZ";
              CHAR_ADDRESS<=CONV_STD_LOGIC_VECTOR(COUNTER,6)
                WHEN STATE=WRITERAM AND COUNTER<35 ELSE
                                CONV_STD_LOGIC_VECTOR(COUNTER-35,6)
                WHEN STATE=WRITERAM AND COUNTER>35    AND    COUNTER<70 ELSE
                                CONV_STD_LOGIC_VECTOR(COUNTER-70,6)
                WHEN STATE=WRITERAM AND COUNTER>70 AND COUNTER<81 ELSE
                                "000000";
          PROCESS(CLOCK,RESET)
           BEGIN
              IF RESET='0' THEN
                  STATE<=IDLE;
                  COUNTER<=0;
                  FLAG<='0';
                  DIV_COUNTER<=0;
             ELSIF CLOCK'EVENT AND CLOCK='1' THEN
                  CASE STATE IS
                   WHEN IDLE =>
                      IF FLAG='0' THEN
                         STATE<=SETFUNCTION;
                         FLAG<='1';
                         COUNTER<=0;
                         DIV_COUNTER<=0;
                       ELSE
                         IF DIV_COUNTER< DIVSS THEN
                            DIV_COUNTER<=DIV_COUNTER+1;
                            STATE<=IDLE;
                        ELSE
                            DIV_COUNTER<=0;
                            STATE<=SHIFT;
```

```
                END IF;
              END IF;
            WHEN CLEAR =>
              STATE <= SETMODE;
            WHEN RETURNCURSOR =>
                STATE <= WRITERAM;
            WHEN SETMODE =>
                STATE <= WRITERAM;
            WHEN SWITCHMODE =>
                STATE <= CLEAR;
            WHEN SHIFT =>
                STATE <= IDLE;
            WHEN SETFUNCTION =>
                STATE <= SWITCHMODE;
            WHEN SETCGRAM =>
                STATE <= IDLE;
            WHEN SETDDRAM =>
                STATE <= WRITERAM;
            WHEN READFLAG =>
                STATE <= IDLE;
            WHEN WRITERAM =>
                IF COUNTER=34 THEN
                    STATE <= SETDDRAM;
                    COUNTER<=COUNTER+1;
                ELSIF
                    COUNTER/=34 AND COUNTER<69 THEN
                    STATE <= WRITERAM;
                    COUNTER<=COUNTER+1;
                ELSE
                    STATE <= SHIFT;
                END IF;
            WHEN READRAM =>
                STATE <= IDLE;
            WHEN OTHERS =>
                STATE <= IDLE;
            END CASE;
          END IF;
      END PROCESS;
END ARCHITECTURE BEHV;
```

5. 编译源文件，编译无误后选择 File→Create/Update →Create Symbol File for Current File 菜单，分别生成符号文件.BSF。
6. 建立新工程，工程名称及顶层文件名称为 LCD，选择 File→New 菜单，创建图形设计文件。如图 4-6-1 所示，完成液晶显示实验的设计。将创建的图形设计文件保存为 LCD.BDF 作为整个设计的顶层文件。
7. 选择 Tools→Compiler Tool 菜单，编译文件。
8. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-6-6 所示。

表 4-6-6 液晶显示实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	DATA [6]	PIN_8
DATA [0]	PIN_2	DATA [7]	PIN_11
DATA [1]	PIN_3	EN	PIN_12
DATA [2]	PIN_4	RW	PIN_13
DATA [3]	PIN_5	RS	PIN_14
DATA [4]	PIN_6	RESET	PIN_194
DATA [5]	PIN_7		

9. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 LCD.SOF 文件。
10. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-6-2 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

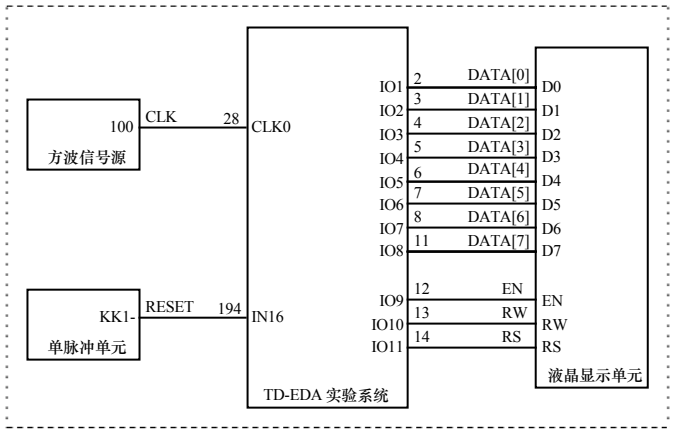


图 4-6-2 液晶显示实验接线图

11. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
12. 配置完成后，观察实验现象验证设计的正确性。

4.7 PSII 接口实验

4.7.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 学习 PSII 键盘接口的基本原理；
3. 掌握 PSII 键盘接口控制器的设计方法。

4.7.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块，PC101 键盘一个。

4.7.3 实验内容

键盘是一种输入装置，对键盘上按键开关的每个动作，由键盘内部的控制电路（其核心是一单片微处理器）转换成相应的代码，通过带屏蔽的螺旋状电缆传送到系统。

表 4-7-1 PC101 键盘扫描码转换表

按键	扫描码	按键	扫描码	按键	扫描码	按键	扫描码
A	1C	0	45	APPS	E0,2F	D ARROW	E0,72
B	32	1	16	ENTER	5A	R ARROW	E0,74
C	21	2	1E	ESC	76	NUM	77
D	23	3	26	F1	05	KP /	E0,4A
E	24	4	25	F2	06	KP *	7C
F	2B	5	2E	F3	04	KP -	7B
G	34	6	36	F4	0C	KP +	79
H	33	7	3D	F5	03	KP EN	E0,5A
I	43	8	3E	F6	0B	KP .	71
J	3B	9	46	F7	83	KP 0	70
K	42	`	0E	F8	0A	KP 1	69
L	4B	-	4E	F9	01	KP 2	72
M	3A	=	55	F10	09	KP 3	7A
N	31	\	5D	F11	78	KP 4	6B
O	44	BKSP	66	F12	07	KP 5	73
P	4D	SPACE	29	PRNT	E0,12	KP 6	74
Q	15	TAB	0D	SCROLL	7E	KP 7	6C
R	2D	CAPS	58	[	54	KP 8	75
S	1B	L SHFT	12	INS	E0,70	KP 9	7D
T	2C	L CTRL	14	HOME	E0,6C	]	5B

续表 4-7-1 PC101 键盘扫描码转换表

按键	扫描码	按键	扫描码	按键	扫描码	按键	扫描码
U	3C	L WIN	E0,1F	PG UP	E0,7D	;	4C
V	2A	L ALT	11	DEL	E0,71	‘	52
W	1D	R SHFT	59	END	E0,69	,	41
X	22	R CTRL	E0,14	PG DN	E0,7A	.	49
Y	35	R WIN	E0,27	U ARROW	E0,75	/	4A
Z	1A	R ALT	E0,11	L ARROW	E0,6B		

系统通过两类键盘程序与键盘发生关联。一类是硬件中断程序－中断 09H，另一类是软件中断程序－INT 16H。它们各自完成的功能是不同的：前者由按键动作引发，负责把键盘代码（称为扫描码）转换为相应 ASCII 码存入键盘缓冲区；后者由应用程序调用，从键盘缓冲区内取出按键产生的 ASCII 码。PC101 键盘扫描码转换如表 4-7-1 所示。

通常，将一个键盘视为类似二维矩阵的行列结构。键盘扫视程序周期性地对行、列进行扫视，根据回收的信息最终确定当前按键的行、列位置值。

PSII 键盘接口通常使用专用芯片实现。由于 PSII 键盘或鼠标串行输出信号速度较高，普通单片机无法接收，本实验使用 TD-EDA 实验系统的 PSII 单元，设计一个 PSII 接口控制器，将 PSII 接口的键盘接到实验系统上，每按下键盘上一个按键，该键的扫描码即以十六进制形式显示在数码管上。PSII 控制模块由两部分组成：PSII 控制模块 PSCON 完成 PSII 接口的时序控制和接口功能；数码管扫描显示模块 SCANLED 将按键的扫描码动态的显示在 LED 数码管上便于观察。其顶层原理图如图 4-7-1 所示。

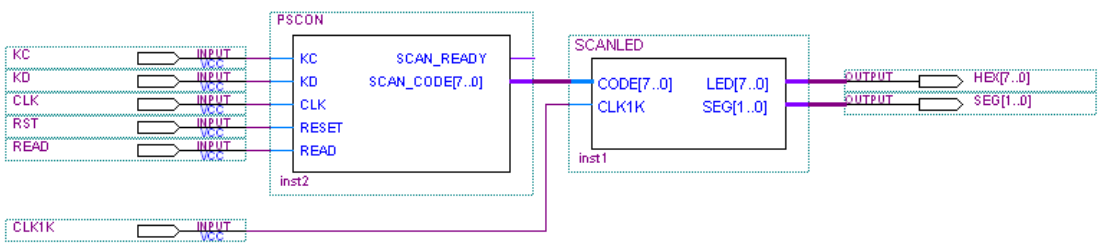


图 4-7-1 PSII 实验顶层设计电路图

4.7.4 实验步骤

1. 运行 Quartus II 软件，分别建立新工程，选择 File→New 菜单，创建 VHDL 描述语言设计文件，分别编写 PSCON.VHD、MULX.VHD、DECODER7.VHD。
2. PSII 控制模块 PSCON 的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
ENTITY PSCON IS
  PORT( KC : IN STD_LOGIC;--键盘扫描时钟
        KD : IN STD_LOGIC;--键盘输入数据
       CLK : IN STD_LOGIC;--系统时钟
     RESET: IN STD_LOGIC;--复位
      READ: IN STD_LOGIC;--读
 SCAN_READY: OUT STD_LOGIC;--忙标志
      SCAN_CODE: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));--七段数码管显示数据
END ENTITY PSCON;

ARCHITECTURE BEHV OF PSCON IS
   SIGNAL INCNT     : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL SHIFTIN : STD_LOGIC_VECTOR(8 DOWNTO 0);
   SIGNAL READ_CHAR : STD_LOGIC;
   SIGNAL INFLAG    : STD_LOGIC;
   SIGNAL READY_SET : STD_LOGIC;
   SIGNAL KEYBOARD_CLK_FILTERED :STD_LOGIC;
   SIGNAL FILTER : STD_LOGIC_VECTOR(7 DOWNTO 0);

   BEGIN
     PROCESS(READ,READY_SET)
       BEGIN
          IF READ='1' THEN SCAN_READY<='0';
          ELSIF READY_SET 'EVENT AND READY_SET='1' THEN
              SCAN_READY<='1';
          END IF;
     END PROCESS;

     CLOCK_FILTER: PROCESS
        BEGIN
           WAIT UNTIL CLK'EVENT AND CLK='1';
             FILTER(6 DOWNTO 0)<=FILTER(7 DOWNTO 1);
             FILTER(7)<=KC;
              IF FILTER="11111111" THEN KEYBOARD_CLK_FILTERED<='1';
              ELSIF FILTER="00000000" THEN KEYBOARD_CLK_FILTERED<='0';
              END IF;
     END PROCESS CLOCK_FILTER;

     PROCESS
```

```
        BEGIN
    WAIT UNTIL(KEYBOARD_CLK_FILTERED 'EVENT AND KEYBOARD_CLK_FILTERED='1');
            IF RESET='1' THEN
               INCNT<="0000";
               READ_CHAR<='0';
            ELSE
              IF KD='0' AND READ_CHAR='0' THEN
               READ_CHAR<='1';
               READY_SET<='0';
              ELSE
                IF READ_CHAR='1' THEN
                   IF INCNT<"1001" THEN
                      INCNT<=INCNT+1;
                  SHIFTIN(7 DOWNTO 0)<=SHIFTIN(8 DOWNTO 1);
                  SHIFTIN(8)<=KD;
                  READY_SET<='0';
                     ELSE
                       SCAN_CODE<=SHIFTIN(7 DOWNTO 0);
                       READ_CHAR<='0';
                       READY_SET<='1';
                       INCNT <="0000";
                     END IF;
                  END IF;
              END IF;
           END IF;
       END PROCESS;
END ARCHITECTURE BEHV;
```

3. 数码管扫描显示模块 SCANLED 的原理图如图 4-7-2 所示

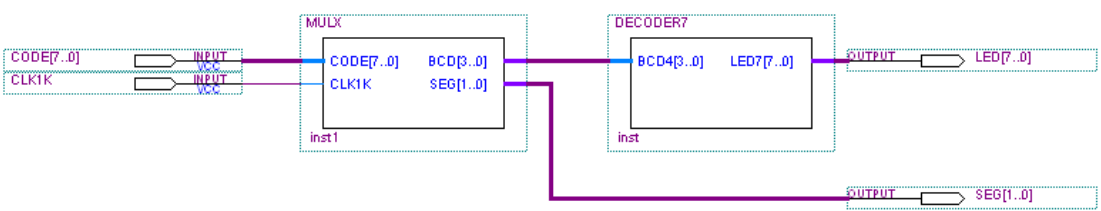


图 4-7-2 显示模块 SCANLED 电路图

4. MULX 模块的 VHDL 源程序如下：

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MULX IS
    PORT
  (CODE   : IN   STD_LOGIC_VECTOR(7 DOWNTO 0);--四位 BCD 码输入
   CLK1K : IN   STD_LOGIC;
   BCD    : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);--七位输出
   SEG    : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));--段选信号
END ENTITY MULX;

ARCHITECTURE BEHV OF MULX IS
     TYPE STATE IS(ST0,ST1);
     SIGNAL CURRENT_STATE,NEXT_STATE:STATE;
  BEGIN
     SS1:PROCESS(CLK1K)
      BEGIN
        IF(CLK1K='1' AND CLK1K 'EVENT) THEN
          CURRENT_STATE<=NEXT_STATE;
        END IF;
     END PROCESS;
     SS2:PROCESS(CURRENT_STATE)
      BEGIN
        CASE CURRENT_STATE IS
          WHEN ST0=> BCD<=CODE(3 DOWNTO 0);SEG<="10" ;NEXT_STATE<=ST1;
          WHEN ST1=> BCD<=CODE(7 DOWNTO 4);SEG<="01" ;NEXT_STATE<=ST0;
        END CASE;
      END PROCESS;
END ARCHITECTURE BEHV;
```

5. 根据前述实验内容自行设计译码模块 DECODER7 单元。
6. 编译源文件，编译无误后选择 File→Create/Update →Create Symbol File for Current File 菜单，分别生成符号文件.BSF。
7. 建立新工程，工程名称及顶层文件名称为 PSII，选择 File→New 菜单，创建图形设计文件。如图 4-7-1 所示，完成 PSII 接口实验的设计。将创建的图形设计文件保存为 PSII.BDF 作为整个设计的顶层文件。
8. 选择 Tools→Compiler Tool 菜单，编译文件。
9. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-7-3 所示。

表 4-7-3 PSII 接口实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_28	HEX [6]	PIN_8
CLK1K	PIN_29	HEX [7]	PIN_11
HEX [0]	PIN_2	KC	PIN_12
HEX [1]	PIN_3	KD	PIN_13
HEX [2]	PIN_4	SEG [0]	PIN_16
HEX [3]	PIN_5	SEG [1]	PIN_15
HEX [4]	PIN_6	READ	PIN_175
HEX [5]	PIN_7	RST	PIN_194

10. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 PSII.SOF 文件。
11. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-7-3 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

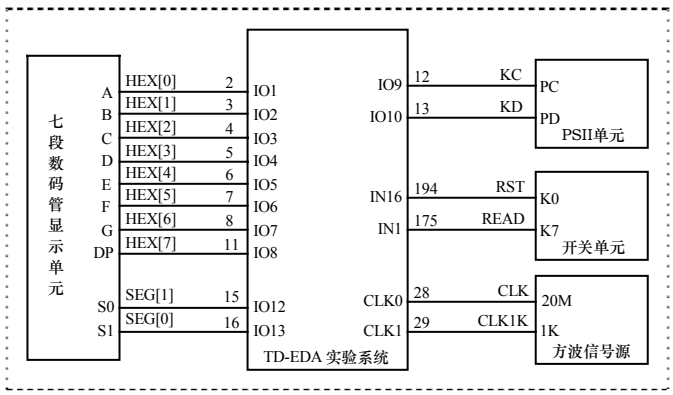


图 4-7-3 PSII 接口实验接线图

12. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
13. 配置完成后，观察实验现象验证设计的正确性。

4.8 VGA 显示实验

4.8.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 学习 VGA 显示接口的基本原理；
3. 掌握 VGA 接口控制器的设计方法。

4.8.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块，VGA 显示器一台。

4.8.3 实验内容

计算机显示器的显示有许多标准，常见的有 VGA、SVGA 等。常见的彩色显示器，一般由 CRT（阴极射线管）构成，彩色是由 R、G、B（红：Red，绿：Green，蓝：Blue）三基色组成。显示是用逐行扫描的方式解决，阴极射线枪发出电子束打在涂有荧光粉的荧光屏上，产生 RGB 三基色，合成一个彩色像素。扫描从屏幕的左上方开始，从左到右，从上到下，进行扫描，每扫完一行，电子束回到屏幕的左边下一行的起始位置。在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行行同步；扫描完所有行，用场同步信号进行场同步，并使扫描回到屏幕的左上方，同时进行场消隐，预备下一场的扫描。

对于普通的 VGA 显示器，其引出线共含 5 个信号：R、G、B：三基色信号；HS：行同步信号；VS：场同步信号。对这 5 个信号的时序驱动，要严格遵循“VGA 工业标准”，即 640×480×60 模式，否则会损害 VGA 显示器。显示过程中 HS 和 VS 的极性可正可负，显示器内可自动转换为正极性逻辑。现以正极性为例说明 CRT 的工作过程。R、G、B 为正极性信号，即高电平有效。当 VS＝0、HS＝0 时，CRT 显示的内容为亮的过程，即正向扫描过程约为 26us。当一行扫描完毕，行同步 HS＝1，约需 6us；其间，CRT 扫描产生消隐，电子束回到 CRT 左边下一行的起始位置(X＝0，Y＝1)；当扫描完 480 行后，CRT 的场同步 VS＝1，产生场同步使扫描线回到 CRT 的第一行第一列(X＝0，Y＝0)处（约为两个行周期）。HS 和 VS 的时序如图 4-8-1 所示。

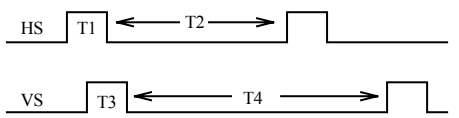


图 4-8-1 行同步 HS 和场同步 VS 的时序图

在图 4-8-1 中，T1 为行同步消隐（约为 6us)；T2 为行显示时间（约为 26us)；T3 为场同步消隐（2 行周期)；T4 为场显示时间（480 行周期)。设计的彩条信号发生器可以通过外部控制产生三种显示模式，共六种显示变化，如表 4-8-1 所示，其中的颜色编码如表 4-8-2 所示。

表 4-8-1 彩条信号发生器的六种显示变化

1	横彩条	1：白黄青绿品红黑蓝	2：黑蓝红品绿青黄白
2	竖彩条	1：白黄青绿品红黑蓝	2：黑蓝红品绿青黄白
3	棋盘格	1：棋盘格显示模式 1	2：棋盘格显示模式 2

表 4-8-2 彩条信号发生器的颜色编码

颜色	黑	蓝	红	品	绿	青	黄	白
R	0	0	0	0	1	1	1	1
G	0	0	1	1	0	0	1	1
B	0	1	0	1	0	1	0	1

VGA 工业标准要求的频率：

时钟频率(Clock Frequency)：25.175MHz（像素输出的频率）

行频(Line Frequency)：31469Hz

场频(Field Frequency)：59.94Hz（每秒图象刷新频率）

设计 VGA 图像显示控制要注意两个问题：一个是时序驱动，这是完成设计的关键，时序稍有偏差，显示就不正常，甚至会损坏彩色显示器；另一个是 VGA 信号的电平驱动。

一般 VGA 的显示控制都使用专用的显示控制器。本实验使用 TD-EDA 实验系统的 VGA 单元，用 FPGA 来实现 VGA 图像显示控制器，这在产品开发设计中有许多实际应用。其顶层原理图如图 4-8-2 所示。

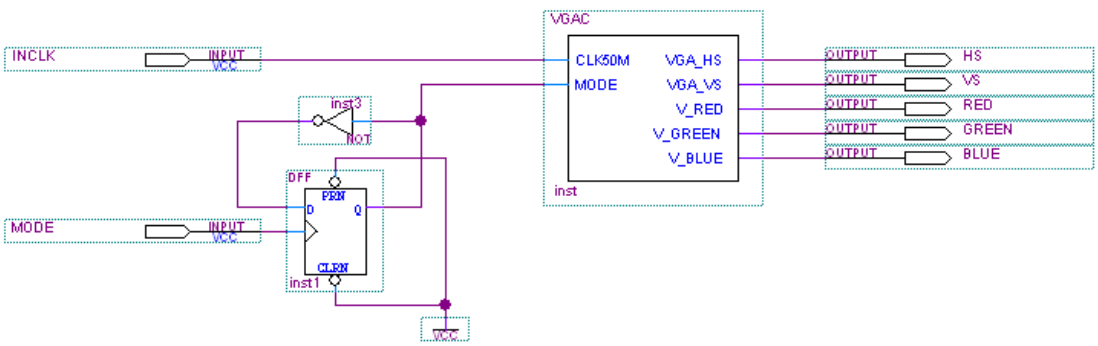


图 4-8-2 VGA 显示实验顶层设计电路图

4.8.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，选择 File→New 菜单，创建 VHDL 描述语言设计文件，编写 VGA 控制模块 VGAC 的 VHDL 源程序如下：

```
--VGA 显示器彩条信号发生器
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
ENTITY VGAC IS
   PORT(CLK50M : IN    STD_LOGIC;--扫描时钟
        MODE : IN    STD_LOGIC;   -- 模式选择
      VGA_HS : OUT STD_LOGIC;   --行同步信号
      VGA_VS : OUT STD_LOGIC;   --场同步信号
       V_RED : OUT STD_LOGIC;   --红色
     V_GREEN : OUT STD_LOGIC;   --绿色
      V_BLUE : OUT STD_LOGIC    );     --蓝色
END ENTITY VGAC;
ARCHITECTURE BEHV OF VGAC IS
    SIGNAL VGA_HS1,VGA_VS1: STD_LOGIC;
    SIGNAL FCLK,CLK : STD_LOGIC;
    SIGNAL MMD : STD_LOGIC_VECTOR(1 DOWNTO 0);--方式选择
    SIGNAL FS    : STD_LOGIC_VECTOR(5 DOWNTO 0);
    SIGNAL CC    : STD_LOGIC_VECTOR(4 DOWNTO 0);--行同步/横彩条生成
    SIGNAL LL    : STD_LOGIC_VECTOR(8 DOWNTO 0);--场同步/竖彩条生成
    SIGNAL GRBX    : STD_LOGIC_VECTOR(3 DOWNTO 1);--X 横彩条
    SIGNAL GRBY    : STD_LOGIC_VECTOR(3 DOWNTO 1);--Y 竖彩条
    SIGNAL GRBP    : STD_LOGIC_VECTOR(3 DOWNTO 1);
    SIGNAL GRB     : STD_LOGIC_VECTOR(3 DOWNTO 1);

    BEGIN
       GRB(3)<=(GRBP(3)XOR MODE) AND VGA_HS1 AND VGA_VS1;--GRB(3)为绿色信号
       GRB(2)<=(GRBP(2)XOR MODE) AND VGA_HS1 AND VGA_VS1;--GRB(2)为红色信号
       GRB(1)<=(GRBP(1)XOR MODE) AND VGA_HS1 AND VGA_VS1;--GRB(1)为蓝色信号
       PROCESS(MODE)
         BEGIN
           IF MODE'EVENT AND MODE='0' THEN    --三种模式选择
                IF MMD="10" THEN MMD<="00";
                ELSE MMD<=MMD+1;
                END IF;
           END IF;
       END PROCESS;
       PROCESS(MMD)
         BEGIN
            IF MMD="00"          THEN GRBP<=GRBX;--MMD="00"时显示 X 横彩条
            ELSIF MMD="01" THEN GRBP<=GRBY;--MMD="01"时显示 Y 竖彩条
          ELSIF MMD="10" THEN GRBP<=GRBX XOR GRBY;--MMD="10"时显示 X、Y 棋盘格
               ELSE GRBP<="000";
```

```
        END IF;
END PROCESS;
PROCESS(CLK50M)
  BEGIN
   IF CLK50M 'EVENT AND CLK50M='1' THEN --对 50MHz 信号进行 47 分频
         IF FS=46 THEN FS<="000000";      --FS 的数值可以在 45～48 之间进行调整
         ELSE FS<=FS+1;
         END IF;
       END IF;
END PROCESS;
FCLK<=FS(5);            --FS(5)为 CLK50M 信号的 47 分频   50M/47=1064KH
PROCESS(FCLK)
  BEGIN
    IF FCLK 'EVENT AND FCLK='1' THEN     --对 FCLK 进行三十分频
       IF CC=29 THEN CC<="00000";
       ELSE CC<=CC+1;
       END IF;
     END IF;
END PROCESS;
CLK<=CC(4);              --CC(4)为 FCLK[FS(5)]信号的 30 分频
                         --CC(4)为 CLK50MHz 信号的 1410 分频
PROCESS(CLK)
  BEGIN
     IF CLK 'EVENT AND CLK='0' THEN
        IF LL=481 THEN LL<="000000000";
        ELSE LL<=LL+1;
        END IF;
     END IF;
END PROCESS;
PROCESS(CC,LL)
  BEGIN
     IF CC>23 THEN VGA_HS1<='0';     --行同步消隐 HS=0 约需 6us
     ELSE VGA_HS1<='1';              --正向扫描过程行显示时间 HS＝1 约需 26us
     END IF;
   IF LL>479 THEN VGA_VS1<='0';      --场同步消隐 2 行周期 VS＝0
     ELSE VGA_VS1<='1';              --场显示时间 480 行周期 VS＝1
     END IF;
END PROCESS;
```

```
    PROCESS(CC,LL)
      BEGIN                                          --横彩条
            IF CC<3    THEN GRBX<="111";
          ELSIF CC<6    THEN GRBX<="110";
          ELSIF CC<9    THEN GRBX<="101";
          ELSIF CC<12 THEN GRBX<="100";
          ELSIF CC<15 THEN GRBX<="011";
          ELSIF CC<18 THEN GRBX<="010";
          ELSIF CC<21 THEN GRBX<="001";
          ELSE    GRBX<="000";
            END IF;

            IF LL<60      THEN GRBY<="111";--竖彩条
          ELSIF LL<120    THEN GRBY<="110";
          ELSIF LL<180    THEN GRBY<="101";
          ELSIF LL<240    THEN GRBY<="100";
          ELSIF LL<300    THEN GRBY<="011";
          ELSIF LL<360    THEN GRBY<="010";
          ELSIF LL<420    THEN GRBY<="001";
          ELSE    GRBY<="000";
            END IF;
    END PROCESS;

     VGA_HS<=VGA_HS1;
     VGA_VS<=VGA_VS1;
     V_RED<=GRB(2);
     V_GREEN<=GRB(3);
     V_BLUE<=GRB(1);
END ARCHITECTURE BEHV;
```

2. 编译源文件，编译无误后选择 File→Create/Update →Create Symbol File for Current File 菜单，分别生成符号文件 VGAC.BSF。
3. 建立新工程，工程名称及顶层文件名称为 VGA，选择 File→New 菜单，创建图形设计文件。如图 4-8-2 所示，完成 VGA 显示实验的设计。将创建的图形设计文件保存为 VGA.BDF 作为整个设计的顶层文件。
4. 选择 Tools→Compiler Tool 菜单，编译文件。
5. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-8-3 所示。

表 4-8-3 VGA 显示实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
INCLK	PIN_153	RED	PIN_2
MODE	PIN_194	GREEN	PIN_5
HS	PIN_12	BLUE	PIN_8
VS	PIN_13		

6. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 VGA.POF 文件。
7. 使用 TD-EDA 实验系统及 SOPC 开发板，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 AS 下载接口。选择 Tools→Programmer 菜单，对芯片进行配置。
8. 调整显示器的分辨率及刷新率。在 Windows 下打开“显示属性”对话框，点击“高级”按钮，在如图 4-8-3 所示显示器对话框中选择“适配器”标签页。

图 4-8-3 显示器对话框

9. 在“适配器”标签页中点击“列出所有模式”按钮，在如图 4-8-4 所示的“列出所有模式”窗口中选择“640×480，256 色，60 赫兹”。

注：有些显示器不具备存储显示模式的功能，这种显示器作实验时，显示的图像可能会有问题。

图 4-8-4 列出显示器所有模式窗口

10. 如图 4-8-5 所示进行实验接线，仔细检查确保接线无误后打开电源。

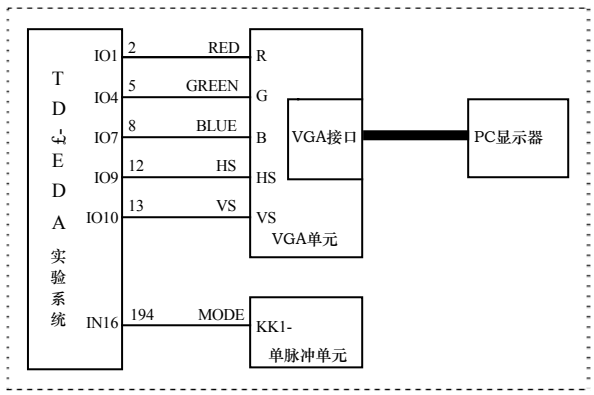


图 4-8-5 VGA 显示实验接线图

11. 观察实验现象验证设计的正确性。

4.8.5 实验扩展

在本实验的基础上设计可显示横彩条与棋盘格相间的 VGA 彩条信号发生器或者设计可显示英语字母的 VGA 信号发生器。

4.9 DDS 信号发生器实验

4.9.1 实验目的

1. 学习复杂数字系统自顶向下的设计方法；
2. 掌握 DDS 的工作原理及设计方法。

4.9.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块，示波器一台。

4.9.3 实验内容

DDS 即(Direct Digital Synthesizer 直接数字合成器)是一种新型的频率合成技术。DDS 技术是一种把一系列数字形式的信号通过 DAC 转换成模拟信号的合成技术，目前使用最广泛的一种 DDS 方式是利用高速存储器作查找表，然后通过高速 DAC 输出已经用数字形式存储的波形。

DDS 技术具有频率切换时间短，频率分辨率高，频率稳定度高，输出信号的频率和相位可以快速切换，输出相位可连续，并且在改变时能够保持相位的连续，很容易实现频率、相位和幅度的数字控制。它在相对带宽、频率转换时间、相位连续性、高分辨率以及集成化等一系列性能指标方面远远超过了传统频率合成技术。因此在现代电子系统及设备的频率源设计中，尤其在通信领域，直接数字频率合成器的应用越来越广泛。

DDS 系统的原理如图 4-9-1 所示。一般包括基准时钟、相位累加器、幅度/相位转换电路(即图 1 中的波形存储器)、D/A 转换器和低通滤波器(LPF)等几部分。

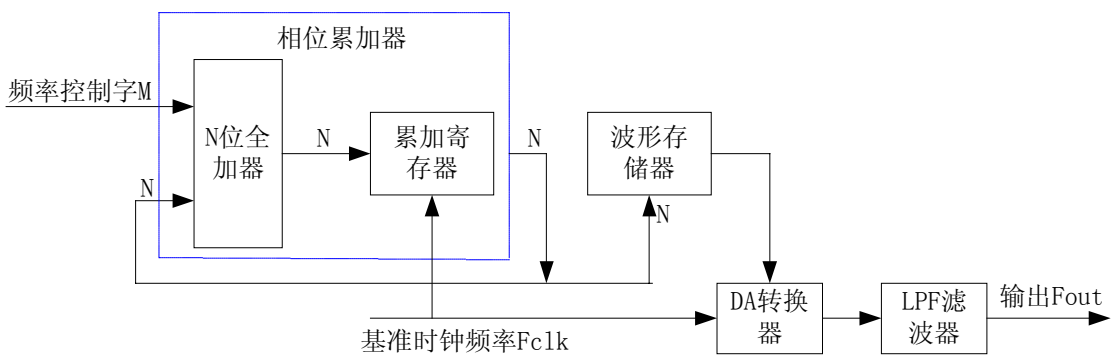


图 4-9-1 DDS 系统的基本原理图

图 4-9-1 中虚方框部分是 DDS 的核心单元，它可以采用 CPLD/FPGA 来实现。图中的相位累加器由N位全加器和N位累加寄存器级联而成，可对频率控制字的二进制码进行累加运算，是典型的反馈电路。

频率控制字 M 和相位控制字分别控制 DDS 输出正(余)弦波的频率和相位。每来一个时钟脉冲，相位寄存器以步长 M 递增。相位寄存器的输出与相位控制字相加，其结果作为正(余)弦查找表的地址。正(余)弦查找表的数据存放在 ROM 中，内部存有一个周期的正弦波信号的数字幅

度信息，每个查找表的地址对应于正弦波中 0°～360°范围内的一个相位点。查找表把输入的地址信息映射成正(余)弦波的数字幅度信号，同时输出到数模转换器 DAC 的输入端，DAC 输出的模拟信号经过低通滤波器 (LPF)，可得到一个频谱纯净的正(余)弦波。

DDS 具体工作过程如下：每来一个时钟脉冲 Fclk，N 位全加器将频率控制数据 M 与累加寄存器输出的累加相位数据 N 相加，把相加后的结果送至累加寄存器的输入端。累加寄存器一方面将上一时钟周期作用后所产生的新的数据反馈到加法器的输入端，以使加法器在下一时钟的作用下继续与频率控制数据 M 相加；另一方面将这个值作为取样地址值送入幅度/相位转换电路，此电路根据取样地址输出相应的波形数据。最后经 D/A 转换器和低通滤波器将波形数据转换成所需要的模拟波形。

DDS 输出信号的频率由下式决定：$Fout=(2^M/2^N)\times Fclk$ (2^M 代表取样点数，M 为频率控制字、2^N 代表存储器中存储数据的多少，N 代表累加器的位数，Fclk 代表基准时钟频率) 。调节 M 可以改变取样的点数，从而改变频率。

假定基准时钟为 70MHz，累加器为 16 位，则 Fclk=70MHz ，$Y=2^{16}=65536$ (N＝16)，设 M＝12 则 $X=2^{12}=4096$，所以 $Fout=(4096/65536)\times 70=4.375MHz$。可见，通过设定相位累加器位数 N、频率控制字 M 和基准时钟的值，就可以产生任一频率的输出。DDS 的频率分辨率定义为：$Fout=Fclk/2^N$ 由于基准时钟一般是固定的，因此相位累加器的位数就决定了频率的分辨率。

本实验中使用的基准时钟频率为 5MHz、相位累加器 N 为 10 位、频率控制字 M 为 15 位。根据 DDS 原理可知需要使用 32K 的存储器。所以输出的正弦波频率范围为 160Hz～160K。分辨率为 160Hz 做为频率控制字的步进值。

在 SOPC 开发板中完成的功能：设计十位频率控制字输入，经过 FPGA 中相位累加器的计算后，产生 15 位的全加器输出作为存储器的地址。

信号发生单元的 Flash 存储器 AT29C256 在出厂时已经写入了一个周期的 32K 正弦波数据，从相应地址中读出数据作为 DA 转换器的数据输入，经过 DA 转换形成模拟信号，经过运放电路进行滤波输出正弦波。信号发生单元的原理图参考 1.2 节的说明。

4.9.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 SIGNAL。
2. 选择 File→New 菜单，创建图形设计文件，如图 4-9-2 所示设计 DDS 顶层电路图。

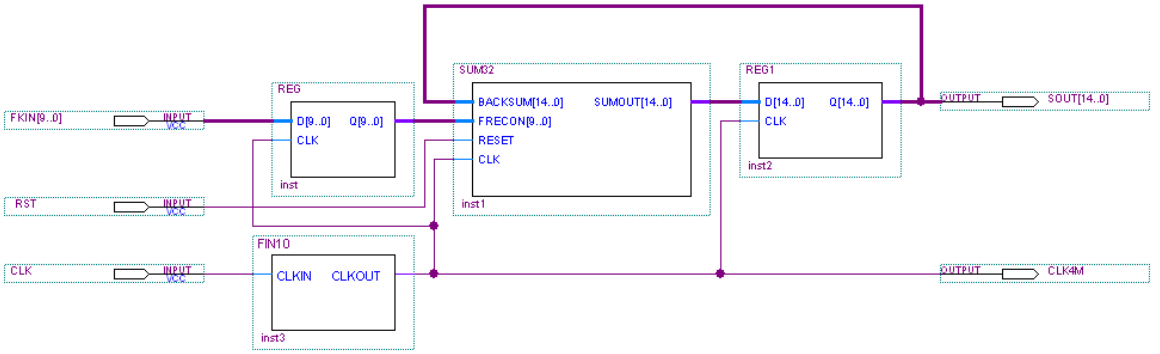


图 4-9-2 DDS 信号发生器顶层设计电路图

3. 在图 4-9-2 中 SUM32 模块是 DDS 信号源的相位累加器模块。VHDL 源程序如下：

```
--SUM32.VHD 程序是 DDS 信号源的相位累加器模块。
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY SUM32 IS
  PORT(BACKSUM : IN STD_LOGIC_VECTOR(14 DOWNTO 0);      --32 位相位累加
       FRECON : IN STD_LOGIC_VECTOR(9 DOWNTO 0);        --15 位频率控制字
         RESET : IN STD_LOGIC;   --全局复位
           CLK : IN STD_LOGIC;   --系统时钟，5MHz   5MHz/2^15=160Hz
       SUMOUT: OUT STD_LOGIC_VECTOR(14 DOWNTO 0));
END ENTITY SUM32;
ARCHITECTURE BEHV OF SUM32 IS
   SIGNAL TEMP: STD_LOGIC_VECTOR(14 DOWNTO 0);
      BEGIN
        PROCESS(CLK,RESET) IS
         BEGIN
           IF RESET='1' THEN
                TEMP<="000000000000000";
            ELSE
             IF (CLK'EVENT AND CLK='1') THEN
                    TEMP<=TEMP+FRECON;
             END IF;
           END IF;
         SUMOUT<=TEMP;
       END PROCESS;
END ARCHITECTURE BEHV;
```

4. 在图 4-9-2 中 FIN10 模块实现 10 分频生成 5M 的标准时钟信号；REG 模块和 REG1 模块实现寄存器的功能。根据前述实验内容自行设计 FIN10、REG、REG1 单元，完成其功能。
5. 选择 Tools→Compiler Tool 菜单，编译 SIGNAL.BDF 源文件。编译无误后建立仿真波形文件进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-9-1 所示。
7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
8. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-9-3 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。将 JP3 短路块接到 OE＝0、DIR＝0。仔细检查确保接线无误后打开电源。

表 4-9-1 DDS 信号发生器实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
FKIN [0]	PIN_2	SOUT [1]	PIN_57
FKIN [1]	PIN_3	SOUT [2]	PIN_156
FKIN [2]	PIN_4	SOUT [3]	PIN_158
FKIN [3]	PIN_5	SOUT [4]	PIN_159
FKIN [4]	PIN_6	SOUT [5]	PIN_160
FKIN [5]	PIN_7	SOUT [6]	PIN_161
FKIN [6]	PIN_8	SOUT [7]	PIN_162
FKIN [7]	PIN_11	SOUT [8]	PIN_163
FKIN [8]	PIN_12	SOUT [9]	PIN_164
FKIN [9]	PIN_13	SOUT [10]	PIN_165
CLK	PIN_153	SOUT [11]	PIN_166
CLK5M	PIN_19	SOUT [12]	PIN_167
RST	PIN_194	SOUT [13]	PIN_168
SOUT [0]	PIN_56	SOUT [14]	PIN_169

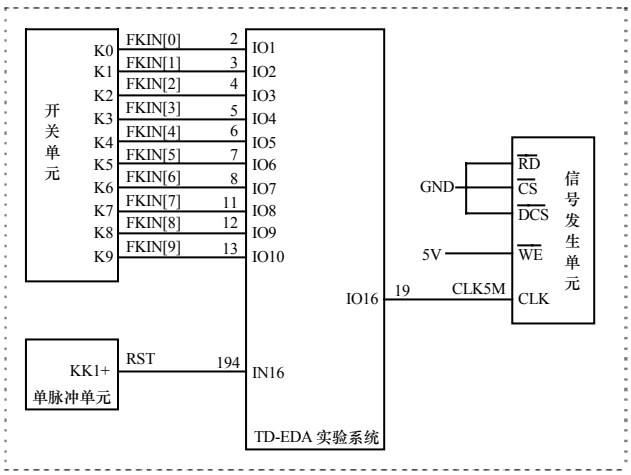


图 4-9-3 DDS 信号发生器实验接线图

9. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置。
10. 配置完成后，使用示波器观察 OUT 端输出的正弦波信号，改变 K0～K9 观察波形的变化，验证设计的正确性，由于正弦波的频率范围比较宽所以有些频率滤波效果不好。
11. 实验完成后将 JP3 短路块接到 OE＝1、DIR＝1。

4.9.5 实验扩展

在本实验实现功能的基础上，加入相位控制字，将一个具有频率控制和相位控制的 DDS 核心部分(相位累加器)重新进行设计和硬件验证。

4.10 数字频率计实验

4.10.1 实验目的

1. 学习复杂数字系统的设计方法；
2. 掌握数字频率计的原理及设计方法；
3. 掌握单片机系统与 CPLD/FPGA 联合开发的设计方法。

4.10.2 实验设备

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块，示波器或逻辑分析仪一台。

4.10.3 实验内容

常用的测频率方法主要有测频法和测周期法两种。测频法就是在确定的闸门时间 Tw 内，记录被测信号的变化周期数(或脉冲个数)Nx，则被测信号的频率为：Fx=Nx/Tw。测周期法需要有标准信号的频率 Fs，在待测信号的一个周期 Tx 内，记录标准频率的周期数 Ns，则被测信号的频率为：Fx=Fs/Ns。

这两种方法的计数值会产生±1 个字误差，并且测试精度与计数器中记录的数值 Nx 或 Ns 有关。为了保证测试精度，一般对于低频信号采用测周期法；对于高频信号采用测频法，由于测试时很不方便，所以人们提出等精度测频方法。

等精度测频法是在直接测频法的基础上发展起来的。它的闸门时间不是固定值，而是被测信号周期的整数倍，即与被测信号同步。因此消除了对被测信号计数所产生的±1 个字误差，并且达到了在整个测试频段的等精度测量。它消除了直接测评方法中对测量频率需要采用分段测试的局限性，在整个测试频段内精度能够保持恒定，不随所测信号的变化而变化。同时较宽的测频范围能够满足许多实际应用，测频原理如图 4-10-1 所示。

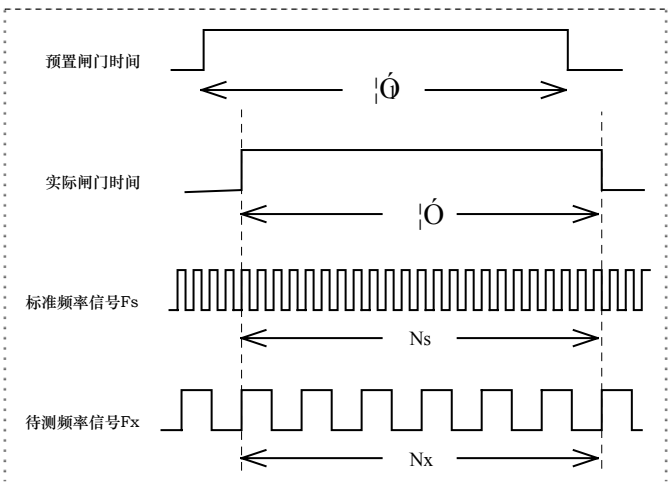


图 4-10-1 等精度测频波形图

在测量过程中，有两个计数器分别对标准信号 Fx 和被测信号 Fs 同时计数。首先给出闸门开启信号(预置闸门上升沿)，此时计数器并不开始计数，而是等到被测信号的上升沿到来时，计数器才真正开始计数。然后预置闸门关闭信号(下降沿)到时，计数器并不立即停止计数，而是等到被测信号的上升沿到来时才结束计数，完成一次测量过程。可以看出，实际闸门时间 τ 与预置闸门时间 τ1 并不严格相等，但差值不超过被测信号的一个周期。

设在一次实际闸门时间 τ 中计数器对被测信号的计数值为 Nx，对标准信号的计数值为 Ns。标准信号的频率为 Fs，则被测信号的频率为：$Fx=\frac{Nx}{Ns}\times Fs$ (4.10.1)

等精度测频的实现方法如图 4-10-2 所示。CNT1 和 CNT2 是两个可控计数器，标准频率(Fs)信号从 CNT1 的时钟输入端 CLK 输入；经整形后的被测信号(Fx)从 CNT2 的时钟输入端 CLK 输入。每个计数器中的 CEN 输入端为时钟使能，控制时钟输入，从而控制两个计数器的计数。若预置门信号为高电平(预置时间开始)，当被测信号的上升沿到来时，预置门信号通过 D 触发器输出实际阀门信号，用此信号启动两个计数器计数，分别计算被测信号 Fx 和标准信号 Fs 的个数 Nx 和 Ns，并在实际阀门关闭时及时的将数据保存到锁存器中，然后将计数初值重新赋为零。锁存器主要是用于保存两个 32 位的 Nx、Ns 数据，若预置门信号为低电平(预置时间结束)，当被测信号的上升沿到来时，预置门信号通过 D 触发器输出实际阀门信号(实际阀门时间结束)，用此信号关闭计数器的计数。这样就实现了预置门信号和被测信号的同步，消除了±1 个字的误差。

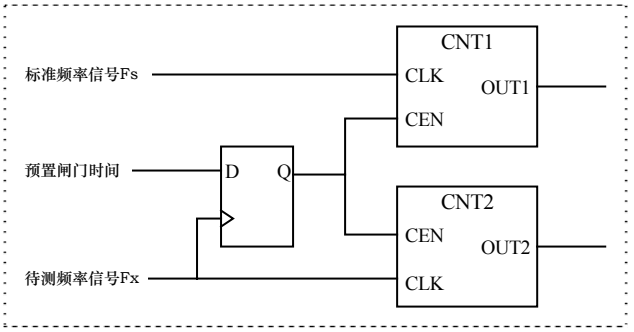


图 4-10-2 等精度测频实现方法原理图

由公式(4.10.1)可知，若忽略标准频率 Fs 的误差，则等精度测频可能产生的相对误差为：

δ=(|Fxe-Fx|/Fxe)×100% (4.10.2) 其中 Fxe 为被测信号频率的准确值。

在测量中，由于 Fx 计数的起停时间都是由该信号的上升沿触发的，在闸门时间 τ 内对 fx 的计数 Nx 无误差(τ=NxTx)；对 Fs 的计数 Ns 最多相差一个数的误差，即|ΔNs|≤1,其测量频率为：

Fxe=[Nx/(Ns+ΔNs)]/Fs (4.10.3)

将式(4.10.1)和(4.10.3)代入式(4.10.2)，并整理得：

δ=|ΔNs|/Ns≤1/Ns=1/(τ·fs)

由上式可以看出，测量频率的相对误差与被测信号频率的大小无关，仅与闸门时间和标准信号频率有关，即实现了整个测试频段的等精度测量。闸门时间越长，标准频率越高，测频的相对误差就越小。标准频率可由稳定性好、精度高的高频晶体振荡器产生，在保证测量精度不变的前提下，提高标准信号频率，可使闸门时间缩短，即提高测试速度。根据以上分析可知等精度测频法具有三个特点：

(1) 相对测量误差与被测频率的高低无关；

(2) 增大 Tpr 或 Fs 可以增大 Ns，减少测量频率误差，提高测量精度；

(3) 测量精度与预置门宽度和标准频率有关，与被测信号的频率无关，在预置门和常规测闸门时间相同而被测信号频率不同的情况下，等精度测量法的测量精度不变。

本实验要求设计一个等精度数字频率计测频系统，能够测量方波信号的频率，用七段数码管显示被测信号的频率。此实验将 FPGA 设计与单片机开发结合起来，用 FPGA 实现定时、分频、计数等功能，用单片机实现数据的处理。很好的弥补了 FPGA 在数据处理方面的不足，大大提高了计算的精度。使本设计能够很好的达到高精度的要求,系统的连接框图如图 4-10-3 所示。

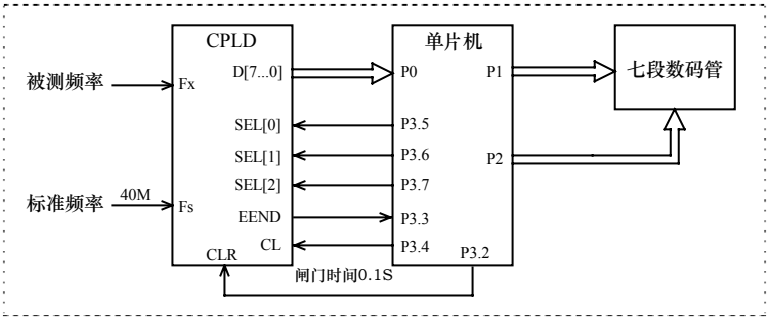


图 4-10-3 等精度测频系统连接框图

单片机的 P0 口接 8 位数据 DATA[7…0]负责读取测频数据，P1 口和 P2 口用于 LED 的动态扫描显示。P3 口控制多路通道的数据选择(如清零信号、测量标准频率计数值、测量代测频率计数值等)。DATA[7…0]为两计数器的计数值输出，两个计数器都是 32 位的二进制计数器(4 个 8 位计数值)。单片机通过 SEL[0]～SEL[2]选通端分别读出 CPLD 两个计数器的 4 个 8 位计数值(共 8 个 8 位计数值)，预置闸门时间位 0.1S， Fs 为标准频率信号输入端，此实验使用的标准信号频率是 50M；Fx 为经过整形后的被测信号输入端(输入给 FPGA 的都是脉冲信号)。

4.10.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 HPREFRE。
2. 选择 File→New 菜单，创建图形设计文件，如图 4-10-4 所示设计数字频率计顶层电路图。

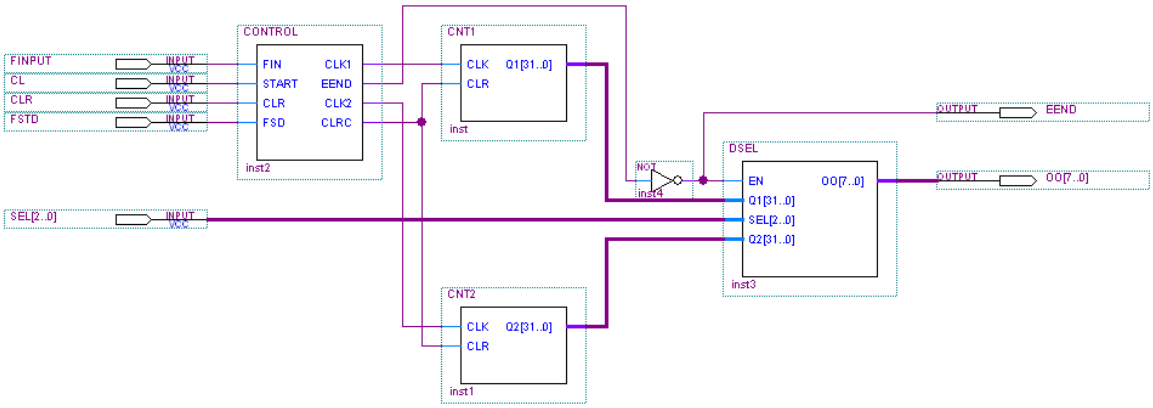


图 4-10-4 数字频率计顶层设计电路图

3. 图 4-10-4 中 CONTROL 模块是数字频率计的控制模块。VHDL 源程序如下：

```
--测量频率控制模块  CONTROL.VHD
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY CONTROL IS
    PORT(FIN,START,CLR,FSD: IN STD_LOGIC;
         CLK1,EEND,CLK2,CLRC : OUT STD_LOGIC);
END ENTITY CONTROL;
ARCHITECTURE ART OF CONTROL IS
  SIGNAL QQ1: STD_LOGIC;
     BEGIN
       PROCESS(FSD,CLR,START) IS
          BEGIN
            IF CLR='1' THEN QQ1<='0';
            ELSIF (FIN'EVENT AND FIN='1')
            THEN QQ1<=START;
            END IF;
       END PROCESS;
      CLRC<=CLR;   EEND<=QQ1;
      CLK1<=FIN AND QQ1;
      CLK2<=FSD AND QQ1;
END ARCHITECTURE ART;
```

4. 在图 4-10-4 中 CNT1 模块和 CNT2 模块实现标准频率和待测频率的 32 位计数器功能，DSEL 模块实现了分时读出计数值，送到单片机进行计算的功能。根据前述实验内容自行设计 CNT1、CNT2、DSEL 单元。
5. 选择 Tools→Compiler Tool 菜单，编译源文件。编译无误后建立仿真波形文件进行仿真。
6. 分析仿真结果，仿真正确后选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 4-10-1 所示。

表 4-10-1　数字频率计实验引脚分配表

引脚名称	引脚顺序	引脚名称	引脚顺序
DATA [0]	PIN_2	FIN	PIN_28
DATA [1]	PIN_3	CLR	PIN_175
DATA [2]	PIN_4	EEND	PIN_176
DATA [3]	PIN_5	CL	PIN_177
DATA [4]	PIN_6	SEL[0]	PIN_180
DATA [5]	PIN_7	SEL[1]	PIN_181
DATA [6]	PIN_8	SEL[2]	PIN_182
DATA [7]	PIN_11	FSTD	PIN_153

7. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编辑，生成可以配置到 FPGA 的 SOF 文件。
8. 完成 FPGA 设计的功能后，打开 Keil C51 软件，设计单片机控制与运算程序。关于单片机的开发，本书不作详细的说明。这里只给出主要程序的流程图，如图 4-10-5、图 4-10-6 所示编写单片机程序。

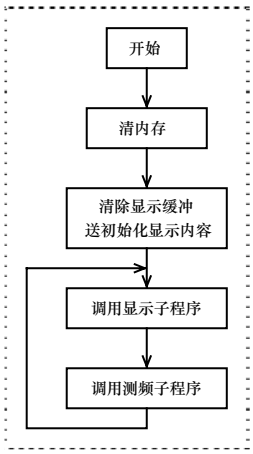


图 4-10-5 主程序流程图

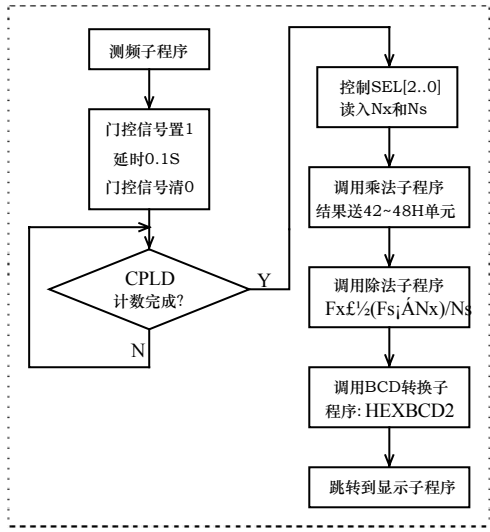


图 4-10-6 频率测量子程序流程图

9. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 4-10-7 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

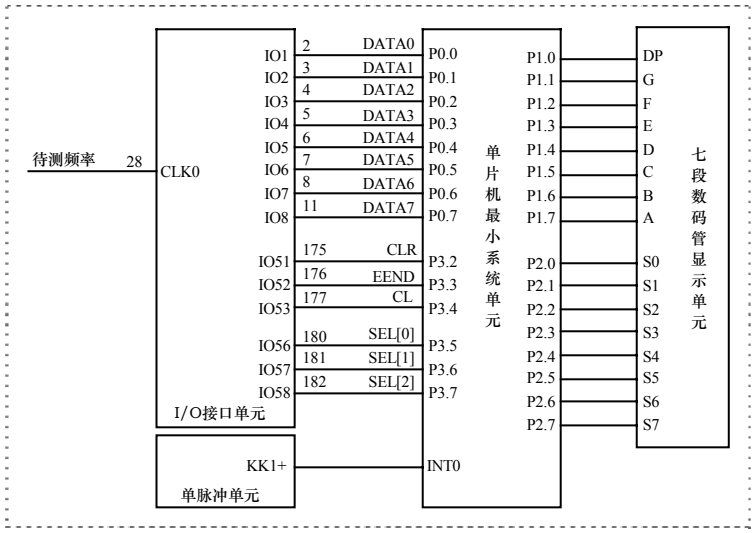


图 4-10-7 数字频率计实验接线图

10. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置
11. 配置完成后，改变输入方波的频率观察七段数码管显示的是否正确，验证设计的正确性。

4.10.5 实验扩展

在本实验实现功能的基础上，增加周期测量功能、脉宽测量功能、占空比测量功能重新进行设计和硬件验证。

第五章 SOPC 嵌入式系统实验

SOPC(System On a Programmable Chip)是指用可编程逻辑技术把整个系统放到一块硅片上。它是一种特殊的嵌入式系统。一方面它是片上系统(SOC)，即由单个芯片完成整个系统的主要逻辑功能；另一方面它是可编程系统，具有灵活的设计方式，可以裁剪、扩充、升级并具备软硬件在系统可编程的功能。SOPC 技术结合了 SOC 和 FPGA 各自的优点，一般具有以下特征：

1. 至少包含一个嵌入式内核；
2. 具有小容量片内高速 RAM 资源；
3. 丰富的 IP Core 资源可供用户选择；
4. 足够的片上可编程逻辑资源；
5. 具有处理器调试接口和 FPGA 编程接口；
6. 可包含部分可编程模拟电路；
7. 单芯片、低功耗、微封装。

2000 年 Altera 公司发布了第一款用于可编程逻辑器件的可配置的软核 Nios 处理器。Nios 是基于 RISC 技术的通用嵌入式处理器芯片软内核，它特别为可编程逻辑进行了优化设计，也为可编程单芯片系统设计了一套综合的解决方案。2004 年 6 月 Altera 公司推出了真正的 32 位软核心 NiosII 嵌入式处理器。Quartus II 软件包括的 SOPC Builder 工具是专为基于 NIOS II 处理器进行 SOPC 开发的软件工具。SOPC Builder 针对可编程系统(SOPC)的各种应用自动完成 IP 核（包括嵌入式处理器、协处理器、外设、存储器和用户设定的逻辑）的添加、参数设置和连接等操作。SOPC Builder 节约了原先系统集成工作中所需要的大量时间，使设计人员能够在几分钟内将概念转化成为真正的系统。

本章提供了五个实验项目作为 SOPC 嵌入式系统实验，通过这些实验使学生了解 SOPC 技术的概念，掌握 SOPC Builder 工具及 Nios II IDE 集成开发环境的使用方法、NIOS II 处理器系统的设计方法及 NIOS II 处理器的系统结构。

5.1 HELLO LED 实验

5.1.1 实验目的：

1. 了解基于 FPGA 的嵌入式系统开发的步骤；
2. 了解 Nios II 软核心处理器的系统结构。

5.1.2 实验设备：

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

5.1.3 实验内容：

本实验在 SOPC Builder 中设计一个基于 Nios II 软核的最小的嵌入式系统，使用 Nios II IDE 调试软件提供的 Hello LED 模板程序进行实验。在 TD-EDA 教学实验系统上观察实验现象。

5.1.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 SOPC1。
2. 选择 File→New 菜单，创建图形设计文件 SOPC1.BDF，打开图形编辑器界面。
3. 选择菜单 Tools→SOPC Builder，如图 5-1-1 所示，启动 SOPC Builder 工具。

图 5-1-1 启动 SOPC Builder 工具

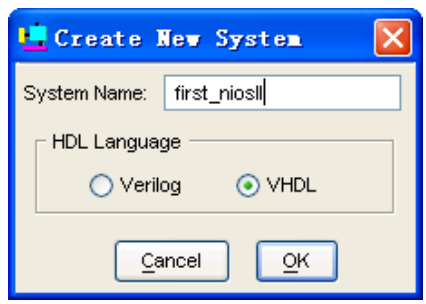


图 5-1-2 Create New System 设置向导窗口

4. 如图 5-1-2 所示设置向导界面的 System Name 中输入 NIOSII 系统名称：first_niosII，并选择 VHDL 或 Verilog 硬件描述语言。
5. 点击”OK”按钮，显示出 Altera SOPC Builder –first_niosII 窗口和系统元件列表。在 SOPCBuilder 中的 Target 下拉列表中选择：TD_SOPC_Board。

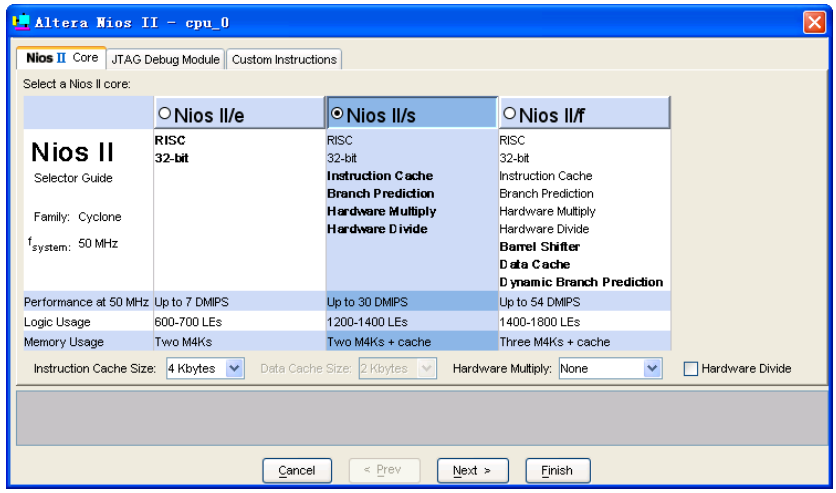


图 5-1-3 Nios II Core 项目栏

6. 在 Avalon 模块下选择 Nios II Processor-Altera Corporation，点击”Add…”按钮，出现如图 5-1-3 所示的 Nios II 的设置向导，在 Nios II Core 栏中选择 Nios II/s 选项，在 Instruction Cache Size 中选择 4Kbytes。
7. 点击”JTAG Debug Module”标签页，如图 5-1-4 所示选择 Level2 选项。

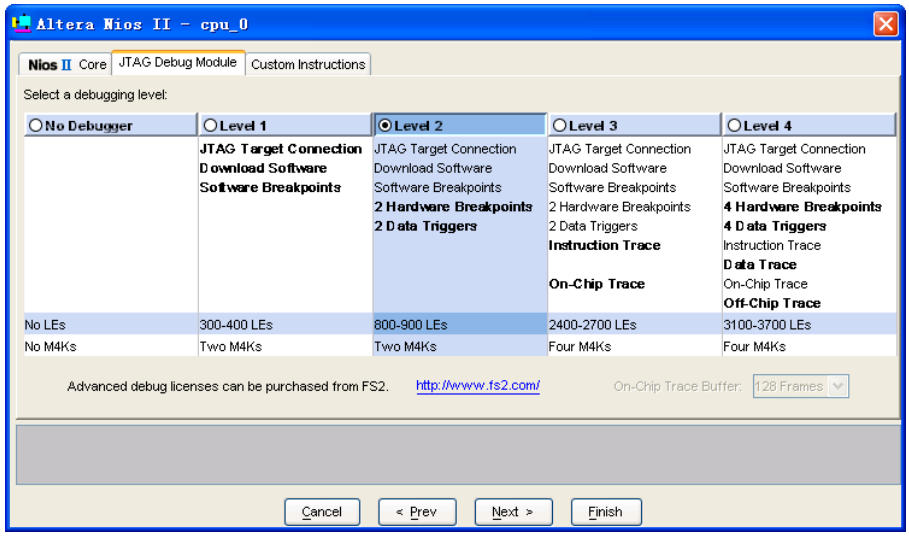


图 5-1-4 JTAG Debug Module 标签页

8. 点击”Finish”按钮返回 SOPC Builder 窗口。鼠标右键单击”CPU_0”，在如图 5-1-5 所示的菜单中选择”Rename”，将 CPU_0 重命名为 CPU。

注意：对模块命名是应遵循以下规则：

(1) 名字最前面应该使用英文；

(2) 能使用的字符只有英文字母、数字和”_”；

(3) 不能连续使用”_”符号，名字的最后也不能使用”_”。

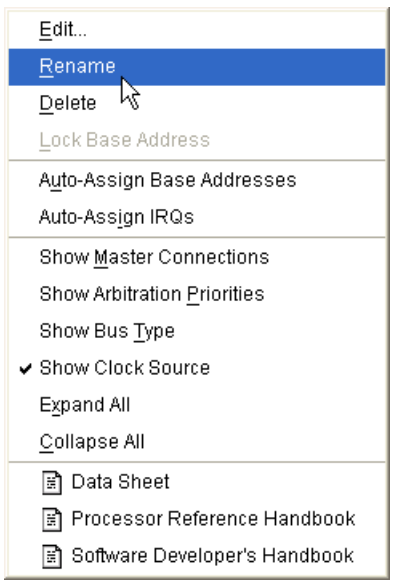


图 5-1-5 器件设备右键菜单

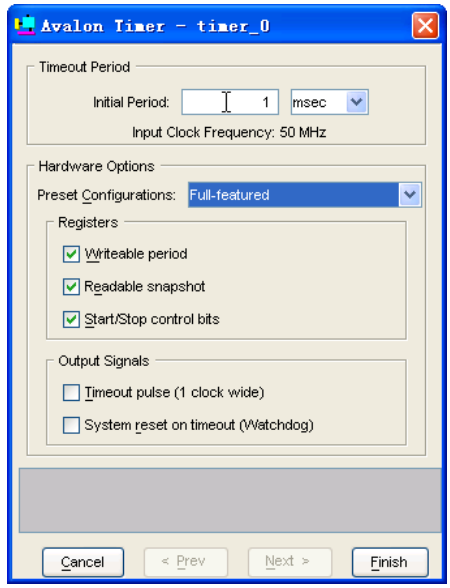


图 5-1-6 定时器设置向导

9. 在 Avalon 模块中选择 Other 下的 Interval timer，点击”Add…”按钮，添加定时器。定时器对于 HAL 系统库中的器件驱动非常有用，比如 JTAG UART 驱动使用定时器来实现 10 秒钟的暂停。
10. 在如图 5-1-6 所示的定时器设置向导的 Initial Period 选项中选择 1 msec，在 Preset Configurations 选项中选择 Full-featured。点击”Finish”按钮完成定时器的设置，返回 SOPC Builder 窗口。将 timer_0 重命名为 system_timer。
11. 在 Avalon 模块中选择 Communication 下的 JTAG UART，点击”Add…”按钮，添加 JTAG UART 接口。JTAG UART 是 Nios II 系统嵌入式处理器新添加的接口元件，通过内嵌在 Altera FPGA 内部的联合测试行动组(JTAG ——Joint Test Action Group)电路，在 PC 主机和 SOPC Builder 系统之间进行串行字符流通信。
12. 在如图 5-1-7 所示的 JTAG UART 设置向导中保持系统默认的选项，点击”Finish”按钮完成 JTAG UART 的设置，返回 SOPC Builder 窗口。将 jtag_uart_0 重命名为 jtag_uart。
13. 在 Avalon 模块中选择 Memory 下的 On-Chip Memory(Ram or Rom)，点击”Add…”按钮，添加 FPGA 片内的 SRAM 存储器。
14. 如图 5-1-8 所示的片内 RAM 存储器设置向导中，确定存储器大小为 2KB。点击”Finish”按钮完成片内 RAM 的设置，返回 SOPC Builder 窗口。将 onchip_memory_0 重命名为 onchip_memory。

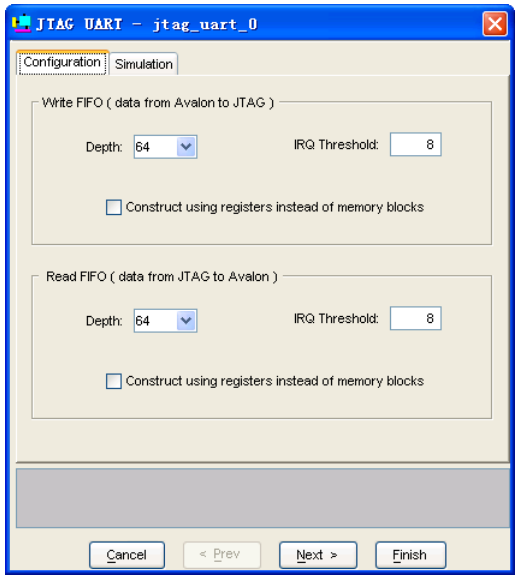


图 5-1-7 JTAG UART 设置向导

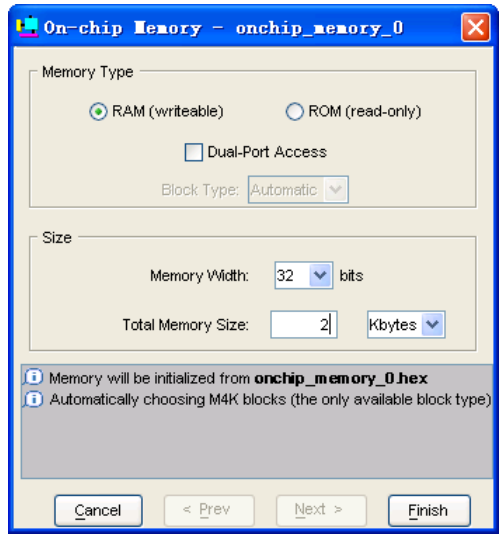


图 5-1-8 片内 RAM 存储器设置向导

15. 在 Avalon 模块中选择 Memory 下的 EPCS Serial Flash Controller，点击”Add…”按钮，添加串行配置存储器芯片 EPCS1。
16. 如图 5-1-9 所示的串行配置存储器设置向导中，确定存储器单元为 U5。点击”Finish”按钮完成串行配置存储器的设置，返回 SOPC Builder 窗口。

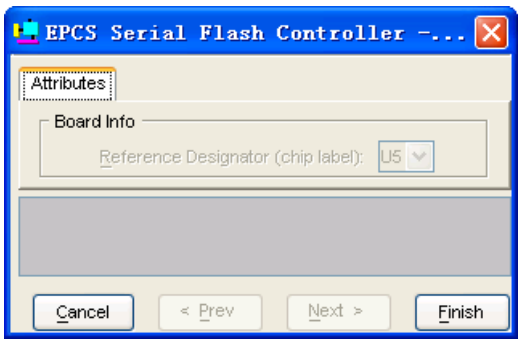


图 5-1-9 串行配置存储器设置

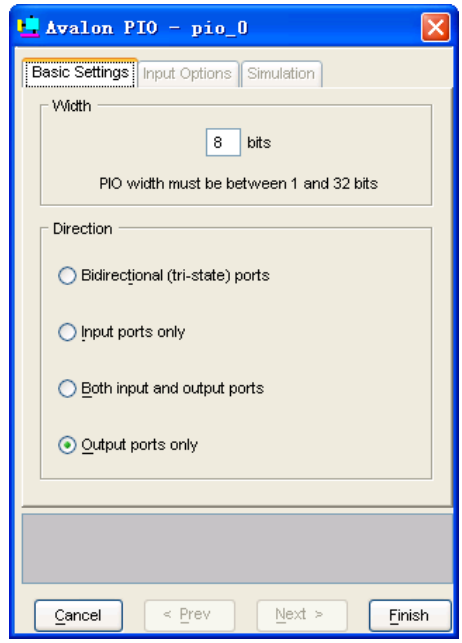


图 5-1-10 I/O 设置向导

17. 在 Avalon 模块中选择 Other 下的 PIO(Parallel I/O)，点击”Add…”按钮，如图 5-1-10 所示的 Avalon PIO-pio_0 设置向导中，添加 IO 输出接口。将 pio_0 重命名为 led_pio。
18. 至此已完成 Nios II 系统模块的设置，SOPC Builder 会给用户的 Nios II 系统模块分配默认的基地址，用户也可以更改这些默认值。此实验中将 epcs_controller 的基地址设置为 0x00000000。

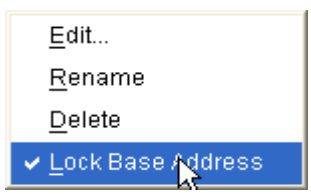


图 5-1-11 Module 菜单

图 5-1-12 System 菜单

19. 在 SOPC Builder 的模块表中点击 epcs_controller 的 BASE，输入 0x00000000 后回车。如图 5-1-11 所示，选择 Module 菜单中的 Lock Base/Address，会在 epcs_controller 的基地址旁出现一个锁子的图标。
20. 如图 5-1-12 所示，选择 System 菜单中的 Auto -Assign Base Addresses。Auto -Assign Base Addresses 可以使 SOPC Builder 给其他没有锁定的地址重新分配地址，从而解决地址映射冲突的问题。

21. 如图 5-1-13 所示，显示了最终的系统配置及其地址映射。

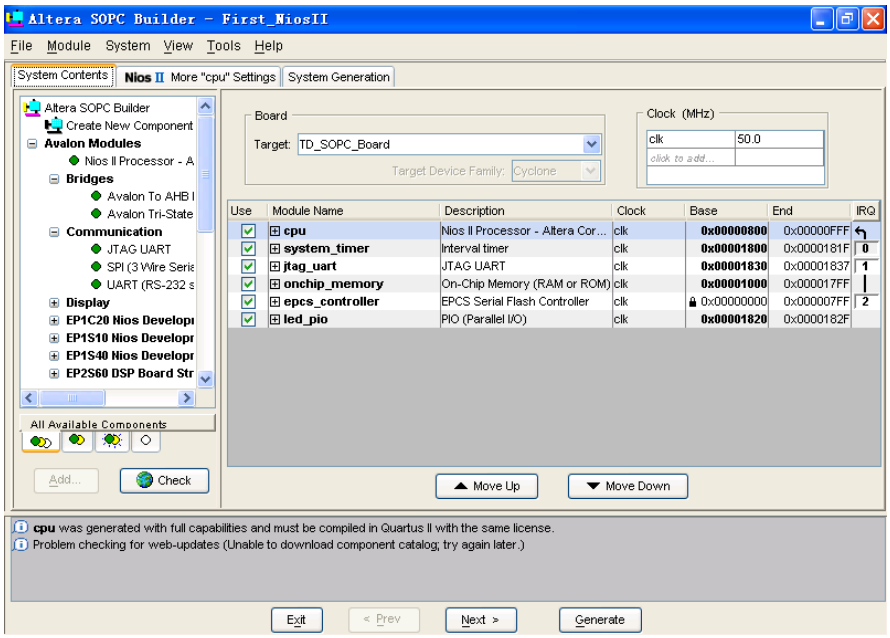


图 5-1-13　最终的 Nios II 系统配置及其地址映射

22. 选择 Nios II More"cpu"Settings 选项，在如图 5-1-14 所示的系统设置中，对系统进行进一步的设置，指定系统的复位地址和执行地址。

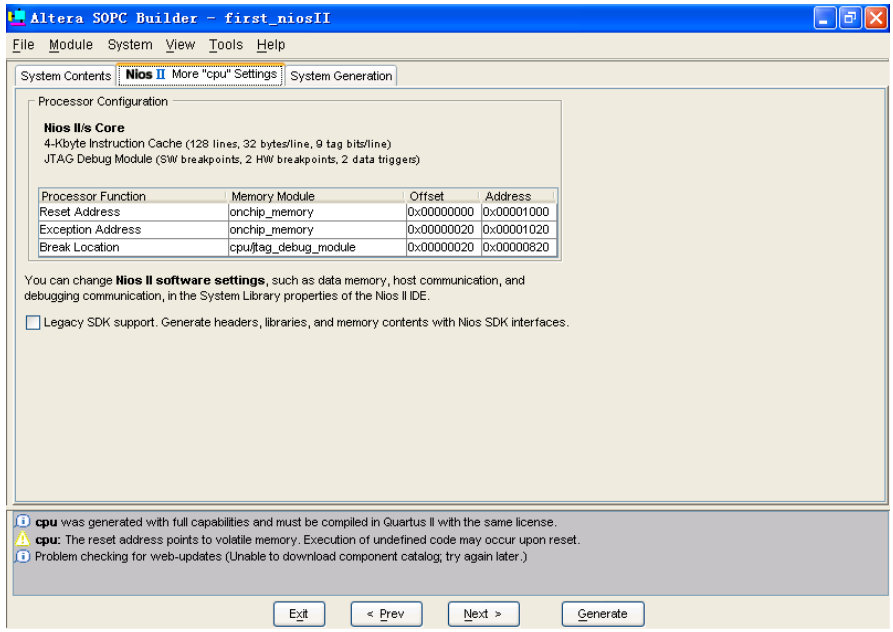


图 5-1-14　系统设置

23. 选择”System Generation”选项，在 SOPC Builder 生成页中选中 HDL 选项；如果安装了 ModelSim 软件并需要仿真此设计，也可以选择 Simulation 选项。
24. 点击”Generate”按钮，SOPC Builder 根据用户设定的不同选项，在生成的过程中执行不同的操作。如图 5-1-15 所示，系统生成完成后点击”EXIT”按钮退出 SOPC Builder。

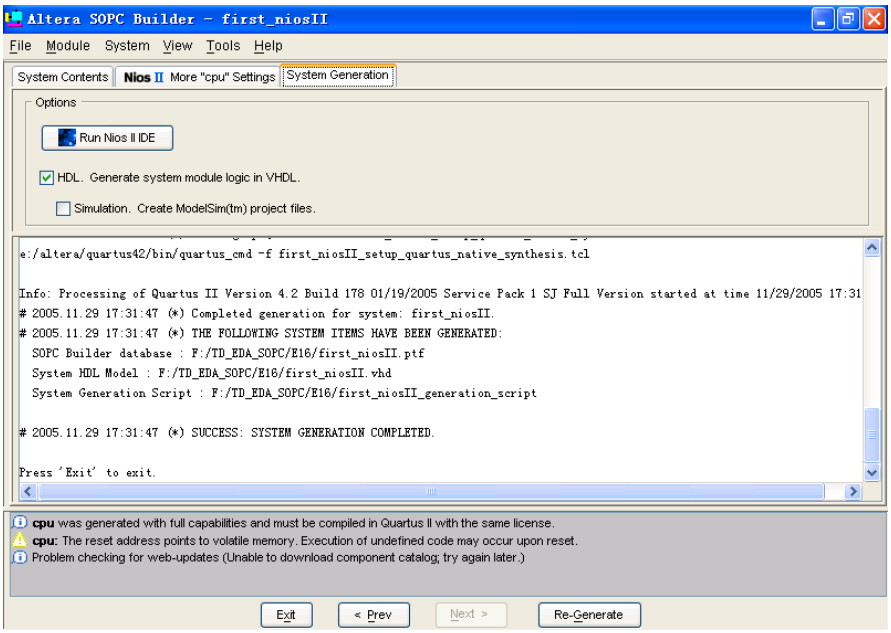


图 5-1-15 SOPC Builder 生成页

25. 在 Nios II 系统生成过程中，SOPC Builder 会生成用户系统模块的图标(Symbol)，可以将该图标像添加其他 Quartus II 图标一样添加到当前项目的 BDF 文件中。在 Quartus II 软件中双击 BDF 文件窗口，如图 5-1-16 所示的 Symbol 对话框，在 Project 目录下选择 first_niosII，会出现一个代表所建立的 NiosII 系统的图标。点击”OK”按钮，将其放入 BDF 文件中。

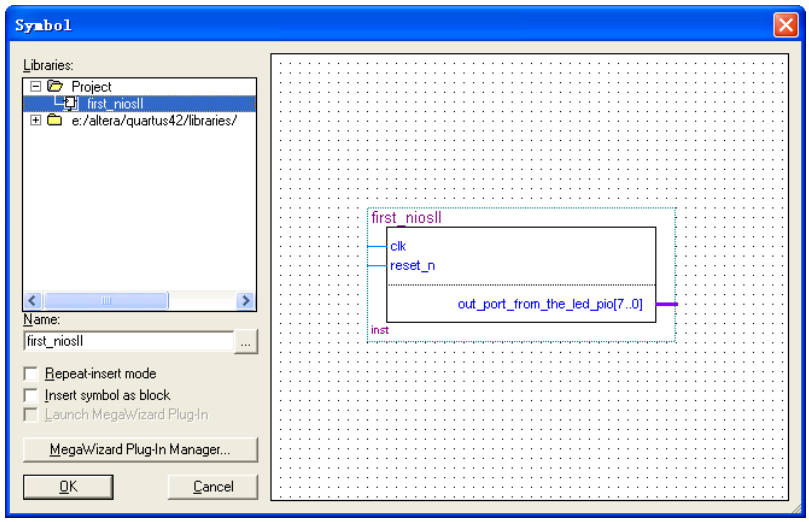


图 5-1-16 Symbol 对话框

26. 将 first_niosII 模块与输入(input)、输出(output)接口连接， 在 Quartus II 软件中选择 Assignments→Device，在弹出的 Settings 窗口中点击”Device&Pin Options…”按钮在”Device&Pin Options”对话框中选择”Unused Pin”标签页，在如图 5-1-17 所示的芯片引脚对话框中选择第一项”As inuts,tri-stated”选项，将所有无用的引脚置为输入，三态。

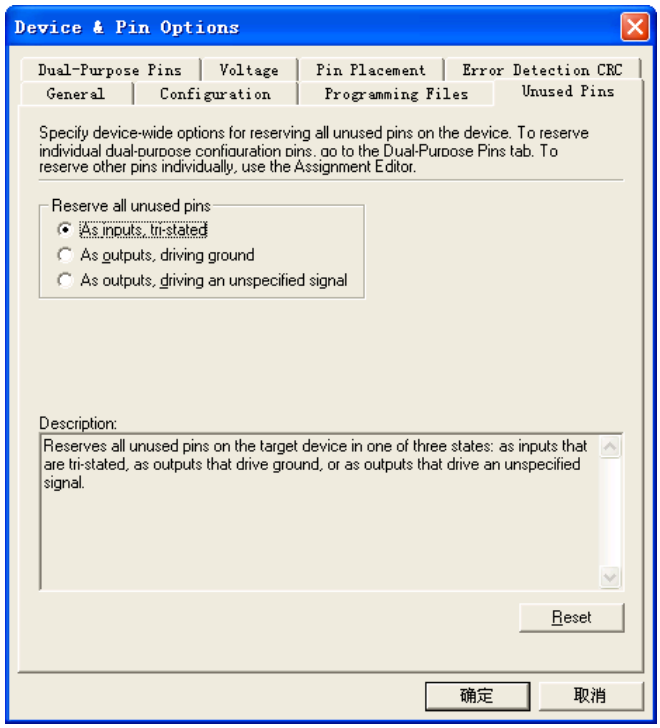


图 5-1-17 芯片引脚设置对话框

27. 选择 Processing→Start → Start Analysis&Synthesis 菜单，对系统进行分析、综合。
28. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 5-1-1 所示。

表 5-1-1 HELLO LED 实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_153	LED [3]	PIN_5
RST	PIN_131	LED [4]	PIN_6
LED [0]	PIN_2	LED [5]	PIN_7
LED [1]	PIN_3	LED [6]	PIN_8
LED [2]	PIN_4	LED [7]	PIN_11

29. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编译，生成可以配置到 FPGA 的 SOF 文件。
30. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 5-1-18 所示进行实验接线，将 ByteBlaster

II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

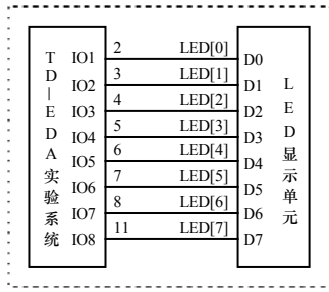


图 5-1-18 HELLO LED 实验接线图

31. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置，至此硬件设计工作已经完成。
32. 运行 NiosII IDE 软件开发环境，如图 5-1-19 所示。

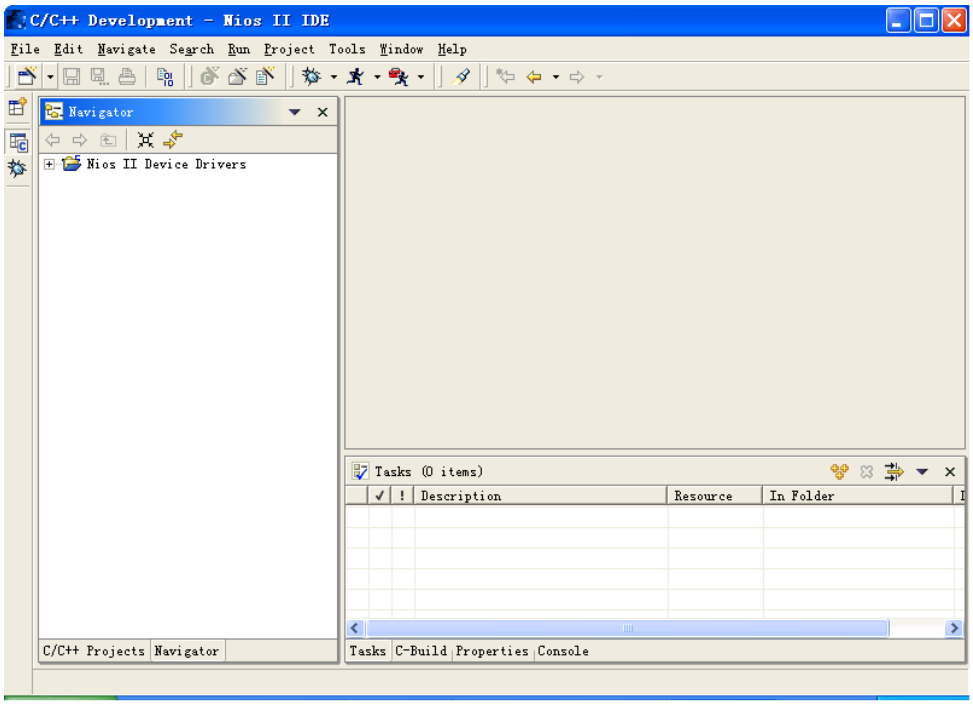


图 5-1-19 Nios II IDE 软件开发环境界面

33. 选择 File→New→C/C++ Application 菜单，建立一个新工程，如图 5-1-20 所示。
34. 在 Nios II IDE 新工程向导的 Name 中填入软件工程的名称：hello_led；在 SOPC Builder System 中选择硬件配置文件(PTF 文件)所在的目录，指向当前硬件设计系统；在 Select Project Template 中选择使用的模板 Hello LED，完成后点击”Finish”按钮。

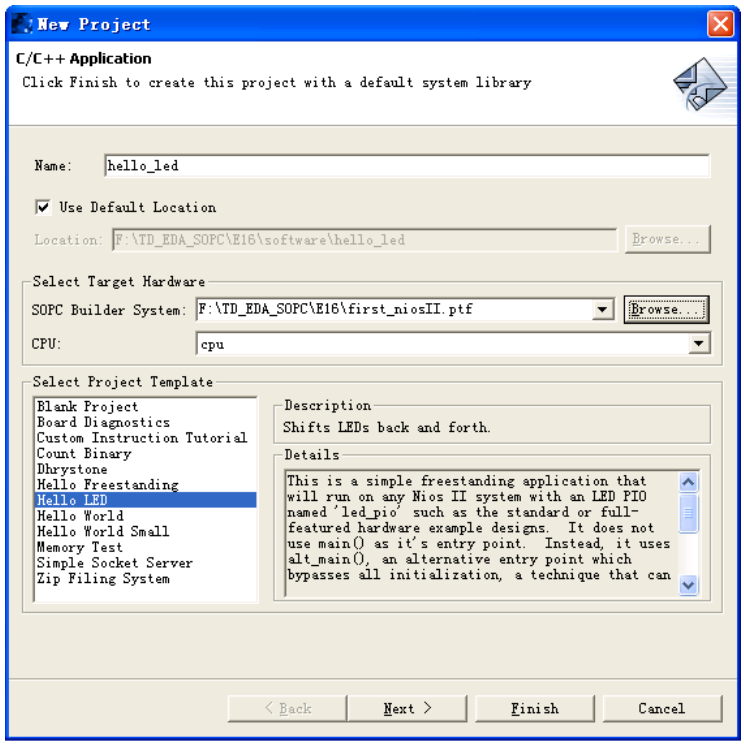


图 5-1-20 Nios II IDE 新工程向导

35. 在 Nios II IDE 文本编辑器中描述了 Hello LED 模板提供的源程序 hello_led.C，源程序如下：

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
int main (void) __attribute__ ((weak, alias ("alt_main")));
int alt_main (void)
{ alt_u8 led = 0x2;
  alt_u8 dir = 0;
  volatile int i;
  while (1)
  { if (led & 0x81)
    {dir = (dir ^ 0x1);
   }
    if (dir)
    { led = led >> 1;}
    else
    {led = led << 1;}
```

```
        IOWR_ALTERA_AVALON_PIO_DATA(LED_PIO_BASE, led);
        i = 0;
        while (i<200000)
          i++;
    }
    return 0;
}
```

36. 程序编写完成后，选择菜单 Project→Build Project 对工程进行编辑，在如图 5-1-21 所示的 C-Builder 窗口中看到编译结果。

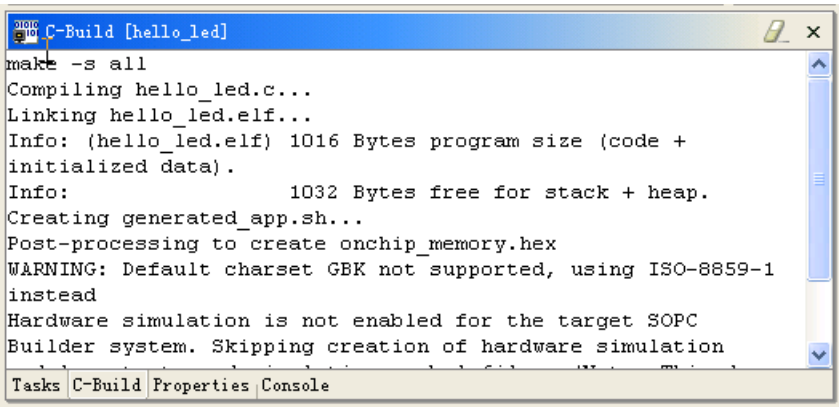


图 5-1-21 C-Builder 窗口

37. 编译完成后如图 5-1-22 所示，选择 Run→Run As→Nios II Hardware 菜单，运行程序，将程序下载后可以在开发板上观察到实验现象。

图 5-1-22 运行(Run)窗口

38. 选择菜单 Run→Debug As→Nios II Hardware 进行调试。通过调试可以观察变量、设置断点、查看存储器内容等。此实验中在程序 i=0 处设置断点，运行程序到断点，可以看到 LED 按照顺序依次点亮。Nios II IDE 软件提供了丰富的调试手段，在使用中用户需要多学多用。

5．2 外部 SRAM 扩展实验

5.2.1 实验目的：

1. 了解基于 FPGA 的嵌入式系统开发的步骤；
2. 了解 Nios II 软核处理器的系统结构;
3. 了解基于 Nios II 处理器的程序设计过程。

5.2.2 实验设备：

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

5.2.3 实验内容：

本实验在 SOPC Builder 中配置 Nios II 系统+外部 SRAM，设计一个基于 Nios II 软核的嵌入式系统，使用 Nios II IDE 调试软件编写程序实现交通灯控制的功能。在 EDA 教学实验系统上观察实验现象。

5.2.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 SOPC2。
2. 选择 File→New 菜单，创建图形设计文件 SOPC2.BDF，打开图形编辑器界面。
3. 选择菜单 Tools→SOPC Builder，启动 SOPC Builder 工具。在 Create New System 设置向导窗口的 System Name 中输入系统名称：Traffic，并选择 VHDL 硬件描述语言。
4. 在 SOPC Builder 中的 Target 下拉列表中选择：TD_SOPC_Board。在 Altera SOPC Builder –TRAFFIC 窗口中设计此实验所需的 NiosII 系统。
5. 在 Avalon 模块下分别添加 Nios II Processor、Interval timer、 JTAG UART、EPCS Serial Flash Controller，分别重命名为：cpu、system_timer、jtag_uart、epcs_controller。
6. 在 Avalon 模块中选择 Memory 下的 IDT71V416 SRAM，点击”Add…”按钮，添加 SRAM 存储器，SOPC 开发板使用的 SRAM 存储器 IS61LV25616 与 IDT71V416 兼容。
7. 如图 5-2-1 所示的 SRAM 存储器设置向导，在 Attributes 栏中，确定存储器大小为 1024KB。点击”Finish”按钮完成 SRAM 的设置，返回 SOPC Builder 窗口。将 sram_0 重命名为 ext_ram。
8. 在 Avalon 模块中选择 Bridges 下的 Avalon Tri-State Bridge，点击”Add…”按钮，添加外部 RAM 总线。SOPC Builder 是使用 Avalon 接口来连接片上元件和 Avalon 主从端口的。在 SOPC 开发板上，要实现 Nios II 系统与 FPGA 片外的存储器通信，用户就必须在 Avalon 总线和连接的外部存储器的总线之间添加一个桥。
9. 如图 5-2-2 所示的 Avalon Tri-state Bridge 设置向导，默认情况下 Registered 选项是选中的，点击”Finish”按钮完成总线的设置，返回 SOPC Builder 窗口。将 tri_state_bridge_0 重命名为 ext_ram_bus。

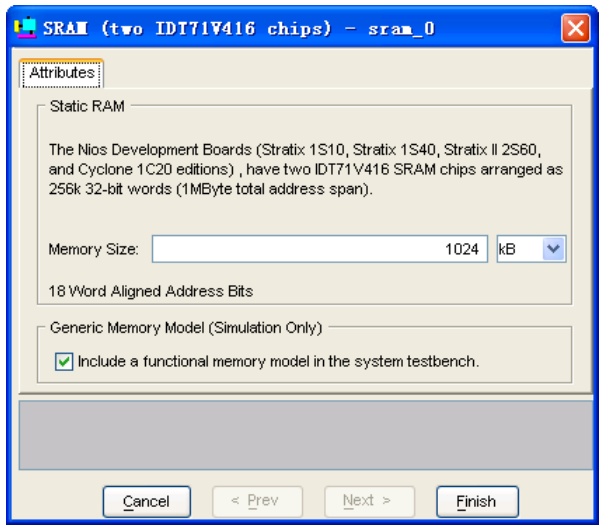


图 5-2-1 SRAM 存储器设置向导

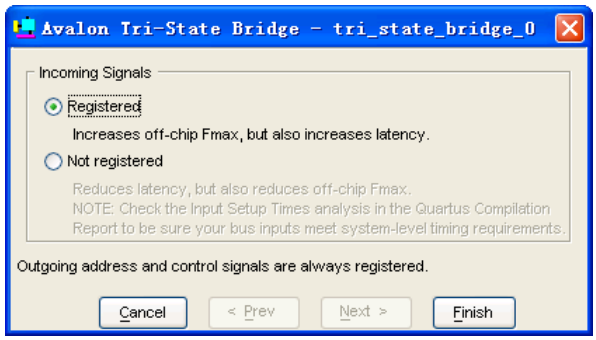


图 5-2-2 Avalon Tri-state Bridge 设置向导

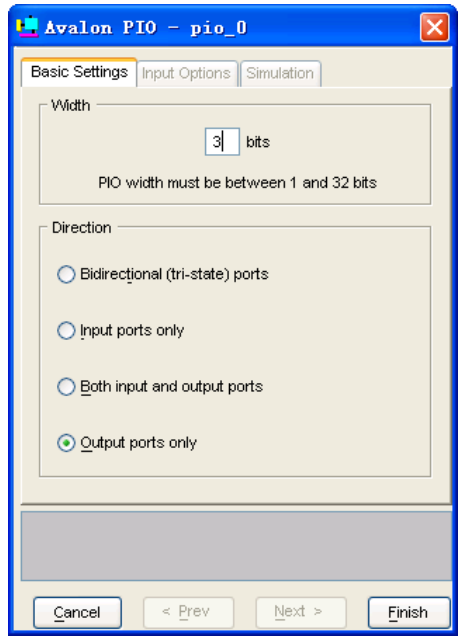


图 5-2-3 I/O 设置向导

10. 在 Avalon 模块中选择 Other 下的 PIO(Parallel I/O)，点击”Add…”按钮，在如图 5-2-3 所示的 Avalon PIO-pio_0 设置向导中，添加 IO 输出接口。将 pio_0 重命名为 North_South_PIO。继续添加 PIO(Parallel I/O)，分别重命名为 West_East_PIO(3bits Output)、SEVEN_SEG_PIO(8bits Output)、SEG_PIO(2bits Output)。
11. 在 SOPC Builder 的模块表中点击 ext_ram 的 BASE，输入 0x00000000 后回车，选择 Module 菜单中的 Lock Base/Address，在 ext_ram 的基地址旁出现一个锁子的图标。

12. 选择 System 菜单中的 Auto -Assign Base Addresses，对系统的基地址进行重新分配；选择 System 菜单中的 Auto -Assign IRQs，对系统的中断进行重新分配。
13. 如图 5-2-4 所示，显示了最终的系统配置及其地址映射。

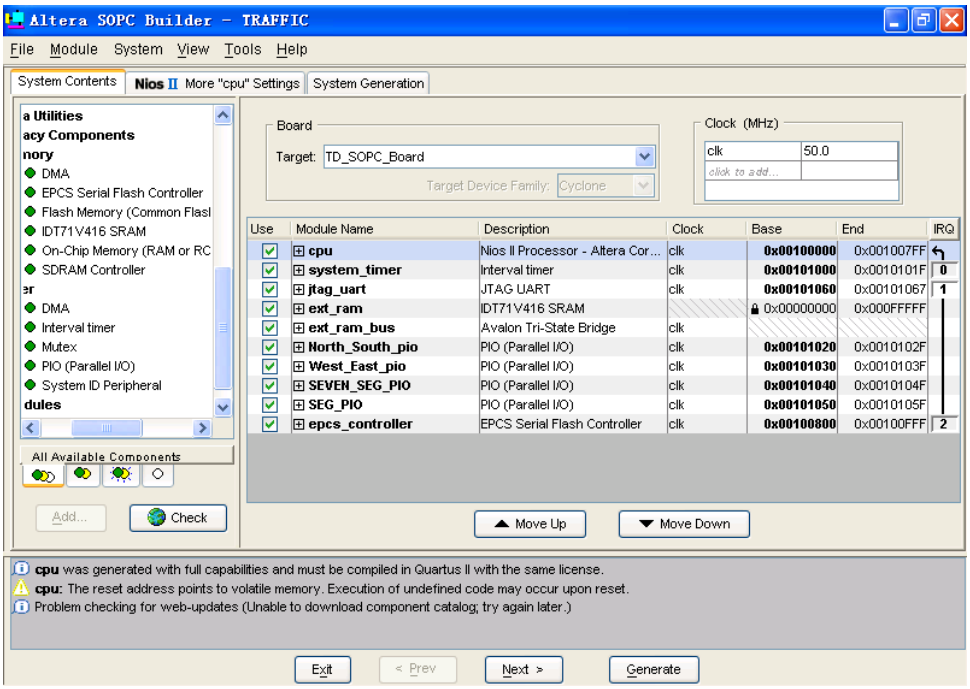


图 5-2-4　最终的 Nios II 系统配置及其地址映射

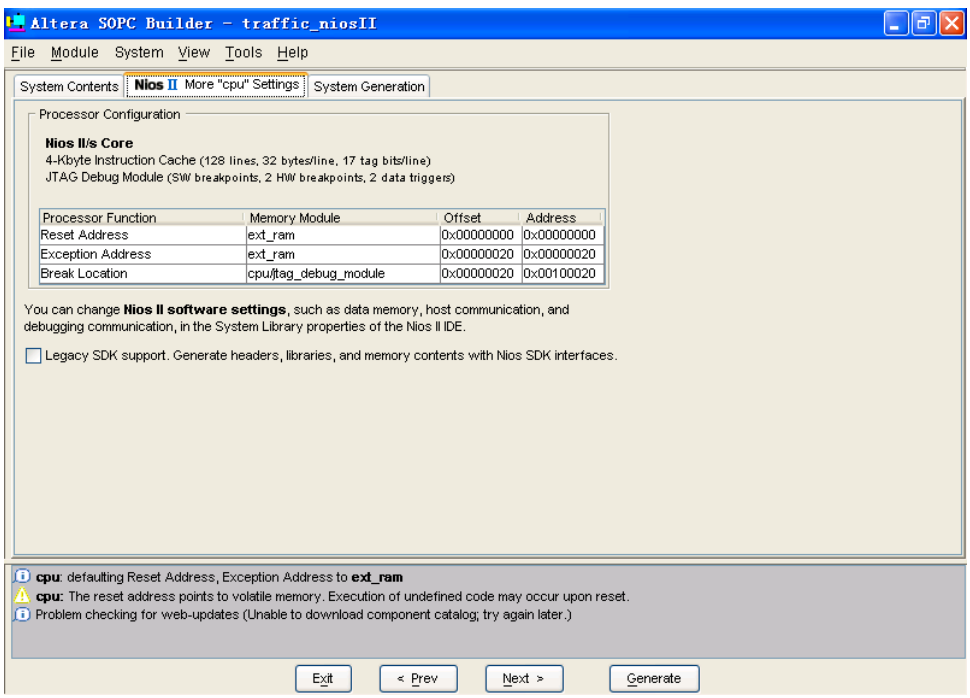


图 5-2-5　系统设置

14. 选择 Nios II More”cpu”Settings 选项，在如图 5-2-5 所示的系统设置中，对系统进行进一步的设置，指定系统的复位地址和执行地址为 ext_ram。
15. 选择 System Generation 选项,在 SOPC Builder 生成页中选中 HDL 选项。点击”Generate”生成硬件系统文件，完成后点击”EXIT”按钮退出 SOPC Builder。
16. 在 Quartus II 软件中双击 BDF 文件窗口，在 Symbol 对话框中点击 Project 目录，选择 TRAFFIC，将其放入图形设计文件窗口。
17. 将 TRAFFIC 模块与输入(input)、输出(output)、双向(bidir)接口连接， 在 Quartus II 软件中将所有无用的引脚置为输入状态，三态。
18. 选择 Processing→Start → Start Analysis&Synthesis 菜单，对系统进行分析、综合。
19. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 5-2-1 所示。

表 5-2-1 外部 SRAM 扩展实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_153	SHA_D [5]	PIN_104
RST	PIN_131	SHA_D [6]	PIN_101
SHA_A [2]	PIN_128	SHA_D [7]	PIN_100
SHA_A [3]	PIN_127	SHA_D [8]	PIN_78
SHA_A [4]	PIN_126	SHA_D [9]	PIN_79
SHA_A [5]	PIN_125	SHA_D [10]	PIN_82
SHA_A [6]	PIN_124	SHA_D [11]	PIN_83
SHA_A [7]	PIN_98	SHA_D [12]	PIN_84
SHA_A [8]	PIN_95	SHA_D [13]	PIN_85
SHA_A [9]	PIN_94	SHA_D [14]	PIN_86
SHA_A [10]	PIN_93	SHA_D [15]	PIN_87
SHA_A [11]	PIN_88	SHA_D [16]	PIN_66
SHA_A [12]	PIN_106	SHA_D [17]	PIN_67
SHA_A [13]	PIN_107	SHA_D [18]	PIN_68
SHA_A [14]	PIN_108	SHA_D [19]	PIN_73
SHA_A [15]	PIN_143	SHA_D [20]	PIN_74
SHA_A [16]	PIN_113	SHA_D [21]	PIN_75
SHA_A [17]	PIN_132	SHA_D [22]	PIN_76
SHA_A [18]	PIN_133	SHA_D [23]	PIN_77
SHA_A [19]	PIN_134	SHA_D [24]	PIN_58
SHA_D [0]	PIN_122	SHA_D [25]	PIN_59
SHA_D [1]	PIN_121	SHA_D [26]	PIN_60
SHA_D [2]	PIN_120	SHA_D [27]	PIN_61
SHA_D [3]	PIN_119	SHA_D [28]	PIN_62
SHA_D [4]	PIN_105	SHA_D [29]	PIN_63

续表 5-2-1 外部 SRAM 扩展实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
SHA_D [30]	PIN_64	WE [1]	PIN_18
SHA_D [31]	PIN_65	WE [2]	PIN_19
SRAM_BE[0]	PIN_116	SEVEN [0]	PIN_2
SRAM_BE[1]	PIN_117	SEVEN [1]	PIN_3
SRAM_BE[2]	PIN_114	SEVEN [2]	PIN_4
SRAM_BE[3]	PIN_115	SEVEN [3]	PIN_5
SRAM_CS	PIN_123	SEVEN [4]	PIN_6
SRAM_WE	PIN_99	SEVEN [5]	PIN_7
SRAM_OE	PIN_118	SEVEN [6]	PIN_8
NS [0]	PIN_12	SEVEN [7]	PIN_11
NS [1]	PIN_13	SEG [0]	PIN_20
NS [2]	PIN_14	SEG [1]	PIN_21
WE[0]	PIN_17		

20. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编译，生成可以配置到 FPGA 的 SOF 文件。
21. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 5-2-6 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。仔细检查确保接线无误后打开电源。

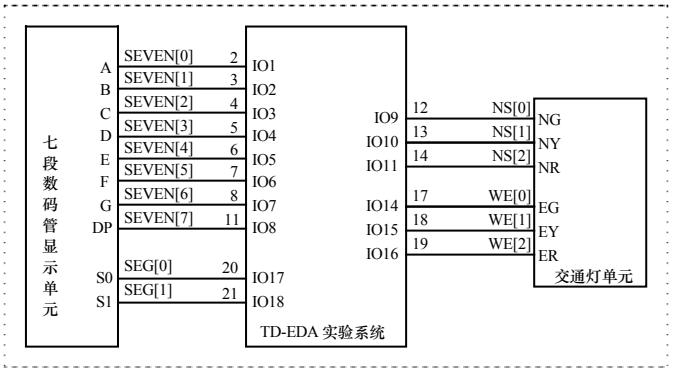


图 5-2-6 外部 SRAM 扩展实验接线图

22. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置，至此硬件设计工作已经完成。
23. 运行 NiosII IDE 软件开发环境，选择 File→New→C/C++ Application 菜单，建立新工程。在 Nios II IDE 新工程向导的 Name 中输入软件工程名称：TRAFFIC_NIOSII；在 SOPC Builder System 中选择硬件配置文件(PTF 文件)所在的目录；在 Select Project Template 中选择使用的模板 Blank Project，完成后点击”Finish”按钮。
24. 选择 File→New→File 菜单，新建文件 TRAFFIC.C，在 Nios II IDE 文本编辑器中编写程序，源程序如下：

```
#include "alt_types.h"
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
static alt_u8 count;
static alt_u8 nsled = 0x1;
static alt_u8 seg ;
static alt_u8 weled =0x4;
static void sevenseg_set_hex(int hex)
{    static alt_u8 segments[16] = {
       0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, /* 0-9*/
       0x77, 0x7A, 0x39, 0x6E, 0x79, 0x71 };                       /* a-f */
   unsigned int data = segments[hex%10] |(segments[hex/10]<<8);
 IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE, data);
}
static void count_led()
{#ifdef NORTH_SOUTH_PIO_BASE
   IOWR_ALTERA_AVALON_PIO_DATA(NORTH_SOUTH_PIO_BASE, nsled);
#endif}
static void count_led2()
{#ifdef WEST_EAST_PIO_BASE
   IOWR_ALTERA_AVALON_PIO_DATA(WEST_EAST_PIO_BASE, weled);
#endif}
static void count_sevenseg()
{#ifdef SEVEN_SEG_PIO_BASE
   sevenseg_set_hex(count);
#endif}
static void seven_seg()
{#ifdef SEG_PIO_BASE
   IOWR_ALTERA_AVALON_PIO_DATA(SEG_PIO_BASE, seg);
#endif}
   void seg_select(int high ,int low)
     { seg=0xfe;
        seven_seg();
        count = low;
        count_sevenseg();
        usleep(2000);
```

```
    seg=0xfd;
    seven_seg();
    count = high;
    count_sevenseg();
    usleep(2000);
  }
int main(void)
  { int i=60,j,ge,shi;
    while(1)
      {   count_led();
          count_led2();
          while(i)
            {    i--;
                 ge=i%10;
                 shi=i/10;
                 for(j=0;j<250;j++)
                 seg_select(shi,ge);
            }
                 i=10;
                 nsled =nsled << 1;
                 count_led();
                 weled =weled >>1;
                 count_led2();
          while(i)
          {    i--;
               ge=i%10;
               shi=i/10;
               for(j=0;j<250;j++)
               seg_select(shi,ge);
          }
               i=30;
               nsled =nsled << 1;
               count_led();
               weled =weled >> 1;
               count_led2();
          while(i)
          {    i--;
               ge=i%10;
               shi=i/10;
```

```
            for(j=0;j<250;j++)
            seg_select(shi,ge);
        }
            i=10;
            nsled =nsled >> 1;
            count_led();
            weled =weled << 1;
            count_led2();
        while(i)
        {   i--;
            ge=i%10;
            shi=i/10;
            for(j=0;j<250;j++)
            seg_select(shi,ge);
        }
            i=60;
            nsled =0x1;
            count_led();
            weled =0x4;
            count_led2();
    }
}
```

25. 程序编写完成后，选择菜单 Project→Build Project 对工程进行编辑。
26. 选择 Run→Run As→Nios II Hardware 菜单，运行程序，程序下载后可以在开发板上观察到实验现象。
27. 选择菜单 Run→Debug As→Nios II Hardware 对程序进行调试。

5.3 添加用户外设实验

5.3.1 实验目的:

1. 了解 Nios II 软核处理器的系统结构;
2. 了解用户自定制 Avalon 从外设的设计过程;
3. 了解基于 Nios II 处理器的程序设计过程。

5.3.2 实验设备:

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块，示波器一台。

5.3.3 实验内容:

Nios II 包括一个常用外围设备及接口库，这个库在 Altera FPGA 中可以免费使用。对于只使用系统模块内部外设的系统，用户不必考虑 Avalon 外设连接 Avalon 总线的细节。然而，大多数系统需要连接片外的存储器设备。用户必须手工将系统模块外的外设（包括片外设备）连接到 Avalon 总线端口。此外，许多系统通过三态总线将 Avalon 信号驱动到片外，从而通过同样的地址和数据物理引脚可以访问多个片外设备。

由于 NiosII 是一个位于 FPGA 中的软核处理器，用户开发的外围设备和接口可以通过引入向导轻松的引入到 Nios II 处理器系统中，为设计再利用提供了简便的方法。

本实验对 Avalon Slave 外设的设计进行介绍，设计一个 PWM 外设，PWM 的输出将连接到 FPGA 外的 LED 上，通过控制 PWM 外设寄存器可以对 LED 的亮度进行控制，也可通过示波器观察 PWM 的输出脉冲。PWM 脉冲宽度调制电路，在控制系统的应用中比较常见。PWM 的具体设计要求如下:

(1) 要求信号的周期可调。
(2) 脉冲的宽度可调。
(3) 可以控制 PWM 的输出使能。

5.3.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 AVALON_PWM。
2. 选择 File→New 菜单，创建 VHDL 描述语言设计文件 AVALON_PWM.VHD，在文本编辑器界面中编写 VHDL 程序，源程序如下:

```
--PWM 脉冲调制器组件 VHDL 源程序

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
ENTITY AVALON_PWM IS
PORT(CLK : IN STD_LOGIC;
     WR_DATA : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
     BYTE_N  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
     CS      :  IN STD_LOGIC;
     WR_N :   IN STD_LOGIC;
     ADDR :   IN STD_LOGIC;
     CLR_N : IN STD_LOGIC;
     RD_DATA: OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
     PWM_OUT: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END ENTITY AVALON_PWM;

ARCHITECTURE BEHV OF AVALON_PWM IS
  SIGNAL DIV , DUTY,COUNTER : STD_LOGIC_VECTOR(31 DOWNTO 0);
  SIGNAL PWM_ON : STD_LOGIC;
    BEGIN
      RD_DATA<=DIV WHEN ADDR='0' ELSE DUTY;
   WRITE: PROCESS(CLK,CLR_N)
     BEGIN
       IF (CLR_N='0') THEN
          DIV   <=(OTHERS=>'0');
          DUTY <=(OTHERS=>'0');
       ELSIF CLK'EVENT AND CLK='1' THEN
           IF CS='1' AND WR_N='0' THEN
              IF ADDR='0' THEN
                 IF BYTE_N(3)='0' THEN
                    DIV(31 DOWNTO 24)<=WR_DATA(31 DOWNTO 24);
                  ELSE
                    DIV(31 DOWNTO 24)<=DIV(31 DOWNTO 24);
                 END IF;
               IF BYTE_N(2)='0' THEN
                    DIV(23 DOWNTO 16)<=WR_DATA(23 DOWNTO 16);
                  ELSE
                    DIV(23 DOWNTO 16)<=DIV(23 DOWNTO 16);
                 END IF;
               IF BYTE_N(1)='0' THEN
                    DIV(15 DOWNTO 8)<=WR_DATA(15 DOWNTO 8);
                  ELSE
```

```
                    DIV(15 DOWNTO 8)<=DIV(15 DOWNTO 8);
                END IF;
                IF BYTE_N(0)='0' THEN
                    DIV(7 DOWNTO 0)<=WR_DATA(7 DOWNTO 0);
                ELSE
                    DIV(7 DOWNTO 0)<=DIV(7 DOWNTO 0);
                END IF;
            ELSE
                IF BYTE_N(3)='0' THEN
                    DUTY(31 DOWNTO 24)<=WR_DATA(31 DOWNTO 24);
                ELSE
                    DUTY(31 DOWNTO 24)<=DUTY(31 DOWNTO 24);
                END IF;
                IF BYTE_N(2)='0' THEN
                    DUTY(23 DOWNTO 16)<=WR_DATA(23 DOWNTO 16);
                ELSE
                    DUTY(23 DOWNTO 16)<=DUTY(23 DOWNTO 16);
                END IF;
                IF BYTE_N(1)='0' THEN
                    DUTY(15 DOWNTO 8)<=WR_DATA(15 DOWNTO 8);
                ELSE
                    DUTY(15 DOWNTO 8)<=DUTY(15 DOWNTO 8);
                END IF;
                IF BYTE_N(0)='0' THEN
                    DUTY(7 DOWNTO 0)<=WR_DATA(7 DOWNTO 0);
                ELSE
                    DUTY(7 DOWNTO 0)<=DUTY(7 DOWNTO 0);
                END IF;
            END IF;
        END IF;
    END IF;
END PROCESS;
DIVIDER: PROCESS(CLK,CLR_N)
  BEGIN
    IF CLR_N='0' THEN
        COUNTER<=(OTHERS=>'0');
    ELSIF   CLK'EVENT AND CLK='1' THEN
        IF COUNTER>=CONV_INTEGER(DIV) THEN
            COUNTER<=(OTHERS=>'0');
```

```
            ELSE
                COUNTER<=COUNTER+'1';
            END IF;
        END IF;
    END PROCESS;
    DUTY_CYCLE: PROCESS(CLK,CLR_N)
      BEGIN
        IF CLR_N='0' THEN
       PWM_ON<='1';
     ELSIF    CLK'EVENT AND CLK='1' THEN
      IF COUNTER>=CONV_INTEGER(DUTY) THEN
        PWM_ON<='0';
       ELSIF COUNTER="00000000000000000000000000000000" THEN
        PWM_ON<='1';
      ELSE
       PWM_ON<=PWM_ON;
       END IF;
    END IF;
     END PROCESS;
   PWM_OUT<=PWM_ON & PWM_ON & PWM_ON & PWM_ON ;
  END BEHV;
```

3. 选择 Processing→Start → Start Analysis&Synthesis 菜单，对程序进行分析、综合。
4. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 SOPC3。
5. 选择 File→New 菜单，创建图形设计文件 SOPC3.BDF，打开图形编辑器界面。
6. 选择 Tools→SOPC Builder 菜单，启动 SOPC Builder 工具。在 System Name 对话框中输入系统名称：PWM_NIOSII，并选择 VHDL 硬件描述语言。
7. 在 SOPC Builder 中的 Target 下拉列表中选择：TD_SOPC_Board。在 Altera SOPC Builder –PWM_NIOSII 窗口中设计此实验所需的 NiosII 系统。
8. 选择 File→New Component 菜单，新建一个外设。在如图 5-3-1 所示的 VHDL 标签页中点击”Add HDL File”按钮，在对话框中选择 AVALON_PWM.VHD 文件。
9. 在 Signals 标签页中，手动更改 Signal Type 列表的信号，如图 5-3-2 所示。
10. 在 Interfaces 标签页中，如图 5-3-3 所示进行设置，Slave addressing:中选择 Registers。
11. 在 Component Wizard 标签页中，如图 5-3-4 所示进行设置，元件命名为 AVALON_PWM。
12. 点击 Finish 按钮，完成 AVALON_PWM 元件的设置。设置完成后可以在 SOPC Builder 的 System Contents 元件模拟池中可以看到添加了一个 User Logic 选项，其中的元件就是用户自己定制的 AVALON_PWM 元件。

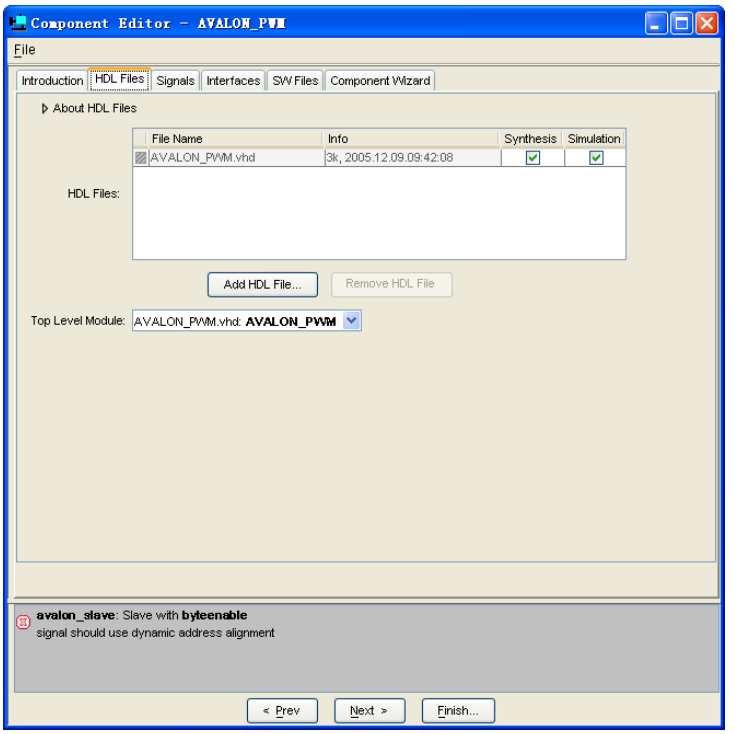


图 5-3-1　添加 VHDL 文件标签页

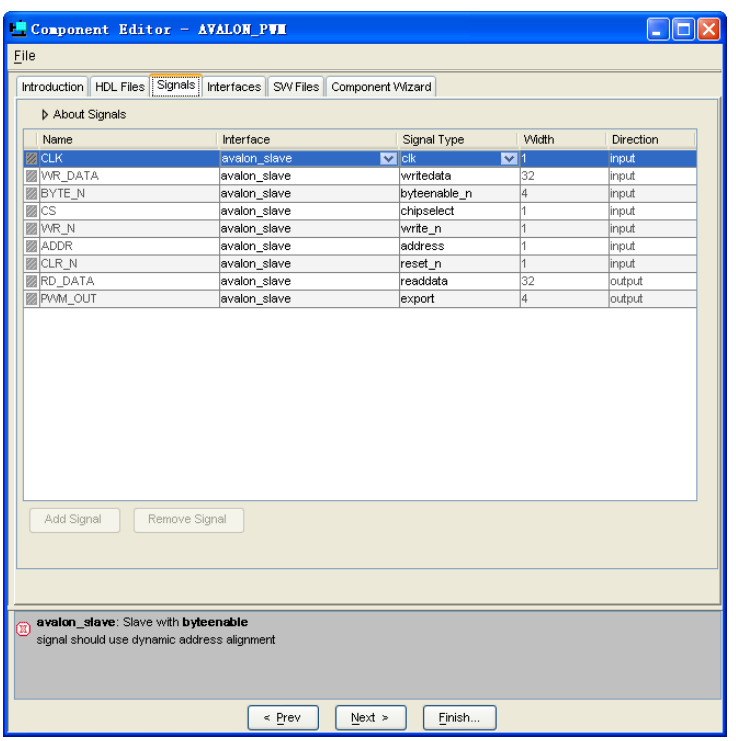


图 5-3-2　Signals 标签页

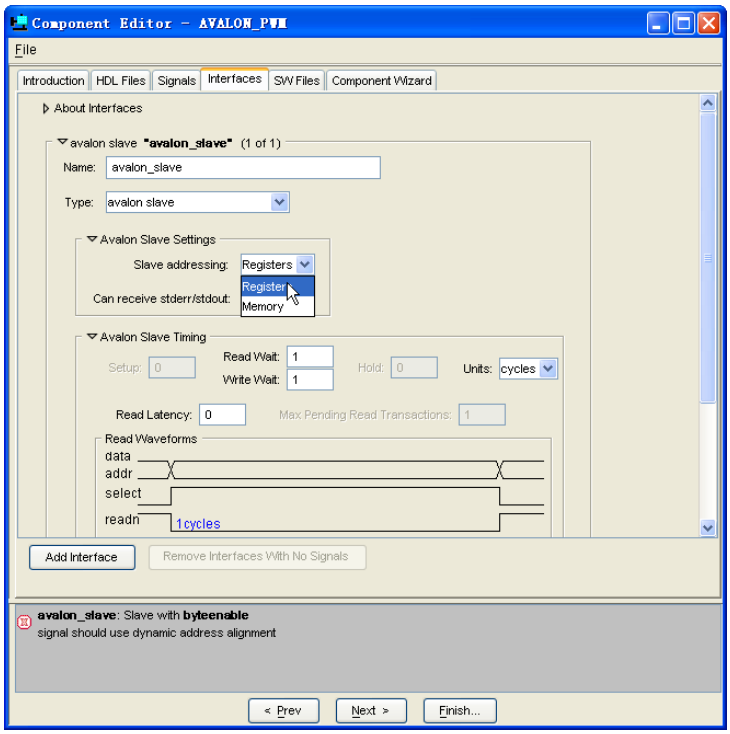


图 5-3-3 Interfaces 标签页

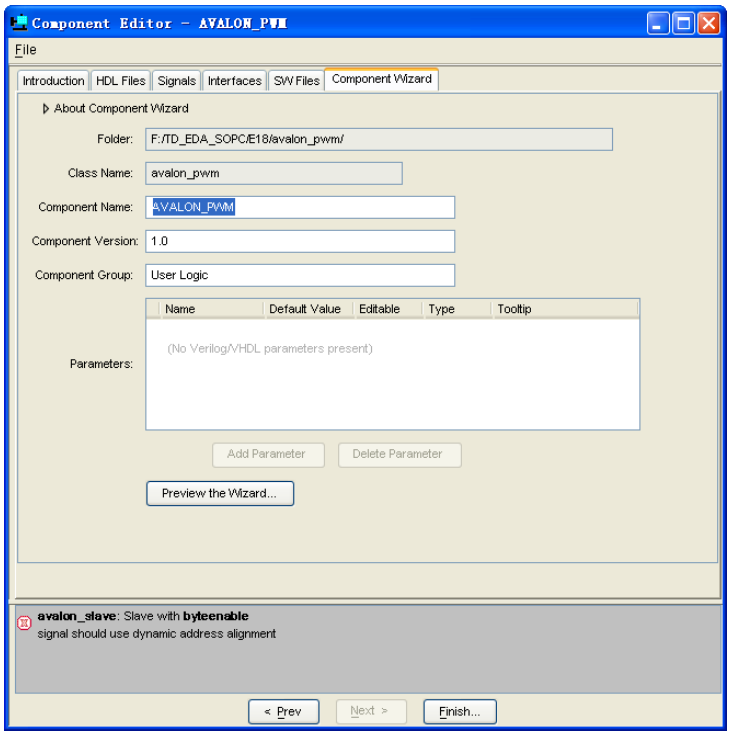


图 5-3-4 Component Wizard 标签页

13. 在 Avalon 模块下分别添加 Nios II Processor、Interval timer、 JTAG UART、IDT71V416 SRAM、System ID Peripheral、EPCS Serial、Avalon Tri-State Bridge、AVALON_PWM，分别重命名为：cpu、system_timer、jtag_uart、ext_ram、sysid、epcs_controller 、ext_ram_bus、my_pwm。
14. 在 SOPC Builder 的模块表中点击 ext_ram 的 BASE，输入 0x00000000 后回车，选择 Module 菜单中的 Lock Base/Address。
15. 选择 System 菜单中的 Auto -Assign Base Addresses，对系统的基地址进行重新分配；选择 System 菜单中的 Auto -Assign IRQs，对系统的中断进行重新分配。
16. 如图 5-3-5 所示，显示了最终的系统配置及其地址映射。

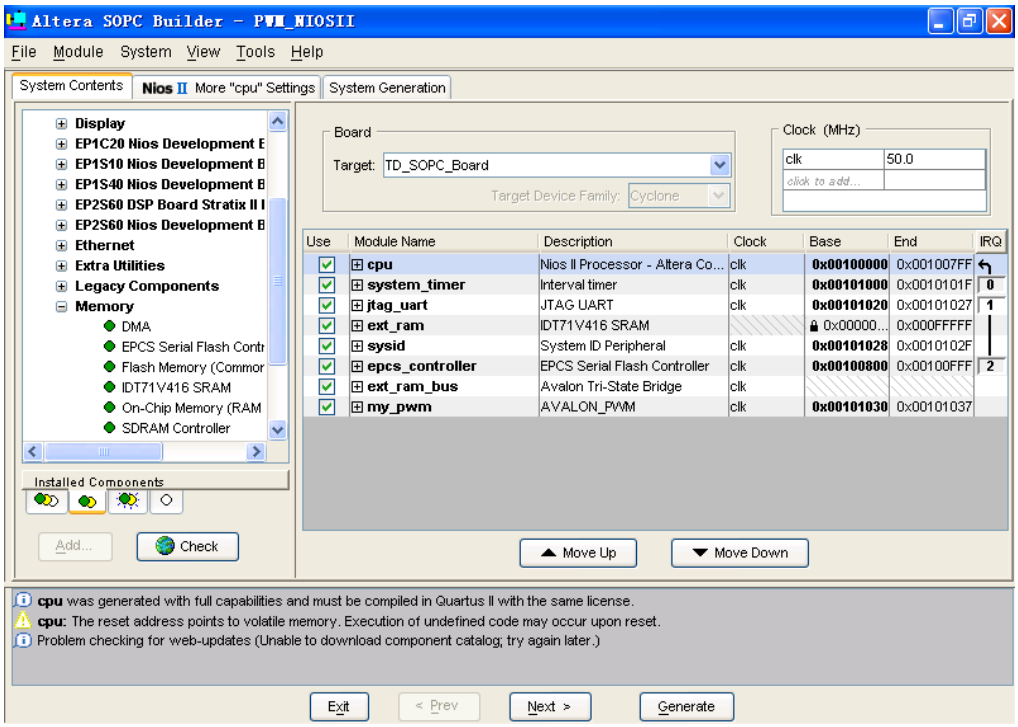


图 5-3-5 最终的 Nios II 系统配置及其地址映射

17. 选择 Nios II More"cpu"Settings 选项，对系统进行进一步设置，指定系统的复位地址和执行地址为 ext_ram。选择 System Generation 选项，在 SOPC Builder 生成页中选中 HDL 选项。点击"Generate"生成硬件系统文件，完成后点击"EXIT"按钮退出 SOPC Builder。
18. 在 BDF 文件窗口，选择 PWM_NIOSII，将其放入图形设计文件窗口中。
19. 将 PWM_NIOSII 模块与输入(input)、输出(output)、双向(bidir)接口连接，将所有无用的引脚置为输入状态，三态。
20. 选择 Processing→Start → Start Analysis&Synthesis 菜单，对系统进行分析、综合。
21. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 5-3-1 所示。

表 5-3-1 添加用户外设实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_153	SHA_D [12]	PIN_84
RST	PIN_131	SHA_D [13]	PIN_85
SHA_A [2]	PIN_128	SHA_D [14]	PIN_86
SHA_A [3]	PIN_127	SHA_D [15]	PIN_87
SHA_A [4]	PIN_126	SHA_D [16]	PIN_66
SHA_A [5]	PIN_125	SHA_D [17]	PIN_67
SHA_A [6]	PIN_124	SHA_D [18]	PIN_68
SHA_A [7]	PIN_98	SHA_D [19]	PIN_73
SHA_A [8]	PIN_95	SHA_D [20]	PIN_74
SHA_A [9]	PIN_94	SHA_D [21]	PIN_75
SHA_A [10]	PIN_93	SHA_D [22]	PIN_76
SHA_A [11]	PIN_88	SHA_D [23]	PIN_77
SHA_A [12]	PIN_106	SHA_D [24]	PIN_58
SHA_A [13]	PIN_107	SHA_D [25]	PIN_59
SHA_A [14]	PIN_108	SHA_D [26]	PIN_60
SHA_A [15]	PIN_143	SHA_D [27]	PIN_61
SHA_A [16]	PIN_113	SHA_D [28]	PIN_62
SHA_A [17]	PIN_132	SHA_D [29]	PIN_63
SHA_A [18]	PIN_133	SHA_D [30]	PIN_64
SHA_A [19]	PIN_134	SHA_D [31]	PIN_65
SHA_D [0]	PIN_122	SRAM_BE [0]	PIN_116
SHA_D [1]	PIN_121	SRAM_BE [1]	PIN_117
SHA_D [2]	PIN_120	SRAM_BE [2]	PIN_114
SHA_D [3]	PIN_119	SRAM_BE [3]	PIN_115
SHA_D [4]	PIN_105	SRAM_WE	PIN_99
SHA_D [5]	PIN_104	SRAM_OE	PIN_118
SHA_D [6]	PIN_101	SRAM_CS	PIN_123
SHA_D [7]	PIN_100	PWM[0]	PIN_2
SHA_D [8]	PIN_78	PWM[1]	PIN_3
SHA_D [9]	PIN_79	PWM[2]	PIN_4
SHA_D [10]	PIN_82	PWM[3]	PIN_5
SHA_D [11]	PIN_83		

22. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编译，生成可以配置到 FPGA 的 SOF 文件。

23. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 5-3-6 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。

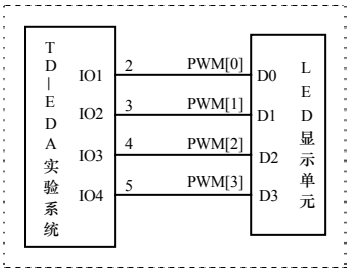


图 5-3-6 添加用户外设实验接线图

24. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置，至此硬件设计工作已经完成。
25. 运行 NiosII IDE 软件开发环境，选择 File→New→C/C++ Application 菜单，建立新工程。在 Nios II IDE 新工程向导的 Name 中填入软件工程的名称：PWM；在 SOPC Builder System 中选择硬件配置文件（PTF 文件）所在的目录；在 Select Project Template 中选择使用的模板 Blank Project，完成后点击”Finish”按钮。
26. 选择 File→New→File 菜单，新建文件 PWM.C，在 Nios II IDE 文本编辑器中编写程序，源程序如下：

```
#include <stdio.h>
#include "altera_avalon_pwm.h"
#include "system.h"

int main()
{ int rx_char;
  char line[100];
  printf("Hello from Nios II!\n");
  printf("Tang Du Instrument Company \n");
  printf("\nPlease enter an LED intensity between 1 to 4 (0 to exit)\n");
  IOWR_ALTERA_AVALON_PWM_DIVIDER(MY_PWM_BASE,0xFF);
  IOWR_ALTERA_AVALON_PWM_DUTY(MY_PWM_BASE,0xFF);
  while (1)
 { fgets(line, sizeof(line),stdin);
   sscanf(line,"%d",&rx_char);
    switch (rx_char)
    {case 4:
        IOWR_ALTERA_AVALON_PWM_DUTY(MY_PWM_BASE,0xFF);
         printf("Level 4 intensity\n");
         break;
     case 3:
```

```
            IOWR_ALTERA_AVALON_PWM_DUTY(MY_PWM_BASE,0x70);
              printf("Level 3 intensity\n");
              break;
            case 2:
              IOWR_ALTERA_AVALON_PWM_DUTY(MY_PWM_BASE,0x40);
              printf("Level 2 intensity\n");
              break;
            case 1:
              IOWR_ALTERA_AVALON_PWM_DUTY(MY_PWM_BASE,0x20);
              printf("Level 1 intensity\n");
              break;
            case 0:
              return 0;
              break;
            default:
              printf("Please enter an integer value from 0 to 4\n");
              break;
        }
    }
     return 0;
}
```

27. 程序编写完成后，选择菜单 Project→Build Project 对工程进行编辑。
28. 选择 Run→Run As→Nios II Hardware 菜单，运行程序，在如图 5-3-7 所示的 Console 窗口中看到程序运行结果。输入 1～4 在开发板上可以观察到 LED 的亮度变化，也可用示波器观察输出的波形。

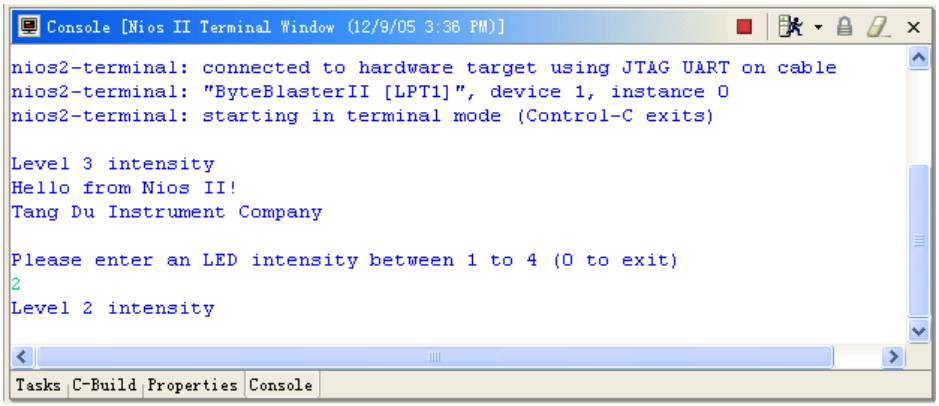


图 5-3-7 Console 窗口运行结果

5.4 串口通讯实验

5.4.1 实验目的:

1. 熟悉基于 FPGA 的嵌入式系统开发的步骤;
2. 熟悉 Nios II 软核处理器的系统结构及串口通讯模块的原理;
3. 熟悉基于 Nios II 处理器的程序设计过程。

5.4.2 实验设备:

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

5.4.3 实验内容:

本实验在 SOPC Builder 中配置 Nios II 系统+外部 SRAM，设计一个基于 Nios II 软核的嵌入式系统，使用 Nios II IDE 调试软件编写程序，借助串口调试工具软件，实现嵌入式系统与 PC 机之间进行串口通讯的功能。

5.4.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 SOPC4。
2. 选择 File→New 菜单，创建图形设计文件 SOPC4.BDF，打开图形编辑器界面。
3. 选择菜单 Tools→SOPC Builder，启动 SOPC Builder 工具。在 System Name 对话框中输入系统名称：UART_NIOSII，并选择 VHDL 硬件描述语言。
4. 在 SOPC Builder 中的 Target 下拉列表中选择:TD_SOPC_Board。在 Altera SOPC Builder –UART_NIOSII 窗口中设计此实验所需的 NiosII 系统。
5. 在 Avalon 模块下分别添加 Nios II Processor、Interval timer、 JTAG UART、EPCS Serial Flash Controller、IDT71V416 SRAM、Avalon Tri-State Bridge，分别重命名为：cpu、system_timer、jtag_uart、epcs_controller 、ext_ram、ext_ram_bus。
6. 在 Avalon 模块中选择 Communication 下的 UART(RS-232 serial port)，点击”Add…”按钮，添加串行通讯模块。
7. 如图 5-4-1 所示的 UART 设置向导， NIOS II UART 模块是通用的串行接口，可以设置波特率、数据位数、校验方式和停止位数，并可选择控制信号。UART 在 Altera 器件内实现简单的 RS-232 异步发送与接收逻辑。通过两个外部引脚(TXD 和 RXD)发送和接收串行数据，用 6 个 16 位寄存器进行软件控制和数据通信。在图 5-4-1 中选择波特率为 115200，其它按照系统默认的选项不需要更改。将 uart_0 重命名为：uart。
8. 在 SOPC Builder 的模块表中点击 ext_ram 的 BASE，输入 0x00000000 后回车，选择 Module 菜单中的 Lock Base/Address。

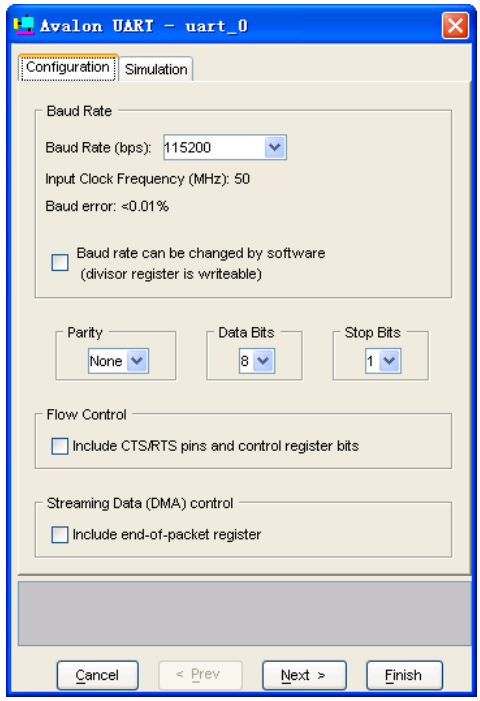


图 5-4-1 UART 设置向导

9. 选择 System 菜单中的 Auto -Assign Base Addresses，对系统的基地址进行重新分配；选择 System 菜单中的 Auto -Assign IRQs，对系统的中断进行重新分配。
10. 如图 5-4-2 所示，显示了最终的系统配置及其地址映射。

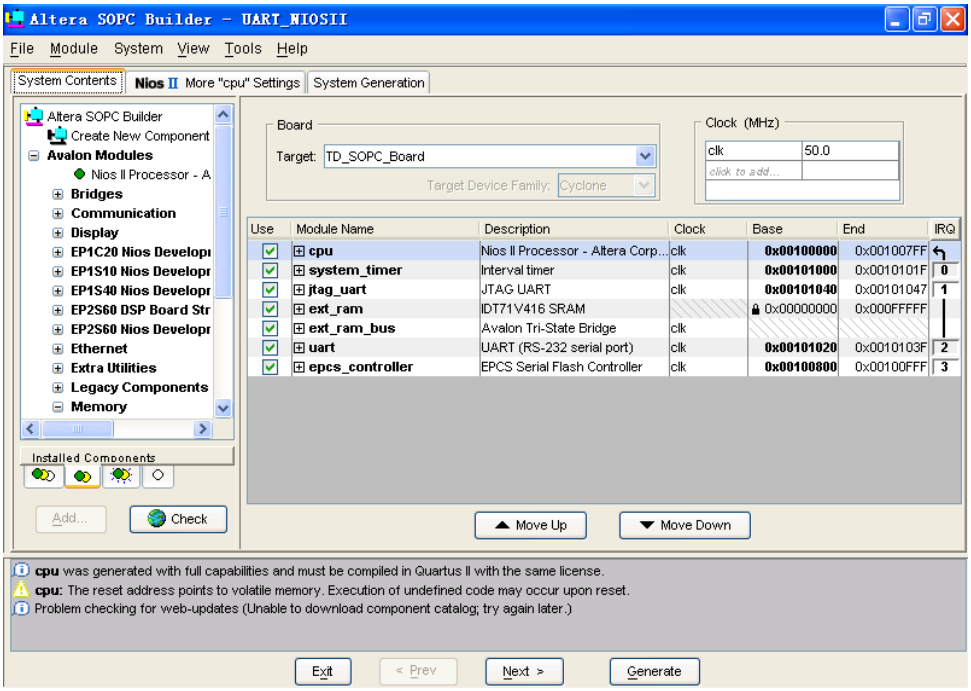


图 5-4-2 最终的 Nios II 系统配置及其地址映射

11. 选择 Nios II More”cpu”Settings 选项，对系统进行进一步的设置，指定系统的复位地址和执行地址为 ext_ram。选择 System Generation 选项，在 SOPC Builder 生成页中选中 HDL 选项。点击”Generate”生成硬件系统文件，完成后点击”EXIT”按钮退出 SOPC Builder。
12. 在 BDF 文件窗口，选择 UART_NIOSII，将其放入图形设计文件窗口中。
13. 将 UART_NIOSII 模块与输入(input)、输出(output)、双向(bidir)接口连接，将所有无用的引脚置为输入状态，三态。
14. 选择 Processing→Start → Start Analysis&Synthesis 菜单，对系统进行分析、综合。
15. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 5-4-1 所示。

表 5-4-1 串口通讯实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_153	SHA_D [6]	PIN_101
RST	PIN_131	SHA_D [7]	PIN_100
SHA_A [2]	PIN_128	SHA_D [8]	PIN_78
SHA_A [3]	PIN_127	SHA_D [9]	PIN_79
SHA_A [4]	PIN_126	SHA_D [10]	PIN_82
SHA_A [5]	PIN_125	SHA_D [11]	PIN_83
SHA_A [6]	PIN_124	SHA_D [12]	PIN_84
SHA_A [7]	PIN_98	SHA_D [13]	PIN_85
SHA_A [8]	PIN_95	SHA_D [14]	PIN_86
SHA_A [9]	PIN_94	SHA_D [15]	PIN_87
SHA_A [10]	PIN_93	SHA_D [16]	PIN_66
SHA_A [11]	PIN_88	SHA_D [17]	PIN_67
SHA_A [12]	PIN_106	SHA_D [18]	PIN_68
SHA_A [13]	PIN_107	SHA_D [19]	PIN_73
SHA_A [14]	PIN_108	SHA_D [20]	PIN_74
SHA_A [15]	PIN_143	SHA_D [21]	PIN_75
SHA_A [16]	PIN_113	SHA_D [22]	PIN_76
SHA_A [17]	PIN_132	SHA_D [23]	PIN_77
SHA_A [18]	PIN_133	SHA_D [24]	PIN_58
SHA_A [19]	PIN_134	SHA_D [25]	PIN_59
SHA_D [0]	PIN_122	SHA_D [26]	PIN_60
SHA_D [1]	PIN_121	SHA_D [27]	PIN_61
SHA_D [2]	PIN_120	SHA_D [28]	PIN_62
SHA_D [3]	PIN_119	SHA_D [29]	PIN_63
SHA_D [4]	PIN_105	SHA_D [30]	PIN_64
SHA_D [5]	PIN_104	SHA_D [31]	PIN_65

续表 5-4-1 串口通讯实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
SRAM_BE [0]	PIN_116	SRAM_WE	PIN_99
SRAM_BE [1]	PIN_117	SRAM_OE	PIN_118
SRAM_BE [2]	PIN_114	RXD	PIN_3
SRAM_BE [3]	PIN_115	TXD	PIN_2
SRAM_CS	PIN_123		

16. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编译，生成可以配置到 FPGA 的 SOF 文件。
17. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 5-4-3 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。将 JP2 短路块的 T 与 T/CP、R 与 R/CP 短接，仔细检查确保接线无误后打开电源。

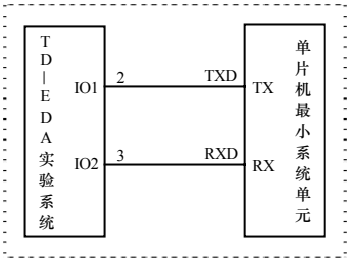


图 5-4-3 串口通讯实验接线图

18. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，对芯片进行配置，至此硬件设计工作已经完成。
19. 运行 NiosII IDE 软件开发环境，选择 File→New→C/C++ Application 菜单，建立新工程。在 Nios II IDE 新工程向导的 Name 中填入软件工程的名称：UART；在 SOPC Builder System 中选择硬件配置文件（PTF 文件）所在的目录；在 Select Project Template 中选择使用的模板 Blank Project，完成后点击”Finish”按钮。
20. 选择 File→New→File 菜单，新建文件 UART.C，在 Nios II IDE 文本编辑器中编写程序，源程序如下：

```
#include <stdio.h>
#include "system.h"
#include <unistd.h>
#include "string.h"
#define   buf_size (33)
int main(void)
{char *msg="This is my first uart file ";
 char buffer[buf_size];
 int i,j;
```

```
  FILE *fp;
   for(j=0;j<10;j++)
   {fp=fopen("/dev/uart","w");
   if (fp)
    {fprintf(fp,"%s\n",msg);
      fclose(fp);}
}
while(1)
   {fp=fopen("/dev/uart","r");
    fread(buffer,buf_size,1,fp);
    fclose(fp);
    if((buffer[7]=='0')&&(buffer[8]=='0')&&(buffer[9]=='0'))
      {printf("data stop\n");
       break;}
    else
    {for(i=0;i<33;i++)
      {printf("%c",buffer[i]); }
     printf("\n");}
}
}
```

21. 程序编写完成后对工程进行编辑，查看编辑结果。
22. 打开串口调试工具软件，将波特率设置为 115200。选择 Run→Run As→Nios II Hardware 菜单，运行程序。程序下载后如图 5-4-4 所示在串口调试工具软件的接收区看到 NIOSII 系统发送的数据。

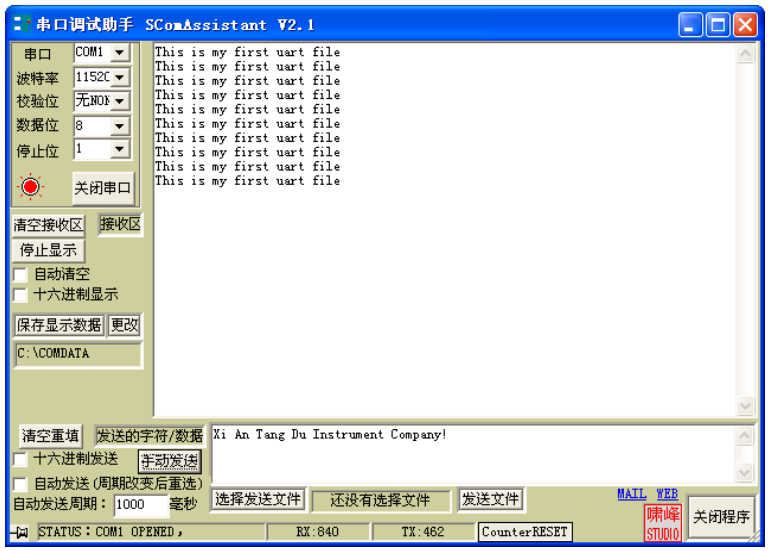


图 5-4-4 串口调试工具软件

23. 在发送区填入要发送的数据 Xi An Tang Du Instrument Company!，点击”手动发送”按钮，在 Nios II IDE 软件的 Console 窗口可以看到 NIOSII 系统接收到的数据，如图 5-4-5 所示。

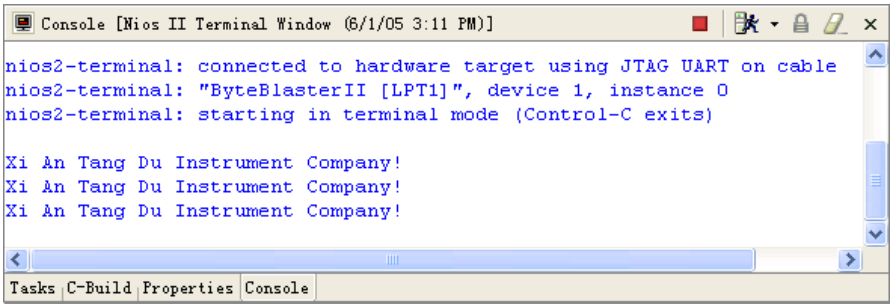


图 5-4-5 NIOS II 系统接收到的数据

24. 实验完成后将 JP2 短路块的 T 与 T/MC、R 与 R/MC 短接。

5.5 外部 FLASH 扩展实验

5.5.1 实验目的:

1. 熟悉基于 FPGA 的嵌入式系统开发的步骤;
2. 熟悉 Nios II 软核处理器的系统结构;
3. 熟悉基于 Nios II 处理器的程序设计及调试方法。

5.5.2 实验设备:

PC 微机一台，TD-EDA 实验箱一台，SOPC 开发板一块。

5.5.3 实验内容:

本实验在 SOPC Builder 中配置 Nios II 系统+外部 SRAM+外部 Flash 存储器，设计一个基于 Nios II 软核的嵌入式系统，使用 Nios II IDE 调试软件编写程序实现一个简易的数字钟功能。

5.5.4 实验步骤

1. 运行 Quartus II 软件，建立新工程，工程名称及顶层文件名称为 SOPC5。
2. 选择 File→New 菜单，创建图形设计文件 SOPC5.BDF，打开图形编辑器界面。
3. 选择菜单 Tools→SOPC Builder，启动 SOPC Builder 工具。在 System Name 对话框中输入系统名称：CLOCK，并选择 VHDL 硬件描述语言。
4. 在 SOPC Builder 中的 Target 下拉列表中选择:TD_SOPC_Board。在 Altera SOPC Builder –CLOCK 窗口中设计此实验所需的 NiosII 系统。
5. 在 Avalon 模块下分别添加 Nios II Processor、Interval timer、 JTAG UART、EPCS Serial Flash Controller、IDT71V416 SRAM、Avalon Tri-State Bridge、PIO(Parallel I/O)、，分别重命名为：cpu、system_timer、jtag_uart、epcs_controller、ext_ram、ext_ram_bus、seven_seg_pio(8bits Output)、seg_pio(8bits Output)。
6. 在 Avalon 模块中选择 Memory 下的 Flash Memory(Common Flash Interface)，点击”Add…”按钮，添加 Flash 存储器，SOPC 开发板使用的 Flash 存储器是 AM29LV160DT-90EC。
7. 在如图 5-5-1 所示的 Flash 存储器设置向导的 Address Width：中选择”20 bits”、Data Width：中选择”16 bits”、Reference Designator（chip label)：中显示 U4。
8. 选择 Flash 存储器设置向导的 Timing 标签页，在如图 5-5-2 所示的 Flash 存储器时序设置向导中，Setup：中填入 45、Wait：中填入 160、Hold：中填入 35，单击”Finish”按钮完成 Flash 的设置，返回 SOPC Builder 窗口。右键单击”cfi_flash_0”，将 cfi_flash_0 重命名为 ext_flash。

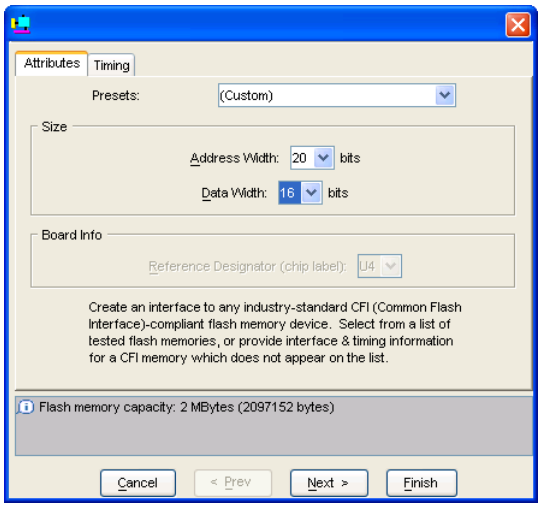


图 5-5-1 Flash 存储器设置向导

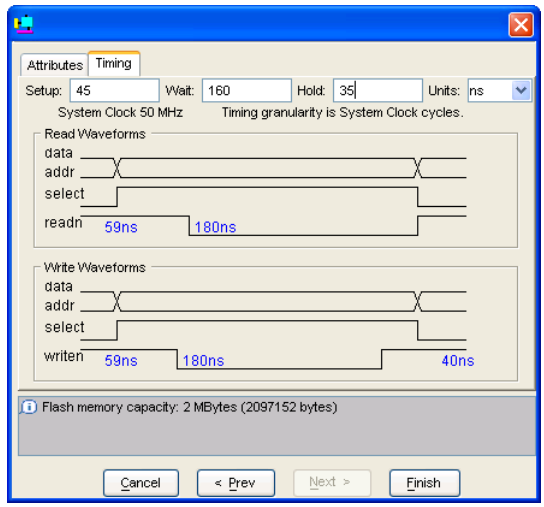


图 5-5-2 Flash 存储器时序设置向导

9. 在 Avalon 模块中选择 Other 下的 PIO(Parallel I/O)，点击”Add…”按钮，在如图 5-5-3 所示的 Avalon PIO-pio_0 输入设置向导中，选择”input ports only”、宽度 Width：2bits。
10. 点击”Input Options”标签页，在如图 5-5-4 所示的 Input Options 标签页向导中，在 Edge Capture Register 下打开 Synchronously capture 选择 Falling Edge。在 Interrupt 下打开 Generate IRQ 选择 Edge。单击”Finish”按钮完成设置。将 pio_0 重命名为 button_pio。

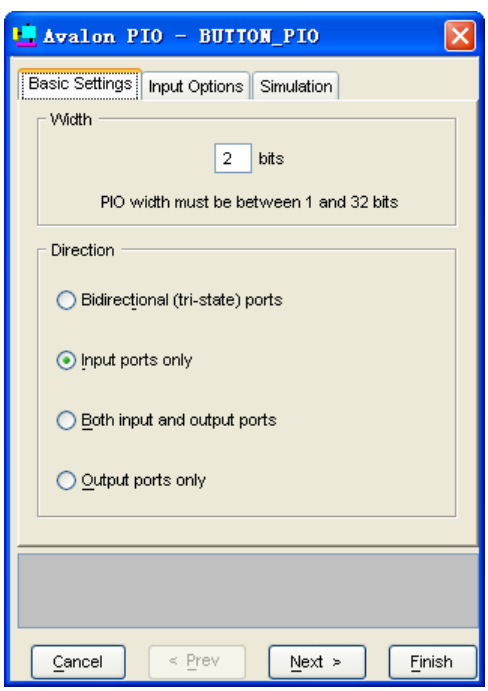


图 5-5-3 输入设置向导

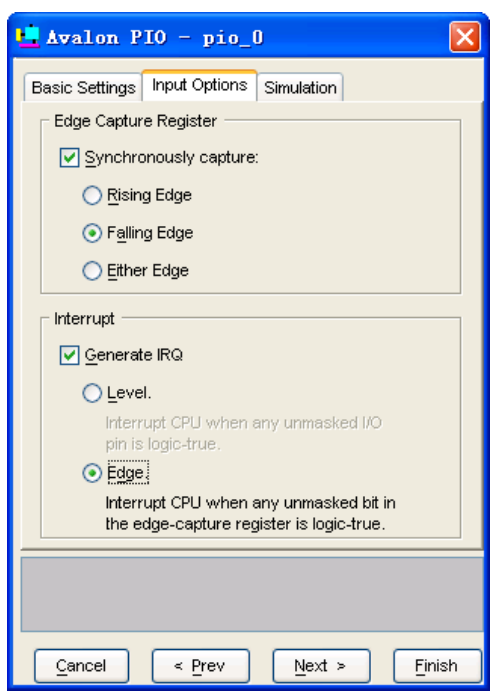


图 5-5-4 Input Options 标签页向导

11. 在 SOPC Builder 的模块表中点击 ext_ram 的 BASE，输入 0x00000000 后回车，选择 Module 菜单中的 Lock Base/Address，在 ext_ram 的基地址旁出现一个锁子图标。
12. 选择 System 菜单中的 Auto -Assign Base Addresses，对系统的基地址进行重新分配；选择 System 菜单中的 Auto -Assign IRQs，对系统的中断进行重新分配。如图 5-5-5 所示，显示了最终的系统配置及其地址映射。

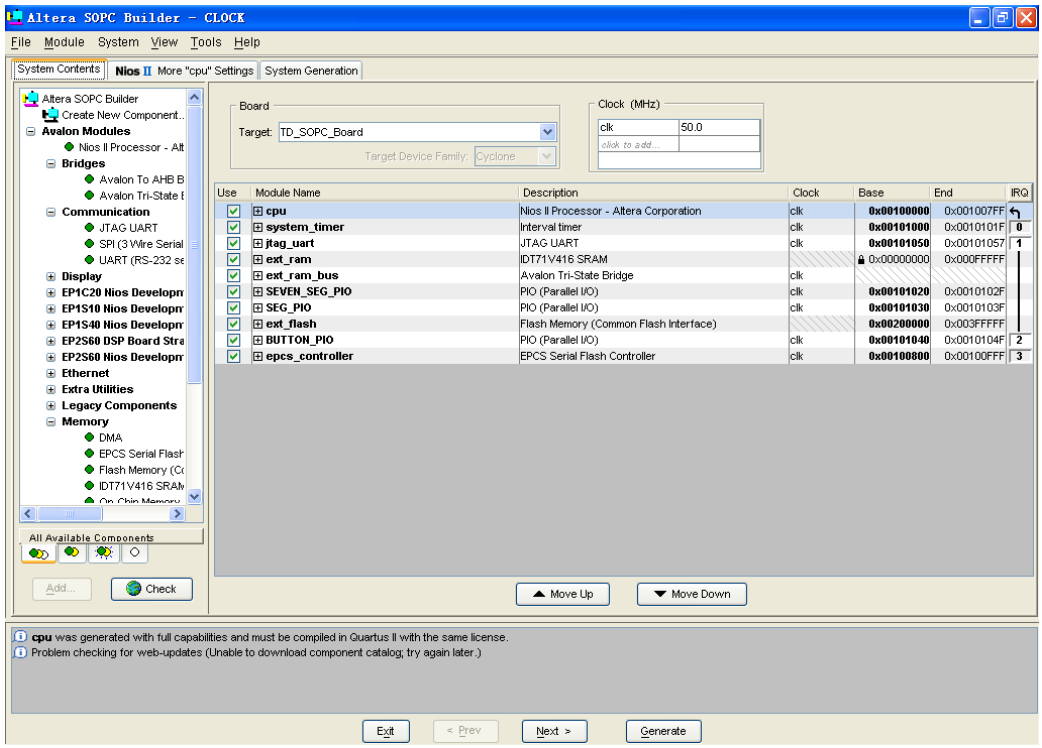


图 5-5-5 最终的 Nios II 系统配置及其地址映射

13. 选择 Nios II More"cpu"Settings 选项，对系统进行进一步的设置，指定系统的复位地址 Reset Address 指定为 ext_flash，执行地址 Exception Address 指定为 ext_ram。选择 System Generation 选项，在 SOPC Builder 生成页中选中 HDL 选项。点击"Generate"生成硬件系统文件，完成后点击"EXIT"按钮退出 SOPC Builder。
14. 在 BDF 文件窗口，选择 CLOCK，将其放入图形设计文件窗口中。将 CLOCK 模块与输入(input)、输出(output)、双向(bidir)接口连接，将所有无用的引脚置为输入状态，三态。
15. 选择 Processing→Start → Start Analysis&Synthesis 菜单，对系统进行分析、综合。
16. 选择 Assignments→Assignment Editor 菜单，对工程进行引脚分配。分配结果如表 5-5-1 所示。

表 5-5-1 外部 FLASH 扩展实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
CLK	PIN_153	SHA_D [20]	PIN_74
RST	PIN_131	SHA_D [21]	PIN_75
SHA_A [1]	PIN_141	SHA_D [22]	PIN_76
SHA_A [2]	PIN_128	SHA_D [23]	PIN_77
SHA_A [3]	PIN_127	SHA_D [24]	PIN_58
SHA_A [4]	PIN_126	SHA_D [25]	PIN_59
SHA_A [5]	PIN_125	SHA_D [26]	PIN_60
SHA_A [6]	PIN_124	SHA_D [27]	PIN_61
SHA_A [7]	PIN_98	SHA_D [28]	PIN_62
SHA_A [8]	PIN_95	SHA_D [29]	PIN_63
SHA_A [9]	PIN_94	SHA_D [30]	PIN_64
SHA_A [10]	PIN_93	SHA_D [31]	PIN_65
SHA_A [11]	PIN_88	SRAM_BE [0]	PIN_116
SHA_A [12]	PIN_106	SRAM_BE [1]	PIN_117
SHA_A [13]	PIN_107	SRAM_BE [2]	PIN_114
SHA_A [14]	PIN_108	SRAM_BE [3]	PIN_115
SHA_A [15]	PIN_143	SRAM_CS	PIN_123
SHA_A [16]	PIN_113	SRAM_WE	PIN_99
SHA_A [17]	PIN_132	SRAM_OE	PIN_118
SHA_A [18]	PIN_133	FLASH_WE	PIN_137
SHA_A [19]	PIN_134	FLASH _CS	PIN_138
SHA_A [20]	PIN_135	FLASH _OE	PIN_139
SHA_D [0]	PIN_122	FLASH _BYTE	PIN_140
SHA_D [1]	PIN_121	SEVEN [0]	PIN_2
SHA_D [2]	PIN_120	SEVEN [1]	PIN_3
SHA_D [3]	PIN_119	SEVEN [2]	PIN_4
SHA_D [4]	PIN_105	SEVEN [3]	PIN_5
SHA_D [5]	PIN_104	SEVEN [4]	PIN_6
SHA_D [6]	PIN_101	SEVEN [5]	PIN_7
SHA_D [7]	PIN_100	SEVEN [6]	PIN_8
SHA_D [8]	PIN_78	SEVEN [7]	PIN_11
SHA_D [9]	PIN_79	SEG [0]	PIN_12
SHA_D [10]	PIN_82	SEG [1]	PIN_13
SHA_D [11]	PIN_83	SEG [2]	PIN_14
SHA_D [12]	PIN_84	SEG [3]	PIN_15

续表 5-5-1 外部 FLASH 扩展实验引脚分配结果

引脚名称	引脚顺序	引脚名称	引脚顺序
SHA_D [13]	PIN_85	SEG [4]	PIN_16
SHA_D [14]	PIN_86	SEG [5]	PIN_17
SHA_D [15]	PIN_87	SEG [6]	PIN_18
SHA_D [16]	PIN_66	SEG [7]	PIN_19
SHA_D [17]	PIN_67	BUTTON [0]	PIN_183
SHA_D [18]	PIN_68	BUTTON [1]	PIN_194
SHA_D [19]	PIN_73		

17. 选择 Tools→Compiler Tool 菜单，点击”Start”按钮对此工程进行编译，生成可以配置到 FPGA 的 SOF 文件。
18. 使用 TD-EDA 实验系统及 SOPC 开发板，如图 5-5-6 所示进行实验接线，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 AS 下载接口，仔细检查确保接线无误后打开电源。

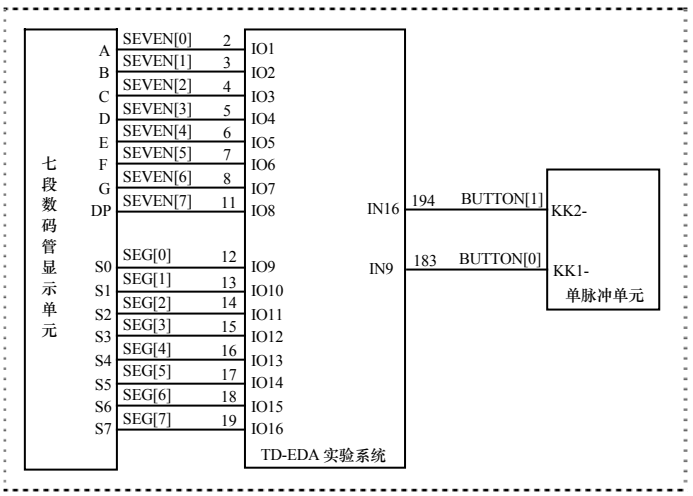


图 5-5-6 外部 FLASH 扩展实验接线图

19. 在 Quartus II 软件中，选择 Tools→Programmer 菜单，选择 Active Serial Programming 下载模式，将系统生成的 SOPC5.POF 文件下载到开发板上的串行配置存储器 EPCS1 中，对芯片进行配置，至此硬件设计工作已经完成。
20. 运行 NiosII IDE 软件开发环境，选择 File→New→C/C++ Application 菜单，建立新工程。在 Nios II IDE 新工程向导的 Name 中填入软件工程的名称：CLOCK；在 SOPC Builder System 中选择硬件配置文件（PTF 文件）所在的目录；在 Select Project Template 中选择 Blank Project 模板，完成后点击”Finish”按钮。
21. 选择 File→New→File 菜单，新建文件 CLOCK.C，在 Nios II IDE 文本编辑器中编写程序，源程序如下：

```
#include "alt_types.h"
#include <stdio.h>
#include <unistd.h>
#include "system.h"
#include "sys/alt_irq.h"
#include "altera_avalon_pio_regs.h"
static alt_u8 hour;
static alt_u8 minute;
static alt_u8 second;
static alt_u8 day;
static alt_u8 max_day;
static alt_u8 month;
static alt_u16 year;
static int flag;
static int begin;
static alt_u8 count;
static alt_u8 seg;
volatile int edge_capture;

static void sevenseg_set_hex(int hex)
{   static alt_u8 segments[16] = {
      0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, /* 0-9*/
      0x77, 0x7A, 0x39, 0x6E, 0x79, 0x71 };                       /* a-f */
   unsigned int data = segments[hex%10] |(segments[hex/10]<<8);
 IOWR_ALTERA_AVALON_PIO_DATA(SEVEN_SEG_PIO_BASE, data);}
static void count_sevenseg()
{#ifdef SEVEN_SEG_PIO_BASE
       sevenseg_set_hex(count);
    #endif}
static void seven_seg()
{#ifdef SEG_PIO_BASE
   IOWR_ALTERA_AVALON_PIO_DATA(SEG_PIO_BASE, seg);
#endif}
 static void handle_button_interrupts(void *context,alt_u32 id)
      {volatile int*edge_capture_ptr=(volatile int *)context;
       *edge_capture_ptr=IORD_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE);
       IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE,0);}
      static void init_button_pio()
        {void*edge_capture_ptr=(void*)&edge_capture;
```

```
        IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_PIO_BASE,0xf);
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_PIO_BASE,0x0);
        alt_irq_register(BUTTON_PIO_IRQ,edge_capture_ptr,handle_button_interrupts);}
void seg_select(int gao,int di,int cs1,int cs2)
    { seg = cs1;
      seven_seg();
      count = di;
      count_sevenseg();
      usleep(2000);
      seg=cs2;
      seven_seg();
      count = gao;
      count_sevenseg();
      usleep(2000);}
      static void last_day()
     {if(month==4||month==6||month==9||month==11)
                max_day=30;
      else if(month==2)
            {if((year%4==0&&year%100!=0)||year%400==0)
                max_day=29;
             else
                max_day=28;}
          else
          max_day=31;    }
    static void initial_time()
   {hour=17;
    minute=42;
    second=20;
    year=2005;
    month=12;
    day=5;
    flag=0;
    begin=0;}
   static void display_time(int second,int minute,int hour)
   {    int j,ge,shi,cs1,cs2;
        for(j=0;j<85;j++)
            {ge=second%10;
             shi=second/10;
             cs1=0xbf;
```

```
                cs2=0x7f;
                seg_select(ge,shi,cs1,cs2);
                ge=minute%10;
                shi=minute/10;
                cs1= 0xef;
                cs2= 0xdf;
                seg_select(ge,shi,cs1,cs2);
                ge=hour%10;
                shi=hour/10;
                cs1= 0xfb;
                cs2= 0xf7;
                seg_select(ge,shi,cs1,cs2);}
}
static void display_day(int day,int month,int year)
{       int jj,gej,shij,csj1,csj2,ba;
        for(jj=0;jj<85;jj++)
                { gej=day%10;
                  shij=day/10;
                  csj1= 0xbf;
                  csj2= 0x7f;
                  seg_select(gej,shij,csj1,csj2);
                  gej=month%10;
                  shij=month/10;
                  csj1= 0xEF;
                  csj2= 0xDF;
                  seg_select(gej,shij,csj1,csj2);
                  gej=year%1000;
                  ba=gej%100;
                  gej=gej/100;
                  shij=year/1000;
                  csj1=0xFE;
                  csj2=0xFD;
                  seg_select(gej,shij,csj1,csj2);
                  gej=ba%10;
                  shij=ba/10;
                  csj1=0xFB;
                  csj2=0xF7;
                  seg_select(gej,shij,csj1,csj2); }
}
```

```
int main(void)
{   #ifdef BUTTON_PIO_BASE
    init_button_pio();
 #endif
 initial_time();
 while(1)
      {   if(begin==0)
            { second++;
             if(second>=60)
               {second=0;
                minute++; }
             if(minute>=60)
                {minute=0;
                 hour++; }
             if(hour>=24)
                {hour=0;
                 day++;}
               last_day();
             if(day>max_day)
                {day=1;
                 month++;
                 last_day();}
               if(month>12)
                {month=1;
                 year++;}
            }
          if(edge_capture==1)
            {display_day(day,month,year);
            edge_capture==0;}
          else
            {display_time(second,minute,hour);
             edge_capture==1;}
            }
      }
```

22. 程序编写完成后对工程进行编辑，查看编辑结果。
23. 选择 Run→Run As→Nios II Hardware 菜单，运行程序，可以在开发板上观察到实验现象。选择菜单 Run→Debug As→Nios II Hardware 可以对程序进行调试。
24. 调试完成后可以将程序下载到开发板上的 Flash 存储器中，将 ByteBlaster II 下载电缆插入 SOPC 开发板的 JTAG 下载接口。

25. 选择 Tools→Flash Programmer 菜单，在如图 5-5-7 所示的 Flash 编程器窗口中选择 New 按钮，新建一个编程文件，在 Name 栏中输入 CLOCK_PRO，注意 Target Hardware 栏中指定 CLOCK.ptf 文件所在的目录。

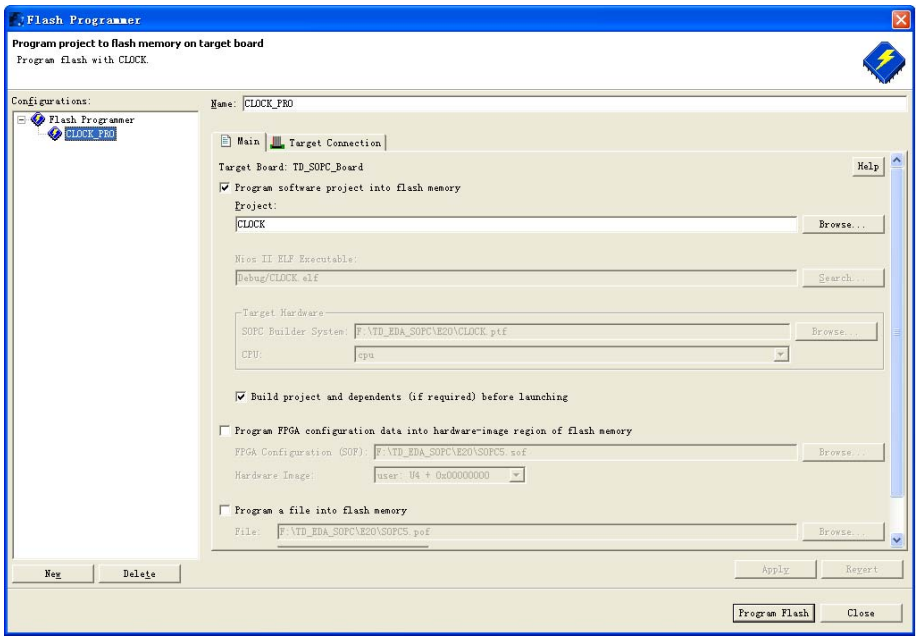


图 5-5-7 Flash 编程器窗口

26. 点击 Apply 按钮，点击 Program Flash 按钮，对 Flash 存储器进行编程。
27. Flash 存储器下载的过程可以在 Console 窗口观察到，如图 5-5-8 所示，编程完成后，重新打开电源可以观察到实验现象。

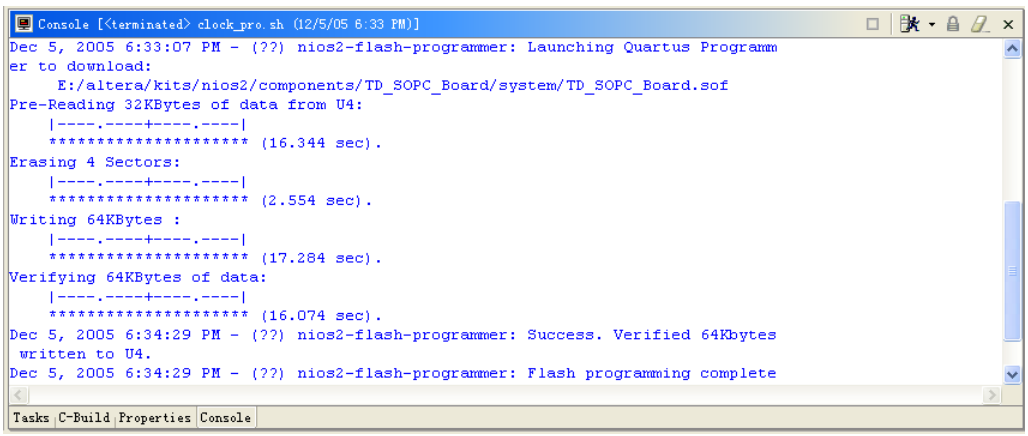


图 5-5-8 Console 窗口下载过程

附录 1 Cyclone 系列器件简介

Cyclone 现场可编程门阵列系列器件是基于 1.5V、0.13um、全铜层 SRAM 工艺，其密度增加至 20060 个逻辑元件(LE)，RAM 增加至 288Kb。它具有用于时钟的锁相环及 DDR SDR 和快速周期 RAM(FCRAM)存储器所需的专用双数据速率(DDR)接口等。支持多种 I/O 标准，包括 640Mbps 的 LVDS，以及速率为 33MHz 和 66MHz、数据宽度为 32 位和 64 位的 PCI。

Cyclone 系列器件具有如下特性(参见附表 1)

- 2910~20060 个逻辑单元(LE)。
- 多达 294 912 位 RAM(36 864B)。
- 支持低成本串行配置器件(EPCS1 或 EPCS4)配置。
- 支持 LVTTL、LVCOMS、SSTL-2 和 SSTL-3 I/O 标准。
- 支持 66MHz，32 位 PCI 标准和高速(331Mb/s)LVDS。
- 两个 PLL 提供时钟倍频和相移。
- 多达八个全局时钟线，每个逻辑阵列块(LAB)可利用六个时钟资源。
- 支持 DDR SDRAM(133MHz)、FCRAM 和单数据速率(SDR)SDRAM。
- 支持知识产权(IP)核，包括 Altera MegaCore 函数和 AMPP mega 函数。

附表 1 Cyclone 器件特性

特性	EP1C3	EP1C6	EP1C12	EP1C20
逻辑单元(LE)	2910	5980	12060	20060
M4K RAM 块(128×36 位)	13	20	52	64
总 RAM 位	59 904	92 160	239 616	294 912
锁相环(PLL)	1	2	2	2
最大用户 I/O 引脚	104	185	249	301

Cyclone 系列器件采用方型扁平封装(QFP-Quad Flat Pack)和节省空间的 FBGA (Fineline Ball Grid Array) 封装，如附表 2 所示。

附表 2 Cyclone 系列器件封装和 I/O 引脚数

器件	100 引脚 TQFP	144 引脚 TQFP	240 引脚 PQFP	324 引脚 FBGA	400 引脚 FBGA
EP1C3	65	104			
EP1C6		98	185		
EP1C12			173	249	
EP1C20				233	301

注：TQFP:细型四边有引线扁平封装 PQFP:塑料四边有引线扁平封装

Cyclone 系列器件结构的特点如下所述：

1. 新型可编程架构

Cyclone 系列器件是基于一种全新的低成本架构，从设计之初就充分考虑了节省成本，因此可以为价格敏感的应用提供全新的可编程的解决方案。低成本 FPGA 的设计过程要面临许多的挑战，其中最具挑战性的就是如何在性能、特性及价格之间找到一个合适的定位。FPGA 设计师必须找到一个平衡点，以确保在可编程片上系统(SOPC)方案中既可以提供充足的逻辑单元和存储器容量，又不会使价格提高。

Cyclone 器件设计时选择了较小的封装形式，以提供给用户足够的 I/O 引脚和良好的功耗特性。在此基础上，根据封装的物理尺寸定义裸片连接点的最大尺寸，装入尽可能多的逻辑结构和存储器块，从而保证每种封装都装入最多的逻辑资源。

2. 嵌入式存储资源

Cyclone 器件为在 FPGA 上实现低成本的数字信号处理(DSP)系统提供了一个理想的平台。它为设计工程师提供了灵活的硬件解决方案，能够实现设计中所需的多个乘法器。

Cyclone 器件中的 M4K 块可用来实现软乘法器，以满足图像处理、音频处理和消费类电子系统的需要。软乘法器可以根据所需数据位宽、系数位宽来定制，并且根据需要选择精度。利用 M4K 块，可采用并行乘法方式或分布式运算方式来实现不同数据宽度的软乘法器。这两种不同的实现方法提供了等待时间、存储器利用率和乘法器尺寸上的灵活性。在 Cyclone 器件的 M4K 块中可以实现的乘法器的数量如附表 3 所示。

附表 3 在 M4K 块中实现 18×18 位乘法器

器件	M4K 块的数量	用 M4K 块可实现的乘法器的数量
EP1C3	13	5
EP1C6	20	7
EP1C12	52	20
EP1C20	64	25

3. 专用外部存储器接口电路

DDR SDRAM 拥有与 SDR 相同的结构，但是在时钟的上下沿都传输数据，从而使数据交换的带宽加倍。FCRAM 则是一种延迟时间比较低、基于 SRAM 功能架构的存储器件。在大容量、低功耗的应用环境下，FCRAM 提供了更好的性能。和 SDRAM 类似，FCRAM 支持在时钟的上下两个沿进行数据交换，适用于流水线存储和预置数据操作，与 SDRAM 架构的存储器相比，所需的访问时钟周期大大减少。

Cyclone 器件通过片内内嵌的专用接口电路实现与双数据速率(DDR)SDRAM 和 FCRAM 以及单数据速率(SDR)SDRAM 器件进行快速可靠的数据交换，最高速率可达到 266Mbps。如果再结合针对 Cyclone 器件优化的即取即用的 IP(Intellectual Property)控制器核，工程师可以在几分钟之内将一个 SDRAM 和 FCRAM 的功能合并到一个系统中。

4. 支持的接口及协议

Cyclone 器件支持多种串行总线和网络接口，还支持广泛的通信协议，如以太网协议。这些接口和协议被广泛应用于消费品、工业和通信产品中。Altera 也为这方面的应用提供了一系列的

专门针对 Cyclone FPGA 结构优化的 IP 核。

(1) PCI

PCI 是一个标准的总线型接口，通常用于集成组件、外设插板，还用于处理器和存储系统之间的内部连接。Cyclone 器件兼容 3.3V PCI 局部总线规范 2.2 版本，支持高达 66MHz 的 32 位 PCI 总线。Cyclone 器件中的 I/O 单元经过专门设计，可以匹配严格的 PCI 标准所要求的建立和保持时间。为了提供最大的灵活性，每个输入信号都可以通过两个独立的延时路径输入到不同的芯片区域。

(2) SDRAM 及 FCRAM 接口

Cyclone 器件可以通过内建的专用接口与单数据速率和双数据速率 SDRAM 连接。

(3) 10/100 及千兆以太网

以太网是局域网(LAN)中使用最广泛的访问方式，其定义的标准是 IEEE 802.3 标准。用 Cyclone 器件实现的以太网媒体存取控制器与物理层器件的接口速率可以达到 10Mbps、100Mbps 或 1Gbps 的最大带宽。如果结合针对 Cyclone 器件优化的 IP 核，用户可以很容易地在 Cyclone 芯片中实现以太网的 MAC 功能。

(4) 串行总线接口

Cyclone 器件支持一系列的串行总线接口，如串行外设接口(SPI)、IIC、IEEE 1394 标准和通用串行总线(USB)。通过在 Cyclone 器件中实现 SPI 和 IIC 标准，可以在集成电路、处理器和外设之间提供一个低速的通信链路。IEEE1394 和 USB 也可以在处理器、计算机和其它器件之间建立一条链接。Cyclone 器件可以用来实现与 PHY 器件的总线控制和接口功能。

(5) 通信协议

Cyclone 器件支持一系列的通信协议，包括 E1、E3、T1、T3 和 SONET/SDH 等。E1 和 E3 是欧洲数字传输标准；T1 和 T3 是相应的北美数字传输标准；SONET/SDH 是光纤上的数字传输标准。这些通信接口协议一般用于中低端通信设备中，Cyclone 器件可以满足这些应用在性能上、逻辑密度上和系统特性上的需求。

5. 锁相环的实现

Cyclone 器件内置最多 2 个增强型锁相环，可给用户提供高性能的时钟管理能力，如频率合成、可编程移相、片外时钟输出、可编程占空比、失锁检测以及高速差分时钟信号的输入和输出等。Cyclone 的锁相环电路具有时钟合成功能，内部实际运行的时钟可以不同于输入的时钟频率。每个锁相环可以提供 3 个不同频率的输出。锁相环提供两个比例因子分别为 m 和 n 的除法计数器，其中的 m、n 和后比例计数器(g0、g1 和 e)可以设置成从 1 到 32 之间的任意整数。

Cyclone 的锁相环还可以实现对一个应用进行时分复用的功能，这样对于某些特定的电路就可以在一个时钟周期内运行多次。通过时分复用，可以用较少的逻辑资源来实现所需要的功能，因此可以利用这种共享资源的方法来增加芯片内的可用资源。Cyclone 中的每个锁相环还可以有一个差分的或单端的片外时钟输出。每个锁相环有一对片外时钟输出管脚，该输出管脚可以支持附表 4 所示的多种 I/O 标准。外部时钟输出可以用作系统时钟或用来同步整个板上的不同器件，其时钟反馈特性可以用来补偿内部的延时或使输出的时钟与输入时钟相位对齐。

附表 4 中 m、n 除法计数器和后比例计数器的范围从 1 到 32；最小的项移为 VCO 周期除以 8。如果以度为单位增加，Cyclone 器件的输出至少可以以 45 度递增，更小的增加度数有可能受到频率和分频系数的限制。Cyclone 的锁相环具有可编程移相的能力。用户可以在一个时间单元

内对时钟进行移相，最高分辨率达到 150ps（皮秒）。可编程移相特性一般用于匹配那些关键时序路径上时钟沿的约束，如建立时间和保持时间的约束。

附表 4　Cyclone 锁相环特性

特性	锁相环支持
时钟倍频及分频	m、n 除法计数器和后比例计数器
相移	分辨率最高到 150ps 递增
可编程占空比	3
内部时钟输出数目	2
片外时钟输出数目	最多 1 对差分或 1 个单端信号
输入及输出时钟可支持的 I/O 标准	LVTTL 、 LVCMOS 、 2.5/1.8/1.5V 、 3.3V PCI 、 SSTL-2Class I&II、SSTL-3Class I&II、LVDS

Cyclone PLL 的相位锁定信号用来指示输出时钟相对于参考时钟相位已经完全稳定地锁定。它一般用于系统控制和同步整个板子上的其它不同器件。Cyclone 的锁相环具有可编程占空比的能力。可编程占空比使得锁相环可以产生不同占空比的输出时钟。

6. I/O 特性

Cyclone 器件可以支持差分的 I/O 标准，如 LVDS 和去抖动差分信号(RSDS)，当然也支持单端的 I/O 标准，如 LVTTL、LVCMOS、SSTL 和 PCI。

Cyclone 器件可以支持最大 129 个通道的 LVDS 和 RSDS。Cyclone 器件内的 LVDS 缓冲器可以支持最高达 640Mbps 的数据传输速度。与单端的 I/O 标准相比，这些内置于 Cyclone 器件内部的 LVDS 缓冲器保持了信号的完整性，并具有更低的电磁干扰(EMI)和更低的电源功耗。

Cyclone 器件提供常用的单端 I/O 标准的支持，如 LVTTL、LVCMOS、SSTL-2、SSTL-3 和 PCI，用于与板上其他器件的接口。单端 I/O 可以提供比差分 I/O 标准更强的电流驱动能力，主要应用在与高性能存储器的接口中，如双数据速率(DDR)的 SDRAM 和 FCRAM 器件。

7. 支持 Nios II 系列嵌入式处理器

Cyclone 器件可以实现 Nios II 嵌入式处理器，而且只占用不到 600 个逻辑单元(LE)，因此在含多达 20060 个 LE 的最大 Cyclone 器件中，可以将多个 Nios II 处理器集成到一个 Cyclone 器件中。Nios II 处理器和外围设备占有约 600LE。开发人员通过往 Nios II 处理器指令集中增加定制指令，可以加速软件算法。定制指令可以在一个时钟周期的时间内完成复杂的处理任务，为系统优化提供了一种高性价比的解决方案。用户添加的定制指令可以访问存储器和 Nios II 系统外部的逻辑，提供了高效、灵活的访问数据和逻辑资源的能力。定制指令允许设计者灵活、轻便地设计高端软件，同时保留了并行硬件操作在可编程逻辑器件(PLD)中的性能优势。

8. 配置方案

串行配置器件系列包括 EPCS1 和 EPCS4 两个产品，分别提供 1Mb 和 4Mb 的存储容量。该配置器件在保证低成本的同时还具备在系统编程(ISP)能力和多次编程能力，且具有包括 ISP 和 Flash 存储器访问接口等特性，8 引脚小外形封装，增加了在低价格、小面积应用领域的使用机会。串行配置器件最高达 64Mb 的存储容量使得它为 Stratis II 系列器件提供了一种价格敏感、小型化的配置方案。

SOPC 开发板上使用的是 Cyclone 系列的 EP1C6Q240C8 芯片。该芯片的引脚定义如附表 5 所示。该芯片的引脚功能如附表 6 所示。

附表 5　EP1C6Q240 引脚功能

引脚	组号	VREF 组	引脚名称/功能	可选功能	配置功能	DQS144
1	B1	VREF0B1	IO	LVDS14p	INIT_DONE	DM1L
2	B1	VREF0B1	IO	LVDS14n		DQ1L0
3	B1	VREF0B1	IO	LVDS13p	CLKUSR	DQ1L1
4	B1	VREF0B1	IO	LVDS13n		
5	B1	VREF0B1	IO	VREF0B1		
6	B1	VREF0B1	IO			DQ1L2
7	B1	VREF0B1	IO	LVDS12p		DQ1L3
8	B1	VREF0B1	IO	LVDS12n		
9	B1	VREF0B1	VCCIO1			
10	B1	VREF0B1	GND			
11	B1	VREF0B1	IO	DPCLK1		
12	B1	VREF0B1	IO	LVDS11p		
13	B1	VREF0B1	IO	LVDS11n		
14	B1	VREF0B1	IO	LVDS10p		
15	B1	VREF0B1	IO	LVDS10n		
16	B1	VREF0B1	IO	LVDS9p		
17	B1	VREF0B1	IO	LVDS9n		
18	B1	VREF0B1	IO	LVDS8p		
19	B1	VREF0B1	IO	LVDS8n		
20	B1	VREF0B1	IO	LVDS7p		
21	B1	VREF0B1	IO	LVDS7n		
22	B1	VREF0B1	VCCIO1			
23	B1	VREF1B1	IO	VREF1B1		
24	B1	VREF1B1	IO		nCSO	
25	B1	VREF1B1	DATAO		DATA0	
26	B1	VREF1B1	nCONFIG		nCONFIG	
27		VREF1B1	VCCA_PLL1			
28	B1	VREF1B1	CLK0	LVDSCLK1p		
29	B1	VREF1B1	CLK1	LVDSCLK1n		
30		VREF1B1	GNDA_PLL1			
31		VREF1B1	GNDG_PLL1			
32	B1	VREF1B1	nCEO		nCEO	
33	B1	VREF1B1	nCE		nCE	

续附表 5 EP1C6Q240 引脚功能

引脚	组号	VREF 组	引脚名称/功能	可选功能	配置功能	DQS144
34	B1	VREF1B1	MSEL0		MSEL0	
35	B1	VREF1B1	MSEL1		MSEL1	
36	B1	VREF1B1	DCLK		DCLK	
37	B1	VREF1B1	IO		ASDO	
38	B1	VREF1B1	IO	PLL1_OUTp		
39	B1	VREF1B1	IO	PLL1_OUTn		
40	B1	VREF2B1	GND			
41	B1	VREF2B1	IO			
42	B1	VREF2B1	IO	LVDS6p		
43	B1	VREF2B1	IO	LVDS6n		
44	B1	VREF2B1	IO	LVDS5p		
45	B1	VREF2B1	IO	LVDS5n		
46	B1	VREF2B1	IO	LVDS4p		
47	B1	VREF2B1	IO	LVDS4n		
48	B1	VREF2B1	IO	LVDS3p		
49	B1	VREF2B1	IO	LVDS3n		
50	B1	VREF2B1	IO	DPCLK0		DQS1L
51	B1	VREF2B1	VCCIO1			
52	B1	VREF2B1	GND			
53	B1	VREF2B1	IO	LVDS2p		
54	B1	VREF2B1	IO	LVDS2n		
55	B1	VREF2B1	IO	VREF2B1		
56	B1	VREF2B1	IO			DQ1L4
57	B1	VREF2B1	IO	LVDS1p		DQ1L5
58	B1	VREF2B1	IO	LVDS1n		DQ1L6
59	B1	VREF2B1	IO	LVDS0p		DQ1L7
60	B1	VREF2B1	IO	LVDS0n		
61	B4	VREF2B4	IO	LVDS71p		
62	B4	VREF2B4	IO	LVDS71n		
63	B4	VREF2B4	IO	LVDS70p		
64	B4	VREF2B4	IO	LVDS70n		
65	B4	VREF2B4	IO	LVDS69p		DQ1B7
66	B4	VREF2B4	IO	LVDS69n		DQ1B6
67	B4	VREF2B4	IO	LVDS68p		DQ1B5
68	B4	VREF2B4	IO	LVDS68n		DQ1B4
69	B4	VREF2B4	GND			

续附表 5 EP1C6Q240 引脚功能

引脚	组号	VREF 组	引脚名称/功能	可选功能	配置功能	DQS144
70	B4	VREF2B4	VCCIO4			
71		VREF2B4	GND			
72		VREF2B4	VCCINT			
73	B4	VREF2B4	IO	DPCLK7		DQS1B
74	B4	VREF2B4	IO	VREF2B4		
75	B4	VREF2B4	IO	LVDS67p		
76	B4	VREF2B4	IO	LVDS67n		
77	B4	VREF2B4	IO	LVDS66p		
78	B4	VREF2B4	IO	LVDS66n		
79	B4	VREF2B4	IO			
80	B4	VREF2B4	IO	LVDS65p		
81	B4	VREF2B4	IO	LVDS65n		
82	B4	VREF2B4	IO	LVDS64p		
83	B4	VREF2B4	IO	LVDS64n		
84	B4	VREF2B4	IO	LVDS63p		
85	B4	VREF2B4	IO	LVDS63n		
86	B4	VREF1B4	IO	LVDS62p		
87	B4	VREF1B4	IO	LVDS62n		
88	B4	VREF1B4	IO			
89	B4	VREF1B4	GND			
90	B4	VREF1B4	VCCINT			
91	B4	VREF1B4	GND			
92	B4	VREF1B4	VCCIO4			
93	B4	VREF1B4	IO	VREF1B4		
94	B4	VREF1B4	IO	LVDS61p		DM1B
95	B4	VREF1B4	IO	LVDS61n		
96	B4	VREF1B4	IO	LVDS60p		
97	B4	VREF1B4	IO	LVDS60n		
98	B4	VREF1B4	IO	LVDS59p		
99	B4	VREF1B4	IO	LVDS59n		
100	B4	VREF1B4	IO	LVDS58p		
101	B4	VREF1B4	IO	LVDS58n		
102	B4	VREF0B4	IO	LVDS57p		
103	B4	VREF0B4	IO	LVDS57n		
104	B4	VREF0B4	IO	LVDS56p		
105	B4	VREF0B4	IO	LVDS56n		

续附表 5 EP1C6Q240 引脚功能

引脚	组号	VREF 组	引脚名称/功能	可选功能	配置功能	DQS144
106	B4	VREF0B4	IO			
107	B4	VREF0B4	IO	VREF0B4		
108	B4	VREF0B4	IO	DPCLK6		
109		VREF0B4	GND			
110		VREF0B4	VCCINT			
111	B4	VREF0B4	GND			
112	B4	VREF0B4	VCCIO4			
113	B4	VREF0B4	IO	LVDS55p		DQ1B3
114	B4	VREF0B4	IO	LVDS55n		DQ1B2
115	B4	VREF0B4	IO	LVDS54p		DQ1B1
116	B4	VREF0B4	IO	LVDS54n		DQ1B0
117	B4	VREF0B4	IO	LVDS53p		
118	B4	VREF0B4	IO	LVDS53n		
119	B4	VREF0B4	IO	LVDS52p		
120	B4	VREF0B4	IO	LVDS52n		
121	B3	VREF2B3	IO	LVDS51n		
122	B3	VREF2B3	IO	LVDS51p		
123	B3	VREF2B3	IO	LVDS50n		
124	B3	VREF2B3	IO	LVDS50p		
125	B3	VREF2B3	IO	LVDS49n		DQ1R7
126	B3	VREF2B3	IO	LVDS49p		DQ1R6
127	B3	VREF2B3	IO	VREF2B3		
128	B3	VREF2B3	IO			
129	B3	VREF2B3	GND			
130	B3	VREF2B3	VCCIO3			
131	B3	VREF2B3	IO	DPCLK5		DQS1R
132	B3	VREF2B3	IO	LVDS48n		DQ1R5
133	B3	VREF2B3	IO	LVDS48p		DQ1R4
134	B3	VREF2B3	IO	LVDS47n		DM1R
135	B3	VREF2B3	IO	LVDS47p		
136	B3	VREF2B3	IO	LVDS46n		
137	B3	VREF2B3	IO	LVDS46p		
138	B3	VREF2B3	IO	LVDS45n		
139	B3	VREF2B3	IO	LVDS45p		
140	B3	VREF2B3	IO	LVDS44n		
141	B3	VREF2B3	IO	LVDS44p		

续附表 5　EP1C6Q240 引脚功能

引脚	组号	VREF 组	引脚名称/功能	可选功能	配置功能	DQS144
142	B3	VREF2B3	GND			
143	B3	VREF1B3	IO	PLL2_OUTn		
144	B3	VREF1B3	IO	PLL2_OUTp		
145	B3	VREF1B3	CONF_DONE		CONF_DONE	
146	B3	VREF1B3	nSTATUS		nSTATUS	
147	B3	VREF1B3	TCK		TCK	
148	B3	VREF1B3	TMS		TMS	
149	B3	VREF1B3	TDO		TDO	
150		VREF1B3	GNDG_PLL2			
151		VREF1B3	GNDA_PLL2			
152	B3	VREF1B3	CLK3	LVDSCLK2n		
153	B3	VREF1B3	CLK2	LVDSCLK2p		
154		VREF1B3	VCCA_PLL2			
155	B3	VREF1B3	TDI		TDI	
156	B3	VREF1B3	IO	VREF1B3		
157	B3	VREF0B3	VCCIO3			
158	B3	VREF0B3	IO	LVDS43n		
159	B3	VREF0B3	IO	LVDS43p		
160	B3	VREF0B3	IO	LVDS42n		
161	B3	VREF0B3	IO	LVDS42p		
162	B3	VREF0B3	IO	LVDS41n		
163	B3	VREF0B3	IO	LVDS41p		
164	B3	VREF0B3	IO	LVDS40n		
165	B3	VREF0B3	IO	LVDS40p		
166	B3	VREF0B3	IO	LVDS39n		
167	B3	VREF0B3	IO	LVDS39p		DQ1R3
168	B3	VREF0B3	IO	LVDS38n		DQ1R2
169	B3	VREF0B3	IO	LVDS38p		DQ1R1
170	B3	VREF0B3	IO	DPCLK4		
171	B3	VREF0B3	GND			
172	B3	VREF0B3	VCCIO3			
173	B3	VREF0B3	IO	LVDS37n		
174	B3	VREF0B3	IO	LVDS37p		
175	B3	VREF0B3	IO			DQ1R0
176	B3	VREF0B3	IO	VREF0B3		
177	B3	VREF0B3	IO	LVDS36n		

续附表 5　EP1C6Q240 引脚功能

引脚	组号	VREF 组	引脚名称/功能	可选功能	配置功能	DQS144
178	B3	VREF0B3	IO	LVDS36p		
179	B3	VREF0B3	IO	LVDS35n		
180	B3	VREF0B3	IO	LVDS35p		
181	B2	VREF0B2	IO	LVDS34n		
182	B2	VREF0B2	IO	LVDS34p		
183	B2	VREF0B2	IO	LVDS33n		
184	B2	VREF0B2	IO	LVDS33p		
185	B2	VREF0B2	IO	LVDS32n		DQ0T0
186	B2	VREF0B2	IO	LVDS32p		DQ0T1
187	B2	VREF0B2	IO	LVDS31n		DQ0T2
188	B2	VREF0B2	IO	LVDS31p		DQ0T3
189	B2	VREF0B2	VCCIO2			
190	B2	VREF0B2	GND			
191		VREF0B2	VCCINT			
192		VREF0B2	GND			
193	B2	VREF0B2	IO	DPCLK3		DQS0T
194	B2	VREF0B2	IO	VREF0B2		
195	B2	VREF0B2	IO			
196	B2	VREF0B2	IO	LVDS30n		
197	B2	VREF0B2	IO	LVDS30p		
198	B2	VREF0B2	VCCINT	LVDS29n		
199	B2	VREF0B2	GND	LVDS29p		
200	B2	VREF1B2	IO	LVDS28n		
201	B2	VREF1B2	IO	LVDS28p		
202	B2	VREF1B2	IO	LVDS27n		
203	B2	VREF1B2	IO	LVDS27p		
204	B2	VREF1B2	VCCINT	LVDS26n		
205	B2	VREF1B2	GND	LVDS26p		
206	B2	VREF1B2	IO	LVDS25n		DM0T
207	B2	VREF1B2	IO	LVDS25p		
208	B2	VREF1B2	IO	VREF1B2		
209	B2	VREF1B2	VCCIO2			
210	B2	VREF1B2	GND			
211	B2	VREF1B2	VCCINT			
212	B2	VREF1B2	GND			
213	B2	VREF1B2	IO			

续附表 5　EP1C6Q240 引脚功能

引脚	组号	VREF 组	引脚名称/功能	可选功能	配置功能	DQS144
214	B2	VREF1B2	IO	LVDS24n		
215	B2	VREF1B2	IO	LVDS24p		
216	B2	VREF2B2	IO	LVDS23n		
217	B2	VREF2B2	IO	LVDS23p		
218	B2	VREF2B2	IO	LVDS22n		
219	B2	VREF2B2	IO	LVDS22p		
220	B2	VREF2B2	VCCINT	LVDS21n		
221	B2	VREF2B2	GND	LVDS21p		
222	B2	VREF2B2	IO			
223	B2	VREF2B2	IO	LVDS20n		
224	B2	VREF2B2	IO	LVDS20p		
225	B2	VREF2B2	IO	LVDS19n		
226	B2	VREF2B2	IO	LVDS19p		
227	B2	VREF2B2	IO	VREF2B2		
228	B2	VREF2B2	IO	DPCLK2		
229		VREF2B2	VCCINT			
230		VREF2B2	GND			
231	B2	VREF2B2	VCCIO2			
232	B2	VREF2B2	GND			
233	B2	VREF2B2	IO	LVDS18n		DQ0T4
234	B2	VREF2B2	IO	LVDS18p		DQ0T5
235	B2	VREF2B2	IO	LVDS17n		DQ0T6
236	B2	VREF2B2	IO	LVDS17p		DQ0T7
237	B2	VREF2B2	IO	LVDS16n		
238	B2	VREF2B2	IO	LVDS16p		
239	B2	VREF2B2	IO	LVDS15n	DEV_OE	
240	B2	VREF2B2	IO	LVDS15p	DEV_CLRn	

附表 6　EP1C6Q240 引脚定义

引脚名称	引脚类型	引脚说明
电源和参考引脚		
VCCIO[1..4]	电源	组 1-4 I/O 电源，每个组可以支持不同的电压给输出缓冲器提供电源用于所有 I/O 标准，也给输入缓冲器提供电源用于 LVTTL、LVCMOS、1.5V、1.8V、2.5V 和 3.3V PCI I/O 标准
VCCINT	电源	内部逻辑阵列电源，也给输入缓冲器提供电源用于 LVDS、SSTL2 和 SSTL3 I/O 标准
VREF[0..2]B[1..4]	I/O，输入	组 1-4 参考电压，不使用参考电压时可用作用户 I/O 引脚
VCCA_PLL[1..2]	电源	PLL[1..2]模拟电源，即使不使用 PLL，也必须接 1.5V
GNDA_PLL[1..2]	地	PLL[1..2]模拟地，可以接电路板的地平面
GNDG_PLL[1..2]	地	PLL[1..2]保护地，可以接电路板的地平面
配置和 JTAG 引脚		
CONF_DONE	双向(漏极开路)	专用配置状态引脚，不可用作用户 I/O 引脚
nSTATUS	双向(漏极开路)	专用配置状态引脚，不可用作用户 I/O 引脚
nCONFIG	输入	专用配置控制输入，下降沿复位目标器件，上升沿开始配置。当 nCONFIG 是低电平时所有 I/O 引脚处于三态
DCLK	输入(PS 方式)，输出(AS 方式)	专用配置时钟引脚，在 PS 方式中，DCLK 是时钟输入，用于同步从外部输入到 Cyclone 器件的配置数据；在 AS 方式中，DCLK 是时钟输出，作为控制器的 Cyclone 器件输出时钟
DATA0	输入	专用配置数据输入
nCE	输入	低有效芯片选通，用于检测器件链中有效器件的专用芯片选通输入，低电平时选通器件，高电平时不选通器件
nCEO	输出	低有效芯片选通输出，当器件配置完成时输出低电平，在多器件配置时，接后续器件的 nCE 引脚
ASDO	I/O，输出	Cyclone 器件的主动串行数据输出，用于 AS 方式，Cyclone 器件控制配置，在 ASDO 上输出地址和控制信息，在 PS 方式中可用作用户 I/O 引脚
nCSO	I/O，输出	允许/禁止串行配置器件片选输出，用于 AS 方式，Cyclone 器件控制配置，通过 nCSO 输出低电平允许串行配置器件，在 PS 方式中可用作用户 I/O 引脚
INIT_DONE	I/O，输出(漏极开路)	两用引脚，不作为 INIT_DONE 时可用作用户 I/O 引脚，作为 INIT_DONE 时指示器件已经进入用户方式，器件配置后可以用作用户 I/O 引脚
CLKUSR	I/O，输入	可选的用户时钟输入，用于同步一个或多个器件的初始化，配置后可以用作用户 I/O 引脚
DEV_CLRn	I/O，输入	两用引脚，可以替换所用器件寄存器的清零输入，低电平时所有寄存器清零，高电平时所有寄存器按设计中的定义工作

续附表 6　EP1C6Q240 引脚定义

引脚名称	引脚类型	引脚说明
DEV_OE	I/O，输入	两用引脚，可以替换器件上的所有三态输入，低电平时所有 I/O 引脚处于三态，高电平时 I/O 引脚按设计中的定义工作。
MSEL[1..0]	输入	专用方式选择控制引脚，用于设置器件的配置方式
TMS	输入	专用 JTAG 输入引脚
TDI	输入	专用 JTAG 输入引脚
TCK	输入	专用 JTAG 输入引脚
TDO	输出	专用 JTAG 输出引脚
时钟和 PLL 引脚		
CLK0	输入，LVDS 输入	专用全局时钟输入，可选功能是 LVDSCLK1p，用作 PLL1 的差动输入
CLK1	输入，LVDS 输入	专用全局时钟输入，可选功能是 LVDSCLK1n，用作 PLL1 的差动输入
CLK2	输入，LVDS 输入	专用全局时钟输入，可选功能是 LVDSCLK2p，用作 PLL2 的差动输入
CLK3	输入，LVDS 输入	专用全局时钟输入，可选功能是 LVDSCLK2n，用作 PLL2 的差动输入
PLL1_OUTp	I/O，输出	PLL1 外部时钟输出，可被用作差动或单端 I/O 标准，如果不使用 PLL1 的时钟输出，可用作用户 I/O 引脚
PLL1_OUTn	I/O，输出	PLL1 外部时钟输出的反相端，如果时钟输出是单端的，可用作用户 I/O 引脚
PLL2_OUTp	I/O，输出	PLL2 外部时钟输出，可被用作差动或单端 I/O 标准，如果不使用 PLL2 的时钟输出，可用作用户 I/O 引脚
PLL2_OUTn	I/O，输出	PLL2 外部时钟输出的反相端，如果时钟输出是单端的，可用作用户 I/O 引脚
DPCLK[7..0]	I/O	可以连接到全局时钟网络的两用时钟引脚，能被用作高扇出控制信号，例如时钟、清零、IRDY、TRDY 或 DQS 信号
两用 LVDS 和外部存储器接口引脚		
LVDS[0..71]p	I/O，LVDS RX 或 TX	两用 LVDS I/O 通道 0～71，用于接收或传送 LVDS 兼容信号，带“p”后缀的引脚传送差动通道的同相信号。如果不使用 LVDS 接口，可用作用户 I/O 引脚
LVDS[0..71]n	I/O，LVDS RX 或 TX	两用 LVDS I/O 通道 0～71，用于接收或传送 LVDS 兼容信号，带“n”后缀的引脚传送差动通道的反相信号。如果不使用 LVDS 接口，可用作用户 I/O 引脚
LVDSCLK1p	输入，LVDS 输入	PLL1 两用 LVDS 时钟输入，如果不需要 PLL1 的差动输入，可用作 CLK0 输入引脚

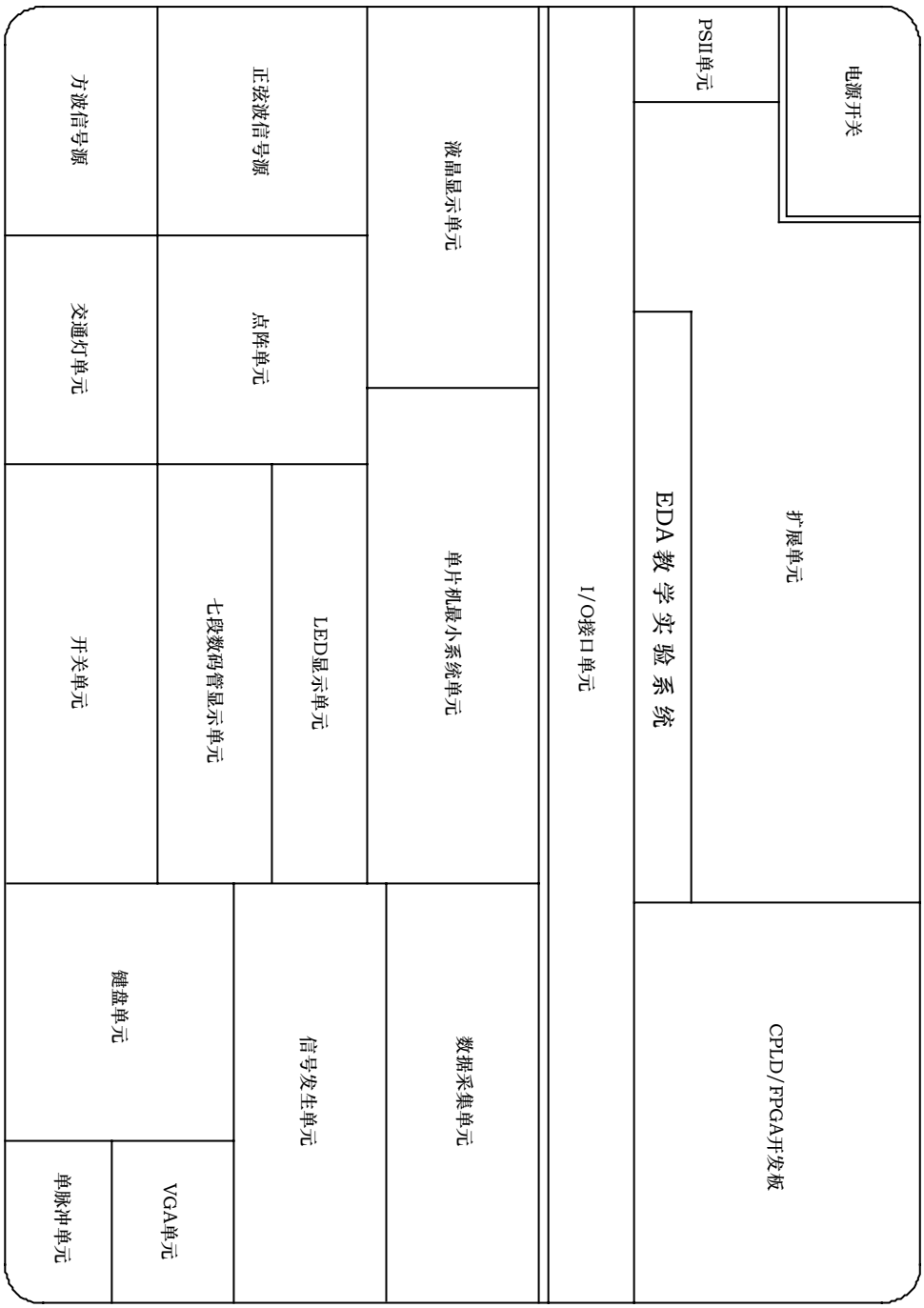
续附表 6　EP1C6Q240 引脚定义

引脚名称	引脚类型	引脚说明
LVDSCLK1n	输入，LVDS 输入	PLL1 两用 LVDS 时钟输入，如果不需要 PLL1 的差动输入，可用作 CLK1 输入引脚
LVDSCLK2p	输入，LVDS 输入	PLL2 两用 LVDS 时钟输入，如果不需要 PLL2 的差动输入，可用作 CLK2 输入引脚
LVDSCLK2n	输入，LVDS 输入	PLL2 两用 LVDS 时钟输入，如果不需要 PLL2 的差动输入，可用作 CLK3 输入引脚
DQS[0..1][L,R,T,B]	I/O	用于外部存储器接口的可选数据选通信号，也用作 DPCLK 引脚，因此 DQS 信号可以连接到全局时钟网络，可编程延时链用来将 DQS 信号移相 90 度或 72 度
DQ[0..1][L,R,T,B]	I/O	用于外部存储器接口的可选数据信号
DM[0..1][L,R,T,B]	I/O	用于外部存储器接口的可选数据屏蔽输出信号

附录 2 系统实验程序清单

数字系统设计基础实验		
文件夹名称	**工程名称**	**实验项目**
E1	GATE.QPF	基本门电路实验
E2	CODER. QPF	编码器实验
E3	DECODER. QPF	译码器实验
E4 ADDER ALU	 ADDER.QPF ALU.QPF	 全加器实验 算术逻辑运算实验
E5	4BITMULT.QPF	乘法器实验
E6 SHIFT8R SHIFT8	 SHIFT8R.QPF SHIFT8.QPF	 并行输入串行输出移位寄存器实验 串行输入并行输出移位寄存器实验
E7	COUNT10.QPF	计数器实验
E8	ODD_DEV_F. QPF	分频器实验
E9 RAM FIFO	 RAM. QPF FIFO. QPF	 RAM 数据存储器实验 FIFO 存储器实验
E10	CDKZHQ. QPF	多路彩灯控制器实验
数字系统设计综合实验		
E11	JIANPAN. QPF	键盘扫描输入实验
E12	SCANLED. QPF	扫描数码显示器实验
E13	DIANZHEN.QPF	点阵显示实验
E14	JTDKZHQ. QPF	交通灯控制实验
E15	CLOCK. QPF	数字钟实验
E16	LCD. QPF	液晶显示实验
E17	PSII. QPF	PSII 接口实验
E18	VGA. QPF	VGA 显示实验
E19	SIGNALR. QPF	DDS 信号发生器实验
E20	HPREFRE. QPF	数字频率计实验
SOPC 嵌入式系统实验		
E21	SOPC1. QPF	HELLO LED 实验
E22	SOPC2. QPF	外部 SRAM 扩展实验
E23	SOPC3. QPF	添加用户外设实验
E24	SOPC4. QPF	串口通讯实验
E25	SOPC5. QPF	外部 FLASH 扩展实验

附录 3　TD-EDA 实验系统布局图



附图 1　TD-EDA 教学实验系统布局图