计算机组成与结构专题实验

实验报告

第四次 程序计数器 PC 与地址寄存器 AR

一、 实验目的

理解地址单元的工作原理。

掌握程序计数器（PC）的两种工作模式，即加 1 计数以及重新设置计数器初始值的方法。

学会地址寄存器（AR）从程序计数器（PC）获取数据和从内部总线（BUS）获取数据的操作方式。

二、 实验原理

1．采用总线多路开关联接方式

地址单元由程序计数器（PC）、地址寄存器（AR）和总线多路开关（BUSMUX）三部分组成。程序计数器（PC）用于指示下一条指令在主存储器中的存放位置，CPU 根据 PC 中的内容来存取指令。由于程序中的指令通常是顺序执行的，因此 PC 具有自动递增的功能。在 T4 时钟脉冲的作用下，PC 的值会自动加 1，以指向下一条指令的地址。此外，在 LDPC 信号的作用下，PC 可以被预置初值，例如在子程序调用或中断响应等场景中。当 LDPC 为高电平时，PC 会从 data[ ] 端接收并装载新的数据；而当 aclr（清零端）为高电平时，PC 会被清零，只有在 aclr 为低电平时，PC 才能正常计数。

地址寄存器（AR，通常采用 DFF_8 或 273 结构）用于锁存访问内存 SRAM 的地址。AR 的地址来源有两个：一是程序计数器（PC）的输出，通常是下一条指令的地址；二是来自内部数据总线的数据，通常是被访问操作数的地址。为了在两路输入数据之间进行切换，FPGA 内部通过总线多路开关（BUSMUX）来实现选择。LDAR 信号与多路选择器的 sel 端相连，当 LDAR 为低电平时，选择 PC 的输出作为 AR 的输入；而当 LDAR 为高电平时，选择内部数据总线的数据作为 AR 的输入。

2．采用 PC、 AR 通过三态门 lpm_bustri 与 BUS 联接

地址单元由程序计数器（PC）、地址寄存器（AR）以及三态门（lpm_bustri）组成，其中程序计数器（PC）和地址寄存器（AR）协同工作，共同生成用于对存储器 RAM 进行读写操作的地址。

程序计数器（PC）的作用是指示下一条指令在主存储器中的具体位置，CPU 依据 PC 中存储的地址信息来获取指令。在时钟脉冲（PC_CLK）的驱动下，PC 具备自动加 1 的功能，能够顺序地指向程序中的下一条指令地址。此外，当接收到 LOAD_PC 信号时，PC 可以被预置一个初始值，这一特性在子程序调用或中断响应等场景中发挥重要作用。当 LOAD_PC 为高电平时，PC 会从 data[7..0]端接收数据并将其加载为新的初始值。RST 是 PC 的清零端，当 RST 为高电平时，PC 会被清零；而当 RST 为低电平时，PC 才能正常进行计数操作。

地址寄存器（AR）采用锁存器（lpm_latch）结构，其主要功能

是锁存用于访问内存 SRAM 的地址信息。

三、 实验任务

实验任务 1

1. 实验准备

按照电路图编辑并输入电路，确保实验台选择 工作模式 0。

对输入的原理图进行编译、引脚锁定，并将程序下载到实验台。

进行硬件实验验证，并与仿真波形图进行比较。

2. 模式设置与输入

使用模式键选择模式 “0”，然后按下系统的复位键。

使用键 2（锁定 PIO12~PIO15）和键 1（PIO8~PIO11）输入 8 位总线数据 B[7..0]，该数据将显示在发光管 D1~D8 和数码管 2/1（PIO16~PIO23） 上。

使用键 5（锁定 PIO4）按下 两次（操作顺序为 0 1 0），产生一个正脉冲，用于高电平清零。

使用键 6（锁定 PIO5）控制 LDAR：

当 LDAR = 0 时，BUSMUX 输出程序计数器 PC 的值。

当 LDAR = 1 时，BUSMUX 输出 D[7..0] 总线数据。

使用键 7（锁定 PIO6）控制 LDPC：

当 LDPC = 1 时，将 D[7..0] 总线数据预置到程序计数器 PC。

当 LDPC = 0 时，程序计数器 PC 处于自动计数状态，对 T4 进行计数。

使用键 8（锁定 PIO7）产生 T4 时钟脉冲，按下 两次 产生一个计数脉冲。

3. 实验操作流程

初始化：

将所有键置为 0。

使用键 2 和键 1 输入数据 34。

按键 5，将程序计数器 PC 清零（操作顺序为 0 1 0）。

观察自动计数功能：

连续按动键 8，观察数码管 8/7 上的输出，即 PC 的值会从 0 开始自动累加 1。

观察数据加载功能：

按键 6,使输出高电平 ‘1’,选通总线上的数据 34 作为 PC 的值。

按键 8，产生一个脉冲上升沿，观察数码管 8/7 上的输出是否显示为 34。

再次按下键 6，使其回到 ‘0’，然后按动键 8，观察 PC 是否重新从初始值开始计数。

预加载数据到程序计数器：

将键 7 置为 1，选通 PC 计数器输出。

使用键 2 和键 1 输入数据 12。

按键 8，产生一个上升脉冲（操作顺序为 0 1 0），将数据 12 预置到 PC 计数器中。

将键 7 置为 0，使 PC 恢复自动计数状态。

观察预加载后的计数功能：

连续按动键 8，观察地址寄存器 AR 的输出是否从 12 开始累加。

实验任务 2

1.实验准备

编辑电路：根据程序计数器的原理图，使用 LPM 库中的元件进行设计：

lpm_latch：用于实现地址锁存器（AR）。

lpm_counter：用于实现程序计数器（PC）。

lpm_bustri：用于实现总线三态输出缓冲器。

选择工作模式：选择 NO.0 工作模式。

2. 输入与配置

输入 8 位地址数据：

使用 键 1 和 键 2 输入 8 位地址数据。

输入的数据将通过发光二极管 D1～D8 显示。

功能键分配：

键 3：RST（复位信号，高电平有效）。

键 4：PC_CLK（程序计数器的时钟信号）。

键 5：LOAD_PC（程序计数器的预置控制信号）。

键 6：INPUT_B（总线数据输入控制信号）。

键 7：PC_B（程序计数器输出控制信号）。

键 8：AR_CLK（地址锁存器的时钟信号）。

显示设置：

数码管 1/2：显示地址锁存器的输出数据 AR[7..0]。

数码管 3/4：显示程序计数器的输出数据 PC[7..0]。

3. 编译与下载

编译原理图：完成电路设计后，对输入的原理图进行编译。

引脚锁定：根据实验系统的要求，锁定引脚配置。

下载程序：将编译后的程序下载到实验台。

4.硬件验证

硬件测试：完成程序下载后，进行硬件实验验证。

观察结果：

观察数码管 1/2 的显示，确认地址锁存器（AR）的输出是否正确。

观察数码管 3/4 的显示，确认程序计数器（PC）的输出是否正确。

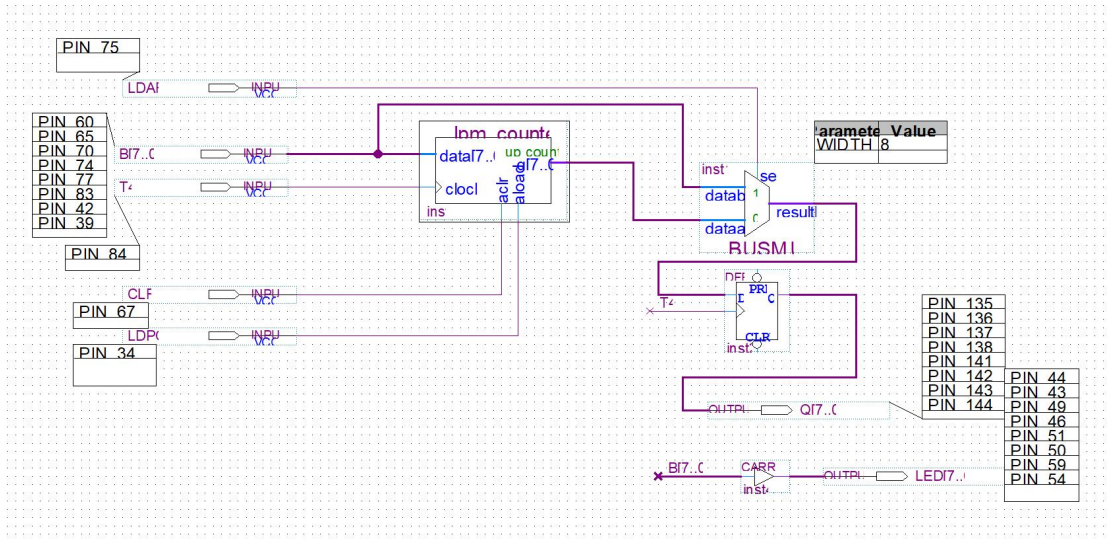
通过按键操作，验证程序计数器的自动加 1 功能以及预置功能是否正常。

对比仿真波形：将硬件实验结果与仿真波形图进行比较，确保实验结果与预期一致

四、 实验步骤及结果

实验任务 1

电路图绘制



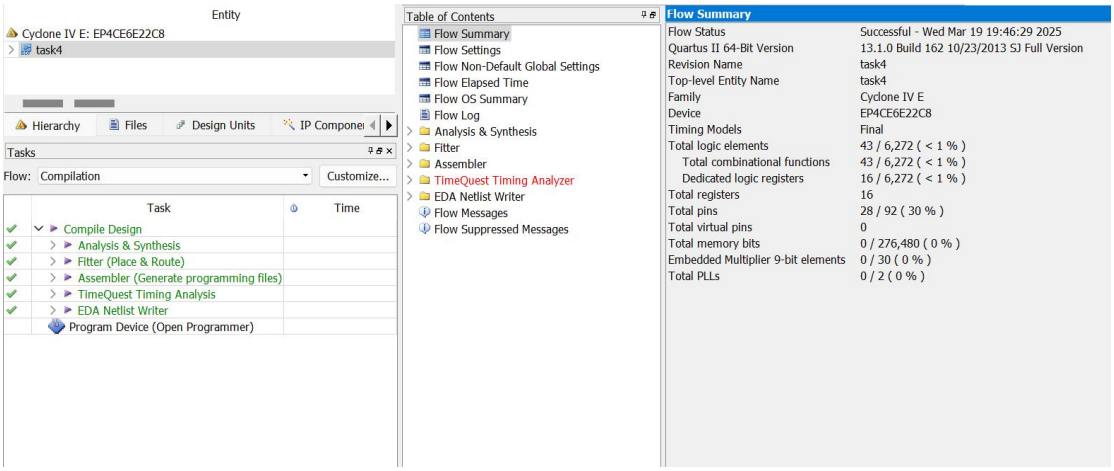
绑定引脚

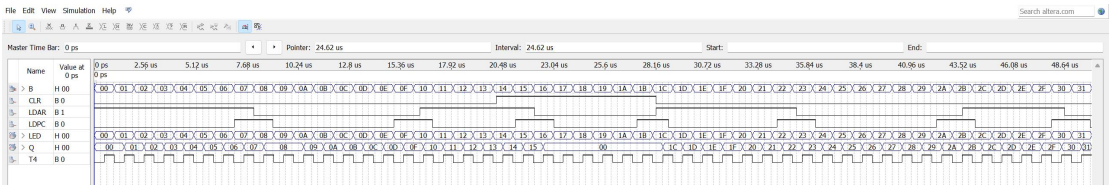
Home | task4.bdf* | Compilation Report - t

<<new>> | Filter on node names: *

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	...		B[0]	Location	PIN_60	Yes			
2	...		B[1]	Location	PIN_65	Yes			
3	...		B[2]	Location	PIN_70	Yes			
4	...		B[3]	Location	PIN_74	Yes			
5	...		B[4]	Location	PIN_77	Yes			
6	...		B[5]	Location	PIN_83	Yes			
7	...		B[6]	Location	PIN_42	Yes			
8	...		B[7]	Location	PIN_39	Yes			
9	...		LDAR	Location	PIN_75	Yes			
10	...		T4	Location	PIN_84	Yes			
11	...		CLR	Location	PIN_67	Yes			
12	...		LDPC	Location	PIN_34	Yes			
13	...		Q[0]	Location	PIN_135	Yes			
14	...		Q[1]	Location	PIN_136	Yes			
15	...		Q[2]	Location	PIN_137	Yes			
16	...		Q[3]	Location	PIN_138	Yes			
17	...		Q[4]	Location	PIN_141	Yes			
18	...		Q[5]	Location	PIN_142	Yes			
19	...		Q[6]	Location	PIN_143	Yes			
20	...		Q[7]	Location	PIN_144	Yes			
21	...		LED[0]	Location	PIN_44	Yes			
22	...		LED[1]	Location	PIN_43	Yes			
23	...		LED[2]	Location	PIN_49	Yes			
24	...		LED[3]	Location	PIN_46	Yes			
25	...		LED[4]	Location	PIN_51	Yes			
26	...		LED[5]	Location	PIN_50	Yes			
27	...		LED[6]	Location	PIN_59	Yes			
28	...		LED[7]	Location	PIN_54	Yes			
29		<<new>>	<<new>>	<<new>>					

编译并下载至 FPGA，仿真

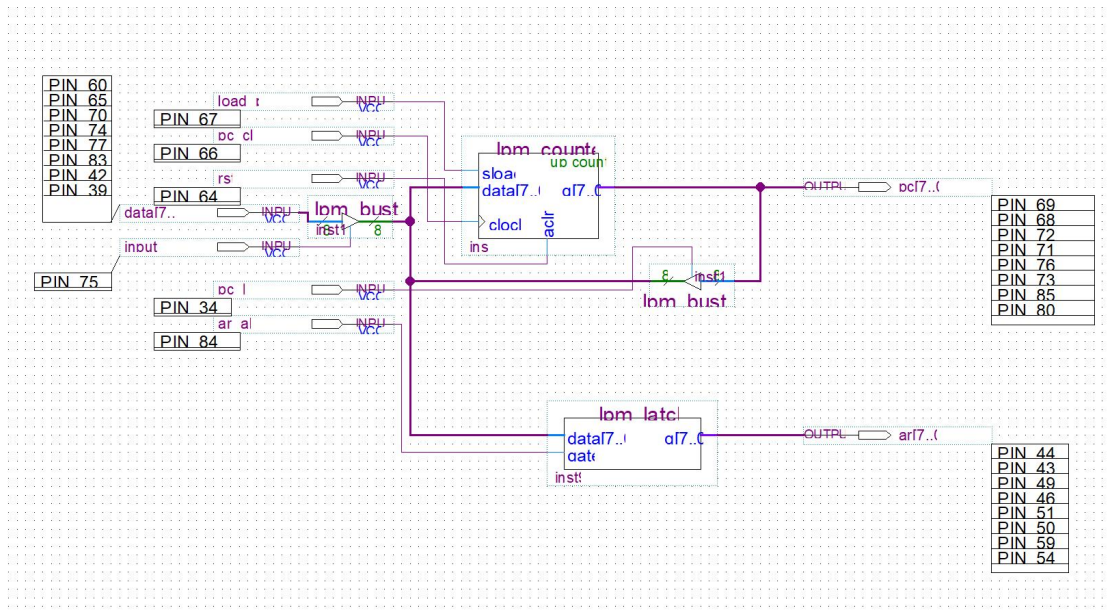




实验结果与预期结果一致。

实验任务 2

电路图绘制

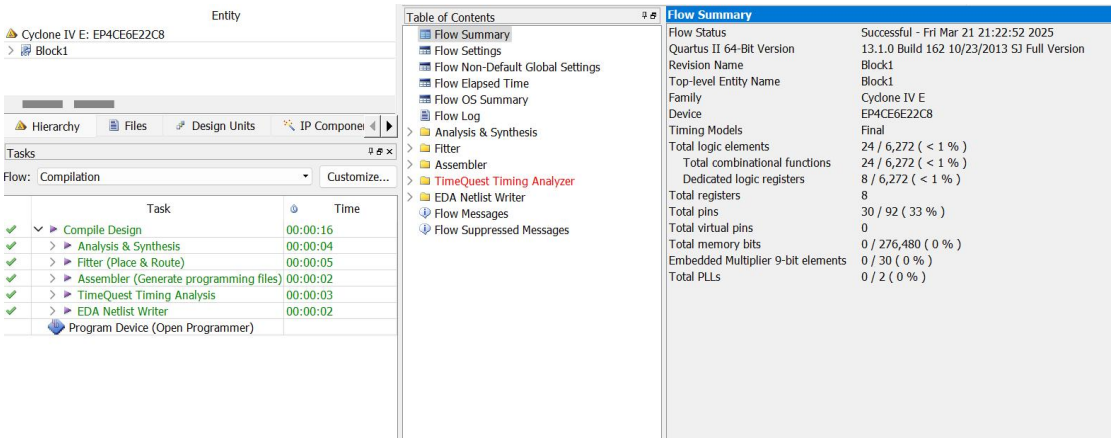


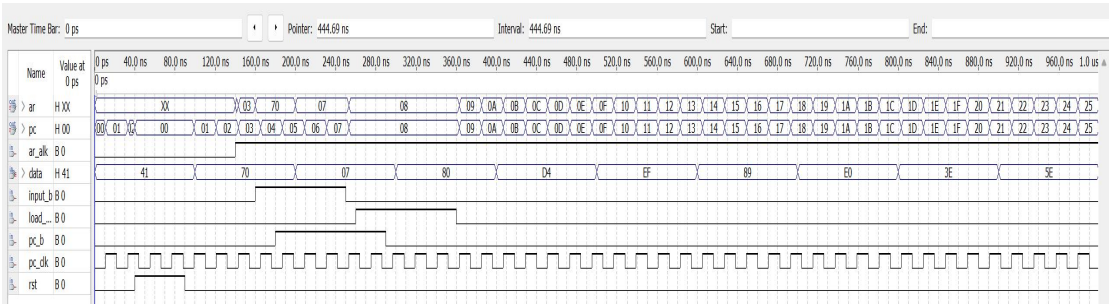
绑定引脚

<<new>> Filter on node names: *

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment
1	...		ar[0]	Location	PIN_44	Yes		
2	...		ar[1]	Location	PIN_43	Yes		
3	...		ar[2]	Location	PIN_49	Yes		
4	...		ar[3]	Location	PIN_46	Yes		
5	...		ar[4]	Location	PIN_51	Yes		
6	...		ar[5]	Location	PIN_50	Yes		
7	...		ar[6]	Location	PIN_59	Yes		
8	...		ar[7]	Location	PIN_54	Yes		
9	...		ar_alk	Location	PIN_84	Yes		
10	...		data[0]	Location	PIN_60	Yes		
11	...		data[1]	Location	PIN_65	Yes		
12	...		data[2]	Location	PIN_70	Yes		
13	...		data[3]	Location	PIN_74	Yes		
14	...		data[4]	Location	PIN_77	Yes		
15	...		data[5]	Location	PIN_83	Yes		
16	...		data[6]	Location	PIN_42	Yes		
17	...		data[7]	Location	PIN_39	Yes		
18	...		input_b	Location	PIN_75	Yes		
19	...		load_pc	Location	PIN_67	Yes		
20	...		pc[0]	Location	PIN_69	Yes		
21	...		pc[1]	Location	PIN_68	Yes		
22	...		pc[2]	Location	PIN_72	Yes		
23	...		pc[3]	Location	PIN_71	Yes		
24	...		pc[4]	Location	PIN_76	Yes		
25	...		pc[5]	Location	PIN_73	Yes		
26	...		pc[6]	Location	PIN_85	Yes		
27	...		pc[7]	Location	PIN_80	Yes		
28	...		pc_b	Location	PIN_34	Yes		
29	...		pc_clk	Location	PIN_66	Yes		
30	...		rst	Location	PIN_64	Yes		
31		<<new>>	<<new>>	<<new>>				

编译并下载至 FPGA，仿真





实验结果与预期结果一致。

五、 实验总结及问题分析

思考题 1，执行分支/转移指令与执行普通指令时，对于实验中两种实现方式，地址单元的操作有何区别？

当执行分支或跳转指令时，无论采用哪种方式，都会从总线上读取目标地址，并将该地址存入 LPM_COUNTER 中，随后继续从该地址取指令执行。

而在执行顺序指令时，两种实现方式均是从 LPM_COUNTER 中读取 PC 的值，依据此值在内存中定位到相应地址的指令。之后，它们会将指令中操作数的地址置于输入数据中。对于第一种方式，这需要提供一个高脉冲给 T4 信号；对于第二种方式，则需使 input_b 信号有效并给 gate 一个高脉冲。不过，当执行跳转指令时，两种方式又会回归到从总线上读取地址并将其存入 LPM_COUNTER 中，然后继续从该地址取指令执行的流程。

思考题 2，从存储器读取运算数据和取指令操作时，对于实验中两种实现方式，地址单元完成的操作有何不同？

在第一种方式中，当需要取指令时，地址单元会将 BUSMUX 设置为 0，此时 LPM_COUNTER 输出的 PC 值会通过 DFF 传输到 AR 中。

而当需要取数据时，地址单元会将 BUSMUX 设置为 1，直接将数据通过 DFF 传输到 AR 中。

相比之下，在第二种方式中，当需要取指令时，指令会依次通过 LPM_BUSTRI、LPM_COUNTER 和 LPM_BUSTRI，最终传输到 LPM_LATCH 并输出。而当需要取数据时，数据会直接通过 LPM_BUSTRI 传输到 LPM_LATCH 并输出。