计算机组成原理课程设计报告

司睿洋* 席梓杰* 孙鹤睿* 刘宜霖*

www.github.com/RuiyangSi/BUPT-ComputerOrganization-CPU

摘要

我们基于 TEC-8 实验平台，成功设计并实现了**基于时序信号控制的流水硬布线控制器**。我们的主要贡献包括：

1. **纠正 TEC-8 平台技术手册的关键理论缺陷**：通过详细实验，我们发现 TEC-8 平台手册说明中关于节拍信号生成机制的描述与实际不符。实际情况是，这些信号的产生并非由 T1 的上升沿触发，而是 **T3 的下降沿**。这一修正**填补了 TEC-8 理论资料的空白**，为未来的教学、研究和开发提供了准确的指导和思考方向。
2. **实现了高性能的流水硬布线控制器**：我们成功构建了以 T3 下降沿为触发条件的流水硬布线控制器。经全面测试，该控制器运行稳定、功能正确。通过精心设计的流程图其输出逻辑与官方微程序控制器的结果**完全一致**，验证了本设计的可行性与精确性。
3. **全面开源项目所有资料**：我们丰富的测试数据集包含 80 余条命令，共计五个任务，并且我们得出了最终卡诺图的化简结果，这些文件包含代码将在 GitHub 上全部开源，旨在为对 TEC-8 平台感兴趣的学者和开发者提供便利，共同推动相关领域的研究与创新。

*表示同等贡献

目录

1 需求分析

我们需要基于 Altera CPM7128 硬件平台，设计并实现一个基于时序信号控制的流水线硬连线控制器。同时满足附加功能，即修改 PC 指针功能，在原指令基础上要求扩指、调试并运行成功。

2 相关理论与技术基础

为方便讲解，我们将在本节对一些概念进行了定义与讲解。

2.1 硬布线控制器

硬布线控制器是将控制部件做成产生专门固定时序控制信号的逻辑电路，产生各种控制信号，这种逻辑电路是由一种门电路和触发器构成的复杂树形逻辑网络，如图 1 所示。当执行不同的机器指令时，通过激活一系列彼此很不相同的控制信号来实现对指令的解释。根据硬布线控制器的基本原理，针对每个控制信号 C_x，可以列出它的函数表达式：

$$C_x = f(I_m, W_i, T_k, B_j) \tag{1}$$

$$C_x \leftarrow f(I_m, W_i, T_k, B_j) \tag{2}$$

其中：I_m 表示机器指令操作码译码器的输出信号，W_i 表示节拍信号发生器输出的节拍信号，T_k 表示时序信号发生器输出的时序信号，B_j 表示状态条件判断信号。

需要注意的是，在公式 1 中我们认为由于是等号所以强调的是在任意时刻均符合上式，而在公式 2 中我们认为由于是 ← 所以是在一定条件下触发。即对于公式 1 我们认为随着输入的改动输出会直接产生变化，我们将其定义成**组合型硬布线控制器**，对于公式 2 我们认为只有当一定条件下才会改变控制信号的值，即控制信号的值具有“记忆”功能，我们将其定义成**时序型硬布线控制器**。

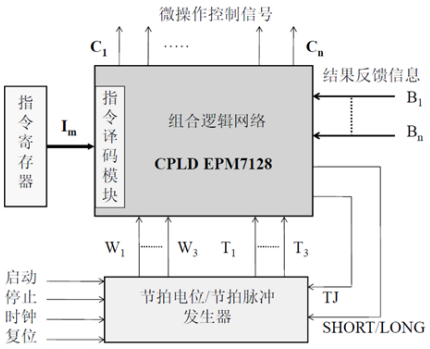

图 1: 硬布线控制器逻辑模块图

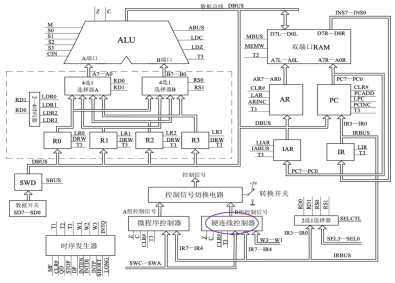

图 2: TEC-8 模型计算机框图

2.2 TEC-8 架构

TEC-8 计算机的整体架构如图 2 所示。本项目的核心任务是设计图中紫色高亮部分的硬布线控制器，即在 CPLD 芯片 **EPM7128** 上实现控制逻辑。架构中的其余模块均为功能固定的预设组件。

我们将在第 3 节详细阐述该架构下的时序系统。为便于读者理解其控制方式，此处以地址寄存器 (AR) 的几个关键控制信号为例进行简要说明：

- **LAR:** 地址寄存器加载使能信号。当其有效时，在 T3 时钟信号的上升沿，数据总线上的内容将被锁存入 AR。
- **ARINC:** 地址寄存器自增信号。当其有效时，在 T3 时钟信号的上升沿，将 AR 的值加一。
- **CLR#:** 地址寄存器异步清零信号（低电平有效），用于立即将 AR 的值清零。

各模块控制信号的定义及控制器引脚说明，详见相关技术手册 [1]，在这里我们不再讲解。

2.3 指令的执行

一条完整的指令是由一个或多个基本操作（称为“微操作”）组成的序列。为了确保这些微操作能按正确的顺序执行，硬布线控制器采用**节拍信号**（如 W1, W2, W3）来划分指令的执行阶段。

在每一个 CPU 周期内，控制器的核心任务是根据当前指令、所处的节拍（W_i）以及系统状态，生成一组特定的控制信号，来精确地触发并执行一个预设的微操作。通过这种方式，控制器驱动着多个微操作在正确的节拍下按序完成，从而实现一条完整指令的功能。

为了适应不同复杂度的指令，TEC-8 采用了可变长度的指令节拍。大部分指令的执行需要两个节拍 (W1, W2)，而部分简单或复杂指令则分别需要一个或三个节拍。

为实现这种灵活性，时序发生器（其位置见图 2 左下角）引入了 short 和 long 两个额外的控制信号。节拍状态的转换逻辑如下：

- **标准两节拍指令：** 在默认情况下（即 short 和 long 信号均无效时），系统会执行标准的 W1 → W2 流程，完成后复位。

- **单节拍指令：**当控制器在 W1 节拍期间发出 short 信号时，时序发生器将不会进入 W2，而是保持在 W1 状态，从而实现单节拍执行。
- **三节拍指令：**当控制器在 W2 节拍期间发出 long 信号时，时序发生器会在 W2 之后额外转换到 W3 状态，以支持需要更多操作步骤的复杂指令。

这种机制使得控制器能够根据指令需求，动态地调整执行周期，从而在灵活性与效率之间取得了良好的平衡。

3 时序系统

时序系统是数字逻辑电路的“脉搏”，它为系统提供了统一的时钟基准，确保所有操作能够同步、协调地进行。正是基于精确的时序控制，寄存器等状态元件才得以可靠地锁存和“记忆”数据，从而使复杂的运算与控制流程成为可能。

鉴于其核心地位，本章将深入剖析 TEC-8 平台的时序系统，详细分析其关键节拍电位的生成机制与逻辑关系，为后续**时序型硬布线控制器**的设计奠定坚实的基础。

3.1 TEC-8 中的时钟信号

在对 TEC-8 模型进行了基本介绍后，我们在此着重分析其核心的时钟信号。通过观察可以得出两个关键的同步规则：

- 对存储器 (RAM) 的写入操作，由 T2 时钟信号的上升沿触发。
- 对所有寄存器的加载操作与处理操作，均由 T3 时钟信号的上升沿触发。

这样设计的好处在于，它将访存和寄存器操作严格地用时钟周期分开，从而显著降低了控制器逻辑的设计复杂度。

3.2 节拍信号与时钟信号的关系

我们在第 2.3 节和第 3.1 节分别探讨了节拍信号与时钟信号的核心作用。一个关键问题随之而来：这两种信号是如何相互作用的？或者说，是什么事件触发了节拍状态（如从 W1 到 W2）的转换？

根据该平台的传统理解或部分技术资料 [1]，通常认为节拍信号的切换发生在 T1 时钟的上升沿。然而，我们通过实验发现，这一描述与硬件的真实行为完全不符。我们的实验最终导向了一个明确且颠覆性的结论，即：**节拍信号的状态转换，是在 T3 时钟信号的下降沿被触发的**。

如图 3 所示，左图为通过 T1 上升沿改变节拍信号，右图为通过 T3 下降沿改变节拍信号，我们将通过实验证明左图是错误的。

为了通过实验验证节拍信号的真实触发机制，我们设计并运行了一段关键的测试程序（其逻辑如算法 1 所示）。该实验的核心思想是，在单拍模式下，观察一次“清零-执行”操作后节拍寄存器的最终状态。

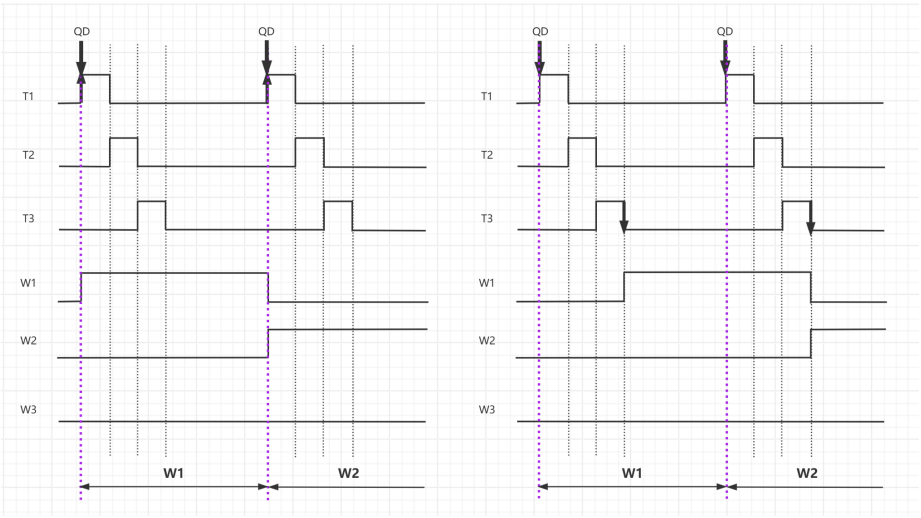

图 3: TEC-8 时序图的修正：错误（左）与正确（右）的对比。注：*为清晰展示触发机制的差异，此图工作于单拍模式 (连拍模式下 T1 上升沿与 T3 下降沿天然对齐)，其中 QD 代表单步执行事件。在初始状态（第一次 QD 按下前），所有节拍及控制信号已通过 CLR# 信号异步清零。*

实验的初始条件与具体步骤如下：

1. 首先，通过 CLR# 信号的下降沿，将所有节拍寄存器（S[3:1]）与控制状态清零。
2. 随后，触发一次单步执行事件（按下 QD 按钮）。

基于图 3 提出的两种不同时序理论，我们对实验结果进行了预测：

- **假设 A（旧理论）：**如果节拍信号的转换由 T1 上升沿触发（如图 3 左图所示），那么在 QD 事件后的第一个 T1 周期，节拍状态就应变为 W1。因此，在随后的 T3 下降沿，节拍寄存器 S[3:1] 应锁存 W1 的值，即二进制的 001。
- **假设 B（新理论）：**如果节拍信号的转换由 T3 下降沿触发（如图 3 右图所示），那么在 QD 事件后的第一个 T3 下降沿之前，节拍状态将始终保持为清零后的 000。因此，当 T3 下降沿到来时，寄存器 S[3:1] 将锁存当前节拍值，结果应为 000。

实验观测结果显示，S[3:1] 的最终状态为 000，这与假设 B 的预测完全吻合，并明确地否定了假设 A。因此，我们得出结论：技术手册中关于节拍信号由 T1 上升沿触发的描述是错误的。在使用示波器输出真实图像后我们得到了正确结论：**正确的触发机制是在 T3 时钟信号的下降沿**。因此，我们推断技术手册中关于 T1 上升沿触发的描述是错误的。至于为何采用 T3 下降沿触发的机制，我们认为这并非源于某个特定的理论，而是一种将系统稳定性置于首位的设计哲学体现。虽然理论上存在多种可行的时序方案，但 TEC-8 平台最终采用了在 T3 下降沿切换节拍的稳健策略，以牺牲潜在的极致速度来换取系统时序的绝对可靠。

Algorithm 1: 实验代码

Input: 异步低电平清零信号 CLR#, 时钟信号 T3, 节拍信号 W[3:1]

Output: 节拍寄存器 S[3:1], 其余控制状态 U

```
if a falling edge occurs on CLR# then
    S[3:1] ← 0;
    U ← 0;
else if a falling edge occurs on T3 then
    S[3:1] ← W[3:1];
end
```

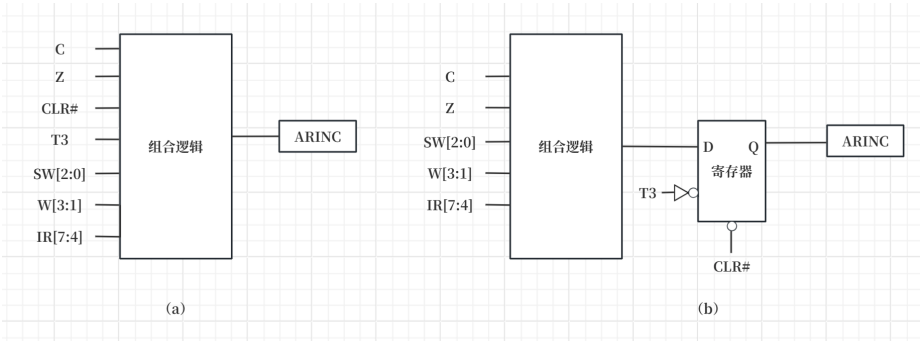

图 4: 控制器范式的逻辑结构对比

4 系统设计

我们已经在前面理解了控制器的理论基础，本节我们将重点讲述如何设计一个控制器。

4.1 设计控制器的两种范式

根据第 2.1 节的定义，硬布线控制器在设计上主要存在两种范式：组合型与时序型。它们的核心区别在于控制信号的生成方式，这直接决定了其工作特性与稳定性，如图 4 所示。

- **组合型硬布线控制器：**如图 4 左测所示，这种范式完全由组合逻辑门电路构成。其输出（即控制信号）是当前所有输入（指令、节拍、状态等）的瞬时函数。这意味着任何输入的微小波动或毛刺，都可能会被直接、实时地反映到输出端，容易因信号竞争而产生冒险，从而导致系统工作异常。
- **时序型硬布线控制器：**如图 4 右侧所示，这种范式在组合逻辑的基础上，引入了时钟控制的存储元件（如触发器或寄存器）。其控制信号的更新并不会随输入实时变化，而是在一个确定的时钟事件（在 `TEC-8` 中，即 `T3` 信号的下降沿）到来时，才将组合逻辑的计算结果同步锁存到输出。

综合比较，**时序型控制器具有显著优势**：由于其同步更新的特性，它能有效“过滤”掉输入信号在非同步时刻的抖动和毛刺，具有更强的抗干扰能力和时序稳定性，从根本上避免了组合逻辑中常见的冒险问题。

表 1: 指令系统

名称	助记符	功能	指令格式		
			IR7-IR4	**IR3-IR2**	**IR1-IR0**
空指令	`NOP`	无	0000	XX[1]	XX
加法	`ADD Rd, Rs`	Rd $\leftarrow$ Rd[2] + Rs[3]	0001	Rd	Rs
减法	`SUB Rd, Rs`	Rd $\leftarrow$ Rd $-$ Rs	0010	Rd	Rs
逻辑与	`AND Rd, Rs`	Rd $\leftarrow$ Rd $\wedge$ Rs	0011	Rd	Rs
加 1	`INC Rd`	Rd $\leftarrow$ Rd + 1	0100	Rd	XX
取数	`LD Rd, [Rs]`	Rd $\leftarrow$ [Rs]	0101	Rd	Rs
存数	`ST Rs, [Rd]`	Rs $\rightarrow$ [Rd]	0110	Rd	Rs
C 条件转移	`JC addr`	如果 C=1, 则 PC $\leftarrow$ @[4] + offset[5]	0111	offset	
Z 条件转移	`JZ addr`	如果 Z=1, 则 PC $\leftarrow$ @ + offset	1000	offset	
无条件转移	`JMP [Rd]`	PC $\leftarrow$ Rd	1001	Rd	XX
输出	`OUT [Rs]`	DBUS $\leftarrow$ Rs	1010	XX	Rs
移动值	`MOV Rd, Rs`	Rd $\leftarrow$ Rs	1011	Rd	Rs
比较	`CMP Rd, Rs`	Rd $-$ Rs	1100	Rd	Rs
逻辑非	`NOT Rd`	Rd $\leftarrow \neg$Rd	1101	Rd	XX
停机	`STP`	暂停运行	1110	XX	XX

[1]XX 代表任意值.

[2]Rd 代表目的寄存器号.

[3]Rs 代表源寄存器号.

[4]@ 代表当前 PC 的值.

[5]offset 是一个 4 位的补码有符号数.

因此，考虑到系统可靠性的要求，本项目最终采用**时序型硬布线控制器**作为设计方案。

4.2 指令集架构

指令集架构（ISA, Instruction Set Architecture）是计算机的抽象模型，它定义了处理器能够理解和执行的所有指令、寄存器及寻址模式，是软件与硬件之间最核心的规约。因此，一个明确的 ISA 是设计任何（包括硬布线）控制器的根本依据和首要步骤。

本项目以 TEC-8 平台的标准指令集为基础，并根据第 1 节的需求分析对其进行了功能扩充。最终确定的完整指令集架构详见表 1。

4.3 顺序时序硬布线控制器

4.3.1 非取指执行部分

我们设计了一个不依赖于外部节拍信号（W1, W2, W3）的控制器设计方法，称之为“内部状态机”。该方法通过构建一个自定义的状态机来精确控制非取指阶段的微操作流程，其状态转移逻辑如图 5 所示。

该方案的核心是引入一个内部状态变量 w（图中红色标注），其值明确地指示了状态机在下一个 CPU 周期应迁移到的目标状态。例如，在一个操作执行框图的末尾，会指定下一个 w 的值，从而驱动流程前进。

特别地，对于“写寄存器”和“读寄存器”等多步操作，在完成最后一个步骤后，我们将 w 置为一个确定的空闲状态 w0。在此状态下，所有控制信号均无效。通过这种设计使得硬布线控制器的状态跳转逻辑，在概念上模拟了微程序控制器的步进执行方式。

4.3.2 取指执行部分

在取指执行部分，我们依然采用内部状态机模型进行精确控制。我们在这里重点阐述如何实现第 1 节所要求的扩展 PC 功能，即在程序开始执行前，将数据开关上设定的值加载至程序计数器（PC），作为程序的起始地址。

为实现该功能，我们引入了一个状态标志位 FLAG。系统启动时，FLAG 默认为 0，此时控制器允许用户通过数据开关输入期望的 PC 初始值；该加载操作完成后，FLAG 将被置为 1，控制器随之进入常规的取指执行阶段。这一完整流程如图 6 所示。

图 5: 非指令执行模式下的时序控制器流程图 (取指执行分支在图 6,7 中给出)

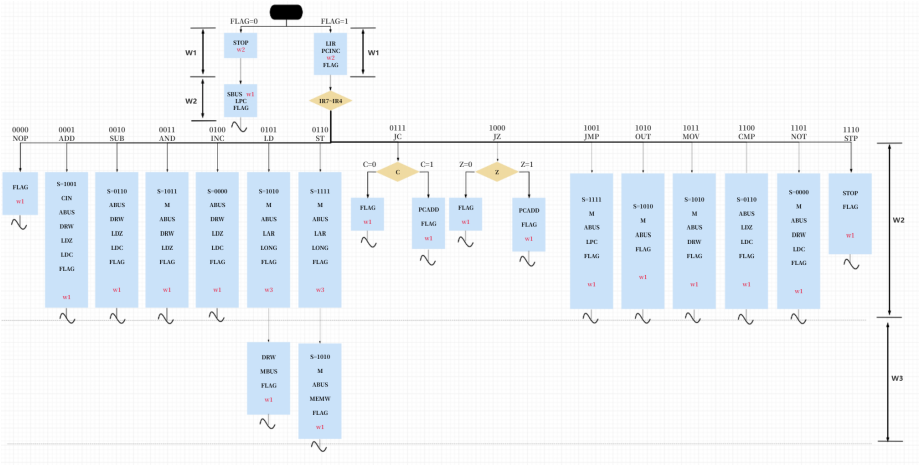

图 6: 指令执行模式下的顺序时序控制器流程图

4.4 流水时序硬布线控制器

流水与顺序的区别只在取指执行阶段，故在非取指执行比分与第 4.3.1 节描述方法一样。故在下面只讲述流水的设计。

4.4.1 流水中存在的冒险分析

设计流水线控制器的首要任务是识别并解决潜在的冒险问题，以确保指令能够正确、高效地执行。我们在 TEC-8 架构上实现一个两级流水线，即将第一条指令的“执行”阶段（EX）与第二条指令的“取指”阶段（IF）并行处理。基于此模型，我们对两种经典的流水线冒险进行分析：

- **结构冒险：**当两条指令在同一时刻需要访问同一个硬件资源时，会产生结构冒险。在 TEC-8 架构中，由于采用了双端口 RAM（如图 2 所示），“取指令”（通过程序计数器 PC 访问右端口）和“执行阶段的访存”（通过地址寄存器 AR 访问左端口）可以并行进行，互不干扰。因此，我们设计的两级流水线在理论上不存在结构冒险。
- **控制冒险：**当处理器无法确定下一条应该执行的指令地址时，会产生控制冒险，这主要由分支或跳转指令引起。在这个两级流水线中，当第一条指令（例如 JMP）处于“执行”阶段并计算出新的 PC 值时，第二条指令已经在“取指”阶段被取出来了（依据旧的 PC+1 地址）。如果跳转发生，那么已经被取出的第二条指令就是错误的，必须被废弃。因此，所有转移和跳转指令（JC, JZ, JMP）都会引入控制冒险。

综上所述，我们的流水线设计需要重点解决由条件判断和无条件跳转带来的控制冒险问题。

4.4.2 取指执行部分

根据第 4.4.1 节的分析，为规避控制冒险，我们对 JC，JZ，JMP 等控制转移指令不送入流水线，而其余指令则正常执行。详细流程如图 7 所示。

基于此我们可视化了流水与顺序时空图展示，如图 8,9 所示。从这里可以发现通过流水优化可以减少运行节拍数量。

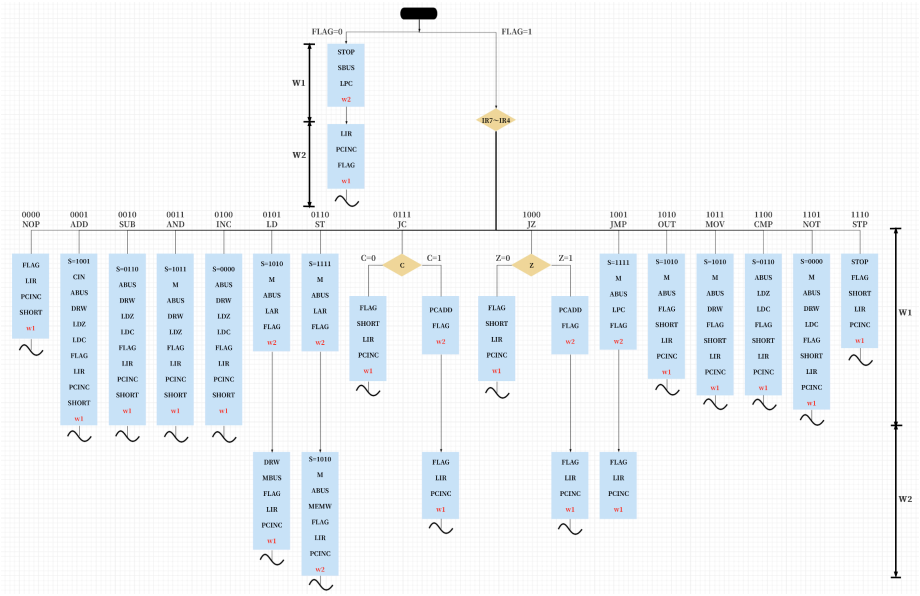

图 7: 指令执行模式下的流水时序控制器流程图

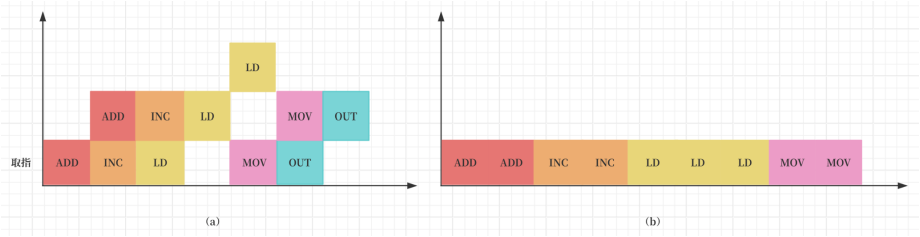

图 8: 流水与顺序时空图展示

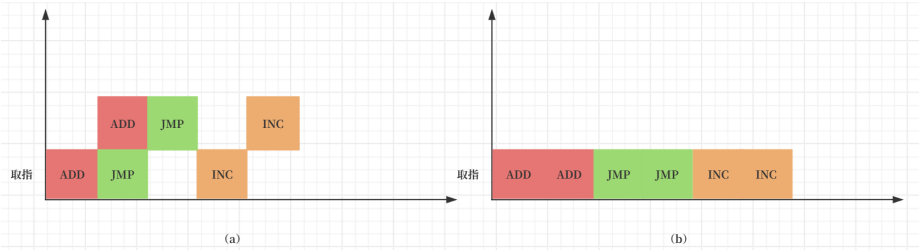

图 9: 流水与顺序时空图展示

5 系统测试

5.1 测试集介绍

为全面验证我们所设计的顺序及流水线控制器的逻辑完备性与功能正确性，我们精心设计了六组具有明确针对性的测试数据集，**总指令数超过 80 条**。这些测试集旨在实现对各种复杂情况的全面覆盖。

5.2 测试数据展示

本节将展示一组具有代表性的测试用例，以具体说明我们所设计控制器的功能正确性。该测试用例的初始状态设定为：R0=80H, R1=AAH, R2=55H, R3=01H 以及 Mem[80H]=CCH。其完整的指令序列详见表 2。经真机测试，程序在地址 AAH 处执行 STP 指令后正常停机，各寄存器、内存及标志位的最终状态均与预期一致，具体如下：

```
R0: A9H        R1: A9H  R2: 01H  R3: 00H
Mem[80H]: AAH  C: 1     Z: 1
```

这一结果验证了所设计控制器在处理算术、逻辑、访存及跳转指令时的正确性。为保证报告篇幅简洁，每一步执行的详细状态变化，连同其余五组测试用例，**均已在我们的项目网站开源**。

表 2: 测试数据展示

地址	**程序指令**	**机器码**
00H	NOP	00000000
01H	ST R1, [R0]	01100001
02H	MOV R2, R1	10111001
03H	LD R1, [R0]	01010100
04H	ADD R1, R3	00010111
05H	INC R3	01001100
06H	SUB R1, R3	00100111
07H	NOT R2	11011000
08H	AND R2, R1	00111001
09H	CMP R2, R3	11001011
0AH	JZ ODH	10000010
0BH	JC OEH	01110010
0CH	INC R3	01001100
0DH	INC R3	01001100
0EH	MOV R0, R3	10110011
0FH	SUB R3, R3	00101111
10H	MOV R0, R1	10110001
11H	JMP [R0]	10010000
...	...	...
A9H	OUT R3	10100011
AAH	STP	11100000

6 思考与讨论

6.1 T1 时钟信号的作用分析

我们在第 3.1 节中明确了 T2 和 T3 信号在数据访存与寄存器操作中的核心作用，并在第 3.2 节中证实了节拍状态的转换由 T3 的下降沿触发。然而，一个重要的问题依然存在：T1 时钟信号在其中扮演了什么角色？尽管在 TEC-8 的主要数据通路图（图 2）中未见其直接应用，但我们认为，T1 的存在是保障整个系统时序稳定性的关键，其核心作用是提供一个必要的“缓冲”周期。

为了理解这一点，我们可以设想一个没有 T1 信号的简化时序模型。在这种模型下，一个周期的 T3 下降沿将与下一个周期的 T2 上升沿对齐。这会引发一个严重的时序问题：

1. 在 T3 下降沿，产生下一个周期所需的控制信号。
2. 如果下一个周期 T2 上升沿与 T3 下降沿对齐，那么新的控制信号在还未完全生成并稳定下来，就到达了它需要发挥作用的时刻，这会导致操作的错误。

因此，T1 周期的引入，实质上是在两个连续的关键操作（上一周期的输出控制信号和本周期的访存）之间，插入了一个宝贵的“缓冲区”。它确保了在 T3 下降沿完成输出控制信号后，控制器有充足的时间（整个 T1 周期）来根据新的节拍和指令，稳定地生成下一个周期（T2,T3）所需的所有控制信号，是保证设计可靠、稳健的关键。

6.2 从组合逻辑到时序逻辑的系统性转换方法

在控制器设计初期，通常会先构建一个组合逻辑模型，但如第 4.1 节所述，纯组合逻辑控制器易受毛刺和竞争冒险的影响，稳定性不足。因此，需要将其系统性地转换为时序逻辑。一种常见的、但设计上较为复杂的方案是构建全新的自定义状态机（如第 4.3.1 节的探索方案）。

然而，基于我们对 TEC-8 平台真实时序的精确理解（即节拍由 T3 下降沿触发），我们发现并总结出一种更为直接、高效的转换方法。该方法包含两个关键步骤：

1. **对所有输出进行寄存：**这是最核心的一步。我们将每一个控制信号的输出端，都增加一个由 T3 下降沿同步的 D 型触发器（寄存器），如图 4 所示。即从原先的组合逻辑 (公式 1) 被改造为时序逻辑 (公式 2)。通过引入寄存器锁存输出值，我们使得所有控制信号只在 T3 的下降沿这一确定的时刻更新，从而彻底“过滤”掉了输入信号的抖动，根除了组合逻辑的冒险问题。

2. **对控制条件进行“提前一拍”设计：**引入寄存器虽然解决了稳定性问题，但它也带来了一个副作用：所有控制信号的输出都引入了一个节拍周期的延迟。原先在 W2 节拍下需要生效的信号，现在要等到 W3 节拍才能从寄存器输出。为了补偿这个延迟，我们必须对产生信号的组合逻辑条件进行“预判”，即提前一个节拍来计算。具体做法是，在编写逻辑代码时，所有原来由 W2 节拍信号作为条件的逻辑，现在都改为由 W1 作为条件；原来由 W3 作为条件的，则改为由 W2 作为条件。这样，控制器在 W1 周期就已经计算好了 W2 周期所需的信号，并在 T3 下降沿将其存入寄存器，从而确保在 W2 周期开始时，正确的控制信号已经准备就绪。

该“输出寄存 + 条件提前”的两步法，被证明是从一个纯组合逻辑模型，便捷、可靠地转换到高性能时序型硬布线控制器的系统性方法。

7 未来工作

我们将在这节提出我们并没有完成的工作以及还未解决的问题。

7.1 通过逻辑化简优化控制器

除了实现基本功能外，我们还探讨了进一步优化控制器硬件资源占用的可能性。一种经典且有效的方法是采用卡诺图对控制信号的生成逻辑进行化简。

理论上，通过对每一个控制信号的逻辑表达式（特别是涉及指令操作码 `IR[7:4]` 的部分）进行卡诺图化简，可以得到最简的“与或”表达式。这会带来两方面的好处：

1. **减少逻辑门数量：**最简表达式意味着实现相同功能所需的逻辑门更少，这可以降低在 CPLD 或 FPGA 上占用的硬件资源。
2. **降低传播延迟：**更少的逻辑门级数通常也意味着更短的信号传播延迟，这有助于提升控制器的整体性能，甚至可能提高系统的最高时钟频率。

我们已经完成了该优化方案的理论分析，得出了所有控制信号的指令集组合译码表以及对应的卡诺图化简结果，并已在我们的**开源项目**中提供。然而，由于课程设计时间的限制，我们最终实现的版本未集成这一深度优化。尽管如此，这无疑是一个有价值的探索方向，可以用于未来进一步提升该控制器性能的研究中。

7.2 组合型硬布线控制器吞指令问题

在对组合型硬布线控制器的测试中，我们观察到一种“吞指令”的异常现象，例如，一个预期的 `ADD, INC, SUB` 指令序列，在实际执行中会错误地变为 `ADD, SUB, SUB`。我们推断，此现象的直接原因是取指周期中的 `PCINC` 与 `LIR` 信号之间存在时序上的竞争冒险。然而，对于这种竞争在具体门级电路中的确切物理成因，以及为何时序型控制器能从根本上规避此问题，我们承认仅通过当前实验手段难以给出详尽的硬件底层解释。但从体系结构设计的更高层面来看，时序型控制器通过将 `LIR` 和 `PCINC` 等关键操作在寄存器中“记忆”住，从架构层面消除了此类竞争风险。

8 总结

本项目基于 TEC-8 实验平台，成功设计、实现并验证了一款高性能、基于时序逻辑的流水时序型硬布线控制器。

项目的核心贡献在于，我们通过实验发现并纠正了 TEC-8 平台官方技术手册中关于时序控制的一个关键理论缺陷。我们明确地证明了，作为指令执行阶段划分依据的节拍信号（W1，W2，W3），其状态转换的真实触发事件是 T3 时钟的下降沿，而非手册所称的 T1 上升沿。这一发现不仅为我们后续的控制器设计奠定了正确的时序基础，也为该平台的未来教学与研究提供了重要的理论修正。

在设计阶段，我们首先对控制器的两种设计范式进行了深入辨析，阐明了时序型控制器相比于组合型控制器在抗干扰和稳定性上的显著优势，并因此确立了我们的设计方案。我们首先实现了一个功能完备的顺序时序控制器，随后，为提升性能，我们进一步设计并实现了一个两级流水线控制器。在流水线设计中，我们对潜在的结构冒险和控制冒险进行了分析，并通过对跳转指令不进行流水处理的策略，有效规避了控制冒险带来的风险。

为保证设计的正确性与完备性，我们构建了包含六组、总计超过 80 条指令的测试集，对所设计的控制器进行了全面测试。测试结果表明，我们设计的控制器运行稳定，功能正确，其输出逻辑与官方微程序控制器的结果完全一致。

最终，我们将本项目的所有设计文档、流程图、测试用例以及作为理论成果的卡诺图化简结果，全部在 GitHub 平台开源，以期为对 TEC-8 平台感兴趣的研究者和开发者提供一份有价值的参考。

A 团队分工

我们小组分工明确，整个项目流程严格遵循了“理论分析-实现测试-总结归档”的三个阶段。在项目初期，全体成员共同完成了核心理论的分析与探讨；项目中期的核心任务为技术验证，我们对组合逻辑与两种时序逻辑共三版控制器代码进行了全面的编写和系统测试，并同步构建了项目的开源资料；项目后期，团队则合力完成了技术报告的撰写、校对与整合。

具体分工如表 3 所示。尽管每位成员在不同阶段的侧重点有所区别，但项目的整体推进与成功离不开所有人的紧密协作与关键投入，**因此我们一致认为团队成员均为同等贡献**。

表 3: 团队分工

成员	**分工**
司睿洋	理论分析, 代码编写, 文档编写
席梓杰	理论分析, 开源资料编写, 辅助系统测试, 文档编写, 文档检查
孙鹤睿	理论分析, 开源资料编写, 辅助系统测试, 文档编写, 文档检查
刘宜霖	理论分析, 代码检查, 主要系统测试, 文档编写，文档检查

B 工作日志

B.1 2025.06.29(周日)

我们通过问题驱动的方式讨论了以下问题，为我们后续项目的开发奠定了基础：

课题选择 我们一致认为，项目应从基础功能入手，逐步完善并实现拓展功能。在时间充裕的前提下，再挑战更具探索性的自主学习类课题。

指令集架构（ISA）的选择 考虑到项目初期聚焦于基础功能的实现与拓展，我们决定采用团队熟悉的 `TEC-8` 指令集架构。这一选择将统一我们的开发标准，为后续代码编写提供便利。

团队分工策略 经讨论，我们认识到本项目代码难以进行多人并行编写。因此，核心目标转为如何高效、准确地完成代码。我们制定了如下分工模块：1) 指令流程图与译码表设计；2) 代码实现；3) 系统仿真与测试；4) 创新点探索；5) 技术报告撰写。此分工旨在明确职责，提高效率。

报告撰写规范 鉴于报告工作量较大，需要团队协作完成。为规避跨平台兼容性问题、Microsoft Word 复杂的排版操作以及 Markdown 的格式局限性，我们决定采用 LaTeX 进行报告撰写，以确保文档的专业性与一致性。

开发环境与版本控制 项目将基于 `TEC-8` 架构，因此选用我们熟练掌握的 Quartus 作为主要开发平台。为实现高效的版本控制与代码同步，所有项目文件将通过 Git 进行管理，并托管于 GitHub 仓库。

潜在创新点探讨 我们对潜在的创新点进行了理论探讨，主要集中在两个方向：一是架构拓展，例如设计带有阵列乘法器的 ALU，通过拓展指令集来体现其加速优势；二是指令集优化，设计效率更高、功能更实用的新指令。目前，架构拓展的挑战在于项目是否允许修改 ALU 或使用额外的 I/O 引脚；而指令集优化的创新性可能相对较弱。为此，我们指派一名成员专项研究这两种方案及其他可能路径的可行性与创新潜力。

流水线控制器设计方案　鉴于基础的硬布线控制器实现相对简单，我们将精力重点投入到流水线控制器的设计中。我们分析了两种流水线实现范式:。第一种是基于**指令周期**进行宏观划分的两阶段流水线，其核心思想是将一条指令完整的“执行阶段”与下一条指令的“取指阶段”进行重叠。第二种是基于**时钟周期**进行微观划分的经典多级流水线（如五段式），它力求将指令周期分解为多个更精细的、且每个都能在一个时钟周期内完成的独立阶段。考虑到 TEC-8 的架构限制（如缺少 Cache、缓冲寄存器等），实现第二种方案存在困难。因此，我们初步决定采用第一种方案，即并行化当前指令的执行周期与下一条指令的取址周期。该方案的关键挑战在于精确划分执行周期内的控制信号以避免资源冲突。此项研究已分配给一名成员进行深入分析。

测试方法　我们一致认为，严谨的测试是保障代码质量的关键。为克服纯硬件测试的繁琐性与验证局限性，我们计划采用仿真测试与真机验证相结合的策略。由于团队对仿真测试流程尚不完全熟悉，已安排一名成员专项研究并制定详细的仿真测试方案。

遗留问题　由于时间所限，部分理论问题尚待进一步探究，包括：TEC-8 中控制信号与时钟信号共同作用于控制逻辑的原理（对比单控制信号方案）；T1 时钟信号的具体用途；T2/T3 时钟周期的设计意图，以及这些设计与标准计算机体系结构理论的异同。这些问题将作为我们后续学习与研究的重点。

B.2　2025.06.30(周一)

我们在昨天实现了了一份基于组合逻辑的（即控制信号不具备记忆功能），但是今天上课就被老师批评了这种设计思路，详细说明见第 4 节。基于此我们设计出第一版基于时序逻辑控制的代码，但是发生了空拍问题。同时为了推进项目的并行性我们同时去推理其中组合逻辑部分的公式以及卡诺图化简，与测试功能。公式推进由于双人合作十分顺利，测试同学也掌握了高超的测试技巧。但是代码部分的问题始终无法解决，在胡乱乱改以结果为导向后正确了，但是**理论**知识不符合，即摁一次 QD 在 T3 下降沿前 W3/W2/W1 信号均为 0，基于此我们认为**理论部分存在极大问题**，但由于时间太晚了当天我们并没有深究为什么而重写了 W 信号，改成了我们上课

认为是对的 W 信号。

B.3 2025.07.01(周二)

我们重写了 W 信号对着 100（写寄存器）进行了初步的验证，发现消除了错拍的行为，于是我们认为自己的项目有救了（因为所有状态都是一样的，只要能写对一个就可以把所有都写对），于是我们在今天很快编写出一份顺序时序型硬布线控制器代码并且开始编写测试数据集以及流水的流程图。晚上由于闲来无事我们对着昨天遗留的问题再次做了个小实验进行了验证，发现理论确实存在问题。

B.4 2025.07.02(周三)

由于昨天已经写完了顺序时序型硬布线控制器和流水的流程图，所以大概只用了 20min 就写完了一版流水硬布线控制器并且经过测试完全没有问题，所以我们开始进行写作环节（画图与文字分开并行执行），任务在今天就已经结束了。

C 个人心得体会

C.1 司睿洋

在正文详述的理论收获之外，我更想分享一些实践中的切身体会。

首先是协作模式的突破。我第一次走出熟悉的“宿舍圈”与新伙伴组队，真正体验了“聚是一团火，散是满天星”的协作：每个人在自己的模块上独立攻坚，又能合力为共同的目标添砖加瓦。

其次，这次经历让我对“知行关系”有了更深的洞见。它让我坚信，代码本身并非障碍，无法融会贯通的理论才是。当我们将理论彻底理清后，编码过程便行云流水，一气呵成且精准无误，这给了我极大的信心。

然而，项目带来的最大惊喜，是赋予了我一种更高维度的勇气——当实践结果与理论预测相悖时，敢于去审视甚至挑战理论本身的勇气。这份宝贵的收获，让我深刻理解了何为对**理论的忠诚与审慎的怀疑**。我希望在未来的科研道路上，**能永远保有这份清醒与勇气**。

C.2 席梓杰

系统优化思维的建立 与组员合作进行卡诺图化简的过程让我体会到了工程优化的重要性。我们将控制信号冗余表达式化简为最优形式，不仅节省了宝贵的 FPGA 资源，更重要的是培养了我在资源约束下寻求最优解的工程思维。

测试方法论的系统提升 在设计测试集时，我逐渐认识到测试的针对性和全面性同等重要。针对 JC、JZ 等不同的跳转指令，我们设计了专门的测试用例来验证各种边界条件和异常情况。更重要的是，当第一次测试仅关注最终结果而忽略中间过程导致错误定位困难时，我们改进了测试策略：逐步观察每个执行阶段的 D7-D0 数据灯状态。正是这种细致的过程分析让我们发现了 ST 指令中 Rd 和 Rs 操作码写反的关键错误，这让我深刻理解了过程验证对于复杂系统调试的重要性。

流水线架构设计的深入理解 为了更好地支持流水线功能的开发和测试，我和组员将指令流程图的重新整合，在这个过程中，我对 SHORT 和 LONG 控制信号的作用机制以及冒险冲突有了更本质的认识。这种对微程序控制

器时序控制的深入理解，让我从单纯的功能实现者成长为能够把握系统架构全貌的设计者。

C.3 孙鹤睿

在硬布线控制器的设计实践中，我首先深入研究了其 CPU 架构与信号传输时序，绘制了基础及流水线两种模式的控制器流程图，细致梳理了寄存器操作、存储器访问与取指流程。这一过程使我对控制信号的状态变化有了更深刻的理解，为后续理论学习奠定了坚实基础。

然而，当将流程图转化为 Verilog 代码并进行真机测试时，我们意外发现 CPU 周期中多出了一个空拍现象。尽管通过计算机组成原理课程的学习，我已对硬布线控制器理论有了系统掌握，但对于时钟周期与 CPU 周期的转换逻辑仍存在模糊之处，导致一时难以解释这一异常。面对挑战，我们并未回避，而是通过一整天的头脑风暴，提出了自己的理论假设，并最终完善了时钟周期的解决方案，同时验证了原有理论的局限性。这一突破让我深刻体会到：**理论的边界往往在实践中被重新定义，而真正的理解源于对细节的执着追问**。

随后，我与团队成员协作完成了取指指令的时序设计。针对代码冗余问题，我们创新性地运用卡诺图进行逻辑化简，最终实现了更简洁高效的时序表达式。**化繁为简的过程，不仅是技术的优化，更是思维的淬炼**。经过反复测试，基础与流水线硬布线控制器的功能均得到验证，代码设计的正确性得以确认。

这次课程设计不仅深化了我对硬布线控制器与 CPU 架构的理解，更将抽象理论转化为生动实践，弥补了认知空白。在团队合作中，我们思维碰撞、分工明确、互学共进，短短数日收获颇丰，既领略了实践的乐趣，也感受到协作的力量。**当逻辑的严谨遇上创造的激情，工程的魅力便在此刻熠熠生辉**。

C.4 刘宜霖

在计组课设的实验中我主要作为测试人员参与这个硬布线控制器项目，我有以下几点深刻体会：

理论与实践的反差带来的成长 在测试 W 信号生成机制时，我们惊讶地发现 TEC-8 官方文档的理论描述与硬件实际行为存在差异。这个发现让我深

刻体会到：硬件设计不能完全依赖文档，必须通过严谨的测试来验证理论假设。当测试数据反复证明 W 信号实际由 T3 下降沿触发（而非文档所述的 T1 上升沿）时，这种”颠覆认知”的体验让我对数字电路时序控制有了更本质的理解。

测试策略的重要性　我们采用测试与代码编写并行的策略：在测试没有流水线的代码的同时编写下一版含流水线的代码，提高了工作效率

逆向工程思维的锻炼　当遇到”胡乱修改后偶然正确”的情况时，我们没有止步于表面现象，而是通过分析代码逐步执行的逻辑，锁定导致代码正确或出错的关键位置，尝试通过理论进行解释并用逻辑分析仪捕获完整指令周期信号等方式最终发现是 T3 下降沿的亚稳态导致控制信号竞争，这个调试过程极大提升了我的硬件问题诊断能力。对严谨性的新认知

这个项目让我从单纯的”功能验证者”成长为能主动参与架构设计的测试人员，特别是对时序逻辑的测试方法论有了系统性提升，这为未来从事更复杂的数字系统验证工作打下了坚实基础。

D 调试过程中遇到的问题

得益于前期扎实的设计与论证并且安排的写代码同学经验丰富，本项目的代码实现过程十分顺畅，未进行任何因代码本身问题所进行的调试过程。

但是项目开发过程中遇到的主要挑战，是在设计初期发现第一版时序型硬布线控制器存在“空拍”现象。我们针对此问题进行了深入分析并给出了解决方案，具体细节已在第 3.2, 4.3.1, 6.2 节中详述，在这里不再介绍。

参考文献

[1] 清华大学科教仪器厂. *TEC-8* 计算机组成和数字逻辑实验系统指导书.