圣诞小火车奇遇之旅

第 12 组

- 易秋月 3210105945
- 林智扬 3210104118
- 李欣奇 3210105139

目录

目录

微控制器介绍

一、微控制器介绍

1、ARM Cortex-M 系列微控制器概况：

ARM 公司从 ARMv6-ARMv7 时代开始 ARM 开始使用 A、R、M 系列来命名其新的处理器。

A 系列为应用处理器（Application Processor），其中 A 可以理解为 Application，面向移动计算，智能手机，服务器等市场的的高端处理器。这类处理器运行在很高的时钟频率（超过 1GHz），支持像 Linux，Android，MS Windows 和移动操作系统等完整操作系统需要的内存管理单元（MMU）。如果规划开发的产品需要运行上述其中的一个操作系统，需要选择 ARM 应用处理器.

R 系列为实时处理器（Real-time Processors），其中 R 为 RealTime，面向实时应用的高性能处理器系列，例如硬盘控制器，汽车传动系统和无线通讯的基带控制。多数实时处理器不支持 MMU，不过通常具有 MPU、Cache 和其他针对工业应用设计的存储器功能。实时处理器运行在比较高的时钟频率（例如 200MHz 到 >1GHz ），响应延迟非常低。虽然实时处理器不能运行完整版本的 Linux 和 Windows 操作系统，但是支持大量的实时操作系统（RTOS）。

M 系列为微控制器处理器（MicrocontrollerProcessors），其中 M 指的是 Microcontroler 目前主要有 M0、M0+、M3、M4、M7 以及新发布不久的基于 ARMv8-M 构架的 M23、M33，其中 M23 为 M0&M0+的升级，M33 为 M3、M4 的升级。微控制器处理器通常设计成面积很小和能效比很高。通常这些处理器的流水线很短，最高时钟频率很低（虽然市场上有此类的处理器可以运行在 200Mhz 之上）。并且，新的 Cortex-M 处理器家族设计的非常容易使用。因此，ARM 微控制器处理器在单片机和深度嵌入式系统市场非常成功和受欢迎。性能天梯如下图：

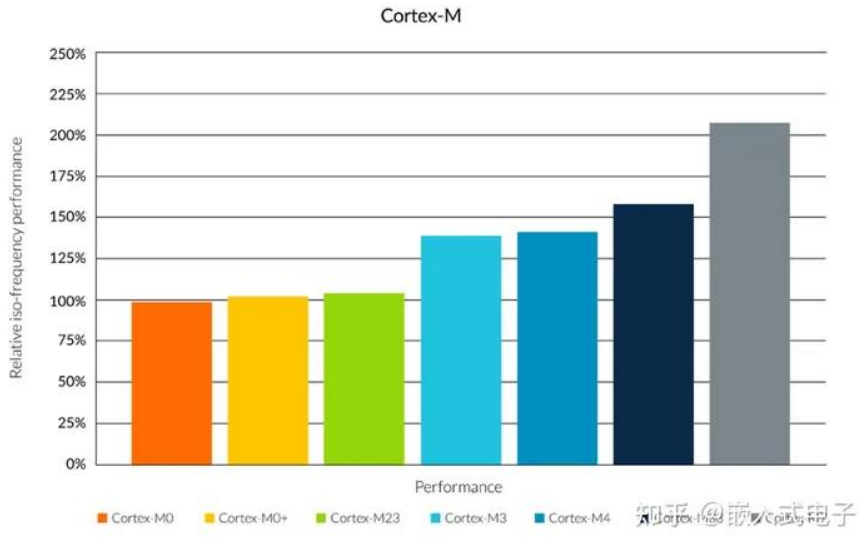


图 1 性能天梯对比

上图为同等主频下，各个内核可以提供的运算性能的大概的对比关系，评判一个处理器处理性能一直是一个难题，有很多的评判标准，上图是 ARM 官网提供的，Cortex-M 系列在 ARM 官网上一直以 Coremark 分数为主要评测标准，可作为参考，真实的性能对比需要具体

应用。

2、Cortex-M 处理器家族

Cortex-M 处理器家族更多的集中在低性能端，但是这些处理器相比于许多微控制器使用的传统处理器性能仍然很强大。例如，Cortex-M4 和 Cortex-M7 处理器应用在许多高性能的微控制器产品中，最大的时钟频率可以达到 400Mhz。

当然，性能不是选择处理器的唯一指标。在许多应用中，低功耗和成本是关键的选择指标。因此，Cortex-M 处理器家族包含各种产品来满足不同的需求：

Cortex-M0	面向低成本，超低功耗的微控制器和深度嵌入应用的非常小的处理器（最小 12K 门电路）
Cortex-M0+	针对小型嵌入式系统的最高能效的处理器，与 Cortex-M0 处理器接近的尺寸大小和编程模式但是具有扩展功能，如单周期 I/O 接口和向量表重定位功能
Cortex-M1	针对 FPGA 设计优化的小处理器，利用 FPGA 上的存储器块实现了紧横合内存 (TCM)。和 Cortex-M0 有相同的指令集
Cortex-M3	针对低功耗微控制器设计的处理器，面积小但是性能强劲，支持可以处理器快速处理复杂任务的丰富指令集。具有硬件除法器和乘加指令(MAC).并且，M3 支持全面的调试和跟踪功能使软件开发者可以快速的开发他们的应用
Cortex-M4	不但具备 Cortex-M3 的所有功能，并且扩展了面向数字信号处理(DSP)的指令集，比如单指今多数据指令(SMID)和更快的单周期 MAC 操作此外，它还有一个可选的支持 IEEE754 浮点标准的单精度浮点运算单元
Cortex-M7	针对高端微控制器和数据处理密集的应用开发的高性能处理器。具备 Cortex-M4 支持的所有指令功能，扩展支持双精度浮点运算，并且具备扩展的存储器功能，例如 Cache 和紧耦合存储器(TCM)
Cortex-M23	面向超低功耗，低成本应用设计的小尺寸处理器，和 Cortex-M0 相似，但是支持各种增强的指令集和系统层面的功能特性。M23 还支持 TrustZone 安全扩展
Cortex-M33	主流的处理器设计，与之前的 Cortex-M3 和 Cortex-M4 处理器类似，但系统设计更灵活，能耗比更高效，性能更高。M33 还支持 TrustZone 安全扩展

3、微处理器为 Cortex-M3 的微控制器 STM32F103ZE

- 核心：STM32F103ZE 采用 ARM Cortex-M3 32 位处理器核心，运行频率高达 72 MHz。
- 存储：具有 512 KB 的闪存存储器和 64 KB 的静态随机存取存储器（SRAM）。
- 外设：包括多个通用定时器、串行通信接口（SPI、I2C）、通用异步收发器（USART）、

模拟-数字转换器（ADC）等。

- 连接：支持多种外部连接接口，如 USB、CAN、以太网等。
- 电源：具有多种电源管理功能，包括低功耗模式和多种电源模式选择。
- 封装：该微控制器的封装形式为 144 引脚 LQFP。

以本课程实验采用的微控制器 **STM32F103ZET6** 为例：

外观方面，STM32F103ZET6 采用的是 176 引脚的 LQFP 封装，在封装上与其他系列的 STM32 微控制器保持了一致性。该封装形式易于焊接和布局，便于将其集成到具体的应用中。同时，该封装也提供了丰富的 I/O 引脚，满足了各种应用的需求。

关于性能，STM32F103ZET6 内置了一颗 ARM Cortex-M3 内核，运行频率可高达 72MHz。该内核具备高性能、低功耗、高集成度等特点，能够满足日常应用中对处理器性能的要求。此外，其还配备了 256KB 的闪存和 64KB 的 SRAM，可以存储大量的应用程序和数据。同时，该微控制器还支持种外设接口，如 USB、SPI、I2C 等，便于与其他设备进行通信和扩展。

在应用场景方面，STM32F103ZET6 适用于广泛的应用领域。首先，在工业自动化领域，该微制器的高性能和丰富的接口能够满足工业控制系统对处理能力和通信能力的要求。其次，在消费电子领域，STM32F103ZET6 可以用于电视机、空调、音响等产品中，通过各种接口实现与其他设备的连接和控制。此外，在智能家居领域，该微控制器可以作为智能家居控制主板，实现对家居设备的管理和控制。另外，在智能交通领域，STM32F103ZET6 可以用于交通信号灯、ETC 系统等设备，实现车辆和设备的智能化控制。总之，STM32F103ZET6 具有广泛的应用领域，并且能够满足不同领域中对处理器性能和通信能力的要求

除了上述介绍的特点外，STM32F103ZET6 还具有其他一些重要的特性。首先，它支持多种电源模式，包括 Halt、Stop、Standby 等模式，以满足不同应用对功耗的要求。其次，该微控制器具备强大的中断系统，可实现多个外设的优先级管理和中断处理。此外，STM32F103ZET6 还具备丰富的时钟系统和时钟源，能够满足不同应用对时钟稳定性和精确性的需求。

综上所述，STM32F103ZET6 是一款性能卓越的 32 位 ARM Cortex-M3 微控制器。其具备高性能、低功耗、丰富的接口等特点，适用于广泛的应用场景。无论是工业控制、消费电子、智能家居还是智能交通等领域，STM32F103ZET6 都能够提供稳定可靠的处理能力和通信能力，满足各种应用的需求。同时，其外观封装简便易用，适合在各种应用中集成使用。

编程工具/软件开发平台

对比与选择

二、编程工具/软件开发平台的对比与选择

1、Keil

1）集成开发环境（IDE）：Keil 提供了一个全面的集成开发环境，包括编辑器、编译器、调试器和仿真器，使开发人员可以在一个统一的界面中进行嵌入式软件开发。

2）支持多种嵌入式平台：Keil 支持多种嵌入式平台，包括 ARM、8051 和 C166 等，可以方便地开发不同类型的嵌入式系统。

3）强大的编译器：Keil 提供了高效的编译器，可以生成高质量的机器代码，优化程序性能和内存占用。

4）丰富的调试功能：Keil 具有强大的调试功能，可以进行单步调试、断点调试、变量监视

5）等，帮助开发人员快速定位和修复问题。

6）支持多种编程语言：Keil 支持多种编程语言，包括 C、C++和汇编语言，可以根据开发需求选择适合的编程语言。

7）生态系统支持：Keil 有一个庞大的生态系统，包括广泛的开发者社区、技术支持和第三方工具支持，可以帮助开发人员更好地应对嵌入式开发挑战。

2、IAR

1）跨平台支持：IAR 可以在多个操作系统上运行，包括 Windows 和 Linux。

2）多种编程语言支持：IAR 支持多种编程语言，包括 C、C++和汇编语言。

3）优化编译器：IAR 提供了高度优化的编译器，可以生成高效的嵌入式代码，以提高系统性能和节省存储空间。

4）调试功能：IAR 集成了强大的调试功能，包括源码级调试、断点设置、变量监视等，可以

5）帮助开发人员快速定位和解决问题。

6）仿真器支持：IAR 支持多种仿真器和调试器，可以与各种硬件平台进行无缝集成。

7）丰富的库支持：IAR 提供了丰富的软件库，包括通信协议、驱动程序和算法库等，可加快开发速度。

8）可扩展性：IAR 支持插件和扩展，可以根据项目的需求进行定制和扩展。

3、STM32CubeIDE

1）基于 Eclipse 平台，提供了强大的代码编辑器、调试器和构建系统。

2）集成了 STM32CubeMX，一个图形化配置工具，用于快速生成 STM32 的初始化代码和配置文件。

3）支持多种编译器和调试器，如 GCC、IAR Embedded Workbench、ST-Link 等。

4）提供了丰富的调试功能，包括实时变量查看、断点设置、跟踪等。

5）支持版本控制系统，如 Git，方便团队协作和代码管理。

6）提供了丰富的示例代码和应用程序模板，加速开发过程。

综合上述列举 3 个编程工具的特点，我们对其功能进行如下评价：

功能评价	Keil	IAR	STM32CubeIDE
中断调试	支持	支持	支持
AAPCS 规范的 C/C++与汇编混合编程	支持	支持	支持
MCU 与外部电路的模拟	支持	不支持	支持
优点	1.老牌 IDE，教程较多，容易找到各种问题的解决办法；2.基于Keil 的软件工程参考资料多	1.提供了丰富的文档资料和技术支持；2.操作界面进行了现代化改进，直观清晰	1.ST 官方软件，对 STM32 开发友好；2.基于 Eclispse 工具链，界面更现代化
缺点	1.界面不现代化；2.编辑功能有待改善	1.工具链闭源，限制了对工具本身的的定制和扩展；2.操作难度相对较高	1.推出时间不长，存在一些问题；2.相对 Keil 更耗电脑 CPU 资源

通过以上对比分析，我们最后选择 Keil 作为主要编程工具。

不同算法选择理由

三、不同算法选择理由

由于本项目的代码部分相对较少，涉及的核心算法只有速度检测算法，因此这里重点探讨速度检测算法的算法选择。

1、帧间差分法

对视频序列进行物体检测和跟踪，获取物体在连续帧中的位置信息。计算相邻帧之间物体位置的差异，可以使用欧氏距离或其他相似度度量方法。将差异除以时间间隔，得到物体的平均速度。

当视频中存在移动物体的时候，相邻帧（或相邻三帧）之间在灰度上会有差别，求取两帧图像灰度差的绝对值，则静止的物体在差值图像上表现出来全是 0，而移动物体特别是移动物体的轮廓处由于存在灰度变化为非 0，当绝对值超过一定阈值时，即可判断为运动目标，从而实现目标的检测功能。 帧间差分法的优点是算法实现简单，程序设计复杂度低；对光线等场景变化不太敏感，能够适应各种动态环境，有着比较强的鲁棒性。缺点是不能提取出对象的完整区域，对象内部有“空洞”，只能提取出边界，边界轮廓比较粗，往往比实际物体要大。对快速运动的物体，容易出现鬼影的现象，甚至会被检测为两个不同的运动物体，对慢速运动的物体，当物体在前后两帧中几乎完全重叠时，则检测不到物体。故该方法一般适用于简单的实时运动检测的情况。

2、DeepSORT 目标跟踪算法

DeepSORT（Deep Learning + SORT）是一种基于深度学习和卡尔曼滤波的目标跟踪算法。它通过结合 YOLOv5 等目标检测器的输出和 SORT（Simple Online and Realtime Tracking）算法的轨迹管理，实现对视频中目标的准确跟踪。使用深度学习模型（如 ResNet）提取目标的特征向量，将其用于目标的身份验证和关联。使用卡尔曼滤波器来预测目标的位置和速度，并通过将检测和预测结果进行关联，提供平滑的目标轨迹。使用匈牙利算法将当前帧的检测结果与上一帧的跟踪结果进行关联，以最大化目标标识的一致性。

它结合了卷积神经网络(CNN)和递归神经网络(RNN)的优势，能够在复杂背景下跟踪多个目标。DeepSORT 算法包含三个部分:目标检测、特征提取和目标跟踪。

首先，DeepSORT 通过目标检测算法(如 YOLO、Faster R-CNN 等)检测出图像中的目标，并使用卷积神经网络从目标中提取特征。接下来，使用递归神经网络(LSTM) 对目标进行跟踪，并根据目标的运动状态更新目标的状态。与传统的目标跟踪算法相比，DeepSORT 具有更高的准确性和效率，并且可以跟踪多个目标。它在许多实际场景中得到了广泛应用，如视频监控、自动驾驶、人机交互等。

3、传感器测速

3.1 车辆传感器

霍尔效应：速度传感器中包含一个霍尔元件，当车轮转动时，通过车轮上的齿轮或者磁铁，产生的磁场变化会影响霍尔元件，从而产生电压信号。根据这些信号的频率或脉冲数，

可以计算出车辆的速度。

磁阻效应：另一种常见的方法是利用磁阻传感器。在车轮上安装有磁性的传动齿轮，当车轮旋转时，磁阻传感器感知到磁场的变化，产生相应的电压信号。通过测量这些信号的变化，可以计算出车辆的速度。

3.2 轨道传感器

传感器固定在轨道上，监测每个轨道区间的状况，当出现变化时判定为列车经过，从而检测列车的位置和速度。

在速度检测方式的选择上，常见的速度检测方式主要包括雷达测速、激光测速、视频测速、车载 GPS 测速、传感器测速等，由于本实验的成本有限，开发板的处理器和内存大小有限，硬件设施受到许多制约，不能采用最初设想的使用外接摄像头模块，基于深度学习从摄像头拍摄的实时视频中进行目标追踪从而准确计算列车速度的方式，因此我们将目光转向了成本更低、操作方式更简单、实现难度更低的传感器测速方式。

在传感器的选择上，出于成本考虑，我们采购的红外传感器模块功能相对较为简单，输出信号只有高低电平之分，没有具体数值和其他有参考价值的信息，因此我们决定将传感器按照区间固定在轨道上，当列车经过时根据传感器持续输出高电平的时间来估计列车的平均速度。

C/C++与汇编语言代码编写规范

四、C/C++与汇编语言代码编写规范

广为采用的编码规范：

- GNU Coding Standards
- Guidelines for the Use of the C Language in Vehicle Based Software
- C++ Coding Guidelines
- SUN Code Conventions for Java

1、C/C++代码编写规范

（1）文件排版方面

①包含头文件

1.1 先系统头文件，后用户头文件。

1.2 系统头文件，稳定的目录结构，应采用包含子路径方式。

1.3 自定义头文件，不稳定目录结构，应在 dsp 中指定包含路径。

1.4 系统头文件应用：#include <xxx.h>

1.5 自定义同文件应用：#include "xxx.h"

1.6 只引用需要的头文件。

②h 和 cpp 文件

2.1 头文件命名为*.h，内联文件命名为*.inl；C++文件命名为*.cpp

2.2 文件名用大小写混合，或者小写混合。例如 DiyMainview.cpp，infoview.cpp。不要用无意义的名称：例如 XImage.cpp；SView.cpp；xlog.cpp；

2.3 头文件除了特殊情况，应使用#ifdef 控制块。

2.4 头文件#endif 应采用行尾注释。

2.5 头文件，首先是包含代码块，其次是宏定义代码块，然后是全局变量，全局常量，类型定义，类定义，内联部分。

2.6 CPP 文件，包含指令，宏定义，全局变量，函数定义。

③文件结构

3.1 文件应包含文件头注释和内容。

3.2 函数体类体之间原则上用 2 个空行，特殊情况下可用一个或者不需要空行。

④空行

4.1 文件头、控制块，#include 部分、宏定义部分、class 部分、全局常量部分、全局变量部分、函数和函数之间，用两个空行。

（2）注释方面

①文件头注释

1.1 作者，文件名称，文件说明，生成日期(可选)

②函数注释

2.1 关键函数必须写上注释，说明函数的用途。

2.2 特别函数参数，需要说明参数的目的，由谁负责释放等等。

2.3 除了特别情况，注释写在代码之前，不要放到代码行之后。

2.4 对每个#else 或#endif 给出行末注释。

2.5 关键代码注释，包括但不限于：赋值，函数调用，表达式，分支等等。

2.6 善未实现完整的代码，或者需要进一步优化的代码，应加上 // TODO ...

2.7 调试的代码，加上注释 // only for DEBUG

2.8 需要引起关注的代码，加上注释 // NOTE ...

2.9 对于较大的代码块结尾，如 for,while,do 等，可加上 // end for|while|do

（3）命名方面

①原则

1.1 同一性：在编写一个子模块或派生类的时候，要遵循其基类或整体模块的命名风格，保持命名风格在整个模块中的同一性。

1.2 标识符组成：标识符采用英文单词或其组合，应当直观且可以拼读，可望文知意，用词应当准确，避免用拼音命名。

1.3 最小化长度 && 最大化信息量原则：在保持一个标识符意思明确的同时，应当尽量缩短其长度。

1.4 避免过于相似：不要出现仅靠大小写区分的相似的标识符，例如"i"与"I"，"function"与"Function"等等。

1.5 避免在不同级别的作用域中重名：程序中不要出现名字完全相同的局部变量和全局变量，尽管两者的作用域不同而不会发生语法错误，但容易使人误解。

1.6 正确命名具有互斥意义的标识符：用正确的反义词组命名具有互斥意义的标识符，如："nMinValue" 和 "nMaxValue"，"GetName()" 和"SetName()" ….

1.7 避免名字中出现数字编号：尽量避免名字中出现数字编号，如 Value1,Value2 等，除非逻辑上的确需要编号。这是为了防止程序员偷懒，不肯为命名动脑筋而导致产生无意义的名字（因为用数字编号最省事）。

（4）代码风格方面

①Tab 和空格

1.1 每一行开始处的缩进只能用 Tab，不能用空格，输入内容之后统一用空格。除了最开始的缩进控制用 Tab，其他部分为了对齐，需要使用空格进行缩进。这样可以避免在不同的编辑器下显示不对齐的情况。

1.2 在代码行的结尾部分不能出现多余的空格。

1.3 不要在"::","->","."前后加空格。

1.4 不要在", ","; "之前加空格。

②类型定义和{

2.1 类，结构，枚举，联合：大括号另起一行

③函数

3.1 函数体的{需要新起一行，在{之前不能有缩进。

3.2 除了特别情况，函数体内不能出现两个空行。

3.3 除了特别情况，函数体内不能宏定义指令。

3.4 在一个函数体内，逻揖上密切相关的语句之间不加空行，其它地方应加空行分隔。

3.5 在头文件定义的 inline 函数，函数之间可以不用空行，推荐用一个空行。

④代码行

4.1 一行代码只做一件事情，如只定义一个变量，或只写一条语句。这样的代码容易阅读，并且方便于写注释。

4.2 多行变量定义，为了追求代码排版美观，可将变量竖向对齐。

4.3 代码行最大长度宜控制在一定个字符以内，能在当前屏幕内全部可见为宜。

⑤宏

5.1 不要用分号结束宏定义。

5.2 函数宏的每个参数都要括起来。

5.3 不带参数的宏函数也要定义成函数形式。

（5）类型

定义指针和引用时*和&紧跟类型。

尽量避免使用浮点数，除非必须。

用 typedef 简化程序中的复杂语法。

避免定义无名称的类型。例如：typedef enum { EIdle, EActive } TState;

少用 union，如果一定要用，则采用简单数据类型成员。

用 enum 取代(一组相关的)常量。

不要使用魔鬼数字。

尽量用引用取代指针。

定义变量完成后立即初始化，勿等到使用时才进行。

如果有更优雅的解决方案，不要使用强制类型转换。

（6）性能

使用前向声明代替#include 指令。Class M;

尽量用++i 代替 i++。即用前缀代替后缀运算。

尽量在 for 循环之前，先写计算估值表达式。

尽量避免在循环体内部定义对象。

避免对象拷贝，尤其是代价很高的对象拷贝。

避免生成临时对象，尤其是大的临时对象。

注意大尺寸对象数组。

80-20 原则。

（7）兼容性

遵守 ANSI C 和 ISO C++国际标准。

确保类型转换不会丢失信息。

注意双字节字符的兼容性。

注意运算溢出问题。

不要假设类型的存储尺寸。

不要假设表达式的运算顺序。

不要假设函数参数的计算顺序。

不要假设不同源文件中静态或全局变量的初始化顺序。

不要依赖编译器基于实现、未明确或未定义的功能。

将所有#include 的文件名视为大小写敏感。

避免使用全局变量、静态变量、函数静态变量、类静态变量。在使用静态库，动态库，多线程环境时，会导致兼容性问题。

不要重新实现标准库函数,如 STL 已经存在的。

2、汇编语言代码编写规范

（1）文件结构

每个汇编程序通常分为两个文件。一个文件用于保存程序的声明（declaration），称为头文件。另一个文件用于保存程序的实现（implementation），称为定义（definition）文件。头文件以“.h”为后缀，定义文件以“.asm”为后缀,宏定义文件以“.mac”为后缀。

1.1 头文件

用 .include di.h 头文件（编译器将从用户的工作目录开始搜索）。

每个标号定义时,要加空格时都用”Tab”键,定义符和定义值要整齐,每个定义符后都在同一列加上”;”(分号),分号后加上每个标号的注释。

1.2 函数

原则上函数内不允许对绝对地址进行操作。

函数内不要分配大的数组，占用堆栈空间。

函数内尽量不要使用数字和字符等常量，而要用标识符常量，便于以后的修改。

每个函数体不得超过 150 行（不含注释）

（2）程序的版式

2.1 对齐

要求编辑中对齐使用 Tab,而编辑软件中将 Tab 设为 8 个字符位置,且跳格不用空格代替。

每行程序宽度为 6 个 Tab 宽,即 8*6=48 个字符,第 49 列为分号,分号后写注释。

标号从行首(即第 1 列)开始输入,标号后不直接跟汇编指令,换行后加一个 Tab 键后再输入汇编助记符,输完助记符后再加一个 Tab 键才输操作数。

2.2 注释

使用中文做注释。

汇编语言是低级语言,原则上要求每行都加上注释。最少注释率不得低于 50%。

边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

注释应当准确、易懂，防止注释有二义性。错误的注释不但无益反而有害。

尽量避免在注释中使用缩写，特别是不常用缩写。

注释的位置应与被描述的汇编指令相邻，汇编语言一般把注释放在指令的右方。

（3）命名规则

标识符应当直观且可以拼读，可望文知意，不必进行“解码”。

标识符最好采用英文单词或其组合,便于记忆和阅读。切忌使用汉语拼音来命名。

程序中的英文单词一般不会太复杂，用词应当准确。例如不要把 CurrentValue 写成 NowValue。

单词连写时,用第一个字母大写来区分。如: CurrentValue。

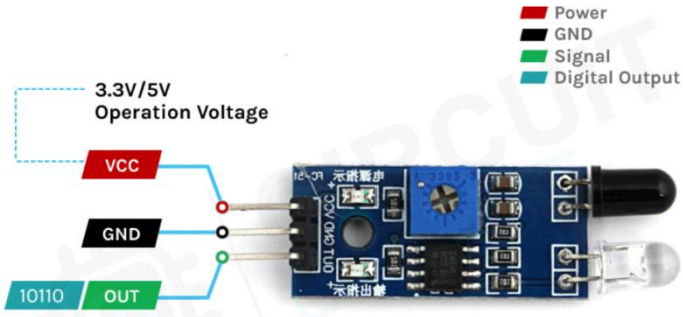
算法原理

五、算法原理

基于红外传感器的 STM32 速度检测算法：

1、红外传感器模块工作原理

红外接近传感器或红外传感器它发射红外光以感知周围环境，并可用于检测物体的运动。由于这是一个无源传感器，它只能测量红外辐射。



VCC 连接到 开发板上的 5V 引脚的红外传感器的电源引脚。

OUT 引脚为 5V TTL 逻辑输出。低表示未检测到运动;高表示检测到运动。

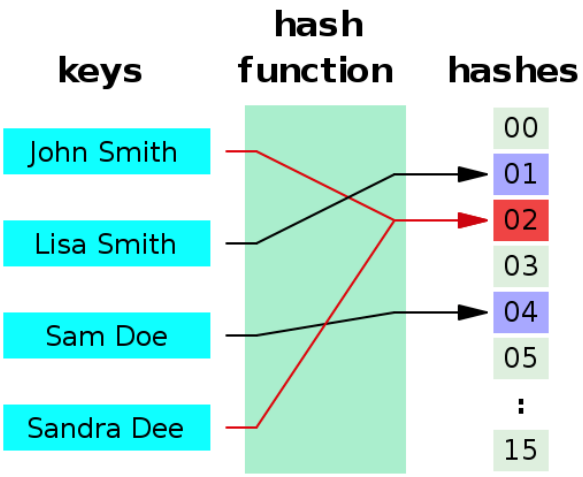
GND 连接到开发板的接地引脚。

它由两个主要组件组成：第一个是红外发射器部分，第二个是红外接收器部分。在发射器部分，使用红外 LED，在接收器部分，光电二极管用于接收红外信号，经过信号处理和调理，将获得输出。红外传感器的工作原理是向红外发光二极管施加电压，红外发光二极管发出红外光。该光在空气中传播并撞击物体，然后光电二极管传感器接收。如果物体近，反射光会更强，如果物体离得远，反射光会更弱当传感器触发时，它通过输出引脚发送低电平，任何类型的微控制器都可以检测到该信号以执行特定任务。该模块内置了两个板载 LED，其中一个在电源可用时亮起，另一个在电路被触发时打开。该传感器功耗低、成本低、坚固耐用，并且具有宽感应范围，可以调整灵敏度。红外传感器模块可由 3.3V 和 5V 电源供电，传感器可测量的距离为 2-10 厘米。

2、实时速度计算

采用公式速度=路程/时间进行计算，将红外传感器垂直于轨道放置，当红外传感器检测到信号时启动计时器开始计时，当未检测到信号时停止计时，记录这段时间的长度，在这段时间内列车走过的路程即为列车本身的长度，路程÷时间即可得到这段区间内列车的平均速度。

3、实时速度哈希演算



哈希算法(Hash Function)，将任意长度的二进制值串映射为固定长度的二进制值串，这个映射的规则就是哈希算法，而通过原始数据映射之后得到的二进制值串就是哈希值。构成哈希算法的条件：从哈希值不能反向推导出原始数据（所以哈希算法也叫单向哈希算法）；对输入数据非常敏感，哪怕原始数据只修改了一个 Bit，最后得到的哈希值也大不相同；散列冲突的概率要很小，对于不同的原始数据，哈希值相同的概率非常小；哈希算法的执行效率要尽量高效，针对较长的文本，也能快速地计算出哈希值。在实验中该算法主要用于速度预测与检验功能。

外部电路实现

六、外部电路实现

1. 总览

小车：

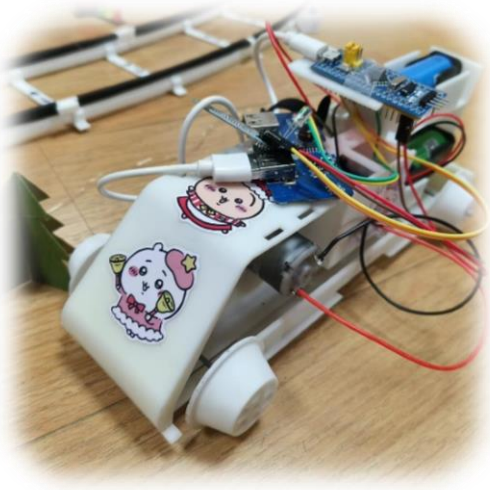
1) 列车建模及加工
2) STM32F103C8T6 主控制器（最小系统板）
3) R280 电机
4) HC-05 蓝牙模块
5) 18650 电池 3 节

主控制器（STM32F103ZET6）：

1) 中控屏
2) 信号灯模块 5 个
3) 红外传感器模块 5 个
4) HC-05 蓝牙模块

2. 机械设计部分

主要结构：



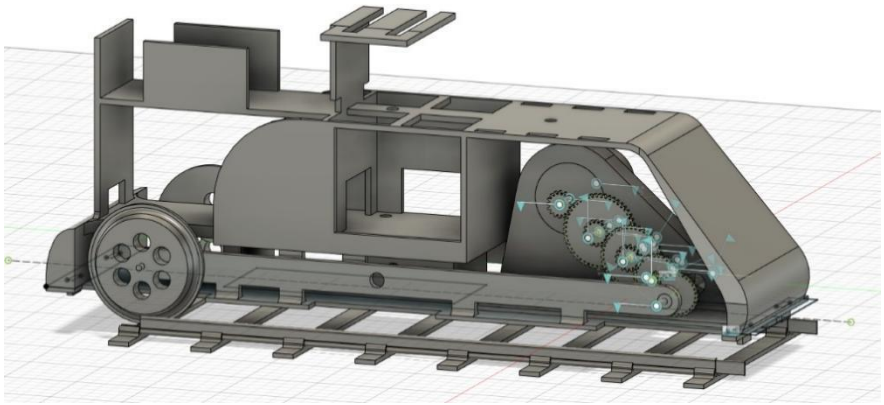
1. 设计原理

总体来讲，小火车运动的能量来源是位于后舱底部的两节 9250mWh 容量的 18650 锂电池串联得到的 7.4V-8.4V 电压。电池给 L298N 电机驱动模块供电（要求 VCC 大于 5V），L298N 把从单片机得到的 PWM 信号电流放大后输出给电机，使电机达到预定转速和扭矩。而小火车的轮子采用圆台结构，圆台侧面与粘贴了胶条的轨道表面接触，将前车轮轴的旋转运动转化为小火车的线性运动。需要注意的是，由于圆台侧面内外的周长不同，小火车可以根据转弯半径自动调节接触点位置，当接触点位于外侧时，周长小，转化成的线速度小；当接触点位于内侧时，周长大，转化成的线速度大。小火车可以基于此原理实现差速转弯。

2. 主体结构

i. 车体架构

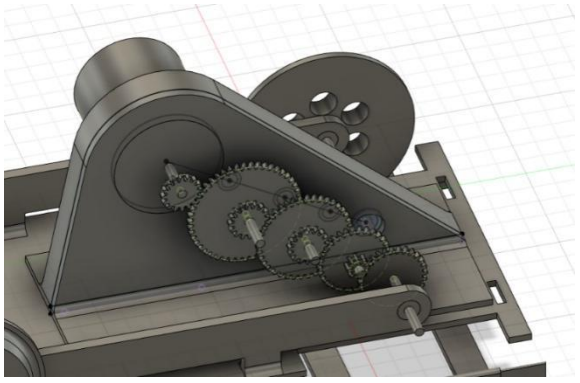
车体整体由底板、电机架、减速器（由 3 级双齿轮和轴组成）、前轮轴（驱动轴）、后轮轴（从动轴）、车轮和各种连接件（如轴承、垫片等）组成。



其中，车架设计了后舱两个，用于存放 18650 锂电池。后舱下部（48*80）存放 2 节 18650，用于电机驱动模块供电；上部存放 1 节，用于给单片机供电。车架中部下层存放 L298N 电机驱动模块（45*45），第二层存放电源管理模块（3.7-5V USB），第三层存放 STM32F103C8T6 最小系统板单片机。第二层前部可以用来固定蓝牙模块和未来的拓展模块。

ii. 电动机安装位置

电机安装在小火车前部底层，通过电机架固定在底板上，电机架上同时还留有减速器中各轴的轴承孔（5）和轴孔（2.5mm 孔 2mm 轴）。轴中心连线与水平线成 45° 角。



iii. 传动系统

本设计使用了五级级联齿轮减速器，将驱动力级联或串联，通过多组齿轮间的啮合（一个大齿轮驱动另一小齿轮）来实现减速，这样组合可以得到较大的减速比。

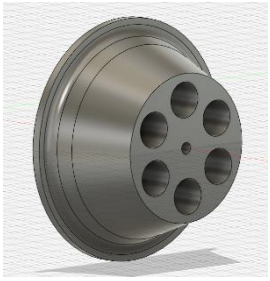
减速器将 R280 电机提供的 8000+ r/min 的高转速减速到合理的范围内。根据减速器实际的采用齿轮齿数：

轴#1	轴#2	轴#3	轴#4	轴#5
16	48-20	36-16	30-8	28

计算得传动比=（48/16）*（36/20）*（30/16）*（28/8）≈35

因此，在圆台的截面圆直径范围为 24-34mm 范围下，可以计算到当电机满功率运行时，内外最大线速度分别为 287mm/s 和 407mm/s。

iv. 轮子和轴等连接件



轮子：

轮子侧面由轮缘和正常的圆台侧面组成，其中车轮最外侧直径 24.721mm，轮缘和圆台交界处直径 34.187mm。轮缘的作用是将小火车底部固定在轨道以内，防止火车脱轨。

轴：

全车使用的所有轴都采用 2mm 直径的钢轴，但实际使用发现其很容易形变，以后的机械设计可以考虑更粗的轴。

齿轮：

齿轮模数采用的是 0.5，齿轮的一些关键参数如下：

模数：直径/齿数；只有模数相等的齿轮才能流畅与运行。

传动比：齿数之比（被动/主动）

压力角：常见的有 20°

中心距：M*（n1+n2）/2

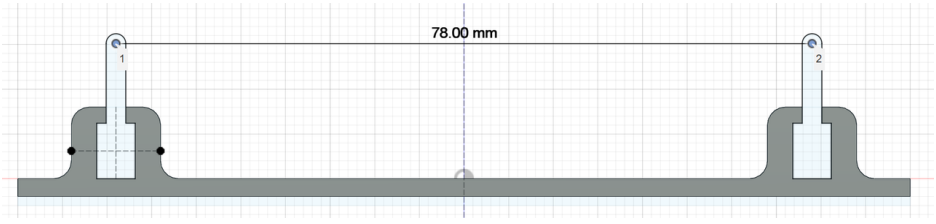
轴承：

采用 682zz 轴承，尺寸为 2*5*2.5；内径为 2mm，外径 5mm，厚度 2.5mm。

v. 轨道和木枕

1. 轨道间距设计

实测火车轮轴内测间距 66~69mm（轮缘内侧间距），外侧间距 80~84mm。最大轨距不超过 80mm 可保证火车不掉。理想的接触点间距 75mm 左右。



设计轨道中心点间距 78mm。由于轨道表面会贴 1mm 厚的胶条（增大摩擦力），实际接触点间距大约在 74~78mm。

2. 加工方式

木枕、轨道的加工采用零件化的方式，这样转移、拼接会方便很多。此外，还可以通过调整一个个的木枕和轨道的角度自定义地改变轨道的弯曲角度和方向，对于后期改变路线很有帮助。

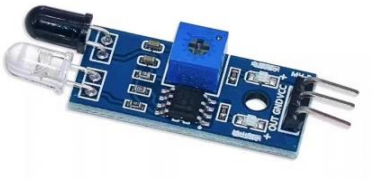
3. 连接方式

木枕与轨道的连接几乎是不可拆卸连接，而轨道与轨道的连接（卡扣式）则是可拆卸连接。对于拼接方式，由于是榫卯连接，不同连接部位的可拆卸性不同，最后采用了至多左右 2 轨道+若干木枕的方式存储和拼接零件，这样拼接、拆卸方便，工作量不大，且可转移性也比较高。

3. 硬件电路部分

1. 主控板（STM32F103ZET6）

模块名	作用	个数	协议	引脚接口个数
红外传感模块	感知列车位置	5	I2C/串口	1
信号灯模块	指示状态及自定义控制	5	串口	3
蓝牙模块	对小火车行为下命令	1	串口	2



红外传感模块

信号灯模块

HC-05 蓝牙模块

引脚对应表：

模块名	**类型**	**引脚**	**模块名**	**类型**	**引脚**
信号灯 1 - Green	OUT	PF6	传感器 1	IN	PC3
信号灯 1 - Yellow	OUT	PC13	传感器 2	IN	PC1
信号灯 1 - Red	OUT	PE0	传感器 3	IN	PA5

信号灯 2 - Green	OUT	PC4	传感器 4	IN	PA6
信号灯 2 - Yellow	OUT	PF12	传感器 5	IN	PC2
信号灯 2 - Red	OUT	PF11	信号灯 4 - Yellow	OUT	PC0
信号灯 3 - Green	OUT	PD6	信号灯 4 - Red	OUT	PF13
信号灯 3 - Yellow	OUT	PE1	信号灯 5 - Green	OUT	PF8
信号灯 3 - Red	OUT	PG15	信号灯 5 - Yellow	OUT	PF7
信号灯 4 - Green	OUT	PA7	信号灯 5 - Red	OUT	PF9

蓝牙模块接口为 PA9（TX），PA10（RX）。

2. 小车板（STM32F103C8T6）

模块名	作用	个数	引脚接口个数	类型
L298N 模块	驱动电机	1	1	OUT（PA6）
BOOST 模块	将锂电池的 3.7V 转化为 5V	1	0	/
蓝牙模块	接受中控板发出的信号	2	2	/

设计与调试技巧

七、设计技巧

1、列车

1）功能分析：在设计小车结构之前，明确小车的功能需求，本项目中小车采用类火车结构，需要在列车轨道上运行，并且需要与最小系统板连接，需要提前预留出对应空间。

2）供电模块：小车的供电模块包括电池、电机、驱动电路等。在设计时需要考虑电源的稳定性、电机的功率与效率，以及驱动电路的控制方式。

3）结构材料选择：根据设计需求和制造成本，选择合适的结构材料，如金属、塑料、碳纤维等。由于我们采用 3D 打印的方式进行制造，材料选择空间较小，承重为第一考虑因素。

2、轨道

1）系统集成：轨道结构设计需要与之前的列车设计相匹配，确保轨道与车辆的匹配度，从而提高列车的平稳性和安全性，降低脱轨的风险；

2）结构设计：结构设计需要考虑承载能力、几何形状、连接方式等。主要包括轨道的横断面设计、轨道连接方式以及轨道的支撑结构。

3）弯道设计：需要考虑列车的最大行驶速度、列车的车长等因素，弯道弧度过大会导致列车极易脱轨，弧度过小会导致轨道过长，占据空间过大。

3、中控屏

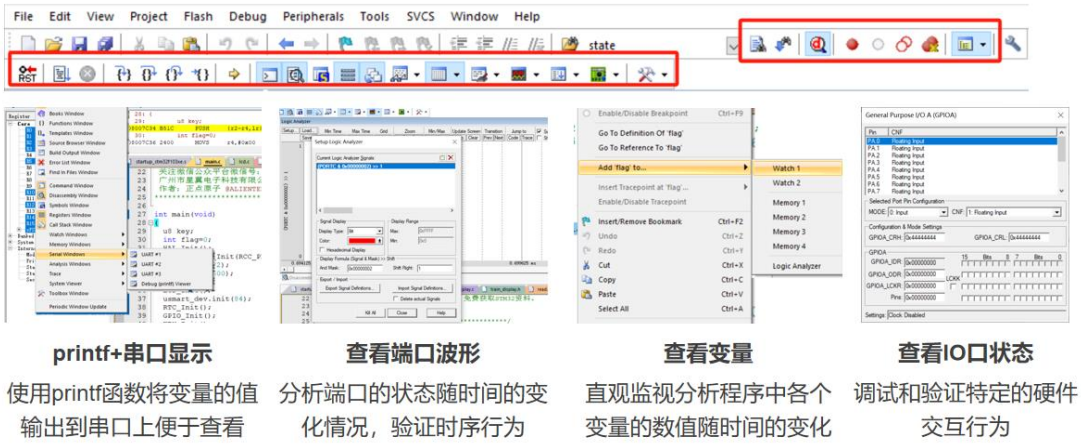
1）时序设计：TFTLCD 屏幕需要精确的时序控制。在设计时需要确保微控制器的时钟频率和外设时序要求相匹配。时序的不匹配可能导致屏幕显示异常或者不稳定。

2）显示控制：需要编写适配于具体 TFTLCD 屏幕的显示驱动程序。负责将显示数据转换为屏幕可识别的格式，并通过合适的接口（如并行接口或 SPI 接口）发送数据到屏幕。除此之外还需要控制屏幕的刷新率以确保显示内容的稳定性和流畅性。这需要结合显示驱动程序和微控制器的性能来进行优化。

3）显示优化：针对 STM32F103ZET6 的资源限制，需要优化显示数据的处理和存储，以最大程度地节约微控制器的资源。如在数据存储上需要选择适当的数据结构来组织和存储显示数据以确保高效的访问和处理。对于需要频繁更新显示的实时数据需要设计高效的数据处理算法以确保及时更新并减少对微控制器资源的占用。由于屏幕空间有限，在设计过程中还需要精简显示内容以确保信息的清晰度和可读性。

八、调试技巧

点击Debug调试按钮，进入调试状态，就会出现如下图的**Debug Toolbar调试工具栏。**



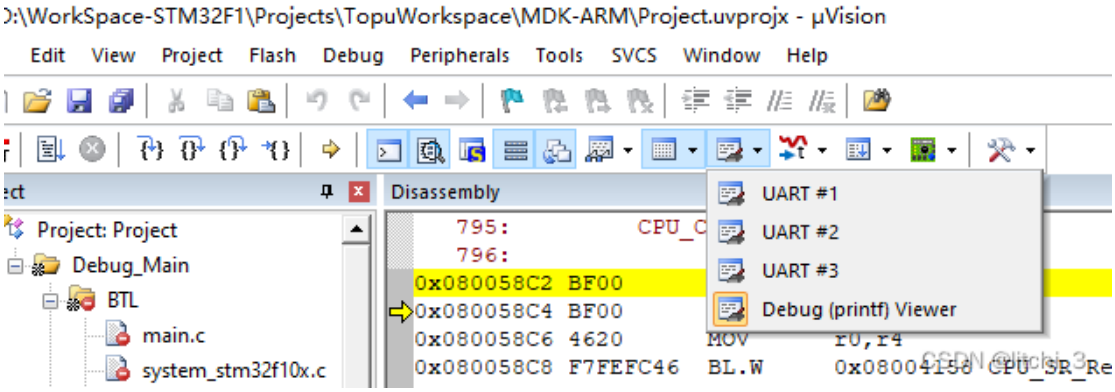
printf+串口显示	查看端口波形	查看变量	查看IO口状态
使用printf函数将变量的值输出到串口上便于查看	分析端口的状态随时间的变化情况，验证时序行为	直观监视分析程序中各个变量的数值随时间的变化	调试和验证特定的硬件交互行为

➢ Printf+串口显示

在使用 Keil 仿真时想要查看程序的打印信息，主要有两种方法。方法一：使用 keil 自带的 Debug（printf）viewer 窗口打印；方法二：使用串口打印。

一、方法一：使用 keil 自带的 Debug（printf）viewer 窗口打印

Serial windows 窗口——选择 Debug（printf）Viewer 窗口



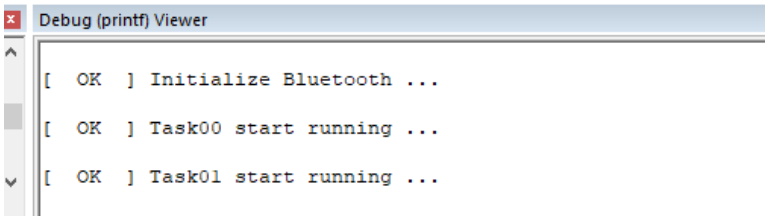
将 printf 函数重定向到 ITM 口，在程序中添加相关代码

```
#define ITM_Port8(n)    (*((volatile unsigned char *)(0xE0000000+4*n)))
#define ITM_Port16(n)   (*((volatile unsigned short*)(0xE0000000+4*n)))
#define ITM_Port32(n)   (*((volatile unsigned long *)(0xE0000000+4*n)))

#define DEMCR           (*((volatile unsigned long *)(0xE000EDFC)))
#define TRCENA          0x01000000
struct __FILE { int handle; /* Add whatever is needed */ };
FILE __stdout;
FILE __stdin;

int fputc( int ch, FILE *f ) {
    if (DEMCR & TRCENA)
    {
```

```
        while (ITM_Port32(0) == 0);
        ITM_Port8(0) = ch;
    }
    return(ch);
}
```

代码添加完成后打开魔术棒的 Debug 窗口，调试器选择后，打开 setting 窗口，选择 Trace，配置 ITM Stimulus Ports 后，使能相应 port。

配置好后进行仿真，查看 Debug(printf) Viewer 窗口，可以看到调试信息。



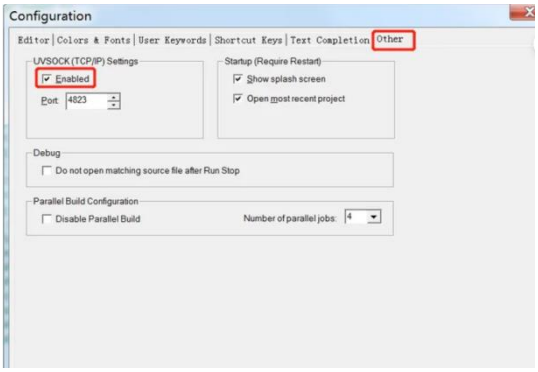
二、方法二：重定向到串口打印

使用 fputc 函数重定向到串口打印

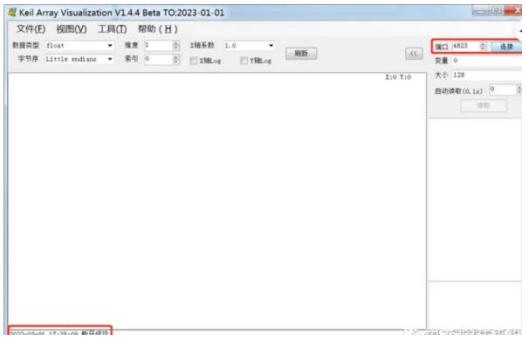
➢ 查看端口波形

使用配合 Keil 使用的调试助手软件 Keil Array Visualization 来查看波形

在 Keil 的 Edit->Configuration 菜单下，使能下图中的选项。



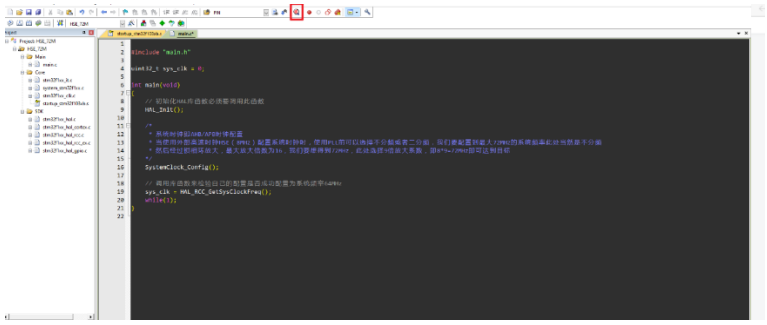
连接仿真器，烧写程序，进入调试状态。打开软件，输入端口号，点击连接。



连接成功后，输入需要查看的变量名或地址（16 进制地址）；输入需要查看的变量大小（该大小以字节为单位）；选择数据类型；选择字节序（STM32 为小端格式）。然后点击读取即可查看波形。也可以输入自动读取的间隔，点击读取，开始自动读取。

➢ 查看变量

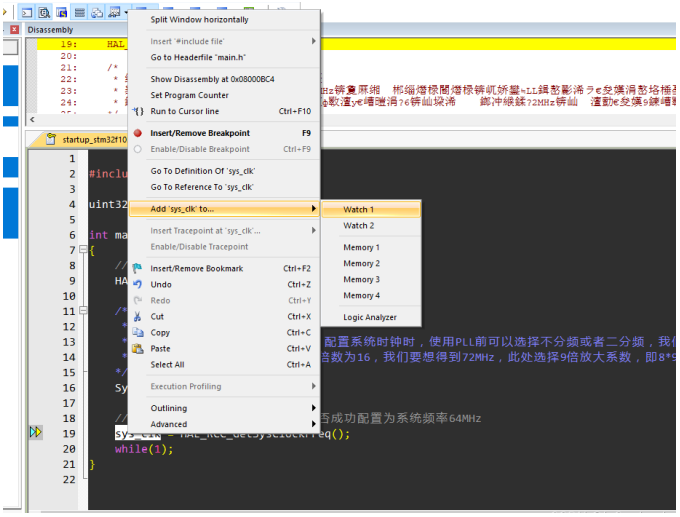
1. 打开工程进入调试界面



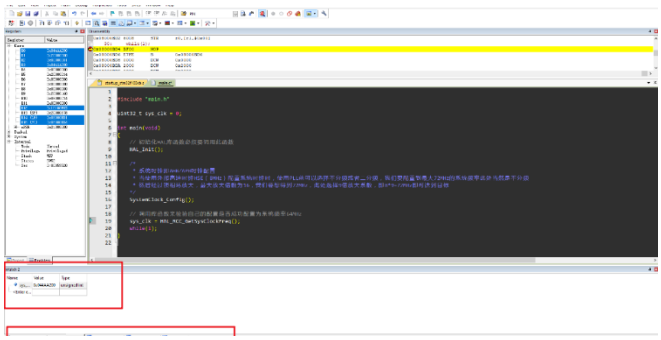
进入调试界面需要连接芯片

2. 选择需要查看的变量

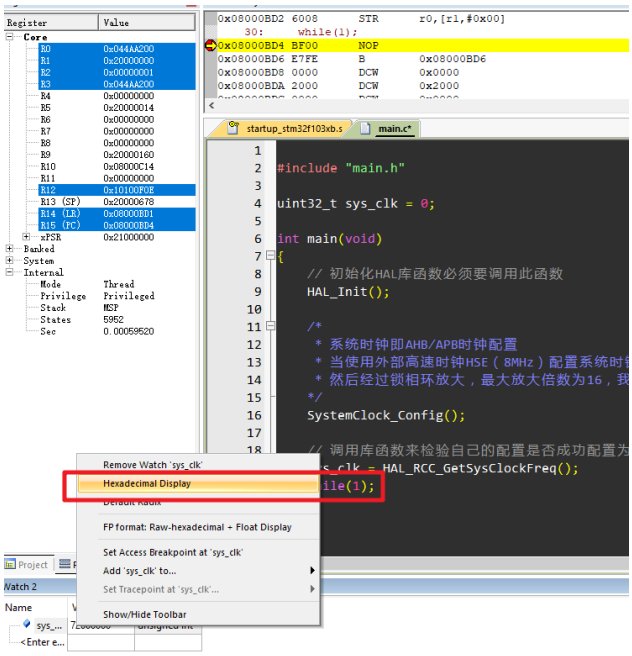
右击选中“Add 'sys_clk' to... ”的变量,watch1，watch 都可



4、找到窗口，窗口中的值为当前变量值

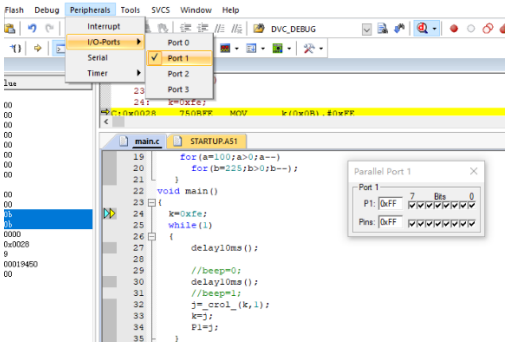


5、右击 去勾√Hex 显示（16 禁止） 以十进制显示



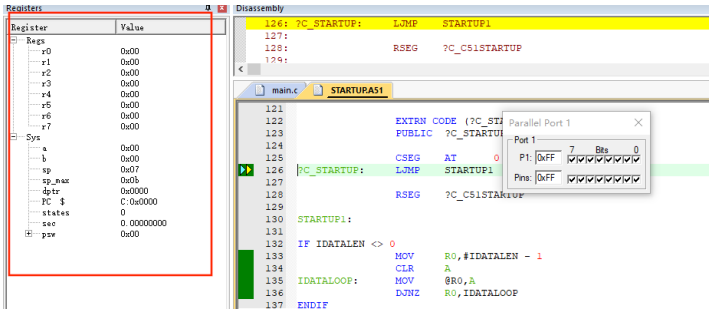
➢ 查看 IO 口状态

实现在单步执行代码时，查看硬件 IO 口电平变化和变量值的变化。先将硬件 IO 口模拟器打开。



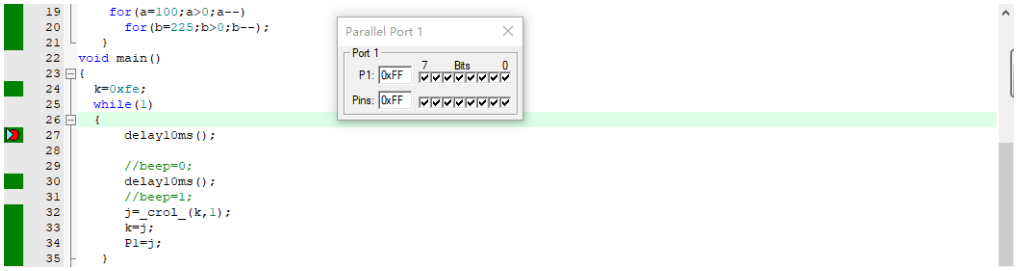
显示的是软件模拟出的单片机 P1 口 8 位口线的状态，单片机上电后 I/O 口全为 1,即十六进制的 0xFF。

4.sec 查看单句运行时间!

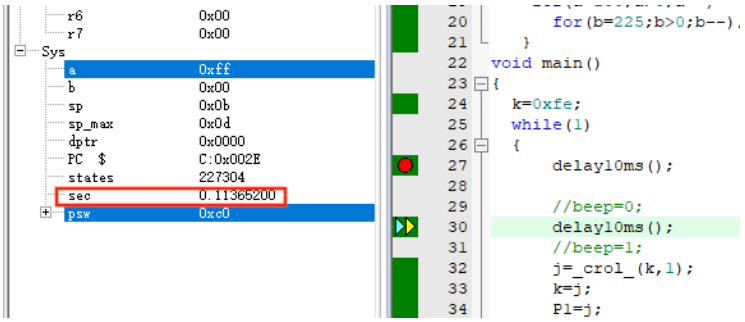


左侧的寄存器窗口中可以到一些寄存器名称及它们的值，本小节的核心部分" sec "，它后面显示的数据就是程序代码执行所用的时间，单位是秒。通过添加断点，单步执行就可以得到我们想要的时间数据。

5.添加断点



我们在延时函数处，添加断点，然后运行，会看到延时时间。



我们选择跳过函数，会看到时间。

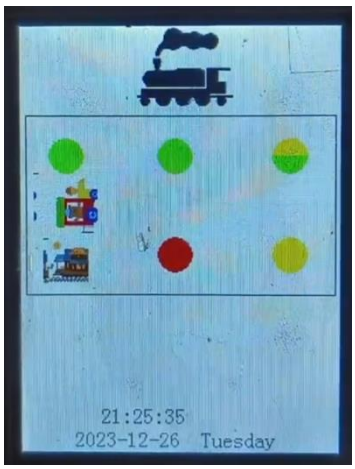
程序实现结果

九、程序实现结果

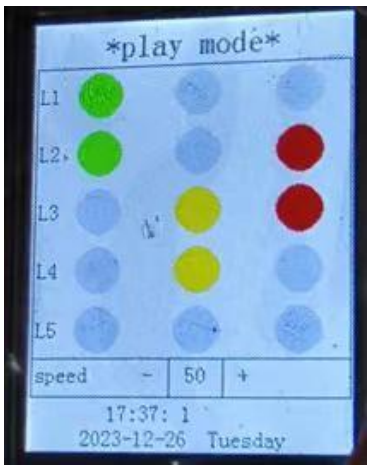
1、中控屏显示效果

本项目在中控板的 TFTLCD 屏幕上共进行了三个界面的绘制，分别是封面、模式一界面、模式二界面，同时使用按键输入功能，从而在三个界面之间实现来回切换，按下一次 WK_UP 按键，即可实现一次切换，切换顺序为封面—模式一—封面—模式二—封面。

封面展示



模式一界面

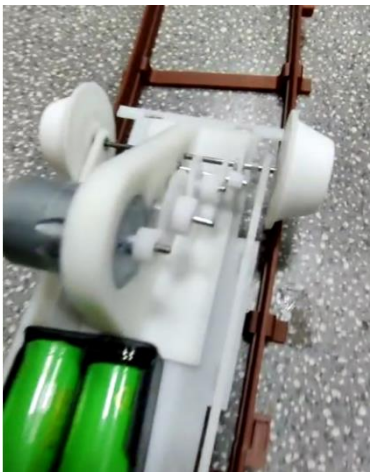


模式二界面

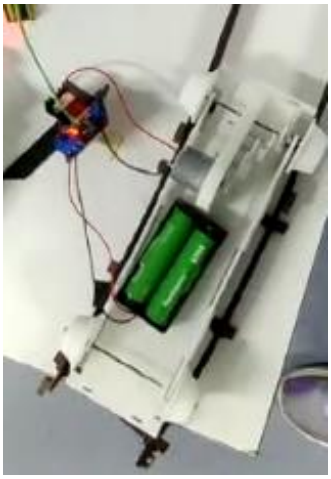
2、列车运行效果

通过程序进行 PWM 波输出控制电机转动从而控制小车的运行状态，可通过调整其占空比从而改变列车的运行速度，实现模拟真实的列车运行状态。

地面运行



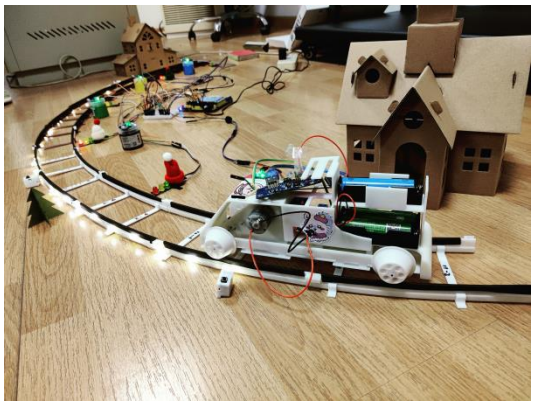
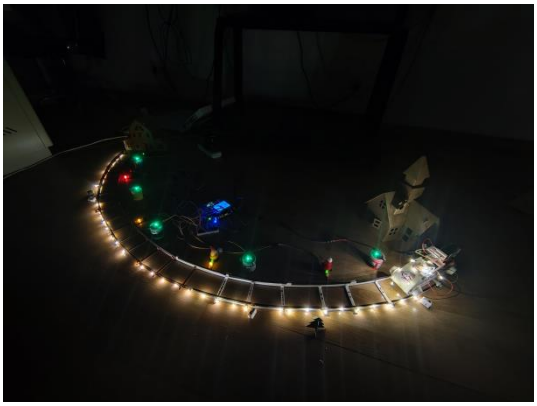
轨道运行（无橡胶条）



轨道运行（有橡胶条）

3、组合联调效果

当中控屏部分和列车轨道部分各自能够正常运行后，我们搭建整体模型联调，由于列车具有一定长度，轨道弧度不能过大否则极易发生脱轨现象，调整合适的轨道弧度，设置合适的信号灯与红外传感模块的位置和间距，根据项目主题搭配相宜的景观装饰后整个列车运行模型的效果如下所示（更完整的运行效果已在视频中呈现）。

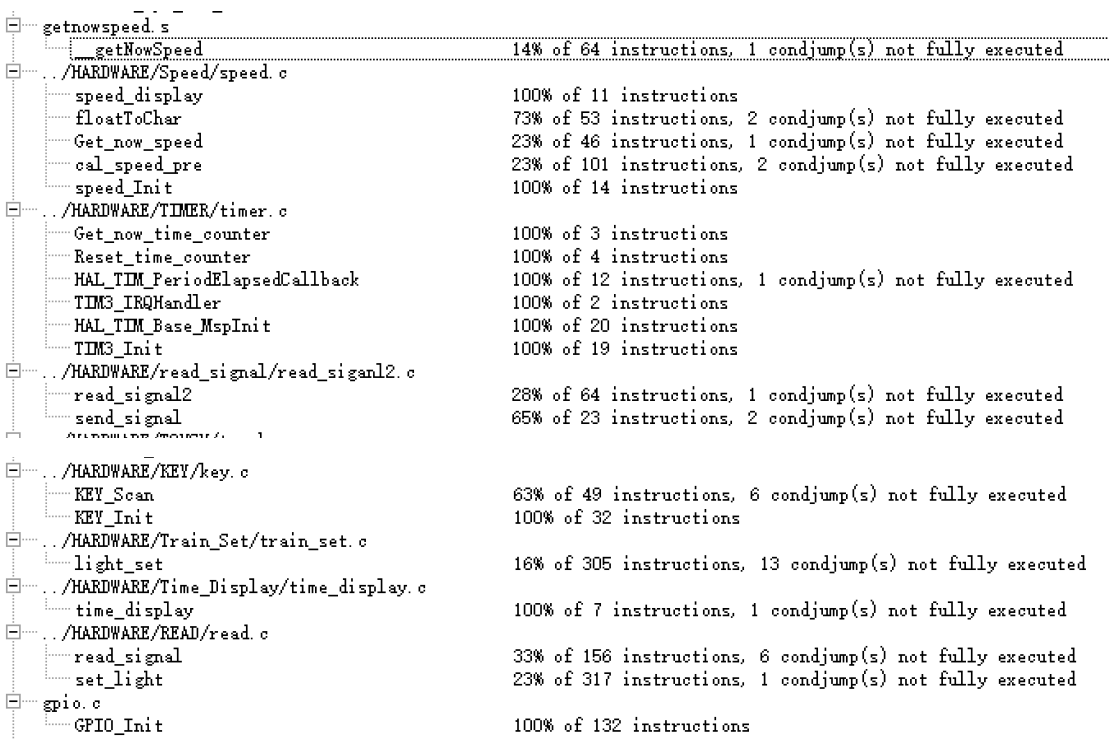
初次搭建效果（此时轨道尚未成环）


二次搭建效果（此时轨道已成环）

程序性能

十、程序性能

1. 代码覆盖率：



由于 STlink 是一款比较基础的调试器，不支持 ARM 的 ETM，因此不能一边运行程序，一边得到真实的代码覆盖率。ETM(Embedded Trace Macrocell)，是一种硬件调试工具，是 ARM 核心中的一个重要部分。它主要被用来非入侵地跟踪 ARM 核心的执行。通过 ETM，可以得到详细的程序执行路径，包括分支、程序计数器的值、指令的执行状态等信息。

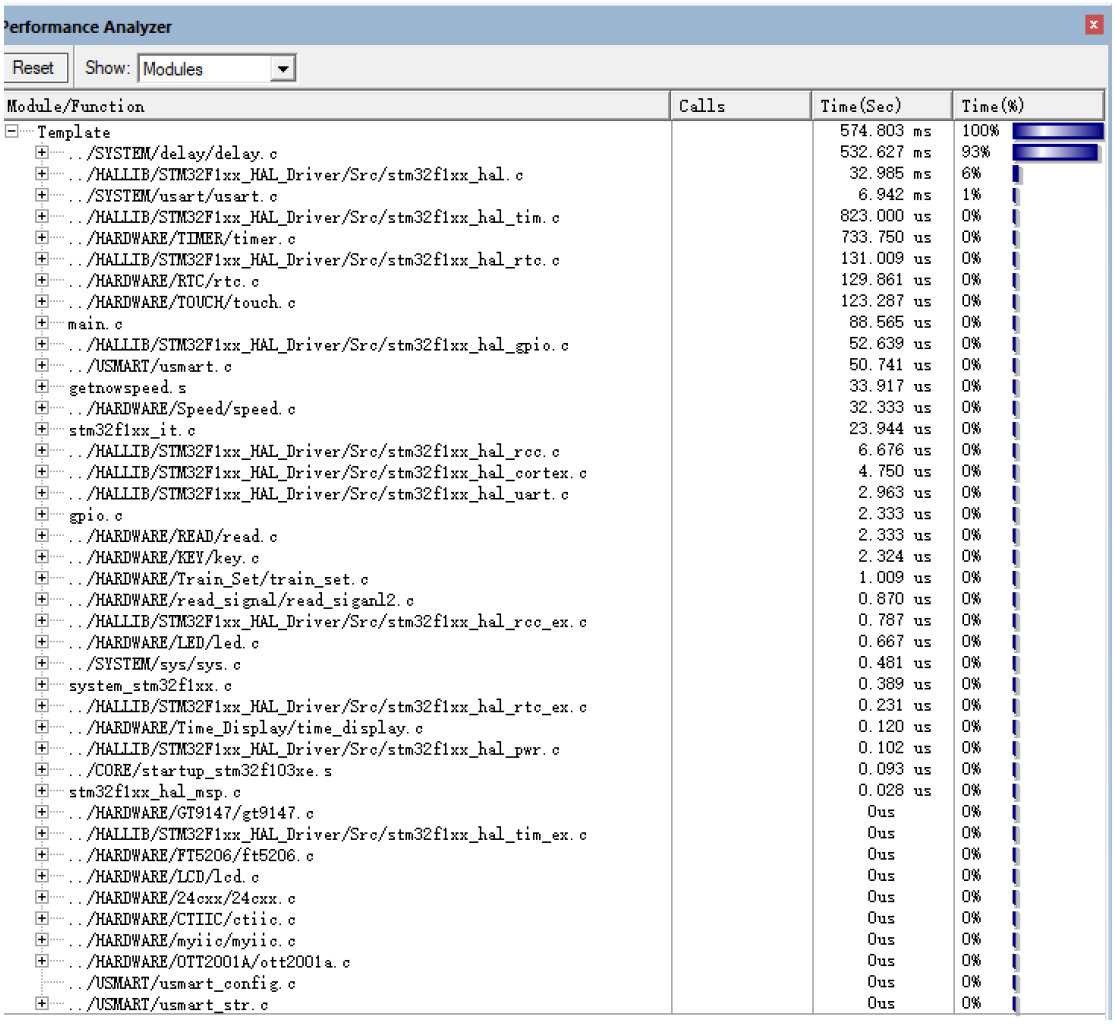
在这种情况下，我们只能通过攥写另外的测试程序来得到一些粗略的代码覆盖率。但是这样的话会带来两个比较严重的问题，一是由于 LCD 屏幕用到了外部内存，simulator 是不能访问这些内存地址的，因此需要在测试代码覆盖率时将所有和 LCD 屏幕有关的所有调用函数全部注释掉；而这也将影响得到的代码覆盖率（所有与 LCD 有关的函数都无法执行）。二是在很多情况下，代码内部是存在各种条件分支的，由于测试例不全等情况，总会有一些条件分支无法被遍历到，因此导致了测得的代码覆盖率远远小于真实的代码覆盖率。例如汇编代码 getnowspeed.s 就是这样的情况。在图中的*** not fully executed***都是类似的情况。

因此，我认为最好的解决方案是更换更好的连接器，例如 J-Link 等能够支持 ETM 的连接器。虽然 STlink 在很多情况下能满足我们的需求了，但既然课程要求了我们测试代码覆盖率，我认为课程老师也应该提供这样的硬件设备。

2. 时间和空间分析

本项目没有在时间和空间上较复杂的算法，因此时间复杂度、空间复杂度都可以忽略不计。

时间上：



Module/Function	Calls	Time(Sec)	Time(%)
Template		574.803 ms	100%
../SYSTEM/delay/delay.c		532.627 ms	93%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal.c		32.985 ms	6%
../SYSTEM/usart/usart.c		6.942 ms	1%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_tim.c		823.000 us	0%
../HARDWARE/TIMER/timer.c		733.750 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_rtc.c		131.009 us	0%
../HARDWARE/RTC/rtc.c		129.861 us	0%
../HARDWARE/TOUCH/touch.c		123.287 us	0%
main.c		88.565 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_gpio.c		52.639 us	0%
../USMART/usmart.c		50.741 us	0%
getnowspeed.s		33.917 us	0%
../HARDWARE/Speed/speed.c		32.333 us	0%
stm32f1xx_it.c		23.944 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_rcc.c		6.676 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_cortex.c		4.750 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_uart.c		2.963 us	0%
gpio.c		2.333 us	0%
../HARDWARE/READ/read.c		2.333 us	0%
../HARDWARE/KEY/key.c		2.324 us	0%
../HARDWARE/Train_Set/train_set.c		1.009 us	0%
../HARDWARE/read_signal/read_siganl2.c		0.870 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_rcc_ex.c		0.787 us	0%
../HARDWARE/LED/led.c		0.667 us	0%
../SYSTEM/sys/sys.c		0.481 us	0%
system_stm32f1xx.c		0.389 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_rtc_ex.c		0.231 us	0%
../HARDWARE/Time_Display/time_display.c		0.120 us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_pwr.c		0.102 us	0%
../CORE/startup_stm32f103xe.s		0.093 us	0%
stm32f1xx_hal_msp.c		0.028 us	0%
../HARDWARE/GT9147/gt9147.c		0us	0%
../HALLIB/STM32F1xx_HAL_Driver/Src/stm32f1xx_hal_tim_ex.c		0us	0%
../HARDWARE/FT5206/ft5206.c		0us	0%
../HARDWARE/LCD/lcd.c		0us	0%
../HARDWARE/24cxx/24cxx.c		0us	0%
../HARDWARE/CTIIC/ctiic.c		0us	0%
../HARDWARE/myiic/myiic.c		0us	0%
../HARDWARE/OTT2001A/ott2001a.c		0us	0%
../USMART/usmart_config.c		0us	0%
../USMART/usmart_str.c		0us	0%

可以从这张图中看到用来处理按键的 delay 功能反而占了程序运行的绝大多数时间，没有任何其他功能函数能够与之比较。哪怕是加入了哈希处理速度数据之后，接近 O(1)的时间复杂度使之甚至不能出现在这张图中。

对于汇编部分的代码分析，我们发现将原有代码改为自行攥写的汇编后反而执行时间变多了。C 语言代码执行时间 30.556us，汇编 33.917us。优化百分比**-12.1%**。

getnowspeed.s		33.917 us
__getNowSpeed	99	33.917 us
../HARDWARE/Speed/speed.c		32.333 us
speed_display	1	0.278 us
floatToChar	1	0.722 us
Get_now_speed	100	30.556 us
cal_speed_pre	1	0.528 us
speed_Init	1	0.250 us

对于时间反而增加的原因，总结可能有以下 3 点：

编译器优化：编译器会根据指定的优化等级对 C 代码进行一系列的优化，包括循环展

开，常量折叠，消除无效计算等。这可能会使 C 代码的执行效率超过手工优化的汇编代码。

指令调度：现代 CPU 包含**流水线和并行执行技术**，根据代码的依赖关系，不同的指令可能会并行执行，或者在等待数据时执行其他的指令。编译器通常会进行指令调度优化，改变指令的顺序以提高 CPU 的利用率。如果汇编代码没有考虑这些因素，可能就会出现执行效率低的情况。

指令选择：现代 CPU 通常包含多种实现相同功能的指令，这些指令可能在执行时间和功耗等方面有差异。编译器会根据目标平台的特性选择最优的指令。如果手写的汇编代码没有使用最优的指令，可能会导致执行时间增加。

空间上：

```
==============================================================================

      Code (inc. data)   RO Data    RW Data    ZI Data      Debug

      9274       1684        698        356       2116     535772   Grand Totals
      9274       1684        698         88       2116     535772   ELF Image Totals (compressed)
      9274       1684        698         88          0          0   ROM Totals

==============================================================================

    Total RO  Size (Code + RO Data)                 9972 (   9.74kB)
    Total RW  Size (RW Data + ZI Data)              2472 (   2.41kB)
    Total ROM Size (Code + RO Data + RW Data)      10060 (   9.82kB)

==============================================================================

==============================================================================

      Code (inc. data)   RO Data    RW Data    ZI Data      Debug

      8452       1694        700        356       2116     534841   Grand Totals
      8452       1694        700         88       2116     534841   ELF Image Totals (compressed)
      8452       1694        700         88          0          0   ROM Totals

==============================================================================

    Total RO  Size (Code + RO Data)                 9152 (   8.94kB)
    Total RW  Size (RW Data + ZI Data)              2472 (   2.41kB)
    Total ROM Size (Code + RO Data + RW Data)       9240 (   9.02kB)

==============================================================================
```

两张图中的第一张图是 get_now_speed()用 C 语言实现的情况下的总空间消耗，而第二张图则是用汇编代替后的总空间消耗。可以看到二者 SRAM 的大小相同，都是 2472B，但 RO size(Read-Only Size)不同，这代表存储在程序内存中的常量数据和代码段的大小不同。

因此可以看出 ARMCC 编译器对这部分 C 代码进行了一些优化，通过增加命令数量来得到了更好的时间性能表现。结合之前所学的知识，极有可能的一种情况是 ARMCC 将这一部分的计算拆解成了流水线架构，从而可以并行执行多项计算任务，从而减少了运行时间，但也同时增加了一些用于存储命令的空间。

代码总行数

自行扩展及其实现方法说明

参考文献

十一、代码总行数

主控板：

文件名/函数名	行数	文件名/函数名	行数
Read.c/.h	260	Train_set.c/.h	260
Read_signal2.c/.h	70	Show_picture()	10
Speed.c/.h	120	Main.c/.h	100
Time_display.c/.h	50	__getnowspeed()	120(汇编)
Timer.c/.h	10		
Train_display.c/.h	80	总代码量	1070

小车板：

（由于小车板使用的是标准库，不是 HAL 库，因此代码量计算方式有所区别）

文件名/函数名	行数	文件名/函数名	行数
PWM.c/.h	60	Serial.c/.h	50/170
Main.c	60	总代码量	170

十二、自行扩展及其实现方法说明

1. 机械部分：

1.1 减速器与电机直接融合。

1.2 两级齿轮或其他方式直接驱动。

1.3 取消轴承，直接采用松配合方式。

1.4 选用更粗的轴和杆。

1.5 小火车的长宽比减少。

2. 电路部分：

2.1 供电板。

2.2 车站部分电路（传感器、控制等）。

2.3 小火车蜂鸣器。

2.4 信号灯译码器。

3. 软件部分

3.1 实时速度哈希演算。

3.2 到站时间实时计算。

十三、参考文献

C++编码规范，陈世忠，人民邮电出版社，2002

高质量程序设计指南：C++/C 语言，林锐等，电子工业出版社，2003
STM32F1 开发指南(精英版)-库函数版本_V1.3
陈启军，嵌入式开发及其应用